Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему : «Розробка додатку для бронювання спортивних майданчиків»

Виконав: студент групи ІПЗ20-1
Спеціальність 121 «Інженерія програмного забезпечення»
Макух Олександр Вікторович
(прізвище та ініціали)
Керівник д.т.н., професор кафедри комп'ютерних наук та інженерії програмного забезпечення Яковенко В.О
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
в.о. завідувача кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доцент кафедри кібербезпеки та інформаційних технологій Прокопович - Ткаченко Д.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Макух О.В. Розробка додатку для бронювання спортивних майданчиків

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справи та фінансів, Дніпро, 2024.

Кваліфікаційна робота присвячена розробці додатку для бронювання спортивних майданчиків. У роботі розглянуті методи та технічні засоби, необхідні для реалізації системи, сформульовані вимоги до програмного забезпечення. У рамках дослідження було проведено аналіз існуючих рішень, визначено їхні переваги та недоліки, а також обґрунтовано вибір технологічного стеку для створення власного додатку.

В процесі роботи було проведено проектування та розробку системи, включаючи детальний опис архітектури, вибір патернів проектування та реалізацію основних функціональних модулів, проведено тестування системи. Результати дослідження мають практичне значення, оскільки вони сприяють автоматизації процесів бронювання, підвищуючи ефективність і зручність користування для користувачів.

Розроблена система має серверну та клієнтську частини. Програмний продукт являє собою додаток, створений на мові програмування C# та з використанням системи керування базами даних Microsoft SQL Server. Система забезпечує високу продуктивність, безпеку та можливість масштабування завдяки використанню сучасних технологій та принципів дизайну інтерфейсу користувача.

Ключові слова: розробка, C#, .NET, Microsoft SQL Server, Windows Forms, бронювання спортивних майданчиків.

ABSTRACT

Makukh O.V. Development of an application for booking sports grounds

Qualification work for a bachelor's degree in speciality 121 'Software Engineering.' - University of Customs and Finance, Dnipro, 2024.

The qualification work is devoted to the development of an application for booking sports grounds. The work considers the methods and technical means necessary for the implementation of the system, formulates software requirements. The study analysed existing solutions, identified their advantages and disadvantages, and justified the choice of a technology stack for creating an in-house application.

In the course of the study, the system was designed and developed, including a detailed description of the architecture, selection of design patterns and implementation of the main functional modules, and testing of the system. The results of the study are of practical importance, as they contribute to the automation of booking processes, increasing efficiency and usability for users.

The developed system has server and client parts. The software product is an application created in the C# programming language and using the Microsoft SQL Server database management system. The system provides high performance, security and scalability through the use of modern technologies and user interface design principles.

Keywords: development, C#, .NET, Microsoft SQL Server, Windows Forms, sports grounds booking.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних.

ПЗ – Програмне забезпечення.

СУБД – Система управління базами даних.

C# – Об’єктно-орієнтована мова програмування.

CLR– Common Language Runtime.

MS – Microsoft.

SQL – Structured query language.

WF – Windows Forms.

ВСТУП

Актуальність дослідження: У сучасному світі, де активний спосіб життя та заняття спортом стають все більш популярними, виникає потреба у зручному та ефективному інструменті для бронювання спортивних майданчиків. Традиційні методи бронювання через телефон або особисто стають менш зручними та ефективними в умовах швидкого темпу життя. Розробка додатку для бронювання спортивних майданчиків дозволяє автоматизувати цей процес, зробити його більш доступним та зручним для користувачів. Актуальність дослідження підкріплюється зростаючим попитом на цифрові рішення для управління часом та ресурсами, а також необхідністю оптимізації роботи спортивних комплексів.

Цифровізація процесів бронювання не лише спрощує життя користувачам, а й допомагає спортивним закладам підвищити ефективність роботи та краще задовольняти потреби клієнтів. Додаток дозволить уникнути конфліктів при бронюванні, забезпечити прозорий графік використання майданчиків та полегшити адміністративну роботу персоналу. Крім того, автоматизована система бронювання відкриває нові можливості для аналізу статистичних даних, виявлення пікових періодів попиту та оптимізації ресурсів відповідно до поведінки користувачів.

Ще однією перевагою розробки додатку є екологічний аспект. Цифровий процес бронювання зменшує потребу у паперових документах, друкованих розкладах та інших фізичних носіях інформації, тим самим зменшуючи вплив на навколишнє середовище.

Таким чином, створення зручного та функціонального додатку для бронювання спортивних майданчиків є актуальним завданням, що відповідає сучасним тенденціям цифровізації, оптимізації ресурсів та екологічності. Воно забезпечить користувачам зручний та ефективний спосіб планування своїх спортивних заходів, а спортивним закладам - більш організовану роботу та кращий клієнтський досвід.

Мета даної кваліфікаційної роботи є розробка функціонального та зручного у використанні додатку для бронювання спортивних майданчиків, який забезпечить ефективне управління бронюваннями, зручний інтерфейс для користувачів та можливість масштабування системи для різних видів спортивних майданчиків.

Методи дослідження: метод теорії інформації, обробка та аналіз інформації, методи проектування та розробки програмного забезпечення, методи, проектування баз даних.

У відповідності до поставленої мети в кваліфікаційній роботі ставились та вирішувались наступні завдання дослідження:

1. Аналіз існуючих рішень у сфері бронювання спортивних майданчиків.
2. Проектування архітектури.
3. Розробка додатку з використанням мови програмування C#.
4. Провести тестування системи.

Об'єкт дослідження: процес автоматизації бронювання спортивних майданчиків та управління цими бронюваннями з використанням програмного забезпечення.

Предмет дослідження: методи та засоби розробки програмного забезпечення для бронювання спортивних майданчиків, включаючи архітектурні рішення, моделі даних та інтерфейс користувача.

Практичне значення одержаних результатів - автоматизувати процес бронювання, зменшити адміністративні витрати та покращити обслуговування клієнтів. Користувачі отримують зручний інструмент для швидкого бронювання, управління та перегляду наявних спортивних майданчиків, що сприяє збільшенню їх задоволеності. Спортивні комплекси можуть ефективніше управляти своїми ресурсами, підвищувати рівень завантаженості майданчиків та отримувати більш точну аналітику щодо використання своїх послуг.

Структура роботи:

Розділ 1 Аналіз предметної області. У даному розділі виконано та проаналізовано публікації стосовно теми бронювання спортивних майданчиків. Обґрунтовано актуальність теми та доцільності розробки програмного продукту

Розділ 2 Аналіз джерел та засобів реалізації. У даному розділі буде проаналізовано та обрано технології та засоби для реалізації програмного продукту. Обрано середовище розробки, мову програмування, систему управління базою даних, технології, систему контролю версій.

Розділ 3 Розробка та тестування додатку для бронювання спортивних майданчиків. В даному розділі буде спроектовано архітектуру, проектування та реалізацію бази даних, розроблено інтерфейс, серверну частину та тестування готового програмного продукту.

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 18 найменувань, 1 додатку. Обсяг роботи 77 сторінок, 28 рисунок.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування актуальності теми

Зростання популярності спорту та фізичної активності є фактором, який актуальний для розробки та впровадження інноваційних технологій для сучасних спортивних майданчиків. Сучасний світ стає все більш свідомим про важливість здорового способу життя, що призводить до зростання попиту на заняття іншими видами спорту. Підвищена обізнаність про переваги фізичної активності та здорового способу життя у суспільстві, профілактика хронічних захворювань та покращення фізичного і психічного здоров'я є головними мотиваторами для регулярних занять спортом та активним способом життя. Урбанізація та щільна забудова міських зон створюють потреби в громадських спортивних майданчиках та відкритих зонах для фізичної активності в межах міста. Популяризація здорового способу життя в корпоративній культурі та серед студентства також стимулює попит на сучасні спортивні споруди в офісах, на території університетів та навчальних закладів. Розвиток спортивної інфраструктури має стратегічне значення для сприяння масовому спорту та підвищення рівня фізичної підготовки населення.

Ефективне управління ресурсами є важливим чинником для успішного функціонування спортивних майданчиків. Власники цих об'єктів часто стикаються з низкою викликів, пов'язаних з оптимізацією використання наявних ресурсів, таких як часові обмеження, максимальне завантаження майданчиків та ефективний розподіл робочого часу персоналу.

Часові обмеження є одним з ключових факторів, які необхідно враховувати під час управління спортивними майданчиками. Зазвичай, ці об'єкти мають чітко визначені години роботи, протягом яких вони доступні для відвідувачів. Ефективне планування та розподіл часу між різними видами діяльності, такими як тренування, змагання чи оренда для приватних заходів, є важливим для оптимального використання наявних ресурсів.

Крім того, максимальне завантаження майданчиків також є важливим питанням для власників. Надмірне навантаження може призвести до перевтоми персоналу, незадоволення клієнтів та прискореного зносу обладнання. З іншого боку, недостатнє завантаження може негативно вплинути на прибутковість об'єкта. Тому необхідно ретельно планувати та відстежувати рівень використання майданчиків, щоб забезпечити їх оптимальне завантаження.

Ефективний розподіл робочого часу персоналу також є ключовим фактором для успішного управління спортивними майданчиками. Власники повинні забезпечити належну кількість персоналу для обслуговування клієнтів, проведення тренувань та підтримання чистоти й порядку на об'єкті. Неефективний розподіл робочого часу може призвести до перевантаження або недостатнього завантаження працівників, що може негативно вплинути на якість обслуговування та задоволеність клієнтів.

Розробка додатку для бронювання може стати ефективним рішенням для вирішення цих проблем. Такий додаток дозволить клієнтам легко бронювати час на майданчиках, а власникам – ефективно керувати графіком використання ресурсів. Додаток може надавати інформацію про доступність майданчиків у реальному часі, дозволяючи клієнтам швидко знаходити вільні слоти та робити бронювання. Для власників це означає можливість відстежувати попит, оптимізувати завантаження майданчиків та ефективно розподіляти ресурси персоналу.

Крім того, додаток для бронювання може забезпечити зручний спосіб оплати послуг, знижуючи навантаження на персонал та підвищуючи ефективність фінансових операцій. Він також може надавати аналітичні дані про використання майданчиків, що допоможе власникам приймати більш обґрунтовані рішення щодо планування та розвитку свого бізнесу.

Зручність та доступність інструментів бронювання є ключовими факторами, які визначають вибір користувачів при пошуку майданчиків для

спортивних занять. У сучасному швидкоплинному світі люди цінують свій час і шукають максимально зручні рішення для задоволення своїх потреб.

Традиційні методи бронювання, такі як телефонні дзвінки або особистий візит на об'єкт, можуть бути незручними та витрачати багато часу. Користувачі часто змушені чекати на лінії або приїжджати на місце, щоб дізнатися про доступність майданчиків та зробити бронювання. Це може призвести до розчарування та незадоволеності, особливо для тих, хто веде активний спосіб життя та має обмежені вільні години.

Розробка додатку для бронювання спортивних майданчиків може стати ефективним рішенням цієї проблеми. Такий додаток забезпечить користувачам швидкий та зручний спосіб пошуку та бронювання майданчиків у будь-який час та в будь-якому місці. Користувачі зможуть переглядати доступні слоти, порівнювати різні варіанти та робити бронювання, не витрачаючи багато часу та зусиль.

Додаток для бронювання також може надавати додаткову корисну інформацію про об'єкти, такі як опис майданчиків, доступне обладнання, розклад занять та відгуки інших користувачів. Це дозволить користувачам приймати більш обґрунтовані рішення при виборі місця для занять.

Задоволення потреби користувачів у зручних інструментах бронювання може допомогти спортивним майданчикам залучити більше клієнтів та підвищити їх лояльність. Користувачі, які отримають позитивний досвід бронювання через додаток, з більшою ймовірністю будуть повертатися на ці майданчики та рекомендувати їх своїм друзям і родичам.

Таким чином, розробка додатку для бронювання спортивних майданчиків може стати ефективним рішенням для задоволення потреби користувачів у зручності та доступності інструментів бронювання, що, в свою чергу, дозволить власникам майданчиків залучити більше клієнтів та підвищити їх лояльність.

Технологічні інновації справді трансформують галузь спорту, відкриваючи нові можливості для покращення користувацького досвіду та

підвищення ефективності управління об'єктами. Розробка інноваційного додатку для бронювання спортивних майданчиків може стати важливим кроком у цифровізації цієї галузі.

Технології також відкривають нові можливості для аналітики та прийняття рішень на основі даних. Додаток для бронювання може збирати та аналізувати дані про поведінку користувачів, їхні переваги та моделі використання майданчиків. Ця інформація може бути використана власниками для вдосконалення своїх послуг, розробки ефективних маркетингових стратегій та визначення найбільш перспективних напрямків розвитку бізнесу.

1.2 Аналіз існуючих аналогів програмного продукту. Обґрунтування доцільності розробки програмного продукту

Аналіз існуючих рішень для бронювання спортивних майданчиків є надзвичайно важливим етапом у розробці власного додатку. Ретельний аналіз конкурентів дозволяє отримати цілісне уявлення про поточний стан ринку бронювання спортивних майданчиків, включаючи провідних гравців, їхні пропозиції, стратегії та позиціонування. Це допомагає визначити можливості для диференціації та знайти незадоволені потреби користувачів. Крім того, аналіз існуючих рішень дозволяє ідентифікувати сильні сторони конкурентів, такі як унікальні функції, привабливий дизайн або ефективні маркетингові стратегії, а також виявити їхні слабкі сторони та недоліки, які можна використати для створення конкурентної переваги власного додатку.

Вивчаючи успішні додатки для бронювання спортивних майданчиків, можна виявити кращі практики та стандарти, які необхідно імплементувати у власному продукті. Це може стосуватися інтерфейсу користувача, функціональності, інтеграцій або інших аспектів, які сприяють задоволенню потреб користувачів. Аналіз моделей ціноутворення та монетизації існуючих додатків допоможе визначити оптимальну цінову стратегію для власного

продукту. Дослідження партнерств та інтеграцій між додатками для бронювання та іншими суб'єктами галузі може відкрити нові можливості для співпраці та розширення функціональності власного додатку. Аналіз технологічних платформ та рішень, які використовуються конкурентами, допоможе зробити обґрунтований вибір для власного додатку, враховуючи масштабованість, гнучкість та можливості інтеграції.

Глибоке розуміння існуючих рішень дозволить визначити унікальні переваги та інноваційні функції, які допоможуть власному додатку вирізнятися на ринку та привернути увагу користувачів. Таким чином, ретельний аналіз існуючих рішень для бронювання спортивних майданчиків є критично важливим для розробки конкурентоспроможного та успішного додатку, який задовольнить потреби користувачів та забезпечить стійку конкурентну перевагу на ринку.

Для аналіз було обрано чотири додатки за схожою тематикою для бронювання спортивних майданчиків:

- CeleBreak
- Skedda
- Palms
- Playseek

CeleBreak є популярним додатком для бронювання футбольних майданчиків, який спрямований на задоволення потреб аматорських футбольних команд, ентузіастів та любителів цього виду спорту. Основною метою додатку є спрощення процесу організації футбольних матчів та забезпечення зручного доступу до спортивних майданчиків [1].

Додаток CeleBreak є яскравим прикладом того, як технології можуть змінити спосіб, у який люди займаються спортом, зокрема футболом. CeleBreak має інтуїтивно зрозумілий інтерфейс, який забезпечує швидке та безпроблемне бронювання обраних майданчиків. Користувачі можуть вказати бажані дати, час і тривалість заняття, а також здійснити оплату онлайн за допомогою інтегрованих платіжних систем.

Додаток дозволяє створювати або приєднуватися до існуючих футбольних груп, що полегшує організацію матчів між командами та сприяє соціальній взаємодії серед гравців. Користувачі можуть залишати відгуки та оцінки про майданчики, що забезпечує прозорість та допомагає іншим гравцям робити обґрунтований вибір. CeleBreak надсилає нагадування про заплановані матчі та бронювання, допомагаючи користувачам тримати свій розклад під контролем. Додаток також може збирати та відображати статистику ігор, рейтинги команд та гравців, що підвищує змагальний дух та мотивацію. CeleBreak став популярним серед футбольної спільноти завдяки своїй зручності, доступності та можливості організовувати футбольні заходи без зайвих зусиль. Він заощаджує час та полегшує координацію, що дозволяє гравцям зосередитися на самій грі та насолоджуватися своїм улюбленим видом спорту.

