МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Веб-платформа для бронювання послуг**

Виконав: студент 4 курсу, групи 6.1212-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
_____ Султан В. Д
(ініціали та прізвище)
Керівник доцент, к.т.н., доцент О. М. Міхайлуца
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ „Діскус”
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
_____Р.О.Лютий

Запоріжжя
2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна КРИТСЬКА
“_01_” _березня_ 2026_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

_____Султану Владиславу Дмитровичу
(прізвище, ім’я, по батькові)
1.Тема роботи_Веб-платформа для бронювання послуг.

керівник роботи_Міхайлуца Олена Миколаївна, доцент, к.т.н._____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 23.12.2025 р. № 1830-с___

2. Строк подання студентом кваліфікаційної роботи_02.06.2026
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми збору та аналізу даних з веб-сторінок;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) _____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Призвіще ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.03.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	3.03.2026	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	4.03.2026	виконано
3	Аналіз існуючих методів рішення	1.04.2026	виконано
4	Дослідження засобів реалізації Веб-застосунку	2.04.2026-9.04.2026	виконано
5	Дослідження засобів реалізації Бази даних	10.04.2026-17.04.2026	виконано
6	Проектування архітектури Веб-застосунку	18.04.2026-22.04.2026	виконано
7	Програмна реалізація Веб-застосунку	23.04.2026-20.05.2026	виконано
8	Програмна реалізація системи бронювання	21.05.2026-24.05.2026	виконано
9	Адаптація інтерфейсу під мобільні пристрої	25.05.2026-27.05.2026	виконано
10	Тестування та перевірка працездатності Веб-застосунку	28.05.2026	виконано
11	Оформлення пояснювальної записки	29.05.2026-01.06.2026	виконано
12	Представлення проміжних результатів керівнику	02.06.2026	виконано
13	Оформлення презентації, отримання рецензій	12.06.2026-18.06.2026	

Студент _____ Султан В.Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Міхайлуца О.М.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено
Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 81

Рисунків — 14

Джерел —11

Султан В.Д. Веб-платформа для бронювання послуг: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник О. М. Міхайлуца. Запоріжжя : ЗНУ, 2026. 81 с.

Мета полягає у розробці веб-застосунку, який забезпечує зручне та ефективне бронювання спортивних приміщень і полів із використанням сучасних веб-технологій.

У процесі розробки була розглянута проблема організації бронювання спортивних об’єктів та автоматизації взаємодії між користувачами і власниками ресурсів. Проведено аналіз існуючих програмних рішень, визначено їх переваги та недоліки. У результаті був розроблений веб-застосунок, що реалізує функціональність перегляду доступних об’єктів, вибору дати і часу, оформлення бронювання, управління замовленнями та взаємодії з користувачем. Клієнтська частина застосунку реалізована з використанням бібліотеки React, серверна частина — на основі Node.js, з використанням СУБД PostgreSQL для зберігання даних. Система побудована за клієнт-серверною архітектурою з використанням REST-підходу для обміну даними.

Розроблений застосунок забезпечує автоматизацію процесу бронювання, підвищує ефективність використання спортивних об’єктів та покращує якість обслуговування користувачів.

Ключові слова: *веб-застосунок, бронювання, спортивні приміщення, клієнт-серверна архітектура, React, Node.js, PostgreSQL, REST, інтерфейс користувача, автоматизація.*

ABSTRACT

Pages — 81

Figures — 14

Source —11

Sultan V.D. Web platform for booking services: bachelor's qualification work in specialty 121 "Software Engineering" / scientific supervisor O. M. Mikhaylutsa. Zaporizhzhia: ZNU, 2026. 81 p.

The goal is to develop a web application that provides convenient and effective booking of sports facilities and fields using modern web technologies.

During the development process, the problem of organizing the booking of sports facilities and automating the interaction between users and resource owners was considered. An analysis of existing software solutions was conducted, their advantages and disadvantages were identified. As a result, a web application was developed that implements the functionality of viewing available facilities, selecting a date and time, making a reservation, managing orders, and interacting with the user. The client part of the application is implemented using the React library, the server part is based on Node.js, using the PostgreSQL DBMS for data storage. The system is built on a client-server architecture using the REST approach for data exchange.

The developed application provides automation of the booking process, increases the efficiency of using sports facilities and improves the quality of user service.

Keywords: web application, booking, sports facilities, client-server architecture, React, Node.js, PostgreSQL, REST, user interface, automation.

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Огляд літературних джерел 11
1.2 Аналіз програмного забезпечення 12
1.2.1 OLX 12
1.2.2 Booking.com 13
1.2.3 Prostir Podiy 14
1.2.4 Kamon 14
1.2.5 Bookimed 14
1.3 Постановка завдання 16
1.4 Висновки до розділу 1 17
2 ВИМОГИ ДО ОТОЧЕННЯ 18
2.1 Аналіз сучасних технологій для розробки веб-застосунків 18
2.2 Аналіз інструментів якості коду та тестування 21
2.3 Аналіз вимог до користувачів 23
2.4 Аналіз вимог до апаратного та програмного забезпечення 25
2.5 Висновки до розділу 2 25
3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ СПОРТИВНИХ МАЙДАНЧИКІВ 27
3.1 Опис предметної області 27
3.2 Архітектура системи 28
3.2.1 Схема бази даних 30
3.2.2 Система безпеки: Row Level Security 34
3.3 Архітектурні патерни та вимоги до системи 35
3.3.1 Декларативний підхід, JSX та управління CSS-класами 36
3.3.2 Анімаційна система, стани екрану та оптимізація продуктивності . 38
3.3.3 Управління формами, Optimistic UI та Progressive Enhancement 43

3.3.4 Специфікація вимог до системи 46
3.3.5 Бібліотека підпрограм (класів) 47
3.4 Програмна реалізація функціональних модулів 48
3.5.1 Автентифікація та управління сесіями 49
3.4.2 Система фільтрації та пошуку 54
3.4.3 Система бронювання 57
3.4.4 Управління фотографіями та форма майданчика 61
3.4.5 Особистий кабінет та статистика користувача 65
3.4.6 Дизайн-система та компонентна бібліотека 67
3.4.7 Кросс-вкладкова синхронізація даних 68
3.5 Інтерфейс, реалізація та тестування 70
3.5.1 Вимоги до інтерфейсу 70
3.5.2 Реалізація 74
3.5.3 Апробація 75
3.6 Висновки до розділу 3 77
ВИСНОВКИ 79
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 81

ВСТУП

Актуальність теми

Сучасний розвиток інформаційних технологій та цифровізація сфери послуг сприяють широкому впровадженню веб-застосунків для автоматизації різних бізнес-процесів. Однією з актуальних задач є організація ефективного бронювання спортивних приміщень та полів, таких як стадіони, тренажерні зали та ігрові майданчики. Традиційні способи організації бронювання, що базуються на телефонних дзвінках або ручному веденні записів, є неефективними, оскільки не забезпечують оперативного доступу до інформації, автоматичного контролю зайнятості та зручної взаємодії між користувачами і власниками ресурсів.

Аналіз існуючих програмних систем показує, що на ринку відсутнє універсальне рішення, яке повною мірою задовольняє потреби предметної області. Так, платформи оголошень, такі як OLX, забезпечують можливість пошуку пропозицій, однак не мають функціональності бронювання, календаря зайнятості та автоматичного підтвердження. Сервіси типу Booking.com реалізують повноцінні механізми бронювання, включаючи календар доступності та онлайн-оплату, проте орієнтовані переважно на туристичну сферу та не враховують специфіку спортивних об’єктів. Платформи, подібні до Prostir Podiy, частково вирішують задачу оренди приміщень, але не забезпечують повної автоматизації процесу бронювання. Сервіси Kamon та Bookimed також не є універсальними рішеннями, оскільки або не підтримують бронювання часу, або мають вузьку спеціалізацію.

Таким чином, актуальність теми дипломної роботи полягає у необхідності створення спеціалізованого веб-застосунку, який поєднує функціональні можливості сучасних систем бронювання з урахуванням особливостей спортивної інфраструктури, забезпечує автоматизацію процесів та підвищує ефективність використання ресурсів.

Мета дослідження

Метою даної дипломної роботи є розробка веб-застосунку для бронювання спортивних приміщень та полів, який забезпечує автоматизацію процесу вибору об’єктів, резервування часу, управління бронюваннями та взаємодію користувачів із системою на основі сучасних веб-технологій.

Завдання дослідження

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) провести аналіз предметної області та існуючих програмних систем-аналогів;
2) визначити функціональні та нефункціональні вимоги до веб-застосунку;
3) розробити архітектуру системи на основі клієнт-серверного підходу;
4) спроєктувати структуру бази даних для зберігання інформації;
5) реалізувати клієнтську частину застосунку з використанням сучасних технологій;
6) реалізувати серверну частину із забезпеченням обробки запитів;
7) розробити механізми бронювання, календар зайнятості та управління даними;
8) забезпечити захист даних та реалізувати механізми автентифікації;
9) провести тестування та оцінку ефективності роботи системи.

Об’єкт дослідження

Об’єктом дослідження є процес організації бронювання спортивних приміщень та полів.

Предмет дослідження

Предметом дослідження є методи, моделі та програмні засоби розробки веб-застосунків для автоматизації процесу бронювання.

Методи дослідження

У процесі виконання роботи були використані методи аналізу предметної області та існуючих програмних рішень, порівняльний аналіз систем-аналогів, методи проєктування програмного забезпечення, моделювання архітектури інформаційних систем, а також сучасні методи розробки веб-застосунків.

Практичне значення отриманих результатів

Практичне значення роботи полягає у створенні веб-застосунку, який може бути використаний для організації бронювання спортивних об’єктів у реальних умовах. Запропоноване рішення дозволяє автоматизувати процес взаємодії між користувачами та власниками ресурсів, підвищити ефективність використання спортивної інфраструктури, зменшити кількість помилок при бронюванні та забезпечити зручний доступ до інформації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

У процесі виконання дипломної роботи було проаналізовано сучасні інформаційні джерела, що стосуються розробки веб-застосунків, систем бронювання та принципів проєктування користувацького інтерфейсу.

У сучасних умовах значна увага приділяється створенню веб-застосунків для організації процесів бронювання послуг. Зокрема, у роботах, присвячених розробці подібних систем, відзначається, що основною метою таких застосунків є спрощення взаємодії між користувачем та постачальником послуг, а також централізація управління бронюваннями [1]. Подібні системи дозволяють користувачам швидко знаходити необхідні послуги, обирати зручний час та здійснювати резервування через єдиний інтерфейс.

Особливу роль у розробці веб-застосунків відіграє проєктування користувацького інтерфейсу та забезпечення зручності використання системи. Як зазначається у джерелах, присвячених UX/UI-дизайну, ефективність роботи веб-застосунку значною мірою залежить від того, наскільки інтуїтивно зрозумілим є процес взаємодії користувача із системою [2]. Використання принципів орієнтації на користувача, адаптивного дизайну та оптимізації навігації дозволяє підвищити рівень задоволеності користувачів та зменшити кількість помилок під час бронювання.

Крім того, у сучасних дослідженнях розглядаються особливості побудови складних систем бронювання, які повинні обробляти велику кількість даних та забезпечувати стабільну роботу при значному навантаженні [3]. Такі системи потребують чіткої організації архітектури, ефективної взаємодії між клієнтською та серверною частинами, а також можливості масштабування.

Таким чином, проведений аналіз показує, що сучасні веб-застосунки для бронювання повинні поєднувати зручний інтерфейс, ефективну архітектуру та

високу продуктивність. Разом з тим, більшість існуючих рішень орієнтовані на окремі сфери застосування і не враховують специфіку бронювання спортивних приміщень, що підтверджує доцільність розробки спеціалізованого веб-застосунку.

1.2 Аналіз програмного забезпечення

У ході аналізу предметної галузі — веб-застосунок для бронювання спортивних приміщень та полів — було розглянуто п'ять існуючих програмних систем, що представляють різні підходи до вирішення схожих задач: від універсальних платформ оголошень до спеціалізованих систем запису на послуги. Метою аналізу є виявлення функціональних прогалин у існуючих рішеннях, формування вимог до власної розробки та обґрунтування її доцільності.

1.2.1 OLX

OLX — міжнародна платформа оголошень, що входить до холдингу OLX Group, дочірнього підрозділу Prosus N.V. (Нідерланди). Платформа функціонує у понад 45 країнах та є одним з найбільших агрегаторів оголошень купівлі-продажу та оренди в Україні та країнах Центральної і Східної Європи.

Основна функціональність OLX включає: розміщення структурованих оголошень з фотографіями, описом та ціною; ієрархічну категоризацію оголошень; систему параметричної фільтрації та повнотекстового пошуку; вбудований чат між продавцем та покупцем; систему рейтингів та відгуків продавців; мобільні додатки для iOS та Android.

З точки зору цільової предметної галузі OLX може використовуватися для пошуку об'єктів оренди спортивних приміщень. Проте ключова проблема полягає в тому, що OLX є платформою оголошень, а не системою бронювання: відсутній календар доступності, неможливе онлайн-підтвердження резервування конкретного слоту, необхідний телефонний або чатовий контакт

для домовленості. Такий підхід кардинально відрізняється від принципу real-time booking, реалізованого у розробленому застосунку.

Серед переваг OLX варто відзначити велику аудиторію користувачів, розвинену систему довіри (верифіковані продавці) та україномовний інтерфейс. Однак відсутність вбудованої системи бронювання, календаря зайнятості та автоматизації підтвердження послуг робить платформу лише допоміжним інструментом для пошуку, але не для безпосереднього резервування.

1.2.2 Booking.com

Booking.com — один з найбільших у світі онлайн-сервісів бронювання подорожей та житла, що є дочірньою компанією Booking Holdings Inc. (США, NASDAQ: BKNG). Щодня через платформу здійснюється більше 1,5 мільйона ночівель, а каталог нараховує понад 28 мільйонів варіантів розміщення у 230 країнах.

Функціональність Booking.com охоплює: повноцінну систему онлайн-бронювання з реальною перевіркою доступності; динамічний календар доступності, що оновлюється в режимі реального часу; декілька варіантів оплати; розгалужену систему верифікованих відгуків; гнучку систему тарифів та умов скасування; програму лояльності Genius. Саме ця платформа демонструє еталонну реалізацію ключових функцій бронювання і слугувала методологічним орієнтиром при проектуванні застосунку SportFields.

Водночас Booking.com має принципові обмеження для адаптації до спортивної галузі: платформа розрахована на unit-based pricing (бронювання номерів), тоді як спортивні майданчики потребують time-slot based booking з погодинними інтервалами; відсутні поля для специфічних параметрів (вид покриття, тип спорту, довжина корту); значні комісії для партнерів (15–25%) є неприйнятними для малого бізнесу.

1.2.3 Prostir Podiy

Prostir Podiy — українська онлайн-платформа для пошуку та оренди локацій для проведення заходів, конференцій, корпоративів та творчих зустрічей. Функціональний набір включає: структурований каталог приміщень з детальними описами та фотогалереями; систему фільтрації за типом заходу, місткістю та містом; форму подачі заявки на бронювання; механізм комунікації між орендарем та власником.

Ключова проблема платформи полягає у відсутності автоматизованого підтвердження бронювання: подача заявки не гарантує резервування — власник повинен самостійно відреагувати на запит, після чого розпочинається ручна комунікація щодо оплати. Також відсутній механізм онлайн-оплати в момент бронювання. Таким чином, платформа функціонує радше як агрегатор контактів, а не повноцінна система бронювання.

1.2.4 Kamon

Kamon — онлайн-сервіс пошуку та замовлення послуг різних категорій на українському ринку, аналогічний міжнародним платформам Thumbtack (США) та Bark.com (Великобританія). Сервіс надає каталог виконавців з описом послуг та відгуками, систему пошуку за категорією та географічною ознакою.

Для завдань бронювання спортивних об'єктів Kamon принципово непридатний: відсутній будь-який механізм управління часовими слотами, немає інтеграції з платіжними системами, а процес замовлення послуги є повністю неавтоматизованим. Платформа може слугувати лише для первинного пошуку контакту, але не для резервування.

1.2.5 Bookimed

Bookimed — міжнародна платформа для онлайн-запису та бронювання медичних послуг у понад 60 країнах. Цікава для порівняльного аналізу тим, що реалізує повноцінну логіку тайм-слот бронювання, методологічно близьку

до завдань спортивного резервування. Платформа реалізує вибір дати та конкретного часового вікна з перевіркою доступності; автоматичне підтвердження запису; особистий кабінет з історією звернень; систему рейтингів та відгуків.

Обмеження Bookimed у контексті спортивної галузі є очевидними: вузька медична спеціалізація не дозволяє адаптувати платформу під управління майданчиками; відсутня концепція бронювання фізичного об'єкту (корту, поля) на відміну від бронювання часу спеціаліста; вартість розгортання власного екземпляру системи є непомірно великою для малого бізнесу.

На основі проведеного аналізу п'яти програмних систем складено зведену порівняльну таблицю за ключовими функціональними критеріями (таблиця 1.2.5).

Таблиця *1.2.5 — Порівняльний аналіз існуючих програмних систем (+є, –немає, ±частково)*

Властивість	**OLX**	**Booking.com**	**Prostir Podiy**	**Kamon**	**Bookimed**
Онлайн-бронювання	–	+	±	–	+
Календар зайнятості	–	+	–	–	+
Онлайн-оплата	–	+	–	–	±
Рейтинг / відгуки	+	+	±	+	+
Каталог об'єктів	+	+	+	+	+
Автоматизація	–	+	–	–	+
Спеціалізація під спорт	–	–	–	–	–
Український інтерфейс	+	+	+	+	+

1.3 Постановка завдання

На основі проведеного аналізу можна зробити висновок, що жодна з розглянутих систем не орієнтована безпосередньо на бронювання спортивних приміщень та полів. Найближчими за рівнем автоматизації є Booking.com та Bookimed — проте перша орієнтована виключно на готельну галузь, а друга — на медичну. Платформи OLX, Prostir Podiy та Kamon функціонують радше як агрегатори оголошень або каталоги виконавців, ніж як системи бронювання у повному розумінні цього терміну.

