МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСТИТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

### Інформаційна система «Шахи»

Курсовий проект з дисципліни «Бази даних»

|  |  |
| --- | --- |
| Виконавці  здобувачі ВО гр. ПІ-121 | Туревський Д.В.,  Крапив’янський Д.А. |
| Керівник викладач | Білоус І.В |

**Чернігів 2024**

## ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи з дисципліни «Бази даних»

Туревський Д.В. , гр. ПІ-121

Крапив’янський Д.А. , гр. ПІ-121

Тема роботи: БД "Шахи"

**Передбачувані технічні та експлуатаційні результати роботи:**

Програмна система «Шахи» являє собою загальнодоступний додаток, який дозволить відслідковувати основні події в світі шахів нові турніри, гравців, титули та інше, а також керувати базою даних через зручний інтерфейс.

Розробка БД повинна включати:

1) побудову концептуальної моделі предметної області, що включає не менш 3-х об'єктів, пов'язаних між собою взаємозв'язками типу 1: М, у вигляді діаграми «сутність-зв'язок»;

2) побудову нормалізованої логічної моделі предметної області (не менше 3-х таблиць) із зазначенням: первинних і зовнішніх ключів, типів даних атрибутів;

3) реалізацію на рівні структури БД засобів забезпечення цілісності даних: унікальність і обов'язковість введення первинних ключів; підтримка посилальної цілісності для зовнішніх ключів; значення атрибутів за замовчуванням і обов'язковість введення заданих атрибутів;

4) реалізацію базових запитів до БД на пошук і додавання даних;

5) реалізацію бізнес-логіки зі сторони серверу БД: тригерів, збережених процедур та представлень;

6) реалізацію розділеного доступу до БД зі сторони серверу БД: користувачі та їх права;

7) реалізацію надійності збереження даних: механізми реплікації та кластеризації даних.

Взаємодія з БД має бути реалізована в архітектурі клієнт-сервер з наступними відокремленими рівнями:

1. бази даних;
2. доступу до даних з використанням технологій об’єктно-реляційного маппінгу;
3. обробки даних (бізнес-логіки);
4. сервісів;
5. інтерфейсу**.**

**ЗМІСТ**

[**ВСТУП** 4](#_Toc155970239)

[**1. АНАЛІЗ ВИРІШУВАНОЇ ЗАДАЧІ** 5](#_Toc155970240)

[1.1 Аналіз предметної області 5](#_Toc155970241)

[1.2 Мета і завдання системи 7](#_Toc155970242)

[1.3 Призначення системи 8](#_Toc155970243)

[1.4 Вимоги до системи 9](#_Toc155970244)

[**2. ПРОЕКТУВАННЯ** 10](#_Toc155970245)

[2.1 Проектування таблиць бази даних 10](#_Toc155970246)

[2.2 Проектування логічної системи БД 12](#_Toc155970247)

[2.3 Проектування архітектури системи 14](#_Toc155970248)

[2.4 Технології реалізації системи 15](#_Toc155970249)

[2.5 Проектування бізнес-логіки та бізнес-правил 17](#_Toc155970250)

[2.6 Проектування шару відображення 19](#_Toc155970251)

[**3. РОЗРОБКА СИСТЕМИ** 22](#_Toc155970252)

[3.1 Розробка бази даних системи 22](#_Toc155970253)

[3.1.1 Розробка фізичної схеми бази даних 22](#_Toc155970254)

[3.1.2 Забезпечення цілісності даних 22](#_Toc155970255)

[3.2 Реалізація бізнес-логіки зі сторони серверу 25](#_Toc155970256)

[3.3 Розробка модулів системи 29](#_Toc155970257)

[3.3.1 Розробка модулів шару бізнес-логіки 29](#_Toc155970258)

[3.3.2 Розробка модулів шару доступу до даних 32](#_Toc155970259)

[3.3.3 Розробка модулів шару сервісів 50](#_Toc155970260)

[3.4 Тестування створеної системи 54](#_Toc155970261)

[**ВИКОРИСТАНІ ДЖЕРЕЛА** 58](#_Toc155970262)

# **ВСТУП**

Шахи, як древній та високоінтелектуальний гра, завжди привертало увагу людей своєю стратегічністю та величністю. Ця гра, що виникла в Індії, перетнула вікові та культури, залишаючи свій неперевершений слід у світі настільних інтелектуальних розваг. У сучасному інформаційному суспільстві, де технології та програмне забезпечення стають неотдільною частиною нашого щоденного життя, шахи не залишаються осторонь цього тренду.

Проектування будь-якої інформаційної системи пов'язане з управлінням та обробкою даних, і шахова гра не є винятком. Інтерес до шахів, як до інтелектуальної гри, визначає необхідність в створенні високоефективної та зручної інформаційної платформи для любителів цієї гри.

У світі, де віртуальні арени та онлайн-платформи змагань розвиваються стрімкими темпами, створення інноваційної інформаційної системи для шахів є актуальною задачею. Великий попит на високоякісні інтерактивні платформи для гри в шахи створює необхідність у створенні системи, яка не лише полегшить сам процес гри, але й надасть користувачам нові можливості для вивчення стратегій, взаємодії та спілкування.

Створення інформаційної системи для гри в шахи відкриває безліч можливостей для надання інформації та розвитку у шаховому співтоваристві. Однією з ключових функцій системи буде надання ігровим учасникам доступу до розмаїття інформації, яка поглиблює їхнє розуміння гри та підвищує їхній рівень гри.

Система надає детальну інформацію про титули гравців, визначаючи їхні досягнення та рівень в шаховому світі. Гравці матимуть можливість вивчати історію своїх власних досягнень та досягнень інших учасників, що створить стимул для покращення власної гри.

Турнірна інформація також буде включена в систему, дозволяючи гравцям брати участь у різноманітних турнірах, дізнаватися про події та результати. Можливість взаємодії з іншими учасниками, обговорення стратегій та аналіз партій допомагатиме гравцям розвивати свої шахові навички та сприяти взаємопідтримці в спільноті.

Отже, розробка інформаційної системи для гри в шахи визначається не тільки потребами шахових гравців у зручній платформі, але й стрімким розвитком віртуальних ігор та інтересом до інтелектуальних розваг. Ця інноваційна система має на меті перетворити гру в шахи в захопливий та навчальний досвід для всіх, хто цінує глибокий інтелектуальний виклик.

# **1. АНАЛІЗ ВИРІШУВАНОЇ ЗАДАЧІ**

## 1.1 Аналіз предметної області

Обрана нами предметна область “Шахи” буде створена з метою відслідковування перебігу подій в світі шахів, тобто буде використовуватися для створення онлайн-застосунку, що буде являти собою довідник з можливістю його частого оновлення та пошуку потрібної інформації різними способами. Щодня з’являється нові гравці, стратегії, проводяться турніри для виявлення найкращого гравця. Відслідковувати це самому є практично неможливим тому онлайн-ресурс для цього є досить потрібним. Тому для створення застосунку ми повинні обрати самі важливі аспекти гри в шахи на міжнародному рівні.

На основі складеного опису ми створили модель предметної області що складається з основних аспектів гри в шахи, а саме:

1. Шахіст;

2. Титул;

3. Матч.

4. Дебют;

5. Використання дебюту

6. Турнір;

7. Участь у турнірі

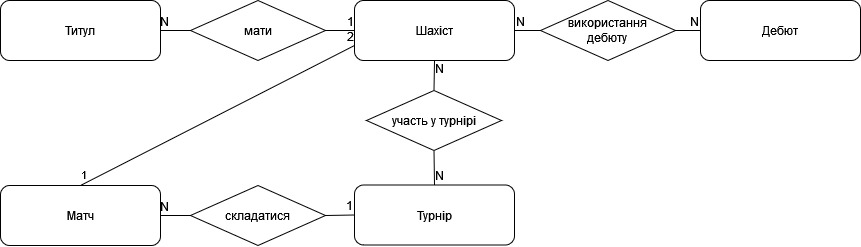
На рисунку 1.1 наведено наведено уточнену модель бази даних:

Рисунок 1.1 – Уточнена модель бази даних

Після створення моделі для нашої предметної області можна побачити, що вона складається з семи сутностей, які які пов’язані між собою зв’язком типу «1:М». Такий зв’язок означає, що екземпляри однієї сутності «1» відповідають декільком записам іншої «М» . Для кожної сутності визначили первинний ключ(код), а також неключові атрибути:

1. Шахіст. Атрибутами сутності шахіст будуть: « код », « ім'я », « стать », « дата народження », « країна », « національність », « рейтинг », « матчі/рік », « перемог »

2. Титул. Атрибутами сутності титул будуть: « код », « назва », « рік », « код гравця».

3. Матч. Атрибутами сутності матч будуть: « код », « код турніра », « код гравця 1 », « код гравця 2 », « дата », « рахунок гравця 1 », « рахунок гравця 2 ».

4. Дебют. Атрибутами сутності дебют будуть: « код », « назва », « категорія », « рік », « найвідоміший гравець ».

5. Використання дебюту. Атрибутами данної сутності будуть: « код », « код дебюту », « код гравця », « число балів ».

6. Турнір. Атрибутами сутності турнір будуть: « код », « тип », « назва », « дата початку », « дата завершення », « режим проведення », « країна », « місто », « призовий фонд », « кількість матчів », « кількість учасників ».

7. Участь у турнірі. Атрибутами данної сутності будуть: « код », « код гравця », « код турніру », « дата реєстрації ».

Для таблиць всі поля « код » є первинними ключами, а поля « код … » зовнішніми.

Складемо ряд істинних висловлювань на основі моделі предметної області для визначення параметрів між об’єктами сутностей.

* Шахіст може мати декілька титулів, бути участником в кількох матчах та брати участь в кількох турнірах, також він може використовувати різні дебюти .
* Турнір має кілька матчів та тільки одного чемпіона.
* Матч обов’язково має двух гравців.

Як можна побачити головною сутністю нашої предметної області є саме гравець як і в більшості видів спорту в результаті ми отримуємо, що створені зв’язки будуть відноситися до типу «один-до-багатьох», зовнішні ключи яких посилаються на батьківські сутності і не можуть містити порожні значення. Також необхідно зазначити, що батьківські сутності можуть існувати, навіть якщо на них не посилається жодна з дочірніх.

Із зроблених припущень створимо ER-діаграму нашої бази даних.

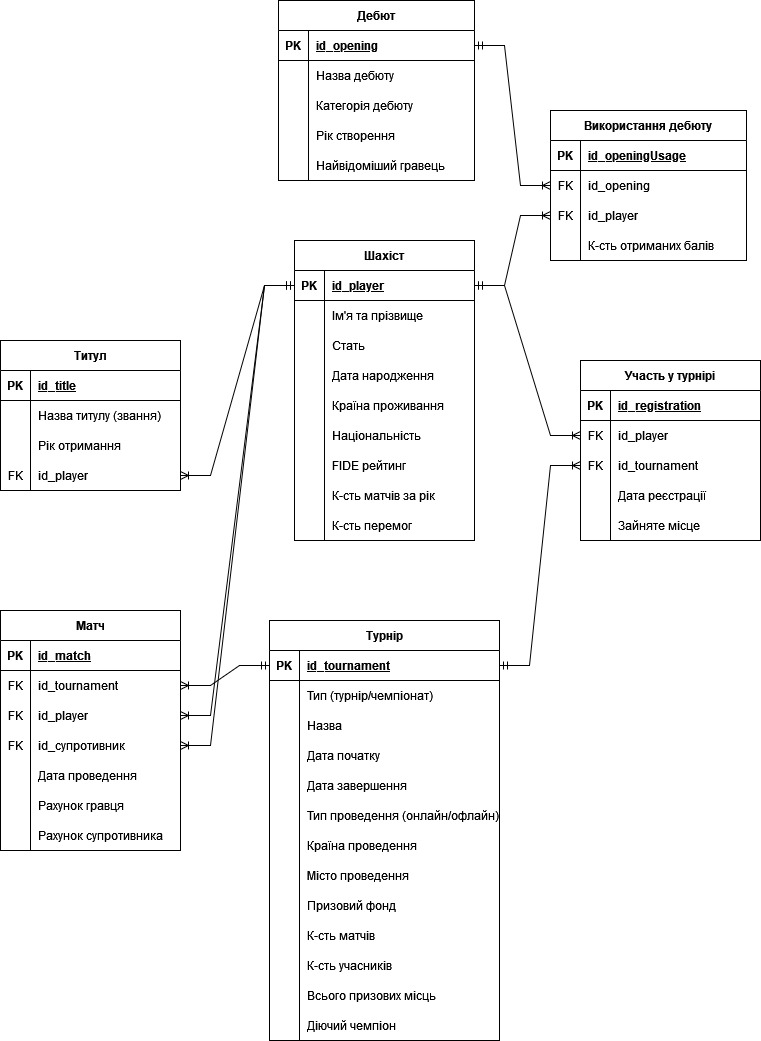
є

Рисунок 1.2 – ER-діаграму бази даних "Шахи"

## 1.2 Мета і завдання системи

Основною метою системи, що буде створенна для данної предметної області є надання гравцям в шахи користої інформації, що може їх допомогти в підготовці чи плануванню своїх цілей. Тобто завдання буде полягати в тому щоб створити зручну інформаційну систему для:

* Відслідковування створення нових турнірів.
* Відслідковування появи нових гравців.
* Можливість переглядати титули, проведені матчі та використані стратегії, як всіх загалом так і конкретниї гравців.

Тобто в результаті в нас повинен бути онлайн-ресурс, що допоможе гравцям в шахи відслідковувати наявні турніри та отримувати користну інформації для кращої підготовки до них.

## 1.3 Призначення системи

Система передбачає можливість використання в таких режимах: адміністратор, користувач, а також незареєстрований користувач. В результаті ми встановимо відповідні обмеження для різних типів користувачів.

Так як ми маємо досить багато сутностей в яких реалізовані схожі функції та можливості їх опис буде більш зрозумілим у вигляді таблиць.

Таблиця 1.1 – Опис можливостей незареєстрованого користувача

|  |  |
| --- | --- |
| Незареєстрований | |
| Функція | Опис |
| Перегляд | Можливість переглядати вміст основних таблиць бази даних |
| Перегляд зв'язаних сутностей | Можливість переглядати інформацію що пов'язана з конкретною сутністью.  Наприклад: Всі титули отримані певним гравцем. |
| Пошук | Можливість використовувати пошук з обиранням поля пошуку та введенням значення. |
| Доступ до історії змін в турнірах | Можливість переглядати внесені зміни в створенні турніри. |
| Статистика | Можливість перегляду статичтики сайту, кількості об'єктів в таблицях. |
| Вхід | Можливість виконати вхід на сайт з отриманням більшого функціоналу. |

Таблиця 1.2 – Опис можливостей звичайного користувача

|  |  |
| --- | --- |
| Користувач | |
| Функція | Опис |
| Перегляд | Можливість переглядати вміст основних таблиць бази даних |
| Перегляд зв'язаних сутностей | Можливість переглядати інформацію що пов'язана з конкретною сутністью.  Наприклад: Всі титули отримані певним гравцем. |
| Пошук | Можливість використовувати пошук з обиранням поля пошуку та введенням значення. |
| Доступ до історії змін в турнірах | Можливість переглядати внесені зміни в створенні турніри. |
| Статистика | Можливість перегляду статичтики сайту, кількості об'єктів в таблицях. |
| Додавання | Можливість додавати нові об'єкти до всіх основних таблиць бази даних. |
| Вихід | Можливість виконати вихід з сайту з втратою розширеного функціоналу. |

Таблиця 1.3 – Опис можливостей адміністратора

|  |  |
| --- | --- |
| Адміністратор | |
| Функція | Опис |
| Перегляд | Можливість переглядати вміст основних таблиць бази даних |
| Перегляд зв'язаних сутностей | Можливість переглядати інформацію що пов'язана з конкретною сутністью.  Наприклад: Всі титули отримані певним гравцем. |
| Пошук | Можливість використовувати пошук з обиранням поля пошуку та введенням значення. |
| Доступ до історії змін в турнірах | Можливість переглядати внесені зміни в створенні турніри. |
| Статистика | Можливість перегляду статичтики сайту, кількості об'єктів в таблицях. |
| Додавання | Можливість додавати нові об'єкти до всіх основних таблиць бази даних. |
| Редагування | Можливість редагувати об'єкти всіх основних таблиць бази даних. |
| Видалення | Можливість видалення об'єктів всіх основних таблиць бази даних. |
| Вихід | Можливість виконати вихід з сайту з втратою розширеного функціоналу. |

## 1.4 Вимоги до системи

Вимоги до системи полягають в створенні онлайн-ресурсу для надання корисної інформації для шахістів, щодо різних аспектів об’єктами яких є турніри, титули, матчі, дебюти(стратегії), матчі, участь в турнірах та відслідковування використаних дебютів .

Також потрібно враховувати вимоги бізнес-логіки та обмежень бази даних. Тобто потрібно врахувати особливості типів даних, що ми використовуємо, а саме реалізувати вибір значень в обмеженнях типу “enum”, обмеження в правильності дати початку та закінчення турнірів. Логіка повинна бути представлена у вигляді основних дій додавання, редагування та видалення для всіх основних таблиць. Далі повинна бути реалізована розширена логіка системи пошуку за різними даними(полями) та можливість переходу до вибору сутностей, що пов’язані з конкретним об’єктом.

