**Розробка системи**

**Розробка бази даних системи**

**Розробка фізичної схеми бази даних**

На основі логічної схеми бази данних побудовали фізичну модель бази даних (рисунок .1).

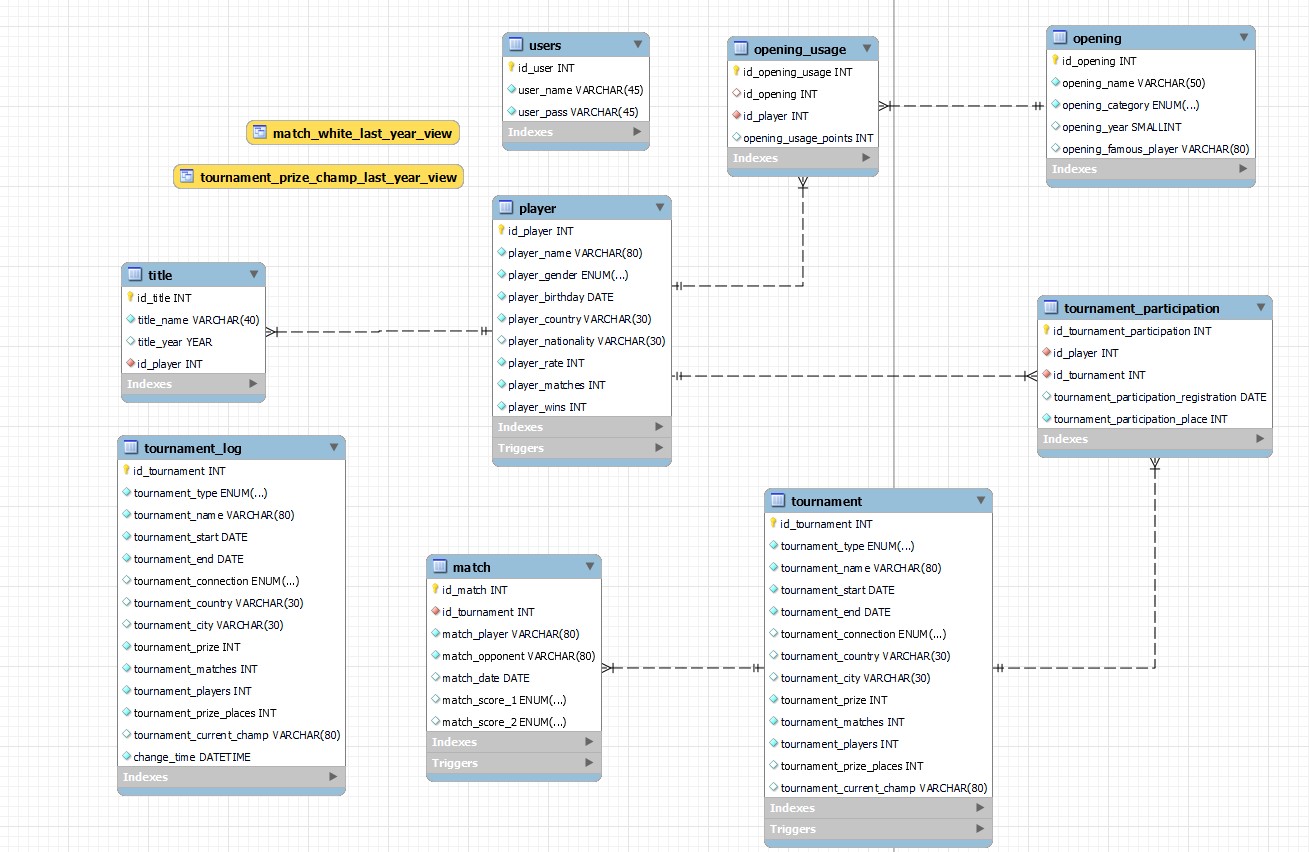


Рисунок .1 – Фізична схема бази даних

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

**Забезпечення цілісності даних**

Обмеження цілісності створенних таблиць наведені на рисунках х.2 – х..

