****

**Praca projektowa Bazy danych**  
**GUI Esport League Of Legends**

Autorzy: Wiktor Grzybowski, Krzysztof Adamski

*Wiktor Grzybowski*

Kierunek: Informatyka 2 rok 1 st, grupa lab 1

Rzeszów 2025

Spis treści

[1. Specyfikacja tematu projektu (przedstawienie projektowanej rzeczywistości) 2](#_Toc562566863)

[2. Aspekt projektowy bazy danych 3](#_Toc1570790735)

[3. Aspekt projektowy niezbędnych funkcjonalności 5](#_Toc1863855701)

[3.2. Funkcje zwracające zestawy wyników (SELECT) 7](#_Toc1436853620)

[3.3. Triggery 8](#_Toc1801896304)

[4. Przedstawienie powstałej bazy danych wraz z wdrożonymi mechanizmami 8](#_Toc1394815685)

[5. Koncepcja dostępu zdalnego do bazy danych 9](#_Toc1446738846)

[6. Opis realizacji dostępu zdalnego do bazy danych wraz z prezentacjąDostęp do bazy realizowany jest w następujący sposób: 10](#_Toc1021046024)

[7. Warstwa użytkowa projektu 12](#_Toc258806808)

[8. Przeniesienie bazy danych z relacyjnej na nierelacyjną8.1. Analiza i kroki do transportowania bazy relacyjnej na model nierelacyjny (słownie) 20](#_Toc28150562)

[9. PodsumowanieProjekt “E-sport League of Legends” demonstruje: 22](#_Toc1456558148)

[10. Literatura 23](#_Toc597474119)

1. **Specyfikacja tematu projektu (przedstawienie projektowanej rzeczywistości)**

Projekt dotyczy tworzenia i zarządzania bazą danych poświęconą e-sportowi w grze **League of Legends**. W świecie e-sportu funkcjonują różne drużyny (Teamy) z określonych regionów (Europa, Ameryka Północna, Korea Południowa itp.), biorące udział w turniejach (zarówno regionalnych, jak i międzynarodowych). Każda drużyna ma swoich zawodników (graczy), trenerów oraz dokonuje transferów zawodników pomiędzy drużynami. Mecze są komentowane przez komentatorów, a sama gra pozwala zawodnikom wybierać różne postacie (championy). W kontekście e-sportu kluczowe jest śledzenie statystyk meczowych, takich jak liczba wygranych, przegranych, zabójstw, asyst, zgonów itp.

Główne elementy świata przedstawionego:

* **Drużyny** – uczestnicy rozgrywek, posiadają określoną nazwę i pochodzą z danego regionu.
* **Regiony** – np. EU, NA, KR, CN, BR itd., każdy region posiada nazwę i skrót.
* **Gracze** – należą do drużyn, mają statystyki (wygrane, przegrane, eliminacje, zgony, asysty) i przypisane pozycje w grze (Top, Jungle, Mid, ADC, Support).
* **Trenerzy** – są przypisani do drużyn, odpowiadają za szkolenie i rozwój graczy.
* **Komentatorzy** – osoby komentujące rozgrywki, przypisywane do poszczególnych meczów.
* **Turnieje** – imprezy e-sportowe, w których drużyny z danego lub różnych regionów konkurują o tytuły.
* **Mecze** – rozgrywki w ramach turniejów, z udziałem komentatora, a także konkretnych graczy i wybieranych przez nich postaci.
* **Postacie** (championy w LoL-u) – gracze wybierają je podczas meczu.
* **Transfery** – przeniesienia zawodników z jednej drużyny do innej (zawierają informację o tym, jaki gracz przeszedł z drużyny1 do drużyny2).

Celem projektu jest umożliwienie:

1. Przechowywania informacji o drużynach, graczach, trenerach, komentatorach i turniejach.
2. Rejestrowania meczów i transferów zawodników.
3. Prowadzenia statystyk, w tym automatycznej aktualizacji drużyny zawodnika po ogłoszeniu transferu.
4. Dokonywania analizy danych (top gracze KDA, najwięcej wygranych, najczęściej wybierane postacie, turnieje w poszczególnych regionach itp.).
5. Udostępnienia interfejsu zdalnego (poprzez aplikację .NET/Blazor/Entity Framework) do zarządzania i wyświetlania danych.
6. **Aspekt projektowy bazy danych**
   * Baza danych została zaprojektowana w **PostgreSQL**. Najważniejsze tabele:

* **druzyny** (druzynaid, nazwa, regionid)
  + Przechowuje podstawowe informacje o drużynach.
  + Powiązana przez klucz obcy regionid z tabelą **regiony**.
* **regiony** (regionid, nazwa, skrot)
  + Określa regiony (np. Europe, EU).
  + Zawiera unikalne ID regionu, nazwę i skrót.
* **gracze** (graczid, nickname, druzynaid, przegrane, wygrane, eliminacje, asysty, zgony, pozycja)
  + Przechowuje informacje o każdym graczu (m.in. ID drużyny, statystyki, pozycja).
  + Powiązanie przez druzynaid z tabelą **druzyny**.
  + W tabeli gracze znajduje się trigger pozwalający na usuwanie powiązanych transferów przed fizycznym usunięciem gracza (tzw. *on delete trigger*).
* **trenerzy** (trenerid, imie, nazwisko, druzynaid)
  + Informacje o trenerach, którzy są przypisani do konkretnej drużyny.
* **komentatorzy** (komentatorid, imie, nazwisko, nickname)
  + Dane o komentatorach, którzy mogą być przypisani do meczów.
* **mecze** (meczid, komentatorid, turniejid)
  + Każdy mecz przypisany jest do turnieju i komentatora.
  + Pośrednio łączy graczy z postaciami przez tabelę **xyz** (opis niżej).
* **postacie** (postacid, nazwa)
  + Spis wszystkich championów (postaci) dostępnych w LoL.
* **turnieje** (turniejid, nazwaturnieju, rodzaj, organizator, adres, regionid, data)
  + Lista turniejów wraz z informacją o organizatorze, adresie (lokalizacji), przypisaniu do regionu i dacie.
* **transfery** (transferid, graczid, druzyna1, druzyna2)
  + Rejestruje przeniesienia zawodników pomiędzy drużynami.
  + W bazie zaimplementowano **trigger** tr\_transfer\_gracza wywoływany po wstawieniu nowego transferu. Powoduje on automatyczną aktualizację drużyny gracza w tabeli gracze (graczid zostaje przypisany do nowej drużyny - druzyna2).
* **xyz** (uniqueid, graczid, meczid, postacid, wynik)
  + Tabela łącząca informację o tym, **kto** (gracz) grał w **którym** meczu i **jaką** postacią.
  + Dodatkowo wynik może przechowywać np. punktację, informację o zwycięstwie itp.
  + Każda z tabel posiada klucz główny (PRIMARY KEY), a klucze obce (FOREIGN KEY) gwarantują spójność danych (np. gracze(druzynaid) -> druzyny(druzynaid)).

**Nadrzędnym założeniem projektu** było utworzenie relacji pozwalających na:

* łatwe śledzenie przynależności graczy do drużyn i regionów,
* automatyczną obsługę transferów (zmiana przynależności gracza w momencie wstawienia rekordu w tabeli **transfery**),
* szeroką analitykę (statystyki meczowe, turniejowe) dzięki zestawowi **Funkcji** i **Procedur** w bazie danych,
* integrację z warstwą aplikacji w .NET (Blazor + EF Core).

1. **Aspekt projektowy niezbędnych funkcjonalności**

**3.1. Procedury składowane**

1. **dodajDruzyne(p\_nazwa, p\_regionid)**
2. Dodaje nowy rekord do tabeli Druzyny.
3. **dodajGracza(p\_nickname, p\_druzynaid, p\_pozycja)**

Dodaje nowy rekord do tabeli Gracze.

1. **dodajMecz(p\_komentatorid, p\_turniejid)**

Tworzy nowy mecz, przypisując go do wskazanego komentatora i turnieju.

1. **dodajPostac(p\_nazwa)**

Dodaje nową postać do tabeli Postacie.

1. **dodajRegion(p\_nazwa, p\_skrot)**

Dodaje region o wskazanych parametrach.

1. **dodajTransfer(p\_graczid, p\_druzyna1, p\_druzyna2)**

Dodaje rekord w tabeli Transfery. **Po dodaniu** wywołuje się *trigger* tr\_transfer\_gracza, który aktualizuje drużynę gracza.

1. **dodajTrenera(p\_imie, p\_nazwisko, p\_druzynaid)**

Dodaje trenera do danej drużyny.

1. **dodajTurniej(...)**

Tworzy nowe wydarzenie (turniej).

1. **edytujDruzyne(p\_druzynaid, p\_nazwa, p\_regionid)**

Edytuje istniejącą drużynę.

