НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри комп'ютерних наук**
_____**Ігор БОЛБОТ** (підпис)	_____**Белла ГОЛУБ** (підпис)
«_____» _____2026 р.	«_____» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система обліку статистичних даних для футбольного менеджера»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор _____ **Роман РУДЕНСЬКИЙ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Олексій СТЕПАНОВ**
(підпис)

Виконав _____ **Максим БУРДИНА**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____Белла ГОЛУБ
(підпис)

«___» _____2026 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Бурдині Максиму Віталійовичу
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи

«Система обліку статистичних даних для футбольного менеджера» затверджена наказом від « 06 » січня 2026 р. № 46 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: аналіз предметної області, вимоги до вебсистеми Football Manager, дані про користувачів, команди, гравців, матчі, турніри та статистику.

Перелік питань, що підлягають дослідженню:
1. Аналіз предметної області управління футбольною командою.
2. Вибір і обґрунтування засобів розробки вебсистеми.
3. Проєктування структури системи та бази даних.
4. Формування статистичної, аналітичної та звітної інформації.

Дата видачі завдання « 06 » січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Олексій СТЕПАНОВ**
(підпис)

Завдання взяв до виконання _____ **Максим БУРДИНА**
(підпис)

ЗМІСТ

ВСТУП 4
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Постановка завдання 7
1.2 Огляд офіційних джерел та існуючих рішень 9
1.3 Моделювання предметної області 14
2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 18
2.1 Логічна модель даних 18
2.2 Вибір системи управління базою даних 22
2.3 Створення інформаційної бази даних 24
3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 32
3.1 Організаційна структура програмного забезпечення 32
3.2 Вибір інструментарію для створення ППЗ 34
3.3 Алгоритмізація та програмування програмних модулів 36
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 53
4.1 Тестування системи 53
4.2 Вимоги до апаратного та програмного забезпечення 58
4.3 Склад інсталяційного пакету 61
ВИСНОВКИ 64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 66
Додаток А 69
Додаток Б 76
Додаток В 79
Додаток Д 85
Додаток Е 90

ВСТУП

Актуальність теми бакалаврської кваліфікаційної роботи зумовлена зростанням ролі статистичних даних у сучасному футболі. Управління командою, оцінювання ефективності гравців, аналіз результатів матчів, контроль турнірного становища та підготовка аналітичних висновків дедалі більше ґрунтуються на систематизованій інформації. Це підтверджується як практикою офіційних футбольних організацій, що постійно публікують календарі, турнірні таблиці, матчеві та індивідуальні статистичні показники, так і сучасними підходами до футбольної аналітики [1]–[5]. В українському футбольному середовищі особливе значення мають офіційні ресурси Української асоціації футболу та Української Прем'єр-ліги, які забезпечують інформаційну підтримку змагань, відображають результати матчів, статистику та турнірні дані [1], [2]. На міжнародному рівні важливість якісних футбольних даних і засобів їх аналізу також підкреслюється у матеріалах FIFA та UEFA [3], [4]. Усе це свідчить про необхідність створення спеціалізованих програмних систем, здатних автоматизувати облік і опрацювання статистичної інформації.

Метою роботи є розробка веборієнтованої системи обліку статистичних даних для футбольного менеджера, яка забезпечує зберігання, обробку, перегляд і аналіз інформації про користувачів, команди, гравців, матчі, турніри, статистичні показники та звітні матеріали. Об'єктом дослідження є процеси обліку та використання статистичних даних у діяльності футбольного менеджера. Предметом дослідження є методи, моделі та програмні засоби побудови інформаційної системи для автоматизації роботи зі статистикою футбольної команди.

Для досягнення поставленої мети в роботі необхідно здійснити аналіз предметної області, визначити функціональні потреби системи, обґрунтувати вибір технологій розробки, спроєктувати структуру бази даних і реалізувати основні програмні модулі. У межах проєкту створено систему, що підтримує

роботу з обліковими записами користувачів, командами, гравцями, матчами, турнірами, стартовими складами, турнірною таблицею, а також скаутськими й аналітичними звітами. Серверна частина реалізована на платформі Node.js із використанням вебфреймворку Express [6], [7], а збереження даних організовано за допомогою SQLite [8]. Такий підхід дає змогу поєднати відносну простоту реалізації з достатніми можливостями для структурованого зберігання та опрацювання інформації.

Практичне значення розроблюваної системи полягає в тому, що вона дає змогу централізувати статистичні дані, зменшити кількість ручних операцій, підвищити зручність доступу до інформації та покращити інформаційну підтримку управлінських рішень. Використання єдиної системи спрощує облік гравців, ведення матчів, перегляд індивідуальної й командної статистики, формування стартових складів і підготовку звітів, що є важливим для ефективної організації роботи у футбольному середовищі.

Основні положення бакалаврської кваліфікаційної роботи були апробовані у вигляді доповіді «Система обліку статистичних даних для футбольного менеджера», прийнятої для оприлюднення на науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем». У доповіді було висвітлено актуальність автоматизації обліку футбольної статистики, загальну ідею програмної системи та основні напрями її практичного застосування.

Структура пояснювальної записки побудована відповідно до логіки дослідження та розробки програмної системи. У першому розділі розглядаються питання системного аналізу предметної області, постановки завдання, огляду інформаційних джерел та моделювання предметної області. У другому розділі висвітлюються питання інформаційного забезпечення, логічної моделі даних, вибору системи управління базою даних і створення інформаційної бази [8]. Третій розділ присвячено прикладному програмному забезпеченню, вибору інструментарію та реалізації основних програмних модулів із використанням сучасних вебтехнологій [6], [7]. У четвертому розділі розглядаються питання

тестування, вимог до апаратного та програмного забезпечення, а також впровадження та експлуатації створеної системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Предметна область дослідження охоплює процеси збирання, зберігання, обробки та використання статистичних даних у діяльності футбольного менеджера. У сучасному футболі прийняття рішень дедалі більше спирається не лише на загальні результати матчів, а й на деталізовану інформацію про склад команди, індивідуальні показники гравців, турнірне становище, календар змагань і підсумкову аналітику. Це підтверджується як офіційними інформаційними ресурсами українського футболу, на яких систематично подаються матчі, результати, турнірні таблиці та статистичні матеріали [9], [10], так і сучасними підходами до спортивної аналітики, що розглядають дані як основу для оцінювання гри та підтримки управлінських рішень [12], [13].

Проблема полягає в тому, що за відсутності єдиної автоматизованої системи облік статистики футбольної команди часто здійснюється у розрізненому вигляді. Частина відомостей може зберігатися у таблицях, частина у текстових записах або звітах, а частина взагалі не має впорядкованої форми. Такий підхід ускладнює оновлення даних, порівняння показників, формування висновків і підготовку звітних матеріалів. Окремі труднощі виникають під час ведення календаря матчів, контролю турнірного становища, перегляду індивідуальної статистики гравців і роботи зі складами команди. Крім того, футбольна діяльність функціонує в межах регламентованого змагального середовища, що вимагає впорядкованості та узгодженості інформації [11].

Призначення розроблюваної системи полягає в автоматизації обліку статистичних даних для футбольного менеджера та інших користувачів, які беруть участь у роботі з командою. Система має забезпечувати централізоване зберігання інформації, зручний доступ до неї, розмежування прав доступу залежно від ролі користувача та можливість подальшого аналізу накопичених

даних. У контексті реалізованого проєкту така система орієнтована на підтримку повсякденних управлінських і аналітичних процесів, пов’язаних із футбольною командою.

Відповідно до логіки розробленої системи, вона повинна зберігати та обробляти дані про користувачів, команди, гравців, турніри, матчі, статистику гравців, турнірну таблицю, стартові склади, скаутські та аналітичні звіти. Такі дані утворюють єдину інформаційну базу, у межах якої пов’язані між собою спортивні, організаційні та аналітичні відомості. Наявність у системі кількох взаємопов’язаних груп даних дає змогу отримувати не лише окремі записи, а й формувати узагальнену картину стану команди, результатів її виступів та ефективності окремих гравців.

У системі підлягають автоматизації основні процеси роботи з футбольною статистикою. До них належать облік гравців і їхніх характеристик, ведення матчів і результатів, перегляд індивідуальної та командної статистики, відображення турнірної таблиці, керування стартовими складами, а також формування скаутських і аналітичних звітів. Окреме значення має автоматизація роботи з матчами та турнірною інформацією, оскільки офіційні футбольні ресурси демонструють, що саме календар змагань, результати та таблиця є базовими складовими інформаційного супроводу футбольного процесу [9], [10]. Водночас сучасні підходи до футбольної аналітики підкреслюють важливість накопичення детальніших даних про дії гравців та їх використання для оцінювання ефективності [12], [13].

Важливим аспектом постановки завдання є врахування категорій користувачів системи та їхніх повноважень. У проєкті передбачено ролі адміністратора, тренера, скаута, аналітика, гравця та користувача, який очікує підтвердження. Адміністратор здійснює керування обліковими записами та ролями. Тренер працює з даними своєї команди, гравцями, матчами, складами та пов’язаною статистикою. Скаут створює скаутські звіти щодо гравців, аналітик формує аналітичні звіти та працює з матчевими даними, а гравець має обмежений доступ до перегляду інформації. Такий розподіл функцій забезпечує

відповідність системи реальним сценаріям використання у футбольному середовищі.

До основних функціональних вимог системи належать реєстрація та автентифікація користувачів, адміністрування акаунтів і ролей, зберігання й перегляд інформації про команди та гравців, перегляд статистики, ведення матчів, робота з турнірною таблицею, збереження складів, а також створення та перегляд скаутських і аналітичних звітів. До основних нефункціональних вимог належать зручність використання, надійність, захист даних, достатня швидкість роботи та стабільне збереження інформації в базі даних. Система повинна бути зрозумілою для різних категорій користувачів, забезпечувати цілісність і доступність даних, а також підтримувати безпечну роботу з обліковими записами та статистичною інформацією.

Отже, постановка завдання полягає у створенні веборієнтованої системи обліку статистичних даних для футбольного менеджера, яка забезпечить централізоване зберігання інформації, автоматизацію основних процесів роботи з командами, гравцями, матчами, турнірами, складами та звітами, а також надасть користувачам зручний інструмент для перегляду, аналізу й використання футбольної статистики в управлінській діяльності.

1.2 Огляд офіційних джерел та існуючих рішень

Під час проєктування системи обліку статистичних даних для футбольного менеджера доцільно проаналізувати вже наявні інформаційні сервіси, які працюють зі спортивною статистикою та футбольними даними. Такий аналіз дає змогу визначити, які функції є найбільш затребуваними, які підходи до подання інформації використовуються на практиці, а також які обмеження існують у готових рішеннях. Серед найбільш відомих сервісів, пов’язаних зі статистикою та футбольною інформацією, доцільно виділити Sofascore і Transfermarkt, оскільки вони охоплюють різні аспекти роботи з футбольними даними [14]–[17].

Сервіс Sofascore призначений насамперед для оперативного відображення спортивної інформації в реальному часі. На офіційному сайті зазначено, що платформа забезпечує покриття футбольних ліг, кубків і турнірів, надає результати матчів, статистику, турнірні таблиці, календарі та інші супровідні дані [14]. Окремою особливістю Sofascore є система рейтингів гравців, що формується на основі аналізу статистичних показників. За інформацією самого сервісу, алгоритм аналізує велику кількість даних, формує рейтинг гравця в режимі реального часу та оновлює його протягом матчу [15].

Основними перевагами сервісу Sofascore є:

1. оперативне оновлення результатів матчів;
2. наявність статистики команд і гравців;
3. зручне відображення календаря, турнірних таблиць і рахунків;
4. використання рейтингової оцінки футболістів.

Разом із тим Sofascore має певні обмеження з погляду внутрішньої роботи футбольного менеджера. Сервіс орієнтований переважно на масового користувача та перегляд готової статистики. Він не призначений для адміністрування власних облікових записів команди, керування ролями користувачів, формування внутрішніх складів або збереження скаутських та аналітичних звітів у межах окремої інформаційної системи.

Рис. 1.1 – Інтерфейс сервісу Sofascore

Іншим важливим прикладом існуючого рішення є сервіс Transfermarkt. Його призначення полягає у наданні інформації про футбольні трансфери, ринкову вартість гравців, склади команд, контракти, статистику виступів і пов’язані з цим новини [16]. На відміну від Sofascore, який більше акцентує увагу на матчевій статистиці та результатах у реальному часі, Transfermarkt зосереджується на довгостроковій інформації про кар’єру гравця, його трансферну історію, зміну ринкової вартості та склад клубу.

Основними перевагами сервісу Transfermarkt є:

1. велика база даних про гравців, клуби й трансфери;
2. відображення ринкової вартості футболістів;
3. наявність історичної інформації про кар’єру гравців;
4. зручність аналізу трансферної активності клубів.

Недоліком Transfermarkt у межах поставленої задачі є те, що сервіс більше орієнтований на трансферну та довідкову інформацію, ніж на внутрішню роботу футбольного менеджера. Він не забезпечує керування власними користувачами, ролями, складами, матчами та звітами в межах окремої команди. Крім того, акцент на ринковій вартості та трансферній інформації не покриває повністю потребу в щоденному статистичному супроводі команди.

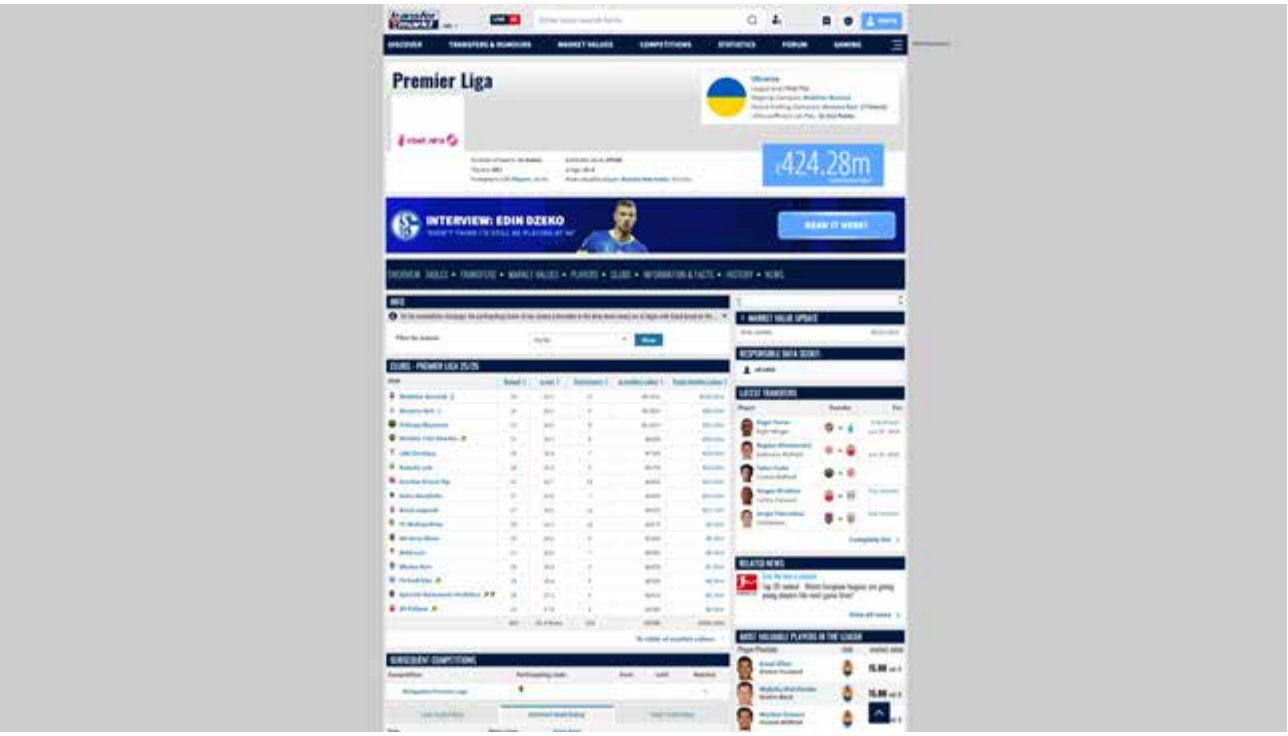

Рис. 1.2 – Інтерфейс сервісу Transfermarkt

Для узагальнення результатів аналізу доцільно порівняти Sofascore, Transfermarkt і розроблювану систему за основними критеріями.

Таблиця 1

Порівняння існуючих рішень із розроблюваною системою

Критерій	**Sofascore**	**Transfermarkt**	**Розроблювана система**
Основне призначення	Перегляд результатів і статистики матчів	Трансферна, ринкова та довідкова інформація	Облік статистичних даних футбольного менеджера
Матчева статистика	Є	Частково	Є
Дані про гравців	Є	Є	Є
Турнірні таблиці	Є	Частково	Є
Формування складу	Немає	Немає	Є

Критерій	**Sofascore**	**Transfermarkt**	**Розроблювана система**
Скаутські звіти	Немає	Частково як довідкова інформація	Є
Аналітичні звіти	Немає	Немає	Є
Керування ролями користувачів	Немає	Немає	Є
Внутрішнє адміністрування системи	Немає	Немає	Є

Як видно з таблиці 1.1, існуючі сервіси мають значні можливості для перегляду футбольної інформації, однак вони не орієнтовані на внутрішню роботу конкретної команди. Sofascore є зручним інструментом для перегляду результатів матчів і статистики, а Transfermarkt корисний для аналізу трансферної та ринкової інформації. Водночас жоден із цих сервісів не забезпечує повний набір функцій, необхідний для внутрішнього обліку футбольного менеджера. Розроблювана система відрізняється тим, що поєднує облік гравців, матчів, турнірів, статистики, складів, звітів і ролей користувачів у межах одного програмного середовища.

Отже, створення власної системи обліку статистичних даних є доцільним, оскільки вона орієнтована не лише на перегляд готової інформації, а й на організацію внутрішньої роботи з футбольними даними. Такий підхід дозволяє адаптувати функціональність системи до потреб тренера, аналітика, скаута, адміністратора та гравця.

1.3 Моделювання предметної області

Моделювання предметної області є важливим етапом розробки інформаційної системи, оскільки дає змогу визначити основних користувачів, функціональні можливості системи та логіку виконання ключових процесів. Для опису системи обліку статистичних даних для футбольного менеджера доцільно використати UML-діаграми, зокрема діаграму прецедентів і діаграму діяльності. Діаграма прецедентів застосовується для відображення взаємодії акторів із функціями системи [18], а діаграма діяльності дає змогу описати послідовність дій, умови переходів і загальну логіку виконання процесу [19].

У межах предметної області розроблюваної системи основними сутностями є користувачі, команди, гравці, матчі, турніри, статистичні показники, склади, скаутські та аналітичні звіти. Система призначена для автоматизації роботи з футбольною інформацією та забезпечує доступ до функцій відповідно до ролі користувача. У програмному продукті передбачено ролі адміністратора, тренера, аналітика, скаута та гравця. Кожна роль має власний набір дій, що відповідає її призначенню в системі.