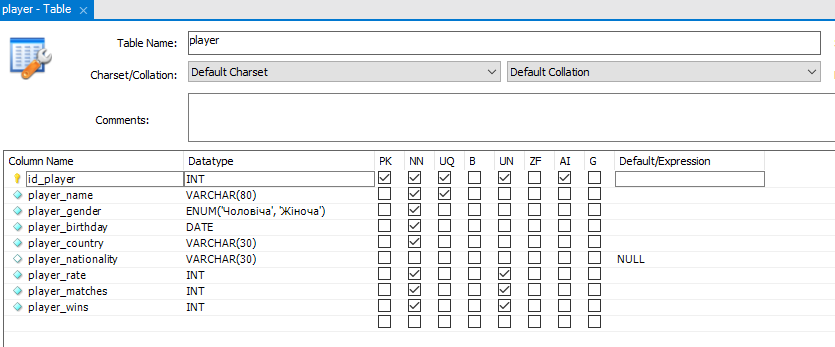


Рисунок .2 – Обмеження цілісності таблиці player

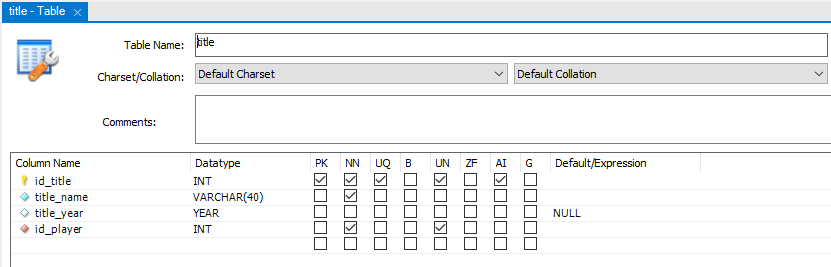
****

Рисунок .3 – Обмеження цілісності таблиці title

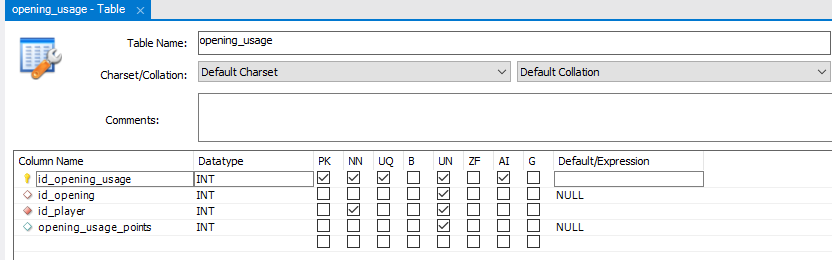
****

Рисунок .4 – Обмеження цілісності opening\_usage

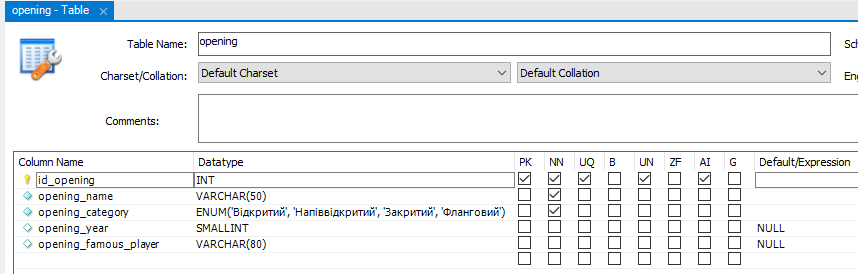
****

Рисунок .5 – Обмеження цілісності таблиці opening

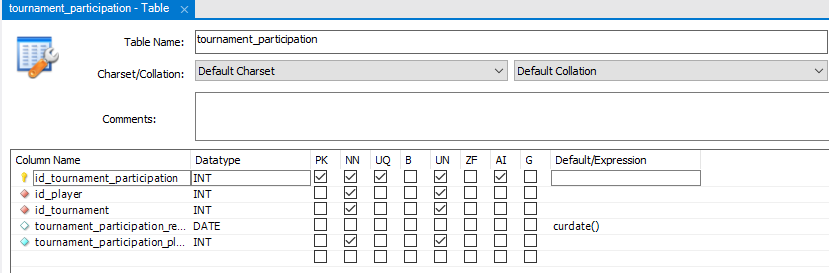
****

Рисунок .6 – Обмеження цілісності таблиці tournament\_participation

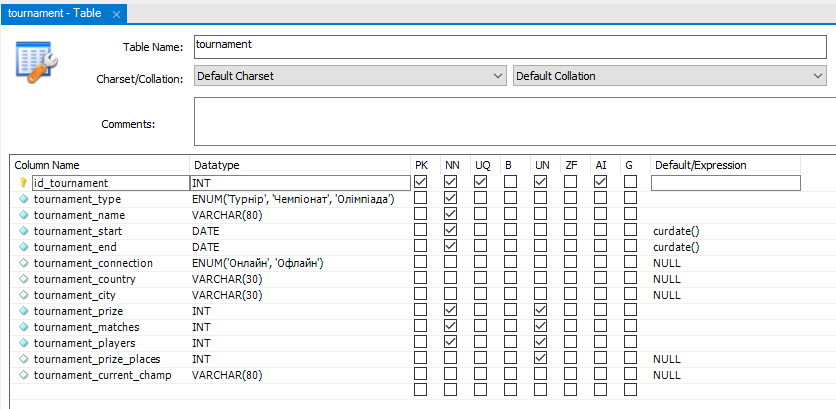
****

Рисунок .7 – Обмеження цілісності таблиці tournament

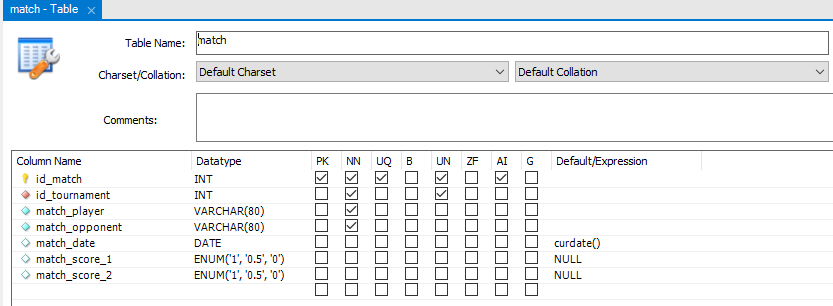
****

Рисунок .8 – Обмеження цілісності таблиці match

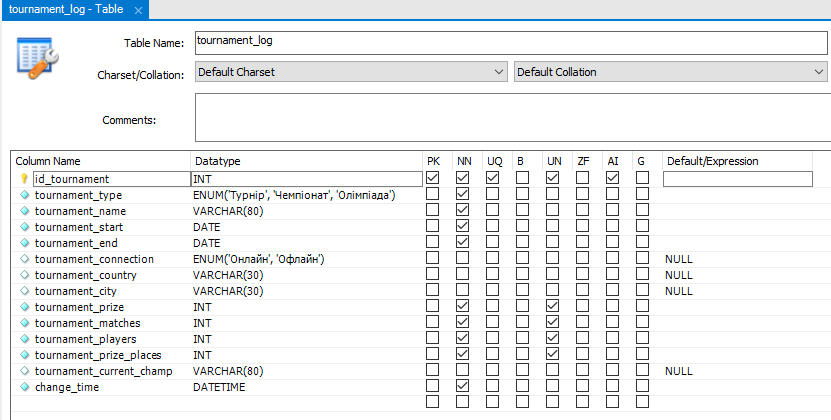
****

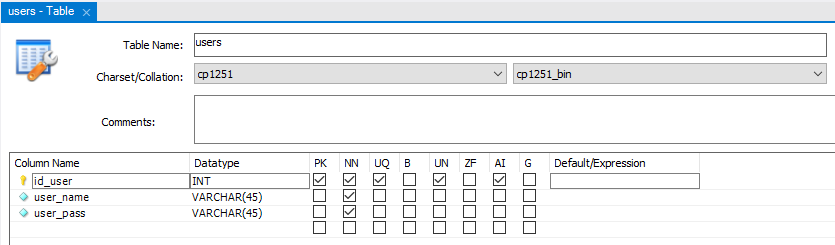
Рисунок .9 – Обмеження цілісності таблиці tournament\_log****

Рисунок .10 – Обмеження цілісності таблиці users

**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 х.1.

Лістинг х.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 х.2.

Лістинг х.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 х.3.

Лістинг х.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 х.4.

Лістинг х.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 х.5.

Лістинг х.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 х.6.

Лістинг х.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 ;

**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

**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 х.7.

Лістинг х.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.

**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 х.8.

Лістинг х.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 х.9.

Лістинг х.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 х.11.

Лістинг х.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.

**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 х.12.

Лістинг х.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 х.11-х.13.