# **2. ПРОЕКТУВАННЯ**

## 2.1 Проектування таблиць бази даних

Опис таблиць БД на основі аналізу бази даних із вказаними типами полів, обмеженнями цілісності описом наведено у таблиці 4**.**

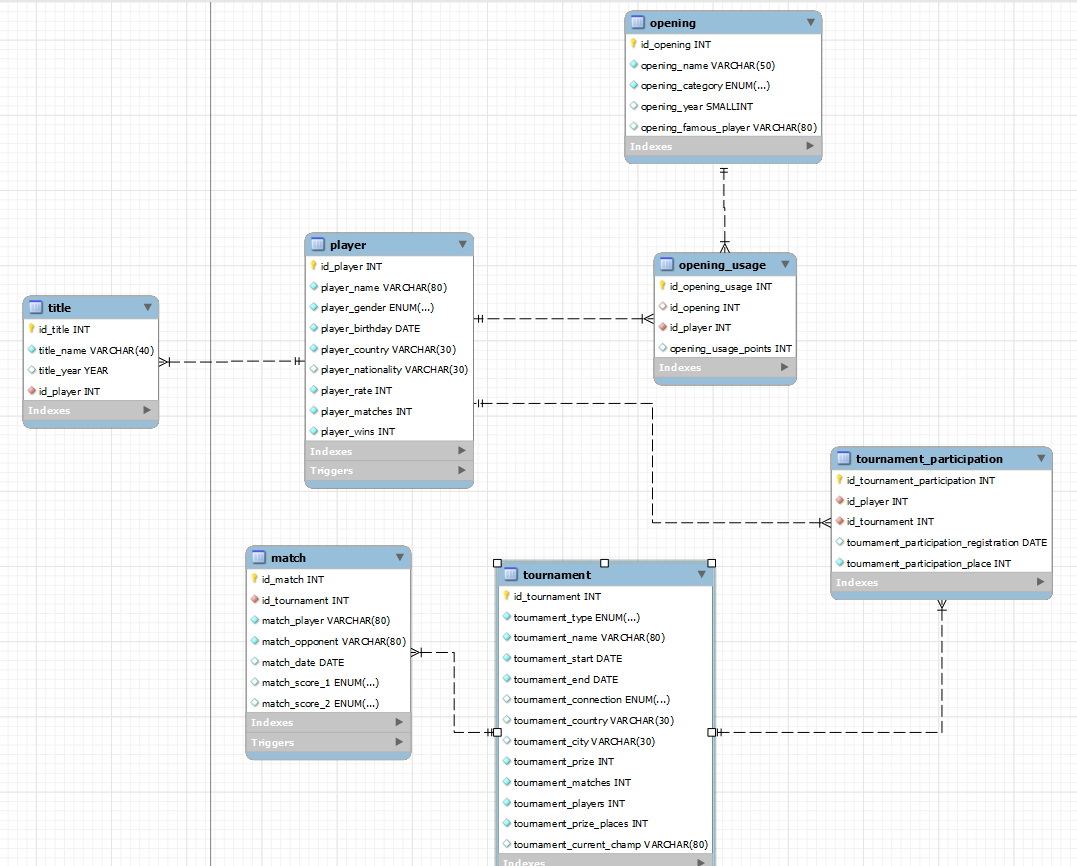
Таблиця 2.1 – Опис атрибутів об’єктів «Шахи»

|  |  |  |  |
| --- | --- | --- | --- |
| **Об’єкт** | **Атрибут** | **Тип** | **Обмеження** |
| Шахіст | Код\_гравця | Число | унікальне, ключове поле |
| Ім’я та прізвище | Строка | розміром до 80 символів |
| Стать | Строка | розміром до 8 символів, приймає значення «чоловіча», «жіноча» |
| Дата народження | Дата | старше “01/01/2015” |
| Країна проживання | Строка | розміром до 30 символів |
| Національність | Строка | розміром до 30 символів |
| FIDE рейтинг | Число | [2000…3500] |
| К-сть матчі за рік | Число | [0…500] |
| К-сть перемог | Число | [0…500], значення не може бути більшим, ніж у полі «К-сть матчів за рік» |
| Титул | Код\_титулу | Число | унікальне, ключове поле |
| Назва титуту | Строка | розміром до 40 символів |
| Рік отримання | Число | [1930…2023] |
| Код\_гравця | Число | зовнішнє посилання |
| Використання дебюту | Код\_використання | Число | унікальне, ключове поле |
| Код\_дебюту | Число | зовнішнє посилання |
| Код\_гравця | Число | зовнішнє посилання |
| К-сть отриманих балів | Число | [0…100] |
| Дебют | Код\_дебюту | Число | унікальне, ключове поле |
| Назва дебюту | Строка | розміром до 50 символів |
| Категорія дебюту | Строка | розміром до 30 символів |
| Рік створення | Число | [1930…2023] |
| Найбільш відомий гравець | Строка | розміром до 80 символів |
| Участь у турнірі | Код\_реєстрації | Число | унікальне, ключове поле |
| Код\_гравця | Число | зовнішнє посилання |
| Код\_турніра | Число | зовнішнє посилання |
| Дата реєстрації | Число | старше “01/02/2023”  за замовчуванням поточна дата |
| Зайняте місце | Число | [0…100], значення не може бути більшим, ніж у полі таблиці Турнір «Всього призових місць» |
| Турнір | Код\_турніра | Число | унікальне, ключове поле |
| Тип | Строка | розміром до 9 символів, приймає значення «турнір», «чемпіонат» |
| Назва | Строка | розміром до 80 символів |
| Дата початку | Дата | старше “01/02/2023”  за замовчуванням поточна дата |
| Дата завершення | Дата | старше “01/02/2023”  за замовчуванням поточна дата; значення не може бути меншим, ніж у полі «Дата початку» |
| Тип проведення | Строка | розміром 6 символів, приймає значення «онлайн», «офлайн» |
| Країна проведення | Строка | розміром до 30 символів |
| Місто проведення | Строка | розміром до 30 символів |
| Призовий фонд | Число в грошовому записі |  |
| К-сть матчів | Число | [0…100] |
| К-сть учасників | Число | [0…100] |
| Всього призових місць | Число | [0…100] |
| Діючий чемпіон | Строка | розміром до 80 символів |
| Матч | Код\_матча | Число | унікальне, ключове поле |
| Код\_турніра | Число | зовнішнє посилання |
| Код\_таблиці | Число | зовнішнє посилання |
| Код\_супротивника | Число | зовнішнє посилання |
| Дата проведення | Дата | старше “01/02/2023”  за замовчуванням поточна дата; значення не може бути меншим, ніж у полі таблиці Турнір «Дата початку», і більшим, ніж у полі таблиці Турнір «Дата завершення» |
| Рахунок гравця | Число | [0…1], приймає значення 0, 0.5, 1 |
| Рахунок супротивника | Число | [0…1], приймає значення 0, 0.5, 1 |

В данній таблиці була вказана вся основна інформація про таблиці, їх поля, типи та обмеження. В тому числі були вказані обмеження типу “enum”, тобто вибору первного варінту такі як стать, типи турнірів та інше. Далі на основі цього ми буде проектувати логічну схему бд.

## 2.2 Проектування логічної системи БД

На основі отриманого опису таблиць побудували логічну модель бази даних, яка зображена на рисунку 2.1.

Рисунок 2.1 — Логічна схема бази даних

База даних «Шахи» складається з 7 таблиць при чому вони мають чітку неподільність значень в кожному полі, записи таблиці не впливають один на одного та розміщення записів в таблиці ніяк не впливає на інші, так як кожний із рядків унікальні, а також що є дуже важливим для нормалізації першого рівня, первинний ключ для кожної із створених таблиць не повторюється в межах кожної з них і має конкретну роль ідентифікатора запису.

Таблиці в даній базі можна розділити на первинні та допоміжні, первинні таблицями є матч, гравець, турнір, титул, стратегії вони є початковими та несуть в собі незалежну інформації про сутності бази. Допоміжні таблиці створені з метою уникнення зв'язка багато до багатьох, так як він є небажаним та його потрібно уникати для створення правильної БД. Тому були створенні таблиці учаті в турнірі та використання стратерій.

Також в базі відсутнє дублювання інформації з різних таблиць, так як вони всі пов'язані між собою зовнішніми ключами. Основною таблицею бази є гравець інформація про якого є основною, всі інші розширюють дані про різні аспекти його досягнень, такі як титули, стратерії, учать в матчах.

## 2.3 Проектування архітектури системи

Архітектура системи відповідає типу клієнт-сервер, призначена для організації доступу користувачів до бази даних (рисунок 2.2). Її можна розділяти їх на чотири рівні:

1. Інтерфейсу користувача;
2. Сервісів;
3. Бізнес-логіка;
4. Доступ до даних бази.

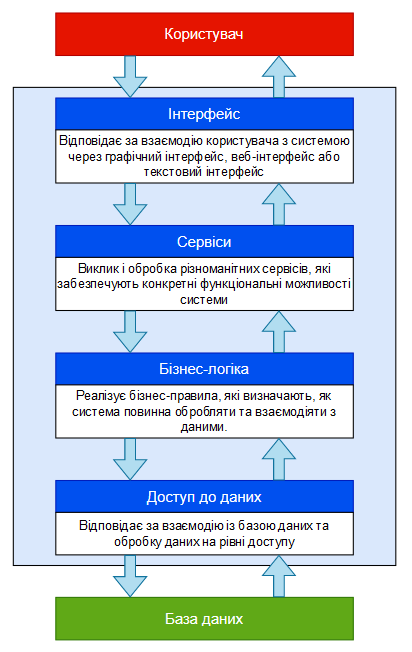
Архітектура системи має такий вигляд:

Рисунок 2.2 – Архітектура системи

Більш детально про рівні системи та їх функції наведено далі.

1) Інтерфейс користувача:

Цей рівень відповідає за взаємодію користувача з системою. Його завдання - надати зручний та ефективний інтерфейс для користувачів взаємодіяти з системою.

Функції:

* Прийом введених даних від користувача.
* Показ інформації користувачу у зручній формі.
* Може включати графічний інтерфейс, веб-інтерфейс, текстовий інтерфейс тощо.

2) Сервіси:

Цей рівень відповідає за обробку конкретних функцій та послуг системи. Сервіси можуть включати авторизацію, аутентифікацію, обробку платежів, тощо.

Функції:

* Виклик і обробка різноманітних сервісів, які забезпечують конкретні функціональні можливості системи.
* Забезпечення взаємодії між рівнями бізнес-логіки і інтерфейсом користувача

3) Бізнес-логіка:

Цей рівень визначає логіку та правила обробки даних системи. Реалізує бізнес-правила, які визначають, як система повинна обробляти та взаємодіяти з даними.

Функції:

* Визначення та реалізація бізнес-логіки, такої як обчислення, валідація та обробка даних згідно з вимогами бізнесу.
* Забезпечення інтеграції різних сервісів для досягнення конкретного бізнес-функціоналу.

4) Доступ до даних бази:

Цей рівень відповідає за взаємодію із базою даних та обробку даних на рівні доступу. Забезпечує збереження та витяг даних із бази даних.

Функції:

* Виконання запитів до бази даних та отримання необхідної інформації.
* Забезпечення безпеки та цілісності даних на рівні доступу.

## 2.4 Технології реалізації системи

Для розробки данної системи були використані багато різноманітних технологій, що використовувались в різних рівнях програми. Наведемо які саме технології задіяні в проекті.

1. HTML, є стандартною мовою розмітки для створення та відображення веб-сторінок. Вона визначає структуру документа та включає елементи, які дозволяють вставляти текст, зображення, посилання, форми та інші елементи на веб-сторінці.

2. CSS, є мовою стилізації, що використовується для оформлення вигляду та форматування веб-документів, написаних мовою HTML або XML. CSS дозволяє розділити структуру документа від його представлення, що полегшує зміну вигляду веб-сторінок без зміни їх структури.

3. Expression Language (EL) - це мова виразів, яка використовується в технологіях, пов'язаних із веб-розробкою, таких як JavaServer Pages. EL дозволяє вставляти вирази прямо в сторінки JSP або інші частини веб-застосунків, спрощуючи доступ до даних та виконання виразів безпосередньо в шаблонах сторінок.

4. JavaScript - це високорівнева, об'єктно-орієнтована мова програмування, яка використовується для створення динамічних веб-сайтів. Вона є однією з ключових технологій для розробки клієнтської частини веб-додатків

5. JSTL, або JavaServer Pages Standard Tag Library, є бібліотекою тегів, розробленою для спрощення і поліпшення роботи з JavaServer Pages. JSTL вводить нові теги для виконання рутинних задач у JSP-сторінках та дозволяє розробникам використовувати ці теги для виконання різних операцій, таких як робота з циклами, умовні перевірки, робота з колекціями і форматування даних.

6. Java Servlet - це Java-клас, який реалізує інтерфейс javax.servlet.Servlet і призначений для обробки запитів і відповідей на веб-сервері або веб-контейнері. Сервлети використовуються для розробки динамічних веб-сайтів та веб-додатків на платформі Java.

7. JUnit 5 - фреймворк для тестування програмного забезпечення в мові програмування Java. Він призначений для автоматизації процесу написання та виконання юніт-тестів, які перевіряють коректність роботи окремих частин програми (юнітів) або цілі програми.

8. Hibernate - це фреймворк для роботи з базами даних в Java, який забезпечує мапування об'єктної моделі даних на реляційну модель бази даних. Використання Hibernate спрощує роботу з базами даних та дозволяє розробникам взаємодіяти з даними в базі даних, використовуючи об'єктно-орієнтований код. А саме реалізує технологію ORM.

ORM (Object-Relational Mapping) — це концепція або технологія, яка використовується в розробці програмного забезпечення для забезпечення взаємодії між об'єктно-орієнтованою моделлю даних в програмному коді та реляційною базою даних. ORM спрощує роботу з базами даних, адаптуючи об'єктно-орієнтовану модель до реляційної моделі бази даних.

9. Java Server Runtime забезпечує виконання Java-додатків на серверному боці та забезпечує надійність, масштабованість та ефективність у вирішенні завдань веб-розробки та інших серверних застосувань.

Були наведені всі основні технології, що були використані в розробці данної системи як для розробки інтерфейсу так і для логіки додатку.

## 2.5 Проектування бізнес-логіки та бізнес-правил

Бізнес-логіка та бізнес-правила реалізується на рівні програмного забезпечення за допомогою взаємодії різних частин додатку такий як сервлети, сервіси, доступ до даних. Для правильної її реалізації ми створимо діаграми, що будуть надавати нам логічний принцип роботи кожного з них.

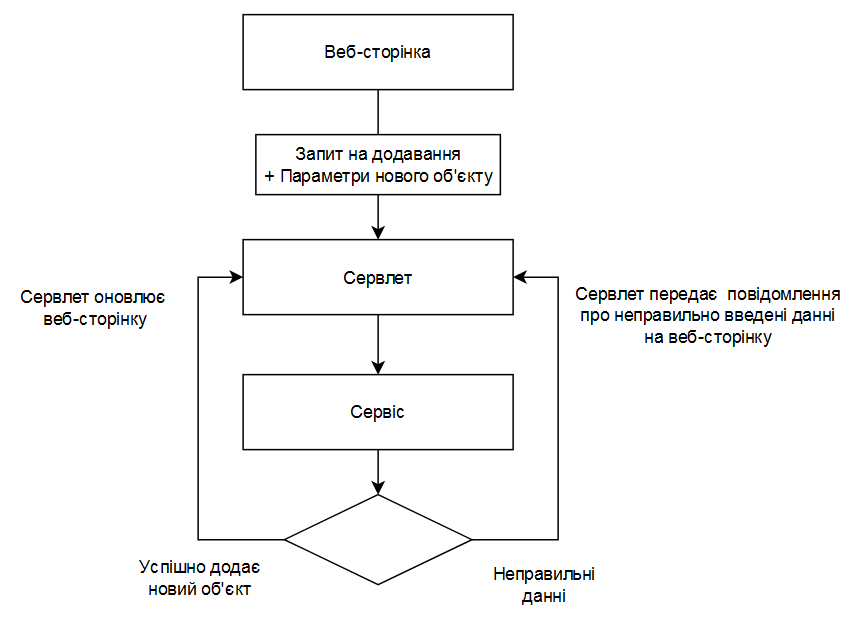
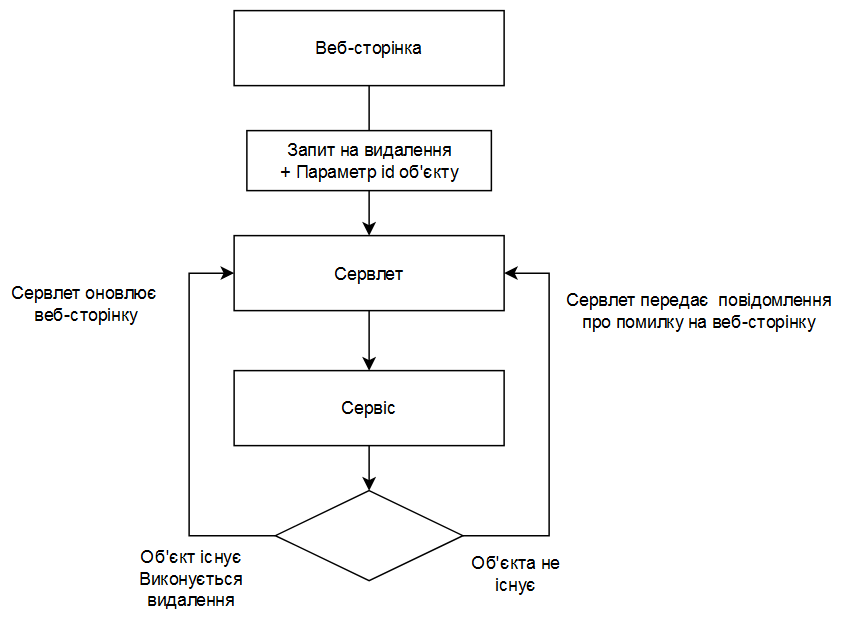
Діаграма для додавання об'єкту рис 2.3 , аналогічний для всіх основних таблиць.

Рисунок 2.3 – Діаграма додавання об'єкту

Діаграма для видалення об'єкту рис 2.4 , аналогічний для всіх основних таблиць.

Рисунок 2.4 – Діаграма видалення об'єкту

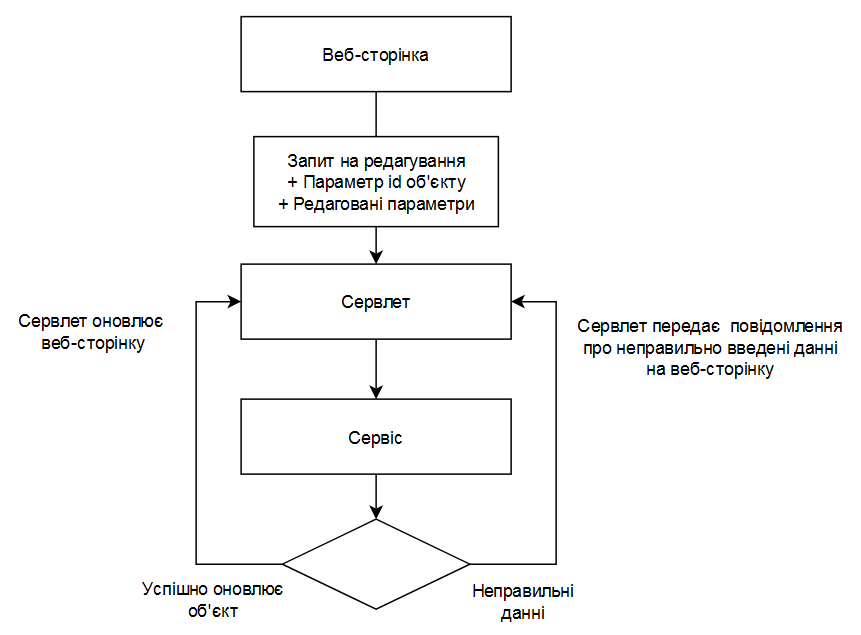
Діаграма для редагування об'єкту рис 2.5 , аналогічний для всіх основних таблиць

Рисунок 2.5 – Діаграма редагування об'єкту

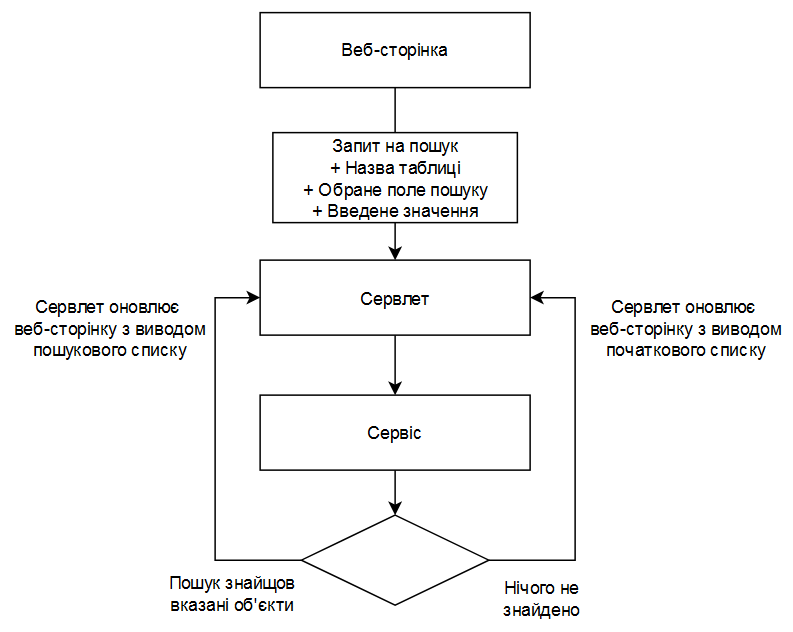
Крім основних дій з базою даних ми також реалізуємо систему пошуку. Діаграма пошуку наведена на рис. 2.6.

Рисунок 2.6 – Діаграма пошуку об’єктів

## 2.6 Проектування шару відображення

Початковий макет головної сторінки до її розробки має вигляд наведений на рисунку 2.7. Фінальний результат наведений на рисунку 2.8. Всі сторінки мають аналогічний дизайн та макет, також мають однакове розміщення кнопок, посилань. Основною відмінністю між сторінками відображення таблиць є поля наявність чи відсутність функцій переходу на пов’язану з об’єктом інформацію.

Також на кожній сторінці присутня навігаційна панель, також в реалізації відображення буде використовуватися JavaScript для спрощення роботи з системою прав доступу до веб-сторінки та створення анімації авторизації користувача.

Загалом на веб-ресурсі є чотири основних види макетів, унікальний макет головної сторінки, макет типу відображення для всіх основних таблиць, макет типу змін для редагування та видалення об’єктів таблиць або авторизації та останній тип для перегляду зв’язаних з даним об’єктом сутностей.

