НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмна система для обслуговування турнірів з баскетболу 3х3»

Виконав:
студент IV курсу, групи КП-92
Остапенко Іван Петрович _____

Керівник:
Асистент кафедри ПЗКС,
Комісар Дмитро Олександрович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ЕІ, к.т.н., доцент
Вунтесмері Юрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2022 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Остапенку Івану Петровичу

1. Тема проєкту «Програмна система для обслуговування турнірів з баскетболу 3х3», керівник проєкту Комісар Дмитро Олександрович, асистент, затверджені наказом по університету від «31» травня 2023 р. №2107-с.

2. Термін подання студентом проєкту «16» червня 2023 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз процесу обслуговування турнірів, аналіз наявних рішень;
   - аналіз технологій розробки вебзастосунків та мобільних застосунків;
   - розроблення програмної системи з обслуговування турнірів з баскетболу 3х3;
   - опис розробленої системи;
   - аналіз розробленої системи.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма використання системи (креслення);
   - модель «сутність-зв'язок» (креслення);

- схема даних (плакат);
- алгоритм формування розкладу (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2022	
2.	Розроблення та узгодження технічного завдання	25.11.2022	
3.	Розроблення структури програмної системи	15.12.2022	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2022	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2023	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2023	
7.	Реалізація програмної системи	10.03.2023	
8.	Тестування програмної системи	17.03.2023	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2023	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	12.04.2023	
11.	Підготовка графічної частини дипломного проєкту	21.04.2023	
12.	Оформлення документації дипломного проєкту	26.05.2023	

Студент Іван ОСТАПЕНКО

Керівник проєкту Дмитро КОМІСАР

АНОТАЦІЯ

Метою дипломного проєкту є автоматизація процесу обслуговування турнірів з баскетболу 3х3.

У процесі дипломного проєктування було виконано порівняльний аналіз наявних рішень, сучасних технологій розробки вебзастосунків та мобільних застосунків.

Розроблено програмну систему обслуговування турнірів з баскетболу 3х3, що складається з трьох компонентів, а саме: вебзастосунок для організаторів турнірів, мобільний застосунок для асистентів суддів, серверний застосунок.

Вебзастосунок для організаторів турнірів забезпечує можливість реєстрації команд, створення розкладу, перегляд деталей матчів та їх протоколи.

Мобільний застосунок для суддівського складу забезпечує доступ до перегляду розкладу, електронне ведення протоколів гри.

Розроблено систему доступів для організаторів та для суддівського складу.

Описано шляхи подальшого розвитку проєкту.

ABSTRACT

The goal of this diploma project is to automate the process of servicing 3x3 basketball tournaments.

In the process of diploma development, a comparative analysis of existing solutions, modern technologies for the development of web applications and mobile applications was performed.

A software system for servicing 3x3 basketball tournaments consists of three components: a web application for tournament organizers, a mobile application for referee assistants, a server application.

The web application for tournament organizers provides the ability to register teams, create a schedule, view match details and their protocols.

The mobile application for referees provides access to viewing the schedule, taking protocols of matches.

An access system has been developed for organizers and judges.

The ways of further development of the project are described.

ДП.045490-01-90 Програмна система для обслуговування турнірів з баскетболу 3х3.
Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045490-02-91	Програмна система для	5	
	обслуговування турнірів		
	з баскетболу 3х3.		
	Технічне завдання		
ДП.045490-03-81	Програмна система для	50	
	обслуговування турнірів		
	з баскетболу 3х3.		
	Пояснювальна записка		
ДП.045490-04-51	Програмна система для	4	
	обслуговування турнірів		
	з баскетболу 3х3.		
	Програма та методика		
	тестування		
ДП.045490-05-34	Програмна система для	11	
	обслуговування турнірів		
	з баскетболу 3х3.		
	Керівництво користувача		
ДП.045490-06-99	Програмна система для	1	
	обслуговування турнірів		
	з баскетболу 3х3.		
	Функціональність		
	системи. Діаграма		
	використання		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045490-07-99	Програмна система для	1	
	обслуговування турнірів		
	з баскетболу 3x3.		
	Функціональність		
	системи. Модель		
	«сутність-зв'язок»		
ДП.045490-08-98	Програмна система для	1	
	обслуговування турнірів		
	з баскетболу 3x3.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2022 р.

ПРОГРАМНА СИСТЕМА ОБСЛУГОВУВАННЯ ТУРНІРІВ З БАСКЕТБОЛУ 3Х3

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро КОМІСАР

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ОСТАПЕНКО

2022

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: програмна система обслуговування турнірів з баскетболу 3х3.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості обслуговування турнірів з баскетболу 3х3.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмна система має складатися з компонентів:

1. Вебзастосунок для організаторів турнірів, що має такі основні функції:
   1.1. Можливість реєстрації команд, майданчиків, суддів.
   1.2. Можливість перегляду розкладу.
   1.3. Можливість перегляду деталей, протоколу матчів.
2. Мобільний застосунок для асистентів суддів, що має такі основі функції:
   2.1. Можливість перегляду розкладу турніру.

2.2. Можливість обслуговувати матч за допомогою введення ігрових моментів.

3. Серверна частина, що має такі основні функції:
   3.1. Можливість зберігати та надавати доступ до інформації щодо команд, майданчиків, суддів.
   3.2. Формулювати та надавати доступ до розкладу турніру на основі зареєстрованих команд.
   3.3. Можливість зберігати інформацію щодо ігрових моментів.

Розробку виконати з використанням:

1. Для вебклієнту – React.
2. Для мобільного застосунку – React Native.
3. Для серверної частини – Python.
4. Бази даних – Mongo DB.
5. Протокол взаємодії між компонентами системи – GraphQL.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   – «Діаграма використання системи»;
   – «Модель «сутність-зв’язок».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……. 16.11.2022

Розроблення та узгодження технічного завдання…….. 27.11.2022

Розроблення структури програмної системи…….. 15.12.2022

Розроблення дизайну сторінок та графічних елементів……. 03.02.2023

Реалізації програмної системи…….. 15.03.2023

Тестування програмної системи…… 07.04.2023

Підготовка матеріалів текстової частини проєкту…….. 28.04.2023

Підготовка матеріалів графічної частини проєкту……. 12.05.2023

Оформлення технічної документації проєкту……..…... 25.05.2023

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

ПРОГРАМНА СИСТЕМА ОБСЛУГОВУВАННЯ ТУРНІРІВ З БАСКЕТБОЛУ 3х3

Пояснювальна записка

ДП.045490-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро КОМІСАР

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ОСТАПЕНКО

2023

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

AGPL – Affero General Public License (загальна громадська ліцензія Affero GNU);

ASGI – Asynchronous Server Gateway Interface (інтерфейс шлюзу асинхронного сервера);

API – Application Programming Interface (прикладний програмний інтерфейс);

CLI – Command-line interface (інтерфейс командного рядка);

DOM – Document Object Model (об’єктна модель документа);

FIBA – Fédération internationale de basket-ball (Міжнародна федерація баскетболу);

JSON – JavaScript Object Notation;

HTTP – Hyper Text Transfer Protocol (протокол передачі гіпертекстових документів);

MVC – Model-View-Controller (модель-вид-контролер);

MD5 – Message Digest 5;

TCP – Transmission Control Protocol (протокол керування передаванням);

JWT – JSON Web Token;

SHA – Secure Hash Algorithm;

SPA – Single-Page Applications (односторінковий додаток);

БД – Бази даних;

СКБД – Система керування базами даних;

УСЛ – Українська стрітбольна ліга;

ФБУ – Федерація баскетболу України.

ВСТУП

Програмне забезпечення може допомогти людству зменшити кількість часу, що необхідно для виконання певної роботи, зробити її більш якісно, зменшити витрати різноманітних ресурсів, забезпечувати зберігання, аналіз інформації, тощо в різних сфер діяльності, спорт одна з таких.

Проведення турнірів є важливим заходом для популяризації спорту. В залежності від виду спорту та масштабу турніру відрізняються умови їх проведення.

Баскетбол 3х3 – один з різновидів баскетболу, в якому дві команди трійками грають на одному баскетбольному кільці на півкорту.

Метою дипломного проєкту є покращення рівня та полегшення обслуговування турнірів з баскетболу 3х3 шляхом автоматизації.

В загальному, система обслуговування турнірів може мати наступні ролі користувачі: організатори, судді, гравці, тренери, спостерігачі, букмекери; складатися з багатьох модулів, що вирішують різноманітні завдання.

У даному дипломному проєкті розглянуто завдання реєстрації команд, формування розкладу матчів, протоколізація матчів. Розроблено два клієнтських застосунки для організаторів та суддів (асистентів суддів).

1. АНАЛІЗ ПРОБЛЕМИ

1.1. Актуальність проблеми

Баскетбол 3х3 – це різновид баскетболу, який грають трійками, з одним щитом і на півкорту. Він розвинувся з вуличного баскетболу та на зараз є одним із найпопулярнішим міським командним видом спорту у світі. Баскетбол 3х3 став регулярним змаганням на Європейських іграх починаючи з 2015.

Баскетбол 3х3 був доданий до олімпійської програми для літніх Олімпійських ігор 2020 року в Токіо, Японія, як для чоловіків, так і для жінок.

Основний організатор міжнародних змагань з баскетболу 3х3 – FIBA 3х3 – організація, що об'єднує всі національні баскетбольні федерації, яка визначає основні напрями розвитку світового баскетболу [3].

В Україні існує декілька організацій, що займаються популяризацією баскетболу 3х3: ФБУ [4], УСЛ [5]. Діяльність цих організацій спрямована на популяризацію спорту шляхом проведення турнірів. Також ФБУ мають на меті підвищити рівень майстерності баскетболістів, формувати національні збірні.

УСЛ у 2013 році організовувала 31 турнір, у 2019 році – 93 турніри, 2021 році – 46 турніри (з врахуванням карантинних обмежень). Сезони 2020 року та 2022 року були скасовані через форс-мажорні обставини, але не зважаючи на це, турніри проводилися, але не в рамках сезону. На зараз у 2023 році вже заплановано сезон, що складається з 9 турнірів, в яких будуть організовані збори коштів для підтримки Збройних Сил України [5]. Турніри відбудуться з дотриманням всіх норм безпеки – в разі оголошення повітряної тривоги змагання будуть призупинені, а всі присутні мають пройти до найближчого укриття.

Окрім всеукраїнських організацій багато турнірів проводять місцеві організатори, наприклад, громадське об’єднання Баскетбол Львів, деякі

райони м. Києва проводить спартакіади серед школярів та студентів. Баскетбольні клуби проводять турніри серед своїх гравців.

Обслуговування турнірів є актуальною проблемою, оскільки баскетбол 3х3 продовжує набувати популярність, а проведення турнірів набуває систематизованого характеру.

1.2. Завдання процесу проведення турнірів

Програмна система проведення турнірів може виконувати наступні завдання:

Task 1. Реєстрація турнірів:

- Task 1.1. Внесення деталей про турнір.
- Task 1.2. Інформування користувачів про турнір.
- Task 1.3. Збереження даних про турніри.
- Task 1.4. Аналіз турнірів, порівняльна характеристика турнірів.

Task 2. Обслуговування турніру:

- Task 2.1. Реєстрація команд.
- Task 2.2. Формулювання розкладу ігор.
- Task 2.3. Надання інформації користувачам щодо перебігу турніру.
- Task 2.4. Обслуговування матчу.
- Task 2.5. Надання інформації користувачам щодо перебігу матчу.

Як і будь-який спорт, баскетбол 3х3 має чіткий і вичерпний набір правил до гри та до обслуговування матчів. Є затвердження форма протоколу гри.

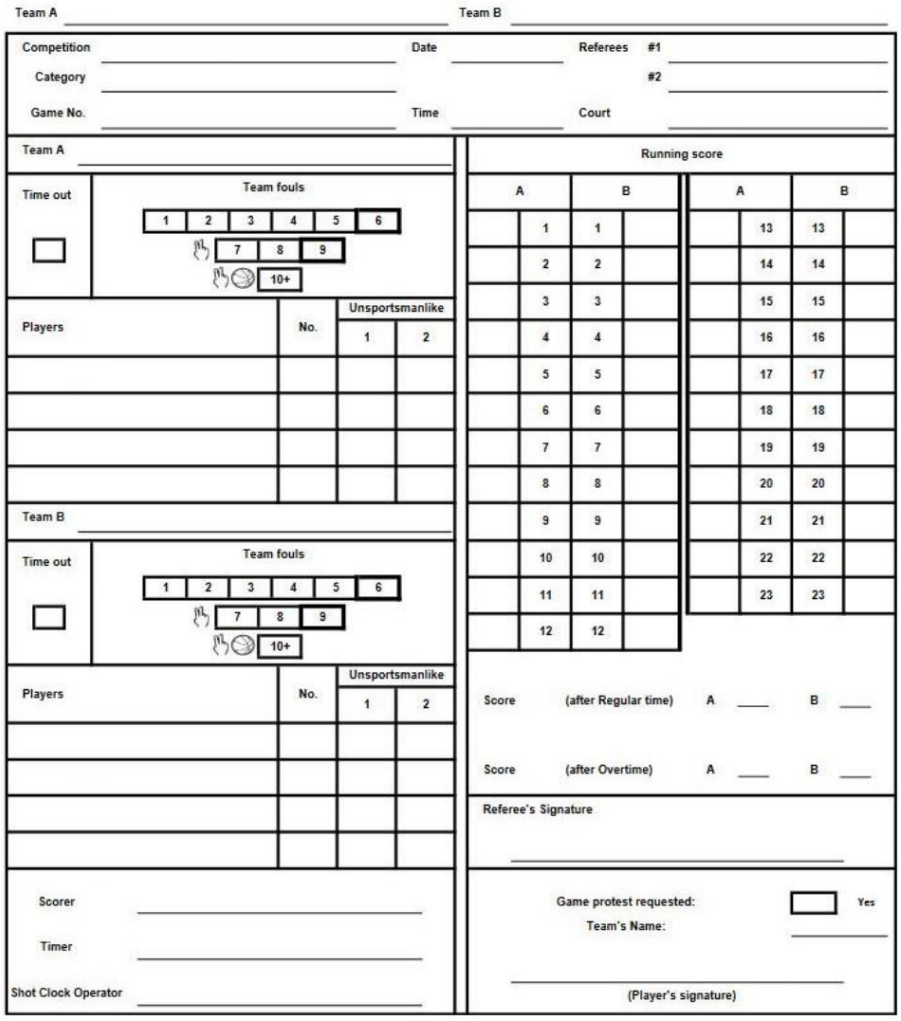


Рис. 1.1. Шаблон протоколу згідно FIBA [6]

1.3. Опитування потенційних користувачів та споживачів

Серед опитаних користувачів були:

- гравці або тренер;
- вболівальники;
- організатори.

Загальні висновки, щодо опитування:

- завдання інформування користувачів реалізовано частково: іноді це реалізовано у паперовому вигляді, що є незручно, деяка відкрита інформація не надається;
- користувачі зацікавлені в отриманні статистики та відслідковуванні декількох матчів паралельно;
- отримано декілька ідей щодо розвинення системи, а саме:

- доступ до системи офлайн;
- фотозвіти;
- питання-відповідь онлайн сесії з організаторами до турніру;

- користувачі зацікавлені в отримані інформації про турнір, перебіг матчів на смартфон.

Були виконано спроби зв’язатися з організаторами шляхом надсилання повідомлень на офіційні сторінки в інстаграмі. Організатори УСЛ, Баскетбол Львів зацікавлені в розробці українського продукту, організатор ФБУ не надав відповіді.

1.4. Аналіз рішень

1.4.1. *Рішення FIBA*

FIBA 3x3 має свою систему для проведення турнірів, що складається з:

- Сайту play.fiba3x3.com, що вирішує завдання Task 1.1-1.4, Task 2.1-2.2.

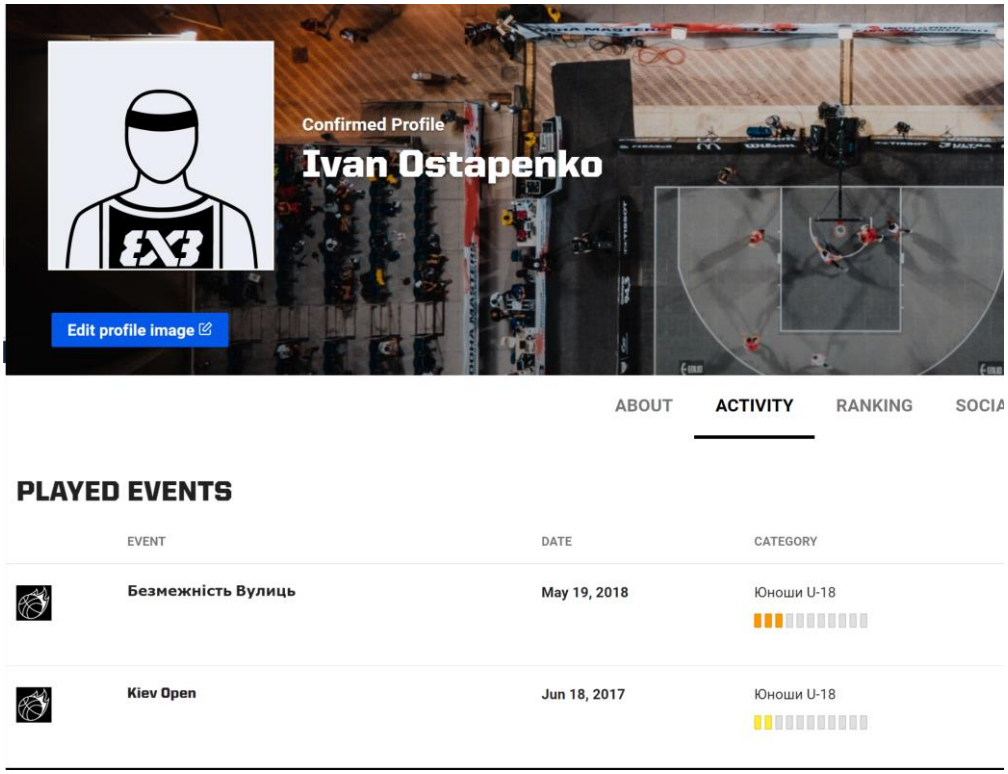


Рис. 1.2. Сторінка користувача сайту рішення від FIBA

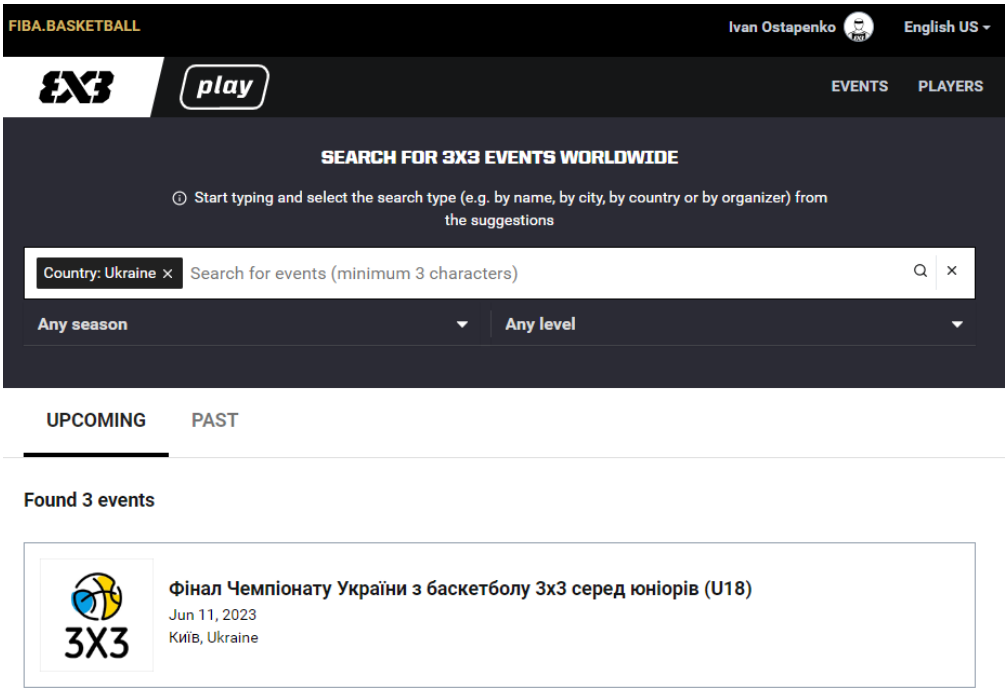


Рис. 1.3. Сторінка пошуку турнірів сайту рішення від FIBA

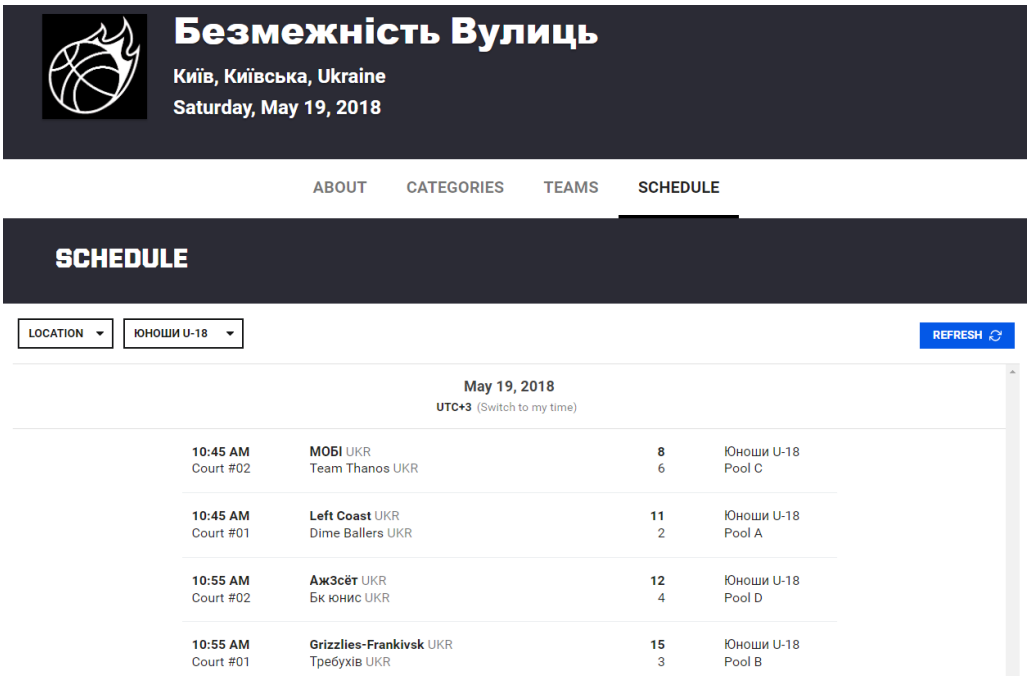


Рис. 1.4. Сторінка розкладу турніру сайту рішення від FIBA

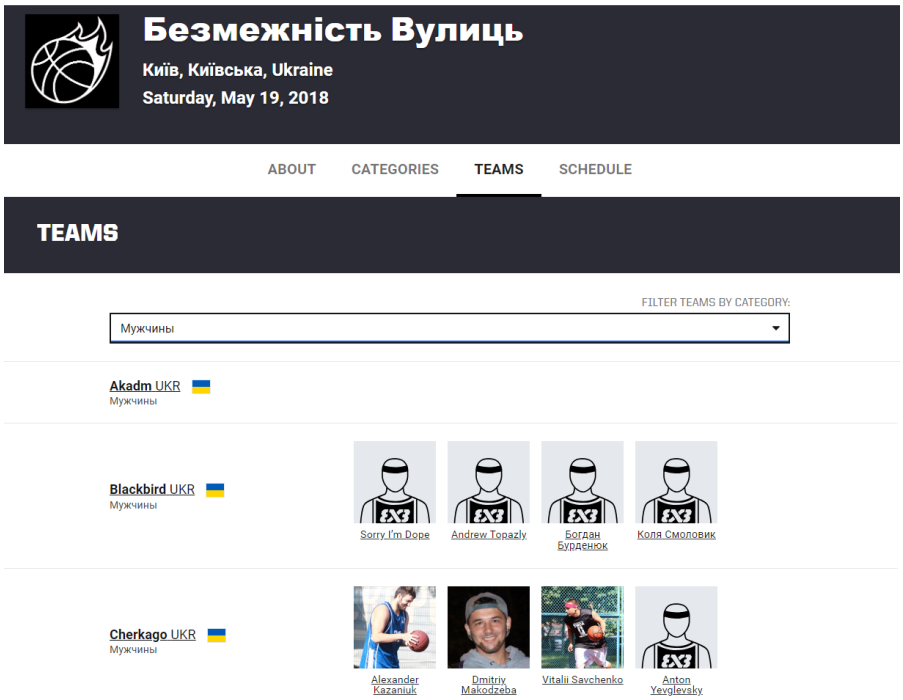


Рис. 1.5. Сторінка команд-учасників турніру сайту рішення від FIBA

- Застосунок для обслуговування матчу, що призначений для:
  - ведення протоколу гри;
  - виведення поточної ситуації гри на екран;
  - надання інформації для онлайн трансляцій.

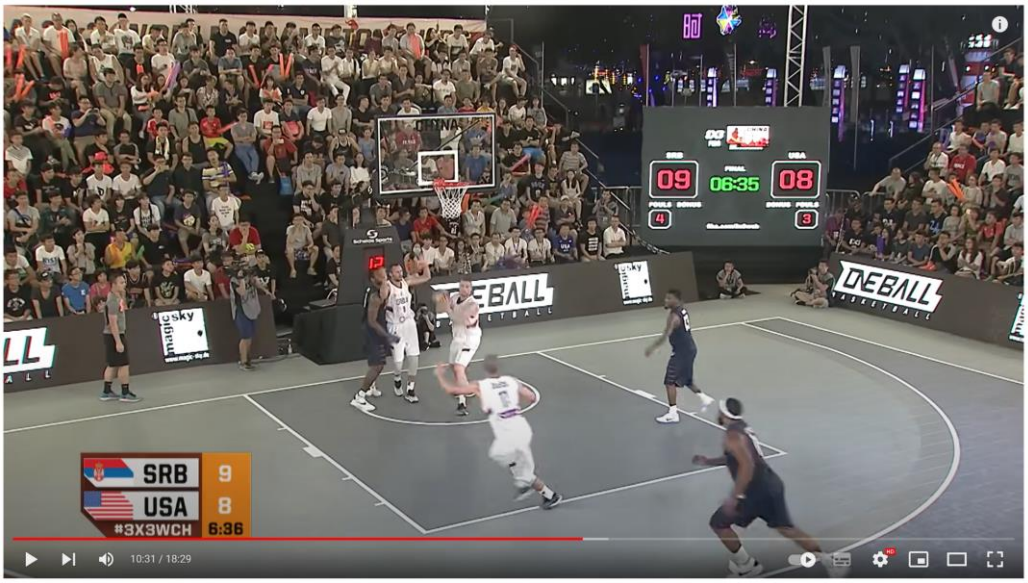


Рис. 1.6. Трансляція гри на якій видно екран та титри інформування поточної ситуації гри

Недоліки рішення від FIBA 3x3:

- Не синхронізованість даних:
  - для оновлення розкладу необхідно вносити зміни вручну;
  - немає миттєвого доступу до протоколів гри після зіграного матчу, доступ надається після внесення даних організатором після завершення турніру.
- Немає інформування користувачів про зміну у розкладі, завершення матчу, тощо.
- Не у всіх організаторів є доступ до використання даної системи.

1.4.2. *Рішення українських організаторів*

Кожен організатор у свій спосіб проводить турніри.

Якщо турнір всеукраїнський, то ФБУ використовує рішення від FIBA. Якщо локальний, то зазвичай для реєстрації використовується система FIBA або шляхом подання паперової заяви. Розклад формується вручну. Протоколізація матчів ведеться у паперовому вигляді.

