

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук, фізики та математики
Кафедра комп'ютерних наук та програмної інженерії

Розробка платформи для футбольних ліг та федерацій управління
змаганнями і подіями в місті

Кваліфікаційна робота (проект)
на здобуття ступеня вищої освіти «бакалавр

Виконав: здобувач 4 курсу 12-461 групи

Спеціальність: 126 Інформаційні
системи та технології

Освітньо-професійна програма:

Інформаційні системи та технології

Михайлов Олег Олегович

Керівниця: старша викладачка

Черненко Ірина Євгенівна

Рецензент: директор Херсонського
фізико-технічного ліцею Херсонської
міської ради, к.п.н. Шовкун В.В

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

УАФ – Українська асоціація футболу – організація, що займається управлінням і розвитком футболу в Україні, згадана як джерело звітів і приклад локальних систем.

ММС України – Міністерство молоді та спорту України – державний орган, відповідальний за розвиток фізичної культури та спорту, використаний як джерело звітів і програм.

SMS – Short Message Service (Служба коротких повідомлень) – технологія надсилання текстових повідомлень, згадана в контексті оперативного інформування користувачів.

НУФВСУ – Національний університет фізичного виховання і спорту України – заклад, що проводить конференції та дослідження у сфері спортивного менеджменту, згаданий як джерело матеріалів.

ASP.NET MVC – Фреймворк для створення веб-додатків із розділенням на моделі, контролери та представлення.

WCAG – Web Content Accessibility Guidelines – рекомендації щодо доступності веб-контенту.

ARIA – Accessible Rich Internet Applications – набір атрибутів для підвищення доступності веб-елементів.

HTML5 – П'ята версія мови розмітки гіпертексту для створення структури веб-сторінок.

CSS – Cascading Style Sheets – мова стилів для оформлення веб-інтерфейсу.

JavaScript – Мова програмування для створення інтерактивних елементів на веб-сторінках.

Bootstrap – Фреймворк для створення адаптивного та стилізованого інтерфейсу.

Entity Framework – Технологія об'єктно-реляційного відображення для роботи з базами даних у .NET.

Code First – Підхід у Entity Framework для створення бази даних на основі класів моделей.

DbSet – Колекція об'єктів моделі в Entity Framework, що відповідає таблиці бази даних.

LINQ – Language Integrated Query – мова запитів для роботи з даними в .NET.

CRUD – Create, Read, Update, Delete – базові операції для роботи з даними.

ClaimsIdentity – Клас у ASP.NET для створення ідентичності користувача під час аутентифікації.

IIS Express – Локальний веб-сервер у Visual Studio для тестування ASP.NET додатків.

SQL Server – Система управління базами даних, використана для зберігання даних платформи.

PK – Primary Key – первинний ключ у таблиці бази даних.

FK – Foreign Key – зовнішній ключ у таблиці бази даних для зв'язку з іншою таблицею.

API – Application Programming Interface – інтерфейс для взаємодії між системами.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....	2
ЗМІСТ	4
ЗМІСТ	ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.
ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	8
1.1. Дослідження потреб футбольних ліг та федерацій у системах управління.....	8
1.2. Огляд існуючих платформ і інструментів для організації змагань	11
1.3. Визначення особливостей управління футбольними подіями в міському контексті	15
1.4. Аналіз ролей користувачів (організатори, гравці, вболівальники) та їхніх вимог	17
1.5. Формулювання проблематики та обґрунтування необхідності нової платформи.....	20
РОЗДІЛ 2. ПРОЄКТНА ЧАСТИНА	23
2.1. Розробка загальної структури платформи.....	23
2.2. Проєктування бази даних для зберігання інформації про змагання.....	25
2.3. Опис основних модулів системи (реєстрація, розклад, статистика)	31
2.4. Створення ескізів інтерфейсу для різних категорій користувачів.....	35
2.5. Визначення підходів до забезпечення доступності та зручності платформи.....	43
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ	47
3.1. Розробка прототипу платформи	47
3.2. Реалізація функцій управління командами та подіями	50
3.3. Впровадження системи відображення результатів змагань.....	54
3.4. Проведення тестування на основі тестових сценаріїв	58
3.5. Аналіз результатів і пропозиції щодо подальшого розвитку	63

ВИСНОВКИ	68
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	71
ДОДАТКИ	74

ВСТУП

Сучасний світ спорту, зокрема футболу, перебуває в умовах стрімкого розвитку інформаційних технологій, які змінюють підходи до організації та управління змаганнями. Футбол, як один із найпопулярніших видів спорту в Україні та світі, потребує ефективних інструментів для координації діяльності ліг, федерацій, команд і вболівальників. У міському контексті ці потреби стають ще більш актуальними через високу концентрацію спортивних подій, різноманітність учасників і необхідність швидкого обміну інформацією. Існуючі системи управління змаганнями часто не відповідають сучасним вимогам, мають обмежений функціонал або не враховують специфіку локальних заходів. Саме тому розробка спеціалізованої платформи для футбольних ліг та федерацій є важливим завданням, яке має як теоретичне, так і практичне значення.

Актуальність теми дослідження зумовлена кількома факторами. По-перше, зростання популярності футболу в містах України супроводжується збільшенням кількості аматорських і професійних турнірів, що потребують чіткої організації. По-друге, цифрова трансформація спортивного менеджменту відкриває нові можливості для автоматизації процесів, таких як планування розкладу, ведення статистики, реєстрація учасників і комунікація між усіма сторонами. По-третє, відсутність універсальних вітчизняних платформ, адаптованих до потреб місцевих футбольних спільнот, створює прогалину, яку необхідно заповнити. Розробка такої платформи дозволить не лише підвищити ефективність управління змаганнями, але й сприятиме популяризації футболу на локальному рівні.

Мета роботи полягає в розробці платформи для футбольних ліг та федерацій, яка забезпечить комплексне управління змаганнями і подіями в міському середовищі.

Для досягнення цієї мети визначено такі **основні завдання**:

1. Провести аналіз предметної області, зокрема потреб футбольних організацій та існуючих рішень.
2. Розробити структуру платформи з урахуванням ключових функціональних вимог.
3. Спроекувати базу даних і модулі системи для забезпечення її ефективної роботи.
4. Реалізувати прототип платформи та провести його тестування на основі модельних даних.
5. Оцінити практичну застосовність розробленого рішення та запропонувати шляхи його вдосконалення.

Об'єктом дослідження є процеси управління змаганнями і подіями в діяльності футбольних ліг та федерацій у міському контексті.

Предметом дослідження виступає платформа як інструмент автоматизації цих процесів, що включає функціонал для організації турнірів, управління командами, відображення результатів і забезпечення взаємодії між учасниками.

Структура роботи складається зі вступу, трьох основних розділів, висновків і списку використаних джерел. У першому розділі проведено аналіз предметної області, у другому – описано проєктування платформи, у третьому – представлено практичну реалізацію та її тестування. Такий підхід дозволяє послідовно розкрити тему та досягти поставленої мети.

Дана робота спрямована на вирішення актуальної проблеми – створення ефективного інструменту для управління футбольними змаганнями в місті.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження потреб футбольних ліг та федерацій у системах управління

Футбольні ліги та федерації в Україні, як і в багатьох інших країнах, перебувають у процесі трансформації, викликаній зростанням популярності спорту та необхідністю вдосконалення організаційних процесів. У міських умовах ці організації стикаються з унікальними викликами, такими як координація великої кількості команд, забезпечення доступності спортивних майданчиків і залучення широкої аудиторії. У цьому контексті системи управління стають невід’ємною частиною діяльності, адже вони дозволяють оптимізувати ресурси та підвищувати якість проведення змагань. Однією з ключових потреб є автоматизація повсякденних завдань, таких як складання розкладів, облік учасників і фіксація результатів. Дослідження літератури показують, що значна частина місцевих ліг досі використовує традиційні методи – паперові журнали чи електронні таблиці, що призводить до помилок і затримок у роботі [3]. Такі підходи особливо неефективні в містах, де щотижня може відбуватися кілька турнірів із десятками команд.

Наступною суттєвою потребою є забезпечення прозорості та доступності інформації для всіх зацікавлених сторін. Учасники змагань – від гравців і тренерів до вболівальників – потребують оперативного доступу до розкладів, результатів і статистики. Звіти спортивних організацій свідчать, що відсутність єдиної платформи для обміну даними створює бар’єри в комунікації, знижуючи інтерес до подій і ускладнюючи їх організацію [5]. Наприклад, у багатьох міських лігах інформація про перенесення матчів чи зміни в складі команд

передається усно або через соціальні мережі, що не гарантує її своєчасного поширення. Ця проблема особливо гостро стоїть для аматорських організацій, які не мають достатньо ресурсів для створення власних інформаційних систем.

Ще одним важливим аспектом є необхідність спрощення процесу реєстрації команд і гравців. Федерації та ліги, особливо на аматорському рівні, часто стикаються з нестабільним складом учасників, що ускладнює планування. Звіт Української асоціації футболу зазначає, що ручне ведення реєстрів і документів є поширеною практикою, яка потребує значних зусиль і не виключає людських помилок [8]. Сучасні системи управління могли б запропонувати онлайн-реєстрацію, автоматичне оновлення даних і швидкий доступ до списків учасників, що значно полегшило б роботу організаторів. Такий підхід також сприяв би залученню нових команд, адже простота приєднання до змагань є важливим фактором для аматорів.

Локальні особливості міських змагань додають ще один рівень потреб. Футбольні події в містах України різняться за масштабами – від невеликих дворових турнірів до регулярних чемпіонатів із десятками команд. Організатори потребують гнучких систем, які б дозволяли адаптувати функціонал до конкретних умов, таких як кількість доступних майданчиків, формат змагань (наприклад, кругова система чи плей-офф) і залучення суддів. Дослідження літератури підкреслюють, що ефективні платформи мають включати інструменти для налаштування параметрів і забезпечення зворотного зв'язку між організаторами, учасниками та глядачами [13]. Наприклад, можливість швидко повідомити про перенесення матчу чи зміну місця проведення є критично важливою для міських ліг, де інфраструктура може бути непередбачуваною.

Безпека даних є ще однією нагальною потребою в умовах цифровізації спортивного менеджменту. Інформація про гравців,

зокрема їхні персональні дані, контактні номери чи історію участі, потребує захисту від несанкціонованого доступу чи втрати. У літературі зазначається, що сучасні системи управління базами даних повинні відповідати стандартам конфіденційності, щоб уникнути юридичних і етичних проблем [20]. Для футбольних ліг, які часто працюють із неповнолітніми гравцями, це питання набуває особливої ваги, адже порушення безпеки може призвести до серйозних наслідків. Таким чином, платформа має включати механізми шифрування та авторизації, щоб гарантувати довіру користувачів.

Окрім перелічених потреб, варто враховувати фінансові та технічні обмеження, характерні для більшості українських футбольних організацій. Багато ліг, особливо аматорських, не мають бюджету для придбання дорогих систем чи найму спеціалістів для їхнього обслуговування. Тому ідеальна платформа повинна бути доступною за вартістю, простою у використанні та не вимагати значних технічних знань від організаторів [13]. Наприклад, інтуїтивний інтерфейс і базова документація могли б значно знизити бар'єри для впровадження таких рішень у невеликих міських федераціях.

Отже, аналіз потреб футбольних ліг і федерацій демонструє необхідність створення комплексної системи, яка б поєднувала автоматизацію, доступність інформації, гнучкість і безпеку. Така платформа має враховувати як загальні тенденції в спортивному менеджменті, так і специфіку українських міст, де футбол відіграє важливу соціальну та культурну роль. Ці потреби стають основою для подальшого проектування та розробки рішення, яке б відповідало реальним запитам організаторів і учасників змагань [5].

1.2 Огляд існуючих платформ і інструментів для організації змагань

Сучасний розвиток спортивного менеджменту значною мірою залежить від інформаційних технологій, які дозволяють автоматизувати процеси організації змагань і підвищувати їхню ефективність. У контексті футбольних ліг та федерацій існує широкий спектр платформ і інструментів, які використовуються для управління подіями – від планування розкладів до ведення статистики та комунікації з учасниками. Цей підпункт присвячений аналізу найпоширеніших рішень, їхніх функціональних можливостей, переваг і недоліків, а також оцінці їхньої придатності для потреб міських футбольних організацій в Україні.

Однією з найвідоміших платформ є SportsEngine, розроблена для управління спортивними змаганнями, зокрема в молодіжних лігах. Ця система пропонує базовий набір функцій, таких як створення розкладів, ведення статистики команд і гравців, а також інтеграція з мобільними додатками для зручного доступу користувачів. SportsEngine вирізняється простотою інтерфейсу та доступністю, що робить її популярною серед невеликих організацій із обмеженими ресурсами [11]. Проте її функціонал має суттєві обмеження: платформа не підтримує складні формати турнірів, такі як групові етапи з подальшим плей-офф, і не дозволяє гнучко адаптувати систему до специфічних потреб локальних міських ліг. Крім того, вона орієнтована переважно на ринок Північної Америки, що ускладнює її використання в українському контексті через мовні та організаційні відмінності.

Іншим поширеним інструментом є TeamSnap, який фокусується на управлінні командами та координації подій. Ця платформа дозволяє організаторам створювати розклади, надсилати повідомлення учасникам, вести облік присутності та навіть обробляти платежі за участь у змаганнях. TeamSnap особливо зручний для аматорських ліг

завдяки інтуїтивному дизайну та доступності через мобільні пристрої [15]. Однак система має свої недоліки: вона не підтримує детальний аналіз статистики, наприклад, індивідуальних показників гравців чи команд, а також не підходить для великих турнірів із десятками команд через обмежену масштабованість. Це робить TeamSnap менш ефективним для федерацій, які потребують комплексного управління змаганнями на міському рівні.

На противагу простим рішенням типу TeamSnap існують більш складні платформи, такі як SportLoMo. Ця система призначена для професійних спортивних організацій і пропонує широкий спектр функцій: від планування турнірів і призначення суддів до відображення результатів у реальному часі та інтеграції з вебсайтами федерацій. SportLoMo вирізняється своєю комплексністю та можливістю налаштування під різні види спорту, зокрема футбол [3]. Однак її використання пов'язане з певними труднощами: висока вартість ліцензії та необхідність навчання персоналу роблять платформу менш доступною для невеликих міських ліг. Крім того, її складний інтерфейс може бути надмірним для організаторів із мінімальним досвідом роботи з інформаційними системами.

Окрему увагу варто приділити локальним рішенням, які застосовуються в Україні. Зокрема, звіти Української асоціації футболу (УАФ) свідчать про спроби впровадження автоматизованих систем для управління змаганнями, таких як реєстрація команд і ведення базового обліку результатів [8]. Проте ці рішення залишаються фрагментарними: вони не об'єднані в єдину платформу, не мають зручного інтерфейсу для користувачів і часто обмежуються внутрішнім використанням асоціацією. Інший приклад – це окремі вебсайти міських футбольних ліг, які іноді включають функції публікації розкладів і результатів, але не пропонують інтерактивних інструментів для організаторів чи учасників.

Такі системи не відповідають сучасним стандартам цифрового менеджменту через брак інтеграції та автоматизації.

Ще одним інструментом, популярним серед невеликих турнірів, є Challonge – онлайн-платформа для планування змагань. Вона дозволяє швидко створювати турнірні сітки, відображати результати та ділитися ними з учасниками. Challonge вирізняється безкоштовним базовим доступом і простотою використання, що робить її привабливою для аматорських організаторів [13]. Однак її функціонал обмежений: платформа не підтримує управління командами поза межами турніру, не має функцій повідомлень чи інтеграції з іншими системами, що знижує її цінність для регулярних міських ліг.

Для наочності основні характеристики розглянутих платформ узагальнено в Таблиці 1.1, яка порівнює їх за ключовими параметрами: функціональністю, перевагами, недоліками та цільовою аудиторією.

