

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ФОРМУВАННЯ
КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ
SOFTWARE FOR SUPPORTING TEAM FORMATION FOR
FOOTBALL TRAINING SESSIONS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-151

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БУРМА Н.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛІК О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (КОД І НАЙМЕНУВАННЯ)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БУРМИ Нікіти Сергійовича
 (ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення підтримки формування команд для занять футболом. Software for Supporting Team Formation for Football Training Sessions
 керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	БСЛОВА А.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Нікіта БУРМА
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Валерій ЛЬОВКІН
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 122 с., 5 табл., 44 рис., 3 дод., 18 джерел.

БРОНЮВАННЯ МАЙДАНЧИКІВ, ВЕБДОДАТОК, ОРГАНІЗАЦІЯ МАТЧІВ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СПОРТИВНА ПЛАТФОРМА.

Об'єкт роботи – процеси формування футбольних команд, організації турнірів та управління спортивними об'єктами в аматорському футболі. Предмет роботи – програмне забезпечення підтримки формування команд для занять футболом. Мета роботи – розробити програмне забезпечення підтримки формування команд для занять футболом для формування футбольної спільноти та цифрової підтримки аматорського спорту.

Матеріали, методи та технічні засоби: мова TypeScript, фреймворки NestJS та React, система управління базами даних PostgreSQL, ORM TypeORM, JWT автентифікація, Leaflet для картографії, супутні базові засоби веброзробки.

Результати. Розроблена програмна платформа забезпечує створення профілів гравців з медіа-портфоліо, систему запрошень та заявок на вступ до команд, повноцінне управління турнірами з призовими фондами, календарне бронювання майданчиків з системою затвердження власниками, відстеження детальної статистики матчів та рейтингову систему оцінювання якості спортивних об'єктів.

Висновки. Створена програмна платформа вирішує проблему роз'єднаності аматорського футболу, поєднуючи соціальну, організаційну та комерційну складові в єдину екосистему.

Галузь використання. Аматорський та напівпрофесійний футбол, організація місцевих турнірів, управління спортивними комплексами, створення футбольних спільнот.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 122 pages, 5 tables, 44 figures, 3 appendixes, 18 sources.

VENUE BOOKING, WEB APPLICATION, MATCH ORGANIZATION, SOFTWARE, SPORTS PLATFORM.

The object of the work is the process of forming football teams, organizing tournaments and managing sports facilities in amateur football. The subject of the work is software to support the formation of teams for football. The purpose of the work is to develop software to support the formation of teams for football for the formation of a football community and digital support for amateur sports.

Materials, methods and technical means: TypeScript language, NestJS and React frameworks, PostgreSQL database management system, TypeORM ORM, JWT authentication, Leaflet for mapping, related basic web development tools.

Results. The developed platform provides player profile creation with media portfolios, invitation and application system for team joining, comprehensive tournament management with prize funds, calendar-based venue booking with owner approval system, detailed match statistics tracking and rating system for venue quality evaluation.

Conclusions. The platform addresses fragmentation in amateur football by combining social, organizational, and commercial features into one ecosystem.

Field of use. Amateur and semi-professional football, local tournament organization, sports complex management, football community creation.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Огляд стану аматорського футболу в Україні та організаційних аспектів координації між різними його учасниками.....	11
1.2 Аналіз існуючих рішень та їх обмежень.....	13
1.3 Формування концепції розроблюваного програмного забезпечення підтримки формування команд для занять футболом	18
1.4 Висновки за розділом 1	19
2 Проєктування програми.....	20
2.1 Визначення функціональних вимог користувачів програмного забезпечення підтримки формування команд для занять футболом	20
2.2 Визначення засобів для розробки програмного забезпечення підтримки формування команд для занять футболом	26
2.3 Структура бази даних програмного забезпечення підтримки формування команд для занять футболом	29
2.4 Висновки за розділом 2	40
3 Розробка програми	41
3.1 Визначення порядку роботи з програмним забезпеченням підтримки формування команд для занять футболом	41
3.2 Реалізація основних класів програмного забезпечення підтримки формування команд для занять футболом	43
3.3 Реалізація основних програмних функцій програмного забезпечення підтримки формування команд для занять футболом	49
3.4 Висновки за розділом 3	55
4 Експлуатація та тестування програми	57

4.1 Призначення та умови виконання програми підтримки формування команд для занять футболом	57
4.2 Логіка експлуатації програми підтримки формування команд для занять футболом	57
4.3 Тестування програми підтримки формування команд для занять футболом	69
4.4 Висновки за розділом 4	72
Висновки	73
Перелік джерел посилання	75
Додаток А Технічне завдання	77
Додаток Б Текст програми	83
Додаток В Слайди презентації	113

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
DOM	– Document Object Model;
JWT	– JSON Web Token;
Type ORM	– Type Object-Relational Mapping;
ААФУ	– Асоціація аматорського футболу України;
ПЗ	– програмне забезпечення;
СУБД	– система управління базами даних;
ТЗ	– технічне завдання.

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується активним впровадженням цифрових рішень у всі сфери людської діяльності, включаючи спорт та фізичну культуру. Особливо актуальним це стає для аматорського спорту, який потребує ефективних інструментів координації між різними учасниками такої спортивної діяльності.

Аматорський футбол як один з найпопулярніших видів спорту в Україні демонструє значний потенціал для розвитку, проте стикається з системними викликами саме в організації діяльності. Традиційні методи координації між гравцями, командами, організаторами турнірів та власниками спортивних об'єктів показують свою неефективність в умовах зростаючого масштабу аматорської футбольної діяльності.

Проблема координації учасників спортивної спільноти не є унікальною для України. Аналогічні виклики спостерігаються в багатьох країнах світу, що призвело до появи різноманітних цифрових платформ та додатків. Однак існуючі рішення часто мають фрагментарний характер, вирішуючи лише окремі аспекти спортивної діяльності без комплексного підходу до потреб всіх учасників.

Розвиток вебтехнологій та мобільних додатків відкриває нові можливості для створення інтегрованих платформ, які можуть забезпечити ефективну взаємодію між всіма учасниками аматорського футболу. Такі платформи мають потенціал не лише з точки зору автоматизації існуючих процесів, але й здатні створити принципово нові можливості для розвитку спортивної спільноти.

Особливий інтерес представляє концепція екосистемного підходу до організації аматорського футболу, коли платформа функціонує не просто як набір окремих сервісів, а як цілісна система, де активність кожного учасника створює цінність для інших. Реалізація такого підходу вимагає глибокого

розуміння специфіки аматорського футболу та застосування сучасних технологічних рішень.

Усе це формує цілий ряд факторів, які вказують на актуальність створення програмного забезпечення (ПЗ) для організаційної підтримки даного напрямку спортивної діяльності.

Враховуючи визначену актуальність дослідження, мета роботи може бути сформульована як: розробка програмного забезпечення підтримки формування команд для занять футболом.

Досягнення мети вимагає розв'язання зв'язаних між собою наступних завдань:

- аналіз існуючого стану організації аматорського футболу та організаційних аспектів координації між різними його учасниками;
- визначення вимог до ПЗ підтримки формування команд для занять футболом;
- проектування ПЗ підтримки формування команд для занять футболом;
- реалізація і тестування ПЗ підтримки формування команд для занять футболом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд стану аматорського футболу в Україні та організаційних аспектів координації між різними його учасниками

Футбол є одним із наймасовіших видів спорту в Україні та охоплює як професійний, так й аматорський рівень. При цьому, аматорський футбол виконує важливу соціальну функцію – залучає населення до активного способу життя, сприяє формуванню локальних спільнот та розвитку спортивної культури на місцевому рівні. Орієнтовна кількість осіб, що грають у футбол на аматорському рівні в Україні, у тому числі осіб, що беруть участь у неофіційних змаганнях, становить щонайменше 360 тисяч громадян [1].

Значну частку цієї спільноти складають молоді гравці. Так, за даними регіональних управлінь спорту, у різноманітних дитячо-юнацьких футбольних гуртках бере участь близько 280 тисяч осіб віком від 8 до 17 років [1]. Крім того, опитування, проведене в рамках проєкту UEFA GROW, свідчить, що майже кожна друга дитина в Україні в тій чи іншій формі грає у футбол, що підтверджує глибоку інтегрованість цієї гри в молодіжне середовище [2].

На загальнонаціональному рівні аматорські футбольні змагання організовуються Асоціацією аматорського футболу України (ААФУ). У сезоні 2024/2025 в чемпіонаті України серед аматорів беруть участь 20 команд, поділених на дві групи [3]. Крім чемпіонату, проводиться Кубок України серед аматорів, а також регулярні регіональні першості в кожній області. Наприклад, у Запорізькій області щорічно відбувається обласний чемпіонат серед аматорських команд, організований Запорізькою обласною асоціацією футболу [4].

Це свідчить не лише про популярність футболу в Україні, а й про високий рівень залучення до нього на всіх рівнях – від вуличних матчів до участі у міжнародних аматорських турнірах. Однак за цими значними цифрами приховуються серйозні структурні проблеми, які перешкоджають повноцінному розвитку галузі.

Однією із основних проблем розвитку аматорського футболу є фрагментарність спільноти, яка проявляється в ускладненні пошуку команди відповідного рівня та свідчить про відсутність ефективних механізмів об'єднання гравців.

Крім того, організатори аматорських футбольних турнірів на регіональному та нижчих рівнях стикаються з проблемами координації між командами, що часто призводить до скасування або перенесення матчів. Що, у свою чергу, зумовлює нерівномірність завантаження футбольних майданчиків та знижує економічну ефективність їх експлуатації.

Варто зазначити, що інфраструктурний потенціал аматорського футболу в Україні значно перевищує поточний рівень його використання. Останні роки в Україні активно розвивалась матеріально-технічна база, що дозволяє значній кількості громадян долучатися до гри. За даними Міністерства регіонального розвитку, у період з 2017 року в Україні було побудовано більше 1 300 футбольних полів зі штучним покриттям, значна частина яких призначена саме для масових занять спортом [5]-[6]. Всього в Україні функціонує більше 3 000 футбольних полів різного типу покриття, близько 2 000 спортивних комплексів, що надають можливість проведення футбольних матчів, а також близько 1 000 критих спортивних об'єктів, придатних для проведення міні-футболу [5]-[6].

Проте середня завантаженість всіх цих об'єктів становить лише 45-60% від максимальної, що вказує на недостатньо ефективне використання наявних ресурсів.

Ця неефективність частково пояснюється відсутністю системного підходу до управління спортивною діяльністю. Показовим є той факт, що лише незначна частка аматорських команд ведуть облік статистики гравців, що унеможливорює об'єктивне оцінювання розвитку як окремих спортсменів, так і команд загалом.

Виявлені дисфункції формують взаємопов'язаний комплекс проблем, які системно перешкоджають розвитку аматорського футболу:

- фрагментарність спільноти, тобто відсутність єдиної платформи для взаємодії між учасниками футбольної спільноти, що призводить до ізоляції команд та гравців в межах локальних кіл;

- неефективність командоутворення, що впливає на стабільність складу команд;

- проблеми організації турнірів виявляються в тому, що організатори витрачають більшість свого часу на базові адміністративні процедури замість розвитку спортивної складової турнірів. Це значно знижує якість змагань;

- відсутність ефективної системи бронювання призводить до неефективного використання спортивних об'єктів (стадіонів, спортивних комплексів, футбольних майданчиків та полів), більшість з яких простоюють близько половини потенційного часу роботи;

- відсутність механізмів оцінки, накопичення та зберігання інформації щодо професіонального рівня гравців та якості спортивних об'єктів значно ускладнює розвиток аматорської футбольної спільноти, адже без ефективної системи об'єктивних критеріїв оцінки неможливо відслідковувати поступовий розвиток індивідуальних навичок гравців, якості організації футбольних змагань тощо.

Виявлені проблеми та потенціал розвитку аматорського футболу в Україні обумовлюють необхідність пошуку ефективних технологічних рішень для їх вирішення. Однак перш ніж приступити до розробки нової платформи, важливо проаналізувати існуючі рішення та з'ясувати, чому вони не задовольняють повною мірою наявні потреби.

1.2 Аналіз існуючих рішень та їх обмежень

Пошук технологічних рішень з організації аматорського футболу варто розпочати з аналізу існуючих платформ, які спрямовані на вирішення подібних завдань в цій та суміжних сферах діяльності. Більшість існуючих спортивних платформ розвивалися еволюційно, адаптуючи загальні підходи

до конкретних потреб, що призвело до появи декількох специфічних категорій платформ.

Першою категорією платформ, яка природно привертає увагу, є системи організації подій загального призначення. Одним із представників цього підходу є Meetup (рис. 1.1) [7] – міжнародний сервіс, який розширив свою первинну концепцію соціальних зустрічей на спортивну сферу.

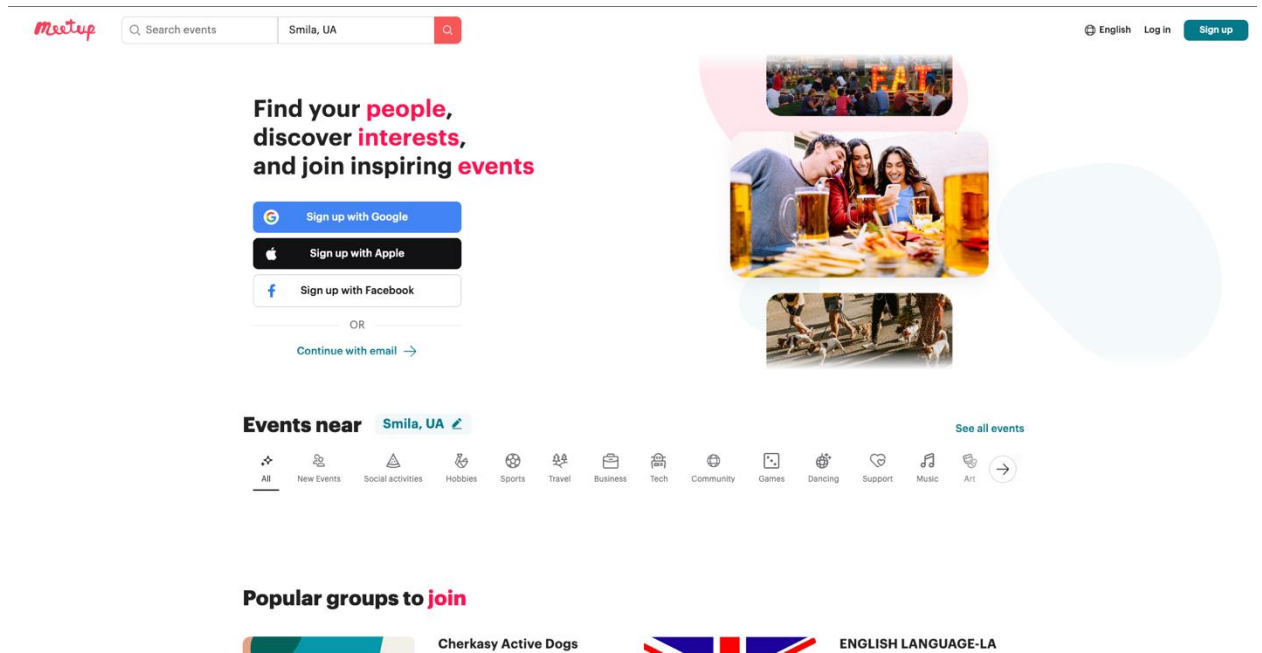


Рисунок 1.1 – Інтерфейс платформи Meetup для спортивних подій [7]

Філософія Meetup базується на принципі разових зустрічей за інтересами, що добре працює для багатьох видів соціальної активності, але створює фундаментальні протиріччя зі специфікою командних видів спорту. Адже, якщо для книжкового клубу або фотопрогулянки змінний склад учасників є природним явищем, то для футбольної команди це є критичною проблемою. Meetup дозволяє створювати групи та організовувати події, але його архітектура принципово не передбачає формування стабільних колективів з постійним складом учасників.

Це обмеження породжує низку функціональних недоліків: відсутність системи командування, неможливість відстеження індивідуального прогресу гравців, брак інтеграції з системою бронювання спортивних об'єктів

та відсутність механізмів проведення турнірів. По суті, Meetup залишається сервісом для разової гри, не еволюціонуючи у напрямку підтримки довгострокової та сталої спортивної діяльності.

Усвідомлення обмежень подієвих платформ природно приводить до другої категорії рішень – систем бронювання спортивних об'єктів. Ця категорія впливає з того, що спорт, на відміну від соціальних активностей, критично залежить від наявності спортивної інфраструктури, а її ефективне використання потребує технологічного посередництва.

Представником цього підходу є Playtomic (рис. 1.2) [8] – іспанська платформа, яка демонструє принципово іншу філософію організації спортивної діяльності. Playtomic еволюціонував з туристичної індустрії, адаптувавши моделі бронювання готелів до спортивних об'єктів.

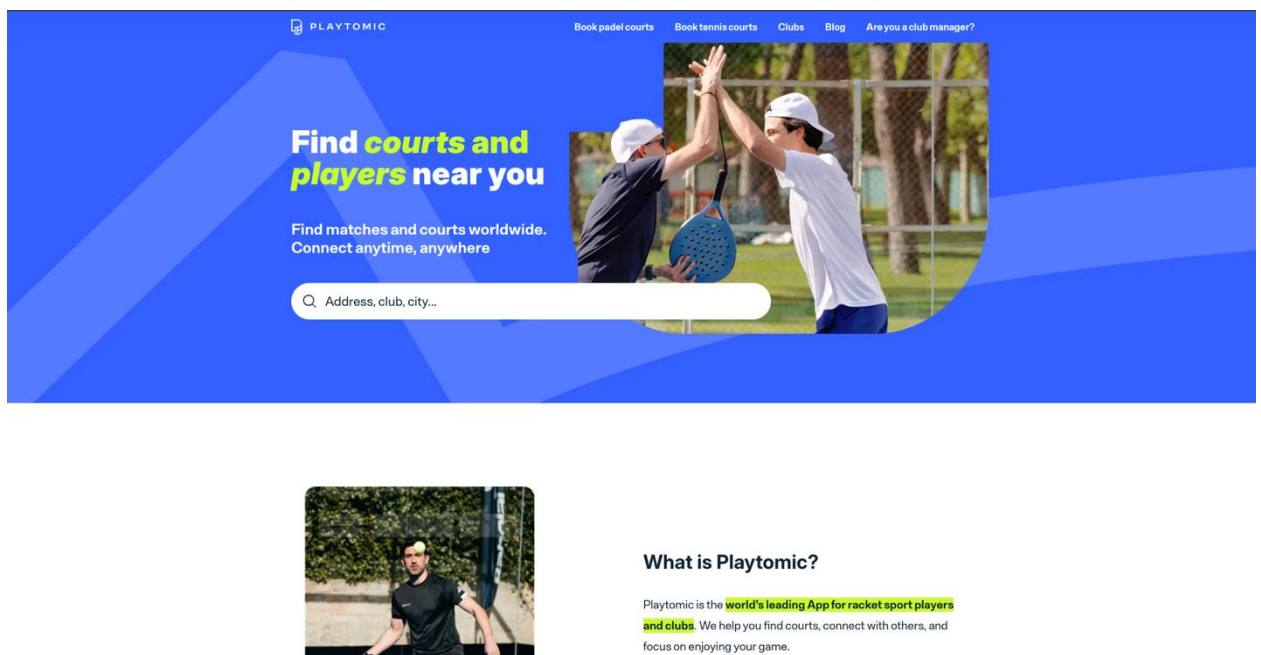


Рисунок 1.2 – Інтерфейс додатку Playtomic [8]

Архітектура платформи Playtomic відображає інфраструктурно-центричне мислення, центром якого є спортивний об'єкт із визначеними характеристиками (розмір, покриття, цінова політика тощо). Завданням платформи є максимально ефективно розподілити наявний час роботи між користувачами.

Цей підхід породжує потужну функціональність: інтуїтивне онлайн-бронювання з миттєвою оплатою, систему користувацьких рейтингів об'єктів, геолокаційний пошук найближчих майданчиків та інтеграцію з персональними календарями.

Однак інфраструктурний підхід створює принципово інші обмеження порівняно з подієвим підходом Meetup. Зокрема Playtomic розглядає користувачів як індивідуальних споживачів послуг бронювання, що добре працює для індивідуальних видів спорту (теніс, падел), але не враховує командну природу футболу. Платформа не передбачає механізмів формування постійних колективів, організації турнірів чи відстеження командної статистики. По суті, Playtomic залишається "цифровим реєстратором" фізичних ресурсів, не еволюціонуючи у напрямку спортивної спільноти.

Обмеження як подієвого, так і інфраструктурного підходів закономірно призводять до появи третьої категорії рішень – спортивних соціальних мереж. Ця категорія народилася з усвідомлення, що спорт є принципово соціальною діяльністю, яка потребує не тільки координації подій чи бронювання ресурсів, але й постійної комунікації між учасниками.

Показовим представником цього напрямку є HEJA (рис. 1.3) [9] – скандинавська платформа, яка демонструє комунікаційно-центричний підхід до організації спортивної діяльності. На відміну від Meetup, що фокусується на подіях, чи Playtomic, що зосереджується на ресурсах, HEJA ставить у центр уваги взаємодію між учасниками.

Філософія HEJA базується на принципі команда як мікросоціум: платформа розглядає спортивний колектив як самодостатню спільноту з власною культурою, історією та традиціями. Це породжує унікальну функціональність: багатий інструментарій для створення командної ідентичності (профілі з медіа-контентом), засоби внутрішньоконандної комунікації, механізми планування спільної діяльності та можливості демонстрації досягнень зовнішній аудиторії.

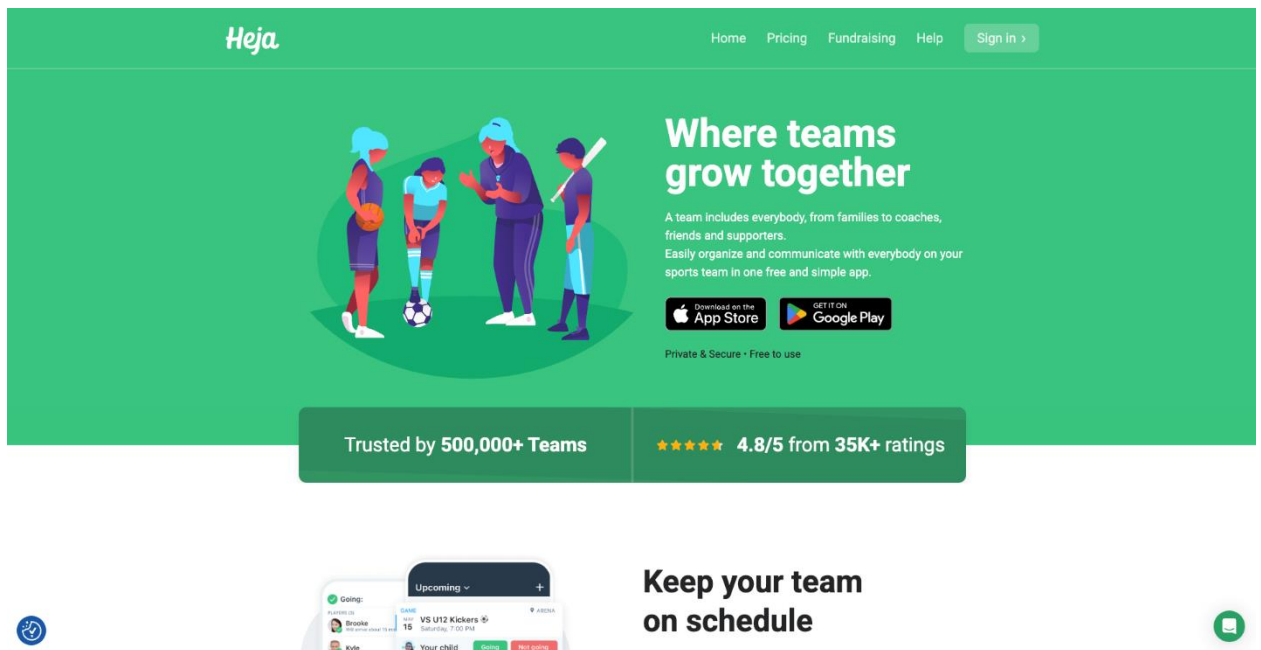


Рисунок 1.3 – Інтерфейс додатку HEJA [9]

Однак комунікаційний підхід HEJA має принципове обмеження: він передбачає існування готових, сформованих команд, які потребують лише інструментів для покращення внутрішньої взаємодії. HEJA ефективно підтримує життєдіяльність команд, які вже знайшли один одного, але не допомагає гравцям, які шукають команду, чи командам, що потребують нових учасників. Це створює парадокс для спільноти українського аматорського футболу, де основною проблемою є саме відсутність дієвих механізмів сприяння командоутворенню.

Виявлені обмеження існуючих рішень, у поєднанні з проаналізованими раніше проблемами галузі, створюють сприятливі умови для розробки принципово нового підходу до організації аматорського футболу. Цей підхід має базуватися на комплексному баченні потреб усіх учасників футбольної спільноти та забезпечувати їх ефективну взаємодію в рамках єдиної екосистеми.

1.3 Формування концепції розроблюваного програмного забезпечення підтримки формування команд для занять футболом

В ході проведеного аналізу було встановлено ключову проблему технічного забезпечення функціонування спільноти аматорського футболу, а саме: наявні платформи намагаються адаптувати готові архітектурні рішення з інших галузей діяльності до специфіки аматорського футболу, що неминуче призводить до виникнення низки системних обмежень. Натомість для забезпечення ефективного розвитку цієї сфери діяльності потрібен принципово інший підхід – створення архітектури, що виходить з унікальної природи аматорського футболу як соціально-спортивного явища.

Адже аматорський футбол – це особлива форма соціальної активності, яка поєднує фізичну діяльність, командну взаємодію, змагальність та прагнення до самореалізації. В динаміці ця діяльність характеризується цілою низкою якісних та кількісних показників кожної зі сторін такої взаємодії (гравців, команд, спортивної інфраструктури, організаторів турнірів, а також самих турнірів). Всі ці особливості вимагають якісно нового підходу, який не копіюватиме логіку інших галузей, а органічно вирощується з потреб самої футбольної спільноти (табл. 1.1).

