Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка програмного забезпечення сервісу замовлення та планування масових заходів

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань
Студентка гр. ІПЗ-21-1
_____ М.В. Куропятник

Керівник кваліфікаційної роботи _____ А.М. Стрюк

Завідувач кафедри _____ А.М. Стрюк

Кривий ріг
2025

Криворізький національний університет
Факультет: Інформаційних технологій
Кафедра: Моделювання на програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А.М. Стрюк
«___»_____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу

Студентці групи ІПЗ-21-1 Куропятник Марині Володимирівні

1. Тема: Розробка програмного забезпечення сервісу замовлення та планування масових заходів
   Затверджено наказом по КНУ № 199 від 10.04.2025
2. Термін подання студентом закінченої роботи 20.06.2025
3. Вихідні дані по роботі: мови програмування PHP та JavaScript, мова розмітки HTML, каскадна таблиця стилів CSS, система керування базами даних MySQL, локальний сервер Apache24.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): обрати напрямок рішення завдання, проаналізувати існуючі на ринку аналоги, спроектувати архітектуру програмного забезпечення, визначитися зі способами програмної реалізації проекту, описати ключові програмні рішення, проілюструвати основні форми та інтерфейс веб-сайту.
5. Перелік ілюстративного матеріалу: Діаграма варіантів використання, Контекстна діаграма DFD, Перший рівень декомпозиції для користувача-клієнта, Перший рівень декомпозиції для користувача-партнера, Таблиці бази даних eventbooker, Атрибути таблиць бази даних, Головна сторінка EventBooker, Налаштування для пошуку локацій, Форма входу, Форми реєстрації для користувачів, Головна сторінка партнера, Індикатор, Видалити акаунт.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Визначення теми кваліфікаційної роботи	27.11.2024
2	Вивчення літератури за темою	05.02.2025 - 20.02.2025
3	Планування напрямку рішення завдання, робота над архітектурою ПЗ	25.02.2025 - 10.03.2025
4	Програмна реалізація дипломного проекту	11.03.2025 - 13.04.2025
5	Затвердження результатів програмної реалізації проекту з керівником дипломної роботи	14.04.2025
6	Написання текстової частини роботи	20.04.2025 - 22.05.2025

Дата видачі завдання: «10» квітня 2025 р.

Студент _____ М.В. Куропятник

Керівник роботи _____ А.М. Стрюк

РЕФЕРАТ

ФОРМА, СЕРВЕР, ІНТЕРФЕЙС, ЗАХІД, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ДІАГРАМА, БАЗА ДАНИХ, ВЕБ-САЙТ.

Пояснювальна записка: 49 с., 3 табл., 13 рис., 2 дод., 50 джерел.

У роботі проведено аналіз існуючих веб-сервісів для бронювання приміщень, розглянуто їх архітектурні особливості, сильні та слабкі сторони, а також порівняно підходи до реалізації основних функцій: фільтрації, пошуку, персоналізації й управління даними користувачів. Визначено вимоги до сучасної системи бронювання, з урахуванням очікувань різних категорій користувачів – клієнтів та партнерів.

Для вирішення поставленої задачі обрано монолітну архітектуру та базові веб-технології: HTML, CSS, JavaScript, PHP і MySQL. Реалізовано основний функціонал: реєстрація та аутентифікація користувачів, пошук і фільтрація локацій за параметрами, система бронювання, можливість додавання і редагування власних локацій для партнерів, а також організовано персональні кабінети для різних типів користувачів.

У межах кваліфікаційної роботи спроєктовано структуру бази даних, розроблено інтерфейс користувача, описано процес обробки даних та забезпечено захист інформації на базовому рівні. Система пройшла тестування в локальному середовищі на сервері Apache, результати якого підтвердили стабільність роботи, зручність та відповідність функціоналу поставленим вимогам.

ABSTRACT

FORM, SERVER, INTERFACE, EVENT, SOFTWARE ARCHITECTURE, DIAGRAM, DATABASE, WEBSITE.

Thesis in 49 p., 3 tables., 13 fig., 2 app., 50 references.

The thesis provides an analysis of existing web services for venue booking, examines their architectural features, strengths and weaknesses, and compares approaches to implementing core functions such as filtering, searching, personalization, and user data management. The requirements for a modern booking system are identified, taking into account the expectations of different categories of users – clients and partners.

To address the set task, a monolithic architecture and basic web technologies – HTML, CSS, JavaScript, PHP, and MySQL – were chosen. The core functionality was implemented: user registration and authentication, location search and filtering by parameters, booking system, the ability for partners to add and edit their own locations, as well as personal accounts for different user roles.

As part of the qualification work, the database structure was designed, the user interface was developed, the data processing workflow was described, and a basic level of information security was ensured. The system was tested in a local environment on an Apache server, and the results confirmed its stable operation, user-friendliness, and compliance with the specified functional requirements.

ЗМІСТ

ВСТУП

Планування масових заходів – це процес, який потребує ретельної підготовки та організації. Шукати ідеальне місце для свята або ділової зустрічі, самостійно знайти спеціалістів, яких можна долучити до організації або проведення заходу – задача непроста. Я замислилася над тим як полегшити цей процес. Ми давно звикли до того, що більшість процесів можна автоматизувати та спростити, тож і планування різних подій в тому числі. При виборі місця, бронюванні, комунікації з організаторами необхідно, щоб усе працювало швидко та зручно. Саме тому у мене виникла ідея створити програмне забезпечення, яке б поєднувало всі ці можливості.

Моя дипломна робота присвячена розробці веб-застосунку EventBooker для замовлення та організації масових заходів. Основною метою я визначила створення інтуїтивно зрозумілої системи. Важливо не просто зробити сайт, а продумати його логіку, структуру взаємодії та способи масштабування у майбутньому. У результаті реалізовано базовий функціонал для гостей сайту та для зареєстрованих клієнтів та партнерів. Гості можуть переглядати доступні місця, список послуг та контактну інформацію, а після авторизації, клієнтам відкриваються додаткові можливості.

Загалом, проєкт став хорошою можливістю пройти шлях від ідеї до повноцінного програмного продукту. Робота над ним дозволила не лише закріпити знання у сфері веб-розробки, баз даних та проєктування інтерфейсів, а й краще зрозуміти принципи побудови зручного і масштабованого сервісу. Таку систему можна застосовувати в реальному бізнесі – наприклад, для організаторів заходів, власників просторів або компаній, які займаються організацією заходів.

1 ОБҐРУНТУВАННЯ НАПРЯМКУ РІШЕННЯ ЗАВДАННЯ

Останніми роками сайти для пошуку та бронювання місць для різних заходів стали популярними на іноземному ринку. Це не дивно, адже такі сервіси суттєво полегшують життя тим, хто планує заходи, та людям, які здають приміщення в оренду. Саме тому виникла ідея створити власний веб-сайт, де будь-хто може швидко знайти потрібну локацію для події, легко її забронювати та без зайвих проблем домовитися з власником. Основним завданням було зробити інтерфейс максимально простим, інтуїтивним і зручним, без складних меню чи додаткових функцій, які могли б заплутати користувачів.

Щоб сайт справді був корисним і зручним, я спочатку ретельно розглянула, що саме шукають і очікують користувачі від подібних сервісів. Завдяки цьому вдалося визначити ключові функції, які варто включити до розробки. Це пошук місць проведення заходів, можливість їх бронювання, перегляд своїх бронювань, додавання нових локацій, а також можливість отримати контактну інформацію інших учасників сервісу.

Перший етап роботи полягав у створенні діаграм, які наочно показували, як саме користувачі будуть взаємодіяти з сайтом. Завдяки цим схемам вдалося визначити дві основні групи користувачів: «Клієнтів», які шукають і бронюють місця, і «Партнерів», власників локацій. Для клієнтів було створено можливість швидко знаходити приміщення, бронювати їх та замовляти додаткові послуги. Партнери ж отримали зручний інструмент для додавання власних приміщень, перегляду бронювань та отримання контактних даних своїх клієнтів.

Особливу увагу я приділила функції пошуку. На сайті є два варіанти пошуку: простий і розширений. Простий варіант добре підходить тим, хто ще не визначився із конкретними вимогами і хоче просто переглянути всі

доступні варіанти. Розширений же пошук дає змогу вказати точні параметри, наприклад, тип події, що значно спрощує процес вибору та економить час.

У процесі розробки свідомо не використовувалися готові фреймворки чи спеціалізовані інструменти. Весь код було написано на основі базових технологій: HTML, CSS, JavaScript і PHP. HTML і CSS дали змогу створити зрозумілий та приємний зовнішній вигляд сайту. Завдяки JavaScript сторінки стали інтерактивними, з можливістю миттєвої реакції на дії користувача без зайвого перезавантаження. PHP взяла на себе всю серверну частину роботи, від обробки запитів і генерації сторінок до взаємодії з базою даних MySQL.

Завдяки обраним інструментам сайт вийшов простим, стабільним та зрозумілим. Також важливим моментом стала можливість легко тестувати весь функціонал на локальному сервері Apache. Це дозволило швидко знаходити і виправляти помилки, забезпечуючи стабільну роботу сервісу ще до того, як він стане доступним широкому загалу.

У результаті вийшов сучасний, приємний для користувачів веб-сайт, який дозволяє швидко знаходити потрібні місця, легко бронювати їх та ефективно комунікувати між клієнтами та власниками локацій. Саме такий простий, зрозумілий і зручний сервіс був початковою метою цієї розробки.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Огляд вже існуючих рішень – це, по суті, стартова точка будь-якого серйозного проєкту. Без такого аналізу важко зрозуміти, в якому напрямку рухатися і чим дійсно можна відрізнятися від конкурентів. Саме тому я присвятила час детальному знайомству з популярними сайтами та сервісами, що так чи інакше працюють із бронюванням місць, організацією подій чи комунікацією між власниками простору і потенційними клієнтами. У процесі дослідження хотілося не просто знайти, що в кого працює добре, а й побачити слабкі місця, на які користувачі часто скаржаться. Особливо важливо було оцінити зручність інтерфейсу, швидкість пошуку, спосіб подачі інформації, логіку авторизації та реєстрації, а також варіанти персоналізації сервісу під різні групи користувачів. Такий аналіз дозволив виділити для себе найкращі практики та ідеї, які можна використати у власній роботі, а ще – зрозуміти, чого варто уникати. У цьому розділі докладно описую результати вивчення архітектури програмного забезпечення й функціоналу деяких популярних платформ, які так чи інакше пов’язані з моєю темою. Особливу увагу звертаю на ті рішення, що справді вплинули на моє бачення структури й можливостей майбутнього продукту.

2.1 Дослідження архітектури ПЗ

Перш ніж створювати власний сайт, було цікаво подивитися, як організовано роботу у вже відомих і великих проектах. Я не просто вивчала їх із точки зору користувача, а намагалася розібратися, як всередині побудовані їхні програмні компоненти: як зберігаються й обробляються дані, як відбувається зв’язок між клієнтською та серверною частиною, як реалізована робота з базою даних і що відбувається у момент взаємодії користувача з інтерфейсом. Також звертала увагу на те, наскільки легко масштабувати систему, чи є можливість додавати нові модулі й функції без кардинальних змін у коді. У процесі аналізу хотілося зрозуміти, які підходи найкраще

підходять саме для веб-сервісу з бронювання місць, де все має працювати максимально просто, швидко і без перебоїв. Особливо цікаво було порівняти, наскільки відрізняється архітектура у великих комерційних сервісів, де працює команда розробників, і у менших сайтів, які часто підтримує одна людина. Це дало змогу визначити, що для мого проєкту більше підходить класичний монолітний підхід із чітким розділенням логіки на окремі компоненти, але без зайвого ускладнення структури. У підсумку, завдяки цьому дослідженню вдалося підібрати оптимальні ідеї для побудови власної архітектури й уникнути багатьох типових помилок.

2.1.1 електронна дошка оголошень ОЛХ

OLX – це, мабуть, один із найвідоміших українських онлайн-сервісів, через який кожен бодай раз щось купував, продавав чи шукав. Саме цей проект першим спав на думку, коли йшлося про аналіз аналогів, бо його основна ідея та логіка дуже перегукуються з тим, що я хотіла реалізувати у своєму веб-сайті. На OLX можна знайти найрізноманітніші товари, послуги й пропозиції з оренди приміщень, а система взаємодії між користувачами (продавцями і покупцями) схожа на модель «клієнт–партнер», яку я запровадила для свого проекту. Результати дослідження у таблиці 1.2:

Таблиця 1.2 - Аналіз архітектури ОЛХ

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Клієнтська частина	Пошук і фільтрація оголошень	Продавці змушені вказувати лише одну категорію, хоча товар може відповідати кільком, через що покупці можуть його не знайти в інших розділах.	Багато зручних фільтрів за різноманітними критеріями. Шукати товари швидко та зручно

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Клієнтська частина	Створення, редагування, видалення оголошень	Явні недоліки відсутні	Зрозумілий інтерфейс роботи з оголошеннями
	Сторінки авторизації та профілю користувача	Можливі вразливості при ненадійному зберіганні паролів	Зручно відстежувати власні оголошення та повідомлення
	Месенджер для спілкування між продавцем і покупцем	Явні недоліки відсутні	Назва чату відповідає назві товару. Зручно для продавця
Серверна частина	Авторизація і автентифікація	Явні недоліки відсутні	Особливі переваги над аналогами відсутні
	CRUD-операції для оголошень	Якщо відсутня перевірка контенту, це відкриває шлях до небажаних матеріалів	Особливі переваги серед аналогів відсутні
	Модерація контенту	Продавець не може редагувати невідповідне оголошення, лише створити нове	Вчасне реагування на порушення

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Серверна частина	Платежі	Недостатній захист платіжних даних може призвести до витоку реквізитів банківських карток	Фінансовий партнер ОЛХ сертифікований за стандартами безпеки VISA/Mastercard – найпоширеніші платіжні системи

2.1.2 Peerspace

Peerspace – це сучасний онлайн-сервіс, який спеціалізується на пошуку і бронюванні найрізноманітніших локацій: від фотостудій і креативних просторів до залів для корпоративів чи святкових подій. У процесі дослідження цього сайту звернула увагу, що основна архітектура проєкту побудована довкола швидкого і зручного підбору потрібного приміщення. Для цього реалізовано продуману систему фільтрів за типом події, місткістю, ціною, наявністю додаткових опцій чи навіть відгуками попередніх гостей. У Peerspace кожна локація має докладний опис, фотографії, актуальний календар бронювань і чітко позначені правила користування. Вразило те, що для кожної сторони – як для орендарів, так і для власників приміщень – створено окремі особисті кабінети, де зручно керувати своїми замовленнями, повідомленнями та історією бронювань. Орендар може додати локацію у вибране, переглянути деталі, зв’язатися з власником безпосередньо або залишити відгук після проведеної події. Паралельно власники приміщень отримують повний контроль над своїми пропозиціями, можуть редагувати опис, встановлювати