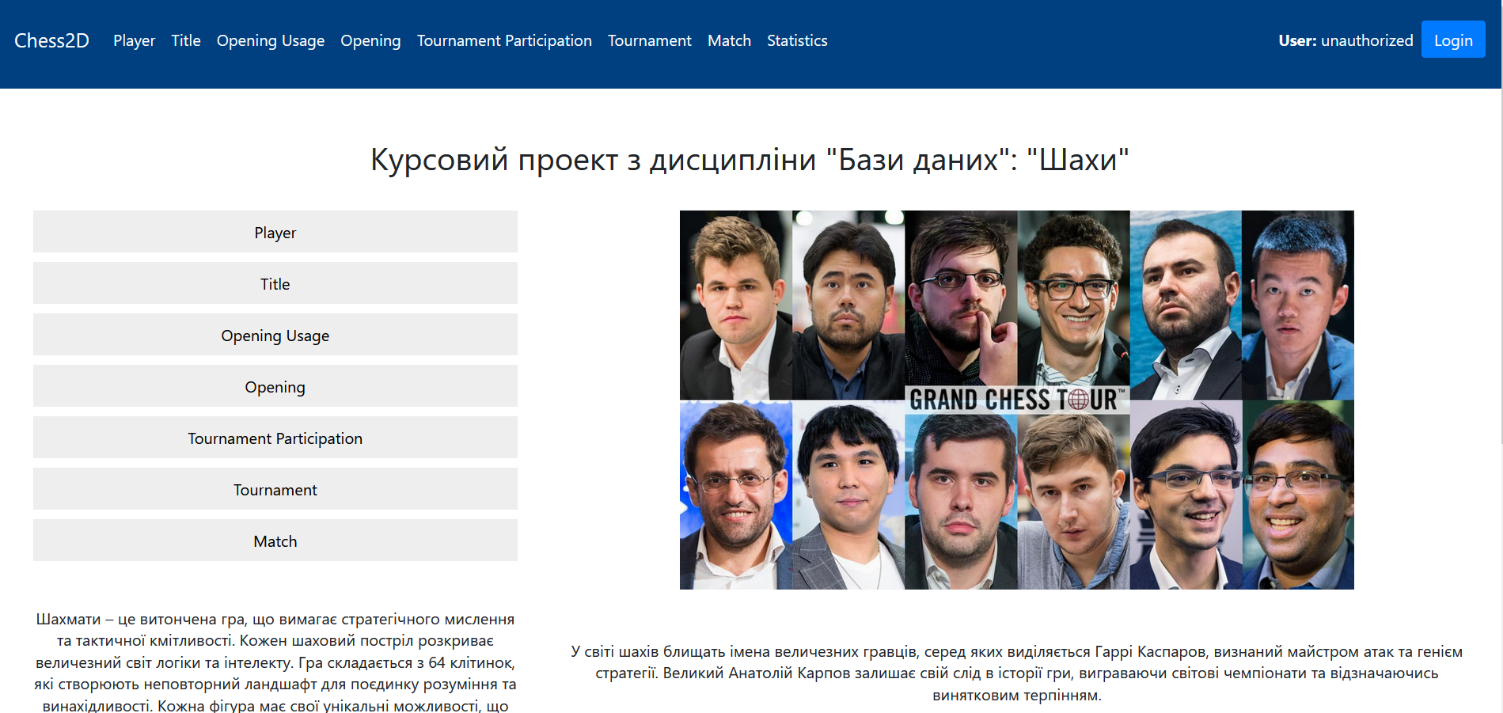
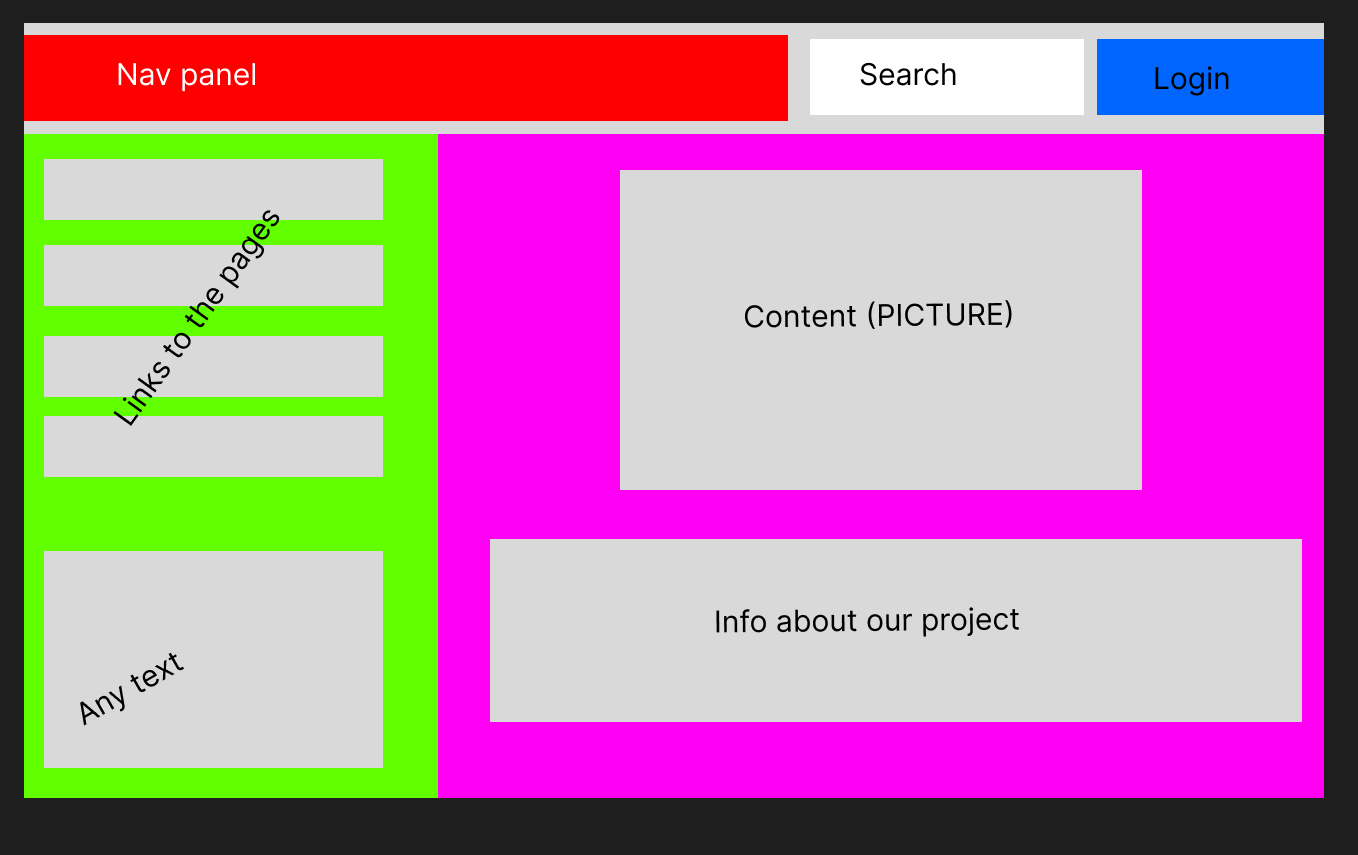
****Рисунок 2.7 – Схематичне зображення плану макету сайту

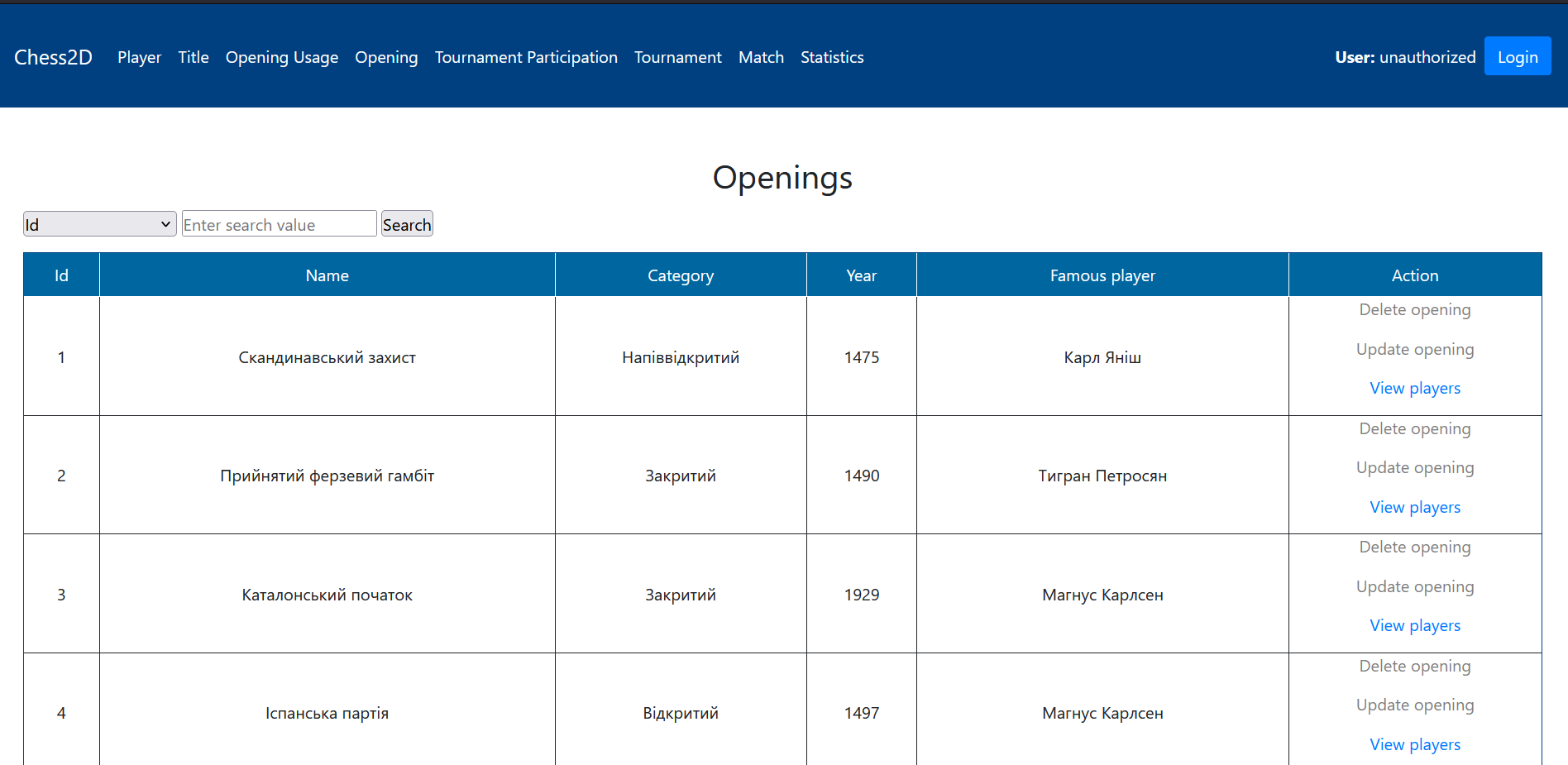
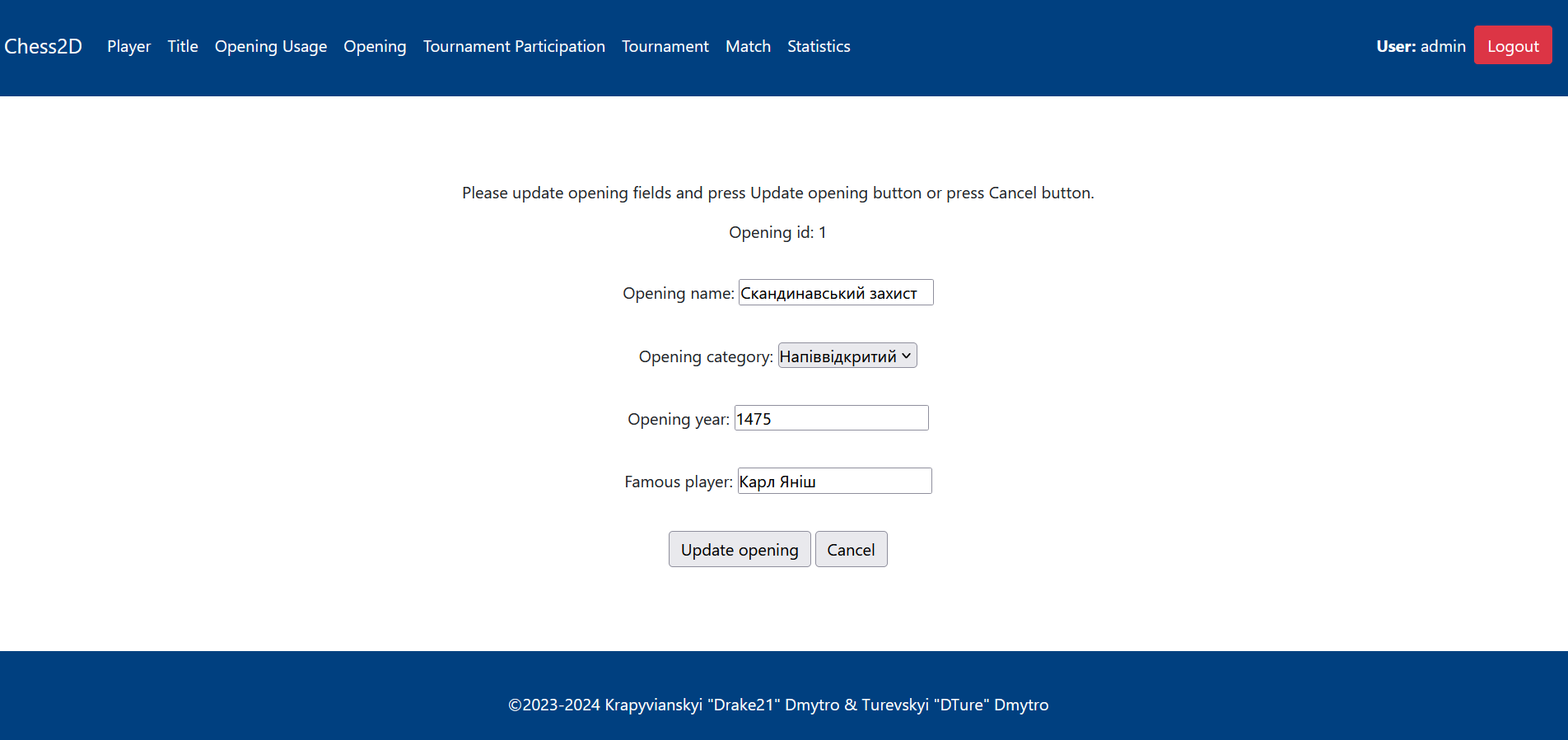
Рисунок 2.8 – Фінальний вигляд головної сторінки

Далі будуть представлені по одному виду кожного типу макету, що був використаний для проектування шару відображення в проекті.

1. Макету відображення. (для відображення вмісту таблиць)

2. Макету змін. (для авторизації, редагування, видалення)

3. Макету зв’язку. (для відображення сутностей зв’язаних с об’єктом)

Рисунок 2.9 – Макет для сторінок типу відображення

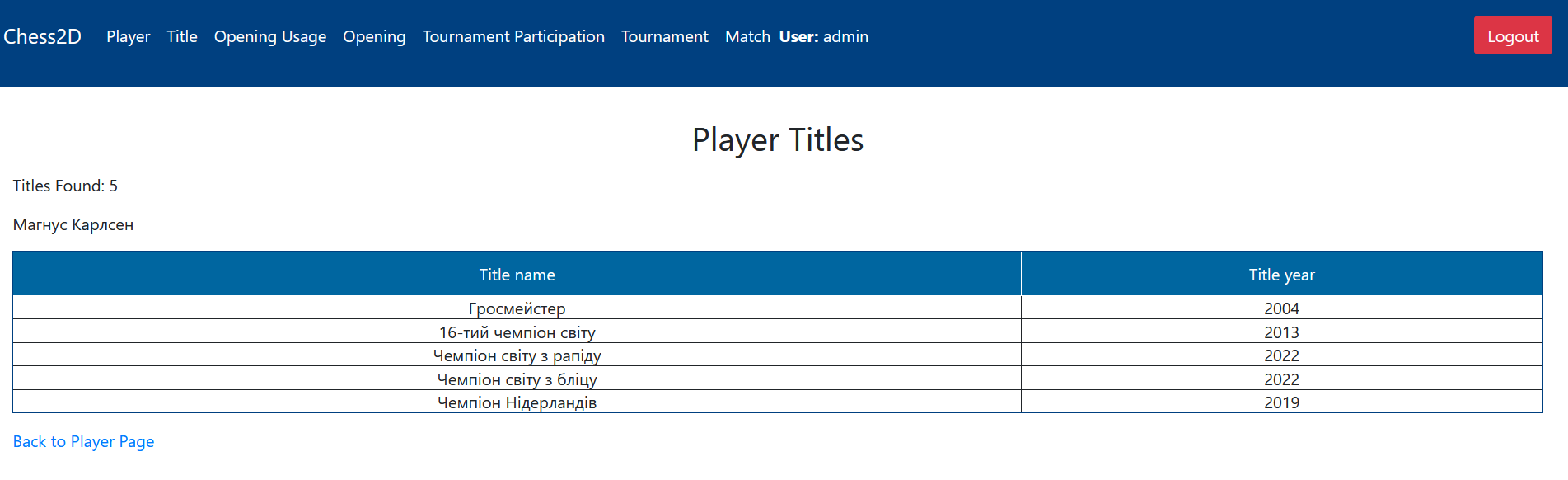
Рисунок 2.10 – Макет для сторінок типу змін

Рисунок 2.11 – Макет для сторінок типу зв’язку

# **3. РОЗРОБКА СИСТЕМИ**

## 3.1 Розробка бази даних системи

### 3.1.1 Розробка фізичної схеми бази даних

На основі логічної схеми бази данних побудовали фізичну модель бази даних (рисунок 3.1).

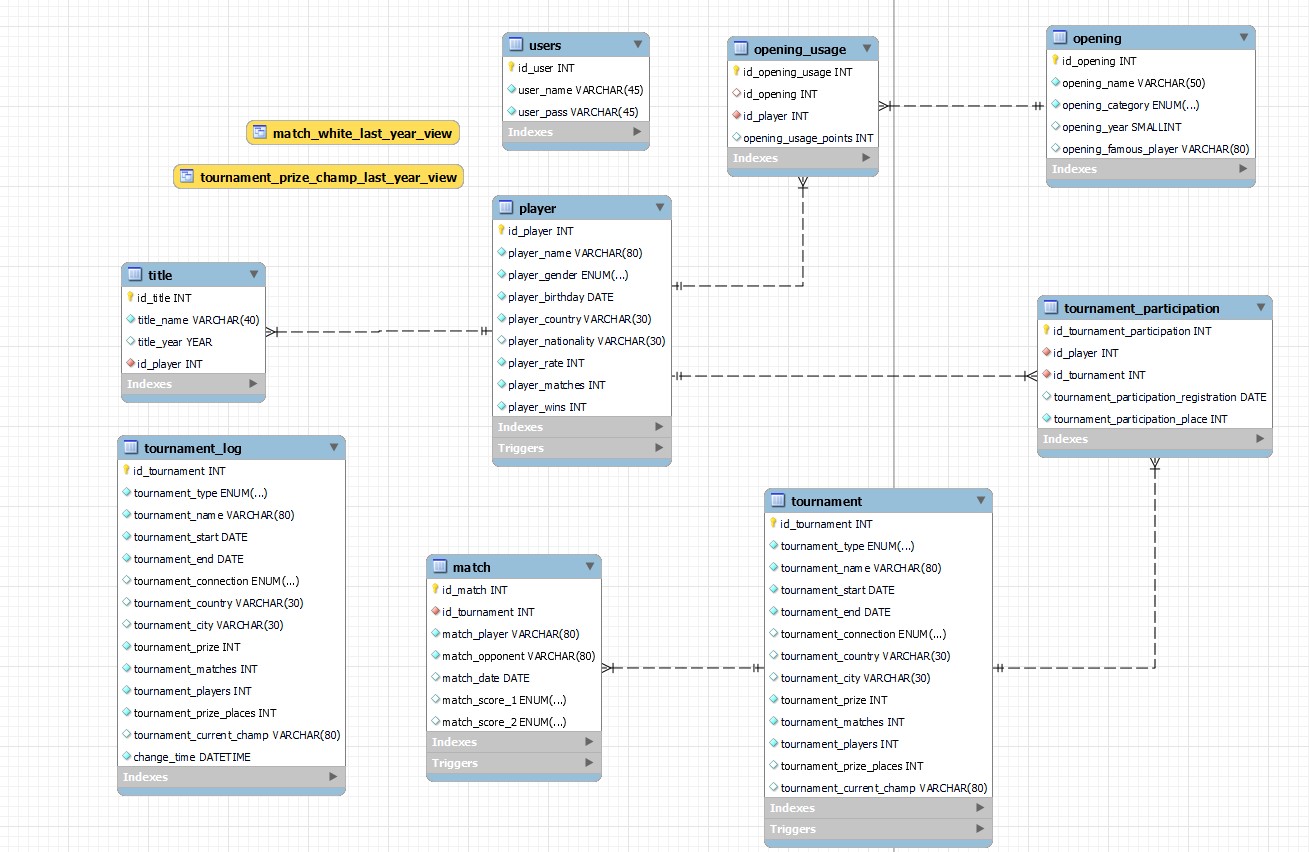


Рисунок 3.1 – Фізична схема бази даних

Перейдемо до детального опису бази даних, створеної в СУБД MySQL.

