МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Острозька академія»

Навчально-науковий інститут інформаційних технологій та бізнесу

Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістра

на тему: « **ЗАСТОСУВАННЯ C++ ДЛЯ РОЗРОБКИ ВЕБ-ДОДАТКІВ**»

Виконав: студент 2 курсу, групи МУП-21

другого (магістерського) рівня вищої освіти

спеціальності 122 Комп'ютерні науки

освітньо-професійної програми «Управління проєктами»

Козак Володимир Володимирович

Керівник: викладач, фахівець-практик,

Ляховчук Сергій Васильович

Рецензент: кандидат технічних наук, доцент,

доцент кафедри прикладної математики та кібербезпеки

Донецького національного університету імені Василя Стуса,

Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних

_____ (проф., д.е.н. Кривицька О.Р.)

Протокол №5 від « 04 » грудня 2025

Острог, 2025

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня магістра

Тема: *Застосування C++ для розробки веб-додатків*

Автор: Козак Володимир Володимирович

Науковий керівник: *Ляховчук Сергій Васильович, викладач, фахівець-практик*

Захищена «……»… 2025 року.

Пояснювальна записка до кваліфікаційної роботи: *с.82, рис.9 , джерела.17*

Ключові слова: *веб-додатки, C++, Crow, Angular, REST API, продуктивність, масштабованість, безпека.*

Короткий зміст праці:

Магістерська робота присвячена дослідженню та застосуванню мови програмування C++ для розробки веб-додатків. У роботі проаналізовано сучасні підходи та технології веб-розробки з акцентом на використання високопродуктивних рішень на основі C++.

Розроблено архітектуру веб-додатку, реалізовано серверну частину на базі C++ із застосуванням фреймворку Crow і клієнтську частину з використанням Angular. Забезпечено інтеграцію компонентів через REST API із використанням JSON.

Практична частина включає розробку функціонального прототипу, тестування його продуктивності та безпеки.

Результати дослідження підтверджують доцільність використання C++ у веб-розробці для створення високопродуктивних, масштабованих і надійних систем, що відповідають сучасним вимогам. Робота є внеском у розвиток ефективних методів створення серверних веб-рішень із оптимізованим використанням ресурсів.

ANNOTATION
of qualification paper
for master's degree

Theme: *Application of C++ for Web Application Development*

Author: *Kozak Volodymyr Volodymyrovych*

Supervisor: *Liakhovchuk Serhii Vasylovych, lecturer, specialist-practitioner*

Defended *"……" …… 2025.*

Explanatory note to the qualification work: *82 p., 9 pic., 17 sources.*

Keywords: *web applications, C++, Crow, Angular, REST API, performance, scalability, security.*

Summary of the work:

This master's qualification paper is devoted to the research and application of the C++ programming language for web application development. The paper analyzes modern approaches and technologies in web development with an emphasis on the use of high-performance solutions based on C++.

The architecture of a web application was developed; the server-side part was implemented in C++ using the Crow framework, and the client-side part was created using Angular. Integration of system components was ensured through a REST API using the JSON data format.

The practical part includes the development of a functional prototype and testing its performance and security.

The research results confirm the feasibility of using C++ in web development for creating high-performance, scalable, and reliable systems that meet modern requirements. The work contributes to the development of efficient methods for building server-side web solutions with optimized resource utilization.

ЗМІСТ

ВСТУП

Сучасна web-розробка охоплює широкий набір технологій, що дають змогу створювати швидкі, масштабовані та безпечні додатки. Для серверної частини web-додатків традиційно застосовуються мови програмування, оптимізовані для швидкої та зручної розробки, зокрема JavaScript (Node.js), Python, PHP, Java та C#. Проте зростаючі вимоги до продуктивності та ефективного використання апаратних ресурсів сприяють зростанню інтересу до використання C++ у сфері web-програмування.

C++ є однією з найпотужніших мов, що надає розробникам низькорівневий контроль над пам’яттю та високу продуктивність. Завдяки цьому вона стає ефективним вибором для розробки високонавантажених серверних рішень, роботи з мережевими протоколами та створення інтерактивних web-додатків із використанням WebAssembly.

Однак, застосування C++ у web-розробці залишається відносно рідкісним через складність мови, обмежену кількість спеціалізованих фреймворків і необхідність ручного керування ресурсами.

Актуальність дослідження зумовлена стрімким розвитком хмарних обчислень та високонавантажених web-систем, що вимагають максимально ефективного використання обчислювальних ресурсів. У цьому контексті C++ виступає як потужний інструмент для створення продуктивних серверних додатків, здатних конкурувати з традиційними технологіями. Проте широке застосування C++ у web-розробці залишається обмеженим, що викликає необхідність дослідження його переваг, недоліків та доцільності використання у цій сфері.

Метою дослідження є аналіз можливостей застосування C++ для розробки web-додатків, оцінка його переваг і недоліків у порівнянні з іншими технологіями, а також практична реалізація web-додатку на базі C++.

Для досягнення поставленої мети передбачено виконання таких завдань:

- дослідити сучасні підходи до web-розробки та визначити роль C++ у цій сфері;
- порівняти C++ з іншими мовами web-програмування за критеріями продуктивності, безпеки, масштабованості та зручності розробки;
- проаналізувати наявні інструменти та фреймворки для web-розробки на C++;
- розробити та протестувати web-додаток із використанням C++ і оцінити його ефективність;
- визначити перспективи використання C++ для web-додатків і сформулювати рекомендації щодо його застосування.

Об’єктом дослідження є технології та інструменти, що використовуються для створення web-додатків.

Предметом дослідження виступає застосування мови програмування C++ у web-розробці, її порівняння з іншими мовами та практична реалізація web-додатку.

РОЗДІЛ 1

АНАЛІЗ WEB-РОЗРОБКИ ТА ЗАСТОСУВАННЯ C++

Сучасна web-розробка охоплює широкий спектр технологій і підходів, що дозволяють створювати динамічні, інтерактивні та масштабовані веб-додатки.

Вибір мови програмування для розробки веб-додатків є одним із ключових факторів, що визначає продуктивність, безпеку, підтримку та можливості інтеграції в рамках проекту.

У цьому розділі ми розглянемо основні підходи до розробки web-додатків, аналізуючи популярні мови програмування, такі як JavaScript, Python, PHP, Java і C#, а також детально зупинимося на застосуванні мови C++ у web-розробці.

1.1 Основні підходи до розробки web-додатків

Web-додаток – це прикладна програма, створена за архітектурою «клієнт-сервер», де в ролі клієнта використовується web-браузер, а обробка даних відбувається на web-сервері (рис. 1.1.1). Взаємодія між клієнтом і сервером здійснюється за допомогою протоколу HTTP.

Загальна схема роботи web-додатку

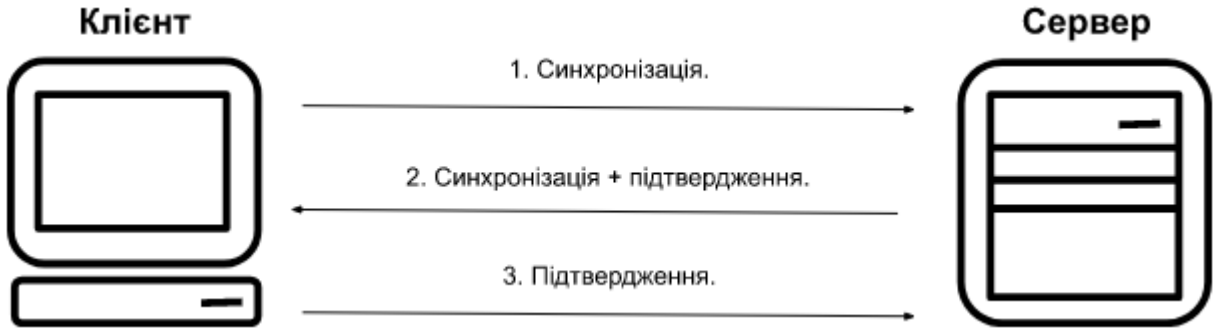

Рис. 1.1.1 Загальна схема взаємодії користувача з web-додатком.

Користувач через браузер надсилає HTTP-запит до певної URL-адреси, яка відповідає ресурсам на web-сервері. Якщо ця адреса веде до динамічного ресурсу (наприклад, web-додатку), сервер виконує відповідну програму, що генерує HTML-сторінку і передає її назад клієнту для відображення в браузері.

Основний функціонал web-додатку реалізується на стороні сервера.

Зазвичай web-додаток не є самостійною програмою, а працює у середовищі виконання, яке може бути сервером або контейнером додатків. Таке середовище виконує такі завдання:

- отримує від web-сервера всі дані, пов’язані з HTTP-запитом;
- визначає, яку програму слід запустити для обробки запиту;
- створює необхідні об'єкти для коректної роботи додатку;
- завантажує відповідний додаток і передає йому керування;
- web-додаток обробляє запит, формує HTML-сторінку, зберігає її у відповідному об'єкті і завершує виконання;
- web-сервер створює HTTP-відповідь, включає в неї згенеровану сторінку та відправляє клієнту, який надіслав запит. [1]

Розробка веб-додатків охоплює широкий спектр технологій та методів, що дозволяють створювати інтерактивні, динамічні та масштабовані системи. Для цього існують різні підходи, які можуть залежати від рівня абстракції, інструментів, що використовуються, а також специфіки конкретного проекту.

Основні підходи до розробки веб-додатків можна поділити на три великі категорії:

1. **Підходи, що базуються на програмуванні або скриптах**: Це зовнішні програми чи скрипти, а також розширення для веб-серверів. До цієї категорії відносяться методи, які дозволяють розширювати функціональність веб-додатків за допомогою програмування на стороні сервера або клієнта.
2. **Підходи, що ґрунтуються на використанні шаблонів веб-сторінок**: Цей метод включає вставки скриптів і спеціальних серверних тегів в шаблони сторінок. Веб-сторінки автоматично генеруються на сервері на основі

заздалегідь створених шаблонів, що значно спрощує створення динамічного контенту.

3. **Об'єктні середовища (каркаси, фреймворки)**: Це програмні каркаси та фреймворки, які надають набір інструментів і бібліотек для спрощення розробки веб-додатків. Фреймворки дозволяють автоматизувати багато аспектів розробки, таких як обробка запитів, підключення до бази даних, а також управління маршрутизацією.

Хоча між цими категоріями можуть бути певні перетини, а також існують різні точки зору на те, до якої категорії належить конкретна технологія, більшість широко відомих методів розробки веб-додатків можна чітко віднести до однієї з цих категорій. [2]

1.2 Популярні мови програмування для web

У сучасній web-розробці використовується широкий спектр мов програмування, кожна з яких має свої особливості, переваги та області застосування.