ціни, відповідати на запити та отримувати аналітику щодо своїх бронювань. Такі функції й структурованість Peerspace стали для мене реальною основою для натхнення: саме ідея персональних кабінетів, інтерактивних фільтрів та зручної комунікації між учасниками платформи лягла в основу моїх власних підходів до розробки функціоналу EventBooker. Результати дослідження у таблиці 2.2:

Таблиця 2.2 - Аналіз архітектури Peerspace

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Клієнтська частина	Перегляд приміщень з фільтрами	Явні недоліки відсутні	Багато категорій для будь-якої активності. Зручний пошук
	Бронювання на конкретну дату	Можливість підробки запиту при відсутності перевірки автентичності	Форма бронювання має поля, що уточнюють всі деталі. Зручно орендодавцю
	Перегляд історії замовлень	Дані можуть кешуватись у браузері без належного контролю	Легко знайти місце для повторного бронювання

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Клієнтська частина	Надсилання повідомлень	Явні недоліки відсутні	Особливі переваги серед аналогів відсутні
	Авторизація та реєстрація	Явні недоліки відсутні	Особливі переваги серед аналогів відсутні
Серверна частина	Збереження даних про об’єкти	Ризик SQL-ін’єкцій без параметризованих запитів	Надійне централізоване зберігання інформації
	Обробка запитів на бронювання	Якщо є затримка при оновлені, користувач може робити подвійні бронювання	Автоматизація процесу резервування
	Система повідомлень	Наявність зловмисного контенту у повідомленнях без фільтрації	Покращення взаємодії між користувачами
	Інтеграція платіжної системи	Недостатній захист платіжних даних – витік реквізитів	Безпечні транзакції через сторонні перевірені сервіси

2.1.3 Airbnb

Airbnb – один із найвідоміших сервісів для пошуку й бронювання житла по всьому світу, і хоча основна ідея цього проєкту орієнтована на короткострокову оренду квартир чи будинків, у його роботі є багато моментів, які можна застосувати й у сфері подій. Найбільше мене зацікавила система зворотного зв'язку та рейтингу, яка мотивує і власників, і гостей бути відповідальними й чесними у взаємодії. Окрім цього, в Airbnb дуже добре продумана комунікація – завжди можна швидко написати власнику, уточнити деталі чи домовитися про додаткові послуги. Сервіс активно використовує повідомлення, сповіщення і зручний особистий кабінет для обох сторін. Вивчаючи цю платформу, я зрозуміла, наскільки важливо для користувачів мати простий, зрозумілий і безпечний механізм спілкування, а також змогу контролювати свої бронювання без зайвих кроків. Результати дослідження у таблиці 3.2:

Таблиця 3.2 - Аналіз архітектури Airbnb

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Клієнтська частина	Авторизація та реєстрація	Явні недоліки відсутні	Можливість входу за допомогою соцмереж, підтримка двофакторної автентифікації
	Пошук житла	Явні недоліки відсутні	Зручна інтерактивна карта та система фільтрів

Компонент	Основні функції	Недоліки продукту/поширені вразливості в аналогах	Переваги
Клієнтська частина	Бронювання житла	Явні недоліки відсутні	Автоматична перевірка доступності
	Обмін повідомленнями	Явні недоліки відсутні	Особливі переваги серед аналогів відсутні
Серверна частина	CRUD-операції для оголошень	Якщо відсутня перевірка контенту, це відкриває шлях до небажаних матеріалів	Особливі переваги серед аналогів відсутні
	Обробка бронювання	Без перевірки доступності дати – можливі конфлікти бронювання	Автоматизація процесу резервування
	Система повідомлень	Наявність зловмисного контенту у повідомленнях без фільтрації	Покращення взаємодії між користувачами

У результаті проведеного аналізу і детального порівняння з уже існуючими рішеннями, вдалося чітко визначити перелік основних компонентів, які мають бути обов’язково реалізовані у моєму програмному продукті. Це не лише стандартні функції, такі як пошук і бронювання місць, а

й усі ті деталі, які забезпечують справжню зручність і ефективність для кінцевого користувача. Важливим моментом стало також розуміння того, як саме повинні взаємодіяти окремі частини системи між собою, щоб процес користування був простим, а виконання основних дій – максимально швидким та інтуїтивно зрозумілим. На основі отриманих висновків були сформульовані принципи реалізації ключових блоків програми, з урахуванням кращих практик і уникаючи типових помилок, з якими стикаються інші подібні сервіси. Це стосується як структури бази даних, так і логіки взаємодії між клієнтською та серверною частиною, організації особистих кабінетів для різних типів користувачів, а також механізмів захисту і перевірки даних. Весь цей комплекс напрацювань, ідей та рішень став надійною основою для визначення вимог до функціоналу системи. Саме на базі цих висновків було заплановано структуру майбутнього веб-застосунку, продумано його основні сценарії використання та обрано підходи до розробки, які дозволять надалі легко масштабувати та вдосконалювати систему залежно від нових потреб і очікувань користувачів.

2.2 Дослідження навігаційної структури сайтів

Один з ключових факторів успішності програмного продукту – це зручність та зрозумілість у використанні. Саме тому під час розробки власного веб-сайту я багато уваги приділила вивченню сучасних тенденцій у сфері дизайну й організації інтерфейсів. Зараз більшість користувачів звикли до простих і зрозумілих веб-платформ, тому очікують, що новий сервіс буде не лише привабливим зовні, а й інтуїтивно легким у користуванні. Я проаналізувала різноманітні підходи до побудови інтерфейсів, вивчила, як подають інформацію та організовують дії на великих платформах, і намагалася взяти найкраще з побаченого.

Ось деякі поширені принципи побудови інтерфейсу:

1) Мінімалізм та простота;
2) Однотипні шаблони дій;

Кнопки мають схожий вигляд та однаково розташовуються на формах. Форми мають однакову логіку;

3) Вся інформація не показується одразу;

Для легшого сприйняття інформації, додаткові опції та дані можуть бути відображені на окремих веб-сторінках або модальних вікнах.

Особливу увагу я приділила вивченню інтерфейсів таких сайтів, як airbnb.com.ua, peerspace.com та olx.ua. Ці сервіси – справжні лідери у своїй галузі, тому не дивно, що їхні рішення багато в чому стали стандартом для користувачів по всьому світу. Їхні сайти майже досконалі з точки зору структури, візуальної привабливості та логіки подачі даних. Водночас, навіть у таких великих проєктах можна помітити певні недоліки, з якими стикаються користувачі у реальному житті.

Наприклад, головна сторінка airbnb.com.ua дійсно містить багато корисної інформації, але через це частина важливих опцій або даних може залишитися непоміченою. Іноді користувач просто не знає, що є додаткові фільтри, опції чи цікаві пропозиції, якщо вони не потрапили в зону його уваги на першому екрані. Це питання структурування даних і подачі інформації, і я зробила для себе висновок, що краще винести окремі розділи або додаткові опції на підменю чи виділити для них окремі сторінки. Таким чином, кожна сторінка стає менш перевантаженою, і користувач може спокійно ознайомитись із усіма можливостями сайту, не відчуваючи навантаження від великої кількості інформації на головній сторінці. Це також підвищує ймовірність того, що користувач знайде саме те, що шукав, і не загубиться у різноманітті функцій.

Ще одна важлива деталь стосується OLX. На цьому сервісі під час зміни підкатегорії товару фільтри, які були налаштовані раніше, скидаються. На практиці це створює певні незручності, особливо якщо покупець хоче швидко знайти потрібний товар, але змушений щоразу заново виставляти фільтри. Такі дрібні речі можуть впливати на загальне враження від користування сайтом і навіть відштовхувати людей, які цінують свій час. Після знайомства з цим

кейсом я для себе вирішила, що потрібно подбати про збереження обраних налаштувань або, щонайменше, зробити так, щоб фільтрація завжди працювала послідовно і зрозуміло для користувача.

В цілому, аналізуючи чужі рішення, я намагалася звертати увагу не лише на зовнішній вигляд і анімації, а й на ті дрібниці, які справді впливають на комфорт користування платформою. Для мене важливо було, щоб сайт не лише красиво виглядав, а й залишався логічним і простим у навігації. Саме тому у власному проєкті я від початку будувала структуру так, щоб кожен розділ мав чітке призначення, не було зайвого дублювання інформації, а всі ключові дії можна було виконати буквально у кілька кліків. Я намагалася уникати типових помилок, які траплялися на інших ресурсах, і зробити інтерфейс максимально дружнім до користувача незалежно від його досвіду чи технічних навичок.

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПЗ

Після дослідження предметної області та пропозицій на ринку, наступним важливим кроком роботи є проєктування програмного забезпечення. Моя робота має містити наступні основні програмні функції:

1) Реєстрація та авторизація користувачів;
2) Розмежування функціоналу сайту за ролями;
3) Бронювання місця для заходу;
4) Пошук місця з можливістю використання фільтру;
5) Додавання та редагування оголошення;
6) Перегляд заброньованих місць з можливістю використання календаря.
7) Ознайомлення з додатковими послугами для клієнтів;

3.1 Вибір архітектури програмного забезпечення

Під час планування того, як доцільно побудувати архітектуру мого проєкту, я стикнулася з вибором між трьома способами це зробити, які включають монолітну архітектуру, модульну та архітектуру з дублюванням функцій.

3.1.1 Монолітна архітектура

Монолітна архітектура – це традиційний підхід, за якого весь застосунок створюється як єдина цілісна система. Усі його компоненти тісно інтегровані та розгортаються разом на одному сервері або кластері серверів. Така архітектура має ряд переваг: вона відносно проста у реалізації й відладці, адже всі частини взаємодіють безпосередньо, що дозволяє швидко виявляти та усувати неполадки. Управління монолітним застосунком також менш складне, оскільки вся логіка працює в межах єдиної платформи. Крім того, моноліт дозволяє швидко розпочати розробку – завдяки своїй цілісності проєкт легше зрозуміти і можна одразу переходити до кодування без налаштування взаємодії між службами.

Втім, у міру зростання проєкту монолітна архітектура може породжувати труднощі. По-перше, виникають обмеження масштабованості: розширити систему важко, оскільки всі компоненти залежні один від одного. Додавання нових функцій у великий моноліт уповільнюється, адже будь-яка зміна може зачепити інші частини коду. Є ризик, що помилка в одному модулі здатна порушити роботу всього застосунку. По-друге, підтримувати чітку модульність у межах моноліту з часом дедалі складніше. Спільна кодова база неминуче призводить до порушення принципів низького зв’язку та високої когерентності: із ростом системи компоненти моноліту стають тісніше пов’язаними, а їхня відповідальність розмивається. Це ускладнює тестування й супровід: зміна однієї частини моноліту може вимагати перевірки всього застосунку перед релізом.

Попри зазначені недоліки, монолітна архітектура залишається доцільною для відносно невеликих або середніх проєктів зі стабільним набором функцій. У таких випадках переваги простоти переважають потребу в масштабуванні. Зокрема, для невеликого веб-сайту, розробленого самостійно без використання фреймворків, монолітний підхід є практично виправданим і зручним. Розробник може зосередитися на бізнес-логіці, не витрачаючи ресурсів на налагодження взаємодії між сервісами або розподілену інфраструктуру. В межах моноліту навіть без фреймворку можна реалізувати базову структуру на кшталт Model-View-Controller: наприклад, мати єдиний вхідний скрипт (front controller), що аналізує запит і передає управління відповідним модулям-«контролерам»; використовувати окремі файли чи класи для роботи з даними (моделі) та шаблони представлення (види). Такий підхід забезпечує певне розділення відповідальності всередині моноліту і полегшує супровід. При цьому вся функціональність залишається в одному проєкті, що спрощує розгортання та підтримку. Достатньо оновити та запустити один застосунок на сервері, щоб внести зміни. Таким чином, для одиночного розробника маленького сайту монолітна архітектура дозволяє швидко отримати робочий результат з мінімальними накладними витратами.

3.1.2 Модульна архітектура з багатьма компонентами

Модульна архітектура, що складається з багатьох компонентів, передбачає поділ системи на ряд незалежних частин із чіткими взаємозв’язками. Найпоширенішим прикладом такого підходу є мікросервісна архітектура. У цій моделі єдиний застосунок розбивається на набір дрібніших автономних сервісів, кожен з яких відповідає за конкретну функціональність і взаємодіє з іншими через прості інтерфейси (API). По суті, кожен компонент є окремим модулем з власним набором завдань, слабко пов’язаним з іншими.

Подібна багато-компонентна архітектура забезпечує низку суттєвих переваг. По-перше, вона покращує ізоляцію збоїв: збій одного модуля не паралізує всю систему, тому великий застосунок може продовжувати роботу, навіть якщо один зі сервісів відмовив. Це підвищує надійність системи в цілому. По-друге, кожен компонент можна масштабувати незалежно. Якщо певна частина навантажена більше (наприклад, модуль обробки зображень), її можна запустити в кількох екземплярах, не масштабуючи всю систему. Такий підхід оптимізує використання ресурсів і спрощує горизонтальне масштабування. По-третє, полегшується розробка і розгортання оновлень: окремий сервіс можна змінювати, не торкаючись інших модулів. Команди розробників можуть працювати над різними сервісами паралельно, без конфліктів у загальному коді. Більше того, з’являється можливість технологічної різноманітності: кожен сервіс за потреби може бути реалізований на відповідній мові чи з використанням оптимального технологічного стеку, незалежно від інших. Наприклад, один модуль може використовувати PHP, інший – Python, якщо це дає переваги для його задач. Така гнучкість спрощує інтеграцію сторонніх рішень і дозволяє вибирати найефективніші інструменти для кожного компонента.