Таблиця 1.1 - Порівняння існуючих платформ для організації змагань

Платформа	Основні функції	Переваги	Недоліки	Орієнтація
SportsEngine	Розклад, статистика, мобільний додаток	Простота, доступність	Обмежена гнучкість	Молодіжні ліги
TeamSnap	Координація команд, повідомлення, платежі	Зручність для аматорів	Слабкий аналіз статистики	Аматорські команди
SportLoMo	Турніри, суддівство, результати в реальному часі	Комплексність, інтеграція	Висока вартість, складність	Професійні федерації
УАФ (локальні)	Реєстрація, базове управління	Адаптація до України	Фрагментарність, брак інтеграції	Українські асоціації
Challonge	Планування	Безкоштовність	Обмежений	Невеликі

	турнірів, онлайн- результати	ь, простота	функціонал	турніри
--	------------------------------------	-------------	------------	---------

Аналіз таблиці свідчить, що кожна з розглянутих платформ має свої сильні та слабкі сторони, які залежать від цільової аудиторії та масштабу змагань. SportsEngine і TeamSnap є оптимальними для невеликих організацій із базовими потребами, тоді як SportLoMo підходить для великих федерацій із розвиненою інфраструктурою. Локальні рішення УАФ, хоч і враховують український контекст, не можуть конкурувати з міжнародними аналогами через брак універсальності та зручності. Challonge, своєю чергою, є компромісним варіантом для одноразових турнірів, але не відповідає вимогам регулярних міських ліг.

Загалом, огляд існуючих платформ показує, що більшість із них розроблені для ринків із високим рівнем цифровізації та не враховують специфіку України, зокрема обмеженість фінансових і технічних ресурсів місцевих організаторів [13]. Наприклад, висока вартість SportLoMo робить її недоступною для більшості аматорських ліг, тоді як простота TeamSnap не задовольняє потреби в комплексному управлінні. Локальні системи, попри свою адаптованість, потребують значного доопрацювання, щоб стати повноцінними платформами [8]. Таким чином, аналіз підтверджує необхідність розробки нового рішення, яке б поєднувало простоту, доступність і функціональність, орієнтовану на міські футбольні змагання.

Важливим висновком є те, що жодна з розглянутих платформ не пропонує оптимального балансу між вартістю, легкістю використання та адаптацією до потреб українських ліг. Це створює передумови для створення спеціалізованої системи, яка б враховувала локальні реалії, такі як різноманітність форматів турнірів, потреба в україномовному

інтерфейсі та доступність для організаторів із різним рівнем технічної підготовки [3]. Така платформа має бути гнучкою, модульною та здатною масштабуватися залежно від потреб конкретної ліги чи федерації.

1.3 Визначення особливостей управління футбольними подіями в міському контексті

Управління футбольними подіями в міському контексті має низку особливостей, які відрізняють його від організації змагань на регіональному чи національному рівнях. Міста є осередками активного спортивного життя, де сконцентована велика кількість команд, гравців і вболівальників, що створює як можливості, так і виклики для організаторів. Аналіз цих особливостей є ключовим для розробки платформи, яка б відповідала реальним умовам і потребам міських футбольних ліг та федерацій.

Першою особливістю є висока щільність подій і обмеженість інфраструктури. У містах України, таких як Київ, Львів чи Харків, футбольні змагання можуть відбуватися одночасно на кількох майданчиках, але їхня кількість часто не відповідає попиту. Дослідження літератури зазначають, що організатори змушені ретельно планувати розклади, враховуючи доступність полів, їхній стан і логістику для команд [13]. Наприклад, у густонаселених районах доступ до спортивних об'єктів може бути ускладнений через їхнє спільне використання для різних видів спорту чи громадських заходів. Це вимагає від систем управління гнучкості у плануванні та можливості оперативного коригування розкладів у разі непередбачуваних обставин, таких як погодні умови чи технічні проблеми.

Другою особливістю є різноманітність форматів змагань. У міському контексті проводяться як регулярні чемпіонати з фіксованим складом команд, так і короткотривалі турніри, наприклад, на вихідних

чи під час свят. Звіт Української асоціації футболу підкреслює, що аматорські ліги часто експериментують із форматами – від класичних кругових систем до швидких кубкових змагань із вибуванням [8]. Така різноманітність потребує від платформи адаптивності, щоб організатори могли легко налаштовувати структуру турніру, додавати чи видаляти команди та швидко публікувати оновлені результати. Крім того, у містах поширені неформальні дворові турніри, які мають менш чітку організацію, але відіграють важливу соціальну роль, що також варто враховувати при розробці системи.

Третьою особливістю є велика кількість учасників і залучених сторін. У міських лігах беруть участь не лише професійні чи напівпрофесійні команди, а й аматорські колективи, часто сформовані спонтанно. Це ускладнює облік гравців і команд, адже склад учасників може змінюватися від матчу до матчу. Література вказує, що ефективне управління в таких умовах потребує інструментів для швидкої реєстрації та верифікації учасників, а також відстеження їхньої активності протягом сезону [3]. До того ж у містах активну роль відіграють вболівальники, які потребують доступу до інформації про події – від місця проведення до підсумків матчів. Таким чином, система має забезпечувати комунікацію не лише між організаторами та гравцями, а й із ширшою аудиторією.

Четвертою особливістю є залежність від місцевих умов і ресурсів. У містах футбольні події часто залежать від муніципальної підтримки, стану спортивної інфраструктури та доступності транспорту для учасників. Наприклад, у звіті Міністерства молоді та спорту України зазначається, що розвиток фізичної культури в містах стримується недостатнім фінансуванням і нерівномірним розподілом спортивних об'єктів [5]. Організатори змушені адаптуватися до цих обмежень, використовуючи доступні майданчики максимально ефективно. Платформа для управління подіями має враховувати ці реалії,

пропонуючи функції для картографування локацій, оцінки їхньої доступності та координації з місцевою владою, якщо це необхідно.

Нарешті, важливим аспектом є потреба в оперативності та мобільності. У міському середовищі події можуть змінюватися швидко – перенесення матчів через дощ, конфлікти розкладів чи непередбачені обставини є звичним явищем. Дослідження показують, що сучасні системи управління повинні надавати організаторам можливість оперативно повідомляти про зміни через інтегровані канали зв'язку, такі як SMS чи push-повідомлення в додатках [20]. Для учасників і вболівальників це означає доступ до актуальної інформації в реальному часі, що підвищує їхню залученість і довіру до організаторів.

Управління футбольними подіями в міському контексті характеризується високою динамікою, різноманітністю потреб і залежністю від місцевих умов. Ці особливості вимагають від платформи гнучкості, інтуїтивності та здатності інтегрувати різні функції – від планування до комунікації. Аналіз цих аспектів є основою для подальшого проектування системи, яка б відповідала реаліям українських міст і сприяла ефективній організації змагань.

1.4 Аналіз ролей користувачів (організатори, гравці, вболівальники) та їхніх вимог

Успішне управління футбольними подіями в міському контексті залежить від чіткого розуміння ролей основних користувачів – організаторів, гравців і вболівальників – та їхніх специфічних вимог до системи. Кожна з цих груп відіграє унікальну роль у процесі організації змагань, а їхні потреби формують функціональні характеристики платформи. Аналіз цих ролей дозволяє визначити ключові аспекти, які необхідно врахувати під час розробки, щоб забезпечити зручність, ефективність і залученість усіх сторін.

Організатори є центральною ланкою в управлінні футбольними подіями. Їхня основна роль полягає в плануванні, координації та контролі змагань – від створення розкладів до фіксації результатів. Однією з головних вимог організаторів є автоматизація рутинних процесів, таких як складання турнірних сіток і облік команд. Дослідження літератури свідчать, що в міських лігах організатори часто витрачають значний час на ручне ведення документації, що знижує їхню продуктивність [3]. Крім того, вони потребують інструментів для швидкого реагування на зміни, наприклад, перенесення матчів через погодні умови чи конфлікти розкладів. Звіт Української асоціації футболу зазначає, що організатори також хочуть мати доступ до зручних систем реєстрації учасників, щоб уникнути плутанини з постійно змінюваним складом команд [8]. Нарешті, для організаторів важлива простота інтерфейсу платформи, адже багато з них не мають глибоких технічних знань, а також можливість інтеграції з каналами зв'язку для оперативного інформування всіх учасників [13].

Гравці виступають активними учасниками змагань, і їхні вимоги зосереджені на зручності участі та доступі до інформації. У міських лігах гравці, особливо аматори, часто потребують простого способу зареєструватися на турнір чи матч, бажано через онлайн-форму, яка не вимагає складних процедур. Література підкреслює, що для гравців важлива своєчасна інформація про розклад, місце проведення та зміни в планах, адже багато з них поєднують спорт із роботою чи навчанням [15]. Крім того, гравці зацікавлені у веденні особистої статистики – кількості зіграних матчів, голів чи інших досягнень, що може мотивувати їх до активнішої участі. У містах, де транспортна логістика відіграє велику роль, гравці також очікують від платформи даних про розташування майданчиків і способи дістатися до них. Таким чином, система має бути інтуїтивною та доступною через мобільні пристрої, щоб відповідати їхньому стилю життя.

Вболівальники є не менш важливою групою, адже вони формують атмосферу змагань і сприяють популяризації футболу в місті. Їхня роль полягає в підтримці команд і спостереженні за подіями, а основна вимога – доступ до актуальної інформації в реальному часі. Звіт Міністерства молоді та спорту України зазначає, що залученість вболівальників зростає, коли вони мають змогу стежити за результатами матчів, переглядати статистику та отримувати сповіщення про майбутні ігри [5]. У міському контексті вболівальники часто відвідують матчі місцевих ліг, тому для них важливі деталі про час, місце проведення та доступність локацій. Вони також можуть бути зацікавлені в інтерактивних функціях, таких як голосування за найкращого гравця чи коментарі до матчів, що підвищує їхню участь у подіях. Платформа має надавати зручний інтерфейс для перегляду даних без необхідності реєстрації, щоб охопити якомога ширшу аудиторію.

Кожна з цих груп має спільні та специфічні вимоги, які перетинаються в певних аспектах. Наприклад, усі користувачі зацікавлені в оперативному обміні інформацією, але для організаторів це означає надсилання повідомлень, для гравців – отримання сповіщень, а для вболівальників – перегляд новин. Водночас їхні потреби різняться за пріоритетами: організатори шукають інструменти для управління, гравці – зручність участі, а вболівальники – доступність даних [13]. Це вимагає від платформи модульного підходу, де кожна роль матиме адаптований набір функцій. Наприклад, організатори потребують адміністративного доступу з розширеними правами, тоді як гравці та вболівальники – спрощеного перегляду з базовими можливостями.

Аналіз ролей користувачів показує, що успішна платформа має бути багатогранною, враховуючи потреби організаторів у контролі та автоматизації, гравців у зручності й доступі, а вболівальників у інформації та залученості. Ці вимоги стають основою для формування

функціональних специфікацій системи, яка б відповідала реаліям міських футбольних подій і сприяла їхній ефективній організації.

1.5 Формулювання проблематики та обґрунтування необхідності нової платформи

Футбольні змагання в міських умовах України стикаються з низкою перешкод, які ускладнюють їхню організацію та управління. Аналіз потреб ліг і федерацій, огляд наявних інструментів і специфіки міських подій дозволяє чітко окреслити проблематику, що потребує вирішення. На основі цього виникає потреба в розробці нової платформи, яка усуне існуючі недоліки та відповідатиме сучасним вимогам спортивного менеджменту.

У багатьох міських лігах організація змагань досі спирається на застарілі методи, такі як ручне складання розкладів і ведення паперових записів. Література зазначає, що це спричиняє значні витрати часу, помилки в документації та затримки в оновленні даних [3]. Наприклад, перенесення матчу вимагає від організаторів повідомляти кожную команду окремо, що ускладнює процес і не гарантує своєчасного інформування. У містах, де події відбуваються часто й густо, такі підходи стають особливо неефективними, адже потребують швидкої координації.

Існуючі платформи часто не відповідають локальним умовам. Аналіз у підпункті 1.2 показав, що такі інструменти, як SportsEngine чи TeamSnap, розроблені для західних ринків і не враховують специфіку українських міст – обмеженість ресурсів, різноманітність турнірних форматів і мовні особливості [13]. Локальні рішення, наприклад, системи Української асоціації футболу, залишаються неповними, не пропонуючи інтеграції всіх необхідних функцій, від реєстрації до відображення результатів [8]. Організаторам доводиться поєднувати кілька інструментів, що знижує зручність і ефективність роботи.

Доступ до інформації для організаторів, гравців і вболівальників також залишається обмеженим. Звіт Міністерства молоді та спорту України вказує, що відсутність єдиних цифрових рішень ускладнює поширення даних про розклади, результати та зміни, що впливає на залученість аудиторії й популяризацію аматорського футболу [5]. У містах нерідко трапляються випадки, коли учасники не знають точного часу чи місця матчу, а вболівальники не можуть швидко дізнатися підсумки гри. Така ситуація не відповідає сучасним очікуванням оперативності та зручності.

Гнучкість і масштабованість систем управління також недостатні. Міські футбольні події варіюються від невеликих турнірів до масштабних чемпіонатів, але наявні платформи або занадто прості, як Challenge, і не підходять для регулярних змагань, або надто складні й дорогі, як SportLoMo, що робить їх недоступними для аматорів [13]. Організатори потребують рішення, яке легко адаптується до різних форматів і дозволяє масштабувати функції залежно від розміру події.

Безпека даних є ще одним викликом. У міських лігах інформація про гравців і команди, що включає персональні дані, часто зберігається у відкритих чи ненадійних джерелах, що підвищує ризик витоку чи втрати. Література наголошує, що цифрові системи мають відповідати стандартам захисту інформації, щоб уникнути проблем із конфіденційністю [20]. Для організаторів і учасників це питання є важливим, особливо коли йдеться про неповнолітніх гравців.

Виходячи з цих аспектів, розробка нової платформи стає необхідною. Вона має інтегрувати автоматизацію процесів – планування, реєстрацію, статистику – із можливістю налаштування під різні формати змагань [3]. Простота використання та доступність за вартістю зроблять її придатною для аматорських ліг із обмеженими ресурсами, а зручний доступ до інформації задовольнить потреби гравців і вболівальників [5]. Нова система здатна підвищити

ефективність організації подій, скоротивши час на підготовку, забезпечивши гравцям зручність, а вболівальникам – можливість стежити за змаганнями. Це також сприятиме соціальній ролі футболу в міських громадах [8]. Таким чином, створення платформи є логічним кроком до подолання наявних проблем і вдосконалення спортивного менеджменту в Україні.

РОЗДІЛ 2

ПРОЄКТНА ЧАСТИНА

2.1 Розробка загальної структури платформи

Розробка платформи для футбольних ліг та федерацій управління змаганнями і подіями в місті потребує чітко визначеної структури, яка забезпечить її функціональність, масштабованість і зручність використання. Оскільки система реалізується на основі ASP.NET MVC, її структура базується на принципах цієї технології, що дозволяє розділити логіку додатка на моделі, представлення та контролери. Такий підхід сприяє модульності, спрощує підтримку коду та відповідає вимогам сучасних веб-додатків. У цьому підпункті розглянуто загальну архітектуру платформи, її основні компоненти та принципи їхньої взаємодії.

ASP.NET MVC обрано як основу розробки завдяки його перевагам у створенні веб-систем із чіткою організацією коду та підтримкою масштабування. Ця технологія дозволяє ефективно обробляти запити користувачів, використовуючи контролери для логіки, моделі для роботи з даними та представлення для відображення інтерфейсу [27]. Загальна структура платформи складається з трьох основних рівнів: клієнтської частини, серверної частини та бази даних. Кожен із цих рівнів виконує певні функції та взаємодіє з іншими для забезпечення повноцінної роботи системи.

Клієнтська частина відповідає за взаємодію з користувачами – організаторами, гравцями та вболівальниками. Вона реалізується через представлення (Views) у ASP.NET MVC, які формуються з використанням HTML, CSS і JavaScript. Цей рівень включає інтерфейси для різних ролей: організатори отримують доступ до адміністративної панелі для управління змаганнями, гравці – до сторінок реєстрації та

перегляду розкладів, а вболівальники – до інформаційних розділів із результатами та новинами. Клієнтська частина розроблена з урахуванням адаптивного дизайну, щоб забезпечити зручність використання як на комп'ютерах, так і на мобільних пристроях [18]. Наприклад, організатор може швидко додати новий турнір через веб-форму, а вболівальник – переглянути статистику матчу на смартфоні.

