Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

РОЗРОБКА СИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ ЗМАГАНЬ З БОРОТЬБИ

Текстова частина до курсової роботи

за спеціальністю „Програмна інженерія” 6.050103

Керівник курсової роботи
ас. Яремко С.А.
(підпис)
“____” _____ 2022 р.
Виконав студент
Майстер І.С.
“____” _____ 2022 р.

Київ 2022

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія Програмного Забезпечення»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“10” жовтня 2021 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Майстеру Івану факультету інформатики 3-го курсу навчання
ТЕМА Розробка системи для організації змагань з боротьби_____
Вихідні дані:

- Серверна частина застосунку з основною логікою
- Клієнтська частина для учасників, тренерів та суддів

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Анотація
Вступ
1 Аналіз предметної області.
2 Проектування системи.
3 Реалізація системи.
Висновки
Список літератури

Дата видачі “10” жовтня 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	10.10.2021	
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	24.10.2021	
3.	Складання плану кваліфікаційної роботи та узгодження з науковим керівником	07.11.2021	
4.	Написання розділів роботи	01.03.2022	
5.	Проміжний контроль виконання роботи	01.02.2022	
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	29.03.2022	
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	25.01.2022	
	Розділ 2 (аналітично-дослідницька частина)	01.03.2022	
	Розділ 3 (проектно-рекомендаційна частина)	29.03.2022	
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	01.06.2022	
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	07.06.2022	
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК	

Графік узгоджено 10 жовтня 2021 р.

Науковий керівник Яремко Соломія Андріївна

Виконавець курсової роботи Майстер Іван Сергійович

ЗМІСТ

Анотація

У даній роботі розглядаються загальні проблеми пов’язанні із використанням технічних засобів під час проведення спортивних змагань. На основі результатів дослідження був спроектований та реалізований застосунок, який задовольняє усі потреби користувачів на змаганнях, причому як для підготовки, самого проведення та навіть аналізу результатів спортсменів після змагань, що дає змогу не обмежувати користування сервісу лише на час змагань. Функціонал застосунку розділений за ролями та має авторизацію та аутентифікацію.

Ключові слова: застосунок, спорт, змагання, спортсмен, електронна система, користувач, поєдинок.

Вступ

Час плинний, а розвиток – вічний, особливо у сучасному світі. Щодня з’являються все нові технології, нові застосунки, багато повсякденних речей вже автоматизовано. Тому сучасне суспільство неможливо уявити без усіляких роботів чи девайсів, здатних застосовуватися на практиці для пришвидшення чи удосконалення різних процесів. Але у той же час люди повинні залишатися активними, саме тому досі є популярними різні види спорту та можливість змагатися один з одним.

Перші спортивні змагання з’явилися близько 1000 років тому, але вони так само розвиваються: змінюються правила, збільшується рівень підготовки спортсменів, а також, зменшується кількість «людських» помилок під час проведення певних змагань за допомогою інтеграції існуючих інформаційних систем та технологій.

Дана робота присвячена технічному розвитку у сфері спорту, а саме автоматизації проведення різних спортивних змагань.

Метою дослідження є створення технічних інструментів, що сприятимуть розвитку проведення змагань та застосуванню їх на практиці задля спрощення цього процесу та економії часу.

Постановка задачі:

1. Виконати аналіз процесу підготовки та проведення змагань та визначити їх недоліки.
2. Зробити огляд відомих технічних застосунків, які використовуються для різних типів змагань.
3. Спроектувати власну систему для забезпечення спортивних змагань на різних рівнях підготовки. Щоб краще проаналізувати проблематику та рішення, обмежити тему спорту бойовими мистецтвами.

4. Розробити демоверсію власного застосунку на основі отриманих даних під час аналізу та проектування. Зробити висновки щодо можливостей застосування розробленої системи.

1 Аналіз предметної області

1.1 Призначення системи

Початково спортивні змагання з бойових мистецтв проводилися без будь-яких технічних засобів та електронних систем.

Бокові суді мали стежити за поєдинком та рахувати кількість влучань кожного спортсмена, і далі після завершення поєдинку результат, записаний на листочку, віддавався керівнику килима.

Рефері стежив за дотримуванням правил поєдинку та втручався у разі необхідності при порушення цих правил. Після зупинки поєдинку призначав штрафні санкції для одного або обох спортсменів.

Керівник килиму слідкував за часом поєдинку та підраховував кількість штрафів, які призначив рефері. Після завершення поєдинку керівник килиму отримував результати поєдинку від бокових судів на листочку, підраховував загальний рахунок поєдинку та визначав переможця. Результат поєдинку записувався у бланк відповідної категорії та, після завершення фінального поєдинку, віддавався організаторам змагань для внесення їх у протокол та подальшого нагородження спортсменів.

За допомогою сучасних інформаційних засобів всі ці процеси можна автоматизувати. Тепер підрахунок балів бокового суді відбувається при натисканні на клавішу джойстика. Сигнал одразу передаються на комп’ютер закріплений за цим килимом і бал відображається на екрані комп’ютера та головному екрані.

Зауваження та штрафи, які вносить рефері, фіксуються головою килиму одразу у комп’ютері, і так само відображаються на обох екранах.

Час рахується на комп’ютері одразу від початку поєдинку, та після завершення часу результат поєдинку виводиться на екран автоматично і паралельно зберігається у базі, якою керують організатори змагань.

Організатори відслідковують за результатами онлайн на іншому комп’ютері, а далі використовують ці дані для нагородження.

Також перед початком змагань система генерує розклад поєдинків, тому організатори роздруковують та оприлюднюють цей розклад. Деякі системи дають можливості для перегляду розкладу онлайн.

1.2 Огляд аналогів

Аналіз ринку показав, що насправді існує багато застосунків, що допомагають у вирішенні питання проведення різноманітних спортивних змагань, проте більшість з них не є універсальними для будь-якого виду спорту.

Найбільшу частку займають системи для проведення футбольних турнірів, далі йдуть більш універсальні програми – для командних видів спорту з м’ячем, і лише невелику кількість - для проведення змагань з усіх інших видів спорту, або хоча б певну частину з них. Розглянемо трохи детальніше декілька аналогів та проаналізуємо їх.

1.2.1 «Исток-Турнир»

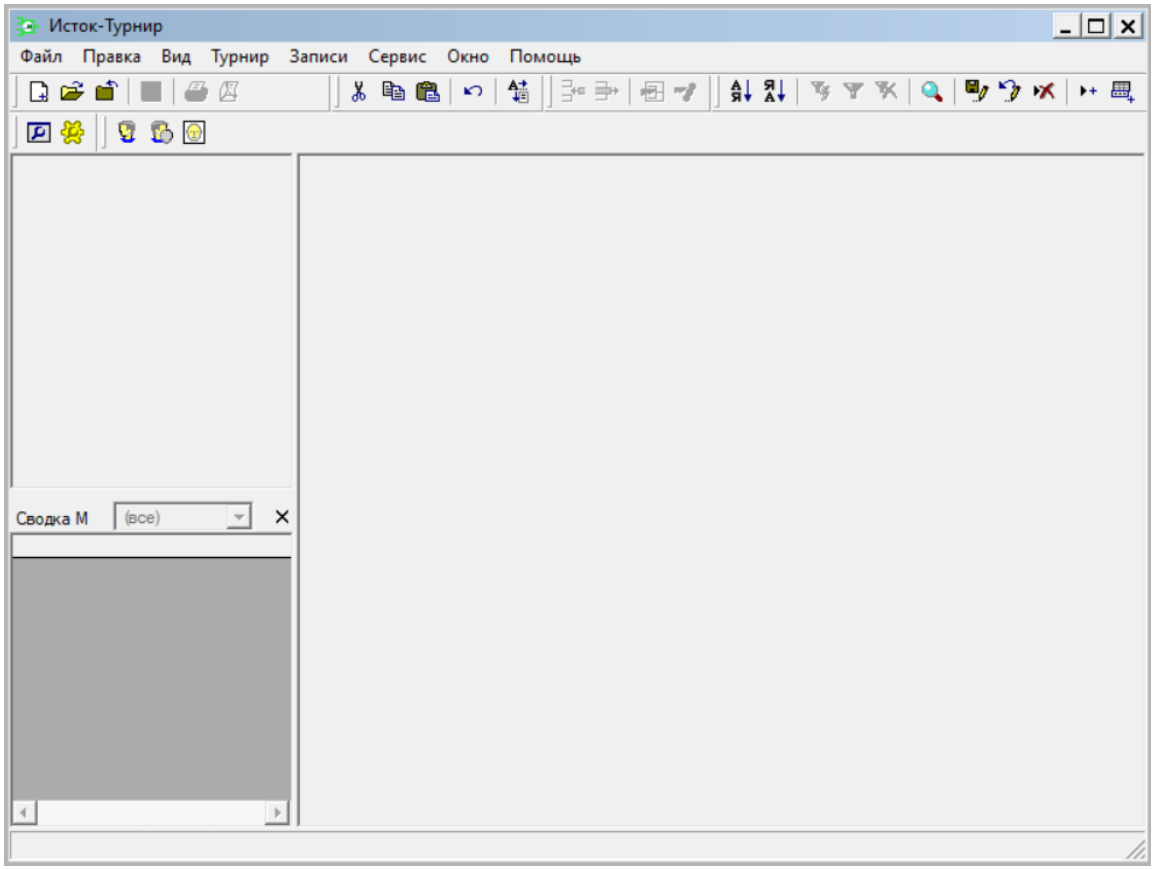

Рисунок 1.1 – Головна сторінка та меню застосунку «Исток-Турнир»

Програма призначена для організаторів, головних суддів та керівників, які проводять турніри міжнародного, місцевого чи клубного рівня серед дорослих та дітей з ігрових та бойових видів спорту. Реалізовані режими особистих та командних змагань.

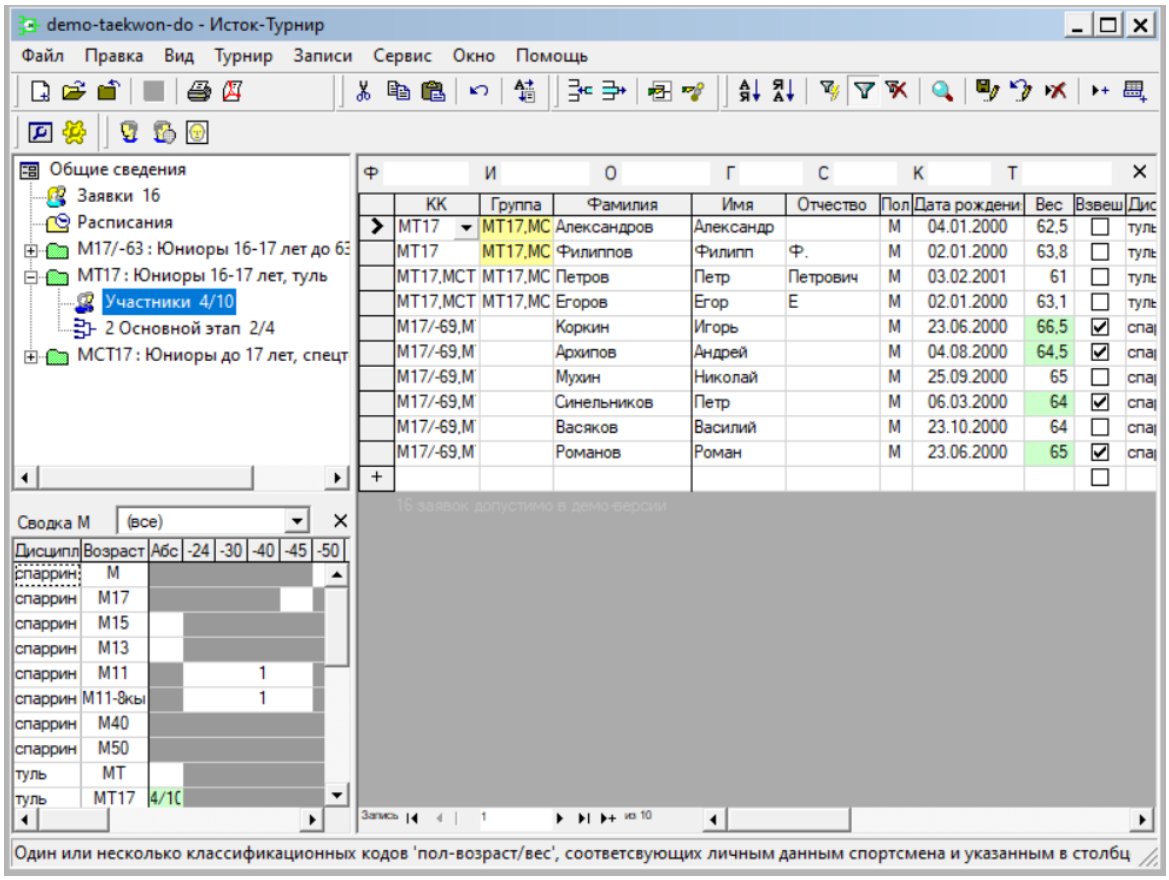

Рисунок 1.2 – Сторінка з групами застосунку «Исток-Турнир»

Для огляду можливостей застосунку використовується демоверсія, де при першому вході у програму пропонується обрати вид спорту і далі створити або новий турнір, або використати вже створений тестовий. Загалом у програмі реалізовано багато можливостей для реєстрації спортсменів, підготовки змагань та різних налаштувань системи. Проте слід зауважити, що інтерфейс програми досить незручний та навіть незрозумілий, тому на початку важко розбиратися із тим, що і як працює.

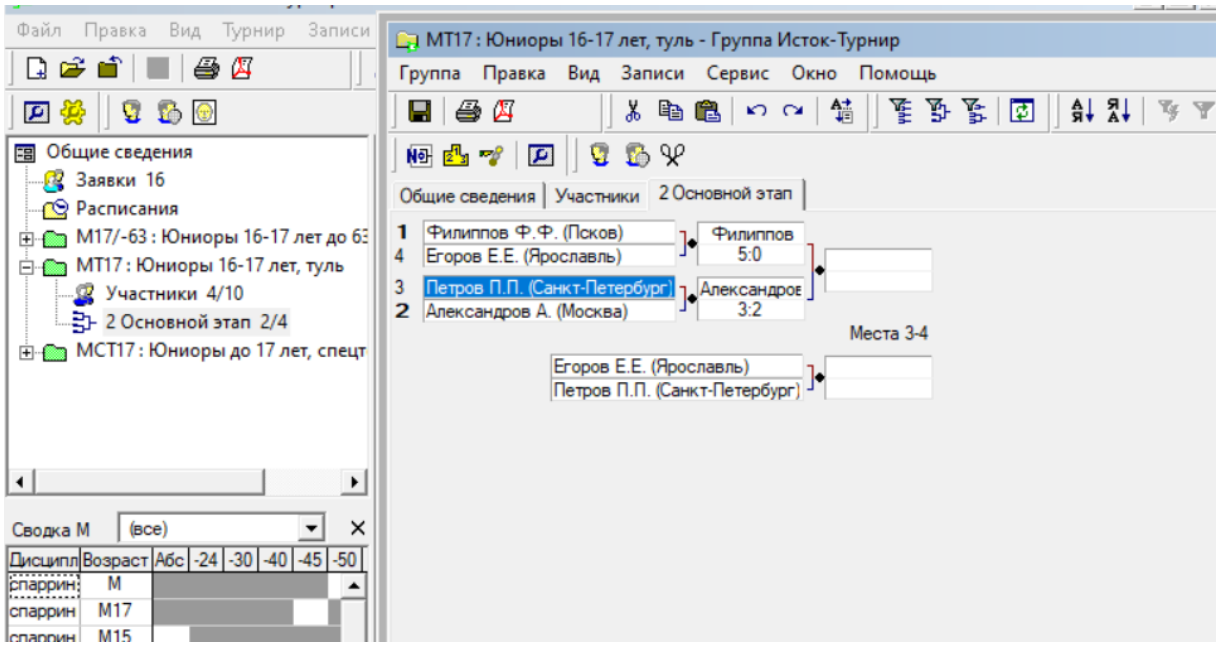

Рисунок 1.3 – Сторінка поєдинків застосунку «Исток-Турнир»

Також застосунок забезпечує безпосередньо етап проведення змагань та розподіл місць учасників, але лише завдяки ручному вводу усіх результатів поєдинків.