JavaScript: Лідер у фронтенд-розробці

JavaScript був створений для роботи зі скриптами на стороні клієнта, але завдяки розвитку Node.js став також популярним для серверної частини. Ця мова є невід'ємною частиною веб-розробки, оскільки підтримується практично всіма веб-браузерами без додаткових плагінів.

Переваги JavaScript:

- **Універсальність:** Можна використовувати як на клієнтській, так і на серверній стороні, що дозволяє програмістам застосовувати одну мову на всіх етапах розробки.
- **Велика спільнота:** Існує безліч фреймворків і бібліотек, таких як React, Angular та Vue.js, які полегшують створення інтерактивних веб-додатків.

- **Масштабованість:** Node.js забезпечує ефективну роботу з асинхронним ввід/вивід, що дозволяє створювати масштабовані додатки.

Python: Мова для простоти та ефективності

Python здобув популярність завдяки простоті та читабельності коду. У веб-розробці Python широко використовують завдяки фреймворкам, таким як Django і Flask, які спрощують створення веб-додатків.

Переваги Python:

- **Простий синтаксис:** Чистий і зрозумілий код спрощує підтримку та прискорює розробку.
- **Могутні бібліотеки:** Готові рішення для складного функціоналу без необхідності писати багато коду.
- **Висока продуктивність:** Можливість інтеграції з іншими мовами для підвищення ефективності.

Ruby та Ruby on Rails: Оптимальний вибір для стартапів

Ruby в поєднанні з фреймворком Ruby on Rails надає спрощений підхід до розробки веб-додатків, відомий як "Convention over Configuration" (Угода понад налаштування). Це робить Ruby ідеальним вибором для стартапів і швидкого прототипування.

Переваги Ruby:

- **Стандартні налаштування:** Угоди за замовчуванням допомагають швидше почати розробку.
- **Автоматизоване тестування:** Вбудовані інструменти для тестування і забезпечення якості коду.
- **Елегантний синтаксис:** Легко читається і підтримується.

C#: Потужність та гнучкість для веб-додатків

C# від Microsoft зазвичай використовують разом із фреймворком .NET для створення веб-додатків. Ця мова з широкою підтримкою бібліотек та інтеграцією з базами даних є хорошим вибором для складних веб-проектів.

Переваги C#:

- **Надійність:** Сувора типізація та об'єктно-орієнтоване програмування забезпечують високу стабільність і масштабованість.
- **Багатий набір інструментів:** .NET Framework прискорює розробку і спрощує виконання складних завдань.
- **Безпека та управління пам'яттю:** Підвищують надійність веб-додатків.

PHP: Надійний вибір для динамічних веб-сайтів

PHP — одна з найпоширеніших мов для створення динамічних веб-сайтів і додатків, завдяки простоті інтеграції з веб-серверами та базами даних.

Переваги PHP:

- **Простота:** Легко почати працювати, і велика підтримка хостинг-платформ.
- **Фреймворки:** Laravel та Symfony забезпечують готові рішення для часто зустрічаючихся завдань.
- **Міцна спільнота:** Велика кількість документації і форумів полегшує вирішення проблем.

Go: Сучасний підхід до серверної розробки

Go, розроблений Google, став популярним завдяки своїй високій продуктивності для серверної розробки, особливо при обробці конкурентних операцій і мікросервісів.

Переваги Go:

- **Простий синтаксис:** Легко вивчити і швидко розпочати роботу.
- **Конкурентність:** Вбудована підтримка багатопоточних додатків.
- **Висока ефективність:** Продуктивність робить Go ідеальним для масштабованих систем.

TypeScript: Надбудова для JavaScript з типізацією

TypeScript був створений Microsoft як розширення JavaScript, додаючи строгий контроль типів і об'єктно-орієнтовані можливості, що робить його ідеальним для великих проектів.

Переваги TypeScript:

- **Строга типізація:** Допомагає запобігти багатьом помилкам під час компіляції.
- **Інтеграція з фреймворками:** Підтримка таких фреймворків, як Angular та Vue.js.
- **Масштабованість:** Чітка структура і модульність підвищують підтримку коду.

Kotlin: Мова для веб і мобільної розробки

Kotlin, розроблений JetBrains, швидко набирає популярність завдяки своїй безпеці, простоті та сумісності з Java.

Переваги Kotlin:

- **Сумісність з JVM:** Доступ до багатьох бібліотек і фреймворків Java.
- **Типова безпека:** Підтримка нульової безпеки зменшує можливі помилки.
- **Гнучкість:** Підтримує як функціональне, так і об'єктно-орієнтоване програмування. [3]

1.3 Використання C++ у web-розробці

У динамічному світі веб-розробки різні мови програмування зайняли лідируючі позиції, кожна з яких відповідає певним потребам і вподобанням розробників. Однією з таких мов, що пройшла випробування часом і залишається ключовою в сфері розробки програмного забезпечення, є C++. Традиційно вона відома своєю потужністю, ефективністю та продуктивністю в системному програмуванні, тому її рідко розглядають як перший вибір для веб-розробки.

Однак, із розвитком технологій розширюються й можливості мов програмування. [4]

Багато раннього веб-функціоналу було написано на C++ із використанням Common Gateway Interface (CGI) — специфікації, яка дозволяє веб-серверам виконувати зовнішні програми для обробки HTTP-запитів. Одним із перших застосувань CGI було оброблення веб-форм.

Назва CGI походить з ранніх днів розвитку Інтернету, коли веб-розробники прагнули підключити наявні інформаційні системи, такі як бази даних, до веб-серверів. CGI-програма виконувалася сервером і забезпечувала «шлюз» між веб-сервером і застарілою інформаційною системою.

Для кожного вхідного HTTP-запиту веб-сервер створює окремий CGI-процес, який завершує свою роботу після обробки запиту. Створення та знищення процесів може споживати значно більше ресурсів (процесорної потужності та пам’яті), ніж безпосередня генерація відповіді. Це особливо проблематично, якщо CGI-програма виконується у віртуальному середовищі. При великій кількості HTTP-запитів навантаження може швидко перевантажити сервер.

Модель «один процес на кожен запит» (рис 1.3.1) робить CGI-програми простими у реалізації, але водночас обмежує їхню ефективність і масштабованість.

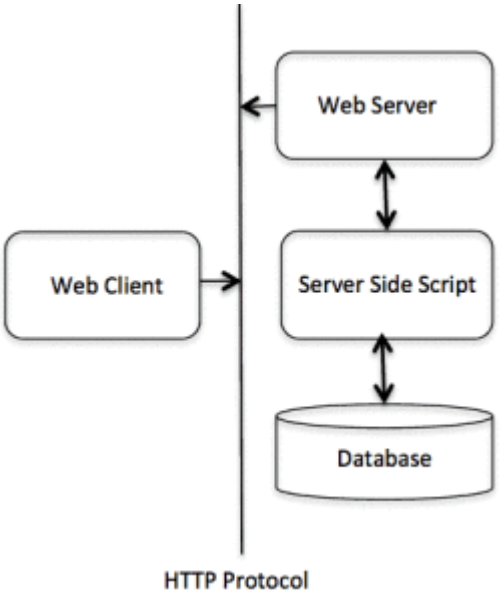

Рис 1.3.1 Архітектура CGI

При високих навантаженнях витрати операційної системи на створення та завершення процесів стають суттєвими. Крім того, така модель ускладнює повторне використання ресурсів, наприклад, збереження з’єднань із базою даних чи використання кешування в оперативній пам’яті.

Щоб вирішити проблему масштабованості CGI, компанія Open Market розробила FastCGI — варіацію Common Gateway Interface — і вперше представила її у своєму веб-серверному продукті в середині 1990-х років.

FastCGI дозволяє зовнішнім довготривалим процесам обробляти кілька запитів, не створюючи новий процес для кожного з них. Кожен процес застосунку прослуховує сокет, а веб-сервер передає HTTP-запит через протокол FastCGI лише для динамічного контенту, тоді як статичний контент зазвичай обробляється безпосередньо веб-сервером.

Кожен окремий FastCGI-процес може обробляти багато запитів протягом свого життєвого циклу, уникаючи витрат на створення та завершення процесу для кожного запиту.

Сучасною альтернативою використанню CGI або FastCGI є мікросервіси або веб-сервіси з власними HTTP-серверними компонентами, які працюють за реверс-проксі-сервером, таким як Nginx, що оптимізований для обробки статичного контенту, балансування навантаження та забезпечення безпеки. [5]

Сучасні фреймворки та бібліотеки C++, разом із розвитком веб-технологій, відкрили можливості для використання переваг C++ у веб-додатках.

Продуктивність:

C++ відомий своєю високою швидкістю роботи та ефективністю. Оскільки це компільована мова, її код перекладається у машинний ще до виконання, що забезпечує швидку роботу програм і раціональне використання ресурсів системи. Завдяки цьому C++ широко застосовується в продуктивно-критичних додатках, таких як ігрові рушії, де кожна частка секунди має значення.

Портативність:

C++ є високопортативною мовою і активно використовується для створення системних застосунків, що становлять значну частину операційних систем Windows, Linux та Unix.

Контроль і передбачуваність:

C++ надає розробникам детальний контроль над управлінням пам’яттю та ресурсами системи, що дозволяє ефективно оптимізувати роботу додатків. На відміну від мов із вбудованим збором сміття, C++ дозволяє вручну керувати виділенням і звільненням пам’яті, мінімізуючи втрати продуктивності та уникаючи проблемних "вузьких місць".

Інтеграція з наявними системами:

Багато організацій мають великі кодові бази, написані на C або C++, особливо у сферах телекомунікацій, фінансів і гейм-індустрії. Використовуючи C++ у веб-розробці, компанії можуть легко інтегрувати існуючі системи з сучасними веб-інтерфейсами, розширюючи функціональність та продовжуючи життєвий цикл свого ПЗ.

Доступ до низькорівневих API:

Часто веб-додатки потребують взаємодії з низькорівневими API для роботи з мережею, файловою системою або апаратним прискоренням. C++ забезпечує прямий доступ до таких API через бібліотеки, наприклад, Boost і ASIO, що дозволяє створювати високопродуктивні веб-додатки з мінімальними накладними витратами на абстракцію.

Хоча переваги використання C++ у веб-розробці є беззаперечними, важливо враховувати й пов’язані з цим труднощі та особливості:

Складність навчання:

C++ є складнішою мовою, ніж C, оскільки вимагає глибшого розуміння таких концепцій, як об'єктно-орієнтоване програмування та шаблони. Це може ускладнити процес навчання та використання, особливо для новачків.

