

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Веб-додаток для організації та адміністрування
футбольних змагань»**

Виконав студент групи ІПЗкп-23-1
БАЙЛЮ Владислав Вікторович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2025

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«01» серпня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БАЙЛО Владислав Вікторович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Веб-додаток для організації та адміністрування футбольних змагань»**,

затверджена Наказом закладу вищої освіти від **«01» серпня 2025 р., №270/с Д**

Вихідні дані для роботи: Об'єктом дослідження є процес організації, адміністрування та моніторингу футбольних змагань за допомогою веб-технологій.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	01.08.2025р.	01.08.2025р.
2	Чижмотря О.В.	01.08.2025р.	01.08.2025р.
3	Чижмотря О.В.	01.08.2025р.	01.08.2025р.

РЕФЕРАТ

Бакалаврська випускна кваліфікаційна робота складається із розробки веб-додатку для організації та адміністрування футбольних змагань, а також супровідної пояснювальної записки. Документ містить 64 сторінок, включає 14 ілюстрацій, 15 таблиць, 30 джерел інформації та 5 додатків.

Основною метою дослідження є створення сучасного веб-додатку, який забезпечить зручну організацію футбольних турнірів, управління командами, формування заявок на матчі та перегляд результатів.

Архітектура рішення побудована на основі клієнт-серверної моделі з використанням стеку технологій MERN (MongoDB, Express.js, React, Node.js). Веб-додаток розроблено з використанням сучасного фреймворку React і бібліотеки Tailwind CSS для створення адаптивного та зручного інтерфейсу.

Користувачі системи мають можливість створювати турніри, формувати команди, подавати заявки на матчі, переглядати розклади, результати, статистику, а також стежити за оновленням даних у режимі реального часу.

Ключові слова: Веб-додаток, MERN, MongoDB, React, Node.js, футбольні змагання.

					ІІЗ.КР.Б – 121 – 25 – ІІЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Веб-додаток для організації та адміністрування футбольних змагань			Літ.	Арк.	Акрушів	
Розроб.		Байло В.В.									
Керівник		Чижмотря О.В.								3	64
								Житомирська політехніка, група ІІЗкп-23-1			
Рецензент		Єфремов Ю.М.									
Зав. каф..		Вакалюк Т.А.									

ABSTRACT

The bachelor's final qualification work consists of the development of a web application for the organization and administration of football competitions, as well as an accompanying explanatory note. The document contains 64 pages, includes 14 illustrations, 15 tables, 30 sources of information and 5 appendices.

The main goal of the research is to create a modern web application that will provide convenient organization of football tournaments, team management, formation of applications for matches and viewing results.

The solution architecture is built on the basis of the client-server model using the MERN technology stack (MongoDB, Express.js, React, Node.js). The web application is developed using the modern React framework and the Tailwind CSS library to create an adaptive and user-friendly interface.

System users have the ability to create tournaments, form teams, submit applications for matches, view schedules, results, statistics, and monitor data updates in real time.

Keywords: Web application, MERN, MongoDB, React, Node.js, football competitions.

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ТА РОЗРОБКА СИСТЕМИ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури веб-орієнтованої системи планування робочого часу	16
1.4 Обґрунтування вибору інструментальних засобів	18
Висновки до першого розділу.....	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ	22
ФУТБОЛЬНИМИ ЗМАГАННЯМИ	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури	
системи	22
2.2 Розробка бази даних системи	28
2.3 Проектування та реалізація алгоритмів роботи	30
2.4 Реалізація функціоналу додатку	32
Висновки до другого розділу	35
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ	36
3.1 Порядок встановлення та налаштування параметрів системи.....	36
3.2 Структура інтерфеєсу	37
3.3 Тестування роботи програмного комплексу.....	40
Висновки до третього розділу	42
ВИСНОВКИ.....	43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	44
ДОДАТКИ.....	47

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface.

БД – База даних

SQL – Structured Query Language.

CRUD – Create, Read, Update, Delete.

MERN – MongoDB, Express.js, React, Node.js.

JWT – JSON Web Token.

UI – User Interface.

UX – User Experience.

REST API – Representational State Transfer Application Programming Interface.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність. Автоматизація проведення футбольних турнірів є актуальною задачею, оскільки традиційне адміністрування змагань (реєстрація команд, складання розкладу матчів, ведення турнірної таблиці) часто виконується вручну або за допомогою розрізних інструментів (таблиць Excel, телефонних дзвінків тощо). Це призводить до помилок, затримок у оновленні результатів та незручностей для учасників і організаторів. Веб-додаток, що автоматизує управління футбольними змаганнями, дозволяє ефективніше планувати матчі, оперативно публікувати результати та забезпечувати прозорість даних для всіх зацікавлених сторін. Цифрове рішення сприятиме розвитку аматорського і дитячо-юнацького футболу, спрощуючи взаємодію між організаторами, тренерами, гравцями та суддями, а також підвищуючи загальний рівень організації турнірів в умовах сучасних вимог діджиталізації спорту.

Мета роботи. Метою випускної роботи є розробка веб-додатку для автоматизації процесів організації та адміністрування футбольних змагань. Зокрема, передбачено створення системи, що надає можливість керувати користувачами різних ролей (адміністратор, організатор, тренер, гравець, суддя), формувати турніри та їх розклад матчів, вести облік команд і гравців, призначати суддів на ігри, фіксувати результати матчів та генерувати турнірну статистику. Такий програмний продукт покликаний забезпечити досягнення основного результату – ефективної автоматизації рутинних задач футбольного турніру, мінімізації людського фактору при обробці даних і підвищення оперативності управління змаганнями.

Завдання роботи. Для досягнення поставленої мети необхідно вирішити наступні завдання: проаналізувати предметну область і виявити перелік задач, що підлягають автоматизації у процесі проведення футбольного турніру, а також розглянути існуючі аналогічні програмні продукти (системи управління спортивними змаганнями) та їх функціональні можливості; визначити архітектуру та загальну структуру веб-додатку, обґрунтувати вибір технологій реалізації (технологічний стек) і сформулювати вимоги до апаратного та програмного

					ІІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечення сервера; спроектувати структурні компоненти системи, включно з базою даних (сутності та зв'язки), модульною архітектурою додатку, а також розробити ключові алгоритми роботи системи і приклади реалізації основних функцій та класів; реалізувати прототип веб-додатку згідно проєкту, виконати опис його інтерфейсу та перевірити працездатність шляхом тестування основних сценаріїв використання системи.

Об'єкт і предмет дослідження. Об'єктом дослідження є процес автоматизації адміністрування футбольних матчів та турнірів. Предметом дослідження виступають вебтехнології та програмні засоби, що використовуються для управління змаганнями, зокрема методи розробки веб-додатків для організації спортивних турнірів (використання сучасних фреймворків, бібліотек та протоколів реального часу).

Методи дослідження. У роботі використано методи системного аналізу для визначення вимог і функцій системи, методи об'єктно-орієнтованого проєктування для побудови структури додатку, а також засоби вебпрограмування на базі MERN-стеку. MERN (MongoDB, Express.js, React, Node.js) було обрано як сучасний популярний стек технологій для розробки динамічних веб-додатків з єдиною мовою програмування JavaScript на клієнті та сервері. При розробці інтерфейсу використовувалися сучасні інструменти фронтенд-розробки (Vite, React, Tailwind CSS). Інформаційною базою дослідження послужили документація обраних технологій, огляди існуючих рішень для спортивних ліг, а також власний аналіз вимог предметної області.

Практичне значення. Результатом роботи є прототип веб-додатку, який може бути впроваджений для організації локальних футбольних ліг та турнірів у містах України. Практичне значення розробки полягає в тому, що система здатна спростити та прискорити проведення змагань на аматорському рівні, підвищити ефективність комунікації між учасниками (наприклад, своєчасне інформування про розклад і результати матчів) та забезпечити централізоване зберігання й обробку даних турніру. Запропонований підхід може бути використаний футбольними школами, дитячо-юнацькими спортивними закладами,

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

аматорськими лігами для цифровізації своєї діяльності. Веб-додаток є універсальним щодо географії застосування та масштабування – за рахунок архітектури він може розгорнутися на сервері і бути доступним у будь-якому регіоні, де є доступ до Інтернету, що робить його придатним для широкого практичного застосування.

Структура роботи. Пояснювальна записка викладена на приблизно 60 сторінках друкованого тексту і складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі (теоретичному) виконано аналіз предметної області та існуючих програмних рішень, обґрунтовано вибір архітектури системи та інструментальних засобів її реалізації, а також сформульовано вимоги до серверної частини. Другий розділ присвячено проєктуванню веб-додатку: визначено сценарії використання (use cases), наведено структуру бази даних згідно з вимогами системи, описано модулі та компоненти додатку, алгоритми ключових процесів і продемонстровано приклади реалізації функціоналу (включно з описом ролей користувачів). Третій розділ містить опис інтерфейсу користувача розробленої системи, особливостей роботи з нею (в тому числі перемикання мови інтерфейсу), інструкції щодо встановлення та налаштування, а також результати тестування основних функцій веб-додатку на прикладах. У заключній частині наведено висновки за результатами роботи і окреслено напрямки подальшого удосконалення системи. Додатки містять допоміжні матеріали: фрагменти програмного коду, діаграми UML (варіанти використання, модель даних), приклади форм інтерфейсу тощо.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ТА РОЗРОБКА СИСТЕМИ

1.1 Постановка задачі

У сучасних умовах цифровізації спорту особливого значення набувають системи автоматизації, що дозволяють підвищити ефективність організації та управління спортивними змаганнями. Зокрема, футбольні турніри потребують швидкого доступу до актуальної інформації про розклад матчів, результати, склади команд та статистичні дані. В умовах великої кількості учасників, обмеженого часу на підготовку та високих вимог до прозорості організації, ручне адміністрування турнірів втрачає свою ефективність, призводить до затримок, помилок та труднощів в комунікації між організаторами, суддями, тренерами, гравцями та глядачами.

Водночас, аналіз існуючих рішень, які представлені на ринку, зокрема таких як Enjore, SportsEngine Tourney, Tournify, свідчить, що переважна більшість з них орієнтована на великі ліги або потребує значних фінансових вкладень. Часто такі сервіси не передбачають зручної локалізації, мають складний функціонал, або не пристосовані для аматорських та локальних змагань в Україні. Крім того, частина рішень не забезпечує миттєвого оновлення рахунків матчів або передбачає складну структуру налаштувань, що не є зручним для невеликих турнірів.

У ході аналізу предметної області було виявлено перелік функціональних вимог, які є обов'язковими для сучасної системи управління футбольними змаганнями: підтримка різних ролей користувачів, таких як адміністратор, організатор, тренер, гравець, суддя; можливість створення та редагування турнірів організатором; створення команд та формування їх складу з наявних гравців; планування матчів із призначенням часу, місця та судді; введення та оновлення результатів матчів у реальному часі; автоматичне формування турнірних таблиць та статистики; зручна навігація та багатомовність інтерфейсу.

Таким чином, сформульовано основну задачу дослідження – розробити веб-додаток, який забезпечить повноцінну автоматизацію організації та адміністрування футбольних змагань з урахуванням потреб локальних аматорських ліг, з можливістю реєстрації учасників, створення турнірів,

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

планування матчів, фіксації результатів, ведення турнірних таблиць та оперативного оновлення даних на клієнтській стороні в режимі реального часу.

Розроблюваний веб-додаток має бути доступним через браузер, підтримувати мультимовність (українська та англійська мови), бути адаптивним для різних пристроїв, мати зручний інтерфейс та забезпечувати надійний обмін даними між клієнтом і сервером. Особлива увага приділяється використанню сучасних технологій, таких як WebSocket для обміну повідомленнями у режимі реального часу та MERN-стек для повноцінної побудови системи з єдиною мовою програмування.

Для досягнення мети необхідно виконати детальний аналіз аналогічних рішень та вимог предметної області, розробити структуру бази даних з урахуванням ролей та зв'язків між сутностями, спроектувати та реалізувати клієнтську і серверну частини додатку з використанням MERN-стеку, забезпечити підтримку мультимовного інтерфейсу, реалізувати обмін даними в реальному часі через WebSocket, а також протестувати систему на коректність виконання основних функцій.

Очікується, що реалізація розробленої системи дозволить суттєво підвищити якість проведення футбольних змагань, зменшити ризик помилок, забезпечити оперативну комунікацію між учасниками турніру та створити зручну платформу для локальних футбольних ліг в Україні.

1.2 Аналіз аналогів програмного продукту

Перед початком розробки нового веб-додатку для організації та адміністрування футбольних змагань важливо провести аналіз існуючих програмних рішень. Вивчення аналогів дозволяє виявити сильні сторони, слабкі місця та обмеження наявних продуктів, що, своєю чергою, дає змогу розробити конкурентоспроможну систему, яка зможе ефективно задовольнити потреби кінцевих користувачів. Аналіз аналогічних систем дає змогу чітко зрозуміти очікування аудиторії, а також сформулювати вимоги до функціоналу та інтерфейсу власної розробки.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

У цьому розділі розглянуто основні популярні програмні платформи для організації футбольних турнірів: Enjore, Tournify, SportsEngine Tourney, Tournament Software та Tournament Manager. Для кожного продукту буде проведено огляд функціональних можливостей, визначено переваги та недоліки, а також сформовано висновок про відмінності розроблюваної системи.