УСЛ для кожного завдання використовує своє рішення:

- Task 1.1-1.3 (реєстрація турнірів): за допомогою соціальних мереж
- Task 1.4 (аналіз турнірів): не реалізовано;
- Task 2.1 (реєстрація команд): за допомогою соціальних мереж або гугл-форм;
- Task 2.2 (формування розкладу): вручну;
- Task 2.3 та Task 2.5 (надання інформації): на папері, в соціальних мережах, вживу;
- Task 2.4 (обслуговування матчу): на папері.

2. ЗАПРОПОНОВАНЕ РІШЕННЯ

2.1. Опис компонентів системи

Система обслуговування турнірів, що розроблена у даному дипломному проєкті, має на меті вирішення завдань Task 2.2 та Task 2.4. Також у спрощеному вигляді реалізовано інші підзавдання Task 2:

- організатор реєструє команди (Task 2.1);
- сервер забезпечує API для інформування користувачів щодо перебігу матчів та турнірів, реалізовано інформування на клієнтських застосунках організаторів та суддів (Task 2.3 та Task 2.5).

Після аналізу опитування організаторів було вирішено обрати вебплатформу для застосунку для організаторів, оскільки їм необхідно отримувати більше даних про перебіг турніру, мають більші функціональні можливості, мають можливість користуватися ноутбуком під час проведення турніру.

Після аналізу опитування асистентів суддів було обрано кросплатформну мобільну платформу для застосунку, оскільки смартфон є у кожного під рукою.

Система складається з серверного застосунку, вебзастосунку для організаторів, кросплатформного мобільного застосунку для асистентів суддів. Система має клієнт-серверну архітектуру, при чому існує два клієнти у вигляді вебзастосунку та мобільного додатку.

Рис. 2.1. Загальна схема системи

Серверний застосунок забезпечує:

- авторизацію та розмежування доступу:
  - для організаторів;
  - для суддів;
- збереження даних:
  - про турнір;
  - про команди;
  - про майданчики;
  - про суддів;
  - про матчі;
- формулювання даних:
  - розклад матчів (Task2.2);
- інформування інших компонентів системи щодо:
  - змін/оновлені розкладу (Task2.3);
  - перебігу матчів (Task2.5).

Вебзастосунок для організаторів забезпечує можливість:

- введення даних:
  - про команди;
  - про майданчики;
  - про суддів;
  - про розкладу;
- перегляду розкладу, деталей обраного матчу.

Мобільний застосунок для асистентів суддів забезпечує:

- ведення протоколу гри;
- перегляду розкладу.

Система має наступні ролі користувачів:

- організатор;
- асистент судді.

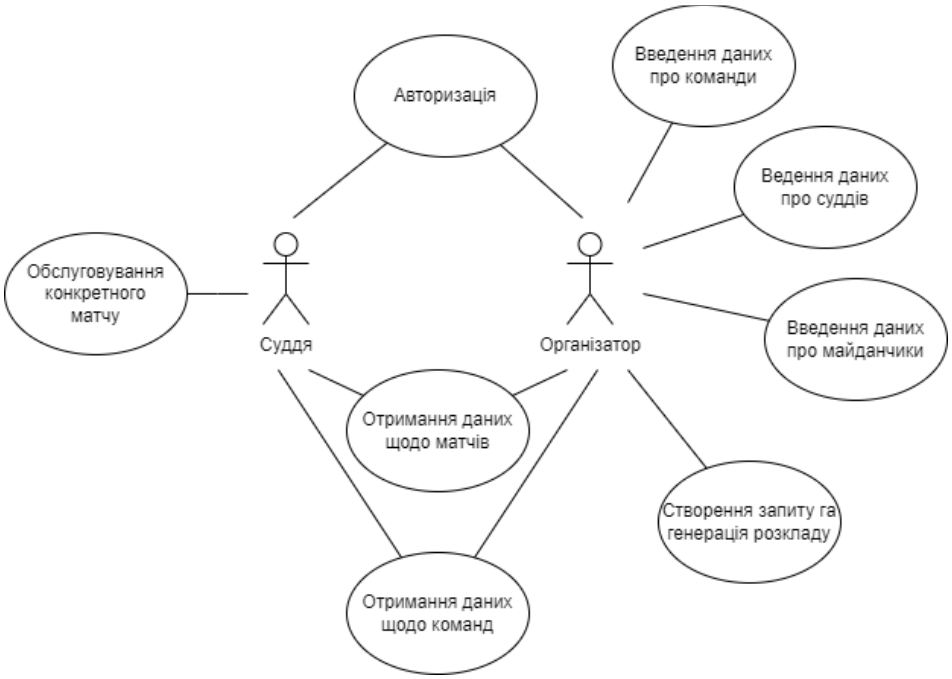


Рис. 2.2. Діаграма використання системи

2.2. Діаграма “сутність-зв’язок”

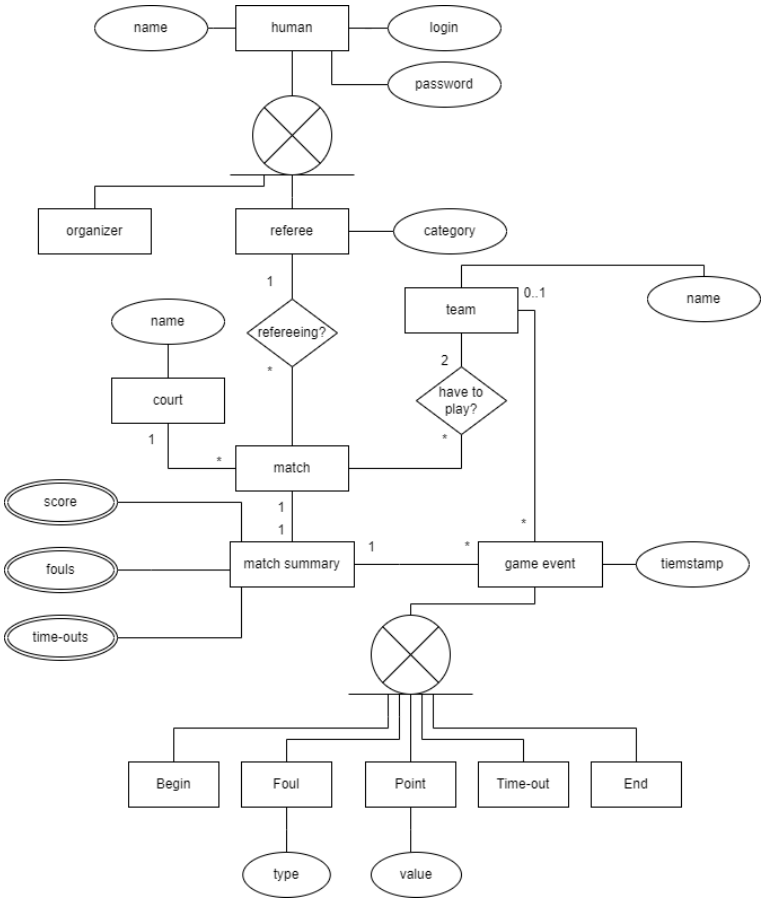


Рис. 2.3. Діаграма“сутність-зв’язок” (entity-relationship diagram)

Основні сутності:

- Людина, що має ім’я, логін та пароль.
- Організатор, що є людиною.
- Суддя, що є людиною, а також має категорію і може обслуговувати матчі.
- Команда має назву.
- Гра, що складається з:
  - двох команд, що грають;
  - судді, що обслуговує матч;
  - спортивний майданчик, на якому відбувається гра;
  - час за розкладом.
- Звіт до гри містить поля:
  - загальний рахунок;
  - кількість фолів команд;
  - кількість використаних тайм-аутів команд;
  - посилання на ігрові моменти.
- Ігровий момент містить час та додаткові дані залежно від типу ігрового моменту:
  - початок матчу;
  - фол (порушення правил) містить поле “тип фолу”;
  - набране очко містить поле “кількість очків”;
  - взяття таймауту;
  - завершення матчу.

3. ОБРАННЯ ТЕХНОЛОГІЙ

3.1. Обрання протоколу взаємодії між компонентами

Найпопулярніші протоколи взаємодії в архітектурі клієнт-сервер: HTTP, WebSocket, GraphQL [7].

HTTP – це протокол, в якому обмін повідомленнями відбувається за схемою “запит-відповідь” [8].

Основні переваги:

- Вебкешування, що дозволяє тимчасово зберігати (кешувати) дані задля зменшення серверних затримок. Система вебкешу зберігає копії даних, що проходять через неї. Подальші запити можуть бути виконані з кешу за певних умов.
- Легкість документування API. Для цього існують спеціальні інструменти, наприклад, Swagger [9].

Основні недоліки:

- Неможливість відправки повідомлення з сервера до клієнта (для досягнення такого ефекту клієнт може періодично відправляє запити, або використовувати інший протокол паралельно).

WebSocket – це протокол, що призначений для обміну даними між клієнтом та сервером в режимі реального часу. Реалізується двонаправленим повнодуплексним канал зв'язку через один TCP-сокет [10].

Основна перевага – це висока швидкість передачі даних, в порівнянні з HTTP, немає необхідності для кожного запиту створювати нове підключення, підключення створюється один раз на початку взаємодії.

Основні недоліки:

- Більше навантаження на сервер, оскільки йому завжди треба утримувати підключення з клієнтами.
- Складність у документуванні API.

GraphQL – це мова запитів і маніпуляції даними з відкритим кодом для API і середовище виконання для обслуговування запитів з наявних

даних. Працює на основі WebSockets [11].

Основні переваги:

- Легкість у документуванні API. Схема GraphQL вже є документацією, у використані інших інструментів немає потреби.
- Швидкість передачі даних (працює на основі WebSocket).
- Можливість клієнту вказувати які саме дані він хоче отримати за допомогою одного запиту.

Основний недолік – більше навантаження на сервер (працює на основі WebSocket).

Проранжуємо протоколи за допомогою порівняльної таблиці (табл. 3.1).

Таблиця 3.1

Порівняльна таблиця протоколів

Параметр	Вага	HTTP	WebSocket	GraphQL
Швидкість передачі даних	1	3	1	2
Легкість у використанні	2	1	3	2
Швидкість розробки	3	2	3	1
Бажання розробника	2	2	3	1
Сумарно		15	22	10

Обрано протокол GraphQL, оскільки він має найменше сумарне число у табл. 3.1. Основним фактором вибору є швидкість розробки.

Основними елементами GraphQL є input, type, mutation та subscription:

- Тип (type) визначає структуру даних в GraphQL. Він описує, які

поля має об'єкт і який тип даних у кожного поля. Серед стандартних типів є int, float, string, boolean, ID. Існує можливість створення своїх типів даних, що містять поля різних типів даних. Клієнт може виконати запит на отримання цих даних за допомогою HTTP-запиту.

- Вхід (input) використовується для передачі даних в мутації (mutation).
- Мутація (mutation) використовується для зміни або оновлення даних на сервері, представляє операції, що змінюють стан сервера, наприклад створення, оновлення або видалення даних. Клієнт може виконати мутацію за допомогою HTTP-запиту.
- Підписка (Subscription) дозволяє клієнту підписатися на зміни даних на сервері, використовується для отримання даних в режимі реального часу за допомогою з’єднанню по websocket між клієнтом та сервером.

3.2. Обрання технології для реалізації серверної частини

На зараз найпопулярніші мови програмування для реалізації серверу: JavaScript, Python, Java, PHP [12]. Перевірено, що для кожної мови програмування з переліку існує реалізація сервера GraphQL [11]. Поділимо ці мови за критерієм мова, що компілюється та мова, що інтерпретується.

Підхід мов, що компілюються: в процесі компіляції з вихідного коду створюється машинний файл, що може бути запущений на цільовій машині. Основна перевага – це висока швидкість роботи, оскільки цільова машина напряму отримує команди застосунку. Основний недолік – це швидкість компіляції.

Мови, що інтерпретується: в процесі інтерпретації вихідний код за допомогою інтерпретатора перетворюється у відповідні команди цільовій машині в режимі виконання програми. Основна перевага – це можливість майже одразу запустити застосунок після змін у вихідному коді, що значно

пришвидшує розробку програмного продукту. Основний недолік – це низька швидкість роботи, оскільки існує додатковий застосунок інтерпретатор, що перетворює команди програми на команди зрозумілі цільовій машині.

Таблиця 3.2

Порівняльна таблиця мов

Параметр	Вага	Мова, що компілюється	Мова, що інтерпретується.
Швидкість роботи	1	1	2
Швидкість розробки	3	2	1
Легкість налагодження	2	1	2
Бажання розробника	2	2	1
Сумарно		13	11

Обрано мови, що інтерпретується, оскільки вони мають менше сумарне число у табл. 3.2. Основним фактором вибору була швидкість розробки. Найпопулярніші мови, що інтерпретуються: Python, JavaScript, PHP.

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією, що підтримує парадигми програмування: об’єктно-орієнтований, процедурний, функціональний, аспектно-орієнтований [13, 14].

Переваги:

- відкритий код;
- містить велику кількість модулів, що включають документацію та легко можуть бути підключені до проєкту;
- відносно легкий та уніфікований синтаксис;
- висока швидкість розробки.

Недоліки:

- низька швидкодія;
- відсутність статичної типізації;
- неможливість модифікації вбудованих класів (наприклад, int, str, list).

JavaScript – це високорівнева, інтерпретована мова програмування, яка зазвичай використовується для розробки динамічних вебсайтів і вебдодатків, також можна використовувати для написання серверів. Існує багато бібліотек для цього, найпопулярніші з них – Node.js, Express.js, Koa.js, Next.js.

Особливості JavaScript для написання серверу:

- асинхронність;
- модульність;
- підтримка пакетного менеджера npm.

PHP – це високорівнева скриптова мова програмування, яка використовується для розробки серверних додатків та динамічних вебсайтів.

Особливості PHP:

- проста мова для вивчення;
- велика спільнота;
- часті вдосконалення оновлення мови.

Таблиця 3.3

Порівняльна таблиця мов, що інтерпретуються

Параметр	Вага	Python	JavaScript	PHP
Наявність документації	2	1	3	2
Швидкість розробки	2	1	2	3

Продовження табл. 3.3

Параметр	Вага	Python	JavaScript	PHP
Бажання розробника	3	2	1	3
Сумарно		10	13	21

Обрано мову програмування Python, оскільки вона має менше сумарне число у табл. 3.3. Основними факторами для вибору була швидкість розробки та наявність документації.

3.3. Обрання бази даних для серверної частини

Існує декілька типів баз даних: реляційні, нереляційні (документоорієнтовані, графові).

До основних переваг реляційних БД відносять:

- Структурованість. Дані організовані у вигляді таблиць.
- Цілісність даних. Є можливість встановити обмеження та зв’язки між таблицями.
- Рішення для роботи у багатокористувацькому режимі. Залежно від конкретної реляційної БД мають рішення щодо уникнення одночасного доступу/запити на зміни даних.
- Можливість проведення транзакцій, що дозволяє проводити групу операцій як одна.

До основних недоліків реляційних БД відносять:

- Низька гнучкість системи. Якщо схема БД змінюється, а вже існує певна кількість записів, то можуть виникати трудності з переходом з однієї схеми на іншу (міграцією).
- Складність масштабування. При великій кількості даних збільшується час обробки даних, збільшується навантаження на реплікацію.

Документоорієнтовані БД зберігають дані у вигляді документів.

Основні переваги:

- Гнучка схема, що означає, що кожен документ може мати свою власну структуру. Це дозволяє додавати, видаляти або змінювати поля в документах без необхідності зміни схеми бази даних.
- Підтримка вкладених структур, що дозволяє вкладати один документ в інший, що надає можливість створювати багатоструктурні моделі даних.
- Горизонтальне масштабування, що означає наявності можливості легкого масштабування системи шляхом додавання нових серверів або вузлів. Це збільшує загальну місткість БД, а також розподіляє навантаження серед вузлів.

Основним недоліком є недостатня підтримка складних операцій, наприклад, операцій JOIN або віконних функцій.

Враховуючи схему даних, що зображена на рис. 2.3, доцільно обрати документно-орієнтовану БД, оскільки є декілька спеціалізацій (Specialization), наприклад, “людина” може бути організатором, суддею або гравцем; ігровий момент, може бути набраним очком, фолом, використання тайм-ауту або заміна гравця.

Обрано одну із найпопулярніших безкоштовних документо-орієнтованих бази даних – MongoDB.

MongoDB – документо-орієнтована СКБД. Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3. Зберігає документи в JSON-подібному форматі. Має відносно гнучку мову запитів. Може створювати індекси для різних типів даних. Може працювати до парадигми Map/Reduce. Має засоби шардінгу, реплікації. Має інструменти для журналювання операцій до БД. Ефективно зберігає бінарні дані великих обсягів, наприклад, фото та відео.

З недоліків була велика кількість проблем з безпекою, що поступово вирішилися в наступних версіях [15].

3.4. Обрання технології для написання клієнтської частини для організаторів

Найпопулярніші фреймворки для написання клієнтських частин вебзастосунків: Vue, React, Angular [16].

Vue.js – це прогресивний JavaScript-фреймворк для розробки користувацького інтерфейсу. Він використовується для створення SPA та динамічних інтерфейсів для вебсайтів.

Особливості Vue.js:

- Легкий для вивчення.
- Використовує компонентний підхід та принцип реактивного зв’язування даних.
- Містить багато готових рішень, бібліотек та плагінів.

React – відкрита JavaScript бібліотека для створення клієнтських частин вебзастосунків. Розробляється Meta (раніше Facebook) і спільнотою індивідуальних розробників. Серед найвідоміших користувачів React є Khan Academy, Netflix, Yahoo, Atlassian. Відповідає патерну модель-вид-контролер (MVC). Основною перевагою фреймворку є підтримка віртуального DOM браузера, що дає змогу фреймворку оновлювати не всю сторінку загалом, а лише елементи, у яких змінилися дані. У поєднанні з іншими JavaScript бібліотеками дозволяє розробнику швидко створювати якісні вебзастосунки [17].

Angular – це фреймворк для розробки вебдодатків, розроблений компанією Google. Він надає повний набір інструментів для побудови складних SPA. Angular базується на TypeScript, розширенні JavaScript, яке надає можливості статичної типізації. Це може пришвидшити виявлення помилок та покращує підтримку інтегрованих інструментів розробки. Angular використовується для розробки великих, масштабованих вебдодатків.

Таблиця 3.4

Порівняльна таблиця інструментів розробки вебзастосунків

Параметр	Вага	Vue	React	Angular
Наявність документації	2	3	1	2
Швидкість розробки	3	2	1	3
Наявність компонент	2	3	2	1
Сумарно		18	9	15

Обрано React, оскільки він має найменше сумарне число у табл. 3.4. Основним фактором вибору є швидкість розробки.

3.5. Обрання технології для написання клієнтської частини для суддів

Найпопулярніші інструменти для розробки кросплатформенних мобільних додатків: React Native, Flutter, Qt (QML) [18].

React Native – відкрита JavaScript бібліотека для створення інтерфейсу користувача для Android, Android TV, iOS, macOS, tvOS, Web, Windows і UWP. Принципи роботи React Native практично ідентичні React, що стало ще однією причиною обрання відповідної технології [19].

Переваги:

- швидкість і легкість розробки;
- кросплатформний інструмент;
- існування гарних інструментів для відладки;
- велика спільнота розробників;
- сумісність з 3d party нативними бібліотеками.

Недоліки:

- швидкодія застосунку нижче ніж у випадку, якщо написано

нативною мовою платформи;

- не ефективний для складних інтерфейсів;
- не велика кількість модулів.

Flutter – це відкритий фреймворк розробки користувацького інтерфейсу, розроблений компанією Google. Має власний рушій рендерингу, що показує кращу продуктивність серед інших кросплатформних фреймворків. Надає багато готових компонентів стилю. Підтримує концепцію віджетів до графічного інтерфейсу. Має гарний інструментарій для пришвидшення розробки.

Qt – це потужна багатоплатформова фреймворк для розробки програмного забезпечення, що використовується для створення додатків з графічним інтерфейсом користувача. Має реалізації на Python та C++/ Одним з ключових компонентів Qt є QML (Qt Meta-Object Language) – декларативна мова для опису інтерфейсів користувача, що містить реалізацію лише на C++. Основні особливості C++: висока швидкодія роботи програми, низька швидкість розробки.

Таблиця 3.5

Порівняльна таблиця інструментів розробки кросплатформенних мобільних додатків

Параметр	Вага	React Native	Flutter	Qt (QML)
Наявність документації	1	1	2	3
Швидкість розробки	3	1	2	3
Бажання розробника	2	1	3	2
Сумарно		6	13	16

Обрано React Native, оскільки він має найменше сумарне число у табл. 3.5. Основним фактором вибору є швидкість розробки.

3.6. Загальна архітектура програмної системи

Загальну схему обраних технологій представлено на рис. 3.1.

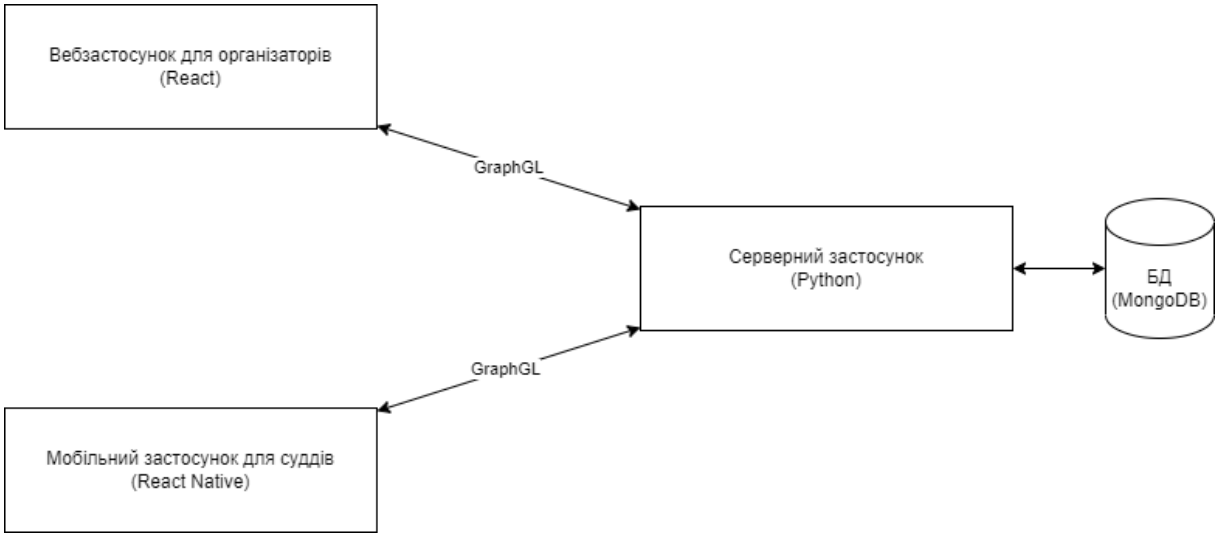


Рис. 3.1. Загальна схема програмної системи

Обрано наступні технології:

- протокол взаємодії “клієнт-сервер” – GraphQL;
- СКБД – MongoDB;
- фреймворк для клієнтського вебзастосунку – React;
- фреймворк для клієнтського мобільного застосунку – React Native.

4. РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

4.1. GraphQL схема

Створені сутності (input та type) представлено у табл. 4.1.

Таблиця 4.1

Перелік сутностей GraphQL схеми

Сутність	Вхідні дані	Вихідні дані

Суддя	input JudgeInput{
	login: String!
	name: String!
	password: String!
	category: String!
	}
		type Judge{
		login: String!
		name: String!
		category: String!
		}

Команда	input TeamInput{
	name: String!
	}
		type Team{
		_id: ID!
		name: String!
		}

Майданчик	input PoolInput{
	name: String!
	}
		type Pool{
		_id: ID!
		name: String!
		}

Матч	Генерується сервером	type Match{
		_id: ID!
		team_1: Team
		team_source_1: Match
		team_2: Team
		team_source_2: Match
		planned_time: Float
		pool: Pool
		judge: Judge
		round: String
		events: [GameEvent]
		summary: MatchSummary
		}

Ігрова подія	input GameEventInput{
	event_type: String!
	timestamp: Float!
	match: Int!
	team: Int
	score: Int
	}
		type GameEvent{
		_id: ID!
		event_type: String!
		timestamp: Float!
		match: Match!
		team: Team
		score: Int
		}

Звіт матчу	Генерується сервером, на основі ігрових моментів	type MatchSummary{
		score_1: Int!
		score_2: Int!
		fouls_1: Int!
		fouls_2: Int!
		timeout_1: Int!
		timeout_2: Int!
		state: String!
		winner_team: Int
		}

Продовження табл. 4.1

Сутність	Вхідні дані	Вихідні дані

Пакет авторизації	Генерується сервером	`type TokenAuth{`
		`  token: String!`
		`  role: String!`
		`}`

Створені запити (query) представлено у табл. 4.2.

Таблиця 4.2

Перелік запитів GraphQL схеми

Запит	Опис запиту
`judges: [Judge]`	Для отримання даних про суддів
`teams: [Team]`	Для отримання даних про команди
`pools: [Pool]`	Для отримання даних про майданчики
`matches: [Match]`	Для отримання даних про матчі
`isStarted: Boolean!`	Для отримання даних про початок турніру

Всі запити у відкритому доступі.

Створені мутації (mutation) представлено у табл. 4.3.

Таблиця 4.3

Перелік мутацій GraphQL схеми

Мутація	Опис мутації
`tokenAuth(username: String!, password: String!): TokenAuth`	Для авторизації
`addJudge(judge_input: JudgeInput!): Judge!`	Для додавання судді
`addTeam(team_input: TeamInput!): Team!`	Для додавання команди
`addPool(pool_input: PoolInput!): Pool!`	Для додавання майданчика

Продовження табл. 4.3

Мутація	Опис мутації
`addGameEvent(event_input: GameEventInput!): GameEvent!`	Для додавання ігрового моменту
`generateSchedule(start_time: Float!, game_duration: Float!, game_pause: Float!): [Match]`	Для генерації розкладу

Мутація авторизації у відкритому доступі.

Для клієнтів адміністраторів доступні мутації: додавання судді, команди, майданчика та генерації розкладу.

Для клієнтів суддів доступна лише мутація додавання ігрового моменту.

Створені підписки (subscription) представлено у табл. 4.4.

Таблиця 4.4

Перелік запитів GraphQL схеми

Підписка	Опис підписки
`tournamentStarted: Boolean!`	Інформування про початок турніру
`judgesUpdated: [Judge]`	Інформування про зміни даних суддів
`teamsUpdated: [Team]`	Інформування про зміни даних команд
`poolsUpdated: [Pool]`	Інформування про зміни даних майданчиків
`matchesUpdated: [Match]`	Інформування про зміни даних матчів
`tournamentStarted: Boolean!`	Інформування про початок турніру

Всі підписки у відкритому доступі.

Під час реалізації виявлено недолік GraphQL протоколу, а саме: він не підтримує перелічувані типи (enum) під час мутацій (mutation) [20].

4.2. Реалізація серверної частини

Діаграма потоків серверу зображено на рис. 4.1.

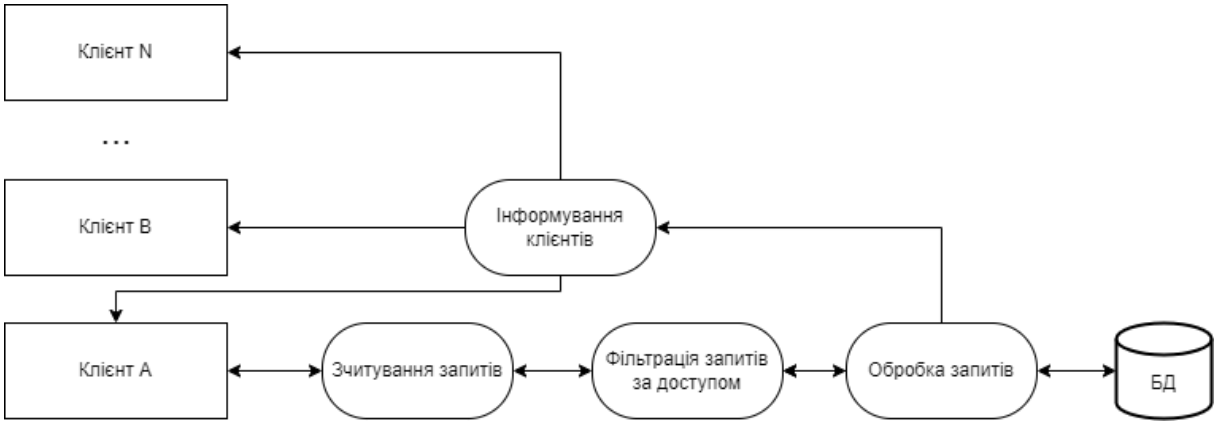


Рис. 4.1. Діаграма потоків серверу

Зчитування даних за протоколом GraphQL відбувається за допомогою бібліотеки Ariadne [21].

