POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Praca inżynierska

Na kierunku **Informatyka**
w specjalności **Inżynieria Oprogramowania**

System wspomagający działanie klubu siatkarskiego

A system supporting the operation of a volleyball club

Dariusz Bochyński
Numer albumu: 99497

Promotor: dr Beata Pańczyk

Lublin rok 2026

Streszczenie

Praca inżynierska przedstawia proces tworzenia systemu informatycznego, który wspomaga działanie profesjonalnego klubu siatkarskiego. Aplikacja została zbudowana przy wykorzystaniu technologii Express oraz React. Pracę rozpoczęto od analizy rynku pod kątem istniejących rozwiązań, co pozwoliło na zdefiniowanie celu oraz wymagań projektu. W rozdziale dotyczącym projektowania systemu przedstawiono scenariusze i diagramy przypadków użycia, diagram klas, BPMN oraz sekwencji. Obrazują one działanie procesów w aplikacji. Pokazano również strukturę relacyjnej bazy danych opartej na MySQL oraz projekty interfejsu graficznego aplikacji. Podczas prezentowania kolejnych etapów tworzenia aplikacji, za zgodą klubu wykorzystywany był wizerunek klubu Bogdanka LUK Lublin, jako przykładowej drużyny. W kolejnej części pracy pokazano proces implementacji systemu z zastosowaniem ról użytkowników o różnych uprawnieniach, na początku części serwerowej a następnie aplikacji klienckiej. Zaprezentowano tam i wyjaśniono najistotniejsze funkcje, takie jak panele pracowników oraz administratorów i pokazano interfejsy użytkowników. Następna część poświęcona została sprawdzaniu poprawności działania aplikacji za pomocą testów manualnych, które nie wykazały błędów. Na końcu znalazło się podsumowanie i wnioski, które zostały wyciągnięte podczas tworzenia aplikacji.

Słowa kluczowe: system informatyczny, Express, React, MySQL, klub siatkarski

Abstract

The engineering thesis presents the process of developing an information system that supports the operation of a professional volleyball club. The application was built using Express and React technologies. The work began with a market analysis of existing solutions, which made it possible to define the objective and requirements of the project.

The chapter devoted to system design presents scenarios and diagrams of use cases, class diagrams, BPMN diagrams, and sequence diagrams. These illustrate the operation of processes within the application. The structure of a relational database based on MySQL as well as the designs of the application's graphical user interface are also presented. During the presentation of subsequent stages of application development, with the consent of the club, the image of the Bogdanka LUK Lublin club was used as an example team.

The next part of the thesis describes the system implementation process, including the use of user roles with different levels of authorization, starting with the server-side component and then the client-side application. The most important functionalities are presented and explained, such as employee and administrator panels, along with the user interfaces.

The following section is devoted to verifying the correctness of the application's operation using manual tests, which did not reveal any errors. The thesis concludes with a summary and conclusions drawn during the development of the application.

Keywords**:** information system, Express, React, MySQL, volleyball club

ZGODA NA NIEKOMERCYJNE WYKORZYSTANIE WIZERUNKU KLUBU I ZAWODNIKÓW

Ja, niżej podpisana Anna Sobka, rzecznik prasowy klubu **Bogdanka LUK Lublin**, działając w imieniu i na rzecz klubu **Bogdanka LUK Lublin**, oświadczam, że wyrażam zgodę na niekomercyjne wykorzystanie wizerunku klubu **Bogdanka LUK Lublin**, w tym jego nazwy, herbu, barw klubowych oraz wizerunku zawodników występujących w barwach klubu, w zakresie niezbędnym do realizacji pracy dyplomowej inżynierskiej.

Zgoda dotyczy wykorzystania powyższych elementów wyłącznie w celach edukacyjnych i naukowych, w szczególności w pracy inżynierskiej pt. **„System wspomagający działanie klubu siatkarskiego"**, przygotowywanej na potrzeby ukończenia studiów inżynierskich na **Wydziale Elektrotechniki i Informatyki Politechniki Lubelskiej**.

Wizerunek klubu oraz zawodników może być wykorzystywany w treści pracy, materiałach graficznych, zrzutach ekranu oraz prezentacjach związanych z obroną pracy, pod warunkiem że:

- wykorzystanie ma charakter wyłącznie niekomercyjny,
- nie narusza dóbr osobistych zawodników ani dobrego imienia klubu,
- nie będzie używane do celów marketingowych ani reklamowych.

Niniejsza zgoda nie obejmuje prawa do dalszego rozpowszechniania materiałów w celach komercyjnych oraz nie upoważnia do ich modyfikacji w sposób naruszający wizerunek klubu lub zawodników.

Oświadczenie zostaje wydane dobrowolnie i bezterminowo, wyłącznie na potrzeby wskazanej pracy inżynierskiej.

Miejscowość, data: Lublin, 02.02.2026

LKPS Lublin Sp. z o.o.
ul. Czeremchowa 11, 20-807 Lublin
NIP: 7123422481, REGON: 389594564
KRS: 0000971631

Podpis.....

Sobka Anna

Spis treści

1. Wstęp

Profesjonalny sport drużynowy jest kojarzony głównie z wynikami danego sportowca czy drużyny. Należy jednak pamiętać, że sukces zespołu składa się często z pracy wielu osób, również tych odpowiedzialnych za marketing oraz wizerunek. Każda drużyna rywalizująca w najwyższej dywizji siatkarskiej w Polsce powinna dysponować systemem informatycznym, który kompleksowo spaja wszystkie instancje klubu w jedną spójną całość.

Prezentowane praca inżynierska przedstawia system oferujący zintegrowaną przestrzeń do zarządzania klubem siatkarskim łącząc w sobie między innymi funkcjonalności takie jak sklep klubowy, galerię multimedialną, artykuły prasowe czy podgląd aktualnego składu drużyny. Zastosowany został podział na role użytkowników o rożnym stopniu uprawnień, który umożliwia pracę osobom zatrudnionym przez klub, które mają do dyspozycji panele pracownicze lub administratorskie. Mogą dokonywać tam zmian, które następnie będą widoczne dla pozostałych użytkowników serwisu. Takie rozwiązanie stanowi kompleksową odpowiedź na wyzwania z jakimi mierzy się na co dzień profesjonalny klub siatkarski, który dba o właściwą ekspozycję swojej marki. Przedstawiona aplikacja oferuje wygodną i przejrzystą aplikację internetową, która wspiera zarządzanie klubem siatkarskim oraz ułatwia promowanie treści, które mogą być pomocne w rozwoju profesjonalnej drużyny.

Podsumowując, praca obejmuje projektowanie oraz implementacje aplikacji, która ułatwi funkcjonowanie profesjonalnemu klubowi siatkarskiemu. Dzięki zastosowaniu szkieletów programistycznych takich jak Express i React projekt jest przygotowany do wdrożenia oraz dalszego rozwoju, dzięki czemu jest odpowiedzią na realne problemy użytkowników.

W pracy jako przykład, za zgodą klubu wykorzystywany jest wizerunek Bogdanki LUK Lublin, drużyny, której autor pracy jest czynnym kibicem oraz wolontariuszem z wieloletnim stażem. Realizacja projektu była wynikiem zaangażowania autora oraz zainteresowania omawianym zagadnieniem.

2. Cel i zakres pracy

2.1. Cel pracy

Celem pracy jest zbudowanie działającego systemu informatycznego wspomagającego funkcjonowanie profesjonalnego klubu siatkarskiego. Ma on integrować wszystkie instancje klubu w formie aplikacji internetowej.

2.2. Zakres pracy

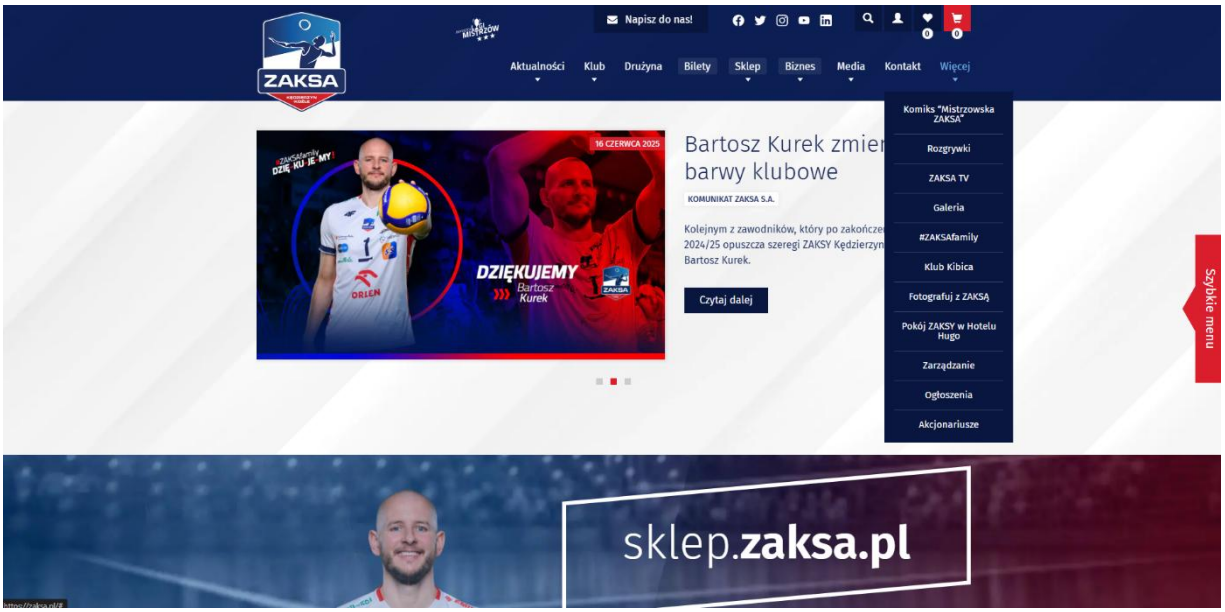
- analiza rynku pod kątem istniejących rozwiązań – zbadanie systemów informatycznych o tematyce zbliżonej do projektowanego systemu oraz sprawdzenie ich funkcjonalności.
- projekt systemu informatycznego – wykonanie dokumentacji projektowej:
    - określenie wymagań funkcjonalnych oraz niefunkcjonalnych projektowanego systemu informatycznego.
    - utworzenie diagramów przypadków użycia dla każdej z zaprojektowanych ról użytkowników w systemie.
    - opisanie scenariuszy przypadków użycia.
    - utworzenie diagramu klas dla projektowanego systemu informatycznego.
    - utworzenie diagramów sekwencji dla projektowanego systemu informatycznego.
    - utworzenie diagramów BPMN dla projektowanego systemu informatycznego.
    - zaprojektowanie bazy danych potrzebnej do funkcjonowania systemu.
    - zaprojektowanie graficznego interfejsu użytkownika dla opracowywanego systemu informatycznego.
- wybór odpowiednich technologii oraz narzędzi i rozwiązań do realizacji aplikacji – do tworzenia poszczególnych części systemu wykorzystano:
    - implementacja części klienckiej - React.[5]
    - implementacja części serwerowej - Express.[8]
    - wykorzystywana baza danych - MySQL.[13]
- implementacja systemu informatycznego – tworzenie aplikacji poprzez kodowanie jej w oparciu o wcześniejsze poprzednie etapy projektu.
- testowanie systemu – sprawdzanie utworzonej aplikacji pod kątem poprawności funkcjonowania.
- podsumowanie – przedstawienie całego projektu oraz wyciągnięcie wniosków.

3. Analiza rynku

Przeprowadzenie analizy było kluczowe podczas projektowania systemu, gdyż na podstawie istniejących już rozwiązań należało podjąć decyzje dotyczącą wizji jak najlepiej funkcjonującej aplikacji.

Każdy klub siatkarski, występujący w najwyższej lidze rozgrywkowej jaką jest Plus Liga posiada własną stronę internetową, dzięki której prowadzi komunikację ze społecznością dodając wpisy dotyczące wydarzeń związanych z drużyną, pozwala na sprawdzenie składu czy kupno biletów na nadchodzące mecze. W celu dokonania analizy wybrano pięć pozycji należących do obecnie czołowych zespołów.

Zaksa Kędzierzyn-Koźle na swojej witrynie internetowej [15] udostępnia użytkownikowi podstawowe funkcjonalności, takie jak czytanie aktualności, sprawdzenie składu osobowego drużyny, kupno biletu lub produktów klubowych, kontakt czy galeria. Strona jest czytelna, ale górne menu nawigacyjne jest bardzo złożone (Rysunek 3.1), co spowalnia wyszukiwanie treści wybranych przez użytkownika. Należy zaznaczyć również, że sprzedaż biletów prowadzona jest przez zewnętrzny podmiot zajmujący się obsługą wydarzeń.



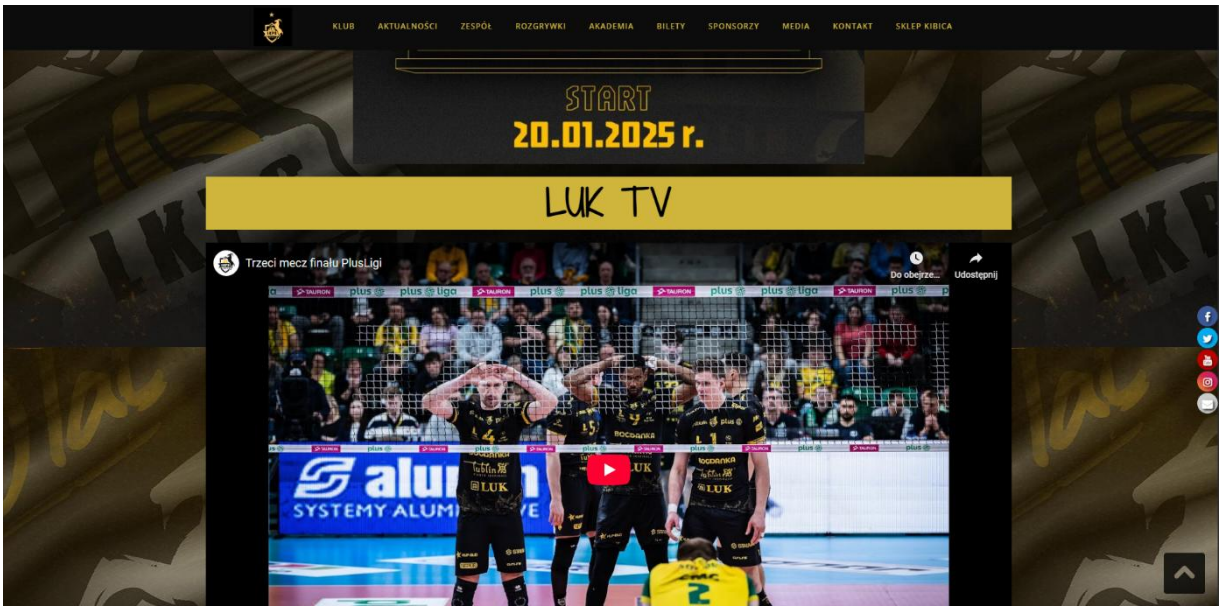
Rysunek 3.1 Strona internetowa Zaksy Kędzierzyn-Koźle

Widok mobilny strony (Rysunek 3.2) jest spójny i przejrzysty. Dostęp do funkcjonalności systemu jest prosty a obsługa intuicyjna. Na każdej platformie aplikacja jest responsywna, odpowiedzi systemu następują błyskawicznie.

Rysunek 3.2 Widok mobilny strony internetowej Zaksy Kędzierzyn-Koźle

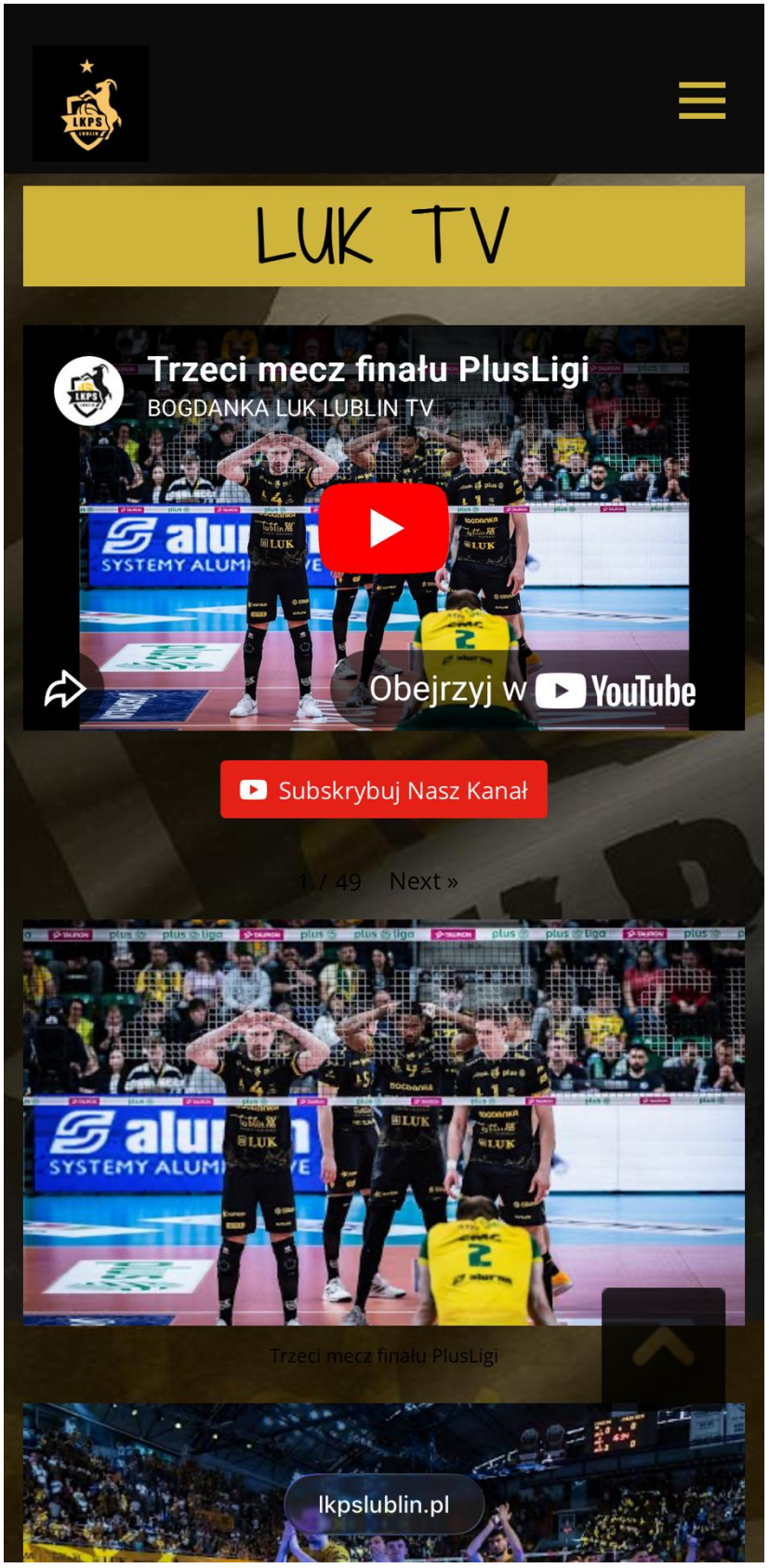
Następna analizie została poddana strona drużyny Bogdanka LUK Lublin [3]. Wyróżnia ją wysoki kontrast kolorystyczny, co poprawia jej odbiór [7]. Zawiera ona podobne funkcjonalności do powyżej analizowanego przykładu, przy czym układ elementów strony jest znacznie mniej czytelny. Wyeksponowane zostało wiele elementów, które mogłyby zostać

pokazane w sposób subtelny, kroje czcionek nie są jednolite, podstrony nie robią wrażenia spójnych (Rysunek 3.3).



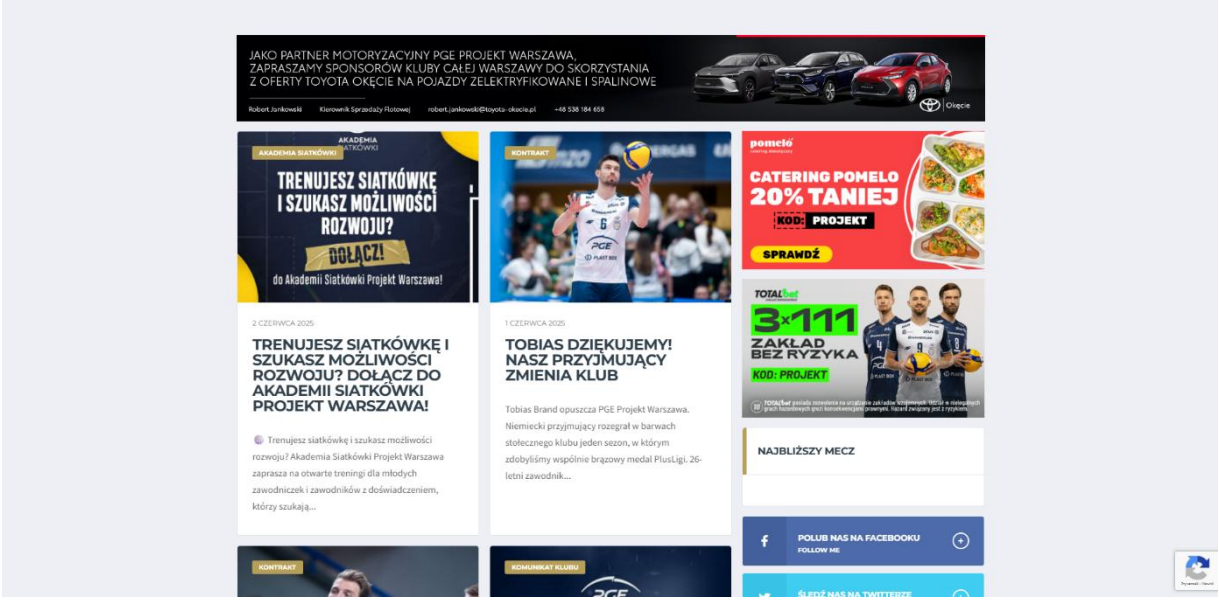
Rysunek 3.3 Strona internetowa Bogdanki LUK Lublin

Widok mobilny omawianej aplikacji (Rysunek 3.4) cechuje się podobną stylistyką jak wygląd systemu w przeglądarce komputerowej, przez co strona jest mało czytelna i nie zachęca użytkownika do korzystania z niej. Charakteryzuje ją brak spójności, jednak responsywność jest na dobrym poziomie.



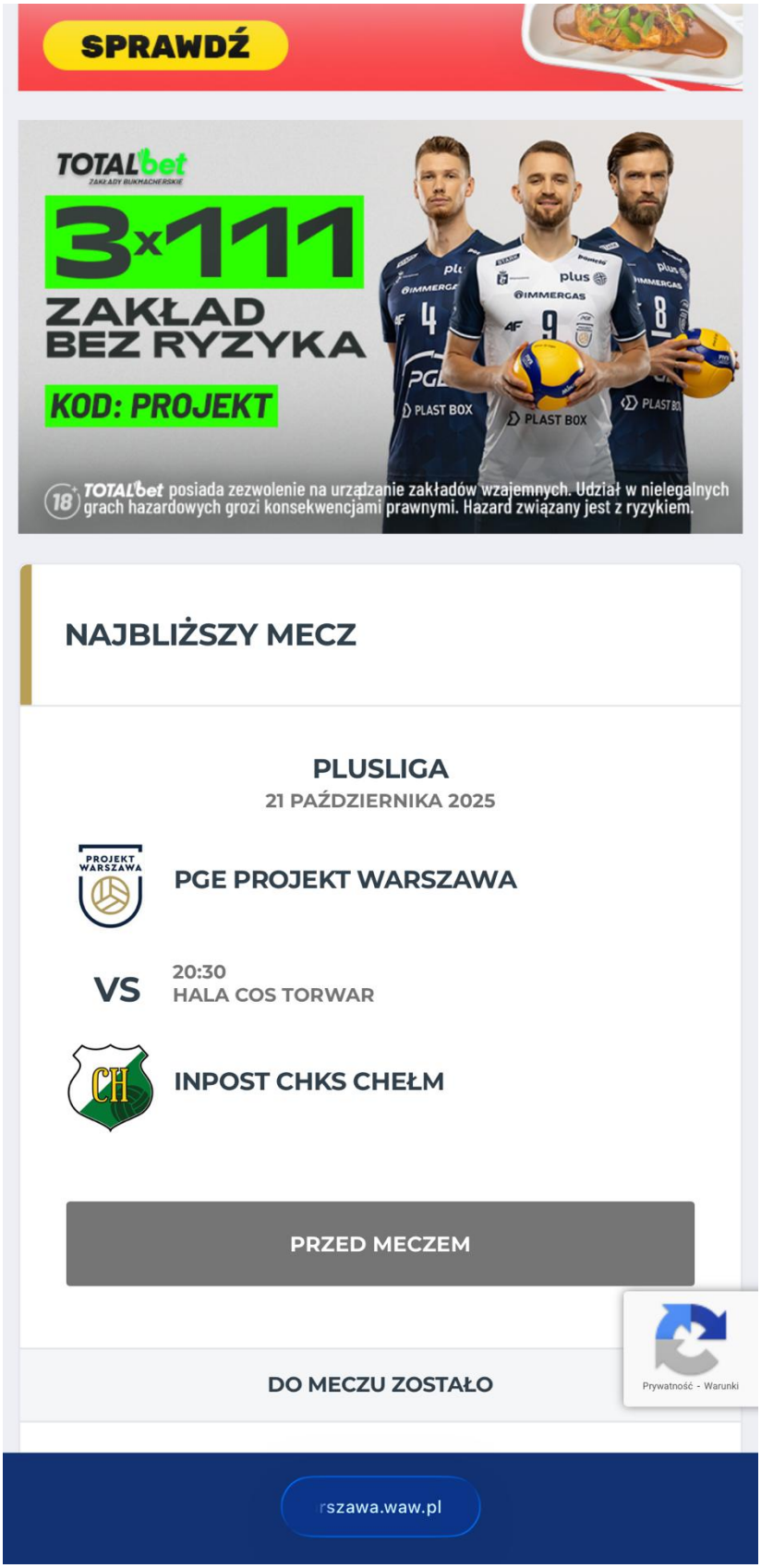
Rysunek 3.4 Widok mobilny strony internetowej Bogdanki LUK Lublin

Portal należący do Projektu Warszawa [11] sprawia wrażenie spójnego i przystępnego dla użytkowników, gdyż wszystkie funkcje opisane są w prosty sposób [9]. Jednak na niekorzyść tej strony internetowej działa umieszczenie na niej wielu niepasujących do kolorystyki i stylu strony reklam [4] (Rysunek 3.5).



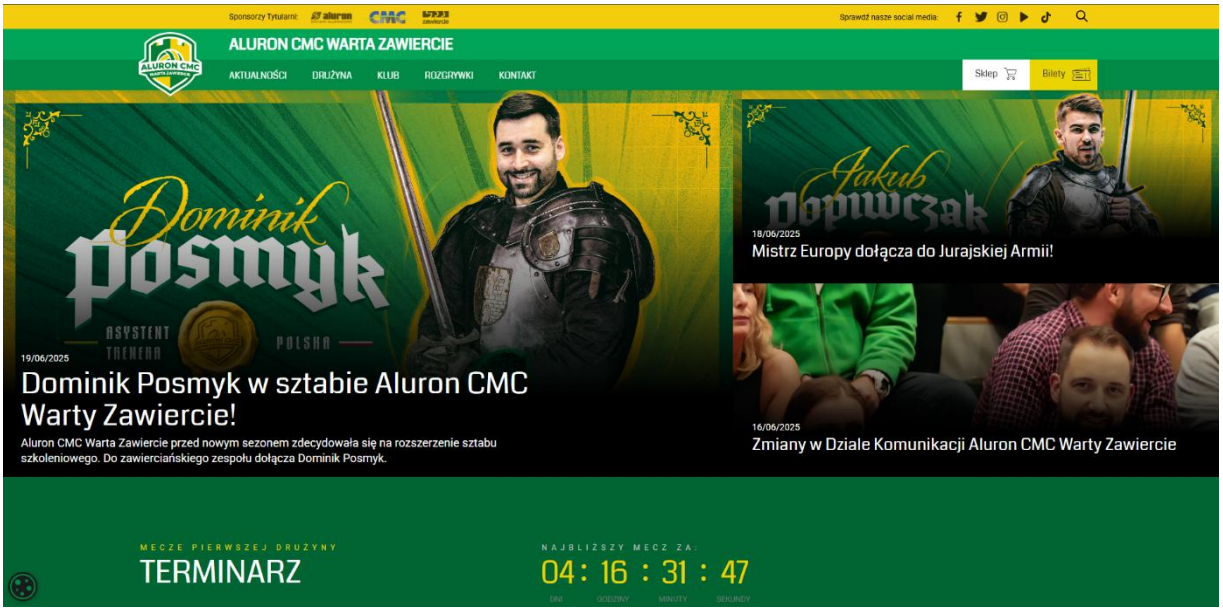
Rysunek 3.5 Strona internetowa Projektu Warszawa

Widok mobilny (Rysunek. 3.6.) jest czytelny oraz responsywny. Poruszanie się po stronie nie sprawia użytkownikom problemów.



Rysunek 3.6 Widok mobilny strony internetowej Projektu Warszawa

Strona Aluron CMC Warty Zawiercie [1] wśród konkurencji wyróżnia własny system do sprzedaży biletów. Jest to rzadkością, gdyż konkurencyjne serwisy często używają do tego zewnętrznych operatorów wydarzeń. Innym ciekawym rozwiązaniem na tej witrynie jest zegar czasu rzeczywistego, który odlicza czas do kolejnego meczu, budując napięcie i podkreślając wagę tego wydarzenia (Rysunek 3.7).



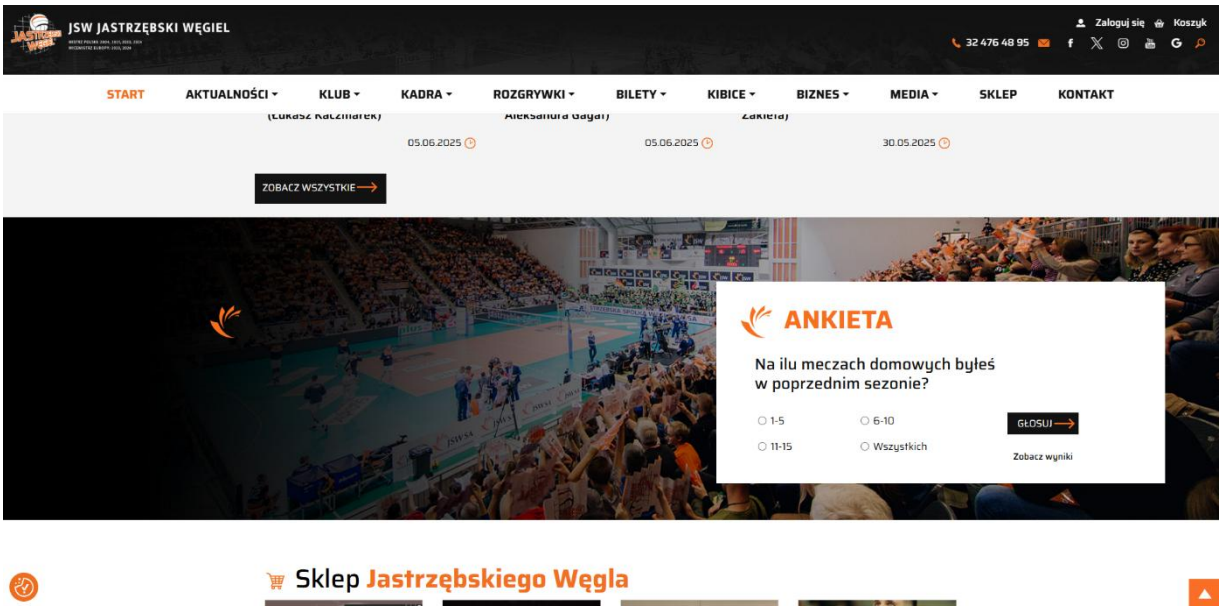
Rysunek 3.7 Strona internetowa Aluron CMC Warty Zawiercie

W widoku mobilnym (Rysunek 3.8.) aplikacji w sposób subtelny i spójny przedstawione zostały funkcjonalności, które znalazły się w wersji na przeglądarki komputerowe. Responsywność strony na wszystkich platformach jest na wysokim poziomie, wszelkie operacje wykonywane są błyskawicznie.

Rysunek 3.8 Widok mobilny strony internetowej Aluron CMC Warty Zawiercie

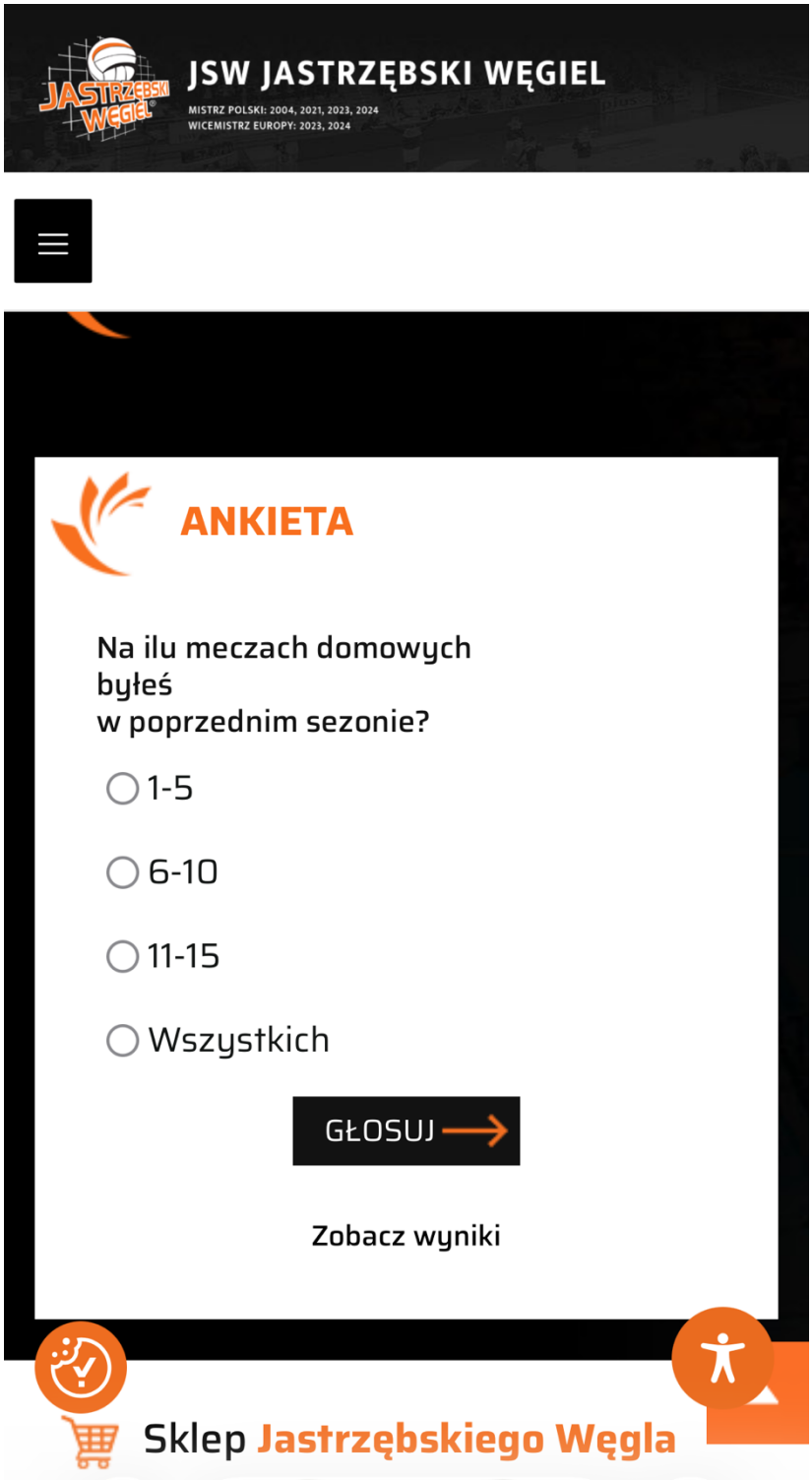
Ostatnim analizowanym przykładem jest strona drużyny JSW Jastrzębski Węgiel [10], która wchodzi z użytkownikiem w jeszcze większą interakcję poprzez proste ankiety, które znajdują się na ekranie głównym. Aby wziąć w niej udział użytkownik nie musi być zalogowany, co na pewno zachęca internautów do zaznaczenia jednej z opcji. Taki zabieg może

być pomocny dla osób zajmujących się marketingiem, ponieważ w prosty sposób mogą poznać preferencje odbiorców (Rysunek 3.9).