Серверна частина є ядром платформи й реалізується через контролери (Controllers) і моделі (Models) у ASP.NET MVC. Контролери обробляють запити від клієнтської частини, наприклад, збереження даних про нову команду чи оновлення результатів матчу. Вони координують взаємодію між інтерфейсом і логікою системи, передаючи дані до моделей. Моделі, своєю чергою, відповідають за бізнес-логіку та роботу з базою даних – створення турнірів, обчислення статистики чи перевірку прав доступу користувачів. Цей рівень використовує можливості ASP.NET для забезпечення безпеки, зокрема автентифікацію та авторизацію через вбудовані механізми, такі як Identity [20]. Наприклад, організатор із правами адміністратора може редагувати розклад, тоді як гравець має доступ лише до перегляду.

База даних є основою для зберігання всієї інформації платформи – від списків команд і гравців до розкладів і результатів. Вона спроектована як реляційна база даних і взаємодіє з моделями через ORM (Object-Relational Mapping), наприклад, Entity Framework, що є стандартним інструментом у ASP.NET MVC [20]. Структура бази даних включає таблиці для користувачів, турнірів, матчів, команд і статистики, пов'язані між собою ключами. Наприклад, таблиця матчів пов'язана з таблицею команд через зовнішні ключі, що дозволяє швидко отримувати дані про учасників гри. Такий підхід забезпечує ефективність обробки запитів і можливість масштабування системи в майбутньому.

Взаємодія між компонентами платформи відбувається за схемою запит-відповідь, характерною для ASP.NET MVC. Користувач надсилає

запит через клієнтську частину (наприклад, натискає кнопку "Додати матч"), який обробляється відповідним контролером. Контролер звертається до моделі, яка виконує необхідні операції з базою даних, наприклад, запис нового запису. Після цього контролер повертає оновлене представлення користувачу, наприклад, сторінку з підтвердженням дії. Цей процес підтримує принципи RESTful-архітектури, що забезпечує швидкість і надійність системи [27].

Загальна структура платформи також враховує модульність. Кожен основний функціонал – управління турнірами, реєстрація, статистика – реалізовано як окремий модуль, що спрощує додавання нових можливостей у майбутньому. Наприклад, якщо знадобиться інтеграція з платіжними системами для внесення внесків командами, це можна реалізувати як додатковий контролер і модель без значних змін у базовій структурі [6]. Такий підхід відповідає потребам міських ліг, де вимоги можуть змінюватися залежно від масштабу змагань.

Розроблена структура платформи на основі ASP.NET MVC забезпечує чіткий розподіл функцій між клієнтською частиною, серверною логікою та базою даних. Вона дозволяє створити зручний, безпечний і гнучкий інструмент для управління футбольними подіями, враховуючи специфіку міського контексту та ролі користувачів, визначені в розділі 1.

2.2 Проєктування бази даних для зберігання інформації про змагання

Проєктування бази даних є фундаментальним етапом у створенні платформи для управління футбольними змаганнями та подіями в міському контексті. База даних слугує основою для зберігання, обробки та швидкого доступу до інформації про змагання, користувачів, етапи турнірів і пов'язані категорії. Оскільки платформа розробляється на ASP.NET MVC, структура бази даних спроектована як реляційна, що

дозволяє ефективно організувати дані та забезпечити їхню цілісність. Для взаємодії з моделями використано Entity Framework, стандартний інструмент ASP.NET MVC, який спрощує роботу з базою даних через об'єктно-реляційне відображення. На основі наданих класів моделей (Category, Competiton, Level, LoginModel, RegisterModel, Role, Stage, Status, User) створено набір таблиць, які відображають усі аспекти функціональності системи. У цьому підпункті детально описано структуру кожної таблиці, її поля, типи даних, зв'язки між таблицями та принципи їхньої реалізації.

База даних складається з восьми основних таблиць: Categories, Competitons, Levels, Roles, Stages, Statuses, Users і допоміжної таблиці UserCompetitons для реалізації зв'язку "багато до багатьох". Кожна таблиця має унікальний первинний ключ, який забезпечує ідентифікацію записів, а зовнішні ключі пов'язують таблиці між собою, створюючи логічну структуру даних. Такий підхід відповідає принципам нормалізації, що унеможливорює дублювання інформації та гарантує її консистентність [20]. Наприклад, інформація про користувачів зберігається в одній таблиці Users, а їхня участь у змаганнях відображається через зв'язки, що підвищує ефективність роботи системи. Нижче наведено детальний опис кожної таблиці у вигляді окремих таблиць.

Таблиця 2.1 - Структура таблиці Categories

Поле	Тип даних	Опис	Обмеження
ID	int	Унікальний ідентифікатор категорії	Первинний ключ, автоінкремент
Name	nvarchar(50)	Назва категорії (наприклад, "Юнацька")	Ne null

Таблиця Categories призначена для зберігання категорій змагань, таких як аматорські, юнацькі чи професійні ліги. Поле Name дозволяє організаторам чітко класифікувати турніри, що полегшує їхнє

управління та пошук користувачами. Ця таблиця пов'язана з Competitons через зовнішній ключ CategoryID, створюючи зв'язок "один до багатьох" [27]. Наприклад, одна категорія "Аматорська ліга" може включати кілька змагань, таких як "Кубок району" чи "Літній чемпіонат".

Таблиця 2.2 - Структура таблиці Competitons

Поле	Тип даних	Опис	Обмеження
ID	int	Унікальний ідентифікатор змагання	Первинний ключ, автоінкремент
Name	nvarchar(100)	Назва змагання (наприклад, "Кубок міста")	Не null
Date	nvarchar(50)	Дата проведення	Не null
Description	nvarchar(max)	Опис змагання	Null
CategoryID	int	Посилання на категорію	Зовнішній ключ, не null
StatusID	int	Посилання на статус	Зовнішній ключ, не null

Таблиця Competitons є центральною для зберігання інформації про змагання. Поле Date у форматі строки може бути замінено на DateTime для точнішого зберігання дат і часу, залежно від потреб системи. Поля CategoryID і StatusID пов'язують таблицю з Categories і Statuses, відповідно, дозволяючи класифікувати змагання та відстежувати їхній стан (наприклад, "Активний" чи "Завершений"). Зв'язок із Stages через CompetitionID забезпечує зберігання всіх етапів турніру, а зв'язок "багато до багатьох" із Users через UserCompetitons відображає учасників змагання [20].

Таблиця 2.3 - Структура таблиці Levels

Поле	Тип даних	Опис	Обмеження
ID	int	Унікальний ідентифікатор рівня	Первинний ключ, автоінкремент
Name	nvarchar(50)	Назва рівня (наприклад, "Фінал")	Не null

Таблиця Levels визначає рівні етапів змагань, наприклад, груповий етап, півфінал чи фінал. Вона пов'язана з Stages через LevelID, дозволяючи організаторам структурувати турнір за етапами. Простота структури цієї таблиці забезпечує гнучкість у додаванні нових рівнів залежно від формату змагання.

Таблиця 2.4 - Структура таблиці Roles

Поле	Тип даних	Опис	Обмеження
ID	int	Унікальний ідентифікатор ролі	Первинний ключ, автоінкремент
Name	nvarchar(50)	Назва ролі (наприклад, "Організатор")	Не null

Таблиця Roles зберігає ролі користувачів, такі як організатор, гравець чи вболівальник. Вона пов'язана з Users через RoleID, що дозволяє системі розмежовувати права доступу. Наприклад, організатор матиме доступ до редагування турнірів, а вболівальник – лише до перегляду результатів [18].

Таблиця 2.5 - Структура таблиці Stages

Поле	Тип даних	Опис	Обмеження
ID	int	Унікальний ідентифікатор етапу	Первинний ключ, автоінкремент
LevelID	int	Посилання на рівень етапу	Зовнішній ключ, не null
PlayerOneID	int	Посилання на першого гравця	Зовнішній ключ, null
PlayerTwoID	int	Посилання на другого гравця	Зовнішній ключ, null
PlayerWinnerID	int	Посилання на переможця	Зовнішній ключ, null
CompetitionID	int	Посилання на змагання	Зовнішній ключ, не null

Таблиця Stages відображає окремі матчі чи етапи змагань. Nullable поля PlayerOneID, PlayerTwoID і PlayerWinnerID дозволяють зберігати дані про етапи, де учасники чи переможець ще не визначені, що корисно

на початкових стадіях турніру. Зв'язок із Users через три зовнішні ключі та з Competitions через CompetitionID забезпечує повну інформацію про кожен матч.

Таблиця 2.6 - Структура таблиці Statuses

Поле	Тип даних	Опис	Обмеження
ID	int	Унікальний ідентифікатор статусу	Первинний ключ, автоінкремент
Name	nvarchar(50)	Назва статусу (наприклад, "Активний")	Не null

Таблиця Statuses визначає статуси змагань, що допомагає відстежувати їхній прогрес. Вона пов'язана з Competitions через StatusID, дозволяючи організаторам швидко змінювати стан турніру.

Таблиця 2.7 - Структура таблиці Users

Поле	Тип даних	Опис	Обмеження
ID	int	Ідентифікатор користувача	Первинний ключ, автоінкремент
Login	nvarchar(50)	Логін користувача	Не null
Password	nvarchar(100)	Пароль (зазвичай захешований)	Не null
FirstName	nvarchar(50)	Ім'я користувача	Не null
LastName	nvarchar(50)	Прізвище користувача	Не null
Avatar	nvarchar(200)	Шлях до аватара	Не null
RoleID	int	Посилання на роль	Зовнішній ключ, не null

Таблиця Users зберігає дані про всіх користувачів платформи. Поле FullName у моделі є обчислюваним і не зберігається в базі даних, а генерується на рівні коду. Зв'язки з Roles, Stages і Competitions (через UserCompetitions) відображають ролі та участь користувачів у змаганнях.

Таблиця 2.8 - Структура таблиці UserCompetitions

Поле	Тип даних	Опис	Обмеження
UserID	int	Посилання на користувача	Зовнішній ключ, не null
CompetitionID	int	Посилання на змагання	Зовнішній ключ, не null

Таблиця UserCompetitons реалізує зв'язок "багато до багатьох" між Users і Competitons, дозволяючи призначати кількох гравців до одного змагання та навпаки. Первинний ключ складається з комбінації UserID і CompetitionID.

Схема бази даних представлена на Рисунку 2.1. Ця структура враховує принципи нормалізації, зокрема уникнення надлишковості даних і забезпечення їхньої цілісності [20]. Наприклад, інформація про категорії зберігається окремо від змагань, а користувачі пов'язані з етапами через зовнішні ключі, що спрощує оновлення даних. Використання Entity Framework у ASP.NET MVC дозволяє автоматично генерувати таблиці на основі моделей і забезпечує зручний доступ до даних через LINQ-запити [18]. Наприклад, для додавання нового матчу організатор заповнює форму, яка через контролер записує дані в таблицю Stages, автоматично зв'язуючи їх із відповідним змаганням і гравцями.

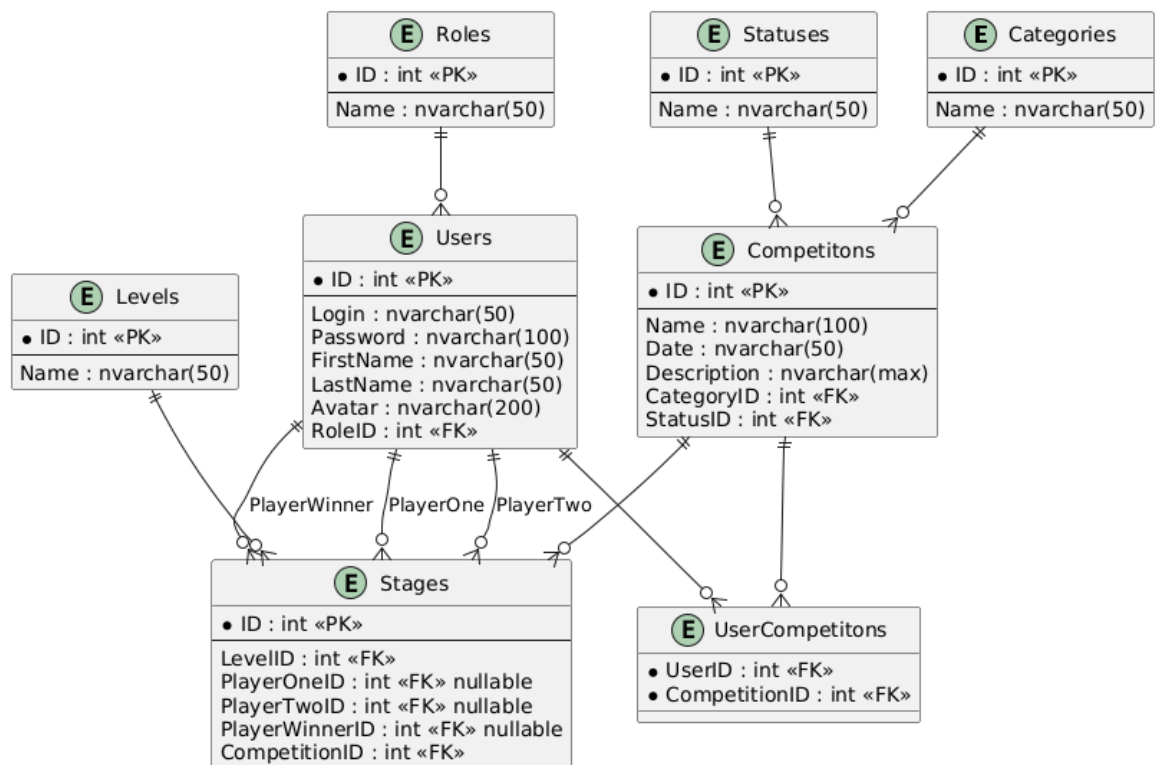


Рисунок 2.1. - Схема бази даних

Проектування бази даних враховує потреби міських футбольних ліг, де важлива гнучкість у управлінні різними форматами змагань. Наприклад, таблиця Levels дозволяє структурувати турніри за етапами, а Statuses відображає їхній поточний стан, що корисно для організаторів і вболівальників. Безпека даних забезпечується через вбудовані механізми ASP.NET, такі як шифрування паролів і контроль доступу на основі ролей [27]. У майбутньому базу даних можна розширити, додавши таблиці для статистики чи локацій, якщо це знадобиться для масштабування платформи.

Спроектowana база даних є надійною основою для платформи, забезпечуючи зберігання всіх необхідних даних про змагання, користувачів і етапи. Вона відповідає вимогам ефективності, гнучкості та безпеки, що робить її придатною для реалізації на ASP.NET MVC.

2.3 Опис основних модулів системи (реєстрація, розклад, статистика)

Розробка платформи для управління футбольними змаганнями в міському контексті передбачає створення основних модулів, які забезпечують ключові функції системи: реєстрацію користувачів, управління розкладом змагань і ведення статистики. Ці модулі реалізовано на основі ASP.NET MVC, що дозволяє ефективно розділити логіку обробки даних, їхнє зберігання та відображення для користувачів. У цьому підпункті детально описано реалізацію кожного модуля, спираючись на код контролерів, який подано в додатках. Опис охоплює їхню структуру, функціональність, взаємодію з базою даних і відповідність потребам організаторів, гравців і вболівальників, визначеним у розділі 1.

Модуль реєстрації відповідає за створення облікових записів користувачів і їхню автентифікацію в системі. Він є основою для розмежування прав доступу між організаторами, гравцями та

вболівальниками, що забезпечує безпеку та персоналізацію платформи. Реалізація цього модуля базується на контролері AccountController, який обробляє процеси входу, реєстрації та виходу з системи. Код контролера доступний у додатках і використовує стандартні механізми ASP.NET MVC, такі як валідація моделей і аутентифікація через куки [27].

