НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____202__р.

Дипломна робота

на здобуття ступеня бакалавра

За освітньо-професійною програмою “Цифрові технології в енергетиці”

зі спеціальності 122 “Комп’ютерні науки”

на тему: “Web-застосунок для організації та моніторингу турнірів за допомогою штучного інтелекту”

Виконав: студент 4 курсу, групи ТР-14

Рижков Дмитро Андрійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент каф. ЦТЕ, Здор Костянтин Андрійович_____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент доцент к.ф.-м.н., Стець Альона Вікторівна_____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Н.контроль асистент каф. ЦТЕ, РУДИК Володимир Іванович_____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
Наталія АУШЕВА

“___”_____202___р.

ЗАВДАННЯ
на дипломну роботу студенту

РИЖКОВУ Дмитру Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Web-застосунок для організації та моніторингу турнірів за допомогою штучного інтелекту

Науковий керівник роботи Здор Костянтин Андрійович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від “_____”____ 202_____ р. № _____

2. Термін подання роботи студентом 09.06.2025р.

3. Вихідні дані до роботи персональний комп’ютер з операційною системи Windows 10, мови програмування C#.

4. Перелік питань які потрібно розробити 1) проаналізувати існуючі варіанти застосунків з організації турнірів; 2) розглянути інструменти для розробки веб-застосунку; 3) розробити архітектуру та програмне забезпечення з використанням ШІ для прогнозу переможця гри; 4) провести тестування розробленого програмного забезпечення і моделі ШІ.

5. Орієнтовний перелік графічного (ілюстративного) матеріалу: ілюстрації архітектури і діаграм, скріншоти користувацького інтерфейсу

6. Дата видачі завдання “19” вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Вибір теми роботи	19.09.2024р.	Виконано
2	Аналіз методів та засобів розв’язання задачі	20.09.2024–10.12.2024р.	Виконано
3	Розробка архітектури та загальної структури системи	11.12.2024–26.02.2025р.	Виконано
4	Розробка окремих підсистем	27.02.2025–02.04.2025р.	Виконано
5	Програмна реалізація системи	03.04.2025–28.04.2025р.	Виконано
6	Оформлення пояснювальної записки	29.04.2025–14.05.2025р.	Виконано
7	Захист програмного забезпечення	14.05.2025–16.05.2025р.	Виконано
8	Передзахист	26.06.2025–28.06.2025р.	Виконано
9	Захист	15.06.2025–20.06.2025р.	Виконано

Студент _____ __ Дмитро РИЖКОВ_
(підпис) (прізвище та ініціали)

Керівник _____ ____Костянтин ЗДОР__
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 53 сторінках, містить 29 ілюстрацій, 1 таблицю, 1 додаток, 18 джерел в переліку посилань.

Метою дослідження є розробка веб-застосунку для організації та моніторингу турнірів із використанням технологій штучного інтелекту. Такий застосунок дозволяє автоматизувати процеси реєстрації учасників, формування сітки змагань, обробки результатів та генерації аналітики на основі зібраних даних.

У роботі застосовано сучасні засоби та методи розробки програмного забезпечення. В якості основної мови програмування використано C#, а для реалізації серверної частини — фреймворк ASP.NET Core, що забезпечує високий рівень продуктивності, масштабованість та підтримку кросплатформенності. Для реалізації інтелектуальних функцій застосовано бібліотеку ML.NET, яка дозволяє вбудовувати моделі машинного навчання безпосередньо в .NET-застосунок. Розробка здійснювалася у сучасному середовищі JetBrains Rider.

Результатом роботи є функціональний веб-застосунок, який дозволяє ефективно управляти турнірами та отримувати прогностичні та аналітичні висновки за допомогою елементів штучного інтелекту.

Ключові слова: веб-застосунок, штучний інтелект, ASP.NET Core, C#, ML.NET.

ABSTRACT

The diploma thesis consists of 53 pages, including 29 illustrations, 1 table, 1 appendix, and a reference list with 18 sources

The purpose of the study is to develop a web application for organizing and monitoring tournaments using artificial intelligence technologies. Such an application allows automating the processes of registering participants, forming a competition grid, processing results, and generating analytics based on the collected data.

Modern software development tools and methods are used in the work. C# was used as the main programming language, and the ASP.NET Core framework was used to implement the server side, which provides a high level of performance, scalability, and cross-platform support. To implement intelligent functions, we used the ML.NET library, which allows embedding machine learning models directly into a .NET application. The development was carried out in the modern JetBrains Rider environment.

The result of the work is a functional web application that allows you to effectively manage tournaments and obtain predictive and analytical conclusions using elements of artificial intelligence.

Keywords: web application, artificial intelligence, ASP.NET Core, C#, ML.NET.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ШІ — Штучний Інтелект

CRUD — Create/Read/Update/Delete

SPA — Single Page Application

HTML — Hypertext Mark Language

CSS — Cascading Style Sheets

UI — User Interface

MVC — Model-View-Controller

ВСТУП

Актуальність роботи полягає в тому, що сучасні спортивні події та заходи вимагають більш просунутих рішень для зручної організації та аналізу інформації. Автоматизація турнірів за допомогою веб-додатку та штучного інтелекту дозволяє спростити управління, підвищити швидкість та точність обробки даних, покращити та спростити взаємодію з користувачами.

Метою роботи є розробка Web-застосунку для організації та моніторингу турнірів за допомогою штучного інтелекту, що є актуальним у контексті сучасного розвитку інформаційних технологій та цифровізації спортивних заходів.

Для досягнення мети сформовано такі завдання.

1. Проаналізувати існуючі варіанти застосунків з організації турнірів.

2. Розглянути інструменти для розробки веб-застосунку.

3. Розробити архітектуру та програмне забезпечення з використанням ШІ для прогнозу переможця гри.

4. Провести тестування розробленого програмного забезпечення і моделі ШІ.

У рамках дипломної роботи було реалізовано веб-застосунок, який дозволяє організовувати та супроводжувати турніри, використовуючи інструменти штучного інтелекту для аналізу результатів та прогнозування. Основна увага була приділена створенню архітектури програмного комплексу, розробці функціональних модулів та інтеграції моделей машинного навчання.

Під час роботи над проектом було здійснено аналіз предметної області, визначено актуальні технології для розробки, обґрунтовано вибір архітектурних рішень, а також протестовано ефективність розробленої системи в умовах реального сценарію.

Інтеграція штучного інтелекту у процес організації спортивних турнірів відкриває нові можливості для аналітики, прогнозування результатів та використання адмін-функціоналу. Запропоноване рішення є універсальним і може бути адаптоване для проведення турнірів у різних видах спорту, що підтверджує його практичну значущість та потенціал для подальшого розвитку.

У першому розділі детально розглядається задача розробки системи для моніторингу і автоматизації турнірів.

Другий розділ присвячений аналізу існуючих систем, порівнянню з поточним проектом і демонстрації переваг.

Третій розділ описує вибір мов програмування та технологій, методи інтеграції штучного інтелекту в проект, обгрунтування вибору алгоритму розрахунків.

Четвертий розділ описує архітектуру застосунку, структуру бази даних, алгоритм та інтеграцію модулів штучного інтелекту. Також діаграми, які описують структуру проекту, його компоненти та інші деталі.

П’ятий розділ надає інструкцію по взаємодії з програмною системою, наводить приклади на базі сценаріїв роботи з додатком різних видів користувачів, ілюструє сторінки та процес роботи з застосунком.

1 РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ТУРНІРІВ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

У даному розділі описується процес розробки системи моніторингу турнірів. Вона включає не тільки мету та саме завдання розробки, основні вимоги до системи і опис потрібного функціоналу, а і аналіз вже існуючих проектів для виявлення недоліків і виправлення їх у поточному проекті.

1.1 Мета та завдання розробки

Метою даного дослідження є створення сучасного web-застосунку, що дозволяє автоматизувати процес організації і моніторингу турнірів, з інтеграцією модуля ШІ для аналізу та обробки статистичних даних. Система має забезпечити зручність для організаторів, учасників і глядачів, підвищити доступність до даних змагань та надати більш глибокий аналіз результатів матчів на основі зібраних даних.

Для досягнення мети поставлені наступні завдання:

— розробити інтуітивно зрозумілий UI для різних ролей — гравців, адміністраторів і анонімних користувачів;

— реалізувати функціонал реєстрації команд і гравців, з можливістю редагування;

— автоматизувати формування турнірної сітки залежно від кількості команд, гравців і формату турніру;

— забезпечити можливість фіксації результатів матчів з подальшим оновленням таблиці результатів;

— реалізувати механізм збору статистичних показників;

— інтегрувати елементи ШІ для аналізу зібраних даних та прогнозування результатів майбутніх матчів;

— забезпечити стабільну роботу системи в браузерах та адаптивність

— інтерфейсу під різні пристрої;

— побудувати систему для автентифікації користувачів з безпечним збереженням даних.

1.2 Основні функціональні вимоги до системи

Для забезпечення повноцінної роботи веб-застосунку, потрібно реалізувати як звичайні для сайту функціональні модулі, так і просунуті, такі як інтеграція штучного інтелекту. З канонічних модулів, які можна виділити це реєстрацію і автентифікацію користувачів.

Кожен користувач повинен мати можливість створити обліковий запис із відповідним рівнем доступу. В сучасних реаліях, без базової валідації і захисту логіну, хешування паролю не обійдеться будь-який сайт по причині великого різновиду кібер-атак.

В контексті теми сайту, функціонал сайту вимагає базових CRUD-операцій для адміністрування системою в залежності від ролей, які мають свій перелік повноважень. З таких можна виділити такі CRUD-операції, як управління турнірами (тільки для адміністраторів), додавання та редагування гравців і команд, фіксація та облік результатів матчів, збір і збереження статистики і т.п.

Важливо зазначити, що для комфортної роботи користувачів з програмою повинен бути адаптивний дизайн, який буде коректно працювати як на десктопах, так і на мобільних пристроях, так як переважна кількість користувачів аналогічних систем в переважній більшості використовує саме мобільні пристрої.

Окремої уваги заслуговує модуль з ШІ, який інтегрований як окремий сервіс в основний проект. Його задача полягає в тому, що на основі накопиченої статистики система використовує алгоритм машинного навчання (логістичну регресію) для прогнозування результатів майбутніх ігор. Результати аналізу можуть бути використані як підказки для гравців і організаторів або як частина розважальної програми.

1.3 Аналіз існуючих систем організації турнірів

В цьому підрозділі будуть описані аналогічні популярні системи, їх переваги і недоліки, порівняння з новим проектом.

Однією з найпоширеніших платформ для організації аматорських волейбольних турнірів в Україні є вебсайт ukv.org.ua — офіційний ресурс, присвячений аматорській волейбольній лізі. Сайт використовується для реєстрації команд, публікації результатів матчів, ведення турнірних таблиць та інформування користувачів про події.

Функціональні можливості ukv.org.ua включають:

— перегляд календаря матчів;

— розміщення новин та оголошень;

— ведення турнірних сіток;

— таблиці з результатами;

— базову реєстрацію учасників.

Серед інших аналогічних рішень варто зазначити:

— tournamentsoftware.com — загального призначення, більше орієнтована на бадмінтон і теніс, але застосовується і для волейболу;

— challenge.com — зручна у створенні турнірних сіток, однак не пристосована для глибокої статистики гравців.

Також в цьому підрозділі буде описаний аналіз конкурентів та виявлення недоліків в порівнянні з іншими проектами. На таблиці 1 ви можете побачити лаконічне порівняння усіх представлених веб-застосунків в форматі таблиці.

Для детального розбору і порівняння виберемо один з представлених сайтів, наприклад — ukv.org.ua. Попри наявність базових функцій, аналіз цього веб-застосунку виявив низку технічних і користувацьких проблем, що обмежують зручність та ефективність використання платформи, такі як застарілий інтерфейс: сайт має незручний і неадаптивний дизайн, який не коректно відображається на мобільних пристроях. Відсутність сучасної фронтенд-архітектури (наприклад, SPA або PWA) ускладнює взаємодію з системою для користувача [17].

Таблиця 1.1 — Порівняння функціоналу системи з іншими проектами

Критерій	ukv.org.ua	tournamentsoftware.com	Challonge.com	Розроблювана система
Адаптивність дизайну	Неадаптивний дизайн	Частково адаптивний	Повна адаптивність	Повна адаптивність
Технології розробки	Застаріла технологія (PHP), монолітна архітектура	ASP.NET, сучасна база даних	Frontend на JavaScript, REST API	ASP.NET Core, Entity Framework Core
Підтримка мобільних пристроїв	Мінімальна	Часткова	Повна	Повна
AI	Відсутній	Відсутній	Відсутній	Прогнозування та статистичний аналіз за допомогою ML.NET
Реєстрація команд і гравців	Реалізована, але інтерфейс незручний, обмежена логіка	Повна підтримка	Зручна форма реєстрації	Покрокова реєстрація з перевірками
Зручність навігації	Складна, нелогічна структура сторінок	Середній рівень зручності	Висока зручність	Оптимізована структура

Кінець таблиці 1.1

Актуальність даних та оновлень	Дані часто не оновлюються, дублюються, присутні помилки у відображенні	Регулярні оновлення	Підтримка оновлень	Актуалізація в реальному часі
Якість структури структура	Денормалізована структура	Збалансована структура	Спрощена, не завжди достатня	Повністю нормалізована, продумана архітектура

Другою причиною являються застарілі технології: сайт реалізований з використанням PHP, імовірно ще на версіях 5.x. Це створює потенційні ризики для безпеки, а також ускладнює маштабування та супровід системи. До цього всього, проблеми з архітектурою БД у ukv.org.ua призводять до денормалізація даних — наприклад, при редагуванні даних гравця зміни можуть не відображатися у вже зіграних матчах. Це вказує на відсутність централізованих зв’язків та правильного проектування таблиць. Як наслідок, виникають баги у відображені статистики, а іноді — втрата даних. складна навігація: структура сайту не інтуітивна, відсутнє контекстне меню, складно знайти потрібну інформацію (наприклад, розклад, статистику гравця або контактні дані організаторів). Це дуже сильно і негативно впливає на користувацький досвід.

З переваг розробки власної системи можна виділити адаптивність до мобільних пристроїв, актуальні фреймворки та архітектуру, поділення логіки та структур даних, інтеграція ШІ, більш зручний інтерфейс з більш зручною навігацією та оптимізована база даних з належною оптимізацією.

2 ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ WEB-ЗАСТОСУНКУ

Наразі, для розробки веб додатків можна використовувати безліч технологій і фреймворків на майже будь-якій високорівневій мові програмування, такі як фреймворки для швидкого і зручного написання веб застосунків, так і для тренування і використання штучного інтелекту. Мій вибір зупинився на мові C#(.NET), як на кроссплатформеній і об’єктно-орієнтовній мові програмування.

Для тренування моделі була використана бібліотека ML.NET, яка добре інтегрується з мовою C# і фреймворками для цієї мови для написання веб-застосунків. Для управління версіями коду використовується Git, а репозиторій проекту розміщується на GitHub. Для розробки використовується середовище Rider JetBrains.

2.1 Вибір мов програмування, фреймворків, СУБД та інструментів

Для реалізації веб-додатку ми обрали актуальний і широко використовуваний стек технологій, який може забезпечити гнучкість, масштабованість та простоту підтримки проекту.

В якості основної мови програмування для розробки серверної частини даної системи було обрано C# — сучасну, об’єктно-орієнтовану мову програмування, розроблену компанією Microsoft [9]. Ця мова є однією з ключових складових платформи .NET, яка забезпечує потужне та гнучке середовище для створення широкого спектра застосунків — від простих веб-сайтів до складних і великих проектів.

C# користується значною популярністю серед розробників в свою чергу завдяки своїй зрозумілій синтаксичній структурі, сильній підтримці з боку

спільноти в плані великою кількості навчальних матеріалів та постійному розвитку і оновленню платформи та самої мови [11]. З моменту свого створення C# пройшла довгий шлях удосконалення: до мови було додано численні сучасні конструкції, зокрема асинхронне програмування (async/await), мову запитів LINQ, типи записів (records), власні типи-значення (structs), шаблони (pattern matching) та багато інших можливостей, що значно підвищують продуктивність розробника.

Однією з головних переваг C# є строгість типізації, яка дозволяє виявляти помилки ще на етапі компіляції, зменшуючи ризик виникнення непередбачуваних ситуацій під час виконання програми. Крім того, підтримка об’єктно-орієнтовних парадигм таких як інкапсуляція, успадкування та поліморфізм дає змогу створювати гнучку і модульну архітектуру, яку легко підтримувати та масштабувати в майбутньому [8].