Rysunek 3.9 Strona internetowa klubu JSW Jastrzębski Węgiel

Widok mobilny aplikacji (Rysunek 3.10.) wyróżnia ikona prowadząca do funkcjonalności pozwalającej na poprawę dostępności witryny, poprzez dostawanie wyglądu strony. To działanie może poprawić odbiór aplikacji oraz przyciągnąć większe grono użytkowników. Strona internetowa nie ma problemu z działaniem i responsywnością na żadnej z dostępnych platform.



Rysunek 3.10 Widok mobilny strony internetowej JSW Jastrzębski Węgiel

Analiza rynku wykazała, że większość stron należących do klubów siatkarskich posiada powtarzające się funkcjonalności takie jak aktualności, skład czy sklep. Poszczególne podmioty starają się zachęcić potencjalnych odbiorców wprowadzając niewielkie udogodnienia. Jednakże wszystkie z wymienionych aplikacji skupiają się wyłącznie na kibicach, pracownicy klubu nie

mają tam uwzględnionych własnych wyspecjalizowanych funkcjonalności. W tej pracy przedstawiony zostanie proces projektowania, tworzenia oraz implementacji systemu, który będzie łączył w sobie najlepsze rynkowe rozwiązania z nowatorskimi pomysłami na zarządzanie i rozwój profesjonalnego klubu siatkarskiego.

4. Projekt systemu

Celem niniejszej pracy inżynierskiej było stworzenie aplikacji umożliwiającej wspomaganie zarządzania profesjonalnym klubem siatkarskim.

4.1. Specyfikacja wymagań funkcjonalnych

Zdefiniowanie wymagań funkcjonalnych było jednym z pierwszych etapów projektowania aplikacji. Poniżej przedstawiono wymagania funkcjonalne dla aktorów: użytkownik niezalogowany, użytkownik zalogowany, pracownik klubu, administrator.

Użytkownik niezalogowany – użytkownik nieposiadający konta w serwisie

1. Rejestracja:
    - założenie nowego konta użytkownika.
2. Logowanie:
    - zalogowanie się na istniejące konto.
3. Artykuły prasowe:
    - przegląd listy artykułów,
    - przegląd treści artykułu.
4. Historia klubu:
    - przegląd osi czasu i wydarzeń historycznych.
5. Wyniki drużyny:
    - przegląd wyników meczów,
    - przegląd informacji o spotkaniach (data, miejsce, wynik, zawodnik meczu).
6. Skład drużyny:
    - przegląd listy zawodników i sztabu szkoleniowego,
    - przegląd szczegółowych informacji o członkach drużyny.
7. Galeria:
    - przegląd zdjęć i materiałów wideo.
8. Sponsorzy:
    - przegląd listy sponsorów,
    - dostęp do linków sponsorów (strony WWW, media społecznościowe).
9. Ankiety:
    - udział w anonimowych ankietach dotyczących działalności klubu.

10. Sklep klubowy:
    - przegląd dostępnych produktów klubowych.
11. Bilety:
    - przekierowanie do zewnętrznego systemu sprzedaży biletów.

Użytkownik zalogowany – użytkownik zalogowany do serwisu

1. Konto użytkownika:
    - edycja danych osobowych,
    - zmiana hasła użytkownika,
    - usunięcie konta z portalu.
2. Artykuły prasowe:
    - dodanie reakcji w formie emotikonów,
    - dodanie komentarza,
    - usunięcie komentarza.
3. Sklep klubowy:
    - dodanie produktu do koszyka,
    - usunięcie produktu z koszyka,
    - złożenie zamówienia.
4. Widomości do administratora:
    - wysłanie zgłoszenia błędu lub uwagi do administratora.

Pracownik klubu – użytkownik zatrudniony przez klub

1. Artykuły prasowe:
    - dodanie nowego artykułu,
    - edycja artykułu,
    - usunięcie artykułu.
2. Wyniki drużyny:
    - Dodawanie wyniku spotkania,
    - edycja informacji o meczu,
    - oznaczenie zawodnika meczu.
3. Galeria:
    - Dodawanie nowego zdjęcia lub filmu,
    - edycja materiału multimedialnego,

- sunięcie materiału multimedialnego.

4. Sponsorzy:
    - dodanie nowego sponsora do listy,
    - edytowanie informacji o sponsorach,
    - przesłanie prośby o usunięcie sponsora do administratora.
5. Ankiety:
    - podgląd wyników ankiet,
    - dodawanie ankiet,
    - modyfikacja ankiet,
    - usuwanie ankiet.
6. Sklep klubowy:
    - przegląd zamówień użytkowników,
    - zmiana statusu zamówienia,
    - modyfikacja informacji o stanie magazynowym danego produktu.
7. Panel pracownika:
    - dostęp do narzędzi dedykowanych realizacji zadań klubowych.

Administrator – użytkownik sprawujący nadzór nad prawidłowym działaniem systemu

1. Artykuły prasowe:
    - pełna kontrola nad zawartością i uprawnieniami w sekcji.
2. Historia klubu:
    - edycja treści osi czasu,
    - dodanie nowych wydarzeń do osi czasu,
    - usunięcie wydarzeń z osi czasu.
3. Wyniki drużyny:
    - zarządzanie wynikami i informacjami o spotkaniach.
4. Skład drużyny:
    - zarządzanie listą zawodników i sztabu szkoleniowego.
5. Galeria:
    - zarządzanie materiałami multimedialnymi.
6. Sponsorzy:
    - rozpatrywanie próśb o usunięcie sponsora,

- pełna kontrola nad listą sponsorów.

7. Ankiety:
    - zarządzanie ankietami.
8. Sklep klubowy:
    - zarządzanie sklepem klubowym.
9. Bilety:
    - zarządzanie integracją funkcjonalności zewnętrznego systemu sprzedaży biletów.
10. Wiadomości administratora:
    - zarządzanie zgłoszeniami od użytkowników.
11. Panel administratorski
    - zarządzanie zawartością serwisu i użytkownikami.

4.2. Specyfikacja wymagań niefunkcjonalnych

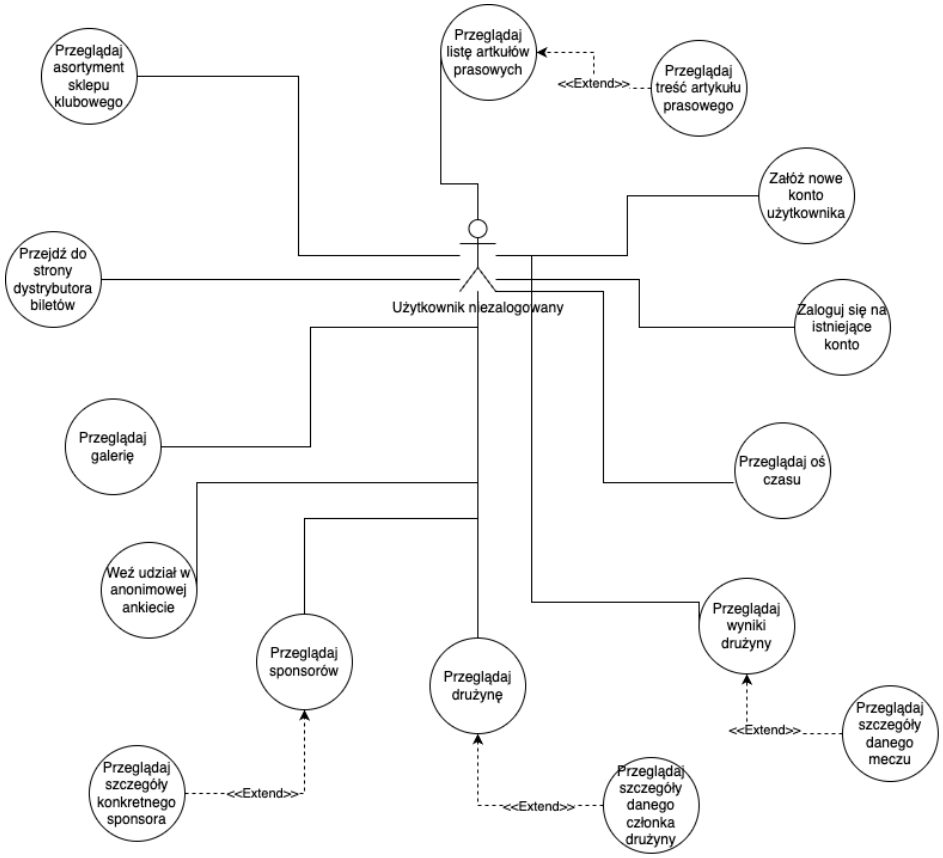
Aplikacji zostały postawione następujące wymagania niefunkcjonalne:

1. Użytkownicy powinni mieć dostęp do systemu korzystając z najpopularniejszych przeglądarek internetowych w wersjach takich jak:
    - Google Chrome – w wersji 136.0.7103.113 lub nowszej
    - Opera – w wersji 119.0.5497.29 lub nowszej
    - Microsoft Edge – w wersji 134.0.3124.93 lub nowszej
    - Safari – w wersji 18.4 (20621.1.15.11.10) lub nowszej
2. Wszystkie hasła w systemie powinny być przechowywany w niejawnej formie.
3. Interfejs aplikacji internetowej powinien być intuicyjny oraz przejrzysty.
4. Aplikacja powinna automatycznie dostosowywać układ i rozmiar elementów graficznych do różnych rozdzielczości ekranów, zapewniając poprawne działanie.
5. System powinien być zaprojektowany w sposób umożliwiający łatwe zwiększenie zasobów w przypadku wzrostu liczby użytkowników

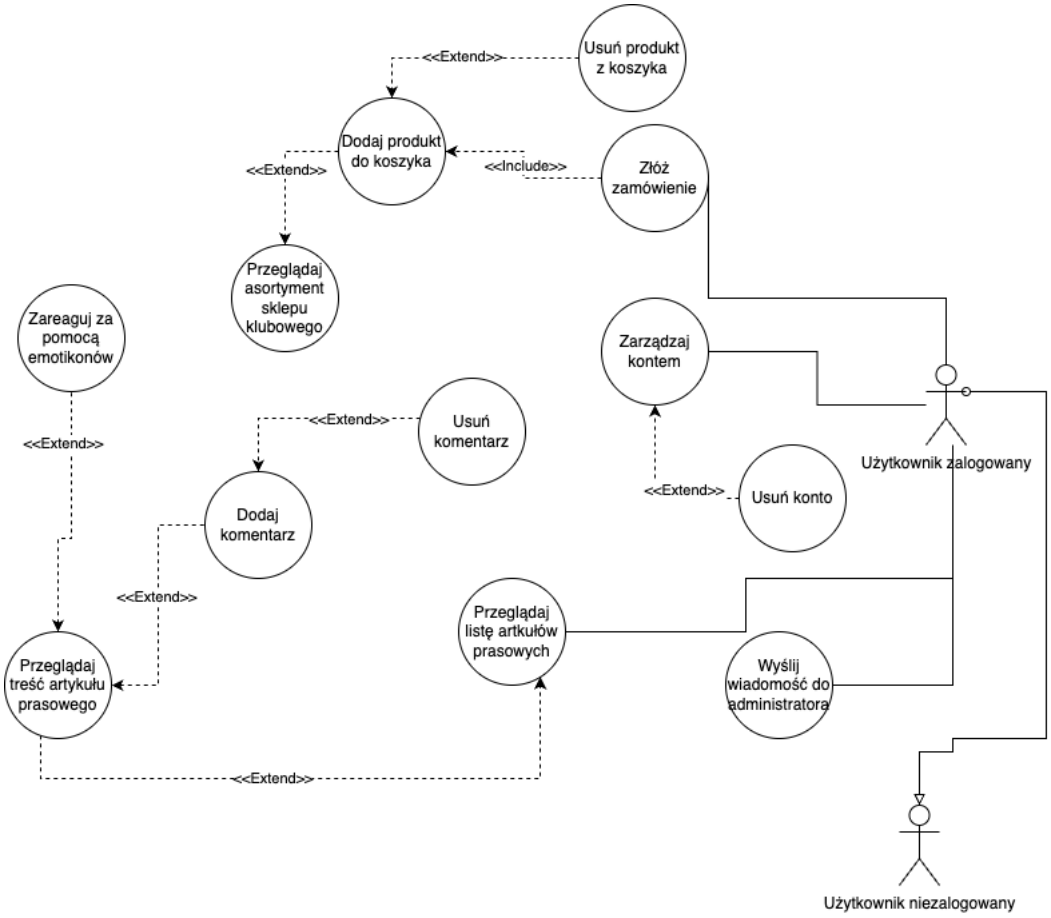
4.3. Diagramy przypadków użycia

Diagramy przypadków użycia są ważną częścią notacji UML (*ang*. *Unified Modelling Language*), która jest szeroko stosowanym standardem wykorzystywanym do modelowania

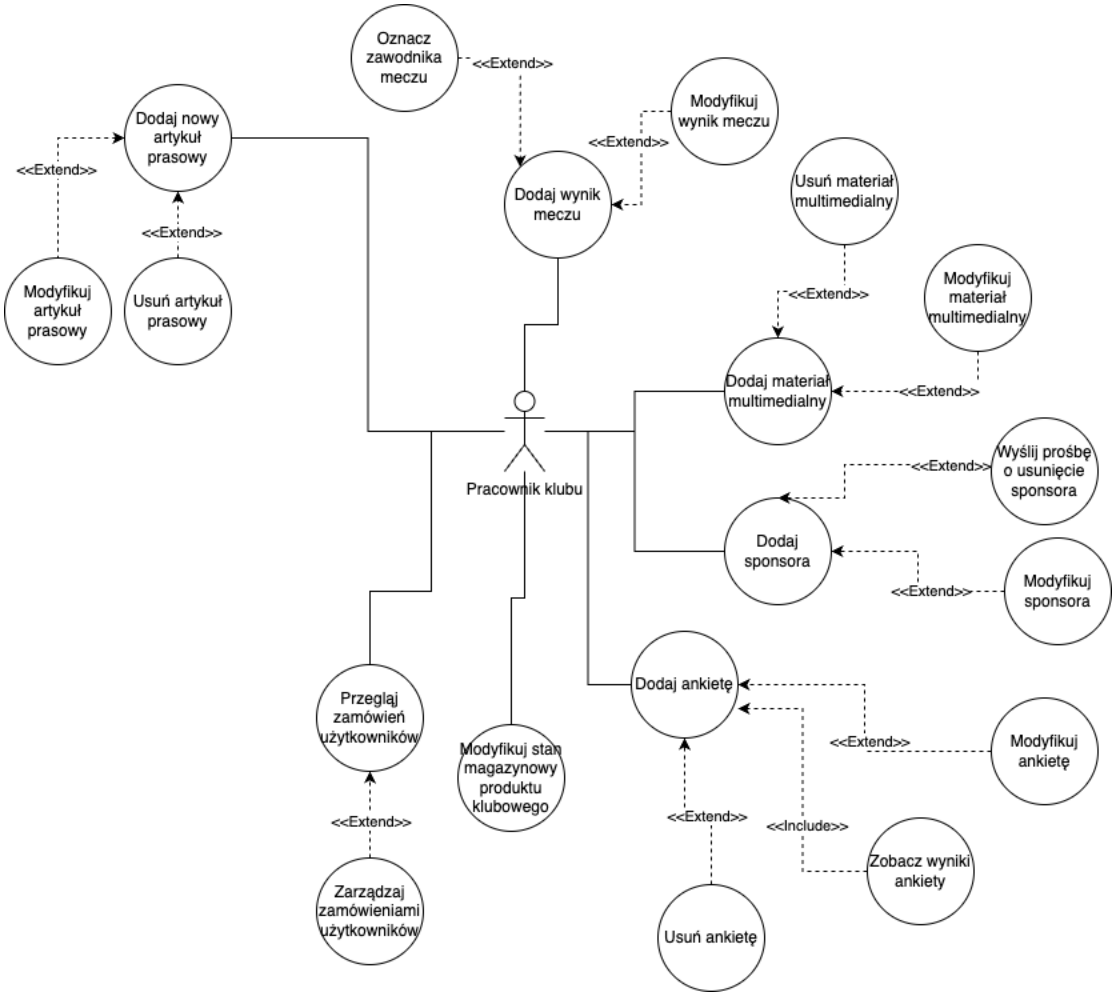
systemów informatycznych. Pokazują one funkcjonalności aplikacji z punktu widzenia użytkowników - aktorów. Ilustrują relacje pomiędzy użytkownikami a systemem, co pomaga w zdefiniowaniu potrzeb funkcjonalnych. Diagramy przypadków użycia dla zaprojektowanego systemu przedstawiono na Rysunkach 4.1 – 4.4.



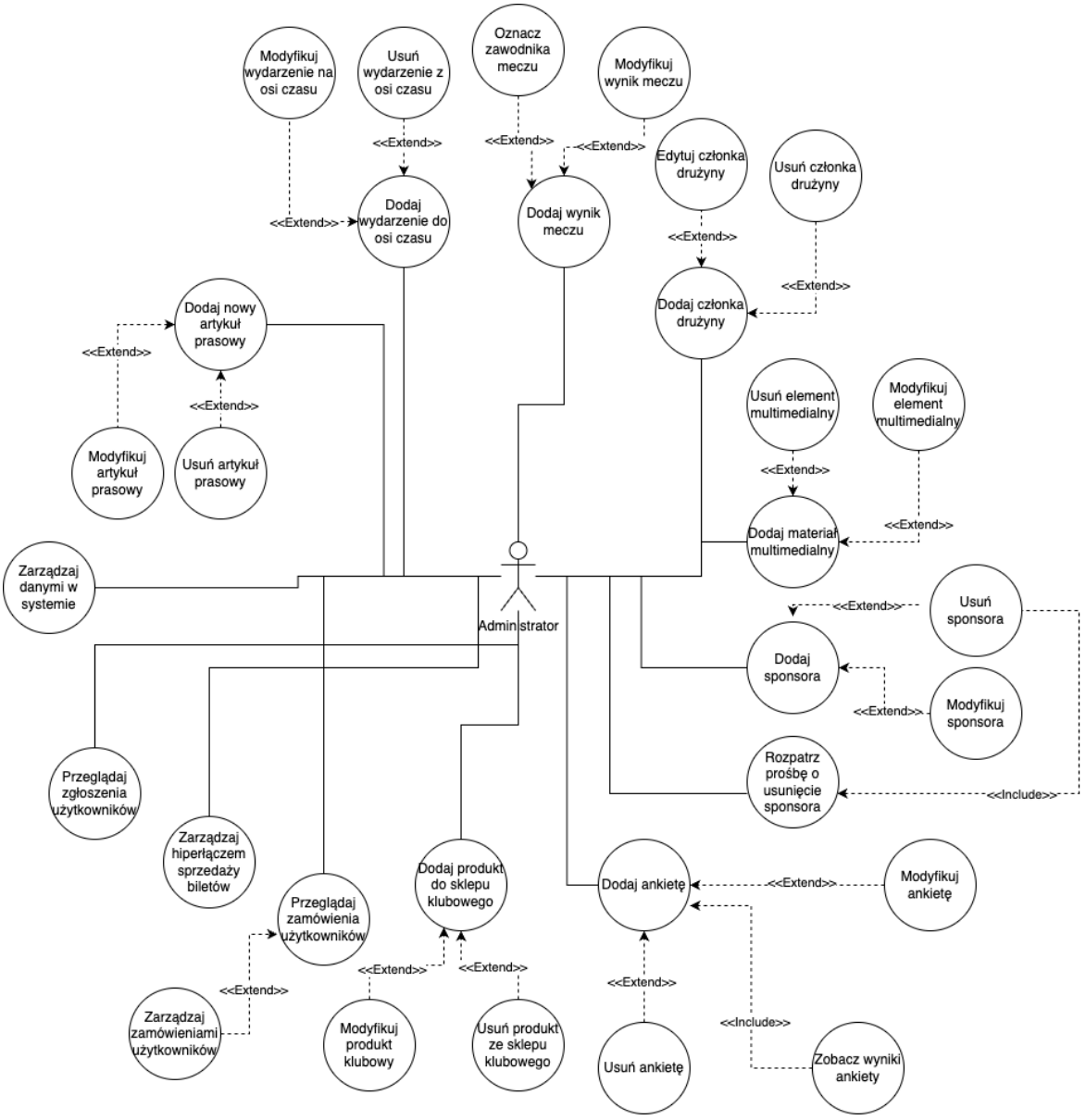
Rysunek 4.1. Diagram przypadków użycia użytkownika niezalogowanego



Rysunek 4.2. Diagram przypadków użycia użytkownika zalogowanego



Rysunek 4.3. Diagram przypadków użycia pracownika klubu



Rysunek 4.4. Diagram przypadków użycia administratora

4.4. Scenariusze przypadków użycia

Scenariusze przypadków użycia skupiają się na opisie kroków interakcji pomiędzy aktorami a systemem w celu osiągnięcia określonego celu. Muszą one zdefiniować w jaki sposób projektowany system ma zachowywać się podczas różnych sytuacji, w skład których wchodzą przebiegi podstawowe, alternatywne oraz wyjątkowe, co ułatwia projektowanie, testowanie oraz zrozumienie danej funkcjonalności. Rozdział ten przedstawia przykładowe scenariusze przypadków użycia dla każdej ze zdefiniowanych ról użytkowników.

Scenariusze przypadków użycia użytkownika niezalogowanego

W tabelach 4.1 – 4.3 przedstawiono scenariusze przypadków użycia użytkownika niezalogowanego, które pokazują korzystanie z poszczególnych funkcjonalności przeznaczonej dla tej grupy użytkowników.

Tabela 4.1. Przegląd artykułów prasowych przez użytkownika niezalogowanego

Atrybut	Opis
Nazwa scenariusza	Przegląd artykułów prasowych przez użytkownika niezalogowanego
Abstrakt	Użytkownik niezalogowany przegląda opublikowane artykuły prasowe dotyczące działalności klubu.
Aktorzy	Użytkownik niezalogowany (gość), system
Warunki początkowe	• Użytkownik otwiera stronę bez logowania • W systemie znajdują się dostępne artykuły
Warunki końcowe	• Użytkownik przeczytał artykuł, nie wykonując operacji wymagających logowania
Flow	Krok
Podstawowy	1. Użytkownik przechodzi do sekcji „Artykuły prasowe” 2. System wyświetla listę dostępnych artykułów 3. Użytkownik wybiera artykuł z listy 4. System wyświetla pełną treść wybranego artykułu
Alternatywny	2a. Użytkownik korzysta z wyszukiwarki lub filtrów - system zawęża listę artykułów
Wyjątkowy	3a. Artykuł został usunięty - system wyświetla komunikat o braku treści
Notatki	
Problem	Rozwiązanie
Możliwość wejścia na nieistniejący artykuł przez zapisany link	Przekierowanie na stronę błędu z komunikatem informacyjnym

Tabela 4.2. Udział w ankiecie przez użytkownika niezalogowanego

Atrybut	Opis
Nazwa scenariusza	Udział w ankiecie przez użytkownika niezalogowanego
Abstrakt	Użytkownik niezalogowany bierze udział w anonimowej ankiecie dotyczącej klubu.
Aktorzy	Użytkownik niezalogowany (gość), system
Warunki początkowe	• Ankieta jest aktywna i publicznie dostępna • Użytkownik nie jest zalogowany
Warunki końcowe	• Głos użytkownika zostaje zapisany w systemie jako anonimowy
Flow	
Podstawowy	1. Użytkownik przechodzi do sekcji „Ankiety” 2. System wyświetla aktywne ankiety 3. Użytkownik wybiera jedną z nich 4. Użytkownik zaznacza odpowiedź 5. Użytkownik potwierdza wybór 6. System zapisuje głos i wyświetla podsumowanie wyników
Alternatywny	2a. Użytkownik przegląda archiwalne wyniki ankiet (jeśli system je udostępnia)
Wyjątkowy	4a. Użytkownik nie zaznaczył odpowiedzi - system nie pozwala na zatwierdzenie głosu
Notatki	

Problem	Rozwiązanie
Możliwość wielokrotnego głosowania z tego samego urządzenia	Ograniczenie głosowania przez zapis ciasteczek lub IP

Tabela 4.3. Przejście do zakupu biletów przez użytkownika niezalogowanego

Atrybut	Opis
Nazwa scenariusza	Przejście do zakupu biletów przez użytkownika niezalogowanego
Abstrakt	Użytkownik niezalogowany zostaje przekierowany do zewnętrznej platformy, na której może zakupić bilet na wydarzenie sportowe.
Aktorzy	Użytkownik niezalogowany (gość), system
Warunki początkowe	• Bilety na wydarzenie są dostępne • Użytkownik znajduje się na stronie aplikacji
Warunki końcowe	• Użytkownik zostaje przeniesiony do strony zewnętrznej zajmującej się sprzedażą biletów
Flow	
Podstawowy	1. Użytkownik przechodzi do sekcji „Bilety” 2. System wyświetla dostępne wydarzenia z opcją zakupu 3. Użytkownik wybiera wydarzenie 4. Użytkownik klika przycisk „Kup bilet” 5. System przekierowuje użytkownika na stronę zewnętrznego operatora biletowego
Alternatywny	2a. Użytkownik przeszukuje listę wydarzeń według daty lub lokalizacji
Wyjątkowy	3a. Wydarzenie zostało odwołane lub brak dostępnych biletów - system wyświetla komunikat i blokuje przekierowanie

Notatki	
Problem	Rozwiązanie
Brak jasnej informacji, że zakup odbywa się poza systemem	Dodanie widocznego komunikatu przed przekierowaniem

Scenariusze przypadków użycia użytkownika zalogowanego

W tabelach 4.4 – 4.6 przedstawiono scenariusze przypadków użycia użytkowników zalogowanych, które pokazują korzystanie z poszczególnych funkcjonalności przeznaczonej dla tej grupy użytkowników.

Tabela 4.4. Edycja danych konta przez użytkownika zalogowanego

Atrybut	Opis
Nazwa scenariusza	Edycja danych konta przez użytkownika zalogowanego
Abstrakt	Użytkownik zalogowany aktualizuje swoje dane osobowe, takie jak adres e-mail czy hasło.
Aktorzy	Użytkownik zalogowany, system
Warunki początkowe	• Użytkownik jest zalogowany do serwisu • Dane użytkownika są zapisane w systemie
Warunki końcowe	• Nowe dane zostają zapisane, a użytkownik jest o tym informowany
Flow	
Podstawowy	1. Użytkownik loguje się do systemu 2. Przechodzi do sekcji „Moje konto” 3. Wybiera opcję „Edytuj dane” 4. Aktualizuje wybrane informacje (np. adres e-mail) 5. Potwierdza zmiany 6. System zapisuje nowe dane i wyświetla komunikat o powodzeniu operacji
Alternatywny	4a. Użytkownik zmienia tylko hasło - system wymaga podania starego hasła przed zatwierdzeniem
Wyjątkowy	4b. Użytkownik wpisuje błędne dane (np. niepoprawny format adresu e-mail) – system wyświetla błąd i nie zapisuje zmian
Notatki	

Problem	Rozwiązanie
Brak informacji o wymogach dotyczących hasła	Dodanie wskazówek pod polem zmiany hasła

Tabela 4.5. Dodanie komentarza do artykułu prasowego

Atrybut	Opis
Nazwa scenariusza	Dodanie komentarza do artykułu prasowego
Abstrakt	Zalogowany użytkownik dodaje komentarz pod artykułem prasowym opublikowanym przez klub.
Aktorzy	Użytkownik zalogowany, system
Warunki początkowe	• Użytkownik jest zalogowany • Artykuł jest publicznie dostępny
Warunki końcowe	• Komentarz użytkownika zostaje zapisany i widoczny pod artykułem
Flow	
Podstawowy	1. Użytkownik przechodzi do sekcji „Artykuły prasowe” 2. Wybiera artykuł z listy 3. Przechodzi do sekcji komentarzy 4. Wpisuje treść komentarza 5. Zatwierdza komentarz 6. System zapisuje komentarz i wyświetla go pod artykułem
Alternatywny	3a. Użytkownik odpowiada na inny komentarz - system zapisuje odpowiedź jako komentarz zagnieżdżony
Wyjątkowy	4a. Użytkownik próbuje zatwierdzić pusty komentarz - system wyświetla błąd walidacji
Notatki	

Problem	Rozwiązanie
Możliwość wprowadzenia treści nieodpowiednich	Zastosowanie automatycznej moderacji lub filtrowania treści

Tabela 4.6. Złożenie zamówienia w sklepie klubowym

Atrybut	Opis
Nazwa scenariusza	Złożenie zamówienia w sklepie klubowym
Abstrakt	Scenariusz przedstawia proces zakupu produktu przez zalogowanego użytkownika w sklepie internetowym klubu
Aktorzy	Użytkownik zalogowany, system
Warunki początkowe	• Użytkownik posiada aktywne konto i jest zalogowany do serwisu • Produkt jest dostępny w sklepie
Warunki końcowe	• Zamówienie zostaje zapisane w systemie • Użytkownik otrzymuje potwierdzenie złożenia zamówienia
Flow	
Podstawowy	1. Użytkownik przechodzi do sekcji „Sklep klubowy” 2. System wyświetla dostępne produkty 3. Użytkownik dodaje wybrany produkt do koszyka 4. Użytkownik przechodzi do podglądu koszyka 5. Użytkownik zatwierdza zamówienie 6. System zapisuje zamówienie w bazie danych 7. System informuje użytkownika o złożeniu zamówienia
Alternatywny	4a. Użytkownik usuwa produkt z koszyka przed zatwierdzeniem - zamówienie nie zostaje złożone
Wyjątkowy	3a. Produkt jest niedostępny - system wyświetla komunikat o braku produktu, zamówienie nie może zostać kontynuowane
Notatki	

Problem	Rozwiązanie
Możliwość wielokrotnego kliknięcia przycisku zamówienia	Zastosowanie mechanizmu blokady wielokrotnego wysyłania formularza

Scenariusze przypadków użycia pracownika klubu

W tabelach 4.7 – 4.9 przedstawiono scenariusze przypadków użycia pracowników klubu, które pokazują korzystanie z poszczególnych funkcjonalności zarządzania systemem przeznaczonej dla tej grupy użytkowników.

Tabela 4.7. Dodanie nowego artykułu prasowego przez pracownika klubu

Atrybut	Opis
Nazwa scenariusza	Dodanie nowego artykułu prasowego przez pracownika klubu
Abstrakt	Pracownik klubu dodaje nowy artykuł do sekcji z wiadomościami klubowymi
Aktorzy	Pracownik klubu, system
Warunki początkowe	• Pracownik jest zalogowany i ma uprawnienia do edycji treści
Warunki końcowe	• Nowy artykuł jest widoczny w sekcji „Artykuły prasowe” dla wszystkich użytkowników
Flow	
Podstawowy	1. Pracownik loguje się do systemu 2. Przechodzi do sekcji „Artykuły prasowe” 3. Wybiera opcję „Dodaj nowy artykuł” 4. Wypełnia formularz artykułu (tytuł, treść, ewentualnie obraz) 5. Zatwierdza dodanie 6. System zapisuje artykuł i publikuje go
Alternatywny	4a. Pracownik zapisuje artykuł jako szkic - artykuł nie jest jeszcze widoczny dla użytkowników
Wyjątkowy	4b. Brak wymaganego pola (np. tytułu) - system wyświetla błąd walidacji i nie pozwala zatwierdzić formularza

Notatki	
Problem	Rozwiązanie
Brak automatycznego zapisu wersji roboczej	Implementacja autozapisu lub przypomnienia o zapisaniu szkicu

Tabela 4.8 Dodanie materiału do galerii przez pracownika klubu

Atrybut	Opis
Nazwa scenariusza	Dodanie materiału do galerii przez pracownika klubu
Abstrakt	Pracownik klubu dodaje nowe multimedia (zdjęcia lub wideo) do sekcji galerii dostępnej dla wszystkich użytkowników.
Aktorzy	Pracownik klubu, system
Warunki początkowe	• Pracownik jest zalogowany i posiada odpowiednie uprawnienia • Galeria zawiera już wcześniejsze multimedia
Warunki końcowe	• Nowe multimedia są widoczne w galerii
Flow	
Podstawowy	1. Pracownik loguje się do systemu 2. Przechodzi do sekcji „Galeria” 3. Wybiera opcję „Dodaj multimedia” 4. Wybiera plik z lokalnego urządzenia 5. Dodaje tytuł i opcjonalny opis 6. Zatwierdza przesyłanie 7. System zapisuje plik i wyświetla go w galerii
Alternatywny	4a. Pracownik dodaje wiele plików jednocześnie - system w ustalonej kolejności przesyła je do galerii
Wyjątkowy	4b. Pracownik wybiera plik w nieobsługiwanym formacie - system odrzuca plik i wyświetla komunikat o błędzie
Notatki	

Problem	Rozwiązanie
Brak informacji o dopuszczalnych formatach	Dodanie listy akceptowanych formatów przed wyborem pliku

Tabela 4.9. Edycja wyniku meczu przez pracownika klubu

Atrybut	Opis
Nazwa scenariusza	Edycja wyniku meczu przez pracownika klubu
Abstrakt	Pracownik klubu edytuje dane meczu, uzupełniając wynik oraz oznaczając zawodnika meczu.
Aktorzy	Pracownik klubu, system
Warunki początkowe	• Pracownik jest zalogowany • Dane dotyczące meczu są już wprowadzone w systemie
Warunki końcowe	• Zaktualizowany wynik i zawodnik meczu są widoczni w sekcji „Wyniki drużyny”
Flow	
Podstawowy	1. Pracownik loguje się do systemu 2. Przechodzi do sekcji „Wyniki drużyny” 3. Wybiera mecz z listy 4. Wprowadza wynik spotkania i wskazuje zawodnika meczu 5. Zatwierdza zmiany 6. System zapisuje zmodyfikowane dane i aktualizuje widok dla użytkowników
Alternatywny	4a. Pracownik edytuje również miejsce rozegrania meczu - system zapisuje wszystkie zmiany
Wyjątkowy	4b. Brak wskazania zawodnika meczu - system wyświetla ostrzeżenie i wymaga uzupełnienia pola

Notatki	
Problem	Rozwiązanie
Możliwość przypadkowej edycji danych	Dodanie mechanizmu potwierdzenia zmian

Scenariusze przypadków użycia administratora

W tabelach 4.10. – 4.12. przedstawiono scenariusze przypadków użycia administratora, które pokazują korzystanie z funkcjonalności zarządzania systemem przeznaczonej do pełnej kontroli nad systemem.