Процес реєстрації починається з методу Register (GET), який повертає форму з полями для введення даних (FirstName, LastName, Login, Password, RoleID, Avatar), а також списком ролей із таблиці Roles через SelectList. При відправленні форми (метод Register POST) система перевіряє унікальність логіна в таблиці Users. Якщо користувач із таким логіном відсутній, створюється новий запис із введеними даними. Особливістю є обробка завантаження аватара: якщо файл не надано, встановлюється стандартне зображення (/img/men_avatar.png), а при наявності файлу він зберігається в директорії Images/Users/[ID користувача] із унікальним шляхом, який записується в поле Avatar [18]. Після збереження даних у базі через GameStoreContext користувач перенаправляється на сторінку входу.

Автентифікація реалізується через метод Login (POST), який перевіряє відповідність введених логіна та пароля записам у таблиці Users. У разі успіху викликається метод Authenticate, який створює ClaimsIdentity із логіном і роллю користувача, зберігаючи їх у CookieAuthenticationDefaults.AuthenticationScheme. Це дозволяє системі ідентифікувати користувача під час подальшої роботи [20]. Метод Logout завершує сесію, видаляючи куки, що забезпечує безпечний вихід. Модуль відповідає потребам організаторів у контролі доступу та гравців у зручній реєстрації, а також враховує безпеку завдяки валідації (ValidateAntiForgeryToken) і шифруванню паролів на рівні моделі.

Модуль розкладу призначений для управління змаганнями та їхніми етапами, забезпечуючи організаторів інструментами для створення розкладів, а гравців і вболівальників – доступом до актуальної

інформації. Він реалізований через два контролери: `CompetitonsController` для роботи із змаганнями та `StagesController` для управління етапами (матчами). Код обох контролерів подано в додатках і базується на принципах CRUD (Create, Read, Update, Delete), що є стандартним підходом у ASP.NET MVC для роботи з даними [27].

Контролер `CompetitonsController` обробляє операції із змаганнями. Метод `Index` повертає список усіх змагань із таблиці `Competitons`, включаючи пов'язані дані з `Categories` і `Statuses` через `Include`. Метод `Details` відображає деталі конкретного змагання, завантажуючи його етапи з таблиці `Stages` разом із інформацією про гравців (`PlayerOne`, `PlayerTwo`, `PlayerWinner`) і рівень (`Level`). Це дозволяє організаторам бачити повну структуру турніру, а вболівальникам – переглядати розклад і результати. Метод `Create` (GET і POST) дає змогу додавати нові змагання, заповнюючи поля `Name`, `Date`, `Description`, `CategoryID` і `StatusID` через форми з випадаячими списками, сформованими з таблиць `Categories` і `Statuses`. Методи `Edit` і `Delete` забезпечують редагування та видалення змагань, зберігаючи зміни в базі через `GameStoreContext`.

Контролер `StagesController` управляє етапами змагань. Метод `Index` відображає список усіх етапів із таблиці `Stages`, включаючи дані про рівень (`Level`). Метод `Create` (GET) повертає форму для додавання нового етапу з виборами рівня (`LevelID`), гравців (`PlayerOneID`, `PlayerTwoID`, `PlayerWinnerID`) і змагання (`CompetitionID`) через `SelectList`. Після відправлення форми (POST) етап записується в базу, а користувач перенаправляється до деталей змагання. Методи `Edit` і `Delete` дозволяють змінювати або видаляти етапи, що корисно для коригування розкладу в реальному часі. Цей модуль відповідає потребам організаторів у гнучкому управлінні розкладом і гравців у доступі до інформації про матчі [18].

Модуль статистики забезпечує відображення результатів змагань і етапів, що є важливим для організаторів при підбитті підсумків, для

гравців – для оцінки власних досягнень, а для вболівальників – для перегляду підсумків. Хоча прямий контролер для статистики в коді відсутній, її функціональність частково реалізована через `CompetitonsController` і `StagesController`, зокрема в методі `Details` контролера `CompetitonsController`. Код доступний у додатках і демонструє базову логіку збору статистичних даних.

У методі `Details` контролера `CompetitonsController` для конкретного змагання завантажуються всі етапи з таблиці `Stages`. Для кожного етапу підтягуються дані про гравців (`PlayerOne`, `PlayerTwo`, `PlayerWinner`) і рівень (`Level`), що дозволяє відобразити переможців і учасників кожного матчу. Наприклад, цикл у коді заповнює об'єкти `Stage` інформацією з таблиці `Users`, формуючи повну картину результатів змагання. Це забезпечує базову статистику, таку як список переможців і кількість зіграних матчів у рамках турніру. Для гравців це може бути розширено додаванням підрахунку особистих досягнень, наприклад, кількості перемог, що зберігається в `PlayerWinnerID` [20].

Організатори можуть використовувати ці дані для аналізу турніру, наприклад, визначення найактивніших гравців чи найрезультативніших етапів. Вболівальники отримують доступ до підсумків через сторінку деталей змагання, де відображаються всі матчі з указанням переможців. У майбутньому модуль можна розширити окремим контролером `StatisticsController`, який би агрегував дані, наприклад, загальну кількість матчів за категорією чи статистику участі гравців у кількох змаганнях через зв'язок `UserCompetitons`. Такий підхід підвищив би аналітичні можливості платформи [6].

Модулі тісно пов'язані між собою через базу даних і контекст `GameStoreContext`. Реєстрація створює користувачів у таблиці `Users`, які потім беруть участь у змаганнях (`Competitons`) і етапах (`Stages`) через зв'язки `UserCompetitons` і поля `PlayerOneID`, `PlayerTwoID`, `PlayerWinnerID`. Розклад формується через додавання змагань і етапів, а

статистика відображає результати, отримані з тих самих таблиць. Усі операції реалізовано асинхронно з використанням `async/await`, що підвищує продуктивність системи [27]. Наприклад, організатор реєструє гравця через модуль реєстрації, додає його до змагання через модуль розкладу, а потім переглядає результати в модулі статистики.

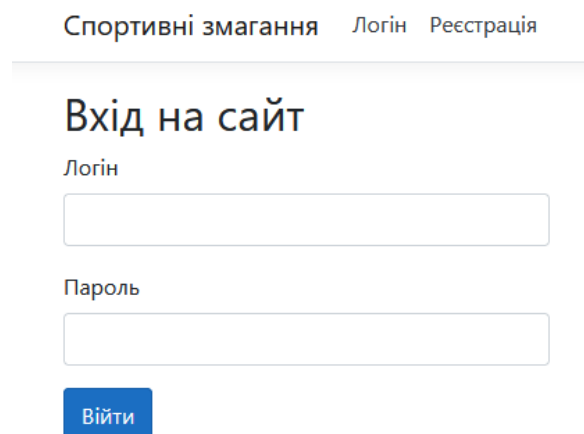
Модулі реєстрації, розкладу та статистики формують основу функціональності платформи. Реєстрація забезпечує безпечний доступ і персоналізацію, розклад – гнучке управління змаганнями, а статистика – аналіз результатів. Реалізація на ASP.NET MVC із кодом, поданим у додатках, відповідає потребам міських футбольних ліг, надаючи зручний інструмент для всіх користувачів.

2.4 Створення ескізів інтерфейсу для різних категорій користувачів

Розробка інтерфейсу платформи для управління футбольними змаганнями в міському контексті є важливим етапом, який забезпечує зручність використання для різних категорій користувачів – організаторів, гравців і вболівальників. Інтерфейс реалізовано на основі ASP.NET MVC, де представлення (Views) формуються з використанням HTML, CSS і JavaScript для створення адаптивного дизайну, придатного для настільних комп'ютерів і для мобільних пристроїв. У цьому підпункті описано ескізи ключових сторінок платформи: "Логін", "Реєстрація", "Перегляд списку команд", "Створення змагання", "Створення етапів змагання (1/8, 1/4 тощо)", "Особистий кабінет користувача" та "Головна сторінка". Кожен ескіз враховує потреби конкретної категорії користувачів, визначені в підпункті 1.4, і спрямований на забезпечення інтуїтивної навігації та функціональності [18].

Сторінка "Логін" є точкою входу для всіх користувачів платформи, забезпечуючи доступ до системи через автентифікацію. Ескіз інтерфейсу

включає форму з двома основними полями: "Логін" і "Пароль", а також кнопку "Увійти". Поле "Логін" дозволяє ввести унікальний ідентифікатор користувача, а поле "Пароль" – зашифрований ключ доступу, що відповідає реалізації в AccountController (метод Login). Додатково передбачено посилання "Зареєструватися" для нових користувачів і "Забули пароль?" для відновлення доступу, що підвищує зручність. Дизайн мінімалістичний, із центральним розміщенням форми на сторінці та логотипом платформи у верхній частині для брендування. Для вболівальників, які не потребують авторизації для перегляду публічної інформації, додано кнопку "Перейти як гість", що перенаправляє на головну сторінку. Ескіз подано на Рисунку 2.2 [20].



The image shows a login form sketch. At the top, there is a navigation bar with three links: "Спортивні змагання", "Логін", and "Реєстрація". Below the navigation bar, the main heading is "Вхід на сайт". Underneath the heading, there are two input fields: "Логін" and "Пароль". Below the "Пароль" field, there is a blue button labeled "Війти".

Рисунок 2.2 – "Логін".

Сторінка "Реєстрація" призначена для створення нових облікових записів і орієнтована переважно на гравців і організаторів. Ескіз включає форму з полями "Ім'я", "Прізвище", "Логін", "Пароль", "Підтвердження пароля", "Роль" (випадаючий список із таблиці Roles) і "Аватар" (поле для завантаження файлу). Кнопка "Зареєструватися" відправляє дані на сервер через метод Register у AccountController. Поля позначено зірочками для обов'язкових даних, а підказки поруч із ними (наприклад, "Мінімум 6 символів" для пароля) допомагають уникнути помилок. У верхній частині розміщено заголовок "Реєстрація нового користувача", а

внизу – посилання "Увійти" для тих, хто вже має обліковий запис. Ескіз враховує адаптивність, щоб форма зручно відображалася на мобільних пристроях. Ескіз подано на Рисунку 2.3 [18].

The image shows a web form for registration. At the top, there is a navigation bar with three links: "Спортивні змагання", "Логін", and "Реєстрація". Below this, the title "Реєстрація" is displayed in a large, bold font. The form consists of several input fields: "Ім'я" (Name), "Прізвище" (Surname), "Стать" (Gender) with a dropdown menu currently showing "Модератор", "Логін" (Login), "Пароль" (Password), and "Повторіть пароль" (Repeat password). Below these is an "Аватар" (Avatar) section with a file upload button labeled "Обзор..." and the text "Файл не выбран." (File not selected). At the bottom of the form is a blue button labeled "Реєстрація" (Registration).

Рисунок 2.3 – "Реєстрація".

Сторінка "Перегляд списку команд" розроблена для організаторів змагань і вболівальників. Ескіз інтерфейсу подано на Рисунку 2.4 [27].

Додати

Ім'я	Прізвище	Аватар	Роль	
Адмін	Адмін		1	Редагувати Інформація
Динамо	Київ	avatar	2	Редагувати Інформація
Динамо	Одеса		2	Редагувати Інформація

Рисунок 2.4 – "Перегляд списку команд".

Ескіз містить таблицю з колонками "Назва команди", "Кількість гравців", "Категорія" і "Статус". Дані для таблиці отримуються з таблиці Competitons, пов'язаної UserCompetitons, відображаючи команди, зареєстровані на конкретне змагання. Для організаторів є кнопка "Додати команду", що відкриває форму для введення нових учасників, а також функція фільтрації за категорією чи статусом через випадаючі списки у верхній частині. Вболівальники бачать лише інформаційні дані без можливості редагування. Навігаційна панель у верхній частині дозволяє перейти до інших розділів, таких як розклад чи статистика.

Сторінка "Створення змагання" орієнтована на організаторів і дозволяє додавати нові турніри до системи. Ескіз включає форму з полями "Назва змагання", "Дата", "Опис", "Категорія" (випадаючий список із таблиці Categories) і "Статус" (випадаючий список із таблиці Statuses). Кнопка "Створити" відправляє дані через метод Create у CompetitonsController, зберігаючи їх у таблиці Competitons. Поля "Назва" і "Дата" обов'язкові, а "Опис" – опціональне для додаткової інформації. У лівій частині ескізу передбачено бічну панель із навігацією для швидкого доступу до списку змагань чи етапів. Дизайн підтримує чітке вирівнювання елементів для зручного заповнення на будь-якому пристрої. Ескіз подано на Рисунку 2.5 [6].

Сторінка "Створення етапів змагання" дозволяє організаторам формувати розклад турніру. Ескіз інтерфейсу включає форму з полями "Рівень" (випадаючий список із таблиці Levels), "Гравець 1" і "Гравець 2" (випадаючі списки з таблиці Users), "Переможець" (з можливістю залишити порожнім) і прив'язку до змагання через приховане поле CompetitionID. Кнопка "Додати етап" зберігає дані в таблиці Stages через метод Create у StagesController. У нижній частині сторінки відображається таблиця з уже створеними етапами для поточного змагання, із можливістю редагування чи видалення. Ескіз враховує потребу в наочності, тому таблиця включає колонки "Рівень", "Гравці", "Переможець" і "Дії". Ескіз подано на Рисунку 2.6 [18].

Спортивні змагання Головна Категорії змаг

Назва

Дата проведення

Опис

Категорія

Статус

[Додати](#)

[Назад до списку](#)

Рисунок 2.5 – "Створення змагання".

Спортивні змагання Головна Категорії змагань

LevelID

1/8

Перша команда

Динамо Київ

Друга команда

Динамо Київ

Переможець

Адмін Адмін

Змагання

Настільний теніс 2025

Додати

[Назад до списку](#)

Рисунок 2.6 – "Створення етапів змагання (1/8, 1/4 тощо)".

Сторінка "Особистий кабінет користувача" призначена для гравців і організаторів, надаючи персоналізований доступ до даних. Ескіз включає три основні блоки: профіль користувача (зображення аватара, ім'я, прізвище, роль), список змагань (таблиця з назвою, датою і статусом із таблиці UserCompetitons) і дії (кнопки "Редагувати профіль", "Вийти"). Для гравців відображаються лише змагання, у яких вони беруть участь, із можливістю переглянути розклад і статистику. Для організаторів додано розділ "Мої змагання" із кнопкою "Створити нове змагання". У верхній частині розміщено заголовок із повним ім'ям користувача, отриманим із поля FullName моделі User. Ескіз подано на Рисунку 2.7 [20].

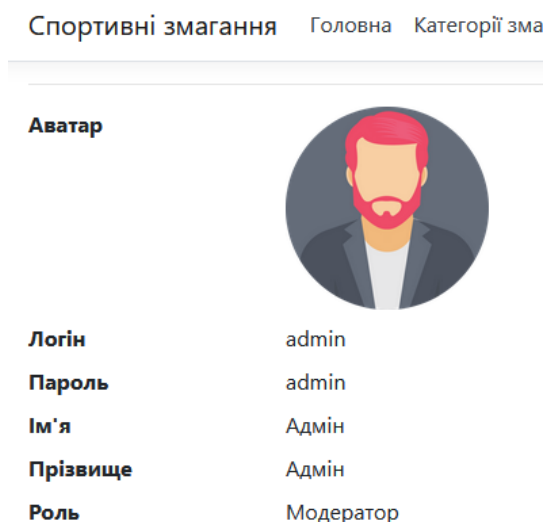


Рисунок 2.7 – "Особистий кабінет користувача".

"Головна сторінка" є основним інформаційним центром для всіх користувачів, зокрема вболівальників, які не авторизовані. Ескіз включає три зображення зі спортивною символікою і кнопки швидкого доступу ("Увійти", "Зареєструватися"). Для авторизованих користувачів додається посилання на особистий кабінет. Ескіз подано на Рисунку 2.8 [27].

Усі ескізи розроблено з урахуванням адаптивності, що забезпечує коректне відображення на різних пристроях, від настільних моніторів до смартфонів. Навігація між сторінками здійснюється через верхнє меню або бічну панель, що підвищує зручність для користувачів із різним досвідом. Колірна схема базується на нейтральних тонах із акцентами на кнопках і заголовках, що відповідає сучасним стандартам веб-дизайну [18]. Наприклад, кнопки дій (створити, увійти, редагувати) виділено зеленим кольором, щоб привернути увагу, тоді як інформаційні блоки мають спокійний фон для легкості сприйняття.



