

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Веб-система жеребкування»

Виконав студент групи ПЗ-20-1
ВОЛКОВ Олександр Геннадійович

Керівник роботи:
СЕМЕРІКОВ Сергій Олексійович

Консультант:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ВЛАСЕНКО Олег Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВОЛКОВ Олександр Геннадійович**

Керівник роботи: **СЕМЕРІКОВ Сергій Олексійович**

Тема роботи: **«Веб-система жеребкування»**,

затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с

Вихідні дані для роботи: **веборієнтована система жеребкування турнірів з використанням принципів об'єктно-орієнтованого програмування.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р. С. Семеріков С. О.	27.02.2024р.	12.04.2024р.
2	Савіцький Р. С. Семеріков С. О.	27.02.2024р.	16.05.2024р.
3	Савіцький Р. С. Семеріков С. О.	27.02.2024р.	30.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку системи для жеребкування турнірів та пояснювальної записки. Пояснювальна записка до випускної роботи складається зі вступу, 3-х розділів, висновків, списку використаних джерел із 15 найменувань, 1-го додатку, 12-ти таблиць та 42-х ілюстрацій по тексту. Повний обсяг роботи становить 66 сторінок, з них містить 57 сторінок основного тексту.

Метою роботи є створення системи для жеребкування, використовуючи алгоритми об'єктно-орієнтованого програмування.

В першому розділі проаналізовано проєкт та поставлено задачу на кваліфікаційну роботу. Виконано порівняння аналогічних продуктів та визначено основний функціонал додатку, обґрунтовано вибір технологій.

В другому розділі спроектовано систему жеребкування турнірів, визначено об'єктно-орієнтовану структуру програмного продукту. Складено алгоритми роботи системи, розроблено та описано програмний код застосунку.

В третьому розділі описано схему розгортання системи. Наведено приклади інтерфейсу додатку. Проведено тестування, використовуючи підходи мануального тестування та браузерні інструменти.

В результаті було розроблено програмний продукт зі зручним адаптивним інтерфейсом, який дозволяє користувачам створювати та керувати турніри та проводити жеребкування.

КЛЮЧОВІ СЛОВА: ЖЕРЕБКУВАННЯ, ТУРНІР, MVC, ПАТЕРН, WEB, УЧАСНИКИ

					ІПЗ.КР.Б-121-24-ПЗ	
						3

ABSTRACT

The bachelor thesis consists of a draw system application and an explanatory note. The explanatory note to the final thesis consists of an introduction, 3 chapters, conclusions, a list of used sources with 15 titles, 1 appendix, 12 tables and 42 illustrations in the text. The full volume of the work is 66 pages, of which 57 pages contain the main text.

The objective of this work is to create a system for conducting tournament draws using object-oriented programming algorithms.

In the first section, we analyze the project and define the task for the qualification work. We compare similar products, determine the main functionality of the application, and justify our technology choices.

The second section focuses on designing the draw system and defining the object-oriented structure of the software product. We compile algorithms for the system's operation and develop and describe the application's software code.

The third section describes the system deployment scheme, provides screenshots of the application's interface, and details testing conducted using manual approaches and browser tools.

As a result, a software product was developed with a user-friendly, responsive interface that allows users to create and manage tournaments and conduct draws.

KEYWORDS: DRAWING, TOURNAMENT, MVC, PATTERN, WEB, PARTICIPANTS.

					ИИ3.КР.Б-121-24-И3	
						4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
1.1. АНАЛІЗ ПРОЄКТУ ТА ПОСТАНОВКА ЗАДАЧІ	9
1.2. АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ.....	11
1.3. ВИБІР АРХІТЕКТУРИ СИСТЕМИ ЖЕРЕБКУВАННЯ	16
1.4. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	20
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ЖЕРЕБКУВАННЯ ТУРНІРІВ.....	24
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	24
2.2. Розробка бази даних системи.....	29
2.3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ СИСТЕМИ.....	33
2.4. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СИСТЕМИ ЖЕРЕБКУВАННЯ ТУРНІРІВ	35
Висновки до другого розділу	43
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СИСТЕМОЮ ЖЕРЕБКУВАНЬ.....	44
3.1. Порядок встановлення та налаштування параметрів платформи	44
3.2. Структура інтерфейсу системи жеребкування турнірів.....	45
3.3. Тестування платформи	51
Висновки до третього розділу	53
ВИСНОВКИ.....	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	56
ДОДАТКИ.....	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ІТ – інформаційні технології

СКБД – система керування базами даних

CRUD – create, read, update, delete

DI – dependency injection

EF – Entity Framework

JSX – JavaScript Syntax eXtension

LINQ – language integrated query

MVC – model view controller

ORM – object-relational mapping

					ІПЗ.КР.Б-121-24-ПЗ	
						6

ВСТУП

Актуальність теми лежить у тому, що у сьогоденні інформаційні нововведення та технології суттєво змінили хід проведення багатьох заходів. Зростання популярності виконання операцій в інтернеті, популяризація спорту, ігор, змагань та потреби до автоматизації буквально кожного процесу, що оточує людство, роблять розробку вебсистеми для жеребкування турнірів незмірно актуальною. Система забезпечує зручність і доступність користувачам, дає змогу для автоматизації процесів жеребкування. Крім того, збільшення кількості методів та ресурсів, спрямованих у спрощенні та автоматизації процесів для змагальних тематик утворює додатковий попит на саме їх використання різними верстами суспільства. Отже, розробка платформи відповідає сучасним потребам ринку та розвиває промисел розваг, культури та товариського змагання.

Зі зростанням попиту на інтернет-послуги, більшої мірою, завдяки розвитку технологій, які спрямовані на спрощення та доступність для клієнтів, потреби в розвагах також не стоять на місці та зазнають змін. Важливим фактором стала пандемія 2020-го року, коли люди, сидячи вдома, почали грати в комп'ютерні ігри різних жанрів. Це призвело до притоку нових поціновувачів, деякий відсоток з яких, захотіли проводити або брати участь у всіляких турнірах різного рівня професійності.

Метою випускної кваліфікаційної роботи бакалавра є розробка вебсистеми для жеребкування турнірів.

Завдання обумовлені встановленою метою:

- проведення аналізу існуючих вебсистем для жеребкування турнірів;
- визначення структури та архітектури системи;
- вибір технологій для реалізації проєкту;
- проєктування об'єктно-орієнтованої системи;
- створення бази даних;
- описання структури інтерфейсу;
- тестування проєкту.

Об’єктом дослідження є веборієнтована система для жеребкування турнірів різних видів.

Предметом дослідження предметом дослідження є розробка системи для жеребкування з використанням алгоритмів для якісного виконання поставленого користувачем завдання.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези [1].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ЖЕРЕБКУВАННЯ

1.1. Аналіз проєкту та постановка задачі

Метою випускної кваліфікаційної роботи бакалавра є розробка вебсистеми для жеребкування турнірів. Ця система дозволить автоматично проводити жеребкування учасників турнірів, забезпечуючи об'єктивність та випадковість вибору. Вона буде включати функціональність для введення даних про учасників, налаштування правил жеребкування, а також відображення результатів. Така система сприятиме ефективному проведенню спортивних турнірів та інших подій.

У сучасному світі організація та проведення спортивних турнірів стає все більш популярною та вимагає нових підходів до організації. З метою спрощення та оптимізації цього процесу та забезпечення справедливого та ефективного розподілу учасників, розробляється вебсистема для жеребкування турнірів.

Проведення спортивних змагань та турнірів завжди викликало великий інтерес у спортивній спільноті. Однак, з розвитком технологій та зростанням кількості учасників, необхідність у зручних та ефективних інструментах для організації таких подій стає більш актуальною. У цьому контексті, розробка системи для жеребкування турнірів стає важливим етапом у забезпеченні успішного та справедливого проведення спортивних змагань. Ціль цього проєкту полягає у створенні високоякісної та функціональної вебсистеми, що відповідає потребам сучасної спортивної громадськості.

Така система може бути багатофункціональною, проте для її застосування необхідні такі основні модулі:

- модуль авторизації, в якому користувач матиме змогу зареєструватись, для користування системою;
- модуль створення турніру, де користувач матиме змогу вибрати пасуючі параметри для створення турніру;

- модуль менеджменту учасників створеного турніру, де користувач зможе керувати користувачами та їх вхідними для турніру даними;
- модуль перегляду існуючих турнірів, де користувач зможе переглядати попередньо створені турніри та зручно переходити по сайту;
- модуль турнірних сіток, що дасть змогу користувачеві прослідкувати створення самих сіток та керувати переможцями матчів між учасниками.

Ключові кроки створення вебдодатка включають:

1. Дизайн сайту:

- розробка концептуального дизайну;
- створення UI елементів;
- дизайн вебсторінок.

2. Архітектура клієнтської сторони:

- вибір технологій для клієнтської частини;
- створення UI компонентів.

3. Архітектура серверної сторони:

- вибір технологій для серверної частини;
- розробка API для зв'язку з клієнтом.

4. Система авторизації та ролей:

- механізми реєстрації та авторизації;
- система ролей для управління доступом.

5. Основний функціонал:

- створення турнірів;
- жеребкування матчів різними методами.

6. тестування:

- перевірка функціональності;
- тестування інтерфейсу.

Результатом виконання плану дій є створення платформи, за допомогою якої користувачі матимуть змогу створювати, керувати та переглядати створені матчі та турнірні сітки, що дасть змогу ефективно та доступно працювати як і з

					ІПЗ.КР.Б-121-24-ПЗ	
						10

чемпіонатами, так і з аматорськими змаганнями. Іншими словами, користувач зможе взаємодіяти з різними типами матчів.

1.2. Аналіз аналогів програмного продукту

В сучасному світі, де інформаційні технології стають все більш важливими, вебсистеми здобувають популярність. Однією з таких систем є вебсистема жеребкування, яка дозволяє проводити випадковий вибір або розподіл ресурсів, будь то призи, місця в групі або інші об'єкти.

Аналіз аналогів програмного продукту є важливим етапом в розробці вебсистеми. Він допомагає визначити сильні та слабкі сторони існуючих рішень, зрозуміти, які функціональності можуть бути корисними для нашої системи, а також визначити можливі напрямки для покращення.

Для аналізу аналогів вебсистеми жеребкування ми рекомендуємо вивчити існуючі рішення, такі як системи жеребкування для лотерей, розподілу місць на подіях, випадковий вибір відповідей у голосуваннях тощо.

Turnir.net.ua [2] – це онлайн-платформа, спеціально розроблена для проведення жеребкування спортивних турнірів (рис. 1.1). Її функціонал зосереджений на цій задачі, пропонуючи гнучкі налаштування для різних форматів змагань.

Платформа має простий і зрозумілий інтерфейс, що робить її доступною для користувачів з різним досвідом, але й водночас, він відчувається відносно старим, що може не сподобатись деяким користувачам, особливо молодшого віку. Turnir.net.ua гарантує високий рівень безпеки, захищаючи дані учасників та забезпечуючи чесність жеребкування. Швидка обробка запитів та можливість масштабування дозволяють використовувати платформу для організації масштабних турнірів.

Turnir.net.ua відрізняється від аналогів вузькою спеціалізацією, що робить його максимально зручним та ефективним для проведення жеребкування. Платформа має простий інтерфейс, гарантує високий рівень безпеки, швидко обробляє запити та може використовуватися для організації масштабних змагань.

Переваги Turnir.net.ua:

- функціональність: гнучкі налаштування для різних форматів змагань, різноманітні алгоритми жеребкування, можливість створення груп та підгруп, встановлення правил та обмежень, зберігання результатів жеребкування;
- безпека: високий рівень захисту від несанкціонованого доступу, гарантія конфіденційності даних учасників;
- швидкодія: швидка обробка запитів, здатність обслуговувати велику кількість учасників;
- масштабованість: можливість використовувати платформу для організації масштабних турнірів.

Серед недоліків ресурсу Turnir.net.ua можна виділити неінтуїтивність інтерфейсу – важко освоїти технічні функції звичайному користувачеві. Іншими словами, створення самого турніру вимагає додаткової консультації. Вебсистема наразі підтримує лише проведення жеребкування для небагатьох турнірів, що є суттєвим мінусом для організаторів змагань з інших видів спорту, не перерахованих у опціях. Повертаючись до інтерфейсу, він може здатися застарілим та незручним для користувачів, які звикли до більш сучасних та елегантних вебсайтів.

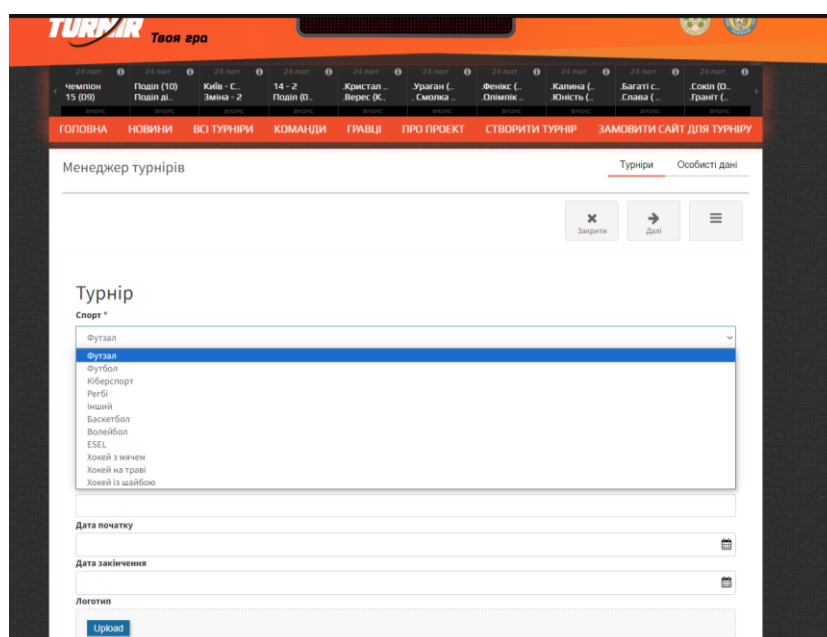


Рис. 1.1 – Вигляд інтерфейсу turnir.net.ua

Chess.com [3] – це онлайн-платформа, спеціально розроблена для онлайн матчів з шахів. Враховуючи, що шахи – це один з найпопулярніших видів спорту в світі, особливо після глобального карантину, шахи отримали новий погляд. Тому, сама ідея гри в шахи на професійному рівні зросла, збільшилася і зріс попит на створення турнірів, що дозволяє сама платформа.