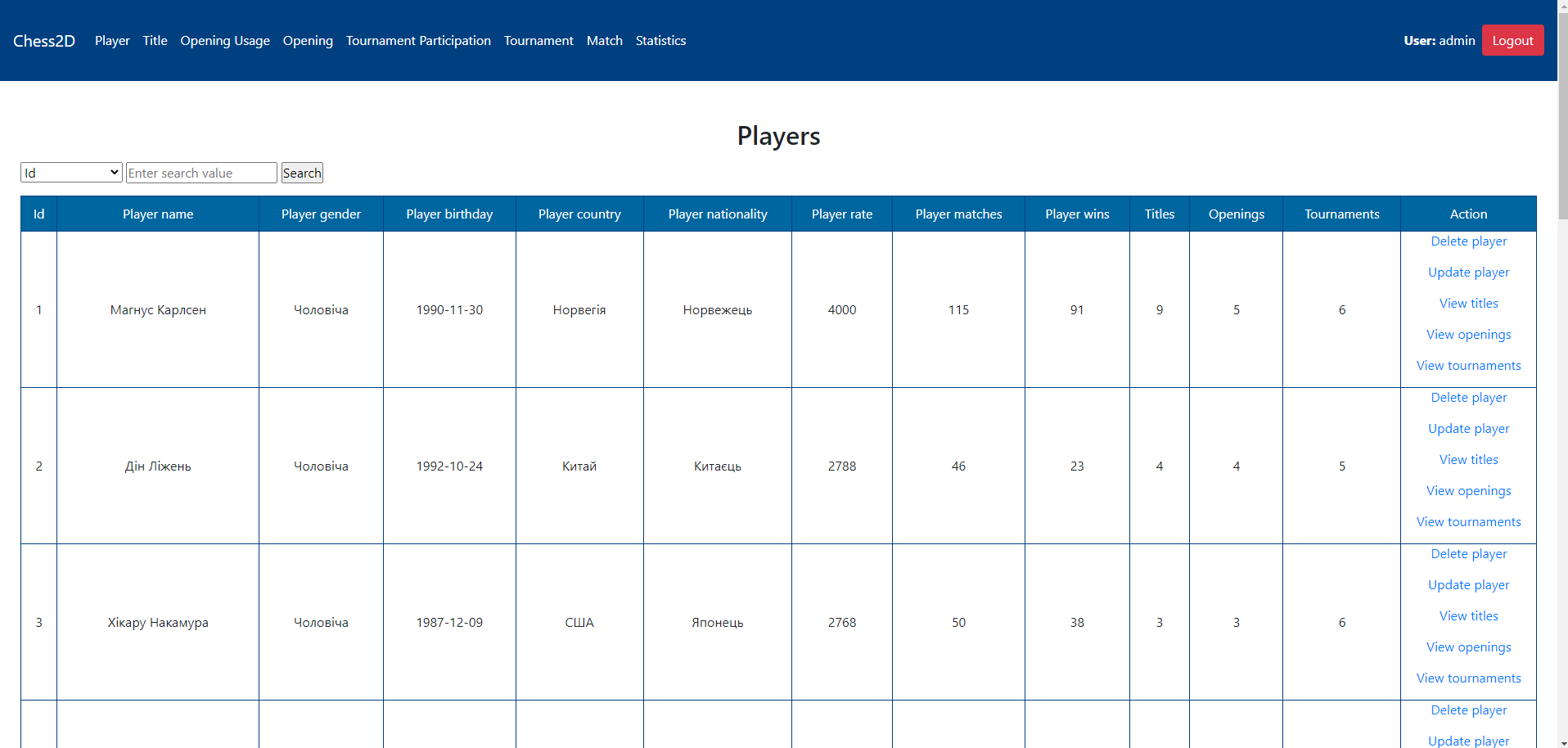


Рисунок .11 – Головна сторінка сутності Player

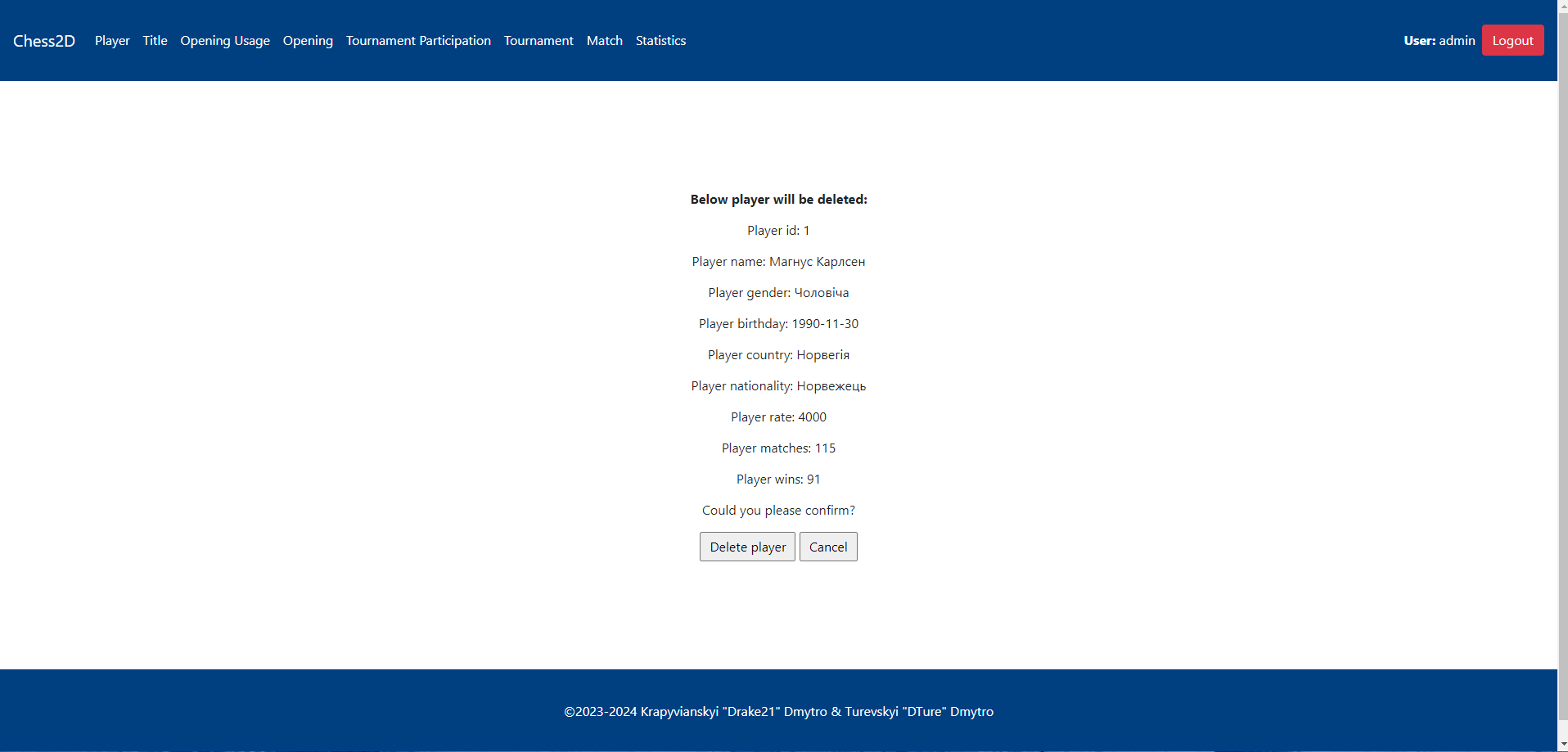


Рисунок .12 – Сторінка видалення запису в сутності Player

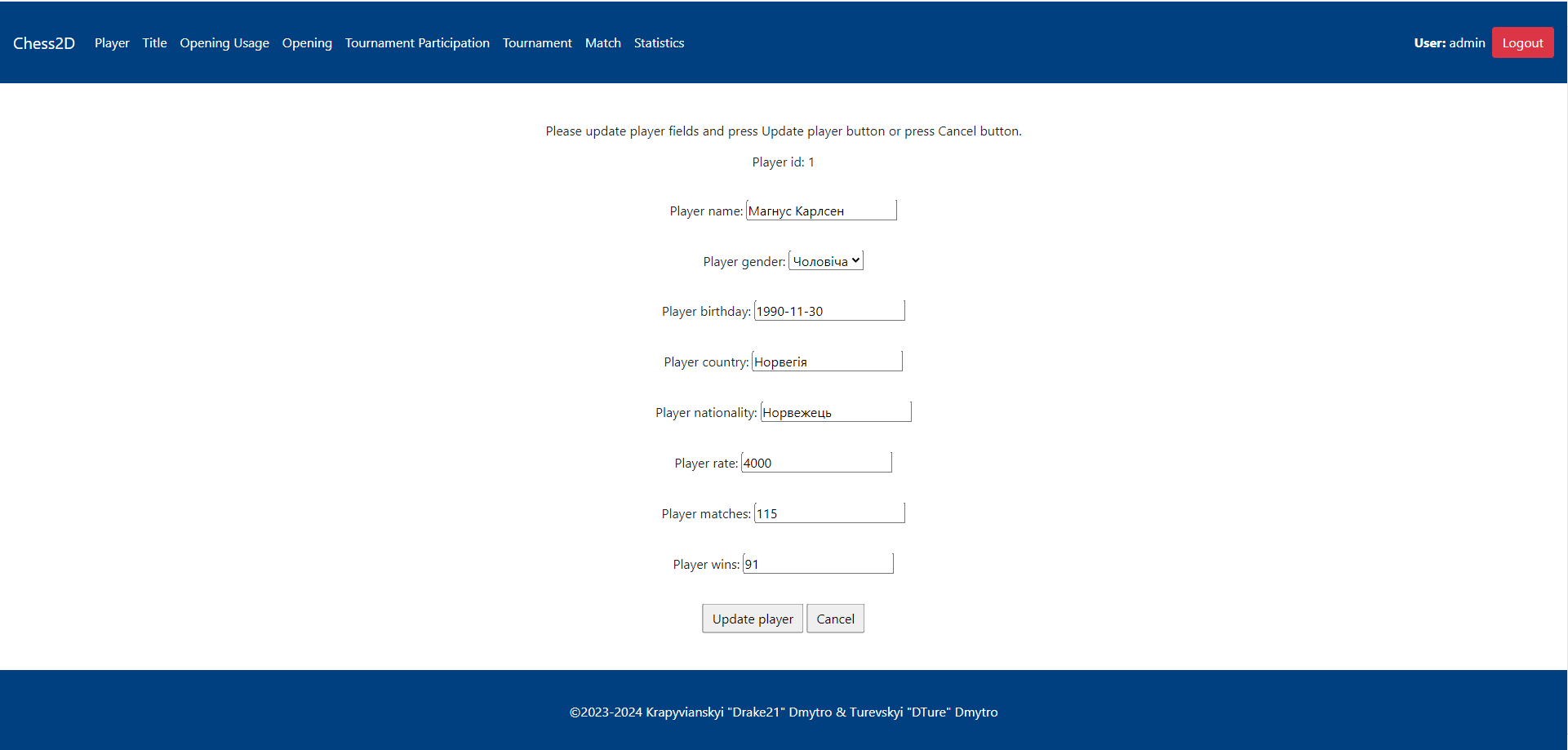


Рисунок .13 – Сторінка оновлення запису в сутності Player

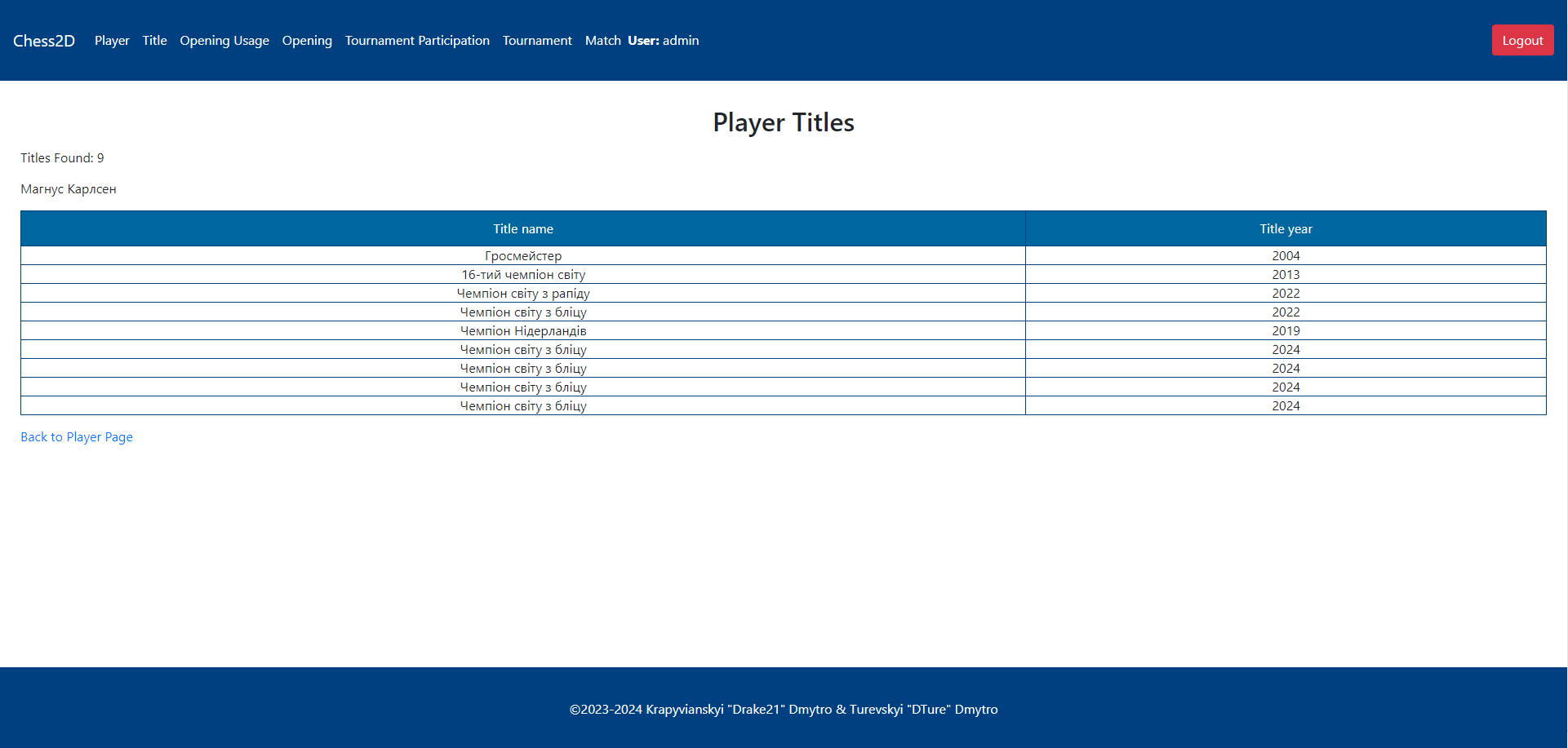


Рисунок .14 – Опціональна сторінка зі всіма титулами певнего гравця