Enjore[1] – це популярна платформа для організації спортивних змагань (рис. 1.1), зокрема футбольних турнірів. Сервіс дозволяє створювати турніри, формувати розклади матчів, вести турнірні таблиці, а також публікувати новини та статистику.

Серед основних переваг варто виокремити зручний інтерфейс та мобільну адаптацію, що забезпечує комфортне користування на різних пристроях. Система дозволяє створювати турніри будь-якої складності, включно з груповими етапами, плейоф та автоматичною побудовою календаря матчів. Додатковою перевагою є наявність публічного вебсайту з результатами турніру.

Водночас платформа має певні недоліки. Зокрема, повноцінний функціонал доступний лише за підпискою, безкоштовна версія має суттєві обмеження. Також система не підтримує українську мову та має обмежені можливості кастомізації для локальних турнірів.

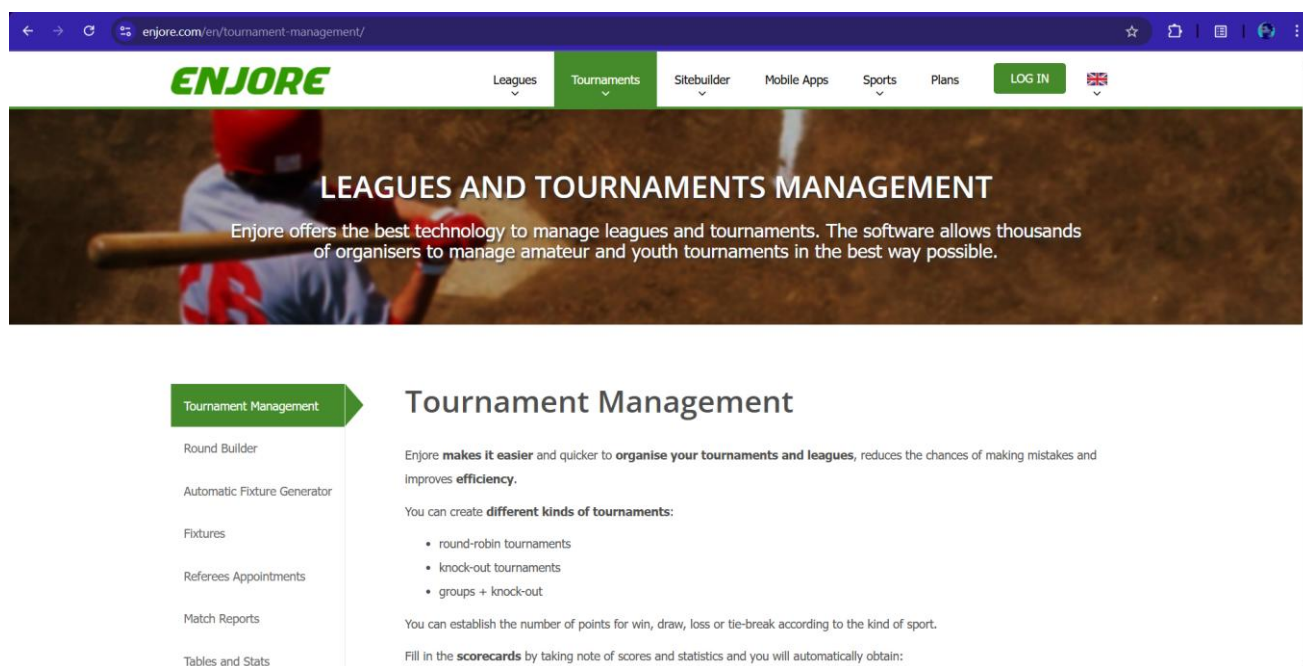


Рис. 1.1 – Сторінка з інформацією про створення турнірів

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Tournify[2] – це вебзастосунок (рис. 1.2), який дозволяє організовувати спортивні турніри та створювати розклади матчів за допомогою візуального редактора. Платформа автоматично формує турнірні таблиці та публікує їх онлайн.

Основними перевагами Tournify є простота у використанні, можливість планувати матчі з урахуванням часових обмежень, а також доступ до платформи з будь-якого пристрою. Інтерфейс зрозумілий і не потребує спеціальної підготовки для використання.

До недоліків слід віднести те, що безкоштовна версія платформи є обмеженою, а оновлення результатів не здійснюється в реальному часі. Платформа підтримує переважно англійську мову, а система управління ролями користувачів реалізована недостатньо гнучко.

Рис. 1.2 – Сторінка створення нового турніру

SportsEngine Tourney[3] – це комплексна система для управління спортивними турнірами (рис. 1.3), яка широко використовується у США та Канаді. Платформа надає розширені можливості адміністрування турнірів, підтримку мобільних додатків та інтеграцію систем оплати.

Основними перевагами є потужний функціонал для ведення статистики, автоматичне формування звітів та підтримка push-сповіщень через мобільний

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

застосунок. Платформа також підтримує онлайн-оплату турнірних внесків та дозволяє проводити турніри будь-якого масштабу.

Недоліком є висока вартість використання платформи, складна система налаштувань, відсутність української локалізації та орієнтації на локальні аматорські ліги. Для невеликих організаторів платформа є складною у впровадженні.

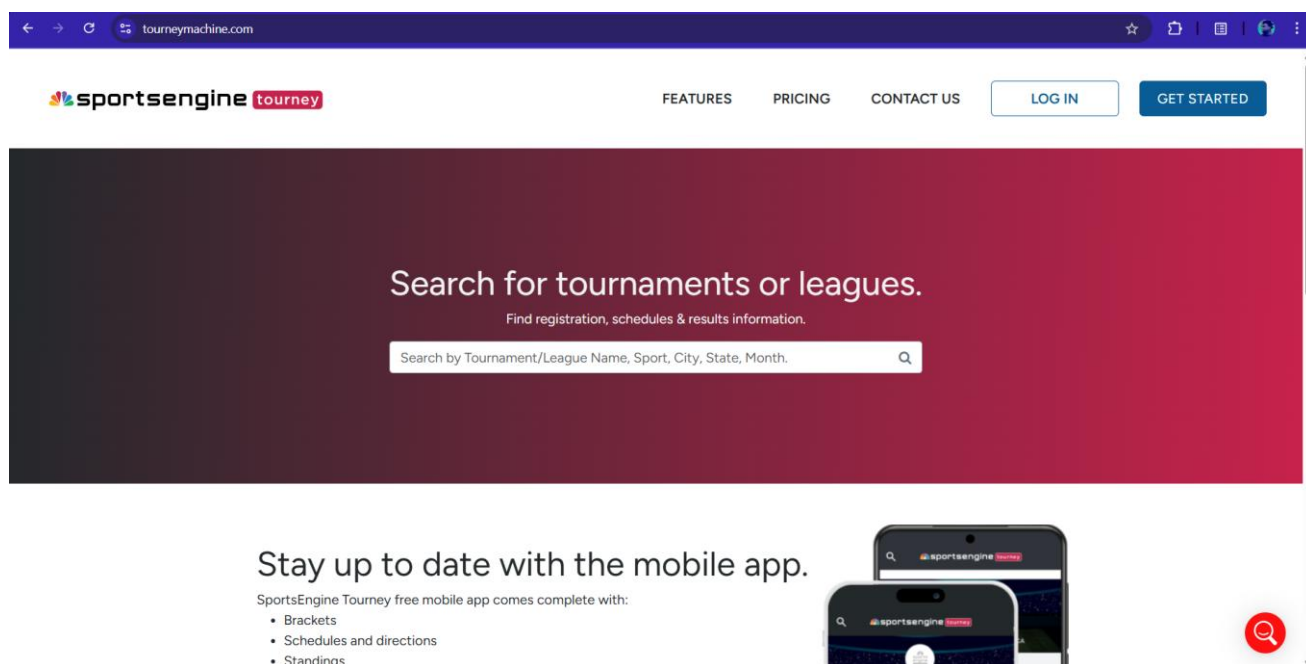


Рис. 1.3 – Головна сторінка з веб-додатку SportsEngine Tourney

Tournament Software[4] – це програмне забезпечення для організації турнірів у різних видах спорту (рис. 1.4). Платформа дозволяє створювати турнірні сітки, вести розклади матчів та забезпечує базову статистику.

Основною перевагою є гнучкість у побудові турнірної сітки, можливість підтримки різних форматів змагань (групи, сітки, плей-оф), а також наявність додаткових модулів для розширення функціоналу.

До недоліків слід віднести обмежену роботу з ролями користувачів, відсутність оновлення в реальному часі та вузьку спеціалізацію під конкретні види спорту. Крім того, інтерфейс не є адаптованим для мобільних пристроїв.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

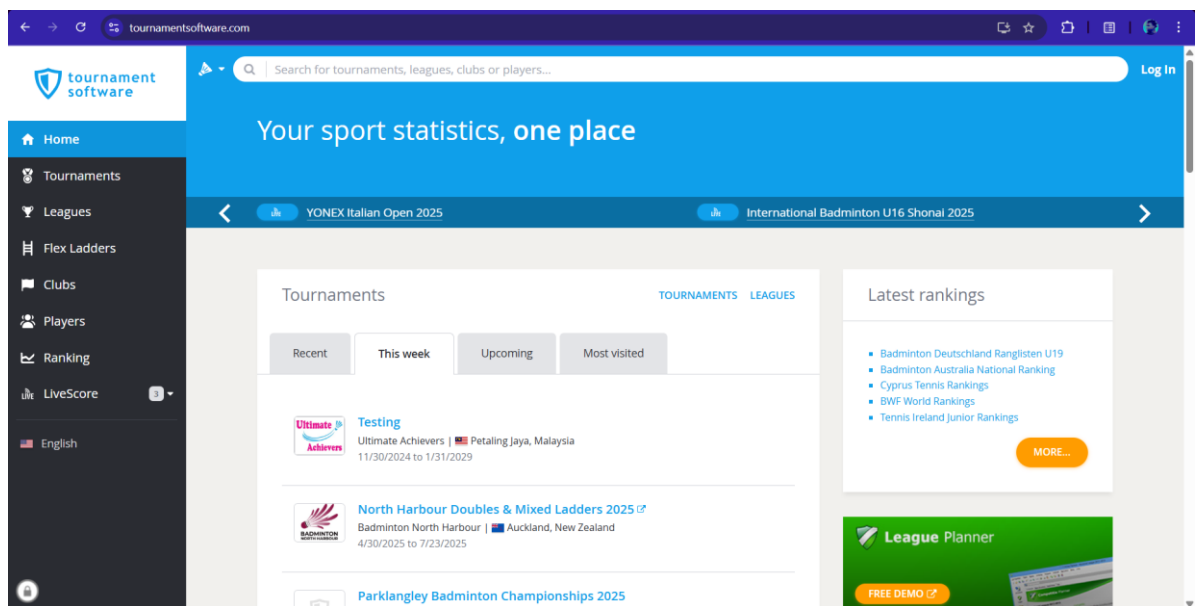


Рис. 1.4 – Головна сторінка з вибором турніру

На основі отриманих даних, складемо порівняльну таблицю (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця аналогів

Критерій	Enjore	Tournify	SportsEngine Tourney	Tournament Software	OpenFootball
Створення турнірів	+	+	+	+	+
Автоматичне формування календаря	+	+	+	+	+
Робота в реальному часі	-	-	+	-	+
Підтримка різних ролей	+/-	+/-	+	-	+
Мультимовність	+/-	+/-	-	+/-	+ ua/en
Інтерфейс для мобільних	+	+	+	+/-	+
Вартість	Платно	Платно	Платно	Платно	Безкоштовно
Оновлення в реальному часі	-	-	+	-	+

Після того, як був проведений аналіз аналогічних рішень можна зробити висновок, що більшість сучасних платформ орієнтовані на великі спортивні організації та потребують фінансових витрат для повноцінного використання. Частина продуктів не передбачає підтримки української мови, не забезпечує оновлення в реальному часі або має обмеження у гнучкому налаштуванні прав доступу для різних ролей користувачів.

Веб-додаток OpenFootball має такі переваги: безкоштовний доступ до всього функціоналу, підтримку багаторівневої системи ролей (адміністратор, організатор, тренер, гравець, суддя), роботу в режимі реального часу з використанням WebSocket, підтримку української та англійської мов, адаптивний інтерфейс, зручний як для десктопних, так і для мобільних пристроїв, а також орієнтацію на локальні турніри з урахуванням потреб українського ринку. Таким чином, розроблювана система має всі шанси стати зручною, доступною та ефективною платформою для організації футбольних змагань різного масштабу, з урахуванням потреб місцевих організаторів.

1.3 Вибір архітектури веб-орієнтованої системи планування робочого часу

Для розробки веб-додатку OpenFootball було обрано клієнт-серверну архітектуру на основі технологічного стеку MERN (MongoDB, Express.js, React, Node.js)[5]. Такий підхід дозволяє реалізувати зручний і динамічний користувацький інтерфейс, забезпечити ефективну обробку даних на сервері та гарантувати масштабованість системи.

У клієнт-серверній архітектурі клієнт (веб-додаток на React) надсилає HTTP-запити до сервера (на Node.js з Express.js), сервер обробляє запити, звертається до бази даних (MongoDB) і повертає результати у відповідь клієнту.

Основна взаємодія між клієнтом і сервером реалізована за допомогою REST API, яке дозволяє зручно управляти ресурсами додатку через стандартні HTTP-методи.

API (Application Programming Interface) – це сукупність методів та правил, за допомогою яких різні програмні компоненти можуть обмінюватися даними [6].

