МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна технологія аналізу результатів спортивних подій»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Кузіков Б.О.**

Студент гр. ІН–61 — **Монько Є.М.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую _____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 р.

Завдання

до випускної роботи

Студента четвертого курсу, групи ІН-61 спеціальності «Інформатика» денної форми навчання Монька Єгора Миколайовича.

Тема: «Інформаційна технологія аналізу результатів спортивних подій»

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: 1) аналітичний огляд проблеми дослідження; 2) постановка задачі й формування завдань дослідження; 3) проектування бази даних інформаційної системи; 4) розробка архітектури інформаційної системи; 5) реалізація інформаційної системи, 6) аналіз результатів.

Дата видачі завдання “_____”_____2020 р.

Керівник випускної роботи _____ Кузіков Б.О.

Завдання прийняв до виконання _____ Монько Є.М.

РЕФЕРАТ

Записка: 59 стор., 34 рис., 8 додатків, 16 джерел.

Об'єкт дослідження – Інформаційна технологія аналізу результатів спортивних подій.

Мета роботи – розробка інформаційної системи, котра допоможе аналізувати помилки букмекерських компаній на передбачення результатів футбольних матчах в найпопулярніших лігах.

Результати – виконано вибір методів вирішення поставленої задачі; реалізовано інформаційну систему, котра регулярно збирає дані, базу даних для їх зберігання, а також веб-додаток для візуалізації інформації.

ІНФОРМАЦІЙНА СИСТЕМА, ФУТБОЛ, ФРЕЙМВОРК, HTML, CSS, JAVASCRIPT, JAVA, API, JSON, SPARK, POSTGRE

ЗМІСТ

ВСТУП

Про популярність ставок на спорт в сучасному світі можуть свідчити дані від FIFA. Наприклад, за їх підрахунками загальний обсяг ставко на матчі Чемпіонату світу з футболу у 2018 році складав сто тридцять шість мільярдів євро. У середньому на один матч люди ставили приблизно два мільярди євро. На фінал – сім мільярдів. Тож правильне передбачення результатів для компаній є важливим для отримання прибутку.

Беручи до уваги постійну популярність цієї сфери та існування на ринку багатьох букмекерських контор, можна зробити висновок про відсутність на цей час алгоритму, який зміг би зі стовідсотковою точністю робити прогнози відносно спортивних подій, які відбудуться. Такий алгоритм навряд чи буде створено, адже на результат спортивного матчу впливає надзвичайно велика кількість факторів, багато із яких не піддаються прогнозуванню. Незважаючи на це, чимало провідних університетів у світі розробляють алгоритм, який мав би високу точність передбачення. Отже, результат спортивної події є наслідком збігу багатьох факторів, кожен з яких має різні ступені впливу. Відсутність очевидних зв'язків між факторами впливу робить неможливим стовідсоткове прогнозування результату експертами, які базуються на власних логічних висновках.

Метою цієї роботи є розробка інформаційної системи, котра допоможе аналізувати помилки букмекерських компаній. Для аналізу було обрано один з найпопулярніших видів спорту – футбол.

Для реалізації поставленої мети необхідно виконати наступні **завдання**:

- проаналізувати існуючі рішення проблеми та сформулювати задачу дослідження;
- спроектувати інформаційну систему аналізу передбачень спортивних подій;
- реалізувати інформаційну систему;

1. АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Аналіз проблеми

Для аналізу можливих переможців було створено багато систем. Наприклад, вгадати переможців за допомогою штучного інтелекту на чемпіонат Євро-2016 намагалися такі компанії, як Google, Yahoo, Microsoft та інші. Більшість з них вважала фаворитом збірну Німеччини. Ніхто з них не бачив фаворитом реального переможця – збірну Португалії.

Розробленням методів та моделей займалися такі науковці: С. Добсон, А. Ротштейн, Дж. Годдард, С. Штовба. Стівен Добсон і Джон Годдард запропонували модель, де основним фактором вибрано кількість голів, забитих кожною командою при персональних зустрічах. Модель побудована близько на 30 сезонах даних. Інший науковець, Ротштейн А. використовув алгоритм ставок методом нечіткої логіки на основі 12 чемпіонатів з футболу Фінляндії. Алгоритм розрахунку ставок на футбол проводиться через нейрони і генетичне навчання. Штовба С. прогнозував розподіл місць у турнірній таблиці чемпіонату України з футболу на основі нечіткої логіки. Буурсма обрав набір функцій і використав ряд класифікацій алгоритмів, включаючи просту і логістичну регресії, байєсівську мережу, і дерево прийняття рішень, щоб передбачити результат футбольного матчу. Його передбачення мали три виходи (перемога команди господаря, нічия, перемога гостьової команди). Ймовірності цих трьох результатів були розраховані і результат із найбільшою ймовірністю був обраний [1].

Студенти Корнелльського університету створили систему на основі машинного навчання, котра декількома способами спрогнозувала переможця останнього Чемпіонату світу [2]. В їх результатах Франція мала шанси пройти до фіналу. Хорватія мала тільки шість відсотків потрапити до фіналу.

Таким чином, ми бачимо, що на даний момент не існує системи, котра б могла безпомилково визначити результат матчу. А це означає, що як науковці, так і самі

букмекерські компанії роблять неточні передбачення. Це дає можливість на основі власного аналізу зробити передбачення на ту чи іншу спортивну подію.

Існують сервіси, котрі зберігають данні минулих матчів та данні букмекерів для них для подальшого аналізу з метою передбачення наступних результатів. Прикладом такого сервісу є **oddsportal.com** (див. рис. 1.1).

21/05, 14:00	**Liverpool** - Middlesbrough	**3:0**	1.13	9.49	21.41	9
14/05, 13:15	West Ham - **Liverpool**	**0:4**	5.83	4.08	1.63	9
07/05, 12:30	**Liverpool** - Southampton	**0:0**	1.58	4.21	6.16	9
01/05, 19:00	Watford - **Liverpool**	**0:1**	5.77	4.28	1.61	9
23/04, 15:30	**Liverpool** - Crystal Palace	**1:2**	1.51	4.43	7.15	9
16/04, 12:30	West Brom - **Liverpool**	**0:1**	4.57	3.69	1.85	9
08/04, 14:00	Stoke - **Liverpool**	**1:2**	3.43	3.32	2.27	9
05/04, 19:00	**Liverpool** - Bournemouth	**2:2**	1.43	4.76	8.14	9
01/04, 11:30	**Liverpool** - Everton	**3:1**	1.73	3.77	5.44	8
19/03, 15:30	Manchester City - **Liverpool**	**1:1**	1.95	3.80	3.94	8
12/03, 15:00	**Liverpool** - Burnley	**2:1**	1.27	6.10	12.55	8
04/03, 16:30	**Liverpool** - Arsenal	**3:1**	1.99	3.65	3.99	8
27/02, 19:00	Leicester - **Liverpool**	**3:1**	5.59	4.07	1.65	8
11/02, 16:30	**Liverpool** - Tottenham	**2:0**	2.23	3.41	3.46	8

Рисунок 1.1 Приклад даних з сайту oddsportal.com

Як бачимо, представлена історія матчів, їх результати та ставки. Але таке відображення даних не є зручним для аналізу результатів певної команди чи ліги в цілому. Воно не є наочним і складне для сприйняття. Також немає можливості для підрахунку кількості помилок передбачень для певної команди протягом сезону, місяця тощо. По факту, це є просте відображення даних у вигляді звичайного списку, відсортоване за командами.

Існують сайти, котрі пропонують придбати історичні дані про матчі та ставки на них. Інформація буде надана в файлах (наприклад .csv). Також існують сайти з безкоштовними даними, наприклад **football-data.co.uk/englandm.php**. Цей ресурс надає дані у вигляді .csv фалу (див. рис. 1.2).

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Date	Time	HomeTeai	AwayTean	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF	HC	AC
2	9/8/2019	20:00	Liverpool	Norwich	4	1	H	4	0	H	M Oliver	15	12	7	5	9	9	11	2
3	########	12:30	West Ham	Man City	0	5	A	0	1	A	M Dean	5	14	3	9	6	13	1	1
4	########	15:00	Bournemc	Sheffield U	1	1	D	0	0	D	K Friend	13	8	3	3	10	19	3	4
5	########	15:00	Burnley	Southamp	3	0	H	0	0	D	G Scott	10	11	4	3	6	12	2	7
6	########	15:00	Crystal Pal	Everton	0	0	D	0	0	D	J Moss	6	10	2	3	16	14	6	2
7	########	15:00	Watford	Brighton	0	3	A	0	1	A	C Pawson	11	5	3	3	15	11	5	2
8	########	17:30	Tottenhan	Aston Villa	3	1	H	0	1	A	C Kavanag	31	7	7	4	13	9	14	0
9	########	14:00	Leicester	Wolves	0	0	D	0	0	D	A Marrine	15	8	1	2	3	13	12	3
10	########	14:00	Newcastle	Arsenal	0	1	A	0	0	D	M Atkinso	9	8	2	2	12	7	5	3
11	########	16:30	Man Unite	Chelsea	4	0	H	1	0	H	A Taylor	11	18	5	7	15	13	3	5
12	17/08/201	12:30	Arsenal	Burnley	2	1	H	1	1	D	M Dean	16	18	9	5	13	11	10	7
13	17/08/201	15:00	Aston Villa	Bournemc	1	2	A	0	2	A	M Atkinso	22	12	7	4	10	13	10	5
14	17/08/201	15:00	Brighton	West Ham	1	1	D	0	0	D	A Taylor	16	8	4	3	11	10	8	6
15	17/08/201	15:00	Everton	Watford	1	0	H	1	0	H	L Mason	12	8	2	2	11	11	4	7
16	17/08/201	15:00	Norwich	Newcastle	3	1	H	1	0	H	S Attwell	15	10	8	3	9	11	7	5
17	17/08/201	15:00	Southamp	Liverpool	1	2	A	0	1	A	A Marrine	14	15	3	6	10	6	5	9
18	17/08/201	17:30	Man City	Tottenhan	2	2	D	2	1	H	M Oliver	30	3	10	2	14	4	13	2
19	18/08/201	14:00	Sheffield U	Crystal Pal	1	0	H	0	0	D	D Coote	15	6	3	4	16	11	8	4
20	18/08/201	16:30	Chelsea	Leicester	1	1	D	1	0	H	O Langfor	14	12	5	3	9	14	4	5
21	19/08/201	20:00	Wolves	Man Unite	1	1	D	0	1	A	J Moss	6	9	2	2	17	8	4	6
22	23/08/201	20:00	Aston Villa	Everton	2	0	H	1	0	H	M Oliver	7	12	3	1	10	18	0	6
23	24/08/201	12:30	Norwich	Chelsea	2	3	A	2	2	D	M Atkinso	6	23	5	8	9	9	1	8
24	24/08/201	15:00	Brighton	Southamp	0	2	A	0	0	D	K Friend	12	12	3	4	9	10	8	5
25	24/08/201	15:00	Man Unite	Crystal Pal	1	2	A	0	1	A	P Tierney	22	5	3	3	8	18	8	1
26	24/08/201	15:00	Sheffield U	Leicester	1	2	A	0	1	A	A Madley	8	10	3	2	11	6	7	4
27	24/08/201	15:00	Watford	West Ham	1	3	A	1	1	D	C Kavanag	23	16	3	10	12	11	8	7

Рисунок 1.2 Приклад даних з football-data.co.uk

Такі дані можуть бути зручними для подальшої роботи з ними, адже вони добре структуровані. Але проблема цього ресурсу в тому, що він надає інформацію тільки про команди Англії. А також ці данні не будуть оновлюватися. Необхідно кожного разу знову завантажувати файл для подальшої роботи.

Soccerstats пропонує багато інформації про всі аспекти гри. Цей сайт охоплює численні різні ліги з усього світу (див. рис. 1.3). Він дає можливість безпосередньо аналізувати результат спортивних подій. Але цей ресурс не має можливості аналізу передбачень букмекерів.

Рисунок 1.3 Приклад даних з soccerstats.com

Сервіс **whoscored** також надає розширену статистику футбольних матчів, статистику команд та ліг (див. рис. 1.4). Але як і попередній ресурс, не містить даних про букмекерські компанії.

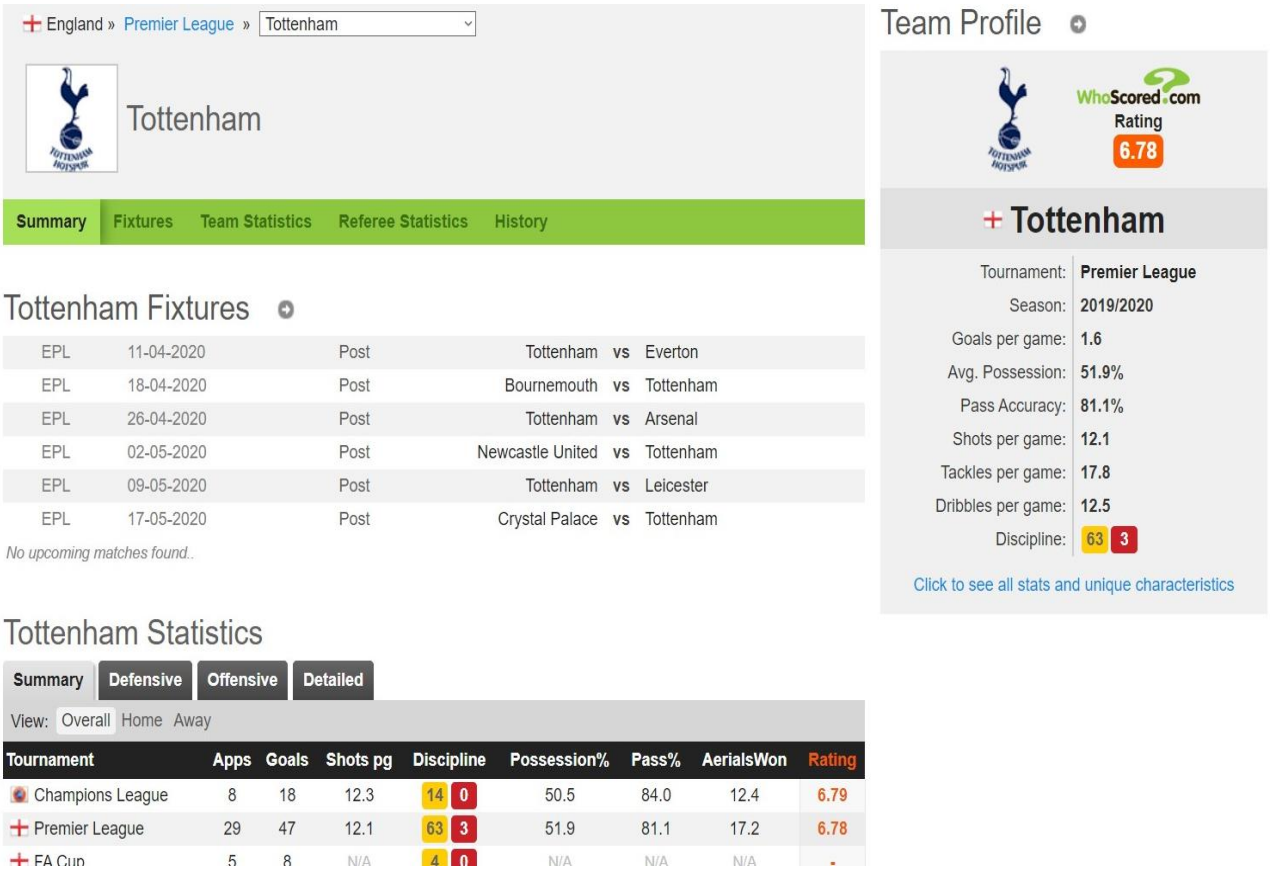

Рисунок 1.4 Приклад даних з ресурсу whoscored.com

Ресурс **flashscore** призначений для аналізу статистики окремих команд. Надає статистику про матчі, трансфери та поточний склад команд. Існує можливість перегляду передбачень букмекерів, але це реалізовано тільки як перегляд даних на один матч (див. рис. 1.5). Загальна статистика для команди чи ліги відсутня.

