

Inżynieria Oprogramowania

Dokumentacja Projektu

pt.: „Zarządzanie klubem piłkarskim”

Data wykonania: 08.10.2024

Grupa: L9
Szyszka Joanna
Szymonik Dawid
Zając Jakub
Janik Michał
Stygar Dawid
Tułecki Kacper
Szczepański Jan
Szytuła Nikodem

1. Wstęp

Naszym projektem jest kompleksowa aplikacja przeznaczona do zarządzania klubem piłkarskim, zaprojektowana z myślą o trenerach i zawodnikach. Oferuje szereg funkcji, które ułatwiają organizację oraz monitorowanie działań związanych z drużyną, takich jak ankiety meczowe, statystyki czy zwyczajne informowanie o zbliżających się treningach i meczach.

2. Cel i założenia projektu

Głównym celem projektu jest stworzenie wszechstronnej aplikacji dedykowanej zarządzaniu klubem piłkarskim, która usprawni codzienne działania trenerów i zawodników. Aplikacja ma zapewnić intuicyjny interfejs oraz zestaw funkcji, które umożliwiają kompleksowe zarządzanie drużyną, planowanie treningów, monitorowanie stanu zdrowia zawodników oraz analizę statystyk meczowych.

Założenia projektu obejmują:

- Stworzenie prostego w obsłudze interfejsu, który będzie przyjazny dla użytkowników o różnym stopniu zaawansowania technologicznego.
- Zapewnienie kompleksowych funkcji zarządzania drużyną, tak aby trenerzy mogli efektywnie organizować treningi i planować spotkania.
- Skupienie się na bezpieczeństwie danych użytkowników poprzez odpowiednie mechanizmy uwierzytelniania i autoryzacji.

Ze względu na złożoność aplikacji w projekcie brało udział liczne grono osób. Poniżej podano imię, nazwisko oraz funkcję, jaką pełniła dana osoba:

Nikodem Szytuła: Project lider, Backend developer,

Dawid Szymonik: Backend developer, Dev-Ops,

Joanna Szyszka: Tworzenie makiet, Backend developer,

Jan Szczepański: Frontend developer, Tworzenie bazy danych,

Kacper Tułecki: Frontend developer,

Dawid Stygar: Frontend developer,
Michał Janik: Frontend developer,
Jakub Zając: Tworzenie bazy danych.

Środowisko sprzętowe

Z uwagi na dużą złożoność projektu konieczne było wykorzystanie odpowiedniego sprzętu. Podczas tworzenia aplikacji wykorzystywano sprzęt o następujących wymaganiach:

- Wymaganiem dotyczącym komputera jest zainstalowany i sprawnie działający na nim system Windows 10 lub nowszy.
- Zalecana pamięć operacyjna 1 GB RAM.
- Karta graficzna 256 MB bądź lepsza.
- Ekran rozdzielcość minimum 1920x1080

Środowisko programowania

Aby zapewnić wydajność, skalowalność i intuicyjność aplikacji do zarządzania klubem piłkarskim, wykorzystano nowoczesne technologie i narzędzia programistyczne. Poniżej przedstawiam szczegółowy opis środowiska programistycznego użytego do tworzenia aplikacji.

- JavaScript
- Visual Studio Code
- Express.js
- React
- PostgreSQL

.1. Opis działania projektu

Projekt zakłada stworzenie kompleksowej aplikacji dostępnej przez przeglądarkę internetową. Aplikacja składa się z kilku głównych modułów, które obejmują:

-Moduł zarządzania drużynami, umożliwiający tworzenie nowych drużyn, dodawanie zawodników oraz definiowanie ich ról w zespole.

-Moduł planowania treningów i harmonogramu spotkań, gdzie trenerzy mogą tworzyć i zarządzać sesjami treningowymi oraz dodawać mecze i inne wydarzenia.

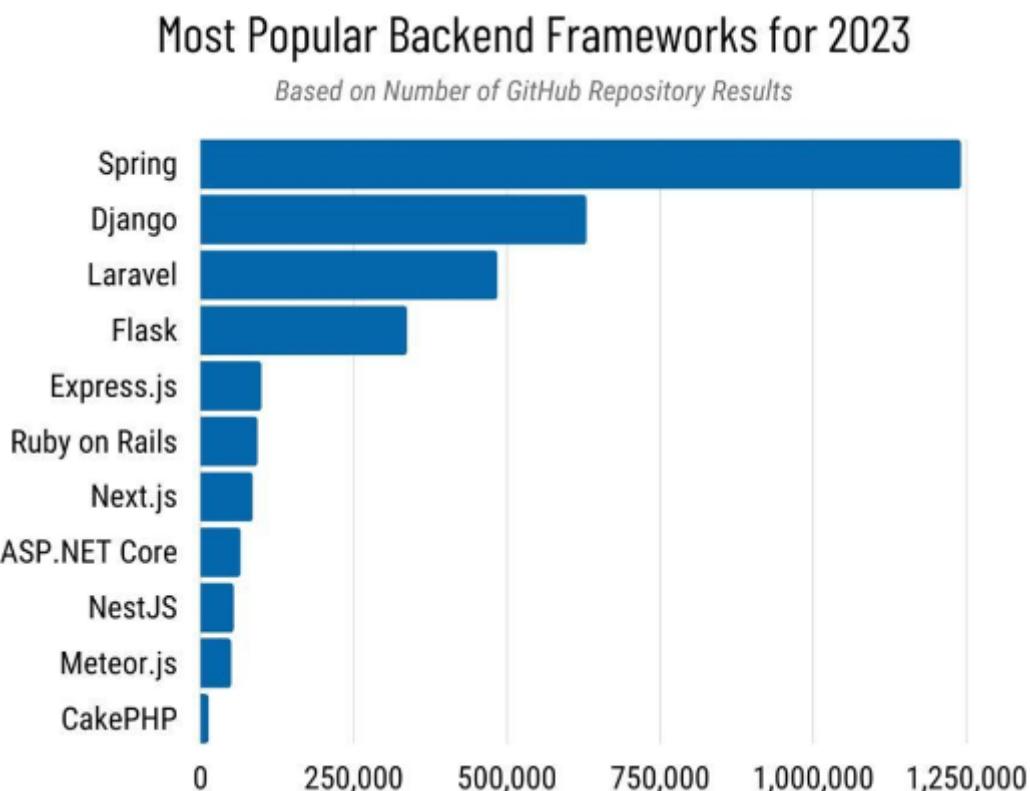
-Moduł ankiet meczowych, gdzie zawodnicy mogą zgłaszać swoje samopoczucie, ewentualne kontuzje oraz inne istotne informacje po zakończonym spotkaniu.

-Moduł statystyk, który pozwala na analizę wyników drużyny oraz indywidualnych osiągnięć zawodników, takich jak liczba strzelonych goli czy kartek.

1. Wykorzystane technologie

Do przygotowania projektu potrzebne było podjęcie decyzji odnośnie technologii, które będą wykorzystane. Biorąc pod uwagę najczęściej wykorzystywane w ubiegłym roku frameworki, zdecydowano na wybór Django, jednakże w trakcie pracy nad projektem okazało się, że powoduje on spore trudności ze zgraniem modeli frameworka z bazą danych. Z uwagi na to zdecydowano, że technologia ta zostanie zmieniona na express.js, z którą praca okazała się dużo łatwiejsza.

Do frontendu wybrany został react ze względu na jego prostotę i ilość dostępnych materiałów na jego temat. Bazę danych natomiast postanowiono stworzyć w PostgreSQL.



Rysunek 1 Wykres przedstawiający najbardziej popularne frameworki backendowe w 2023

Do stworzenia projektu wykorzystano technologie, które umożliwiają pracę w łatwy i szybki sposób. Wyboru dokonano na podstawie wiedzy autorów projektu oraz praktycznemu sprawdzeniu kilku innych technologii.

Tabela 1 Wykorzystane technologie

	Technologia
Frontend	React
Backend	Express.js
Baza danych	PostgreSQL
Hosting	Vercel

.2. Narzędzia

Do stworzenia naszej aplikacji wykorzystaliśmy wiele różnych narzędzi. Poza tymi, które były już wspomniane, korzystaliśmy z Vercela, czyli platformy do hostingu frontendowej części aplikacji. Dzięki temu mieliśmy możliwość automatycznego wdrażania aplikacji za pomocą integracji z systemami kontroli wersji. Każda zmiana w repozytorium automatycznie uruchamiała proces budowania i wdrażania, co znaczaco przyspieszyło cykl rozwoju aplikacji.

Wykorzystaliśmy również system kontroli wersji GitHub, dzięki któremu mieliśmy bezpieczne miejsce do przechowywania kodu źródłowego, umożliwiające łatwy dostęp do niego dla wszystkich członków zespołu. Udało się dzięki temu zefektyzować współpracę zespołową poprzez funkcje takie jak pull requesty, które pozwalają na przeglądanie i zatwierdzanie zmian w kodzie przed ich połączeniem do głównej gałęzi.

Do zarządzani projektem wykorzystaliśmy narzędzie Trello, które umożliwia współpracę w czasie rzeczywistym. Członkowie zespołu mogą dodawać komentarze, załączać pliki, ustawiać terminy realizacji i przypisywać zadania do konkretnych osób. Dzięki temu komunikacja i koordynacja zadań w zespole są bardziej efektywne.

.3. Biblioteki i Frameworks

W naszym projekcie kluczowe role odegrały dwa narzędzia: React i Express.js. Każde z nich przyczyniło się do różnych aspektów funkcjonalności i wydajności aplikacji.

- **React** to popularna biblioteka JavaScript stworzona przez Facebooka, służąca do budowania interfejsów użytkownika (UI). React umożliwia tworzenie dynamicznych, responsywnych aplikacji internetowych poprzez komponentową strukturę, która pozwala na ponowne wykorzystanie kodu i łatwiejsze zarządzanie dużymi aplikacjami. Wykorzystany został przy tworzeniu frontendu.
- **Express.js** to minimalistyczny i elastyczny framework dla Node.js, służący do tworzenia aplikacji serwerowych. Express.js upraszcza proces budowy serwera HTTP

oraz definiowania endpointów API, co czyni go idealnym wyborem dla backendu naszej aplikacji. Wykorzystaliśmy go przy tworzeniu backendu.

• **Implementacja**

W tym rozdziale przedstawiono szczegółowy opis implementacji systemu zarządzania klubem piłkarskim. Głównym celem implementacji było stworzenie wydajnego, łatwego w użyciu i elastycznego narzędzia do zarządzania drużynami, zawodnikami, sprzętem oraz wydarzeniami sportowymi.

System został zaprojektowany z myślą o trzech głównych użytkownikach: trenerach, administratorach i zawodnikach. Każdy z tych użytkowników ma dostęp do specyficznych funkcji, które zostały szczegółowo opisane w pozostałych rozdziałach.

.1. Struktura projektu

Struktura projektu to sieć powiązań między różnymi encjami w systemie zarządzania drużyną sportową. Na diagramie możemy zauważać centralną rolę graczy, którzy są połączeni z wieloma innymi elementami systemu. Każdy gracz może być przypisany do określonego zespołu, uczestniczyć w wydarzeniach oraz korzystać z wypożyczania

sprzętów. Ponadto, gracze są monitorowani pod względem statystyk i odpowiedzi na pytania z kwestionariuszy, co pozwala na bieżące śledzenie ich kondycji i postępów.

Poniższy diagram szczegółowo przedstawia wyżej wymienione oraz pozostałe powiązania.

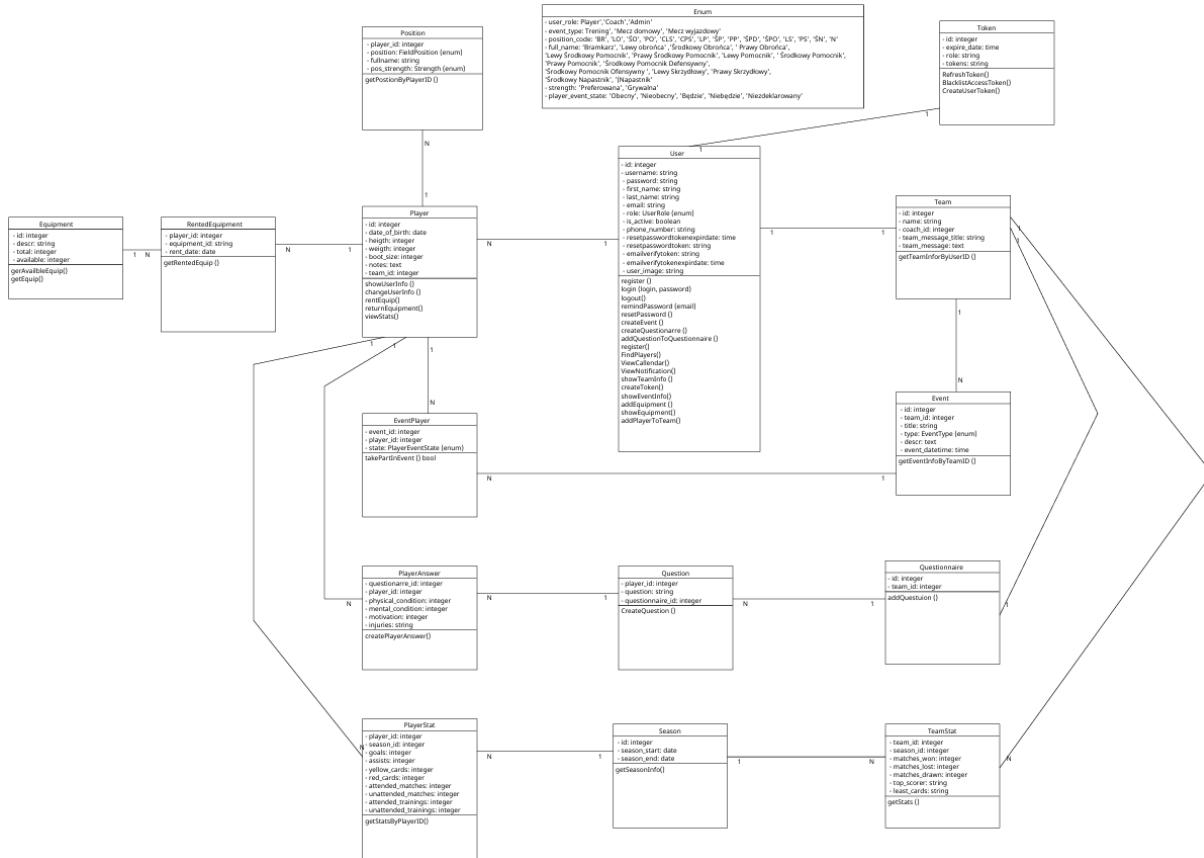


Diagram funkcji

2. Funkcjonalności

Aplikacja zawiera kilka ważnych funkcji, które pomagają w zarządzaniu klubem piłkarskim. W projekcie wyróżniono kilka głównych funkcjonalności:

- Obsługa administratora
 - Pokazanie listy drużyn
 - pokazanie nazwy drużyny

- edycja drużyny
- edycja nazwy drużyny
- edycja logo drużyny
- usunięcie drużyny
- edycja trenera
- usunięcie trenera
- kopiowanie linki do założenia konta
- Stworzenie drużyny
 - dodanie nazwy drużyny
 - dodanie logo drużyny
 - zatwierdzenie utworzenia drużyny
 - wygenerowanie linki dla trenera
- Obsługa sesji
 - logowanie administratora
 - podanie nazwy użytkownika
 - podanie hasła
 - przypomnienie hasła
 - zalogowanie użytkownika
- Obsługa zawodnika
 - Obsługa panelu głównego
 - Pokazanie informacji o zawodniku
 - Pokazanie imienia i nazwiska zawodnika
 - Pokazania pozycji zawodnika
 - Pokazania nazwy klubu piłkarskiego
 - Pokazania powiadomienia
 - Pokazanie typu powiadomienia
 - Pokazanie daty powiadomienia
 - Pokazanie treści powiadomienia
 - Pokazanie ankieta meczowej
 - Pokazanie nazwy klubu piłkarskiego

- Pokazanie typu meczu
- Pokazanie daty meczu
- Pokazanie godziny meczu
- Pokazanie ilości obecnych zawodników
- Uzupełnianie ankiety
- Pokazanie kalendarza
- Wyświetlenie dni miesiąca z wydarzeniami
- Pokazanie wydarzenia
- Pokazanie statystyk
- Wylogowanie zawodnik
- Obsługa statystyk drużyny
 - Pokazanie informacji o klubie
 - Pokazania nazwy klubu piłkarskiego
 - Pokazanie imienia i nazwiska trenera
 - Pokazanie statystyk drużyny
 - Pokazanie ilości wygranych meczów
 - Pokazanie ilości przegranych meczów
 - Pokazanie ilości remisów
 - Pokazanie imienia i nazwiska najlepszego strzelca
 - Pokazanie imienia i nazwiska zawodnika z najmniejszą liczbą kartek
 - Pokazanie wyników ankiety
 - Pokazanie nazwy klubu piłkarskiego
 - Pokazanie daty wydarzenia
 - Pokazanie godziny wydarzenia
 - Pokazanie ilości obecnych zawodników
 - Podgląd ankiety
 - Pokazanie tabeli ligowej
 - Pokazanie nazwy klubu piłkarskiego
 - Pokazanie nazwy klubu piłkarskiego
 - Pokazanie ilości punktów drużyny

- Obsługa profilu zawodnika
 - Pokazanie informacji o zawodniku
 - Pokazanie imienia i nazwiska zawodnika
 - Pokazanie pozycji zawodnika
 - Pokazanie nazwy klubu piłkarskiego
 - Pokazanie daty urodzenia zawodnika
 - Pokazanie wzrostu zawodnika
 - Pokazanie wagi zawodnika
 - Pokazanie rozmiaru buta zawodnika
 - Pokazanie adresu e-mail zawodnika
 - Pokazanie numeru telefonu zawodnika
 - Pokazanie statystyk zawodnika
 - Pokazanie ilości goli
 - Pokazanie ilości asyst
 - Pokazanie ilości żółtych kartek
- Pokazanie ilości czerwonych kartek
- Pokazanie obecności na treningach
- Pokazanie obecności na meczach
 - Pokazanie wypożyczonego sprzętu
 - Pokazanie nazwy i rozmiaru sprzętu
- Obsługa wypożyczania sprzętu
 - Pokazanie dostępnego sprzętu
 - Pokazanie nazwy i rozmiaru sprzętu
 - Wypożyczenie sprzętu
 - Pokazanie wypożyczonego sprzętu
 - Pokazanie nazwy i rozmiaru sprzętu
 - Zwrócenie wypożyczonego sprzętu
- Obsługa sesji
 - Rejestracja zawodnika
 - Wpisanie imienia

- Wpisanie nazwiska
- Wpisanie adresu e-mail
- Wpisanie nazwy użytkownika
- Wpisanie hasła
- Potwierdzenie hasła
- Wpisanie daty urodzenia
- Wpisanie wzrostu
- Wpisanie wagi
- Wpisanie rozmiaru buta
- Dodanie zdjęcia profilowego
- Logowanie zawodnika
- Podanie nazwy użytkownika
- Podanie hasła
- Przypomnienie hasła
- Zalogowanie zawodnika
- Wylogowanie zawodnika

- Obsługa trenera
 - Obsługa panelu głównego
 - Pokazanie informacji o trenerze
 - Pokazanie imienia i nazwiska zawodnika
 - Pokazania nazwy klubu piłkarskiego
 - Obsługa powiadomień
 - Dodawanie powiadomienia
 - Pokazanie powiadomienia
 - Pokazanie typu powiadomienia
 - Pokazanie daty powiadomienia
 - Pokazanie treści powiadomienia
 - Pokazanie obecności na wydarzeniu
 - Pokazanie nazwy klubu piłkarskiego

- Pokazanie typu meczu
- Pokazanie daty meczu
- Pokazanie godziny meczu
- Pokazanie ilości obecnych zawodników
- Uzupełnianie obecności
- Pokazanie imienia i nazwiska zawodnika
- Pokazanie pozycji zawodnika
- Zaznaczanie obecności zawodnika
- Pokazanie kalendarza
- Wyświetlenie dni miesiąca
- Pokazanie wydarzenia
- Pokazanie dnia i miesiąca
- Pokazanie godziny treningu
- Pokazanie godziny meczu
- Dodanie wydarzenia
- Dodanie dnia i miesiąca wydarzenia
- Dodanie godziny wydarzenia
- Dodanie nazwy wydarzenia
- Zatwierdzenie wydarzenia
- Odrzucenie wydarzenia
- Pokazanie tabeli ligowej
- Pokazanie nazwy klubu piłkarskiego
- Pokazanie ilości punktów drużyny
- Wylogowanie trenera
- Obsługa statystyk drużyn
 - Pokazanie informacji o klubie
 - Pokazanie nazwy klubu piłkarskiego
 - Pokazanie imienia i nazwiska trenera
 - Pokazanie statystyk drużyny
 - Pokazanie ilości wygranych meczy

- Pokazanie ilości przegranych meczów
- Pokazanie ilości remisów
- Pokazanie imienia i nazwiska najlepszego strzelca
- Pokazanie imienia i nazwiska zawodnika z najmniejszą ilością kartek
- Edycja statystyk drużyny
- Pokazanie wyników ankiety
- Pokazanie nazwy klubu piłkarskiego
- Pokazanie daty wydarzenia
- Pokazanie godziny wydarzenia
- Pokazanie ilości obecnych zawodników
- Pokazanie zdjęcia drużyny
- Obsługa zawodników
 - Pokazanie listy drużyn
 - Pokazanie nazwy drużyny
 - Kopiowanie linku do dodawania zawodników
 - Pokazanie danych zawodników
 - Pokazanie daty urodzenia zawodnika
 - Pokazanie wzrostu zawodnika
 - Pokazanie wagi zawodnika
 - Pokazanie rozmiaru buta zawodnika
 - Pokazanie adresu e-mail zawodnika
 - Pokazanie numeru telefonu zawodnika
 - Usunięcie danych zawodnika
 - Pokazania statystyk zawodnika
 - Pokazanie ilości goli
 - Pokazanie ilości asyst
 - Pokazanie ilości czerwonych kartek
 - Pokazanie ilości żółtych kartek
 - Pokazanie obecności na treningach
 - Pokazanie obecności na meczach

- Pokazanie wypożyczonego sprzętu
- Pokazanie nazwy i rozmiaru sprzętu
- Obsługa sprzętu piłkarskiego
 - Dodanie nowego sprzętu
 - Dodanie nazwy sprzętu
 - Dodanie rozmiaru
 - Pokazanie dostępnego sprzętu
 - Pokazanie nazwy i rozmiaru sprzętu
 - Wypożyczenie sprzętu
 - Usunięcie sprzętu
 - Pokazanie wypożyczonego sprzętu
 - Pokazanie nazwy i rozmiaru sprzętu
 - Pokazanie imienia i nazwiska właściciela sprzętu
- Obsługa sesji
 - Rejestracja trenera
 - Wpisanie imienia
 - Wpisanie nazwiska
 - Wpisanie adresu e-mail
 - Wpisanie nazwy użytkownika
 - Wpisanie hasła
 - Potwierdzenie hasła
 - Logowanie trenera
 - Podanie nazwy użytkownika
 - Podanie hasła
 - Przypomnienie hasła
 - Podanie nowego adresu e-mail
 - Zatwierdzenie nowego adresu e-mail
 - Zalogowanie użytkownika
 - Wylogowanie trenera

Rozpisane funkcjonalności:

Szyszka Joanna

- 1.3.3.1 Pokazanie listy zawodników
- 1.2.1.2 Pokazywanie powiadomienia

Szymonik Dawid

- 1.3.4.1 Dodawanie sprzętu
- 1.3.4.4 Usuwanie sprzętu

Szytuła Nikodem

- 1.2.2.1 Pokazanie informacji o klubie
- 1.3.2.3 Edycja statystyk drużyny od strony trenera

Zając Jakub

- 1.2.4.3 Pokazanie wypożyczonego sprzętu
- 1.2.4.4 Zwrócenie wypożyczonego sprzętu

Janik Michał

- 1.2.4.1 Pokazanie dostępnego sprzętu
- 1.2.4.2 Wypożyczenie sprzętu

Stygar Dawid

- 1.2.3.1 Pokazanie informacji o zawodniku
- 1.2.3.2 Pokazanie statystyk zawodnika

Tułecki Kacper

- 1.2.5.1 Rejestracja zawodnika

-1.2.2.2 Pokazanie statystyk drużyny od strony gracza

Szczepański Jan

-1.3.3.2 Pokazanie statystyk zawodników od strony trenera

-1.3.2.1 Edycja statystyk zawodników od strony trenera

Nikodem Szytula

1.2.2.1 Pokazanie informacji o klubie

```
require('dotenv').config()
//const util = require('util')
const cors = require('cors')
const express = require('express')
const app = express()

// DATABASE
const {ConnectDB} = require('../Database/ConnectDB')

// ROUTERS
const LoginRouter = require('../Routes/AuthRouter')
const AppRouter = require('../Routes/AppRouter')
const TeamStatsRouter = require('../Routes/TeamStatsRouter')
const UserRouter = require('../Routes/UserRouter')
const EquipmentRouter = require('../Routes/EquipmentRouter')
const CoachRouter = require('../Routes/CoachRouter')

// MIDDLEWARE
app.use(express.json())
app.use(cors({origin: 'http://localhost:3000'}))

// PATHS
app.use('/auth', LoginRouter)
app.use('/app', AppRouter)
app.use('/team_stats', TeamStatsRouter)
app.use('/user', UserRouter)

app.use('/equipment', EquipmentRouter)
app.use('/coach', CoachRouter)

const PORT = process.env.SERVER_PORT || 3000
ConnectDB().then(
  app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)})
)
```

```

const express = require('express')
const Router = express.Router()

// CONTROLLERS
const TeamStatsViewController = require('../Controllers/TeamStatsControllers/TeamStatsViewController')
const TeamStatisticsController = require('../Controllers/TeamStatsControllers/TeamStatisticsController')
const TeamInfoController = require('../Controllers/TeamStatsControllers/TeamInfoController')
const QuestionaryController = require('../Controllers/TeamStatsControllers/QuestionaryController')
const TeamFormViewController = require('../Controllers/TeamStatsControllers/TeamFormViewController')

// MIDDLEWARE
const authenticateToken = require('../Middleware/authenticateToken')
//const TeamFormViewController = require('../Controllers/TeamStatsControllers/TeamFormViewController')

// ROUTES
Router.get('/', authenticateToken, TeamStatsViewController)
Router.get('/statistics', authenticateToken, TeamStatisticsController)
Router.get('/info', authenticateToken, TeamInfoController)
Router.get('/form', authenticateToken, TeamFormViewController)
Router.get('/questionary', authenticateToken, QuestionaryController)

module.exports = Router

```

```

1  const {User, Players, Teams} = require('../Models/models')
2
3  const TeamInfoController = async (req, res) => {
4
5    try {
6      const team_info = await User.findAll({
7        include: [
8          {
9            model: Players,
10           attributes: ['id'],
11           include: [
12             {
13               model: Teams,
14               attributes: ['name', 'coach_id'],
15               // where: {
16               //   coach_id: Sequelize.col('user.id')
17               //}
18             }
19           ],
20           where: {
21             username: req.user.name
22           }
23         }
24       ],
25     })
26
27     const coach_id = user[0].player.team.dataValues.coach_id
28
29     try {
30       const coach_info = await User.findOne({
31         attributes: ['first_name', 'last_name'],
32         where: {
33           id: coach_id
34         }
35       })
36
37       if(coach_info.length>0){
38         return res.json({team_info:user, coach:coach_info})
39       }
40
41     } catch (error) {
42       return res.json({detail: error})
43     }
44
45   } catch (error) {
46     return res.json({detail:error})
47   }
48 }
49
50
51
52
53 module.exports = TeamInfoController