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

У веброзробці API фактично реалізується на сервері, щоб клієнтські додатки могли надсилати запити й отримувати відповіді.

REST (Representational State Transfer) – це стиль побудови веб-API, що базується на ідеях використання HTTP-методів (GET, POST, PUT, DELETE та інші) для взаємодії з ресурсами. Кожна URL-адреса в REST-додатку представляє певний ресурс, а зміна цього ресурсу здійснюється через стандартні HTTP-запити. Такий підхід забезпечує простоту, зручність та масштабованість веб-додатків [7].

Передача даних між клієнтом і сервером здійснюється у форматі JSON (JavaScript Object Notation), який є зручним і широко використовуваним стандартом для сучасних веб-додатків.

JSON (JavaScript Object Notation) – це легкий та універсальний формат обміну даними, який широко використовується у веб-додатках, разом особливо з REST API. JSON має структуроване представлення даних у вигляді «ключ-значення», що робить його зручним для читання людиною та ефективним для обробки програмами. Більшість сучасних мовних програм підтримує роботу з JSON, що робить його ідеальним вибором для передачі даних між клієнтом і сервером [8].

Для розробки адаптивного та сучасного інтерфейсу було використано Tailwind CSS – зручний та гнучкий фреймворк для швидкої верстки [9].

У додатку реалізовано розмежування прав доступу відповідно до ролей користувачів: Admin має можливість створювати користувачів з ролями organizer, coach, player, referee та здійснювати повне управління контентом сайту через окрему адмін-панель; Organizer відповідає за створення турнірів, команд, призначення тренерів, гравців та суддів; Coach має доступ до управління заявками на матчі; Player і Referee можуть переглядати власну статистику; Viewer отримує доступ до публічної інформації без необхідності авторизації.

Реалізація серверної логіки виконується із використанням патерну Controller – Service – Repository, що забезпечує чіткий розподіл відповідальностей у структурі додатку: контролер відповідає за прийом HTTP-запитів, сервіс обробляє бізнес-логіку, а репозиторій взаємодіє з базою даних MongoDB.

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

У веб-додатку для кожної дії передбачено використання Data Transfer Object (DTO), що дозволяє визначати структуру даних для конкретних запитів (реєстрація, оновлення профілю, створення турніру тощо).

Для забезпечення захищеної аутентифікації користувачів застосовується технологія JWT (JSON Web Token). Токен видається користувачу після успішного входу в систему і надалі використовується для авторизації доступу до захищених ресурсів.

У разі подальшого масштабування проєкту доцільно впровадити контейнеризацію за допомогою Docker, що дозволить розгортати серверну частину та базу даних у контейнерах для підвищення зручності адміністрування та розширення проєкту.

Таким чином, обрана архітектура та набір технологій повністю відповідають вимогам до розробки сучасного, масштабованого та зручного у використанні веб-додатку для організації та адміністрування футбольних змагань.

1.4 Обґрунтування вибору інструментальних засобів

Для розробки веб-додатку «OpenFootball» важливо обрати такі інструменти та технології, які повністю відповідають вимогам до створення сучасної, масштабованої та зручної системи.

Visual Studio Code (VS Code) – безкоштовний та один із найпопулярніших редакторів коду з розширеною підтримкою плагінів [10]. Він забезпечує зручне середовище для розробки фронтенду та бекенду, включаючи підтримку JavaScript, TypeScript, Node.js, React та інших технологій. За допомогою плагінів VS Code можна ефективно працювати з REST API, перевіряти правильність запитів, контролювати якість коду та здійснювати тестування.

Postman – інструмент для тестування API, який дає змогу створювати та надсилати HTTP-запити, перевіряти правильність відповіді серверу, формувати колекції запитів для організованої роботи над великими проєктами [11].

Postman дозволяє легко моделювати взаємодію між клієнтом і сервером.

Docker – платформа для контейнеризації, яка забезпечує зручне розгортання серверної частини та бази даних у вигляді ізольованих контейнерів. Використання

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

Docker дозволяє легко переносити додаток на інші сервери та забезпечує стабільність середовища розробки [12].

Використані технології

Клієнтська частина:

React – це сучасний JavaScript-фреймворк для створення динамічних односторінкових додатків. Завдяки віртуальному DOM React забезпечує високу продуктивність при оновленні інтерфейсу без перезавантаження сторінки. Ця технологія ідеально підходить для реалізації функціоналу, який потребує оперативного оновлення даних, наприклад, оновлення рахунку матчу у реальному часі.

Tailwind CSS – це CSS-фреймворк, який дозволяє швидко створювати адаптивний і стильний інтерфейс за допомогою готових класів. Tailwind дає змогу реалізувати сучасний дизайн із мінімальними витратами часу.

i18next – бібліотека для підтримки мультимовності веб-додатку. Дає змогу зручно організувати перемикання мов інтерфейсу без перезавантаження сторінки, що є важливою перевагою для додатка з українською та англійською мовами [13].

Серверна частина:

Node.js – серверне середовище, яке забезпечує швидку обробку запитів завдяки неблокуючій моделі вводу-виводу. Node.js дозволяє створювати високопродуктивні серверні програми для роботи в реальному часі [14].

Express.js – веб-фреймворк для Node.js, який спрощує розробку REST API. Він забезпечує зручну маршрутизацію запитів і підтримку middleware, що полегшує обробку HTTP-запитів [15].

MongoDB – нереляційна база даних, яка забезпечує гнучке зберігання даних у форматі BSON (подібному до JSON). MongoDB дозволяє швидко виконувати операції читання та запису, що критично для динамічних веб-додатків [16].

JWT (JSON Web Token) – технологія для реалізації аутентифікації та авторизації користувачів. Вона забезпечує безпечний контроль доступу до різних частин додатку відповідно до ролей користувачів [17].

Інші інструменти та мови:

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

JavaScript – мова програмування, яка використовується для написання як клієнтської, так і серверної частини додатку. Завдяки універсальності JavaScript дозволяє створювати повноцінний проєкт на основі одного стеку технологій [18].

TypeScript – надбудова над JavaScript, яка додає типізацію та забезпечує додатковий рівень перевірки коду ще на етапі розробки. Це підвищує надійність та якість програмного забезпечення [19].

Вибрані інструменти та технології дозволяють розробити продуктивний, масштабований і зручний у використанні веб-додаток, який забезпечить ефективну організацію футбольних змагань та відповідатиме сучасним вимогам до подібних систем.

Висновки до першого розділу

У першому розділі було сформульовано основні задачі, які необхідно вирішити для створення ефективного веб-додатку для організації та адміністрування футбольних змагань – OpenFootball. Проведено детальний аналіз існуючих аналогів, таких як Enjore, Tournify, SportsEngine Tourney та Tournament Software. У результаті аналізу були виявлені основні переваги та недоліки конкурентних продуктів, що дозволило визначити відмінності та унікальні особливості власної розробки.

Запропонований веб-додаток покликаний усунути виявлені обмеження, забезпечуючи безкоштовний доступ до функціоналу, зручний інтерфейс з підтримкою української мови, роботу в режимі реального часу та чітку структуру ролей користувачів.

На основі дослідження було обґрунтовано вибір клієнт-серверної архітектури з використанням REST API для побудови запитів та Socket.IO для обміну даними в режимі реального часу. Особливу увагу приділено використанню архітектурних патернів Controller – Service – Repository, Dependency Injection та Data Transfer Object, що забезпечують високу модульність, підтримуваність та масштабованість системи.

Також обґрунтовано вибір технологічного стеку MERN (MongoDB, Express.js, React, Node.js), а для розробки сучасного та адаптивного інтерфейсу

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

застосовано Tailwind CSS. Всі прийняті архітектурні та технологічні рішення спрямовані на створення продуктивного, гнучкого та зручного у використанні веб-додатку, здатного забезпечити ефективну організацію футбольних змагань різного масштабу.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФУТБОЛЬНИМИ ЗМАГАННЯМИ

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

У цьому розділі розглянуто основний функціонал веб-додатку для організації та адміністрування футбольних змагань. Головною метою системи є забезпечення зручного середовища для створення, управління та супроводження футбольних турнірів із чітким розмежуванням доступу відповідно до ролей користувачів.

Для наочного представлення основного функціоналу розроблено діаграму варіантів використання (рис. 2.1), яка демонструє взаємодію користувачів із системою.

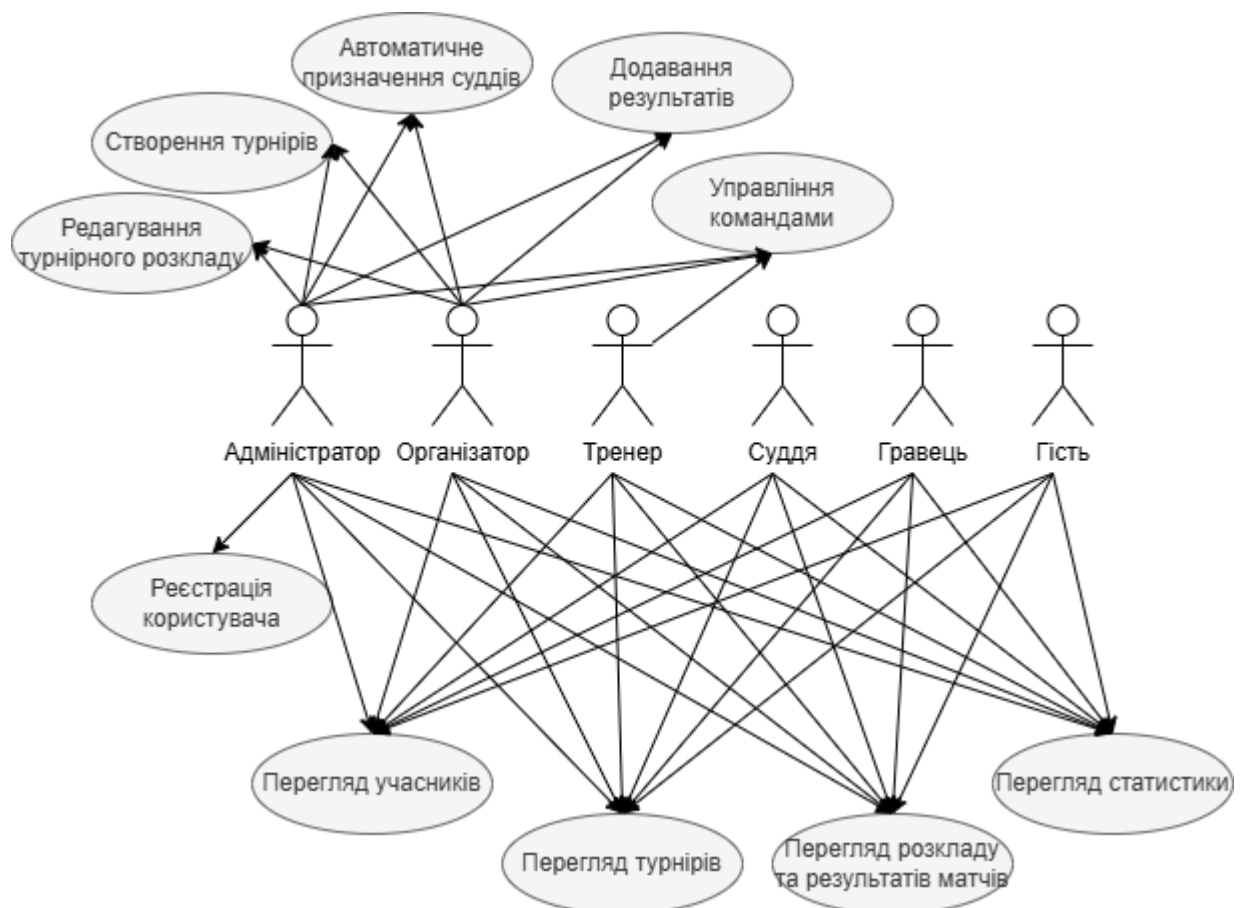


Рис. 2.1 – Діаграма варіантів використання

Опис діаграм варіантів використання, представлених у таблицях 2.1 – 2.10 (табл. 2.1 – 2.10).

Таблиця 2.1

Реєстрація користувача

Назва варіанту використання	Реєстрація користувача
Основні актори	Адміністратор
Короткий опис	Адміністратор реєструє в системі всіх користувачів. Коли користувач зареєстрований він видає логін та пароль користувачу.

Таблиця 2.2

Перегляд учасників

Назва варіанту використання	Перегляд учасників
Основні актори	Адміністратор, Організатор, Тренер, Суддя, Гравець, Гість
Короткий опис	Адміністратор, Організатор, Тренер, Суддя, Гравець та Гість мають можливість переглядати учасників які є в системі на сторінці, а саме Тренерів, Суддів та Гравців.

Таблиця 2.3

Перегляд турнірів

Назва варіанту використання	Перегляд турнірів
Основні актори	Адміністратор, Організатор, Тренер, Суддя, Гравець, Гість
Короткий опис	Адміністратор, Організатор, Тренер, Суддя, Гравець та Гість мають можливість переглядати турніри які є в системі на сторінці.

Таблиця 2.4

Перегляд розкладу та результатів матчів

Назва варіанту використання	Перегляд розкладу та результатів матчів
Основні актори	Адміністратор, Організатор, Тренер, Суддя, Гравець, Гість
Короткий опис	Адміністратор, Організатор, Тренер, Суддя, Гравець та Гість мають можливість переглядати результати матчів та дивитись їх розклад на сторінці.