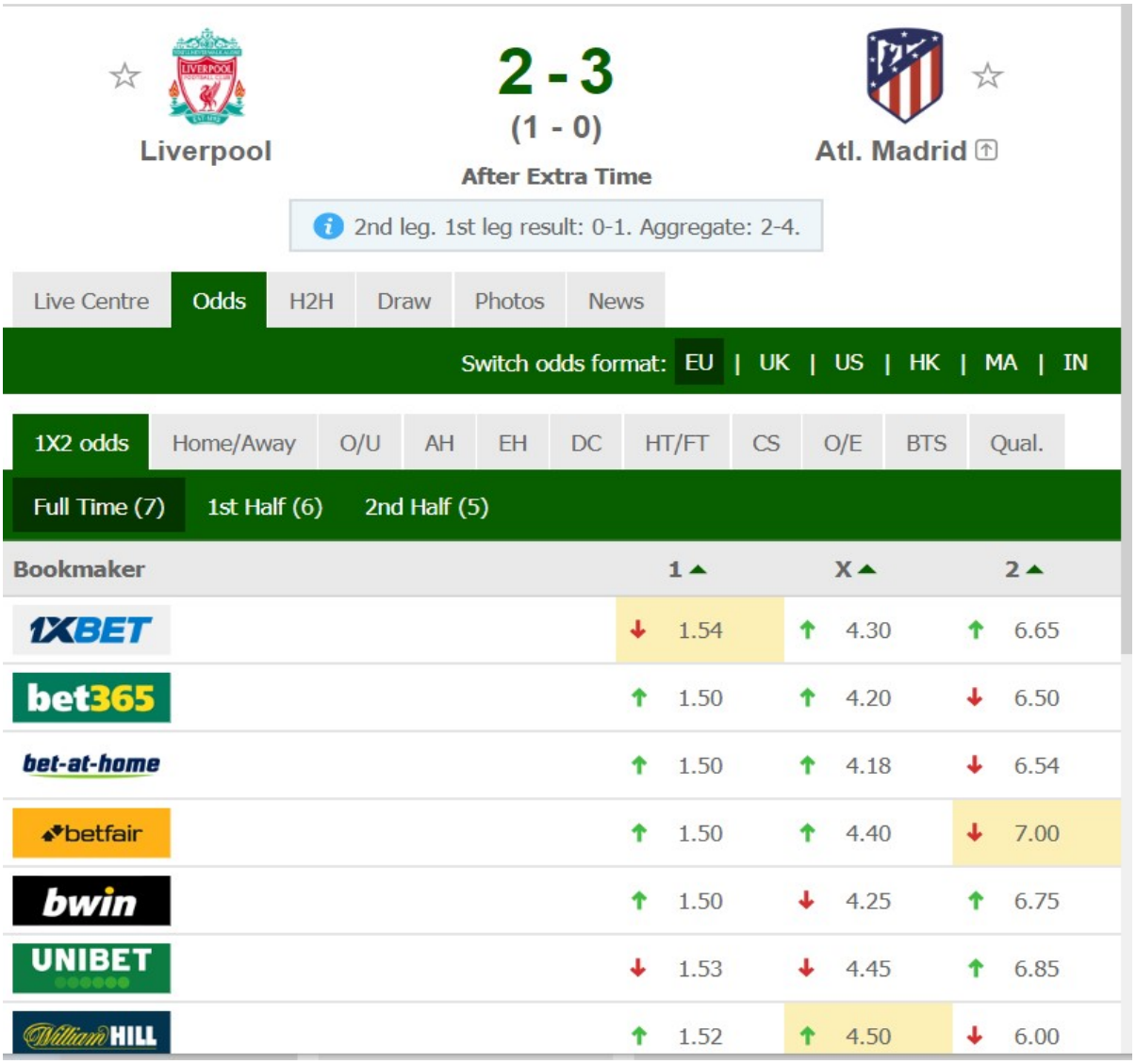

Рисунок 1.5 Приклад даних з ресурсу flashscore.com

Сайт **ice2bet** надає різну статистику для аналізу матчів та ліг. Серед переваг цього ресурсу можна відзначити можливість перегляду даних для ліг чи окремої команди та варіативність у сортуванні даних (див. рис. 1.6). Недоліками є незрозумілий інтерфейс без пояснень значення деяких типів сортування даних та відсутність в статистиці даних букмекерських компаній.

Рисунок 1.6 Приклад даних з сайту ice2bet.com

Також існують багато API для отримання передбачень на матчі та їх результати в реальному часі. Така система у сукупності з інструментом для візуалізації даних цілком може задовольнити вимоги нашої інформаційної системи. Але майже всі API мають обмежену кількість запитів за певний проміжок часу, що унеможливлює роботу з великою кількістю користувачів. Тому найдоцільніше буде розробити додаток, котрий регулярно буде збирати актуальні дані за певний проміжок часу і зберігати їх у базу даних інформаційної системи.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP.

Прикладни́й програ́мний інтерфе́йс (інтерфейс програмування застосунків, інтерфейс прикладного програмування) (англ. Application Programming Interface, API) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Сайт **sportmonks** надає зручний API для отримання даних про футбольні матчі та передбачення на них (див. рис. 1.7). Дані надаються у форматі JSON у відповідь на HTTP-запит.

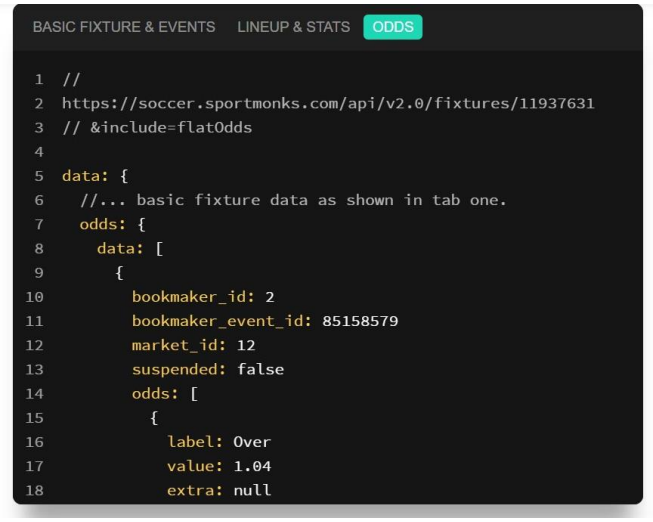

Рисунок 1.7 Приклад даних з ресурсу sportmonks.com

Значним недоліком цього ресурсу є обмеженість безкоштовного тарифного плану. Дані можна отримувати тільки з двох ліг – Данії та Шотландії.

Ресурс **allsportsapi** надає дані про хокей, футбол, крикет та ін. Для футбольних матчів існує можливість отримання даних про результати ігор та ставки на них у форматі JSON (див. рис. 1.8).

Рисунок 1.8 Сторінка документації allsportsapi.com з прикладом даних про ставки

Недоліком даного ресурсу, як і попереднього, є обмеженість безкоштовного плану в кількості ліг. Цей ресурс щороку надає доступ до даних двох випадкових.

Сервіс **apifootball** (див. рис. 1.9) надає статистику матчів та дані ставок. У безкоштовному плані доступ надається тільки до двох ліг. Доступ до передбачень букмекерів надається за додаткову плату.

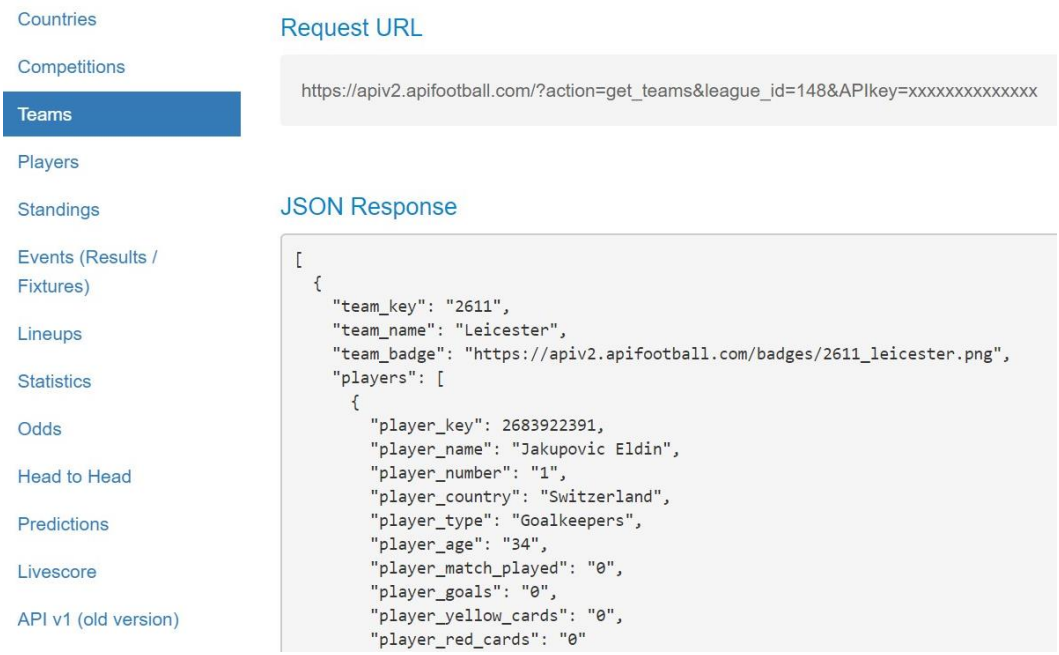

Рисунок 1.9 Приклад документації apifootball.com

Сервіс **football-data.org** надає безкоштовні десять запитів на хвилину і дані з дванадцяти чемпіонатів. Приклад типового запиту:

```
$.ajax({
  headers: { 'X-Auth-Token': 'YOUR_API_TOKEN' },
  url: 'http://api.football-data.org/v2/matches?status='LIVE',
  dataType: 'json',
  type: 'GET',
}).done(function(response) {
  // do something with the response, e.g. isolate the id of a linked resource
  console.log(response);
});
```

[3]

У відповідь ми отримаємо JSON, котрий буде обробляти додаток. Таким чином, можна отримати дані про результати матчів.

Сервіс **the-odds-api.com** надає дані про ставки різних букмекерських компаній на певні матчі (див. рис. 1.10). Отримання даних реалізується за допомогою REST API:

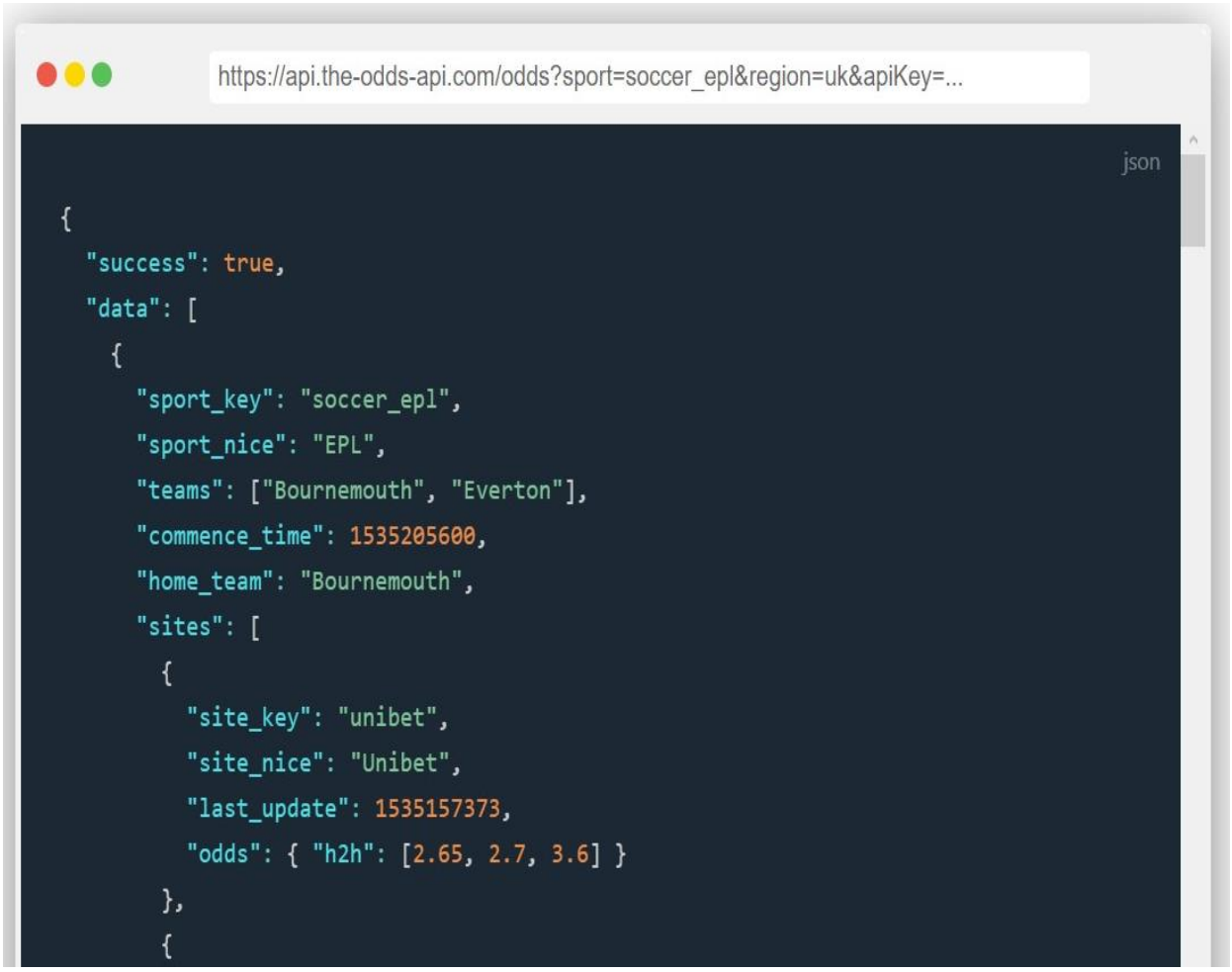

Рисунок 1.10 Приклад запиту і відповіді з ресурсу the-odds-api.com

Як бачимо, принцип взаємодії простий і зрозумілий. Безкоштовний тариф ресурсу – п’ятсот запитів на місяць. Тож для отримання даних один раз на день на такого обмеження достатньо [4].

Таблиця 1.1

Порівняння сервісів надання даних

Ресурс	Наявність зручного інтерфейсу користувача	Можливість отримувати актуальні дані регулярно	Передбачення букмекерів	Декілька ліг	Сортування вісх даних за різними критеріями	Користування в безкоштовному плані
oddsportal.com	+	+	+	+	-	+
football-data.co.uk	-	-	+	-	+	+
soccerstats.com	+	+	-	+	+	+
whoscored.com	+	+	-	+	+	+
flashscore.com	+	+	+	+	-	+
ice2bet.com	+	+	-	+	+	+
sportmonks.com	-	+	+	+	+	-
allsportsapi.com	-	+	+	+	+	-
apifootball.com	-	+	+	+	+	-
football-data.org	-	+	-	+	+	+
the-odds-api.com	-	+	+	+	+	+

Оскільки варіантів отримання даних існує багато, то проблема лишається за їх візуалізацією. Найзручніший спосіб взаємодії з додатком у сучасних умовах різноманіття платформ і операційних систем – веб-додаток. Такий варіант є ідеальним для вирішення проблеми дослідження. Кінцевому користувачу не потрібно встановлювати додаток для користування – достатньо тільки перейти за посиланням.