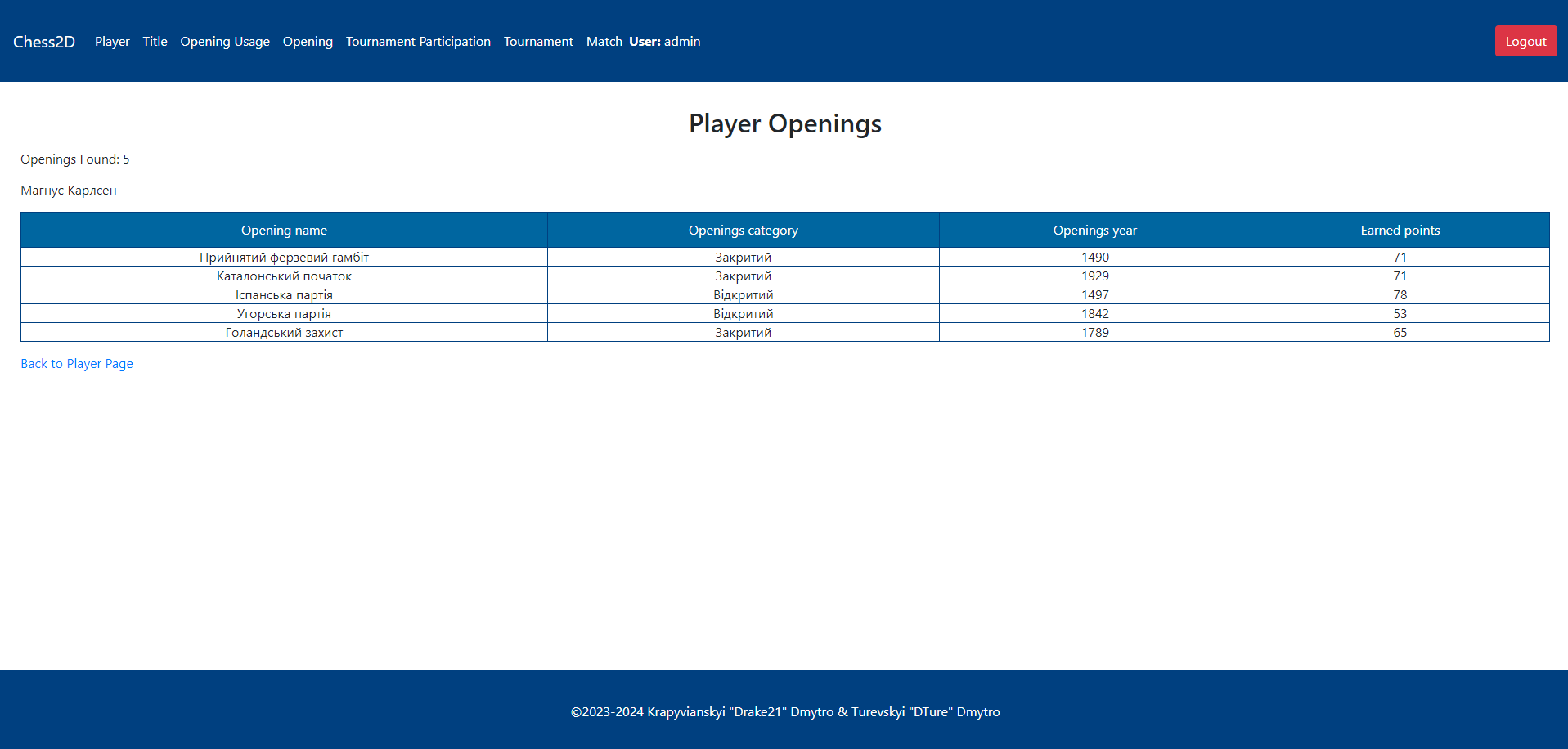


Рисунок .15 – Опціональна сторінка зі всіма початковими стратегіями певнего гравця

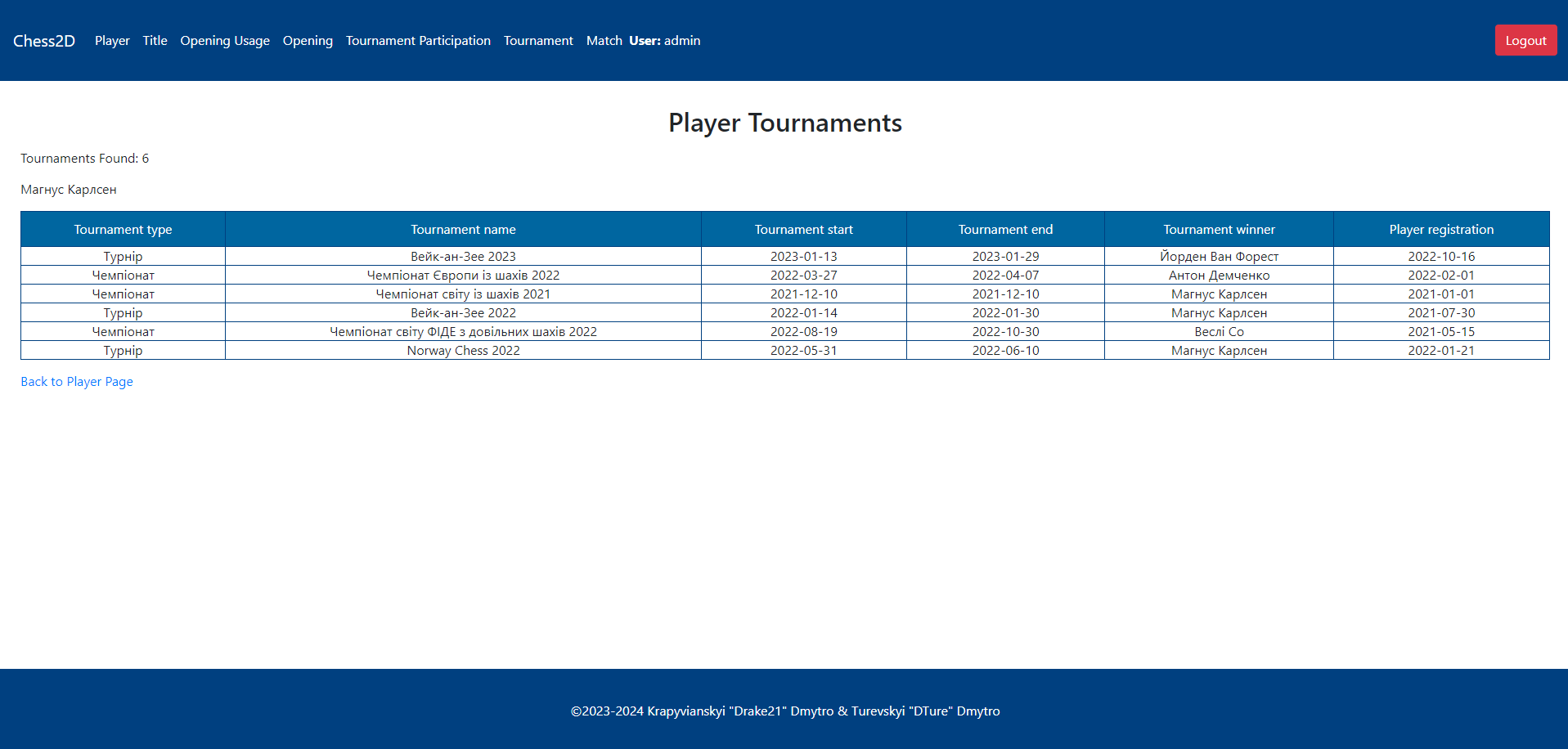


Рисунок .16 – Опціональна сторінка зі всіма турнігами певнего гравця

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 х.17. Для виходу з облікового запису достатньо натиснути кнопку «Logout», що переадресує користувача на сторінку входу до системи.