```

Modele potrzebne do tej funkcjonalności:



```
5   class User extends Model {}
6   ^ User.init(
7   ^
8   {
9       id: {
10           type: DataTypes.INTEGER,
11           allowNull: false,
12           primaryKey: true,
13           autoIncrement:true
14       },
15       username: {
16           type: DataTypes.STRING,
17           allowNull: false
18       },
19       password: {
20           type: DataTypes.STRING,
21           allowNull: false
22       },
23       first_name: {
24           type: DataTypes.STRING,
25           allowNull: false
26       },
27       last_name: {
28           type: DataTypes.STRING,
29           allowNull: false
30       },
31       email: {
32           type: DataTypes.STRING,
33           allowNull: false
34       },
35       role: {
36           type: DataTypes.ENUM('Player', 'Coach', 'Admin'),
37           allowNull: false
38       },
39       is_active: {
40           type: DataTypes.BOOLEAN,
41           allowNull: false,
42           defaultValue:false
43       },
44       phone_number: {
45           type: DataTypes.STRING,
46           allowNull: true,
47       },
48       resetpasswordtoken: {
49           type: DataTypes.STRING,
50           allowNull: true
51       },
52       resetpasswordtokenexpirdate: {
53           type: DataTypes.TIME,
54           allowNull: true
55       },
56       emailverifytoken: {
57           type: DataTypes.STRING,
58           allowNull: true
59       },
60       emailverifytokenexpirdate: {
61           type: DataTypes.TIME,
62           allowNull: true
63       },
64       },
65       {
66           sequelize,
67           modelName: 'users'
68       }
69   )
70 }
```

```
class Players extends Model {}
Players.init(
{
  id:{
    type: DataTypes.INTEGER,
    allowNull:false,
    primaryKey:true
  },
  date_of_birth:{
    type: DataTypes.DATE,
    allowNull:true
  },
  height:{
    type:DataTypes.INTEGER,
    allowNull:true
  },
  weight:{
    type:DataTypes.INTEGER,
    allowNull:true
  },
  boot_size:{
    type:DataTypes.INTEGER,
    allowNull:true
  },
  notes:{
    type:DataTypes.TEXT,
    allowNull:true
  },
  team_id:{
    type:DataTypes.INTEGER,
    allowNull:false,
  }
},
{
  sequelize,
  modelName:'players'
}

)
class Teams extends Model {}
Teams.init(
{
  id:{
    type: DataTypes.INTEGER,
    allowNull:false,
    primaryKey:true,
    autoIncrement:true
  },
  name:{
    type: DataTypes.STRING,
    allowNull:false,
  },
  coach_id:{
    type:DataTypes.INTEGER,
    allowNull:false
  },
  team_message_title:{
    type:DataTypes.STRING,
    allowNull:true
  },
  team_message:{
    type:DataTypes.TEXT,
    allowNull:true
  }
},
{
  sequelize,
  modelName:'teams'
}
)
```

Część frontendowa:

```

1  import { useState, useEffect } from 'react';
2  import './Style/TeamStatistic/club.css'
3
4
5  const Club = () => {
6    const [clubName, setClubName] = useState('')
7    const [coachName, setCoachName] = useState('')
8    const [coachSurname, setCoachSurname] = useState('')
9
10   useEffect(() => {
11     if(localStorage.getItem('team_info.name') !== null){
12       setClubName(localStorage.getItem('team_info.name'))
13       setCoachName(localStorage.getItem('team_info.coach_name'))
14       setCoachSurname(localStorage.getItem('team_info.coach_surname'))
15     }
16     else{
17       fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/team_stats/info/`, {
18         mode: 'cors',
19         method: 'GET',
20         headers: {"Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}`},
21       }).then(response=>response.json()).then(data=>{
22         if(data.detail){
23           console.log(data.detail)
24         }
25         else{
26           console.log(data.team_info[0].player.team.name)
27           setClubName(data.team_info[0].player.team.name)
28           setCoachName(data.coach.first_name)
29           setCoachSurname(data.coach.last_name)
30
31           localStorage.setItem('team_info.name', data.team_info[0].player.team.name)
32           localStorage.setItem('team_info.coach_name', data.coach.first_name)
33           localStorage.setItem('team_info.coach_surname', data.coach.last_name)
34         }
35       })
36     }
37   })
38
39   return (
40     <div id="club" className='paneShadow'>
41       <div id="clubimage">
42         |   <img alt="Logo klubu" id="clublogo"/>
43       </div>
44       <div id="clubinformation">
45         <div className="clubnazwa"><b>{clubName}</b></div>
46         <br/>
47         <div className="clubnazwa"><label>Trener - {coachName} {coachSurname}</label></div>
48       </div>
49     </div>
50   );
51 }
52
53 export default Club;

```

Konfiguracja i uruchomienie serwera:

- Połączenie z bazą danych jest nawiązywane przy starcie serwera, który nasłuchuje na określonym porcie.

Konfiguracja tras (routingu):

- Różne ścieżki są skonfigurowane przy użyciu routerów, które obsługują różne funkcjonalności, poszczególne routery odpowiadają za:
 - AuthRouter do obsługi logowania i uwierzytelniania.
 - AppRouter do obsługi głównych funkcji aplikacji.
 - TeamStatsRouter do zarządzania statystykami zespołów.
 - UserRouter do zarządzania informacjami o użytkownikach.
 - EquipmentRouter do zarządzania sprzętem.
 - CoachRouter do zarządzania informacjami o trenerze.

Do obecnej funkcjonalności trasa routingu wygląda następująco:
 Router.get('/info', authenticateToken, TeamInfoController)

Ta trasa pozwala uwierzytelnionym użytkownikom na uzyskanie informacji o drużynie. Middleware authenticateToken zapewnia, że żądanie jest wykonywane przez zalogowanego użytkownika. Po pomyślnej autoryzacji, kontroler TeamInfoController obsługuje żądanie dostarczenia informacji o drużynie.

Kontrolery:

- Kontrolery odpowiadają za obsługę żądań
- Przykładowo, TeamInfoController pobiera informacje o zespole i trenerze użytkownika na podstawie danych zapisanych w bazie.
 - Wykorzystuje on modele User, Players oraz Teams, zainportowane z pliku models, do wykonywania zapytań do bazy danych.
 - Kontroler wykonuje zapytanie do bazy danych przy użyciu Sequelize, aby znaleźć użytkownika na podstawie jego nazwy użytkownika (req.user.name). W ramach tego - zapytania pobierane są również powiązane dane graczy i zespołów.
 - Po zidentyfikowaniu zespołu, kontroler wykonuje kolejne zapytanie do bazy danych, aby uzyskać informacje o trenerze na podstawie coach_id

Middleware:

- Middleware authenticateToken służy do uwierzytelniania żądań, zapewniając, że tylko autoryzowani użytkownicy mają dostęp do określonych zasobów.

Modele danych:

- Modele danych, to jest: Players, Teams, Users, definiują strukturę tabel w bazie danych, wykorzystując Sequelize ORM.
- Każdy model zawiera pola z odpowiednimi typami danych i ograniczeniami.

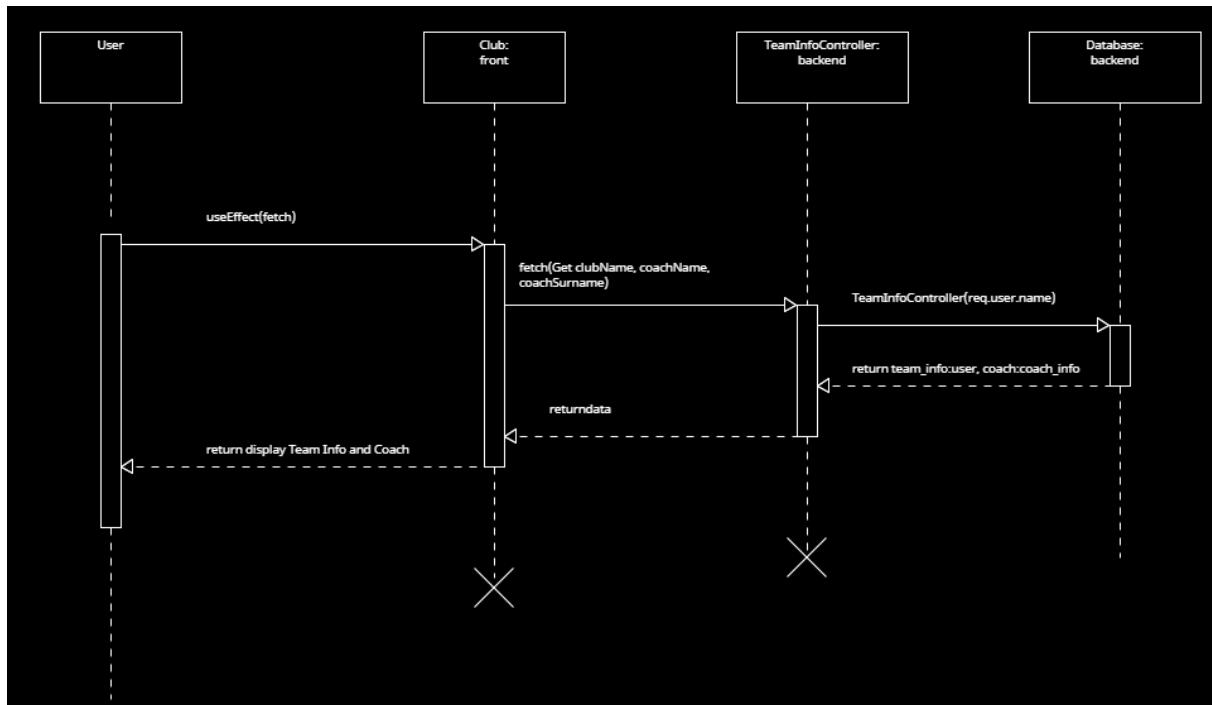
Interakcja z frontendem (React):

- Komponent React, np. Club, pobiera dane o klubie i trenerze z backenu przy pomocy fetch API.
- Dane te przechowuje w stanie komponentu przy użyciu useState.
- Wyświetla dane o klubie i trenerze w odpowiednich elementach HTML, zapewniając użytkownikowi aktualne informacje.

Jak wygląda przepływ danych:

- 1.Użytkownik otwiera stronę:** Komponent React wysyła żądanie do backenu, aby pobrać dane o zespole.
- 2.Backend przetwarza żądanie:** Router przekazuje żądanie do odpowiedniego kontrolera.
- 3.Kontroler pobiera dane:** Kontroler odczytuje dane z bazy przy pomocy modeli Sequelize i zwraca je do frontenu.
- 4.Frontend aktualizuje UI:** Otrzymane dane są zapisywane w stanie komponentu React i wyświetlane użytkownikowi.

Diagram sekwencji:



1.2.3.4 Edycja statystyk drużyny od strony trenera

Część backendu:

```
index.js > ...
1  require('dotenv').config()
2  //const util = require('util')
3  const cors = require('cors')
4  const express = require('express')
5  const app = express()
6
7
8  // DATABASE
9  const {ConnectDB} = require('./Database/ConnectDB')
10
11 // ROUTERS
12 const LoginRouter = require('./Routes/AuthRouter')
13 const AppRouter = require('./Routes/AppRouter')
14 const TeamStatsRouter = require('./Routes/TeamStatsRouter')
15 const UserRouter = require('./Routes/UserRouter')
16 const EquipmentRouter = require('./Routes/EquipmentRouter')
17 const CoachRouter = require('./Routes/CoachRouter')
18
19 // MIDDLEWARE
20 app.use(express.json())
21 app.use(cors({origin: "http://localhost:3000"}))
22
23 // PATHS
24 app.use('/auth', LoginRouter)
25 app.use('/app', AppRouter)
26 app.use('/team_stats', TeamStatsRouter)
27 app.use('/user', UserRouter)
28
29 app.use(['/equipment', EquipmentRouter])
30 app.use('/coach', CoachRouter)
31
32
33
34
35 const PORT = process.env.SERVER_PORT || 3000
36 ConnectDB().then(
37   () => app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)})
38 )
39
40
```

```

JS CoachRouter.js X JS models.js JS EquipmentAddController.js JS RentedEqController.js JS
Routes > JS CoachRouter.js > ...
1   const express = require('express')
2   const Router = express.Router()
3
4
5   // CONTROLLERS
6   const CoachLaunchViewController = require('../Controllers/CoachControllers/CoachLaunchViewController')
7   const CreateNewPlayerTokenController = require('../Controllers/CreateUserController/
CreateNewPlayerTokenController')
8   const UserListController = require('../Controllers/CoachControllers/UserListController.js')
9   const UserIdViewDataController = require('../Controllers/CoachControllers/UserViewDataController.js')
10
11  const GetPlayerStatistics = require('../Controllers/CoachControllers/UserViewStatisticController.js')
12  const UserUpdatePlayerStatistics = require('../Controllers/CoachControllers/UserUpdatePlayerStatistics.
js')
13  const UserViewEquipmentController = require('../Controllers/CoachControllers/
UserViewEquipmentController')
14  const NotificationEditController = require('../Controllers/CoachControllers/NotificationEditController.
js')
15  const EquipmentAddController = require('../Controllers/CoachControllers/EquipmentAddController.js')
16  const EquipmentDeleteController = require('../Controllers/CoachControllers/EquipmentDeleteController.
js')
17  const TeamStatisticEditController = require('../Controllers/CoachControllers/
TeamStatisticEditController.js')
18
19  const checkNewPlayerTokenController = require('../Controllers/CreateUserController/
CheckNewPlayerTokenController.js')
20  const CheckIfUserExistsController = require('../Controllers/CreateUserController/
CheckIfUserExistsController.js')
21  const CreateNewPlayerController = require('../Controllers/CreateUserController/
CreateNewPlayerController.js')
22
23  // MIDDLEWARE
24  const authenticateToken = require('../Middleware/authenticateToken')
25  const authenticateCoach = require('../Middleware/authenticateCoach')
26
27  // ROUTES
28
29
30  Router.get('/', authenticateToken, authenticateCoach, CoachLaunchViewController)
31  Router.get('/getcreateplayerToken', authenticateToken, authenticateCoach,
CreateNewPlayerTokenController)
32  Router.get('/list', authenticateToken, authenticateCoach, UserListController)
33  Router.get('/statistic', authenticateToken, authenticateCoach, GetPlayerStatistics)
34  Router.get('/data', authenticateToken, authenticateCoach, UserIdViewDataController)
35  //Router.get('/equipment/rented', authenticateToken, authenticateCoach, UserViewEquipmentController)
36  Router.post('/check', checkNewPlayerTokenController)
37  Router.get('/list', authenticateToken, authenticateCoach, UserListController)
38  Router.post('/checkUser', CheckIfUserExistsController)
39  Router.post('/create/:token', CreateNewPlayerController)
40
41  Router.post('/statistic/update', authenticateToken, authenticateCoach, UserUpdatePlayerStatistics)
42  Router.post('/notification/update', authenticateToken, authenticateCoach, NotificationEditController)
43  Router.post('/equipment/update', authenticateToken, authenticateCoach, EquipmentAddController)
44  Router.delete('/equipment/delete/:id', authenticateToken, authenticateCoach, EquipmentDeleteController)
45  Router.post('/teamstatistic/update', authenticateToken, authenticateCoach, TeamStatisticEditController)
46
47  module.exports = Router

```

```
Controllers > CoachControllers > JS NotificationEditController.js > [x] NotificationEditController
1  const { User, Teams } = require("../Models/models")
2
3
4  const NotificationEditController = async (req, res) => {
5    try {
6
7      const {team_message, team_message_title} = req.body;
8      console.log("powiadomienie", team_message_title);
9
10     const team_id = await User.findOne({
11       attributes: ['id'],
12       where:{ 
13         username: req.user.name
14       },
15       include:[{
16         model: Teams,
17         attributes:['id'],
18       }]
19     })
20     console.log('eee',team_id.team.id)
21
22     const update_team_notification = await Teams.update(
23       {
24         team_message_title,
25         team_message
26       },
27       {
28         where: {id: team_id.team.id },
29         returning: true,
30         plain: true
31       }
32     );
33
34
35     return res.json({update_team_notification})
36
37
38
39
40
41   } catch (error) {
42     console.error("Error updating notification:", error);
43     return res.json({ detail: error });
44   }
45 }
46
47
48 module.exports = NotificationEditController
```

Modele potrzebne do tej funkcjonalności:

```
)  
class Teams extends Model {}  
Teams.init(  
{  
    id:{  
        type: DataTypes.INTEGER,  
        allowNull:false,  
        primaryKey:true,  
        autoIncrement:true  
    },  
    name:{  
        type: DataTypes.STRING,  
        allowNull:false,  
    },  
    coach_id:{  
        type:DataTypes.INTEGER,  
        allowNull:false  
    },  
    team_message_title:{  
        type:DataTypes.STRING,  
        allowNull:true  
    },  
    team_message:{  
        type:DataTypes.TEXT,  
        allowNull:true  
    }  
},  
{  
    sequelize,  
    modelName: 'teams'  
}  
)
```

Część frontendowa:

```

src > Component > Main > JS NoticesPane.js > [x] NoticesPane > [x] handleSubmit
  1  ✓ import { useState, useEffect } from "react";
  2    import '../../../../../Style/Main/NoticesPane.css'
  3    const notifIcon = require '../../../../../Icons/notifiIcon.png')
  4
  5  ✓ const NoticesPane = ({toggleSharedState, sharedState}) => {
  6
  7    const [teamMessageTitle, setTeamMessageTitle] = useState('')
  8    const [teamMessage, setTeamMessage] = useState('')
  9    const [editTeamMessageTitle, setEditTeamMessageTitle] = useState('')
 10   const [editTeamMessage, setEditTeamMessage] = useState('')
 11   const [role, setRole] = useState('')
 12   const [visibility,setVisibility] = useState({visibility: "hidden"});
 13
 14
 15  ✓ const fetchUpdateNotification = async () => {
 16    fetch(`process.env.REACT_APP_SERVER_ADDRESS}/coach/notification/update`, {
 17      mode: 'cors',
 18      method: 'POST',
 19      headers: {
 20        "Content-Type": "application/json",
 21        "authorization": `Bearer ${localStorage.getItem('access_token')}`
 22      },
 23      body: JSON.stringify( {
 24        team_message_title: editTeamMessageTitle,
 25        team_message: editTeamMessage
 26      })
 27    })
 28    .then(response => response.json())
 29    .then(data => {
 30      if (data.detail) {
 31        console.log('Error:', data.detail);
 32      } else {
 33        console.log('Updated data:', data.body);
 34        toggleSharedState()
 35      }
 36    })
 37    .catch(error => {
 38      console.error('Error updating team meassage', error)
 39    })
 40  }
 41  ✓ const visibilityOn = () =>
 42  {
 43    window.scrollTo(0, 0);
 44    document.body.style.overflow = 'hidden';
 45    setVisibility({visibility: "visible"});
 46  }
 47  ✓ const visibilityOff = () =>
 48  {
 49    document.body.style.overflow = 'auto';
 50    setVisibility({visibility: "hidden"});

```

```

47     const visibilityOff = () =>
48         setVisibility({visibility: "hidden"});
49     }
50
51
52
53     const handleSubmit = () => {
54         // Implement the logic to save the updated statistics
55         fetchUpdateNotification();
56         setTeamMessageTitle(editTeamMessageTitle);
57         setTeamMessage(editTeamMessage);
58         visibilityOff();
59     }
60
61
62     const fetchNotification = async() => {
63         fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/app/team/notification/`, {
64             mode: 'cors',
65             method: 'GET',
66             headers: {"Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}`},
67         }).then(response=>response.json()).then(data=>{
68             console.log(data.user)
69             setTeamMessageTitle(data.notification.player.team.team_message_title)
70             setTeamMessage(data.notification.player.team.team_message)
71             // setRole(data.user_cred.role)
72
73             // localStorage.setItem('user.role', data.user_cred.role)
74             localStorage.setItem('notification.team_message_title', data.notification.player.team.team_message_title)
75             localStorage.setItem('notification.team_message', data.notification.player.team.team_message)
76         })
77     }
78     useEffect(() => {
79         // if(localStorage.getItem('notification.team_message_title') !== null){
80         //     setTeamMessageTitle(localStorage.getItem('notification.team_message_title'))
81         //     setTeamMessage(localStorage.getItem('notification.team_message'))
82         //     setRole(localStorage.getItem('role'))
83         // } else{
84             setRole(localStorage.getItem('role'))
85             fetchNotification();
86         }
87     })
88
89     return (
90         <div id="noticesPane" className='paneShadow'>
91             <div id="notification">
92                 <div id="nLeft">
93                     |   <img src={notifIcon} alt="icon" />
94                 </div>
95                 <div id="nMiddle">
96                     <div id="nTitle">
97                         {teamMessageTitle}

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

90     <div id="noticesPane" className='paneShadow'>
91         <div id="notification">
92             <div id="nLeft">
93                 <img src={notifIcon} alt="icon" />
94             </div>
95             <div id="nMiddle">
96                 <div id="nTitle">
97                     {teamMessageTitle}
98                 </div>
99                 <div id="nMessage">
100                     {teamMessage}
101                 </div>
102             </div>
103             <div id="nRight">
104                 {role === 'Coach'? <div className="smallRedBtn" onClick={visibilityOn}>Edytuj</div>:
105                 <div></div>)
106             </div>
107         </div>
108
109         <div id="notifBox" style=(visibility) >
110             <div id="notifWindow">
111                 <div id="notifEditTitle">
112                     Edytuj Powiadomienie
113                 </div>
114                 <div id="notifInputs">
115                     <p className="nEditP">Tytuł:</p>
116                     <input type="text" id="nTitleInput" min="0" value={editTeamMessageTitle} onChange={(e)=>setEditTeamMessageTitle(e.target.value)}/>
117                     <p className="nEditP">Treść:</p>
118                     <textarea id="nMessageTextArea" value={editTeamMessage} onChange={(e)=>setEditTeamMessage(e.target.value)}/>
119                 </div>
120                 <div id="notifButtons">
121                     <div className="smallRedBtn" onClick={visibilityOff}>Wróć</div>
122                     <div className="smallRedBtn" onClick={handleSubmit}>Zapisz</div>
123                 </div>
124             </div>
125         </div>
126     </div>
127   </div>
128 );
129 }
130 export default NoticesPane

```

Konfiguracja i uruchomienie serwera:

- Połączenie z bazą danych jest nawiązywane przy starcie serwera, który nasłuchiwa na określonym porcie.

Konfiguracja tras (routingu):

```
Router.post('/notification/update', authenticateToken, authenticateCoach, NotificationEditController)
```

Ta trasa pozwala uwierzytelnionym trenerom na aktualizację powiadomień dla swoich zawodników. Middleware authenticateToken zapewnia, że żądanie jest wykonywane przez zalogowanego użytkownika, natomiast authenticateCoach weryfikuje, że użytkownik posiada uprawnienia trenera. Po pomyślnej autoryzacji, kontroler NotificationEditController obsługuje żądanie aktualizacji powiadomień.

Kontrolery:

Kontroler NotificationEditController odpowiada za aktualizację powiadomień zespołu na podstawie danych użytkownika. Wykorzystuje modele User i Teams, zimportowane z pliku models, do wykonywania zapytań do bazy danych przy użyciu Sequelize.

Kontroler najpierw wykonuje zapytanie do bazy danych, aby znaleźć użytkownika na podstawie jego nazwy użytkownika (req.user.name). W ramach tego zapytania pobierane są również powiązane dane zespołu. Po zidentyfikowaniu zespołu, kontroler aktualizuje powiadomienia zespołu na podstawie danych przesłanych w żądaniu (team_message, team_message_title).

Middleware:

- Middleware authenticateToken służy do uwierzytelniania żądań, zapewniając, że tylko autoryzowani użytkownicy mają dostęp do określonych zasobów.
- Middleware authenticateCoach dodatkowo weryfikuje, czy użytkownik posiada uprawnienia trenera, co zapewnia, że tylko trenerzy mogą wykonywać określone operacje.

W przypadku kontrolera NotificationEditController, oba middleware są używane, aby upewnić się, że tylko autoryzowani trenerzy mogą aktualizować powiadomienia zespołu.

Modele danych:

- Modele danych, to jest: Players, Teams, Users, definiują strukturę tabel w bazie danych, wykorzystując Sequelize ORM.
- Każdy model zawiera pola z odpowiednimi typami danych i ograniczeniami.

W poniższym przykładzie przedstawiono, jak komponent NoticesPane na frontendzie React współpracuje z kontrolerem NotificationEditController na backendzie.

Opis działania:

1. Inicjalizacja stanu:
 - Komponent inicjalizuje stany teamMessageTitle, teamMessage, editTeamMessageTitle, editTeamMessage, role oraz visibility, które odpowiadają za przechowywanie tytułu i treści wiadomości zespołu oraz stan widoczności okna edycji.
2. Pobieranie aktualnych powiadomień:
 - Funkcja fetchNotification wysyła żądanie GET do serwera, aby pobrać aktualne powiadomienia zespołu. Otrzymane dane są następnie zapisywane w stanach teamMessageTitle i teamMessage.
3. Aktualizacja powiadomień:

- Funkcja fetchUpdateNotification wysyła żądanie POST do kontrolera NotificationEditController, aby zaktualizować tytuł i treść powiadomienia zespołu.
- Żądanie zawiera nagłówek autoryzacyjny z tokenem dostępu oraz dane do aktualizacji (tytuł i treść wiadomości) w formacie JSON.
- Po pomyślnej aktualizacji, komponent wywołuje funkcję toggleSharedState, aby zsynchronizować stan aplikacji.

4. Interfejs użytkownika:

- Komponent renderuje powiadomienie zespołu oraz przycisk edycji, który jest widoczny tylko dla użytkowników o roli "Coach".
- Po kliknięciu przycisku edycji, otwiera się okno edycji, gdzie użytkownik może zmienić tytuł i treść powiadomienia.
- Przyciski "Wróć" i "Zapisz" pozwalają na zamknięcie okna edycji bez zapisywania zmian lub zapisanie nowych danych odpowiednio.

Jak wygląda przepływ danych:

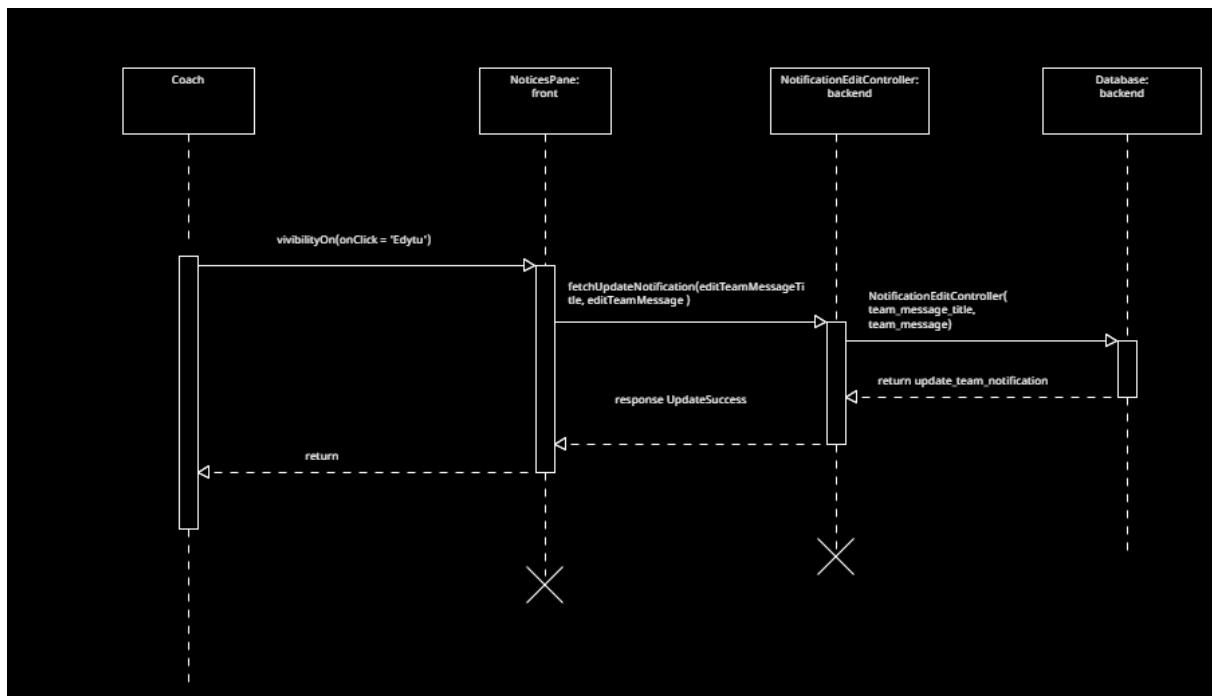
1.Użytkownik otwiera stronę: Komponent React wysyła żądanie do backendu, aby pobrać dane o zespole.