Існує багато Javascript бібліотек для побудови діаграм і графіків на клієнтській стороні. Наприклад **chartjs.org** – open-source бібліотека з необхідним функціоналом (див. рис. 1.11). Вона є повністю безкоштовною і має змістовну документацію.

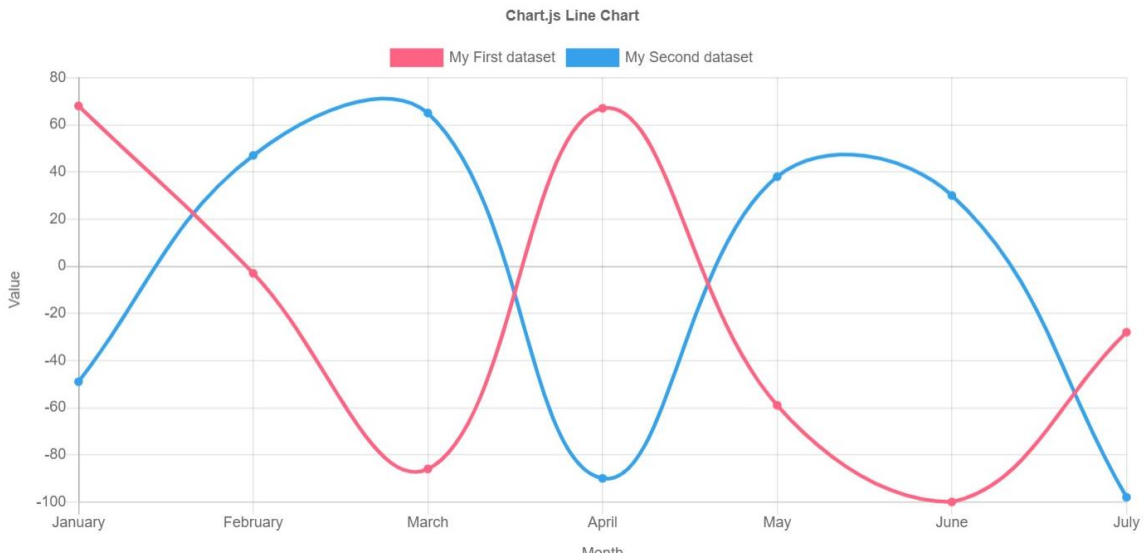

Рисунок 1.11 Приклад графіку, побудованого за допомогою chartjs

Конфігурація також є простою і зрозумілою:

```
var ctx = document.getElementById('myChart').getContext('2d');
var myChart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
        datasets: [{
            label: '# of Votes',
            data: [12, 19, 3, 5, 2, 3],
```

```
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [
                'rgba(255, 99, 132, 1)',
                'rgba(54, 162, 235, 1)',
                'rgba(255, 206, 86, 1)',
                'rgba(75, 192, 192, 1)',
                'rgba(153, 102, 255, 1)',
                'rgba(255, 159, 64, 1)'
            ],
            borderWidth: 1
        }]
    },
    options: {
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
```

Бібліотека **amcharts** надає можливість візуалізувати дані за допомогою великої кількості різноманітних діаграм та графіків (див. рис. 1.12). Ця бібліотека є однією з найпопулярніших. Доступ до неї безкоштовний, але відображення брендингу можна прибрати тільки в платному використанні.

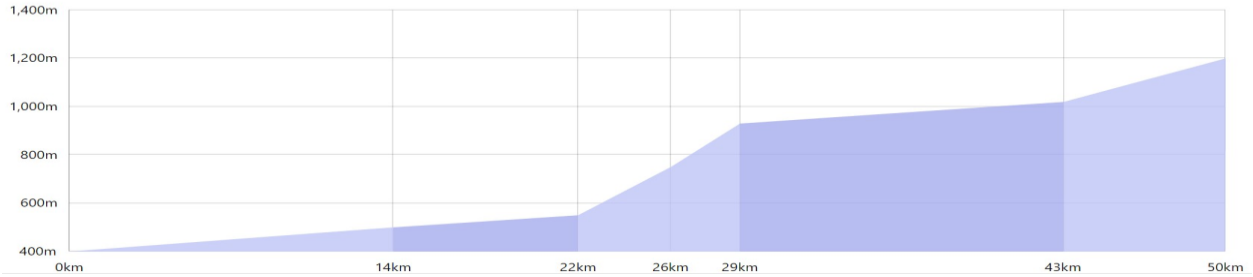

Рисунок 1.12 Приклад діаграми, побудованої за допомогою amcharts

Бібліотека **anychart** (див. рис. 1.13) має схожий набір функцій і схожу модель використання – безкоштовно з водяним знаком; платна версія – без брендування.

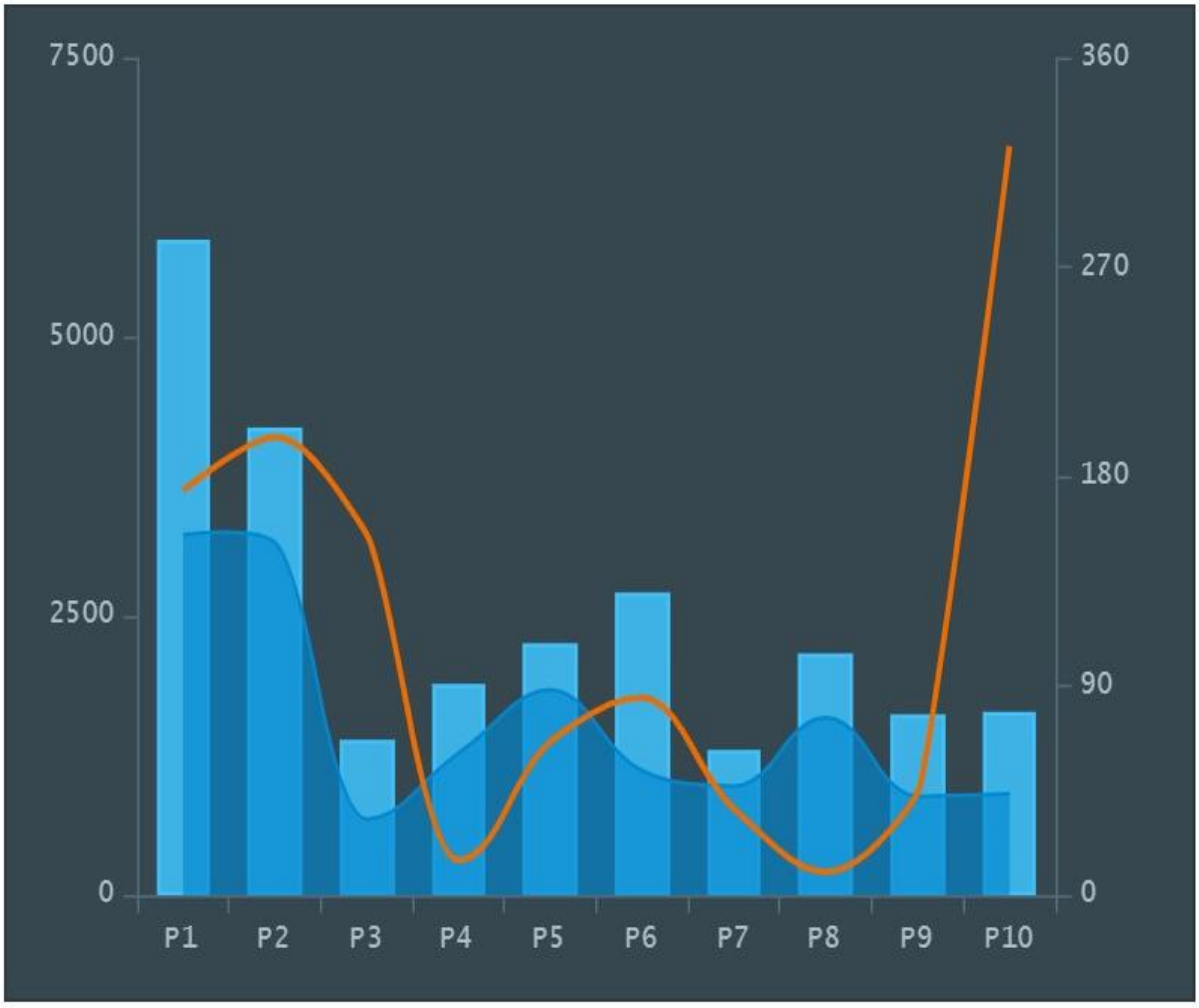

Рисунок 1.13 Приклад діаграми anychart

Таблиця 1.2

Порівняння методів віуалізації

Бібліотека	Відкритий код	Зручність документації	Безкоштовне використання
chartjs	+	+	+
amcharts	-	+	+
anychart	-	+	+

Отже, проблема дослідження реалізується в двох аспектах – збір необхідних актуальних даних та зручна і зрозуміла для кінцевого користувача візуалізація цих даних.

1.2 Постановка задачі дослідження

Основною задачею дослідження є розробка інформаційної системи, котра допоможе користувачеві аналізувати дані футбольних матчів окремих команд або ліги протягом сезону.

Для реалізації цієї задачі її умовно можна розділити на дві. Перша – збирання та зберігання даних. Друга – надання користувачеві зручного інтерфейсу для візуалізації цих даних.

Система повинна містити дані про футбольні матчі декількох ліг та ставки на ці матчі. Спираючись на описані у попередньому розділі API, для дослідження необхідно обрати декілька ліг, наявних в обох сервісах.

Збір даних має відбуватися щодня. Таким чином, кожного разу будуть збиратися передбачення на наступний ігровий день і результати матчів попереднього. З усіх отриманих даних необхідно вибрати тільки потрібні для системи. Збирання даних має бути стабільне і регулярне. А, отже, воно має працювати на окремому сервері, який гарантує щоденну роботу у визначений час.

Данні мають зберігатися у зручній для подальшого використання СУБД. Для складної структури необхідна реляційна база даних, що дозволить реалізувати зв’язки між окремими сутностями.

База даних теж має бути розміщена на сервері, що дасть можливість в будь-який момент працювати з нею. Для цього на сервері необхідно налаштувати середовище для віддаленого доступу. У подальшому це полегшить розробку інформаційної системи.

Увесь користувацький інтерфейс повинен надаватися через браузер, що дасть можливість не орієнтуватися при розробці на конкретну операційну систему кінцевого користувача. Веб-додаток не потребує інсталяції. Він повинен працювати на сервері для можливості отримати доступ до даних у будь-який момент.

Інтерфейс повинен бути мінімалістичний і зрозумілий. Для візуалізації необхідно використовувати бібліотеку, котра одночасно надає необхідний функціонал і використовує якомога менше ресурсів пристрою користувача.

Клієнтська частина має бути легкою для швидкого завантаження сторінки. Але при цьому сторінки мають бути функціональні та з мінімальною необхідністю перезавантажень.

Необхідно реалізувати перегляд статистики в цілому по всім лігам для їх порівняння. Також необхідно реалізувати візуалізацію даних для кожної ліги окремо. Статистика окремих команд у рамках певної ліги має бути представлена графіком з розбиттям по місяцям і можливістю переглянути матчі в кожному окремому місяці. Користувачу повинен надаватися функціонал зміни мінімальної різниці в коефіцієнтах, котра буде вважатися помилкою.

2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Проектування бази даних інформаційної системи

Для роботи більшості додатків у сучасному світі необхідна база даних для збереження і зручної роботи з інформацією. У наш час існує багато варіантів СУБД

Системи управління базами даних (СУБД) – це набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних [5].

Найвідоміші СУБД – Oracle, Postgre, MySql і так далі. Вони відрізняються функціоналом та ресурсами, котрі використовують. Оскільки наша інформаційна система буде розміщена на сервері з незначною потужністю, нам підійде Postgre. Ця СУБД є збалансованим рішенням і пропонує достатній функціонал при невеликому використанні системних ресурсів.

PostgreSQL – об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

Першим кроком для проектування бази даних є визначення основних потоків даних у системі. Для цього необхідно створити DFD діаграму.

Діаграма потоків даних (англ. Data Flow Diagram) – це модель проектування, графічне представлення «потоків» даних в інформаційній системі [6]. Ця діаграма необхідна для усвідомлення процесів в системі та зображення потоків інформації. Нам необхідно спроектувати діаграми на нульовому (див. рис. 2.1) та першому рівнях (див. рис. 2.2).

Рисунок 2.1 DFD діаграма нульового рівня

Як бачимо, дві зовнішні сутності для взаємодії з системою – користувач та адміністратор.

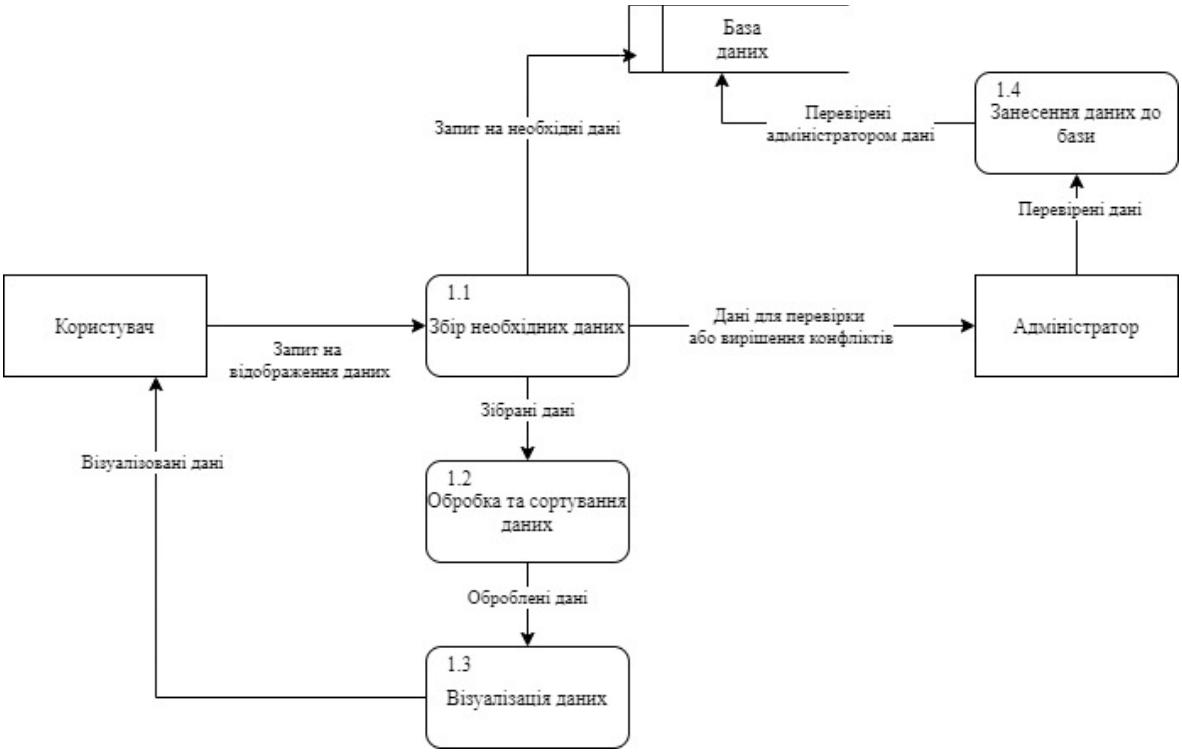

Рисунок 2.2 DFD діаграма першого рівня

Наступним кроком для проектування бази даних інформаційної системи є створення ERD діаграми (див. рис. 2.3). Цей вид діаграми описує сутності та зв’язки між ними. На основі такої діаграми можна безпосередньо створити таблиці нашої бази даних.