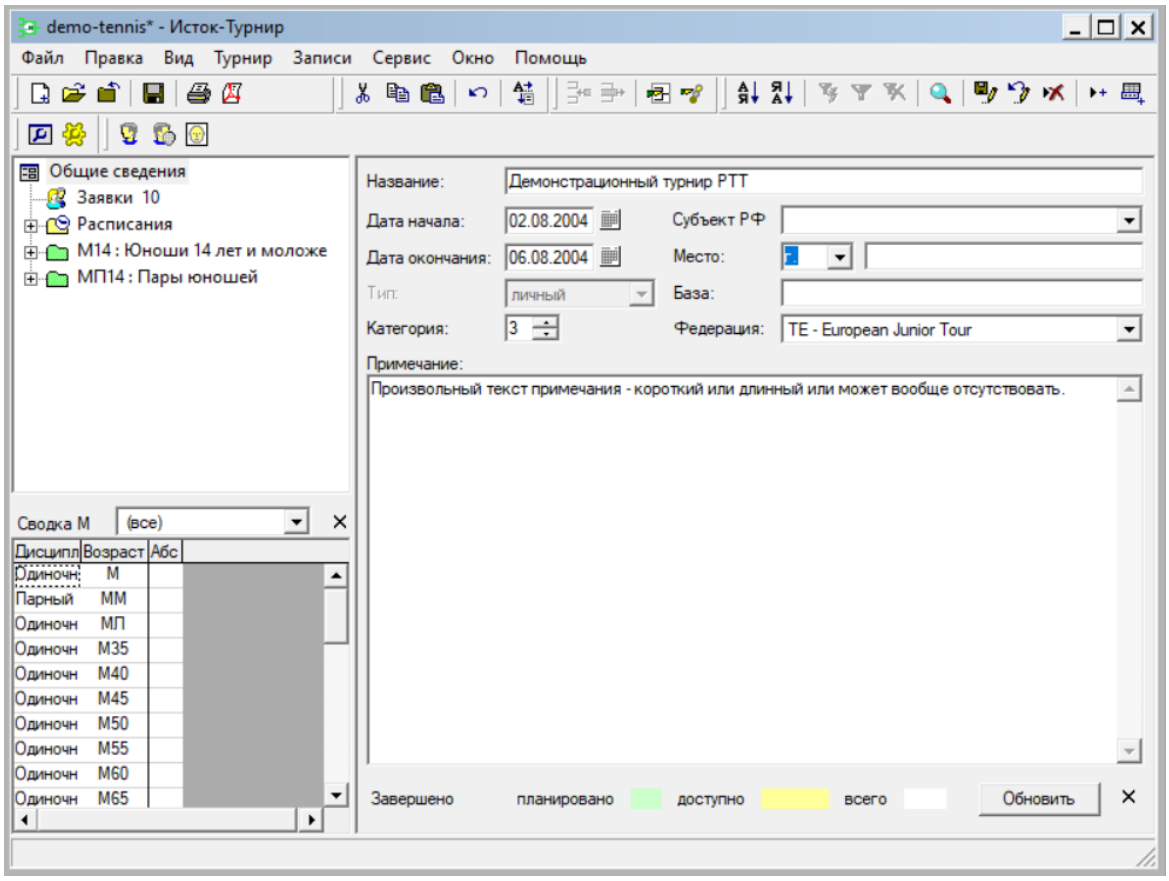

Рисунок 1.4 – Сторінка іншого виду спорту застосунку «Исток-Турнир»

Для створення турніру з іншого виду спорту, надається список можливих видів, і у випадку цієї зміни з’являється попередження про те, що інтерфейс має змінитися. І як ми бачимо на Рис. 1.4, при створенні турніру з тенісу меню дійсно отримало певні зміни.

1.2.2 Tournament Assistant

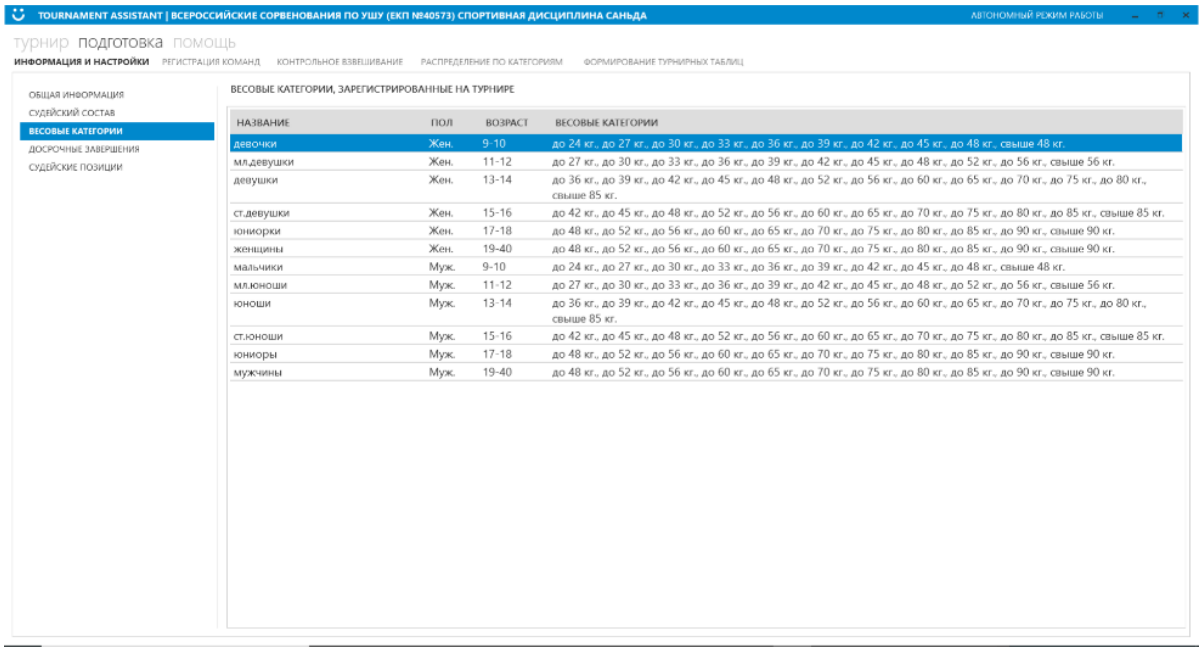

Рисунок 1.5 – Головна сторінка застосунку Tournament Assistant

Програма призначена для організації турнірів різного рівня серед дорослих та дітей лише з бойових видів спорту. Передбачено режим особистих виступів, режим для проведення командних виступів – не реалізовано.

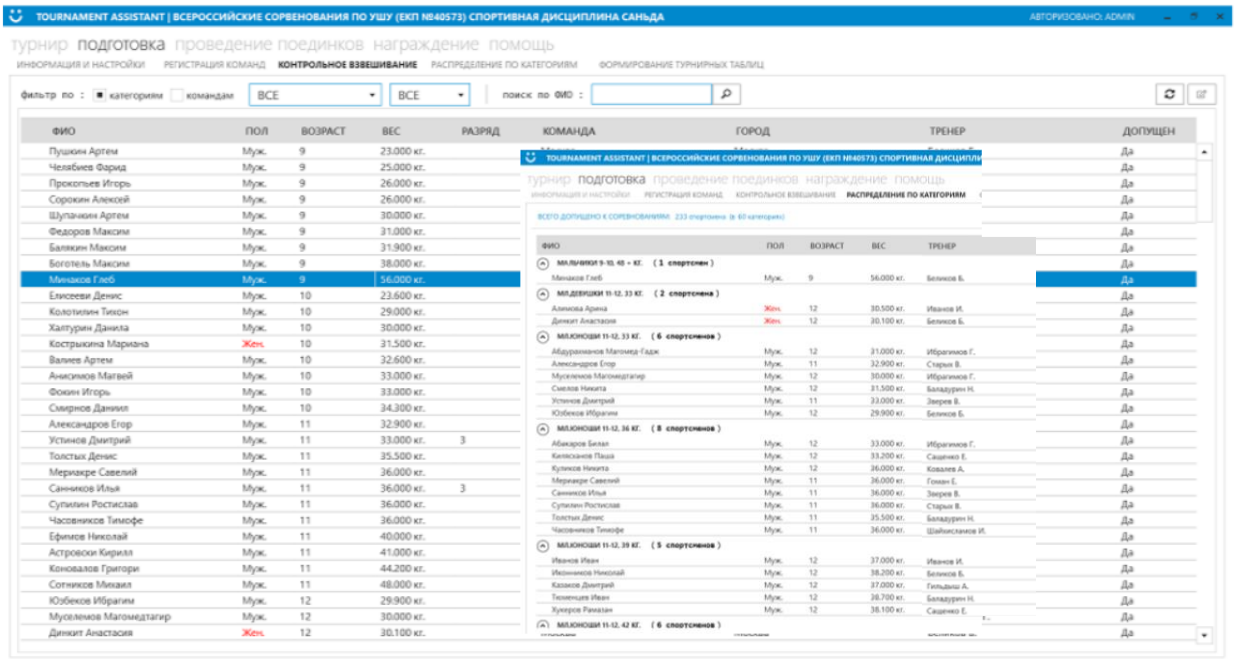

Рисунок 1.6 – Сторінки підготовки змагань застосунку Tournament Assistant

Загалом усі основні можливості для підготовки та проведення змагань наявні, є можливість створення нових турнірів, реєстрація команди та учасників, окремий функціонал для проведення зважування та жеребкування у категоріях. Найголовніша відмінність від попереднього застосунку є значно зручніший і зрозумілий інтерфейс та дизайн самої програми, що робить її, по-перше, простішою для опанування та використання, по-друге, візуально приємнішою та, по-третє, навіть уявно більш функціональною.

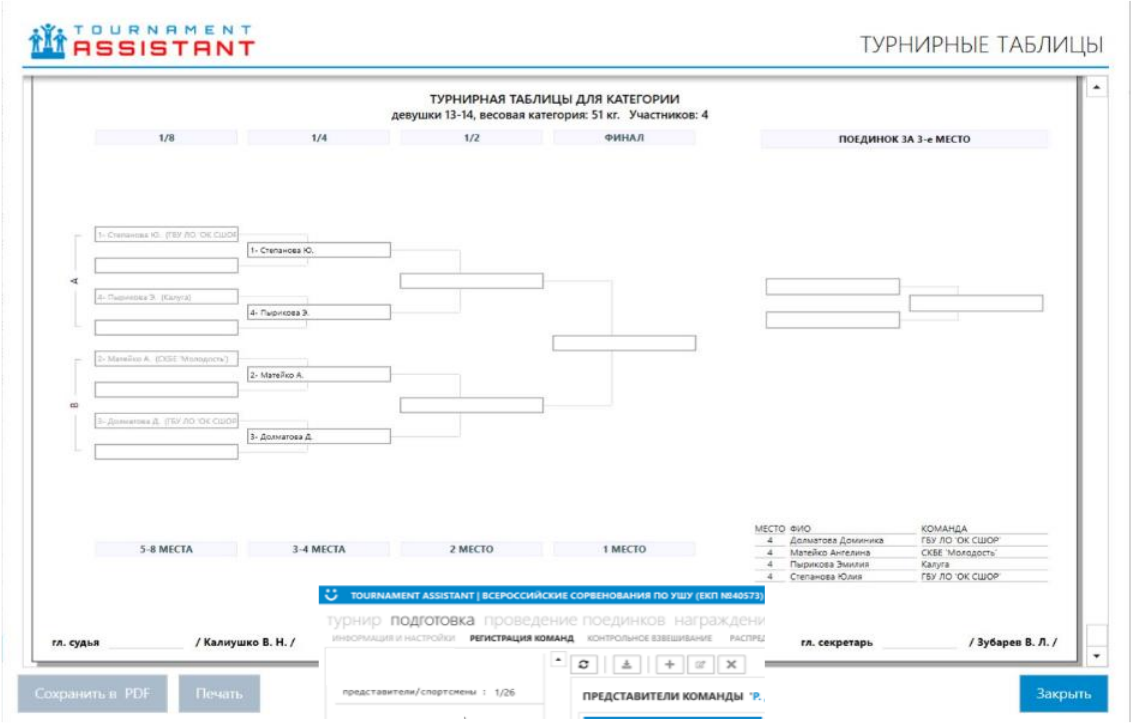

Рисунок 1.7 – Сторінка поєдинків застосунку Tournament Assistant

У застосунку так само є можливість для автоматизування процесу проведення боїв та збереження результатів, проте сам рахунок поєдинку вноситься вручну. Також варто відзначити реалізований засіб для подання заявки файлом та зворотній - отримувати файл із результатами кожної категорії з можливістю відправки на друк для нагородження.

1.2.3 Tournament Planner

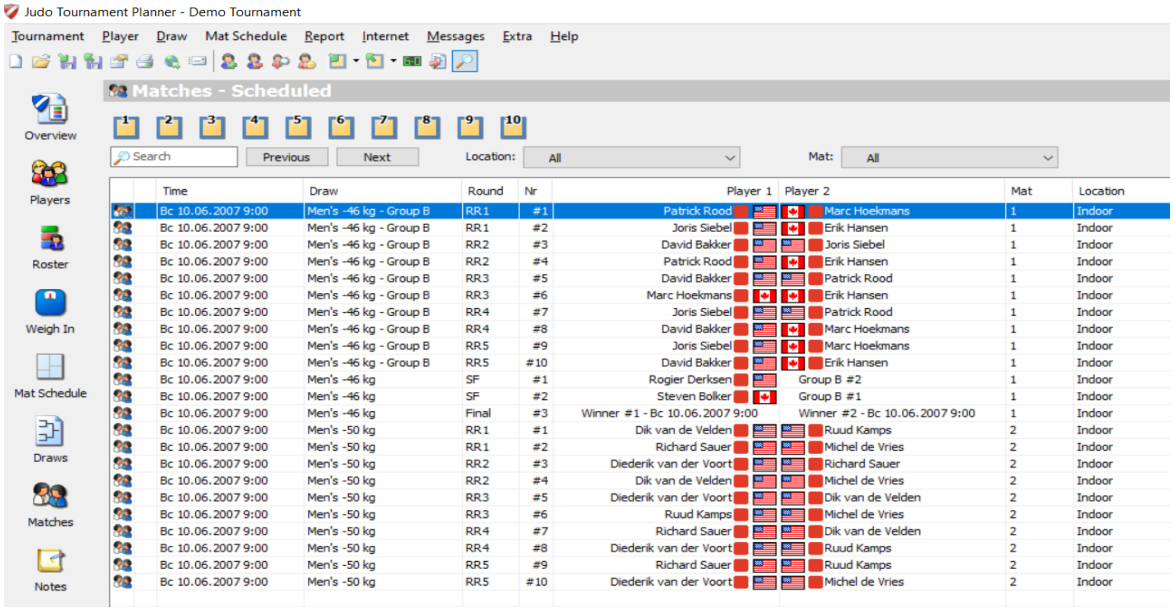

Рисунок 1.8 – Сторінка розкладу поєдинків застосунку Tournament Planner

Програма є досить схожим аналогом до попереднього застосунку, проте навпаки для ігрових видів спорту. Кардинальних відмінностей у функціоналі нема, так само є реєстрація, жеребкування, фіксування результатів кожного окремого матчу та навіть система для зважування , що є зайвим для ігрових видів спорту, тому не зрозуміло чи це технічна помилка при розробці, чи навпаки додаткова можливість для потенційного розширення списку видів спорту для яких підходить дана система.