Tabela 4.10. Dodanie wydarzenia do osi czasu przez administratora

Atrybut	Opis
Nazwa scenariusza	Dodanie wydarzenia do osi czasu przez administratora
Abstrakt	Administrator dodaje nowe wydarzenie do osi czasu prezentującej najważniejsze momenty w historii klubu.
Aktorzy	Administrator, system
Warunki początkowe	• Administrator jest zalogowany i posiada uprawnienia do edycji osi czasu
Warunki końcowe	• Nowe wydarzenie jest widoczne dla wszystkich użytkowników w sekcji „Historia klubu”
Flow	
Podstawowy	1. Administrator loguje się do systemu 2. Przechodzi do sekcji „Historia klubu” 3. Wybiera opcję „Dodaj wydarzenie” 4. Wypełnia formularz (tytuł wydarzenia, data, opis, opcjonalnie grafika) 5. Zatwierdza dodanie wydarzenia 6. System zapisuje wydarzenie i umieszcza je na osi czasu
Alternatywny	4a. Administrator dodaje wydarzenie z przyszłą datą - system akceptuje i oznacza je jako „nadchodzące”
Wyjątkowy	4b. Administrator nie uzupełnia wymaganych pól - system wyświetla komunikat o błędach i nie pozwala zapisać wydarzenia
Notatki	
Problem	Rozwiązanie
Brak chronologicznego sortowania wydarzeń	Automatyczne sortowanie osi czasu według daty

Tabela 4.11. Zarządzanie użytkownikami przez administratora

Atrybut	Opis
Nazwa scenariusza	Zarządzanie użytkownikami przez administratora
Abstrakt	Administrator zarządza kontami użytkowników poprzez ich blokowanie lub trwałe usuwanie z systemu.
Aktorzy	Administrator, system
Warunki początkowe	• Administrator jest zalogowany i posiada uprawnienia administracyjne • W systemie istnieją konta użytkowników
Warunki końcowe	• Użytkownik zostaje zablokowany lub usunięty z systemu
Flow	
Podstawowy	1. Administrator loguje się do systemu 2. Przechodzi do „Panelu administracyjnego” 3. Wybiera sekcję „Użytkownicy” 4. Wyszukuje konkretnego użytkownika 5. Wybiera jedną z dostępnych akcji: „Zablokuj konto” lub „Usuń konto” 6. System przetwarza żądanie i wykonuje wybraną akcję 7. Administrator otrzymuje potwierdzenie wykonania operacji
Alternatywny	
Wyjątkowy	
Notatki	

Problem	Rozwiązanie
Ryzyko przypadkowego usunięcia konta	Wprowadzenie okna potwierdzającego decyzję przed wykonaniem operacji

Tabela 4.12. Rozpatrywanie prośby o usunięcie sponsora przez administratora

Atrybut	Opis
Nazwa scenariusza	Rozpatrywanie prośby o usunięcie sponsora.
Abstrakt	Administrator rozpatruje prośbę o usunięcie sponsora, która została wystosowana przez pracownika klubu.
Aktorzy	Administrator, pracownik klubu, system
Warunki początkowe	• Administrator jest zalogowany i posiada odpowiednie uprawnienia • Administrator otrzymał prośbę o usunięcie sponsora od pracownika klubu
Warunki końcowe	• Prośba zostają rozpatrzona pomyślnie, sponsor usunięty
Flow	
Podstawowy	1. Administrator loguje się do systemu 2. Przechodzi do „Panelu administracyjnego” 3. Wybiera sekcję „Sponsorzy” 4. Wybiera zakładkę „Prośby o usunięcie sponsora” 5. Wybiera konkretną prośbę z listy 6. Czyta treść prośby 7. Wybiera z dostępnych opcji pozycję „zatwierdź” 8. System przetwarza żądanie i wykonuje wybraną akcję 9. Administrator otrzymuje potwierdzenie wykonania operacji
Alternatywny	6a. Treść prośby jest pusta
Wyjątkowy	7a. Administrator wybiera opcję „odrzuć”, sponsor nie zostaje usunięty

Notatki	
Problem	Rozwiązanie
Możliwość wysłania prośby o usunięcie sponsora bez uzasadnienia	Wprowadzenie odpowiedniej walidacji pola z uzasadnieniem

4.5. Diagram klas

Diagram klas (*ang. Class Diagram*) to jeden z najważniejszych diagramów UML (*ang. Unified Modeling Language*), służący do modelowania statycznej struktury systemu. Przedstawia klasy, ich atrybuty, metody oraz powiązania pomiędzy nimi. Dzięki temu stanowi podstawę do projektowania architektury oprogramowania oraz odwzorowania logiki biznesowej. Poniższy rysunek przedstawia diagram klas dla projektowanego systemu.

Diagram klas

Na rysunku 4.5. znajduje się graficzna interpretacja diagramu klas dla projektowanego systemu, na którym znajdują się proponowane klasy, które zostaną użyte podczas implementacji założonych wcześniej funkcjonalności aplikacji.

Rysunek 4.5. Diagram klas

Opis klas

1. User - reprezentuje użytkownika systemu. Przechowuje dane logowania, informacje osobiste, rolę użytkownika i jego interakcje z systemem.
2. Article - reprezentuje artykuł prasowy publikowany przez klub. Zawiera tytuł, treść, informacje o autorze, dacie publikacji i statusie. Umożliwia zarządzanie treściami informacyjnymi.
3. Comment - reprezentuje komentarz dodany przez użytkownika do artykułu. Przechowuje treść komentarza, informacje o jego autorze oraz ewentualne powiązanie z innym komentarzem.
4. ArticleReaction - reprezentuje reakcję emocjonalną (np. polubienie) użytkownika na artykuł. Łączy użytkownika i artykuł z typem wyrażonej reakcji.
5. TeamMember - reprezentuje członka drużyny klubu siatkarskiego, zarówno zawodnika, jak i członka sztabu szkoleniowego. Zawiera dane osobowe, pozycję, numer i inne informacje.
6. Match - reprezentuje mecz siatkarski. Przechowuje informacje o dacie, miejscu, przeciwniku, wyniku oraz najlepszym graczu.
7. MatchPlayer - klasa asocjacyjna łącząca mecz (Match) z zawodnikiem (TeamMember), który w nim uczestniczył. Może przechowywać dodatkowe statystyki zawodnika z danego meczu.
8. GalleryItem - reprezentuje pojedynczy element w galerii multimedialnej klubu. Zawiera tytuł, opis, ścieżkę do pliku oraz informacje o typie i osobie dodającej.
9. ClubHistoryEvent - reprezentuje ważne wydarzenie z historii klubu, przeznaczone do wyświetlenia na osi czasu. Zawiera tytuł, datę, opis i opcjonalną grafikę.
10. Sponsor - reprezentuje sponsora klubu. Przechowuje nazwę, logo, link do strony internetowej oraz opcjonalny opis.
11. SponsorRemovalRequest - reprezentuje formalną prośbę pracownika klubu o usunięcie sponsora z listy. Przechowuje informacje o sponsorze, zgłaszającym, przyczynie i statusie prośby.
12. Survey - reprezentuje ankietę stworzoną w systemie. Zawiera pytanie ankietowe, informacje o twórcy, statusie aktywności oraz o tym, czy jest anonimowa.
13. SurveyOption - reprezentuje jedną z możliwych opcji odpowiedzi w ramach danej ankiety (Survey).
14. SurveyVote - reprezentuje pojedynczy głos oddany w ankiecie. Łączy ankietę (Survey), wybraną opcję (SurveyOption) oraz opcjonalnie głosującego użytkownika (User).
15. Product - reprezentuje produkt dostępny do zakupu w sklepie klubowym. Zawiera nazwę, opis, cenę, stan magazynowy, kategorię i zdjęcie.
16. Order - reprezentuje zamówienie złożone przez użytkownika w sklepie klubowym. Przechowuje informacje o zamawiającym, dacie, całkowitej kwocie, statusie oraz adresach.
17. OrderItem - reprezentuje pojedynczą pozycję w zamówieniu (Order). Łączy zamówienie z produktem (Product) i określa zamówioną ilość oraz cenę w momencie zakupu.

18. TicketEvent - reprezentuje wydarzenie, na które można zakupić bilety poprzez zewnętrzny system. Przechowuje nazwę, datę wydarzenia oraz link do platformy biletowej.
19. AdminMessage - reprezentuje wiadomość, zgłoszenie błędu lub uwagę wysłaną przez użytkownika do administratora systemu. Zawiera treść, temat, informacje o nadawcy i statusie zgłoszenia.

Opis relacji między klasami

1. User - Article

Definiuje, który użytkownik (User) jest autorem danego artykułu (Article). Umożliwia identyfikację twórcy treści i powiązanie artykułów z konkretnymi kontami użytkowników.

2. Article - Comment

Łączy artykuły (Article) z komentarzami (Comment) dodanymi przez użytkowników. Pozwala na wyświetlanie dyskusji pod konkretnymi publikacjami.

3. User - Comment

Określa, który użytkownik (User) napisał dany komentarz (Comment). Umożliwia identyfikację autora wypowiedzi.

4. Comment - Comment

Umożliwia tworzenie zagnieżdżonych dyskusji, gdzie jeden komentarz (Comment) może być bezpośrednią odpowiedzią na inny komentarz (Comment).

5. Article - ArticleReaction

Łączy artykuły (Article) z reakcjami emocjonalnymi (ArticleReaction, np. polubienia) pozostawionymi przez użytkowników. Pozwala na ocenę popularności treści.

6. User - ArticleReaction

Identyfikuje, który użytkownik (User) wyraził konkretną reakcję (ArticleReaction) na dany artykuł.

7. Match - TeamMember

Wskazuje, który członek drużyny (TeamMember) został wybrany jako najlepszy zawodnik danego meczu (Match).

8. Match - MatchPlayer

Łączy mecz (Match) z poszczególnymi wpisami o uczestnictwie graczy (MatchPlayer), tworząc skład na dany mecz.

9. TeamMember - MatchPlayer

Łączy konkretnego członka drużyny (TeamMember) z jego wpisem o uczestnictwie (MatchPlayer) w danym meczu.

10. User - GalleryItem

Określa, który użytkownik dodał dany element (GalleryItem) do galerii.

11. User - ClubHistoryEvent

Identyfikuje użytkownika, który dodał konkretne wydarzenie (ClubHistoryEvent) do osi czasu historii klubu.

12. User - Sponsor

Wskazuje, który użytkownik wprowadził dane sponsora (Sponsor) do systemu.

13. Sponsor - SponsorRemovalRequest

Łączy konkretnego sponsora (Sponsor) z prośbą (SponsorRemovalRequest) o jego usunięcie z listy sponsorów.

14. User - SponsorRemovalRequest

Identyfikuje użytkownika, który zainicjował prośbę (SponsorRemovalRequest) o usunięcie sponsora.

15. User - SponsorRemovalRequest

Wskazuje użytkownika, który podjął decyzję dotyczącą prośby (SponsorRemovalRequest) o usunięcie sponsora.

16. User - Survey

Określa, który użytkownik jest autorem danej ankiety (Survey).

17. Survey - SurveyOption

Łączy ankietę (Survey) z zestawem możliwych opcji odpowiedzi (SurveyOption).

18. Survey - SurveyVote

Powiązuje ankietę (Survey) z poszczególnymi głosami (SurveyVote) oddanymi przez użytkowników.

19. SurveyOption - SurveyVote

Wskazuje, która konkretna opcja odpowiedzi (SurveyOption) została wybrana w danym głosie (SurveyVote).

20. User - SurveyVote

Identyfikuje użytkownika (User), który oddał dany głos (SurveyVote) w ankiecie (jeśli ankieta nie jest anonimowa).

21. User - Order

Łączy użytkownika (User) ze złożonymi przez niego zamówieniami (Order) w sklepie klubowym.

22. Order - OrderItem

Łączy zamówienie (Order) z poszczególnymi produktami (OrderItem) wchodzącymi w jego skład, określając ich ilość i cenę w momencie zakupu.

23. Product - OrderItem

Wskazuje, który konkretny produkt (Product) jest częścią danej pozycji zamówienia (OrderItem).

24. User - TicketEvent

Określa, który użytkownik jest odpowiedzialny za zarządzanie informacjami o danym wydarzeniu biletowym (TicketEvent) i linkiem do sprzedaży.

25. User - AdminMessage

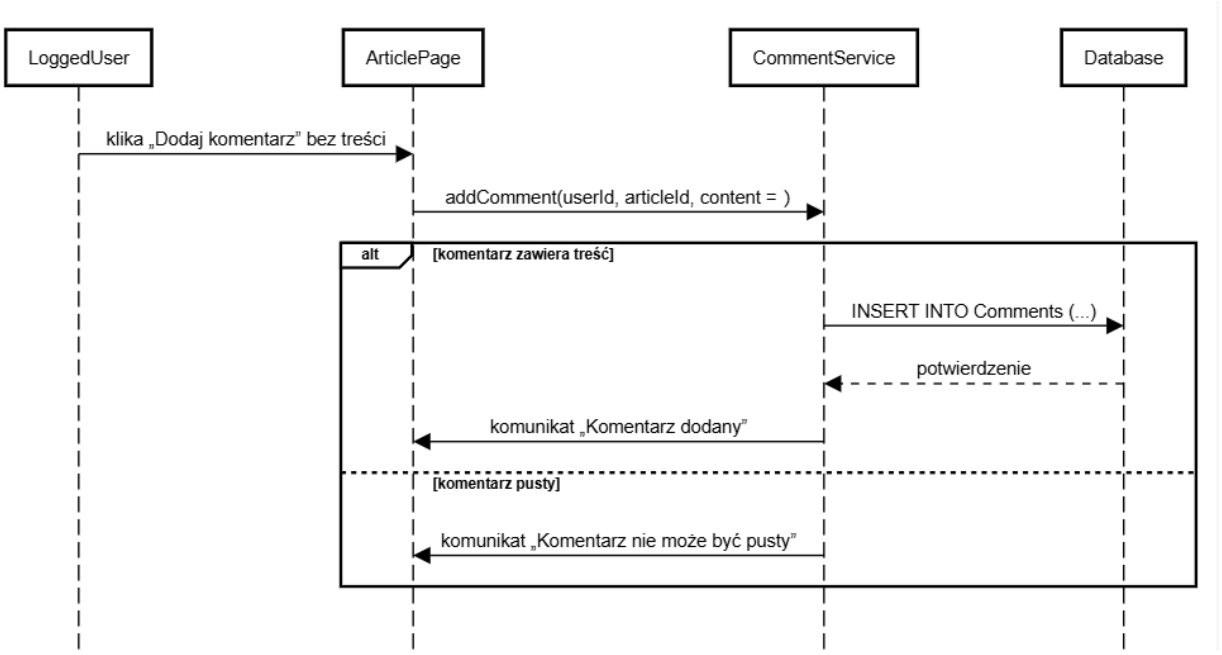
Identyfikuje użytkownika (User), który wysłał wiadomość (AdminMessage) do administratora systemu.

26. User - AdminMessage

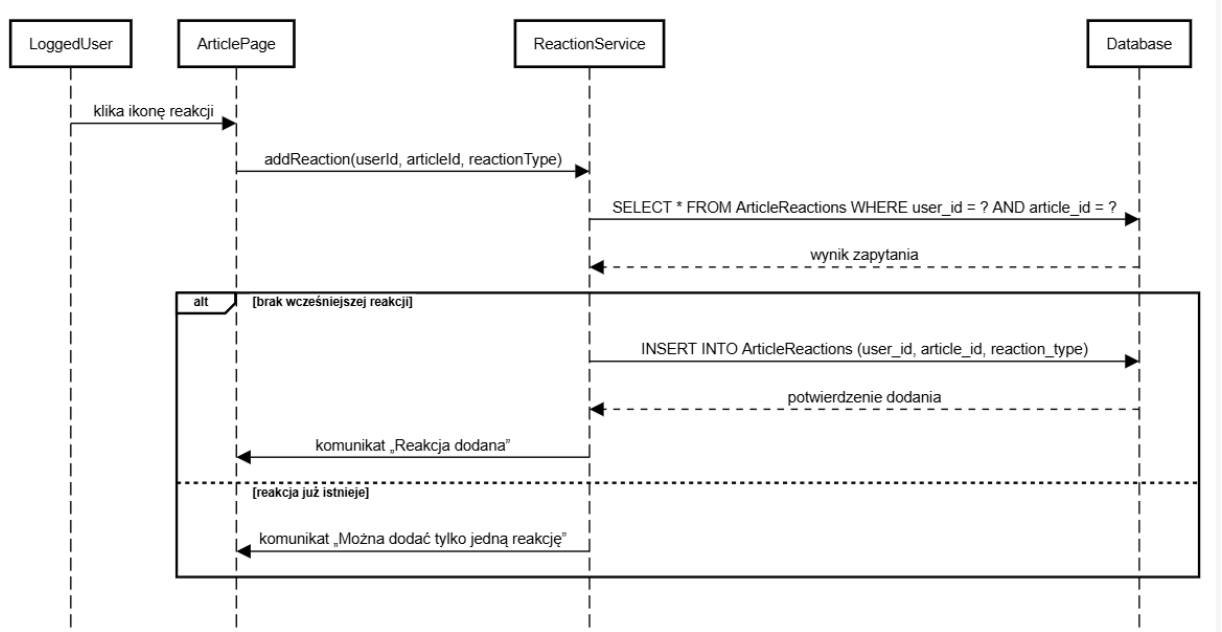
Wskazuje użytkownika, który rozpatrzył zgłoszenie lub wiadomość (AdminMessage).

4.6. Diagramy sekwencji

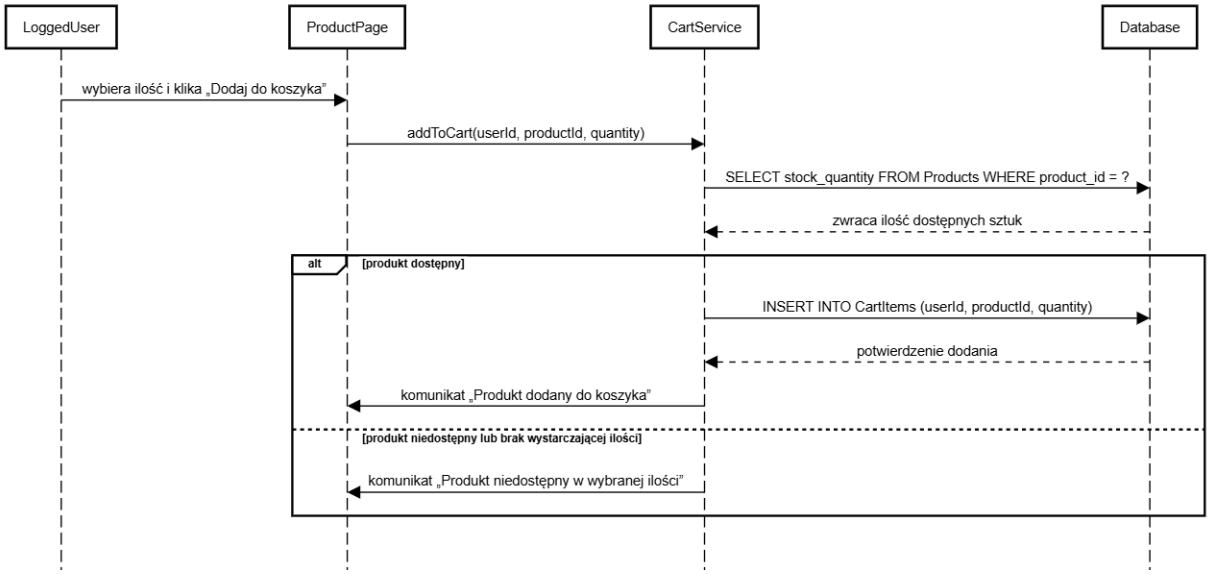
Diagram sekwencji (*ang. Sequence Diagram*) to jeden z diagramów UML służący do modelowania interakcji między obiektami lub komponentami systemu w ujęciu czasowym. Jego głównym celem jest pokazanie, w jaki sposób poszczególne elementy komunikują się ze sobą podczas realizacji określonego przypadku użycia. Diagramy sekwencji podczas projektowania aplikacji pełnią ważną rolę, ponieważ pozwalają lepiej zrozumieć przepływ logiki w danym scenariuszu. Na rysunkach 4.6. - 4.10. przedstawiono przykłady diagramów sekwencji dla każdej z ról użytkowników w projektowanym systemie.



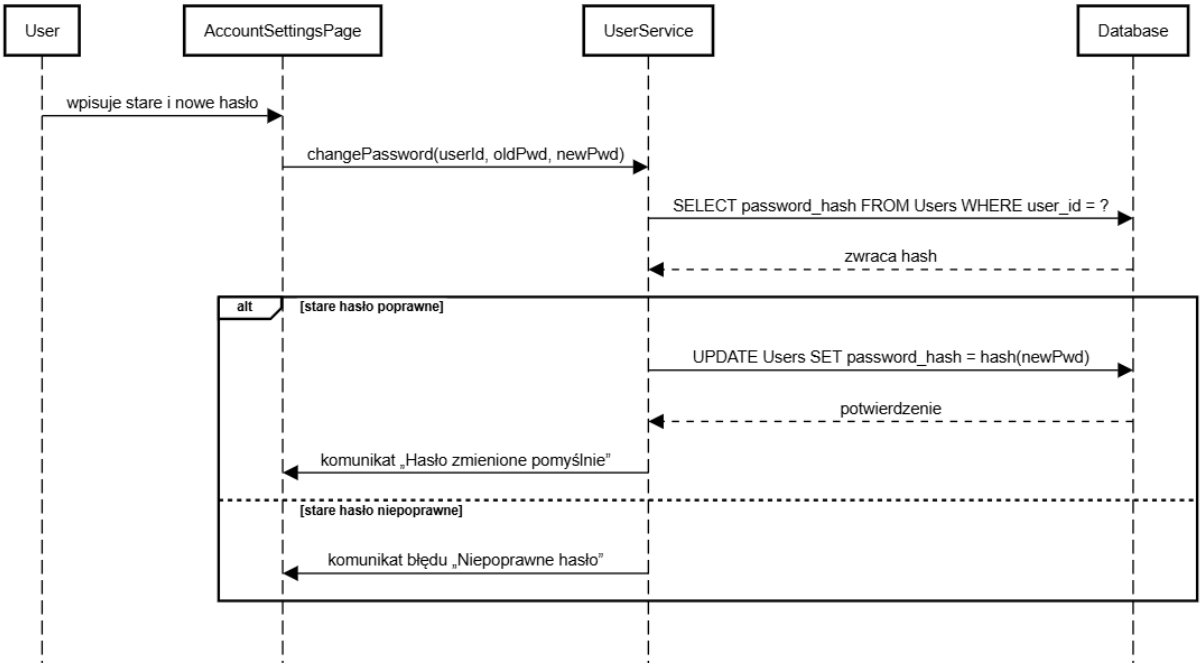
Rysunek 4.6. Dodawanie komentarza bez treści pod artykułem prasowym przez użytkownika zalogowanego



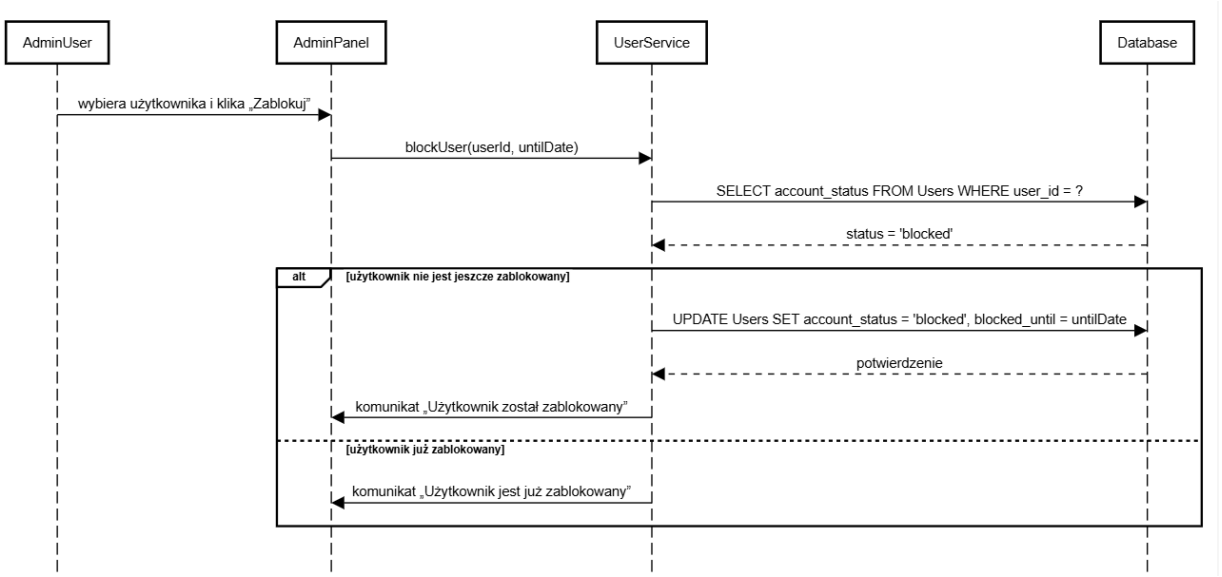
Rysunek 4.7. Dodanie reakcji do artykułu prasowego przez użytkownika zalogowanego



Rysunek 4.8. Dodanie produktu do koszyka przez użytkownika zalogowanego



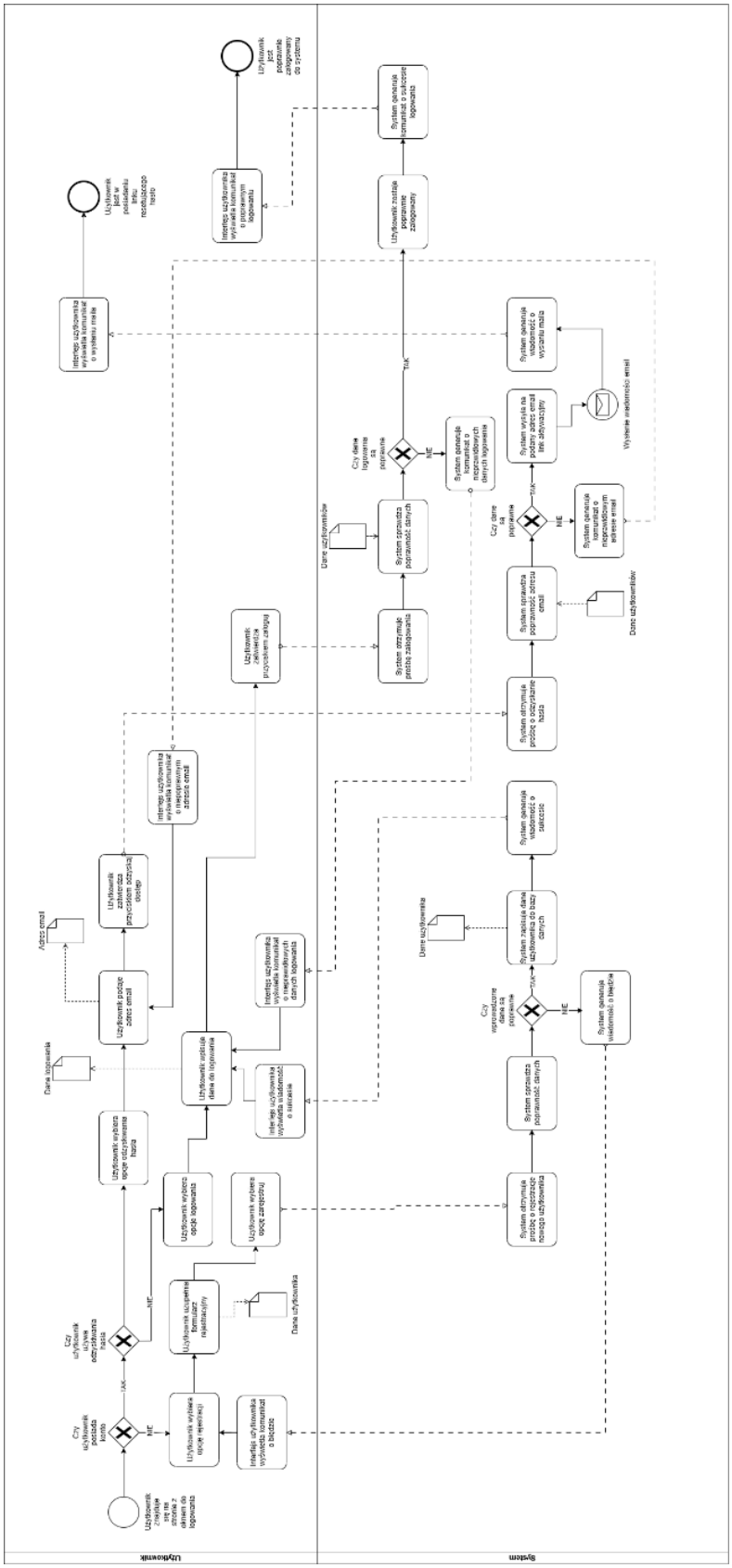
Rysunek 4.9. Zmiana hasła przez użytkownika zalogowanego



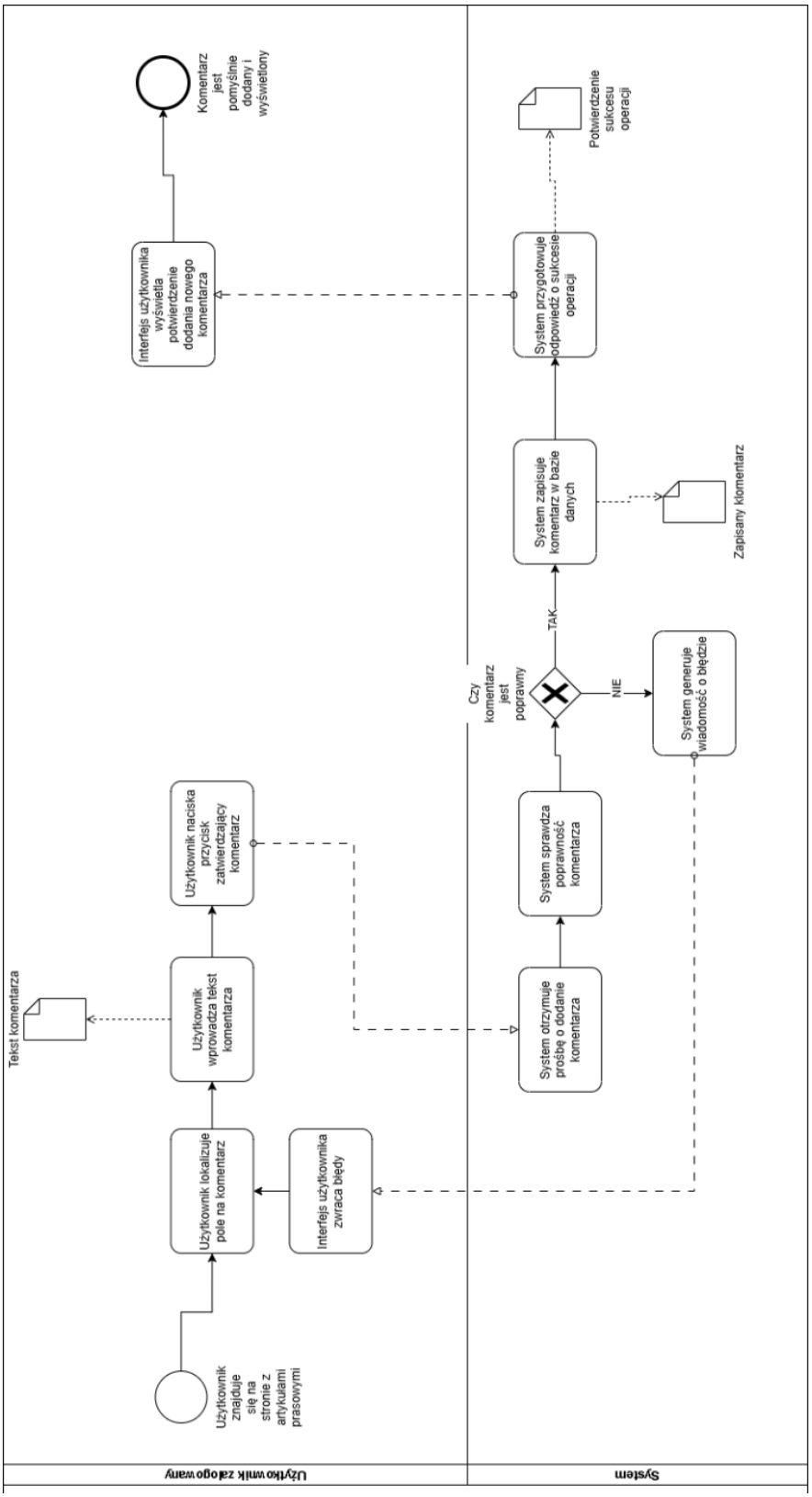
Rysunek 4.10. Blokowanie użytkownika przez administratora

4.7. Diagramy BPMN

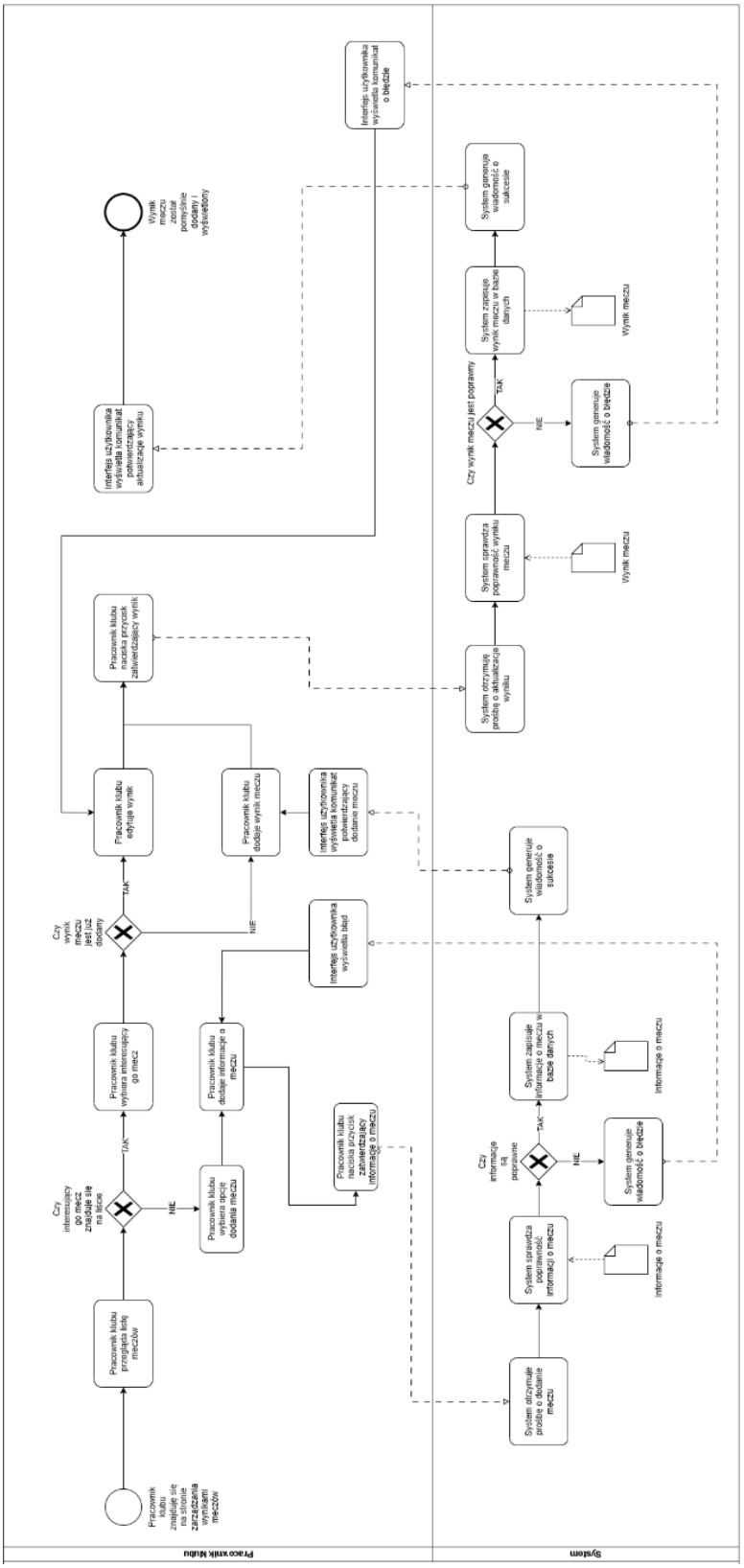
BPMN (ang. *Business Process Model and Notation*) jest standardem, który służy do modelowania procesów biznesowych. Zapewnia on wspólny język opisu procesów, który jest zrozumiały zarówna dla programistów jak i analityków biznesowych. Diagramy pozwalają przedstawić w przejrzysty sposób przedstawić przebieg procesu, jego uczestników, decyzje oraz interakcje z systemem. Na rysunkach 4.11. - 4.14. przedstawiono przykładowe diagramy BPMN dla każdej z ról użytkowników w projektowanym systemie.



