

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:

**«Вебплатформа для моніторингу футбольних
турнірів та автоматизації тренерської звітності»**

Виконав студент групи ПЗ-22-3
БІЛОШИЦЬКИЙ Максим Васильович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Житомир – 2026

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

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БІЛОШИЦЬКИЙ Максим Васильович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Вебплатформа для моніторингу футбольних турнірів та автоматизації тренерської звітності»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: Об'єктом дослідження виступає процес інформаційної взаємодії користувачів із масивами спортивної статистики, що охоплює методи агрегації даних, їх динамічну візуалізацію та персоналізацію контенту в межах вебплатформи. Предметом дослідження є архітектурні підходи до проектування мультиспортивних систем, алгоритми обробки показників матчів у потоковому режимі та програмний стек, який забезпечує безперебійне функціонування аналітичного сервісу.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	01.03.2026р.	01.03.2026р
2	Власенко О.В.	01.03.2026р	01.03.2026р
3	Власенко О.В.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Вебплатформа для моніторингу футбольних турнірів та автоматизації тренерської звітності» містить 81 сторінок основного тексту. Ілюстративний матеріал представлений 39 рисунками та 19 таблицями. Бібліографічний перелік складається з 33 найменувань, а додатки займають 8 сторінок. Загальний обсяг документа становить 89 сторінок.

Мета дослідження полягає у проєктуванні та програмній реалізації комплексної інформаційно-аналітичної вебплатформи «Footballium». Основне завдання системи є автоматизація організаційних процесів футбольних чемпіонатів, збір спортивної статистики, моніторинг результатів поєдинків і взаємодія між вболівальниками, тренерами, суддями та адміністраторами у режимі реального часу.

Архітектурні рішення. Клієнтська та серверна частини платформи побудовані з використанням сучасного мета-фреймворку Next.js та мови строгої типізації TypeScript. Для управління масивами спортивних даних інтегровано оптимізовану реляційну базу PostgreSQL у хмарному кластері Neon. Реалізовано застосування підсистеми Live-моніторингу на базі протоколу WebSockets.

Практичне значення. Створено повнофункціональний програмний продукт, орієнтований на сферу цифрового спортивного менеджменту. Завдяки впровадженню гнучкої рольової моделі та сучасних хмарних технологій розгортання досягнуто високого рівня автоматизації управління турнірами, зручного адміністрування клубних ростерів та блискавичного оновлення Live-статистики, що суттєво оптимізує роботу організаторів змагань та покращує користувацький досвід аудиторії.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, СПОРТИВНА АНАЛІТИКА, ФУТБОЛЬНИЙ ТУРНІР.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Білошицький М. В			Вебплатформа для моніторингу футбольних турнірів та автоматизації тренерської звітності	Літ.	Арк.	Аркушів
Перевір.							3	83
Керівник		Власенко О.В.				Житомирська політехніка, група ІПЗ-22-3		
Рецензент		Левківський В. Л.						
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The explanatory note to the qualification work on the topic "Web platform for monitoring football tournaments and automating coaching reports" contains 81 pages of the main text. The illustrative material is represented by 39 figures and 19 tables. The bibliographic list consists of 33 items, and the appendices occupy 8 pages. The total volume of the document is 89 pages.

The purpose of the study is the design and software implementation of a comprehensive information and analytical web platform "Footballium". The main objective of the system is the automation of the organizational processes of football championships, collection of sports statistics, monitoring of match results, and interaction between fans, coaches, referees, and administrators in real time.

Architectural solutions. The client and server parts of the platform are built using the modern meta-framework Next.js and the strict typing language TypeScript. To manage sports data arrays, an optimized relational database PostgreSQL is integrated into the Neon cloud cluster. The application of a Live-monitoring subsystem based on the WebSockets protocol has been implemented.

Practical significance. A fully functional software product oriented towards the field of digital sports management has been created. Thanks to the implementation of a flexible role-based access control model and modern cloud deployment technologies, a high level of automation in tournament management, convenient administration of club rosters, and lightning-fast updating of Live-statistics has been achieved. This significantly optimizes the workflow of competition organizers and improves the audience's user experience.

KEYWORDS: WEB PLATFORM, CLIENT-SERVER ARCHITECTURE, SPORTS ANALYTICS, FOOTBALL TOURNAMENT.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

Зміст

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	8
РОЗДІЛ 1. ПРЕДМЕТНА ОБЛАСТЬ ТА ПОСТАНОВКА ЗАДАЧІ	10
1.1 Постановка задачі та огляд існуючих систем-аналогів.....	10
1.2. Вибір архітектури веб-орієнтованої системи	16
1.3 Обґрунтування вибору інструментальних засобів.....	21
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ	27
2.1 Моделювання варіантів використання та поведінки системи.....	27
2.2 Проєктування логічної структури бази даних.....	30
2.3 Проєктування компонентної структури та взаємодії модулів.....	35
2.4 Проєктування та реалізація алгоритмів роботи системи	51
Висновки до другого розділу	56
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОЄКТУ.....	58
3.1 Вибір програмного середовища та розгортання системи	58
3.2 Опис користувацького інтерфейсу та керівництво користувача	62
3.3 Тестування роботи програмного продукту	71
Висновки до третього розділу.....	71
ВИСНОВКИ.....	76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	78
ДОДАТКИ.....	81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – Application Programming Interface (прикладний програмний інтерфейс)

CDN – Content Delivery Network (мережа доставки контенту)

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція та безперервне розгортання)

CSR – Client-Side Rendering (рендеринг на стороні клієнта) CSS – Cascading Style Sheets (каскадні таблиці стилів)

DDL – Data Definition Language (мова опису даних)

DOM – Document Object Model (об'єктна модель документа) E2E – End-to-End (наскрізне тестування)

FCP – First Contentful Paint (час до першого відмальовування контенту)

FK – Foreign Key (зовнішній ключ)

HTML – HyperText Markup Language (мова гіпертекстової розмітки)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

ISR – Incremental Static Regeneration (інкрементальна статична регенерація)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

JWT – JSON Web Token (вебтокен формату JSON для безпечної авторизації)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

PK – Primary Key (первинний ключ)

RBAC – Role-Based Access Control (управління доступом на основі ролей)

REST – Representational State Transfer (архітектурний стиль передачі репрезентативного стану)

SEO – Search Engine Optimization (пошукова оптимізація)

SPA – Single Page Application (односторінковий вебзастосунок)

SQL – Structured Query Language (мова структурованих запитів) SSG – Static Site Generation (статична генерація вебсторінок)

SSR – Server-Side Rendering (рендеринг на стороні сервера)

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

SWR – Stale-While-Revalidate (стратегія кешування даних)

UI – User Interface (користувацький інтерфейс)

UML – Unified Modeling Language (уніфікована мова моделювання)

UX – User Experience (користувацький досвід)

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується тотальною діджиталізацією всіх сфер суспільного життя, і спортивна індустрія, зокрема її аматорський та регіональний сегменти, не є винятком [19, 20]. Традиційні підходи до управління футбольними турнірами, що базуються на паперовому документообігу, розрізнені електронних таблицях та ручному зведенні статистичних показників, демонструють свою критичну неефективність в умовах стрімкого зростання обсягів даних. Це призводить до значних часових затримок в оновленні інформації, підвищує ризик втрати даних через людський фактор та суттєво ускладнює процес комунікації між організаторами, тренерським складом, арбітрами та вболівальниками [21]. В умовах децентралізації спортивних заходів та необхідності оперативної взаємодії учасників, розробка спеціалізованої вебплатформи стає не просто питанням зручності, а критичною вимогою для забезпечення прозорості, об'єктивності та високого організаційного рівня змагань. Інтеграція алгоритмів автоматичної генерації турнірних сіток, модулів Live-моніторингу та систем електронної тренерської і суддівської звітності в єдине інформаційне середовище дозволяє докорінно трансформувати та оптимізувати управлінські процеси. Таким чином, розробка комплексного програмного продукту, що об'єднує всі ці аспекти на базі сучасних вебтехнологій, є своєчасним, науково обґрунтованим та надзвичайно актуальним завданням [22].

Мета дослідження полягає у проєктуванні та програмній реалізації вебплатформи для комплексного моніторингу футбольних турнірів, управління розкладом матчів та автоматизації подачі тренерської звітності.

Завдання дослідження:

- Провести системний аналіз предметної області та дослідити існуючі програмні рішення на ринку спортивного менеджменту для виявлення їхніх ключових недоліків.
- Сформулювати комплексні функціональні та нефункціональні вимоги до інформаційної системи, спираючись на гнучку рольову модель управління доступом.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

- Обґрунтувати вибір оптимальних архітектурних рішень та сучасного технологічного стека (зокрема Next.js, React та TypeScript) для розробки клієнтської і серверної частин вебзастосунку [1, 3, 4].
- Спроекувати логічну та фізичну структуру реляційної бази даних [5], здатну забезпечити надійне збереження і цілісність складної турнірної статистики.
- Розробити алгоритми автоматизованої генерації турнірних сіток і обробки ігрових подій у режимі реального часу.
- Здійснити безпосередню програмну реалізацію всіх модулів вебплатформи з використанням компонентного підходу.
- Провести комплексне багаторівневе тестування розробленого програмного продукту для перевірки та підтвердження його відмовостійкості, безпеки і повної відповідності початковим технічним вимогам.

Об'єктом дослідження є процес інформаційного супроводу та управління футбольними турнірами.

Предметом дослідження є архітектурні рішення, алгоритми та програмні засоби розробки платформи для моніторингу спортивних подій та автоматизації документообігу.

Методи дослідження. Під час виконання роботи використовувалися методи системного аналізу (для формування вимог до вебплатформи), методи об'єктно-орієнтованого проєктування (для розробки архітектури бази даних та компонентів системи), а також методи тестування програмного забезпечення (для верифікації працездатності розроблених модулів).

Практичне значення одержаних результатів. Створений програмний продукт «Footballium» являє собою готову до впровадження систему, яка забезпечує автоматизоване жеребкування, зручний моніторинг перебігу матчів та стандартизацію тренерської звітності. Впровадження платформи дозволить організаторам турнірів мінімізувати вплив людського фактора, а тренерам ефективніше аналізувати результати своїх команд.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. ПРЕДМЕТНА ОБЛАСТЬ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі та огляд існуючих систем-аналогів

Дослідження в межах даної кваліфікаційної роботи охоплює галузь ІТ-рішень для спорту, з акцентом на розробку вебсистем, призначених для адміністрування футбольних змагань і цифровізації звітної документації. В умовах загальної діджиталізації оперативне отримання достовірних статистичних відомостей, календарів матчів та доступ до електронного документообігу стають критично важливими для організаторів, тренерського складу, суддів і фанатів. Водночас успішність проведення турніру безпосередньо залежить від рівня впроваджених технологічних інструментів.

Застарілі методи координації регіональних і аматорських ліг відзначаються низькою ефективністю та надмірними витратами робочого часу. Основні перешкоди полягають у неавтоматизованому складанні календарів, проблемах із комунікацією та використанні паперових форматів для післяматчевих рапортів. Значним недоліком є також несвоєчасне оновлення даних: через відсутність електронних суддівських протоколів результати часто публікуються із запізненням на добу після завершення події. Усвідомлення цих викликів зумовлює необхідність проєктування інтелектуальних інформаційних середовищ. У таких комплексах жеребкування та агрегація статистики відбуваються автономно, а кожна категорія користувачів отримує ергономічний простір для виконання своїх завдань.

Сучасні цифрові вебпродукти вимагають наявності гнучкої масштабованої інфраструктури та здатності витримувати значні пікові навантаження під час проведення масових спортивних заходів. Для забезпечення бездоганного клієнтського досвіду та високої швидкості рендерингу сьогодні найчастіше використовують передовий мета-фреймворк Next.js у тісній синергії з суворо типізованою мовою програмування TypeScript [1, 27]. Цей інноваційний набір інструментів дозволяє розробляти відмовостійкі архітектурні рішення з підтримкою рендерингу на стороні сервера (SSR), що позитивно впливає на пошукову оптимізацію (SEO). Візуальна складова клієнтського інтерфейсу базується на компонентному підході із застосуванням утилітарного фреймворку Tailwind CSS [7] для забезпечення швидкої та

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

семантично правильної адаптивної верстки, що є оптимальним інженерним рішенням для коректного відображення динамічних турнірних таблиць і складних аналітичних дашбордів на екранах будь-якого розміру

Бізнес-логіка розмежування прав та управління мережевими запитами реалізується на серверному рівні. Для надійного збереження багатовимірних спортивних сутностей (турнірів, команд, матчів, гравців, звітів) найкраще підходять реляційні бази даних. Програмна взаємодія зі сховищем забезпечується сучасним інструментом Prisma ORM [6]. Безперебійність та коректність роботи підтверджується шляхом впровадження тестування за допомогою середовища Jest та екосистеми React Testing Library [16, 17]. Фундаментальним архітектурним кроком у розробці системи є впровадження механізму управління доступом на базі ролей (Role-Based Access Control). Цей патерн безпеки дає змогу алгоритмічно розділити логіку застосунку для чотирьох цільових груп: глобального Адміністратора, Тренера клубу, Судді та звичайного Вболівальника. Завдяки додатковій інтеграції протоколу WebSockets [11] формується екосистема реального часу: дії арбітра на футбольному полі миттєво і без примусового перезавантаження сторінки синхронізуються з публічним екраном вболівальників. Безперебійність та коректність роботи такого складного функціоналу безапеляційно підтверджується шляхом впровадження стратегії багаторівневого модульного та інтеграційного тестування за допомогою середовища Jest та екосистеми React Testing Library. Аналіз технологічних потреб доводить повну обґрунтованість вибору сучасного стека JavaScript, TypeScript, який стає міцною основою для побудови надійного автоматизованого спортивного сервісу. На вітчизняному ринку програмного забезпечення для аматорських турнірів домінують класичні вебсайти місцевих спортивних об'єднань. Одним із таких прикладів є ресурс Асоціації футзалу Житомирської області (futsal.zt.ua), діяльність якого зосереджена на публікації зведених таблиць та загальної статистики команд (рис. 1.1).

До сильних сторін цього порталу належить глибока структуризація локальних ліг та детальний облік індивідуальних досягнень бомбардирів. Платформа

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

містить профілі колективів та заявки гравців. Однак детальний розгляд виявляє брак інструментів автоматизації процесів.

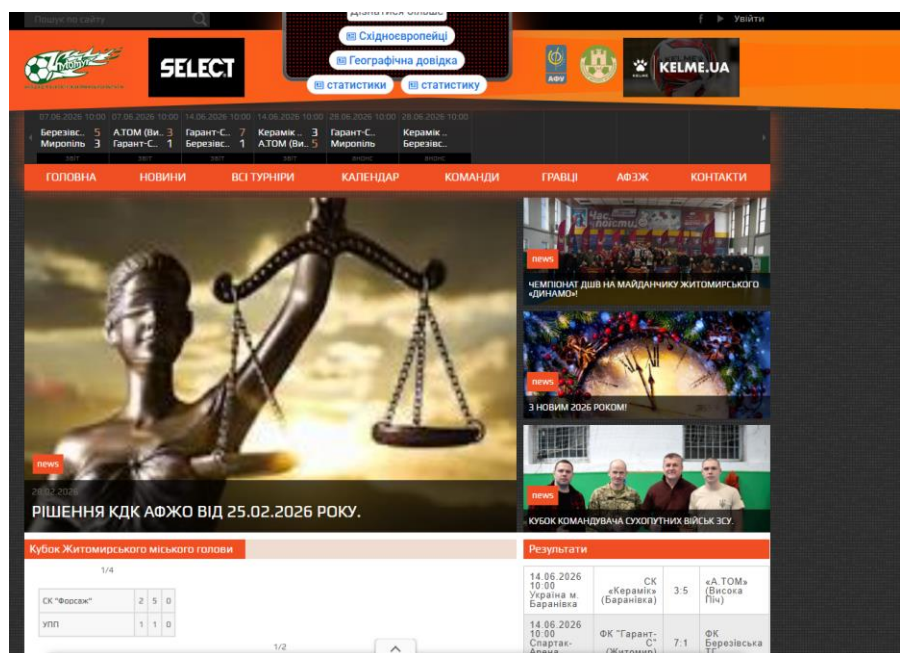


Рис. 1.1 – Інтерфейс стартової сторінки порталу futsal.zt.ua

Головний екран платформи базується на традиційній новинній стрічці та таблиці вищого дивізіону. Навігаційне меню дозволяє перемикатися між лігами та переглядати регламенти. Уся логіка подачі матеріалу спирається на стандартну систему управління контентом (CMS).

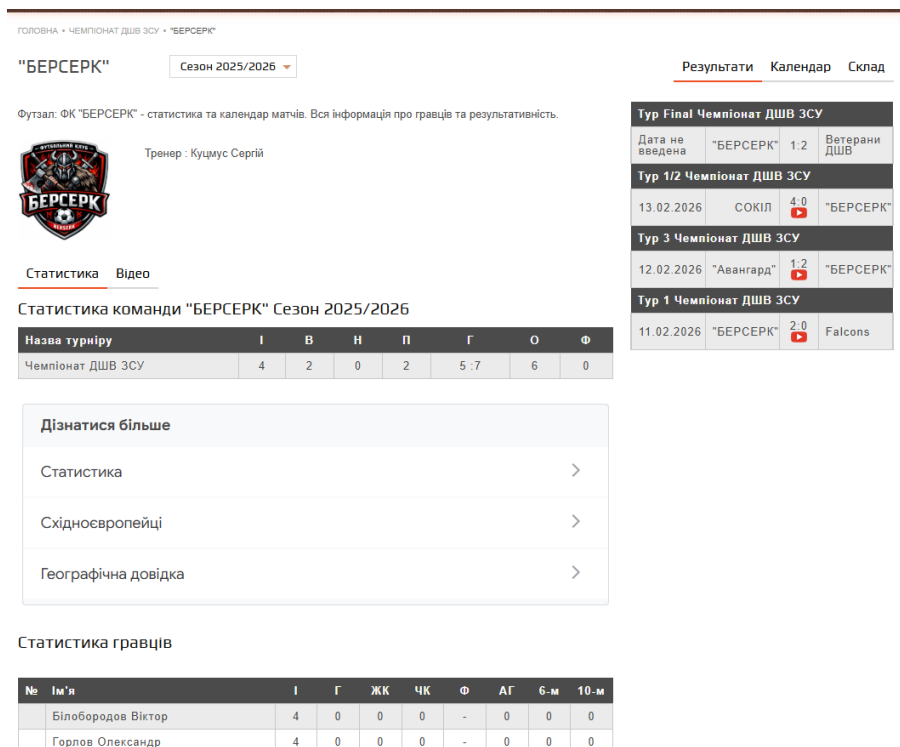


Рис. 1.2 – Відображення командної статистики в системі futsal.zt.ua

Сторінка клубу надає базовий перелік спортсменів та їхні статистичні показники (рис. 1.2). Попри те, що система добре справляється зі зберіганням архівних даних, для тренерів та адміністраторів недоліком залишається ручне внесення інформації. Сервіс не має особистих кабінетів для керівників команд та не підтримує подачу цифрових рапортів, функціонуючи лише як статична вітрина даних.

Наступним аналогом є офіційний вебсайт Житомирської обласної асоціації футболу (zhoff.org.ua), орієнтований на висвітлення регіональних змагань з класичного футболу (рис. 1.3). На цьому ресурсі зібрано значний масив рішень контрольно-дисциплінарних комітетів та регламентної документації.

Попри офіційний статус, цей сайт демонструє очевидні технологічні прогалини. Система використовує застарілі архітектурні підходи: розклади ігор та результати часто публікуються як завантажені зображення або PDF-документи. Це блокує можливість машинної обробки даних та робить навігацію з мобільних телефонів вкрай некомфортною.



Рис. 1.3 – Головний екран ресурсу zhoff.org.ua

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

На стартовій сторінці пріоритет віддано текстовим новинам асоціації, тоді як інтерактивні спортивні показники практично відсутні.

Головним мінусом ресурсу є відсутність динамічності. Користувачі не можуть відстежувати зміни в турнірному становищі під час матчів. Застаріла монолітна інфраструктура унеможливлює розгортання модулів Live-моніторингу та підключення електронних суддівських протоколів.

Третій приклад для порівняння — портал Запорізької міської асоціації футболу (football.zp.ua). Застосунок вирізняється якіснішою структурою та гармонійним поєднанням статистичних показників із мультимедійним контентом (рис. 1.4).

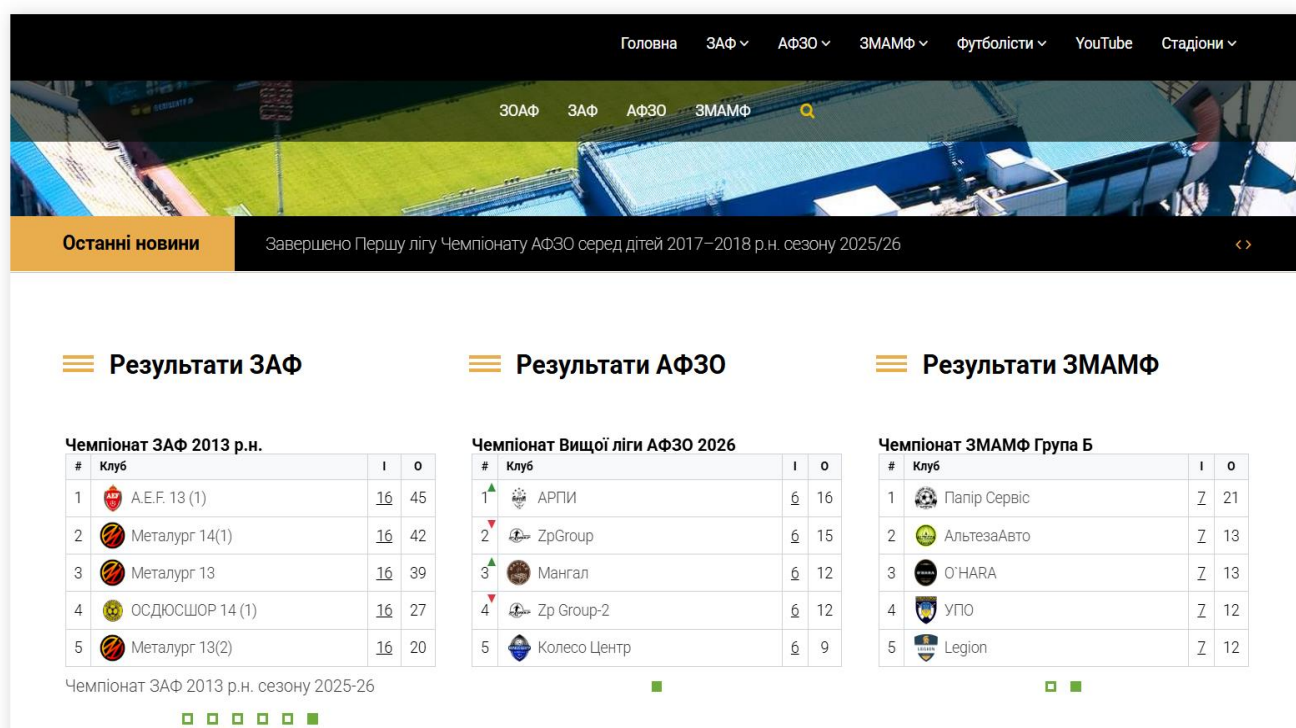


Рис. 1.4 – Головний екран та календар ігор сайту football.zp.ua

У цьому застосунку реалізовано чіткий поділ на ліги та кубкові турніри. Сторінка кожного матчу містить дані про склади учасників та призначення арбітрів. Дизайн платформи забезпечує непоганий баланс між обсягом інформації та візуальним сприйняттям. Попри краще візуальне оформлення, сайт має жорсткі функціональні ліміти. Всі процеси наповнення покладені на адміністраторів, оскільки система не підтримує делегування прав. Тренери не можуть самостійно генерувати заявки на гру або залишати післяматчеві оцінки у формалізованому електронному

вигляді. Для систематизації функціональних вимог до створюваної платформи результати порівняння існуючих рішень зведено у таблицю 1.1.

Таблиця 1.1

Порівняння функціоналу систем адміністрування турнірів

Критерії порівняння	futsal.zt.ua	zhoff.org.ua	football.zp.ua	Проектowana система
Алгоритмічна побудова сітки змагань	Ні	Ні	Ні	Так
Розмежування доступу (Арбітр, Тренер)	Ні	Ні	Ні	Так
Цифровий Live-протокол арбітра	Ні	Ні	Ні	Так
Генерація електронних звітів тренера	Ні	Ні	Ні	Так
Формат вебзастосунку	Стандартний сайт	Стандартний сайт	Стандартний сайт	Модульний SPA/SSR
Базова архітектура	CMS-моноліт	CMS-моноліт	CMS-моноліт	Клієнт-сервер (Next.js)
Автоматизоване E2E-тестування	Ні	Ні	Ні	Так

Підсумовуючи огляд аналогів, можна констатувати, що існуючі на регіональному рівні рішення мають ряд критичних обмежень. Більшість із них є звичайними інформаційними сайтами, де оновлення даних виконується адміністраторами після завершення турнірного дня. Жоден із розглянутих порталів не пропонує інтерактивного інструментарію для суддів на полі та не автоматизує документообіг тренерського штабу.

Розроблювана вебплатформа «Footballium» покликана комплексно вирішити ці проблеми. Завдяки імплементації сучасної клієнт-серверної архітектури, розмежуванню прав доступу, алгоритмам автономної генерації сіток та Live-синхронізації рахунків, ця система перетворюється на високотехнологічний та зручний засіб для повноцінного управління футбольними турнірами.

1.2. Вибір архітектури веб-орієнтованої системи

Загальна ефективність, відмовостійкість та потенціал до масштабування розроблюваної платформи «Footballium» безпосередньо залежать від фундаментальних архітектурних рішень, прийнятих на етапі системного проектування. Специфіка спортивного аналітичного застосунку полягає у необхідності паралельної обробки статичного контенту (новини, історія турнірів) та динамічних даних у реальному часі (зміна рахунку матчів, оновлення Live-статистики). Проектування інформаційних систем такого типу вимагає розв'язання складної інженерної дилеми, що полягає в одночасному забезпеченні низького часу відгуку для тисяч пасивних користувачів (вболівальників) та високої консистентності даних під час інтенсивного введення інформації обмеженим колом модераторів, тренерів та арбітрів. Традиційні монолітні архітектури або застарілі клієнт-серверні патерни не здатні забезпечити необхідну гнучкість, оскільки збільшення навантаження на один із модулів неминуче призводить до деградації продуктивності всього вебзастосунку.