1. **edytujGracza(...)**

Zmienia dane gracza, w tym statystyki (przegrane, wygrane, zabójstwa, asysty, zgony).

1. **edytujKomentatora(...)**

Zmienia dane komentatora (imię, nazwisko, nickname).

1. **edytujMecz(...)**

Zmienia przypisanie meczu do innego komentatora lub turnieju.

1. **edytujPostac(...)**

Aktualizuje nazwę postaci.

1. **edytujRegion(...)**

Zmienia parametry regionu (np. skrót).

1. **edytujTransfer(...)**

Zmienia dane dotyczące transferu (kogo i dokąd przeniesiono).

1. **edytujTrenera(...)**

Zmienia dane trenera.

1. **edytujTurniej(...)**

Aktualizuje informacje o turnieju.

1. **usunDruzyne(p\_druzynaid)**

Usuwa drużynę o podanym ID.

1. **usunGracza(p\_graczid)**

Usuwa wybranego gracza.

1. **usunKomentatora(p\_komentatorid)**

Usuwa komentatora.

1. **usunRegion(p\_regionid)**

Usuwa region (np. jeśli dany region nie jest już używany).

1. **usunTrenera(p\_trenerid)**

Usuwa trenera.

*(Procedury usuwające: w przypadku usunGracza uruchamiany jest trigger kasujący transfery powiązane z tym graczem.)*

### **3.2. Funkcje zwracające zestawy wyników (SELECT)**

* **get\_all\_teams(), get\_all\_players(), get\_all\_regions()**

Pozwalają pobrać wszystkie drużyny, graczy, regiony.

* **get\_all\_tournaments(), get\_all\_matches(), get\_all\_coaches(), get\_all\_commentators(), get\_all\_characters(), get\_all\_transfers()**

Pozwalają uzyskać *pełne* dane z poszczególnych tabel (lub tzw. widoków logicznych).

* **get\_players\_with\_teams()**

Zwraca listę graczy wraz z nazwą drużyny (JOIN).

* **get\_coaches\_by\_region(regionid)**

Zwraca trenerów, których drużyny są z określonego regionu.

* **get\_teams\_by\_region(regionid)**

Zwraca drużyny przypisane do określonego regionu.

* **get\_top\_kda\_players()**

Oblicza wskaźnik KDA (K+A)/Deaths lub (K+A), jeśli 0 zgonów, i zwraca TOP 10.

* **get\_top\_winners()**

Zwraca graczy z największą liczbą wygranych.

* **get\_most\_picked\_characters(), get\_most\_picked\_character()**

Zwracają informację o najczęściej wybieranych postaciach w meczach.

* **get\_best\_team\_in\_tournament(turniejid)**

Określa najlepszą drużynę w danym turnieju na podstawie liczby wygranych.

* **get\_tournaments\_count\_by\_region()**

Zwraca, ile turniejów odbywa się w poszczególnych regionach.

* **get\_teams\_with\_most\_players()**

Drużyny z największą liczbą graczy.

* **get\_teams\_with\_most\_transfers()**

Drużyny najczęściej biorące udział w transferach (zarówno odchodzący, jak i przychodzący gracz).

* **get\_average\_kills\_and\_deaths\_by\_player()**

Oblicza średnie zabójstwa i zgony na gracza.

* **get\_top\_assisters\_by\_team()**

Top 5 graczy z największą liczbą asyst.

* **get\_match\_count\_by\_commentator()**

Informacja, ilu meczom dany komentator przewodził.

### **3.3. Triggery**

1. **tr\_transfer\_gracza** (AFTER INSERT ON **transfery**)
   1. Po wstawieniu nowego transferu przenosi gracza do drużyny docelowej (druzyna2) w tabeli gracze.
2. **trigger\_usun\_transfery** (BEFORE DELETE ON **gracze**)
   1. Przed usunięciem gracza usuwa wszystkie rekordy w tabeli **transfery**, w których uczestniczył.
3. **Przedstawienie powstałej bazy danych wraz z wdrożonymi mechanizmami**

**Tabela regiony**

* Kolumny: regionid (PK), nazwa, skrot.
* Przykładowe dane:

**Tabela druzyny**

* Kolumny: druzynaid (PK), nazwa, regionid (FK -> regiony.regionid).
* Przykładowe dane:

**Tabela gracze**

* + Kolumny: graczid (PK), nickname, druzynaid (FK), przegrane, wygrane, eliminacje, asysty, zgony, pozycja.
  + Tabela powiązana jest z transfery triggerem, który usuwa wpisy o transferach przy usunięciu gracza.

**Tabela trenerzy**

* + Kolumny: trenerid (PK), imie, nazwisko, druzynaid (FK).

**Tabela komentatorzy**

* + Kolumny: komentatorid (PK), imie, nazwisko, nickname.

**Tabela mecze**

* + Kolumny: meczid (PK), komentatorid (FK -> komentatorzy), turniejid (FK -> turnieje).

**Tabela turnieje**

* + Kolumny: turniejid (PK), nazwaturnieju, rodzaj, organizator, adres, regionid (FK -> regiony), data.

**Tabela postacie**

* + Kolumny: postacid (PK), nazwa.

**Tabela transfery**

* + Kolumny: transferid (PK), graczid (FK -> gracze), druzyna1 (FK -> druzyny), druzyna2 (FK -> druzyny).
  + Trigger tr\_transfer\_gracza (AFTER INSERT) – aktualizuje druzynaid w gracze.

**Tabela xyz**

* + Kolumny: uniqueid (PK), graczid (FK -> gracze), meczid (FK -> mecze), postacid (FK -> postacie), wynik.
  + Tabela łącząca kluczowe informacje o przebiegu meczu.

**4.2. Mechanizmy zaimplementowane w bazie**

* **Procedury** (dodaj, edytuj, usuń) – Umożliwiają enkapsulację logiki przytypowych operacjach CRUD.
* **Funkcje** – Zapewniają złożone zestawienia i analizy danych (statystyki, rankingi).
* **Triggery** – Automatyzują operacje (np. przeniesienie gracza do nowej drużyny, usunięcie powiązanych danych).

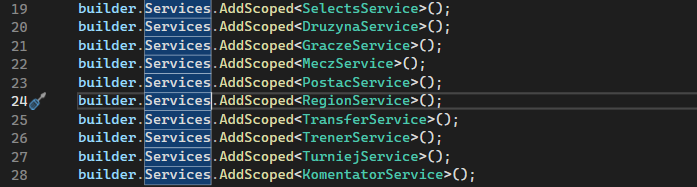
1. **Koncepcja dostępu zdalnego do bazy danych**

Aplikacja została stworzona w **.NET (Blazor Server)** i komunikuje się z bazą danych PostgreSQL poprzez **Entity Framework Core**. Dostęp zdalny zakłada:

1. **Warstwę API / warstwę usług**:
   1. W projekcie zaimplementowano klasy *Service* (np. DruzynaService, GraczeService, KomentatorService itd.), które są odpowiedzialne za wykonywanie procedur po stronie bazy.
   2. Każdy *service* wykorzystuje obiekt AppDbContext i metody (funkcje async) wywołujące procedury SQL bądź odczytujące dane.
2. **Konfigurację połączenia**:
   1. W pliku konfiguracyjnym .json (lub w appsettings.json) znajduje się connectionString do bazy.
   2. EF Core pobiera connectionString w metodzie UseNpgsql(connectionString).
3. **Model warstwy prezentacji**:
   1. W Blazorze używane są strony *.razor*, które wstrzykują (przez [Inject]) konkretne serwisy (SelectsService, DruzynaService itp.).
   2. Metody w serwisach są wywoływane przez operacje w interfejsie (np. w OnInitializedAsync, w metodach obsługi formularza).
   3. Wyniki (np. listy graczy, drużyn) są renderowane w postaci tabel.
4. **Bezpieczeństwo**:
   1. Aplikacja może być wdrożona (deploy) na serwerze z certyfikatem SSL, łącząc się z bazą.
   2. Mechanizmy autoryzacji i autentykacji (nie zostały tu szczegółowo uwzględnione w kodzie, ale można je rozszerzyć).
5. **Planowane funkcjonalności w dostępie zdalnym**:
   1. Panel administracyjny do zarządzania (CRUD) elementami bazy: gracze, drużyny, transfery, mecze, turnieje itp.
   2. Publiczny wgląd w statystyki (np. widok: “TOP gracze”, “najczęściej wybierane postacie”).
   3. Ewentualna integracja z zewnętrznymi usługami e-sportowymi.
6. **Opis realizacji dostępu zdalnego do bazy danych wraz z prezentacją**  
   **Dostęp do bazy realizowany jest w następujący sposób:**