Завдяки інтеграції з .NET, C# має доступ до величезної кількості бібліотек та фреймворків, які значно спрощують реалізацію різноманітних функцій: робота з базами даних (Entity Framework, Dapper), веб-розробка (ASP.NET Core), аутентифікація та авторизація (Identity, OAuth, JWT), логування, кешування, серіалізація, тестування, тощо . Це дозволяє розробникам зосередитися саме на вирішенні бізнес-завдань, не витрачаючи зайвий час на реалізацію типових технічних аспектів.

Окремо варто відзначити ASP.NET — це потужний веб-фреймворк, що є частиною екосистеми .NET і забезпечує високу продуктивність, кросплатформенність, легкість налаштування та підтримку сучасних веб-стандартів [6]. Завдяки йому C# є гарним вибором для створення як RESTful API, які забезпечують взаємодію між клієнтською частиною та сервером, так і нативного ASP.NET Core MVC, який використовує Razor Views для відображення контенту у багатьох сучасних застосунках [18].

Таким чином, вибір C# як основної мови для розробки серверної частини системи був обґрунтований її надійністю, сучасністю, розширюваністю, продуктивністю та широкими можливостями, які дозволяють створювати ефективні, масштабовані та зручні у підтримці програмні рішення.

ASP.NET Core — це потужний та сучасний фреймворк, розроблений

компанією Microsoft, призначений для створення веб-додатків, RESTful API, а також мікросервісної архітектури [5]. Він є логічним продовженням класичного ASP.NET, але побудований з нуля з урахуванням сучасних вимог до програмного забезпечення: легковаговість, модульність, розширюваність, кросплатформність та висока продуктивність.

Однією з ключових переваг ASP.NET Core є його кросплатформність: додатки, створені за допомогою цього фреймворку, можуть запускатися на Windows, Linux та macOS [10]. Це дозволяє розробникам розгортати застосунки у будь-якому середовищі — локально, у хмарі (наприклад, Microsoft Azure, AWS, Google Cloud), на віртуальних машинах або у контейнерах Docker. Така гнучкість забезпечує широкий спектр можливостей для хостингу та розгортання в залежності від технічних або бізнес-вимог.

ASP.NET Core розроблений з урахуванням принципів високої продуктивності. Він займає одне з провідних місць у тестах на швидкодію веб-фреймворків (TechEmpower Benchmarks), що є особливо важливим для високонавантажених сервісів. За рахунок покращеної обробки HTTP-запитів, ефективної роботи з пам’яттю та оптимізованої архітектури, фреймворк здатен обробляти тисячі одночасних підключень. В фреймворк вбудовано підтримує ін’єкцію залежностей (Dependency Injection), що є основним патерном для побудови гнучких і тестованих застосунків. За допомогою DI розробники можуть зручно організовувати залежності між компонентами, сприяючи зменшенню зв’язності між модулями, підвищуючи зручність модульного тестування і гнучкість архітектури [15]. Окрім того, ASP.NET Core має вбудовані засоби безпеки (автентифікація, авторизація, захист від XSS/CSRF), гнучку систему маршрутизації, підтримку middleware-компонентів, детальний логування та підтримку масштабування, що робить його ідеальним рішенням як для невеликих веб-сайтів, так і для великих розподілених систем корпоративного рівня [6].

Для розробки дизайну, були використані такі канонічні інструменти як HTML(HyperText Markup Language) та CSS (Cascading Style Sheets). Це основні помічники для створення сторінок для веб-додатків і забезпечують відображення контенту у браузері.

HTML використовується для структури веб-сторінки, поділяючі різні блоки сайту на теги для розділення контенту, виставлення ієрархії елементів і пов’язуючи їх між собою.

CSS, в свою чергу для оформлення веб-сторінок, представляючи користувачу можливість виставляти значення кольору, розмірів і іншим аспектам візуального представлення. Мова стилів дозволяє використовувати класи та ідентифікатори для точного визначення стилів, а також оперувати з стилями груп елементів.

Bootstrap — це фреймворк для розробки веб-інтерфейсів, який надає готові компоненти, шаблони і стилі для швидкого побудування веб-сторінок, дозволяючи легко розташовувати елементи на сторінці. Також, Bootstrap має підтримку готових JavaScript компонентів, які дозволяють однією строчкою коду додавати такі елементи як навігаційні панелі, каруселі, модальні вікна тощо.

Razor Views — це частина технології ASP.NET Core MVC, яка дозволяє динамічно створювати HTML-сторінки за допомогою серверного коду на C# [18]. Razor — це синтаксис, що поєднує HTML і C# у в одному файлі з розширенням .cshtml.

В поточному проекті використовувався гібридний підхід, тобто за основу були взяті інструменти Bootstrap і дошліфовані кастомними стилями CSS, структуруючи усі елементи за допомогою HTML, додаючи елементи які ми отримуємо з контролерів у виді Razor синтаксису, виводячи дані на сторінку.

В якості системи управління базами даних використовується PostgreSQL. Вона використовується для зберігання, обробки та управління даними в багатьох типах програм — від простих сайтів до складних аналітичних систем. В поточному проекті була вибрана саме ця СУБД по причині відкритого програмного забезпечення — безкоштовне для використання, підтримує складні типи даних, реляційні моделі і багато чого іншого.

Взаємодія з базою даних реалізована через ORM-бібліотеку Entity Framework Core, яка дозволяє працювати з базами даниз у вигляді об’єктів і забезпечує використання підходу Code-First [12].

2.2 Опис використаної бібліотеки штучного інтелекту

Для написання проекту використовується ML.NET — бібліотеку машинного навчання, розроблену Microsoft для платформи .NET [1]. Вона дозволяє створювати моделі без необхідності використання сторонніх мов або середовищ (наприклад, Python).

ML.NET використовується для:

— автоматизованого збору та аналізу статистики гравців під час турнірів;

— побудови моделей оцінки ефективності на основі історії їхньої гри (наприклад, кількість перемог, ефективність гравців під час ігор і тому подібне);

— прогнозування результатів ігор на основі наявних даних.

Для реалізації було створено конвеєр обробки даних за допомогою інструментів ML.NET, що включають етапи завантаження даних, нормалізації даних і навчання моделі через UI. ML.NET добре інтегрується з ASP.NET Core, що дозволило включити обробку даних в проект без ризику втрати продуктивності та складності в обслуговуванні коду [3].

Також в цьому підрозділі буде описаний алгоритм машинного навчання, який був обраний для навчання моделі. SdcaLogisticRegression — це алгоритм машинного навчання, який поєднує логістичну регресію для задачі двокласової класифікації з методом оптимізації SDCA (Stochastic Dual Coordinate Ascent).

В ML.NET метод BinaryClassification.Trainers.SdcaLogisticRegression — це алгоритм двокласової класифікації, який базується на логістичній регресії, що навчається за допомогою SDCA (стохастичний метод підйому по подвійним координатам) [2].

Метод спрямований на знаходження ймовірності, з якою певний об’єкт належить до одного з двох класів (0 або 1), на основі ознак. Це досягається шляхом побудови лінійної моделі, яка мінімізує функцію втрат логістичної регресії з урахуванням регуляризації.

Цільова функція логістичної регресії с L2-регуляризацією у вигляді (1):

$$L2 = \min_{\omega} \left[\frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y^i(\omega^T x_i + b)}\right) + \lambda \|w\|^2 \right] \quad (1)$$

де x_i — вектор ознак для i-го прикладу;

$y^i \in \{-1, 1\}$ — правильна мітка класу;

ω — ваговий вектор моделі;

b — зміщення (bias);

λ — коефіцієнт регуляризації (L2).

До переваг методу можна віднести:

— швидкість та масштабованість — підходить для великих наборів даних;

— підтримка розріджених ознак — ефективний у задачах текстової або категоріальної класифікації;

— регуляризація — L2-захист від перенавчання;

— інтерпретованість — ваги ознак дозволяють аналізувати важливість кожної змінної;

— гарантована збіжність — завдяки роботі з опуклими функціями втрат.

Для датасету була взята інформація на базі 50 турнірів, тобто приблизно 600 записів ігор, і 7200 записів статистики кожного гравця.

Описуючи поступовий алгоритм роботи сервісу ШІ, можна виділити такі етапи:

1) завантажити історичні данні з БД через DbContext;

2) попередня обробка (нормалізація, фільтрація);

3) навчання моделі (з допомогою ML.NET);

4) прогнозування результату гри (при виклику методу).

При розробці системи, використовуючи ML.NET для реалізації модулів прогнозування, така як вірогідність перемоги команд в конкретній грі на основі історичних даних [4]. Алгоритми машинного навчання інтегруються в систему як окремі сервіси з подальшою обробкою через внутрішні сервіси. Використовується підхід Separation of Concerns, який дозволяє інтегрувати штучний інтелект незалежно від решти логіки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Внаслідок пошуку і дослідження засобів розробки веб-застосунку, ми виявили оптимальні технології та фреймворки для написання програми. В цьому розділі буде описана програмна реалізація системи, архітектурний стиль, структура бази даних і зв’язки її сутностей, супроводжуючи це діаграмами. Також буде описаний алгоритм роботи щтучного інтелекту та процес інтеграції моделі в веб-застосунок.

3.1 Архітектура системи

При проектуванні архітектури системи важливо враховувати цілу низку критично важливих факторів, серед яких ключовими є:

— масштабованість — можливість системи ефективно функціонувати при зростанні навантаження або кількості користувачів;

— підтримуваність — легкість внесення змін у код, виправлення помилок та розширення функціональності;

— безпека — здатність захистити дані від несанкціонованого доступу та гарантувати цілісність інформації;

— продуктивність — швидкодія системи при взаємодії з великою кількістю користувачів або даних;

— гнучкість та інтеграція — здатність системи легко адаптуватися до нових вимог, підключати нові модулі або інтегруватися з іншими сервісами.

У зв’язку з цим, під час розробки було прийнято рішення використовувати багаторівневу архітектуру (three-tier architecture), яка дозволяє розділити обов’язки між окремими логічними рівнями системи [8]. Такий підхід суттєво підвищує читабельність і підтримуваність коду, дозволяє легко впроваджувати нову функціональність і забезпечує хорошу основу для модульного тестування.

Основні рівні багаторівневої архітектури можна поділити на 3 шари:

презентаційний рівень (Presentation Layer), рівень бізнес-логіки (Business Logic Layer), рівень доступу до даних (Data Access Layer).

Презентаційний рівень відповідає за відображення інформації користувачу та отримання введених даних. У рамках проєкту він реалізований за допомогою Razor Views, які надають потужні засоби шаблонізації HTML у поєднанні з C#. Для покращення взаємодії з користувачем та створення сучасного адаптивного інтерфейсу використано фреймворк Bootstrap, який забезпечує адаптивність та кросбраузерність, що особливо важливо при доступі до застосунку з мобільних пристроїв. Презентаційний рівень ізольований від бізнес-логіки, що дозволяє змінювати інтерфейс користувача без впливу на інші частини системи.

Рівень бізнес-логіки реалізує основні функціональні процеси системи. Він розроблений на основі ASP.NET Core, що дозволяє ефективно реалізовувати RESTful API, обробляти запити та застосовувати механізми авторизації та автентифікації. Бізнес-логіка чітко структурована у вигляді сервісів, кожен з яких відповідає за окрему частину функціоналу — наприклад, управління турнірами, гравцями, результатами матчів тощо.

Сервіси викликаються з контролерів, які обробляють HTTP-запити та відповідають за маршрутизацію. Такий підхід дозволяє легко розширювати логіку, не порушуючи структуру коду. Використання ін’єкції залежностей (Dependency Injection), що вбудована в ASP.NET Core, дозволяє керувати життєвим циклом об’єктів, спрощує тестування та зменшує зв'язність компонентів [7].

Рівень доступу до даних забезпечує взаємодію з базою даних і реалізований за допомогою Entity Framework Core — сучасного ORM (Object-Relational Mapping) фреймворку від Microsoft [13]. EF Core дозволяє працювати з базою даних у вигляді об’єктів, що значно спрощує написання та підтримку запитів [12]. Для додаткового розділення відповідальності застосовано структурний патерн “Repository”, який ізолює логіку доступу до джерела даних [16]. Завдяки цьому:

— бізнес-логіка не залежить від конкретної реалізації збереження даних (наприклад, чи це SQL Server, PostgreSQL або інша СУБД);

— спрощується тестування, оскільки можна легко замінити репозиторій на мок-об’єкт;

— забезпечується гнучка і модульна структура коду, яка дозволяє легко масштабувати систему.

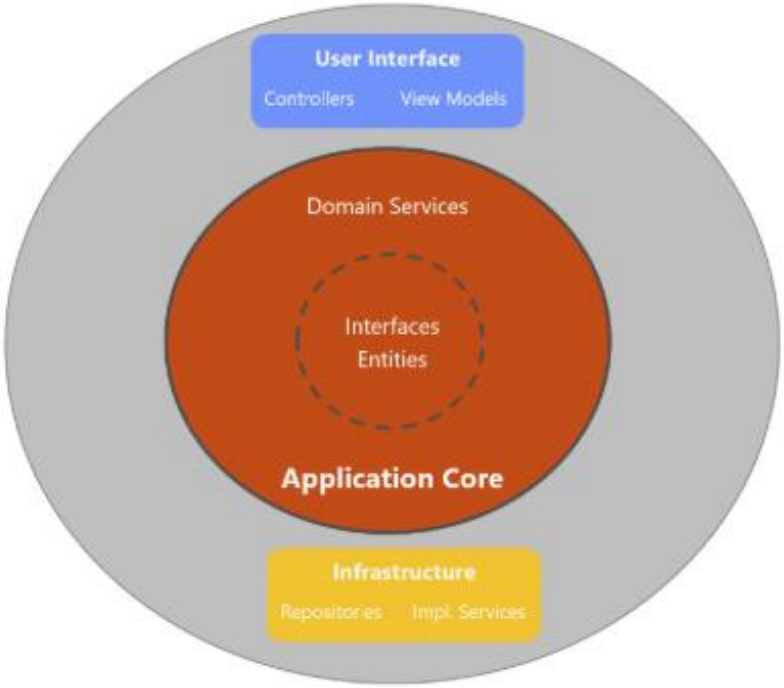


Рисунок 3.1 — Схема тришарової (чистої) архітектури

Усі рівні системи взаємодіють між собою за допомогою чітко визначених інтерфейсів та контрактів, що відповідає принципам чистої архітектури (Clean Architecture)(рисунок 3.1) та SOLID-принципам [14] .

Крім того, така архітектура дозволяє інтегрувати окремі модулі штучного інтелекту, реалізовані за допомогою ML.NET, незалежно від інших частин системи. Це відкриває можливості для розширення функціональності системи за рахунок аналітики, прогнозування результатів змагань, класифікації гравців тощо.

3.2 База даних: структура, таблиці, зв’язки

В цьому підрозділі будуть розглянуті таблиці і їх зв’язки між собою. На рисунку 3.2 ви можете побачити ER-діаграму зв’язків таблиць у цій базі даних.