Крім того, інституційні та цифрові рішення, що підтримують розвиток аматорського футболу, мають враховувати високий ступінь децентралізації цього явища. Йдеться не лише про фізичне розміщення команд у різних регіонах, а й про варіативність організаційних моделей, рівнів залученості та культурних контекстів. Тому ефективна архітектура повинна бути гнучкою, модульною та здатною до адаптації в умовах обмежених ресурсів і швидко змінних соціальних реалій, що притаманні аматорському спорту. Такий підхід дозволить не лише відобразити, але й підсилити соціальну динаміку, яка формує екосистему аматорського футболу.

Таблиця 1.1 – Результати порівняння аналогів та розробки ПЗ підтримки формування команд для занять футболом

Критерій	Назва платформи			
	Meetup	Playtomic	HEJA	SportPlatform
Реєстрація гравців	Є	Є	Є	Є
Профілі команд	Немає	Немає	Є	Є
Верифікація	Немає	Є	Немає	Є
Пошук гравців	Обмежений	Немає	Немає	Є
Заявки на вступ	Немає	Немає	Немає	Є
Пошук майданчиків	Немає	Є	Немає	Є
Онлайн бронювання	Немає	Є	Немає	Є
Статистика гравців	Немає	Немає	Є	Є
Чат	Є	Немає	Немає	Немає

1.4 Висновки за розділом 1

Комплексне дослідження предметної області виявило наявні проблеми організації аматорського футболу в Україні: при значних масштабах залученості учасників галузь страждає від низки системних проблем – фрагментарності спільноти, нестабільності команд та неефективного використання спортивної інфраструктури. Аналіз трьох категорій існуючих платформ показав, що кожна з них механічно адаптує архітектурні принципи з інших галузей, вирішуючи лише частину проблем. Створення технічного рішення, що виходить з унікальної природи аматорського футболу як соціально-спортивного явища, сприятиме розвитку аматорської футбольної екосистеми в Україні.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Визначення функціональних вимог користувачів програмного забезпечення підтримки формування команд для занять футболом

Повний перелік функціональних вимог до програми для об'єднання футбольних гравців у команди, організації турнірів та координації використання спортивних майданчиків представлений у технічному завданні (ТЗ) на роботу в додатку А.

Відповідно до результатів аналізу предметної області та ТЗ, доцільно розподілити користувачів на чотири функціональні ролі:

- неавторизований користувач;
- гравець;
- організатор турнірів;
- власник майданчику.

Неавторизований користувач – відвідувач платформи, який має обмежений доступ до загальної інформації. Такий користувач може переглядати головну сторінку з описом сервісів та загальну статистику платформи. Для нього доступна реєстрація з можливістю вибору типу облікового запису, авторизація через стандартну форму входу з автоматичним перенаправленням на відповідну панель управління, а також можливість відновлення паролю через електронну пошту.

Неавторизованому користувачу має бути доступна наступна функціональність ПЗ (рис. 2.1):

- перегляд головної сторінки з описом платформи;
- перегляд загальної статистики;
- реєстрація в системі з вибором ролі;
- авторизація через форму входу;
- відновлення пароля.

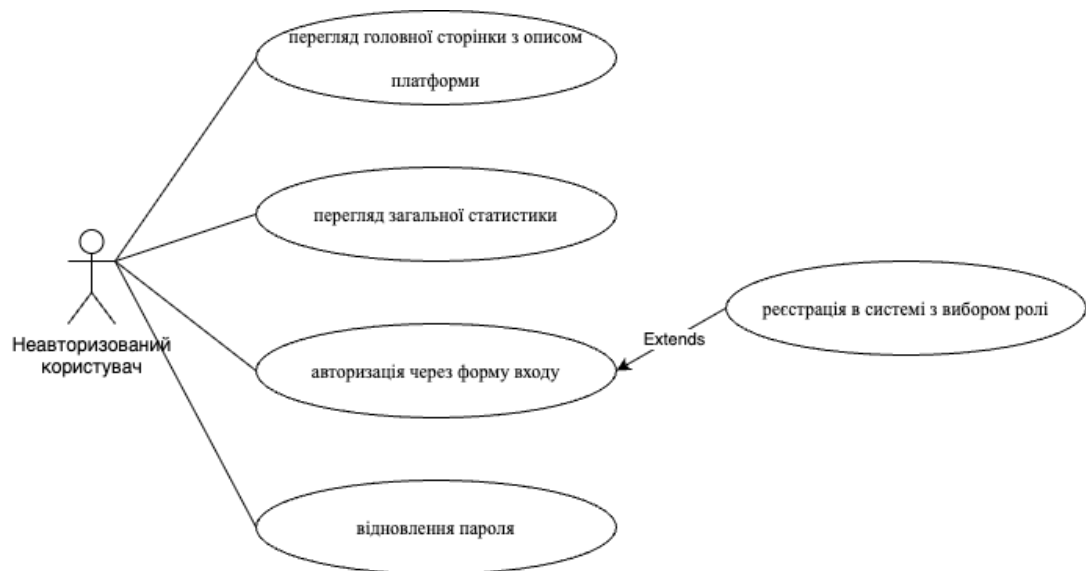


Рисунок 2.1 – Діаграма прецедентів програмного забезпечення підтримки формування команд для занять футболом для неавторизованого користувача

Центральною фігурою виступає гравець, кількість яких становитиме більшу частину користувацької бази. Гравці разом із організаторами турнірів генерують основну активність платформи.

Гравець створює розширений профіль, що включає обов’язкові поля особистої інформації: ім’я, вік, стать, місцезнаходження, а також спортивні характеристики – рівень навичок, досвід гри та переважну позицію на полі. Реєстрація відбувається у два етапи: спочатку створюється базовий обліковий запис, потім заповнюється детальний спортивний профіль. Система передбачає обов’язкову верифікацію через електронну пошту та можливість завантаження медіа-контенту для формування візуального спортивного портфоліо.

Командна діяльність гравця включає широкий спектр можливостей: від створення власних команд з автоматичним отриманням статусу капітана до участі в існуючих колективах через систему заявок та запрошень. Капітани отримують розширені права управління складом, включаючи обробку заявок на вступ та надсилання персональних запрошень. Кожна команда може створити власну ідентичність через завантаження логотипу та опис діяльності. Гравці зберігають свободу вибору – вони можуть залишити команду в будь-

який момент, а капітани мають право розформувати неактивний колектив.

Турнірна система надає гравцям можливість пошуку змагань з гнучкою фільтрацією за статусом проведення, форматом гри та розміром призового фонду. Після подачі командної заявки на участь гравець може відстежувати її статус через персональний кабінет, а також переглядати турнірні таблиці та результати.

Для організації тренувань та товариських матчів платформа пропонує інтегровану систему бронювання майданчиків. Геолокаційний пошук з інтерактивною картою дозволяє швидко знайти доступні об'єкти поблизу. Календарне бронювання забезпечує вибір зручних часових слотів, а персональний кабінет надає повний контроль над історією та статусами бронювань з можливістю скасування за необхідності.

Соціальний компонент платформи реалізується через розширені можливості пошуку інших гравців за спортивними критеріями – рівнем гри, географічним розташуванням та ігровою позицією. Користувачі можуть переглядати детальні профілі та статистику інших учасників, а також брати участь у формуванні репутації майданчиків через систему рейтингів та відгуків.

Гравцю має бути доступний наступний функціонал ПЗ (рис. 2.2):

- створення та редагування профілю з медіа-контентом;
- пошук інших гравців за критеріями (рівень, місце проживання, позиція);
- пошук команд та подачу заявок на вступ;
- створення власних команд та управління складом;
- надсилання запрошень іншим гравцям до команди;
- пошук доступних майданчиків;
- бронювання майданчиків на певний час;
- пошук активних турнірів;
- подача заявок на участь команди в турнірах;
- перегляд персональної статистики матчів;
- завантаження фото та відео з матчів;

- оцінювання якості майданчиків;
- перегляд історії бронювань та матчів.



Рисунок 2.2 – Діаграма прецедентів програмного забезпечення підтримки формування команд для занять футболом для гравця

Третім типом користувачів є організатор турнірів – каталізатор змагальної активності платформи. Він отримує комплексний інструментарій для управління турнірами, починаючи від створення змагань з детальною конфігурацією параметрів. Система підтримує різні формати проведення – олімпійську систему на вибування, кругову лігову систему та швейцарську систему. Організатор встановлює фінансові параметри турніру, включаючи розмір вступних внесків та структуру призового фонду, визначає часові

рамки – дедлайн реєстрації, дати початку та завершення змагань, а також максимальну кількість команд-учасниць.

Управління заявками відбувається через зручну панель, де організатор може переглядати детальну інформацію про команди-претенденти та приймати рішення щодо їх допуску до турніру. Після формування складу учасників організатор планує розклад матчів, призначає майданчики для кожної гри та вносить результати зустрічей. Система автоматично веде турнірну таблицю та статистику, а організатор має можливість управляти рейтингами використаних майданчиків на основі якості проведених матчів.

Організатору турнірів має бути доступний наступний функціонал ПЗ (рис. 2.3):

- створення нових турнірів з налаштуваннями (формат, дати, призовий фонд);
- управління заявками команд;
- створення розкладу матчів;
- призначення майданчиків для матчів;
- введення результатів матчів;
- оцінювання якості майданчиків;
- управління статистикою турніру.

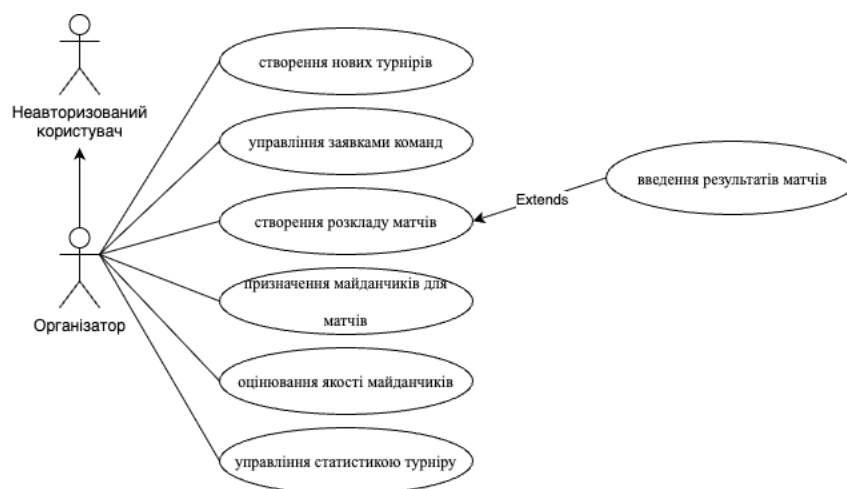


Рисунок 2.3 – Діаграма прецедентів програмного забезпечення підтримки формування команд для занять футболом для організатора

Четвертий тип користувача – власник майданчику він забезпечує матеріальну базу для всієї екосистеми. Власники надають доступ до необхідної інфраструктури для проведення матчів та тренувань (рис. 2.4).

Управління спортивними об'єктами починається зі створення детальних карток майданчиків, що включають всю необхідну інформацію: назву, вартість оренди за годину, тип покриття, розміри поля та опис. Візуальна презентація забезпечується через галерею фотографій, що допомагає користувачам оцінити якість об'єкта до бронювання.

Система бронювань надає власнику повний контроль над процесом резервування. Всі вхідні запити потрапляють до єдиної панелі управління, де власник може детально розглянути кожну заявку та прийняти рішення про підтвердження або відхилення. Візуалізація завантаженості через календарний інтерфейс допомагає оптимізувати використання майданчику та уникнути можливих накладок у розкладі.

Бізнес-аналітика забезпечує власника всіма необхідними даними для ефективного управління: статистика використання показує популярні часи та дні тижня, фінансові звіти демонструють доходи та рентабельність об'єкту, а система відгуків та рейтингів надає зворотний зв'язок від користувачів для постійного покращення якості послуг.



Рисунок 2.4 – Діаграма прецедентів програмного забезпечення підтримки формування команд для занять футболом для власника майданчиків

Така чотирирівнева типологія користувачів створює збалансовану екосистему, де кожен учасник отримує цінність від взаємодії з іншими. Гравці знаходять команди та турніри, організатори отримують активних учасників для своїх змагань, а власники майданчиків забезпечують стабільний потік клієнтів через інтегровану систему бронювань.

2.2 Визначення засобів для розробки програмного забезпечення підтримки формування команд для занять футболом

Розробка комплексної екосистемної платформи для аматорського футболу вимагає ретельного підходу до вибору технологічних рішень. Аналіз функціональних вимог, визначених у попередньому розділі, дозволив сформулювати ключові критерії для оцінки потенційних технологій: масштабованість для підтримки зростаючої користувацької бази, типобезпека для мінімізації помилок у складній бізнес-логіці, продуктивність для забезпечення швидкого відгуку системи, можливість інтеграції з геопросторовими сервісами для забезпечення функціональності пошуку майданчиків, а також підтримку real-time взаємодії для оновлення статистики матчів.

З огляду на зазначені критерії, особливу увагу слід приділити вибору архітектурного стилю системи та відповідних технологічних стеків.

Серверна частина платформи є критичним компонентом, що забезпечує виконання бізнес-логіки, управління даними та інтеграцію з зовнішніми сервісами. Для вибору оптимального рішення було проведено порівняльний аналіз трьох провідних Node.js фреймворків, які найкраще відповідають вимогам проєкту (табл. 2.1) [10]-[12].

На основі проведеного аналізу обрано NestJS як основний серверний фреймворк. Його модульна архітектура та нативна підтримка TypeScript забезпечують оптимальну основу для розробки складної бізнес-логіки екосистемної платформи.

Таблиця 2.1 – Порівняння Node.js фреймворків для серверної розробки

Критерій	NestJS	Express.js	Fastify
Архітектурний підхід	Модульна архітектура з Dependency Injection	Мінімалістичний, middleware-based	Плагін-орієнтована архітектура
TypeScript підтримка	Нативна, повна інтеграція	Потребує додаткової конфігурації	Часткова підтримка
Продуктивність	Висока	Середня	Висока
Тестування	Вбудовані інструменти	Потребує сторонніх бібліотек	Базова підтримка
Масштабованість коду	Відмінна (модулі, сервіси)	Потребує власної організації	Добра (через плагіни)
Документація	Детальна, з прикладами	Мінімальна	Добра
Розмір спільноти	56 тисяч уподобань на GitHub	62 тисяч уподобань на GitHub	29 тисяч уподобань на GitHub

Система управління базами даних (СУБД) має критичне значення для забезпечення надійного зберігання та швидкого доступу до даних платформи. Особливу увагу приділено можливостям роботи з геопросторовими даними, необхідними для функції пошуку майданчиків.

Для вибору оптимального рішення було проведено порівняльний аналіз трьох реляційних СУБД, які найкраще відповідають вимогам проекту (табл. 2.2) [13]-[15].

Вибір PostgreSQL обумовлений унікальними можливостями PostGIS для геопросторових операцій, підтримкою JSONB для гнучкого зберігання метаданих та потужним оптимізатором запитів для складних операцій.

Таблиця 2.2 – Порівняння реляційних СУБД

Критерій	PostgreSQL	MySQL	MariaDB
Геопросторові функції	Повний набір	Базові функції	Базові функції
JSON підтримка	Є	Є	Є
Продуктивність JOIN	Відмінна	Добра	Добра
Транзакції	Повна ACID, MVCC	ACID з обмеженнями	ACID з обмеженнями
Розширюваність	Власні типи, функції	Обмежена	Обмежена
Хмарна підтримка	Всі провайдери	Всі провайдери	Обмежена

Вибір PostgreSQL обумовлений унікальними можливостями PostGIS для геопросторових операцій, підтримкою JSONB для гнучкого зберігання метаданих та потужним оптимізатором запитів для складних операцій.

Клієнтська частина платформи відповідає за користувацький інтерфейс та взаємодію із серверним Application Programming Interface (API). Даний вибір впливає на швидкість розробки, продуктивність та користувацький досвід.

Для вибору оптимального рішення порівняємо frontend фреймворки (табл. 2.3) [16]-[18].

React обрано, зважаючи на потужну екосистему, гнучкість в архітектурних рішеннях та відмінну продуктивності через Virtual Document Object Model (DOM), що критично важливо для динамічних оновлень статистики матчів.

Таблиця 2.3 – Порівняння frontend фреймворків

Критерій	React	Angular	Vue.js
Архітектура	Бібліотека, компонентна	Повний фреймворк, Model-View-Controller	Прогресивний фреймворк
TypeScript	Опціональний	Обов'язковий	Опціональний
Продуктивність	Висока	Середня	Висока
Екосистема	Величезна	Комплексна	Збалансована
Компонентна модель	Функціональні + класові	Класові	Об'єктні
State Management	Redux, MobX, Context	RxJS, NgRx	Vuex

2.3 Структура бази даних програмного забезпечення підтримки формування команд для занять футболом

Проектування структури бази даних для футбольної платформи базується на результатах аналізу функціональних вимог та обраного технологічного стеку. Модель даних розроблена відповідно до принципів реляційного проектування з використанням третьої нормальної форми для забезпечення цілісності даних та мінімізації надлишковості.

База даних організована навколо п'яти логічних доменів: управління користувачами, командоутворення, турнірна система, інфраструктурне забезпечення та медіа-контент. Кожен домен містить набір взаємопов'язаних таблиць, що моделюють відповідні бізнес-процеси платформи.

Таблиця UserCredentials зберігає автентифікаційні дані користувачів системи та має наступні стовпці:

– id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор запису, генерується автоматично;

- email (VARCHAR (255), UNIQUE, NOT NULL) – електронна адреса користувача, використовується як логін для входу в систему;
- password (VARCHAR (255), NOT NULL) – хешований пароль користувача, зберігається з використанням bcrypt алгоритму;
- refreshToken (VARCHAR (255), NULL) – токен для оновлення сесії, дозволяє підтримувати автентифікацію без повторного введення паролю;
- confirmToken (VARCHAR (255), NULL) – токен для підтвердження електронної пошти, генерується при реєстрації;
- confirmEmail (BOOLEAN, NOT NULL, DEFAULT FALSE) – статус підтвердження електронної адреси;
- resetPasswordToken (VARCHAR (255), NULL) – токен для відновлення паролю, має обмежений термін дії;
- createdAt (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата та час створення облікового запису.

Таблиця UserTypes є довідником типів користувачів системи та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор типу користувача;
- type_name (VARCHAR (50), NOT NULL) – назва типу користувача (PLAYER, ORGANIZER, VENUE_OWNER).

Таблиця User є центральною сутністю системи, що пов'язує автентифікаційні дані з типом користувача та профільною інформацією. Вона має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор користувача;
- credentials (INTEGER, FOREIGN KEY REFERENCES UserCredentials (id), ON DELETE CASCADE) – посилання на автентифікаційні дані;
- type (INTEGER, FOREIGN KEY REFERENCES UserTypes (id)) – тип користувача, визначає доступну функціональність.

Таблиця GamerProfile містить розширену інформацію про користувачів типу PLAYER та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор профілю;
- user_id (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE, UNIQUE) – посилання на користувача;
- name (VARCHAR (100), NULL) – повне ім'я гравця;
- age (INTEGER, NULL) – вік гравця, використовується для підбору команд відповідного рівня;
- sex (VARCHAR(10), NULL) – стать гравця (male/female);
- location (VARCHAR (255), NULL) – місце проживання, використовується для геолокаційного пошуку;
- skill_lvl (INTEGER, NULL) – рівень майстерності від 1 до 10;
- experience (VARCHAR (50), NULL) – досвід гри (beginner, amateur, semi-pro, professional);
- position (VARCHAR (50), NULL) – основна позиція на полі (goalkeeper, defender, midfielder, forward).

Таблиця Team зберігає інформацію про футбольні команди, створені на платформі та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор команди;
- name (VARCHAR (100), NOT NULL) – назва команди, має бути унікальною в межах платформи;
- description (TEXT, NULL) – опис команди, її цілі та філософія гри;
- logo_url (VARCHAR (255), NULL) – шлях до файлу логотипу команди;
- is_active (BOOLEAN, NOT NULL, DEFAULT TRUE) – статус активності команди, використовується замість фізичного видалення;
- captain_id (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE) – користувач-капітан з розширеними правами управління;
- created_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата створення команди.

Таблиця `team_members` є зв'язковою таблицею для реалізації зв'язку many-to-many між командами та гравцями. Вона має наступні стовпці:

- `team_id` (INTEGER, FOREIGN KEY REFERENCES Team (id), ON DELETE CASCADE) – ідентифікатор команди;
- `user_id` (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE) – ідентифікатор гравця;
- PRIMARY KEY (`team_id`, `user_id`) – композитний первинний ключ, що забезпечує унікальність членства.

Таблиця `TeamInvitation` управляє процесом запрошення нових гравців до команди капітанами та має наступні стовпці:

- `id` (INTEGER, PRIMARY KEY) – унікальний ідентифікатор запрошення;
- `team_id` (INTEGER, FOREIGN KEY REFERENCES Team (id), ON DELETE CASCADE) – команда, що запрошує;
- `invitee_email` (VARCHAR (255), NOT NULL) – email запрошеного гравця;
- `status` (VARCHAR (20), NOT NULL, DEFAULT 'pending') – статус запрошення (pending, accepted, rejected);
- `invited_by` (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE) – користувач, що надіслав запрошення;
- `created_at` (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата створення запрошення.

Таблиця `TeamJoinRequest` обробляє заявки гравців на вступ до команд та має наступні стовпці:

- `id` (INTEGER, PRIMARY KEY) – унікальний ідентифікатор заявки;
- `team_id` (INTEGER, FOREIGN KEY REFERENCES Team (id), ON DELETE CASCADE) – команда, до якої подається заявка;
- `requester_id` (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE) – гравець, що подає заявку;

– status (VARCHAR (20), NOT NULL, DEFAULT 'pending') – статус заявки (pending, accepted, rejected);

– created_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата подачі заявки.

Таблиця Tournament містить інформацію про турніри, організовані на платформі та має наступні стовпці:

– id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор турніру;

– name (VARCHAR (200), NOT NULL) – назва турніру;

– description (TEXT, NULL) – детальний опис турніру, правила та умови участі;

– format (VARCHAR (50), NOT NULL) – формат проведення (knockout, league, swiss);

– maxTeams (INTEGER, NOT NULL) – максимальна кількість команд-учасниць;

– registrationDeadline (DATE, NOT NULL) – крайній термін реєстрації команд;

– startDate (DATE, NOT NULL) – дата початку турніру;

– endDate (DATE, NOT NULL) – планована дата завершення;

– status (VARCHAR (20), NOT NULL, DEFAULT 'upcoming') – статус турніру (upcoming, ongoing, completed, cancelled);

– entryFee (DECIMAL (10,2), NULL) – вступний внесок за участь;

– prizePool (DECIMAL (10,2), NULL) – загальний призовий фонд;

– organizer (INTEGER, FOREIGN KEY REFERENCES User (id)) – організатор турніру;

– created_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата створення турніру;

– updated_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP) – дата останнього оновлення.

Таблиця TournamentApplication управляє заявками команд на участь у турнірах та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор заявки;
- tournament_id (INTEGER, FOREIGN KEY REFERENCES Tournament (id), ON DELETE CASCADE) – турнір;
- team_id (INTEGER, FOREIGN KEY REFERENCES Team (id), ON DELETE CASCADE) – команда-заявник;
- status (VARCHAR (20), NOT NULL, DEFAULT 'pending') – статус заявки (pending, approved, rejected);
- applied_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата подачі заявки.

Таблиця Match зберігає інформацію про матчі в рамках турнірів та має наступні стовпці:

- id (UUID, PRIMARY KEY) – унікальний ідентифікатор матчу у форматі UUID;
- tournamentId (INTEGER, FOREIGN KEY REFERENCES Tournament (id)) – турнір, в рамках якого проводиться матч;
- team1Name (VARCHAR (100), NOT NULL) – назва першої команди (денормалізована для швидкого доступу);
- team2Name (VARCHAR (100), NOT NULL) – назва другої команди;
- team1Id (INTEGER, FOREIGN KEY REFERENCES Team (id), NULL) – ідентифікатор першої команди;
- team2Id (INTEGER, FOREIGN KEY REFERENCES Team (id), NULL) – ідентифікатор другої команди;
- scheduledDate (TIMESTAMP, NOT NULL) – запланована дата та час проведення;
- venue (VARCHAR (255), NULL) – місце проведення матчу;
- matchFormat (VARCHAR (50), NULL) – формат матчу (90 хвилин, 2x45, 2x30);

- status (ENUM ('scheduled', 'in_progress', 'completed', 'cancelled'), NOT NULL, DEFAULT 'scheduled') – статус матчу;
- team1Score (INTEGER, NULL) – кількість голів першої команди;
- team2Score (INTEGER, NULL) – кількість голів другої команди;
- round (VARCHAR (50), NULL) – раунд турніру (1/16, 1/8, quarterfinal, semifinal, final);
- notes (TEXT, NULL) – додаткові примітки до матчу;
- createdAt (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата створення запису;
- updatedAt (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP) – дата оновлення.

Таблиця MatchStatistic фіксує детальну статистику подій матчу та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор події;
- matchId (UUID, FOREIGN KEY REFERENCES Match (id), ON DELETE CASCADE) – матч;
- teamName (VARCHAR (100), NOT NULL) – назва команди, до якої належить подія;
- playerName (VARCHAR (100), NOT NULL) – ім'я гравця, пов'язаного з подією;
- type (ENUM ('goal', 'assist', 'yellow_card', 'red_card', 'substitution_in', 'substitution_out'), NOT NULL) – тип події;
- minute (INTEGER, NULL) – хвилина матчу, коли сталася подія;
- assistedBy (VARCHAR (100), NULL) – ім'я гравця-асистента (для голів);
- createdAt (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – час фіксації події.

Таблиця GroundTypes є довідником типів спортивних майданчиків матчу та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор типу;

– type_name (VARCHAR (50), NOT NULL) – назва типу (football_field, futsal_court, artificial_turf).