Konfiguracja EF Core

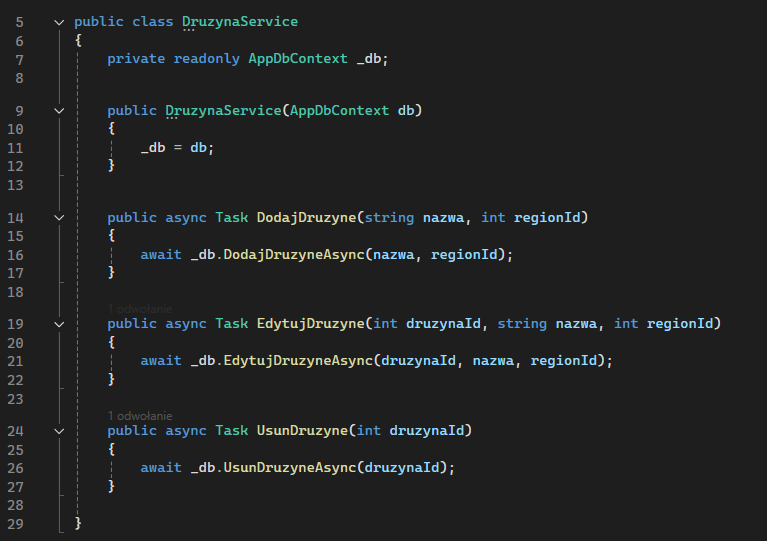
W Program.cs (startowym) konfigurujemy usługi:

Następnie rejestrujemy serwisy  
**AppDbContext**

Klasa AppDbContext dziedziczy po DbContext i zawiera metody wywołujące procedury/funkcje SQL (np. DodajDruzyneAsync, EdytujGraczaAsync). Wewnątrz używany jest Database.ExecuteSqlRawAsync(...) do wywołania procedur.

**Warstwa Service**

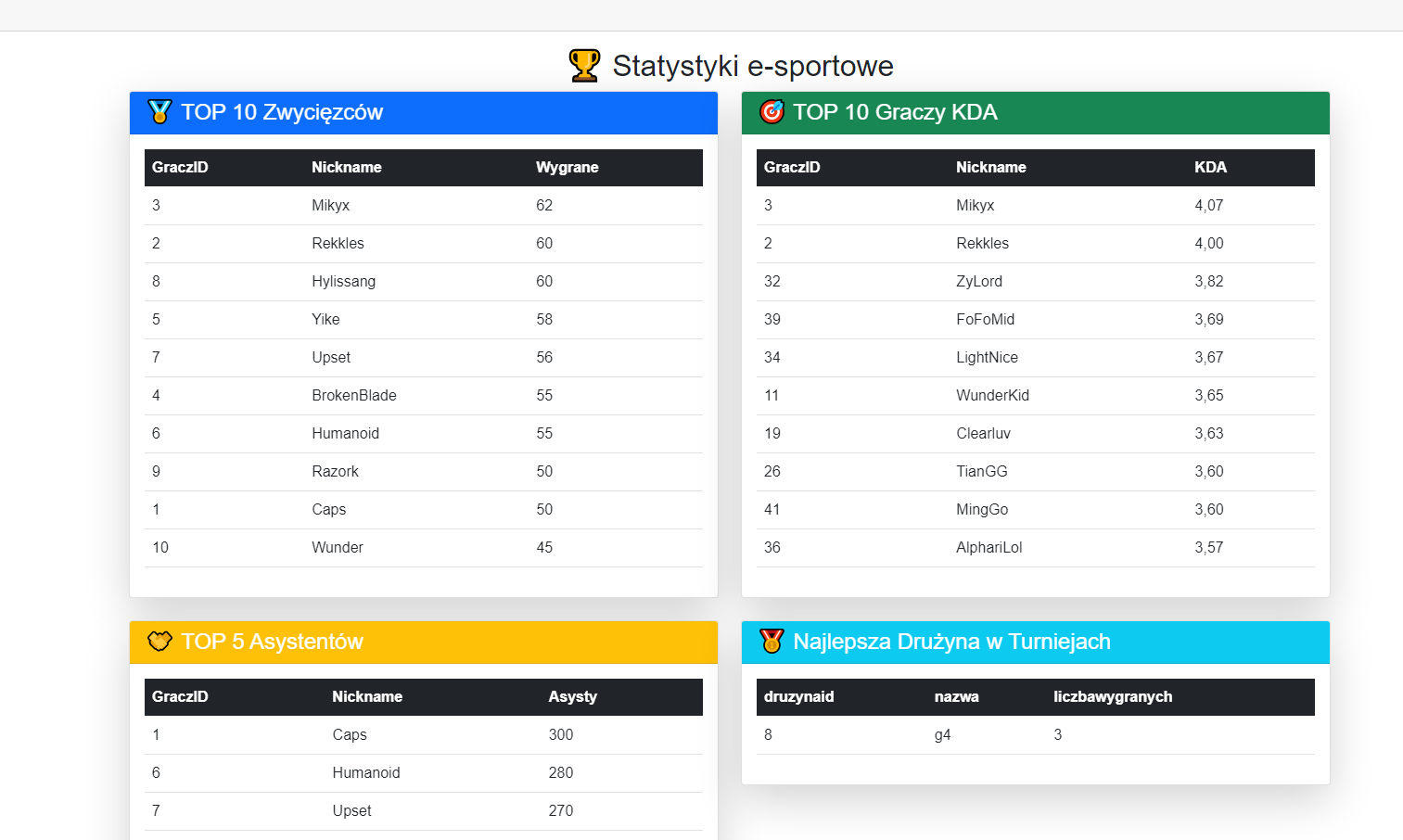
Przykład:

**Warstwa prezentacji (Blazor .razor)**

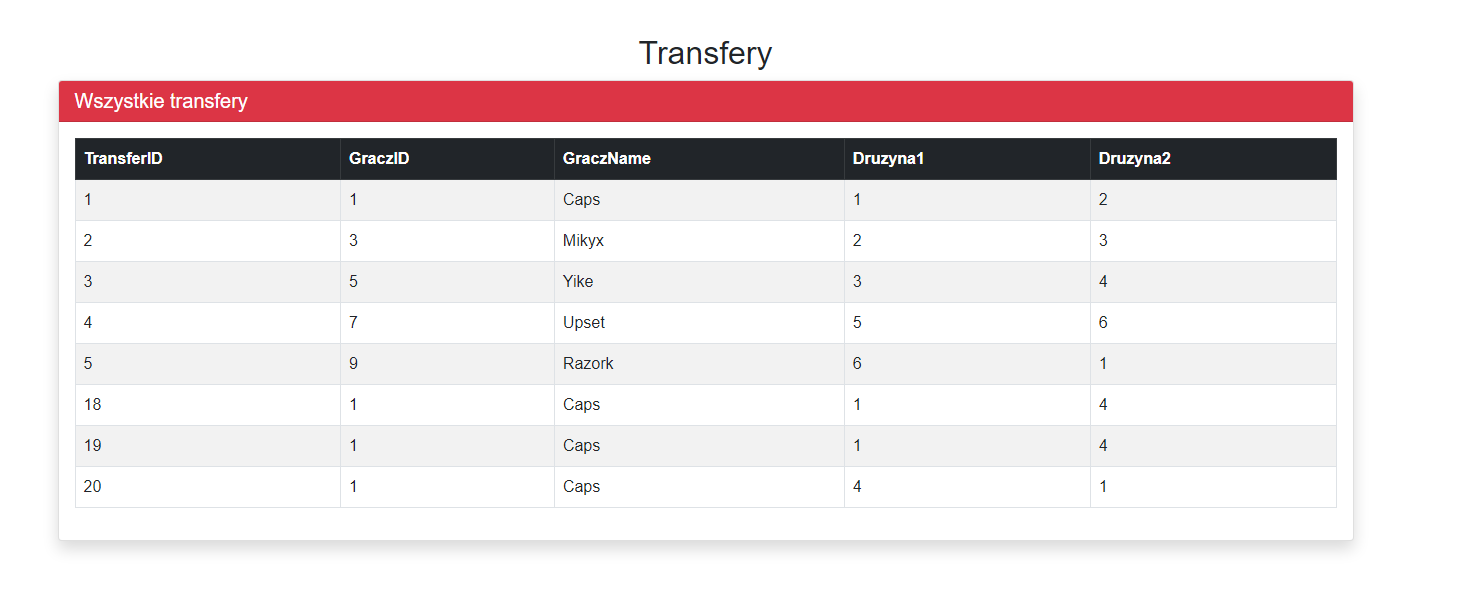
* Każda strona (.razor) posiada metodę OnInitializedAsync(), w której np. wywołujemy SelectsService w celu pobrania danych (np. wszystkich drużyn, regionów).
* Dostęp do CRUD jest zrealizowany przez EditForm, który wypełniamy danymi, a wywołanie przycisku submit uruchamia metodę z serwisu (np. DodajDruzyne).