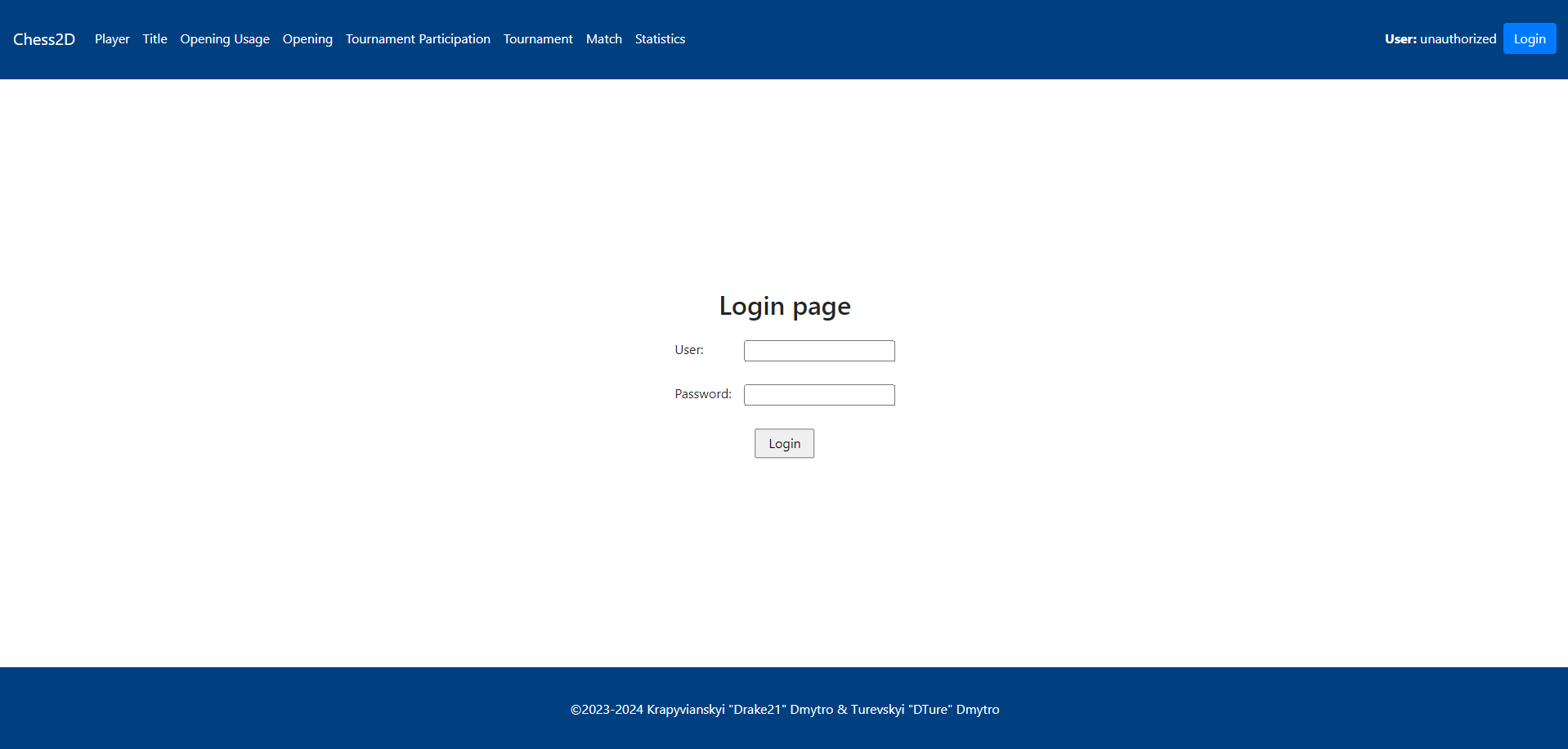


Рисунок .17 – Сторінка автентифікації

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

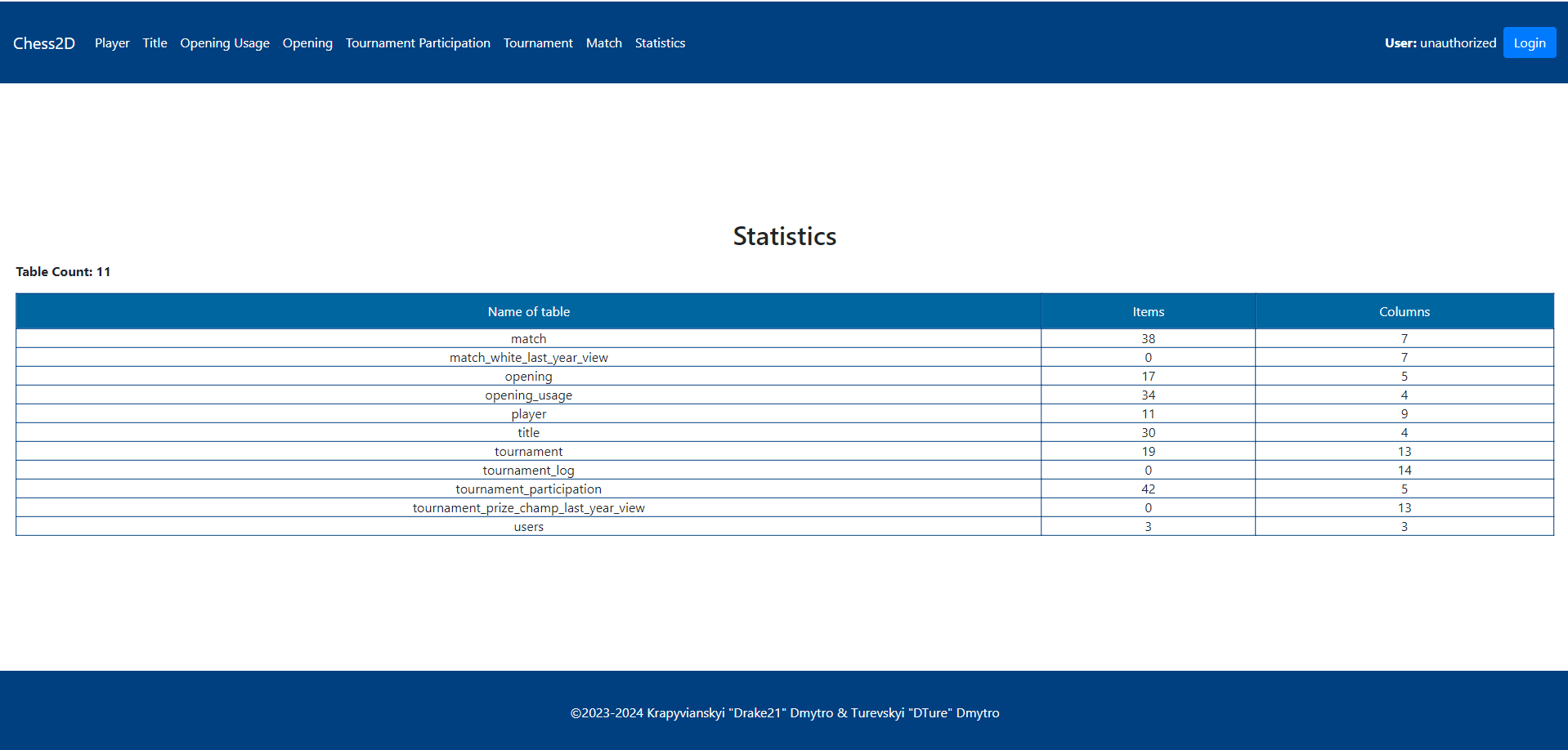


Рисунок .18 – Сторінка базової сторінки

**Тестування створеної системи**

Коректність роботи системи з даними перевіримо на сутності tournament.

