Міністерство освіти і науки України
Університет митної справи та фінансів
Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: Розробка веб-додатку для бронювання футбольного поля

Виконав: студент групи К23-1м
Спеціальність: 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Сіліверстов Максим Євгенійович
(прізвище та ініціали)
Керівник: к. ф.-м. н., доц. Лебідь О. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент: ДМСУ, Спеціалізоване управління розробки та супроводження програмного забезпечення, Департамент з питань цифрового розвитку, цифрових трансформацій та цифровізації
(місце роботи)
головний державний інспектор відділу розробки програмного забезпечення
(посада)
Бахтін О. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Сіліверстов М.Є. Розробка веб-додатку для бронювання футбольного поля.

Кваліфікаційна робота на здобуття освітнього ступеня магістра за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єктом дослідження є процес автоматизації бронювання футбольних полів.

Предметом дослідження є алгоритми динамічного управління даними бронювання в інтерактивних веб-додатках.

Метою роботи є створення інтерактивного веб-додатку для автоматизації бронювання футбольного поля, який забезпечує зручність для користувачів та адміністраторів.

Робота присвячена розробці веб-додатку для бронювання футбольного поля, який включає динамічний календар, форму для запитів, інтерактивну систему відображення бронювань та інтеграцію з базою даних. Основна увага приділяється розробці функціоналу для автоматизації процесу бронювання, який дозволяє відображати доступні дати, керувати запитами користувачів та інтегрувати систему сповіщень. У роботі також реалізовано функціонал для авторизації та реєстрації користувачів, що гарантує безпеку даних і стабільність роботи додатку.

Результатом роботи є створення сучасного веб-додатку, який забезпечує зручність використання для клієнтів і ефективне управління для адміністраторів. Розроблений додаток має адаптивний дизайн, що дозволяє використовувати його на різних пристроях, і є гнучким до розширення функціоналу в майбутньому.

Ключові слова: ВЕБ-ДОДАТОК, КАЛЕНДАР БРОНЮВАННЯ, АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, ІНТЕРАКТИВНІСТЬ.

ANNOTATION

Siliverstov M.E. Development of a web add-on for booking a football field.

Qualification work for obtaining a master's degree in the specialty 122 Computer Science. – University of Customs and Finance, Dnipro, 2025.

The object of investigation is the process of automating the armoring of football fields.

The subject of research is algorithms for dynamic management of reservation data in interactive web applications.

The method of work is the creation of an interactive web application for automating the reservation of a football field, which will provide reliability for operators and administrators.

This work is dedicated to the development of a web add-on for football field reservations, which includes a dynamic calendar, a request form, an interactive reservation display system and integration with a database. The main focus is on developing functionality to automate the booking process, which allows you to display available dates, process customer requests and integrate the notification system. The robot also has functionality for authorization and registration of accounts, which guarantees data security and stability of the robot.

The result of the work is the creation of a daily web application that will ensure reliable access for clients and effective management for administrators. An additional advantage is the adaptive design, which allows you to use it on different devices, and even expand the functionality in the future.

Keywords: WEB ADD-ON, BOOKING CALENDAR, AUTOMATION, DATABASE, INTERACTIVITY.

ЗМІСТ

ВСТУП

У сучасному світі автоматизація процесів займає ключову роль у різних галузях, включаючи спорт. Зокрема, автоматизація бронювання спортивних ресурсів дозволяє значно підвищити ефективність роботи та покращити взаємодію з користувачами. Розробка веб-додатків для управління бронюванням є важливим напрямом у сфері інформаційних технологій.

Робота присвячена створенню веб-додатку для бронювання футбольного поля, який забезпечить зручність для користувачів та адміністрації. Додаток дозволить автоматизувати процеси реєстрації, авторизації, вибору доступного часу, обробки запитів, а також управління даними про користувачів та бронювання. Основний акцент робиться на розробці зручного та інтерактивного інтерфейсу, а також надійного бекенду для обробки даних.

Веб-додаток використовує сучасні технології веб-розробки, включаючи HTML5, CSS3, JavaScript, а також базу даних SQLite для збереження інформації. Робота забезпечує інтерактивну взаємодію з користувачами завдяки реалізації календаря для вибору доступних днів та часу, системи сповіщень та динамічного відображення статусу бронювання.

Дана кваліфікована робота є чудовою можливістю для застосування знань у галузі веб-розробки, створення баз даних та інтеграції фронтенду з бекендом. Він дозволяє не тільки вдосконалити технічні навички, але й розробити практичний продукт, який буде корисним для кінцевих користувачів.

Метою проєкту є створення інтерактивного веб-додатку, який забезпечить автоматизацію процесу бронювання футбольного поля, зробить його зручним для користувачів та ефективним для адміністрації.

Об’єктом дослідження є процес автоматизації бронювання футбольних полів.

Предметом дослідження є алгоритми динамічного управління даними бронювання в інтерактивних веб-додатках.

Для досягнення поставленої мети в кваліфікаційній роботі ставились та вирішувались наступні завдання дослідження:

– аналіз вимог до серверного додатка для бронювання футбольного поля;

– огляд сучасних технологій розробки серверних додатків та API;

– проєктування архітектури серверного додатка;

– реалізація інтерактивного веб-інтерфейсу з використанням RESTful API.

Методи дослідження: у роботі використовувалися сучасні методи розробки серверних додатків, зокрема мова програмування C++, фреймворк Crow для побудови API, база даних SQLite для зберігання даних, а також мови HTML, CSS і JavaScript для створення клієнтської частини.

Практичне значення отриманих результатів: система автоматизує процес бронювання футбольного поля, покращує взаємодію між адміністрацією поля та клієнтами, забезпечує зручний доступ до бронювань через веб-інтерфейс та підвищує ефективність роботи завдяки інтеграції з базою даних і адаптивному дизайну для різних пристроїв.

У роботі будуть розглянуті аспекти дизайну інтерфейсу, розробки клієнтської та серверної частини додатку, інтеграції бази даних, а також забезпечення надійності та безпеки системи. Дослідження зосереджується на використанні сучасних технологій і підходів у веб-розробці для досягнення найкращого результату.

Проєкт дозволяє поєднати інноваційні рішення та творчий підхід, створюючи якісний продукт, який є важливим кроком для кар’єри у сфері інформаційних технологій.

Новизна даної роботи полягає у розробці унікального алгоритму перевірки конфліктів під час бронювання, який дозволяє враховувати різні аспекти розкладу, включаючи накладення часових проміжків, технічні

перерви між бронюваннями та максимальні обмеження на кількість запитів. Запропонований алгоритм забезпечує гнучкість і масштабованість системи, дозволяючи швидко обробляти запити навіть за умови високого навантаження.

Мій внесок у розробку подібних систем полягає у створенні ефективного механізму перевірки конфліктів, який може бути адаптований для використання в різних середовищах, таких як оренда приміщень, спортивних майданчиків або інших сервісів бронювання. Унікальність алгоритму полягає в його здатності враховувати технічні перерви й забезпечувати швидкий аналіз розкладу з мінімальним навантаженням на сервер і базу даних.

Цей підхід сприяє підвищенню надійності та зручності користування системою, забезпечуючи безперебійну роботу навіть за умов великої кількості одночасних запитів. Реалізований алгоритм є цінним доповненням до сучасних систем бронювання, покращуючи їхню продуктивність і функціональність.

Кваліфікаційна робота складається з 63 сторінки, включає вступ, три розділи, висновки, список використаних джерел, який складається з 30 позицій та додатку. Робота містить 17 рисунків.

РОЗДІЛ 1. ЗАДАЧА ТА АКТУАЛЬНІСТЬ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ КОМУНІКАЦІЇ ТА ЕМОЦІЙНОГО РОЗВИТКУ

1.1 Постановка задачі та основи для розробки

У сучасному світі автоматизація процесів є невід'ємною частиною багатьох сфер діяльності. Особливо це стосується спортивної індустрії, де ефективне управління ресурсами, такими як футбольні поля, є важливим завданням для забезпечення зручності користувачів та зменшення навантаження на адміністрацію. Одним із найпоширеніших проблемних моментів є бронювання футбольних полів, яке часто здійснюється вручну, через телефон або безпосередній візит. Це не лише створює незручності для клієнтів, але й збільшує ризик помилок через людський фактор.

Основна задача даного проєкту – створити інтерактивний веб-додаток для автоматизації процесу бронювання футбольних полів. Додаток повинен дозволяти користувачам швидко та зручно переглядати доступні дати, залишати запити на бронювання, а також отримувати підтвердження чи відмову в реальному часі. Для адміністрації має бути передбачений функціонал управління бронюванням, включаючи можливість блокування певних дат для технічного обслуговування або інших потреб.

Ключові задачі проєкту:

– автоматизація процесу бронювання, яка дозволить уникнути ручного внесення даних;

– створення інтуїтивно зрозумілого інтерфейсу для перегляду доступних слотів і подачі заявок;

– інтеграція бази даних для зберігання інформації про користувачів, бронювання та їх статуси;

– захист даних користувачів та безпечна авторизація/реєстрація;

– закладання основ для майбутнього розширення функціоналу, наприклад, додавання мобільної версії або підтримки кількох мов.

Автоматизація процесів бронювання є важливою через низку причин:

– автоматизовані системи дозволяють зменшити витрати часу як для клієнтів, так і для адміністрації;

– інтерактивний додаток надає користувачам можливість швидко виконувати операції та отримувати зворотний зв’язок;

– електронні системи забезпечують точність даних і мінімізують ризик дублювання чи втрати інформації;

– зручний доступ до спортивних ресурсів сприяє їх популяризації серед населення.

Особливо актуальним є створення системи для бронювання футбольних полів, оскільки футбол є одним із найпопулярніших видів спорту у світі. Багато аматорських і професійних команд, а також звичайні любителі спорту потребують зручного доступу до цих ресурсів. Розробка веб-додатку передбачає використання сучасних технологій та інструментів, що забезпечують ефективність, масштабованість та зручність.

Серверна частина реалізується на C++ з використанням фреймворку Crow для створення RESTful API. Цей вибір обумовлений високою продуктивністю C++, можливістю точного управління ресурсами та легкістю інтеграції з іншими компонентами системи [1].

Сервер забезпечує - обробку запитів від клієнтів (перегляд доступних слотів, реєстрація, авторизація) виконує SQLite для збереження та отримання інформації, також безпеку даних, включаючи шифрування паролів та захист від несанкціонованого доступу. SQLite використовується для зберігання інформації про користувачів, бронювання, статуси слотів та інші дані. Ця база даних обрана через її легкість у налаштуванні, відсутність необхідності в додаткових сервісах і високу продуктивність для невеликих проєктів [2].

Інтерфейс веб-додатку створюється з використанням HTML5, CSS3 і JavaScript. Основний функціонал клієнтської частини включає:

– інтерактивний календар для перегляду доступних дат і часу;

– форми для реєстрації, авторизації та подачі заявок;

– відображення сповіщень і стану бронювань.

Динамічність додатку забезпечується за допомогою AJAX-запитів до сервера. Додаток будується за моделлю клієнт-сервер, де клієнтська частина відповідає за взаємодію з користувачем, а серверна – за обробку даних і логіку. Така архітектура забезпечує модульність і легкість у масштабуванні.

Інструменти розробки які стали основою для розробки:

– CMake – для збирання серверної частини;

– VS Code – як основне середовище розробки;

– Git – для контролю версій і командної роботи.

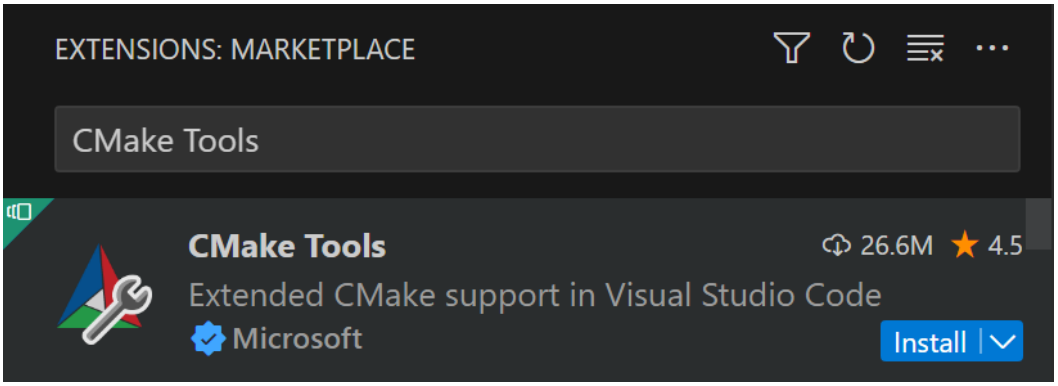

Рисунок 1.1 – Зображення доповнення в CMake в VSCode

CMake – це потужний інструмент для автоматизації складання програмного забезпечення, який забезпечує кросплатформенність та підтримку різних компіляторів і операційних систем. Він дозволяє розробникам автоматизувати процес компіляції, створення виконуваних файлів і бібліотек, що є особливо корисним для великих проєктів із багатьма залежностями. Основна концепція роботи CMake базується на використанні конфігураційних файлів CMakeLists.txt, у яких описується структура проєкту, вихідні файли, залежності та параметри складання [3].

Файл CMakeLists.txt визначає, які файли коду потрібно скомпілювати, які бібліотеки підключити та які параметри компіляції застосувати.

Наприклад, простий файл може включати визначення мінімальної версії CMake, ім’я проєкту та список файлів, що підлягають компіляції. Після створення цього файлу команда cmake використовується для генерації файлів складання, таких як Makefile або проєкти для Visual Studio, залежно від платформи розробки. Процес компіляції запускається окремою командою cmake --build, яка виконує всі необхідні дії для створення виконуваних файлів або бібліотек.

CMake є надзвичайно гнучким інструментом, який дозволяє налаштовувати параметри складання залежно від операційної системи, компілятора або навіть певних конфігурацій проєкту. Він підтримує багатокомпонентні проєкти, інтеграцію зі сторонніми бібліотеками через пакетні менеджери, такі як vcpkg, і спрощує управління залежностями. У вашому проєкті CMake виконує роль основного інструменту для автоматизації складання серверної частини, реалізованої на C++, а також для інтеграції з бібліотеками, такими як Crow та SQLite. Завдяки своїй гнучкості та можливостям CMake спрощує процес розробки та дозволяє зосередитися на написанні коду, залишаючи всі технічні аспекти складання автоматизованими. Хоча CMake є надзвичайно потужним, новачки можуть відчувати складнощі у його освоєнні через специфічний синтаксис і різноманітність можливостей. Однак його переваги, такі як універсальність, можливість працювати з багатьма мовами програмування та інтеграція з сучасними інструментами розробки, роблять його одним із найкращих виборів для побудови сучасних програмних продуктів. У вашому випадку, використання CMake гарантує стабільне складання та сумісність між різними платформами, забезпечуючи простоту інтеграції залежностей і ефективність управління проєктом [4].

Розроблений веб-додаток забезпечить зручний інструмент для автоматизації бронювання футбольного поля, зробивши цей процес швидким, прозорим і доступним для користувачів. Для адміністрації він дозволить ефективніше управляти ресурсами, зменшити навантаження та покращити якість обслуговування клієнтів.

У наступних розділах детально розглядатимуться вибір інструментів, аналіз існуючих рішень та методи реалізації основного функціоналу.

1.2 Інструменти розробки для створення серверу

Розробка серверної частини веб-додатку вимагає використання сучасних інструментів, які забезпечують продуктивність, масштабованість і зручність роботи. Для створення серверу на мові програмування C++ важливо обрати надійні та ефективні засоби, які допоможуть автоматизувати процеси розробки, компіляції та тестування. Одним із ключових елементів є інтегроване середовище розробки (IDE). Visual Studio Code або Visual Studio є популярними варіантами завдяки їхній підтримці багатьох мов програмування, вбудованим інструментам для налагодження та широкій екосистемі розширень.

Інструмент CMake відіграє центральну роль у створенні серверу, автоматизуючи процес компіляції та забезпечуючи кросплатформенну підтримку. Він дозволяє генерувати файли складання, що підходять для різних систем, таких як Makefile для Linux або проєкти Visual Studio для Windows. Для управління бібліотеками й залежностями в проєкті використовується пакетний менеджер vcpkg. Завдяки йому можна легко інтегрувати сторонні фреймворки, наприклад, Crow для створення RESTful API та SQLite для роботи з базою даних.

Crow є сучасним C++-фреймворком, спеціально розробленим для створення високопродуктивних веб-серверів. Він дозволяє просто реалізувати маршрутизацію, обробку HTTP-запитів і роботу з JSON-форматом, що значно спрощує розробку серверної логіки. Для взаємодії з базою даних використовується SQLite, яка забезпечує легкість інтеграції, невисокі системні вимоги та підтримку транзакцій, що є важливим для забезпечення цілісності даних.

Для тестування роботи сервера часто використовуються інструменти, такі як Postman, які дозволяють відправляти HTTP-запити до серверу та перевіряти його відповіді. Вони спрощують процес перевірки правильності реалізації маршрутизації та API. Для навантажувального тестування можна застосовувати спеціальні утиліти, такі як Apache JMeter, що дозволяють оцінити, як сервер обробляє велику кількість одночасних запитів.

Крім того, система контролю версій, така як Git, є важливим інструментом для управління кодом і співпраці в команді. Вона дозволяє зберігати історію змін, відстежувати прогрес у проєкті та працювати над кодом у кількох розробників одночасно. Веб-платформи на зразок GitHub або GitLab спрощують інтеграцію системи контролю версій у робочий процес і забезпечують доступ до інструментів CI/CD (безперервної інтеграції та доставки) [5].