Платформа має мінімалістичний та зрозумілий інтерфейс, що робить її доступною для користувачів з різним досвідом. Chess.com гарантує безпеку даних користувачів за допомогою шифрування. Ефективна робота з великою кількістю користувачів дозволяє використовувати платформу для організації масштабних турнірів. Інтеграція з соціальними мережами робить процес реєстрації та участі в змаганнях максимально зручним.

Переваги Chess.com:

- функціональність: гнучкі налаштування для різних форматів турнірів, автоматичний розрахунок очок та рейтингів, можливість створення груп та підгруп, публікація результатів;
- інтерфейс: простий та зрозумілий, не перевантажений зайвими елементами;
- безпека: шифрування даних користувачів, гарантія конфіденційності;
- швидкодія: ефективна робота з великою кількістю користувачів;
- інтеграція з соцмережами: зручна реєстрація та участь у змаганнях.

Chess.com – це універсальний інструмент для організаторів шахових турнірів, який робить процес розподілу місць чесним, прозорим та максимально зручним.

Найбільшим мінусом даного ресурсу є той факт, що для того, щоб хостити турнір, потрібно мати підписку Голд чи вище (рис. 1.2).

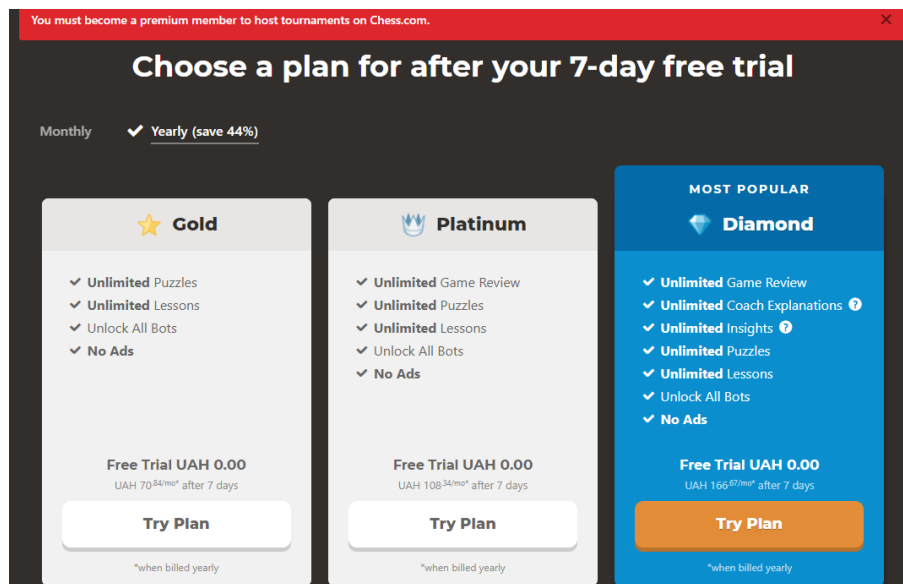


Рис. 1.2 – Повідомлення щодо підписки на Chess.com

Winner [4] – це мобільний додаток, спеціально розроблений для організації спортивних голосувань (рис. 1.3). Він пропонує зручний та інтуїтивний інтерфейс, а також широкий спектр функцій, що робить його ідеальним інструментом для організаторів змагань.

Основні переваги Winner:

- швидке створення турнірів: додаток створює турніри доволі швидко;
- мобільний додаток: Winner доступний як мобільний додаток, що робить його зручним для використання на будь-якому місці та в будь-який час;
- велика кількість форматів: платформа дозволяє обирати різні формати турніру та методи жеребкування;
- двофакторна аутентифікація: Winner використовує двофакторну аутентифікацію для додаткового захисту ваших даних;
- оптимізована робота на слабких пристроях: Додаток оптимізований для роботи на слабких пристроях, що робить його доступним для широкого кола користувачів;
- підтримка різних мов: додаток доступний різними мовами, що робить його зручним для користувачів з усього світу;
- система подій в турнірі: учасники отримують сповіщення про різні типи подій, залежні від обраної гри.

Головним мінусом програми є його плюс: доступність лише на мобільних пристроях. Winner доступний лише для користувачів Android та IOS, що може бути незручним для користувачів персональних комп’ютерів.

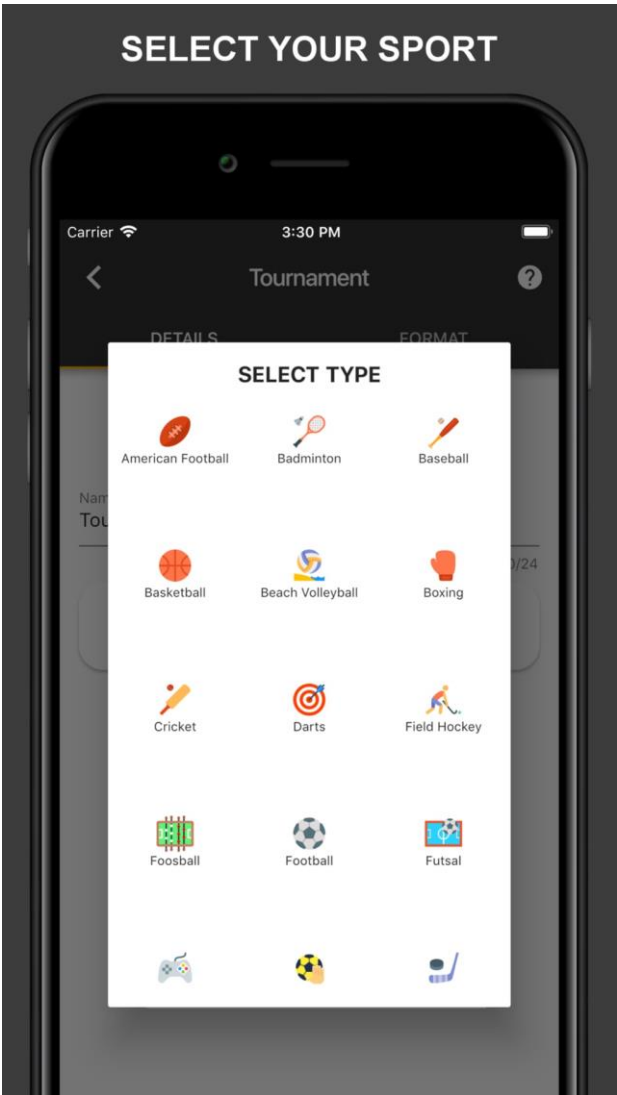


Рис. 1.3 – Вигляд інтерфейсу Winner

Підсумовуючи, chess.com має гарний інтерфейс, швидкісну дію та є захищеним і перевіреним мільйонами користувачів вебзастосунком, але для організації турнірів потрібно оплатити підписку. Winner це інтуїтивний, багатомовний мобільний застосунок. Його мобільність також є і його недоліком. turnir.net.ua являється сайтом з неінтуїтивним інтерфейсом та малою кількістю інформації про проведення матчів призводять до потенційного спантеличення серед користувачів, і водночас, має малу кількість опцій для вибору спорту.

Наглядніше порівняння знаходиться в таблиці 1.1:

					ІПЗ.КР.Б-121-24-ПЗ	
						15

Таблиця 1.1

Аналог	Функціональність	Інтерфейс	Безпека	Швидкодія	Інші аспекти
Turnir.net.ua	Проведення жеребкування для спортивних турнірів	Не інтуїтивний	Реєстрація через гугл чи фейсбук	Швидка обробка запитів	Підтримка масштабування
Chess.com	Розподіл місць на шахових турнірах	Мінімалістичний та зрозумілий інтерфейс	Шифрування даних користувачів	Ефективна робота з великою кількістю користувачів	Інтеграція з соціальними мережами
Winner створення турнірів	Випадковий вибір учасників для спортивних голосувань	Мобільний додаток з інтуїтивним інтерфейсом	Двофакторна аутентифікація	Оптимізована робота на слабких пристроях	Підтримка різних мов

Отже, у порівнянні з аналогічними програмами, додаток, що розробляється, включає всі основні функції, необхідні для зручного використання системи для жеребкування. Більше того, в деяких аспектах цей застосунок навіть перевершує своїх конкурентів.

1.3. Вибір архітектури системи жеребкування

Згадуючи слово «архітектура», неважко зробити асоціацію з вишуканими соборами Європи, які протягом століть зберігають свою велич і можуть. Їх вражаючі арки та колони не лише захоплюють, а й слугують фундаментом стійкості та довговічності будівлі. Подібно до цього, архітектура програмного забезпечення є невидимим фундаментом, на якому тримається вся інформаційна система, забезпечуючи її якість, доступність та довговічність.

Правильно спроектована архітектура спрощує подальшу розробку та впровадження системи. Адекватне планування архітектури сприяє подальшій розробці, впровадженню, експлуатації та модифікації системи [5].

Під час розробки будь-якої програмної системи, вибір відповідної архітектури визначає основні принципи та напрямки подальшої роботи. Це ключовий етап, що вимагає ретельного аналізу потреб користувачів, вимог до системи та особливостей проєкту. У цьому контексті, підбір оптимальної архітектурної моделі є важливим для забезпечення ефективності, масштабованості та гнучкості розроблюваного продукту.

Для досягнення поставлених цілей та врахування особливостей проєкту, було розглянуто декілька ключових архітектурних патернів, які були використані для побудови системи для жеребкування турнірів. Кожен з них має свої переваги та обмеження, і правильний вибір допоможе забезпечити успішний розвиток та ефективне функціонування додатку.

Отже, беручи до уваги вибір архітектури системи можна виділити наступні архітектурні патерни:

Патерн Клієнт-Сервер [6] (Client-Server) передбачає розділення системи на дві основні частини: клієнтську (інтерфейс користувача) та серверну (логіка та база даних) (рис. 1.4). Взаємодія відбувається за принципом запит-відповідь, що робить систему більш масштабованою та підтримує багатокористувацьку роботу.

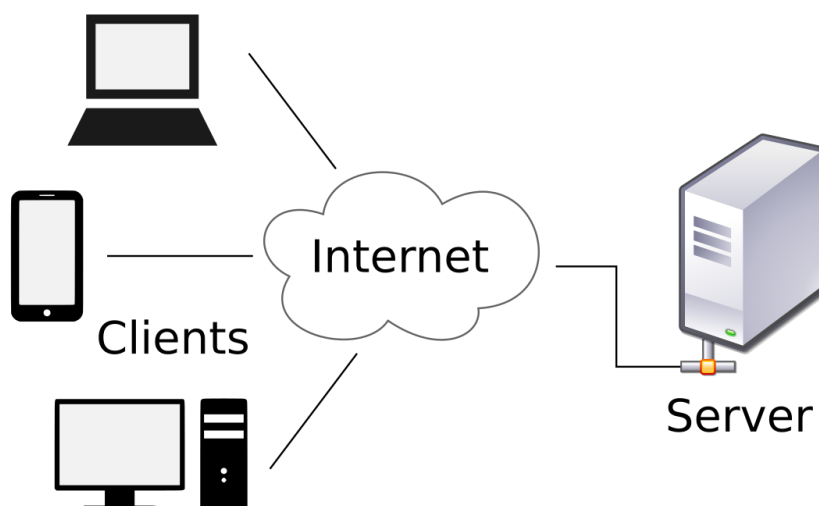


Рис. 1.4 – Клієнт-серверна архітектура

Модель-Вид-Контролер [7] (Model-View-Controller, MVC) архітектурний/дизайнерський шаблон (рис. 1.5), який розділяє додаток на три основні логічні компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Кожен архітектурний компонент призначений для обробки певних аспектів розробки додатка. Він ізолює бізнес-логіку від презентаційного шару. Сьогодні MVC є одним із найбільш часто використовуваних галузевих стандартів у веб-розробці для створення масштабованих та розширюваних проєктів. Також його використовують для проєктування мобільних додатків.