Однак архітектура з багатьох компонентів має і недоліки, які потрібно врахувати. Найперше – вища складність системи. Розробникам потрібно продумати чіткі межі відповідальності сервісів та способи їх взаємодії. Комунікація між численними сервісами ускладнює налагодження і

тестування. Необхідно врахувати безліч сценаріїв відмов (недоступність сервісу, мережеві затримки тощо). Виникає додаткова затримка при обміні даними між модулями – навіть швидкі мережеві виклики додають додаткове навантаження. Якщо для виконання запиту користувача потрібно послідовно звернутися до десятка мікросервісів, час відповіді може зрости на сотні мілісекунд або й більше. Частково цю проблему вирішують шаблони на кшталт «API Gateway» або використання GraphQL, проте вони лише приховують складність інтеграції, не усуваючи її повністю. По-друге, зростають вимоги до інфраструктури: потрібно розгорнути і підтримувати в робочому стані багато сервісів, налаштувати для них середовище (сервери або контейнери, бази даних для кожного, систему контейнеризації тощо). Це веде до збільшення витрат на інфраструктуру й DevOps-супровід проєкту. Для коректної роботи системи необхідно впроваджувати засоби моніторингу, централізованого логування, механізми резервування на рівні сервісів – усе це додає складності в порівнянні з монолітом. Нарешті, поріг входження в систему з мікросервісною архітектурою вищий: нові члени команди мають орієнтуватися в розподіленій системі, розуміти протоколи взаємодії, що може потребувати додаткового часу на навчання.

Вибираючи багатокомпонентну архітектуру, слід брати до уваги масштаб і потреби проєкту. Такий підхід виправданий головним чином для великих і складних систем, що мають перспективу росту і високі вимоги до масштабованості та доступності. У подібних випадках вигоди (гнучкість, надійність, масштабування) перевищують накладні витрати на підтримку складної структури. Натомість для невеликих веб-сайтів або прототипів введення мікросервісів може бути зайвим. Фахівці відзначають, що попри привабливість мікросервісів, іноді краще залишитися на моноліті – особливо якщо команда не має достатнього досвіду з розподіленими системами. Перехід «маленького» проєкту на мікросервісну архітектуру без нагальної потреби здатен лише додати проблем і ускладнити розробку. Таким чином, модульна

багатосервісна архітектура є потужним інструментом, але застосовувати його варто тоді, коли складність проєкту це виправдовує.

3.1.3 Архітектура з дублюванням функцій

Архітектура з дублюванням функцій передбачає, що певні важливі підзадачі в системі виконуються кількома компонентами паралельно або дублюються в різних місцях. Іншими словами, вводиться надлишковість: кілька компонентів відповідають за вирішення однієї і тієї ж задачі. Мета цього підходу – підвищити надійність і безвідмовність роботи програмного забезпечення. Якщо один компонент виходить з ладу або дає некоректний результат, інший (дублюючий) компонент все одно зможе виконати потрібну функцію, таким чином система продовжить працювати. Подібні рішення часто застосовуються у критичних до відмов системах: наприклад, в банківських застосунках, авіаційних чи космічних програмах, де неприпустимий навіть короткочасний збій. У таких випадках дублювання може бути реалізоване як паралельне виконання (два чи більше екземпляри компонента працюють одночасно і результати порівнюються або використовується резервний у разі відмови основного) або як резервування «гарячого»/«холодного» standby, коли один компонент активно працює, а інший перебуває в режимі очікування і вступає в роботу при відмові основного.

Дублювання функцій дійсно підвищує стійкість системи до збоїв обладнання чи випадкових чинників, проте у випадку програмного забезпечення воно має свої особливості. Варто зауважити, що програмні помилки зазвичай не є випадковими подіями – вони детерміновані й повторювані. Тому просте паралельне використання двох ідентичних компонентів з одним і тим самим кодом не гарантує надійності. Якщо в коді є дефект, він проявиться в обох копіях одночасно. Як наслідок, банальне дублювання модулів без змін їхньої реалізації не усуває ризик спільного збою. Щоб архітектура з дублюванням функцій справді дала ефект, дублюючі компоненти мають відрізнятися у внутрішній реалізації, тобто використовувати різні способи розв’язання однакової задачі. Такий підхід

відомий як N-версійне програмування: розробляються декілька версій одного модуля різними командами або з використанням різних алгоритмів, після чого їх результати перевіряються на збіг. Наприклад, три незалежні програми можуть обчислювати один і той самий параметр; якщо одна з них дасть збій або хибний результат, два інші «перевизначать» його, забезпечуючи правильність роботи системи. Цей метод суттєво підвищує надійність, але значно ускладнює розробку і збільшує її вартість, тому застосовується переважно у відповідальних галузях.

В результаті дослідження наявних рішень щодо проєктування програмного забезпечення, я вирішила зупинитися на монолітній архітектурі. Свій вибір обґрунтовую тим, що через самостійну роботу над проєктом, зникає потреба у надмірній наочності структури програмного продукту. А також не зважаючи на те, що моя система має потенціал для масштабування, вона наразі не є занадто великою та складною. Зважаючи на це, для мене значно легше розробляти програму дотримуючись монолітної архітектури.

3.2 Функціональне моделювання

Під час роботи над дипломним проєктом – вебсайтом для бронювання заходів – функціональне моделювання відіграло надзвичайно важливу роль. На цьому етапі я зосередилася на тому, що саме має робити система, не заглиблюючись у деталі як вона це реалізовуватиме. Такий підхід фактично перетворив список вимог на наочні схеми, і це виявилося дуже корисним. Для мене, як розробниці, створення діаграм стало зручним способом продумати всі нюанси логіки роботи програми та переконатися, що не лишилося нечітких моментів, які згодом могли б спричинити проблеми при розробці.

3.2.1 Діаграма варіантів використання

Один із перших кроків цього моделювання – побудова діаграми варіантів використання (Use Case), що зображена на рисунку 3.1. Я створила цю діаграму у Visual Paradigm Online, щоб чітко побачити всі ролі користувачів і їхні можливі дії на сайті. На ній простими фігурками зображено,

хто саме взаємодіє з системою і які дії для них доступні. Візуалізація цих сценаріїв від першої особи допомогла мені переконатися, що я врахувала всю необхідну функціональність.

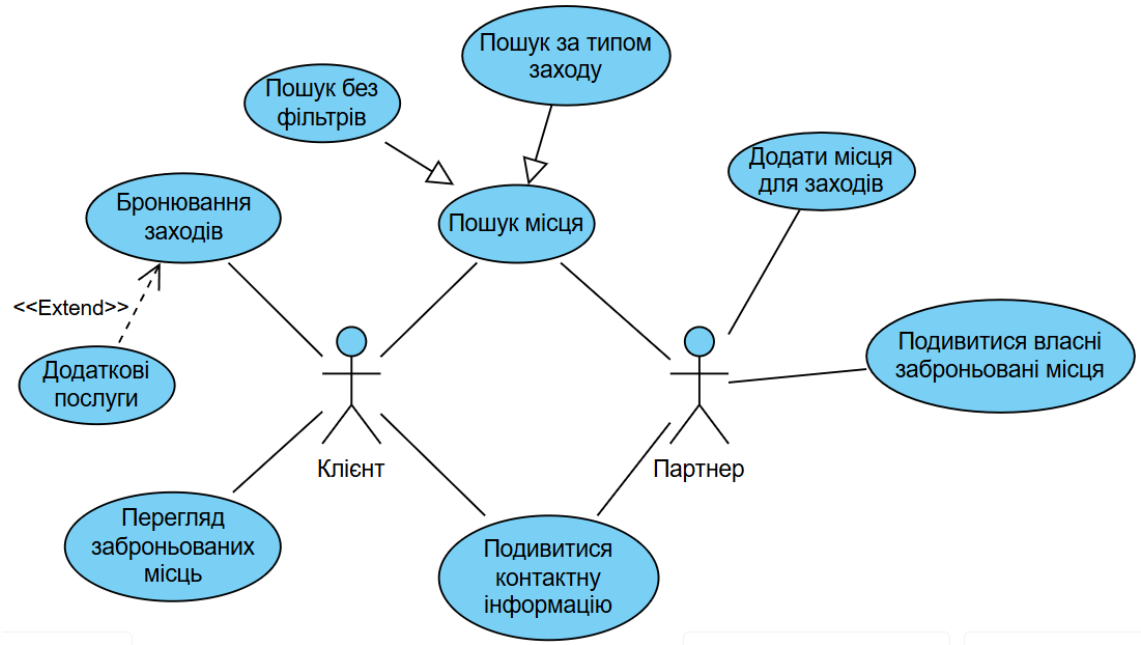

Рисунок 3.1 – Діаграма варіантів використання

Наразі передбачено два актора – клієнт та партнер. Клієнт має доступ до наступних дій: пошук місця, бронювання заходів, обрати додаткові послуги, подивитися контактну інформацію. Щодо можливостей партнера, сюди входять: додавання місця для заходів, доступ до пошуку місць, перегляд власних місць, які забронювали, перегляд контактної інформації.

У діаграмі використовується три види зв’язків, а саме:

1) Асоціація (суцільна лінія);

Використовується між суб’єктом та варіантом використання. Показує які функціональні можливості системи може використовувати актор.

2) Узагальнення (суцільна лінія з трикутним вказівником);

Використовується у ситуаціях коли один елемент базується на іншому елементі моделі.

3) Розширення (пунктирна лінія зі стрілкою, що має назву «extend»);

Використовується коли один варіант використання розширює поведінку іншого. У моєму випадку, через форму бронювання є можливість потрапити

на сторінку додаткових послуг щоб обрати необхідні послуги під час бронювання місця для заходу.

3.2.2 Діаграма потоків даних

Наступним кроком було зобразити потік даних у системі, і тут у пригоді стала DFD (діаграма потоків даних). Контекстна діаграма потоків даних зображена на рисунку 3.2.

Я намалювала декомпозицію DFD у Creately, щоб простежити, як інформація рухається між різними частинами сайту. Завдяки побудові цієї діаграми, я змогла виявити де-які неоднозначності, і одразу продумала як їх розв’язати, доки це ще на папері, а не в коді.

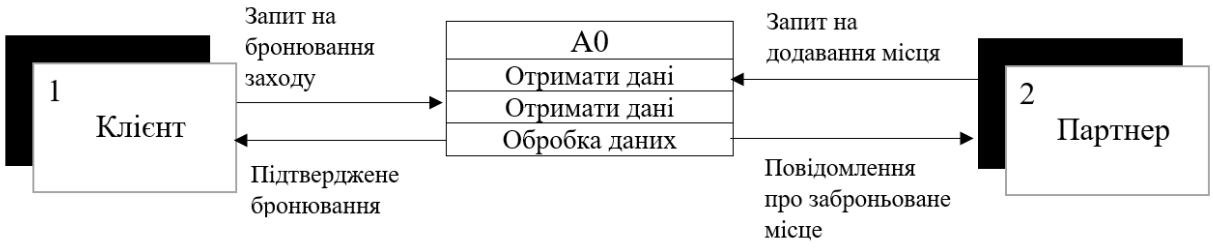

Рисунок 3.2 - Контекстна діаграма DFD

Оскільки моя система створюється для двох різних типів користувачів – клієнтів і партнерів, – дуже важливо продумати детально, як саме виглядатиме робота для кожного з них. Просто описати загальну логіку для всіх недостатньо, адже і клієнти, і партнери мають абсолютно різні. Перший рівень декомпозиції для користувача-клієнта (рисунок 3.3):

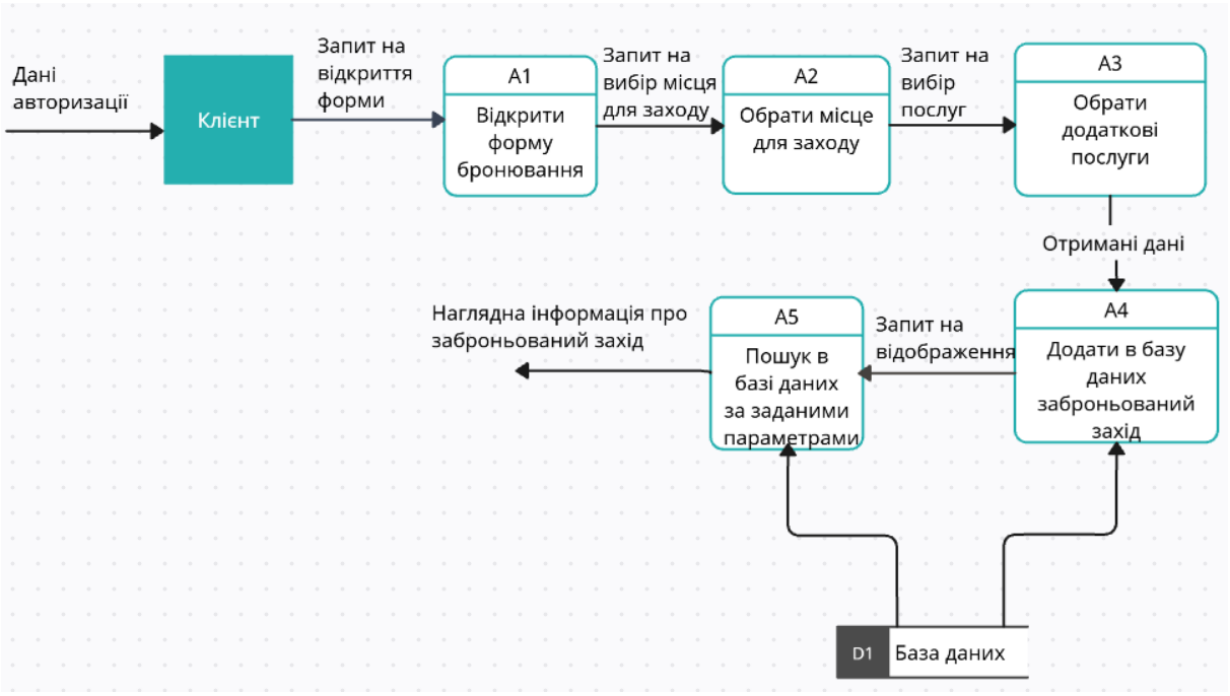

Рисунок 3.3 - Перший рівень декомпозиції для користувача-клієнта

Перший рівень декомпозиції для користувача-партнера (рисунок 3.4):