C++ є однією з найбільш популярних мов програмування для створення високопродуктивних систем. У даному проєкті серверна частина веб-додатку реалізована на C++ із використанням бібліотеки Crow для розробки RESTful API.

C++ була розроблена Б’ярном Страуструпом у кінці 1970-х та на початку 1980-х років. Страуструп створив C++ як розширення мови програмування C з метою поєднати можливості низькорівневого програмування C з вищорівневими можливостями об’єктно-орієнтованого програмування.

В 1979 році Страуструп розпочав розробку мови C++ під назвою «C with Classes». Вона поєднувала концепції об’єктно-орієнтованого програмування, такі як класи, спадкування та поліморфізм, з можливостями мови C. Пізніше, в 1983 році, було створено остаточну версію мови, яка отримала назву C++.

Основними метою Страуструпа було створення мови, яка була б ефективною та могла б бути використана для розробки широкого спектру додатків, зокрема ігор. Він прагнув доєднати можливості об’єктно-орієнтованого програмування з потужними можливостями мови C, що дозволяє розробникам писати ефективний та масштабований код.

У подальшому розвитку C++ були внесені різні важливі вдосконалення. У 1998 році було опубліковано стандарт ISO/IEC 14882:1998, відомий як C++98 або C++03, який включав деякі нові можливості, такі як простори імен та шаблони. Пізніше, в 2011 році, був випущений стандарт C++11, що приніс значні зміни та додаткові можливості, такі як автоматичне виведення типів, розширені шаблони, лямбда-вирази та ініціалізація списком.

У 2014 році був опублікований стандарт C++14, який вніс невеликі покращення та розширення. Потім, в 2017 році, був випущений стандарт C++17, який включав більшість функціональних можливостей та розширень, запропонованих у попередніх версіях [6].

Останнім випущеним стандартом є C++20, який був ухвалений у 2020 році. Цей стандарт включає значні поліпшення, такі як модулі, концепти, додаткові функції для обробки рядків та паралельного програмування.

Так як C++ є пріоритетною мовою для студентів у вивченні програмування, ось об’єктивні причини вибору C++ для серверної частини:

– висока продуктивність, де C++ забезпечує максимальну швидкість виконання коду завдяки ефективній компіляції в машинний код. Веб-сервери, написані на C++, здатні обробляти велику кількість запитів за короткий час, що робить мову ідеальною для систем із високими вимогами до продуктивності;

– точний контроль над ресурсами і завдяки ручному управлінню пам’яттю, C++ дозволяє оптимізувати використання ресурсів сервера, що є критично важливим для роботи під великим навантаженням. Це забезпечує стабільність роботи системи навіть за умов обмеженого обчислювального ресурсу;

– гнучкість та масштабованість C++ дозволяє створювати модульні та масштабовані системи, які легко розширювати шляхом додавання нового функціоналу. Це дає змогу інтегрувати додаткові компоненти, наприклад, захист даних, системи сповіщень чи підтримку кількох мов;

– підтримка багатопоточності. C++ має вбудовану підтримку багатопотоковості (multithreading), що дозволяє створювати високопродуктивні сервери для обробки кількох запитів одночасно. У проєкті веб-додатку це використовується для паралельної обробки запитів клієнтів;

– широкий спектр бібліотек та фреймворків доступно для C++, які спрощують розробку серверних додатків. У даному проєкті використовується фреймворк Crow, який забезпечує створення RESTful API та полегшує роботу з HTTP-запитами.

У проєкті важливо забезпечити високу продуктивність серверної частини, надійність обробки запитів та можливість масштабування функціональності. Для досягнення цих цілей було обрано мову програмування C++, яка є одним із найкращих рішень для розробки високоефективних серверів.

Серверна частина реалізована за допомогою фреймворку Crow. Він надає зручні інструменти для створення маршрутизації, обробки HTTP-запитів та взаємодії з базами даних. Сервер на C++ інтегрується з базою даних SQLite для збереження інформації про користувачів, бронювання та їх статуси. Це забезпечує швидкий доступ до даних і мінімізує затримки [7].

Для забезпечення одночасної обробки кількох запитів використовується багатопоточність. Це досягається через багатопоточний запуск серверу. Також C++ дозволяє інтегрувати бібліотеки для шифрування даних, наприклад, OpenSSL, який забезпечує безпечну обробку паролів та захист даних користувачів.

Взагалі, Crow – це легкий та продуктивний фреймворк для створення RESTful API на C++. Основні його переваги це:

– простота використання: зручний синтаксис для створення маршрутів;

– швидкість роботи. Мінімальне споживання ресурсів і висока продуктивність;

– гнучкість: підтримка динамічної генерації відповідей і роботи з JSON-даними.

```
const int port_address = 5500;
    crow::SimpleApp app;
    crow::mustache::set_base(".");

    CROW_ROUTE(app, "/")([]() {
        crow::mustache::context view_context;
        auto home_page = crow::mustache::load("index.html");
        return home_page.render();
    });

    app.port(port_address).multithreaded().run();
```

Рисунок 1.2 – Зображення коду для запуску серверу на Crow, написаний на мові програмування C++

Кросплатформеність є важливою властивістю в сучасному світі програмування, особливо в галузі ігрової розробки. Завдяки росту популярності мобільних пристроїв та різноманітності операційних систем, розробники стикаються з потребою створювати програми, які працюють на різних платформах. Однак, розробка окремих версій для кожної платформи може бути трудомісткою та часозатратною. Тут на сцену виходить мова програмування C++ зі своєю кросплатформеністю.

Кросплатформеність C++ базується на його стандартній бібліотеці і використовується разом з інструментарієм розробки, таким як компілятори та фреймворки, які підтримують кросплатформеність. Завдяки цьому, розробники можуть писати код на C++ один раз і компілювати його для різних платформ [8].

Переваги кросплатформеності C++:

– висока переносимість коду, написаного на C++, може працювати на різних операційних системах, включаючи Windows, macOS, Linux, iOS та Android. Це дозволяє розробникам досягати більшої аудиторії гравців та покривати різноманітні платформи без необхідності переписування коду;

– економія часу та зусиль, завдяки кросплатформеності C++, розробники можуть уникнути необхідності в створенні окремих версій гри для

кожної платформи. Це значно економить час та зусилля, оскільки код можна перевикористовувати на різних платформах з деякими незначними змінами;

– загальна база коду, використання C++ дозволяє створювати загальну базу коду, яка може включати загальні модулі, функції та бібліотеки. Це полегшує управління та підтримку кодової бази, оскільки зміни можна вносити в одному місці та автоматично відображати їх на всіх платформах;

– оновлення та патчі, завдяки кросплатформеності C++, розробники можуть ефективно випускати оновлення та патчі для своїх ігор на різних платформах. Зміни, внесені в код, можуть бути швидко скомпільовані та розповсюджені для всіх платформ, що забезпечує більш швидке та координоване оновлення гри;

– широкий спектр інструментів та підтримка, C++ має широку підтримку серед різних інструментів розробки, компіляторів та фреймворків, які підтримують кросплатформену розробку. Це дозволяє розробникам обирати ті інструменти, які найкраще відповідають їх потребам та отримувати підтримку від широкої спільноти.

Кросплатформеність мови програмування C++ є значним фактором успіху в ігровій розробці. Вона дозволяє розробникам створювати ігри, які працюють на різних платформах, економлячи час, зусилля та ресурси. Завдяки великому вибору інструментів та підтримки, C++ стає потужним інструментом для створення кросплатформених ігор, які можуть досягти широкої аудиторії гравців та забезпечити успішну кар’єру в галузі ігрового розробника [9].

Мова програмування C++ володіє розширеними можливостями, які дозволяють розробникам створювати гнучкі та потужні рішення в ігровій розробці. Серед цих можливостей особливо важливі шаблони, метапрограмування та рефлексія. Вони надають розробникам інструменти для створення складних механік, систем та компонентів, що забезпечують більшу гнучкість та продуктивність в розробці ігор.

Шаблони є одним з ключових елементів мови C++. Вони дозволяють створювати загальні шаблони класів та функцій, які можуть бути параметризовані типами даних. Це дає можливість створювати гнучкі та перевикористовувані компоненти, які можуть працювати з різними типами даних. В ігровій розробці шаблони дозволяють створювати загальні алгоритми, контейнери та системи, що спрощує розробку та підтримку кодової бази.

Метапрограмування в C++ дозволяє розробникам використовувати компіляцію для генерації коду та оптимізації в процесі компіляції. За допомогою метапрограмування можна створювати код на стадії компіляції, що дає більшу ефективність та швидкість виконання. В ігровій розробці метапрограмування може бути використано для генерації оптимізованого коду шляхом використання розріджених структур даних, обчислень на стадії компіляції та інших технік.

Рефлексія в C++ дозволяє отримувати метадані про класи, об’єкти та функції в процесі виконання програми. Це дає можливість динамічно отримувати доступ до даних та виконувати операції над об’єктами без потреби використання заздалегідь відомих типів. В ігровій розробці рефлексія дозволяє створювати гнучкі системи компонентів, зчитувати та змінювати властивості об’єктів на льоту, а також виконувати інші завдання, що спрощують розробку складних систем.

C++ є ідеальним вибором для розробки серверної частини веб-додатку завдяки своїй продуктивності, гнучкості та широкому набору інструментів. Використання мови дозволяє створювати надійні, масштабовані системи з мінімальними затримками та високою продуктивністю. Інтеграція з базою даних SQLite і використання фреймворку Crow спрощує розробку та забезпечує високу ефективність у реалізації поставлених задач.

SQLite – це реляційна база даних, яка працює без окремого серверного середовища, як у випадку з MySQL чи PostgreSQL. Вона представлена у вигляді бібліотеки, яку можна вбудувати безпосередньо в додаток. Для

використання SQLite не потрібно встановлювати чи налаштовувати серверну частину, що значно спрощує роботу з базою даних [10].

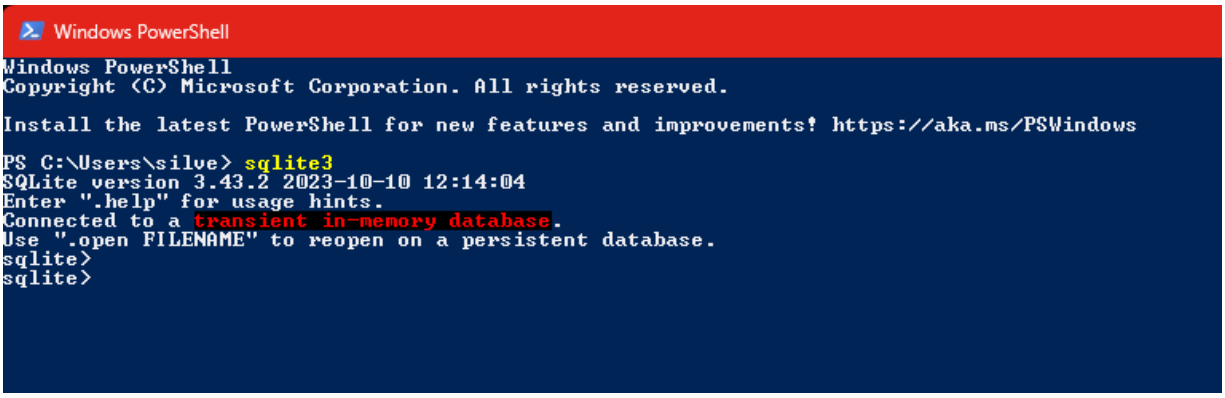

Рисунок 1.3 – Зображення встановленого sqlite в Windows Powershell

Особливості SQLite:

– низькі вимоги до ресурсів. SQLite потребує мінімум системних ресурсів, що робить її ідеальним вибором для мобільних додатків та інших середовищ із обмеженими обчислювальними можливостями;

– підтримка SQL. База даних забезпечує сумісність із більшістю стандартних SQL-запитів, що дозволяє виконувати основні операції з даними;

– простота інтеграції. SQLite працює через API бібліотеки, що доступна для багатьох мов програмування. Це дозволяє легко додавати її у будь-який додаток;

– транзакції. Підтримка транзакцій забезпечує цілісність даних навіть при одночасному виконанні декількох операцій;

– універсальність. SQLite підходить для мобільних і десктопних додатків, браузерів та інших програм, де немає потреби в складних серверних системах.

SQLite має просту структуру, яка включає кілька ключових елементів.

– усі дані бази SQLite зберігаються в одному файлі з розширенням .sqlite або .db. Такий файл можна легко переносити чи копіювати, що спрощує резервне копіювання;

– таблиці використовуються для зберігання даних і визначаються схемою, яка включає назви стовпців, їх типи даних і обмеження;

– таблиці складаються зі стовпців, що зберігають окремі атрибути даних, і записів, які представляють собою рядки з конкретними значеннями;

– для прискорення пошуку SQLite дозволяє створювати індекси, які значно оптимізують доступ до даних у таблицях;

– використовуючи мову SQL, можна виконувати операції додавання, змінення, видалення та вибору даних;

– SQLite підтримує транзакції, що дозволяють виконувати кілька операцій як єдине ціле, забезпечуючи цілісність даних.

Переваги SQLite:

– легка інтеграція. SQLite не потребує серверного середовища, що спрощує її встановлення та використання;

– мінімальні ресурси. База даних працює швидко навіть на пристроях із обмеженими можливостями;

– автономність. Усі дані зберігаються в одному файлі, що полегшує резервне копіювання та перенесення;

– сумісність. SQLite підтримує багато мов програмування (Python, C++, Java тощо).

Недоліки SQLite:

– Обмежена багатокористувацька підтримка. SQLite не підходить для обробки великої кількості одночасних запитів, що може спричинити блокування.

– Масштабованість. SQLite не оптимізована для великих обсягів даних чи високих навантажень.

– Відсутність віддаленого доступу. База даних працює локально і не підтримує мережеве підключення.

– Складні аналітичні запити. SQLite не призначена для великих аналітичних операцій.

SQLite використовується в різних сферах таких як мобільні додатки для зберігання локальних даних у програмах для Android та iOS. Також є десктопні програми які підходять для невеликих застосунків, таких як органайзери чи текстові редактори. Одні з найпоширеніших це вбудовані системи, яка ідеальна для пристроїв з обмеженими ресурсами (телевізори, роутери, GPS-пристрої). Деякі браузери, які використовують бази даних, наприклад, Google Chrome зберігає історію перегляду та закладки за допомогою SQLite. І останнє - прототипи. SQLite часто використовують для тестування і створення демонстраційних проєктів.

SQLite є відмінним вибором для невеликих проєктів, тестових додатків та систем, де важливі легкість використання, автономність і мінімальні ресурси. Однак для великих обсягів даних чи високонавантажених систем рекомендується використовувати MySQL, PostgreSQL або інші більш потужні СУБД.

1.3 Інтерфейс веб-додатку та інтерактивний календар

Веб-інтерфейс є центральною частиною будь-якого веб-додатку, адже саме через нього користувачі взаємодіють із системою. У розробці веб-інтерфейсу використовуються три основні технології: HTML, CSS і JavaScript. HTML забезпечує структуру сторінки, CSS – її візуальне оформлення, а JavaScript додає інтерактивність та динамічну поведінку.

HTML (HyperText Markup Language) – це мова розмітки, яка відповідає за створення основної структури веб-сторінки. У цьому проєкті HTML використовується для організації елементів, таких як календар, форми бронювання, кнопки й інформаційні блоки. Наприклад, для створення інтерактивного календаря використовується контейнер, у який динамічно додаються дні залежно від отриманих даних із сервера. HTML забезпечує

логічну і зрозумілу структуру, що є основою для подальшої стилізації та додавання функціоналу.

CSS (Cascading Style Sheets) надає можливість створити сучасний і привабливий дизайн для веб-сторінки. Стилізація елементів, таких як кнопки, поля форм, чи календар, дозволяє зробити інтерфейс інтуїтивно зрозумілим для користувача. У проєкті застосовуються різні стилі для виділення доступних, заброньованих чи недоступних днів у календарі. Наприклад, доступні дні позначаються зеленим кольором, а заброньовані – червоним. CSS дозволяє не лише додавати кольори, а й забезпечувати плавні переходи між станами, наприклад, при наведенні курсору на елемент.

JavaScript відіграє ключову роль у забезпеченні інтерактивності веб-додатку. У нашому випадку JavaScript використовується для динамічного оновлення календаря, обробки кліків на доступні дати та надсилання запитів на сервер. Наприклад, при завантаженні сторінки JavaScript отримує список доступних і заброньованих днів із сервера у вигляді JSON-даних, після чого генерує елементи календаря на основі цих даних. Також JavaScript дозволяє обробляти дії користувача, такі як вибір дати для бронювання, відправка форми або відображення сповіщень.

Інтерактивний календар є важливим елементом цього веб-додатку, оскільки він дозволяє користувачам легко обирати дати та час для бронювання. Календар генерується динамічно за допомогою JavaScript, а інформація про доступність днів отримується із серверної частини, написаної на C++. Завдяки цьому календар завжди показує актуальні дані про доступні та заброньовані дні. Візуальна диференціація доступних і недоступних днів полегшує навігацію, а можливість вибору дати з кліком робить процес бронювання швидким і зручним.