1. **Warstwa użytkowa projektu**

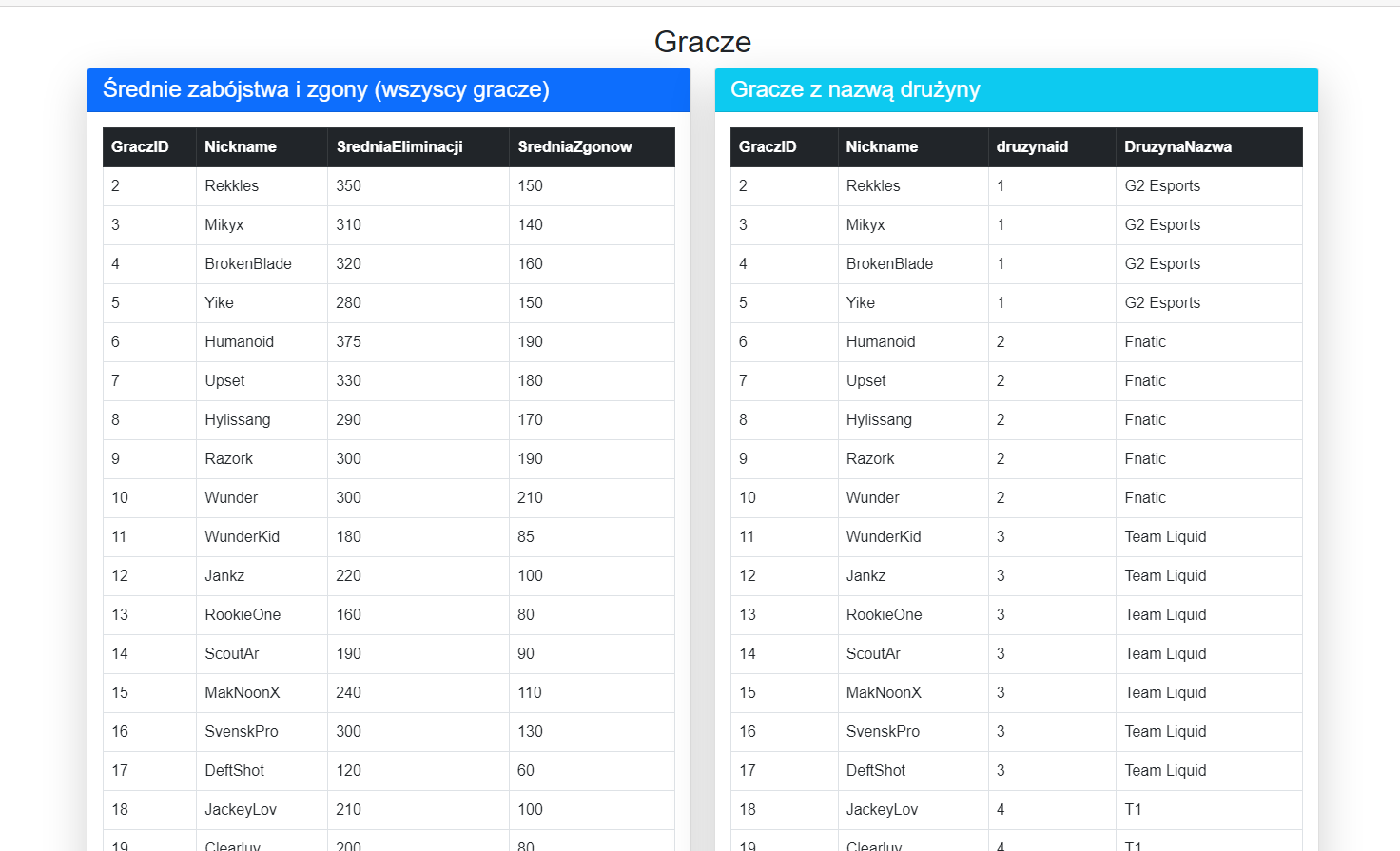
**7.1. Strona główna ("/")** Cel: Strona główna wyświetla najważniejsze statystyki e-sportowe w formie kart (tzw. widgetów), np. TOP zwycięzcy, TOP KDA, TOP asystenci, najlepsza drużyna w turnieju, drużyna z największą liczbą transferów itp. Mechanizm: Przy renderowaniu strona wywołuje szereg metod w warstwie usług (SelectsService), które pobierają dane z funkcji bazy danych (np. get\_top\_kda\_players(), get\_best\_team\_in\_tournament()). Następnie wyświetlane są w formie tabel w kartach Bootstrap. Przykładowe elementy na stronie: Kafelek 🏅 TOP 10 Zwycięzców – pokazuje listę graczy z największą liczbą wygranych. Kafelek 🎯 TOP 10 Graczy KDA – gracze z najlepszym wskaźnikiem KDA. Kafelek 🥇 Najlepsza Drużyna w Turniejach – wynik funkcji get\_best\_team\_in\_tournament(...).

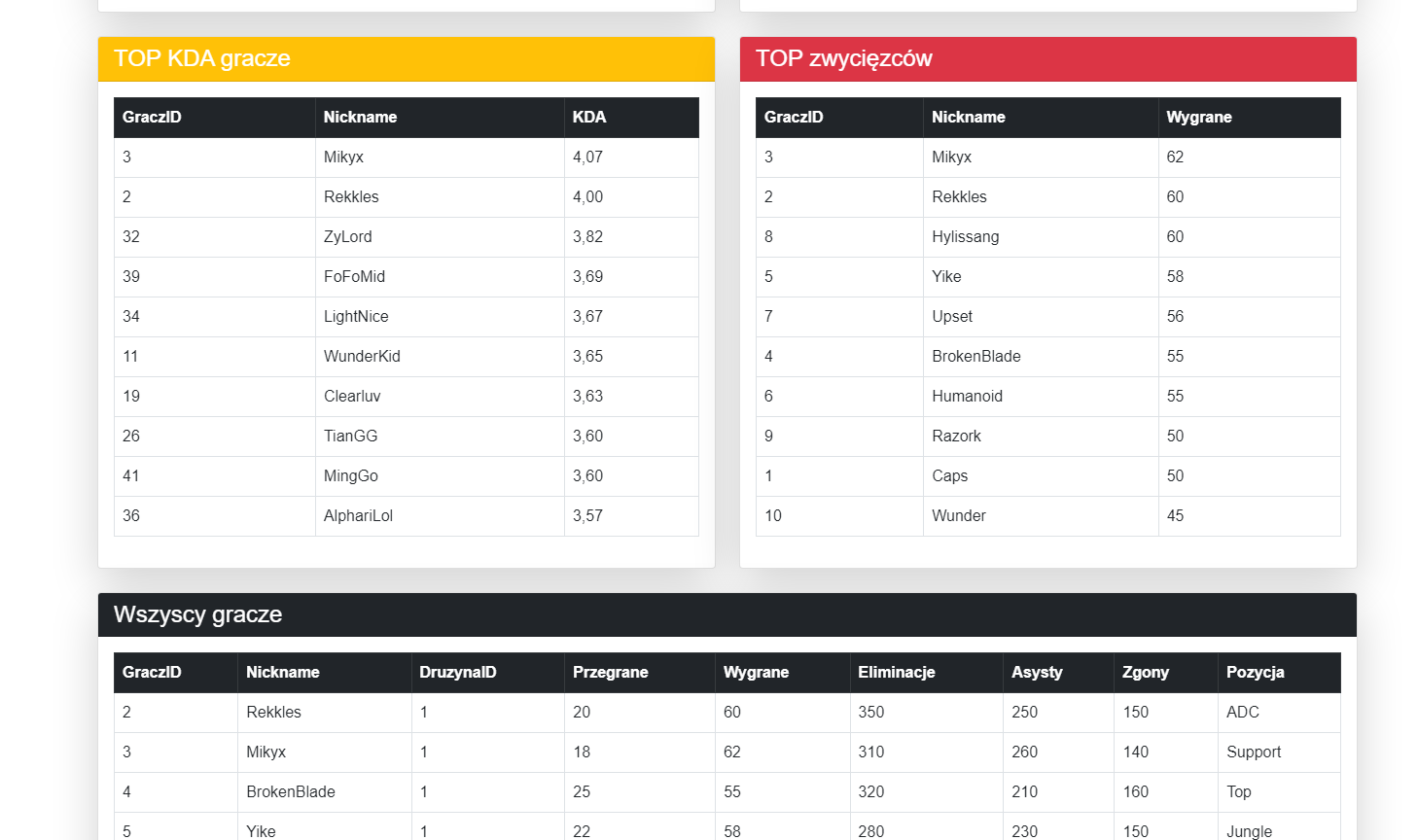


**7.2. Podstrona „/Transfery”** Cel: Prezentuje wszystkie transfery graczy między drużynami. Mechanizm: Wczytuje dane z funkcji bazy get\_all\_transfers(), a następnie wyświetla w formie tabeli z takimi kolumnami jak TransferID, GraczID, GraczName, Druzyna1, Druzyna2. Co widać: Użytkownik może zobaczyć, kto (który gracz) i z której drużyny przeszedł do jakiej drużyny.