2.Backend przetwarza żądanie: Router przekazuje żądanie do odpowiedniego kontrolera.

3.Kontroler pobiera dane: Kontroler odczytuje dane z bazy przy pomocy modeli Sequelize i zwraca je do frontendu.

4.Frontend aktualizuje UI: Otrzymane dane są zapisywane w stanie komponentu React i wyświetlane użytkownikowi.

Diagram sekwencji:



Joanna Szyszka

-1.3.3.1 Pokazanie listy zawodników

```
1  require('dotenv').config()
2  //const util = require('util')
3  const cors = require('cors')
4  const express = require('express')
5  const app = express()
6
7
8  // DATABASE
9  const {ConnectDB} = require('../Database/ConnectDB')
10
11 // ROUTERS
12 const LoginRouter = require('../Routes/AuthRouter')
13 const AppRouter = require('../Routes/AppRouter')
14 const TeamStatsRouter = require('../Routes/TeamStatsRouter')
15 const UserRouter = require('../Routes/UserRouter')
16 const EquipmentRouter = require('../Routes/EquipmentRouter')
17 const CoachRouter = require('../Routes/CoachRouter')
18
19
20 // MIDDLEWARE
21 app.use(express.json())
22 app.use(cors({origin: "http://localhost:3000"}))
23
24 // PATHS
25 app.use('/auth', LoginRouter)
26 app.use('/app', AppRouter)
27 app.use('/team_stats', TeamStatsRouter)
28 app.use('/user', UserRouter)
29
30 app.use('/equipment', EquipmentRouter)
31 app.use('/coach', CoachRouter)
32
33
34
35 const PORT = process.env.SERVER_PORT || 3000
36 ConnectDB().then(
37   | app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)})
38 )
39
```

```

1  const {User,Players,Teams, Player_positions, Positions} = require '../../../../../Models/models'
2
3  const UserListController = async (req, res) => {
4    try {
5      const coach = await User.findOne({ where: { username: req.user.name, role: 'Coach' } });
6      //console.log('Zalogowany trener:', coach);
7
8      if (!coach) {
9        //console.log('Coach not found');
10       return res.status(404).json({ detail: "Coach not found" });
11     }
12
13
14
15      const teams = await Teams.findAll({ where: { coach_id: coach.id } });
16      //console.log('Drużyny prowadzone przez trenera:', teams);
17
18      if (teams.length === 0) {
19        console.log('No teams found for this coach');
20        return res.status(404).json({ detail: "No teams found for this coach" });
21      }
22
23
24
25      const list = await User.findAll({
26        attributes:['first_name', 'last_name'],
27        include: [
28          {
29            model: Players,
30            attributes: ['id'],
31            include:[
32              {
33                model: Player_positions,
34                attributes: ['position_id'],
35                include:[
36                  {
37                    model:Positions,
38                    attributes: ['position_code']
39                  }
40                ],
41                where: { team_id: teams[0].id },
42              },
43              where: { role: 'Player' }
44            );
45            //console.log('Zawodnicy w drużynach:', list);
46
47            if (list.length === 0) {
48              console.log('No players found for the teams of this coach');
49              return res.status(404).json({ detail: "No players found for the teams of this coach" });
50            }
51
52            if(list.length != 0){
53              return res.json({
54                list:list
55              })
56            }
57            else{
58              throw new Error("no events")
59            }
60
61          } catch (error) {
62            return res.json({detail: error})
63          }
64
65        module.exports = UserListController

```

Poniżej widzimy modele wykorzystywane w tej funkcjonalności, definiując one strukturę danych dotyczącą pozycji zawodnika, istniejących pozycji oraz samego zawodnika.

```
275     class Player_positions extends Model{}  
276     Player_positions.init(  
277     {  
278         player_id: {  
279             type: DataTypes.INTEGER,  
280             allowNull: false,  
281             primaryKey: true,  
282             // references: {  
283                 //     model: Players,  
284                 //     key: 'id'  
285                 // }  
286         },  
287         position_id: {  
288             type: DataTypes.INTEGER,  
289             allowNull: false,  
290             primaryKey: true,  
291             // references: {  
292                 //     model: Positions,  
293                 //     key: 'position_id'  
294                 // }  
295         },  
296         pos_strength: {  
297             type: DataTypes.ENUM('Preferowana', 'Grywalna'),  
298             allowNull: false  
299         },  
300     },  
301     {  
302         sequelize,  
303         modelName: 'player_positions',  
304     }  
305 )  
  
250 class Positions extends Model {}  
251 Positions.init(  
252 {  
253     position_id:{  
254         type: DataTypes.INTEGER,  
255         allowNull:false,  
256         primaryKey:true  
257     },  
258     position_code:{  
259         type: DataTypes.ENUM('BR', 'LO', 'ŚO', 'PO', 'CLS', 'CPS', 'LP', 'ŚP', 'PP', 'ŚPD', 'ŚPO', 'LS', 'PS', 'ŚN', 'N'),  
260         allowNull:true,  
261         primaryKey:true  
262     },  
263     full_name:{  
264         type: DataTypes.STRING(50),  
265         allowNull:false ,  
266     },  
267 },  
268 {  
269     sequelize,  
270     modelName:'positions'  
271 }  
272 )
```

```
208 class Players extends Model {}  
209 Players.init(  
210 {  
211   id:{  
212     type: DataTypes.INTEGER,  
213     allowNull:false,  
214     primaryKey:true  
215   },  
216   date_of_birth:{  
217     type: DataTypes.DATE,  
218     allowNull:true  
219   },  
220   height:{  
221     type:DataTypes.INTEGER,  
222     allowNull:true  
223   },  
224   weight:{  
225     type:DataTypes.INTEGER,  
226     allowNull:true  
227   },  
228   boot_size:{  
229     type:DataTypes.INTEGER,  
230     allowNull:true  
231   },  
232   notes:{  
233     type:DataTypes.TEXT,  
234     allowNull:true  
235   },  
236   team_id:{  
237     type:DataTypes.INTEGER,  
238     allowNull:false,  
239   }  
240 },  
241 {  
242   sequelize,  
243   modelName:'players'  
244 }  
245  
246 })  
247
```

Część frontendowa:

```

1 // Components / CoachView / # PlayerList_CV.js / # MyPlayerList_CV.js / # returnMyPlayerList_CV.js / # handleCopyClick
2
3 import { useState, useEffect } from 'react';
4
5 import "../../Style/CoachView/PlayerList_CV.css";
6
7 const PlayerList_CV = ({ onSelectPlayer, sharedState }) => {
8   const [fetchedPlayers, setFetchedPlayers] = useState([]);
9   const [alertMessage, setAlertMessage] = useState('');
10  const [showAlert, setShowAlert] = useState(false);
11
12  const [players, setPlayers] = useState([]);
13  const [createPlayerToken, setCreatePlayerToken] = useState();
14
15  const fetchPlayerList_CV = () => {
16    fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/list/`, {
17      mode: 'cors',
18      method: 'GET',
19      headers: { 'Content-Type': 'application/json', 'authorization': `Bearer ${localStorage.getItem('access_token')}` },
20    }).then(response => response.json()).then(data => {
21      if (data.detail) {
22        console.log(data.detail)
23      } else {
24
25        //console.log(data.eq)
26        // for(let i=0; i<data.eq.length; i++){
27        //   console.log(data.eq[i])
28        //   setEquipment([]);
29        //}
30        setFetchedPlayers(data.list)
31        console.log(fetchedPlayers)
32      }
33    })
34  }
35  useEffect(() => {
36    fetchPlayerList_CV()
37  }, [sharedState]);
38
39  const handleCopyClick = async () => {
40    try {
41      const response = await fetch('YOUR_BACKEND_ENDPOINT');
42      if (!response.ok) {
43        throw new Error('Network response was not ok');
44      }
45      const data = await response.json();
46      // Assume the link is returned in the 'link' field of the response data
47      const link = data.link;
48
49      // Handle copying the link to the stash here (e.g., using the clipboard API)
50      navigator.clipboard.writeText(link).then(() => {
51        window.alert('Link copied successfully: ' + link);
52
53        // Handle copying the link to the stash here (e.g., using the Clipboard API)
54        navigator.clipboard.writeText(link).then(() => {
55          window.alert('Link copied successfully: ' + link);
56        });
57      })
58    } catch (error) {
59      console.error('Error fetching link:', error);
60      setAlertMessage('Failed to copy link.');
61      setShowAlert(true);
62      setTimeout(() => setShowAlert(false), 3000); // Hide alert after 3 seconds
63    }
64    window.alert("asdadasd");
65  };
66
67  return (
68    <div id="playerList_CV">
69      <div className="addPlayerCard_CV">
70        <i className="icon-plus-alternative"/>
71        <span className="addName_CV bStyle">DODAJ ZAWODNIKA</span>
72        <button className="copy_CV" onClick={handleCopyClick}>Kopiuj</button>
73      </div>
74      <div id="listOfPlayers_CV">
75        {fetchedPlayers.map(player => (
76          <div key={player.player.id} onClick={() => onSelectPlayer(player.player.id)} className="playerCard_CV">
77            
78            <span className="playerName_CV bStyle">(${player.first_name} ${player.last_name}) ${player.player_positions.map(position => position.position.position_code)}</span>
79          </div>
80        ))
81      </div>
82    </div>
83  );
84}
85
86 export default PlayerList_CV;

```

Ta funkcjonalność realizowana jest za pomocą kontrolera UserListController. Kontroler ten jest odpowiedzialny za obsługę żądania, które ma na celu pobranie listy zawodników zarządzanych przez zalogowanego trenera.

Na początku widzimy importowanie potrzebnych modułów i modeli, takich jak: User, Players, Teams, Player_positions, Positions.

-Dalej, kontroler próbuje znaleźć zalogowanego trenera na podstawie jego nazwy użytkownika i roli.

- Jeśli trener został znaleziony, kontroler pobiera drużynę prowadzoną przez tego trenera.
- Kontroler następnie pobiera listę zawodników należących do drużyn prowadzonych przez trenera. W tym celu używa modelu **User** i łączy dane z modelami **Players**, **Player_positions**, i **Positions**, aby uzyskać szczegółowe informacje o zawodnikach i ich pozycjach.

Przykład użycia:

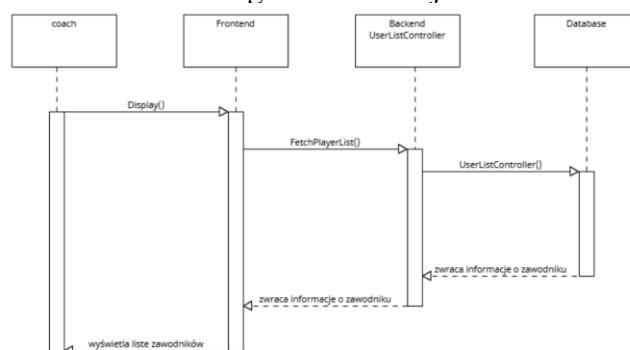
1. Warunki

- Trener jest zalogowany do systemu.
- Trener ma przypisane drużyny i zawodników w systemie.

2. Działanie

- Trener loguje się do systemu za pomocą swojego konta.
- System uwierzytelnia trenera i generuje token uwierzytelniający.
- Trener wysyła żądanie GET na endpoint /list, aby zobaczyć listę zawodników.
- System odbiera żądanie i używa middleware authenticateToken oraz authenticateCoach
- Kontroler UserListController przetwarza żądanie:
- Znajduje zalogowanego trenera w bazie danych.
- Pobiera drużyny prowadzone przez tego trenera.
- Pobiera listę zawodników przypisanych do tych drużyn wraz z ich pozycjami.

Diagram sekwencji



-1.2.1.2 Pokazywanie powiadomienia

Przykładowy model, wykorzystany w tej funkcjonalności. Pozostałe zostały wstawione w opisie innych funkcjonalności. Definiuje on strukturę danych drużyny.

```
1  )!>
2  class Teams extends Model {}
3  Teams.init(
4  {
5      id:{
6          type: DataTypes.INTEGER,
7          allowNull:false,
8          primaryKey:true,
9          autoIncrement:true
10     },
11     name:{
12         type: DataTypes.STRING,
13         allowNull:false,
14     },
15     coach_id:{
16         type:DataTypes.INTEGER,
17         allowNull:false
18     },
19     team_message_title:{
20         type:DataTypes.STRING,
21         allowNull:true
22     },
23     team_message:{
24         type:DataTypes.TEXT,
25         allowNull:true
26     }
27 },
28 {
29     sequelize,
30     modelName:'teams'
31 }
32 )
```

Część frontendowa:

```

src > Component > Main > js NoticesPane.js > (1) NoticesPane > (2) handleSubmit
1  ✓ import { useState, useEffect } from "react";
2  import '../../../../../Style/Main/NoticesPane.css'
3  const notifIcon = require '../../../../../Icons/notifiIcon.png'
4
5  ✓ const NoticesPane = ({toggleSharedState, sharedState}) => {
6
7      const [teamMessageTitle, setTeamMessageTitle] = useState('')
8      const [teamMessage, setTeamMessage] = useState('')
9      const [editTeamMessageTitle, setEditTeamMessageTitle] = useState('')
10     const [editTeamMessage, setEditTeamMessage] = useState('')
11     const [role, setRole] = useState('')
12     const [visibility, setVisibility] = useState({visibility: "hidden"});
13
14
15    ✓ const fetchUpdateNotification = async () => {
16        fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/notification/update`, {
17            mode: 'cors',
18            method: 'POST',
19            headers: {
20                "Content-Type": "application/json",
21                "authorization": `Bearer ${localStorage.getItem('access_token')}`
22            },
23            body: JSON.stringify( {
24                team_message_title: editTeamMessageTitle,
25                team_message: editTeamMessage
26            })
27        })
28        .then(response => response.json())
29        .then(data => {
30            if (data.detail) {
31                console.log('Error:', data.detail);
32            } else {
33                console.log("Updated data:", data.body);
34                toggleSharedState();
35            }
36        })
37        .catch(error => {
38            console.error('Error updating team meassage', error)
39        })
40    }
41    ✓ const visibilityOn = () =>
42    {
43        window.scrollTo(0, 0);
44        document.body.style.overflow = 'hidden';
45        setVisibility({visibility: "visible"});
46    }
47    ✓ const visibilityOff = () =>
48    {
49        document.body.style.overflow = 'auto';
50        setVisibility({visibility: "hidden"});
51    }

```

```

47     const visibilityOff = () =>
48       setVisibility({visibility: "hidden"});
49   }
50
51
52   const handleSubmit = () => {
53     // Implement the logic to save the updated statistics
54     fetchUpdateNotification();
55     setTeamMessageTitle(editTeamMessageTitle);
56     setTeamMessage(editTeamMessage);
57     visibilityOff();
58   }
59
60
61   const fetchNotification = async() => {
62     fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/app/team/notification/`, {
63       mode: 'cors',
64       method: 'GET',
65       headers: {'Content-Type': "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}`},
66     }).then(response=>response.json()).then(data=>{
67       console.log(data.user)
68       setTeamMessageTitle(data.notification.player.team.team_message_title)
69       setTeamMessage(data.notification.player.team.team_message)
70       // setRole(data.user_cred.role)
71
72       // localStorage.setItem('user.role', data.user_cred.role)
73       localStorage.setItem('notification.team_message_title', data.notification.player.team.team_message_title)
74       localStorage.setItem('notification.team_message', data.notification.player.team.team_message)
75     })
76   }
77   useEffect(() => {
78     // if(localStorage.getItem('notification.team_message_title') !== null){
79     //   setTeamMessageTitle(localStorage.getItem('notification.team_message_title'))
80     //   setTeamMessage(localStorage.getItem('notification.team_message'))
81     //   setRole(localStorage.getItem('role'))
82     //}
83     // else{
84     //   setRole(localStorage.getItem('role'))
85     //   fetchNotification();
86   })
87
88
89   return (
90     <div id="noticesPane" className="paneShadow">
91       <div id="notification">
92         <div id="nLeft">
93           <img src={notifIcon} alt="icon" />
94         </div>
95         <div id="nMiddle">
96           <div id="nTitle">
97             {teamMessageTitle}
98           </div>
99           <div id="nMessage">
100             {teamMessage}
101           </div>
102         </div>
103         <div id="nRight">
104           {role === 'Coach' ? <div className="smallRedBtn" onClick={visibilityOn}>Edytuj</div>:
105             <div></div>}
106         </div>
107       </div>
108
109       <div id="notifBox" style={visibility} >
110         <div id="notifWindow">
111           <div id="notifeditTitle">
112             Edytuj Powiadomienie
113           </div>
114           <div id="notifInputs">
115             <p className="nedit">Tytuł:</p>
116             <input type="text" id="nTitleInput" min="0" value={editTeamMessageTitle} onChange={(e)=>setEditTeamMessageTitle(e.target.value)}/>
117             <p className="nedit">Treść:</p>
118             <textarea id="nMessageTextArea" value={editTeamMessage} onChange={(e)=>setEditTeamMessage(e.target.value)}/>
119           </div>
120           <div id="notifButtons">
121             <div class="smallRedBtn" onClick={visibilityOff}>Wróć</div>
122             <div class="smallRedBtn" onClick={handleSubmit}>Zapisz</div>
123           </div>
124         </div>
125       </div>
126     </div>
127   );
128 }
129
130 export default NoticesPane

```

Część backendowa:

```
js index.js > ...
1  require('dotenv').config()
2  //const util = require('util')
3  const cors = require('cors')
4  const express = require('express')
5  const app = express()
6
7
8  // DATABASE
9  const {ConnectDB} = require('./Database/ConnectDB')
10
11 // ROUTERS
12 const LoginRouter = require('./Routes/AuthRouter')
13 const AppRouter = require('./Routes/AppRouter')
14 const TeamStatsRouter = require('./Routes/TeamStatsRouter')
15 const UserRouter = require('./Routes/UserRouter')
16 const EquipmentRouter = require('./Routes/EquipmentRouter')
17 const CoachRouter = require('./Routes/CoachRouter')
18
19 // MIDDLEWARE
20 app.use(express.json())
21 app.use(cors({origin: "http://localhost:3000"}))
22
23 // PATHS
24 app.use('/auth', LoginRouter)
25 app.use('/app', AppRouter)
26 app.use('/team_stats', TeamStatsRouter)
27 app.use('/user', UserRouter)
28
29 app.use('/equipment', EquipmentRouter)
30 app.use('/coach', CoachRouter)
31
32
33
34
35 const PORT = process.env.SERVER_PORT || 3000
36 ConnectDB().then(
37   () => app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)})
38 )
39
40
```

```
1  Routes > JS AppRouter.js > ...
2  1  const express = require('express')
3  2  const Router = express.Router()
4  3
5  4
6  5 // CONTROLLERS
7  6  const MainViewController = require('../Controllers/AppControllers/
8  7   MainViewController')
9  8  const UserProfileController = require('../Controllers/AppControllers/
10  9   UserProfileController')
11 10  const NotificationController = require('../Controllers/AppControllers/
12 11   NotificationController')
13 12  const TeamStatisticsController = require('../Controllers/TeamStatsControllers/
14 13   TeamStatisticsController')
15 14  const CalendarGetEventController = require('../Controllers/AppControllers/
16 15   CalendarGetEventController')
17 16 // const NotificationEditController = require('../Controllers/AppControllers/
18 17   NotificationEditController.js')
19 18
20 19 // MIDDLEWARE
21 20  const authenticateToken = require('../Middleware/authenticateToken')
22 21
23 22 // ROUTES
24 23  Router.get('/', authenticateToken, MainViewController)
25 24  Router.get('/user/profile', authenticateToken, UserProfileController)
26 25  Router.get('/team/notification', authenticateToken, NotificationController)
27 26 // Router.post('/team/notification/update', authenticateToken,
28 27   NotificationEditController)
29 28
30 29
31 30
32 31
33 32 module.exports = Router
```

```

Controllers > AppControllers > JS NotificationController.js > [o] NotificationController
  1  const {User, Players, Teams} = require('../Models/models')
  2
  3  const NotificationController = async (req, res) => {
  4      try {
  5          const notifi_data = await User.findOne({
  6              attributes: ['id'],
  7              include: [
  8                  {
  9                      model: Players,
 10                      attributes:[ 'id' ],
 11                      include: [
 12                          {
 13                              model: Teams,
 14                              attributes: [ 'team_message_title', 'team_message' ]
 15                          }
 16                      ]
 17                  },
 18                  where: {
 19                      username: req.user.name
 20                  }
 21              })
 22
 23          if(notifi_data.length != 0){
 24              return res.json({notification:notifi_data})
 25          }
 26
 27      } catch (error) {
 28          return res.json({'detail':error})
 29      }
 30  }
 31
 32
 33
 34  module.exports = NotificationController

```

Funkcjonalność "pokazywanie powiadomień" umożliwia trenerowi i zawodnikom przeglądanie powiadomień dotyczących drużyny, takich jak wiadomości od trenera. Dane te są pobierane z serwera i wyświetlane w interfejsie użytkownika.

- Na początku pliku importowane są niezbędne moduły takie jak: User, Players, Teams.
- Następnie definiowany jest kontroler NotificationController.
- Pobierane są również powiadomienia z serwera przez komponent React.
- Wyświetlenie powiadomień w interfejsie użytkownika.

Zadania frontendu:

- Komponent React jest odpowiedzialny za pobieranie i wyświetlanie powiadomień oraz umożliwiający edycję powiadomień przez trenera.

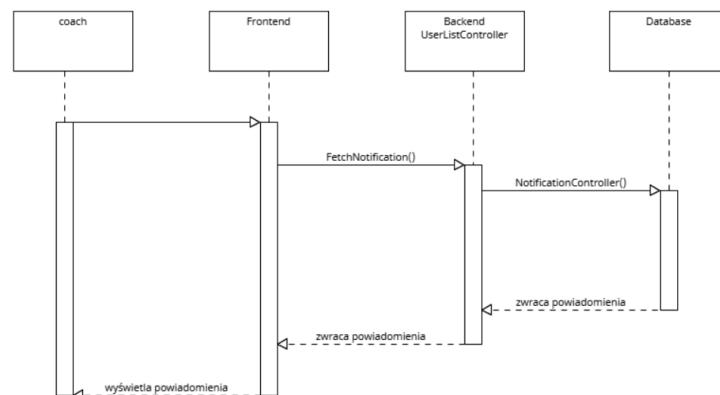
Zadania backendu:

- Kontroler jest odpowiedzialny za pobieranie danych powiadomień z bazy danych oraz zwracanie ich do klienta.

Przykład użycia:

- Trener loguje się do systemu za pomocą swojego konta.
 -System uwierzytelnia trenera i generuje token uwierzytelniający.
 -Trener przechodzi do sekcji powiadomień w aplikacji.
 -Komponent React **NoticesPane** inicjalizuje się i wykonuje żądanie GET do serwera, aby pobrać aktualne powiadomienia.
 -Serwer odbiera żądanie, uwierzytelnia token trenera, pobiera powiadomienia z bazy danych i zwraca je do klienta.
 -Komponent React otrzymuje dane powiadomień i wyświetla je w interfejsie użytkownika.