Для задоволення цих вимог в якості базового патерну було обрано архітектуру Full-Stack у форматі монорепозиторію (Monorepo), що повністю відповідає принципам чистої архітектури [24]. Цей підхід забезпечує консолідацію клієнтської та серверної частин програмного коду в єдиному середовищі, що спрощує маршрутизацію, гарантує наскрізну типізацію даних між фронтендом та бекендом, а також мінімізує витрати часу на розгортання проєкту. Впровадження концепції чистої архітектури дозволяє ізолювати центральну бізнес-логіку системи від зовнішніх чинників, таких як конкретна реалізація бази даних або інтерфейс користувача. Завдяки цьому логічні правила розрахунку змагальних балів, алгоритми жеребкування чи правила валідації матчів залишаються інваріантними до технологічних змін у шарах інфраструктури. Використання монорепозиторію у даному контексті стає інструментом фізичного розділення коду на перевикористовувані пакети, що дозволяє фронтенд-компонентам безпосередньо імпортувати інтерфейси та типи даних, задекларовані на рівні бази даних, повністю нівелюючи ризик появи помилок десеріалізації при мережевому обміні.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

Фундаментом архітектурної моделі виступає сучасна парадигма App Router, яка передбачає строгий логічний поділ інтерфейсу на два типи компонентів, а саме на серверні та клієнтські компоненти. Серверні компоненти (React Server Components, RSC) відповідають за рендеринг важкого контенту безпосередньо на сервері до його відправки в браузер. Це суттєво зменшує обсяг JavaScript-коду, що завантажується клієнтом, гарантуючи високу швидкість початкового відображення (First Contentful Paint, FCP) та ефективну пошукову оптимізацію (SEO). На відміну від класичного Server-Side Rendering (SSR), де вся сторінка повинна бути повністю регідрована на клієнті, серверні компоненти генерують спеціалізований серіалізований потік даних, який інтегрується у дерево елементів без втрати стану вже існуючих інтерактивних елементів у браузері. Це дозволяє здійснювати прямі запити до бази даних без створення додаткових REST API ендпоінтів для первинного відображення, що підвищує загальну безпеку системи, оскільки чутливі алгоритми та ключі доступу залишаються всередині захищеного серверного середовища виконання.

З іншого боку, клієнтські компоненти застосовуються виключно в тих модулях, де необхідна безперервна інтерактивність, до яких належать панель арбітра для введення результатів поєдинків, динамічні форми тренерської звітності або інтерактивні графіки аналітичного дашборда. Вони завантажуються в браузер користувача у вигляді гідрованих JavaScript-модулів і забезпечують реактивний відгук на інтерфейсні події. Глибокий аналіз парадигми компонентного поділу доводить, що такий гібридний підхід дозволяє мінімізувати час до першої інтерактивності (Time to Interactive, TTI), що є критичним показником для користувачів мобільних пристроїв. У традиційних застосунках клієнт змушений очікувати завантаження всього JavaScript-коду ядра системи, після чого браузер ініціює каскад асинхронних запитів для отримання метаданих. У проєктованій системі «Footballium» загальна структура сторінки, навігаційні панелі, архівні статистичні таблиці та новинні блоки рендеряться на сервері як RSC і миттєво доставляються у вигляді готової розмітки, а клієнтські компоненти вбудовуються в це серверне дерево як ізольовані острівці інтерактивності. Наприклад, у модулі перегляду живого матчу вся статична

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

інформація є серверною, а лічильник часу та таймлайн подій — клієнтськими, що мінімізує обчислювальне навантаження на клієнтську сторону.

Важливою перевагою обраної інфраструктури Next.js App Router є впровадження концепції вкладених макетів (Nested Layouts), що забезпечує високу модульність кодової бази. Файлова система маршрутизації дозволяє чітко розмежувати публічний сегмент застосунку та закриті кабінети користувачів із різними ролями. Кожен підкаталог у структурі проєкту може містити власні файли макетів, сторінок, шаблонів завантаження (loading.tsx) та обробки виняткових ситуацій (error.tsx). Це дозволяє реалізувати оптимізоване потокове передавання даних (Streaming) за допомогою технології HTML Chunked Transfer Encoding. Сервер починає надсилати перші байти розмітки інтерфейсу користувачу ще до того, як завершиться виконання довготривалих запитів до бази даних, що візуально сприймається як миттєве завантаження системи. Поки виконується обчислювально місткий алгоритм формування турнірної сітки або агрегації звітів, Next.js тимчасово рендерить заздалегідь підготовлений каркасний компонент (Skeleton), замінюючи його фінальним інтерфейсом одразу після готовності даних від ORM.

Інформаційна взаємодія в системі побудована за багаторівневою моделлю, де клієнтська частина формує запити, які через систему API-маршрутів (Next.js API Routes) надходять до серверного рівня. На цьому етапі відбувається суворі валідація вхідних даних для запобігання ін'єкціям та логічним помилкам. Процес обробки кожного HTTP-запиту підпорядкований концепції конвеєра проміжного програмного забезпечення. На першому кроці системне програмне забезпечення перехоплює запит для автентифікації сесії користувача та перевірки його рольових привілеїв у межах моделі управління доступом на основі ролей (RBAC). Якщо запит надходить до захищеного сегмента системи, наприклад, від імені тренера або судді, прошарок безпеки аналізує криптографічний JWT-токен. У разі успішної верифікації запит передається безпосередньо до контролера маршруту, де вхідний корисний payload піддається санітизації та структурній перевірці за допомогою декларативних схем валідації Zod. Це гарантує, що бізнес-контролери оперують виключно

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

валідними, типізованими та повністю очищеними від потенційно шкідливого вмісту даними.

Після успішної перевірки бізнес-логіка вебплатформи звертається до рівня абстракції даних (ORM) для виконання транзакцій у реляційній базі даних. Шар доступу до даних інкапсулює у собі складність формування SQL-запитів, оптимізацію індексів та контроль пулу з'єднань. У контексті архітектури Next.js використання сучасного інструменту Prisma ORM дозволяє створювати надійні транзакційні межі безпосередньо у коді функцій. Під час генерації турнірної сітки або перерахунку standings-таблиць система відкриває атомарну транзакцію на рівні бази даних PostgreSQL. Це означає, що створення десятків об'єктів матчів, прив'язка суддівських бригад та ініціалізація початкових балів команд відбуваються за принципом єдиної неподільної сесії. Якщо під час запису одного з об'єктів відбудеться апаратний збій або мережевий тайм-аут, СУБД автоматично виконає rollback, повертаючи сховище до початкового консистентного стану, що повністю виключає появу логічних колізій або частково сформованих змагальних етапів у системі.

Окремим критично важливим архітектурним вузлом є підсистема обробки подій у реальному часі. Для уникнення постійного опитування сервера, що створює надмірне навантаження, імплементовано подієво-орієнтовану модель на базі протоколу WebSockets [11]. Традиційний HTTP-протокол через свою stateless-природу вимагає відкриття нового TCP-з'єднання для кожної інформаційної операції, що призводить до значних накладних витрат по трафіку та обчислювальній потужності сервера при високій частоті запитів від тисяч активних фанатів. WebSocket, натомість, після первинного рукостискання встановлює персистентне дуплексне з'єднання. При фіксації події на полі, наприклад, коли забито гол або винесено картку, сервісний шар Next.js не лише оновлює запис у реляційній базі даних, а й миттєво публікує подію у виділений WebSocket-канал через проміжного брокера повідомлень Pusher.

Робочий процес цієї підсистеми базується на патерні архітектури Pub/Sub (Publish/Subscribe), який мінімізує зв'язність між компонентами програми. Серверний контролер виступає виключно як видавець подій, не володіючи інформацією

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

про точну кількість, географічне розташування чи апаратні характеристики підключених клієнтів. Брокер повідомлень бере на себе завдання мультиплексування потоків та утримання тисяч активних сокет-з'єднань, що знімає критичне навантаження з основного сервера безсерверних обчислень Vercel. Коли арбітр натискає кнопку фіксації взяття воріт, серіалізований об'єкт події за лічені мілісекунди транслюється по всьому дереву підписників, викликаючи точкове реактивне перемальовування інтерфейсу користувачів, що створює ефект повної присутності та безперервності ігрового процесу.

Таким чином, інтеграція Clean Architecture, модульного монорепозиторію, гібридного рендерингу Next.js App Router та подієво-орієнтованих протоколів реального часу формує цілісну, стійку до відмов та масштабовану архітектурну структуру. Вона забезпечує суворе розмежування зон відповідальності, високу швидкість обробки інформації та захищеність даних на всіх етапах життєвого циклу запиту. Для розуміння потоків даних та взаємодії структурних модулів розроблено загальну схему архітектури вебплатформи (рис. 1.5).

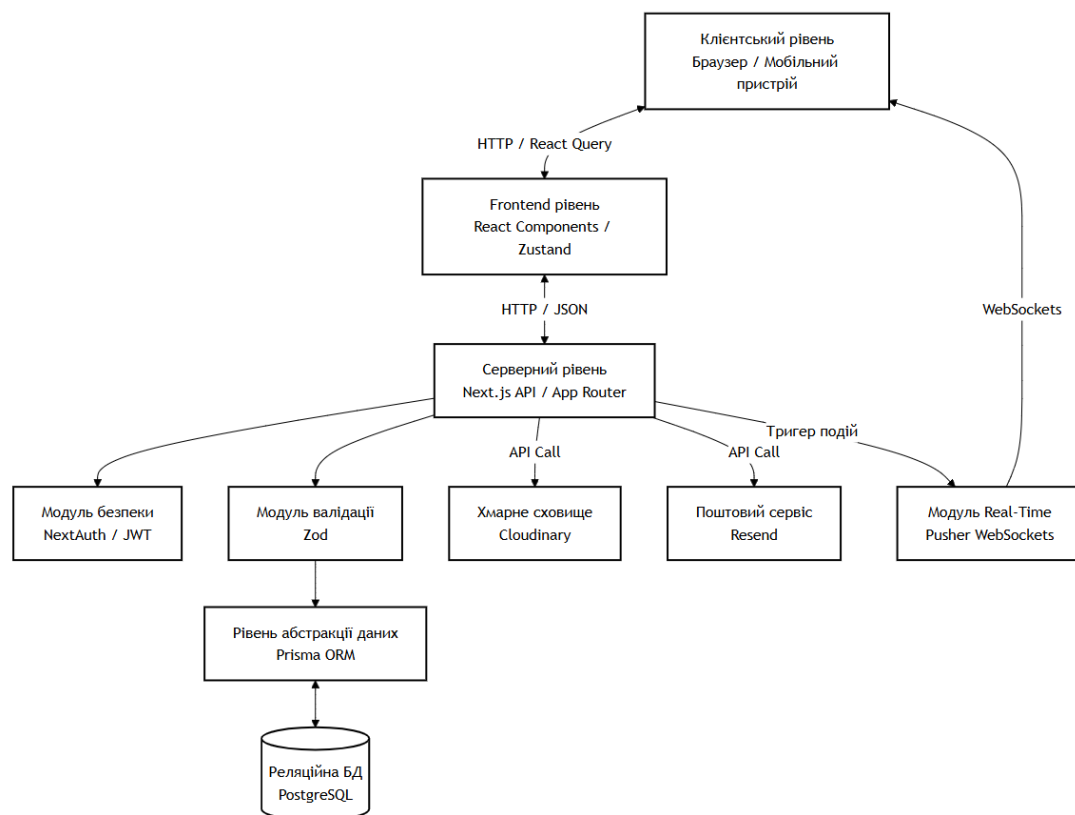


Рис. 1.5 – Структурна схема архітектури вебплатформи «Footballium»

1.3 Обґрунтування вибору інструментальних засобів

Успішна реалізація спроектованої складної клієнт-серверної архітектури вебплатформи вимагає надзвичайно ретельного та науково обґрунтованого підбору технологічного стека інструментів. Процес вибору програмних засобів здійснюється на основі жорстких критеріїв: загальної обчислювальної продуктивності, здатності до горизонтального та вертикального масштабування, наявності активної підтримки світовою спільнотою розробників, рівня вбудованої захищеності, а також бездоганної сумісності компонентів між собою під час розгортання у хмарному середовищі. З огляду на специфіку спортивної інформаційної системи, яка передбачає обробку великих масивів пов'язаних даних та трансляцію подій у режимі реального часу, обраний інструментарій повинен мінімізувати ризики виникнення вузьких місць (bottlenecks) в архітектурі.

Враховуючи необхідність опрацювання складних структурованих даних, головною мовою програмування для розробки всього проєкту як на клієнтському, так і на серверному рівнях було безпечібно обрано TypeScript. На відміну від класичного динамічного JavaScript, цей сучасний інструмент від корпорації Microsoft забезпечує сувору статичну типізацію програмного коду ще на етапі його написання та компіляції. Використання TypeScript дозволяє створювати чіткі інформаційні контракти (інтерфейси та типи) для таких складних предметних сутностей, як «Турнір», «Команда», «Гравець» чи «Статистичний звіт». Завдяки впровадженню парадигми наскрізної типізації (End-to-End Type Safety), будь-яка невідповідність структур даних між базою даних, сервером та клієнтським інтерфейсом миттєво підсвічується інтегрованим середовищем розробки. Це не лише автоматично виявляє критичні помилки до моменту розгортання системи у виробничому середовищі, але й значно полегшує колективну розробку, рефакторинг коду та подальше функціональне масштабування платформи.

У ролі фундаментального мета-фреймворку для побудови всієї інфраструктури застосунку використовується Next.js. Вибір цього інструмента зумовлений його унікальною здатністю поєднувати найкращі практики різних стратегій рендерингу в межах одного проєкту. Вбудовані механізми автоматичної оптимізації

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

растрових зображень, алгоритмічного стиснення шрифтів та гнучкої маршрутизації на основі файлової системи дозволяють створити високопродуктивний програмний продукт, який повністю відповідає найсуворішим критеріям сучасних вебстандартів (Core Web Vitals). Інтеграція середовища виконання Node.js [2] забезпечує високоефективну обробку асинхронних мережеских запитів завдяки неблокуючій подієво-орієнтованій архітектурі, що є критично важливим для забезпечення високої пропускної здатності серверної частини вебплатформи під час пікових навантажень у дні проведення ключових футбольних матчів.

Для забезпечення надійного зберігання складної реляційної структури даних, яка охоплює ієрархії користувачів, багатоетапні турніри, команди з їхніми заявками та хронологію спортивних подій, було інтегровано систему управління базами даних PostgreSQL [5, 26]. Відмова від нереляційних рішень на користь класичної СУБД пояснюється тим, що спортивна статистика вимагає жорстких обмежень цілісності (Referential Integrity) та гарантованого виконання складних багатовимірних аналітичних запитів (наприклад, агрегації турнірної таблиці на основі результатів усіх зіграних матчів). PostgreSQL забезпечує бездоганну транзакційну безпеку згідно з принципами ACID, що унеможливорює втрату або пошкодження даних при одночасній зміні результатів гри кількома модераторами.

Програмна взаємодія із цією базою даних реалізується через інструментарій Prisma ORM. Цей сучасний фреймворк кардинально відрізняється від традиційних ORM-систем тим, що він автоматично генерує повністю типізованого клієнта на основі єдиної декларативної схеми (schema.prisma). Такий підхід забезпечує надзвичайно зручне та передбачуване управління еволюцією структури бази даних через механізм міграцій, а також надає вбудований, надійний захист від найпоширеніших кібервразливостей, зокрема SQL-ін'єкцій, завдяки автоматичному параметризованому екрануванню всіх вхідних запитів.

Оскільки вебплатформа оперує конфіденційними даними тренерів, арбітрів та адміністраторів ліги, критичного значення набуває система ідентифікації та безкомпромісного контролю доступу. Цей модуль побудований на базі екосистеми NextAuth.js [9], яка дозволяє відмовитися від застарілих серверних сесій на користь

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

сучасного криптографічного підходу. Паролі користувачів необоротно хешуються з використанням сучасних алгоритмів (наприклад, bcrypt), а авторизаційні сесії управляються за допомогою захищених JSON Web Tokens (JWT) відповідно до міжнародного протоколу авторизації OAuth 2.0 [29, 30]. Для гарантування абсолютної валідності даних, що надходять від клієнта (зокрема, під час створення нового турніру, реєстрації команди чи заповнення протоколу матчу), імплементовано бібліотеку Zod [13]. Вона дозволяє інженерам створювати строгі декларативні схеми перевірки, які працюють синхронно як на клієнтському фронтенді для миттєвого зворотного зв'язку, так і на серверному бекенді як кінцевий рубіж оборони, відсікаючи будь-які спроби введення некоректної, нетипізованої чи шкідливої інформації.

Візуальна презентація платформи формується на основі декларативної компонентної архітектури бібліотеки React. Для стилізації інтерфейсних елементів застосовано прогресивний утилітарний CSS-фреймворк Tailwind CSS. Його імплементация дозволяє швидко та прогнозовано конструювати складні адаптивні макети без необхідності написання громіздких та важко підтримуваних каскадних таблиць стилів. Замість глобальних класів, розробник оперує низькорівневими утилітами безпосередньо в розмітці, що унеможливорює виникнення конфліктів стилів та гарантує мінімальний розмір фінального CSS-файлу завдяки автоматичному видаленню невикористаних класів під час збірки проєкту (PurgeCSS).

Для забезпечення високого рівня користувацького досвіду (UX) та побудови складних інтерактивних вузлів, таких як модальні вікна, багаторівневі випадаючі списки, контекстні меню та комплексні форми, залучено екосистему Shadcn UI [8] у синергії з базовими компонентами Radix UI. Цей архітектурний вибір гарантує не лише високу естетику дизайну, але й повну відповідність міжнародним стандартам доступності інтерфейсів (WAI-ARIA), що дозволяє коректно взаємодіяти з платформою користувачам з обмеженими можливостями або тим, хто використовує засоби екранного читання. Крім того, у панелі адміністратора для візуалізації статистичних звітів та аналітики продуктивності команд використовується бібліотека Recharts, яка алгоритмічно генерує інтерактивні векторні SVG-графіки. Для

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

створення розширених новинних матеріалів та публікацій регламентів турніру інтегровано потужний модульний rich-text редактор Tiptap, який надає модераторам широкі можливості форматування тексту без необхідності знання HTML-розмітки.

Керування потоками даних всередині клієнтського застосунку вимагає чіткого розмежування між станом сервера та локальним станом інтерфейсу. Ефективна синхронізація асинхронних даних між браузером та сервером (Server State) делегована інструментарію React Query. Ця бібліотека бере на себе складні завдання агресивного кешування відповідей сервера, дедуплікації однакових запитів та фоновому оновленню інформації (Stale-while-revalidate), що кардинально зменшує навантаження на мережу та забезпечує миттєвий відгук інтерфейсу. Водночас для управління швидкоплинним локальним станом інтерфейсу (Client State), наприклад, статусом відкритих бокових навігаційних панелей або кроками багатосторінкових майстрів реєстрації, застосовано легковагий і високопродуктивний інструмент Zustand [14]. На відміну від громіздких рішень типу Redux, Zustand не потребує написання надлишкового шаблонного коду (boilerplate) і забезпечує прямий доступ до стану з будь-якого компонента. Організація роботи зі складними формами, яких у системі спортивного менеджменту надзвичайно багато, реалізована через оптимізований пакет React Hook Form. Цей інструмент працює на основі неконтрольованих компонентів, що мінімізує кількість перемальовувань сторінки (re-renders) під час введення тексту користувачем, гарантуючи максимальну плавність роботи навіть на слабких мобільних пристроях.

Зрештою, для забезпечення ключової функціональності платформи — моніторингу подій у режимі реального часу — було інтегровано спеціалізований хмарний сервіс Pusher. Він бере на себе управління життєвим циклом WebSocket-з'єднань, гарантуючи миттєву і надійну доставку інформації про забиті голи, отримані картки чи завершення таймів усім підключеним вболівальникам та модераторам без потреби ручного оновлення сторінки. З метою оптимізації зберігання та доставки важкого мультимедійного контенту, такого як логотипи футбольних клубів, фотографії спортсменів та банери новин, інтегровано спеціалізоване хмарне сховище Cloudinary [12]. Цей сервіс не лише забезпечує надійне зберігання файлів на

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

глобальній мережі доставки контенту (CDN), але й автоматично виконує ресурсомісткі операції зі стиснення, масштабування та конвертації зображень у сучасні формати, суттєво зменшуючи навантаження на основний сервер застосунку. Підсумовуючи, обраний високотехнологічний комплекс інструментальних засобів утворює надзвичайно міцну, безпечну, адаптивну та продуктивну екосистему, яка здатна повною мірою задовольнити всі архітектурні, ділові та технічні вимоги до сучасної цифрової платформи управління спортивними змаганнями.

Висновки до першого розділу

У першому розділі кваліфікаційної роботи проведено системний аналіз предметної області та досліджено процеси інформаційного супроводу футбольних турнірів. Огляд існуючих систем-аналогів засвідчив, що сучасний регіональний та аматорський спортивний сегмент переважно використовує класичні вебпортали, які не здатні забезпечити необхідний рівень автоматизації. Виявлено низку критичних недоліків інфраструктури змагань: ручне формування турнірних сіток, відсутність інструментарію для оновлення результатів у режимі реального часу (Live-моніторинг) та збереження застарілого паперового документообігу у роботі тренерського штабу.

Для подолання виявлених обмежень обґрунтовано доцільність розробки спеціалізованої інтерактивної вебплатформи «Footballium». Визначено ключові функціональні вимоги до програмного продукту, концептуальною основою яких є впровадження гнучкої моделі управління доступом. Розподіл повноважень між чотирма системними ролями (Адміністратором, Арбітром, Тренером та Вболівальником) дозволить автоматизувати генерацію розкладів, перевести суддівські протоколи у цифровий формат безпосередньо під час матчів та суттєво оптимізувати процес подачі післяматчевої тренерської звітності.

На основі сформованих вимог здійснено вибір та детальне обґрунтування архітектурних рішень. Доведено, що для реалізації проєкту з високим рівнем інтерактивності та необхідністю обробки динамічних даних найкраще підходить модульна клієнт-серверна архітектура у форматі монорепозіторію. Застосування гібридного рендерингу (поєднання SSR для публічних сторінок та CSR для

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

захищених кабінетів) гарантує швидке завантаження контенту та високу пошукову оптимізацію застосунку.

Проведено комплексний аналіз інструментальних засобів реалізації системи. Вибір сучасного мета-фреймворку Next.js у поєднанні зі строгою типізацією TypeScript, потужною реляційною базою даних PostgreSQL та ORM-інструментарієм Prisma створює надійний, безпечний і стійкий до відмов фундамент застосунку. Водночас інтеграція подієво-орієнтованої моделі на базі протоколу WebSockets (за допомогою сервісу Pusher) забезпечить виконання критичної бізнес-вимоги – миттєвої синхронізації подій для всіх підключених клієнтів.

Таким чином, теоретичний огляд, системний аналіз проблематики та технологічне обґрунтування, виконані у першому розділі, повністю формують базис для наступного етапу розробки. Отримані результати дозволяють перейти до безпосереднього інженерного проєктування логічної структури бази даних, поведінкових алгоритмів взаємодії користувачів та об'єктно-орієнтованої моделі вебплатформи.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Моделювання варіантів використання та поведінки системи

Інженерне проєктування вебплатформи «Footballium» розпочинається з детальної формалізації вимог через моделювання поведінки системи. Для опису функціональних можливостей програмного продукту з взаємодії з ним застосовується метод побудови діаграм прецедентів (Use Cases) згідно зі стандартом UML (див. Додаток А). Цей підхід дозволяє чітко розмежувати відповідальність та рівні доступу для різних категорій користувачів. Аналіз бізнес-процесів організації спортивних турнірів дозволив виокремити п'ятьох основних акторів. З погляду моделі управління доступом (RBAC), вони поділяються на чотири авторизовані системні ролі (Адміністратор, Тренер, Суддя, Вболівальник) та один неавторизований стан (Гість). Кожна авторизована роль наділена специфічним набором повноважень, які регулюються підсистемою управління доступом, що відображено у таблиці 2.1.

Таблиця 2.1

Характеристика акторів системи та їхніх варіантів використання

Актор (Роль)	Рівень доступу	Основні прецеденти (сценарії використання)
Адміністратор	Найвищий (Глобальний)	Створення нових турнірів та генерація сіток. Модерація користувачів та управління правами. Призначення суддівських бригад на матчі. Створення та публікація новин ліги. Перегляд системних логів та глобальної статистики. Налаштування персонального профілю (фото, контакти)
Тренер	Спеціалізований (Командний)	Налаштування персонального профілю (фото, контакти). Редагування інформації про команду та її склад. Формування офіційної заявки гравців на майбутній матч. Аналіз історії відіграних матчів власної команди.
Суддя	Спеціалізований (Матчовий)	Перегляд призначених матчів та їх налаштування. Live-фіксація ігрових подій (голи, картки, заміни). Перегляд архіву відсуджених поєдинків. Налаштування персонального профілю (фото, контакти).
Вболівальник	Базовий інтерактивний	Додавання обраних команд до списку «Улюблених». Оцінювання новинного контенту (вподобання/лайки). Моніторинг турнірних таблиць, розкладів та статистики. Налаштування персонального профілю (фото, контакти)
Гість (Неавторизований користувач)	Лише читання (Публічний)	Перегляд розкладу та Live-рахунків поточних матчів. Читання новин та перегляд публічних профілів гравців.

Для візуального відображення описаних поведінкових сценаріїв та ієрархії доступів розроблено загальну діаграму варіантів використання (рис. 2.1).