Таблиця GroundLocationCity містить довідник міст та має стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор міста;
- city (VARCHAR (100), NOT NULL) – назва міста.

Таблиця GroundLocationCountry містить довідник країн та має стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор країни;
- country (VARCHAR (100), NOT NULL) – назва країни.

Таблиця GroundLocation комбінує місто та країну для повної адреси та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор локації;
- city_id (INTEGER, FOREIGN KEY REFERENCES GroundLocationCity (id), ON DELETE CASCADE) – місто;
- country_id (INTEGER, FOREIGN KEY REFERENCES GroundLocationCountry (id), ON DELETE CASCADE) – країна.

Таблиця GroundCredentials містить детальну інформацію про спортивні майданчики та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор майданчика;
- title (VARCHAR (200), NOT NULL) – назва майданчика;
- position (JSON, NOT NULL) – географічні координати {latitude, longitude};
- photo_url (VARCHAR(255), NULL) – URL основної фотографії майданчика;
- price_per_hour (DECIMAL (10,2), NOT NULL, DEFAULT 0) – вартість оренди за годину;
- type (INTEGER, FOREIGN KEY REFERENCES GroundTypes (id), ON DELETE CASCADE) – тип майданчика;
- location (INTEGER, FOREIGN KEY REFERENCES GroundLocation (id), ON DELETE CASCADE) – розташування.

Таблиця GroundOwner пов'язує майданчики з їх власниками та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор зв'язку;
- ownerId (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE) – власник майданчика;
- groundId (INTEGER, FOREIGN KEY REFERENCES GroundCredentials (id), ON DELETE CASCADE) – майданчик.

Таблиця VenueBooking управляє бронюваннями майданчиків та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор бронювання;
- venue_id (INTEGER, FOREIGN KEY REFERENCES GroundCredentials (id)) – майданчик;
- user_id (INTEGER, FOREIGN KEY REFERENCES User (id), NULL) – користувач, що бронює;
- user_email (VARCHAR (255), NULL) – email для незареєстрованих користувачів;
- booking_date (DATE, NOT NULL) – дата бронювання;
- start_time (TIME, NOT NULL) – час початку;
- end_time (TIME, NOT NULL) – час завершення;
- status (VARCHAR (20), NOT NULL, DEFAULT 'pending') – статус (pending, approved, cancelled);
- created_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата створення бронювання.

Таблиця VenueRating зберігає оцінки майданчиків від користувачів та має наступні стовпці:

- id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор оцінки;
- match_id (UUID, FOREIGN KEY REFERENCES Match (id), NULL) – пов'язаний матч;

– ground_id (INTEGER, FOREIGN KEY REFERENCES GroundCredentials (id)) – майданчик;

– player_id (INTEGER, FOREIGN KEY REFERENCES User (id)) – користувач, що оцінює;

– player_name (VARCHAR (100), NOT NULL) – ім'я користувача;

– rating (INTEGER, NOT NULL, CHECK (rating >= 1 AND rating <= 5)) – оцінка від 1 до 5;

– comments (TEXT, NULL) – текстовий відгук;

– created_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата створення оцінки.

Таблиця Media управляє медіафайлами, завантаженими користувачами, та має наступні стовпці:

– id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор файлу;

– title (VARCHAR (200), NOT NULL) – назва файлу;

– description (TEXT, NULL) – опис контенту;

– file_path (VARCHAR (500), NOT NULL) – шлях до файлу на сервері;

– file_type (VARCHAR (10), NOT NULL) – тип файлу (image, video);

– mime_type (VARCHAR (50), NOT NULL) – MIME-тип (image/jpeg, video/mp4);

– file_size (INTEGER, NOT NULL) – розмір файлу в байтах;

– match_date (DATE, NULL) – дата матчу, до якого відноситься контент;

– tags (TEXT, NULL) – JSON масив тегів для пошуку;

– is_public (BOOLEAN, NOT NULL, DEFAULT TRUE) – публічність контенту;

– uploaded_by (INTEGER, FOREIGN KEY REFERENCES User (id), ON DELETE CASCADE) – користувач-власник;

– team_id (INTEGER, FOREIGN KEY REFERENCES Team (id), ON DELETE CASCADE, NULL) – пов'язана команда;

– uploaded_at (TIMESTAMP, NOT NULL, DEFAULT CURRENT_TIMESTAMP) – дата завантаження (рис. 2.5).

2.4 Висновки за розділом 2

В результаті виконання розділу 2 завершено комплексний етап проєктування платформи організації аматорського футболу, в результаті якого розроблено детальну технічну архітектуру системи, готової для практичної реалізації. В ході проведеної роботи послідовно було трансформовано концептуальні ідеї, сформульовані в першому розділі, у конкретні проєктні рішення, які охоплюють всі аспекти майбутньої системи.

Вибір технологічного стеку ґрунтувався на системному підході до оцінки доступних рішень. В результаті порівняльного аналізу трьох ключових категорій (серверна платформа, СУБД, клієнтський фреймворк) було обрано оптимальну комбінацію технологій. Це NestJS, що забезпечує модульну архітектуру та типобезпеку на сервері, PostgreSQL з PostGIS, які надають потужні геопросторові можливості для пошуку майданчиків, React, який створює динамічний користувацький інтерфейс, та TypeScript, який на всіх рівнях мінімізує кількість помилок під час розробки. Кожне технологічне рішення обґрунтовано через призму специфічних вимог платформи, що гарантує їх оптимальність для поставлених завдань.

Розроблена проєктна документація дозволяє перейти до наступного етапу реалізації розробки програмного забезпечення. Обґрунтований технологічний стек дозволяє досягти високого рівня ефективності розробки та подальшої підтримки програми, а наявна структура бази даних мінімізує можливі ризики виникнення архітектурних помилок на етапі програмування. Комплексність та системність проведеного проєктування підтверджують готовність платформи до практичної реалізації.

3 РОЗРОБКА ПРОГРАМИ

3.1 Визначення порядку роботи з програмним забезпеченням підтримки формування команд для занять футболом

Розроблена платформа для підтримки аматорського футболу реалізує комплексний підхід до управління спортивною діяльністю через вебдодаток з роль-орієнтованим інтерфейсом. Програма забезпечує взаємодію чотирьох типів користувачів: неавторизованих відвідувачів, зареєстрованих гравців, організаторів турнірів та власників спортивних майданчиків.

Порядок роботи з програмою включає наступні етапи:

- ініціалізацію сесії з завантаженням головної сторінки та загальної статистики платформи;
- автентифікацію користувача через реєстрацію з вибором ролі або авторизацію з використанням email та пароля;
- верифікацію облікового запису через підтвердження електронної пошти токеном;
- налаштування профілю відповідно до обраної ролі користувача;
- виконання основної діяльності згідно з функціональними можливостями ролі;
- завершення сесії з безпечним виходом та очищенням токенів автентифікації.

Алгоритм взаємодії з програмою передбачає централізоване управління станом через React Context API, автоматичне оновлення JSON Web Token (JWT), геопросторовий пошук майданчиків через PostGIS, real-time оновлення турнірних сіток та інтеграцію з картографічними сервісами для візуалізації локацій спортивних об'єктів.

Технологічна архітектура програми побудована на принципах модульності з чітким розподілом відповідальності між компонентами системи (рис. 3.1).

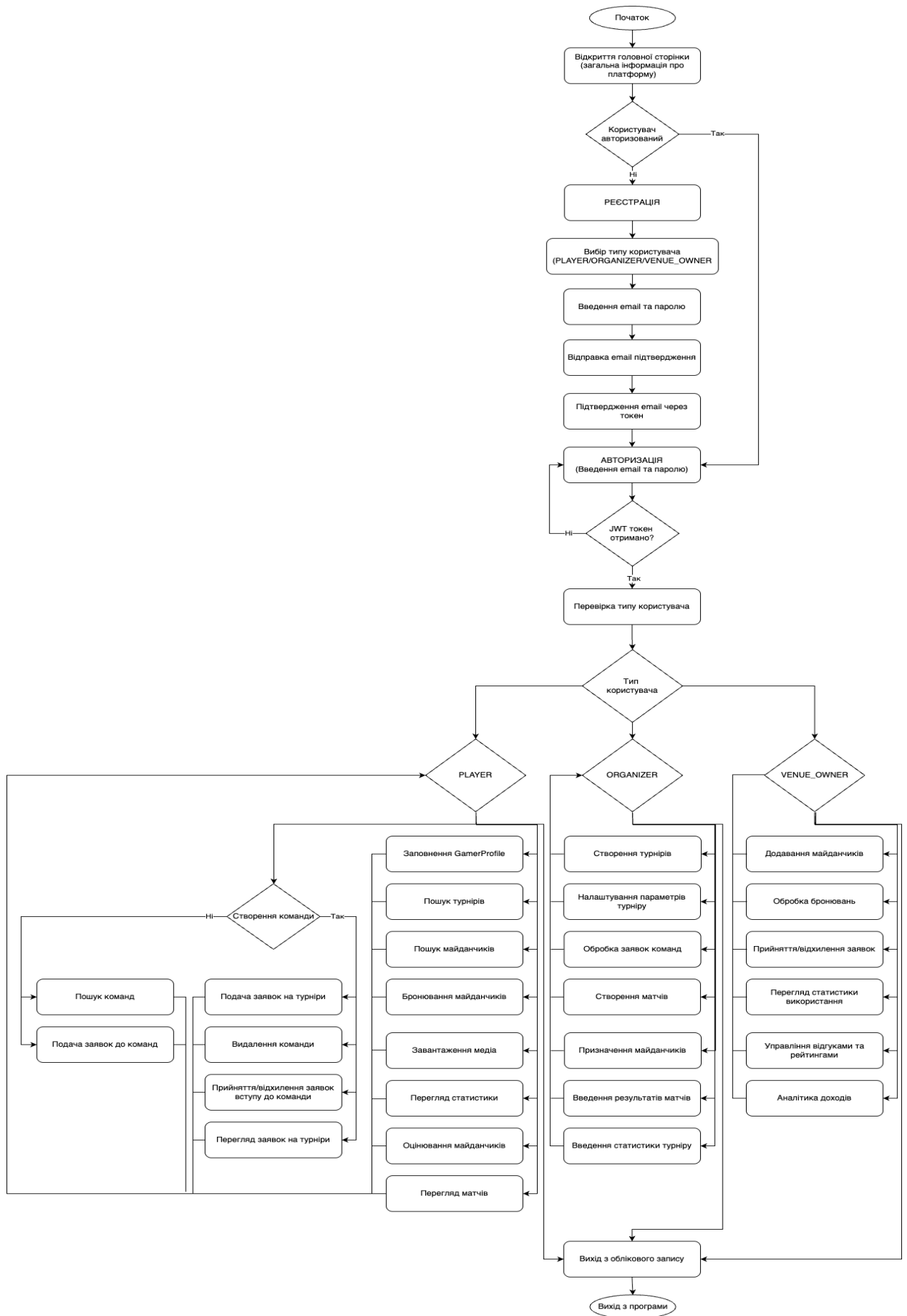


Рисунок 3.1 – Схема функціонування ПЗ підтримки формування команд для занять футболом

Фронтенд-частина реалізована з використанням React TypeScript з styled-components для стилізації, забезпечуючи адаптивний користувацький інтерфейс та seamless взаємодію між різними модулями. Бекенд-система побудована на NestJS фреймворку з Type Object-Relational Mapping (Type ORM) для роботи з PostgreSQL базою даних, що гарантує масштабованість, безпеку та високу продуктивність системи.

Безпека та автентифікація в програмі реалізована через багаторівневу систему захисту, включаючи JWT токени з автоматичним оновленням, хешування паролів з використанням bcrypt, верифікацію email адрес через одноразові токени та роль-орієнтований контроль доступу. Кожна операція в системі проходить валідацію на рівні контролерів, сервісів та бази даних, забезпечуючи цілісність даних та захист від несанкціонованого доступу до чутливої інформації користувачів та фінансових даних турнірів.

3.2 Реалізація основних класів програмного забезпечення підтримки формування команд для занять футболом

Модель користувачів програми для підтримки формування команд для занять футболом була реалізована сукупністю взаємопов'язаних класів, що забезпечують повний життєвий цикл управління даними користувачів, команд, майданчиків та турнірної діяльності. Архітектура класів побудована на принципах нормалізації бази даних з використанням TypeORM для забезпечення цілісності та ефективності зберігання інформації.

Клас User реалізує основну сутність користувача системи з атрибутами: id типу INTEGER з декоратором @PrimaryGeneratedColumn як унікальний ідентифікатор користувача. Зв'язки включають credentials типу ManyToOne з UserCredentials для автентифікаційних даних, type типу ManyToOne з UserTypes для визначення ролі користувача та gamerProfile типу OneToOne з GamerProfile для розширеної інформації про гравця.

Клас `UserCredentials` зберігає автентифікаційні дані з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn`, `email` типу `VARCHAR` з декоратором `@Column ({unique: true})` для унікальної електронної адреси, `password` типу `VARCHAR` для хешованого паролю, `refreshToken` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для токена оновлення сесії, `confirmToken` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для токена підтвердження `email`, `confirmEmail` типу `BOOLEAN` з декоратором `@Column ({default: false})` для статусу підтвердження, `resetPasswordToken` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для токена відновлення паролю та `createdAt` типу `TIMESTAMP` з декоратором `@CreateDateColumn`.

Клас `UserTypes` містить довідник типів користувачів з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn` та `type_name` типу `VARCHAR` з декоратором `@Column` для назви ролі (`PLAYER`, `ORGANIZER`, `VENUE_OWNER`).

Клас `GamerProfile` розширює інформацію про гравців з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn`, `name` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для повного імені, `age` типу `INTEGER` з декоратором `@Column ({nullable: true})` для віку гравця, `sex` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для статі, `location` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для місця проживання, `skill_lvl` типу `INTEGER` з декоратором `@Column ({nullable: true})` для рівня майстерності від 1 до 10, `experience` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для досвіду гри та `position` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для ігрової позиції. Зв'язки включають `user_id` типу `OneToOne` з `User` для прив'язки до основного профілю користувача.

Клас `Team` реалізує сутність футбольної команди з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn`, `name` типу `VARCHAR` з декоратором `@Column` для назви команди, `description` типу `TEXT` з

декоратором `@Column ({nullable: true})` для опису команди, `logo_url` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для посилання на логотип, `is_active` типу `BOOLEAN` з декоратором `@Column ({default: true})` для статусу активності та `created_at` типу `TIMESTAMP` з декоратором `@CreateDateColumn`. Зв'язки включають `captain` типу `ManyToOne` з `User` для капітана команди з розширеними правами та `members` типу `ManyToMany` з `User` через `join table 'team_members'` для списку учасників.

Клас `TeamInvitation` управляє запрошеннями до команд з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn`, `invitee_email` типу `VARCHAR` з декоратором `@Column` для email запрошеного гравця, `status` типу `VARCHAR` з декоратором `@Column ({default: 'pending'})` для статусу запрошення (`pending`, `accepted`, `rejected`) та `created_at` типу `TIMESTAMP` з декоратором `@CreateDateColumn`. Зв'язки включають `team` типу `ManyToOne` з `Team` для команди що запрошує та `invited_by` типу `ManyToOne` з `User` для користувача, що надіслав запрошення.

Клас `TeamJoinRequest` обробляє заявки на вступ до команд з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn`, `status` типу `VARCHAR` з декоратором `@Column ({default: 'pending'})` для статусу заявки та `created_at` типу `TIMESTAMP` з декоратором `@CreateDateColumn`. Зв'язки включають `team` типу `ManyToOne` з `Team` для команди-цілі та `requester` типу `ManyToOne` з `User` для гравця-заявника.

Клас `GroundCredentials` зберігає інформацію про спортивні майданчики з атрибутами: `id` типу `INTEGER` з декоратором `@PrimaryGeneratedColumn`, `title` типу `VARCHAR` з декоратором `@Column` для назви майданчика, `position` типу `JSON` з декоратором `@Column ({type: 'json'})` для геопросторових координат у форматі `{latitude, longitude}`, `photo_url` типу `VARCHAR` з декоратором `@Column ({nullable: true})` для URL основної фотографії та `price_per_hour` типу `DECIMAL` з декоратором `@Column ({type: 'decimal', precision: 10, scale: 2, default: 0})` для вартості оренди. Зв'язки включають `type` типу `ManyToOne` з

GroundTypes для типу майданчика та location типу ManyToOne з GroundLocation для розташування.

Клас GroundOwner пов'язує майданчики з власниками з атрибутами: id типу INTEGER з декоратором @PrimaryGeneratedColumn. Зв'язки включають ownerId типу ManyToOne з User для власника майданчика та groundId типу ManyToOne з GroundCredentials для майданчика.

Клас Tournament реалізує управління турнірами з атрибутами: id типу INTEGER з декоратором @PrimaryGeneratedColumn, name типу VARCHAR з декоратором @Column для назви турніру, description типу TEXT з декоратором @Column ({type: 'text', nullable: true}) для детального опису, format типу VARCHAR з декоратором @Column для формату проведення (knockout, league, swiss), maxTeams типу INTEGER з декоратором @Column для максимальної кількості команд, registrationDeadline типу DATE з декоратором @Column ({type: 'date'}) для крайнього терміну реєстрації, startDate типу DATE з декоратором @Column ({type: 'date'}) для дати початку, endDate типу DATE з декоратором @Column ({type: 'date'}) для дати завершення, status типу VARCHAR з декоратором @Column ({default: 'upcoming'}) для статусу турніру, entryFee типу DECIMAL з декоратором @Column ({type: 'decimal', precision: 10, scale: 2, nullable: true}) для вступного внеску, prizePool типу DECIMAL з декоратором @Column ({type: 'decimal', precision: 10, scale: 2, nullable: true}) для призового фонду, createdAt типу TIMESTAMP з декоратором @CreateDateColumn та updatedAt типу TIMESTAMP з декоратором @UpdateDateColumn. Зв'язки включають organizer типу ManyToOne з User з декоратором @JoinColumn ({eager: true}), applications типу OneToMany з TournamentApplication та matches типу OneToMany з Match.

Клас TournamentApplication управляє заявками команд з атрибутами: id типу UUID з декоратором @PrimaryGeneratedColumn ('uuid'), tournamentId типу INTEGER з декоратором @Column, captainId типу INTEGER з декоратором @Column, teamName типу VARCHAR з декоратором @Column,

message типу TEXT з декоратором @Column ({type: 'text', nullable: true}) для додаткового повідомлення, status типу ENUM з декоратором @Column ({type: 'enum', enum: ApplicationStatus, default: ApplicationStatus.PENDING}) для статусу заявки, rejectionReason типу VARCHAR з декоратором @Column ({nullable: true}) для причини відхилення, createdAt типу TIMESTAMP з декоратором @CreateDateColumn та updatedAt типу TIMESTAMP з декоратором @UpdateDateColumn. Зв'язки включають tournament типу ManyToOne з Tournament та captain типу ManyToOne з User.

Клас Match зберігає інформацію про матчі з атрибутами: id типу UUID з декоратором @PrimaryGeneratedColumn ('uuid'), tournamentId типу INTEGER з декоратором @Column, team1Name типу VARCHAR з декоратором @Column, team2Name типу VARCHAR з декоратором @Column, team1Id типу INTEGER з декоратором @Column ({nullable: true}), team2Id типу INTEGER з декоратором @Column ({nullable: true}), scheduledDate типу TIMESTAMP з декоратором @Column ({type: 'timestamp'}), venue типу VARCHAR з декоратором @Column ({nullable: true}), matchFormat типу VARCHAR з декоратором @Column ({nullable: true}), status типу ENUM з декоратором @Column ({type: 'enum', enum: MatchStatus, default: MatchStatus.SCHEDULED}), team1Score типу INTEGER з декоратором @Column ({nullable: true}), team2Score типу INTEGER з декоратором @Column ({nullable: true}), round типу VARCHAR з декоратором @Column ({nullable: true}), notes типу TEXT з декоратором @Column ({type: 'text', nullable: true}), createdAt типу TIMESTAMP з декоратором @CreateDateColumn та updatedAt типу TIMESTAMP з декоратором @UpdateDateColumn. Зв'язки включають tournament типу ManyToOne з Tournament, team1 типу ManyToOne з Team з декоратором @JoinColumn ({nullable: true}), team2 типу ManyToOne з Team з декоратором @JoinColumn ({nullable: true}) та venueRatings типу OneToMany з VenueRating.

Клас MatchStatistic фіксує події матчу з атрибутами: id типу INTEGER з декоратором @PrimaryGeneratedColumn, matchId типу UUID з декоратором

@Column, teamName типу VARCHAR з декоратором @Column, playerName типу VARCHAR з декоратором @Column, type типу ENUM з декоратором @Column ({type: 'enum', enum: StatisticType}) для типу події (GOAL, ASSIST, YELLOW_CARD, RED_CARD, SUBSTITUTION_IN, SUBSTITUTION_OUT), minute типу INTEGER з декоратором @Column ({nullable: true}) для хвилини події, assistedBy типу VARCHAR з декоратором @Column ({nullable: true}) для асистента та createdAt типу TIMESTAMP з декоратором @CreateDateColumn. Зв'язки включають match типу ManyToOne з Match.

Клас VenueRating зберігає оцінки майданчиків з атрибутами: id типу INTEGER з декоратором @PrimaryGeneratedColumn, rating типу INTEGER з декоратором @Column та @Check для обмеження значень від 1 до 5, comment типу TEXT з декоратором @Column ({type: 'text', nullable: true}), venueName типу VARCHAR з декоратором @Column, matchId типу UUID з декоратором @Column, userId типу INTEGER з декоратором @Column, groundId типу INTEGER з декоратором @Column ({nullable: true}) та createdAt типу TIMESTAMP з декоратором @CreateDateColumn. Зв'язки включають match типу ManyToOne з Match, user типу ManyToOne з User та ground типу ManyToOne з GroundCredentials з декоратором @JoinColumn ({nullable: true}). Клас має унікальний індекс @Index (['match', 'user']) для запобігання дублюванню оцінок.

Клас Media управляє медіафайлами з атрибутами: id типу INTEGER з декоратором @PrimaryGeneratedColumn, title типу VARCHAR з декоратором @Column для назви файлу, description типу TEXT з декоратором @Column ({type: 'text', nullable: true}) для опису, file_path типу VARCHAR з декоратором @Column для шляху до файлу, file_type типу VARCHAR з декоратором @Column для типу файлу (image, video), mime_type типу VARCHAR з декоратором @Column для MIME-типу, file_size типу INTEGER з декоратором @Column для розміру в байтах, match_date типу DATE з декоратором @Column ({type: 'date', nullable: true}), tags типу TEXT з декоратором @Column ({type: 'text', nullable: true}) для JSON масиву тегів, is_public типу BOOLEAN з

декоратором `@Column ({default: true})` для публічності та `uploaded_at` типу `TIMESTAMP` з декоратором `@CreateDateColumn`. Зв'язки включають `uploaded_by` типу `ManyToOne` з `User` та `team` типу `ManyToOne` з `Team` з декоратором `@JoinColumn ({nullable: true})`.

3.3 Реалізація основних програмних функцій програмного забезпечення підтримки формування команд для занять футболом

Програмне забезпечення підтримки формування команд для занять футболом реалізує комплексну систему функцій, що охоплюють весь спектр операцій з управління користувачами, командами, майданчиками та турнірною діяльністю. Функціональна архітектура побудована на основі сервіс-орієнтованого підходу з використанням `NestJS` фреймворку, що забезпечує модульність, масштабованість та підтримуваність системи.

Автентифікація користувачів підтримується функцією `register` з параметрами `email` типу `string`, `password` типу `string`, `userType` типу `string` та `res` типу `ExpressResponse`, яка повертає `Promise <any>` та створює новий обліковий запис з хешуванням паролю через `bcrypt` алгоритм, генерацією `confirmToken` для `email` підтвердження, збереженням у `UserCredentials` таблиці та відправкою `email` через `MailService` з `HTML` шаблонами.

Розширена реєстрація гравців підтримується функцією `register_profile` з параметрами `email` типу `string`, `name` типу `string`, `age` типу `number`, `sex` типу `string`, `location` типу `string`, `skill_lvl` типу `number`, `experience` типу `string` та опціональним `position` типу `string`, яка повертає `Promise <any>` та заповнює `GamerProfile` таблицю з валідацією `skill_lvl` від 1 до 10, збереженням ігрових характеристик та створенням зв'язку `OneToOne` з `User`.

Перевірка облікових даних підтримується функцією `login` з параметрами `email` типу `string`, `password` типу `string` та `res` типу `ExpressResponse`, яка повертає `Promise <any>` та виконує `bcrypt.compare` для валідації паролю, генерує `access` та `refresh JWT` токени з 15-хвилинним та 7-

денним терміном дії, оновлює refreshToken поле у UserCredentials таблиці, встановлює HTTP-only cookies та повертає користувацькі дані з типом ролі.

Підтвердження електронної пошти підтримується функцією confirmUserEmail з параметром token типу string, яка повертає Promise <UserCredentials | null> та знаходить користувача через WHERE умову на confirmToken, встановлює confirmEmail у true, очищає confirmToken поле та зберігає зміни для активації облікового запису.

Відновлення паролю підтримується функціями refreshPassword з параметрами email типу string, password типу string та опціональним access_token типу string, яка генерує resetPasswordToken через crypto.randomBytes, зберігає його у базі даних та надсилає email з посиланням через MailService, та confirmNewPassword з параметрами token типу string і password типу string для підтвердження нового паролю з bcrypt хешуванням.

Управління командами підтримується функцією createTeam з параметрами userEmail типу string, name типу string та опціональним description типу string, яка повертає Promise <any> та створює Team запис з автоматичним призначенням створювача як captain через ManyToOne зв'язок, додає його до members через ManyToMany зв'язок з join table 'team_members', встановлює is_active у true та зберігає created_at timestamp.