**7.3. Podstrona „/Gracze”** Cel: Wyświetla listę wszystkich graczy oraz różne statystyki związane z graczami. Mechanizm: Przy wczytaniu wykonuje zapytania do bazy: get\_average\_kills\_and\_deaths\_by\_player() – oblicza średnią eliminacji i zgonów. get\_players\_with\_teams() – gracze wraz z nazwami drużyn. get\_top\_kda\_players(), get\_top\_winners(). get\_all\_players() – pełna lista wszystkich graczy. Co widać: Każda sekcja jest osobno prezentowana w formie karty i tabeli. M.in. można przejrzeć szczegółowe statystyki wszystkich zawodników z bazy.

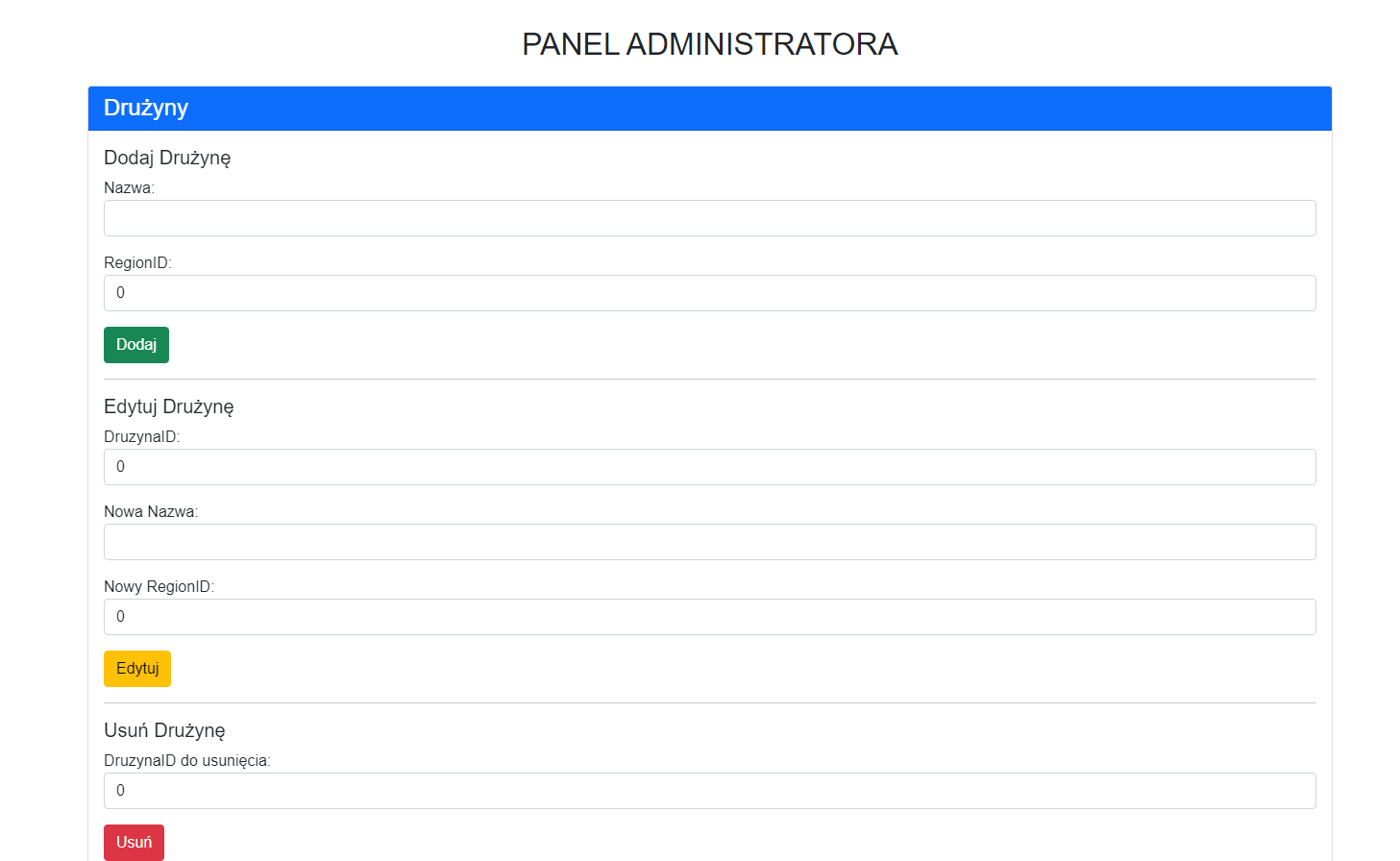




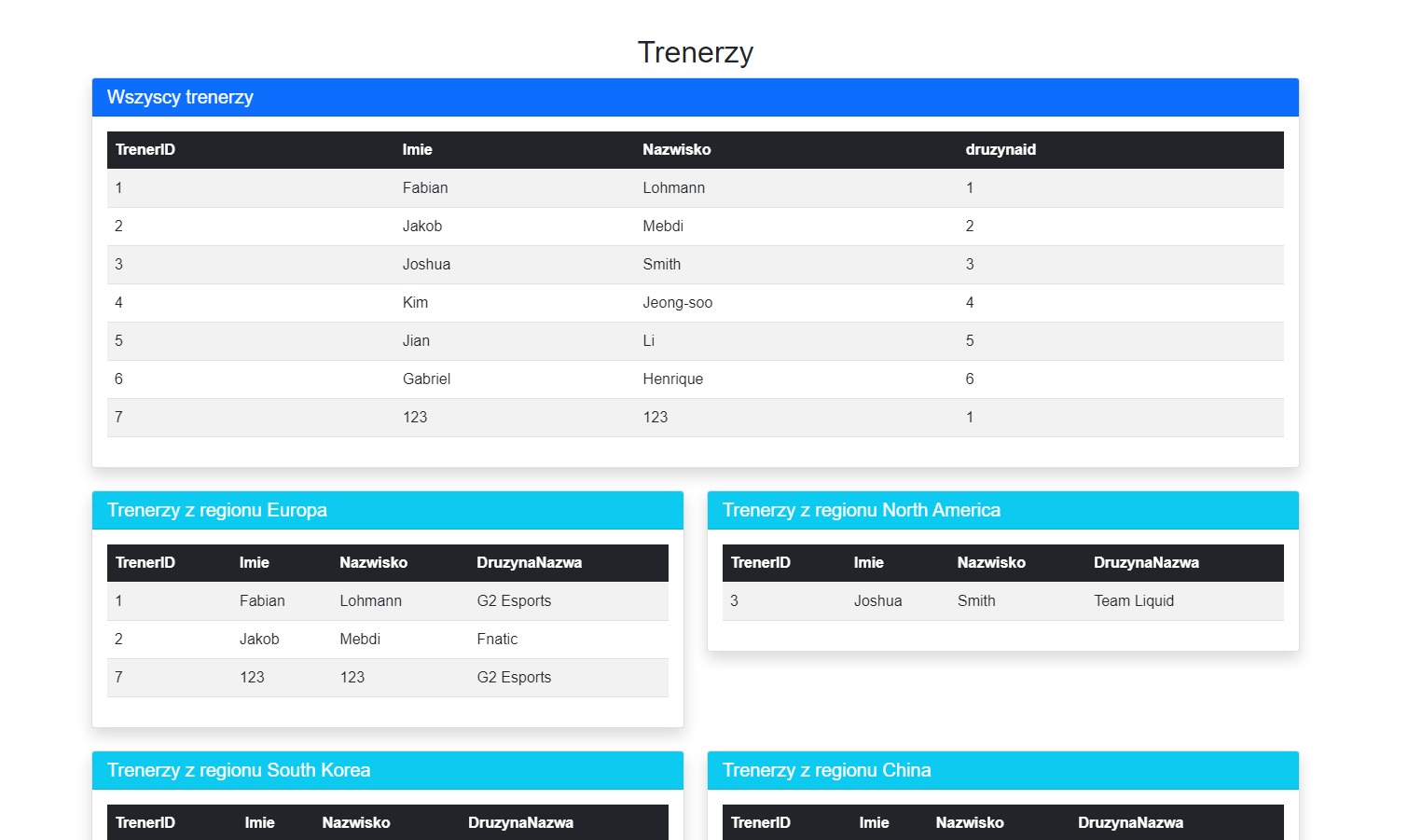
**7.4. Podstrona „/Mecze”** Cel: Przedstawia wszystkie mecze zapisane w bazie, wraz z ich ID, przydzielonym komentatorem i turniejem. Mechanizm: Korzysta z metody get\_all\_matches(), która dodatkowo mapuje dane z tabeli xyz (łączenie gracza i postaci). W efekcie wyświetlana jest lista meczów oraz (opcjonalnie) uczestniczące drużyny i gracze. Co widać: Tabela “Wszystkie mecze” z kolumnami MeczID, KomentatorID, TurniejID, a także z zestawieniem drużyn i graczy uczestniczących w danym meczu (jeśli zaimplementowano widok łączony).