Зв’язок між фронтендом і сервером здійснюється через RESTful API. Наприклад, при кліку на доступну дату JavaScript надсилає запит на сервер із вибраною датою, а сервер у відповідь підтверджує або відхиляє запит залежно від актуального стану бронювання. Це дозволяє динамічно оновлювати

інформацію на сторінці без її повного перезавантаження, що покращує користувацький досвід.

RESTful API (Representational State Transfer Application Programming Interface) – це архітектурний підхід до створення веб-сервісів, який широко використовується для організації взаємодії між клієнтськими та серверними додатками. Основна ідея REST полягає в тому, що всі дані організовані у вигляді ресурсів, які доступні через унікальні URI (Uniform Resource Identifier). Наприклад, інформація про бронювання може бути доступною за адресою /api/bookings, а деталі конкретного бронювання – за /api/bookings/{id}.

RESTful API використовує стандартні HTTP-методи для роботи з ресурсами. Наприклад, GET дозволяє отримувати дані, POST – створювати нові ресурси, PUT – оновлювати існуючі, а DELETE – видаляти їх. Завдяки цьому API стає інтуїтивно зрозумілим для розробників і легко інтегрується з іншими системами. У відповідь на запити сервер повертає дані у форматі JSON, який є легким для обробки як людиною, так і машиною. JSON використовується для передачі структури даних, таких як списки, об’єкти та вкладені елементи.

RESTful API будується на принципі безстанності, що означає, що сервер не зберігає інформацію про стан клієнта між запитами. Кожен запит до API містить усю необхідну інформацію для його виконання. Такий підхід забезпечує масштабованість і спрощує підтримку системи. Крім того, RESTful API використовує коди стану HTTP, щоб інформувати клієнта про результати виконання запиту. Наприклад, 200 OK означає успішну операцію, 404 Not Found – ресурс не знайдено, а 500 Internal Server Error свідчить про проблему на стороні сервера.

Для реалізації RESTful API у вашому проєкті використовується мова програмування C++ разом із фреймворком Crow. Цей фреймворк спрощує створення маршрутизації для обробки запитів, а також забезпечує роботу з JSON-форматом. Серверна частина отримує запит від клієнта, аналізує його,

взаємодіє з базою даних SQLite для отримання або збереження інформації та повертає результат у вигляді HTTP-відповіді.

Наприклад, запит на створення нового бронювання може виглядати так: клієнт надсилає POST-запит із інформацією про дату, час і користувача, а сервер перевіряє дані, зберігає їх у базі та повертає відповідь із підтвердженням операції. Водночас інтерактивність API тестується за допомогою інструментів, таких як Postman, які дозволяють створювати запити, переглядати відповіді та виявляти помилки.

RESTful API забезпечує простоту інтеграції між клієнтською та серверною частинами вашої системи. Завдяки зрозумілій структурі, стандартизованим методам і гнучкості у використанні цей підхід дозволяє створювати ефективні, масштабовані й надійні веб-додатки, які легко підтримувати та розширювати. HTML створює структуру календаря. Кожен день представлений окремим елементом, який відображається у вигляді інтерактивного блоку. Ці блоки можуть бути згенеровані динамічно на основі даних, отриманих із серверної частини. JavaScript відповідає за обробку цих даних і їх відображення у вигляді інтерактивних елементів. Наприклад, доступні дні відображаються зеленим кольором, тоді як заброньовані – червоним. Такі кольорові індикатори полегшують навігацію для користувачів, дозволяючи їм швидко розпізнавати доступні для бронювання слоти.

Крім статичного відображення дат, інтерактивність календаря забезпечується можливістю користувача взаємодіяти з ним. JavaScript дозволяє реагувати на кліки, відкривати форми для бронювання або показувати інформацію про статус дня. Наприклад, якщо користувач обирає доступну дату, календар може відображати форму для вибору часу та введення контактної інформації. Усе це відбувається без перезавантаження сторінки, завдяки інтеграції JavaScript із сервером через RESTful API.

CSS відіграє важливу роль у створенні візуального вигляду календаря. Календар може бути адаптованим до різних пристроїв завдяки використанню гнучких сіток і адаптивних стилів. Це дозволяє додатку залишатися зручним

як на великих екранах, так і на мобільних пристроях. Окрім цього, CSS забезпечує плавні переходи між станами елементів календаря, наприклад, при наведенні курсору або виборі дати.

Ще однією важливою особливістю інтерактивного календаря є його зв’язок із серверною частиною. Уся інформація про доступність днів динамічно завантажується із сервера. Це означає, що календар завжди показує актуальні дані, навіть якщо зміни були внесені адміністратором у реальному часі. Наприклад, якщо дата була заброньована іншими користувачами, календар автоматично оновлюється, щоб відобразити її як недоступну. Інтерактивний календар поєднує в собі функціональність і зручність використання. Його розробка базується на сучасних веб-технологіях, що дозволяє забезпечити простоту навігації, естетичний вигляд і можливість інтеграції з серверною частиною для динамічного оновлення даних. Це робить календар важливим компонентом системи, який забезпечує якісний користувацький досвід.

1.4 Висновки до першого розділу

У першому розділі було розглянуто основні аспекти, пов’язані з постановкою задачі, обґрунтуванням вибору технологій і підходів для розробки веб-додатку для бронювання футбольного поля. Проведений аналіз підтвердив актуальність даного проєкту, оскільки автоматизація процесу бронювання дозволяє підвищити зручність для користувачів і оптимізувати процес управління ресурсами.

Було визначено, що C++ є оптимальною мовою програмування для реалізації серверної частини завдяки своїй продуктивності, стабільності та широкому спектру інструментів, таких як фреймворк Crow. Crow забезпечує легке створення RESTful API, що є ключовим елементом для інтеграції серверної та клієнтської частин. SQLite було обрано як базу даних через її

легкість, автономність та продуктивність, що робить її ідеальною для додатків із середніми обсягами даних.

Особливу увагу було приділено інтерактивному веб-інтерфейсу, який забезпечує просту та зручну взаємодію користувача із системою. Календар став центральним елементом клієнтської частини, який дозволяє користувачам обирати дати та час для бронювання. Використання HTML, CSS і JavaScript для розробки клієнтської частини забезпечує сучасний вигляд додатку та високий рівень інтерактивності.

Також було проаналізовано роль та можливості технологій фронтенду і бекенду, що стали основою для створення архітектури проєкту. Це дозволило визначити ключові компоненти системи, які гарантують продуктивність, масштабованість і зручність розробки.

Таким чином, перший розділ заклав концептуальну основу для подальшої реалізації проєкту. Визначення мети, обґрунтування вибору технологій та постановка задачі створили чіткий план для розробки, який забезпечить досягнення поставлених цілей та ефективність створюваного веб-додатку.

РОЗДІЛ 2. ПОБУДОВА АРХІТЕКТУРИ, АНАЛІЗ ТА ВИБІР МЕТОДІВ РІШЕННЯ

2.1 Вибір архітектури додатку та технологій для реалізації

Аналіз існуючих веб-застосунків для бронювання дозволяє краще зрозуміти їхню архітектуру, переваги та недоліки, а також визначити найкращі практики, які можуть бути застосовані у проєкті. Більшість сучасних рішень базується на клієнт-серверній архітектурі, де клієнтська частина відповідає за взаємодію з користувачем, а серверна – за обробку запитів, зберігання даних і забезпечення їхньої безпеки.

Клієнтська частина таких додатків створюється за допомогою HTML, CSS і JavaScript. Для покращення динамічності та зручності часто використовуються сучасні фреймворки, такі як React, Angular або Vue.js. Основний акцент у клієнтській частині робиться на простоту використання, швидкість взаємодії та доступність для користувачів із різними технічними знаннями. Ключовими елементами клієнтського інтерфейсу є інтерактивний календар, форми для введення даних і інформаційні панелі, що відображають доступні опції бронювання, історію замовлень або стан поточних запитів.

JavaScript є ключовою технологією для створення інтерактивності та динамічності клієнтської частини веб-додатків. Це мова програмування, яка працює безпосередньо у браузері, дозволяючи змінювати вміст сторінки без її перезавантаження. JavaScript забезпечує гнучкість у розробці клієнтського інтерфейсу, інтеграцію з сервером через API та створення багатофункціональних елементів, таких як інтерактивний календар або форми для введення даних.

JavaScript підтримує модульність і компонентну архітектуру, що дозволяє організовувати код у зрозумілий і зручний спосіб. Бібліотеки для роботи з JSON, такі як fetch або axios, спрощують обмін даними між клієнтом

і сервером, а події, обробники та функції дозволяють створювати інтерактивні елементи інтерфейсу. Наприклад, кліки на дати календаря можуть запускати обробку подій, надсилати запити на сервер та оновлювати стан сторінки без перезавантаження.

React – це бібліотека, розроблена компанією Facebook, яка дозволяє створювати компоненти, що є будівельними блоками інтерфейсу. Вона забезпечує ефективне управління станом за допомогою Virtual DOM, що мінімізує переробку сторінки при зміні даних. React ідеально підходить для великих проєктів, де важлива модульність і повторне використання компонентів.

Angular – це повноцінний фреймворк, розроблений компанією Google, який надає інструменти для створення масштабованих додатків. Він використовує архітектуру MVC (Model-View-Controller) і TypeScript, що додає строгість і надійність у розробку. Angular підходить для складних додатків із багатьма взаємодіями та великою кількістю залежностей.

Vue.js – це легкий фреймворк, який забезпечує баланс між простотою та потужністю. Він використовує реактивність даних і дозволяє швидко створювати інтерактивні елементи. Vue.js підходить для середніх за розміром проєктів, де важливі швидкість розробки та легкість інтеграції.

Фреймворки значно спрощують управління станом і оновлення інтерфейсу. Наприклад, за допомогою Vue.js можна створити інтерактивний календар, який автоматично оновлюється при зміні доступних дат, що надходять із серверу. Angular дозволяє реалізувати складні сценарії, такі як валідація форм або управління автентифікацією користувачів. React, завдяки своїй популярності, має багату екосистему бібліотек і розширень, які забезпечують додаткові можливості для інтерактивності та оптимізації продуктивності.

JavaScript і сучасні фреймворки є основою для створення зручного та функціонального клієнтського інтерфейсу, що дозволяє розробляти адаптивні,

швидкі та інтерактивні додатки, які відповідають сучасним вимогам користувачів.

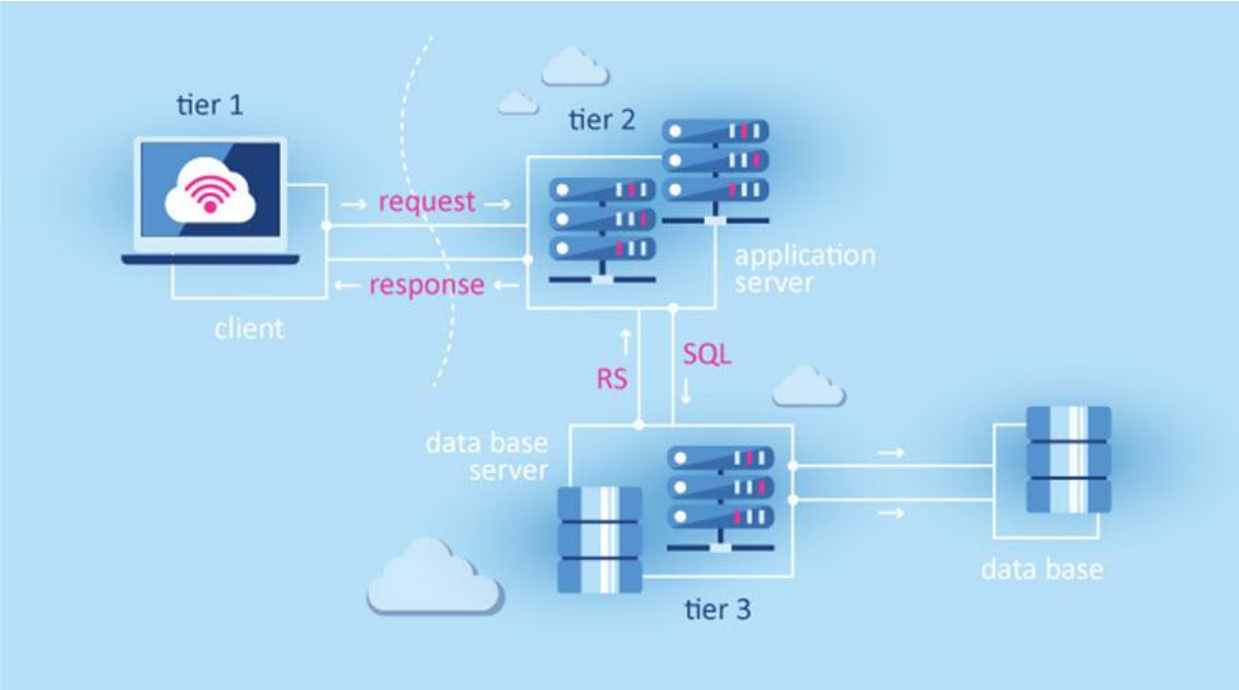

Рисунок 2.1 – Зображення клієнт-серверної архітектури

Серверна частина забезпечує обробку запитів, перевірку даних та взаємодію з базою даних. Популярні платформи, такі як Booking.com, OpenTable або Calendly, використовують RESTful API або GraphQL для комунікації між клієнтом і сервером. Сервер виконує роль посередника, який перевіряє запити, зберігає інформацію в базі даних і повертає відповідь у форматі JSON, що дозволяє легко інтегрувати ці дані в інтерфейс. Для зберігання інформації про користувачів, бронювання та доступні дати зазвичай застосовуються реляційні бази даних, такі як MySQL або PostgreSQL, або нереляційні бази, такі як MongoDB.

У сучасному світі автоматизація процесів бронювання стає важливим інструментом для підвищення зручності та ефективності у сфері послуг. Веб-додатки для бронювання дозволяють користувачам легко та швидко знаходити доступні ресурси, планувати події та керувати своїми запитами. Одним із найяскравіших прикладів такого рішення є платформа Booking.com, яка

пропонує широкий спектр функціональності для пошуку та бронювання готелів, квартир та інших місць розміщення. Аналіз структури та функціональності цього додатку дозволяє глибше зрозуміти принципи роботи успішних веб-застосунків, виділити найкращі практики та адаптувати їх до розробки власних рішень. Booking.com є одним із найвідоміших веб-застосунків для бронювання житла та послуг, і його структура може слугувати зразком для розробки схожих систем. Додаток побудований на клієнт-серверній архітектурі, що дозволяє чітко розділити обов'язки між клієнтською частиною, яка відповідає за взаємодію з користувачем, і серверною, яка обробляє дані та запити. Клієнтська частина Booking.com реалізована з використанням сучасних технологій, таких як React, що дозволяє створювати динамічний і швидкий інтерфейс. Основні елементи інтерфейсу включають інтерактивний календар для вибору дат, форми для введення даних, фільтри та списки доступних пропозицій. Серверна частина працює на потужних інструментах, таких як Node.js і Python, і використовує MySQL для зберігання структурованих даних. Вибір MySQL пояснюється її здатністю ефективно обробляти складні запити й забезпечувати цілісність даних, що є критичним для систем, які працюють із транзакціями та великими обсягами даних.

Одним із важливих аспектів роботи веб-додатків є вибір бази даних. Реляційні бази даних, такі як MySQL, відрізняються строгістю у визначенні схем і забезпечують надійність роботи зі складними запитами. Це робить їх придатними для таких проєктів, як Booking.com, де дані мають чітку структуру. Водночас нереляційні бази даних, наприклад MongoDB, пропонують більшу гнучкість завдяки відсутності строгих схем, що дозволяє зберігати слабкоструктуровані або динамічні дані у форматі JSON. MongoDB добре підходить для систем, які вимагають швидкого масштабування та роботи з великими масивами слабкоструктурованих даних, таких як історія переглядів або персоналізовані рекомендації.

Порівнюючи MySQL і MongoDB, можна сказати, що перша є ідеальним вибором для додатків, які потребують високої точності, підтримки транзакцій

і складних запитів. Наприклад, у Booking.com MySQL дозволяє працювати з датами бронювання, цінами та іншими чітко структурованими даними. MongoDB, у свою чергу, краще підходить для застосунків, які працюють з динамічними записами, такими як додатки соціальних мереж або системи моніторингу.

Для проєкту з бронювання футбольного поля важливо врахувати специфіку даних і обрати базу даних, яка найкраще відповідає вимогам системи. SQLite або MySQL є оптимальними варіантами завдяки їх простоті у використанні, підтримці транзакцій і надійності для зберігання структурованих даних. У той же час, MongoDB може бути цікавою альтернативою, якщо система розшириться і виникне потреба у гнучкому зберіганні даних. Аналіз структури Booking.com і порівняння реляційних та нереляційних баз даних дозволяє обрати оптимальні рішення для створення ефективної та продуктивної системи. Такий підхід допоможе реалізувати зручний і надійний додаток, який буде відповідати потребам користувачів.

Одним із важливих аспектів успішних веб-додатків є інтерактивність. Наприклад, інтерактивний календар дозволяє користувачам швидко обирати доступні дати та час. Візуальні підказки, такі як кольорове виділення забронbованих і доступних днів, спрощують взаємодію з системою. Інтеграція автоматичних сповіщень про статус бронювання чи наближення часу події додає зручності й покращує користувацький досвід.