Фільтрація запитів за доступом відбувається за допомогою перевірки JWT-токену з заголовку зчитаного запиту.

JWT – це стандарт токена доступу на основі JSON. Складається з трьох рядків розділені символом “.”. Перші два рядки – це заголовок та тіло (вміст). Третій рядок – це підпис, що створений хешуванням визначеним алгоритмом перших двох рядків [22, 23].

Клієнт має змогу отримати відповідний токен як відповідь на запит авторизації передавши логін і пароль. Паролі користувачів зберігаються у захешованому вигляді у базі даних.

Для зчитування jwt-токену і додавання контексту, щодо користувача, що зробив запит, було встановлено параметр-функцію context_value класу GraphQL. Також встановлено параметр-функцію обробки помилок error_formatter класу GraphQL, для зменшення серверних деталей помилок заради безпеки серверу [21].

Налаштування мережевого доступу до сервера змінено за допомогою бібліотеки starlette. Starlette – це легкий набір інструментів ASGI, який

ідеально підходить для створення асинхронних вебсервісів на Python. Містить підтримку HTTP, WebSocket, сесій та хуків [24].

Алгоритм хешування паролей обирався серед SHA256 та MD5. З 2011 року MD5 алгоритм вважається ненадійним [25]. Алгоритм SHA256 працює у 2.5 рази повільніше ніж MD5 [26]. Так як для системи швидкодія алгоритму SHA256 задовольняє, тому було обрано алгоритм SHA256.

Для роботи з MongoDB використовується бібліотека pymongo [27].

Для запуску сервера, використовується утиліта uvicorn. Uvicorn – це реалізація вебсервера ASGI для Python. Зараз Uvicorn підтримує HTTP/1.1 і WebSockets, чого достатньо для використання GraphQL взаємодії [28].

Алгоритм обробки мутацій на додавання сутностей команд, суддів, майданчиків полягає в записі їх у базу даних і відправка повідомлення про зміну клієнтам-підписникам.

Рис. 4.2. Алгоритм обробки мутацій на додавання сутностей команд, суддів, майданчиків

Алгоритм формування розкладу полягає у створенні оптимального можливого розкладу.

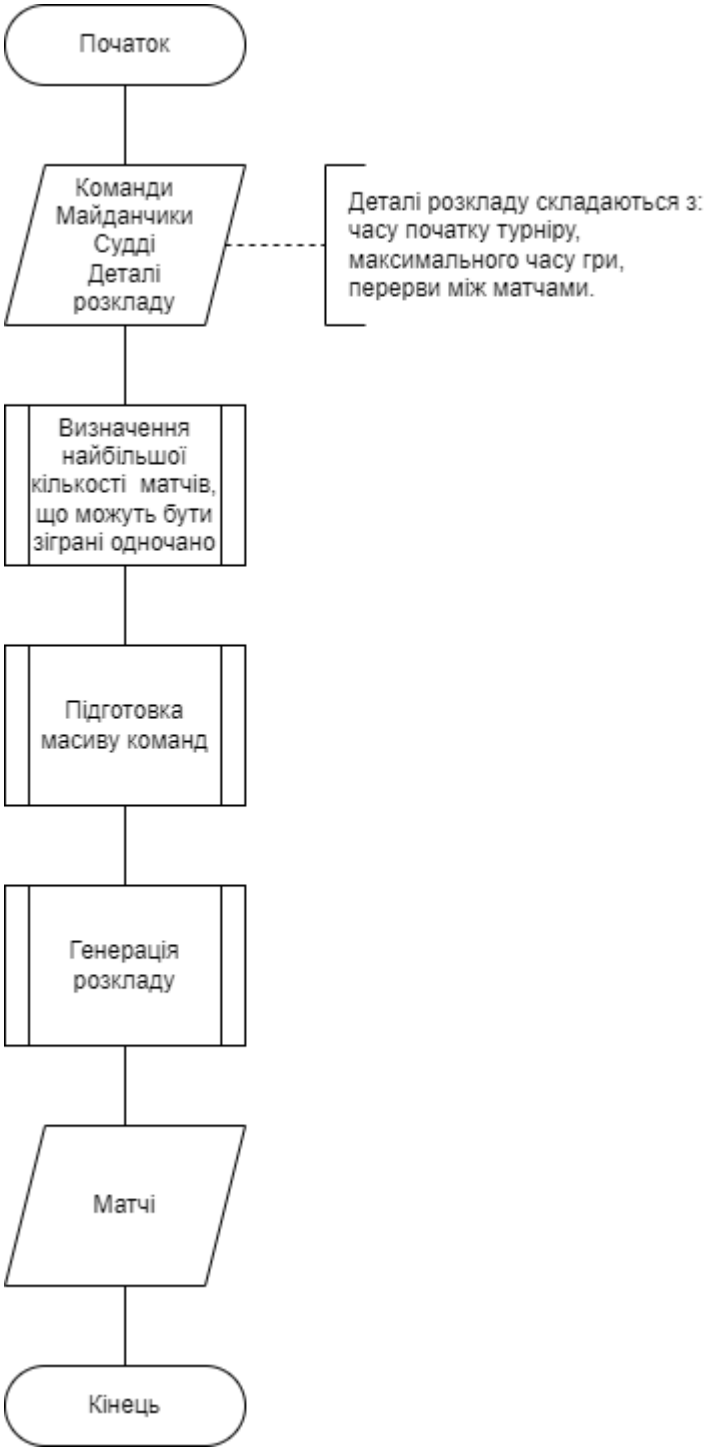


Рис. 4.3. Алгоритм формування розкладу

Процес визначення найбільшої кількості матчів, що може бути зіграні одночасно, полягає у знаходження мінімального числа між кількістю

майданчиків та суддів.

Процес підготовки масиву команд полягає у:

- Збільшенні кількості команд до числа, що кратне 2^k, шляхом додавання пустих команд. Кратність кількості команд на 2^k є необхідною умовою для використання олімпійської системи.
- Перемішуванні випадковим чином утвореного масиву.

Алгоритм генерації розкладу подано у вигляді псевдокоду.

Лістинг 4.1. Псевдокод генерації розкладу

```
input: teams, pools, judges, start_time, max_game_time, pause_duration,
max_parllel_games

match_time = max_game_time + pause_duration
round_count = math.log2(len(teams)))
all_matches = []
matches = []

for i in range(len(teams) // 2):
    match = {
        "team1": teams[2 * i],
        "team_source_1": None,
        "team2": teams[2 * i + 1],
        "team_source_2": None,
        "planned_time": start_time + i // max_parallel_games * (match_time),
        "pool": pool_ids[i % len(pools)],
        "judge": judges_ids[i % len(judges)],
        "round": f"1/{2 ** (round_count - 1)}"
    }
    new_start_time = match["planned_time"] + match_time
    matches.append(match)

for i in range(from: round_count - 1, to: 1):
    matches_in_round = []
    start_time = new_start_time
    for j in range(len(matches) // 2):
        match = {
            "team1": -1,
            "team_source_1": matches[2 * j],
            "team2": -1,
            "team_source_2": matches[2 * j + 1],
            "planned_time": start_time + j // max_parallel_games * (match_time),
            "pool": pools[j % len(pools)],
            "judge": judges[j % len(judges)],
            "round": f"1/{2 ** (i - 1)}"
        }
        new_start_time = match["planned_time"] + match_time
        matches_in_round.append(match)
    matches = matches_in_round
    all_matches.append(matches_in_round)



output: all_matches
```

Інформування користувачів відбувається за допомогою можливостей

Ariadne [21] та Broadcaster [29].

4.3. Реалізація клієнтських частин

4.3.1. *Загальний підхід до реалізації клієнтських частин*

Клієнтські частини реалізовано за допомогою мови програмування JavaScript, фреймворків React та React Native для вебзастосунку та мобільного застосунку відповідно.

Фреймворки React та React Native схожі за принципом роботи. Вони будують віртуальне дерево, підтримують MVC патерн, використовують хуки.

Хук – це спеціальна функція, яка дозволяє «підключатися» до функцій React [17].

Було використано хуки React (React Native):

- useState – це хук стану функціональних компонентів;
- useEffect – це хук, що виконується після рендерингу компонента, або про зміні встановлених в хук залежностей;
- useContext – це хук, що дозволяє передавати контекст (дані) між компонентами, а також автоматично оновлювати залежні компоненти при зміні даних контексту.

Для взаємодії з сервером за допомогою протоколу GraphQL обрано бібліотеку Apollo Client. Apollo Client – це повнофункціональний кешуючий клієнт GraphQL з інтеграцією для React, React Native, Angular тощо [30].

Було використано хуки Apollo Client:

- useQuery – хук для виконання GraphQL запитів;
- useMutation – хук для виконання GraphQL мутацій;
- useSubscription – хук для реалізації GraphQL підписки.

Для роботи з компонентами вводу даних використовується принцип подвійного зв’язування [31].

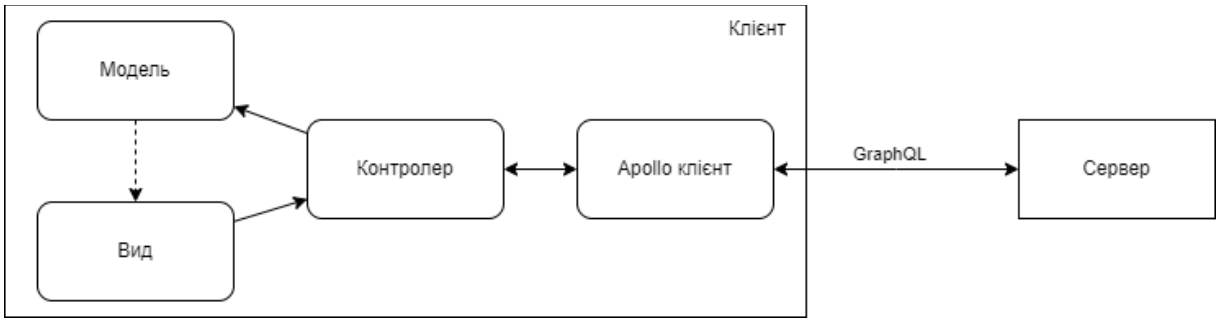


Рис. 4.4. Загальна архітектура клієнтських застосунків

Зв’язки між контролером, моделю та видом відповідають патерну MVC. Контролер може викликати методи Apollo клієнта, наприклад, мутацію tokenAuth для авторизації і залежно від відповіді, змінювати модель. Apollo клієнт може генерувати події щодо отримання нових даних шляхом підписки, які обробляє контролер.

4.3.2. *Реалізація клієнтської частини для організаторів*

Для розробки застосунку використовується бібліотека vite, що надає доступ до інструментів конфігурації та відладки [32].

Застосунок створено за парадигмою SPA. SPA – це застосунок, що вміщується на одній сторінці з метою забезпечення користувачу досвід близький до користування настільною програмою, а також зменшує навантаження на мережу.

Було розроблено 6 сторінок: Login, About, Teams, Pools, Judges, Schedule.

Сторінка Login забезпечує авторизацію користувача і запису сховище отриманого JWT-токену від відповіді на мутацію авторизації.

Сторінка About містить інформацію про систему.

Сторінки Teams, Pools, Judges призначені для перегляду і додавання відповідних сутностей. Загальна схема UI компонентів сторінок зображена на рис. 4.5.

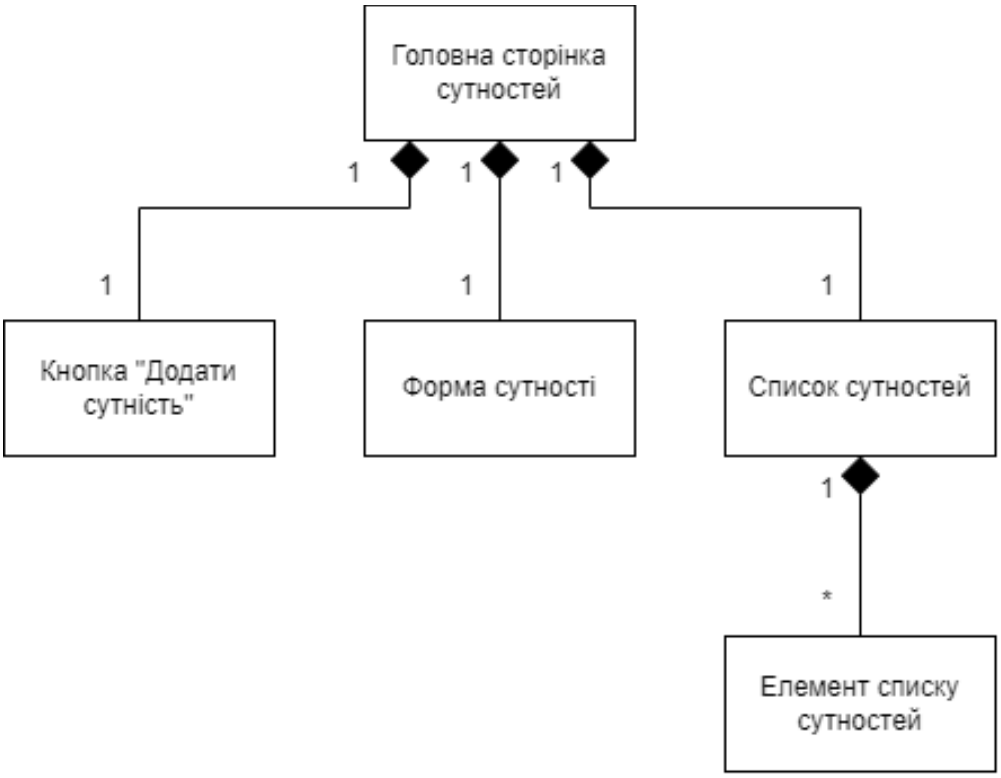


Рис. 4.5. Загальна схема UI компонентів сторінок Teams, Pools, Judges

Сторінки Teams, Pools, Judges мають два стани:

- До початку турніру, під час якого доступна кнопка “Додати <сутність>”, при натисканні якої стає доступна форма для додавання сутності, доступний список доданих сутностей.
- Після початку турніру, під час якого доступний лише перегляд списку сутностей.

Головна сторінка сутностей містить хуки для отримання сутностей з серверу (useQuery), для підписки щодо змін сутностей (useSubscription).

Форма сутності містить хук для додавання сутності (useMutation).

Сторінка Schedule має схожу поведінку з сторінками Teams, Pools, Judges. Окрім того, що форма додавання сутності змінена на форму внесення даних для генерації турніру (час початку турніру, максимальний час гри, перерва між матчами), що може змінюватися залежно від турніру. Сутностями для сторінки Schedule є матчі, що мають стани зображені на рис. 4.6.

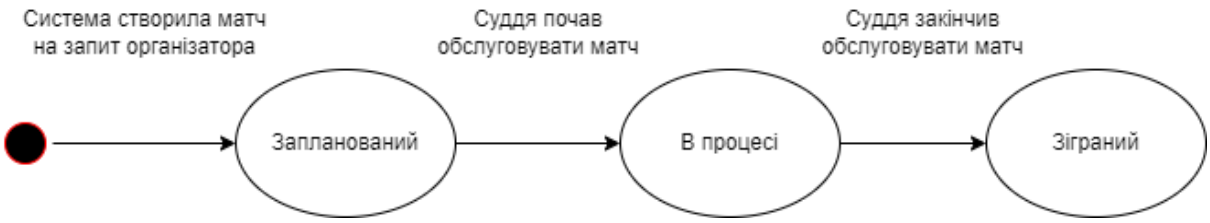


Рис. 4.6. Діаграма станів матчу

Сутність матчу створюється сервером на запит організатора створити розклад. Початковий стан сутності є “Запланований”.

В стані “Запланований” клієнт мобільного застосунку має можливість почати обслуговувати матч, що переводить матч у стан “В процесі”.

В стані “В процесі” клієнт мобільного застосунку має можливість вводити ігрові моменти та закінчити матч, що переводить матч у стан “Зіграний”.

В станах “В процесі” та “Зіграний” клієнт вебзастосунку має можливість переглянути протокол гри.

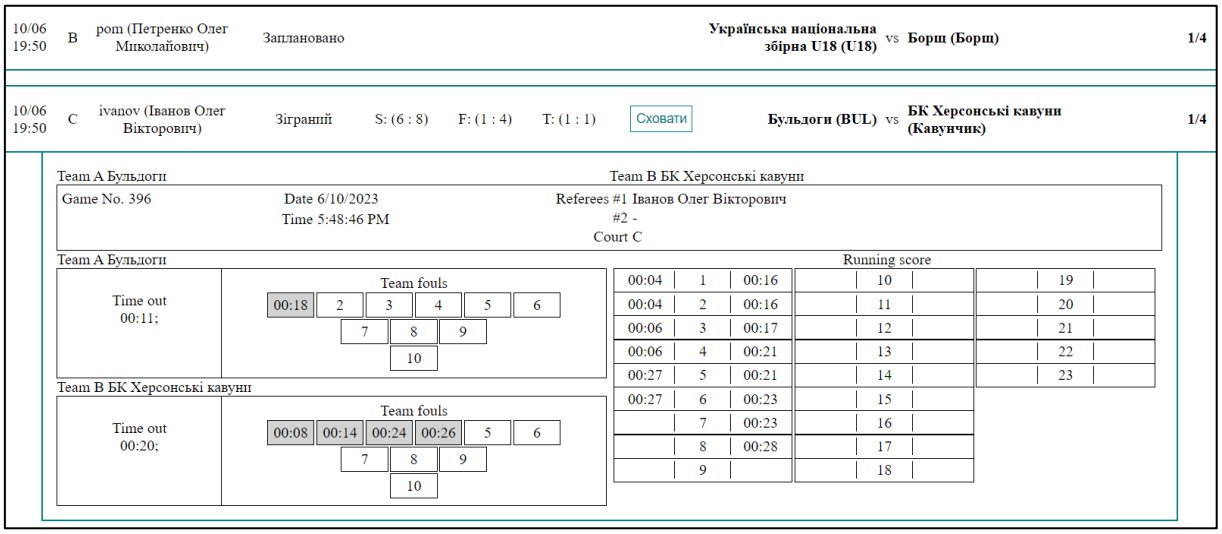


Рис. 4.7. Інтерфейс користувача вебзастосунку. Перегляд протоколу гри.

4.4.3. *Реалізація клієнтської частини для суддів*

У розробників React Native є декілька інструментів до створення додатків, а саме: React Native CLI та expo. Рекомендують для початківців у

мобільній розробці обирати expo, оскільки він містить набір інструментів та сервісів над React Native, що значно прискорює та полегшує розробку.

Таблиця 4.5

Порівняльна характеристика React Native CLI та expo

Параметр	React Native CLI	expo
Можливість інтеграції нативних модулів, наприклад, на Java або Object-C	Так	Ні
Розмір застосунку “Hello world”	5 МБ	25 МБ
Необхідна наявність Android Studio або XCode для запуска проєктів	Так	Ні
Легкість поширення, сумісного використання застосунок за допомогою посилання, не формуючи весь файл .apk або .ipa	Складно	Легко

Для мобільного застосунку немає необхідності інтеграції нативних модулів, також не критичний розмір застосунку. Враховуючи переваги expo, що полегшують і прискорюють розробку, обрано expo.

У мобільному застосунку для суддів розроблено 3 сторінки: LoginScreen, MatchList, MatchService.

Призначення і реалізація сторінки LoginScreen аналогічні до компоненту Login з клієнтської частини для організаторів.

Сторінка MatchList призначена для перегляду списку матчів. Отримання списку даних застосовується комбінацією запиту на отримання відповідних сутностей з сервера (query matches) та підписки на зміни цих сутностей (subscription matchesUpdated). Також є можливість перейти на обслуговування конкретного матчу, якщо для матчу виконуються наступні вимоги:

1) цей матч має обслуговувати поточний користувач;

2) визначено команди, що мають грати;

3) цей матч ще не закінчився.

Сторінка MatchService призначення для обслуговування матчу, а саме для введення ігрових подій, використовуючи мутацію на додавання відповідної сутності (mutation addGameEvent).

Сторінка MatchService має два стани:

- Матч ще не почався, при якому є можливість почати обслуговувати матч.
- Матч зараз обслуговується, при якому є можливість введення ігрових подій, а також закінчити обслуговувати матч. Загальна схема UI компонентів зображена на рис. 4.8.

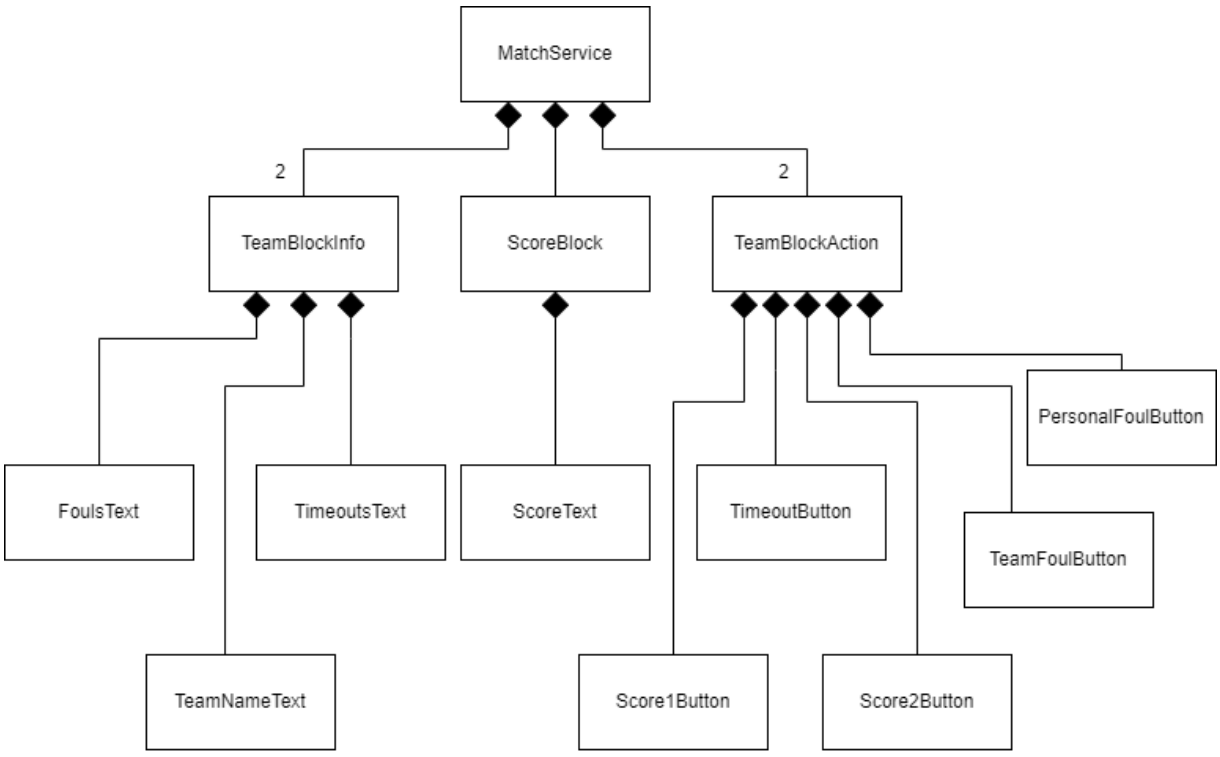


Рис. 4.8. Загальна схема UI компоненту MatchService в стані обслуговування матчу

Компонент TeamBlockInfo призначений для відображення кількості фолів (FoulsText), кількості таймаутів (TimeoutsText), назви команди (TeamNameText). Компонент MatchService включає по екземпляру

компонента TeamBlockInfo для кожної команди.

Компонент TeamBlockAction призначений для відображення кнопок для введення ігрових моментів, а саме: забито одне очко (Score1Button), забито два очки (Score2Button), взято таймаут (TimeoutButton), командний фол (TeamFoulButton), персональний фол (PersonalFoulButton). Компонент MatchService включає по екземпляру компонента TeamBlockAction для кожної команди.

Компонент ScoreBlock відповідає за відображення поточного рахунку матча (ScoreText). Компонент MatchService включає один екземпляр компонента ScoreBlock.

Користувацький інтерфейс зображений на рис. 4.9.

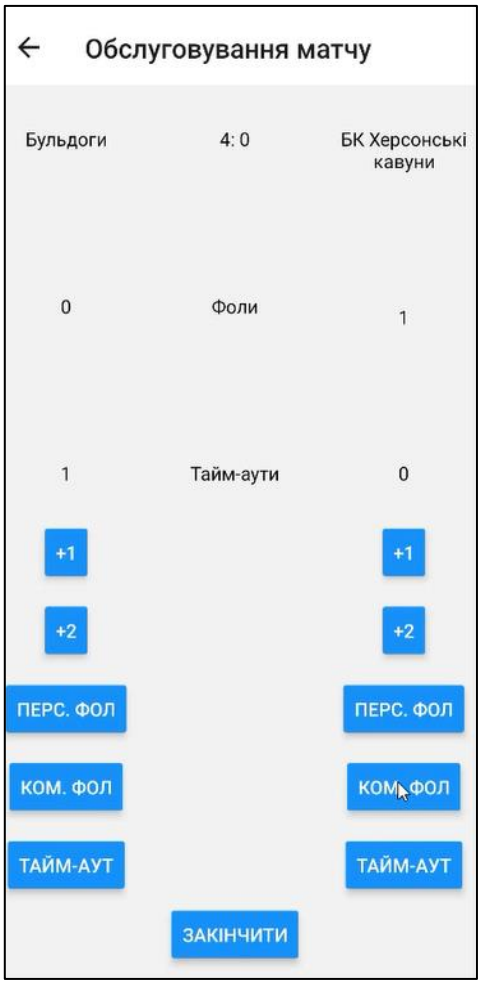


Рис. 4.9. Користувацький інтерфейс компоненту MatchService в стані обслуговування матчу

4.4. Тестування програмної системи

4.4.1. *Автоматизоване тестування*

За допомогою стандартної бібліотеки unittest на стороні серверу написано декілька автоматизованих тестів, що перевіряють роботу модуля авторизації та сховища даних.

Тест 1. Функція перевірки пароля

Підготовка: в системі зареєстрований користувач з login, password, access.

Вхідні дані: масив з вірними та не вірними парами login та password.

Лістинг 4.2. Автоматизований тест функції перевірки пароля

```
    def test_check_password(self):
        insert_or_update_user("admin", "!admin", "organizer")

        data = [
            {"login": "admin", "password": "!admin", "result": True},
            {"login": "admin", "password": "admin", "result": False},
            {"login": "admin2", "password": "!admin", "result": False},
            {"login": "admin", "password": "organizer", "result": False},
            {"login": "organizer", "password": "admin", "result": False},
            {"login": "organizer", "password": "!admin", "result": False},
        ]
        for obj in data:
            self.assertEqual(check_password(obj["login"], obj["password"]),
obj["result"])
```

Тест 2. Функція перевірки генерації токену

Підготовка: в системі зареєстрований користувач з login, password, access.

Вхідні дані: масив з вірними та не вірними парами login та access.