Запрошення гравців підтримується функцією inviteToTeam з параметрами teamId типу number, inviterEmail типу string та inviteeEmail типу string, яка повертає Promise <any> та перевіряє права капітана через WHERE умову captain_id, валідує існування команди та запрошеного користувача, створює TeamInvitation запис зі статусом 'pending', надсилає email запрошення з HTML шаблоном та зберігає invited_by зв'язок.

Обробка запрошень підтримується функцією respondToInvitation з параметрами invitationId типу number, userEmail типу string та accept типу boolean, яка повертає Promise <any> та оновлює статус запрошення, при accept = true додає користувача до members через INSERT INTO

team_members, при accept = false видаляє запрошення з TeamInvitation таблиці та надсилає сповіщення результату.

Подача заявок на вступ підтримується функцією sendJoinRequest з параметрами teamId типу number та userEmail типу string, яка повертає Promise <any> та створює TeamJoinRequest запис з валідацією існування команди через JOIN до Team, перевіркою на дублювання через WHERE умови на team_id та requester_id, встановленням статусу 'pending' та збереженням created_at timestamp.

Обробка заявок підтримується функцією respondToJoinRequest з параметрами requestId типу number, userEmail типу string та accept типу boolean, яка повертає Promise <any> та перевіряє права капітана команди через JOIN запит, оновлює статус заявки, при схваленні додає користувача до members та видаляє заявку з TeamJoinRequest, надсилає email сповіщення результату.

Пошук команд підтримується функцією searchAvailableTeams з параметром userEmail типу string, яка повертає Promise <any> та виконує LEFT JOIN між Team та team_members таблицями, фільтрує команди де користувач не є членом через WHERE NOT EXISTS, застосовує is_active = true умову та повертає список доступних команд з captain інформацією.

Завантаження логотипів підтримується функцією uploadTeamLogo з параметрами teamId типу number, userEmail типу string та logoPath типу string, яка повертає Promise <any> та перевіряє права капітана, валідує формат файлу через mime-type перевірку (.jpg, .png, .gif), оновлює logo_url поле у Team таблиці та зберігає шлях до файлу у uploads/teams директорії.

Покидання команди підтримується функцією leaveTeam з параметрами teamId типу number та userEmail типу string, яка повертає Promise <any> та видаляє користувача з team_members через DELETE FROM запит, перевіряє чи не є користувач капітаном команди, оновлює склад команди та зберігає зміни у базі даних.

Розпуск команд підтримується функцією `disbandTeam` з параметрами `teamId` типу `number` та `userEmail` типу `string`, яка повертає `Promise <any>` та перевіряє права капітана через `captain_id` порівняння, встановлює `is_active` у `false` замість фізичного видалення для збереження історії, очищає всі `members` зв'язки та зберігає команду для архівних цілей.

Отримання членів команди підтримується функцією `getTeamMembers` з параметром `teamId` типу `number`, яка повертає `Promise <any>` та виконує JOIN запити до `User`, `UserCredentials` та `GamerProfile` таблиць, повертає деталізовану інформацію про всіх учасників включаючи `name`, `position`, `skill_lvl`, `experience` та `captain` статус.

Створення майданчиків підтримується функцією `createGround` з параметрами `email` типу `string`, `title` типу `string`, `position` об'єкт з координатами `{latitude, longitude}`, `type` типу `string` та `price_per_hour` типу `number`, яка повертає `Promise <any>` та валідує геопросторові координати, створює `GroundCredentials` запис, встановлює власника через `GroundOwner` зв'язок та зберігає локаційні дані у `GroundLocation`.

Управління фотографіями підтримується функцією `updateGroundPhoto` з параметрами `groundId` типу `number` та `photoUrl` типу `string`, яка повертає `Promise <any>` та перевіряє права власника через `GroundOwner` зв'язок, валідує формат зображення, оптимізує файл, зберігає попередню фотографію для `rollback` та оновлює `photo_url` поле.

Редагування майданчиків підтримується функцією `updateGround` з параметрами `id` типу `number`, `email` типу `string` та опціональними `title`, `position`, `type`, `price_per_hour`, яка повертає `Promise <any>` та перевіряє права власника, валідує зміни, оновлює відповідні поля у `GroundCredentials` та зберігає модифікації з `timestamp`.

Бронювання часових слотів підтримується функцією `bookSlot` з параметром `bookingData` об'єкт з полями `groundId` типу `number`, `date` типу `string`, `start_time` та `end_time` типу `string`, `userEmail` типу `string`, яка повертає `Promise <any>` та перевіряє доступність через SELECT запит до `VenueBooking`, валідує

часовий діапазон, створює запис зі статусом 'pending', розраховує вартість та надсилає сповіщення власнику.

Підтвердження бронювань підтримується функцією `updateBookingStatus` з параметрами `userEmail` типу `string`, `bookingId` типу `number` та `status` типу `string`, яка повертає `Promise <any>` та перевіряє права власника майданчика через `GroundOwner` зв'язок, оновлює статус на 'approved' або 'rejected', надсилає email сповіщення користувачу та записує зміни у `VenueBooking`.

Скасування бронювань підтримується функцією `cancelUserBooking` з параметрами `userEmail` типу `string` та `bookingId` типу `number`, яка повертає `Promise <any>` та валідує право скасування, оновлює статус на 'cancelled', надсилає сповіщення власнику майданчика, обробляє повернення коштів при необхідності та записує причину скасування.

Геопросторовий пошук підтримується функцією `getAvailableGrounds` з параметрами `date` типу `string` та `time` типу `string`, яка повертає `Promise <any[]>` та виконує складний PostGIS запит з `ST_Distance` функцією, JOIN до `VenueBooking` для фільтрації зайнятих майданчиків, WHERE умовами на дату та час, ORDER BY відстані та повертає список доступних об'єктів з координатами.

Аналітика майданчиків підтримується функцією `getAnalytics` з параметром `userEmail` типу `string`, яка повертає `Promise <any>` та розраховує статистичні показники через агрегаційні SQL функції SUM, COUNT, AVG, включаючи загальні доходи, кількість бронювань, коефіцієнт підтвердження, місячну динаміку та популярні часові слоти.

Організація турнірів підтримується функцією `createTournament` з параметрами `tournamentData` об'єкт та `organizerEmail` типу `string`, яка повертає `Promise<Tournament>` та валідує дати через DATE порівняння, створює `Tournament` запис, встановлює `organizer` зв'язок з `User`, розраховує `prizePool` на основі `entryFee` та `maxTeams`, встановлює статус 'upcoming' та зберігає метадані.

Управління турнірами підтримується функціями `updateTournament` з параметрами `id` типу `number` та `updateData` об'єкт, яка оновлює `Tournament` поля з валідацією прав організатора, та `deleteTournament` з параметром `id` типу `number`, яка встановлює статус `'cancelled'` замість фізичного видалення для збереження статистики.

Заявки на турніри підтримується функцією `createApplication` з параметрами `captainEmail` типу `string` та `createApplicationDto` об'єкт, яка повертає `Promise <TournamentApplication>` та валідує права капітана команди, перевіряє `registrationDeadline` через DATE порівняння, створює заявку зі статусом `PENDING`, генерує `UUID` ідентифікатор та зберігає у `TournamentApplication`.

Обробка заявок підтримується функцією `updateApplicationStatus` з параметрами `applicationId` типу `string`, `organizerEmail` типу `string` та `updateStatusDto` об'єкт, яка повертає `Promise <TournamentApplication>` та перевіряє права організатора турніру, оновлює статус на `APPROVED` або `REJECTED`, зберігає `rejectionReason` при відхиленні та надсилає email сповіщення капітану.

Створення матчів підтримується функцією `createMatch` з параметрами `createMatchDto` об'єкт та `organizerEmail` типу `string`, яка повертає `Promise <Match>` та створює матч з `UUID` ідентифікатором, прив'язує до турніру через `tournamentId`, встановлює команди через `team1Id` та `team2Id` зв'язки, призначає `venue` та `scheduledDate` з валідацією конфліктів.

Оновлення результатів підтримується функцією `updateMatchResult` з параметрами `matchId` типу `string`, `resultDto` об'єкт та `organizerEmail` типу `string`, яка повертає `Promise <any>` та оновлює `team1Score` і `team2Score`, створює `MatchStatistic` записи для кожної події (голи, асисті, картки), перераховує турнірні таблиці для league формату та визначає переможців knockout турнірів.

Статистика матчів підтримується функцією `getMatchStatistics` з параметром `matchId` типу `string`, яка повертає `Promise <any>` та агрегує статистику через `GROUP BY` запити до `MatchStatistic`, включає голи, асисті,

картки та заміни з деталізацією по командах та гравцях, сортує за minute події та повертає структуровані дані.

Статистика гравців підтримується функцією `getPlayerStatistics` з параметром `userEmail` типу `string`, яка повертає `Promise <any>` та виконує складні JOIN запити до `MatchStatistic`, `Match`, `TournamentApplication` таблиць, розраховує індивідуальні показники включаючи голи, асисті, картки, зіграні матчі, групує по типах подій та повертає агреговані результати.

Оцінювання майданчиків підтримується функцією `createOrUpdateRating` з параметрами `matchId` типу `string`, `userId` типу `number`, `rating` типу `number`, `venueName` типу `string` та опціональними `comment` і `groundId`, яка повертає `Promise <VenueRating>` та створює або оновлює оцінку з валідацією діапазону 1-5, перевіряє унікальність через `Index (['match', 'user'])` та зберігає коментар.

Середні оцінки підтримується функцією `getVenueAverageRating` з параметром `venueName` типу `string`, яка повертає `Promise <{averageRating: number, totalRatings: number}>` та розраховує середню оцінку через AVG агрегаційну функцію, підраховує загальну кількість через COUNT та групує за `venueName`.

Управління медіаконтентом підтримується функцією `uploadMedia` з параметрами `userEmail`, `title`, `description`, `filePath`, `fileType`, `contentType`, `fileSize` типів `string/number` та опціональними `matchDate`, `tags`, `teamId`, яка повертає `Promise <any>` та валідує формат файлу, створює `Media` запис, встановлює `uploaded_by` зв'язок з `User`, опціонально прив'язує до `Team`, зберігає метадані та організовує по тегах JSON формату.

3.4 Висновки за розділом 3

Реалізація програми підтримки формування команд для занять футболом забезпечила повну функціональність всіх заявлених можливостей системи. Розроблені класи та функції охоплюють весь спектр діяльності

користувачів платформи – від базової автентифікації до складних алгоритмів організації турнірів.

Серверна архітектура на основі NestJS продемонструвала ефективність модульного підходу, де кожен сервіс (AuthService, TeamService, TournamentService, VenueBookingService) відповідає за чітко визначену область функціональності. Програма успішно реалізує всі визначені в проєкті функціональні вимоги та готова до практичного використання в екосистемі аматорського футболу.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення та умови виконання програми підтримки формування команд для занять футболом

ПЗ підтримки формування команд для занять футболом призначено для автоматизації організації аматорського футболу через створення цифрової екосистеми, що об'єднує гравців, організаторів турнірів та власників спортивних майданчиків. Основне функціональне призначення полягає у вирішенні проблем фрагментарності футбольної спільноти через забезпечення ефективного командування, координації турнірної діяльності та оптимального використання спортивної інфраструктури.

Програма реалізує чотири ключові напрямки автоматизації: управління користувачами з роль-орієнтованим доступом, підтримку процесів командування через систему пошуку та запрошень, комплексне управління турнірами від реєстрації до підведення підсумків та календарне бронювання спортивних об'єктів з інтеграцією картографічних сервісів. Додатково система забезпечує ведення статистики гравців та команд, рейтингування якості майданчиків та медіа-архівування спортивних подій.

Для експлуатації програми особливих вимог не висувається, програма функціонує в архітектурі клієнт-сервер з вебінтерфейсом, що забезпечує доступ через стандартні браузері без необхідності встановлення спеціалізованого програмного забезпечення.

4.2 Логіка експлуатації програми підтримки формування команд для занять футболом

Взаємодія користувачів з програмою розпочинається з головної сторінки, що містить презентацію основних можливостей платформи та форми входу в систему (рис. 4.1). Неавторизовані користувачі мають доступ лише до загальної інформації про платформу та можливості створення нового

облікового запису, оскільки вся функціональність системи доступна виключно після авторизації.

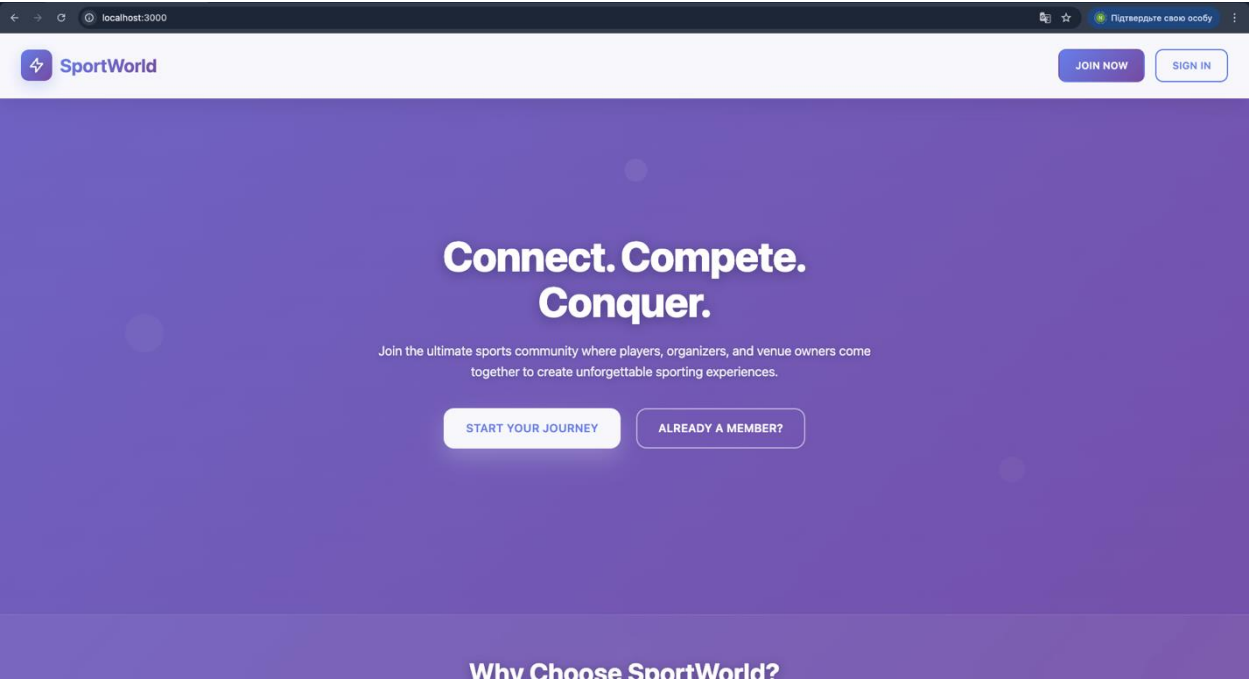


Рисунок 4.1 – Головна сторінка програми

Процес реєстрації передбачає обов’язковий вибір одного з трьох типів користувача: гравець, організатор турнірів або власник майданчиків (рис. 4.2).

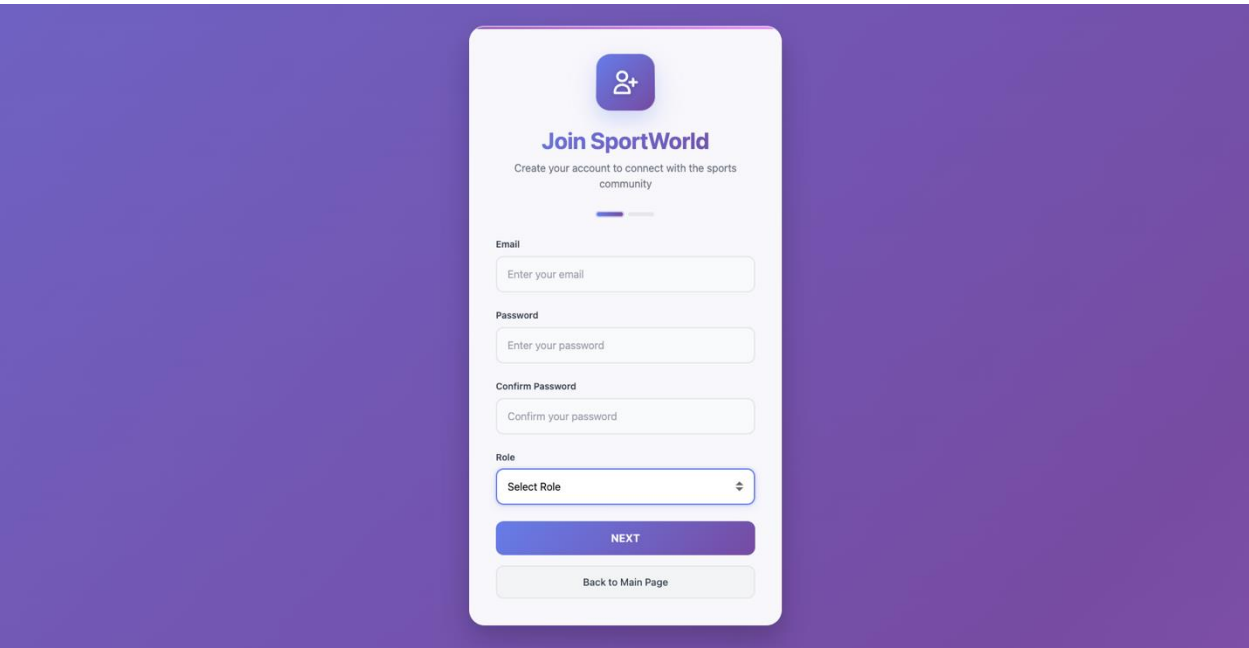


Рисунок 4.2 – Форм реєстрації

Після заповнення реєстраційної форми з email, паролем та типом користувача система автоматично створює базовий профіль та надсилає лист для підтвердження електронної адреси. Після підтвердження email користувач отримує повний доступ до функціональності системи та перенаправляється на відповідну панель управління згідно з обраним типом облікового запису.

Робота з програмою контролюється через роль-орієнтовану систему навігації, що надає доступ до спеціалізованих інструментів відповідно до типу авторизованого користувача. Після входу в систему кожен тип користувача отримує індивідуальну панель управління з персоналізованим меню та функціональністю.

Після авторизації гравець потрапляє на персоналізовану панель управління, що служить центральним хабом для всіх спортивних активностей на платформі (рис. 4.3).

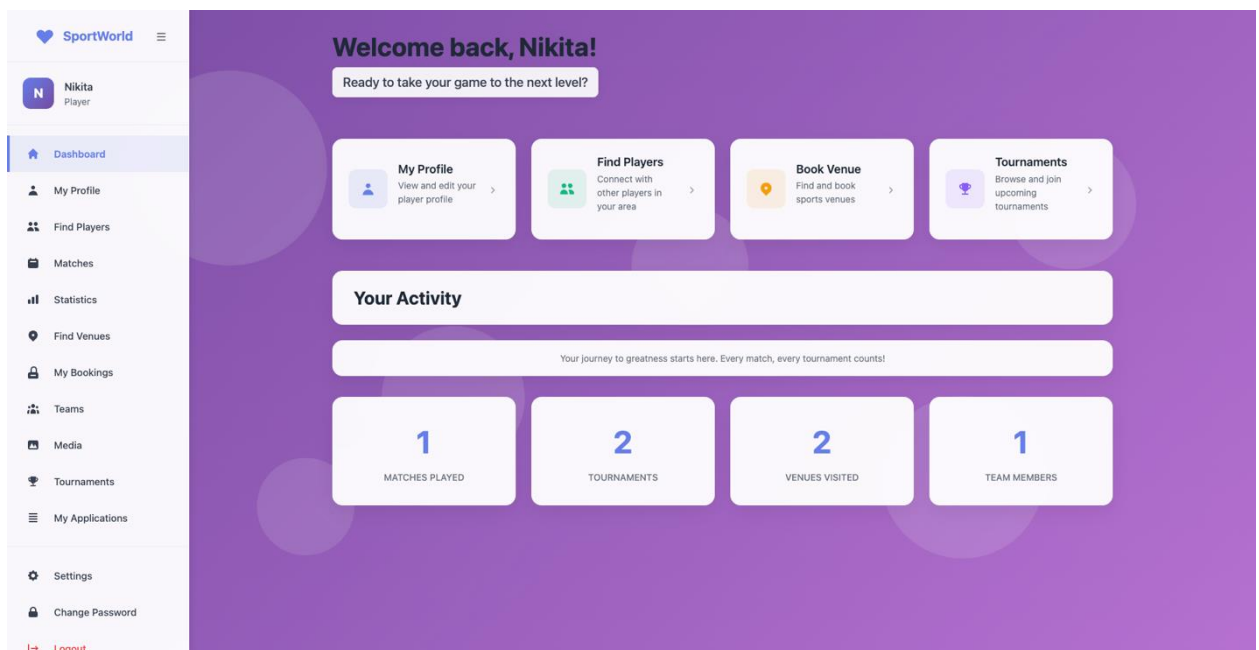


Рисунок 4.3 – Головна сторінка гравця

Головна сторінка гравця надає доступ до основних розділів, що охоплюють повний життєвий цикл спортивної діяльності: від управління особистим профілем до участі в змаганнях та бронювання спортивних об'єктів.

Профіль гравця включає особисті дані, спортивні характеристики, ігрову позицію та досвід (рис. 4.4, рис 4.5). Інтегрована система завантаження медіа-контенту дозволяє гравцям створювати портфолію з фотографіями та відеороликами матчів, що підвищує привабливість профілю для потенційних команд та організаторів турнірів.

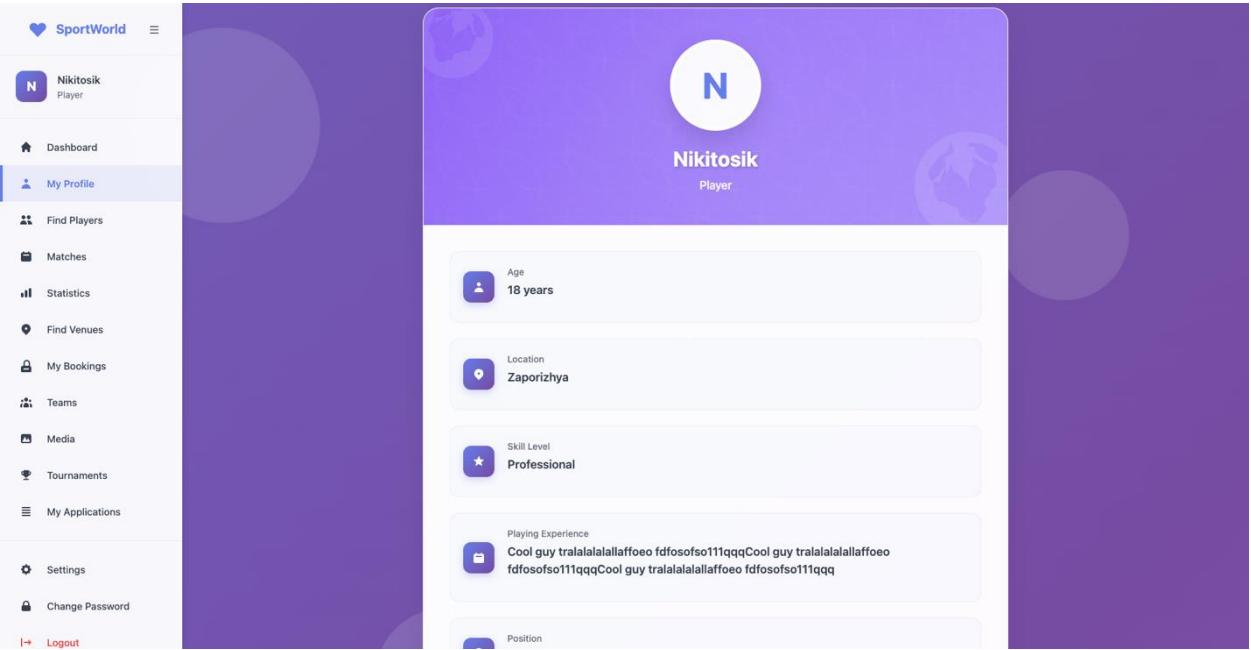


Рисунок 4.4 – Сторінка профілю гравця

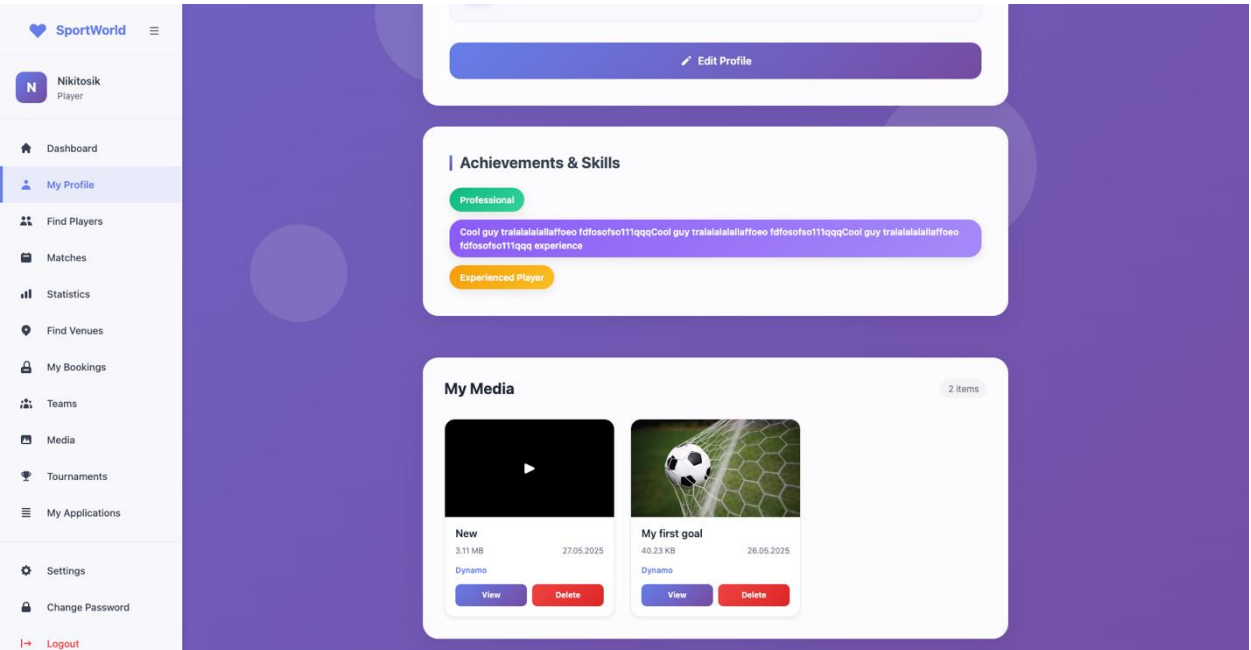


Рисунок 4.5 – Продовження сторінки профілю гравця

Платформа реалізує складну систему командування з множинними рівнями взаємодії. Гравці можуть створювати власні команди, автоматично отримуючи статус капітана з адміністративними привілеями, або приєднуватися до існуючих колективів через систему заявок та запрошень. Функціональність пошуку гравців забезпечує розширені можливості фільтрації за ім'ям, локацією, позицією та рівнем майстерності, відображаючи детальні профілі з медіа-галереями та статистикою активності (рис. 4.6).