Попри численні переваги, існуючі рішення мають і свої недоліки. Деякі системи перевантажені функціями, що може ускладнити взаємодію для звичайного користувача. Обмежена локалізація, відсутність підтримки різних мов або валют, а також висока ресурсомісткість можуть стати значними перепонами для широкого використання таких додатків. Крім того, складні системи часто мають високі вимоги до швидкості інтернет-з’єднання, що робить їх менш доступними для користувачів у регіонах із поганим покриттям.

Зважаючи на це, структура додатку для бронювання футбольного поля буде розроблена з урахуванням найкращих практик: зручний і простий

інтерфейс, інтерактивний календар, інтеграція зі сповіщеннями та оптимізація для швидкої роботи навіть на пристроях із невеликими ресурсами. Уникнення перевантаження функціями дозволить створити систему, яка буде ефективною, зручною та доступною для широкої аудиторії. Це забезпечить конкурентоспроможність і дозволить задовольнити потреби користувачів у сучасному автоматизованому процесі бронювання.

Вибір архітектури додатку та відповідних технологій є одним із ключових етапів розробки програмного забезпечення. У даному проєкті, присвяченому створенню веб-додатку для автоматизації бронювання футбольного поля, було прийнято рішення використовувати клієнт-серверну архітектуру. Такий підхід дозволяє забезпечити чітке розділення між клієнтською частиною, яка відповідає за інтерфейс і взаємодію з користувачем, та серверною частиною, яка обробляє запити, зберігає дані та забезпечує їх безпеку.

Клієнт-серверна архітектура забезпечує модульність, масштабованість і легкість у підтримці системи. Клієнтська частина реалізована з використанням HTML, CSS і JavaScript, що дозволяє створити інтуїтивно зрозумілий і привабливий інтерфейс. Серверна частина розроблена на C++ із використанням фреймворку Crow, що забезпечує продуктивну роботу з HTTP-запитами та RESTful API. Для зберігання даних обрана база даних SQLite, яка ідеально підходить для невеликих додатків, завдяки своїй простоті, автономності та продуктивності.

Клієнтська частина виконує роль інтерфейсу для користувача. HTML забезпечує структурну основу сторінки, CSS – її візуальне оформлення, а JavaScript додає інтерактивність та дозволяє динамічно оновлювати контент без необхідності перезавантаження сторінки. Календар і форми для бронювання інтегровані з сервером за допомогою JavaScript, використовуючи функції для надсилання запитів через API. Такий підхід забезпечує високу зручність і швидкість роботи для кінцевого користувача.

Серверна частина є ядром додатку, яке відповідає за обробку запитів і взаємодію з базою даних. Використання C++ у серверній частині гарантує високу продуктивність і стабільність. Фреймворк Crow дозволяє швидко налаштувати маршрути для обробки HTTP-запитів, включаючи отримання доступних дат для бронювання, створення нових бронювань, скасування існуючих запитів та керування користувачами. Інтеграція з SQLite забезпечує надійне зберігання даних із підтримкою транзакцій, що гарантує їхню цілісність.

Для передачі даних між клієнтом і сервером використовується формат JSON, який є стандартом для веб-додатків. JSON забезпечує легкість у передачі структурованої інформації, такої як списки доступних дат чи підтвердження бронювання. RESTful API, побудоване на базі HTTP, дозволяє реалізувати чітку і просту структуру запитів, що полегшує інтеграцію клієнтської і серверної частин.

JSON (JavaScript Object Notation) – це легкий текстовий формат обміну даними, який став стандартом у сучасній веб-розробці. Він базується на простій структурі, яка складається з пар "ключ-значення", що робить його зручним для читання та обробки як людиною, так і машиною. Формат JSON використовується для передачі структурованих даних між клієнтом і сервером у додатках. Наприклад, дані про бронювання можуть бути представлені у вигляді JSON-об’єкта, який включає такі ключі, як "user_id", "date" та "time_slot". Цей формат дозволяє зручно передавати як прості дані (рядки, числа, логічні значення), так і складні структури, наприклад, вкладені об’єкти або масиви.

Однією з головних переваг JSON є його універсальність. Його підтримують майже всі сучасні мови програмування, включаючи JavaScript, C++, Python і Java. У JavaScript JSON є невід’ємною частиною екосистеми, і розробники можуть легко перетворювати об’єкти в JSON за допомогою функції JSON.stringify() і зворотно за допомогою JSON.parse(). У C++ для роботи з JSON використовуються спеціальні бібліотеки, такі як JSON for

Modern C++, які дозволяють створювати, аналізувати та обробляти JSON-дані в кілька рядків коду.

JSON став популярним завдяки своїй простоті та компактності. Він займає менше місця, ніж формати на кшталт XML, завдяки відсутності зайвих тегів. Це особливо важливо в сучасних веб-додатках, де швидкість передачі даних є критичною. JSON використовується у поєднанні з RESTful API для організації ефективної взаємодії між клієнтською та серверною частинами. Наприклад, клієнт може надіслати POST-запит із JSON-даними про бронювання, а сервер повертає відповідь у тому ж форматі, яка підтверджує успішне створення бронювання або інформує про помилку.

Попри численні переваги, JSON має деякі обмеження. Наприклад, він не підтримує коментарі, що може ускладнити документування великих файлів. Крім того, робота з великими обсягами бінарних даних може бути менш ефективною порівняно з іншими форматами, такими як Protobuf. Проте для більшості сучасних веб-додатків JSON залишається найзручнішим і найуніверсальнішим форматом обміну даними. Його використання дозволяє будувати масштабовані та ефективні системи, які легко інтегруються між собою.

Також у процесі розробки було враховано питання безпеки. Серверна частина включає механізми перевірки вхідних даних для запобігання SQL-ін’єкціям та XSS-атакам. Паролі користувачів шифруються перед збереженням у базу даних, а комунікація між клієнтом і сервером захищена за допомогою HTTPS.

HTTPS (HyperText Transfer Protocol Secure) – це захищений протокол передачі даних, який використовується для безпечного обміну інформацією між клієнтом і сервером у мережі Інтернет. На відміну від HTTP, який передає дані у відкритому вигляді, HTTPS шифрує усі дані за допомогою протоколу SSL/TLS, що забезпечує конфіденційність і захист від несанкціонованого доступу. Це означає, що навіть якщо хтось перехопить передані дані, він не зможе їх розшифрувати без відповідного ключа.

Ключовою особливістю HTTPS є використання цифрових сертифікатів, які гарантують автентичність серверу. Коли клієнт підключається до серверу через HTTPS, спочатку відбувається процес встановлення захищеного з’єднання. Сервер надсилає клієнту свій цифровий сертифікат, який перевіряється на автентичність. Якщо сертифікат є дійсним, між клієнтом і сервером встановлюється зашифрований канал для передачі даних. Цей процес називається SSL/TLS рукостисканням і виконується перед передачею будь-яких даних. HTTPS широко використовується для захисту конфіденційної інформації, такої як паролі, фінансові транзакції, особисті дані та інші важливі дані, які передаються через Інтернет. Захищений протокол став стандартом для веб-додатків, оскільки він не лише забезпечує безпеку, але й підвищує довіру користувачів до сайту. Багато сучасних браузерів навіть відображають попередження, якщо сайт не використовує HTTPS, що додатково стимулює перехід на захищені протоколи.

Використання HTTPS має важливе значення для забезпечення цілісності даних. Шифрування гарантує, що дані не будуть змінені під час передачі. Наприклад, якщо клієнт надсилає запит на бронювання через веб-додаток, використання HTTPS забезпечить, що ця інформація буде захищена від можливих атак, таких як "людина посередині" (MITM).

Впровадження HTTPS є відносно простим завдяки сучасним інструментам, таким як сертифікати Let’s Encrypt, які дозволяють автоматизувати процес отримання і оновлення цифрових сертифікатів. Для серверної частини, реалізованої на C++, часто використовуються бібліотеки для підтримки SSL/TLS, такі як OpenSSL. HTTPS не лише підвищує безпеку, але й позитивно впливає на SEO, оскільки пошукові системи віддають перевагу сайтам із захищеним з’єднанням.

Обрана архітектура та технології відповідають вимогам проєкту, забезпечують продуктивність, надійність та масштабованість системи. Це дозволяє створити сучасний веб-додаток, який легко підтримувати та розширювати, враховуючи майбутні потреби користувачів.

2.2 Встановлення програмного забезпечення та підготовка середовища розробки

Процес встановлення програмного забезпечення та налаштування середовища розробки є важливим етапом підготовки до роботи над проєктом. Для реалізації веб-додатку з бронювання футбольного поля обрано технології, які вимагають певних інструментів та налаштувань. Основними компонентами середовища розробки є C++ для серверної частини, SQLite для зберігання даних, а також HTML, CSS і JavaScript для клієнтської частини.

Для роботи з серверною частиною на мові C++ було обрано текстовий редактор Visual Studio Code (VS Code), який забезпечує зручність у написанні коду та інтеграцію з іншими інструментами. Спершу необхідно встановити сам редактор, завантаживши його з офіційного сайту. Після цього додаються необхідні розширення, такі як C/C++ Extension Pack, що дозволяє працювати з кодом C++ у VS Code.

Visual Studio Code (VS Code) – це безкоштовний, кросплатформений текстовий редактор, створений корпорацією Microsoft. Його популярність зумовлена багатофункціональністю, швидкістю роботи та широкою підтримкою мов програмування. Основна мета VS Code – забезпечити розробників інструментом, який поєднує легкість текстового редактора з можливостями повноцінного інтегрованого середовища розробки (IDE). Він працює на Windows, macOS і Linux, що робить його універсальним вибором для будь-якого розробника.

VS Code має інтуїтивно зрозумілий інтерфейс із можливістю налаштування. Розробники можуть змінювати кольорові теми, макети панелей і шорткати, що дозволяє адаптувати середовище під власні потреби. Окрім цього, редактор підтримує широкий спектр розширень через вбудований магазин, що дозволяє додавати функціональність, наприклад, інтеграцію з Git, підтримку додаткових мов програмування чи спеціалізованих інструментів для роботи з базами даних, тестуванням чи налаштуваннями серверів.

Однією з головних особливостей VS Code є його підтримка IntelliSense – розумної системи автодоповнення, яка аналізує код і пропонує релевантні варіанти. IntelliSense підтримує не лише ключові слова мов програмування, а й функції, змінні, імпорти та навіть коментарі. Це значно підвищує швидкість розробки й зменшує кількість помилок. Для налагодження коду VS Code має вбудований дебагер, який підтримує багато мов програмування, включаючи C++, Python, JavaScript та інші. Дебагер дозволяє встановлювати точки зупинки, переглядати значення змінних і навіть виконувати складні сценарії для аналізу помилок. Це робить VS Code не лише редактором для написання коду, а й повноцінним інструментом для його тестування та оптимізації.

Однією з сильних сторін VS Code є інтеграція з Git. У редакторі доступні всі основні функції системи контролю версій, такі як коміти, створення гілок, вирішення конфліктів і перегляд історії змін. Це дозволяє розробникам керувати своїми проєктами безпосередньо з редактора, не вдаючись до сторонніх інструментів.

VS Code активно використовується для веб-розробки завдяки чудовій підтримці HTML, CSS і JavaScript. Для цих мов доступні численні розширення, які забезпечують автодоповнення, лінтинг і навіть прев'ю сторінок у реальному часі. Наприклад, плагіни для роботи з фреймворками Angular, React чи Vue.js дозволяють інтегрувати специфічні функції, які спрощують розробку клієнтської частини.

Для серверної розробки на C++ VS Code пропонує розширення для підтримки компіляторів, дебагерів і бібліотек, таких як CMake та vcpkg. Це дозволяє легко налаштувати середовище для створення серверів із використанням сучасних фреймворків, таких як Crow або Boost.

Крім основного функціоналу, VS Code підтримує інтеграцію із системами CI/CD, такими як Jenkins чи Azure DevOps, що дозволяє автоматизувати процеси тестування та деплою безпосередньо з редактора. Це особливо важливо для команд, які працюють над великими проєктами, де автоматизація є невід'ємною частиною робочого процесу.

Завдяки своїй гнучкості, продуктивності та широкій екосистемі розширень Visual Studio Code став незамінним інструментом для багатьох розробників по всьому світу. Його переваги – швидкість роботи, простота налаштування та потужний функціонал – дозволяють використовувати його для будь-яких завдань, від написання простого коду до створення масштабних веб-додатків і серверних систем.

Наступним кроком є встановлення компілятора C++ для збирання та виконання серверного коду. На Windows найзручнішим вибором є Microsoft Build Tools, які входять до складу Visual Studio. Після встановлення необхідно додати шлях до компілятора (cl.exe) у змінні середовища PATH, щоб забезпечити доступ до нього з терміналу.

Для роботи з фреймворком Crow необхідно встановити пакетний менеджер vcpkg, який дозволяє легко додати бібліотеки до проєкту. Після завантаження та налаштування vcpkg виконується команда для встановлення Crow і SQLite:

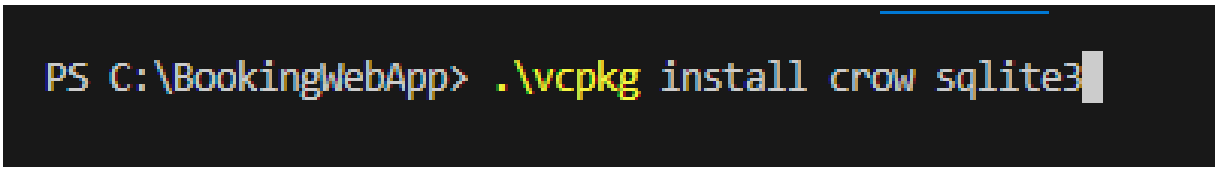

Рисунок 2.2 – Зображення команди для встановлення CROW і SQLite3

Після встановлення програмного забезпечення потрібно налаштувати середовище розробки для роботи з проєктом. У VS Code створюється робоча директорія, яка містить всі необхідні файли, такі як вихідний код, конфігураційні файли та ресурси додатку. Для збирання проєкту використовується CMake – інструмент, що дозволяє налаштувати процес компіляції.

У кореневій директорії проєкту створюється файл CMakeLists.txt, який описує конфігурацію проєкту:

cmake_minimum_required(VERSION 3.10)

```
project(FootballBookingServer)

set(CMAKE_CXX_STANDARD 17)

add_executable(server main.cpp)
target_link_libraries(server PRIVATE ${CROW_LIBRARIES} sqlite3).
```

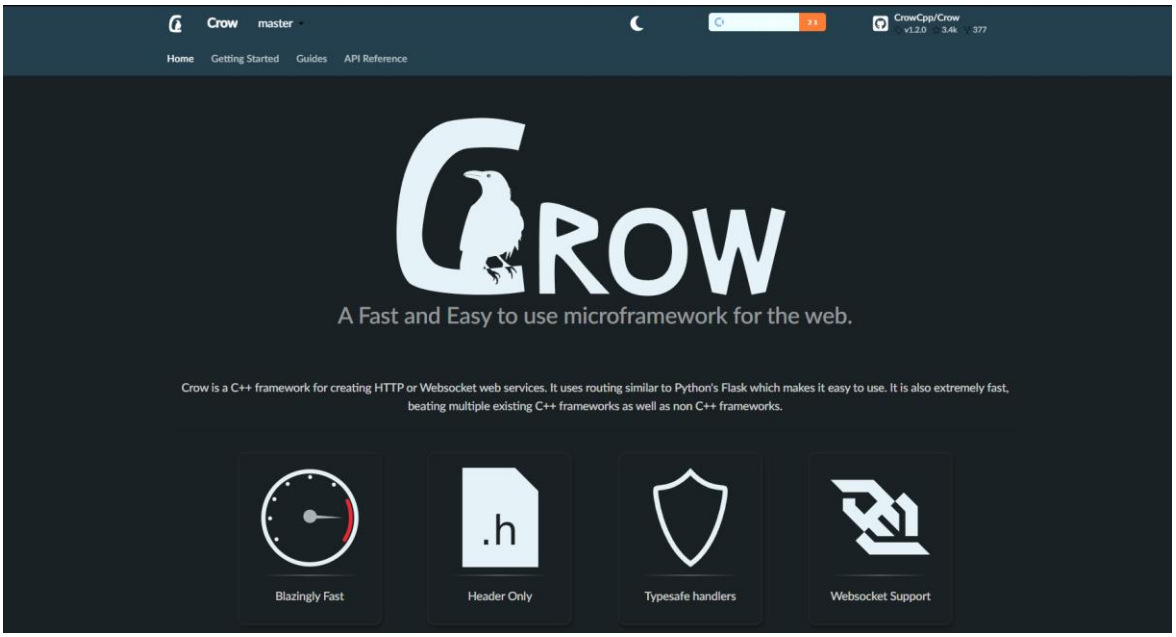

Рисунок 2.3 – Зображення сайту для завантаження фреймворку CROW

Для роботи з CMake у VS Code встановлюється розширення CMake Tools, яке забезпечує інтеграцію з редактором.

Для зберігання інформації про користувачів, бронювання та доступні дати було обрано SQLite. Ця база даних не потребує додаткових налаштувань, адже всі дані зберігаються у вигляді одного файлу. Щоб взаємодіяти з SQLite у C++, підключається бібліотека, яка входить до складу vcpkg.

У базі даних створюється таблиця для зберігання бронювань:

```
CREATE TABLE bookings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    date TEXT,
```

```
    time_slot TEXT,
    status TEXT
);
```

Файл бази даних зберігається у кореневій директорії проєкту, а сервер взаємодіє з ним через API SQLite. Клієнтська частина створюється за допомогою HTML, CSS і JavaScript. Для розробки використовується текстовий редактор VS Code. У кореневій директорії проєкту створюється папка public, яка містить HTML-файли, стилі та JavaScript-скрипти. Основний файл index.html відповідає за структуру інтерфейсу, а стилі у файлі styles.css забезпечують візуальне оформлення.