Перегляд статистики

Назва варіанту використання	Перегляд статистики
Основні актори	Адміністратор, Організатор, Тренер, Суддя, Гравець, Гість
Короткий опис	Адміністратор, Організатор, Тренер, Суддя, Гравець та Гість мають можливість переглядати статистику конкретних матчів.

Таблиця 2.6

Редагування турнірного розкладу

Назва варіанту використання	Редагування турнірного розкладу
Основні актори	Адміністратор, Організатор
Короткий опис	Адміністратор та Організатор мають можливість редагувати турнірний розклад.

Таблиця 2.7

Створення турнірів

Назва варіанту використання	Створення турнірів
Основні актори	Адміністратор, Організатор
Короткий опис	Адміністратор та Організатор мають можливість створювати турніри.

Таблиця 2.8

Автоматичне призначення суддів

Назва варіанту використання	Автоматичне призначення суддів
Основні актори	Адміністратор, Організатор
Короткий опис	Адміністратор та Організатор мають можливість автоматично призначити суддів на матч.

Додавання результатів

Назва варіанту використання	Додавання результатів
Основні актори	Адміністратор, Організатор
Короткий опис	Адміністратор та Організатор мають можливість додавати результати матчу.

Таблиця 2.10

Назва варіанту використання	Управління командами
Основні актори	Адміністратор, Організатор, Тренер
Короткий опис	Адміністратор, Організатор та Тренер мають можливість редагувати команди.

Основна мета створення веб-додатку полягає в наданні користувачам зручного засобу для організації, адміністрування та моніторингу футбольних змагань у режимі реального часу, з доступом до актуальної інформації про турніри, матчі, команди, гравців і суддів.

З урахуванням бізнес-вимог, визначено, що основною функціональною ціллю є розробка веб-додатку, який забезпечуватиме організаторам, адміністраторам, тренерам, гравцям, суддям та глядачам зручний доступ до функціоналу для управління турнірами, формування команд, оперативного оновлення результатів матчів та перегляду статистики. У процесі формування вимог до системи враховувались очікування кінцевих користувачів щодо зручності інтерфейсу, простоти використання, доступності платформи та актуальності інформації.

Проект орієнтовано на локальні футбольні змагання, аматорські турніри та школи, які потребують сучасного, простого у впровадженні та безкоштовного рішення для організації футбольних подій. Це зумовлює необхідність створення доступного, зрозумілого та ефективного інструменту для адміністрування футбольних турнірів.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		25

Організатор та тренер в додатку повинен мати можливість за потребою швидко авторизуватися, щоб отримати доступ до персоналізованого функціоналу. Після входу організатор має можливість створювати та адмініструвати власні турніри. Тренери повинні мати змогу формувати заявки на матчі, а гравці, судді та глядачі переглядати результати та статистику без обмежень.

Основна ідея – забезпечити простий, швидкий та зрозумілий процес організації змагань, управління командами та перегляду інформації. Інтерфейс повинен бути інтуїтивно зрозумілим, виглядати сучасно та привабливо, щоб користувачі могли легко орієнтуватися у функціоналі без потреби в додаткових інструкціях.

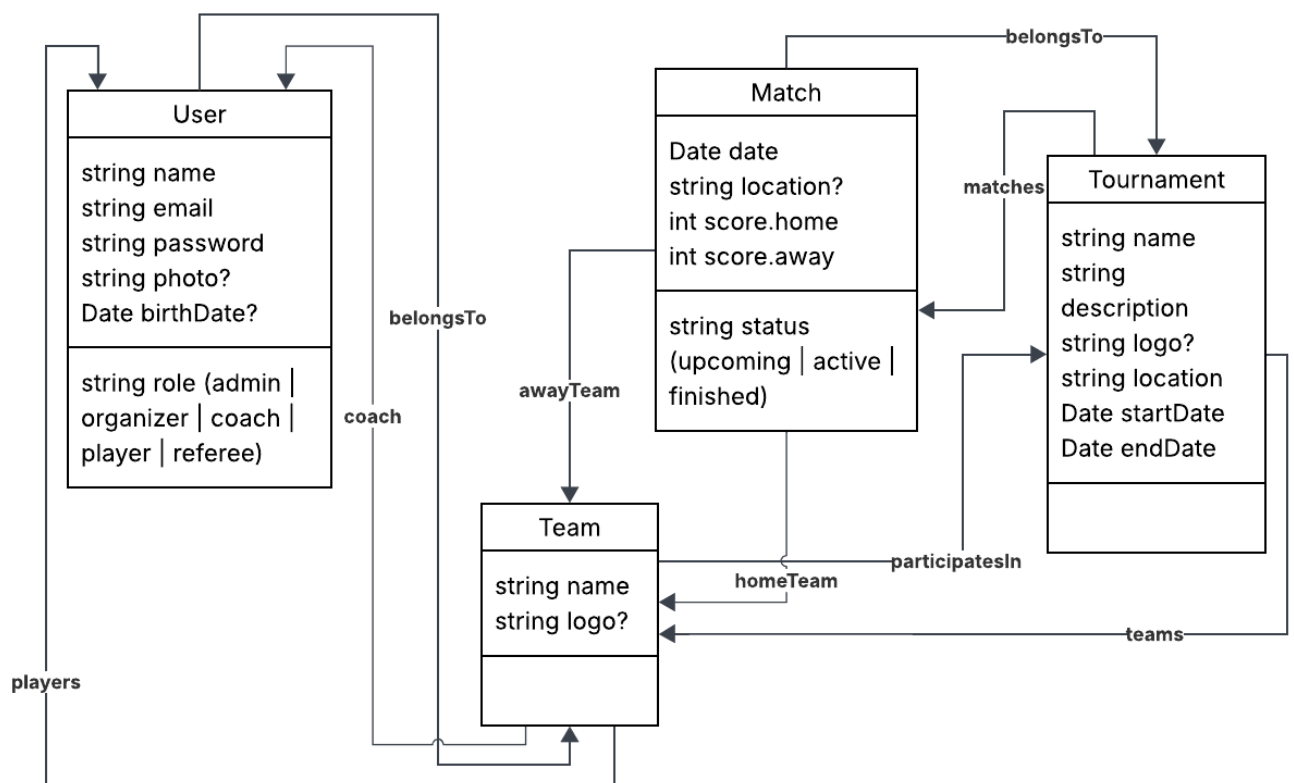


Рис. 2.2 – Діаграма класів

Діаграма класів системи (рис. 2.2) відображає чотири основні сутності, які лежать в основі роботи веб-додатку: User, Team, Tournament та Match. Кожна з них відповідає окремій колекції у базі даних MongoDB та описана у вигляді схеми Mongoose із зазначенням типів полів і зв'язків між об'єктами.

Клас User використовується для зберігання даних про користувачів системи, включно з адміністраторами, організаторами, тренерами, гравцями та суддями. Він містить поля для зберігання імені, унікальної електронної пошти, хешованого пароля, ролі, фотографії та дати народження. Крім цього, користувач може бути пов'язаний із конкретною командою, що зберігається у полі team. Один користувач може належати лише до однієї команди, а тренер може тренувати лише одну команду одночасно.

Клас Team описує футбольну команду та містить інформацію про її назву, логотип, турнір, у якому вона бере участь, тренера та список гравців. Зв'язок із турніром здійснюється через поле tournament, що містить посилання на відповідний об'єкт колекції Tournament. Поле coach вказує на користувача з роллю тренера, а players – на масив користувачів з роллю гравця. Таким чином, одна команда належить одному турніру, має одного тренера та може включати в себе багато гравців.

Клас Tournament зберігає дані про футбольні турніри. Він містить назву, опис, логотип, місце проведення, дати початку та завершення, а також списки команд і матчів, що належать до турніру. Зв'язки реалізовано через масиви teams та matches. Один турнір може включати багато команд і багато матчів. Важливою особливістю є каскадне видалення матчів при видаленні турніру, що запобігає накопиченню «сирітських» записів у базі даних.

Клас Match відповідає за зберігання інформації про футбольні матчі. Він містить посилання на турнір, у рамках якого відбувається матч, дві команди-учасниці, дату, місце проведення, статус (запланований, активний або завершений) та рахунок гри. Кожен матч належить лише одному турніру і завжди має рівно дві команди-учасниці.

Зв'язки між класами побудовані таким чином, щоб відображати реальну бізнес-логіку системи. Один користувач може бути частиною лише однієї команди, але команда може містити багато гравців. Кожна команда бере участь лише в одному турнірі, але турнір може включати багато команд. Один турнір може мати велику кількість матчів, а кожен матч завжди пов'язаний із двома

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

командами. Така структура забезпечує логічну цілісність даних і дозволяє гнучко управляти зв'язками між об'єктами в межах веб-додатку.

2.2 Розробка бази даних системи

У системі використовується база даних, структура якої відповідає сутностям веб-додатку та бізнес-логіці його роботи. Колекції створені за допомогою бібліотеки Mongoose, що дозволяє описувати їх у вигляді схем із відповідною типізацією. Такий підхід спрощує створення, читання та підтримку коду, забезпечуючи зручну роботу з об'єктно-документною моделлю.

Взаємодія з базою даних організована через контролери, які приймають HTTP-запити від клієнта та передають їх у сервіси для обробки. Сервіси, своєю чергою, працюють з репозиторіями, які надають доступ до CRUD-операцій із базою даних MongoDB. Така архітектура забезпечує модульність, чітке розділення обов'язків та полегшує підтримку й масштабування додатку.

Структура бази даних включає декілька основних колекцій, які описують користувачів, ролі, турніри, команди та матчі, забезпечуючи необхідні зв'язки між ними.

Таблиця 2.11 «Users» основна колекція, яка зберігає інформацію про користувачів системи, включно з їх унікальним ідентифікатором, ім'ям, роллю та паролем. Вона пов'язана з іншими колекціями, що розділяють користувачів за ролями: адміністратор, організатор, тренер, гравець, суддя.

Колекції Admins, Organizers, Coaches, Players та Referees посилаються на основну колекцію Users за допомогою зовнішнього ключа. Такий підхід дозволяє зберігати загальні дані про користувача в одній таблиці та додавати специфічну інформацію відповідно до ролі.

Таблиця 2.12 «Tournaments» Колекція Tournaments містить інформацію про створені турніри, включно з назвою, детальною інформацією та посиланням на організатора.

Таблиця 2.13 «Teams» Колекція Teams зберігає дані про команди, включно з назвою та списком гравців, які належать до команди.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.14 «Matches» Колекція Matches зберігає інформацію про заплановані та завершені матчі, включно з датою, часом, локацією, учасниками матчу та результатом. У колекції матчів також зберігаються склади обох команд, що формуються тренерами перед матчем.

Таблиця 2.11

Опис таблиці Users

Поле	Тип	Призначення
id	ObjectId, PK, unique	Унікальний ідентифікатор
email	varchar(255)	Використовується як логін
name	varchar(255)	Ім'я та прізвище користувача
role	varchar(255)	Роль користувача (admin, organizer, coach, player, referee)
photo	varchar(255)	Шлях до фото
password	varchar(255)	Пароль
birthDate	Date	День народження
team	ObjectId, (FK → Team)	Команда

Таблиця 2.12

Опис таблиці Tournaments

Поле	Тип	Призначення
id	ObjectId, PK, unique	Унікальний ідентифікатор
name	varchar(255)	Назва турніру
description	varchar(255)	Опис турніру
logo	varchar(255)	Посилання на логотип турніру
location	varchar(255)	Місце проведення турніру
startDate	Date	Дата початку турніру
endDate	Date	Дата кінця турніру
teams	ObjectId (FK → Team)	Список всіх команд які є в турнірі
matches	ObjectId (FK → Match)	Список всіх матчів які є в турнірі

Таблиця 2.13

Опис таблиці Teams

Поле	Тип	Призначення
id	ObjectId, PK, unique	Унікальний ідентифікатор
name	varchar(255)	Назва команди
logo	varchar(255)	Посилання на логотип команди
tournament	ObjectId (FK → Tournament)	Турнір в якому бере участь
coach	ObjectId (FK → Users)	Тренер команди
players	ObjectId (FK → Users)	Список гравців команди

Таблиця 2.14

Опис таблиці Matches

Поле	Тип	Призначення
id	ObjectId, PK, unique	Унікальний ідентифікатор
tournament	ObjectId (FK → Tournaments)	Посилання на турнір
date	Date	Дата проведення матчу
location	varchar(255)	Локація проведення матчу
homeTeam	ObjectId (FK → Teams)	Перша команда
awayTeam	ObjectId (FK → Teams)	Друга команда
score	varchar(255)	Результат матчу
status	varchar(255)	Статус матчу

2.3 Проектування та реалізація алгоритмів роботи

Веб-додаток для організації та адміністрування футбольних змагань простий у використанні та містить усі необхідні функції для комфортної роботи користувачів із різними ролями. Алгоритми взаємодії між клієнтською частиною та сервером побудовані таким чином, щоб забезпечити швидку обробку запитів та захищену авторизацію.

Одним із базових алгоритмів є процедура авторизації користувача. На діаграмі активностей (рис. 2.3) зображено послідовність обміну даними між клієнтським додатком та сервером під час авторизації.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

Процес авторизації розпочинається з того, що користувач вводить електронну пошту та пароль у клієнтському інтерфейсі веб-додатку. Після цього клієнтська частина формує HTTP-запит і надсилає введені дані на сервер для перевірки.