Для відображення функціональних можливостей системи було побудовано діаграму прецедентів. На ній показано основних акторів і функції, з якими вони взаємодіють. Адміністратор виконує підтвердження реєстрації, зміну ролей акаунтів і видалення облікових записів. Тренер переглядає дані про команду, працює зі звітами та формує склад. Аналітик має доступ до перегляду командної інформації, звітів, редагування даних матчів, аналізу матчевої статистики та формування аналітичних звітів. Скаут переглядає дані про команду, звіти та формує скаутські звіти. Гравець має доступ до перегляду даних про команду. Така діаграма дає змогу наочно представити розподіл функціональних можливостей між ролями користувачів [18].

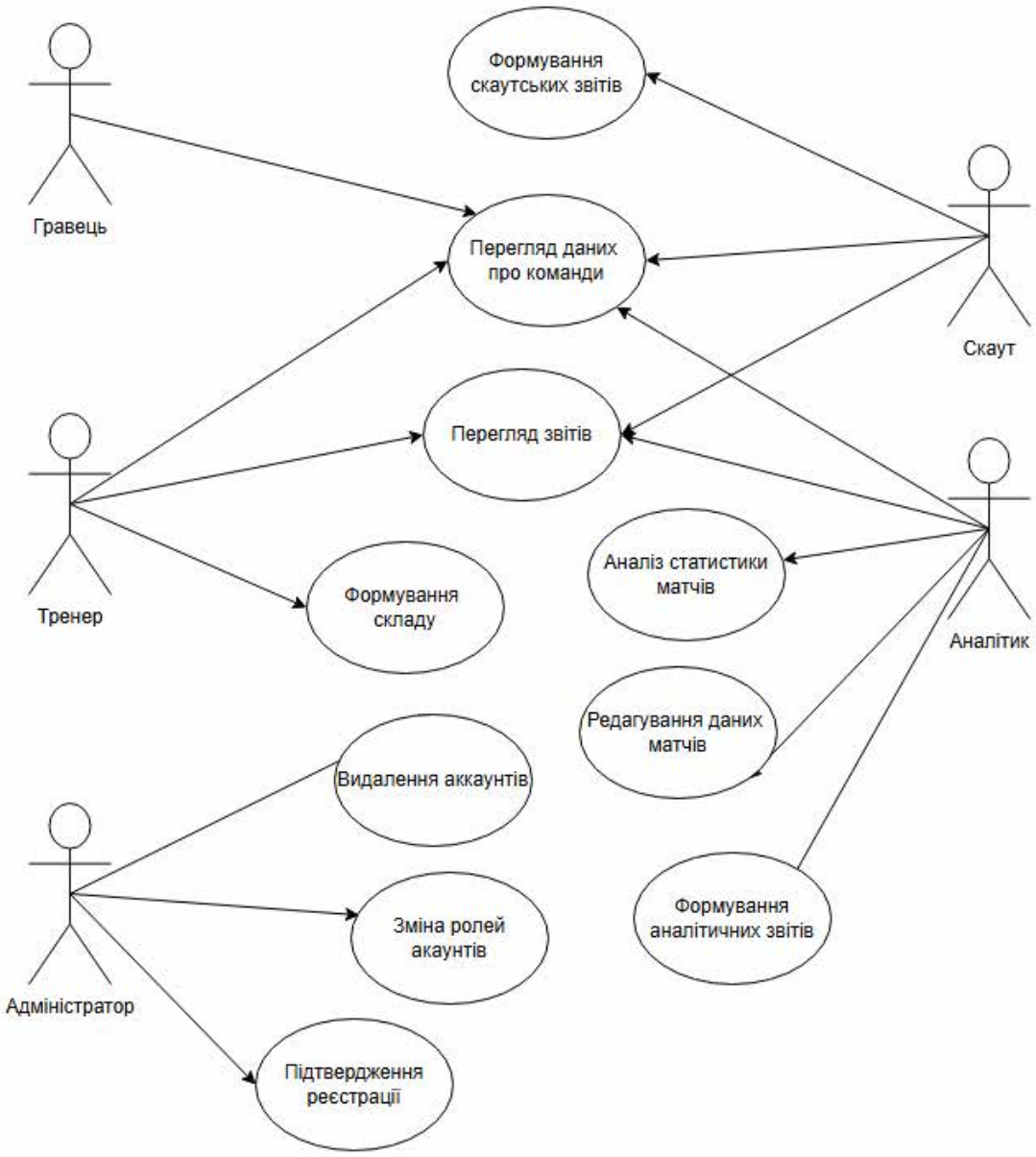

Рис. 1.3 – Діаграма прецедентів системи обліку статистичних даних для футбольного менеджера

Діаграма прецедентів відображає межі взаємодії користувачів із програмною системою та показує, які функції доступні кожній ролі. Завдяки цьому можна чітко визначити відповідальність адміністратора, тренера, аналітика, скаута та гравця в межах системи. Такий підхід спрощує подальше проєктування програмного забезпечення, оскільки основні сценарії використання стають структурованими та зрозумілими.

Для опису загальної логіки роботи користувачів було побудовано діаграму діяльності. Вона відображає процес входу до системи, перевірку наявності

облікового запису, можливість реєстрації нового користувача та подальший перехід до функцій відповідно до ролі. Якщо обліковий запис існує, система надає користувачу доступ до відповідного функціоналу. Якщо обліковий запис відсутній, користувач може пройти процедуру реєстрації. Після створення акаунта адміністратор має можливість підтвердити користувача та призначити йому відповідну роль.

У діаграмі діяльності окремо показано логіку дій кожної ролі. Адміністратор підтверджує реєстрацію, змінює ролі акаунтів, видаляє облікові записи та зберігає внесені зміни. Тренер переглядає інформацію про команду, працює зі звітами, формує та зберігає склад. Аналітик переглядає командні дані, редагує інформацію про матч, аналізує статистику матчів, формує та зберігає аналітичний звіт. Скаут переглядає дані про команду, звіти, формує та зберігає скаутський звіт. Гравець переглядає інформацію про команду, особисту статистику та розклад матчів. Діаграма діяльності дає змогу показати послідовність виконання цих дій і логіку переходу між основними процесами.

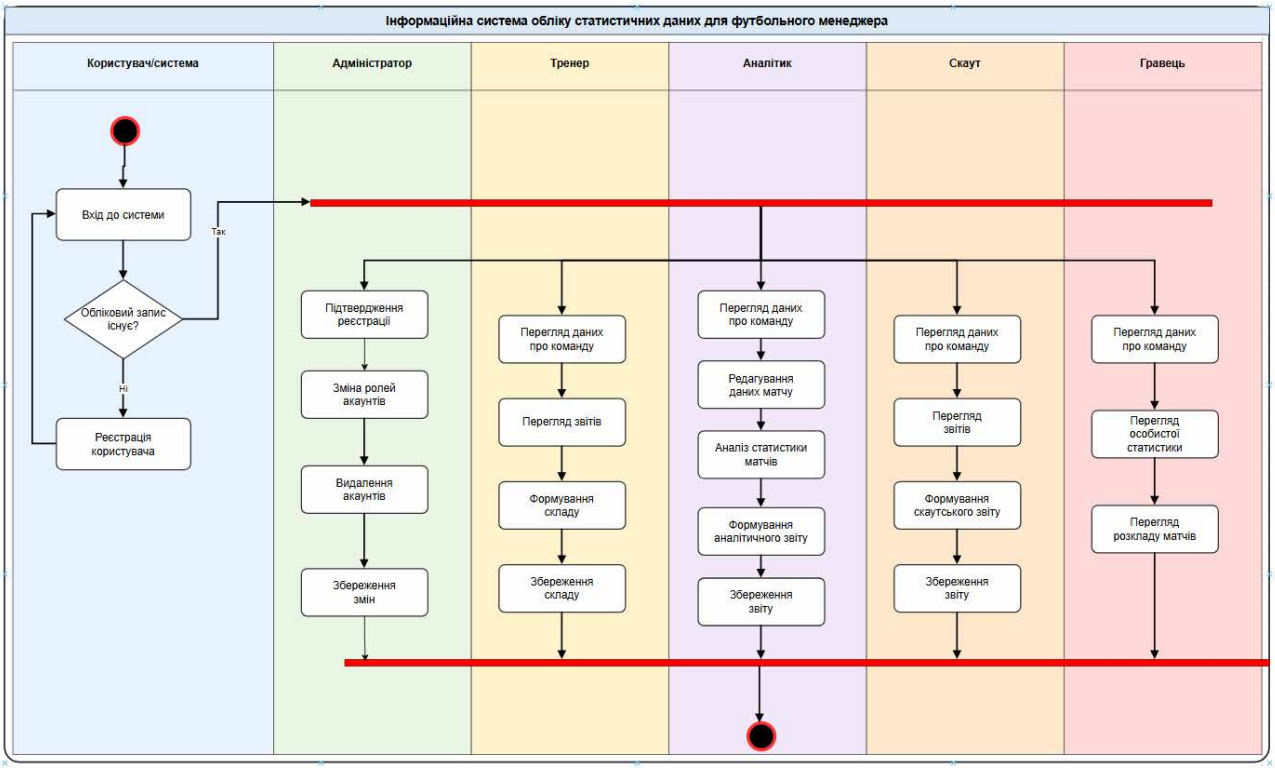

Рис. 1.4 – Діаграма діяльності процесу роботи користувачів у системі

Таким чином, моделювання предметної області дозволило формалізувати основні ролі користувачів, функції системи та логіку виконання ключових процесів. Діаграма прецедентів показує функціональні можливості системи з погляду користувачів, а діаграма діяльності відображає послідовність дій під час роботи з програмним продуктом. Використання цих діаграм є доцільним, оскільки вони забезпечують зрозуміле подання структури системи та створюють основу для подальшого проєктування інформаційного й програмного забезпечення.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних є основою інформаційного забезпечення програмної системи, оскільки визначає склад основних сутностей, їхні атрибути та зв’язки між ними. Для системи обліку статистичних даних для футбольного менеджера логічна модель побудована на основі реляційного підходу, за якого інформація зберігається у вигляді взаємопов’язаних таблиць [20]. Такий підхід є доцільним для цієї предметної області, оскільки дані про користувачів, команди, гравців, матчі, турніри, склади та звіти мають чітку структуру й логічні зв’язки між собою.

У межах розроблюваної системи основними сутностями є users, teams, players, tournaments, matches, match_statistics, player_statistics, standings, lineups, lineup_players, scout_reports та analyst_reports. Кожна з цих таблиць відповідає окремому інформаційному об’єкту предметної області.

До основних таблиць логічної моделі даних належать:

1. users — зберігає дані користувачів системи та їхні ролі;
2. teams — містить інформацію про футбольні команди;
3. players — зберігає дані про гравців;
4. tournaments — описує турніри та сезони;
5. matches — містить інформацію про матчі;
6. player_statistics і match_statistics — зберігають статистичні показники;
7. standings — відображає турнірне становище команд;
8. lineups і lineup_players — використовуються для формування складів;
9. scout_reports і analyst_reports — забезпечують роботу зі звітами.

Такий розподіл таблиць дає змогу логічно відокремити різні групи даних і водночас забезпечити зв’язок між ними. Наприклад, гравці пов’язані з командами, матчі — з турнірами та командами, статистика — з гравцями або матчами, а звіти — з користувачами, які їх створюють.

Таблиця users зберігає дані про користувачів системи, зокрема ім’я, прізвище, електронну адресу, роль і прив’язку до команди. Таблиця teams містить інформацію про футбольні команди: назву, місто, тренера та стадіон. Зв’язок між користувачами й командами реалізується через поле team_id, що дає змогу визначити, до якої команди належить користувач.

Таблиця players використовується для зберігання відомостей про футболістів. Вона містить персональні дані гравця, його вік, позицію та посилання на команду. Завдяки зв’язку players.team_id з таблицею teams система може відображати склад конкретної команди та обмежувати роботу тренера лише гравцями його команди. Для збереження статистичних показників футболістів використовується таблиця player_statistics, яка пов’язана з таблицями players і tournaments. У ній зберігаються кількість матчів, голів, асистів і рейтинг гравця в межах певного турніру.

Для обліку змагань у системі використовується таблиця tournaments, у якій зберігаються назва турніру, тип, сезон і статус. З нею пов’язані таблиці standings, matches і player_statistics. Таблиця standings відображає турнірне становище команд і містить дані про кількість зіграних матчів, перемог, нічиїх та набраних очок. Вона має зовнішні ключі tournament_id і team_id, що дає змогу формувати турнірну таблицю для конкретного змагання. Зовнішній ключ використовується для встановлення зв’язку між таблицями та забезпечення посилальної цілісності даних [21].

Таблиця matches призначена для зберігання інформації про матчі. Вона містить посилання на турнір, домашню команду, виїзну команду, тур, дату матчу та статус. Поля home_team_id і away_team_id є зовнішніми ключами, які посилаються на таблицю teams і визначають відповідно домашню та виїзну команду матчу. Такий підхід дозволяє зберігати інформацію про обидві команди-

учасниці без дублювання даних про них у таблиці матчів. Для детального опису завершених матчів використовується таблиця match_statistics, яка пов’язана з таблицею matches через поле match_id і містить показники володіння м’ячем та очікуваних голів для обох команд.

Окремий блок логічної моделі становлять таблиці, пов’язані зі складами команд. Таблиця lineups зберігає інформацію про склад команди, його схему, назву та користувача, який створив або зберіг склад. Вона пов’язана з таблицями teams і users. Таблиця lineup_players є проміжною таблицею між складами та гравцями. Вона містить посилання на склад, гравця, роль у складі та позиційний ключ. Така структура дає змогу формувати стартовий склад, резерв і запасних гравців без дублювання даних про футболістів.

Для підтримки звітної та аналітичної роботи в системі передбачено таблиці scout_reports і analyst_reports. Таблиця scout_reports зберігає скаутські звіти щодо гравців і містить посилання на гравця та користувача-скаута, який створив звіт. У ній також фіксуються сильні сторони, рівень ризику та потенційний рейтинг футболіста. Таблиця analyst_reports містить аналітичні звіти, пов’язані з користувачем-аналітиком, темою звіту, періодом, знімком метрик і висновком. Наявність цих таблиць дає змогу накопичувати не лише числову статистику, а й текстову аналітичну інформацію.

Для наочного подання логічної структури бази даних використовується ER-діаграма. ER-діаграма є моделлю, яка відображає основні сутності предметної області, їхні атрибути та зв’язки між ними [22]. У контексті розроблюваної системи вона дає змогу показати, які таблиці входять до складу бази даних, які поля є первинними та зовнішніми ключами, а також яким чином пов’язані між собою користувачі, команди, гравці, матчі, турніри, склади та звіти. Використання ER-діаграми спрощує розуміння структури інформаційної бази й допомагає перевірити логічну узгодженість зв’язків між таблицями.

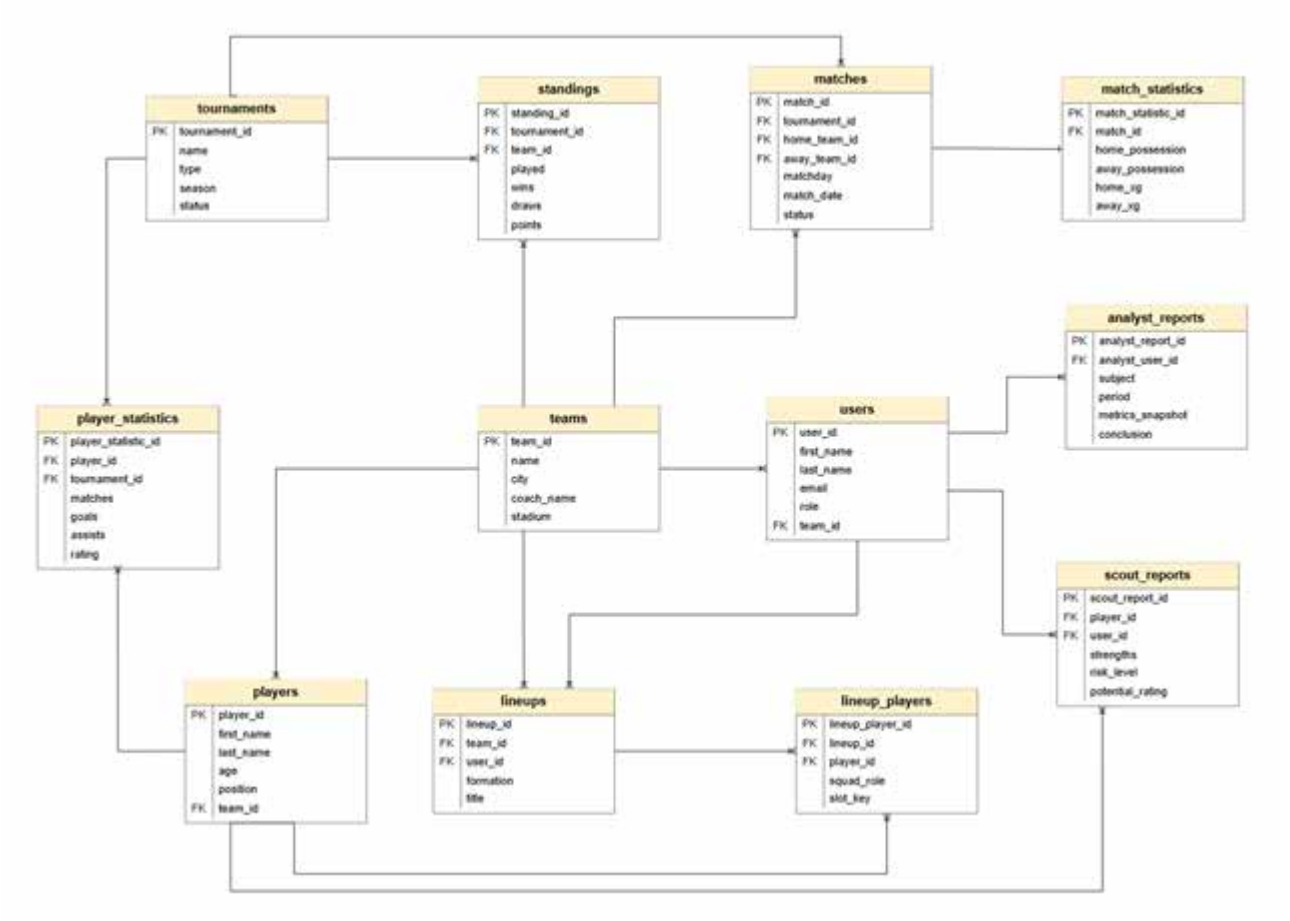

Рис. 2.1 – Логічна модель даних системи обліку статистичних даних для футбольного менеджера

На рисунку 2.1 подано логічну модель даних системи. Вона відображає основні таблиці, первинні та зовнішні ключі, а також зв’язки між сутностями. Центральними сутностями моделі є teams, users, players, matches і tournaments, оскільки саме навколо них формується більшість зв’язків. Команди пов’язані з користувачами, гравцями, матчами, турнірною таблицею та складами. Гравці пов’язані зі статистикою, скаутськими звітами та складом команди. Матчі пов’язані з турнірами, командами та матчевою статистикою. Така структура забезпечує цілісне зберігання інформації та дає змогу виконувати вибірку даних для формування статистичних і звітних матеріалів.

Отже, побудована логічна модель даних відповідає потребам предметної області та охоплює основні інформаційні об’єкти системи. Вона забезпечує збереження даних про користувачів, команди, гравців, турніри, матчі, статистику, склади та звіти, а також встановлює зв’язки між ними. Завдяки такій

структурі система може підтримувати рольову роботу користувачів, відображати статистичну інформацію, формувати склади, працювати з турнірними таблицями та накопичувати аналітичні матеріали.