Для взаємодії з сервером використовується JavaScript через функції fetch(), які надсилають запити до RESTful API. Наприклад, завантаження доступних дат реалізується наступним кодом:

```
fetch('/api/available-dates')
    .then(response => response.json())
    .then(data => {
        console.log(data);
    });
```

Після налаштування середовища виконується тестовий запуск серверної частини. Сервер запускається за допомогою наступної команди:

```
PS C:\BookingWebApp> .\build\server.exe
```

Рисунок 2.4 – Зображення команди для запуску сервера

Встановлення програмного забезпечення та налаштування середовища розробки забезпечили готовність до реалізації проєкту. Використання сучасних інструментів, таких як VS Code, CMake, vcpkg і SQLite, дозволяє ефективно створювати серверну та клієнтську частини, забезпечуючи

продуктивну роботу над додатком. Отриманий результат створює надійну основу для подальшої розробки функціональності веб-додатку.

2.3 Побудова архітектури сервера на C++ з використанням Crow

Серверна частина веб-додатку реалізована на C++ з використанням фреймворку Crow, який забезпечує просту та ефективну роботу з RESTful API. Crow дозволяє легко налаштовувати маршрути для обробки запитів, працювати з JSON-форматом і підтримує багатопоточність для обробки декількох клієнтських запитів одночасно. Це робить його ідеальним вибором для створення продуктивного та масштабованого веб-сервера.

Архітектура сервера побудована таким чином, щоб забезпечити чітке розділення логіки. Основні компоненти включають маршрути для обробки HTTP-запитів, модулі для взаємодії з базою даних SQLite та механізми для забезпечення безпеки. На етапі ініціалізації сервер створюється як екземпляр crow::SimpleApp, де визначаються всі маршрути та налаштовується багатопоточний режим для підвищення продуктивності. Наприклад, сервер запускається на порту 8080 із можливістю обробки паралельних запитів.

Для взаємодії з базою даних використовується SQLite, що забезпечує зберігання інформації про користувачів, бронювання та статуси днів у календарі. База даних підключається при запуску сервера, а запити до неї виконуються через SQL-інструкції. Наприклад, сервер може отримати список доступних дат, виконуючи SQL-запит до таблиці бронювань, і повертати ці дані у вигляді JSON-формату. Такі запити інтегруються з маршрутами API, які обробляють клієнтські запити на отримання, створення чи скасування бронювання.

Кожен маршрут API відповідає за певну функцію. Наприклад, маршрут для створення бронювання приймає POST-запит із даними про дату, час і користувача. Ці дані перевіряються на коректність і додаються в базу даних.

У разі успішного виконання сервер повертає підтвердження, яке клієнтська частина використовує для оновлення інтерфейсу.

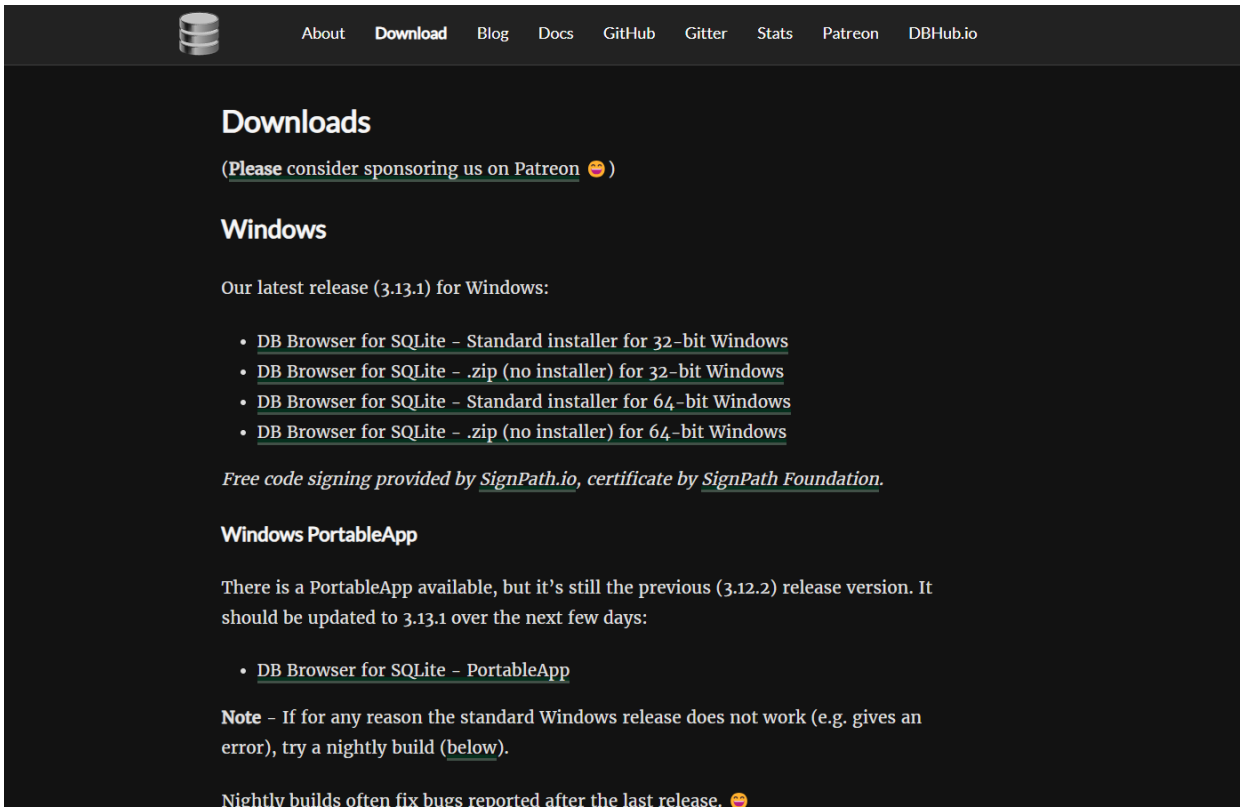

Рисунок 2.5 – Зображення сайту з SQLite3

Безпека є важливим аспектом серверної архітектури. Паролі користувачів шифруються перед збереженням у базу даних, а запити клієнтів перевіряються на наявність можливих вразливостей, таких як SQL-ін'єкції. Крім того, використання HTTPS забезпечує захист даних, які передаються між клієнтом і сервером.

Сервер працює в багатопоточному режимі, що дозволяє обробляти кілька запитів одночасно. Це досягається завдяки вбудованим можливостям Crow і дозволяє додатку залишатися продуктивним навіть під високим навантаженням. Інтеграція з клієнтською частиною через RESTful API забезпечує зручну взаємодію, а використання JSON-формату спрощує передачу та обробку даних.

Сервер, що працює в багатопоточному режимі, здатний обробляти кілька запитів одночасно, розподіляючи навантаження між різними потоками. Багатопоточність забезпечує високу продуктивність і ефективність, особливо в умовах, коли сервер отримує великий обсяг одночасних запитів від клієнтів. Такий підхід дозволяє уникнути блокування основного потоку під час виконання тривалих операцій, наприклад, доступу до бази даних або роботи з файловою системою. У багатопоточному сервері кожен запит може оброблятися окремим потоком, що значно скорочує час очікування для клієнтів.

У контрасті з однопоточним сервером, де всі запити обробляються послідовно, багатопоточність дозволяє досягти більшого рівня паралелізму. Однопоточний сервер може бути ефективним для простих додатків із низьким навантаженням, оскільки в ньому відсутні складнощі, пов’язані з управлінням потоками. Однак, якщо в черзі накопичується велика кількість запитів, однопоточний сервер змушений обробляти їх послідовно, що призводить до затримок і може викликати невдоволення користувачів.

Багатопоточний сервер краще використовує ресурси процесора, розподіляючи роботу між доступними ядрами. Наприклад, у сервері на C++ з використанням бібліотеки Crow багатопоточність дозволяє запускати обробку HTTP-запитів паралельно, кожен у власному потоці. Це досягається за допомогою механізмів багатопоточності, таких як бібліотека std::thread у стандартній бібліотеці C++ або більш сучасні інструменти, як-от std::async.

У багатопоточному сервері потрібно враховувати складнощі, пов’язані із синхронізацією доступу до спільних ресурсів, наприклад, бази даних або глобальних змінних. Якщо кілька потоків одночасно змінюють спільні дані, це може призвести до стану гонки (race condition) або інших проблем. Для вирішення таких ситуацій використовуються механізми синхронізації, такі як м’ютекси (std::mutex) або умови очікування (std::condition_variable).

Однопоточний сервер, у свою чергу, не потребує додаткової синхронізації, оскільки всі запити виконуються в одному потоці, послідовно.

Це значно спрощує логіку програмування, але робить сервер вразливим до перевантаження. Наприклад, якщо одна операція займає багато часу (наприклад, складний запит до бази даних), усі інші запити будуть змушені чекати завершення цієї операції.

Багатопоточний сервер забезпечує кращу масштабованість у сучасних багатоядерних системах. Наприклад, сервер, який запускає 8 потоків на машині з 8 ядрами, може одночасно обробляти запити з використанням усіх ядер процесора, тоді як однопоточний сервер використовуватиме лише одне ядро. Однак багатопоточність також ускладнює розробку та налагодження, оскільки помилки, пов’язані з потоками, часто важко виявити та виправити. Вибір між однопоточним і багатопоточним сервером залежить від специфіки завдання. Однопоточний сервер може бути достатнім для невеликих додатків із низьким навантаженням, але для складних систем із великою кількістю одночасних клієнтів багатопоточність є необхідною для забезпечення швидкої і стабільної роботи. Багатопоточний сервер забезпечує високу продуктивність, але вимагає від розробників додаткової уваги до управління потоками і синхронізації даних.

Архітектура сервера на базі C++ і Crow гарантує високу продуктивність, стабільність і масштабованість. Це дозволяє реалізувати всі необхідні функції веб-додатку, забезпечуючи зручність для користувачів і ефективність для адміністрації.

2.4 Висновки до другого розділу

Було проведено аналіз існуючих веб-застосунків для бронювання, їхньої архітектури та функціональних особливостей, а також визначено найкращі практики для реалізації проєкту. Розглянуто популярні рішення, такі як Booking.com, OpenTable та Calendly, які показали високий рівень інтерактивності, зручності для користувачів та ефективності обробки запитів.

Аналіз цих застосунків дозволив виділити ключові елементи, які будуть враховані під час розробки: інтерактивний календар, гнучкість у налаштуванні часових слотів, автоматичні сповіщення та простота користувацького інтерфейсу.

Було зроблено висновок, що клієнт-серверна архітектура є оптимальним вибором для проєкту. Такий підхід дозволяє забезпечити модульність, гнучкість у масштабуванні та зручність підтримки системи. На основі цього обрано технології для реалізації: HTML, CSS і JavaScript для клієнтської частини, а також C++ із використанням Crow для серверної частини. База даних SQLite визначена як основний інструмент для зберігання інформації про користувачів і бронювання, оскільки вона забезпечує простоту в роботі, продуктивність і стабільність. Результати аналізу показали, що основними недоліками існуючих систем є перевантаження функціями, складність інтерфейсу для нових користувачів та висока ресурсомісткість. Врахування цих недоліків дозволить уникнути подібних проблем у розроблюваному додатку. Особлива увага буде приділена створенню легкого, інтуїтивно зрозумілого інтерфейсу, який забезпечить високу швидкість роботи навіть на пристроях із обмеженими ресурсами.

Таким чином, другий розділ закладає міцну основу для подальшої розробки додатку, враховуючи сильні сторони та недоліки існуючих рішень. Отримані висновки дозволяють сформувати чітке бачення функціональних можливостей системи, оптимізувати вибір технологій та визначити напрями для подальшої роботи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ БРОНЮВАННЯ ФУТБОЛЬНОГО ПОЛЯ

3.1 Реалізація серверної частини: обробка запитів та взаємодія з базою даних

Серверна частина є критично важливим компонентом веб-додатку, адже саме вона відповідає за обробку запитів від клієнтської частини, взаємодію з базою даних і забезпечення цілісності та безпеки даних. У даному проєкті серверна частина реалізована на мові програмування C++ із використанням фреймворку Crow та бази даних SQLite.

Сервер побудований за принципом RESTful API, що дозволяє клієнтській частині взаємодіяти з сервером через HTTP-запити. Кожен маршрут відповідає за виконання конкретної операції: отримання даних, створення запису, оновлення чи видалення.

Фреймворк Crow забезпечує простий і зручний інструмент для визначення маршрутів. Наприклад, для отримання списку доступних днів у календарі можна створити маршрут:

```
CROW_ROUTE(app, "/api/slots")([]() {
    auto slots = getAvailableSlots();
    crow::json::wvalue res;
    res["slots"] = slots;
    return crow::response(res);
});
```

Рисунок 3.1 – Зображення коду для отримання списку доступних днів

Запити на сервер обробляються у багатопоточному режимі, що дозволяє одночасно обслуговувати декілька клієнтів без значних затримок.

SQLite обрано як база даних завдяки її легкості, автономності та високій продуктивності для невеликих проєктів. База даних містить таблиці для зберігання інформації про користувачів, бронювання та статуси днів у календарі.

```
    sqlite3_open("booking.db", &db);
    std::string createTable = R"(
        CREATE TABLE IF NOT EXISTS bookings (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            time TEXT
        );
```

Рисунок 3.2 – Зображення коду для створення таблиці бронювань

```
void selectBAase()
{
    sqlite3* db;
    sqlite3_open("database.db", &db);

    std::string query = "SELECT date FROM bookings WHERE status = 'booked'";
    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, query.c_str(), -1, &stmt, nullptr);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
    const char* date = reinterpret_cast<const char*>(sqlite3_column_text(stmt, 0));
    std::cout << "Booked date: " << date << std::endl;
}

sqlite3_finalize(stmt);
sqlite3_close(db);
}
```

Рисунок 3.3 – Зображення коду для отримання списку забронюваних ячеек в календарі

Серверна частина веб-додатку реалізує кілька основних маршрутів, які забезпечують обробку запитів від клієнтської частини та взаємодію з базою

даних. Кожен маршрут відповідає за виконання конкретної функції та є важливим для роботи всієї системи.

Маршрут GET /api/available-dates відповідає за повернення списку доступних для бронювання дат. Коли клієнтська частина надсилає запит на цей маршрут, сервер отримує актуальні дані з бази даних SQLite про доступні дні. Результат передається у вигляді JSON-масиву, що дозволяє клієнту динамічно оновлювати інтерфейс календаря. Цей маршрут забезпечує користувачам доступ до актуальної інформації без необхідності ручного оновлення сторінки.

Маршрут POST /api/book призначений для обробки запитів на створення нового бронювання. Користувач надсилає дані, такі як обрана дата, час і свої контактні дані, через форму на клієнтській частині. Сервер приймає ці дані, перевіряє їх коректність, а потім додає запис у таблицю бронювань у базі даних. Якщо операція виконується успішно, сервер повертає підтвердження, яке інформує користувача про успішне створення бронювання. У разі виникнення помилок сервер надсилає відповідний код помилки.

Маршрут DELETE /api/cancel/:id використовується для скасування бронювання. Користувач або адміністратор надсилає запит із зазначенням ідентифікатора бронювання, яке потрібно скасувати. Сервер отримує цей ідентифікатор, знаходить відповідний запис у базі даних і видаляє його. Цей маршрут корисний для обробки ситуацій, коли клієнт змінює свої плани або коли адміністратор хоче звільнити час для технічного обслуговування.

Маршрут POST /api/register відповідає за реєстрацію нових користувачів. Користувач надсилає дані, такі як ім’я, електронна пошта та пароль, які сервер перевіряє на коректність. Пароль перед збереженням у базу даних шифрується для забезпечення безпеки. У разі успішної реєстрації сервер створює новий запис у таблиці користувачів і надсилає клієнту підтвердження.

Маршрут POST /api/login реалізує авторизацію користувачів. Користувач вводить свої облікові дані (електронну пошту та пароль), які сервер порівнює з даними в базі. Якщо перевірка успішна, сервер генерує

токен авторизації, який використовується для доступу до захищених маршрутів. У разі невдалої авторизації клієнт отримує повідомлення про помилку.

Таким чином, кожен маршрут API виконує специфічну функцію, яка забезпечує повноцінну роботу додатку. Вони інтегруються з базою даних для отримання та збереження інформації, а також забезпечують динамічну та безпечну взаємодію між сервером і клієнтом.

```
    // Роут для створення бронювання
    CROW_ROUTE(app, "/api/book").methods("POST"_method)([&db](const crow::request& req) {
    auto body = crow::json::load(req.body);
    std::string date = body["date"].s();
    std::string time_slot = body["time_slot"].s();
    int user_id = body["user_id"].i();

    std::string query = "INSERT INTO bookings (user_id, date, time_slot, status) VALUES (?, ?, ?, 'booked')";
    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, query.c_str(), -1, &stmt, nullptr);
    sqlite3_bind_int(stmt, 1, user_id);
    sqlite3_bind_text(stmt, 2, date.c_str(), -1, SQLITE_STATIC);
    sqlite3_bind_text(stmt, 3, time_slot.c_str(), -1, SQLITE_STATIC);

    if (sqlite3_step(stmt) == SQLITE_DONE) {
        sqlite3_finalize(stmt);
        return crow::response(200, "Booking successful");
    } else {
        sqlite3_finalize(stmt);
        return crow::response(500, "Failed to book");
    }
});
```

Рисунок 3.4 – Зображення коду створення нового запиту для бронювання

Для захисту даних користувачів і запобігання несанкціонованому доступу на сервері реалізовано кілька механізмів безпеки:

– шифрування паролів користувачів перед збереженням у базу даних (наприклад, через бібліотеку OpenSSL);

– використання токенів авторизації для перевірки прав доступу до захищених маршрутів;

– перевірка даних, що надходять від клієнтів, щоб уникнути SQL-ін’єкцій і XSS-атак.