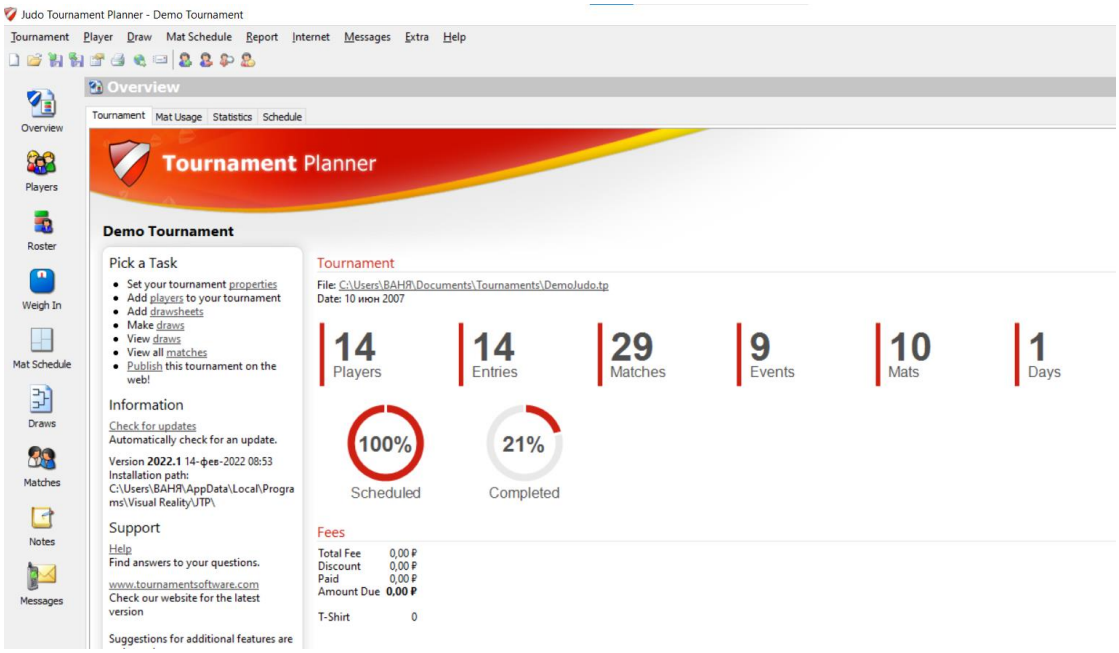

Рисунок 1.9 – Інформаційна сторінка застосунку Tournament Planner

Єдиною корисною відмінністю є реалізація для перегляду підрахованої інформації та статистики змагань, причому навіть у режимі реального часу. Такий інструмент є досить потужним для вирішення проблем підрахунку часу проведення змагань та розкладу поєдинків.

Загалом застосунок є чимось середнім між першими двома аналогами.

1.2.4 Електронна система Таеквон-до ІТФ

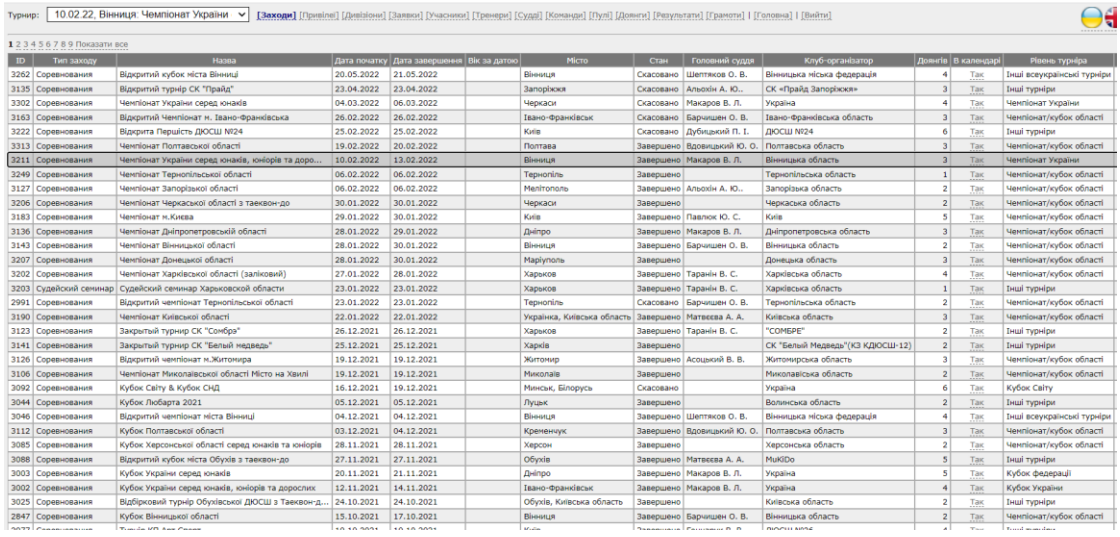

Рисунок 1.10 – Головна сторінка застосунку

Даний веб-застосунок є електронною базою для керування та проведення усіх змагань з таеквон-до ІТФ в Україні. Інтерфейс програми так само не найзручніший, проте базовий функціонал повністю реалізований. Найбільша відмінність – розподіл можливостей за ролями, який існує всього три: спортсмен, тренер, адміністратор.

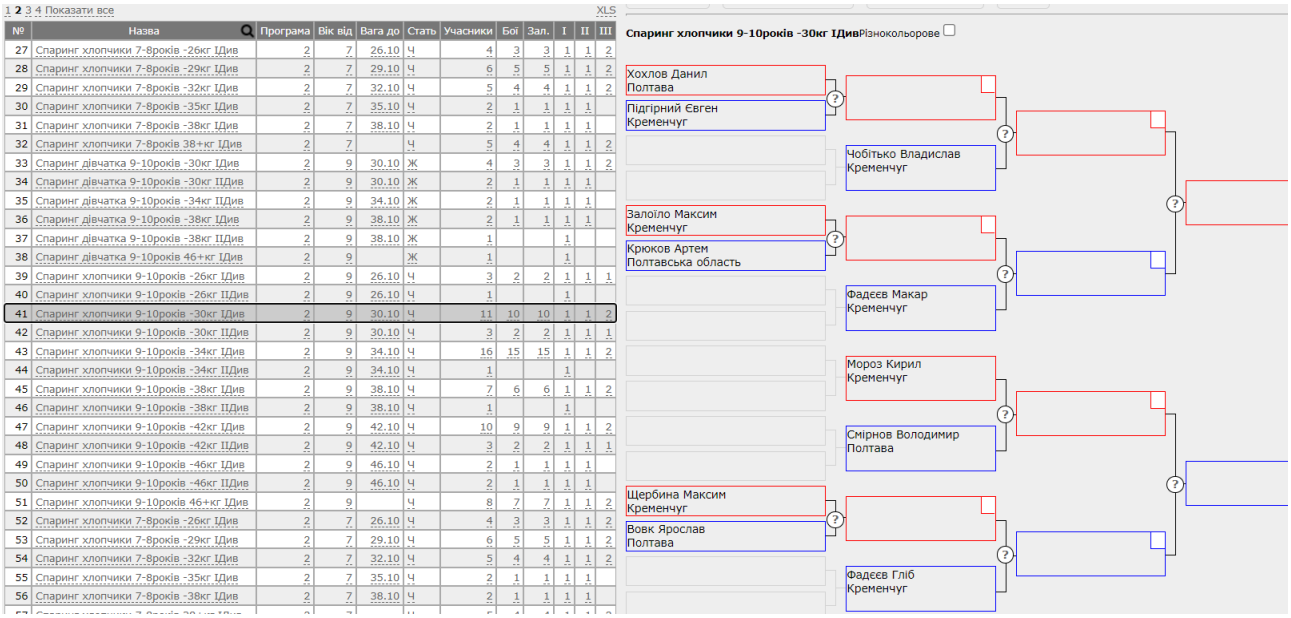

Рисунок 1.11 – Сторінка поєдинків застосунку

Усім типам користувачів надається доступ до перегляду усіх сторінок застосунку, де можна знайти інформацію про турніри, поточний етап підготовки, учасників та пулі (пулі – від англ. pool – група, сукупність).

Тренер має додаткові можливості для додавання нових спортсменів, редагування їх інформації та створення нових заявок для змагань.

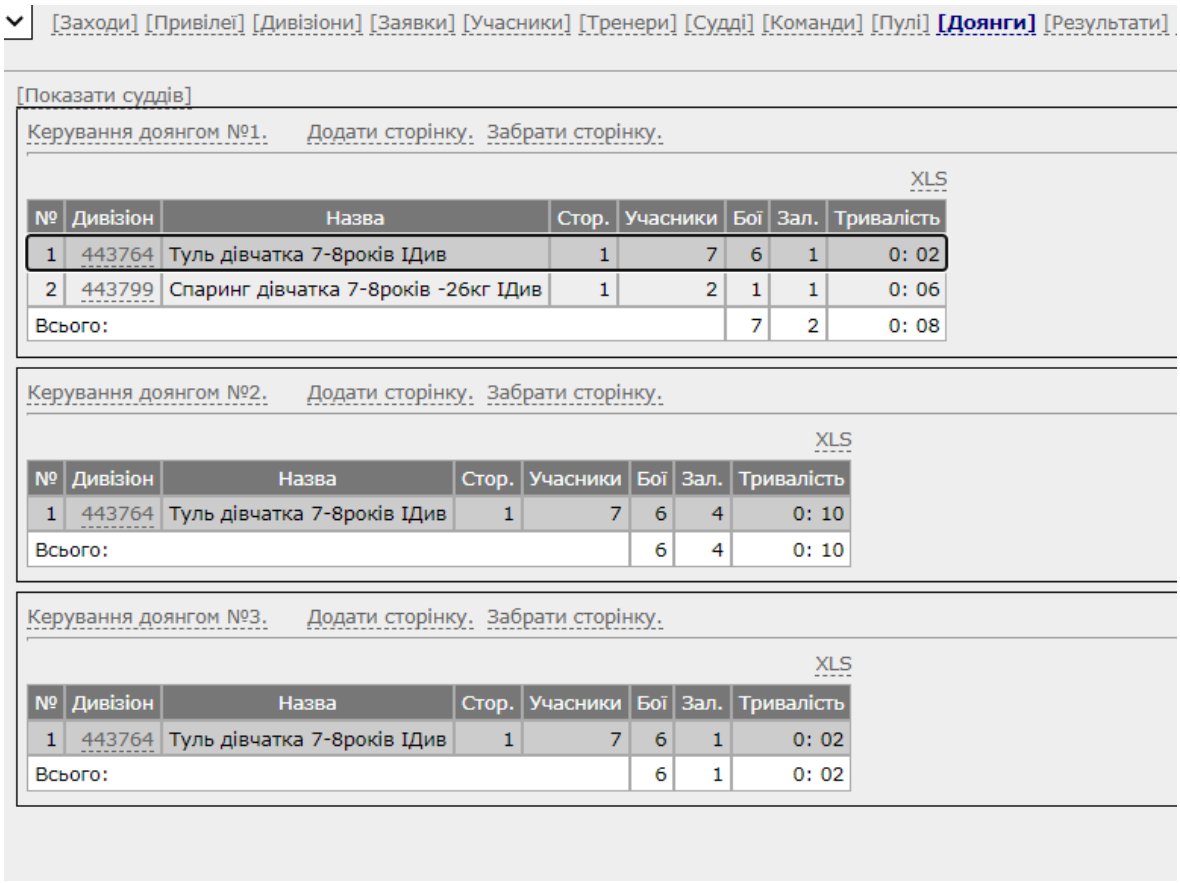

Рисунок 1.12 – Сторінка розкладу поєдинків

Адміністратор керує змаганнями, тому у нього додаткові можливості у вигляді створення нових турнірів, підтвердження заявок від тренерів, створення та розподіл пулів, проведення зважування та верифікації даних та ваги учасників у базі та, найголовніше, створення поєдинків у базі для проведення самих змагань.

Загальне порівняння можливостей представлених аналогів:

Функціонал та можливості	**«Исток-Турнир»**	**Tournament Assistant**	**Tournament Planner**	**Електронна система Таеквон-до ІТФ**
Види спорту	**ігрові** (теніс, сквош, бадмінтон та інші), **єдиноборства** (бокс, боротьба, карате та інші).	**єдиноборства** (бокс, боротьба, карате та інші).	**ігрові** (теніс, сквош, бадмінтон та інші).	**таеквон-до**
Вікові категорії	**Так**	**Так**	**Так**	**Так**
Вагові категорії	**Так**	**Так**	**Ні**	**Так**
Командні змагання	**Ні**	**Так**	**Так**	**Так**
Системи проведення змагань				
Кругова	**Ні**	**Так**	**Так**	**Ні**
Олімпійська	**Так**	**Так**	**Так**	**Так**
Упорядкування учасників та складання турнірних таблиць	**Так**	**Ні**	**Так**	**Ні**
Ручне жеребкування та розстановка	**Так**	**Так**	**Так**	**Так**
Комп'ютерне жеребкування	**Так**	**Так**	**Так**	**Так**
Складання розкладу	**Ні**	**Ні**	**Так**	**Так**
Внесення результатів				
Вручну	**Так**	**Так**	**Так**	**Так**
Електронно	**Ні**	**Ні**	**Ні**	**Так**
Розподіл за ролями	**Ні**	**Так**	**Ні**	**Так**
Перегляд для інших користувачів	**Ні**	**Ні**	**Ні**	**Так**
Зручний інтеріейс	**Ні**	**Так**	**Так**	**Ні**

Таблиця 1.1 – Детальне порівняння аналогів

Після детального аналізу застосунків, ми бачимо, що чим більше видів спорту охоплює конкретна програма, тим її функціонал одразу зменшується.

Також прослідковується відсутність або, навпаки, наявність необхідного функціоналу в залежності від видів спорту, яких цей застосунок покриває. Помітно, що більшість застосунків не підтримують розподіл за ролями, але це можна пояснити тим, що підтримка таких застосунків буде набагато складніше і тому автори могли відмовитися від цього. Проте без розподілу за ролями втрачається можливість будь-якого перегляду інформації для спортсменів, що насправді є дуже потрібним для кращого планування змагань і запобіганню втрати часу.

1.3 Опитування потенційних користувачів

Було проведено опитування потенційних клієнтів або людей, які вже користувались чи працювали зі схожими програмами, для того, щоб більш детально проаналізувати вже наявні проблеми та можливі вдосконалення, які стануть в нагоді.