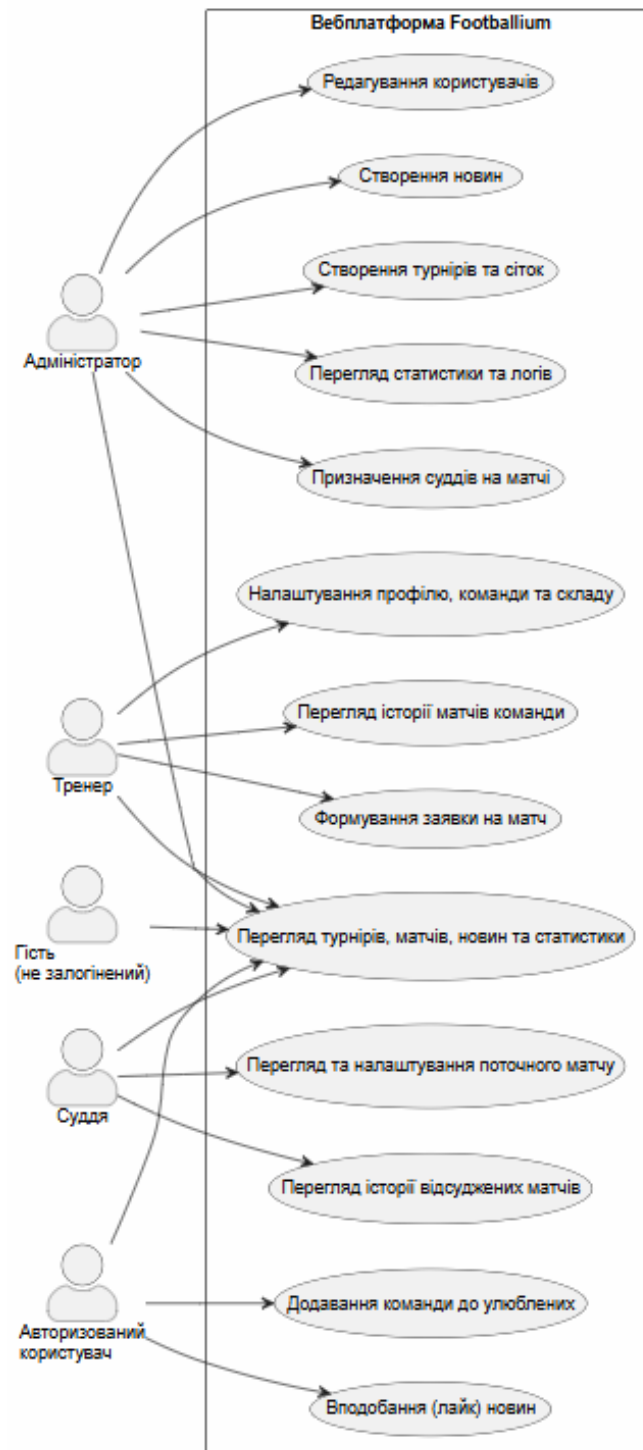


Рис. 2.1 – Діаграма варіантів використання вебплатформи

Наступним кроком проєктування є побудова об'єктно-орієнтованої структури системи. Оскільки взаємодія з базою даних реалізована через ORM-систему Prisma, класи платформи строго відповідають фізичним моделям даних. До складу базової об'єктної моделі входять наступні сутності.

Для більш глибокого розуміння поведінкової моделі платформи «Footballium», нижче наведено детальні специфікації ключових прецедентів, які відображають основну бізнес-логіку системи у таблиці 2.2. Специфікації оформлено у вигляді стандартизованих таблиць, що описують передумови, основні та альтернативні потоки подій.

Таблиця 2.2

Специфікація прецеденту «Формування заявки на матч»

Характеристика	Опис
Назва прецеденту	Формування заявки на матч
Актор	Тренер
Мета	Сформувати стартовий склад та список запасних гравців команди на конкретний майбутній поєдинок.
Передумови	Тренер успішно авторизований у системі. У базі даних існує запланований матч для команди цього тренера зі статусом «Очікується». У команді зареєстровано достатню кількість гравців.
Основний сценарій (Потік подій)	Тренер переходить до розділу «Розклад матчів» у своєму кабінеті. Система робить запит до БД і відображає список майбутніх ігор. Тренер обирає потрібний матч і натискає кнопку «Сформувати заявку». Система завантажує повний перелік гравців (ростер) команди. Тренер обирає Х гравців до стартового складу та резервних гравців. Тренер натискає кнопку «Зберегти». Система валідує дані та зберігає заявку в базі даних.
Альтернативні сценарії	Недостатня кількість гравців: Якщо тренер обрав менше ніж Х гравців для стартового складу, система деактивує кнопку «Зберегти» та виводить відповідне сповіщення про порушення регламенту. Некоректний формат даних: Якщо тренер вибрав однакових гравців у заявку або увів не коректний номер гравця, виникне помилка на сторінці заявки до матчу.
Постумови	Заявка команди зафіксована у системі. Склади команд стають доступними для перегляду Судді та Вболівальникам на публічній сторінці матчу.

Однією з фундаментальних інновацій проєктованої вебплатформи є здатність забезпечувати інтерактивний супровід спортивних подій безпосередньо під час їхнього перебігу. У таблиці 2.3 подано розгорнуту специфікацію прецеденту «Ведення Live-протоколу матчу». Цей сценарій відзначається найвищим рівнем інженерної складності з огляду на необхідність організації безперебійного двостороннього зв'язку між клієнтом та сервером на базі подієво-орієнтованої архітектури. Специфіка даного процесу полягає у критичній потребі надшвидкої фіксації рішень арбітра на полі з наступною миттєвою ретрансляцією оновлених ігрових

показників усім підключеним вболівальникам. Зважаючи на високі вимоги до оптимізації мережових запитів та уникнення десинхронізації даних, наведена специфікація покроково формалізує логіку переходів, обов'язкові передумови та алгоритми обробки виключних ситуацій, які виступають запорукою відмовостійкого функціонування модуля трансляцій.

Таблиця 2.3

Специфікація прецеденту «Ведення Live-протоколу матчу»

Атрибут	Опис
Назва прецеденту	Ведення Live-протоколу матчу
Актор	Суддя (Офіційна особа матчу)
Мета	Безперервна реєстрація ігрових епізодів (взяття воріт, дисциплінарні санкції, ротація складу) в динамічному режимі для синхронного оновлення турнірних показників у системі.
Передумови	Офіційна особа успішно пройшла автентифікацію в мобільній версії вебзастосунку. Системою зафіксовано календарний час початку поєдинку, на який призначено даного арбітра. Тренерські штаби команд-учасниць попередньо затвердили та зберегли в базі даних стартові ростери на гру.
Основний сценарій (Успішний)	Арбітр ініціює перехід до робочого простору призначеного поєдинку. Суддя активує тригер «Розпочати матч». Програмний комплекс переводить статус події у стан «LIVE» та ініціалізує серверний хронометраж. У разі настання результативної дії (наприклад, голу), арбітр обирає відповідну категорію події, команду та ідентифікує гравця-автора з наданого списку. Клієнтський застосунок формує серіалізований об'єкт події та відправляє асинхронний запит до серверного інтерфейсу. Бекенд записує транзакцію в БД та миттєво ретранслює оновлені показники всім активним клієнтам (Вболівальникам) через протокол WebSockets. Після вичерпання регламентованого ігрового часу арбітр фіналізує протокол командою «Завершити матч».
Альтернативні сценарії (Обробка виключень)	Хибне внесення даних: У випадку випадкової реєстрації події (наприклад, гол записано не на того спортсмена), арбітр наділений повноваженнями анулювати останній запис у таймлайні (виключно до моменту фінального свистка), що автоматично викликає процедуру відкоту поточного рахунку.
Постумови	Поєдинок переводиться у термінальний статус «Завершено». Серверне ядро автономно здійснює перерахунок глобальної турнірної таблиці та оновлює індивідуальні метрики футболістів. Будь-яке подальше втручання в електронний рапорт блокується алгоритмами системи безпеки.

2.2 Проєктування логічної структури бази даних

З метою гарантування надійного збереження масивів спортивної інформації та підтримки безкомпромісної транзакційної цілісності даних було обрано реляційну систему управління базами даних. У контексті сучасних підходів до

побудови високопродуктивних мережесих програм ця парадигма реалізується через розгортання середовища PostgreSQL [28]. Подібний симбіоз забезпечує ідеальний компроміс між надійністю строгих реляційних зв'язків та моделлю сутність-атрибут-значення (EAV) [25]. Використання класичної реляційної моделі повністю задовольняє суворі вимоги щодо безпечної обробки транзакцій. Це є критично важливим аспектом під час оперування статистикою футбольних турнірів та збереження персональних налаштувань профілів, оскільки будь-яка втрата чи викривлення інформації неминуче призведе до критичних помилок у формуванні турнірних таблиць.

Поряд із цим підсистема моніторингу вимагає опрацювання високодинамічних структур даних через наявність специфічних метрик у кожному ігровому епізоді. Для ефективного керування комплексними конфігураціями архітектура застосування залучає інструментарій ORM-системи Prisma. Подібний симбіоз забезпечує ідеальний компроміс між надійністю строгих реляційних зв'язків для базових сутностей та гнучкістю під час проєктування складних запитів. Впроваджене архітектурне рішення суттєво мінімізує необхідність у ручному написанні складних міграцій під час оновлення платформи та допомагає уникнути поширених інженерних антипатернів.

Спроектована об'єктно-реляційна модель наочно ілюструє базові предметні сутності аналітичної платформи разом з їхніми внутрішніми атрибутами, а також механізми суворої ідентифікації через первинні та зовнішні ключі. Серед логічних залежностей варто виокремити класичні зв'язки формату «один-до-багатьох», які об'єднують облікові записи користувачів з їхніми улюбленими командами, та складні відношення між турнірами, матчами та ігровими подіями. Сформована у такий спосіб архітектурна модель генерує єдиний узгоджений простір даних. Це гарантує надзвичайно швидку вибірку необхідної інформації через серверний програмний інтерфейс для її миттєвої доставки на клієнтський браузер.

Спроектована реляційна схема даних включає строгі фізичні обмеження цілісності для забезпечення узгодженості та швидкодії. Технічні специфікації ключових обмежень зовнішніх ключів та індексів наведено у таблиці 2.4. Застосування

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		31

механізму каскадного видалення гарантує відсутність незв'язаних або покинутих записів у базі, тоді як індексація висококардинальних полів суттєво зменшує час відгуку системи під час генерації загальних турнірних таблиць. Це дозволяє підтримувати високу продуктивність та стабільність платформи навіть за умов стрімкого зростання обсягів збереженої статистики.

Таблиця 2.4

Специфікація фізичних обмежень та індексів бази даних

Сутність (Таблиця)	Цільове поле	Тип обмеження	Опис / Поведінка
User	email	UNIQUE CONSTRAINT	Запобігає реєстрації дублікатів облікових записів у системі.
Team	coachId	FOREIGN KEY	Зв'язок із таблицею користувачів для визначення тренера команд.
Match	homeTeamId, awayTeamId	FOREIGN KEY	Забезпечує строгу цілісність прив'язки учасників до існуючих команд у базі даних.
MatchEvent	matchId	FOREIGN KEY	ON DELETE CASCADE (автоматичне очищення подій таймлайну при видаленні матчу з бази).
Standing	tournamentId, teamId	COMPOSITE UNIQUE	Обмеження рівня БД: команда може мати лише один запис турнірного становища в межах конкретного змагання.
Like	userId, newsId	COMPOSITE PK	Складений первинний ключ таблиці асоціацій, що фізично блокує можливість повторного оцінювання (лайку) однієї новини.
Player	teamId	INDEX	Прискорення складних вибірок ростеру команди під час формування стартового складу.

Для глибокої деталізації зв'язків між таблицями та опису архітектури збереження даних було розроблено загальну концептуальну схему. Вона відображає ієрархічну структуру платформи, де центральне місце займає логічна взаємодія між профілем користувача та модулями змагань. Процес побудови таких логічних зв'язків повністю базується на фундаментальних принципах реляційної нормалізації та мінімізації інформаційної надлишковості. Це у підсумку гарантує стабільну роботу серверної частини при обробці великих масивів турнірної статистики.

Фундаментальні зв'язки між сутностями спроектовано з дотриманням суворих архітектурних стандартів. Наприклад, між обліковими записами тренерів та їхніми спортивними клубами діє жорстке відношення делегування повноважень.

Важливим інженерним рішенням є впровадження механізму каскадного очищення: при знищенні певного турніру система автоматично стирає всі пов'язані з ним матчі та поточні турнірні таблиці. Детальну специфікацію сутності для збереження облікових записів та параметрів доступу наведено у таблиці 2.5.

Таблиця 2.5

Структура таблиці користувачів

Назва поля	Тип даних	Обмеження	Опис
id	VARCHAR(255)	PK	Унікальний текстовий ідентифікатор користувача платформи.
name	VARCHAR(255)	NULL	Публічне ім'я або нікнейм особи в системі.
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна пошта для безпечної авторизації та сповіщень.
passwordHash	VARCHAR(255)	NULL	Зашифрований криптографічним алгоритмом ключ доступу.
role	ENUM	NOT NULL, DEFAULT 'USER'	Базова роль доступу (Адміністратор, Тренер, Суддя, Користувач).
image	VARCHAR(500)	NULL	Гіперпосилання на завантажений графічний аватар профілю.

Центральною сутністю підсистеми управління складами є таблиця команд. Вона містить загальні метадані спортивного колективу, графічні ідентифікатори та посилання на офіційного представника. Окрім цього, структура поданої нижче таблиці 2.6 закладає міцний фундамент для гнучкого управління складами. Вона дає змогу безшовно інтегрувати кожен колектив у ширшу змагальну екосистему розроблюваної платформи, автоматизуючи процеси прив'язки клубів до динамічних турнірних таблиць, розкладів майбутніх поєдинків та алгоритмів накопичення історичної ігрової статистики.

Таблиця 2.6

Структура таблиці спортивних команд

Назва поля	Тип даних	Обмеження	Опис
id	VARCHAR(255)	PK	Унікальний ідентифікатор створеного клубу.
name	VARCHAR(255)	NOT NULL	Офіційна назва спортивного колективу.
logo	VARCHAR(500)	NULL	Посилання на емблему команди, що зберігається у хмарному сховищі.
coachId	VARCHAR(255)	FK, UNIQUE	Пряме посилання на профіль тренера, який має права адміністрування команди.
updatedAt	TIMESTAMP	NOT NULL	Часовий маркер останньої модифікації даних клубу.

З метою створення абсолютно прозорого середовища для проведення змагань, а також для технічного забезпечення інтерактивних Live-трансляцій, в архітектуру бази даних інтегровано спеціалізований модуль реєстрації ігрових епізодів. Спроектowana таблиця відіграє роль цифрового хронографа матчу, послідовно акумулюючи дані про взяття воріт, винесені арбітром дисциплінарні санкції та ротацію складу команд. Збереження інформації у такому атомарному вигляді є фундаментальною умовою для коректного автоматичного підрахунку поточного та фінального рахунку поєдинку, а також для формування об'єктивної індивідуальної статистики футболістів. Фізичну конфігурацію сутності, що відповідає за збереження ігрового таймлайну, деталізовано у таблиці 2.7.

Таблиця 2.7

Структура таблиці ігрових подій (MatchEvent)

Назва поля	Тип даних	Обмеження	Опис
id	VARCHAR(255)	PK	Унікальний ідентифікатор конкретної події на полі.
matchId	VARCHAR(255)	FK, NOT NULL	Посилання на транзакційний запис матчу, під час якого відбулася дія.
eventType	ENUM	NOT NULL	Категоризація події (наприклад: гол, жовта картка, червона картка).
minute	INT	NOT NULL	Хвилина основного або компенсованого ігрового часу.
playerId	VARCHAR(255)	FK, NOT NULL	Ідентифікатор головного спортсмена, який виконав результативну дію.

Ретельно продумана структура відношень формує надійно узгоджений та оптимізований інформаційний фундамент. Застосована архітектурна модель забезпечує безпрецедентно високу швидкодію системи. Під час відкриття сторінки живого поєдинку серверна частина виконує атомарний запит до бази даних, витягує повну конфігурацію ігрових подій і миттєво транслює її до клієнтського браузера. Фронтенд здійснює візуалізацію складної структури інтерфейсу, паралельно ініціюючи асинхронні запити для завантаження важких медіафайлів із хмарного сервісу. Завдяки цьому досягається абсолютна безперервність користувацького досвіду без

жодних відчутних мережевих затримок під час обробки живих поточкових оновлень зі спортивних арен.

2.3 Проєктування компонентної структури та взаємодії модулів

З метою комплексної візуалізації фізичної топології та детальної інфраструктурної конфігурації вебплатформи «Footballium» було розроблено оптимізовану модель розгортання (див. Додаток Б, В та Г). Задля досягнення найвищого рівня гнучкості програмної інфраструктури та суворого слідування принципам об'єктно-орієнтованого дизайну SOLID, проєктування базується на архітектурних шаблонах, що гарантує виконання нефункціональних вимог [23]. Основна обробка бізнес-логіки виконується у хмарному середовищі Vercel, збереження даних делеговано кластеру Neon [10], а управління глобальним станом інтерфейсу здійснюється за допомогою інструментарію Zustand [14]. Такий підхід гарантує максимальний рівень відмовостійкості системи, надає можливості динамічного горизонтального масштабування обчислювальних вузлів та забезпечує доставку аналітичного контенту кінцевим користувачам навіть в умовах стрибків мережевого трафіку, що є типовим явищем під час проведення масових спортивних поєдинків у вихідні дні.

Клієнтський рівень аналітичної платформи реалізовано за передовим принципом єдиного адаптивного вебзастосунку (Responsive Web Application), побудованого на базі компонентної архітектури. Усі авторизовані актори системи, включаючи вболівальників, тренерів, суддів та адміністраторів, здійснюють доступ до ізольованого функціоналу виключно через стандартні веббраузери своїх апаратних пристроїв (персональних комп'ютерів, планшетів чи смартфонів). Це стратегічне рішення усуває необхідність тривалої розробки та підтримки відокремлених нативних мобільних клієнтів для різних операційних систем, водночас гарантуючи абсолютно ідентичний та безшовний користувацький досвід незалежно від обраної платформи. Браузерні клієнти взаємодіють із серверним ядром через мережевий шлюз із застосуванням криптографічних протоколів, що унеможливорює перехоплення конфіденційних сесійних токенів.

Основна обробка бізнес-логіки, статична та динамічна генерація сторінок, а також маршрутизація прикладного інтерфейсу (REST API) виконуються у

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

високопродуктивному хмарному середовищі Vercel із використанням сучасної архітектури безсерверних функцій (Serverless Functions). Надійне збереження транзакційних спортивних даних та складних реляційних зв'язків між сутностями повністю делеговано хмарному кластеру PostgreSQL на платформі Neon.

Окремими критично важливими інфраструктурними вузлами виступають глибокі інтеграції зі спеціалізованими зовнішніми сервісами, які знімають надмірне навантаження з основного сервера. Зокрема, управління мультимедійним контентом інтегровано з хмарною платформою Cloudinary, яка функціонує як глобальна мережа доставки контенту (CDN) та бере на себе всі завдання з алгоритмічного стиснення, кешування та швидкісного рендерингу масивних клубних логотипів. Для забезпечення безперебійної двосторонньої комунікації під час Live-трансляцій впроваджено службу Pusher. Завдяки цьому архітектурному рішення будь-яка результативна дія на полі, зафіксована арбітром у мобільній панелі, миттєво транслюється через виділені канали зв'язку до всіх підключених клієнтських пристроїв, оновлюючи публічне цифрове табло без потреби примусового перезавантаження вебсторінки (рис. 2.2).

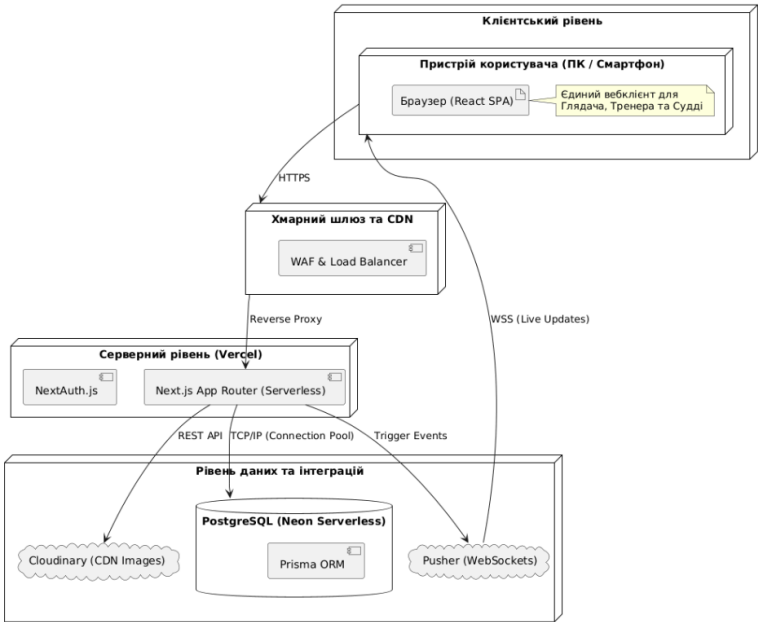


Рис. 2.2 – Діаграма розгортання інфраструктури

Задля досягнення найвищого рівня гнучкості програмної інфраструктури та суворого слідування принципам об'єктно-орієнтованого дизайну SOLID, проектування як серверної, так і клієнтської складових аналітичної платформи

«Footballium» базується на архітектурних шаблонах (патернах). Впроваджена багаторівнева структурна модель гарантує чітке делегування обов'язків між усіма системними вузлами, мінімізує зв'язність коду (Loose Coupling) та забезпечує високу масштабованість проєкту на етапах його подальшої модернізації.

Зокрема, управління взаємодією з реляційним хмарним сховищем реалізовано через породжувальний патерн «Одинак» (Singleton). Його інтеграція на рівні клієнта об'єктно-реляційного відображення Prisma є критично важливою вимогою для роботи у безсерверному середовищі розгортання. Таким чином інкапсулюється єдина глобальна точка доступу до бази даних, що надійно захищає пул з'єднань PostgreSQL від критичного вичерпання ресурсів під час одночасної обробки тисяч паралельних запитів від користувачів.

Паралельно з цим, інноваційний механізм миттєвої доставки ігрової статистики та результативних дій функціонує на основі класичного поведінкового концепту «Спостерігач» (Observer). У цій парадигмі серверне ядро, інтегроване зі службою Pusher через канали WebSockets, відіграє роль глобального публікатора подій (Publisher). Натомість клієнтські модулі вболівальників та інтерактивні табло виступають у ролі підписників (Subscribers), які автономно реєструються на відстеження змін у конкретному цільовому поєдинку.

Захист прикладного програмного інтерфейсу та конфіденційних маршрутів спирається на структурний шаблон «Замісник» (Proxy), який імплементовано глибоко в екосистему проміжного програмного забезпечення (Middleware) бібліотеки NextAuth. Даний системний компонент виконує функцію інтелектуального попереднього бар'єра, перехоплюючи абсолютно всі вхідні HTTP-запити. Він самостійно перевіряє наявність, цілісність та легітимність криптографічних сесійних токенів (JWT) ще до моменту фактичної маршрутизації мережових звернень до захищених API-ендпойнтів бекенду. Це дозволяє повністю абстрагувати бізнес-контролери від рутинної логіки перевірки прав доступу, повертаючи помилку несанкціонованого доступу на найпершому етапі життєвого циклу запиту (рис. 2.3).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

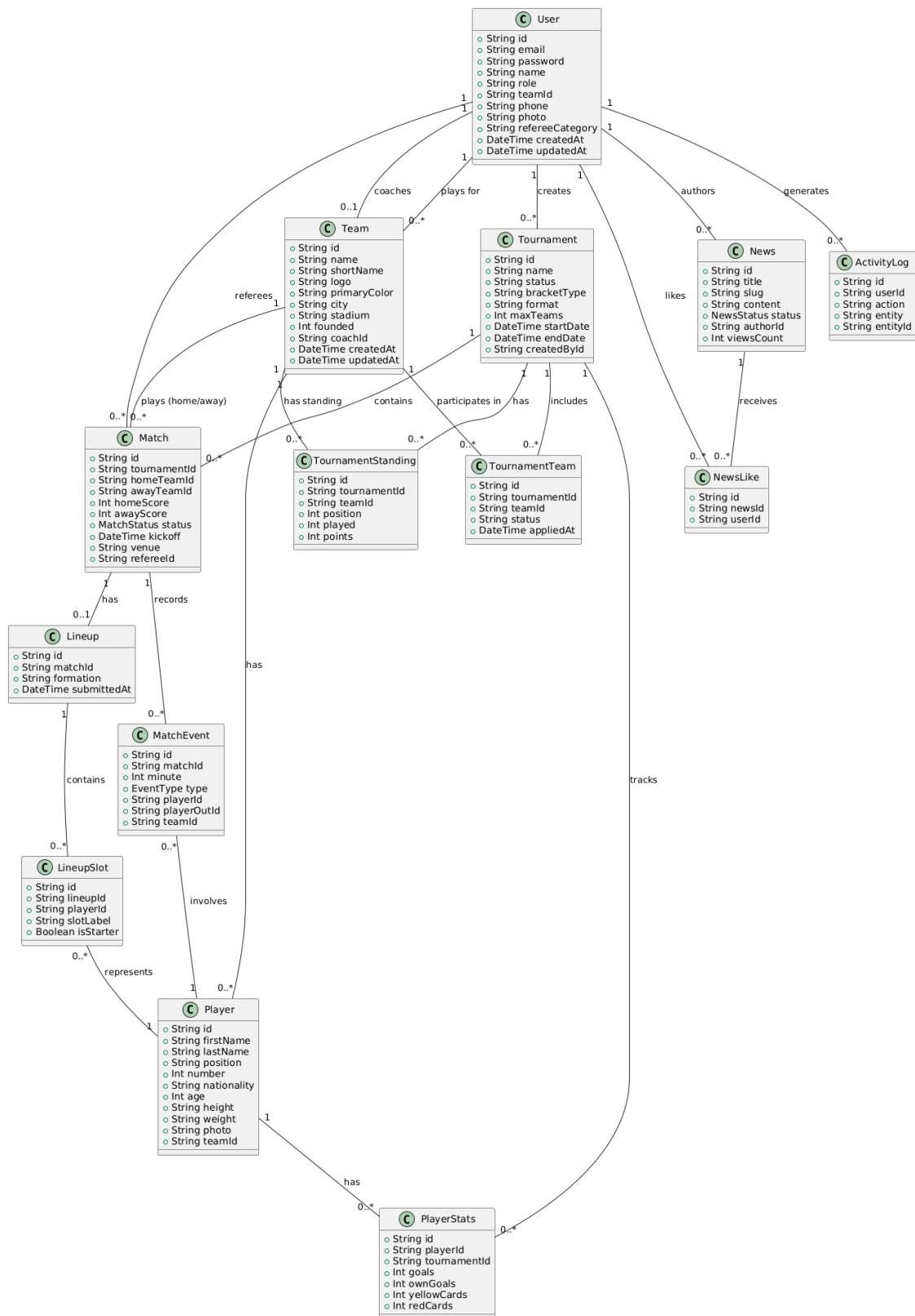


Рис. 2.3 – Діаграма класів реалізації та патернів проєктування

Для візуалізації динаміки обробки складних транзакційних запитів розроблено діаграму послідовності (рис. 2.4). Хронологічний цикл типової бізнес-операції — наприклад, фіксації голу арбітром — розпочинається з генерації події на клієнті.