Diagram sekwenacji



Dawid Szymonik

-1.3.4.1 Dodawanie sprzętu

```
1  require('dotenv').config()
2  //const util = require('util')
3  const cors = require('cors')
4  const express = require('express')
5  const app = express()
6
7
8  // DATABASE
9  const {ConnectDB} = require('../Database/ConnectDB')
10
11 // ROUTERS
12 const LoginRouter = require('../Routes/AuthRouter')
13 const AppRouter = require('../Routes/AppRouter')
14 const TeamStatsRouter = require('../Routes/TeamStatsRouter')
15 const UserRouter = require('../Routes/UserRouter')
16 const EquipmentRouter = require('../Routes/EquipmentRouter')
17 const CoachRouter = require('../Routes/CoachRouter')
18
19 // MIDDLEWARE
20 app.use(express.json())
21 app.use(cors({origin: "http://localhost:3000"}))
22
23 // PATHS
24 app.use('/auth', LoginRouter)
25 app.use('/app', AppRouter)
26 app.use('/team_stats', TeamStatsRouter)
27 app.use('/user', UserRouter)
28
29 app.use('/equipment', EquipmentRouter)
30 app.use('/coach', CoachRouter)
31
32
33
34
35 const PORT = process.env.SERVER_PORT || 3000
36 ConnectDB().then(
37   () => app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)))
38 )
39
40
41
```

Definicja tras:



```
Routes.js: CoachRoutes.js (47 lines)
1 const express = require('express')
2 const Router = express.Router()
3
4
5 // CONTROLLERS
6 const CoachLaunchViewController = require('../Controllers/CoachControllers/CoachLaunchViewController')
7 const CreateNewPlayerTokenController = require('../Controllers/CreateUserController/CreateNewPlayerTokenController')
8 const UserListController = require('../Controllers/CoachControllers/UserListController.js')
9 const UserIdViewDataController = require('../Controllers/CoachControllers/UserViewDataController.js')
10
11 const GetPlayerStatistics = require('../Controllers/CoachControllers/UserViewStatisticController.js')
12 const UserUpdatePlayerStatistics = require('../Controllers/CoachControllers/UserUpdatePlayerStatistics.js')
13 const UserViewEquipmentController = require('../Controllers/CoachControllers/UserViewEquipmentController')
14 const NotificationEditController = require('../Controllers/CoachControllers/NotificationEditController.js')
15 const EquipmentAddController = require('../Controllers/CoachControllers/EquipmentAddController.js')
16 const EquipmentDeleteController = require('../Controllers/CoachControllers/EquipmentDeleteController.js')
17 const TeamStatisticEditController = require('../Controllers/CoachControllers/TeamStatisticEditController.js')
18
19 const CheckNewPlayerTokenController = require('../Controllers/CreateUserController/CheckNewPlayerTokenController.js')
20 const CheckIfUserExistsController = require('../Controllers/CreateUserController/CheckIfUserExistsController.js')
21 const CreateNewPlayerController = require('../Controllers/CreateUserController/CreateNewPlayerController.js')
22
23 // MIDDLEWARE
24 const authenticateToken = require('../Middleware/authenticateToken')
25 const authenticateCoach = require('../Middleware/authenticateCoach')
26
27 // ROUTES
28
29
30 Router.get('/', authenticateToken, authenticateCoach, CoachLaunchViewController)
31 Router.get('/getCreatePlayerToken', authenticateToken, authenticateCoach, CreateNewPlayerTokenController)
32 Router.get('/list', authenticateToken, authenticateCoach, UserListController)
33 Router.get('/statistic', authenticateToken, authenticateCoach, GetPlayerStatistics)
34 Router.get('/data', authenticateToken, authenticateCoach, UserIdViewDataController)
35 //Router.get('/equipment/rented', authenticateToken, authenticateCoach, UserViewEquipmentController)
36 Router.post('/check', checkNewPlayerTokenController)
37 Router.get('/list', authenticateToken, authenticateCoach, UserListController)
38 Router.post('/checkUser', CheckIfUserExistsController)
39 Router.post('/create/:token', CreateNewPlayerController)
40
41 Router.post('/statistic/update', authenticateToken, authenticateCoach, UserUpdatePlayerStatistics)
42 Router.post('/notification/update', authenticateToken, authenticateCoach, NotificationEditController)
43 Router.post('/equipment/update', authenticateToken, authenticateCoach, EquipmentAddController)
44 Router.delete('/equipment/delete/:id', authenticateToken, authenticateCoach, EquipmentDeleteController)
45 Router.post('/teamstatistic/update', authenticateToken, authenticateCoach, TeamStatisticEditController)
46
47 module.exports = Router
```

```
1  const { Equipment, User, Teams } = require("../Models/models")
2
3
4
5  const EquipmentAddController = async(req,res) => {
6    try {
7      const {descr} = req.body;
8      console.log('nazwa i rozmiar', descr)
9
10     const teamId = await User.findOne({
11       attributes: ['id'],
12       where: {
13         username: req.user.name
14       },
15       include: [
16         {model: Teams,
17          attributes:['id'],
18        }
19      ]
20    })
21
22    console.log('id teamu w eq: ',teamId.team.id)
23
24    const add_equip_team = await Equipment.create({
25
26      descr: descr,
27      team_id: teamId.team.id,
28      available: 'true'
29
30    }
31    )
32
33    return res.status(201).json({ "message": 'created' })
34
35  } catch (error) {
36    console.error("Error adding equipments", error);
37    return res.json({ detail: error });
38  }
39
40
41  module.exports = EquipmentAddController
```

Poniżej widzimy modele wykorzystywane w tej funkcjonalności, definiują one strukturę danych wyposażenia oraz zespołów.

```
413     class Equipment extends Model {}
414     Equipment.init(
415       {
416         id:{
417           type: DataTypes.INTEGER,
418           allowNull: false,
419           primaryKey: true,
420           autoIncrement: true
421         },
422         descr:{
423           type: DataTypes.STRING,
424           allowNull: true,
425         },
426         available:{
427           type: DataTypes.INTEGER,
428           allowNull: true
429         },
430         team_id:{
431           type: DataTypes.INTEGER,
432           allowNull: false,
433         }
434       },
435     ),
436     {
437       sequelize,
438       modelName: 'equipments'
439     }
440   )
441 }
```

```
466     class Teams extends Model {}
467     Teams.init(
468         {
469             id:{
470                 type: DataTypes.INTEGER,
471                 allowNull:false,
472                 primaryKey:true,
473                 autoIncrement:true
474             },
475             name:{
476                 type: DataTypes.STRING,
477                 allowNull:false,
478             },
479             coach_id:{
480                 type:DataTypes.INTEGER,
481                 allowNull:false
482             },
483             team_message_title:{
484                 type:DataTypes.STRING,
485                 allowNull:true
486             },
487             team_message:{
488                 type:DataTypes.TEXT,
489                 allowNull:true
490             }
491         },
492         {
493             sequelize,
494             modelName:'teams'
495         }
496     )
497 )
```

Część frontendowa:

```
1  import "../../../../Style/Main/MainTemplate.css"
2  import NavBar from "../../NavBar"
3  import AddEquipmentPane from "../../CoachView/ManageEquipment/AddEquipmentPane";
4  import AvailableEquipmentPaneCoach from "../../CoachView/ManageEquipment/AvailableEquipmentPaneCoach";
5  import RentedEquipmentPaneLarge from "../../EquipmentView/RentedEquipmentPaneLarge"
6  import { useState, useEffect } from "react";
7
8
9
10 const ManageEquipmentView = () => {
11
12     // DAWID SZYMONIK -----
13     const [PageContent, setPageContent] = useState('')
14
15     const logoutHandler = (event) => {
16         const refsh_token = localStorage.getItem('refresh_token')
17         const data = { "token": refsh_token }
18
19         fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/auth/revoke_token/`, {
20             mode: 'cors',
21             method: 'POST',
22             headers: { "Content-Type": "application/json" },
23             body: JSON.stringify(data)
24         }).then(response => { response.json() }).then(data => {
25             console.log(data)
26             localStorage.removeItem('access_token')
27             localStorage.removeItem('refresh_token')
28             window.location.href = '/login/'
29         }).catch(err => { console.log(err) })
30     }
31
32
33     useEffect(() => {
34
35         fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/equipment/`, {
36             mode: 'cors',
37             method: 'GET',
38             headers: {
39                 "authorization": `Bearer ${localStorage.getItem('access_token')}`
40             }
41         }).then(response => {
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
315
316
317
317
318
319
319
320
321
322
322
323
324
324
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1
```

```

41   }).then(response => {
42
43     if (response.status == 403) {
44       throw new Error("access_token expired")
45     }
46     else return response.json()
47
48   }).then(data => {
49     setPageContent(data.content)
50
51   }).catch(err => {
52     if (err == 'Error: access_token expired') {
53       if (window.confirm("Sesja wygasła. Czy chcesz ją odnowić?")) {
54
55         /// ODNOWIENIE SESJI
56         fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/auth/refresh_token/`, {
57           mode: 'cors',
58           method: 'POST',
59           headers: { "Content-Type": "application/json", "Access-Control-Allow-Origin": "allow" },
60           body: JSON.stringify({ token: `${localStorage.getItem('refresh_token')}` })
61         })
62         .then(response => response.json())
63         .then(data => {
64           localStorage.setItem('access_token', data.accessToken)
65           window.location.reload()
66         })
67
68       } else {
69         logoutHandler()
70       }
71     }
72   })
73 }
74
75 const [sharedState, setSharedState] = useState(false);
76
77 const toggleSharedState = () => {setSharedState(!sharedState)}
78
79
80
81   return (
82     <div>
83       <div id="box">
84         <div id="bar">
85           | <NavBar/>
86         </div>
87         <div id="leftSide">
88           <div className="headers">Dodaj Nowy Sprzęt</div>
89           <AddEquipmentPane toggleSharedState={toggleSharedState} sharedState={sharedState}/>
90           <div className="headers">Dostępny Sprzęt</div>
91           <AvailableEquipmentPaneCoach toggleSharedState={toggleSharedState} sharedState={sharedState}/>
92         </div>
93         <div id="rightSide">
94           <div className="headers">Wypożyczony Sprzęt</div>
95           <RentedEquipmentPanelLarge toggleSharedState={toggleSharedState} sharedState={sharedState}/>
96           <div id="bottom"></div>
97         </div>
98       </div>
99     </div>
100   );
101 }
102
103 export default ManageEquipmentView

```

Funkcjonalność "dodawanie sprzętu sportowego" jest realizowana za pomocą kontrolera EquipmentAddController. Kontroler ten jest odpowiedzialny za obsługę żądania, które ma na celu dodanie nowego sprzętu sportowego do systemu.

- Na początku importowane są niezbędne moduły i modele takie jak: Equipment, User, Teams.
- Kontroler pobiera dane sprzętu.

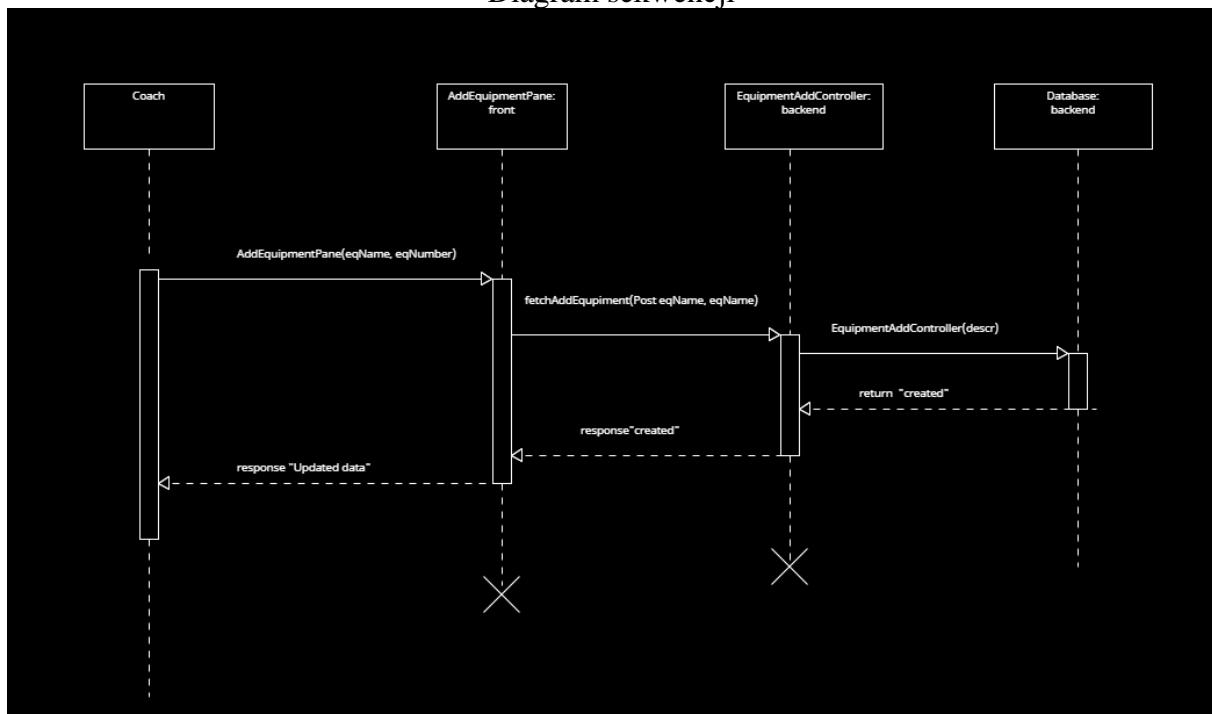
- Kontroler następnie próbuje znaleźć zalogowanego trenera na podstawie jego nazwy użytkownika, a także drużyny, które ten trener prowadzi.
- Jeśli trener i jego drużyny zostały znalezione, kontroler dodaje nowy sprzęt do bazy danych.

Funkcjonalność "dodawanie sprzętu sportowego" polega na przetwarzaniu żądania HTTP, uwierzytelnianiu trenera, weryfikacji jego uprawnień, a następnie dodaniu nowego sprzętu do bazy danych.

Przykład użycia:

1. Warunki
 - Trener jest zalogowany do systemu.
2. Przebieg
 - Trener loguje się do systemu za pomocą swojego konta.
 - System uwierzytelnia trenera i generuje token uwierzytelniający.
 - Trener przechodzi do sekcji zarządzania sprzętem w aplikacji.
 - Trener wybiera opcję dodania nowego sprzętu.
 - Trener wprowadza szczegóły nowego sprzętu, takie jak opis (np. "Piłka nożna, rozmiar 5").
 - Trener wysyła żądanie POST na endpoint /equipment/add, aby dodać nowy sprzęt.

Diagram sekwencji



-1.3.4.4 Usuwanie sprzętu

```
1  require('dotenv').config()
2  //const util = require('util')
3  const cors = require('cors')
4  const express = require('express')
5  const app = express()
6
7
8  // DATABASE
9  const {ConnectDB} = require('../Database/ConnectDB')
10
11 // ROUTERS
12 const LoginRouter = require('../Routes/AuthRouter')
13 const AppRouter = require('../Routes/AppRouter')
14 const TeamStatsRouter = require('../Routes/TeamStatsRouter')
15 const UserRouter = require('../Routes/UserRouter')
16 const EquipmentRouter = require('../Routes/EquipmentRouter')
17 const CoachRouter = require('../Routes/CoachRouter')
18
19
20 // MIDDLEWARE
21 app.use(express.json())
22 app.use(cors({origin: "http://localhost:3000"}))
23
24 // PATHS
25 app.use('/auth', LoginRouter)
26 app.use('/app', AppRouter)
27 app.use('/team_stats', TeamStatsRouter)
28 app.use('/user', UserRouter)
29
30 app.use('/equipment', EquipmentRouter)
31 app.use('/coach', CoachRouter)
32
33
34
35 const PORT = process.env.SERVER_PORT || 3000
36 ConnectDB().then(
37   () => app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`))}
38 )
39
```



```
1  const express = require('express')
2  const Router = express.Router()
3
4
5  // CONTROLLERS
6  const CoachLaunchViewController = require('../Controllers/CoachControllers/CoachLaunchViewController')
7  const CreateNewPlayerTokenController = require('../Controllers/CreateUserController/CreateNewPlayerTokenController')
8  const UserListController = require('../Controllers/CoachControllers/UserListController.js')
9  const UserIdViewDataController = require('../Controllers/CoachControllers/UserViewDataController.js')
10
11 const GetPlayerStatistics = require('../Controllers/CoachControllers/UserViewStatisticController.js')
12 const UserUpdatePlayerStatistics = require('../Controllers/CoachControllers/UserUpdatePlayerStatistics.js')
13 const UserViewEquipmentController = require('../Controllers/CoachControllers/UserViewEquipmentController')
14 const NotificationEditController = require('../Controllers/CoachControllers/NotificationEditController.js')
15 const EquipmentAddController = require('../Controllers/CoachControllers/EquipmentAddController.js')
16 const EquipmentDeleteController = require('../Controllers/CoachControllers/EquipmentDeleteController.js')
17 const TeamStatisticEditController = require('../Controllers/CoachControllers/TeamStatisticEditController.js')
18
19 const checkNewPlayerTokenController = require('../Controllers/CreateUserController/CheckNewPlayerTokenController')
20 const CheckIfUserExistsController = require('../Controllers/CreateUserController/CheckIfUserExistsController')
21 const CreateNewPlayerController = require('../Controllers/CreateUserController/CreateNewPlayerController.js')
22
23 // MIDDLEWARE
24 const authenticateToken = require('../Middleware/authenticateToken')
25 const authenticateCoach = require('../Middleware/authenticateCoach')
26
27 // ROUTES
28
29
30 Router.get('/', authenticateToken, authenticateCoach, CoachLaunchViewController)
31 Router.get('/getCreatePlayerToken', authenticateToken, authenticateCoach, CreateNewPlayerTokenController)
32 Router.get('/list', authenticateToken, authenticateCoach, UserListController)
33 Router.get('/statistic', authenticateToken, authenticateCoach, GetPlayerStatistics)
34 Router.get('/data', authenticateToken, authenticateCoach, UserIdViewDataController)
35 //Router.get('/equipment/rented', authenticateToken, authenticateCoach, UserViewEquipmentController)
36 Router.post('/check', checkNewPlayerTokenController)
37 Router.get('/list', authenticateToken, authenticateCoach, UserListController)
38 Router.post('/checkUser', CheckIfUserExistsController)
39 Router.post('/create/:token', CreateNewPlayerController)
40
41 Router.post('/statistic/update', authenticateToken, authenticateCoach, UserUpdatePlayerStatistics)
42 Router.post('/notification/update', authenticateToken, authenticateCoach, NotificationEditController)
43 Router.post('/equipment/update', authenticateToken, authenticateCoach, EquipmentAddController)
44 Router.delete('/equipment/delete/:id', authenticateToken, authenticateCoach, EquipmentDeleteController)
45 Router.post('/teamstatistic/update', authenticateToken, authenticateCoach, TeamStatisticEditController)
46
47 module.exports = Router
```

```
1  controllers > CoachControllers > js EquipmentDeleteController.js > ...
2  const { Equipment } = require("../Models/models");
3
4
5  const EquipmentDeleteController = async(req,res) =>{
6      try {
7
8          const equipId = req.params.id;
9
10         console.log("usuniety sprzet id", equipId)
11
12
13         const delete_eq = await Equipment.destroy({
14             where:{
15                 id: equipId
16             }
17         })
18         if (delete_eq) {
19             return res.json({ message: 'deleted' });
20         } else {
21             return res.status(404).json({ detail: 'Equipment not found' });
22         }
23
24
25     } catch (error) {
26         console.error("Error deleting equip:", error);
27         return res.status(500).json({ detail: error });
28     }
29
30 }
31
32 module.exports = EquipmentDeleteController
```

Część frontendowa:

```
1 import '../../../../../Style/CoachView/ManageEquipment/AvailableEquipmentPaneCoach.css'
2 import { useState, React, useEffect} from "react"
3
4 const AvailableEquipmentPaneCoach = ({toggleSharedState, sharedState}) => {
5   const [aEquipment, setAEquipment] = useState([])
6
7 // DAWID SZYMONIK -----
8
9   const fetchAvailableEquipmentPaneCoach = () => {
10     fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/equipment/available/`, {
11       mode: 'cors',
12       method: 'GET',
13       headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
14     }).then(response => response.json()).then(data => {
15       if (data.detail) {
16         console.log(data.detail)
17       } else {
18
19         //console.log(data.eq)
20         // for(let i=0; i<data.eq.length; i++){
21         //   console.log(data.eq[i])
22         //   setAEquipment([])
23         //}
24         setAEquipment(data.eq)
25         console.log(aEquipment)
26       }
27     })
28   }
29
30   useEffect(() => {
31     fetchAvailableEquipmentPaneCoach()
32   }, [sharedState])
33 }
```

```

35  // REENT EQUIPMENT
36  const rentEq = (id) => {
37    fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/equipment/rent/${id}/` , {
38      mode: 'cors',
39      method: 'PATCH',
40      headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
41    }).then(response => response.json()).then(data => {
42      if(data.message == 'created'){
43        toggleSharedState()
44        fetchAvailableEquipmentPaneCoach()
45      }
46    })
47  }
48
49
50 // DAWID SZYMONIK - koniec -----
51 const deleteEq = (id) => {
52   fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/equipment/delete/${id}/` , {
53     mode: 'cors',
54     method: 'DELETE',
55     headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
56   }).then(response => response.json()).then(data => {
57     if(data.message == 'deleted'){
58       setAEquipment(prevEquipment => prevEquipment.filter(item => item.id !== id));
59       toggleSharedState()
60     }
61   })
62 }
63
64
65   return (
66     <div id="availableEquipmentPaneCoach" className='paneShadow'>
67       <div id="scrollBoxAC" className='redScrollbar'>
68         {aEquipment.map(item => {
69           if(item.available == true){
70             return(
71               <p key={item.id}>
72                 <span className="bStyle listStyle">
73                   {item.descr}
74                 </span>
75                 <div className="avEqBtn">
76                   <div className="eqBtnPos" style={{display: 'flex', gap: '10px'}}>
77                     <div className="eqButton" onClick={() => {rentEq(item.id)}}>Wypożycz</div>
78                     <div className="redBtnPos" style={{border: '1px solid red', padding: '2px 10px', border-radius: '5px'}}>
79                       <div className="smallRedBtn" onClick={() => {deleteEq(item.id)}}>Usuń</div>
80                     </div>
81                   </div>
82                 </div>
83               )>
84             )
85           )
86         );
87       });
88     }
89   )
90   export default AvailableEquipmentPaneCoach

```

Funkcjonalność usuwanie sprzętu sportowego" pozwala trenerowi na usunięcie istniejącego sprzętu z systemu.

-Na początku widzimy importowanie niezbędnych modułów i modeli, w tym przykładu jest to model equipment. Nie dodano w tym miejscu zdjęć kodu, pokazującego tworzenie modeli, gdyż został on załączony przy innej funkcjonalności.

-Następnie mamy definicję kontrolera equipmentDeleteController.

-Kontroler pobiera identyfikator sprzętu z parametrów żądania.

-Kontroler próbuje usunąć sprzęt z bazy danych na podstawie pobranego identyfikatora:

Equipment.destroy: Metoda Sequelize używana do usunięcia rekordu z bazy danych.

where: { id: equipId }: Warunek, który określa, że usuwany rekord musi mieć identyfikator równy equipId.

-Jeśli sprzęt został pomyślnie usunięty, kontroler zwraca odpowiedź z wiadomością "deleted". W przeciwnym razie, zwraca odpowiedź z kodem statusu 404 i komunikatem "Equipment not found".

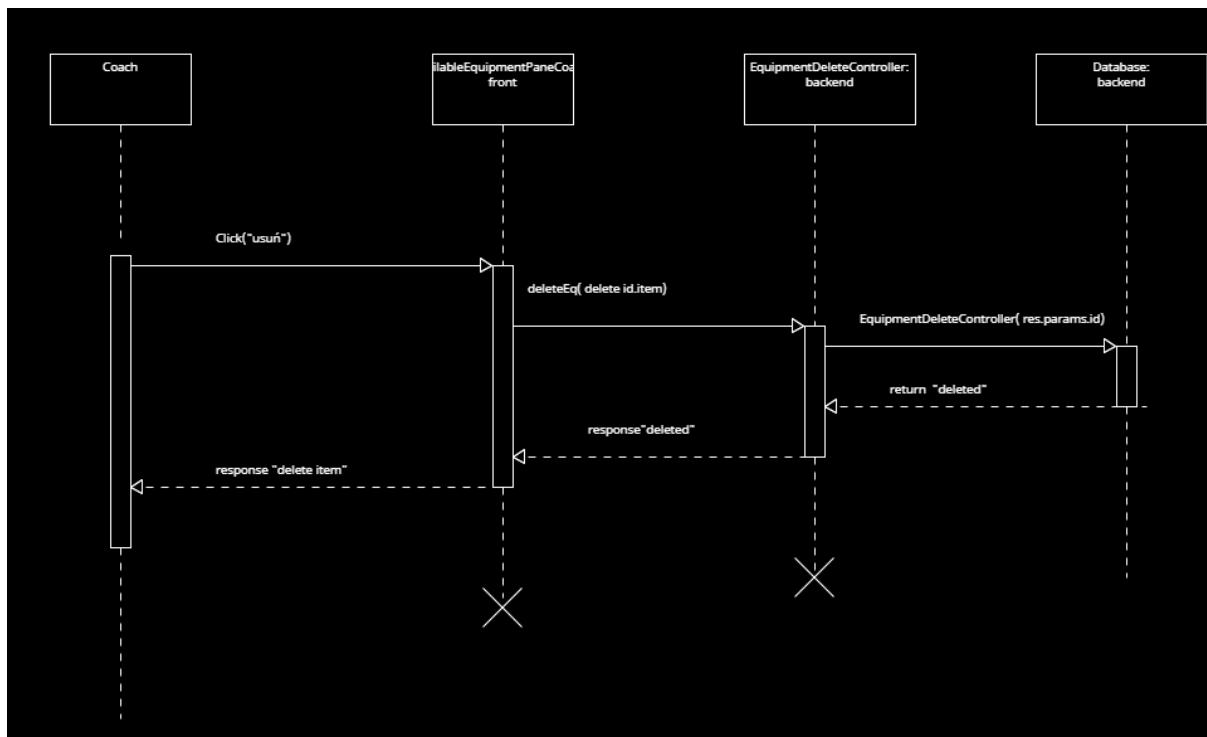
Funkcjonalność "usuwanie sprzętu sportowego" polega na przetwarzaniu żądania HTTP DELETE, uwierzytelnianiu trenera, weryfikacji uprawnień, a następnie usunięciu wpisu sprzętu z bazy danych.

Przykład użycia:

1. Przebieg

- Uwierzytelnianie trenera przy użyciu middleware authenticateToken i authenticateCoach.
- Pobieranie identyfikatora sprzętu z parametrów żądania.
- Usunięcie sprzętu z bazy danych na podstawie identyfikatora.
- Zwracanie odpowiedzi o pomyślnym usunięciu sprzętu lub komunikatów o błędach.

Diagram sekwencji



Kacper Tułecki

Pokazanie statystyk drużyny od strony gracza

Frontend

Importowanie modułów

```
import { useState, useEffect } from 'react';
import '../Style/TeamStatistic/TeamInformation.css'
```

Kod importuje z biblioteki React 'useState' i 'useEffect' oraz plik CSS, który zawiera style dla tego komponentu.

Definicja komponentu 'TeamInformation'

```
const TeamInformation = () => {
```

Tworzymy komponent funkcyjny 'TeamInformation'

Hooki 'useState'

```
const [wins, setWins] = useState('')
const [loses, setLoses] = useState('')
const [draws, setDraws] = useState('')
const [topScorer, setTopScorer] = useState('')
const [leastCards, setLeastCards] = useState('')
```

Tworzone zostają pięć stanów za pomocą hooka ‘useState’ do przechowywania odpowiednio liczby wygranych meczy, liczbę przegranych meczy, liczbę remisów, nazwę najlepszego strzelca i osobę z najmniejszą ilością kartek

Hook ‘useEffect’

```
useEffect(() => {
  if(localStorage.getItem('team_stats.matches_won') !== null){
    setWins(localStorage.getItem('team_stats.matches_won'))
    setLoses(localStorage.getItem('team_stats.matches_lost'))
    setDraws(localStorage.getItem('team_stats.matches_drawn'))
    setTopScorer(localStorage.getItem('team_stats.top_scorer'))
    setLeastCards(localStorage.getItem('team_stats.least_cards'))
  }
  else{
    fetch(`process.env.REACT_APP_SERVER_ADDRESS)/team_stats/statistics/`, {
      mode: 'cors',
      method: 'GET',
      headers: {"Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}`},
    }).then(response=>response.json()).then(data=>{
      if(data.detail){
        console.log(data.detail)
      }else{
        console.log(data.stats)
        setWins(data.stats.player.team_stat.matches_won)
        setLoses(data.stats.player.team_stat.matches_lost)
        setDraws(data.stats.player.team_stat.matches_drawn)
        setTopScorer(data.stats.player.team_stat.top_scorer)
        setLeastCards(data.stats.player.team_stat.least_cards)

        localStorage.setItem('team_stats.matches_won', data.stats.player.team_stat.matches_won)
        localStorage.setItem('team_stats.matches_lost', data.stats.player.team_stat.matches_lost)
        localStorage.setItem('team_stats.matches_drawn', data.stats.player.team_stat.matches_drawn)
        localStorage.setItem('team_stats.top_scorer', data.stats.player.team_stat.top_scorer)
        localStorage.setItem('team_stats.least_cards', data.stats.player.team_stat.least_cards)
      }
    })
  }
})
```

Zostaje użyty hook ‘useEffect’, który jest używany do pobrania danych. Najpierw sprawdzamy, czy dane są zapisane w ‘localStorage’. Jeśli tak, to ustawiamy stany za pomocą tych danych. W przeciwnym razie, wysyłane jest żądanie ‘fetch’ do serwera, aby pobrać dane, a następnie zapisujemy je w stanach i w ‘localStorage’.

Renderowanie komponentu

```
return (
    <div id="informationmain" className='paneShadow'>
        <br/>
        <div className='informationblocks'>
            <div className='informationleft bStyle'>Ilość Wygranych Meczy:</div>
            <div className='informationright'>{wins}</div>
        </div>
        <div className='informationblocks'>
            <div className='informationleft bStyle'>Ilość Przegranych Meczy:</div>
            <div className='informationright'>{loses}</div>
        </div>
        <div className='informationblocks'>
            <div className='informationleft bStyle'>Ilość Remisów:</div>
            <div className='informationright'>{draws}</div>
        </div>
        <div className='informationblocks'>
            <div className='informationleft bStyle'>Najlepszy Strzelec:</div>
            <div className='informationright'>{topScorer}</div>
        </div>
        <div className='informationblocks'>
            <div className='informationleft bStyle'>Najmniejsza Ilość Kartek:</div>
            <div className='informationright'>{leastCards}</div>
        </div>
        <br/><br/>
    </div>
);
```

Komponent renderuje dane statystyk drużyny, które są wyświetlane w oddzielnych blokach z odpowiednim opisem

Eksportowanie komponentu

```
export default TeamInformation;
```

Na końcu eksportujemy komponent ‘TeamInformation’, aby mógł być używany w innych częściach aplikacji.