Рис. 2.3 – Діаграма активностей для авторизації користувача

Сервер обробляє запит, звертаючись до бази даних для пошуку користувача з відповідною електронною поштою. У разі знаходження облікового запису сервер виконує перевірку правильності введеного пароля, порівнюючи його з хешованим паролем у базі даних.

Якщо введені облікові дані правильні, сервер створює JWT-токен, який підтверджує успішну авторизацію користувача та надсилає цей токен у відповідь клієнтському додатку. Токен зберігається у локальному сховищі браузера та використовується для подальших запитів до захищених маршрутів серверу.

У разі введення некоректного пароля або відсутності користувача в базі, сервер надсилає клієнту повідомлення про помилку, яке відображається користувачеві на екрані.

Кожен користувач, залежно від своєї ролі (admin, organizer, coach, player, referee, viewer), має доступ до певного функціоналу, реалізованого на основі чітко побудованих алгоритмів взаємодії із сервером та базою даних. Наприклад, адміністратор створює нових користувачів і керує всією системою, організатор створює турніри, формує команди, призначає тренерів, гравців і суддів, тренер подає заявку на матч у своєму особистому кабінеті, а глядачі мають можливість переглядати розклади, результати, статистику та поточні матчі без реєстрації.

Завдяки чіткій архітектурі, побудованій на основі патерну Controller – Service – Repository, усі запити клієнта обробляються поетапно: від контролера до сервісу, далі до репозиторію, який взаємодіє з базою даних MongoDB.

Алгоритми, що реалізовані у веб-додатку, забезпечують швидку авторизацію, безпечну роботу із захищеними маршрутами, ефективно розмежування прав доступу, а також надійне управління турнірами, командами та матчами.

Це дозволяє створити стабільний, безпечний та зручний веб-додаток для організації футбольних змагань, який відповідає сучасним вимогам до систем реального часу.

2.4 Реалізація функціоналу додатку

Розглянемо ключові етапи реалізації серверної та клієнтської частин веб-додатку для організації та адміністрування футбольних змагань.

Створення нових користувачів у системі (рис. 2.4) виконується через панель адміністратора. Адміністратор вводить ім'я, електронну пошту, пароль та роль користувача (організатор, тренер, гравець, суддя). На сервері перевіряється

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

унікальність email, а пароль шифрується перед збереженням у базу даних. У разі виникнення помилки повертається відповідне повідомлення, що забезпечує стабільність роботи та зручність обробки помилок.

```
router.post('/users', async (req, res) => {
  const { name, email, password, role } = req.body;
  if (!['organizer', 'coach', 'player', 'referee'].includes(role)) {
    return res.status(400).json({ message: 'Invalid role' });
  }

  try {
    const exists = await User.findOne({ email });
    if (exists) {
      return res.status(400).json({ message: 'Email already in use' });
    }

    const hashed = await bcrypt.hash(password, 10);
    const newUser = new User({ name, email, password: hashed, role });
    await newUser.save();
  }
});
```

Система підтримує вхід для користувачів з наступними ролями: admin, organizer, coach. Під час авторизації сервер перевіряє наявність користувача за email та звіряє пароль. У разі успішної авторизації формується JWT-токен, який повертається клієнту разом із основною інформацією про користувача. Це дозволяє організувати розмежування доступу в залежності від ролі.

```
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid credentials' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

    const token = jwt.sign(
      { id: user._id, role: user.role },
      process.env.JWT_SECRET, { expiresIn: '2h' });
    res.json({
      token,
      user: { _id: user._id, name: user.name, role: user.role }
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
```

У подальших запитах клієнт надсилає токен в заголовок авторизації. Сервер перевіряє валідність токена, і в разі успіху дозволяє доступ до захищених маршрутів. Якщо токен відсутній або недійсний, то повертається помилка 401.

```
const jwt = require('jsonwebtoken');
module.exports = function (req, res, next) {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'No token, authorization denied' });
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: 'Token is not valid' });
  }
};
```

Користувач з роллю «організатор» має доступ до створення та налаштування турнірів. Під час створення турніру організатор вказує основні параметри: назву, опис, логотип, місце проведення, дату початку, дату кінця, тип турніру (груповий етап або плей-офф).

```
router.post("/tournaments", async (req, res) => {
  try {
    const {
      name, gender, season, location,
      startDate, groupStage = false,
      groupLegs = 1,
      playoff = false
    } = req.body;
    const tour = new Tournament({
      name, gender, season, location, startDate, groupStage,
      groupLegs, playoff, organizer: req.user.id,
      teams: [] // спочатку без команд
    });
    await tour.save();
    res.status(201).json(tour);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Error creating tournament" });
  }
});
```

Для кожного турніру організатор додає команди, які автоматично додаються до турніру через відповідний серверний маршрут. При створенні команди вона зв'язується з турніром і зберігається у базі даних.

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

```

router.put("/tournaments/:id/add-team", async (req, res) => {
  try {
    const { teamId } = req.body;
    const tour = await Tournament.findOne({ _id: req.params.id, organizer:
req.user.id });
    if (!tour) return res.status(404).json({ message: "Tournament not found"
});
    if (!tour.teams.includes(teamId)) {
      tour.teams.push(teamId);
      await tour.save();
    }
    res.json(tour);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Error adding team" });
  }
});

```

У системі активно використовуються типізовані інтерфейси, які забезпечують правильну структуру обміну даними між клієнтською та серверною частинами. Завдяки цьому вдається мінімізувати помилки при передачі інформації та забезпечити коректну роботу застосунку.

Висновки до другого розділу

У цьому розділі було детально розглянуто, як саме побудований і працює веб-додаток для організації та адміністрування футбольних змагань. Було описано основні сценарії взаємодії користувача із системою.

Архітектура додатку побудована з використанням сучасних перевірених підходів, таких як розділення на контролери, сервіси та репозиторії, що дозволяє підтримувати чітку структуру та спрощує подальший розвиток системи. База даних реалізована з використанням MongoDB, що забезпечує зручне зберігання користувачів, турнірів, команд та матчів у вигляді взаємопов'язаних документів.

Окрему увагу приділено забезпеченню безперервної роботи веб-додатку: реалізовано алгоритми автентифікації, авторизації з розмежуванням прав доступу, створення та редагування основних сутностей. Усі ключові бізнес-процеси адаптовані для ефективного використання організаторами, тренерами, гравцями та суддями у межах одного веб-додатку.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1 Порядок встановлення та налаштування параметрів системи

Розгортання та конфігурація веб-додатку для організації та адміністрування футбольних змагань передбачає покрокове налаштування кожного з його компонентів, а також забезпечення коректної взаємодії усіх частин у єдиній системі.

Для наочного представлення архітектури програмного комплексу та процесу його розгортання підготовлено діаграму розгортання (рис. 3.1).

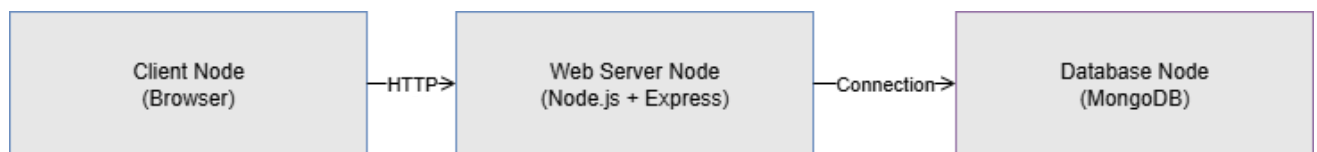


Рис. 3.1 – Діаграма розгортання веб-додатку для футбольних змагань

На діаграмі показано основні компоненти системи та їх взаємодію. Клієнтська частина веб-додатку, яка працює у браузері користувача, взаємодіє із сервером через HTTP-запити з використанням протоколу REST API.

У системі не використовується WebSocket або інші механізми роботи в режимі реального часу. Оновлення інформації на сторінках здійснюється шляхом звичайних HTTP-запитів, що дозволяє забезпечити достатній рівень актуальності даних у межах поставлених бізнес-вимог.

Серверна частина реалізована за допомогою Node.js з використанням фреймворку Express. Для зберігання всіх даних (користувачів, турнірів, команд, матчів) використовується база даних MongoDB, яка забезпечує гнучке документно-орієнтоване збереження та легку масштабованість.

Основні етапи встановлення системи включають встановлення необхідного програмного забезпечення, такого як Node.js, MongoDB та менеджер пакетів npm, клонування серверної та клієнтської частини проєкту з репозиторію, налаштування змінних середовища, зокрема URI підключення до бази даних, секретного ключа для JWT-автентифікації та даних адміністратора. Далі виконується запуск серверної частини та клієнтського додатку, після чого проводиться перевірка доступності основних маршрутів API та коректної роботи

інтерфейсу.

Завдяки спрощеній архітектурі система є зручною для розгортання, обслуговування та подальшого розширення функціоналу.

3.2 Структура інтерфейсу

Користувач взаємодіє із системою через веб-додаток. При першому вході користувач бачить головну публічну сторінку (рис. 3.2), яка надає загальну інформацію про футбольні турніри, актуальні матчі та новини.

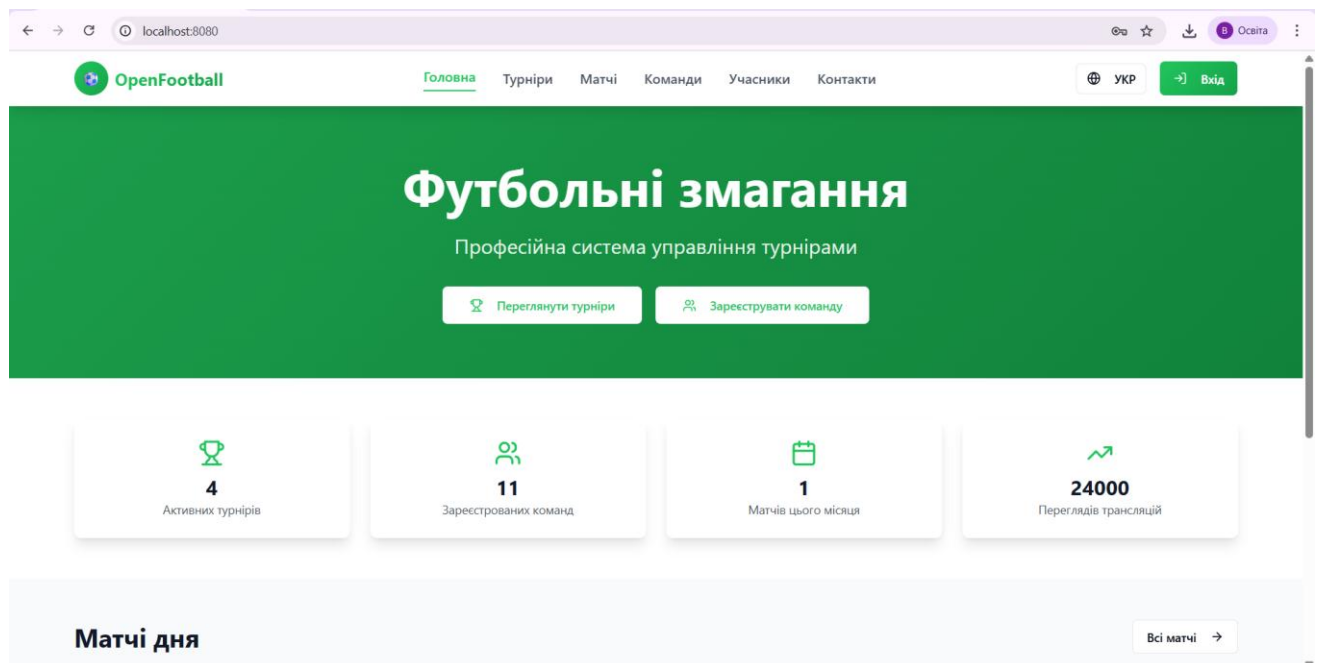


Рис. 3.2 – Головна сторінка веб-додатку

На головній сторінці відображається список турнірів, а також блоки зі статистикою, новинами та найближчими матчами. Доступна навігація до таких розділів: турніри, матчі, команди, учасники та контакти.