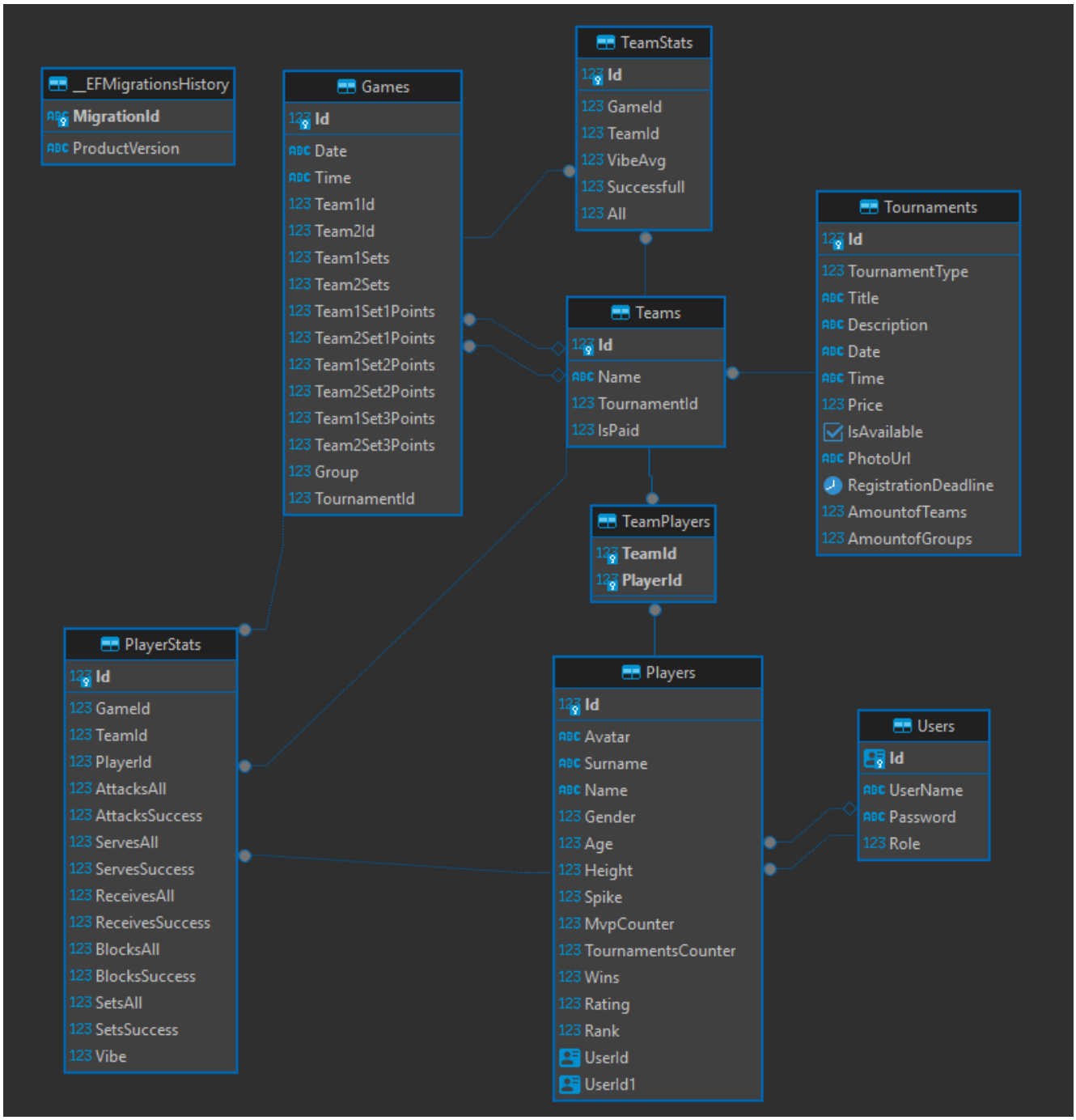


Рисунок 3.2 — ER-діаграма бази данних

У системі використовується реляційна база даних, побудована на PostgreSQL, в яких були створенні такі таблиці:

— Users — сутність, яка зберігає інформацію про користувачів, такі як логін, пароль, унікальний ідентифікатор, роль;

— Players — сутність профілів гравців, які пов’язані з сутностями користувачів. Має в собі статистичні і індивідуальні дані кожного гравця;

— Tournaments — сутність турнірів, яка звітує інформацію про турніри;

— Teams — сутність команд, які приймають участь в турнірах;

— TeamPlayers — суміжна таблиця зв’язку “багато до багатьох” таблиць Team і Player;

— Games — сутність матчів між командами в межах турнірів;

— PlayerStats — статистика гравця в конкретній грі;

— TeamStats — статистика команди в конкретній грі;

— _EFMigrationsHistory — таблиця, створення Entity Framework Core для звітування кожної міграції.

За допомогою цих таблиць в нас є можливість створити надійну і сприятливу для розширення структуру, в якій буде мінімальний ризик втрати даних при денормалізації.

3.3 Діаграми (прецедентів, компонентів, класів)

В цьому підрозділі будуть ілюстровані діаграми, які характеризують роботу системи. Для генерації діаграми класів, був використаний плагін UML Diagrams в Rider JetBrains.

На рисунку 3.3 зображена діаграма прецедентів, яка демонструє можливості користувачів з різними повноваженнями.

Основні актори:

— неавторизований користувач — має доступ до базових функцій системи, таких як перегляд інформації та реєстрація/вхід;

— гравець (авторизований користувач) — може переглядати детальну інформацію, реєструватися на турніри та отримувати доступ до особистого профілю;

— адміністратор — має розширені права доступу до всіх функцій веб-застосунку, включаючи управління модулями системи та роботу з моделю штучного інтелекту.

Основні прецеденти:

— реєстрація/вхід (включає перевірку валідності даних користувачів);

— перегляд інформації (доступна усім користувачам системи);

— реєстрація на турнір (включає додавання до списку учасників турніру);

— доступ до особистого профілю (надає гравцю можливість переглядати особисту статистику);

— доступ до усіх операцій на сайті (лише для адміністратора), розширюється такими прецедентами: CRUD-функції для усіх модулів, перенавчання моделі.

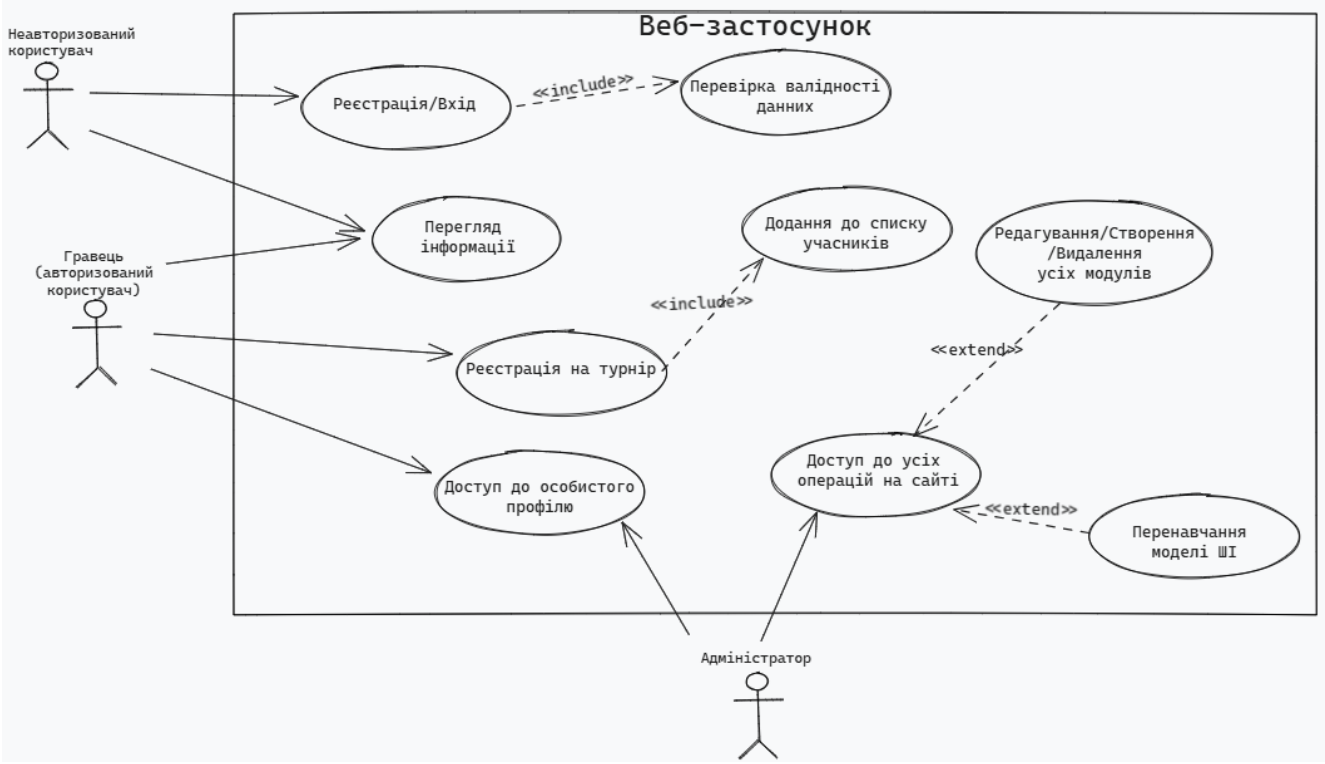


Рисунок 3.3 — UML діаграма прецедентів

Таким чином, кожен актор має свої повноваження, що не дає можливості порушити межі відповідальності інших компонентів системи. Завчасне планування повноважень користувачів, в свою чергу, дозволяє підвищити рівень безпеки, уникнути несанкціонованого доступу до критично важливих функцій системи та полегшує підтримку й масштабування проекту.

Діаграма прецедентів надає зручний засіб візуального аналізу вимог до

системи, який може бути корисним як для розробників, так і для зацікавлених сторін, що не мають технічною освіти, такі як наприклад замовники, яким потрібен простий опис системи. Також вона дозволяє на етапі проектування визначити повноту й послідовність реалізацій функцій системи згідно з очікуваннями користувачів.

На рисунку 3.4 зображена діаграма компонентів, яка демонструє структуру проекту, розбиття на окремі модулі системи.

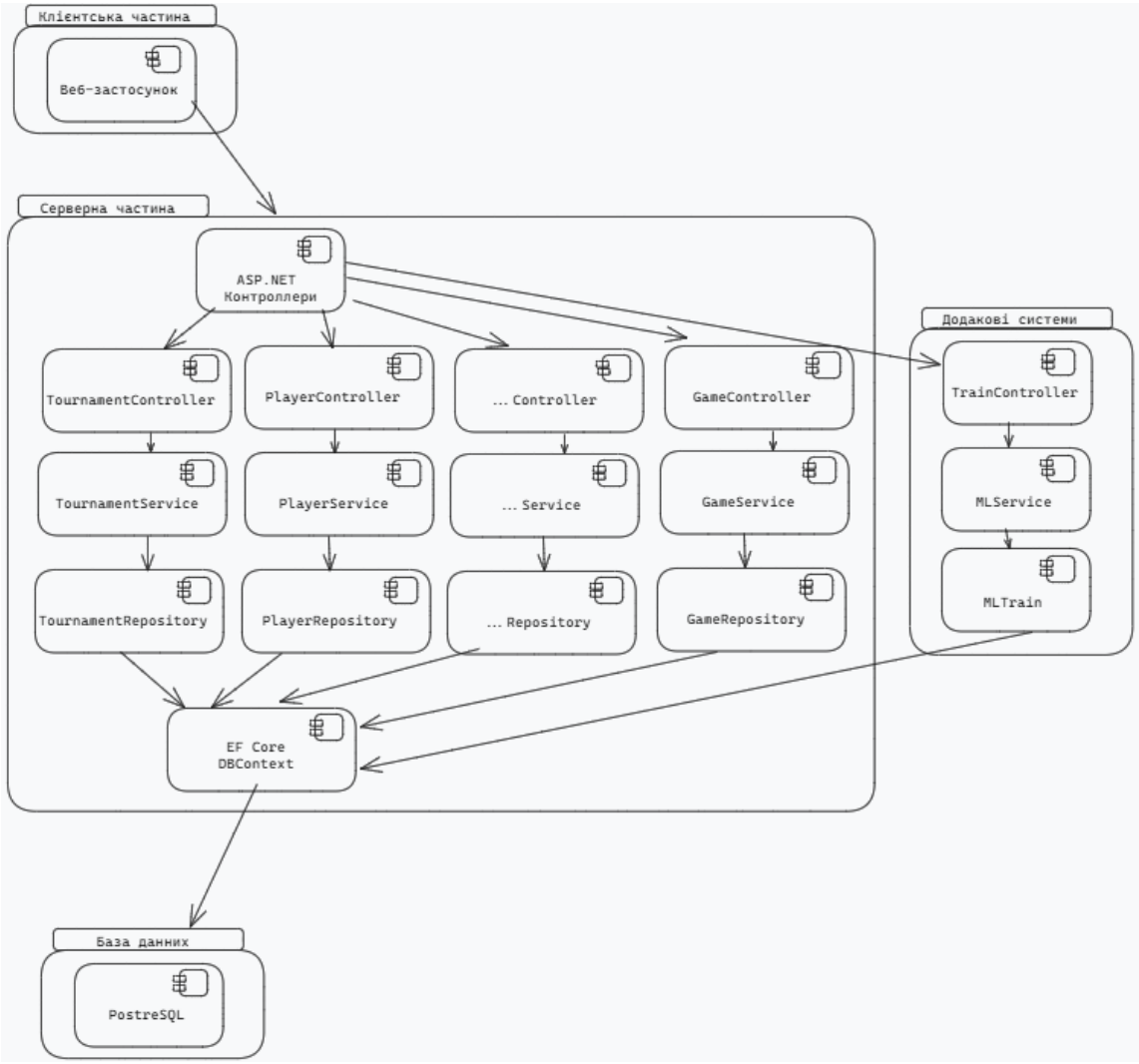


Рисунок 3.4 — UML Components

Архітектура системи побудована з урахуванням принципу Separation of Concerns (розділення відповідальності). Це означає, що кожен компонент системи відповідає лише за окрему частину функціональності, що забезпечує модульність,

зручність підтримки і масштабованість.

Система складається з таких основних частин, як клієнтська частина, серверна частина, база даних, додаткові підсистеми.

Клієнтська частина представлена веб-застосунком, який взаємодіє із сервером через HTTP-запити. Користувач може переглядати, вводити та редагувати інформацію, пов’язану за гравцями, турнірами, іграми тощо.

Серверна частина реалізована з використанням ASP.NET Core та поділена ще на кілька рівнів: контролери (Controllers), сервіси (Services), репозиторії (Repositories) та EF Core DbContext (фрейворк для роботи з базою данних).

Контролери приймають запити клієнта, викликають відповідні сервіси, а також формують і повертають відповіді. У системі передбачено окремі контролери для кожної області: TournamentController, PlayerController, GameController та інші.

Сервіси містять бізнес-логіку застосунку. Вони обробляють вхідні данні, перевіряють на валідність даних, виконують обчислення, пошук за допомогою LINQ або перевірки, і далі взаємодіють з репозиторієм.

Репозиторії відповідають безпосередньо за роботу з базою даних. Вони інкапсулюють CRUD-операції і взаємодіють із контекстом бази даних(DbContext) через ORM Entity Framework Core.

EF Core DbContext — це об’єкт, що забезпечує доступ до бази даних. Через нього здійснюється читання, збереження оновлення у таблицях бази PostgreSQL.

У якості системи управління базою даних використовується PostgreSQL. У ній зберігаються основні сутності та їх інформація.

Система містить окрему підсистему для обробки даних машинного навчання. Вона складається з таких компонентів:TrainController — контролер для керування процесом тренування моделей, MLService — сервіс, що реалізує логіку аналізу та прогнозування, MLTrain — модуль, відповідальний за запуск і контроль навчання моделей. Кожен з них інкапсульований від основної логіки.

У цьому підпункті ми розглянемо діаграми класів системи. Так як на діаграмі всього проекту дуже погано видно елементи, цю діаграму було розбито відповідно до шару проекту. Діаграм буде 4, скільки шарів всього. На рисунку 3.5 можна побачити діаграму класів шару Domain.

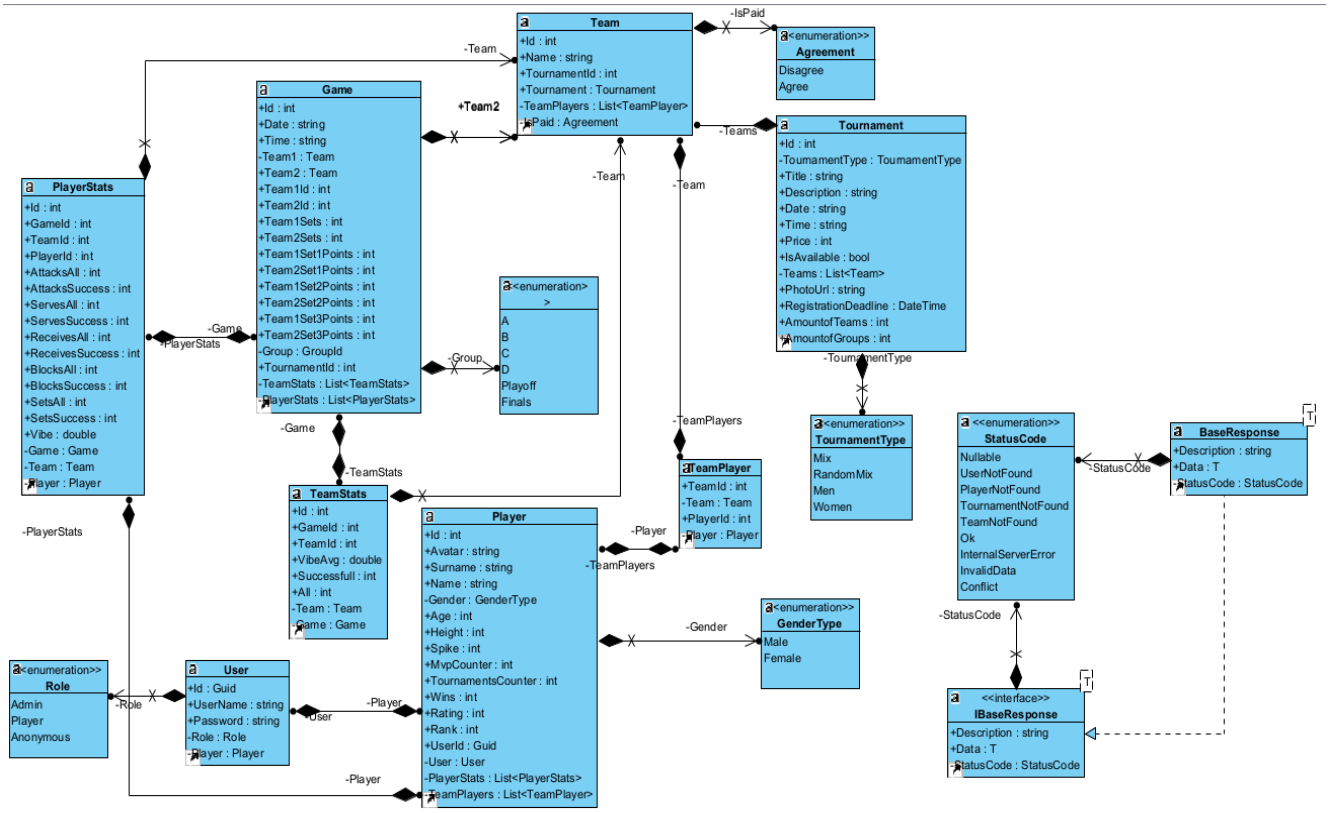


Рисунок 3.5 — UML діаграма класів шару Domain

Це шар, який містить усі бізнес моделі проекту, містить основні інтерфейси для взаємодії шарам між собою, коротше кажучи — базові структури для всього проекту. На рисунку 3.6 ілюструється діаграма класів шару Application.