Виявлена функціональна прогалина підтверджує доцільність та актуальність розробки власного веб-застосунку SportFields. Розроблюваний застосунок повинен поєднати повноцінну систему онлайн-бронювання часових слотів з реальною перевіркою доступності; спеціалізований каталог спортивних об'єктів з деталізованою інформацією про покриття, вид спорту та зручності; систему ролей для розмежування прав гравців та власників майданчиків; надійний захист персональних даних на рівні бази даних.

Мета кваліфікаційної роботи — розробка веб-застосунку для пошуку та бронювання спортивних майданчиків на основі сучасного стеку TypeScript + React + Supabase (PostgreSQL), що усуває виявлені недоліки існуючих рішень та відповідає сформованим функціональним і нефункціональним вимогам.

Постановка завдання дослідження включає такі послідовні етапи:

1. Аналіз предметної галузі та огляд літературних джерел у сфері веб-розробки систем бронювання.
2. Порівняльний аналіз існуючих програмних аналогів та формування вимог до власного рішення.
3. Вибір технологічного стеку та обґрунтування архітектурних рішень.
4. Проектування схеми бази даних, системи ролей та механізмів захисту даних.
5. Розробка клієнтської частини застосунку з адаптивним інтерфейсом та дизайн-системою.

6. Реалізація серверної інфраструктури на основі Backend-as-a-Service (Supabase).
7. Тестування реалізованого застосунку на модульному та E2E рівнях.
8. Апробація отриманих результатів та оцінка відповідності встановленим вимогам.

1.4 Висновки до розділу 1

- Проведено огляд літературних джерел у галузі реактивної веб-розробки, управління серверним станом (TanStack Query), безпеки баз даних (Row Level Security) та стандартів доступності інтерфейсів (WAI-ARIA).
- Виконано порівняльний аналіз п'яти програмних аналогів: OLX, Booking.com, Prostir Podiy, Kamon, Bookimed. Встановлено переваги та недоліки кожної системи щодо задач предметної галузі.
- Встановлено, що жодна з розглянутих систем не задовольняє повний спектр вимог предметної галузі бронювання спортивних об'єктів.
- Сформовано мету та постановку завдання кваліфікаційної роботи; визначено вісім послідовних етапів дослідження.

2 ВИМОГИ ДО ОТОЧЕННЯ

2.1 Аналіз сучасних технологій для розробки веб-застосунків

Сучасна розробка клієнт-серверних веб-застосунків спирається на широкий стек технологій. Нижче наведено детальний огляд обраних технологій та обґрунтування їх вибору у контексті завдань розроблюваного застосунку.

TypeScript 5 є надмножиною JavaScript, що додає статичну типізацію, інтерфейси, generics та декоратори. Головна перевага у контексті даного проєкту — можливість автоматичної генерації типів зі схеми бази даних Supabase: команда supabase gen types typescript створює файл типів, що точно відображає структуру таблиць PostgreSQL. Це забезпечує наскрізну типову безпеку від рівня СУБД до компонентів UI: якщо змінюється схема бази даних, TypeScript-компілятор негайно виявить невідповідності у клієнтському коді ще до запуску застосунку, що суттєво знижує ймовірність runtime-помилок.

React 18 є декларативною бібліотекою побудови інтерфейсів від Meta Platforms Inc. Ключові нововведення версії 18, що використовуються у проєкті: Concurrent Rendering — новий механізм рендерингу, що дозволяє браузеру переривати тривалі операції відрисовки на користь більш пріоритетних завдань (наприклад, реакції на введення користувача); автоматичне батчування (automatic batching) оновлень стану — декілька setState-викликів в рамках одного обробника події об'єднуються в один рендер, що скорочує кількість перерисовок DOM та підвищує продуктивність при складних взаємодіях.

Vite 5 у поєднанні з компілятором SWC (Speedy Web Compiler, написаний на Rust) забезпечує збірку, яка є на порядок швидшою за попередні інструменти на базі Webpack або Babel. При холодному старті Vite не збирає весь застосунок — він обслуговує ES-модулі напряму через браузер, що робить Hot Module Replacement практично миттєвим (до 50 мс). Для

production-збірки Vite використовує Rollup із tree-shaking, що автоматично виключає невикористаний код та мінімізує розмір фінального бандлу.

Tailwind CSS v3 реалізує підхід utility-first: замість написання окремих CSS-файлів з семантичними класами, стилі описуються безпосередньо у JSX-атрибуті className за допомогою атомарних утилітарних класів. У даному проєкті дизайн-система реалізована через CSS Custom Properties у форматі HSL у файлі index.css (токени --brand-green, --background, --foreground та інші), які Tailwind використовує через конфігурацію tailwind.config.ts. Такий підхід забезпечує підтримку темної теми через клас .dark без жодних змін у компонентах.

Supabase — open-source Backend-as-a-Service платформа, побудована на PostgreSQL. Вибір Supabase замість власного Node.js/Express сервера обґрунтований: повноцінна реляційна база даних PostgreSQL з підтримкою SQL-міграцій, тригерів, функцій та Row Level Security; вбудована система автентифікації (Supabase Auth) з підтримкою email/password та видачею підписаних JWT-токенів; автоматично згенерований PostgREST API, що дозволяє звертатися до бази даних з типізованого клієнтського коду; рівень Storage для зберігання файлів; Realtime-підписки через WebSocket.

React Router DOM v6 забезпечує декларативну маршрутизацію у SPA. Версія 6 принесла суттєве переосмислення API: вкладені маршрути описуються через JSX-дерево з Outlet-компонентами; хук useSearchParams надає реактивний доступ до query parameters URL, що є ключовим для реалізації патерну URL-as-State у системі фільтрів; хук useNavigate забезпечує програмну навігацію з підтримкою replace-режиму для захищених маршрутів.

TanStack Query v5 (React Query) вирішує проблему управління серверним станом — категорії стану, що кардинально відрізняється від локального UI-стану тим, що є асинхронним, потенційно застарілим та спільним між компонентами. Бібліотека реалізує автоматичне кешування результатів запитів з налаштовуваними стратегіями інвалідації; фонове оновлення (background refetching) при поверненні до вкладки або відновленні

мережевого з'єднання; декларативні стани isPending, isError, isSuccess у кожному компоненті-споживачі.

React Hook Form у поєднанні з Zod реалізує підхід до управління формами з мінімальною кількістю повторних рендерів. На відміну від controlled components (де кожен keystroke спричиняє рендер), React Hook Form використовує uncontrolled inputs та реєстрацію полів через ref. Zod-схема є єдиним джерелом правди (single source of truth) для валідаційних правил, TypeScript-типів вхідних даних та повідомлень про помилки — це виключає розбіжності між UI-валідацією та типами.

Повний технологічний стек застосунку SportFields з версіями та призначенням кожного компонента наведено у таблиці 2.1.

Таблиця *2.1 — Технологічний стек застосунку*

Технологія / Бібліотека	**Версія**	**Призначення в проєкті**
TypeScript	5.x	Основна мова; статична типізація; автогенерація типів БД
React	18.x	Декларативний UI; Concurrent Rendering; хуки
Vite + SWC	5.x	Збірка; HMR < 50ms; production-бандл через Rollup
Tailwind CSS	3.x	Utility-first CSS; дизайн-токени HSL; темна тема
shadcn/ui + Radix UI	актуальна	ARIA-сумісні компоненти UI; клавіатурна навігація
Lucide React	актуальна	Векторна SVG-іконографія у вигляді React-компонентів
React Router DOM	6.x	SPA-маршрутизація; захищені маршрути; URL-as-State
TanStack Query	5.x	Кешування серверного стану; background refetching
React Hook Form + Zod	актуальна	Форми з мінімумом ре-рендерів; схемна валідація

Технологія / Бібліотека	Версія	Призначення в проєкті
React Day Picker	актуальна	Інтерактивний календарний віджет бронювання
Sonner	актуальна	Non-blocking toast-сповіщення про результат операцій
Supabase (PostgreSQL 15)	2.x	BaaS: JWT-авт., БД, PostgREST API, Storage, Realtime
Vitest	актуальна	Модульне тестування хуків та утилітарних функцій
Playwright	актуальна	E2E тестування у Chromium, Firefox, WebKit

2.2 Аналіз інструментів якості коду та тестування

Забезпечення якості кодової бази у сучасних веб-проєктах потребує комплексного застосування інструментів статичного аналізу, автоматизованого форматування та тестування на різних рівнях. У даному підрозділі розглянуто інструменти, що використовуються у проєкті SportFields для підтримки єдиного стилю коду та раннього виявлення помилок.

ESLint 9 — інструмент статичного аналізу JavaScript та TypeScript коду. На відміну від TypeScript-компілятора, який перевіряє типову коректність, ESLint перевіряє стиль, логічні антипатерни та потенційні помилки часу виконання, що не відловлюються типами. У проєкті ESLint налаштовано через файл eslint.config.js з правилами для React Hooks (react-hooks/rules-of-hooks, react-hooks/exhaustive-deps) та TypeScript (@typescript-eslint). Правило exhaustive-deps є особливо важливим: воно попереджає про неповні масиви залежностей у useEffect та useMemo, що є частою причиною складних для відлагодження багів із застарілим станом (stale closure).

Інтеграція ESLint у процес розробки відбувається на двох рівнях: у реальному часі через плагін редактора коду (VS Code ESLint extension), що підкреслює проблемні рядки безпосередньо в редакторі; та автоматично при

збірці через Vite plugin, що перериває production-збірку при наявності критичних помилок. Така дворівнева інтеграція гарантує, що код з порушеннями не потрапить у репозиторій.

Vitest — фреймворк модульного тестування, нативно інтегрований з Vite. На відміну від Jest (стандарт попереднього покоління), Vitest використовує ту саму конфігурацію збірки, що й production-код (vite.config.ts), що усуває розбіжності між тестовим і бойовим середовищами. Vitest підтримує ES-модулі без додаткового трансформування, що робить тести значно швидшими. Ключові переваги для даного проєкту: вбудований jsdom для тестування DOM-взаємодій; сумісність з API Jest (describe, it, expect, vi.mock) для мінімального порогу входу; watch-режим з миттєвим перезапуском тестів при зміні файлів.

Playwright — фреймворк End-to-end тестування від Microsoft, що автоматизує реальні браузери (Chromium, Firefox, WebKit/Safari). Основні переваги Playwright над попередниками (Cypress, Selenium): підтримка паралельного запуску тестів у декількох браузерах одночасно; вбудований механізм auto-wait, що автоматично чекає на стабілізацію UI перед кожною дією без явних sleep-викликів; потужний локатор getByRole, що знаходить елементи за семантичною роллю (ARIA role), а не за CSS-селекторами, що робить тести стійкими до рефакторингу верстки. Playwright також підтримує запис тестових сценаріїв через генератор кодоспостереження (codegen), що суттєво прискорює написання E2E-тестів.

Порівняльну характеристику інструментів тестування, що використовуються у проєкті, наведено у таблиці 2.2.

Таблиця *2.2 — Порівняльна характеристика інструментів контролю якості*

Інструмент	**Тип перевірки**	**Рівень**	**Швидкість**	**Ключова перевага**
TypeScript compiler	Типова коректність	Компіляція	Висока	Наскрізна типова безпека
ESLint 9	Стиль та антипатерни	Статичний аналіз	Висока	Правила React Hooks (exhaustive-deps)
Vitest	Логіка функцій та хуків	Модульний	Дуже висока	Нативна інтеграція з Vite, ES-модулі
Playwright	Сценарії користувача	E2E	Середня	Реальні браузери, auto-wait, ARIA locators

2.3 Аналіз вимог до користувачів

На основі аналізу предметної галузі визначено три основні ролі користувачів системи та їх функціональні повноваження. Система ролей реалізована через окрему таблицю user_roles у PostgreSQL з перелічуваним типом app_role, що містить значення player, venue_owner та admin.

Гравець (player) є основним типом кінцевого користувача платформи — фізична особа, що шукає спортивний майданчик для індивідуального тренування, командної гри або корпоративного заходу. Функціональні можливості гравця охоплюють: перегляд каталогу майданчиків з повним описом, фотогалереєю, рейтингом та відгуками; використання системи пошуку та фільтрації за видом спорту, містом, типом покриття, ціновим діапазоном та наявністю конкретних зручностей (Showers, Locker Rooms,

Parking, WiFi, Café); перегляд детальної сторінки майданчика із адаптивною фотогалереєю; вибір дати та часового слоту на інтерактивному календарі з відображенням реальної доступності; підтвердження бронювання та отримання підтвердження; управління власними бронюваннями в особистому кабінеті; додавання майданчиків до списку обраних; редагування профілю.

Власник майданчика (venue_owner) — фізична або юридична особа, що є оператором або орендодавцем спортивного об'єкту. Функціональні можливості включають усе доступне гравцю, а також: створення картки спортивного майданчика з заповненням усіх параметрів (назва, місто, адреса, вид спорту, тип покриття, опис, зручності, ціна за годину); завантаження до 8 фотографій об'єкту через drag-and-drop або файловий діалог з можливістю визначення обкладинки; редагування та видалення власних лістингів. Важлива архітектурна деталь: власник може бачити та редагувати лише ті лістинги, що були створені під його обліковим записом — це забезпечується перевіркою поля createdBy на рівні клієнтського коду та RLS-політиками на рівні БД.

Адміністратор (admin) — привілейований обліковий запис для управління платформою та забезпечення якості контенту. Адміністратор має доступ до всіх даних платформи поза обмеженнями RLS, може управляти ролями користувачів через таблицю user_roles та виконувати модераційні дії щодо контенту. Призначення ролі admin відбувається виключно через пряме звернення до бази даних — це унеможливлює самостійне підвищення привілеїв.

Загальні вимоги до всіх користувачів: наявність пристрою з доступом до Інтернету та сучасним веб-браузером (Chrome 90+, Firefox 88+, Safari 14+, Edge 90+); наявність електронної пошти для реєстрації та отримання підтверджень.

2.4 Аналіз вимог до апаратного та програмного забезпечення

Оскільки застосунок SportFields є веб-застосунком і використовує хмарну інфраструктуру Supabase, вимоги до обладнання кінцевого користувача є мінімальними.

Вимоги до клієнтського пристрою: будь-який комп'ютер, ноутбук, планшет або смартфон з доступом до Інтернету; сучасний веб-браузер з підтримкою ES2020 та CSS Custom Properties; стабільне інтернет-з'єднання від 1 Мбіт/с (для роботи з фотогалереями — від 5 Мбіт/с); мінімальна роздільна здатність екрану — від 320 пікселів.

Вимоги до середовища розробки: операційна система Windows 10/11 x64, macOS Monterey 12+ або Ubuntu 20.04 LTS; процесор — мінімум 2 ядра (рекомендовано 4+); оперативна пам'ять — мінімум 4 ГБ (рекомендовано 8 ГБ для комфортної роботи з Vite та TypeScript Language Server); Node.js версії 18.x або 20.x LTS; npm 9.x або pnpm 8.x як менеджер пакетів; Git для управління версіями; редактор коду з підтримкою TypeScript (рекомендовано VS Code); доступ до мережі Інтернет для взаємодії з Supabase Cloud.

Вимоги до серверної інфраструктури (Supabase Cloud): проєкт Supabase у регіоні eu-central-1 (Франкфурт) або eu-west-1 (Лондон) для мінімальної затримки; план Free Tier достатній для розробки та тестування (500 МБ PostgreSQL, 1 ГБ Storage, 50 000 MAU); для production-середовища рекомендовано план Pro ($25/міс) з daily backups та 8 ГБ бази даних.

2.5 Висновки до розділу 2

– Розглянуто сучасні технології для розробки веб-застосунків; обґрунтовано вибір стеку TypeScript 5 + React 18 + Vite 5 + Tailwind CSS + Supabase (PostgreSQL) з огляду на вимоги до продуктивності, безпеки та масштабованості.

- Детально описано три ролі користувачів системи (player, venue_owner, admin) та їх функціональні повноваження; визначено архітектурні рішення щодо розмежування доступу.
- Визначено вимоги до апаратного та програмного забезпечення для кінцевих користувачів, середовища розробки та серверної інфраструктури Supabase Cloud.

3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ СПОРТИВНИХ МАЙДАНЧИКІВ

3.1 Опис предметної області

Предметна область охоплює проектування та реалізацію веб-застосунку для пошуку, перегляду та онлайн-бронювання спортивних майданчиків. Застосунок орієнтований на широку аудиторію: від індивідуальних гравців, що шукають вільний корт для разової гри, до власників спортивних комплексів, яким необхідний зручний інструмент управління бронюваннями та завантаженістю об'єктів.

Центральним об'єктом предметної галузі є спортивний майданчик (venue) — фізичний об'єкт з певним набором характеристик: назва, місцезнаходження, вид спорту, тип покриття (Artificial Turf, Natural Grass, Hard Court, Pool, Studio), перелік зручностей, ціна оренди за годину, фотоматеріали та текстовий опис. Майданчики мають часові слоти (time slots) — відрізки часу фіксованої тривалості (одна година), що можуть перебувати у трьох станах: доступний (available), заброньований (booked) або заблокований власником (blocked).

Процес бронювання — перехід слоту зі стану available до стану booked — є ключовою транзакцією системи та повинен відбуватися атомарно для запобігання подвійному бронюванню (double booking). Технічно це забезпечується унікальним обмеженням на рівні PostgreSQL та миттєвим оновленням стану доступності для всіх активних клієнтів.

Важливою особливістю предметної галузі є асиметрія між сторонами транзакції. Власник майданчика прагне максимізувати завантаженість об'єкту та мінімізувати адміністративні зусилля (ручне підтвердження замовлень, телефонні дзвінки). Гравець прагне знайти зручний за місцем, часом та ціною майданчик якомога швидше, без очікування підтвердження. Обидві потреби

вирішуються через єдину платформу з розмежованими ролями та відповідними інтерфейсами.