Лістинг 4.3. Автоматизований тест функції генерації токену

```
    def test_get_token(self):
        insert_or_update_user("admin", "!admin", "organizer")
        insert_or_update_user("ivan", "!ivan", "judge")

        data = [
            {"user": "admin", "access": "organizer", "result": True},
            {"user": "admin", "access": "judge", "result": False},
            {"user": "admin", "access": "!organizer", "result": False},
            {"user": "ivan", "access": "organizer", "result": False},
            {"user": "ivan", "access": "judge", "result": True},
            {"user": "ivan", "access": "!organizer", "result": False},
```

```
        ]

        for obj in data:
            token = get_token(obj["user"])
            user = get_user_by_token(token)
            self.assertEqual(user["access"] == obj["access"], obj["result"])
```

4.4.2. *Ручне тестування*

Тест 1. Перевірка авторизації користувачів на вебзастосунку для організаторів та на мобільному застосунку для суддів.

Підготовка:

- У системі зареєстровано користувачів, що наведені в табл. 4.6.

Таблиця 4.6

Зареєстровані користувачі

Логін	Пароль	Доступ
admin	!admin	організатор
ivan	!ivan	суддя

Кроки:

1. Ввести значення “login” та “password” у поля авторизації.
2. Натиснути “Увійти”.

Таблиця 4.7

Вхідні дані та очікувані результати на клієнтах для тесту 1

Вхідні дані	Очікуваний результат на клієнті для організаторів	Очікуваний результат на клієнті для суддів
`login: “admin”` `password: “!admin”`	Перехід на основну сторінку	Помилка
`login: “ivan”` `password: “!ivan”`	Помилка	Перехід на основну сторінку
`login: “ivan”` `password: “admin”`	Помилка	Помилка

Продовження табл. 4.7

Вхідні дані	Очікуваний результат на клієнті для організаторів	Очікуваний результат на клієнті для суддів
`login: "admin"` `password: "ivan"`	Помилка	Помилка
`login: "!admin"` `password: "admin"`	Помилка	Помилка
`login: "!123"` `password: "123"`	Помилка	Помилка
`login: "админ"` `password: "!админ "`	Помилка	Помилка

Тест 2. Перевірка підписки клієнтів на генерацію розкладу

Підготовка:

- Відкритий додаток для суддів, користувач авторизований.
- Відкритий додаток для організаторів, користувач ввів необхідні дані для генерації розкладу.

Кроки:

1. У додатку для організаторів натиснути кнопку “Згенерувати розклад”.
2. Ввести дані:
   - часу початку турніру (06.08.2024);
   - максимального часу гри (17);
   - перерви між іграми на майданчику (3).
3. У додатку для організатора натиснути кнопку “Згенерувати розклад”.

Очікуваний результат:

1. У додатку для організаторів відкриється “Форма для генерації розкладу”.
2. Введені дані відображаються коректно.
3. Протягом 2 секунд обидва додатки відображатимуть згенерований розклад.

Тест 3. Перевірка підписки клієнтів на введення ігрових моментів матчу

Підготовка:

- Відкритий додаток для суддів, обслуговує матч.
- Відкритий додаток для організаторів, користувач знаходиться на сторінці розкладу.

Кроки:

1. Додати одне очко першій команді, за допомогою натискання на кнопку “+1”.
2. Додати два очки першій команді, за допомогою натискання на кнопку “+2”.
3. Зафіксувати командний фол першій команді, за допомогою натискання на кнопку “КОМ. ФОЛ”.
4. Зафіксувати персональний фол першій команді, за допомогою натискання на кнопку “ПЕРС. ФОЛ”.
5. Зафіксувати таймаут першій команді, за допомогою натискання на кнопку “ТАЙМ-АУТ”.
6. Кроки 1-5 для другої команди.
7. Завершити обслуговування матчу, за допомогою натискання на кнопку “ЗАКІНЧИТИ”.

Очікуваний результат:

1. У додатках для суддів та для організаторів змінився рахунок для першої команди на 1.
2. У додатках для суддів та для організаторів змінився рахунок для першої команди на 2.
3. У додатках для суддів та для організаторів змінилася кількість фолів для першої команди на 1.
4. У додатках для суддів та для організаторів змінилася кількість фолів для першої команди на 1.
5. У додатках для суддів та для організаторів змінилася кількість

таймаутів для першої команди на 1.

6. Очікуваний результат 1-5 для другої команди.
7. У додатку для суддів відкриється список матчів з оновленими як мінімум двома матчами (перший, той, який був обслуговуваний, другий, якщо є наступний раунд). У додатку для організаторів оновиться розклад матчів, відповідний матч буде відмічений як “Зіграний”.

5. ШЛЯХИ ПОКРАЩЕННЯ ПРОЄКТУ

5.1. Аналіз реалізованого рішення

Дана програмна система реалізувала поставлені завдання, а саме:

- автоматичне формування розкладу (Task 2.2);
- створення протоколів, шляхом введення ігрових моментів суддею (асистентом судді) (Task 2.4);
- забезпечення відкритого API для інформування щодо змін в розкладі, оновлення поточної ситуації матчу в режимі реального часу (Task 2.3, Task 2.5).

Завдання реєстрації команд, суддів, майданчиків (Task 2.1) покладено на організаторів, що є тимчасовим рішенням. В загальній системі завдання реєстрації покладено на користувачів-гравців.

Система розрахована на обслуговування одного турніру. Для обслуговування декількох турнірів одночасно, є можливість запуску серверів системи на різних адресах. Немає можливості об’єднувати дані з декількох турнірів.

5.2. Можливості розширення системи

Розширення розроблених компонентів:

- Розширення сутності команда. Зараз сутність команди містить одне поле назви команди. Додати сутність гравця, який є членом команди. Додати персоналізацію подій в матчі: персональний фол та отримане очко з прив’язкою до конкретного гравця, що дає змогу бачити статистику щодо гравця, можливість автоматично визначати найбільш цінного гравця (MVP).
- На зараз програмна система немає можливостей для редагування даних, що могло бути корисним для користувачів.
- На зараз програмна система не розрахована на випадок, коли суддя випадково натисне якийсь ігровий момент, немає можливість

відміни попереднього введеної події.

- Для списку сутностей немає фільтрів та пагінації. Наприклад, було б дуже корисно для користувачів фільтрувати список матчів за параметрами:
  - за станом гри: зіграний, запланований, в процесі;
  - за командою;
  - за майданчиком;
  - за гравцем;
  - за суддею.
- Підтримка різних форматів турнірних сіток (на зараз є лише олімпійська система).
- Підтримка різних форматів відображення (у вигляді сітки, у вигляді списку).
- Підтримка відправки протоколів на задану адресу.
- Підтримка обслуговування декількох турнірів, категорій одночасно використовуючи спільне сховище даних.
- Підтримка декількох мов, наприклад, додати локалізацію англійської мови.
- Підтримка декількох тем, наприклад, світла та темна.

Створення нових компонентів системи:

- Інформування бажаючих користувачів про турніри, наприклад, шляхом відправлення повідомлення на пошту.
- Створення застосунків для гравців, що дає змогу змістити процес реєстрації з організаторів на гравців. Надає можливість персонально інформувати учасників, що їх матч скоро почнеться.
- Інтеграція системи з системою FIBA.
- Проведення голосування за найціннішого гравця турніру.
- Ведення відеотрансляції.
- Формування фотозвітів.

- Виведення поточної ситуації гри на екран.
- Розширення програмної системи до соціальної мережі баскетболістів, що може охопити і інших спортсменів схожої діяльності, наприклад, данкерів, фрістайлерів.
- Розширення програмної система можливістю перегляду баскетбольних майданчиків поблизу, влаштовувати зустрічі, місцеві турніри.

Співпраця з іншими користувачами:

- Встановлення реклами спонсорів.
- Надання інформації букмекерам.

ВИСНОВОК

Метою дипломного проєкту було створення програмної системи для обслуговування турнірів з баскетболу 3х3.

Проведено аналіз предметної області, опитано користувачів, сформульовано вимоги до програмної системи.

Проаналізовано сучасні протоколи взаємодії компонентів клієнт-серверної архітектури, технології розробки серверної частини, вебзастосунків та мобільних застосунків. Обрано Python для написання серверної частини, React для написання вебклієнта, React Native для написання мобільного застосунку, протокол GraphQL для взаємодії між клієнтами та сервером, MongoDB для сховища даних.

Розроблена система забезпечує:

- розмежування доступу на ролі організатора та судді;
- можливість додавання команд, майданчиків, акаунтів суддів;
- автоматичного формування розкладу;
- можливість введення ігрових моментів матчу;
- можливість перегляду протоколу матчів.

Розробка виконана у повному обсязі згідно з положеннями Технічного завдання. Проведено тестування відповідно до затвердженої програми та методики тестування. Вказано напрямки вдосконалення програмної системи.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Unified Modeling Language specification [Електронний ресурс] / — Режим доступу до ресурсу: https://www.omg.org/spec/UML (дата звернення 22.04.2023).
2. Леоненков А. В. Самоучитель UML 2.0. СПб : БХВ-Петербург, 2007.
3. FIBA.basketball [Електронний ресурс] — Режим доступу до ресурсу: https://www.fiba.basketball/ (дата звернення 22.04.2023)..
4. Федерація Баскетболу України [Електронний ресурс] — Режим доступу до ресурсу: https://fbu.ua/ (дата звернення 22.04.2023)..
5. Українська стритбольна ліга [Електронний ресурс] — Режим доступу до ресурсу: http://streetball.in.ua/ (дата звернення 22.04.2023)..
6. 2022 Official basketball rules [Електронний ресурс] // FIBA. – 2022. — Режим доступу до ресурсу: https://www.fiba.basketball/documents/official-basketball-rules/current.pdf (дата звернення 22.04.2023)..
7. Kamali A. gRPC, Restful API, GraphQL, Web Socket, TCP Sockets and UDP, WebTransport — Beyond modern client server communications [Електронний ресурс] / Amir Kamali. – 2020. — Режим доступу до ресурсу: https://tech-lead.medium.com/grpc-vs-restful-api-vs-graphql-web-socket-tcp-sockets-and-udp-beyond-client-server-43338eb02e37 (дата звернення 22.04.2023).
8. Hypertext Transfer Protocol – HTTP/1.1 [Електронний ресурс] / [R. Fielding, U. Irvine, J. Gettys та ін.] // Network Working Group. – 1999. — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc2616 (дата звернення 22.04.2023).
9. Swagger [Електронний ресурс] // SmartBear — Режим доступу до ресурсу: https://swagger.io/ (дата звернення 22.04.2023).
10. Fette I. The WebSocket Protocol [Електронний ресурс] / I. Fette, A. Melnikov. – 2011. — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення 22.04.2023).

11. GraphQL [Електронний ресурс] // GraphQL Fondation — Режим доступу до ресурсу: https://graphql.org/ (дата звернення 22.04.2023).
12. The Best 10 Backend Programming Languages [Електронний ресурс] — Режим доступу до ресурсу: https://blog.back4app.com/backend-programming-languages-list/ (дата звернення 22.04.2023).
13. Python [Електронний ресурс] — Режим доступу до ресурсу: https://www.python.org/ (дата звернення 22.04.2023).
14. Bhagat V. Pros and Cons of Python Programming Language [Електронний ресурс] / Varun Bhagat. – 2022. — Режим доступу до ресурсу: https://www.pixelcrayons.com/blog/python-pros-and-cons (дата звернення 22.04.2023).
15. MongoDB [Електронний ресурс] — Режим доступу до ресурсу: https://www.mongodb.com/ (дата звернення 22.04.2023).
16. Sakovich N. Top Most Popular Frontend Frameworks 2023 [Електронний ресурс] / Natalia Sakovich. – 2023. — Режим доступу до ресурсу: https://www.sam-solutions.com/blog/best-frontend-framework/ (дата звернення 22.04.2023).
17. React [Електронний ресурс] — Режим доступу до ресурсу: https://react.dev/ (дата звернення 22.04.2023).
18. Sakovich N. Cross-Platform Mobile Development: Five Best Frameworks [Електронний ресурс] / Sakovich. – 2023. — Режим доступу до ресурсу: https://www.sam-solutions.com/blog/cross-platform-mobile-development/ (дата звернення 22.04.2023).
19. React Native [Електронний ресурс] — Режим доступу до ресурсу: https://reactnative.dev/ (дата звернення 22.04.2023).
20. GraphQL – Passing an enum value directly to a mutation as an argument? [Електронний ресурс]. – 2019. — Режим доступу до ресурсу: https://stackoverflow.com/questions/47704615/graphql-passing-an-enum-value-directly-to-a-mutation-as-an-argument (дата звернення 22.04.2023).

21. Ariadne [Електронний ресурс] — Режим доступу до ресурсу: https://ariadnegraphql.org/ (дата звернення 22.04.2023).
22. JSON Web Tokens [Електронний ресурс] — Режим доступу до ресурсу: https://jwt.io/ (дата звернення 22.04.2023).
23. JSON Web Token (JWT) [Електронний ресурс] // Internet Engineering Task Force (IETF). – 2015. — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення 22.04.2023).
24. Starlette [Електронний ресурс] — Режим доступу до ресурсу: https://www.starlette.io/ (дата звернення 22.04.2023).
25. Turner S. Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms [Електронний ресурс] / S. Turner, L. Chen. – 2011. — Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc6151 (дата звернення 22.04.2023).
26. Crypto++ 5.5 Benchmarks [Електронний ресурс] — Режим доступу до ресурсу: https://web.archive.org/web/20081015103105/http://www.cryptopp.com/benchmarks.html (дата звернення 22.04.2023).
27. PyMongo 4.3.3 Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://pymongo.readthedocs.io/en/stable/ (дата звернення 22.04.2023).
28. Uvicorn [Електронний ресурс] — Режим доступу до ресурсу: https://www.uvicorn.org/ (дата звернення 22.04.2023).
29. Broadcaster [Електронний ресурс] — Режим доступу до ресурсу: https://pypi.org/project/broadcaster/ (дата звернення 22.04.2023).
30. Apollo [Електронний ресурс] — Режим доступу до ресурсу: https://www.apollographql.com/ (дата звернення 22.04.2023).
31. Two-way Binding Helpers [Електронний ресурс] — Режим доступу до ресурсу: https://legacy.reactjs.org/docs/two-way-binding-helpers.html (дата звернення 22.04.2023).

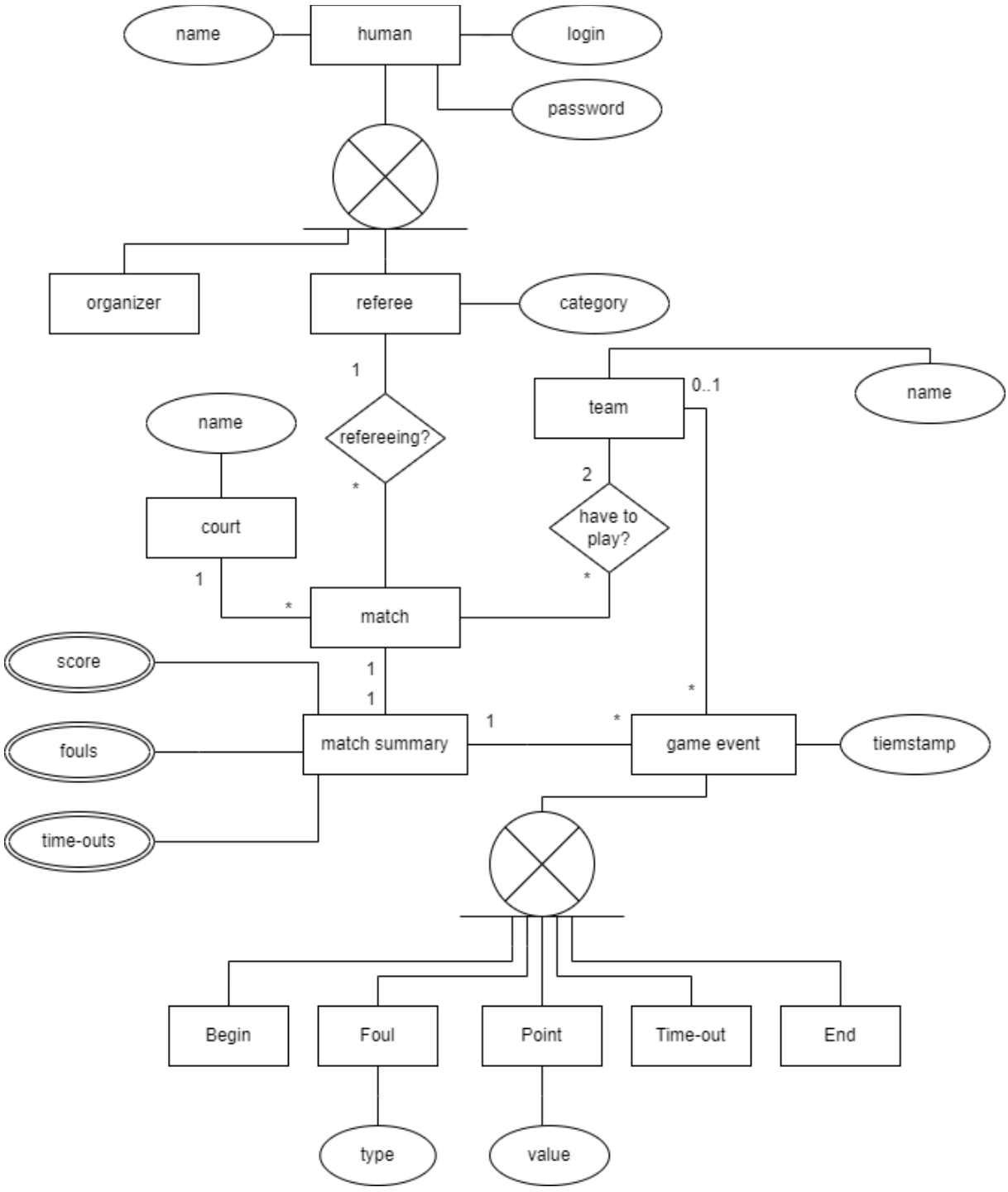
32. Vite [Електронний ресурс] — Режим доступу до ресурсу: https://vitejs.dev/ (дата звернення 22.04.2023).

ДОДАТКИ

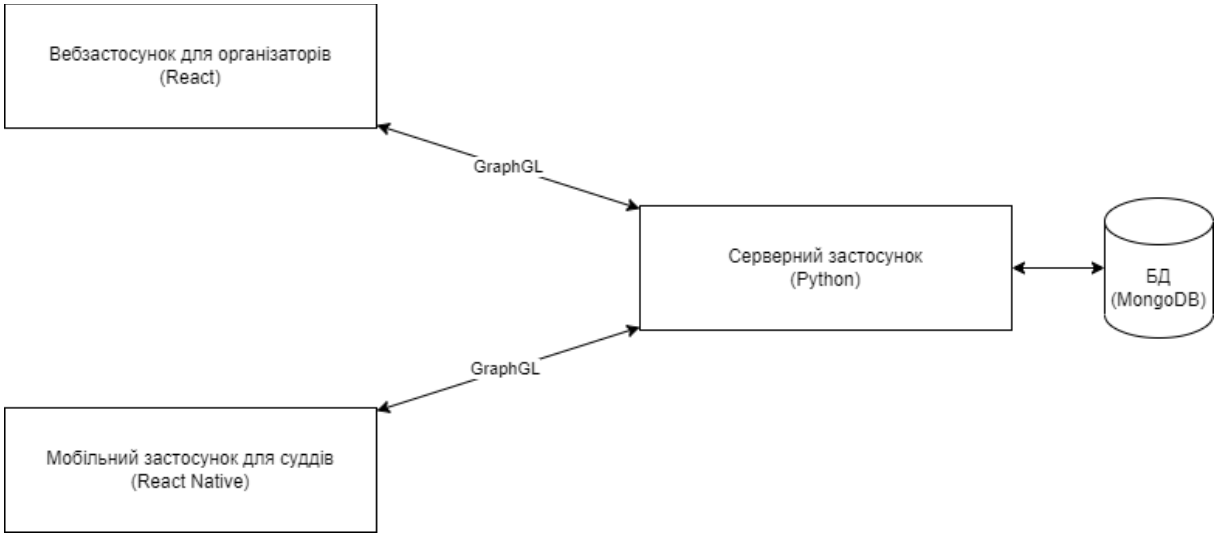
Додаток 1

Копії графічних матеріалів

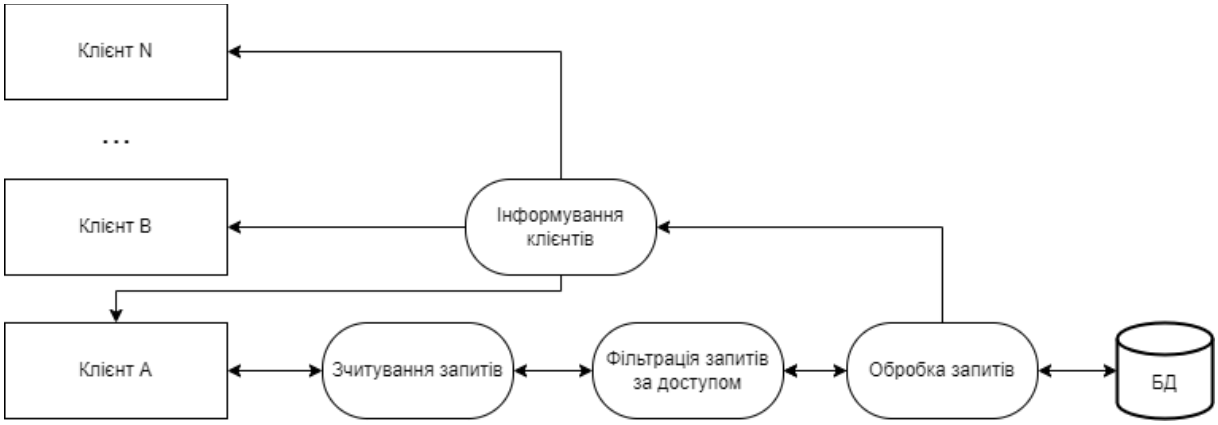
ДП.045490-06-99
Програмна система для обслуговування турнірів з баскетболу 3х3.
Функціональність системи.
Діаграма використання системи



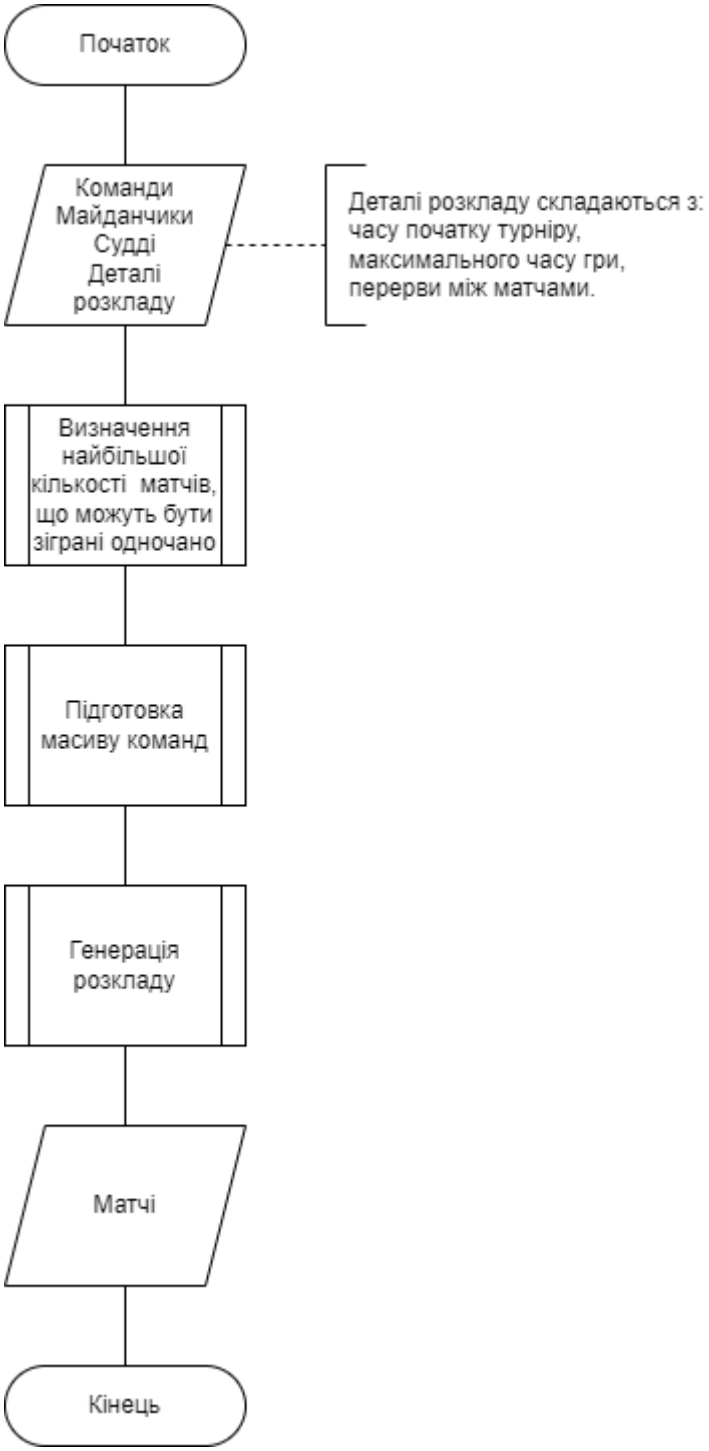
ДП.045490-07-99
Програмна система для обслуговування турнірів з баскетболу 3х3.
Функціональність системи.
Модель «сутність-зв’язок»



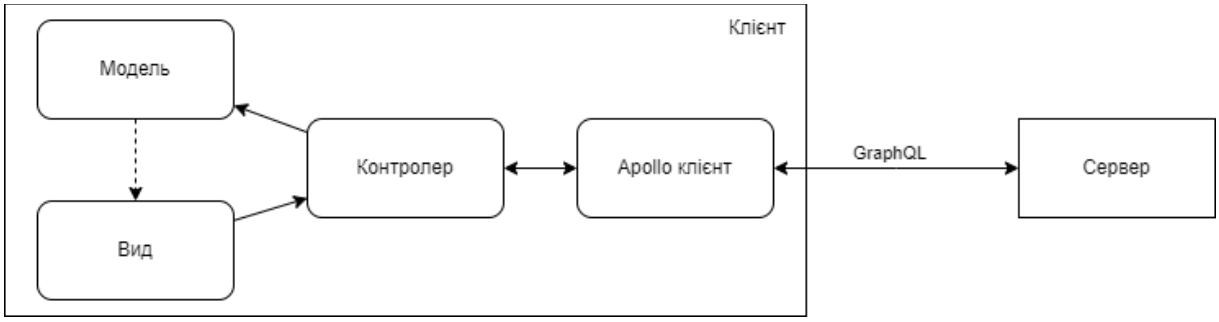
Програмна система для обслуговування турнірів з баскетболу 3х3.
Загальна архітектура системи.
Остапенко І. П., група КП-92