Система управління членством включає двосторонній процес: гравці можуть подавати заявки на вступ до команд, а капітани – надсилати прямі запрошення з детальним переглядом профілів кандидатів. Інтелектуальна система попереджає дублювання запрошень та автоматично відфільтровує гравців, що вже перебувають у складі інших команд, забезпечуючи ефективність процесу рекрутингу.

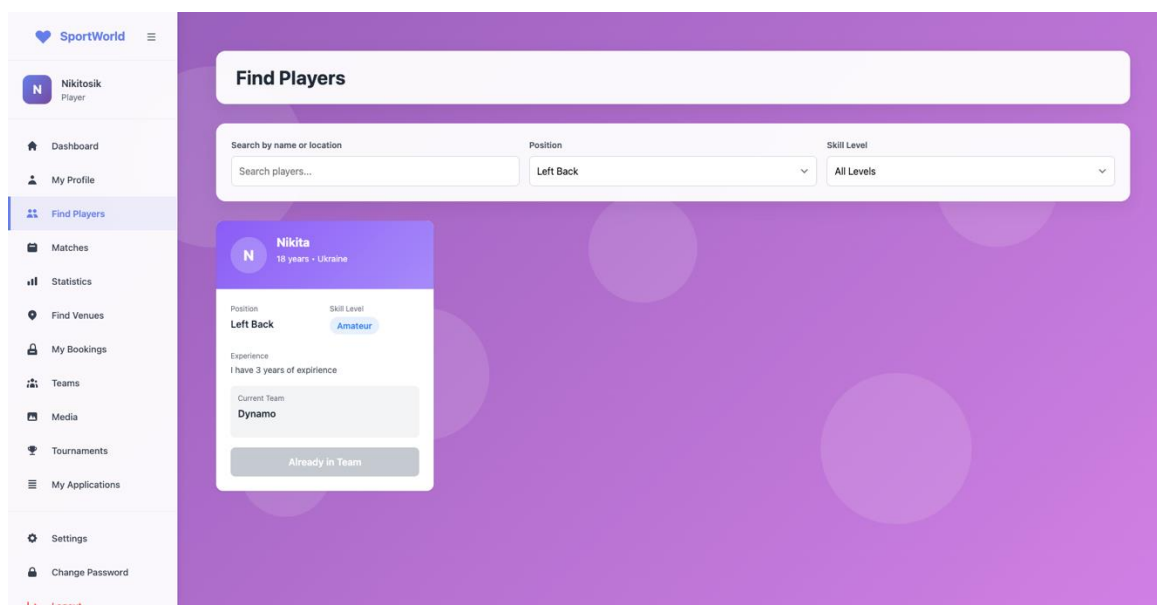


Рисунок 4.6 – Сторінка пошуку гравців з фільтрами та результатами

Турнірна система надає гравцям комплексні можливості участі в організованих змаганнях. Каталог турнірів підтримує розширену фільтрацію за форматом (кубковий, чемпіонатний, швейцарський), статусом проведення, розміром призового фонду та датами проведення (рис. 4.7). Процес подачі

заявок реалізований через спеціалізований модал, що забезпечує структурований збір інформації про команду-заявника та інтеграцію з командною системою для автоматичної валідації права участі.

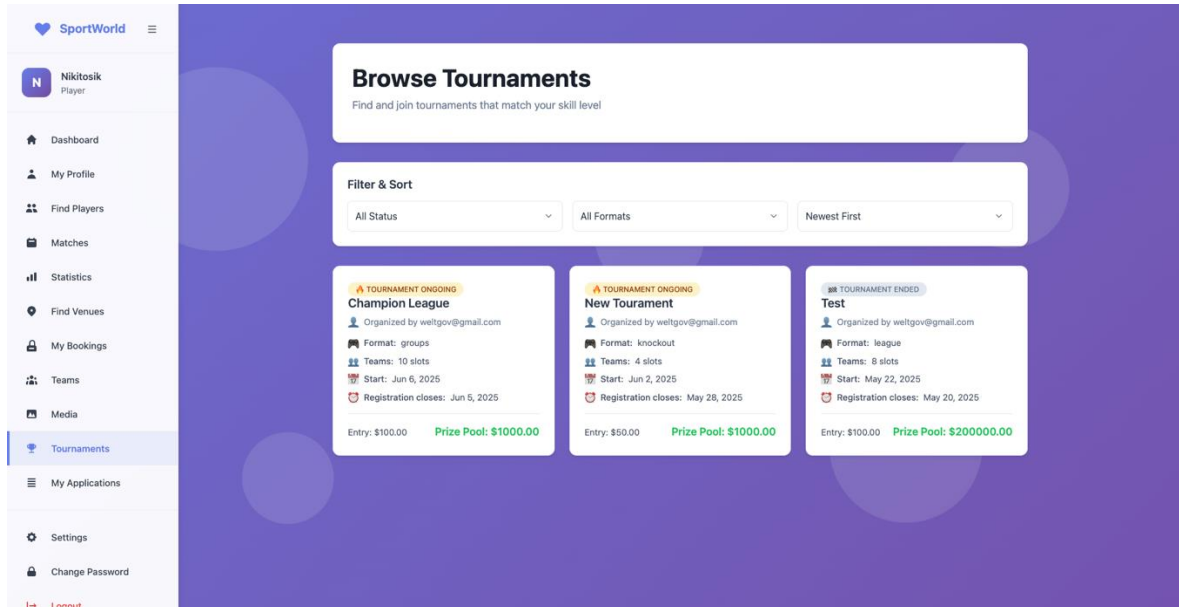


Рисунок 4.7 – Сторінка пошуку та приєднання до турнірів

Функціональність пошуку та бронювання майданчиків реалізовано з інтуїтивним календарним інтерфейсом (рис. 4.8). Інтерактивний календар відображає доступність об'єктів у місячному форматі з індикаторами зайнятості, дозволяючи гравцям швидко знаходити вільні часові слоти. Система запобігання конфліктів бронювання автоматично блокує перетинання бронювань, забезпечуючи коректність планування використання спортивних об'єктів.

Багатоетапний процес бронювання включає подачу запиту, систему підтвердження з боку власників об'єктів та відстеження статусу бронювань. Інтеграція з системою статистики дозволяє відстежувати історію використання майданчиків окремими гравцями та командами для аналітичних цілей.

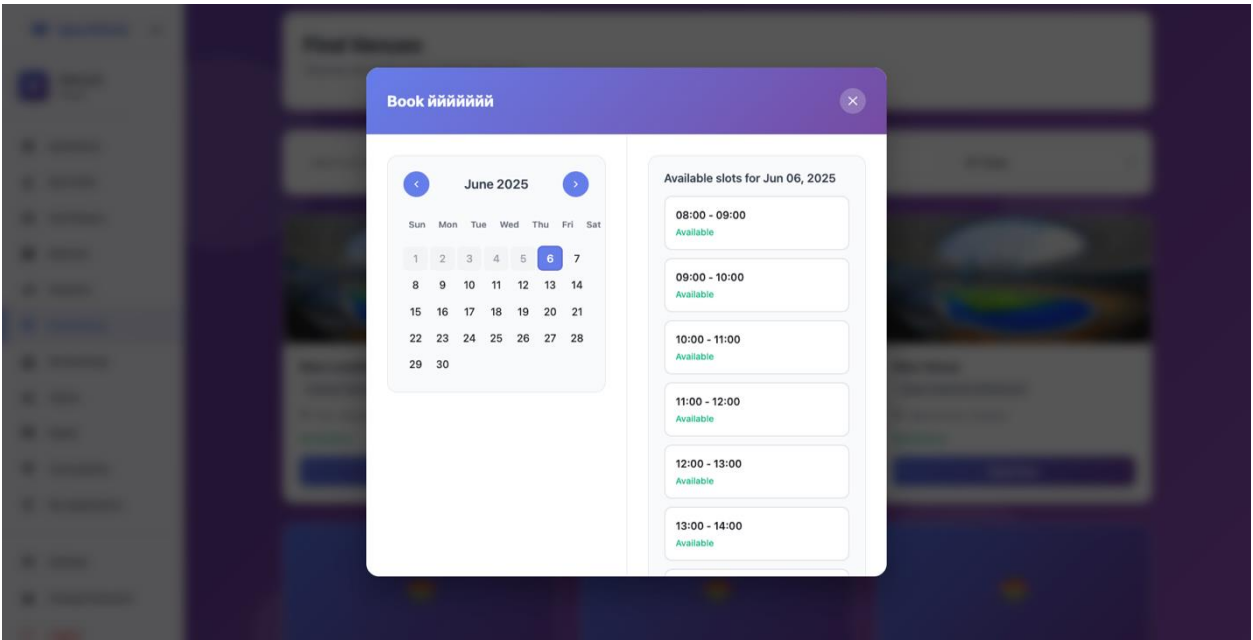


Рисунок 4.8 – Вікно для бронювання майданчиків

Функціональність організатора турнірів представлена багатоаспектною системою управління, що дозволяє спортивним організаторам створювати, координувати та контролювати змагання від початкового планування до підведення підсумків. Головна сторінка організатора інтегрує п’ять ключових компонентів: домашню панель з аналітикою, створення турнірів, портфоліо існуючих змагань, планування матчів та облік результатів (рис. 4.9).

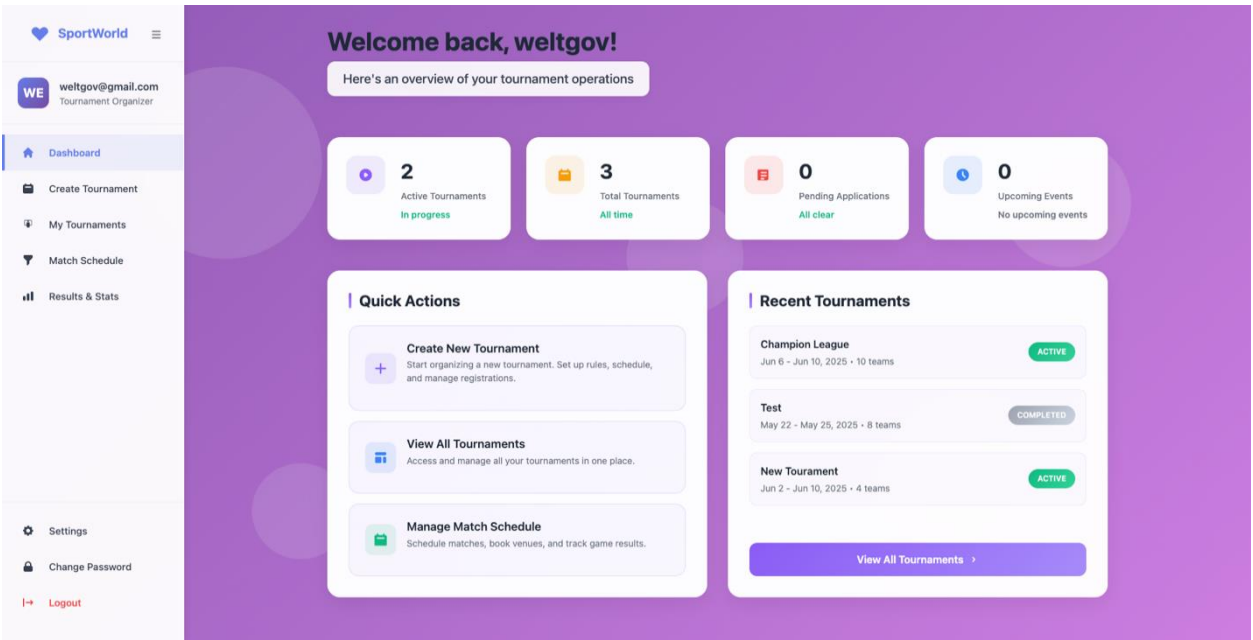


Рисунок 4.9 – Головна сторінка організатора турнірів

Система створення турнірів забезпечує комплексне налаштування параметрів змагань (рис. 4.10). Організатори визначають базову інформацію, включаючи назву, формат та ліміти учасників. Платформа забезпечує інтелектуальну валідацію дат, гарантуючи логічну послідовність: терміни реєстрації передують початку турніру, а дати початку – завершенню змагань. Фінансові параметри включають налаштування реєстраційних внесків та призових фондів з підтримкою десяткової точності для різноманітних моделей монетизації турнірів.

Рисунок 4.10 – Сторінка створення турнірів

Портфоліо турнірів відображається через панель MyTournaments з картковим інтерфейсом, де кожен турнір представлений з повними метаданими: форматом, максимальною кількістю команд, розкладом та динамічними індикаторами статусу (рис. 4.11).

Система розраховує поточний стан турнірів в реальному часі на основі співвідношення з актуальною датою, надаючи організаторам миттєвий ситуаційний контроль.

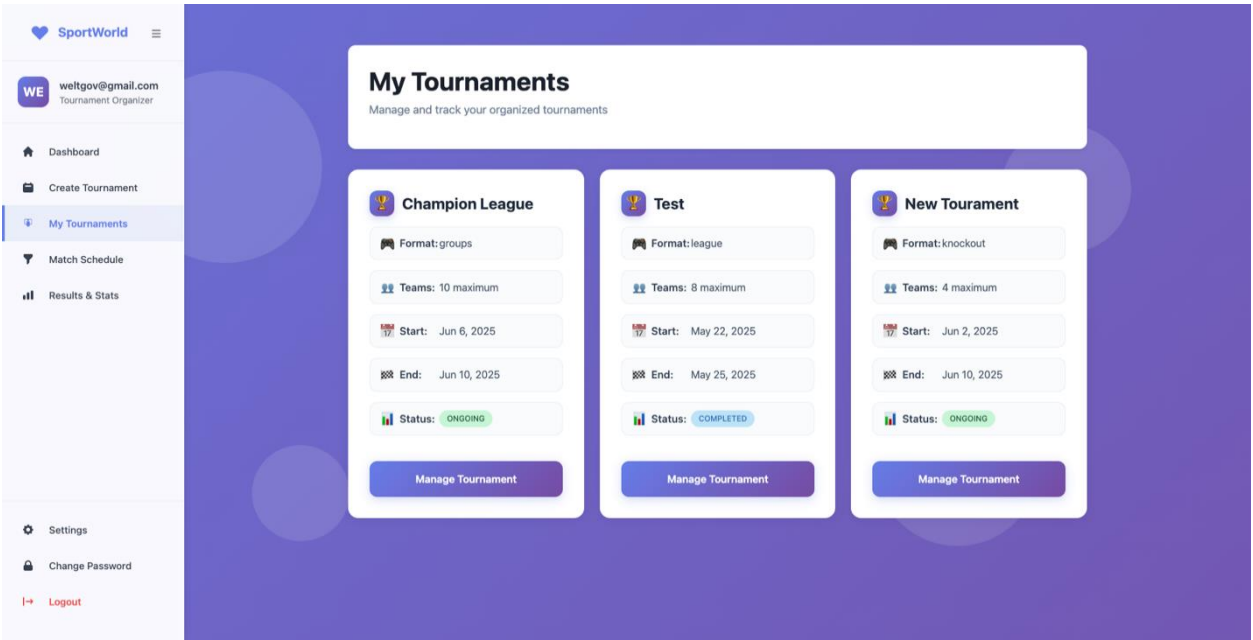


Рисунок 4.11 – Сторінка перегляду власних турнірів

Планування матчів реалізоване з підтримкою подвійного режиму перегляду: списком та календарем (рис. 4.12). Календарний вигляд забезпечує візуальне управління часовою лінією з обмеженнями по датах, запобігаючи плануванню матчів поза межами турніру. Система включає динамічну фільтрацію турнірів, відображаючи лише майбутні змагання для запобігання конфліктів планування.

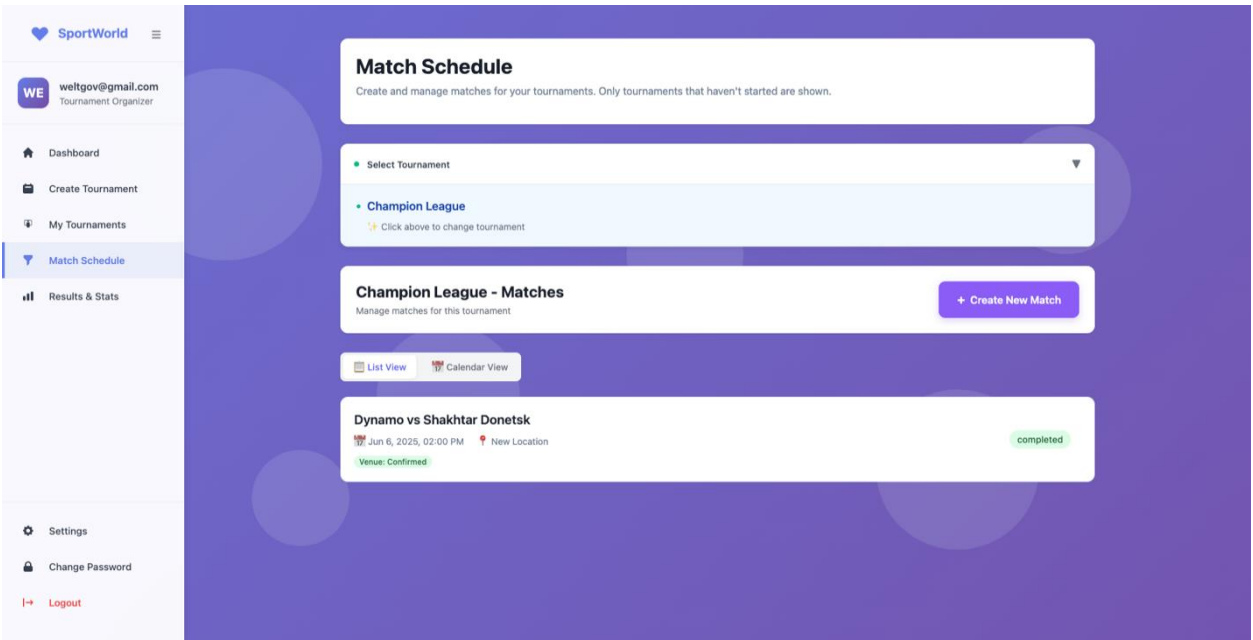


Рисунок 4.12 – Сторінка перегляду та планування матчів

Інтеграція з системою управління майданчиками відбувається через перевірку доступності в реальному часі. Платформа запитує доступні об'єкти на основі дат та часу матчів, представляючи організаторам керований список підходящих споруд. Процес бронювання включає вибір часових слотів з погодинною деталізацією, автоматичне відображення цін та миттєве підтвердження бронювань. Після бронювання інформація про майданчик автоматично інтегрується в записи матчів з оновленням запланованого часу.

Комплексний облік результатів та прогресу змагань забезпечує детальний запис результатів матчів, включаючи рахунки, статистику гравців, авторів голів та примітки до матчу (рис. 4.13). Платформа підтримує різноманітні статистичні категорії та надає структурований ввід даних для комплексного ведення архіву матчів.

Відстеження прогресу змагань реалізовано через систему станів матчів з автоматичним оновленням статусів на основі планування та часового прогресу. Індикатори стану допомагають організаторам підтримувати контроль над часовими рамками змагань та забезпечувати своєчасне проведення турнірних подій.

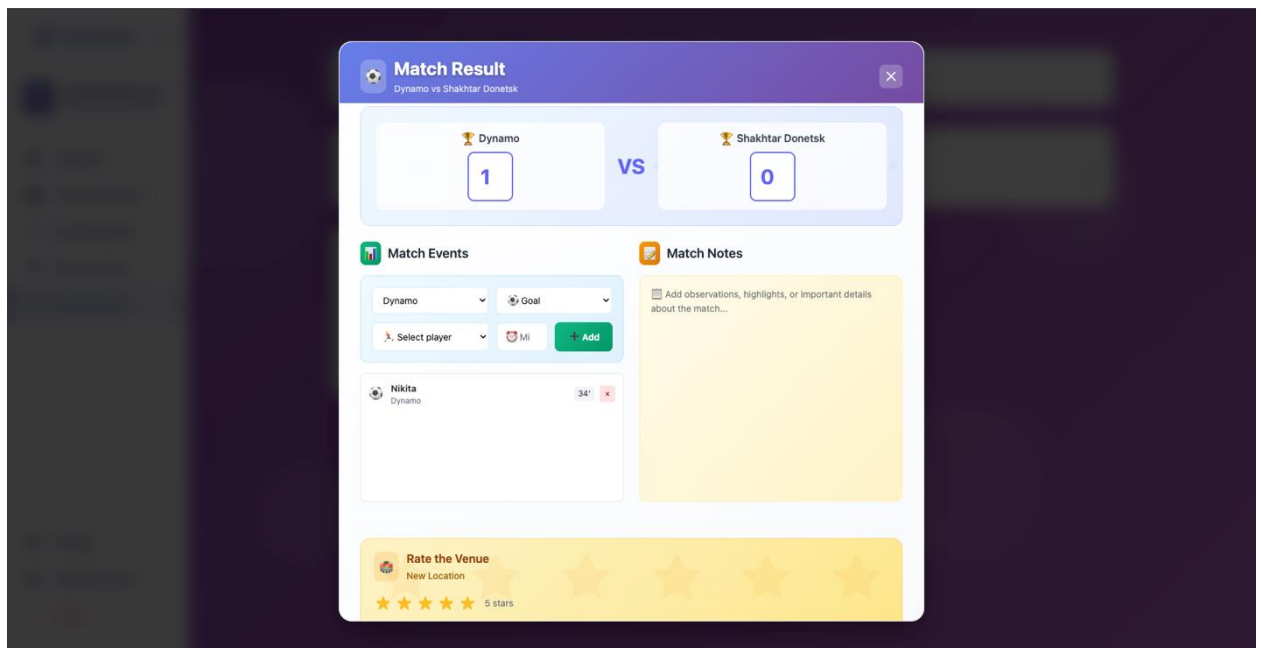


Рисунок 4.13 – Сторінка перегляду та планування матчів

Функціональність власника майданчиків представляє собою комплексну бізнес-платформу управління, що дозволяє власникам спортивних споруд оптимізувати використання об'єктів, керувати бронюваннями та аналізувати ефективність через розвинені dashboard-інтерфейси. Система інтегрує управління портфоліо об'єктів, обробку запитів на бронювання та бізнес-аналітику в єдиному робочому середовищі (рис. 4.14).

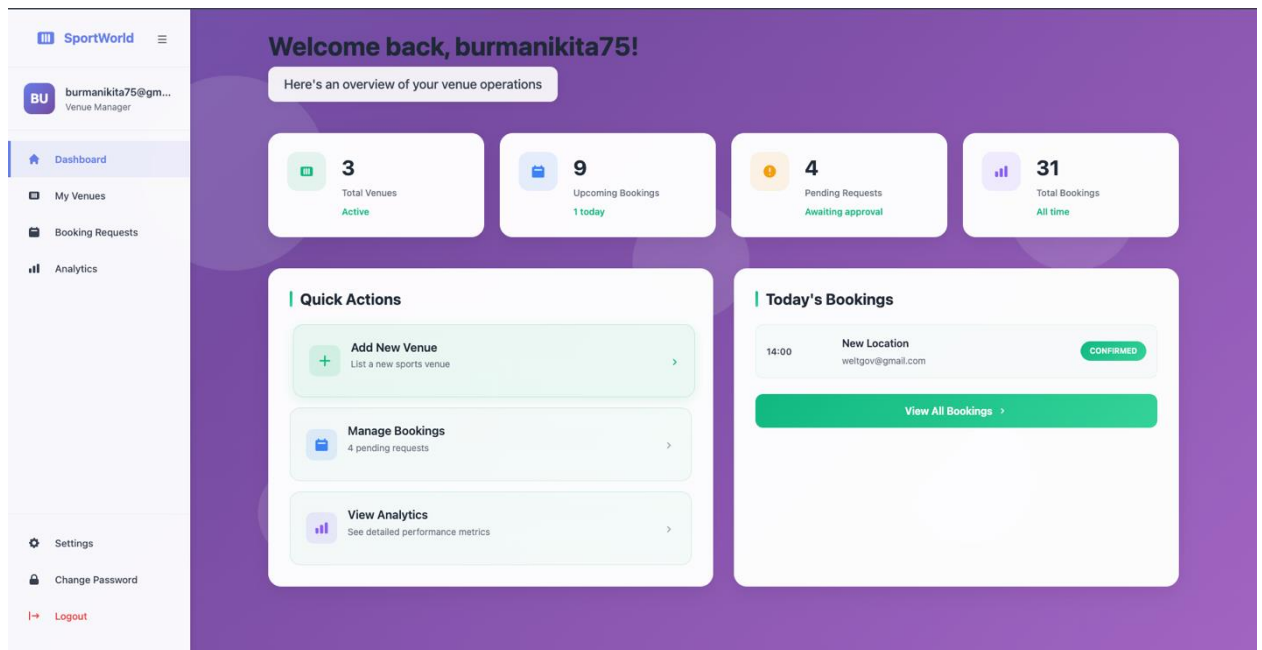


Рисунок 4.14 – Головна сторінка власника майданчиків

Портфоліо-менеджмент спортивних об'єктів забезпечує централізований хаб управління майданчиками, де власники можуть створювати, модифікувати та моніторити свої майданчики. Система підтримує комплексні профілі майданчиків, включаючи локаційні дані з точним координатним картуванням, типи споруд для різноманітних спортивних конфігурацій та динамічні моделі ціноутворення з погодинними тарифами. Можливості управління фотографіями забезпечують візуальну презентацію об'єктів через інтегровану функціональність завантаження зображень.