Користувач, який має обліковий запис, може натиснути кнопку «Увійти» (рис. 3.3), де відкривається форма авторизації. Після введення email та пароля, користувач перенаправляється на персональну сторінку залежно від своєї ролі.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

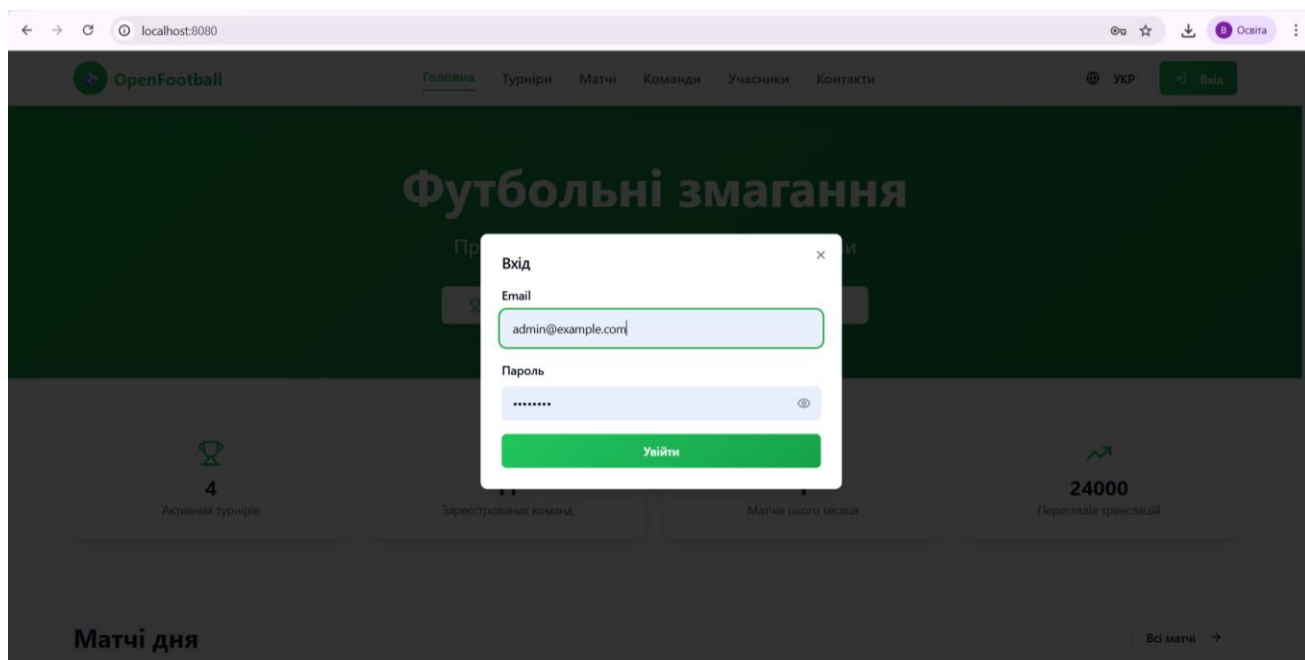


Рис. 3.3 – Форма авторизації

Якщо користувач авторизований з роллю адміністратора, він потрапляє до адміністративної панелі (рис. 3.4), де має змогу створювати, редагувати, видаляти користувачів з різними ролями: організатор, тренер, гравець, суддя.

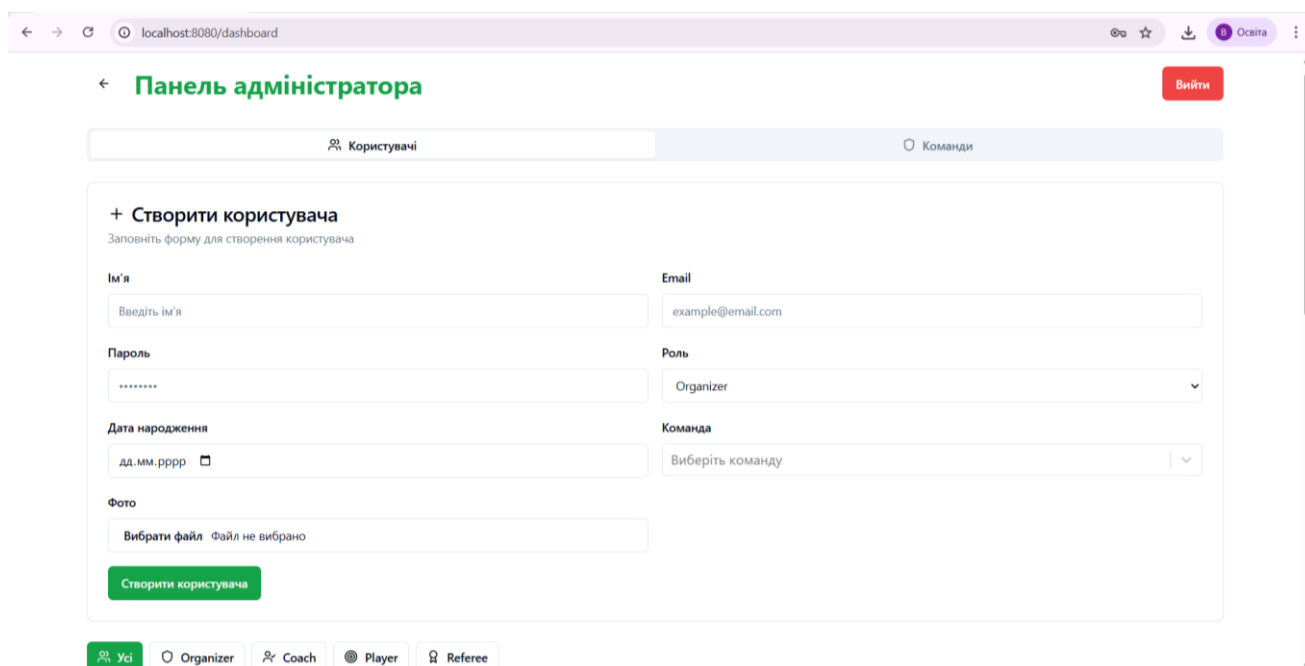


Рис. 3.4 – Панель адміністратора управління користувачами

На сторінці адміністратора всі користувачі розділені за ролями. Доступна функція сторінкової навігації (по 10 записів на сторінку), а також можливість редагувати або видалити будь-якого користувача.

Користувач з роллю організатор потрапляє на сторінку «Мої турніри» (рис. 3.5), де відображаються турніри, які створив саме цей організатор.

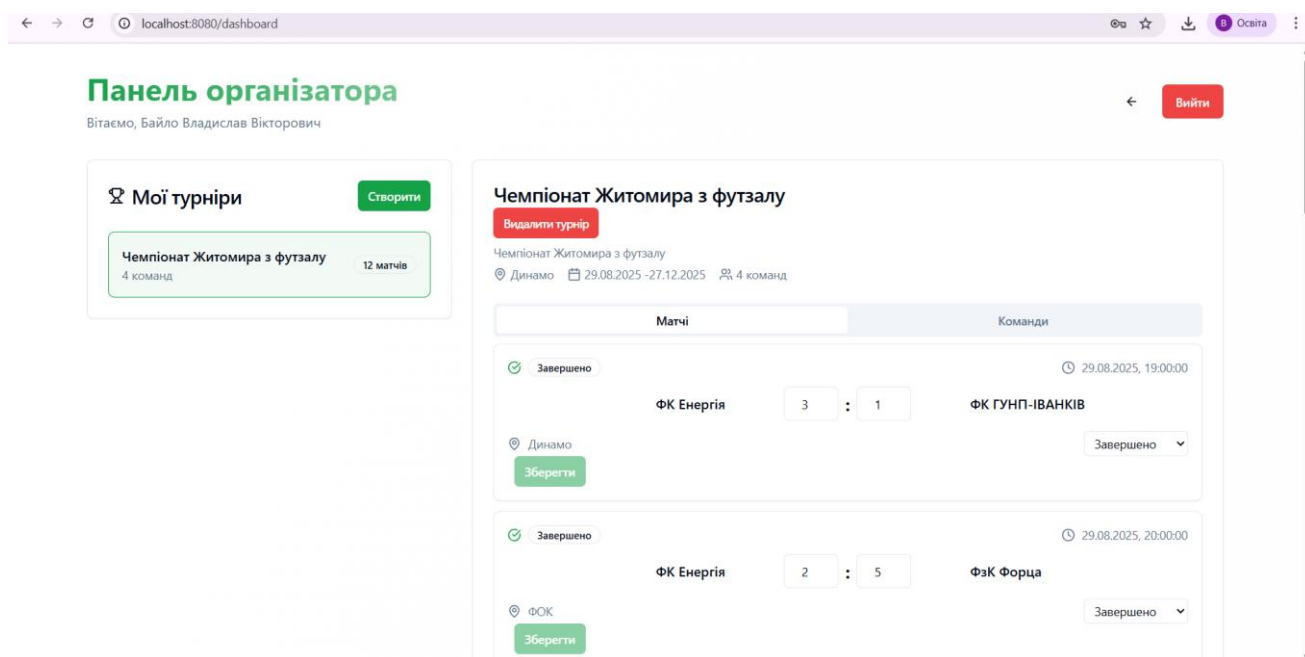


Рис. 3.5 – Панель організатора (список турнірів)

Організатор може створити новий турнір, налаштувати його параметри: назва, опис, локація, дата початку та завершення, список команд, формат турніру (груповий етап з одним або двома колами, або плей-офф) та логотип турніру.

Після натискання кнопки створити організатор може перейти до налаштування турніру (рис. 3.6).

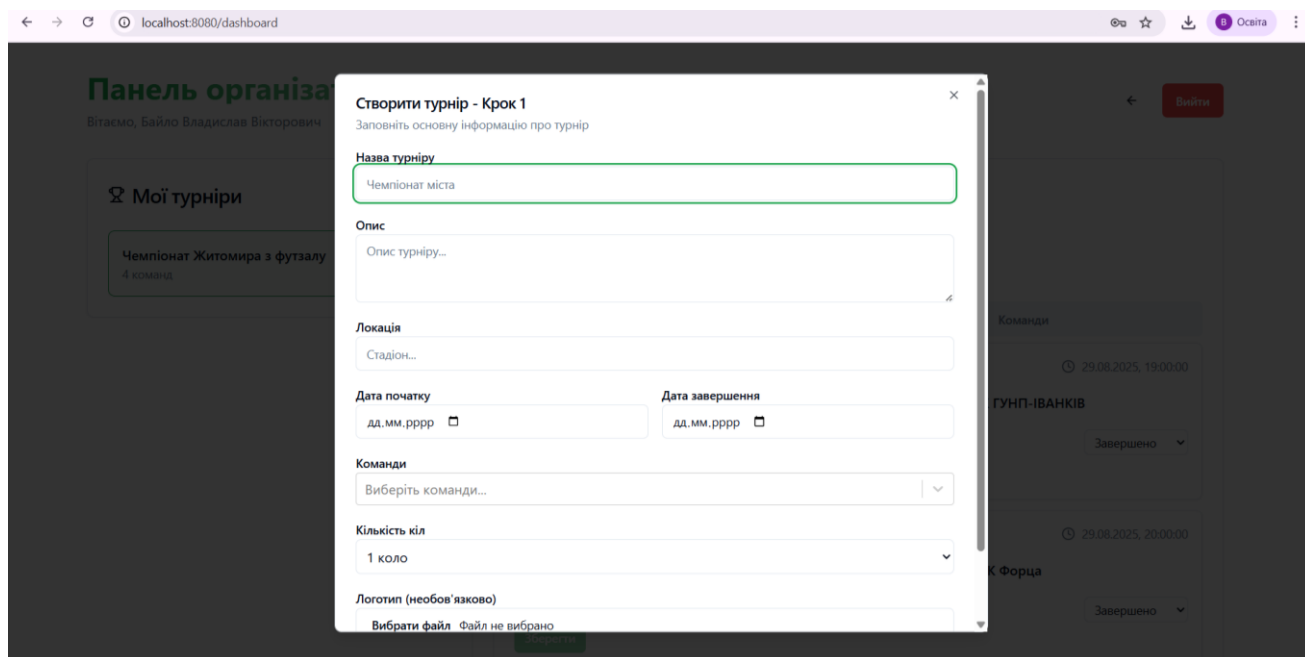


Рис. 3.6 – Налаштування турніру

Організатор може додати команди, змінити їх склад, редагувати інформацію про команду або видалити її. Також автоматично формується турнірна сітка залежно від обраного формату турніру.

Загальна публічна сторінка матчів (рис. 3.7) доступна для всіх користувачів, навіть без авторизації. Вона відображає календар матчів усіх турнірів, результати та поточний статус матчів.

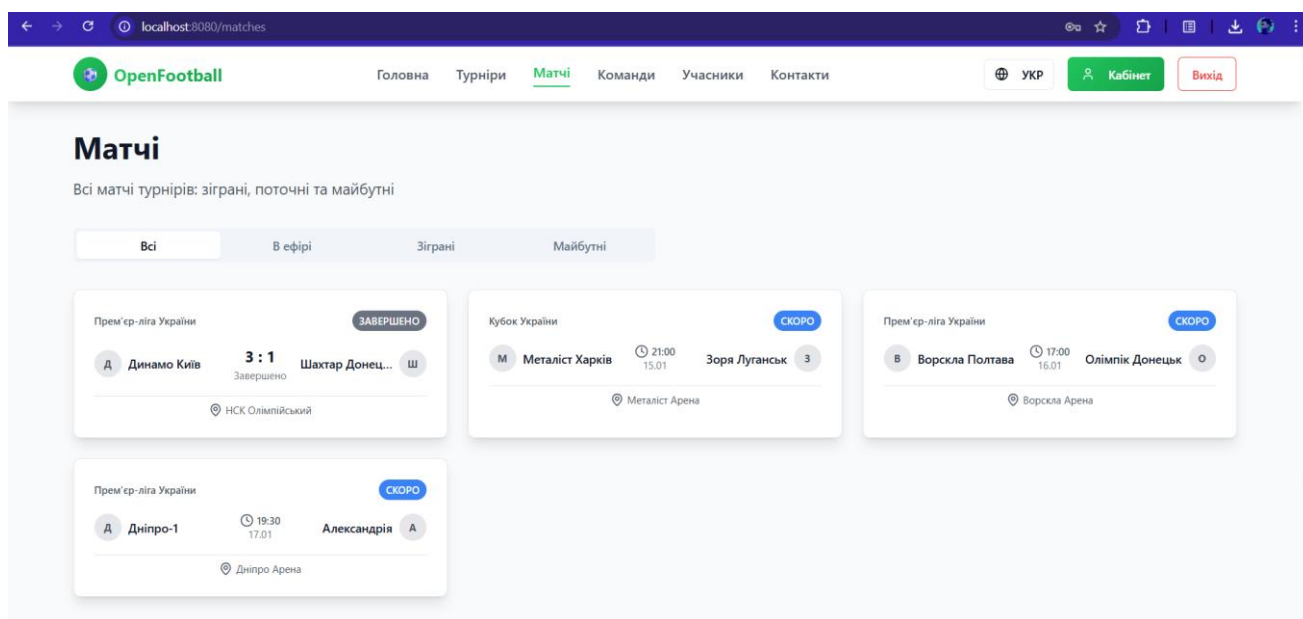


Рис. 3.7 – Публічна сторінка матчів

Таким чином, веб-додаток має чітку структуру з розподілом прав доступу, персональними кабінетами для кожної ролі та зручною навігацією для управління футбольними змаганнями.