2.2 Вибір системи управління базою даних

Вибір системи управління базою даних є важливим етапом розробки інформаційної системи, оскільки саме СУБД забезпечує збереження, обробку, пошук і цілісність даних. Для системи обліку статистичних даних для футбольного менеджера база даних повинна підтримувати зберігання інформації про користувачів, команди, гравців, турніри, матчі, статистичні показники, склади та звіти. Оскільки ці дані мають чітку структуру та пов’язані між собою, доцільним є використання реляційної моделі даних [20].

У процесі розробки програмної системи як систему управління базою даних було обрано SQLite. SQLite є вбудованою реляційною СУБД, яка не потребує окремого серверного процесу та зберігає всю базу даних в одному файлі [8]. Така особливість робить її зручною для невеликих і середніх вебзастосунків, навчальних проєктів, прототипів і систем, де важливими є простота розгортання та мінімальні вимоги до адміністрування.

Основними причинами вибору SQLite для розроблюваної системи є:

1. простота встановлення та розгортання;
2. збереження бази даних в одному файлі;
3. підтримка SQL-запитів і реляційної структури;
4. можливість використання первинних і зовнішніх ключів;
5. достатня функціональність для навчального та демонстраційного вебзастосунку;
6. зручна інтеграція з Node.js.

Для системи обліку статистичних даних футбольного менеджера ці переваги є важливими, оскільки проєкт потребує зручного локального запуску, надійного збереження структурованої інформації та можливості швидко

демонструвати роботу системи без складного налаштування серверної інфраструктури.

Таблиця 2

Обґрунтування вибору SQLite

Критерій	**Значення для проєкту**
Простота розгортання	Не потребує окремого сервера бази даних
Зберігання даних	Уся база зберігається в одному файлі
Підтримка SQL	Дозволяє створювати таблиці, зв’язки та запити
Інтеграція з Node.js	Є готові бібліотеки для роботи з SQLite
Відповідність масштабу системи	Підходить для локальної та демонстраційної системи
Можливість перенесення	За потреби структура може бути перенесена на MySQL або PostgreSQL

У межах розробленої системи SQLite використовується для зберігання всіх основних даних. У базі даних містяться таблиці користувачів, команд, гравців, турнірів, матчів, турнірної таблиці, статистики гравців, статистики матчів, складів, гравців у складі, скаутських та аналітичних звітів. Завдяки використанню зовнішніх ключів забезпечується зв’язок між таблицями, наприклад між командами та гравцями, матчами та командами, турнірами та турнірною таблицею, складами та гравцями. Зовнішні ключі дають змогу підтримувати посилальну цілісність даних і зменшують ризик появи неузгоджених записів [21].

Порівняно з більш потужними серверними СУБД, такими як MySQL або PostgreSQL, SQLite має простішу архітектуру та не вимагає окремого налаштування сервера бази даних. Для даного проєкту це є перевагою, оскільки система створюється як компактний вебзастосунок для обліку футбольної статистики, а не як високонавантажена корпоративна платформа. Водночас

SQLite підтримує основні можливості, необхідні для роботи системи: створення таблиць, виконання SQL-запитів, зв’язування даних, транзакції та збереження структурованої інформації.

Важливою перевагою SQLite є також зручність інтеграції з серверною частиною, реалізованою на Node.js. У проєкті доступ до бази даних здійснюється через відповідні модулі для роботи з SQLite, що дає змогу серверній частині виконувати запити до таблиць, отримувати статистичні дані, створювати нові записи, оновлювати інформацію та видаляти непотрібні дані. Такий підхід забезпечує взаємодію між клієнтським інтерфейсом, серверною логікою та інформаційною базою.

Разом із тим SQLite має певні обмеження. Вона не є оптимальним вибором для великих високонавантажених систем із великою кількістю одночасних користувачів і складною багатосерверною інфраструктурою. Проте для розроблюваної системи ці обмеження не є критичними, оскільки проєкт орієнтований на зберігання структурованих статистичних даних, демонстрацію функціональності та роботу в межах помірного навантаження. У разі подальшого масштабування система може бути перенесена на серверну СУБД, наприклад PostgreSQL або MySQL, без суттєвої зміни логіки предметної області.

Отже, вибір SQLite як системи управління базою даних є обґрунтованим для даного програмного продукту. Вона забезпечує простоту розгортання, підтримку реляційної структури, збереження даних в одному файлі, можливість використання зовнішніх ключів і достатню функціональність для реалізації системи обліку статистичних даних для футбольного менеджера.

2.3 Створення інформаційної бази даних

2.3.1 Створення та ініціалізація бази даних

Створення інформаційної бази є практичним етапом реалізації логічної моделі даних. У розробленій системі обліку статистичних даних для футбольного менеджера використовується SQLite, особливістю якої є

збереження всієї бази даних в одному файлі [8]. Такий підхід є зручним для локального розгортання, тестування та демонстрації системи, оскільки не потребує окремого серверного процесу бази даних.

У межах проєкту файл бази даних створюється під час відкриття підключення до SQLite. Після цього активується підтримка зовнішніх ключів і виконується SQL-файл зі схемою бази даних. Це дає змогу автоматично створити необхідні таблиці, обмеження та індекси під час запуску серверної частини системи.

```
    const DB_PATH = path.join(__dirname, "..", "..", "database", "football_manager.db");

    dbInstance = await open({
      filename: DB_PATH,
      driver: sqlite3.Database
    });

    await dbInstance.exec("PRAGMA foreign_keys = ON;");
    const schema = await fs.readFile(SCHEMA_PATH, "utf8");
    await dbInstance.exec(schema);
```

Рис. 2.2 – Фрагмент коду створення та ініціалізації бази даних

У наведеному фрагменті визначається шлях до файлу football_manager.db, відкривається підключення до бази даних, активується підтримка зовнішніх ключів і виконується файл схеми. Команда PRAGMA foreign_keys = ON є важливою, оскільки в SQLite перевірка зовнішніх ключів має бути ввімкнена окремо.

2.3.2 Створення основних таблиць

Інформаційна база системи містить таблиці для збереження даних про користувачів, команди, гравців, турніри, матчі, статистику, склади, скаутські та аналітичні звіти. Кожна таблиця відповідає окремій сутності предметної області, а реляційна структура дозволяє пов’язати ці сутності між собою [20].

Однією з основних таблиць є users, у якій зберігаються облікові записи користувачів системи. Таблиця містить персональні дані, електронну адресу, хеш пароля, роль користувача та посилання на команду.

```
CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  first_name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  email TEXT NOT NULL UNIQUE,
  password_hash TEXT NOT NULL,
  role TEXT NOT NULL CHECK (role IN ('admin', 'pending', 'coach', 'scout', 'analyst', 'player')),
  team_id INTEGER,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE SET NULL
);
```

Рис. 2.3 – Фрагмент SQL-коду створення таблиці користувачів

Поле id є первинним ключем таблиці. Поле email має обмеження UNIQUE, що унеможливлює повторну реєстрацію користувачів з однаковою адресою. Поле role має обмеження CHECK, завдяки якому система допускає лише визначені ролі: admin, pending, coach, scout, analyst і player.

Важливою таблицею є matches, яка зберігає дані про матчі, турніри та команди-учасниці. У ній передбачено окремі поля для домашньої та виїзної команди.

У таблиці matches поле tournament_id визначає турнір, у межах якого проводиться матч. Поля home_team_id і away_team_id посилаються на таблицю teams та визначають домашню і виїзну команди. Для поля status задано обмеження допустимих значень, що дозволяє контролювати стан матчу.

```
CREATE TABLE IF NOT EXISTS matches (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tournament_id INTEGER NOT NULL,
  home_team_id INTEGER NOT NULL,
  away_team_id INTEGER NOT NULL,
  matchday INTEGER NOT NULL DEFAULT 1,
  match_date TEXT NOT NULL,
  venue TEXT,
  status TEXT NOT NULL DEFAULT 'scheduled'
    CHECK (status IN ('scheduled', 'live', 'finished', 'postponed')),
```

```
        home_score INTEGER,
        away_score INTEGER,
        notes TEXT DEFAULT '',
        created_at TEXT DEFAULT CURRENT_TIMESTAMP,
        updated_at TEXT DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (tournament_id) REFERENCES tournaments(id) ON
DELETE CASCADE,
        FOREIGN KEY (home_team_id) REFERENCES teams(id) ON DELETE
CASCADE,
        FOREIGN KEY (away_team_id) REFERENCES teams(id) ON DELETE
CASCADE
      );
```

Рис. 2.4 – Фрагмент SQL-коду створення таблиці матчів

2.3.3 Зв’язки та обмеження цілісності

Для забезпечення цілісності даних у базі використовуються первинні та зовнішні ключі. Первинний ключ однозначно ідентифікує запис у таблиці, а зовнішній ключ встановлює зв’язок між таблицями та забезпечує посилальну цілісність [21]. Завдяки цьому система може коректно пов’язувати гравців із командами, матчі з турнірами, звіти з користувачами, а статистичні записи з відповідними гравцями або матчами.

У базі також використовуються обмеження NOT NULL, UNIQUE, CHECK, ON DELETE CASCADE і ON DELETE SET NULL. Обмеження NOT NULL визначає обов’язкові поля, UNIQUE запобігає дублюванню унікальних значень, а CHECK контролює допустимі значення для ролей, статусів матчів і ролей гравців у складі. Правило ON DELETE CASCADE застосовується для автоматичного видалення залежних записів, а ON DELETE SET NULL дозволяє зберегти запис, але прибрати посилання на видалену сутність.

Окремо слід зазначити структуру складів команди. Таблиця lineups зберігає загальну інформацію про склад, а таблиця lineup_players деталізує, які саме футболісти входять до складу та яку роль виконують.

```
      CREATE TABLE IF NOT EXISTS lineups (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        team_id INTEGER NOT NULL,
```

```
        title TEXT NOT NULL,
        formation TEXT NOT NULL,
        created_by INTEGER NOT NULL,
        created_at TEXT DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE
CASCADE,
        FOREIGN KEY (created_by) REFERENCES users(id) ON DELETE
CASCADE
    );
```

Рис. 2.5 – Фрагмент SQL-коду створення таблиці складів

У таблиці lineups поле team_id визначає команду, для якої створено склад, а поле created_by посилається на користувача, який створив відповідний запис. Це дозволяє зберігати не лише сам склад, а й інформацію про його автора.

2.3.4 Індекси та оптимізація запитів

Для підвищення швидкодії роботи з базою даних використовуються індекси. Індекс у базі даних є спеціальною допоміжною структурою, яка прискорює пошук записів за певними полями. Його доцільно створювати для тих атрибутів, які часто використовуються під час фільтрації, сортування або з’єднання таблиць [23]. У розробленій системі такими полями є ідентифікатори команд, турнірів, гравців, матчів і дата проведення матчу.

```
    CREATE INDEX IF NOT EXISTS idx_players_team_id ON
players(team_id);
    CREATE INDEX IF NOT EXISTS idx_matches_tournament_id ON
matches(tournament_id);
    CREATE INDEX IF NOT EXISTS idx_matches_home_team_id ON
matches(home_team_id);
    CREATE INDEX IF NOT EXISTS idx_matches_away_team_id ON
matches(away_team_id);
    CREATE INDEX IF NOT EXISTS idx_matches_match_date ON
matches(match_date);
```

Рис. 2.6 – Фрагмент SQL-коду створення індексів

Індекс idx_players_team_id прискорює отримання списку гравців певної команди. Індекси idx_matches_home_team_id і idx_matches_away_team_id дають змогу швидше знаходити матчі, у яких бере участь конкретна команда. Індекс idx_matches_match_date корисний для відображення календаря матчів і

сортування ігор за датою. Таким чином, індексація допомагає зробити роботу системи швидшою під час виконання типових запитів.

2.3.5 Наповнення, оновлення структури та службові процеси

Після створення таблиць база даних наповнюється початковими демонстраційними даними. До них належать команди, користувачі різних ролей, гравці, турніри, турнірні таблиці, склади, скаутські й аналітичні звіти. Таке наповнення потрібне для перевірки працездатності системи та демонстрації основних можливостей програмного продукту.

Для надійного додавання початкових даних використовується транзакція. Транзакція дозволяє виконати групу операцій як єдине ціле: якщо всі запити виконуються успішно, зміни підтверджуються командою COMMIT; якщо виникає помилка, виконується ROLLBACK, і база повертається до попереднього стану [24].

```
        try {
          await db.exec("BEGIN");

          await db.run(
            `INSERT INTO teams (id, name, city, short_name, founded_year, stadium,
coach_name)
             VALUES (?, ?, ?, ?, ?, ?, ?)`,
            team.id,
            team.name,
            team.city,
            team.short_name,
            team.founded_year,
            team.stadium,
            team.coach_name
          );

          await db.exec("COMMIT");
        } catch (error) {
          await db.exec("ROLLBACK");
          throw error;
        }
```

Рис. 2.7 – Фрагмент коду початкового наповнення бази даних

У системі також передбачено прості процеси оновлення структури бази даних. Якщо в таблиці відсутнє певне поле, воно може бути додане за допомогою

команди ALTER TABLE. Такий підхід дозволяє розширювати структуру бази без повного видалення вже наявних даних.

```
        const columns = await db.all(`PRAGMA table_info(${tableName})`);
        const hasColumn = columns.some((column) => column.name === columnName);

        if (!hasColumn) {
          await db.exec(`ALTER TABLE ${tableName} ADD COLUMN ${columnName} ${definition}`);
        }
```

Рис. 2.8 – Фрагмент коду перевірки та додавання нового поля до таблиці

Збережені процедури в системі не використовуються, оскільки SQLite не підтримує їх у класичному вигляді, як серверні СУБД. Тригери також не застосовуються в поточній реалізації, оскільки необхідна логіка перевірки прав, оновлення даних, обробки запитів і формування результатів реалізована на рівні серверної частини програмного застосунку. База даних у цьому проєкті відповідає за надійне зберігання структурованої інформації, а бізнес-логіка виконується програмними модулями системи.

Отже, інформаційна база системи створюється як SQLite-файл і містить набір взаємопов’язаних таблиць, первинних і зовнішніх ключів, обмежень цілісності, індексів і початкових даних. Така структура забезпечує збереження інформації про користувачів, команди, гравців, матчі, турніри, статистику, склади та звіти, а також створює основу для взаємодії бази даних із серверною частиною системи.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення визначає склад основних частин системи та характер взаємодії між ними. Розроблена система обліку статистичних даних для футбольного менеджера має веборієнтовану архітектуру, у якій клієнтська частина взаємодіє із серверною частиною через HTTP-запити, а серверна частина забезпечує обробку даних, перевірку доступу, виконання бізнес-логіки та роботу з базою даних.

Для наочного подання організаційної структури програмного забезпечення використано діаграму пакетів, яка у UML застосовується для відображення пакетів системи та залежностей між ними [26]. У межах розроблюваної системи виділено кілька основних пакетів: клієнтський інтерфейс, механізм зовнішнього доступу через Cloudflare Tunnel, серверну частину з бізнес-логікою та пакет бази даних.

Клієнтська частина системи реалізована засобами HTML, CSS і JavaScript. Вона відповідає за відображення інтерфейсу користувача, роботу зі сторінками авторизації, реєстрації та основної панелі системи. Через клієнтський інтерфейс користувач може переглядати дані про команду, гравців, матчі, турніри, статистику, звіти та виконувати дії відповідно до своєї ролі.

Серверна частина реалізована на платформі Node.js із використанням фреймворку Express [6], [7]. Вона містить API-маршрути, middleware, контролери, сервіси та модуль конфігурації бази даних. API-маршрути приймають запити від клієнтської частини, middleware використовується для авторизації та обробки помилок, контролери відповідають за обробку HTTP-запитів, а сервіси реалізують основну бізнес-логіку системи. Модуль конфігурації бази даних забезпечує підключення до SQLite, виконання схеми бази даних, запуск початкового наповнення та службових оновлень структури.

Для зовнішнього доступу до локального сервера може використовуватися Cloudflare Tunnel. У такому випадку Cloudflare надає тимчасове HTTPS-посилання в домені *.trycloudflare.com і проксує запити до локального сервера, що працює на localhost:3000 [25]. Це дає змогу демонструвати роботу системи без повноцінного розгортання на віддаленому хостингу. Водночас Cloudflare Tunnel у цій структурі розглядається як зовнішній механізм доступу, а не як частина програмного коду системи.

Збереження інформації здійснюється у SQLite-базі даних football_manager.db [8]. Структура таблиць описується у файлі schema.sql, а початкове наповнення даними виконується через файли seeds.js та expandedSeeds.js. База даних містить інформацію про користувачів, команди, гравців, матчі, турніри, статистику, склади, скаутські та аналітичні звіти.

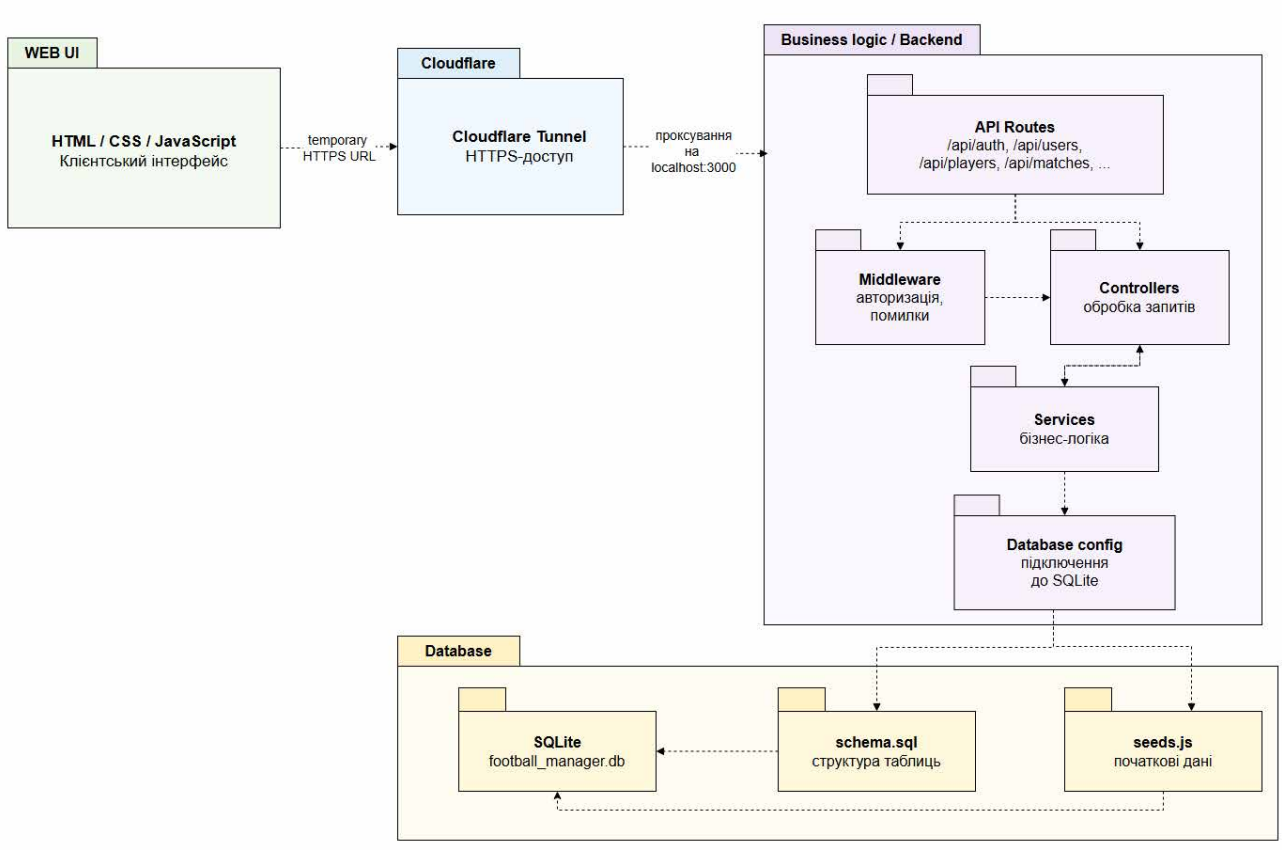

Рис. 3.1 – Діаграма пакетів програмної системи

На діаграмі пакетів показано архітектурну організацію програмної системи. Пакет WEB UI містить клієнтський інтерфейс, реалізований засобами

HTML, CSS і JavaScript. Пакет Cloudflare відображає механізм зовнішнього доступу через Cloudflare Tunnel, який надає тимчасове HTTPS-посилання та проксує запити до локального сервера. Пакет Business logic / Backend містить основні компоненти серверної частини: API-маршрути, middleware, контролери, сервіси та модуль конфігурації бази даних. Пакет Database містить SQLite-файл бази даних, SQL-схему та файли початкового наповнення.