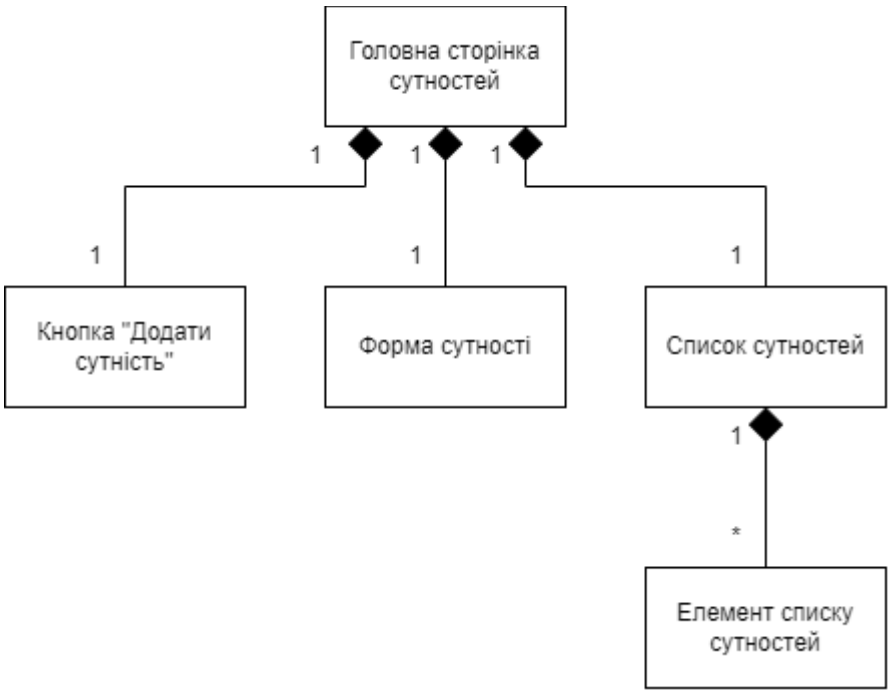
Програмна система для обслуговування турнірів з баскетболу 3x3.
Діаграма потоків серверу.
Остапенко І. П., група КП-92



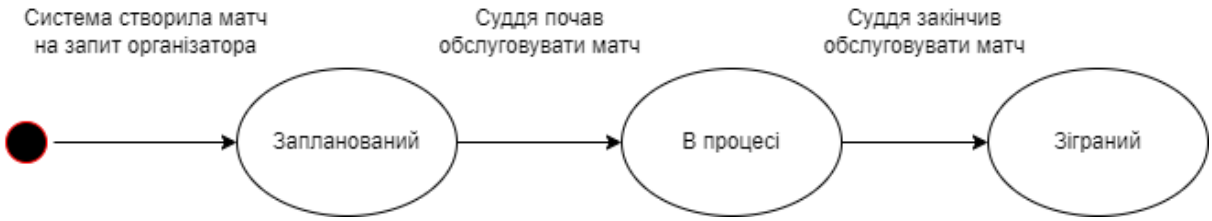
ДП.045490-09-99
Програмна система для обслуговування турнірів з баскетболу 3х3.
Алгоритм створення розкладу.
Блок-схема.
Остапенко І. П., група КП-92



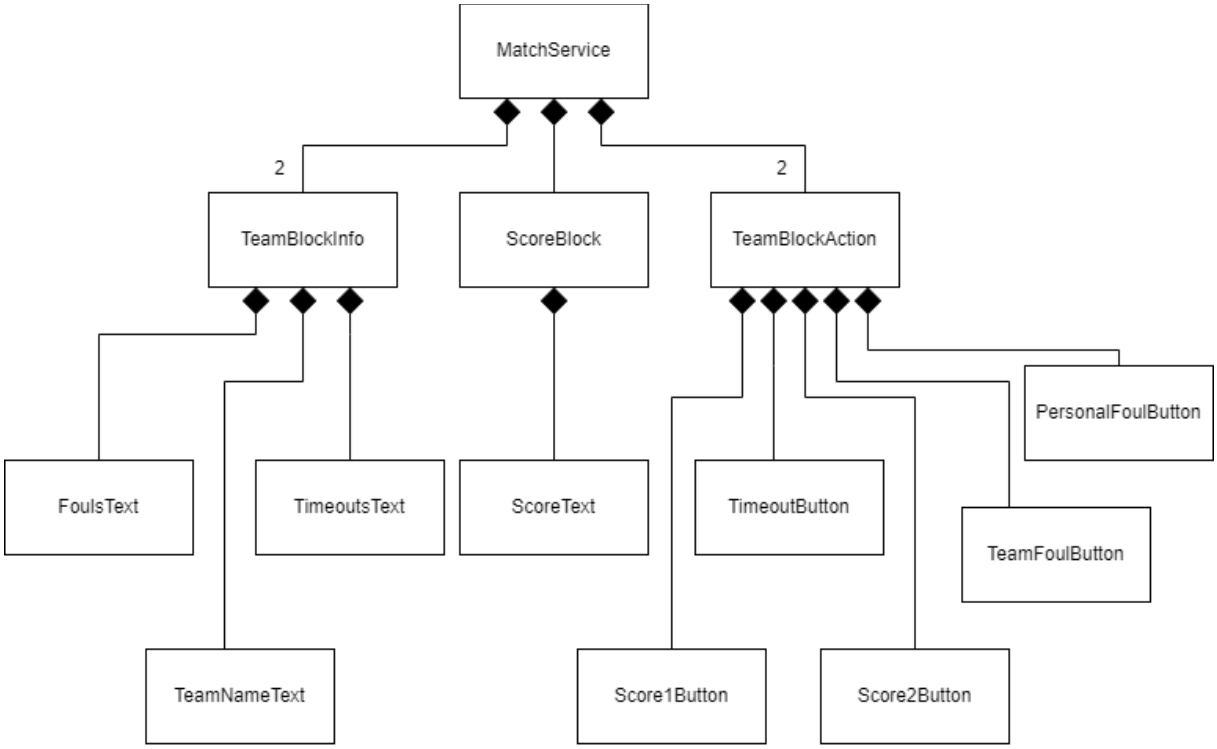
Програмна система для обслуговування турнірів з баскетболу 3х3.
Загальна структура клієнтів системи.
Остапенко І. П., група КП-92



Програмна система для обслуговування турнірів з баскетболу 3x3.
Структура клієнтської частини вебзастосунку для організаторів. Загальна схема UI компонентів сторінок Teams, Pools, Judges.
Остапенко І. П., група КП-92



Програмна система для обслуговування турнірів з баскетболу 3х3.
Діаграма станів матчу.
Остапенко І. П., група КП-92



Програмна система для обслуговування турнірів з баскетболу 3х3.
Структура клієнтської частини мобільного застосунку для суддів. Загальна схема UI компонентів сторінки MatchService в стані “матч зараз обслуговується”.
Остапенко І. П., група КП-92

Додаток 2

Лістинг програм

Лістинг програми серверної частини

main.py

```
from ariadne import make_executable_schema, format_error
from ariadne.asgi import GraphQL
from ariadne.asgi.handlers import GraphQLTransportWSHandler
from graphql import GraphQLError
from starlette.applications import Starlette
from starlette.middleware import Middleware
from starlette.middleware.cors import CORSMiddleware

from api.app import type_defs, types_objects
from api.authorization import get_context_value
from api.subscriptions import broadcast

schema = make_executable_schema([type_defs], types_objects)


def my_format_error(error: GraphQLError, debug: bool = False) -> dict:
    if debug:
        return format_error(error, debug)

    formatted = error.formatted
    del formatted["locations"]
    return formatted


graphql = GraphQL(
    schema=schema,
    error_formatter=my_format_error,
    context_value=get_context_value,
    websocket_handler=GraphQLTransportWSHandler(),
)

middleware = [
    Middleware(CORSMiddleware, allow_headers=["*"], allow_origins=['*'],
allow_methods=("GET", "POST", "OPTIONS"))
]

app = Starlette(
    debug=True,
    on_startup=[broadcast.connect],
    on_shutdown=[broadcast.disconnect],
    middleware=middleware
)

app.mount("/graphql/", graphql)
app.add_websocket_route("/graphql/", graphql)
```

storage.py

```
import json
import math

from db import db

from pymongo import ReturnDocument
from inspect import currentframe, getframeinfo

import random

from api.broadcasters import broadcast

class Storage:

    def __init__(self):
        self.db = db
```

```
        self.__sequence_document = self.db['sequences']
        self.__players = self.db['players']
        self.__teams = self.db['teams']
        self.__matches = self.db['matches']
        self.__schedule = self.db['schedule']
        self.__pools = self.db['pools']

    def __generate_next_id(self, field):
        if self.__sequence_document.count_documents({"field": field}) == 0:
            self.__sequence_document.insert_one({"field": field, "sequence_value":
0})
        sequence_object = self.__sequence_document.find_one_and_update(
            {"field": field},
            {"$inc": {"sequence_value": 1}},
            return_document=ReturnDocument.AFTER
        )
        return sequence_object["sequence_value"]

    def __add_entity(self, table, data):
        if "PK" in data:
            result = table.find_one({"PK": data["PK"]})
            if result:
                return False, result
        data["_id"] = self.__generate_next_id(table.name)
        result = table.insert_one(data)
        if result:
            return True, result.inserted_id
        print(getframeinfo(currentframe()))
        return False, None

    def __get_entity_by_id(self, table, entity_id):
        if entity_id is None:
            return None
        if entity_id == -1:
            return None
        return table.find_one({"_id": entity_id})

    def __get_all_entities(self, table):
        return list(table.find({}))

    def get_all_judges(self):
        return list(self.db.users.find({"access": "judge"}, {"_id": 0}))

    def get_all_players(self):
        return self.__get_all_entities(self.__players)

    def add_player(self, data):
        status, player = self.__add_entity(self.__players, data)
        if status:
            return player
        return None

    def get_all_teams(self):
        return self.__get_all_entities(self.__teams)

    def get_team_by_id(self, _id):
        if _id == -2:
            return {
                "name": "",
                "short_name": "",
                "_id": -2
            }
        return self.__get_entity_by_id(self.__teams, _id)

    def add_team(self, data):
        status, team_id = self.__add_entity(self.__teams, data)
        if status:
```

```
            return self.get_team_by_id(team_id)
        return None

    def get_all_pools(self):
        return self.__get_all_entities(self.__pools)

    def get_pool_by_id(self, _id):
        return self.__get_entity_by_id(self.__pools, _id)

    def get_judge_by_login(self, login):
        return self.db.users.find_one({"login": login})

    def add_pool(self, data):
        status, pool_id = self.__add_entity(self.__pools, data)
        if status:
            return self.get_pool_by_id(pool_id)
        return None

    def get_match(self, match_id):
        return self.__get_entity_by_id(self.__matches, match_id)

    def get_match_by_id(self, _id):
        return self.__get_entity_by_id(self.__matches, _id)

    def is_tournament_start(self):
        return len(self.get_all_matches())

    def get_all_matches(self):
        return self.__get_all_entities(self.__matches)

    def get_matches(self, matches_ids):
        return list(self.db.matches.find({"_id": {"$in": matches_ids}}))

    def add_match(self, data):
        for team_id in [data["team1"], data["team2"]]:
            if team_id != -1 and team_id != -2:
                if not self.__get_entity_by_id(self.__teams, team_id):
                    return False, None
        return self.__add_entity(self.__matches, data)

    def get_schedule(self):
        return self.__get_all_entities(self.__schedule)

    # start time and pause duration in minutes
    def generate_play_off_schedule(self, teams_ids, start_time, pool_ids,
judges_ids, max_game_time=None,
                                   pause_duration=None):
        if max_game_time is None:
            max_game_time = 20
        if pause_duration is None:
            pause_duration = 5

        round_count = math.ceil(math.log2(len(teams_ids)))
        extended_teams_ids = teams_ids + [-2] * (2 ** round_count - len(teams_ids))
        random.shuffle(extended_teams_ids)
        matches = []
        new_start_time = start_time

        max_parallel_games = min(len(pool_ids), len(judges_ids))
        for i in range(len(extended_teams_ids) // 2):
            match = {
                "team1": extended_teams_ids[2 * i],
                "team_source_1": None,
                "team2": extended_teams_ids[2 * i + 1],
                "team_source_2": None,
                "planned_time": start_time + i // max_parallel_games *
(max_game_time + pause_duration),
```

```
                "pool": pool_ids[i % len(pool_ids)],
                "judge": judges_ids[i % len(judges_ids)],
                "round": f"1/{2 ** (round_count - 1)}"
            }
            new_start_time = match["planned_time"] + pause_duration + max_game_time
            result, match_id = self.add_match(match)
            if not result:
                print(getframeinfo(currentframe()))
                return False, None
            matches.append(match_id)

        for i in range(round_count - 1, 0, -1):
            matches_in_round = []
            start_time = new_start_time
            for j in range(len(matches) // 2):
                match = {
                    "team1": -1,
                    "team_source_1": matches[2 * j],
                    "team2": -1,
                    "team_source_2": matches[2 * j + 1],
                    "planned_time": start_time + j // max_parallel_games *
(max_game_time + pause_duration),
                    "pool": pool_ids[j % len(pool_ids)],
                    "judge": judges_ids[j % len(judges_ids)],
                    "round": f"1/{2 ** (i - 1)}"
                }
                new_start_time = match["planned_time"] + pause_duration +
max_game_time
                result, match_id = self.add_match(match)

                if not result:
                    print(getframeinfo(currentframe()))
                    return False, None

                matches_in_round.append(match_id)
            matches = matches_in_round

        return matches

    async def add_game_event(self, game_event):
        result, _id = self.__add_entity(self.db.game_events, game_event)
        if result:
            match_id = game_event["match"]
            updated_matches_ids = [match_id]
            if game_event["event_type"] == "END":
                summary = self.get_match_summary(match_id)
                winner_team = summary["winner_team"]
                match_in_next_round = self.db.matches.find_one({"team_source_1":
match_id})
                if match_in_next_round:
                    self.db.matches.update_one({"team_source_1": match_id},
{"$set": {"team1": winner_team}})
                    updated_matches_ids.append(match_in_next_round["_id"])
                match_in_next_round = self.db.matches.find_one({"team_source_2":
match_id})
                if match_in_next_round:
                    self.db.matches.update_one({"team_source_2": match_id},
{"$set": {"team2": winner_team}})
                    updated_matches_ids.append(match_in_next_round["_id"])
                # create_report(match_id)
            if len(updated_matches_ids) == 3:
                print("login error add_game_event")
            await broadcast.publish(channel="matches",
message=json.dumps(updated_matches_ids))
            return self.__get_entity_by_id(self.db.game_events, _id)
        return None
```

```
    def get_game_event_by_id(self, _id):
        return self.__get_entity_by_id(self.db.game_events, _id)

    def get_game_events_by_match_id(self, match_id):
        return list(self.db.game_events.find({"match": match_id}))

    def __events_pointers_filter(self, team_id, game_events):
        return list(filter(lambda event: "team" in event and event["event_type"] ==
"SCORE" and event["team"] == team_id, game_events))

    def get_scores(self, match, game_events):
        team1_events = self.__events_pointers_filter(match["team1"], game_events)
        team2_events = self.__events_pointers_filter(match["team2"], game_events)

        scores = []

        for i in range(1, 24):
            scores.append({
                "id": i,
                "A": -1,
                "B": -1,
            })
            for events, field in [[team1_events, "A"], [team2_events, "B"]]:
                if len(events):
                    scores[-1][field] = events[0]["timestamp"]
                    if events[0]["score"] == 2:
                        events[0]["score"] = 1
                    else:
                        events.pop(0)
        return scores
    def get_match_summary(self, match_id):
        match = self.get_match(match_id)
        game_events = self.get_game_events_by_match_id(match_id)
        if not game_events:
            return None
        answer = {
            "score_1": 0,
            "score_2": 0,
            "fouls_1": 0,
            "fouls_2": 0,
            "timeout_1": 0,
            "timeout_2": 0,
            "state": "STARTED",
            "started_time": 0,
            "timeout_times_1": [],
            "timeout_times_2": [],
            "fouls_times_1": [],
            "fouls_times_2": [],
            "scores": [],
        }

        for game_event in game_events:
            if game_event["event_type"] in ["START", "END"]:
                if game_event["event_type"] == "END":
                    answer["state"] = "ENDED"
                if game_event["event_type"] == "START":
                    answer["started_time"] = game_event["timestamp"]
                continue
            number_team = ""
            if game_event["team"] == match["team1"]:
                number_team = "1"
            if game_event["team"] == match["team2"]:
                number_team = "2"
            if not number_team:
                print("login error game event")
                continue
            if game_event["event_type"] == "SCORE":
```

```
                answer[f'score_{number_team}'] += game_event["score"]
            if game_event["event_type"] in ["TEAM_FOUL", "PERSONAL_FOUL"]:
                answer[f'fouls_{number_team}'] += 1
                answer[f'fouls_times_{number_team}'] += [game_event["timestamp"]]
            if game_event["event_type"] == "TIMEOUT":
                answer[f'timeout_{number_team}'] += 1
                answer[f'timeout_times_{number_team}'] += [game_event["timestamp"]]

        if answer["state"] == "ENDED":
            answer["winner_team"] = match["team1"] if answer['score_1'] >
answer['score_2'] else match["team2"]
        answer["scores"] = self.get_scores(match, game_events)
        return answer


if __name__ == "__main__":
    st = Storage()
    for team in ["team1", "team2", "team3", "team4"]:
        print(st.add_team({"PK": team, "name": team}))
    for pool in ["A", "B", "C"]:
        print(st.add_pool({"PK": pool, "name": pool}))
```

app.py

```
from ariadne import load_schema_from_path

from api.match import match
from api.queries import queries
from api.mutations import mutation
from api.subscriptions import subscription


type_defs = load_schema_from_path("api/schema.graphql")
types_objects = [queries, mutation, match, subscription]
```

authorization.py

```
from datetime import datetime

import jwt
import hashlib

import unittest

from api.config import JWT_SECRET_KEY, JWT_ALGORITHM
from db import db
from storage import Storage


def get_user_by_token(jwt_token):
    result = jwt.decode(jwt_token, JWT_SECRET_KEY, algorithms=[JWT_ALGORITHM])
    login = result.get("payload").get("login")
    if not login:
        return {}
    return db.users.find_one({"login": login})


def get_token(login):
    return jwt.encode({"payload": {"login": login, "exp":
datetime.now().timestamp()}}, JWT_SECRET_KEY,
                      algorithm=JWT_ALGORITHM)


def get_role(login):
    user = db.users.find_one({"login": login})
    if user:
```

```
        return user["access"]
    return ""


def check_password(login, password):
    user = db.users.find_one({"login": login})
    if user:
        return user["password_hash"] == hashlib.sha256(password.encode('utf-
8')).hexdigest()
    return False


def insert_or_update_user(login, password, access, data=None):
    set_dict = {
        "password_hash": hashlib.sha256(password.encode('utf-8')).hexdigest(),
        "access": access,
    }
    if data:
        set_dict.update(data)

    db.users.update_one(
        {"login": login},
        {"$set": set_dict},
        upsert=True
    )
    return db.users.find_one({"login": login})


def get_context_value(request):
    auth = request.headers.get("Authorization")
    user = {}
    if auth:
        user = get_user_by_token(auth.split(" ")[1])
    return {'request': request, "user": user}


def level_required_required(info, accesses):
    user = info.context["user"]
    if user:
        if user["access"] in accesses:
            return True
    else:
        print(info)

    raise Exception("Permission Error")


class TestStringMethods(unittest.TestCase):

    def test_check_password(self):
        insert_or_update_user("admin", "!admin", "organizer")

        data = [
            {"login": "admin", "password": "!admin", "result": True},
            {"login": "admin", "password": "admin", "result": False},
            {"login": "admin2", "password": "!admin", "result": False},
            {"login": "admin", "password": "organizer", "result": False},
            {"login": "organizer", "password": "admin", "result": False},
            {"login": "organizer", "password": "!admin", "result": False},
        ]
        for obj in data:
            self.assertEqual(check_password(obj["login"], obj["password"]),
obj["result"])

    def test_get_token(self):
        insert_or_update_user("admin", "!admin", "organizer")
        insert_or_update_user("ivan", "!ivan", "judge")
```

```
        data = [
            {"user": "admin", "access": "organizer", "result": True},
            {"user": "admin", "access": "judge", "result": False},
            {"user": "admin", "access": "!organizer", "result": False},
            {"user": "ivan", "access": "organizer", "result": False},
            {"user": "ivan", "access": "judge", "result": True},
            {"user": "ivan", "access": "!organizer", "result": False},
        ]

        for obj in data:
            token = get_token(obj["user"])
            user = get_user_by_token(token)
            self.assertEqual(user["access"] == obj["access"], obj["result"])


if __name__ == "__main__":
    unittest.main()
```

broadcaster.py

```
from broadcaster import Broadcast

broadcast = Broadcast("memory://")
```

match.py

```
from ariadne import ObjectType
from storage import Storage

st = Storage()

match = ObjectType("Match")


@match.field("team_1")
def resolve_team_1(obj, *_):
    return st.get_team_by_id(obj["team1"])


@match.field("team_2")
def resolve_team_2(obj, *_):
    return st.get_team_by_id(obj["team2"])


@match.field("team_source_1")
def resolve_team_source_1(obj, *_):
    return st.get_match_by_id(obj["team_source_1"])


@match.field("team_source_2")
def resolve_team_source_2(obj, *_):
    return st.get_match_by_id(obj["team_source_2"])


@match.field("pool")
def resolve_pool(obj, *_):
    return st.get_pool_by_id(obj["pool"])


@match.field("judge")
def resolve_judge(obj, *_):
    return st.get_judge_by_login(obj["judge"])


@match.field("events")
```

```
def resolve_events(obj, *_):
    return st.get_game_events_by_match_id(obj["_id"])


@match.field("summary")
def resolve_summary(obj, *_):
    summary = st.get_match_summary(obj["_id"])
    return summary
```

mutations.py

```
import json

from ariadne import MutationType
from api.authorization import get_token, get_role, check_password,
level_required_required, insert_or_update_user
from storage import Storage

from api.subscriptions import broadcast

mutation = MutationType()

st = Storage()


@mutation.field("addTeam")
async def add_team(_, info, team_input):
    if st.get_all_matches():
        raise Exception("tournament is started")
    level_required_required(info, "organizer")
    team = team_input.copy()
    team["PK"] = team["name"]
    team = st.add_team(team)
    if team is not None:
        await broadcast.publish(channel="teams", message=json.dumps([team]))
        return team
    raise Exception("team can't to be added")


@mutation.field("addPool")
async def add_pool(_, info, pool_input):
    if st.get_all_matches():
        raise Exception("tournament is started")
    level_required_required(info, "organizer")
    pool = pool_input.copy()
    pool["PK"] = pool["name"]
    pool = st.add_pool(pool)
    if pool is not None:
        await broadcast.publish(channel="pools", message=json.dumps([pool]))
        return pool
    raise Exception("pool can't to be added")


@mutation.field("generateSchedule")
async def generate_schedule(_, info, start_time, game_duration, game_pause):
    level_required_required(info, "organizer")
    pools_ids = [pool["_id"] for pool in st.get_all_pools()]
    teams_ids = [team["_id"] for team in st.get_all_teams()]
    judges_ids = [judges["login"] for judges in st.get_all_judges()]

    st.generate_play_off_schedule(teams_ids, int(start_time) // 60 // 1000,
                                  pools_ids,
                                  judges_ids,
                                  max_game_time=game_duration,
                                  pause_duration=game_pause)
    matches_ids = [match["_id"] for match in st.get_all_matches()]
```

```
    await broadcast.publish(channel="matches", message=json.dumps(matches_ids))
    await broadcast.publish(channel="tournament",
message=json.dumps(len(st.get_all_matches())))
    return st.get_all_matches()


@mutation.field("tokenAuth")
def token_auth(_, info, username, password):
    if not check_password(username, password):
        return {"token": "", "role": ""}
    return {"token": get_token(username), "role": get_role(username)}


@mutation.field("addJudge")
async def add_judge(_, info, judge_input):
    level_required_required(info, "organizer")
    user = insert_or_update_user(judge_input["login"], judge_input["password"],
"judge",
                                 {
                                     "name": judge_input["name"],
                                     "category": judge_input["category"],
                                 })

    del user["_id"]
    await broadcast.publish(channel="judges", message=json.dumps([user]))
    return user


@mutation.field("addGameEvent")
async def add_game_event(_, info, event_input):
    level_required_required(info, "judge")
    answer = await st.add_game_event(event_input)
    return answer
```

queries.py

```
from ariadne import QueryType

from storage import Storage

queries = QueryType()

st = Storage()


@queries.field("teams")
def resolve_teams(obj, info):
    return st.get_all_teams()


@queries.field("pools")
def resolve_pools(obj, info):
    return st.get_all_pools()


@queries.field("matches")
def resolve_matches(obj, info):
    return st.get_all_matches()


@queries.field("judges")
def resolve_judges(obj, info):
    return st.get_all_judges()


@queries.field("isStarted")
```

```
def resolve_is_started(obj, info):
    return st.is_tournament_start()
```

schema.graphql

```
type Team{
    _id: ID!
    name: String!
    short_name: String!
}

input TeamInput{
    name: String!
    short_name: String!
}

type Pool{
    _id: ID!
    name: String!
}

input PoolInput{
    name: String!
}

type Score{
    id: ID!
    A: Float!
    B: Float!
}

type MatchSummary{
    score_1: Int!
    score_2: Int!
    fouls_1: Int!
    fouls_2: Int!
    timeout_1: Int!
    timeout_2: Int!
    state: String!
    winner_team: Int
    started_time: Float
    timeout_times_1: [Float]!
    timeout_times_2: [Float]!

    fouls_times_1: [Float]!
    fouls_times_2: [Float]!

    scores: [Score]!
}

type Match{
    _id: ID!
    team_1: Team
    team_source_1: Match
    team_2: Team
    team_source_2: Match
    planned_time: Float
    pool: Pool
    judge: Judge
    round: String
    events: [GameEvent]
    summary: MatchSummary
}

enum GameEventType{
    START,
```

```
    END,
    TIMEOUT,
    TEAM_FOUL,
    PERSONAL_FOUL,
    SCORE,
}

# GraphQL does support passing the enum directly as an argument;  GameEventType

type GameEvent{
    _id: ID!
    event_type: String!
    timestamp: Float!
    match: Match!
    team: Team
    score: Int
}


input GameEventInput{
    event_type: String!
    timestamp: Float!
    match: Int!
    team: Int
    score: Int
}

type TokenAuth{
    token: String!
    role: String!
}

input JudgeInput{
    login: String!
    name: String!
    password: String!
    category: String!
}

type Judge{
    login: String!
    name: String!
    category: String!
}


type Query {
    teams: [Team]
    pools: [Pool]
    matches: [Match]
    judges: [Judge]
    isStarted: Boolean!
}

type Mutation {
    addJudge(judge_input: JudgeInput!): Judge!
    addGameEvent(event_input: GameEventInput!): GameEvent!
    addTeam(team_input: TeamInput!): Team!
    addPool(pool_input: PoolInput!): Pool!
    generateSchedule(start_time: Float!, game_duration: Float!, game_pause:
Float!): [Match]

    tokenAuth(username: String!, password: String!): TokenAuth
}

type Subscription{
    tournamentStarted: Boolean!
```

```
    matchesUpdated: [Match]
    poolsUpdated: [Pool]
    teamsUpdated: [Team]
    judgesUpdated: [Judge]
}
```

subscriptions.py

```
import json
from api.broadcasters import broadcast
from ariadne import SubscriptionType

from storage import Storage

subscription = SubscriptionType()

st = Storage()


@subscription.source("matchesUpdated")
async def source_matchesUpdated(_, info):
    print("matches updated subscription.source")
    async with broadcast.subscribe(channel="matches") as subscriber:
        async for event in subscriber:
            match_ids = json.loads(event.message)
            matches = st.get_matches(match_ids)
            yield matches


@subscription.field("matchesUpdated")
def resolve_message(event, info):
    print("matches updated subscription.field", event, info)
    return event


@subscription.source("poolsUpdated")
async def source_matchesUpdated(_, info):
    print("pools updated subscription.source")
    async with broadcast.subscribe(channel="pools") as subscriber:
        async for event in subscriber:
            yield json.loads(event.message)


@subscription.field("poolsUpdated")
def resolve_message(event, info):
    print("pools updated subscription.field", event, info)
    return event


@subscription.source("teamsUpdated")
async def source_teamsUpdated(_, info):
    print("teams updated subscription.source")
    async with broadcast.subscribe(channel="teams") as subscriber:
        async for event in subscriber:
            yield json.loads(event.message)


@subscription.field("teamsUpdated")
def resolve_message(event, info):
    print("teams updated subscription.field", event, info)
    return event


@subscription.source("tournamentStarted")
async def source_tournamentStarted(_, info):
    print("tournamentStarted subscription.source")
```

```
    async with broadcast.subscribe(channel="tournament") as subscriber:
        async for event in subscriber:
            yield json.loads(event.message)


@subscription.field("tournamentStarted")
def resolve_message(event, info):
    print("tournamentStarted subscription.field", event, info)
    return event


@subscription.source("judgesUpdated")
async def source_judgesUpdated(_, info):
    print("judgesUpdated subscription.source")
    async with broadcast.subscribe(channel="judges") as subscriber:
        async for event in subscriber:
            yield json.loads(event.message)


@subscription.field("judgesUpdated")
def resolve_message(event, info):
    print("judgesUpdated subscription.field", event, info)
    return event
```