Тривалість розробки:

Написання та налагодження коду на C++ зазвичай займає більше часу, ніж у високорівневих мовах, оскільки потребує ручного керування пам’яттю та явного оголошення типів. Хоча сучасні IDE та інструменти розробки спрощують цей процес, програмісти мають бути готові витратити додатковий час на написання та тестування коду.

Ризики безпеки:

Через низькорівневий характер C++ розробники можуть зіткнутися з проблемами безпеки, такими як переповнення буфера, витоки пам’яті та помилки роботи з покажчиками. Щоб уникнути цих загроз, необхідно дотримуватися найкращих практик безпечного кодування: перевіряти межі масивів, валідувати вхідні дані та коректно керувати пам’яттю.

Обмежена підтримка веб-фреймворків:

На відміну від мов, таких як JavaScript або Python, які мають розвинену екосистему веб-фреймворків, C++ пропонує менший вибір спеціалізованих рішень. Хоча такі фреймворки, як Wt, CppCMS і Crow, надають можливості для створення веб-додатків на C++, розробники можуть відчувати нестачу інструментів та підтримки спільноти у порівнянні з іншими мовами.

Попри зазначені виклики, C++ залишається ефективним вибором у таких сценаріях веб-розробки:

- **Системи реального часу:** Додатки, що вимагають швидкої обробки даних, наприклад, онлайн-ігрові платформи, сервіси потокового мультимедіа та фінансові торгові системи, можуть значно виграти від продуктивності C++.
- **Високопродуктивні обчислення:** Веб-додатки, що виконують складні розрахунки, такі як наукові симуляції, аналіз великих даних і машинне навчання, можуть використовувати можливості C++ для паралельної обробки та досягнення максимальної швидкості виконання.
- **Вбудована веб-розробка:** Із зростанням популярності пристроїв Інтернету речей (IoT) та вбудованих систем збільшується потреба у веб-інтерфейсах для взаємодії з ними. Завдяки своїй ефективності та портативності C++ добре підходить для розробки веб-додатків, орієнтованих на пристрої з обмеженими ресурсами. [6]

Хоча C++ не є першою мовою, яка спадає на думку для веб-розробки, його неперевершена продуктивність, портативність і контроль роблять його привабливим вибором для певних сценаріїв. Використовуючи переваги C++ разом

із сучасними веб-технологіями, розробники можуть створювати високопродуктивні веб-додатки, що розширюють межі можливого. Незалежно від того, чи йдеться про ігрові платформи в реальному часі, системи високочастотного трейдингу чи вбудовані IoT-пристрої, C++ продовжує доводити свою актуальність у постійно змінюваному світі веб-розробки.

1.4 Порівняння C++ з іншими мовами у web-розробці

У світі веб-розробки вибір мови програмування залежить від вимог проекту, включаючи продуктивність, масштабованість, зручність розробки та доступність бібліотек і фреймворків. Хоча C++ традиційно не асоціюється з веб-розробкою через свою складність і низькорівневу природу, його беззаперечні переваги в продуктивності, контролі над системними ресурсами та портативності роблять його конкурентоспроможним варіантом для певних задач.

У цьому порівнянні розглянемо, як C++ співвідноситься з іншими популярними мовами, такими як JavaScript, Python, PHP та Ruby, у контексті веб-розробки, визначаючи сильні та слабкі сторони кожної з них залежно від конкретних вимог.

Продуктивність

C++ відомий своєю високою продуктивністю, що робить його ідеальним для додатків, які потребують швидкої обробки даних та низької затримки, таких як ігри або реальний час.

- **C:** має подібну продуктивність до C++, оскільки обидві мови компілюються в машинний код, що дає високу швидкість виконання.
- **Java:** зазвичай працює повільніше, оскільки використовує віртуальну машину (JVM), яка інтерпретує байт-код замість прямої компіляції в машинний код.
- **Python:** є більш інтерпретованою мовою, ніж Java, тому її продуктивність часто нижча за C++.

- **PHP:** є інтерпретованою мовою, і хоча її швидкість виконання зросла завдяки оновленням, вона не досягає рівня C++ у питаннях продуктивності.
- **Ruby:** також інтерпретується, і хоча його продуктивність значно покращена в останніх версіях, він зазвичай поступається C++ у швидкості виконання.
- **JavaScript:** має високу продуктивність завдяки сучасним механізмам компіляції, таким як V8 у Chrome, але все одно поступається C++ в питаннях обробки великих обсягів даних і швидкості.

Управління пам'яттю

C++ використовує ручне управління пам'яттю, що надає розробникам максимальний контроль над ресурсами. Однак це вимагає обережності, оскільки можуть виникнути проблеми, такі як витоки пам'яті, якщо пам'ять не звільняється коректно.

- **C:** також використовує ручне управління пам'яттю.
- **Java:** застосовує автоматичний збір сміття, що спрощує розробку, але може вплинути на продуктивність, адже процес збору сміття може стати затримкою.
- **Python:** також має автоматичний збір сміття, що спрощує управління пам’яттю, але не дає такого рівня контролю, як у C++.
- **PHP:** має автоматичний збір сміття, що полегшує розробку, але знижує можливість оптимізації пам'яті.
- **Ruby:** також має автоматичне управління пам'яттю через збір сміття, що зручно, але не забезпечує такого рівня контролю, як у C++.
- **JavaScript:** використовує автоматичний збір сміття в середовищі виконання, але дозволяє розробникам контролювати пам'ять через управління об'єктами.

Типізація

C++ використовує статичну типізацію, що допомагає виявляти помилки на етапі компіляції, але робить код більш громіздким.

- **C:** також має статичну типізацію, подібно до C++.
- **Java:** є мовою з сильною статичною типізацією і суворими правилами типів, що робить код безпечнішим, але інколи важким для роботи.
- **Python:** має динамічну типізацію, що дозволяє писати більш лаконічний код, але збільшує ризик помилок, які можна виявити тільки під час виконання.
- **PHP:** використовує слабку статичну типізацію, що дозволяє легко працювати з типами даних, але може призводити до проблем з типами на етапі виконання.
- **Ruby:** має динамічну типізацію, що робить код коротким і зручним, але інколи може спричинити труднощі при відлагодженні.
- **JavaScript:** також є динамічно типізованою мовою, що полегшує розробку, але може спричиняти проблеми з типами під час виконання.

Парадигми програмування

C++ підтримує кілька парадигм програмування: процедурне, об'єктно-орієнтоване та узагальнене програмування, що робить його універсальним інструментом для різних завдань.

- **C:** зазвичай використовує процедурне програмування.
- **Java:** повністю об'єктно-орієнтована мова, з великим акцентом на інкапсуляцію, наслідування та поліморфізм.
- **Python:** підтримує процедурне, об'єктно-орієнтоване та функціональне програмування, що дає розробникам велику гнучкість.
- **PHP:** основною парадигмою є процедурне програмування, хоча з підтримкою ООП в новіших версіях.
- **Ruby:** в першу чергу об'єктно-орієнтована мова, хоча також підтримує функціональне програмування.
- **JavaScript:** підтримує як об'єктно-орієнтоване, так і функціональне програмування, що дозволяє комбінувати різні підходи.

Складність вивчення

C++ є мовою середньої складності для вивчення, через складний синтаксис і необхідність керувати пам'яттю вручну.

- **C:** має схожий рівень складності з C++, оскільки також вимагає ручного управління пам'яттю та працює з низьким рівнем абстракції.
- **Java:** зазвичай легше вивчити, ніж C++, завдяки більш простій типізації та автоматичному збору сміття.
- **Python:** вважається однією з найпростіших мов для початківців завдяки простому синтаксису і читабельності коду.
- **PHP:** має простий синтаксис і відносно низький поріг для початку роботи, але зручний лише для веб-розробки.
- **Ruby:** має простий синтаксис і зручний для розробки, особливо в контексті веб-додатків.
- **JavaScript:** має помірну складність для новачків, хоча вивчення асинхронних концепцій і сучасних фреймворків може бути складним.

Області застосування

C++ застосовується в різних галузях, де потрібна висока продуктивність і контроль над ресурсами, таких як ігри, системне програмування, наукові обчислення та вбудовані системи.

- **C:** використовується для системного програмування, операційних систем і вбудованих систем.
- **Java:** широко використовується для розробки мобільних додатків (Android), веб-сервісів і корпоративних рішень.
- **Python:** популярний для веб-розробки, автоматизації, наукових обчислень, обробки даних і створення сценаріїв.
- **PHP:** домінує в веб-розробці, зокрема для створення серверної частини веб-додатків.
- **Ruby:** переважно використовується для веб-розробки (особливо в контексті Ruby on Rails).

- **JavaScript:** є основною мовою для розробки клієнтської частини веб-додатків, а також використовується для серверної розробки за допомогою Node.js.

Вибір мови програмування завжди залежить від конкретних вимог проекту та рівня кваліфікації команди розробників. C++ пропонує безпрецедентну продуктивність та детальний контроль над ресурсами, але його ефективне застосування вимагає глибоких знань і досвіду.

Завдяки своїм можливостям, C++ стане найкращим вибором для проектів, де ключовими факторами є висока продуктивність, оптимізація ресурсів та специфічні технічні вимоги. Однак, для більшості веб-додатків, що потребують швидкого розгортання, масштабованості та зручності у підтримці, краще обирати мови з більш широкою екосистемою та інструментами для веб-розробки.

Враховуючи ці фактори, важливо ретельно оцінювати потреби проекту та обирати інструменти, які найкраще відповідають його цілям.

Висновки до розділу 1

У першому розділі було здійснено детальний аналіз основних підходів до розробки веб-додатків, популярних мов програмування для вебу, а також ролі C++ у цьому контексті. Основні підходи до розробки web-додатків варіюються залежно від специфіки проекту та вимог до функціональності. Мови програмування, такі як JavaScript, Python, PHP, Java та C#, займають лідируючі позиції завдяки своїм потужним екосистемам, бібліотекам і підтримці широкого спектра веб-функцій.

C++ у веб-розробці є менш поширеним вибором, але завдяки своїй високій продуктивності, точному контролю над пам’яттю та ресурсами, а також можливості інтеграції з іншими системами, він є цінним інструментом для специфічних задач, таких як розробка високопродуктивних систем, вбудованих додатків або програм, що потребують оптимізації ресурсів.

Порівняння C++ з іншими мовами виявило його сильні сторони, зокрема в продуктивності та контролі над пам’яттю, проте також були визначені й недоліки,

зокрема складність розробки та підтримки коду, ризики безпеки та обмеження масштабованості у порівнянні з більш популярними мовами для вебу.