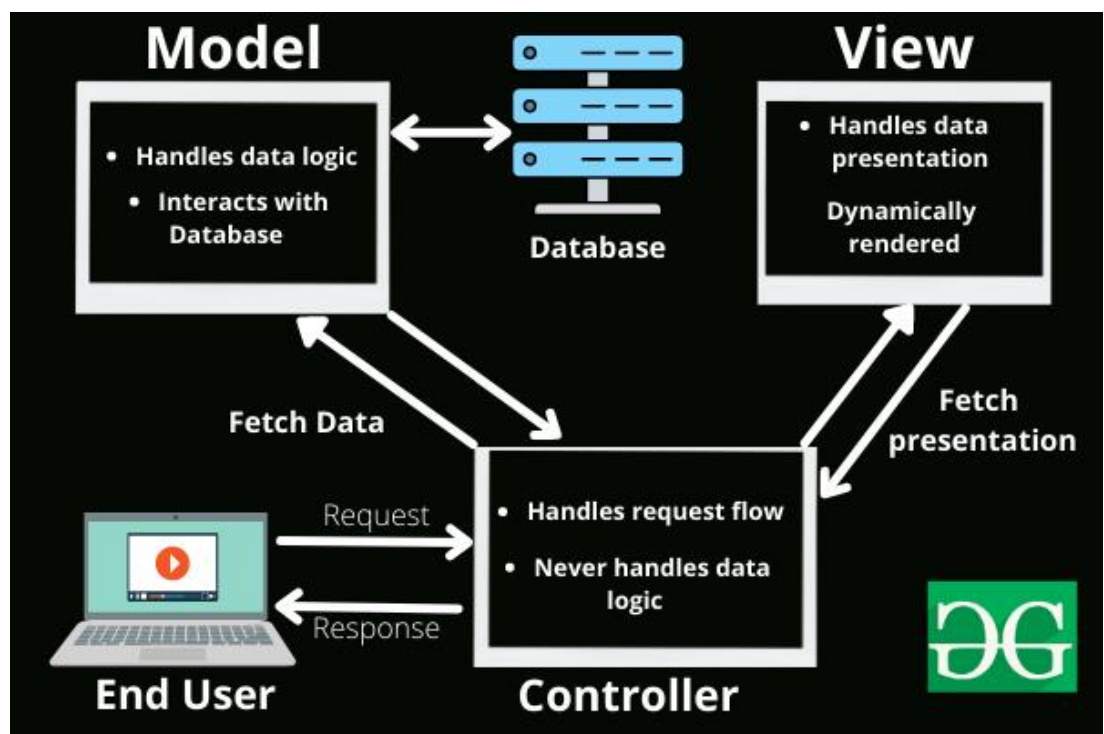


Рис. 1.5 – Архітектура Модель-Вид-Контролер (MVC)

Рис. 1.6 ілюструє архітектуру мікросервісів [8]. Виглядає вона наступним чином: користувач взаємодіє з додатком через інтерфейс користувача (App), який надсилає запити до незалежних мікросервісів. Кожен мікросервіс виконує окрему функцію додатку і має власну базу даних для автономного зберігання та керування даними. Мікросервіси обробляють отримані запити, працюють зі своїми базами даних і повертають результати через інтерфейс користувача назад

до користувача. Така архітектура забезпечує високу гнучкість, дозволяючи легко масштабувати та обслуговувати систему. Дана структура є надмірно ускладненою для обраної тематики проєкту.

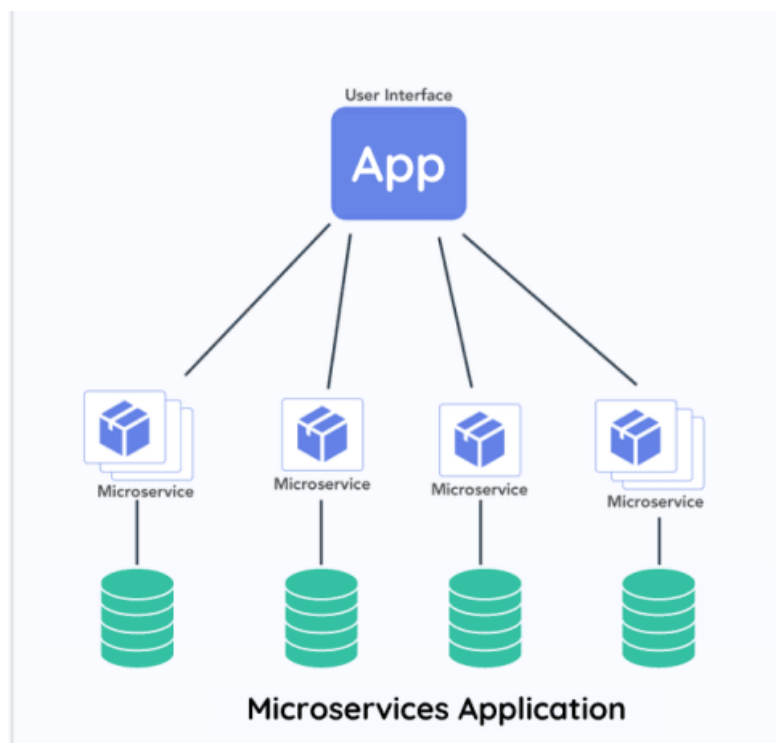


Рис. 1.6 – Мікросервісна архітектура

Опис архітектури, що буде реалізована:

Система для жеребкування турнірів буде базуватися на архітектурному патерні Модель-Вид-Контролер (MVC). Цей патерн обрано через його гнучкість, простоту у розробці та підтримці, а також здатність до швидкого реагування на зміни у вимогах користувачів. Також важливим аспектом являється структурованість самої архітектури, вона допомагає розділити логіку додатку на окремі компоненти, що значно спрощують орієнтованість розробника та обслуговування проєкту. Також важливо відмітити той факт, що одні компоненти не впливатимуть на інші. Іншими словами, логіка не є одним цілим із відображенням самого проєкту, бо моделі відповідають за бізнес логіку, в той час коли View відповідають за презентацію даних користувачеві. Крім того, під час навчання в університеті цей патерн розглядався найчастіше.

Зважаючи на перелічені фактори, для системи було обрано MVC архітектуру.

1.4. Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

Визначення інструментарію для створення програмного забезпечення є ключовим кроком у процесі його розробки. Це впливає на кінцеву якість програми, її ефективність та швидкість виконання задуманих функцій.

Оскільки планується, що система буде вебдодатком, має сенс розподілити технології для клієнтської (фронтенд) та серверної (бекенд) частин.

Для розробки бекенду було вибрано мову програмування C#. C# – це сучасна об'єктно-орієнтована мова з строгою типізацією, яка дозволяє створювати різноманітні типи програм, що характеризуються безпекою та надійністю у середовищі .NET [9].

C# є мовою програмування, яка базується на принципах об'єктної та компонентної орієнтації. Вона включає в себе конструкції, що дозволяють ефективно реалізовувати ці концепції, що робить C# ідеальною для розробки та використання програмних компонентів. З часу свого започаткування C# постійно розвивалася, отримуючи нові можливості для підтримки сучасних вимог до програмування та найкращих практик у цій галузі. В основі своїй, C# залишається мовою з сильною об'єктно-орієнтованою парадигмою, де ви створюєте типи даних та визначаєте їхню поведінку.

Платформа .NET для розробки вебдодатків пропонує ASP.NET Core. ASP.NET Core – це фреймворк з відкритим кодом, який підтримує різні операційні системи і забезпечує високу продуктивність. Він призначений для створення сучасних вебдодатків та сервісів, які ефективно працюють у хмарному середовищі та мають з'єднання з Інтернетом [9].

Вона дозволяє:

- забезпечувати безпечне виконання додатків;
- мати високу продуктивність, тобто платформа оптимізована для швидкої роботи;
- автоматично керувати пам'яттю, завдяки збирачу сміття (garbage collector);

- працювати на різних операційних системах (кросплатформенність);
- працювати в єдиній та уніфікованій платформі: створює різноманітні додатки для Windows середовищ;
- покращити продуктивність, можливості повторного використання коду та його підтримку.

Завдання, що знаходиться під дослідженням, містить дані з визначеною структурою та численними взаємозв'язками. Враховуючи це, було вирішено використовувати реляційну базу даних для зберігання інформації. Оскільки робота проводиться з технологічним стеком Microsoft, вибір СКБД припав на SQL Server [10].

Дана СКБД є частиною .NET екосистеми, та має ряд переваг серед інших конкурентів:

- неперевершене відновлення: MS SQL Server повністю усвідомлює важливість ваших даних. Тому MS SQL Server містить багато складних функцій, які дозволяють відновлювати та відновлювати дані, які були втрачені або пошкоджені. Хоча деякі окремі записи не можуть бути відновлені, все ж таки, можливо відновити повну базу даних за допомогою деяких передових інструментів відновлення, що містяться у базі даних MS SQL Server;
- надійність та безпека: MS SQL Server вважається одним з найбезпечніших серверів баз даних, оскільки використовує складні алгоритми шифрування, які роблять практично неможливим проникнення через захисні шари, встановлені користувачем. MS SQL Server не є сервером баз даних з відкритим кодом, що зменшує ризик атак на сервер бази даних;
- підтримка складних запитів та транзакцій: SQL Server підтримує складні аналітичні запити та транзакції, що дозволяє реалізовувати складну бізнес-логіку.

У середовищі, де використовується об'єктно-орієнтований підхід, для роботи з базами даних ефективно застосовувати ORM систему. Вона дозволяє

конвертувати об'єкти програмних класів у записи баз даних, створюючи таким чином «віртуальну об'єктну базу», яка інтегрується з мовою програмування.

Dapper є легковагим мікро-ORM, який забезпечує відмінну продуктивність та контроль над SQL запитами, ідеально підходить для додатків, яким потрібна висока продуктивність та контроль за генерованим SQL. Entity Framework пропонує більш повне рішення ORM з розширеними можливостями та функціоналом, що покращує продуктивність, автоматизуючи багато взаємодій з базою даних, і краще підходить для проєктів, які працюють з кількома джерелами даних.

Entity Framework Core [11] — це легка, розширювана та кросплатформна версія популярної технології доступу до даних Entity Framework, яка має відкритий код.

EF Core може виступати в ролі об'єктно-реляційного маппера (ORM), що:

- дозволяє розробникам .NET працювати з базами даних, використовуючи об'єкти .NET;
- усуває необхідність писати більшість коду доступу до даних, який зазвичай доводиться створювати вручну.

Кожен проєкт потребує правильної та логічної валідації даних. Інструмент за замовчуванням від ASP.NET Core на основі атрибутів гарно впорується з даною задачею.

Для створення вебінтерфейсу обрано JavaScript, яка є однією з найпопулярніших та потужних мов програмування для розробки клієнтської частини вебдодатків, а також мовою, що дозволяє створювати динамічні вебсторінки, що в свою чергу, пасує проєкту та заданій задачі. Для стилізації та верстки використовуватимуться HTML та CSS.

Висновки до першого розділу

Для кваліфікаційної роботи бакалавра була поставлена задача. Визначено основні модулі, необхідні для забезпечення зручності використання вебдодатка.

Проведено дослідження існуючих аналогів, виявлено їхні переваги та недоліки. Створено зведену таблицю з порівнянням цих аналогів та власної розробки для кваліфікаційної роботи бакалавра.

Порівняно можливі архітектурні рішення для додатка, визначено їхні плюси та мінуси для різних типів програмних систем. Обрано архітектуру, яка найкраще підходить для цього застосунку.

Вибрано технології для реалізації бекенду та фронтенду відповідно до вимог проекту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ЖЕРЕБКУВАННЯ ТУРНІРІВ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Система жеребкування турнірів є веборієнтованою, тому взаємодія із нею лежатиме через браузер. Використовуючи сайт, користувачі матимуть змогу долучатись до системи. Тому, якісно проаналізувавши предметну область, було виділено два види акторів: гості, можливості яких охоплюватимуть простий перегляд інформації щодо створення турнірів та учасники, які матимуть змогу створювати, видаляти та керувати турнірами та матчами, обирати методи жеребкування, які в свою чергу, даватимуть змогу якісно та чесно виконувати саме жеребкування. Далі представлено наочне відображення цього у діаграмі варіантів використання (рис. 2.1) та її опис (табл. 2.1 – 2.5).

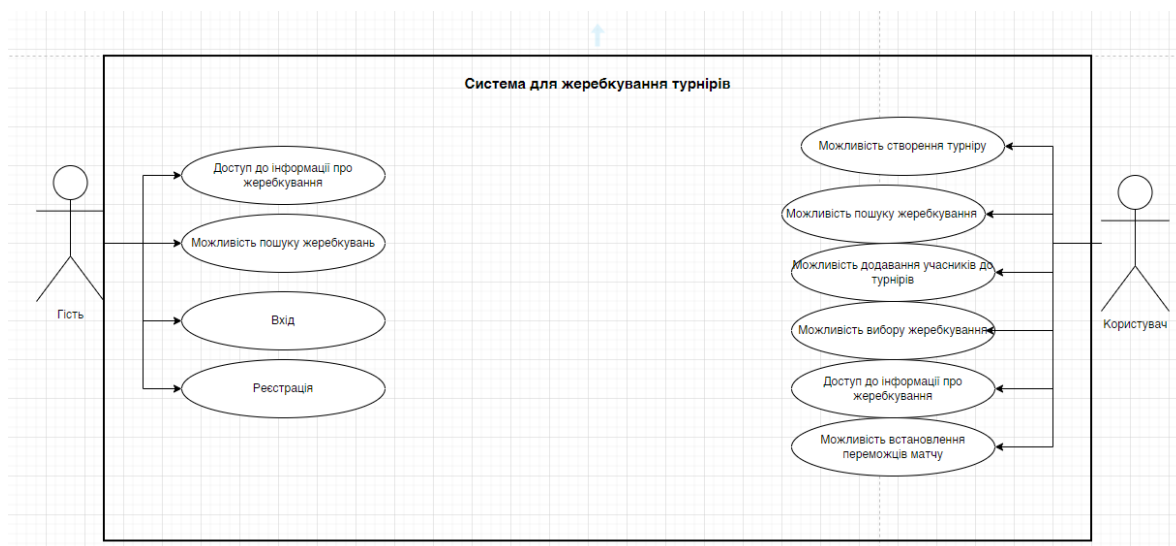


Рис. 2.1 – Діаграма варіантів використання

Таблиця 2.1

Опис прецеденту «Можливість створення турніру»

Назва варіанту використання	Можливість створення турніру
Основні актори	Користувач
Короткий опис	Користувач може створювати турніри та задавати різні параметри

Таблиця 2.2

Опис прецеденту «Можливість додавання учасників до турніру»

Назва варіанту використання	Можливість додавання учасників до турніру
Основні актори	Користувач
Короткий опис	Після створення турніру, користувач може додати n-ну кількість користувачів до турніру

Таблиця 2.3

Опис прецеденту «Можливість вибору жеребкування»

Назва варіанту використання	Можливість вибору жеребкування
Основні актори	Користувач
Короткий опис	Користувач під час створення турніру може вибрати метод для жеребкування в турнірі [12]

Таблиця 2.4

Опис прецеденту «Можливість встановлення переможців матчу»

Назва варіанту використання	Можливість встановлення переможців матчу
Основні актори	Користувач
Короткий опис	Користувач має змогу в створеному турнірі встановлювати переможців матчів

Таблиця 2.5

Опис прецеденту «Доступ до інформації про жеребкування»

Назва варіанту використання	Доступ до інформації про жеребкування
Основні актори	Користувач

Короткий опис	Користувач може переглядати інформацію щодо існуючих методів жеребкування
---------------	---

Зібравши варіанти використання для системи жеребкування турнірів, було виділено основні вектори, за якими має бути спрямовано рух для створення додатку. Враховуючи вищезгадані дані, було складено вимоги до застосунку, що розробляється.