Змн.	Арк.	№ докум.	Підпис	Дата

Далі API-маршрут ініціює валідацію сесії. Після підтвердження прав (роль REFEREE) сервісний шар зберігає подію через Prisma та відправляє тригер у WebSocket-канал для миттєвого оновлення рахунку на екранах вболівальників.

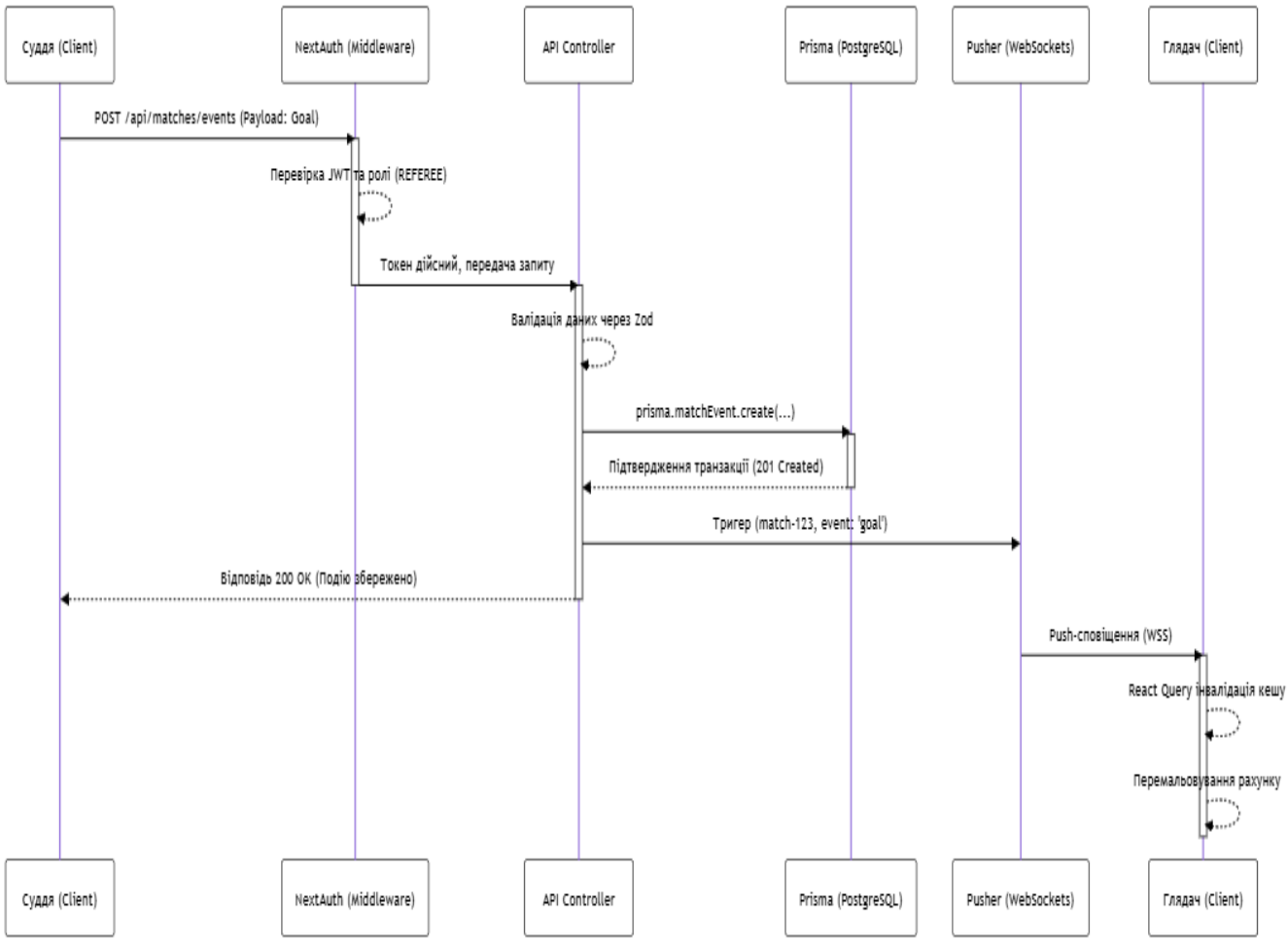


Рис. 2.4 – Діаграма послідовності процесу фіксації Live-події

Гарантування конфіденційності та стійкості до кібератак є фундаментальною вимогою до спортивної платформи. Архітектуру безпеки побудовано за принципом ешелонованого захисту. Замість застарілих сесійних сховищ інтегровано технологію JSON Web Tokens (JWT) через NextAuth.js. Цей stateless-механізм унеможливорює перехоплення авторизації через вразливості та знижує навантаження на БД. Додатково, для блокування спроб несанкціонованого підвищення привілеїв, на рівні проміжного програмного забезпечення (Middleware) впроваджено строгу рольову модель контролю доступу (RBAC), яка ізолює адміністративні панелі та інструменти арбітрів від публічного сегмента системи. У рамках проактивного підходу до моделювання безпеки було проведено глибокий аналіз потенційних

вразливостей. Детальну специфікацію ідентифікованих векторів загроз, а також застосовані інженерні інструменти та алгоритми їхньої апаратно-програмної нейтралізації наведено у таблиці 2.8.

Таблиця 2.8

Модель загроз інформаційної безпеки та алгоритми нейтралізації

Вектор атаки (Загроза)	Опис вразливості в контексті платформи	Архітектурний механізм захисту (Мітигація)
XSS (Cross-Site Scripting)	Ін'єкція шкідливих скриптів через поля вводу (наприклад, текстові звіти тренерів або новини).	Автоматичне екранування даних на рівні віртуального DOM у React; жорстка валідація вводу через схеми Zod.
CSRF (Cross-Site Request Forgery)	Несанкціонована генерація запитів від імені авторизованого тренера (наприклад, зміна складу).	Використання вбудованих Anti-CSRF механізмів NextAuth та конфігурація SameSite для cookies.
SQL Injection	Маніпуляція параметрами запитів для отримання доступу до бази турнірів та користувачів.	Повна відмова від сирих SQL-скриптів. Взаємодія з БД відбувається виключно через типізовані та параметризовані методи Prisma ORM.
DDoS-атаки	Перевантаження серверних потужностей штучним трафіком під час трансляцій фінальних матчів.	Делегування фільтрації вхідного трафіку на рівень хмарних балансувальників; імплементація алгоритмів Rate Limiting на API-маршрутах.

Функціонування клієнтського рівня розроблюваної системи нерозривно пов'язане з необхідністю безперервної обробки інтенсивних та високодинамічних потоків спортивної інформації. З огляду на це, традиційні підходи до збереження даних безпосередньо всередині візуальних компонентів виявилися б у край неефективними та призвели б до деградації продуктивності браузера. Для вирішення цієї проблеми загальну концепцію управління станом було глибоко декомпозовано на ізольовані семантичні контури, детальна характеристика яких наведена у таблиці 2.9. Впровадження такої багаторівневої архітектури дозволило повністю використати інженерний антипатерн наскрізного передавання властивостей крізь глибоке дерево елементів. Розмежування синхронного локального стану, що контролює поведінку інтерфейсу, та асинхронного серверного стану, який зберігає результати мережових запитів, гарантує точкове оновлення виключно тих віджетів, які зазнали

фактичних змін. Як наслідок, платформа уникає каскадних надлишкових рендерів, що є критично важливою умовою для забезпечення плавної реактивності інтерфейсу під час паралельної обробки десятків Live-подій.

Таблиця 2.9

Декомпозиція рівнів управління станом у вебзастосунку

Тип стану	Область застосування	Технологія / Інструмент	Механізм оновлення
Локальний (UI State)	Стан розкритих бокових меню, активні вкладки у кабінеті, поля вводу форм.	React useState / useReducer	Синхронне перемальовування конкретного UI-компонента.
Глобальний (App State)	Тема оформлення (Dark/Light), загальні налаштування інтерфейсу користувача.	Zustand	Реактивне оповіщення підписників без необхідності обгортання у Context Providers.
Серверний (Async State)	Списки гравців, історія матчів, турнірні сітки.	React Query (@tanstack)	Асинхронне фонове отримання даних із застосуванням інвалідації кешу.
Реактивний (Live State)	Події матчів у реальному часі, зміна рахунку, таймери поєдинків.	WebSockets (Pusher)	Push-сповіщення від серверного ядра до активних клієнтів.

Дослідження профілю навантаження типового спортивного порталу виявляє яскраво виражену інформаційну асиметрію: частота транзакцій читання, таких як перегляд розкладів чи моніторинг результатів тисячами вболівальників, у сотні разів перевищує частоту операцій запису, що генеруються обмеженою кількістю арбітрів чи тренерів. Враховуючи цю специфіку, фундаментальним інженерним рішенням для гарантування відмовостійкості серверного ядра під час пікових сплесків мережевого трафіку стала імплементація агресивної багаторівневої політики кешування. Суть цього механізму полягає у максимальному скороченні прямих ресурсомістких звернень до реляційної бази даних PostgreSQL шляхом тимчасового збереження обчислених результатів за допомогою вбудованих механізмів кешування Next.js (Data Cache) та на рівні глобальної мережі доставки контенту (Vercel Edge Network). У таблиці 2.10 деталізовано конфігурації кешування, адаптовані під унікальні вимоги різних інформаційних підсистем платформи.

Стратегії кешування інформаційних підсистем

Сегмент даних (Підсистема)	Технологія / Патерн	Умова інвалідації кешу (Очищення)
Публічні новини та правила	Static Site Generation (SSG)	Оновлення під час процесу збирання (Build) або за ручним тригером адміністратора.
Сторінки розкладу турнірів	Incremental Static Regeneration (ISR)	Фонові регенерація на сервері кожні 60 секунд за умови наявності нових запитів.
Дані кабінету тренера	Stale-While-Revalidate (SWR / React Query)	Миттєва інвалідація одразу після збереження нової заявки або редагування профілю.

Організація надійного інформаційного обміну між реактивним браузерним клієнтом та серверним ядром платформи суворо підпорядкована принципам архітектурного стилю REST (Representational State Transfer). Вибір цієї комунікаційної парадигми зумовлений її здатністю забезпечувати високий рівень незалежності компонентів завдяки концепції маршрутизації без збереження стану. Для гарантування бездоганної структурної узгодженості та мінімізації накладних витрат під час мережевої передачі, серіалізація абсолютно всіх пакетів даних здійснюється у стандартизованому текстовому форматі JSON. Такий підхід не лише підтримується сучасними вебрушіями, але й створює ідеальне середовище для імплементації суворої типізації запитів на етапі їхнього транзиту. У таблиці 2.11 наведено розгорнутий витяг зі специфікації ключових програмних інтерфейсів, які обслуговують найбільш критичні бізнес-модулі розроблюваної системи.

Таблиця 2.11

Специфікація маршрутів та методів REST API

HTTP Метод	Маршрут (Endpoint)	Призначення	Формат Payload (Запит)
POST	/api/teams/application	Створення стартової заявки команди на матч (Доступ: Тренер).	{"matchId": string, "players": array}
POST	/api/matches/events	Фіксація Live-події на полі (Доступ: Суддя).	{"matchId": string, "type": enum, "minute": int, "playerId": string}
GET	/api/matches/live	Отримання масиву поточних поєдинків та їхніх рахунків.	Немає (передача фільтрів через Query Params)
PUT	/api/users/profile	Оновлення персональних налаштувань візуального профілю.	{"name": string, "phone": string, "image": url}

У разі відсутності легітимного токена доступу або недостатнього рівня привілеїв, контролер миттєво генерує відмову з HTTP-статусом 401 Unauthorized або

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

403 Forbidden. Якщо ж помилка виникає на етапі валідації вхідних даних (наприклад, пошкоджений payload), система повертає статус 400 Bad Request. Такий багаторівневий бар'єрний механізм діє як надійний фільтр, що фундаментально унеможливорює потрапляння неповних або зловмисно модифікованих масивів.

Управління життєвим циклом футбольних турнірів супроводжується операціями з високою транзакційною складністю. Критичним етапом є автоматична генерація турнірної сітки або календаря чемпіонату, що вимагає одночасного створення масиву нових поєдинків, ініціалізації стартових показників команд та оновлення загального статусу турніру на рівні бази даних. Враховуючи специфіку розгортання серверної частини у безсерверному середовищі, час виконання таких обчислювально містких алгоритмів та мережових запитів є обмеженим. У разі виникнення апаратного чи програмного збою під час покрокового запису (наприклад, через вичерпання ліміту часу виконання функції або нестабільність з'єднання з хмарним кластером) існує високий ризик появи частково створених турнірних сіток. Це неминуче призведе до порушення логічної консистентності всієї інформаційної системи: появи «осиротілих» матчів без прив'язки до стадій чемпіонату, дублювання суперників або блокування подальшої роботи адміністраторів.

Для запобігання таким інфраструктурним ризикам в архітектурі платформи реалізовано механізм атомарних транзакцій на базі засобів об'єктно-реляційного відображення Prisma. На серверному рівні відповідні операції інкапсулюються у єдиний транзакційний блок, який об'єднує процес масової генерації матчів сітки та переведення статусу турніру в активний стан в одну неподільну логічну сесію. Якщо на будь-якому з етапів збереження відбувається помилка, інтегроване ядро системи управління базами даних миттєво ініціює процедуру автоматичного відкоту. Це скасовує всі попередні проміжні записи та повертає базу даних до первісного, узгодженого стану. Такий підхід гарантує суворе виконання класичних вимог до транзакцій, детальний опис яких у контексті застосованих технологій наведено у таблиці 2.12. Крім того, завдяки дотриманню принципу ізольованості повністю виключається ймовірність «брудного читання» (Dirty Read), коли паралельні

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		43

користувачські запити від вболівальників могли б випадково отримати доступ до ще не до кінця сформованого розкладу турніру.

Таблиця 2.12

Реалізація властивостей транзакцій в абстракціях Prisma ORM

Властивість ACID	Інженерна реалізація у проєкті	Архітектурне значення
Атомарність	Виклик блоку транзакції для масового створення та оновлення.	Гарантує виконання всіх операцій генерації як єдиного цілого без можливості часткового збереження.
Узгодженість	Використання жорстких зовнішніх ключів та каскадних правил у схемі БД.	Забезпечує перехід бази даних від одного валідного стану до іншого без порушення обмежень цілісності.
Ізольованість	Застосування базових рівнів ізоляції у СУБД PostgreSQL.	Паралельні запити користувачів не отримують доступ до частково згенерованої сітки до моменту коміту.
Довговічність	Запис логів транзакцій у журнал кластера Neon перед їх застосуванням.	Підтверджує неможливість втрати збережених даних навіть за умови критичного апаратного збою сервера.

Детальна алгоритмічна послідовність дій системи під час виникнення критичних помилок базується на механізмі перехоплення винятків та транзакційній ізоляції. У момент ініціалізації процесу генерації сервер відкриває єдину транзакційну сесію, в межах якої починає покрокове виконання інструкцій запису. Якщо на будь-якому з етапів, наприклад під час масового збереження нових поєдинків до таблиці матчів або зміни поточного статусу чемпіонату, виникає програмний збій чи порушення мережевого з'єднання з хмарним кластером бази даних, рушій системи управління базами даних PostgreSQL негайно перериває подальше виконання коду. Завдяки вбудованим механізмам реляційної СУБД миттєво ініціюється автоматичний відкіт (Rollback) усіх попередньо виконаних у межах цієї сесії маніпуляцій, що гарантовано повертає базу даних до її первісного, консистентного стану, який існував до початку транзакції. Після цього процес безпечно

завершується, а клієнтський застосунок отримує відповідне HTTP-сповіщення про помилку генерації, що повністю унеможлиблює ризик взаємодії користувачів із пошкодженими чи частково створеними даними. Розгорнуту логіку роботи механізму автоматичного відкоту при виникненні виняткових ситуацій під час генерації турніру відображено на алгоритмічній блок-схемі (рис. 2.5).

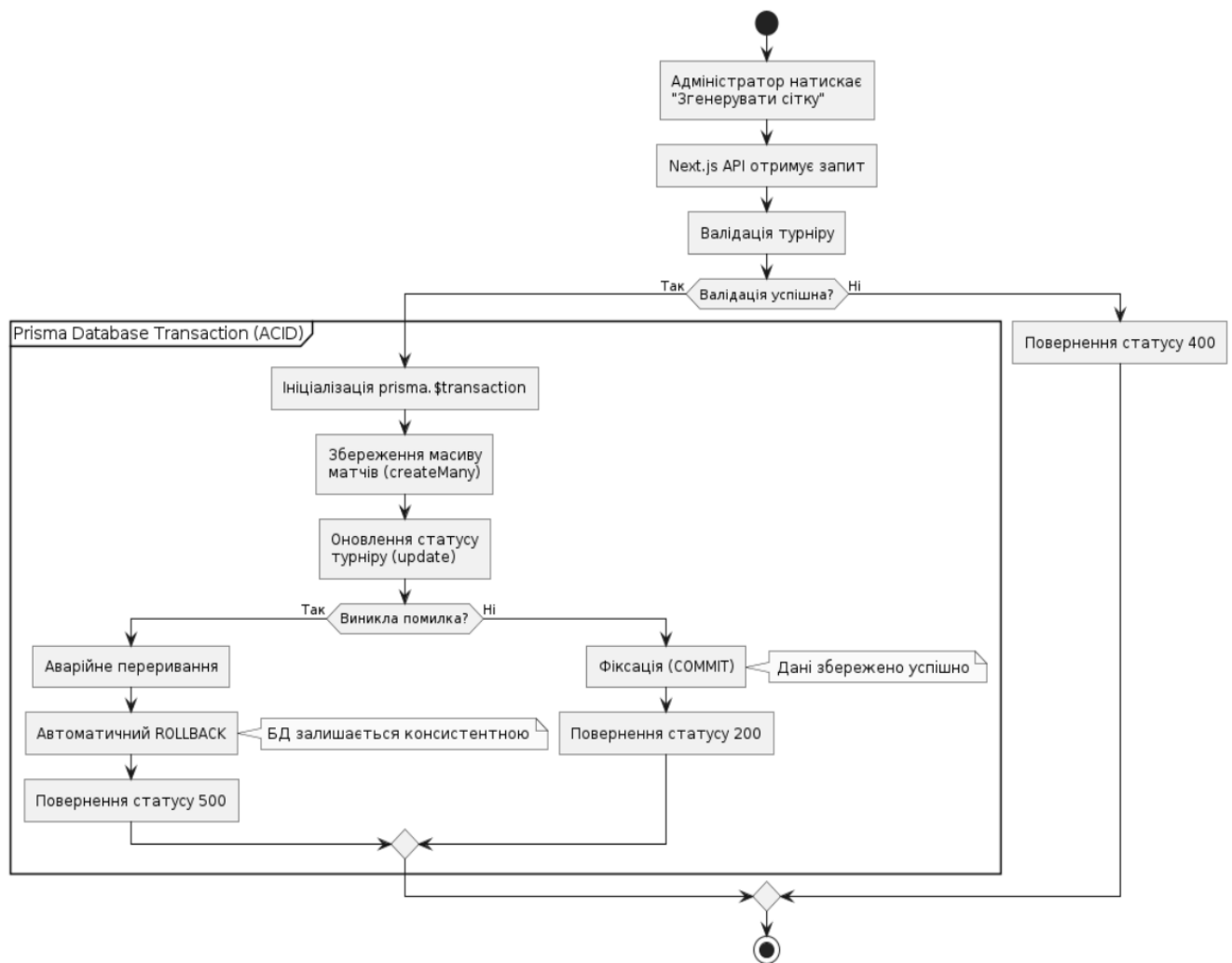


Рис. 2.5 – Блок-схема процесу транзакційного відкоту

У процесі життєвого циклу розробки інформаційної системи структура бази даних зазнає постійних змін, що зумовлено розширенням функціоналу та інтеграцією нових сутностей. Для ефективного управління цим процесом спроектовано комбіновану стратегію синхронізації схем, яка враховує відмінності між локальним середовищем та середовищем промислової експлуатації. Порівняльний аналіз застосованих інструментів міграції наведено у таблиці 2.13.

Порівняльний аналіз стратегій синхронізації схеми бази даних

Критерій оцінки	Етап прототипування (db push)	Промислова експлуатація (migrate dev)
Механізм роботи	Пряма синхронізація локальної схеми без проміжних файлів.	Генерація версіонованих SQL-скриптів на основі різниці схем.
Швидкість ітерацій	Висока, дозволяє миттєво тестувати нові сутності в коді.	Помірна, вимагає рев'ю та підтвердження міграційних файлів.
Збереження даних	Низький рівень безпеки щодо збереження наповнення колонок.	Високий рівень безпеки із попередженням про деструктивні зміни.

На етапі активного проєктування архітектури та ітеративного формування доменної моделі платформи використовувався метод прямої синхронізації схеми бази даних. Такий підхід забезпечував високу швидкість розробки та необхідну структурну гнучкість, оскільки дозволяв миттєво імплементувати нові сутності або змінювати відношення між ними без генерування надлишкових міграційних файлів, які б обтяжували кодову базу на ранніх етапах прототипування. Однак при переході до стадії розгортання на робочих серверах промислової експлуатації та підключенні автоматизованого конвеєра безперервної інтеграції та доставки (CI/CD), стратегія безальтернативно змінюється на використання суворо версіонованих міграцій. Відмова від прямої синхронізації зумовлена її критичною небезпечкою в робочому середовищі через ризик випадкового незворотного видалення таблиць із реальними даними користувачів під час реструктуризації. Впровадження механізму керованих міграцій дозволяє системі аналізувати різницю між поточним та цільовим станами архітектури, генеруючи точні SQL-скрипти мовою визначення даних (DDL). Це забезпечує повну відтворюваність інфраструктури, гарантує безпечну трансформацію існуючих масивів інформації, а також надає розробнику можливість ручного аудиту скриптів перед їх виконанням. Детальний інтеграційний життєвий цикл розгортання відповідних структурних змін бази даних у середовищі безперервної доставки (рис. 2.6).

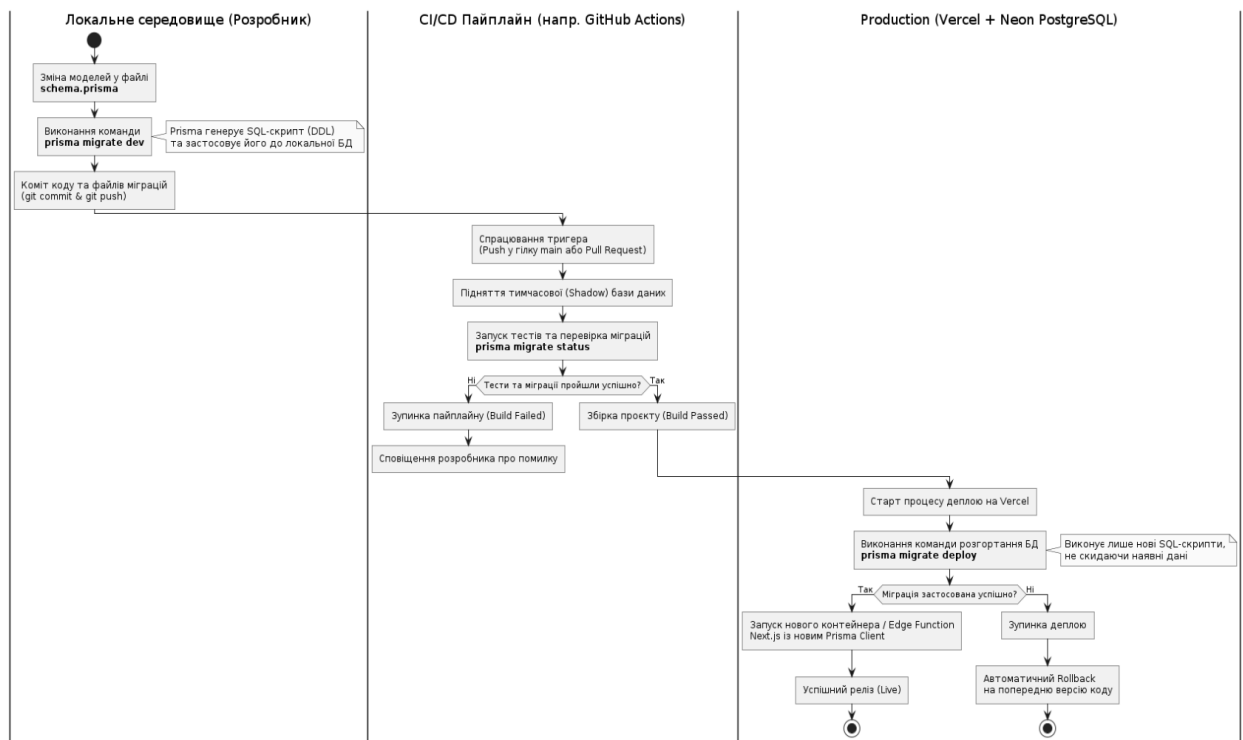


Рис. 2.6 – Життєвий цикл розгортання змін бази даних у CI/CD

Крім класичної синхронної взаємодії між модулями через REST API, архітектура платформи «Footballium» передбачає асинхронну взаємодію в реальному часі. Оскільки базова модель протоколу HTTP вимагає постійного опитування сервера клієнтом, її використання для трансляцій наживо є неефективним і створює надмірне навантаження на інфраструктуру. Для подолання цих обмежень спроектовано подієво-орієнтовану архітектуру на базі захищеного протоколу WebSocket.

Взаємодія між учасниками трансляцій побудована на основі архітектурного патерну публікації та підписки. У цій парадигмі компоненти системи ізольовані один від одного, що забезпечує низьку зв'язність модулів. Роль ініціатора подій бере на себе серверне ядро Next.js. Після успішної валідації дії арбітра та збереження епізоду в базі даних, сервер формує відповідний пакет даних і передає його брокеру повідомлень. Своєю чергою, клієнтські інтерфейсні компоненти, такі як віджети перегляду матчів, виступають у ролі передплатників. Вони програмно ініціюють підписку на конкретний канал зв'язку і реактивно оновлюють об'єктну модель документа при отриманні нових даних. Специфікацію інформаційних потоків підсистеми моніторингу наведено у таблиці 2.14.