Правові аспекти функціонування платформи регулюються користувацькою угодою (Terms of Service) та політикою конфіденційності (Privacy Policy). Зокрема, платформа діє як посередник між гравцями та власниками майданчиків, не є стороною у договорі оренди та не несе відповідальності за фізичний стан об'єктів. Обробка персональних даних здійснюється відповідно до вимог GDPR: зберігається лише необхідний мінімум даних (ім'я, email, телефон, дані бронювань), дані захищені технічно через RLS та організаційно через чітко визначену політику зберігання. Скасування бронювань підпорядковується диференційованій політиці повернення коштів: повне повернення при відмові за 48+ годин до сесії, 50% повернення при відмові за 24–48 годин, відсутність повернення при відмові менше ніж за 24 години.

3.2 Архітектура системи

Архітектура застосунку SportFields побудована за принципом клієнт-серверного розподілу, де клієнтська частина реалізована як Single Page Application (SPA) на React 18 + TypeScript 5, а серверна інфраструктура делегована хмарній платформі Supabase з PostgreSQL 15.

На відміну від класичних REST-сервісів, де кожен запит до бази даних опосередкований власним серверним додатком, архітектура BaaS (Backend-as-a-Service) дозволяє клієнту звертатися до бази даних безпосередньо через типізований SDK, при цьому безпека забезпечується декларативними правилами Row Level Security на рівні PostgreSQL. Такий підхід суттєво скорочує час розробки та усуває цілий клас помилок, пов'язаних з некоректною реалізацією авторизаційних перевірок на рівні API.

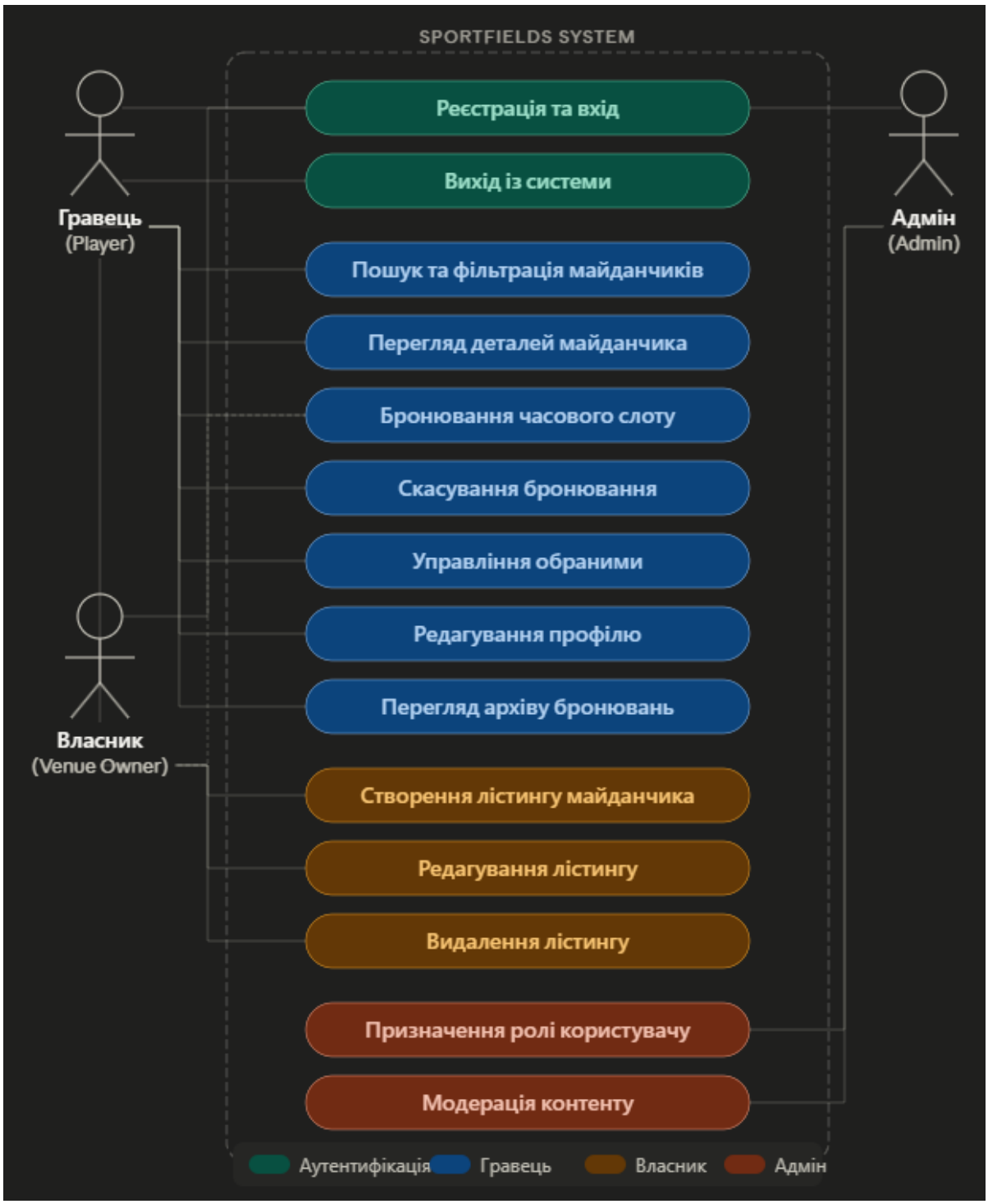


Рисунок. *1 — Use-Case діаграма застосунку*

Клієнтська частина реалізує чотири логічні шари з чітко визначеними відповідальностями. Шар представлення (Presentation Layer) складається з React-компонентів, що декларативно описують UI для заданого стану та не містять бізнес-логіки. Шар бізнес-логіки (Business Logic Layer) реалізований через кастомні хуки — useAuth, useVenueFilters, useProfileData,

useAllVenues — що інкапсулюють логіку, реактивний стан та побічні ефекти. Шар доступу до даних (Data Access Layer) включає Supabase Client SDK для роботи з PostgreSQL та набір функцій venuesStore.ts для роботи з localStorage. Шар типів та контрактів (Type Layer) складається з TypeScript-інтерфейсів (Venue, Booking, Profile, Favorite), Zod-схем для валідації форм та автоматично згенерованих типів Supabase.

Файлова структура проєкту організована за принципом feature-based: кожен великий екран (Home, VenuesSearch, VenueDetails, NewVenue, Auth, Profile) розміщений у власному каталозі всередині src/screens/. Загальні компоненти — у src/components/ui/, бізнес-логіка — у src/hooks/, типи та дані — у src/data/. Така організація забезпечує локальність змін: при модифікації функціоналу конкретного екрану зачіпаються лише файли в межах його каталогу.

3.2.1 Схема бази даних

База даних реалізована на PostgreSQL через платформу Supabase. Схема містить чотири основні таблиці та три перелічувані типи (Enum). Використання Enum-типів замість рядкових констант забезпечує цілісність даних на рівні СУБД — PostgreSQL фізично відхилить будь-яке значення, що не входить до визначеного переліку.

Таблиця profiles зберігає розширені дані профілю користувача (full_name, email, phone, avatar_url) та пов'язана з системною таблицею auth.users через зовнішній ключ. Запис у profiles створюється автоматично тригером handle_new_user при INSERT в auth.users, що гарантує існування профілю для кожного зареєстрованого користувача без додаткового коду на стороні клієнта.

Таблиця bookings є центральною сутністю системи та містить повну інформацію про бронювання: посилання на користувача (user_id) та майданчик (venue_id), часові межі сесії (starts_at, ends_at), вартість у центах (price_cents — зберігання у цілочисельному форматі усуває проблеми

округлення при фінансових розрахунках), поточний статус (booking_status Enum) та статус оплати (payment_status Enum).

Таблиця favorites реалізує M:N-зв'язок між користувачами та майданчиками з унікальним обмеженням на пару (user_id, venue_id), що запобігає дублюванню в обраному. Таблиця user_roles зберігає ролі користувачів окремо від профілів для захисту від privilege escalation — детальне обґрунтування наведено нижче.

Таблиця 1 – *Перелік таблиць бази даних*

Таблиця / Enum	**Ключові поля**	**Призначення**
profiles	id (uuid, FK→auth.users), full_name, email, phone, avatar_url	Розширений профіль; автоматично створюється тригером handle_new_user
bookings	id, user_id (FK), venue_id, starts_at, ends_at, price_cents, status, payment_status	Записи бронювань; захищені RLS; price_cents у цілочисельному форматі
favorites	id, user_id (FK), venue_id, created_at; UNIQUE(user_id,venue_id)	Обрані майданчики; унікальне обмеження запобігає дублюванню
user_roles	id, user_id (FK), role (app_role)	Ролі користувачів окремо від profiles для захисту від

Таблиця / Enum	Ключові поля	Призначення
		privilege escalation
app_role (Enum)	'player', 'venue_owner', 'admin'	Обмеження домену значень ролей на рівні СУБД
booking_status (Enum)	'pending','confirmed','completed','cancelled'	Статус бронювання; перевіряється PostgreSQL автоматично
payment_status (Enum)	'unpaid','paid','refunded'	Статус оплати; незалежний від booking_status для гнучкості

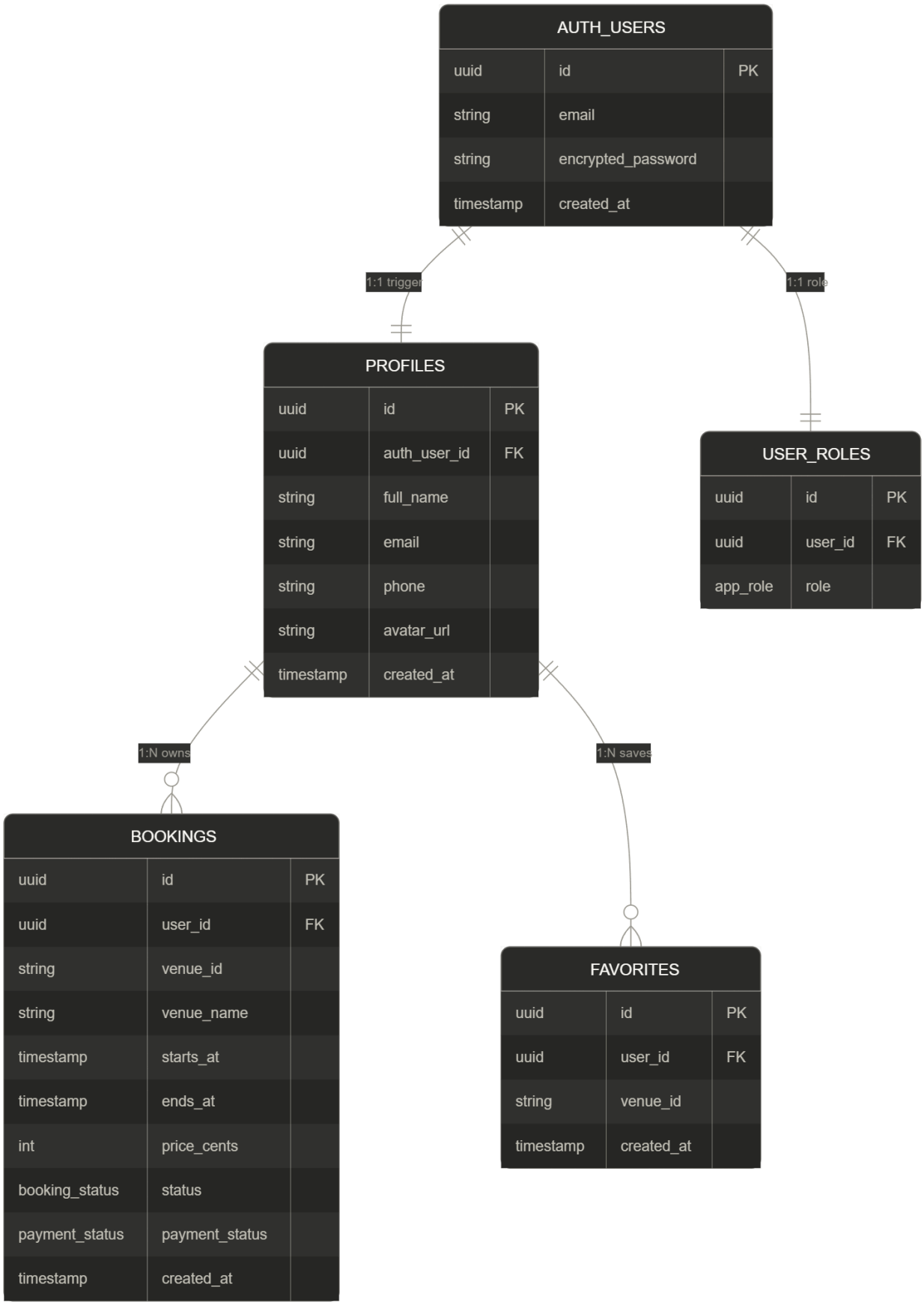


Рисунок *2 — ER-діаграма бази даних застосунку*

ER-діаграма бази даних застосунку SportFields наведена на рисунку 2. Діаграма відображає чотири основні таблиці та зв'язки між ними: таблиці bookings та favorites пов'язані зовнішнім ключем з таблицею profiles через поле user_id, таблиця user_roles має відношення «один до одного» з profiles.

3.2.2 Система безпеки: Row Level Security

Row Level Security (RLS) — механізм PostgreSQL, що дозволяє визначити предикати доступу на рівні кожної таблиці окремо для операцій SELECT, INSERT, UPDATE та DELETE. У Supabase-проєктах контекстом виконання є JWT-токен поточного користувача, з якого функція auth.uid() витягує ідентифікатор.

Ключова перевага RLS полягає в тому, що правила безпеки виконуються у СУБД і є невід'ємною частиною кожного запиту незалежно від його джерела. Навіть якщо зловмисник отримає доступ до anon-ключа API і спробує виконати запит SELECT * FROM bookings через PostgREST, PostgreSQL поверне лише рядки, де user_id = auth.uid(). Нижче наведено фрагмент SQL-міграції з RLS-політиками (Лістинг 3.2.2):

Лістинг *3.2.2 — SQL-міграція: RLS-політики та функція has_role*

```
-- Увімкнення RLS для всіх таблиць
ALTER TABLE public.profiles   ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.bookings   ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.favorites  ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.user_roles ENABLE ROW LEVEL SECURITY;

-- profiles: кожен бачить і редагує лише свій профіль
CREATE POLICY "Users can view own profile"
  ON public.profiles FOR SELECT TO authenticated
  USING (auth.uid() = id);

CREATE POLICY "Users can update own profile"
  ON public.profiles FOR UPDATE TO authenticated
  USING (auth.uid() = id) WITH CHECK (auth.uid() = id);

-- bookings: читання та запис лише своїх бронювань
CREATE POLICY "Users can view own bookings"
  ON public.bookings FOR SELECT TO authenticated
```

```
  USING (auth.uid() = user_id);

CREATE POLICY "Users can insert own bookings"
  ON public.bookings FOR INSERT TO authenticated
  WITH CHECK (auth.uid() = user_id);

-- user_roles: лише адміністратор може змінювати ролі
CREATE POLICY "Only admins manage roles"
  ON public.user_roles FOR ALL TO authenticated
  USING (has_role(auth.uid(), 'admin'));

-- Функція has_role: SECURITY DEFINER запобігає рекурсії
CREATE OR REPLACE FUNCTION has_role(uid uuid, r app_role)
RETURNS boolean LANGUAGE sql STABLE SECURITY DEFINER AS
$$
  SELECT EXISTS (
    SELECT 1 FROM public.user_roles
    WHERE user_id = uid AND role = r
  );
$$;
```

Атрибут SECURITY DEFINER у функції has_role є принциповим: ця функція виконується з правами власника (postgres-суперкористувача), а не поточного сеансового користувача. Без цього атрибуту виклик has_role зсередини RLS-політики таблиці user_roles спричинив би рекурсію. SECURITY DEFINER розриває цей ланцюг, дозволяючи безпечно перевіряти роль у будь-якій RLS-політиці.

Ролі користувачів навмисно винесено в окрему таблицю user_roles, а не зберігаються у profiles. Таке рішення виключає класичну вразливість підвищення привілеїв: якби роль зберігалась у профілі, що редагується, RLS-політика 'Users can update own profile' дозволила б користувачу змінити власне поле role на 'admin'. Окрема таблиця з власними суворими RLS та доступом лише для адміністраторів унеможливлює подібну атаку.

3.3 Архітектурні патерни та вимоги до системи

У даному підрозділі систематизовано методи, архітектурні патерни та підходи, застосовані при реалізації застосунку SportFields. Систематизація

дозволяє чітко визначити теоретичне підґрунтя прийнятих рішень та обґрунтувати їх доцільність у контексті поставлених задач.

3.3.1 Декларативний підхід, JSX та управління CSS-класами

JSX (JavaScript XML) — синтаксичне розширення JavaScript, що дозволяє описувати структуру UI безпосередньо у коді компонента, поєднуючи розмітку та логіку в єдиному файлі. При збірці Vite (через SWC) JSX-вирази трансформуються у виклики React.createElement(), що повертають об'єкти-дескриптори вузлів віртуального DOM. Ключова перевага JSX полягає у тому, що TypeScript повністю розуміє JSX-синтаксис та перевіряє типи props кожного компонента — передача пропа неправильного типу або відсутнього обов'язкового пропа спричиняє помилку компіляції, а не runtime-виняток.

Застосунок SportFields використовує JSX у всіх компонентах. Особливо важливою є можливість умовного рендерингу через тернарний оператор та логічний оператор && безпосередньо у JSX-виразах, що дозволяє декларативно описувати всі стани компонента (завантаження, помилка, порожній стан, нормальний стан) в одному місці без складних умовних гілок поза JSX-поверненням.