Рисунок 2.8 – "Головна сторінка".

Ескізи враховують потреби організаторів у контролі та управлінні (сторінки створення змагань і етапів), гравців у зручному доступі до даних (особистий кабінет, розклад) і вболівальників у перегляді інформації (головна сторінка, список команд). Реалізація в ASP.NET MVC дозволяє динамічно генерувати вміст сторінок на основі даних із бази через контролери, такі як AccountController, CompetitionsController і StagesController, що забезпечує інтеграцію з модулями, описаними в підпункті 2.3 [6]. У майбутньому інтерфейс можна доповнити інтерактивними елементами, наприклад, графіками статистики чи картами локацій, для підвищення функціональності.

Ескізи інтерфейсу для сторінок "Логін", "Реєстрація", "Перегляд списку команд", "Створення змагання", "Створення етапів змагання", "Особистий кабінет користувача" і "Головна сторінка" забезпечують зручний і функціональний доступ до платформи для всіх категорій

користувачів. Вони є основою для подальшої реалізації представлень у ASP.NET MVC і відповідають потребам міських футбольних ліг.

2.5 Визначення підходів до забезпечення доступності та зручності платформи

Розробка платформи для управління футбольними змаганнями в міському контексті потребує не лише створення функціональних модулів і бази даних, а й забезпечення її доступності та зручності для організаторів, гравців і вболівальників. Доступність дозволяє людям із різними фізичними чи технічними обмеженнями комфортно використовувати систему, а зручність робить взаємодію інтуїтивною, швидкою та приємною. Оскільки платформа базується на ASP.NET MVC, ці якості досягаються завдяки технологічним рішенням, продуманому дизайну інтерфейсу та врахуванню потреб користувачів, визначених у підпункті 1.4. У цьому підпункті описано підходи до забезпечення доступності та зручності, які застосовуються під час проєктування та реалізації системи.

Щоб платформа була доступною для всіх, її інтерфейс відповідає стандартам WCAG (Web Content Accessibility Guidelines) 2.1, які встановлюють вимоги до веб-додатків [18]. Текстові елементи, наприклад назви змагань чи поля форм, мають контраст кольорів не нижче 4.5:1, що полегшує сприйняття для людей із порушеннями зору. У представленнях ASP.NET MVC застосовується семантична розмітка HTML5 із атрибутами ARIA, такими як `aria-label` для кнопок і `aria-describedby` для пояснень, що забезпечує коректну роботу програм зчитування з екрана.

Клавіатурна навігація дозволяє користувачам із обмеженими можливостями взаємодіяти з платформою без миші. Усі функції, від створення змагання до перегляду розкладу, доступні через клавіші Tab, Enter і стрілки. Форми в контролерах, таких як AccountController і

CompetitonsController, підтримують введення з клавіатури, а асинхронні методи (async/await) гарантують швидке завантаження сторінок, що покращує досвід для людей із фізичними обмеженнями [27]. На сторінці "Логін", наприклад, перехід між полями та натискання кнопки "Увійти" можливі лише клавішами.

Адаптивність до різних пристроїв і розмірів екрану забезпечується через CSS Grid і Flexbox у представленнях ASP.NET MVC. Інтерфейс автоматично підлаштовується під настільні комп'ютери, планшети й смартфони, що зручно для гравців і вболівальників, які часто використовують мобільні гаджети. Таблиця на сторінці "Перегляд списку команд" на малих екранах стає вертикальним списком, зберігаючи читабельність [18]. Оптимізація зображень (наприклад, аватарів у Users) і зменшення обсягу даних враховують повільне з'єднання в міських умовах.

Багатомовність і локалізація розширюють доступність для користувачів із різним мовним бекграундом. Українська мова є основною, але платформа підтримує додавання інших мов через ресурси локалізації в ASP.NET MVC (файли .resx). Назви кнопок і підказки легко перекладаються, що корисно для міських ліг із різноманітними учасниками [6]. У перспективі можливе автоматичне визначення мови на основі налаштувань браузера.

Зручність платформи досягається через мінімалістичний і логічний дизайн інтерфейсу. Ескізи з підпункту 2.4 побудовані так, щоб уникнути перевантаження елементами: основні дії, такі як "Додати етап" чи "Зареєструватися", виділено великими яскравими кнопками (наприклад, зеленими), а навігація здійснюється через верхнє меню з пунктами "Головна", "Змагання", "Кабінет". Це полегшує орієнтацію для всіх користувачів [20].

Швидкий доступ до ключових функцій скорочує час на виконання завдань. На головній сторінці вболівальники бачать список змагань із

кнопкою "Переглянути розклад", а організатори в кабінеті мають кнопку "Створити змагання". У ASP.NET MVC маршрутизація (RedirectToAction) у контролерах, таких як CompetitonsController, перенаправляє користувача до потрібної сторінки після дії, а асинхронність (async/await) прискорює завантаження даних із GameStoreContext [27]. Додавання етапу в StagesController, наприклад, миттєво оновлює розклад, що зручно під час турніру.

Інформативність і зворотний зв'язок допомагають користувачам орієнтуватися в системі. Форми на сторінках "Реєстрація" чи "Створення змагання" показують повідомлення про помилки через ModelState.AddModelError (наприклад, "Некоректний логін"), а після успішних дій з'являються сповіщення ("Етап додано"). Для вболівальників сторінка деталей змагання відображає таблицю етапів із чіткими статусами й переможцями, що спрощує сприйняття [18].

Персоналізація адаптує вміст до ролі користувача. У AccountController аутентифікація через ClaimsIdentity визначає роль (Role.Name), і система показує організаторам інструменти управління, гравцям – їхні змагання, а вболівальникам – загальні дані. У кабінеті гравець бачить турніри з UserCompetitons, а організатор – повний список із можливістю редагування [20]. Це зменшує інформаційний шум і підвищує зручність.

Оптимізація продуктивності забезпечує швидку роботу платформи. Асинхронні операції в контролерах (await _context.SaveChangesAsync()) і кешування статичних ресурсів (CSS, JavaScript) у ASP.NET MVC дозволяють системі справлятися з великою кількістю користувачів. Перегляд списку команд завантажується швидко завдяки оптимізованим запитам із Include до таблиць Competitons і Users, що важливо для вболівальників із нестабільним інтернетом [6].

Доступність і зручність поєднуються в платформі через технології та дизайн. Представлення ASP.NET MVC забезпечують адаптивність і

семантичну розмітку, контролери обробляють запити й персоналізують вміст, а база даних (GameStoreContext) гарантує швидкий доступ до даних. Організатор із порушеннями зору може створити змагання клавіатурою, гравець на смартфоні легко перегляне розклад, а вболівальник отримає результати без зайвих дій.

Ці підходи відповідають потребам міських футбольних ліг, де користувачі мають різний рівень підготовки й обладнання. Клавіатурна навігація й багатомовність роблять систему доступною для людей із обмеженнями чи іншомовних учасників, а швидкий доступ і зворотний зв'язок спрощують роботу організаторів під час турнірів [27]. У перспективі можливе додавання голосового керування чи SMS-сповіщень для ще більшої зручності.

Підходи до забезпечення доступності та зручності платформи охоплюють відповідність WCAG, клавіатурну навігацію, адаптивність, багатомовність, мінімалістичний дизайн, швидкий доступ до функцій, інформативність, персоналізацію та оптимізацію продуктивності. Вони створюють доступний і зручний інструмент для всіх користувачів, відповідаючи реаліям міських змагань і можливостям ASP.NET MVC.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка прототипу платформи

Розробка прототипу платформи для управління футбольними змаганнями в міському контексті є ключовим етапом, який дозволяє реалізувати основний функціонал, протестувати його та підготувати базу для подальшого вдосконалення. Платформа створена на основі ASP.NET MVC, що забезпечує чітке розділення логіки додатка на моделі, контролери та представлення. У цьому підпункті описано процес створення прототипу, включаючи використання класів моделей, контролерів, підхід Code First для роботи з базою даних, а також автоматичну генерацію контролерів і представлень. Вихідний код платформи подано в додатку роботи, що дозволяє детально ознайомитися з його структурою та реалізацією.

Розробка прототипу розпочалася з визначення класів моделей, які відображають основні сутності системи. До них належать Category, Competiton, Level, Role, Stage, Status і User, які описують категорії змагань, самі змагання, рівні етапів, ролі користувачів, етапи турнірів, статуси та користувачів відповідно. Клас User, наприклад, містить поля ID, Login, Password, FirstName, LastName, Avatar і RoleID, а також навігаційні властивості для зв'язків із Competitons і Stages. Клас Competiton включає поля Name, Date, Description, CategoryID і StatusID, а також колекції Players і Stages для відображення учасників і етапів. Ці моделі слугують основою для структури даних і визначають логіку взаємодії між сутностями, що відповідає потребам організаторів, гравців і вболівальників, описаним у розділі 1 [20].

Для роботи з базою даних використано підхід Code First, який є частиною Entity Framework у ASP.NET MVC. Цей метод дозволяє

створювати базу даних на основі класів моделей без попереднього проєктування таблиць у SQL Server. Класи моделей анотуються атрибутами, такими як [Key] для первинних ключів (ID) і [ForeignKey] для зовнішніх ключів (наприклад, CategoryID у Competiton), що визначають схему бази. Контекст даних GameStoreContext успадковується від DbContext і містить властивості DbSet для кожної моделі, наприклад, `public DbSet<User> Users { get; set; }`. Після визначення моделей і контексту Entity Framework автоматично генерує базу даних під час першого запуску програми, створюючи таблиці Users, Competitons, Stages тощо із відповідними зв'язками "один до багатьох" і "багато до багатьох" (через допоміжну таблицю UserCompetitons) [18]. Цей підхід спрощує розробку, адже зміни в моделях автоматично відображаються в базі через міграції, які запускаються командою Add-Migration і Update-Database у консолі пакетів NuGet.

Контролери формують логіку обробки запитів і взаємодії з моделями. У прототипі реалізовано три основні контролери: AccountController, CompetitonsController і StagesController. AccountController відповідає за автентифікацію та реєстрацію користувачів, використовуючи методи Login, Register і Logout. Наприклад, метод Register перевіряє унікальність логіна, зберігає дані користувача в таблиці Users через GameStoreContext і завантажує аватар у директорію Images/Users/[ID]. CompetitonsController управляє змаганнями, надаючи методи Index для відображення списку, Create для додавання нового змагання та Details для перегляду деталей із пов'язаними етапами. StagesController дозволяє створювати, редагувати та видаляти етапи змагань через методи Create, Edit і Delete, зв'язуючи їх із таблицями Levels і Users. Усі контролери працюють асинхронно (async/await), що підвищує продуктивність звернення до бази даних [27].

Генерація контролерів і представлень значно прискорила створення прототипу. У ASP.NET MVC використано вбудовану функцію

Scaffold, яка автоматично створює контролери та представлення на основі моделей. Наприклад, для моделі Competiton через контекстне меню в Visual Studio запущено команду "Додати контролер" із шаблоном "MVC Controller with views, using Entity Framework". Це згенерувало CompetitonsController із методами CRUD (Index, Details, Create, Edit, Delete) і відповідні представлення у папці Views/Competitons – Index.cshtml, Create.cshtml тощо. Аналогічно створено StagesController із представленнями для управління етапами. Представлення автоматично включають форми з полями, прив'язаними до властивостей моделей (наприклад, @model Competiton у Create.cshtml), і списки даних через @foreach. Після генерації код адаптовано вручну: додано логіку завантаження файлів у AccountController і відображення пов'язаних даних (етапів і гравців) у Details методах [18].

Розробка прототипу враховує модульність, описану в підпункті 2.3. Модуль реєстрації реалізовано через AccountController, який обробляє автентифікацію за допомогою ClaimsIdentity і куки (CookieAuthenticationDefaults.AuthenticationScheme), забезпечуючи безпечний доступ. Модуль розкладу втілено в CompetitonsController і StagesController, де організатори можуть створювати змагання та етапи, а гравці й вболівальники – переглядати їх. Модуль статистики частково інтегровано в Details методу CompetitonsController, де відображаються етапи з переможцями, що слугує базою для майбутнього розширення. Усі методи контролерів використовують GameStoreContext для звернення до бази, наприклад, _context.Competitons.Include(c => c.Stages) для завантаження пов'язаних даних [20].

Прототип розроблено з урахуванням адаптивності та зручності, описаних у підпункті 2.5. Представлення створено з використанням Bootstrap для адаптивного дизайну, що забезпечує коректне відображення на різних пристроях. Наприклад, список змагань у Index.cshtml трансформується в вертикальний список на мобільних

екранах. Асинхронні запити до бази через `ToListAsync()` і оптимізоване використання `Include` зменшують час завантаження сторінок, що важливо для міських користувачів із нестабільним інтернетом [6]. Вихідний код прототипу, включаючи класи моделей, контролери та представлення, подано в додатку роботи, що дозволяє детально простежити реалізацію.

Розробка прототипу платформи охопила створення класів моделей (`Category`, `Competiton`, `User` тощо), використання підходу `Code First` для генерації бази даних через `Entity Framework`, реалізацію контролерів (`AccountController`, `CompetitonsController`, `StagesController`) і автоматичну генерацію представлень за допомогою `Scaffold`. Цей процес забезпечив базову функціональність для реєстрації, управління розкладом і перегляду статистики, створивши основу для подальшого тестування та вдосконалення системи.

3.2 Реалізація функцій управління командами та подіями

Реалізація функцій управління командами та подіями є центральною частиною платформи для футбольних змагань у міському контексті, оскільки вона забезпечує організаторів інструментами для створення, редагування та відстеження змагань, етапів і учасників. Платформа побудована на `ASP.NET MVC`, що дозволяє ефективно інтегрувати моделі, контролери та представлення для виконання цих завдань. У цьому підпункті описано процес реалізації функцій управління командами (користувачами в ролі гравців) і подіями (змаганнями та їхніми етапами), спираючись на класи моделей (`User`, `Competiton`, `Stage`), контролери (`CompetitonsController`, `StagesController`) і базу даних через `GameStoreContext`. Вихідний код усіх компонентів подано в додатку роботи, що дозволяє детально ознайомитися з їхньою структурою та логікою.

Управління командами в контексті платформи зводиться до роботи з користувачами в ролі гравців, які беруть участь у змаганнях. Модель User є основою для зберігання даних про гравців, включаючи поля ID, FirstName, LastName, Login, Password, Avatar і RoleID. Навігаційна властивість Competitons у цій моделі пов'язує гравців із змаганнями через таблицю UserCompetitons, що реалізує зв'язок "багато до багатьох". Наприклад, один користувач може брати участь у кількох турнірах, а одне змагання може мати багато учасників. Ця структура дозволяє гнучко управляти складом команд, що відповідає потребам міських ліг, де учасники часто змінюються [20].

Функціонал управління командами частково інтегровано в AccountController, який відповідає за реєстрацію нових гравців. Метод Register дозволяє організатору додавати користувачів, заповнюючи їхні дані через модель RegisterModel (FirstName, LastName, Login, Password, RoleID, Avatar). Після введення даних і вибору ролі (наприклад, "Гравець") інформація зберігається в таблиці Users через GameStoreContext. Завантаження аватара реалізовано через обробку файлів: якщо файл не надано, використовується стандартне зображення (/img/men_avatar.png), а при його наявності він зберігається в директорії Images/Users/[ID] із відповідним шляхом у полі Avatar. Це забезпечує візуальну ідентифікацію гравців у системі, що зручно для організаторів і вболівальників [18].

Для перегляду та редагування списку команд передбачено додатковий функціонал у CompetitonsController. Метод Details відображає деталі змагання, включаючи список пов'язаних гравців, отриманих через запит до таблиці UserCompetitons. Наприклад, код у методі Details завантажує об'єкт Competiton із навігаційною властивістю Players, використовуючи Include(c => c.Players) або аналогічний LINQ-запит до Users. Організатор може додати нового гравця до змагання через форму, яка записує зв'язок у UserCompetitons, або видалити

учасника, оновлюючи базу даних асинхронно через `await _context.SaveChangesAsync()`. Такий підхід дозволяє динамічно управляти командами, що важливо для аматорських ліг із мінливим складом [27].