3.3 Тестування роботи програмного комплексу

З метою перевірки коректності функціонування розробленого веб-додатку було проведено комплексне тестування, яке охоплює як основні бізнес-функції системи, так і допоміжні механізми, що забезпечують стабільність та безпеку роботи. Тестування проводилося у середовищі розробника з використанням локального сервера Node.js та бази даних MongoDB, а також у браузері Google Chrome із застосуванням інструментів розробника для відстеження запитів і відповідей сервера.

Насамперед було протестовано процес авторизації користувача. Перевірка виконувалася за декількома сценаріями: вхід з правильними обліковими даними,

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

вхід з невірним паролем, вхід з неіснуючим користувачем, спроба доступу до захищених маршрутів без токена та з простроченим токеном. У кожному випадку система поводитися відповідно до очікувань: при правильних даних користувач успішно отримував JWT-токен і був перенаправлений до особистого кабінету, а при помилкових або відсутніх даних система повертала інформативне повідомлення про помилку та блокувала доступ.

Далі було перевірено функціонал створення турніру організатором. У тестовому сценарії організатор заповнював усі необхідні поля форми (назву турніру, опис, локацію, дати проведення, формат) та ініціював створення. Перевірялися як стандартний сценарій з коректними даними, так і випадки введення некоректних або неповних даних. Система коректно обробляла запити, валідувала вхідну інформацію та зберігала новий турнір у базі даних. Після успішного створення турнір автоматично з'являвся у списку на сторінці організатора, що підтвердило синхронність роботи інтерфейсу з сервером.

Окрему увагу приділено тестуванню формування заявки на матч тренером. У процесі тестування перевірялися сценарії додавання гравців до заявки, видалення гравців перед підтвердженням, подання заявки з неповним складом та повторне редагування вже надісланої заявки. Система у всіх випадках обробляла запити без помилок, своєчасно оновлювала дані в інтерфейсі та забезпечувала відображення актуальної інформації у базі даних.

Також проведено тестування можливості видалення турніру організатором. Перевірялися як стандартний сценарій видалення, так і випадок, коли турнір мав пов'язані команди та матчі. У результаті підтверджено, що реалізоване каскадне видалення матчів спрацьовує коректно, а команди залишаються у базі, але втрачають прив'язку до видаленого турніру. Інтерфейс у реальному часі оновлював список турнірів після виконання операції.

Додатково було протестовано роботу пошуку та фільтрації користувачів, команд і турнірів, а також механізм пагінації. Перевірка включала тестові запити з різними комбінаціями фільтрів і ключових слів. Система стабільно повертала

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

релевантні результати, а швидкість обробки запитів залишалася на належному рівні навіть при великій кількості тестових записів.

Усі проведені тести підтвердили коректність реалізації основного функціоналу веб-додатку. Авторизація, управління турнірами, формування заявок, робота зі списками користувачів та команд виконуються стабільно і без критичних помилок. Зафіксовані незначні затримки при оновленні окремих даних можуть бути усунені у майбутніх оновленнях системи шляхом оптимізації запитів та використання додаткового кешування на стороні клієнта..

Висновки до третього розділу

У третьому розділі було детально розглянуто, як користувач взаємодіє з веб-додатком, як виглядає інтерфейс та як реалізована логіка обробки запитів під час використання основних функцій. Особливу увагу приділено зручності роботи з додатком – від моменту авторизації до керування турнірами, командами, складанням заявок на матчі та перегляду результатів.

Було проведено тестування ключових сценаріїв використання: авторизації, створення турніру, подання заявки на матч та видалення. Тестування підтвердило працездатність основного функціоналу, хоча у деяких випадках були зафіксовані незначні затримки при оновленні результатів, що потребує подальшої оптимізації.

Загалом, веб-додаток продемонстрував стабільну роботу, зрозумілу навігацію та позитивний користувацький досвід, що підтверджує доцільність його подальшого впровадження для організації футбольних змагань.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		42

ВИСНОВКИ

У межах цієї бакалаврської кваліфікаційної роботи було створено повноцінну вебсистему для організації та адміністрування футбольних змагань, яка поєднує зручний користувацький інтерфейс, стабільний серверний функціонал. Враховуючи зростаючу популярність локальних та аматорських футбольних турнірів, розроблена система має значну практичну цінність, оскільки забезпечує ефективну організацію змагань та зручність їх адміністрування.

На етапі аналізу були вивчені потреби цільових користувачів (адміністраторів, організаторів, тренерів, гравців, суддів та глядачів), розглянуто аналогічні програмні продукти та визначено оптимальний підхід до побудови архітектури веб-додатку. В результаті розробки побудовано чітку клієнт-серверну структуру, де фронтенд відповідає за інтерфейс і взаємодію з користувачем, серверна частина обробляє бізнес-логіку, а база даних MongoDB забезпечує надійне зберігання структурованої інформації про користувачів, команди, турніри та матчі.

Функціонал авторизації, створення турнірів, управління командами, подання заявок на матчі та перегляд результатів працюють як єдиний узгоджений механізм, що забезпечує комфортну взаємодію з системою. Проведене тестування підтвердило коректність основних процесів і продемонструвало стабільну роботу додатку, хоча виявлені незначні затримки при оновленні результатів в окремих випадках можуть бути усунуті у наступних версіях проєкту.

У результаті реалізовано не просто інформаційну систему, а зручний інструмент для повноцінного адміністрування футбольних змагань на локальному рівні. Веб-додаток здатний значно спростити процес організації турнірів, підвищити прозорість змагань та забезпечити доступність актуальної інформації для всіх учасників та глядачів. Реалізація цього проєкту демонструє, як сучасні цифрові технології можуть бути ефективно використані для розвитку спорту, підвищення якості організації змагань і створення комфортного цифрового середовища для всіх учасників процесу.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Enjore. Платформа для організації турнірів [Електронний ресурс]. – Режим доступу: <https://www.enjore.com/> (дата звернення: 01.08.2025).
2. Tournify. Вебзастосунок для турнірів [Електронний ресурс]. – Режим доступу: <https://www.tournifyapp.com/> (дата звернення: 01.08.2025).
3. SportsEngine. Офіційний сайт (рішення для турнірів) [Електронний ресурс]. – Режим доступу: <https://www.sportsengine.com/> (дата звернення: 02.08.2025).
4. Tournament Software. Офіційний сайт [Електронний ресурс]. – Режим доступу: <https://www.tournamentsoftware.com/> (дата звернення: 02.08.2025).
5. MERN Stack – огляд (MongoDB + Express + React + Node) [Електронний ресурс]. – Режим доступу: <https://www.mongodb.com/languages/mern-stack> (дата звернення: 03.08.2025).
6. MDN. API – Glossary [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Glossary/API> (дата звернення: 03.08.2025).
7. MDN. REST – Glossary [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Glossary/REST> (дата звернення: 04.08.2025).
8. JSON.org – Introducing JSON [Електронний ресурс]. – Режим доступу: <https://www.json.org/> (дата звернення: 04.08.2025).
9. Tailwind CSS. Документація [Електронний ресурс]. – Режим доступу: <https://tailwindcss.com/docs> (дата звернення: 05.08.2025).
10. Visual Studio Code. Офіційний сайт [Електронний ресурс]. – Режим доступу: <https://code.visualstudio.com/> (дата звернення: 05.08.2025).
11. React Documentation. Getting Started [Електронний ресурс]. – Режим доступу: <https://react.dev/learn> (дата звернення: 06.08.2025).
12. Docker: 101 Tutorial [Електронний ресурс]. – Режим доступу: <https://www.docker.com/101-tutorial> (дата звернення: 06.08.2025).
13. i18next. Документація [Електронний ресурс]. – Режим доступу: <https://www.i18next.com/> (дата звернення: 07.08.2025).

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат		44

14. Node.js. Офіційна документація [Електронний ресурс]. – Режим доступу: <https://nodejs.org/en/docs> (дата звернення: 07.08.2025).
15. Express.js. Документація [Електронний ресурс]. – Режим доступу: <https://expressjs.com/> (дата звернення: 08.08.2025).
16. MongoDB. Офіційна документація [Електронний ресурс]. – Режим доступу: <https://www.mongodb.com/docs/> (дата звернення: 08.08.2025).
17. JWT. Introduction [Електронний ресурс]. – Режим доступу: <https://jwt.io/introduction> (дата звернення: 09.08.2025).
18. MDN. JavaScript Guide [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide> (дата звернення: 09.08.2025).
19. TypeScript. Документація [Електронний ресурс]. – Режим доступу: <https://www.typescriptlang.org/docs/> (дата звернення: 10.08.2025).
20. React Router. Документація [Електронний ресурс]. – Режим доступу: <https://reactrouter.com/en/main> (дата звернення: 11.08.2025).
21. WebSockets API – MDN [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 12.08.2025).
22. PostgreSQL. Офіційна документація [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/> (дата звернення: 13.08.2025).
23. React Hook Form. Документація [Електронний ресурс]. – Режим доступу: <https://react-hook-form.com/get-started> (дата звернення: 13.08.2025).
24. bcrypt. GitHub [Електронний ресурс]. – Режим доступу: <https://github.com/kelektiv/node.bcrypt.js> (дата звернення: 14.08.2025).
25. TanStack React Query. Документація [Електронний ресурс]. – Режим доступу: <https://tanstack.com/query/v4/docs/react/overview> (дата звернення: 14.08.2025).
26. Duckett J. HTML and CSS: Design and Build Websites. – Wiley, 2011. – 490 с.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дат		

27. Flanagan D. JavaScript: The Definitive Guide. – 7th ed. – O'Reilly Media, 2020. – 706 с.

28. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2004. – 529 с.

29. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002. – 560 с.

30. Ткачук І. Особливості вибору NoSQL та SQL баз даних [Електронний ресурс] // DOU. – 13.01.2023. – Режим доступу: <https://dou.ua/lenta/articles/types-of-databases/> (дата звернення: 18.08.2025).

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат		46

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дат		

index.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const authRoutes = require('./routes/auth');
const adminRoutes = require('./routes/admin');
const organizerRoutes = require('./routes/organizer');
const userRoutes = require("./routes/users");
const publicRoutes = require('./routes/public');
const coachRoutes = require('./routes/coach');

const app = express();
app.use(cors());
app.use(express.json());

// API маршрути
app.use('/api/auth', authRoutes);
app.use('/api/admin', adminRoutes);
app.use('/api/organizer', organizerRoutes);
app.use("/api/users", userRoutes);
app.use('/api/public', publicRoutes);
app.use('/api/coach', coachRoutes);

const path = require('path');
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

app.use((req, res) => {
  res.status(404).json({ message: 'Route not found' });
});

const PORT = process.env.PORT || 5000;
mongoose.connect(process.env.MONGO_URI)
  .then(() => app.listen(PORT, () => console.log(`Server running on port ${PORT}`)))
  .catch(err => console.error('DB connection error:', err));
```

routes/auth.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const router = express.Router();

router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid credentials' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

    const token = jwt.sign(
      { id: user._id, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '2h' }
    );

    res.json({
      token,
      user: { _id: user._id, name: user.name, role: user.role }
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

module.exports = router;
```

admin.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const User = require('../models/User');
const Team = require('../models/Team');
const Tournament = require('../models/Tournament');
const auth = require('../middleware/auth');
const requireRole = require('../middleware/role');
const upload = require('../middleware/upload');

const router = express.Router();

router.use(auth, requireRole('admin'));

router.post('/users', upload.single('photo'), async (req, res) => {
  const { name, email, password, role, birthDate, team } = req.body;

  if (!['organizer', 'coach', 'player', 'referee'].includes(role)) {
    return res.status(400).json({ message: 'Invalid role' });
  }

  try {
    const exists = await User.findOne({ email });
    if (exists) {
      return res.status(400).json({ message: 'Email already in use' });
    }

    const hashed = await bcrypt.hash(password, 10);

    const newUser = new User({
      name,
      email,
      password: hashed,
      role,
      photo: req.file ? `/uploads/${req.file.filename}` : null,
      birthDate: birthDate || null,
      team: team || null
    });

    await newUser.save();

    if (team) {
      const foundTeam = await Team.findById(team);
      if (foundTeam) {
        if (role === 'player') {
          foundTeam.players.push(newUser._id);
        } else if (role === 'coach' && !foundTeam.coach) {
          foundTeam.coach = newUser._id;
        }
      }
    }
  }
});
```

```

    }
    await foundTeam.save();
  }
}

await newUser.populate('team');

res.status(201).json({
  _id: newUser._id,
  name: newUser.name,
  email: newUser.email,
  role: newUser.role,
  photo: newUser.photo,
  birthDate: newUser.birthDate,
  team: newUser.team
});

} catch (err) {
  console.error(err);
  res.status(500).json({ message: 'Server error' });
}
});

router.put('/users/:id/team', async (req, res) => {
  const { teamId } = req.body;