Skedda є потужним інструментом для управління бронюванням різноманітних приміщень та майданчиків, який користується популярністю серед організацій та підприємств, що надають приміщення для занять спортом, конференцій, студій та інших заходів. Додаток забезпечує зручний та гнучкий спосіб організації розкладу та управління ресурсами, що робить його привабливим для широкого спектру користувачів [2].

Однією з основних переваг Skedda є його висока гнучкість налаштувань. Користувачі можуть налаштовувати систему відповідно до своїх потреб, включаючи різні типи майданчиків, часові слоти, ціни та правила бронювання. Це дозволяє використовувати Skedda для управління різноманітними ресурсами – від спортивних майданчиків до конференц-залів та студій.

Skedda також пропонує інтеграцію з різними календарними сервісами, що дозволяє користувачам синхронізувати розклади з особистими або корпоративними календарями. Це забезпечує зручність планування та знижує ризик конфліктів у розкладі. Додаток підтримує інтеграцію з популярними календарями, такими як Google Calendar та Microsoft Outlook.

Інтерфейс Skedda відзначається зручністю та інтуїтивною зрозумілістю. Користувачі можуть легко переглядати доступні слоти, здійснювати бронювання та керувати своїми замовленнями. Візуальне представлення розкладу допомагає швидко орієнтуватися в доступності ресурсів та здійснювати бронювання без зайвих зусиль.

Однією з ключових функцій Skedda є можливість деталізованого управління доступністю та правилами бронювання. Адміністратори можуть встановлювати різні правила для різних типів користувачів, обмежувати доступність певних слотів та встановлювати спеціальні тарифи. Це дозволяє оптимізувати використання ресурсів та задовольнити потреби різних груп клієнтів.

Проте, складність налаштувань може бути викликом для деяких користувачів. Незважаючи на зручний інтерфейс, для налаштування системи відповідно до конкретних потреб може знадобитися певний час та технічні знання. Деякі користувачі можуть вважати це недоліком, особливо якщо вони не мають достатньої технічної підготовки.

Palms - спеціалізований додаток для бронювання спортивних майданчиків. Він забезпечує ефективну автоматизацію процесів для користувачів і власників майданчиків. Додаток орієнтований на створення зручного та інтуїтивного інтерфейсу для полегшення керування бронюваннями, розкладами та фінансами [3].

Одна з головних переваг Palms - створення персоналізованих розкладів для різних груп користувачів. Це дозволяє власникам оптимізувати використання ресурсів, встановлювати пріоритети для певних клієнтів чи груп, надавати спеціальні умови постійним клієнтам. Така функціональність підвищує гнучкість управління та задовольняє потреби різних категорій користувачів.

Palms інтегрується з CRM-системами, що дозволяє ефективно керувати клієнтською базою. Власники можуть зберігати історію бронювань, контакти клієнтів, аналізувати поведінку користувачів для покращення сервісу та

маркетингових стратегій. Це сприяє підвищенню рівня обслуговування та лояльності клієнтів.

Додаток підтримує онлайн-оплати, спрощуючи фінансові операції. Користувачі можуть швидко та безпечно оплачувати бронювання через додаток, підвищуючи зручність використання. Інтеграція з різними платіжними системами забезпечує широкий вибір методів оплати.

Інтерфейс Palms розроблений з урахуванням потреб користувачів, забезпечуючи інтуїтивність та простоту використання. Користувачі легко переглядають доступні слоти, бронюють їх і керують замовленнями. Візуалізація розкладу допомагає швидко орієнтуватися в доступності ресурсів та уникати конфліктів.

Palms - потужний та ефективний інструмент для керування бронюванням спортивних майданчиків. Його персоналізовані розклади, інтеграція з CRM-системами, підтримка онлайн-оплат та зручний інтерфейс роблять його привабливим для власників і користувачів. Однак висока вартість підписки може стати обмежувальним фактором для деяких організацій.

Playseek - універсальний додаток для бронювання спортивних майданчиків різних видів. Забезпечує зручність для аматорів і професіоналів. Відрізняється мультифункціональністю, підтримуючи бронювання для тенісу, баскетболу, волейболу, футболу, бадмінтону та інших видів спорту. Має зручний інтуїтивний інтерфейс для перегляду слотів, бронювання та керування замовленнями. Інтегрований з платіжними системами для онлайн-оплати. Дозволяє залишати відгуки та оцінки майданчиків для прозорості. Пропонує функції соціальної взаємодії для організації групових занять та заходів [4].

Таблиця 1.1- Таблиця переваг та недоліків додатків для бронювання спортивних майданчиків

Додаток	Переваги	Недоліки
CeleBreak	- Простий і зручний інтерфейс - Інтеграція з платіжними системами для онлайн-оплат - Соціальні функції для взаємодії між гравцями	- Обмеженість підтримки інших видів спорту, окрім футболу - Обмежена географічна доступність
Skedda	- Висока гнучкість налаштувань - Інтеграція з календарними сервісами - Деталізоване управління розкладом та доступністю ресурсів	- Складність налаштувань для нових користувачів -Потреба в технічних знаннях для налаштування
Palms	- Персоналізовані розклади для різних груп користувачів - Інтеграція з CRM-системами - Підтримка онлайн-оплат - Висока безпека даних	- Висока вартість підписки - Обмежена доступність для невеликих клубів та майданчиків
Playseek	- Мультифункціональність (підтримка багатьох видів спорту) - Інтуїтивно зрозумілий інтерфейс - Підтримка онлайн-оплат - Відгуки та оцінки майданчиків - Соціальні функції для організації групових занять	- Складність налаштування для нових користувачів - Потреба в додатковій увазі для підвищення зручності використання

1.3. Висновки до першого розділу. Постановка задач дослідження.

На основі проведеного аналізу предметної області та існуючих рішень для бронювання спортивних майданчиків можна зробити наступні висновки. Розробка додатку для бронювання спортивних майданчиків є актуальною через зростання популярності спорту та фізичної активності. Сучасні користувачі потребують зручних і швидких інструментів для організації свого часу та занять, а власники спортивних майданчиків – ефективного управління ресурсами.

Ринок додатків для бронювання вже має декілька популярних рішень, таких як CeleBreak, Skedda, Palms та Playseek. Кожне з цих рішень має свої переваги та недоліки, що дозволяє визначити ключові аспекти для подальшого покращення. Наприклад, CeleBreak спрощує організацію футбольних матчів, Skedda пропонує високу гнучкість налаштувань, Palms інтегрується з CRM-системами для кращого управління клієнтами, а Playseek підтримує різні види спорту.

На основі аналізу існуючих рішень, можна виділити основні напрями для покращення майбутнього додатку: забезпечення інтуїтивно зрозумілого інтерфейсу, інтеграція з календарями та CRM-системами, надання аналітичних даних про використання майданчиків. Це дозволить створити конкурентоспроможний продукт, який задовольнить потреби користувачів та власників спортивних майданчиків.

Таким чином, розробка додатку для бронювання спортивних майданчиків має важливе значення для сучасного ринку, оскільки дозволяє оптимізувати процеси управління та задовольнити потреби користувачів у зручних і ефективних інструментах. Впровадження такого додатку сприятиме покращенню користувацького досвіду, підвищенню лояльності клієнтів та ефективному управлінню спортивними ресурсами.

Кінцевим результатом дослідницької роботи стане створення ефективного програмного рішення для бронювання спортивних майданчиків, яке відповідатиме сучасним вимогам і тенденціям у цій галузі. Це рішення сприятиме покращенню користувацького досвіду, підвищенню лояльності клієнтів та ефективному управлінню спортивними ресурсами.

РОЗДІЛ 2. АНАЛІЗ ДЖЕРЕЛ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір технологій та засобів розробки для додатку

Вибір технологій та засобів розробки є фундаментальним етапом у створенні будь-якого додатку, особливо коли йдеться про розробку складних інформаційних систем пов'язаних з бронювання. Цей процес вимагає ретельного аналізу та врахування численних факторів для забезпечення ефективної, надійної та масштабованої реалізації проекту.

Розробка починається з вивчення наявних рішень на ринку, їхніх переваг та недоліків. Цей аналіз допомагає зрозуміти поточний стан технологій, тенденції та найкращі практики у відповідній галузі. Після цього необхідно визначити відповідні середовища розробки, мови програмування та системи управління базами даних.

Вибір технологічного стеку залежить від низки факторів, серед яких: вимоги проекту, компетенції та досвід розробників, наявність активної спільноти підтримки, рівень документації та ресурсів для навчання. Важливо також врахувати масштабованість, безпеку, продуктивність та інтеграційні можливості обраних технологій.

Вибір архітектури додатку є критично важливим етапом під час розробки. Від правильного архітектурного рішення залежать не лише якість та функціональність програми, але й її стабільність, можливість масштабування, зручність підтримки та подальшого розвитку. Ретельно продумана архітектура дозволяє легко додавати нові функції та змінювати існуючі без необхідності значного переписування коду, що особливо актуально для додатків, які повинні розширюватися або модифікуватися з плином часу. Крім того, вдала організація коду та розподіл завдань між різними компонентами дозволяють оптимально використовувати системні ресурси, забезпечуючи високу продуктивність додатку.

Не менш важливим є те, що гарна архітектура сприяє створенню стабільного та надійного додатку, здатного працювати без збоїв і помилок. Використання перевірених паттернів проектування може значно покращити якість коду та його стійкість до помилок. Ще одною перевагою добре спроектованої архітектури є полегшення процесів підтримки та тестування програми, що особливо цінується у довгострокових проектах, де підтримка може тривати роками. Завдяки чіткій структурі стає легше виявляти та виправляти помилки, а також здійснювати всебічне тестування додатку.

Дизайн додатку відіграє надзвичайно важливу роль, адже він безпосередньо впливає на зручність використання, естетичну привабливість та загальну ефективність взаємодії користувача з програмою. Це правило стосується й додатків, де продуманий дизайн є запорукою комфортної роботи з програмним забезпеченням. Орієнтований на користувача дизайн забезпечує інтуїтивно зрозумілий інтерфейс, який легко освоюється і з яким приємно працювати. Він включає зручне розташування елементів керування, чіткі інструкції та логічну навігацію, що підвищує продуктивність користувачів і знижує ймовірність помилок.

Естетично привабливий та функціональний інтерфейс робить додаток більш привабливим для користувачів, тому важливо використовувати сучасні та узгоджені елементи управління, щоб забезпечити професійний вигляд програми. Консистентний дизайн, що використовує уніфіковані шаблони та стилі у всьому додатку, сприяє легшому навчанню користувачів та знижує когнітивне навантаження.

Крім того, оптимізований дизайн може покращити загальну продуктивність додатку завдяки мінімізації зайвих елементів, оптимізації графіки та забезпеченню швидкого відгуку інтерфейсу. Дизайн також відіграє важливу роль у передачі бренду, адже узгоджені кольори, логотипи та стилі допомагають створити впізнаваний образ і підвищити довіру користувачів до продукту. Отже, ретельно продуманий дизайн є необхідною умовою для забезпечення не лише функціональності, але й привабливості та зручності

використання, що в кінцевому підсумку сприяє успіху програмного продукту на ринку.

Вибір мови програмування та бази даних є важливим етапом у процесі розробки додатків, адже ці рішення мають далекосяжні наслідки для продуктивності, масштабованості, підтримки та майбутнього розвитку програмного продукту. Для розробки програмного забезпечення вкрай важливо обрати найбільш оптимальні інструменти, які забезпечать ефективну реалізацію запланованого функціоналу та повністю відповідатимуть вимогам проекту. При цьому слід враховувати не лише поточні потреби, але й можливості для подальшого розширення та модифікації програми в майбутньому.

Вибір мови програмування впливає на швидкість розробки, кросплатформеність, підтримку спільноти розробників та наявність бібліотек і фреймворків. Кожна з них має свої переваги та недоліки, які необхідно ретельно проаналізувати в контексті конкретного проекту.

База даних, у свою чергу, визначає спосіб зберігання, обробки та захисту даних додатку. Вибір полягає між реляційними базами даних, такими як SQL Server чи MySQL, або ж використати нереляційні NoSQL рішення, як наприклад MongoDB чи Cassandra.

Крім основних технологій, під час планування проекту також варто розглянути допоміжні бібліотеки, фреймворки та інструменти для тестування, збірки, розгортання та моніторингу додатку. Ретельно продуманий вибір всього технологічного стеку забезпечить створення ефективного, масштабованого та легкого у підтримці додатку, здатного відповідати вимогам сьогодення та майбутнього.

2.2. Аналіз програмних засобів

Аналіз доступних програмних засобів є невід'ємною частиною процесу розробки додатків, адже передбачає ретельне вивчення та оцінку різноманітних фреймворків, бібліотек та інструментів, які можуть бути використані для створення проекту. Основною метою такого аналізу є ретельний відбір найбільш ефективних, сучасних та зручних у використанні інструментів, які ідеально відповідатимуть специфічним вимогам проекту та забезпечать його успішну реалізацію.

Крім безпосередніх технологій розробки, важливо також оцінити інструменти для тестування, збірки, розгортання та моніторингу додатку, адже вони дозволяють забезпечити належну якість програмного продукту та спростити його супровід. Після ретельного аналізу та співставлення всіх переваг і недоліків різних програмних засобів, можна зробити обґрунтований вибір оптимальної комбінації технологій, яка дозволить створити високоефективний, масштабований та легкий в підтримці додаток, що повністю відповідатиме поставленим вимогам.

Для розробки додатку було обрано такі інструменти: платформу WinForms з використанням мови програмування C#, для управління даними в додатку було обрано реляційну базу даних Microsoft SQL Server, для інтерфейсу користувача було обрано концепцію MVP (Model-View-Presenter) та системи контролю версій Git.

2.3 Середовище розробки

Середовище розробки відіграє ключову роль у процесі створення додатку, забезпечуючи зручність та ефективність написання, тестування та налагодження коду. Існує багато середовищ розробки, кожне з яких має свої особливості та переваги.

Розробка додатку для бронювання спортивних майданчиків потребує використання надійного та функціонального середовища розробки. У цьому випадку було обрано Microsoft Visual Studio, що є потужним та широко використовуваним інструментом для створення .NET додатків, включаючи WinForms. Вибір Visual Studio обґрунтований низкою переваг, які забезпечують ефективність і зручність у процесі розробки (див. рис. 2.1) [5].

Visual Studio тісно інтегрований з .NET Framework і надає повну підтримку для розробки WinForms додатків. Це включає інтуїтивно зрозумілий дизайнер форм та потужний редактор коду, який спрощує процес створення інтерфейсів користувача та написання програмного коду. Крім того, середовище пропонує розширюваність завдяки підтримці великої кількості плагінів та розширень, що дозволяє адаптувати його функціональність під конкретні потреби проекту.

Одною з ключових переваг Visual Studio є наявність потужних інструментів для налагодження коду. Розробники можуть використовувати покрокове виконання коду, встановлювати точки зупинки та аналізувати змінні, що значно полегшує процес виявлення та виправлення помилок. Середовище також забезпечує гнучку інтеграцію з популярними системами контролю версій, такими як Git, що сприяє ефективному управлінню кодом та співпраці між розробниками.

Рисунок 2.1 – Логотип Visual Studio

Важливою перевагою Visual Studio є його комфортний та інтуїтивний інтерфейс, який підвищує продуктивність розробників. Навіть для тривалих

сесій роботи інтерфейс залишається зручним та не створює зайвого навантаження на користувача (див. рис. 2.2).