Таким чином, організаційна структура програмного забезпечення побудована за принципом розподілу відповідальності між окремими частинами системи. Клієнтська частина забезпечує взаємодію користувача з інтерфейсом, серверна частина виконує обробку запитів і бізнес-логіку, Cloudflare Tunnel може використовуватися для зовнішнього доступу до локального застосунку, а база даних відповідає за збереження структурованої інформації. Така організація спрощує супровід системи, полегшує розуміння її архітектури та створює основу для подальшого розвитку програмного продукту.

3.2 Вибір інструментарію для створення ППЗ

Вибір інструментарію для створення прикладного програмного забезпечення здійснювався з урахуванням призначення системи, її архітектури та функціональних вимог. Оскільки розроблювана система обліку статистичних даних для футбольного менеджера є веборієнтованою, доцільним було використання технологій, які забезпечують роботу клієнтської частини, серверної логіки, бази даних, авторизації користувачів і зовнішнього доступу до застосунку.

Для створення системи було використано такі основні засоби:

1. HTML, CSS і JavaScript — для реалізації клієнтської частини;
2. Node.js та Express — для створення серверної частини й API;
3. SQLite — для зберігання структурованих даних;
4. bcrypt і JWT — для захисту облікових записів та автентифікації;

5. Cloudflare Tunnel — для демонстрації доступу до локального вебзастосунку через тимчасове HTTPS-посилання.

Для реалізації клієнтської частини було використано HTML, CSS і JavaScript. HTML застосовується для побудови структури вебсторінок, CSS — для оформлення інтерфейсу, а JavaScript — для реалізації динамічної взаємодії користувача із системою [27]–[29]. У межах проєкту ці технології використовуються для сторінок входу, реєстрації та основної панелі системи, де користувач може працювати з командами, гравцями, матчами, турнірами, статистикою, складами та звітами. Вибір цих засобів є доцільним, оскільки вони є базовими технологіями веброзробки та не потребують додаткового клієнтського фреймворку для реалізації функціональності даної системи.

Серверна частина програмного продукту реалізована на платформі Node.js. Node.js дає змогу виконувати JavaScript-код на сервері та створювати мережеві застосунки, що обробляють запити від клієнтської частини [6]. Для побудови вебсерверу та API використано фреймворк Express, який забезпечує зручну організацію маршрутів, middleware та обробку HTTP-запитів [7]. У системі Express використовується для створення маршрутів авторизації, роботи з користувачами, гравцями, командами, турнірами, матчами, складами, звітами та інформаційною панеллю.

Для збереження даних було обрано SQLite. Ця СУБД є вбудованою, не потребує окремого серверного процесу та зберігає базу даних в одному файлі [8]. Для даного проєкту це є перевагою, оскільки система може бути легко запущена локально, протестована та продемонстрована без складного налаштування інфраструктури. SQLite використовується для зберігання інформації про користувачів, команди, гравців, матчі, турніри, статистику, склади, скаутські та аналітичні звіти.

Для захисту облікових записів у системі застосовано бібліотеку bcrypt, яка використовується для хешування паролів перед збереженням у базі даних [30]. Це дозволяє не зберігати паролі у відкритому вигляді та підвищує безпеку системи. Для автентифікації користувачів використовується механізм JSON Web

Token. JWT є відкритим стандартом, який застосовується для компактного передавання інформації між сторонами у вигляді JSON-об’єкта [31]. У межах системи токен використовується для підтвердження особи користувача та перевірки його доступу до захищених маршрутів.

Для організації зовнішнього доступу до локального сервера може використовуватися Cloudflare Tunnel. Цей інструмент створює тимчасове HTTPS-посилання та проксує запити до локального застосунку, що працює на localhost:3000 [25]. Його використання є зручним для демонстрації роботи системи без повноцінного розгортання на віддаленому сервері. При цьому Cloudflare Tunnel не є частиною програмного коду, а виступає допоміжним засобом доступу до вебзастосунку.

Отже, обраний інструментарій відповідає архітектурі та призначенню розроблюваної системи. HTML, CSS і JavaScript забезпечують реалізацію клієнтського інтерфейсу, Node.js та Express відповідають за серверну частину й API, SQLite використовується для зберігання структурованих даних, а bcrypt і JWT забезпечують базовий рівень захисту облікових записів і доступу до системи. Такий набір технологій є достатнім для реалізації функціональних можливостей системи обліку статистичних даних для футбольного менеджера та подальшого розвитку програмного продукту.

3.3 Алгоритмізація та програмування програмних модулів

Алгоритмізація та програмування програмних модулів є одним із ключових етапів створення прикладного програмного забезпечення, оскільки саме на цьому етапі визначається логіка роботи системи, порядок обробки запитів, взаємодія користувача з інтерфейсом, перевірка прав доступу та робота з базою даних. У розробленій системі обліку статистичних даних для футбольного менеджера програмна логіка побудована за модульним принципом. Серверна частина містить окремі маршрути, контролери, сервіси, middleware та

конфігурацію бази даних. Така організація дає змогу розділити відповідальність між частинами програмного коду та спростити подальшу підтримку системи.

Загальна логіка роботи програмної системи полягає в тому, що користувач взаємодіє з клієнтським інтерфейсом, реалізованим засобами HTML, CSS і JavaScript. Дії користувача перетворюються на HTTP-запити до серверної частини, яка працює на Node.js та Express [6], [7]. Сервер приймає запит через відповідний API-маршрут, виконує перевірку авторизації, передає запит до контролера, а контролер звертається до сервісу, де реалізована бізнес-логіка. Після цього сервіс взаємодіє з базою даних SQLite, отримує або змінює необхідні дані та повертає результат назад до контролера. Контролер формує відповідь у форматі JSON, яку клієнтська частина використовує для оновлення інтерфейсу.

У серверній частині системи основні API-маршрути підключаються у файлі застосунку. Вони відповідають за авторизацію, роботу з користувачами, командами, гравцями, турнірами, матчами, складами, звітами та інформаційною панеллю.

```
app.use("/api/auth", authRoutes);
app.use("/api/teams", teamRoutes);
app.use("/api/players", playerRoutes);
app.use("/api/tournaments", tournamentRoutes);
app.use("/api/matches", matchRoutes);
app.use("/api/lineups", lineupRoutes);
app.use("/api/reports", reportRoutes);
app.use("/api/dashboard", dashboardRoutes);
app.use("/api/users", userRoutes);
```

Рис. 3.2 – Фрагмент коду підключення API-маршрутів системи

У наведеному фрагменті показано, що кожна група функцій винесена в окремий маршрут. Наприклад, /api/auth відповідає за реєстрацію та авторизацію, /api/players — за роботу з гравцями, /api/matches — за матчі, /api/lineups — за склади, а /api/reports — за скаутські та аналітичні звіти.

Такий підхід робить програмну структуру зрозумілою та дозволяє розвивати окремі частини системи незалежно одна від одної.

```
async function registerUser(payload) {
  const db = getDb();

  if (await getUserByEmail(payload.email)) {
    throw new AppError("A user with this email already exists.", 409);
  }

  const result = await db.run(
    `INSERT INTO users (first_name, last_name, email, password_hash, role,
team_id)
     VALUES (?, ?, ?, ?, 'pending', ?)`,
    payload.firstName.trim(),
    payload.lastName.trim(),
    payload.email.trim().toLowerCase(),
    await bcrypt.hash(payload.password, 10),
    normalizeOptionalNumber(payload.teamId)
  );

  return getUserById(result.lastID);
}
```

Рис. 3.3 – Фрагмент коду реєстрації користувача

Алгоритм реєстрації користувача починається з введення персональних даних у формі реєстрації. Після надсилання форми клієнтська частина передає дані на серверний маршрут реєстрації. На сервері виконується перевірка коректності даних, після чого сервіс авторизації перевіряє, чи не існує вже користувач із такою електронною адресою. Якщо користувач уже існує, система повертає повідомлення про помилку. Якщо адреса вільна, пароль хешується за допомогою bcrypt, а новий обліковий запис створюється з роллю pending, тобто користувач очікує підтвердження адміністратором [30].

У цьому фрагменті реалізовано основні дії алгоритму реєстрації: перевірку наявності користувача, хешування пароля, нормалізацію даних і створення нового запису в таблиці users. Важливо, що новий користувач не отримує одразу повний доступ до системи, а створюється зі статусом pending. Це відповідає

логіці системи, за якої адміністратор повинен підтвердити користувача та призначити йому відповідну роль.

Алгоритм авторизації користувача виконується після введення електронної адреси та пароля. Сервер перевіряє, чи існує користувач із вказаною адресою, після чого порівнює введений пароль із хешем, збереженим у базі даних. Якщо користувача не знайдено або пароль не збігається, система повертає помилку. Якщо обліковий запис має роль pending, вхід блокується до підтвердження адміністратором. Якщо всі перевірки пройдено успішно, сервер формує JWT-токен, який використовується для подальшої автентифікації користувача [31].

У наведеному коді реалізовано перевірку облікових даних користувача. Пароль не порівнюється у відкритому вигляді, а перевіряється через bcrypt.compare, що підвищує безпеку системи. Також із відповіді видаляється поле password_hash, тому хеш пароля не передається на клієнтську частину.

Після успішної авторизації система створює JWT-токен. У токен записуються ідентифікатор користувача та його роль. Надалі цей токен передається в заголовку запитів і використовується middleware для перевірки доступу.

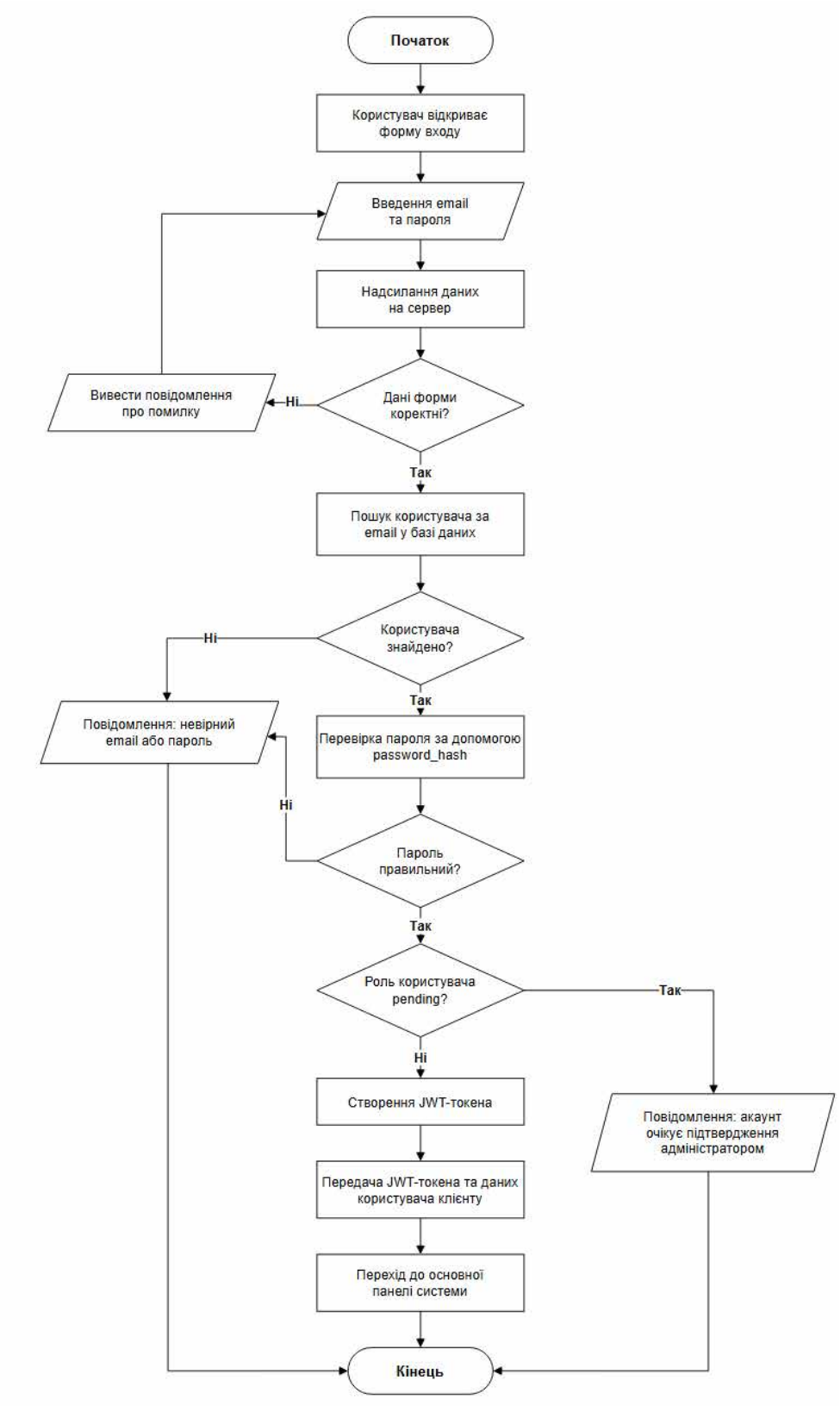

Рис. 3.4 – Блок-схема алгоритму авторизації користувача

```
async function authenticateUser(email, password) {
  const db = getDb();
  const user = await db.get(
    "SELECT * FROM users WHERE email = ?",
    String(email).trim().toLowerCase()
  );
  if (!user || !(await bcrypt.compare(password, user.password_hash))) {
    throw new AppError("Invalid email or password.", 401);
  }
  if (user.role === "pending") {
    throw new AppError("Your account is awaiting administrator approval.",
403);
  }
  const { password_hash, ...publicUser } = user;
  return publicUser;
}
```

Рис. 3.5 – Фрагмент коду авторизації користувача

```
function createToken(user) {
  return jwt.sign(
    {     id: user.id,
      role: user.role
    },
    JWT_SECRET,
    { expiresIn: "8h" }
  );}
```

Рис. 3.6 – Фрагмент коду створення JWT-токена

Алгоритм визначення ролі користувача та надання доступу реалізовано через middleware. Спочатку система перевіряє наявність токена в заголовку запиту. Потім токен розшифровується, після чого за ідентифікатором отримуються актуальні дані користувача з бази. Якщо користувача не знайдено або токен недійсний, запит відхиляється. Якщо користувач існує, його дані записуються в req.user, після чого запит може бути переданий далі до контролера.

```
const authenticateToken = asyncHandler(async (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
```

```
    if (!token) {
      throw new AppError("Authentication token is missing.", 401);
    }

    const decoded = jwt.verify(token, JWT_SECRET);
    const user = await authService.getUserById(decoded.id);

    if (!user) {
      throw new AppError("User not found.", 401);
    }

    req.user = user;
    next();
  });
```

Рис. 3.7 – Фрагмент коду перевірки токена користувача

Для розмежування доступу використовується функція authorizeRoles. Вона приймає перелік ролей, яким дозволено виконання певної дії. Якщо роль поточного користувача не входить до дозволених, система повертає помилку доступу. Саме цей механізм використовується для обмеження можливостей адміністратора, тренера, аналітика, скаута та гравця.

```
  function authorizeRoles(...roles) {
    return (req, res, next) => {
      if (!req.user || !roles.includes(req.user.role)) {
        return next(new AppError("You do not have permission to access this resource.", 403));
      }

      return next();
    };
  }
```

Рис. 3.8 – Фрагмент коду перевірки ролі користувача

Наприклад, маршрути адміністрування користувачів доступні лише адміністратору. Це реалізовано у маршруті /api/users, де спочатку виконується перевірка токена, а потім перевірка ролі admin.

```
router.use(authenticateToken, authorizeRoles("admin"));


router.get("/", userController.listUsers);
router.patch("/:id", userController.updateUser);
router.delete("/:id", userController.deleteUser);
```

Рис. 3.9 – Фрагмент коду обмеження доступу до адміністрування акаунтів

Адміністратор має можливість переглядати список користувачів, змінювати їхні ролі, прив’язувати до команд і видаляти акаунти. При цьому система містить додаткові перевірки безпеки. Наприклад, адміністратор не може видалити власний акаунт або забрати у себе роль адміністратора. Також у разі видалення адміністратора перевіряється, щоб у системі залишився принаймні один обліковий запис із роллю admin.

Алгоритм роботи тренера з формуванням складу команди побудований навколо маршруту /api/lineups. Цей маршрут доступний тільки користувачам із роллю coach. Тренер обирає схему, стартових гравців, запасних і резервних футболістів. Після надсилання даних сервер перевіряє, чи всі вибрані гравці належать до відповідної команди. Якщо до складу потрапив гравець іншої команди, система повертає помилку. Якщо дані коректні, склад зберігається у таблиці lineups, а окремі футболісти складу — у таблиці lineup_players.

Процес збереження складу виконується в межах транзакції. Це важливо, оскільки створення складу складається з кількох операцій: спочатку створюється запис у таблиці lineups, а потім додаються записи в lineup_players. Якщо під час будь-якої операції виникає помилка, виконується ROLLBACK, і база даних повертається до попереднього стану. Якщо всі операції успішні, виконується COMMIT.

У цьому алгоритмі важливим є те, що система перевіряє належність гравців до команди ще до збереження складу. Завдяки цьому тренер не може випадково

додати до складу футболіста з іншої команди. Також використання транзакції забезпечує цілісність даних: склад не буде створений частково, якщо під час збереження одного з гравців виникне помилка.

Алгоритм роботи скаута зі звітами реалізований через маршрути /api/reports/scout. Отримувати скаутські звіти можуть тренер і скаут, але створювати нові скаутські звіти може лише користувач із роллю scout. Перед збереженням звіту контролер перевіряє коректність введених даних. Після цього сервіс записує звіт у таблицю scout_reports, де зберігаються гравець, автор звіту, контекст матчу, сильні та слабкі сторони, тактична відповідність, рівень ризику, потенційний рейтинг, орієнтовна вартість і рекомендації.