Реалізація серверної частини забезпечує ефективну обробку запитів, зберігання та маніпуляцію даними про бронювання. Завдяки використанню

C++ та SQLite, сервер гарантує високу продуктивність і стабільність, а інтеграція з клієнтською частиною через RESTful API дозволяє створити зручний і надійний веб-додаток.

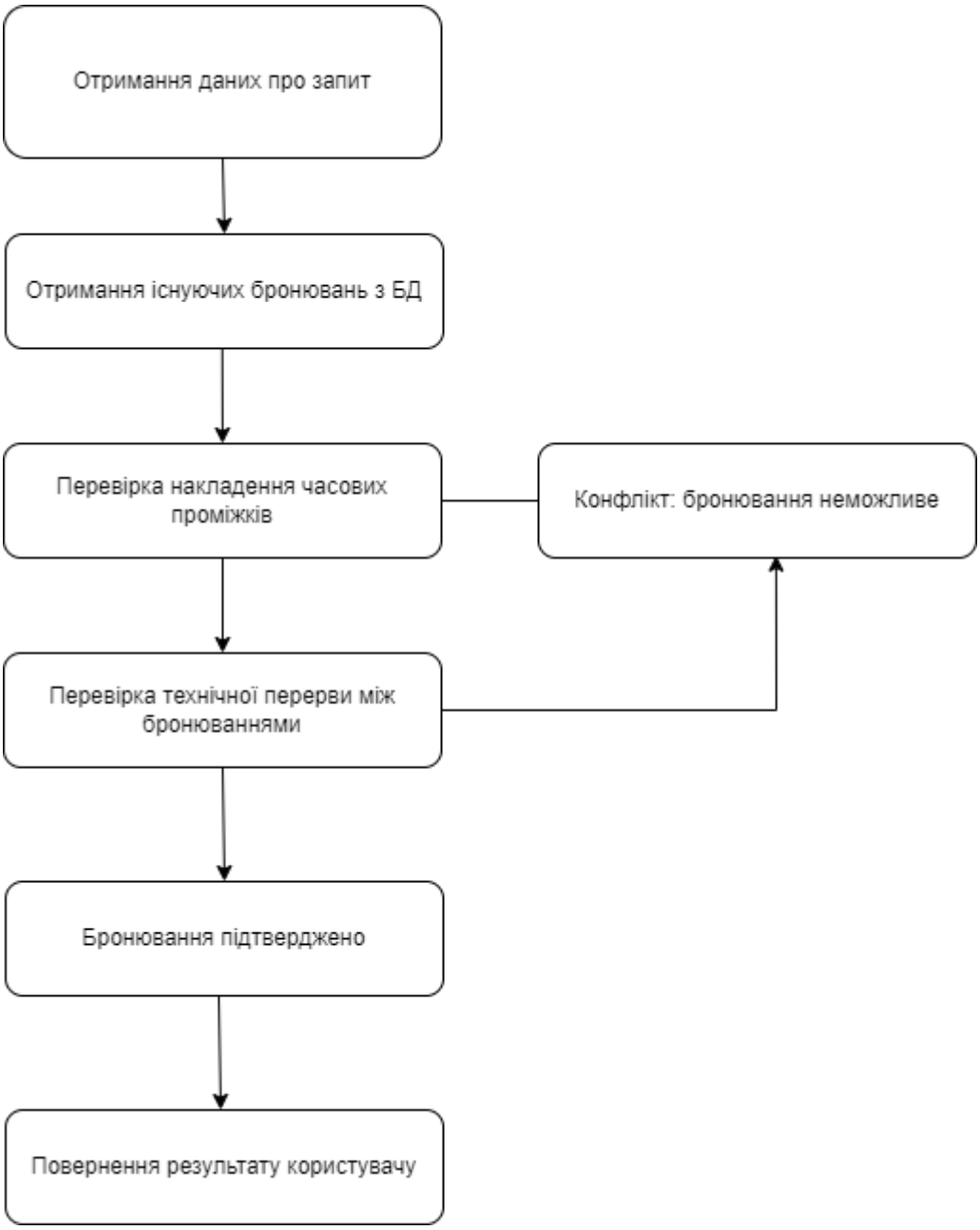

Рисунок 3.5 – Зображення блок-схеми алгоритму перевірки конфліктів під час бронювання

Алгоритм перевірки конфліктів під час бронювання має на меті забезпечити правильність і надійність створення нових бронювань, уникнувши накладень та порушень у розкладі. На першому етапі сервер отримує дані про запит, включаючи часовий проміжок нового бронювання та

відповідну дату. Далі система звертається до бази даних, щоб отримати всі існуючі бронювання для зазначеного періоду.

Після отримання даних відбувається аналіз можливих конфліктів. Спершу перевіряється, чи перетинається часовий проміжок нового бронювання з уже існуючими записами. Якщо таке накладення виявлено, алгоритм завершує роботу з повідомленням про неможливість бронювання. У разі відсутності накладень система переходить до наступного етапу — перевірки технічної перерви. Цей етап враховує час, необхідний для підготовки або обслуговування між бронюваннями. Якщо буферний інтервал не дотримано, алгоритм також завершується повідомленням про конфлікт.

Якщо ж усі перевірки пройдені успішно, нове бронювання визнається дійсним. Алгоритм завершується поверненням позитивної відповіді користувачу. Завдяки такій послідовності дій забезпечується цілісність розкладу та уникнення помилок у створенні бронювань, а врахування технічних перерв додає гнучкості й адаптивності системі. Алгоритм є універсальним і може бути адаптований для різних систем бронювання, підвищуючи їхню ефективність і зручність використання.

3.2 Розробка інтерактивного веб-інтерфейсу: календар, форми та інтеграція з сервером

Веб-інтерфейс є важливою частиною додатку, яка забезпечує взаємодію користувача із системою. Для реалізації інтерактивності та зручності використання в цьому проєкті застосовано сучасні технології веб-розробки: HTML для створення структури сторінки, CSS для оформлення та JavaScript для додавання динамічності. Основними елементами веб-інтерфейсу є інтерактивний календар і форми для реєстрації, авторизації та бронювання, які інтегровані з сервером для забезпечення функціональності.

Календар є центральним елементом інтерфейсу, який дозволяє користувачам переглядати доступні дати та здійснювати бронювання.

Календар розроблений з використанням HTML для структури, CSS для стилізації та JavaScript для динамічної роботи.

HTML забезпечує базову структуру календаря. Наприклад, кожен день представлено у вигляді окремого -елемента. JavaScript використовується для динамічного заповнення календаря даними з сервера. Дані про доступність днів надходять у форматі JSON, після чого генеруються відповідні елементи HTML. CSS додає візуальне оформлення, наприклад, виділення доступних днів зеленим кольором, а заброньованих – червоним.

Користувач може вибрати доступний день, після чого відкривається форма для подальшого бронювання. Вибір дати здійснюється за допомогою JavaScript, який реагує на події кліку і відправляє відповідний запит до сервера.

Форми використовуються для збору інформації від користувачів, наприклад, для реєстрації, авторизації та здійснення бронювання. HTML надає основну структуру форм із полями вводу, кнопками та іншими елементами. CSS додає стилізацію, щоб форми були візуально привабливими й зручними для заповнення.

JavaScript забезпечує обробку даних форм на клієнтському боці перед їх відправкою на сервер. Наприклад, перед надсиланням форми реєстрації перевіряється, чи всі поля заповнені, чи відповідають вони необхідному формату (наприклад, чи валідний формат електронної пошти).

Форма для бронювання включає поля для вибору часу, введення контактних даних і підтвердження дати. Після заповнення даних і натискання кнопки «Забронювати» форма надсилає запит на сервер, який обробляє інформацію і відповідає статусом операції.

Інтеграція веб-інтерфейсу з сервером реалізована через RESTful API. JavaScript використовує функцію fetch() для надсилання HTTP-запитів до серверних маршрутів, отримання даних і оновлення інтерфейсу без перезавантаження сторінки.

При завантаженні сторінки JavaScript надсилає запит до маршруту /api/available-dates, щоб отримати список доступних і заброньованих дат. На основі цих даних генерується календар. Коли користувач обирає дату для бронювання, JavaScript надсилає POST-запит до маршруту /api/book із переданими параметрами, такими як обрана дата, час і контактні дані.

```
fetch('/api/available-dates')
    .then(response => response.json())
    .then(data => {
        data.forEach((day, index) => {
            const dayElement = document.createElement('div');
            dayElement.classList.add('day');
            dayElement.textContent = index + 1;
            if (day.available) {
                dayElement.classList.add('available');
                dayElement.addEventListener('click', () => selectDate(index + 1));
            } else {
                dayElement.classList.add('booked');
            }
            document.querySelector('.calendar').appendChild(dayElement);
        });
    });
```

Рисунок 3.6 – Зображення коду для отримання доступних дат

```
document.querySelector('#bookingForm').addEventListener('submit', (e) => {
    e.preventDefault();
    const formData = new FormData(e.target);
    fetch('/api/book', {
        method: 'POST',
        body: JSON.stringify({
            date: formData.get('date'),
            time: formData.get('time'),
            user: formData.get('user')
        }),
        headers: {
            'Content-Type': 'application/json'
        }
    })
    .then(response => response.json())
    .then(data => alert(data.message));
});
```

Рисунок 3.7 – Зображення коду для обробки форми бронювання

Розробка інтерактивного веб-інтерфейсу базується на використанні HTML, CSS і JavaScript для створення зручного, динамічного та функціонального інтерфейсу. Інтеграція з сервером забезпечує актуальність даних, що відображаються в календарі, і дозволяє користувачам легко виконувати всі необхідні операції, від реєстрації до бронювання. Календар,

форми та зручна інтеграція з сервером роблять веб-додаток ефективним інструментом для автоматизації процесу бронювання.

Процес покращення, перевірки, тестування та оптимізації веб-застосунку є ключовим етапом розробки, який спрямований на забезпечення стабільності, продуктивності та зручності використання. Покращення охоплюють як функціональні, так і нефункціональні аспекти системи. Наприклад, на клієнтській стороні додатку можна оптимізувати час завантаження сторінки через мініфікацію CSS і JavaScript, застосування кешування статичних ресурсів та скорочення кількості запитів до сервера. На серверній стороні важливими аспектами є покращення роботи з базою даних через оптимізацію SQL-запитів та впровадження механізмів кешування для зниження навантаження.

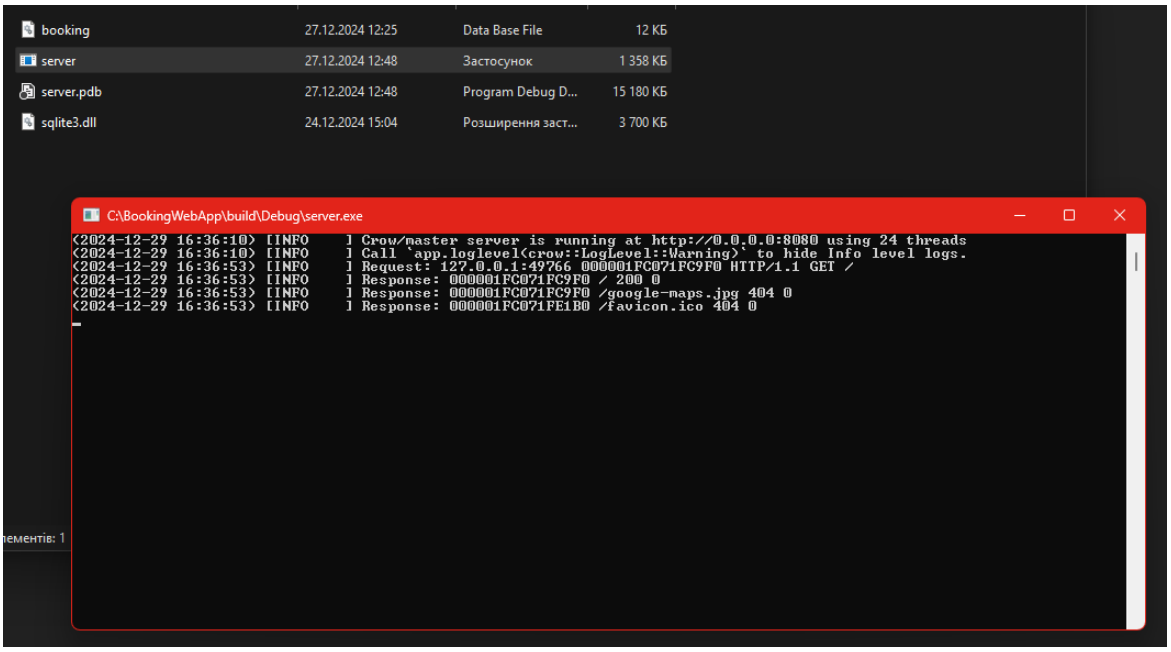

Рисунок 3.8 – Зображення запуску сервера server.exe

Перевірка функціональності передбачає тестування роботи інтерактивного інтерфейсу, наприклад, календаря, який має правильно відображати доступні та заброньовані дати. Тестування охоплює перевірку коректності обробки запитів до серверної частини, таких як створення,

скасування та отримання бронювань. Інтеграційне тестування забезпечує коректну взаємодію клієнтської частини із серверною через RESTful API.

Оптимізація серверної частини включає перевірку її здатності обробляти запити під великим навантаженням за допомогою стрес-тестів. Для цього використовуються інструменти, які дозволяють симулювати велику кількість одночасних клієнтів, перевіряючи продуктивність системи та швидкість виконання операцій. Додатково оптимізація може включати багатопоточну обробку запитів та аналіз затримок при взаємодії із базою даних SQLite.

Фінальне тестування проводиться для перевірки стабільності веб-додатку на різних пристроях і браузерах. Особлива увага приділяється адаптивності інтерфейсу, щоб гарантувати зручність використання на мобільних та настільних пристроях. Регресійне тестування допомагає переконатися, що нововведення не вплинули негативно на існуючий функціонал.

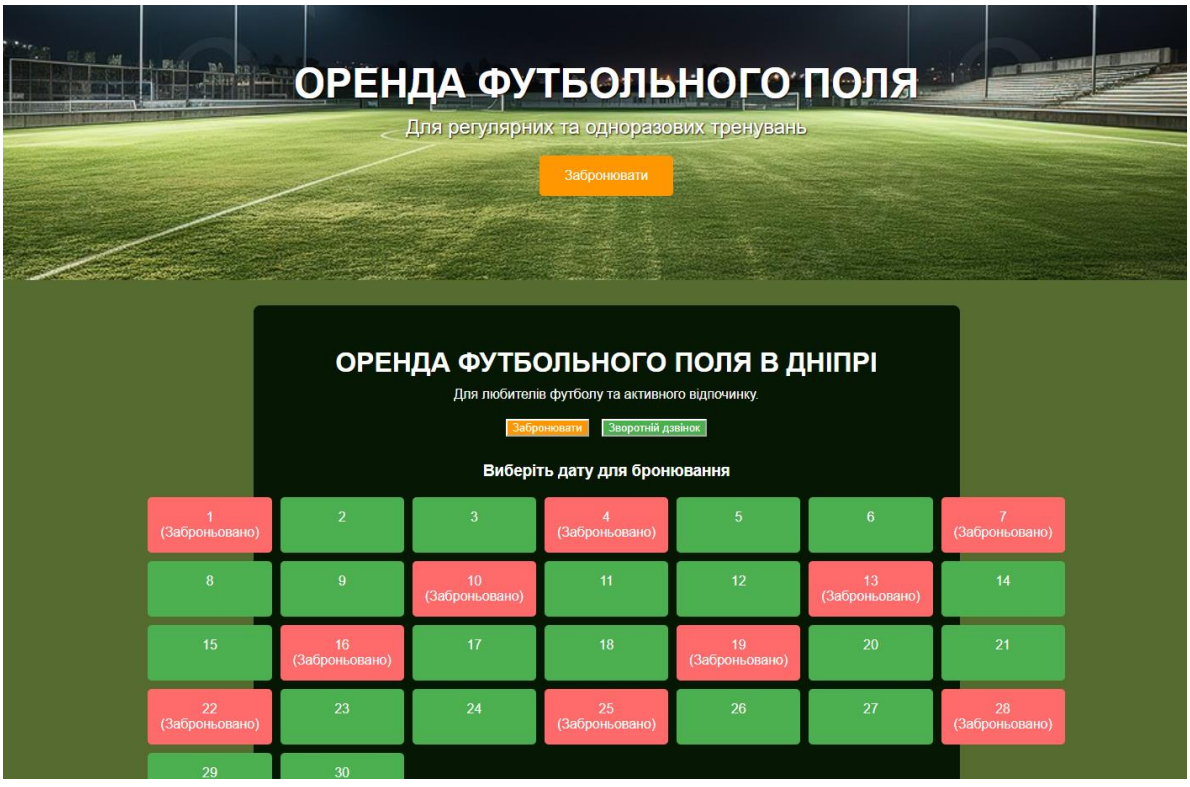

Рисунок 3.9 – Зображення вигляду веб-застосунку

Після завершення усіх етапів тестування і оптимізації веб-застосунок готовий до використання. Висока продуктивність, стабільність і зручність

інтерфейсу стають основними показниками успішної розробки та впровадження системи.