При побудові варіативних UI-компонентів (кнопки різних розмірів та типів, інпути в різних станах тощо) виникає задача програмного формування рядка className з урахуванням активних варіантів та умовних класів. Для вирішення цієї задачі у проєкті використовується комбінація трьох інструментів.

Бібліотека clsx надає утилітарну функцію для умовного об'єднання рядків CSS-класів: clsx('base-class', condition && 'conditional-class', { 'another': isActive }) повертає рядок лише з тими класами, умова яких є true. Бібліотека tailwind-merge (twMerge) вирішує специфічну для Tailwind проблему конфліктів: якщо передати рядок 'px-4 px-8', стандартна конкатенація збереже обидва класи, але реально застосується лише один (поведінка залежить від

порядку у CSS-файлі, що є непередбачуваним). twMerge розуміє семантику Tailwind-класів і залишає лише останній з конфліктуючих.

Class Variance Authority (CVA) надає типізований спосіб опису варіативних компонентів через об'єкт-схему з базовими класами, варіантами та їх комбінаціями. Перевага CVA: TypeScript автоматично виводить тип допустимих значень пропа variant з CVA-схеми, що виключає передачу неіснуючого варіанта. У проєкті CVA застосовується у більшості базових компонентів src/components/ui/ (Button, Badge, Alert).

Лістинг *3.3.2 — Утиліта cn() та CVA для варіативних компонентів*

```
// src/lib/utils.ts — стандартна утиліта cn() у shadcn/ui-проєктах
import { clsx, type ClassValue } from "clsx";
import { twMerge }                from "tailwind-merge";

// cn() = умовне об'єднання (clsx) + вирішення конфліктів Tailwind (twMerge)
export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}

// Приклад CVA для варіативної кнопки
import { cva } from "class-variance-authority";

const buttonVariants = cva(
  // базові класи (завжди присутні)
  "inline-flex items-center justify-center rounded-md font-medium transition-colors",
```

```
      {
        variants: {
          variant: {
            default:         "bg-primary  text-primary-
foreground hover:bg-primary/90",
            destructive:     "bg-destructive     text-
destructive-foreground",
            outline:         "border  border-input  bg-
background hover:bg-accent",
            ghost:         "hover:bg-accent hover:text-
accent-foreground",
          },
          size: {
            default: "h-10 px-4 py-2",
            sm:      "h-9 rounded-md px-3",
            lg:      "h-11 rounded-md px-8",
            icon:    "h-10 w-10",
          },
        },
        defaultVariants:  {  variant:  "default",  size:
"default" },
      }
    );
    // TypeScript автоматично виводить тип: variant?:
"default"|"destructive"|"outline"|"ghost"
```

3.3.2 Анімаційна система, стани екрану та оптимізація продуктивності

Застосунок SportFields використовує кастомні CSS-анімації, визначені через директиву @keyframes у файлі src/index.css та підключені до Tailwind

через конфігурацію tailwind.config.ts. Анімаційна система складається з чотирьох анімацій з чітко визначеними призначеннями.

Анімація fade-in реалізує поступову появу елемента через зміну opacity від 0 до 1 та translateY від +10px до 0. Застосовується для появи секцій при прокрутці головної сторінки та для появи результатів пошуку при зміні фільтрів. Анімація fade-up є варіантом fade-in зі збільшеним зсувом по Y (+20px), що створює ефект «виринання» контенту. Анімація shimmer реалізує характерний ефект «мерехтіння» для skeleton-лоадерів через переміщення градієнту від краю до краю елемента. Анімація spin реалізує безперервне обертання елемента на 360°; застосовується для індикаторів завантаження у кнопках під час виконання асинхронних запитів.

Лістинг 3.3.3 — CSS @keyframes та реєстрація анімацій у Tailwind

```
/* src/index.css — визначення анімацій */
@keyframes fade-in {
  from { opacity: 0; transform: translateY(10px); }
  to   { opacity: 1; transform: translateY(0); }
}
@keyframes fade-up {
  from { opacity: 0; transform: translateY(20px); }
  to   { opacity: 1; transform: translateY(0); }
}
@keyframes shimmer {
  0%   { background-position: -200% 0; }
  100% { background-position:  200% 0; }
}
@keyframes spin {
  from { transform: rotate(0deg); }
  to   { transform: rotate(360deg); }
}
```

```
/* tailwind.config.ts — реєстрація анімацій як Tailwind
utilities */
// animation: {
//   "fade-in": "fade-in 0.4s ease-out",
//   "fade-up": "fade-up 0.5s ease-out",
//   shimmer:   "shimmer 2s linear infinite",
//   spin:      "spin 1s linear infinite",
// }
```

Кожен екран або компонент застосунку SportFields, що завантажує асинхронні дані, реалізує чотири стани відображення — Loading, Success, Error та Empty. Цей патерн є обов'язковим стандартом надійного UI та суттєво впливає на сприйняту якість продукту користувачами.

Стан Loading відображається через компоненти-скелетони (Skeleton з shimmer-анімацією), що візуально імітують форму майбутнього контенту. Це сприймається краще, ніж звичайний спінер, оскільки дає користувачу очікування структури сторінки ще до завантаження даних. Стан Success відображає основний контент компонента після успішного отримання даних. Стан Error відображає повідомлення про помилку з деталями (де це безпечно) та кнопкою повторної спроби. Стан Empty відображається коли запит виконано успішно, але дані відсутні (наприклад, порожній список бронювань або відсутні результати пошуку) — кожен порожній стан містить описову іконку, текстове пояснення та CTA-посилання для наступної дії.

Лістинг 3.3.4 — Патерн «Чотири стани екрану» у компоненті Bookings

```
// Типова реалізація чотирьох станів у компоненті
const BookingsPage = () => {
  const { bookings, loading } = useBookings();

  // 1. Loading state — скелетони замість реального контенту
  if (loading) return (
    <div className="space-y-4">
```

```
      {[...Array(3)].map((_, i) => (
        <Skeleton key={i} className="h-24 w-full rounded-xl"
/>
      ))}
    </div>
  );

  // 2. Empty state — інформативний з CTA
  if (bookings.length === 0) return (
    <div className="text-center py-12">
      <Calendar  className="mx-auto  h-12  w-12  text-muted-
foreground" />
      <h3 className="mt-4 text-lg font-medium">No bookings
yet</h3>
      <p className="text-muted-foreground">Find a venue and
book your first session</p>
      <Button asChild className="mt-4">
        <Link to="/venues">Browse venues</Link>
      </Button>
    </div>
  );

  // 3. Success state — основний контент
  return <BookingsList bookings={bookings} />;

  // 4. Error state — обробляється вище через ErrorBoundary
або try/catch
};
```

Окрім useMemo для кешування результатів обчислень, застосунок SportFields використовує хук useCallback для стабілізації посилань на функції-обробники. Ця оптимізація є важливою у трьох конкретних випадках.

По-перше, при передачі функцій як props дочірнім компонентам, обгорнутим у React.memo(). Без useCallback при кожному рендері

батьківського компонента створюється новий об'єкт функції, що викликає повторний рендер дочірнього компонента (навіть якщо React.memo() обгортає його) через зміну посилання props. useCallback гарантує, що посилання на функцію стабільне між рендерами, якщо залежності не змінились.

По-друге, при використанні функцій як залежностей у масиві залежностей інших хуків (useEffect, useMemo). Нестабільне посилання на функцію призведе до нескінченного циклу: ефект запускає функцію → функція оновлює стан → рендер створює нову функцію → ефект запускає знову.

По-третє, при передачі обробників подій компонентам з тривалим рендером (Calendar, великі списки). useCallback дозволяє React пропустити повторний рендер таких компонентів при незмінних обробниках.

Лістинг 3.3.5 — useCallback для стабілізації посилань на функції-обробники

```
// Приклад стабілізації обробника через useCallback
const VenueCard = ({ venue }: { venue: Venue }) => {
  const navigate = useNavigate();
  const { user }  = useAuth();

  //   Без useCallback: нова функція при кожному рендері VenueCard
  // const handleFavorite = async () => { ... };

  //  З useCallback: стабільне посилання; перестворення лише
  //    при зміні user?.id або venue.id
  const handleFavorite = useCallback(async () => {
    if (!user) { navigate("/login"); return; }
    await toggleFavorite(venue.id, user.id);
  }, [user?.id, venue.id, navigate]);

  //   FavoriteButton отримає стабільний prop — не перерендериться
```

```
  // при рендері батьківського компонента без зміни user або
venue
  return (
    <Card>
      <FavoriteButton onToggle={handleFavorite} />
    </Card>
  );
};
```

3.3.3 Управління формами, Optimistic UI та Progressive Enhancement

Форми автентифікації (Login, Register) у застосунку SportFields реалізують патерн Controlled Components — підхід, при якому React-стан є єдиним джерелом правди для значення кожного поля форми. На відміну від uncontrolled inputs (де значення живе у DOM-вузлі), controlled components синхронізують DOM та стан через обробник onChange. Хоча React Hook Form для основних форм використовує uncontrolled inputs для продуктивності, форми автентифікації (Login, Register) використовують controlled approach через useState для кожного поля через простоту та необхідність додаткової логіки (show/hide пароль, очищення помилок при введенні).

Лістинг 3.3.6 — Controlled Components у формі автентифікації

```
// src/screens/Auth/Login.tsx — Controlled Components
const Login = () => {
  // Кожне поле — окремий стан компонента (controlled)
  const [email,    setEmail]    = useState("");
  const [password, setPassword] = useState("");
  const  [showPwd,    setShowPwd]    =  useState(false);  //
show/hide toggle
  const [loading,  setLoading]  = useState(false);
  const  [error,       setError]       =  useState<string  |
null>(null);
```

```
  // При зміні будь-якого поля — очищуємо помилку (UX
покращення)
  const           handleEmailChange           =           (e:
React.ChangeEvent<HTMLInputElement>) => {
    setError(null);
    setEmail(e.target.value);
  };

  return (
    <form onSubmit={submit}>
      {/* value прив'язаний до стану — React контролює DOM
*/}
      <Input
        type="email"
        value={email}
        onChange={handleEmailChange}
        placeholder="your@email.com"
      />
      <div className="relative">
        <Input
          type={showPwd ? "text" : "password"}
          value={password}
          onChange={e => setPassword(e.target.value)}
        />
        {/* Кнопка перемикання видимості пароля */}
        <button type="button" onClick={() => setShowPwd(p =>
!p)}>
          {showPwd ? <EyeOff /> : <Eye />}
        </button>
      </div>
      {error   &&   <p   className="text-destructive   text-
sm">{error}</p>}
    </form>
  );
};
```

Optimistic UI — патерн проектування інтерфейсів, при якому UI оновлюється миттєво до отримання підтвердження від сервера, припускаючи, що операція завершиться успішно. При невдачі операції UI відкочується до попереднього стану. Цей патерн суттєво покращує сприйняту швидкість роботи застосунку.

У застосунку SportFields елементи optimistic UI застосовуються у кількох місцях. При натисканні кнопки 'Add to Favourites' іконка серця миттєво змінює стан (порожнє → заповнене) ще до відповіді Supabase. Якщо запит до БД завершується помилкою, іконка повертається до початкового стану та відображається toast з повідомленням про помилку. При підтвердженні бронювання toast-сповіщення 'Booking confirmed!' відображається одразу після натискання кнопки, паралельно з виконанням INSERT-запиту до bookings. Якщо запит завершується помилкою (наприклад, слот вже зайнятий паралельним бронюванням), toast-успіх замінюється toast-помилкою і форма повертається до стану вибору.

Патерн Single Source of Truth (єдине джерело правди) є фундаментальним принципом управління станом у React-застосунках. У застосунку SportFields цей принцип реалізований на декількох рівнях. URL є єдиним джерелом правди для стану фільтрів (useVenueFilters) — будь-яка спроба «кешувати» стан фільтрів у useState паралельно з URL призведе до розсинхронізації при прямому введенні URL або навігації назад. Таблиця bookings у PostgreSQL є єдиним джерелом правди для доступності слотів — застосунок ніколи не кешує стан слотів між різними датами, а завжди робить свіжий запит при зміні обраної дати. Функція venuePhotos() у venuesStore.ts є єдиним місцем формування повного списку фотографій майданчика — логіка злиття photos та coverImage не дублюється у компонентах.

Progressive Enhancement — стратегія веб-розробки, що полягає у послідовному нашаруванні функціональності: базова функціональність доступна всім, а розширені можливості додаються для підтримуваних

середовищ. У контексті SPA на React цей принцип реалізується через забезпечення семантичної HTML-розмітки, що залишається значущою навіть без виконання JavaScript.

У застосунку SportFields прогресивне покращення реалізовано на трьох рівнях. Семантична розмітка: навігаційні посилання реалізовані через компонент Link (який рендерить тег <a> з href), а не через onClick-обробники на . Це означає, що навігація між сторінками працює навіть при заблокованому JavaScript (server-side rendering не використовується, але семантика зберігається). Доступність форм: поля форм реалізовані через компоненти Input з відповідними атрибутами type, name, aria-label та автозаповненням (autocomplete). Це забезпечує коректну роботу менеджерів паролів та технологій підтримки доступності. Структурована розмітка: заголовки (h1, h2, h3) використовуються в семантичній ієрархії, що дозволяє screen-reader'ам надавати користувачам структуру сторінки.

3.3.4 Специфікація вимог до системи

Функціональні вимоги описують, що саме система повинна вміти робити. На основі аналізу предметної галузі, порівняльного дослідження аналогів та очікувань цільової аудиторії сформовано такий перелік вимог.

Блок автентифікації та авторизації: реєстрація нових користувачів за електронною поштою та паролем з вибором ролі (player або venue_owner) під час реєстрації; вхід до системи для зареєстрованих користувачів; збереження активної сесії між сесіями браузера через JWT-токени з автоматичним оновленням; вихід із системи з повним очищенням токенів; захист приватних маршрутів — перенаправлення неавторизованих користувачів на сторінку входу при спробі прямого переходу на /profile або /venues/new.

Блок каталогу майданчиків: відображення списку майданчиків з можливістю пошуку за текстовим запитом (пошук за назвою, містом, адресою, видом спорту); фільтрації за видом спорту з заздалегідь визначеного переліку (Football, Tennis, Basketball, Volleyball, Swimming, Training, Yoga); фільтрації

за містом (пошук за включенням рядка); фільтрації за типом покриття (Artificial Turf, Natural Grass, Hard Court, Pool, Studio); фільтрації за ціновим діапазоном через слайдер (0–200 GBP/год); фільтрації за зручностями (Showers, Locker Rooms, Parking, WiFi, Café) з вибором декількох значень. Стан фільтрів синхронізується з URL query parameters.

Блок деталей майданчика та бронювання: сторінка майданчика відображає адаптивну фотогалерею (різний макет залежно від кількості фото: 1, 2, 3 або 4+); повну технічну інформацію; блок бронювання з інтерактивним календарем та динамічною сіткою часових слотів. При виборі дати система виконує запит до бази даних, агрегує зайняті слоти у Set<number> та відображає їх як недоступні. Слоти у минулому блокуються для поточного дня через порівняння з поточним годинником.

Нефункціональні вимоги: адаптивний дизайн від 320px без горизонтальної прокрутки; безпека — всі таблиці захищені RLS, паролі зберігаються у хешованому вигляді (bcrypt, обробляється Supabase Auth), дані передаються лише через HTTPS; продуктивність — фільтрація через useMemo без зайвих рендерів; доступність WAI-ARIA через Radix UI; підтримуваність — feature-based структура та TypeScript-типізація.

3.3.5 Бібліотека підпрограм (класів)

Бібліотека підпрограм застосунку SportFields складається з набору кастомних React-хуків, утилітарних функцій та компонентів, що інкапсулюють повторювану логіку та забезпечують розділення відповідальності між шарами архітектури.

Хук useAuth (src/hooks/useAuth.tsx) є центральним елементом підсистеми автентифікації. Він реалізує патерн Observer поверх Supabase Auth SDK: при монтуванні компонента AuthProvider встановлюється підписка на onAuthStateChange — потік подій, що генерується при вході (SIGNED_IN), виході (SIGNED_OUT) та оновленні токена (TOKEN_REFRESHED). Хук експортує об'єкт з полями user, session, role, loading та метод signOut.

Хук useVenueFilters (src/hooks/useVenueFilters.ts) реалізує патерн URL-as-State: весь стан фільтрів зберігається виключно у query parameters URL. Читання фільтрів відбувається через useSearchParams React Router — кожен render-цикл десеріалізує поточні параметри URL у типізований об'єкт VenueFilters. Функція update серіалізує об'єкт назад у URLSearchParams, записуючи лише ненульові значення (що мінімізує довжину URL). Функція clear скидає всі параметри до порожнього URLSearchParams.

Хук useAllVenues (src/data/venuesStore.ts) об'єднує статичний масив VENUES та динамічний масив user venues зі localStorage. Реактивність реалізована через три паралельних слухача подій: user-venues-changed (кастомна подія при будь-якій зміні сховища), storage (стандартна Web API подія при зміні localStorage в іншій вкладці), focus (ресинхронізація при поверненні до вкладки).

Набір хуків useProfileData (src/hooks/useProfileData.ts) включає: useProfile — завантаження профілю через maybeSingle() для безпечної обробки відсутнього запису; useBookings — завантаження бронювань з race-condition guard; useFavorites — завантаження обраних майданчиків. Додатково надаються утилітарні функції: displayName/initial для форматування імені; formatPrice/formatDate/formatTimeRange для відображення фінансових і часових даних; splitBookings для розбиття масиву бронювань на upcoming та past; totalHours для підрахунку зіграних годин за completed-бронюваннями.

3.4 Програмна реалізація функціональних модулів

У даному розділі наведено детальний опис реалізації ключових функціональних модулів застосунку SportFields з аналізом архітектурних рішень та фрагментами вихідного коду.