### 3.1.2 Забезпечення цілісності даних

Обмеження цілісності створенних таблиць наведені на рисунках 3.2 – 3.10

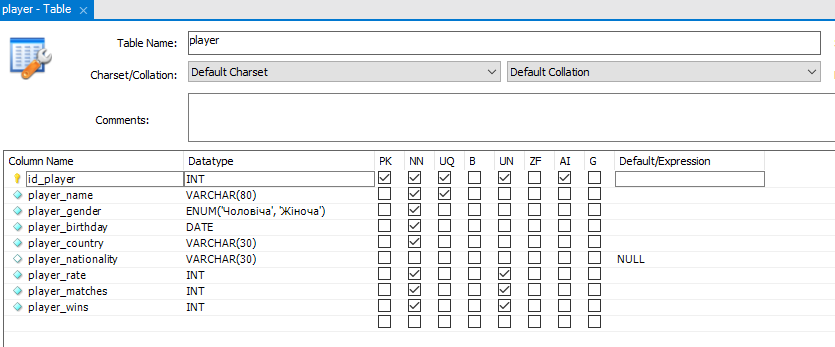


Рисунок 3.2 – Обмеження цілісності таблиці player

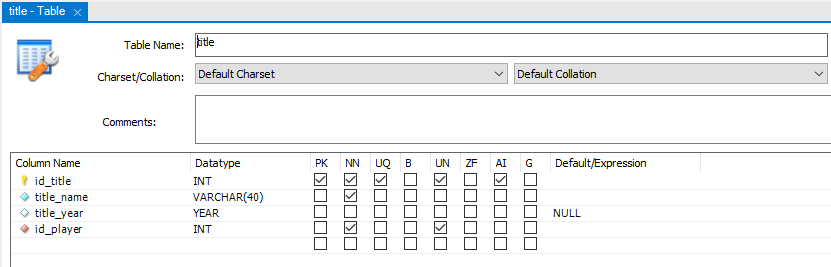
****

Рисунок 3.3 – Обмеження цілісності таблиці title

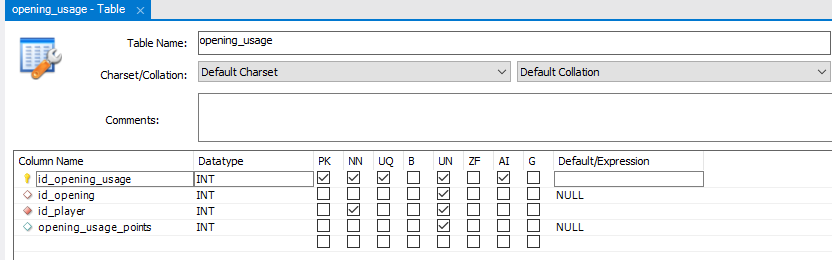
****

Рисунок 3.4 – Обмеження цілісності opening\_usage

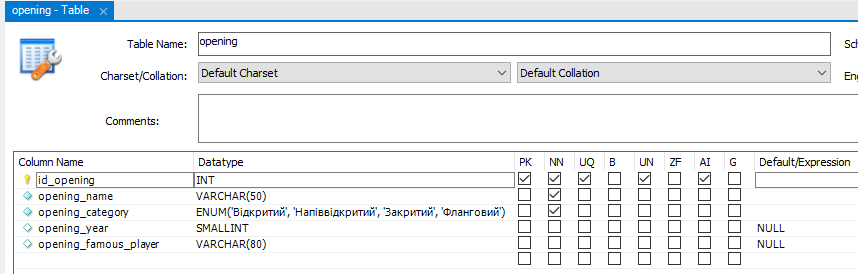
****

Рисунок 3.5 – Обмеження цілісності таблиці opening

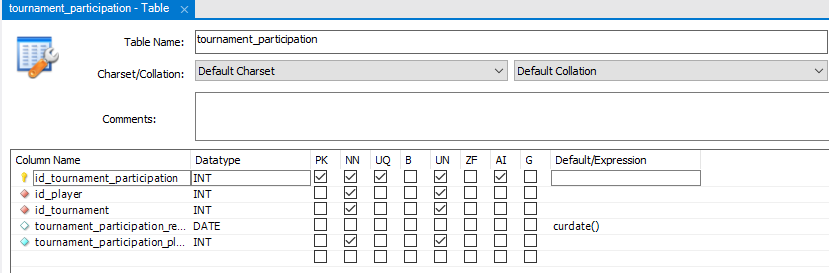
****

Рисунок 3.6 – Обмеження цілісності таблиці tournament\_participation

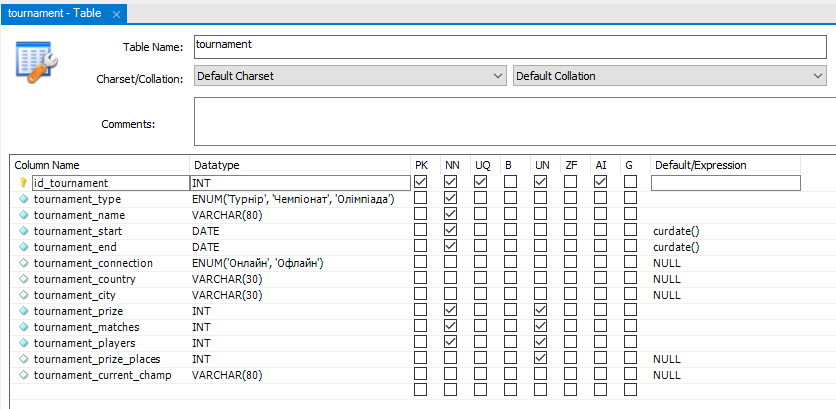
****

Рисунок 3.7 – Обмеження цілісності таблиці tournament

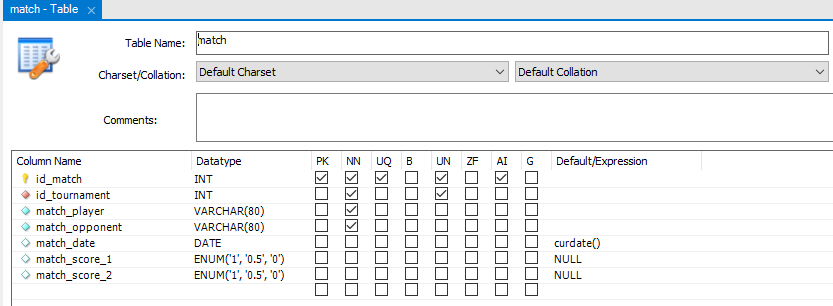
****

Рисунок 3.8 – Обмеження цілісності таблиці match

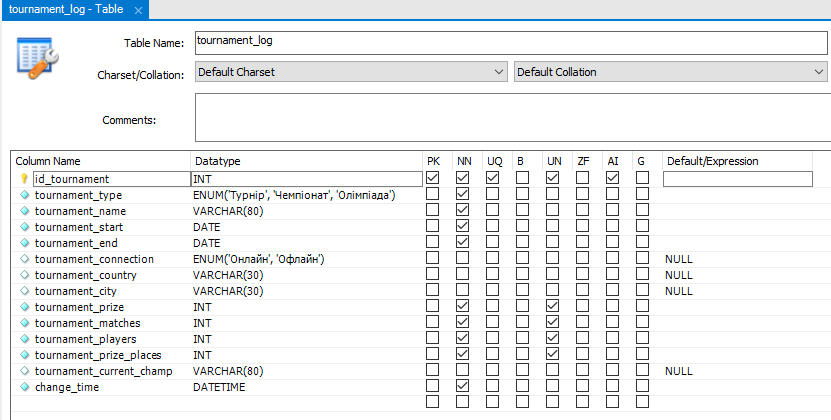
****

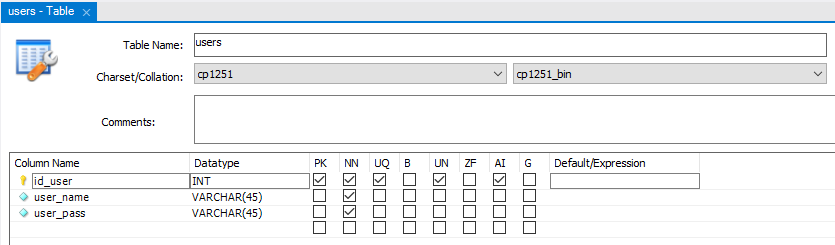
Рисунок 3.9 – Обмеження цілісності таблиці tournament\_log****

Рисунок 3.10 – Обмеження цілісності таблиці users

## 3.2 Реалізація бізнес-логіки зі сторони серверу

Реалізували спроектовані у пункті \_\_ тригери та збережені процедури, опис яких наведено у таблицях х.х-х.х.

Створили процедуру, що зберігається, яка, при додаванні нового запису у таблицю player, буде перевіряти наявність у таблиці запису з таким ім’ям. Якщо запис наявний – з’являється помилка.

Код процедури для додавання нового запису у таблицю player наведено у лістингу 3.1.

Лістинг 1.1 – Запит для створення процедури додавання нового запису у таблицю player

DELIMITER //

CREATE DEFINER=`root`@`localhost`

PROCEDURE insert\_new\_player

(IN id\_player INT, IN new\_player\_name VARCHAR(80),

IN player\_gender ENUM('Чоловіча', 'Жіноча'),

IN player\_birthday DATE, IN player\_country VARCHAR(30),

IN player\_nationality VARCHAR(30), IN player\_rate INT,

IN player\_matches INT, IN player\_wins INT)

BEGIN

DECLARE count INT DEFAULT 0;

SELECT COUNT(\*) INTO count FROM player

WHERE player.player\_name = new\_player\_name;

IF count = 0

THEN INSERT INTO player VALUES

(id\_player, new\_player\_name, player\_gender, player\_birthday,

player\_country, player\_nationality, player\_rate,

player\_matches, player\_wins);

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = "Error: the player what you are

trying to insert has already been inserted";

END IF;

END //

DELIMITER ;

Створили процедуру, що зберігається, яка сортує записи в таблиці player за рейтингом та, якщо декілька записів мають однакові значення того поля, за кількістю виграних матчів.

Код процедури для сортування записів в таблиці player за рейтингом та за кількістю виграних матчів наведено у лістингу 3.2.

Лістинг 3.2 – Запит для створення процедури сортування записів в таблиці player за рейтингом та за кількістю виграних матчів

DELIMITER //

CREATE DEFINER=`root`@`localhost`

PROCEDURE sort\_players\_by\_rating()

BEGIN

SELECT \* FROM player ORDER BY player\_rate DESC,

player\_wins DESC;

END //

DELIMITER ;

Створили процедуру, що зберігається, яка підраховує загальну суму виграну відповідним шахістом у турнірах. Якщо шахіст не виграв жодного турніру - з’являється помилка.

Код процедури для підрахування загальної суми виграної відповідним шахістом у турнірах наведено у лістингу 3.3.

Лістинг 3.3 – Запит для створення процедури підрахування загальної суми виграної відповідним шахістом у турнірах

DELIMITER //

CREATE DEFINER=`root`@`localhost` PROCEDURE

sum\_of\_prizes\_won\_by\_player

(IN search\_player\_name VARCHAR(80),

OUT sum\_of\_prizes BIGINT)

BEGIN

DECLARE count INT DEFAULT 0;

SELECT COUNT(\*) INTO count FROM tournament

WHERE tournament\_current\_champ = search\_player\_name;

IF count <> 0

THEN SELECT SUM(tournament\_prize) INTO sum\_of\_prizes

FROM tournament WHERE tournament\_current\_champ

= search\_player\_name;

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = "Error: there is no such player";

END IF;

END //

DELIMITER ;

Створили тригер, який при додаванні записів у таблицю tournament додає їх до таблиці tournament\_log для створення механізму логування.

Код тригеру для імплементації механізму логування таблиці tournament\_log наведено у лістингу 3.4.

Лістинг 3.4 – Запит для створення тригеру імплементації механізму логування таблиці tournament

DELIMITER //

CREATE DEFINER=`root`@`localhost`

TRIGGER tournament\_insert\_logging

AFTER INSERT

ON chess.tournament FOR EACH ROW

BEGIN

INSERT INTO tournament\_log VALUES

(NEW.id\_tournament, NEW.tournament\_type,

NEW.tournament\_name, NEW.tournament\_start,

NEW.tournament\_end, NEW.tournament\_connection,

NEW.tournament\_country, NEW.tournament\_city,

NEW.tournament\_prize, NEW.tournament\_matches,

NEW.tournament\_players, NEW.tournament\_prize\_places,

NEW.tournament\_current\_champ, NOW());

END //

DELIMITER ;

Створили тригер, який перед додаванням записів у таблицю player перевіряє виключну ситуацію, коли шахіст має більше виграних партій, ніж зіграних.

Код тригеру перевірки виключної ситуації при додаванні нового шахіста наведено у лістингу 3.5.

Лістинг 3.5 – Запит для створення тригеру перевірки виключної ситуації при додаванні нового шахіста

DELIMITER //

CREATE DEFINER=`root`@`localhost`

TRIGGER player\_matches\_trigger

BEFORE INSERT

ON chess.player FOR EACH ROW

BEGIN

IF NEW.player\_matches < NEW.player\_wins

THEN SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT =

'Error: invalid values of matches and wins.';

END IF;

END //

DELIMITER ;

Створили тригер, який перед додаванням записів у таблицю match перевіряє виключну ситуацію, коли значення отриманих рахунків кожного з супротивників не є дійсними.

Код тригеру перевірки виключної ситуації при додаванні нового матчу наведено у лістингу 3.6.

Лістинг 3.6 – Запит для створення тригеру перевірки виключної ситуації при додаванні нового матчу

DELIMITER //

CREATE DEFINER=`root`@`localhost`

TRIGGER match\_points\_trigger

BEFORE INSERT

ON chess.match FOR EACH ROW

BEGIN

IF

(NEW.match\_score\_1 = '1' AND NEW.match\_score\_2 = '1')

OR

(NEW.match\_score\_1 = '0' AND NEW.match\_score\_2 = '0')

OR

(NEW.match\_score\_1 = '0.5' AND NEW.match\_score\_2 = '1')

OR

(NEW.match\_score\_1 = '1' AND NEW.match\_score\_2 = '0.5')

OR

(NEW.match\_score\_1 = '0.5' AND NEW.match\_score\_2 = '0')

OR

(NEW.match\_score\_1 = '0' AND NEW.match\_score\_2 = '0.5')

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Error: invalid values of match points.';

END IF;

END //

DELIMITER ;

## 3.3 Розробка модулів системи

Наша система має 4 модулі, а саме:

1. Модуль шару доступу до даних
2. Модуль шару бізнес-логіки
3. Модуль шару сервісів
4. Модуль шару відображення

### 3.3.1 Розробка модулів шару бізнес-логіки

На цьому рівні за допомогою сервлетів відбувається фільтрація даних, введенних користувачем, забезпечення виконання основних правил бізнес-логіки та перевірка рівня доступу користувача, обробка повідомлень від шару доступу до даних і перевірка даних, отриманих від шару сервісів та клієнта.

Код сервлету для сутності Player наведено у лістингу 3.7.

Лістинг 3.7 – Код сервлету для сутності Player

package servlet;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import dao.HibernateDAOChess;

import domain.Player;

import service.PlayerService;

public class PlayerServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private PlayerService playerService;

@Override

public void init() {

playerService = new PlayerService();

}

@Override

protected void doPost(HttpServletRequest req,

HttpServletResponse resp)

throws ServletException, IOException {

req.setCharacterEncoding("UTF-8");

String actionType = req.getParameter("actiontype");

if ("addplayer".equalsIgnoreCase(actionType)) {

try {

req.setCharacterEncoding("UTF-8");

String player\_name =

req.getParameter("player\_name");

String player\_gender =

req.getParameter("player\_gender");

String player\_birthday =

req.getParameter("player\_birthday");

String player\_country =

req.getParameter("player\_country");

String player\_nationality =

req.getParameter("player\_nationality");

String player\_rate =

req.getParameter("player\_rate");

String player\_matches =

req.getParameter("player\_matches");

String player\_wins =

req.getParameter("player\_wins");

playerService.addPlayer(player\_name, player\_gender,

player\_birthday, player\_country, player\_nationality,

player\_rate, player\_matches, player\_wins);

req.getRequestDispatcher("player.jsp").forward(req, resp);

} catch (Exception e) {

req.setAttribute("error", e.getMessage());

req.getRequestDispatcher("player.jsp").forward(req, resp);

}

} else if ("deleteplayer".equalsIgnoreCase(actionType)) {

String player\_id = req.getParameter("player\_id");

playerService.deletePlayer(player\_id);

req.getRequestDispatcher("player.jsp")

.forward(req, resp);

} else if ("updateplayer".equalsIgnoreCase(actionType)) {

try {

String player\_id =

req.getParameter("player\_id");

String player\_name =

req.getParameter("player\_name");

String player\_gender =

req.getParameter("player\_gender");

String player\_birthday =

req.getParameter("player\_birthday");

String player\_country =

req.getParameter("player\_country");

String player\_nationality =

req.getParameter("player\_nationality");

String player\_rate =

req.getParameter("player\_rate");

String player\_matches =

req.getParameter("player\_matches");

String player\_wins =

req.getParameter("player\_wins");

playerService.updatePlayer(player\_id, player\_name,

player\_gender, player\_birthday, player\_country,

player\_nationality, player\_rate, player\_matches,

player\_wins);

req.getRequestDispatcher("player.jsp").forward(req, resp);

} catch (Exception e) {

req.setAttribute("error", e.getMessage());

req.getRequestDispatcher("updateplayer.jsp")

.forward(req, resp);

}

} else if ("searchplayer".equalsIgnoreCase(actionType)) {

String field = req.getParameter("field");

String value = req.getParameter("value");

List<Player> searchP =

HibernateDAOChess.getInstance()

.getPlayerDAO().searchPlayers(field, value);

req.setAttribute("searchResults", searchP);

req.getRequestDispatcher("player.jsp")

.forward(req, resp);

}

}

}

Сервлети для всіх інших сутностей є подібними, тому не будемо наводити код кожного з них.

### 3.3.2 Розробка модулів шару доступу до даних

Шар доступу до даних представляє собою набір класів, що реалізують запити до БД, написані на мові SQL за посередництвом Hibernate. У цьому шарі надсилаються запити та обробляються отримані результати.

Код класу, який описує сутність Player, наведено у лістингу 3.8.