Backend

```
// require('dotenv').config()
// const util = require('util')
const cors = require('cors')
const express = require('express')
const app = express()

// DATABASE
const {ConnectDB} = require('../Database/ConnectDB')

// ROUTERS
const LoginRouter = require('../Routes/AuthRouter')
const AppRouter = require('../Routes/AppRouter')
const TeamStatsRouter = require('../Routes/TeamStatsRouter')
const UserRouter = require('../Routes/UserRouter')
const EquipmentRouter = require('../Routes/EquipmentRouter')
const CoachRouter = require('../Routes/CoachRouter')

// MIDDLEWARE
app.use(express.json())
app.use(cors({origin: "http://localhost:3000"}))

// PATHS
app.use('/auth', LoginRouter)
app.use('/app', AppRouter)
app.use('/team_stats', TeamStatsRouter)
app.use('/user', UserRouter)

app.use('/equipment', EquipmentRouter)
app.use('/coach', CoachRouter)

const PORT = process.env.SERVER_PORT || 3000
ConnectDB().then(
  app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)})
)
```

```
const express = require('express')
const Router = express.Router()

// CONTROLLERS
const TeamStatsViewController = require('../Controllers/TeamStatsControllers/TeamStatsViewController')
const TeamStatisticsController = require('../Controllers/TeamStatsControllers/TeamStatisticsController')
const TeamInfoController = require('../Controllers/TeamStatsControllers/TeamInfoController')
const QuestionaryController = require('../Controllers/TeamStatsControllers/QuestionaryController')
const TeamFormViewController = require('../Controllers/TeamStatsControllers/TeamFormViewController')

// MIDDLEWARE
const authenticateToken = require('../Middleware/authenticateToken')
//const TeamFormViewController = require('../Controllers/TeamStatsControllers/TeamFormViewController')

// ROUTES
Router.get('/', authenticateToken, TeamStatsViewController)
Router.get('/statistics', authenticateToken, TeamStatisticsController)
Router.get('/info', authenticateToken, TeamInfoController)
Router.get('/form', authenticateToken, TeamFormViewController)
Router.get('/questionary', authenticateToken, QuestionaryController)

module.exports = Router
```

Potrzebne modele do funkcjonalności

```
class Team_stats extends Model {}
Team_stats.init(
{
    team_id:{
        type: DataTypes.INTEGER,
        allowNull:false,
        primaryKey: true,
        autoIncrement: true
    },
    season_id:{
        type: DataTypes.INTEGER,
        allowNull:true,
    },
    matches_won:{
        type: DataTypes.INTEGER,
        allowNull:true,
        defaultValue:0
    },
    matches_lost:{
        type: DataTypes.INTEGER,
        allowNull:true,
        defaultValue:0
    },
    matches_drawn:{
        type: DataTypes.INTEGER,
        allowNull:true,
        defaultValue:0
    },
    top_scorer:{
        type: DataTypes.STRING,
        allowNull:true,
    },
    least_cards:{
        type: DataTypes.STRING,
        allowNull:true,
    },
},
{
    sequelize,
    modelName: 'team_stats'
}
)
```

```
class User extends Model {
  User.init(
    {
      id: [
        type: DataTypes.INTEGER,
        allowNull: false,
        primaryKey: true,
        autoIncrement:true
      ],
      username: {
        type: DataTypes.STRING,
        allowNull: false
      },
      password: {
        type: DataTypes.STRING,
        allowNull: false
      },
      first_name: {
        type: DataTypes.STRING,
        allowNull: false
      },
      last_name: {
        type: DataTypes.STRING,
        allowNull: false
      },
      email: {
        type: DataTypes.STRING,
        allowNull: false
      },
      role: {
        type: DataTypes.ENUM('Player', 'Coach', 'Admin'),
        allowNull: false
      }
    }
  )
}
```

```
is_active: {
  type: DataTypes.BOOLEAN,
  allowNull: false,
  defaultValue:false
},
phone_number: {
  type: DataTypes.STRING,
  allowNull: true,
},
resetpasswordtoken: {
  type: DataTypes.STRING,
  allowNull: true
},
resetpasswordtokenexpirdate: {
  type: DataTypes.TIME,
  allowNull: true
},
emailverifytoken: {
  type: DataTypes.STRING,
  allowNull: true
},
emailverifytokenexpirdate: {
  type: DataTypes.TIME,
  allowNull: true
},
},
{
  sequelize,
  modelName: 'users'
}
```

```
class Players extends Model {}
Players.init(
{
    id:{
        type: DataTypes.INTEGER,
        allowNull:false,
        primaryKey:true
    },
    date_of_birth:{
        type: DataTypes.DATE,
        allowNull:true
    },
    height:{
        type:DataTypes.INTEGER,
        allowNull:true
    },
    weight:{
        type:DataTypes.INTEGER,
        allowNull:true
    },
    boot_size:[
        type:DataTypes.INTEGER,
        allowNull:true
    ],
    notes:{
        type:DataTypes.TEXT,
        allowNull:true
    },
    team_id:{
        type:DataTypes.INTEGER,
        allowNull:false,
    }
},
{
    sequelize,
    modelName: 'players'
}
)
```

Importowanie modeli

```
const {User, Players, Team_stats} = require('../Models/models')
```

Kod importuje modele 'User', 'Players' i 'Team_stats' z pliku 'models'

Definicja kontrolera 'TeamStatisticController'

```
const TeamStatisticsController = async (req, res) => {
  try {
    const team_stats = await User.findOne({
      attributes: ['id'],
      include: [
        {
          model: Players,
          attributes: ['id'],
          include: [
            {
              model: Team_stats,
              attributes: ['matches_won', 'matches_lost', 'matches_drawn', 'top_scorer', 'least_cards']
            }
          ]
        },
        where: {
          username: req.user.name
        }
      }
    });

    if(team_stats.length != 0){
      return res.json({stats:team_stats})
    }

  } catch (error) {
    return res.json({'detail':error})
  }
}

module.exports = TeamStatisticsController
```

'try' próbujemy pobrać dane z bazy danych.

Przy pomocy 'User.findOne':

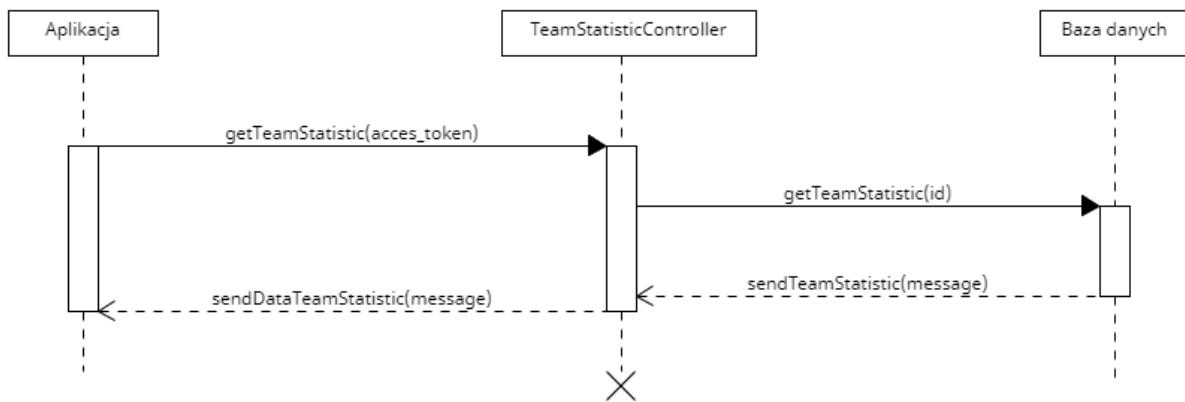
- 'findOne' znajdujemy pierwszego użytkownika, który spełnia podane warunki
- 'attributes : ['id']' ogranicza wynik atrybutu do 'id' użytkownika
- 'include' określa, że chcemy włączyć powiązane dane z modeli 'Players' i 'Team_stats'
- 'where: {username: req.user.name}' filzuje użytkownika według nazwy użytkownika pobranej z żądania

Następnie następuje sprawdzanie wyników. Jeśli 'team_stats' istnieje, to dane są zwracane w formacie JSON.

Block 'catch' zostanie uruchomiony, gdy wystąpi błąd, a odpowiedź zostaje wysłana w formacie JSON, która zawiera rodzaj błędu

Na końcu jest eksportowany kontroler 'TeamStatisticsController'.

Diagram sekwencyjny



Rejestracja zawodnika

Frontend

```
Importowanie modułów
> src > Component > BeforeLogin > RegistrationForm.js > RegistrationForm
import '../../../../../Style/BeforeLogin/RegistrationForm.css'
import { useState, useEffect, useRef } from 'react';

import { validEmail } from '../Regex.js';
import {useParams} from 'react-router-dom'
```

Kod importuje CSS, hooki Reacta, funkcję ‘validEmail’ do walidacji e-maila oraz hook ‘esuParams’ do pobierania parametrów z URL.

Definicja komponentu ‘RegistrationForm’ oraz Hooki ‘useState’

```
const RegistrationForm = () => {
  const [page, setPage]=useState(1);
  const [name, setName]=useState('');
  const [surname, setSurname]=useState('');
  const [email, setEmail]=useState('');
  const [login, setLogin]=useState('');
  const [password, setPassword]=useState('');
  const [passwordRepeat, setPasswordRepeat]=useState('');
  const [birthday, setBirthday]=useState('');
  const [height, setHeight]=useState();
  const [bootnumber, setBootNumber]=useState();
  const [weight, setWeight]=useState();
  const [islogin, setIsLogin]=useState(true);
  const [ismail, setIsMail]=useState(true);
  const [ispassword, setIsPassword]=useState(true);
  const [information, setInformation]=useState('');
  const [informationpassword, setInformationPassword]=useState('');
```

Tworzymy komponent ‘RegistrationForm’ oraz używamy hooka ‘useState’ do definiowania stanów dla pól formularza oraz zarządzania aktualną stroną formularza, ich stanami i informacjami.

Funkcja 'useDisplayImage'

```
const [image, setImage] = useState("");
💡 const imageRef = useRef(null);
function useDisplayImage() {
  const [result, setResult] = useState("");

  function uploader(e) {
    const imageFile = e.target.files[0];

    const reader = new FileReader();
    reader.addEventListener("load", (e) => {
      setResult(e.target.result);
    });

    reader.readAsDataURL(imageFile);
  }

  return { result, uploader };
}

const { result, uploader } = useDisplayImage();
```

Definiowana jest funkcja do wyświetlania obrazu po jego wgraniu.

Hook 'useEffect' oraz funkcja 'checkToken'

```
const checkToken = (token) => {
  fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/check`, {
    method: 'POST',
    mode: 'cors',
    headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
    body: JSON.stringify({
      'token': token
    })
  }).then(response => response.json())
  .then(data => {
    console.log(data)
  })
}

useEffect(() => {
  checkToken(token);
}, [ ])
}
```

Używamy 'useEffect', aby wywołać funkcję 'checkToken', która wysyła żądanie POST do serwera, aby sprawdzić token.

Funkcja nawigacyjna i wysyłania formularza

```
const gotoFirstPageRegistration = () =>{
  setPage(1);
}

const sendRegistrationPage = () =>{
  fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/create/${token}`, {
    method: 'POST',
    mode: 'cors',
    headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
    body: JSON.stringify({
      login: login,
      name: name,
      surname: surname,
      password: password,
      secondpassword: passwordRepeat,
      email: email,
      birthday: birthday,
      height: height,
      weight: weight,
      bootnumber: bootnumber,
    })
  }).then(response => response.json())
  .then(data => {
    if(data.message=='successfully created'){
      document.location.href='/login'
    }
  })
  setPage(1);
}
```

Funkcje obsługują przejście do pierwszej strony formularza i wysyłanie danych rejestracyjnych na serwer.

Funkcja 'cheeckPasswords'

```

const CheckPasswords = (e) =>{
    e.preventDefault();
    if(!validEmail.test(email))
    {
        setIsMail(false);
        setInformation("Zle napisany email");
    }
    else {
        setIsMail(true);
        setInformation('');

        fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/checkUser`, {
            method: 'POST',
            mode: 'cors',
            headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
            body: JSON.stringify({
                username: login,
                email: email,
            })
        })
        .then(response => response.json())
        .then(data => {
            if (data.message == false) {
                setInformation("Istnieje takie konto o takim emailu lub loginie.");
                setIsLogin(false);
                setIsMail(false);
            }
            else {
                setInformation("");
                setIsLogin(true);
                setIsMail(true);
            }
        })
    }
}

if(password!==passwordRepeat || password.length==0)
{
    setInformationPassword("Hasła nie są takie same / Brak hasła / Hasło jest z krótkie /");
    setIsPassword(false);
}
else
{
    setInformationPassword("");
    setIsPassword(true);
}
if(islogin && ismail && ispassword && password.length>4)
{
    setPage(2);
}
}

```

Funkcja ta waliduje dane użytkownika, sprawdza zgodność haseł oraz dostępność loginu i e-maila.

Renderowanie komponentu

```

if(page==1){return (
<div>
  <div id="registrationformmain">
    <form id="registrationformform">
      <label id="registrationformtitle"><b>Rejestracja</b></label>
      <div id="registrationformstart">
        <label id="registrationformlabelinfo">Stwórz konto</label>
        <br/>
        <label id="registrationforminformation1">(information)<br/>{informationpassword}</label>
        <div id="registrationforminformation">
          <div className="registrationformblocks">
            <label className="registrationformlabel1">Imię:</label>
            <input type="text" className="registrationforminput" onChange={(e)=>setName(e.target.value)} value={name} required/>
            <label className="registrationformlabel1">Nazwisko:</label>
            <input type="text" className="registrationforminput" onChange={(e)=>setSurname(e.target.value)} value={surname} required/>
            <label className="registrationformlabel1">Mail:</label>
            {isEmail && <input type="email" className="registrationforminput" onChange={(e)=>setEmail(e.target.value)} value={email} required/>}
            {isEmail && <input type="text" className="registrationforminputbad" onChange={(e)=>setEmail(e.target.value)} value={email} required/>}
          </div>
          <div className="registrationformblocks">
            <label className="registrationformlabel1">Login:</label>
            {isLogin && <input type="text" className="registrationforminput" onChange={(e)=>setLogin(e.target.value)} value={login} required/>}
            {isLogin && <input type="text" className="registrationforminputbad" onChange={(e)=>setLogin(e.target.value)} value={login} required/>}
            <label className="registrationformlabel1">Hasło:</label>
            {isPassword && <input type="password" className="registrationforminput" id="pass" onChange={(e)=>setPassword(e.target.value)} value={password} required/>}
            {isPassword && <input type="password" className="registrationforminputbad" id="pass" onChange={(e)=>setPassword(e.target.value)} value={password} required/>}
            <label className="registrationformlabel1">Powtórz Hasło:</label>
            {isPassword && <input type="password" id="pass" className="registrationforminput" onChange={(e)=>setPasswordRepeat(e.target.value)} value={passwordRepeat} required/>}
            {isPassword && <input type="password" id="pass" className="registrationforminputbad" onChange={(e)=>setPasswordRepeat(e.target.value)} value={passwordRepeat} required/>}
          </div>
        </div>
        <div id="registrationformbuttondiv"><button id="registrationformbuttonmove" onClick={CheckPasswords}>Przejdź dalej</button></div>
      </form>
    </div>
  </div>
)
}
if(page==2){return (
<div>
  <div id="registrationformmain">
    <form id="registrationformform">
      <label id="registrationformtitle"><b>Rejestracja</b></label>
      <div id="registrationformstart">
        <label id="registrationformlabelinfo">Stwórz konto</label>
        <div id="registrationforminformation">
          <div className="registrationformblocks">
            <label className="registrationformlabel1">Data urodzenia:</label>
            <input type="date" className="registrationforminput" onChange={(e)=>setBirthday(e.target.value)} value={birthday} required/>
            <label className="registrationformlabel1">Wzrost:</label>
            <input type="number" className="registrationforminput" onChange={(e)=>setHeight(e.target.value)} value={height} required/>
            <label className="registrationformlabel1">Waga:</label>
            <input type="number" className="registrationforminput" onChange={(e)=>setWeight(e.target.value)} value={weight} required/>
          </div>
          <div className="registrationformblocks">
            <label className="registrationformlabel1">Numer buta:</label>
            <input type="number" className="registrationforminput" onChange={(e)=>setBootNumber(e.target.value)} value={bootnumber} required/>
            <label className="registrationformlabel1">Mojaj zdjcie:</label>
            <div id="imgregisterform">
              <input type="file" id="registrationformimage" onChange={(e)=>setImage(e.target.files[0]);uploader(e)} accept="image/jpeg,image/png"/><br/>
              {result && <img ref={imageRef} src={result} id="registrationformimg"/>
            </div>
          </div>
        </div>
        <div id="registrationformbuttondivsecond">
          <div className="registrationformbuttonblock">
            <button id="registrationformbuttonmove" onClick={gotoFirstPageRegistration}>Wstecz</button>
          </div>
          <div className="registrationformbuttonblock">
            <button id="registrationformbuttonreg" onClick={sendRegistrationPage}>Rejestracja</button>
          </div>
        </div>
      </div>
    </form>
  </div>
)
)}

```

Renderowanie zależy od wartości stanu 'page'. Na pierwszej stronie zbieramy podstawowe informacje, a na drugiej dodatkowe szczegóły oraz zdjęcie użytkownika.

Eksportowanie komponentu

```
export default RegistrationForm;
```

Eksportujemy komponent 'RegistrationForm', aby można było go używać w innych częściach aplikacji.

Backend

```

15 index.js > ...
1   const dotenv = require('dotenv').config()
2   //const util = require('util')
3   const cors = require('cors')
4   const express = require('express')
5   const app = express()
6
7
8   // DATABASE
9   const {ConnectDB} = require('../Database/ConnectDB')
10
11  // ROUTERS
12  const LoginRouter = require('../Routes/AuthRouter')
13  const AppRouter = require('../Routes/AppRouter')
14  const TeamStatsRouter = require('../Routes/TeamStatsRouter')
15  const UserRouter = require('../Routes/UserRouter')
16  const EquipmentRouter = require('../Routes/EquipmentRouter')
17  const CoachRouter = require('../Routes/CoachRouter')
18
19
20  // MIDDLEWARE
21  app.use(express.json())
22  app.use(cors({origin: "http://localhost:3000"}))
23
24  // PATHS
25  app.use('/auth', LoginRouter)
26  app.use('/app', AppRouter)
27  app.use('/team_stats', TeamStatsRouter)
28  app.use('/user', UserRouter)
29
30  app.use('/equipment', EquipmentRouter)
31  app.use('/coach', CoachRouter)
32
33
34
35  const PORT = process.env.SERVER_PORT || 3000
36  ConnectDB().then(
37    () => app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)})
38  )
39

1  const express = require('express')
2  const Router = express.Router()
3
4
5  // CONTROLLERS
6  const CoachLaunchViewController = require('../Controllers/CoachControllers/CoachLaunchViewController')
7  const CreateNewPlayerTokenController = require('../Controllers/CreateUserController/CreateNewPlayerTokenController')
8  const UserListController = require('../Controllers/CoachControllers/UserListController.js')
9  const UserIdViewDataController = require('../Controllers/CoachControllers/UserViewDataController.js')
9
10 const GetPlayerStatistics = require('../Controllers/CoachControllers/UserViewStatisticController.js')
11 const UserUpdatePlayerStatistics = require('../Controllers/CoachControllers/UserUpdatePlayerStatistics.js')
12 const UserViewEquipmentController = require('../Controllers/CoachControllers/UserViewEquipmentController')
13 const NotificationEditController = require('../Controllers/CoachControllers/NotificationEditController.js')
14 const EquipmentAddController = require('../Controllers/CoachControllers/EquipmentAddController.js')
15 const EquipmentDeleteController = require('../Controllers/CoachControllers/EquipmentDeleteController.js')
16 const TeamStatisticEditController = require('../Controllers/CoachControllers/TeamStatisticEditController.js')
17
18 const checkNewPlayerTokenController = require('../Controllers/CreateUserController/CheckNewPlayerTokenController.js')
19 const CheckIfUserExistsController = require('../Controllers/CreateUserController/CheckIfUserExistsController.js')
20 const CreateNewPlayerController = require('../Controllers/CreateUserController/CreateNewPlayerController.js')
21
22 // MIDDLEWARE
23 const authenticateToken = require('../Middleware/authenticateToken')
24 const authenticateCoach = require('../Middleware/authenticateCoach')
25
26 // ROUTES

```

```
// ROUTES

Router.get('/', authenticateToken, authenticateCoach, CoachLaunchViewController)
Router.get('/getCreatePlayerToken', authenticateToken, authenticateCoach, CreateNewPlayerTokenController)
Router.get('/list',authenticateToken,authenticateCoach,UserListController)
Router.get('/statistic',authenticateToken,authenticateCoach,GetPlayerStatistics)
Router.get('/data',authenticateToken,authenticateCoach,UserIdViewDataController)
//Router.get('/equipment/rented',authenticateToken,authenticateCoach,UserViewEquipmentController)
Router.post('/check', checkNewPlayerTokenController)
Router.get('/list',authenticateToken,authenticateCoach, UserListController)
Router.post('/checkUser', CheckIfUserExistsController)
Router.post('/create/:token', CreateNewPlayerController)

Router.post('/statistic/update', authenticateToken,authenticateCoach,UserUpdatePlayerStatistics)
Router.post('/notification/update', authenticateToken,authenticateCoach,NotificationEditController)
Router.post('/equipment/update', authenticateToken,authenticateCoach,EquipmentAddController)
Router.delete('/equipment/delete/:id', authenticateToken,authenticateCoach,EquipmentDeleteController)
Router.post('/teamstatistic/update', authenticateToken,authenticateCoach,TeamStatisticEditController)

module.exports = Router
```

Modele

```

class User extends Model {}
User.init(
{
  id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    primaryKey: true,
    autoIncrement:true
  },
  username: {
    type: DataTypes.STRING,
    allowNull: false
  },
  password: {
    type: DataTypes.STRING,
    allowNull: false
  },
  first_name: {
    type: DataTypes.STRING,
    allowNull: false
  },
  last_name: {
    type: DataTypes.STRING,
    allowNull: false
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false
  },
  role: {
    type: DataTypes.ENUM('Player', 'Coach', 'Admin'),
    allowNull: false
  },
  is_active: {
    type: DataTypes.BOOLEAN,
    allowNull: false,
    defaultValue:false
  },
  phone_number: {
    type: DataTypes.STRING,
    allowNull: true,
  },
  resetpasswordtoken: {
    type: DataTypes.STRING,
    allowNull: true
  },
  resetpasswordtokenexpirdate: {
    type: DataTypes.TIME,
    allowNull: true
  },
  emailverifytoken: {
    type: DataTypes.STRING,
    allowNull: true
  },
  emailverifytokenexpirdate: {
    type: DataTypes.TIME,
    allowNull: true
  },
},
{
  sequelize,
  modelName: 'users'
})

```

```
class Create_user_token extends Model{}
Create_user_token.init(
{
  id:{
    type: DataTypes.INTEGER,
    allowNull: false,
    primaryKey: true,
    autoIncrement: true
  },
  expire_date:{
    type: DataTypes.TIME,
    allowNull: false,
  },
  role: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  token: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  team_id: {
    type: DataTypes.INTEGER,
    allowNull: false
  }
},
{
  sequelize,
  modelName: 'create_user_token',
  tableName: 'create_user_token'
})

```

```
class Players extends Model {}
Players.init(
{
  id:{
    type: DataTypes.INTEGER,
    allowNull:false,
    primaryKey:true
  },
  date_of_birth:{
    type: DataTypes.DATE,
    allowNull:true
  },
  height:{
    type:DataTypes.INTEGER,
    allowNull:true
  },
  weight:{
    type:DataTypes.INTEGER,
    allowNull:true
  },
  boot_size:{
    type:DataTypes.INTEGER,
    allowNull:true
  },
  notes:{
    type:DataTypes.TEXT,
    allowNull:true
  },
  team_id:{
    type:DataTypes.INTEGER,
    allowNull:false,
  }
},
{
  sequelize,
  modelName:'players'
})

```

Kod modułu 1

```

const {User} = require('../Models/models')
const {Op} = require('sequelize')

const CheckIfUserExistsController = async (req, res) => {

    try {
        console.log(req.body.username)
        const user = await User.findOne({
            attributes: ['id'],
            where: {
                [Op.or]: [
                    {
                        username: {
                            [Op.eq]: req.body.username
                        },
                        email: {
                            [Op.eq]: req.body.email
                        }
                    }
                ]
            }
        })

        if(!user){
            return res.status(200).json({'message':true})
        }
        else{
            return res.status(200).json({'message':false})
        }

    } catch (error) {
        return res.status(400).json({'detail':error})
    }
}

module.exports = CheckIfUserExistsController

```

Na początku importujemy model 'User' z pliku 'models' oraz operator 'Op' z pakietu 'sequelize', który jest używany do budowania warunków zapytań. Kontroler 'CheckIfUserExistsController' jest funkcją, która przyjmuje obiekty 'req' i 'res' jako argumenty. W części 'try' jest wyszukiwany użytkownik, któremu odpowiada 'username' lub 'email' podany w żądaniu. Jeżeli takiego użytkownika nie znaleziono, zostaje zwracana odpowiedź ze statusem 200 i wiadomością 'true'. W przeciwnym przypadku zwracamy odpowiedź ze statusem 200 i wiadomością 'false'. W przypadku wystąpienia błędu, zwracamy odpowiedź ze statusem 400 i szczegółami błędu. Na końcu eksportujemy 'CheckIfUserExistsController', aby można było go używać w innych częściach aplikacji.

Kod modułu 2

```

const [User, Create_user_token, Players] = require('../Models/models')

const CreateNewPlayerController = async (req, res) => {
  try {
    const {
      name,
      surname,
      email,
      login,
      password,
      secondpassword,
      birthday,
      height,
      weight,
      bootnumber,
    } = req.body

    console.log(password)

    const token = await Create_user_token.findOne({
      attributes: ['team_id'],
      where: {
        token: req.params.token
      }
    })

    if(!token){
      return res.status(403).json({detail: 'token not found'})
    }else{
      const team_id = token.dataValues.team_id

      const new_user = await User.create({
        username: login,
        password: password,
        first_name: name,
        last_name: surname,
        email: email,
        is_active: false,
        role: 'Player',
      })

      if(new_user){
        const new_player = await Players.create({
          id: new_user.dataValues.id,
          date_of_birth: birthday,
          height: height,
          weight: weight,
          boot_size: bootnumber,
          team_id: team_id,
        })

        if(new_player){
          return res.status(201).json({message:'successfully created'})
        }else{
          return res.status(400).json({detail:'error creating user'})
        }
      }else{
        return res.status(400).json({detail:'error creating user'})
      }
    }

    } catch (error) {
      return res.status(400).json({detail:error.message})
    }
  }
}

module.exports = CreateNewPlayerController

```

Na początku importujemy modele 'User', 'Create_user_token' oraz 'Players' z plików modeli. Następnie definiujemy kontroler 'CreateNewPlayerController', który przyjmuje jako argumenty 'req' i 'res'. Najpierw dekonstruujemy dane z 'req.body', które zawierają informacje o nowym użytkowniku. Następnie sprawdzamy, czy nasz token jest ważny. Jeśli jest to wyciągamy 'team_id' z tokenu. Tworzymy nowego użytkownika w tabeli 'User' z danymi podanymi w żądaniu. Użytkownik jest tworzony z rolą 'Player' i domyślnie nie jest aktywny. Jeśli nowy użytkownik zostanie pomyślnie utworzony, tworzymy nowego gracza w tabeli 'Players', powiązanego z nowo utworzonym użytkownikiem i przypisanego do zespołu (team_id). Jeśli gracz zostanie pomyślnie utworzony, zwracamy odpowiedź ze statusem 201 i komunikatem "successfully created". W przeciwnym razie zwracamy odpowiedź ze statusem 400 i odpowiednim komunikatem o błędzie.

Jeśli w trakcie wykonywania zapytania wystąpi błąd, zwracamy odpowiedź ze statusem 400 i szczegółami błędu.

Na końcu eksportujemy 'CreateNewPlayerController', aby można było go używać w innych częściach aplikacji.