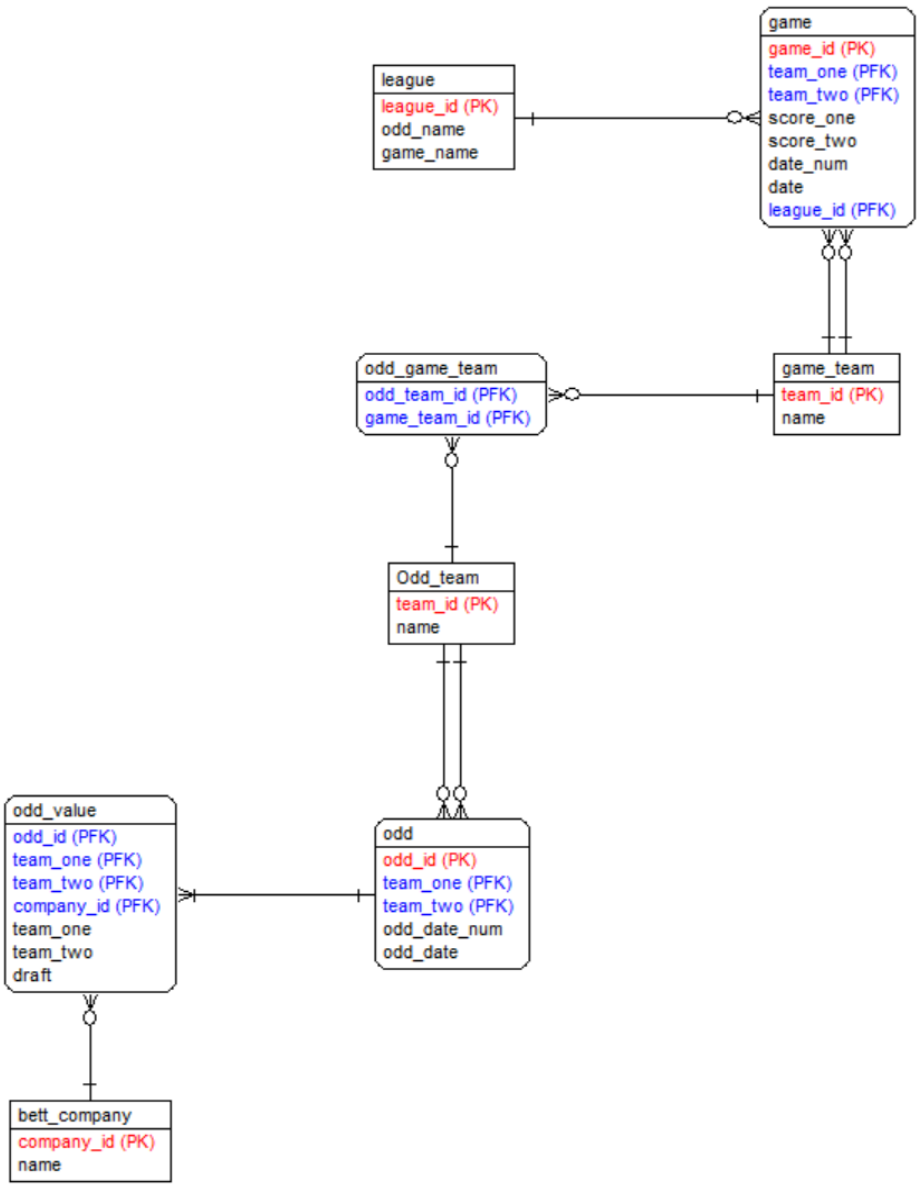

Рисунок 2.3 ERD діаграма

Таким чином, наша діаграма містить 7 сутностей. Опис цих сутностей необхідний для більш чіткої характеристики бази даних

Таблиця 2.1

Опис сутностей дігарами ERD

Сутність	Опис сутності	Таблиця	Поле	Опис поля
1	2	3	4	5
bett_ company	Представляє певну букмекерську компанію.	bett_ companies	company_id (PK)	Ідентифікатор букмекерської компанії, необхідний для однозначної ідентифікації.
			name	Назва букмекерської компанії.
league	Представляє певну лігу.	Leagues	league_id (PK)	Ідентифікатор ліги, необхідний для однозначної ідентифікації.
			odd_name	Ім’я ліги, отримане з API надання букмекерських даних.
			game_name	Ім’я ліги, отримане з API надання результатів матчів.

Продовження таблиці 2.1

1	2	3	4	5
game_team	Представляє певну команду, отриману з API результатів матчів.	game_teams	team_id (PK)	Унікальний ідентифікатор сутності, необхідний для однозначної ідентифікації.
			name	Ім’я команди, отримане з API надання результатів матчів.
odd_team	Представляє певну команду, отриману з API передбачень букмекерів.	odd_teams	team_id (PK)	Унікальний ідентифікатор сутності, необхідний для однозначної ідентифікації.
			name	Ім’я команди, отримане з API надання передбачень букмекерів.
game	Представляє певний матч.	games	game_id (PK)	Унікальний ідентифікатор, необхідний для однозначної ідентифікації сутності.
			team_one (PFK)	Ідентифікатор першої команди матчу, посилається на сутність game_team.
			team_two (PFK)	Ідентифікатор другої команди матчу, посилається на сутність game_team.

Продовження таблиці 2.1

1	2	3	4	5
			score_one	Кількість забитих голів першої команди.
			score_two	Кількість забитих голів другої команди.
			date_num	Число, що відповідає даті проведення матчу.
			date	Дата проведення матчу.
			league_id (PFK)	Ідентифікатор ліги, в котрій відбувався матч, посилається на сутність league.
odd	Представляє матч, на котрий є ередбачення від букмекерів.	odds	odd_id (PK)	Унікальний ідентифікатор, необхідний для однозначної ідентифікації сутності.
			team_one (PFK)	Ідентифікатор першої команди матчу, посилається на сутність odd_team.
			team_two (PFK)	Ідентифікатор другої команди матчу, посилається на сутність odd_team.
			odd_date_ num	Число, що відповідає даті проведення матчу.
			odd_date	Дата проведення матчу.
odd_value	Представляє значення передбачень від	odd_values	odd_id (PFK)	Ідентифікатор матчу, на котрий є передбачення, посилається на сутність odd.

Закінчення таблиці 2.1

1	2	3	4	5
	певного букмекера для конкретного матча.		company_id (PFK)	Ідентифікатор букмекерської компанії, котра зробила передбачення, посилається на сутність bett_company.
			team_one	Значення коефіцієнту на перемогу першої команди.
			team_two	Значення коефіцієнту на перемогу другої команди.
			draft	Значення коефіцієнту на нічию.
odd_game_team	Відображає зв'язок між командами, отриманими з API для передбачень результатів та самих результатів матчів.	odd_game_teams	game_team_id (PFK)	Ідентифікатор команди з API для результатів матчів, посилається на сутність game_team.
			odd_team_id (PFK)	Ідентифікатор команди з API для передбачень, посилається на сутність odd_team.

2.2. Архітектура інформаційної системи

Для реалізації інформаційної системи можна вибрати декілька варіантів реалізації додатку – це може бути клієнт-серверний додаток, окремий додаток з інтерфейсом, тощо. Для нашої інформаційної системи найдоцільнішим варіантом є розробка веб-додатку. Адже за наявності сервера, котрий забезпечить безперешкодну роботу, такий додаток є універсальним і не потребує від користувача нічого, окрім браузеру.

У нашій інформаційній системі повинні бути два окремих незалежних компоненти – безпосередньо веб-додаток, необхідний для обробки та візуалізації даних для користувача, та додаток для збирання даних (див. рис. 2.4).

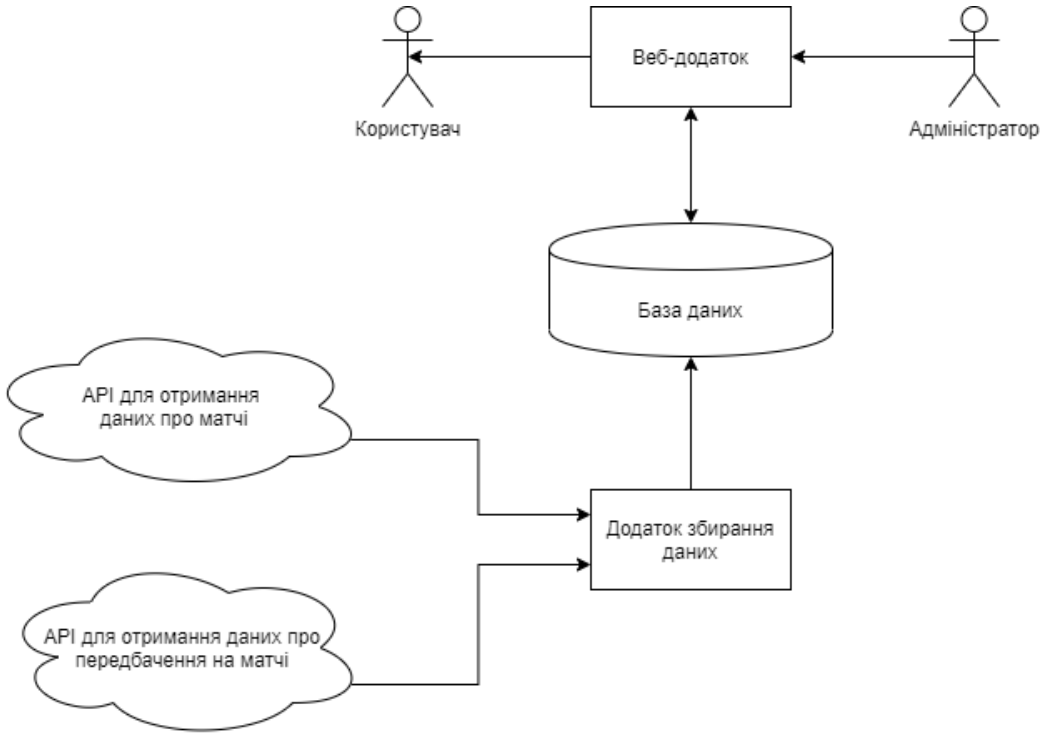

Рисунок 2.4 Схема роботи інформаційноїі системи

Додаток для збирання даних був реалізований в рамках виробничої практики і працював протягом усього футбольного сезону. Оскільки на той час база даних інформаційної системи ще не була розроблена, додаток збирав дані в файл у зручному форматі для подальшого використання.

Додаток необхідно зібрати в jar-файл та розмістити на сервері. Для його регулярної роботи потрібно налаштувати cron-таблицю. Такий спосіб регулярного виконання забезпечить надійність збору даних, адже навіть при перезавантаженні системи додаток буде запускатися вчасно. Кожного дня він збирає дані передбачень букмекерів на матчі наступного дня і результати матчів попереднього.

Дані записуються в окремі файли для результатів матчів (див. рис. 2.5) і для передбачень (див. рис. 2.6).

```
PPL|Vitória FC|0|CD Tondela|0|1565637300|2019-08-12T19:15
CL|Qarabağ Ağdam FK|0|APOEL FC|2|1565713800|2019-08-13T16:30
CL|Rosenborg BK|3|NK Maribor|1|1565715600|2019-08-13T17:00
CL|FK Dynamo Kyiv|3|Club Brugge KV|3|1565717400|2019-08-13T17:30
CL|FC København|1|FK Crvena Zvezda|1|1565719200|2019-08-13T18:00
CL|Ferencvárosi TC|0|GNK Dinamo Zagreb|4|1565719200|2019-08-13T18:00
CL|AFC Ajax|3|PAOK FC|2|1565721000|2019-08-13T18:30
CL|PAE Olympiakos SFP|2|Medipol Başakşehir FK|0|1565721000|2019-08-13T18:30
CL|LASK Linz|3|FC Basel 1893|1|1565721000|2019-08-13T18:30
CL|Celtic FC|3|FC CFR 1907 Cluj|4|1565721900|2019-08-13T18:45
CL|FC Porto|2|FK Krasnodar|3|1565722800|2019-08-13T19:00
```

Рисунок 2.5 Приклад даних результатів матчів

```
soccer_portugal_primeira_liga|Tondela|Vitoria Setubal|1565637300|2019-08-12T22:15|{Unibet|3.55|2.1|3.2|888sport|3.45|2.05|3.1|William Hill|3.75|
soccer_france_ligue_one|Angers|Lyon|1565981100|2019-08-16T21:45|{Unibet|7.0|1.4|4.9|888sport|7.0|1.4|4.8|1xBet|8.0|1.43|5.05|Ladbrokes|7.5|1.4|4
soccer_germany_bundesliga|Bayern Munich|Hertha Berlin|1565980200|2019-08-16T21:30|{Bet Victor|1.17|19.0|8.0|Marathon Bet|1.17|19.25|8.2|1xBet|1.
soccer_portugal_primeira_liga|FamalicÃ£o|Rio Ave FC|1565983800|2019-08-16T22:30|{Bet Victor|2.6|2.75|3.3|Paddy Power|2.4|2.7|3.1|Betfred|2.5|2.7
soccer_spain_la_liga|Athletic Bilbao|Barcelona|1565982000|2019-08-16T22:00|{Paddy Power|5.0|1.67|3.6|Sky Bet|4.6|1.73|3.8|Marathon Bet|5.05|1.74
soccer_epl|Arsenal|Burnley|1566041400|2019-08-17T14:30|{Unibet|1.3|9.0|5.4|888sport|1.33|9.5|5.5|1xBet|1.34|9.9|5.75|Sky Bet|1.36|8.0|4.8|Williar
soccer_epl|Liverpool|Southampton|1566050400|2019-08-17T17:00|{Unibet|1.45|6.5|4.6|888sport|1.48|6.75|4.7|1xBet|1.5|6.85|4.8|Sky Bet|1.5|6.0|4.33
soccer_epl|Newcastle United|Norwich City|1566050400|2019-08-17T17:00|{Unibet|3.3|2.2|3.2|888sport|3.4|2.25|3.3|1xBet|3.44|2.28|3.44|Sky Bet|3.25
soccer_epl|Everton|Watford|1566050400|2019-08-17T17:00|{Unibet|1.68|4.8|3.8|888sport|1.72|4.9|3.9|1xBet|1.79|4.86|3.9|Sky Bet|1.75|4.5|3.75|Will
soccer_epl|Brighton and Hove Albion|West Ham United|1566050400|2019-08-17T17:00|{Unibet|2.5|2.8|3.25|888sport|2.55|2.88|3.3|1xBet|2.6|2.9|3.44|S
soccer_epl|Aston Villa|Bournemouth|1566050400|2019-08-17T17:00|{Unibet|2.32|3.0|3.35|888sport|2.35|3.05|3.45|1xBet|2.38|3.11|3.58|Sky Bet|2.25|3
soccer_epl|Manchester City|Tottenham Hotspur|1566059400|2019-08-17T19:30|{Unibet|1.34|8.0|5.1|888sport|1.37|8.5|5.2|1xBet|1.39|8.8|5.35|Sky Bet|
```

Рисунок 2.6 Приклад даних передбачень на матчі

У подальшому ці дані будуть перетворені на sql-скрипт для заповнення бази даних.

За обробку і візуалізацію даних для кінцевого користувача буде відповідати веб-додаток. Оскільки на сервері, під час проходження виробничої практики, вже було встановлено JVM, то доцільним буде використання Java і для цієї задачі.