Лістинг 3.8 – Код класу-сутності Player

package domain;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Convert;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import javax.persistence.Transient;

import domain.Enums.PlayerGender;

import settings.StringToPlayerGenderConverter;

@Entity

@Table(name = "player")

public class Player {

public Player() {

}

public Player(String name, String gender, String birthday,

String country, String nationality, int rate,

int matches, int wins) {

this.name = name;

this.gender = PlayerGender.getGenderByInput(gender);

this.birthday = birthday;

this.country = country;

this.nationality = nationality;

this.rate = rate;

this.matches = matches;

this.wins = wins;

}

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_player", unique = true, nullable = false)

private Long id;

@Column(name = "player\_name", nullable = false)

private String name;

@Column(name = "player\_gender", nullable = false)

@Convert(converter = StringToPlayerGenderConverter.class)

private PlayerGender gender;

@Column(name = "player\_birthday", nullable = false)

private String birthday;

@Column(name = "player\_country", nullable = false)

private String country;

@Column(name = "player\_nationality", nullable = true)

private String nationality;

@Column(name = "player\_rate", nullable = false)

private int rate;

@Column(name = "player\_matches", nullable = false)

private int matches;

@Column(name = "player\_wins", nullable = false)

private int wins;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "player")

private List<Title> titles = new ArrayList<>();

@OneToMany(cascade = CascadeType.ALL, mappedBy = "player")

private List<OpeningUsage> openingUsage = new ArrayList<>();

@OneToMany(cascade = CascadeType.ALL, mappedBy = "player")

private List<TournamentPart> tournamentPart

= new ArrayList<>();

@Transient

private int titleNumber;

@Transient

private int openingNumber;

@Transient

private int tournamentNumber;

@Transient

private String registration;

@Transient

private int openingUsagePoints;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getGender() {

return gender.getGender();

}

public void setGender(String gender) {

this.gender = PlayerGender.getGenderByInput(gender);

}

public String getBirthday() {

return birthday;

}

public void setBirthday(String birthday) {

this.birthday = birthday;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

public String getNationality() {

return nationality;

}

public void setNationality(String nationality) {

this.nationality = nationality;

}

public int getRate() {

return rate;

}

public void setRate(int rate) {

this.rate = rate;

}

public int getMatches() {

return matches;

}

public void setMatches(int matches) {

this.matches = matches;

}

public int getWins() {

return wins;

}

public void setWins(int wins) {

this.wins = wins;

}

public List<Title> getTitles() {

return titles;

}

public void setTitles(List<Title> titles) {

this.titles = titles;

}

public List<OpeningUsage> getOpeningUsage() {

return openingUsage;

}

public void setOpeningUsage(List<OpeningUsage> openingUsage) {

this.openingUsage = openingUsage;

}

public List<TournamentPart> getTournamentPart() {

return tournamentPart;

}

public void setTournamentPart(List<TournamentPart>

tournamentPart) {

this.tournamentPart = tournamentPart;

}

public int getTitleNumber() {

return titleNumber;

}

public void setTitleNumber(int titleNumber) {

this.titleNumber = titleNumber;

}

public int getOpeningNumber() {

return openingNumber;

}

public void setOpeningNumber(int openingNumber) {

this.openingNumber = openingNumber;

}

public int getTournamentNumber() {

return tournamentNumber;

}

public void setTournamentNumber(int tournamentNumber) {

this.tournamentNumber = tournamentNumber;

}

public String getRegistration() {

return registration;

}

public void setRegistration(String registration) {

this.registration = registration;

}

public int getOpeningUsagePoints() {

return openingUsagePoints;

}

public void setOpeningUsagePoints(int openingUsagePoints) {

this.openingUsagePoints = openingUsagePoints;

}

@Override

public String toString() {

return "Player [id=" + id + ", name=" + name + ",

gender=" + getGender() + ", birthday=" + birthday + ",

country=" + country + ",nationality=" + nationality +

", rate=" + rate + ", matches=" + matches + ", wins="

+ wins + ", titles.size()=" + titles.size() + ",

openingUsage.size()=" + openingUsage.size() + ",

tournamentParticipation.size()=" + tournamentPart.size()+ "]\n\n";

}

}

Класи, які описують структуру інших сутностей для всіх інших сутностей, є подібними, тому не будемо наводити код кожного з них.

Код головного класу додатку HibernateDAOChess наведено у лістингу 3.9.

Лістинг 3.9 – Код класу HibernateDAOChess

package dao;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.hibernate.cfg.Environment;

import org.hibernate.service.ServiceRegistry;

import domain.\*;

public class HibernateDAOChess {

private static HibernateDAOChess instance;

private PlayerDAO playerDAO;

private UserDAO userDAO;

private TournamentDAO tournamentDAO;

private OpeningDAO openingDAO;

private TitleDAO titleDAO;

private MatchDAO matchDAO;

private OpeningUsageDAO openingUsageDAO;

private TournamentPartDAO tournamentPartDAO;

private TournamentLogDAO tournamentLogDAO;

private GlobalStatisticsDAO globalStatisticsDAO;

private Session session;

public static HibernateDAOChess getInstance() {

if (null == instance) {

instance = new HibernateDAOChess();

}

return instance;

}

public Session getSession() {

if (null == session) {

Configuration configuration

= new Configuration();

configuration.setProperty(Environment.DRIVER,

"com.mysql.cj.jdbc.Driver");

configuration.setProperty(Environment.URL,

"jdbc:mysql://localhost:3306/chess");

configuration.setProperty(Environment.USER,

"root");

configuration.setProperty(Environment.PASS,

"PolishDream\_04");

configuration.setProperty(Environment.DIALECT,

"org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty

(Environment.HBM2DDL\_AUTO, "none");

configuration.setProperty(Environment.SHOW\_SQL,

"true");

configuration.addAnnotatedClass(User.class)

.addAnnotatedClass(Player.class)

.addAnnotatedClass(Title.class)

.addAnnotatedClass(Match.class)

.addAnnotatedClass(Opening.class)

.addAnnotatedClass(Tournament.class)

.addAnnotatedClass(OpeningUsage.class)

.addAnnotatedClass(TournamentPart.class)

.addAnnotatedClass(TournamentLog.class) .addAnnotatedClass(GlobalStatistics.class);

StandardServiceRegistryBuilder serviceRegistryBuilder

= new StandardServiceRegistryBuilder();

serviceRegistryBuilder.applySettings

(configuration.getProperties());

ServiceRegistry serviceRegistry =

serviceRegistryBuilder.build();

SessionFactory sessionFactory =

configuration.buildSessionFactory(serviceRegistry);

session = sessionFactory.openSession();

}

return session;

}

public UserDAO getUserDAO() {

if (null == userDAO) {

userDAO = new UserDAO(getSession());

}

return userDAO;

}

public PlayerDAO getPlayerDAO() {

if (null == playerDAO) {

playerDAO = new PlayerDAO(getSession());

}

return playerDAO;

}

public TitleDAO getTitleDAO() {

if (null == titleDAO) {

titleDAO = new TitleDAO(getSession());

}

return titleDAO;

}

public MatchDAO getMatchDAO() {

if (null == matchDAO) {

matchDAO = new MatchDAO(getSession());

}

return matchDAO;

}

public OpeningUsageDAO getOpeningUsageDAO() {

if (null == openingUsageDAO) {

openingUsageDAO

= new OpeningUsageDAO(getSession());

}

return openingUsageDAO;

}

public OpeningDAO getOpeningDAO() {

if (null == openingDAO) {

openingDAO = new OpeningDAO(getSession());

}

return openingDAO;

}

public TournamentPartDAO getTournamentPartDAO() {

if (null == tournamentPartDAO) {

tournamentPartDAO

= new TournamentPartDAO(getSession());

}

return tournamentPartDAO;

}

public TournamentDAO getTournamentDAO() {

if (null == tournamentDAO) {

tournamentDAO

= new TournamentDAO(getSession());

}

return tournamentDAO;

}

public TournamentLogDAO getTournamentLogDAO() {

if (null == tournamentLogDAO) {

tournamentLogDAO

= new TournamentLogDAO(getSession());

}

return tournamentLogDAO;

}

public GlobalStatisticsDAO getGlobalStatisticsDAO() {

if (null == globalStatisticsDAO) {

globalStatisticsDAO

= new GlobalStatisticsDAO(getSession());

}

return globalStatisticsDAO;

}

public void closeSession() {

getSession().close();

}

}

Код класу, який відповідає за SQL-запити для всіх інших сутностей, PlayerDAO наведено у лістингу 3.11.

Лістинг 3.11 – Код класу PlayerDAO

package dao;

import java.math.BigInteger;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import org.hibernate.Criteria;

import org.hibernate.SQLQuery;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.criterion.Restrictions;

import domain.Player;

public class PlayerDAO {

private Session session;

public PlayerDAO(Session session) {

this.session = session;

}

public Player createPlayer(Player player) {

Transaction transaction = session.beginTransaction();

session.saveOrUpdate(player);

transaction.commit();

return player;

}

public Player updatePlayer(Player player) {

Transaction transaction = session.beginTransaction();

session.merge(player);

transaction.commit();

return player;

}

public void deletePlayer(Player player) {

Transaction transaction = session.beginTransaction();

session.delete(player);

transaction.commit();

}

public void deletePlayerById(Long playerId) {

Player player = (Player) session.get

(Player.class, playerId);

deletePlayer(player);

}

public List<Player> getAllPlayers() {

SQLQuery query = session.createSQLQuery

("select \* from player").addEntity(Player.class);

List<Player> playerList = query.list();

return playerList;

}

public List<Player> getPlayersByName(String name) {

Criteria criteria = session.createCriteria(Player.class)

.add(Restrictions.eq("player\_name", name));

return criteria.list();

}

public Player getPlayerById(Long idPlayer) {

Player player = (Player) session.get

(Player.class, idPlayer);

return player;

}

public List<Player> getAllPlayersWithAllAdditionalInfo() {

SQLQuery query = session.createSQLQuery("SELECT p.\*, "

+ "(SELECT COUNT(\*) FROM title t WHERE t.id\_player =

p.id\_player) as title\_count, "

+ "(SELECT COUNT(\*) FROM opening\_usage ou WHERE ou.id\_player =

p.id\_player) as opening\_count, "

+ "(SELECT COUNT(DISTINCT tp.id\_tournament) FROM

tournament\_participation tp WHERE tp.id\_player = p.id\_player)

as tournament\_count "

+ "FROM player p").addEntity(Player.class) .addScalar("title\_count")

.addScalar("opening\_count")

.addScalar("tournament\_count");

List<Object[]> results = query.list();

List<Player> playerList = new ArrayList<>();

for (Object[] result : results) {

Player player = (Player) result[0];

BigInteger titleNumber = (BigInteger) result[1];

BigInteger openingNumber = (BigInteger) result[2];

BigInteger tournamentNumber = (BigInteger) result[3];

player.setTitleNumber(titleNumber.intValue());

player.setOpeningNumber(openingNumber.intValue());

player.setTournamentNumber(tournamentNumber.intValue());

playerList.add(player);

}

return playerList;

}

public List<Player> getPlayersByTournamentId(Long tournamentId) {

SQLQuery query = (SQLQuery) session.createSQLQuery(

"SELECT p.\*, tp.tournament\_participation\_registration "

+ "FROM player p "

+ "INNER JOIN tournament\_participation tp ON p.id\_player = tp.id\_player "

+ "WHERE tp.id\_tournament = :tournamentId "

+ "ORDER BY p.id\_player ASC")

.addEntity(Player.class)

.addScalar("tournament\_participation\_registration")

.setParameter("tournamentId", tournamentId);

List<Object[]> results = query.list();

List<Player> playerList = new ArrayList<>();

for (Object[] result : results) {

Player player = (Player) result[0];

Date registration = (Date) result[1];

SimpleDateFormat dateFormat

= new SimpleDateFormat("yyyy-MM-dd");

String strRegistration

= dateFormat.format(registration);

player.setRegistration(strRegistration);

playerList.add(player);

}

return playerList;

}

public List<Player> getPlayersByOpeningId(Long openingId) {

SQLQuery query = (SQLQuery) session

.createSQLQuery

("SELECT p.\*, ou.opening\_usage\_points " + "FROM player p "

+ "INNER JOIN opening\_usage ou ON p.id\_player = ou.id\_player "

+ "WHERE ou.id\_opening = :openingId "

+ "ORDER BY p.id\_player ASC")

.addEntity(Player.class).addScalar("opening\_usage\_points")

.setParameter("openingId", openingId);

List<Object[]> results = query.list();

List<Player> playerList = new ArrayList<>();

for (Object[] result : results) {

Player player = (Player) result[0];

int openingUsagePoints = (Integer) result[1];

player.setOpeningUsagePoints(openingUsagePoints);

playerList.add(player);

}

return playerList;

}

public List<Player> searchPlayers(String field, String value) {

String queryString = "SELECT p.\*, "

+ "(SELECT COUNT(\*) FROM title t WHERE t.id\_player = p.id\_player)

as title\_count, "

+ "(SELECT COUNT(\*) FROM opening\_usage ou WHERE

ou.id\_player = p.id\_player) as opening\_count, "

+ "(SELECT COUNT(DISTINCT tp.id\_tournament) FROM

tournament\_participation tp WHERE tp.id\_player = p.id\_player) as

tournament\_count " + "FROM player p " + "WHERE p." + field + " = :value";

SQLQuery query = (SQLQuery)session.createSQLQuery

(queryString).addEntity(Player.class).addScalar("title\_count")

.addScalar("opening\_count").addScalar("tournament\_count")

.setParameter("value", value);

List<Object[]> results = query.list();

List<Player> playerList = new ArrayList<>();

for (Object[] result : results) {

Player player = (Player) result[0];

BigInteger titleNumber = (BigInteger) result[1];

BigInteger openingNumber = (BigInteger) result[2];

BigInteger tournamentNumber = (BigInteger) result[3];

player.setTitleNumber(titleNumber.intValue());

player.setOpeningNumber(openingNumber.intValue());

player.setTournamentNumber(tournamentNumber.intValue());

playerList.add(player);

}

return playerList;

}

}

Класи, які відповідають за SQL-запити для всіх інших сутностей, є подібними, тому не будемо наводити код кожного з них.

### 3.3.3 Розробка модулів шару сервісів

Шар відображення представлений у вигляді набору JSP-сторінок та додаткового класу для коректного відображення даних.

Для кожної сутності була створена сторінка додавання, редагування, а також перегляду усіх записів з можливістю видалення даних. У деяких сутностей є сторінки зі зв’язаними записами з інших таблиць.

Код класу, який відповідає за коректне відображення даних, PlayerListDatabean наведено у лістингу 3.12.

Лістинг 3.12 – Код класу PlayerListDatabean

package databean;

import java.util.List;

import dao.HibernateDAOChess;

import domain.Player;

public class PlayerListDatabean {

private List<Player> playerList;

public PlayerListDatabean() {

playerList = HibernateDAOChess.getInstance()

.getPlayerDAO().getAllPlayersWithAllAdditionalInfo();

}

public List<Player> getPlayerList() {

return playerList;

}

public void setPlayerList(List<Player> playerList) {

this.playerList = playerList;

}

}

Класи, які відповідають за коректне відображення даних для всіх інших сутностей, є подібними, тому не будемо наводити код кожного з них.

Вигляд набору JSP-сторінок для сутності Player наведено на рисунках 3.11-3.13.