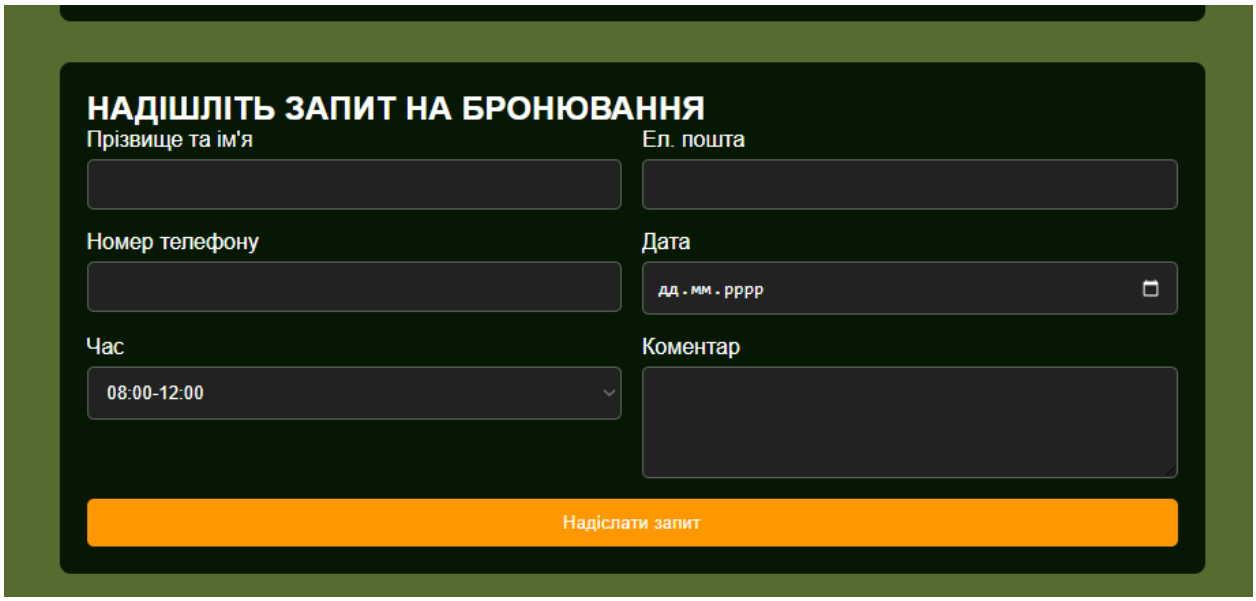

Рисунок 3.10 – Зображення запит на бронювання

Запит на бронювання є одним із ключових елементів функціональності веб-додатку, який забезпечує взаємодію користувача із системою для резервування доступних слотів. Коли користувач обирає дату та час у календарі, створюється запит до серверної частини додатку, яка перевіряє доступність обраного слоту та зберігає інформацію про бронювання у базі даних. Цей процес реалізується через RESTful API, що дозволяє ефективно організувати обмін даними між клієнтською та серверною частинами.

Запит на бронювання зазвичай виконується через HTTP POST-запит. У його тілі передаються дані, такі як обрана дата, час та інформація про користувача (наприклад, ім’я чи унікальний ідентифікатор). Сервер отримує ці дані, виконує перевірку на відповідність вимогам і наявність вільного слоту. Якщо слот доступний, сервер додає запис у базу даних, фіксуючи інформацію про бронювання, і повертає користувачеві відповідь про успішне завершення операції. У разі конфлікту, наприклад, якщо слот уже зайнятий, сервер повертає відповідь із повідомленням про помилку.

Для забезпечення зручності користувачів у клієнтській частині додатку інтегрована функція динамічного оновлення. Після успішного створення бронювання календар автоматично оновлюється, позначаючи обраний слот як заброньований. Це досягається завдяки використанню JavaScript і API-запитів без необхідності перезавантаження сторінки.

Важливим аспектом є безпека запитів на бронювання. Для захисту даних користувачів система повинна забезпечувати перевірку вхідних даних і запобігання можливим атакам, таким як SQL-ін’єкції. Додатково може використовуватися авторизація користувача, наприклад, через токен, щоб гарантувати, що тільки автентифіковані користувачі мають право створювати бронювання.

Запит на бронювання також враховує транзакційність операцій, особливо у випадку великих систем із кількома користувачами, які можуть одночасно намагатися забронювати один і той самий слот. Використання транзакцій у базі даних забезпечує цілісність даних і виключає можливість дублювання бронювань.

3.3 Тестування, оптимізація та покращення веб-застосунку

Тестування та оптимізація програми є ключовими аспектами, що забезпечують її надійну роботу, високу продуктивність і готовність до реального використання. Процес тестування починається з функціонального тестування, яке включає перевірку роботи всіх основних маршрутів API. Наприклад, під час створення бронювання необхідно переконатися, що сервер коректно обробляє POST-запити, зберігає інформацію в базі даних і повертає відповідь із підтвердженням. Для цього використовуються інструменти, такі як Postman, які дозволяють вручну відправляти запити й отримувати відповіді. Щоб уникнути впливу людського фактору, функціональність можна тестувати

автоматизовано за допомогою бібліотек для C++, таких як Google Test, які дозволяють програмно перевіряти правильність роботи різних функцій.

Навантáжувальне тестування дозволяє оцінити, як сервер працює під великим навантаженням. Наприклад, інструмент Apache JMeter дає змогу симулювати сотні або навіть тисячі одночасних запитів, що надходять до серверу. Це дає змогу зрозуміти, чи може сервер обробляти високий трафік без значних затримок або відмов у роботі. Під час такого тестування звертається увага на час відповіді, кількість успішно оброблених запитів і можливі помилки, які можуть виникати під навантаженням.

Тестування коректності даних є важливим для перевірки роботи з базою даних. Наприклад, необхідно переконатися, що після створення бронювання в SQLite зберігаються всі необхідні дані, а операції видалення або оновлення не призводять до порушення цілісності інформації. Для цього можна використовувати SQL-запити вручну або автоматизовані тести, які перевіряють стан бази даних до і після виконання операцій.

Оптимізація роботи програми починається з бази даних. Використання індексів для часто запитуваних колонок у таблицях SQLite значно підвищує швидкість виконання запитів. Крім того, підготовлені запити (prepared statements) дозволяють мінімізувати накладні витрати на парсинг SQL-коду, що особливо важливо для програм із великою кількістю запитів. Наприклад, якщо сервер обробляє запити на створення бронювань, підготовлений запит дозволяє багаторазово використовувати заздалегідь скомпільований SQL-код із різними параметрами. Кешування є ефективним способом зменшення навантаження на базу даних і підвищення швидкості відповіді сервера. Якщо дані, такі як доступні дати для бронювання, змінюються рідко, їх можна зберігати в оперативній пам’яті. Це дозволяє серверу відповідати на запити набагато швидше, оскільки доступ до пам’яті значно швидший, ніж до бази даних.

Багатопоточна обробка запитів є ключовим аспектом для серверів із високим трафіком. У багатоядерних системах розподіл обробки між потоками

дозволяє ефективно використовувати ресурси процесора. Наприклад, бібліотека Crow автоматично підтримує багатопоточність, розподіляючи вхідні запити між доступними потоками. Це забезпечує рівномірне навантаження на процесор і дозволяє одночасно обробляти велику кількість запитів. Проте варто враховувати можливі проблеми із синхронізацією, якщо кілька потоків взаємодіють із загальними ресурсами, такими як база даних. Для вирішення цих проблем використовуються механізми синхронізації, наприклад, м’ютекси або блокування. Для зниження затримок у мережі важливо оптимізувати розмір відповіді сервера. Наприклад, використання стиснення, такого як gzip, дозволяє значно зменшити обсяг даних, що передаються клієнту, без втрати інформації. Це особливо важливо для великих об’єктів JSON, які повертаються у відповідь на запити. Крім того, сервер може кешувати статичний вміст, такий як CSS і JavaScript-файли, щоб зменшити кількість запитів до нього.

Профілювання продуктивності дозволяє виявити вузькі місця у програмі, які можуть впливати на її швидкодію. Наприклад, інструмент Valgrind допомагає аналізувати використання пам’яті, тоді як gprof дозволяє виявляти найбільш ресурсоємні функції у програмі. Виявлені проблеми можна оптимізувати, наприклад, переписавши повільні цикли або скоротивши кількість обчислень у критичних секціях коду.

Логування і моніторинг забезпечують відстеження роботи програми в реальному часі. Лог-файли дозволяють записувати інформацію про виконані запити, помилки та час обробки кожного з них. Це корисно для діагностики проблем і виявлення невідповідностей у роботі програми. Моніторинг, наприклад, із використанням Prometheus, дає змогу аналізувати продуктивність сервера у динаміці, виявляти пікові навантаження та налаштовувати алерти у разі досягнення критичних значень.

Захист від перевантаження є важливим для стабільної роботи сервера. Використання механізмів обмеження швидкості запитів (rate limiting) дозволяє запобігти перевантаженню сервера, коли занадто багато запитів надходять

одночасно. Наприклад, якщо один користувач надсилає занадто багато запитів за короткий проміжок часу, сервер може тимчасово заблокувати його, щоб забезпечити стабільну роботу для інших користувачів.

Тестування та оптимізація програми забезпечують її готовність до роботи в реальних умовах із високими вимогами до продуктивності, стабільності та безпеки. Кожен із цих етапів має своє значення, і разом вони гарантують, що сервер буде ефективно виконувати свої завдання та відповідати очікуванням користувачів. Ці процеси спрямовані на забезпечення стабільності, продуктивності та безпеки програми, що особливо важливо для реальних умов експлуатації. Завдяки системному підходу до тестування та оптимізації можна уникнути багатьох проблем і підвищити якість кінцевого продукту.

Тестування програми починається з перевірки її функціональності. Функціональне тестування дозволяє переконатися, що всі основні маршрути API працюють коректно, зокрема, створення бронювань, отримання доступних дат або авторизація користувачів. Таке тестування виконується з використанням інструментів, як-от Postman, які імітують клієнтські запити й дозволяють перевіряти відповіді сервера. Важливо також автоматизувати цей процес за допомогою спеціалізованих бібліотек, таких як Google Test, щоб знизити ризик людської помилки та підвищити ефективність перевірки.

Ще одним важливим аспектом є навантáжувальне тестування, яке дає змогу оцінити, як програма поводиться під великим навантаженням. Інструменти на кшталт Apache JMeter дозволяють симулювати сотні одночасних запитів і аналізувати, чи здатен сервер обробляти їх без значних затримок. Це дозволяє ідентифікувати проблемні місця в архітектурі програми, такі як обмеження у швидкодії бази даних чи недостатнє використання багатопоточності.

Тестування також охоплює перевірку коректності даних, що особливо важливо при роботі з базами даних. Наприклад, після створення нового бронювання необхідно переконатися, що всі дані зберігаються правильно, а

операції оновлення чи видалення не призводять до пошкодження інформації. Для цього розробляються спеціальні сценарії тестування, які дозволяють відстежувати стан бази даних до і після виконання певних операцій.

Після тестування розпочинається етап оптимізації, який має на меті покращити швидкість і надійність програми. Одним із ключових напрямів оптимізації є робота з базою даних. Використання індексів для часто запитуваних даних значно скорочує час виконання SQL-запитів. Також підготовлені запити (prepared statements) дозволяють оптимізувати доступ до бази даних і підвищити безпеку, зокрема, захиститися від SQL-ін’єкцій.

Оптимізація роботи сервера включає реалізацію кешування даних, які часто запитуються, наприклад, списків доступних дат для бронювання. Це дозволяє зменшити кількість запитів до бази даних і скоротити час відповіді. Багатопоточність є ще одним важливим елементом оптимізації, оскільки вона дозволяє одночасно обробляти декілька запитів, використовуючи ресурси багатоядерних процесорів. Це особливо важливо для серверів із високим навантаженням, де одночасно працює багато користувачів.

Не менш важливою є оптимізація взаємодії з клієнтом. Зменшення обсягу переданих даних за допомогою стиснення (наприклад, gzip) дозволяє знизити затримки в мережі й прискорити роботу програми. Окрім того, моніторинг роботи сервера в реальному часі, наприклад, за допомогою Prometheus, дозволяє відстежувати стан системи та оперативно реагувати на зростання навантаження.

Захист програми є окремим напрямом, який охоплює тестування на вразливості. Наприклад, впровадження валідації вхідних даних і використання захищених запитів дозволяє уникнути атак, таких як SQL-ін’єкції чи міжсайтове виконання скриптів (XSS). Також реалізація механізмів обмеження швидкості запитів (rate limiting) запобігає перевантаженню сервера під час пікових навантажень або злочинної активності.

Підсумовуючи, тестування та оптимізація програми є критично важливими етапами розробки, які забезпечують її стабільну роботу,

продуктивність і безпеку. Завдяки ретельній перевірці кожного компоненту й впровадженню сучасних методів оптимізації, можна створити програму, яка буде відповідати високим вимогам користувачів і забезпечувати якісну взаємодію в реальних умовах експлуатації.

3.4 Висновки до третього розділу

У даному розділі було розглянуто реалізацію інтерактивного веб-інтерфейсу, створення серверної частини для обробки запитів і взаємодії з базою даних, а також процес інтеграції між клієнтською та серверною частинами додатку. Завдяки використанню сучасних технологій, таких як HTML, CSS і JavaScript для фронтенду, а також C++ і SQLite для бекенду, вдалося створити функціональний і зручний веб-додаток для автоматизації процесу бронювання футбольного поля.

Розробка інтерактивного календаря стала ключовим елементом системи, який дозволяє користувачам легко переглядати доступні дати, вибирати час і здійснювати бронювання. Використання JavaScript для динамічного оновлення даних забезпечує швидкий відгук системи, що покращує взаємодію з користувачем. Серверна частина на C++ продемонструвала свою ефективність у обробці запитів і забезпеченні взаємодії з базою даних SQLite. Реалізовані маршрути API дозволяють виконувати всі необхідні операції, включаючи отримання даних, створення нових бронювань і керування існуючими запитами. Інтеграція з базою даних забезпечує збереження та обробку інформації про бронювання, користувачів і доступність ресурсів.

Таким чином, у результаті роботи над розділом вдалося створити сучасний веб-додаток, який відповідає вимогам зручності, функціональності та надійності. Розроблений додаток не лише автоматизує процес бронювання, а й забезпечує гнучкість для подальшого розширення функціональності.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи всі завдання, визначені на початку, були успішно вирішені. Перш за все, проведено аналіз вимог до серверного додатка для бронювання футбольного поля, що дозволило чітко визначити функціональні та нефункціональні потреби системи. Виконаний огляд сучасних технологій розробки серверних додатків дозволив обрати оптимальні інструменти, зокрема мову програмування C++, фреймворк Crow для створення RESTful API та базу даних SQLite для зберігання інформації.

У рамках роботи здійснено проєктування архітектури серверного додатка, що охоплює багатопоточну обробку запитів, маршрутизацію API та взаємодію з базою даних. Окрім цього, реалізовано клієнтський веб-інтерфейс із використанням HTML, CSS і JavaScript, зокрема інтерактивний календар для відображення доступних слотів для бронювання.

Результатом роботи стала інтегрована система, яка забезпечує автоматизацію процесів бронювання, швидкий доступ до інформації для користувачів та ефективну взаємодію між адміністрацією та клієнтами. Завдяки адаптивному дизайну веб-інтерфейсу система є зручною для використання на різних пристроях, що значно підвищує її доступність.

Усі отримані результати мають практичне значення та можуть бути використані для подальшого вдосконалення системи, включно з інтеграцією нових функцій і забезпеченням більшої масштабованості.

Розробка веб-додатку для бронювання футбольного поля є важливим кроком у професійному становленні у сфері інформаційних технологій. Проєкт демонструє здатність застосовувати сучасні інструменти програмування, такі як C++ та SQLite, для створення високоефективного і надійного програмного забезпечення.

Цей веб-додаток слугує прикладом того, як сучасні технології можуть бути інтегровані для вирішення реальних завдань, автоматизуючи процеси та

підвищуючи зручність взаємодії з користувачами. Основна мета – створити інтуїтивно зрозумілий і функціональний додаток, що відповідає сучасним вимогам, – була досягнута.

Кваліфікаційна робота відображає не лише технічні знання, але й здатність розробляти практичні рішення, що задовольняють потреби користувачів. Створення цього додатку стало можливістю продемонструвати навички у веб-програмуванні, роботі з базами даних та інтеграції фронтенду і бекенду.

Проєкт також слугує основою для подальшого розвитку, відкриваючи можливості вдосконалення функціональності та розширення можливостей додатка. Набуті знання і навички забезпечують здатність до вирішення складних завдань у сфері ІТ, впровадження інноваційних підходів та подальшого професійного зростання.