Moduł 3

```
require('dotenv').config()
//const util = require('util')
const cors = require('cors')
const express = require('express')
const app = express()

// DATABASE
const {ConnectDB} = require('../Database/ConnectDB')

// ROUTERS
const LoginRouter = require('../Routes/AuthRouter')
const AppRouter = require('../Routes/AppRouter')
const TeamStatsRouter = require('../Routes/TeamStatsRouter')
const UserRouter = require('../Routes/UserRouter')
const EquipmentRouter = require('../Routes/EquipmentRouter')
const CoachRouter = require('../Routes/CoachRouter')

// MIDDLEWARE
app.use(express.json())
app.use(cors({origin: "http://localhost:3000"}))

// PATHS
app.use('/auth', LoginRouter)
app.use('/app', AppRouter)
app.use('/team_stats', TeamStatsRouter)
app.use('/user', UserRouter)

app.use('/equipment', EquipmentRouter)
app.use('/coach', CoachRouter)

const PORT = process.env.SERVER_PORT || 3000
ConnectDB().then(
  app.listen(PORT, () => {console.log(`Server is listening on port ${PORT} ...`)}
)
```

```
4
5 // CONTROLLERS
6 const CoachLaunchViewController = require('../Controllers/CoachControllers/CoachLaunchViewController')
7 const CreateNewPlayerTokenController = require('../Controllers/CreateUserController/CreateNewPlayerTokenController')
8 const UserListController = require('../Controllers/CoachControllers/UserListController.js')
9 const UserIdViewDataController = require('../Controllers/CoachControllers/UserViewDataController')
10
11 const GetPlayerStatistics = require('../Controllers/CoachControllers/UserViewStatisticController')
12 const UserUpdatePlayerStatistics = require('../Controllers/CoachControllers/UserUpdatePlayerStatisticsController')
13 const UserViewEquipmentController = require('../Controllers/CoachControllers/UserViewEquipmentController')
14 const NotificationEditController = require('../Controllers/CoachControllers/NotificationEditController')
15 const EquipmentAddController = require('../Controllers/CoachControllers/EquipmentAddController')
16 const EquipmentDeleteController = require('../Controllers/CoachControllers/EquipmentDeleteController')
17 const TeamStatisticEditController = require('../Controllers/CoachControllers/TeamStatisticEditController')
18
19 const checkNewPlayerTokenController = require('../Controllers/CreateUserController/CheckNewPlayerTokenController')
20 const CheckIfUserExistsController = require('../Controllers/CreateUserController/CheckIfUserExistsController')
21 const CreateNewPlayerController = require('../Controllers/CreateUserController/CreateNewPlayerController')
22
23 // MIDDLEWARE
24 const authenticateToken = require('../Middleware/authenticateToken')
25 const authenticateCoach = require('../Middleware/authenticateCoach')
26
27 // ROUTES
```

Kod

```

const crypto = require('crypto')
const {Create_user_token} = require('../Models/models')
const {Op} = require('sequelize')

const checkNewPlayerTokenController = async (req, res) => [
    const createAccountToken = crypto.randomBytes(32).toString('hex')

    const now = new Date()
    const date = {
        year: now.getFullYear(),
        month: now.getMonth(),
        day: now.getDate(),
        hour: now.getHours(),
        minute: now.getMinutes(),
        second: now.getSeconds()
    }

    const TodayDateString = `${date.year}-${date.month}-${date.day} ${date.hour}:${date.minute}:${date.second}`

    try {
        console.log(now)
        const query = await Create_user_token.findOne({
            where: {
                token: req.body.token,
                expire_date: {
                    [Op.lt]: now
                }
            }
        })
        // obsługa dalsza
        .then(token => {
            res.json({'token': 'Found'})
        })
        .catch((err) => {
            return res.json({'detail': err})
        })
    } catch (error) {
        return res.status(400).json({'detail': error})
    }
]

module.exports = checkNewPlayerTokenController

```

Na samym początku importujemy:

- ‘crypto’ - biblioteka, używana tutaj do generowania losowego tokenu
- ‘Create_user_token’ - model reprezentujący tabelę w bazie danych
- ‘Op’ - operator logiczny z Sequelize, używany do budowania zapytań

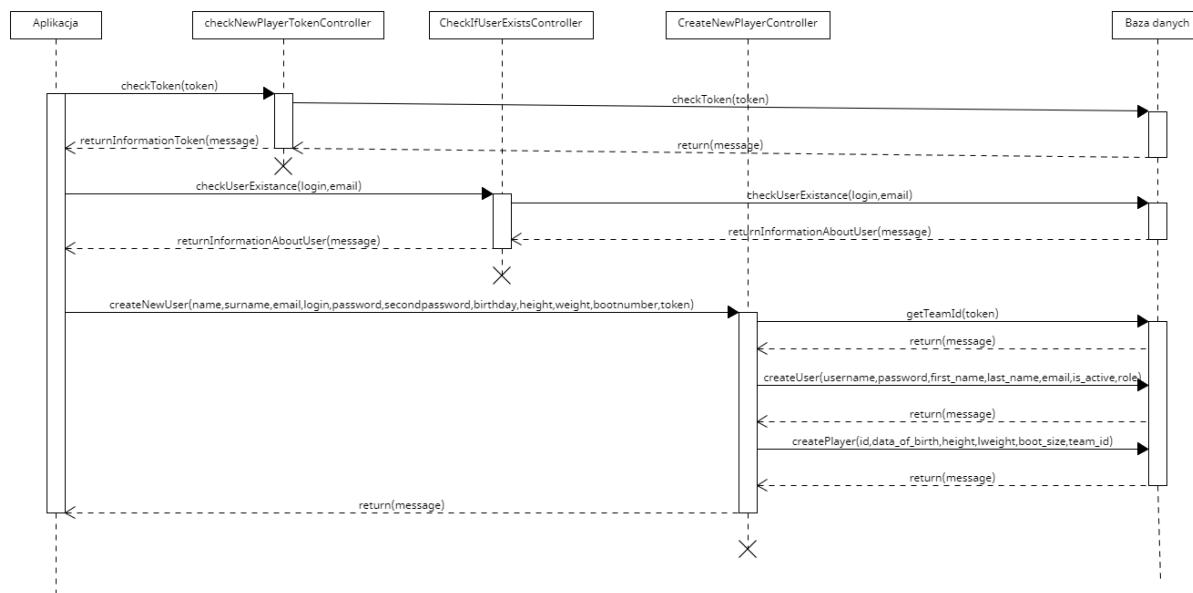
Następnie definiujemy funkcję ‘checkNewPlayerTokenController’ z argumentami ‘req’ i ‘res’. Potem generujemy losowy token składający się z 32 bajtów i konwertujemy go na format szesnastkowy oraz tworzęłańcuch znaków ‘TodayDatastring’ reprezentujący aktualną datę i czas.

W ‘try’ wykonuje zapytanie do bazy danych, szukając rekordu w tabeli ‘Create_user_token’, który ma token równy tokenowi z ‘req.body.token’ oraz datę wygaśnięcia wcześniejszą niż bieżąca data.

Następnie obsługuję wyniki zapytania. Jeśli znajdzie pasujący rekord, zwraca odpowiedź JSON z informacją, że token został znaleziony. Jeśli wystąpił błąd, zwraca odpowiedź JSON z detalami błędu.

Na końcu jest sprawdzane, czy wystąpił jakikolwiek błąd w bloku ‘try’. Jeśli wystąpił błąd, zwraca odpowiedź z kodem statusu 400 i szczegółami błędu.

Diagram sekwencyjny



Jakub Zajęc

- 1.2.4.3 Pokazanie wypożyczonego sprzętu
- 1.2.4.3 Zwrócenie wypożyczonego sprzętu

Frontend

Plik: AvailableEquipmentPane.js

Opis działania:

Komponent AvailableEquipmentPane jest odpowiedzialny za wyświetlanie dostępnego sprzętu, który użytkownik może wypożyczyć.

Przepływ danych:

```
import '../../../../../Style/EquipmentView/AvailableEquipmentPane.css';

import { useState, React, useEffect } from "react";

const AvailableEquipmentPane = ({ toggleSharedState, sharedState }) => {
    const [aEquipment, setAEquipment] = useState([]);

    const fetchAvailableEquipmentPane = () => {
        fetch(`process.env.REACT_APP_SERVER_ADDRESS}/equipment/available/`,
        {
            mode: 'cors',
            method: 'GET',
            headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
        }).then(response => response.json()).then(data => {
            if (data.detail) {
                console.log(data.detail);
            } else {
                setAEquipment(data.eq);
            }
        });
    };

    useEffect(() => {
        fetchAvailableEquipmentPane();
    }, [sharedState]);

    const rentEq = (id) => {
        fetch(`process.env.REACT_APP_SERVER_ADDRESS}/equipment/rent/${id}/`,
        {
            mode: 'cors',
        }
    );
}
```

```

        method: 'PATCH',
        headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
    }).then(response => response.json()).then(data => {
        if (data.message === 'created') {
            toggleSharedState();
            fetchAvailableEquipmentPane();
        }
    });
};

return (
    <div id="availableEquipmentPane" className='paneShadow'>
        <div id="scrollBoxA" className='redScrollbar'>
            {aEquipment.map(item => {
                if (item.available) {
                    return (
                        <p key={item.id}>
                            <span className="bStyle listStyle">
                                {item.descr}
                            </span>
                            <div className="eqButton" onClick={() => {
                                rentEq(item.id)
                            }}>Wypożycz</div>
                        </p>
                    );
                }
            })
        </div>
    </div>
);
};

export default AvailableEquipmentPane;

```

Plik: RentedEquipmentPaneLarge.js

Opis działania:

Komponent RentedEquipmentPaneLarge jest odpowiedzialny za wyświetlanie sprzętu wypożyczonego przez użytkownika oraz umożliwienie jego zwrotu.

Przepływ danych:

```

import '../../../../../Style/EquipmentView/RentedEquipmentPaneLarge.css';
import { useState, React, useEffect } from "react";

const RentedEquipmentPaneLarge = ({ toggleSharedState, sharedState }) => {

```

```

const [rEquipment, setREquipment] = useState([]);
const [role, setRole] = useState('');
const [name, setName] = useState('');
const [surname, setSurname] = useState('');

const fetchRentedEquipmentPane = () => {
    fetch(` ${process.env.REACT_APP_SERVER_ADDRESS}/equipment/rented/` , {
        mode: 'cors',
        method: 'GET',
        headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
    }).then(response => response.json()).then(data => {
        if (data.detail) {
            console.log(data.detail);
            setREquipment([]);
        } else {
            setREquipment(data.eq);
        }
    });
};

useEffect(() => {
    fetchRentedEquipmentPane();
}, [sharedState]);

const returnEq = (id) => {

fetch(` ${process.env.REACT_APP_SERVER_ADDRESS}/equipment/return/${id}/` , {
    mode: 'cors',
    method: 'PATCH',
    headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
}).then(response => response.json()).then(data => {
    if (data.message === 'successfully deleted') {
        toggleSharedState();
        fetchRentedEquipmentPane();
    }
});
};

return (
    <div id="rentedEquipmentPaneLarge" className='paneShadow'>
        <div id="scrollBoxB" className='redScrollbar'>
            {rEquipment.map(item => {
                return (
                    <p key={item.id}>
                        <span className="bStyle listStyle">

```

```

                {item.descr}
            </span>
            {role === 'Coach' ? <div
className='rInfoBtn'>{name} {surname}</div> :
                    <div className="rEqButton" onClick={() => {
returnEq(item.id) }}>Zwróć</div>
                </p>
            );
        )})
    </div>
</div>
);
};

export default RentedEquipmentPaneLarge;

```

Plik: RentedEquipmentPane.js

Opis działania:

Komponent RentedEquipmentPane wyświetla listę sprzętu wypożyczonego przez użytkownika, ale bez opcji zwrotu.

Przepływ danych:

```

import '../../../../../Style/UserView/RentedEquipmentPane.css';
import { useState, React, useEffect } from "react";

const RentedEquipmentPane = () => {
    const [equipment, setEquipment] = useState([]);

    const fetchRentedEquipmentPane = () => {
        fetch(`process.env.REACT_APP_SERVER_ADDRESS}/equipment/rented/`, {
            mode: 'cors',
            method: 'GET',
            headers: { "Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}` },
        }).then(response => response.json()).then(data => {
            if (data.detail) {
                console.log(data.detail);
            } else {
                setEquipment(data.eq);
            }
        });
    };
}

```

```

useEffect(() => {
    fetchRentedEquipmentPane();
}, []);

return (
    <div id="rentedEquipmentPane" className='paneShadow'>
        <div id="scrollBox" className='redScrollbar'>
            {equipment.map(eq => (
                <p key={eq.id}>
                    <span className="bStyle listStyle">
                        {eq.descr}
                    </span>
                </p>
            ))}
        </div>
    </div>
);
};

export default RentedEquipmentPane;

```

Backend

Model User

Model użytkownika przechowuje dane takie jak ID, nazwa użytkownika, hasło, imię, nazwisko, email, rola (Player, Coach, Admin) oraz dodatkowe informacje jak numer telefonu i tokeny do resetowania hasła i weryfikacji emaila.

```

class User extends Model { }
User.init(
{
    id: {
        type: DataTypes.INTEGER,
        allowNull: false,
        primaryKey: true,
        autoIncrement: true
    },
    username: {
        type: DataTypes.STRING,
        allowNull: false
    },
    password: {
        type: DataTypes.STRING,
        allowNull: false
    },
}
,
```

```
first_name: {
    type: DataTypes.STRING,
    allowNull: false
},
last_name: {
    type: DataTypes.STRING,
    allowNull: false
},
email: {
    type: DataTypes.STRING,
    allowNull: false
},
role: {
    type: DataTypes.ENUM('Player', 'Coach', 'Admin'),
    allowNull: false
},
is_active: {
    type: DataTypes.BOOLEAN,
    allowNull: false,
    defaultValue: false
},
phone_number: {
    type: DataTypes.STRING,
    allowNull: true,
},
resetpasswordtoken: {
    type: DataTypes.STRING,
    allowNull: true
},
resetpasswordtokenexpirdate: {
    type: DataTypes.TIME,
    allowNull: true
},
emailverifytoken: {
    type: DataTypes.STRING,
    allowNull: true
},
emailverifytokenexpirdate: {
    type: DataTypes.TIME,
    allowNull: true
},
},
},
{
    sequelize,
    modelName: 'users'
}
);
```

Model Rented_equipments

Model Rented_equipments przechowuje informacje o wypożyczonym sprzęcie, w tym ID gracza (player_id) i ID sprzętu (equipment_id), a także datę wypożyczenia (rent_date).

```
class Rented_equipments extends Model { }
Rented_equipments.init(
{
    player_id: {
        type: DataTypes.INTEGER,
        allowNull: false,
        primaryKey: true
    },
    equipment_id: {
        type: DataTypes.STRING,
        allowNull: false,
        primaryKey: true
    },
    rent_date: {
        type: DataTypes.DATE,
        allowNull: true
    }
},
{
    sequelize,
    modelName: 'rented_equipments'
})
;
```

Model Equipment

Model Equipment przechowuje dane o sprzęcie, w tym jego ID, opis (descr), dostępność (available) i ID drużyny (team_id).

```
class Equipment extends Model { }
Equipment.init(
{
    id: {
        type: DataTypes.INTEGER,
        allowNull: false,
        primaryKey: true,
        autoIncrement: true
    },
    descr: {
        type: DataTypes.STRING,
        allowNull: true,
    },
    available: {
        type: DataTypes.INTEGER,
        allowNull: true
    },
    team_id: {
        type: DataTypes.INTEGER,
        allowNull: false,
    }
},
{
    sequelize,
    modelName: 'equipments'
})
;
```

Kontroler RentedEqController

Endpoint /equipment/rented/:

- Pobiera listę wypożyczonego sprzętu dla zalogowanego użytkownika.
- Najpierw znajduje użytkownika na podstawie nazwy użytkownika (`req.user.name`).
- Następnie znajduje wszystkie wypożyczenia związane z ID użytkownika.
- Pobiera szczegóły sprzętu na podstawie listy ID sprzętu i zwraca je w odpowiedzi.

```

const { Sequelize } = require('sequelize');
const { User, Rented_equipments, Equipment } = require('../Models/models');

const RentedEqController = async (req, res) => {
    try {
        console.log("rented endpoint -----");
        console.log(req.user.name);

        const user = await User.findOne({
            attributes: ['id'],
            where: { username: req.user.name }
        });

        if (!user) {
            res.status(404).json({ 'detail': 'No user found' });
        }

        const rented_equipment = await Rented_equipments.findAll({
            where: { player_id: user.dataValues.id }
        }).then(async rent_eqips => {

            if (rent_eqips.length !== 0) {
                const equip_id_list = [];

                for (let i = 0; i < rent_eqips.length; i++) {
                    equip_id_list.push(rent_eqips[i].equipment_id);
                }

                const equipment = await Equipment.findAll({
                    where: {
                        id: [
                            [Sequelize.Op.in]: equip_id_list
                        ]
                    }
                });

                if (equipment.length !== 0) {
                    res.json({ eq: equipment });
                } else {
                    res.status(400).json({ 'detail': 'Equipment find error' });
                }
            } else {
        
```

```
        return res.status(200).json({ 'detail': 'No rented equipment'
});}
    }
});

} catch (error) {
    res.json({ detail: error });
}
};

module.exports = RentedEqController;
```

Kontroler AvailableEqController

Endpoint /equipment/available/:

- Pobiera listę dostępnego sprzętu.
- Znajduje wszystkie elementy sprzętu i zwraca je w odpowiedzi.

```
const { Equipment } = require('../..../Models/models');

const AvailableEqController = async (req, res) => {
    try {
        const equipment = await Equipment.findAll();

        if (equipment.length !== 0) {
            res.json({ eq: equipment });
        } else {
            res.status(404).send({ "detail": `No equipment` });
        }
    } catch (error) {
        res.json({ detail: error });
    }
};

module.exports = AvailableEqController;
```

Podsumowanie

System zarządzania sprzętem sportowym składa się z kilku komponentów frontendowych, które komunikują się z backendem za pomocą API. Użytkownik może przeglądać dostępny sprzęt i wypożyczać go, a także przeglądać i zwracać wypożyczony sprzęt. Backend zarządza

danymi użytkowników, wypożyczeń oraz sprzętu, zapewniając odpowiednie endpointy do obsługi tych operacji.

Tutaj kolejna osoba

Jan Szczepański

-1.3.3.1 Wyświetlenie danych zawodników z perspektywy trenera

Frontend:

Plik: ManageTeam.js

```
import "../../Style/CoachView/ManageTeam.css";
import NavBar from "../NavBar.js";
import PlayerData from "./PlayerDataPane_CV.js";
import PlayerStats from "./PlayerStatisticsPane_CV.js";
import RentedEquipment from "./RentedEquipmentPane_CV.js";
import PlayerList from "./PlayerList_CV.js";
import { useState, useEffect } from "react";

const ManageTeam = () => {
  const [selectedPlayerId, setSelectedPlayerId] = useState(null);

  const [PageContent, setPageContent] = useState('')

  const handleSelectPlayer = (playerId) => {
    setSelectedPlayerId(playerId);
  };
}
```

```
return (
  <div id="boxManageTeam_CV">
    <div id="bar">
      <NavBar/>
    </div>
    <div id="playerListCV">
      <PlayerList onSelectPlayer={handleSelectPlayer} />
    </div>
    <div id="playerStatsCV">
      <div id="playerDataBoxCV">
        <PlayerData playerId={selectedPlayerId} />
      </div>
      <div id="playerStatsBoxCV">
        <PlayerStats playerId={selectedPlayerId} />
      </div>
      <div id="rentedEqBoxCV">
        <RentedEquipment playerId={selectedPlayerId} />
      </div>
    </div>
  </div>
);
```

Panel ManageTeam.js odpowiada za wyświetlenie listy graczy oraz ich statystyk. Stan selectedPlayerId i funkcja setSelectedPlayerId służą do tego, aby po naciśnięciu na gracza z listy, panele obok wyświetliły jego statystyki, na postawie id gracza. Do komponentu PlayerList_CV.js przekazywana jest funkcja handleSelectPlayer, która obsługuje

kliknięcie gracza, zmieniając stan selectedPlayerId. Ten stan jako parametr jest przekazywany do komponentów PlayerDataPane_CV oraz PlayerStatisticsPane_CV.

Plik: PlayerDataPane_CV.js

```
const PlayerDataPane_CV = ({ playerId }) => {
  const [playerData, setPlayerData] = useState("");
  const [playerDataSpecif, setPlayerDataSpecif] = useState("");

  const fetchPlayerData = async (playerId) => {
    console.log('select id', playerId)
    try {

      const response = await fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/data?playerId=${playerId}`, {
        mode: 'cors',
        method: 'GET',
        headers: {
          "Content-Type": "application/json",
          "authorization": `Bearer ${localStorage.getItem('access_token')}`
        }
      });
      const data = await response.json();
      if (data.detail) {
        console.log('Error:', data.detail);
      } else {
        console.log('Fetched data player:', data.user_info_id);
        setPlayerData(data.user_info_id) // Dodaj log
        setPlayerDataSpecif(data.user_info_id.player)
      }
    } catch (error) {
      console.error('Error fetching player info:', error);
    }
  };

  useEffect(() => {
    if (playerId) {
      fetchPlayerData(playerId);
    }
  }, [playerId]);
}
```

```

if (!playerId) {
    placeHolder1 = " ";
    placeHolder2 = " ";
    placeHolder3 = " ";
    placeHolder4 = " ";
    placeHolder5 = " ";
    placeHolder6 = " ";
} else {
    placeHolder1 = playerDataSpecif.date_of_birth;
    placeHolder2 = playerDataSpecif.height;
    placeHolder3 = playerDataSpecif.weight;
    placeHolder4 = playerDataSpecif.boot_size;
    placeHolder5 = playerData.email;
    placeHolder6 = playerData.phone_number;
}

return (
    <div id="playerDataPane_CV">
        <h2 id="whiteFont" class="whiteTextShadow">Dane Zawodnika</h2>
        <div id="dataBoxDataPane_CV">
            <div id="positionDeleteBox_CV">
                <p><span className="bStyle playerPositionStyle">BRAMKARZ</span></p>
                <button id="deleteButton_CV" onClick={handleDeleteClick}>Usuń</button>
            </div>
            <p><span className="bStyle">Data Urodzenia: {placeHolder1}</span></p>
            <p><span className="bStyle">Wzrost: {placeHolder2} cm</span></p>
            <p><span className="bStyle">Waga: {placeHolder3} Kg</span></p>
            <p><span className="bStyle">Rozmiar Buta: {placeHolder4}</span></p>
            <p><span className="bStyle">Mail: {placeHolder5}</span></p>
            <p><span className="bStyle">Nr Telefonu: {placeHolder6}</span></p>
        </div>
    </div>
);

```

Zmienna stany playerData przechowuje dane gracza pobrane z tabeli users w bazie danych, natomiast playerDataSpecif przechowuje dane z tabeli players w bazie danych. FetchPlayerData pobiera dane gracza na podstawie jego id przekazanego jako parametr do komponentu. Po zrenderowaniu się komponentu, wywołuję się useEffect hook, który wywołuje FetchPlayerData z parametrem id obecnie zaznaczonego gracza. Za każdym razem, kiedy playerId zmieni wartość, fetchPlayerData zostanie ponownie wywołane.

Backend

Nazwa pliku: UserViewDataController.js

```
const {User, Players, Player_stats,} = require('../Models/models')

const UserIdViewDataController = async(req,res) => {

    try {
        const { playerId } = req.query
        if (!playerId) {
            return res.status(400).json({ message: 'Brak id zawodnika' });
        }
        console.log("id zawodnika_data", playerId);

        const user_info_id = await User.findOne({
            attributes: ['id','email', 'phone_number'],
            include: [
                {model: Players,
                 attributes: ['date_of_birth', 'height', 'weight', 'boot_size']}
            ],
            where: {
                id: playerId
            }
        })
        console.log(user_info_id)
        if (user_info_id) {
            return res.json({ user_info_id });
        } else {
            return res.json({ detail: "No data found for the given player ID" });
        }

    } catch (error) {
        return res.json({ 'detail':error.message})
    }
}

module.exports = UserIdViewDataController
```

```
class Players extends Model {}  
Players.init(  
{  
    id:{  
        type: DataTypes.INTEGER,  
        allowNull:false,  
        primaryKey:true  
    },  
    date_of_birth:{  
        type: DataTypes.DATE,  
        allowNull:true  
    },  
    height:{  
        type:DataTypes.INTEGER,  
        allowNull:true  
    },  
    weight:{  
        type:DataTypes.INTEGER,  
        allowNull:true  
    },  
    boot_size:{  
        type:DataTypes.INTEGER,  
        allowNull:true  
    },  
    notes:{  
        type:DataTypes.TEXT,  
        allowNull:true  
    },  
    team_id:{  
        type:DataTypes.INTEGER,  
        allowNull:false,  
    }  
},  
{  
    sequelize,  
    modelName:'players'  
}  
)
```

```
class User extends Model {}  
User.init(  
{  
    id: {  
        type: DataTypes.INTEGER,  
        allowNull: false,  
        primaryKey: true,  
        autoIncrement:true  
    },  
    username: {  
        type: DataTypes.STRING,  
        allowNull: false  
    },  
    password: {  
        type: DataTypes.STRING,  
        allowNull: false  
    },  
    first_name: {  
        type: DataTypes.STRING,  
        allowNull: false  
    },  
    last_name: {  
        type: DataTypes.STRING,  
        allowNull: false  
    },  
    email: {  
        type: DataTypes.STRING,  
        allowNull: false  
    },  
    role: {  
        type: DataTypes.ENUM('Player', 'Coach', 'Admin'),  
        allowNull: false  
    },  
},
```

```
    is_active: {
      type: DataTypes.BOOLEAN,
      allowNull: false,
      defaultValue:false
    },
    phone_number: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    resetpasswordtoken: {
      type: DataTypes.STRING,
      allowNull: true
    },
    resetpasswordtokenexpirdate: {
      type: DataTypes.TIME,
      allowNull: true
    },
    emailverifytoken: {
      type: DataTypes.STRING,
      allowNull: true
    },
    emailverifytokenexpirdate: {
      type: DataTypes.TIME,
      allowNull: true
    },
  },
  {
    sequelize,
    modelName: 'users'
  }
}
```

Importujemy modele User oraz Player.

Następnie definiowana jest asynchroniczna funkcja “UserIdViewDataController”, której zadaniem jest obsługa żądań HTTP, przyjmuje ona argumenty:

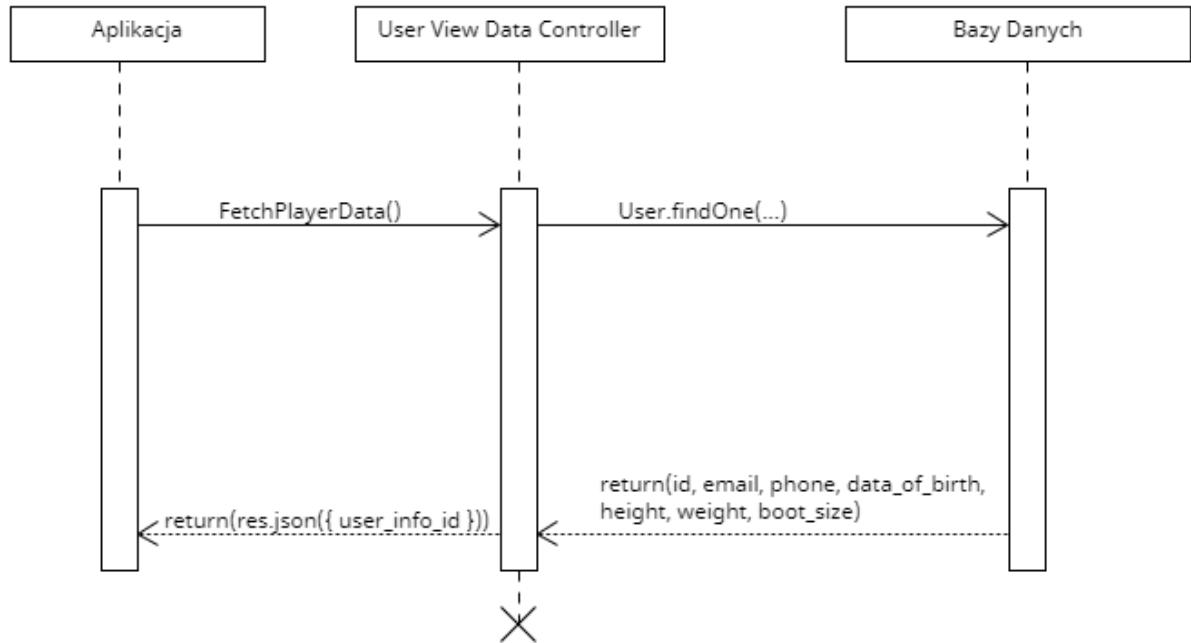
- req (request object)
- res (response object)

Rozpoczyna blok try-catch, aby obsłużyć potencjalne błędy. Destrukturyzuje playerId z parametrów zapytania żądania.