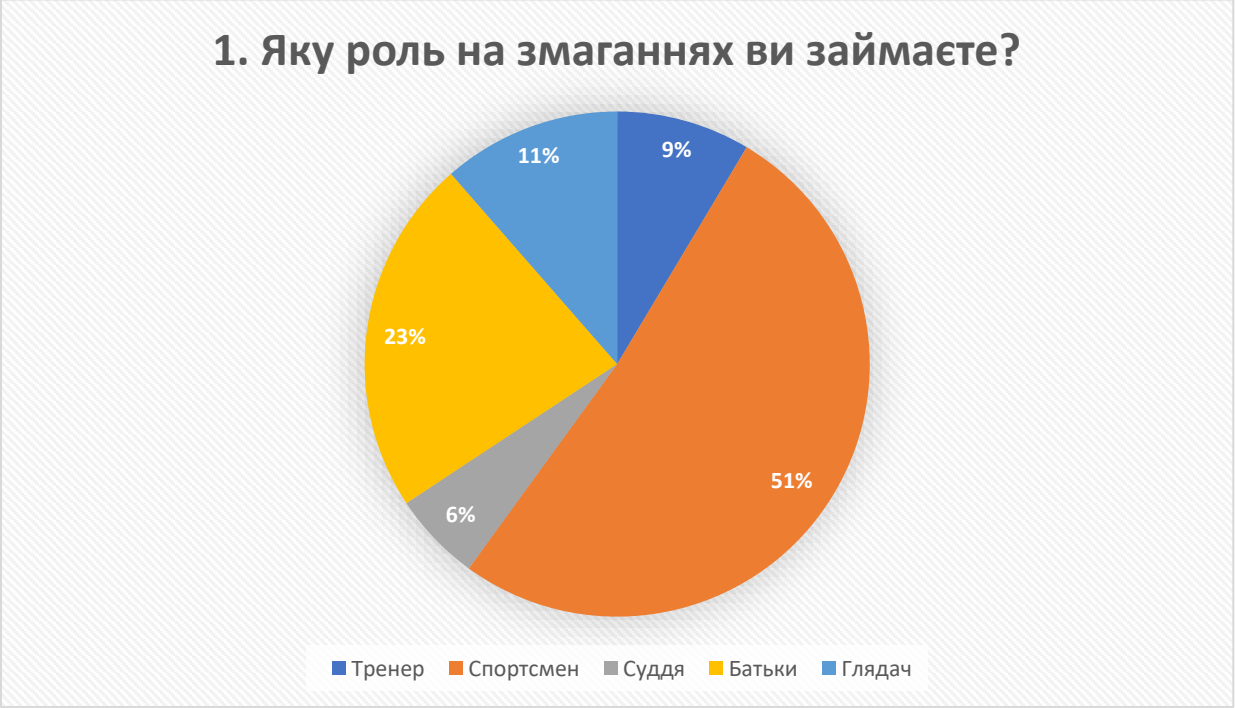

Рисунок 1.13 – Діаграма результатів першого питання

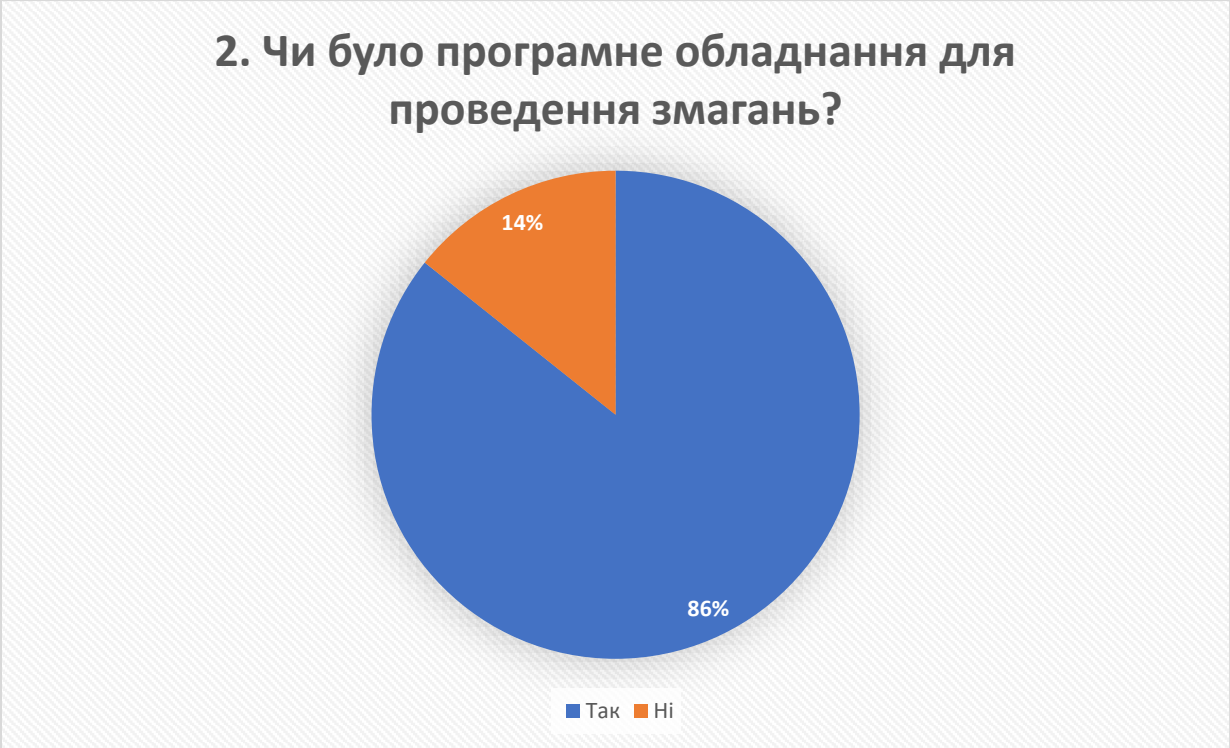

Рисунок 1.14 – Діаграма результатів другого питання

Рисунок 1.15 – Діаграма результатів третього питання

Рисунок 1.16 – Діаграма результатів четвертого питання

Рисунок 1.17 – Діаграма результатів п’ятого питання

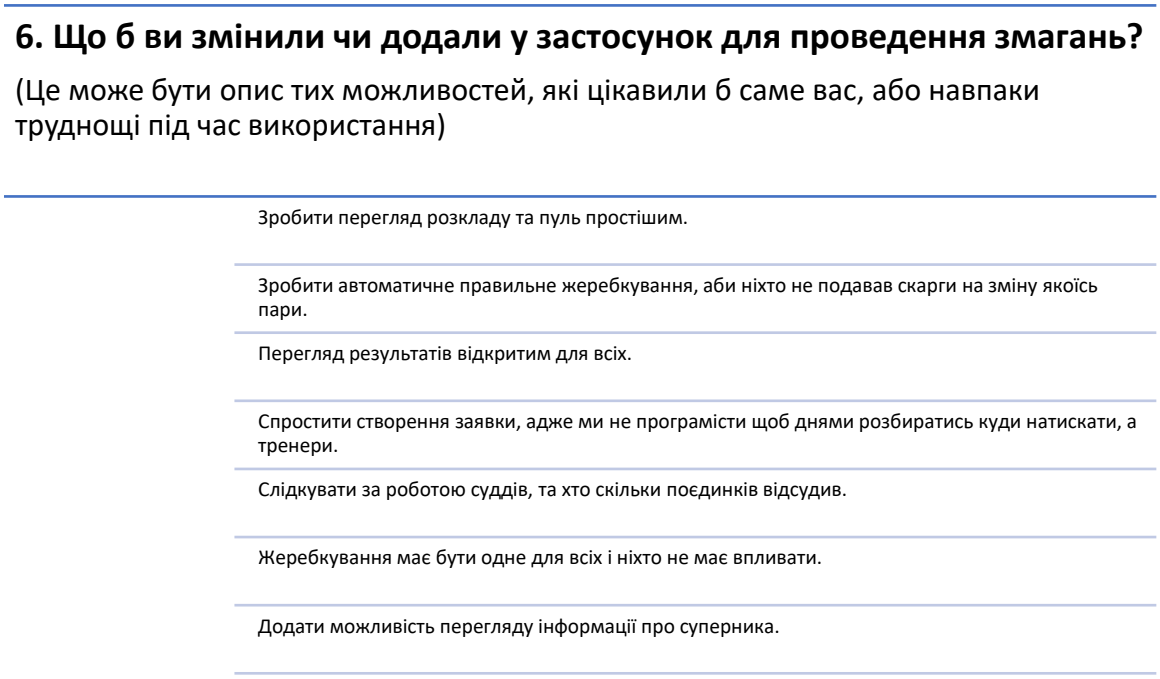

Рисунок 1.18 – Діаграма результатів шостого питання

Після проведення опитування користувачів, серед яких були присутні різні типи користувачів, з’явилася нова цікава інформація.

Досі існують змагання, які проводяться без залучення технічного обладнання, також з’ясувалося, що очевидною проблемою для користувачів є відсутність або обмеженість перегляду потрібної інформації, пов’язаної з їх виступами. Тобто частина спортсменів потребує отримувати більше інформації.

Також багато користувачів скаржилися на незрозумілість та незручність інтерфейсу застосунку, навіть якщо він є. Це свідчить про те, що не всі користувачі використовують максимум функціоналу свого застосунку.

Цікаво, що велика частина спортсменів не отримує базової статистики своїх виступів, яка б допомагала розвиватися у спорті або могла б мотивувала.

Текстові рекомендації від опитаних людей про те, щоб можна було змінити чи додати у застосунок (Таблиця 1.1), в основному пов’язані з проблемою відсутності інформації про змагання, актуальні дані під час змагань та після; простотою користунням; і також частина користувачів турбується за чесність проведення жеребкування, аби на процес ніхто не впливав і він залежив лише від критеріїв жеребкування та випадковості розподілу комп’ютера.

2 Проектування системи

2.1 Типи користувачів

Отож, після аналізу системи, чітко зрозуміло, що для проектування системи потрібно обов’язково зробити розподіл можливостей за ролями користувачів аби ефективно підтримувати роботу усіх шарів.

На змаганнях обов’язково є організатори, які створюють турнір та додають у нього вагові та вікові категорії, отримують та обробляють заявки, потім мають зробити розподіл спортсменів по категоріям, провести зважування і вже під час самих змагань забезпечувати проведення поєдинків та контролювати збереження коректних результатів. За це будуть відповідати «адміністратори», їх можливості і функціонал буде найбільший.

Також обов’язково на змаганнях присутній тренер, який у системі буде посередником між спортсменами та організаторами змагань. Він до змагань займається створенням і відправленням, а також додаванням до заявки спортсменів, перевіривши коректність інформації, та під час зважування контролює всі дії, які відбувається з кожним із його спортсменів у базі.

І найголовніше, мають бути спортсмени, які саме беруть участь у змаганнях. Вони безпосередньої участі у роботі застосунку не мають, проте корисно було б надавати їм доступ до перегляду інформації аби вони мали час для підготовки та коригування свого часу виступу. Також у більшості бойових видів спорту мають бути судді, які обов’язково мають бути спортсменами. Вони можуть бути як діючі спортсмени, проте на конкретних змаганнях не беруть участь з певних причин, або можуть бути спортсменами в минулому, але все одно в основі це має бути спортсмен. Тому розподіл ролей застосунку має приблизно такий вигляд:

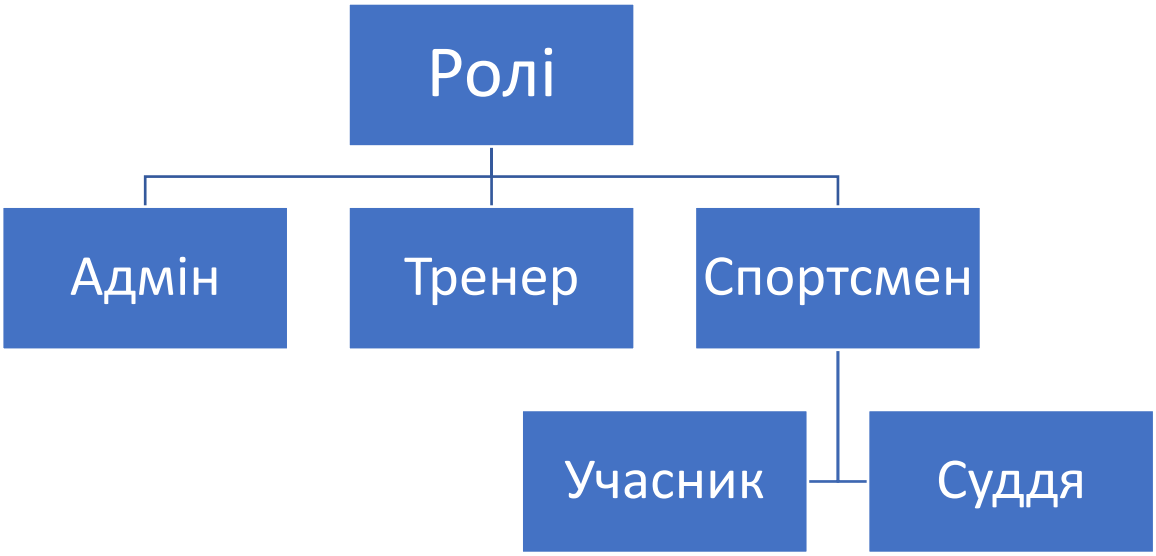

Рисунок 2.1 – Схема розподіл ролей

2.2 Функціональні та нефункціональні вимоги

Перед початком проектування системи потрібно визначити функціональні та нефункціональні вимоги.

- Обов’язково для усіх типів користувачів повинна бути авторизація та аутентифікація;
- Без авторизації та автентифікації доступ надається лише до перегляду календаря подій.

Далі вимоги будуть поділятися по кожній ролі. Для усіх ролей буде доступне в меню пункт налаштування, де можна змінити свій пароль.

Функціональні вимоги адміністратора:

- Федерація керує видачою членських квитків для спортсменів та тренерів, тому ці дані має додавати лише адміністратор;
- Додавати нових адміністраторів може лише вже наявні адміністратори;

- Доступ до користуванням цього сервісу надається адміністратором - він додає номер членського квитка та інформацію до бази і тим самим створює початковий профіль спортсмена чи тренера;
- Адміністратор повністю керує змаганнями - він має можливість створювати, редагувати і видаляти будь-яку інформацію пов’язану із змаганнями;
- Адміністратор повинен мати функціонал для обробки та підтвердження заявок від тренерів;
- Адміністратор керує та проводить зважування, розподіляє суддів на окремі створені у базі спортивні килими для змагань;
- Адмін проводить жеребкування і далі створює розклад. За створеним розкладом розподіляє поєдинки на різні килими;
- Також має можливість змінювати звання та пояси у спортсменів - після проведення атестації автоматично мають змінюватися ранг поясу тих спортсменів, які були заявлені на атестацію.

Із нефункціональних вимог:

- Адміністратор повинен мати можливість проводити жеребкування за обраними критеріями. Критерії визначаються правилами змагань, але існує всього три, які між собою можна об’єднувати:
    1) Розводити спортсменів з одного міста.
    2) Розводити спортсменів з одного клубу.
    3) Розводити спортсменів одного тренера.
- Адміністратор забезпечує кожен килим технічним засобом та надає доступ керівнику кожного килими для проведення поєдинків;
- При зміні у розкладі обов’язково перевіряти зміни в реальному часі для коректного відображення у інших типів користувачів;
- Функціонал атестація не включається у застосунок, тому замість цього використовувати автоматичне оновлення рангу для обраних спортсменів вручну;

Далі щодо функціональних вимог тренерів:

- Головна задача тренера створення, редагування, видалення заявок та додавання до заявки своїх спортсменів та суддів;
- При створенні профілю спортсмена за членським квитком одразу автоматично встановлюється його тренер, проте у подальшій роботі застосунку лише поточний тренер може проводити зміни на іншого тренера у спортсмена, щоб спортсмен не міг залишитися без тренера;
- Має можливість робити будь-які зміни у профілі спортсмена окрім зміни поясу чи звання;
- Тренер може переглядати свої заявки, список своїх учнів та дивитися на їх результати.