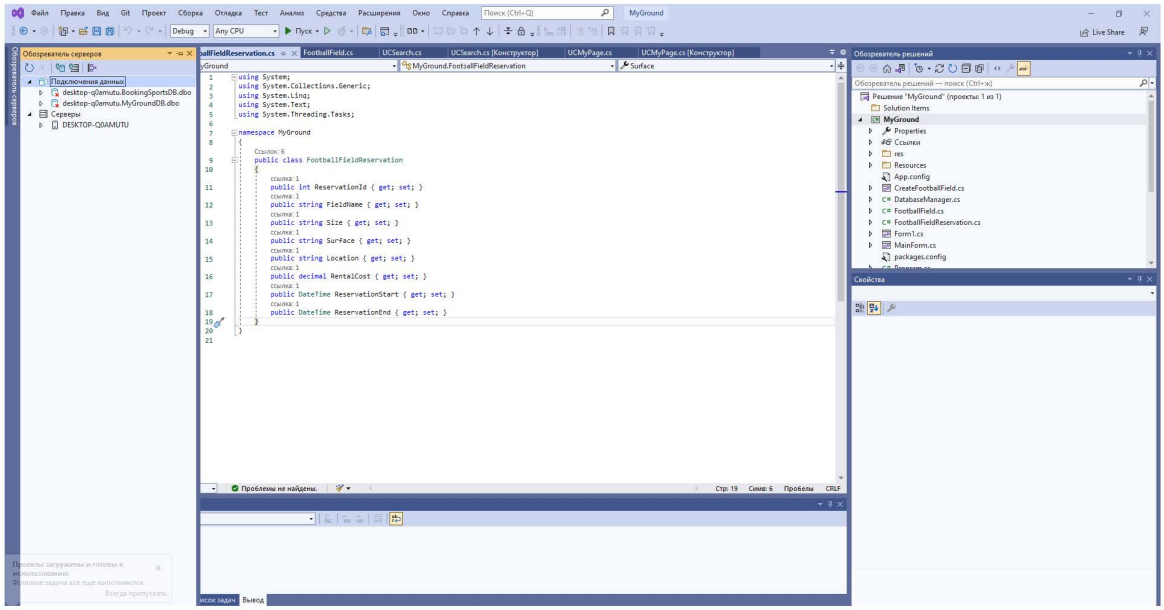

Рисунок 2.2 – Стартова сторінка MS Visual Studio 2022

Для порівняння розглянемо інші популярні середовища розробки:

– JetBrains Rider;

– Eclipse;

– Visual Studio Code.

JetBrains Rider - середовище розробки, яке пропонує низку переваг для розробників .NET. Окрім повної підтримки C#, VB.NET, ASP.NET, ASP.NET Core, Xamarin, Unity та інших технологій .NET, Rider має багато інших корисних функцій, які підвищують продуктивність та зручність роботи (див. рис. 2.3) [6].

Одним із ключових переваг Rider є його інтелектуальний редактор коду, який забезпечує розширені можливості автозавершення, рефакторингу, навігації та пошуку. Він також має вбудовані інструменти для аналізу коду та виявлення потенційних проблем, що допомагає підвищити якість коду [6].

Rider тісно інтегрований з інструментами JetBrains, такими як ReSharper, що додає додаткові можливості для рефакторингу, аналізу коду та продуктивності. Ця інтеграція дозволяє розробникам отримати максимальну віддачу від екосистеми JetBrains .

Середовище розробки Rider також має потужні інструменти для налагодження, які дозволяють розробникам покроково виконувати код, встановлювати точки зупинки, аналізувати змінні та стек викликів. Це допомагає ефективно знаходити та виправляти помилки в коді.

Окрім того, Rider пропонує підтримку популярних систем контролю версій, таких як Git, Mercurial та Subversion, що полегшує співпрацю в команді розробників. Він також має вбудовані інструменти для роботи з базами даних, що може бути корисним для розробників, які працюють з додатками, що взаємодіють з базами даних.

Важливою перевагою Rider є його кросплатформеність. Середовище розробки доступне для Windows, macOS та Linux, що дозволяє розробникам працювати на їхніх улюблених операційних системах без втрати функціональності.

Незважаючи на свою потужність, Rider має досить простий у використанні та налаштований інтерфейс користувача, який можна персоналізувати відповідно до власних уподобань. Він також має підтримку плагінів, що дозволяє розширювати його функціональність за потребою.

Загалом, JetBrains Rider є високопродуктивним та гнучким середовищем розробки для .NET, яке пропонує широкий спектр можливостей та інструментів, що допомагають розробникам створювати якісні додатки більш ефективно та зручно.

Серед недоліків можна виділити його вартість, оскільки Rider є платним продуктом, що може бути перешкодою для невеликих команд або окремих розробників, хоча JetBrains пропонує різні ліцензії та знижки. Крім того, Rider може вимагати більше системних ресурсів порівняно з легшими альтернативами, що може бути проблемою для розробників з обмеженими апаратними ресурсами. Також слід зазначити, що для розробників, які раніше не працювали з продуктами JetBrains, може знадобитися певний час для освоєння середовища та його численних функцій, оскільки воно має криву навчання.

Рисунок 2.3 – Логотип компанії JetBrains Rider

Eclipse є одним з найпопулярніших та найпотужніших середовищ розробки з відкритим кодом, яке широко використовується для розробки програмного забезпечення на різних мовах програмування (див. рис. 2.4). Хоча Eclipse традиційно асоціюється з розробкою на Java, завдяки своїй модульній архітектурі та широкому набору плагінів, він підтримує багато інших мов та технологій [7].

Переваги Eclipse**:**

- підтримка багатьох мов програмування;
- модульна архітектура;
- потужні інструменти для розробки на Java;
- інтеграція з системами контролю версій;
- підтримка DevOps.

Недоліки Eclipse**:**

- складність налаштування;
- обмежена підтримка .NET порівняно зі спеціалізованими середовищами для .NET.

Eclipse підтримує широкий спектр мов програмування та технологій завдяки своїй модульній архітектурі та великій кількості плагінів. Він є безкоштовним та має відкритий код, що робить його привабливим для індивідуальних розробників та малих команд. Проте, складність налаштування, вимоги до ресурсів та обмежена підтримка .NET роблять його менш оптимальним вибором для розробки додатків на платформі .NET

порівняно з спеціалізованими середовищами, такими як Microsoft Visual Studio або JetBrains Rider.

Рисунок 2.4 – Логотип компанії Eclipse

Visual Studio Code (VS Code) є легким, але потужним редактором коду, розробленим компанією Microsoft (див. рис. 2.4) . Він популярний серед розробників завдяки своїй кросплатформеності, широким можливостям налаштування та підтримці великої кількості мов програмування і технологій через розширення.

Причини не вибору Visual Studio Code у порівнянні з іншими IDE, такими як Visual Studio або JetBrains Rider, VS Code може бути менш потужним для розробки додатків на .NET, зокрема для WinForms. Він не має такої повної підтримки .NET та інструментів для розробки WinForms, як інші спеціалізовані IDE.

У VS Code не така глибока інтеграція з .NET та C# порівняно з Visual Studio або Rider. Це може вплинути на продуктивність розробки, особливо для великих проектів на .NET.

Рисунок 2.5 – Логотип Visual Studio Code

Враховуючи специфіку проекту з розробки додатку для бронювання спортивних майданчиків на платформі WinForms, Microsoft Visual Studio є найбільш підходящим середовищем розробки. Воно забезпечує повну інтеграцію з .NET Framework, потужні засоби для розробки, налагодження та тестування додатків, а також підтримує всі необхідні інструменти для ефективної роботи з C# та WinForms. Хоча інші середовища розробки мають свої переваги, Visual Studio залишається оптимальним вибором для цього проекту завдяки своїм можливостям та зручності використання.

2.4 Мова програмування

При розробці додатку для бронювання спортивних майданчиків важливо вибрати відповідну мову програмування, яка забезпечить ефективну розробку, високу продуктивність та легкість підтримки. Для нашого проекту було обрано мову програмування C#.

C# - це сучасна багатоплатформова мова програмування, розроблена компанією Microsoft. Вона тісно пов'язана з платформою .NET Framework та бібліотекою .NET Core, які забезпечують багатий набір класів та бібліотек для розробки різноманітних додатків [8].

Вибір C# як мови програмування часто зумовлений низкою переваг. По-перше, C# є основною мовою для розробки на платформі .NET Framework, що забезпечує повну сумісність з усіма інструментами та бібліотеками .NET [9]. Це дозволяє максимально використовувати можливості платформи та інтегруватися з іншими технологіями Microsoft. Крім того, C# має відмінну підтримку WinForms, що є ключовим для створення додатків з графічним інтерфейсом користувача.

Перевагою C# є поєднання простоти та потужності. Ця мова має багатий набір функцій, таких як автоматичне управління пам'яттю, LINQ (Language Integrated Query), асинхронне програмування, що підвищує продуктивність та

зручність розробки [10]. Важливим аспектом є безпека та управління пам'яттю через механізм збору сміття (Garbage Collection), що знижує ризик помилок і покращує стабільність додатків [11].

Нарешті, C# може похвалитися розвиненою спільнотою розробників та великою кількістю навчальних ресурсів, бібліотек, фреймворків та підтримки. Це робить розробку на C# більш доступною та зручною для широкого кола розробників [12].

Порівняно з іншими популярними мовами програмування, C# має як переваги, так і недоліки. Розглянемо деякі основні відмінності:

Java є універсальною мовою з широкою підтримкою кросплатформених рішень, багатою екосистемою та великою спільнотою. Однак, на відміну від C#, Java не має такої тісної інтеграції з WinForms і платформою .NET, що робить її менш зручною для розробки додатків з графічним інтерфейсом користувача на Windows.

Python відома своєю простотою та зручністю використання, має велику кількість бібліотек для різноманітних завдань, включаючи наукові обчислення, машинне навчання тощо. Водночас, Python не є основною мовою для розробки десктопних додатків з графічним інтерфейсом на Windows, і підтримка WinForms для Python обмежена.

C++ є потужною мовою з високою продуктивністю, що дозволяє створювати ефективні додатки з низьким рівнем доступу до апаратних ресурсів. Проте, C++ потребує більше зусиль для управління пам'яттю та відлагодження, що робить розробку більш складною та трудомісткою порівняно з C#.

2.5 Мова структурованих запитів SQL

Мова структурованих запитів SQL – це стандартизована мова програмування, яку використовують для управління реляційними базами даних та для виконання різних операцій над даними цих баз [13].

Основні цілі використання мови SQL [14]:

– створення, додавання, оновлення та видалення рядків даних;

– зміна структури індексів та таблиць бази даних;

– отримання різних підмножин інформації з систем управління реляційними базами даних.

2.6 Системи управління базами даних

Система управління базами даних (СУБД) відіграє фундаментальну роль у розробці сучасних додатків. Вона забезпечує централізоване та структуроване середовище для зберігання й ефективного управління величезними обсягами даних, уникаючи дублювання та неузгодженості. СУБД гарантує цілісність збережених даних завдяки механізмам перевірки правил і обмежень, а також захищає дані від несанкціонованого доступу або модифікації, забезпечуючи їхню конфіденційність і несуперечність.

Крім того, СУБД надає потужні інструменти для швидкого доступу до потрібної інформації за допомогою індексування та оптимізації запитів, дозволяючи ефективно працювати з величезними масивами даних. Підтримка транзакцій гарантує атомарність операцій із даними, забезпечуючи узгодженість даних у випадку відмов або перервань. Сучасні СУБД можуть масштабуватися для обробки колосальних обсягів даних та одночасних запитів, що є критично важливим для високонавантажених додатків.

При розробці додатку для бронювання спортивних майданчиків цього проекту була обрана Microsoft SQL Server (MS SQL) як основна СУБД.

Вибір Microsoft SQL Server (MS SQL) як системи управління базами даних для розробки додатку бронювання спортивних майданчиків було зумовлено кількома ключовими факторами. По-перше, MS SQL Server забезпечує повну сумісність та інтеграцію з іншими технологіями Microsoft, такими як .NET Framework та WinForms, що використовуватимуться в цьому

проекті. Це дозволяє легко застосовувати інструменти та бібліотеки для роботи з базою даних, спрощуючи процес розробки.

MS SQL Server відомий своєю високою продуктивністю та здатністю ефективно обробляти великі обсяги даних, підтримуючи складні запити та транзакції для швидкого доступу та обробки даних. Водночас, ця СУБД пропонує розширені функції безпеки, включаючи шифрування даних, управління правами доступу та аудит дій користувачів, забезпечуючи надійний захист конфіденційної інформації.

MS SQL Server має потужний набір інструментів для адміністрування баз даних та розробки додатків, таких як SQL Server Management Studio та Visual Studio, що забезпечують зручність управління, моніторингу, оптимізації та інтеграції з процесом розробки. Ця СУБД надає надійні засоби для резервного копіювання та відновлення даних, мінімізуючи ризики їх втрати та забезпечуючи доступність даних у випадку збоїв [15].

2.7 Програмна технологія .NET Framework

.NET Framework - це програмна платформа, розроблена компанією Microsoft, яка забезпечує уніфіковане середовище для створення, розгортання та виконання різноманітних типів додатків та веб-служб. Вона є однією з найбільш широко використовуваних технологій для розробки програмного забезпечення на Windows[16].

.NET Framework складається з двох основних компонентів:

— Середовище виконання (Common Language Runtime, CLR);

— Бібліотека класів (Framework Class Library, FCL).

.NET Framework широко використовується для розробки різноманітних додатків, включаючи настільні, веб, мобільні та хмарні рішення. Він є важливою частиною екосистеми Microsoft і тісно інтегрується з іншими технологіями та інструментами компанії.

2.8 Система контролю версій Git

Git - це розподілена система керування версіями коду, яка дозволяє ефективно відстежувати зміни в файлах проекту та співпрацювати в команді розробників [17]. На відміну від централізованих систем керування версіями, Git забезпечує кожного учасника проекту локальною копією повної історії розробки, що значно підвищує продуктивність роботи та забезпечує резервне копіювання. Git був розроблений з метою ефективного розподіленого, невілінійного робочого процесу .

Основними перевагами Git є його швидкодія, простота у використанні та розгалуженій нелінійній розробці, можливості для злиття гілок та розв'язання конфліктів, а також забезпечення цілісності даних завдяки незмінності історії коміті. Git дозволяє легко переміщатись між різними версіями коду, створювати гілки для ізолювання експериментальних функцій, об'єднувати їх з основною гілкою після завершення та тестування. Розробники можуть працювати паралельно над різними частинами коду без заважання один одному. Загалом, Git є потужним та гнучким інструментом, який суттєво підвищує ефективність командної розробки програмного забезпечення.

2.9 Висновки до другого розділу

Даний розділ детально висвітлює процес вибору технологічного стеку для розробки складних інформаційних систем, зокрема додатків для бронювання. Ретельний аналіз сучасних технологій дозволив обґрунтовано обрати архітектуру, мову програмування, систему управління базами даних та середовище розробки. Висновки цього розділу підкреслюють важливість правильного вибору технологій та середовища розробки для успішної реалізації додатку з урахуванням вимог проекту, досвіду розробників, підтримки спільноти, документації та ресурсів для навчання.

Обрана архітектура сприятиме стабільності, масштабованості, зручності підтримки та розвитку додатку, а продуманий дизайн забезпечить інтуїтивний інтерфейс, естетичну привабливість та ефективність взаємодії користувачів з програмою. Мова програмування C# була вибрана за її можливості швидкої розробки, підтримку кросплатформеності, наявність багатої бібліотеки та фреймворків, тоді як Microsoft SQL Server забезпечить надійність та ефективність зберігання, обробки та захисту даних. Microsoft Visual Studio обрано як основне середовище розробки завдяки його інтеграції з .NET Framework, зручним інструментам для налагодження та підтримці системи контролю версій Git. Порівняно з іншими середовищами, такими як JetBrains Rider, Eclipse та Visual Studio Code, Visual Studio є найбільш оптимальним вибором для розробки WinForms додатків. Використання додаткових фреймворків та бібліотек сприятиме покращенню продуктивності, забезпечить масштабованість та легкість у підтримці додатку. Загалом, вибір технологічного стеку, мови програмування, бази даних та середовища розробки, обґрунтований у цьому розділі, забезпечить ефективну, надійну та масштабовану розробку додатків для бронювання, відповідно до сучасних вимог та тенденцій галузі.

РОЗДІЛ 3. РОЗРОБКА ТА ТЕСТУВАННЯ ДОДАТКУ ДЛЯ БРОНЮВАННЯ СПОРТИВНИХ МАЙДАНЧИКІВ

3.1 Архітектура програмного продукту

Створення програмного продукту відбувається за дотриманням певних правил, які встановлюються при виборі однієї з моделей розробки програмного забезпечення. Для реалізації поставленого завдання була обрана каскадна модель, також відома як водоспадна модель. Під моделлю зазвичай розуміється структура, яка визначає послідовність виконання та взаємозв'язок процесів, дій і завдань протягом життєвого циклу. Каскадна модель розробки програмного забезпечення передбачає виконання етапів у строгій послідовності, де кожен наступний крок розпочинається лише після завершення попереднього. Ця модель характеризується лінійним підходом, де процес розробки рухається вниз, наче водоспад, проходячи всі стадії від визначення вимог до експлуатації та супроводу готового продукту. Схема каскадного процесу розробки програмного забезпечення зображена на рисунку 3.1, який наочно ілюструє послідовність етапів та їх взаємозв'язок.