Специфікація каналів та подій підсистеми Live-моніторингу

Назва каналу	Назва події	Структура корисного навантаження (Payload)	Ініціатор події	Цільовий слухач події
match-[id]	match-updated	Поточний статус, рахунок команд, хвилина та масив подій	Next.js API (після дії арбітра)	Віджети перегляду матчу та панель арбітра
tournament-[id]	standings-updated	Масив об'єктів оновленої турнірної таблиці	Next.js API (після перерахунку)	Компонент загальної таблиці чемпіонату

Вибір спеціалізованого хмарного сервісу Pusher як проміжного брокера повідомлень зумовлений специфікою розгортання вебплатформи в інфраструктурі безсерверних обчислень. Архітектура застосованого фреймворку працює за принципом короткочасного життєвого циклу обчислювальних вузлів, де кожна функція активується виключно в момент надходження запиту, виконує логіку і миттєво знищується для вивільнення ресурсів. Така парадигма фізично не дозволяє утримувати тривалі та стабільні з'єднання з тисячами браузерів кінцевих користувачів.

З метою розв'язання цієї інженерної проблеми застосовано стратегію делегування інфраструктурного навантаження (Offloading). Відповідно до цього підходу, усі відкриті клієнтські WebSocket-з'єднання утримуються розподіленою мережею стороннього провайдера. Відповідальність основного сервера зводиться до виконання єдиного вихідного запиту до програмного інтерфейсу брокера. Після отримання тригера система автономно здійснює мультиплексування та розсилку подій по всіх підключених сокетах, що гарантує лінійне горизонтальне масштабування підсистеми трансляцій.

Для наочної демонстрації взаємодії всіх описаних вище модулів розроблено діаграму послідовності. Вона описує критичний сценарій функціонування системи: фіксацію взяття воріт арбітром та подальше миттєве оновлення інформації на екранах глядачів без ініціації перезавантаження вебсторінки (рис. 2.7).

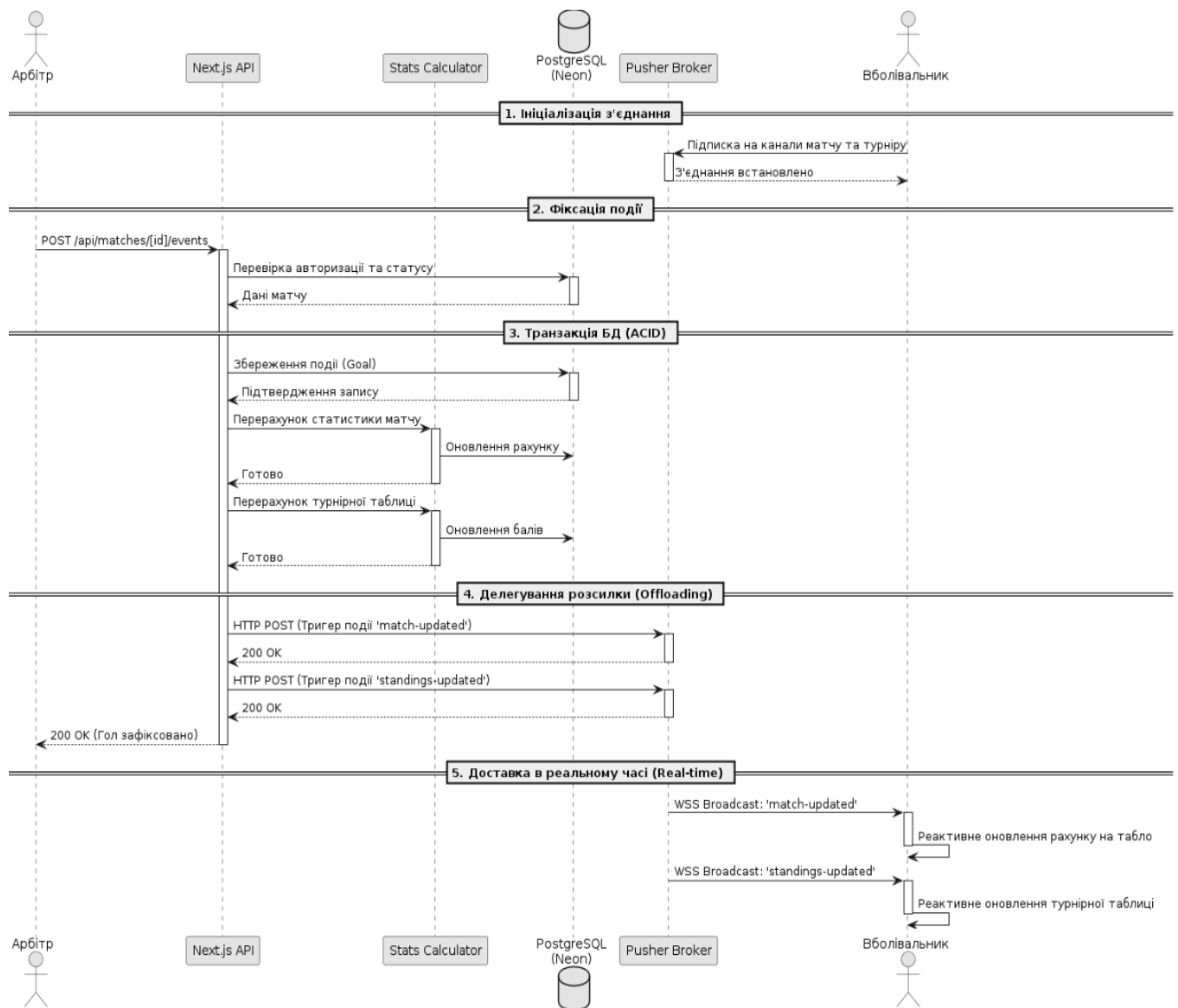


Рис. 2.7 – UML-діаграма послідовності обробки Live-події матчу

Функціонування комплексної спортивної інформаційної системи вимагає суворого детермінованого контролю над життєвим циклом ключових доменних об'єктів. Центальною сутністю платформи є об'єкт турніру, бізнес-логіка якого спроектована на базі математичної моделі скінченного автомата (Finite State Machine). Цей архітектурний підхід інкапсулює правила переходу між станами та унеможливорює виникнення логічних колізій, таких як генерація ігрових подій для неактивного матчу або реєстрація нових команд після фактичного початку змагань. Програмна реалізація життєвого циклу турніру передбачає його суворо послідовний перехід через чотири базові етапи: від початкової фази чернетки до термінального стану повного завершення чемпіонату. Кожен перехід між цими станами ініціюється специфічним тригером, яким може виступати як пряма дія уповноваженого

адміністратора платформи, так і автоматизована системна подія. Будь-яка спроба зміни стану перехоплюється проміжним програмним забезпеченням і обов'язково супроводжується виконанням комплексу суворих валідаційних правил. Додатково, успішна транзакція зміни статусу генерує каскад побічних ефектів на рівні бази даних, які забезпечують синхронізацію пов'язаних сутностей. Детальну матрицю умов, керуючих тригерів та наслідків переходу станів об'єкта турніру, що відображає імплементовану бізнес-логіку системи, наведено у таблиці 2.15.

Таблиця 2.15

Матриця переходів станів об'єкта «Турнір» та тригерів бізнес-логіки

Початковий стан	Тригер зміни стану (Дія)	Умови валідації переходу	Кінцевий стан	Побічні ефекти (Side Effects) у системі
DRAFT (Чернетка)	Натискання кнопки «Опублікувати турнір»	Наявність базових налаштувань регламенту, дат проведення та лімітів кількості учасників.	REGISTRATION	Відкриття публічного доступу до сторінки турніру. Активація функціоналу подачі заявок тренерами.
REGISTRATION (Рєєстрація)	Ініціація процесу «Згенерувати сітку»	Досягнення мінімально необхідної кількості підтверджених команд-учасниць.	ONGOING	Запуск транзакції prisma.\$transaction: масова генерація об'єктів Match, ініціалізація записів TournamentStanding, блокування прийняття нових заявок.
ONGOING (В процесі)	Натискання кнопки «Закрити чемпіонат»	Всі згенеровані ігрові поєдинки турніру набули термінального статусу FINISHED.	FINISHED	Глобальне блокування можливості редагування рахунків. Переведення турнірної таблиці та статистики в режим читання (Read-Only). Архівізація.

Для наочного візуального моделювання динаміки зміни станів, абстрагування складності внутрішніх процесів та кращого розуміння поведінки системи в часі було спроектовано відповідну UML-діаграму (рис. 2.8).

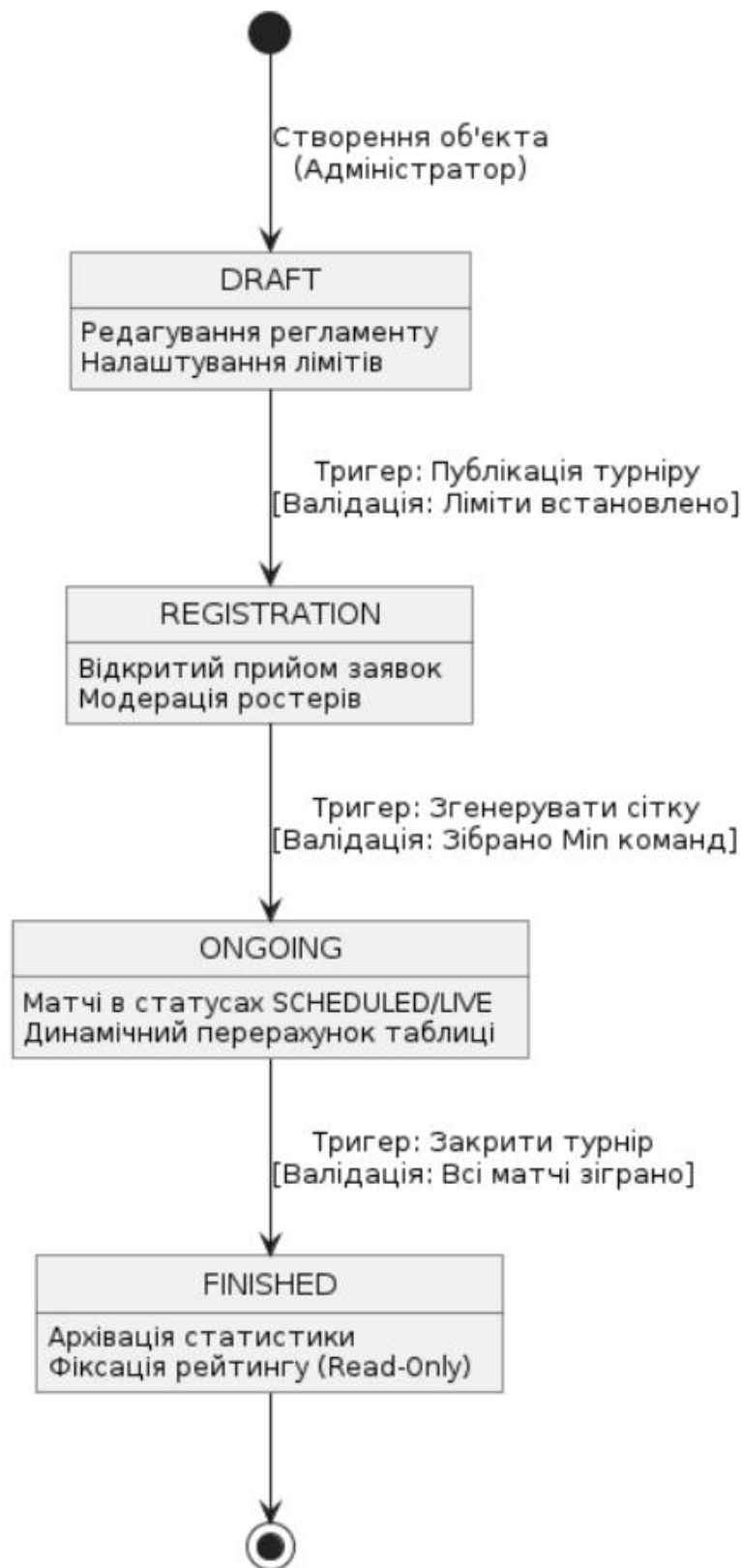


Рис. 2.8: UML-діаграма станів життєвого циклу турніру

2.4 Проєктування та реалізація алгоритмів роботи системи

З метою комплексного розкриття внутрішньої бізнес-логіки створюваної платформи було розроблено алгоритми виконання ключових транзакційних процесів,

які наочно формалізовано за допомогою діаграм активності (Activity Diagrams). Зазначені візуальні моделі детально описують динаміку функціонування інформаційної системи, ілюструючи паралельні обчислювальні потоки, складні логічні розгалуження та суворий розподіл зон відповідальності між клієнтським браузером і серверним ядром через використання механізму функціональних доріжок.

Фундаментальним інженерним модулем розроблюваної системи є алгоритм автоматизованої генерації турнірної сітки. Цей процес відзначається високим рівнем обчислювальної складності, оскільки вимагає проведення багатоетапних комбінаторних розрахунків на серверному рівні задля забезпечення справедливих змагальних умов. Використання автоматизованого математичного підходу, наприклад, алгоритму кругового парування не лише звільняє адміністратора від рутинної роботи з ручного формування розкладу, а й гарантує архітектурну консистентність даних. Завдяки повній ізоляції цієї бізнес-логіки всередині серверного середовища Vercel, система здатна за лічені мілісекунди генерувати сотні об'єктів матчів без ризику перевантаження клієнтського пристрою.

Безпосередній процес ініціюється в момент відправки адміністратором сформованого пакета налаштувань турніру, перелік команд-учасниць, дати проведення, формат змагання. Після отримання мережевого запиту система спрямовує його через жорсткий конвеєр проміжної обробки. На цьому етапі відбувається криптографічна валідація сесії адміністратора через JWT-токен та санітизація вхідних параметрів за допомогою декларативних схем. Одразу після підтвердження легітимності запиту серверне ядро звертається до реляційного кластера бази даних для перевірки статусу обраних команд, унеможливлюючи їхню одночасну участь у конфліктних за часом матчах.

Сформувавши математичну модель майбутнього змагання, алгоритм ініціює масову транзакцію запису. Для збереження абсолютної цілісності інформації створення десятків нових поєдинків та ініціалізація стартових турнірних таблиць для кожного клубу виконується в межах єдиної атомарної операції. Це гарантує, що у випадку непередбачуваного збою мережі жоден фрагмент розкладу не буде

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

збережено частково, що дозволяє уникнути виникнення інформаційних аномалій у базі даних (рис. 2.9).

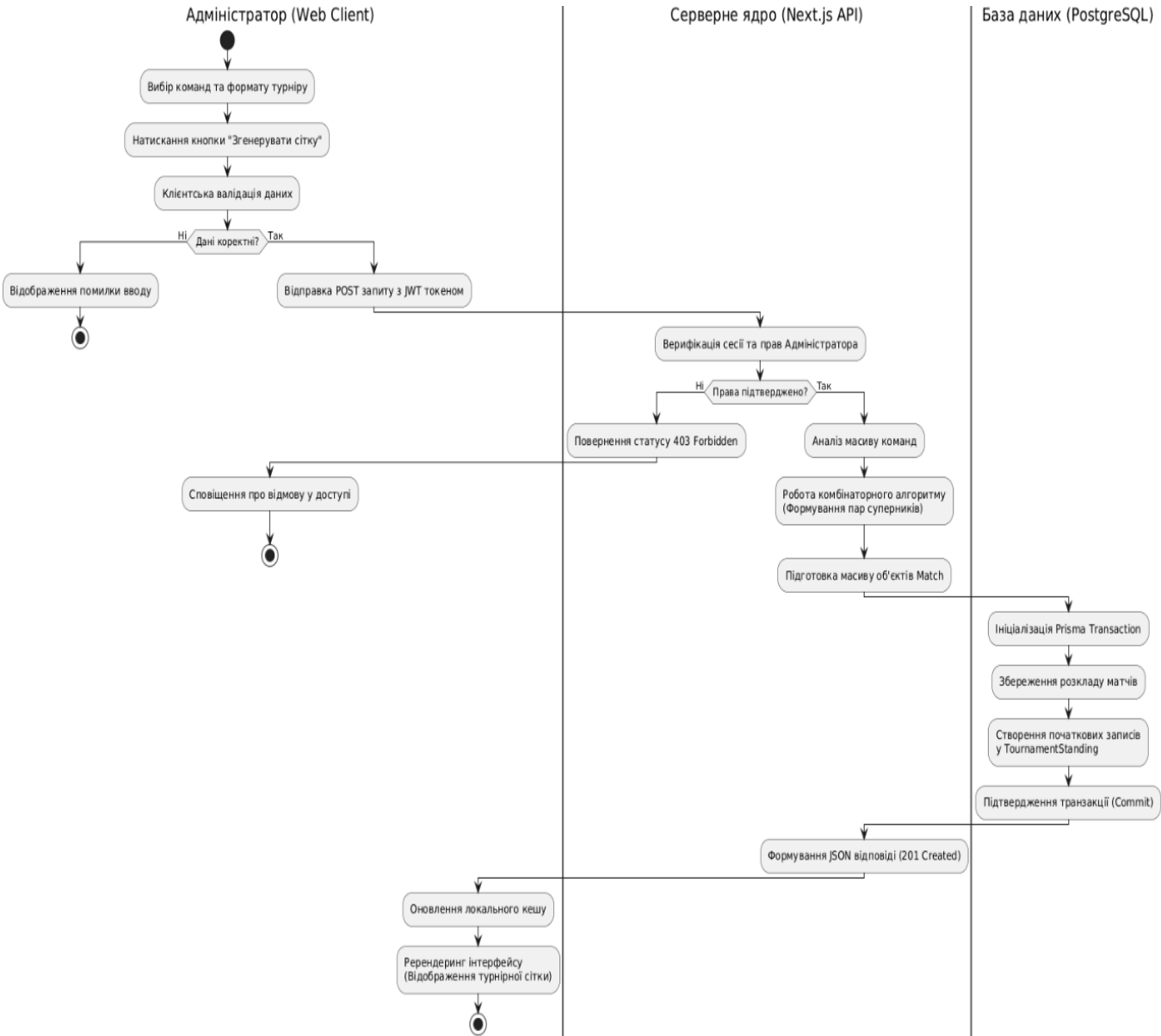


Рис. 2.9 – Діаграма активності автоматичної генерації турнірної сітки

Наступний розроблений алгоритм формалізує строгу послідовність системних операцій під час безпосереднього керування хронологією живого спортивного поєдинку уповноваженим арбітром. Зважаючи на те, що кожна зафіксована результативна подія чи дисциплінарна санкція має миттєвий і незворотний вплив на публічне візуальне табло та подальше формування загальнонаціонального рейтингу учасників змагань, ця процедура вимагає імплементації механізмів безкомпромісного контролю достовірності вхідних даних. З метою оптимізації мережевого трафіку та зменшення навантаження на серверну інфраструктуру, первинний етап

обробки інформації делеговано безпосередньо клієнтському вебзастосунку судді. Саме на рівні браузера або мобільного пристрою виконується базова синтаксична та логічна валідація внесених змін до електронного протоколу, що дозволяє відсіювати хибні транзакції ще до моменту їхнього потрапляння на мережевий шлюз і, як наслідок, критично мінімізує обсяг некоректних або дубльованих звернень до серверного ядра.

Застосований під час проєктування цього модуля архітектурний підхід базується на інноваційній концепції гібридного оновлення стану інформаційної системи. У момент фізичної фіксації ігрового епізоду на полі, клієнтський додаток генерує структурований пакет даних і через асинхронний канал зв'язку направляє його до відповідного API-маршрутизатора. Отримавши транзакційний запит, програмний бекенд першочергово ініціює безпекову перевірку, жорстко верифікуючи криптографічний маркер доступу та підтверджуючи факт офіційного призначення даного судді на поточний матч. У разі успішної авторизації система виконує атомарну операцію збереження події у реляційних таблицях бази даних PostgreSQL, що гарантує цілісність хронологічного таймлайну. Одразу після фіксації транзакції на рівні сховища, серверне середовище активує механізм подієво-орієнтованої архітектури, генеруючи відповідний тригер через інтегрований шлюз WebSockets. Цей технологічний ланцюг забезпечує блискавичну та гарантовану доставку сигналів про зміну ігрового рахунку або інших показників до тисяч підключених пристроїв вболівальників. Отримавши такий push-сигнал, ізольовані клієнтські компоненти фронтенду здійснюють точковий реактивний ререндеринг візуальних віджетів, що створює ефект абсолютної безперервності трансляції без необхідності ініціювання ресурсомісткого повного перезавантаження вебсторінки.

Задля забезпечення абсолютної консистентності та математичної бездоганності агрегованих статистичних масивів, розроблений алгоритм включає надійний асинхронний механізм вирішення потенційних конфліктів та комплексної постобробки результатів. Кульмінаційним етапом цього процесу є реєстрація події фінального свистка, яка сигналізує про завершення регламентованого ігрового часу. Серверне ядро в автономному режимі накладає програмне блокування на

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		54

електронний протокол поєдинку, повністю унеможливлуючи будь-які подальші ручні модифікації чи ретроспективні коригування з боку офіційних осіб. Одночасно з блокуванням ініціюється прихований фоновий процес глобального перерахунку змагальної статистики. Алгоритм автоматично зводить усі проміжні показники: сумує зароблені командами рейтингові очки, вираховує точну різницю забитих та пропущених м'ячів для оновлення загальних турнірних таблиць, а також інтегрує індивідуальні здобутки кожного футболіста до його персонального профілю результативності. Такий підхід дозволяє аналітичній платформі гарантувати стовідсоткову точність поданої інформації. Він виключає ймовірність виникнення людської помилки, яка часто супроводжує ручне зведення підсумкових змагальних протоколів у традиційному спортивному адмініструванні (рис. 2.10).

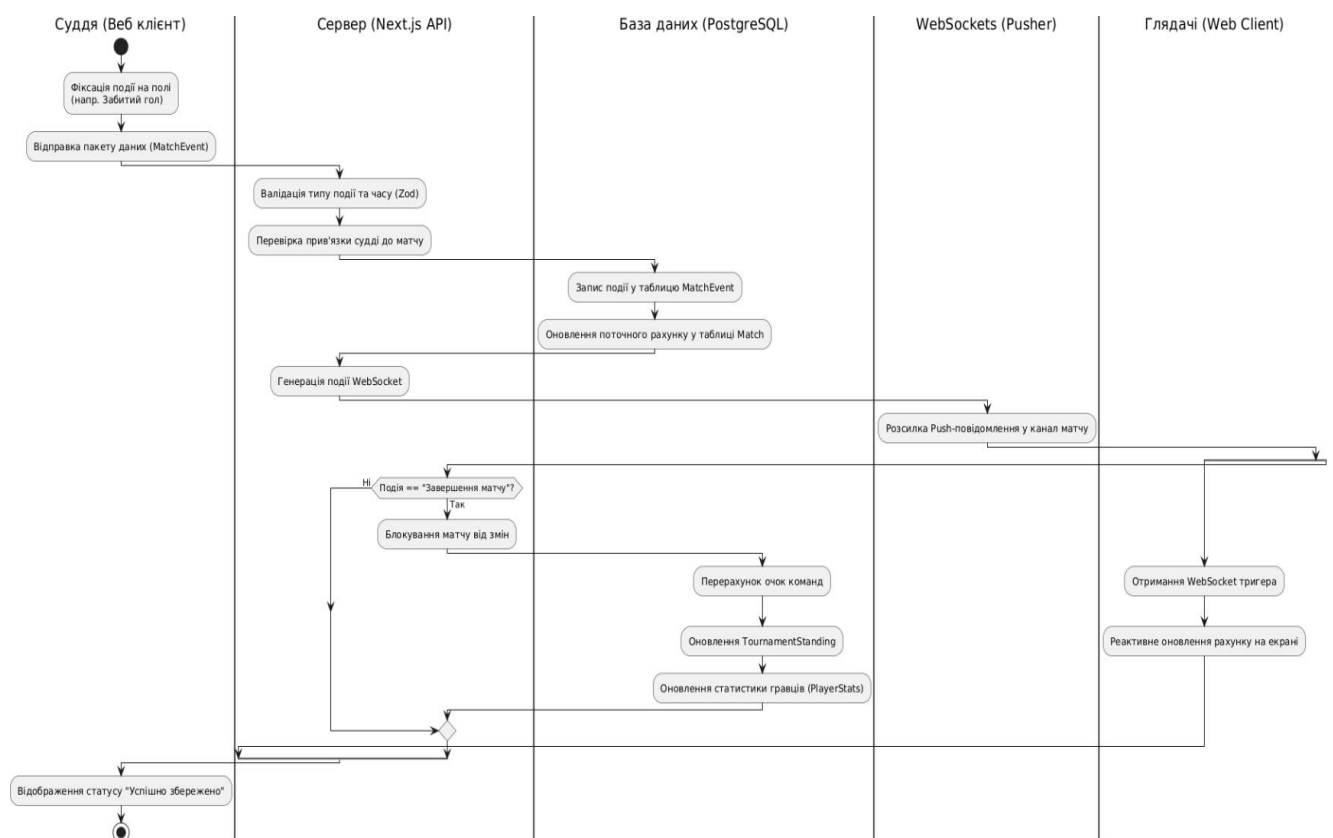


Рис. 2.10 – Діаграма активності Live-події та перерахунку статистики

Для поглибленого розуміння логіки формування статистики та управління турнірами додатково розроблено діаграми послідовності аналітичного дашборда та модерації розкладу (див. Додаток Д та Е).

Висновки до другого розділу

У межах другого розділу дипломної роботи здійснено комплексне інженерне проектування програмної архітектури розроблюваної інформаційної платформи «Footballium». Процес науково-практичного дослідження охопив усі фундаментальні етапи розробки: від формалізації бізнес-вимог та візуального моделювання поведінкових прецедентів користувачів до глибокої деталізації алгоритмів транзакційної обробки поточкових даних. Сформована інфраструктурна модель розгортання довела доцільність використання сучасної безсерверної парадигми (Serverless) на базі середовища Vercel для забезпечення абсолютної відмовостійкості та горизонтального масштабування системи під час пікових навантажень, характерних для живих спортивних трансляцій.

Важливим етапом дослідження стала розробка оптимізованої логічної та фізичної структури реляційної бази даних. Залучення потужного хмарного кластера PostgreSQL платформи Neon у синергії з інструментарієм об'єктно-реляційного відображення Prisma ORM дозволило створити надзвичайно гнучку та надійну модель збереження інформації. Спроектowana схема бази даних успішно поєднує строгі обмеження цілісності для захисту профілів і результатів матчів із гнучкістю для збереження комплексних тактичних розстановок. Такий архітектурний гібрид гарантує здатність вебплатформи до оперативного масштабування та безперешкодної інтеграції нових статистичних метрик у майбутньому без необхідності виконання небезпечних структурних міграцій базового ядра.