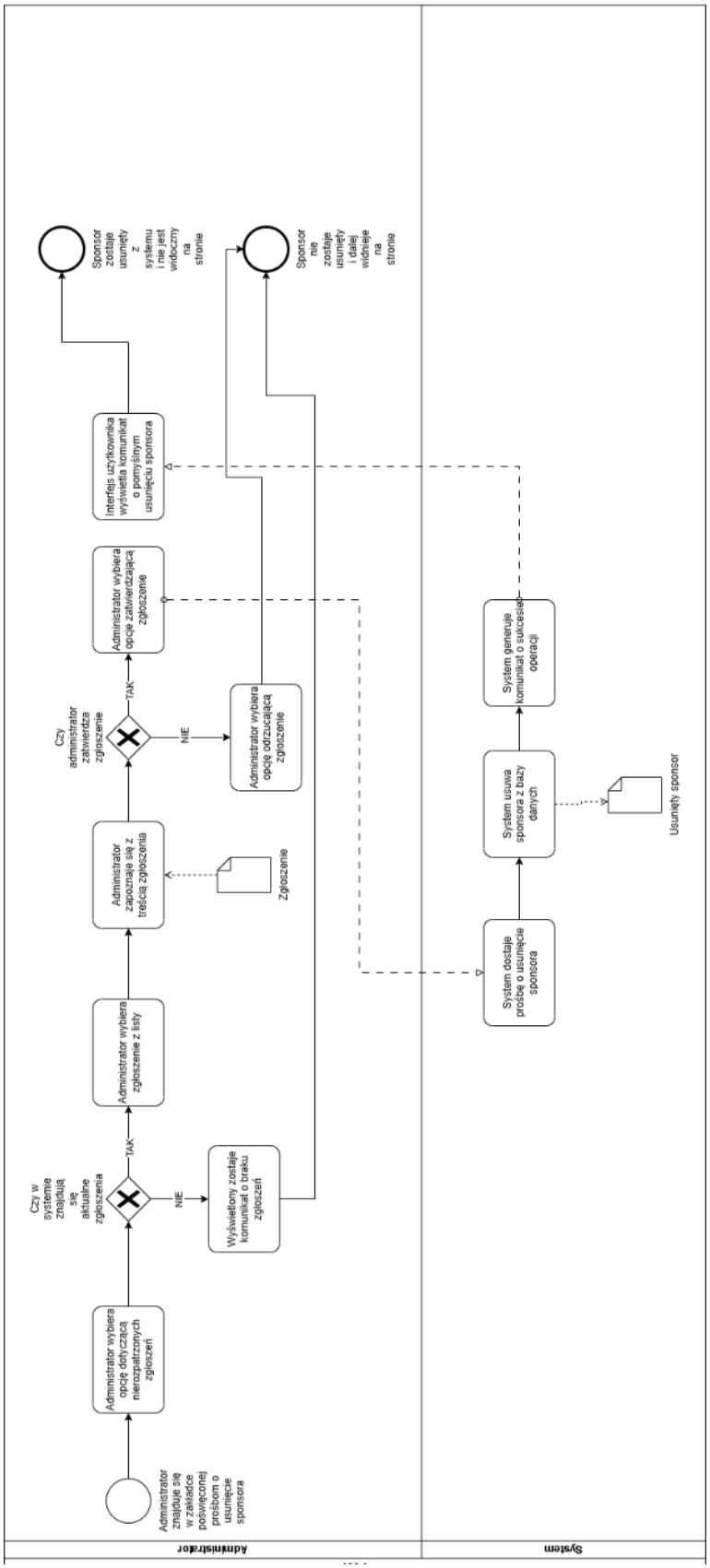
Rysunek 4.11. Logowanie do systemu



Rysunek 4.12. Dodanie komentarza przez użytkownika zalogowanego



Rysunek 4.13. Zarządzanie wynikami meczu przez pracownika klubu



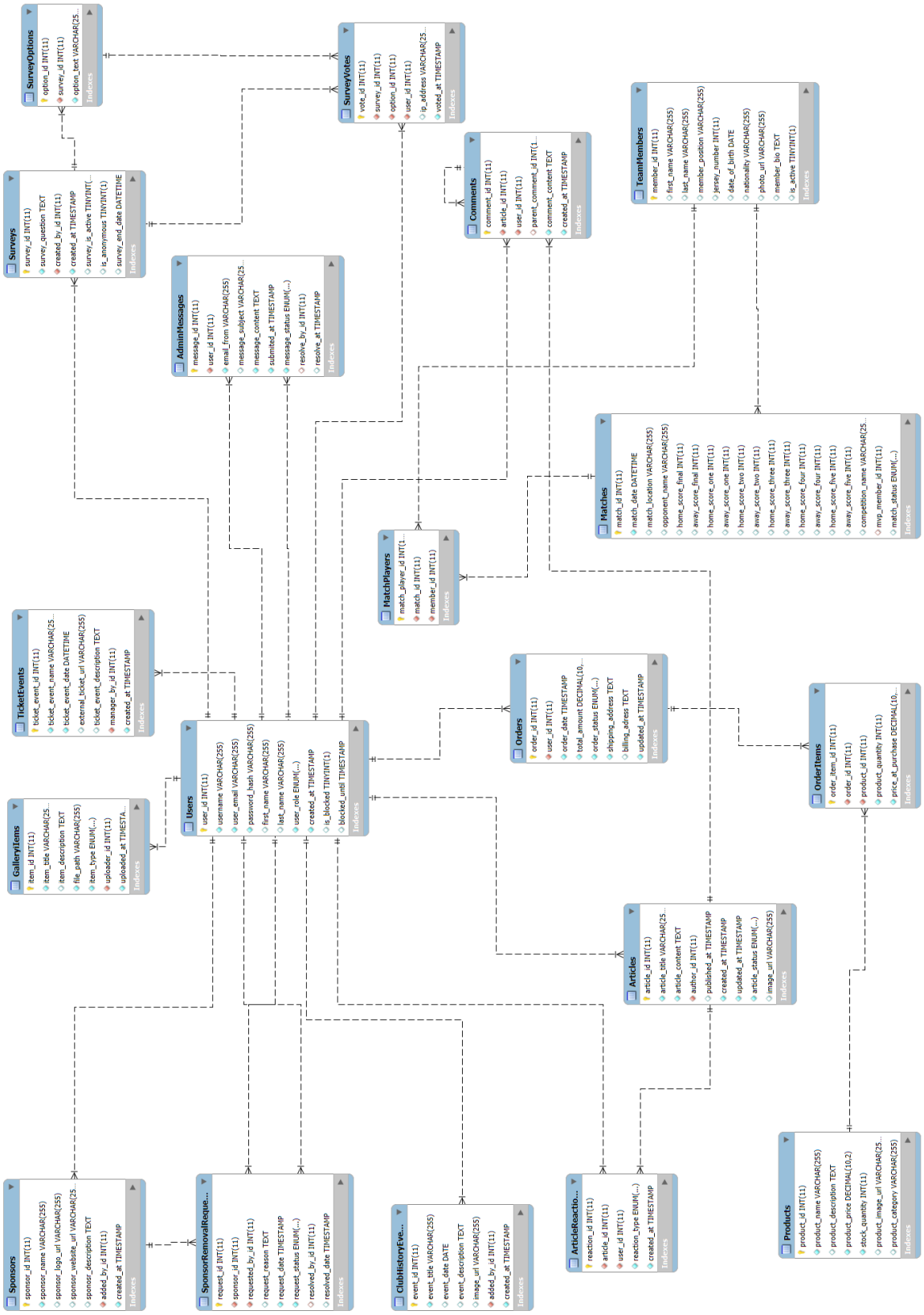
Rysunek 4.14. Rozpatrywanie prośby o usunięcie sponsora

4.8. Projekt bazy danych

Do przechowywania danych w projektowanej aplikacji zostanie użyta relacyjna baza danych MySQL[12]. Na rysunku 4.15. przedstawiono strukturę danych systemu. W kolejnych rozdziałach opisane zostały poszczególne tabele potrzebne do realizacji bazy danych.

Opis encji

W tabelach 4.13. – 4.31. opisano strukturę poszczególnych encji znajdujących się w projektowanej bazie danych. Znajduje się tam nazwa atrybutu, typ danych oraz krótki opis danego atrybutu.



Rysunek 4.15. Diagram ERD projektowanej bazy danych

Tabela 4.13. Encja Users

Nazwa	Typ	Opis
user_id	INT	Identyfikator
username	VARCHAR	Nazwa użytkownika
user_email	VARCHAR	Email użytkownika
password_hash	VARCHAR	Zaszyfrowane hasło
first_name	VARCHAR	Imię użytkownika
last_name	VARCHAR	Nazwisko użytkownika
user_role	ENUM	Rola użytkownika
created_at	TIMESTAMP	Data utworzenia konta
is_blocked	BOOLEAN	Czy konto jest zablokowane
blocked_until	TIMESTAMP	Data odblokowania konta

Tabela 4.14. Enacja Articles

Nazwa	Typ	Opis
article_id	INT	Identyfikator
article_title	VARCHAR	Tytuł artykułu prasowego
article_content	TEXT	Treść artkułu
author_id	INT	Identyfikator autora
published_at	TIMESTAMP	Data publikacji
created_at	TIMESTAMP	Data utworzenia
updated_at	TIMESTAMP	Data uaktualnienia
article_status	ENUM	Status artykułu
image_url	VARCHAR	Link do obrazu

Tabela 4.15. Encja Comments

Nazwa	Typ	Opis
comment_id	INT	Identyfikator komentarza
article_id	INT	Identyfikator artykułu
user_id	INT	Identyfikator użytkownika
parent_comment_id	INT	Komentarz nadrzędny
comment_content	TEXT	Treść komentarza
created_at	TIMESTAMP	Data utworzenia

Tabela 4.16. Encja ArticleReactions

Nazwa	Typ	Opis
reaction_id	INT	Identyfikator reakcji
article_id	INT	Identyfikator artykułu
user_id	INT	Identyfikator użytkownika
reaction_type	ENUM	Typ reakcji
created_at	TIMESTAMP	Data dodania reakcji

Tabela 4.17. Encja TeamMembers

Nazwa	Typ	Opis
member_id	INT	Identyfikator członka
first_name	VARCHAR	Imię
last_name	VARCHAR	Nazwisko
member_position	VARCHAR	Pozycja w drużynie
jersey_number	INT	Numer koszulki
date_of_birth	DATE	Data urodzenia
nationality	VARCHAR	Narodowość
photo_url	VARCHAR	Link do zdjęcia
member_bio	TEXT	Nota biograficzna
is_active	BOOLEAN	Czy jest obecnie członkiem

Tabela 4.18. Encja: Matches

Nazwa	Typ	Opis
match_id	INT	Identyfikator meczu
match_date	DATETIME	Data meczu
match_location	VARCHAR	Miejsce rozgrywek
opponent_name	VARCHAR	Nazwa przeciwnika
home_score_final	INT	Wynik końcowy gospodarz
away_score_final	INT	Wynik końcowy gość
home_score_one	INT	Gospodarze pierwszy set
away_score_one	INT	Goście pierwszy set
home_score_two	INT	Gospodarze drugi set
away_score_two	INT	Goście drugi set
home_score_three	INT	Gospodarze trzeci set
away_score_three	INT	Goście trzeci set
home_score_four	INT	Gospodarze czwarty set
away_score_four	INT	Goście czwarty set
home_score_five	INT	Gospodarze piąty set
away_score_five	INT	Goście piąty set
competition_name	VARCHAR	Nazwa rozgrywek
mvp_member_id	INT	Identyfikator zawodnika
match_status	ENUM	Status meczu

Tabela 4.19. Encja MatchPlayers

Nazwa	Typ	Opis
match_player_id	INT	Identyfikator
match_id	INT	Identyfikator meczu
member_id	INT	Identyfikator zawodnika

Tabela 4.20. GalleryItems

Nazwa	Typ	Opis
item_id	INT	Identyfikator elementu
item_title	VARCHAR	Tytuł elementu
item_description	TEXT	Opis elementu
file_path	VARCHAR	Ścieżka do pliku
item_type	ENUM	Typ elementu
uploader_id	INT	Identyfikator dodającego
uploaded_at	TIMESTAMP	Czas dodania elementu

Tabela 4.21. ClubHistoryEvents

Nazwa	Typ	Opis
event_id	INT	Identyfikator wydarzenia
event_title	VARCHAR	Nazwa wydarzenia
event_date	DATE	Data wydarzenia
event_description	TEXT	Opis wydarzenia
image_url	VARCHAR	Ścieżka do grafiki
added_by_id	INT	Identyfikator dodającego
created_at	TIMESTAMP	Czas dodania wpisu

Tabela 4.22. Sponsors

Nazwa	Typ	Opis
sponsor_id	INT	Identyfikator sponsora
sponsor_name	VARCHAR	Nazwa sponsora
sponsor_logo_url	VARCHAR	Ścieżka do logo
sponsor_website_url	VARCHAR	Link do strony
sponosr_description	TEXT	Opis sponsora
added_by_id	INT	Identyfikator dodającego
created_at	TIMESTAMP	Czas dodania sponsora

Tabela 4.23. SponsorRemovalRequests

Nazwa	Typ	Opis
request_id	INT	Identyfikator prośby
sponsor_id	INT	Identyfikator sponsora
requested_by_id	INT	Identyfikator zgłaszającego
request_reason	TEXT	Powód zgłoszenia
request_date	TIMESTAMP	Data zgłoszenia prośby
request_status	ENUM	Status prośby
resolved_by_id	INT	Identyfikator administratora
resolved_date	TIMESTAMP	Data rozpatrzenia prośby

Tabela 4.24. Surveys

Nazwa	Typ	Opis
survey_id	INT	Identyfikator ankiety
survey_question	TEXT	Pytanie ankiety
created_by_id	INT	Identyfikator tworzącego
created_at	TIMESTAMP	Czas utworzenia ankiety
survey_is_active	BOOLEAN	Czy ankieta jest aktywna
is_anonymous	BOOLEAN	Czy ankieta jest anonimowa
survey_end_date	DATETIME	Data zakończania ankiety

Tabela 4.25. SurveyOptions

Nazwa	Typ	Opis
option_id	INT	Identyfikator opcji
survey_id	INT	Identyfikator ankiety
option_text	VARCHAR	Treść opcji odpowiedzi

Tabela 4.26. SurveyVotes

Nazwa	Typ	Opis
vote_id	INT	Identyfikator głosu
survey_id	INT	Identyfikator ankiety
option_id	INT	Identyfikator opcji
user_id	INT	Identyfikator użytkownika
ip_address	VARCHAR	Adres IP głosującego
voted_at	TIMESTAMP	Czas oddania głosu

Tabela 4.27. Products

Nazwa	Typ	Opis
product_id	INT	Identyfikator produktu
product_name	VARCHAR	Nazwa produktu
product_description	TEXT	Opis produktu
product_price	DECIMAL	Cena jednostkowa produktu
stock_quantity	INT	Ilość produktu w magazynie
product_image_url	VARCHAR	Ścieżka do zdjęcia
product_category	VARCHAR	Kategoria produktu

Tabela 4.28. Orders

Nazwa	Typ	Opis
order_id	INT	Identyfikator zamówienia
user_id	INT	Identyfikator użytkownika
order_date	TIMESTAMP	Data zamówienia
total_amount	DECIMAL	Całkowita kwota
order_status	ENUM	Status zamówienia
shipping_address	TEXT	Adres dostawy
billing_adress	TEXT	Adres do faktury
updated_at	TIMESTAMP	Data aktualizacji

Tabela 4.29. OrderItems

Nazwa	Typ	Opis
order_item_id	INT	Identyfikator
order_id	INT	Identyfikator zamówienia
product_id	INT	Identyfikator produktu
product_quantity	INT	Ilość produktu
price_at_purchase	DECIMAL	Cena w momencie zakupu

Tabela 4.30. TicketEvents

Nazwa	Typ	Opis
ticket_event_id	INT	Identyfikator wydarzenia
ticket_event_name	VARCHAR	Nazwa wydarzenia
ticket_event_date	DATETIME	Data wydarzenia
external_ticket_url	VARCHAR	Link do biletów
ticket_event_description	TEXT	Opis wydarzenia
manager_by_id	INT	Identyfikator zarządcy
created_at	TIMESTAMP	Czas utworzenia wpisu

Tabela 4.31. AdminMessages

Nazwa	Typ	Opis
message_id	INT	Identyfikator wiadomości
user_id	INT	Identyfikator użytkownika
email_from	VARCHAR	Email użytkownika
message_subject	VARCHAR	Temat wiadomości
message_content	TEXT	Treść wiadomości
submited_at	TIMESTAMP	Czas wysłania wiadomości
message_status	ENUM	Status wiadomości
resolve_by_id	INT	Identyfikator administratora
resolve_at	TIMESTAMP	Czas rozwiązania

Tabele są połączone następującymi relacjami:

Users - Articles: związek jeden-do-wielu (1:N), gdzie jeden użytkownik może być autorem wielu artykułów, ale każdy artykuł musi mieć przypisanego dokładnie jednego autora.

Users - Comments: związek jeden-do-wielu (1:N), jeden użytkownik może napisać wiele komentarzy, ale każdy komentarz jest przypisany do dokładnie jednego użytkownika.

Users - ArticleReactions: związek jeden-do-wielu (1:N), użytkownik może dodać wiele reakcji do artykułów, jednak dla jednego artykułu użytkownik może mieć tylko jedną reakcję danego typu.

Users - GalleryItems: związek jeden-do-wielu (1:N), gdzie jeden użytkownik (np. pracownik lub administrator) może dodać wiele elementów do galerii, każdy element musi mieć przypisanego jednego autora.

Users - ClubHistoryEvents: związek jeden-do-wielu (1:N), gdzie jeden użytkownik (administrator) może dodać wiele wydarzeń historycznych, ale każde wydarzenie historyczne przypisane jest do dokładnie jednego użytkownika, który je dodał.

Users - Sponosors: związek jeden-do-wielu (1:N), jeden użytkownik (pracownik lub administrator) może dodać wielu sponsorów, ale każdy sponsor jest dodany przez dokładnie jednego użytkownika.

Users - SponsorsRemovalRequests: związek wiele-do-wielu (M:N), ponieważ użytkownik (pracownik) może zgłosić wiele próśb o usunięcie sponsora, a inny użytkownik (administrator) może rozpatrywać wiele takich próśb. Każda prośba jest jednak powiązana z dokładnie jednym zgłaszającym oraz jednym rozpatrującym.

Users - Surveys: związek jeden-do-wielu (1:N), jeden użytkownik (pracownik lub administrator) może stworzyć wiele ankiet, ale każda ankieta ma dokładnie jednego autora.

Users - SurveyVotes: związek jeden-do-wielu (1:N), użytkownik może oddać wiele głosów w różnych ankietach, ale w ramach jednej ankiety użytkownik może oddać tylko jeden głos.

Users - Orders: związek jeden-do-wielu (1:N), jeden użytkownik może złożyć wiele zamówień, ale każde zamówienie jest przypisane do dokładnie jednego użytkownika.

Users - TicketEvents: związek jeden-do-wielu (1:N), jeden użytkownik (administrator) może zarządzać wieloma wydarzeniami biletowymi, każde wydarzenie jest przypisane do dokładnie jednego administratora.

Users - AdminMessages: związek wiele-do-wielu (M:N), ponieważ użytkownik może wysłać wiele wiadomości do administratora, a administrator może rozwiązać wiele wiadomości. Każda wiadomość ma przypisanego jednego autora oraz ewentualnie jednego rozwiązującego.

Articles - Comments: związek jeden-do-wielu (1:N), jeden artykuł może mieć wiele komentarzy, ale każdy komentarz jest przypisany do dokładnie jednego artykułu.

Articles - ArticlesReactions: związek jeden-do-wielu (1:N), jeden artykuł może posiadać wiele reakcji od różnych użytkowników, ale każda reakcja odnosi się do dokładnie jednego artykułu.

Comments - Comments: relacja samoodwołująca (self-join), jeden komentarz może być odpowiedzią na inny komentarz, ale każdy komentarz będący odpowiedzią wskazuje na dokładnie jeden komentarz nadrzędny.

TeamMembers - Matches: związek jeden-do-wielu (1:N), jeden członek drużyny może zostać wyróżniony jako MVP w wielu meczach, ale w danym meczu MVP przypisany jest do jednego zawodnika.

TeamMembers - MatchPlayers: związek jeden-do-wielu (1:N), jeden członek drużyny może brać udział w wielu meczach, ale każdy wpis w tabeli udziału w meczu (MatchPlayers) dotyczy dokładnie jednego zawodnika.

Matches - MatchPlayers: związek jeden-do-wielu (1:N), jeden mecz może mieć wielu zawodników przypisanych do składu, ale każdy wpis w tabeli MatchPlayers odnosi się do dokładnie jednego meczu.

Surveys - SurveyOptions: związek jeden-do-wielu (1:N), jedna ankieta może posiadać wiele możliwych opcji odpowiedzi, ale każda opcja odpowiedzi należy do dokładnie jednej ankiety.

Survey - SurveyVotes: związek jeden-do-wielu (1:N), w jednej ankiecie może zostać oddanych wiele głosów, ale każdy głos musi być przypisany do dokładnie jednej ankiety.

SurveyOptions - SurveyVotes: związek jeden-do-wielu (1:N), jedna opcja odpowiedzi może być wybrana wielokrotnie przez różnych użytkowników, ale każdy głos wskazuje dokładnie jedną opcję.

Products - OrderItems: związek jeden-do-wielu (1:N), jeden produkt może występować w wielu pozycjach zamówienia, ale każda pozycja zamówienia odnosi się do dokładnie jednego produktu.

Orders - OrderItems: związek jeden-do-wielu (1:N), jedno zamówienie może składać się z wielu pozycji (produktów), ale każda pozycja zamówienia przypisana jest do dokładnie jednego zamówienia.

Sponsors - SponsorRemovalRequests: związek jeden-do-wielu (1:N), jeden sponsor może być przedmiotem wielu próśb o usunięcie (choć zazwyczaj istnieje tylko jedna aktywna), ale każda prośba dotyczy dokładnie jednego sponsora.

4.9. Projekt interfejsu graficznego

Zaprojektowanie widoków użytkownika pełni ważną rolę podczas planowania wyglądu aplikacji, skupiając się na doświadczeniach użytkownika. Głównym celem interfejsu graficznego jest zapewnienie prostego oraz intuicyjnego poruszania się po aplikacji. Poniżej przedstawiono przykładowe szkice interfejsu użytkownika, pokazujące koncepcyjny wygląd projektowanego systemu.

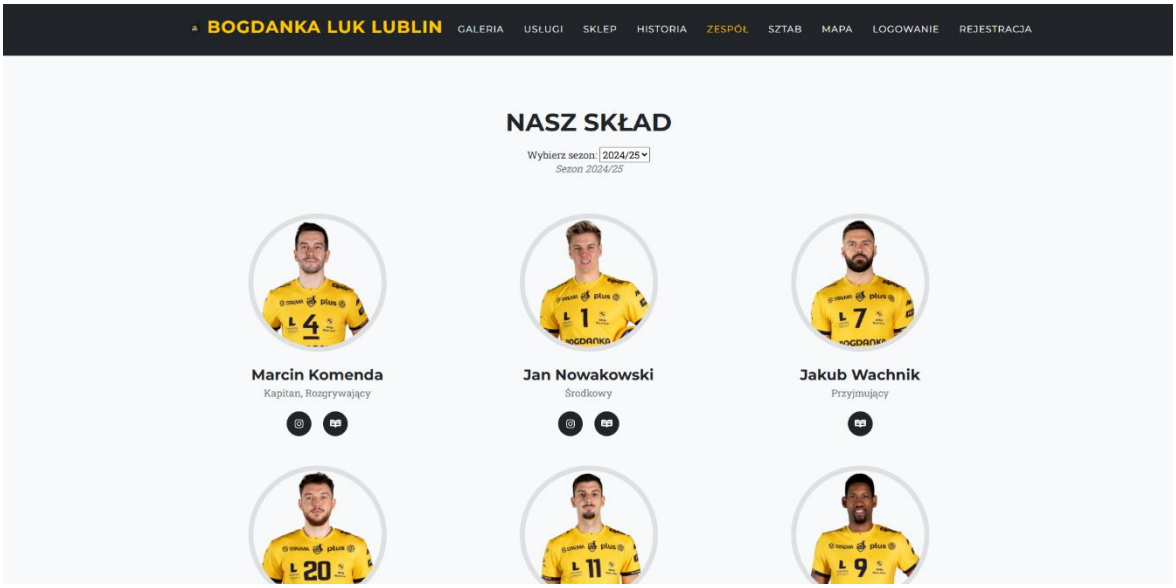
Rysunek 4.16. przedstawia ekran rejestracji nowego użytkownika do systemu. Musi on podać niezbędne dane, aby uzyskać dostęp do wszystkich funkcjonalności użytkownika zalogowanego.

Rysunek 4.16. Rejestracja nowego użytkownika

Rysunek 4.17. przedstawia oś czasu z najważniejszymi wydarzeniami w historii klubu, którą może przeglądać użytkownik niezalogowany.

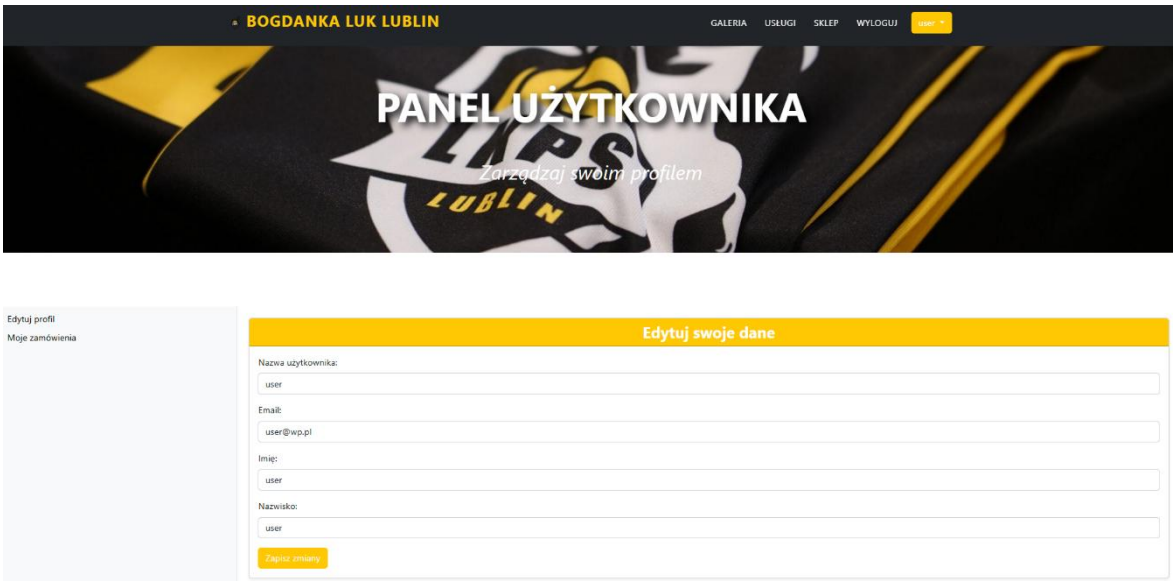
Rysunek 4.17. Przeglądanie osi czasu przez użytkownika niezalogowanego

Rysunek 4.18. przedstawia ekran umożliwiający użytkownikowi niezalogowanemu na przeglądanie składu drużyny. Może on sprawdzić oficjalne statystyki lub media społecznościowe poszczególnych graczy.



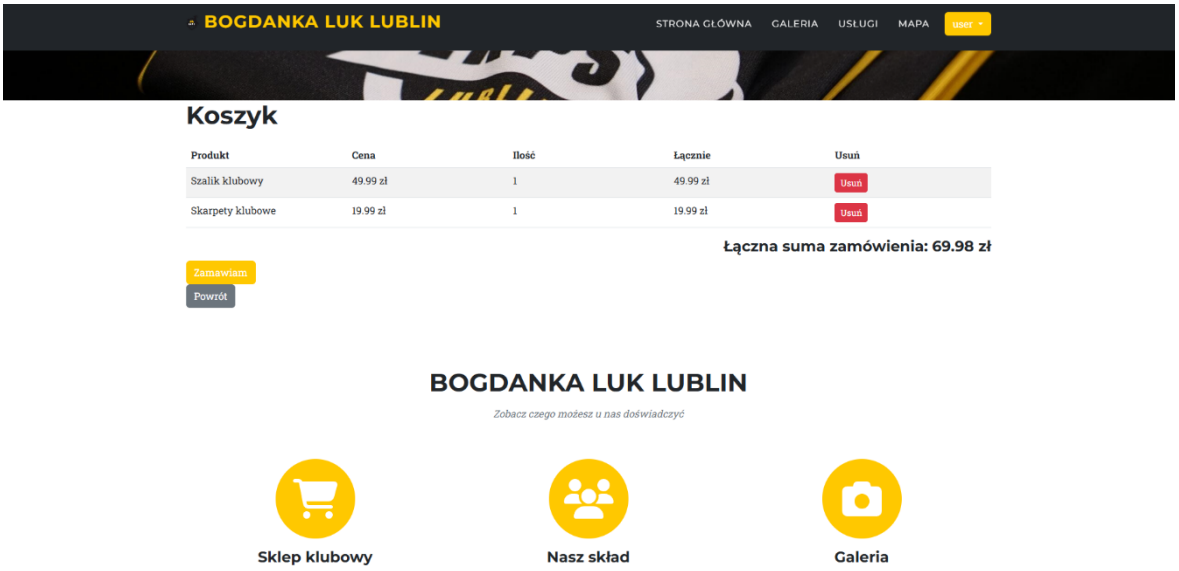
Rysunek 4.18. Przeglądanie składu drużyny przez użytkownika niezalogowanego

Rysunek 4.19 przedstawia ekran umożliwiający użytkownikowi modyfikacji danych osobowych zapisanych w systemie.



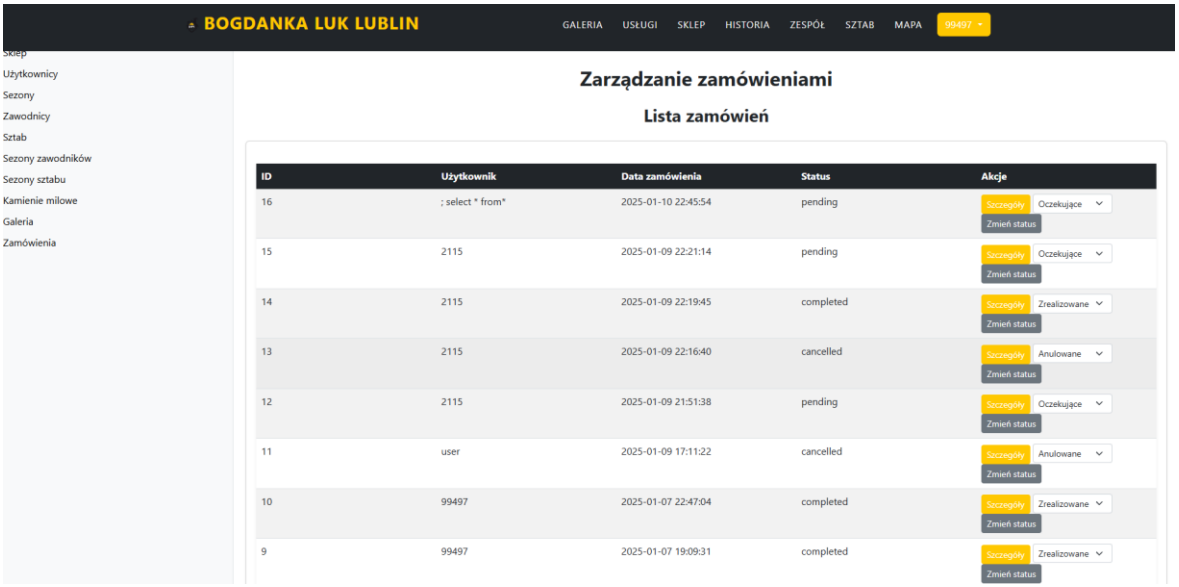
Rysunek 4.19. Modyfikacja danych przez użytkownika zalogowanego

Rysunek 4.20. przedstawia ekran składania zamówienia w sklepie klubowym przez użytkownika, który ma konto w systemie i jest zalogowany. Może on dodać wiele produktów do swojego koszyka a następnie sfinalizować zamówienie.