Рисунок 3.1- Каскадна модель розробки програмного забезпечення.

Етапи каскадної моделі включають:

– Аналіз вимог - На цьому етапі збираються та документуються всі вимоги до програмного продукту від замовника або користувачів. Вимоги мають бути чіткими, зрозумілими та повними;

– Проектування- На основі зібраних вимог розробляється проект системи, який включає архітектуру, дизайн інтерфейсу, бази даних та інших компонентів;

– Реалізація (програмування) - Розробники реалізують програмний код згідно з розробленим проектом системи;

– Впровадження - Після успішного тестування програмний продукт вводиться в експлуатацію та встановлюється у кінцевих користувачів.

Перехід на наступний етап можливий лише після повного завершення поточного. Будь-які зміни вимог на пізніших стадіях можуть вимагати повернення до попередніх етапів, що робить каскадну модель негнучкою. Однак ця модель забезпечує чітку структуру та дисципліну розробки для проектів зі стабільними вимогами.

Модель-Представлення-Презентер (MVP) є одним із найпоширеніших архітектурних патернів для розробки програмного забезпечення з чітким розділенням обов'язків між логікою бізнесу, логікою представлення та інтерфейсом користувача [18]. MVP був обраний для розробки додатку для бронювання спортивних майданчиків завдяки своїм перевагам. По-перше, він забезпечує розділення відповідальностей між Model (логіка бізнесу), View (інтерфейс користувача) та Presenter (логіка представлення), що спрощує розробку, тестування та підтримку кожного компонента окремо, надаючи гнучкість у разі внесення змін (див. рис. 3.2).

MVP сприяє високій тестованості, оскільки Presenter, який містить всю логіку представлення, не залежить від View, що дозволяє легко писати юніт-тести для перевірки функціональності додатку, забезпечуючи надійність та стабільність. Завдяки чіткому розподілу між компонентами, MVP спрощує процес підтримки та масштабованості додатку, дозволяючи змінювати логіку

бізнесу та інтерфейс користувача незалежно один від одного, полегшуючи внесення змін та розширення функціональності.

Для додатку з багатьма екранними формами та складною логікою представлення, MVP забезпечує зручність управління станами інтерфейсу користувача, де Presenter відповідає за обробку подій користувача та оновлення View, роблячи додаток більш організованим і зрозумілим. Нарешті, MVP є незалежним від конкретних технологій або платформ, що дозволяє використовувати цей патерн у різних середовищах розробки, забезпечуючи гнучкість у виборі інструментів та технологій для реалізації додатку.

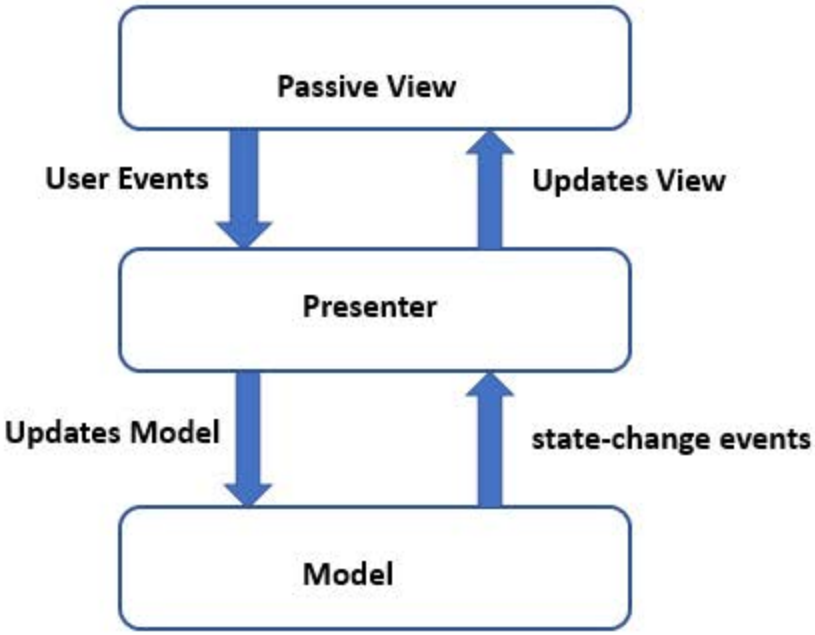

Рисунок 3.2- Архітектура MVP.

3.2 Проєктування та реалізація бази даних

При роботі з великими обсягами даних у базах даних, користувачі стикаються з проблемою стрімкого зростання інформації, що ускладнює процеси її обробки та модифікації. Внесення змін в окремі поля вручну,

особливо якщо ці зміни потрібно повторити для всього набору даних, а інколи навіть для декількох наборів, вимагає величезних витрат часу та зусиль, перетворюючись на вкрай виснажливе завдання. Керування такими масивами інформації стає надзвичайно складним і трудомістким процесом через геометричне зростання обсягів даних.

Програмне забезпечення для керування базами даних призначене для систематизації, перегляду, пошуку, створення, оновлення, видалення та маніпулювання даними, які зберігаються у структурованому вигляді в реляційних базах даних. Більшість систем керування реляційними базами даних використовують мову структурованих запитів SQL для взаємодії з даними.

Фундаментальною одиницею зберігання даних у такій базі даних є таблиці, що складаються з рядків та стовпців. Рядки виступають у ролі записів, тоді як стовпці представляють різні поля даних. Кожен рядок містить комірки інформації, що стосуються конкретної сутності. Комірки в межах одного рядка є різними фрагментами даних або атрибутами, пов'язаними з цією сутністю. Така структура забезпечує зручність аналізу, пошуку, оновлення та модифікації даних у таблиці.

Для повноцінного функціонування додатку бронювання спортивних майданчиків була розроблена спеціальна реляційна база даних за допомогою СУБД Microsoft SQLServer.

База даних для додатку бронювання спортивних майданчиків складається з трьох основних таблиць:

- FootballFields;
- Reservations;
- Users.

Кожна таблиця зберігає відповідні дані та має свої ключі та зв’язки. Таблиця FootballFields зберігає інформацію про футбольні поля.

Детальну структуру можна побачити в таблиці 3.1.

Таблиця 3.1

Структура таблиці FootballFields

Назва стовпця	Тип даних	Опис
Id	Ціле число (int)	Унікальний ідентифікатор кожного футбольного поля.
FieldName	Текстовий (nvarchar)	Назва футбольного поля (довжина до 100 символів).
OwnerId	Ціле число (int)	Зовнішній ключ, вказує на власника футбольного поля (зв’язок з таблицею Users).
Size	Текстовий (nvarchar)	Розмір футбольного поля (довжина до 50 символів).
Surface	Текстовий (nvarchar)	Покриття футбольного поля (довжина до 50 символів).
Location	Текстовий (nvarchar)	Адреса футбольного поля (довжина до 100 символів).
RentalCost	Десяткове число (decimal)	Вартість оренди футбольного поля (точність 10,2).
City	Текстовий (nvarchar)	Місто, де розташоване футбольне поле (довжина до 100 символів, може бути null).

Таблиця Reservations у базі даних системи бронювання футбольних полів є центральним сховищем інформації про всі бронювання, здійснені користувачами, і забезпечує зв'язок між ключовими сутностями, такими як футбольні поля, користувачі та самі бронювання. Детальну структуру можна побачити в таблиці 3.2

Таблиця 3.2

Структура таблиці Reservations

Назва стовпця	Тип даних	Опис
Id	Ціле число (int)	Унікальний ідентифікатор кожного бронювання.
FieldId	Зовнішній ключ (FK)	Вказує на заброньоване футбольне поле (зв'язок з таблицею FootballFields).
OwnerId	Зовнішній ключ (FK)	Вказує на користувача, який зробив бронювання (зв'язок з таблицею Users).
ReservationStart	Дата та час (datetime)	Початок бронювання.
ReservationEnd	Дата та час (datetime)	Кінець бронювання.
RentalCost	Десяткове число (decimal)	Вартість оренди за період бронювання (точність 10,2).
AcceptedByUserId	Зовнішній ключ (FK)	Вказує на користувача, який прийняв бронювання (зв'язок з таблицею Users).

Таблиця Users є фундаментальним компонентом бази даних, що забезпечує безпечне та організоване зберігання інформації про користувачів, керування правами доступу, персоналізацію користувацького досвіду та цілісність даних у системі. Детальну структуру можна побачити в таблиці 3.3

Таблиця 3.3

Структура таблиці Users

Назва стовпця	Тип даних	Опис
Id	Ціле число (int)	Унікальний ідентифікатор кожного користувача.
Email	Текстовий (nvarchar)	Електронна адреса користувача (довжина до 255 символів, не може бути null).
Name	Текстовий (nvarchar)	Ім’я користувача (довжина до 100 символів, може бути null).
Surname	Текстовий (nvarchar)	Прізвище користувача (довжина до 255 символів, може бути null).
HashPass	Текстовий (nvarchar)	Хешований пароль користувача (довжина до 255 символів, не може бути null).
Locations	Текстовий (nvarchar)	Локації користувача (довжина до 255 символів, може бути null).
RoleName	Текстовий (nvarchar)	Роль користувача (наприклад, адміністратор, користувач) (довжина до 255 символів, може бути null).
PhoneNumber	Текстовий (nvarchar)	Номер телефону користувача (довжина до 20 символів, може бути null).

Зв’язки між таблицями є важливими аспектами баз даних. Вони дозволяють об’єднувати інформацію з різних таблиць, що допомагає виконувати потужні запити та аналізувати дані.

Основні типи зв’язків:

- відношення один до одного;
- один до багатьох;
- та багато до багатьох.

При моделюванні даних важливо визначити ці зв'язки та налаштувати таблиці та їхні поля відповідно. Діаграми взаємозв'язків сутностей допомагають візуально представити ці зв'язки. На рисунку 3.3 приведена діаграма «сутність-зв'язок», котра використовується для створення моделі даних.

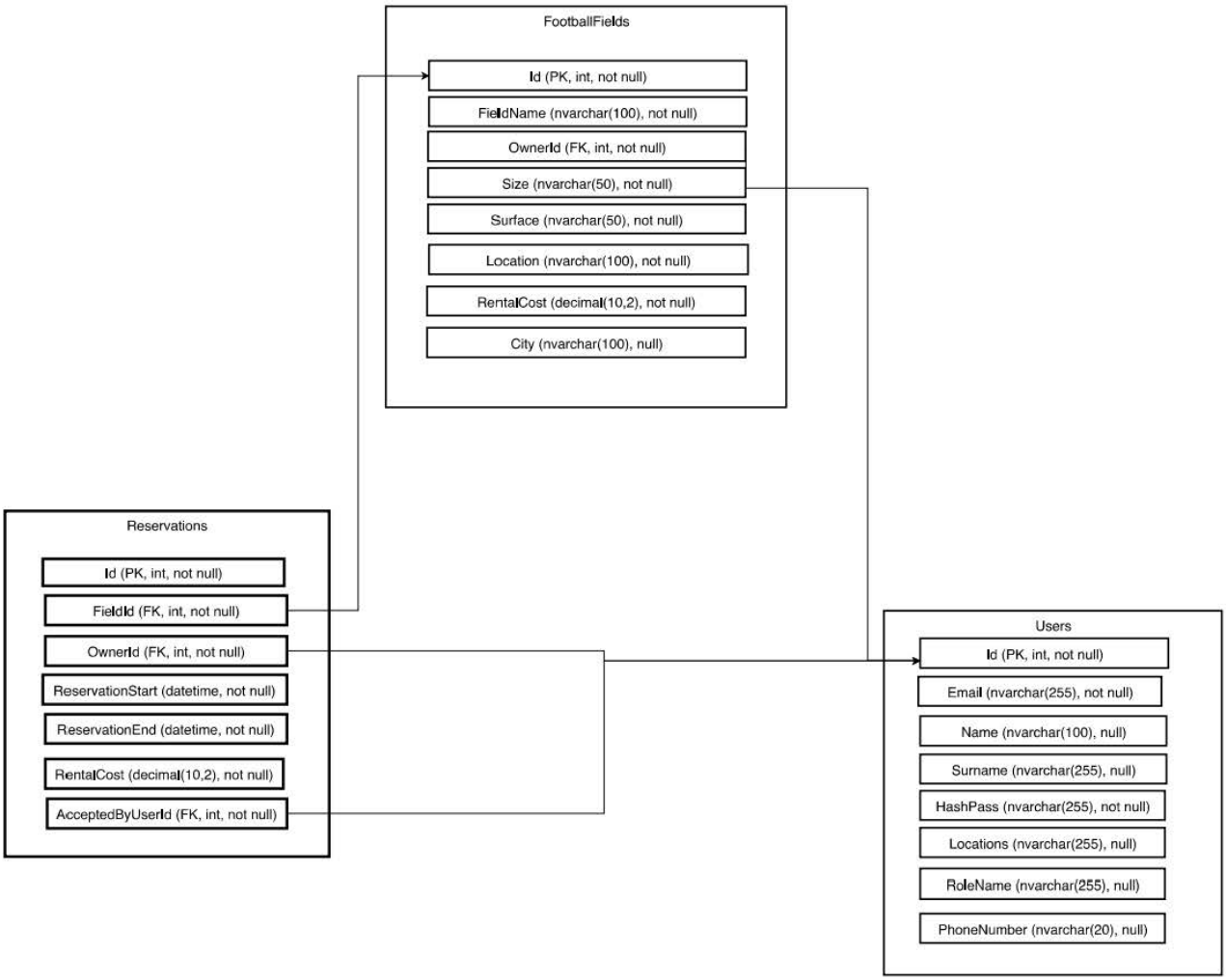

Рисунок 3.3- – Проєкт схеми даних системи у вигляді ERD

Реалізовуємо базу даних в СУБД Microsoft SQLServer. Результати наведено на рисунках 3.4 - 3.6.

Таблиця FootballFields буде містити дані про спортивні майданчики, а саме: унікальний ID майданчика, назву, ID власника поля, розмір, покриття, вулицю, вартість, місто.

DESKTOP-Q0AMUTU....ndDB - dbo.Users DESKTOP-Q0AMUTU....dbo.Res

Column Name	Data Type	Allow Nulls
Id	int	☐
FieldName	nvarchar(100)	☐
OwnerId	int	☐
Size	nvarchar(50)	☐
Surface	nvarchar(50)	☐
Location	nvarchar(100)	☐
RentalCost	decimal(10, 2)	☐
City	nvarchar(100)	☑
		☐

Рисунок 3.4 – Зміст таблиці FootballFields

Таблиця Reservations буде містити дані про бронювання майданчика, а саме: унікальний ID, ID майданчика, ID власника, початок оренди та кінець, вартість оголошення, ID користувача який забронював майданчик.

DESKTOP-Q0AMUTU....ndDB - dbo.Users DESKTOP-Q0AMUTU....

Column Name	Data Type	Allow Nulls
Id	int	☐
FieldId	int	☐
OwnerId	int	☐
ReservationStart	datetime	☐
ReservationEnd	datetime	☐
RentalCost	decimal(10, 2)	☐
AcceptedByUserId	int	☑
		☐

Рисунок 3.5 – Зміст таблиці Reservations

Таблиця Users містить дані про користувачів, а саме: унікальний ID, електрону пошту, ім'я, прізвище, хешований пароль, локацію (місто в якому живе користувач), його роль, номер телефону.

DESKTOP-Q0AMUTU....ndDB - dbo.Users | DESKTOP-Q0AMUTU....d

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	Email	nvarchar(255)	☐
	Name	nvarchar(100)	☑
	Surname	nvarchar(255)	☑
	HashPass	nvarchar(255)	☐
	Locations	nvarchar(255)	☑
	RoleName	nvarchar(255)	☑
	PhoneNumber	nvarchar(20)	☑
▶			☐

Рисунок 3.6 – Зміст таблиці Users

Отже, основні таблиці були створені, реалізована БД та її схема зв'язку, а це означає, що можна переходити до другого етапу – розробка інтерфейсу.