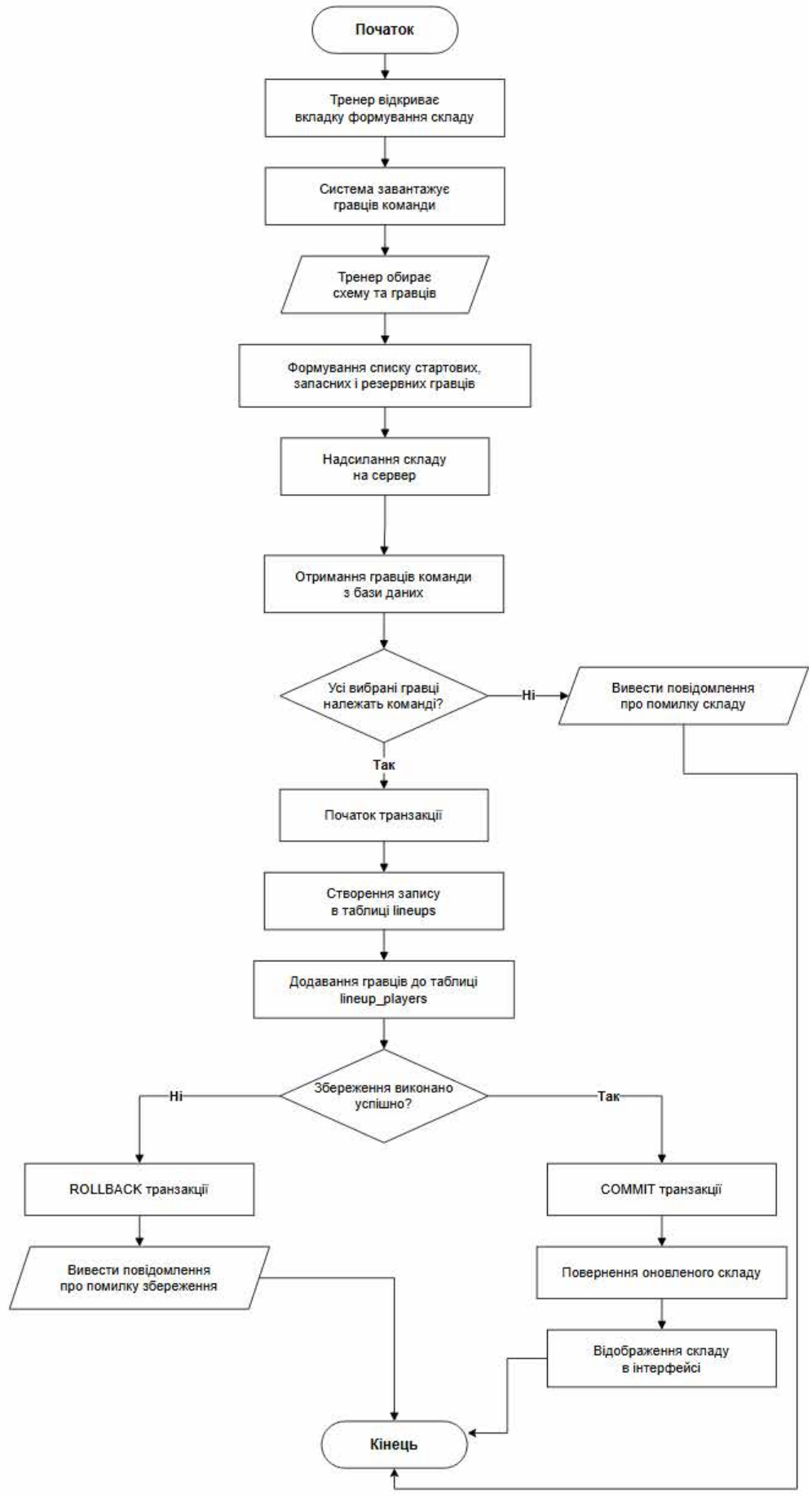

Рис. 3.10 – Блок-схема алгоритму формування складу команди

```
async function saveLineup(payload, createdBy) {
  const db = getDb();
  const teamId = Number(payload.teamId);
  const players = await db.all("SELECT id FROM players WHERE team_id =
?", teamId);
  const allowedIds = new Set(players.map((p) => p.id));
  const entries = [...payload.starters, ...payload.bench, ...payload.reserve];
  if (entries.some((e) => !allowedIds.has(e.playerId))) {
    throw new AppError("Player is not in this team.", 400);
  }
  const lineup = await db.run(
    `INSERT INTO lineups (team_id, formation, created_by)
     VALUES (?, ?, ?)`,
    teamId,
    payload.formation,
    createdBy
  );

  for (const entry of entries) {
    await db.run(
      `INSERT INTO lineup_players (lineup_id, player_id, squad_role)
       VALUES (?, ?, ?)`,
      lineup.lastID,
      entry.playerId,
      entry.squad_role
    );
  }
  return getCurrentLineup(teamId);
}
```

Рис. 3.11 – Фрагмент коду збереження складу команди

```
router.get("/scout", authorizeRoles("coach", "scout"),
reportController.getScoutReports);

router.post("/scout", authorizeRoles("scout"),
reportController.createScoutReport);

router.get("/analyst", authorizeRoles("coach", "analyst"),
reportController.getAnalystReports);

router.post("/analyst", authorizeRoles("analyst"),
reportController.createAnalystReport);
```

Рис. 3.12 – Фрагмент коду розмежування доступу до звітів

Алгоритм створення скаутського звіту передбачає зчитування даних із форми, перевірку обов'язкових полів, визначення користувача-скаута та запис інформації до бази даних. Після успішного створення система повертає оновлений список звітів.

Алгоритм роботи аналітика побудований подібним чином, але орієнтований на аналітичні матеріали. Аналітик може створювати аналітичні звіти, які містять тему, період, знімок метрик, аналіз, висновки щодо тенденцій, фактори ризику, план дій і загальні висновки. Тренер має право переглядати аналітичні звіти, але створення таких звітів доступне лише аналітику.

Окремим напрямом роботи аналітика є редагування матчів. У системі маршрут оновлення матчу доступний лише користувачу з роллю analyst. Аналітик може змінювати дату, місце проведення, статус, рахунок і примітки до матчу. Після оновлення, якщо матч має статус finished, система може створити статистику матчу, якщо вона ще не була створена.

```
async function createScoutReport(payload, userId) {
  const db = getDb();
  const result = await db.run(
    `INSERT INTO scout_reports (
      player_id, scout_user_id, match_context, strengths, weaknesses,
      tactical_fit, risk_level, potential_rating, estimated_value, recommendations
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
    Number(payload.player_id),
    userId,
    String(payload.match_context || "").trim(),
    String(payload.strengths).trim(),
```

```
    String(payload.weaknesses).trim(),
    String(payload.tactical_fit || "").trim(),
    String(payload.risk_level || "medium").trim(),
    Number(payload.potential_rating || 0),
    Number(payload.estimated_value || 0),
    String(payload.recommendations).trim()
  );
  return result.lastID;
}
```

Рис. 3.13 – Фрагмент коду створення скаутського звіту

```
async function createAnalystReport(payload, userId) {
  const db = getDb();
  const result = await db.run(
    `INSERT INTO analyst_reports (
      analyst_user_id, subject, period, metrics_snapshot, analysis,
      trend_summary, risk_factors, action_plan, conclusions
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)`,
    userId,
    String(payload.subject).trim(),
    String(payload.period || "").trim(),
    String(payload.metrics_snapshot || "").trim(),
    String(payload.analysis).trim(),
    String(payload.trend_summary || "").trim(),
    String(payload.risk_factors || "").trim(),
    String(payload.action_plan || "").trim(),
    String(payload.conclusions).trim()  );
  return result.lastID; }
```

Рис. 3.14 – Фрагмент коду створення аналітичного звіту

```
router.get("/", matchController.listMatches);
router.get("/my-team", matchController.listMyTeamMatches);
router.get("/:id", matchController.getMatch);
router.put("/:id", authorizeRoles("analyst"), matchController.updateMatch);
```

Рис. 3.15 – Фрагмент коду маршруту редагування матчу аналітиком

Робота backend із базою даних SQLite реалізована через модуль конфігурації бази даних. Під час запуску серверної частини створюється або відкривається файл football_manager.db, вмикається підтримка зовнішніх ключів, виконується схема бази даних, запускаються службові оновлення структури та початкове наповнення даними. Таким чином, база даних ініціалізується автоматично під час старту системи.

```
dbInstance = await open({
  filename: DB_PATH,
  driver: sqlite3.Database
});
await dbInstance.exec("PRAGMA foreign_keys = ON;");

const schema = await fs.readFile(SCHEMA_PATH, "utf8");
await dbInstance.exec(schema);
await runMigrations(dbInstance);
await seedDatabase(dbInstance);
```

Рис. 3.16 – Фрагмент коду ініціалізації SQLite-бази даних

Взаємодія з базою даних виконується через SQL-запити. Для отримання одного запису використовується db.get, для отримання списку записів — db.all, а для додавання, оновлення або видалення записів — db.run. Такий підхід використовується в сервісах системи, які відповідають за користувачів, гравців, матчі, склади та звіти. Наприклад, під час отримання матчу система виконує SQL-запит із приєднанням таблиць турнірів і команд, щоб повернути не лише ідентифікатори, а й назви команд і турніру.

Крім звичайних операцій вибірки та збереження, у системі реалізовані службові процеси оновлення структури бази даних. Вони перевіряють наявність потрібних полів у таблицях і за потреби додають їх. Це дає змогу оновлювати структуру бази без повного видалення вже наявних даних.

```
async function ensureColumn(db, tableName, columnName, definition) {
  const columns = await db.all(`PRAGMA table_info(${tableName})`);
  const hasColumn = columns.some((column) => column.name === columnName);

  if (!hasColumn) {
    await db.exec(`ALTER TABLE ${tableName} ADD COLUMN ${columnName} ${definition}`);
  }
}
```

Рис. 3.17 – Фрагмент коду службового оновлення структури бази даних

Отже, алгоритмізація програмних модулів системи побудована на послідовній взаємодії клієнтської частини, API-маршрутів, middleware, контролерів, сервісів і бази даних. Авторизація та реєстрація забезпечують контроль доступу користувачів, рольова модель визначає набір доступних функцій, тренер працює зі складами команди, скаут і аналітик формують відповідні звіти, а SQLite забезпечує збереження всієї структурованої

інформації. Такий підхід дозволяє реалізувати основні функціональні можливості системи та підтримувати цілісність даних під час роботи з футбольною статистикою

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмної системи обліку статистичних даних для футбольного менеджера було проведено з метою перевірки коректності роботи основних функціональних можливостей, стабільності взаємодії між клієнтською частиною, серверною логікою та базою даних SQLite, а також відповідності реалізованого функціоналу поставленим вимогам.

У процесі тестування було перевірено основні сценарії роботи користувачів системи. Насамперед перевірялася авторизація користувача: система повинна коректно приймати email і пароль, знаходити відповідний обліковий запис у базі даних, перевіряти пароль та надавати доступ лише підтвердженим користувачам. У разі введення неправильних даних система відображає повідомлення про помилку, а користувачі зі статусом pending не отримують доступу до основних функцій до підтвердження адміністратором.

Окремо було протестовано процес реєстрації нового користувача. Після заповнення реєстраційної форми в базі даних створюється новий обліковий запис із початковою роллю pending. Такий підхід дозволяє адміністратору контролювати доступ до системи та призначати користувачам відповідні ролі: тренера, аналітика, скаута або гравця.

Також було перевірено роботу адміністративної частини системи. Адміністратор має можливість переглядати список облікових записів, підтверджувати нових користувачів, змінювати їхні ролі та видаляти акаунти за потреби. Під час тестування було встановлено, що зміни коректно зберігаються в базі даних і впливають на доступ користувача до відповідних розділів системи.

Для ролі тренера перевірялася робота з командною інформацією, перегляд гравців, статистики, звітів та формування складу команди. У процесі тестування

було підтверджено, що тренер бачить лише гравців своєї команди, може сформувати склад, обрати футболістів для стартового складу, запасу та резерву, після чого дані зберігаються в базі даних і відображаються в інтерфейсі.

Окрему увагу було приділено перевірці роботи з матчами та турнірами. У системі передбачено перегляд списку матчів, поділ матчів на заплановані та завершені, а також відображення матчів у межах турнірів. Для завершених матчів перевірялася можливість перегляду статистики матчу, зокрема володіння м’ячем, очікуваних голів та інших показників. Для аналітика було перевірено можливість редагування даних матчу, зокрема оновлення рахунку та статусу матчу.

Крім того, було протестовано роботу зі звітами. Скаут має можливість формувати скаутські звіти щодо гравців, а аналітик створює аналітичні звіти за результатами матчів і статистичних показників. Тренер, у свою чергу, може переглядати сформовані звіти для подальшого використання в роботі з командою.

Для демонстрації результатів тестування доцільно додати моментальні знімки основних екранів програмного продукту.