РОЗДІЛ 2

ПОРІВНЯННЯ НАЯВНИХ ІНСТРУМЕНТІВ ДЛЯ WEB-РОЗРОБКИ НА C++

Розробка веб-додатків на C++ вимагає використання спеціалізованих інструментів, фреймворків та бібліотек, які дозволяють ефективно взаємодіяти з серверною частиною, базами даних, а також забезпечувати інтеграцію з frontend-частиною. Оскільки C++ є мовою низького рівня, вибір інструментів для веб-розробки має велике значення для забезпечення продуктивності, масштабованості та простоти розгортання додатків.

У цьому розділі ми розглянемо основні фреймворки та бібліотеки для розробки серверної частини на C++, зокрема легкі та потужні рішення для роботи з HTTP, WebSockets, а також асинхронні фреймворки. Крім того, ми зупинимося на інструментах для роботи з базами даних, засобах інтеграції C++ з frontend-частиною через REST API, WebSockets та WebAssembly. Окрему увагу буде приділено інструментам для контейнеризації та масштабування додатків за допомогою таких технологій, як Docker, Nginx, FastCGI та Kubernetes.

Цей розділ допоможе зрозуміти, які інструменти та підходи найкраще підходять для різних аспектів веб-розробки на C++, а також дозволить зробити обґрунтований вибір для розробки високопродуктивних та масштабованих веб-додатків.

2.1. Фреймворки та бібліотеки для серверної частини

Перш ніж перейти безпосередньо до огляду інструментів для C++, варто зрозуміти ключову відмінність між бібліотеками та фреймворками. Обидва ці підходи передбачають використання готового коду для спрощення розробки, проте мають суттєві концептуальні відмінності.

Головна технічна різниця полягає у принципі "інверсії управління" (Inversion of Control, IoC). Використовуючи бібліотеку, розробник самостійно

визначає, коли та як викликати її функції, зберігаючи повний контроль над виконанням програми. Натомість фреймворк бере на себе управління потоком виконання: він надає певні точки розширення, де можна додати власний код, але самостійно вирішує, коли і як його викликати.

Таким чином, бібліотеки дають більше гнучкості, дозволяючи вибудовувати архітектуру за власним бажанням, тоді як фреймворки забезпечують стандартизований підхід до розробки, спрощуючи інтеграцію, але водночас накладаючи певні обмеження. Вибір між цими інструментами залежить від потреб проєкту та рівня контролю, який необхідний розробникам над логікою серверної частини веб-додатку. [7]

Серверна частина веб-додатків є критично важливою для забезпечення стабільної роботи системи, обробки запитів, взаємодії з базами даних, а також для реалізації логіки додатка.

У C++ існує кілька потужних фреймворків і бібліотек, які дозволяють ефективно створювати серверні додатки з високою продуктивністю, мінімальними затримками та гнучкістю в налаштуваннях.

У цьому підрозділі розглядаються основні фреймворки та бібліотеки, що використовуються для створення серверної частини веб-додатків на C++, кожен з яких має свої переваги, особливості використання та напрямки оптимізації. Вибір конкретного інструменту залежить від вимог проекту, таких як обсяг навантаження, необхідність в реальному часі, підтримка асинхронних запитів або простота інтеграції з іншими технологіями.

1. CppCMS

CppCMS — це високопродуктивний фреймворк для веб-розробки, орієнтований на програмістів C++. Він спеціально розроблений для ефективної роботи під великим навантаженням, забезпечуючи при цьому надзвичайно швидку генерацію динамічного веб-контенту. На відміну від традиційних веб-фреймворків, що використовують багатопотокову обробку, CppCMS

застосовує асинхронний підхід, що робить його надзвичайно швидким і економним у використанні ресурсів.

Фреймворк надає повний контроль над контекстом веб-додатку, що особливо корисно для розробників, які прагнуть глибокої кастомізації своїх рішень.

Основні особливості:

- **Висока продуктивність і низька затримка** – Оптимізована архітектура дозволяє швидко обробляти велику кількість паралельних запитів із мінімальними затримками.
- **Сучасні стандарти C++** – Підтримка новітніх можливостей мови забезпечує використання сучасних інструментів розробки.
- **Вбудована безпека** – Фреймворк автоматично захищає веб-додатки від поширених вразливостей, забезпечуючи безпечне середовище за замовчуванням.
- **Гнучка інтеграція** – Дозволяє легко підключати інші бібліотеки C++, що спрощує розробку складних додатків.

Сфери застосування:

Ідеально підходить для платформ, що вимагають високої продуктивності та ефективності у використанні ресурсів, зокрема торгових платформ у реальному часі, популярних веб-порталів та додатків для обробки великих обсягів даних.

2. Wt (Web Toolkit)

Wt — це повноцінна бібліотека C++ для створення веб-додатків, які за своїм інтерфейсом нагадують традиційні настільні програми. Вона реалізує підхід, заснований на використанні віджетів, що дозволяє розробникам створювати інтерактивні веб-додатки з миттєвими реакціями на дії користувача. Фреймворк бере на себе обробку запитів і рендеринг HTML, використовуючи WebSockets, що робить його придатним для розробки веб-додатків у реальному часі.

Основні особливості:

- **Модель програмування на основі віджетів** – API Wt нагадує API графічних бібліотек для настільних додатків, що значно полегшує перехід для розробників із досвідом у створенні GUI.
- **Підтримка WebSockets** – Фреймворк дозволяє встановлювати двосторонній зв’язок між клієнтом і сервером, що є критично важливим для додатків з онлайн-чатом, інформаційними панелями у реальному часі та інструментами для спільної роботи.
- **Інтеграція з базами даних** – Wt містить вбудовану ORM (Object-Relational Mapping), що значно спрощує роботу з базами даних і управління даними.
- **Масштабованість** – Фреймворк спроєктований таким чином, щоб ефективно працювати на кількох серверах, що дозволяє створювати великі веб-додатки.

Сфери застосування:

Wt ідеально підходить для створення корпоративних рішень, веб-інструментів та багатофункціональних інтернет-додатків, що вимагають високого рівня інтерактивності.

3. Pistache

Pistache — це сучасний REST-фреймворк для C++, що дозволяє швидко та легко створювати RESTful веб-служби. Він розроблений з акцентом на високу продуктивність та масштабованість, що робить його ідеальним для побудови архітектури мікросервісів. Завдяки простому API, Pistache легко освоїти і застосовувати, навіть якщо розробник не має досвіду роботи з веб-розробкою на C++.

Основні особливості:

- **Простий і зручний API**: Pistache надає API, що легко освоїти навіть новачкам у веб-розробці на C++.
- **Висока продуктивність і масштабованість**: Фреймворк добре справляється з великим навантаженням, що робить його ідеальним для мікросервісів і розподілених систем.

- **Синхронна та асинхронна обробка**: Pistache підтримує обидва підходи до обробки запитів, що дає розробникам гнучкість у виборі найкращого методу для їхніх додатків.
- **Легка інтеграція з іншими бібліотеками**: Це дозволяє розширювати фреймворк при необхідності додавання специфічної функціональності.

Сфери застосування:

Pistache ідеально підходить для створення RESTful API та мікросервісів, які повинні ефективно обробляти багато запитів і швидко відповідати на них. Він особливо підходить для додатків, що вимагають надшвидкої комунікації з мінімальними затримками між сервісами.

4. Crow

Crow — ще один потужний C++ фреймворк, що дозволяє розробникам легко створювати веб-додатки та API. Він пропонує надзвичайно ефективний інтерфейс для обробки HTTP-запитів та побудови відповідей на них. Crow багато чого запозичив з фреймворка Express.js, завдяки чому його синтаксис і використання знайомі розробникам, які мають досвід роботи з JavaScript-фреймворками.

Основні особливості:

- **Легкість у використанні**: Crow є легким і зручним для користувача, що робить його відмінним вибором для розробників, які потребують швидкого та простого рішення для запуску додатків.
- **Підтримка JSON**: Фреймворк має вбудовану підтримку JSON, що є необхідним для розробки веб-API та додатків. Це полегшує обробку даних у форматі, який підтримується більшістю веб-додатків.
- **Синхронна та асинхронна обробка**: Crow надає можливість вибору між синхронною та асинхронною обробкою запитів, що дозволяє ефективно виконувати виклики в залежності від вимог.

- **Система маршрутизації, схожа на Express.js**: Crow реалізує систему маршрутизації, натхненну Express.js, що робить визначення маршрутів простим і зручним, а застосування HTTP-запитів — інтуїтивно зрозумілим.

Сфери застосування:

Crow є більше фреймворком, ніж бібліотекою, і націлений на розробників, яким потрібне легке та зручне рішення для створення веб-API. Він особливо підходить для веб-додатків середнього і малого розміру та для розробки мікросервісів.

5. Boost

Boost.Beast — це частина бібліотек Boost і забезпечує вбудовану підтримку HTTP та WebSocket протоколів, розроблених за допомогою Boost.Asio. Цей фреймворк ідеально підходить для розробників, яким потрібно мінімізувати вплив на комунікацію HTTP, але при цьому зберегти всі переваги та недоліки WebSocket. Beast дуже налаштовуваний, що дозволяє створювати додатки з точним контролем за мережевими комунікаціями.

Основні особливості:

- **Низькорівневий контроль**: Boost.Beast надає низькорівневий контроль над HTTP та WebSocket протоколами, що підходить для розробників, які хочуть детально налаштувати мережеву комунікацію за своїми вимогами.
- **Бібліотеки Boost**: Як частина добре відомої та перевіреної бібліотеки Boost, Beast отримує вигоду від великої документації та підтримки спільноти Boost.
- **Висока продуктивність**: Завдяки інтеграції з Boost.Asio, Boost.Beast забезпечує високу продуктивність для ефективної обробки асинхронних мережевих операцій.
- **Докладна документація**: Beast має детальну документацію та приклади, що спрощує процес навчання та впровадження для розробників.

Сфери застосування:

Boost.Beast — це ідеальний вибір для розробників, які потребують точного контролю над HTTP та WebSocket комунікаціями, особливо для складних веб-додатків, що вимагають налаштування протоколів або висококласних мережевих функцій, розроблених під конкретні потреби.

6. Restbed

Restbed — це всеохоплюючий фреймворк для створення RESTful веб-сервісів, орієнтований на розробку масштабованих і високопродуктивних додатків. Його основною перевагою є асинхронна обробка запитів. Restbed також забезпечує легку інтеграцію SSL/TLS, не знижуючи продуктивність розробників. Завдяки модульній архітектурі, розробники можуть використовувати лише необхідні компоненти, що дозволяє зробити Restbed легким для створення веб-сервісів.