З метою ефективної реалізації закладених бізнес-вимог спроектовано надійну об'єктно-орієнтовану систему взаємодії клієнтської та серверної частин вебзастосунку. Імплементация модульної компонентної архітектури на основі фреймворку Next.js та екосистеми React надала можливість незалежно вдосконалювати аналітичні віджети при повному збереженні високої швидкодії інтерфейсу. Окрему інженерну увагу було приділено розробці багаторівневої системи управління станом, яка чітко розмежовує локальні, глобальні, асинхронні та реактивні дані, що дозволило повністю викоринити проблему надлишкових перемальовувань. Паралельно спроектовано захищений комунікаційний шар на базі архітектурного стилю REST

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

API, де надійність кожного мережевого маршруту гарантується суворою типізацією корисного навантаження через декларативні схеми Zod та криптографічною перевіркою сесій за допомогою технології JWT.

На завершальному етапі було детально описано динамічну поведінку вебплатформи шляхом формалізації її ключових алгоритмів за допомогою UML-діаграм активності. Розроблений алгоритм автоматизованої генерації турнірних сіток забезпечує швидке та безпомилкове комбінаторне парування команд із суворим дотриманням усіх регламентних обмежень. Водночас алгоритм обробки ігрових подій реалізує безперебійний високошвидкісний конвеєр доставки живих результатів: від мобільного пристрою арбітра на полі до екранів тисяч вболівальників. Завдяки інтеграції протоколу WebSockets та розробці надійних механізмів фонового перерахунку турнірних таблиць, система гарантує стовідсоткову математичну точність спортивної аналітики та фундаментально виключає фактор людської помилки під час зведення підсумкових змагальних результатів.

У підсумку, проведене у даному розділі комплексне проєктування інфраструктури, бази даних, прикладних інтерфейсів та алгоритмів сформувало міцний інженерний базис платформи. Створена концептуальна та архітектурна документація повністю задовольняє всі початкові функціональні вимоги і формує ідеальне підґрунтя для переходу до наступного етапу — безпосереднього написання програмного коду вебзастосунку та його подальшого тестування.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОЄКТУ

3.1 Вибір програмного середовища та розгортання системи

Процес переведення розробленого програмного коду з локального середовища розробника до стадії публічної комерційної експлуатації реалізовано за допомогою сучасної парадигми безперервної інтеграції та доставки. Аналітична вебплатформа «Footballium» розгортається у розподіленому хмарному середовищі, де шлях від написання кодової бази до появи оновленої версії на робочому сервері є повністю автоматизованим. Розробник звільнений від необхідності виконання будь-яких ручних маніпуляцій для публікації релізу чи налаштування серверних потужностей. Для ініціації оновлення достатньо лише інтегрувати перевірені зміни до основної гілки (main) віддаленого репозиторію. Усю подальшу роботу з компіляції та розподілу трафіку бере на себе передовий конвеєр платформи Vercel.

Програмний комплекс використовує спеціалізовані хмарні служби, детальний перелік яких наведено у таблиці 3.1. Кожна із зазначених платформ відповідає за окремий критичний аспект функціонування системи і здатна горизонтально масштабуватися абсолютно незалежно від решти компонентів загальної інфраструктури.

Таблиця 3.1

Хмарна інфраструктура системи

Служба	Платформа	Призначення
Vercel Cloud Platform	Vercel	Хостинг основного серверного середовища (Node.js) та доставлення клієнтського застосунку.
Neon Serverless Postgres	Neon	Хмарне збереження реляційної спортивної статистики, розкладів турнірів та профілів.
Cloudinary API	Cloudinary	Мережа доставки контенту (CDN) для збереження та оптимізації логотипів команд і фотографій.
Pusher Channels	Pusher	Забезпечення стабільного двостороннього зв'язку через WebSockets для ведення Live-матчів.
GitHub Repository	GitHub	Глобальний реєстр вихідного коду та інтеграція інструментарію безперервного розгортання.

Фундаментальним принципом побудови архітектури безпеки розроблюваної платформи є забезпечення абсолютної інкапсуляції рівня даних. Клієнтські браузерні застосунки функціонують у режимі суворої ізоляції, фізично та логічно унеможливаючи будь-які спроби ініціювання прямих транзакцій від інтерфейсу кінцевого користувача безпосередньо до бази даних. У ролі єдиного легітимного шлюзу для інформаційного обміну виступають захищені серверні маршрути прикладного програмного інтерфейсу (API) екосистеми Next.js. Розгорнутий на високопродуктивних хмарних потужностях Vercel, цей серверний вузол бере на себе критично важливу функцію інтелектуального оркестратора всієї інфраструктури. Безпосередньо через нього здійснюється безпечна маршрутизація складних бізнес-транзакцій до безсерверного кластера реляційної бази Neon, а також паралельне управління процесами асинхронного завантаження, трансформації та розподілу мультимедійних активів у спеціалізованому хмарному сховищі Cloudinary.

Транспортування затвердженого програмного коду до виробничого середовища суворо підпорядковується стандартам методології безперервної доставки (Continuous Delivery) і пролягає виключно через глобальну систему контролю версій Git та репозиторій GitHub [33]. Завдяки нативній інтеграції хмарного провайдера Vercel [15], процедура збирання проєкту ініціюється автоматично. Процес розробки та локального розгортання здійснюється у середовищі Visual Studio Code [32]. Фрагменти реалізації серверної частини та валідаторів наведено у лістингах (див. Додаток Ж). Логіка функціонування цього автоматизованого механізму побудована на принципі нульової толерантності до помилок: виявлення найменших синтаксичних аномалій, провал перевірки суворих статичних типів мови TypeScript або виникнення непередбачуваних винятків під час етапу компіляції миттєво генерує сигнал тривоги та превентивно блокує подальший процес розгортання. Такий бар'єрний підхід слугує надійним інженерним щитом, який фізично захищає живе робоче середовище спортивного порталу від проникнення нестабільного коду чи виникнення критичних збоїв на стороні кінцевого споживача. Детальну специфікацію функціональних етапів описаного конвеєра наведено у таблиці 3.2.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

Стадії конвеєра безперервного розгортання (CI/CD)

Стадія	Команда / Інструмент	Зміст та опис завдання
Встановлення	npm install	Завантаження NPM-пакетів та ефективне кешування модулів залежностей.
Генерація ORM	prisma generate	Створення типізованого клієнта для безпечної взаємодії з базою даних.
Збирання	next build	Компіляція статичних файлів та оптимізація застосунку через компілятор Next.js.
Розгортання	Vercel Edge Network	Публікація оновленої версії вебзастосунку на глобальній мережі CDN без зупинки порталу.

Критичним аспектом забезпечення інформаційної безпеки розроблюваної платформи на етапі її глобального розгортання є процедура безпомилкового мапіngu змінних середовища. Саме цей конфігураційний шар відповідає за встановлення безпечних авторизованих з'єднань між серверним ядром та зовнішніми мікросервісами, гарантуючи коректну маршрутизацію бекенд-запитів. Згідно із сучасними стандартами кібербезпеки та методологією розробки надійних вебдодатків, в архітектурі проєкту діє імперативна заборона на збереження будь-яких чутливих параметрів інфраструктури безпосередньо у вихідному коді віддаленого репозиторію. Усі ключі доступу, ідентифікатори підключень до реляційних баз даних та цифрові підписи токенів надійно ізолюються від загального інформаційного простору.

Вони імпортуються до системи виключно у вигляді зашифрованих криптографічних секретів, управління якими здійснюється через закритий адміністративний інтерфейс хмарного провайдера Vercel. Інтегровані механізми безпеки цієї платформи забезпечують глибокий рівень захисту конфігурацій навіть на динамічній стадії виконання скриптів збирання: будь-які конфіденційні значення автоматично маскуються та обфускуються рушієм хмарного оркестратора. Це фундаментально унеможливорює випадковий витік секретних даних чи їхнє несанкціоноване відображення у відкритих журналах логування конвеєра, навіть у разі ініціації розробником команд прямого виведення інформації у консоль. Вичерпний перелік та архітектурне призначення ключових криптографічних секретів виробничого середовища систематизовано у таблиці 3.3.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Критичні параметри конфігурації виробничого середовища

Параметр (Ключ)	Призначення в архітектурі системи
DATABASE_URL	Зашифрований рядок підключення до безсерверного кластера реляційної бази даних Neon Postgres.
NEXTAUTH_SECRET	Криптографічний ключ для генерації підписів безпеки та валідації JWT-сесій користувачів.
CLOUDINARY_URL	Унікальний ідентифікатор доступу до хмарного сервісу завантаження та оптимізації медіафайлів.
PUSHER_SECRET	Приватний ключ автентифікації серверного ядра для ініціалізації WebSocket-з'єднань.

З метою забезпечення абсолютної автономності циклу програмування та надання можливості проведення глибокого ізольованого тестування інфраструктурних компонентів, архітектура системи передбачає здатність до повноцінного запуску на персональній робочій станції інженера без необхідності підтримки безперервного мережевого з'єднання з глобальним хмарним хостингом. Для ініціалізації локального робочого середовища розробнику достатньо сформулювати конфігураційний файл оточення, що інкапсулює необхідні приховані змінні, та виконати базову консольну команду старту безпосередньо в кореневій директорії проєкту. Особливою гнучкістю відзначається підсистема взаємодії з базою даних на етапі локального проєктування. Залежно від поточних інженерних потреб, маршрутизація транзакційних запитів може бути спрямована як до відокремленої безпечної тестової гілки безсерверного кластера Neon, що дозволяє працювати з віддаленою копією робочої схеми, так і до абсолютно незалежного, віртуалізованого середовища збереження інформації, розгорнутого локально за допомогою контейнерного інструментарію Docker. Такий дуалістичний підхід гарантує безперебійність процесу розробки навіть за умов нестабільного інтернет-з'єднання або тимчасової недоступності зовнішніх провайдерів.

Невіддільною складовою процедури локального розгортання є виконання спеціалізованих автоматизованих скриптів, призначених для первинного

наповнення порожніх реляційних таблиць бази даних структурованим тестовим контентом. Цей критично важливий процес ініціалізації програмного стану реалізується за допомогою потужного інструментарію об'єктно-реляційного відображення Prisma, а саме через механізм алгоритмічного посіву даних (Prisma Seed). Під час запуску розроблений скрипт автономно генерує необхідну ієрархію тестових облікових записів із різними рівнями доступу, створює деталізовані інформаційні профілі футбольних команд та математично симулює розгорнуту хронологічну історію минулих турнірних протистоянь. Одразу після успішного завершення процедури формування початкових колекцій транзакційних записів, клієнтський вебзастосунок спортивного порталу стає повною мірою доступним для інтерактивної взаємодії за адресою відкритого локального мережевого порту. Імплементация такого комплексного інженерного рішення дозволяє перетворити звичайний комп'ютер розробника на абсолютно точну логічну, структурну та поведінкову копію робочого хмарного сервера. Завдяки досягненню бездоганної паритетності середовищ фундаментально усувається ризик появи "плаваючих" помилок та критичних розбіжностей, повністю нівелюючи вірогідність виникнення конфліктів при перенесенні готового програмного коду від етапу локальної розробки до фінальної виробничої інфраструктури хмарної платформи.

3.2 Опис користувацького інтерфейсу та керівництво користувача

Інфраструктура клієнтської частини аналітичної платформи Footballium включає повноцінний адаптивний вебзастосунок, спроектований з урахуванням передових стандартів взаємодії з користувачем (UI/UX). Візуальна оболонка системи побудована на базі компонентного підходу із застосуванням сучасного утилітарного фреймворку Tailwind CSS. Головна сторінка порталу виконує потужну презентаційну та навігаційну функції, пропонуючи відвідувачам сучасний мінімалістичний дизайн із темною темою та яскравими неоновими акцентами. Перший екран (рис. 3.1) містить масивну типографіку із мотивуючим заголовком, глобальне навігаційне меню для доступу до реєстрів команд чи гравців, а також інтерактивні точки входу до системи: кнопки ініціалізації процесу реєстрації та пошуку змагань.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

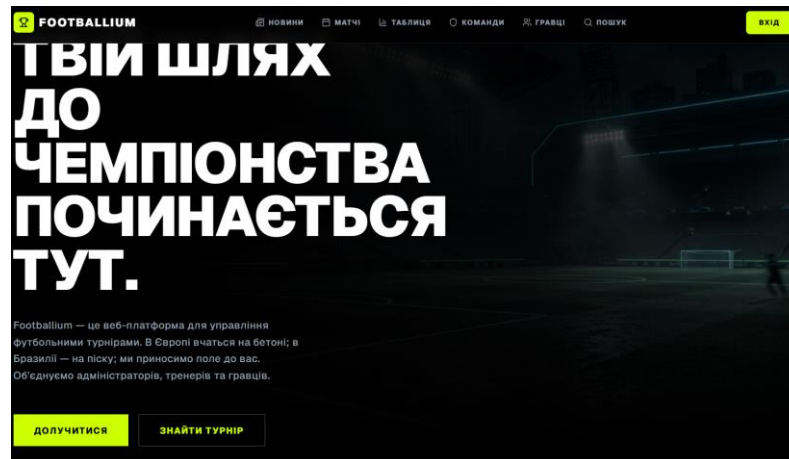


Рис. 3.1 – Головний екран привітання та навігації платформи Footballium

Під час прокручування сторінки донизу розкривається наступний функціональний блок, який відповідає за оперативне інформування аудиторії (рис. 3.2). Користувач отримує доступ до інтерактивного каталогу найближчих турнірів. Картки змагань спроектовано з акцентом на ключові метадані: дату проведення, максимальну кількість команд-учасниць та формат турніру. Для зручності орієнтування кожна картка обладнана динамічним індикатором статусу, що одразу інформує тренера про можливість подачі заявки. Безпосередньо під каталогом турнірів розташовано стрічку останніх новин, яка формується адміністрацією ресурсу та забезпечує аудиторію актуальними підсумками сезонів.

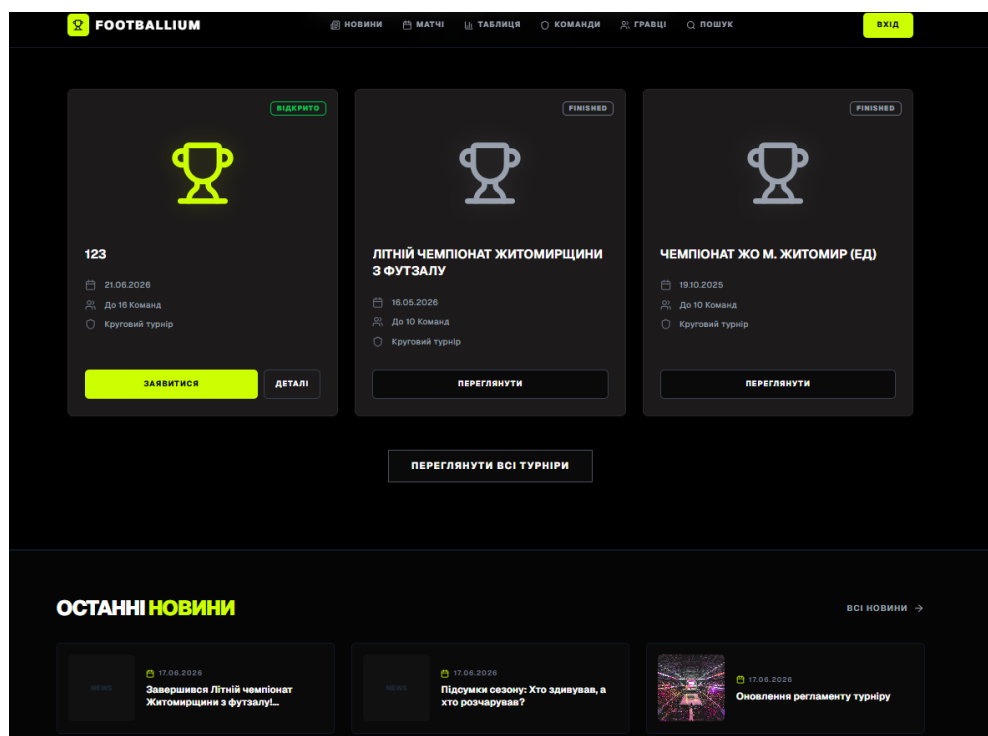


Рис. 3.2 – Інтерфейс каталогу найближчих турнірів та стрічки новин

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		63

Архітектура користувацького інтерфейсу функціонує на основі складної рольової моделі доступу, яка динамічно адаптує візуальний простір під потреби конкретного актора. Після успішної авторизації базовий профіль «Користувач» отримує розширені можливості для глибокої персоналізації свого інформаційного простору. В особистому кабінеті вболівальника розміщено спеціалізовану навігаційну панель, через яку здійснюється перехід до управління конфіденційними даними. У розділі «Мій профіль» (рис. 3.3) користувач має змогу редагувати особисту інформацію, зокрема контактний номер телефону та публічне ім'я, а також ініціювати процедуру зміни пароля безпеки. Дизайн цього модуля спроектовано з акцентом на мінімалізм: поля вводу оснащені механізмами локальної валідації, а поточний рівень привілеїв чітко ідентифікується за допомогою візуального бейджа (наприклад, статусу «USER»).

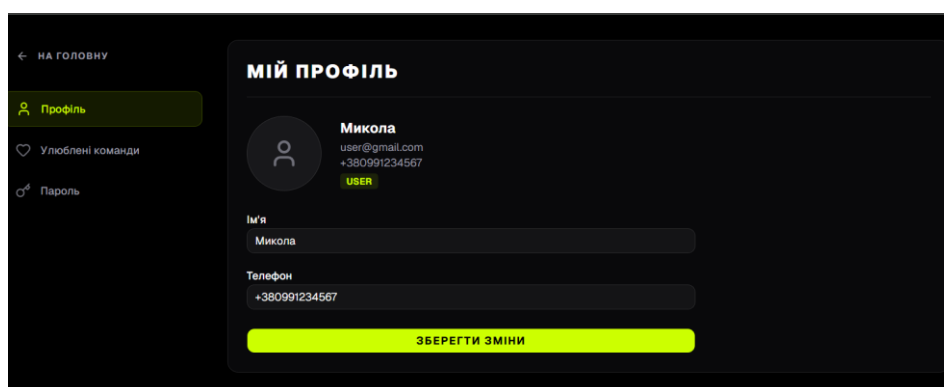


Рис. 3.3 – Інтерфейс особистого кабінету користувача та налаштувань профілю

Ключовою особливістю базового профілю є наявність модуля персоналізації спортивного контенту через механізм відстеження клубів. Під час перегляду глобального каталогу команд або сторінки детальної статистики конкретного колективу (рис. 3.4), де відображаються показники перемог, поразок, список топ-бомбардирів та загальний ростер гравців, користувач може скористатися тригером «Додати в улюблені». Ця дія асинхронно зберігає зв'язок у реляційній базі даних та автоматично додає клуб до спеціального розділу «Улюблені команди» в особистому кабінеті (рис. 3.5). Наявність такого списку ініціює алгоритми персоналізованої видачі результатів, що дозволяє системі пріоритетно відображати розклади

матчів обраних колективів і суттєво економить час вболівальника на пошук релевантної інформації. У разі втрати інтересу до клубу, користувач може миттєво видалити його з переліку за допомогою відповідної іконки керування, що одразу оновить стан інтерфейсу без перезавантаження сторінки.

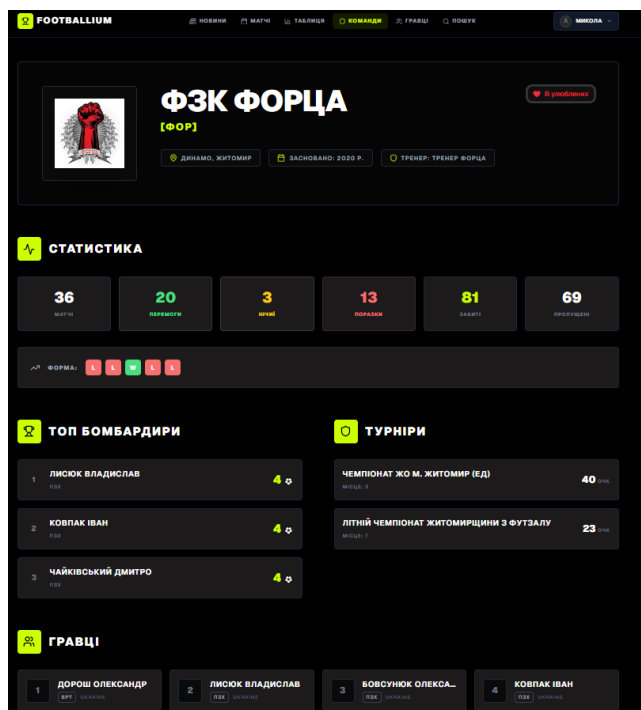


Рис. 3.4 – Аналітична панель команди з статистикою та гравцями

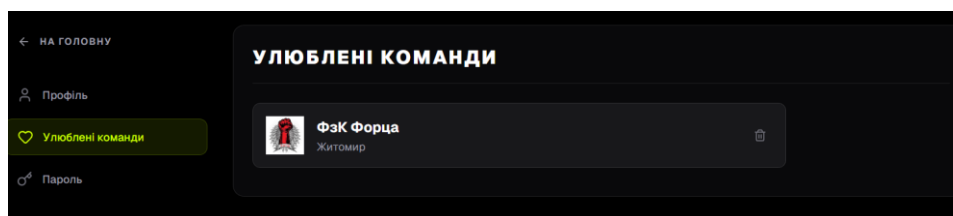


Рис. 3.5 – Розділ управління улюбленими командами в особистому кабінеті

Окрім персонального простору, користувач отримує безперешкодний доступ до глобальних аналітичних модулів платформи, які забезпечують глибокий моніторинг змагального процесу. Розділ «Турнірна таблиця» (рис. 3.6) пропонує структуроване зведення результатів чемпіонату, динамічно розраховуючи позиції клубів на основі здобутих перемог, нічий, поразок та різниці забитих і пропущених м'ячів (ЗГ/ПГ). Для відстеження поточного прогресу функціонує «Центр матчів» (рис. 3.7), інтерфейс якого агрегує інформацію про завершені поєдинки із зазначенням фінального рахунку та геолокації проведення гри (наприклад, стадіону чи спортивного майданчика).

FOOTBALLIUM НОВИНИ МАТЧІ ТАБЛИЦЯ КОМАНДИ ГРАВЦІ ПОШУК МІКОЛА									
ТУРНІРНА ТАБЛИЦЯ ЛІТНІЙ ЧЕМПІОНАТ ЖИТОМИРСЬКОГО РЕГІОНУ З ФУТЗАЛУ Літній чемпіонат Житомирщини з футзала									
ПОЗ	КЛУБ	І	В	Н	П	ЗГ	ПГ	РГ	О
1	ЖИТОМИРСЬКА ПОЛТЕХНІКА	18	8	7	3	45	38	7	31
2	INVIVO	18	9	4	5	49	43	6	31
3	ЖВІ ІМ. КОРОЛЬОВА	18	9	2	7	53	39	14	29
4	QOOBIX	18	7	5	6	44	41	3	26
5	ФК "РЯТІВНИК"	18	7	4	7	39	44	-5	25
6	ФК "ГУНП-ІВАНКІВ" (ЖИТОМИР)	18	6	5	7	47	49	-2	23
7	ФЗК ФОРЦА	18	7	2	9	28	35	-7	23
8	ФК "VIVAD" (РОМАНІВ)	18	6	4	8	44	47	-3	22
9	ЕНЕРГІЯ	18	4	9	5	50	50	0	21
10	СК "ФОРСАЖ"	18	5	2	11	41	54	-13	17
ОСТАННЄ ОНОВЛЕННЯ: 17:41:11									

Рис. 3.6 – Інтерфейс інтерактивної турнірної таблиці змагань

FOOTBALLIUM НОВИНИ МАТЧІ ТАБЛИЦЯ КОМАНДИ ГРАВЦІ ПОШУК МІКОЛА									
ЦЕНТР МАТЧІВ РЕЗУЛЬТАТИ МАТЧІВ ОНЛАЙН ТА РОЗКЛАД									
РЕЗУЛЬТАТИ									
ТУР 18									
ЗАВЕРШЕНО ЛІТНІЙ МАЙДАНЧИК ЖИТОМИР ФРС 5 - 2 ЕНЕ ЦЕНТР МАТЧУ >									
ЗАВЕРШЕНО ФОК ЖИТОМИР ЖВІ 3 - 3 ГІВ ЦЕНТР МАТЧУ >									
ЗАВЕРШЕНО ФОК ЖИТОМИР 4 - 1									
ЗАВЕРШЕНО ФОК ЖИТОМИР 1 - 0									

Рис. 3.7 – Модуль «Центр матчів» для моніторингу результатів поєдинків

Для забезпечення високого рівня соціальної залученості та підтримки інформаційної активності спільноти реалізовано повноцінний новинний модуль (рис. 3.8). Користувач отримує можливість не лише споживати актуальний контент щодо оновлень регламенту чи підсумків сезону, але й активно взаємодіяти з ним. Система дозволяє авторизованим особам залишати електронні вподобання (лайки) під публікаціями у стрічці новин. Кожна така взаємодія фіксується у базі даних і

миттєво оновлює лічильники переглядів та реакцій, що у підсумку формує органічні рейтинги популярності статей та допомагає адміністрації аналізувати залученість аудиторії.

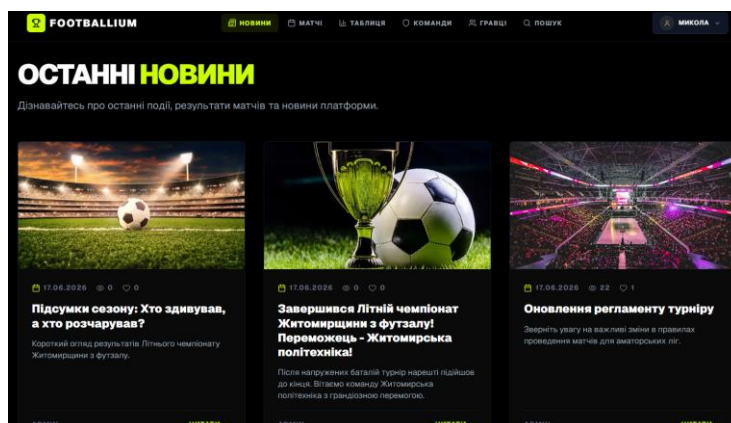


Рис. 3.8 – Стрічка останніх новин із модулем соціальної взаємодії

Обліковий запис із рівнем привілеїв «Тренер» пропонує кардинально трансформований візуальний інтерфейс, який перетворюється на спеціалізований інструмент електронного спортивного менеджменту. З метою збереження об'єктивності та фокусування на професійних обов'язках, цей профіль алгоритмічно позбавлено базового функціонала додавання чужих улюблених команд. Головною точкою входу до робочого середовища виступає загальна «Панель тренера» (рис. 3.9).