Висновки цього проєкту підтверджують здатність до системного підходу, творчого вирішення задач та ефективного використання сучасних технологій для створення веб-додатків, що є важливими перевагами у професійній діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lippman S. B., Lajoie J., Moo B. E. C++ Primer. 2012. 976 p.
2. C++ Documentation. URL: https://devdocs.io/cpp/.
3. Williams A. C++ Concurrency in Action: Practical Multithreading.
4. Alexandrescu A. Modern C++ Design: Generic Programming and Design Patterns Applied. 2001. 285 p.
5. Meyers S. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. 2014. 334 p.
6. Stroustrup B. Programming: Principles and Practice Using C++. 2014. 1312 p.
7. Nystrom R. Game Programming Patterns. 2014. 354 p.
8. SQLite Documentation. URL: https://www.sqlite.org/docs.html.
9. Crow Framework Documentation. URL: https://crowcpp.org.
10. RESTful API Design Guidelines. URL: https://restfulapi.net.
11. Johnson B. Professional Visual Studio 2019. 2020. 374 p.
12. Chowdhury K. Mastering Visual Studio 2019. 2019. 374 p.
13. Martin J. Visual Studio Cookbook. 2016. 368 p.
14. Johnson B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers. 2019. 192 p.
15. CMake Documentation. URL: https://cmake.org/documentation.
16. Fisher G. The Essential Guide to SQLite for Developers. 2018. 250 p.
17. Postman API Testing Documentation. URL: https://learning.postman.com/docs/getting-started/introduction/.
18. Best Practices for REST API Development. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api.
19. Bruce Johnson. Visual Studio Code and Development Tools. 2021. 400 p.
20. Vcpkg Documentation. URL: https://vcpkg.io/en/getting-started.html.
21. Nishanov G. Effective Coroutine Programming with C++. 2020. 312 p.

22. Josuttis N. The C++ Standard Library: A Tutorial and Reference. 2019. 1128 p.

23. Sutter H. Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions. 2000. 240 p.

24. Hinnant H., Garland R., Wong A. Concurrency with Modern C++. 2022. 480 p.

25. Boost Library Documentation. URL: https://www.boost.org/doc/libs/.

26. SQLite Optimization Tips. URL: https://www.sqlite.org/optoverview.html.

27. OpenSSL Documentation. URL: https://www.openssl.org/docs/.

28. Valgrind Documentation. URL: https://valgrind.org/docs/.

29. Crow Framework GitHub Repository. URL: https://github.com/CrowCpp/Crow.

30. Prometheus Monitoring Documentation. URL: https://prometheus.io/docs/.

ДОДАТОК

main.cpp

```
#include "crow.h"
#include "database.cpp" // Файл для роботи з базою даних
#include <fstream>
#include <sstream>

std::string read_file(const std::string& filepath) {
    std::ifstream file(filepath);
    if (!file.is_open()) {
        return "<h1>404 - File Not Found</h1>";
    }
    std::stringstream buffer;
    buffer << file.rdbuf();
    return buffer.str();
}

int main() {

    initDatabase();

    crow::SimpleApp app;

    // Роут для головної сторінки
    CROW_ROUTE(app, "/")
    ([]() {
        std::string html_content = read_file("C:/BookingWebApp/public/index.html");
        return crow::response(html_content);
    });

    CROW_ROUTE(app, "/api/slots")([]() {
        auto slots = getAvailableSlots();
        crow::json::wvalue res;
        res["slots"] = slots;
        return crow::response(res);
    });

    // Роут для створення бронювання
    CROW_ROUTE(app, "/api/book").methods("POST"_method)([&db](const crow::request&
req) {
    auto body = crow::json::load(req.body);
    std::string date = body["date"].s();
    std::string time_slot = body["time_slot"].s();
    int user_id = body["user_id"].i();

    std::string query = "INSERT INTO bookings (user_id, date, time_slot, status) VALUES (?, ?,
?, ‘booked’)";
    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, query.c_str(), -1, &stmt, nullptr);
    sqlite3_bind_int(stmt, 1, user_id);
```

```
    sqlite3_bind_text(stmt, 2, date.c_str(), -1, SQLITE_STATIC);
    sqlite3_bind_text(stmt, 3, time_slot.c_str(), -1, SQLITE_STATIC);

    if (sqlite3_step(stmt) == SQLITE_DONE) {
        sqlite3_finalize(stmt);
        return crow::response(200, "Booking successful");
    } else {
        sqlite3_finalize(stmt);
        return crow::response(500, "Failed to book");
    }
});

    app.port(8080).multithreaded().run();
}

database.cpp
#include <sqlite3.h>
#include <vector>
#include <string>

sqlite3* db;

void initDatabase() {
    sqlite3_open("booking.db", &db);
    std::string createTable = R"(
        CREATE TABLE IF NOT EXISTS bookings (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            time TEXT
        );
    )";
    sqlite3_exec(db, createTable.c_str(), nullptr, nullptr, nullptr);
}

void selectBAase()
{
    sqlite3* db;
    sqlite3_open("database.db", &db);

    std::string query = "SELECT date FROM bookings WHERE status = ‘booked’";
    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, query.c_str(), -1, &stmt, nullptr);

    while (sqlite3_step(stmt) == SQLITE_ROW) {
    const char* date = reinterpret_cast<const char*>(sqlite3_column_text(stmt, 0));
    std::cout << "Booked date: " << date << std::endl;
}

bool isConflict(int startTime, int endTime, const std::vector<std::pair<int, int>>& bookings, int buffer)
{
for (const auto& booking : bookings)
{
int existingStart = booking.first;
```

```
int existingEnd = booking.second;
if (!(endTime + buffer <= existingStart || startTime >= existingEnd + buffer))
{
return true;
}
}
return false;
}

sqlite3_finalize(stmt);
sqlite3_close(db);
}

bool createBooking(const std::string& name, const std::string& time) {
    std::string sql = "INSERT INTO bookings (name, time) VALUES (?, ?);";
    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, sql.c_str(), -1, &stmt, nullptr);
    sqlite3_bind_text(stmt, 1, name.c_str(), -1, SQLITE_STATIC);
    sqlite3_bind_text(stmt, 2, time.c_str(), -1, SQLITE_STATIC);
    bool success = (sqlite3_step(stmt) == SQLITE_DONE);
    sqlite3_finalize(stmt);
    return success;
}

std::vector<std::string> getAvailableSlots() {
    // Повертає слоти, які не зайняті
    std::vector<std::string> slots = {"10:00", "10:30", "11:00"}; // Для прикладу
    return slots;
}

index.html
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Оренда футбольного поля</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #556b2f;
            color: #fff;
        }

        h1, h2, h3 {
            margin: 0;
            padding: 0;
        }

        p {
            margin: 10px 0;
```

```
}
  .hero-section {
    background-image: url('C:/Users/silve/Desktop/1.jpg');
    background-size: cover;
    background-position: center;
    text-align: center;
    padding: 100px 20px;
}

.hero-content h1 {
    font-size: 3rem;
    font-weight: bold;
    margin-bottom: 10px;
    text-shadow: 1px 1px 2px black;

}

.hero-content p {
    font-size: 1.5rem;
    margin-bottom: 20px;
    text-shadow: 1px 1px 2px black;
}

.hero-content .btn {
    padding: 15px 30px;
    font-size: 1rem;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

.btn.primary {
    background-color: #ff9800;
    color: #000;
}

.btn.primary:hover {
    background-color: #e68900;
}

.btn.secondary {
    background-color: #4caf50;
    color: #fff;
}

.btn.secondary:hover {
    background-color: #3e8e41;
}

/* Секція інформації */
.info-section {
    padding: 50px 20px;
```

```
    background-color: #061804;
}

.info-content {
    text-align: center;
    margin-bottom: 30px;
}

.info-content h2 {
    font-size: 2rem;
    margin-bottom: 10px;
}

.info-content .info-buttons {
    display: flex;
    gap: 15px;
    justify-content: center;
    margin-top: 20px;
}

/* Календар */
.calendar-section {
    margin-top: 30px;
    text-align: center;
}

.calendar {
    display: grid;
    grid-template-columns: repeat(7, 1fr);
    gap: 10px;
    justify-content: center;
    margin-top: 20px;
}

.day {
    padding: 15px;
    text-align: center;
    background: #4caf50;
    color: #fff;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s ease;
}

.day.booked {
    background: #ff6b6b;
    cursor: not-allowed;
}

.day:hover:not(.booked) {
    background: #3e8e41;
}
```

```
/* Додаткові секції */
section {
    margin: 30px auto;
    padding: 20px;
    max-width: 800px;
    background: #061804;
    border-radius: 8px;
    color: #fff;
}

/* Таблиця цін */
.pricing-table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    text-align: center;
}

.pricing-table th,
.pricing-table td {
    padding: 10px;
    border: 1px solid #444;
}

.pricing-table th {
    background: #333;
    color: #fff;
}

/* Форма бронювання */
form {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 15px;
}

form .form-group {
    display: flex;
    flex-direction: column;
}

form .form-group label {
    margin-bottom: 5px;
}

form .form-group input,
form .form-group select,
form .form-group textarea {
    padding: 10px;
    border: 1px solid #555;
    border-radius: 5px;
```

```
            background: #222;
            color: #fff;
        }

        form button {
            grid-column: span 2;
            padding: 10px;
            background: #ff9800;
            border: none;
            border-radius: 5px;
            color: #000;
            cursor: pointer;
        }

        form button:hover {
            background: #e68900;
        }

        /* Контактна інформація */
        .contact-section img {
            width: 100%;
            margin-top: 15px;
            border-radius: 5px;
        }
    </style>
</head>
<body>
    <header class="hero-section">
        <div class="hero-content">
            <h1>ОРЕНДА ФУТБОЛЬНОГО ПОЛЯ</h1>
            <p>Для регулярних та одноразових тренувань</p>
            <button class="btn primary">Забронювати</button>
        </div>
    </header>

    <main>
        <section class="info-section">
            <div class="info-content">
                <h2>ОРЕНДА ФУТБОЛЬНОГО ПОЛЯ В ДНІПРІ</h2>
                <p>Для любителів футболу та активного відпочинку.</p>
                <div class="info-buttons">
                    <button class="btn primary">Забронювати</button>
                    <button class="btn secondary">Зворотній дзвінок</button>
                </div>
            </div>
            <div class="calendar-section">
                <h3>Виберіть дату для бронювання</h3>
                <div class="calendar" id="calendar">
                    <!-- Дні будуть динамічно додаватися за допомогою JavaScript -->
                </div>
            </div>
        </section>
```

```
<!-- Секція Ціна оренди -->
<section class="pricing-section">
  <h2>ЦІНА ОРЕНДИ</h2>
  <p>Вказана вартість оренди залежить від часу доби та розташування.</p>
  <table class="pricing-table">
    <thead>
      <tr>
        <th>Час</th>
        <th>Будні</th>
        <th>Вихідні</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>08:00-12:00</td>
        <td>400 грн</td>
        <td>450 грн</td>
      </tr>
      <tr>
        <td>12:00-18:00</td>
        <td>450 грн</td>
        <td>500 грн</td>
      </tr>
      <tr>
        <td>18:00-22:00</td>
        <td>500 грн</td>
        <td>550 грн</td>
      </tr>
    </tbody>
  </table>
</section>

<!-- Секція Надішліть запит -->
<section class="booking-form-section">
  <h2>НАДІШЛІТЬ ЗАПИТ НА БРОНЮВАННЯ</h2>
  <form id="booking-form">
    <div class="form-group">
      <label for="name">Прізвище та ім’я</label>
      <input type="text" id="name" name="name" required>
    </div>
    <div class="form-group">
      <label for="email">Ел. пошта</label>
      <input type="email" id="email" name="email" required>
    </div>
    <div class="form-group">
      <label for="phone">Номер телефону</label>
      <input type="text" id="phone" name="phone" required>
    </div>
    <div class="form-group">
      <label for="date">Дата</label>
      <input type="date" id="date" name="date" required>
```

```
            </div>
            <div class="form-group">
              <label for="time">Час</label>
              <select id="time" name="time" required>
                <option value="08:00-12:00">08:00-12:00</option>
                <option value="12:00-18:00">12:00-18:00</option>
                <option value="18:00-22:00">18:00-22:00</option>
              </select>
            </div>
            <div class="form-group">
              <label for="message">Коментар</label>
              <textarea id="message" name="message" rows="4"></textarea>
            </div>
            <button type="submit" class="btn primary">Надіслати запит</button>
          </form>
        </section>

        <!-- Секція Контакти -->
        <section class="contact-section">
          <h2>ДЕ НАС ЗНАЙДЕТЕ?</h2>
          <p>м. Дніпро<br>вул. Гоголя 23А<br>Навчально-тренувальний комплекс</p>
          <p>Не можете знайти?<br>Телефонуйте: <a href="tel:+380992721532">+380 99 272
15 32</a></p>
          <img src="google-maps.jpg" alt="Google Maps">
        </section>
      </main>

      <script>
        const calendar = document.getElementById('calendar');

        // Імітація даних про стан днів (1 = заброньовано, 0 = доступно)
        const days = Array(30).fill(0).map((_, i) => (i % 3 === 0 ? 1 : 0)); // Кожен 3-й день
заброньовано

        // Створення днів у календарі
        days.forEach((status, index) => {
          const day = document.createElement('div');
          day.classList.add('day');
          day.textContent = index + 1;

          if (status === 1) {
            day.classList.add('booked');
            day.textContent += ' (Заброньовано)';
          } else {
            day.addEventListener('click', () => {
              alert(`Ви вибрали день ${index + 1} для бронювання!`);
            });
          }

          calendar.appendChild(day);
        });
      </script>
```

```
</body>
</html>
```

style.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #121212;
    color: #fff;
}

h1, h2, h3 {
    margin: 0;
    padding: 0;
}

p {
    margin: 10px 0;
}

/* Секція героя */
.hero-section {
    background-image: url('data:image/jpeg; =');
    background-size: cover;
    background-position: center;
    text-align: center;
    padding: 100px 20px;
}

.hero-content h1 {
    font-size: 3rem;
    font-weight: bold;
    margin-bottom: 10px;
}

.hero-content p {
    font-size: 1.5rem;
    margin-bottom: 20px;
}

.hero-content .btn {
    padding: 15px 30px;
    font-size: 1rem;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

.btn.primary {
    background-color: #ff9800;
    color: #000;
```

```
}

.btn.primary:hover {
    background-color: #e68900;
}

.btn.secondary {
    background-color: #4caf50;
    color: #fff;
}

.btn.secondary:hover {
    background-color: #3e8e41;
}

/* Секція інформації */
.info-section {
    padding: 50px 20px;
    background-color: #1e1e1e;
}

.info-content {
    text-align: center;
    margin-bottom: 30px;
}

.info-content h2 {
    font-size: 2rem;
    margin-bottom: 10px;
}

.info-content .info-buttons {
    display: flex;
    gap: 15px;
    justify-content: center;
    margin-top: 20px;
}

/* Календар */
.calendar-section {
    margin-top: 30px;
    text-align: center;
}

.calendar {
    display: grid;
    grid-template-columns: repeat(7, 1fr);
    gap: 10px;
    justify-content: center;
    margin-top: 20px;
}
```

```
.day {
   padding: 15px;
   text-align: center;
   background: #4caf50;
   color: #fff;
   border-radius: 5px;
   cursor: pointer;
   transition: background 0.3s ease;
}

.day.booked {
   background: #ff6b6b;
   cursor: not-allowed;
}

.day:hover:not(.booked) {
   background: #3e8e41;
}
```

app.js

```
const calendar = document.getElementById('calendar');

// Імітація даних про стан днів (1 = заброньовано, 0 = доступно)
const days = Array(30).fill(0).map((_, i) => (i % 3 === 0 ? 1 : 0)); // Кожен 3-й день
заброньовано

// Створення днів у календарі
days.forEach((status, index) => {
   const day = document.createElement('div');
   day.classList.add('day');
   day.textContent = index + 1;

   if (status === 1) {
      day.classList.add('booked');
      day.textContent += ' (Заброньовано)';
   } else {
      day.addEventListener('click', () => {
         alert(`Ви вибрали день ${index + 1} для бронювання!`);
      });
   }

   calendar.appendChild(day);
});

fetch('/api/available-dates')
   .then(response => response.json())
   .then(data => {
      data.forEach((day, index) => {
         const dayElement = document.createElement('div');
         dayElement.classList.add('day');
         dayElement.textContent = index + 1;
         if (day.available) {
            dayElement.classList.add('available');
```

```
            dayElement.addEventListener('click', () => selectDate(index + 1));
        } else {
            dayElement.classList.add('booked');
        }
        document.querySelector('.calendar').appendChild(dayElement);
    });
});

document.querySelector('#bookingForm').addEventListener('submit', (e) => {
    e.preventDefault();
    const formData = new FormData(e.target);
    fetch('/api/book', {
        method: 'POST',
        body: JSON.stringify({
            date: formData.get('date'),
            time: formData.get('time'),
            user: formData.get('user')
        }),
        headers: {
            'Content-Type': 'application/json'
        }
    })
    .then(response => response.json())
    .then(data => alert(data.message));
});
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(BookingWebApp)

# Вкажи шлях до vcpkg
set(CMAKE_TOOLCHAIN_FILE
"C:/BookingWebApp/vcpkg/scripts/buildsystems/vcpkg.cmake")

# Знайди бібліотеку Crow
find_package(Crow CONFIG REQUIRED)

# Додай виконуваний файл
add_executable(server src/main.cpp)
# Підключи бібліотеку Crow
target_link_libraries(server PRIVATE Crow::Crow)
# Знайти бібліотеку SQLite3
find_package(SQLite3 REQUIRED)

# Додати SQLite3 до ваших таргетів
target_link_libraries(server PRIVATE SQLite::SQLite3)
```