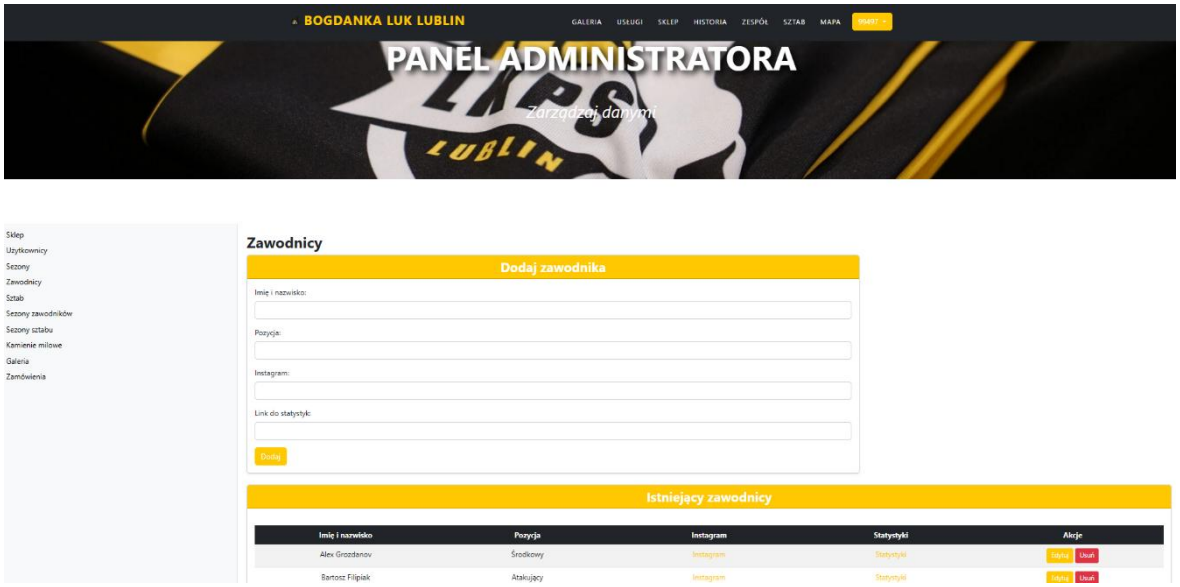
Rysunek 4.20. Składanie zamówienia na produkty klubowe przez użytkownika zalogowanego

Rysunek 4.21. przedstawia ekran umożliwiający pracownikom klubu na zarządzanie zamówieniami przychodzącymi od użytkowników zalogowanych.



Rysunek 4.21. Zarządzanie zamówieniami użytkowników przez pracownika klubu

Rysunek 4.22. przedstawia ekran umożliwiający administratorowi zarządzanie danymi znajdującymi się w systemie.



Rysunek 4.22. Panel administratora

5. Implementacja

W tym rozdziale przedstawiona została implementacja systemu informatycznego składająca się z dwóch aplikacji:

- aplikacji serwerowej, zapewniającej autoryzację użytkowników, realizacje procesów biznesowych oraz obsługę bazy danych;
- aplikacji klienckiej, zapewniającej graficzny interfejs dla użytkowników systemu, korzystających z przeglądarki internetowej.

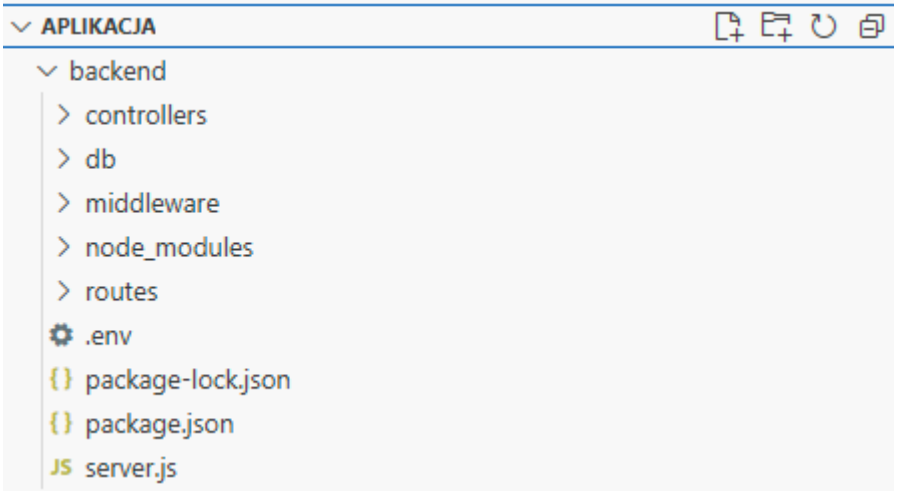
Utworzona aplikacja bazuje na języku JavaScript z wykorzystaniem różnych szkieletów programistycznych oraz bibliotek. Do utworzenia części serwerowej wykorzystano szkielet programistyczny Express w wersji 5.1.0. Do wyprodukowania części klienckiej zastosowano bibliotekę React w wersji 19.2.0.

5.1. Implementacja części serwerowej

Struktura projektu

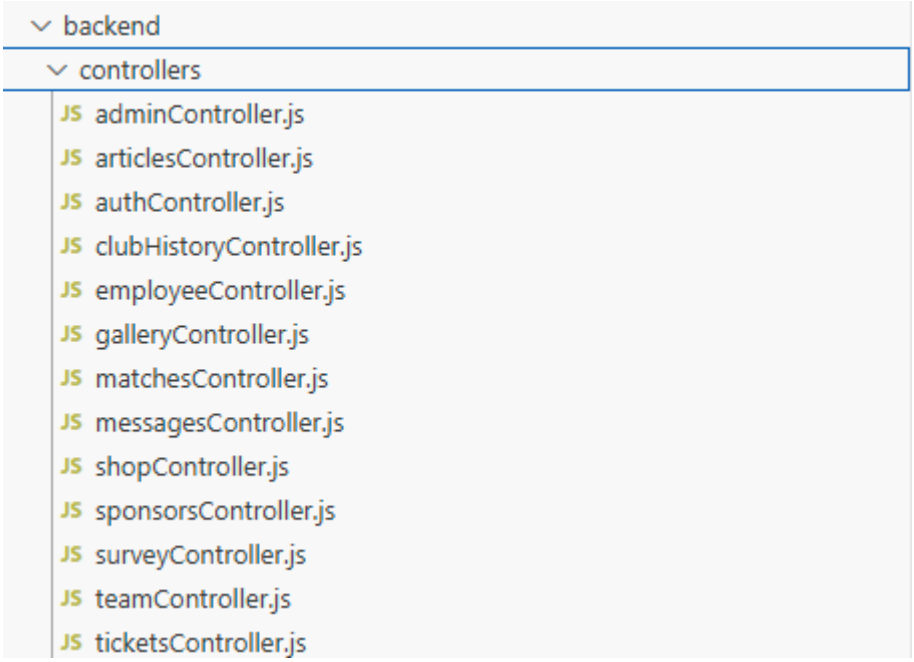
Opisana struktura katalogów jest typowym rozwiązaniem dla aplikacji Node.js z zastosowanym szkieletem programistycznym Express. Separacja kodu na kontrolery, middleware, trasy i konfigurację bazy danych ułatwia utrzymanie projektu oraz testowanie poszczególnych elementów.

Rysunek 5.1. przedstawia strukturę katalogów w aplikacji serwerowej, gdzie można wyszczególnić poszczególne foldery oraz pliki, odpowiadające za konkretne elementy systemu.



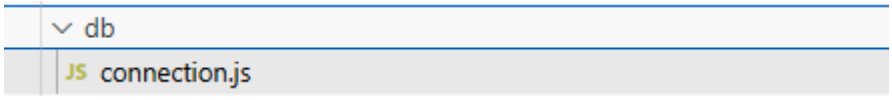
Rysunek 5.1. Struktura projektu, część serwerowa

Katalog **controllers** (Rysunek 5.2.) – znajduje się tam kod odpowiadający za obsługę żądań HTTP, pochodzących z aplikacji klienckiej. Kontrolery pośredniczą w wymianie danych między użytkownikiem systemu a bazą danych. Każdy z kontrolerów zawiera funkcje, które przetwarzają dane z części klienckiej i zwracają odpowiednie wyniki. Przykładowo, gdy użytkownik chce dodać nowy mecz, żądanie trafia do odpowiedniego kontrolera, który przetwarza te dane i wysyła je do bazy danych.



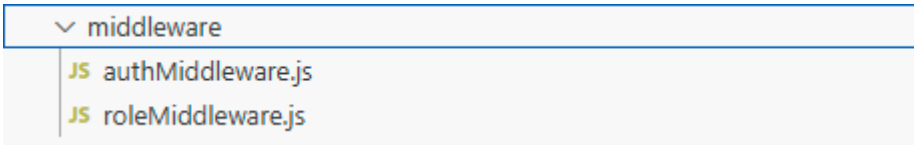
Rysunek 5.2 Zawartość katalogu controllers

Katalog **db** (Rysunek 5.3.) – odpowiada z przechowywanie plików odpowiedzialnych za poprawne połączenie aplikacji z bazą danych MySQL.



Rysunek 5.3. Zawartość katalogu db

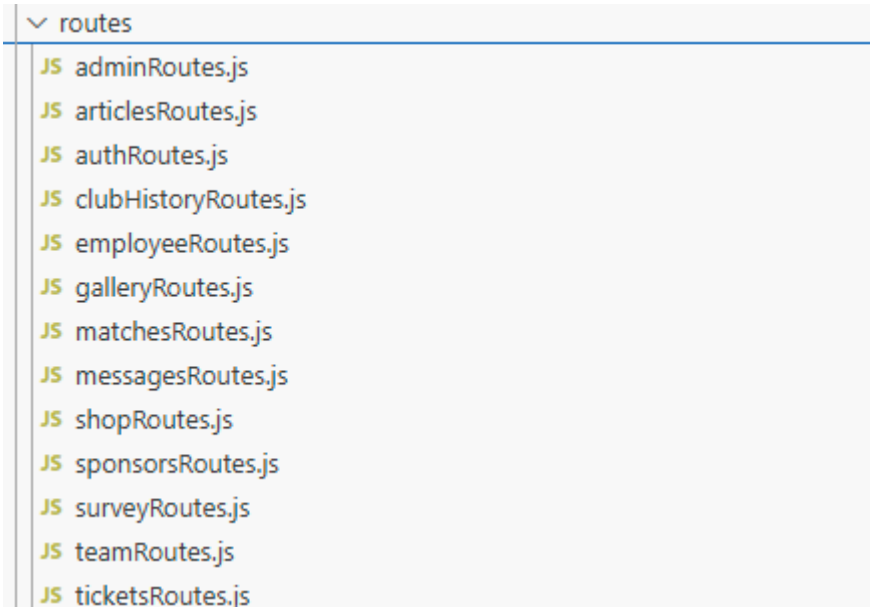
Katalog **middleware** (Rysunek 5.4.) – zawiera funkcje, które przetwarzają żądania HTTP przed dotarciem do kontrolerów. Middleware [2] może np. weryfikować, czy użytkownik jest zalogowany, czy ma odpowiednie uprawnienia, lub sprawdzać poprawność przesłanych danych. Dzięki middleware aplikacja jest bezpieczniejsza i bardziej niezawodna.



Rysunek 5.4. Zawartość katalogu middleware

Katalog **node_modules** – przechowywane są tam wszystkie biblioteki zewnętrzne wykorzystywane w projekcje. Katalog jest generowany automatycznie podczas tworzenia projektu na podstawie pliku **package.json**.

Katalog **routes** (Rysunek 5.5.) – przechowuje pliki w których zdefiniowane zostały trasy w postaci ścieżek URL. Są one odpowiedzialne za definiowanie punktów końcowych aplikacji.



Rysunek 5.5. Zawartość katalogu routes

Plik **.env** - zawiera zmienne konfiguracyjne wrażliwe na bezpieczeństwo, takie jak hasła do bazy danych, tokeny dostępu czy inne dane, które nie powinny być widoczne w kodzie źródłowym.

Pliki **package-lock.json** i **package.json** - manifest projektu, w którym zapisane są wszystkie wymagane biblioteki (zależności) oraz wersje Node.js. Plik **package-lock.json** zawiera dokładne informacje o zainstalowanych wersjach wszystkich pakietów. Dzięki tym plikom każda osoba pracująca nad projektem może zainstalować dokładnie te same wersje bibliotek.

Plik **server.js** (Listing 5.1.) – stanowi punkt wejścia aplikacji serwerowej i odpowiada za inicjalizację oraz konfigurację serwera Express. Na wstępie kod importuje wszystkie niezbędne biblioteki, w tym szkielet programistyczny Express do obsługi żądań HTTP, middleware CORS do umożliwienia komunikacji z aplikacją kliencką, moduł dotenv do załadowania zmiennych środowiskowych oraz wszystkie moduły tras (routes) odpowiadające poszczególnym funkcjom aplikacji, takie jak autentykacja, zarządzanie meczami, artykułami, galerią, biletami czy sklepem.

Następnie aplikacja wczytuje zmienne konfiguracyjne z pliku .env, w którym przechowywane są wrażliwe dane takie jak tajny klucz JWT (JSON Web Token) czy dane dostępu do bazy danych. Kod wypisuje również komunikaty diagnostyczne w konsoli, aby zweryfikować poprawne załadowanie konfiguracji.

Serwer jest następnie konfigurowany za pomocą middleware: CORS umożliwia obsługę żądań pochodzących z aplikacji klienckiej uruchomionej na porcie 3000, natomiast middleware express.json() pozwala na prawidłowe parsowanie przychodzących żądań w formacie JSON. Wszystkie trasy są następnie przypisane do odpowiednich ścieżek URL w strukturze /api/, które kierują żądania HTTP do właściwych kontrolerów.

Listing 5.1. Kod zawarty w pliku server.js

backend > JS server.js > ...

```
import express from "express";
import cors from "cors";
import dotenv from "dotenv";
import authRoutes from "./routes/authRoutes.js";
import teamRoutes from './routes/teamRoutes.js';
import clubHistoryRoutes from './routes/clubHistoryRoutes.js';
import articlesRoutes from "./routes/articlesRoutes.js";
import galleryRoutes from "./routes/galleryRoutes.js";
import ticketsRoutes from "./routes/ticketsRoutes.js";
import surveyRoutes from "./routes/surveyRoutes.js";
import sponsorsRoutes from "./routes/sponsorsRoutes.js";
import matchesRoutes from "./routes/matchesRoutes.js";
import shopRoutes from "./routes/shopRoutes.js";
import messagesRoutes from "./routes/messagesRoutes.js";
import adminRoutes from "./routes/adminRoutes.js";
import employeeRoutes from "./routes/employeeRoutes.js";

dotenv.config();

console.log("JWT_SECRET:", process.env.JWT_SECRET ? "ZAŁADOWANY" : "BRAK");
console.log("DB_NAME:", process.env.DB_NAME);
console.log(" PORT:", process.env.PORT);

const app = express();

app.use(cors({
  origin: "http://localhost:3000",
  credentials: true
}));

app.use(express.json());

app.use("/api/auth", authRoutes);
app.use("/api/clubhistory", clubHistoryRoutes);
app.use("/api/articles", articlesRoutes);
app.use("/api/gallery", galleryRoutes);
app.use("/api/tickets", ticketsRoutes);
app.use("/api/surveys", surveyRoutes);
app.use("/api/sponsors", sponsorsRoutes);
app.use("/api/matches", matchesRoutes);
app.use("/api/shop", shopRoutes);
app.use("/api/messages", messagesRoutes);
app.use("/api/admin", adminRoutes);
app.use("/api/team", teamRoutes);
app.use("/api/employee", employeeRoutes);
app.use("/api/sponsor-removal-requests", sponsorsRoutes);

app.get("/", (req, res) => {
  res.send("API działa");
});

const PORT = process.env.PORT || 4000;
app.listen(PORT, () => console.log(`Server działa na porcie ${PORT}`));
```

Na koniec aplikacja nasłuchuje na porcie określonym w zmiennych środowiskowych (domyślnie 4000) i wypisuje potwierdzający komunikat. Plik server.js jest zatem niezbędnym elementem infrastruktury aplikacji, bez którego komunikacja między aplikacją kliencką a bazą danych byłaby niemożliwa.

Kontrolery

Stanowią kluczowy element architektury aplikacji serwerowej opartej na szkielecie programistycznym Express, pełniąc rolę pośrednika między żądaniami HTTP przychodzącymi z aplikacji klienckiej a operacjami na bazie danych. Każdy kontroler zawiera funkcje odpowiadające konkretnym operacjom biznesowym, takie jak pobieranie danych, ich walidacja, przetwarzanie oraz zwracanie wyników do klienta. Odpowiednia organizacja i implementacja kontrolerów zapewnia czytelność kodu, łatwość jego utrzymania oraz możliwość efektywnego rozszerzania aplikacji o nowe funkcjonalności.

W prezentowanej aplikacji występują cztery role użytkowników w systemie, które cechuje rożny stopień uprawnień oraz odmienne możliwości. Ich uprawnienia zostały zdefiniowane w rozdziale dotyczącym wymagań funkcjonalnych. Są to:

- użytkownik niezalogowany – osoba, która nie założyła jeszcze konta. Posiada dostęp do podstawowych treści.
- użytkownik zalogowany – osoba, która posiada konto w serwisie, może korzystać ze wszystkich możliwości, jakie daje strona użytkownikom zewnętrznym.
- pracownik klubu – osoba zatrudniona przez klub, która zajmuje się poszczególnymi elementami w mediach klubowych, takimi jak zdjęcia, artykuły prasowe czy badanie opinii fanów.
- administrator – osoba o najwyższych uprawnieniach w systemie, upoważniona do nadzorowania pracy innych użytkowników oraz kontroli nad newralgicznymi danymi.

W dalszej części tego rozdziału zaprezentowano implementacje najbardziej istotnych funkcji potrzebnych do segmentu aplikacji związanego z drużyną, które znajdują się w pliku **teamController.js**. Jest to pobieranie aktywnych członków drużyny, pobieranie informacji o konkretnych członkach drużyny oraz dodawanie nowych członków drużyny.

Uprawnienia do edycji sekcji drużyny ma jedynie administrator, aby pracownik klubu nie mógł omyłkowo usunąć zawodników mających ważny kontrakt z klubem, który jest prawnym właścicielem ich wizerunku. Mechanizm ten może zapobiegać błędom.

Trasy

Trasy stanowią kluczowy element architektury aplikacji serwerowej opartej na szkielecie programistycznym Express, odpowiadając za mapowanie żądań HTTP na odpowiadające im kontrolery oraz definiując strukturę interfejsu aplikacyjnego (API). Każda trasa określa metodę HTTP (GET, POST, PUT, DELETE), ścieżkę URL oraz kontroler, który powinien obsługiwać dane żądanie, umożliwiając w ten sposób kierowanie ruchu sieciowego do właściwych funkcji. Odpowiednia organizacja i strukturyzacja tras zapewnia przejrzystość logiki routingu, ułatwia

utrzymanie kodu, a także umożliwia systematyczne i efektywne rozszerzanie aplikacji o nowe punkty dostępu API.

Na listingu 5.2. przedstawiono plik **teamRoutes.js**, który definiuje trasy dla elementów aplikacji związanych z drużyną. Na wstępie kod importuje szkielet programistyczny Express oraz wszystkie kontrolery odpowiadające operacjom na danych drużyny, a także middleware autoryzacyjne odpowiadające za weryfikację tożsamości użytkownika i sprawdzenie jego uprawnień dostępu.

Kod tworzy router Express, który następnie mapuje poszczególne metody HTTP na odpowiadające im kontrolery i endpointy (punkty dostępu API). Trasy publiczne, dostępne dla wszystkich użytkowników, obejmują pobranie listy wszystkich aktywnych członków drużyny (metoda GET na /) oraz pobranie szczegółów konkretnego członka (metoda GET na /:id). Te operacje nie wymagają uwierzytelnienia ani specjalnych uprawnień, ponieważ udostępniają jedynie informacje publiczne o drużynie.

Trasy chronione, dostępne wyłącznie dla użytkowników posiadających odpowiednie uprawnienia, obejmują dodanie nowego członka (metoda POST na /), aktualizację danych istniejącego członka (metoda PUT na /:id) oraz usunięcie członka drużyny (metoda DELETE na /:id). Każda z tych tras poprzedzona jest middleware authMiddleware, które weryfikuje autentyczność użytkownika, oraz middleware checkRole, które sprawdza, czy użytkownik posiada wymagane role ('admin'). Taka struktura zabezpieczeń zapewnia, że wrażliwe operacje mogą być wykonywane wyłącznie przez autoryzowanych użytkowników.

Plik jest eksportowany jako moduł i następnie importowany w głównym pliku serwera (server.js), gdzie jest przypisywany do ścieżki /api/team, co powoduje, że wszystkie zdefiniowane trasy są dostępne pod adresami w formacie /api/team/....

Listing 5.2. Kod pliku teamRoutes.js

backend > routes > JS teamRoutes.js > ...

```
1   import express from "express";
2   import {
3     getAllTeamMembers,
4     getTeamMemberById,
5     addTeamMember,
6     updateTeamMember,
7     deactivateTeamMember
8   } from "../controllers/teamController.js";
9   import authMiddleware, { checkRole } from "../middleware/authMiddleware.js";
10
11  const router = express.Router();
12
13  // Publiczne endpointy
14  router.get("/", getAllTeamMembers);
15  router.get("/:id", getTeamMemberById);
16
17  // Chronione endpointy (tylko dla administratorów)
18  router.post("/", authMiddleware, checkRole('admin'), addTeamMember);
19  router.put("/:id", authMiddleware, checkRole('admin'), updateTeamMember);
20  router.delete("/:id", authMiddleware, checkRole('admin'), deactivateTeamMember);
21
22  export default router;
```

W sekcji dotyczącej drużyny występuje pięć endpointów. Tabela 5.1. przedstawia ich charakterystykę.

Tabela 5.1.

Endpoint	Metoda HTTP	Rola	Opis
/api/team/	GET	wszyscy	pobieranie pełnej listy zawodników oraz sztabu szkoleniowego
/api/team/:id	GET	wszyscy	pobieranie szczegółów danego członka drużyny
/api/team/	POST	administrator	dodanie nowego zawodnika lub członka sztabu szkoleniowego
/api/team/:id	PUT	administrator	edycja danych istniejącego członka drużyny
/api/team/:id	DELETE	administrator	usunięcie członka drużyny

Listing 5.3. przedstawia funkcję **getAllTeamMembers**, która jest kontrolerem odpowiadającym za pobranie listy wszystkich aktywnych członków drużyny z bazy danych i odesłanie ich do aplikacji klienckiej. Funkcja akceptuje żądanie HTTP (req) oraz obiekt odpowiedzi (res) i działa w trybie asynchronicznym, co pozwala na oczekiwanie na rezultat zapytania do bazy danych bez blokowania wykonywania pozostałego kodu.

Na wstępie funkcja wysyła zapytanie SQL do bazy danych w celu pobrania wszystkich członków zespołu, których status isactive wynosi TRUE. Zapytanie pobiera szeroki zakres informacji o każdym członku, obejmując dane personalne, parametry fizyczne, numer koszulki, stanowisko w drużynie, datę dołączenia do klubu oraz linki do profili mediów społecznościowych. Rezultaty są sortowane najpierw według pozycji na boisku, następnie według numeru koszulki i nazwiska.

Po pomyślnym pobraniu danych funkcja dzieli członków drużyny na dwie grupy: zawodników (posiadających numer koszulki) i sztab szkoleniowy (bez przydzielonego numeru). Następnie funkcja zwraca odpowiedź w formacie JSON zawierającą obie grupy oraz całkowitą liczbę pobieranych osób.

Obsługa błędów wykorzystywana w aplikacji

W przypadku wystąpienia błędu podczas wykonania zapytania do bazy danych, funkcja przechwytuje wyjątek, wypisuje komunikat błędu w konsoli serwera i zwraca odpowiedź HTTP ze statusem 500 (błąd serwera) wraz z komunikatem informującym o niepowodzeniu operacji. Takie postępowanie zapewnia bezpieczną obsługę błędów oraz informuje stronę kliencką o problemach podczas pobierania danych.

Listing 5.3. Kod funkcji umożliwiającej pobieranie wszystkich aktywnych zawodników

```
// Pobranie wszystkich aktywnych członków drużyny
export async function getAllTeamMembers(req, res) {
  try {
    console.log("Pobieranie wszystkich członków drużyny");

    const [rows] = await pool.query(
      `SELECT memberid, firstname, lastname, memberposition, jerseynumber,
              dateofbirth, nationality, height, weight, photourl, memberbio,
              socialinstagram, socialfacebook, socialtwitter, isactive, joindate
       FROM TeamMembers
       WHERE isactive = TRUE
       ORDER BY
        CASE
         WHEN memberposition IN ('Rozgrywający', 'Przyjmujący', 'Atakujący', 'Środkowy', 'Libero') THEN 1
         ELSE 2
        END,
        jerseynumber ASC, lastname ASC`
    );

    console.log(`Znaleziono ${rows.length} członków drużyny`);

    // Rozdziel na zawodników i sztab
    const players = rows.filter(member => member.jerseynumber !== null);
    const staff = rows.filter(member => member.jerseynumber === null);

    res.json({
      players,
      staff,
      total: rows.length
    });

  } catch (err) {
    console.error("Błąd podczas pobierania członków drużyny:", err);
    res.status(500).json({ message: "Błąd serwera podczas pobierania drużyny" });
  }
}
```

Funkcja **getTeamMemberById**, przedstawiona na listingu 5.4. jest kontrolerem odpowiadającym za pobranie szczegółowych informacji o konkretnym członku drużyny na podstawie jego identyfikatora. Funkcja akceptuje żądanie HTTP zawierające parametr ID w ścieżce URL oraz obiekt odpowiedzi, a następnie wykonuje asynchroniczne zapytanie do bazy danych w celu pobrania pełnych danych o wybranym członku zespołu.

Na wstępie funkcja pobiera identyfikator członka z parametrów żądania HTTP, a następnie wysyła zapytanie SQL do bazy danych. Zapytanie pobiera szczegółowe dane zawodnika lub sztabu szkoleniowego, obejmując informacje osobiste, parametry fizyczne, rolę w klubie, numer stroju oraz dane dotyczące mediów społecznościowych i daty dołączenia do drużyny. Zastosowanie sparametryzowanego zapytania (ze znakami zapytania) stanowi ważny mechanizm zabezpieczenia przed atakami typu SQL injection.

Po wykonaniu zapytania funkcja najpierw sprawdza, czy wynik zawiera rekordy. Jeśli członek drużyny o podanym identyfikatorze nie został znaleziony, funkcja zwraca odpowiedź HTTP ze statusem 404 (nie znaleziono) wraz z odpowiednim komunikatem. W przypadku pozytywnego rezultatu funkcja zwraca odpowiedź w formacie JSON zawierającą wszystkie szczegółowe informacje o członku drużyny.

W sytuacji, gdy wystąpi błąd jest on obsługiwany w standardowy sposób.

Listing 5.4. Kod umożliwiający pobieranie szczegółów o konkretnych członkach drużyny

```
// Pobranie szczegółów konkretnego członka drużyny
export async function getTeamMemberById(req, res) {
  try {
    const { id } = req.params;
    console.log(" Pobieranie szczegółów członka drużyny:", id);

    const [rows] = await pool.query(
      `SELECT memberid, firstname, lastname, memberposition, jerseynumber,
              dateofbirth, nationality, height, weight, photourl, memberbio,
              socialinstagram, socialfacebook, socialtwitter, isactive, joindate, createdat
       FROM TeamMembers
       WHERE memberid = ?`,
      [id]
    );

    if (rows.length === 0) {
      console.log(" Członek drużyny nie znaleziony");
      return res.status(404).json({ message: "Członek drużyny nie został znaleziony" });
    }

    console.log(" Znaleziono członka:", rows[0].firstname, rows[0].lastname);
    res.json(rows[0]);

  } catch (err) {
    console.error(" Błąd podczas pobierania szczegółów członka:", err);
    res.status(500).json({ message: "Błąd serwera podczas pobierania szczegółów" });
  }
}

```

Funkcja **addTeamMember** przedstawiona na listingu 5.5. jest kontrolerem odpowiadającym za dodanie nowego członka do drużyny do bazy danych i jest dostępna wyłącznie dla użytkowników posiadających odpowiednie uprawnienia (administratorów i pracowników klubu). Funkcja akceptuje żądanie HTTP zawierające dane nowego członka drużyny w formacie JSON oraz obiekt odpowiedzi, a następnie przeprowadza proces walidacji i zapisania danych do bazy.

Na wstępie funkcja pobiera z żądania HTTP wszystkie potrzebne informacje o nowym członku, obejmujące dane personalne, dane sportowe, parametry fizyczne, linki do zdjęcia profilowego oraz linki do kont w mediach społecznościowych. Następnie kod przeprowadza walidację wymaganych pól, sprawdzając czy zostały dostarczone wymagane dane. W przypadku braku któregokolwiek z wymaganych pól funkcja zwraca odpowiedź HTTP ze statusem 400 (nieprawidłowe żądanie) wraz z komunikatem opisującym, które pole jest obowiązkowe.

Po pozytywnym ukończeniu walidacji funkcja wysyła zapytanie SQL do bazy danych w celu wstawienia nowego rekordu. Zapytanie zawiera wszystkie dostarczone dane oraz automatycznie ustawia status aktywności na TRUE (członek jest aktywny) i bieżący znacznik czasowy (NOW()) jako datę utworzenia rekordu.

Po pomyślnym dodaniu członka drużyny funkcja zwraca odpowiedź HTTP ze statusem 201 (utworzono) wraz z komunikatem potwierdzającym operację oraz identyfikatorem nowo utworzonego rekordu, który pozwala aplikacji klienckiej identyfikować dodanego członka. W przypadku błędu podczas wykonania zapytania do bazy danych jest on obsługiwany w standardowy sposób.

Listing 5.5. Kod umożliwiający dodawanie członków drużyny przez użytkowników o odpowiednich uprawnieniach

```
// Dodanie nowego członka drużyny (tylko dla administratora/pracownika klubu)
export async function addTeamMember(req, res) {
  try {
    console.log("Dodawanie nowego członka drużyny");

    const {
      firstname, lastname, memberposition, jerseynumber,
      dateofbirth, nationality, height, weight, photourl,
      memberbio, socialinstagram, socialfacebook, socialtwitter, joindate
    } = req.body;

    // Walidacja wymaganych pól
    if (!firstname || !lastname || !memberposition || !dateofbirth || !nationality) {
      return res.status(400).json({
        message: "Wymagane pola: imię, nazwisko, pozycja, data urodzenia, narodowość"
      });
    }

    const [result] = await pool.query(
      `INSERT INTO TeamMembers
      (firstname, lastname, memberposition, jerseynumber, dateofbirth, nationality,
       height, weight, photourl, memberbio, socialinstagram, socialfacebook,
       socialtwitter, isactive, joindate, createdat)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, TRUE, ?, NOW())`,
      [firstname, lastname, memberposition, jerseynumber, dateofbirth, nationality,
       height, weight, photourl, memberbio, socialinstagram, socialfacebook,
       socialtwitter, joindate]
    );

    console.log("Dodano członka drużyny:", firstname, lastname);

    res.status(201).json({
      message: "Członek drużyny został dodany pomyślnie",
      memberid: result.insertId
    });

  } catch (err) {
    console.error("Błąd podczas dodawania członka drużyny:", err);
    res.status(500).json({ message: "Błąd serwera podczas dodawania członka" });
  }
}
```

Middleware

Middleware stanowią kluczowy element architektury aplikacji serwerowej opartej na szkielecie programistycznym Express, pełniąc rolę pośrednika przetwarzającego żądania HTTP przed dotarciem do kontrolerów oraz po ich obsłużeniu. Każde middleware zawiera funkcje odpowiadające konkretnym operacjom, takim jak weryfikacja autentyczności użytkownika, sprawdzenie uprawnień dostępu, walidacja danych wejściowych czy rejestrowanie zdarzeń. Odpowiednia organizacja i implementacja middleware zapewnia bezpieczeństwo aplikacji

a także umożliwia systematyczne i efektywne zarządzanie przepływem żądań HTTP w całej aplikacji.

Na listingu 5.6. przedstawiona została funkcja **authMiddleware** znajdująca się w pliku **authMiddleware.js**. Odpowiada ona za weryfikacje autentyczności użytkownika poprzez walidację tokenu JWT (JSON Web Token) oraz sprawdzenie statusu konta użytkownika w bazie danych. Middleware wykonywane jest przed dotarciem żądania do chronionych kontrolerów i stanowi pierwszą linię obrony bezpieczeństwa aplikacji.

Na wstępie middleware pobiera nagłówek autoryzacji z żądania HTTP, zawierający token uwierzytelniający w formacie "Bearer [token]". Jeśli nagłówek nie jest obecny lub nie rozpoczyna się od słowa kluczowego "Bearer", middleware natychmiast zwraca odpowiedź HTTP ze statusem 401 (brak autoryzacji) i kończy przetwarzanie żądania. Następnie kod pobiera sam token z nagłówka poprzez rozdzielenie ciągu znaków na podstawie rozdzielającej spacji i pobranie drugiej części tokentu w formie długiego ciągu znaków.

W bloku try middleware przeprowadza weryfikację tokenu przy użyciu tajnego klucza JWT_SECRET przechowywanego w zmiennych środowiskowych. Jeśli token jest ważny, dekodowany jest do obiektu zawierającego informacje o użytkowniku (takie jak identyfikator użytkownika). Następnie middleware wysyła zapytanie do bazy danych w celu potwierdzenia, że użytkownik o tym identyfikatorze nadal istnieje w systemie oraz sprawdza, czy jego konto nie zostało zablokowane. Jeśli użytkownik nie istnieje, middleware zwraca odpowiedź ze statusem 401, a jeśli jego konto jest zablokowane, zwraca odpowiedź ze statusem 403 (dostęp zakazany).

Jeśli wszystkie sprawdzenia przebiegną pomyślnie, middleware przypisuje zdekodowane dane użytkownika do obiektu żądania (req.user) i wywołuje funkcję next(), umożliwiającą żądaniu przejście do następnego middleware lub kontrolera. W bloku catch kod obsługuje błędy związane z tokenem, jeśli token wygasł, zwraca status 401 z odpowiednim komunikatem, a w przypadku innych błędów zwraca status 403 (token nieprawidłowy).