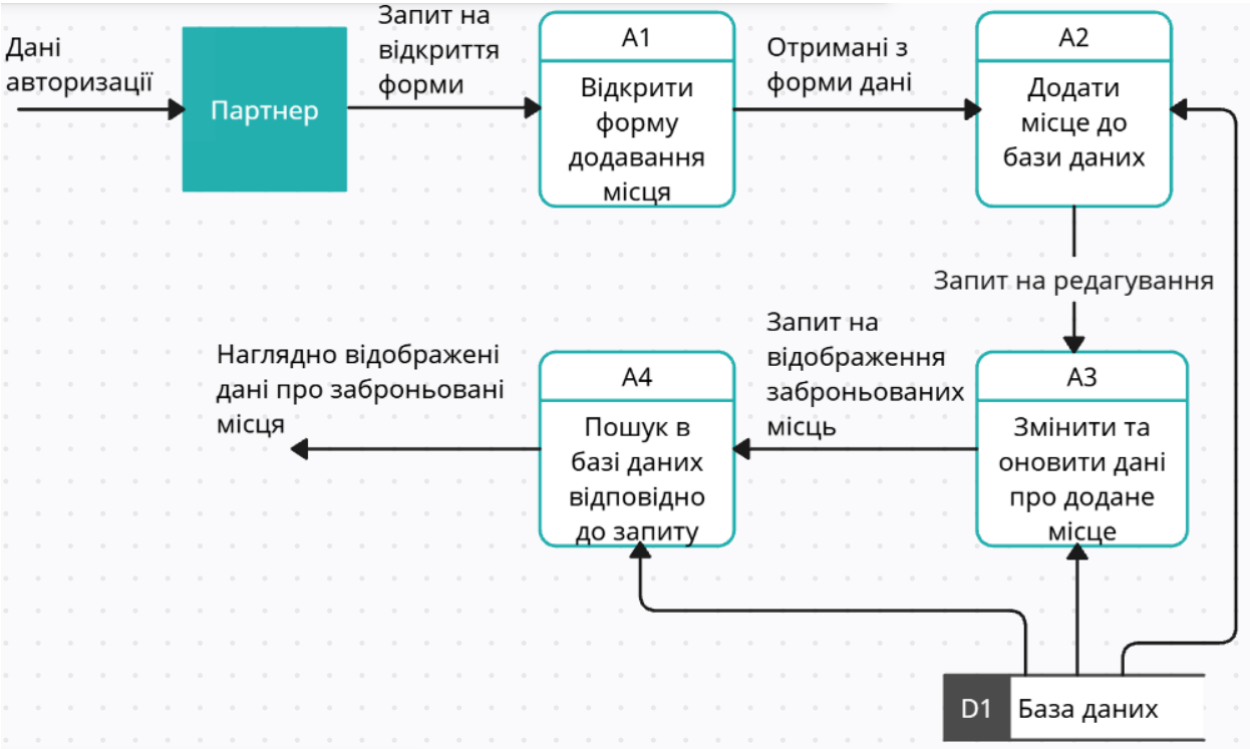

Рисунок 3.4 - Перший рівень декомпозиції для користувача-партнера

Отже, створення функціональних діаграм було не просто формальністю для звіту, а реальною підмогою в розробці. Візуальне моделювання дало мені «карту» системи, з якою набагато легше рухатися від задуму до працюючого сайту. Діаграма варіантів використання показала, хто і що робитиме на моєму сайті, допомогла побачити систему очима різних користувачів. DFD зі свого боку продемонструвала, *як* саме інформація тече всередині системи, і пересвідчилася, що всі процеси узгоджені між собою. Я згадую, як на початку мала певні туманні уявлення про деякі функції, але після моделювання все стало на свої місця – я точно знала, які компоненти потрібні і як вони взаємодіють. Діаграми я створювала за допомогою Visual Paradigm Online та Creately, і ці інструменти суттєво спростили мені життя на етапі проєктування. У результаті цього підходу я відчувала набагато більше впевненості, переходячи до безпосередньої реалізації. Це ніби спочатку прокласти маршрут на мапі: коли чітко бачиш шлях, розробляти сайт для бронювання стає набагато зрозуміліше й простіше.

Так як моя система не має занадто складну та багатошарову структуру, я вважаю доречним зупинитися на першому рівні декомпозиції. На цьому рівні наглядно продемонстровані всі ключові функції системи та потоки даних. Надмірне перевантаження кількістю елементів погіршить сприйняття діаграми.

Під час побудови діаграм потоків даних, я краще зрозуміла принцип функціонування системи та вдосконалила ідеї щодо реалізації веб-сайту. Спланувавши структуру коду краще, я також визначилась з тим, які файли у проєкті мають бути.

3.3 Структура бази даних

База даних – це серце будь-якого сучасного програмного продукту, і в моєму випадку вона відіграє дійсно ключову роль. Саме завдяки грамотно спроєктованому й організованому сховищу вдається не просто зберігати інформацію, а надавати користувачам набагато ширший набір можливостей.

Якщо подумати, практично всі сучасні сервіси, якими ми користуємось щодня, працюють саме через базу даних: зберігають особисті профілі, дані про покупки, історію взаємодій, налаштування й безліч іншого. Без неї веб-сайт перетворився б просто на набір статичних сторінок, де нічого не змінюється, а кожен користувач бачить одне й те саме.

У моєму проєкті база даних теж стоїть в основі всього функціоналу. Вона з'єднує між собою всі компоненти сайту й дозволяє реалізувати інтерактивність, яку сучасний користувач вважає вже стандартом. Тут зберігається не лише інформація для авторизації й аутентифікації – тобто логіни, паролі, електронні адреси, ролі користувачів – а й усі важливі дані, без яких сервіс просто не працював би. Зокрема, йдеться про перелік доступних локацій, які додають партнери, їх детальні описи, фотографії, ціни, кількість доступних місць, а також усі активні та завершені бронювання, які здійснюють клієнти.

Що цікаво, база не лише "запам'ятовує" інформацію, а й допомагає ефективно опрацьовувати запити. Наприклад, коли користувач заходить на сайт, саме база даних підказує, чи є у нього раніше заброньовані локації, які саме місця ще вільні, або які послуги пропонує той чи інший партнер. Усе це робить взаємодію з сайтом справді персоналізованою, бо кожен бачить і використовує тільки ті опції, які стосуються саме його облікового запису. Тобто, клієнт отримує доступ до свого списку бронювань, а партнер – до власних об'єктів і запитів від потенційних орендарів.

Я намагалася побудувати структуру бази так, щоб усі ці дані зберігалися не лише безпечно, а й максимально логічно. З одного боку, це дозволяє зберігати порядок у всій системі, а з іншого – швидко знаходити потрібну інформацію, виводити її на екран, змінювати чи видаляти. Це особливо важливо для сайтів, де дані постійно оновлюються, а кількість користувачів поступово зростає.

Коли я працювала над створенням і налаштуванням бази, було важливо не просто забезпечити її працездатність, а й передбачити можливість

майбутнього розширення функціоналу. Саме тому закладалася гнучка структура, яка дозволяє легко додавати нові сутності, змінювати правила зв’язку між таблицями чи інтегрувати додаткові опції. Всі ці нюанси відображені на схемі, яка наочно демонструє, як саме організовано зберігання інформації у моїй системі (рисунок 3.5). Такий підхід дав змогу не лише зробити сервіс зручним та функціональним уже зараз, а й залишив запас для подальшого розвитку проєкту.

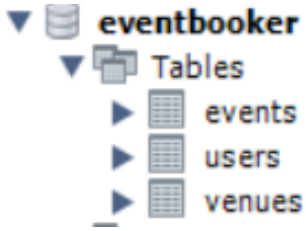

Рисунок 3.5 - Таблиці бази даних eventbooker

Вміст таблиць моєї бази даних та зв’язок між атрибутами таблиць відображено на схемі (рисунок 3.6):

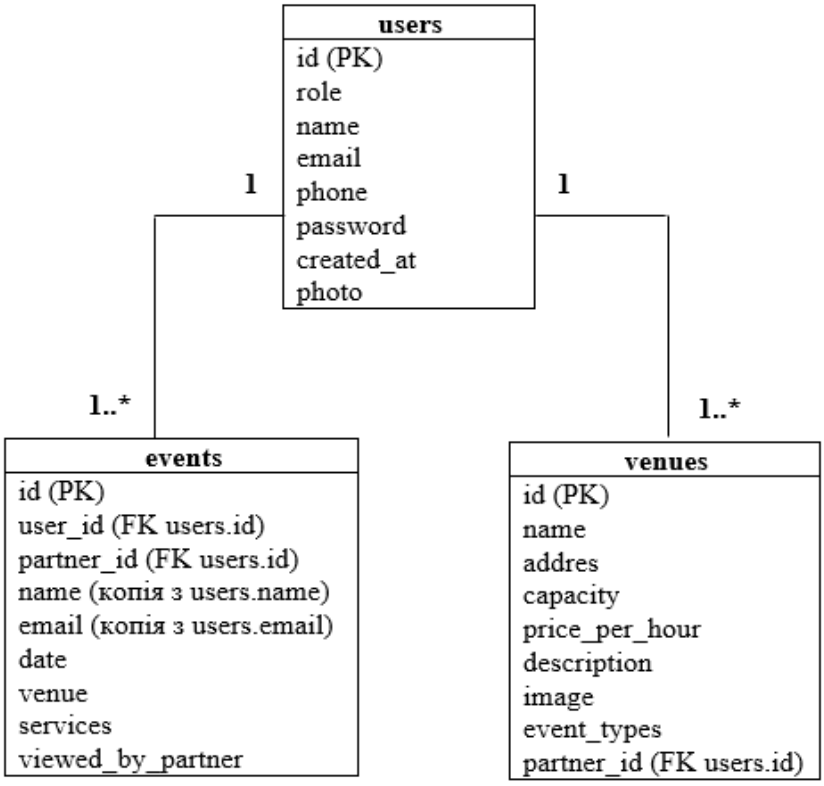

Рисунок 3.6 - Атрибути таблиць бази даних

На діаграмі я позначила первинні ключі (PK) та зовнішні ключі (FK). Первинні ключі – це унікальні ідентифікатори таблиці. Зовнішні ключі призначені для зв’язку таблиць між собою, тобто коли один атрибут посилається на сутність в іншій таблиці.

Таблиця users містить інформацію про зареєстрованих користувачів. Данні цієї таблиці перш за все використовуються при авторизації користувачів, а також їх ідентифікації під час роботи з сайтом для доступу до індивідуальних можливостей кожного типу користувача.

Таблиця events зберігає інформацію про заброньовані заходи користувачами та якому партнеру належить локація, що була заброньована. Атрибут viewed_by_partner допомагає виконувати функцію індикатора, який повідомляє партнеру про нові бронювання. Якщо інформація про нове бронювання була переглянута, індикатор вимикається.

Таблиця venues зберігає дані про місця, які були додані користувачами-партнерами. Атрибут event_types зберігає характеристики, за якими реалізовано фільтрацію локацій.

4 ВИБІР ПРОГРАМНИХ ЗАСОБІВ

У процесі розробки веб-сайту було здійснено підбір програмних засобів, які відповідають завданням проєкту, дозволяють ефективно реалізувати його функціональність та забезпечують стабільну роботу системи в умовах локального тестування. Обрані засоби охоплюють інструменти для створення бази даних, обрані мови програмування та організації серверного середовища.

4.1 База даних

Для зберігання та обробки даних у межах мого проєкту я зупинилася на системі керування базами даних MySQL. Це рішення виявилося найбільш логічним і зручним саме для такої задачі, як створення сучасного веб-сайту для бронювання заходів. З одного боку, MySQL – це вже давно перевірений варіант, який використовують як великі компанії, так і розробники менших сайтів, а з іншого – він чудово підходить для навчальних і особистих проєктів, де потрібна надійність, простота й швидкість.

Вибираючи інструмент для роботи з даними, я звертала увагу не лише на популярність MySQL, а й на те, наскільки він сумісний із PHP. Для мене важливо було, щоб робота з базою не перетворювалася на постійне вирішення дрібних технічних проблем і конфліктів. Саме тому MySQL став ідеальним варіантом: інтеграція бази з логікою сайту дійсно проходила дуже просто і невимушено. Це означає, що всі основні дії – додавання нових локацій, реєстрація користувачів, збереження бронювань – відбуваються «під капотом» без зайвих складнощів. Мені не довелося витрачати час на довге налаштування чи пошук рішень для типових задач, усе працювало інтуїтивно і практично. Це особливо оцінила на етапі тестування – зміни в коді одразу відображалися у базі, і було легко перевіряти роботу різних сценаріїв у реальному часі.

Не менш важливим для мене було питання структури самої бази даних. Я намагалася не просто кудись зберігати інформацію, а зробити так, щоб усі дані були впорядковані й логічно пов’язані між собою. Це, до речі, дуже

допомагає, коли проєкт стає складнішим, з’являється більше користувачів, локацій, послуг чи інших сутностей. Спочатку, ще на етапі проєктування, я продумала, які саме зв’язки мають бути в базі: що, з чим і як взаємодіє. Особливу увагу приділила первинним ключам – це такі унікальні ідентифікатори для кожного запису. Завдяки їм, наприклад, можна не боятися, що однакові назви чи адреси зіб’ють з пантелику систему: кожен запис завжди знає, хто він і де лежить. А от зовнішні ключі допомогли мені встановити чіткі зв’язки між сутностями – наприклад, легко дізнатися, які бронювання належать якому користувачу або яка локація зареєстрована саме цим партнером. Усе це робить структуру гнучкою й надійною: додавати нові таблиці чи поля, або змінювати логіку роботи можна без страху “зламати” вже готову систему.

Коли мова йде про переваги MySQL, не можна оминути зручну індексацію. На практиці це виглядає так: навіть якщо у таблиці вже кілька тисяч записів, пошук потрібної інформації займає долі секунди. Це важливо, бо користувачі завжди очікують швидкої реакції, особливо під час вибору локації чи перегляду історії своїх бронювань. Система підтримує й транзакції – це ще одна корисна функція, що дозволяє бути впевненим у цілісності даних. Наприклад, якщо під час оформлення бронювання стається якийсь збій, жодна «неповна» дія не буде записана.

Ще один плюс – це стабільна робота MySQL на різних операційних системах. Для мене було важливо мати можливість починати з локального сервера на Windows, а потім легко перенести проект на віддалений сервер із Linux чи будь-якою іншою ОС, не переписуючи нічого у частині роботи з базою.

Дуже допомогла і величезна спільнота користувачів MySQL. Якщо раптом виникало питання чи проблема, майже завжди знаходила відповідь або готове рішення в мережі. Це реально економить час і сили, особливо коли стикаєшся з чимось незвичним чи вперше пробуєш якусь нову функцію. Окрім

того, існує багато додаткових інструментів для бекапу, безпеки, оптимізації – усе це можна інтегрувати у систему за потреби.