Вимоги користувачів

1. Зручний доступ до інформації про жеребкування та результати.
2. Можливість пошуку жеребкувань.
3. Зручність використання інтерфейсу для перегляду та участі в жеребкуванні.
4. Можливість створення жеребкувань.
5. Можливість видалення жеребкувань.
6. Можливість редагування жеребкувань.

Вимоги гостей:

1. Лаконічний інтерфейс.
2. Доступний опис методів жеребкування.
3. Реєстрація.
4. Вхід в систему.

Функціональні вимоги:

1. В системі повинна бути доступною можливість автентифікації та авторизації користувачів. Відповідно, має бути реалізована можливість реєстрації вищезгаданих.
2. Створення зрозумілого інтефейсу з метою досягнення доступності задля нових користувачів системи, а також гостей.
3. Можливість користувачам виконувати менеджмент створених турнірів.
4. Автоматизоване проведення жеребкування за заданими правилами.

Не функціональні вимоги:

					ІПЗ.КР.Б-121-24-ПЗ	
						26

1. Сприйняття

- час, потрібний для освоєння веборієнтованого додатку – 5-15 хвилин;
- час відповіді системи для звичайних запитів не повинен перевищувати 5 секунд, а для більш складних запитів – 10 секунд;
- інтерфейс представлення вебдодатку повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки.

2. Надійність

- доступність;
- середній час безперервної роботи – 45 робочих днів;
- максимальна норма помилок та дефектів в роботі системи – 1 помилка на 100 запитів користувача.

3. Продуктивність: система повинна підтримувати мінімум 15 одночасно працюючих користувачів, пов'язаних спільною БД.

Зазвичай програмні системи розробляються з використанням об'єктно-орієнтованого програмування. Це стало неофіційним стандартом, тому рекомендується створити об'єктно-орієнтовану модель системи. Для цього найкраще підходить діаграма класів. Після аналізу функціональних вимог до додатку було виділено основні доменні сутності (рис. 2.2).

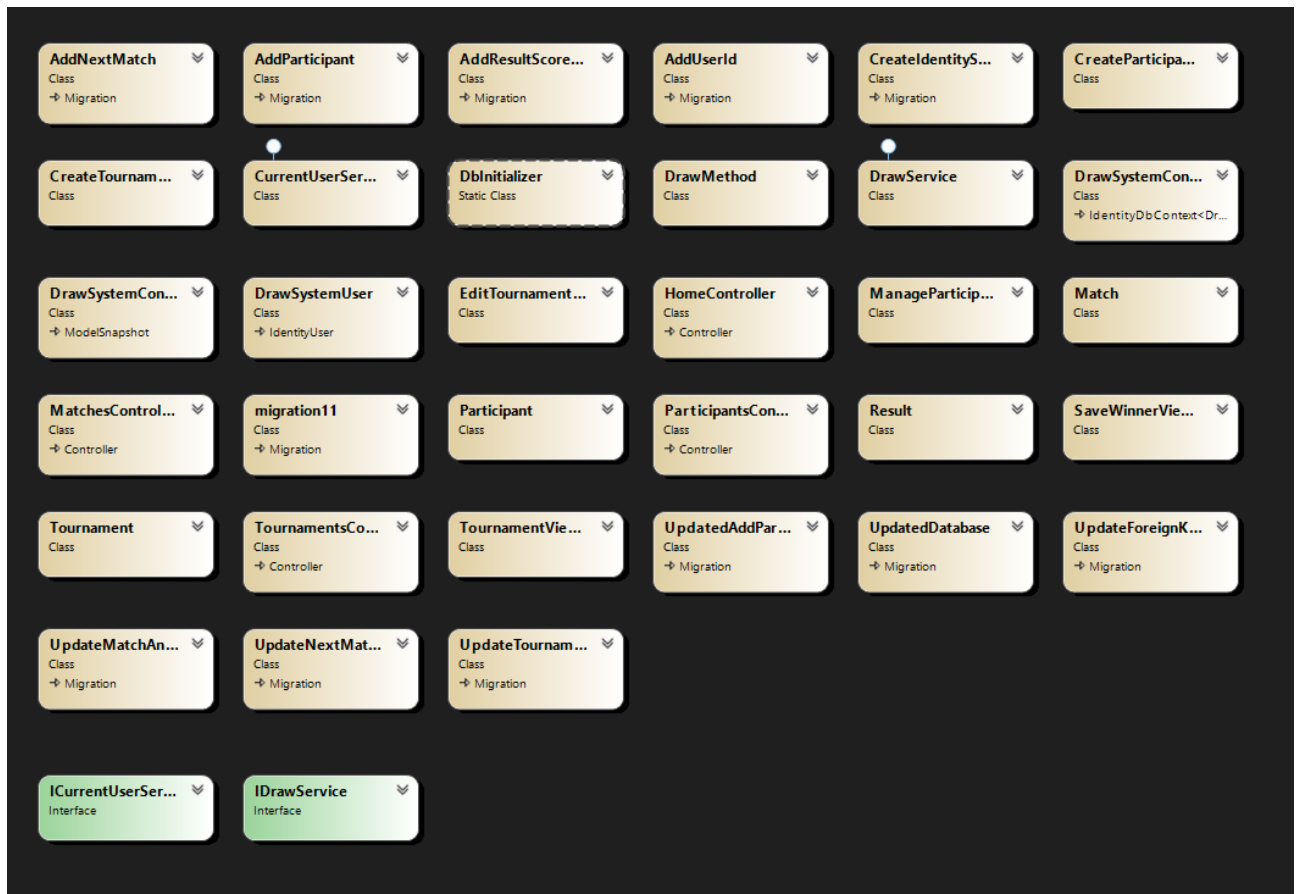


Рис. 2.2 – Діаграма класів

На діаграмі відображено наступні класи:

- CreateParticipant: клас для створення нового учасника;
- CreateTournament: клас для створення нового турніру;
- CurrentUserService: клас для обслуговування поточного користувача;
- DbInitializer: статичний клас для ініціалізації бази даних;
- DrawMethod: клас, який визначає метод жеребкування;
- DrawService: клас для обслуговування процесу жеребкування;
- DrawSystemContextModelSnapshot: клас для створення знімку моделі контексту системи жеребкування;
- DrawSystemUser: клас користувача системи жеребкування;
- EditTournament: клас для редагування турніру;
- HomeController: контролер для головної сторінки;
- ManageParticipants: клас для управління учасниками;
- Match: клас для представлення матчу;

- MatchesController: контролер для управління матчами;
- Migration11: клас для виконання міграцій бази даних;
- Participant: клас для представлення учасника;
- ParticipantsController: контролер для управління учасниками;
- Result: клас для представлення результату матчу;
- SaveWinnerViewModel: клас для збереження даних про переможця;
- Tournament: клас сутності турніру;
- TournamentsController: контролер для управління турнірами;
- TournamentViewModel: клас для представлення даних турніру;
- UpdateAddParticipant: клас для оновлення даних про додавання учасника;
- UpdateDatabase: клас для оновлення бази даних;
- UpdateForeignKey: клас для оновлення зовнішніх ключів;
- UpdateMatchAndTournament: клас для оновлення даних про матчі та турніри;
- UpdateNextMatch: клас для оновлення інформації про наступний матч;
- UpdateTournament: клас для оновлення даних про турнір;
- ICurrentUserService: інтерфейс для обслуговування поточного користувача;
- IDrawService: інтерфейс для обслуговування процесу жеребкування.

2.2. Розробка бази даних системи

Провівши ретельний аналіз, було виявлено, що для збереження даних найкраще пасує реляційна СКБД Microsoft SQL Server.

База даних системи для жеребкування складається із наступних таблиць, наведених на діаграмі (рис. 2.3).

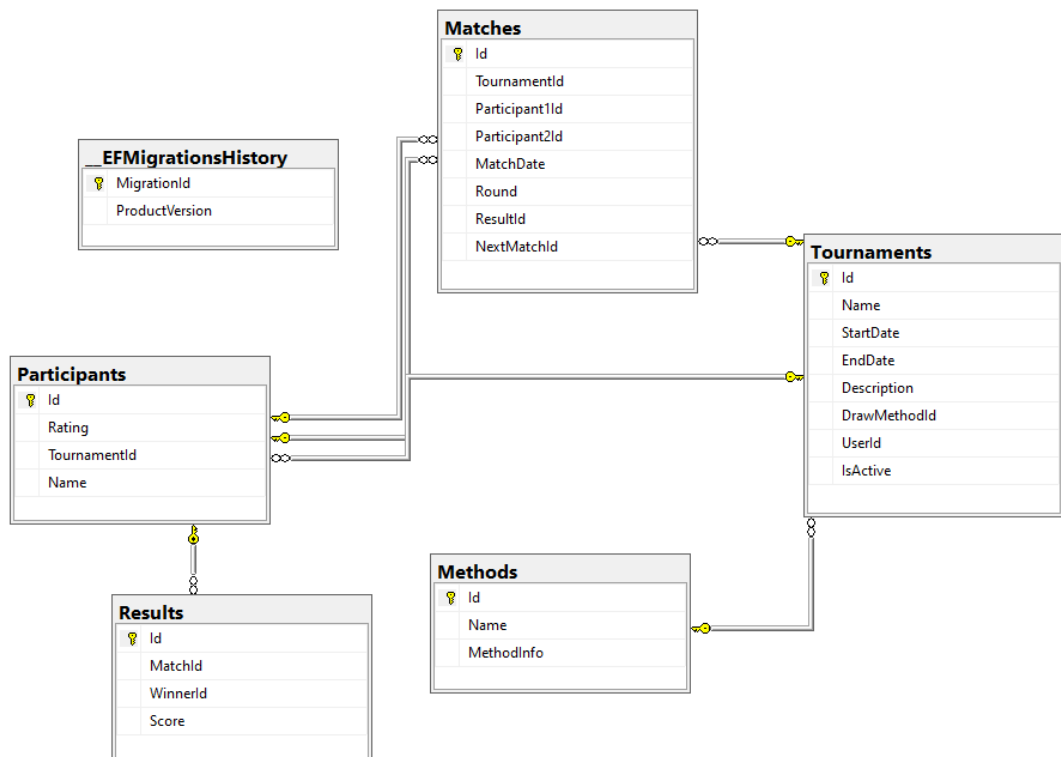


Рис. 2.3 – Діаграма бази даних

В наступному списку перелічені таблиці, присутні в базі даних:

- __EFMigrationsHistory – таблиця, створена Entity Framework Core для трекінгу міграцій;
- Matches – таблиця матчів для турніру;
- Participants – таблиця учасників турніру;
- Tournaments – таблиця турнірів;
- Results – таблиця результатів проведення матчу;
- Methods – таблиця методів для жеребкування турнірів.

Далі наведено детальний опис полів кожної таблиці (табл. 2.6 – 2.11).

Таблиця 2.6

Опис полів таблиці «__EFMigrationsHistory»

Назва	Тип даних	PK	FK	Опис
MigrationId	nvarchar(150) not null	+	-	Ідентифікаційний номер міграції
ProductVersion	nvarchar(32), not null	-	-	Версія Entity Framework Core

Таблиця 2.7

Опис полів таблиці «Participants»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер користувача
Name	nvarchar(450), not null	-	-	Ім'я користувача
TournamentId	int, not null	-	Tournament.Id	Ідентифікаційний номер турніру
Raiting	int, not null	-	-	Рейтинг користувача (числовий)

Таблиця 2.8

Опис полів таблиці «Results»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер результату
MatchId	int, not null	-	Match.Id	Ідентифікаційний номер матчу
WinnerId	int, not null	-	Participant.Id	Ідентифікаційний номер переможця
Score	int, not null	-	-	Рахунок

Таблиця 2.9

Опис полів таблиці «Matches»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер матчу
TournamentId	int not null	-	Tournament.Id	Ідентифікаційний номер турніру

Продовження таблиці 2.9

Participant1Id	int, null	-	Participant.Id	Ідентифікаційний номер першого учасника
Participant2Id	int, null	-	Participant.Id	Ідентифікаційний номер другого учасника
Round	int, null	-	-	Номер раунду
ResultId	int, null	-	Result.Id	Ідентифікаційний номер результату
NextMatchId	int, null	-	-	Ідентифікаційний номер наступного матчу

Таблиця 2.10

Опис полів таблиці «Tournaments»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер турнірів
Name	nvarchar(250), not null	-	-	Назва турніру
StartDate	datetime not null	-	-	Початкова дата турніру
EndDate	datetime, not null	-	-	Кінцева дата турніру
Description	nvarchar(500) not null	-	-	Опис турніру
DrawMethodId	Int, not null	-	DrawMethod.I d	Ідентифікаційний номер методу жеребкування
UserId	Int, not null	-	User.Id	Ідентифікаційний номер користувача
IsActive	boolean, not null	-	-	Перевірка на активність турніру

Опис полів таблиці «Methods»

Назва	Тип даних	PK	FK	Опис
Id	Int, not null	+	-	Ідентифікаційний номер методу жеребкування
Name	nvarchar(200) not null	-	-	Назва методу
MethodInfo	nvarchar(500), not null	-	-	Опис методу для жеребкування

2.3 Проєктування та реалізація алгоритмів роботи системи

Безумовно, використання діаграм та схем є важливим для наочного опису алгоритмів та станів системи. Тому, було складено діаграму активності створення жеребкування після авторизації (рис. 2.4). Вона складається із доріжок користувача і системи. Спочатку користувача має здійснити реєстрацію/взід до акаунту. Якщо реєстрація успішна, то подача він має доступ до акаунту. Далі, користувач може створити жеребкування, ввівши коректно дані в поля. В іншому випадку, користувачу потрібно буде перепройти стадію заповнення даних для жеребкування турніру.