Лістинг програмного забезпечення можна побачити в додатку А.

3.3 Розробка інтерфейсу

Зважаючи на важливість простоти при створенні інтерфейсу, мінімалістичний дизайн є найкращим вибором як для розробника, так і для користувача. Для розробників це означає менше часу, витраченого на дизайн, а для користувачів - зручність використання [19]. Тому для реалізації програми для маркетингових досліджень було обрано середовище Windows Forms для побудови інтерфейсу.

Windows Forms - це платформа інтерфейсу користувача для створення класичних додатків Windows. Вона забезпечує один з найефективніших способів створення класичних програм за допомогою візуального конструктора в Visual Studio. Такі функції, як розміщення візуальних

елементів керування шляхом перетягування, полегшують створення класичних додатків [20].

Основними елементами інтерфейсу є стандартні форми Windows Forms в Панелі елементів(див. рис. 3.7). Серед них можна виділити:

- Button (Кнопка);
- Label (Мітка);
- TextBox (Текстове поле);
- ComboBox (Комбіноване поле);
- DataGridView (Таблиця даних);
- DateTimePicker (Вибір дати і часу);
- CheckBox (Прапорець);
- RadioButton (Перемикач);
- PictureBox (Зображення);
- Panel (Панель).

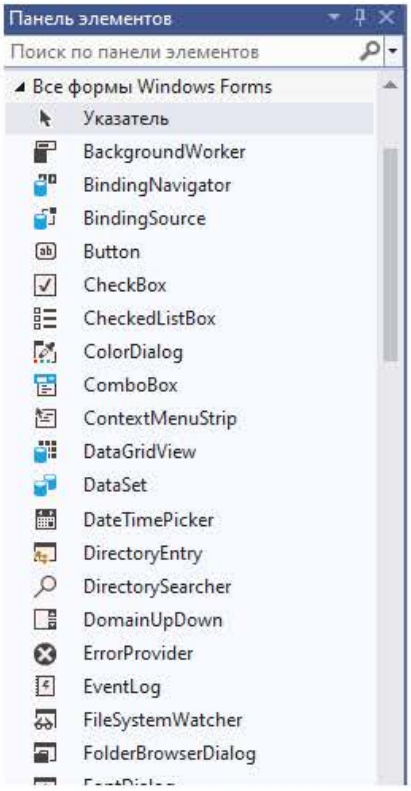

Рисунок 3.7 – Панель елементів

Користуючись панелю елементів. Створюємо форму для авторизації (див. рис. 3.8), використовуючи Panel, TextBox, PictureBox, Label та Button.

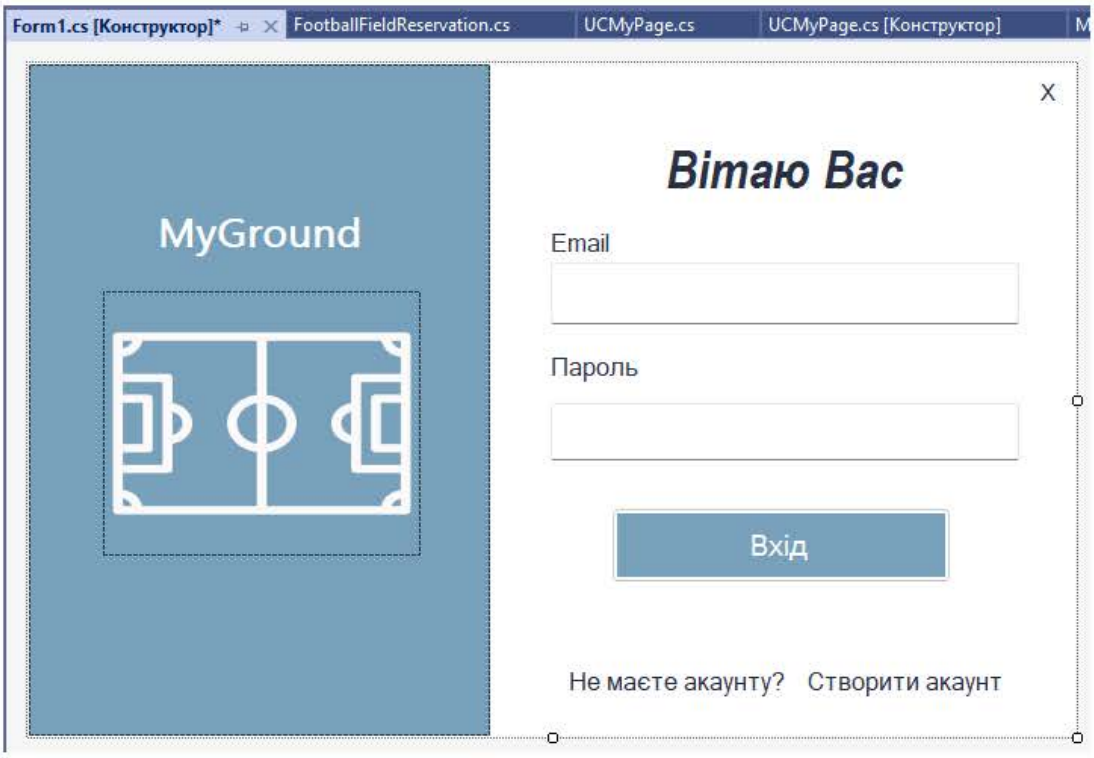

Рисунок 3.8 –Макет форми авторизації

Аналогічні дії проводимо з формою для реєстрації (див. рис. 3.9).

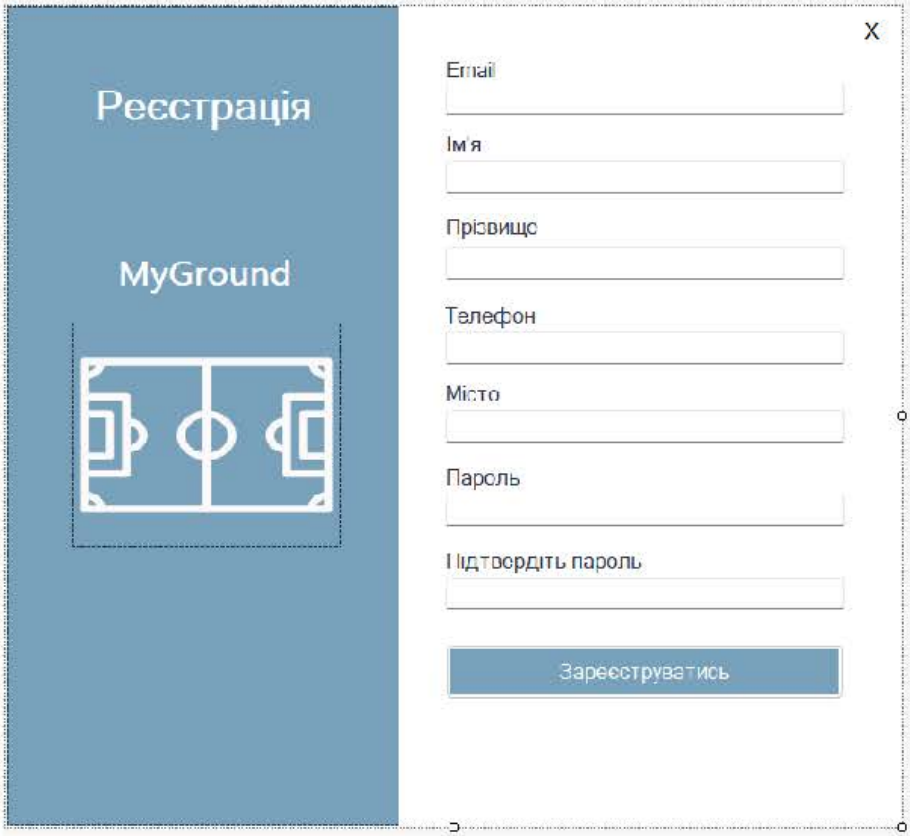

Рисунок 3.9 – Макет форми реєстрації

Коли форми авторизації та реєстрації створені, переходимо до наступного кроку це створення головної форми додатку (див. рис. 3.10). Ліворуч меню з кнопками. Після натиску на відповідну з них буде відбуватись перехід між User Control.

Рисунок 3.10 – Макет головної форми

Кожній кнопці відповідає певний макет типу User Control. Це дає змогу не створювати багато форм, а мати одну головну та здійснювати навігацію в додатку через користувацькі елементи керування (див. рис. 3.11).

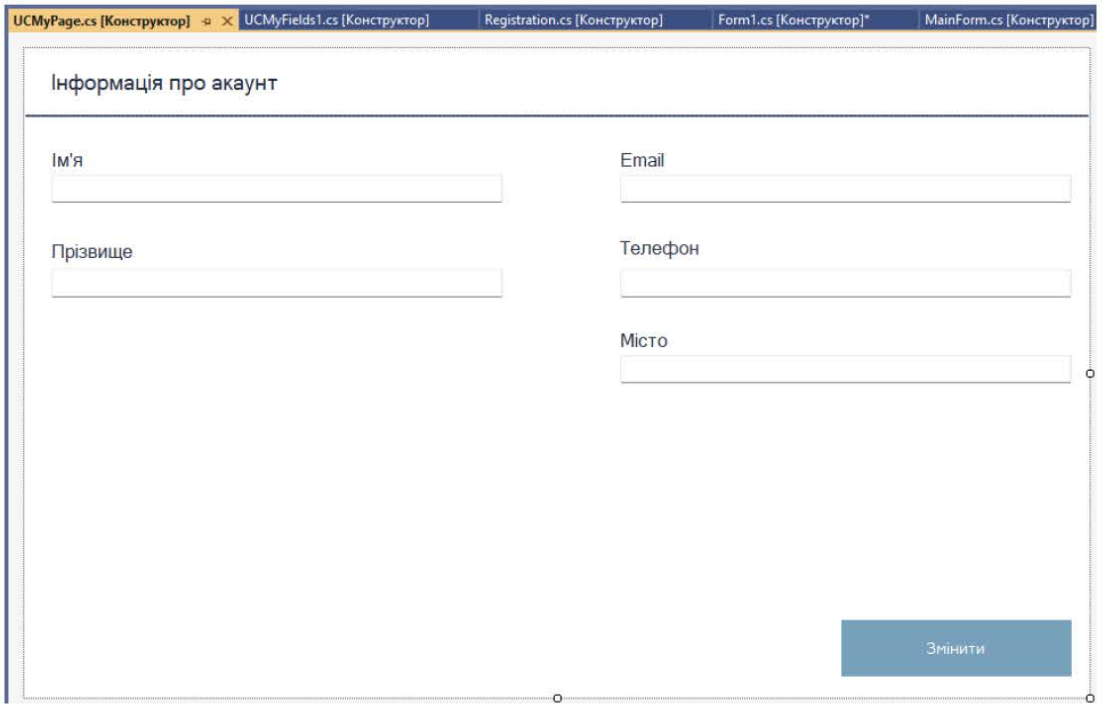

Рисунок 3.11 – Макет користувацького елементу керування

3.4 Інструкція роботи користувача з системою.

В нашому додатку існують три ролі:

– Користувач;

– Менеджер ;

– Адміністратор.

Розглянемо дії за звичайного користувача.

При запуску додатку відкривається вікно авторизації (див. рис. 3.12).

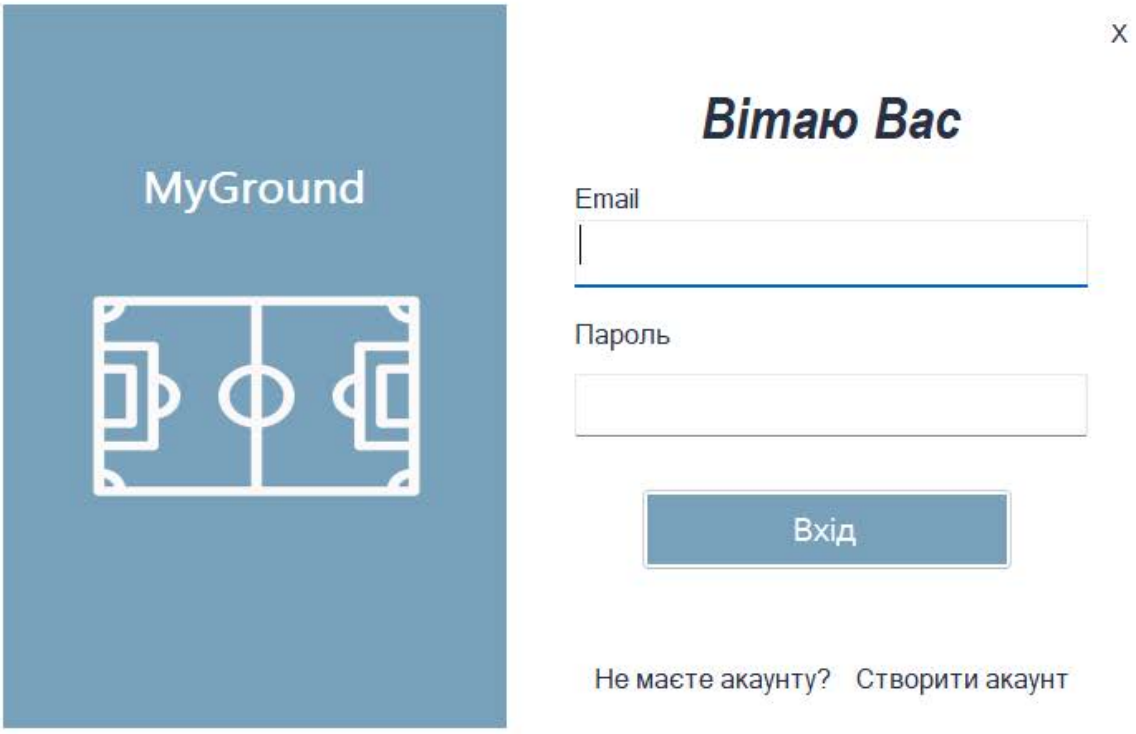

Рисунок 3.12 – Вікно авторизації

Якщо користувач вперше користується додатком і в нього не має акаунту, то він може натиснути на «Створити акаунт», після чого відкриється вікно реєстрації (див. рис. 3.13).

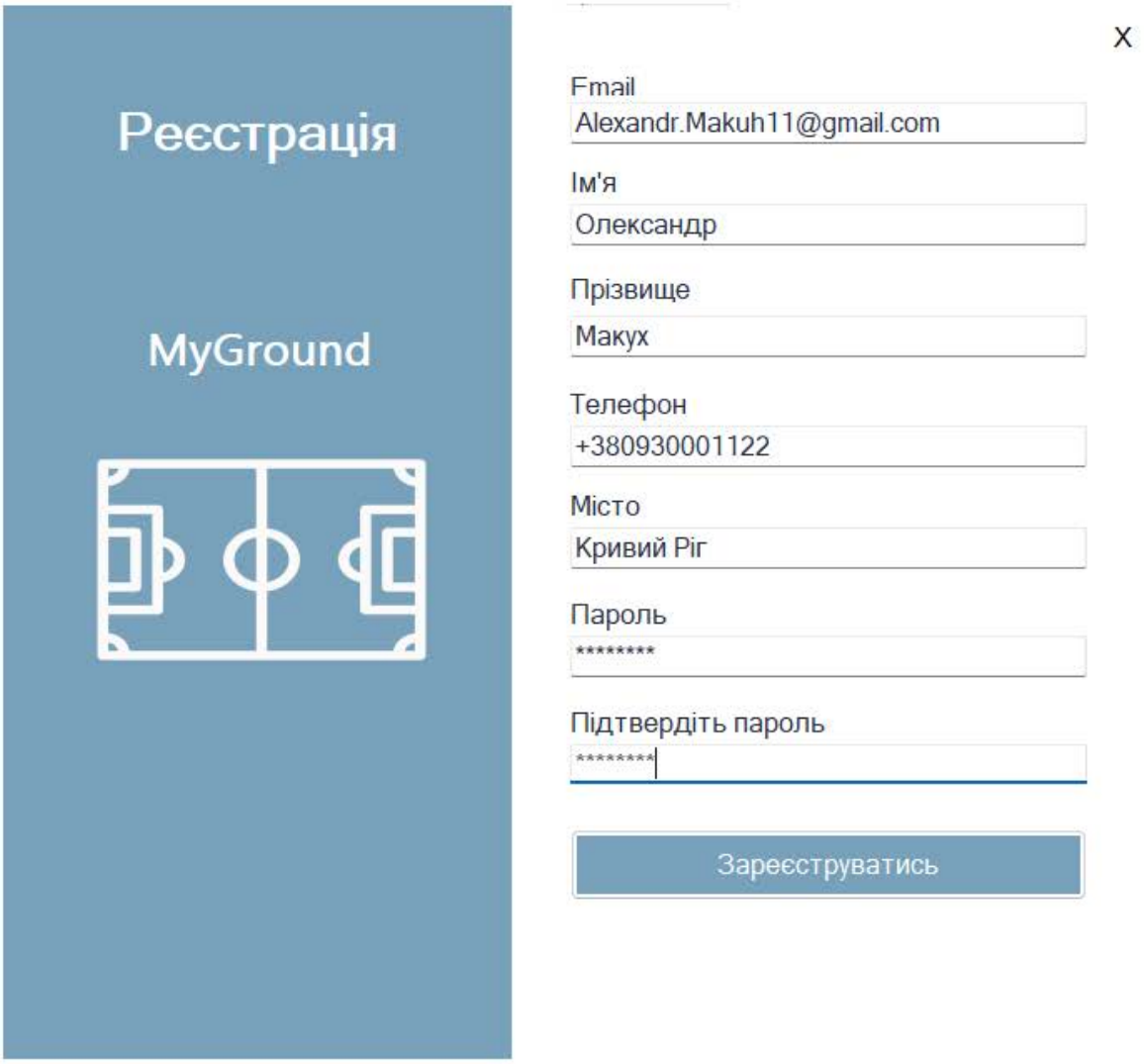

Рисунок 3.13 – Вікно реєстрації

Якщо ведені дані не відповідають вимогам, то користувач буде інформований відповідним повідомленням. Такі випадки трапляються якщо електронна адреса вже є в базі даних, якщо телефон не відповідає певному формату, якщо паролі не збігаються чи пароль менше 8 символів (див. рис. 3.14).