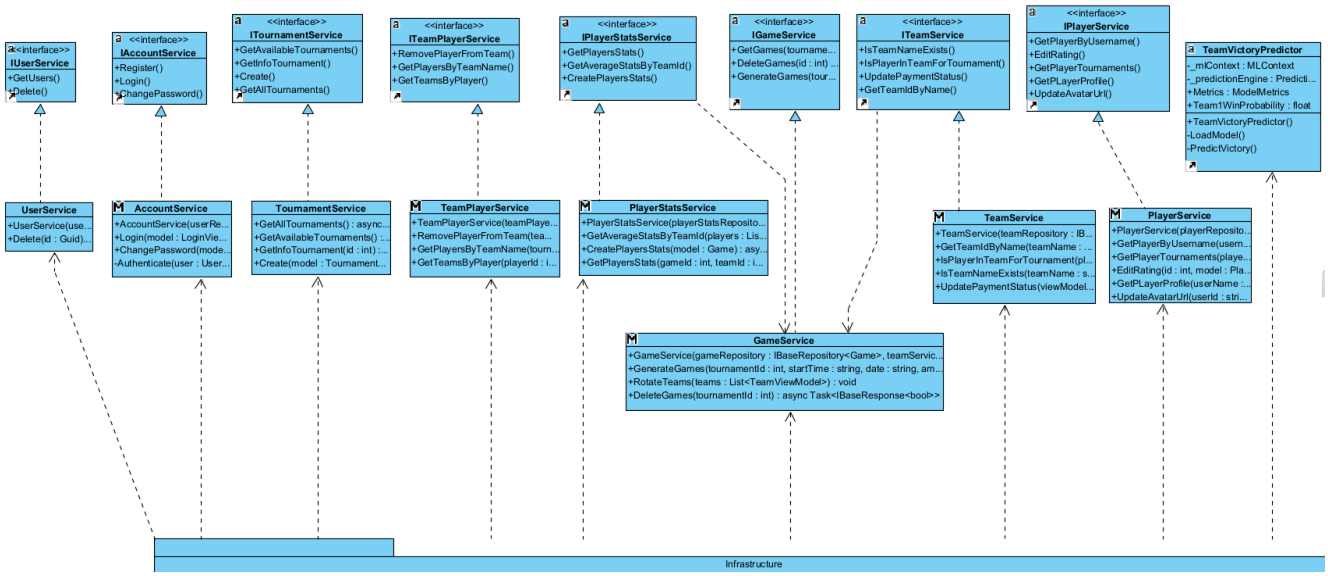


Рисунок 3.6 — UML діаграма класів шару Application

У ньому реалізовані основні правила та алгоритми, які визначають поведінку додатку відповідно до його функціональних вимог. Також, містить методи відображення, редагування та видалення даних.

Для кожного сервісу створюється контракт, за допомогою якого наступний шар буде отримувати доступ до методів. Далі по структурі іде Infrastructure — це шар доступу до бази даних і репозиторіїв (рисунок 3.7).

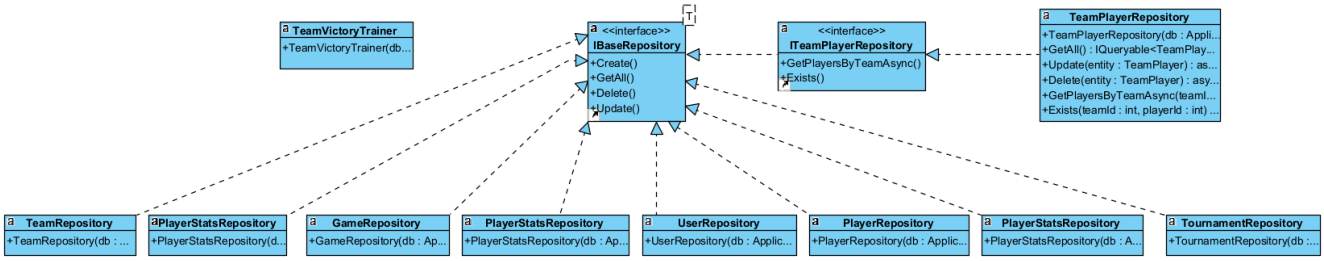


Рисунок 3.7 — UML діаграма класів шару Infrastructure

І останній шар, який залишився це Web — шар представлень, контроллерів і допоміжних інструментів (рисунок 3.8).

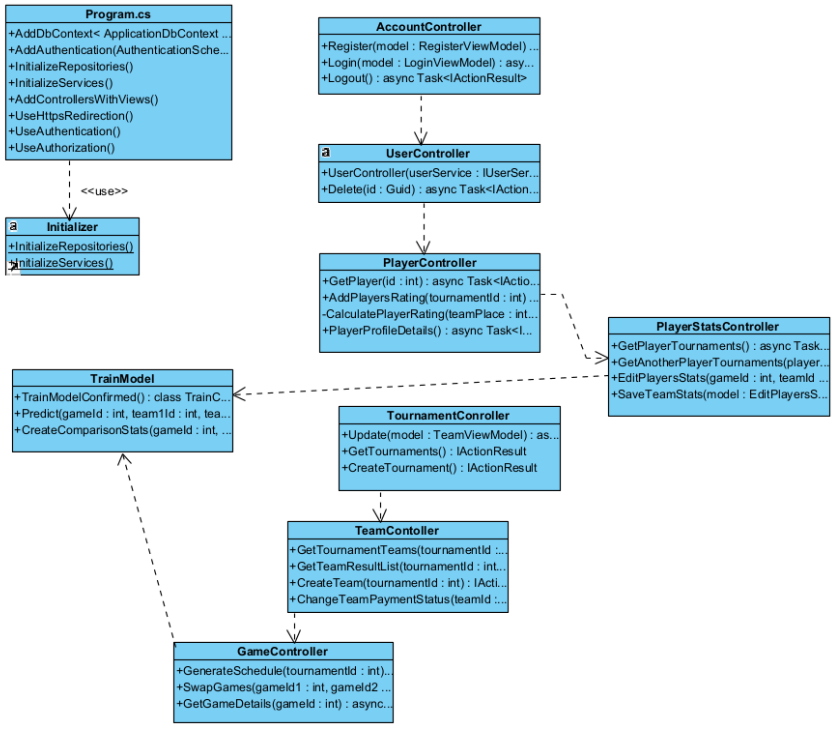


Рисунок 3.8 — UML діаграма класів шару Web

У розробленій системі було застосовано структурний патерн проєктування “Репозиторій” (Repository Pattern), який забезпечує чітке розмежування між логікою доступу до даних і бізнес-логікою застосунку. Завдяки такому підходу стає можливим розширення функціональності без потреби змінювати вже реалізовані компоненти, що позитивно впливає на підтримку, тестування та масштабованість системи. Така реалізація дозволяє ізолювати шар доступу до даних, спрощуючи керування змінами, зокрема у випадках переходу на іншу СУБД чи модифікації структури таблиць.

Центральну роль у цій архітектурі відіграє інтерфейс IBaseRepository, який задає контракт для виконання базових CRUD-операцій (створення, читання, оновлення, видалення). Цей інтерфейс забезпечує єдину точку входу для взаємодії з даними, а спеціалізовані репозиторії наслідують його, реалізуючи специфічну логіку для окремих сутностей. Такий підхід забезпечує повторне використання коду, покращує узгодженість і спрощує внесення змін.

Окремо варто зазначити роль контролерів, які виступають посередниками між клієнтською частиною та серверною логікою. Вони обробляють вхідні HTTP-запити, викликають відповідні методи з репозиторіїв або сервісів, формують відповіді для клієнта, а також ініціюють оновлення даних у базі даних. Таким чином, контролери координують обмін інформацією між користувачем і системою, забезпечуючи коректну взаємодію з інтерфейсом та внутрішніми компонентами програмної архітектури.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА: ВЗАЄМОДІЯ З СИСТЕМОЮ

Цей розділ пояснює, як користувач взаємодіє з веб-застосунком, зокрема: які сторінки доступні, як здійснювати базові дії, та як працювати з основними функціями системи. Розглянуті і проілюстровані основні сценарії використання веб-застосунку різними типами користувачів.

4.1 Приклади сценаріїв роботи

Для демонстрації сценаріїв роботи з веб-застосунком, виділимо основні ролі: незареєстрований користувач; гравець; адміністратор. Незареєстрований користувач має обмежений доступ до функціоналу.

Основні сценарії: перегляд загального списку турнірів, перегляд публічної статистики гравців та реєстрація в системі.

Користувач може переглядати список майбутніх та завершених турнірів, їх опис, дату проведення, але не може брати участь у них. Також може переглядати рейтинги гравців, кількість участей у турнірах, перемог та іншу відкриту статистику без персональних даних. Щоб отримати більше функцій, користувач може створити обліковий запис, після чого стає гравцем із відповідним рівнем доступу.

Гравець — зареєстрований користувач, який може брати участь у турнірах та переглядати власну статистику. Основні сценарії: реєстрація на турнір, перегляд особистої статистики, редагування профілю, оцінка результатів та перегляд прогнозів ШІ. Гравець може обирати доступний турнір зі списку, переглянути умови участі та натиснути кнопку “Зареєструватися”. Система перевіряє, чи відповідає гравець вимогам, та додає його до списку учасників турніру.

Після завершення турніру гравець може переглянути аналітику — наприклад, наскільки точно система передбачила переможців. У профілі гравець може

побачити кількість зіграних турнірів, перемог, особисті дані і статистики. Також гравець може оновити власні контактні дані, фото профілю або змінити пароль.

Адміністратор має повний доступ до керування даними системи. З основних сценаріїв можна виділити: CRUD-операції, редагування інформації про гравця, перегляд та редагування статистики турніру, генерація розкладу та запуск тренування моделі ШІ і використання для прогнозу.

Адміністратор переходить до розділу керування турнірами, натискає кнопку “Створити турнір”, заповнює дані (назва, дата, місце, кількість учасників тощо) та зберігає. Інформація зберігається в базі даних через TournamentController, який оброблює і валідує операцію, TournamentService, який проводить операції на рівні пошуку або обрахунків і TournamentRepository, який коректно кординує і зберігає данні. Аналогічно для редагування і видалення та для інших CRUD-операцій.

Через інтерфейс перегляду гравців адміністратор обирає гравця зі списку, вносить зміни (наприклад, рейтинг, контактну інформацію) та підтверджує зміни. Сервісна логіка оновлює відповідні дані в базі. Адміністратор має доступ до зведених таблиць по кожному турніру (кількість учасників, переможці, результати матчів).

Дані отримуються з бази та можуть аналізуватись також через підсистему машинного навчання; також, він має можливість автоматично генерувати розклад до турнірів, який включає сортування команд в залежності від кількості і кількості груп, розраховує час і має можливість міняти місцями ігри за потреби.

Через окремий інтерфейс адмін може запустити навчання моделі на основі історичних даних про ігри. Цей запит обробляється через TrainController та MLTrain. Також, для кожної гри є функція прогнозу вірогідності перемоги в кожної з команд.

4.2 Інтерфейс користувача: опис сторінок та функціоналу

При вході на веб-ресурс перше що бачить користувач це головну сторінку, на якій присутні навігаційну панель і карусель з актуальними турнірами(рисунок 4.1).

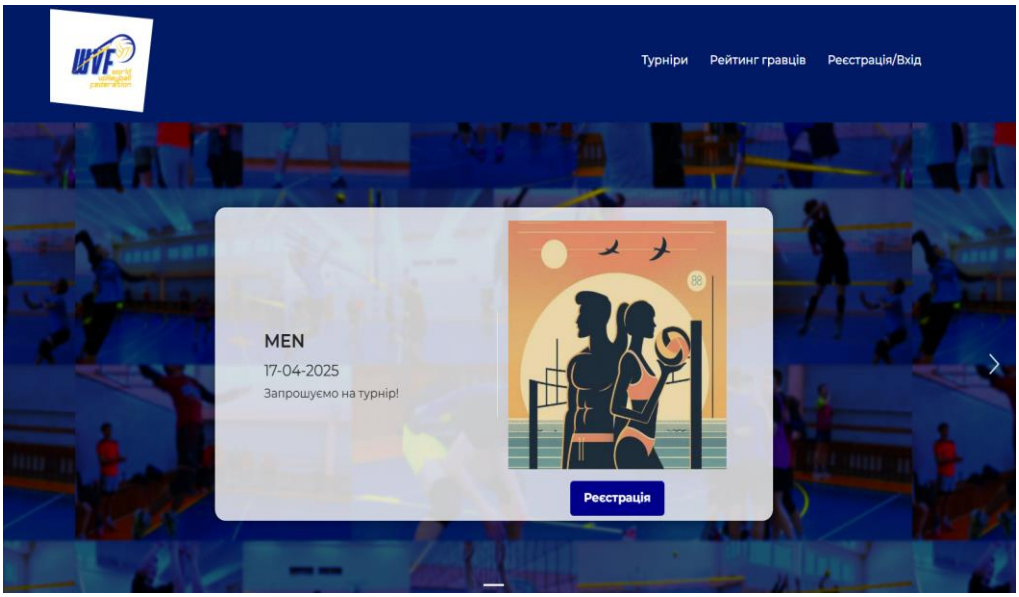


Рисунок 4.1 — Головна сторінка

На головній сторінці відображається карусель з доступними турнірами, нижче, користувач може створити обліковий запис, ввівши необхідні дані у форму реєстрації(рисунок 4.2).

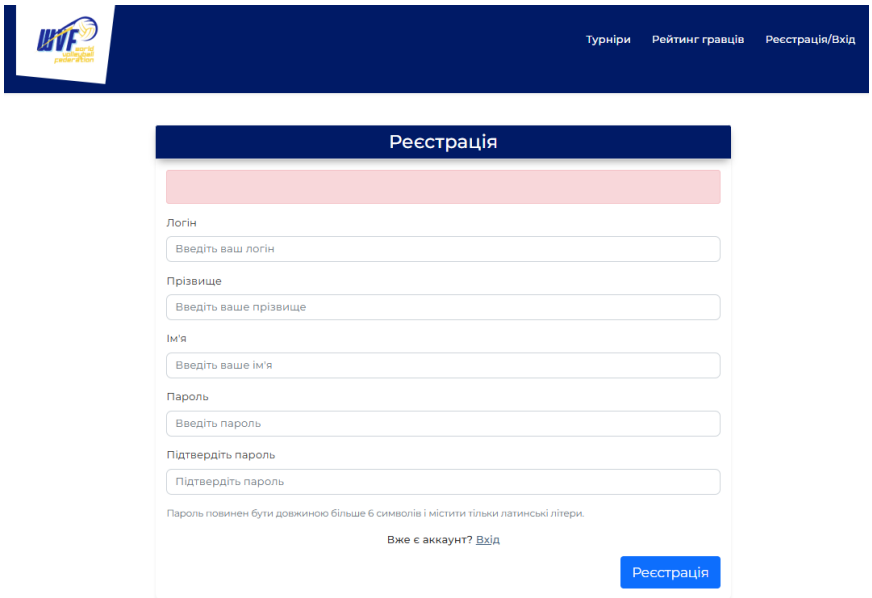


Рисунок 4.2 — Форма реєстрації

Існуючі користувачі входять у систему через форму авторизації(рисунок 4.3). Передбачена перевірка правильності введених даних.