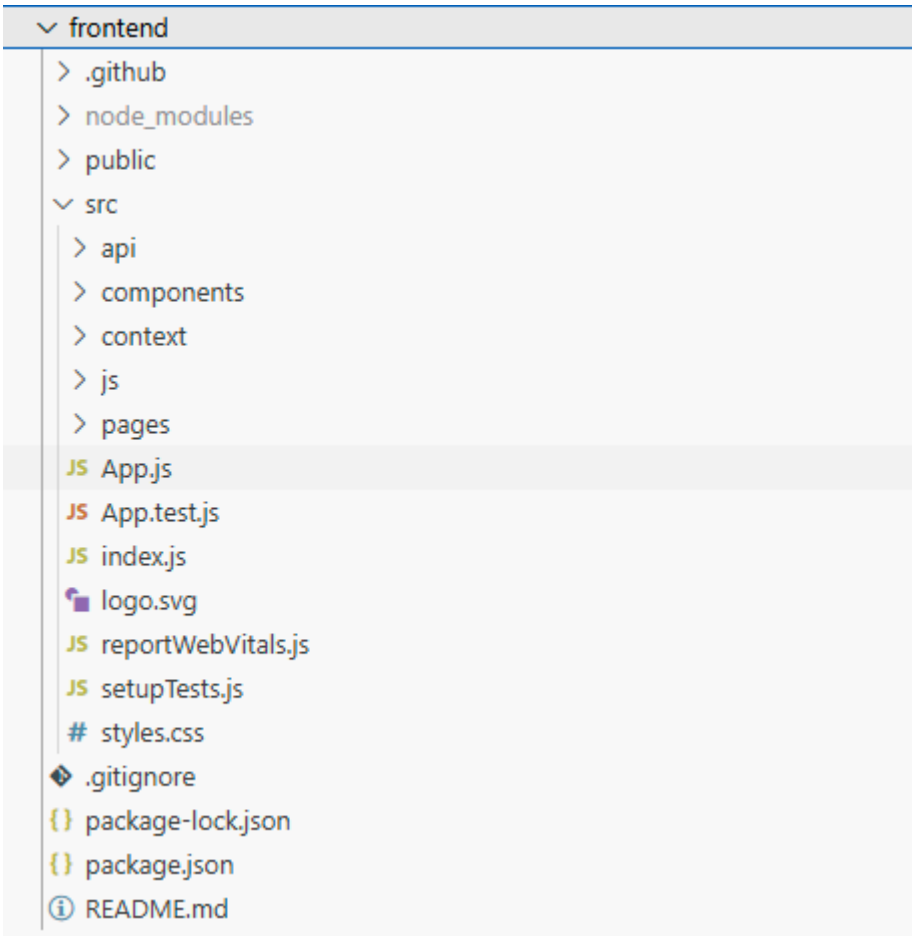
Listing 5.6. Fragment kodu authMiddleware.js

```
7   export default async function authMiddleware(req, res, next) {
8     const authHeader = req.headers.authorization;
9
10    if (!authHeader || !authHeader.startsWith('Bearer ')) {
11      return res.status(401).json({ message: "Brak tokenu uwierzytelniającego" });
12    }
13
14    const token = authHeader.split(" ")[1];
15
16    try {
17      // Weryfikacja tokenu
18      const decoded = jwt.verify(token, process.env.JWT_SECRET);
19
20      // Sprawdzenie czy użytkownik nadal istnieje i nie jest zablokowany
21      const [rows] = await pool.query(
22        "SELECT userid, userrole, isblocked FROM Users WHERE userid = ?",
23        [decoded.userid]
24      );
25
26      if (rows.length === 0) {
27        return res.status(401).json({ message: "Użytkownik nie istnieje" });
28      }
29
30      if (rows[0].isblocked) {
31        return res.status(403).json({ message: "Konto zostało zablokowane" });
32      }
33
34      req.user = decoded;
35      next();
36
37    } catch (err) {
38      if (err.name === 'TokenExpiredError') {
39        return res.status(401).json({ message: "Token wygasł" });
40      }
41      return res.status(403).json({ message: "Nieprawidłowy token" });
42    }
43  }
```

5.2. Implementacja części klienckiej

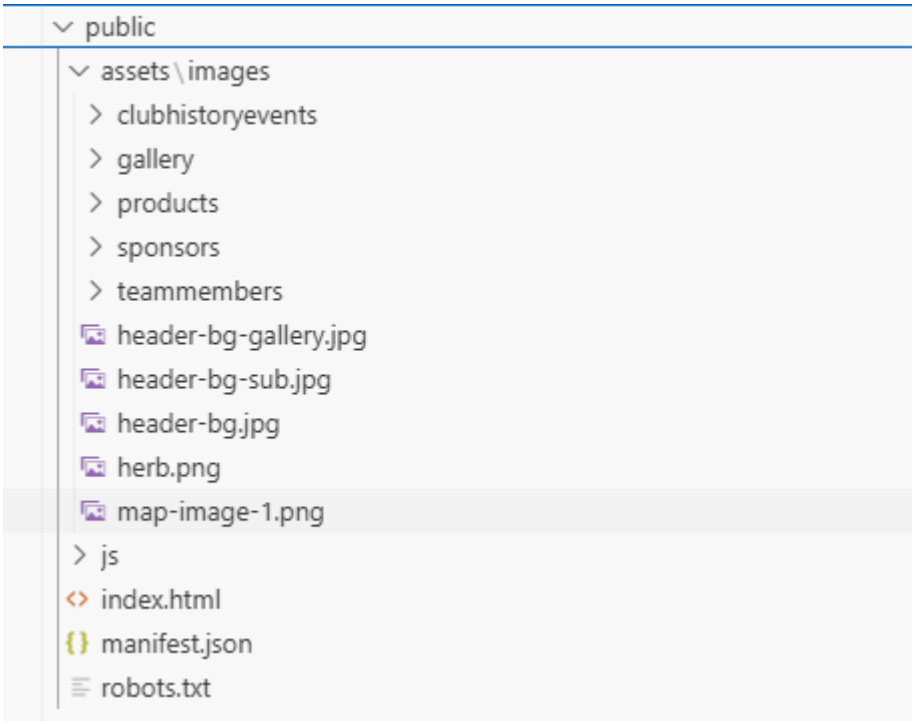
Struktura projektu

Rysunek 5.6. przedstawia strukturę katalogów w aplikacji klienckiej, gdzie można wyszczególnić poszczególne foldery i pliki, odpowiadające za konkretne elementy systemu:



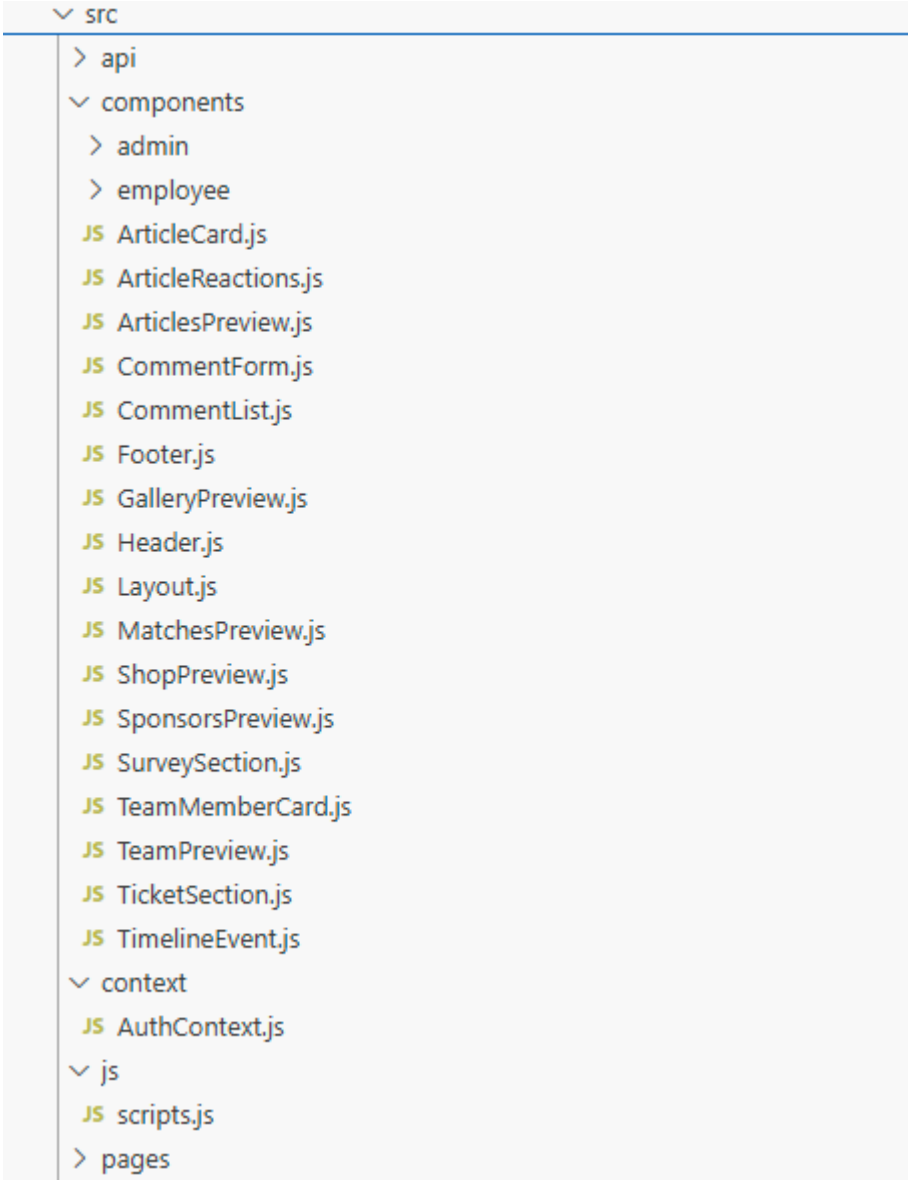
Rysunek 5.6. Struktura aplikacji klienckiej

Katalog **public** (Rysunek 5.7.) zawiera statyczne zasoby aplikacji, które przesyłane są bezpośrednio do przeglądarki bez ich uprzedniego przetwarzania. Znajdują się tutaj pliki takie jak grafiki, miniatury oraz inne zasoby statyczne.



Rysunek 5.7. Zawartość katalogu public

Katalog **src** (Rysunek 5.8.) to główny katalog kodu źródłowego aplikacji React, zawierający całą logikę aplikacji klienckiej.



Rysunek 5.8. Zawartość katalogu scr

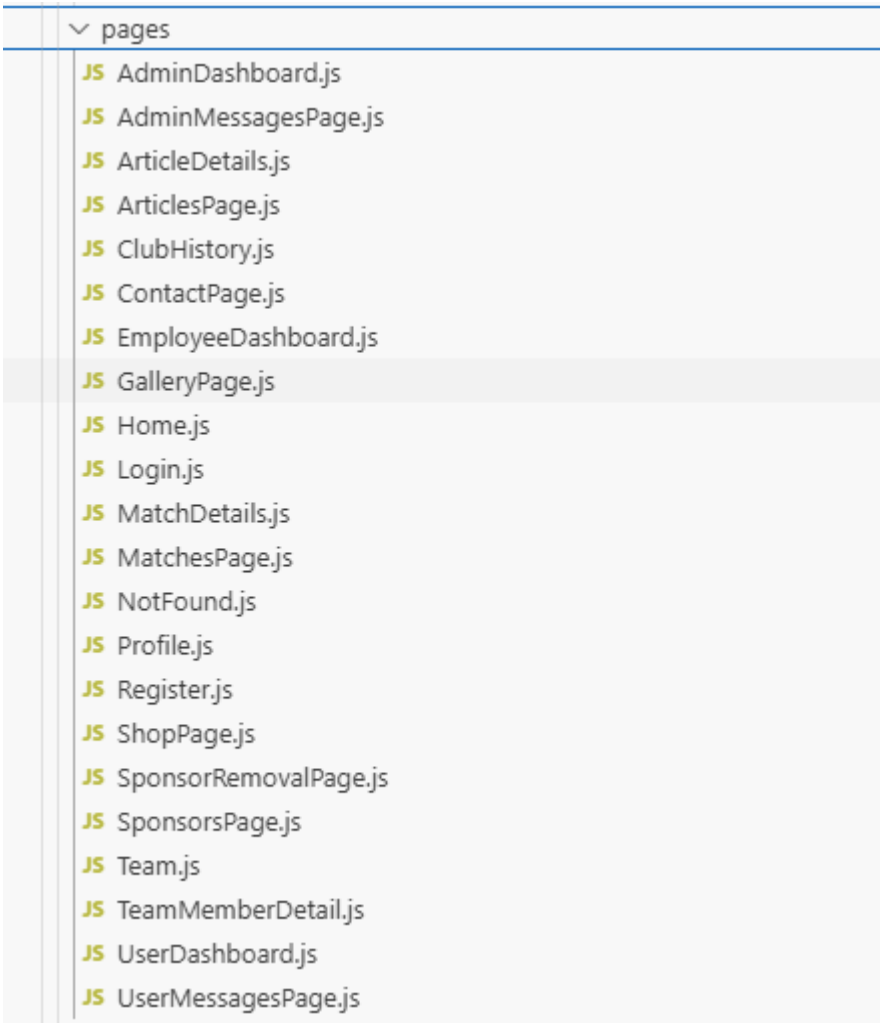
Podkatalog **api** zawiera funkcje odpowiadające za komunikację z aplikacją serwerową. Znajdują się tutaj moduły obsługujące żądania do różnych endpointów API, takie jak żądania dotyczące drużyny, meczy, artykułów czy sklepu.

Podkatalog **components** zawiera komponenty React, wielokrotnie wykorzystywane bloki interfejsu użytkownika. Komponenty mogą reprezentować elementy takie jak nagłówek, nawigacja, karty, formularze czy przyciski.

Podkatalog **context** zawiera konteksty React (React Context API), które służą do globalnego zarządzania stanem aplikacji. Umożliwia to przekazywanie danych między komponentami. Znajdują się tam konteksty takie jak autentykacja użytkownika.

Podkatalog **js** zawiera pomocnicze pliki JavaScript, takie jak funkcje użytkowe, stałe aplikacji, konfigurację lub inne logiczne struktury wspierające komponenty.

Podkatalog **pages** (Rysunek 5.9.) zawiera komponenty React reprezentujące całe strony aplikacji. Każda strona w ma swój plik w tym katalogu i jest najczęściej mapowana do ścieżki URL za pomocą React Router.



Rysunek 5.9. Zawartość podkatalogu pages

Plik **App.js** (Listing 5.7. – 5.8.) to główny komponent aplikacji React, zawierający konfigurację routingu oraz ogólną strukturę aplikacji.

Listing 5.7. Fragment kodu App.js zawierający importy stron oraz komponentów

frontend > src > JS App.js > ...

```
1   import React from "react";
2   import { Routes, Route } from "react-router-dom";
3   import Layout from "./components/Layout";
4   import Home from "./pages/Home";
5   import Login from "./pages/Login";
6   import Register from './pages/Register';
7   import NotFound from "./pages/NotFound";
8   import Profile from "./pages/Profile";
9   import Team from './pages/Team';
10  import TeamMemberDetail from './pages/TeamMemberDetail';
11  import ArticleDetails from "./pages/ArticleDetails";
12  import ArticlesPage from "./pages/ArticlesPage";
13  import { AuthProvider } from "./context/AuthContext";
14  import GalleryPage from './pages/GalleryPage';
15  import SponsorsPage from "./pages/SponsorsPage";
16  import MatchesPage from "./pages/MatchesPage";
17  import MatchDetails from "./pages/MatchDetails";
18  import ShopPage from "./pages/ShopPage";
19  import ContactPage from "./pages/ContactPage";
20  import UserMessagesPage from "./pages/UserMessagesPage";
21  import AdminMessagesPage from "./pages/AdminMessagesPage";
22  import AdminDashboard from "./pages/AdminDashboard";
23  import EmployeeDashboard from "./pages/EmployeeDashboard";
24  import UserDashboard from "./pages/UserDashboard";
25  import SponsorRemovalPage from "./pages/SponsorRemovalPage";
```

Listing 5.8. Fragment kodu App.js zawierający konfigurację routingu

```
function App() {
  return (
    <AuthProvider>
      <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        <Route path="login" element={<Login />} />
        <Route path="register" element={<Register />} />
        <Route path="profile" element={<Profile />} />
        <Route path="team" element={<Team />} />
        <Route path="team/:id" element={<TeamMemberDetail />} />
        <Route path="articles" element={<ArticlesPage />} />
        <Route path="articles/:id" element={<ArticleDetails />} />
        <Route path="/gallery" element={<GalleryPage />} />
        <Route path="/sponsors" element={<SponsorsPage />} />
        <Route path="/matches" element={<MatchesPage />} />
        <Route path="/matches/:id" element={<MatchDetails />} />
        <Route path="/shop" element={<ShopPage />} />
        <Route path="/contact" element={<ContactPage />} />
        <Route path="/my-messages" element={<UserMessagesPage />} />
        <Route path="/admin/messages" element={<AdminMessagesPage />} />
        <Route path="/admin/dashboard" element={<AdminDashboard />} />
        <Route path="/employee/dashboard" element={<EmployeeDashboard />} />
        <Route path="/user/dashboard" element={<UserDashboard />} />
        <Route path="sponsor-removal/:sponsorid" element={<SponsorRemovalPage />} />

        <Route path="*" element={<NotFound />} />
      </Route>
    </Routes>

    </AuthProvider>

  );
}

export default App;

```

Plik **index.js** (Listing 5.9.) to punkt wejścia całej aplikacji React. Zawiera kod, który renderuje główny komponent (App) do DOM-u przeglądarki.

Listing 5.9. Kod pliku index.js

frontend > src > JS index.js > ...

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./styles.css";
import App from "./App";
import { BrowserRouter } from "react-router-dom";
import { AuthProvider } from "./context/AuthContext";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <AuthProvider>
      <App />
    </AuthProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Plik **styles.css** zawiera globalne style CSS dla całej aplikacji.

Wygląd strony

Na wygląd strony składa się kilka plików. Na listingu 5.10. przedstawiono układ strony zastosowany w prezentowanej aplikacji.

Listing 5.10. Kod zawierający standardowy układ strony

```
export default function Layout() {
  return (
    <>
      <Header />
        <Outlet />
      <Footer />
    </>
  );
}
```

Listing 5.11. przedstawia nagłówek, który zawarty jest na każdej stronie widocznej dla użytkowników. Zawiera on między innymi pasek górny na którym widoczne są elementy nawigacyjne umożliwiające swobodne poruszanie się po aplikacji.

Listing 5.11. Fragment kodu Header.js zawierając elementy nawigacyjne

```
<div className="collapse navbar-collapse" id="navbarResponsive">
  <ul className="navbar-nav text-uppercase ms-auto py-4 py-lg-0">
    <li className="nav-item">
      <a className="nav-link" href="/#news">Aktualności</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="/#shop">Sklep</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="/#tickets">Bilety</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="/#team">Zespół</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="/#history">Historia</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="/#gallery">Galeria</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="/#sponsors">Sponsorzy</a>
    </li>
```

Listing 5.12. przedstawia stopkę, która jest nieodłącznym elementem każdej strony widocznej przez użytkownika w prezentowanej aplikacji. Są w niej zawarte informacje dotyczące autora aplikacji oraz linki do mediów społecznościowych drużyny.

Listing 5.12. Fragment kodu Footer.js zawierający schemat budowy stopki

```
export default function Footer() {
  return (
    <footer className="footer py-4 bg-dark text-light">
      <div className="container">
        <div className="row align-items-center text-center text-lg-start">
          <div className="col-lg-4 mb-3 mb-lg-0">
            <small>Copyright &copy; Dariusz Bochyński 2025</small>
          </div>
          <div className="col-lg-4 mb-3 mb-lg-0">
            <a
              className="btn btn-dark btn-social mx-2"
              href="https://x.com/LukLublin"
              aria-label="Twitter"
            >
              <i className="fab fa-twitter"></i>
            </a>
            <a
              className="btn btn-dark btn-social mx-2"
              href="https://www.facebook.com/LUKLublin"
              aria-label="Facebook"
            >
              <i className="fab fa-facebook-f"></i>
            </a>
            <a
              className="btn btn-dark btn-social mx-2"
              href="https://www.instagram.com/bogdanka_luk_lublin/"
              aria-label="Instagram"
            >
              <i className="fab fa-instagram"></i>
            </a>
            <a
              className="btn btn-dark btn-social mx-2"
              href="https://www.youtube.com/@LKPSLublinTV"
              aria-label="YouTube"
            >
```

Listing 5.13. przedstawia stronę startową aplikacji klienckiej, którą widzą użytkownicy po uruchomieniu systemu. Jest to główny widok, który zawiera podglądy najważniejszych podstron serwisu. Są to uproszczone wersje, pobierające mniej danych, aby niezależnie od wielkości bazy danych strona była przejrzysta i czytelna. Ich zastosowanie pokazano na listingu 5.14.

Listing 5.13. Fragment kodu Home.js zawierający stronę główną aplikacji

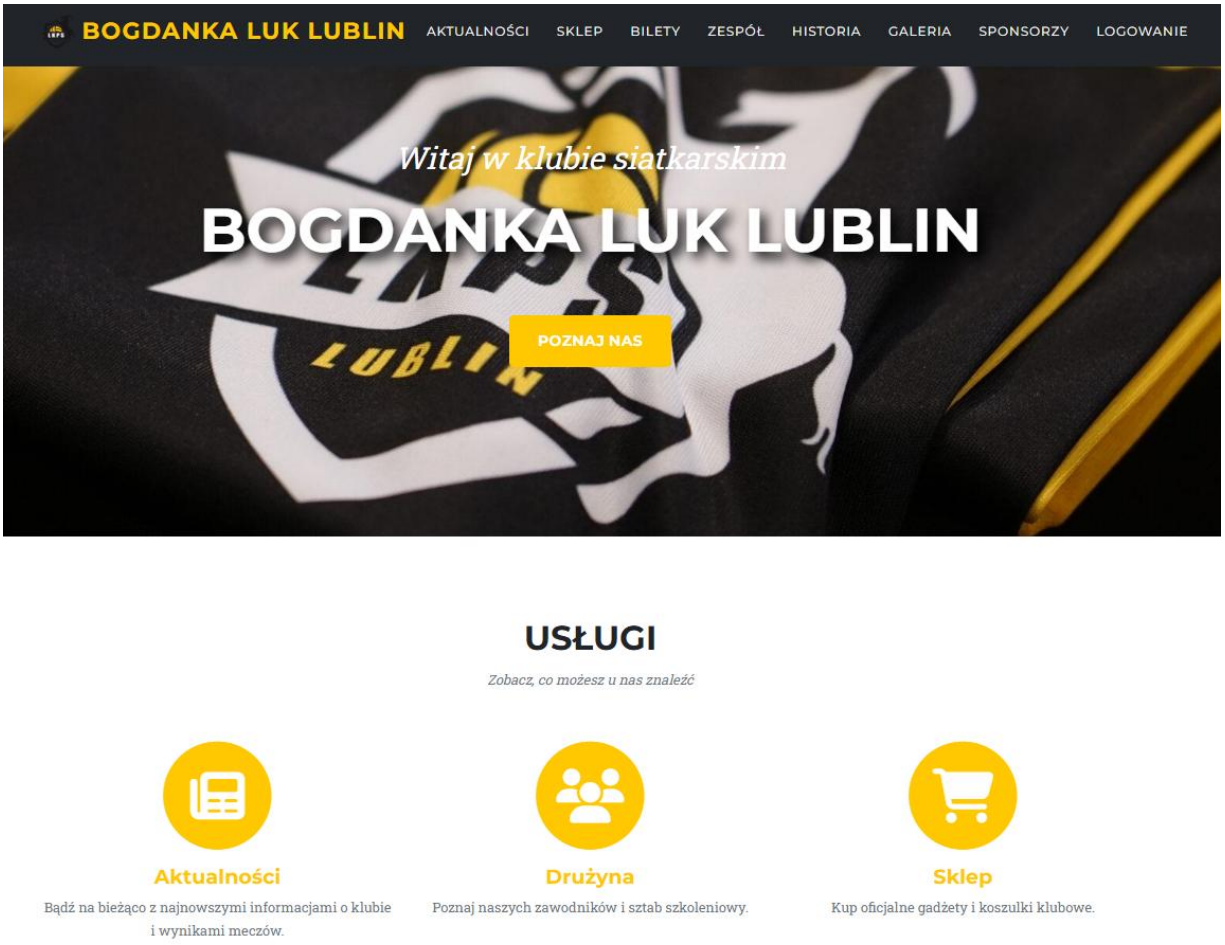
frontend > src > pages > JS Home.js > Home

```
export default function Home() {
  return (
    <>
      {/* Masthead */}
      <header className="masthead">
        <div className="container">
          <div className="masthead-subheading">Witaj w klubie siatkarskim</div>
          <div className="masthead-heading text-uppercase">Bogdanka LUK Lublin</div>
          <a className="btn btn-primary btn-xl text-uppercase" href="#services">
            Poznaj nas
          </a>
        </div>
      </header>
      {/* Services Section */}
      <section className="page-section" id="services">
        <div className="container">
          <div className="text-center">
            <h2 className="section-heading text-uppercase">Usługi</h2>
            <h3 className="section-subheading text-muted">
              Zobacz, co możesz u nas znaleźć
            </h3>
          </div>
          <div className="row text-center">
            <div className="col-md-4">
              <Link to="/articles" className="text-decoration-none">

                <span className="fa-stack fa-4x">
                  <i className="fas fa-circle fa-stack-2x text-primary"></i>
                  <i className="fas fa-newspaper fa-stack-1x fa-inverse"></i>
                </span>

                <h4 className="team-heading-link">Aktualności</h4>
              </Link>
              <p className="text-muted">
                Bądź na bieżąco z najnowszymi informacjami o klubie i wynikami meczów.
              </p>
            </div>
            <div className="col-md-4">
              <Link to="/team" className="text-decoration-none">
                <span className="fa-stack fa-4x">
                  <i className="fas fa-circle fa-stack-2x text-primary"></i>
                  <i className="fas fa-users fa-stack-1x fa-inverse"></i>
                </span>
```

Listing 5.14. Fragment kodu Team.js przedstawiający odwołania do stron i komponentów

```
      </section>
      <section className="page-section" id="news">
        <ArticlePreview />
      </section>
      <section className="page-section" id="matches">
        <MatchesPreview />
      </section>
      <section className="page-section" id="shop">
        <ShopPreview />
      </section>
      <section className="page-section" id="tickets">
        <TicketsSection />
      </section>
      <section className="page-section" id="team">
        <TeamPreview />
      </section>
      <section className="page-section" id="history">
        <ClubHistory />
      </section>
      <section className="page-section" id="gallery">
        <GalleryPreview />
      </section>
      <section className="page-section" id="survey">
        <SurveySection />
      </section>
      <section className="page-section" id="sponsors">
        <SponsorsPreview />
      </section>
    </>
```

Rysunek 5.10. przedstawia finalny widok, który widzi odbiorca po załadowaniu strony głównej. Może on z tego miejsca przejść do każdej funkcjonalności aplikacji.



Rysunek 5.10. Widok strony głównej aplikacji

Zespół

W skład modułu tworzącego funkcjonalność zespołu wchodzi kilka komponentów. Są to:

TeamMemberCard.js (Listing 5.15.), jest komponentem, który wyświetla kartę członka drużyny z jego podstawowymi informacjami. Komponent przyjmuje dwa parametry: obiekt player zawierający dane członka drużyny oraz opcjonalny parametr showDetails, który określa, czy mają być wyświetlone dodatkowe szczegóły.

Komponent zawiera dwie pomocnicze funkcje: calculateAge oblicza wiek zawodnika na podstawie jego daty urodzenia, a getPositionBadgeClass zwraca odpowiedni kolor znacznika dla pozycji na boisku.

Interfejs karty wyświetla zdjęcie profilowe, numer koszulki w górnym lewym rogu, imię i nazwisko, pozycję na boisku oraz przycisk "Zobacz profil" kierujący do strony poświęconej szczegółom danego zawodnika. Jeśli parametr showDetails jest ustawiony na true, komponent dodatkowo wyświetla szczegółowe dane oraz linki do profili mediów społecznościowych.

Listing 5.15. Fragment kodu TeamMemberCard.js

```
4   export default function TeamMemberCard({ player, showDetails = false }) {
5     const calculateAge = (dateOfBirth) => {
6       const today = new Date();
7       const birthDate = new Date(dateOfBirth);
8       let age = today.getFullYear() - birthDate.getFullYear();
9       const monthDiff = today.getMonth() - birthDate.getMonth();
10      if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
11        age--;
12      }
13      return age;
14    };
15
16    const getPositionBadgeClass = (position) => {
17      const lowerPos = position.toLowerCase();
18
19      if (lowerPos.includes('libero')) return 'bg-secondary';
20      return 'bg-primary';
21    };
```

Team.js (Listing 5.16.) jest komponentem odpowiadającym za wyświetlenie pełnego składu drużyny z możliwością przełączania między zawodnikami a sztabem szkoleniowym. Komponent wykorzystuje hooki React do zarządzania stanem aplikacji: players i Staff przechowują dane członków drużyny, loading informuje o trwającym ładowaniu danych, error przechowuje komunikaty błędów, a activeTab określa, która karta jest aktualnie wyświetlana.

W momencie załadowania komponentu hook useEffect uruchamia funkcję fetchTeamMembers, która wysyła asynchroniczne żądanie HTTP do aplikacji serwerowej w celu pobrania listy zawodników i sztabu szkoleniowego. Podczas ładowania wyświetlana jest ikona ładowania, a w przypadku błędu wyświetlany jest komunikat informujący użytkownika o problemie.

Interfejs strony zawiera dwie karty: jedną dla zawodników i jedną dla sztabu szkoleniowego, z wyświetlaniem liczby osób w każdej grupie. Po wybraniu konkretnej karty strona wyświetla odpowiednie karty członków. Każda karta zawiera zdjęcie, imię, nazwisko, pozycję i przycisk prowadzący do szczegółów profilu. Strona posiada również przycisk powrotu do strony głównej oraz nagłówek z tytułem i podtytułem.

Listing 5.16. Fragment kodu Team.js

```
export default function Team() {
  const [players, setPlayers] = useState([]);
  const [staff, setStaff] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");
  const [activeTab, setActiveTab] = useState("players");

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

  const fetchTeamMembers = async () => {
    try {
      setLoading(true);
      const res = await api.get("/team");
      setPlayers(res.data.players);
      setStaff(res.data.staff);
    } catch (err) {
      console.error("Błąd pobierania drużyny:", err);
      setError("Nie udało się pobrać składu drużyny");
    } finally {
      setLoading(false);
    }
  };

  if (loading) {
    return (
      <section className="page-section">
        <div className="container">
          <div className="text-center">
            <div className="spinner-border text-primary" role="status">
              <span className="visually-hidden">Ładowanie...</span>
            </div>
            <p className="mt-3 text-muted">Ładowanie składu drużyny...</p>
          </div>
        </div>
      </section>
    );
  }
```

TeamMemberDetails.js (Listing 5.17.) jest komponentem odpowiedzialnym za wyświetlanie szczegółów profili poszczególnych członków drużyny. Komponent pobiera identyfikator członka drużyny z parametru URL przy pomocy hooka useParams i na tej podstawie wysyła żądanie do aplikacji serwerowej w celu pobrania pełnych danych o wybranym członku drużyny. Stan komponentu zawiera trzy zmienne: member przechowuje dane członka, loading informuje o trwającym ładowaniu, a error przechowuje ewentualny komunikat błędu.

Hook useEffect uruchamia się w momencie załadowania komponentu oraz za każdym razem, gdy zmienia się identyfikator w parametrze URL. Funkcja fetchMemberDetails wysyła asynchroniczne żądanie HTTP do części serwerowej aplikacji i pobiera szczegółowe informacje o członku drużyny. Komponent zawiera również dwie pomocnicze funkcje: calculateAge oblicza wiek na podstawie daty urodzenia, a formatDate formatuje daty w polskim formacie.

Interfejs strony jest podzielony na dwie główne kolumny: lewa zawiera zdjęcie profilowe członka z numerem koszulki w górnym lewym rogu, imię, nazwisko, pozycję na boisku oraz linki do mediów społecznościowych. Prawa kolumna zawiera szczegółowe informacje takie jak wiek, narodowość, wzrost, wagę, datę urodzenia i datę dołączenia do klubu, wyświetlane z odpowiednimi ikonami. Jeśli członek ma biografię, jest ona wyświetlana w osobnej karcie poniżej. Na górze strony znajduje się przycisk powrotu do listy drużyny, a w przypadku błędu podczas ładowania wyświetlany jest komunikat błędu z opcją powrotu.

Listing 5.17. Fragment kodu TeamMemberDetails.js

```
export default function TeamMemberDetail() {
  const { id } = useParams();
  const [member, setMember] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    fetchMemberDetails();
  }, [id]);

  const fetchMemberDetails = async () => {
    try {
      setLoading(true);
      const res = await api.get(`/team/${id}`);
      setMember(res.data);
    } catch (err) {
      console.error("Błąd pobierania szczegółów:", err);
      setError("Nie udało się pobrać szczegółów członka drużyny");
    } finally {
      setLoading(false);
    }
  };

  const calculateAge = (dateOfBirth) => {
    const today = new Date();
    const birthDate = new Date(dateOfBirth);
    let age = today.getFullYear() - birthDate.getFullYear();
    const monthDiff = today.getMonth() - birthDate.getMonth();
    if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
      age--;
    }
    return age;
  };

  const formatDate = (dateString) => {
    return new Date(dateString).toLocaleDateString('pl-PL', {
      year: 'numeric',
      month: 'long',
      day: 'numeric'
    });
  };
```

TeamPreview.js (Listing 5.18.) jest to skrócony podgląd drużyny, który jest wyświetlany na stronie głównej aplikacji, pokazujący maksymalnie sześciu pierwszych zawodników drużyny. Komponent wykorzystuje hook useState do zarządzania listą wyświetlanych zawodników oraz stanem ładowania, a hook useEffect uruchamia funkcję loadTeamMembers w momencie załadowania komponentu.

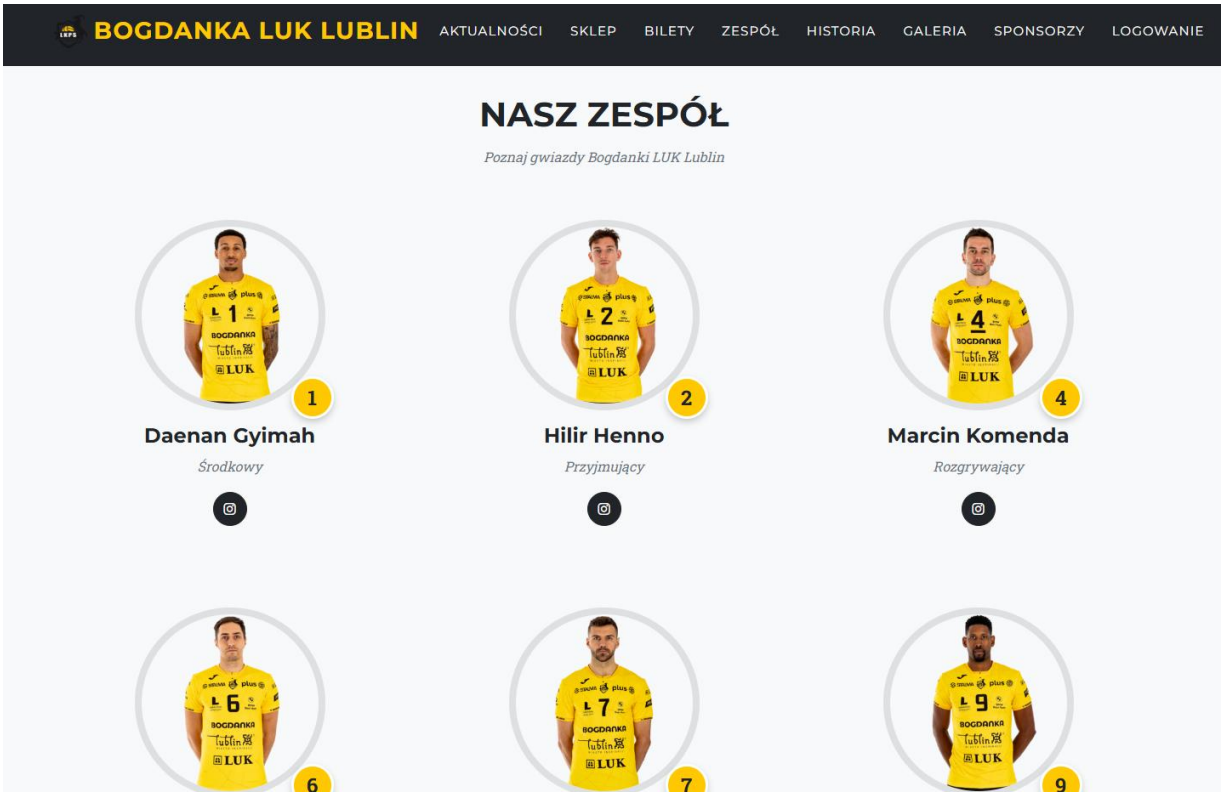
Funkcja loadTeamMembers wysyła asynchroniczne żądanie HTTP do aplikacji serwerowej w celu pobrania pełnej listy zawodników i personelu szkoleniowego. Z otrzymanej odpowiedzi odczytuje listę zawodników i pobiera z niej tylko pierwszych sześciu zawodników przy pomocy metody slice(0, 6), co umożliwia wyświetlenie zwartego podglądu, który będzie przejrzysty i czytelny dla użytkownika. W trakcie ładowania danych wyświetlana jest ikona ładowania, a jeśli lista zawodników jest pusta, wyświetlany jest odpowiedni komunikat informacyjny.