Для мови програмування Java існує багато фреймворків, котрі спрощують розробку веб-додатку. Один з них – Spark.

Spark – мікрофреймворк для створення веб-додатків за допомогою Java чи Kotlin з мінімальними зусиллями [7].

Використання невеликого фреймворку допоможе створити додаток, котрий не потребує значних ресурсів сервера. Це важливо в умовах незначної потужності.

Архітектура додатку може бути на основі одного з патернів проектування. Це дозволить написати добре структурований код, котрий у подальшому можна легко підтримувати. Тоді як система може дуже просто масштабуватися.

Існує багато патернів проектування. Для веб-додатку звичним є використання MVC (див. рис. 2.7).

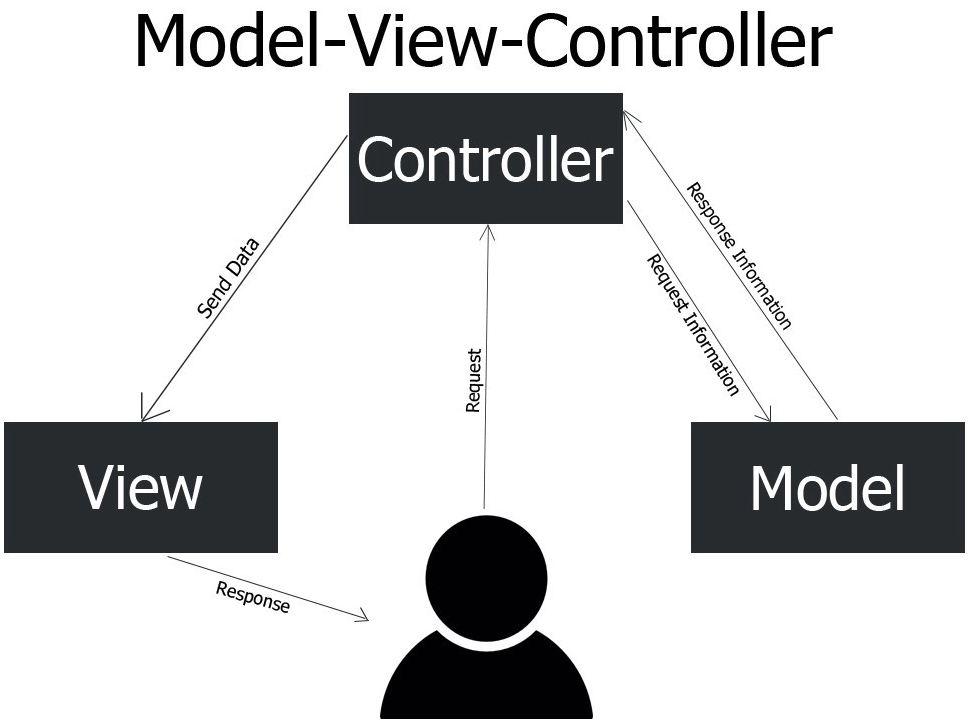

Рисунок 2.7 MVC патерн

Цей шаблон передбачає умовний поділ системи на три взаємопов’язані частини. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін

інтерфейсу користувача. Цей поділ є умовним і може бути дещо змінений в системі (див. рис. 2.8).

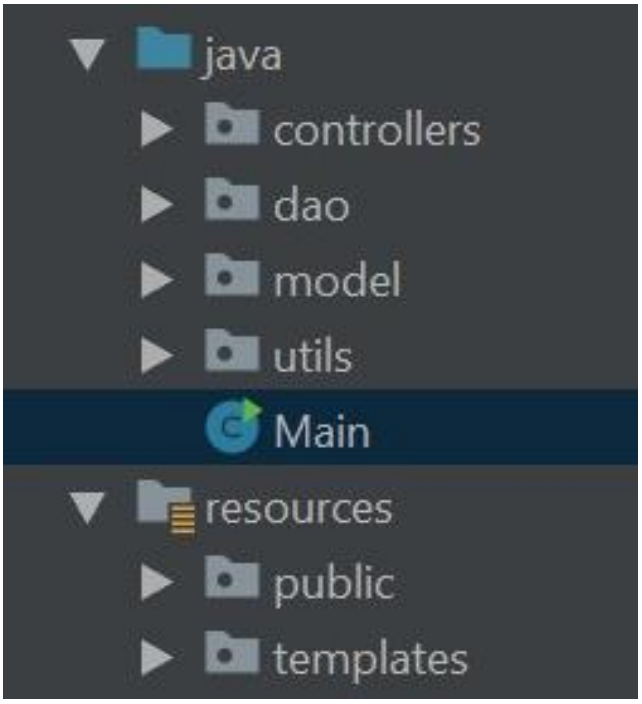

Рисунок 2.8 Структура веб-додатку інформаційної системи

У пакеті controllers містяться всі контролери додатку. Пакет dao використовує відповідає на запити контролерів і повертає об’єкти моделі, отримуючи їх з бази даних. Пакет utils містить необхідні інструменти для роботи додатку, такі як менеджер з’єднань з базою даних. Клас Main є точкою входу нашого додатку. Компонент view патерну MVC представлений пакетом templates, у якому знаходяться html-файли. Саме вони і є інтерфейсом для користувача.

Для зручного збирання нашого додатку у виконуваний jar-файл, використовуватимемо систему збирання Maven.

Apache Maven – фреймворк для автоматизації збирання проектів на основі описання їх структури в файлах за допомогою мови POM (підмножина XML) [8].

Цей фреймворк не впливає на кінцевий додаток, адже бере участь тільки в збиранні додатку. Це спростить використання бібліотек та формування залежностей.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1. Загальна структура інформаційної системи

Інформаційна система складається з бази даних та веб-додатку для обробки та візуалізації даних. База даних створена відповідно до ERD діаграми з другого розділу. Скрипт створення таблиць згенеровано за допомогою додатку CaseStudio.

Веб-додаток реалізовано за патерном MVC (див. рис. 3.1). Він складається з java-класів, котрі відповідають за серверну частину, та html-сторінок, які є інтерфейсом користувача.

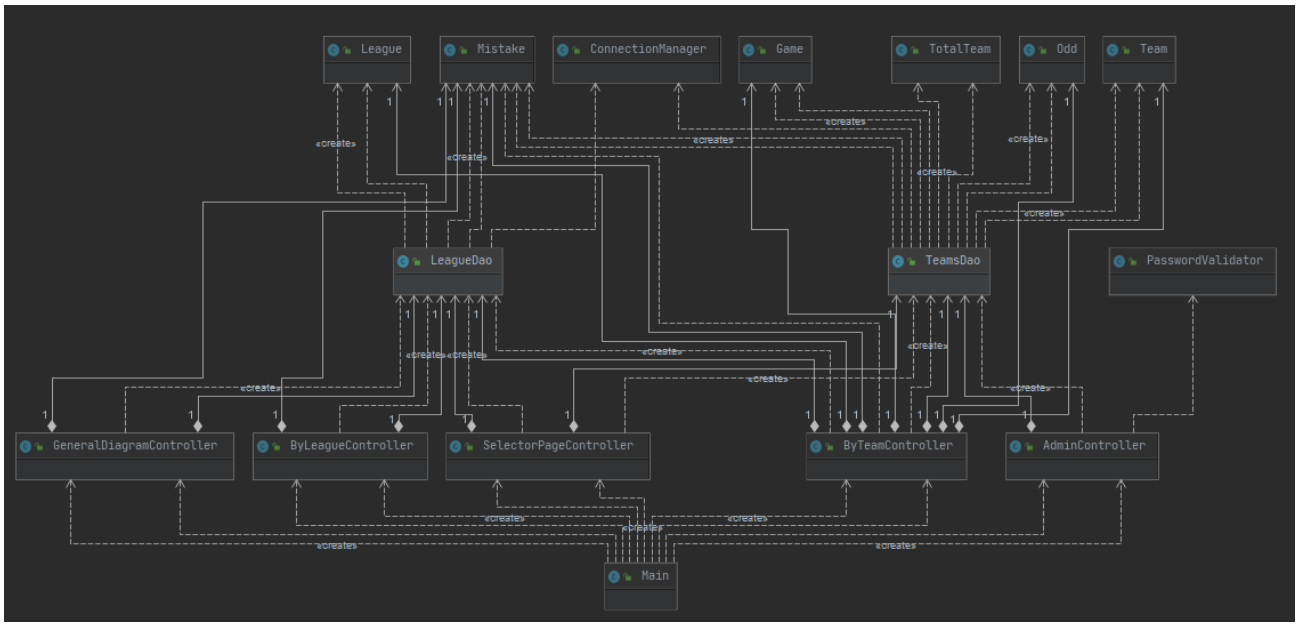

Рисунок 3.1 Діаграма класів веб-додатку

Як можемо бачити з діаграми класів, додаток використовує архітектуру MVC. Точкою входу є клас Main. Він використовує для обробки запитів контролери GeneralDiagramController, ByLeagueController, SelectorPageController, ByTeamController та AdminController. Кожен з них відповідає на запити з певної сторінки:

- GeneralDiagramController обробляє запити зі сторінки з загальною статистикою всіх ліг;

- ByLeagueController обробляє запити, пов’язані зі сторінкою статистики окремої ліги;
- ByTeamController обробляє запити зі сторінки статистики окремої команди;
- SelectorPageController обробляє запити з головної сторінки;
- AdminController обробляє запити зі сторінки адміністратора.

Кожен з контролерів використовує класи моделі та DAO. DAO працює з базою даних, використовуючи клас ConnectionManager та повертає модель контролеру. Кожен з класів DAO відповідає за певну сутність бази даних, для якої отримуються дані:

- LeagueDao отримує дані для окремих ліг чи всіх в цілому;
- TeamsDao отримує дані для окремої команди чи всіх команд в цілому.

Для створення підключення до бази даних необхідно на одному рівні файлової системи поряд з jar файлом розмістити директорію configurations, котра буде містити файл db.properties. Він містить всю необхідну інформацію для підключення а також пароль адміністратора ресурсу.

```
url = "url here"
database = "db name here"
username = "db username"
password = "db password"
adminPassword = "hashed admin password"
```

Такий спосіб конфігурації є безпечним і дозволить змінювати логіни і паролі без повторного збирання додатку.

3.2. Функціонал користувача

Основним функціоналом користувача є відображення візуалізованих даних, відсортованих за окремими критеріями. Для користувача реалізовано чотири веб-сторінки.

Головна сторінка дає можливість обрати варіант сортування даних (див. рис. 3.2).

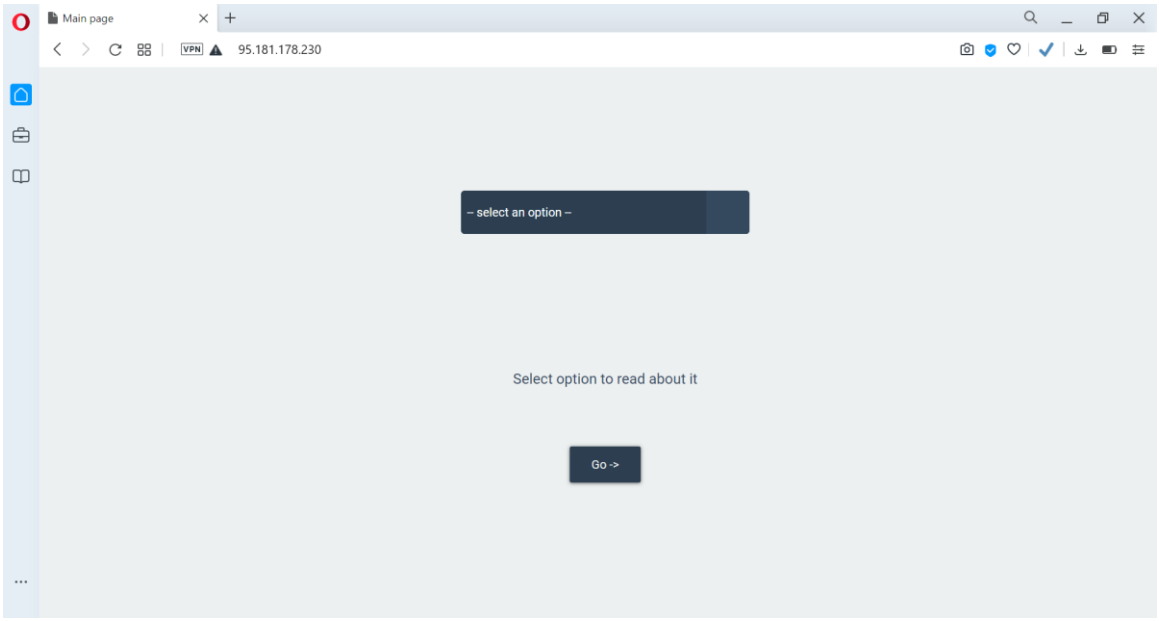

Рисунок 3.2 Головна сторінка

Необхідно обрати один з трьох варіантів сортування даних. При виборі з’явиться його опис (див. рис. 3.3).

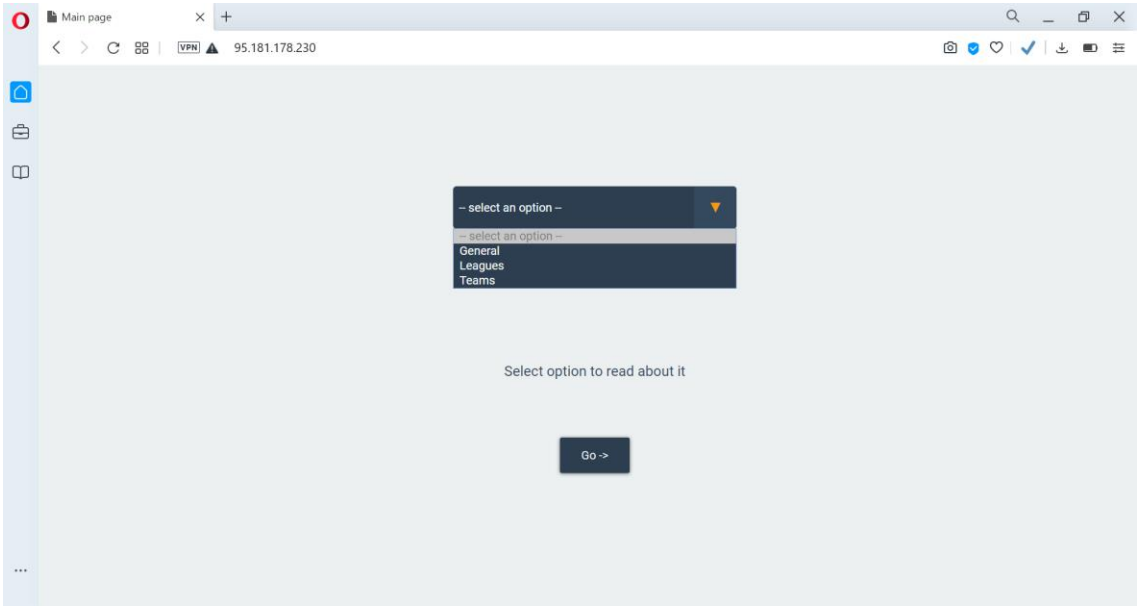

Рисунок 3.3 Варіанти сортування даних

При виборі другого варіанту з’являється додаткове поле зі списком ліг. Обравши третій варіант, спочатку з’являється додаткове поле зі списком ліг, а після вибору ліги з’являється ще одне поле зі списком команд цієї ліги (див. рис. 3.4).