Робота з інформацією в системі організована максимально прозоро: вибірка, додавання, редагування та видалення записів відбуваються за допомогою SQL-запитів. Всі ці запити інтегруються напряму в PHP-код, що дає змогу робити і прості, і складні вибірки, реалізувати фільтри, сортування, аналітику, не створюючи надмірного хаосу у логіці сайту. Я намагалася будувати запити так, щоб вони працювали швидко й без зайвого навантаження на сервер, адже якщо система стане популярною, це буде особливо важливо.

У підсумку можу сказати, що вибір MySQL для цього проекту виявився дуже вдалим. Поєднання простоти налаштування, високої швидкості виконання запитів і можливості легко змінювати або розширювати структуру під нові задачі – усе це зробило базу не просто місцем зберігання інформації, а справжнім серцем мого веб-сайту. Вона стала невід'ємною частиною всієї системи, яка не лише зберігає дані, а й допомагає зручно й швидко працювати з ними для всіх користувачів – незалежно від їх ролі та обсягу інформації. Таке рішення дозволяє не турбуватися про майбутнє розширення або зміну логіки: за потреби можна легко додати новий функціонал, перенести дані на інший сервер чи реалізувати нові типи взаємодій – і все це без додаткових складнощів для вже існуючих користувачів і без ризику втрати чи пошкодження даних.

4.2 Мови програмування

Для реалізації мого веб-сайту було використано одразу кілька мов програмування, кожна з яких відповідає за свою частину роботи проєкту. Всі ці технології давно стали стандартом у веб-розробці, і саме завдяки їх поєднанню вдалося створити дійсно зручний, функціональний і зрозумілий сайт для різних користувачів. Весь процес написання коду проходив максимально прозоро, а вибір мов і підходів пояснюється не лише популярністю, а й практичною зручністю в рамках мого завдання.

HTML став основою всієї клієнтської частини. Саме за допомогою цієї мови верстається каркас сторінки: задаються всі структури, розділи, заголовки, списки, таблиці, кнопки та інші елементи, які бачить користувач. Я від початку відмовилася від шаблонізаторів або автоматичних генераторів і всі сторінки створювала вручну – це дозволило гнучко керувати кожним блоком і розміщенням інформації, уникнути дублювання або появи зайвого коду, а головне – досягти максимальної чіткості й чистоти структури. У процесі розробки довелося багато разів вносити дрібні зміни – і саме завдяки ручній верстці я могла дуже швидко підлаштовувати сторінки під нові ідеї або підправляти розмітку під потреби різних пристроїв.

Для зовнішнього вигляду та стилізації використовується CSS. Саме ця мова відповідає за всі кольори, відступи, шрифти, вирівнювання блоків і адаптацію під мобільні екрани. Я старалася зробити дизайн сучасним, легким і приємним для сприйняття: щоб нічого не відволікало від основної інформації, але водночас сайт виглядав професійно та охайно. Окремо приділяла увагу зручності читання тексту, розмірам кнопок, контрасту – адже це ті дрібниці, які у підсумку вирішують, чи зручно користуватися сайтом. Додатково були реалізовані прості візуальні ефекти – наприклад, плавна зміна кольору кнопки при наведенні, підсвічування активних елементів або м’які анімації при появі повідомлень про помилку чи успіх. Це додає інтерфейсу “живості” і допомагає користувачу розуміти, що система реагує на його дії.

Невід’ємною частиною фронтенду став JavaScript. За допомогою цієї мови вдалося оживити сторінки й зробити їх більш динамічними, а взаємодію – швидкою і зручною. JavaScript використовується всюди, де потрібно реагувати на дії користувача без перезавантаження сторінки: наприклад, для перевірки правильності введення даних у формах, відправлення асинхронних запитів, відкриття та закриття модальних вікон, зміни контенту на сторінці після натискання кнопок тощо. Основну логіку я винесла в окремі js-файли, які підключаються лише тоді, коли це потрібно – такий підхід дозволив зберегти структуру коду зрозумілою й логічною. Крім того, це суттєво

спростило тестування й налагодження: якщо раптом з’являлася помилка, не доводилося довго шукати причину – усе було розкладено по своїх місцях.

Особливістю мого підходу стало те, що я використовувала лише чистий JavaScript, без підключення сторонніх бібліотек чи фреймворків на кшталт jQuery чи React. Це рішення приймалося свідомо, щоб краще зрозуміти саму суть роботи браузера й навчитися реалізовувати навіть складні речі самостійно, а не покладатися на готові компоненти. Такий досвід дозволяє повністю контролювати поведінку інтерфейсу, не боятися конфліктів чи неочікуваних ефектів від оновлення сторонніх пакетів, а також бути готовою до вирішення нестандартних задач, які точно виникають у процесі розробки.

Ще одна важлива складова – це PHP, мова, що відповідає за серверну частину проєкту. PHP уже багато років вважається класикою для веб-розробки, й на власному досвіді я переконалася, що ця технологія чудово підходить для реалізації будь-яких динамічних сценаріїв. У моєму проєкті PHP забезпечує усе, що пов’язано з обробкою запитів від користувача: тут і робота з базою даних MySQL, і обробка форм реєстрації чи бронювання, і генерація контенту сторінок залежно від ролі користувача, і керування сеансами, та авторизацією. Саме завдяки PHP сторінки сайту “оживають”: змінюють наповнення залежно від того, хто саме зараз авторизований, чи є нові бронювання, які дані було введено у форму. Особливо зручно, що PHP-код вбудовується просто в HTML: це дозволяє комбінувати статичний і динамічний контент, виводити дані з бази без зайвих складних конструкцій і одразу реагувати на зміну інформації.

Практично всі операції із збереження, пошуку, редагування чи видалення інформації в системі відбуваються через SQL-запити, які «зшивають» PHP-код із базою MySQL. Це класична зв’язка для веб-розробки, і в моєму випадку вона спрацювала відмінно: можна вільно організовувати навіть складні вибірки, створювати фільтри й сортування для користувачів, формувати статистику чи аналітичні дані за потреби. Я старалась будувати структуру проекту так, щоб усі ці дії були рознесені по різних файлах за

функціоналом – для підключення до бази, обробки форм, відображення контенту чи стилів. Такий поділ дозволяє зберігати гнучкість, не заплутатися у коді й легко знаходити потрібне місце для внесення змін чи виправлення помилок.

Ще однією рисою моєї роботи було те, що я писала код у звичайному текстовому редакторі, без використання спеціалізованих середовищ розробки або IDE. Кожен файл – окрема «цеглинка» системи, яку я могла допрацьовувати, переміщувати чи видаляти за необхідності. Всі ці файли зберігалися у структурованих папках локального сервера, що дозволяло швидко знаходити потрібні скрипти або стилі й організовувати тестування в зручний для себе спосіб.

Обравши для реалізації HTML, CSS, JavaScript і PHP, я змогла повністю охопити як клієнтську, так і серверну частину проєкту, і дійсно відчути, як ці технології спілкуються між собою. Такий підхід не лише дав глибше розуміння сучасної веб-розробки, а й дозволив набагато швидше реагувати на нові вимоги або виправляти помилки прямо “на льоту”. У підсумку я отримала справжню свободу у створенні інтерфейсу, логіки роботи, керуванні даними – і, головне, досвід, який дозволяє впевнено розвивати свій проект у будь-якому напрямку. Кожен рядок коду у цьому проекті – результат особистого вибору й розуміння, як найкраще зробити сайт простим, швидким і дійсно зручним для користувачів.

4.3 Сервер

Для розгортання й тестування мого веб-сайту я вирішила використовувати добре знайомий багатьом веб-розробникам сервер Apache HTTP Server, версії 2.4. Вибір саме цього рішення був цілком свідомим і практичним: Apache – це не просто перша програма, яку радять у підручниках, а справжній стандарт де-факто у світі хостингу й локальної розробки. Він відзначається стабільною роботою, величезною кількістю корисної документації, підтримкою різних операційних систем і, головне, – простотою

у налаштуванні навіть для новачків. Одразу після встановлення можна майже відразу запускати свій сайт, не витрачаючи години на вивчення складних інструкцій чи пошук відповідних модулів.

Однією з основних причин, чому я обрала саме Apache, стала його чудова сумісність із PHP – основною мовою програмування серверної частини мого проєкту. Фактично, це дозволяє організувати таку зв'язку: користувач заходить на сайт, сервер приймає його HTTP-запит, а потім передає його далі на обробку потрібному PHP-скрипту. Після цього вже сформована відповідь повертається у браузер. Усе це відбувається буквально за секунди, й користувач бачить або потрібну сторінку, або повідомлення про помилку – залежно від ситуації. Така схема дуже наочна і добре підходить для того, щоб відчути, як насправді працюють веб-сервіси під капотом.

Сама конфігурація сервера досить проста, але дає можливість гнучко налаштувати роботу під конкретні потреби. Наприклад, можна задати, які типи файлів буде розуміти сервер – це не тільки HTML і PHP, а ще й CSS, JavaScript, зображення, шрифти й навіть окремі медіа-ресурси. Можна визначити, як саме обробляти помилки, наприклад, зробити гарне повідомлення для користувача у разі, якщо сторінка не знайдена або сталася внутрішня помилка сервера. Окремо налаштовуються дозволи на доступ до окремих папок чи файлів, що особливо зручно, коли є потреба приховати службову інформацію від сторонніх очей.

Для свого проєкту я розгорнула сервер у локальному середовищі – тобто весь сайт працював на моєму комп'ютері, і я мала повний контроль над усіма його складовими. Такий підхід виявився надзвичайно зручним: не потрібно було думати про оренду хостингу, налаштування домену чи роботу з віддаленим доступом. Усі зміни в коді можна було відразу перевіряти – достатньо лише зберегти файл і оновити сторінку у браузері. Це суттєво пришвидшило розробку: якщо раптом виникала помилка, її можна було швидко знайти і виправити, не турбуючись, що хтось побачить незавершену або некоректно працюючу функцію.

Ще однією перевагою локального середовища стала повна автономність. Усі компоненти системи – і база даних, і сервер, і всі скрипти – зберігалися й працювали саме на моєму комп’ютері. Це означає, що не потрібні були жодні додаткові ресурси, платні підписки чи сторонні сервіси. Можна було спокійно експериментувати зі структурою сайту, логікою роботи, додавати чи видаляти файли, не хвилюючись за можливу втрату даних чи несанкціонований доступ.

Що цікаво, такий підхід не лише зручний для навчання чи тестування – у майбутньому він дозволяє дуже легко перенести готовий сайт на реальний хостинг, якщо виникне така потреба. Конфігурації Apache стандартні, а структура проєкту добре структурована, тож перехід із локального на віддалений сервер відбувається практично безболісно. Це дає змогу розвивати проєкт у майбутньому, додавати нові функції чи змінювати вже існуючі – усе без необхідності повністю переробляти налаштування чи витрачати час на повторне тестування в іншому середовищі.

Окремо варто відзначити простоту встановлення та налаштування Apache. Для локальної розробки достатньо буквально кількох кліків: можна скористатися спеціальними збірками (наприклад, XAMPP чи OpenServer), або встановити всі компоненти окремо – у будь-якому разі все працює швидко й без зайвих складнощів. Я обрала шлях із ручним налаштуванням, щоб краще зрозуміти, як взаємодіють між собою різні частини системи, і переконатися, що весь функціонал буде працювати саме так, як потрібно мені.

У підсумку використання Apache HTTP Server для локального розгортання дало мені цілий ряд переваг: швидке тестування змін, повний контроль над усією системою, відсутність витрат на сторонні сервіси, можливість детально налаштувати кожен параметр під свої потреби. Такий підхід виявився ідеальним саме для навчального та експериментального проєкту: я могла зосередитися на логіці та функціоналі, не відволікаючись на адміністративні питання чи проблеми з доступом. Обрана технологія повністю покрила всі поточні завдання й залишила простір для подальшого розвитку й масштабування сайту в майбутньому. .

5 КЛЮЧОВІ ПРОГРАМНІ РІШЕННЯ

5.1 Форма бронювання та її обробка

Одна з ключових функцій сайту – це можливість бронювання локації для проведення заходів. Цей процес я розділила на два етапи: відображення форми бронювання для користувача та окрема обробка введених даних. Для цього використовувалися два різних файли:

1) Відображення форми.

Тут реалізовано інтерфейс, який дозволяє користувачу вказати основну інформацію про себе (ім’я, електронну адресу, дату події) і додатково вибрати необхідні послуги. При відкритті цієї сторінки користувач бачить форму для заповнення. Забронювати можливо лише за умови якщо користувач авторизований, тому визначаємо чи увійшов користувач. Якщо необхідно авторизуватися – то кнопка матиме напис «увійти та забронювати» та перенаправить користувача до форми входу. А для користувача, який увійшов до облікового запису, кнопка матиме назву «забронювати», а поля з ім’ям та електронною адресою – заповняться автоматично. Нижче наведено фрагмент коду з формою:

```
    <section class="booking-form">
                <h1>Бронювання заходу</h1>
                <form onsubmit="submitBookingForm(event)">
                    <input type="hidden" name="redirect"
value="events.php">
                    <input type="hidden" name="booking_name"
value="<?= $isLoggedIn ? htmlspecialchars($userName) : '' ?>">
                    <input type="hidden" name="booking_email"
value="<?= $isLoggedIn ? htmlspecialchars($userEmail) : '' ?>">

                    <label for="name">Ім'я:</label>
                    <input type="text" id="name" name="name"
value="<?= $isLoggedIn ? htmlspecialchars($userName) : '' ?>"
required>

                    <label for="email">Email:</label>
                    <input type="email" id="email" name="email"
value="<?= $isLoggedIn ? htmlspecialchars($userEmail) : '' ?>"
required>

                    <label for="date">Дата заходу:</label>
```

```
                    <input type="date" id="date" name="date"
required>

                    <label for="venue">Обрати місце:</label>
                    <input type="text" id="venue" name="venue"
placeholder="Обрати місце"
onclick="redirectToPage('venues1.php')" required>

                    <label for="services">Додаткові
послуги:</label>
                    <input type="text" id="services"
name="services" placeholder="Обрати послуги"
onclick="redirectToPage('services.php')" readonly>

                    <button type="submit" class="btn">
                        <?= $isLoggedIn ? 'Забронювати' :
'Увійти та забронювати' ?>
                    </button>
                </form>
            </section>
```

2) Обробка даних та перевірка перед записом у БД.