Лістинг клієнтської частини для організаторів

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Tournament Admin pages</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import {BrowserRouter} from "react-router-dom";
import  {ApolloClient,  ApolloProvider,  HttpLink,  InMemoryCache,  split}  from
"@apollo/client";
import {setContext} from "@apollo/client/link/context";
import {getMainDefinition} from "@apollo/client/utilities";
import {GraphQLWsLink} from "@apollo/client/link/subscriptions";
import { createClient } from "graphql-ws";

const authLink = setContext((_, { headers }) => {
    const token = localStorage.getItem('token');
    if(token){
        return {
            headers: {
                ...headers,
                Authorization: token ? `Bearer ${token}` : "",
                "Access-Control-Request-Headers": "Content-Type",
            }
        }
    }else{
        return headers
    }
});

const httpLink = new HttpLink({
    uri: `http://${window.location.hostname}:8000/graphql/`,
    credentials: 'same-origin'
});

const wsLink = new GraphQLWsLink(
    createClient({
        url: `ws://${window.location.hostname}:8000/graphql/`,
    })
);

const splitLink = split(
    ({ query }) => {
        const definition = getMainDefinition(query);
        return (
            definition.kind === "OperationDefinition" &&
            definition.operation === "subscription"
        );
    },
    wsLink,
    httpLink,
);
```

```
const client = new ApolloClient({
    link: authLink.concat(splitLink),
    cache: new InMemoryCache(),
});

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrictMode>
        <ApolloProvider client={client}>
            <BrowserRouter>
                <App />
            </BrowserRouter>
        </ApolloProvider>
    </React.StrictMode>
);
```

common.jsx

```
export function myMergeArrays(arr_old, arr_new, key = "_id"){
    const answer = [...arr_old]
    arr_new.forEach((element) => {
        const index = answer.findIndex((old_element) => old_element[key] ===
element[key])
        if(index < 0){
            answer.push(element)
        }else{
            answer[index] = element
        }
    })
    return answer
}
```

App.jsx

```
import React, {useEffect, useState} from "react";
import './styles/App.css';
import Navbar from "./components/UI/Navbar/Navbar";
import {AuthContext} from "./context";
import AppRoutes from "./routes/AppRoutes";
import {useQuery} from "@apollo/client";
import {IS_STARTED} from "./api/queries";
import {useIsTournamentStartedSubscription} from "./api/useSubscriptions";


function App() {

    const [token, setToken] = useState('');
    const [isStarted, setIsStarted] = useState(true);
    const { loading, error, data, refetch } = useQuery(IS_STARTED, {
        fetchPolicy: "no-cache"
    });
    useIsTournamentStartedSubscription(({data}) => {
        refetch()
    })

    useEffect(() => {
        localStorage.setItem('token', '')
    }, [])

    if(data){
        if(isStarted !== data["isStarted"]){
            setIsStarted(data["isStarted"])
        }
    }
```

```
    return (
            <AuthContext.Provider value={{
                token,
                setToken,
                isStarted,
            }}>
                <div>
                    {token && <Navbar/>}
                    <div style={{padding: 3}}>
                        <AppRoutes/>
                    </div>
                </div>
            </AuthContext.Provider>
    );
}

export default App;
```

App.css

```
.item {
    display: flex;
    padding: 15px;
    border: 2px solid teal;
    margin-top: 15px;
    justify-content: space-between;
    align-items: center;
}
.protocolDetails{
    border-left: 2px solid teal;
    border-right: 2px solid teal;
    border-bottom: 2px solid teal;
    padding: 15px;
}
.item-row {
    min-width: 100px;
}


table, th, td {
    font-weight: normal;
}
tr>:nth-child(1){
    text-align:right;
    padding-right: 3px;
}
tr>:nth-child(2){
    text-align:left;
}

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

#root {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.App {
    width: 800px;
}
```

```
.post {
    display: flex;
    padding: 15px;
    border: 2px solid teal;
    margin-top: 15px;
    justify-content: space-between;
    align-items: center;
}

.post__btns {
    display: flex;
}

.post-enter {
    transform: translateX(-350px);
}
.post-enter-active {
    transform: translateX(0px);
    transition: all 500ms ease-in;
}
.post-exit {
    opacity: 1;
}
.post-exit-active {
    transform: translateX(-350px);
    transition: all 500ms ease-in;
}

.page__wrapper {
    margin-top: 30px;
    display: flex;
}

.page {
    padding: 10px;
    border: 1px solid teal;
    cursor: pointer;
}

.page__current {
    border: 2px solid orange;
    font-weight: bold;
}

.navbar {
    height: 50px;
    width: 100vw;
    display: flex;
    align-items: center;
    padding: 0 15px;
    background: lightgray;
}

.navbar__links {
    margin-left: auto;
}
.centered {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

AppRountes.jsx

```
import React, {useContext} from 'react';
import {Navigate, Route, Routes} from "react-router-dom"
```

```
import {routes} from "./routes";
import {AuthContext} from "../context";
import Login from "../components/pages/Login";

const AppRoutes = () => {

    const {token, setToken} = useContext(AuthContext)

    if(token)
        return (
            <Routes>
                {routes.map(route =>
                    <Route exect
                           path={route.path}
                           element={route.component}
                           key={route.path}
                    />
                )}
                <Route path="*" element={<Navigate to="/about"/>}/>
            </Routes>
        );
    return (
        <Routes>
            <Route exect
                   path="/login"
                   element={<Login/>}
            />
            <Route path="*" element={<Navigate to="/login"/>}/>
        </Routes>
    )
};

export default AppRoutes;
```

routes.jsx

```
import About from "../components/pages/About";
import Teams from "../components/pages/teams/Teams";
import {Navigate} from "react-router-dom";
import Pools from "../components/pages/pools/Pools";
import Schedule from "../components/pages/schedule/Schedule";
import Judges from "../components/pages/judges/Judges";

export const routes = [
    {path: '/teams', component: <Teams/>},
    {path: '/pools', component: <Pools/>},
    {path: '/judges', component: <Judges/>},
    {path: '/schedule', component: <Schedule/>},
    {path: '/about', component: <About/>},
    {path: '*', component: <Navigate to="/about" replace={true} />},
]
```

fields.js

```
export const MATCH_FIELDS = `
        _id
        team_1 {
            name
            short_name
        }
        team_source_1 {
            _id
        }
        team_2{
            name
            short_name
```

```
        }
        team_source_2{
            _id
        }
        planned_time
        pool{
            _id
            name
        }
        judge{
            login
            name
        }
        round

        summary{
            score_1
            score_2
            fouls_1
            fouls_2
            timeout_1
            timeout_2
            state
            winner_team
            started_time
            timeout_times_1
            timeout_times_2
            fouls_times_1
            fouls_times_2
            scores{
                id
                A
                B
            }
        }
`

export const TEAM_FIELDS = `
    _id
    name
    short_name
`

export const POOL_FIELDS = `
    _id
    name
`
export const JUDGE_FIELDS = `
    login
    name
    category
`
```

queries.js

```
import {gql} from "@apollo/client";
import {JUDGE_FIELDS, MATCH_FIELDS, POOL_FIELDS, TEAM_FIELDS} from "./fields";

export const GET_TEAMS = gql(`
  query GetTeams {
    teams {
    ` + TEAM_FIELDS + `
    }
  }
`);
```

```
export const GET_POOLS = gql(`
  query GetPools {
    pools {
    ` + POOL_FIELDS + `
    }
  }
`);

export const GET_JUDGES = gql(`
  query GetJudges {
    judges {
    ` + JUDGE_FIELDS + `
    }
  }
`);


export const AUTHORIZATION = gql`
  mutation TokenAuth($username: String!, $password: String!) {
    tokenAuth(username: $username, password: $password) {
      token
      role
    }
  }
`;

export const ADD_TEAM = gql`
    mutation AddTeam($team_input: TeamInput!){
        addTeam(team_input: $team_input){
            _id
        }
    }
`;
export const ADD_POOL = gql`
    mutation AddPool($pool_input: PoolInput!){
        addPool(pool_input: $pool_input){
            _id
        }
    }
`;

export const ADD_JUDGE = gql`
    mutation AddJudge($judge_input: JudgeInput!){
        addJudge(judge_input: $judge_input){
            login
        }
    }
`;

export const GENERATE_SCHEDULE = gql`
    mutation   GenerateSchedule($start_time:   Float!,   $game_duration:   Float!,
$game_pause: Float!){
        generateSchedule(start_time:  $start_time,  game_duration:  $game_duration,
game_pause: $game_pause){
            _id
        }
    }
    `;
export const GET_SCHEDULE = gql(`
  query GetSchedule {
    matches {
        `+ MATCH_FIELDS + `
    }
  }
    `);

export const IS_STARTED = gql(`
```

```
query IsStarted{
    isStarted
}`)
```

useSubscriptions.js

```
import { useSubscription, gql } from "@apollo/client";
import {MATCH_FIELDS, POOL_FIELDS, TEAM_FIELDS, JUDGE_FIELDS} from "./fields";

const MATCHES_UPDATED_SUBSCRIPTION = gql(`
  subscription MatchesUpdated {
    matchesUpdated {
        ` + MATCH_FIELDS + `
    }
  }
`);

export function useMatchesUpdatedSubscription(onData) {
    return useSubscription(MATCHES_UPDATED_SUBSCRIPTION,
        {
            onData,
            shouldResubscribe: true,
            fetchPolicy: "no-cache",
        });
}


const TEAMS_UPDATED_SUBSCRIPTION = gql(`
  subscription TeamsUpdated {
    teamsUpdated {
        ` + TEAM_FIELDS + `
    }
  }
`);

export function useTeamsUpdatedSubscription(onData) {
    return useSubscription(TEAMS_UPDATED_SUBSCRIPTION,
        {
            onData,
            shouldResubscribe: true,
            fetchPolicy: "no-cache",
        });
}

const POOLS_UPDATED_SUBSCRIPTION = gql(`
  subscription PoolsUpdated {
    poolsUpdated {
        ` + POOL_FIELDS + `
    }
  }
`);

export function usePoolsUpdatedSubscription(onData) {
    return useSubscription(POOLS_UPDATED_SUBSCRIPTION,
        {
            onData,
            shouldResubscribe: true,
            fetchPolicy: "no-cache",
        });
}

const JUDGES_UPDATED_SUBSCRIPTION = gql(`
  subscription JudgesUpdated {
    judgesUpdated {
        ` + JUDGE_FIELDS + `
    }
```

```
  }
`);

export function useJudgesUpdatedSubscription(onData) {
    return useSubscription(JUDGES_UPDATED_SUBSCRIPTION,
        {
            onData,
            shouldResubscribe: true,
            fetchPolicy: "no-cache",
        });
}

const IS_START_SUBSCRIPTION = gql(`
subscription TournamentStarted{
    tournamentStarted
}
`)
export function useIsTournamentStartedSubscription(onData) {
    return useSubscription(IS_START_SUBSCRIPTION,
        {
            onData,
            shouldResubscribe: true,
            fetchPolicy: "no-cache",
        });
}
```

GenerateScheduleForm.jsx

```
import React, {useState} from 'react';
import MyInput from "../../UI/input/MyInput";
import MyButton from "../../UI/button/MyButton";

const GenerateScheduleForm = ({create}) => {

    const [options, setOptions] = useState({
        "start_time": Date.now(),
        "game_duration": 17,
        "game_pause": 3,
    })

    const generateSchedule = (e) => {
        e.preventDefault()
        create(options)
    }

    return (
        <div>
            <h3>Форма для генерації розкладу</h3>
            <br/>
            Час початку турніру:
            <MyInput
                type="datetime-local"
                value={options.start_time}
                onChange={e => setOptions({...options, start_time: e.target.value})}
            />
            Максимальний час гри:
            <MyInput
                type="number"
                value={options.game_duration}
                onChange={e      =>      setOptions({...options,      game_duration:
e.target.value})}
            />
            Перерва між іграми на майданчику:
            <MyInput
                type="number"
                value={options.game_pause}
```

```
                onChange={e => setOptions({...options, game_pause: e.target.value})}
            />
            <MyButton
                onClick={generateSchedule}
            >
                Згенерувати розклад
            </MyButton>
        </div>
    );
};

export default GenerateScheduleForm;
```

MatchItem.jsx

```
import React, {useState} from 'react';

import dateFormat from "dateformat";
import MyButton from "../../UI/button/MyButton";
import ProtocolDetails from "./ProtocolDetails";

function getTeamName(team){
    if(!team){
        return "?"
    }
    return team.name + " (" + team.short_name + ")"
}

const MainMatchInfo = ({match, style}) => {
    const getMatchState = () => {
        if(!match.summary){
            return "Заплановано"
        }
        if(match.summary.state === "STARTED"){
            return "В процесі"
        }
        if(match.summary.state === "ENDED"){
            return `Зіграний`
        }

    }

    return (
        <div style={{
            ...style,
            display: "flex",
            width:400,
            justifyContent: "space-between",
            alignItems: "center"}}
        >
            <div style={{width: 40}}>{dateFormat(new Date(match.planned_time * 60 *
1000), "dd/mm HH:MM")}</div>
            <div style={{width: 40, textAlign:"center"}}>{match.pool.name}</div>
            <div   style={{width:   140,   textAlign:"center"}}>{match.judge.login}
({match.judge.name})</div>
            <div style={{width: 150, textAlign:"center"}}>{getMatchState()}</div>
        </div>
    )
}
const MatchCurrentSummary = ({match, style, protocolVisible, setProtocolVisible}) =>
{

    const hardcoded_width = 70

    if(!match.summary){
        return (
```

```
            <div style={{
                ...style,
                display: "flex",
                width:hardcoded_width*5,
                justifyContent: "space-between",
                alignItems: "center"}}
            >
                <div style={{width: hardcoded_width}}></div>
                <div style={{width: hardcoded_width}}></div>
                <div style={{width: hardcoded_width}}></div>
            </div>
        )
    }

    return (
        <div style={{
            ...style,
            display: "flex",
            width:hardcoded_width*5,
            justifyContent: "space-between",
            alignItems: "center"}}
        >
            <div  style={{width:  hardcoded_width}}>S:  ({match.summary.score_1}  :
{match.summary.score_2})</div>
            <div  style={{width:  hardcoded_width}}>F:  ({match.summary.fouls_1}  :
{match.summary.fouls_2})</div>
            <div  style={{width:  hardcoded_width}}>T:  ({match.summary.timeout_1}  :
{match.summary.timeout_2})</div>
            <div style={{width: hardcoded_width}}>
                {protocolVisible       &&       <MyButton       onClick={()       =>
setProtocolVisible(false)}>Сховати</MyButton>}
                {!protocolVisible       &&       <MyButton       onClick={()       =>
setProtocolVisible(true)}>Показати</MyButton>}
            </div>
        </div>
    )
}

const MatchTeams = ({match, style}) => {
    return (
        <div style={{
            ...style,
            display: "flex",
            justifyContent: "space-between",
            alignItems: "center"
        }}>
            <strong   style={{width:   "50%",   textAlign:   "right",   paddingRight:
"9px"}}>{getTeamName(match.team_1)}</strong>
            vs
            <strong   style={{width:   "50%",   textAlign:   "left",   paddingLeft:
"9px"}}>{getTeamName(match.team_2)}</strong>
        </div>
    )
}

const MatchRound = ({match, style}) => {

    const getRoundName = () => {
        if(!match){
            return "?"
        }
        if(match.round === "1/1"){
            return "Фінал"
        }
        return match.round
    }
```

```
    return (
        <div style={{textAlign: "right", ...style}}>
            <strong>{getRoundName(match)}</strong>
        </div>
    )
}
const MatchItem = ({match}) => {

    const [protocolVisible, setProtocolVisible] = useState(false)

    return (
        <div>
            <div className="item">
                <MainMatchInfo style={{width: "15%"}} match={match}/>
                <MatchCurrentSummary     style={{width:     "40%"}}     match={match}
protocolVisible={protocolVisible}  setProtocolVisible={setProtocolVisible}/>
                <MatchTeams style={{width: "35%"}} match={match}/>
                <MatchRound style={{width: "10%"}} match={match}/>
            </div>
            <div style={{width: "1200px", margin: "auto"}}>
                {protocolVisible && <ProtocolDetails match={match}/>}
            </div>
        </div>
    );
};

export default MatchItem;
```

MatchList.jsx

```
import React from 'react';
import MatchItem from "./MatchItem";

const MatchList = ({matches}) => {
    if (!matches || !matches.length) {
        return (
            <h1 style={{textAlign: 'center'}}>
                Матчів не знайдено!
            </h1>
        )
    }

    return (
        <div>
            <h1 style={{textAlign: 'center'}}>
                Розклад
            </h1>
            {matches.map((match) =>
                <MatchItem key={match._id} match={match} />
            )}
        </div>
    );
};

export default MatchList;
```

ProtocolDetails.jsx

```
import React from "react";

const getTimeStr = (time) => {
    return time.toISOString().substring(14, 19);
}

const TeamName = ({match, team_number, style}) =>{
```

```
    if(team_number == 1){
        return (
            <div style={style}>Team A  {match.team_1.name}</div>
        )
    }
    return (
        <div style={style}>Team B  {match.team_2.name}</div>
    )
}

const ProtocolHeader = ({match}) => {
    return (
        <div style={{display: "flex"}}>
            <TeamName style={{width: "50%"}} match={match} team_number={1}/>
            <TeamName style={{width: "50%"}} match={match} team_number={2}/>
        </div>
    )
}

const MainInfo = ({match}) => {

    const start_time = new Date(match.summary.started_time)

    return (
        <div style={{
            display: "flex",
            border: "1px solid black",
            padding: "3px",
        }}>
            <div style={{width: "20%"}}>
                <table>
                    <tbody>
                    <tr>
                        <th>Game No.</th>
                        <th>{match._id}</th>
                    </tr>
                    </tbody>
                </table>
            </div>
            <div style={{width: "25%"}}>
                <table>
                    <tbody>
                    <tr>
                        <th>Date</th>
                        <th>{start_time.toLocaleDateString()}</th>
                    </tr>
                    </tbody>
                    <tbody>
                    <tr>
                        <th> </th>
                        <th> </th>
                    </tr>
                    </tbody>
                    <tbody>
                    <tr>
                        <th>Time</th>
                        <th>{start_time.toLocaleTimeString()}</th>
                    </tr>
                    </tbody>
                </table>
            </div>

            <div style={{width: "50%"}}>
                <table>
                    <tbody>
                    <tr>
                        <th>Referees    #1</th>
```

```
                        <th>{match.judge.name}</th>
                    </tr>
                    </tbody>
                    <tbody>
                    <tr>
                        <th>#2</th>
                        <th>-</th>
                    </tr>
                    </tbody>
                    <tbody>
                    <tr>
                        <th>Court</th>
                        <th>{match.pool.name}</th>
                    </tr>
                    </tbody>
                </table>
            </div>
        </div>
    )
}

const FoulBlock = ({time, index}) => {
    const getColorBlock = () => {
        if(time){
            return "#d3d3d3"
        }else{
            return "white"
        }
    }
    return (
        <div style={{padding: 3, margin: 1, border: "1px solid black", width: "50px",
background: getColorBlock()}}>
            {time && getTimeStr(time)}
            {!time && index}
        </div>
    )

}

const Score = ({score, start}) => {
    const getScoreTime = (value) => {
        if(value === -1){
            return ""
        }else{
            return getTimeStr(new Date(value-start))
        }
    }

    return (
        <div style={{display:"flex", padding: "3px", justifyContent:"space-between",
border: "1px solid black", marginBottom: -1}}>
            <div                        style={{width:                        "33%",
textAlign:"center"}}>{getScoreTime(score.A)}</div>
            <div  style={{width:  "33%",  textAlign:"center",  borderLeft:  "1px  solid
black", borderRight: "1px solid black"}}>{score.id}</div>
            <div                        style={{width:                        "33%",
textAlign:"center"}}>{getScoreTime(score.B)}</div>
        </div>
    )
}
const Scores = ({match}) => {
    const offset = 1
    return (
        <div style={{ display:"flex"}}>
            <div style={{width: "33%", padding: offset, marginTop: -offset}}>
                {match.summary.scores.slice(0,9).map((score)        =>        <Score
key={score.id} score={score} start={match.summary.started_time}/>)}
```

```
            </div>
            <div style={{width: "33%", padding: offset, marginTop: -offset}}>
                {match.summary.scores.slice(9,18).map((score)       =>       <Score
key={score.id} score={score} start={match.summary.started_time}/>)}
            </div>
            <div style={{width: "33%", padding: offset, marginTop: -offset}}>
                {match.summary.scores.slice(18,27).map((score)       =>       <Score
key={score.id} score={score} start={match.summary.started_time}/>)}

            </div>
        </div>
    )
}
const TeamBlock = ({match, team_number}) => {

    const prepareTimes = (times) => {
        return times.map((time) => new Date((time - match.summary.started_time)))
    }

    const prepareFouls = (fouls) => {
        if(fouls.length < 11){
            for(let i = fouls.length; i < 11; ++i){
                fouls.push(undefined)
            }
        }
        return fouls
    }
    const generateData = () => {
        if(team_number == 1){
            return {
                timeouts: prepareTimes(match.summary.timeout_times_1),
                fouls: prepareFouls(prepareTimes(match.summary.fouls_times_1)),
            }
        }else{
            return {
                timeouts: prepareTimes(match.summary.timeout_times_2),
                fouls: prepareFouls(prepareTimes(match.summary.fouls_times_2)),
            }
        }
    }

    const { timeouts, fouls, } = generateData()

    return (
        <div>
            <TeamName match={match} team_number={team_number}/>
            <div style={{display: "flex", border: "1px solid black"}}>
                <div  style={{width:  "30%",  textAlign:"center",    padding:  "6px",
borderRight: "1px solid black"}}>
                    <br/>
                    Time out
                    <br/>
                    {timeouts.map((timeout) => getTimeStr(timeout) + ";")}
                    <br/>
                </div>
                <div style={{width: "70%",  padding: "6px", textAlign:"center"}}>
                    <div>Team fouls</div>
                    <div style={{display: "flex", justifyContent: "center"}}>
                        {fouls.slice(0, 6).map((time, index) => {
                            return     <FoulBlock     key={index+1}     time={time}
index={index+1}/>
                        })}
                    </div>
                    <div style={{display: "flex", justifyContent: "center"}}>
                        {fouls.slice(6, 9).map((time, index) => {
                            return     <FoulBlock     key={index+7}     time={time}
index={index+7}/>
```

```
                    })}
                </div>
                <div style={{display: "flex", justifyContent: "center"}}>
                    <FoulBlock time={fouls[10]} index={10}/>
                </div>
                <div>
                    {fouls.slice(11).map((time) => {
                        return getTimeStr(time) + "; "
                    })}
                </div>
            </div>
        </div>
    </div>
    )
}

const ProtocolDetails = ({match}) => {
    return (
        <div className="protocolDetails">
            <ProtocolHeader match={match}/>
            <MainInfo match={match}/>
            <div style={{display: "flex"}}>
                <div style={{width: "50%", marginRight: "3px"}}>
                    <TeamBlock match={match} team_number={1}/>
                    <TeamBlock match={match} team_number={2}/>
                </div>
                <div style={{width: "50%", marginLeft: "3px"}}>
                    <div style={{textAlign: "center"}}>Running score</div>
                    <Scores match={match}/>
                </div>
            </div>
        </div>
    )
}
export default ProtocolDetails;
```

Schedule.jsx

```
import React, {useContext, useEffect, useState} from 'react';
import {useMutation, useQuery} from "@apollo/client";
import {GENERATE_SCHEDULE, GET_SCHEDULE} from "../../../api/queries";
import MyButton from "../../UI/button/MyButton";
import MatchList from "./MatchList";
import MyModal from "../../UI/MyModal/MyModal";
import GenerateScheduleForm from "./GenerateScheduleForm";
import {myMergeArrays} from "../../../common";
import {useMatchesUpdatedSubscription} from "../../../api/useSubscriptions";
import {AuthContext} from "../../../context";

const Schedule = () => {
    const {isStarted} = useContext(AuthContext)
    const getScheduleResult= useQuery(GET_SCHEDULE, {
        fetchPolicy: "no-cache"
    });
    const           [generateSchedule,           generateScheduleResult]           =
useMutation(GENERATE_SCHEDULE);

    const [modal, setModal] = useState(false);
    const [matches, setMathes] = useState([])
    const [updateMatches, setUpdateMatches] = useState([])

    useEffect(() => {
        if(getScheduleResult.data){
            if(!matches.length){
                setMathes(myMergeArrays(getScheduleResult.data["matches"],
updateMatches))
```

```
            }else{
                setMathes(myMergeArrays(matches, updateMatches))
            }
        }else{
            setMathes(updateMatches)
        }
    }, [getScheduleResult.data, updateMatches])
    useMatchesUpdatedSubscription((info) => {
        const updated_matches = info.data.data.matchesUpdated
        console.log(updated_matches)
        setUpdateMatches(updated_matches)
    })
    const onGenerateScheduleClick = (options) => {
        options.start_time = Date.parse(options.start_time)
        generateSchedule(
            {variables: options}
        ).then(r => {
            setModal(false)
            })
    }

    if(!(getScheduleResult && getScheduleResult.data)){
        return (
            <h3>Загрузка</h3>
        )
    }

    return (
        <div>
            {!isStarted?
                <div>
                    <MyButton     onClick={()     =>     setModal(true)}>Згенерувати
розклад</MyButton>
                        <MyModal visible={modal} setVisible={setModal}>
                            <GenerateScheduleForm create={onGenerateScheduleClick}/>
                        </MyModal>
                </div>
                :
                <MatchList matches={matches}/>
            }
        </div>
    );
};

export default Schedule;
```

About.jsx

```
import React from 'react';

const About = () => {
    return (
        <div>
            Це сторінка організатора турнірів з баскетболу 3х3
        </div>
    );
};

export default About;
```

Login.jsx

```
import React, {useContext, useState} from 'react';
import MyInput from "../UI/input/MyInput";
import MyButton from "../UI/button/MyButton";
import {AuthContext} from "../../context";
```

```
import {useMutation} from "@apollo/client";
import {AUTHORIZATION} from "../../api/queries";

const Login = () => {
    const {token, setToken} = useContext(AuthContext)


    const [credentials, setCredentials] = useState({username: '', password: ''})

    const [authoriazation, { data, loading, error }] = useMutation(AUTHORIZATION);

    const isAuthValid = () => {
        return data && data.tokenAuth && data.tokenAuth.token && data.tokenAuth.role
=== "organizer"
    }
    const login = e => {
        e.preventDefault()
        authoriazation({variables: credentials})
    }

    if(isAuthValid()){
        localStorage.setItem("token", data.tokenAuth.token)
        setToken(data.tokenAuth.token)
    }

    return (
        <div>
            <h1>Сторінка для входу в систему</h1>
            <form onSubmit={login}>
                <MyInput
                    value={credentials.username}
                    onChange={e    =>    setCredentials({...credentials,    username:
e.target.value})}
                    type="text"
                    placeholder="Логін"
                />
                <MyInput
                    value={credentials.password}
                    onChange={e    =>    setCredentials({...credentials,    password:
e.target.value})}
                    type="password"
                    placeholder="Пароль"
                />
                <div className="centered"><MyButton>Увійти</MyButton></div>
            </form>
            <div className="centered">
                {loading && <h6>Виконання запиту</h6>}
                {error && <h6>Помилка зв'язку</h6>}
                {!isAuthValid() && data && !error && !loading && <h6 style={{color:
"red"}}>Не вірний логін або пароль</h6>}
            </div>
        </div>
    );
};

export default Login;
```