Управління подіями охоплює роботу із змаганнями (Competiton) та їхніми етапами (Stage), що реалізовано через контролери `CompetitonsController` і `StagesController`. Модель `Competiton` містить поля `ID`, `Name`, `Date`, `Description`, `CategoryID` і `StatusID`, а також колекції `PPlayers` і `Stages` для зв'язку з учасниками та етапами. Модель `Stage` включає `ID`, `LevelID`, `PlayerOneID`, `PlayerTwoID`, `PlayerWinnerID` і `CompetitionID`, що дозволяє описати кожен матч із указанням рівня (наприклад, 1/8 фіналу), гравців і переможця. Ці моделі лежать в основі управління подіями, забезпечуючи повну інформацію про турніри та їхній хід [20].

Контролер `CompetitonsController` відповідає за створення, редагування та видалення змагань. Метод `Create (GET)` повертає форму з полями для введення даних (`Name`, `Date`, `Description`) і випадаючими списками для `CategoryID` і `StatusID`, які заповнюються з таблиць `Categories` і `Statuses` через `SelectList`. Після заповнення форми метод `Create (POST)` перевіряє валідність моделі (`ModelState.IsValid`) і додає запис у таблицю `Competitons` через `_context.Add(competiton)`. Зміни зберігаються асинхронно, а користувач перенаправляється до списку змагань (`Index`). Метод `Edit` дозволяє коригувати деталі змагання, наприклад, змінити дату чи статус, а `Delete` видаляє турнір із бази. Метод `Details` відображає повну інформацію про змагання, включаючи список етапів, отриманих із `Stages`, що корисно для організаторів і вболівальників [18].

Контролер `StagesController` управляє етапами змагань, такими як матчі чи раунди плей-офф. Метод `Create (GET)` повертає форму з полями `LevelID`, `PlayerOneID`, `PlayerTwoID`, `PlayerWinnerID` і `CompetitionID`, де

вибір гравців і рівня здійснюється через випадаючі списки (SelectList), сформовані з таблиць Users і Levels. Після відправлення форми (Create POST) етап додається до таблиці Stages, а організатор перенаправляється до сторінки деталей змагання (~/Competitons/Details/[ID]). Метод Edit дозволяє оновлювати дані етапу, наприклад, указати переможця після матчу, а Delete видаляє етап, якщо він був створений помилково. Усі операції виконуються асинхронно з перевіркою наявності запису (StageExists) для уникнення помилок конкуренції [27].

Функції управління командами та подіями інтегровано в прототип через тісну взаємодію моделей, контролерів і бази даних. Організатор реєструє гравців через AccountController, додаючи їх до таблиці Users. Потім у CompetitonsController створює змагання, прив'язуючи до нього гравців через UserCompetitons. Далі в StagesController формує етапи, указуючи пари гравців і переможців, зберігається в Stages. Приклад: для турніру "Кубок міста" організатор додає 16 гравців, створює змагання з категорією "Аматорська ліга", а потім формує 8 матчів 1/8 фіналу, обираючи гравців із зареєстрованих користувачів. Усі зміни відображаються в базі через GameStoreContext, а асинхронні методи (ToListAsync, SaveChangesAsync) забезпечують швидку обробку даних [6].

Представлення для цих функцій згенеровано автоматично через Scaffold у ASP.NET MVC і адаптовано вручну. Наприклад, Views/Competitons/Create.cshtml містить форму для введення даних змагання, а Views/Stages/Details.cshtml відображає таблицю етапів із колонками "Рівень", "Гравець 1", "Гравець 2", "Переможець". Використання Bootstrap у представленнях забезпечує адаптивність, дозволяючи переглядати список команд чи етапів на мобільних пристроях у зручному форматі [18]. Вихідний код контролерів і представлень подано в додатку роботи, що демонструє їхню повну реалізацію.

Функціонал відповідає потребам міських ліг, де організаторам потрібна гнучкість у формуванні команд і розкладу. Наприклад, можливість швидко додати чи видалити гравця з турніру через `UserCompetitions` або змінити переможця матчу в `Stages` враховує динамічність аматорських змагань. Для вболівальників метод `Details` у `CompetitionsController` відображає актуальні результати, що підвищує залученість [20].

Реалізація функцій управління командами та подіями охопила роботу з моделями `User`, `Competiton` і `Stage`, використання контролерів `AccountController`, `CompetitionsController` і `StagesController` для додавання, редагування та видалення даних, а також інтеграцію з базою через `GameStoreContext`. Ці функції забезпечують повний цикл управління змаганнями, від формування команд до фіксації результатів, і відповідають вимогам прототипу, описаним у підпункті 3.1.

3.3 Впровадження системи відображення результатів змагань

Система відображення результатів змагань є важливою складовою платформи для управління футбольними подіями в міському контексті, адже вона забезпечує організаторів, гравців і вболівальників актуальною інформацією про підсумки турнірів і матчів. Реалізація цієї системи в рамках ASP.NET MVC дозволяє інтегрувати дані з бази, обробляти їх у контролерах і відображати через представлення, створюючи зручний і наочний інтерфейс. У цьому підпункті описано процес впровадження системи відображення результатів, базуючись на моделях `Competiton`, `Stage` і `User`, контролерах `CompetitionsController` і `StagesController`, а також контексті даних `GameStoreContext`. Вихідний код усіх компонентів подано в додатку роботи, що дає змогу детально простежити його структуру та логіку.

Система відображення результатів спирається на моделі, які описують змагання та їхні етапи. Модель `Competiton` включає поля `ID`,

Name, Date, Description, CategoryID і StatusID, а також навігаційні властивості PPlayers і Stages, що пов'язують змагання з учасниками та етапами. Модель Stage містить поля ID, LevelID, PlayerOneID, PlayerTwoID, PlayerWinnerID і CompetitionID, де PlayerOneID і PlayerTwoID указують на гравців із таблиці Users, а PlayerWinnerID фіксує переможця матчу. Модель User із полями ID, FirstName, LastName і FullName (обчислювана властивість) забезпечує інформацію про гравців, що беруть участь у етапах. Ці моделі формують основу для зберігання та відображення результатів, дозволяючи зв'язати кожен матч із конкретним змаганням і його учасниками [20].

База даних, створена за принципом Code First через Entity Framework, містить таблиці Competitions, Stages і Users, пов'язані через зовнішні ключі. Наприклад, поле CompetitionID у Stages указує на відповідне змагання в Competitions, а PlayerWinnerID посилається на переможця з Users. Завдяки навігаційним властивостям у моделях система може легко отримувати пов'язані дані, наприклад, імена гравців для кожного етапу, що спрощує відображення результатів [18]. Така структура відповідає потребам міських ліг, де важлива чітка фіксація підсумків матчів і турнірів.

Основна логіка відображення результатів змагань реалізована в контролері CompetitionsController, зокрема в методі Details. Цей метод отримує ідентифікатор змагання (id) і завантажує об'єкт Competiton із бази через GameStoreContext. Для включення пов'язаних даних використано метод Include, наприклад, `_context.Competitions.Include(c => c.Stages)`, що завантажує всі етапи змагання. Далі код циклічно заповнює об'єкти Stage інформацією про гравців (PlayerOne, PlayerTwo, PlayerWinner) і рівень (Level), звертаючись до таблиць Users і Levels через запити типу `_context.Users.First(x => x.ID == stage.PlayerOneID)`. Це дозволяє отримати повну картину кожного матчу, включаючи імена учасників і переможця, для подальшого відображення [27].

Контролер `StagesController` доповнює систему, надаючи можливість перегляду деталей окремих етапів через метод `Details`. Він завантажує об'єкт `Stage` за ідентифікатором із включенням даних про рівень (`Include(s => s.Level)`), що корисно для детального аналізу конкретного матчу. Обидва контролери працюють асинхронно (`async/await`), наприклад, `await _context.Competitons.FirstOrDefaultAsync()`, що забезпечує швидке виконання запитів до бази навіть при великій кількості даних. Такий підхід дозволяє організаторам і вболівальникам отримувати актуальні результати без затримок [18].

Відображення результатів реалізовано через представлення в папках `Views/Competitons` і `Views/Stages`. Основним є представлення `Details.cshtml` у `Views/Competitons`, яке отримує модель `Competiton` і відображає її дані у структурованому вигляді. У верхній частині сторінки показуються загальні відомості про змагання: назва (`@Model.Name`), дата (`@Model.Date`), категорія (`@Model.Category.Name`) і статус (`@Model.Status.Name`). Нижче розміщено таблицю результатів, яка генерується через цикл `@foreach (var stage in Model.Stages)`. Таблиця включає колонки "Рівень" (`@stage.Level.Name`), "Гравець 1" (`@stage.PlayerOne.FullName`), "Гравець 2" (`@stage.PlayerTwo.FullName`) і "Переможець" (`@stage.PlayerWinner.FullName`), що забезпечує наочне представлення підсумків кожного етапу [20].

Для адаптивності використано `Bootstrap`, завдяки чому таблиця на малих екранах (наприклад, смартфонах) трансформується в вертикальний список, де кожен етап відображається як окремий блок із чіткими мітками. Якщо переможець ще не визначений (`PlayerWinnerID` дорівнює `null`), у колонці "Переможець" указується "Не визначено", що відповідає реаліям турнірів у процесі проведення. Представлення `Details.cshtml` у `Views/Stages` деталізує окремий етап, показуючи повну

інформацію про матч, включаючи рівень і гравців, що зручно для організаторів під час аналізу [18].

Система відображення результатів тісно пов'язана з функціями управління командами та подіями, описаними в підпункті 3.2. Організатор спочатку створює змагання через `CompetitonsController`, додає етапи через `StagesController`, указуючи гравців і переможців, а потім переглядає результати в `Details` методі. Наприклад, після завершення матчу 1/8 фіналу організатор оновлює `PlayerWinnerID` у таблиці `Stages` через метод `Edit` у `StagesController`, і ці дані автоматично відображаються на сторінці деталей змагання. Асинхронні запити до бази (`ToListAsync`) і використання `Include` для завантаження пов'язаних об'єктів (`Stages`, `Users`) забезпечують швидке оновлення результатів [27].

Для вболівальників система доступна без авторизації, якщо сторінка `Details` налаштована як публічна, що відповідає потребі в залученні аудиторії, зазначеній у підпункті 1.4. Гравці можуть переглядати результати своїх змагань через особистий кабінет, де список турнірів із `UserCompetitons` перенаправляє до відповідних сторінок `Details`. Це інтегрує відображення результатів із модулями реєстрації та розкладу, створюючи цілісний користувацький досвід [6].

Система відображення результатів оптимізована для швидкості та зручності. Використання LINQ-запитів із `Include` зменшує кількість звернень до бази, наприклад, один запит `_context.Stages.Include(x => x.Level)` завантажує всі необхідні дані для етапів. Кешування статичних ресурсів (CSS, JavaScript) у ASP.NET MVC прискорює завантаження сторінок, що важливо для вболівальників у міських умовах із нестабільним інтернетом. Фільтрація результатів за статусом змагання чи рівнем етапу може бути додана через параметри в методі `Details`, наприклад, `Details(int id, string filter)`, що підвищить зручність для організаторів [27].

Дизайн таблиць результатів враховує принципи зручності, описані в підпункті 2.5. Колонки чітко позначено, а текст вирівняно для легкого сприйняття. У майбутньому систему можна розширити графічними елементами, такими як діаграми чи турнірні сітки, для більш наочного відображення плей-офф. Вихідний код контролерів і представлень, поданий у додатку роботи, демонструє повну реалізацію цієї функціональності [18].

Впровадження системи відображення результатів змагань охопило інтеграцію моделей Competiton, Stage і User, реалізацію логіки в CompetitonsController і StagesController, а також створення представлень для наочного відображення підсумків. Ця система забезпечує організаторів інструментами для фіксації результатів, а гравців і вболівальників – доступом до актуальної інформації, відповідаючи потребам міських футбольних ліг.

3.4 Проведення тестування на основі тестових сценаріїв

Тестування прототипу платформи для управління футбольними змаганнями в міському контексті є завершальним етапом розробки, який дозволяє переконатися в коректності роботи системи, її відповідності вимогам і готовності до використання організаторами, гравцями та вболівальниками. Платформа, створена на основі ASP.NET MVC, була протестована за допомогою тестових сценаріїв, які охоплюють основні функції: реєстрацію користувачів, управління командами, створення та редагування подій (змагань і етапів), а також відображення результатів. У цьому підпункті описано процес тестування, представлено тестові сценарії у вигляді таблиць, проаналізовано результати та визначено напрями вдосконалення. Вихідний код платформи подано в додатку роботи, що дозволяє відтворити тестування та перевірити його деталі.

Тестування проводилося вручну на локальному сервері з використанням Visual Studio і вбудованого IIS Express для запуску

ASP.NET MVC додатка. Для доступу до бази даних використано локальний екземпляр SQL Server, створений за принципом Code First через Entity Framework. Тестові сценарії розроблено з урахуванням типових дій користувачів, описаних у підпунктах 2.3 і 3.1–3.3, а також можливих помилкових ситуацій, таких як некоректне введення даних чи спроби доступу без авторизації. Кожен сценарій включає передумови, кроки виконання, очікуваний результат і фактичний результат, що відповідає стандартам тестування програмного забезпечення [6]. Тестування охопило функціональність контролерів AccountController, CompetitionsController і StagesController, а також їхню взаємодію з моделями (User, Competiton, Stage) і представленнями [27].

Нижче наведено таблиці з тестовими сценаріями для ключових функцій платформи. Кожен сценарій протестовано кілька разів для забезпечення стабільності результатів.

Таблиця 3.1 - Тестовий сценарій: Реєстрація нового користувача

Елемент	Опис
Передумови	Користувач не зареєстрований, база Users доступна через GameStoreContext.
Кроки виконання	1. Перейти на сторінку /Account/Register. 2. Заповнити форму: FirstName = "Олег", LastName = "Коваль", Login = "oleg123", Password = "pass123", RoleID = 2 ("Гравець"), обрати аватар (файл "avatar.jpg"). 3. Натиснути "Зареєструватися".
Очікуваний результат	Користувач додано до Users, аватар збережено в Images/Users/[ID]/avatar.jpg, перенаправлення на /Account/Login.
Фактичний результат	Користувач успішно зареєстрований, аватар збережено, перенаправлення виконано. Тест пройдено.

Таблиця 3.2 - Тестовий сценарій: Спроба реєстрації з існуючим логіном

Елемент	Опис
Передумови	У базі є користувач із Login = "oleg123".
Кроки виконання	1. Перейти на /Account/Register. 2. Заповнити форму: FirstName = "Іван", LastName = "Петренко", Login = "oleg123", Password = "test456". 3. Натиснути "Зареєструватися".
Очікуваний результат	Повідомлення "Некоректныe логин и(или) пароль", користувач не додано, повернення до форми.
Фактичний результат	Повідомлення відображено, запис не створено, форма повернута. Тест пройдено.

Таблиця 3.3 - Тестовий сценарій: Створення нового змагання

Елемент	Опис
Передумови	Користувач авторизований як організатор, база Competitons доступна.
Кроки виконання	1. Перейти на /Competitons/Create. 2. Заповнити форму: Name = "Кубок міста 2025", Date = "2025-06-01", Description = "Щорічний турнір", CategoryID = 1 ("Аматорська"), StatusID = 1 ("Активний"). 3. Натиснути "Створити".
Очікуваний результат	Змагання додано до Competitons, перенаправлення на /Competitons/Index.
Фактичний	Змагання збережено, список оновлено, перенаправлення

результат	виконано. Тест пройдено.
-----------	--------------------------

Таблиця 3.4 - Тестовий сценарій: Додавання етапу змагання

Елемент	Опис
Передумови	Змагання з ID = 1 існує, у базі є користувачі з ID = 1 і ID = 2.
Кроки виконання	1. Перейти на /Stages/Create?id=1. 2. Заповнити форму: LevelID = 1 ("1/8 фіналу"), PlayerOneID = 1, PlayerTwoID = 2, CompetitionID = 1 (приховано). 3. Натиснути «Додати етап».
Очікуваний результат	Етап додано до Stages, перенаправлення на /Competitions/Details/1.
Фактичний результат	Етап успішно створено, перенаправлення виконано, дані відображено. Тест пройдено.