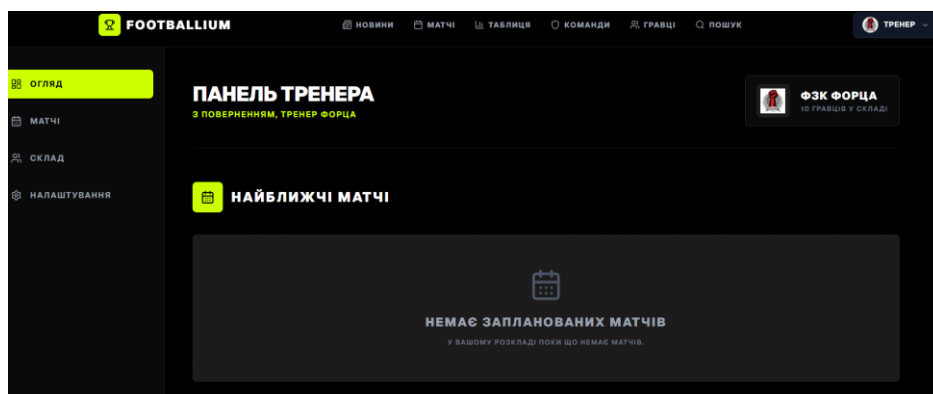


Рис. 3.9 – Головна панель тренера з віджетом найближчих поєдинків

Для глибшого аналізу розкладу розроблено розширену систему моніторингу матчів (рис. 3.10), яка дозволяє тренеру зручно орієнтуватися у змагальному календарі, безперешкодно перемикаючись між результатами минулих ігор та майбутніми протистояннями.

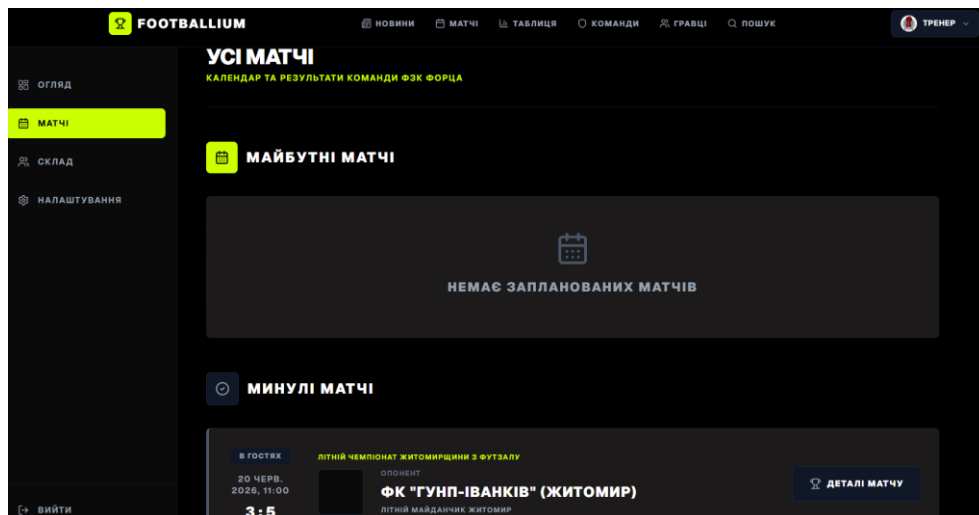


Рис. 3.10 – Розділ моніторингу минулих та запланованих матчів команди

Ключовий акцент тренерського інтерфейсу зміщено на глибоке адміністрування власного колективу. В ізольованому розділі управління тренер отримує повний архітектурний контроль над цифровим профілем клубу. Безпосереднє управління ростером здійснюється через окремий модуль керування складом (рис. 3.11), де кожен спортсмен репрезентований у вигляді окремої інформаційної картки із зазначенням ігрового номера та амплуа. Цей інтерфейс підтримує швидке додавання нових футболістів, редагування існуючих профілів та їхнє видалення, що гарантує підтримку клубної бази даних у завжди актуальному стані.

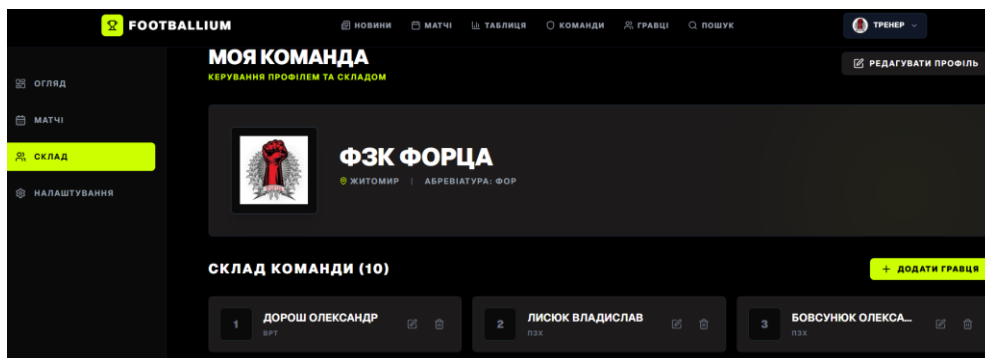


Рис. 3.11 – Робочий простір тренера для управління ростером та гравцями

Важливим елементом архітектури платформи є модуль турнірної взаємодії, через який реалізується офіційна комунікація між спортивним клубом та адміністрацією ресурсу. Для участі у вибраних змаганнях тренер використовує сторінку інформації про турнір (рис. 3.12) для подачі цифрових заявок. Інтерфейс цього модуля містить спеціальний блок статусу, який у режимі реального часу відображає поточний стан модерації заявки (наприклад, індикатор «НА РОЗГЛЯДІ») та

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволяє відкликати її у разі зміни організаційних планів. Безпосередньо перед початком кожного затвердженого календарного поєдинку алгоритми платформи відкривають тренеру доступ до конструктора, де він зобов'язаний заповнити офіційну заявку на матч.

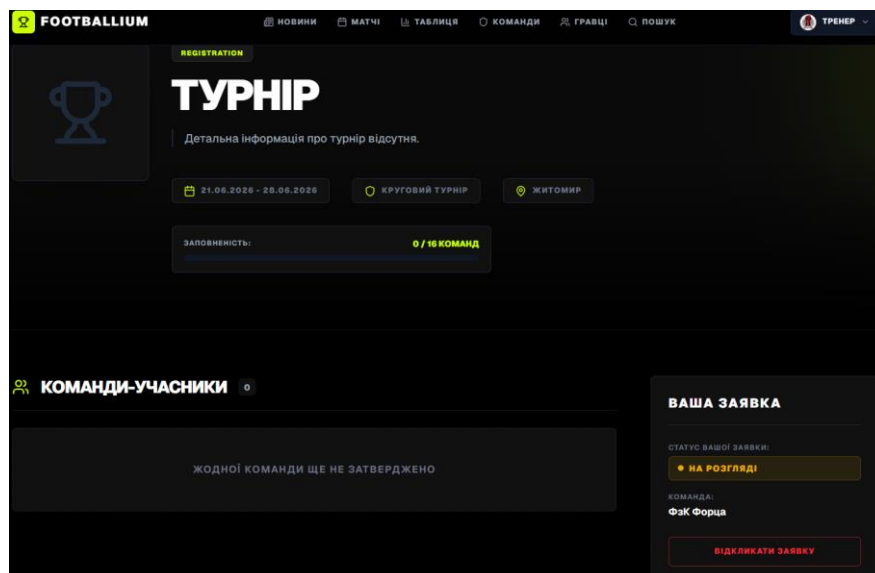


Рис. 3.12 – Модуль управління турнірними заявками

Специфіка професійної діяльності офіційних осіб матчу вимагає наявності максимально оптимізованого та ергономічного інтерфейсу, тому для облікових записів із системною роллю «Суддя» спроектовано спеціалізований аналітичний дашборд. У персональному робочому кабінеті арбітра (рис. 3.13) реалізовано чітке візуальне та логічне розмежування інформаційного простору. Таке архітектурне рішення дозволяє судді безперешкодно орієнтуватися у власному графіку та завчасно готуватися до обслуговування майбутніх спортивних подій.

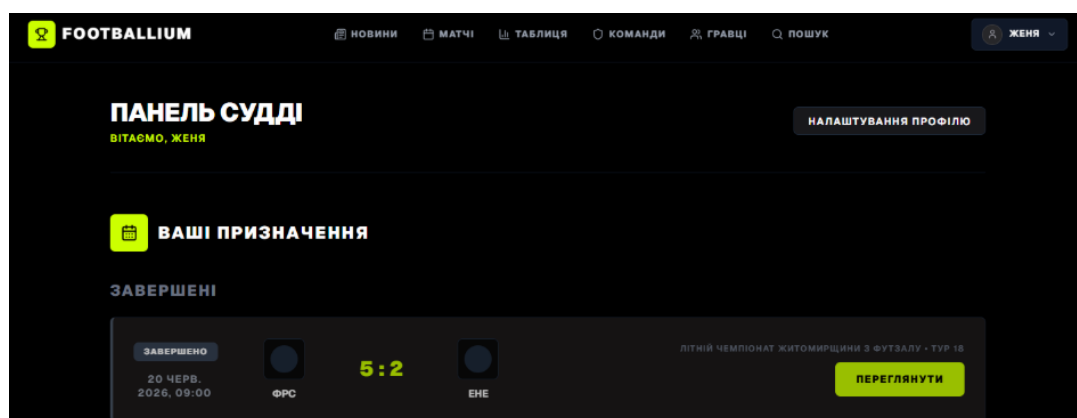


Рис. 3.13 – Головна аналітична панель судді з історією призначень на матчі

Кульмінаційним елементом функціоналу суддівського профілю виступає модуль безпосереднього Live-моніторингу та управління ігровим процесом (рис. 3.14). За допомогою цієї панелі суддя може миттєво класифікувати епізод: зафіксувати взяття воріт, автогол, призначити пенальті, винести дисциплінарну санкцію у вигляді жовтої чи червоної картки, або провести офіційну тактичну заміну.

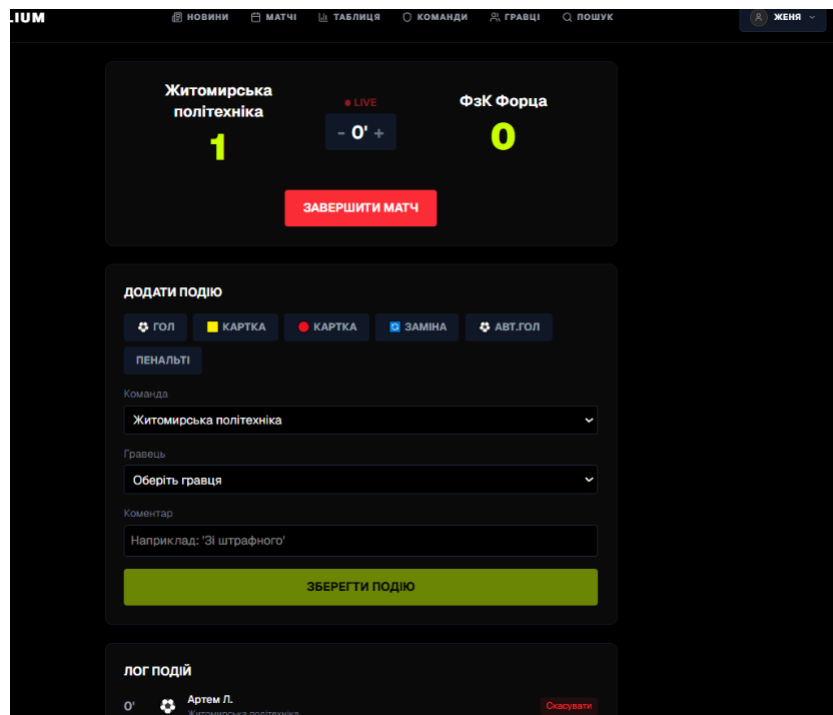


Рис. 3.14 – Інтерфейс панелі арбітра для фіксації подій у режимі реального часу. Глобальне керування та архітектурний контроль над інформаційною системою здійснюються через закриту панель головного адміністратора. Головний екран адміністративного дашборда (рис. 3.15) спроектовано як комплексну аналітичну панель, що агрегує ключові метрики життєдіяльності платформи.

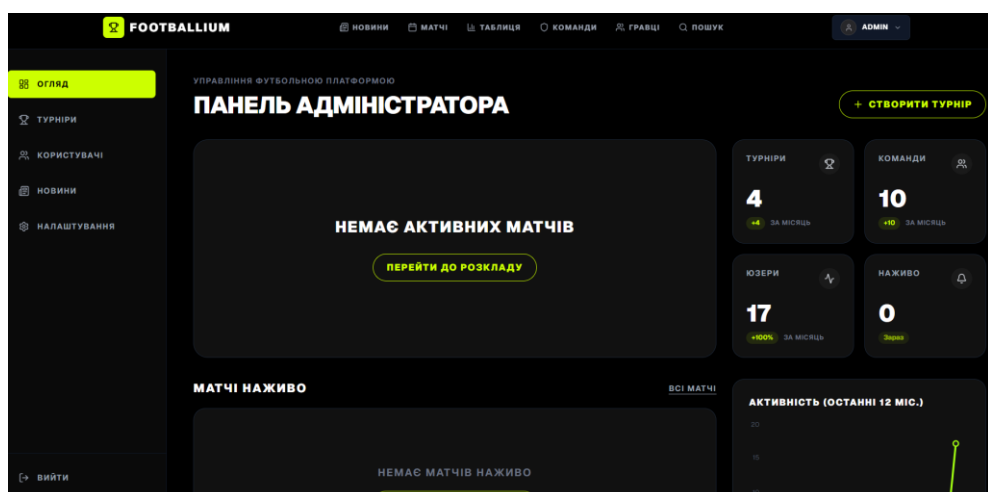


Рис. 3.15 – Глобальна оглядова панель управління головного адміністратора

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

Окремим функціональним блоком системи виступає модуль комплексного управління аудиторією. Через цей інтерфейс адміністратор отримує права на глибоке редагування профілів, зміну ієрархічних ролей, а також можливість превентивно обмежувати доступ порушників до інформаційного ресурсу. Для підтримки високого рівня залученості спільноти та інформування вболівальників у платформу інтегровано повноцінну підсистему управління контентом (рис. 3.16).

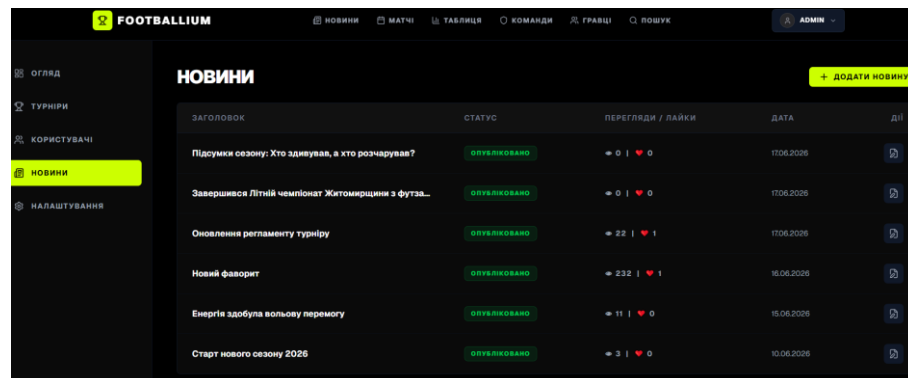


Рис. 3.16 – Модуль управління контентом та стрічкою спортивних новин

3.3 Тестування роботи програмного продукту

Невіддільним етапом життєвого циклу розробки надійного програмного забезпечення є стадія комплексного аудиту та верифікації [31]. Для забезпечення безперебійної роботи розробленої веб-орієнтованої системи управління футбольними турнірами було спроектовано та впроваджено багаторівневу стратегію тестування. Ця стратегія охоплює модульне, інтеграційне та наскрізне тестування, що дозволяє перевіряти систему як на рівні окремих функцій, так і на рівні комплексних користувацьких сценаріїв. Як фундаментальний інструментарій для запуску тестових сценаріїв та перевірки тверджень було обрано фреймворк Jest [16]. У синергії з бібліотекою React Testing Library [17] він дозволив реалізувати глибоке модульне тестування клієнтських компонентів, а інтеграція з тестовим клієнтом Prisma [18] забезпечила верифікацію роботи бази даних. Такий підхід гарантує високий рівень відмовостійкості системи під час обробки динамічних спортивних даних.

Стратегія верифікації на рівні доменної бізнес-логіки фокусується на тестуванні алгоритмів, які не залежать від зовнішніх сервісів та інтерфейсів прикладного програмування. Основна інженерна увага була приділена модулям, що відповідають за ключовий функціонал системи. Зокрема, успішно розроблено та пройдено

набори тестів для файлу генерації турнірних сіток (brackets.ts). У цьому сценарії перевірялася коректність комбінаторного парування команд за алгоритмами стадії плей-оф, а також коректність обробки крайових випадків. Іншим компонентом є модуль математичного розрахунку статистики (stats-calculator.ts), який відповідає за агрегацію турнірних очок та індивідуальних показників гравців. Тестування цих модулів ізольовано від серверного оточення дозволило досягти високої швидкості виконання тестів та гарантувати правильність математичних обчислень.

Інтеграційне тестування серверної частини було зосереджено на перевірці маршрутизаторів. Зокрема, для маршруту управління командами (teams.route.ts) автоматизовані скрипти симулювали запити до відповідних ендпоінтів. Під час тестування верифікувалася коректність структури повернутих даних у форматі JSON, правильність обробки статус-кодів та взаємодія з мокованим клієнтом бази даних. На рівні клієнтського інтерфейсу застосування React Testing Library дозволило програмно змонтувати ізольовані віджети, такі як кнопка додавання клубу до улюблених (FavoriteButton.tsx) та модуль відстеження переглядів (ViewTracker.tsx). Тести перевіряли реакцію компонентів на взаємодію з користувачем, зміну візуального стану після натискання та коректність викликів функцій зворотного зв'язку без повного рендерингу сторінки у браузері. (рис. 3.17).

```
PS F:\Football\Footballium> npx jest
PASS  __tests__/_domain/brackets.test.ts
PASS  __tests__/_domain/stats-calculator.test.ts
PASS  __tests__/_api/teams.route.test.ts
PASS  __tests__/_client/ViewTracker.test.tsx
PASS  __tests__/_client/FavoriteButton.test.tsx

Test Suites: 5 passed, 5 total
Tests:      8 passed, 8 total
Snapshots:  0 total
Time:       2.973 s
Ran all test suites.
```

Рис. 3.17 – Результати виконання тестів у середовищі Jest

Окрім модульного та інтеграційного рівнів, критично важливим етапом перевірки якості програмного продукту є наскрізне тестування (End-to-End, E2E), яке симулює реальну поведінку користувача у браузері. Для реалізації цього рівня автоматизації було інтегровано фреймворк Playwright. Одним із ключових сценаріїв,

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

що підлягав ретельному тестуванню, став процес взаємодії користувача з системою відображення та бронювання квитків на матчі. Автоматизований скрипт відтворює повний шлях користувача: від переходу на сторінку конкретної спортивної події, взаємодії з елементами графічного інтерфейсу для вибору місць, до підтвердження дій. Під час розробки цих тестів окрема увага приділялася налаштуванню динамічних очікувань (timeouts) для асинхронних операцій та правильній ідентифікації локаторів DOM-елементів після рендерингу сторінки сервером. Це дозволило переконатися, що основний бізнес-процес системи працює безвідмовно в умовах, максимально наближених до реальних.

Для об'єктивної кількісної оцінки якості розробленого програмного забезпечення та визначення рівня його перевірки було згенеровано деталізований звіт щодо покриття коду тестами. Як видно зі звіту, згенерованого за допомогою вбудованих інструментів середовища (рис. 3.18), загальний відсоток покриття інструкцій по всіх протестованих файлах становить 43.85%, що є базовим показником, який забезпечує перевірку найбільш критичних вузлів системи і залишає простір для подальшого масштабування тестової бази.

```
PS F:\Football\Footballium> npx jest --coverage
>>
PASS  __tests__/domain/brackets.test.ts
PASS  __tests__/domain/stats-calculator.test.ts
PASS  __tests__/api/teams.route.test.ts
PASS  __tests__/client/ViewTracker.test.tsx
PASS  __tests__/client/FavoriteButton.test.tsx
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	43.85	57.77	50	43.85	
app/api/teams	45.58	33.33	50	45.58	
route.ts	45.58	33.33	50	45.58	16-19,36-68
components	55.26	50	50	55.26	
FavoriteButton.tsx	55.26	50	50	55.26	24-25,35-57,61-69
components/news	0	0	0	0	
ViewTracker.tsx	0	0	0	0	1-17
lib	43.11	69.56	60	43.11	
brackets.ts	88.88	64.28	100	88.88	52-62,84-85
stats-calculator.ts	17.61	77.77	33.33	17.61	26-27,38-140,143-210