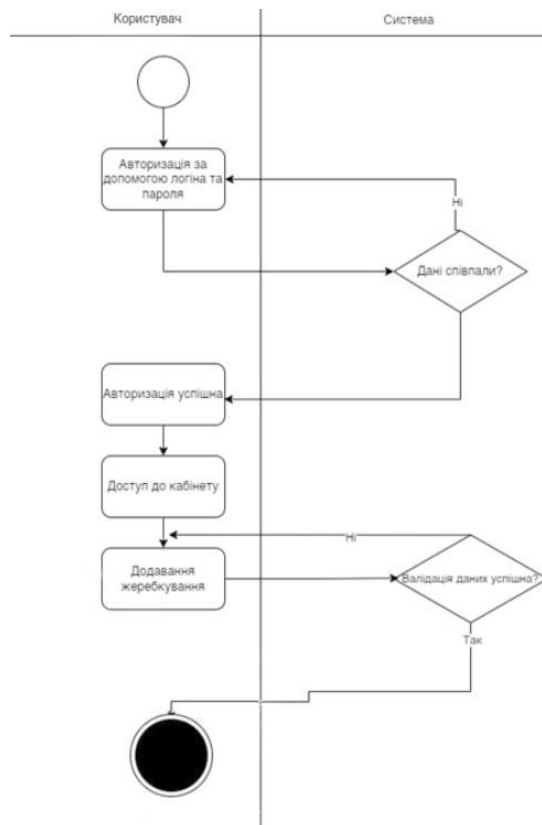


Рис. 2.4 – Діаграма активності створення жеребкування

Для зображення алгоритму створення турніру було складено діаграму послідовності (рис. 2.5). Користувач звертається до TournamentController для створення нового турніру. TournamentController викликає метод створення турніру в TournamentService. TournamentService створює турнір і повертає підтвердження до TournamentController. TournamentController інформує Користувача, що турнір створений. Користувач додає учасників через ParticipantController. ParticipantController викликає метод додавання учасника в ParticipantService. ParticipantService додає учасника і повертає підтвердження до ParticipantController. ParticipantController інформує Користувача, що учасник доданий. Після додавання всіх учасників, Користувач ініціює жеребкування через DrawService. DrawService проводить жеребкування і повертає результати.

TournamentController отримує результати жеребкування від DrawService і зберігає їх у турнірі. TournamentController інформує Користувача про успішне проведення жеребкування і результати.

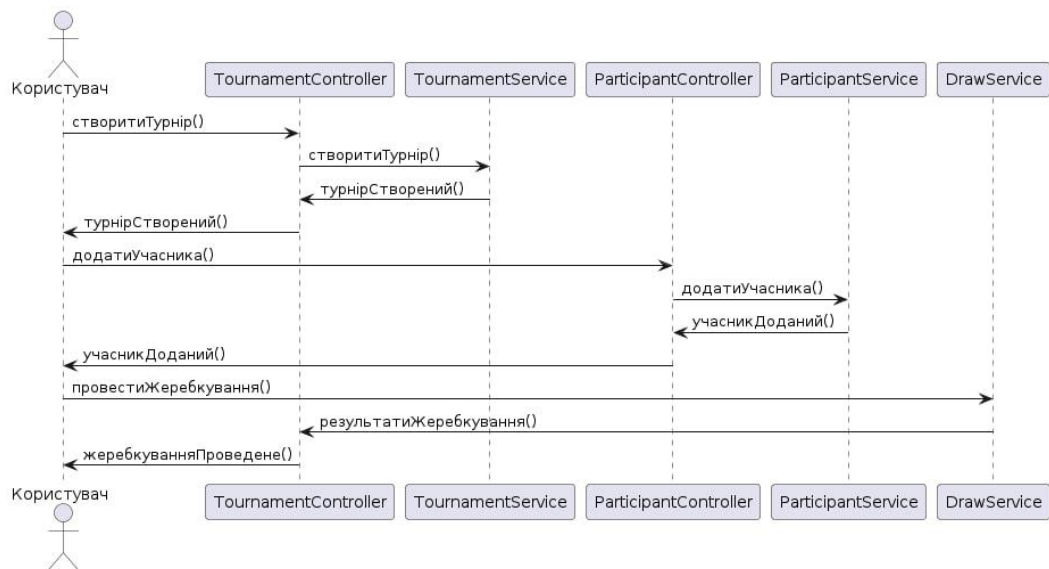


Рис. 2.5 – Діаграма послідовності створення жеребкування

2.4. Реалізація функціоналу системи жеребкування турнірів

Розгляньмо створений функціонал для додавання нового турніру. Наведено відповідний лістинг методу класу контролера (рис. 2.6). Він є обробником POST HTTP запиту на стороні серверу. Атрибут [13] `ValidateAntiForgeryToken` відповідає за забезпечення, що кожен POST-запит є справжнім і захищає додаток від небажаних дій з боку злоумисників.

```

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(CreateTournamentViewModel model)
{
    if (!ModelState.IsValid)
    {
        var methods = await _context.Methods.ToListAsync();
        ViewBag.Methods = methods;
        return View(model);
    }

    var tournament = new Tournament
    {
        Name = model.Name,
        Description = model.Description,
        StartDate = model.StartDate,
        EndDate = model.EndDate,
        DrawMethodId = model.DrawMethodId,
        UserId = _currentUserService.GetUserId(),
    };

    await _context.AddAsync(tournament);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

Рис. 2.6 – Метод додавання турніру класу TournamentsController

Після створення турніру його можна побачити в списку всіх створених турнірів, закріплених за користувачем, створеним за допомогою Microsoft Identity [14] (рис. 2.7).

```

[HttpGet]
1 reference
public async Task<IActionResult> Index()
{
    var tournaments = await _context.Tournaments
        .Include(t => t.Matches)
        .ThenInclude(m => m.Result)
        .Include(t => t.Participants)
        .Include(t => t.DrawMethod)
        .ToListAsync();

    var a = tournaments.Where(t => t.Matches.Any(m => m.Result is not null)).ToList();

    var viewModels = tournaments.Select(t => new TournamentViewModel
    {
        Tournament = t,
        IsParticipantsManageable = !t.Matches.Any(m => !m.IsBye() && m.Result is not null)
    }).ToList();

    return View(viewModels);
}

```

Рис. 2.7 – Перегляд усіх турнірів, створених користувачем

Для зручності, там відображені всі операції, які можна зробити з даним турніром, такі як видалення, перегляд, створення сітки та редагування (рис. 2.8)

```

@using DrawSystem.Models.Entities
@model ManageParticipantsViewModel
<body>
<form method="post" asp-controller="Tournaments" asp-action="ManageParticipants" asp-route-id="@Model.Id">
<div class="mt-3">
<h3>Натисніть аби додати учасника: </h3>
</div>
<div class="mb-2">
<button type="button" class="btn btn-secondary" id="addParticipantBtn"> + </button>
</div>
<div id="participantsContainer" data-draw-method="@ViewBag.DrawMethodName">
@for (var i = 0; i < Model.Participants.Count; i++)
{
<div class="row mb-2">
<div class="col-auto">
<input class="form-control" name="participants[@i].name" value="@Model.Participants[i].Name" />
@if (ViewBag.DrawMethodName == Constants.DrawMethods.SingleEliminationSeeded)
{
<input class="form-control" type="number" min="0" name="participants[@i].rating" value="@Model.Participants[i].Rating" />
}
<input style="display: none" name="participants[@i].id" value="@Model.Participants[i].Id" />
</div>
<div class="col-auto">
<button type="button" class="remove-participant-btn btn btn-danger" >Видалити</button>
</div>
</div>
}
</div>
<div class="mt-3">
<button type="submit" class="btn btn-primary">Manage Participants </button>
</div>
</form>
<script src="js/manageParticipants.js"></script>
</body>

```

Рис. 2.8 – Вигляд усіх створених турнірів

Метод ManageParticipants це метод, який надає змогу ввести до бази даних учасників заданого матчу та викликати інші методи для створення сіток та заповнення них вищезгаданими учасниками (рис. 2.9).

```

[HttpPost]
0 references
public async Task<IActionResult> ManageParticipants(ManageParticipantsViewModel model, int id)
{
    var tournament = await _context.Tournaments
        .Include(t => t.Participants)
        .Include(t => t.Matches)
        .FirstOrDefaultAsync(t => t.Id == id);

    if (tournament is not null)
    {
        tournament.Participants.Clear();
        model.Participants.ForEach(p => p.TournamentId = id);
        tournament.Participants.AddRange(model.Participants);
        await _context.SaveChangesAsync();

        tournament.Matches.Clear();
        await GenerateMatches(tournament);
    }

    return RedirectToAction("Index", "Tournaments");
}

```

Рис. 2.9 – Метод керування учасниками

Важливо насамперед приділити увагу до динамічного додавання та видалення учасників за допомоги Javascript [15] в методі ManageParticipants, зазначеного

на рис. 2.10.

```
const SINGLE_ELIMINATION_SEEDED = "Single Elimination - Seeded";

document.addEventListener('DOMContentLoaded', function () {
    const addParticipantBtn = document.getElementById('addParticipantBtn');
    const participantsContainer = document.getElementById('participantsContainer');
    const drawMethod = participantsContainer.dataset.drawMethod;

    document.querySelectorAll(".remove-participant-btn")
        .forEach(btn => btn.addEventListener('click', removeParticipantHandler));

    let participantCount = participantsContainer.children.length;

    addParticipantBtn.addEventListener('click', function () {
        const inputContainer = document.createElement('div');
        inputContainer.className = 'col-auto';

        const participantInput = document.createElement('input');
        participantInput.type = 'text';
        participantInput.placeholder = 'Введіть ім'я учасника';
        participantInput.className = 'form-control';
        participantInput.name = `participants[${participantCount}].name`;
        inputContainer.appendChild(participantInput);

        if (drawMethod === SINGLE_ELIMINATION_SEEDED) {
            const ratingInput = document.createElement('input');
            ratingInput.type = 'number';
            ratingInput.min = 0;
            ratingInput.placeholder = 'Введіть рейтинг учасника';
            ratingInput.className = 'form-control';
            ratingInput.name = `participants[${participantCount}].rating`;
            inputContainer.appendChild(ratingInput);
        }

        const buttonContainer = document.createElement('div');
        buttonContainer.className = 'col-auto';

        const removeBtn = document.createElement('button');
        removeBtn.className = 'btn btn-danger';
        removeBtn.textContent = 'Видалити';
        removeBtn.addEventListener('click', removeParticipantHandler);

        buttonContainer.appendChild(removeBtn);

        const participantDiv = document.createElement('div');
        participantDiv.className = 'row mb-2';
        participantDiv.appendChild(inputContainer);
        participantDiv.appendChild(buttonContainer);

        participantsContainer.appendChild(participantDiv);
        participantCount++;
    });
});
```

Рис. 2.10 – Динамічне додавання/видаляння учасників для турніру

А сам код у вигляді ManageParticipants.cshtml (рис. 2.11) демонструє створення вебсторінки для керування учасниками турніру, використовуючи Razor Pages в ASP.NET Core. Основні елементи цього коду включають використання моделі ManageParticipantsViewModel та ентиті DrawSystem.Models.Entities. Вебсторінка містить форму, яка дозволяє додавати, редагувати та видаляти учасників турніру. Форма відправляє дані до дії ManageParticipants в контролері Tournaments.

```

@using DrawSystem.Models.Entities
@model ManageParticipantsViewModel
<body>
    <form method="post" asp-controller="Tournaments" asp-action="ManageParticipants" asp-route-id="@Model.Id">
        <div class="mt-3">
            <h3>Написати або додати учасника: </h3>
        </div>
        <div class="mb-2">
            <button type="button" class="btn btn-secondary" id="addParticipantBtn"> + </button>
        </div>
        <div id="participantsContainer" data-draw-method="@ViewBag.DrawMethodName">
            @for (var i = 0; i < Model.Participants.Count; i++)
            {
                <div class="row mb-2">
                    <div class="col-auto">
                        <input class="form-control" name="participants[@i].name" value="@Model.Participants[i].Name" />
                        @if (ViewBag.DrawMethodName == Constants.DrawMethods.SingleEliminationSeeded)
                        {
                            <input class="form-control" type="number" min="0" name="participants[@i].rating" value="@Model.Participants[i].Rating" />
                        }
                    </div>
                    <div class="col-auto">
                        <input style="display: none" name="participants[@i].id" value="@Model.Participants[i].Id" />
                        <div class="col-auto">
                            <button type="button" class="remove-participant-btn btn btn-danger">Видалити</button>
                        </div>
                    </div>
                </div>
            }
        </div>
        <div class="mt-3">
            <button type="submit" class="btn btn-primary">Manage Participants </button>
        </div>
    </form>
    <script src="js/manageParticipants.js"></script>
</body>

```

Рис. 2.11 – Вигляд ManageParticipants

Звернувши увагу до попереднього методу, можна помітити, що там викликається інший метод `GenerateMatches()`, який, в свою чергу, викликає метод `GenerateBrackets`. Розгляньома детальніше останній метод (рис. 2.12). Змінна `totalRounds` відповідає за кількість раундів, за алгоритмом, проводяться вираховування логарифму від двох за основою кількості учасників. Це робиться з метою вираховування кількості раундів, і насправді, є доволі логічним. Таким чином, якщо кількість учасників – 16, то кількість проведених раундів має бути 4, бо $2^4 = 16$. Але якщо кількість учасників не є степенем двійки, то для цього присутній метод `Math.Ceiling`. Таким чином, виконується округлення то більшого значення. Наприклад, кількість учасників – 7, значить, $\log_2 7 \sim 2.8$, але кількість раундів буде 3.