Interfejs komponentu wyświetla zawodników w siatce trzech kolumn, każdy na osobnej karcie zawierającej zdjęcie profilowe, numer koszulki, imię, nazwisko, pozycję oraz ikony linków do profili mediów społecznościowych. Każda karta jest w postaci hiperłącza i prowadzi do szczegółowego profilu danego zawodnika. Na górze sekcji znajduje się nagłówek "Nasz Zespół" również stanowiący link do pełnej strony drużyny, a poniżej kart wyświetlany jest przycisk "Zobacz pełny skład" kierujący na stronę z pełnym składem zespołu.

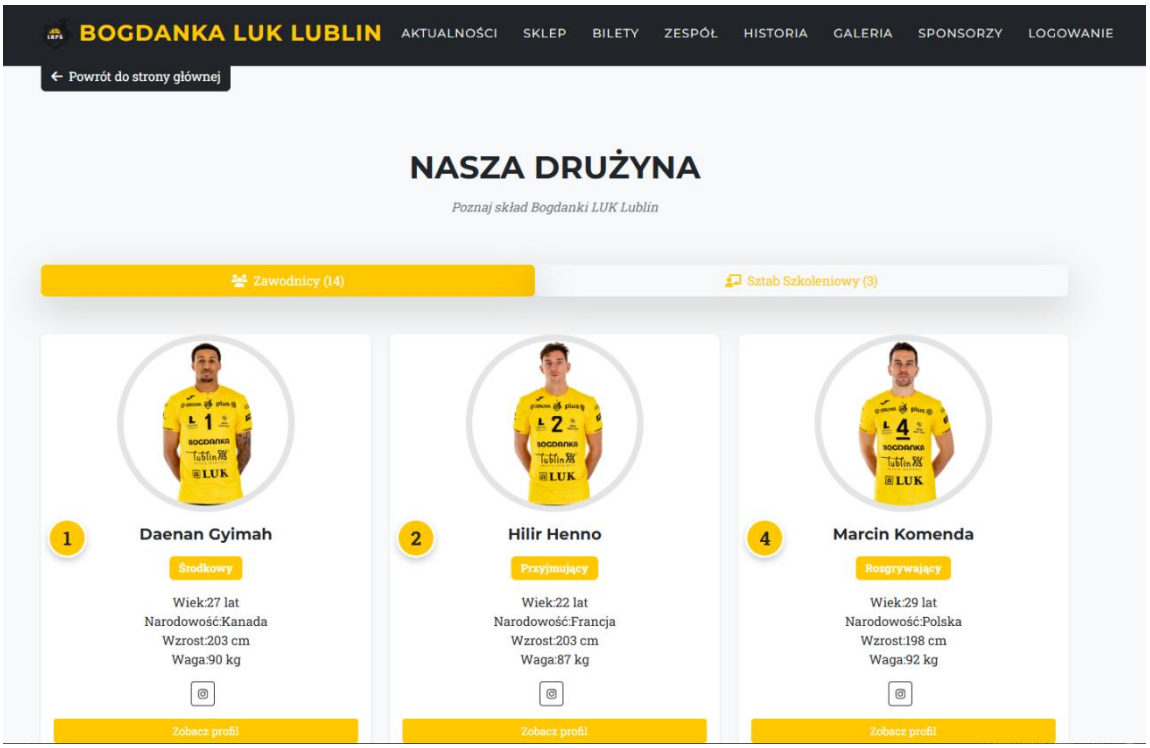
Listing 5.18. Fragment kodu TeamPreview.js

```
export default function TeamPreview() {
  const [teamMembers, setTeamMembers] = useState([]);
  const [teamLoading, setTeamLoading] = useState(true);

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

  const loadTeamMembers = async () => {
    try {
      const res = await api.get("/team");
      const players = res.data.players || [];
      setTeamMembers(players.slice(0, 6));
    } catch (err) {
      console.error("Błąd pobierania zespołu:", err);
    } finally {
      setTeamLoading(false);
    }
  };

  return (
    <section className="page-section bg-light" id="team">
      <div className="container">
        <div className="text-center">
          <Link to="/team" className="text-decoration-none">
            <h2 className="section-heading text-uppercase team-heading-link">
              Nasz Zespół
            </h2>
          </Link>
          <h3 className="section-subheading text-muted">
            Poznaj gwiazdy Bogdanki LUK Lublin
          </h3>
        </div>
```

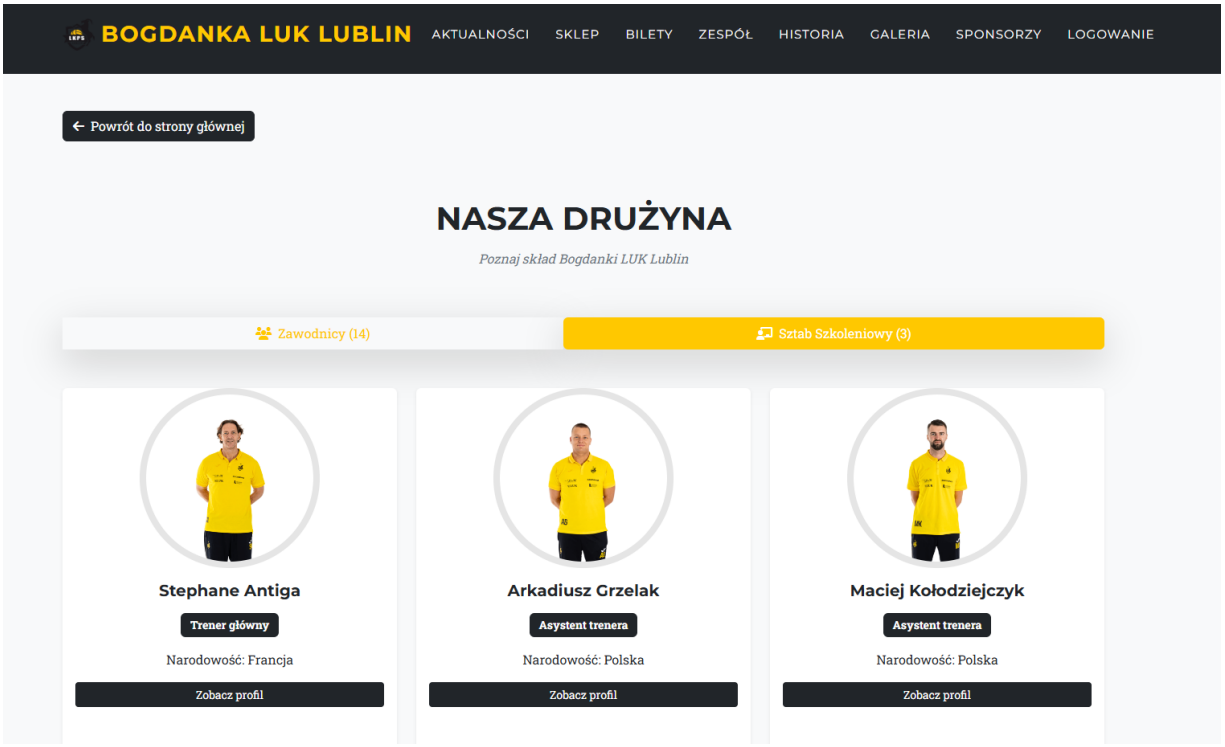
Na rysunkach 5.11. – 5.14. przedstawiono działanie funkcjonalności związanej ze składem. Została ona podzielona na kilka ekranów zawierających różne informacje.



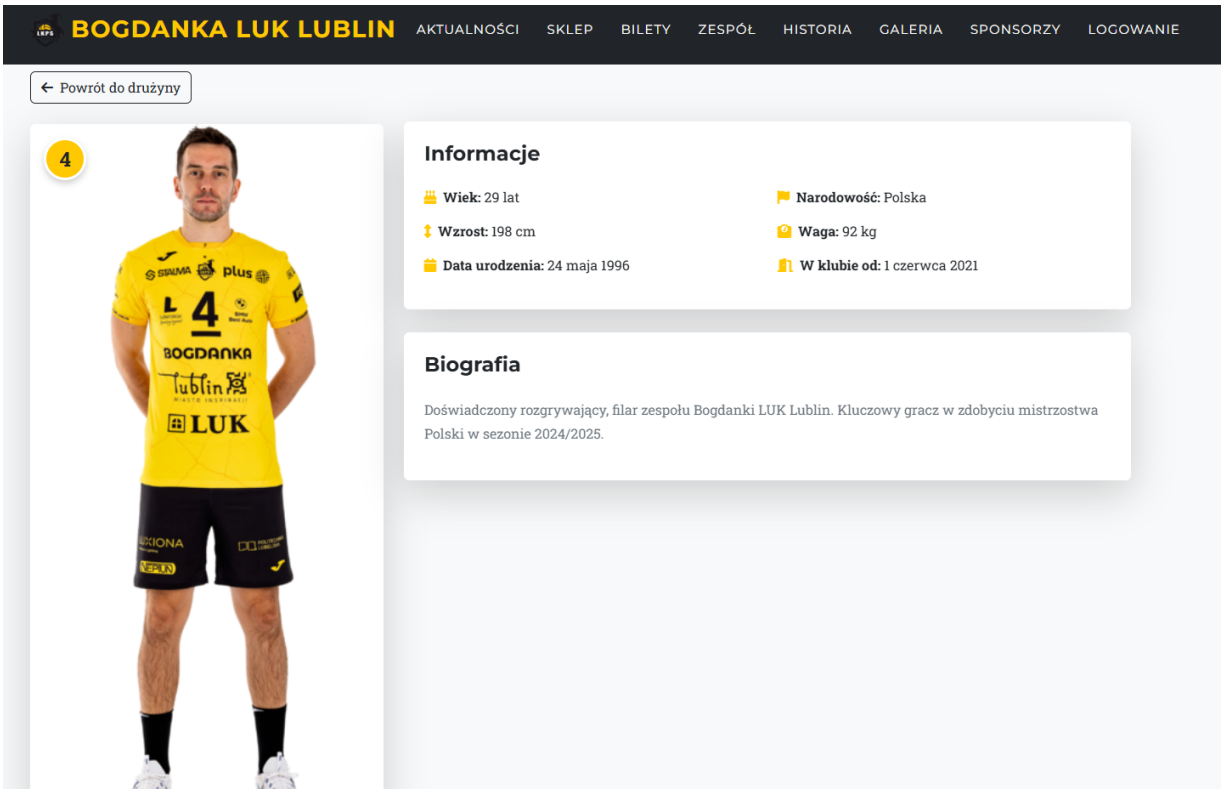
Rysunek 5.11. Widok zespołu z poziomu strony głównej



Rysunek 5.12. Widok strony poświęconej drużynie



Rysunek 5.13. Zakładka przedstawiająca sztab szkoleniowy

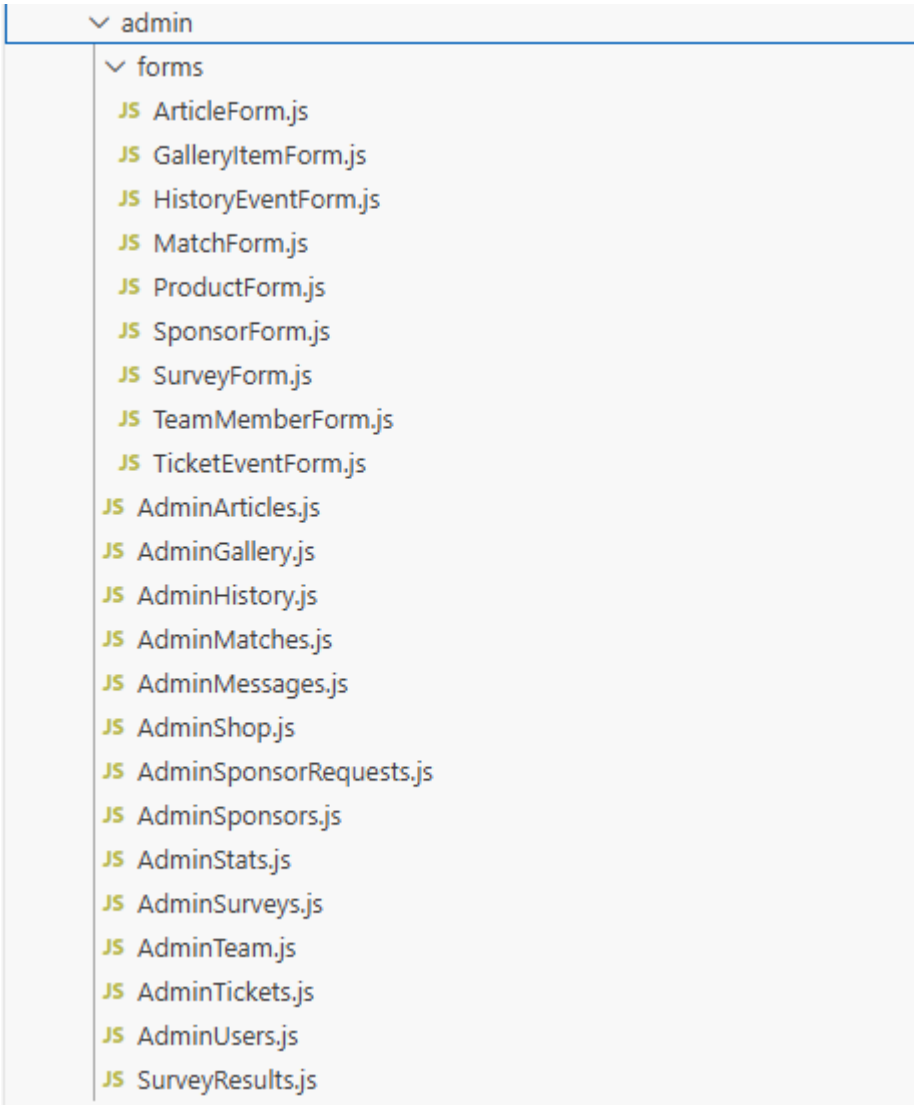


Rysunek 5.14. Szczegółowe informacje na temat danego zawodnika

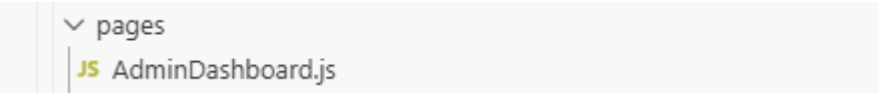
Panel administratora

Jest istotnym elementem w systemie pozwalającym uprzywilejowanym użytkownikom na zarządzanie i kontrolę danych, które przepływają przez aplikację. Zawiera on okno ułatwiające zarządzanie w którym wyświetlane są informacje oraz statystyki serwisu oraz zakładki, które umożliwiają edycję poszczególnych danych na stronie.

W skład panelu administratora wchodzą dokumenty i katalogi przedstawione na rysunkach 5.15. – 5.16.



Rysunek 5.15. Struktura katalogów panelu administratora



Rysunek 5.16. Struktura katalogów panelu administratora cd.

Najważniejsze dokumenty i katalogi w tej strukturze to:

AdminDashborad.js (Listing 5.19.), który znajduje się w katalogu pages. Jest to komponent tworzący panel administracyjny aplikacji, dostępny wyłącznie dla zalogowanego użytkownika z rolą administratora. Po załadowaniu komponent sprawdza, czy bieżący użytkownik ma odpowiednie uprawnienia; jeśli nie, następuje przekierowanie na stronę główną. Następnie wysyłane jest żądanie HTTP do endpointu /admin/stats, a otrzymane statystyki są zapisywane w stanie komponentu i przekazywane do komponentu AdminStats.

Interfejs panelu oparty jest na zakładkach, których aktywna wartość przechowywana jest w stanie activeTab. Każdy przycisk zakładki ustawia inny typ zawartości, na przykład: użytkownicy, drużyna, historia klubu, sklep, bilety, mecze, sponsorzy, ankiety, galeria, artykuły czy wiadomości. W zależności od aktualnie aktywnej zakładki renderowany jest odpowiedni komponent administracyjny, który odpowiada za zarządzanie wybraną częścią systemu z poziomu jednego, spójnego panelu.

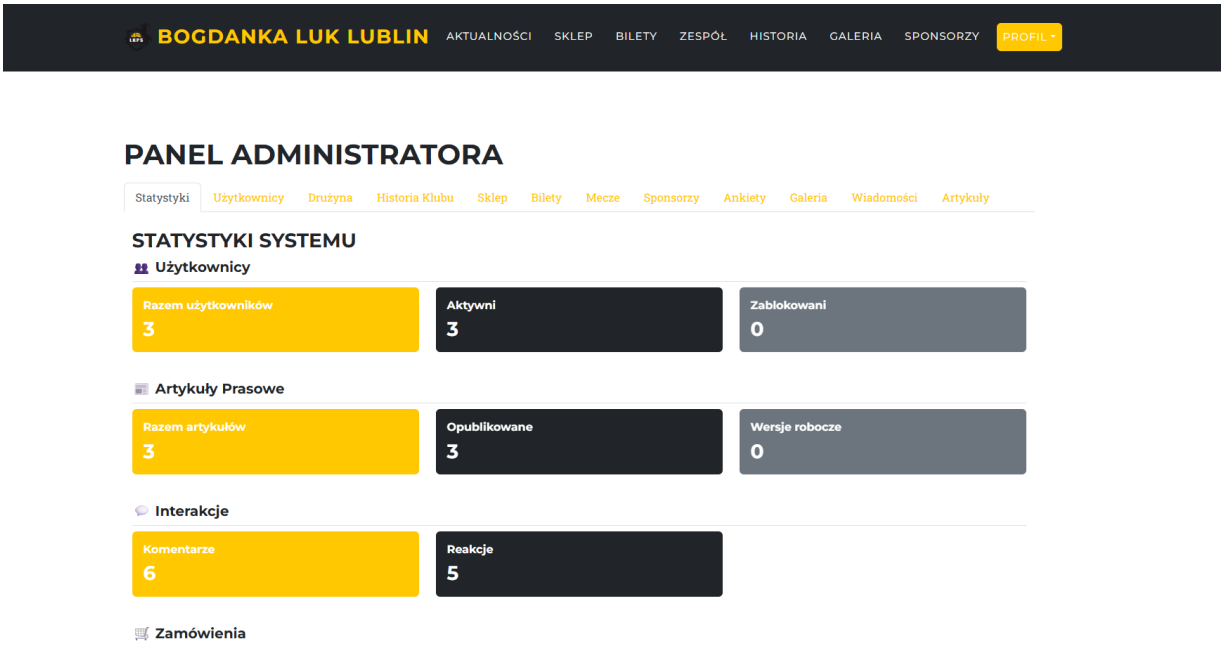
Listing 5.19. Fragment kodu Dashboard.js

```
export default function AdminDashboard() {
  const { user } = useContext(AuthContext);
  const navigate = useNavigate();
  const [stats, setStats] = useState({});
  const [activeTab, setActiveTab] = useState("stats");

  useEffect(() => {
    if (!user || user.userrole !== "admin") {
      navigate("/");
      return;
    }

    api.get("/admin/stats").then(res => setStats(res.data));
  }, [user, navigate]);

  return (
    <section className="container my-5">
      <h2 className="mb-4">Panel Administratora</h2>

      <ul className="nav nav-tabs mb-4">
        <li className="nav-item">
          <button className={`nav-link ${activeTab === "stats" ? "active" : ""}`} onClick={() => setActiveTab("stats")}>
            Statystyki
          </button>
        </li>
        <li className="nav-item">
          <button className={`nav-link ${activeTab === "users" ? "active" : ""}`} onClick={() => setActiveTab("users")}>
            Użytkownicy
          </button>
        </li>
        <li className="nav-item">
          <button className={`nav-link ${activeTab === "team" ? "active" : ""}`} onClick={() => setActiveTab("team")}>
            Drużyna
          </button>
        </li>
```

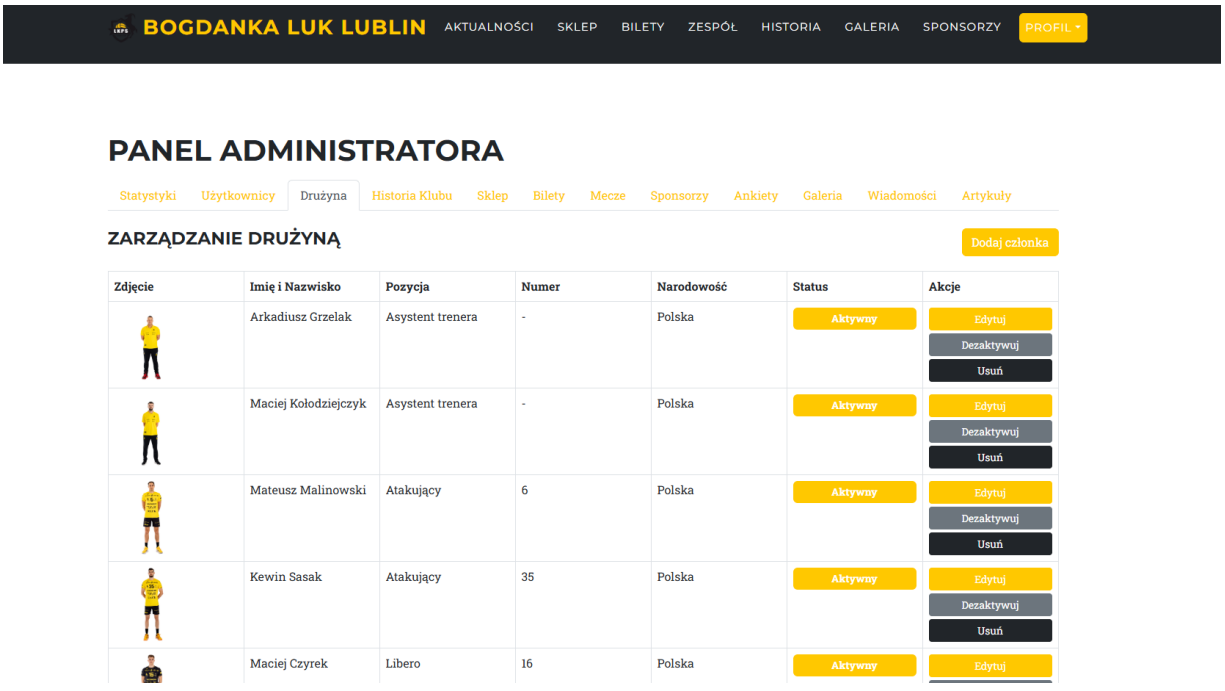
Katalog **admin** zawiera komponenty administracyjne, które odpowiadają za zarządzanie poszczególnych części systemu. Są one wykorzystywane w panelu AdminDashboard.js w formie przełączanych zakładek.

Podkatalog **forms** przechowuje formularze, które umożliwiają edycję poszczególnych obiektów znajdujących się w aplikacji.

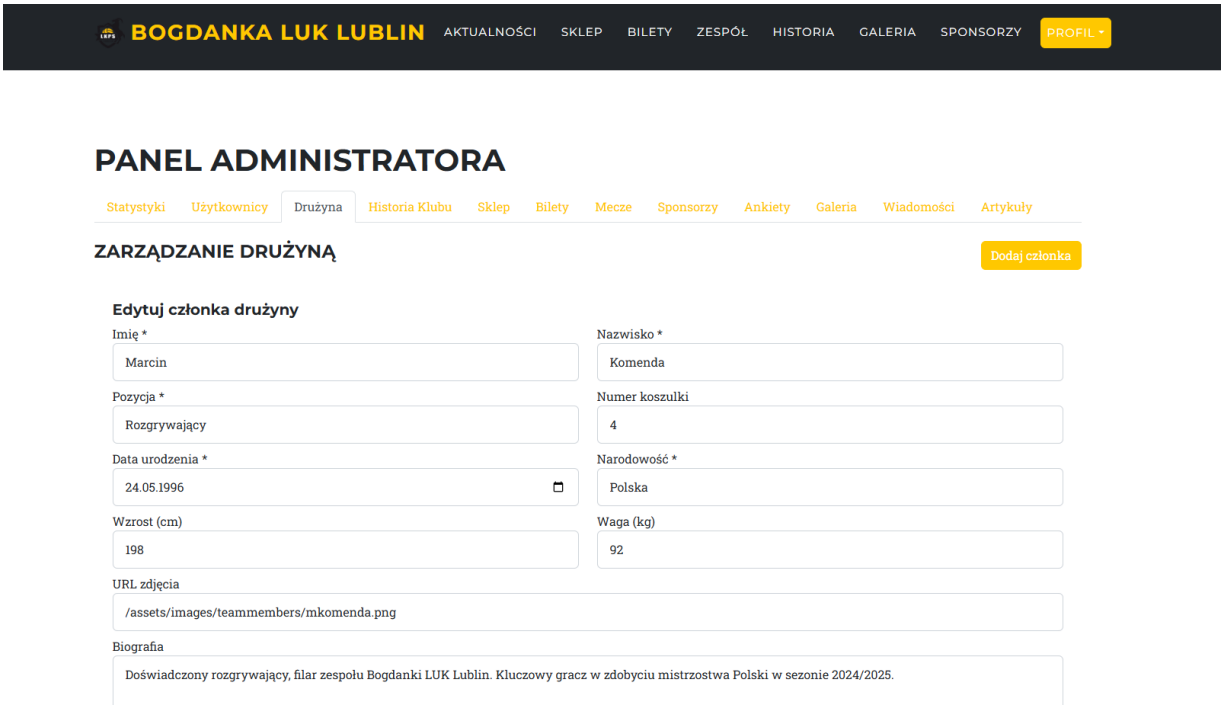
Na rysunkach 5.17. – 5.19. przedstawiono wygląd interfejsu użytkownika dla funkcjonalności panelu administratora.



Rysunek 5.17. Podgląd statystyk w panelu administratora



Rysunek 5.18. Zarządzanie drużyną w panelu administratora



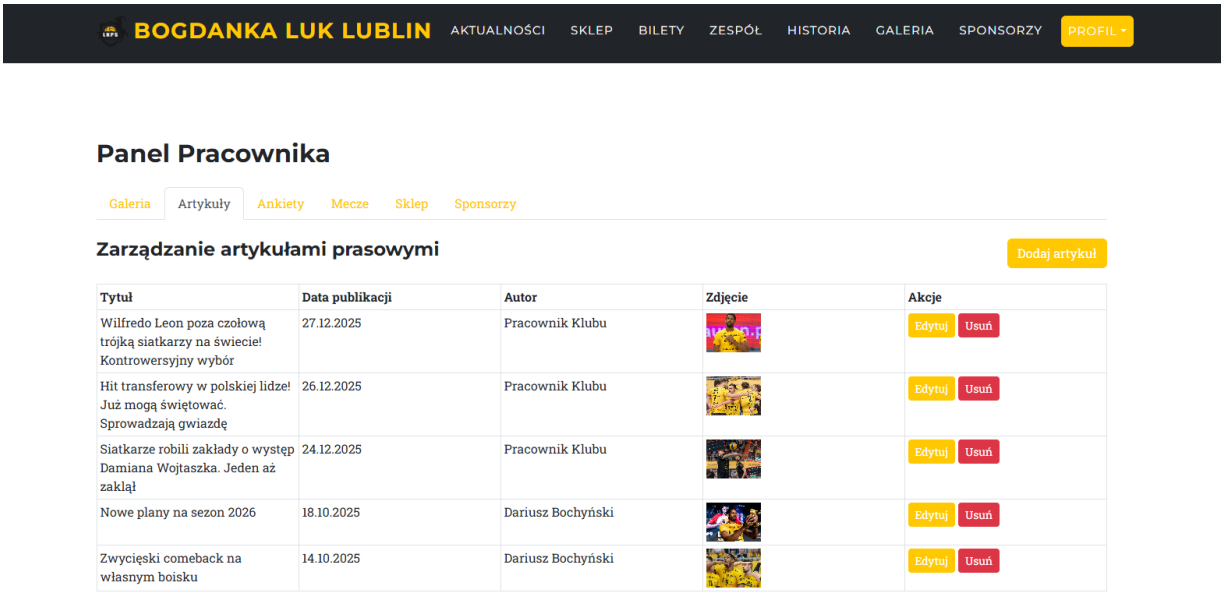
Rysunek 5.19. Edycja członka drużyny z poziomu panelu administratora

Panel pracownika klubu

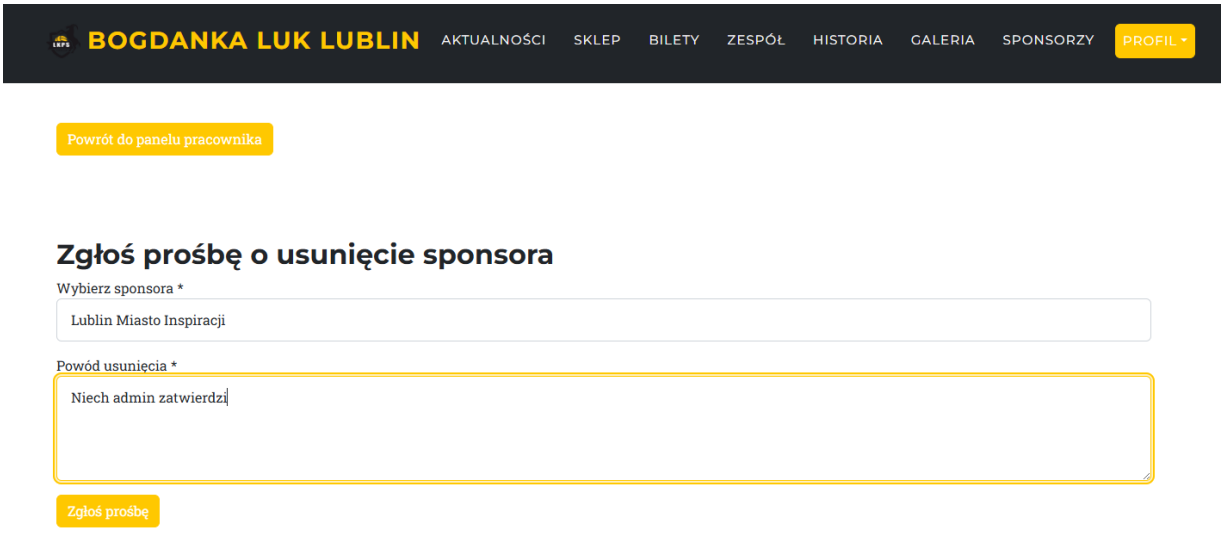
W aplikacji dostępny jest inny panel stworzony do zarządzania zasobami znajdującymi się w systemie. Panel pracownika klubu stworzony został z myślą o osobach zatrudnionych przez klub takich jak fotograf, redaktor, rzecznik prasowy czy pracownik sklepu klubowego. Ma on ograniczone funkcje względem panelu administratora, aby najbardziej istotne dane systemu nie zostały omyłkowo zmienione lub usunięte.

Panel pracownika klubu został zaprojektowany w bardzo podobny sposób jak panel administratora. Struktura plików i katalogów jest analogiczna a użyte metody niemalże identyczne.

Na rysunkach 5.20. przedstawiono końcowy wygląd panelu pracownika. Rysunek 5.21. przedstawia zgłoszenie prośby o usunięcie sponsora, która zostaje następnie wysłana do administratora. Jest to mechanizm, który zapobiega przypadkowego usunięcia istotnych danych.