Результати тестування зображено на рисунках .19 - .28

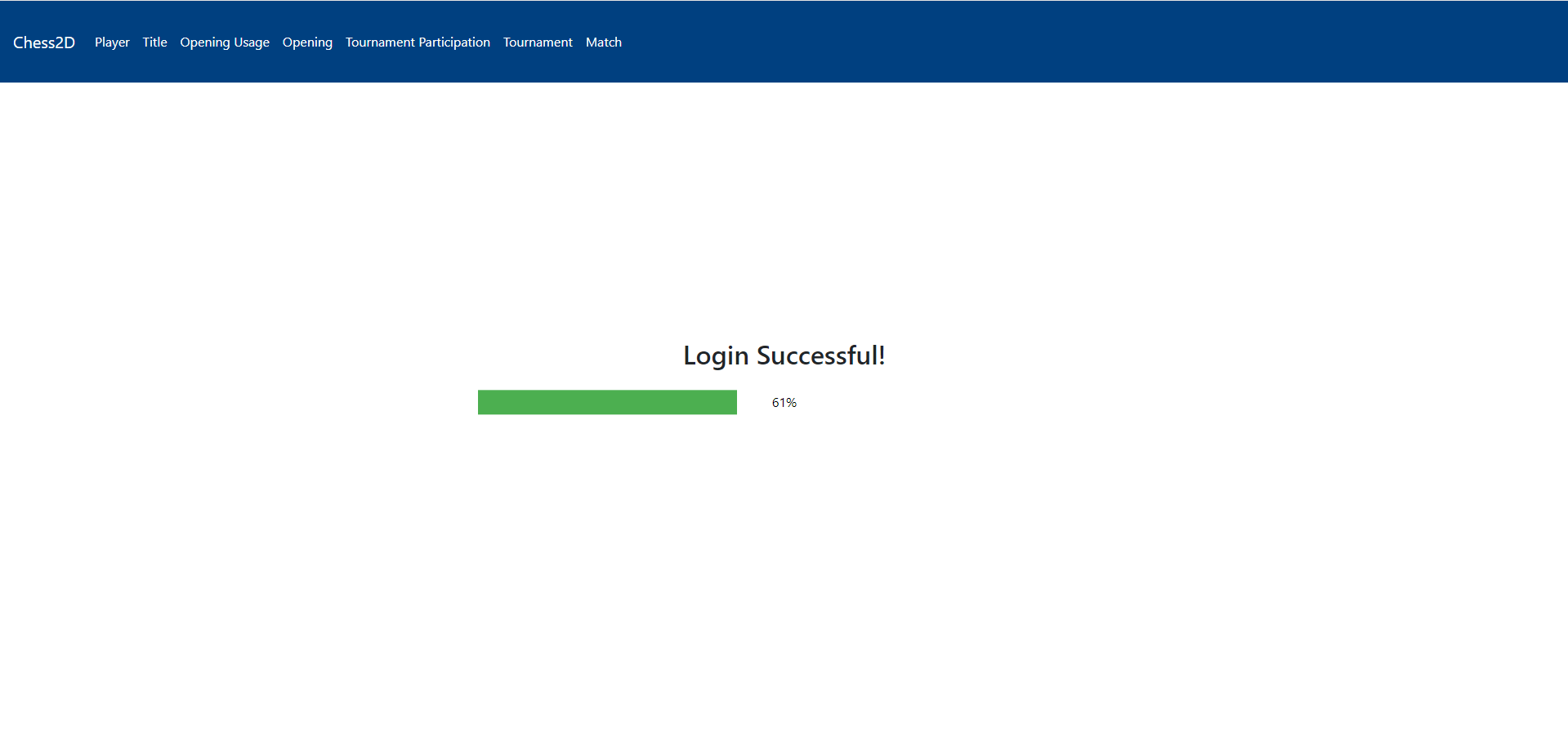


Рисунок .19 – Перевірка вдалого входу до системи

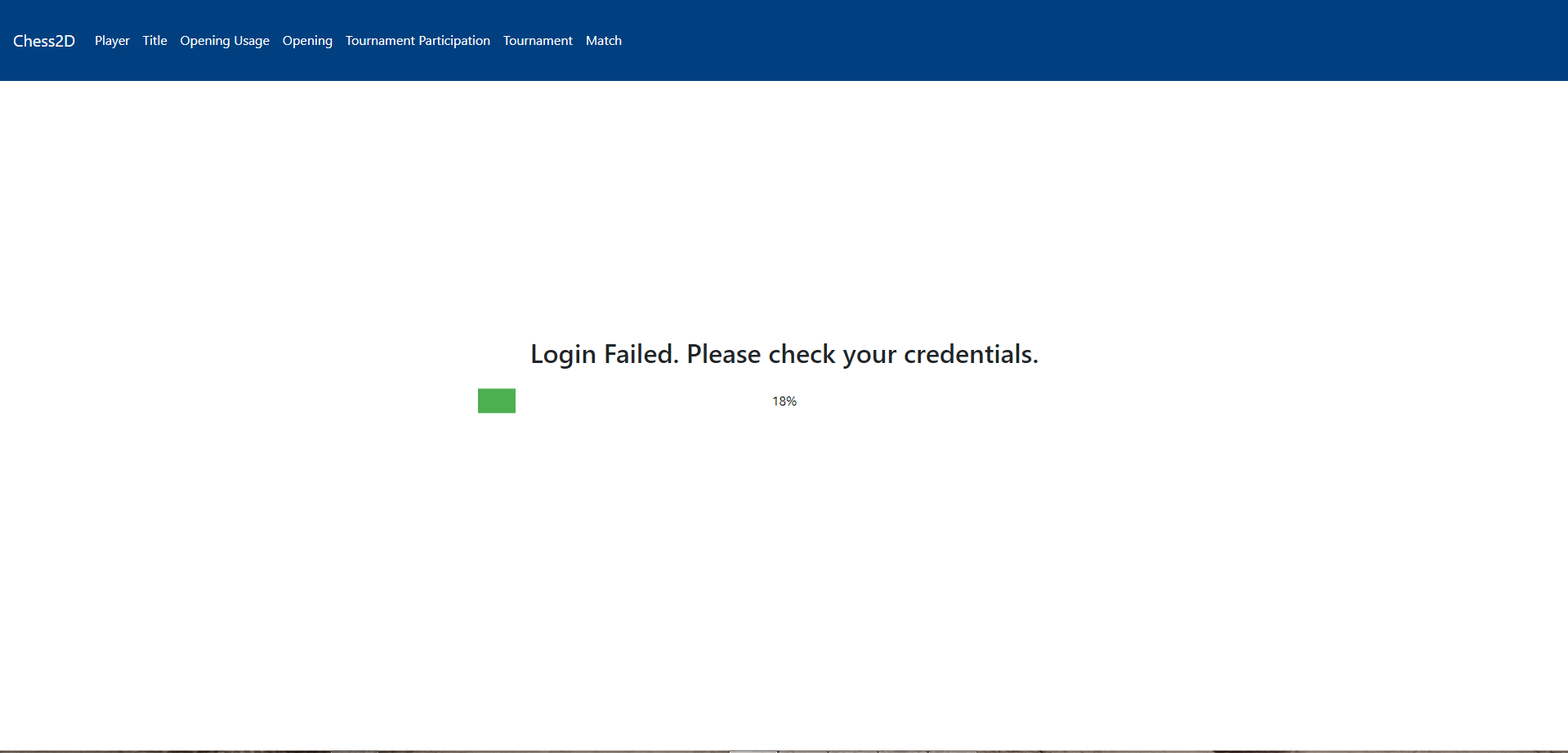


Рисунок .20 – Перевірка невдалого входу до системи

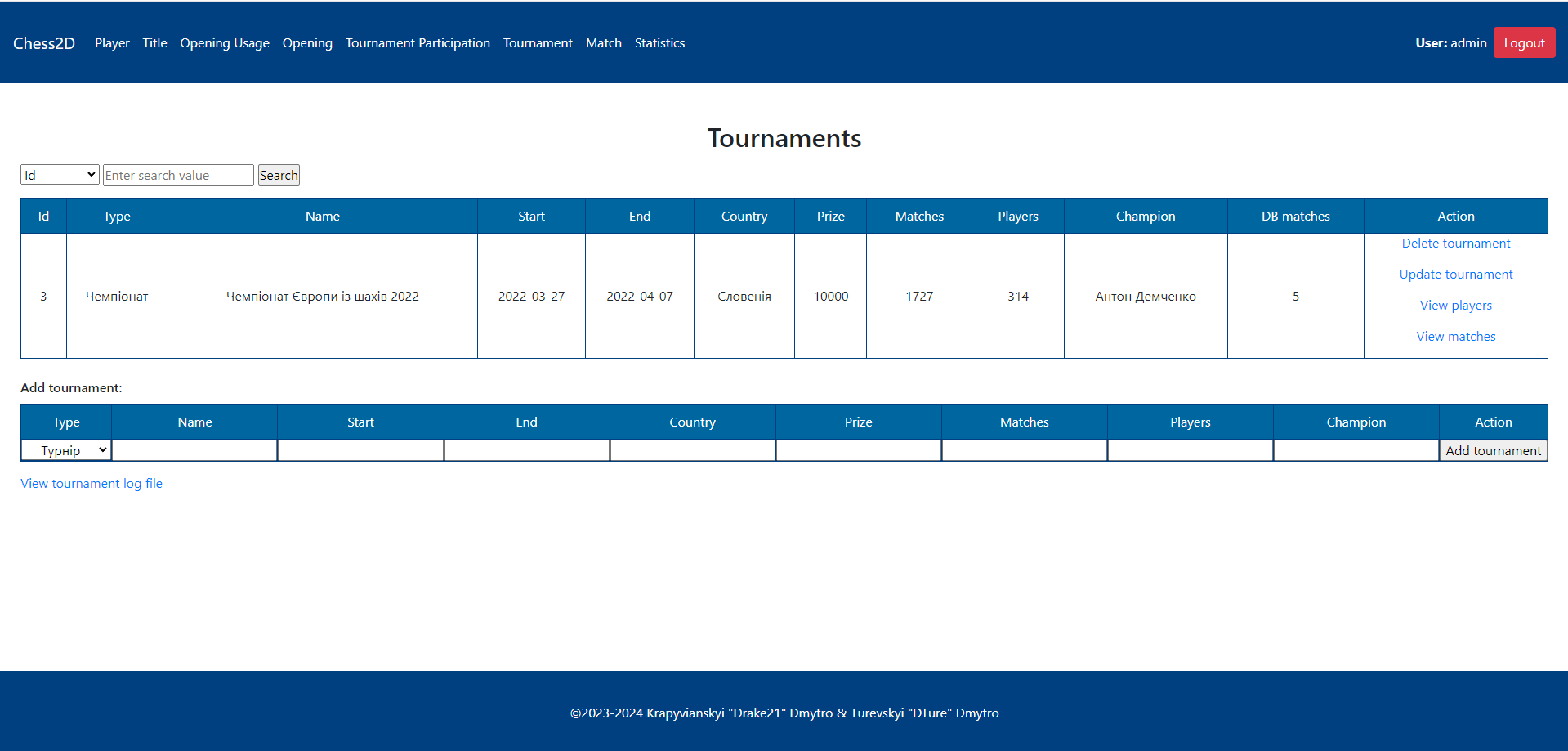


Рисунок .21 – Пошук відповідного турніру