Основні особливості:

- **Асинхронна обробка запитів**: Restbed підтримує асинхронну обробку, що дозволяє ефективно справлятися з великою кількістю одночасних запитів.
- **Модульна архітектура**: Розробники можуть включити в свій додаток тільки необхідні компоненти фреймворку, що допомагає зберегти легкість і швидкість роботи.
- **Інтеграція SSL/TLS**: Просте підключення Restbed до SSL/TLS для забезпечення безпечної комунікації між клієнтами і серверами.
- **Кросплатформеність**: Restbed підтримує різні операційні системи, що дає розробникам гарантію його роботи на будь-якому середовищі.

Сфери застосування:

Restbed — це відмінний вибір для створення безпечних, масштабованих веб-сервісів, що базуються на асинхронній обробці запитів і відповідів, з підтримкою SSL/TLS. Цей фреймворк підходить для додатків, що працюють з чутливими даними або тих, для яких важливий високий рівень безпеки.

Хоча C++ не є найбільш поширеною мовою для веб-розробки, з правильними бібліотеками він може стати потужним інструментом для створення

передових і високопродуктивних веб-додатків. Вище наведені бібліотеки пропонують вибір між потужними фреймворками, такими як CppCMS і Wt, та легкими варіантами, як-от Pistache і Crow. Завдяки цьому можна ефективно використовувати швидкість і продуктивність C++ для створення надійних і масштабованих веб-додатків, обираючи відповідну бібліотеку. [8]

2.2. Засоби роботи з базами даних

У сучасних веб-додатках робота з базами даних є невід’ємною частиною архітектури. Вибір відповідних засобів для взаємодії з базами даних важливий не тільки для забезпечення зручності розробки, але й для досягнення високої продуктивності, масштабованості та безпеки. У цьому розділі буде розглянуто основні інструменти та бібліотеки, що дозволяють ефективно працювати з базами даних в екосистемі C++.

Ми розглянемо як класичні методи доступу до даних через SQL, так і сучасні підходи, які використовують ORM (Object-Relational Mapping) для спрощення роботи з даними.

Правильний вибір технології для роботи з базами даних може значно полегшити процес розробки, забезпечити стабільність додатків і дозволити працювати з великими обсягами інформації.

ODB — це відкритий, крос-платформений і крос-базовий об'єктно-реляційний маппінг (ORM) для C++, який дозволяє зберігати об'єкти C++ в реляційній базі даних без необхідності працювати з таблицями, стовпцями, SQL-запитами або вручну писати код для маппінгу. ODB підтримує реляційні бази даних, такі як SQLite, PostgreSQL, MySQL, Oracle та Microsoft SQL Server. Крім того, ODB має додаткові профілі для бібліотек Boost та Qt, які дозволяють безперешкодно використовувати типи значень, контейнери та розумні вказівники з цих бібліотек у ваших постійних класах C++.

Ця система розроблена таким чином, щоб інтегруватися з вашою архітектурою, займаючись тільки зберіганням об'єктів C++ і не втручаючись у інші функції програми.

Використання ODB для зберігання об'єктів має кілька переваг:

1. **Простота використання**. ODB автоматично генерує код для перетворення даних між вашими класами C++ і базою даних, дозволяючи маніпулювати постійними об'єктами через простий об'єктно-орієнтований API бази даних.
2. **Структура коду**. ODB приховує деталі роботи з базою даних, що дозволяє писати логіку програми за допомогою природних об'єктних термінів, що спрощує та покращує читабельність і розуміння коду.
3. **Безпека**. API для зберігання об'єктів та запитів ODB є статично типізованими. Ви використовуєте ідентифікатори C++ замість рядків для звертання до членів об'єкта, а згенерований код гарантує сумісність типів між C++ та базою даних, що допомагає виявити помилки на етапі компіляції, а не виконання.
4. **Портативність бази даних**. Оскільки код для перетворення даних генерується автоматично, легко перейти від одного постачальника бази даних до іншого або підтримувати кілька баз одночасно.
5. **Оптимальна продуктивність**. ODB розроблений з урахуванням високої продуктивності та мінімального навантаження на пам'ять. Для кожної операції з базою даних використовуються всі доступні методи оптимізації, такі як підготовлені запити та кешування з'єднань, запитів і буферів. Постійні класи не мають накладних витрат по пам'яті на кожен об'єкт, оскільки ODB не додає додаткових "базових" членів у кожен клас і не виділяє окремі структури даних для кожного об'єкта.
6. **Підтримка та обслуговування**. Автоматичне генерування коду та еволюція схеми бази даних мінімізують зусилля, необхідні для адаптації програми до змін у постійних класах. Код для перетворення даних зберігається окремо

від оголошень класів і логіки програми, що полегшує налагодження та обслуговування програми. [9]

ODB є потужним і зручним інструментом для роботи з реляційними базами даних в C++, що дозволяє програмістам ефективно зберігати об'єкти C++ без необхідності писати складний SQL-код або вручну виконувати маппінг між об'єктами та базою даних.

Завдяки автоматичному генеруванню коду та високій продуктивності, ODB спрощує процес розробки, роблячи його більш зручним, безпечним і надійним.

Підтримка кількох популярних СУБД, таких як SQLite, PostgreSQL, MySQL, Oracle і Microsoft SQL Server, забезпечує високу портативність та можливість легко перемикатися між різними базами даних.

SOCI — це бібліотека для доступу до баз даних у C++, яка дозволяє інтегрувати SQL-запити безпосередньо в код, зберігаючи при цьому відповідність стандартам C++. Початково розроблена Мацеєм Собчакам як Simple Oracle Call Interface (SOCI), ця бібліотека стала зручнішою альтернативою для роботи з Oracle та іншими базами даних у C++.

Основною метою SOCI є надання програмістам C++ інтуїтивно зрозумілого способу роботи з SQL-базами даних. Якщо інші бібліотеки здаються занадто складними або важкими у використанні, SOCI може стати ідеальним рішенням завдяки простому та природному API. Бібліотека також пропонує широку інтеграцію з типами даних Boost, а також можливість розширення за допомогою користувацьких типів даних.

Попри те, що SOCI орієнтована на C++, вона також підтримує використання з іншими мовами програмування, що робить її універсальним інструментом для розробників.

Ось кілька ключових переваг SOCI:

1. **Простота використання**: Забезпечує зручний та інтуїтивно зрозумілий API для виконання SQL-запитів і роботи з результатами, що спрощує розробку.

2. **Підтримка кількох СУБД**: SOCI сумісна з найбільш популярними системами управління базами даних, що робить її чудовим вибором для різних проєктів і дозволяє легко змінювати СУБД без великих зусиль.
3. **Інтеграція з C++**: Вона дозволяє працювати з даними безпосередньо як з об'єктами C++, що спрощує роботу з базами даних і зменшує необхідність у великій кількості SQL-коду.
4. **Гнучкість і розширюваність**: SOCI підтримує інтеграцію з користувацькими типами даних і дозволяє налаштовувати бібліотеку під специфічні потреби проєкту.
5. **Підтримка асинхронних операцій**: Для підвищення продуктивності SOCI надає можливість виконувати асинхронні запити, що корисно в додатках з високими вимогами до пропускної здатності та швидкості обробки запитів.
6. **Широка сумісність з іншими бібліотеками**: Бібліотека має хорошу сумісність з іншими популярними C++ бібліотеками, такими як Boost, що дозволяє ефективно використовувати її у більш складних проєктах. [10]

Завдяки своїй простоті, гнучкості та високій продуктивності SOCI є потужним інструментом для розробки додатків, що потребують доступу до баз даних.

Нативні драйвери баз даних — це спеціалізовані бібліотеки, розроблені для прямої взаємодії з движком бази даних конкретної системи управління базами даних (СУБД).

Ці драйвери є критично важливими для розробників, які прагнуть до ефективних низькорівневих операцій з базами даних без накладних витрат, що виникають через додаткові шари абстракції.

Вони дають змогу досягти таких переваг:

1. **Висока масштабованість**: Нативні драйвери забезпечують швидку обробку запитів, що дозволяє масштабувати додаток для обробки великих обсягів даних і високих навантажень. Це є важливим фактором у випадках, коли

система повинна підтримувати десятки тисяч одночасних з’єднань або обробляти великі набори даних у реальному часі.

2. **Оптимізація запитів на рівні бази даних**: Нативні драйвери дозволяють тісно інтегруватися з конкретними можливостями СУБД, що дає змогу використовувати особливості бази даних, такі як індекси, кешування, паралельне виконання запитів та інші можливості для значної оптимізації продуктивності.
3. **Підтримка специфічних функцій бази даних**: Завдяки нативним драйверам, розробники можуть скористатися специфічними функціями конкретних СУБД, такими як зберігання складних типів даних, робота з транзакціями або спеціалізованими інструментами для аналізу та обробки даних, які можуть бути недоступні через ORM або інші бібліотеки.
4. **Глибша інтеграція з іншими технологіями**: Оскільки нативні драйвери безпосередньо працюють із базою даних, це дозволяє використовувати додаткові механізми інтеграції з іншими технологіями, такими як системи черг, кешування, або реалізація специфічних протоколів для з'єднань з базою даних, що може бути критично важливим для великих, розподілених систем.
5. **Контроль за ресурсами**: Використовуючи нативні драйвери, розробники мають більше контролю над управлінням пам’яттю та іншими ресурсами, що дозволяє досягти найкращих результатів у випадках, коли потрібно оптимізувати використання пам’яті та інших системних ресурсів.
6. **Зниження ризику помилок**: Оскільки нативні драйвери не вимагають додаткових абстракцій або перетворень, є менший ризик виникнення помилок, які можуть бути спричинені неправильною інтерпретацією або обробкою запитів через більш загальні абстракції, такі як ORM.

Використання нативних драйверів є ідеальним варіантом для розробників, які працюють із критичними додатками, де продуктивність, точність і контроль над взаємодією з базою даних мають першочергове значення.

Це дозволяє створювати ефективні, масштабовані рішення для обробки даних з високими вимогами до продуктивності. [11]

2.3. Інтеграція C++ з frontend-частиною

Інтеграція C++ з фронтенд-частиною веб-додатків є важливою складовою створення високопродуктивних, масштабованих та ефективних веб-рішень.

Хоча C++ традиційно асоціюється з розробкою серверної логіки, низькорівневим програмуванням і розробкою системного програмного забезпечення, його використання в розробці веб-додатків для фронтенду набуває все більшої популярності завдяки потужним бібліотекам і інструментам для створення інтерактивних користувацьких інтерфейсів.