Після того як всі дані відповідають вимогам користувача зареєстровано. І ми знову переходимо на вікно авторизації

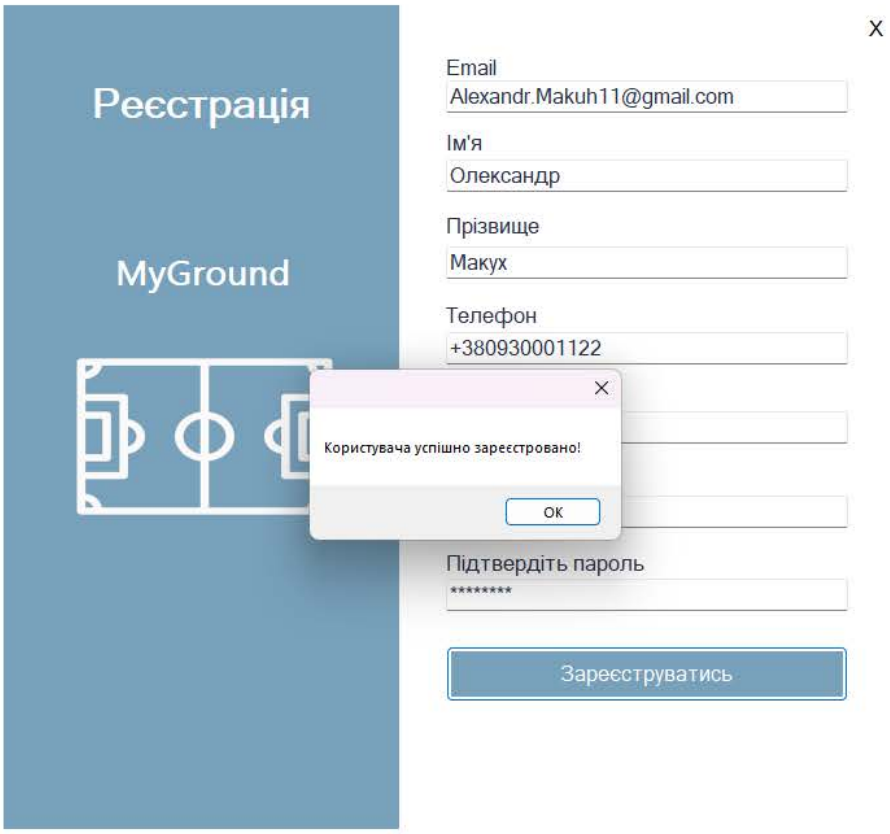

Рисунок 3.14 – Повідомлення про успішну реєстрацію

У разі якщо вхідні дані не є коректними то в доступі відхилено та виведено повідомлення (див. рис. 3.15).

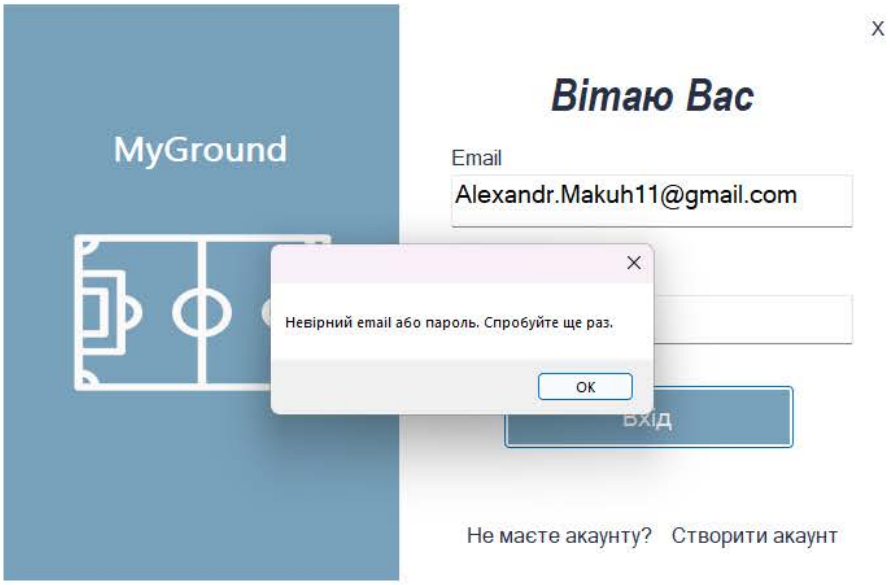

Рисунок 3.15 – Повідомлення про не успішну авторизацію

Після того як електронна пошта та хешований пароль співпадають в базі даних то користувачу дозволено в доступі до додатку. Відкривається головне

вікно. Та вкладка моя сторінка де користувач може змінити дані про себе (див. рис. 3.16).

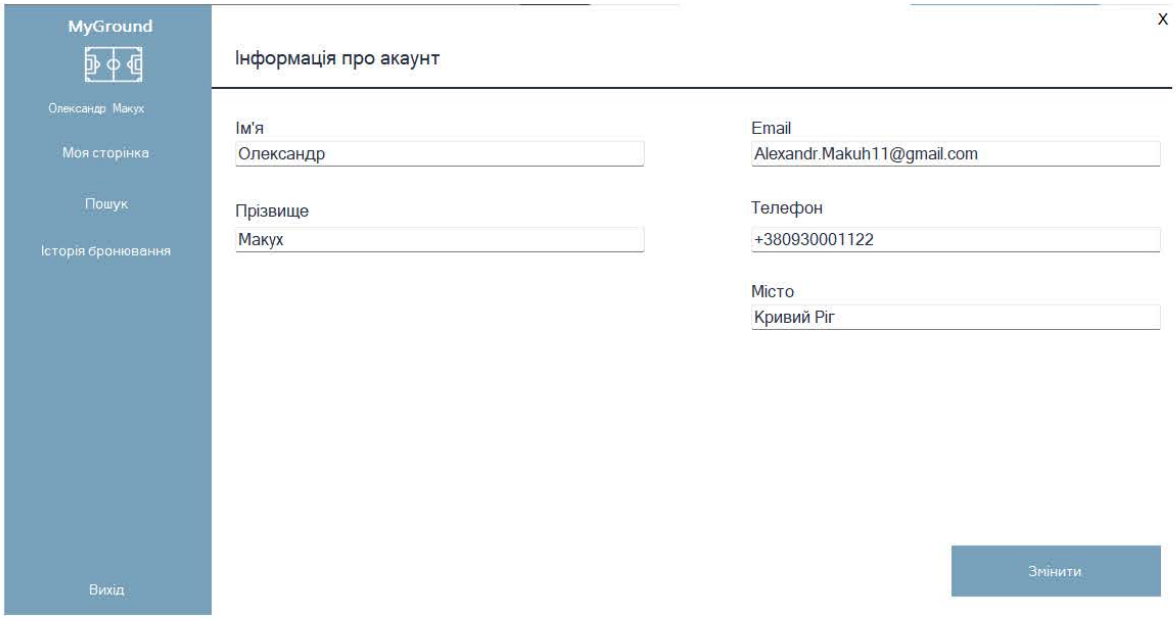

Рисунок 3.16 – Головне вікно

Користувач після натиску в боковому меню «Пошук» переходить на вкладку де міститься інформацію про вільні спортивні майданчики (див. рис. 3.17). Використовуючи фільтр користувач може змінити дату та місце знаходження спортивного майданчика. У відповідному полі він бачить всю інформацію про поле та час бронювання. Після натиску на кнопку забронювати на екран буде виведено повідомлення.

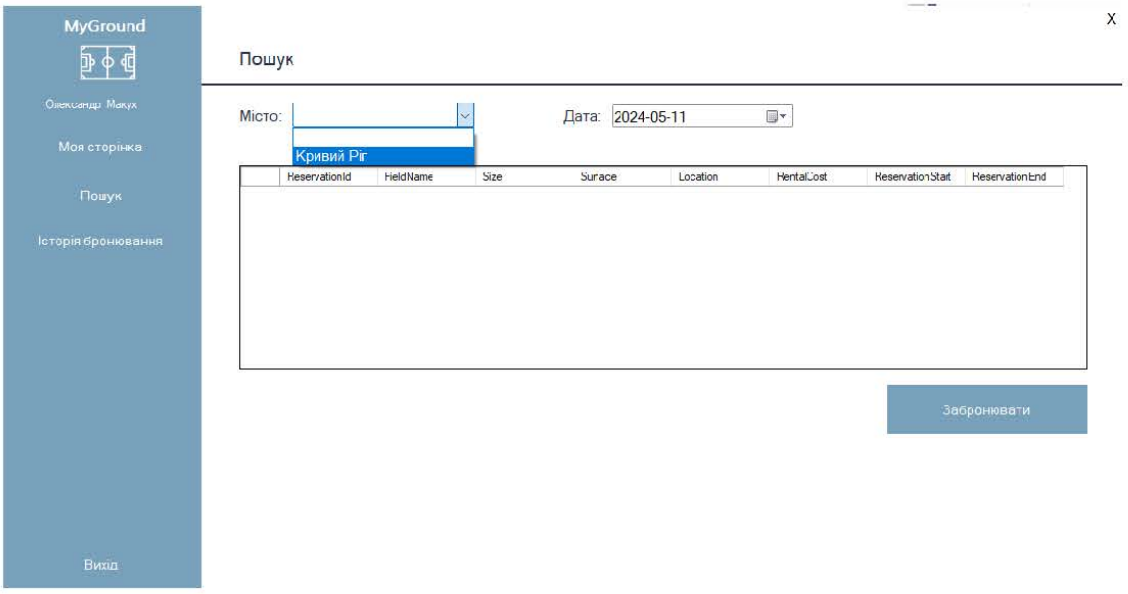

Рисунок 3.17 – Вкладка пошуку вільних майданчиків

Користувач має можливість переглянути інформацію про свою історію бронювання натиснувши на кнопку «Історія бронювання» (див. рис. 3.18).

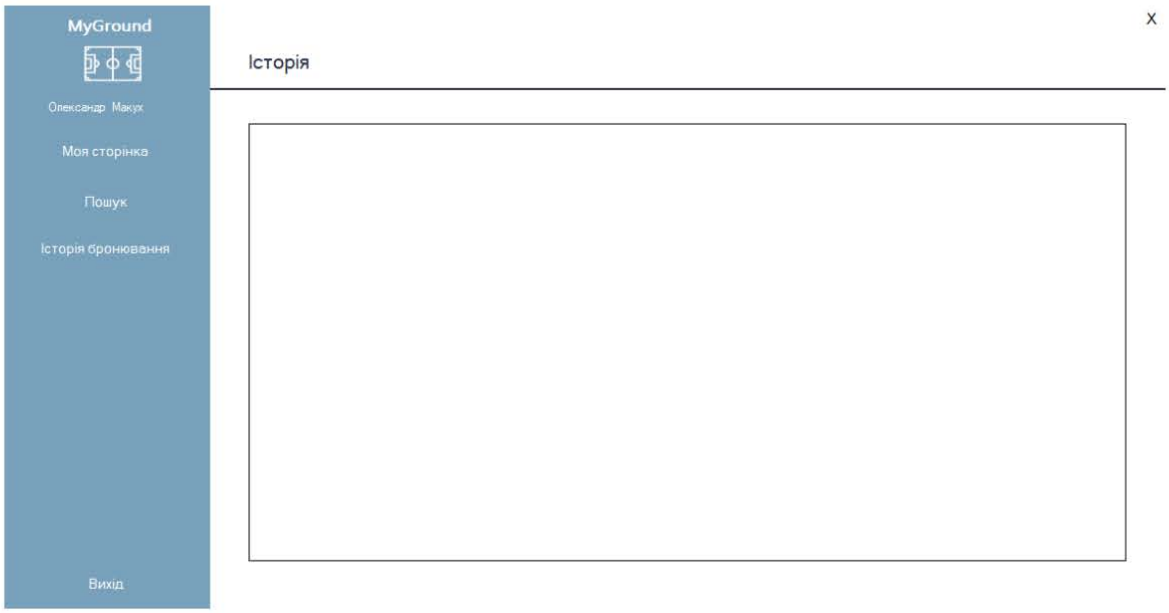

Рисунок 3.18 – Вкладка історії бронювання

Якщо користувач забажає вийти то він має натиснути на кнопку «Вихід» внизу бічного меню.

Наступна роль це – Менеджер. В нього є більше функцій ніж у Користувача. Після успішної авторизації він має додаткові кнопки (див. рис. 3.19).

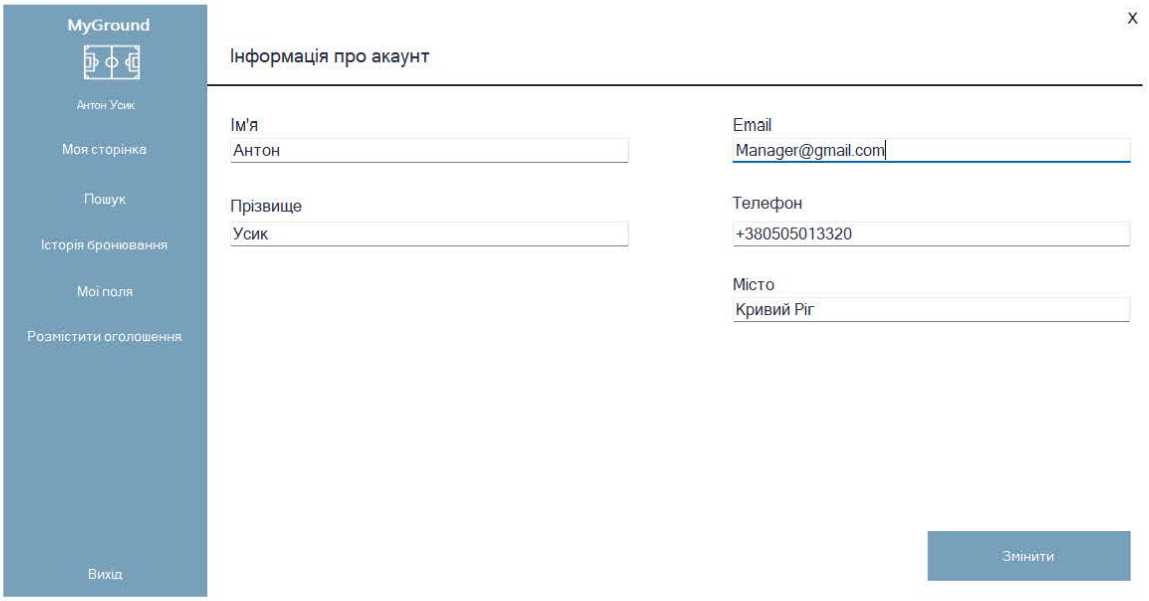

Рисунок 3.19 – Головне меню для менеджера

Після того як він натисне на кнопку «Мої поля» менеджер має змогу переглянути інформацію про створені ним майданчики (див. рис. 3.20).