```

1 reference
public List<Match> GenerateBracket(int tournamentId, int numberOfParticipants)
{
    var matches = new List<Match>();
    var totalRounds = (int)Math.Ceiling(Math.Log2(numberOfParticipants));
    var currentRoundMatches = new List<Match>();

    for (int round = 1; round <= totalRounds; round++)
    {
        int matchesInRound = (int)Math.Pow(2, totalRounds - round);
        for (int matchNumber = 0; matchNumber < matchesInRound; matchNumber++)
        {
            var match = new Match
            {
                TournamentId = tournamentId,
                Round = round
            };
            matches.Add(match);
        }

        currentRoundMatches = matches.Where(m => m.Round == round).ToList();
    }
    return matches;
}

```

Рис. 2.12 – Метод GenerateBracket, який створює сітку для жеребкування

За реалізацію наповнення сіток учасниками відповідає метод GenerateMatches, який в собі містить виклики двох методів LinkMatches для зв'язання поточних та наступних матчів (рис. 2.13) та GetMatchesByRound для отримання матчів відповідно до раунду, якому вони належать (рис. 2.14).

```

1 reference
private void LinkMatches(List<Match> matches)
{
    var rounds = matches.Select(m => m.Round).Distinct().OrderBy(r => r).ToList();
    foreach (var round in rounds)
    {
        var currentRoundMatches = GetMatchesByRound(matches, round);
        var nextRoundMatches = GetMatchesByRound(matches, round + 1);

        if (round == rounds.Last())
        {
            break;
        }

        for (int i = 0; i < currentRoundMatches.Count; i++)
        {
            var match = currentRoundMatches[i];
            match.NextMatchId = nextRoundMatches[i / 2].Id;
        }
    }
}

```

Рис. 2.13 – Метод LinkMatches

```

3 references
private List<Match> GetMatchesByRound(List<Match> matches, int round)
{
    return matches.Where(m => m.Round == round).ToList();
}

```

Рис. 2.14 – Метод GetMatchesByRound

В самому GenerateMatches також присутній виклик важливої функції GetNumByes (рис. 2.15) для вирахування кількості проходів в матчі, коли кількість учасників не являється степенем двійки.

```

1 reference
private int GetNumByes(int numParticipants)
{
    int nextPowerOfTwo = 1;
    while (nextPowerOfTwo < numParticipants)
    {
        nextPowerOfTwo *= 2;
    }
    return nextPowerOfTwo - numParticipants;
}

```

Рис. 2.15 – Метод GetNumByes

Повертаючись до основного метода GenerateMatches (рис. 2.16), варто зазначити, що він є основним в ролі виконання логіки щодо створення самих матчів між учасниками. Спочатку система отримує список учасників і рандомізуємо його порядок. Після того, як вже вищезгаданий метод GetNumByes вирахує кількість проходів учасників, створюються матчі для довільно обраних учасників, що не матимуть пари. Згодом, проводиться підставлення усіх інших учасників в матчах 1x1. Результат зберігається в базу даних.

```

private async Task<List<Match>> GenerateMatches(Tournament tournament)
{
    var participants = tournament.Participants.OrderByDescending(p => p.Rating).ToList();
    if (tournament.DrawMethod?.Name == Constants.DrawMethods.SingleElimination)
    {
        var random = new Random();
        participants = tournament.Participants
            .OrderBy(p => random.Next())
            .ToList();
    }
    var matches = GenerateBracket(tournament.Id, participants.Count);
    await _context.Matches.AddRangeAsync(matches);
    await _context.SaveChangesAsync();

    LinkMatches(matches);

    var round = 1;
    var firstRoundMatches = GetMatchesByRound(matches, round);

    var numByes = GetNumByes(participants.Count);
    for (int i = 0; i < numByes; i++)
    {
        var participant = participants[i];
        var match = firstRoundMatches[i];
        match.Participant1Id = participant.Id;
        match.Participant2Id = null;
        match.MatchDate = tournament.StartDate;
        match.Result = new Result
        {
            WinnerId = participant.Id,
            Winner = participant,
        };

        var nextMatch = matches.FirstOrDefault(m => m.Id == match.NextMatchId);
        if (nextMatch != null)
        {
            nextMatch.Participant1Id = participant.Id;
        }
    }

    for (int i = numByes, j = numByes; i < participants.Count; i += 2, j++)
    {
        var match = firstRoundMatches[j];
        match.Participant1Id = participants[i].Id;
        match.Participant2Id = i + 1 < participants.Count ? participants[i + 1].Id : null;
        match.MatchDate = tournament.StartDate;
    }

    await _context.SaveChangesAsync();
    return matches;
}

```

Рис. 2.16 – Створення матчів

Після створення сітки потрібно визначати переможця кожного матчу, саме з цією метою в MatchesController знаходиться метод SaveWinner (рис. 2.17). Він містить атрибут HttpPost, що свідчить про те, що він працюватиме з відправками форм. Також, знаходить матч по ID з моделі у разі якщо матч не знайдено, повертає NotFound, якщо матч знайдено, оновлює або створює результат для матчу зі вказаним переможцем і зберігає його в базу даних.

```

[HttpPost]
0 references
public async Task<IActionResult> SaveWinner(SaveWinnerViewModel viewModel, int id)
{
    var match = await _context.Matches.Include(m => m.Result).FirstOrDefaultAsync(m => m.Id == viewModel.MatchId);

    if (match is null)
    {
        return NotFound();
    }

    if (match.Result is null)
    {
        match.Result = new Result { WinnerId = viewModel.WinnerId };
    }
    else
    {
        match.Result.WinnerId = viewModel.WinnerId;
    }
}

```

Рис. 2.17 – Метод SaveWinner

Слід детальніше розглянути використання методу SaveWinner. Далі, в ньому розглядається алгоритм для зчитування та оновлення даних для наступного матчу, визначення переможця через модель вигляду, та логіка щодо присвоєння відповідним учасникам місць в наступному матчі (рис. 2.18)

```
var nextMatch = await _context.Matches
    .FirstOrDefaultAsync(m => m.TournamentId == match.TournamentId && m.Id == match.NextMatchId);

nextMatch.MatchDate = DateTime.Now;
var isParticipant1 = match.Participant1Id == viewModel.WinnerId;
if (isParticipant1)
{
    if (nextMatch.Participant1Id is null || nextMatch.Participant1Id == match.Participant2Id)
    {
        nextMatch.Participant1Id = viewModel.WinnerId;
    }
    else
    {
        nextMatch.Participant2Id = viewModel.WinnerId;
    }
}
else
{
    if (nextMatch.Participant1Id is null || nextMatch.Participant1Id == match.Participant1Id)
    {
        nextMatch.Participant1Id = viewModel.WinnerId;
    }
    else
    {
        nextMatch.Participant2Id = viewModel.WinnerId;
    }
}