Navbar.jsx

```
import React, {useContext} from 'react';
import {Link, useNavigate} from "react-router-dom";
import MyButton from "../button/MyButton";
import {AuthContext} from "../../../context";

const Navbar = () => {
    const navigate = useNavigate();
```

```
    const {token, setToken} = useContext(AuthContext);

    const logout = () => {
        setToken('');
        localStorage.removeItem('token')
    }

    const text_to_link = [
        {text: "Інформація", path: "/about"},
        {text: "Майданчики", path: "/pools"},
        {text: "Команди", path: "/teams"},
        {text: "Судді", path: "/judges"},
        {text: "Розклад", path: "/schedule"},
    ]

    return (
        <div className="navbar">
            <div className="navbar__links">
                {text_to_link.map(({text, path}) =>
                    <MyButton key={path} onClick={() => navigate(path)}>
                        {text}
                    </MyButton>
                )}
            </div>
            <MyButton onClick={logout}>
                Вийти
            </MyButton>
        </div>
    );
};

export default Navbar;
```

Лістинг клієнтської частини для суддів

App.js

```
import React from 'react';
import { RootProvider } from './RootProvider';



const App = () => {

  return (
    <RootProvider/>
  );
}

export default App;
```

globalStorage.js

```
import { createGlobalState } from "react-native-global-state-hooks";

export const useGlobalStorage = createGlobalState({
    token: "",
    login: ""
});
```

Navigation.jsx

```
import { createStackNavigator } from '@react-navigation/stack';
import MatchList from './screens/MatchList';
import MatchService from './screens/MatchService';
import { NavigationContainer } from '@react-navigation/native';
import LoginScreen from './screens/LoginScreen';

const Stack = createStackNavigator();

export const Navigation = () => {
  return (
    <NavigationContainer>
        <Stack.Navigator>
            <Stack.Screen   name="Main"   component={LoginScreen}   options={{title:
'Вхід'}} />
            <Stack.Screen  name="MatchList"  component={MatchList}  options={{title:
'Список матчів'}} />
            <Stack.Screen        name="MatchService"        component={MatchService}
options={{title: 'Обслуговування матчу'}} />
        </Stack.Navigator>
    </NavigationContainer>
  );
}
```

RootProvider.jsx

```
import {Navigation} from "./Navigation"
import { ApolloClient, InMemoryCache, HttpLink, split, concat, ApolloProvider } from
'@apollo/client';
import {setContext} from "@apollo/client/link/context";
import {GraphQLWsLink} from "@apollo/client/link/subscriptions";
import {getMainDefinition} from "@apollo/client/utilities";
import { createClient } from "graphql-ws";
import {url} from './config'
import AsyncStorage from '@react-native-async-storage/async-storage';
import { ApolloLink, Observable } from 'apollo-link';

const httpLink = new HttpLink({
```

```
  uri: `http://${url}`,
  credentials: 'include',
});

const request = async (operation) => {
  const token = await AsyncStorage.getItem('@token');
  operation.setContext({
    headers: {
      Authorization: token ? `Bearer ${token}` : ''
    }
  });
};

const requestLink = new ApolloLink((operation, forward) =>
  new Observable(observer => {
    let handle;
    Promise.resolve(operation)
      .then(oper => request(oper))
      .then(() => {
        handle = forward(operation).subscribe({
          next: observer.next.bind(observer),
          error: observer.error.bind(observer),
          complete: observer.complete.bind(observer),
        });
      })
      .catch(observer.error.bind(observer));

    return () => {
      if (handle) handle.unsubscribe();
    };
  })
);

const wsLink = new GraphQLWsLink(
  createClient({
      url: `ws://${url}`,
  })
);

const splitLink = split(
  ({ query }) => {
      const definition = getMainDefinition(query);
      return (
          definition.kind === "OperationDefinition" &&
          definition.operation === "subscription"
      );
  },
  wsLink,
  httpLink,
);

const client = new ApolloClient({
  link: concat(requestLink, splitLink),
  cache: new InMemoryCache(),

  defaultOptions: {
    watchQuery: {
      fetchPolicy: 'no-cache',
      errorPolicy: 'ignore',
    },
    query: {
      fetchPolicy: 'no-cache',
      errorPolicy: 'all',
    },
  }
});
```

```
export const RootProvider = () => {
    return (
      <ApolloProvider client={client}>
        < Navigation />
      </ApolloProvider>
    );
  }
```

LoginScreen.jsx

```
import { Text, StyleSheet, View, Button, TextInput, Alert} from 'react-native'
import React, { useEffect } from 'react'
import { AUTHORIZATION } from '../api/mutations';
import {useMutation} from "@apollo/client";
import { useNavigation } from '@react-navigation/native';
import { useGlobalStorage } from "../globalStorage"
import AsyncStorage from '@react-native-async-storage/async-storage';

export default function LoginScreen (){
    const [globalStorage, setGlobalStorage] = useGlobalStorage()
    const navigation = useNavigation();

    const [login, setLogin] = React.useState('');
    const [password, setPassword] = React.useState('');

    const    [authoriazation,    {    data,    loading,    error,    reset    }]    =
useMutation(AUTHORIZATION);

    React.useEffect(() => {
        reset()
    }, [])

    const onLogin = () => {
        authoriazation({variables: {
            "username": login,
            "password": password
        }}).then(async (some_data) => {
            if(some_data.data && some_data.data.tokenAuth.role === "judge"){
                setGlobalStorage({
                    ...globalStorage,
                    token: some_data.data.tokenAuth.token,
                    login: login,
                })
                await AsyncStorage.setItem('@token', some_data.data.tokenAuth.token)
                navigation.navigate('MatchList')
            }
        })
    }

    let error_text = ""
    if(error){
        error_text = error.message
    }
    if(data){
        if(data.tokenAuth.role !== "judge"){
            error_text = "Не вірний логін або пароль"
        }
    }

    if(error_text){
        Alert.alert(error_text)
        reset()
    }

    return (
        <View style={{justifyContent: 'center', alignItems: 'center', flex:1}}>
```

```
            <TextInput
                style={styles.input}
                onChange={(event) => setLogin(event.nativeEvent.text)}
                value={login}
                placeholder="Логін"
                ty
            />
            <TextInput
                style={styles.input}
                onChange={(event) => setPassword(event.nativeEvent.text)}
                value={password}
                secureTextEntry={true}
                placeholder="Пароль"
            />
            <Button
            onPress={onLogin}
            title="Увійти"
            />
        </View>
    )
}

const styles = StyleSheet.create({
    input: {
      height: 40,
      margin: 12,
      borderWidth: 1,
      width: 200,
      padding: 10
    },
  });
```

MatchItem.jsx

```
import { Text, StyleSheet, View } from 'react-native'
import React from 'react'
import dateFormat from "dateformat";
import { useGlobalStorage } from "../globalStorage"


export default MatchItem = ({match}) => {
    const [globalStorage, setGlobalStorage] = useGlobalStorage()

    const getScoreValue = (number) => {
      if(match.summary){
        if(number == 1){
          return match.summary.score_1
        }else{
          return match.summary.score_2
        }
      }else{
        return "-"
      }
    }


    const getRoundName = () =>{
      if(!match){
          return "?"
      }
      if(match.round === "1/1"){
          return "Фінал"
      }
      return match.round
    }
```

```
    const getTeamName = (team) => {
      if(!team){
          return <Text>?</Text>
      }
      if(team._id == -2){
        return "-"
      }
      if(match.summary && match.summary.state == "ENDED"){
        if(match.summary.winner_team == team._id){
          return <Text style={{fontWeight: 'bold'}}>{team.short_name}</Text>
        }
      }
      return <Text>{team.short_name}</Text>
    }

    return (
      <View style={styles.container}>
        <Text  style={{width:"12%"}}>{dateFormat(new Date(match.planned_time * 60 *
1000), "dd/mm HH:MM")}</Text>
        <Text style={{width:"5%"}}>{match.pool.name}</Text>
        <Text style={{width:"12%"}}>{match.judge.login}</Text>
        <Text                                                    style={{width:"18%",
textAlign:"right"}}>{getTeamName(match.team_1)}</Text>
        <Text style={{width:"4%"}}>vs</Text>
        <Text style={{width:"18%"}}>{getTeamName(match.team_2)}</Text>

        <Text style={{width:"3%", textAlign:"right"}}>{getScoreValue(1)}</Text>
        <Text style={{width:"1%"}}>:</Text>
        <Text style={{width:"3%", textAlign:"left"}}>{getScoreValue(2)}</Text>

        <Text style={{width:"12%", textAlign: "right"}}>{getRoundName(match)}</Text>
      </View>
    )
}

const styles = StyleSheet.create({
    container: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        borderBottomWidth: 2,
        width: '100%',
        height: 40
    }
})
```

MatchList.jsx

```
import React, {useState, useEffect} from 'react'
import MatchItem from './MatchItem';
import { useNavigation } from '@react-navigation/native';

import {useQuery} from "@apollo/client";
import {GET_SCHEDULE} from '../api/queries'

import { useGlobalStorage } from "../globalStorage"
import {useMatchesUpdatedSubscription} from '../api/useSubscriptions'

function myMergeArrays(arr_old, arr_new, key = "_id"){
  if(!arr_new){
    return arr_old
  }
  const answer = [...arr_old]
  arr_new.forEach((element) => {
```

```
      const index = answer.findIndex((old_element) => old_element[key] ===
element[key])
      if(index < 0){
          answer.push(element)
      }else{
          answer[index] = element
      }
  })
  return answer
}

const MatchList = () => {
    const [globalStorage, setGlobalStorage] = useGlobalStorage()
    const navigation = useNavigation();

    const getScheduleResult = useQuery(GET_SCHEDULE, {
      fetchPolicy: "no-cache"
    });


    const [matches, setMathes] = useState([])

    const [updateMatches, setUpdateMatches] = useState([])
    useMatchesUpdatedSubscription((info) => {
        const updated_matches = info.data.data.matchesUpdated
        setUpdateMatches(updated_matches)
    })

  useEffect(() => {
      if(getScheduleResult.data){
          if(!matches.length){
              setMathes(myMergeArrays(getScheduleResult.data.matches,
updateMatches))
          }else{
              setMathes(myMergeArrays(matches, updateMatches))
          }
      }else{
          setMathes(updateMatches)
      }
  }, [getScheduleResult.data, updateMatches])

    const onPressHandler = (match) => {
      if(match.summary){
        if(match.summary.state === "ENDED"){
          Alert.alert("матч вже закінчено")
          return
        }
      }
      if(match.team_1 && match.team_2){
        if(match.judge.login === globalStorage.login){
          navigation.navigate('MatchService', {match})
        }else{
          Alert.alert("Не ваш матч")
        }
      }else{
        Alert.alert("Ще не відомі команди матчу")
      }
    }

    if(getScheduleResult.loading){
      return (
        <Text>Загрузка...</Text>
      )
    }

    if(getScheduleResult.error){
      return (
```

```
        <View style={styles.form}>
          <Text>Помилка</Text>
          <Text>{getScheduleResult.error.message}</Text>
        </View>
      )
    }

    if(matches && !matches.length){
      return (
        <View style={styles.form}>
          <Text>Матчі не знайдено, ймовірно, немає ще розкладу</Text>
        </View>
      )
    }

    return (
      <SafeAreaView >
        <FlatList
            data={matches}
            renderItem={({item}) =>
              <TouchableOpacity onPress={() => onPressHandler(item)}>
                <MatchItem match={item}/>
              </TouchableOpacity>
            }
            keyExtractor={item => item._id}
        />
      </SafeAreaView >
    )
}


const styles = StyleSheet.create({
  form: {
    justifyContent: 'center',
    alignItems: 'center',
    flex:1
  },
});


export default MatchList;
```

MatchService.jsx

```
import { Text, StyleSheet, View, Button } from 'react-native'
import React, {useState} from 'react'
import {useMutation } from '@apollo/client';
import {ADD_GAME_EVENT} from "../api/mutations"
import {useMatchesUpdatedSubscription} from '../api/useSubscriptions'
import { useNavigation } from '@react-navigation/native';

const ScoreBlock = ({score_1, score_2}) => {
  return (
    <View style={styles.column}>
      <Text>{score_1}: {score_2}</Text>
      <Text>Фоли</Text>
      <Text>Тайм-аути</Text>
    </View>
  )
}

const TeamBlockInfo = ({team, fouls, timeouts}) => {
  return (
    <View style={styles.column}>
      <Text style={{textAlign: "center"}}>{team.name}</Text>
      <Text>{fouls}</Text>
```

```
      <Text>{timeouts}</Text>
    </View>
  )
}

const TeamBlockAction = ({team_id, match_id}) => {

  const [addGameEvent, addGameEventResult] = useMutation(ADD_GAME_EVENT)

  const sendGameEvent = (additional_data) => {
    addGameEvent({
      variables: {
        event_input:{
          timestamp: Date.now(),
          match: Number(match_id),
          team: Number(team_id),
          ...additional_data
        }
      }
    })
  }

  return (
    <View style={styles.column}>
      <Button style={styles.button}
        title="+1"
        onPress={() => sendGameEvent({
          event_type: "SCORE",
          score: 1
        })}
      />
      <Button style={styles.button}
        title="+2"
        onPress={() => sendGameEvent({
          event_type: "SCORE",
          score: 2
        })}
      />
      <Button style={styles.button}
        title="перс. фол"
        onPress={() => sendGameEvent({
          event_type: "PERSONAL_FOUL",
        })}
      />
      <Button style={styles.button}
        title="ком. фол"
        onPress={() => sendGameEvent({
          event_type: "TEAM_FOUL",
        })}
      />
      <Button style={styles.button}
        title="тайм-аут"
        onPress={() => sendGameEvent({
          event_type: "TIMEOUT",
        })}
      />
    </View>
  )
}

// export default function MatchService ({match_id, match}){
export default function MatchService (data){

    const navigation = useNavigation();

    const [match, setMatch] = useState({})
    React.useEffect(() => {
```

```
      setMatch(data.route.params.match)
    }, [])

    const [addGameEvent, addGameEventResult] = useMutation(ADD_GAME_EVENT)
    useMatchesUpdatedSubscription(({data}) => {
      if(!data){
        return;
      }
      if(!data.data.matchesUpdated){
        return;
      }
      const index = data.data.matchesUpdated.findIndex((element) => element._id ===
match._id)
      if(index !== -1){
        setMatch(data.data.matchesUpdated[index])
      }
    })

    const startServeMatch = () => {
      addGameEvent({
        variables: {
          event_input:{
            event_type: "START",
            timestamp: Date.now(),
            match: Number(match._id),
          }
        }
      })
    }

    const endServeMatch = () => {
      addGameEvent({
        variables: {
          event_input:{
            event_type: "END",
            timestamp: Date.now(),
            match: Number(match._id),
          }
        }
      }).then(() => {
        navigation.navigate('MatchList')
      })
    }

    if(!match.summary){
      return (
        <View style={{justifyContent: 'center', alignItems: 'center', flex:1}}>
          <Button
            onPress={startServeMatch}
            title="Почати"
          />
        </View>
      )
    }

    return (
      <View style={styles.mainColumn}>
        <View style={styles.mainContainer}>
          <TeamBlockInfo      team={match.team_1}      fouls={match.summary.fouls_1}
timeouts={match.summary.timeout_1}/>
          <ScoreBlock                                  score_1={match.summary.score_1}
score_2={match.summary.score_2}/>
          <TeamBlockInfo      team={match.team_2}      fouls={match.summary.fouls_2}
timeouts={match.summary.timeout_2}/>
        </View>
        <View style={styles.mainContainer}>
          <TeamBlockAction team_id={match.team_1._id} match_id={match._id}/>
```

```
          <TeamBlockAction team_id={match.team_2._id} match_id={match._id}/>
        </View>
        <Button
          onPress={endServeMatch}
          title="Закінчити"
        />
      </View>
    )
}

const styles = StyleSheet.create({
  mainContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    width: '100%',
    height: '47%'
  },
  column: {
    paddingTop: 15,
    paddingBottom: 15,
    flexDirection: 'column',
    justifyContent: 'space-between',
    alignItems: 'center',
    height: '100%',
    width: '30%'
  },
  mainColumn: {
    paddingTop: 15,
    paddingBottom: 15,
    flexDirection: 'column',
    justifyContent: 'space-between',
    alignItems: 'center',
    height: '100%',
    width: '100%'
  },
  button: {
    width: '90%',
  }
})
```

fields.js

```
export const MATCH_FIELDS = `
        _id
        team_1 {
            _id
            name
            short_name
        }
        team_source_1 {
            _id
        }
        team_2{
            _id
            name
            short_name
        }
        team_source_2{
            _id
        }
        planned_time
        pool{
            _id
            name
        }
```

```
        round
        judge{
            login
        }
        summary{
            score_1
            score_2
            fouls_1
            fouls_2
            timeout_1
            timeout_2
            state
            winner_team
        }
`
```

mutations.js

```
import {gql} from "@apollo/client";

export const ADD_GAME_EVENT = gql`
mutation AddGameEvent($event_input: GameEventInput!) {
  addGameEvent(event_input: $event_input) {
    _id
  }
}
`;

export const AUTHORIZATION = gql`
  mutation TokenAuth($username: String!, $password: String!) {
    tokenAuth(username: $username, password: $password) {
      token
      role
    }
  }
`;
```

queries.js

```
import {gql} from "@apollo/client";
import {MATCH_FIELDS} from "./fields";

export const AUTHORIZATION = gql`
  mutation TokenAuth($username: String!, $password: String!) {
    tokenAuth(username: $username, password: $password) {
      token
      role
    }
  }
`;

export const GET_SCHEDULE = gql(`
  query GetSchedule {
    matches {
        `+ MATCH_FIELDS + `
    }
  }
    `);

export const IS_STARTED = gql(`
query IsStarted{
    isStarted
}`)
```

useSubscriptions.js

```
import { useSubscription, gql } from "@apollo/client";
import {MATCH_FIELDS} from "./fields";

const MATCHES_UPDATED_SUBSCRIPTION = gql(`
  subscription MatchesUpdated {
    matchesUpdated {
        ` + MATCH_FIELDS + `
    }
  }
`);

export function useMatchesUpdatedSubscription(onData) {
    return useSubscription(MATCHES_UPDATED_SUBSCRIPTION,
        {
            onData,
            shouldResubscribe: true,
            fetchPolicy: "no-cache",
        });
}
```

Додаток З

Копія презентації

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Програмна система обслуговування для обслуговування турнірів з баскетболу 3x3

Виконав: Остапенко Іван Петрович

Керівник: асистент, Комісар Дмитро Олександрович

Київ – 2023

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: створити програмну систему з обслуговування турнірів з баскетболу 3x3, що частково автоматизує обслуговування турнірів.

Завдання:

1. Проаналізувати процес обслуговування турнірів з баскетболу 3x3.
2. Провести аналіз сучасних технологій розробки вебзастосунків та мобільних додатків.
3. Розробити програмну систему.
4. Протестувати розроблену систему.

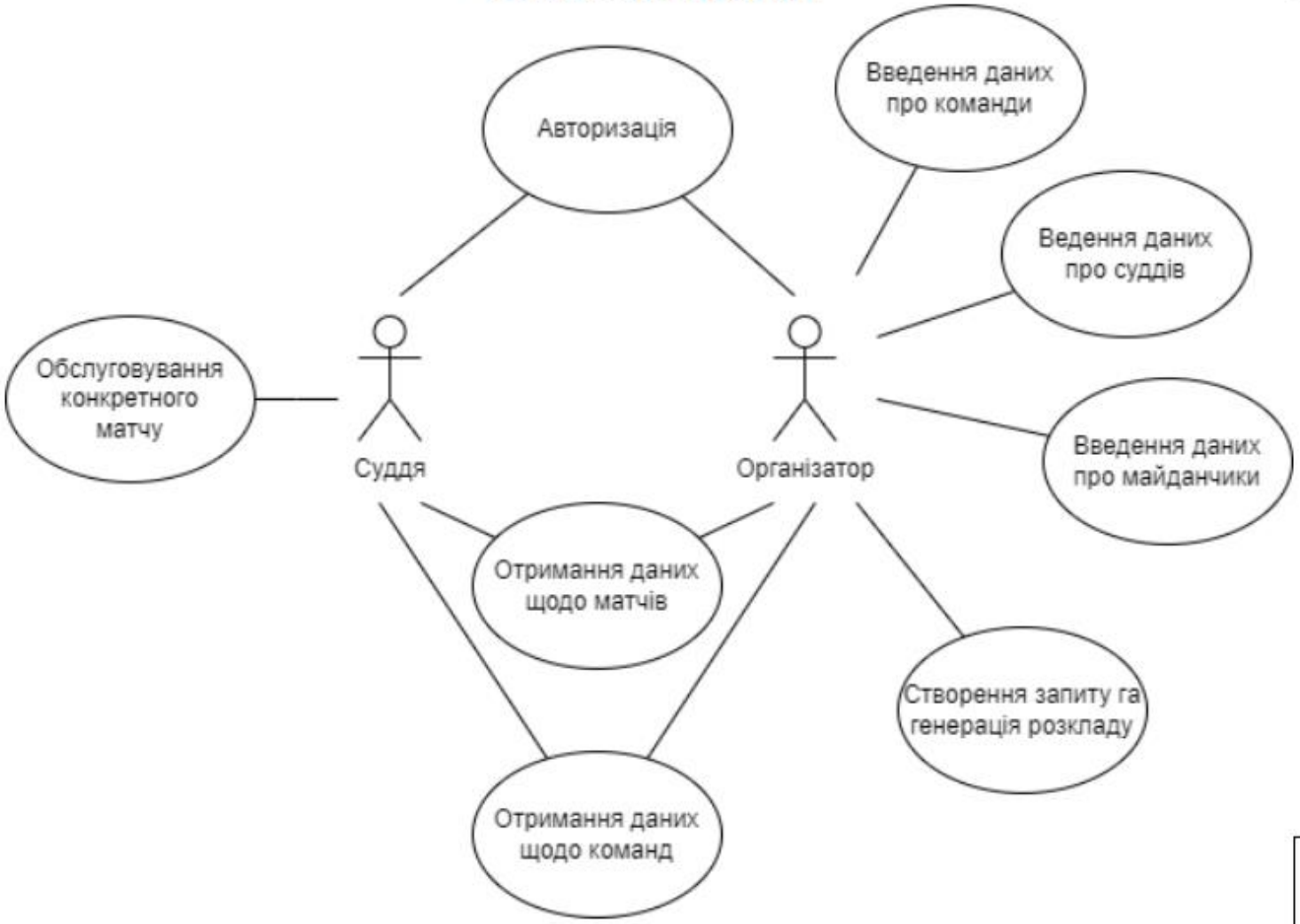
АКТУАЛЬНІСТЬ

Баскетбол 3х3 - популярний вид спорту, що широко розвинувся в Україні. Проведення турнірів має систематичний характер.

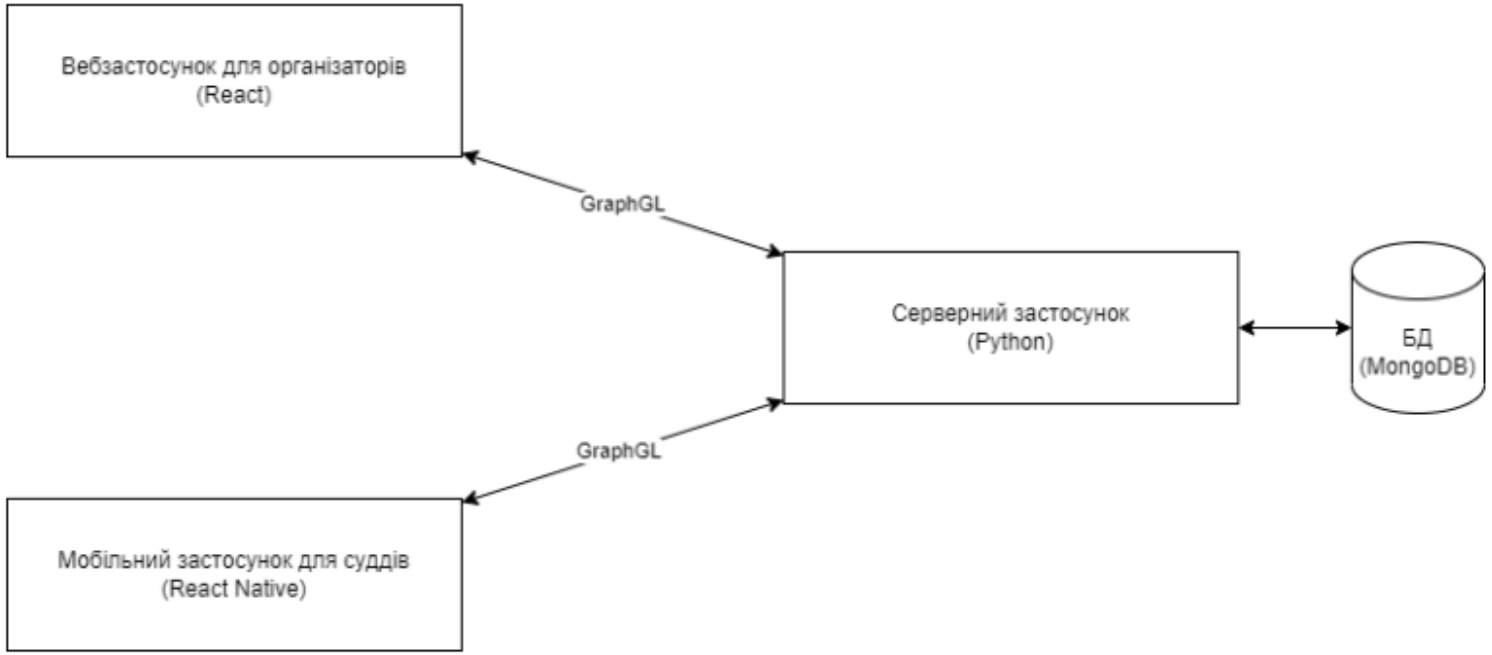
Після проведення опитування гравців, суддів, організаторів, вболівальників, було виявлено потребу у програмній системі, що:

- спрощувала обслуговування турнірів;
- покращувала інформування користувачів щодо перебігу турніру.