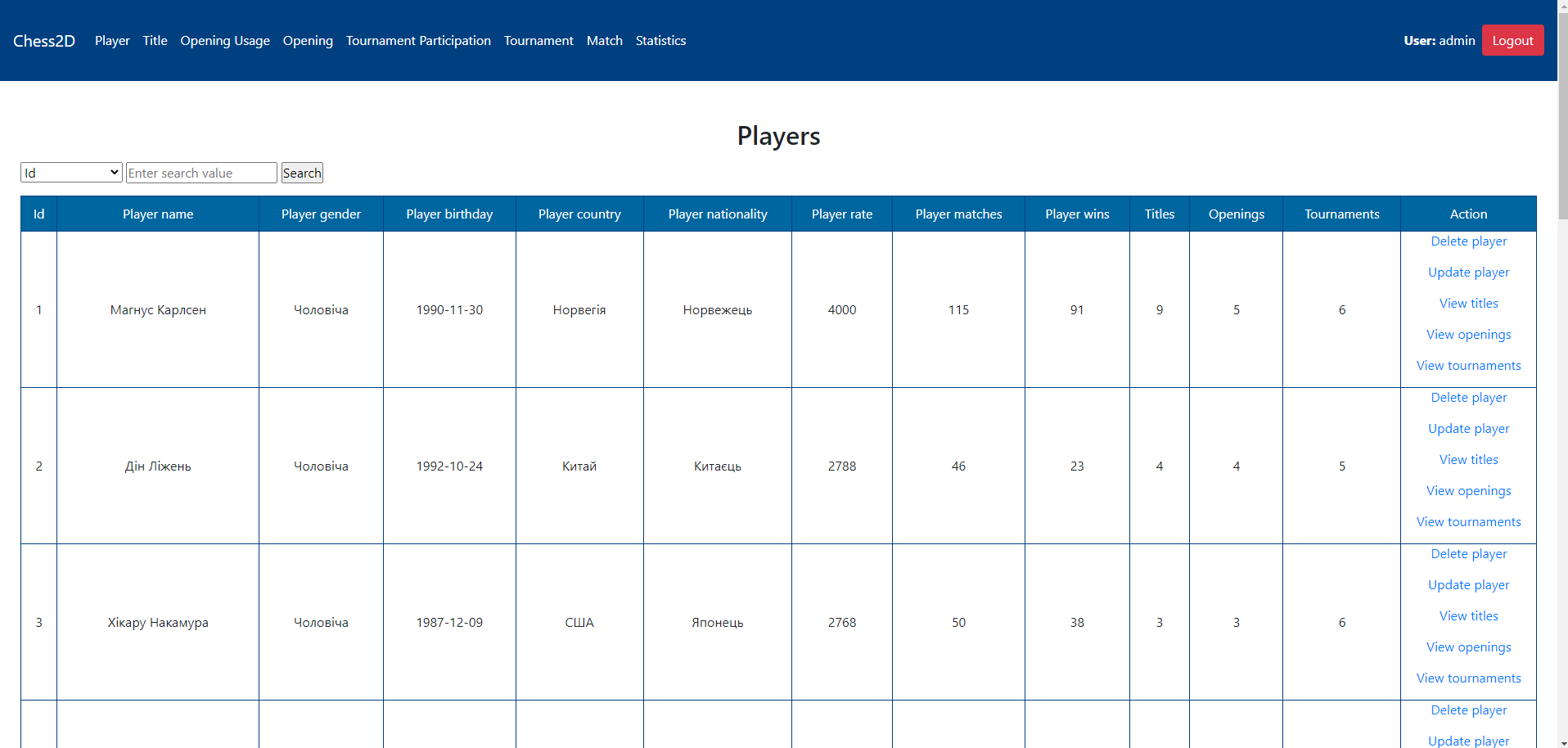


Рисунок 3.11 – Головна сторінка сутності Player

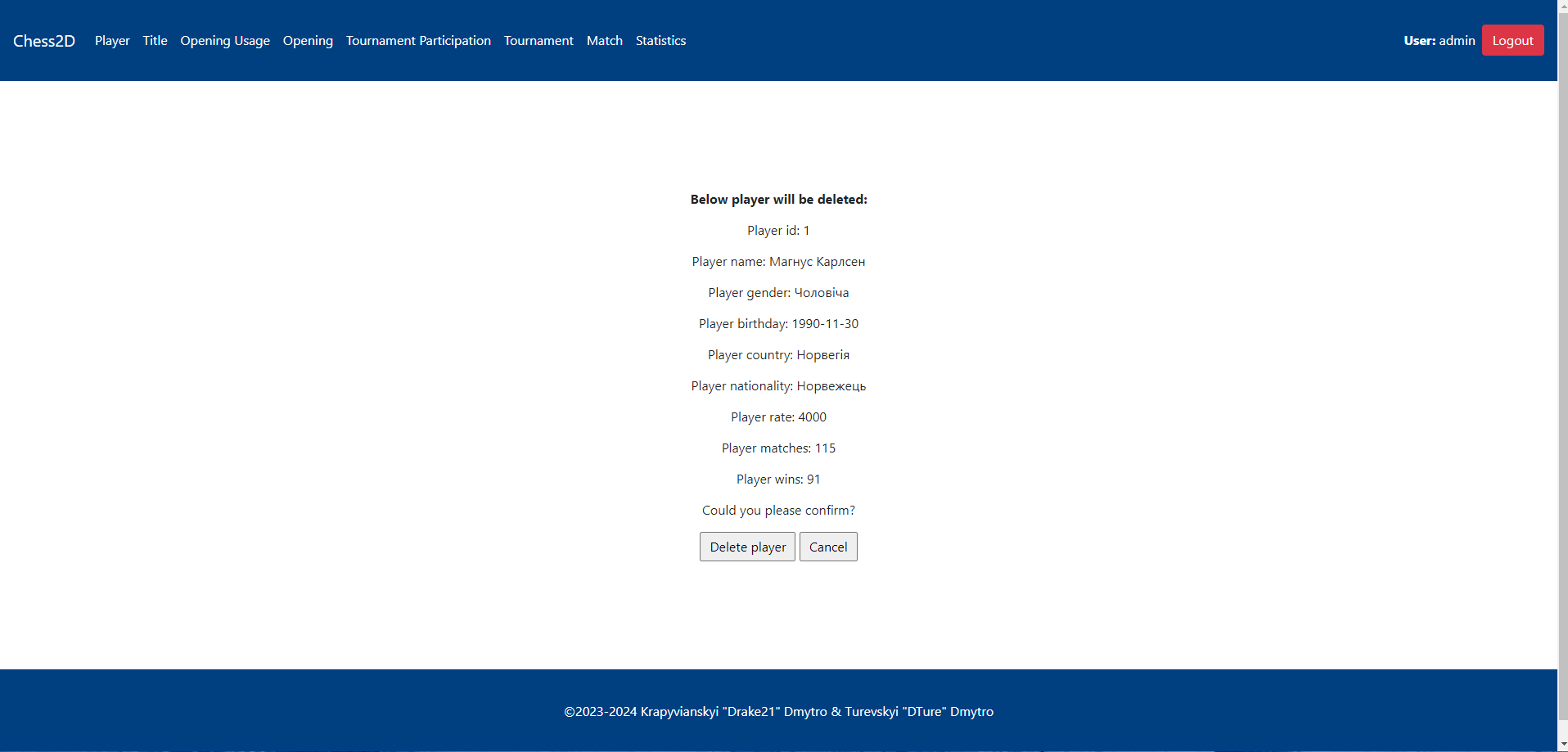


Рисунок 3.12 – Сторінка видалення запису в сутності Player

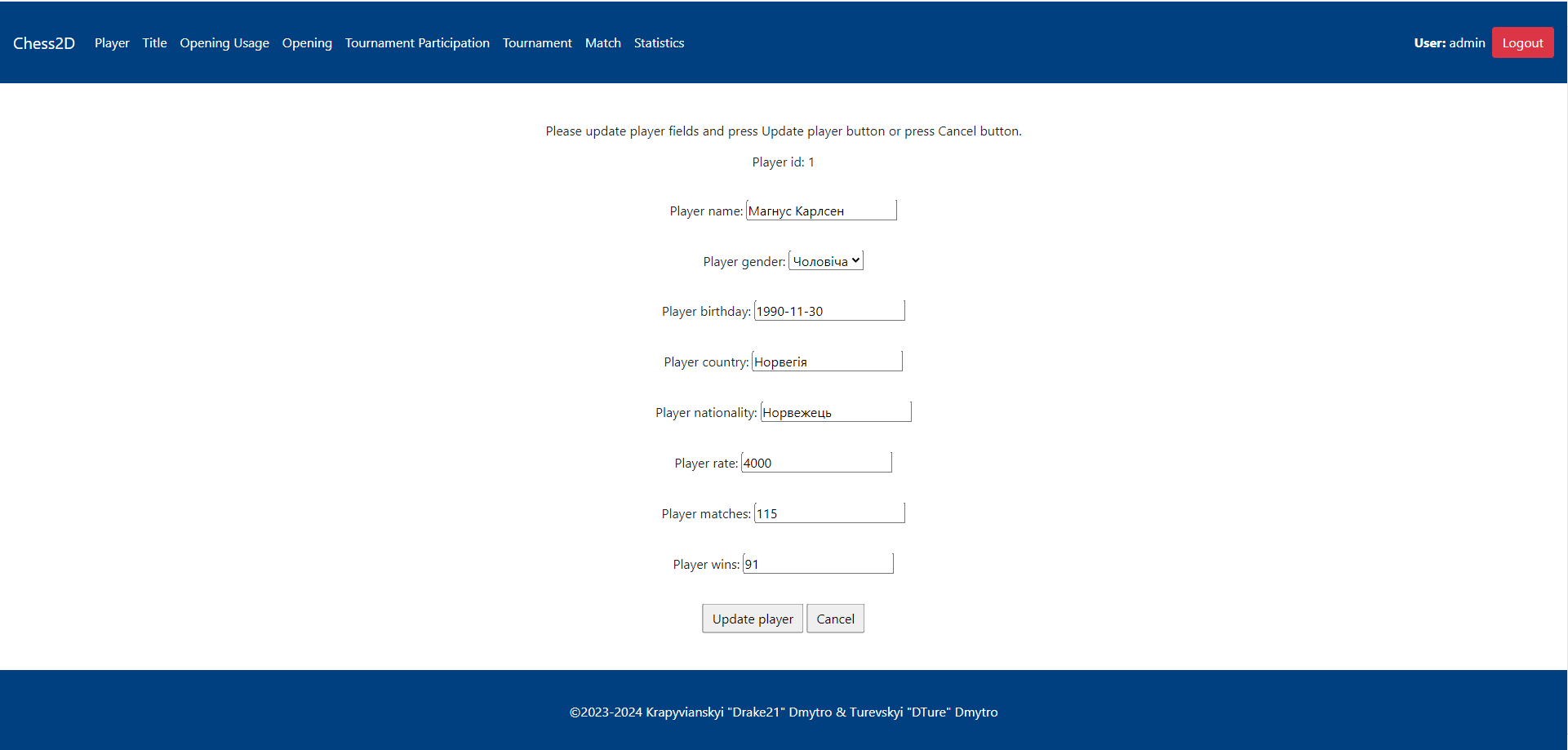


Рисунок 3.13 – Сторінка оновлення запису в сутності Player

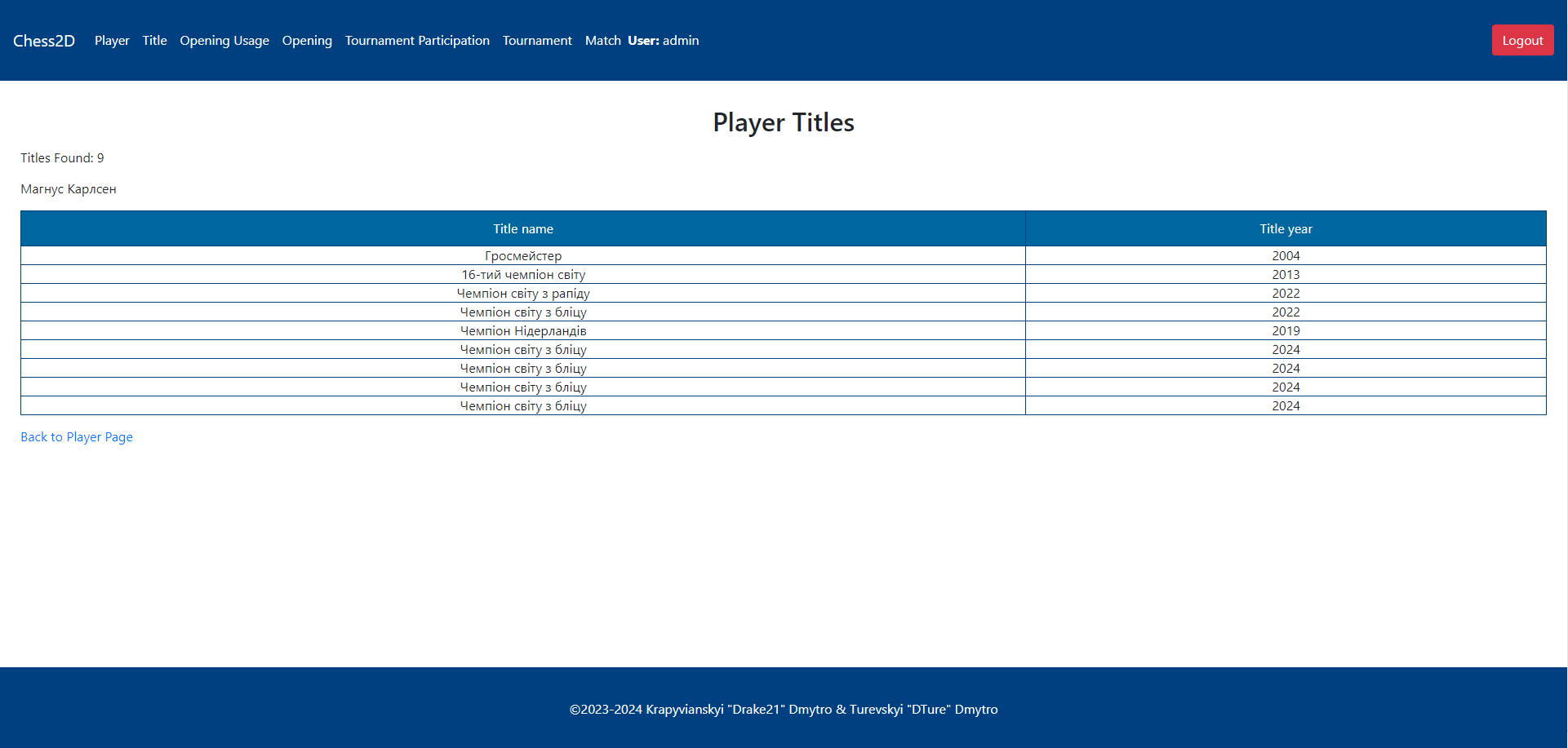


Рисунок 3.14 – Опціональна сторінка зі всіма титулами певнего гравця

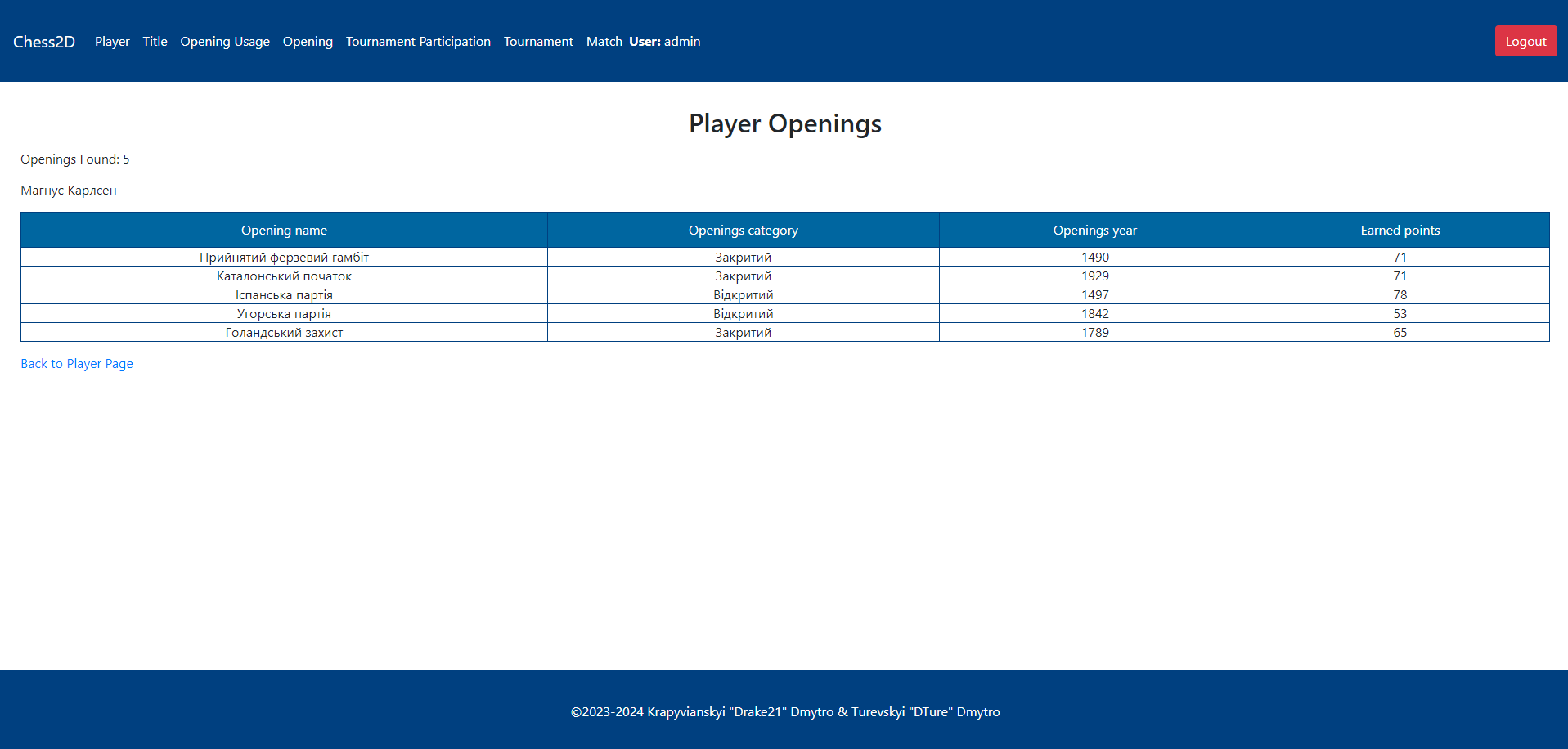


Рисунок 3.15 – Опціональна сторінка зі всіма початковими стратегіями певнего гравця

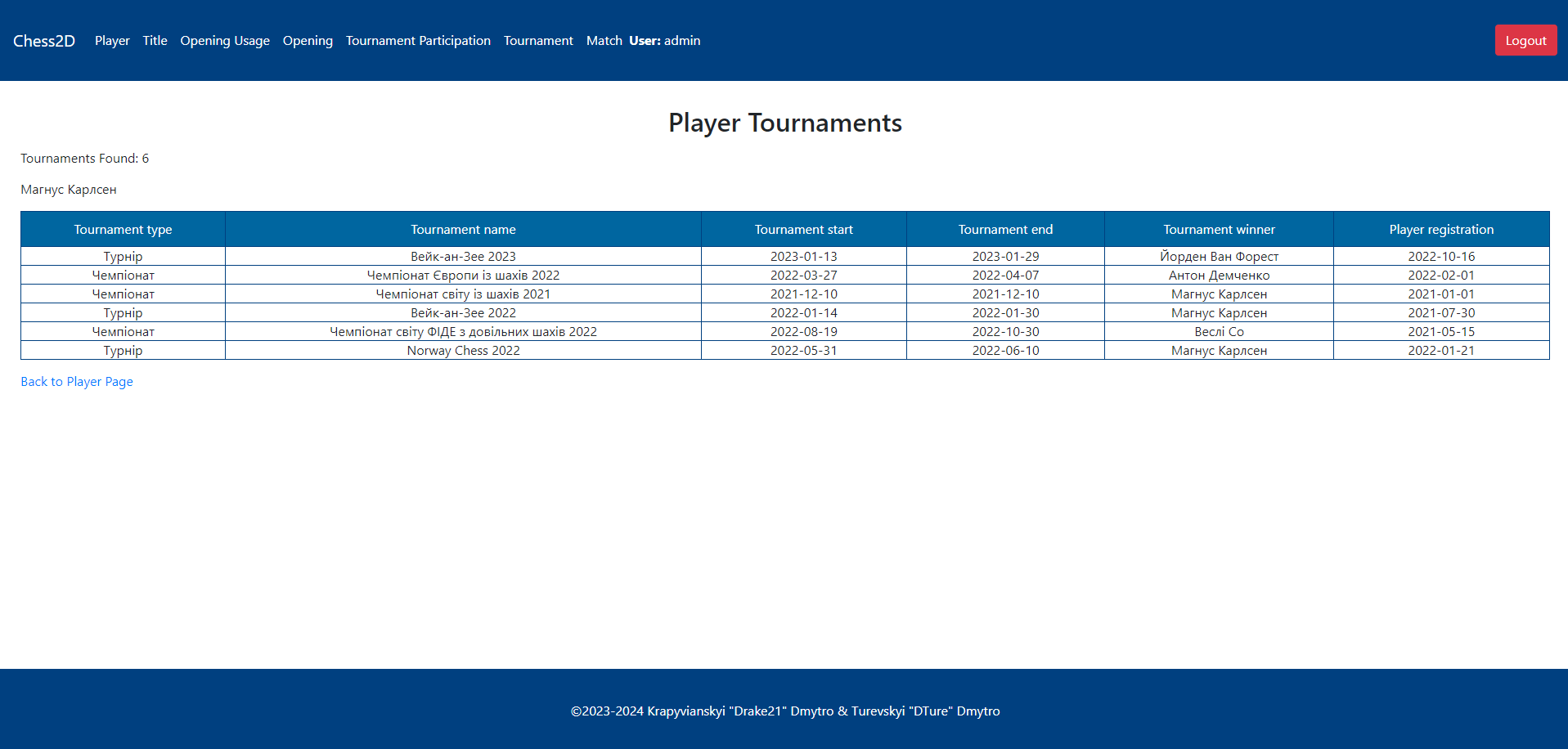


Рисунок 3.16 – Опціональна сторінка зі всіма турнігами певнего гравця

Набори сторінок для всіх інших сутностей є подібними, тому не будемо наводити вигляд кожного з них.

Для того, щоб надати користувачу пройти процедуру автентифікації було додану форму, яка наведена на рисунку 3.17. Для виходу з облікового запису достатньо натиснути кнопку «Logout», що переадресує користувача на сторінку входу до системи.

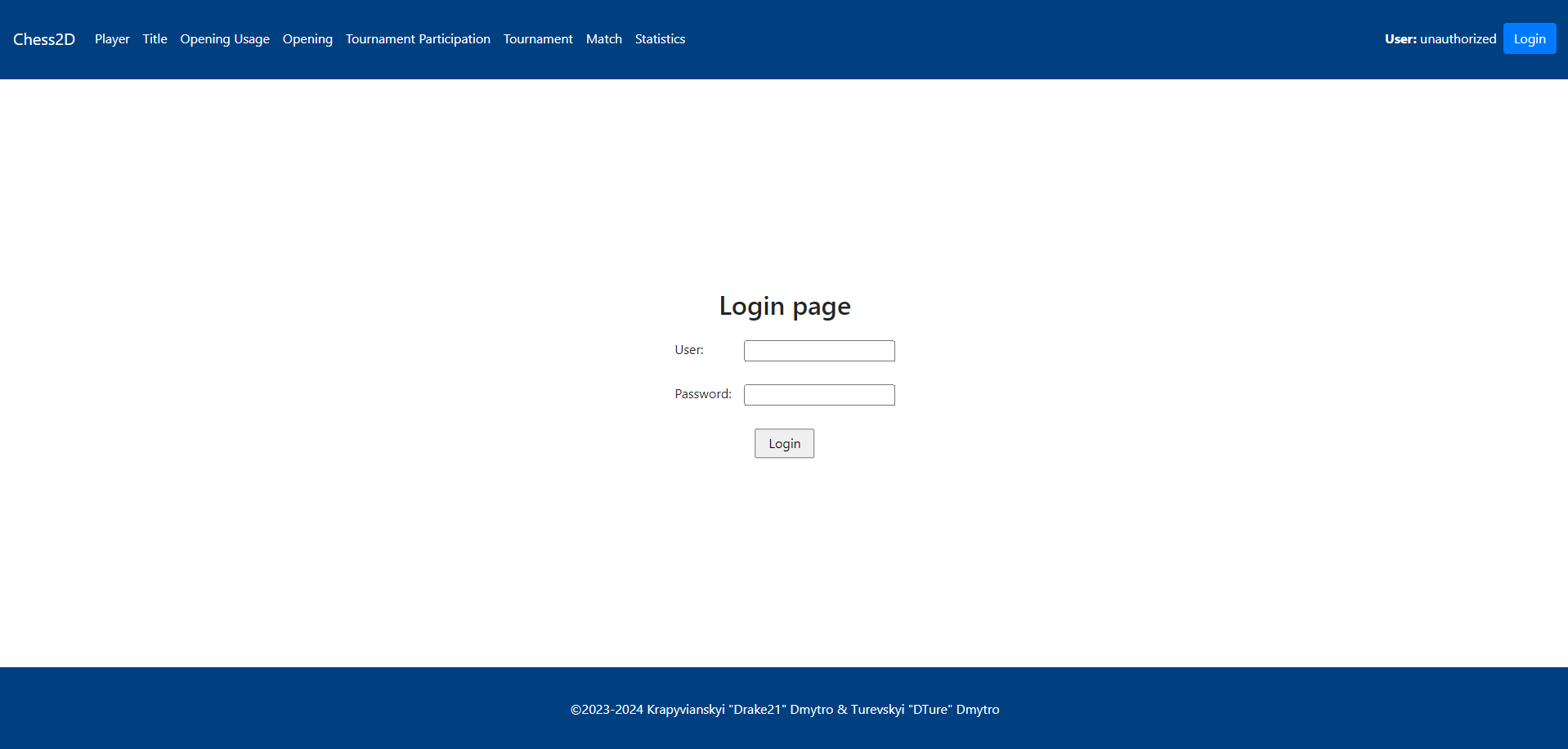


Рисунок 3.17 – Сторінка автентифікації

Створена сторінка з базовою статистикою наведена на рисунку х.18.