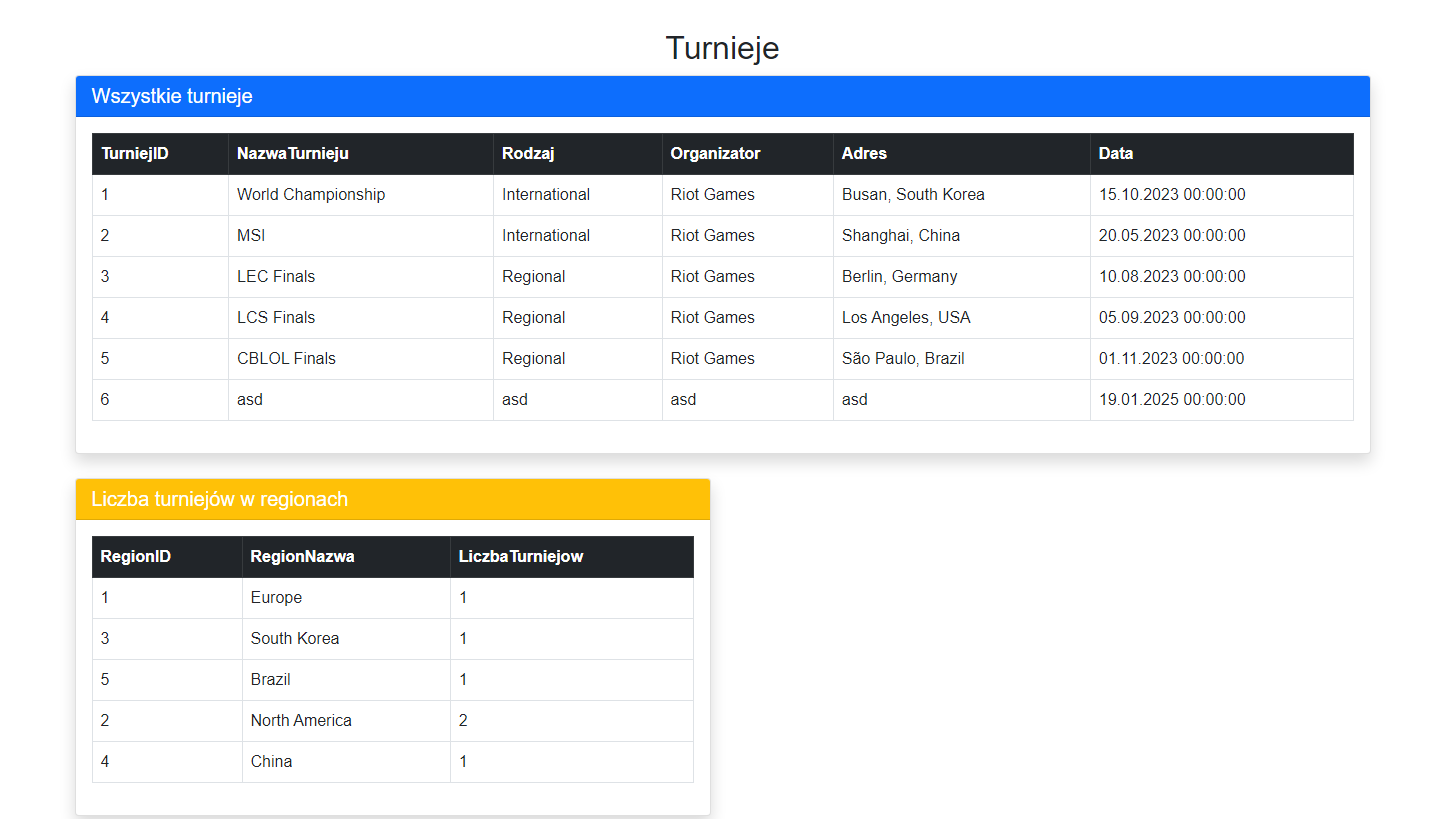
**7.5. Podstrona „/panel-admin”** (Panel Administratora) Cel: Umożliwia wykonywanie operacji CRUD (tworzenie, modyfikacja, usuwanie) na wszystkich głównych obiektach bazy danych (drużyny, gracze, mecze, postacie, regiony, transfery, trenerzy, turnieje, komentatorzy). Mechanizm: W każdej karcie (np. “Drużyny”, “Gracze”, “Komentatorzy” itd.) znajdują się trzy formularze: Dodawanie (Add), Edycja (Edit), Usunięcie (Delete). Po wysłaniu formularza (np. “AddTeamForm”) wywołuje się odpowiednia metoda w serwisie (DruzynaService, GraczeService itp.), która dalej wywołuje procedurę w bazie (np. dodajdruzyne). Co widać: Każda sekcja ma osobny blok interfejsu, np. “Drużyny” – Dodaj Drużynę, Edytuj Drużynę, Usuń Drużynę. “Gracze” – analogiczne formularze do dodania/edytowania/usuwania gracza. “Komentatorzy”, “Mecze”, “Regiony”, “Postacie”, “Transfery”, “Trenerzy”, “Turnieje”.



**7.6. Podstrona „/Trenerzy”** Cel: Wyświetla listę wszystkich trenerów (funkcja get\_all\_coaches()) oraz trenerów w poszczególnych regionach (funkcja get\_coaches\_by\_region(...)). Co widać: Tabele segregujące trenerów – np. “Trenerzy z regionu Europa”, “Trenerzy z regionu North America” itd. Mechanizm: Każda sekcja wywołuje kwerendę z innym parametrem regionu i przedstawia wynik.



**7.7. Podstrona „/Turnieje”** Cel: Zbiorcze informacje o turniejach i statystyki związane z turniejami. Mechanizm: Metoda get\_all\_tournaments() pobiera wszystkie turnieje. Metoda get\_tournaments\_count\_by\_region() wyświetla informację, ile turniejów przypada na dany region. Co widać: Tabela “Wszystkie turnieje” – nazwa, rodzaj, data, region itp. Tabela “Liczba turniejów w regionach” – region + count.