Файл process_booking.php отримує форму після надсилання. Запит надходить через POST, тому спочатку перевіряються всі дані на наявність і правильність. Якщо все гаразд, створюється SQL-запит, щоб додати нове бронювання до бази даних. Якщо вдалось, то відображається повідомлення про успіх, а якщо були помилки, то виникає повідомлення про них. Реалізація в коді така:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $venue_id = $_POST['venue_id'];
    $name = trim($_POST['name']);
    $email = trim($_POST['email']);
    $date = $_POST['date'];
    $services = trim($_POST['services']);

    // Базова валідація: перевірка обов’язкових полів
    if ($name && $email && $date) {
        // Додавання запису до бази
        $sql = "INSERT INTO events (venue, name, email,
date, services) VALUES (?, ?, ?, ?, ?)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("sssss", $venue_id, $name,
$email, $date, $services);
        if ($stmt->execute()) {
            echo \"Бронювання успішно оформлено!\";
        } else {
            echo \"Сталася помилка під час бронювання.
Спробуйте ще раз.\";
```

```
            }
        } else {
            echo \"Будь ласка, заповніть усі обов’язкові
поля.\";
        }
    }
```

5.2 Пошук і фільтрація

Для того щоб користувачі могли знайти найбільш підходящу локацію, на сайті реалізовано два способи пошуку: простий і з використанням фільтрів. Простий пошук дозволяє одразу побачити всі доступні місця, а додатковий фільтр за типом заходу допомагає швидше зорієнтуватися серед багатьох варіантів – наприклад, вибрати тільки ті локації, які підходять для весілля, конференції чи іншої події.

У файлі venues.php є логіка, яка реалізує таку функцію: коли сторінка відкривається без фільтрів, користувач одразу бачить перелік усіх доступних локацій. Якщо ж обрати певний тип заходу у фільтрі, сайт покаже тільки ті місця, які відповідають обраній категорії. Це робиться через перевірку параметрів у запиті (GET), далі формується SQL-запит до бази, і вже за результатом цей список виводиться на сторінці.

```
    // Отримання фільтра з адресного рядка
    $type_filter = isset($_GET['event_type']) ?
$_GET['event_type'] : '';

    // Формування запиту до бази
    if ($type_filter) {
        // Якщо обрано тип події, фільтруємо по ньому
        $sql = "SELECT * FROM venues WHERE FIND_IN_SET(?,
event_types)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("s", $type_filter);
        $stmt->execute();
        $result = $stmt->get_result();
    } else {
        // Якщо фільтр не обрано, показуємо всі локації
        $sql = "SELECT * FROM venues";
        $result = $conn->query($sql);
    }


    // Вивід знайдених локацій
    while ($row = $result->fetch_assoc()) {
```

```
        echo '<div class="venue">';
        echo '<h2>' . htmlspecialchars($row['name']) . '</h2>';
        echo '<p>' . htmlspecialchars($row['address']) .
'</p>';
        echo '<p>Типи подій: ' .
htmlspecialchars($row['event_types']) . '</p>';
        echo '</div>';
    }
```

5.3 Перевірка сесії

В залежності від того, який користувач відвідує сайт (гість, партнер, або клієнт), буде відображено різний контент. Вся логіка базується на сесії та перевірці ролі користувача. При вході в акаунт дані користувача зберігаються у сесії (через масив $_SESSION['user']). З цієї ж сесії отримується тип користувача – клієнт або партнер – а також інші його дані (ім’я, email, фото). У самому коді сторінки перед рендерингом контенту перевіряється, чи є користувач у сесії:

```
<?php
session_start();

// Перевіряємо, чи користувач увійшов
$isLoggedIn = isset($_SESSION['user']);

if ($isLoggedIn) {
    $user = $_SESSION['user'];
    $userName = $user['name'];
    $userEmail = $user['email'];
    $userPhoto = !empty($user['photo']) ? $user['photo'] :
'user.jpg';
}
?>
```

Для неавторизованих користувачів вгорі є кнопка «увійти». Для авторизованих показується їх ім’я, фото профілю, з’являється кнопка «вийти». Також підключено умовне відображення різних можливостей в залежності від ролі користувача. Наприклад, для партнера додано пункт меню «Додати локацію», а для клієнта – «Забронювати захід». Виглядає це приблизно так:

```
<?php if ($isLoggedIn): ?>
    <div class="profile">
        <img src="uploads/<?= htmlspecialchars($userPhoto)
?>" alt="Профіль">
        <span><?= htmlspecialchars($userName) ?></span>
```

```
            <div id="profile-menu" class="profile-menu">
                <p><?= htmlspecialchars($userEmail) ?></p>
                <?php if ($userRole == 'partner'): ?>
                    <a href="add_venue.php">Додати локацію</a>
                    <a href="partner_bookings.php">Мої
бронювання</a>
                <?php elseif ($userRole == 'client'): ?>
                    <a href="booking.php">Забронювати захід</a>
                    <a href="events.php">Мої заходи</a>
                <?php endif; ?>
                <a href="logout.php" class="logout-
btn">Вийти</a>
            </div>
        </div>
    <?php else: ?>
        <a href="login.php" class="login-btn">Увійти</a>
    <?php endif; ?>
```

Такий підхід не лише спрощує використання сайту для кожного, а й робить інтерфейс більш чистим, без зайвих кнопок і непотрібних функцій. До того ж, це підвищує безпеку – випадковий користувач не зможе потрапити в розділи, які йому не належать, і жоден клієнт не отримає доступу до управління локаціями партнерів. Коли весь контент і функціонал динамічно підлаштовуються під конкретну роль, сайт починає виглядати сучасніше, а сама структура легко розширюється під нові потреби, якщо з’являться додаткові ролі чи функції.

5.4 Календар бронювань

В особистому кабінеті партнера реалізовано інтерактивний календар, що допомагає побачити всі дати, на які вже заброньовано певні локації. Є можливість вибрати конкретне місце зі списку – і календар одразу оновиться, показавши тільки ті дати, які зайняті саме для цієї локації. Якщо обрати пункт «усі місця», підсвічуються всі дати, коли хоча б щось із доступних приміщень вже заброньовано. При завантаженні сторінки через AJAX-запит отримуються всі дати з таблиці events, які належать поточному партнеру. Далі ці дати відправляються в календар, і вони підсвічуються як зайняті. Коли партнер змінює вибір у списку місць, сайт знову робить запит до сервера, отримує оновлений список дат для вибраної локації – і календар одразу це відображає.

Реалізація у коді:

```
let bookedDates = [];
const calendar = flatpickr("#calendar", {
    inline: true,
    mode: "multiple",
    disableMobile: true,
    enable: [],
    onReady: highlightBookedDates,
    onMonthChange: highlightBookedDates,
    onYearChange: highlightBookedDates,
});
function highlightBookedDates(selectedDates, dateStr,
instance) {
    setTimeout(() => {
        const dayElements =
instance.calendarContainer.querySelectorAll(".flatpickr-day");
        dayElements.forEach(dayElem => {
            const date = dayElem.dateObj;
            const formatted = instance.formatDate(date, "Y-
m-d");