await _context.SaveChangesAsync();
return RedirectToAction("Index", new { Id = id });
```

Рис. 2.18 – Алгоритм присвоєння учасникам місць в наступних матчах

Висновки до другого розділу

Під час проєктування системи для проведення жеребкування було встановлено і описано можливі варіанти використання, визначено вимоги до користувачів і системи. Також була створена структура системи з орієнтацією на об'єкти, побудована діаграма класів доменних сутностей і використано кілька патернів проєктування.

Була розроблена база даних за допомогою реляційної системи управління базами даних відповідно до визначених вимог.

Також були спроектовані алгоритми системи і створені діаграми для їх ілюстрації.

У результаті був реалізований функціонал додатку, описано ряд модулів системи і наведено лістинги окремих методів та класів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СИСТЕМОЮ ЖЕРЕБКУВАНЬ

3.1. Порядок встановлення та налаштування параметрів платформи

Встановлення та налаштування параметрів системи вимагає встановлення серверного середовища, розгортання серверної та клієнтської частини, а також налаштування зв'язків між компонентами системи.

Для опису процесу розгортання було використано діаграму розгортання (рис. 3.1).

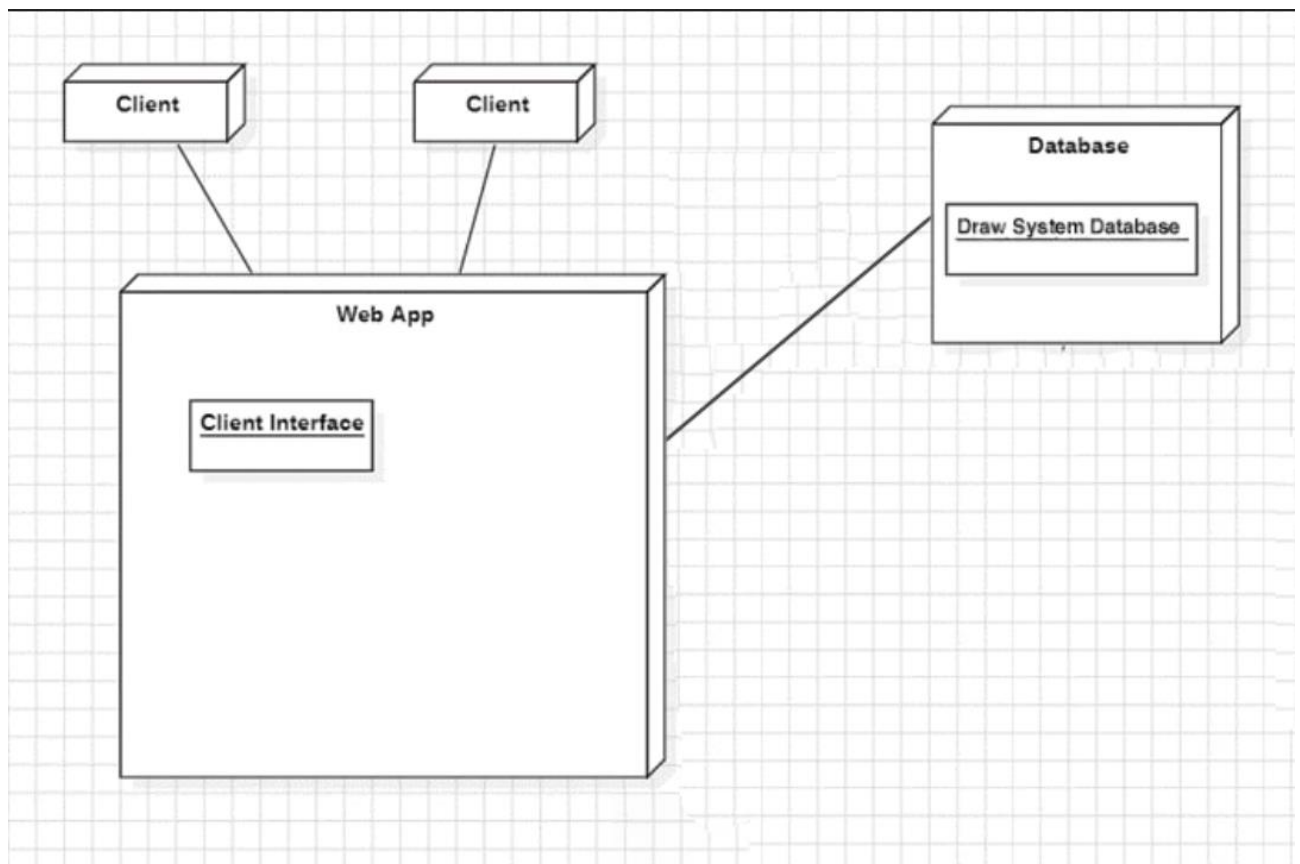


Рис. 3.1 – Діаграма розгортання

Для розгортання використовується браузер, та локальна база даних:

- в браузері виконується клієнтський код, написаний із HTML/CSS та Javascript;
- в базі даних використовуються зв'язки з клієнтською частиною задля злагодженої та швидкої взаємодії між користувачем та системою аби отримати максимально прозаїчний метод для жеребкування турніру.

3.2. Структура інтерфейсу системи жеребкування турнірів

Співпраця із додатком відбувається через вебінтерфейс. Під час заходу на вебсайт, користувач помічає перед собою домашню сторінку яка вказує на коротку інформацію щодо методів жеребкування (рис. 3.2).

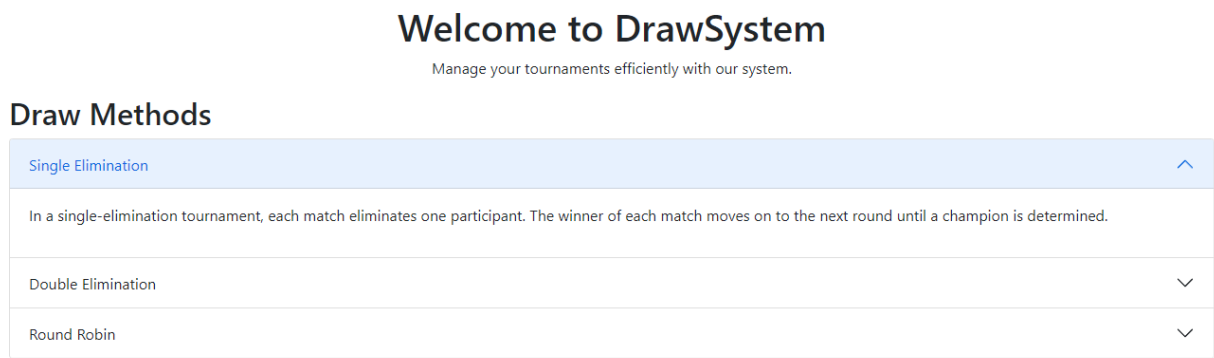


Рис. 3.2 – Домашня сторінка

Далі, гість має змогу увійти до системи (рис. 3.3).

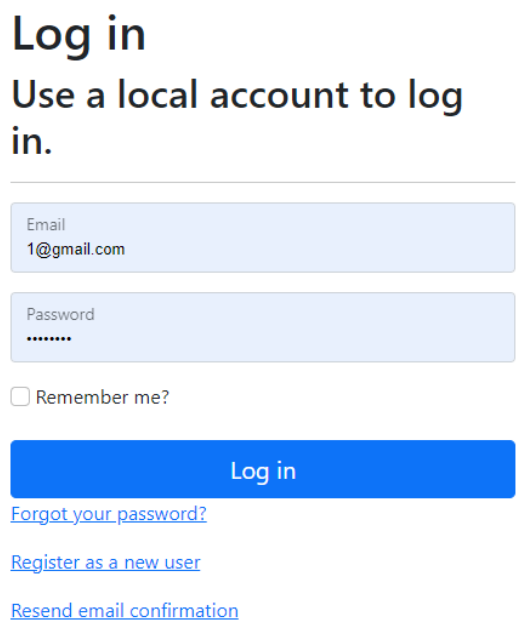


Рис. 3.3 – Сторінка логіну

Якщо у користувача немає облікового запису, він може перейти на сторінку реєстрації (рис. 3.4) по відповідному посиланню.

Register

Create a new account.

Email

Password

Confirm Password

Register

Рис. 3.4 – Сторінка реєстрації

Після входу до облікового запису користувача, у нього/неї з'являється можливість перегляду профілю (рис. 3.5). З меню облікового запису можна редагувати, додавати, видаляти інформацію, змінити пароль та навіть видалити сам профіль користувача.

Profile

Email

Password

Two-factor authentication

Personal data

Profile

Username

1@gmail.com

Phone number

Save

Рис. 3.5 – Сторінка облікового запису

Бокове меню складається із пунктів: «Профіль», «Електронна пошта», «Пароль», «Двофакторна автентифікація», «Персональні дані».

Після успішної реєстрації/входу користувач має змогу створити новий турнір (рис. 3.6)

Name

Start Date

End Date

Description

Select draw method

Create Tournament

Рис. 3.6 – Форма для створення турніру

Створивши новий турнір, користувач для успішного додавання та менеджменту жеребкування матиме заповнити його учасниками. Залежно від вибору методу для жеребкування на етапі створення турніру, вже під час додавання учасників будуть доступні різні параметри для введення (рис. 3.6). Наприклад, для Single Elimination достатньо ввести імена учасників (рис. 3.7), а в Single Elimination – Seeded Draw потрібно також вписувати і рейтинг (рис. 3.8).

Натисніть аби додати учасника:

+

Dre

Видалити

Eminem

Видалити

3

Видалити

4

Видалити

5

Видалити

6

Видалити

Manage Participants

Рис. 3.7 – Заповнення турніру учасниками (Single Elimination)

Натисніть аби додати учасника:

+

1

122

Видалити

2

333

Видалити

3

44

Видалити

4

12

Видалити

Manage Participants

Рис. 3.8 – Заповнення турніру учасниками (Single Elimination – Seeded)

Після успішного створення нового турніру та заповнення його учасниками, можна переглянути інформацію щодо самого турніру (рис. 3.9).

14	Dada	454	Single Elimination	6/20/2024	6/21/2024	Manage (8)	Matches	Edit	Delete
----	------	-----	--------------------	-----------	-----------	----------------------------	-------------------------	----------------------	------------------------

Рис. 3.9 – Результат створення турніру

Також можна змінити деякі вхідні дані щодо турніру. Аби відтворити дану операцію, варто перейти до вкладки Edit, вищезгаданому на рисунку 3.9. В самому компоненті можна замінити дані, вказані на рис. 3.10.

Name

Турнір "Веселка" ✓

Description

Турнір для веселих людей

Edit Tournament

Рис. 3.10 – Редагування інформації щодо турніру

Користувач має змогу перейти до самої сітки турніру. Зі сторінки Index, він або вона, перейшовши за посиланням Matches (рис. 3.11), прямує до створених турнірних сіток.

Турнір "Веселка" - Bracket

Round 1

Матч між 6 і Dre

Select Winner

▼

Update

Match Date: 6/20/2024

Round 2

Матч між TBD і TBD

▼

Match Date:

Матч між TBD і TBD

▼

Match Date:

Round 3

Матч між TBD і TBD

▼

Match Date:

Рис. 3.11 – Створені турнірні сітки

Самі кількості раундів та матчів була розрахована ще на моменті додавання учасників до турніру, далі користувач додаватиме до наступних матчів переможців, і, відповідно, вони також мають змагатись один з одним, що в кінцевому результаті, дає переможця (рис. 3.12).

Турнір "Веселка" - Bracket

Winner of the tournament is 6

Round 1

Матч між 6 і Dre

6

Update

Match Date: 6/20/2024

Матч між 5 і Eminem

Eminem

Update

Match Date: 6/20/2024

Матч між 8 і 3

8

Update

Match Date: 6/20/2024

Матч між 4 і 7

4

Update

Round 2

Матч між 6 і Eminem

6

Update

Match Date: 6/21/2024

Матч між 8 і 4

4

Update

Match Date: 6/21/2024

Round 3

Матч між 6 і 4

6

Update

Match Date: 6/21/2024

Рис. 3.12 – Встановлений переможець турніру

Якщо користувача не задовольнило щось на моменті створення чи керування сіткою, є можливість видалення створеного турніру (рис. 3.13).

9	Dada	21321	Single Elimination - Seeded	6/17/2024	6/18/2024	Matches	Edit	Delete
---	------	-------	-----------------------------	-----------	-----------	-------------------------	----------------------	------------------------

Рис. 3.13 – Можливість видалення

10	Dada	232	Single Elimination	6/17/2024	6/18/2024	Matches	Edit	Delete	
11	Dada	123	Single Elimination - Seeded	6/20/2024	6/22/2024	Matches	Edit	Delete	
12	Dada	12	Single Elimination	6/20/2024	6/21/2024	Matches	Edit	Delete	
13	Dada	213	Single Elimination	6/20/2024	6/21/2024	Matches	Edit	Delete	
14	Турнір "Веселка"	Турнір для веселих людей	Single Elimination	6/20/2024	6/21/2024	Matches	Edit	Delete	
15	New Tour	132	Single Elimination - Seeded	6/21/2024	6/22/2024	Manage (0)	Matches	Edit	Delete

Рис. 3.14 – Результат успішного видалення турніру

Також важливо відмітити, що якщо користувач вже зіграв перший матч та встановив переможця, змінювати кількість учасників вже не вдасться – поле з посиланням на вікно менеджменту користувачів буде відсутнє (рис. 3.14).

У свою чергу, в сіяному методі жеребкування, як було згадано раніше, є можливість прив'язати рейтинг до самого учасника, що поглиблює можливості сервісу (рис. 3.15).

New Tour - Bracket

Round 1

Матч між 3 [1800] і 2 [1350]

3

Update

Match Date: 6/21/2024

Матч між 4 [1150] і 1 [1000]

Select Winner

Update

Match Date: 6/21/2024

Round 2

Матч між 3 [1800] і TBD

3

Match Date: 6/21/2024

Рис. 3.15 – Доданий рейтинг до користувачів

Важливою можливістю додатку є створення турнірних сіток з проходками, у випадку, коли кількість учасників не є результатом калькуляції, а саме, піднесення двійки до степеня. Таким чином, якщо створити турнір та заповнити його не, скажемо вісьмома, а сімома чи дев'ятьма учасниками, мають створюватись «проходи» в довільному порядку (рис. 3.16).

Tour1 - Bracket

Round 1

Матч між 3 і TBD

3

Match Date: 6/21/2024

Матч між 1 і 6

Select Winner

Update

Match Date: 6/21/2024

Матч між 5 і 2

Select Winner

Update

Match Date: 6/21/2024

Матч між 4 і 7

Select Winner

Update

Match Date: 6/21/2024

Round 2

Матч між 3 і TBD

3

Match Date:

Матч між TBD і TBD

Match Date:

Round 3

Матч між TBD і TBD

Match Date:

Рис. 3.16 – Створення турніру з сімома учасника

3.3. Тестування платформи

Для забезпечення стабільності та безперебійності роботи системи, було зроблено тестування.

Для перевірки загальної роботи додатку було використано деякі техніки мануального тестування. Великі та важливі помилки було виявлені та виправлені.

Наприклад, було перевірено, чи користувач матиме змогу долучитися до турнірів, будучи гостем (рис. 3.17).

Log in

Use a local account to log in.

Email

1@gmail.com

Password

.....

☐ Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Рис. 3.17 – При переході до турнірів, виконується переадресація до логіну

За допомогою інструментів розробки в браузері було виконано перевірку на адаптивність застосунку, зокрема перевірено сторінку профілю для смартфона iPhone 14 Pro Max (рис. 3.18).

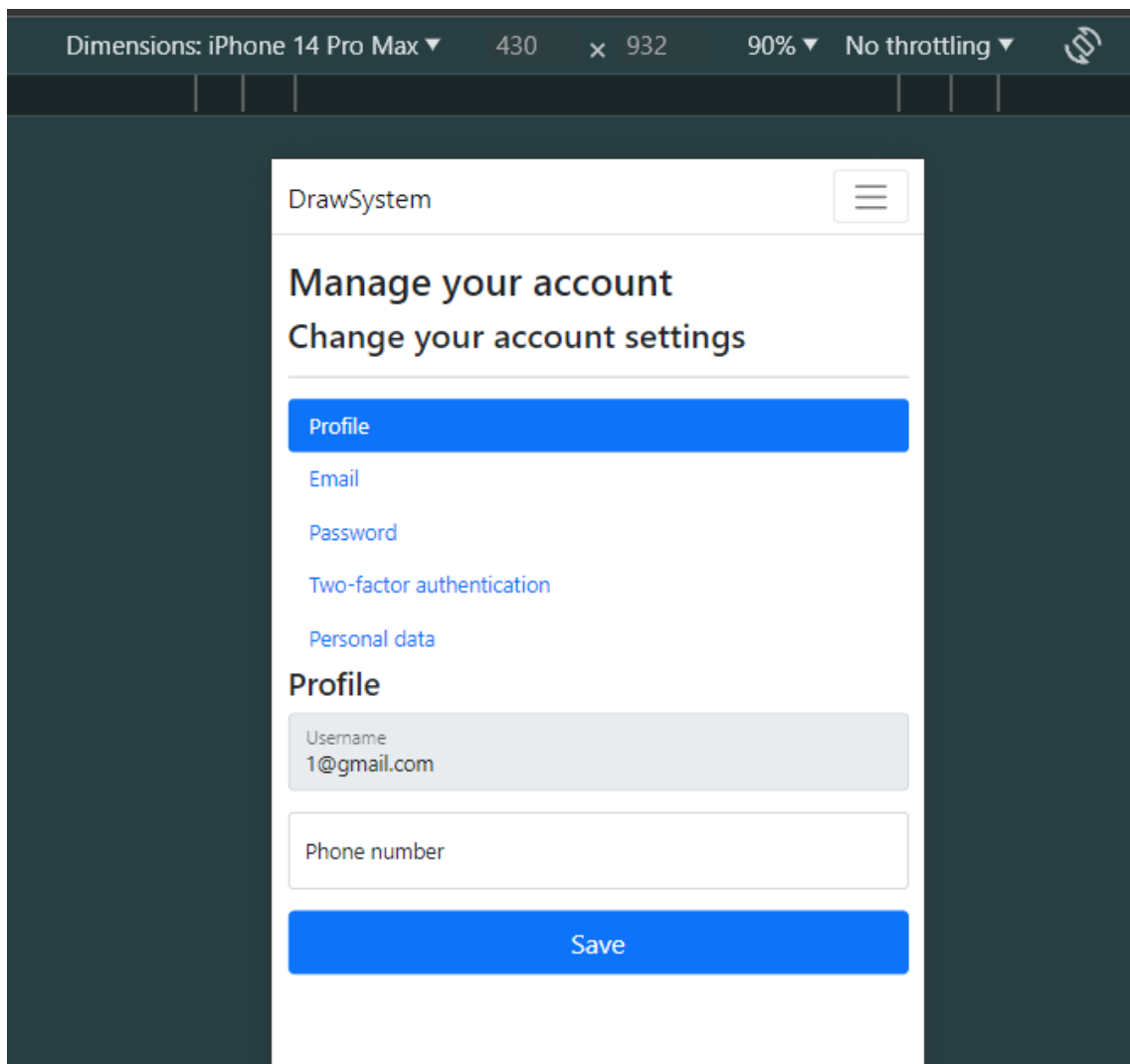


Рис. 3.18 – Адаптивність сторінки профілю

Висновки до третього розділу

У цьому розділі детально розглянуто процес розгортання додатку. Описано компоненти, які розгортаються, а також відповідні середовища, де вони функціонують. Показано взаємозв'язки між цими компонентами та середовищами за допомогою діаграми розгортання.

Представлено інтерфейс програмного застосунку та продемонстровано його основний функціонал. За допомогою скріншотів показано різні варіанти використання системи для відвідувача, організатора та адміністратора.

Під час тестування створено юніт-тести для серверної частини застосунку. Для загальної перевірки роботи системи проведено мануальне тестування, в результаті якого виправлено знайдені помилки. Використовуючи браузерні

інструменти, такі як Devtools, перевірено доступність та адаптивність додатку для мобільних пристроїв.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було досліджено використання інформаційних технологій для реалізації системи жеребкування турнірів. Було поставлено задачу та проведено аналіз існуючих програмних продуктів у цій сфері. Виділено три найбільш релевантні аналоги, порівняно їх функціонал та визначено основні модулі для розробки власної системи. Обґрунтовано вибір архітектури, інструментальних засобів, мов програмування та фреймворків, які планується використовувати. Для застосунку було обрано монолітну архітектуру.

Спроековано систему для жеребкування турнірів. Виділено двох основних користувачів: Користувач та Гість. Визначено сценарії використання системи для кожного з них. Сформовано об'єктно-орієнтовану структуру системи, представлено діаграми класів сутностей додатку та деяких використовуваних патернів проектування. Розроблено базу даних для застосунку, детально описано її таблиці. Спроековано алгоритми реєстрації на турнір та проведення жеребкування. Для їх зображення використано діаграми активності та послідовності відповідно. Після проєктування було реалізовано функціонал системи для жеребкування турнірів.

Продемонстровано інтерфейс та порядок роботи із додатком. Описано алгоритм встановлення та налаштування параметрів платформи. Наведено скріншоти вебінтерфейсу користувача. Описано алгоритм роботи із системою для кожного виду користувачів. Проведено тестування застосунку, використано техніки мануального тестування та браузерні інструменти для перевірки доступності та адаптивності. виправлено критичні помилки. Додаток має інтуїтивний інтерфейс, який дозволяє користувачам без великих зусиль організовувати та брати участь у турнірах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Волков О. Г., Савіцький Р. С. Проєктування та аналіз вимог програмних вебсистем жеребкування турнірів. Тези Всеукраїнської науково-практичної онлайн конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки», 13-17 травня 2024 р. Житомир: ДУ "Житомирська Політехніка", 2024. С. 52.
2. Turnir.net.ua [Електронний ресурс] – Режим доступу до ресурсу: <https://turnir.net.ua>.
3. Chess.com [Електронний ресурс] – Режим доступу до ресурсу: <https://chess.com>.
4. Winner – Tournament Maker App [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=il.talent.winner&hl=en>.
5. Які є види архітектури в програмуванні? [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://devzone.org.ua/qna/iaki-ye-vydy-arkhitektury-v-prohramuvanni>.
6. Клієнт серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>
7. MVC Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/mvc-framework-introduction/>.
8. Моноліт vs Мікросервіси [Електронний ресурс] – Режим доступу до ресурсу: <https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/>.
9. Overview of ASP.NET Core [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0>.
10. SQL Server [Електронний ресурс] – Режим доступу до ресурсу: <https://www.microsoft.com/en-us/sql-server/sql-server-2022>.

11. Entity Framework Core [Электронный ресурс] – Режим доступа до ресурсу: <https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx>.
12. Tournament Formats | Webmazes [Электронный ресурс] – Режим доступа до ресурсу: <https://www.webmazes.net/tournamentFormats/>.
13. Attributes in C# [Электронный ресурс] – Режим доступа до ресурсу: https://www.tutorialspoint.com/csharp/csharp_attributes.htm.
14. Microsoft Identity [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/entra/identity-platform/>.
15. Running Javascript in C# [Электронный ресурс] – Режим доступа до ресурсу: <https://medium.com/@manjulcr7/running-javascript-in-c-combining-two-powerful-languages-6c6d0a3672ab>.

ДОДАТКИ

ЛІСТИНГ ВИХІДНОГО КОДУ