Із нефункціональних вимог:

- Тренеру необхідно одразу бачити зміни у профілі його учнів;
- Обмежити створення заявок на 1 турнір. Кожен турнір лише 1 заявка від тренера, для будь-яких змін використовувати редагування поточної заявки.

Вимоги до останнього типу користувачів — спортсменів:

- Мають можливість перегляду повної інформації про себе, про свої досягнення, результати, навіть статистику, свою поточну участь у змаганні та пулю, причому до жеребкування та після;
- Мають змогу змінювати у профілі фото, своє прізвище та ім’я латиницею, номер телефону та пошту, все інше редагування - відсутнє;
- Доступний пункт меню «суддівство», де можна переглянути історію змагань, на яких користувач працював суддею та побачити наявну інформацію на поточних змаганнях.

Із нефункціональних вимог до спортсменів лише:

- Без налаштованої електронної пошти неможливо буде змінити пароль;
- Відслідковувати при змінення рангу поясу спортсмена, щоб колір меню коректно відображався на новий поточний колір.

2.3 ER-модель

Для покриття усіх описаних можливостей треба правильно спроектувати реляційну модель. Фінальний вигляд ER-моделі:

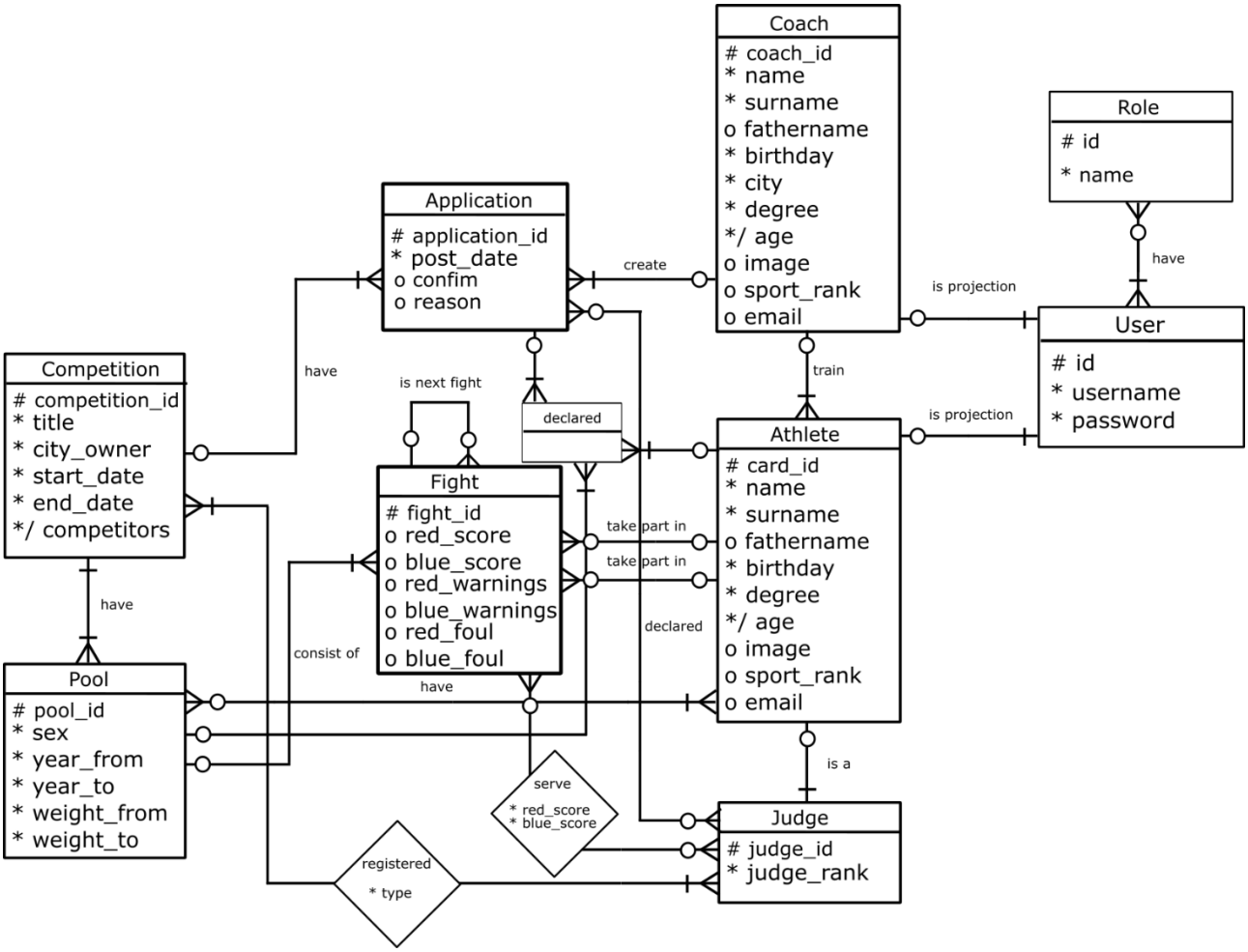

Рисунок 2.2 – ER-модель

Розроблена ER-модель та створенні зв’язки між сутностями у моделі мають забезпечувати попередньо описаний функціонал. Більшість зв’язків були створені на основі вимог до користувачів тому мають бути зрозумілі, проте деякі з них потребують уваги.

Сутності Athlete та Fight мають 2 однакових зв’язки. Дане рішення було викликане тим, що історично у двобої бере участь обов’язково 2 спортсмени. Тому було вирішено змінити зв’язок багато до багатьох на 2 однакових зв’язки один до багатьох.

Також є зв’язок типу 3 між сутностями Athlete, Pool та Application, який забезпечує реєстрацію на певні змагання учасника у певну категорію. Проте у той же час так само присутній зв’язок від Athlete до Pool, але цей зв’язок не є надлишковим. Причиною цього є випадки, коли на етапі подання заявки спортсмен має певну вагу та заявлений у відповідну вагову категорію, але під час зважування вага може критично відрізнятися від заявленої, порушуючи межі вагової категорії, тому керівники будуть зобов’язані записати спортсмена у нову категорію, яка буде відрізнятися від початкової. Таким чином, зв'язок категорії із спортсменом на одних і тих самих змаганнях може різнитися, бо мають різний сенс: один забезпечує інформацією про категорія у заявці, а інший - про реальну.

Схожа ситуація із сутністю Judge, яка має зв’язок із змаганнями через заявку та безпосередньо напряму. Він також не є надлишковим, тому що завчасно не відомо, яка кількість суддів остаточно прибуде на змагання та який рівень конкретно цих змагань. Тому позиція для кожного судді обирається перед початком змагань і не відома на етапі подання заявок.

Сутності User та Role створенні для забезпечення авторизації та аутентифікації реальних користувачів застосунку. User має два однакових зв’язки до сутностей Coach та Athlete та ще й зв'язок один до одного. Можна уникнути такої ситуації, зробивши атрибути реляції User атрибутами Coach та Athlete, проте мета такого рішення пов’язана із безпекою та виокремленням цих даних від інших описових атрибутів, і також для уникнення дублювання атрибутів реляції.

2.4 RDB-модель

Такий вигляд має модель спроектована у базу даних, де з’являються типи даних у атрибутів та замість звичайних зв’язків між сутностями проектоуються зовнішні ключі для зв’язків.

Вигляд RDB-моделі:

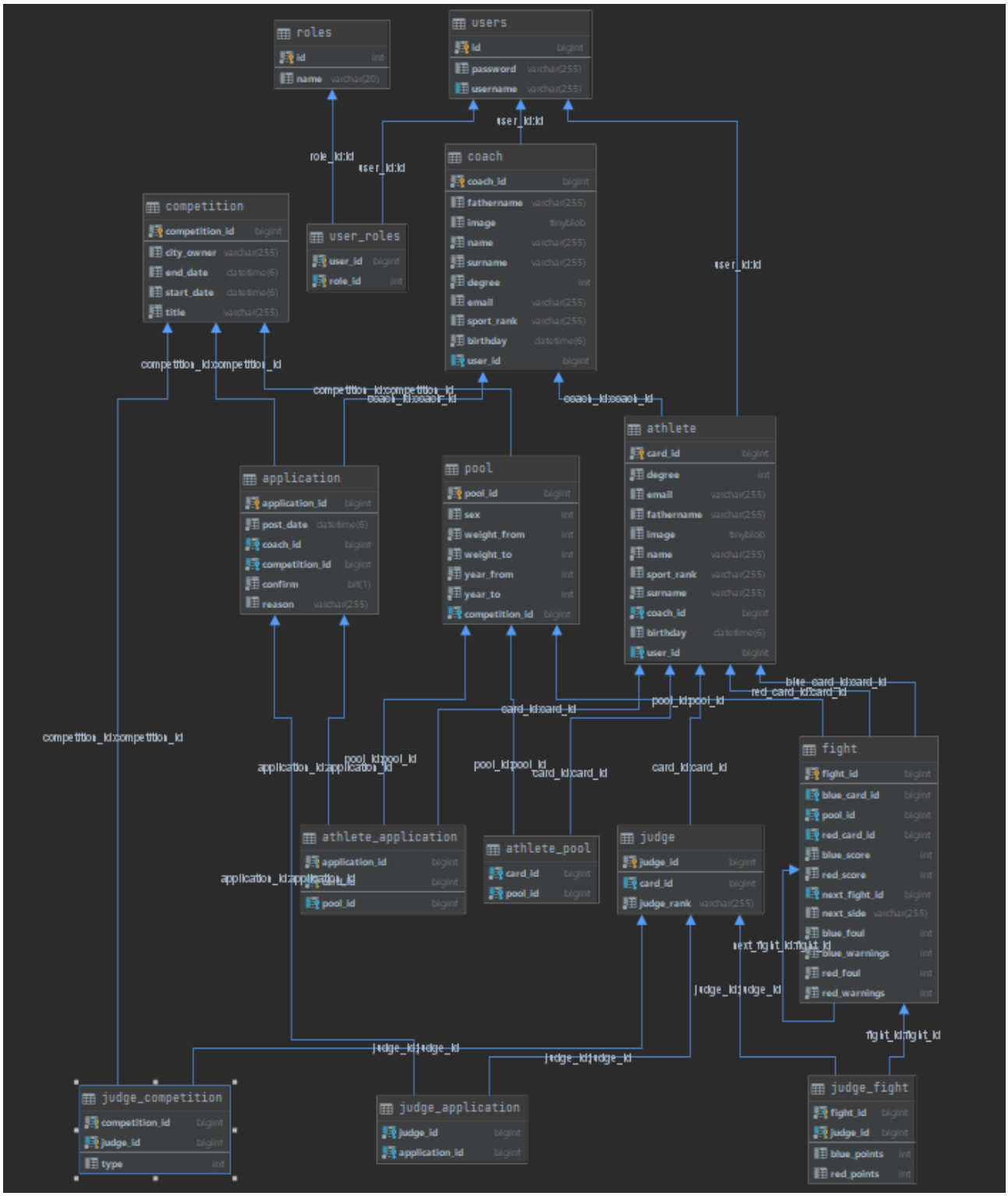

Рисунок 2.3 – RDB-модель

2.5 Прототипи сторінок

Сторінки застосунку мають бути із сучасним та зручним інтерфейсом. Навігаційне меню має однаковий вигляд для всіх користувачів, але відрізняється кольором, який залежить від рангу спортсмена чи поясу. Елементи меню та сторінки для аналогічних дій різних користувачів за ролями - так само однакові.

Прототипи основних сторінок та елементів:

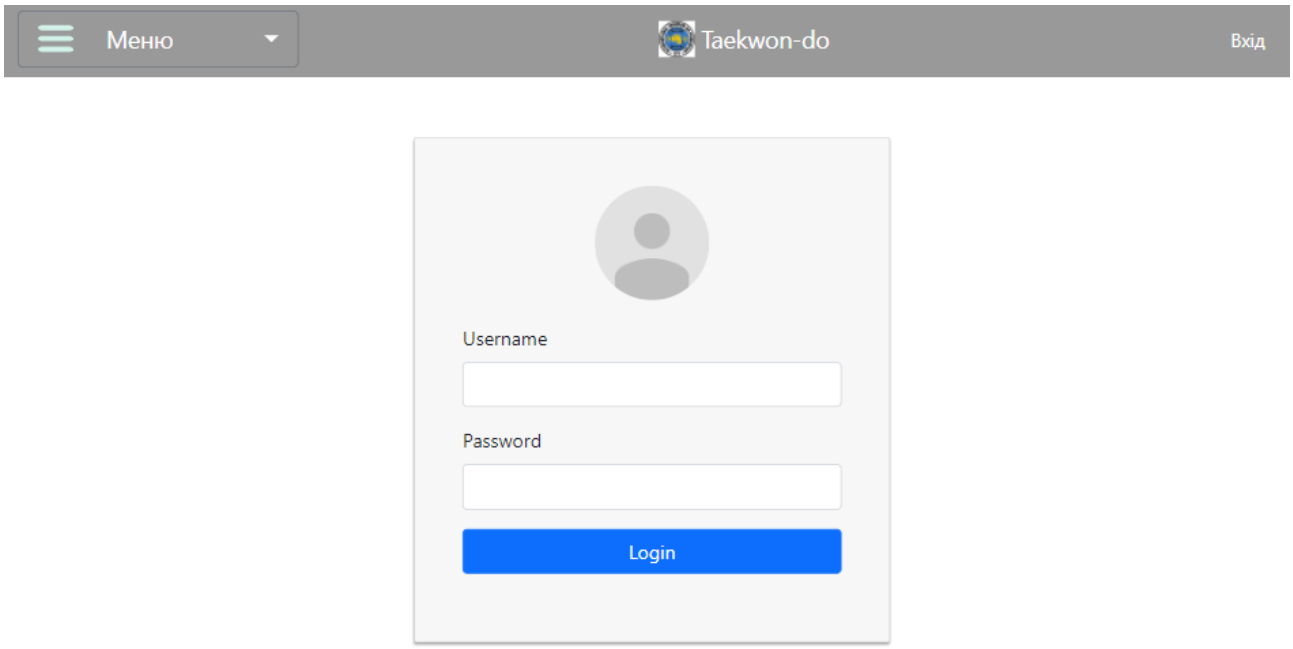

Рисунок 2.4 – Сторінка входу

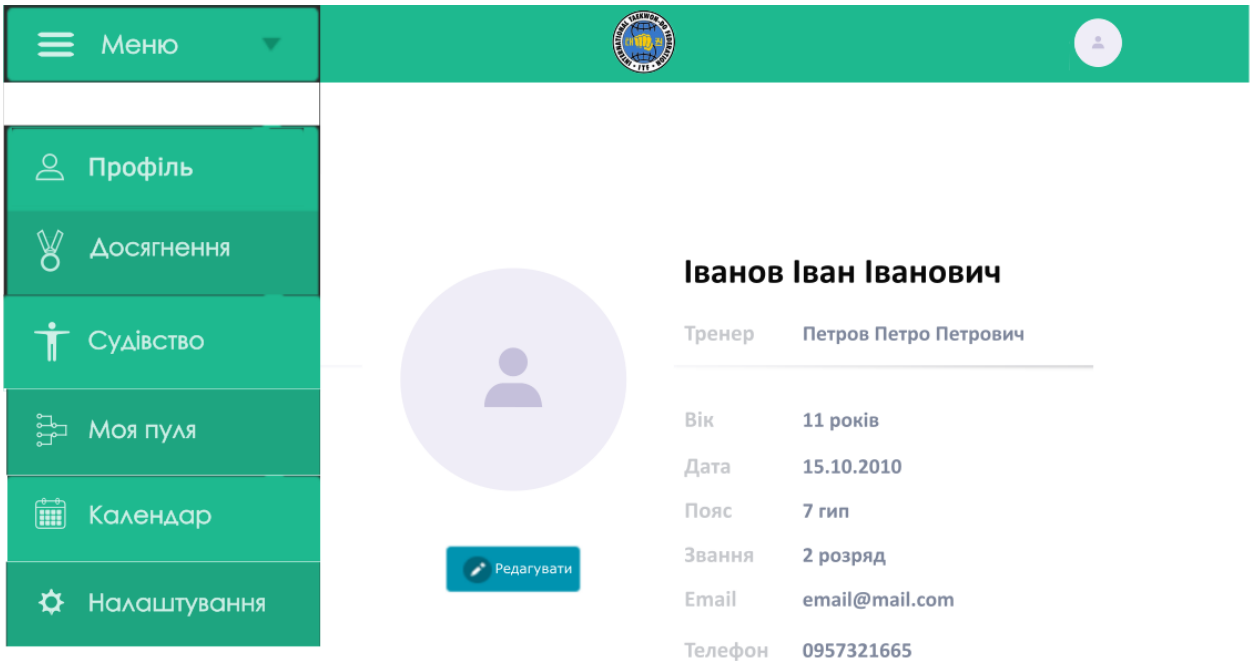

Рисунок 2.5 – Сторінка спортсмена та навігаційне меню

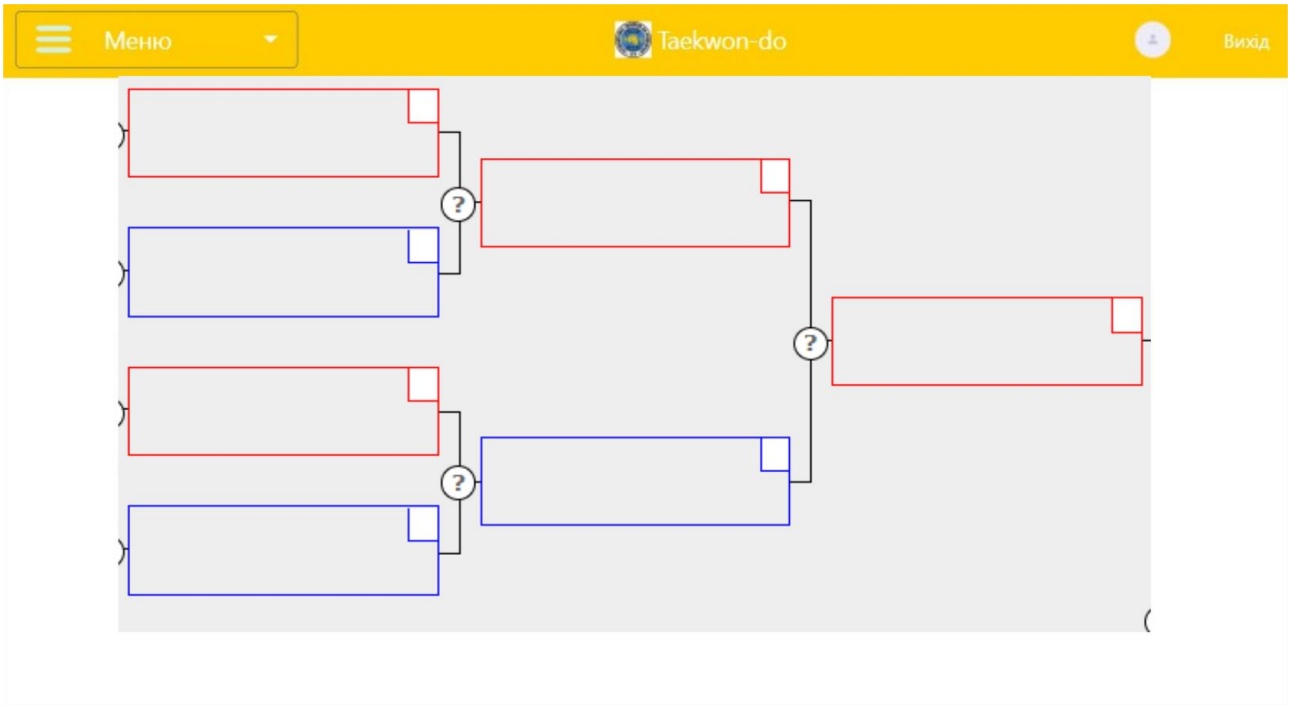

Рисунок 2.6 – Сторінка пулі після жеребкування

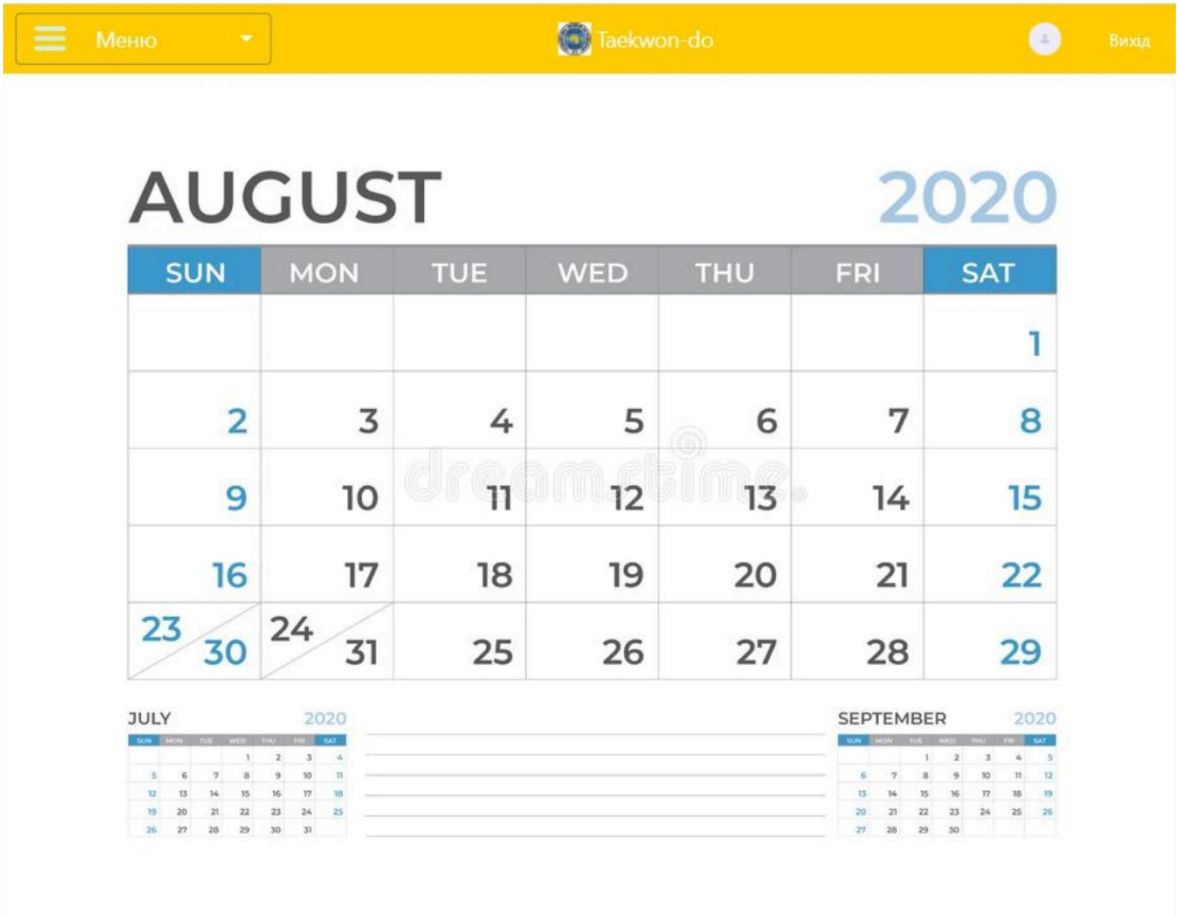

Рисунок 2.7 – Календар змагань

Рисунок 2.8 – Сторінка заявки та вікно додавання спортсменів до заявки

3 Реалізація системи

3.1 Опис обраних технологій

Для створення проекту потрібно визначити стек технологій. Веб застосунок треба обов’язково розділяти на клієнтську та серверну частини.

Перед створення кожного проекту проходить безліч процесів планування та прийняття рішень. Фаза планування має вирішальне значення і може вплинути на майбутнє проекту.

Вибір правильного стеку технологій, який найбільше підходить для реалізації поставленої задачі, може вплинути на час розробки, вартість, якість застосунку, а також на масштабованість, тому важливо ухвалити правильне рішення, навіть якщо потрібно витратити більше часу на аналіз.

3.1.1 Клієнтська частина

Для реалізації клієнтської частини застосунку використовується фреймворк Angular та мова програмування TypeScript.

Angular має вбудовану надійну систему розробки, таку як маршрутизація, робота з токенами та керування станом, що при будь-яких оновленнях джерела даних забезпечує динамічні зміни на сторінці користувача. Також він має достатньо структуровану архітектуру. Застосунок складається з окремих компонентів, які в свою чергу розбиті на три різні файли - логіка, шаблон та стиль.

Логіка компонентів описується у відповідному файлі з використанням мови TypeScript. Загалом мова дуже схожа на мову JavaScript, яка наразі так само є широко розповсюджена для розробки фронт-енду. Головною перевагою мови TypeScript є її нетипізованість, що забезпечує простоту та уникання проблеми обробки даних отриманих з сервера.

Для створення шаблонів та стилі використовується фреймворк Bootstrap, який має набір класів CSS і функції JavaScript та використовуються для полегшення розробки адаптивних сайтів і програм, орієнтованих на мобільні пристрої. Адаптивний дизайн дає змогу веб-сторінці або додатку визначати розмір і орієнтацію екрана користувача та автоматично підлаштовувати відображення відповідно до цього. Набір класів CSS набагато полегшує роботу зі стилями та забезпечує єдиний дизайн для всіх сторінок застосунку.

3.1.2 Серверна частина

Для реалізації серверної частини застосунку використовується фреймворк Spring та мова програмування Java.

Spring підтримує мову програмування Java та дозволяє найпростішим способом створити веб-додаток. Spring має великий функціонал, але його найбільш значущими особливостями є: управління залежностями, автоматична конфігурація та вбудовані контейнери сервлетів.

Щоб прискорити процес управління залежностями, Spring пакує необхідні сторонні залежності для кожного типу програми та надає їх розробнику за допомогою так званих starter-пакетів. Тому всі наявні залежності та їх визначення знаходиться в одному місці, що надає простоту у використанні.

Чудовою можливістю фреймворку є автоматична конфігурація програми. Після вибору відповідного starter-пакета, Spring автоматично налаштовує додаток на основі доданих залежностей. Проте автоматична конфігурація може бути повністю перевизначена в будь-який момент за допомогою налаштувань користувача.

Будь-який веб-додаток на Spring включає вбудований web-сервер, за замовчуванням це є Apache Tomcat. Саме тому при створенні веб-додатку на основі Spring не треба турбуватися про налаштування контейнера сервлетів та

розгортання програми на ньому, що безперечно є більш зручним та простим варіантом розробки.

Spring Security – середовище для аутентифікації та авторизації користувачів. Фреймворк застосовується для захисту програм на Spring. У ньому представлені базові інструменти безпеки, які легко розширюються на вирішення різних завдань. Основним принципом його роботи є ланцюг фільтрів, які послідовно оброблюють дані на етапі аутентифікації, та у разі невдачі хоча б одного з фільтрів доступ користувачу не надається. При спробі не запустити перший стандартний фільтр, не запрацює увесь застосунок, а не лише безпекова частина.

Для зберігання даних використовується реляційна база даних MySQL. Основною причиною використання саме цієї системи управління бази даних є різноманітність наявних типів даних, що і сприяє її частому використанню при розробці різних додатків, особливо для створення веб-сервісів та серверів.

3.2 Реалізація основної частини функціоналу

3.2.1 Реалізація безпекової частини

3.2.1.1 Серверна частина

За безпекову частину застосунку відповідає Spring Security. У SecurityContextHolder міститься інформація про поточний контекст безпеки програми, який включає докладну інформацію про поточного користувача Principal. SecurityContext містить об'єкт Authentication і в разі необхідності отримує інформацію системи безпеки, пов'язану із запитом від користувача.

UserDetails надає необхідну інформацію для побудови об'єкта Authentication, а UserDetailsServiceImpl використовується для створення об'єкту UserDetailsImpl шляхом реалізації єдиного методу інтерфейсу UserDetailsService - loadUserByUsername(String username).

```
@Override
@Transactional
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

  User user = userRepository
        .findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User Not Found with
username: " + username));

  return UserDetailsImpl.build(user);
}
```

Рисунок 3.1 – Код методу loadUserByUsername(String username)

При реєстрації користувача пароль шифрується за допомогою PasswordEncoder та зберігається у такому вигляді у базі даних.

```
if (userRepository.existsByUsername(signUpRequest.getUsername())) {
    return ResponseEntity
            .badRequest()
            .body(new MessageResponse("Error: Username is already taken!"));
}

// Create new user's account
User user = new User(signUpRequest.getUsername(),
        signUpRequest.getEmail(),
        encoder.encode(signUpRequest.getPassword()));
```

Рисунок 3.2 – Код шифрування та створення нового користувача

При аутентифікації користувачеві буде запропоновано увійти в систему, надавши логін та пароль. Ім'я користувача та пароль об'єднуються в екземпляр класу UsernamePasswordAuthenticationToken після чого він передається екземпляру AuthenticationManager для перевірки. Отриманий екземпляр інтерфейсу Authentication передається далі і для користувача встановлюється контекст безпеки шляхом виклику методу SecurityContextHolder.getContext().setAuthentication(). Після цього генерується jwt токен та повертається у тілі відповіді на даний запит користувача.

```
@PostMapping("/signin")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest
loginRequest) {

  Authentication authentication = authenticationManager.authenticate(
      new UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));

  SecurityContextHolder.getContext().setAuthentication(authentication);
  String jwt = jwtUtils.generateJwtToken(authentication);

  UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();
  List<String> roles = userDetails.getAuthorities().stream()
      .map(item -> item.getAuthority())
      .collect(Collectors.toList());

  return ResponseEntity.ok(new JwtResponse(jwt,
                       userDetails.getId(),
                       userDetails.getUsername(),
                       userDetails.getEmail(),
                       roles));
}
```

Рисунок 3.3 – Код методу авторизації користувача

У разі якщо пароль не відповідає імені користувача буде викинуто помилку BadCredentialsException з повідомленням “Bad Credentials”.

У методі configure(HttpSecurity http) класу WebSecurityConfig, де відбувається базова конфігурація безпеки, вказуються URL, які не підлягають безпековій перевірці, тому що є публічними для усіх користувачів та не потребують реєстрації.

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.cors().and().csrf().disable()
    .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and(
)
    .authorizeRequests().antMatchers("/api/auth/**").permitAll()
    .antMatchers("/public/**").permitAll()
    .anyRequest().authenticated();

  http.addFilterBefore(authenticationJwtTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
}
```

Рисунок 3.4 – Код методу configure(HttpSecurity http)

3.2.1.2 Клієнтська частина

Після аутентифікації користувача сервер повертає клієнту згенерований токен. За допомогою AuthService та TokenStorageService отриманий токен зберігається у сховищі поточної сесії.