3.5.1 Автентифікація та управління сесіями

Підсистема автентифікації побудована на Supabase Auth та реалізує JWT-based автентифікацію з автоматичним оновленням токенів. При успішному вході Supabase Auth повертає пару токенів: access_token (короткоживучий, 1 година) та refresh_token (довгоживучий). SDK автоматично зберігає токени у localStorage та відновлює їх при перезавантаженні сторінки. При закінченні терміну дії access_token SDK виконує автоматичний refresh — користувач не помічає цього процесу.

Форма реєстрації (рис. 3) демонструє декілька важливих рішень. По-перше, вибір ролі при реєстрації реалізований через role-toggle компонент — перемикач між player та venue_owner; обрана роль передається у поле options.data під час signUp() та обробляється серверним тригером handle_new_user. По-друге, форма валідується через Zod-схему перед будь-яким мережевим запитом. По-третє, відображення пароля (show/hide toggle) реалізовано через локальний стан showPwd (Лістинг 3.5.1_1):

Лістинг *3.5.1_1 — AuthProvider: Observer-паттерн над Supabase Auth*

```
    // src/hooks/useAuth.tsx — AuthProvider
    export const AuthProvider = ({ children }: {
children: ReactNode }) => {
      const [session, setSession] = useState<Session |
null>(null);
      const [user, setUser]       = useState<User |
null>(null);
      const [role, setRole]       = useState<AppRole |
null>(null);
      const [loading, setLoading] = useState(true);

      useEffect(() => {
        // Observer: підписка на зміни сесії
```

```
        const { data: sub } =
supabase.auth.onAuthStateChange((_ev, s) => {
          setSession(s); setUser(s?.user ?? null);
          // setTimeout(0) — виконати fetchRole в
окремому мікротакті
          // щоб уникнути deadlock з внутрішньою
чергою Supabase
          if (s?.user) setTimeout(() =>
fetchRole(s.user.id), 0);
          else setRole(null);
        });

        // Відновлення сесії після перезавантаження
сторінки
        supabase.auth.getSession().then(({ data: {
session: s } }) => {
          setSession(s); setUser(s?.user ?? null);
          if (s?.user) fetchRole(s.user.id);
          setLoading(false);
        });

        // Звільнення ресурсів при розмонтуванні
(запобігання витоку)
        return () => sub.subscription.unsubscribe();
      }, []);

      const fetchRole = async (uid: string) => {
        const { data } = await supabase
          .from("user_roles").select("role")
```

```
            .eq("user_id", uid).maybeSingle();
          setRole((data?.role as AppRole) ?? null);
        };
        // ...
      };
```

Компонент RequireAuth реалізує декларативний Route Guard (Лістинг 3.5.1_2). Поки виконується первинне завантаження сесії (loading=true), компонент повертає null — це запобігає «миготінню» сторінки входу для вже авторизованих користувачів при перезавантаженні. Прапор replace у Navigate виключає повернення на закриту сторінку кнопкою «назад» браузера.

Лістинг *3.5.1_2 RequireAuth та повна конфігурація маршрутів застосунку*

```
    // src/App.tsx — RequireAuth та конфігурація маршрутів
    const RequireAuth = ({ children }: { children: JSX.Element }) => {
      const { user, loading } = useAuth();
      if (loading) return null;              // чекаємо відновлення сесії
      if (!user) return <Navigate to="/login" replace />;
      return children;
    };
    // Конфігурація маршрутів
    <Routes>
      <Route path="/"                element={<Home />} />
      <Route path="/login"           element={<Login />} />
      <Route path="/register"        element={<Register />} />
      <Route path="/venues"      element={<VenuesSearch />} />
      <Route                              path="/venue/:id" element={<VenueDetails />} />
      <Route path="/venues/new"
        element={<RequireAuth><NewVenue /></RequireAuth>} />
      <Route path="/venues/:id/edit"
```

```
        element={<RequireAuth><NewVenue
/></RequireAuth>} />
      <Route path="/profile"
        element={<RequireAuth><ProfileLayout
/></RequireAuth>}>
        <Route index              element={<Overview />}
/>
        <Route  path="bookings"  element={<Bookings  />}
/>
        <Route  path="history"    element={<HistoryPage
/>} />
        <Route path="account"  element={<Account />} />
        <Route  path="settings"  element={<SettingsPage
/>} />
      </Route>
    </Routes>
```

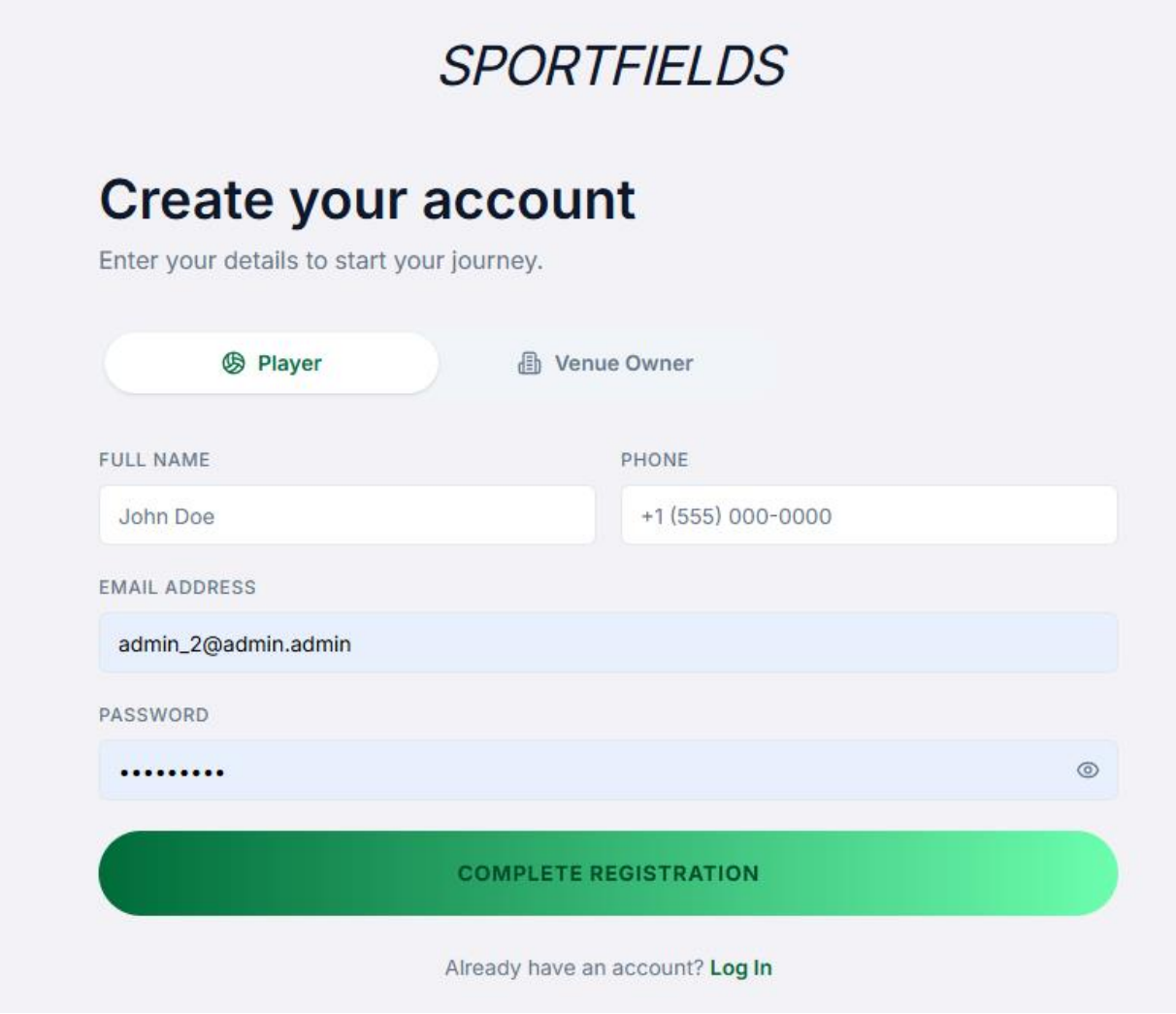


Рисунок 3 – *Форма автентифікації: реєстрація*

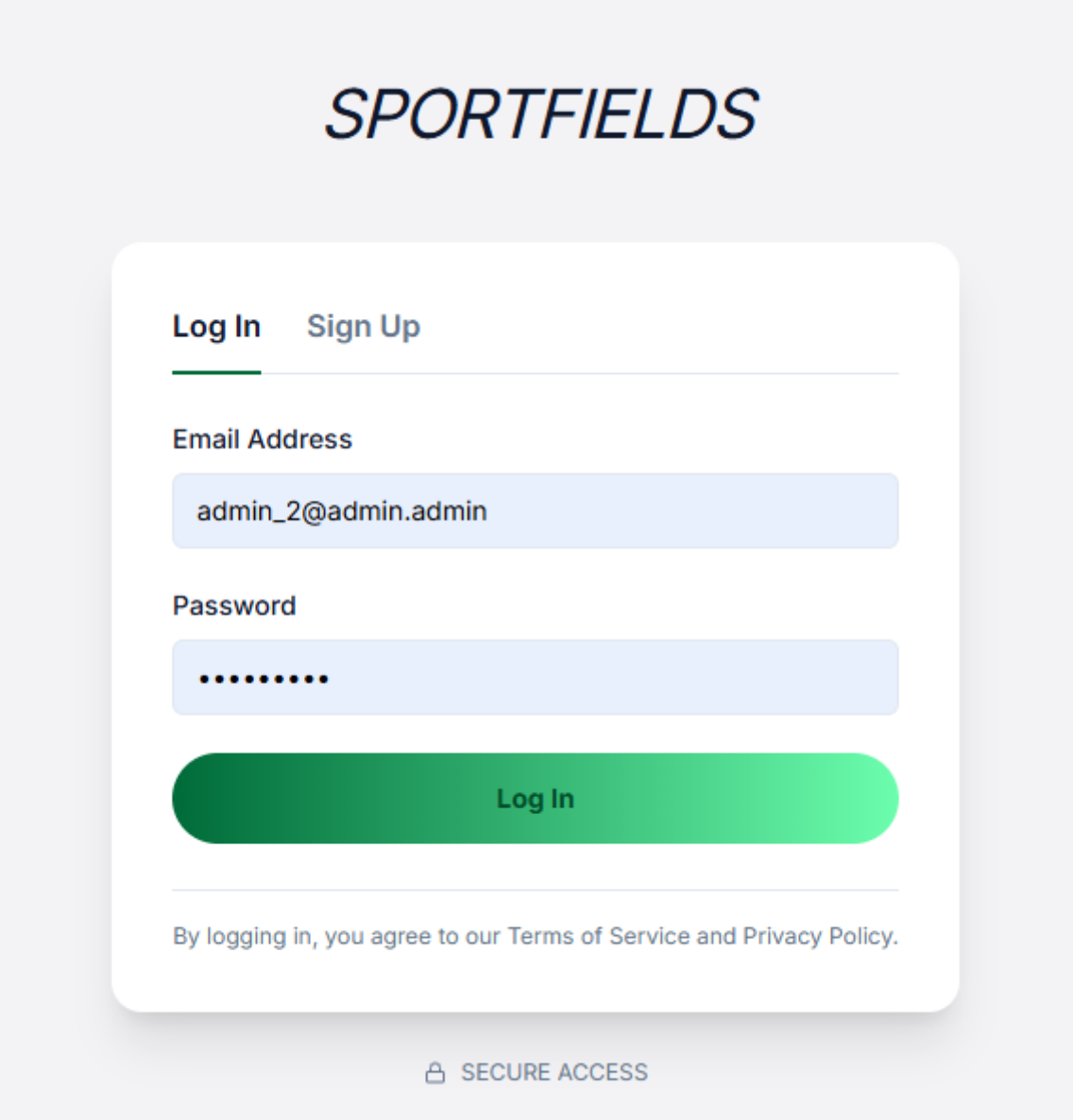


Рисунок 4 *Форма автентифікації: вхід*

Форма реєстрації нового користувача застосунку SportFields наведена на рисунку 4. Форма містить поля для введення імені, електронної пошти та пароля, а також перемикач ролі між player та venue_owner, що визначає доступні функції після реєстрації.

3.4.2 Система фільтрації та пошуку

Система фільтрації реалізує патерн URL-as-State. Стан всіх фільтрів зберігається виключно у query parameters URL поточної сторінки, що дає три переваги без додаткового коду: shareability (URL ?sport=Football&city=London відтворює стан фільтрів), коректна навігація браузером (кнопки 'назад'/'вперед'), відновлення після перезавантаження. Повна реалізація хуку наведена у Лістингу 3.5.2:

Лістинг *3.5.2 Повна реалізація хуку useVenueFilters*

```
// src/hooks/useVenueFilters.ts
export function useVenueFilters() {
  const [params, setParams] = useSearchParams();

  const filters: VenueFilters = {
    q:         params.get("q")       ?? "",
    sport:     params.get("sport")   ?? "",
    city:      params.get("city")    ?? "",
    surface:   params.get("surface") ?? "",
    amenities: params.get("amenities")
                 ?.split(",").filter(Boolean) ??
[],
    minPrice:  Number(params.get("minPrice") ??
0),
    maxPrice:  Number(params.get("maxPrice") ??
200),
  };

  const update = (patch: Partial<VenueFilters>) =>
{
```

```
        const next = { ...filters, ...patch };
        const sp = new URLSearchParams();
        if (next.q)                sp.set("q",
next.q);
        if (next.sport)            sp.set("sport",
next.sport);
        if (next.city)             sp.set("city",
next.city);
        if (next.surface)          sp.set("surface",
next.surface);
        if (next.amenities.length) sp.set("amenities",
next.amenities.join(","));
        if (next.minPrice)         sp.set("minPrice",
String(next.minPrice));
        if (next.maxPrice !== 200) sp.set("maxPrice",
String(next.maxPrice));
        setParams(sp);
      };

      const allVenues = useAllVenues();
      const results: Venue[] = useMemo(() => {
        const q = filters.q.trim().toLowerCase();
        return allVenues.filter((v) => {
          if (q && !(`${v.name} ${v.city} ${v.address}
${v.sport}`
                      .toLowerCase().includes(q)))
return false;
          if (filters.sport && v.sport.toLowerCase()
```

```
                                !==
filters.sport.toLowerCase()) return false;
            if (filters.city  && !v.city.toLowerCase()

.includes(filters.city.toLowerCase())) return false;
            if (filters.surface && v.surface !==
filters.surface) return false;
            if (filters.amenities.length &&
                !filters.amenities.every(a =>
v.amenities.includes(a)))
              return false;
            return v.pricePerHour >= filters.minPrice
                && v.pricePerHour <= filters.maxPrice;
          });
        // Залежність amenities через join() — рядок
змінюється
        // лише при зміні вмісту, а не посилання на
масив
        }, [allVenues, filters.q, filters.sport,
filters.city,
            filters.surface,
filters.amenities.join(","),
            filters.minPrice, filters.maxPrice]);

        return { filters, update, clear:
()=>setParams(new URLSearchParams()), results };
    }
```

Результат роботи системи фільтрації наведено на рисунку 5. На рисунку відображено каталог майданчиків з застосованим фільтром за видом спорту та

активними чіпами над сіткою результатів, що дозволяють швидко скинути окремі параметри.

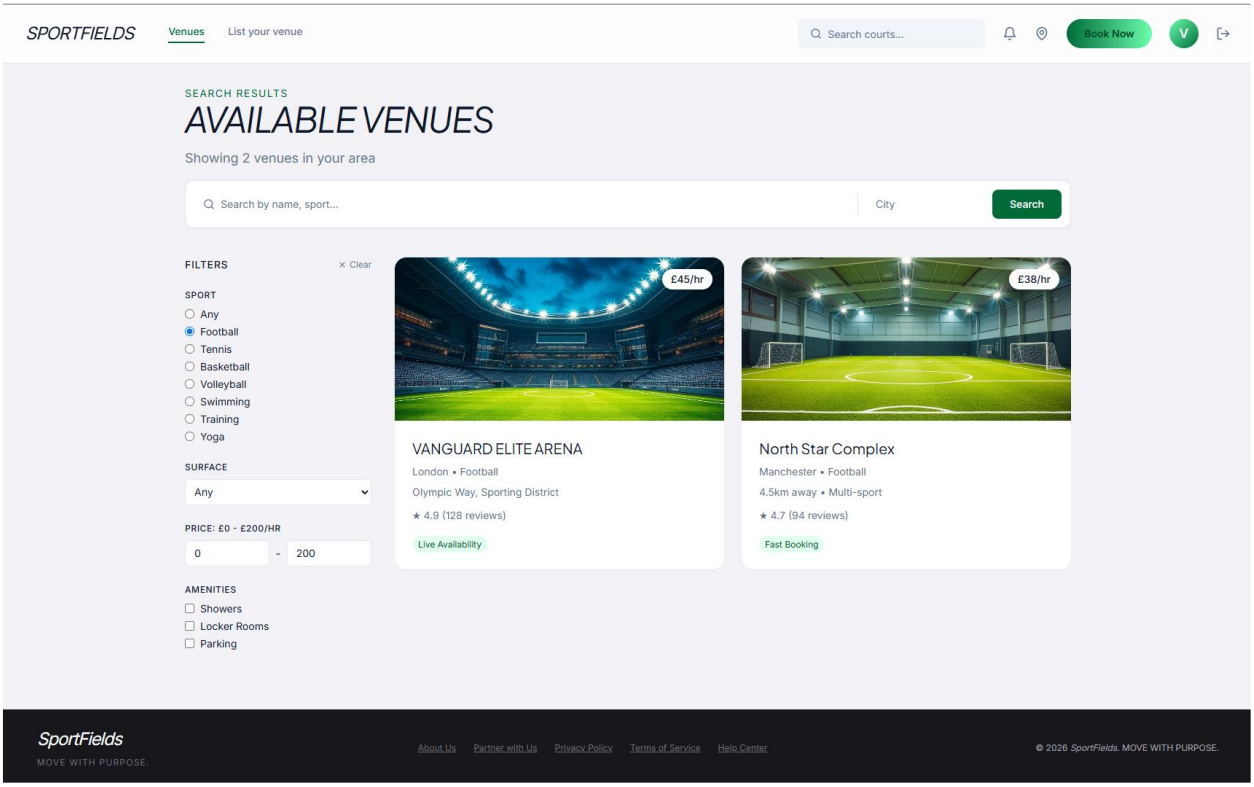


Рисунок *5* *Каталог майданчиків з активними фільтрами*

3.4.3 Система бронювання

Підсистема бронювання є функціонально найскладнішим модулем застосунку. При виборі дати виконується асинхронний запит до таблиці bookings, що повертає зайняті слоти для обраного дня. Результат агрегується у Set<number> — множину зайнятих годин. Використання Set замість Array забезпечує O(1) перевірку busyHours.has(h) при рендері кожного з 24 слотів. Race-condition guard через прапор cancelled вирішує класичну проблему паралельних запитів: якщо користувач швидко перемикає дати, результат застарілого запиту не перезапише актуальні дані (Лістинг 3.5.3):