Інтеграція між C++ і фронтендом зазвичай здійснюється через різні механізми, такі як WebAssembly, JavaScript API або спеціалізовані бібліотеки, що дозволяють C++ коду взаємодіяти з елементами веб-сторінки або обробляти дані на клієнтському боці без необхідності в додаткових серверних запитах.

Ми розглянемо різні підходи до інтеграції C++ з фронтендом, можливості, які вони відкривають, та основні переваги цього підходу для розробки сучасних веб-додатків.

Підходи до інтеграції C++ з фронтенд-частиною

Існує кілька основних способів взаємодії фронтенду з бекендом, написаним на C++. Кожен із них має свої переваги та підходить для певних типів додатків.

Нижче розглянемо ключові підходи, їх принципи роботи та можливості використання.

1. RESTful API:

REST (Representational State Transfer) — це архітектурний стиль, який використовується для створення веб-сервісів. Це один із найпопулярніших підходів для взаємодії між фронтендом і бекендом. REST API працює через стандартні HTTP-запити та відповіді, що дозволяє обмінюватися даними у

структурованому форматі (зазвичай JSON або XML). Бекенд надає набір ендпоінтів (URL-адрес), які відповідають за виконання різних функцій, а фронтенд звертається до цих ендпоінтів для отримання або зміни даних.

Процедура роботи:

Клієнт (фронтенд):

1. Надсилає HTTP-запит до API-ендпоінту на сервері.
2. Вказує метод запиту (GET — отримати дані, POST — створити, PUT — оновити, DELETE — видалити).
3. Може передавати тіло запиту (наприклад, JSON) з необхідними даними для створення чи оновлення об'єкта.

Сервер (бекенд):

1. Отримує HTTP-запит і визначає, який ендпоінт і метод використовуються.
2. Обробляє запит, що може включати взаємодію з базою даних, виконання обчислень або виклик сторонніх сервісів.
3. Формує відповідь із необхідними даними, статус-кодом (наприклад, 200 — успіх, 404 — не знайдено, 500 — помилка сервера) та додатковою інформацією.

Клієнт:

1. Отримує відповідь від сервера.
2. Аналізує статус-код та вміст відповіді.
3. Оновлює інтерфейс користувача або виконує подальші дії на основі отриманої інформації.

Додаткові переваги та особливості REST API:

Легкість у використанні: REST API використовує стандартні HTTP-методи, що робить його простим у розробці та тестуванні.

Масштабованість: REST добре підходить для розподілених систем і мікросервісної архітектури.

Кешування: Можливість кешування відповідей сервера для зменшення навантаження.

Незалежність від клієнтської платформи: REST API може використовуватися будь-якими клієнтськими додатками (веб, мобільними, IoT).

Можливі недоліки:

Обмежена підтримка реального часу: REST не передбачає постійного з'єднання між клієнтом і сервером, що може бути неефективним для додатків, які потребують оновлення в реальному часі.

Перевантаженість даними: Оскільки сервер повертає фіксовані структури, клієнт може отримувати зайві дані, що впливає на продуктивність.

REST API залишається найпоширенішим способом інтеграції між фронтендом і бекендом завдяки простоті, гнучкості та підтримці стандартних веб-протоколів. Однак для додатків, які потребують реального часу або оптимізації запитів, можуть знадобитися альтернативні рішення, такі як WebSockets або GraphQL.

2. WebSockets:

WebSockets — це протокол постійного, двонаправленого зв’язку між клієнтом і сервером. На відміну від традиційного HTTP, який працює за принципом запит-відповідь, WebSockets дозволяють підтримувати безперервне з'єднання, що робить їх ідеальним вибором для додатків, які потребують оновлення в реальному часі, таких як чати, біржові торги, онлайн-ігри або системи моніторингу.

Процес роботи WebSockets:

Клієнт (фронтенд):

1. Встановлює WebSocket-з'єднання із сервером за допомогою спеціальної URL-адреси.
2. Відправляє повідомлення на сервер із даними або запитом.

Сервер (бекенд):

1. Приймає вхідне з'єднання та обробляє повідомлення клієнта.
2. Може надсилати оновлення або відповіді клієнту в будь-який момент часу.

3. Підтримує одночасне підключення великої кількості клієнтів без необхідності відкривати нові HTTP-запити щоразу.

Клієнт:

1. Отримує повідомлення від сервера в реальному часі.
2. Оновлює інтерфейс користувача відповідно до отриманих даних.
3. Може оперативно реагувати на оновлення, що покращує користувацький досвід.

Додаткові переваги WebSockets:

Оновлення в реальному часі: Додатки можуть миттєво отримувати нові дані без потреби періодичного опитування сервера.

Ефективність: Менше накладних витрат у порівнянні з постійними HTTP-запитами, оскільки з'єднання залишається відкритим.

Підтримка одночасних з'єднань: Сервер може працювати з великою кількістю клієнтів, зберігаючи відкриті сесії.

Менша затримка: Зменшений час очікування у порівнянні з REST API, оскільки немає необхідності повторного встановлення з'єднання.

Можливі недоліки:

Складність реалізації: Робота з WebSockets вимагає додаткового налаштування на сервері та обробки постійних з'єднань.

Сумісність: Не всі проксі-сервери або корпоративні мережі підтримують WebSockets, що може створювати проблеми з доступом.

Підтримка безпеки: Для захищеного з'єднання необхідно використовувати wss:// замість ws://, а також впроваджувати додаткові механізми аутентифікації.

WebSockets є потужним рішенням для додатків, що вимагають миттєвого оновлення даних та інтерактивної взаємодії між клієнтом і сервером. Вони забезпечують високу продуктивність і мінімальну затримку, що особливо важливо для онлайн-чатів, біржових платформ та інших сервісів реального часу. Однак у разі простих CRUD-операцій REST API може бути більш зручним і простим у використанні рішенням.

3. Серверний рендеринг (SSR - Server-Side Rendering)

Серверний рендеринг (SSR) — це підхід до рендерингу веб-сторінок, коли сервер формує повноцінний HTML-код сторінки перед відправленням його в браузер користувача. Це суттєво знижує навантаження на клієнтський пристрій, прискорює перший рендер сторінки та покращує SEO, оскільки пошукові системи отримують вже готовий контент.

Процес роботи SSR:

Клієнт (фронтенд):

1. Надсилає HTTP-запит на сервер для отримання певної сторінки.

Сервер (бекенд):

1. Обробляє запит та формує повноцінний HTML-код, використовуючи серверні мови програмування (наприклад, Node.js, PHP, Python, Ruby).
2. Інтегрує у сторінку всі необхідні дані, наприклад, інформацію з бази даних або API.
3. Вбудовує у HTML необхідний JavaScript-код для подальшої взаємодії користувача зі сторінкою.
4. Відправляє згенеровану сторінку клієнту.

Клієнт:

1. Отримує готову HTML-сторінку та миттєво відображає її в браузері.
2. Після завантаження JavaScript-код активується, додаючи динамічну взаємодію та можливість подальшого оновлення контенту без перезавантаження сторінки.

Додаткові переваги SSR:

Швидший перший рендер: Користувач отримує готовий контент ще до завантаження JavaScript, що покращує швидкість завантаження.

Покращене SEO: Пошукові системи отримують статичний HTML-код сторінки, що сприяє кращій індексації.

Підтримка слабких пристроїв: Оскільки обробка контенту відбувається на сервері, клієнтські пристрої не перевантажуються обробкою JavaScript.

Краща доступність: SSR покращує відображення контенту для користувачів з повільним інтернет-з'єднанням або обмеженими можливостями браузера.

Можливі недоліки:

Навантáження на сервер: Оскільки кожен запит вимагає генерації HTML на сервері, високі навантаження можуть збільшити час відповіді та вимагати додаткових ресурсів.

Затримка при взаємодії: Після першого рендеру деякі взаємодії можуть займати більше часу, оскільки браузер повинен завантажити та ініціалізувати JavaScript.

Складність реалізації: Інтеграція SSR може бути складнішою у порівнянні з клієнтським рендерингом, особливо при використанні сучасних фреймворків, таких як React або Vue.

Серверний рендеринг — це ефективний підхід для веб-додатків, які потребують швидкого завантаження, хорошого SEO та підтримки широкого спектра пристроїв. Він особливо корисний для контентних сайтів, блогів, маркетингових платформ і корпоративних сторінок. Однак для інтерактивних SPA-додатків (Single Page Application) може бути доцільніше використовувати гібридний підхід, комбінуючи SSR з клієнтським рендерингом.

4. GraphQL

GraphQL — це мова запитів до API, розроблена компанією Facebook, яка суттєво змінює підхід до взаємодії клієнта з бекенд-сервісами. На відміну від традиційного REST API, де кожен ендпоінт повертає фіксовану структуру даних, GraphQL дозволяє клієнту самостійно визначати, які саме дані йому потрібні. Це робить API більш гнучким, продуктивним і зручним для розробників.

Процес роботи GraphQL

Клієнт (фронтенд):

1. Формує GraphQL-запит, чітко визначаючи структуру та поля, які потрібно отримати.
2. Надсилає запит на GraphQL-сервер.

Сервер (бекенд):

1. Отримує запит і розбирає його, щоб визначити, які дані необхідні.
2. Збирає дані з відповідних джерел (бази даних, зовнішні API, кеш).
3. Формує єдиний об'єкт-відповідь, що точно відповідає структурі запиту.
4. Повертає результат клієнту.

Клієнт:

1. Отримує відповідь у зручному форматі, що містить лише необхідні дані.
2. Використовує отриману інформацію для оновлення інтерфейсу або подальших операцій.

Додаткові переваги GraphQL:

Гнучкість: Клієнт сам визначає, які дані йому потрібні, що дозволяє зменшити обсяг переданих даних.

Менше запитів: GraphQL дозволяє отримати всі необхідні дані одним запитом, у той час як REST API може вимагати кілька запитів до різних ендпоінтів.

Оптимізація роботи з даними: API не повертає зайві поля, що зменшує навантаження на мережу та пришвидшує обробку запитів.

Краще для мобільних додатків: Зменшення обсягу переданих даних критично важливе для мобільних пристроїв із повільним інтернет-з'єднанням.

Можливість об’єднання різних джерел даних: GraphQL може отримувати дані одночасно з різних баз даних, API та мікросервісів.

Можливі недоліки:

Складність налаштування: Порівняно з REST, налаштування GraphQL може вимагати більше зусиль та досвіду.

Витрати на обробку запитів: Динамічний характер GraphQL-запитів може створювати додаткове навантаження на сервер, особливо при складних структурах даних.

Кешування: REST API використовує кешування HTTP, що спрощує повторне використання відповідей. У GraphQL це доводиться реалізовувати вручну.