```
onSubmit(): void {
    const { username, password } = this.form;
    this.authService.login(username, password).subscribe(
      data => {
        this.tokenStorage.saveToken(data.accessToken);
        this.tokenStorage.saveUser(data);
        this.isLoginFailed = false;
        this.isLoggedIn = true;
        this.roles = this.tokenStorage.getUser().roles;
        this.reloadPage();
      },
      err => {
        this.errorMessage = err.error.message;
        this.isLoginFailed = true;
      }
    );
  }
```

Рисунок 3.5 – Код обробки токену

І надалі при кожному запиту користувача цей токен буде автоматично передаватися у хедерах запиту, а саме у Authentication хедері, на сервер, де щоразу перевіряється токен на валідність (детальніше див. пункт 3.2.1.1). Також є методи для того щоб дістати токен із сховища і паралельно з токеном зберігається поточний користувач.

```
public saveToken(token: string): void {
    window.sessionStorage.removeItem(TOKEN_KEY);
    window.sessionStorage.setItem(TOKEN_KEY, token);
}
public getToken(): string | null {
    return window.sessionStorage.getItem(TOKEN_KEY);
}
public saveUser(user: any): void {
    window.sessionStorage.removeItem(USER_KEY);
    window.sessionStorage.setItem(USER_KEY, JSON.stringify(user));
}
public getUser(): any {
    const user = window.sessionStorage.getItem(USER_KEY);
    if (user) {
      return JSON.parse(user);
    }
    return {};
}
```

Рисунок 3.6 – Методи класу TokenStorageService

3.2.2 Алгоритм жеребкування

Загалом процес жеребкування не проста задача у якої на меті правильно розподілити всіх учасників на пари. Очевидно жеребкування можна робити випадковим чином – перемішати список та поступово обирати пари. Але на більшості змагань існують критерії. Так, спортсмени з одного міста не можуть бути розподілення одразу у першу пару, а на деяких змаганнях взагалі не мають зустрітися аж до фіналу, аналогічно для спортсменів одного тренера. Отже варіант випадково розподіляти спортсменів не влаштовує.

Наступна спроба для реалізації жеребкування спробувати динамічно розподіляти для кожного спортсмена підходящу пару. На 1-е місце ставити будь-якого спортсмена на 2-е місце треба ставити спортсмена який не заперечує критеріям 1-го , на 3-є місце спортсмена який не заперечує критеріям 1-го та 2-го і так далі розподіляти аж до кінця списку. Проте зі збільшенням кількості учасників, алгоритм стає дуже важким у масштабуванні, тому що потрібно враховувати велику кількість умов до наступного кожного спортсмена. Тому такий варіант теж не підходить.

Наступна ідея як спроектувати алгоритм жеребкування полягає у тому, що прожеребкована група учасників це просто правильно впорядкований список за певним алгоритмом. Перевагою такого підходу є те, що він не залежить від кількості учасників. Отже задача жеребкування зводиться до створення певного алгоритму сортування.

Представлений в роботі алгоритм був створений на основі 2 ідей.

Перша ідея полягає у використанні підходу динамічного програмування - де основну задачу розбивають на менші під задачі аби звести вирішення основної до мінімуму.

Друга ідея це мати певний ланцюг фільтрів який буде розбивати початковий список на під списки для використання їх у під задачах. Найменші одиниці розподілу для категорії буде для 2 та 3 учасники. Будь-яка інша кількість учасників може бути розбита на найменші одиниці. Наприклад 4 - це два списки по 2 учасники , 5 - два списки по 2 та 3 учасники відповідно, 7 - це 2 списки по 3 та 4 учасники, який в свою чергу розбивається на 2 по 2.

```
public List<Competitor> createPool(List<Competitor> category,
                                   boolean byCoach,boolean byClub,boolean byCity){
List<Competitor> firstPart = new ArrayList<>();
List<Competitor> secondPart = new ArrayList<>();
int size;
// sort by coach criteria
Set<Integer> idCoaches =
freeCompetitor.stream().map(Competitor::getCoach).collect(Collectors.toSet());
if(byCoach && freeCompetitor.size() != idCoaches.size()){…}
// sort by club criteria
Set<Integer> idClubs =
freeCompetitor.stream().map(Competitor::getClub).collect(Collectors.toSet());
if(byClub && freeCompetitor.size() != idClubs.size()){…}
// sort by city criteria
Set<String> idCities =
freeCompetitor.stream().map(Competitor::getCity).collect(Collectors.toSet());
if(byCity && freeCompetitor.size() != idCities.size()){…}

size = category.size()/2;
while (firstPart.size() != size){
    firstPart.add(freeCompetitor.remove(0));
}
secondPart.addAll(freeCompetitor);

temp = createPool(firstPart, byCoach, byClub, byCity);
temp.addAll(createPool(secondPart, byCoach, byClub, byCity));
return temp;
```

Рисунок 3.7 – Загальний вигляд алгоритму сортування

Загальна реалізація алгоритму полягає в тому, що вхідний список потрібно розбити на 2 під списки за критеріями. Для цього початковий список проходить ланцюг фільтрів. На кожному фільтрі список розбивається на під списки відповідно за умовами критерію даного фільтру.

```
Set<String> idCities =
freeCompetitor.stream().map(Competitor::getCity).collect(Collectors.toSet());

if(byCity && freeCompetitor.size() != idCities.size()){
    Map<String,List<Competitor>> criteriaCompetitor = new HashMap<>();

    for(String id : idCities){
        temp = freeCompetitor
                .stream()
                .filter(i -> i.getCity().equals(id))
                .collect(Collectors.toList());
        if (temp.size() != 1) {
            criteriaCompetitor.put(id, temp);
            freeCompetitor.removeAll(temp);
        }
    }
```

Рисунок 3.8 – Розбиття загального списку на під списки у фільтрі

З утворених списків ми ітераційно переносимо непарні елементи у 1-й результуючий список, парні - у 2-й список. Перенос продовжується до тих пір поки у списку не залишиться 1 елемент або жодного. Якщо залишився 1 елемент то він повертається у початковий список.

```
for(String id :criteriaCompetitor.keySet()){
    List<Competitor> tt = criteriaCompetitor.get(id);
    while (tt.size() > 1){
        firstPart.add(tt.remove(0));
        secondPart.add(tt.remove(0));
    }
    freeCompetitor.addAll(tt);
}
```

Рисунок 3.9 – Розбиття на 2 результуючих списки

Після проходження всіх фільтрів, починаючи із найбільш пріоритетного, всі спортсмени, які залишилися у початковому списку, випадковим чином розподіляється до 1-го та 2-го результуючих списків. Така ситуація, що деякі спортсмени не розподілилися під час фільтрування означає для алгоритму, що це «вільний» спортсмен - його можна розподілити на будь-яку позицію і це не буде суперечити критеріям.

Утворені 2 списки рекурсивно повторюють даний процес поки не дійдуть до мінімальних одиниць. У випадку, коли залишається 2 спортсмени ділити далі неможливо тому це і є готова пара. У випадку 3 учасників в результаті фільтрації список розділиться на 1-го та 2-ох учасників, де 2 учасники - це вже так само готова пара, а 1 учасник - це наступний суперник переможця попередньої пари.

```
size = category.size()/2;
while (firstPart.size() != size){
    firstPart.add(freeCompetitor.remove(0));
}
secondPart.addAll(freeCompetitor);

temp = createPool(firstPart, byCoach, byClub, byCity);
temp.addAll(createPool(secondPart, byCoach, byClub, byCity));
return temp;
```

Рисунок 3.10 – Остаточне заповнення списків та рекурсивний спуск

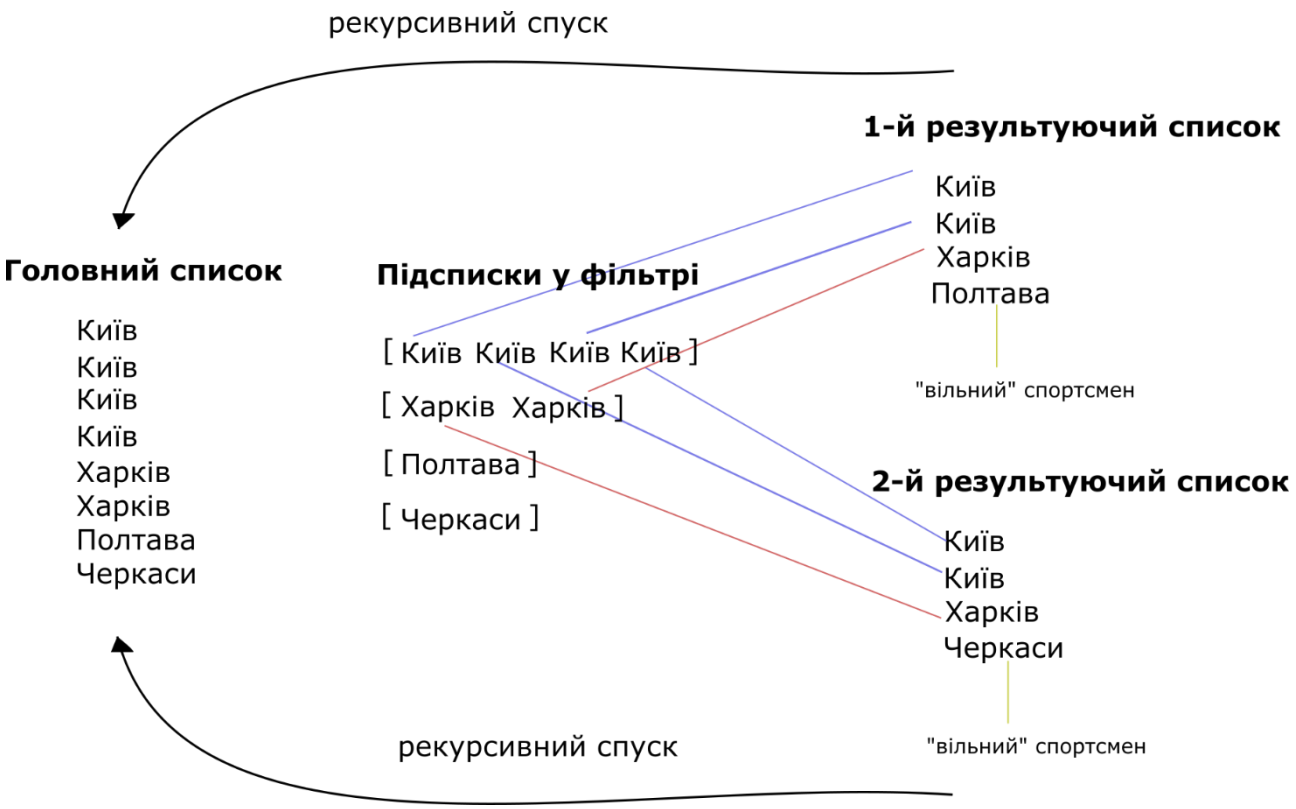

Рисунок 3.11 – Ілюстрація роботи алгоритму

3.2.3 Алгоритм пошуку переможця

Алгоритм пошуку переможця набагато простіший від попереднього алгоритму, проте також є важливим. Реалізація його роботи проста, але особливість у тому, що логіка роботи алгоритму реалізована за допомогою запитів до бази даних мовою JPQL.

Із вихідних даних ми отримаємо інформацію про категорію та атлета, для кого потрібно визначити місце, яке він посів на змагання. На 1-му етапі виконується запит до бази даних, який знаходить фінальний поєдинок цієї категорії.