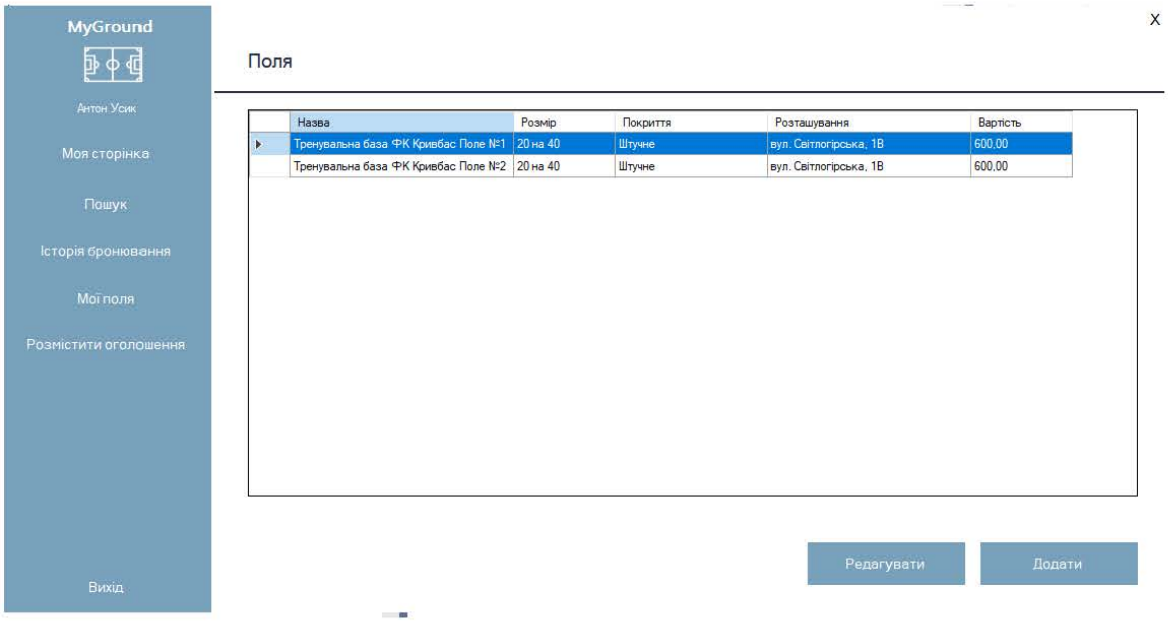

Рисунок 3.20 – Вкладка «Поля»

Менеджер може додати поле натиснувши кнопку. Відкриється форма де він може ввести дані про поле та додати в базу даних (див. рис. 3.21).

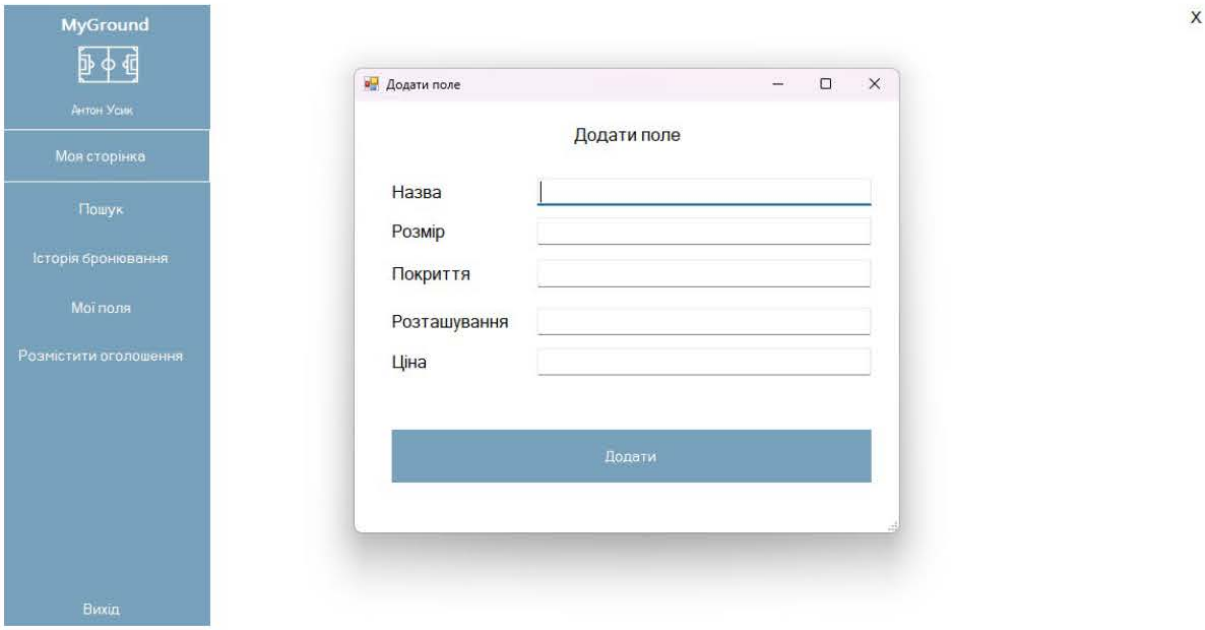

Рисунок 3.21– Вікно додавання поля.

Менеджер натиснувши кнопку розмістити оголошення, перейде на відповідну вкладку в головному вікні. Та він має змогу заповнивши поля створити оголошення що є вільний майданчик (див. рис. 3.22).

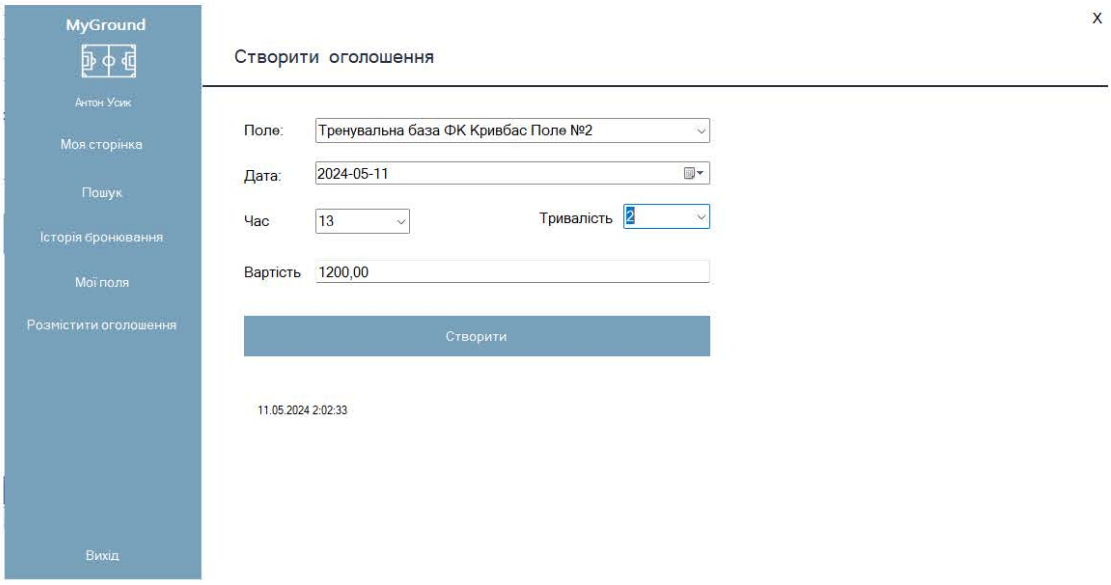

Рисунок 3.22– Вкладка для створення оголошення.

Адміністратор може змінювати роль для кожного користувача. Якщо є домовленість з власником майданчика те що він хоче користуватись додатком то адміністратор надає власнику роль менеджера (див. рис. 3.23).

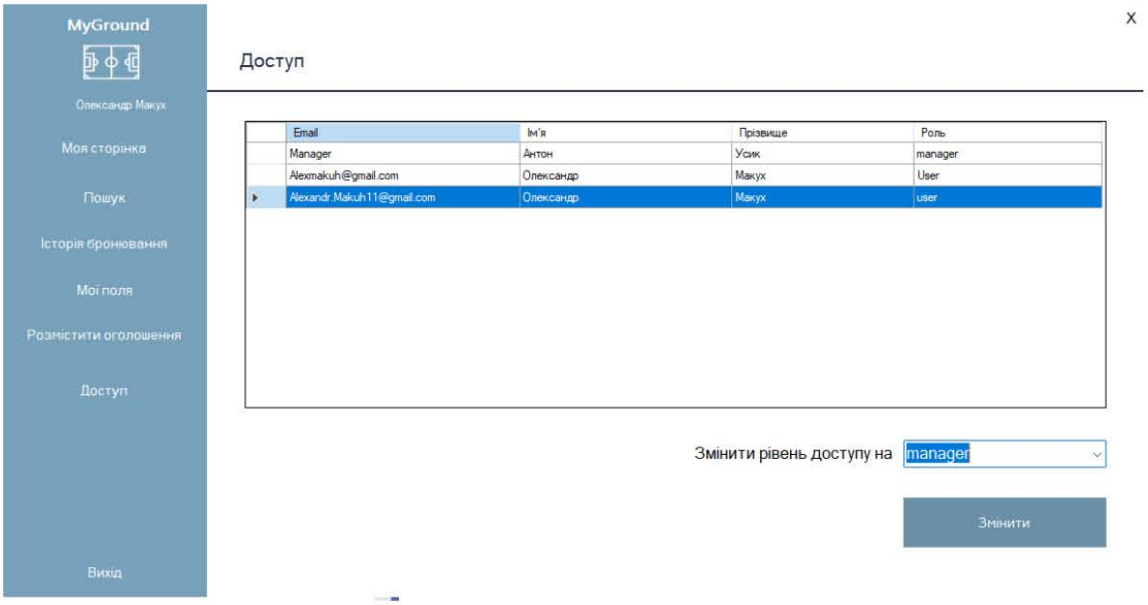

Рисунок 3.23– Вікно зміни доступу.

3.5 Висновки до третього розділу

Розглянули процес розробки та тестування додатку для бронювання спортивних майданчиків. На початку розділу було описано архітектуру програмного продукту, зокрема обрану каскадну модель розробки та патерн

Model-View-Presenter (MVP). Каскадна модель дозволила забезпечити чітку структуру розробки та послідовність виконання етапів, а MVP сприяв розподілу обов'язків та підвищенню тестованості системи.

Проєктування та реалізація бази даних стали наступним важливим етапом. Було створено реляційну базу даних за допомогою СУБД Microsoft SQL Server, яка складається з трьох основних таблиць: FootballFields, Reservations та Users. Це дозволило ефективно зберігати та керувати великими обсягами даних, забезпечуючи цілісність та зручність доступу до них.

Розробка інтерфейсу користувача була виконана з використанням Windows Forms, що дало можливість створити простий і зручний для користувачів інтерфейс. Були розроблені основні форми додатку, включаючи форми авторизації, реєстрації та головну форму з можливістю навігації через користувацькі елементи керування.

Завдяки ретельному підходу до проєктування та реалізації кожного етапу, створений додаток забезпечує стабільну роботу, зручність використання та можливість масштабування. Інструкція для користувачів допомагає легко освоїти роботу з додатком, що підвищує його доступність та ефективність у використанні.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досліджено предметну область бронювання спортивних майданчиків та вимоги, які висуваються до сучасних інформаційних систем цієї сфери. На основі аналізу існуючих аналогічних рішень було визначено їх переваги та недоліки, функціональні можливості та особливості реалізації.

Для розробки додатку було обрано технологічний стек, що складався з мови програмування C# та середовища розробки Microsoft Visual Studio для створення клієнтської частини, а також системи управління базами даних Microsoft SQL Server для реалізації серверної частини та збереження даних. Вибір цих технологій обумовлений їх високою продуктивністю, надійністю, кросплатформеністю та широкими можливостями для розробки сучасних додатків.

У результаті був створений програмний продукт, який має зручний та інтуїтивно зрозумілий користувацький інтерфейс, дозволяє здійснювати пошук вільних спортивних майданчиків за різними критеріями, переглядати їх розклад, бронювати в зручний час і керувати власними бронюваннями. Система також підтримує ролі адміністратора для управління майданчиками, їх розкладами та тарифами.

Розроблений додаток відповідає вимогам до сучасних інформаційних систем, таким як безпека, масштабованість, надійність, зручність використання та можливість подальшого вдосконалення і розширення функціоналу. Він може бути використаний спортивними клубами, фітнес-центрами, муніципальними спортивними об'єктами та іншими організаціями для ефективного управління бронюванням своїх майданчиків.

Отримані результати демонструють стабільну роботу додатку, зручність у використанні та можливість його масштабування. Інструкція для користувачів сприяє легкому освоєнню додатку, що підвищує його доступність та ефективність у використанні.

Практичне значення одержаних результатів полягає у можливості їх використання для подальшого розвитку додатків подібного типу, а також для оптимізації процесу бронювання спортивних майданчиків, що сприяє підвищенню ефективності їх використання.

У перспективі передбачається подальше вдосконалення додатку для бронювання спортивних майданчиків шляхом розширення його функціональності. Зокрема, планується додати можливість бронювання інших типів спортивних об'єктів, таких як басейни, тенісні корти та фітнес-зали. Також буде розроблена мобільна версія для платформ iOS та Android, щоб забезпечити зручний доступ до сервісу з мобільних пристроїв.

Крім того, заплановано інтегрувати додаток з платіжними системами для впровадження можливості онлайн-оплати бронювань, що підвищить зручність для користувачів. Для адміністраторів спортивних комплексів буде додано розширені функції аналітики та генерації звітів, що допоможе оптимізувати управління ресурсами.

Інтерфейс додатку буде розширений для підтримки багатомовності, що зробить його доступним для користувачів з різних країн. Усі ці вдосконалення спрямовані на підвищення зручності, ефективності та масштабованості системи бронювання спортивних майданчиків.

Кваліфікаційна робота виконана у відповідності до стандарту спеціальності 121 – «Інженерія програмного забезпечення» і демонструє володіння такими компетентностями як:

– Здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм;

– Здатність розробляти архітектури, модулі та компоненти програмних систем;

– Володіння знаннями про інформаційні моделі даних, здатність створювати програмне забезпечення для зберігання, видобування та опрацювання даних.

Серед результатів навчання, визначених стандартом кваліфікаційна робота реалізовує наступні:

– Здатність застосовувати методи розробки алгоритмів, конструювання програмного забезпечення та структур даних і знань;

– Володіти мовами системного програмування та методами розробки програм, що взаємодіють з компонентами комп'ютерних систем;

– Застосовувати відповідні сучасні інформаційні системи і технології при проектуванні та розробці програмного забезпечення;

– Вміння працювати з базами даних та забезпечувати їхню інтеграцію з програмними системами;

– Здатність до впровадження і підтримки інформаційної безпеки у програмних продуктах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. CeleBreak [Електронний ресурс]. – Режим доступу: https://celebreak.com/en/

2. Skedda [Електронний ресурс]. – Режим доступу: https://www.skedda.com/home.

3. Palms [Електронний ресурс]. – Режим доступу: https://palms.app/ru-ua.

4. PlaySeek [Електронний ресурс]. – Режим доступу: https://playseek.io/.

5. Visual Studio [Електронний ресурс]. – Режим доступу: https://visualstudio.microsoft.com/.

6. JetBrains Rider [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/rider/.

7. Eclipse Community [Електронний ресурс]. – Режим доступу: https://www.eclipse.org/community/eclipse_newsletter/2020/march/1.php.

8. C# и .NET | Начало работы с Visual Studio [Електронний ресурс]. – Режим доступу: https://metanit.com/sharp/tutorial/1.2.php.