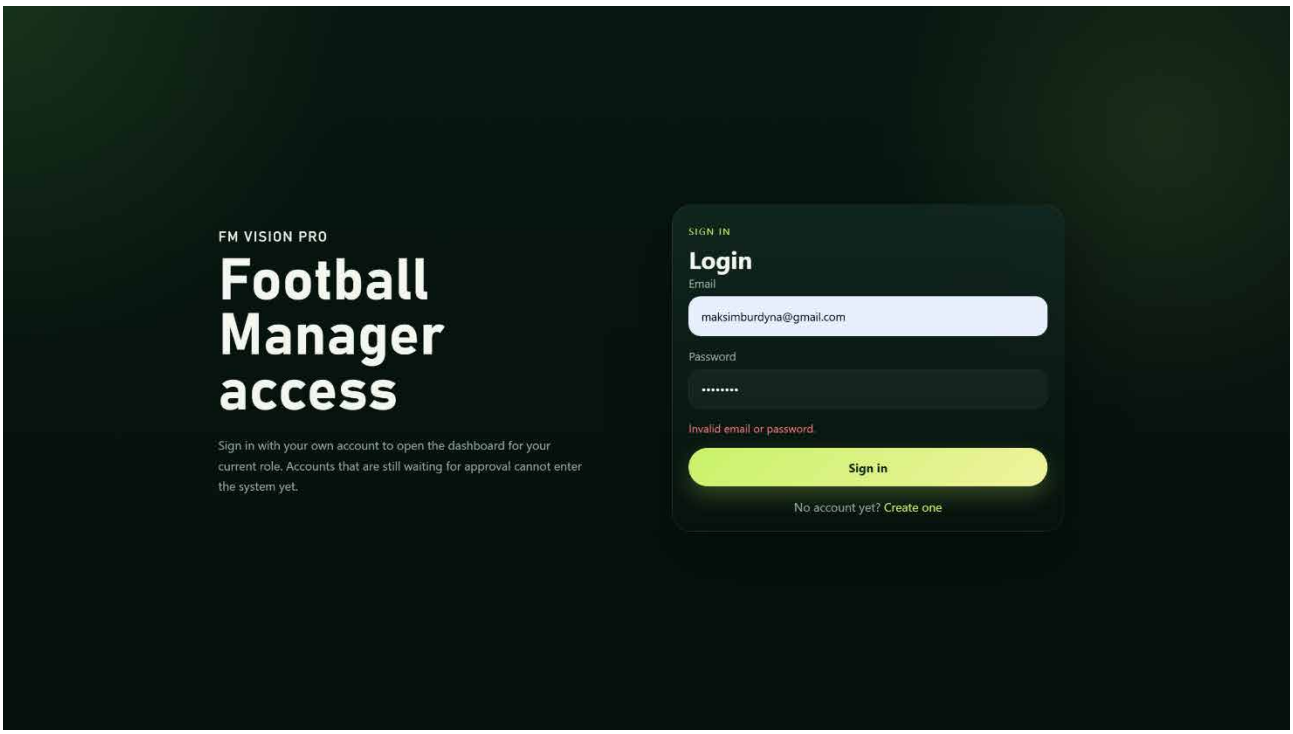

Рис. 4.1 – Форма авторизації користувача

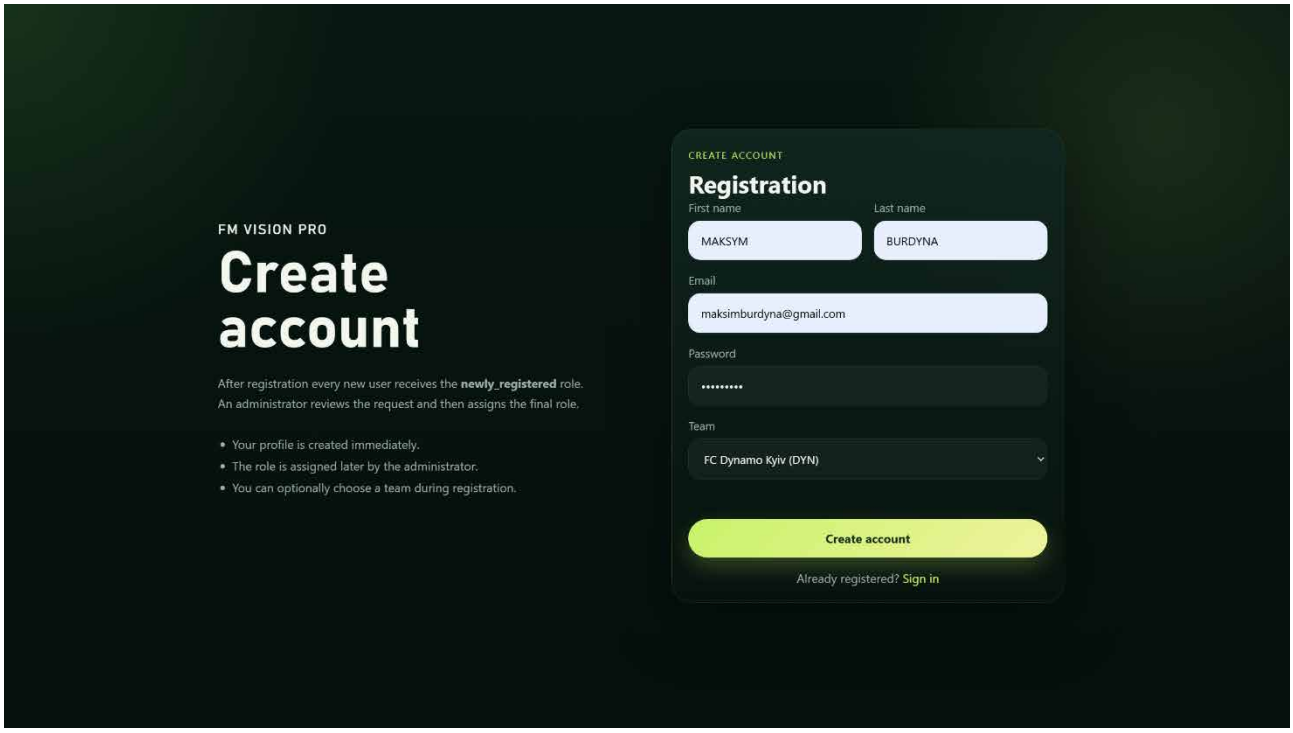

Рис. 4.2 – Форма реєстрації нового користувача

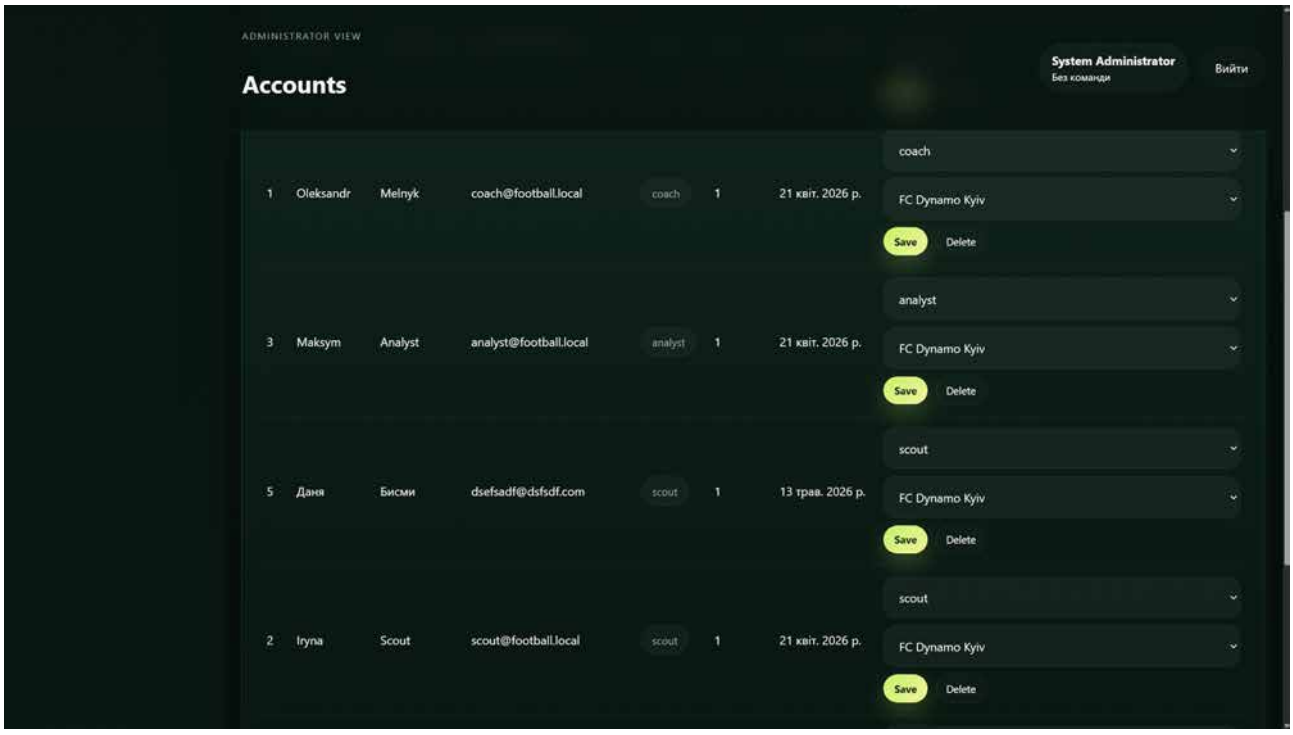

Рис. 4.3 – Панель адміністрування облікових записів

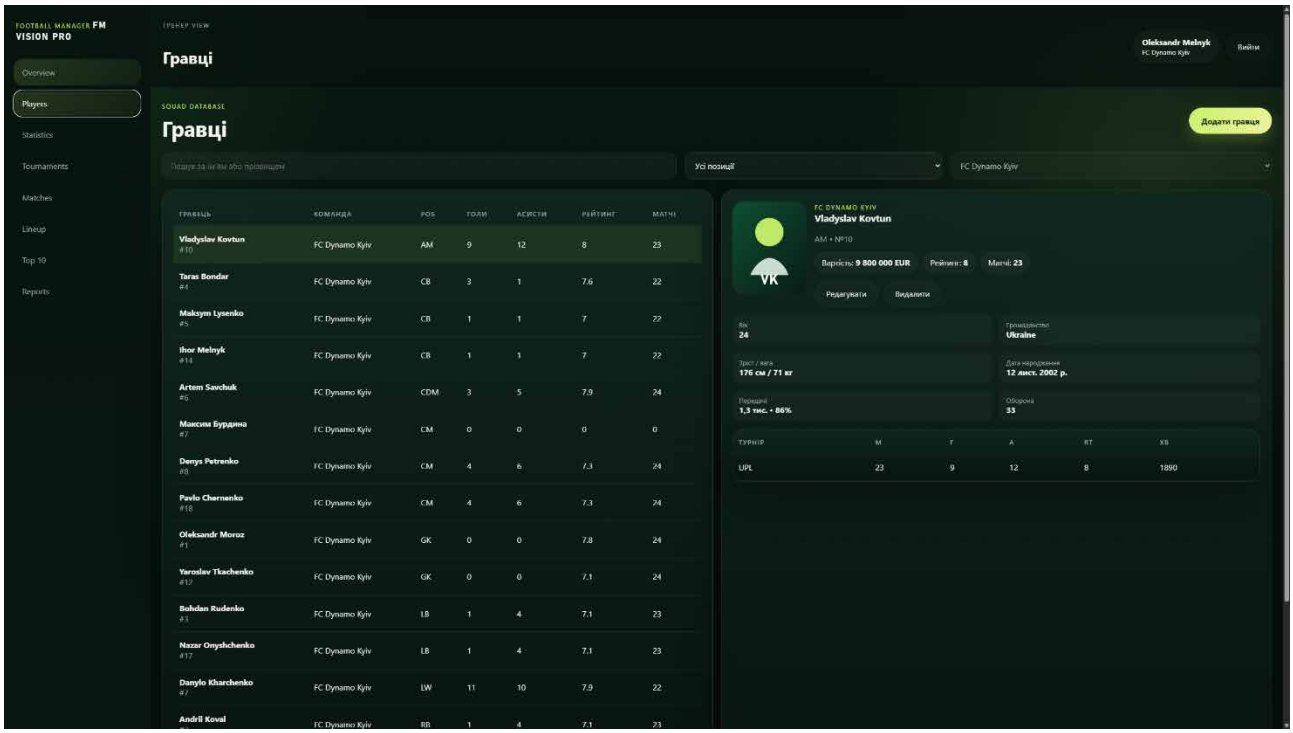

Рис. 4.4 – Перегляд інформації про футбольну команду

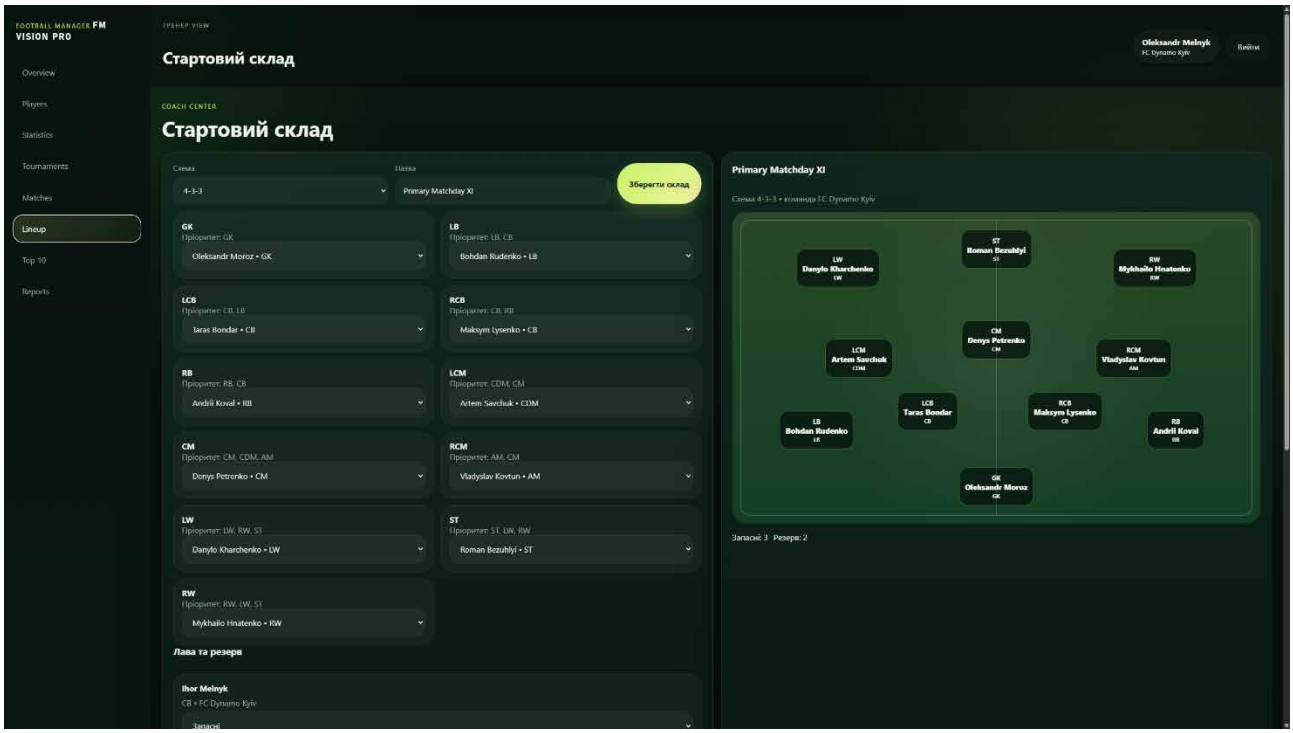

Рис. 4.5 – Формування складу команди тренером

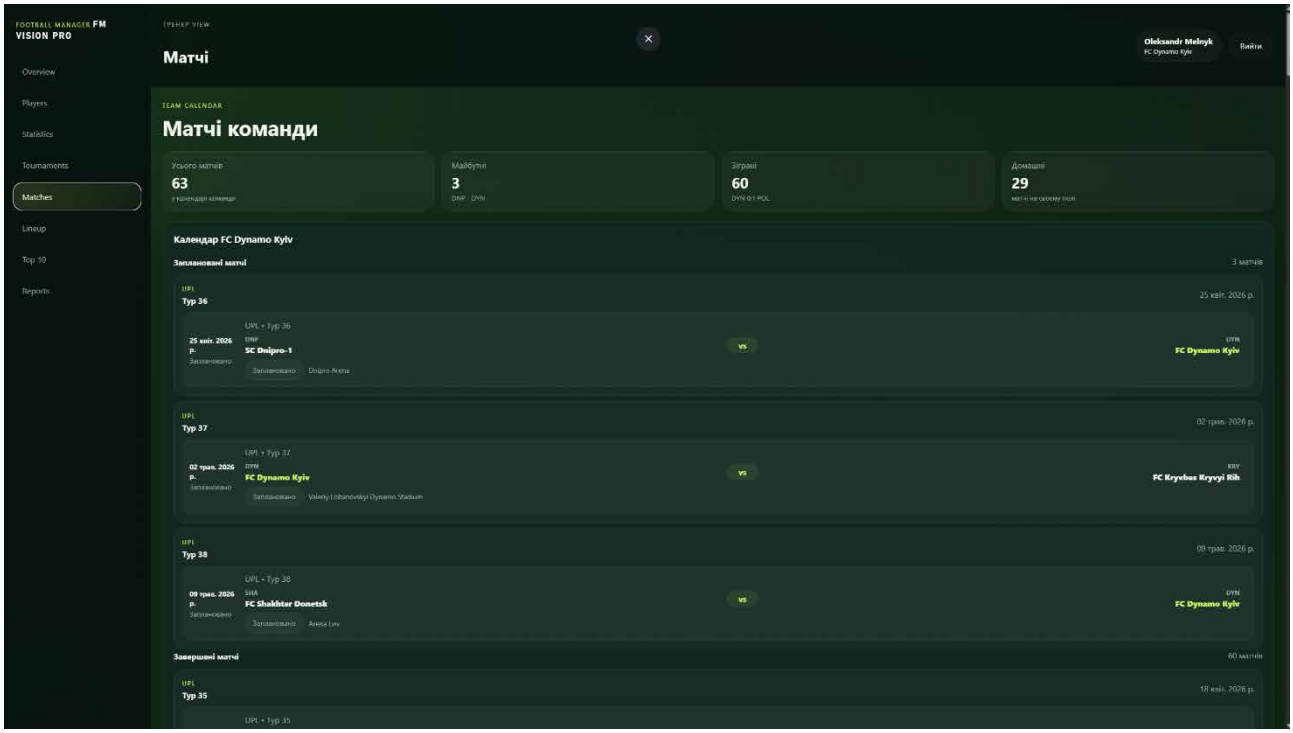

Рис. 4.6 – Перегляд матчів команди

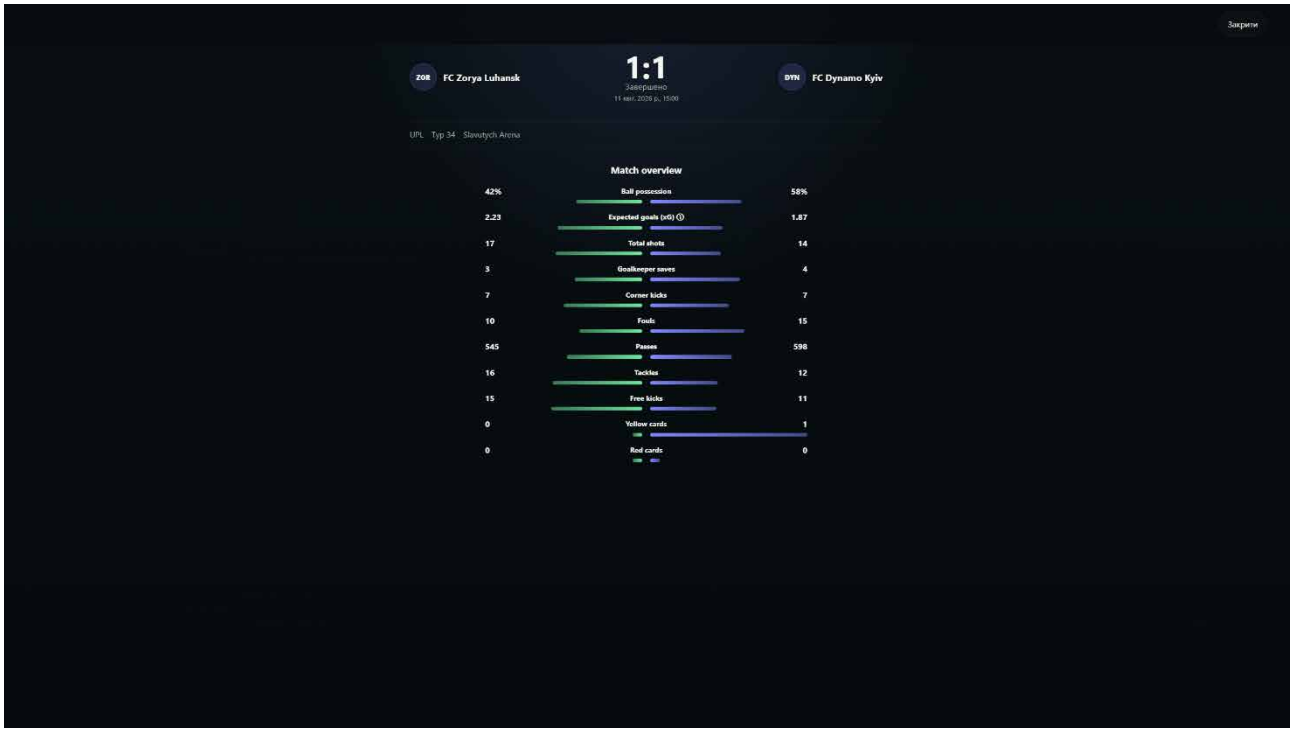

Рис. 4.7 – Перегляд статистики завершеного матчу

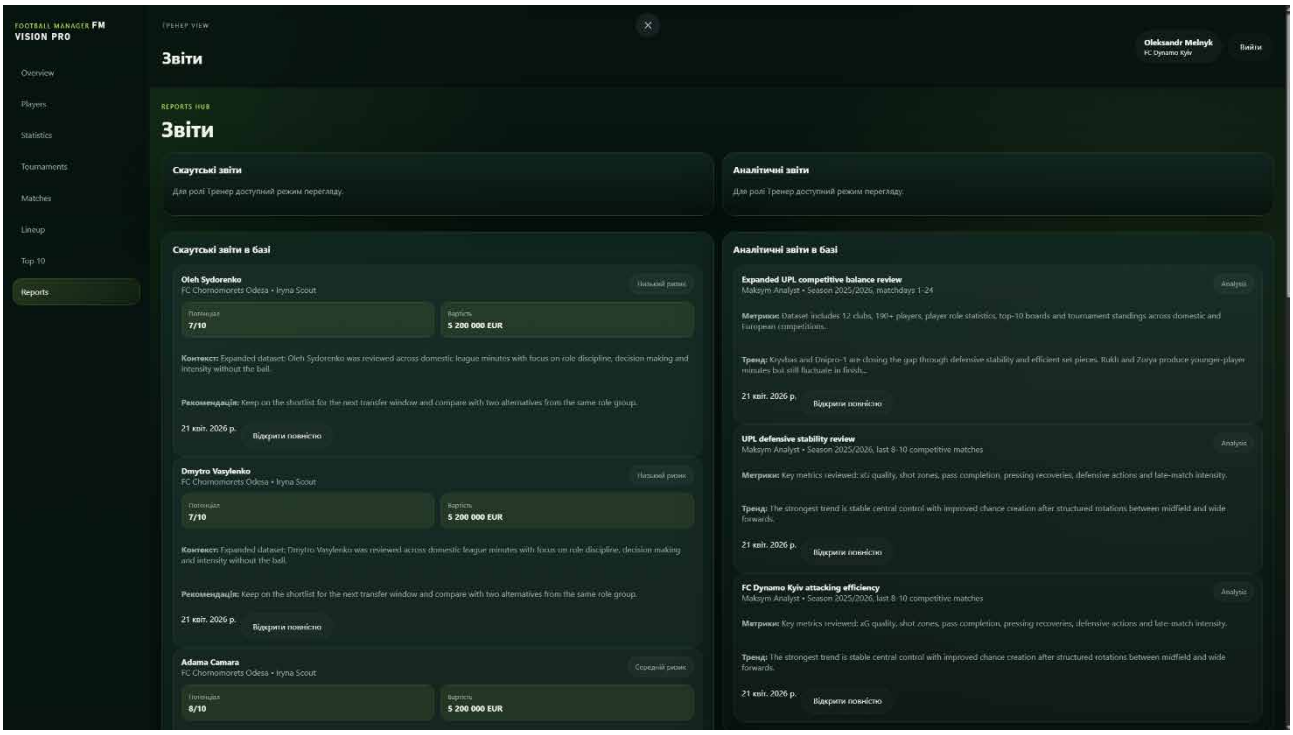

Рис. 4.8 – Робота зі скаутськими та аналітичними звітами

За результатами тестування встановлено, що основні функції системи працюють коректно. Система забезпечує авторизацію користувачів, рольове розмежування доступу, адміністрування акаунтів, перегляд статистичних даних, роботу з матчами, складами та звітами. Проведена перевірка підтвердила працездатність програмного продукту та його відповідність основним функціональним вимогам.