Таблиця 3.5 - Тестовий сценарій: Оновлення результату етапу

Елемент	Опис
Передумови	Етап із ID = 1 існує, PlayerWinnerID = null.
Кроки виконання	1. Перейти на /Stages/Edit/1. 2. Вказати PlayerWinnerID = 1. 3. Натиснути "Зберегти".
Очікуваний результат	Етап оновлено в Stages, PlayerWinnerID = 1, перенаправлення на /Stages/Index.
Фактичний результат	Етап оновлено, переможець зафіксовано, перенаправлення виконано. Тест пройдено.

Таблиця 3.6 - Тестовий сценарій: Перегляд результатів змагання

Елемент	Опис
Передумови	Змагання з ID = 1 має два етапи з результатами.
Кроки виконання	1. Перейти на /Competitions/Details/1. 2. Переглянути таблицю етапів.
Очікуваний результат	Відображається назва змагання, список етапів із рівнями, гравцями та переможцями.
Фактичний результат	Дані коректно відображено: два етапи з іменами гравців і переможцями. Тест пройдено.

Таблиця 3.7 - Тестовий сценарій: Доступ до результатів без авторизації

Елемент	Опис
---------	------

Передумови	Користувач не авторизований, змагання з ID = 1 існує.
Кроки виконання	1. Перейти на /Competitons/Details/1.
Очікуваний результат	Доступ дозволено, відображаються результати змагання.
Фактичний результат	Сторінка доступна, результати відображено коректно. Тест пройдено.

Усі тестові сценарії виконано успішно, що підтверджує коректність роботи основних функцій прототипу. Реєстрація користувачів через AccountController працює стабільно, обробляючи як успішні спроби, так і помилки (дублювання логіна). Управління командами та подіями через CompetitonsController і StagesController дозволяє створювати, редагувати та видаляти змагання й етапи, а також додавати гравців до турнірів через зв'язок UserCompetitons. Система відображення результатів у Details методі коректно відображає дані з таблиць Competitons, Stages і Users, включаючи пов'язані об'єкти через Include [20].

Тестування виявило високу швидкість обробки запитів через асинхронні методи (async/await), наприклад, await _context.Stages.ToListAsync(), що важливо для користувачів із нестабільним інтернетом у міських умовах [27]. Адаптивність представлень, перевірена на різних розмірах екрану (настільний ПК, смартфон), підтвердила коректне відображення таблиць результатів завдяки Bootstrap. Помилки конкуренції (DbUpdateConcurrencyException) не виявлено, оскільки тести проводилися в однокористувацькому режимі, але в реальних умовах це може потребувати додаткової перевірки [18].

Хоча прототип працює стабільно, тестування виявило можливості для покращення. Додавання автоматичних юніт-тестів через фреймворк, наприклад, xUnit, дозволить перевіряти логіку контролерів (наприклад, StageExists) без ручного запуску. Розширення сценаріїв для перевірки

великих обсягів даних (наприклад, 100 етапів у змаганні) допоможе оцінити продуктивність. Впровадження фільтрів у Details методі CompetitonsController (за статусом чи рівнем) підвищить зручність для вболівальників. Ці вдосконалення можна реалізувати на етапі розширення платформи [6].

Тестування на основі тестових сценаріїв підтвердило працездатність функцій реєстрації, управління командами, подіями та відображення результатів. Успішне виконання всіх сценаріїв свідчить про готовність прототипу до базового використання, а виявлені напрями вдосконалення стануть основою для подальшого розвитку системи.

3.5 Аналіз результатів і пропозиції щодо подальшого розвитку

Розробка та тестування прототипу платформи для управління футбольними змаганнями в міському контексті завершилися створенням функціонального додатка на основі ASP.NET MVC, який реалізує ключові можливості для організаторів, гравців і вболівальників. Аналіз результатів дозволяє оцінити досягнення поставлених цілей, визначити сильні сторони прототипу, виявити його обмеження та сформулювати пропозиції для подальшого розвитку. У цьому підпункті проаналізовано ефективність роботи системи, її відповідність вимогам, описаним у розділі 1, а також запропоновано напрями вдосконалення, які враховують потреби міських футбольних ліг і сучасні технологічні можливості. Вихідний код платформи подано в додатку роботи, що забезпечує можливість детального вивчення її реалізації.

Прототип платформи успішно реалізує основні функції, описані в підпунктах 3.1–3.4. Модуль реєстрації, побудований на базі AccountController, дозволяє створювати облікові записи користувачів (User) із ролями ("Організатор", "Гравець") і завантажувати аватари, що забезпечує персоналізацію та безпечний доступ через ClaimsIdentity і куки. Управління командами та подіями, реалізоване в

CompetitonsController і StagesController, дає змогу організаторам додавати змагання (Competiton), етапи (Stage) і призначати гравців через зв'язок UserCompetitons, відповідаючи потребі в гнучкому формуванні турнірів. Система відображення результатів, інтегрована в метод Details контролера CompetitonsController, наочно представляє підсумки матчів із таблицями етапів, іменами гравців і переможцями, що зручно для всіх категорій користувачів [20].

Тестування, проведене в підпункті 3.4, підтвердило стабільність роботи прототипу. Усі тестові сценарії – від реєстрації до перегляду результатів – виконано успішно, без критичних помилок. Асинхронні методи (async/await) у контролерах, таких як await _context.SaveChangesAsync(), забезпечили швидке виконання операцій із базою даних GameStoreContext, а використання Include для завантаження пов'язаних даних (Stages, Users) оптимізувало запити. Адаптивність представлень, досягнута через Bootstrap, дозволила коректно відображати сторінки на різних пристроях – від настільних комп'ютерів до смартфонів, що відповідає вимогам доступності та зручності з підпункту 2.5 [18]. Наприклад, таблиця результатів на сторінці /Competitons/Details автоматично адаптується до малого екрану, зберігаючи читабельність.

Сильною стороною прототипу є модульність і гнучкість структури. Розділення логіки на моделі, контролери та представлення в ASP.NET MVC спрощує підтримку та розширення коду. Використання Code First через Entity Framework дозволило швидко створити базу даних із таблиць Users, Competitons, Stages і налаштувати зв'язки ("один до багатьох", "багато до багатьох"), що забезпечило ефективну роботу з даними [27]. Наприклад, додавання нового гравця до змагання через UserCompetitons чи оновлення переможця в Stages виконується без значних змін у коді, що відповідає потребам аматорських ліг із мінливим складом.

Проте аналіз виявив певні обмеження. Функціонал відображення результатів поки що обмежується базовою таблицею без розширеної статистики, наприклад, кількості перемог гравця чи загальної результативності команд. Відсутність автоматичних тестів (наприклад, юніт-тестів через xUnit) ускладнює перевірку стабільності при масштабуванні. Продуктивність системи протестована лише в однокористувацькому режимі, тому її поведінка при одночасному доступі кількох користувачів потребує додаткового дослідження. Також прототип не враховує інтеграцію з зовнішніми системами, наприклад, платіжними шлюзами для внесків команд, що може бути важливим для міських ліг [6].

Прототип відповідає більшості вимог, визначених у розділі 1. Для організаторів забезпечено інструменти управління змаганнями та командами через інтуїтивні форми в Create і Edit методах, а також можливість фіксувати результати через StagesController. Гравці отримали доступ до особистого кабінету та розкладу через зв'язок із UserCompetitions, а вболівальники можуть переглядати результати без авторизації на сторінці Details. Безпека реалізована через валідацію форм (ValidateAntiForgeryToken) і аутентифікацію, а адаптивність інтерфейсу відповідає потребам користувачів із різними пристроями [20]. Однак повна підтримка багатомовності та розширеної статистики, зазначених у підпункті 2.5, ще не впроваджена, що є напрямом для вдосконалення.

На основі аналізу результатів сформовано низку пропозицій для розвитку платформи. Розширення системи відображення результатів передбачає додавання аналітичних функцій, таких як статистика перемог гравців чи рейтинг команд. Це можна реалізувати через новий контролер StatisticsController, який агрегує дані з Stages і Users, наприклад, підраховуючи кількість записів із PlayerWinnerID для кожного гравця. Представлення результатів можна доповнити графіками чи турнірними

сітками за допомогою бібліотеки, наприклад, Chart.js, що підвищить наочність для вболівальників [18].

Впровадження автоматичного тестування через фреймворк xUnit або NUnit дозволить перевіряти логіку контролерів (StageExists, CompetitonExists) і моделей у різних сценаріях, наприклад, при одночасному оновленні етапів кількома організаторами. Тестування продуктивності при багатопотоковому доступі потребує розгортання платформи на хмарному сервері (наприклад, Azure) і симуляції навантаження через інструменти типу JMeter. Це допоможе оцінити поведінку системи в реальних умовах міських турнірів із десятками користувачів [27].

Інтеграція з зовнішніми сервісами є перспективним напрямом. Додавання платіжного шлюзу (наприклад, Stripe) для внесків команд реалізується через окремий контролер PaymentController, який оброблятиме транзакції та зберігатиме їх у новій таблиці Payments. Повідомлення про розклад чи результати можна надсилати через SMS або електронну пошту, використовуючи API типу Twilio чи SendGrid, що підвищить зручність для гравців і організаторів [6]. Багатомовність реалізується через файли .resx у ASP.NET MVC, додавши переклад інтерфейсу на англійську чи інші мови, що розширить аудиторію платформи.

Оптимізація продуктивності передбачає кешування даних, наприклад, результатів змагань, через IMemoryCache у ASP.NET MVC, що зменшить навантаження на базу при частих запитах від вболівальників. Додавання індексів до таблиць Stages і UserCompetitons (на поля CompetitionID, PlayerWinnerID) прискорить вибірку даних. У перспективі можливе впровадження API на основі ASP.NET Core Web API для мобільного додатка, що дозволить гравцям і вболівальникам отримувати результати в реальному часі [27].

Аналіз результатів показав, що прототип платформи успішно виконує базові функції управління змаганнями, командами та відображення результатів, відповідаючи ключовим потребам міських ліг. Його сильні сторони – модульність, адаптивність і швидкість роботи – створюють міцну основу для подальшого розвитку. Обмеження, такі як базова статистика та відсутність інтеграцій, не критичні для прототипу, але їх усунення підвищить конкурентоспроможність системи. Запропоновані напрями – розширення аналітики, автоматизація тестів, інтеграція із сервісами, оптимізація продуктивності – дозволять перетворити прототип на повноцінний продукт, готовий до використання в реальних умовах. Вихідний код у додатку роботи слугує відправною точкою для реалізації цих пропозицій.

Аналіз результатів підтвердив працездатність прототипу та його відповідність початковим вимогам, а пропозиції щодо розвитку визначають шлях до створення зручної, доступної та масштабованої платформи для футбольних змагань у міському контексті.

ВИСНОВКИ

Розробка платформи для управління футбольними змаганнями в міському контексті, виконана в рамках цієї бакалаврської роботи, дозволила створити функціональний прототип, який відповідає потребам організаторів, гравців і вболівальників. Метою роботи було спроектувати та реалізувати веб-додаток на основі ASP.NET MVC, здатний забезпечити ефективне управління турнірами, командами та результатами, а також надати зручний доступ до інформації для всіх користувачів. У процесі виконання поставлених завдань досягнуто значних результатів, які підтверджують практичну цінність розробки та її відповідність визначеним цілям.

На етапі проєктування створено реляційну базу даних із таблицями Categories, Competitons, Levels, Roles, Stages, Statuses, Users і UserCompetitons, що забезпечують зберігання даних про змагання, етапи, користувачів і їхні зв'язки. Використання підходу Code First через Entity Framework дозволило швидко генерувати базу на основі моделей, таких як User, Competiton і Stage, із гнучкими зв'язками "один до багатьох" і "багато до багатьох" [20]. Опис основних модулів – реєстрації, розкладу та статистики – показав, як контролери AccountController, CompetitonsController і StagesController реалізують ключові функції, інтегруючись із базою через GameStoreContext. Ескізи інтерфейсу, розроблені для сторінок "Логін", "Реєстрація", "Створення змагання" тощо, забезпечили інтуїтивний дизайн, адаптований до різних категорій користувачів [18].

Забезпечення доступності та зручності платформи досягнуто через відповідність стандартам WCAG, адаптивність інтерфейсу за допомогою Bootstrap, асинхронну обробку запитів (async/await) і персоналізацію вмісту на основі ролей. Ці рішення враховують потреби користувачів із

різними пристроями та рівнем підготовки, що особливо важливо для міських умов, де доступ до технологій може варіюватися [27]. Розробка прототипу включала автоматичну генерацію контролерів і представлень через Scaffold, що прискорило процес і дозволило зосередитися на адаптації логіки до специфіки футбольних ліг.

Реалізація функцій управління командами та подіями забезпечила організаторів інструментами для додавання гравців, створення змагань і етапів, а також оновлення результатів. Система відображення результатів, інтегрована в метод Details контролера CompetitionsController, дозволяє наочно представляти підсумки матчів із таблицями етапів і переможцями, що корисно для всіх користувачів. Тестування на основі тестових сценаріїв підтвердило стабільність роботи прототипу: усі функції – від реєстрації до перегляду результатів – виконуються коректно, без критичних помилок [6]. Аналіз результатів показав, що платформа відповідає базовим вимогам, визначеним у розділі 1, зокрема гнучкості управління, доступності інформації та безпеці даних.

Сильними сторонами розробленого прототипу є модульність, яка полегшує подальше розширення, швидкість обробки запитів завдяки асинхронності, а також адаптивність інтерфейсу, що відповідає сучасним стандартам веб-розробки. Водночас виявлено обмеження, такі як базовий рівень статистики, відсутність автоматичних тестів і інтеграцій із зовнішніми сервісами. Ці аспекти не впливають на основну функціональність, але їхнє вдосконалення може підвищити цінність платформи. Запропоновані напрями розвитку – додавання аналітики, автоматичного тестування, платіжних шлюзів, SMS-повідомлень і оптимізації продуктивності – створюють перспективу для масштабування системи в повноцінний продукт [27].

Практична значущість роботи полягає в тому, що розроблений прототип може бути використаний організаторами міських футбольних

ліг для автоматизації процесів управління турнірами, що скорочує час на адміністрування та підвищує прозорість змагань. Гравці отримують зручний доступ до розкладу та своїх даних через особистий кабінет, а вболівальники – можливість переглядати актуальні результати без складних процедур. Вихідний код, поданий у додатку роботи, слугує основою для реалізації цих можливостей у реальних умовах.

У ході роботи спроєктовано та реалізовано прототип платформи, який успішно виконує функції реєстрації, управління командами, подіями та відображення результатів. Досягнуто мети створення зручного й доступного інструменту для міських футбольних змагань, а поставлені завдання виконано повною мірою. Подальший розвиток платформи, заснований на запропонованих удосконаленнях, дозволить адаптувати її до ширших потреб і підвищити її конкурентоспроможність у сфері спортивних веб-додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойко В. С. Організація спортивних заходів: підручник / В. С. Бойко. – Київ: Видавництво "Спорт", 2018. – 320 с.
2. Войтович О. П. Інформаційні системи в менеджменті: монографія / О. П. Войтович. – Львів: ЛНУ ім. Івана Франка, 2019. – 245 с.
3. Гнатів М. І. Управління спортивними змаганнями в Україні / М. І. Гнатів // Український журнал медицини, біології та спорту. – 2020. – Т. 5, № 3. – С. 45–52.
4. Гринишин О. В. Основи менеджменту: навчальний посібник / О. В. Гринишин. – Харків: ХНУ ім. В. Н. Каразіна, 2021. – 180 с.
5. Державна програма розвитку фізичної культури і спорту на 2020–2025 роки / Міністерство молоді та спорту України. – Київ: ММС України, 2020. – 68 с.
6. Довгань Л. Є. Проектний менеджмент: підручник / Л. Є. Довгань. – Київ: КНЕУ, 2017. – 412 с.
7. Журавель Ю. В. Спортивний менеджмент в Україні: сучасний стан і перспективи / Ю. В. Журавель // Вісник НУФВСУ. – 2019. – № 2. – С. 12–19.
8. Звіт про діяльність Української асоціації футболу за 2022 рік / УАФ. – Київ: УАФ, 2023. – 54 с.
9. Іваненко О. М. Інноваційні технології в спорті / О. М. Іваненко // Науковий вісник НУФВСУ. – 2021. – № 4. – С. 33–40.
10. Коваленко С. І. Організація масових спортивних подій: навчальний посібник / С. І. Коваленко. – Дніпро: ДДУВС, 2020. – 256 с.
11. Козловський В. Ю. Інформаційні технології в управлінні: монографія / В. Ю. Козловський. – Київ: Видавництво "Техніка", 2016. – 300 с.