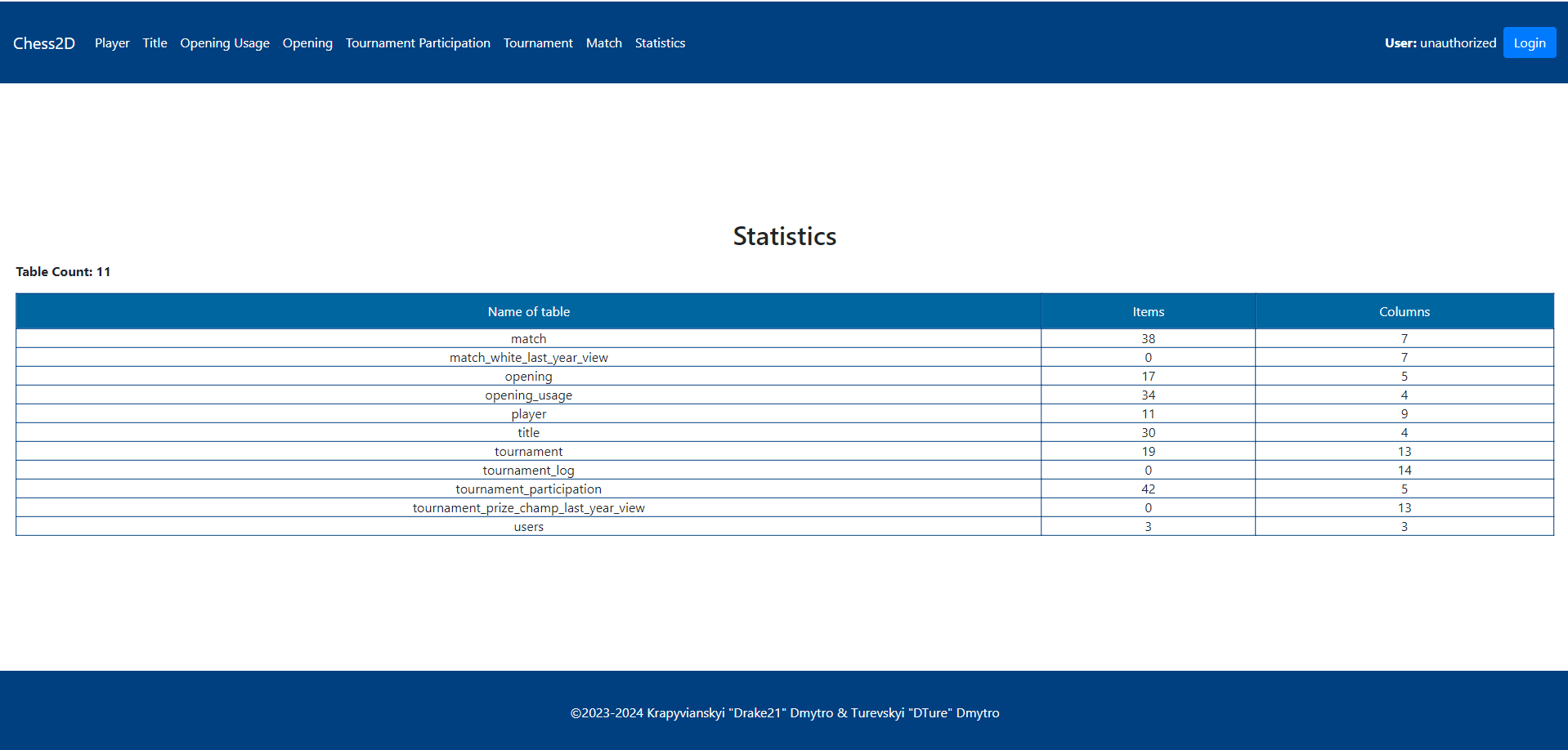


Рисунок 3.18 – Сторінка базової сторінки

## 3.4 Тестування створеної системи

Коректність роботи системи з даними перевіримо на сутності tournament.