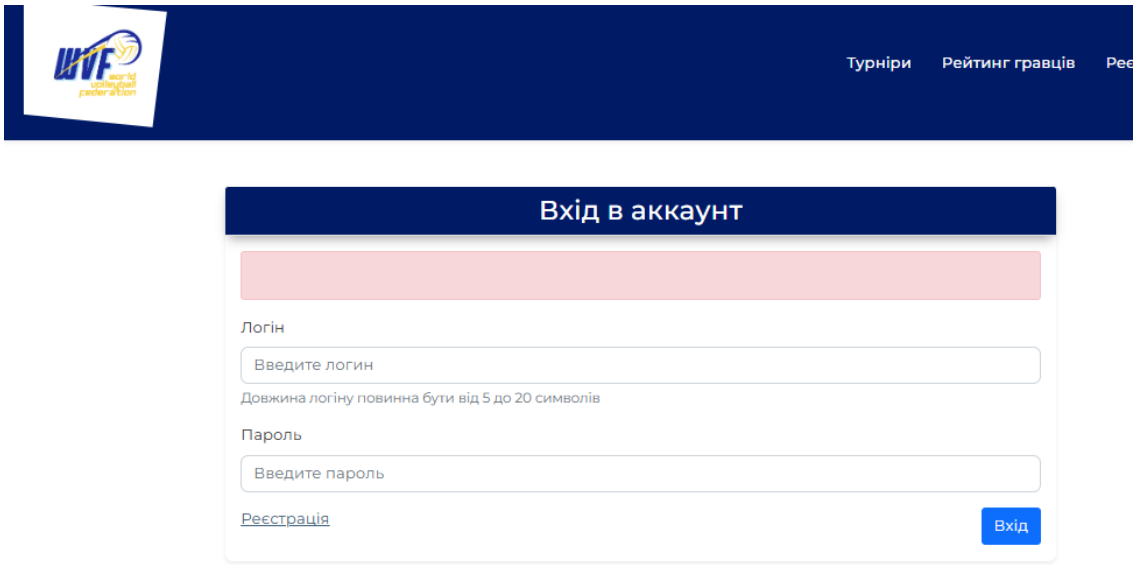


Рисунок 4.3 — Форма входу в аккаунт

Незареєстрований користувач може мати змогу продивитися архів турнірів, подивитися рейтинг гравців або ввійти або створити обліковий запис.

При натисканні на кнопку “Реєстрація” на турнір, якщо незареєстрований користувач буде перекинутий на блок “Зареєструватися”.

На рисунку 4.4 продемонстрована робота валідації користувачів за логіном (не може бути двох користувачів з однаковим логіном).

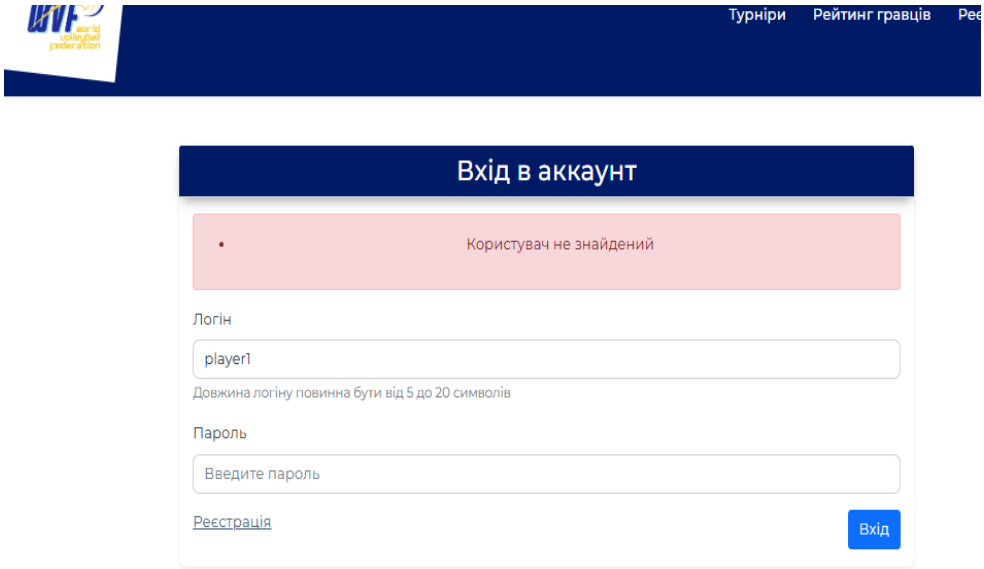


Рисунок 4.4 — Валідація гравця за логіном

На рисунку 4.5 продемонстрована валідація паролю за довжиною і коректністю. Повний функціонал передбачує аутентифікацію користувача.

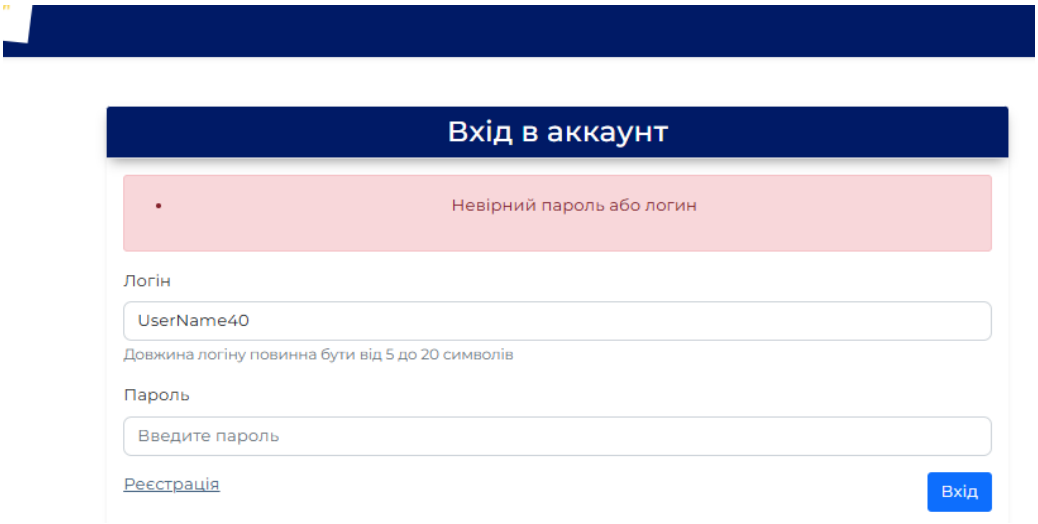


Рисунок 4.5 — Валідація гравця за паролем

Незареєстрований користувач має змогу тільки переглядати інформацію, таку як список турнірів, рейтинг гравців, деталі матчів та статистики. На рисунку 4.6 виведений список турнірів відфільтрований по доступності і даті.

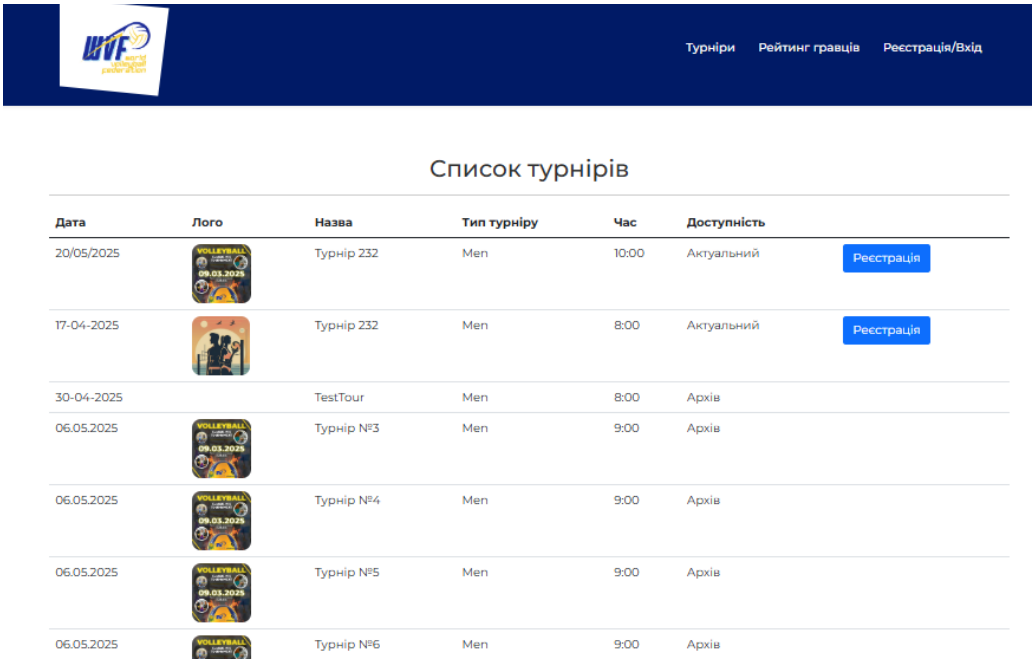


Рисунок 4.6 — Список турнірів

На рисунку 4.7 зображена та сама сторінка, що і на рисунку 4.6, тільки з додатковим функціоналом для адміністраторів.

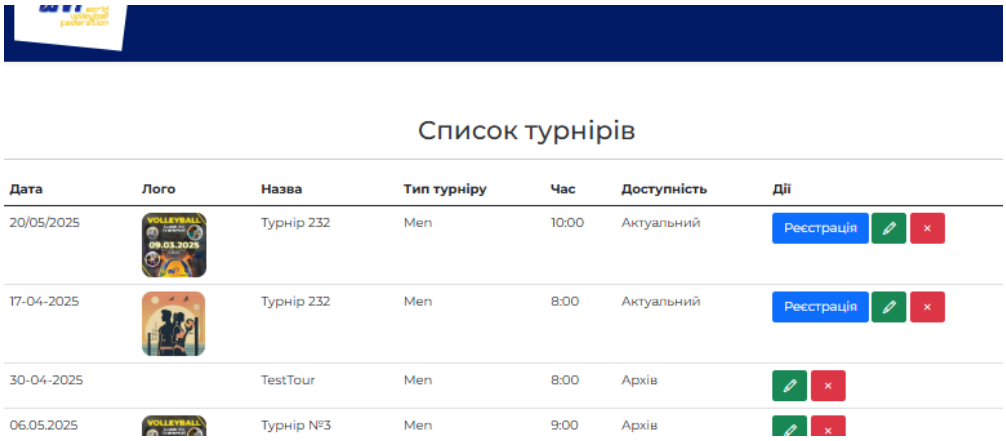


Рисунок 4.7 — Список турнірів для адміністраторів

На рисунку 4.8 зображена сторінка з рейтингом гравців. Після кожного турніру, в залежності від досягнень команди і гравців особисто, нараховуються очки. На головній сторінці виводяться топ-5 гравців кожної статі з цієї рейтингової таблиці.

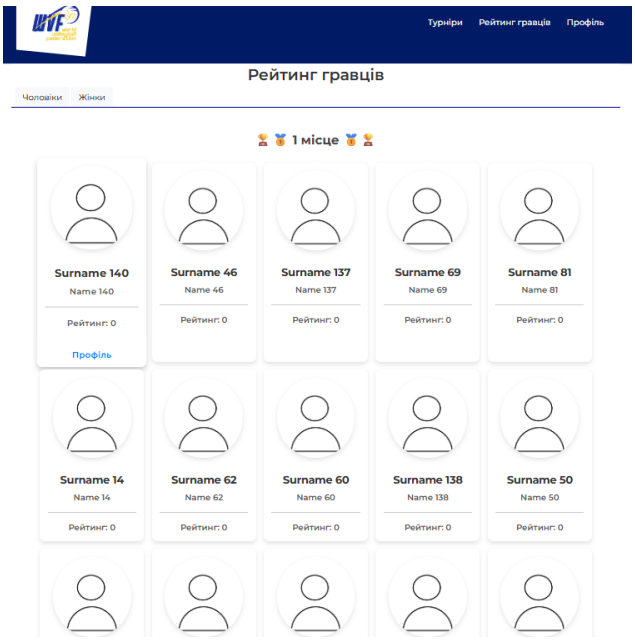


Рисунок 4.8 — Вікно рейтингу гравців

Після реєстрації або входу в аккаунт, профіль відображає персональну інформацію користувача: ім’я, рейтинг, статистику участі у турнірах (кількість, перемоги)(рисунок 4.9). Доступний функціонал редагування профілю, завантаження аватару і перегляду персональної статистики. Кожен гравець має особисте павутиння статистика, яке вираховується на основі його усіх минулих ігор.

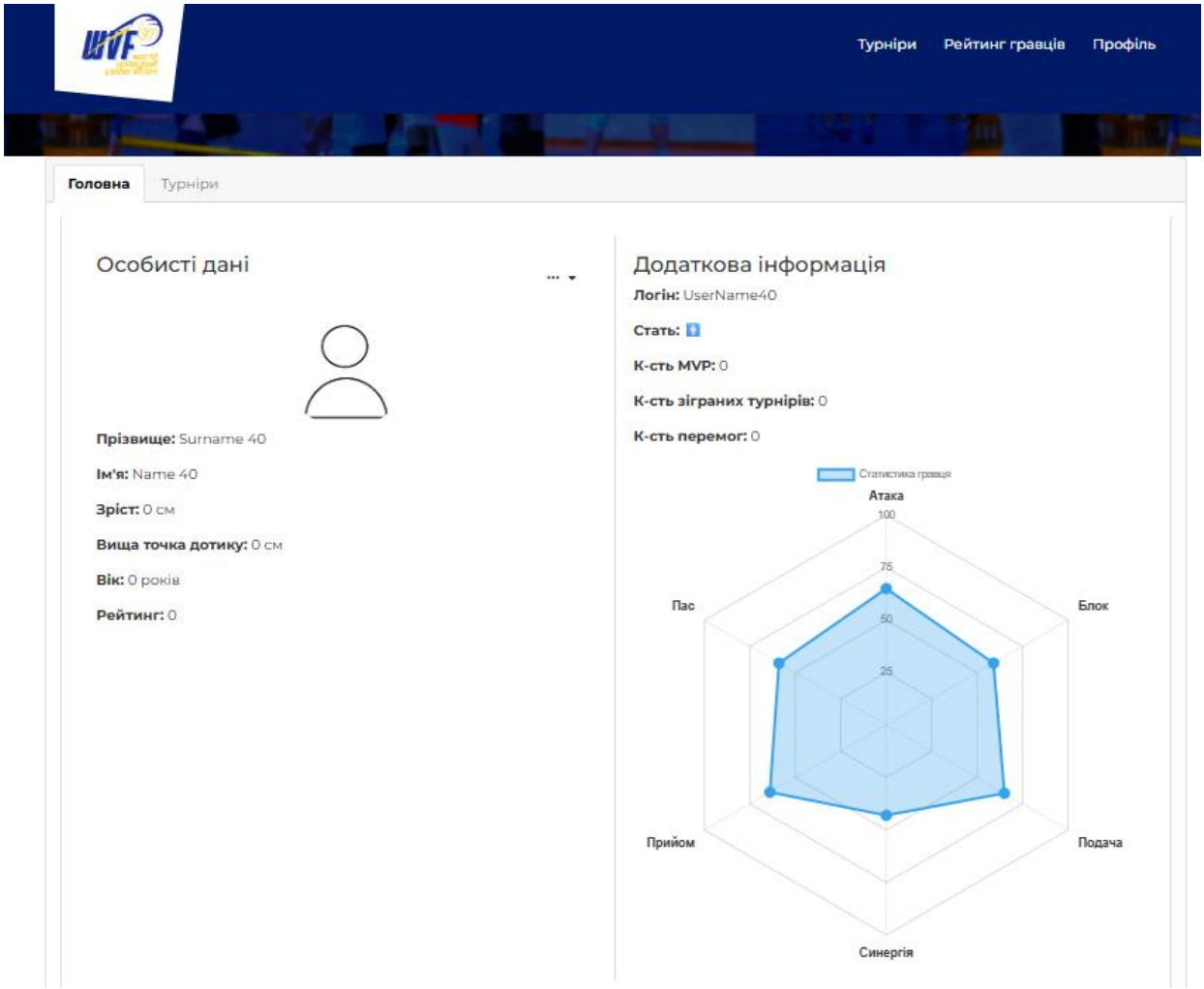


Рисунок 4.9 — Вікно профілю гравця

Також відображено історію турнірів гравця(рисунок 4.11).Реалізована функція редагування профілю (рисунок 4.10).