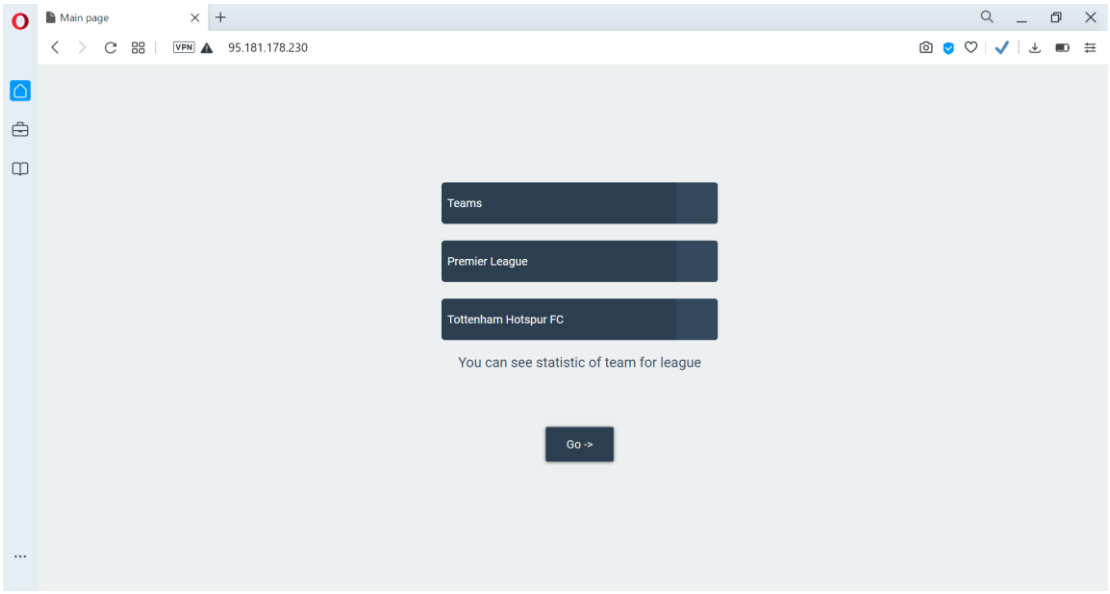

Рисунок 3.4 Сортування даних для окремої команди

Загальна діаграма показує кількість помилок для кожної ліги сумарно від усіх букмекерів у системі та кількість зіграних матчів (див. рис. 3.5). За замовчуванням, мінімальна різниця для похибки дорівнює одиниці. Для його зміни на кожній зі сторінок з даними є можливість налаштувати це значення (див. рис. 3.6). При цьому діаграма оновиться без перезавантаження сторінки.

Рисунок 3.5 Сторінка з загальною діаграмою

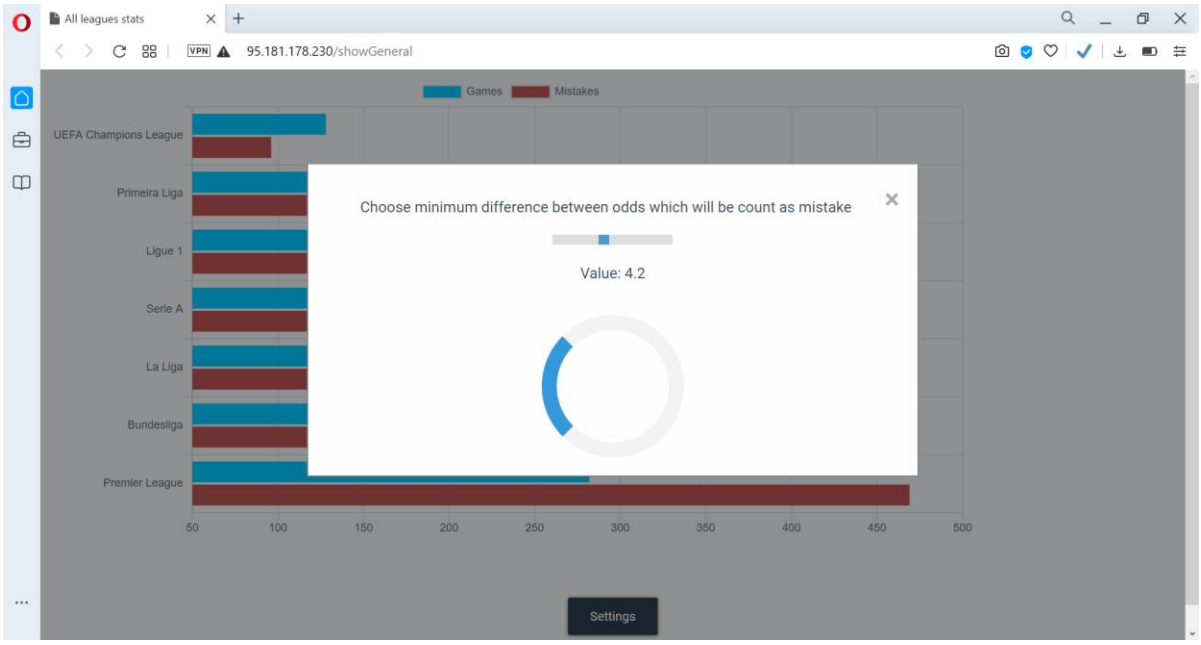

Рисунок 3.6 Вікно налаштувань під час обробки нового значення

Також є можливим відімкнути відображення ігор або помилок, натиснувши на відповідну назву на легенді діаграми.

Діаграма конкретної ліги відображає кількість помилок для кожної з команд ліги. Вони відсортовані за зростанням (див. рис. 3.7).

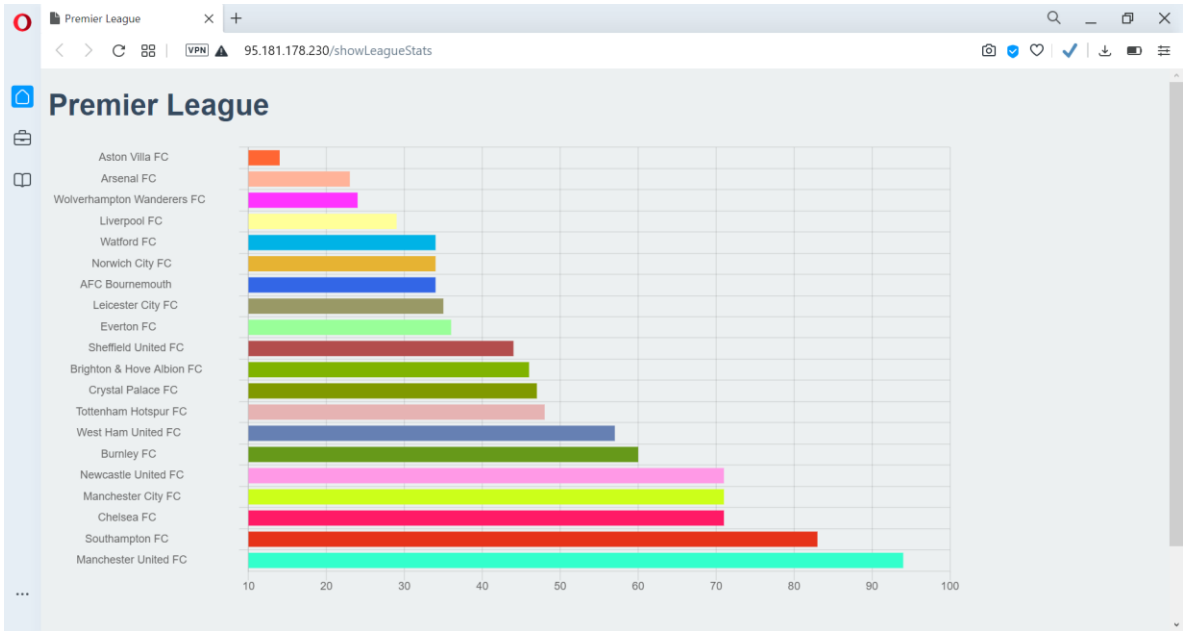

Рисунок 3.7 Дані для окремої ліги

На цій сторінці так само є можливість налаштування мінімальної різниці. А на сторінці окремої команди відображається кількість помилок за місяцями (див. рис. 3.8). При натисканні на певне значення можна переглянути список ігор у відповідному місяці (див. рис. 3.9).

Рисунок 3.8 Статистика окремої команди

При натисканні на елемент легенди графіку можна прибрати його відображення.

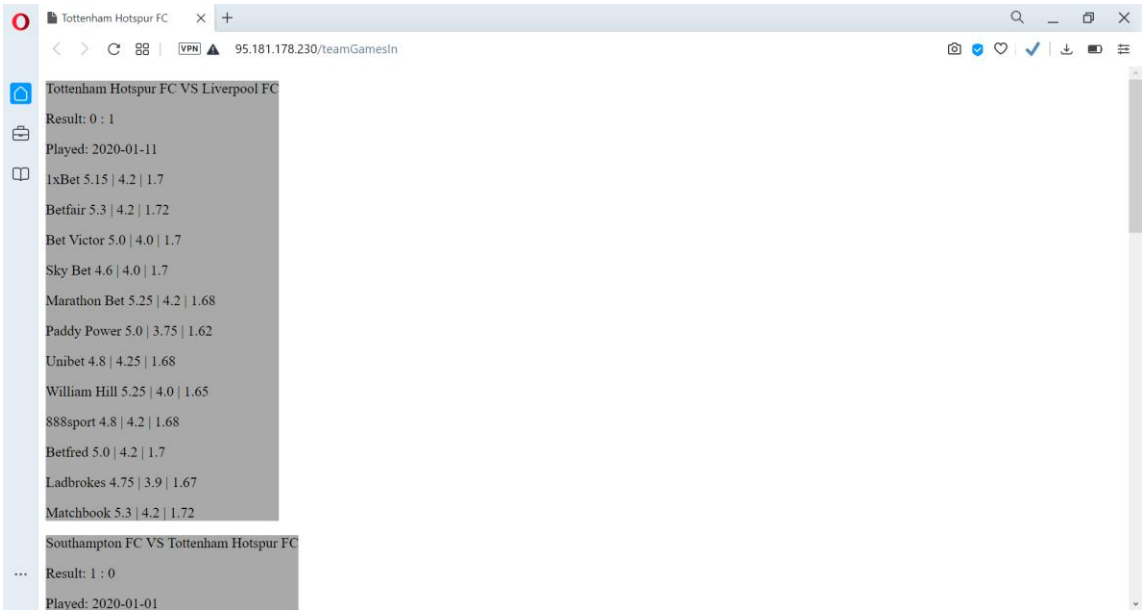

Рисунок 3.9 Інформація про матчі команди в конкретному місяці

У заголовку вказується назва команди і ліга, для якої відображається статистика. Це уточнюється, адже деякі команди мають матчі в двох лігах – своєї країни та Ліги Чемпіонів. При виборі певного значення (не важливо якого графіку), ми побачимо всі ігри відповідного місяця і передбачення всіх букмекерів на цей матч.

3.3. Функціонал адміністратора

Основним функціоналом адміністратора є встановлення зв’язків між командами, отриманими з різних API. Найпростіший і найбільш надійний спосіб це робити – вручну. Проблема виникає через різні назви команд в залежності від API та навіть ліги, у якій вони грали матч. Так, команда FC Internazionale Milano може також мати назву FC Internazionale або Internazionale Milano. Також проблема може виникнути через наявність літер, відмінних від англійського алфавіту.

Щоб потрапити на сторінку, необхідно ввести пароль адміністратора (див. рис. 3.10). У подальшому він буде переданий на сервер та порівняється з зашифрованим за допомогою ByCrypt паролем.

bcrypt – адаптивна криптографічна функція формування ключа, що використовується для безпечного зберігання паролів. Розробники: Нільс Провос і David Mazières. Функція заснована на шифрі Blowfish, вперше представлена на USENIX у 1999 році [9].

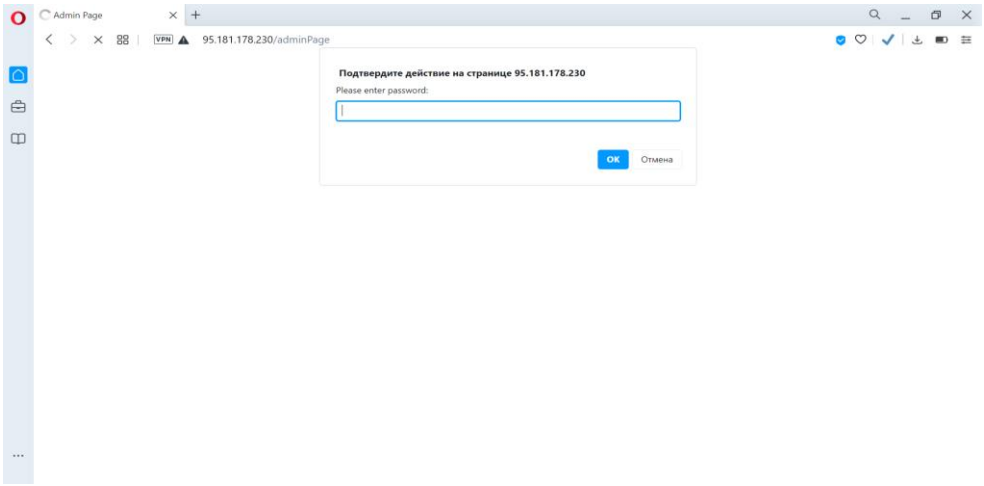

Рисунок 3.10 Запит на ведення пароля адміністратора

При невірному введені пароля сторінка не буде відображена. Для повторного введення пароля, якщо спроба була невдала, необхідно перезавантажити сторінку.

Сторінка складається з трьох основних частин – таблиці, що містить вже відсортовані команди (див. рис. 3.11); таблиці, у якій відображається пара команд, яка буде додана до списку відсортованих; список всіх команд.

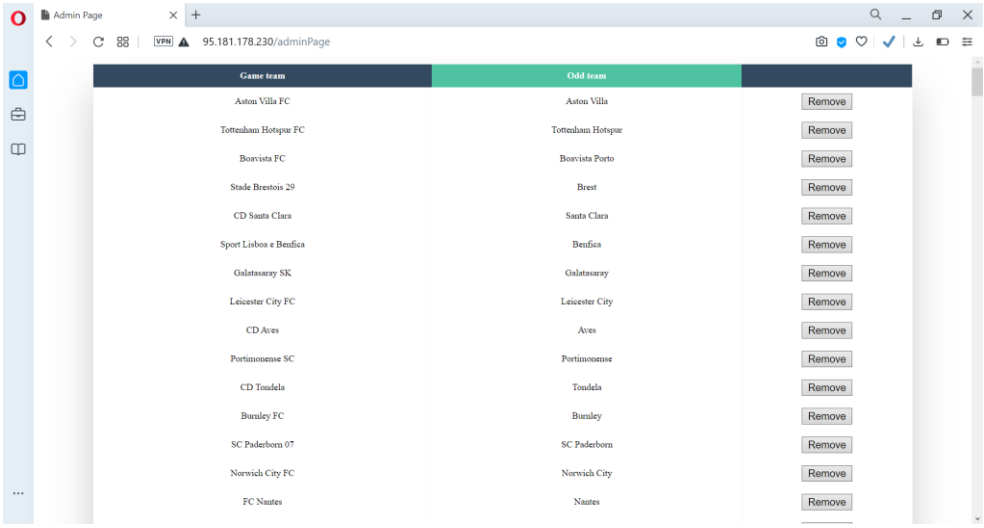

Рисунок 3.11 Таблиця з відсортованими командами

Таким чином, кожну пару команд можна видалити. Тоді в списку всіх команд вони не будуть позначатися як вже відсортовані.