Używa metody findOne Sequelize, aby zapytać model User o użytkownika z podanym playerId. Wybiera atrybuty id, email i phone_number z modelu User iłącza powiązane dane z modelu Players, konkretnie atrybuty date_of_birth, height, weight i boot_size.

Klauzula where zapewnia filtrowanie zapytania według podanego playerId. Wynik jest przechowywany w user_info_id i logowany do konsoli.

Jeśli user_info_id zostało znalezione, zwraca dane jako odpowiedź JSON. Jeśli nie znaleziono danych, zwraca odpowiedź JSON z komunikatem, że nie znaleziono danych dla podanego identyfikatora gracza.



-1.3.3.2 Wyświetlenie i edycja statystyk zawodnika z perspektywy trenera

Plik: PlayerStatisticsPane_CV.js

```

import "../../Style/CoachView/PlayerStatisticsPane_CV.css";

const PlayerStatisticsPane_CV = ({ playerId }) => {
    const [visibility, setVisibility] = useState({ visibility: "hidden" });
    const [playerStats, setPlayerStats] = useState({
        goals: "",
        assists: "",
        red_cards: "",
        yellow_cards: "",
        attended_trainings: "",
        attended_matches: ""
    });

    const visibilityOn = () =>
    {
        window.scrollTo(0, 0);
        document.body.style.overflow = 'hidden';
        setVisibility({ visibility: "visible" });
    }

    const visibilityOff = () =>
    {
        document.body.style.overflow = 'auto';
        setVisibility({ visibility: "hidden" });
    }

const fetchPlayerStats = async (playerId) => {
    console.log('select id', playerId)
    try {

        const response = await fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/statistic?playerId=${playerId}`, {
            mode: 'cors',
            method: 'GET',
            headers: {
                "Content-Type": "application/json",
                "authorization": `Bearer ${localStorage.getItem('access_token')}`
            }
        });
        const data = await response.json();
        if (data.detail) {
            console.log('Error:', data.detail);
        } else {
            console.log('Fetched data:', data.user_stats_view.player.player_stats);
            setPlayerStats(data.user_stats_view.player.player_stats[0]) // Dodaj log

        }
    } catch (error) {
        console.error('Error fetching player statistics:', error);
    }
};

```

```

const fetchUpdatePlayerstats = async (playerId) => {
    console.log('Updating stats for id:', playerId);
    fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/coach/statistic/update?playerId=${playerId}`, {
        mode: 'cors',
        method: 'POST',
        headers: {
            "Content-Type": "application/json",
            "authorization": `Bearer ${localStorage.getItem('access_token')}`
        },
        body: JSON.stringify({
            goals: playerStats.goals,
            assists: playerStats.assists,
            red_cards: playerStats.red_cards,
            yellow_cards: playerStats.yellow_cards
        })
    })
    .then(response => response.json())
    .then(data => {
        if (data.detail) {
            console.log('Error:', data.detail);
        } else {
            console.log('Updated data:', data.body);
        }
    })
    .catch(error => {
        console.error('Error updating player statistics:', error);
    });
};

useEffect(() => {
    if (playerId) {
        fetchPlayerStats(playerId);
    }
}, [playerId]);

const handleInputChange = (e) => {
    const { name, value } = e.target;
    setPlayerStats((prevStats) => ({
        ...prevStats,
        [name]: value
    }));
};

const handleSubmit = () => {
    // Implement the logic to save the updated statistics
    fetchUpdatePlayerstats(playerId)
    visibilityOff();
};

```

```
if (!playerId) {
    placeHolder1 = " ";
    placeHolder2 = " ";
    placeHolder3 = " ";
    placeHolder4 = " ";
    placeHolder5 = " ";
    placeHolder6 = " ";
} else {
    placeHolder1 = playerStats.goals;
    placeHolder2 = playerStats.assists;
    placeHolder3 = playerStats.red_cards;
    placeHolder4 = playerStats.yellow_cards;
    placeHolder5 = playerStats.attended_trainings;
    placeHolder6 = playerStats.attended_matches;
}
//DELETE UP TO HERE AND REPLACE BELOW WITH THE RESPECTIVE VALUES

return (
    <div id="playerStatisticsPane_CV">
        <h2 id="whiteFont" class="whiteTextShadow">Statystyki Zawodnika</h2>
        <div id="dataBoxStatisticsPane_CV">
            <div id="positionDeleteBox_CV">
                <p><span className="bStyle playerPositionStyle" id="whiteFont">BRAMKARZ</span></p>
                <button id="deleteButton_CV" onClick={visibilityOn}>Edytuj</button>
            </div>
            <p><span className="bStyle">Gole: {placeHolder1}</span></p>
            <p><span className="bStyle">Asysty: {placeHolder2}</span></p>
            <p><span className="bStyle">Czerwone Kartki: {placeHolder3}</span></p>
            <p><span className="bStyle">Żółte Kartki: {placeHolder4}</span></p>
            <p><span className="bStyle">Obecność Na Treningach: {placeHolder5}</span></p>
            <p><span className="bStyle">Obecność Na Meczach: {placeHolder6}</span></p>
        </div>
    </div>
```

```
<div id="survBox" style={visibility}>
  <div id="survWindow">
    <div id="survTitle">Zmień Statystyki Gracza</div>
    <div id="survInputs">
      <div className="survOption">
        Gole:
        <input
          type="number"
          name="goals"
          value={playerStats.goals}
          onChange={handleInputChange}
          min="0"
        />
      </div>
      <div className="survOption">
        Asysty:
        <input
          type="number"
          name="assists"
          value={playerStats.assists}
          onChange={handleInputChange}
          min="0"
        />
      </div>
      <div className="survOption">
        Żółte kartki:
        <input
          type="number"
          name="yellowCards"
          value={playerStats.yellow_cards}
          onChange={handleInputChange}
          min="0"
        />
      </div>
    </div>
  </div>
</div>
```

```
        <div className="survOption">
          Czerwone kartki:
          <input
            type="number"
            name="redCards"
            value={playerStats.red_cards}
            onChange={handleInputChange}
            min="0"
          />
        </div>
      <div id="survButtons">
        <div className="survBtn" id="exitBtn" onClick={visibilityOff}>X</div>
        <div className="survBtn" id="okBtn" onClick={handleSubmit}>OK</div>
      </div>
    </div>
  );
};
```

playerStats to stan przechowujący statystyki gracza o id równym przekazanemu do komponentu parametrowi playerId.

FetchPlayerStats fetchuje statystyki gracza na podstawie playerId i przypisuje je do playerStats

W komponencie występuje możliwość edycji statystyk przez naciśnięcie przycisku EDYTUJ. Wywołuję się wtedy funkcja visibilityOn, która pokazuje pop-up, gdzie trener może modyfikować dane.

Samo modyfikowanie odbywa się w funkcji handleInputChange.

Po zatwierdzeniu zmian wywołuje się funkcja handleSubmit, która ukrywa popup, i wywołuje funkcję UpdateFetchPlayerStats.

UpdateFetchPlayerStats przez POST Api aktualizuje bazę danych o zmienione statystyki użytkownika

Backend:

Nazwa pliku: UserViewStatisticController.js

```
const {User, Players, Player_stats,} = require('../Models/models')

const { Op } = require('sequelize');

const GetPlayerStatistics = async (req,res) => {

    try {

        const { playerId } = req.query
        if (!playerId) {
            return res.status(400).json({ message: 'Brak id zawodnika' });
        }
        console.log("id zawodnika", playerId);
        const user_stats_view = await User.findOne({
            attributes: ['id'],
            where: {id: playerId},
            include:[{
                model: Players,
                attributes:['id'],
                include: [{
                    model: Player_stats,
                    attributes:['goals','assists','yellow_cards','red_cards', 'attended_trainings', 'attended_matches']
                }]
            }],
        });
        console.log("Fetched user stats view:", user_stats_view.player.player_stats[0]);
        if (user_stats_view) {
            return res.json({ user_stats_view });
        } else {
            return res.json({ detail: "No stats found for the given player ID" });
        }
    } catch (error) {
        return res.json({'detail':error.message})
    }
};

module.exports = GetPlayerStatistics
```

```
class Player_stats extends Model {}  
Player_stats.init(  
{  
    player_id:{  
        type: DataTypes.INTEGER,  
        allowNull:false,  
        primaryKey:true  
    },  
    season_id:{  
        type: DataTypes.DATE,  
        allowNull:false,  
        primaryKey:true  
    },  
    goals:{  
        type:DataTypes.INTEGER,  
        allowNull:true,  
        defaultValue: 0  
    },  
    assists:{  
        type:DataTypes.INTEGER,  
        allowNull:true,  
        defaultValue: 0  
    },  
    yellow_cards:{  
        type:DataTypes.INTEGER,  
        allowNull:true,  
        defaultValue: 0  
    },  
    red_cards:{  
        type:DataTypes.INTEGER,  
        allowNull:true,  
        defaultValue: 0  
    },  
    attended_matches:{  
        type:DataTypes.INTEGER,  
        allowNull:true,  
        defaultValue: 0  
    },  
},  
}
```

```
    unattended_matches:{  
      type:DataTypes.INTEGER,  
      allowNull:true,  
      defaultValue: 0  
    },  
    attended_trainings:{  
      type:DataTypes.INTEGER,  
      allowNull:true,  
      defaultValue: 0  
    },  
    unattended_trainings:{  
      type:DataTypes.INTEGER,  
      allowNull:true,  
      defaultValue: 0  
    }  
  },  
  {  
    sequelize,  
    modelName:'player_stats'  
  }  
)
```

Importujemy modele User, Player oraz Player_stats, kod dla modeli user oraz player znajduje się w funkcjonalności o 1 wyżej. Importuje również operator Op z Sequelize, który jest używany do tworzenia bardziej złożonych zapytań.

Definiuje asynchroniczną funkcję kontrolera o nazwie GetPlayerStatistics, która przyjmuje dwa argumenty: req (obiekt żądania) i res (obiekt odpowiedzi).

Rozpoczyna blok try-catch, aby obsłużyć potencjalne błędy. Destrukturyzuje playerId z parametrów zapytania żądania. Sprawdza, czy playerId zostało podane. Jeśli nie, zwraca status 400 z komunikatem o błędzie wskazującym, że identyfikator gracza jest brakujący. W przeciwnym razie, loguje identyfikator gracza do konsoli.

Używa metody findOne Sequelize, aby zapytać model User o użytkownika z podanym playerId. Wybiera atrybut id z modelu User i włącza powiązane dane z modelu Players, a następnie z modelu Player_stats, wybierając atrybuty goals, assists, yellow_cards, red_cards, attended_trainings i attended_matches. Wynik jest przechowywany w user_stats_view i logowany do konsoli.

Jeśli user_stats_view zostało znalezione, zwraca dane jako odpowiedź JSON. Jeśli nie znaleziono danych, zwraca odpowiedź JSON z komunikatem, że nie znaleziono statystyk dla podanego identyfikatora gracza.

Nazwa pliku: UserUpdatePlayerStatistics.js

```
const { User, Players, Player_stats } = require('../Models/models'); // Upewnij się, że importujesz Player_stats

const UserUpdatePlayerStatistics = async (req, res) => {
  try {
    const { playerId } = req.query;
    const { goals, assists, red_cards, yellow_cards } = req.body;

    console.log("Updating stats for player ID:", playerId);

    const updatedStats = await Player_stats.update(
      {
        goals,
        assists,
        red_cards,
        yellow_cards
      },
      {
        where: { player_id: playerId },
        returning: true,
        plain: true
      }
    );

    return res.json({ updatedStats });
  } catch (error) {
    console.error("Error updating player statistics:", error);
    return res.json({ detail: error });
  }
};

module.exports = UserUpdatePlayerStatistics
```

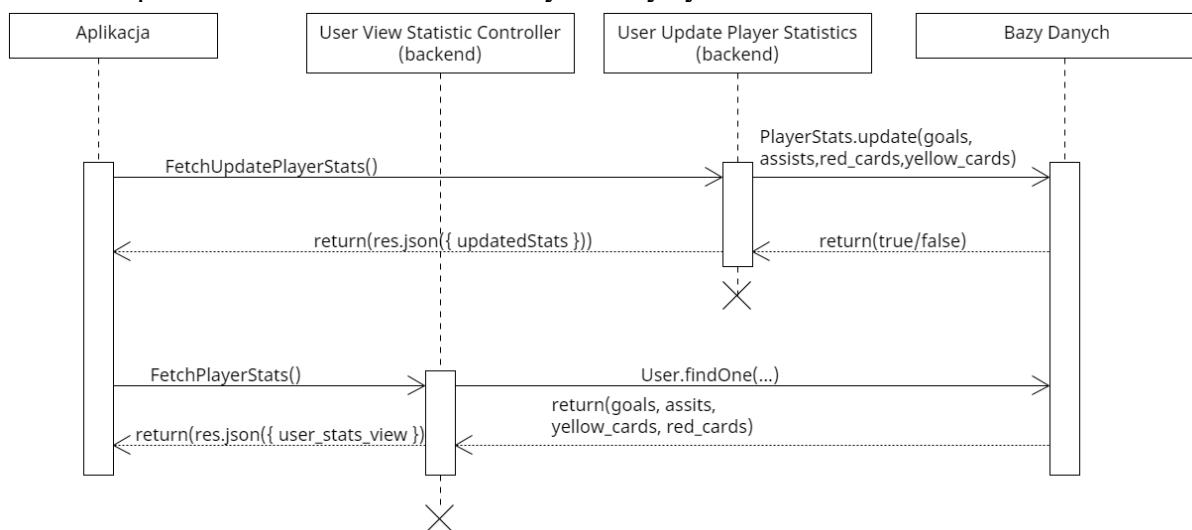
Importujemy model Player_Stats.

Definiuje asynchroniczną funkcję kontrolera o nazwie UserUpdatePlayerStatistics, która przyjmuje dwa argumenty: req (request object) i res (response object).

Destrukturyzuje playerId z parametrów zapytania żądania (req.query) oraz goals, assists, red_cards, yellow_cards z ciała żądania (req.body).

Używa metody update Sequelize, aby zaktualizować model Player_stats. Przekazuje nowe wartości goals, assists, red_cards i yellow_cards. Warunek where określa, że aktualizacja powinna dotyczyć rekordu, którego player_id jest równy playerId. Opcje returning: true i plain: true powodują, że zwracany jest zaktualizowany rekord.

Zwraca odpowiedź JSON z zaktualizowanymi statystykami.



Stygar Dawid

1.2.3.1 Pokazanie informacji o zawodniku

Frontend:

Nazwa pliku: PlayerDataPane.js

```
1 import '../../../../../Style/UserView/PlayerDataPane.css'
2 import { useState, React, useEffect } from "react"
3
4 const PlayerDataPane = () => {
5
6     const [dateOfBirth, setDateOfBirth] = useState('')
7     const [pHeight, setPlayerHeight] = useState('')
8     const [pWeight, setPlayerWeight] = useState('')
9     const [pShoeSize, setPShoeSize] = useState('')
10    const [mail, setMail] = useState('')
11    const [phoneNumber, setPhoneNumber] = useState('')
```

Plik ten rozpoczyna się od importu arkusza stylów oraz następujących trzech react'owych hook'ów: "useState", "useEffect" i "React".

Następnie definiowana jest nazwa komponentu, zaraz za nią definiowane zostaje sześć stanów do przechowywania danych użytkownika takich jak: data urodzenia, wzrost, waga, rozmiar buta, e-mail i numer telefonu.

```

13     useEffect(() => {
14       if(localStorage.getItem('user_info.date_of_birth') !== null){
15         setDateOfBirth(localStorage.getItem('user_info.date_of_birth'))
16         setPlayerHeight(localStorage.getItem('user_info.height'))
17         setPlayerWeight(localStorage.getItem('user_info.weight'))
18         setShoeSize(localStorage.getItem('user_info.shoe_size'))
19         setMail(localStorage.getItem('user_info.mail'))
20         setPhoneNumber(localStorage.getItem('user_info.phone_number'))
21     }
22   } else{
23     fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/user/info/`, {
24       mode: 'cors',
25       method: 'GET',
26       headers: {"Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}`},
27     }).then(response=>response.json()).then(data=>{
28       if(data.detail){
29         console.log(data.detail)
30       }else{
31         console.log(data.user_cred)
32         setDateOfBirth(data.user_cred.player.date_of_birth)
33         setPlayerHeight(data.user_cred.player.height)
34         setPlayerWeight(data.user_cred.player.wieght)
35         setShoeSize(data.user_cred.player.boot_size)
36         setMail(data.user_cred.email)
37         setPhoneNumber(data.user_cred.phone_number)
38
39         localStorage.setItem('user_info.date_of_birth', data.user_cred.player.date_of_birth)
40         localStorage.setItem('user_info.height', data.user_cred.player.height)
41         localStorage.setItem('user_info.weight', data.user_cred.player.wieght)
42         localStorage.setItem('user_info.shoe_size', data.user_cred.player.boot_size)
43         localStorage.setItem('user_info.mail', data.user_cred.email)
44         localStorage.setItem('user_info.phone_number', data.user_cred.phone_number)
45     }
46   })
47 }
48 )
49 })
50

```

Hook useEffect jest używany do pobrania danych użytkownika. Na początku sprawdza, czy dane są zapisane w localStorage. Jeśli są, ustawia je w stanach. Jeśli nie, wykonuje zapytanie do serwera w celu pobrania danych użytkownika i zapisuje je w localStorage, a następnie ustawia w stanach.

```

51   return (
52     <div id="playerDataPane" className='paneShadow'>
53       <div id="playerData">
54         <p><span className="bStyle">Data Urodzenia: </span>{dateOfBirth}</p>
55         <p><span className="bStyle">Wzrost: </span>{pHeight}</p>
56         <p><span className="bStyle">Waga: </span>{pWeight}</p>
57         <p><span className="bStyle">Rozmiar Buta: </span>{pShoeSize}</p>
58         <p><span className="bStyle">Mail: </span>{mail}</p>
59         <p><span className="bStyle">Nr Telefonu: </span>{phoneNumber}</p>
60       </div>
61     </div>
62   );
63 }
64
65 export default PlayerDataPane

```

W kolejnej części zostaje zwrocona struktura HTML, w której wypisane zostają pobrane wcześniej dane.

Na sam koniec, komponent “PlayerDataPane” jest eksportowany”.

Backend:

Nazwa pliku: UserInfoController.js

```
1  const {User, Players} = require('../Models/models')
2
3  const UserInfoController = async (req, res) => {
4
5      try {
6
7          const User_cred = await User.findOne({
8              attributes: ['email', 'phone_number'],
9              include: [
10                  {
11                      model: Players,
12                      attributes: ['date_of_birth', 'height', 'weight', 'boot_size']
13                  },
14                  {
15                      where: {
16                          username: req.user.name
17                      }
18                  }
19              ]
20          })
21
22          if(user_cred.length!=0){
23              return res.json({user_cred:user_cred})
24          }
25      } catch (error) {
26          return res.json({'detail':error})
27      }
28
29  module.exports = UserInfoController
```

Na początku, importowane są modele ”User” oraz ”Player”, które są zgodne ze strukturą bazy danych. Przedstawione one są w następujący sposób:

-”User”

```
5  class User extends Model {}
6  User.init(
7  {
8      id: {
9          type: DataTypes.INTEGER,
10         allowNull: false,
11         primaryKey: true,
12         autoIncrement:true
13     },
14     username: {
15         type: DataTypes.STRING,
16         allowNull: false
17     },
18     password: {
19         type: DataTypes.STRING,
20         allowNull: false
21     },
22     first_name: {
23         type: DataTypes.STRING,
24         allowNull: false
25     },
26     last_name: {
27         type: DataTypes.STRING,
28         allowNull: false
29     },
30     email: {
31         type: DataTypes.STRING,
32         allowNull: false
33     },
```

```
34     role: {
35         type: DataTypes.ENUM('Player', 'Coach', 'Admin'),
36         allowNull: false
37     },
38     is_active: {
39         type: DataTypes.BOOLEAN,
40         allowNull: false,
41         defaultValue:false
42     },
43     phone_number: {
44         type: DataTypes.STRING,
45         allowNull: true,
46     },
47     resetpasswordtoken: {
48         type: DataTypes.STRING,
49         allowNull: true
50     },
51     resetpasswordtokenexpirdate: {
52         type: DataTypes.TIME,
53         allowNull: true
54     },
55     emailverifytoken: {
56         type: DataTypes.STRING,
57         allowNull: true
58     },
59     emailverifytokenexpirdate: {
60         type: DataTypes.TIME,
61         allowNull: true
62     },
63 },
64 },
65 },
66 {
67     sequelize,
68     modelName: 'users'
69 }
70 )
```

-“Player”

```

208 class Players extends Model {}
209 Players.init(
210 {
211   id:{
212     type: DataTypes.INTEGER,
213     allowNull:false,
214     primaryKey:true
215   },
216   date_of_birth:{
217     type: DataTypes.DATE,
218     allowNull:true
219   },
220   height:{
221     type:DataTypes.INTEGER,
222     allowNull:true
223   },
224   weight:{
225     type:DataTypes.INTEGER,
226     allowNull:true
227   },
228   boot_size:{
229     type:DataTypes.INTEGER,
230     allowNull:true
231   },
232   notes:[
233     type:DataTypes.TEXT,
234     allowNull:true
235   ],
236   team_id:{
237     type:DataTypes.INTEGER,
238     allowNull:false,
239   }
240 },
241 {
242   sequelize,
243   modelName:'players'
244 }
245 ]

```

Następnie definiowana jest asynchroniczna funkcja “UserInfoController”, której zadaniem jest obsługa żądań HTTP, przyjmuje ona dwa argumenty:

- req (obiekt żądania)
- res (obiekt odpowiedzi)

W bloku try-catch

Wykonywane jest zapytanie do bazy danych, aby znaleźć użytkownika (User) o nazwie użytkownika req.user.name. Zapytanie zwraca wybrane atrybuty użytkownika (email i phone_number) oraz informacje o powiązanym modelu Players (takie jak date_of_birth,

height, weight i boot_size). Używa await, aby poczekać na zakończenie operacji asynchronicznej.

Jeśli znaleziono użytkownika to zwracane są dane w formacie JSON, a jeśli wystąpił błąd to zwracana jest odpowiedź zawierającą szczegółowy błąd, też w formacie JSON.

Na końcu kontroler “UserInfoController” zostaje eksportowany.

1.2.3.2 Pokazanie statystyk zawodnika

Frontend:

Nazwa pliku: PlayerStatisticsPane.js

```
1 import '../../../../../Style/UserView/PlayerStatisticsPane.css'
2 import { useState, React, useEffect} from "react"
3
4 const PlayerStatisticsPane = () => {
5     const [goalsNumber, setGoals] = useState('')
6     const [assistsNumber, setAssists] = useState('')
7     const [yellowCards, setYellCards] = useState('')
8     const [redCards, setRedCards] = useState('')
9     const [attendanceAtTraining, setAttendanceAtTraining] = useState('')
10    const [attendanceAtMatches, setAttendanceAtMatches] = useState('')
```

Plik ten rozpoczyna się od importu odpowiedniego arkusza stylów oraz importu następujących trzech react'owych hook'ów: “useState”, “useEffect” i “React”.

Definiowane zostaje sześć stanów, które będą służyć do przechowywania statystyk użytkownika, takich jak: liczba goli, liczba asyst, żółtych kartek, czerwonych kartek, obecność na treningach i obecność na meczach.

```

12     useEffect(() => {
13       if(localStorage.getItem('user_stats.goals') !== null){
14         setGoals(localStorage.getItem('user_stats.goals'))
15         setAssists(localStorage.getItem('user_stats.assists'))
16         setYellowCards(localStorage.getItem('user_stats.yellow_cards'))
17         setRedCards(localStorage.getItem('user_stats.red_cards'))
18         setAttendanceAtTraining(localStorage.getItem('user_stats.attendance_at_training'))
19         setAttendanceAtMatches(localStorage.getItem('user_stats.attendance_at_matches'))
20     }
21   else{
22     fetch(`${process.env.REACT_APP_SERVER_ADDRESS}/user/statistics`, {
23       mode: 'cors',
24       method: 'GET',
25       headers: {"Content-Type": "application/json", "authorization": `Bearer ${localStorage.getItem('access_token')}`},
26     }).then(response=>response.json()).then(data=>{
27       if(data.detail){
28         console.log(data.detail)
29       }else{
30         //console.log(data.user_stats.player.player_stats[0].yellow_cards)
31         setGoals(data.user_stats.player.player_stats[0].goals)
32         setAssists(data.user_stats.player.player_stats[0].assists)
33         setYellowCards(data.user_stats.player.player_stats[0].yellow_cards)
34         setRedCards(data.user_stats.player.player_stats[0].red_cards)
35         setAttendanceAtTraining(data.user_stats.player.player_stats[0].attended_trainings)
36         setAttendanceAtMatches(data.user_stats.player.player_stats[0].attended_matches)
37
38         localStorage.setItem('user_stats.goals', data.user_stats.player.player_stats[0].goals)
39         localStorage.setItem('user_stats.assists', data.user_stats.player.player_stats[0].assists)
40         localStorage.setItem('user_stats.yellow_cards', data.user_stats.player.player_stats[0].yellow_cards)
41         localStorage.setItem('user_stats.red_cards', data.user.stats.player.player_stats[0].red_cards)
42         localStorage.setItem('user_stats.attendance_at_training', data.user_stats.player.player_stats[0].attended_trainings)
43         localStorage.setItem('user_stats.attendance_at_matches', data.user_stats.player.player_stats[0].attended_matches)
44       }
45     })
46   }
47 }
48 )

```

Hook useEffect jest używany do pobrania statystyk użytkownika. Na początku sprawdza, czy dane są zapisane w localStorage. Jeśli są, ustawia je w stanach. Jeśli nie, wykonuje zapytanie do serwera w celu pobrania statystyk użytkownika i zapisuje je w localStorage, a następnie ustawia w stanach.

```

49
50   return (
51     <div id="playerStatisticsPane" className='paneShadow'>
52       <p><span className="bStyle">Gole: </span>{goalsNumber}</p>
53       <p><span className="bStyle">Asysty: </span>{assistsNumber}</p>
54       <p><span className="bStyle">Żółte Kartki: </span>{yellowCards}</p>
55       <p><span className="bStyle">Czerwone Kartki: </span>{redCards}</p>
56       <p><span className="bStyle">Obecność Na Treningach: </span>{attendanceAtTraining}</p>
57       <p><span className="bStyle">Obecność Na Meczach: </span>{attendanceAtMatches}</p>
58     </div>
59   );
60 }
61
62 export default PlayerStatisticsPane

```

Na końcu zwracana jest struktura HTML, która wyświetla pobrane powyżej statystyki użytkownika, oraz komponent “PlayerStatisticsPane” jest eksportowany.