Результати тестування зображено на рисунках .19 - .28

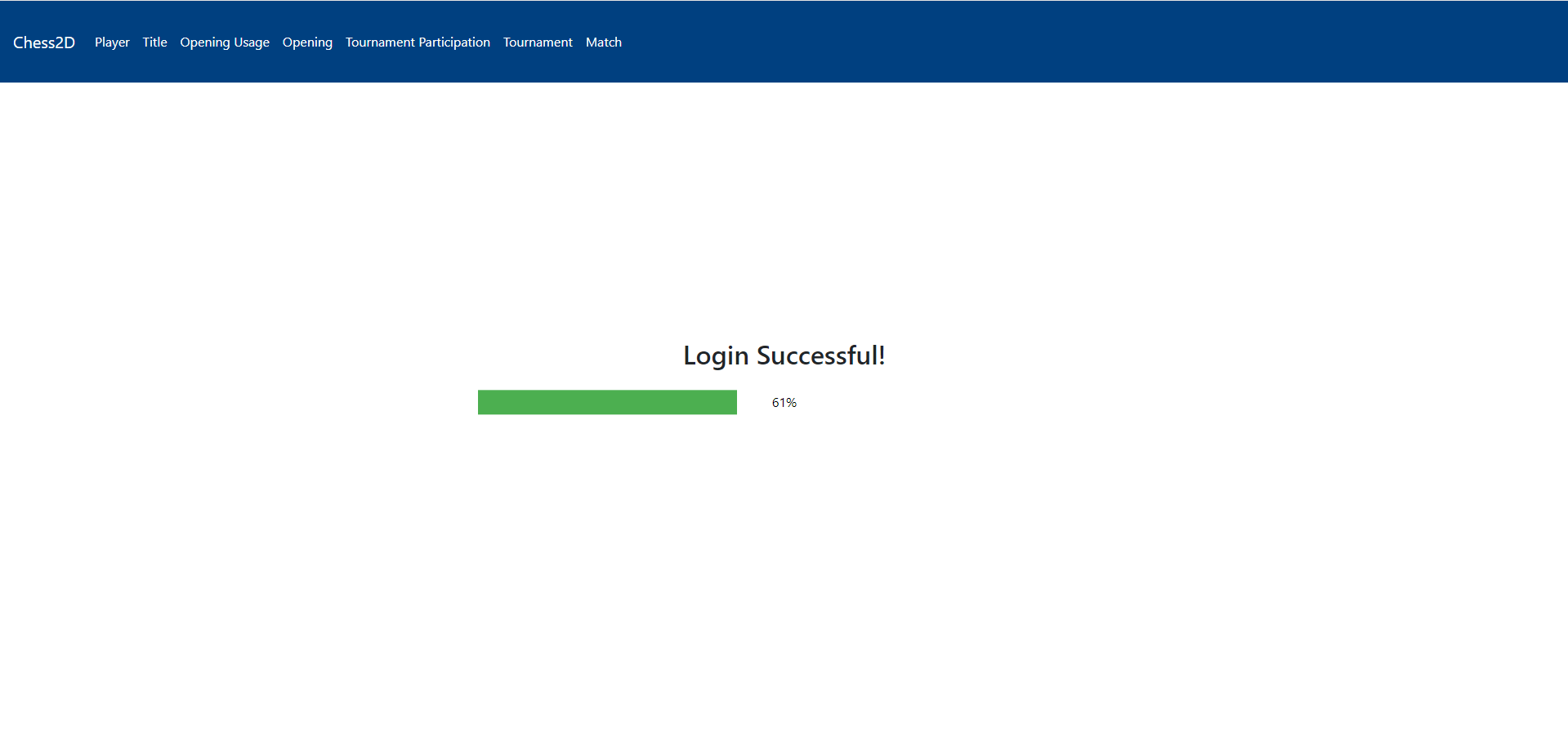


Рисунок 3.19 – Перевірка вдалого входу до системи

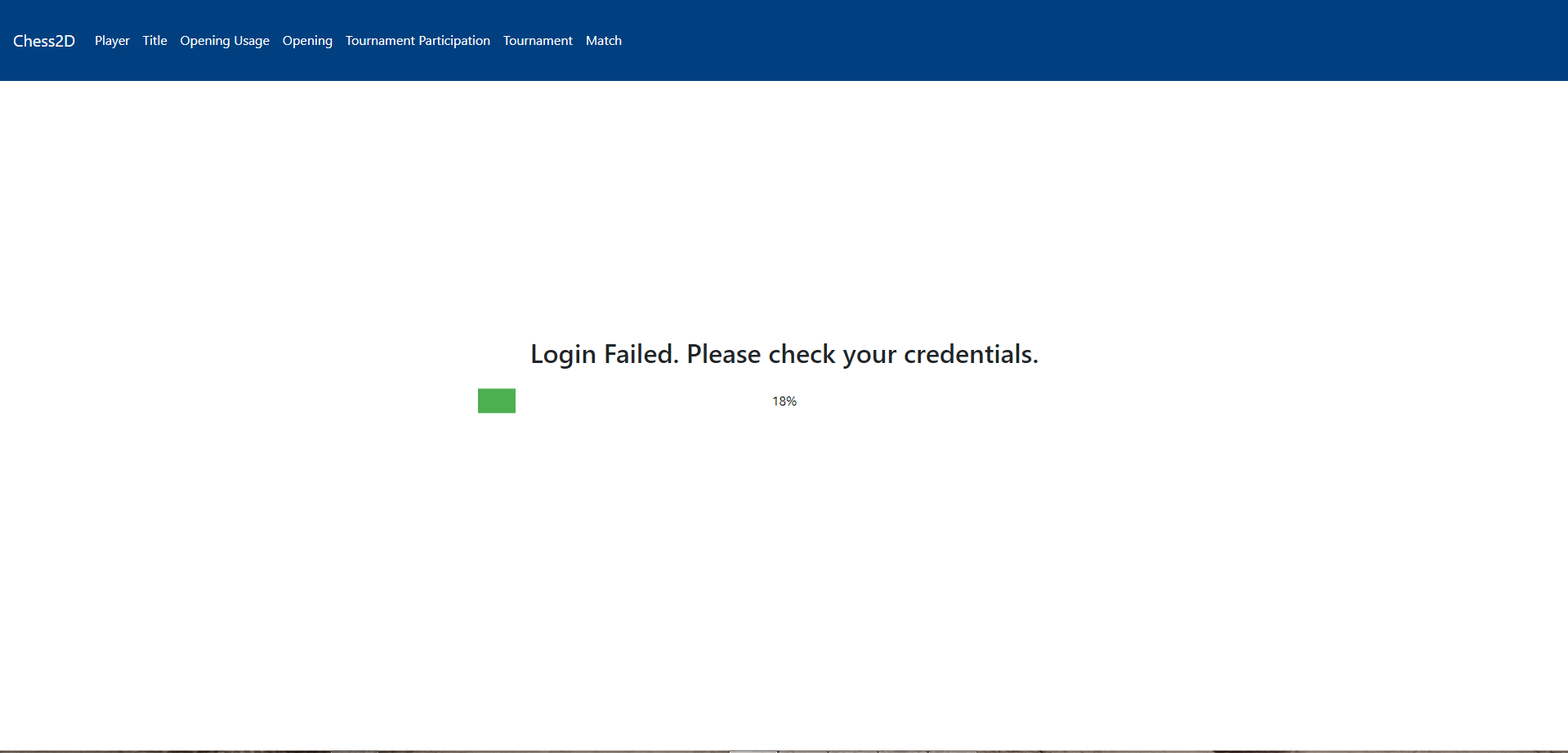


Рисунок 3.20 – Перевірка невдалого входу до системи

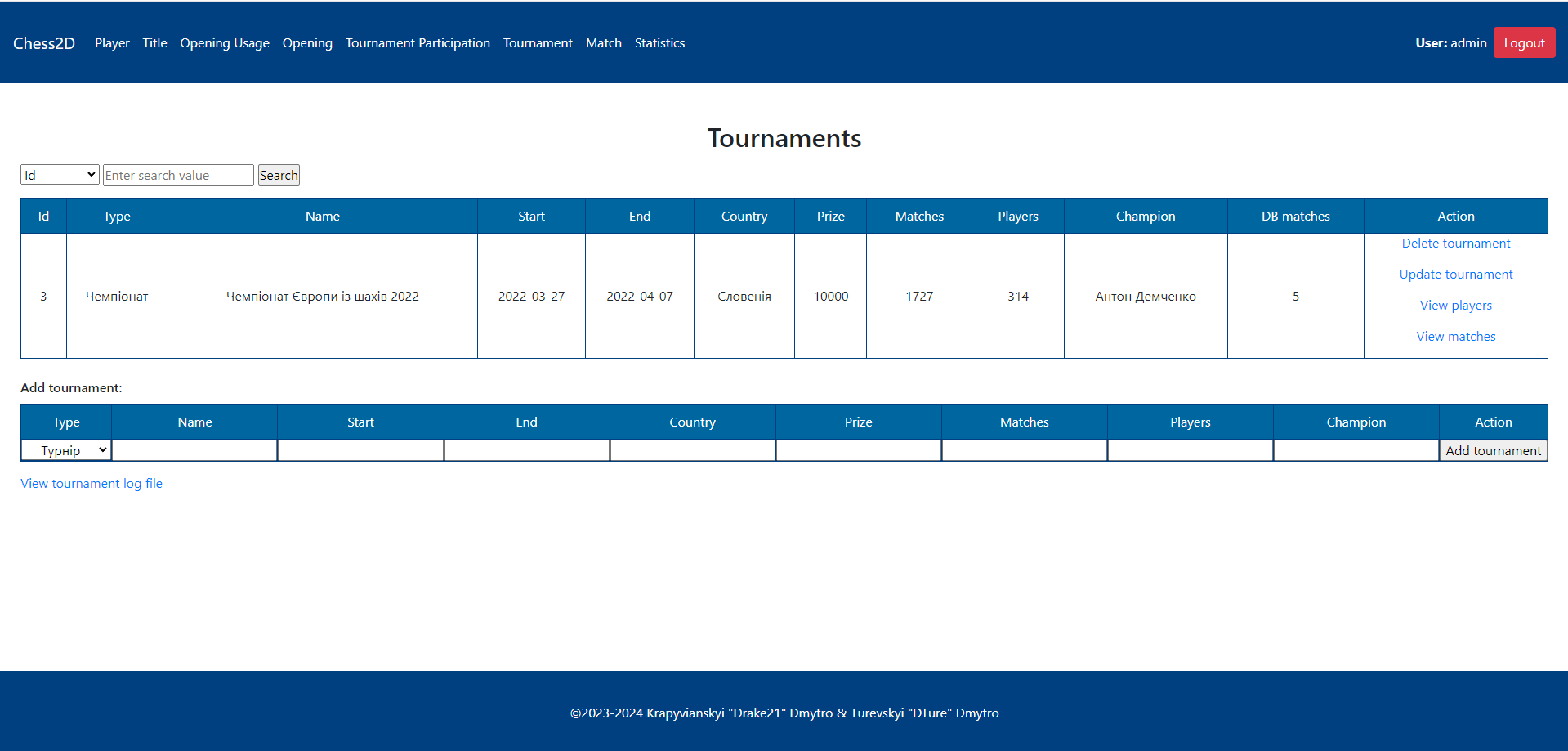


Рисунок 3.21 – Пошук відповідного турніру

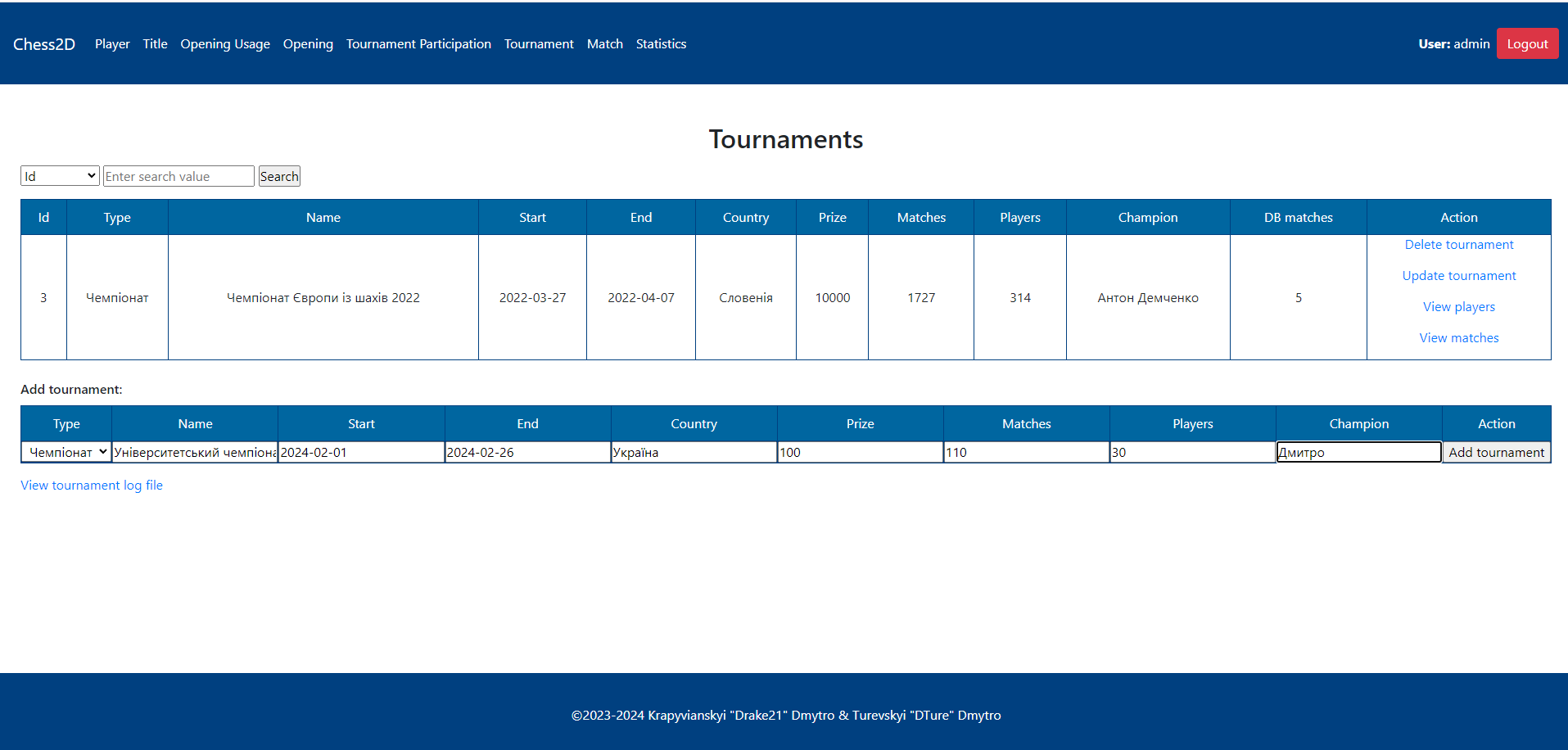


Рисунок 3.22 – Додавання нового турніру

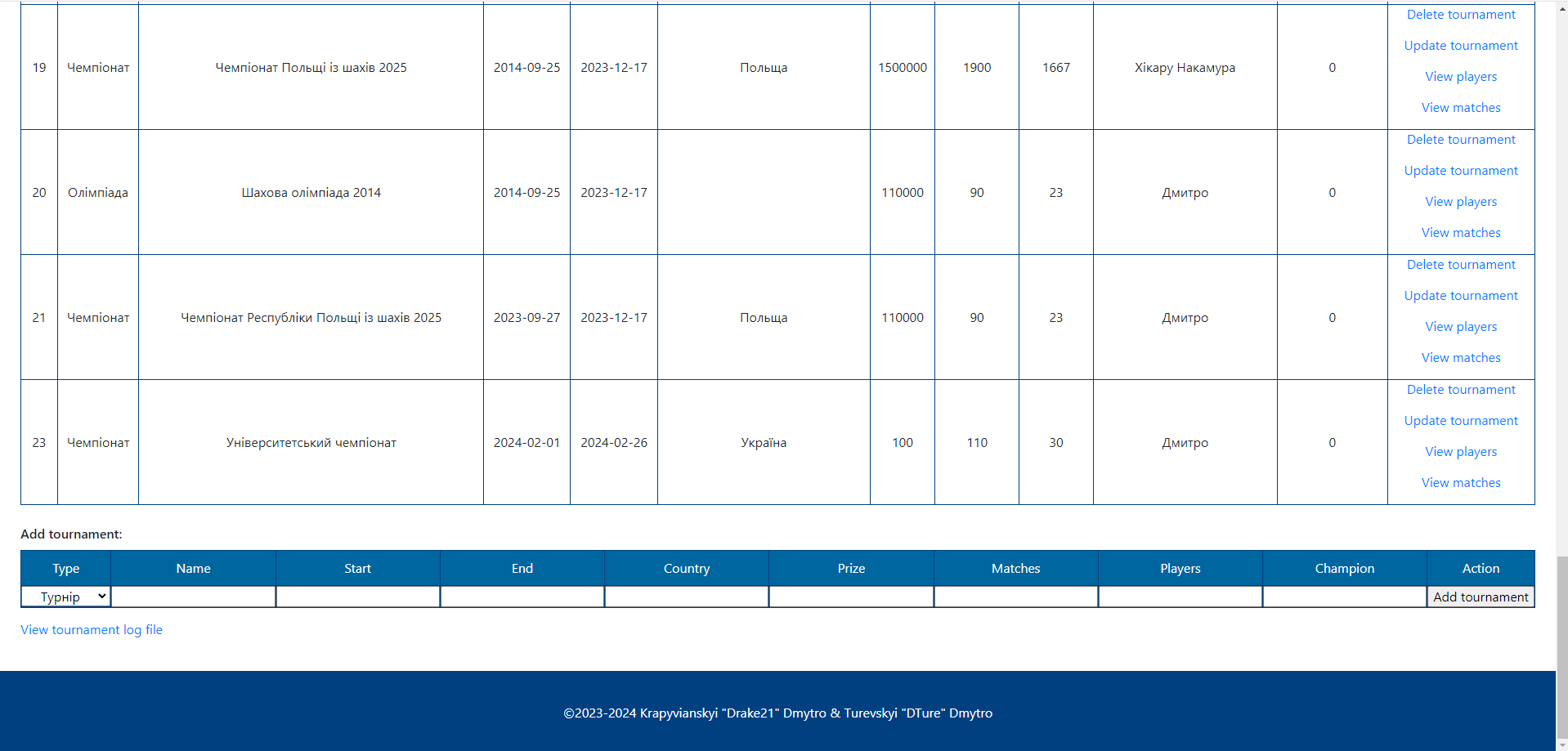


Рисунок 3.23 – Перевірка наявності нового турніру

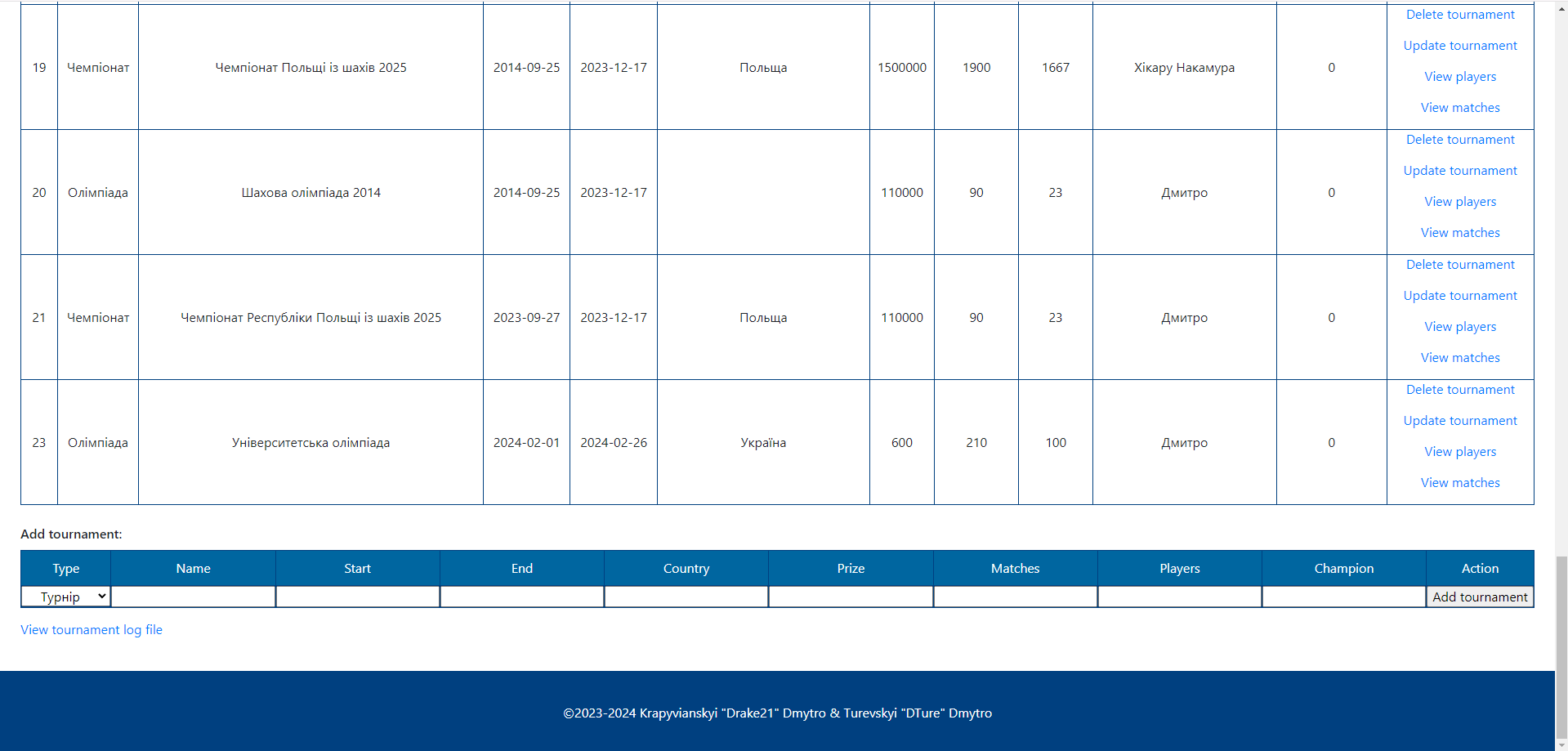


Рисунок 3.24 – Результат оновлення доданого турніру

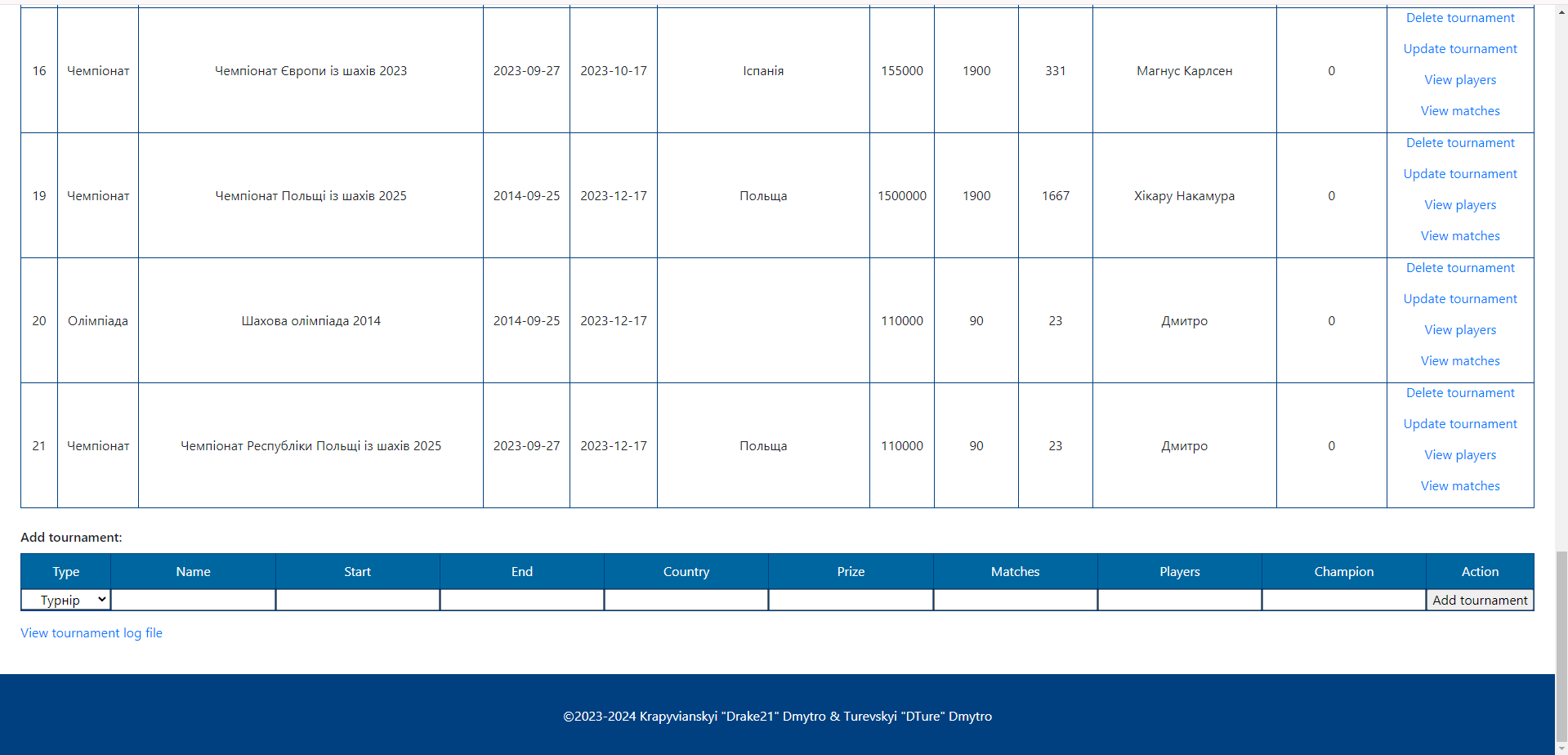


Рисунок 3.25 – Результат видалення доданого турніру

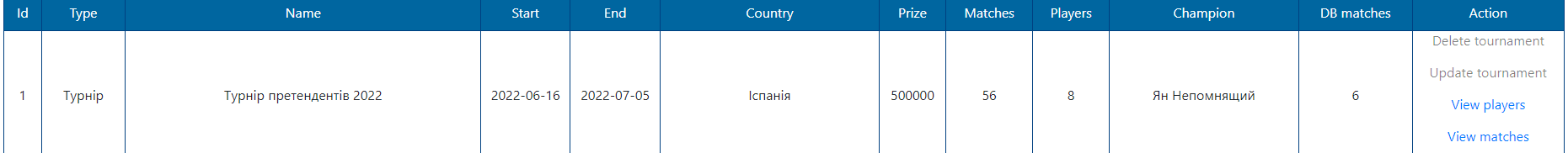


Рисунок 3.26 – Перевірка прав звичайного користувача

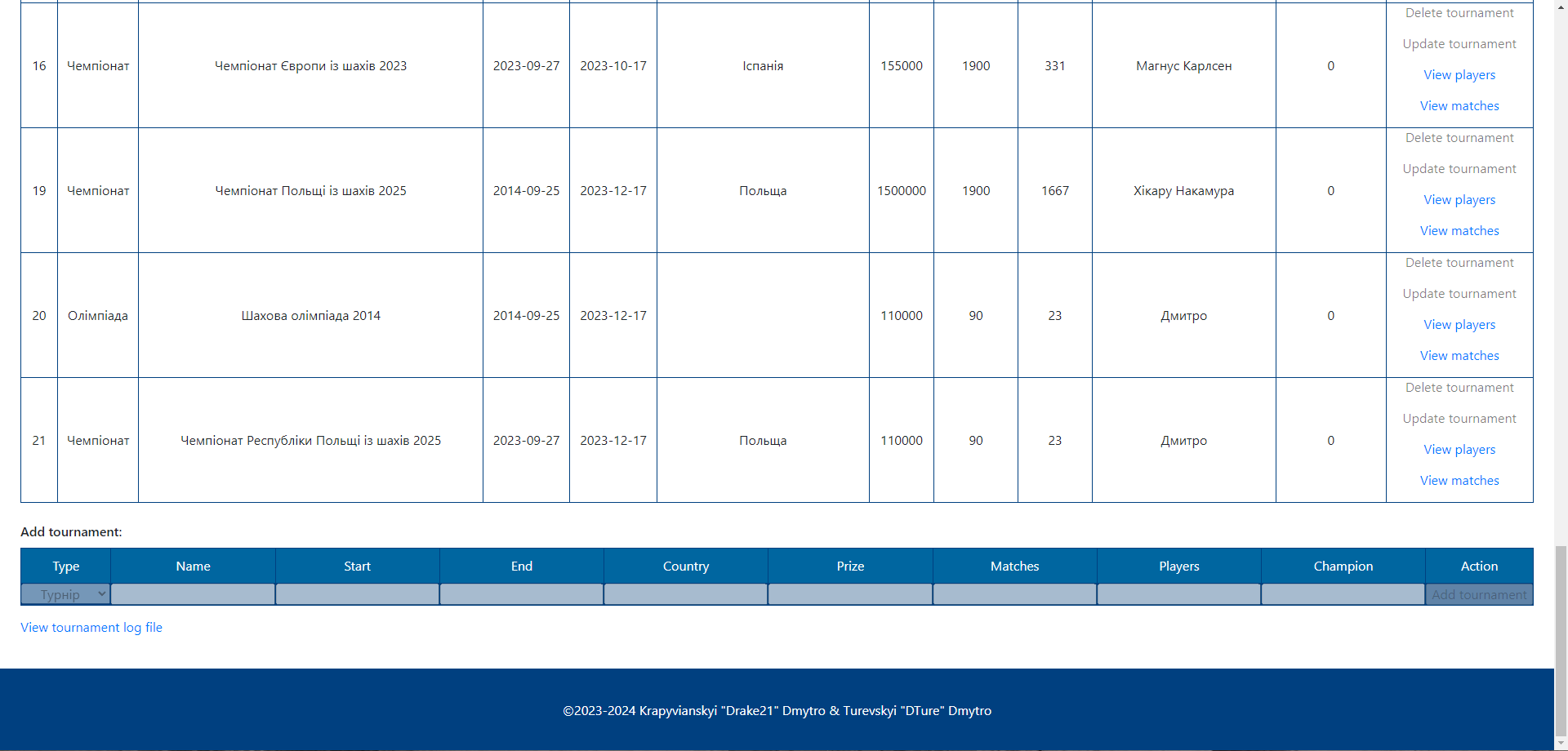


Рисунок 3.27 – Перевірка прав неавторизованого користувача

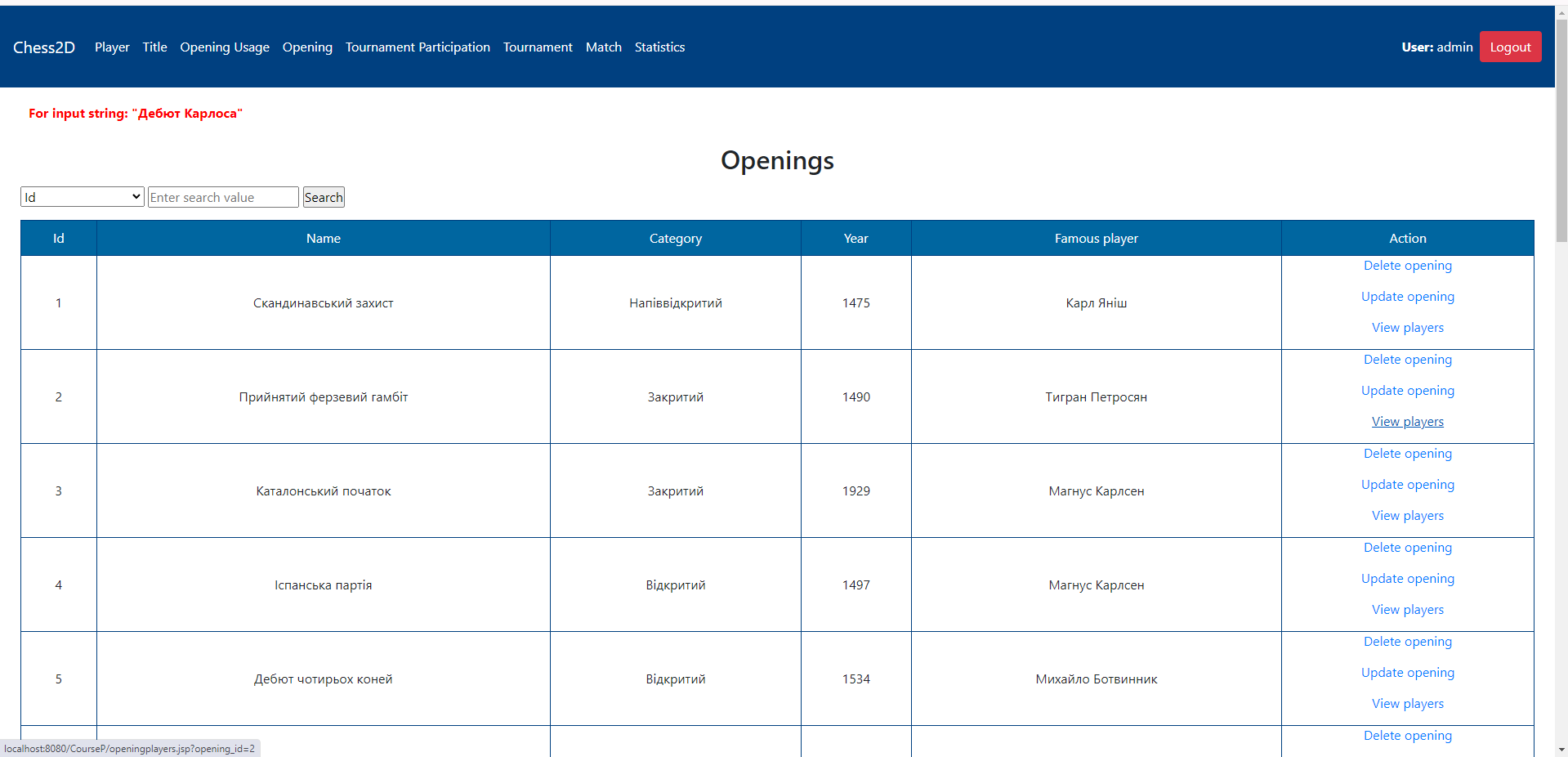


Рисунок 3.28 – Обробка помилки при доданні нового запису

Отже, система реалізована коректно.

# **ВИКОРИСТАНІ ДЖЕРЕЛА**

1. Що таке база даних.[Електронний ресурс].—Режим доступу: http://apeps.kpi.ua/shco-take-basa-danykh.

2. Joel Murach. Murach’s MySQL (3rd Edition). — Mike Murach & Associates, May 1, 2012. — 628 с. — ISBN 978-1890774684.

3. Сравнение MySQL и PostgreSQL.[Електронний ресурс].— Режим доступу: https://losst.ru/sravnenie-mysql-i-postgresql.

4. Часть 5. Сервлеты, Java servlet API. Пишем простое веб-приложение [Електронний ресурс].— Режим доступу: https://javarush.ru/groups/posts/2529-chastjh-5-servletih-pishem-prostoe-veb-prilozhenie.

5. CSS. [Електронний ресурс].— Режим доступу: https://ru.m.wikipedia.org/wiki/CSS.

6. Оптимізація SQL-запитів. Методичні вказівки до виконання розрахунково-графічної роботи з дисципліни «Бази даних» для студентів напряму підготовки 121 – „Інженерія програмного забезпечення”. /Укл.: Білоус І.В. – ЧНТУ, 2020. – 15с. – Електроннi даннi – Режим доступу: https://eln.stu.cn.ua/course/view.php?id=802 , обмежений. – Заголовок з екрану.