GraphQL — це сучасний підхід до взаємодії клієнта з сервером, який надає гнучкість і ефективність у роботі з даними. Він ідеально підходить для складних застосунків, мобільних додатків і великих систем із великою кількістю джерел даних. Однак для простих проєктів або систем із високими вимогами до кешування REST API може залишатися кращим вибором.

Зв’язок між фронтендом і бекендом можна порівняти зі створенням надійного каналу комунікації між двома ключовими складовими застосунку: інтерфейсом користувача та серверною логікою. Розуміння їхніх ролей і вибір оптимальних методів обміну даними є запорукою швидкої, стабільної та безпечної взаємодії.

Залежно від потреб проєкту можна використовувати різні підходи: RESTful API для стандартизованої взаємодії з сервером, WebSockets для миттєвих оновлень у реальному часі, Server-Side Rendering (SSR) для швидкого завантаження контенту та SEO-оптимізації, або GraphQL для гнучкого отримання даних із мінімальним навантаженням.

Ключовим фактором успіху є вибір стратегії, що найкраще відповідає вимогам вашого застосунку, а також ретельне планування та якісна реалізація коду. Оптимізований зв’язок між фронтендом і бекендом не лише покращує продуктивність системи, а й забезпечує користувачам швидкий, зручний і безперебійний досвід взаємодії. [12]

2.4. Інструменти розгортання та DevOps

У сучасній розробці програмного забезпечення ефективне розгортання застосунків та управління інфраструктурою відіграють вирішальну роль у забезпеченні стабільності, масштабованості та безпеки систем. DevOps-підхід поєднує розробку (Development) і операційне управління (Operations), що дозволяє автоматизувати процеси розгортання, тестування та моніторингу, зменшуючи ризики та прискорюючи випуск нових версій продукту.

У сфері розробки на C++ управління складністю великих проєктів та залежностей є постійною проблемою. Практики DevOps для C/C++ спрямовані на оптимізацію процесів побудови, інтеграції та розгортання програмного забезпечення.

Нижче наведені основні вимоги до сучасних практик DevOps для C++, які допомагають підвищити ефективність, зменшити кількість помилок і масштабувати операції на різних платформах:

1. **Управління та повторне використання бінарних файлів**:

Побудова програмного забезпечення з нуля кожного разу може бути дуже неефективною, затратною і схильною до помилок, особливо у великих проєктах. Повторне використання вже зібраних бінарних файлів для різних платформ дозволяє уникнути зайвих зборок, що економить час і ресурси. Основною проблемою є управління цими бінарними файлами, щоб забезпечити їх сумісність, особливо коли йдеться про питання сумісності ABI (Application Binary Interface) між різними компіляторами і платформами. Хороша система DevOps має фокусуватися на послідовному повторному використанні бінарів, при цьому враховуючи ці проблеми сумісності.

2. **Централізоване сховище бінарних файлів**:

У багатьох організаціях бінарні файли зберігаються і управляються в кількох системах, таких як apt (Debian), rpm (Red Hat), nuget (Windows) і pkg (OSX). Керування пакетами в різних системах може бути дорогим, затратним за часом і схильним до помилок. Використання єдиного репозиторію або сервера для зберігання та управління всіма бінарними файлами значно спрощує управління

інфраструктурою. Це не тільки робить зберігання бінарів більш ефективним, а й економить час, надаючи централізоване місце для їх отримання та управління ними, зменшуючи складність підтримки різних систем.

3. **Адаптація до будь-якої системи збірки, платформи та інструменту**:

Різноманітність систем збірки та інструментів, що використовуються в розробці C++, становить ще одну проблему. Ідеальною була б стандартизація на єдиній системі збірки, але в реальності багато проєктів та бібліотек глибоко інтегровані з застарілими системами. Міграція проєктів до нової системи збірки часто вимагає значних зусиль, як у плані часу, так і ресурсів. Тому інструменти DevOps повинні бути досить гнучкими для інтеграції з різними системами збірки та платформами — від старих інструментів до найновіших і найефективніших систем, що дозволяє легко адаптуватися до різних середовищ.

4. **Інтеграція з іншими інструментами**:

Практики DevOps для C/C++ передбачають безперешкодну інтеграцію з різноманітними інструментами екосистеми. Це включає платформи безперервної інтеграції (CI), такі як Jenkins, Travis CI і Appveyor (які часто використовуються для відкритих проєктів), а також популярні IDE, такі як Visual Studio і Xcode. Крім того, інструменти мають добре інтегруватися з фреймворками для тестування, інструментами статичного аналізу, системами покриття коду та системами керування версіями, такими як Git. Забезпечення ефективної взаємодії всіх цих інструментів значно покращує робочий процес розробки та розгортання.

5. **Управління пакетами для ефективного повторного використання ресурсів**:

Пакет — це архів, що містить бінарні файли, конфігураційні файли та метадані, включаючи інформацію про залежності, опис та інше. Менеджери пакетів відіграють важливу роль в екосистемі DevOps, дозволяючи ефективно повторно використовувати існуючі ресурси. Розробники можуть легко інтегрувати сторонні бібліотеки у вигляді попередньо зібраних бінарів, що зменшує необхідність у повторній роботі та пришвидшує цикл розробки програмного

забезпечення. Управління пакетами допомагає збирати залежності разом, спрощуючи співпрацю та забезпечуючи використання правильних версій бібліотек на різних етапах розробки, що знижує кількість помилок і прискорює виведення продукту на ринок.

Враховуючи ці сучасні вимоги DevOps, організації можуть створювати більш ефективні, масштабовані та керовані робочі процеси для розробки на C++. Автоматизація, гнучкість і інтеграція з наявними інструментами відіграють ключову роль у спрощенні складних процесів та покращенні загальної якості розробки програмного забезпечення. [13]

Довгий час C та C++ не мали підтримки менеджерів пакетів, на відміну від інших мов програмування, що спричинило багато проблем у спільноті. Ці мови відчували себе поза увагою з боку DevOps і автоматизації.

Досягнення бінарної сумісності в C та C++ є складним завданням, оскільки додатки націлені на різноманітні платформи та конфігурації, для кожної з яких потрібні окремі бінарні файли. Conan Center — це відкритий пакетний менеджер для C та C++, який надає розробникам рішення для зменшення проблем із бінарною сумісністю, одночасно підтримуючи всі основні операційні системи. Conan допомагає компаніям будувати додатки та автоматизувати керування залежностями між платформами.

Менеджер пакетів Conan є децентралізованим рішенням для C та C++, який дозволяє розробникам обмінюватися пакетами за допомогою моделі push-pull, схожої на Git. Завдяки Conan, DevOps став реальністю для C та C++: Conan використовує "рецепти" — структуровані Python-скрипти, які визначають, як збирати бібліотеку, чітко викликаючи будь-яку систему збірки. Для керування різними конфігураціями та сумісністю ABI Conan використовує "налаштування", що дозволяє при зміні налаштувань створювати модифіковану бінарну версію тієї ж бібліотеки.

Безперервна інтеграція для проєктів C та C++ є складною через специфічні характеристики цих мов програмування та процес компіляції в рідний код, коли

код у заголовних файлах (або навіть в исходних файлах) вбудовується, а також через різні типи зв'язування для статичних та динамічних бібліотек.

Завдяки Conan, зокрема, стало можливим автоматизувати оновлення залежностей і спростити процеси складання і тестування, що дозволяє покращити ефективність та знизити кількість помилок у процесі розробки. Інструменти для автоматизації, які розвиваються в межах Conan, є ключем до ефективної інтеграції і надають можливість більш гнучкого та масштабованого управління проєктами C та C++ (рис. 2.4.1).

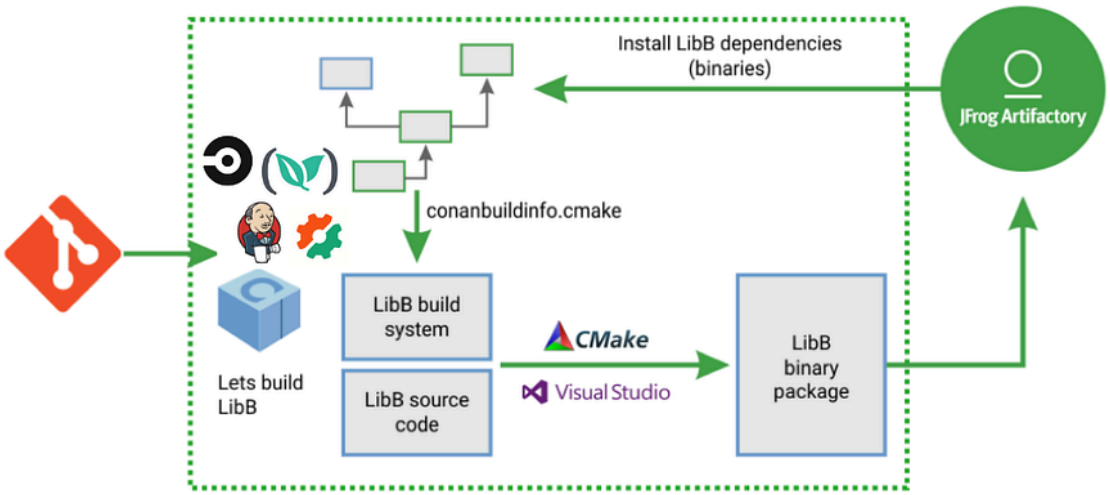

Рис 2.4.1 Приклад циклу безперервної інтеграції для C/C++ з використанням Conan, Artifactory та Jenkins

Наявність добре підтримуваної системи управління пакетами означає наявність загальної мови та бібліотек, які підтримуються однаково всіма основними компіляторами на всіх стандартних операційних системах. Conan чудово працює для вбудованих систем, що допомагає ефективно керувати залежностями та бінарними файлами. Це забезпечує плавність та організованість процесів DevOps, спрощуючи їх впровадження та автоматизацію. [14]

Сучасні практики DevOps для C та C++ забезпечують значне покращення ефективності, стабільності та масштабованості процесів розробки програмного забезпечення, що є критичними для великих проєктів та багатоплатформних додатків. Завдяки інструментам, таким як менеджери пакетів (наприклад, Conan),

автоматизація процесів побудови, інтеграції, тестування та розгортання стає набагато простішою та менш затратною.

Менеджери пакетів, які дозволяють ефективно управляти залежностями та бінарними файлами, є важливим інструментом для досягнення бінарної сумісності та зменшення складності управління проєктами. Особливо важливо, що ці інструменти можуть підтримувати різноманітні платформи, що є важливим аспектом для розробки на C та C++, де кожна платформа має свої вимоги до бінарів і компіляції.