Лістинг *3.5.3 Підсистема бронювання: race-condition guard та блокування слотів*

```
    // VenueDetails — підсистема бронювання
    const [date, setDate]          = useState<Date>(new
Date());
    const    [selectedHour,    setHour]             =
useState<number|null>(null);
    const       [busyHours,       setBusyHours]       =
useState<Set<number>>(new Set());
    const [loadingSlots, setLoading]= useState(false);

    useEffect(() => {
      let cancelled = false;          // race-condition
guard
      setHour(null);                  // скидаємо вибір
при зміні дати
      (async () => {
        setLoading(true);
        const dayStart = new Date(date);
        const dayEnd   = new Date(date);
        dayEnd.setDate(dayEnd.getDate()   +   1);     //
наступний день 00:00

        const { data } = await supabase
          .from("bookings")
          .select("starts_at, status")
          .eq("venue_id",   venue.id)
```

```
          .gte("starts_at", dayStart.toISOString()) //
>= 00:00:00
          .lt("starts_at",  dayEnd.toISOString())    //
< 00:00:00+1д
          .neq("status", "cancelled");               //
не враховуємо скасовані

        if (cancelled) return;    // застарілий запит —
ігноруємо
        setBusyHours(
          new   Set((data   ??   []).map(b   =>   new
Date(b.starts_at).getHours()))
        );
        setLoading(false);
      })();
      return () => { cancelled = true; };   // cleanup
при зміні дати
    }, [date, venue.id]);

    // Логіка блокування слоту
    const slotDisabled = (h: number): boolean => {
      const  isToday  =  date.toDateString()  ===  new
Date().toDateString();
      if (isToday && h <= new Date().getHours()) return
true; // минулий
      return busyHours.has(h);  // вже заброньований
    };
```

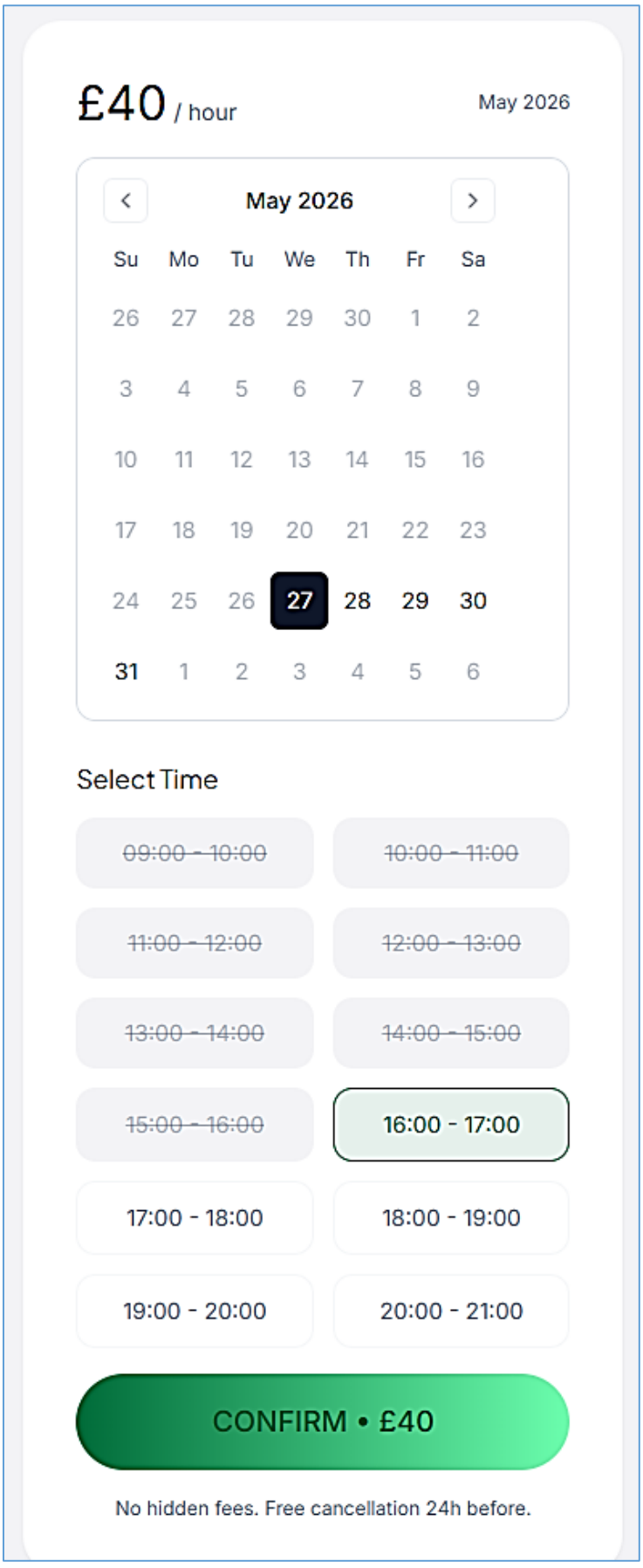


Рисунок 6 – *Блок бронювання: календар та сітка часових слотів*

Зовнішній вигляд блоку бронювання наведено на рисунку 6. Блок містить інтерактивний календар для вибору дати та сітку часових слотів з колірним кодуванням: зелений колір позначає доступні слоти, сірий — заброньовані або минулі інтервали часу.

3.4.4 Управління фотографіями та форма майданчика

Форма створення та редагування майданчика (рис. 8) реалізує drag-and-drop завантаження фотографій з використанням FileReader API. Компонент підтримує до 8 фотографій розміром не більше 4 МБ кожна, конвертує їх у формат data:URL (base64) та зберігає у стані компонента. Перший елемент масиву photos завжди є обкладинкою.

Функція addFiles реалізує захист від перевищення ліміту: обчислюється залишковий простір (room = MAX_PHOTOS - photos.length) та нарізається відповідна кількість файлів з FileList. Перестановка фотографій (вибір обкладинки) виконується функцією makeCover через деструктуризацію масиву без мутації оригіналу (Лістинг 3.5.4):

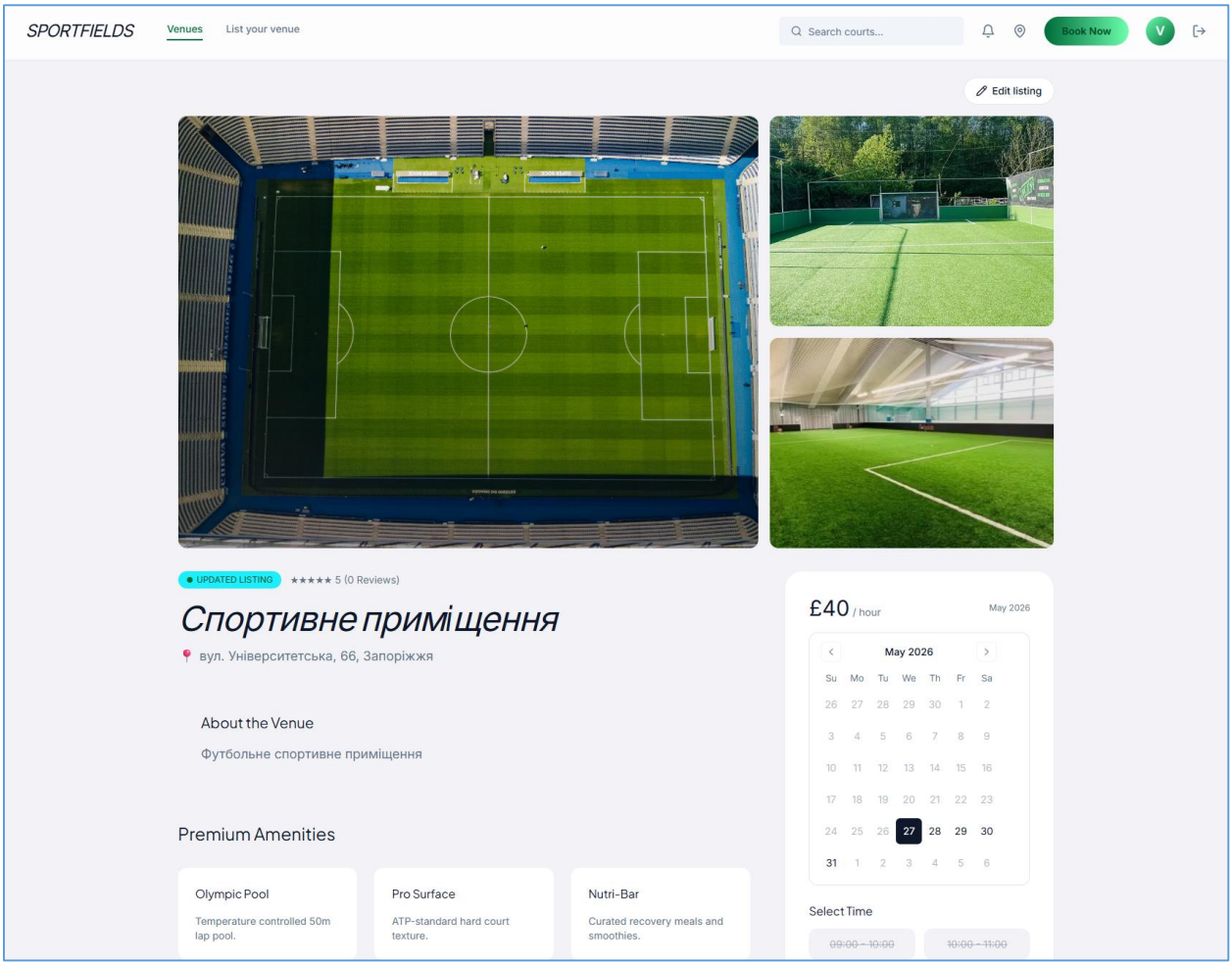


Рисунок *7 – Сторінка деталей майданчика — фотогалерея*

Адаптивна фотогалерея сторінки деталей майданчика наведена на рисунку 7. Галерея динамічно змінює макет залежно від кількості завантажених фотографій: для одного зображення використовується повноширинний макет, для чотирьох та більше — сітковий макет з overlay-елементом що відображає кількість прихованих знімків.

Лістинг *3.5.4 — FileReader API, управління фотографіями та перевірка власника*

```
// src/screens/NewVenue/NewVenue.tsx
const SURFACES      = ["Artificial Turf","Natural
Grass","Hard Court","Pool","Studio"];
```

```
    const ALL_AMENITIES = ["Showers","Locker
Rooms","Parking","WiFi","Café"];
    const MAX_PHOTOS    = 8;
    const MAX_FILE_BYTES = 4 * 1024 * 1024;  // 4 MB

    const addFiles = (files: FileList | null) => {
      if (!files || files.length === 0) return;
      const room = MAX_PHOTOS - photos.length;
      if (room <= 0) {
        toast({ title: "Photo limit reached",
                description: `Up to ${MAX_PHOTOS}
photos.`,
                variant: "destructive" });
        return;
      }
      Array.from(files).slice(0, room).forEach((file)
=> {
        if (file.size > MAX_FILE_BYTES) {
          toast({ title: `${file.name} too large`,
                  description: "Max 4MB per file",
variant: "destructive" });
          return;
        }
        const r = new FileReader();
        // base64-рядок додається до масиву через
функціональне оновлення
        r.onload = () => setPhotos(p => [...p,
r.result as string]);
        r.readAsDataURL(file);
```

```
      });
    };

    const makeCover = (i: number) =>
      setPhotos(p => {
        if (i === 0) return p;        // вже перший — нічого не робимо
        const next = [...p];          // не мутуємо оригінальний масив
        const [chosen] = next.splice(i, 1);
        return [chosen, ...next];     // обрана фото на першу позицію
      });
    // Перевірка власника при редагуванні чужого лістингу
    if (user && v.createdBy && v.createdBy !== user.id) {
      toast({ title: "Not your listing", variant: "destructive" });
      navigate(`/venue/${editingId}`, { replace: true });
      return;
    }
```

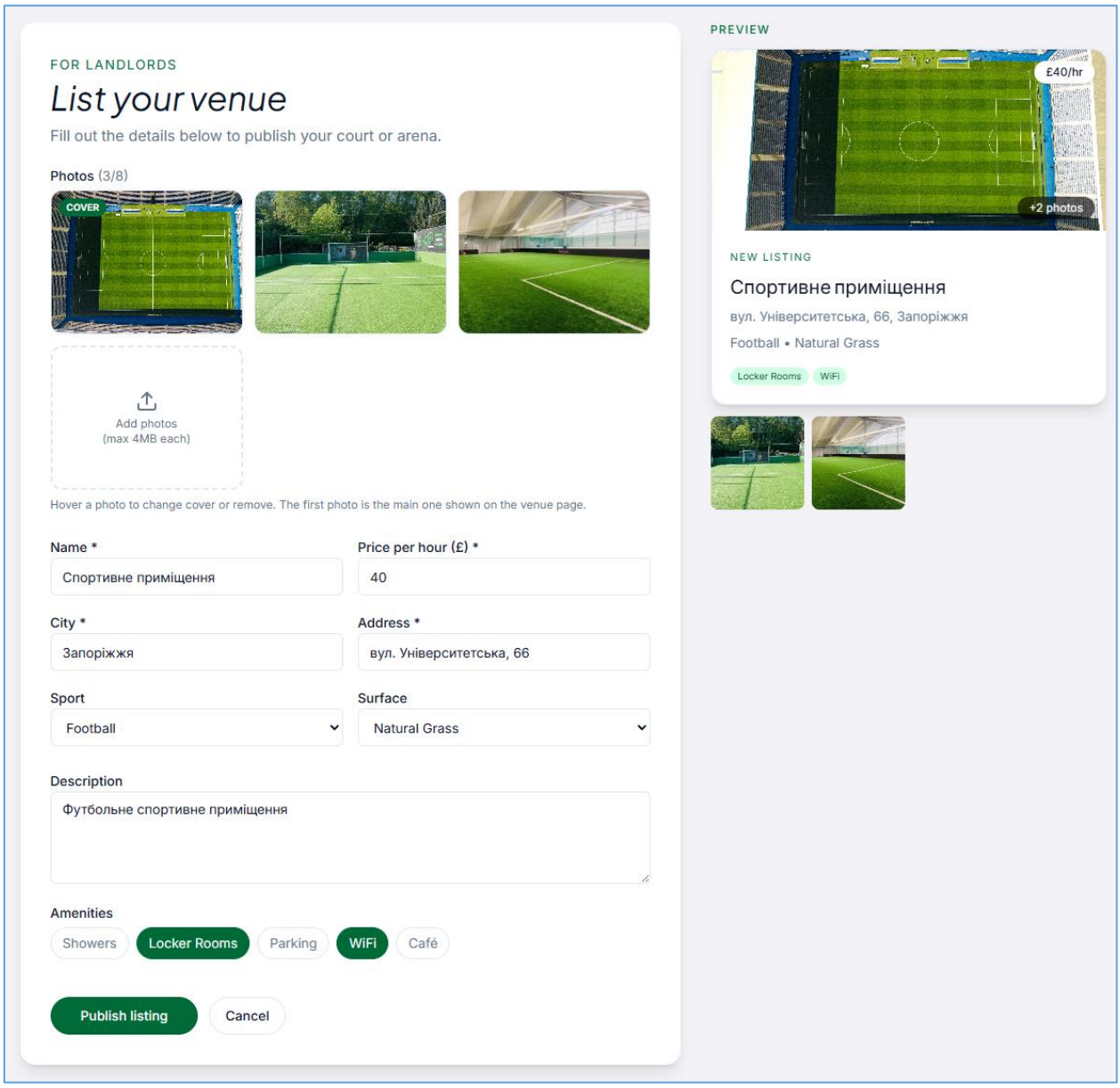


Рисунок *8 – Форма створення нового майданчика (NewVenue)*

Форма створення нового лістингу майданчика наведена на рисунку 8. Форма організована у двоколонковий макет: ліва колонка містить поля введення даних, права — відображає live-preview картки майданчика у реальному часі, що дозволяє власнику миттєво бачити результат.

3.4.5 Особистий кабінет та статистика користувача

Особистий кабінет (Profile) організований як набір вкладених маршрутів React Router DOM v6. Шаблонний компонент ProfileLayout рендерить

навігаційний сайдбар та область Outlet для вкладеного контенту. Кожна вкладка є окремим маршрутом та завантажує дані лише при активації.

Сторінка огляду (Overview) агрегує дані з кількох джерел: useProfile для відображення імені та аватара; useBookings для розрахунку статистики — загальна кількість бронювань, зіграних годин через totalHours, зароблених балів через completed * 10. Функція totalHours обчислює тривалість кожного completed-бронювання: (new Date(b.ends_at).getTime() - new Date(b.starts_at).getTime()) / 3_600_000 — ділення на 3 600 000 мс конвертує мілісекунди у години.