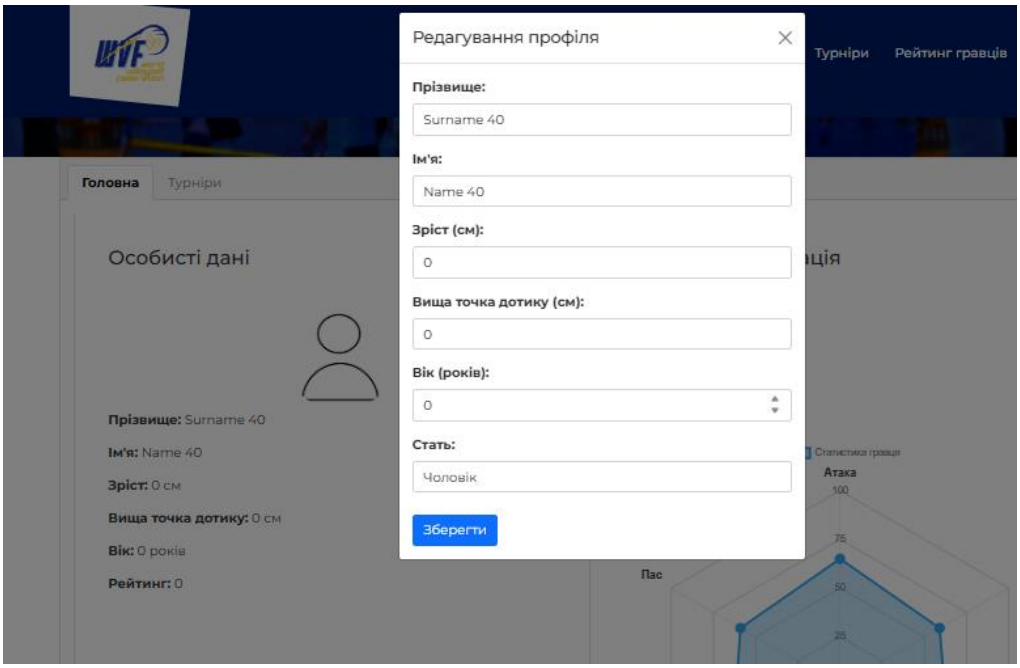


Рисунок 4.10 — Модальне вікно редагування профілю

На рисунку 4.11 в кожного гравця є архів зіграних ним турнірів. На базі цих турнірів вираховується статистика кожного користувача.

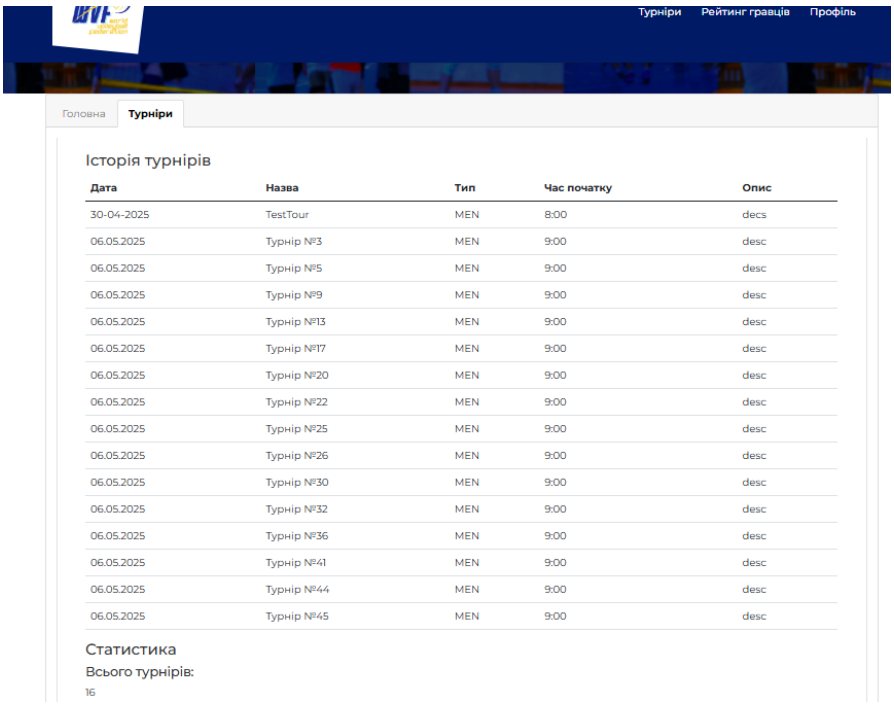


Рисунок 4.11 — Зіграні турніри гравця

Сторінка, присвячена конкретному турніру, надає користувачам повний набір інформації, що стосується його проведення. Зокрема, на ній відображаються такі ключові дані, як дата початку та завершення події, місце проведення (фізичне або онлайн), загальна кількість учасників, а також перелік усіх зареєстрованих гравців, які підтвердили свою участь. Окрім пасивного перегляду, ця сторінка також надає функціональні можливості для взаємодії — зокрема, будь-який користувач із відповідними правами може подати заявку на участь у турнірі через зручну форму реєстрації.

Після завершення турніру ця ж сторінка оновлюється і використовується для публікації фінальних результатів, включно з підсумковими рейтингами, переможцями та статистикою ігор. На рисунку 4.12 продемонстровано інтерфейс адміністрування, що дає організатору змогу редагувати або оновлювати інформацію про турнір у реальному часі.

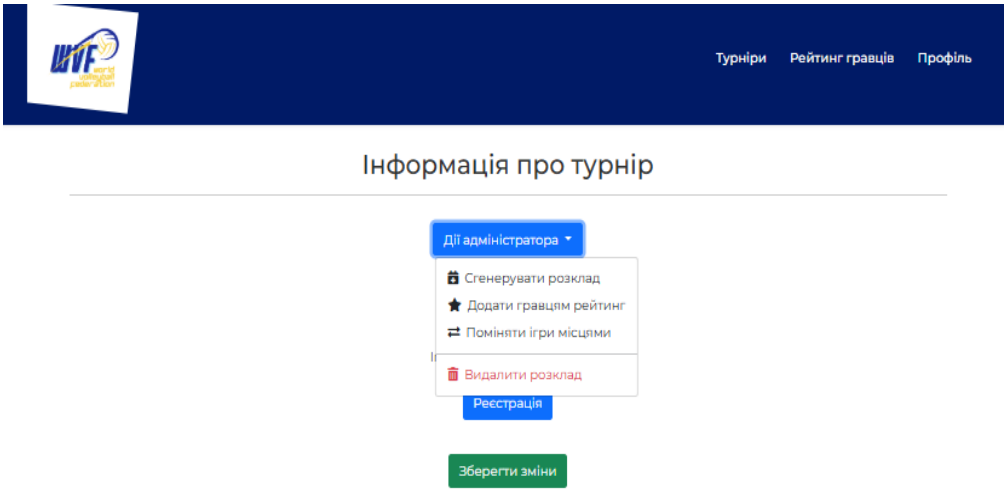
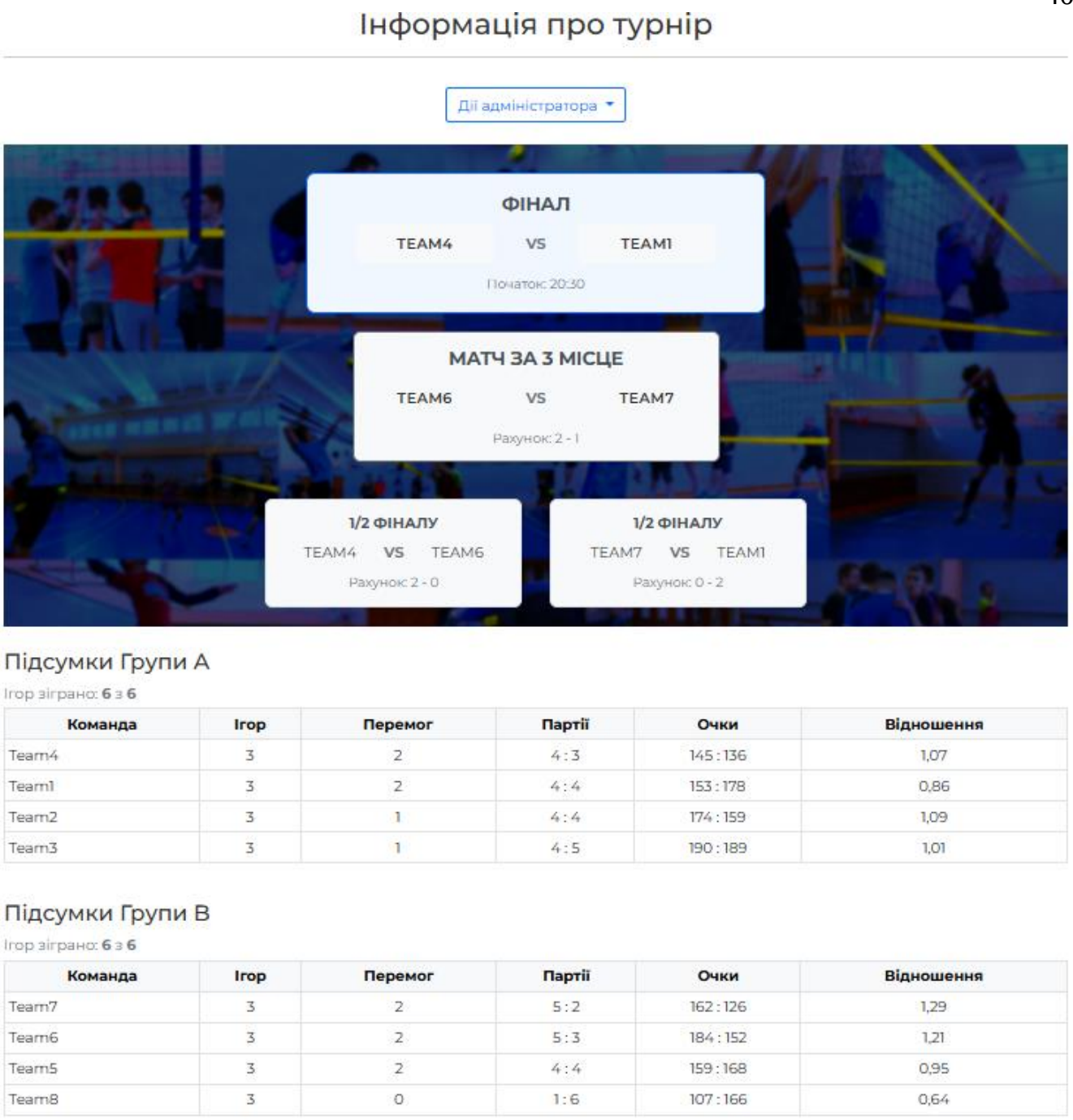


Рисунок 4.12 — Вікно інформації про турнір при відсутності розкладу

У свою чергу, рисунок 4.13 та рисунок 4.14 ілюструють інтерфейс перегляду розкладу матчів, який супроводжується додатковими адміністративними функціями. Ці функції дозволяють вносити зміни до графіку, додавати нові матчі, оновлювати результати, або коригувати дані про гравців, що забезпечує гнучкість і повноцінний контроль над організацією турніру.



Підсумки Групи A

Ігор зіграно: **6** з **6**

Команда	Ігор	Перемог	Партії	Очки	Відношення
Team4	3	2	4 : 3	145 : 136	1,07
Team1	3	2	4 : 4	153 : 178	0,86
Team2	3	1	4 : 4	174 : 159	1,09
Team3	3	1	4 : 5	190 : 189	1,01

Підсумки Групи B

Ігор зіграно: **6** з **6**

Команда	Ігор	Перемог	Партії	Очки	Відношення
Team7	3	2	5 : 2	162 : 126	1,29
Team6	3	2	5 : 3	184 : 152	1,21
Team5	3	2	4 : 4	159 : 168	0,95
Team8	3	0	1 : 6	107 : 166	0,64

Рисунок 4.13 — Сторінка результатів турніру

Адміністратор має можливість міняти ігри місцями для нормалізації розкладу. Виводяться таблиці з детальним досліжженням статистики, вираховуються відношення партій, очок, відношення очок однієї команди до іншої.

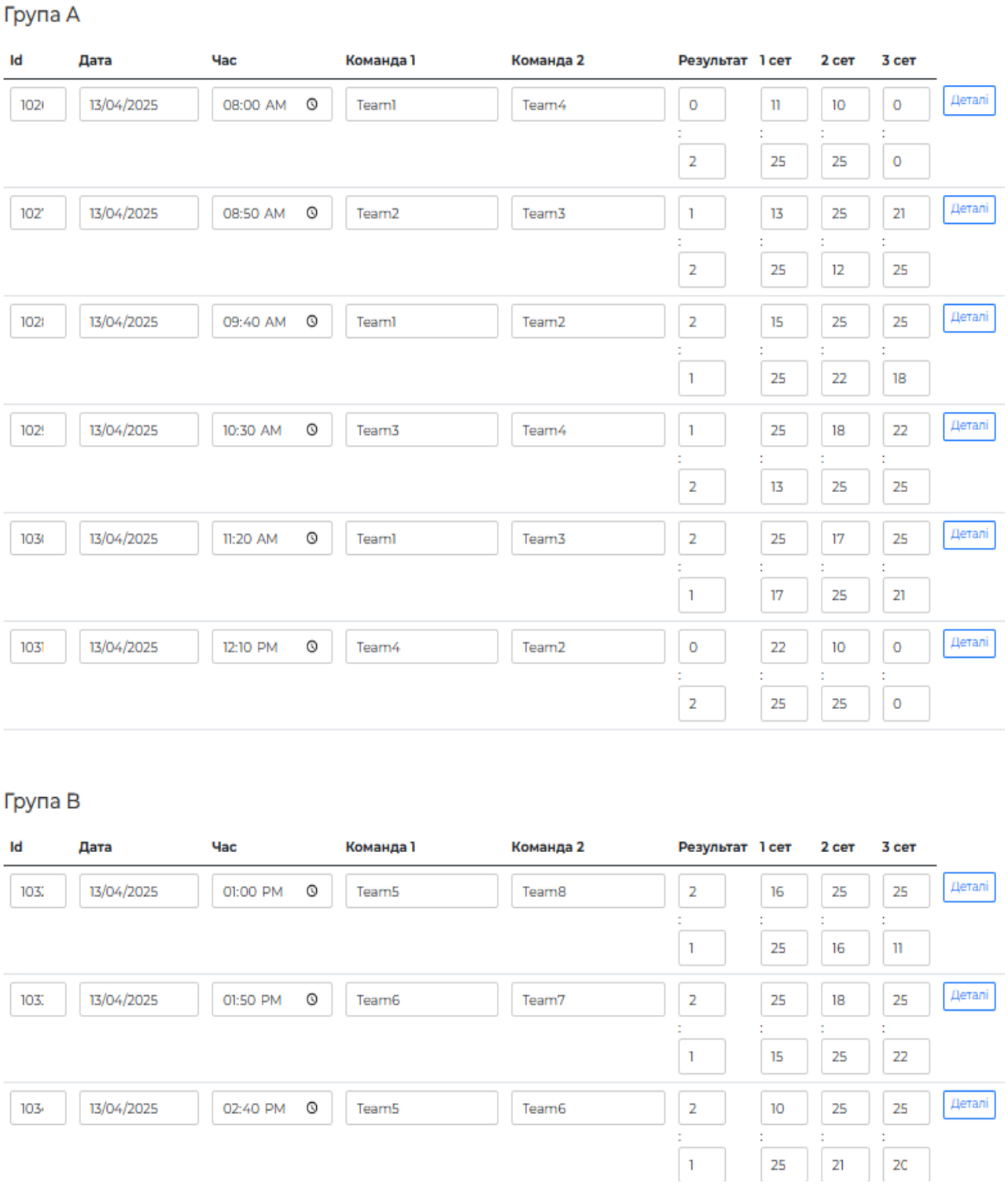


Рисунок 4.14 — Статистичні данні ігор (розширені можливості)

Для гравця стає доступна загальна інформація про турніри, можливість зареєструватися на турнір(рисунок 4.15), а також перехід до особистого профілю. Адміністратору доступна панель керування, що містить посилання на управління турнірами, користувачами та штучним інтелектом.