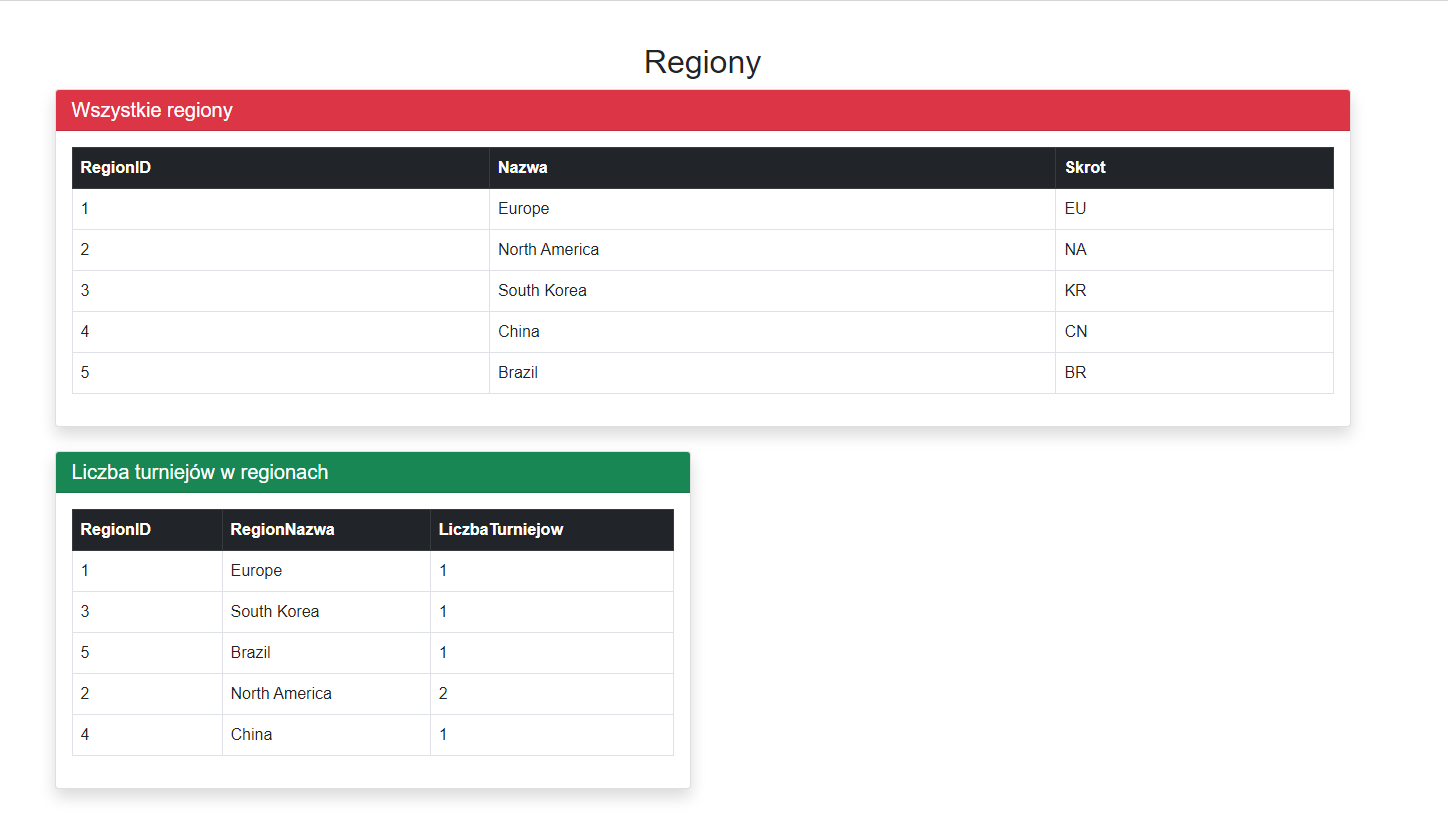


**7.8. Podstrona „/Druzyny”** Cel: Przegląd drużyn oraz ich statystyk (np. drużyny z największą liczbą graczy, drużyny z największą liczbą transferów). Mechanizm: get\_teams\_with\_most\_players(), get\_teams\_with\_most\_transfers(), get\_all\_teams(), get\_teams\_by\_region(...). Co widać: Kafelek (lub tabela) “Drużyny z największą liczbą graczy”. Kafelek “Drużyny z największą liczbą transferów”. Pełna lista drużyn i ewentualnie drużyny z podziałem na regiony.

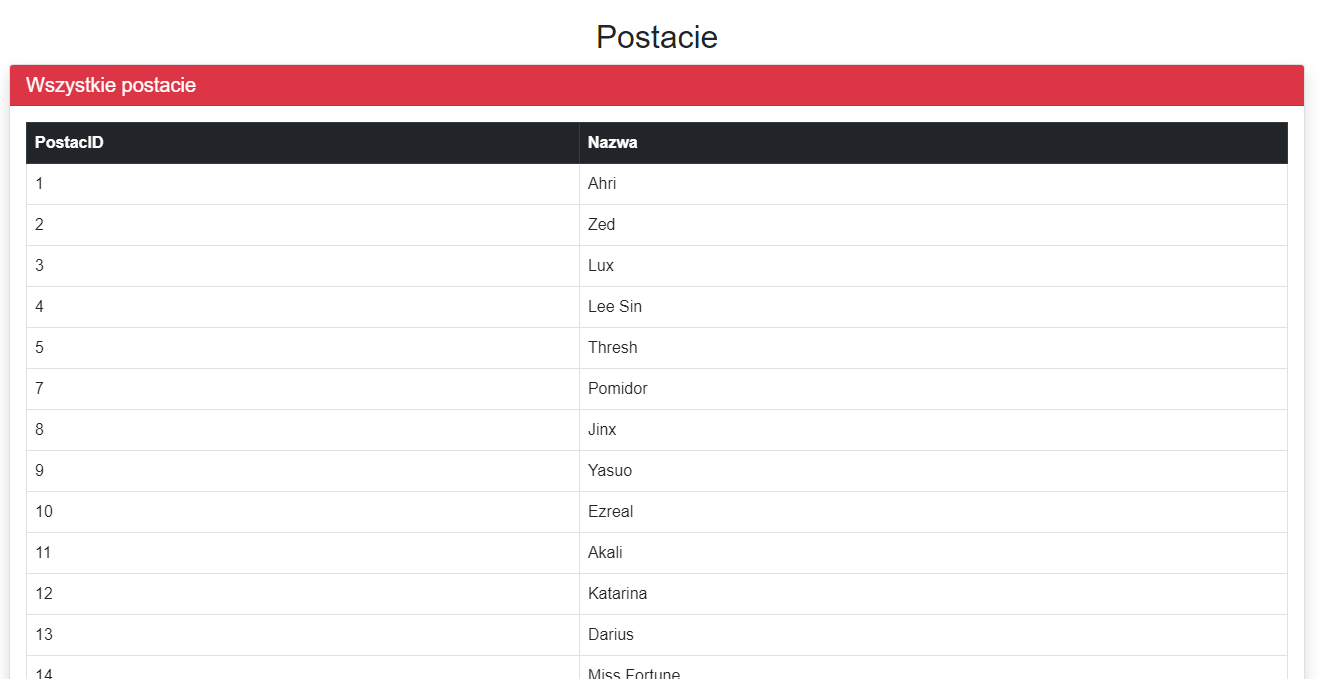


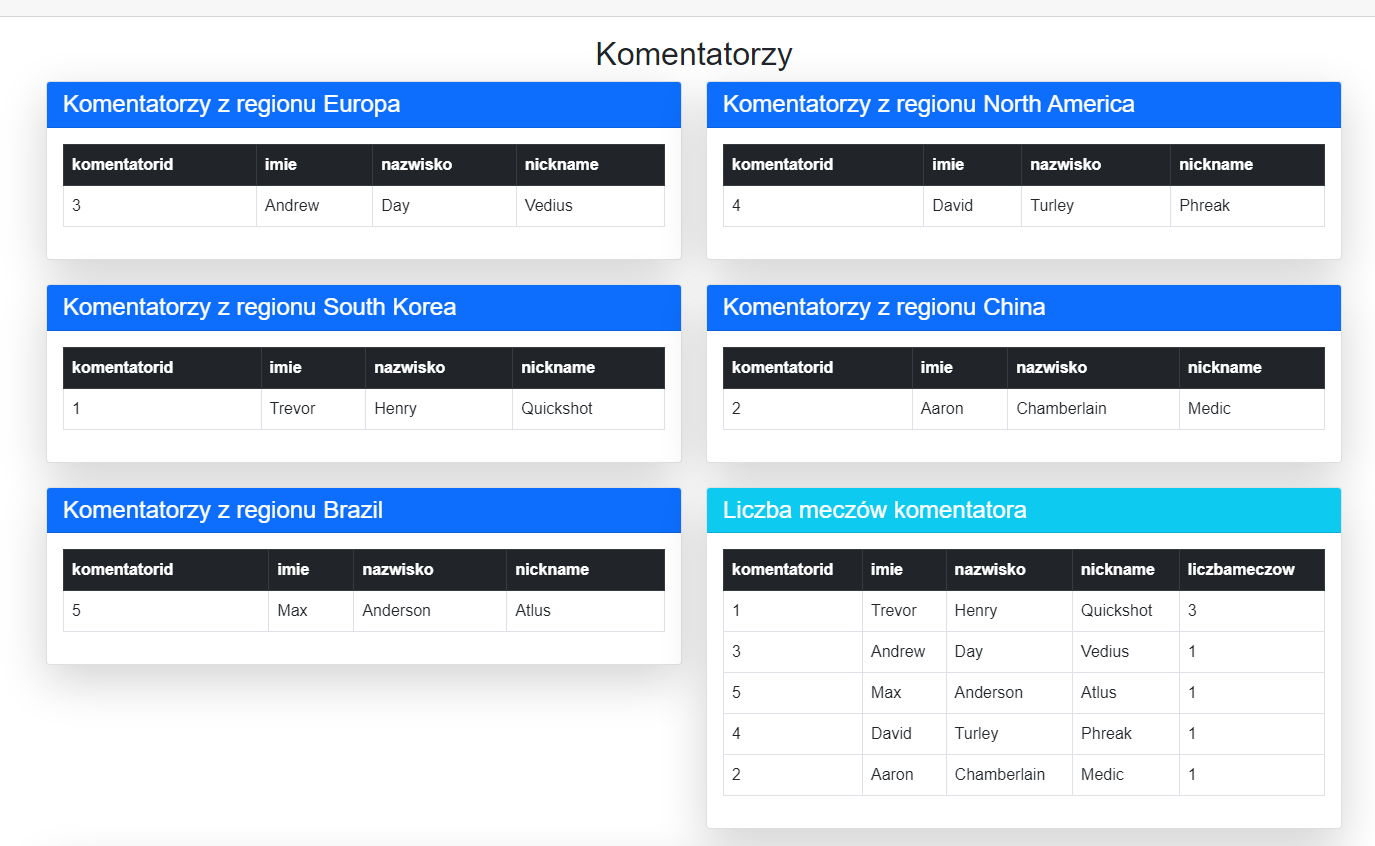
**7.9. Podstrona „/Regiony”** Cel: Prezentacja wszystkich regionów, a także statystyki powiązane z regionami (np. liczba turniejów). Mechanizm: Wywołuje get\_all\_regions() (pełna lista) oraz get\_tournaments\_count\_by\_region(). Co widać: Lista regionów (ID, nazwa, skrót). Tabela “Liczba turniejów w regionach”

.