            if (bookedDates.includes(formatted)) {
                dayElem.classList.add("booked");
            } else {
                dayElem.classList.remove("booked");
            }
        });
    }, 10);
}
function fetchDates(venueId = '') {
    fetch(`?fetch_dates=1&venue_id=${venueId}`)
        .then(res => res.json())
        .then(dates => {
            bookedDates = dates.map(dateStr =>
dateStr.trim());
            highlightBookedDates(null, null, calendar);
        })
        .catch(err => console.error('Помилка завантаження
дат:', err));
}
document.addEventListener("DOMContentLoaded", function () {
    fetchDates(); // показати дати для всіх місць при
старті
document.getElementById("venueSelect").addEventListener("change"
, function () {
        fetchDates(this.value);
    });
});
```

Коли сторінка завантажується чи користувач обирає іншу локацію, JavaScript-скрипт надсилає запит до сервера, отримує перелік зайнятих дат і

одразу ж відображає їх у календарі. Функція highlightBookedDates проходить по кожному дню місяця, перевіряє, чи є ця дата у списку забронюваних, і додає або видаляє клас для візуального виділення. Завдяки цьому партнер у будь-який момент бачить актуальну картину бронювань у зручному та наочному форматі, без потреби оновлювати сторінку вручну.

6 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ В РОБОТІ

Потрапивши на головну сторінку сайту, гість бачить локанічний інтерфейс, де відображаються основні послуги, якими можна скористатися, а саме: забронювати місце для заходу, ознайомитись з додатковими послугами, здати своє приміщення в оренду. У навігаційному меню також є опції подивитись контактну інформацію, локації, які доступні для бронювання, та забронювані клієнтом місця. Скріншот головної сторінки – на рисунку 6.1:

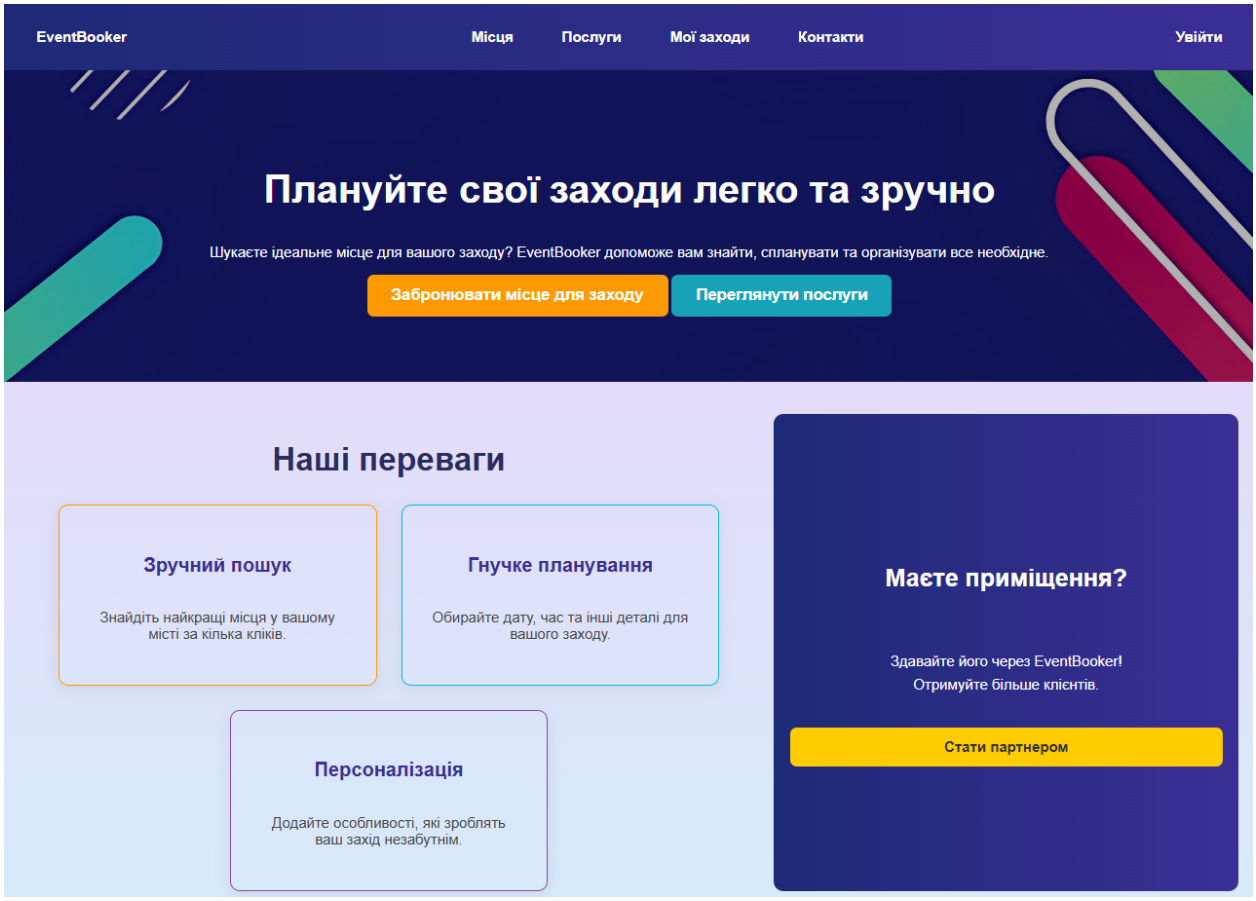

Рисунок 6.1 - Головна сторінка EventBooker

Після переходу на сторінку «місця» можна переглянути наявні для бронювання локації. Є можливість продивитися всі одразу, або відфільтрувати за типом заходу. Форма налаштування пошуку зображена на рисунку 6.2.

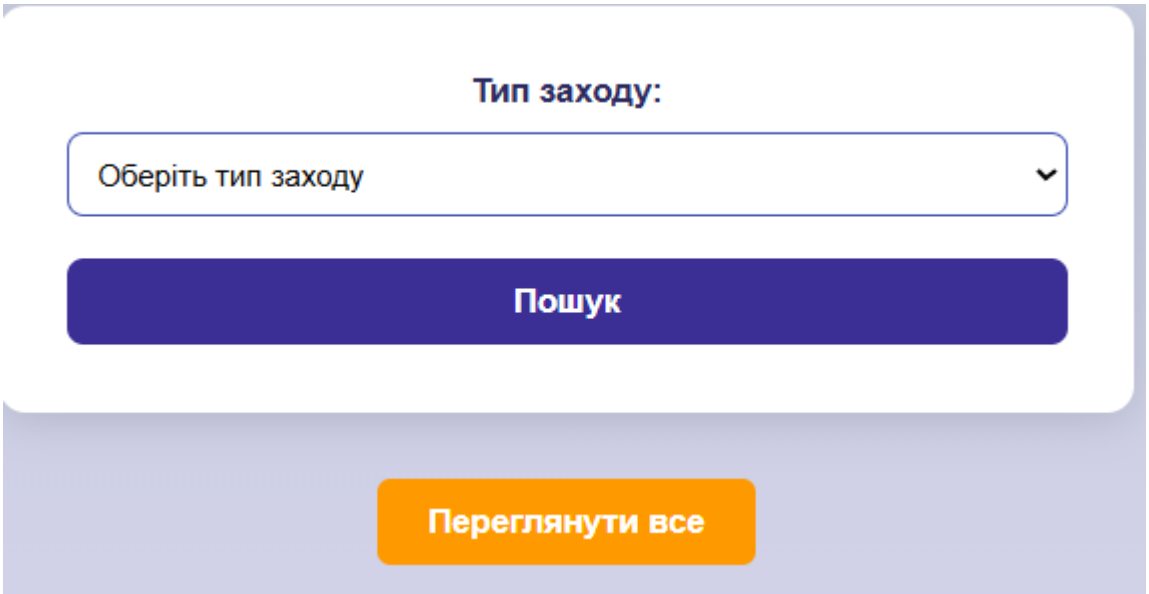

Рисунок 6.2 - Налаштування для пошуку локацій

Форма входу для користувачів передбачає перевірку на дійсність введених даних, а також направляє на сторінку реєстрації (рисунок 6.3).

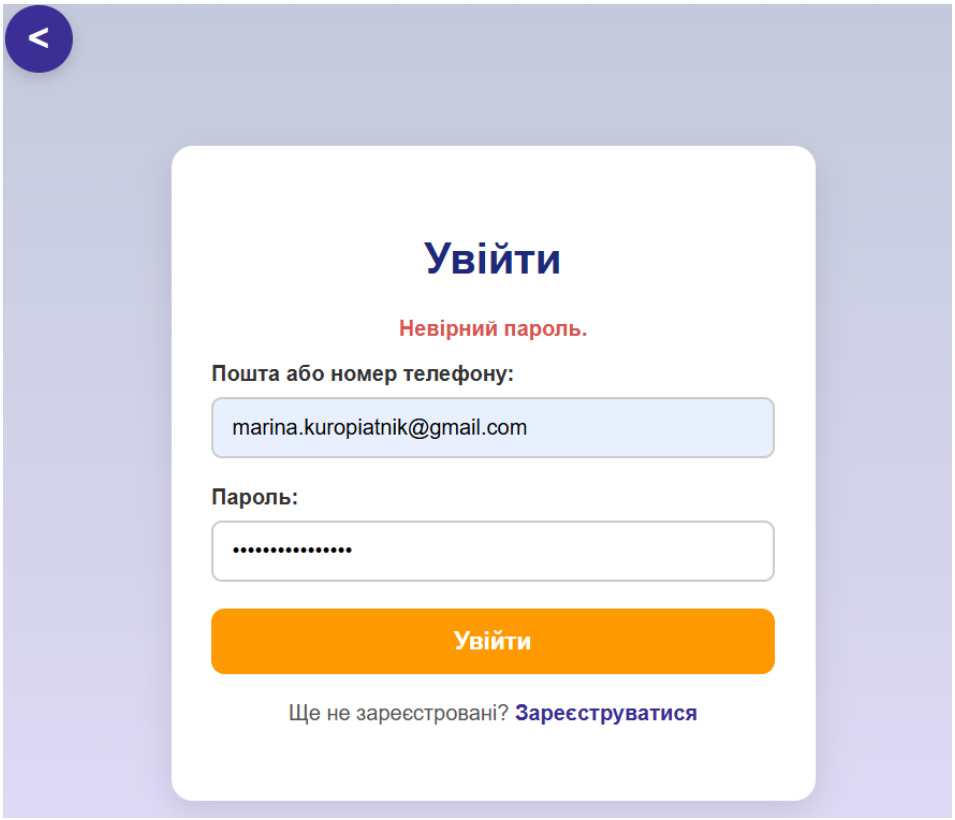

Рисунок 6.3 – Форма входу

Вхід доступний через навігацію сайта, а також через форму бронювання.

Форми реєстрації для клієнтів та партнерів різні. Це зроблено з метою розмежувати користувачів за ролями та надання їм різних повноважень та можливостей (рисунок 6.4).

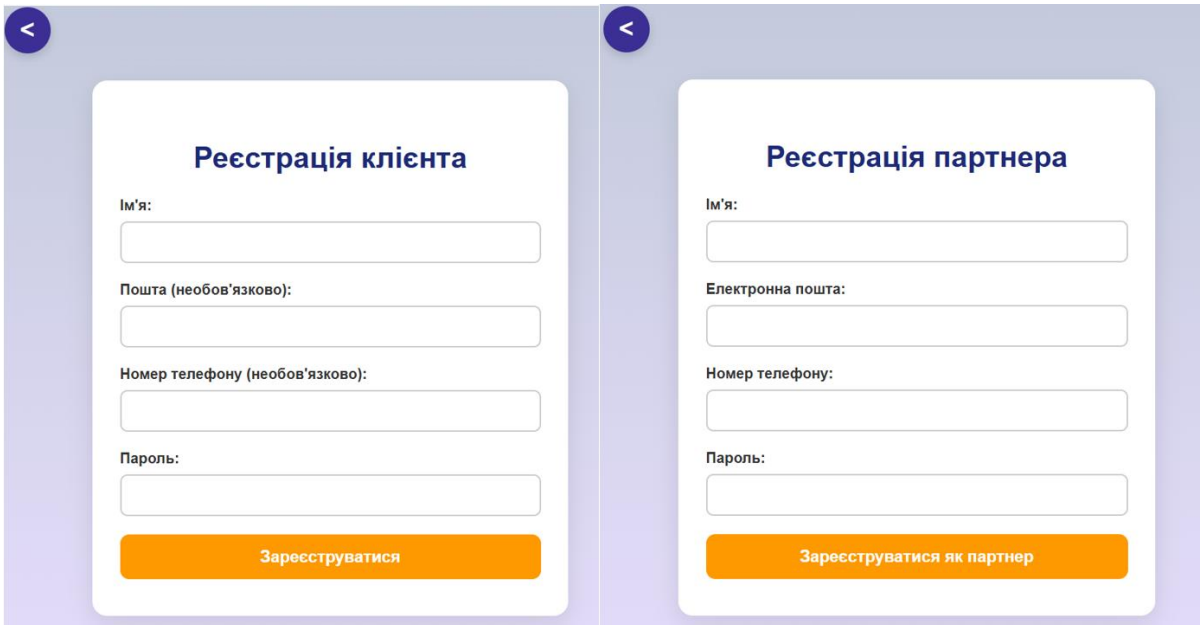

Рисунок 6.4 - Форми реєстрації для користувачів

Для користувача-партнера після входу одразу відкривається доступ до календаря заброньованих локацій, що йому належать та до форми додавання нового місця (рисунок 6.5). Також міняється вміст навігаційного меню. Воно включає сторінки: «місця», «мої місця», «заброньовані місця».

На сторінці «місця» є можливість здійснити пошук та побачити всі оголошення, які були додані користувачами. Сторінка «мої місця» включає тільки ті локації, що були додані користувачем, який наразі авторизований. Сторінка «заброньовані місця» відображає ті місця, які були заброньовані та належать користувачу, під чиїм обліковим записом виконано вхід. А також при наявності нових бронювань та до того, як користувач-партнер їх продивився, біля пункту навігаційного меню «заброньовані місця» блимає червоний індикатор (рисунок 6.6). Після перегляду заброньованих локацій індикатор зникає.

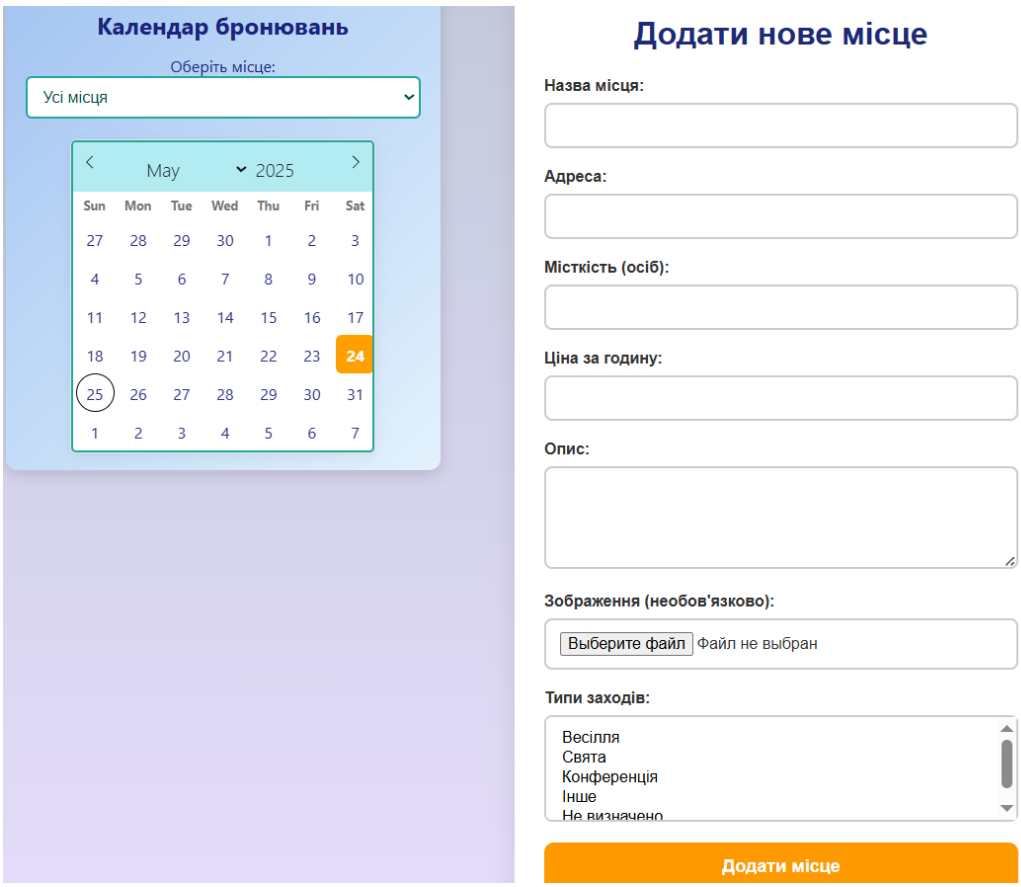

Рисунок 6.5 - Головна сторінка партнера

На календарі бронювань позначається поточна дата, а також дати, на які були заброньовані локації.

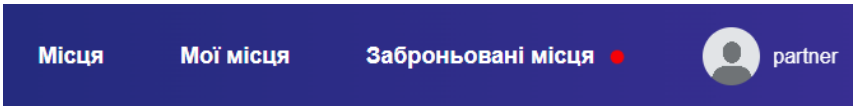

Рисунок 6.6 – Індикатор

Для користувачів доступна опція видалення акаунта. Для цього необхідно натиснути на фото профілю. Відкриється невелике вікно, де вказана електронна пошта та кнопка «видалити акаунт» (рисунок 6.7).

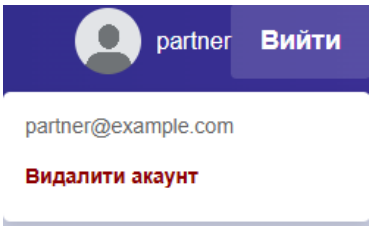

Рисунок 6.7 - Видалити акаунт

Окрім основних функцій, які реалізовані на моєму веб-сайті, присутні інші додаткові опції. Наприклад, є інформаційна сторінка з контактами організації – тут користувач завжди знайде актуальні дані для зв’язку, якщо виникнуть питання або потрібна допомога. Сторінка з переліком послуг дозволяє швидко ознайомитися з усім, що може запропонувати сервіс, а також перейти до детального опису кожної послуги окремо. Це допомагає не загубитися у виборі і дозволяє обрати оптимальний варіант саме під свій захід.

Окремо варто згадати й про можливості, які доступні для зареєстрованих користувачів – партнерів. Для них реалізовано функції редагування та видалення своїх оголошень про доступні для бронювання локації. Це означає, що у будь-який момент власник локації може оновити інформацію про своє приміщення чи видалити його з каталогу, якщо, наприклад, приміщення більше не здається. Аналогічно, користувачі мають змогу видаляти записи про свої бронювання, якщо плани змінилися або захід відмінили. Завдяки таким додатковим функціям робота з сайтом стає не лише зручною, а й по-справжньому гнучкою – кожен може легко керувати своїми оголошеннями та бронюваннями у кілька кліків.

ВИСНОВКИ

Під час роботи над цією дипломною було зроблено справді великий шлях – від ідеї створити зручний онлайн-сервіс для організації подій до повноцінно працюючого веб-сайту, який об’єднує в собі всі ключові можливості для користувачів різних ролей. Основою для розробки стала проста і зрозуміла ідея: полегшити життя тим, хто хоче знайти, забронювати чи запропонувати локацію для будь-якого заходу. Сучасний темп життя диктує свої умови – людям важливо все робити швидко, прозоро і без зайвого стресу. Саме це і лягло в основу концепції проєкту.

Було опрацьовано різні сценарії використання платформи: як виглядатиме процес пошуку місця, що буде потрібно для бронювання, яким чином партнери будуть додавати свої пропозиції. Це дозволило визначити набір функцій, без яких сервіс не зміг би працювати зручно і швидко: гнучкий пошук із фільтрами, оформлення бронювання через зрозумілу форму, можливість керування своїми об’явами для партнерів, а також проста система реєстрації й авторизації.

Розробка самого сайту велася з нуля, без використання складних фреймворків або готових шаблонів. Обрано добре знайомі, універсальні технології – HTML, CSS, JavaScript і PHP. Всі сторінки були зверстані вручну, що дало змогу повністю контролювати як зовнішній вигляд, так і логіку сайту. Завдяки цьому вдалося уникнути зайвої складності й залишити лише найпотрібніше – все, що допомагає користувачу швидко орієнтуватися, робити вибір і оформлювати бронювання. На рівні серверної частини PHP відповідає за взаємодію з базою даних, обробку форм, перевірку авторизації і генерування динамічного контенту під кожного користувача. Такий підхід зробив систему гнучкою для подальших змін і простішою у супроводі.

Окремо варто згадати, що всі основні функції були реалізовані максимально прозоро для користувача. Наприклад, уся логіка бронювання організована так, що дані перевіряються ще до того, як потраплять у базу – це

дає змогу уникати зайвих помилок і гарантувати коректність введеної інформації. Кожен користувач бачить лише ті можливості, які актуальні саме для нього: клієнт може шукати місця і переглядати свої бронювання, партнеру доступні інструменти для додавання, редагування і видалення локацій, а гість взагалі знайомиться з сайтом, не створюючи обліковий запис. До того ж, реалізовано зручний календар для партнерів, де вони можуть бачити всі заброньовані дати своїх локацій, що дуже полегшує планування і контроль над бронюваннями.

Важливо, що розробка проводилась із постійним тестуванням на локальному сервері Apache, а база даних MySQL дозволила легко і швидко зберігати всю необхідну інформацію про локації, бронювання, користувачів і події. Такий підхід дав змогу не лише швидко знайти й виправити недоліки, але й залишити структуру проєкту відкритою для майбутнього розвитку. Наприклад, у перспективі можна додати нові типи користувачів, розширити перелік послуг або реалізувати складніші фільтри пошуку.

У результаті реалізовано сучасний веб-застосунок, який має просту і зрозумілу структуру, мінімалістичний і приємний дизайн, а головне – дозволяє вирішити основне завдання: зібрати в одному місці всі інструменти для організації подій. Користувач може за кілька хвилин знайти підходящу локацію, зв’язатися з власником, оформити бронювання та навіть замовити додаткові послуги для свого заходу. Партнер – власник приміщення – може зручно керувати всіма своїми пропозиціями, бачити стан бронювань, отримувати нові заявки.

Оцінюючи результати цієї роботи, можна сказати, що поставлену мету виконано повністю. Було створено інтуїтивний, легкий у використанні сервіс, який має всі шанси стати надійним помічником для організаторів подій, власників просторів та будь-кого, хто шукає просте і ефективне рішення для планування заходів. Проєкт відкрив широкі можливості для особистого розвитку у сфері веб-програмування, дав досвід повного циклу розробки – від аналізу ідеї до тестування та запуску реального продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек-енд) : навч. посіб. Львів : Вид-во Львівської політехніки, 2018. 248 с.
2. Цеслів О. В. WEB-програмування : навч. посіб. Київ : НТУУ «КПІ», 2011. 298 с.
3. Хайрова Н. Ф., Петрасова С. В. Сучасні технології Web-програмування : навч. посіб. Харків : ФОП Панов А. М., 2020. 112 с.
4. Двірничук К. В., Вацек Д. О. Веб-програмування та веб-дизайн : навч. посіб. Чернівці : Чернів. нац. ун-т ім. Ю. Федьковича, 2022. 472 с.
5. Пасічник О. Г., Пасічник О. В., Стеценко І. В. Основи веб-дизайну : навч. посіб. Київ : Видавнича група BHV, 2009. 336 с.
6. Гаврилов В. П. Інформаційні системи і технології в туризмі : навч. посіб. Харків : ХНЕУ ім. С. Кузнеця, 2016. 168 с.
7. Мальська М. П., Пандяк І. Г. Готельний бізнес: теорія та практика : підручник. 2-ге вид., перероб. і доп. Київ : Центр учбової літератури, 2012. 472 с.
8. Мельниченко С. В. Автоматизовані системи бронювання в туризмі // Культура народів Причорномор'я., 2008. – № 140. – С. 96–100.
9. Балик Н. Р., Мандзюк В. І. MySQL: лабораторний практикум : навч. посіб. Тернопіль : Навчальна книга – Богдан, 2008. 88 с.
10. Зубик Л. В., Карпович І. М., Степанченко О. М. Основи сучасних Web-технологій : навч. посіб. Рівне : НУВГП, 2016. 290 с.
11. Букасов М. М., Галушко Д. О., Катін П. Ю., Хіцко Я. В. Інфраструктура програмного забезпечення Web-застосувань : лаб. практикум: навч. посіб. (електронне видання). Київ : КПІ ім. Ігоря Сікорського, 2023. 53 с.
12. Юскович-Жуковська В. І., Богут О. М. WEB-програмування : підручник. Рівне : Волинські обереги, 2023. 383 с.
13. Мулеса О. Ю. Сучасні інформаційні технології: Web-програмування на боці клієнта. HTML та CSS : навч.-метод. посіб. Ужгород : УжНУ, 2015. 54 с.
14. Жадан Т. А., Антипчук В. І. Характеристика on-line платформ бронювання місць і номерів у закладах розміщення туристичного і готельного бізнесу [Електронний ресурс] // Світові досягнення і сучасні тенденції розвитку туризму та готельно-ресторанного господарства: матеріали II Міжнар. наук.-практ. конф. (Запоріжжя, 10 листопада 2023 р.) / за заг. ред. В. М. Зайцевої. – Запоріжжя : НУ «Запорізька політехніка», 2023. – С. 916–920.
15. Мартін Р. Чистий код: створення, аналіз, рефакторинг / пер. з англ. І. Бондар-Терещенко. Харків : Ранок; Фабула, 2019. 416 с.
16. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Кн. 2: Системи управління базами даних та знань : підручник. 3-тє вид., стер. Львів : Магнолія, 2024. 440 с.
17. Гайдаржі В. І., Ізварін І. В. Бази даних в інформаційних системах : навч. посіб. Київ : Університет «Україна», 2018. 418 с.

18. Бородкіна І. Л., Бородкін Г. О. Інженерія програмного забезпечення : посібник. Київ : НУБіП України, 2018. 248 с.
19. Зеленська Л. М. Івент-менеджмент : навч. посіб. Київ : НАКККіМ, 2018. 148 с.
20. Повалій Т. Л., Світайло Н. Д. Івент-менеджмент : навч. посіб. Суми : СумДУ, 2021. 250 с.
21. Мальська М. П., Худо В. В., Цибух Б. І. Основи туристичного бізнесу : навч. посіб. Київ : Центр учбової літератури, 2012. 348 с.
22. Кучеренко В. Р., Маркітан О. С. Управління проектами : підручник. Харків : ХНЕУ, 2011. 443 с.
23. OLX [Електронний ресурс]. – Режим доступу : https://www.olx.ua.
24. Airbnb (укр. версія) [Електронний ресурс]. – Режим доступу: https://www.airbnb.com.ua.
25. Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org.
26. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol, CA: O'Reilly Media, 2020. 706 p.
27. Cantelon M., Young A., Harter B., Rajlich N. Node.js in Action. 2nd ed. Shelter Island, NY: Manning, 2017. 392 p.
28. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. 2nd ed. Sebastopol, CA: O'Reilly Media, 2019. 343 p.
29. Tatroe K., MacIntyre P. Programming PHP: Creating Dynamic Web Pages. 4th ed. Sebastopol, CA: O'Reilly Media, 2020. 540 p.
30. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed. Boston: Pearson, 2016. 1278 p.
31. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley, CA: New Riders, 2014. 200 p.
32. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2003. 533 p.
33. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 395 p.
34. Duckett J. HTML and CSS: Design and Build Websites. Indianapolis, IN: John Wiley & Sons, 2011. 490 p.
35. Laudon K. C., Traver C. G. E-Commerce 2020–2021: Business, Technology, Society. 16th ed. Harlow: Pearson, 2020. 913 p.
36. Osmani A. Learning JavaScript Design Patterns. 2nd ed. Sebastopol, CA: O'Reilly Media, 2018. 254 p.
37. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. Sebastopol, CA: O'Reilly Media, 2013. 326 p.
38. Gourley D., Totty B. HTTP: The Definitive Guide. Sebastopol, CA: O'Reilly Media, 2002. 656 p.
39. Nixon R. Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. 5th ed. Sebastopol, CA: O'Reilly Media, 2018. 812 p.

40. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken, NJ: John Wiley & Sons, 2012. 296 p.
41. Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education, 2015. 816 p.
42. Crockford D. JavaScript: The Good Parts. Sebastopol, CA: O'Reilly Media, 2008. 176 p.
43. Freeman E., Freeman E., Sierra K., Bates B. Head First Design Patterns. Sebastopol, CA: O'Reilly Media, 2004. 694 p.
44. Shklar L., Rosen R. Web Application Architecture: Principles, Protocols and Practices. 2nd ed. Chichester: John Wiley & Sons, 2009. 408 p.
45. Hunt A., Thomas D. The Pragmatic Programmer. 20th Anniversary ed. Boston: Addison-Wesley, 2019. 352 p.
46. McConnell S. Code Complete. 2nd ed. Redmond, WA: Microsoft Press, 2004. 960 p.
47. Stuttard D., Pinto M. The Web Application Hacker's Handbook. 2nd ed. Indianapolis, IN: Wiley, 2011. 912 p.
48. Booking.com [Електронний ресурс]. – Режим доступу : https://www.booking.com.
49. Eventbrite [Електронний ресурс]. – Режим доступу : https://www.eventbrite.com.
50. Peerspace [Електронний ресурс]. – Режим доступу : https://www.peerspace.com.

ДОДАТОК А

Код файлу add_venue, що відповідає за додавання локацій до БД:

```
<?php
session_start();

// Підключення до бази даних
$conn = new mysqli("localhost", "marina", "24062003",
"eventbooker");
if ($conn->connect_error) {
    die("Помилка з'єднання: " . $conn->connect_error);
}

// Авторизація
if (!isset($_SESSION['user']) || $_SESSION['user']['role']
!== 'partner') {
    header("Location: login.php");
    exit();
}

$partner_id = $_SESSION['user']['id'];
$userName = htmlspecialchars($_SESSION['user']['name']);
$userEmail = htmlspecialchars($_SESSION['user']['email']);
$userPhoto = !empty($_SESSION['user']['photo']) ?
$_SESSION['user']['photo'] : "default.jpg";
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Додати місце</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
<main class="edit-venue-page">
    <nav class="breadcrumb" aria-label="Breadcrumb">
    <a href="my_venues.php">Мої місця</a> &gt; <span>Додати
місце</span>
</nav>


    <h1>Додати місце</h1>
    <form action="add_venue.php" method="post"
enctype="multipart/form-data">
        <label for="name">Назва місця:</label>
        <input type="text" id="name" name="name" required>

        <label for="address">Адреса:</label>
        <input type="text" id="address" name="address"
required>
```

```
            <label for="capacity">Місткість (осіб):</label>
            <input type="number" id="capacity" name="capacity"
required>

            <label for="price_per_hour">Ціна за годину
(грн):</label>
            <input type="number" id="price_per_hour"
name="price_per_hour" required>

            <label for="event_types">Типи заходів:</label>
            <select id="event_types" name="event_types[]"
multiple required>
                <option value="Весілля">Весілля</option>
                <option value="Свята">Свята</option>
                <option
value="Конференція">Конференція</option>
                <option value="Інше">Інше</option>
            </select>

            <label for="image">Зображення:</label>
            <input type="file" id="image" name="image">

            <button type="submit">Додати місце</button>
        </form>
    </main>

    <script>
        function toggleProfileMenu() {
            document.getElementById("profile-
menu").classList.toggle("show");
        }

        document.addEventListener("click", function(event) {
            const profile = document.querySelector(".profile");
            const menu = document.getElementById("profile-
menu");
            if (profile && !profile.contains(event.target)) {
                menu.classList.remove("show");
            }
        });
    </script>
    </body>
    </html>
```

ДОДАТОК Б

Код файлу my_venues.php:

```
<?php
session_start();

$isLoggedIn = isset($_SESSION['user']) &&
!empty($_SESSION['user']['id']);

$servername = "localhost";
$username = "marina";
$password = "24062003";
$database = "eventbooker";

$conn = new mysqli($servername, $username, $password,
$database);
if ($conn->connect_error) {
    die("Помилка з'єднання: " . $conn->connect_error);
}

$userName = $_SESSION['user']['name'] ?? 'Невідомий
користувач';
$userEmail = $_SESSION['user']['email'] ?? '';
$userPhoto = !empty($_SESSION['user']['photo']) ?
$_SESSION['user']['photo'] : 'user.jpg';
$partner_id = $_SESSION['user']['id'] ?? 0;

// Отримання місць
$sql = "SELECT id, name, address, capacity, price_per_hour,
image FROM venues WHERE partner_id = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $partner_id);
$stmt->execute();
$result = $stmt->get_result();
$venues = $result->fetch_all(MYSQLI_ASSOC);
$stmt->close();
$conn->close();
?>

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Мої місця</title>
    <link rel="stylesheet" href="styles.css">
    <script>
        function toggleProfileMenu() {
            document.getElementById("profile-
menu").classList.toggle("show");
```

```
            }

            document.addEventListener("click", function(event)
{
                const profile =
document.querySelector(".profile");
                const menu = document.getElementById("profile-
menu");
                if (!profile.contains(event.target)) {
                    menu.classList.remove("show");
                }
            });
        </script>
    </head>
    <body>
    <header>
        <nav class="navbar">
            <div class="logo">
                <a href="partner_dashboard.php">EventBooker</a>
            </div>
            <ul class="nav-links">
                <li><a href="venues1.php">Місця</a></li>
                <li><a href="my_venues.php">Мої місця</a></li>
                <li>
                    <a href="booked_venues.php" class="booked-
link">
                        Заброньовані місця
                        <?php if ($hasNewBookings): ?>
                            <span class="notification-
dot"></span>
                        <?php endif; ?>
                    </a>
                </li>
            </ul>

            <?php if ($isLoggedIn): ?>
                <div class="profile"
onclick="toggleProfileMenu()">
                    <img src="uploads/<?=
htmlspecialchars($userPhoto) ?>" alt="Профіль">
                    <span><?= htmlspecialchars($userName)
?></span>
                    <div id="profile-menu" class="profile-
menu">
                        <p><?= htmlspecialchars($userEmail)
?></p>
                        <a href="delete_account.php"
class="delete-account" onclick="return confirm('Ви впевнені, що
хочете видалити акаунт?')">Видалити акаунт</a>
                    </div>
                    <a href="logout.php" class="logout-
btn">Вийти</a>
                </div>
```

```
            <?php else: ?>
                <a href="login.php" class="login-
btn">Увійти</a>
            <?php endif; ?>
        </nav>
    </header>

    <main>
        <h1>Мої місця</h1>
        <a href="add_venue.php" class="btn add-venue-
btn">Додати нове місце</a>

        <div class="venues-container">
            <?php if (empty($venues)): ?>
                <p>Ви ще не додали жодного місця.</p>
            <?php else: ?>
                <?php foreach ($venues as $venue): ?>
                    <div class="venue-card">
                        <img src="uploads/<?=
htmlspecialchars($venue['image']) ?>" alt="<?=
htmlspecialchars($venue['name']) ?>">
                        <h3><?=
htmlspecialchars($venue['name']) ?></h3>
                        <p>Адреса: <?=
htmlspecialchars($venue['address']) ?></p>
                        <p>Місткість: <?=
htmlspecialchars($venue['capacity']) ?> осіб</p>
                        <p>Ціна: <?=
htmlspecialchars($venue['price_per_hour']) ?> грн/год</p>
                        <a href="edit_venue.php?id=<?=
$venue['id'] ?>" class="venue-btn">Редагувати</a>
                        <a href="delete_venue.php?id=<?=
$venue['id'] ?>" class="venue-btn venue-btn-delete"
onclick="return confirm('Ви впевнені, що хочете видалити це
місце?');">Видалити</a>
                    </div>
                <?php endforeach; ?>
            <?php endif; ?>
        </div>
    </main>

        <footer>
            <p>&copy; 2025 EventBooker</p>
        </footer>
    </body>
    </html>
```