4.2 Вимоги до апаратного та програмного забезпечення

Для впровадження та стабільної експлуатації системи обліку статистичних даних для футбольного менеджера необхідно забезпечити наявність відповідного апаратного та програмного середовища. Оскільки система є веборієнтованою, користувачі працюють із нею через браузер, а основна обробка запитів, взаємодія з базою даних і виконання бізнес-логіки здійснюються на серверній частині.

Серверна частина системи може бути розгорнута на персональному комп’ютері, ноутбуці або окремому сервері. Для локального запуску та

демонстрації роботи системи достатньо комп’ютера середнього рівня продуктивності. Рекомендовано використовувати процесор із частотою не нижче 2.0 ГГц, не менше 4 ГБ оперативної пам’яті та SSD-накопичувач із вільним простором від 1 ГБ для зберігання файлів проєкту, залежностей Node.js і бази даних SQLite. Для роботи через Cloudflare Tunnel також потрібне стабільне підключення до Інтернету.

Клієнтська частина не потребує встановлення окремого програмного забезпечення на пристрої користувача, оскільки доступ до системи здійснюється через веббраузер. Для роботи з інтерфейсом достатньо персонального комп’ютера, ноутбука, планшета або смартфона з доступом до мережі. Основною вимогою є наявність сучасного браузера, який підтримує HTML, CSS і JavaScript.

Таблиця 3

Мінімальні апаратні вимоги до системи

Компонент	**Мінімальні вимоги**
Процесор	2 ядра, частота від 2.0 ГГц
Оперативна пам’ять	від 4 ГБ
Накопичувач	SSD або HDD, від 1 ГБ вільного місця
Мережа	доступ до Інтернету для роботи через Cloudflare Tunnel
Клієнтський пристрій	ПК, ноутбук, планшет або смартфон

Для функціонування серверної частини необхідно встановити Node.js, який забезпечує виконання backend-частини програми. Серверна логіка реалізована з використанням Express, що відповідає за обробку API-запитів. Для зберігання даних використовується SQLite, тому окреме встановлення серверної СУБД не є обов’язковим: база даних зберігається у файлі football_manager.db.

Також для запуску проєкту потрібен пакетний менеджер npm, за допомогою якого встановлюються залежності, описані у файлі package.json.

Таблиця 4

Програмні вимоги до серверної частини

Компонент	**Призначення**
Операційна система	Windows 10/11 або Linux
Node.js	виконання серверної частини
npm	встановлення залежностей проєкту
Express	обробка API-запитів
SQLite	зберігання даних системи
Cloudflare Tunnel	тимчасовий HTTPS-доступ до локального сервера

Для клієнтської частини потрібен сучасний веббраузер. Під час тестування система може використовуватися в Google Chrome, Microsoft Edge, Mozilla Firefox або інших браузерах, які підтримують сучасні вебстандарти. У разі локального запуску користувач відкриває систему за адресою http://localhost:3000, а при використанні Cloudflare Tunnel доступ здійснюється через тимчасове HTTPS-посилання.

Таблиця 5

Програмні вимоги до клієнтської частини

Компонент	**Мінімальні вимоги**
Веббраузер	Google Chrome, Microsoft Edge, Mozilla Firefox або інший сучасний браузер
Підтримка JavaScript	обов’язкова
Підключення до Інтернету	потрібне для доступу через Cloudflare Tunnel

Компонент	Мінімальні вимоги
Додаткове встановлення	не потрібне

Таким чином, розроблена система не потребує складної інфраструктури для запуску та демонстрації. Використання Node.js, Express і SQLite дозволяє розгорнути програмний продукт локально, а Cloudflare Tunnel забезпечує можливість тимчасового віддаленого доступу до системи без окремого хостингу. Це робить систему зручною для навчального використання, тестування та подальшого розвитку.

4.3 Склад інсталяційного пакету

Інсталяційний пакет системи обліку статистичних даних для футбольного менеджера містить набір файлів і компонентів, необхідних для локального запуску, налаштування, тестування та подальшої експлуатації програмного продукту. Оскільки розроблена система є веборієнтованою, кінцевим користувачам не потрібно встановлювати окремий клієнтський застосунок. Доступ до функціоналу здійснюється через веббраузер після запуску серверної частини системи.

До складу інсталяційного пакету входить клієнтська частина, реалізована засобами HTML, CSS і JavaScript. Вона відповідає за відображення інтерфейсу користувача, сторінок авторизації та реєстрації, панелей різних ролей, розділів команд, гравців, матчів, турнірів, складів, статистики та звітів. Клієнтська частина взаємодіє із серверною через API-запити.

Серверна частина системи реалізована на основі Node.js та Express. Вона містить основні програмні модулі, які забезпечують обробку HTTP-запитів, авторизацію користувачів, перевірку ролей, виконання бізнес-логіки та взаємодію з базою даних. До серверної частини належать модулі маршрутів, контролерів, сервісів, middleware, конфігураційні файли та службові утиліти.

Окремим компонентом інсталяційного пакету є база даних SQLite. Дані системи зберігаються у файлі football_manager.db, що спрощує локальне розгортання програмного продукту, оскільки не потребує встановлення окремого сервера баз даних. Структура бази даних описується у файлі schema.sql, а початкове наповнення системи демонстраційними даними виконується за допомогою файлу seeds.js.

Таблиця 6

Склад інсталяційного пакету

Компонент	**Призначення**
frontend	Файли клієнтського інтерфейсу
backend	Серверна частина системи
backend/routes	API-маршрути системи
backend/controllers	Обробка HTTP-запитів
backend/services	Бізнес-логіка програмних модулів
backend/middleware	Перевірка авторизації та ролей
backend/config	Налаштування підключення до бази даних
database/football_manager.db	Файл бази даних SQLite
database/schema.sql	SQL-структура таблиць
database/seeds.js	Початкові демонстраційні дані
package.json	Опис залежностей і команд запуску
.env	Конфігураційні параметри середовища

Для запуску системи необхідно встановити залежності проєкту за допомогою npm, після чого запустити серверну частину. Після запуску вебсистема стає доступною за локальною адресою http://localhost:3000. У разі потреби тимчасового віддаленого доступу може бути використано Cloudflare

Tunnel, який створює захищене HTTPS-посилання та перенаправляє запити до локального сервера [25].

Для відображення фізичного розміщення компонентів програмної системи використано UML-діаграму розміщення. Такий тип діаграми застосовується для подання вузлів виконання системи, програмних артефактів, що розгортаються на цих вузлах, а також зв’язків між ними [32]. У межах розробленої системи діаграма демонструє взаємодію клієнтського пристрою користувача, Cloudflare Tunnel, серверної частини Node.js + Express і бази даних SQLite.

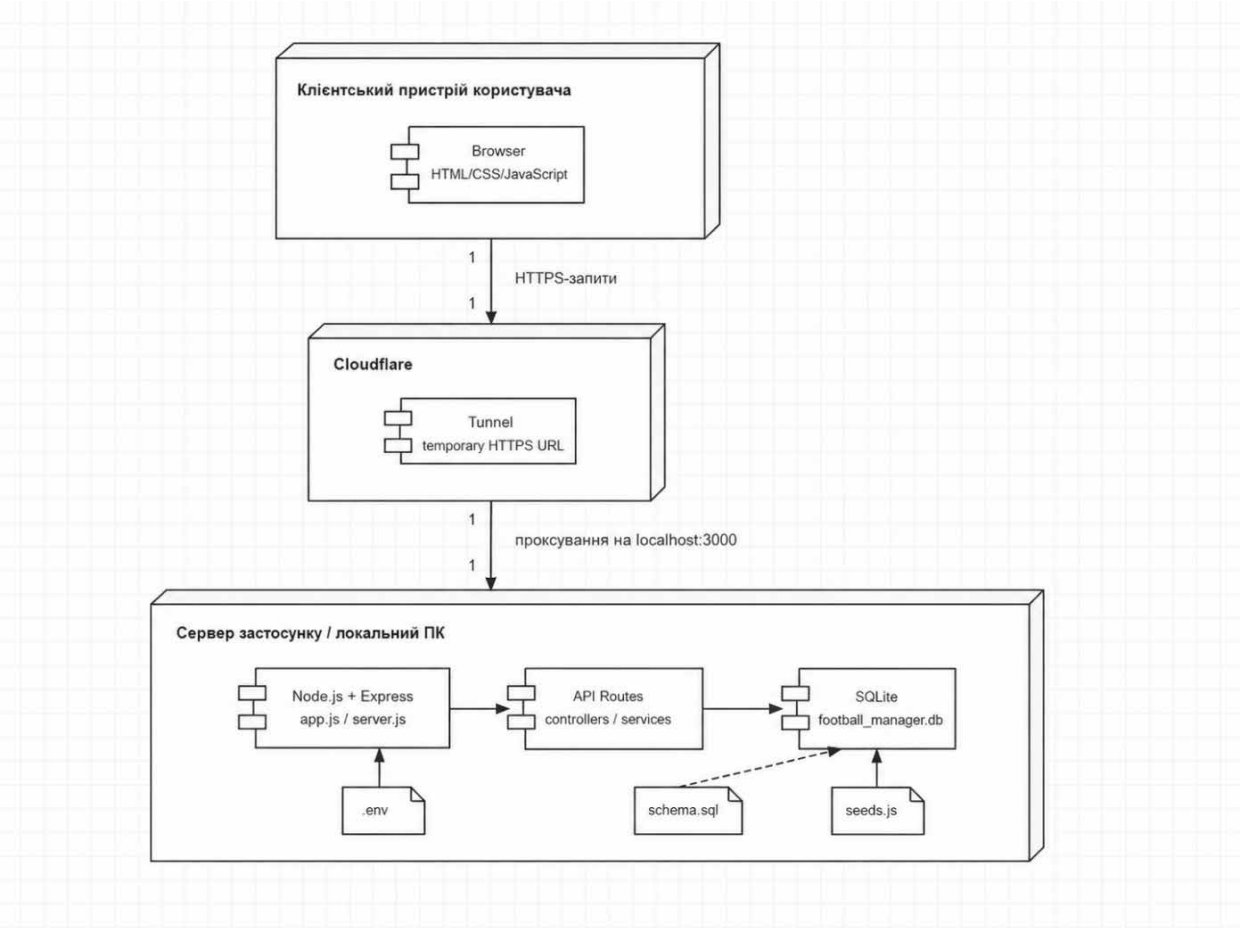

Рис. 4.9 – Діаграма розміщення вебсистеми обліку статистичних даних для футбольного менеджера

Користувачі системи працюють лише з вебінтерфейсом і не мають потреби взаємодіяти з файлами інсталяційного пакету або базою даних напряму. Адміністратор чи розробник використовує інсталяційний пакет для запуску системи, налаштування середовища, підтримки бази даних, перевірки роботи модулів і подальшого розвитку програмного продукту. Такий склад пакету є

достатнім для локальної експлуатації, демонстрації системи та її подальшого вдосконалення.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено веборієнтовану систему обліку статистичних даних для футбольного менеджера. Розроблений програмний продукт призначений для автоматизації роботи з футбольною інформацією, зокрема даними про команди, гравців, матчі, турніри, склади, статистичні показники, скаутські та аналітичні звіти.

У процесі виконання роботи було проведено аналіз предметної області та визначено основні проблеми, пов’язані з ручним або частково автоматизованим обліком футбольної статистики. Було встановлено, що використання спеціалізованої інформаційної системи дозволяє зменшити кількість помилок під час роботи з даними, прискорити доступ до статистичної інформації та забезпечити зручну взаємодію між тренером, аналітиком, скаутом, гравцем і адміністратором.

Під час проєктування системи було визначено основні категорії користувачів і функції, доступні кожній ролі. Адміністратор отримує можливість керувати обліковими записами, підтверджувати реєстрацію, змінювати ролі та видаляти акаунти. Тренер може переглядати інформацію про команду, працювати зі звітами та формувати склад. Аналітик має доступ до аналізу матчів, редагування матчевих даних і створення аналітичних звітів. Скаут формує скаутські звіти, а гравець може переглядати інформацію про команду, особисту статистику та розклад матчів.

Для реалізації системи було використано HTML, CSS і JavaScript для клієнтської частини, Node.js та Express для серверної логіки, а також SQLite для зберігання даних. Такий набір технологій забезпечив простоту локального розгортання, достатню продуктивність для навчального проєкту та зручну організацію взаємодії між інтерфейсом, backend-модулями і базою даних.

У межах роботи було створено логічну модель даних, що охоплює основні сутності системи: користувачів, команди, гравців, турніри, матчі, статистику,

склади та звіти. Також було розроблено програмні модулі для авторизації, реєстрації, перевірки ролей, адміністрування акаунтів, роботи з матчами, формування складів і створення звітів.

Проведене тестування підтвердило працездатність основних функцій системи. Було перевірено авторизацію та реєстрацію користувачів, рольове розмежування доступу, роботу адміністратора з акаунтами, перегляд командної інформації, формування складу, роботу з матчами, статистикою та звітами. Результати тестування показали, що система виконує поставлені функціональні вимоги та може бути використана для демонстрації автоматизованого обліку футбольної статистики.

Отже, мету бакалаврської кваліфікаційної роботи було досягнуто. Розроблена система дозволяє централізовано зберігати та обробляти статистичні дані футбольної команди, забезпечує рольовий доступ до функцій і створює основу для подальшого розвитку програмного продукту. У майбутньому систему можна розширити шляхом додавання детальнішої матчевої аналітики, графіків динаміки показників, експорту звітів у файл, інтеграції з зовнішніми футбольними сервісами та розгортання на повноцінному хостингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Українська асоціація футболу. Офіційний сайт. URL: https://uaf.ua/
2. Українська Прем'єр-Ліга. Офіційний сайт. URL: https://upl.ua/ua
3. FIFA. Football Data. URL: https://inside.fifa.com/innovation/football-data
4. UEFA. Technical Reports. URL: https://www.uefa.com/development/performance-analysis/technical-reports/
5. Cui Y., Pu Z. та ін. Soccer Analytics: A Review. URL: https://www.ieee-jas.net/article/doi/10.1109/JAS.2023.123807?pageType=en
6. Node.js. About Node.js. URL: https://nodejs.org/en/about
7. Express.js. Routing. URL: https://expressjs.com/en/guide/routing.html
8. SQLite. About SQLite. URL: https://www.sqlite.org/about.html
9. Українська Прем'єр-Ліга. Офіційний сайт. URL: https://upl.ua/ua
10. Українська Прем'єр-Ліга. Матчі турнірів. URL: https://www.upl.ua/ua/tournaments/games
11. Українська асоціація футболу. Нормативні документи. URL: https://uaf.ua/en/documents/category/regulations
12. Rein R., Memmert D. Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science. URL: https://link.springer.com/article/10.1186/s40064-016-3108-2
13. Pappalardo L. та ін. PlayeRank: Data-driven Performance Evaluation and Player Ranking in Soccer via a Machine Learning Approach. URL: https://arxiv.org/abs/1802.04987
14. Sofascore. Офіційний сайт. URL: https://www.sofascore.com/
15. Sofascore Rating. URL: https://www.sofascore.com/news/sofascore-rating
16. Transfermarkt. Офіційний сайт. URL: https://www.transfermarkt.com/

17. Transfermarkt. LaLiga market values. URL: https://www.transfermarkt.com/laliga-market-values-valverde-and-pedri-set-records-atletico-suffer-biggest-drop-in-europe/view/news/413278
18. Діаграма прецедентів // Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів
19. Діаграма діяльності // Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_діяльності
20. Реляційна модель даних // Вікіпедія. URL: https://uk.wikipedia.org/wiki/Реляційна_модель_даних
21. Зовнішній ключ // Вікіпедія. URL: https://uk.wikipedia.org/wiki/Зовнішній_ключ
22. Модель «сутність-зв'язок» // Вікіпедія. URL: https://uk.wikipedia.org/wiki/Модель_«сутність-зв'язок»
23. SQLite. CREATE INDEX. URL: https://www.sqlite.org/lang_createindex.html
24. SQLite. Transaction. URL: https://www.sqlite.org/lang_transaction.html
25. Cloudflare Docs. TryCloudflare Quick Tunnels. URL: https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/do-more-with-tunnels/trycloudflare/
26. Package diagram // Wikipedia. URL: https://en.wikipedia.org/wiki/Package_diagram
27. MDN Web Docs. HTML. URL: https://developer.mozilla.org/en-US/docs/Web/HTML
28. MDN Web Docs. CSS. URL: https://developer.mozilla.org/en-US/docs/Web/CSS
29. MDN Web Docs. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript
30. bcrypt. npm package. URL: https://www.npmjs.com/package/bcrypt
31. RFC 7519. JSON Web Token. URL: https://www.rfc-editor.org/rfc/rfc7519