9. Troelsen A., Japikse P. Pro C# 8 with .NET Core 3: навч. посіб. – Apress, 2020. – 1881 с.

10. Крістіан Нейгел та ін. C# 5.0 та платформа .NET 4.5 для професіоналів = Professional C# 5.0 and .NET 4.5. – 2013. – 1440 с.

11. Скіт Д. C# для професіоналів: тонкості програмування, 3-є вид. Пер. з англ. – Львів: «Вільямс», 2014. – 608 с.

12. Мова програмування C# [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp.

13. Боуман Д., Емерсон С., Дарновський М. Практичний посібник по SQL. – Київ: «Діалектика», 1997.

14. SQL [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/sql/.

15. Desktop Guide (Windows Forms .NET) [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/ruru/dotnet/desktop/winforms/overview/?view=netdesktop-6.0.

16. Git [Електронний ресурс]. – Режим доступу: https://git-scm.com/

17. Інтерфейс додатку. Як правильно створити? [Електронний ресурс]. – Режим доступу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/83/index1.html.

18. Windows Forms [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/getting-started-with-windows-forms?view=netframeworkdesktop-4.8.

ДОДАТОК А

Лістинг програмного забезпечення для бронювання спортивних майданчиків.

class FootballFieldReservation

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyGround
{
   public class FootballFieldReservation
   {
      public int ReservationId { get; set; }
      public string FieldName { get; set; }
      public string Size { get; set; }
      public string Surface { get; set; }
      public string Location { get; set; }
      public decimal RentalCost { get; set; }
      public DateTime ReservationStart { get; set; }
      public DateTime ReservationEnd { get; set; }
   }
}}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

class User

```
namespace MyGround
{
   public class User
   {
      public int Id { get; set; }
      public string Email { get; set; }
      public string Name { get; set; }
      public string Surname { get; set; }
      public string HashPass { get; set; }
      public string Locations { get; set; }
```

```
        public string RoleName { get; set; }
        public string PhoneNumber { get; set; }
        // Конструктор класу User
        public User(int id, string email, string name, string surname, string hashPass, string locations, string roleName,
string phoneNumber)
        {
            Id = id;
            Email = email;
            Name = name;
            Surname = surname;
            HashPass = hashPass;
            Locations = locations;
            RoleName = roleName;
            PhoneNumber = phoneNumber;
        }
    }

}
```

class FootballField

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyGround
{
    public class FootballField
    {
        public int Id { get; set; }
        public string FieldName { get; set; }
        public int OwnerId { get; set; }
        public string Size { get; set; }
        public string Surface { get; set; }
        public string Location { get; set; }
        public decimal RentalCost { get; set; }
        public string City { get; set; } // Нове поле
```

```
        public FootballField(int id, string fieldName, int ownerId, string size, string surface, string location, decimal
rentalCost, string city)
        {
            Id = id;
            FieldName = fieldName;
            OwnerId = ownerId;
            Size = size;
            Surface = surface;
            Location = location;
            RentalCost = rentalCost;
            City = city;
        }
    }

}
```

class DatabaseManager

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.Security.Cryptography;
using BCrypt.Net;


namespace MyGround
{
    public class DatabaseManager
    {
        private readonly string connectionString = "Data Source=DESKTOP-Q0AMUTU;Initial
Catalog=MyGroundDB;Integrated Security=True";

        public DatabaseManager()
        {

        }

        public DataTable GetUsers()
        {
            using (SqlConnection connection = new SqlConnection(connectionString))
```

```
        {
            DataTable dataTable = new DataTable();
            SqlDataAdapter dataAdapter = new SqlDataAdapter("SELECT * FROM Users", connection);
            dataAdapter.Fill(dataTable);
            return dataTable;
        }
    }

    public bool Login(string email, string password)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT HashPass FROM Users WHERE Email = @Email";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Email", email);

            connection.Open();
            var result = command.ExecuteScalar();

            if (result == null)
            {
                // Якщо email не знайдено
                return false;
            }

            string hashedPassword = result.ToString();

            // Перевіряємо, чи пароль відповідає хешу з бази даних
            return BCrypt.Net.BCrypt.Verify(password, hashedPassword);
        }
    }

    public void AddUser(string email, string name, string surname, string password, string locations, string
roleName, string phoneNumber)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // Хешуємо пароль перед збереженням
            string hashedPassword = BCrypt.Net.BCrypt.HashPassword(password);
```

```
            string query = "INSERT INTO Users (Email, Name, Surname, HashPass, Locations, RoleName,
PhoneNumber) VALUES (@Email, @Name, @Surname, @HashPass, @Locations, @RoleName,
@PhoneNumber)";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Email", email);
            command.Parameters.AddWithValue("@Name", name);
            command.Parameters.AddWithValue("@Surname", surname);
            command.Parameters.AddWithValue("@HashPass", hashedPassword);
            command.Parameters.AddWithValue("@Locations", locations);
            command.Parameters.AddWithValue("@RoleName", roleName);
            command.Parameters.AddWithValue("@PhoneNumber", phoneNumber);
            connection.Open();
            command.ExecuteNonQuery();
        }
    }
    public int GetUserIdByEmail(string email)
    {
        int userId = -1; // Значення за замовчуванням, якщо електронна адреса не знайдена

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT id FROM Users WHERE Email = @Email";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Email", email);

            connection.Open();
            var result = command.ExecuteScalar(); // Отримуємо id користувача за його email

            if (result != null && result != DBNull.Value)
            {
                userId = Convert.ToInt32(result);
            }
        }
        return userId;
    }
    public User GetUserById(int userId)
    {
        User user = null;

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT * FROM Users WHERE Id = @Id";
```

```
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Id", userId);

            connection.Open();
            SqlDataReader reader = command.ExecuteReader();

            if (reader.Read())
            {
                // Створення об'єкту User з результатів запиту до бази даних
                user = new User(
                    Convert.ToInt32(reader["Id"]),
                    Convert.ToString(reader["Email"]),
                    Convert.ToString(reader["Name"]),
                    Convert.ToString(reader["Surname"]),
                    Convert.ToString(reader["HashPass"]),
                    Convert.ToString(reader["Locations"]),
                    Convert.ToString(reader["RoleName"]),
                    Convert.ToString(reader["PhoneNumber"])
                );
            }

            reader.Close();
        }

        return user;
    }
    public void AddFootballField(string fieldName, int ownerId, string size, string surface, string location, decimal
rentalCost, string city)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = @"INSERT INTO FootballFields (FieldName, OwnerId, Size, Surface, Location,
RentalCost, City)
                    VALUES (@FieldName, @OwnerId, @Size, @Surface, @Location, @RentalCost, @City)";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@FieldName", fieldName);
            command.Parameters.AddWithValue("@OwnerId", ownerId);
            command.Parameters.AddWithValue("@Size", size);
            command.Parameters.AddWithValue("@Surface", surface);
            command.Parameters.AddWithValue("@Location", location);
            command.Parameters.AddWithValue("@RentalCost", rentalCost);
            command.Parameters.AddWithValue("@City", city);
```

```
            connection.Open();
            command.ExecuteNonQuery();
        }
    }
    public List<FootballField> GetFootballFieldsByOwnerId(int ownerId)
    {
        List<FootballField> footballFields = new List<FootballField>();

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT * FROM FootballFields WHERE OwnerId = @OwnerId";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@OwnerId", ownerId);

            connection.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                // Створення об'єкта FootballField з результатів запиту до бази даних
                FootballField field = new FootballField(
                    Convert.ToInt32(reader["Id"]),
                    Convert.ToString(reader["FieldName"]),
                    Convert.ToInt32(reader["OwnerId"]),
                    Convert.ToString(reader["Size"]),
                    Convert.ToString(reader["Surface"]),
                    Convert.ToString(reader["Location"]),
                    Convert.ToDecimal(reader["RentalCost"]),
                    Convert.ToString(reader["City"])
                );

                footballFields.Add(field);
            }

            reader.Close();
        }

        return footballFields;
    }
    public decimal GetFieldCostById(int fieldId)
    {
```

```
        decimal fieldCost = 0;

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT RentalCost FROM FootballFields WHERE Id = @FieldId";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@FieldId", fieldId);

            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            if (reader.Read())
            {
                fieldCost = reader.GetDecimal(0); // Отримання вартості поля з першого стовпця результату запиту
            }
        }

        return fieldCost;
    }
    public void AddReservation(int fieldId, int ownerId, DateTime startDateTime, DateTime endDateTime,
decimal rentalCost)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "INSERT INTO Reservations (FieldId, OwnerId, ReservationStart, ReservationEnd,
RentalCost) VALUES (@FieldId, @OwnerId, @StartDateTime, @EndDateTime, @RentalCost)";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@FieldId", fieldId);
            command.Parameters.AddWithValue("@OwnerId", ownerId);
            command.Parameters.AddWithValue("@StartDateTime", startDateTime);
            command.Parameters.AddWithValue("@EndDateTime", endDateTime);
            command.Parameters.AddWithValue("@RentalCost", rentalCost);

            connection.Open();
            command.ExecuteNonQuery();
        }
    }
    public bool IsEmailExists(string email)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT COUNT(*) FROM Users WHERE Email = @Email";
            SqlCommand command = new SqlCommand(query, connection);
```

```
            command.Parameters.AddWithValue("@Email", email);

            connection.Open();
            int count = (int)command.ExecuteScalar();
            return count > 0;
        }
    }
    public bool VerifyPassword(int userId, string password)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "SELECT HashPass FROM Users WHERE Id = @UserId";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@UserId", userId);

            connection.Open();
            string hashedPassword = (string)command.ExecuteScalar();

            return BCrypt.Net.BCrypt.Verify(password, hashedPassword);
        }
    }
    public void UpdateUser(User user)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = @"UPDATE Users
                    SET Name = @Name, Surname = @Surname, PhoneNumber = @PhoneNumber, Locations =
@Locations
                    WHERE Id = @UserId";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Name", user.Name);
            command.Parameters.AddWithValue("@Surname", user.Surname);
            command.Parameters.AddWithValue("@PhoneNumber", user.PhoneNumber);
            command.Parameters.AddWithValue("@Locations", user.Locations);
            command.Parameters.AddWithValue("@UserId", user.Id);

            connection.Open();
            command.ExecuteNonQuery();
        }
    }
    public List<string> GetUniqueCities()
    {
```

```
    List<string> cities = new List<string>();

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string query = "SELECT DISTINCT City FROM FootballFields";
        SqlCommand command = new SqlCommand(query, connection);

        connection.Open();
        SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            cities.Add(reader["City"].ToString());
        }

        reader.Close();
    }

    return cities;
}
public List<FootballFieldReservation> GetAvailableFields(string city, DateTime date)
{
    List<FootballFieldReservation> availableFields = new List<FootballFieldReservation>();

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string query = @"
    SELECT
        r.Id AS ReservationId,
        ff.FieldName,
        ff.Size,
        ff.Surface,
        ff.Location,
        ff.RentalCost,
        r.ReservationStart,
        r.ReservationEnd
    FROM
        FootballFields ff
    JOIN
        Reservations r ON ff.Id = r.FieldId
    WHERE
        ff.City = @City AND
```

```
        r.ReservationStart <= @Date AND
        r.ReservationEnd >= @Date AND
        r.AcceptedByUserId IS NULL";

        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@City", city);
        command.Parameters.AddWithValue("@Date", date);

        connection.Open();
        SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            FootballFieldReservation field = new FootballFieldReservation
            {
                ReservationId = Convert.ToInt32(reader["ReservationId"]),
                FieldName = reader["FieldName"].ToString(),
                Size = reader["Size"].ToString(),
                Surface = reader["Surface"].ToString(),
                Location = reader["Location"].ToString(),
                RentalCost = Convert.ToDecimal(reader["RentalCost"]),
                ReservationStart = Convert.ToDateTime(reader["ReservationStart"]),
                ReservationEnd = Convert.ToDateTime(reader["ReservationEnd"])
            };

            availableFields.Add(field);
        }

        reader.Close();
    }

    return availableFields;
}

public DataTable GetAllUsersExceptCurrent(int currentUserId)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string query = @"SELECT Email, Name, Surname, RoleName
                         FROM Users
                         WHERE Id <> @CurrentUserId";
        SqlCommand command = new SqlCommand(query, connection);
```

```
            command.Parameters.AddWithValue("@CurrentUserId", currentUserId);

            SqlDataAdapter dataAdapter = new SqlDataAdapter(command);
            DataTable dataTable = new DataTable();
            dataAdapter.Fill(dataTable);

            return dataTable;
        }
    }
  }
}
```

class MainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyGround
{
  public partial class MainForm : Form
  {
    private User user;
    private readonly int idUser;
    private DatabaseManager dbManager;
    public MainForm()
    {
      InitializeComponent();
      dbManager = new DatabaseManager();
    }
    public MainForm(int idUser)
    {
      InitializeComponent();
      this.idUser = idUser;
      dbManager = new DatabaseManager();
      user = dbManager.GetUserById(idUser);
      lblName.Text=user.Name+" "+user.Surname;
      if (user.RoleName == "user")
```

```
    {
        UserLogin();
    }
    if (user.RoleName == "manager")
    {

    }
    if (user.RoleName == "admin")
    {

    }
}
private void UserLogin()
{
    button6.Visible = false;
    button4.Visible = false;
}
private void MainForm_Load(object sender, EventArgs e)
{

}

private void button5_Click(object sender, EventArgs e)
{
    this.Close();
}

private void btnLogOut_Click(object sender, EventArgs e)
{
    Form1 form=new Form1();
    this.Hide();
    form.ShowDialog();

    // Після закриття головного вікна, показуємо знову форму входу
    this.Close();
}
private void addUserContol(UserControl userControl)
{
    userControl.Dock = DockStyle.Fill;
    panel3.Controls.Add(userControl);
    userControl.BringToFront();
}
```

```
private void button1_Click(object sender, EventArgs e)
{
   UCMyPage uCMyPage= new UCMyPage(user);
   addUserContol(uCMyPage);
}

private void button4_Click(object sender, EventArgs e)
{
   UCMyFields1 uCMyFields1 = new UCMyFields1(user);
   addUserContol(uCMyFields1);
}

private void button6_Click(object sender, EventArgs e)
{
   UCPost uCPost = new UCPost(user);
   addUserContol(uCPost);
}

private void label4_Click(object sender, EventArgs e)
{
   this.Close();
}

private void lblName_Click(object sender, EventArgs e)
{

}

private void panel3_Paint(object sender, PaintEventArgs e)
{

}

private void button2_Click(object sender, EventArgs e)
{
   UCSearch search = new UCSearch(user);
   addUserContol(search);
}

private void button3_Click(object sender, EventArgs e)
{
   UCHistory search = new UCHistory(user);
```

```
            addUserContol(search);
        }

        private void button7_Click(object sender, EventArgs e)
        {
            UCAccess access = new UCAccess(user);
            addUserContol(access);
        }
    }
}
```

class Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyGround
{
    public partial class Form1 : Form
    {
        private DatabaseManager dbManager;
        public Form1()
        {
            InitializeComponent();
            dbManager = new DatabaseManager();

        }

        private void lblRegistration_Click(object sender, EventArgs e)
        {
            Registration registration= new Registration();
            registration.ShowDialog();
        }


        private void lblLogin_Click(object sender, EventArgs e)
        {
```

```
}

private void Form1_Load(object sender, EventArgs e)
{

}

private void btnLogin_Click(object sender, EventArgs e)
{
    // Отримання введеного email та пароля з текстових полів
    string email = textBoxEmail.Text;
    string password = textBoxPassword.Text;

    // Перевірка входу в додаток
    bool loginSuccessful = dbManager.Login(email, password);

    // Перевірка результату входу
    if (loginSuccessful)
    {

        int idUser = dbManager.GetUserIdByEmail(email);
        // Якщо вхід успішний, відобразіть головне вікно додатка та закрийте поточне вікно входу
        // Створення екземпляра головного вікна
        MainForm mainForm = new MainForm(idUser); // Підставте власний клас головного вікна
        // Закриття поточної форми входу і відображення головного вікна
        this.Hide();
        mainForm.ShowDialog();

        // Після закриття головного вікна, показуємо знову форму входу
        this.Close();
    }
    else
    {
        // Якщо вхід невдалий, відобразіть повідомлення про помилку
        MessageBox.Show("Невірний email або пароль. Спробуйте ще раз.");
    }


}

private void panel1_Paint(object sender, PaintEventArgs e)
```

```
        {

        }

        private void label4_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```