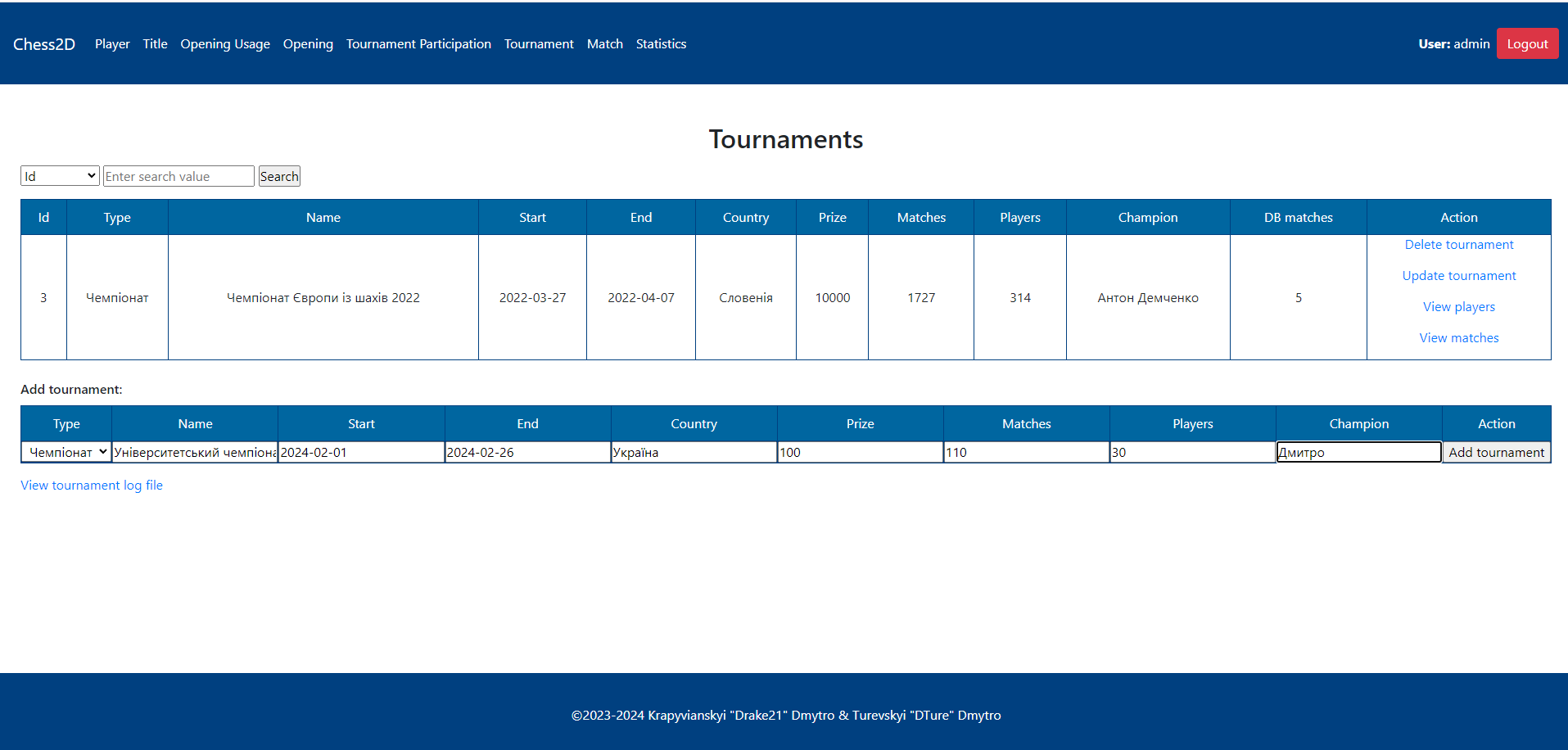


Рисунок .22 – Додавання нового турніру

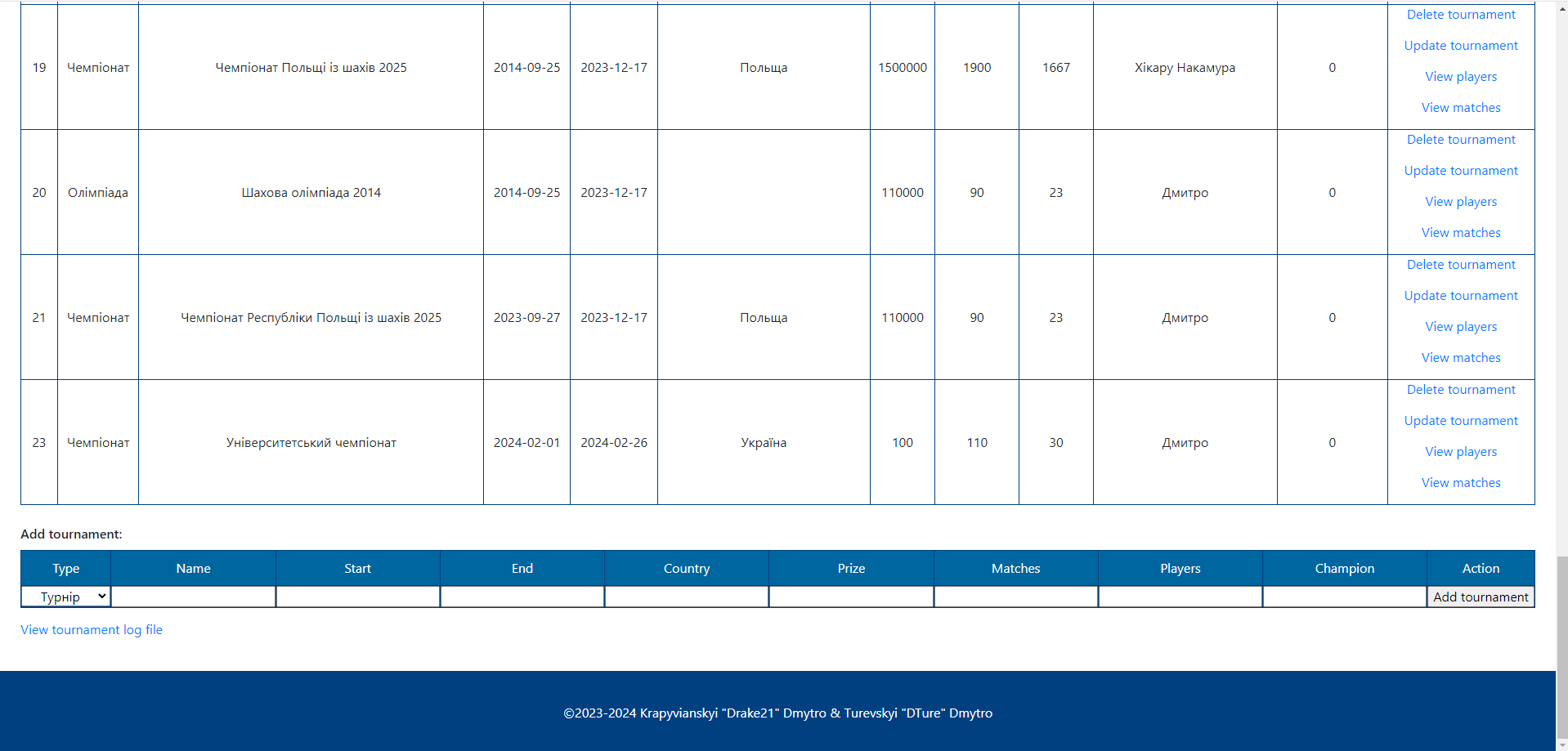


Рисунок .23 – Перевірка наявності нового турніру

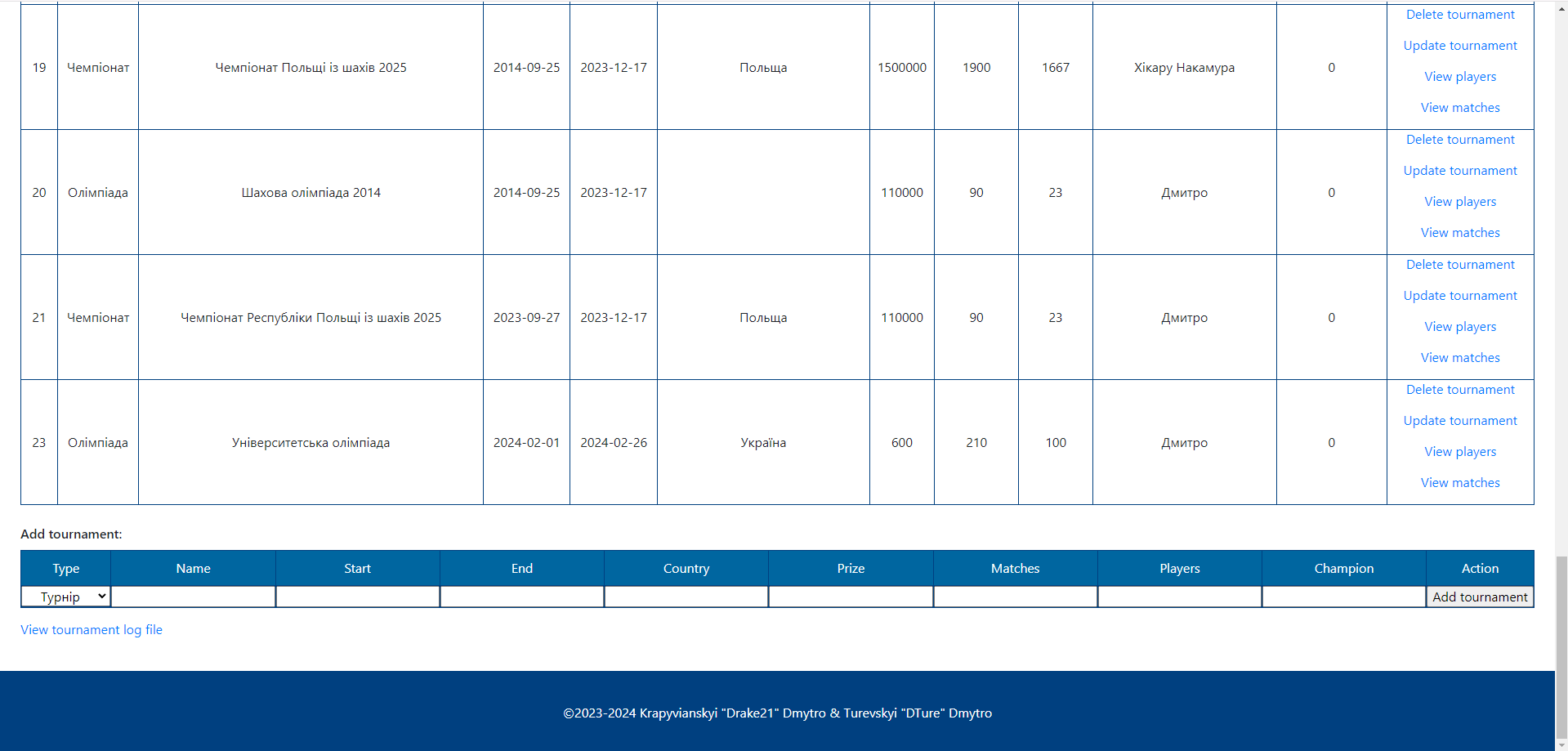


Рисунок .24 – Результат оновлення доданого турніру

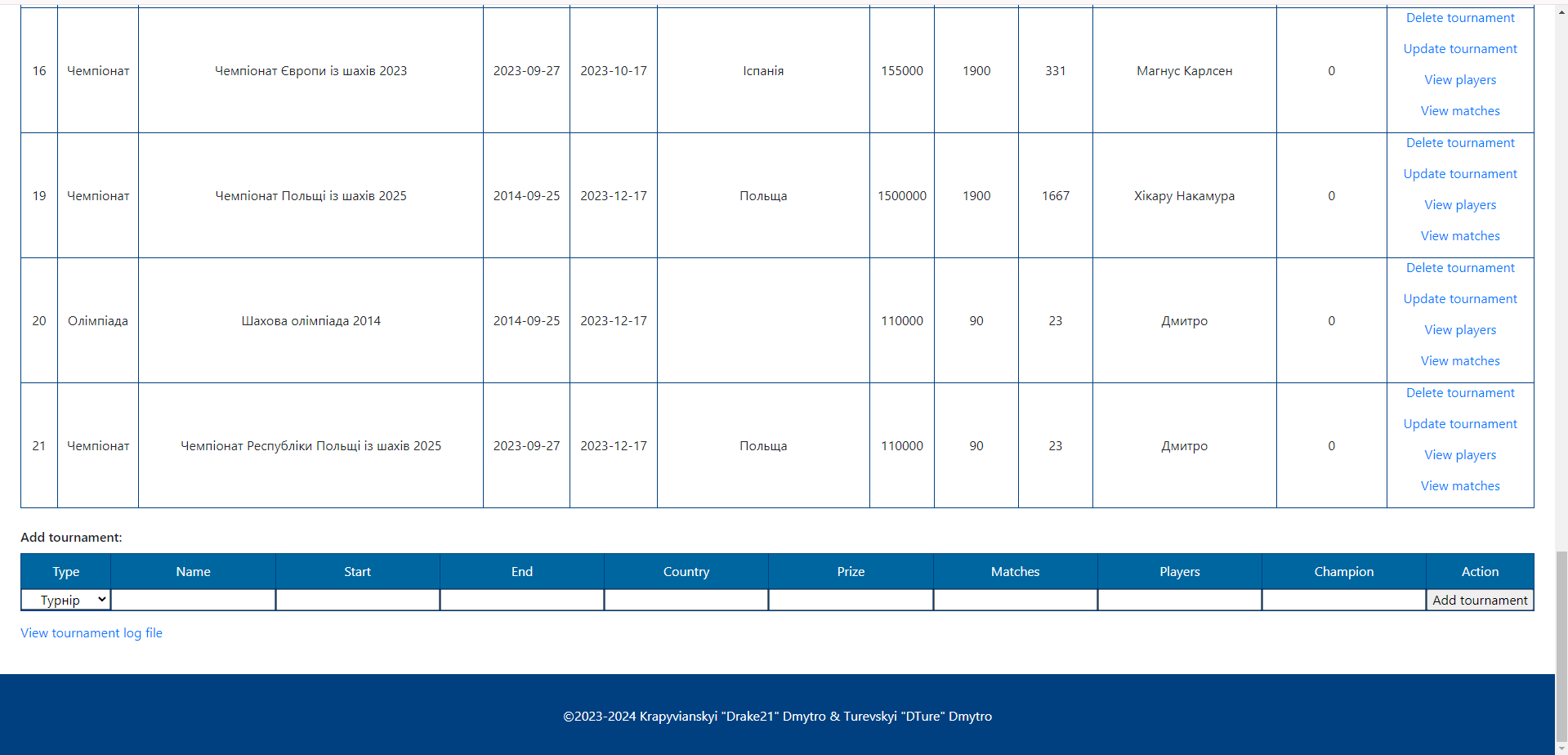


Рисунок .25 – Результат видалення доданого турніру