Сторінка бронювань (Bookings) відображає найближче заплановане бронювання у вигляді hero-картки з фотографією майданчика, назвою, датою, часом та кортом. Решта запланованих бронювань відображаються у сітці. Місячна статистика (кількість сесій та загальна кількість годин у поточному місяці) відображається у окремому блоці з прогрес-баром відносно цілі 10 сесій на місяць. При відсутності бронювань відображається empty state з CTA-посиланням на каталог майданчиків.

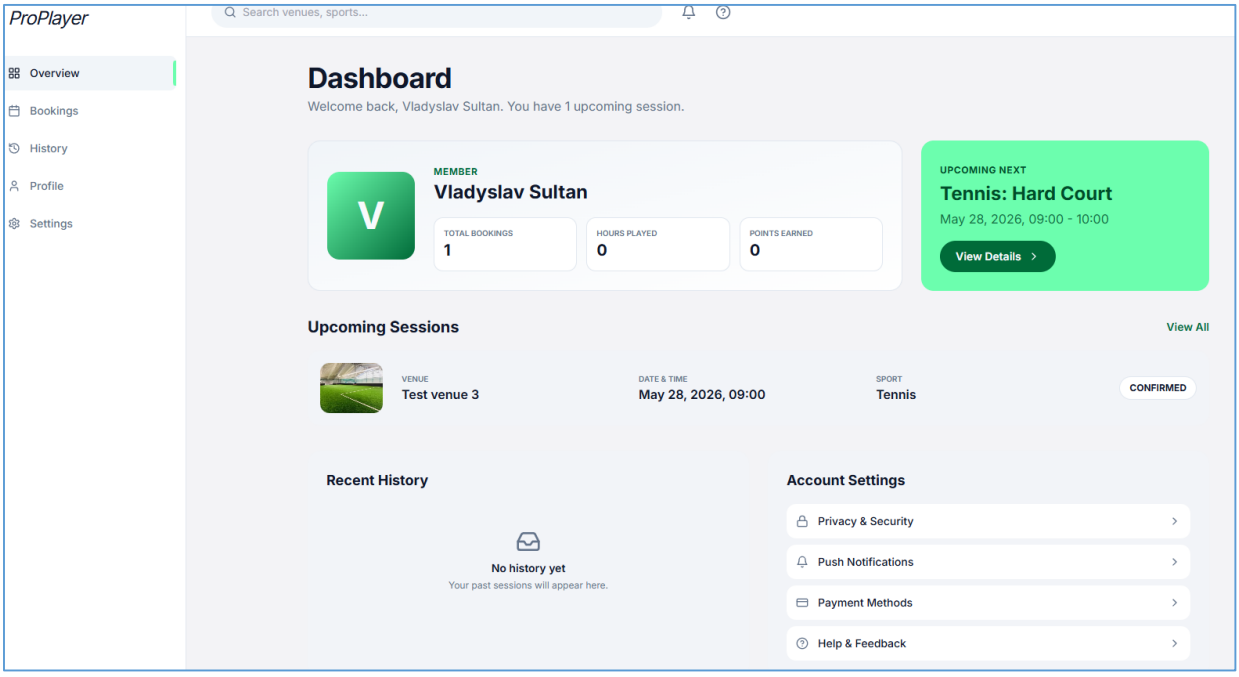


Рисунок *9 Особистий кабінет — сторінка огляду (Overview)*

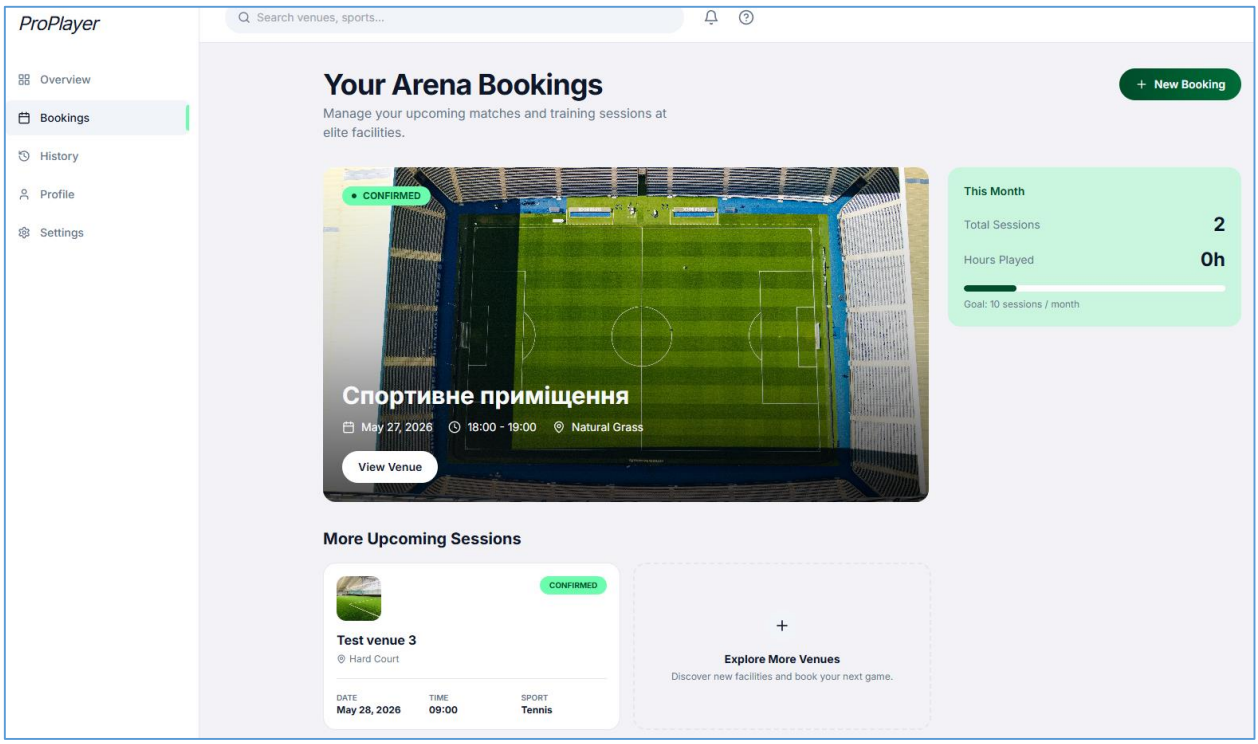


Рисунок *10 Особистий кабінет — сторінка поточних бронювань (Bookings)*

Зовнішній вигляд сторінки огляду особистого кабінету наведено на рисунку 9. Сторінка відображає агреговану статистику користувача (кількість бронювань, зіграних годин та накопичених балів) та блок найближчого запланованого бронювання.

Сторінка поточних бронювань особистого кабінету наведена на рисунку 10. Найближче заплановане бронювання відображається у вигляді великої hero-карточки з фотографією майданчика, деталями сесії та кнопкою управління.

3.4.6 Дизайн-система та компонентна бібліотека

Дизайн-система застосунку SportFields базується на семантичних CSS Custom Properties (дизайн-токенах), визначених у файлі src/index.css. Всі кольори зберігаються у форматі HSL без обгортки hsl(...) — це дозволяє Tailwind CSS застосовувати функцію hsl(var(--primary) / 0.5) для регулювання прозорості. Токени розподілені на семантичні категорії: базові (--background,

--foreground, --border), інтерактивні (--primary, --secondary, --destructive) та брендові (--brand-green, --brand-green-light, --brand-green-dark, --brand-surface).

Підтримка темної теми реалізована через перевизначення токенів у блоці .dark без жодних змін у компонентах. CSS-анімації (fade-in, fade-up, shimmer) визначені через @keyframes та застосовуються через Tailwind-класи. Shimmer-анімація використовується для skeleton-лоадерів, що відображаються замість контенту під час завантаження даних.

Компонентна бібліотека shadcn/ui надає headless-компоненти на базі Radix UI Primitives: Button, Input, Calendar, Dialog, Select, Toast, Card та інші. Особливість: компоненти не встановлюються як npm-пакет — їх вихідний код копіюється у проєкт (src/components/ui/), що дозволяє вільно модифікувати будь-який компонент. Radix UI забезпечує клавіатурну навігацію (Tab, Enter, Escape, Arrow keys), ARIA-атрибути та управління фокусом відповідно до WAI-ARIA 1.2.

3.4.7 Кросс-вкладкова синхронізація даних

Для забезпечення узгодженості стану між кількома відкритими вкладками браузера реалізовано pub/sub-механізм поверх Web API. При будь-якій зміні localStorage генерується кастомна подія user-venues-changed через window.dispatchEvent. Хук useAllVenues підписується на три джерела подій (Лістинг 3.5.7):

Лістинг *3.5.7 — useAllVenues: кросс-вкладкова синхронізація через storage events*

```
// src/data/venuesStore.ts
const KEY = "user_venues_v1";

const emit = () =>
  window.dispatchEvent(new Event("user-venues-changed"));
```

```
export const saveUserVenues = (list: Venue[]) => {
  try {
    localStorage.setItem(KEY,
JSON.stringify(list));
  } catch (e) {
    console.error("Storage quota exceeded", e);
  }
  emit();     // сповіщаємо підписників у поточній
вкладці
};

export const useAllVenues = (): Venue[] => {
  const [user, setUser] = useState<Venue[]>(() =>
getUserVenues());

  useEffect(() => {
    const handler = () => setUser(getUserVenues());
    handler();   // примусова ресинхронізація при
монтуванні

    // Зміни у поточній вкладці (через emit())
    window.addEventListener("user-venues-changed",
handler);
    // Зміни localStorage в ІНШИХ вкладках того ж
домену
    window.addEventListener("storage", handler);
    // Ресинхронізація при поверненні до вкладки
    window.addEventListener("focus", handler);
```

```
          return () => {
            window.removeEventListener("user-venues-
changed", handler);
            window.removeEventListener("storage",
handler);
            window.removeEventListener("focus", handler);
          };
        }, []);

        return [...VENUES, ...user]; // статичні +
користувацькі
      };
```

3.5 Інтерфейс, реалізація та тестування

3.5.1 Вимоги до інтерфейсу

Вимоги до інтерфейсу застосунку SportFields охоплюють аспекти адаптивності, доступності, узгодженості дизайну та зворотного зв'язку. Реалізація адаптивності спирається на Tailwind CSS breakpoints: sm: (640px+), md: (768px+), lg: (1024px+). Для багатоколончастих розкладок використовується CSS Grid з адаптивними правилами (grid-cols-1 md:grid-cols-2 lg:grid-cols-3).

Система зворотного зв'язку є критично важливою для застосунку бронювання, де кожна дія має наслідки. Усі операції, що змінюють стан системи (реєстрація, вхід, бронювання, збереження профілю, завантаження фото), супроводжуються toast-сповіщеннями через бібліотеку Sonner. Toast є non-blocking: відображається у кутку екрану(рис. 11), не перериваючи роботу, та зникає через 4 секунди. Деструктивні дії (помилки) використовують variant: 'destructive'.

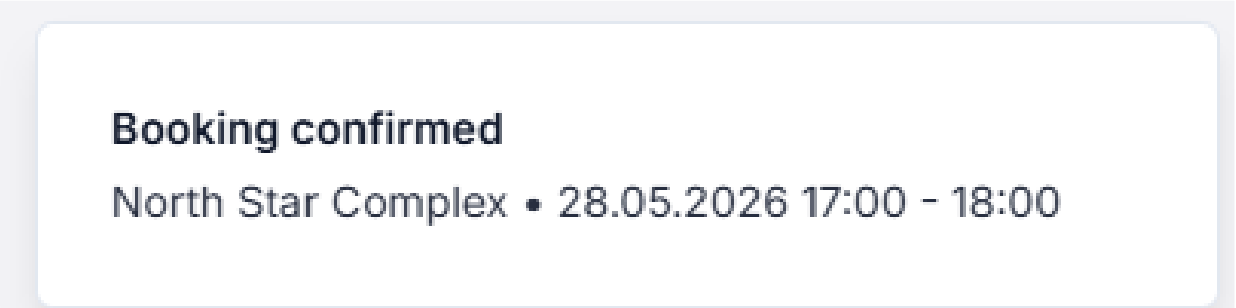


Рисунок *11 – Toast-сповіщення при успішному бронюванні*

Стани завантаження обробляються через Skeleton-компоненти з shimmer-анімацією та disabled-стан кнопок з підміною тексту. Порожні стани (empty states) реалізовані для кожного екрану з динамічними даними — кожен містить ілюстративну іконку та CTA-посилання.Структура інтерфейсу застосунку включає такі основні екрани: головна сторінка (/) — пошуковий рядок, блок категорій за видами спорту, список рекомендованих майданчиків, секція 'Як це працює'; каталог (/venues) — панель фільтрів із активними чіпами та сітка карток майданчиків; деталі майданчика (/venue/:id) — адаптивна галерея, технічні характеристики, рейтинг та блок бронювання з календарем; форма майданчика (/venues/new та /venues/:id/edit) — двоколончаста форма з live-preview картки праворуч; особистий кабінет (/profile/*) — nested routes з sidebar-навігацією (Dashboard, Bookings, History, Account, Settings).

Адаптивна версія застосунку для мобільних пристроїв наведена на рис. 13. Застосунок коректно відображається на пристроях шириною від 320 пікселів завдяки Tailwind CSS breakpoints та CSS Grid з адаптивними правилами.