  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    if (user.team) {
      const oldTeam = await Team.findById(user.team);
      if (oldTeam) {
        if (user.role === 'player') {
          oldTeam.players = oldTeam.players.filter(
            (p) => p.toString() !== user._id.toString()
          );
        } else if (user.role === 'coach' && oldTeam.coach?.toString() ===
user._id.toString()) {
          oldTeam.coach = null;
        }
        await oldTeam.save();
      }
    }

    if (teamId) {
      const newTeam = await Team.findById(teamId);
      if (!newTeam) {
        return res.status(404).json({ message: 'New team not found' });
      }
    }
  }
});

```

```

    }

    if (user.role === 'player') {
      if (!newTeam.players.includes(user._id)) {
        newTeam.players.push(user._id);
      }
    } else if (user.role === 'coach') {
      if (newTeam.coach && newTeam.coach.toString() !== user._id.toString()) {
        return res.status(400).json({ message: 'This team already has a coach' });
      }
      newTeam.coach = user._id;
    }

    await newTeam.save();
    user.team = newTeam._id;
  } else {
    user.team = null;
  }

  await user.save();
  await user.populate('team');

  res.json({
    _id: user._id,
    name: user.name,
    email: user.email,
    role: user.role,
    team: user.team
  });

} catch (err) {
  console.error(err);
  res.status(500).json({ message: 'Server error' });
}
});

router.get("/users", async (req, res) => {
  try {
    const { role, search } = req.query;
    const query = {};

    if (role && role !== "all") query.role = role;

    if (search) {
      query.$or = [
        { name: new RegExp(search, "i") },
        { email: new RegExp(search, "i") },
      ];
    }

    const users = await User.find(query)

```

```

        .select("-password")
        .populate("team")
        .limit(search ? 20 : 0);

    res.json(users);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Server error" });
  }
});

router.put('/users/:id', upload.single('photo'), async (req, res) => {
  try {
    const { name, email, password, role, birthDate, team } = req.body;
    const updateData = { name, email, role };

    if (password && password.trim() !== '') {
      const salt = await bcrypt.genSalt(10);
      updateData.password = await bcrypt.hash(password, salt);
    }

    if (birthDate) updateData.birthDate = new Date(birthDate);

    updateData.team = team && team !== '' ? team : null;

    if (req.file) {
      updateData.photo = `/uploads/${req.file.filename}`;
    }

    const updatedUser = await User.findByIdAndUpdate(
      req.params.id,
      updateData,
      { new: true }
    ).populate('team');

    if (!updatedUser) {
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }

    if (role === 'coach') {
      await Team.updateMany(
        { coach: updatedUser._id },
        { $set: { coach: null } }
      );

      if (team) {
        await Team.findByIdAndUpdate(team, { coach: updatedUser._id });
      }
    }

    res.json(updatedUser);
  }
});

```

```

    } catch (err) {
      console.error(err);
      res.status(500).json({ message: 'Помилка сервера' });
    }
  });

router.delete('/users/:id', async (req, res) => {
  try {
    const deleted = await User.findByIdAndDelete(req.params.id);
    if (!deleted) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json({ message: 'User deleted' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

router.get('/teams', async (req, res) => {
  try {
    const teams = await Team.find()
      .populate('coach', 'name')
      .populate('players', 'name')
      .populate('tournament', 'name');
    res.json(teams);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

router.post('/teams', upload.single('logo'), async (req, res) => {
  const { name, coach, tournament, players } = req.body;

  try {
    if (!name || !coach) {
      return res.status(400).json({ message: 'Name and coach are required' });
    }

    const coachUser = await User.findById(coach);
    if (!coachUser || coachUser.role !== 'coach') {
      return res.status(400).json({ message: 'Invalid coach' });
    }

    if (coachUser.team) {
      const oldTeam = await Team.findById(coachUser.team);
      if (oldTeam) {
        oldTeam.coach = null;
        await oldTeam.save();
      }
    }
  }
});

```

```

    }

    const logoPath = req.file ? `/uploads/${req.file.filename}` : undefined;

    const newTeam = new Team({
      name,
      logo: logoPath,
      tournament: tournament || null,
      coach: coachUser._id,
      players: players ? (Array.isArray(players) ? players : [players]) : []
    });

    await newTeam.save();

    coachUser.team = newTeam._id;
    await coachUser.save();

    if (players) {
      const playerIds = Array.isArray(players) ? players : [players];
      await User.updateMany(
        { _id: { $in: playerIds }, role: 'player' },
        { $set: { team: newTeam._id } }
      );
    }

    await newTeam.populate('coach', 'name role photo');
    await newTeam.populate('players', 'name role photo');

    res.status(201).json(newTeam);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

router.put('/teams/:id', upload.single('logo'), async (req, res) => {
  const { name, coach, tournament, players } = req.body;

  try {
    const team = await Team.findById(req.params.id);
    if (!team) {
      return res.status(404).json({ message: 'Team not found' });
    }

    if (name) team.name = name;
    if (req.file) team.logo = `/uploads/${req.file.filename}`;
    if (tournament !== undefined) team.tournament = tournament || null;

    if (coach) {
      const newCoach = await User.findById(coach);
      if (!newCoach || newCoach.role !== 'coach') {

```

```

    return res.status(400).json({ message: 'Invalid coach' });
  }

  if (newCoach.team && newCoach.team.toString() !== team._id.toString()) {
    const oldTeam = await Team.findById(newCoach.team);
    if (oldTeam) {
      oldTeam.coach = null;
      await oldTeam.save();
    }
  }

  team.coach = newCoach._id;
  newCoach.team = team._id;
  await newCoach.save();
}

if (players) {
  const playerIds = Array.isArray(players) ? players : [players];
  team.players = [];

  for (const playerId of playerIds) {
    const player = await User.findById(playerId);
    if (!player || player.role !== 'player') continue;

    if (player.team && player.team.toString() !== team._id.toString()) {
      const oldTeam = await Team.findById(player.team);
      if (oldTeam) {
        oldTeam.players = oldTeam.players.filter(
          (p) => p.toString() !== player._id.toString()
        );
        await oldTeam.save();
      }
    }

    player.team = team._id;
    await player.save();
    team.players.push(player._id);
  }
}

await team.save();

await team.populate('coach', 'name role photo');
await team.populate('players', 'name role photo');

res.json(team);
} catch (err) {
  console.error(err);
  res.status(500).json({ message: 'Server error' });
}
});

```

```

router.delete('/teams/:id', async (req, res) => {
  try {
    const team = await Team.findById(req.params.id);
    if (!team) {
      return res.status(404).json({ message: 'Team not found' });
    }

    if (team.coach) {
      await User.findByIdAndUpdate(team.coach, { $set: { team: null } });
    }

    if (team.players.length > 0) {
      await User.updateMany(
        { _id: { $in: team.players } },
        { $set: { team: null } }
      );
    }

    await team.deleteOne();

    res.json({ message: 'Team deleted' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

router.get("/users/search", async (req, res) => {
  try {
    const { role, query } = req.query;
    const filter = {};
    if (role) filter.role = role;
    if (query) filter.name = new RegExp(query, "i");

    const users = await User.find(filter).limit(20);
    res.json(users);
  } catch (err) {
    res.status(500).json({ message: "Помилка пошуку" });
  }
});

router.get("/tournaments", async (req, res) => {
  try {
    const tournaments = await Tournament.find().select("name season location startDate");
    res.json(tournaments);
  } catch (err) {
    res.status(500).json({ message: "Помилка при отриманні турнірів" });
  }
});

```

```

router.get("/tournaments/:id", async (req, res) => {
  try {
    const tournament = await Tournament.findById(req.params.id)
      .populate("organizer", "name email")
      .populate("teams", "name");
    if (!tournament) return res.status(404).json({ message: "Турнір не знайдено" });
    res.json(tournament);
  } catch (err) {
    res.status(500).json({ message: "Помилка при отриманні турніру" });
  }
});

```

```

router.post("/tournaments", async (req, res) => {
  try {
    const newTournament = new Tournament({
      name: req.body.name,
      gender: req.body.gender,
      season: req.body.season,
      location: req.body.location,
      startDate: req.body.startDate,
      groupStage: req.body.groupStage,
      groupLegs: req.body.groupLegs,
      playoff: req.body.playoff,
      organizer: req.user._id,
    });

    await newTournament.save();
    res.status(201).json(newTournament);
  } catch (err) {
    res.status(400).json({ message: "Помилка при створенні турніру", error:
err.message });
  }
});

```

```

router.put("/tournaments/:id", async (req, res) => {
  try {
    const updated = await Tournament.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true }
    );
    if (!updated) return res.status(404).json({ message: "Турнір не знайдено" });
    res.json(updated);
  } catch (err) {
    res.status(400).json({ message: "Помилка при оновленні турніру", error:
err.message });
  }
});

```

```

router.delete("/tournaments/:id", async (req, res) => {

```

```
try {
  const deleted = await Tournament.findByIdAndDelete(req.params.id);
  if (!deleted) return res.status(404).json({ message: "Турнір не знайдено" });
  res.json({ message: "Турнір видалено" });
} catch (err) {
  res.status(500).json({ message: "Помилка при видаленні турніру" });
}
});

module.exports = router;
```

organizer.js

```
const express = require("express");
const auth = require("../middleware/auth");
const requireRole = require("../middleware/role");
const Tournament = require("../models/Tournament");
const Team = require("../models/Team");
const Match = require("../models/Match");

const router = express.Router();

router.use(auth, requireRole("organizer"));

router.post("/tournaments", async (req, res) => {
  try {
    const {
      name,
      gender,
      season,
      location,
      startDate,
      groupStage = false,
      groupLegs = 1,
      playoff = false
    } = req.body;

    const tour = new Tournament({
      name,
      gender,
      season,
      location,
      startDate,
      groupStage,
      groupLegs,
      playoff,
      organizer: req.user.id,
      teams: []
    });

    await tour.save();
    res.status(201).json(tour);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Error creating tournament" });
  }
});

router.get("/tournaments", async (req, res) => {
  try {
```

```

    const tours = await Tournament.find({ organizer: req.user.id });
    res.json(tours);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Error fetching tournaments" });
  }
});

router.get("/tournaments/:id", async (req, res) => {
  try {
    const tour = await Tournament.findOne({
      _id: req.params.id,
      organizer: req.user.id
    }).populate("teams", "name");
    if (!tour) return res.status(404).json({ message: "Tournament not found" });
    res.json(tour);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Error fetching tournament" });
  }
});

router.put("/tournaments/:id", async (req, res) => {
  try {
    const tour = await Tournament.findOneAndUpdate(
      { _id: req.params.id, organizer: req.user.id },
      req.body,
      { new: true }
    );
    if (!tour) return res.status(404).json({ message: "Tournament not found or forbidden" });
    res.json(tour);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Error updating tournament" });
  }
});

router.put("/tournaments/:id/add-team", async (req, res) => {
  try {
    const { teamId } = req.body;
    const tour = await Tournament.findOne({ _id: req.params.id, organizer: req.user.id });
    if (!tour) return res.status(404).json({ message: "Tournament not found" });

    if (!tour.teams.includes(teamId)) {
      tour.teams.push(teamId);
      await tour.save();
    }

    res.json(tour);
  }
});

```

```

    } catch (err) {
      console.error(err);
      res.status(500).json({ message: "Error adding team" });
    }
  });

router.post("/tournaments/:id/generate", async (req, res) => {
  try {
    const tour = await Tournament.findOne({ _id: req.params.id, organizer:
req.user.id });
    if (!tour) return res.status(404).json({ message: "Tournament not found" });

    await Match.deleteMany({ tournament: tour._id });

    const teams = tour.teams.map(t => t.toString());
    const matches = [];

    if (tour.groupStage) {
      for (let i = 0; i < teams.length; i++) {
        for (let j = i + 1; j < teams.length; j++) {
          matches.push({ teamA: teams[i], teamB: teams[j] });
          if (tour.groupLegs === 2) {
            matches.push({ teamA: teams[j], teamB: teams[i] });
          }
        }
      }
    }

    if (tour.playoff) {
      const shuffled = teams.sort(() => Math.random() - 0.5);
      for (let k = 0; k < shuffled.length; k += 2) {
        if (shuffled[k + 1]) {
          matches.push({ teamA: shuffled[k], teamB: shuffled[k + 1] });
        }
      }
    }

    const created = await Promise.all(matches.map(async m => {
      const date = new Date(tour.startDate);
      date.setDate(date.getDate() + Math.floor(Math.random() * 5));
      const match = new Match({
        tournament: tour._id,
        teamA: m.teamA,
        teamB: m.teamB,
        date,
        status: "scheduled",
        score: { teamA: null, teamB: null }
      });
      await match.save();
      return match;
    }));
  }
});

```

```

        res.json(created);
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: "Error generating bracket" });
    }
});

router.get("/tournaments/:id/matches", async (req, res) => {
    try {
        const tour = await Tournament.findOne({ _id: req.params.id, organizer:
req.user.id });
        if (!tour) return res.status(404).json({ message: "Tournament not found" });

        const matches = await Match.find({ tournament: tour._id })
            .populate("teamA", "name")
            .populate("teamB", "name")
            .sort("date");
        res.json(matches);
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: "Error fetching matches" });
    }
});

module.exports = router;

```

middleware/auth.js

```
const jwt = require('jsonwebtoken');

module.exports = function (req, res, next) {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'No token, authorization denied' });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: 'Token is not valid' });
  }
};
```