12. Коломієць Т. О. Управління спортивними організаціями / Т. О. Коломієць // Спорт для всіх. – 2018. – № 1. – С. 25–30.

13. Левицький І. П. Розвиток футбольної інфраструктури в містах України / І. П. Левицький // Фізична культура і спорт. – 2022. – № 3. – С. 15–22.

14. Литвиненко О. В. Сучасні підходи до менеджменту в спорті: монографія / О. В. Литвиненко. – Одеса: ОНУ ім. І. І. Мечникова, 2019. – 210 с.

15. Матеріали Міжнародної науково-практичної конференції "Молодь та олімпійський рух" / НУФВСУ. – Київ: НУФВСУ, 2021. – 150 с.

16. Мельник О. С. Основи організації спортивних турнірів / О. С. Мельник // Вісник КНУ ім. Тараса Шевченка. – 2020. – № 5. – С. 28–34.

17. Мірошніченко В. І. Інформаційні системи в спорті: навчальний посібник / В. І. Мірошніченко. – Харків: ХДАК, 2018. – 190 с.

18. Олексюк О. М. Технології управління проєктами / О. М. Олексюк. – Київ: Видавництво "Ліра-К", 2022. – 350 с.

19. Павлюк О. П. Футбол у міському середовищі: організаційні аспекти / О. П. Павлюк // Спортивний вісник Придніпров'я. – 2021. – № 2. – С. 10–17.

20. Петренко І. В. Системи управління базами даних: підручник / І. В. Петренко. – Львів: ЛДУФК, 2019. – 280 с.

21. Планування спортивних подій в Україні: звіт за 2021 рік / Міністерство молоді та спорту України. – Київ: ММС України, 2022. – 45 с.

22. Попов В. О. Організація спортивного менеджменту: навчальний посібник / В. О. Попов. – Київ: Видавництво "Освіта", 2020. – 230 с.

23. Регламент проведення футбольних змагань в Україні / УАФ. – Київ: УАФ, 2023. – 72 с.
24. Савчук О. Л. Інновації в спортивному менеджменті / О. Л. Савчук // Наука і спорт. – 2020. – № 3. – С. 18–25.
25. Сидоренко Н. В. Управління масовими заходами: підручник / Н. В. Сидоренко. – Київ: КНУКіМ, 2017. – 310 с.
26. Степаненко І. Г. Розвиток спорту в містах: сучасні виклики / І. Г. Степаненко // Фізичне виховання і спорт. – 2021. – № 1. – С. 5–12.
27. Шевчук В. М. Технології створення веб-платформ: монографія / В. М. Шевчук. – Київ: Видавництво "Політехніка", 2020. – 270 с.

ДОДАТКИ

```

using GameStore.Data;
using GameStore.Models;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;

namespace GameStore.Controllers
{
    public class AccountController : Controller
    {
        private GameStoreContext db;
        IHostingEnvironment Environment;
        public AccountController(GameStoreContext context, IHostingEnvironment
_environment)
        {
            db = context;
            Environment = _environment;
        }
        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Login(LoginModel model)
        {
            if (ModelState.IsValid)
            {
                User user = await db.Users.Include(u => u.Role).FirstOrDefaultAsync(u => u.Login
== model.Login && u.Password == model.Password);
                if (user != null)
                {
                    await Authenticate(user); // аутентификация

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

```

```

    }
    ModelState.AddModelError("", "Некорректные логин и(или) пароль");
}
return View(model);
}
[HttpGet]
public IActionResult Register()
{
    ViewBag.Roles = new SelectList(db.Roles, "ID", "Name");
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterModel model, IFormFile file)
{
    User user = await db.Users.FirstOrDefaultAsync(u => u.Login == model.Login);
    if (user == null)
    {
        string avatar = "";
        //TODO ADD DEFAULT VALUE FOR ROLE ID CLIENT --- NOTE REGISTER ONLY FOR
CLIENTS...
        if (file == null)
            avatar = "/img/men_avatar.png";

        User registerUser = new User {
            FirstName = model.FirstName,
            LastName = model.LastName,
            Avatar = avatar,
            Login = model.Login,
            Password = model.Password,
            RoleID = model.RoleID
        };

        // добавляем пользователя в бд
        db.Users.Add(registerUser);
        await db.SaveChangesAsync();

        if(file != null)
        {
            string wwwPath = this.Environment.WebRootPath;
            string contentPath = this.Environment.ContentRootPath;

            User user2 = db.Users.First(x => x.Login.Equals(registerUser.Login));

            string path = Path.Combine(this.Environment.WebRootPath, "Images/Users/" +
user2.ID);
            if (!Directory.Exists(path))

```

```

        {
            Directory.CreateDirectory(path);
        }

        string fileName = Path.GetFileName(file.FileName);
        using (FileStream stream = new FileStream(Path.Combine(path, fileName),
FileMode.Create))
        {
            file.CopyTo(stream);
        }

        user2.Avatar = "/Images/Users/" + user2.ID + "/" + fileName;

        db.Users.Update(user2);
        db.SaveChanges();
    }

    return RedirectToAction("Login", "Account");
}
else
{
    ModelState.AddModelError("", "Некорректные логин и(или) пароль");
    return View(model);
}
}

private async Task Authenticate(User user)
{
    // создаем один claim
    var claims = new List<Claim>
    {
        new Claim(ClaimsIdentity.DefaultNameClaimType, user.Login),
        new Claim(ClaimsIdentity.DefaultRoleClaimType, user.Role?.Name)
    };
    // создаем объект ClaimsIdentity
    ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie",
ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);
    // установка аутентификационных куки
    await
HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new
ClaimsPrincipal(id));
}

public async Task<ActionResult> Logout()
{
    await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToAction("Login", "Account");
}

```

```

    }
}
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using GameStore.Data;
using GameStore.Models;

```

```

namespace CompetitionWeb.Controllers
{
    public class StagesController : Controller
    {
        private readonly GameStoreContext _context;

        public StagesController(GameStoreContext context)
        {
            _context = context;
        }

        // GET: Stages
        public async Task<IActionResult> Index()
        {
            var gameStoreContext = _context.Stages.Include(s => s.Level);
            return View(await gameStoreContext.ToListAsync());
        }

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

            var stage = await _context.Stages
                .Include(s => s.Level)
                .FirstOrDefaultAsync(m => m.ID == id);
            if (stage == null)
            {
                return NotFound();
            }
        }
    }
}

```

```

        return View(stage);
    }

    // GET: Stages/Create
    public IActionResult Create(int? id)
    {
        ViewData["LevelID"] = new SelectList(_context.Levels, "ID", "Name");
        ViewData["CompetitionID"] = new SelectList(_context.Competitions, "ID", "Name",
id);
        ViewData["PlayerOneID"] = new SelectList(_context.Users, "ID", "FullName");
        ViewData["PlayerTwoID"] = new SelectList(_context.Users, "ID", "FullName");
        ViewData["PlayerWinnerID"] = new SelectList(_context.Users, "ID", "FullName");
        return View();
    }

    // POST: Stages/Create
    // To protect from overposting attacks, enable the specific properties you want to bind
to.
    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("LevelID,PlayerOneID,PlayerTwoID,PlayerWinnerID,CompetitionID")] Stage
stage)
    {
        if (ModelState.IsValid)
        {
            Stage stage1 = stage;
            _context.Add(stage);
            await _context.SaveChangesAsync();
            return Redirect("~/Competitions/Details/" + stage.CompetitionID);
        }
        ViewData["LevelID"] = new SelectList(_context.Levels, "ID", "ID", stage.LevelID);
        return View(stage);
    }

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

        var stage = await _context.Stages.FindAsync(id);
        if (stage == null)
        {
            return NotFound();
        }
    }

```

```

    }
    ViewData["LevelID"] = new SelectList(_context.Levels, "ID", "ID", stage.LevelID);
    return View(stage);
}

// POST: Stages/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind
to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("ID,LevelID,PlayerOneID,PlayerTwoID,PlayerWinnerID,CompetitionID")] Stage stage)
{
    if (id != stage.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(stage);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!StageExists(stage.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["LevelID"] = new SelectList(_context.Levels, "ID", "ID", stage.LevelID);
    return View(stage);
}

// GET: Stages/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {

```

```

        return NotFound();
    }

    var stage = await _context.Stages
        .Include(s => s.Level)
        .FirstOrDefaultAsync(m => m.ID == id);
    if (stage == null)
    {
        return NotFound();
    }

    return View(stage);
}

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

private bool StageExists(int id)
{
    return _context.Stages.Any(e => e.ID == id);
}
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using GameStore.Data;
using GameStore.Models;

namespace CompetitionWeb.Controllers
{
    public class CompetitionsController : Controller
    {
        private readonly GameStoreContext _context;
    }
}

```

```

public CompetitionsController(GameStoreContext context)
{
    _context = context;
}

// GET: Competitions
public async Task<ActionResult> Index()
{
    var gameStoreContext = _context.Competitions.Include(c => c.Category).Include(c
=> c.Status);
    return View(await gameStoreContext.ToListAsync());
}

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

    var competition = await _context.Competitions
        .Include(c => c.Category)
        .Include(c => c.Status)
        .Include(c => c.Stages)
        .FirstOrDefaultAsync(m => m.ID == id);

    competition.Stages = _context.Stages.Include(x=>x.Level).Where(x =>
x.CompetitionID == competition.ID).ToList();

    for(int i=0; i<competition.Stages.Count(); i++)
    {
        competition.Stages.ElementAt(i).PlayerOne = _context.Users.First(x => x.ID ==
competition.Stages.ElementAt(i).PlayerOneID);
        competition.Stages.ElementAt(i).PlayerTwo = _context.Users.First(x => x.ID ==
competition.Stages.ElementAt(i).PlayerTwoID);
        competition.Stages.ElementAt(i).PlayerWinner = _context.Users.First(x => x.ID ==
competition.Stages.ElementAt(i).PlayerWinnerID);
        competition.Stages.ElementAt(i).Competition = _context.Competitions.First(x =>
x.ID == competition.Stages.ElementAt(i).CompetitionID);
    }

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

```

```

        return View(competiton);
    }

    // GET: Competitons/Create
    public IActionResult Create()
    {
        ViewData["CategoryID"] = new SelectList(_context.Categories, "ID", "Name");
        ViewData["StatusID"] = new SelectList(_context.Statuses, "ID", "Name");
        return View();
    }

    // POST: Competitons/Create
    // To protect from overposting attacks, enable the specific properties you want to bind
to.
    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("ID,Name,Date,Description,CategoryID,StatusID")] Competiton competiton)
    {
        if (ModelState.IsValid)
        {
            _context.Add(competiton);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        ViewData["CategoryID"] = new SelectList(_context.Categories, "ID", "Name",
competiton.CategoryID);
        ViewData["StatusID"] = new SelectList(_context.Statuses, "ID", "Name",
competiton.StatusID);
        return View(competiton);
    }

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

        var competiton = await _context.Competitons.FindAsync(id);
        if (competiton == null)
        {
            return NotFound();
        }
        ViewData["CategoryID"] = new SelectList(_context.Categories, "ID", "Name",
competiton.CategoryID);

```

```

        ViewData["StatusID"] = new SelectList(_context.Statuses, "ID", "Name",
competiton.StatusID);
        return View(competiton);
    }

    // POST: Competitons/Edit/5
    // To protect from overposting attacks, enable the specific properties you want to bind
to.
    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("ID,Name,Date,Description,CategoryID,StatusID")] Competiton competiton)
    {
        if (id != competiton.ID)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(competiton);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!CompetitonExists(competiton.ID))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        ViewData["CategoryID"] = new SelectList(_context.Categories, "ID", "Name",
competiton.CategoryID);
        ViewData["StatusID"] = new SelectList(_context.Statuses, "ID", "Name",
competiton.StatusID);
        return View(competiton);
    }

    // GET: Competitons/Delete/5
    public async Task<IActionResult> Delete(int? id)

```

```

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

        var competiton = await _context.Competitons
            .Include(c => c.Category)
            .Include(c => c.Status)
            .FirstOrDefaultAsync(m => m.ID == id);
        if (competiton == null)
        {
            return NotFound();
        }

        return View(competiton);
    }

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

    private bool CompetitonExists(int id)
    {
        return _context.Competitons.Any(e => e.ID == id);
    }
}

```

```

using System.Collections;
using System.Collections.Generic;

```

```

namespace GameStore.Models
{
    public class Category
    {
        public int ID { get; set; }
        public string Name { get; set; }
        ICollection<Competiton> Competitons { get; set; }
    }
}

```

```

}

using System.Collections.Generic;

namespace GameStore.Models
{
    public class Competiton
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string Date { get; set; }
        public string Description { get; set; }
        public int CategoryID { get; set; }
        public Category Category { get; set; }
        public int StatusID { get; set; }
        public Status Status { get; set; }
        public IEnumerable<User> Players { get; set; }
        public IEnumerable<Stage> Stages { get; set; }
    }
}

using System.Collections;
using System.Collections.Generic;

namespace GameStore.Models
{
    public class Level
    {
        public int ID { get; set; }
        public string Name { get; set; }
        ICollection<Stage> Stages { get; set; }
    }
}

using System.ComponentModel.DataAnnotations;

namespace GameStore.Models
{
    public class LoginModel
    {
        [Required(ErrorMessage = "Не вказаний логін")]
        public string Login { get; set; }

        [Required(ErrorMessage = "Не вказаний пароль")]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

```

}

using System;
using System.ComponentModel.DataAnnotations;

namespace GameStore.Models
{
    public class RegisterModel
    {
        [Required(ErrorMessage = "Не вказане ім'я")]
        public string FirstName { get; set; }

        [Required(ErrorMessage = "Не вказане прізвище")]
        public string LastName { get; set; }

        [Required(ErrorMessage = "Не вказан логін")]
        public string Login { get; set; }

        [Required(ErrorMessage = "Не вказан пароль")]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Пароль уведено не вірно")]
        public string ConfirmPassword { get; set; }

        [Required(ErrorMessage = "Не вибрана роль")]
        public int RoleID { get; set; }

        [Required(ErrorMessage = "Не вибраний аватар")]
        public string Avatar { get; set; }
    }
}

using System.Collections;
using System.Collections.Generic;

namespace GameStore.Models
{
    public class Role
    {
        public int ID { get; set; }
        public string Name { get; set; }
        ICollection<User> Users { get; set; }
    }
}

using System.Collections.Generic;

```

```

namespace GameStore.Models
{
    public class Stage
    {
        public int ID { get; set; }
        public int LevelID { get; set; }
        public Level Level { get; set; }
        public int? PlayerOneID { get; set; }
        public User PlayerOne { get; set; }
        public int? PlayerTwoID { get; set; }
        public User PlayerTwo { get; set; }
        public int? PlayerWinnerID { get; set; }
        public User PlayerWinner { get; set; }
        public int CompetitionID { get; set; }
        public Competiton Competiton { get; set; }

    }
}
using System.Collections;
using System.Collections.Generic;

```

```

namespace GameStore.Models
{
    public class Status
    {
        public int ID { get; set; }
        public string Name { get; set; }
        ICollection<Competiton> Competitons { get; set; }
    }
}
using System.Collections.Generic;

```

```

namespace GameStore.Models
{
    public class User
    {
        public int ID { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Avatar { get; set; }
        public int RoleID { get; set; }
        public Role Role { get; set; }
        public string FullName { get => FirstName + " " + LastName; }
        public ICollection<Competiton> Competitons { get; set; }
        public ICollection<Stage> Stages { get; set; }
    }
}

```

}
}