Rysunek 5.20. Widok panelu pracownika

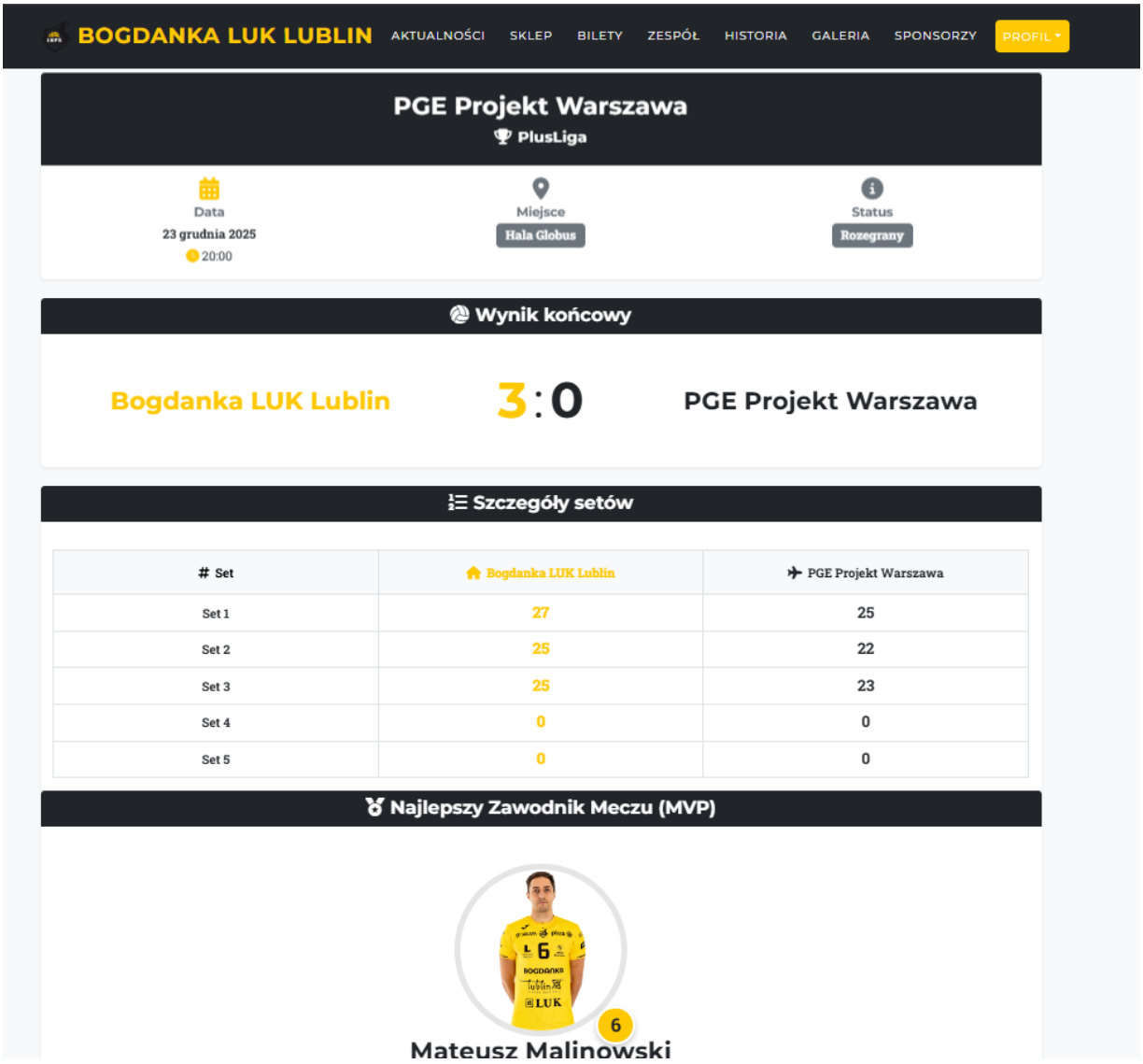


Rysunek 5.21. Prośba o usunięcie sponsora

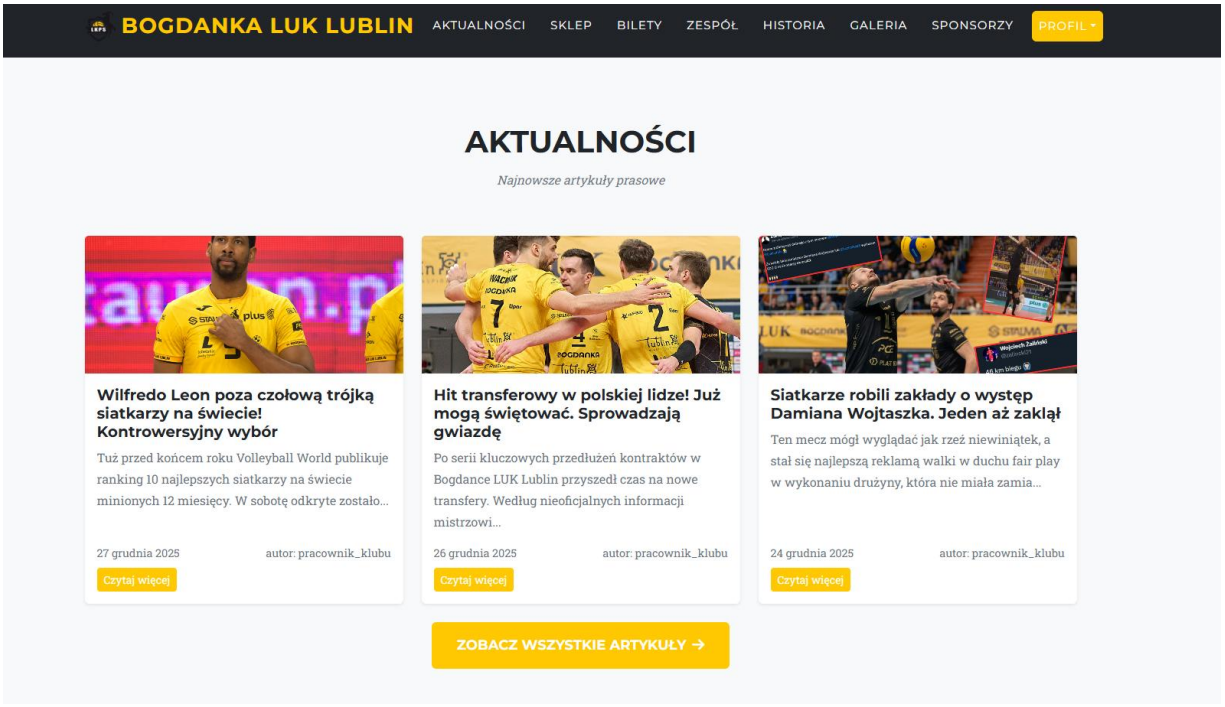
Wygląd pozostałych zaimplementowanych funkcjonalności

Na rysunkach 5.21. – 5.30. zostały przedstawione widoki użytkownika powstałych najważniejszych funkcjonalności, które oferuje prezentowana aplikacja. Interfejsy zostały zaprojektowane w taki sposób, aby ułatwić użytkownikowi poruszanie się po serwisie.

Rysunek 5.22. Strona poświęcona wynikom drużyny

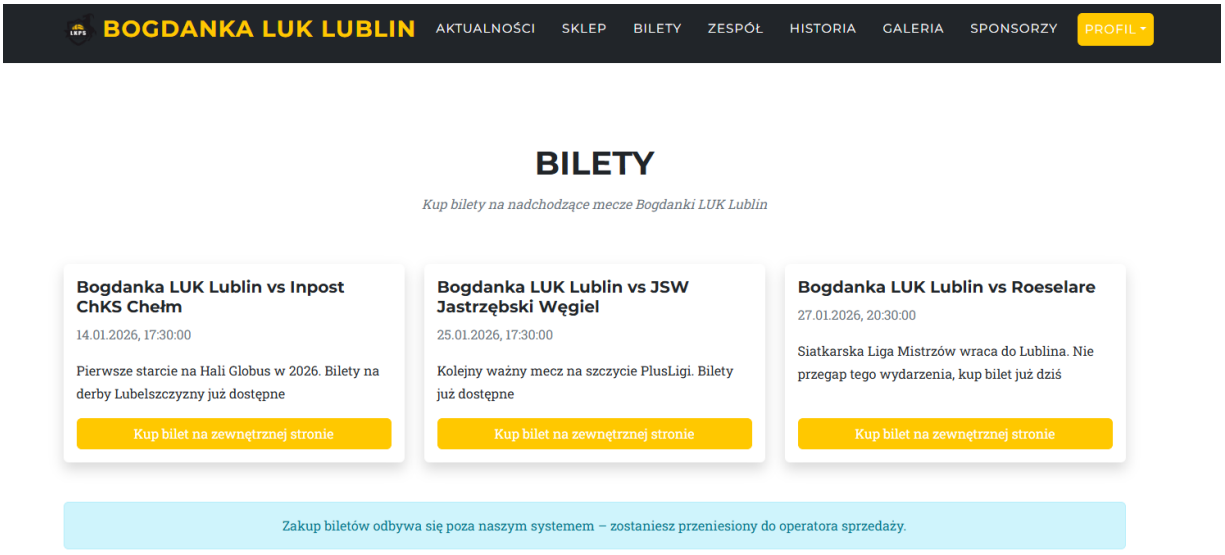


Rysunek 5.23. Strona poświęcona szczegółom danego meczu

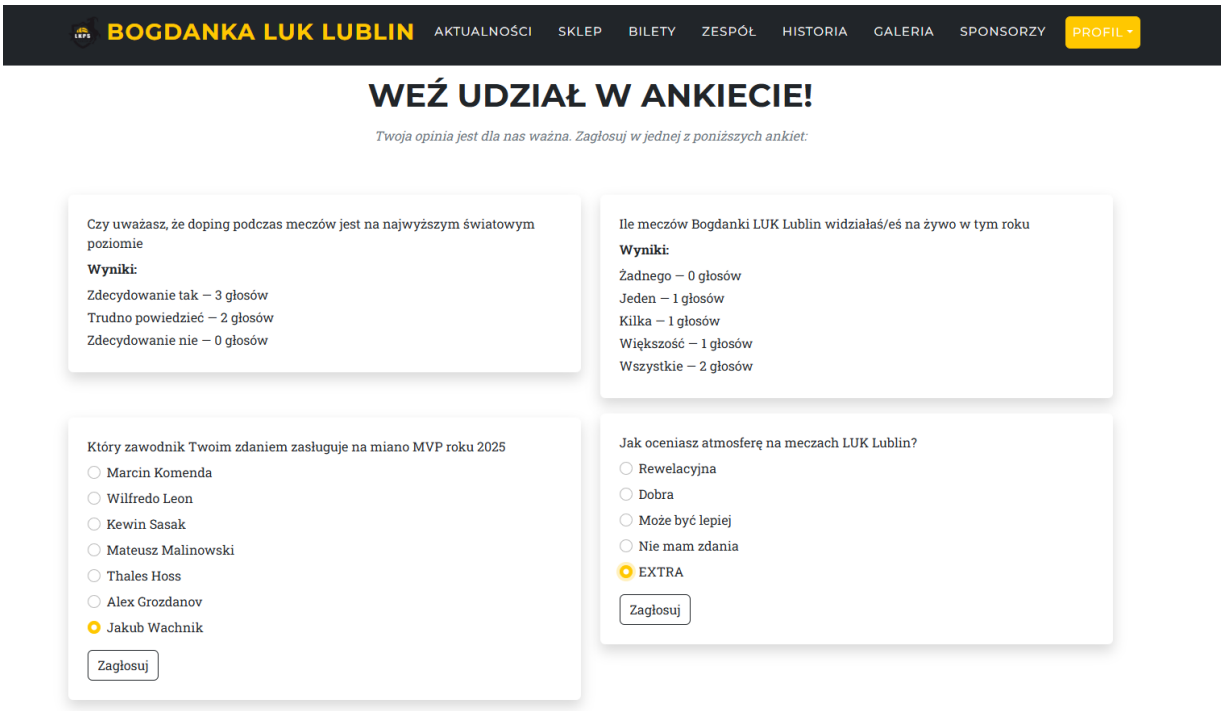


Rysunek 5.24. Podgląd najnowszych artykułów prasowych na stronie głównej. Prezentowane teksy artykułów prasowych pochodzi z WP Sportowe Fakty [14]

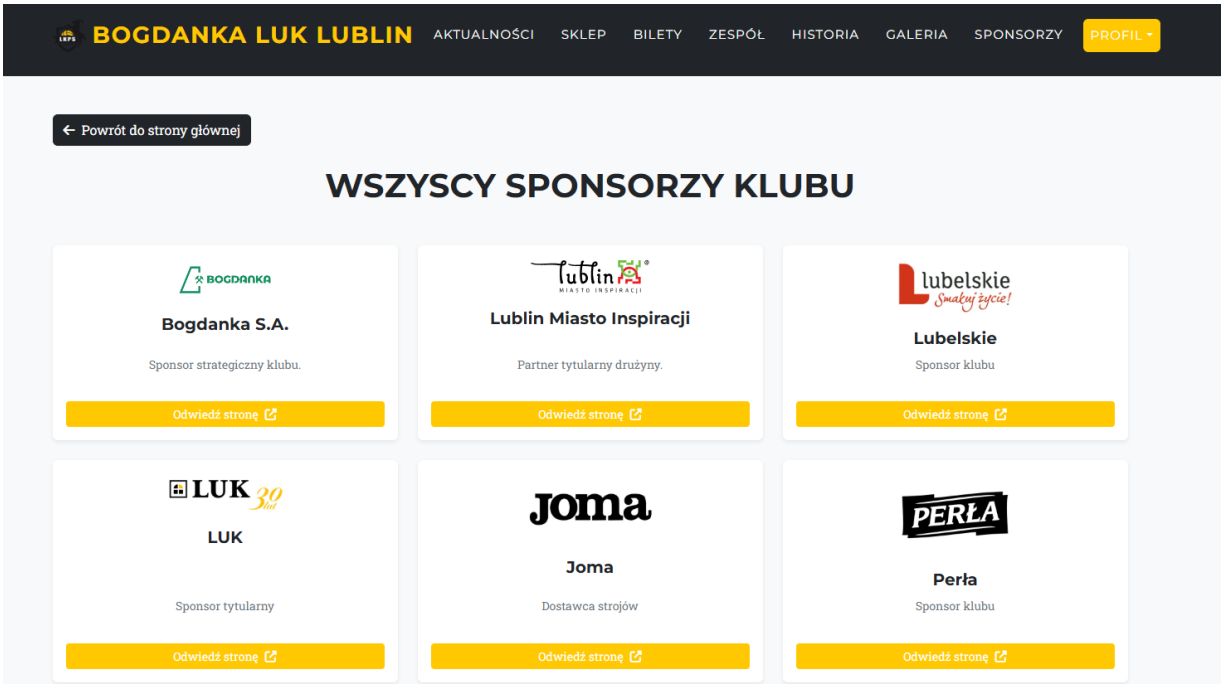
Rysunek 5.25. Widok historii klubu



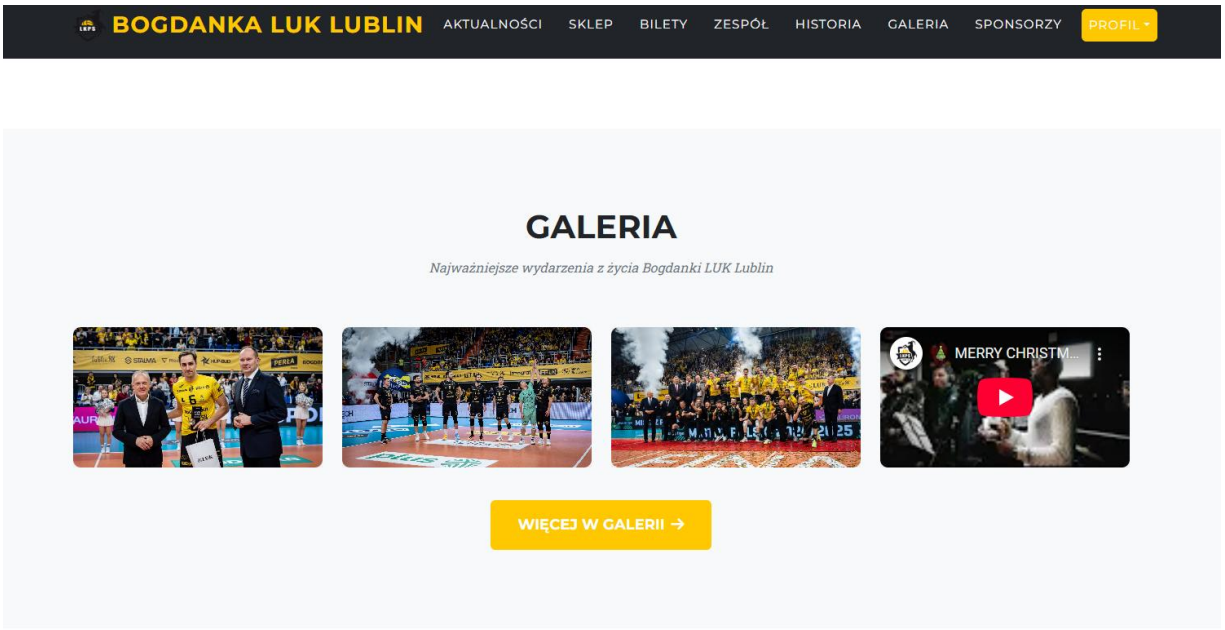
Rysunek 5.26. Funkcjonalność biletów na wydarzenia



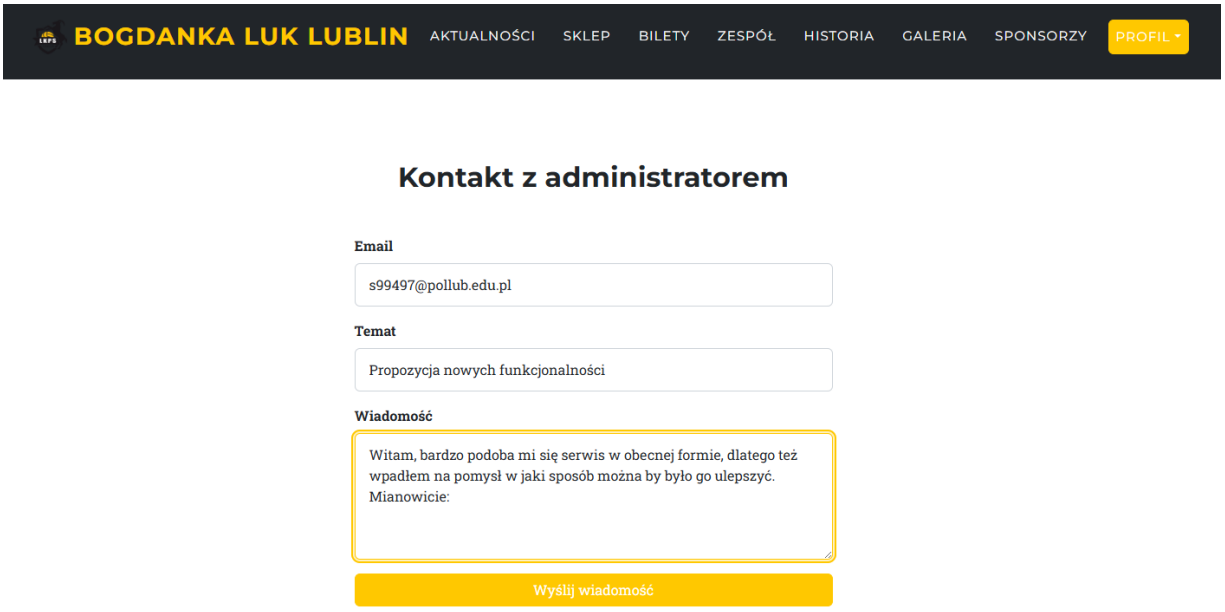
Rysunek 5.27. Funkcjonalność ankiet



Rysunek 5.28. Strona poświęcona sponsorom klubu



Rysunek 5.29. Podgląd galerii na stronie głównej



Rysunek 5.30. Formularz do kontaktu z administratorem

Różnice między projektem UI

Wraz z kolejnymi etapami projektowania aplikacji niewielkim zmianom uległa koncepcja wizualna projektu, jednakże widoki przedstawione w rozdziale 4.9. pomimo różnic stylistycznych zachowują funkcjonalność i przejrzystość, bazując na wcześniej utworzonych szkicach. Dostosowano poszczególne elementy strony, które łączą się z funkcjonalnościami jakie aplikacja oferuje odbiorcom, zachowując przy tym czytelność oraz odpowiedni kontrast. Widoki koncepcyjne poszczególnych stron pomogły zbudować odpowiedni interfejs użytkownika, który umożliwia intuicyjne poruszanie się po aplikacji.

6. Testowanie aplikacji

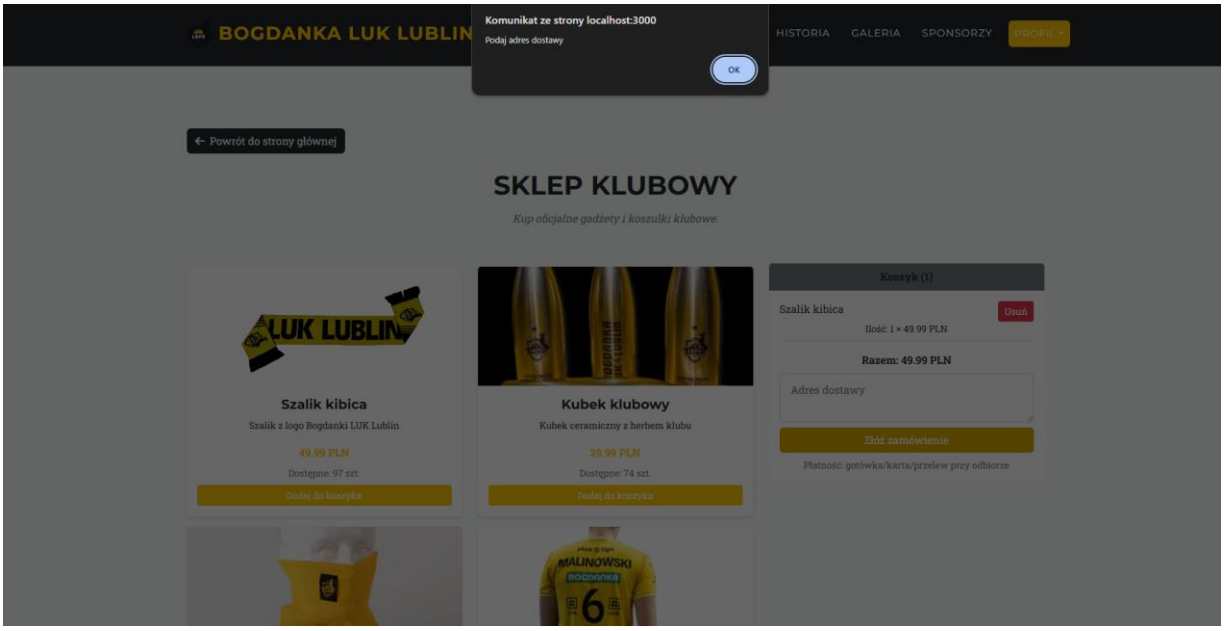
Testowanie utworzonej aplikacji jest kluczowe w procesie wytwarzania oprogramowania. Jego głównym celem jest sprawdzenie poprawności działania i eliminacja znalezionych błędów w funkcjonowaniu.

Do testowania prezentowanej aplikacji zastosowano testy manualne, które wykonywane są ręcznie poprzez interfejs graficzny aplikacji.

6.1. Testy manualne

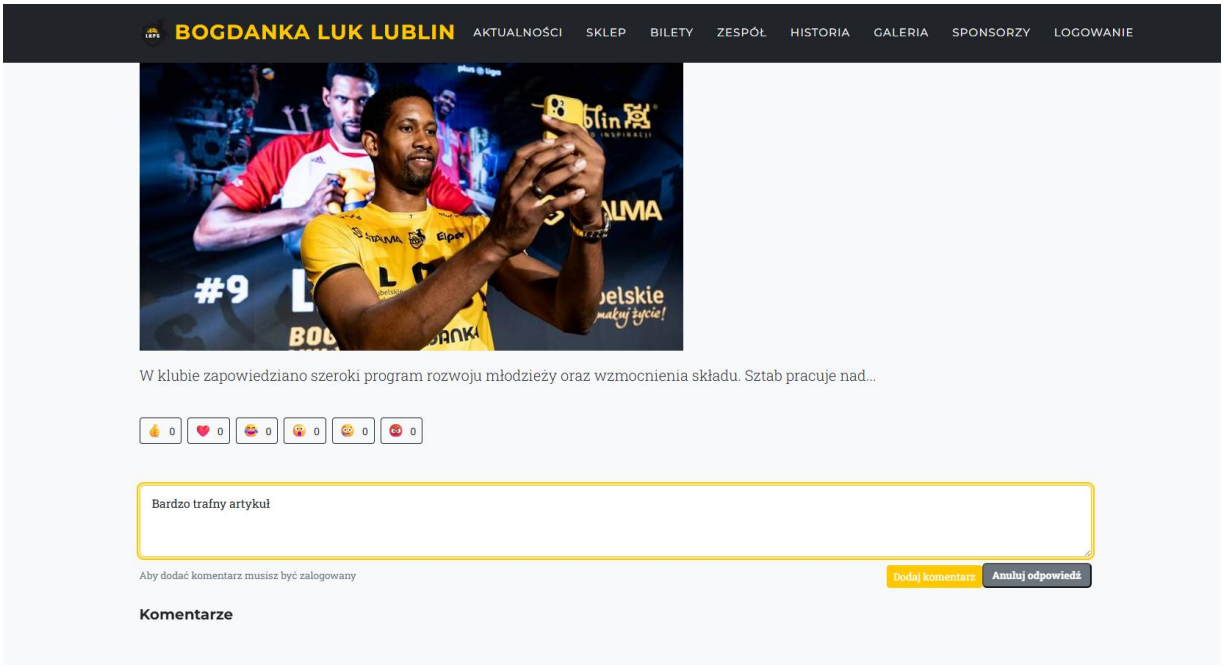
Testy manualne [6] wykonywane są ręcznie i polegają na sprawdzeniu elementów graficznego interfejsu użytkownika bez użycia narzędzi testowych czy skryptów. Wynikiem testów powinno być oczekiwane działanie aplikacji zgodne z dokumentacją techniczną i założeniami projektu.

Rysunek 6.1. przedstawiają testowanie funkcjonalności sklepu klubowego. Zalogowany użytkownik po dodaniu produktu do koszyka chcąc złożyć zamówienie musi wprowadzić swój adres. W przeciwnym wypadku pojawi się komunikat o braku danych w obowiązkowym polu, zamówienie nie zostanie złożone.

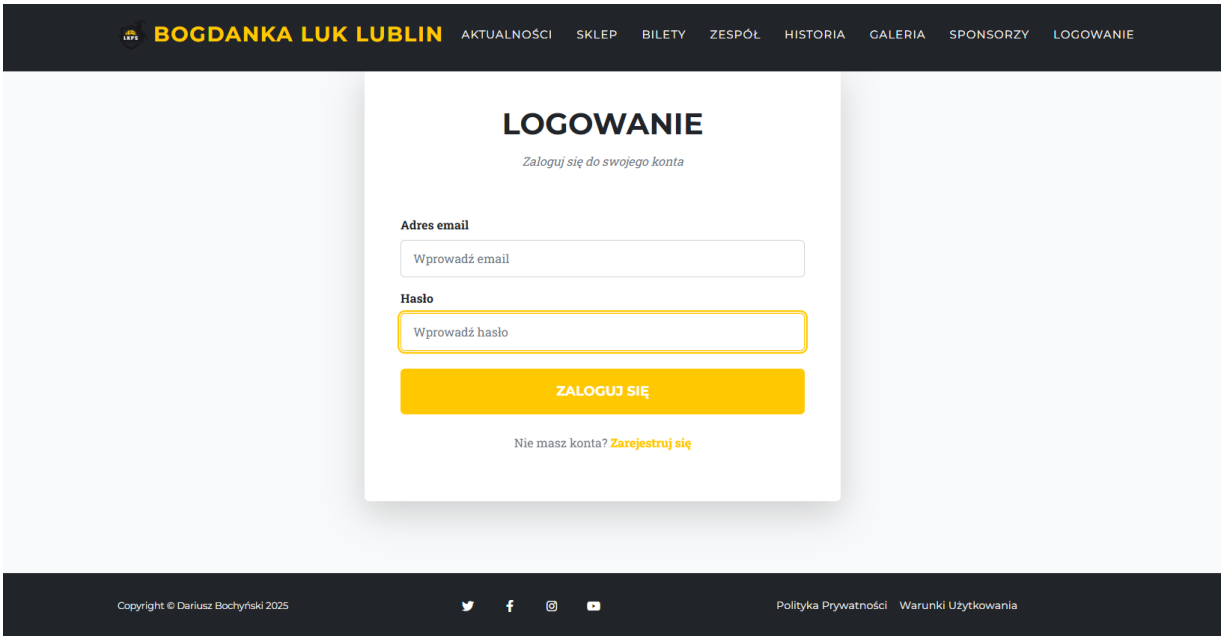


Rysunek 6.1. Testowanie funkcjonalności sklepu klubowego

Na rysunkach 6.2. – 6.3. przedstawiono testowanie funkcjonalności dodawania komentarzy pod artykułami prasowymi przez zalogowanych użytkowników. W momencie próby dodania komentarza przez użytkownika niezalogowanego aplikacja przekieruje go do okna logowania. W przedstawionej sytuacji komentarz nie zostanie dodany.

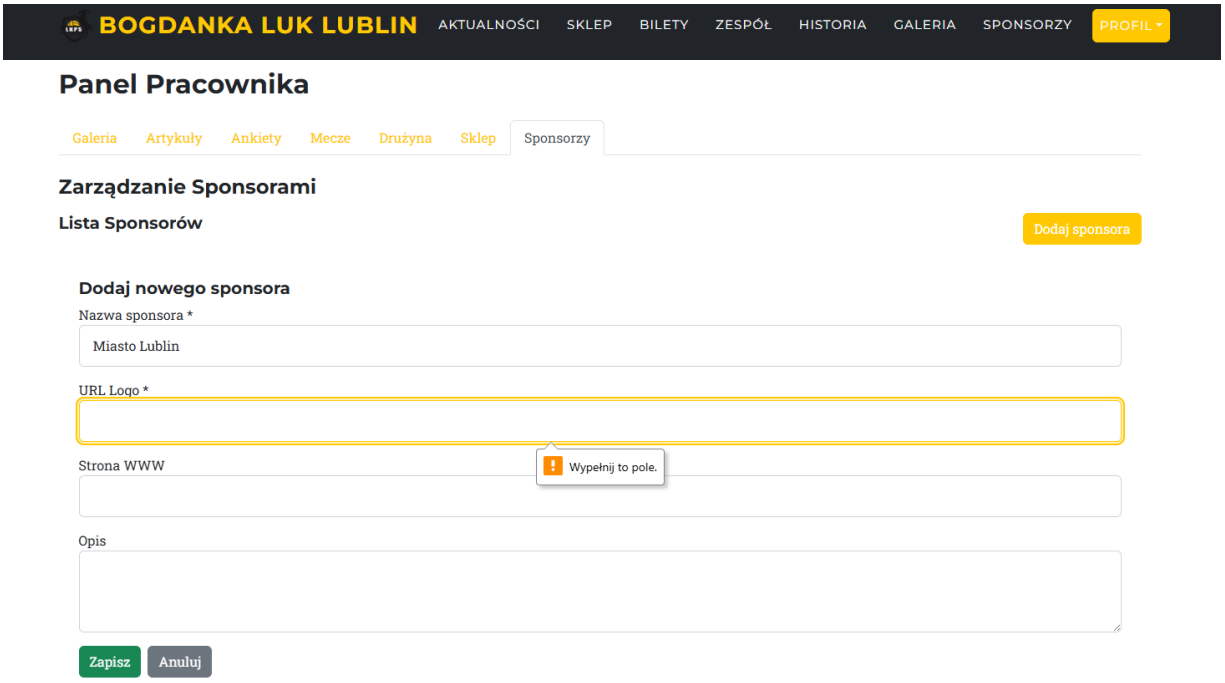


Rysunek 6.2. Testowanie funkcjonalności komentarzy pod artykułami prasowymi



Rysunek 6.3. Testowanie funkcjonalności komentarzy pod artykułami prasowymi cd.

Rysunek 6.4. przedstawia testowanie dodawanie sponsorów przez pracownika klubu. System sprawdza czy wszystkie wymagane pola zostały wypełnione. W przypadku braku obowiązkowych danych dla użytkownika wyświetlany jest komunikat oraz bez wypełnienia wymaganych pól dodanie elementu jest niemożliwe.



Rysunek 6.4. Testowanie dodawania sponsorów w panelu pracownika

7. Wnioski i podsumowanie

Celem pracy inżynierskiej było utworzenie aplikacji internetowej wspomagającej zarządzanie organizacją jaką jest klub siatkarski, spajając rozproszone instancje klubu w jeden spójny system. Cel ten został osiągnięty z wykorzystaniem założonych technologii.

Przykładową drużyną, na której zaprezentowano działanie systemu jest Bogdanka LUK Lublin. Klub wyraził pisemne pozwolenie na wykorzystywanie wizerunku na potrzeby pracy inżynierskiej.

W pracy został przedstawiony pełny proces tworzenia aplikacji internetowej, poczynając od definiowania założeń funkcjonowania aplikacji, aż po jej implementację oraz testy. Proces ten wymagał zapoznania się z istniejącymi już rozwiązaniami i technologiami oraz ich zrozumienia. Umożliwiło to utworzenie własnych autorskich rozwiązań, które zawierają dodatkowe funkcje takie jak na przykład podział na role o różnych uprawnieniach czy panele pracowników i administratorów pozwalające na zażądanie treściami zawartymi w serwisie. Ulepszają one aplikację, spełniając potrzeby wynikające z rynku.

Faza testów aplikacji pozwoliła na eliminacje drobnych błędów, które nie zostały odnalezione w poprzednich etapach wykonywania projektu oraz zasugerowały autorowi pracy, jakie aspekty mogłyby zostać ulepszone lub dodane w kolejnych wersjach w procesie rozwoju aplikacji. Jednym z takich pomysłów mogłoby być dodanie modułu dedykowanego dla trenera drużyny i zawodników, który pozwalałby na ustalanie wyjściowego składu przed meczem oraz integrację danych z dedykowanymi aplikacjami dla statystyków siatkarskich.

Obecna wersja aplikacji jest stabilna i gotowa do wdrożenia i późniejszego rozwoju. Jej komponentowa budowa sprzyja dodawaniu kolejnych modułów i funkcjonalności. Wdrożenie projektu z pewnością ułatwiłoby funkcjonowanie profesjonalnemu klubowi siatkarskiemu.

Proces tworzenia aplikacji rozwinął u autora umiejętność samodzielnej pracy oraz planowania obowiązków w czasie, co jest kluczowe pracach projektowych. Twórca poszerzył również swoją wiedzę na temat narzędzi i metod pracy stosowanych podczas rozwoju aplikacji internetowych oraz pracy z dokumentacją.

Bibliografia

[1] *Aluron CMC Warta Zawiercie – Oficjalna strona klubu.* Pobrane *22 czerwca 2025 z: https://www.aluroncmc.pl/*

[2] Archer R. (2016). *Express.js: Web App Development With Node.js Framework*. CreateSpace Independent Publishing Platform.

[3] *Bogdanka LUK Lublin – Oficjalna strona klubu*. Pobrane *22 czerwca 2025 z: https://lkpslublin.pl/*

[4] Bočaj N., Ahtik J. (2023). *Effects of visual complexity of banner ads on website users' perceptions*. Applied Sciences, MDPI.

[5] Chen S., Thaduri U.R., Ballamudi V.K.R. (2019). *Front-end development in React: an overview*. Engineering International.

[6] Dukes L., Yuan X., Akowuah F. (2013). *A case study on web application security testing with tools and manual testing*. Proceedings of IEEE Southeastcon, IEEE.

[7] Everett H.L. (2014). *Consistency & contrast: a content analysis of web design instruction*. Technical Communication, Society for Technical Communication.

[8] Hahn E. (2016). *Express in Action: Writing, Building, and Testing Node.js Applications*. Simon and Schuster.

[9] Hamidli N. (2023). *Introduction to UI/UX design: key concepts and principles*. Preuzeto.

[10] *JSW Jastrzębski Węgiel – Oficjalna strona klubu*. Pobrane *22 czerwca 2025 z: https://jastrzebskiwegiel.pl/*

[11] *PGE Projekt Warszawa – Oficjalna strona klubu*. Pobrane *22 czerwca 2025 z: https://projektwarszawa.waw.pl/*

[12] Šušter I., Ranisavljević T. (2023). *Optimization of MySQL database*. *Journal of Process Management and New Technologies*.

[13] Wahyudi J., Asbari M., Sasono I., Pramono T., Novitasari D. (2022). *Database management in MySQL*. Edumaspul – Jurnal Pendidikan.

[14] *WP Sportowe Fakty*. Pobrane *28 listopada 2025 z:https://sportowefakty.wp.pl/siatkowka/luk-politechnika-lublin*

[15] *ZAKSA Kędzierzyn-Koźle – Oficjalna strona klubu*. Pobrane *22 czerwca 2025 z: https://zaksa.pl/*