32. Deployment diagram // Wikipedia. URL: https://en.wikipedia.org/wiki/Deployment_diagram

Додаток А

Фрагмент коду 1

```
CREATE TABLE IF NOT EXISTS teams (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  city TEXT NOT NULL,
  short_name TEXT NOT NULL UNIQUE,
  founded_year INTEGER,
  stadium TEXT,
  coach_name TEXT,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  first_name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  email TEXT NOT NULL UNIQUE,
  password_hash TEXT NOT NULL,
   role TEXT NOT NULL CHECK (role IN ('admin', 'pending', 'coach', 'scout', 'analyst', 'player')),
  team_id INTEGER,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS players (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  first_name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  age INTEGER NOT NULL,
  position TEXT NOT NULL,
```

```
  number INTEGER NOT NULL,
  team_id INTEGER NOT NULL,
  nationality TEXT NOT NULL,
  height_cm INTEGER NOT NULL,
  weight_kg INTEGER NOT NULL,
  birth_date TEXT NOT NULL,
  market_value INTEGER NOT NULL DEFAULT 0,
  photo_url TEXT,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS tournaments (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  type TEXT NOT NULL,
  season TEXT NOT NULL,
  country TEXT NOT NULL,
  status TEXT NOT NULL DEFAULT 'active',
  created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS player_statistics (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  player_id INTEGER NOT NULL,
  tournament_id INTEGER NOT NULL,
  season TEXT NOT NULL,
  matches INTEGER NOT NULL DEFAULT 0,
  goals INTEGER NOT NULL DEFAULT 0,
  assists INTEGER NOT NULL DEFAULT 0,
  rating REAL NOT NULL DEFAULT 0,
  minutes INTEGER NOT NULL DEFAULT 0,
  yellow_cards INTEGER NOT NULL DEFAULT 0,
  red_cards INTEGER NOT NULL DEFAULT 0,
  passes INTEGER NOT NULL DEFAULT 0,
```

```
  pass_accuracy REAL NOT NULL DEFAULT 0,
  tackles INTEGER NOT NULL DEFAULT 0,
  interceptions INTEGER NOT NULL DEFAULT 0,
  saves INTEGER NOT NULL DEFAULT 0,
  clean_sheets INTEGER NOT NULL DEFAULT 0,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (player_id) REFERENCES players(id) ON DELETE CASCADE,
  FOREIGN KEY (tournament_id) REFERENCES tournaments(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS standings (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tournament_id INTEGER NOT NULL,
  team_id INTEGER NOT NULL,
  place INTEGER NOT NULL,
  played INTEGER NOT NULL DEFAULT 0,
  wins INTEGER NOT NULL DEFAULT 0,
  draws INTEGER NOT NULL DEFAULT 0,
  losses INTEGER NOT NULL DEFAULT 0,
  goals_for INTEGER NOT NULL DEFAULT 0,
  goals_against INTEGER NOT NULL DEFAULT 0,
  goal_difference INTEGER NOT NULL DEFAULT 0,
  points INTEGER NOT NULL DEFAULT 0,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (tournament_id) REFERENCES tournaments(id) ON DELETE
CASCADE,
  FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS matches (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tournament_id INTEGER NOT NULL,
  home_team_id INTEGER NOT NULL,
  away_team_id INTEGER NOT NULL,
  matchday INTEGER NOT NULL DEFAULT 1,
```

```
        match_date TEXT NOT NULL,
        venue TEXT,
        status TEXT NOT NULL DEFAULT 'scheduled' CHECK (status IN ('scheduled', 'live', 'finished', 'postponed')),
        home_score INTEGER,
        away_score INTEGER,
        notes TEXT DEFAULT '',
        created_at TEXT DEFAULT CURRENT_TIMESTAMP,
        updated_at TEXT DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (tournament_id) REFERENCES tournaments(id) ON DELETE CASCADE,
        FOREIGN KEY (home_team_id) REFERENCES teams(id) ON DELETE CASCADE,
        FOREIGN KEY (away_team_id) REFERENCES teams(id) ON DELETE CASCADE
      );

      CREATE TABLE IF NOT EXISTS match_statistics (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        match_id INTEGER NOT NULL UNIQUE,
        home_possession INTEGER NOT NULL DEFAULT 50,
        away_possession INTEGER NOT NULL DEFAULT 50,
        home_xg REAL NOT NULL DEFAULT 0,
        away_xg REAL NOT NULL DEFAULT 0,
        home_shots INTEGER NOT NULL DEFAULT 0,
        away_shots INTEGER NOT NULL DEFAULT 0,
        home_saves INTEGER NOT NULL DEFAULT 0,
        away_saves INTEGER NOT NULL DEFAULT 0,
        home_corners INTEGER NOT NULL DEFAULT 0,
        away_corners INTEGER NOT NULL DEFAULT 0,
        home_fouls INTEGER NOT NULL DEFAULT 0,
        away_fouls INTEGER NOT NULL DEFAULT 0,
        home_passes INTEGER NOT NULL DEFAULT 0,
        away_passes INTEGER NOT NULL DEFAULT 0,
        home_tackles INTEGER NOT NULL DEFAULT 0,
        away_tackles INTEGER NOT NULL DEFAULT 0,
        home_free_kicks INTEGER NOT NULL DEFAULT 0,
```

```
  away_free_kicks INTEGER NOT NULL DEFAULT 0,
  home_yellow_cards INTEGER NOT NULL DEFAULT 0,
  away_yellow_cards INTEGER NOT NULL DEFAULT 0,
  home_red_cards INTEGER NOT NULL DEFAULT 0,
  away_red_cards INTEGER NOT NULL DEFAULT 0,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  updated_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (match_id) REFERENCES matches(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS lineups (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  team_id INTEGER NOT NULL,
  title TEXT NOT NULL,
  formation TEXT NOT NULL,
  created_by INTEGER NOT NULL,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE CASCADE,
  FOREIGN KEY (created_by) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS lineup_players (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  lineup_id INTEGER NOT NULL,
  player_id INTEGER NOT NULL,
  squad_role TEXT NOT NULL CHECK (squad_role IN ('starter', 'bench', 'reserve')),
  slot_key TEXT NOT NULL,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (lineup_id) REFERENCES lineups(id) ON DELETE CASCADE,
  FOREIGN KEY (player_id) REFERENCES players(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS scout_reports (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  player_id INTEGER NOT NULL,
```

```
  scout_user_id INTEGER NOT NULL,
  match_context TEXT DEFAULT '',
  strengths TEXT NOT NULL,
  weaknesses TEXT NOT NULL,
  tactical_fit TEXT DEFAULT '',
  risk_level TEXT DEFAULT 'medium',
  potential_rating INTEGER DEFAULT 0,
  estimated_value INTEGER DEFAULT 0,
  recommendations TEXT NOT NULL,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (player_id) REFERENCES players(id) ON DELETE CASCADE,
  FOREIGN KEY (scout_user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS analyst_reports (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  analyst_user_id INTEGER NOT NULL,
  subject TEXT NOT NULL,
  period TEXT DEFAULT '',
  metrics_snapshot TEXT DEFAULT '',
  analysis TEXT NOT NULL,
  trend_summary TEXT DEFAULT '',
  risk_factors TEXT DEFAULT '',
  action_plan TEXT DEFAULT '',
  conclusions TEXT NOT NULL,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (analyst_user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE INDEX IF NOT EXISTS idx_players_team_id ON players(team_id);
CREATE INDEX IF NOT EXISTS idx_players_position ON players(position);
CREATE INDEX IF NOT EXISTS idx_users_role ON users(role);
CREATE INDEX IF NOT EXISTS idx_statistics_player_id ON player_statistics(player_id);
CREATE INDEX IF NOT EXISTS idx_statistics_tournament_id ON
player_statistics(tournament_id);
```

```
CREATE INDEX IF NOT EXISTS idx_standings_tournament_id ON standings(tournament_id);
CREATE INDEX IF NOT EXISTS idx_matches_tournament_id ON matches(tournament_id);
CREATE INDEX IF NOT EXISTS idx_matches_home_team_id ON matches(home_team_id);
CREATE INDEX IF NOT EXISTS idx_matches_away_team_id ON matches(away_team_id);
CREATE INDEX IF NOT EXISTS idx_matches_match_date ON matches(match_date);
CREATE INDEX IF NOT EXISTS idx_match_statistics_match_id ON match_statistics(match_id);
CREATE INDEX IF NOT EXISTS idx_lineups_team_id ON lineups(team_id)
```

Додаток Б

Фрагмент коду 2

```
import {
  apiFetch,
  guardPublicAuthPages,
  saveSession,
  setFormMessage,
  showToast
} from "./common.js";

guardPublicAuthPages();

const pageType = document.body.dataset.page;
const loginForm = document.getElementById("loginForm");
const registerForm = document.getElementById("registerForm");
const loginMessage = document.getElementById("loginMessage");
const registerMessage = document.getElementById("registerMessage");

if (pageType === "register") {
  populateTeams();
}

if (loginForm) {
  loginForm.addEventListener("submit", handleLoginSubmit);
}

if (registerForm) {
  registerForm.addEventListener("submit", handleRegisterSubmit);
}

async function populateTeams() {
  const teamSelect = document.getElementById("teamSelect");
```

```
  try {
    const data = await apiFetch("/api/teams", { auth: false });

    for (const team of data.teams) {
      const option = document.createElement("option");
      option.value = team.id;
      option.textContent = `${team.name} (${team.short_name})`;
      teamSelect.appendChild(option);
    }
  } catch (error) {
    setFormMessage(registerMessage, error.message);
  }
}

async function handleLoginSubmit(event) {
  event.preventDefault();
  setFormMessage(loginMessage, "");

  const formData = new FormData(loginForm);
  const payload = Object.fromEntries(formData.entries());

  try {
    const data = await apiFetch("/api/auth/login", {
      method: "POST",
      auth: false,
      body: payload
    });

    saveSession(data.token, data.user);
    showToast("Login successful.", "success");
    window.location.href = "/app";
  } catch (error) {
    setFormMessage(loginMessage, error.message);
  }
```

```
}

async function handleRegisterSubmit(event) {
  event.preventDefault();
  setFormMessage(registerMessage, "");

  const formData = new FormData(registerForm);
  const payload = Object.fromEntries(formData.entries());

  try {
    const data = await apiFetch("/api/auth/register", {
      method: "POST",
      auth: false,
      body: payload
    });

    registerForm.reset();
    setFormMessage(registerMessage, data.message || "Registration submitted for approval.",
"success");
    showToast("Registration submitted for approval.", "success");
    setTimeout(() => {
      window.location.href = "/login";
    }, 1800);
  } catch (error) {
    setFormMessage(registerMessage, error.message);
  }
}
```

Додаток В

Фрагмент коду 3

```
const TOKEN_KEY = "fm_token";
const USER_KEY = "fm_user";

export const roleLabels = {
  admin: "Administrator",
  pending: "Newly registered",
  coach: "Тренер",
  scout: "Скаут",
  analyst: "Аналітик",
  player: "Гравець"
};

export function getToken() {
  return localStorage.getItem(TOKEN_KEY);
}

export function loadSession() {
  const token = localStorage.getItem(TOKEN_KEY);
  const userRaw = localStorage.getItem(USER_KEY);

  return {
    token,
    user: userRaw ? JSON.parse(userRaw) : null
  };
}

export function saveSession(token, user) {
  localStorage.setItem(TOKEN_KEY, token);
  localStorage.setItem(USER_KEY, JSON.stringify(user));
}
```

```
export function clearSession() {
  localStorage.removeItem(TOKEN_KEY);
  localStorage.removeItem(USER_KEY);
}

export function redirectToLogin() {
  window.location.href = "/login";
}

export function redirectToApp() {
  window.location.href = "/app";
}

export function guardPublicAuthPages() {
  const session = loadSession();

  if (session.token) {
    redirectToApp();
  }
}

export function requireAuthPage() {
  if (!getToken()) {
    redirectToLogin();
  }
}

export async function apiFetch(path, options = {}) {
  const { method = "GET", body, auth = true, headers = {} } = options;
  const requestHeaders = { ...headers };

  if (body !== undefined) {
    requestHeaders["Content-Type"] = "application/json";
  }
```

```
  if (auth && getToken()) {
    requestHeaders.Authorization = `Bearer ${getToken()}`;
  }

  const response = await fetch(path, {
    method,
    headers: requestHeaders,
    body: body !== undefined ? JSON.stringify(body) : undefined
  });

  const isJson = response.headers.get("content-type")?.includes("application/json");
  const data = isJson ? await response.json() : null;

  if (!response.ok) {
    if (response.status === 401 && auth) {
      clearSession();
      redirectToLogin();
    }

    throw new Error(data?.message || "Request failed.");
  }

  return data;
}

export function setFormMessage(element, message = "", type = "error") {
  if (!element) {
    return;
  }

  element.textContent = message;
  element.dataset.type = message ? type : "";
}
```

```
export function showToast(message, type = "info") {
  const container = document.getElementById("toastContainer");

  if (!container) {
    return;
  }

  const toast = document.createElement("div");
  toast.className = `toast toast-${type}`;
  toast.textContent = message;
  container.appendChild(toast);

  requestAnimationFrame(() => {
    toast.classList.add("visible");
  });

  setTimeout(() => {
    toast.classList.remove("visible");
    setTimeout(() => toast.remove(), 300);
  }, 2800);
}

export function formatCurrency(value) {
  return new Intl.NumberFormat("uk-UA", {
    style: "currency",
    currency: "EUR",
    maximumFractionDigits: 0
  }).format(Number(value || 0));
}

export function formatCompactNumber(value) {
  return new Intl.NumberFormat("uk-UA", {
    notation: "compact",
    maximumFractionDigits: 1
  }).format(Number(value || 0));
```

```
}

export function formatDate(value) {
 if (!value) {
  return "-";
 }

 return new Intl.DateTimeFormat("uk-UA", {
  day: "2-digit",
  month: "short",
  year: "numeric"
 }).format(new Date(value));
}

export function buildAvatarDataUri(name) {
 const safeName = String(name || "Player");
 const initials = safeName
  .split(" ")
  .filter(Boolean)
  .slice(0, 2)
  .map((part) => part[0].toUpperCase())
  .join("");

 const svg = `
  <svg xmlns="http://www.w3.org/2000/svg" width="320" height="380" viewBox="0 0 320
380">
   <defs>
    <linearGradient id="bg" x1="0%" y1="0%" x2="100%" y2="100%">
     <stop offset="0%" stop-color="#0f5132" />
     <stop offset="100%" stop-color="#091c17" />
    </linearGradient>
   </defs>
   <rect width="320" height="380" rx="28" fill="url(#bg)" />
   <circle cx="160" cy="126" r="58" fill="#c9f36a" opacity="0.95" />
```

```
            <path d="M82 302c18-47 58-71 78-71 23 0 63 23 78 71" fill="#d7e8db" opacity="0.94" />
                <text x="160" y="338" text-anchor="middle" fill="#ffffff" font-size="58" font-family="Bahnschrift, Segoe UI, sans-serif" font-weight="700">${initials}</text>
            </svg>
          `;

          return `data:image/svg+xml;charset=UTF-8,${encodeURIComponent(svg)}`;
        }

        export function escapeHtml(value) {
          return String(value ?? "")
            .replaceAll("&", "&amp;")
            .replaceAll("<", "&lt;")
            .replaceAll(">", "&gt;")
            .replaceAll("", "&quot;")
            .replaceAll("", "&#39;");
        }
```

Додаток Д

Фрагмент коду 4

```
const fs = require("fs/promises");
const path = require("path");
const sqlite3 = require("sqlite3");
const { open } = require("sqlite");
const seedDatabase = require("../../database/seeds");

const DB_PATH = path.join(__dirname, "..", "..", "database", "football_manager.db");
const SCHEMA_PATH = path.join(__dirname, "..", "..", "database", "schema.sql");

let dbInstance;

async function ensureColumn(db, tableName, columnName, definition) {
  const columns = await db.all(`PRAGMA table_info(${tableName})`);
  const hasColumn = columns.some((column) => column.name === columnName);

  if (!hasColumn) {
    await db.exec(`ALTER TABLE ${tableName} ADD COLUMN ${columnName} ${definition}`);
  }
}

async function ensureUsersRoleSupport(db) {
  const usersTable = await db.get(
    "SELECT sql FROM sqlite_master WHERE type = 'table' AND name = 'users'"
  );
  const usersSql = String(usersTable?.sql || "");

  if (usersSql.includes("'admin'") && usersSql.includes("'pending'")) {
```

```
    return;
  }

  await db.exec("PRAGMA foreign_keys = OFF;");

  try {
    await db.exec("BEGIN TRANSACTION;");
    await db.exec("ALTER TABLE users RENAME TO users_legacy;");
    await db.exec(`
      CREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        first_name TEXT NOT NULL,
        last_name TEXT NOT NULL,
        email TEXT NOT NULL UNIQUE,
        password_hash TEXT NOT NULL,
         role TEXT NOT NULL CHECK (role IN ('admin', 'pending', 'coach', 'scout', 'analyst',
'player')),
        team_id INTEGER,
        created_at TEXT DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE SET NULL
      );
    `);
    await db.exec(`
      INSERT INTO users (id, first_name, last_name, email, password_hash, role, team_id,
created_at)
       SELECT id, first_name, last_name, email, password_hash, role, team_id, created_at
       FROM users_legacy;
    `);
    await db.exec("DROP TABLE users_legacy;");
    await db.exec("CREATE INDEX IF NOT EXISTS idx_users_role ON users(role);");
    await db.exec("COMMIT;");
  } catch (error) {
    await db.exec("ROLLBACK;");
    throw error;
  } finally {
```

```
    await db.exec("PRAGMA foreign_keys = ON;");
  }
}

async function runMigrations(db) {
  await ensureUsersRoleSupport(db);

  await ensureColumn(db, "scout_reports", "match_context", "TEXT DEFAULT ''");
  await ensureColumn(db, "scout_reports", "tactical_fit", "TEXT DEFAULT ''");
  await ensureColumn(db, "scout_reports", "risk_level", "TEXT DEFAULT 'medium'");
  await ensureColumn(db, "scout_reports", "potential_rating", "INTEGER DEFAULT 0");
  await ensureColumn(db, "scout_reports", "estimated_value", "INTEGER DEFAULT 0");

  await ensureColumn(db, "analyst_reports", "period", "TEXT DEFAULT ''");
  await ensureColumn(db, "analyst_reports", "metrics_snapshot", "TEXT DEFAULT ''");
  await ensureColumn(db, "analyst_reports", "trend_summary", "TEXT DEFAULT ''");
  await ensureColumn(db, "analyst_reports", "risk_factors", "TEXT DEFAULT ''");
  await ensureColumn(db, "analyst_reports", "action_plan", "TEXT DEFAULT ''");

  await db.run(
    `UPDATE scout_reports
     SET
       match_context = COALESCE(NULLIF(match_context, ''), 'Observed in competitive league and European fixtures with focus on off-ball habits, pressing reactions and decision making under pressure.'),
       tactical_fit = COALESCE(NULLIF(tactical_fit, ''), 'Best suited for a proactive team that attacks through quick wide combinations and needs flexible forwards able to rotate between channels.'),
       risk_level = COALESCE(NULLIF(risk_level, ''), 'medium'),
       potential_rating = CASE WHEN COALESCE(potential_rating, 0) = 0 THEN 8 ELSE potential_rating END,
       estimated_value = CASE WHEN COALESCE(estimated_value, 0) = 0 THEN 9000000 ELSE estimated_value END
     WHERE id IN (SELECT id FROM scout_reports)`
  );
```

```
    await db.run(
      `UPDATE analyst_reports
       SET
         period = COALESCE(NULLIF(period, ''), 'Season 2025/2026, last 8-10 competitive matches'),
         metrics_snapshot = COALESCE(NULLIF(metrics_snapshot, ''), 'Key metrics reviewed: xG quality, shot zones, pass completion, pressing recoveries, defensive actions and late-match intensity.'),
         trend_summary = COALESCE(NULLIF(trend_summary, ''), 'The strongest trend is stable central control with improved chance creation after structured rotations between midfield and wide forwards.'),
         risk_factors = COALESCE(NULLIF(risk_factors, ''), 'Main risks are transition exposure after full-back overlaps and reduced pressing intensity after substitutions.'),
         action_plan = COALESCE(NULLIF(action_plan, ''), 'Prepare opponent-specific pressing triggers, rotate wide players earlier and keep one holding midfielder behind attacking phases.')
       WHERE id IN (SELECT id FROM analyst_reports)`
    );
  }

  async function initDatabase() {
    if (dbInstance) {
      return dbInstance;
    }

    await fs.mkdir(path.dirname(DB_PATH), { recursive: true });

    dbInstance = await open({
      filename: DB_PATH,
      driver: sqlite3.Database
    });

    await dbInstance.exec("PRAGMA foreign_keys = ON;");

    const schema = await fs.readFile(SCHEMA_PATH, "utf8");
```

```
  await dbInstance.exec(schema);
  await runMigrations(dbInstance);
  await seedDatabase(dbInstance);

  return dbInstance;
}

function getDb() {
  if (!dbInstance) {
    throw new Error("Database is not initialized. Call initDatabase() first.");
  }

  return dbInstance;
}

module.exports = {
  DB_PATH,
  getDb,
  initDatabase
};
```

Додаток Е

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій
Декларація про академічну доброчесність та використання ШІ

Я, Бурдина Максим Віталійович , здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему

« Система обліку статистичних даних для футбольного менеджера» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [✓] інструменти ШІ ChatGPT були використані для:
  - [✓] перекладу іншомовних джерел;
  - [✓] стилістичного редагування та перевірки граматики;
  - [✓] допомоги у написанні/відлагодженні програмного коду;
  - [] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« » 2026 р. _____ **Максим БУРДИНА**