У таблиці всіх команд сірим кольором відображається команда, яка вже має відповідну пару з іншого API. Білим кольором – та, котра не має відповідника. Такі команди теж можуть бути, адже не на всі матчі букмекери робили передбачення (див. рис. 3.12).

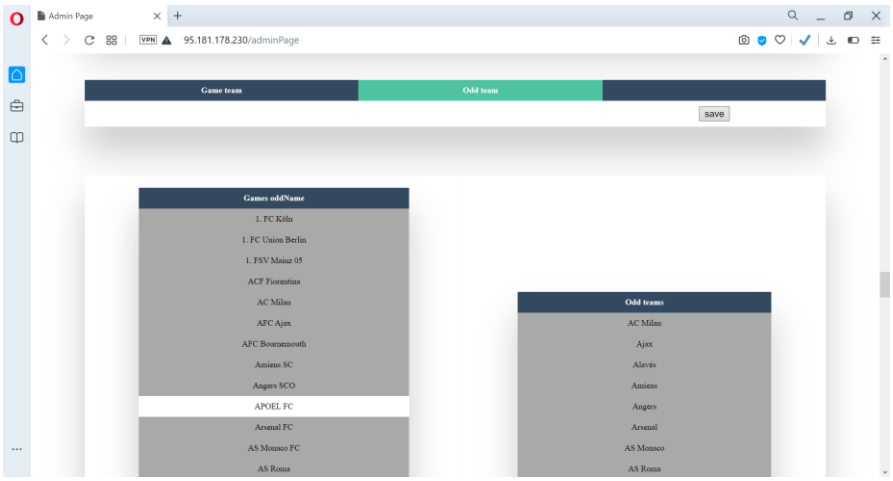

Рисунок 3.12 Таблиця з обраними командами та таблиці зі списками всіх команд

Обрана команда підсвічується кольором (див. рис. 3.13).

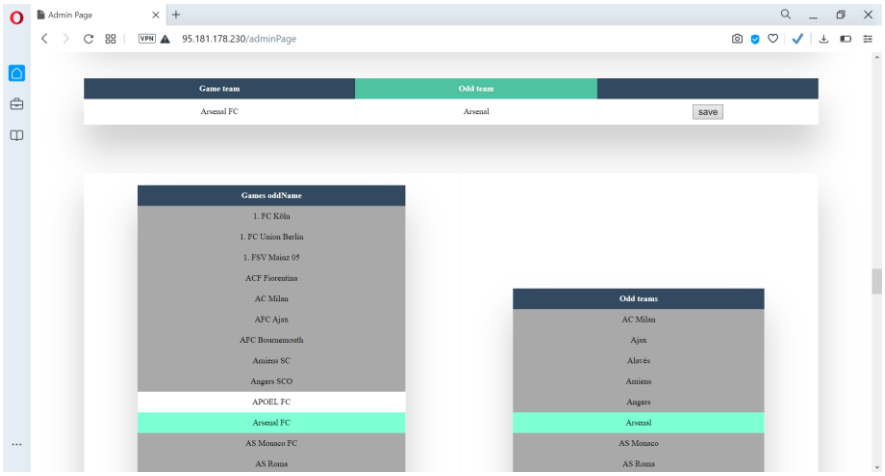

Рисунок 3.13 Обирання команд для додавання до списку відсортованих

При натисканні на команду вона додається до форми зберігання. Після натискання на кнопку «зберегти» пара буде додана до загального списку відсортованих команд.

ВИСНОВКИ

У результаті аналізу проблем дослідження було з’ясовано, що існують схожі додатки, що надають візуалізовані дані для користувача з метою подальшого аналізу, такі як oddsportal.com, soccerstats.com, whoscored.com, flashscore.com, ice2bet.com. Також розглянуто сервіси, котрі надають безпосередньо інформацію без її візуалізації у вигляді JSON, а саме sportmonks.com, allsportsapi.com, apifootball.com, football-data.org, the-odds-api.com. Ці варіанти було проаналізовано та визначено їх переваги та недоліки. Існуючі рішення, розглянуті в дослідженні, не задовольняють вимоги нашої інформаційної системи через брак наочності або неможливість регулярного отримання даних.

Виходячи з аналізу проблем, було сформовано задачі дослідження. Дані необхідно отримувати регулярно з двох джерел – сервісу, котрий надає передбачення на матчі та сервісу з результатами ігор (the-odds-api.com та football-data.org).

Результатом проектування бази даних є DFD діаграма (для відображення потоків інформації в системі) та ERD діаграма (для зображення основних сутностей системи та зв’язків між ними).

Визначено структуру веб-додатку інформаційної системи. Для його реалізації мовою Java обрано шаблон проектування MVC. У результаті розробки інформаційної системи було описано загальну структуру веб-додатку, надано UML-діаграму всіх класів. Структурно-значущі класи було описано для більш чіткого розуміння архітектури додатку. Веб-додаток містить три варіанти візуалізації даних. Доступ до них здійснюється за допомогою головної сторінки. Усі вони виконані в мінімалістичному дизайні. Окремо реалізовано функціонал сторінки адміністратора.

Результатом розробки інформаційної системи є веб-додаток, який здатен візуалізувати дані про футбольні матчі та передбачення букмекерських компаній на

ці матчі. Переглянути працюючий прототип можна за посиланням http://95.181.178.230. Отже, поставлені задачі були виконані. Працюючий прототип інформаційної системи свідчить про досягнення мети дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бєлка Д.О. Моделі прогнозування футбольних матчів – URL: https://naub.oa.edu.ua/2019/моделі-прогнозування-футбольних-мат/
2. Groll A., Ley C., Schauberger G., Van Eetvelde H. – URL: https://arxiv.org/pdf/1806.03208v2.pdf
3. Freitag D. Documentation – URL: https://www.football-data.org/documentation/api
4. The odds-api – URL: https://the-odds-api.com
5. Silberschatz A., Sudarshan S. Database system concepts. New York: McGraw-Hill., 2011.
6. Data Flow Diagram – URL: https://web.archive.org/web/20061023174119/http://www.infoarchgroup.com/qrdfd.htm
7. Spark – URL: http://sparkjava.com
8. Maven – URL: http://maven.apache.org/what-is-maven.html
9. Provos N., Mazières D., A Future-Adaptable Password Scheme. USENIX Annual Technical Conference. 81–92. 2012
10. Кей С. Java SE 8. Вводный курс. «Вильямс». 2014.
11. PL/pgSQL — процедурный язык SQL – URL: https://postgrespro.ru/docs/
12. jQuery.ajax() – URL: https://api.jquery.com/jquery.ajax/
13. HTML The language for building web pages – URL: https://www.w3schools.com
14. Chart.js Simple yet flexible JavaScript charting for designers & developers – URL: https://www.chartjs.org
15. Borovikov R. Top 10 JavaScript Charting Libraries for Every Data Visualization Need – URL: https://hackernoon.com/10-javascript-charting-libraries-data-visualization-b77523d23372

ДОДАТКИ

Додаток 1

В даному додатку представлено код класу Main

```
import controllers.*;
import static spark.Spark.*;
public class Main {
    public static void main(String[] args){
        AdminController adminController = new AdminController();
        SelectorPageController getSelectorPage = new SelectorPageController();
        GeneralDiagramController generalDiagramController = new
GeneralDiagramController();
        ByTeamController byTeamController = new ByTeamController();
        ByLeagueController byLeagueController = new ByLeagueController();
        port(80);
        staticFileLocation("/public");
        get("/adminPage", adminController.getAdminPage);
        post("/resolveTeam", adminController.resolveTeam);
        post("/deleteResolvedTeam", adminController.deleteResolvedTeam);
        post("/checkPassword", adminController.checkPassword);
        get("/mainPage", getSelectorPage.getSelectorPage);
        get("/getLeagues", getSelectorPage.getLeagues);
        get("/getTeams", getSelectorPage.getTeams);
        get("/showGeneral", generalDiagramController.getGeneralDiagram);
        get("/showGeneralWithDiff", generalDiagramController.getGeneralDiagramWithDiff);
        get("/showTeamStat", byTeamController.getGeneralDiagram);
        get("/showTeamStatWithDiff", byTeamController.getGeneralDiagramWithDiff);
        get("/teamGamesIn", byTeamController.getGamesForMonth);
        get("/showLeagueStats", byLeagueController.getLeagueStats);
        get("/showLeagueStatsWithDiff", byLeagueController.getLeagueStatsWithDiff);
        get("/", getSelectorPage.getSelectorPage);

    }
}
```

Додаток 2

В цьому додатку представлено код класу AdminController

```
package controllers;

import dao.TeamsDao;
import spark.ModelAndView;
import spark.Route;
import spark.template.thymeleaf.ThymeleafTemplateEngine;
import utils.PasswordValidator;

import java.util.HashMap;

public class AdminController {
    private TeamsDao teamsDao= new TeamsDao();
    public final Route getAdminPage = ((request, response) -> {
        HashMap<String, Object> params = new HashMap<>();
        params.put("gameTeam", teamsDao.getGameTeams());
        params.put("oddTeam", teamsDao.getOddTeams());
        params.put("totalTeam", teamsDao.getTotalTeams());
        return new ThymeleafTemplateEngine().render(new ModelAndView(params,"adminPage"));
    });

    public final Route checkPassword = ((request, response) -> {
        String password = request.queryParams("password");
        if(PasswordValidator.isPasswordsEqual(password)){
            request.session().attribute("auth", true);
            System.out.println("true");
            return true;
        }
        return "false";
    });
    public final Route resolveTeam = ((request, response) -> {
        if(!(Boolean) request.session().attribute("auth")) return "error";
        int gameId = Integer.parseInt(request.queryParams("gameId"));
        int oddId = Integer.parseInt(request.queryParams("oddId"));
        teamsDao.resolveTeams(gameId, oddId);
        response.status(200);
        return "ok";
    });

    public final Route deleteResolvedTeam = ((request, response) -> {
        if(!(Boolean) request.session().attribute("auth")) return "error";
        int gameId = Integer.parseInt(request.queryParams("gameId"));
        int oddId = Integer.parseInt(request.queryParams("oddId"));
        teamsDao.deleteResolvedTeam(gameId, oddId);
        response.status(200);
        return "ok";
    });
}
```

Додаток 3

В цьому додатку представлено код класу ByLeagueController

```
package controllers;

import dao.LeagueDao;
import model.Mistake;
import org.json.JSONArray;
import org.json.JSONObject;
import spark.ModelAndView;
import spark.Route;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

import java.util.HashMap;
import java.util.List;

public class ByLeagueController {
    LeagueDao leagueDao = new LeagueDao();
    public final Route getLeagueStats = ((request, response) -> {
        int league_id = Integer.parseInt(request.queryParams("league_id"));
        List<Mistake> leagueStatsByTeam = leagueDao.getLeagueStatsByTeam(1, league_id);
        HashMap<String, Object> model = new HashMap<>();

        JSONObject data = new JSONObject();
        JSONArray labels = new JSONArray();
        JSONArray mistakes = new JSONArray();
        for (Mistake teamMistake : leagueStatsByTeam) {
            labels.put(teamMistake.getName());
            mistakes.put(teamMistake.getMistakes());
        }
        data.put("labels", labels);
        JSONArray datasets = new JSONArray();
        JSONObject mistakesDataset = new JSONObject();
        mistakesDataset.put("label", "Mistakes");
        mistakesDataset.put("data", mistakes);
        JSONArray colors = new
JSONArray().put("#FF6633").put("#FFB399").put("#FF33FF").put("#FFFF99").put("#00B3E6").put
(

"#E6B333").put("#3366E6").put("#999966").put("#99FF99").put("#B34D4D").put(

"#80B300").put("#809900").put("#E6B3B3").put("#6680B3").put("#66991A").put(

"#FF99E6").put("#CCFF1A").put("#FF1A66").put("#E6331A").put("#33FFCC").put(

"#66994D").put("#B366CC").put("#4D8000").put("#B33300").put("#CC80CC").put(

"#66664D").put("#991AFF").put("#E666FF").put("#4DB3FF").put("#1AB399").put(

"#E666B3").put("#33991A").put("#CC9999").put("#B3B31A").put("#00E680").put(
```

```
"#4D8066").put("#809980").put("#E6FF80").put("#1AFF33").put("#999933").put(

"#FF3380").put("#CCCC00").put("#66E64D").put("#4D80CC").put("#9900B3").put(
                "#E64D66").put("#4DB380").put("#FF4D4D").put("#99E6E6").put("#6666FF");
        mistakesDataset.put("backgroundColor", colors);
        datasets.put(mistakesDataset);
        data.put("datasets", datasets);
        model.put("json", data.toString());
        model.put("message", leagueDao.getLeagueById(league_id).getOdd_name());
        return new ThymeleafTemplateEngine().render(new ModelAndView(model,
"leagueStats"));
    });

    public final Route getLeagueStatsWithDiff = ((request, response) -> {
        int league_id = Integer.parseInt(request.queryParams("league_id"));
        String diff = request.queryParams("diff");
        List<Mistake> leagueStatsByTeam =
leagueDao.getLeagueStatsByTeam(Double.parseDouble(diff), league_id);
        JSONArray mistakes = new JSONArray();
        for (Mistake teamMistake : leagueStatsByTeam) {
            mistakes.put(teamMistake.getMistakes());
        }
        System.out.println(mistakes.toString());
        return mistakes.toString();
    });
}
```

Додаток 4

В цьому додатку представлено код класу ByTeamController

```
package controllers;

import dao.LeagueDao;
import dao.TeamsDao;
import model.*;
import org.json.JSONArray;
import org.json.JSONObject;
import spark.ModelAndView;
import spark.Route;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

import java.util.HashMap;
import java.util.List;

public class ByTeamController {
    TeamsDao teamsDao = new TeamsDao();
    LeagueDao leagueDao = new LeagueDao();
    public final Route getGeneralDiagram = ((request, response) -> {
        int teamId = Integer.parseInt(request.queryParams("team_id"));
        int league_id = Integer.parseInt(request.queryParams("league_id"));
        List<Mistake> teamMistakes = teamsDao.teamMistakes(1.0, teamId, league_id);
        League league = leagueDao.getLeagueById(league_id);
        Team team = teamsDao.getTeamById(teamId);
        HashMap<String, Object> params = new HashMap<>();
        params.put("json", generateDataJson(teamMistakes));
        params.put("league", league.getOdd_name());
        params.put("team", team.getName());
        return new ThymeleafTemplateEngine().render(new ModelAndView(params,"teams"));
    });

    public final Route getGeneralDiagramWithDiff = ((request, response) -> {
        String diff = request.queryParams("diff");
        int teamId = Integer.parseInt(request.queryParams("team_id"));
        int league_id = Integer.parseInt(request.queryParams("league_id"));
        List<Mistake> teamMistakes = teamsDao.teamMistakes(Double.parseDouble(diff),
teamId, league_id);
        JSONArray games = new JSONArray();
        JSONArray mistakes = new JSONArray();
        for(Mistake teamMistake : teamMistakes){
            games.put(teamMistake.getGames());
            mistakes.put(teamMistake.getMistakes());
        }
        return new JSONArray().put(games).put(mistakes).toString();
    });

    public final Route getGamesForMonth = ((request, response) -> {
        String month = request.queryParams("mounth");
        int teamId = Integer.parseInt(request.queryParams("team_id"));
```

```
        int leagueId = Integer.parseInt(request.queryParams("league_id"));
        HashMap<Game, List<Odd>> gamesForMonth = teamsDao.getGamesForMonth(teamId,
leagueId, month);
        HashMap<String, Object> model = new HashMap<>();
        model.put("games", gamesForMonth);
        String team_name = teamsDao.getTeamById(teamId).getName();
        model.put("team_name", team_name);
        return new ThymeleafTemplateEngine().render(new
ModelAndView(model,"gamesForMonth"));
    });


    private String generateDataJson(List<Mistake> teamMistakes){
        JSONObject data = new JSONObject();
        JSONArray labels = new JSONArray();
        JSONArray games = new JSONArray();
        JSONArray mistakes = new JSONArray();
        for(Mistake teamMistake : teamMistakes){
            labels.put(teamMistake.getName());
            games.put(teamMistake.getGames());
            mistakes.put(teamMistake.getMistakes());
        }
        data.put("labels",labels);
        JSONArray datasets = new JSONArray();
        JSONObject gamesDataset= new JSONObject();
        gamesDataset.put("label","Games");
        gamesDataset.put("data", games);
        JSONObject mistakesDataset= new JSONObject();
        mistakesDataset.put("label", "Mistakes");
        mistakesDataset.put("data", mistakes);
        gamesDataset.put("borderColor", "rgb(0, 196, 255)");
        gamesDataset.put("fill", false);
        gamesDataset.put("lineTension", 0);
        mistakesDataset.put("borderColor", "rgb(185, 83, 83)");
        mistakesDataset.put("fill", false);
        mistakesDataset.put("lineTension", 0);
        datasets.put(gamesDataset);
        datasets.put(mistakesDataset);
        data.put("datasets",datasets);
        return data.toString();
    }
}
```