Кожен майданчик відображає показники ефективності: загальну кількість бронювань, коефіцієнти завантаженості (співвідношення підтверджених бронювань до загальних запитів), та рейтинг (рис. 4.15).

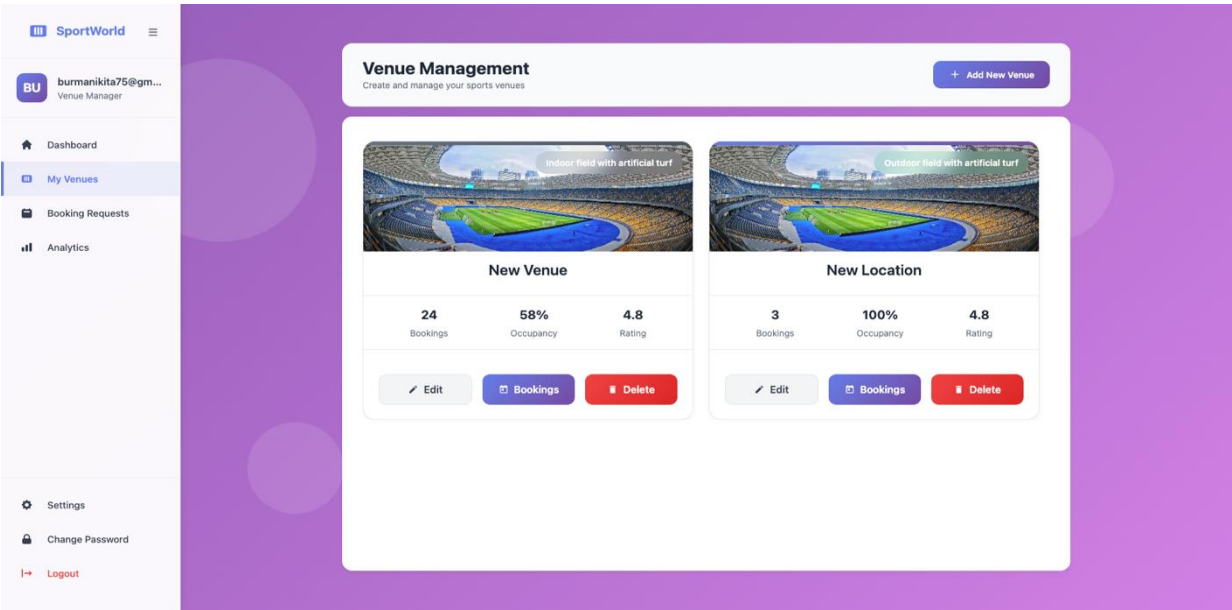


Рисунок 4.15 – Сторінка перегляду власних майданчиків

Багатостатусна система управління бронюваннями реалізує комплексну трирівневу систему, організовуючи запити за статусами: очікуючі (потребують схвалення власника), майбутні (підтверджені майбутні бронювання) та минулі (архівні записи, включаючи скасовані бронювання) (рис. 4.16). Така організація забезпечує ефективну обробку запитів та історичний аналіз використання.

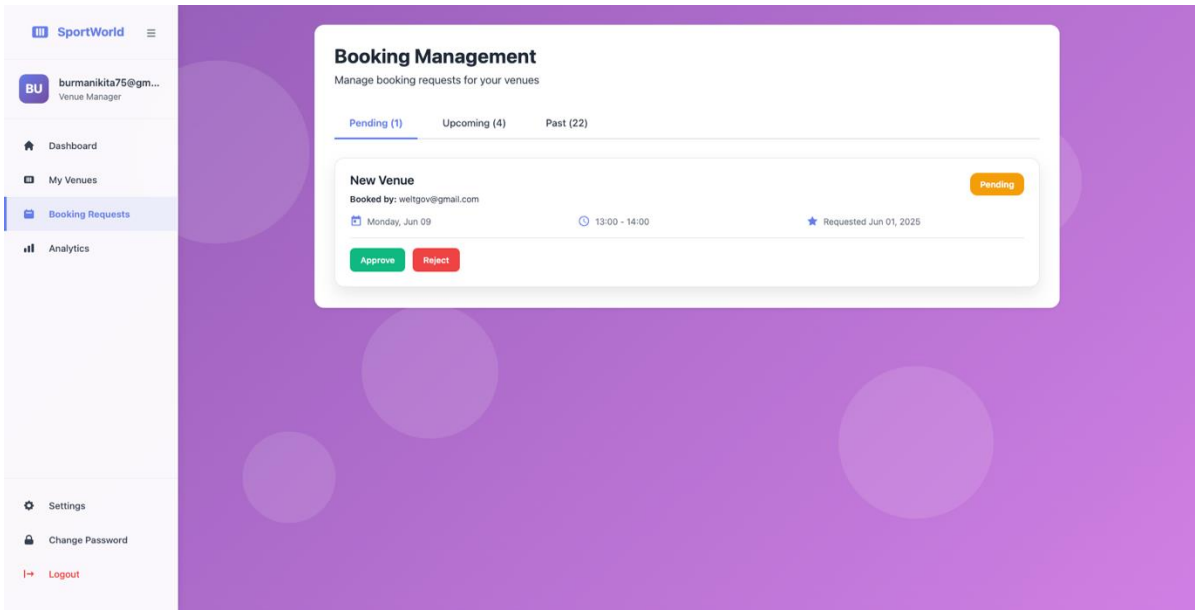


Рисунок 4.16 – Сторінка обробки запитів на бронювання майданчиків

Комплексна аналітична панель забезпечує детальне відстеження доходів, включаючи загальні надходження, середні вартості бронювань та тренди місячних доходів за шестимісячні періоди (рис. 4.17). Розрахунки доходів базуються виключно на підтверджених бронюваннях, забезпечуючи точність фінансової звітності.

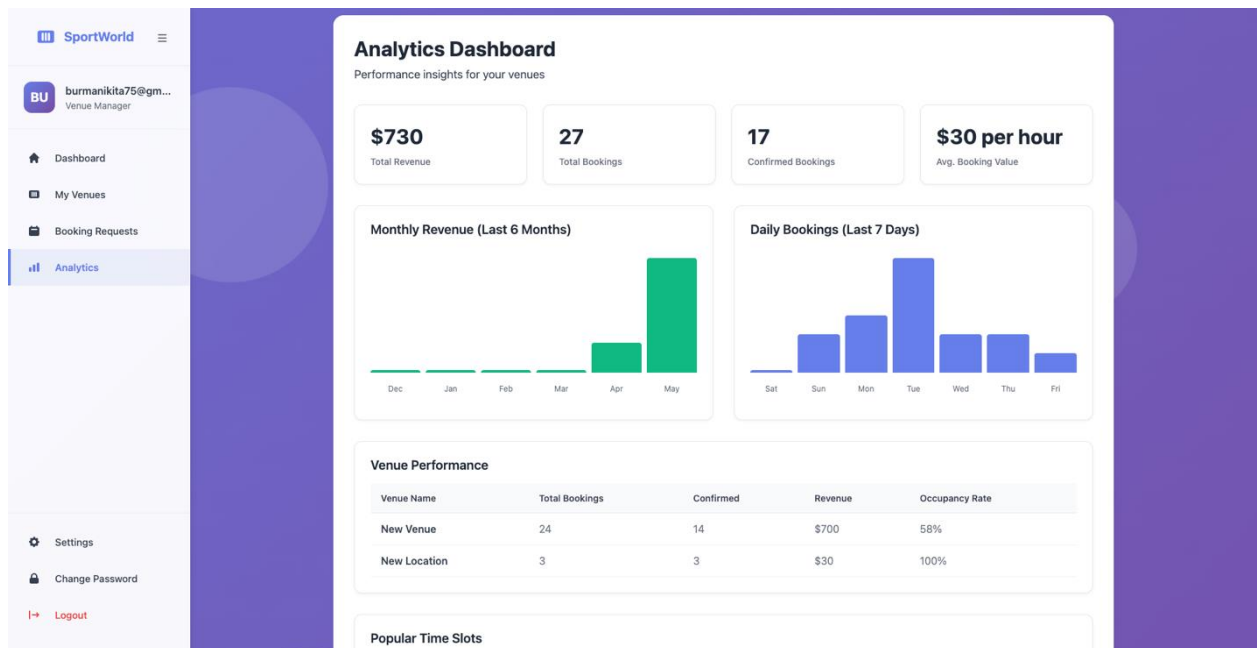


Рисунок 4.17 – Сторінка з аналітикою майданчиків

Операційна аналітика відстежує комплексні метрики бронювань, включаючи загальні запити, коефіцієнти підтвердження, обсяги очікуючих запитів та патерни скасування. Ці метрики дозволяють власникам оптимізувати процеси схвалення та ідентифікувати патерни використання для стратегічного планування розвитку бізнесу.

4.3 Тестування програми підтримки формування команд для занять футболом

Тестування функціональності програмного забезпечення забезпечує відповідність реалізованих функцій встановленим вимогам та специфікаціям. Для систематичної перевірки коректності роботи розробленої нами програми

підтримки формування команд для занять футболом було розроблено чек-лист тестування (табл. 4.1), який охоплює основні функціональні можливості системи з урахуванням різних ролей користувачів системи: неавторизований користувач, гравець, організатор турнірів та власник майданчиків.

Кожна функціональність системи оцінюється з точки зору доступності для конкретної ролі користувача, що дозволяє перевірити правильність реалізації системи контролю доступу та розмежування прав. Позначки «+» та «-» у таблиці свідчать про наявність або відсутність доступу до певної функції для кожного типу користувачів, що забезпечує наочність та зручність використання чек-листа в процесі тестування.

Таблиця 4.1 – Чек-лист тестування програми підтримки формування команд для занять футболом

Функція	Неавторизований користувач	Гравець	Організатор турнірів	Власник майданчиків
1	2	3	4	5
Перегляд головної сторінки	+	+	+	+
Реєстрація	+	+	+	+
Авторизація	+	+	+	+
Перегляд списку турнірів	-	+	+	-
Подача заявки на турнір	-	+	-	-
Створення турніру	-	-	+	-

Продовження таблиці 4.1

1	2	3	4	5
Управління турніром	-	-	+	-
Обробка заявок на турніри	-	-	+	-
Створення матчів	-	-	+	-
Внесення результатів матчів	-	-	+	-
Пошук гравців	-	+	+	-
Перегляд профілів гравців	-	+	-	-
Створення команди	-	+	-	-
Управління командою	-	+	-	-
Пошук команд	-	+	+	-
Перегляд списку майданчиків	-	+	+	+
Бронювання майданчиків	-	+	+	-
Створення майданчика	-	-	-	+
Редагування майданчика	-	-	-	+

Кінець таблиці 4.1

1	2	3	4	5
Управління бронюваннями	-	-	-	+
Перегляд аналітики доходів	-	-	-	+
Завантаження медіа-файлів	-	+	-	+
Редагування профілю	-	+	+	+
Перегляд власної статистики	-	+	+	+

4.4 Висновки за розділом 4

Результатом виконання четвертого розділу дипломної роботи стало успішне впровадження та комплексне тестування розробленої програми підтримки формування команд для занять футболом. Створена система демонструє високий рівень надійності, продуктивності та функціональної повноти, що підтверджується результатами багаторівневого тестування.

ВИСНОВКИ

У результаті виконання дипломної роботи була успішно розроблена програма підтримки формування команд для занять футболом, що об'єднує всіх учасників аматорського футболу в єдиній цифровій екосистемі. Поставлені завдання вирішено в повному обсязі, досягнуто мети створення програмного забезпечення підтримки формування команд для занять футболом.

Проведено глибокий аналіз предметної області, що дозволив виявити ключові проблеми організації аматорського футболу в Україні. Встановлено, що основними викликами є фрагментованість спортивної спільноти, відсутність централізованих механізмів взаємодії між учасниками та неефективне використання спортивної інфраструктури. Аналіз існуючих рішень показав відсутність комплексних платформ, що одночасно задовольняли б потреби гравців, організаторів турнірів та власників спортивних майданчиків. На основі виявлених проблем сформульовано концепцію інтегрованої програми підтримки спортивної діяльності, що передбачає створення єдиного цифрового простору для всіх учасників футбольної екосистеми.

Здійснено проектування архітектури системи на основі визначених функціональних вимог користувачів. Розроблено роле-орієнтовану модель доступу для трьох основних типів користувачів: гравців, організаторів турнірів та власників майданчиків. Проведено порівняльний аналіз технологічних рішень та обґрунтовано вибір оптимального стеку: NestJS для серверної частини, React для клієнтського інтерфейсу, PostgreSQL з PostGIS для зберігання даних.

Спроектовано структуру бази даних з 24 взаємопов'язаними таблицями, що забезпечують ефективне зберігання та обробку всіх необхідних даних. Архітектурні рішення орієнтовані на масштабованість, надійність та легкість підтримки системи.

Реалізовано модульну структуру backend-частини з сервісами автентифікації, управління користувачами, організації турнірів, бронювання майданчиків та ведення статистики. Frontend-частина побудована на компонентному підході з використанням сучасних патернів React. Особливу увагу приділено реалізації критично важливих функцій: системи запрошень до команд, механізму подачі заявок на турніри, календарного бронювання з підтвердженням власниками, розрахунку та візуалізації спортивної статистики. Розроблений код характеризується високою якістю, типобезпекою та відповідністю сучасним стандартам розробки.

Визначено умови експлуатації програми, що включають вимоги до апаратного забезпечення, операційного середовища та мережної інфраструктури. Детально описано логіку використання системи для кожного типу користувачів з посиланнями на відповідні елементи інтерфейсу. Проведено комплексне тестування функціональності програми за розробленим чек-листом, що охоплює всі ролі користувачів та їх можливості. Результати тестування підтвердили повну функціональну готовність системи до експлуатації в реальних умовах.

Виконана дипломна робота повністю відповідає поставленим завданням. Розроблена програма підтримки формування команд для занять футболом являє собою завершений програмний продукт, готовий до впровадження та комерційного використання в сфері організації аматорського спорту України.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Скічко, О. Масовий футбол в Україні [Електрон. ресурс] : цифри і проблеми / О. Скічко. – Режим доступу : <https://1927.kiev.ua/news/view/63916>.
2. UEFA GROW в Україні [Електрон. ресурс] : результати дослідження цінності масового футболу та соціальна місія. – Режим доступу : <https://www.oofa.com.ua/2021/04/28/uefa-grow-v-ukrayini-rezultaty-doslidzhennya-tsinnosti-masovogo-futbolu-ta-sotsialna-misiya>.
3. Асоціація аматорського футболу України [Електрон. ресурс] : чемпіонат України з футболу серед аматорських команд. – Режим доступу : <http://www.aafu.org.ua/championship>.
4. Офіційний сайт ГС «Запорізька обласна асоціація футболу» [Електронний ресурс] : інформація про Запорізьку обласну асоціацію футболу. – Режим доступу: <https://www.zoff.zp.ua>.
5. Рахункова палата України [Електронний ресурс] : звіт про результати аудиту ефективності використання коштів державного бюджету на розвиток спортивної інфраструктури. – Режим доступу : https://rp.gov.ua/upload-files/Activity/Collegium/2019/3-2_2019/Zvit_32_2019.pdf.
6. TyKyiv.com [Електронний ресурс] : гра мільйонів : де у Києві дитині пограти у футбол. – Режим доступу : <https://tykyiv.com/city/gra-milioniv-de-ukiiievi-ditini-pograti-v-futbol>.
7. Meetup. Find Local Groups, Events, and Activities Near You [Electronic resource]. – Access mode : <https://www.meetup.com>.
8. Playtomic : Join the community and book courts online in one [Electronic resource]. – Access mode : <https://playtomic.com>.
9. Heja | Sports team management and communication app [Electronic resource]. – Access mode : <https://heja.io>.
10. NestJS. (n.d.). NestJS Documentation [Electronic resource]. – Access mode : <https://docs.nestjs.com>.
11. Express.js. (n.d.). Express - Node.js web application framework

[Electronic resource]. – Access mode : <https://expressjs.com>.

12. Fastify. (n.d.). Fastify Documentation [Electronic resource]. – Access mode : <https://fastify.dev/docs/v5.2.x>.

13. The PostgreSQL Global Development Group. (n.d.). PostgreSQL 16.9 Documentation [Electronic resource]. – Access mode : <https://www.postgresql.org/docs>.

14. Oracle Corporation. (n.d.). MySQL 8.4 Reference Manual [Electronic resource]. – Access mode : <https://dev.mysql.com/doc>.

15. MariaDB Foundation. (n.d.). MariaDB Server Documentation [Electronic resource]. – Access mode : <https://mariadb.org/documentation>.

16. Meta Platforms, Inc. (n.d.). React – A JavaScript library for building user interfaces [Electronic resource]. – Access mode : <https://react.dev>.

17. Google LLC. (n.d.). Angular – Web framework for building applications [Electronic resource]. – Access mode : <https://angular.dev>.

18. Vue.js. (n.d.). Vue.js – The Progressive JavaScript Framework [Electronic resource]. – Access mode : <https://vuejs.org>.

ДОДАТОК А
Технічне завдання

Вступ

Сучасний аматорський футбол характеризується високою популярністю серед різних вікових груп та соціальних верств, проте організація футбольної діяльності часто ускладнюється відсутністю централізованої системи управління командами, турнірами та спортивними об'єктами. Гравці стикаються з проблемами пошуку команд, організатори турнірів мають складнощі з координацією заявок та складанням турнірних сіток, а власники спортивних майданчиків не мають ефективних інструментів для управління бронюванням та аналітикою використання своїх об'єктів.

Фундаментальною проблемою, що перешкоджає ефективному функціонуванню аматорської футбольної спільноти, є фрагментарність системи взаємодії між її учасниками та відсутність комплексних технологічних рішень для її координації. Це проявляється у вигляді ускладненого командування, неефективної організації турнірів, проблем з бронюванням спортивних об'єктів та відсутності механізмів об'єктивного оцінювання рівня гравців і якості спортивної інфраструктури. Зазначені дисфункції формують взаємопов'язаний комплекс перешкод, що системно стримують розвиток потенціалу аматорського футболу в Україні та обумовлюють необхідність розробки спеціалізованих програмних рішень для їх подолання.

A.1 Підстави для розробки

Основна підстава для виконання роботи сформульована у відповідності з Наказом № 209 від 28 квітня 2025 року, виданим у Національному університеті «Запорізька політехніка», де для виконання даної дипломної кваліфікаційної роботи бакалавра визначено «Програмне забезпечення підтримки формування команд для занять футболом» як тему роботи.

A.2 Призначення розробки

Програмне забезпечення підтримки формування команд для занять футболом розробляється для того, щоб надати усім учасникам аматорського футболу можливість знаходження партнерів для спортивної діяльності, здатних задовольнити їх потреби зокрема за показниками рівня гри, досвіду та географічного розташування, а гравцям безпосередньо знайти ті команди та турніри, які здатні якнайкраще використовувати їхні спортивні навички та амбіції.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Вимоги до програми підтримки закупівлі вовни у виробників за набором функцій складають:

- реєстрація користувачів з вибором ролі;
- авторизація;
- верифікація email адреси;
- відновлення пароля через email;
- створення профілю гравця з персональними даними;
- оновлення профілю гравця;
- завантаження медіа-контенту;
- перегляд статистики гравця;
- перегляд історії участі в турнірах;
- створення нової команди з назвою та описом;
- завантаження логотипу команди;
- автоматичне призначення капітана команди;
- пошук та перегляд доступних команд;
- відправка заявки на вступ до команди;
- прийняття або відхилення заявок на вступ (тільки капітан);

- відправка запрошень гравцям до команди (тільки капітан);
- прийняття або відхилення запрошень від команд;
- залишення команди гравцем;
- розпуск команди капітаном;
- створення турніру;
- оновлення параметрів турніру;
- видалення турніру організатором;
- перегляд всіх доступних турнірів;
- фільтрація турнірів за статусом та форматом;
- подача заявки команди на участь у турнірі;
- перегляд статусу заявки на участь;
- скасування заявки на участь;
- розгляд заявок команд організатором турніру;
- підтвердження або відхилення заявок на участь;
- перегляд статистики турніру;
- створення матчу в рамках турніру;
- призначення команд-учасниць матчу;
- вибір майданчика для проведення матчу;
- внесення результатів матчу;
- оновлення результатів матчу;
- перегляд результатів проведених матчів;
- ведення статистики гравців у матчі;
- оцінка якості майданчика після матчу;
- створення спортивного майданчика;
- завантаження фотографій майданчика;
- оновлення інформації про майданчик;
- видалення майданчика власником;
- перегляд аналітики використання майданчика;
- перегляд всіх доступних майданчиків;
- пошук майданчиків за локацією та доступністю;

- перегляд календаря доступності майданчика;
- вибір дати та часу для бронювання;
- перегляд доступних часових слотів;
- створення запиту на бронювання;
- підтвердження або відхилення бронювання власником;
- скасування бронювання користувачем;
- перегляд історії бронювань користувача;
- аналіз щоденної та місячної статистики бронювань;
- визначення популярних часових слотів;
- перегляд медіа-галереї;
- видалення медіа-контенту;
- підтримка GPS координат для майданчиків.

A.3.2 Умови експлуатації

Для експлуатації програми особливих вимог не висувається, програма функціонує в архітектурі клієнт-сервер з вебінтерфейсом, що забезпечує доступ через стандартні браузері без необхідності встановлення спеціалізованого програмного забезпечення.

A.3.3 Вимоги до складу та параметрів технічних засобів

Для забезпечення стабільної роботи серверної частини системи мінімальні технічні вимоги включають: 8 Гб оперативної пам'яті (рекомендовано 16 Гб) для ефективної обробки запитів від численних користувачів одночасно, чотириядерний процесор та дисковий простір обсягом 40 Гб для зберігання бази даних та медіафайлів. Клієнтська частина потребує лише сучасного веббраузера з підтримкою TypeScript та стабільного інтернет-з'єднання.

A.4 Порядок контролю та приймання

Сумарно 8 тижнів виділено на виконання дипломної кваліфікаційної роботи бакалавра, розподіл між якими має виконуватися за календарним планом, визначеним у завданні на роботі, з забезпеченням результатів, які мають бути досягнуті та які зазначені в календарному плані.

У підсумку робота оцінюється керівником зі створенням відгуку, рецензентом зі створенням рецензії та екзаменаційною комісією за наперед затвердженим графіком.