**7.10. Podstrona „/Postacie”** Cel: Pokazuje wszystkie postacie (championy) i ich popularność. Mechanizm: get\_all\_characters() – lista wszystkich championów. get\_most\_picked\_characters() – najczęściej wybierane postacie przez graczy. Co widać: Tabela “Wszystkie postacie”. Tabela “Najczęściej wybierane postacie” z liczbą wyborów.

 **7.10. Podstrona „/Komentatorzy”** Cel: Pokazuje wszystkich komentatorów Mechanizm: get\_all\_commentators() – lista wszystkich komentatorów. get\_commentators\_by\_region() – wyświetla komentatorów z danego regionu



**Uwagi:**

Każda podstrona korzysta z mechanizmu Blazor/ASP.NET do renderowania dynamicznego (po pobraniu wyników z serwisu). W większości przypadków dane są prezentowane w formie tabel (Bootstrap) z nagłówkami kolumn odpowiadającymi właściwościom zwracanym przez funkcje bazy danych. Komunikaty błędów (np. brak połączenia) lub powodzenia (np. “Drużyna dodana pomyślnie!”) są pokazywane w alertach.

1. **Przeniesienie bazy danych z relacyjnej na nierelacyjną**  
   **8.1. Analiza i kroki do transportowania bazy relacyjnej na model nierelacyjny (słownie)**

Przechowywane dane w tym projekcie są **mocno relacyjne** (wiele tabel, klucze obce, silne powiązania). Natomiast w modelu nierelacyjnym (np. **MongoDB**) musielibyśmy rozważyć:

**Struktura kolekcji**:

Najpierw ustalamy główne “byty” w formie kolekcji (np. drużyny, gracze, mecze, turnieje, regiony itd.).

Część powiązań można “zagnieździć” w dokumentach. Na przykład w kolekcji druzyny przechowywać tablicę obiektów gracze (jeśli zależność jest “1 drużyna - wiele graczy” i rzadko edytujemy dane o graczach).

Inne obiekty (np. transfery) mogą być przechowywane osobno i referencyjnie (np. \_id gracza).

**Brak joinów**:

W MongoDB zapytania *JOIN* nie istnieją w klasycznej formie. Kwerendy łączenia muszą być obsługiwane po stronie aplikacji (lub częściowo za pomocą *lookup pipelines* w Aggregation Framework).

W związku z tym kluczowa jest decyzja, które obiekty zagnieździć, a gdzie używać “referencji” (\_id innego dokumentu).

**Mapowanie struktur**:

Tabele typowo “słownikowe” (np. postacie/regiony) można przenieść do osobnych kolekcji: characters, regions.

Tabele typu xyz (relacja *wiele do wielu*) można w MongoDB rozwiązać wbudowanymi tablicami ID, np. w dokumencie meczu (match) zagnieździć pola: players: [ { playerId: X, champion: Y, stats: { ... } }, ... ].

**Obsługa transakcji**:

MongoDB wspiera transakcje wielodokumentowe w ograniczonym zakresie (w “replica set” i “sharded cluster”). Jeżeli intensywnie korzystamy z transakcji (jak w RDBMS), trzeba odpowiednio zaplanować architekturę.

**Migracja procedur i funkcji**:

Logikę z procedur (np. dodajTransfer) należałoby przenieść do warstwy aplikacji w formie metod. W MongoDB nie przechowujemy *procedur składowanych*; mamy co najwyżej pipeline’y agregacji, skrypty JavaScript w starym mechanizmie map-reduce.

Podsumowując, **konwersja** do bazy nierelacyjnej wymaga zdefiniowania **odpowiednich kolekcji** i ustalenia, które dane będą zagnieżdżone, a które będą “relacjami” poprzez referencje \_id. Zamiast szerokich joinów i kluczy obcych w wielu tabelach, można by mieć:

**8.2. Migracja za pomocą MongoDB (przykład)**

Załóżmy, że mamy zrzut danych CSV bądź JSON z każdej tabeli. Kroki mogłyby wyglądać tak:

1. **Eksport danych z Postgres**:
   1. Tworzymy zapytania SELECT row\_to\_json(...) i zapisywać wynik do pliku.
2. **Transformacja** danych CSV/JSON do struktury JSON zgodnej z docelowym modelem MongoDB:
   1. Łączymy dane z gracze i druzyny w jeden dokument, np. teams.
   2. Uzupełniamy tablicę players.
3. **Wczytanie do Mongo**:
   1. Używamy mongoimport
4. **Weryfikacja**

Po migracji, operacje CRUD i zapytania analityczne byłyby budowane w **MongoDB** (np. z użyciem pipeline’ów agregacji). Złożone statystyki (jak get\_top\_kda\_players) robi się z reguły w warstwie aplikacji lub w *aggregation pipeline*.

Zapytanie:



1. **Podsumowanie**  
   Projekt “E-sport League of Legends” demonstruje:

Jak zaprojektować bazę danych relacyjną z wieloma tabelami i powiązaniami (FK).

Jak korzystać z procedur i funkcji w PostgreSQL, wraz z wyzwalaczami (triggerami), by automatyzować procesy (np. transfer gracza).

Jak w warstwie .NET/Blazor/Entity Framework można zdalnie zarządzać danymi (CRUD, odczyt statystyk).

Jak można przekształcić ściśle relacyjną strukturę w formę nierelacyjną (MongoDB) poprzez zagnieżdżanie obiektów i obsługę logiczną w aplikacji (rozważanie, co wstawić do jednej kolekcji, co przechowywać osobno).

**Najważniejsze elementy**:

**Schemat Bazy** – tabele: drużyny, gracze, turnieje, mecze, postacie, trenerzy, komentatorzy, transfery, regiony, xyz.

**Procedury** – w tym procedury typu dodaj..., edytuj..., usun... oraz funkcje do analizy statystyk.

**Trigger** – m.in. tr\_transfer\_gracza (aktualizacja drużyny przy transferze).

**Aplikacja .NET** – z rozdzieleniem na warstwę DBContext (EF) i warstwę serwisów (np. DruzynaService), a finalnie warstwę prezentacji w Blazorze.

**Możliwość migracji do bazy NoSQL** – wymaga ponownego modelowania oraz rezygnacji z części kluczy obcych na rzecz zagnieżdżania dokumentów lub referencji \_id.

Całość pozwala na kompleksowe zarządzanie danymi e-sportowymi, od tworzenia i edytowania drużyn i graczy, przez rejestrację meczów i transferów, aż po rozbudowane statystyki i przegląd wyników. Dokumentacja i kod demonstrują spójną integrację technologii: PostgreSQL (z PL/pgSQL i funkcjami), Entity Framework Core, Blazor (Razor) i architekturę warstwową aplikacji .NET.

1. **Literatura**

* **Microsoft Learn – ASP.NET Core Blazor**

Oficjalna dokumentacja Blazor, zawierająca przewodniki, przykłady i szczegółowe wyjaśnienia dotyczące tworzenia aplikacji w tej technologii.

<https://learn.microsoft.com/en-us/aspnet/core/blazor>

* **Entity Framework Core Documentation**

Oficjalna dokumentacja Entity Framework Core, narzędzia używanego do zarządzania danymi i komunikacji z bazą danych.

<https://learn.microsoft.com/en-us/ef/core>

* **Stack Overflow – Community Questions and Answers**

Popularne forum, gdzie można znaleźć odpowiedzi na konkretne pytania związane z problemami technicznymi i implementacją.

[https://stackoverflow.com](https://stackoverflow.com/)

* **W3Schools – SQL and Database Tutorials**

Zasoby edukacyjne dotyczące języka SQL, zarządzania bazami danych oraz ich integracji z aplikacjami webowymi.

<https://www.w3schools.com/sql>

* **Blazor University**

Nieoficjalny, ale bardzo rozbudowany przewodnik po Blazorze, z przykładami implementacji komponentów, routingu oraz obsługi danych.

[https://blazor-university.com](https://blazor-university.com/)

* **Bootstrap Documentation**

Oficjalna dokumentacja frameworka CSS Bootstrap, który może być używany do stylizacji i układu elementów w aplikacji.

[https://getbootstrap.com](https://getbootstrap.com/)