Додаток 5

В цьому додатку представлено код класу GeneralDiagramController

```
package controllers;

import dao.LeagueDao;
import model.Mistake;
import spark.ModelAndView;
import spark.Route;
import spark.template.thymeleaf.ThymeleafTemplateEngine;
import org.json.JSONArray;
import org.json.JSONObject;

import java.util.HashMap;
import java.util.List;

public class GeneralDiagramController {
    private LeagueDao leagueDao = new LeagueDao();
    public final Route getGeneralDiagram = ((request, response) -> {
        List<Mistake> leagueMistakeList = leagueDao.getLeagueMistakes(1);
        JSONObject root = new JSONObject();
        root.put("type", "bar");
        JSONObject data = new JSONObject();
        JSONArray labels = new JSONArray();
        JSONArray games = new JSONArray();
        JSONArray mistakes = new JSONArray();
        for(Mistake leagueMistake : leagueMistakeList){
            labels.put(leagueMistake.getName());
            games.put(leagueMistake.getGames());
            mistakes.put(leagueMistake.getMistakes());
        }
        data.put("labels",labels);
        JSONArray datasets = new JSONArray();
        JSONObject gamesDataset= new JSONObject();
        gamesDataset.put("label","Games");
        gamesDataset.put("data", games);
        JSONObject mistakesDataset= new JSONObject();
        mistakesDataset.put("label", "Mistakes");
        mistakesDataset.put("data", mistakes);
        gamesDataset.put("backgroundColor", "rgb(0, 196, 255)");
        mistakesDataset.put("backgroundColor", "rgb(185, 83, 83)");
        datasets.put(gamesDataset);
        datasets.put(mistakesDataset);
        data.put("datasets",datasets);
        root.put("data", data);
        JSONObject options = new JSONObject();
        JSONObject scales = new JSONObject();
        scales.put("xAxes", new JSONArray().put(new JSONObject().put("stacked", "true")));
        scales.put("yAxes", new JSONArray().put(new JSONObject().put("stacked", "true")));
        options.put("scales",scales);
        root.put("options", options);
```

```
        HashMap<String, Object> params = new HashMap<>();
        params.put("json", data.toString());
        return new ThymeleafTemplateEngine().render(new ModelAndView(params,"leagues"));
    });

    public final Route getGeneralDiagramWithDiff = ((request, response) -> {
        String diff = request.queryParams("diff");
        List<Mistake> leagueMistakeList =
leagueDao.getLeagueMistakes(Double.parseDouble(diff));
        JSONObject root = new JSONObject();
        root.put("type", "bar");
        JSONObject data = new JSONObject();
        JSONArray labels = new JSONArray();
        JSONArray games = new JSONArray();
        JSONArray mistakes = new JSONArray();
        for(Mistake leagueMistake : leagueMistakeList){
            labels.put(leagueMistake.getName());
            games.put(leagueMistake.getGames());
            mistakes.put(leagueMistake.getMistakes());
        }
        data.put("labels",labels);
        JSONArray datasets = new JSONArray();
        JSONObject gamesDataset= new JSONObject();
        gamesDataset.put("label","Games");
        gamesDataset.put("data", games);
        JSONObject mistakesDataset= new JSONObject();
        mistakesDataset.put("label", "Mistakes");
        mistakesDataset.put("data", mistakes);
        gamesDataset.put("backgroundColor", "rgb(0, 196, 255)");
        mistakesDataset.put("backgroundColor", "rgb(185, 83, 83)");
        datasets.put(gamesDataset);
        datasets.put(mistakesDataset);
        data.put("datasets",datasets);
        root.put("data", data);
        JSONObject options = new JSONObject();
        JSONObject scales = new JSONObject();
        scales.put("xAxes", new JSONArray().put(new JSONObject().put("stacked", "true")));
        scales.put("yAxes", new JSONArray().put(new JSONObject().put("stacked", "true")));
        options.put("scales",scales);
        root.put("options", options);
        return new JSONArray().put(games).put(mistakes).toString();
    });

}
```

Додаток 6

В цьому додатку представлено код класу SelectorPageController

```
package controllers;

import com.google.gson.Gson;
import dao.LeagueDao;
import dao.TeamsDao;
import spark.ModelAndView;
import spark.Route;
import spark.template.thymeleaf.ThymeleafTemplateEngine;

import java.util.HashMap;

public class SelectorPageController {
    private LeagueDao leagueDao = new LeagueDao();
    private TeamsDao teamDao = new TeamsDao();
    private Gson gson = new Gson();
    public final Route getSelectorPage = ((request, response) -> {
        HashMap<String, Object> params = new HashMap<>();
        return new ThymeleafTemplateEngine().render(new ModelAndView(params,"selector"));
    });

    public final Route getLeagues = ((request, response) -> {
        return gson.toJson(leagueDao.getLeagues());
    });

    public final Route getTeams = ((request, response) -> {
        int league = Integer.parseInt(request.queryParams("league"));
        return gson.toJson(teamDao.getResolvedTeams(league));
    });
}
```

Додаток 7

В цьому додатку представлено код класу LeagueDao

```
package dao;

import model.League;
import model.Mistake;
import utils.ConnectionManager;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class LeagueDao {

    public List<Mistake> getLeagueMistakes(double diff) {
        String sql = "SELECT odd_name, (f).* FROM  (SELECT odd_name,
get_mistakes_for_league(league_id, ?) AS f FROM leagues) sub order by 2;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setBigDecimal(1, BigDecimal.valueOf(diff));
            ResultSet resultSet = statement.executeQuery();
            return pareLeagueMistakes(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public List<Mistake> getLeagueStatsByTeam(double diff, int league) {
        String sql ="select gt.name, sum(get_mistake_for_game(g.game_id, g.score_one,
g.score_two, ?)) m, 0 from games g\n" +
                "join \n" +
                "(select team_one as team_id from games where league_id =?\n" +
                "union \n" +
                "select team_two from games where league_id=?) t\n" +
                "on (g.team_one=t.team_id or g.team_two=t.team_id)\n" +
                "join game_teams gt on gt.team_id=t.team_id\n" +
                "where t.team_id in (select game_team_id from odd_game_teams)\n\n" +
                "group by gt.name order by m;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setBigDecimal(1, BigDecimal.valueOf(diff));
            statement.setInt(2, league);
            statement.setInt(3, league);
            ResultSet resultSet = statement.executeQuery();
            return pareLeagueMistakes(resultSet);
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        return null;
    }
    public List<League> getLeagues() {
        String sql = "select * from leagues;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            ResultSet resultSet = statement.executeQuery();
            return parseLeagues(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public League getLeagueById(int id) {
        String sql = "select * from leagues where league_id=?;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, id);
            ResultSet resultSet = statement.executeQuery();
            return parseLeagues(resultSet).get(0);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    private List<League> parseLeagues(ResultSet resultSet) throws SQLException {
        List<League> leagues = new ArrayList<>();
        while (resultSet.next()) {
            League league = new League();
            league.setLeague_id(resultSet.getInt(1));
            league.setOdd_name(League.generateCustmLeagueName(resultSet.getString(2)));
            league.setGame_name(resultSet.getString(3));
            leagues.add(league);
        }
        return leagues;
    }
    private List<Mistake> pareLeagueMistakes(ResultSet resultSet) throws SQLException {
        List<Mistake> mistakes = new ArrayList<>();
        while (resultSet.next()) {
            Mistake mistake = new Mistake();
            String name = resultSet.getString(1);
            mistake.setName(League.generateCustmLeagueName(name));
            mistake.setMistakes(resultSet.getInt(2));
            mistake.setGames(resultSet.getInt(3));
            mistakes.add(mistake);
        }
        return mistakes;
    }}
```

Додаток 8

В цьому додатку представлено код класу TeamsDao

```
package dao;

import model.*;
import utils.ConnectionManager;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class TeamsDao {

    public List<TotalTeam> getTotalTeams() {
        String sql = "select o.name, g.name, odd_team_id, game_team_id from odd_game_teams
ogt \n" +
                "join odd_teams o on ogt.odd_team_id = o.team_id\n" +
                "join game_teams g on ogt.game_team_id = g.team_id;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            ResultSet resultSet = statement.executeQuery();
            return parseTotalTeams(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public void resolveTeams(int gameId, int oddId) {
        String sql = "insert into odd_game_teams values(?,?)";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, oddId);
            statement.setInt(2, gameId);
            statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void deleteResolvedTeam(int gameId, int oddId) {
        String sql = "delete from odd_game_teams where odd_team_id =? and game_team_id =
?";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, oddId);
```

```
            statement.setInt(2, gameId);
            statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public List<Team> getGameTeams() {
        String sql = "SELECT team_id, name, (select count(*) from odd_game_teams where
game_team_id=team_id)  FROM game_teams order by name;";
        return getSimpleTeams(sql);
    }
    public Team getTeamById(int id){
        String sql = "select team_id, name, 2 from game_teams where team_id = ?";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, id);
            ResultSet resultSet = statement.executeQuery();
            return parseTeams(resultSet).get(0);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    public List<Mistake> teamMistakes(double diff, int team, int league){
        String sql = "select TO_CHAR(g.date, 'Month') m,
sum(get_mistake_for_game(g.game_id, g.score_one, g.score_two, ?)), count(g.game_id),
EXTRACT(month from g.date) mt, EXTRACT(year from g.date) y from games g where (team_one =
? or team_two = ?) and exists (select odd_id from get_odd_for_game(g.game_id)) and
league_id = ? group by m, mt, y order by y, mt;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setBigDecimal(1, BigDecimal.valueOf(diff));
            statement.setInt(2, team);
            statement.setInt(3, team);
            statement.setInt(4, league);
            ResultSet resultSet = statement.executeQuery();
            return pareMistakes(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    public List<Team> getOddTeams() {
        String sql = "SELECT  team_id, name, (select count(*) from odd_game_teams where
odd_team_id=team_id) FROM odd_teams order by name;";
        return getSimpleTeams(sql);
    }
    public List<Team> getResolvedTeams(int league) {
```

```
        String sql = "select distinct gt.team_id, gt.name, 2 from games g join game_teams
gt on (g.team_one=gt.team_id or g.team_two=gt.team_id) where g.league_id=? and (select
odd_id from get_odd_for_game(g.game_id)) is not null order by gt.name;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, league);
            ResultSet resultSet = statement.executeQuery();
            return parseTeams(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public HashMap<Game, List<Odd>> getGamesForMonth(int teamId, int leagueId, String
month){
        String sql = "select g.game_id, g.date, g.score_one, g.score_two, gt1.name,
gt2.name, ov.team_one, ov.team_two, ov.draft, bc.name, g.is_switched\n" +
                "from (select game_id, date, score_one, score_two, team_one, team_two,
league_id, (f).* from (select gm.*, get_odd_for_game(gm.game_id) f from games gm) a) g
\n" +
                "join game_teams gt1 on gt1.team_id = g.team_one\n" +
                "join game_teams gt2 on gt2.team_id = g.team_two\n" +
                "join odd_values ov on ov.odd_id = g.odd_id\n" +
                "join bett_companies bc on bc.company_id = ov.company_id\n" +
                "where (g.team_one = ? or g.team_two=?) and trim(TO_CHAR(g.date,
'Month'))=? and g.league_id=?;";
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, teamId);
            statement.setInt(2, teamId);
            statement.setString(3, month);
            statement.setInt(4, leagueId);
            ResultSet resultSet = statement.executeQuery();
            return parseOdds(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    private List<Team> getSimpleTeams(String sql) {
        try (Connection connection = ConnectionManager.getSimpleConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            ResultSet resultSet = statement.executeQuery();
            return parseTeams(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
```

```
private List<Team> parseTeams(ResultSet resultSet) throws SQLException {
    List<Team> teams = new ArrayList<>();
    while (resultSet.next()) {
        Team team = new Team();
        team.setTeamId(resultSet.getInt(1));
        team.setName(resultSet.getString(2));
        team.setResolved(resultSet.getInt(3)>=1);
        teams.add(team);
    }
    return teams;
}

private List<TotalTeam> parseTotalTeams(ResultSet resultSet) throws SQLException {
    List<TotalTeam> teams = new ArrayList<TotalTeam>();
    while (resultSet.next()) {
        TotalTeam totalTeam = new TotalTeam();
        totalTeam.setOddName(resultSet.getString(1));
        totalTeam.setGameName(resultSet.getString(2));
        totalTeam.setOddTeamId(resultSet.getInt(3));
        totalTeam.setGameTeamId(resultSet.getInt(4));
        teams.add(totalTeam);
    }
    return teams;
}
private List<Mistake> pareMistakes(ResultSet resultSet) throws SQLException {
    List<Mistake> mistakes = new ArrayList<>();
    while (resultSet.next()) {
        Mistake mistake = new Mistake();
        mistake.setName(resultSet.getString(1));
        mistake.setMistakes(resultSet.getInt(2));
        mistake.setGames(resultSet.getInt(3));
        mistakes.add(mistake);
    }
    return mistakes;
}

private HashMap<Game, List<Odd>> parseOdds(ResultSet resultSet) throws SQLException {
    HashMap<Game, List<Odd>> result = new HashMap<>();
    while (resultSet.next()) {
       Game game = new Game();
       game.setGame_id(resultSet.getInt(1));
       game.setDate(resultSet.getDate(2));
       game.setScore_one(resultSet.getInt(3));
       game.setScore_two(resultSet.getInt(4));
       game.setTeam_one(resultSet.getString(5));
       game.setTeam_two(resultSet.getString(6));
       Odd odd = new Odd();
       boolean isSwitched = resultSet.getBoolean(11);
       if(!isSwitched) {
           odd.setTeam_one(resultSet.getDouble(7));
           odd.setTeam_two(resultSet.getDouble(8));
```

```
            }
            else {
                odd.setTeam_one(resultSet.getDouble(8));
                odd.setTeam_two(resultSet.getDouble(7));
            }
            odd.setDraft(resultSet.getDouble(9));
            odd.setBett_company(resultSet.getString(10));
            if(result.containsKey(game)) result.get(game).add(odd);
            else {
                List<Odd> odds = new ArrayList<>();
                odds.add(odd);
                result.put(game, odds);
            }
        }
        return result;
    }
}
```