ДОДАТОК Б
Текст програми

```

@Controller('auth')
export class AuthController {
  constructor(
    private authService: AuthService,
    private mailService: MailService
  ) {}

  @Post('login')
  async login(@Body() loginDto: LoginDto, @Response()
res: ExpressResponse) {
    return await this.authService.login(loginDto.email,
loginDto.password, res);
  }

  @Post('register')
  async register(@Body() registerDto: RegisterDto,
@Response() res: ExpressResponse) {
    return this.authService.register(
      registerDto.email,
      registerDto.password,
      registerDto.user_type,
      res
    );
  }

  @Get('confirm-email')
  @Redirect()
  async confirmEmail(@Query() confirmEmailDto:
ConfirmEmailDto) {
    const token = confirmEmailDto.token;
    const user = await
this.authService.confirmUserEmail(token);

    if (user) {
      return {
        url: 'http://localhost:3000/confirm-
email?message=Email%20confirmed%20successfully&error=false',
        statusCode: 302
      };
    } else {
      return {
        url: 'http://localhost:3000/confirm-
email?message=Invalid%20or%20expired%20token&error=true',
        statusCode: 302
      };
    }
  }
}

```

```

        };
    }
}

    @Post('reset-password')
    async resetPassword(@Body() resetPasswordDto:
ResetPasswordDto) {
        const result = await
this.authService.resetPassword(resetPasswordDto.email);
        return result;
    }

    @Post('confirm-reset-password')
    async confirmResetPassword(@Body() confirmResetDto:
ConfirmResetPasswordDto) {
        const result = await
this.authService.confirmResetPassword(
            confirmResetDto.token,
            confirmResetDto.newPassword
        );
        return result;
    }

    @Post('logout')
    async logout(@Body() body: { email: string },
    @Response() res: ExpressResponse) {
        const result = await
this.authService.logout(body.email);

        res.clearCookie('access_token');
        res.clearCookie('refresh_token');

        return res.json(result);
    }
}

@Controller("tournaments")
export class TournamentController {
    constructor(private tournamentService:
TournamentService) {}

    @Post()
    async createTournament(@Body() tournamentData:
CreateTournamentDto) {

```

```

        const tournament = await
this.tournamentService.createTournament(
        tournamentData,
        tournamentData.organizerEmail
    );

    return {
        success: true,
        tournament,
        message: "Tournament created successfully",
    };
}

@Get()
async getAllTournaments() {
    const tournaments = await
this.tournamentService.getAllTournaments();
    return {
        success: true,
        tournaments,
    };
}

@Get(":id")
async getTournamentById(@Param("id") id: string) {
    const tournamentId = parseInt(id);
    const tournament = await
this.tournamentService.getTournamentById(tournamentId);

    return {
        success: true,
        tournament,
    };
}

@Put(":id")
async updateTournament(@Param("id") id: string, @Body()
updateData: UpdateTournamentDto) {
    const tournamentId = parseInt(id);
    const tournament = await
this.tournamentService.updateTournament(tournamentId,
updateData);

    return {

```

```

        success: true,
        tournament,
        message: "Tournament updated successfully",
    };
}

@Delete(":id")
async deleteTournament(@Param("id") id: string) {
    const tournamentId = parseInt(id);
    await
this.tournamentService.deleteTournament(tournamentId);

    return {
        success: true,
        message: "Tournament deleted successfully",
    };
}

}

@Controller('teams')
export class TeamController {
    constructor(private teamService: TeamService) {}

    @Post()
    @UseGuards(JwtAuthGuard)
    async createTeam(@Body() createTeamDto: CreateTeamDto,
@Request() req: any) {
        const userEmail = req.user.email;
        const team = await
this.teamService.createTeam(createTeamDto, userEmail);

        return {
            success: true,
            team,
            message: 'Team created successfully'
        };
    }

    @Get()
    async getAllTeams() {
        const teams = await this.teamService.getAllTeams();

        return {
            success: true,

```

```

        teams
    };
}

@Post('/:id/join-request')
@UseGuards(JwtAuthGuard)
async sendJoinRequest(@Param('id') teamId: string,
@Request() req: any) {
    const userEmail = req.user.email;
    const result = await
this.teamService.sendJoinRequest(parseInt(teamId), userEmail);

    return result;
}

@Post('/:id/invite')
@UseGuards(JwtAuthGuard)
async invitePlayer(@Param('id') teamId: string, @Body()
inviteDto: InvitePlayerDto, @Request() req: any) {
    const inviterEmail = req.user.email;
    const result = await this.teamService.invitePlayer(
        parseInt(teamId),
        inviteDto.inviteeEmail,
        inviterEmail
    );

    return result;
}

@Put('join-requests/:requestId')
@UseGuards(JwtAuthGuard)
async handleJoinRequest(
    @Param('requestId') requestId: string,
    @Body() handleDto: HandleJoinRequestDto,
    @Request() req: any
) {
    const captainEmail = req.user.email;
    const result = await
this.teamService.handleJoinRequest(
        parseInt(requestId),
        handleDto.action,
        captainEmail
    );
};

```

```

        return result;
    }
}

@Injectable()
export class AuthService {
    constructor(
        private usersService: UsersService,
        private jwtService: JwtService,
        @InjectRepository(User) private userRepository:
Repository<User>,
        @InjectRepository(UserCredentials) private
userCredentialsRepository: Repository<UserCredentials>,
        @InjectRepository(UserTypes) private
userTypesRepository: Repository<UserTypes>,
        private readonly mailService: MailService,
    ) {}

    async register(email: string, password: string,
userType: string, res: ExpressResponse) {
        // Перевірка чи існує користувач
        const userExist = await
this.userCredentialsRepository.findOne({ where: { email } });

        if (userExist) {
            return res.status(401).json({ message: 'This
email is already in use.' });
        }

        // Хешування пароля
        const hashedPassword = await argon2.hash(password);
        const confirmToken = uuidv4();

        // Створення облікових даних
        const user_credentials =
this.userCredentialsRepository.create({
            email,
            password: hashedPassword,
            confirmToken,
            confirmEmail: false,
        });

        await
this.userCredentialsRepository.save(user_credentials);
    }
}

```

```

        // Створення користувача з типом
        const userTypeEntity = await
this.userTypesRepository.findOne({
            where: { type_name: userType }
        });

        if (!userTypeEntity) {
            return res.status(400).json({ message: 'Invalid
user type' });
        }

        const user = this.userRepository.create({
            credentials: user_credentials,
            type: userTypeEntity
        });

        await this.userRepository.save(user);

        // Відправка email для підтвердження
        const confirmLink =
`http://localhost:3001/auth/confirm-
email?token=${confirmToken}`;
        await this.mailService.sendEmailVerification(email,
confirmLink);

        return res.status(200).json({
            message: 'User registered successfully. Please
check your email to confirm your account.'
        });
    }

    async login(email: string, password: string, res:
ExpressResponse) {
        // Пошук користувача
        const userCredentials = await
this.userCredentialsRepository.findOne({
            where: { email },
            select: ['id', 'email', 'password',
'confirmEmail']
        });

        if (!userCredentials) {

```

```

        return res.status(404).json({ message: 'User
not found' });
    }

    if (!userCredentials.confirmEmail) {
        return res.status(403).json({ message: 'Please
confirm your email before logging in' });
    }

    // Перевірка пароля
    const isPasswordValid = await
argon2.verify(userCredentials.password, password);
    if (!isPasswordValid) {
        return res.status(401).json({ message: 'Invalid
password' });
    }

    // Генерація токенів
    const payload = { username: userCredentials.email,
sub: userCredentials.id };
    const token = await this.jwtService.sign(payload, {
expiresIn: '1h' });
    const refreshToken = await
this.jwtService.sign(payload, { expiresIn: '7d' });

    // Збереження refresh token
    userCredentials.refreshToken = refreshToken;
    await
this.userCredentialsRepository.save(userCredentials);

    // Встановлення cookies
    res.cookie('access_token', token, {
        httpOnly: false,
        sameSite: 'lax',
        secure: false,
        maxAge: 60 * 60 * 1000 // 1 година
    });

    res.cookie('refresh_token', refreshToken, {
        httpOnly: true,
        sameSite: 'lax',
        secure: false,
        maxAge: 7 * 24 * 60 * 60 * 1000 // 7 днів
    });

```

```

        return res.json({
            message: 'Login successful',
            access_token: token
        });
    }

    async confirmUserEmail(token: string) {
        const userCredentials = await
this.userCredentialsRepository.findOne({
            where: { confirmToken: token }
        });

        if (!userCredentials) {
            return false;
        }

        userCredentials.confirmEmail = true;
        userCredentials.confirmToken = null;
        await
this.userCredentialsRepository.save(userCredentials);

        return true;
    }
}

@Injectable()
export class GroundService {
    constructor(
        @InjectRepository(GroundCredentials)
        private groundCredentialsRepository:
Repository<GroundCredentials>,
        @InjectRepository(GroundOwner)
        private groundOwnerRepository:
Repository<GroundOwner>,
        @InjectRepository(VenueBooking)
        private venueBookingRepository:
Repository<VenueBooking>,
        @InjectRepository(User)
        private userRepository: Repository<User>,
    ) {}

    async createGround(email: string, title: string,
position: any, type: string, price_per_hour: number) {

```

```

        // Створення майданчика
        const newGround =
this.groundCredentialsRepository.create({
            title: title,
            position: { latitude:
position.location.latitude, longitude:
position.location.longitude },
            price_per_hour: price_per_hour,
        });

        await
this.groundCredentialsRepository.save(newGround);

        // Прив'язка до власника
        const user = await this.findUserByEmail(email);
        const newUserGround =
this.groundOwnerRepository.create({
            ownerId: user,
            groundId: newGround
        });

        await
this.groundOwnerRepository.save(newUserGround);

        return {
            success: true,
            message: 'Successfully created!',
            ground: { id: newGround.id }
        };
    }

    async getAvailableSlots(groundId: number, date: string)
    {
        // Перевірка існування майданчика
        const venue = await
this.groundCredentialsRepository.findOne({
            where: { id: groundId }
        });

        if (!venue) {
            return { success: false, message: 'Venue not
found' };
        }
    }

```

```

        // Отримання існуючих бронювань
        const existingBookings = await
this.venueBookingRepository.find({
            where: {
                venue: { id: groundId },
                booking_date: date as any,
                status: In(['confirmed', 'pending'])
            },
            order: { start_time: 'ASC' }
        });

        // Генерація годинних слотів з 8:00 до 22:00
        const slots = [];
        for (let hour = 8; hour < 22; hour++) {
            const startHour = hour.toString().padStart(2,
'0');

            const endHour = (hour +
1).toString().padStart(2, '0');
            const startTime = `${startHour}:00:00`;

            const isBooked = existingBookings.some(booking
=>
                booking.start_time === startTime
            );

            slots.push({
                id: hour,
                start_time: `${startHour}:00`,
                end_time: `${endHour}:00`,
                isBooked
            });
        }

        return { success: true, slots };
    }

    async bookSlot(bookingData: any) {
        const { groundId, date, start_time, end_time,
userEmail } = bookingData;

        // Перевірка доступності слоту
        const existingBooking = await
this.venueBookingRepository.findOne({
            where: {

```

```

        venue: { id: groundId },
        booking_date: date as any,
        start_time: new
Date(start_time).toTimeString().slice(0, 8),
        status: In(['confirmed', 'pending'])
    }
});

    if (existingBooking) {
        return { success: false, message: 'This time
slot is already booked' };
    }

    // Створення бронювання
    const venue = await
this.groundCredentialsRepository.findOne({
        where: { id: groundId }
    });

    const venueBooking =
this.venueBookingRepository.create({
        venue: venue,
        user_email: userEmail,
        booking_date: date,
        start_time: new
Date(start_time).toTimeString().slice(0, 8),
        end_time: new
Date(end_time).toTimeString().slice(0, 8),
        status: 'pending'
    });

    await
this.venueBookingRepository.save(venueBooking);

    return { success: true, message: 'Booking
confirmed' };
}

const App: React.FC = () => {
    return (
        <AuthProvider>
        <BrowserRouter>
        <div className="App">

```

```

<Routes>
  <Route path="/" element={<Main />} />
  <Route path="/login" element={<Login />} />
  <Route path="/register" element={<Register />}
/>

  <Route path="/confirm-email"
element={<ConfirmEmail />} />
  <Route path="/change-password"
element={<ChangePassword />} />
  <Route path="/confirm-change-password"
element={<ConfirmChangePassword />} />

  {/* Player Dashboard Routes */}
  <Route path="/home" element={<PlayerDashboard
/>}>

    <Route index element={<PlayerHome />} />
    <Route path="profile" element={<GamerProfile
/>} />

    <Route path="teams" element={<Teams />} />
    <Route path="tournaments"
element={<Tournaments />} />
    <Route path="tournaments/:id"
element={<TournamentDetails />} />
    <Route path="venues" element={<FindVenues />}
/>

    <Route path="venues/booking"
element={<VenueBooking />} />
    <Route path="matches" element={<MyMatches />}
/>

    <Route path="bookings" element={<MyBookings
/>} />

    <Route path="find-players"
element={<FindPlayers />} />
    <Route path="media" element={<MediaUpload />}
/>

    <Route path="statistics" element={<Statistics
/>} />

    <Route path="my-tournament-applications"
element={<MyTournamentApplications />} />
    <Route path="settings" element={<Settings />}
/>

  </Route>

  {/* Organizer Dashboard Routes */}

```

```

        <Route path="/organizer"
element={<OrganizerDashboard />}>
            <Route index element={<OrganizerHome />} />
            <Route path="create-tournament"
element={<CreateTournament />} />
            <Route path="tournaments"
element={<MyTournaments />} />
            <Route path="tournaments/:id/manage"
element={<ManageTournament />} />
            <Route path="match-calendar"
element={<MatchCalendar />} />
            <Route path="results" element={<Results />}
/>

    </Route>

    {/* Venue Owner Dashboard Routes */}
    <Route path="/venue-owner"
element={<VenueOwnerDashboard />}>
        <Route index element={<VenueOwnerHome />} />
        <Route path="analytics" element={<Analytics
/>} />
        <Route path="bookings"
element={<BookingManagement />} />
        <Route path="settings" element={<Settings />}
/>

    </Route>

    {/* Ground Controller Routes */}
    <Route path="/ground-controller"
element={<GroundController />} />
        <Route path="/ground-controller/booking"
element={<BookingController />} />
    </Routes>
</div>
</BrowserRouter>
</AuthProvider>
);
};

export default App;

interface AuthContextType {
    user: User | null;
    email: string | null;

```

```

    userType: string | null;
    gamerProfile: GamerProfile | null;
    loading: boolean;
    logout: () => void;
    setEmailInContext: (email: string) => void;
    updateUserFromToken: (token: string) => void;
  }

  export const AuthProvider: React.FC<AuthProviderProps> = ({
    children }) => {
    const [user, setUser] = useState<User | null>(null);
    const [email, setEmail] = useState<string | null>(() =>
      localStorage.getItem("email"));
    const [userType, setUserType] = useState<string |
      null>(null);
    const [gamerProfile, setGamerProfile] =
      useState<GamerProfile | null>(null);
    const [loading, setLoading] = useState(true);

    const updateUserFromToken = useCallback(async (token:
      string) => {
      try {
        const decoded = jwtDecode<User>(token);
        setUser({ ...decoded, token });

        if (decoded.username) {
          setEmail(decoded.username);
          localStorage.setItem("email",
            decoded.username);
          await loadUserType(decoded.username);
          await loadGamerProfile(decoded.username);
        }
      } catch (error) {
        console.error("Ошибка декодирования токена:",
          error);
      }
    }, []);

    const loadUserType = async (userEmail: string) => {
      try {
        const response = await api.get('/users/type', {
          params: { email: userEmail }
        });
        if (response.data.success) {

```

```

        setUserType(response.data.userType);
    }
    } catch (error) {
        console.error("Помилка завантаження типу
користувача:", error);
    }
};

const loadGamerProfile = async (userEmail: string) => {
    try {
        const response = await api.get('/users/gamer-
profile', {
            params: { email: userEmail }
        });
        if (response.data.success) {
            setGamerProfile(response.data.gamerProfile);
        }
    } catch (error) {
        console.error("Помилка завантаження типу
гравця:", error);
    }
};

const logout = async () => {
    try {
        await api.post('/auth/logout', { email: email
|| '' });
    } catch (error) {
        console.error('Logout error:', error);
    } finally {
        setUser(null);
        setEmail(null);
        setUserType(null);
        setGamerProfile(null);

        Cookies.remove("access_token");
        Cookies.remove("refresh_token");
        localStorage.removeItem("email");

        window.location.href = '/login';
    }
};

```

```

const setEmailInContext = (email: string) => {
  setEmail(email);
  localStorage.setItem("email", email);
};

useEffect(() => {
  const initializeAuth = async () => {
    const token = Cookies.get('access_token');

    if (token) {
      await updateUserFromToken(token);
    } else if (email) {
      await loadUserType(email);
      await loadGamerProfile(email);
    }

    setLoading(false);
  };

  initializeAuth();
}, [email, updateUserFromToken]);

return (
  <AuthContext.Provider value={{
    user,
    email,
    userType,
    gamerProfile,
    loading,
    logout,
    setEmailInContext,
    updateUserFromToken
  }}>
    {children}
  </AuthContext.Provider>
);
};

import axios, { AxiosError } from 'axios';
import { InternalAxiosRequestConfig } from 'axios';
import Cookies from 'js-cookie';

const api = axios.create({
  baseURL: 'http://localhost:3001',

```

```

    withCredentials: true,
  });

  // Request interceptor для додавання токена автентифікації
  api.interceptors.request.use(
    (config: InternalAxiosRequestConfig) => {
      const token = Cookies.get('access_token');
      if (token && config.headers) {
        config.headers.Authorization = `Bearer ${token}`;
      }
      return config;
    },
    (error) => {
      return Promise.reject(error);
    }
  );

  // Response interceptor для автоматичного оновлення токена
  api.interceptors.response.use(
    (response) => response,
    async (error: AxiosError) => {
      const originalRequest = error.config as
InternalAxiosRequestConfig & { _retry?: boolean };

      if (error.response?.status === 401 &&
!originalRequest._retry) {
        originalRequest._retry = true;

        try {
          const refreshToken = Cookies.get('refresh_token');

          if (!refreshToken) {
            throw new Error('No refresh token available');
          }

          const response = await axios.post(
            'http://localhost:3001/auth/refresh',
            { refreshToken },
            { withCredentials: true }
          );

          const { access_token } = response.data;
          Cookies.set('access_token', access_token, {
            expires: 1/24, // 1 година

```

```

        sameSite: 'lax'
    });

    if (originalRequest.headers) {
        originalRequest.headers.Authorization = `Bearer
${access_token}`;
    }

    return api(originalRequest);
} catch (refreshError) {
    console.error('Token refresh failed:',
refreshError);

    // Очищення всіх токенів та перенаправлення на
сторінку входу
    Cookies.remove('access_token');
    Cookies.remove('refresh_token');
    localStorage.removeItem('email');

    window.location.href = '/login';
    return Promise.reject(refreshError);
}
}

return Promise.reject(error);
}
);

export default api;

const GamerProfile: React.FC = () => {
    const { email, gamerProfile, updateUserFromToken } =
useContext(AuthContext);
    const [isEditMode, setIsEditMode] = useState(false);
    const [profileData, setProfileData] =
useState<GamerProfileFormData>({
        name: '',
        age: '',
        sex: '',
        location: '',
        skill_lvl: '',
        experience: '',
        position: ''
    });
};

```

```

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();

  try {
    const response = await api.put('/users/gamer-
profile', {
      ...profileData,
      email: email
    });

    if (response.data.success) {
      showToast('success', 'Profile updated
successfully!');

      setIsEditMode(false);

      // Оновлення токена для отримання нових
даних

      const token = Cookies.get('access_token');
      if (token) {
        updateUserFromToken(token);
      }
    }
  } catch (error: any) {
    console.error('Error updating profile:',
error);

    showToast('error',
error.response?.data?.message || 'Failed to update profile');
  }
};

const handleMediaUpload = async (event:
React.ChangeEvent<HTMLInputElement>) => {
  const file = event.target.files?.[0];
  if (!file) return;

  const formData = new FormData();
  formData.append('media', file);
  formData.append('email', email || '');

  try {
    const response = await
api.post('/media/upload', formData, {

```

```

        headers: { 'Content-Type': 'multipart/form-
data' }

    });

    if (response.data.success) {
        showToast('success', 'Media uploaded
successfully!');

        fetchMedia(); // Оновлення списку медіа
    }
    } catch (error: any) {
        console.error('Error uploading media:', error);
        showToast('error', 'Failed to upload media');
    }
};

return (
    <Container>
        <ProfileCard>
            <ProfileHeader>
                <Avatar src={gamerProfile?.avatar ||
'/default-avatar.png'} alt="Profile" />
                <UserInfo>
                    <UserName>{gamerProfile?.name ||
'Player Name'}</UserName>
                    <UserRole>Football
Player</UserRole>
                    <SkillLevel>Skill Level:
{gamerProfile?.skill_lvl || 'N/A'}/10</SkillLevel>
                </UserInfo>
                <ActionButton onClick={() =>
setIsEditMode(!isEditMode)}>
                    {isEditMode ? 'Cancel' : 'Edit
Profile'}
                </ActionButton>
            </ProfileHeader>

            {isEditMode ? (
                <EditForm onSubmit={handleSubmit}>
                    <FormGrid>
                        <FormGroup>
                            <Label>Name</Label>
                            <Input
                                type="text"

```

```

value={profileData.name}
                                onChange={ (e) =>
setProfileData(prev => ({...prev, name: e.target.value}))}
                                placeholder="Enter your
name"

                                />
                                </FormGroup>

                                <FormGroup>
                                <Label>Position</Label>
                                <Select

value={profileData.position}
                                onChange={ (e) =>
setProfileData(prev => ({...prev, position: e.target.value}))}
                                >
                                <option value="">Select
Position</option>
                                <option
value="Goalkeeper">Goalkeeper</option>
                                <option value="Center
Back">Center Back</option>
                                <option value="Left
Back">Left Back</option>
                                <option value="Right
Back">Right Back</option>
                                <option
value="Defensive Midfielder">Defensive Midfielder</option>
                                <option value="Central
Midfielder">Central Midfielder</option>
                                <option
value="Attacking Midfielder">Attacking Midfielder</option>
                                <option value="Left
Winger">Left Winger</option>
                                <option value="Right
Winger">Right Winger</option>
                                <option
value="Striker">Striker</option>
                                </Select>
                                </FormGroup>

                                <FormGroup>

```

```

10)</Label>
                                <Label>Skill Level (1-
                                <Input
                                    type="number"
                                    min="1"
                                    max="10"
                                value={profileData.skill_lvl}
                                onChange={ (e) =>
setProfileData(prev => ({...prev, skill_lvl: e.target.value}))}
                                />
                                </FormGroup>
                                </FormGrid>

                                <FormGroup>
                                    <Label>Experience</Label>
                                    <TextArea
value={profileData.experience}
                                onChange={ (e) =>
setProfileData(prev => ({...prev, experience: e.target.value}))}
                                placeholder="Describe your
football experience..."
                                rows={4}
                                />
                                </FormGroup>

                                <SaveButton type="submit">Save
Changes</SaveButton>
                                </EditForm>
                                ) : (
                                    <ProfileDetails>
                                        <DetailItem>

<DetailLabel>Position:</DetailLabel>

<DetailValue>{gamerProfile?.position || 'Not
specified'}</DetailValue>
                                        </DetailItem>
                                        <DetailItem>

<DetailLabel>Experience:</DetailLabel>

```

```

<DetailValue>{gamerProfile?.experience || 'No experience
provided'}</DetailValue>
                                </DetailItem>
                                <DetailItem>

<DetailLabel>Location:</DetailLabel>

<DetailValue>{gamerProfile?.location || 'Not
specified'}</DetailValue>
                                </DetailItem>
                                </ProfileDetails>
                                })
                                </ProfileCard>

                                <MediaSection>
                                  <SectionTitle>Media Gallery</SectionTitle>
                                  <UploadButton onClick={() =>
document.getElementById('media-upload')?.click()}>
                                    Upload Media
                                  </UploadButton>
                                  <input
                                    id="media-upload"
                                    type="file"
                                    accept="image/*,video/*"
                                    style={{ display: 'none' }}
                                    onChange={handleMediaUpload}
                                  />
                                  <MediaGrid>
                                    {/* Відображення медіа файлів */}
                                  </MediaGrid>
                                </MediaSection>
                                </Container>
                                );
                                };

import { DataSourceOptions } from 'typeorm';
import { ConfigService } from '@nestjs/config';

export const databaseConfig = (configService:
ConfigService): DataSourceOptions => ({
  type: "postgres",
  host: configService.get<string>("DB_HOST", "localhost"),
  port: configService.get<number>("DB_PORT", 5432),

```

```

        username: configService.get<string>("DB_USERNAME",
"postgres"),
        password: configService.get<string>("DB_PASSWORD",
"postgres"),
        database: configService.get<string>("DB_NAME",
"sport_platform"),
        entities: [__dirname + "/../../**/*entity{.ts,.js}"],
        migrations: [__dirname + "/../migrations/*{.ts,.js}"],
        synchronize: configService.get<boolean>("DB_SYNCHRONIZE",
true),
        logging: configService.get<boolean>("DB_LOGGING", false),
        ssl: configService.get<boolean>("DB_SSL", false),
    });

```

```
// Головной модуль добавки
```

```

@Module({
    imports: [
        ConfigModule.forRoot({
            isGlobal: true,
            envFilePath: '.env',
        }),
        TypeOrmModule.forRootAsync({
            imports: [ConfigModule],
            useFactory: databaseConfig,
            inject: [ConfigService],
        }),
        AuthModule,
        UsersModule,
        TeamModule,
        TournamentModule,
        GroundModule,
        MediaModule,
        MatchModule,
    ],
    controllers: [AppController],
    providers: [AppService],
})
export class AppModule {}

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy)
{
    constructor(
        private userService: UsersService,

```

```

        private configService: ConfigService,
    ) {
        super({
            jwtFromRequest:
ExtractJwt.fromAuthHeaderAsBearerToken(),
            ignoreExpiration: false,
            secretOrKey:
configService.get<string>("JWT_SECRET") || 'defaultSecretKey',
        });
    }

    async validate(payload: any): Promise<any> {
        const user = await
this.userService.findByEmail(payload.username);
        if (!user) {
            throw new UnauthorizedException('Invalid
user');
        }
        return { email: user.email, userId: user.id };
    }
}

// JWT Guard для захисту маршрутів
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
    canActivate(context: ExecutionContext) {
        return super.canActivate(context);
    }

    handleRequest(err: any, user: any) {
        if (err || !user) {
            throw err || new UnauthorizedException();
        }
        return user;
    }
}

export class LoginDto {
    @IsEmail({}, { message: 'Please enter a valid email
address' })
    @NotEmpty({ message: 'Email is required' })
    email!: string;

    @NotEmpty({ message: 'Password is required' })

```

```

        @IsString()
        @MinLength(6, { message: 'Password must be at least 6
characters long' })
        @MaxLength(15, { message: 'Password must not exceed 15
characters' })
        password!: string;
    }

    export class RegisterDto {
        @IsEmail({}, { message: 'Please enter a valid email
address' })
        @NotEmpty({ message: 'Email is required' })
        email!: string;

        @NotEmpty({ message: 'Password is required' })
        @IsString()
        @MinLength(6, { message: 'Password must be at least 6
characters long' })
        @MaxLength(15, { message: 'Password must not exceed 15
characters' })
        password!: string;

        @NotEmpty({ message: 'Please select a user type' })
        @IsString()
        @IsIn(['PLAYER', 'ORGANIZER', 'VENUE_OWNER'], {
            message: 'User type must be PLAYER, ORGANIZER, or
VENUE_OWNER'
        })
        user_type!: string;
    }

    // Профіль гравця
    export class UpdateGamerProfileDto {
        @IsString()
        @IsOptional()
        name?: string;

        @IsNumber()
        @IsOptional()
        @Min(16, { message: 'Age must be at least 16' })
        @Max(50, { message: 'Age must not exceed 50' })
        age?: number;

        @IsString()

```

```

        @IsOptional()
        @IsIn(['Male', 'Female', 'Other'], { message: 'Sex must
be Male, Female, or Other' })
        sex?: string;

        @IsString()
        @IsOptional()
        location?: string;

        @IsNumber()
        @IsOptional()
        @Min(1, { message: 'Skill level must be between 1 and
10' })
        @Max(10, { message: 'Skill level must be between 1 and
10' })
        skill_lvl?: number;

        @IsString()
        @IsOptional()
        experience?: string;

        @IsString()
        @IsOptional()
        position?: string;
    }

    // Турнир
    export class CreateTournamentDto {
        @IsString()
        @NotEmpty({ message: 'Tournament name is required' })
        @MinLength(3, { message: 'Tournament name must be at
least 3 characters long' })
        name!: string;

        @IsString()
        @IsOptional()
        description?: string;

        @IsString()
        @NotEmpty({ message: 'Tournament format is required'
    })
        @IsIn(['knockout', 'league', 'group-knockout'], {
            message: 'Format must be knockout, league, or
group-knockout'