Backend:

Nazwa pliku: UserStatisticsController.js

```
1  const {User, Players, Player_stats} = require('../Models/models')
2
3  const UserStatisticsController = async (req, res) => {
4
5      try {
6
7          const User_stats = await User.findall({
8              attributes: ['id'],
9              include:[{
10                  model: Players,
11                  attributes:['id'],
12                  include: [{{
13                      model: Player_stats,
14                      attributes:['goals','assists','yellow_cards','red_cards', 'attended_trainings', 'attended_matches']
15                  }]}
16
17              }],
18              where: {
19                  username: req.user.name
20              }
21
22      })
23
24      if(User_stats.length != 0){
25          return res.json({user_stats: User_stats})
26      }
27
28  } catch (error) {
29      return res.json({'detail':error})
30  }
31
32
33
34
35  module.exports = UserStatisticsController
```

Analogicznie jak we wcześniejszej funkcjonalności importowane są modele "User" oraz "Players", lecz dodatkowo importujemy model "Player_stats", który prezentuje się następująco:

```
307   class Player_stats extends Model {}  
308   Player_stats.init(  
309   {  
310     player_id:{  
311       type: DataTypes.INTEGER,  
312       allowNull:false,  
313       primaryKey:true  
314     },  
315     season_id:{  
316       type: DataTypes.DATE,  
317       allowNull:false,  
318       primaryKey:true  
319     },  
320     goals:{  
321       type:DataTypes.INTEGER,  
322       allowNull:true,  
323       defaultValue: 0  
324     },  
325     assists:{  
326       type:DataTypes.INTEGER,  
327       allowNull:true,  
328       defaultValue: 0  
329     },  
330     yellow_cards:{  
331       type:DataTypes.INTEGER,  
332       allowNull:true,  
333       defaultValue: 0  
334     },  
335     red_cards:{  
336       type:DataTypes.INTEGER,  
337       allowNull:true,  
338       defaultValue: 0  
339     },
```

```
340     attended_matches:{  
341         type:DataTypes.INTEGER,  
342         allowNull:true,  
343         defaultValue: 0  
344     },  
345     unattended_matches:{  
346         type:DataTypes.INTEGER,  
347         allowNull:true,  
348         defaultValue: 0  
349     },  
350     attended_trainings:{  
351         type:DataTypes.INTEGER,  
352         allowNull:true,  
353         defaultValue: 0  
354     },  
355     unattended_trainings:{  
356         type:DataTypes.INTEGER,  
357         allowNull:true,  
358         defaultValue: 0  
359     }  
360  
361 },  
362 {  
363     sequelize,  
364     modelName:'player_stats'  
365 }  
366  
367 )
```

Definiowana jest asynchroniczna funkcja “UserStatisticsController”, której zadaniem jest obsługa żądań HTTP, przyjmuje ona dwa argumenty:

- req (obiekt żądania)
- res (obiekt odpowiedzi)

W bloku try-catch wykonywane jest zapytanie do bazy danych, aby znaleźć użytkownika (User) o nazwie użytkownika req.user.name. Zapytanie zwraca tylko atrybut id użytkownika. Następnie włącza powiązany model Players, zwracając tylko atrybut id gracza, i dodatkowo włącza powiązany model Player_stats, zwracając atrybuty: goals, assists, yellow_cards, red_cards, attended_trainings i attended_matches. Używa await, aby poczekać na zakończenie operacji asynchronicznej.

Jeśli znaleziono statystyki gracza, to zwracana zostaje odpowiedź w formacie JSON zawierającą dane statystyk użytkownika.

Jeśli wystąpił błąd, to zwracana zostaje odpowiedź zawierającą szczegóły błędu w formacie JSON.

Na końcu kontroler “UserStatisticsController” zostaje eksportowany.

```
const UserInfoController = require('../Controllers/UserControllers/UserInfoController')
const UserStatisticsController = require('../Controllers/UserControllers/UserStatisticsController')
```

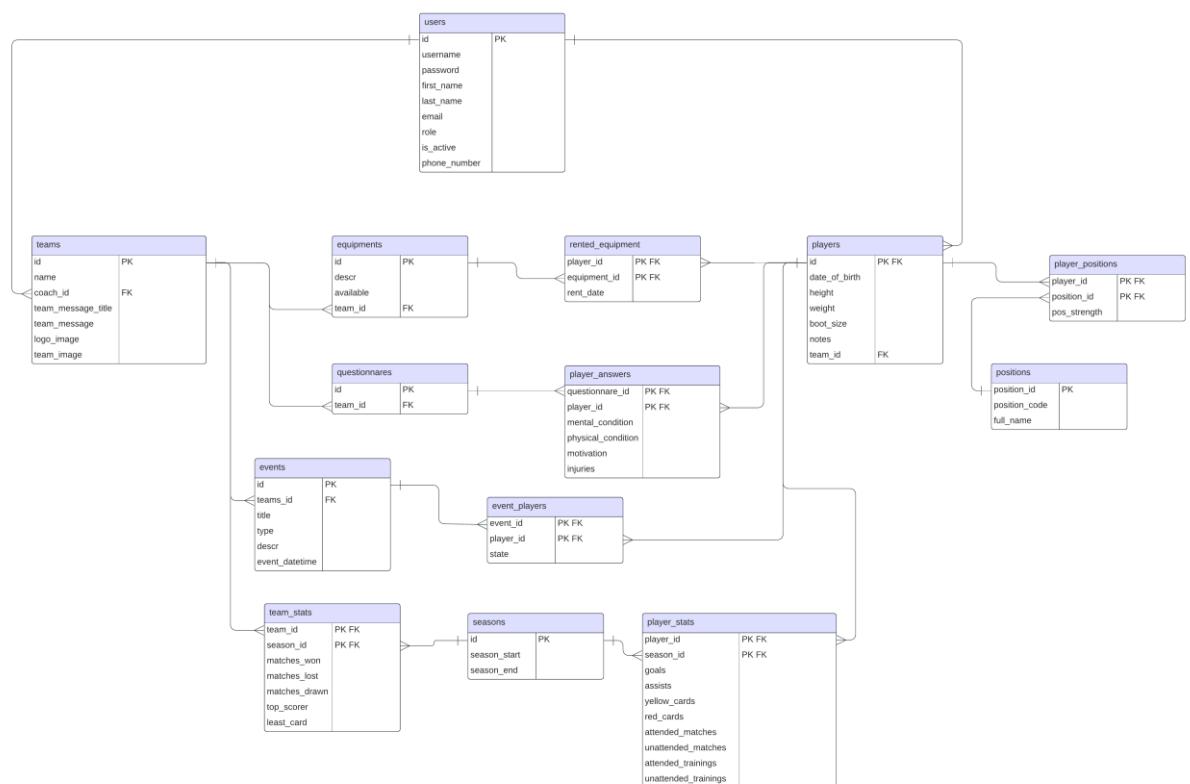
Oba kontrolery importowane są w pliku “UserRouter.js”, który importowany jest w pliku “index.js” w poniższej linii.

```
const UserRouter = require('./Routes/UserRouter')
```

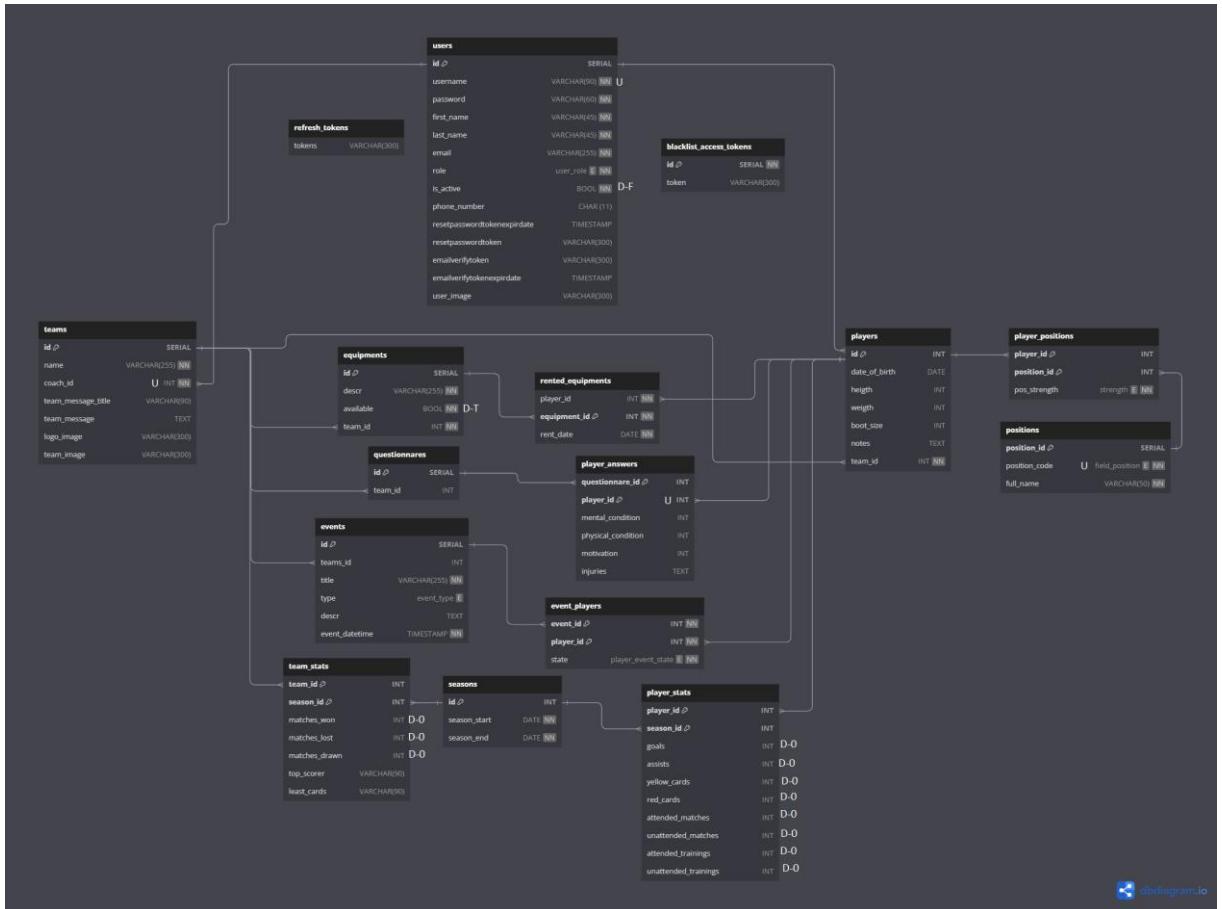
• Dokumentacja dotycząca bazy danych

Część tej dokumentacji przeznaczona jest na przedstawienie i opis diagramów opisujących działanie naszego systemu. Sporzązone diagramy przedstawiają m.in. zależności pomiędzy encjami, hierarchię funkcji w systemie, przepływy danych, przypadki użycia elementów aplikacji przez zewnętrzne podmioty, wykorzystanie odpowiednich funkcji w systemie w zależności od czasu oraz zależności pomiędzy klasami.

.1. Diagram encji



.2. Diagram bazodanowy



NN - Not null, U – unique, PK – primary key, FK – foreign key,

D – default: T- true, F-false, 0 – numerycznie 0

.3. Opis relacji

1 – one, N - many

-**users (1) – id**: Użytkownik może być trenerem jednej drużyny → **teams (1) – coach_id (UNIQUE)**.

-**users (1) – id**: Użytkownik może być jednym graczem → **players (1) – id (PK)**.

-**players (1) – id**: Gracz może umieć grać na wielu pozycjach trenerem → **player_positions (N) – player_id**.

-**positions (1) – position_id**: Na jednej pozycji może umieć grać wielu graczy → **player_positions (N) - position_id**

-**teams (1) – id**: W jednej drużynie może być wielu graczy → **players (N) – team_id**.

-**teams (1) – id**: Drużyna może posiadać wiele sprzętu do wypożyczenia → **equipments (N) - team_id**

-**equipments (1) – id**: Sprzęt może być wypożyczony przez jedną osobę na raz → **rented_equipments (1) – equipment_id (PK)**

-**players (1) – id**: Gracz może wypożyczyć wiele sprzętu → **rented_equipments (N) - player_id**

-**teams (1) – id**: Drużyna może posiadać wiele kwestionariuszy → **questionnaires(N) - team_id**

-**questionnaires (1) – id**: W kwestionariuszu może wziąć udział wielu graczy → **player_answers(N) – questionnaire_id**

-**players (1) – id**: Gracz może wziąć udział w wielu kwestionariuszach → **player_answers(N) - player_id**

-**teams (1) – id**: Drużyna może posiadać wiele wydarzeń → **events(N) - teams_id**

-**events (1) – id**: W wydarzeniu może wziąć udział wielu graczy → **event_players(N) – event_id**

-**players (1) – id**: Gracz może wziąć udział w wielu wydarzeniach → **player_events(N) - player_id**

-**teams(1) – id**: Drużyna może posiadać wiele statystyk w zależności od sezonu → **team_stats(N) - team_id**

-**seasons (1) – id**: W trakcie sezonu statystyki może posiadać wiele drużyn → **team_stats(N) - season_id**

-**seasons (1) – id**: W trakcie sezonu statystyki może posiadać wielu graczy → **player_stats(N) – season_id**

-**players (1) – id**: Gracz może posiadać wiele statystyk w zależności od sezonu → **player_stats(N) - player_id**

.4. Szczegóły tabel

Typy danych:

- INT - wartość liczbową całkowitą
- VARCHAR(n) - dynamiczny łańcuch znaków o długości maksymalnej n
- BOOL - wartość logiczna
- CHAR(n) - łańcuch znaków o stałej liczbie znaków
- TIMESTAMP – uniwersalna wartość daty i czasu
- TEXT – dynamiczny łańcuch znaków o nieokreślonym rozmiarze
- SERIAL - wartość liczbową całkowitą automatycznie przypisującą nowemu rekordowi wartość o 1 większą
- user_role – enum określający rolę użytkownika, może przybierać wartości “Player”, “Coach”, “Admin”
- event_type – enum określający rodzaj wydarzenia, może przybierać wartości “Trening”, “Mecz wyjazdowy”, “Mecz domowy”
- strength - enum określający jak pewnie gracz czuje się na pozycji, może przybierać wartości “Preferowana”, “Grywalana”
- player_event_state - enum określający obecność gracza na wydarzeniu, jak i jego potwierdzenie że będzie obecny, może przyjmować wartości “Obecny”, “Nieobecny”, “Niezadeklarowany”, “Będzie”, “Nie będzie”
- field_position - enum określający kody pozycji w polu, może przybierać wartości “BR”, “LO”, “ŚO”, “PO”, “CLS”, “CPS”, “LP”, “ŚP”, “PP”, “ŚPD”, “ŚPO”, “LS”, “PS”, “ŚN”, “N”

users:

id: Unikalny identyfikator użytkownika.

username: Unikalny login użytkownika.

password: Hasło użytkownika.

first_name: Imię użytkownika.

last_name: Nazwisko użytkownika.

email: Adres e-mail użytkownika.

role: Rola użytkownika (user_role)

is_active: Status aktywności użytkownika (default false).

phone_number: Numer telefonu użytkownika.

resetpasswordtokenexpiredate: Data wygaśnięcia tokenu resetu hasła.

resetpasswordtoken: Token resetu hasła.

emailverifytoken: Token weryfikacji e-maila.

emailverifytokenexpiredate: Data wygaśnięcia tokenu weryfikacji e-maila.

user_image: Ścieżka do obrazu użytkownika.

refresh_tokens:

tokens: Token odświeżania.

blacklist_access_tokens:

id: Unikalny identyfikator tokenu.

token: Token.

teams:

id: Unikalny identyfikator drużyny.

name: Nazwa drużyny.

coach_id: Unikalny identyfikator trenera drużyny.

team_message_title: Tytuł wiadomości drużynowej.

team_message: Treść wiadomości drużynowej.

logo_image: Ścieżka do logo drużyny.

team_image: Ścieżka do obrazu drużyny.

players:

id: Unikalny identyfikator zawodnika.

date_of_birth: Data urodzenia zawodnika.

height: Wzrost zawodnika.

weight: Waga zawodnika.

boot_size: Rozmiar buta zawodnika.

notes: Notatki dotyczące zawodnika.

team_id: Identyfikator drużyny, do której należy zawodnik.

player_positions:

player_id: Identyfikator zawodnika.

position_id: Identyfikator pozycji zawodnika.

pos_strength: Siła zawodnika na danej pozycji. (strength)

positions:

id: Unikalny identyfikator pozycji.

position_code: Unikalny kod pozycji. (field_positions)

full_name: Pełna nazwa pozycji.

equipments:

id: Unikalny identyfikator sprzętu.

descr: Opis sprzętu.

available: Dostępność sprzętu. (default true)

team_id: Identyfikator drużyny, do której należy sprzęt.

rented_equipments:

player_id: Identyfikator zawodnika wypożyczającego sprzęt.

equipment_id: Identyfikator wypożyczonego sprzętu.

rent_date: Data wypożyczenia sprzętu.

questionnaires:

id: Unikalny identyfikator ankiety.

team_id: Identyfikator drużyny, dla której jest ankieta.

player_answers:

player_id: Unikalny identyfikator zawodnika wypełniającego ankietę.

mental_condition: Ocena stanu psychicznego zawodnika.

physical_condition: Ocena stanu fizycznego zawodnika.

motivation: Ocena motywacji zawodnika.

notes: Dodatkowe notatki dotyczące odpowiedzi zawodnika.

events:

id: Unikalny identyfikator wydarzenia.

teams_id: Identyfikator drużyny organizującej wydarzenie.

title: Tytuł wydarzenia.

type: Typ wydarzenia.

descr: Opis wydarzenia.

event_datetime: Data i czas wydarzenia.

event_players:

event_id: Identyfikator wydarzenia.

player_id: Identyfikator zawodnika biorącego udział w wydarzeniu.

state: Stan uczestnictwa zawodnika w wydarzeniu. (player_event_state)

team_stats:

team_id: Identyfikator drużyny.

season_id: Identyfikator sezonu.

matches_won: Liczba wygranych meczów. (Default 0)

matches_lost: Liczba przegranych meczów. (Default 0)

matches_drawn: Liczba remisów. (Default 0)

top_scorer: Najlepszy strzelec.

least_cards: Zawodnik z najmniejszą liczbą kartek.

seasons:

id: Unikalny identyfikator sezonu.

season_start: Data rozpoczęcia sezonu.

season_end: Data zakończenia sezonu.

player_stats:

player_id: Identyfikator zawodnika.

season_id: Identyfikator sezonu.

goals: Liczba zdobytych bramek. (Default 0)

assists: Liczba asyst. (Default 0)

yellow_cards: Liczba żółtych kartek. (Default 0)

red_cards: Liczba czerwonych kartek. (Default 0)

attended_matches: Liczba rozegranych meczów. (Default 0)

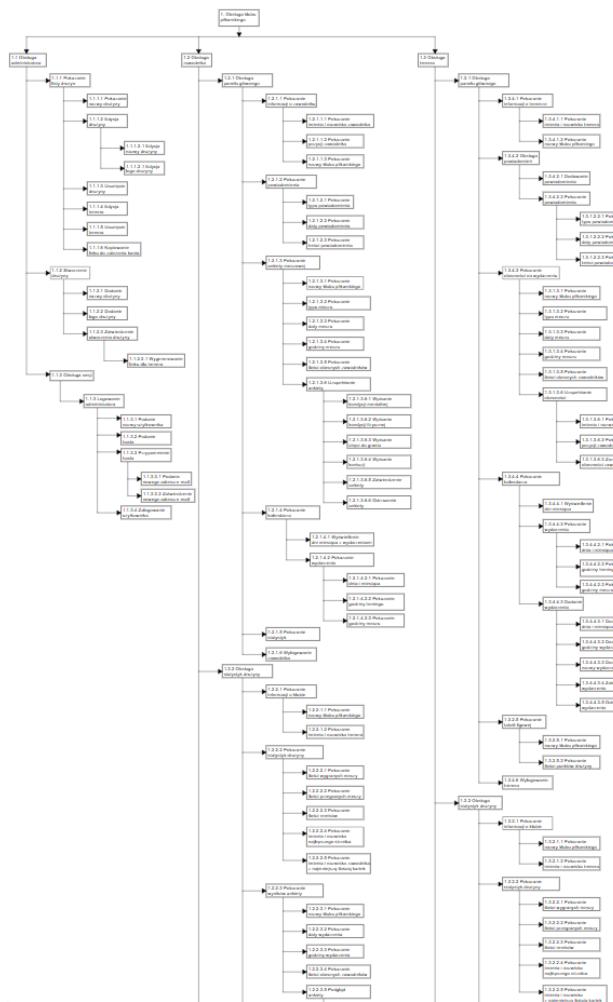
unattended_matches: Liczba opuszczonych meczów. (Default 0)

attended_trainings: Liczba odbytych treningów. (Default 0)

unattended_trainings: Liczba opuszczonych treningów. (Default 0)

3.1. Diagram hierarchii funkcji

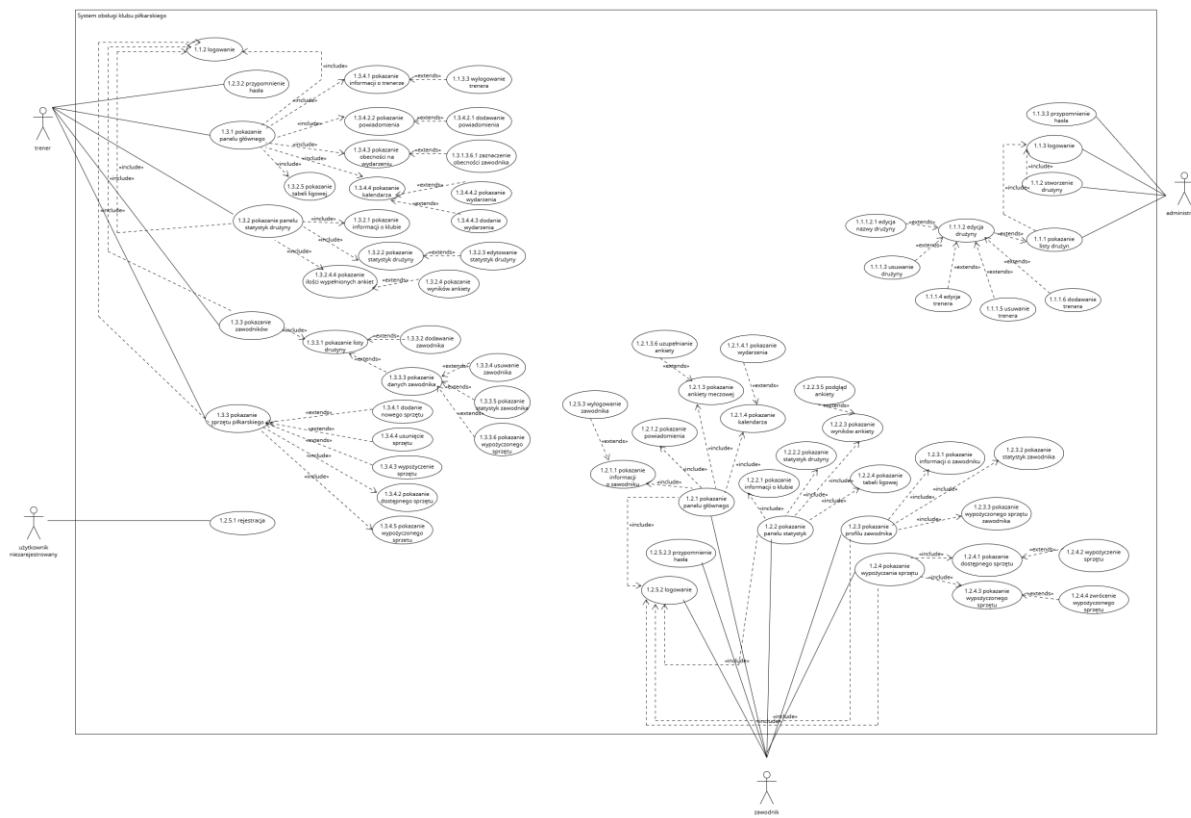
Rysunek ten przedstawia hierarchię funkcji, które zostały odpowiednio zdekomponowane oraz zaimplementowane w systemie. Głównym przeznaczeniem naszej aplikacji jest zarządzanie klubem piłkarskim. Zadanie to zostało podzielone na trzy pomniejsze zadania tak, aby ułatwić ich implementację. Funkcjami tymi są: Obsługa administratora, Obsługa zawodnika, Obsługa trenera. Każda z nich dzieli się na mniejsze funkcje, dzięki czemu można doprecyzować zakres działań związanych z konkretną funkcją. W większości funkcje te dotyczą pokazywania danych, edytowania ich, usuwania oraz dodawania.





3.2. Diagram przypadków użycia

Poniższy diagram przypadków użycia obrazuje większość funkcji, które zostały stworzone w projekcie. Zdefiniowano czterech aktorów, są nimi: Użytkownik niezarejestrowany, Zawodnik, Trener oraz Administrator.



Administrator:

-Zarządzanie użytkownikami: Administrator może dodawać i edytować profile trenerów oraz innych użytkowników systemu, jak również usuwać użytkowników, jeśli zajdzie taka potrzeba.

-Tworzenie drużyn: Administrator może dodawać i edytować drużyny oraz dodawać trenera.

Trener:

-Zarządzanie drużynami: Trener może dodawać nowe drużyny, edytować ich nazwy, zarządzać listą drużyn, przypominać hasła i usuwać drużyny.

-Zarządzanie zawodnikami: Trener ma możliwość rejestracji nowych zawodników, wyświetlania informacji o zawodnikach, przeglądania statystyk zawodników oraz wyników meczów i treningów. Może również edytować i usuwać zawodników.

-Zarządzanie sprzętem: Trener może dodawać nowe sprzęty, wyświetlać listę dostępnych sprzętów, wypożyczać je zawodnikom oraz zarządzać zwrotami i usuwaniem sprzętu.

-Ankiety: Trener może zobaczyć ankiety dla zawodników, przegląda listę ankiet oraz analizuje wyniki tych ankiet.

-Zarządzanie wydarzeniami: Trener może organizować wydarzenia, takie jak mecze i treningi, edytować ich szczegóły oraz usuwać je z kalendarza. Ma również możliwość przeglądania wszystkich wydarzeń.

Zawodnik:

-Przeglądanie informacji: Zawodnik może wyświetlać listy treningów i meczów, przeglądać szczegóły poszczególnych wydarzeń oraz analizować statystyki meczów, treningów i swoje własne.

-Zarządzanie własnym kontem: Zawodnik może zarejestrować się w systemie, logować do niego oraz przypominać hasła w przypadku ich zapomnienia.

-Uczestniczenie w wydarzeniach: Zawodnik może brać udział w treningach i meczach oraz przeglądać harmonogram nadchodzących wydarzeń.

-Zarządzanie sprzętem: Zawodnik ma możliwość wypożyczania dostępnego sprzętu, zwracania go oraz przeglądania dostępnych zasobów.

-Wypełnianie ankiet: Zawodnik ma możliwość wypełniania ankiet dotyczących stanu zdrowia i ewentualnych kontuzji.

• Podsumowanie

Dokumentacja ta ma na celu pokazanie w jaki sposób stworzona została aplikacja do zarządzania klubem piłkarskim oraz jak wiele pracy było konieczne, aby móc zrealizować ten projekt. Uwzględnione w niej zostały zadania związane z tworzeniem bazy danych, backendem oraz frontendem a także szczegółowy opis działania aplikacji od strony użytkownika.

Podsumowując, zrealizowany projekt systemu zarządzania klubem piłkarskim stanowi solidne i funkcjonalne narzędzie, które usprawnia pracę trenerów, administratorów oraz zawodników. System nie tylko zwiększa efektywność zarządzania klubem, ale także poprawia komunikację i organizację wewnętrzną, co w efekcie przyczynia się do lepszych wyników sportowych i satysfakcji wszystkich użytkowników.

Załączniki

- [1] Plik z projektem
- [2] Plik z dokumentami i diagramami

Spis rysunków i tabel

Tabela 1 Wykorzystane technologie 6

Rysunek 2 Wykres przedstawiający najbardziej popularne framework backendowe w 2023 6