```
@Query("select f from Fight f " +
        "inner join f.pool p " +
        "where f.nextFight is null " +
        "and p.pool_id = :id")
Fight findFinalByPool(Long id);
@Query("select f from Fight f " +
        "inner join f.pool p " +
        "where f.nextFight.fight_id = :final_id " +
        "and p.pool_id = :pool_id")
List<Fight> findSemiFinalsByPool(Long pool_id, Long final_id);
```

Рисунок 3.12 – Запити мовою JPQL

Якщо даний спортсмен був учасником фінального поєдинку, то далі, порівнюючи рахунок поєдинку, визначається чи є він переможця, чи ні. В іншому випадку виконується наступний запит до бази даних, який, маючи значення id фінального поєдинку із попереднього запиту, шукає 2 напівфінальні поєдинки. Далі аналогічно перевіряється чи був спортсмен учасником одного з двох поєдинків, та у разі участі це означає, що спортсмен посів 3-є місце, тому що однозначно відомо, що у разі перемоги у напівфінальному поєдинку спортсмен приймав би участь у фінальному поєдинку, проте на попередньому етапі вже було перевірено цю ситуацію.

3.3 Опис Rest API

Реалізований Rest API покриває функціональні можливості застосунку, а також має поділ для кожного типу користувачів і також сутностей, з якими безпосередньо працюють користувачі.

Сегмент / api/auth :

1) POST / signup – забезпечує реєстрацію користувача за даними із тіла запиту. Повертає помилку у разі спроби створити нового користувача з існуючим логіном;
2) POST / signin – забезпечує аутентифікацію користувача за даними із тіла запиту. Повертає помилку у разі помилкового паролю або логіну;
3) POST / reset/{id} – забезпечує встановлення нового паролю за даними із тіла запиту для користувача за параметром id. Повертає помилку у разі помилкового старого паролю, або невалідного нового.

Сегмент / athlete:

1) GET / profile/{id} – повертає повну інформацію про спортсмена за параметром id. Повертає помилку у разі неіснуючого id.
2) GET / results/{id} – повертає інформацію про усі результати спортсмена за параметром id. Повертає помилку у разі неіснуючого id.
3) GET / judge/{id} – повертає суддівську інформацію спортсмена за параметром id. Повертає помилку у разі неіснуючого id.
4) GET / pool/{id} – повертає інформацію про поточну пулю спортсмена за параметром id. Повертає помилку у разі неіснуючого id.

Сегмент / coach:

1) GET / all – повертає повні імена усіх тренерів.
2) GET / profile/{id} – повертає повну інформацію про тренера за параметром id. Повертає помилку у разі неіснуючого id.

3) GET / results/{id} – повертає інформацію про усі результати кожного спортсмена тренера за параметром id. Повертає помилку у разі неіснуючого id.
4) GET / sportsmen/{id} – повертає повну інформацію про кожного спортсмена тренера за параметром id. Повертає помилку у разі неіснуючого id.
5) POST / sportsmen/update – забезпечує оновлення даних спортсмена за даними із тіла запиту. Повертає помилку у разі не успішного оновлення.
6) POST / add/competition/{id}– забезпечує створення нової заявки на змагання за даними із тіла запиту для тренера за параметром id. Повертає помилку у разі не успішного створення або помилкового id.
7) GET / applications/{id}– повертає список усіх заявок тренера за параметром id. Повертає помилку у разі неіснуючого id.
8) GET / possible/competitions/{id}– повертає список змагань на які ще не було створено заявок тренера за параметром id. Повертає помилку у разі неіснуючого id.

Сегмент / application:

1) GET / {id}– повертає повні інформацію про заявку за параметром id.
2) GET / freeSportsmen/{id} – повертає список ще не заявлений спортсменів та список ще не заявлених суддів для заявки за параметром id. Повертає помилку у разі неіснуючого id.
3) GET / pools/{id} – повертає список пулів доступних для заявки за параметром id. Повертає помилку у разі неіснуючого id.
4) POST / add/athlete/{id}– забезпечує додавання списку спортсменів за даними із тіла запиту до заявки за параметром id. Повертає помилку у разі не успішного створення, невалідного списку або помилкового id.
5) POST / add/ judge/{id}– забезпечує додавання списку суддів за даними із тіла запиту до заявки за параметром id. Повертає помилку

у разі не успішного створення, невалідного списку або помилкового id.

6) POST / add/ competition/{id}– забезпечує додавання нової заявки тренера за даними із тіла запиту до змагань за параметром id. Повертає помилку у разі не успішного створення, невалідного списку або помилкового id.
7) DELETE / delete/athlete/{id}– забезпечує видалення спортсмена за даними із тіла запиту із заявки за параметром id. Повертає помилку у разі не успішного видалення, невалідного спорстмена або помилкового id.
8) DELETE / delete / judge/{id}– забезпечує видалення судді за даними із тіла запиту із заявки за параметром id. Повертає помилку у разі не успішного видалення, невалідного судді або помилкового id.

Сегмент / competition:

1) GET / all – повертає повну інформацію про всі змагання.
2) GET / pools/{id} – повертає повну інформацію про всі пулі змагання за параметром id. Повертає помилку у разі неіснуючого id.
3) GET / pool/{id} – повертає повну інформацію пулю за параметром id. Повертає помилку у разі неіснуючого id.
4) GET / athletes/{id} – повертає список не зважених спортсменів для змагання за параметром id. Повертає помилку у разі неіснуючого id.
5) POST / add/ pool/{id}– забезпечує додавання нової пулі за даними із тіла запиту до змагань за параметром id. Повертає помилку у разі не успішного створення, невалідної пулі або помилкового id.
6) DELETE / delete/ pool /{id}– забезпечує видалення пулі за параметром id. Повертає помилку у разі не успішного видалення або помилкового id.
7) POST / generate/ pool/{id}– забезпечує генерації нових пуль за даними із тіла запиту до змагань за параметром id. Повертає помилку у разі не успішного створення, невалідних даних або помилкового id.

8) POST / weigh/athlete - забезпечує додавання спортсмена за даними із тіла запиту до пулі за даними із тіла запиту під час зважування. Повертає помилку у разі не успішного створення, невалідних даних.

Сегмент / public:

1) GET / competitions – повертає повну інформацію про всі змагання для календаря.
2) POST / competition/new – забезпечує додавання нове змагання. Повертає помилку у разі не успішного створення.
3) POST / competition/update – забезпечує оновлення змагань за даними із тіла запиту. Повертає помилку у разі не успішного оновлення, невалідних даних.
4) DELETE / competition/delete/{id} – забезпечує видалення змагань за параметром id. Повертає помилку у разі не успішного видалення або помилкового id.

3.4 Інтерфейс застосунку

3.4.1 Сторінки інтерфейсу

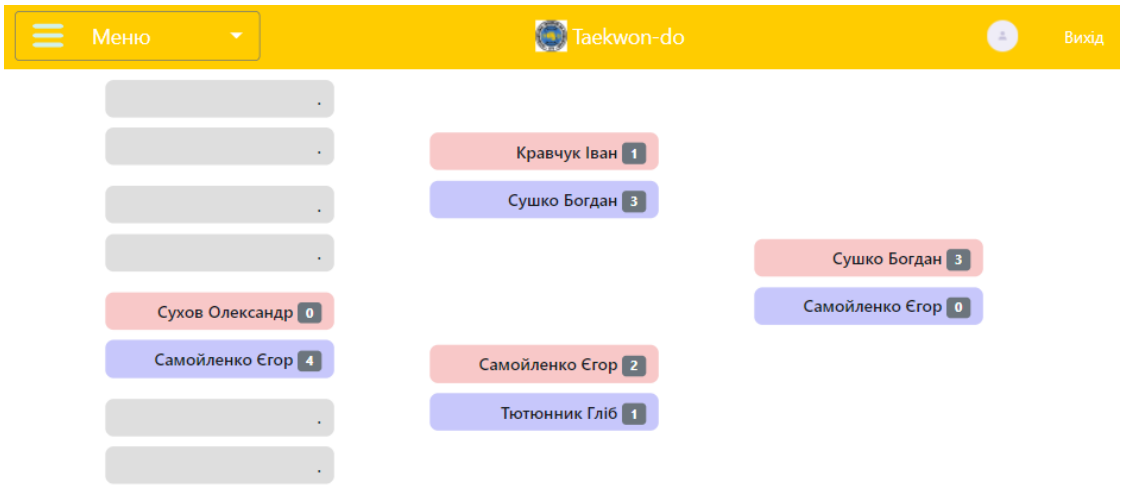

Рисунок 3.13 – Вигляд сторінки пулі після жеребкування

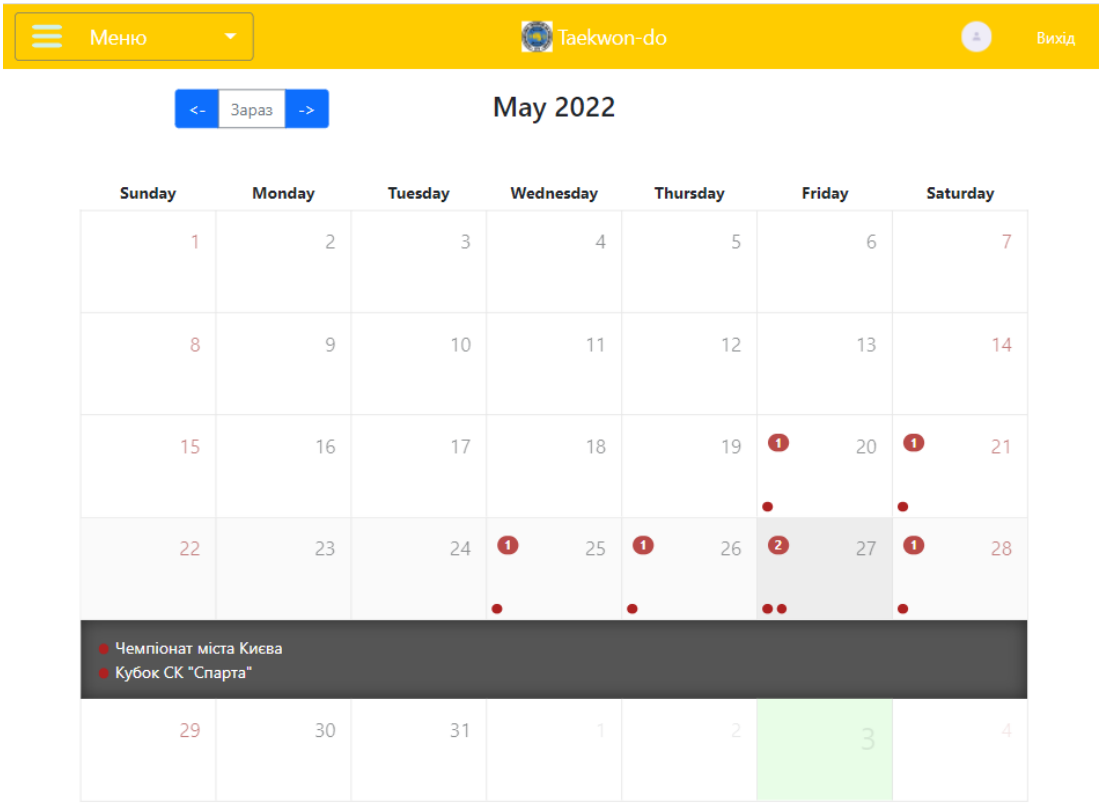

Рисунок 3.14 – Вигляд сторінки календаря

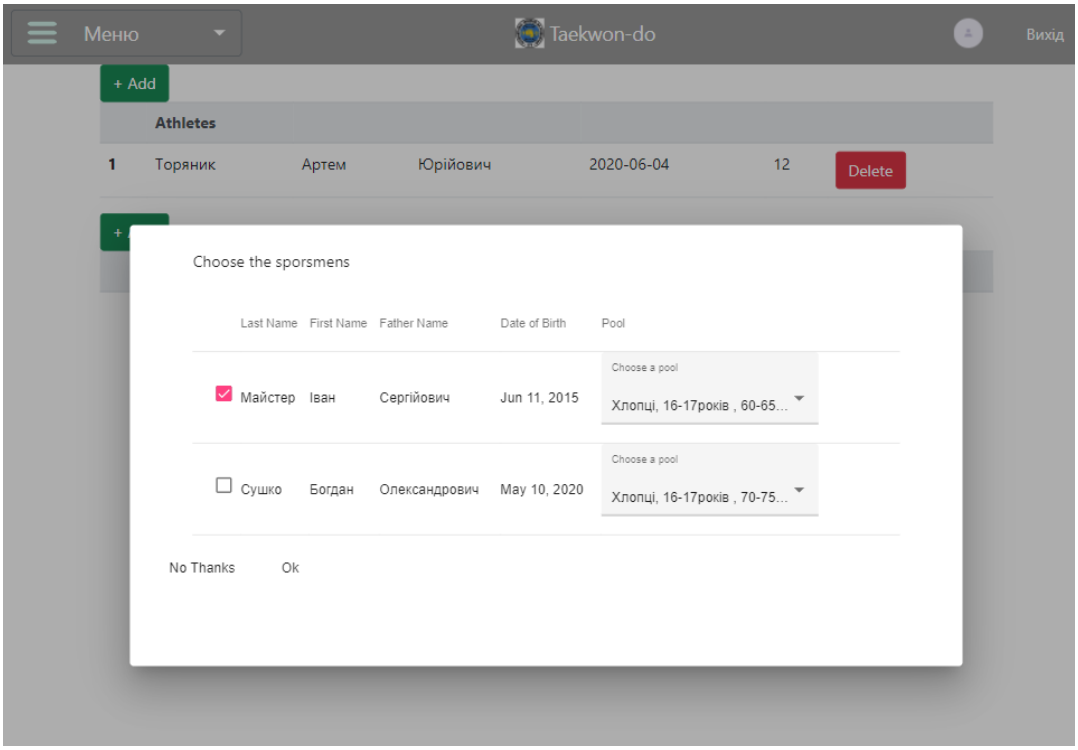

Рисунок 3.15 – Вигляд сторінки заявки та додавання спортсменів

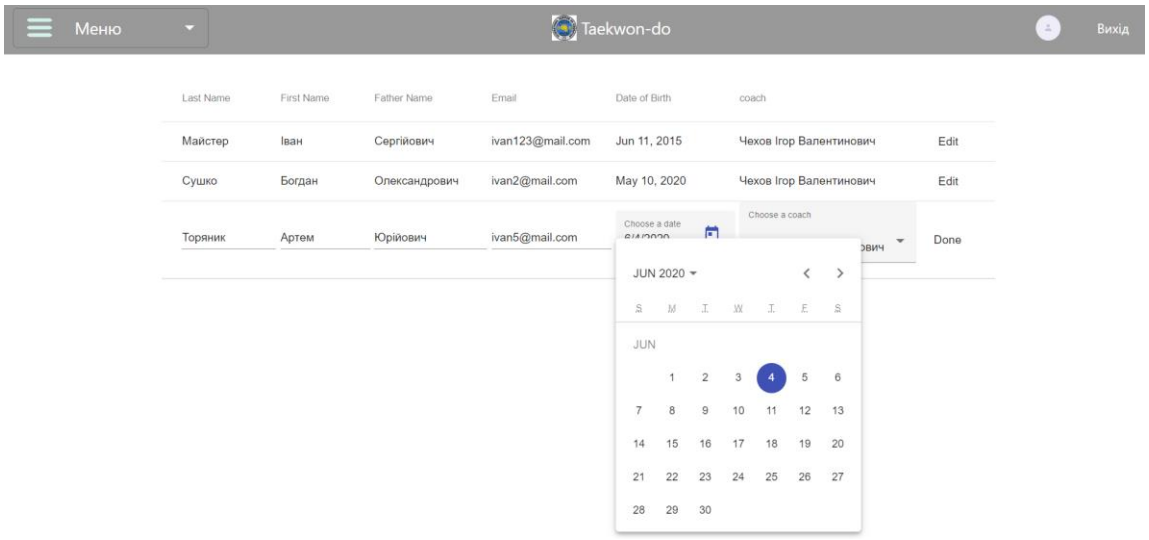

Рисунок 3.16 – Вигляд сторінки спортсменів тренера та їх редагування

3.4.2 Мапа сайту

- Головна
    - Логін
    - Календар
    - Налаштування
    - Моя пуля
    - Суддівство
    - Досягнення
        - Пуля
    - Профіль
    - Заявки
        - Створити
        - Заявка
            - Додати спортсмена
            - Додати суддю
    - Спортсмени
    - Змагання

- Пулі
    - Створити
    - Згенерувати
    - Учасники
- Зважування
    - Пулі

Висновки

Тема спорту завжди була популярною і наразі очевидний її стрімкий розвиток в технічному плані. У даній роботі було детально досліджено повний цикл проведення змагань, починаючи від початкового етапу — реєстрації, закінчуючи аналізом виступів після змагань. Також було розглянуто як у деяких відомих аналогах був реалізували кожен окремий етап та які етапи було пропущено і чому.

Найбільшими двома виявленими проблемами були чесне та незалежне жеребкування та відсутність можливості для використання даного застосунку найбільшим шаром на змаганнях — учасниками. Розроблений застосунок має на меті всунути недоліки своїх аналогів, саме тому було докладено найбільше зусиль для розробки зручного інтерфейсу з максимально можливим функціоналом для учасників змагань.

Після детального аналізу процесу жеребкуванню був створений новий власний алгоритм для забезпечення чесного, незалежного жеребкування.

Наступними кроками для ще більшого покращення системи є створення мобільних застосунків під різні операційні системи, які будуть отримувати дані з одного й того самого сервера, що й веб-застосунок, але надавати функціонал користувачам у мобільному інтерфейсі, що наразі є досить розповсюдженою практикою. Ще одним можливим покращенням даного застосунку - це додавання нового функціоналу, який буде генерувати та надавати користувачам поради на основі тих даних, які спортсмени або тренери будуть вносити після своїх тренувань. Така можливість наддасть ще більше можливостей для керування своїм прогресом та плануванням подальшої підготовки до змагань.

Список літератури

1. Компьютерная система проведения соревнований [Електронний ресурс]. — Режим доступу: https://dic.academic.ru/dic.nsf/ruwiki/1526634
2. TOURNAMENT ASSISTANT [Електронний ресурс]. — Режим доступу: https://tassistant.com/p/
3. ОРГАНІЗАЦІЯ І ПРОВЕДЕННЯ СПОРТИВНИХ ЗМАГАНЬ [Електронний ресурс]. — Режим доступу: https://www.uzhnu.edu.ua/en/infocentre/get/24454
4. Implementing Role Based in a Web App [Електронний ресурс]. — Режим доступу: https://medium.com/bluecore-engineering/implementing-role-based-security-in-a-web-app-89b66d1410e4
5. Вимоги до застосунку [Електронний ресурс]. — Режим доступу: https://uk.myservername.com/features-functional-requirements
6. Huge ER Diagram [Електронний ресурс]. — Режим доступу: https://dataedo.com/blog/do-you-really-need-a-huge-er-diagram-for-the-entire-database
7. Ternary Relationship [Електронний ресурс]. — Режим доступу: https://www.sciencedirect.com/topics/computer-science/ternary-relationship
8. Laurentiu Spilca, Spring Security in Action, 2020 рік - 3 Implementing authentication in Spring Security, 4 Understanding the PasswordEncoder, 5 Using the SecurityContext contract
9. Nate Murray, Ng-book The complete book of Angular 5, 2017 рік – Forms in Angular, DI, HTTP, Routing
10. Angular best practices, security [Електронний ресурс]. — Режим доступу: https://angular.io/guide/security