```

```

    })
    format!: string;

    @IsNumber()
    @Min(2, { message: 'Maximum teams must be at least 2'
  })
    @Max(64, { message: 'Maximum teams must not exceed 64'
  })
    maxTeams!: number;

    @IsDateString({}, { message: 'Registration deadline
must be a valid date' })
    registrationDeadline!: string;

    @IsDateString({}, { message: 'Start date must be a
valid date' })
    startDate!: string;

    @IsDateString({}, { message: 'End date must be a valid
date' })
    endDate!: string;

    @IsNumber()
    @IsOptional()
    @Min(0, { message: 'Entry fee cannot be negative' })
    entryFee?: number;

    @IsNumber()
    @IsOptional()
    @Min(0, { message: 'Prize pool cannot be negative' })
    prizePool?: number;

    @IsEmail()
    @IsNotEmpty()
    organizerEmail!: string;
  }

```

ДОДАТОК В
Слайди презентації

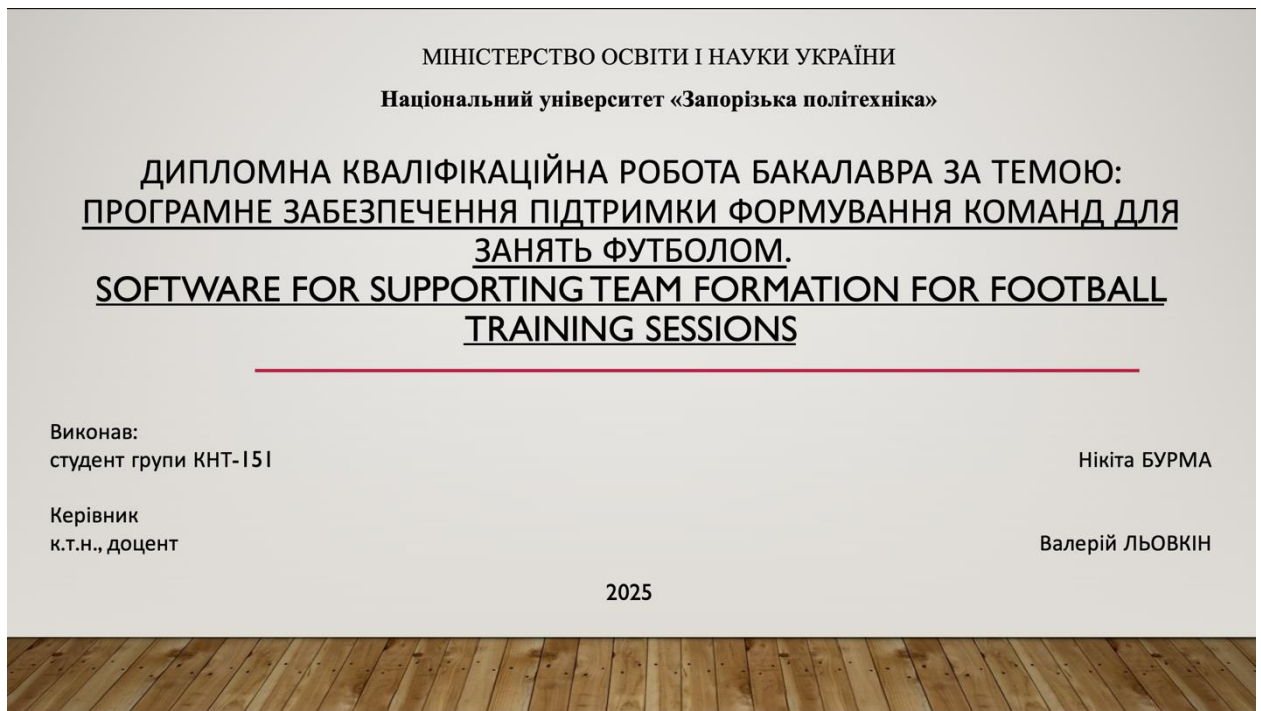


Рисунок В.1 – Титульний слайд

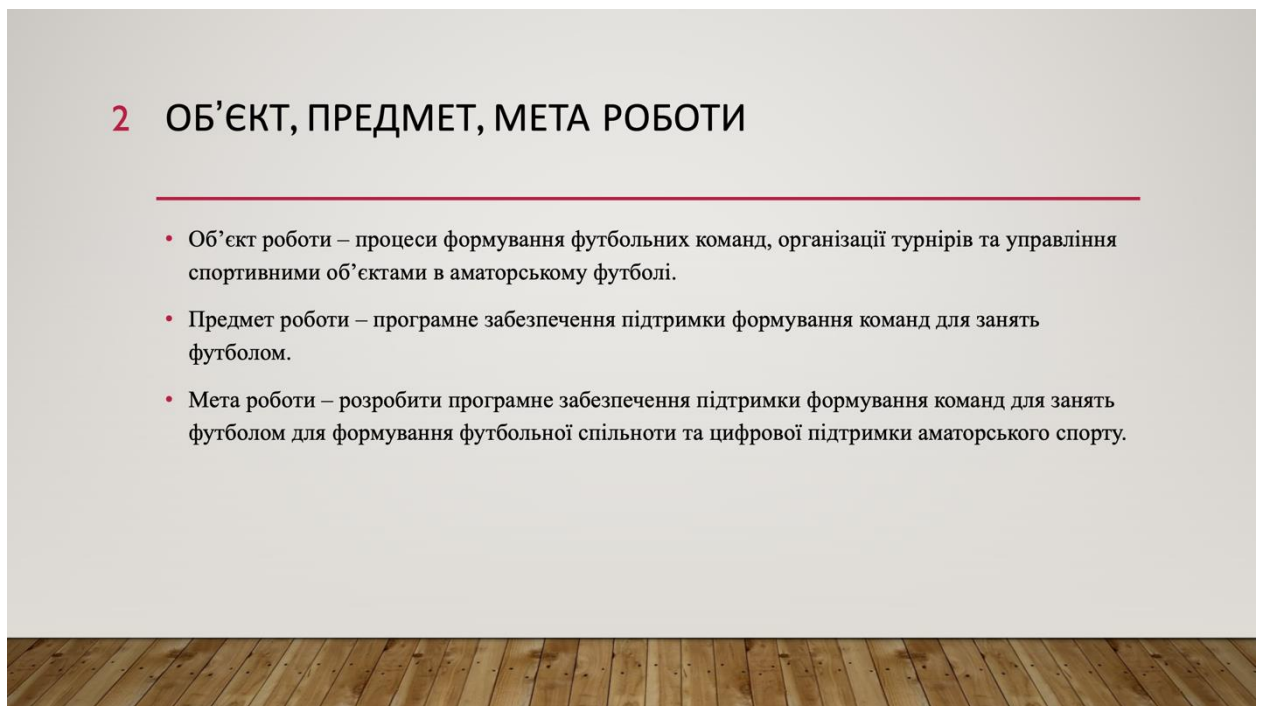


Рисунок В.2 – Об’єкт, предмет, мета роботи

3 ЗАВДАННЯ РОБОТИ

- аналіз існуючого стану організації аматорського футболу та організаційних аспектів координації між різними його учасниками;
- визначення вимог до ПЗ підтримки формування команд для занять футболом;
- проектування ПЗ підтримки формування команд для занять футболом;
- реалізація і тестування ПЗ підтримки формування команд для занять футболом.

Рисунок В.3 – Завдання роботи

4 РЕЗУЛЬТАТИ ПОРІВНЯННЯ АНАЛОГА ТА РОЗРОБКИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

Критерій	Meetup	Playtomic	HEJA	SportPlatform
Реєстрація гравців	Є	Є	Є	Є
Профілі команд	Немає	Немає	Є	Є
Верифікація	Немає	Є	Немає	Є
Пошук гравців	Обмежений	Немає	Немає	Є
Заявки на вступ	Немає	Немає	Немає	Є
Пошук майданчиків	Немає	Є	Немає	Є
Онлайн бронювання	Немає	Є	Немає	Є
Статистика гравців	Немає	Немає	Є	Є
Чат	Є	Немає	Немає	Немає

Рисунок В.4 – Результати порівняння аналога та розробки програмного забезпечення підтримки формування команд для занять футболом

5 ДІАГРАМА ПРЕЦЕДЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

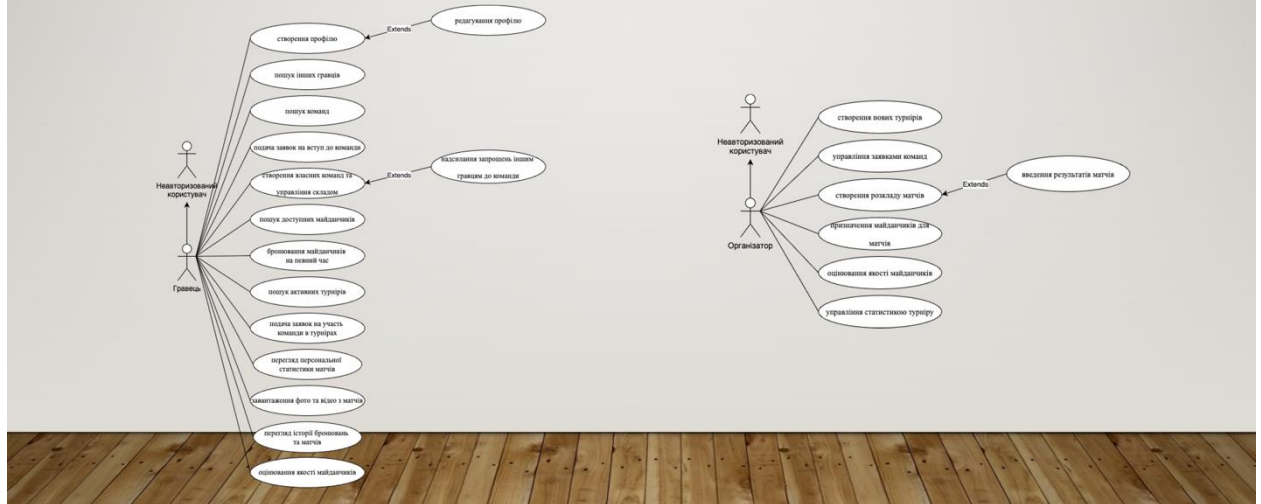


Рисунок В.5 – Діаграма прецедентів програмного забезпечення підтримки формування команд для занять футболом

6 ПОРІВНЯННЯ NODE.JS ФРЕЙМВОРКІВ ДЛЯ СЕРВЕРНОЇ РОЗРОБКИ

Критерій	NestJS	Express.js	Fastify
Архітектурний підхід	Модульна архітектура з DI	Мінімалістичний, middleware-based	Плагін-орієнтована архітектура
TypeScript підтримка	Нативна, повна інтеграція	Потребує додаткової конфігурації	Часткова підтримка
Продуктивність	Висока	Середня	Висока
Тестування	Вбудовані інструменти	Потребує сторонніх бібліотек	Базова підтримка
Масштабованість коду	Відмінна (модулі, сервіси)	Потребує власної організації	Добра (через плагіни)
Документація	Детальна, з прикладами	Мінімальна	Добра
Розмір спільноти	56k GitHub stars	62k GitHub stars	29k GitHub stars

Рисунок В.6 – Порівняння Node.js фреймворків для серверної розробки

7 ПОРІВНЯННЯ РЕЛЯЦІЙНИХ СУБД

Критерій	PostgreSQL	MySQL	MariaDB
Геопросторові функції	Повний набір	Базові функції	Базові функції
JSON підтримка	Є	Є	Є
Продуктивність JOIN	Відмінна	Добра	Добра
Транзакції	Повна ACID, MVCC	ACID з обмеженнями	ACID з обмеженнями
Розширюваність	Власні типи, функції	Обмежена	Обмежена
Хмарна підтримка	Всі провайдери	Всі провайдери	Обмежена

Рисунок В.7 – Порівняння реляційних СУБД

8 ПОРІВНЯННЯ FRONTEND ФРЕЙМВОРКІВ

Критерій	React	Angular	Vue.js
Архітектура	Бібліотека, компонентна	Повний фреймворк, MVC	Прогресивний фреймворк
TypeScript	Опціональний	Обов'язковий	Опціональний
Продуктивність	Висока	Середня	Висока
Екосистема	Величезна	Комплексна	Збалансована
Компонентна модель	Функціональні + класові	Класові	Об'єктні
State Management	Redux, MobX, Context	RxJS, NgRx	Vuex

Рисунок В.8 – Порівняння frontend фреймворків

9 СХЕМА БАЗИ ДАНИХ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ



Рисунок В.9 – Схема бази даних програми підтримки формування команд для занять футболом

10

СХЕМА ФУНКЦІОНУВАННЯ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

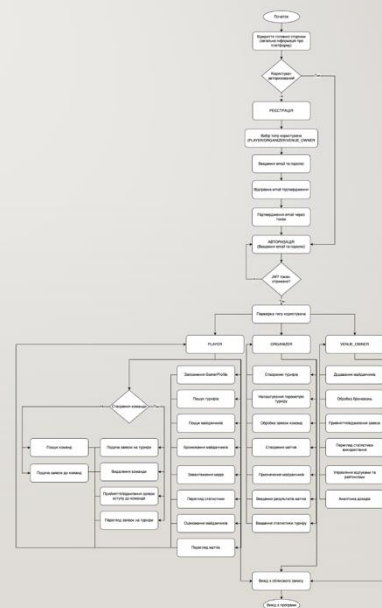


Рисунок В.10 – Схема функціонування програми підтримки формування команд для занять футболом

11 ІНТЕРФЕЙСИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

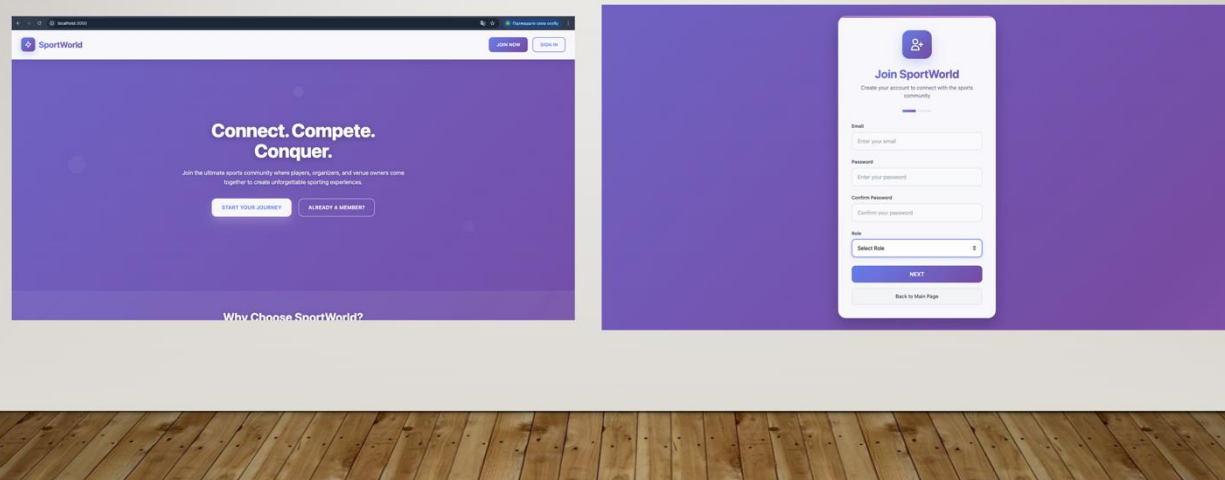


Рисунок В.11 – Інтерфейси програми підтримки формування команд для занять футболом

12 ІНТЕРФЕЙСИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

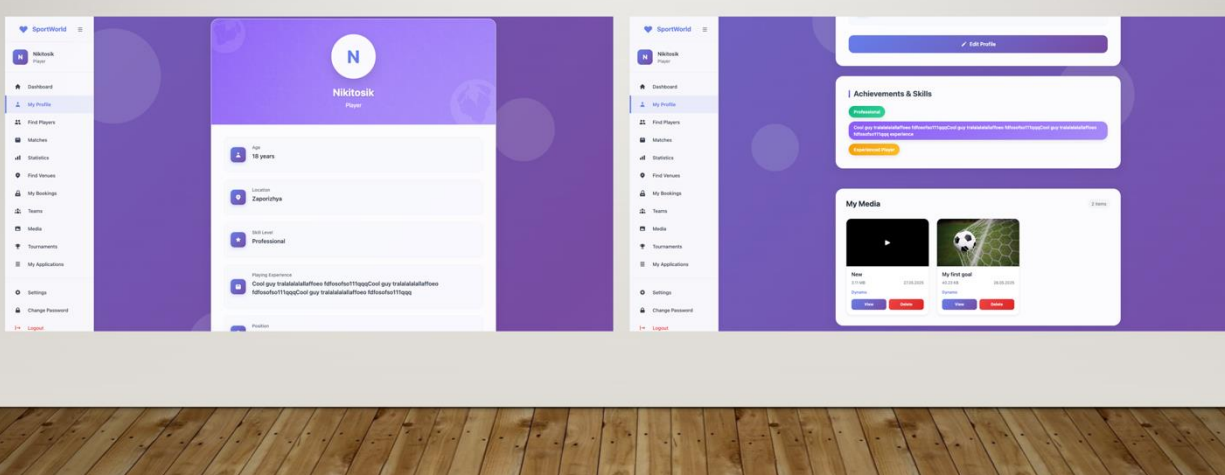


Рисунок В.12 – Інтерфейси програми підтримки формування команд для занять футболом

13 ІНТЕРФЕЙСИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

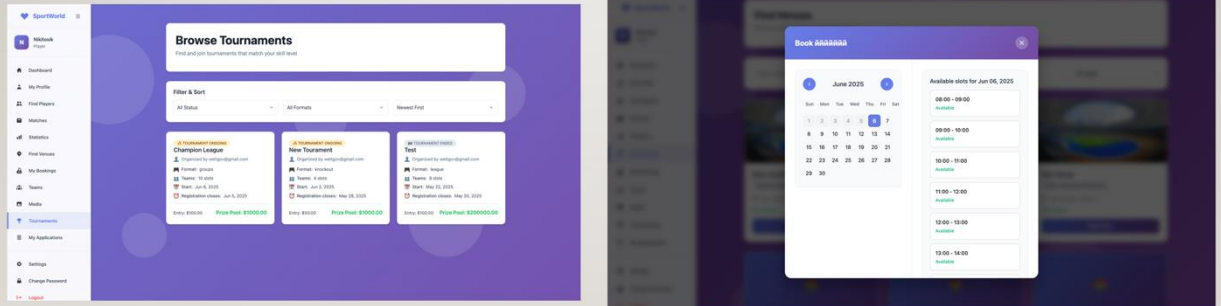


Рисунок В.13 – Інтерфейси програми підтримки формування команд для занять футболом

14 ІНТЕРФЕЙСИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

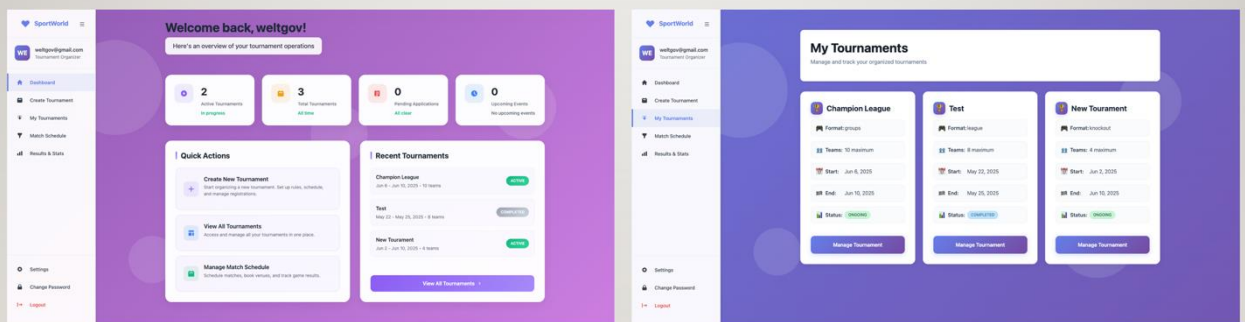


Рисунок В.14 – Інтерфейси програми підтримки формування команд для занять футболом

15 ІНТЕРФЕЙСИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

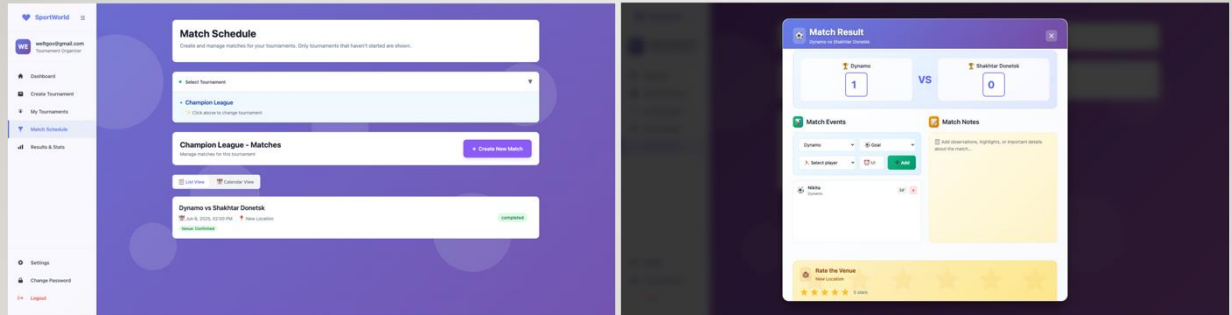


Рисунок В.15 – Інтерфейси програми підтримки формування команд для занять футболом

16 ІНТЕРФЕЙСИ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

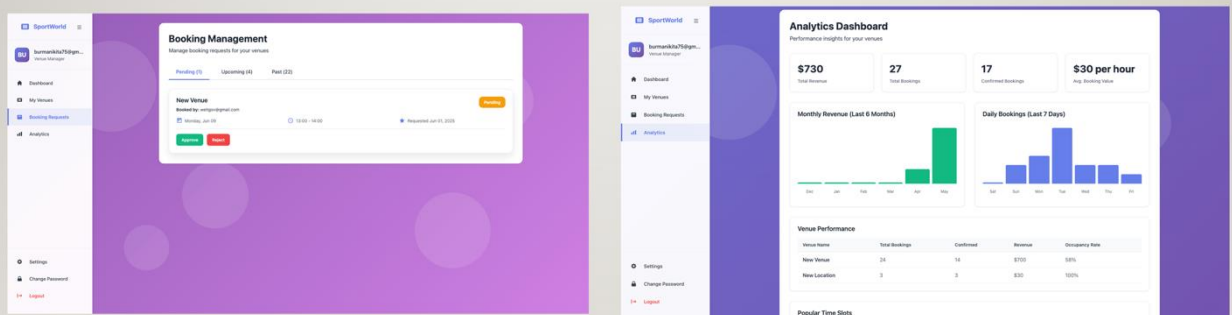


Рисунок В.16 – Інтерфейси програми підтримки формування команд для занять футболом

17 ТЕСТУВАННЯ ПРОГРАМИ ПІДТРИМКИ ФОРМУВАННЯ КОМАНД ДЛЯ ЗАНЯТЬ ФУТБОЛОМ

Функція	Неавторизований користувач	Гравець	Організатор турнірів	Власник майданчиків
I	2	3	4	5
Перегляд головної сторінки	+	+	+	+
Реєстрація	+	+	+	+
Авторизація	+	+	+	+
Перегляд списку турнірів	-	+	+	-
Подання заявки на турнір	-	+	-	-
Створення турніру	-	-	+	-
Управління турніром	-	-	+	-
Обробка заявок на турніри	-	-	+	-
Створення матчів	-	-	+	-

Функція	Неавторизований користувач	Гравець	Організатор турнірів	Власник майданчиків
I	2	3	4	5
Внесення результатів матчів	-	-	+	-
Пошук гравців	-	+	+	-
Перегляд профілів гравців	-	+	-	-
Створення команди	-	+	-	-
Управління командою	-	+	-	-
Пошук команд	-	+	+	-
Перегляд списку майданчиків	-	+	+	+
Бронювання майданчиків	-	+	+	-
Створення майданчика	-	-	-	+
Редагування майданчика	-	-	-	+
Управління бронюваннями	-	-	-	+
Перегляд аналітики доходів	-	-	-	+

Функція	Неавторизований користувач	Гравець	Організатор турнірів	Власник майданчиків
I	2	3	4	5
Завантаження медіа-файлів	-	+	-	+
Редагування профілю	-	+	+	+
Перегляд власної статистики	-	+	+	+

Рисунок В.17 – Тестування програми підтримки формування команд для занять футболом

18 ВИСНОВКИ

- Проведено глибокий аналіз предметної області, що дозволив виявити ключові проблеми організації аматорського футболу в Україні. Встановлено, що основними викликами є фрагментованість спортивної спільноти, відсутність централізованих механізмів взаємодії між учасниками та неефективне використання спортивної інфраструктури. Аналіз існуючих рішень показав відсутність комплексних платформ, що одночасно задовольняли б потреби гравців, організаторів турнірів та власників спортивних майданчиків.
- Здійснено проєктування архітектури системи на основі визначених функціональних вимог користувачів. Розроблено роле-орієнтовану модель доступу для трьох основних типів користувачів: гравців, організаторів турнірів та власників майданчиків. Проведено порівняльний аналіз технологічних рішень та обґрунтовано вибір оптимального стеку: NestJS для серверної частини, React для клієнтського інтерфейсу, PostgreSQL з PostGIS для зберігання даних.
- Реалізовано модульну структуру backend-частини з сервісами автентифікації, управління користувачами, організації турнірів, бронювання майданчиків та ведення статистики. Frontend-частина побудована на компонентному підході з використанням сучасних патернів React.
- Визначено умови експлуатації програми, що включають вимоги до апаратного забезпечення, операційного середовища та мережевої інфраструктури. Детально описано логіку використання системи для кожного типу користувачів з посиланнями на відповідні елементи інтерфейсу. Проведено комплексне тестування функціональності програми за розробленим чек-листом, що охоплює всі ролі користувачів та їх можливості. Результати тестування підтвердили повну функціональну готовність системи до експлуатації в реальних умовах.

Рисунок В.18 – Висновки