```

@model CreateTournamentViewModel

<form method="post">
    <div class="mt-3">
        <label asp-for="Name" class="form-label"></label>
        <input class="form-control" asp-for="Name" />
        <span asp-validation-for="Name" class="invalid-feedback"></span>
    </div>
    <div class="mt-3">
        <label asp-for="StartDate" class="form-label"></label>
        <input class="form-control" asp-for="StartDate" />
        <span asp-validation-for="StartDate" class="invalid-feedback"></span>
    </div>
    <div class="mt-3">
        <label asp-for="EndDate" class="form-label"></label>
        <input class="form-control" asp-for="EndDate" />
        <span asp-validation-for="EndDate" class="invalid-feedback"></span>
    </div>
    <div class="mt-3">
        <label asp-for="Description" class="form-label"></label>
        <textarea class="form-control" asp-for="Description">
        </textarea>
        <span asp-validation-for="@Model.Description" class="invalid-feedback"></span>
    </div>
    <div class="mt-3">
        <select asp-for="DrawMethodId" class="form-select">
            <option disabled selected> Select draw method </option>
            @foreach (DrawMethod item in ViewBag.Methods)
            {
                <option value="@item.Id">@item.Name</option>
            }
        </select>
        <span asp-validation-for="DrawMethodId" class="invalid-feedback"></span>
    </div>

    <div class="mt-3">
        <button type="submit" class="btn btn-primary">Create Tournament </button>
    </div>
</form>

using System.ComponentModel.DataAnnotations;

namespace DrawSystem.Models.ViewModels
{
    public class EditTournamentViewModel
    {
        [Required(ErrorMessage = "Name is required")]
        [Display(Name = "Name")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Description is required")]
        [Display(Name = "Description")]
        public string Description { get; set; }
    }
}

using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using System.Linq;
using Microsoft.EntityFrameworkCore;

```

```

using DrawSystem.Models;
using DrawSystem.Data;
using DrawSystem.Models.Entities;
using Microsoft.AspNetCore.Authorization;
namespace DrawSystem.Controllers
{
    [Authorize]
    public class ParticipantsController : Controller
    {
        private readonly DrawSystemContext _context;

        public ParticipantsController(DrawSystemContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<IActionResult> List()
        {
            var participants = await _context.Users.ToListAsync();
            return View(participants);
        }

        // GET: Participants
        public async Task<IActionResult> Index()
        {
            return View(await _context.Participants.ToListAsync());
        }

        // GET: Participants/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var participant = await _context.Participants
                .FirstOrDefaultAsync(m => m.Id == id);
            if (participant == null)
            {
                return NotFound();
            }

            return View(participant);
        }

        // GET: Participants/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: Participants/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        //public async Task<IActionResult> Create(CreateParticipantViewModel
viewModel)
        //{
            var participant = new Participant
            {
                Name = viewModel.Name,
                Email = viewModel.Email,
                Password = viewModel.Password,
                RegistrationDate = viewModel.RegistrationDate,
            };
            await _context.Participants.AddAsync(participant);
        }
    }
}

```

```

//    await _context.SaveChangesAsync();

//    return View();
//}

// GET: Participants/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var participant = await _context.Participants.FindAsync(id);
    if (participant == null)
    {
        return NotFound();
    }
    return View(participant);
}

// POST: Participants/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("ParticipantID,Name,Email,RegistrationDate")] Participant participant)
{
    if (id != participant.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(participant);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ParticipantExists(participant.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(participant);
}

// GET: Participants/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var participant = await _context.Participants
        .FirstOrDefaultAsync(m => m.Id == id);

```

```

        if (participant == null)
        {
            return NotFound();
        }

        return View(participant);
    }

    // POST: Participants/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var participant = await _context.Participants.FindAsync(id);
        _context.Participants.Remove(participant);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool ParticipantExists(int id)
    {
        return _context.Participants.Any(e => e.Id == id);
    }
}

using DrawSystem.Data;
using DrawSystem.Models.Entities;
using DrawSystem.Models.ViewModels;
using DrawSystem.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace DrawSystem.Controllers
{
    [Authorize]
    public class TournamentsController : Controller
    {
        private readonly DrawSystemContext _context;
        private readonly UserManager<DrawSystemUser> _userManager;
        private readonly ICurrentUserService _currentUserService;

        public TournamentsController(DrawSystemContext context,
            UserManager<DrawSystemUser> userManager,
            ICurrentUserService currentUserService)
        {
            _context = context;
            _userManager = userManager;
            _currentUserService = currentUserService;
        }

        [HttpGet]
        public async Task<IActionResult> Create()
        {
            var methods = await _context.Methods.ToListAsync();
            ViewBag.Methods = methods;
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(CreateTournamentViewModel model)
        {
            if (!ModelState.IsValid)
            {

```

```

        var methods = await _context.Methods.ToListAsync();
        ViewBag.Methods = methods;
        return View(model);
    }

    var tournament = new Tournament
    {
        Name = model.Name,
        Description = model.Description,
        StartDate = model.StartDate,
        EndDate = model.EndDate,
        DrawMethodId = model.DrawMethodId,
        UserId = _currentUserService.GetUserId(),
    };

    await _context.AddAsync(tournament);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

[HttpGet]
public async Task<IActionResult> Index()
{
    var tournaments = await _context.Tournaments
        .Include(t => t.Matches)
        .ThenInclude(m => m.Result)
        .Include(t => t.Participants)
        .Include(t => t.DrawMethod)
        .ToListAsync();

    var a = tournaments.Where(t => t.Matches.Any(m => m.Result is not
null)).ToList();

    var viewModels = tournaments.Select(t => new TournamentViewModel
    {
        Tournament = t,
        IsParticipantsManageable = !t.Matches.Any(m => !m.IsBye() && m.Result
is not null)
    }).ToList();

    return View(viewModels);
}

[HttpGet]
public async Task<IActionResult> Edit(int id)
{
    var tournament = await _context.Tournaments.FindAsync(id);
    return View(tournament);
}

[HttpPost]
public async Task<IActionResult> Edit(EditTournamentViewModel model, int id)
{
    var tournament = await _context.Tournaments.FindAsync(id);
    if (tournament is not null)
    {
        tournament.Name = model.Name;
        tournament.Description = model.Description;
        await _context.SaveChangesAsync();
    }

    return RedirectToAction("Index", "Tournaments");
}

[HttpGet]
public async Task<IActionResult> ManageParticipants(int id)

```

```

{
    var tournament = await _context.Tournaments
        .Include(t => t.DrawMethod)
        .Include(t => t.Participants)
        .FirstOrDefaultAsync(t => t.Id == id);

    if (tournament is null)
    {
        return NotFound();
    }

    ViewBag.DrawMethodName = tournament.DrawMethod.Name;
    var model = new ManageParticipantsViewModel
    {
        Participants = tournament.Participants,
        Id = id,
    };
    return View(model);
}

[HttpPost]
public async Task<IActionResult>
ManageParticipants(ManageParticipantsViewModel model, int id)
{
    var tournament = await _context.Tournaments
        .Include(t => t.Participants)
        .Include(t => t.Matches)
        .Include(t => t.DrawMethod)
        .FirstOrDefaultAsync(t => t.Id == id);

    if (tournament is not null)
    {
        tournament.Participants.Clear();
        model.Participants.ForEach(p => p.TournamentId = id);
        tournament.Participants.AddRange(model.Participants);
        await _context.SaveChangesAsync();

        tournament.Matches.Clear();
        await GenerateMatches(tournament);
    }

    return RedirectToAction("Index", "Tournaments");
}

public List<Match> GenerateBracket(int tournamentId, int numberOfParticipants)
{
    var matches = new List<Match>();
    var totalRounds = (int)Math.Ceiling(Math.Log2(numberOfParticipants));
    var currentRoundMatches = new List<Match>();

    for (int round = 1; round <= totalRounds; round++)
    {
        int matchesInRound = (int)Math.Pow(2, totalRounds - round);
        for (int matchNumber = 0; matchNumber < matchesInRound; matchNumber++)
        {
            var match = new Match
            {
                TournamentId = tournamentId,
                Round = round
            };
            matches.Add(match);
        }

        currentRoundMatches = matches.Where(m => m.Round == round).ToList();
    }
    return matches;
}

```

					ИП3.КР.Б-121-24-ИЗ	
						64

```

    }

    private async Task<List<Match>> GenerateMatches(Tournament tournament)
    {
        var participants = tournament.Participants.OrderByDescending(p =>
p.Rating).ToList();
        if (tournament.DrawMethod?.Name ==
Constants.DrawMethods.SingleElimination)
        {
            var random = new Random();
            participants = tournament.Participants
                .OrderBy(p => random.Next())
                .ToList();

        }
        var matches = GenerateBracket(tournament.Id, participants.Count);
        await _context.Matches.AddRangeAsync(matches);
        await _context.SaveChangesAsync();

        LinkMatches(matches);

        var round = 1;
        var firstRoundMatches = GetMatchesByRound(matches, round);

        var numByes = GetNumByes(participants.Count);
        for (int i = 0; i < numByes; i++)
        {
            var participant = participants[i];
            var match = firstRoundMatches[i];
            match.Participant1Id = participant.Id;
            match.Participant2Id = null;
            match.MatchDate = tournament.StartDate;
            match.Result = new Result
            {
                WinnerId = participant.Id,
                Winner = participant,
            };

            var nextMatch = matches.FirstOrDefault(m => m.Id ==
match.NextMatchId);
            if (nextMatch != null)
            {
                nextMatch.Participant1Id = participant.Id;
            }
        }

        for (int i = numByes, j = numByes; i < participants.Count; i += 2, j++)
        {
            var match = firstRoundMatches[j];
            match.Participant1Id = participants[i].Id;
            match.Participant2Id = i + 1 < participants.Count ? participants[i +
1].Id : null;
            match.MatchDate = tournament.StartDate;
        }

        await _context.SaveChangesAsync();
        return matches;
    }

    private void LinkMatches(List<Match> matches)
    {
        var rounds = matches.Select(m => m.Round).Distinct().OrderBy(r =>
r).ToList();
        foreach (var round in rounds)
        {
            var currentRoundMatches = GetMatchesByRound(matches, round);

```

```

        var nextRoundMatches = GetMatchesByRound(matches, round + 1);

        if (round == rounds.Last())
        {
            break;
        }

        for (int i = 0; i < currentRoundMatches.Count; i++)
        {
            var match = currentRoundMatches[i];
            match.NextMatchId = nextRoundMatches[i / 2].Id;
        }
    }
}

private List<Match> GetMatchesByRound(List<Match> matches, int round)
{
    return matches.Where(m => m.Round == round).ToList();
}

private int GetNumByes(int numParticipants)
{
    int nextPowerOfTwo = 1;
    while (nextPowerOfTwo < numParticipants)
    {
        nextPowerOfTwo *= 2;
    }
    return nextPowerOfTwo - numParticipants;
}

[HttpPost]
public async Task<IActionResult> Delete(int id)
{
    var tournament = await _context.Tournaments.FindAsync(id);
    if (tournament is not null)
    {
        _context.Tournaments.Remove(tournament);
        await _context.SaveChangesAsync();
    }

    return RedirectToAction("Index", "Tournaments");
}
}
}

```