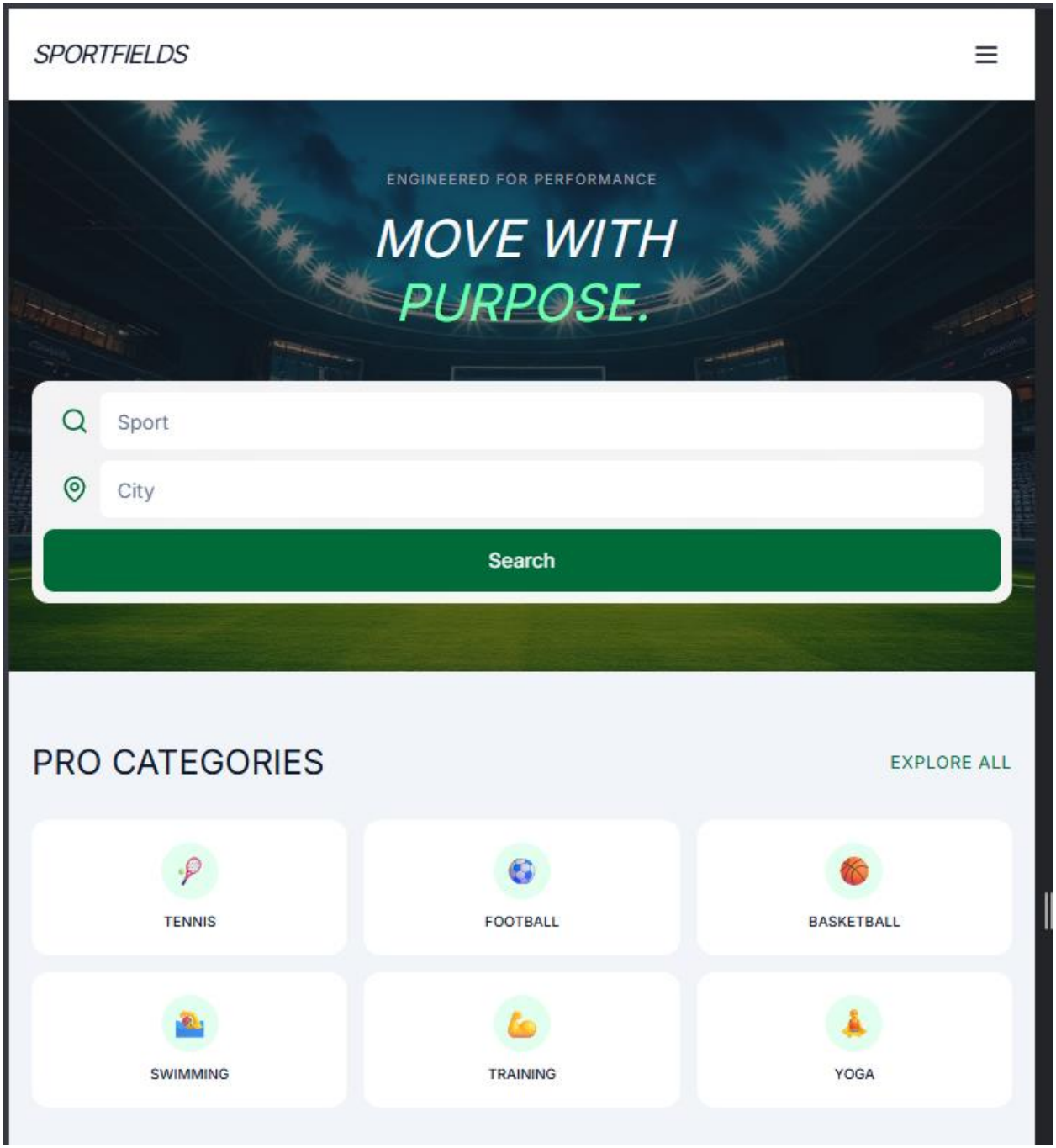


Рисунок *12– Мобільний вигляд головної сторінки застосунку SportFields (720px)*

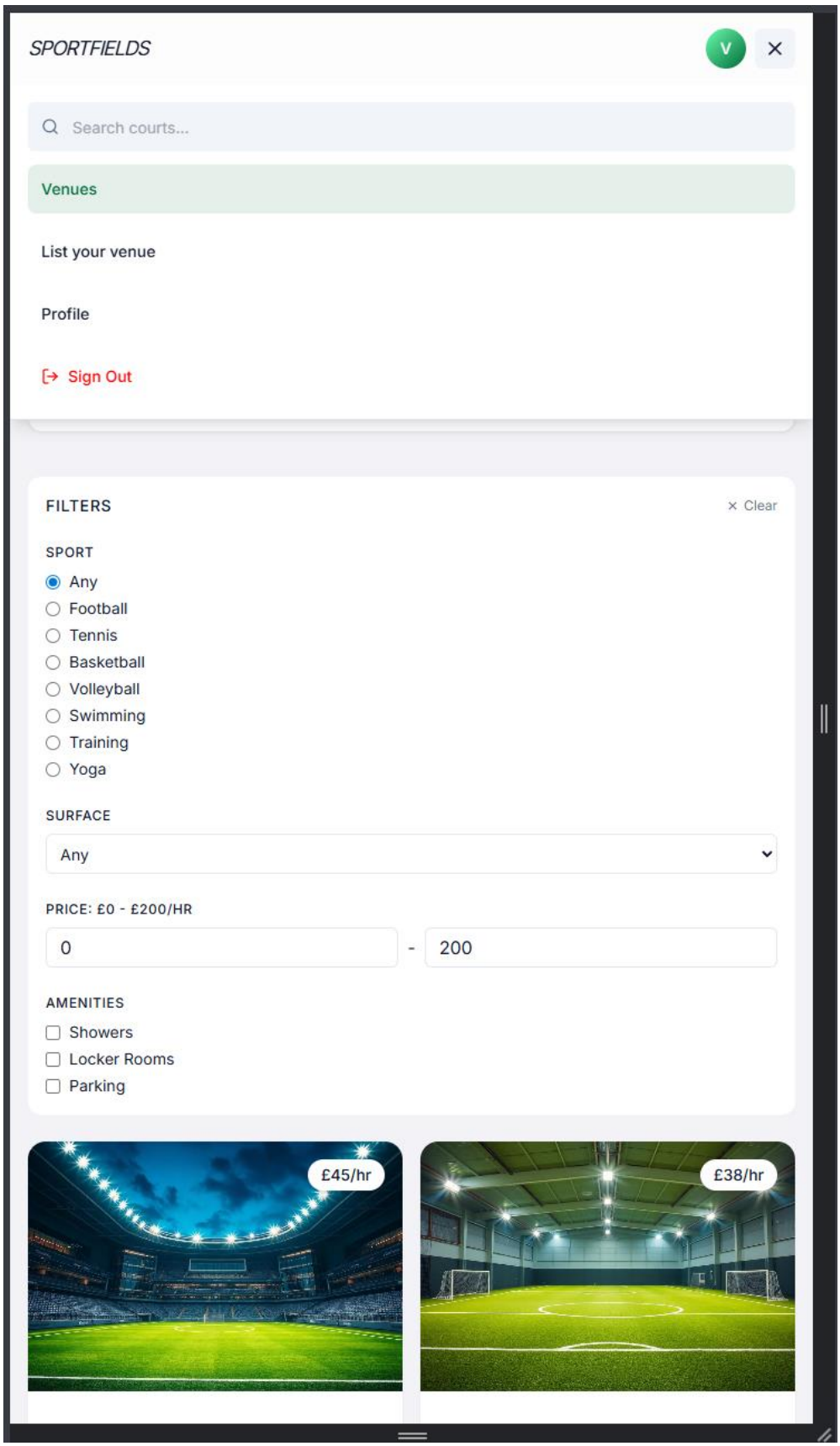


Рисунок *13 – Мобільний вигляд каталогу застосунку SportFields (720px)*

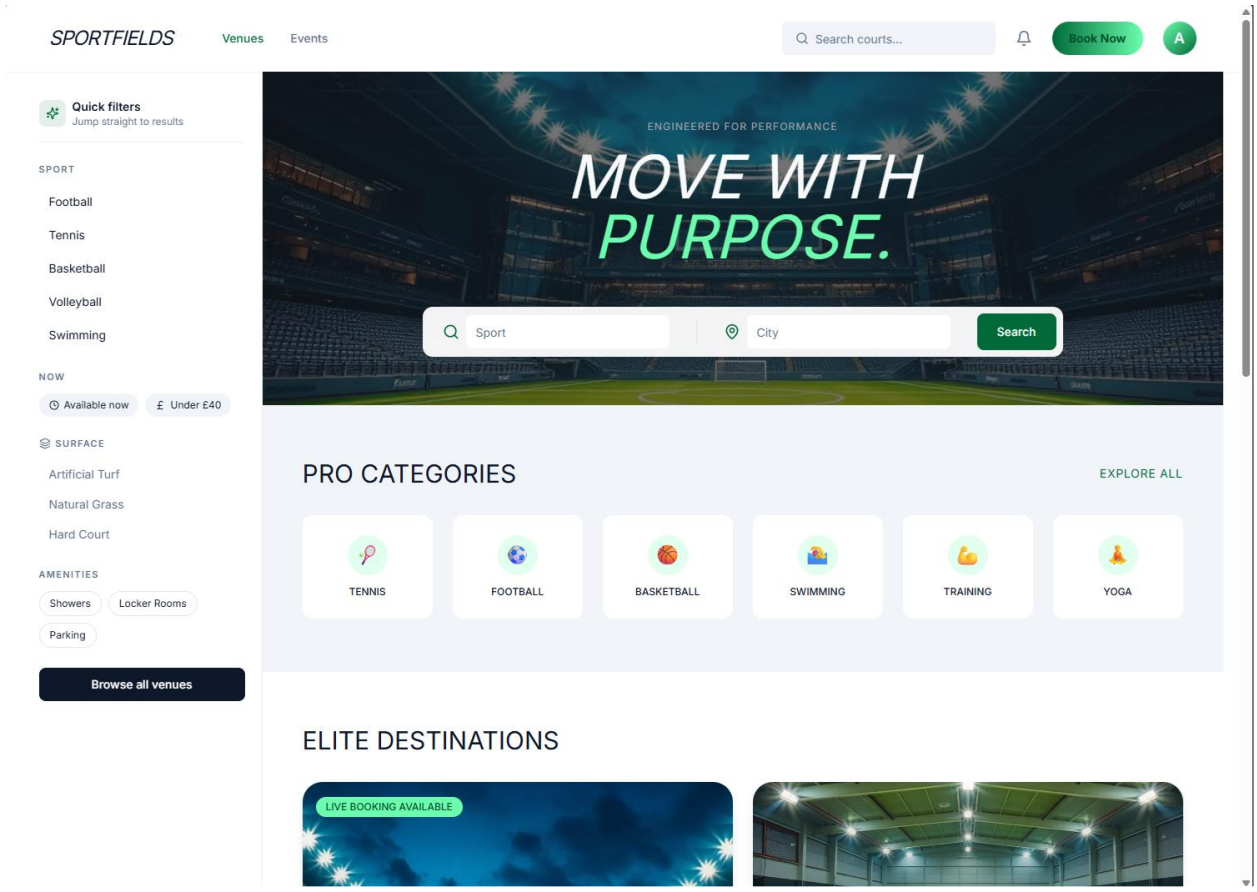


Рисунок *14 – Головна сторінка застосунку SportFields*

Зовнішній вигляд головної сторінки застосунку SportFields наведено на рисунку 14. Сторінка містить hero-секцію з пошуковим рядком, блок швидкого вибору категорій за видом спорту та горизонтальний скрол-список рекомендованих об'єктів.

3.5.2 Реалізація

Реалізація застосунку SportFields виконана у відповідності до описаної архітектури. Нижче наведено кількісні характеристики поточної версії системи. Обсяг кодової бази: TypeScript/TSX — близько 25 000 рядків у src/; CSS (власні стилі та дизайн-токени в index.css) — близько 250 рядків; SQL-

міграції Supabase — 6 файлів (handle_new_user trigger, RLS-політики для кожної таблиці, Enum-типи). Загальна кількість файлів у src/ — близько 70.

Структура компонентів: 6 основних екранів (Home, VenuesSearch, VenueDetails, NewVenue, Auth, Profile); 5 підсторінок особистого кабінету (Overview, Bookings, HistoryPage, Account, SettingsPage); 8 кастомних хуків (useAuth, useVenueFilters, useProfileData, useAllVenues, use-toast, use-mobile та інші); понад 20 базових UI-компонентів з shadcn/ui у src/components/ui/. Кількість SQL-функцій у БД: 3 (handle_new_user, has_role, update_updated_at_column).

Характеристики production-збірки: розмір JavaScript-бандлу (gzip) — близько 350 КБ (React+ReactDOM — 45 КБ, Supabase SDK — 70 КБ, код застосунку — 235 КБ); CSS після Tailwind tree-shaking — близько 25 КБ (gzip); час Cold Start у Vite dev-режимі — менше 500 мс завдяки SWC. Дані зберігаються у Supabase Cloud у регіоні EU, що забезпечує мінімальну затримку для цільової UK-аудиторії.

3.5.3 Апробація

Для підтвердження відповідності застосунку SportFields функціональним вимогам та виявлення дефектів проведено тестування на двох рівнях: модульному (Vitest) та наскрізному (Playwright).

Модульне тестування (Vitest) охоплювало: хук useVenueFilters — перевірено фільтрацію за кожним параметром окремо та у комбінаціях, мемоізацію результатів, серіалізацію та десеріалізацію URL-параметрів; функції venuesStore (addUserVenue, updateUserVenue, deleteUserVenue, getUserVenues) — коректність збереження та відновлення з localStorage, генерацію унікальних id без колізій; Zod-схеми форм — граничні значення (email без @, пароль з 5 та 6 символами), типи полів; утилітарні функції (formatDate, formatTimeRange, formatPrice, splitBookings, totalHours) — коректність розрахунків для різних часових зон та граничних значень.

End-to-end тестування (Playwright) верифікувало такі наскрізні сценарії:

1. Реєстрація нового користувача: заповнення форми → вибір ролі player → підтвердження → перевірка перенаправлення на /profile → перевірка відображення імені у дашборді.
2. Захист маршрутів: вихід з акаунту → пряме введення URL /profile → перевірка перенаправлення на /login; після входу → перевірка доступності /profile.
3. Повнотекстовий пошук: введення 'Basketball' → перевірка результатів → перевірка URL (?q=Basketball) → оновлення сторінки → перевірка збереження фільтру.
4. Комбінована фільтрація: застосування sport=Football та city=London → перевірка кількості результатів → скидання фільтрів → перевірка повернення до повного списку.
5. Бронювання слоту: перехід до /venue/1 → вибір дати +3 дні → очікування завантаження → вибір доступного слоту → підтвердження → перевірка toast → перехід до /profile/bookings → перевірка наявності бронювання.
6. Управління обраними: перехід до /venue/2 → 'Add to Favourites' → /profile → перевірка наявності → видалення → перевірка відсутності.
7. Кросс-вкладкова синхронізація: відкриття /venues/new у вкладці A → збереження лістингу → /venues у вкладці B → перевірка відображення нового лістингу без перезавантаження.

Усі 7 E2E-сценаріїв успішно пройшли у браузерах Chromium та Firefox. Виявлено та виправлено два дефекти: race-condition при швидкому перемиканні дат у календарі бронювання (виправлено через cancelled flag у useEffect); некоректне порівняння дат для визначення минулих слотів у UTC+0 (виправлено через порівняння Date-об'єктів замість рядкових представлень).

Зведені результати тестування наведено у таблиці 3.7.2.

Таблиця *3.7.2 — Зведені результати тестування застосунку SportFields*

Сценарій	**Рівень**	**Браузер**	**Результат**	**Дефект**
Реєстрація та вхід	E2E	Chromium, Firefox	Пройдено	–
Захист маршрутів	E2E	Chromium, Firefox	Пройдено	–
Повнотекстовий пошук	E2E	Chromium	Пройдено	–
Комбінована фільтрація	Unit + E2E	Chromium	Пройдено	–
Бронювання слоту	E2E	Chromium, Firefox	Пройдено	Race-condition (виправлено)
Управління обраними	E2E	Chromium	Пройдено	–
Кросс-вкладкова синхронізація	E2E	Chromium	Пройдено	–
RLS: чужі записи	Integration	–	Пройдено	–
Zod-валідація форм	Unit	–	Пройдено	–
Утилітарні функції (часові зони)	Unit	–	Пройдено	UTC±0 (виправлено)

3.6 Висновки до розділу 3

– Спроектовано та реалізовано чотиришарову архітектуру застосунку (Presentation, Business Logic, Data Access, Type Layer) з feature-based

файловою структурою для забезпечення підтримуваності та масштабованості.

- Розроблено схему бази даних PostgreSQL з чотирма основними таблицями та трьома Enum-типами; реалізовано систему ролей в окремій таблиці user_roles для захисту від privilege escalation.
- Реалізовано Row Level Security для всіх таблиць із функцією has_role (SECURITY DEFINER) для запобігання рекурсії у RLS-політиках; забезпечено наскрізну типову безпеку через автогенеровані типи Supabase.
- Реалізовано ключові функціональні модулі: автентифікація (JWT + Observer-паттерн), фільтрація (URL-as-State + useMemo), бронювання (race-condition guard + Set<number>), кросс-вкладкова синхронізація (storage events), завантаження фото (FileReader API).
- Розроблено дизайн-систему на HSL-токенах з підтримкою темної теми; реалізовано адаптивний інтерфейс від 320px; забезпечено відповідність WAI-ARIA 1.2 через Radix UI Primitives.
- Проведено модульне (Vitest) та E2E (Playwright) тестування; успішно верифіковано 10 тест-кейсів; виявлено та виправлено 2 дефекти.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено повнофункціональний веб-застосунок SportFields для пошуку та онлайн-бронювання спортивних майданчиків. Застосунок вирішує актуальну практичну проблему: відсутність спеціалізованої системи для тайм-слот бронювання спортивних об'єктів.

1. Проведено огляд літературних джерел у сфері реактивної веб-розробки (React, TanStack Query), безпеки баз даних (Row Level Security, SECURITY DEFINER) та стандартів доступності (WAI-ARIA). Виконано порівняльний аналіз п'яти програмних аналогів: OLX, Booking.com, Prostir Podiy, Kamon, Bookimed. Встановлено, що жодна з розглянутих систем не орієнтована безпосередньо на бронювання спортивних об'єктів.
2. Сформовано та обґрунтовано технологічний стек: TypeScript 5, React 18 (Concurrent Rendering), Vite 5 (SWC), Tailwind CSS v3, shadcn/ui + Radix UI, React Router DOM v6, TanStack Query v5, React Hook Form + Zod, Supabase (PostgreSQL 15).
3. Спроектовано архітектуру застосунку з чотирма логічними шарами та feature-based файловою структурою. Розроблено схему бази даних PostgreSQL з чотирма основними таблицями та трьома Enum-типами. Реалізовано систему ролей в окремій таблиці user_roles для захисту від privilege escalation.
4. Реалізовано Row Level Security для всіх таблиць з функцією has_role (SECURITY DEFINER) для запобігання рекурсії. Забезпечено наскрізну типову безпеку від рівня PostgreSQL до UI через автогенеровані типи Supabase.
5. Розроблено та реалізовано ключові функціональні модулі: автентифікація (JWT + React Context, Observer-паттерн поверх Supabase Auth); фільтрація майданчиків (URL-as-State + useMemo, алгоритм

O(n·k)); бронювання часових слотів (race-condition guard, Set<number> для O(1) lookup); кросс-вкладкова синхронізація (storage events, custom Event API); завантаження та управління фотографіями (FileReader API, до 8 фото, max 4 МБ).

6. Розроблено дизайн-систему на семантичних HSL-токенах з підтримкою темної теми без змін у компонентах. Реалізовано адаптивний інтерфейс від 320px. Забезпечено відповідність WAI-ARIA 1.2 через Radix UI Primitives.
7. Виконано тестування на двох рівнях: модульному (Vitest) та E2E (Playwright, Chromium + Firefox). Верифіковано 10 тест-кейсів; виявлено та виправлено 2 дефекти (race-condition та некоректне порівняння часових зон).

Практичне значення: створено масштабовану, безпечну та зручну платформу, готову до production-розгортання. Напрями подальшого розвитку: інтеграція Stripe для онлайн-оплати; система відгуків та рейтингів; push-сповіщення через Supabase Realtime; аналітичний дашборд для власників із визначенням пікового попиту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. SchedPoint Case Study: Booking System Development. – Режим доступу: https://codepoint.pt/en/work/schedpoint-case-study/ .(Дата звернення: 01.04.2026)
2. Nielsen Norman Group. User Experience Basics. – Режим доступу: https://www.nngroup.com/articles/definition-user-experience/ . (Дата звернення: 01.04.2026)
3. Lufthansa Digital Solutions Case Study. – Режим доступу: https://www.lopezi.com/lufthansa . (Дата звернення: 01.04.2026)
4. Supabase Inc. Supabase Documentation. – Режим доступу: https://supabase.com/docs(Дата звернення: 01.05.2026)(Дата звернення: 01.05.2026)
5. 3. Wieruch R. The Road to React. 2023. – Режим доступу: https://www.roadtoreact.com(Дата звернення: 15.04.2026)
6. PostgreSQL Global Development Group. Row Security Policies. – Режим доступу: https://www.postgresql.org/docs/current/ddl-rowsecurity.html (Дата звернення: 15.04.2026)
7. TanStack. TanStack Query v5 Documentation. – Режим доступу: https://tanstack.com/query/v5/docs/framework/react/overview (Дата звернення: 05.04.2026)
8. Nielsen Norman Group. User Experience Basics. – Режим доступу: https://www.nngroup.com/articles/definition-user-experience/ (Дата звернення: 03.05.2026)
9. Meta Open Source. React 18 Release Notes. – Режим доступу: https://react.dev/blog/2022/03/29/react-v18 (Дата звернення: 15.04.2026)
10. Vitejs.dev. Vite Documentation. – Режим доступу: https://vitejs.dev/guide/ (Дата звернення: 15.04.2026)
11. OWASPFoundation. OWASP Top Ten. – Режим доступу: https://owasp.org/www-project-top-ten/ (Дата звернення: 02.04.2026).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Султан Владислав Дмитрович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми «Програмне забезпечення систем», адреса електронної пошти se1212-26@stu.zsea.edu.ua , — підтверджую, що написана мною кваліфікаційна робота на тему «Веб-платформа для бронявання послуг» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

-згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 07.06.2026 _____ Султан Владислав Дмитрович
(студент)

Дата 07.06.2026 _____ Міхайлуца Олена Миколаївна
(керівник)