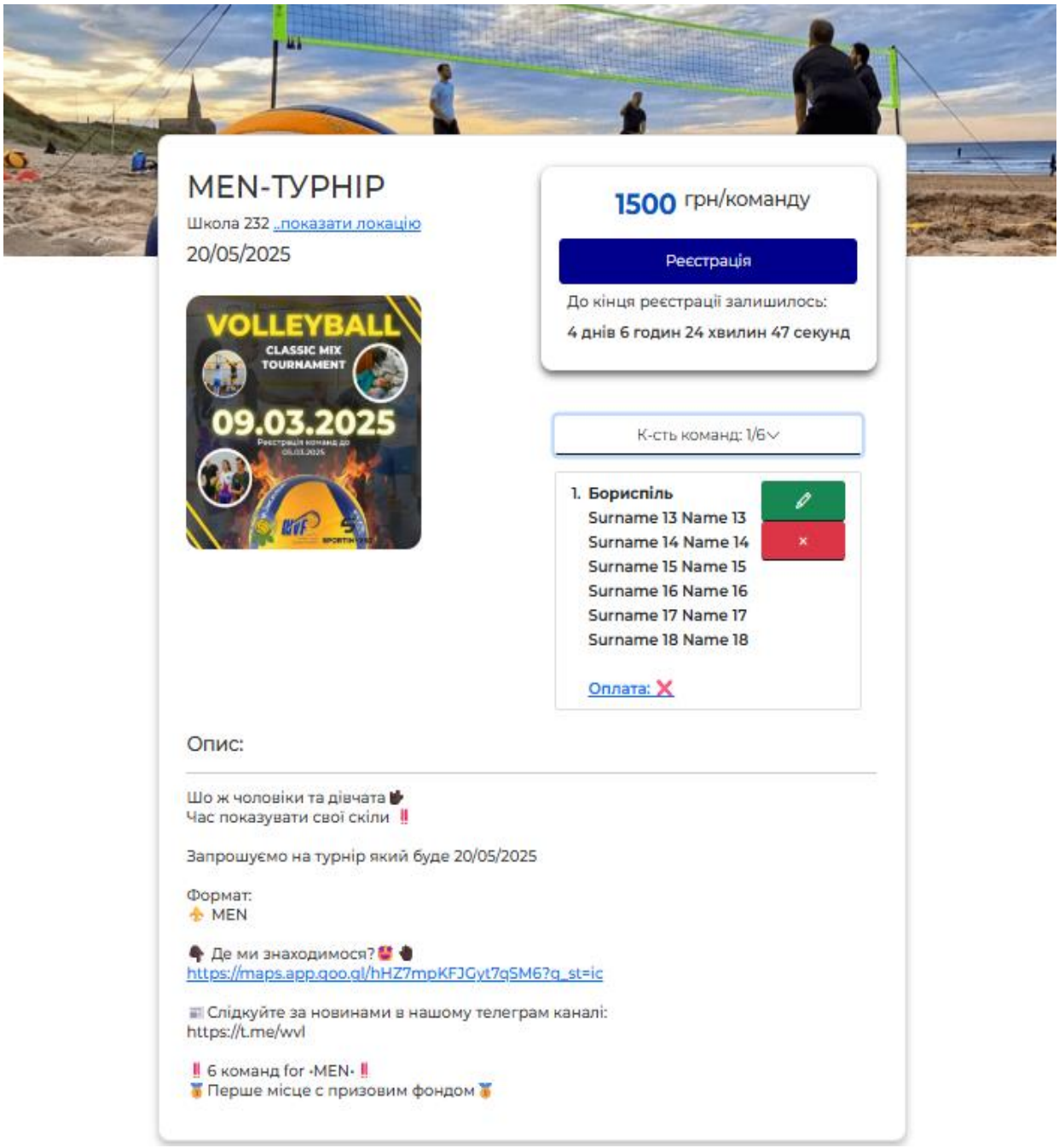


Рисунок 4.15 — Сторінка реєстрації на турнір (з додаковими можливостями для адміністратора)

Відображені усі відомі дані про турнір, списки гравців і команд, які вже зареєструвалися, функціонал зміни статусу оплати і реєстрації на турнір.

Для реєстрації кожної команди, є відповідно сторінка для реєстрації команди(рисунок 4.16). Присутня валідація на повторюванність команд і гравців, на тип турніру і стать гравців відповідно.

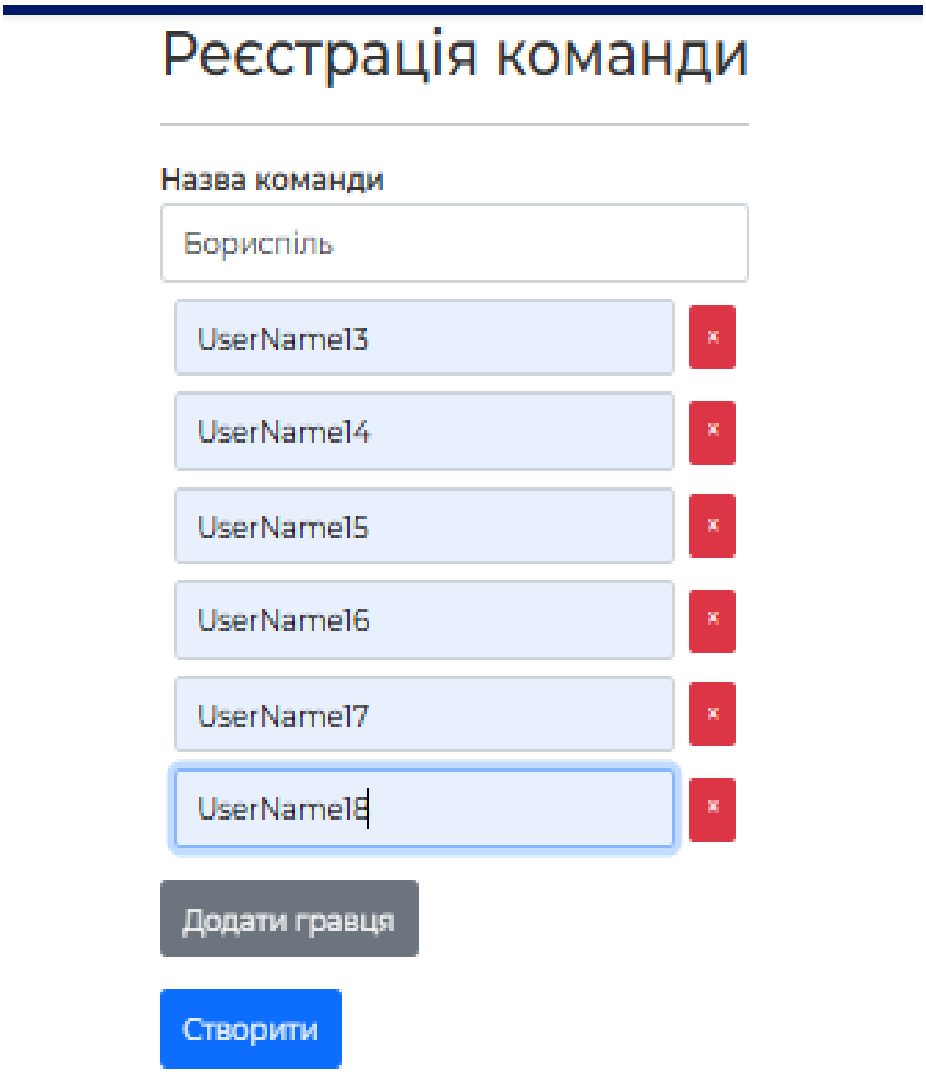


Рисунок 4.16 — Форма створення команди

Сторінка для адміністраторів дозволяє керувати користувачами, запускати перенавчання моделі штучного інтелекту.

Також, майже на кожній іншій сторінці, є перевірка на те, чи є данний користувач адміністратором і в залежності від цього користувачу відображається функціонал в залежності від ролі.

Для адміністратора відповідно вікно для створення турніру(рисунок 4.17). В залежності від типу, кількості команд, кількості груп підлаштовується алгоритм генерації розкладу. Інші інформаційні дані використовуються на інших сторінках.

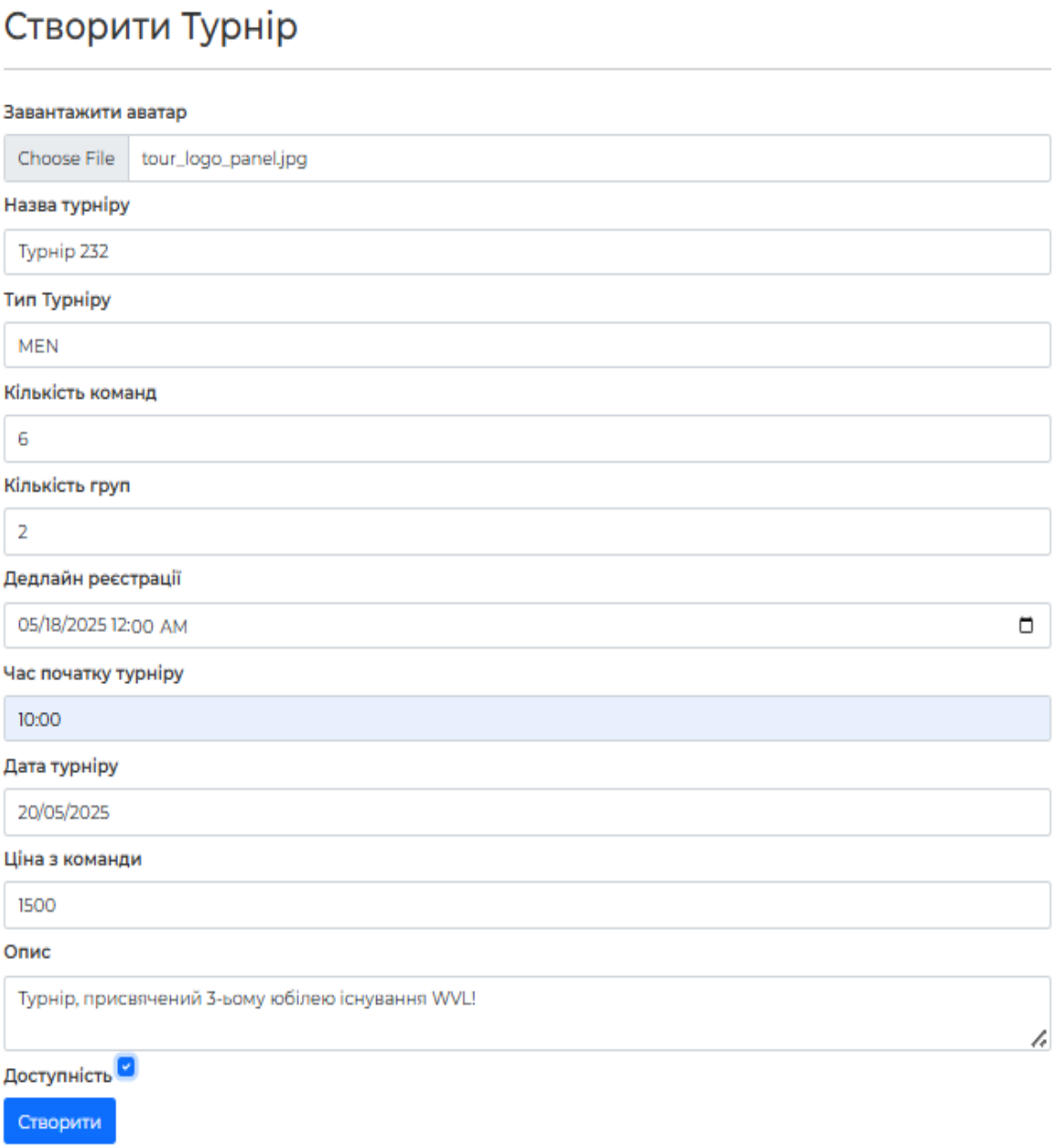


Рисунок 4.17 — Форма створення турнір

Натиснувши на кнопку “Створити” користувача поверне на сторінку з списком турнірів з вже відображеним новим турніром в списку.

На рисунку 4.18 відображається вікно профілю користувача з повноваженнями адміністратора. Сторінка містить особисті дані користувача, функціонал перегляду списку користувачів з можливими подальшими операціями над ними, кнопка переходу на сторінку з дотреновуванням моделі.

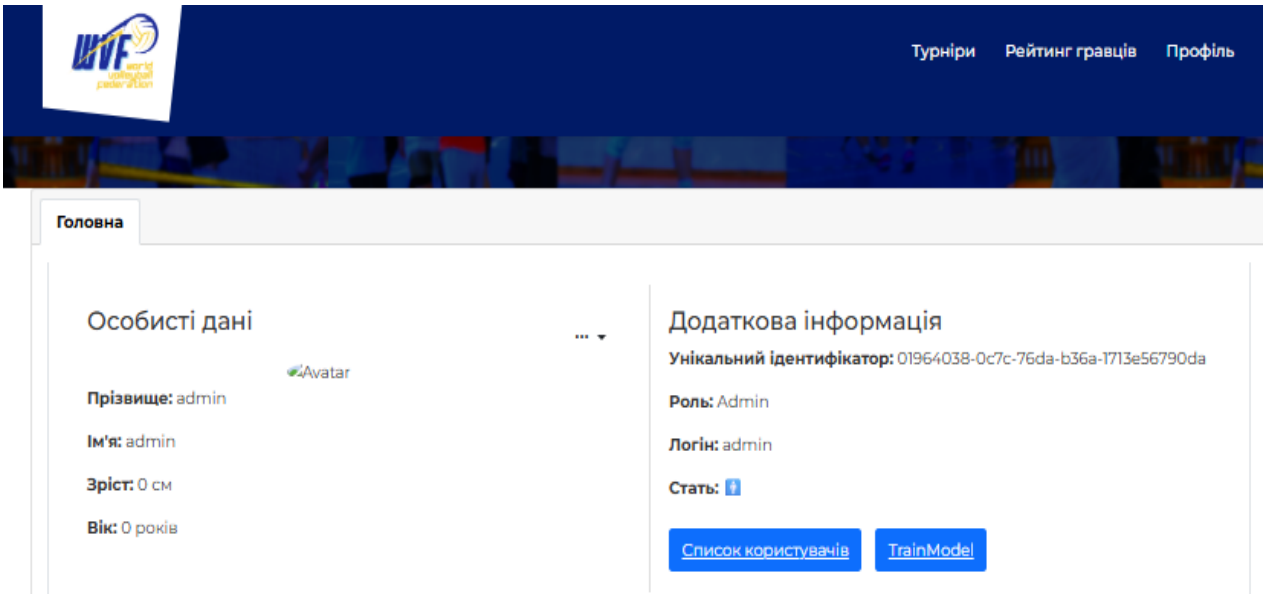


Рисунок 4.18 — Сторінка профілю адміну

На рисунку 4.19 зображена сторінка з можливістю перетренувати модель на новий датасет. Після дотренування моделі, виводиться відповідне повідомлення.

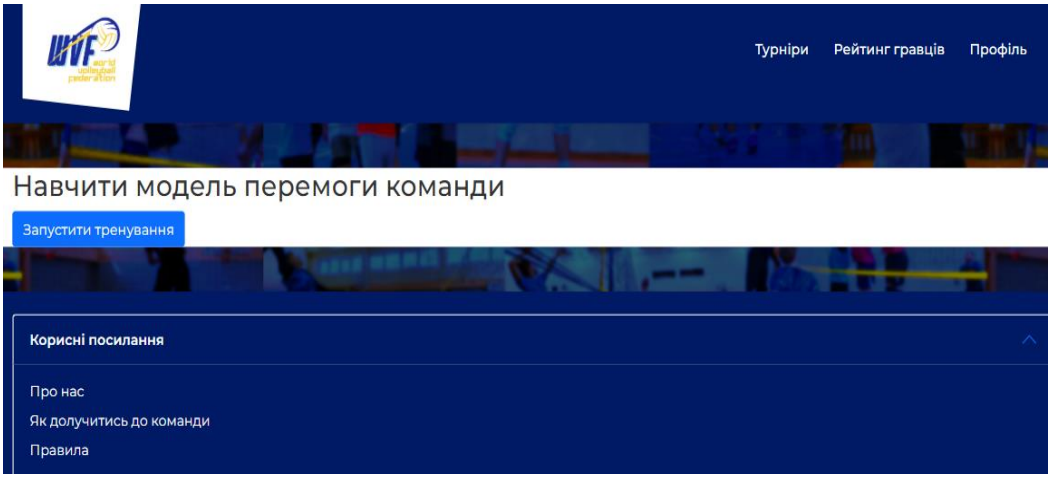


Рисунок 4.19 — Вікно перенавчання моделі

На сторінці інформації про гру(рисунок 4.20) ви можете побачити сторінку з детальною інформацією по гравцям, функціонал прогнозу переможця гри та статистикою кожного гравця.