ДІАГРАМА ВИКОРИСТАННЯ СИСТЕМИ

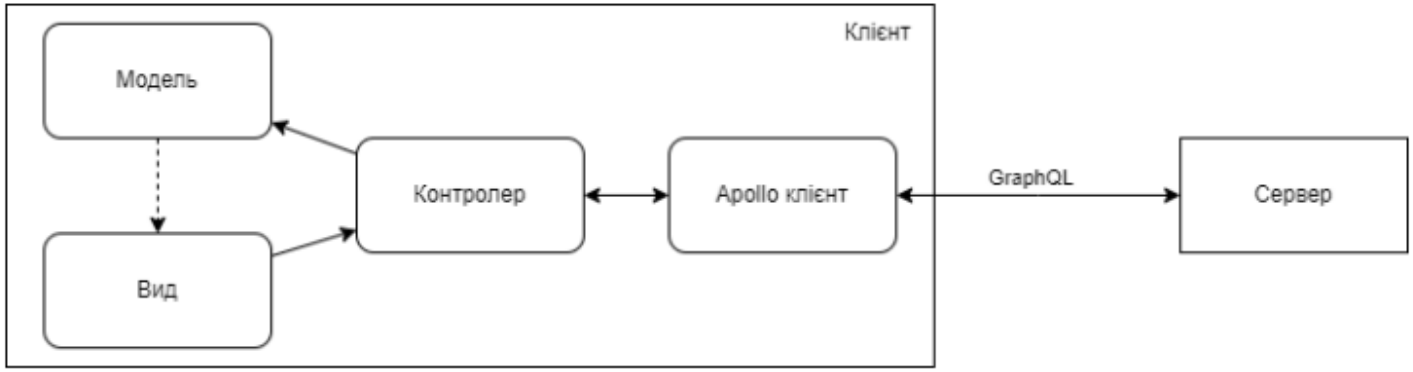


ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ

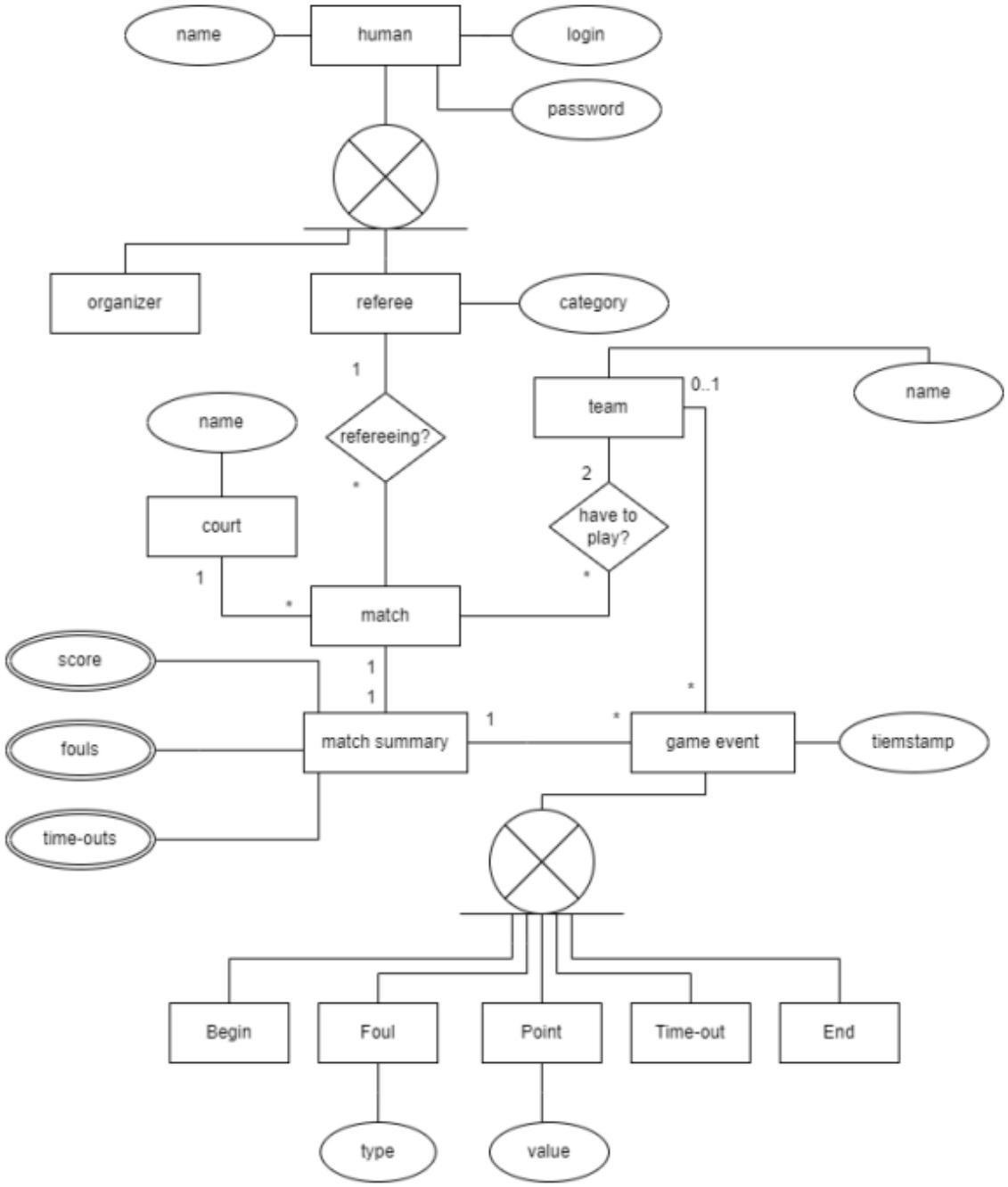


Клієнт-серверна архітектура

АРХІТЕКТУРА КЛІЄНТІВ



АРХІТЕКТУРА СЕРВЕРА


name
human
login
password
organizer
referee
category
1
name
refereeing?
*
court
1
*
match
team
0..1
name
2
have to play?
*
score
fouls
time-outs
1
1
match summary
1
*
game event
*
tiemstamp
Begin
Foul
Point
Time-out
End
type
value

ER-діаграма

ВИСНОВКИ

1. Проведено аналіз предметної області, а саме процес обслуговування турнірів з баскетболу 3х3.
2. Опитано користувачів, сформульовано вимоги до програмної системи.
3. Проаналізовано сучасні протоколи взаємодії компонентів клієнт-серверної архітектури, технології розробки серверної частини, вебзастосунків та мобільних застосунків.
4. Розроблено програмну систему з використанням React, React Native, Python, MongoDB, GraphQL.
5. Протестовано програмну систему шляхом мануального тестування та з використанням unit тестування.
6. Вказано напрямки вдосконалення програмної системи.

Розробка виконана у повному обсязі згідно з положеннями Технічного завдання.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2022 р.

ПРОГРАМНА СИСТЕМА ОБСЛУГОВУВАННЯ ТУРНІРІВ З БАСКЕТБОЛУ 3х3

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро КОМІСАР

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ОСТАПЕНКО

2022

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмна система обслуговування турнірів з баскетболу 3х3, що складається з вебзастосунку, мобільного застосунку, сервера, що написані за допомогою React, React Native, Python відповідно.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів сторінок вебзастосунку та мобільного застосунку;

2) функціональна працездатність авторизації;

3) функціональна працездатність генерації розкладу;

4) розмежування доступу зі сторони сервера;

5) відповідність вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується за допомогою автоматизованого та ручного тестування.

За допомогою автоматизованого тестування перевіряється робота сервера шляхом написання unit тестів.

За допомогою ручного тестування вебзастосунку та мобільного застосунку перевіряється відповідність програмної системи вимогам Технічного завдання.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Автоматизоване тестування серверу виконується засобами бібліотеки unittest.

Працездатність вебзастосунку та мобільного застосунку за допомогою ручного тестування перевіряється шляхом:

1) введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) перевірки синхронізації клієнтів;
4) тестування зручності використання;
5) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

ПРОГРАМНА СИСТЕМА ОБСЛУГОВУВАННЯ ТУРНІРІВ З БАСКЕТБОЛУ 3х3

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро КОМІСАР

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ОСТАПЕНКО

2023

ЗМІСТ

1. Опис програмної системи

Програмна система обслуговування турнірів з баскетболу 3x3 складається з двох застосунків:

1) вебзастосунок для організаторів;
2) мобільний застосунок для суддів.

2. Опис вебзастосунку для організаторів

Доступ до функціональних можливостей доступний лише авторизованим користувачам.

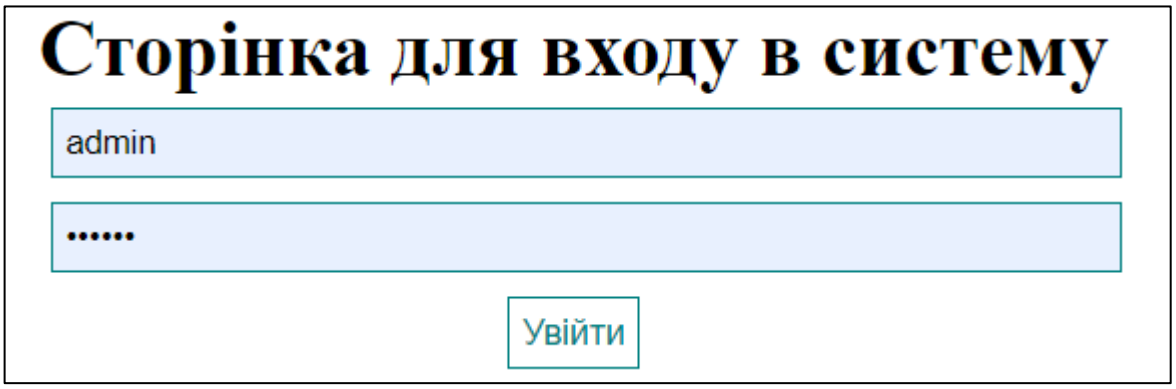


Рис. 2.1. Сторінка авторизації

Навігаційна панель містить переходи до 5 сторінок: “Інформація”, “Майданчики”, “Команди”, “Судді”, “Розклад”, а також можливість вийти з системи.

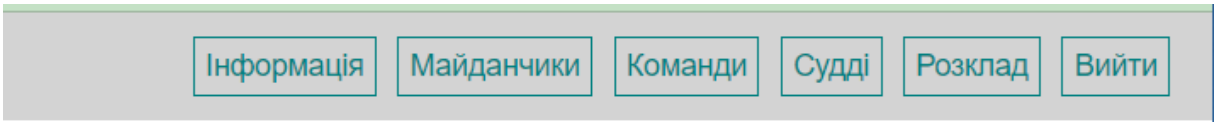


Рис. 2.2. Навігаційна панель

Сторінка “Інформація” містить загальну інформацію про систему.

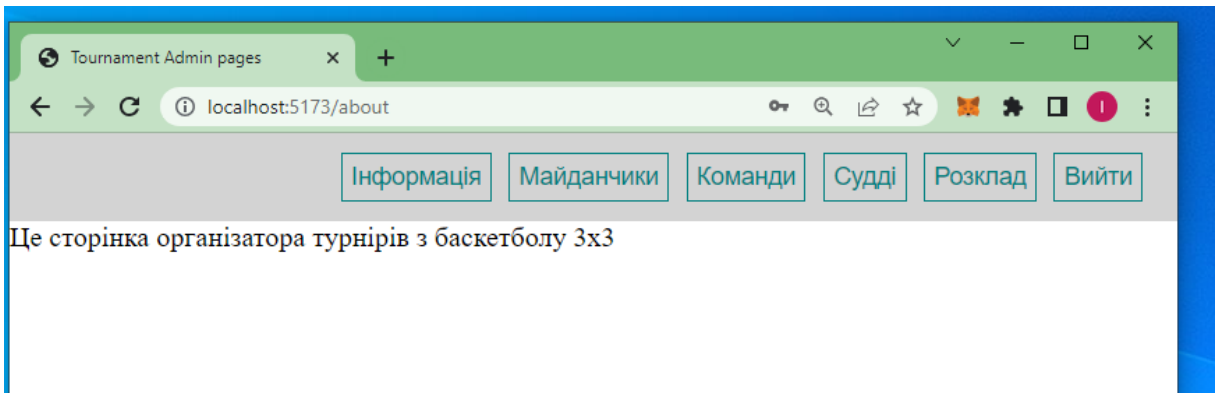


Рис. 2.3. Сторінка “Інформація”

Сторінки “Майданчики”, “Команди”, “Судді” призначені для перегляду вже зареєстрованих майданчиків, команд, суддів відповідно. Якщо розклад ще не сформований, то є можливість зареєструвати відповідповідну сутність.

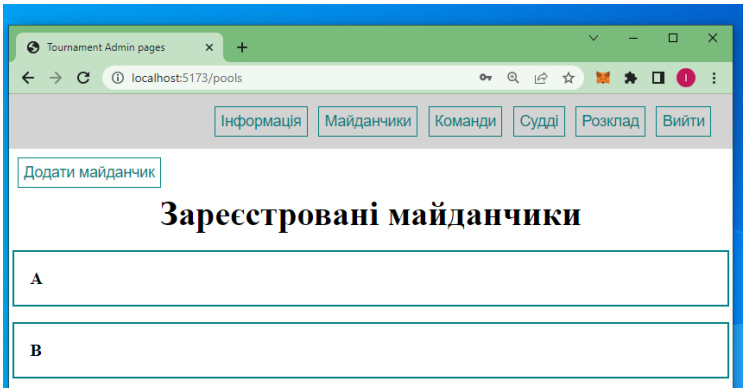


Рис. 2.4. Сторінка “Майданчики”

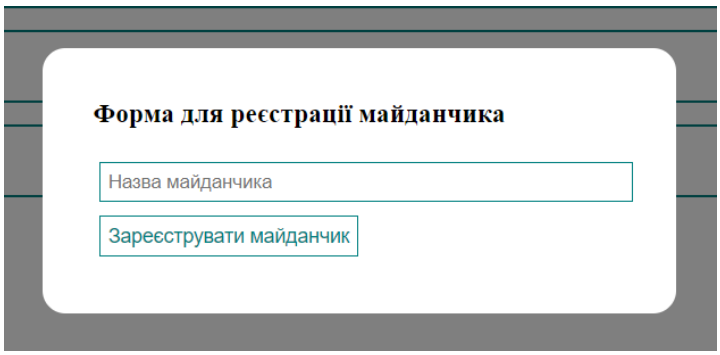


Рис. 2.5. Форма для реєстрації майданчика

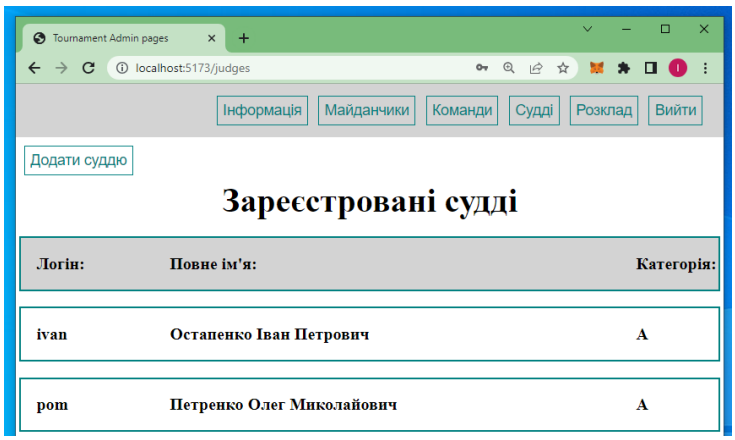


Рис. 2.6. Сторінка “Судді”

Рис. 2.7. Форма для реєстрації судді

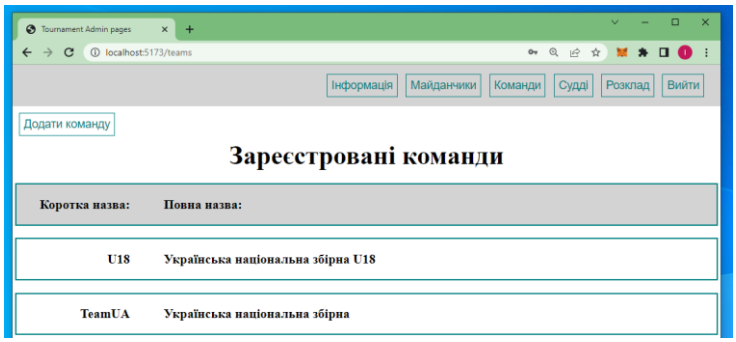


Рис. 2.8. Сторінка “Команди”

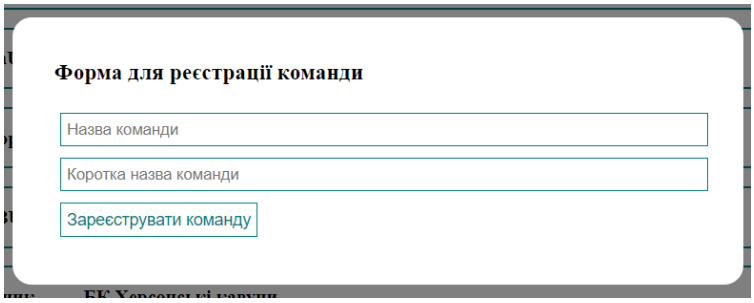


Рис. 2.9. Форма для реєстрації команди

Сторінка “Розклад” призначення для формування розкладу та перегляду перебігу матчів. У користувача є лише одноразово згенерувати розклад.

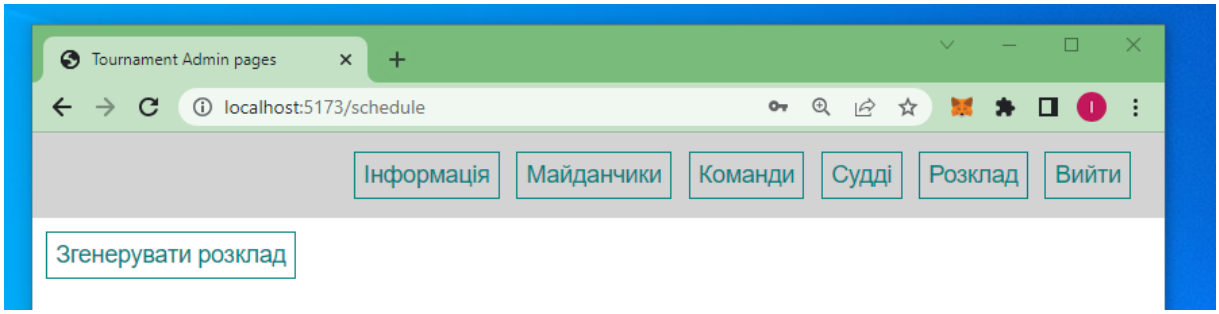


Рис. 2.10. Сторінка “Розклад” до генерації розкладу

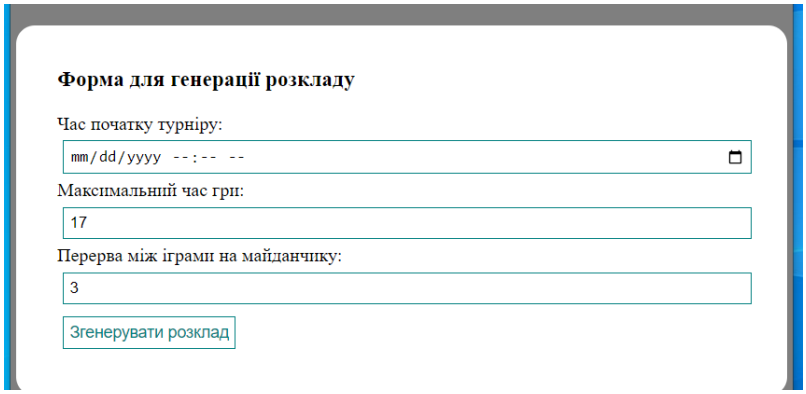


Рис. 2.11. Форма для генерації розкладу

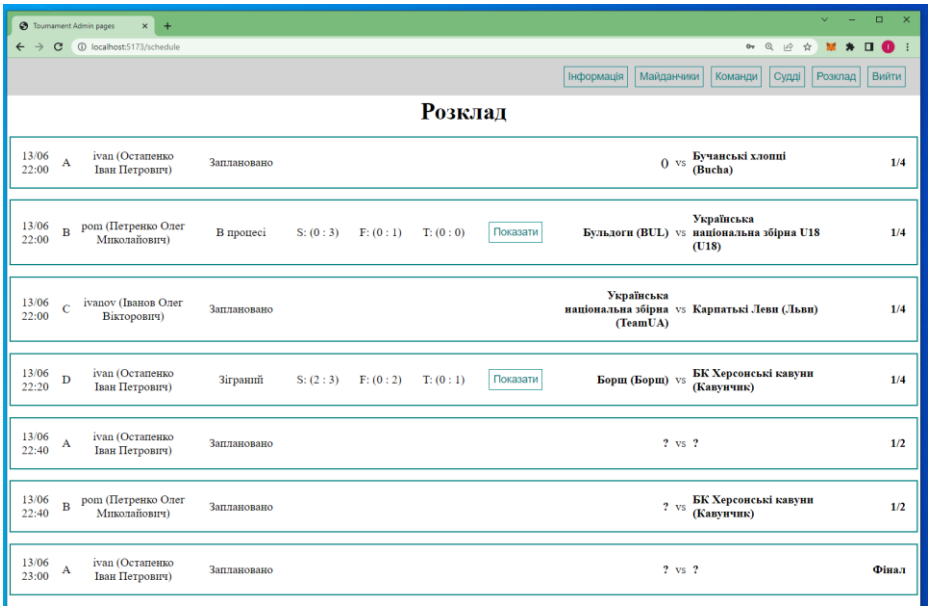


Рис. 2.12. Сторінка “Розклад” після згенерованого розкладу

Значення полів рядку матчу (зліва на право):

1) запланований час початку гри;
2) майданчик де заплановано гру;
3) суддя, що має обслуговувати матч;
4) стадія гри (“Заплановано”, “В процесі”, “Зіграний”);
5) короткі відомості щодо матчу, якщо матч вже почався (кількість очків (S), фолів (F), таймаутів (T));
6) команди, що грають у матчі:
   - якщо “()”, то команда відсутня, що можливо, якщо команд-учасників неповна кількість для олімпійської системи;
   - якщо “?”, то команда ще не визначена, матч попереднього туру ще не закінчився;
   - повна назва команди, а також їх коротка назва;
7) раунд матчу.

При натисканні кнопки “Показати” з’являється можливість переглянути протокол в режимі реального часу (для матчів “В процесі”), або остаточний протокол (для матчів “Зіграний”).

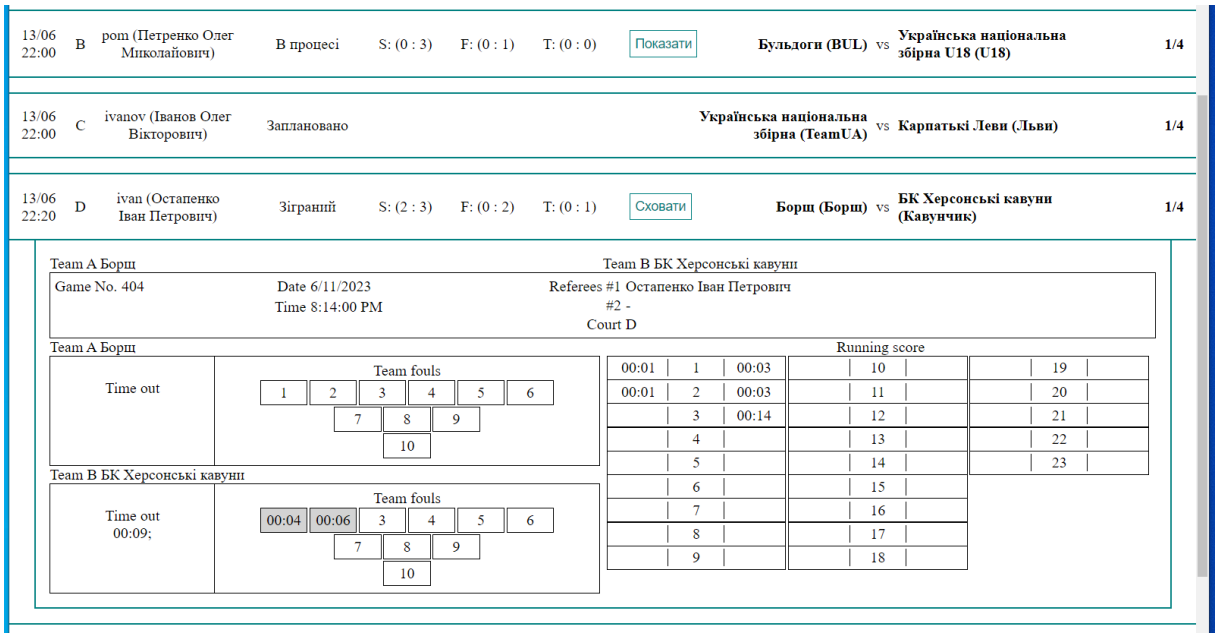


Рис. 2.7. Перегляд протоколу матчу

В протоколі матчів можна переглянути точний ігровий час фіксування події, наприклад, на рис. 2.7:

- команда “Борщ”:
  - забила двоочковий на 1 секунді гри;
- команда “БК Херсонські кавуни”:
  - забила двоочковий на 3 секунді гри;
  - сфолила на 4 та 6 секунді;
  - взяла таймаут на 9 секунді;
  - забила одноочковий на 14 секунді.

3. Опис мобільного застосунку для суддів

Доступ до функціональних можливостей доступний лише авторизованим користувачам. Для отримання логіну та пароля до системи, необхідно звернутися до організаторів, щоб вони зареєстрували користувача.

Рис. 3.1. Сторінка авторизації

Сторінка розкладу матчів зображено на рис. 3.2.

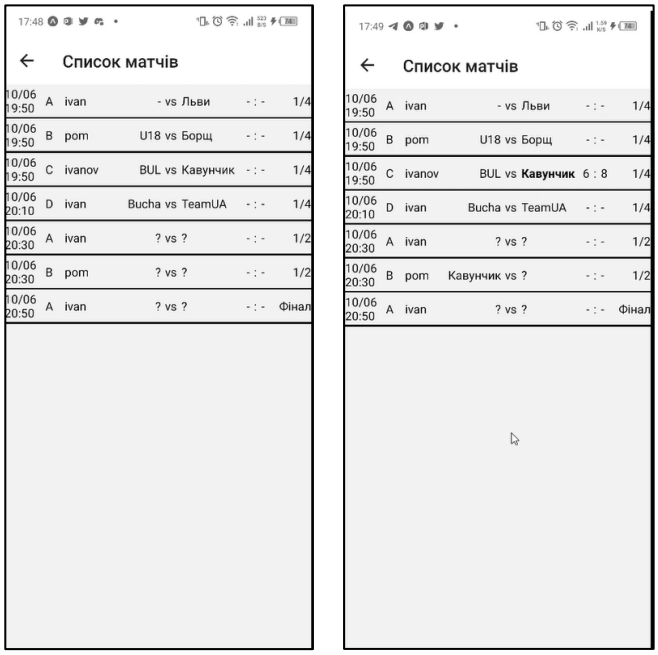


Рис. 3.2. Сторінка розкладу матчів

Значення полів рядку матчу (зліва на право):

1) запланований час початку гри;
2) майданчик де заплановано гру;
3) суддя, що має обслуговувати матч;
4) команди, що грають у матчі:
   - якщо “-”, то команда відсутня, що можливо, якщо команд-учасників неповна кількість для олімпійської системи;
   - якщо “?”, то команда ще не визначена, матч попереднього туру ще не закінчився;
   - коротка назва команди, якщо команда виділена жирним шрифтом, то вона виграла цей матч і матч закінчено;
5) поточний розклад:
   - якщо “- : -” то матч ще не почався;
   - якщо “<число1>: <число2>”, то матч вже почався;
6) раунд матчу.

Для обслуговування матчу, необхідно натиснути на відповідний рядок, з даними про матч.

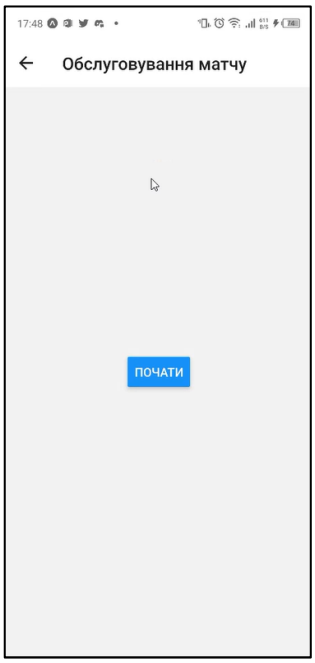


Рис. 3.3. Початкове вікно обслуговування матчу

Для початку обслуговування матчу необхідно натиснути “Почати”.

Рис. 3.4. Основне вікно обслуговування матчу

Згори вікна відображено інформацію про (зверху-вниз):

1) назви команд та поточний рахунок;
2) поточна кількість фолів;
3) поточна кількість тайм-аутів.

У користувача є можливість зафіксувати ігровий момент, шляхом натискання кнопок (зверху-вниз):

1) “+1” – команда закинула одноочковий;
2) “+2” – команда закинула двоочковий;
3) “ПЕРС. ФОЛ” – команда отримала персональний фол;
4) “КОМ. ФОЛ” – команда отримала командний фол;
5) “ТАЙМ-АУТ” – команда використала тайм-аут.

У користувача є можливість закінчити матч, натиснувши кнопку “ЗАКІНЧИТИ”.