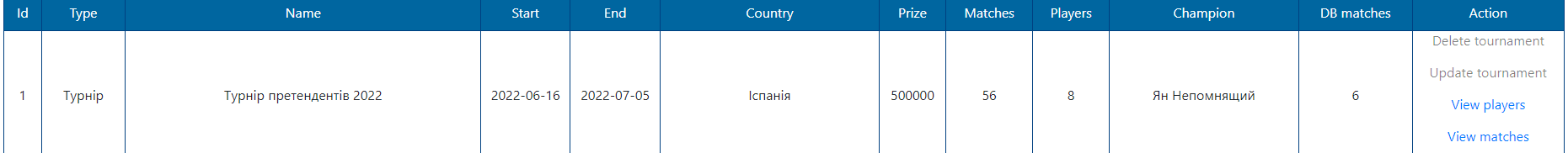


Рисунок .26 – Перевірка прав звичайного користувача

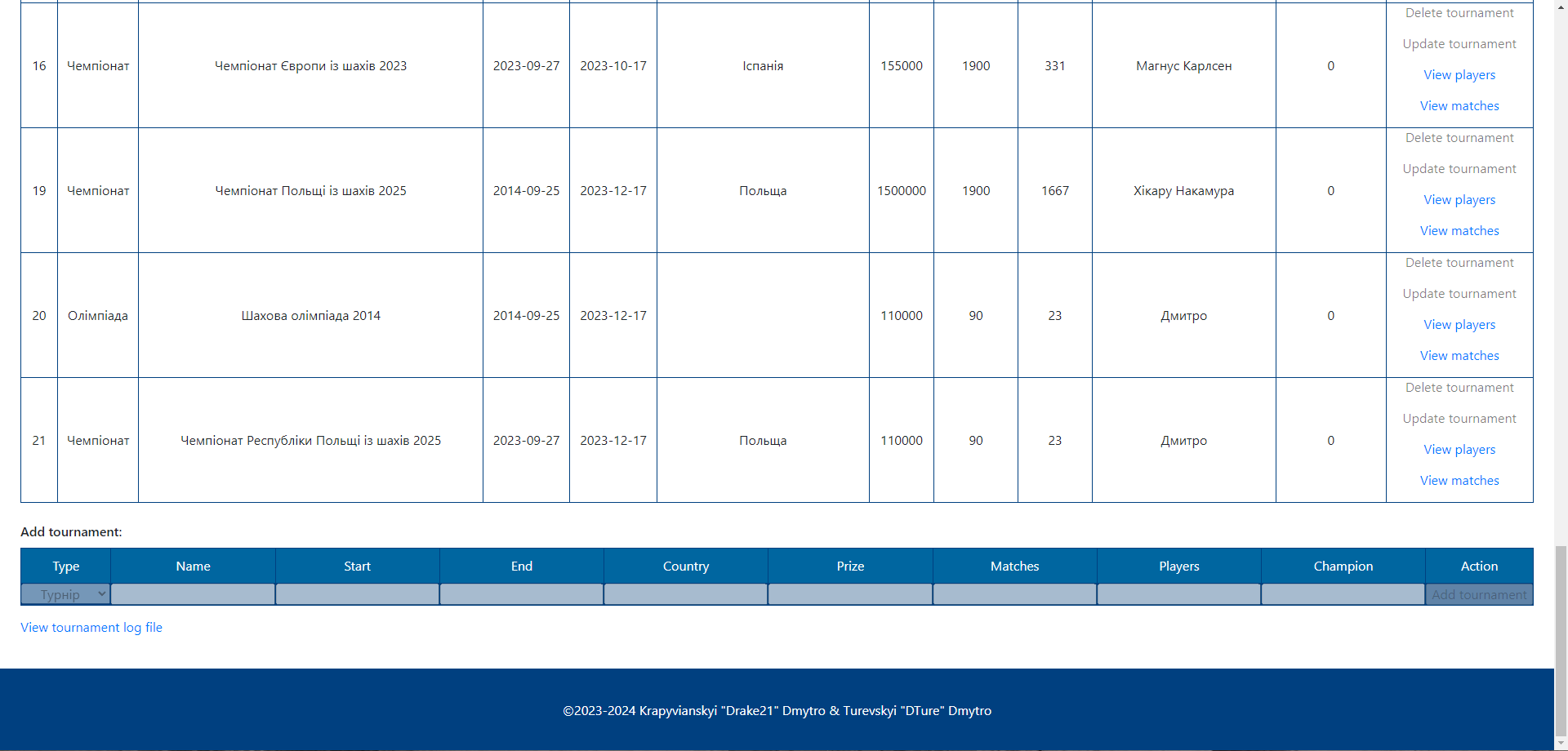


Рисунок .27 – Перевірка прав неавторизованого користувача

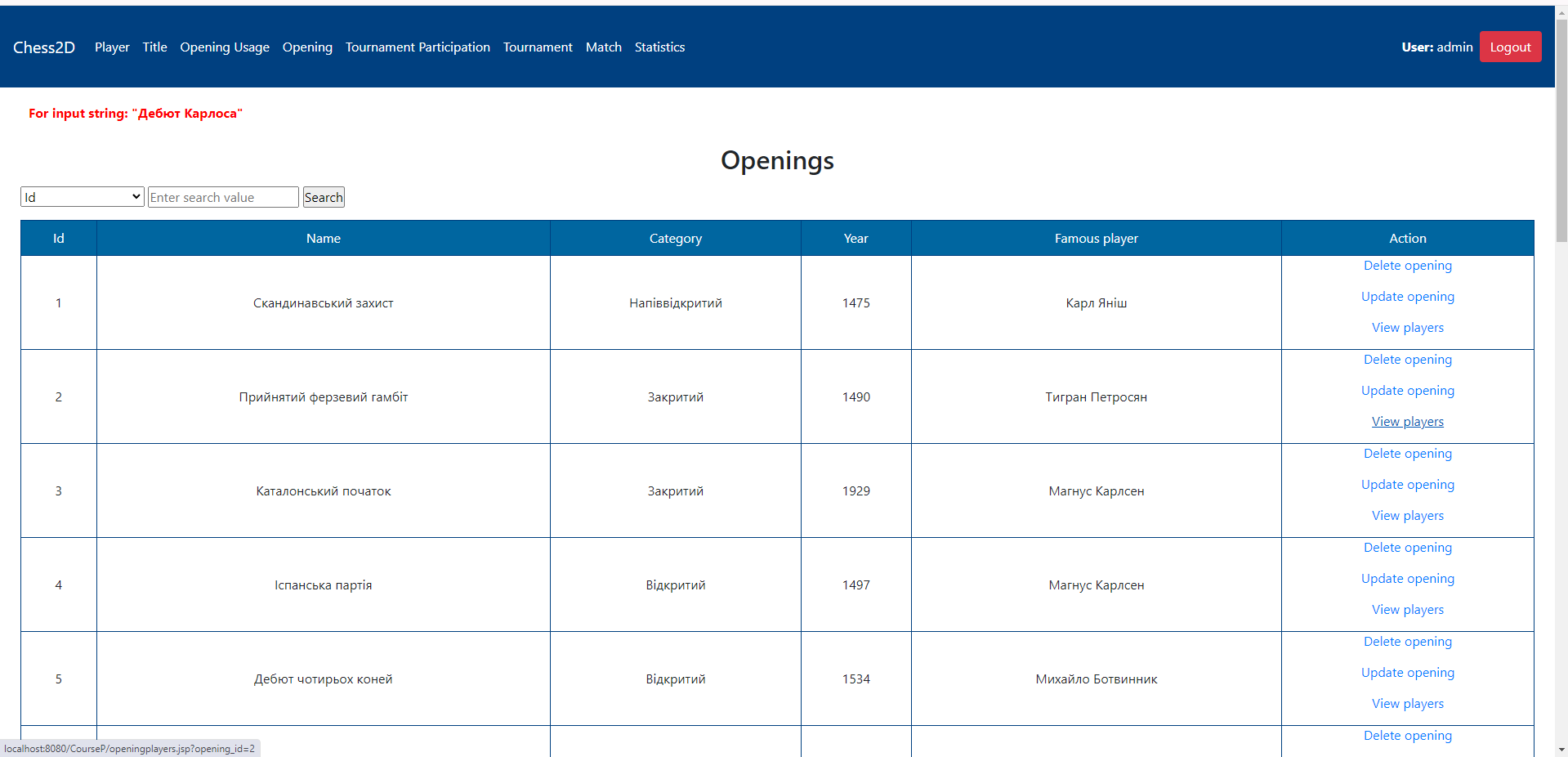


Рисунок .28 – Обробка помилки при доданні нового запису

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.