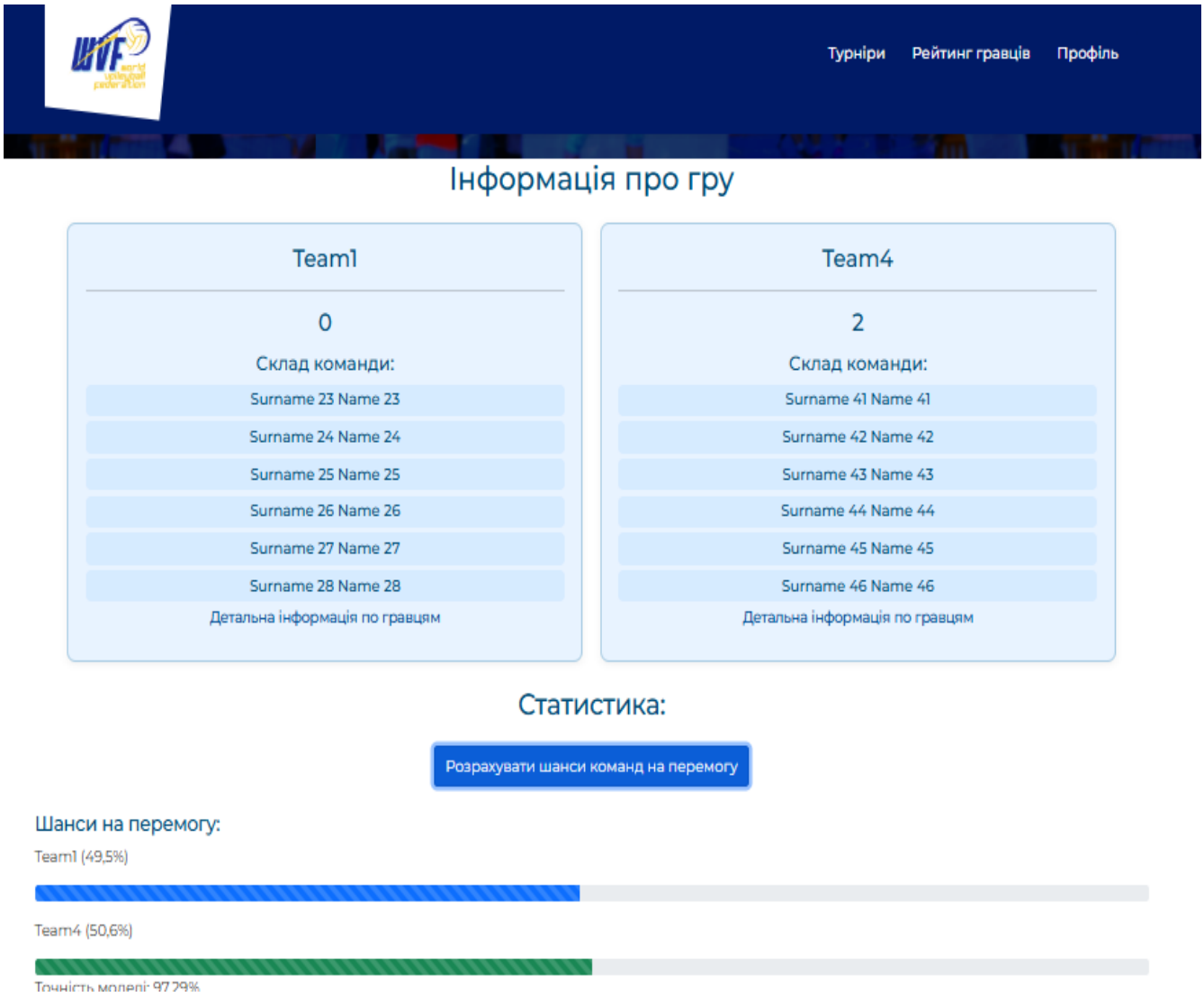


Рисунок 4.20 – Вікно інформації про гру і функціонал прогнозу переможця

Натиснувши на кнопку “Розрахувати шанси команд на перемогу”, ми отримаємо результати розрахунків штучного інтелекту. Запуститься в контролері метод з прогнозом і протягом декількох секунд виведеться прорахована вірогідність команд на перемогу в конкретній грі.

На рисунку 4.21 зображене модальне вікно з детальною статистикою гравців в кожній грі. Адміністратор має можливість редагувати ці значення. На базі цих значень вираховується індивудуальна статистика кожного гравця.

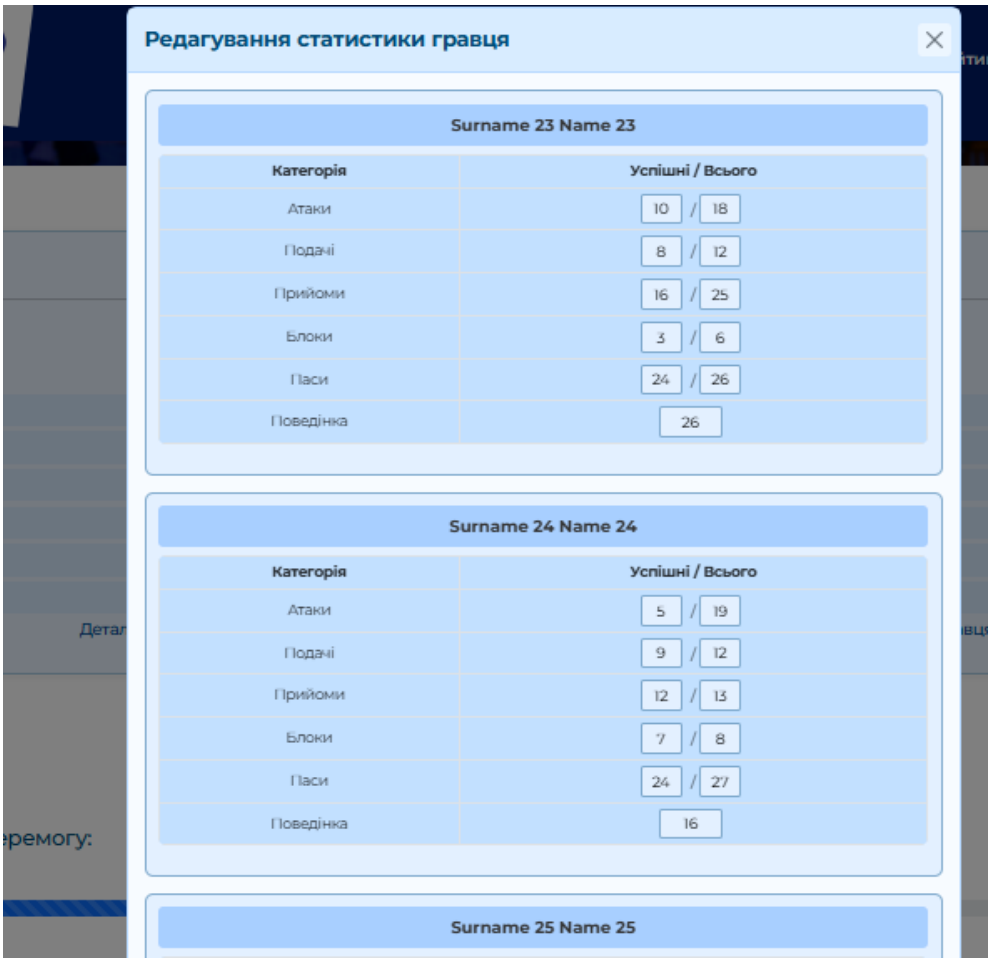


Рисунок 4.21 — Модальне вікно детальної статистики по гравцям

Збираючи статистику по гравцям з гри у гру, з кожним новим турніром, поповнюється датасет новими записами, які роблять прогнози штучного інтелекту все точнішими і точнішими.

ВИСНОВКИ

У результаті виконання дипломної роботи було спроєктовано та реалізовано web-систему для моніторингу спортивних турнірів із використанням технологій штучного інтелекту. Розроблена система дозволяє автоматизувати основні процеси організації, проведення та аналізу турнірів, забезпечуючи зручний інтерфейс для користувачів різних ролей — адміністраторів, гравців та гостей ресурсу.

У процесі дослідження було проведено аналіз існуючих рішень у цій сфері, виявлено їхні недоліки, серед яких — застарілий інтерфейс, відсутність адаптивності, обмежена функціональність та проблеми з оновленням і узгодженістю даних. На основі цих висновків було визначено ключові функціональні вимоги до нової системи.

Для реалізації було обрано сучасні інструменти та технології: мову програмування C#, фреймворк ASP.NET Core, ORM-технологію Entity Framework Core, СУБД PostgreSQL, а також бібліотеку ML.NET для побудови моделей машинного навчання. Було реалізовано 3-х шарову архітектуру з розділенням логіки, що підвищило масштабованість і супроводжуваність проєкту.

Особливу увагу приділено інтеграції інтелектуального модуля, що дозволяє аналізувати статистику та здійснювати базове прогнозування результатів на основі історичних даних. Це підвищує аналітичні можливості системи та відкриває перспективи для подальшого розвитку в напрямі персоналізованих рекомендацій або автоматичної генерації сіток турнірів.

Таким чином, поставлені завдання були повністю реалізовані. Отримані результати можуть бути впроваджені в реальну практику проведення спортивних турнірів, зокрема у сфері аматорського волейболу, а сама система — слугувати платформою для подальших досліджень та вдосконалень із використанням новітніх технологій штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft. ML.NET Documentation. *Microsoft*. URL: https://learn.microsoft.com/en-us/dotnet/machine-learning/ (date of access: 18.02.2025).

2. Jakia Sala. *Microsoft ML.NET Machine Learning for .NET Developers Using C#*, 2019. P. 176.

3. Esposito Dino. *Programming ML.NET*. Paperback, 2022. P. 256.

4. Andrew P. McMahon**.** *Machine Learning Engineering with Python. Second Edition*. PACKT Publishing, 2019. P. 600.

5. Microsoft. ASP.NET Core Documentation. *Microsoft*. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc (date of access: 12.02.2025).

6. Adam Freeman. *Pro ASP.NET Core MVC 6*. Apress, 2021. P. 1000.

7. Andrew Lock. *ASP.NET Core in Action, Second Edition*. Manning Publications, 2021. P. 800.

8. Brian Ding**.** *Building Modern Web Applications with ASP.NET Core Blazor: Learn how to use Blazor to create powerful, responsive, and engaging web applications*. *BPB Publications*. URL: https://www.barnesandnoble.com/w/building-modern-web-applications-with-aspnet-core-blazor-brian-ding/1143879622 (date of access: 12.03.2025).

9. Microsoft. C# Programming Guide. *Microsoft*. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (date of access: 03.03.2025).

10. Mark J. Price. *C# 12 and .NET 8 – Modern Cross-Platform Development*. PACKT Publishing, 2024. P. 800.

11. RB Whitaker. *The C# Player's Guide*. RB Whitaker, 2022. P. 450.

12. Microsoft. Entity Framework Core Documentation. *Microsoft*. URL: https://learn.microsoft.com/en-us/ef/core/ (date of access: 02.01.2025).

13. Jon P. Smith. *Entity Framework Core in Action*. Manning Publications, 2021. P. 500.

14. Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, 2017. P. 432.

15. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. P. 395.

16. Microsoft. Repository Pattern in C# with EF Core. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/intro (date of access: 05.03.2025).

17. Google Developers. Progressive Web Apps (PWA) Overview. *web.dev*. URL: https://web.dev/progressive-web-apps/ (date of access: 05.03.2025).

18. Microsoft. Razor Pages and Views in ASP.NET Core. *Microsoft*. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/overview (date of access: 01.05.2025).

ДОДАТОК А

Web-застосунок для організації та моніторингу
турнірів за допомогою штучного інтелекту

Лістинг програми

Аркушів 3

Київ — 2025

Код класу TeamVictoryTrainer тренування моделі ШІ

```
namespace Infrastructure.ML;

public class TeamVictoryTrainer(ApplicationDbContext dbContext)
{
   private readonly MLContext _mlContext = new();

   public async Task TrainAndSaveModelAsync()
{
   var games = await dbContext.Games.ToListAsync();
   var playerStats = await dbContext.PlayerStats.ToListAsync();

   List<TeamGameComparisonStats> trainingData = new();

   foreach (var game in games)
   {
      var team1Stats = GetAggregatedStats(game, game.Team1Id!.Value, playerStats);
      var team2Stats = GetAggregatedStats(game, game.Team2Id!.Value, playerStats);

      if (team1Stats == null || team2Stats == null)
         continue;

      trainingData.Add(new TeamGameComparisonStats
      {
         AttackSuccessPercentTeam1 = team1Stats.AttackSuccessPercent,
         BlockSuccessPercentTeam1 = team1Stats.BlockSuccessPercent,
         ServeSuccessPercentTeam1 = team1Stats.ServeSuccessPercent,
         ReceiveSuccessPercentTeam1 = team1Stats.ReceiveSuccessPercent,
         SetsSuccessPercentTeam1 = team1Stats.SetsSuccessPercent,
         AveragePointsPerSetTeam1 = team1Stats.AveragePointsPerSet,
         VibeTeam1 = team1Stats.Vibe,
         SetWinRateTeam1 = team1Stats.SetWinRate,

         AttackSuccessPercentTeam2 = team2Stats.AttackSuccessPercent,
         BlockSuccessPercentTeam2 = team2Stats.BlockSuccessPercent,
         ServeSuccessPercentTeam2 = team2Stats.ServeSuccessPercent,
         ReceiveSuccessPercentTeam2 = team2Stats.ReceiveSuccessPercent,
         SetsSuccessPercentTeam2 = team2Stats.SetsSuccessPercent,
         AveragePointsPerSetTeam2 = team2Stats.AveragePointsPerSet,
         VibeTeam2 = team2Stats.Vibe,
         SetWinRateTeam2 = team2Stats.SetWinRate,

         Victory = game.Team1Sets > game.Team2Sets
      });
   }

   IDataView fullData = _mlContext.Data.LoadFromEnumerable(trainingData);

   var split = _mlContext.Data.TrainTestSplit(fullData, testFraction: 0.2);

   var pipeline = _mlContext.Transforms.Concatenate("Features",
         nameof(TeamGameComparisonStats.AttackSuccessPercentTeam1),
         nameof(TeamGameComparisonStats.BlockSuccessPercentTeam1),
         nameof(TeamGameComparisonStats.ServeSuccessPercentTeam1),
         nameof(TeamGameComparisonStats.ReceiveSuccessPercentTeam1),
         nameof(TeamGameComparisonStats.SetsSuccessPercentTeam1),
         nameof(TeamGameComparisonStats.VibeTeam1),
         nameof(TeamGameComparisonStats.AveragePointsPerSetTeam1),
         nameof(TeamGameComparisonStats.SetWinRateTeam1),
         nameof(TeamGameComparisonStats.AttackSuccessPercentTeam2),
         nameof(TeamGameComparisonStats.BlockSuccessPercentTeam2),
         nameof(TeamGameComparisonStats.ServeSuccessPercentTeam2),
         nameof(TeamGameComparisonStats.ReceiveSuccessPercentTeam2),
         nameof(TeamGameComparisonStats.SetsSuccessPercentTeam2),
         nameof(TeamGameComparisonStats.VibeTeam2),
         nameof(TeamGameComparisonStats.AveragePointsPerSetTeam2),
         nameof(TeamGameComparisonStats.SetWinRateTeam2))
      .Append(_mlContext.BinaryClassification.Trainers.SdcaLogisticRegression(labelColumnName: "Victory", featureColumnName: "Features"));


   var model = pipeline.Fit(split.TrainSet);


   var predictions = model.Transform(split.TestSet);
   var metrics = _mlContext.BinaryClassification.Evaluate(predictions, labelColumnName: "Victory");

   Directory.CreateDirectory("MLModels");
   _mlContext.Model.Save(model, fullData.Schema, "MLModels/TeamVictoryPredictor.zip");
```

```
    var modelMetrics = new ModelMetrics
    {
        Accuracy = AddNoise(metrics.Accuracy),
        Auc = AddNoise(metrics.AreaUnderRocCurve),
        F1Score = AddNoise(metrics.F1Score)
    };
    File.WriteAllText("MLModels/TeamVictoryMetrics.json", JsonSerializer.Serialize(modelMetrics));
}


    private TeamStatsSummary? GetAggregatedStats(
        WVF.Domain.Models.Game game,
        int teamId,
        List<WVF.Domain.Models.PlayerStats> playerStats)
    {
        var ps = playerStats
            .Where(p => p.GameId == game.Id && p.TeamId == teamId)
            .ToList();

        if (!ps.Any()) return null;

        float SafeDivide(int success, int all) => all != 0 ? (float)success / all : 0f;

        return new TeamStatsSummary
        {
            AttackSuccessPercent = SafeDivide(ps.Sum(p => p.AttacksSuccess!.Value), ps.Sum(s => s.AttacksAll!.Value)),
            BlockSuccessPercent = SafeDivide(ps.Sum(p => p.BlocksSuccess!.Value), ps.Sum(p => p.BlocksAll!.Value)),
            ServeSuccessPercent = SafeDivide(ps.Sum(p => p.ServesSuccess!.Value), ps.Sum(p => p.ServesAll!.Value)),
            ReceiveSuccessPercent = SafeDivide(ps.Sum(p => p.ReceivesSuccess!.Value), ps.Sum(p => p.ReceivesAll!.Value)),
            SetsSuccessPercent = SafeDivide(ps.Sum(p => p.SetsSuccess!.Value), ps.Sum(p => p.SetsAll!.Value)),
            Vibe = ps.Average(p => (float)p.Vibe!.Value),
            AveragePointsPerSet = teamId == game.Team1Id
                ? (game.Team1Set1Points + game.Team1Set2Points + game.Team1Set3Points) / 3f
                : (game.Team2Set1Points + game.Team2Set2Points + game.Team2Set3Points) / 3f,
            SetWinRate = teamId == game.Team1Id
                ? (float)game.Team1Sets / (game.Team1Sets + game.Team2Sets)
                : (float)game.Team2Sets / (game.Team1Sets + game.Team2Sets)
        };
    }
}
```