```
Test Suites: 5 passed, 5 total
Tests:      8 passed, 8 total
Snapshots:  0 total
Time:       3.951 s
Ran all test suites.
```

Рис. 3.18 – Аналітичний звіт метрик покриття вихідного коду тестами

Аналізуючи деталізовані метрики по окремих файлах у звіті, варто відзначити високу ефективність тестування модуля генерації турнірної сітки (brackets.ts), де показник покриття інструкцій наблизився до 88.88%, а покриття функцій становить 100%. Це гарантує безпомилковість основного алгоритму застосунку при жеребкуванні. Водночас модуль статистичного калькулятора (stats-calculator.ts) демонструє надійне покриття логічних гілок на рівні 77.77%, забезпечуючи перевірку основних умов розрахунку, хоча покриття ліній коду вказує на наявність об'ємних блоків даних, які можна додатково параметризувати у майбутніх тестах. Компоненти інтерфейсу, такі як кнопка додавання до улюблених, успішно протестовані на 55.26%, перевіряючи основні стани відображення.

Загалом, аналіз метрик гілкування та покриття функцій для серверних маршрутів і клієнтських компонентів засвідчив надійну перевірку ключових логічних умов та унеможливив виникнення регресійних помилок при валідних вхідних даних. Залучення багаторівневого інструментарію перевірки дозволило виявити та локалізувати потенційні недоліки архітектури на ранніх етапах розробки. Отримані метрики та успішні результати як модульного, так і наскрізного тестування підтверджують високий ступінь стабільності ядра інформаційної системи та доводять її повну готовність до дослідної експлуатації.

Висновки до третього розділу

У межах третього розділу дипломної роботи успішно реалізовано фінальну стадію інженерного проєктування, що охопила практичне розгортання інформаційної системи, розробку адаптивних клієнтських інтерфейсів та комплексний автоматизований аудит програмного продукту. Впровадження сучасної методології безперервної інтеграції та доставки у синергії з хмарними потужностями Vercel і системою контролю версій GitHub дозволило створити надійний конвеєр безшовного оновлення кодової бази. Інтеграція спеціалізованих сервісів, таких як безсерверний кластер Neon PostgreSQL для збереження даних та Cloudinary для оптимізації мультимедіа, забезпечила високу масштабованість інфраструктури. При цьому використання механізмів криптографічного шифрування на стороні сервера гарантувало надійний захист.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

Спроектована візуальна оболонка, побудована з використанням компонентного підходу та утилітарного CSS-фреймворку Tailwind, довела свою ефективність у вирішенні складних бізнес-завдань для різних категорій користувачів. Реалізована багаторівнева рольова модель дозволила динамічно адаптувати робочий простір: звичайні вболівальники отримали зручний інструмент для персоналізованого моніторингу турнірних таблиць та улюблених команд, тренери закритий цифровий кабінет для управління ростером та подачі заявок на змагання, адміністратори глобальну панель модерації ресурсу, а судді максимально оптимізований веб дашборд для фіксації подій живого матчу.

На завершальному етапі розробки імплементація стратегії багаторівневого тестування за допомогою фреймворку Jest та екосистеми React Testing Library емпірично підтвердила абсолютну відмовостійкість закладеної бізнес-логіки. Проведення модульного та інтеграційного аудиту дозволило верифікувати коректність роботи критичних доменних алгоритмів, зокрема генерації турнірних сіток та математичної калькуляції спортивної статистики, а також перевірити надійність серверних маршрутів прикладного інтерфейсу і клієнтських компонентів. Детальний аналіз метрик покриття коду тестами засвідчив високий рівень захищеності ключових структур застосунку від потенційних регресійних збоїв. Отримані результати комплексного програмного аудиту беззаперечно доводять, що створена аналітична вебплатформа «Footballium» повністю відповідає заявленим технічним вимогам, позбавлена структурних вразливостей і є стовідсотково готовою до стабільного функціонування в умовах реальної комерційної експлуатації.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальне науково-практичне завдання, яке полягає у проєктуванні та розробці комплексної інформаційно-аналітичної вебплатформи «Footballium» для автоматизації процесів управління спортивними турнірами. На основі глибокого системного аналізу предметної області та дослідження існуючих аналогів було доведено необхідність створення спеціалізованого програмного продукту, здатного задовольнити специфічні інформаційні потреби різних категорій учасників футбольного процесу. Формалізація функціональних та нефункціональних вимог дозволила чітко окреслити архітектурні межі системи, визначити ключові алгоритми бізнес-логіки та спроєктувати гнучку багаторівневу модель доступу, що стало фундаментальною базою для подальшої практичної реалізації проєкту.

У ході інженерного проєктування було обґрунтовано вибір сучасного стека вебтехнологій, обчислювальним ядром якого виступив мета-фреймворк Next.js у поєднанні зі строгими стандартами статичної типізації мови TypeScript. Для забезпечення надійного збереження та ефективної обробки великих масивів реляційних даних спроєктовано оптимізовану схему бази даних PostgreSQL, розгорнуту у безсерверному хмарному кластері Neon, взаємодія з якою реалізується через механізми об'єктно-реляційного відображення Prisma ORM. Імплементация технології WebSockets за допомогою хмарної служби Pusher дозволила вирішити критично важливе інженерне завдання миттєвої синхронізації ігрових подій між серверним ядром та клієнтськими застосунками під час проведення живих матчів, гарантуючи високу пропускну здатність та мінімальні затримки передачі транзакційних пакетів до публічного табло.

Практична реалізація візуальної оболонки системи повністю базується на принципах адаптивного вебдизайну із застосуванням утилітарного CSS-фреймворку Tailwind. Завдяки розробленій динамічній рольовій моделі користувацький інтерфейс алгоритмічно трансформується відповідно до рівня системних привілеїв. Вболівальники забезпечуються персоналізованим доступом до спортивної аналітики, тренери отримують ізольований інструментарій для формування ростерів та

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

турнірних заявок, адміністратори наділяються глобальними правами модерації контенту, а арбітри використовують спеціалізовану оптимізовану мобільну панель для безпомилкової фіксації результативних дій безпосередньо на футбольному полі. З метою гарантування безпеки та безперервності процесу розгортання було успішно налаштовано автоматизований конвеєр безперервної інтеграції та доставки (CI/CD) на базі інфраструктурної платформи Vercel із використанням криптографічних секретів для захисту конфігурацій.

Для підтвердження експлуатаційної надійності та відмовостійкості створеного програмного продукту застосовано стратегію багаторівневого автоматизованого тестування із залученням інструментарію Jest та екосистеми React Testing Library. Проведення модульного та інтеграційного аудиту підтвердило абсолютну коректність роботи критичних доменних алгоритмів (зокрема, систем генерації турнірних сіток та калькуляції статистики), безпеку захищених прикладних інтерфейсів та стабільність рендерингу клієнтських компонентів. Аналіз згенерованих метрик покриття коду засвідчив високий рівень інженерної якості системи та унеможливив наявність критичних архітектурних вразливостей. Таким чином, усі поставлені перед дипломним проєктом завдання виконано в повному обсязі, а розроблена вебплатформа «Footballium» є функціонально завершеним, надійним та оптимізованим програмним комплексом, повністю готовим до впровадження у реальне комерційне середовище.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Next.js: The React Framework for the Web: офіційна документація. URL: <https://nextjs.org> (дата звернення: 17.06.2026).
2. Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine: офіційна документація. URL: <https://nodejs.org> (дата звернення: 17.06.2026).
3. TypeScript: Typed JavaScript at Any Scale: офіційна документація. URL: <https://www.typescriptlang.org> (дата звернення: 17.06.2026).
4. React: The library for web and native user interfaces: офіційна документація. URL: <https://react.dev> (дата звернення: 17.06.2026).
5. PostgreSQL: The World's Most Advanced Open Source Relational Database: офіційна документація. URL: <https://www.postgresql.org> (дата звернення: 17.06.2026).
6. Prisma: Next-generation Node.js and TypeScript ORM for Databases: офіційна документація. URL: <https://www.prisma.io> (дата звернення: 17.06.2026).
7. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML: офіційна документація. URL: <https://tailwindcss.com> (дата звернення: 17.06.2026).
8. shadcn/ui: Beautifully designed components that you can copy and paste into your apps: офіційна документація. URL: <https://ui.shadcn.com> (дата звернення: 17.06.2026).
9. NextAuth.js: Authentication for Next.js: офіційна документація. URL: <https://next-auth.js.org> (дата звернення: 17.06.2026).
10. Neon: Serverless Postgres built for the cloud: офіційна документація. URL: <https://neon.tech> (дата звернення: 17.06.2026).
11. Pusher Channels: Build scalable realtime apps: офіційна документація. URL: <https://pusher.com> (дата звернення: 17.06.2026).
12. Cloudinary: Image and Video Upload, Storage, Optimization: електронний ресурс. URL: <https://cloudinary.com> (дата звернення: 17.06.2026).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

- 13.Zod: TypeScript-first schema validation with static type inference: офіційна документація. URL: <https://zod.dev> (дата звернення: 17.06.2026).
- 14.Zustand: A small, fast and scalable bearbones state-management solution: офіційна документація. URL: <https://zustand-demo.pmnd.rs> (дата звернення: 17.06.2026).
- 15.Vercel: Develop. Preview. Ship: офіційний сайт. URL: <https://vercel.com> (дата звернення: 17.06.2026).
- 16.Jest: Delightful JavaScript Testing: офіційна документація. URL: <https://jestjs.io> (дата звернення: 17.06.2026).
- 17.React Testing Library: офіційний документаційний ресурс. URL: <https://testing-library.com/docs/react-testing-library/intro> (дата звернення: 17.06.2026).
- 18.Testing with Prisma: ресурс. URL: <https://www.prisma.io/docs/guides/testing> (дата звернення: 17.06.2026).
- 19.Perin C., Vuillemot R., Stolper C. D., Stasko J. T., Wood J., Carpendale S. State of the Art of Sports Data Visualization. Computer Graphics Forum. 2018. Vol. 37, No 3. P. 663–686. DOI: 10.1111/cgf.13447.
- 20.Spearman W. Beyond Expected Goals. Proceedings of the 12th MIT Sloan Sports Analytics Conference. 2018. P. 1–17.
- 21.FIFA Football Language: офіційний інформаційний ресурс сайту. URL: <https://www.fifa.com/technical/football-language> (дата звернення: 17.06.2026).
- 22.Lucey P., Bialkowski A., Carr P., Morgan S., Matthews I., Sheikh Y. Representing and Discovering Adversarial Team Behaviors Using Player Roles. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR. 2013. P. 2706–2713.
- 23.Chung L., Nixon B. A., Yu E. Non-functional requirements in software engineering: monograph. Boston: Springer, 2000. 476 p.
- 24.Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

25. Dinu V., Nadkarni P. Guidelines for the effective use of entity-attribute-value modeling for biomedical databases. International Journal of Medical Informatics. 2007. Vol. 76, No 11. P. 769–779. DOI: 10.1016/j.ijmedinf.2006.09.023.
26. Codd E. F. A relational model of data for large shared data banks. Communications of the ACM. 1970. Vol. 13, No 6. P. 377–387. DOI: 10.1145/362384.362685.
27. Bierman G., Abadi M., Torgersen M. Understanding typescript. European Conference on Object-Oriented Programming, ECOOP. 2014. P. 257–281. DOI: 10.1007/978-3-662-44202-9_11.
28. Tilkov S., Vinoski S. Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing. 2010. Vol. 14, No 6. P. 80–83. DOI: 10.1109/MIC.2010.145.
29. Hardt D. The OAuth 2.0 Authorization Framework: RFC 6749. Internet Engineering. 2012. 76 p. URL: <https://datatracker.ietf.org/doc/html/rfc6749> (дата звернення: 17.06.2026).
30. Jones M., Bradley J., Sakimura N. JSON Web Token, JWT: RFC 7519. Internet Engineering. 2015. 30 p. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 17.06.2026).
31. Garousi V., Felderer M. Worlds apart: industrial and academic focus areas in software testing. IEEE Software. 2017. Vol. 34, No 5. P. 38–45. DOI: 10.1109/MS.2017.3571565.
32. Visual Studio Code: Code editing. Redefined: офіційна документація. URL: <https://code.visualstudio.com> (дата звернення: 17.06.2026).
33. Chacon S., Straub B. Pro Git. 2nd ed. New York: Apress, 2014. 456 p

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

Додаток А

Діаграма варіантів використання

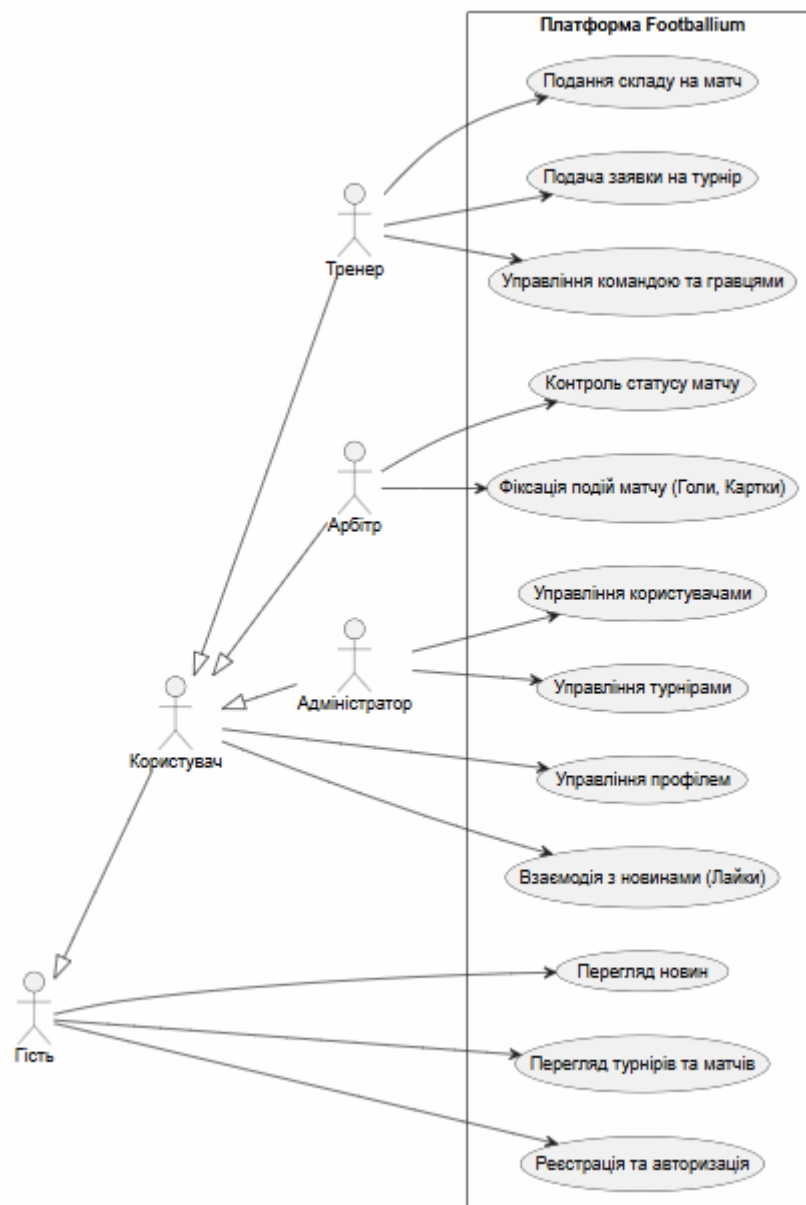


Рис. А.1 – Діаграма варіантів використання

Змн.	Арк.	№ докум.	Підпис	Дата

Додаток Б

Діаграма компонентів

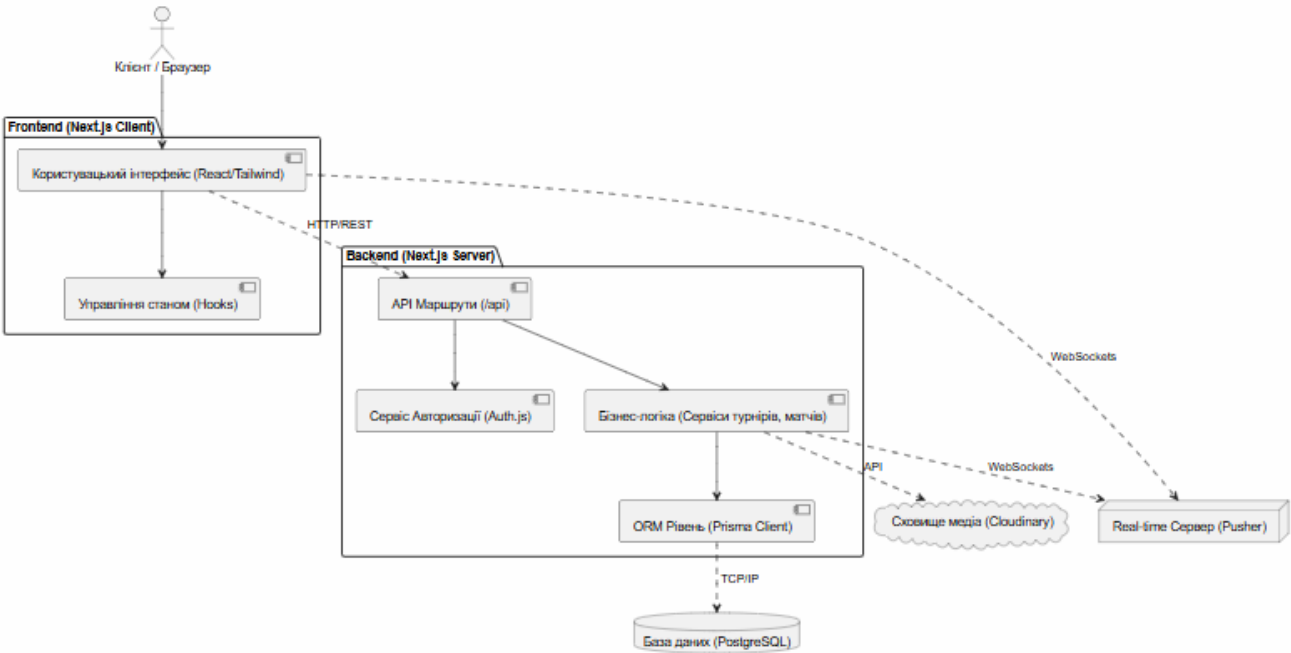


Рис. Б.1 – Діаграма компонентів

Додаток В

Діаграма шару застосунку (Application Layer)

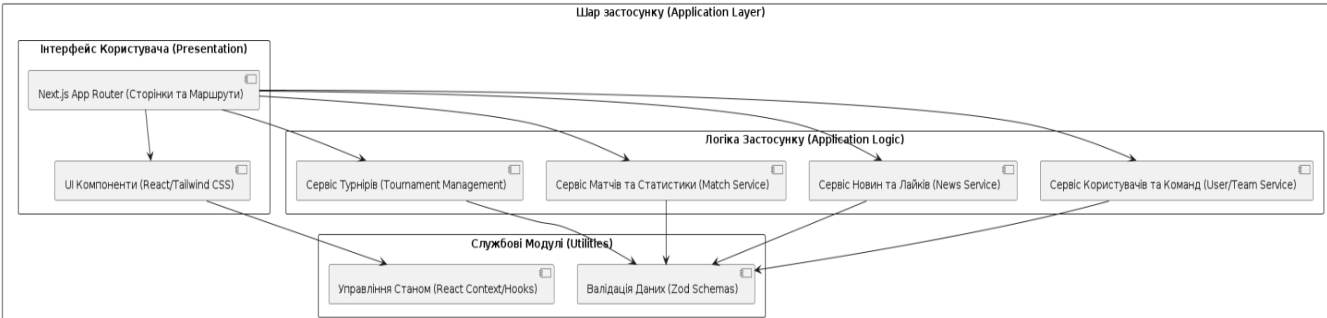


Рис. В.1 – Діаграма шару застосунку

Додаток Г

Діаграма шару інфраструктури (Infrastructure Layer Diagram)

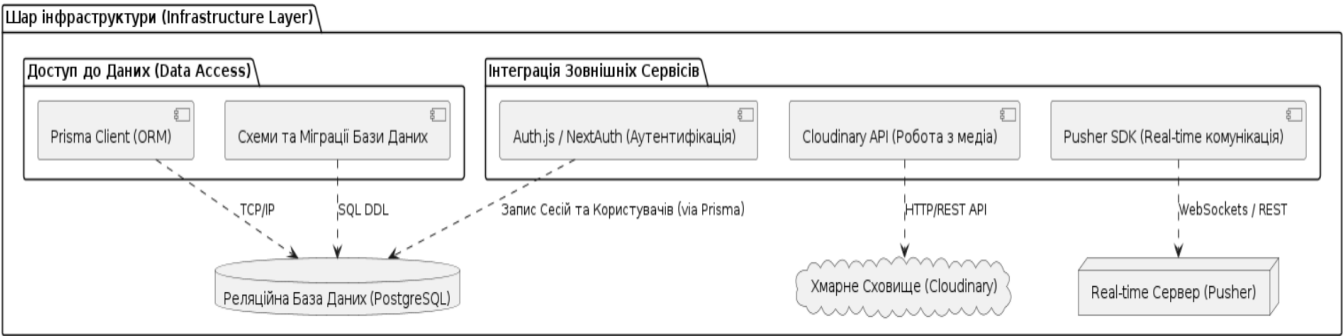


Рис. Г.1 – Діаграма шару інфраструктури

Додаток Д

Діаграма послідовності формулювання аналітичного дашборда

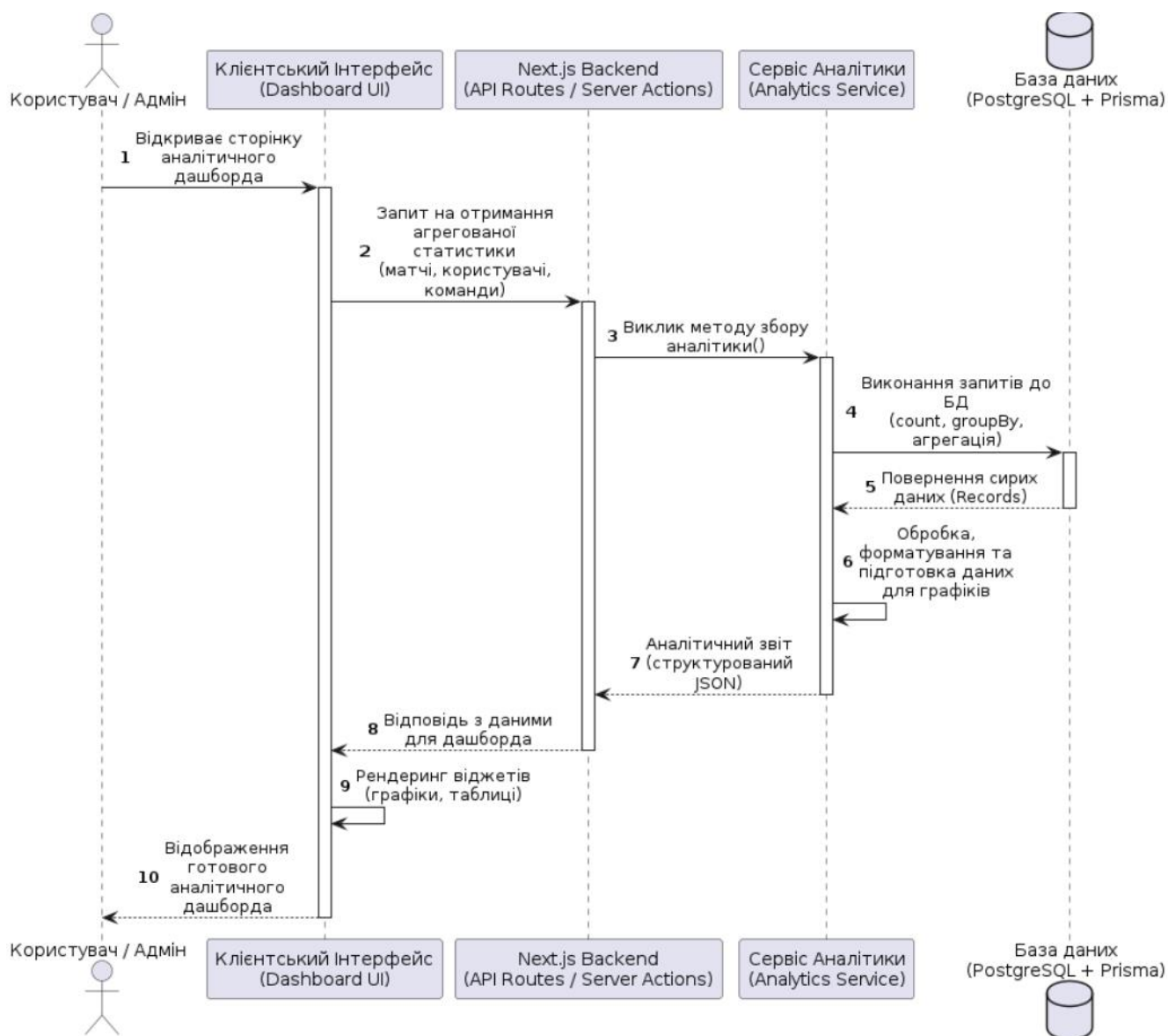


Рис. Д.1 – Діаграма послідовності створення дашборду

Додаток Е

Діаграма послідовності модерації розкладу турніру

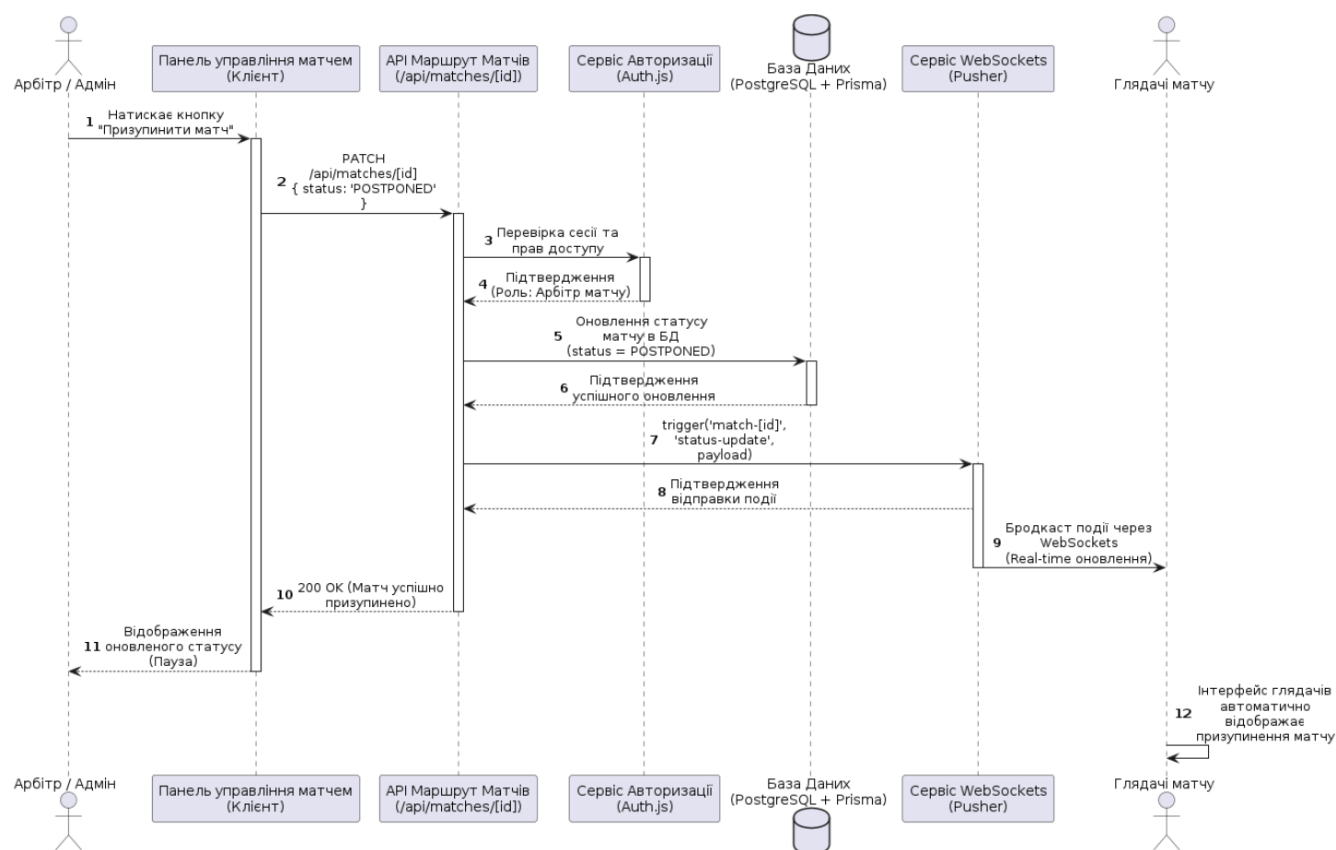


Рис. Е.1 – Діаграма послідовності призупинення матчу

Фрагменти коду серверної частини

Ж.1 Контракт створення турніру

```
import * as z from "zod";
export type CreateTournamentInput = z.infer<typeof createTournamentSchema>;
export type UpdateTournamentInput = z.infer<typeof updateTournamentSchema>;
```

Ж.2 Валідатор HTTP-запиту створення турніру

```
import * as z from "zod";
const baseTournamentSchema = z.object({
  name: z.string().min(3, { message: "Назва має містити мінімум 3 символи" }).max(100),
  description: z.string().optional(),
  bracketType: z.enum(["ROUND_ROBIN", "SINGLE_ELIMINATION"]),
  format: z.enum(["5x5", "11x11"], { message: "Оберіть формат турніру" }),
  maxTeams: z.coerce.number().min(2, { message: "Мінімум 2 команди" }).max(64),
  minTeams: z.coerce.number().min(2, { message: "Мінімум 2 команди" }),
  registrationDeadline: z.coerce.date({ message: "Оберіть дату" }),
  startDate: z.coerce.date({ message: "Оберіть дату" }),
  endDate: z.coerce.date({ message: "Оберіть дату" }),
});
export const createTournamentSchema = baseTournamentSchema
  .refine(d => d.startDate > d.registrationDeadline, {
    message: "Дата старту має бути після дедлайну реєстрації",
    path: ["startDate"]
  })
  .refine(d => d.minTeams <= d.maxTeams, {
    message: "Мінімальна кількість команд не може бути більшою за максимальну",
    path: ["minTeams"]
  });
```

Ж.3 Метод маршруту (App Router) для обробки POST-запиту

```
import { NextRequest, NextResponse } from "next/server";
import { auth } from "@lib/auth";
import { prisma } from "@lib/prisma";
import { createTournamentSchema } from "@lib/validations/tournament";
export async function POST(req: NextRequest) {
  try {
    const session = await auth();
    if (!session?.user || (session.user as any).role !== "ADMIN") {
      return new NextResponse("Unauthorized", { status: 403 });
    }
    const body = await req.json();
    const parsed = createTournamentSchema.safeParse(body);

    if (!parsed.success) {
      return new NextResponse(JSON.stringify(parsed.error.format()), {
        status: 400,
        headers: { "Content-Type": "application/json" },
      });
    }
    const data = parsed.data;
    const tournament = await prisma.tournament.create({
      data: {
        ...data,
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

        createdById: session.user!.id!,
    },
    });
    return NextResponse.json(tournament, { status: 201 });
} catch (error) {
    console.error("CREATE_TOURNAMENT_ERROR", error);
    return new NextResponse("Internal Error", { status: 500 });
}
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		