Інтеграція з платформами безперервної інтеграції, такими як Jenkins, а також використання інструментів для автоматизації оновлення залежностей дозволяє значно скоротити час на розробку, знижує ймовірність помилок та підвищує надійність програмного забезпечення. Це дає можливість командам зосередитися на створенні інноваційних рішень замість витрачання часу на повторювані та затратні процеси побудови і тестування.

Таким чином, правильне використання інструментів для автоматизації та управління залежностями в рамках DevOps дозволяє забезпечити більш стабільне, швидке і ефективне розгортання програмного забезпечення, а також сприяє розвитку та впровадженню нових технологій у галузі розробки на C та C++.

Висновки до розділу 2

У цьому розділі було розглянуто важливі фреймворки, бібліотеки та інструменти, що дозволяють ефективно розробляти серверну частину додатків на C++, інтегрувати їх з базами даних та здійснювати взаємодію з frontend-частиною.

Фреймворки, такі як Crow та Wt, надають необхідні інструменти для створення легких і потужних web-додатків, в той час як Boost.Beast та uWebSockets забезпечують ефективну роботу з HTTP та WebSockets, що є критичними для розробки реальних веб-сервісів та додатків з високими вимогами до продуктивності.

Для роботи з базами даних, використання ORM-бібліотек, таких як ODB, а також бібліотек для прямого доступу до SQL, як SOCI, дозволяє значно спростити

інтеграцію з різними СУБД та зменшити кількість шаблонного коду. Додатково, наявність нативних драйверів для популярних баз даних, як MySQL і PostgreSQL, гарантує високу продуктивність та стабільність роботи з даними.

Інтеграція з frontend-частиною через REST API, WebSockets і WebAssembly надає широкий спектр можливостей для реалізації сучасних веб-додатків, забезпечуючи зв'язок між сервером та клієнтом у реальному часі, а також дозволяє виконувати C++ код безпосередньо в браузері, що відкриває нові горизонти для високопродуктивних веб-додатків.

Таким чином, використання сучасних фреймворків і бібліотек надає розробникам на C++ потужні інструменти для створення ефективних, масштабованих і високопродуктивних серверних рішень, забезпечуючи зручну інтеграцію з базами даних і frontend-частиною, а також підтримку новітніх технологій для веб-розробки.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ WEB-ДОДАТКУ НА C++

Використання C++ для розробки веб-додатків дає можливість отримати високу продуктивність і оптимізацію ресурсів, що особливо важливо для проектів, де важливими є швидкість виконання, масштабованість та ефективне використання апаратних ресурсів. Однак, така розробка має свої специфічні виклики, зокрема, щодо інтеграції з веб-технологіями, управління мережевими запитами та обробки великих обсягів даних у реальному часі.

У цьому розділі буде розглянуто ключові аспекти створення веб-додатку на C++, включаючи вибір інструментів і фреймворків для розробки, а також налаштування середовища розробки та розгортання. Особливу увагу буде приділено вибору відповідних бібліотек для роботи з HTTP-протоколами, та інтеграції з базами даних. У результаті цього процесу буде продемонстровано, як побудувати ефективний та масштабований веб-додаток, що здатен працювати у вимогливих середовищах і забезпечувати високу продуктивність за мінімальних витрат ресурсів.

Також в рамках цього розділу буде розглянуто приклади інтеграції C++ з frontend-частиною. Це дозволить краще зрозуміти, як можна використовувати C++ в сучасних веб-рішеннях і як адаптувати цей потужний інструмент для роботи у веб-середовищі.

3.1. Вибір інструментів та середовища розробки

Розробка веб-додатків на C++ передбачає використання широкого спектра інструментів, включаючи компілятори, фреймворки, бібліотеки для роботи з мережею та базами даних, а також засоби автоматизації, тестування та розгортання. Важливо обрати такі інструменти, які забезпечать баланс між продуктивністю, гнучкістю та легкістю інтеграції.

У цьому підрозділі будуть розглянуті ключові критерії вибору середовища розробки та основні інструменти, які допомагають ефективно створювати, тестувати та підтримувати веб-додатки на C++.

Для розробки було обрано Visual Studio Code (VS Code) — це багатофункціональний редактор коду, розроблений компанією Microsoft, який спрощує процес програмування та підтримує широкий спектр мов і фреймворків.

Завдяки більш ніж десяти мільйонам активних користувачів щомісяця, він став улюбленим інструментом як для початківців, так і для досвідчених розробників. VS Code має потужну екосистему розширень, що підвищують продуктивність і дозволяють налаштовувати середовище під конкретні завдання.

Однією з ключових переваг Visual Studio Code є його велика бібліотека розширень. Користувачі можуть вибирати з тисяч плагінів, які додають підтримку різних мов програмування, інструментів для налагодження та тем оформлення. Наприклад, розширення для Python забезпечує такі можливості, як IntelliSense та підтримка Jupyter Notebook, що робить його ідеальним для роботи як дата-сайентістів, так і розробників програмного забезпечення. Вбудована інтеграція з Git дозволяє легко керувати репозиторіями без необхідності виходу з редактора.

Крім того, VS Code забезпечує інтеграцію з компіляторами GCC та Clang, що дає змогу створювати кросплатформні проєкти. Для управління збіркою використовувалася система CMake, яка забезпечує зручне конфігурування проєкту та автоматичне створення файлів для збірки на різних платформах.

Для налагодження застосунку використовувалися розширення C/C++ Tools від Microsoft та інтегрований термінал, який дозволяє запускати локальний сервер безпосередньо із середовища розробки.

Ще одна значна перевага VS Code — це кросплатформеність. Редактор стабільно працює на Windows, macOS та Linux, що забезпечує єдиний досвід роботи на різних операційних системах. Це особливо корисно для команд, які використовують різні платформи в рамках спільної розробки.

Інтерфейс VS Code розроблений для зручності та ефективності (рис. 3.1.1).

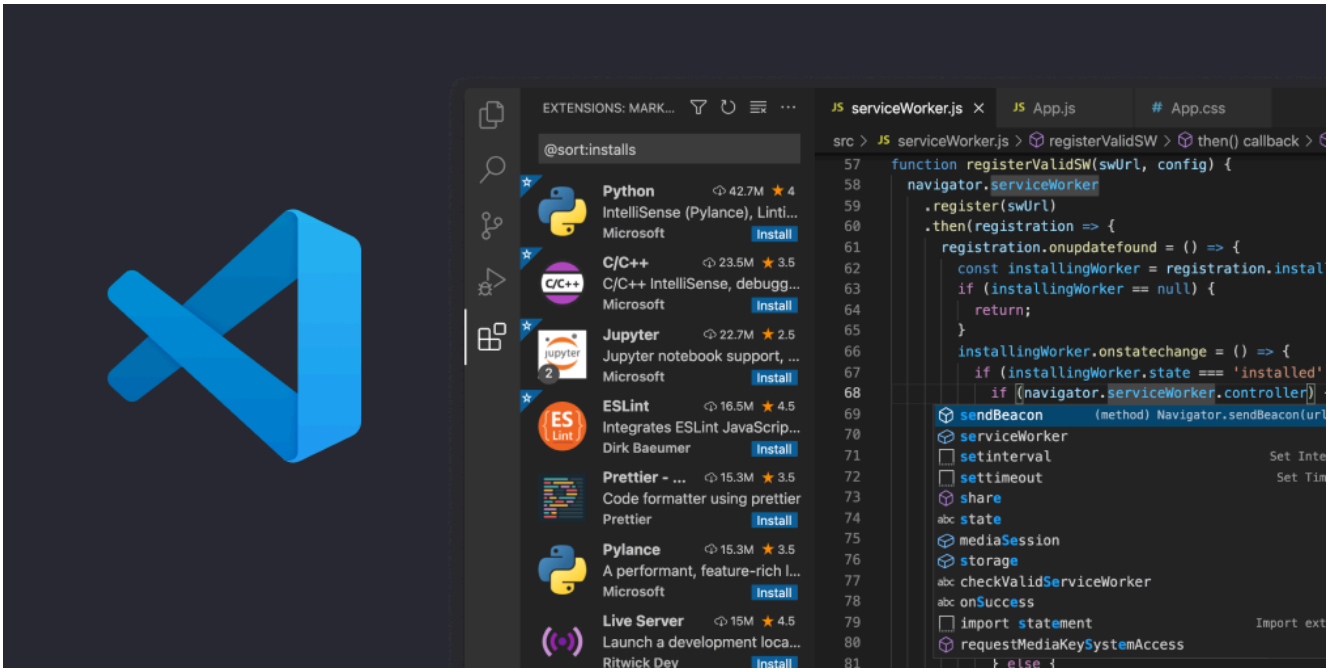

Рис. 3.1.1 Інтерфейс Visual Studio Code

Функції, такі як вкладки для редагування, розділений перегляд і вбудований термінал, допомагають розробникам працювати більш інтуїтивно. Крім того, редактор підтримує функцію спільної роботи в реальному часі через розширення, наприклад Live Share, що є особливо корисним для віддалених команд, які працюють над спільними проєктами.

Переваги:

- **Підтримка різних платформ** — стабільна робота на Windows, macOS та Linux.
- **Велика екосистема розширень** — тисячі плагінів для різних мов і інструментів.
- **Активна спільнота** — постійна підтримка та оновлення.
- **Інтеграція з Git** — зручне керування версіями безпосередньо в редакторі.
- **Гнучкі налаштування** — можливість змінювати теми та макети під власні потреби.
- **Потужні засоби налагодження** — підтримка відлагодження для багатьох мов програмування.

- **Функції спільної роботи** — підтримка роботи в реальному часі (наприклад, через Live Share).
- **Інтеграція з хмарними сервісами** — підтримка збереження, синхронізації та CI/CD-процесів.

Недоліки:

- **Високе споживання ресурсів** — може навантажувати систему, особливо при роботі з великими проєктами.
- **Початковий поріг входу** — потребує часу для освоєння новими користувачами.
- **Деякі функції розподілені між розширеннями** — необхідність додаткового налаштування.
- **Залежність від інтернету** — деякі можливості працюють лише при підключенні до мережі.
- **Опора на розширення** — розширені функції доступні тільки через додаткові модулі.
- **Менш підходить для не-розробників** — основний фокус спрямований на програмістів.
- **Можливі помилки з плагінами сторонніх розробників** — не всі розширення працюють стабільно.
- **Складні конфігурації для просунутих випадків** — може вимагати детального налаштування.
- **Використання VS Code:**
- **Розробка програмного забезпечення** — підтримка багатьох мов програмування.
- **Аналіз та візуалізація даних** — підходить для роботи дата-сайентістів.
- **Співпраця в крос-функціональних командах** — зручне середовище для командної роботи.
- **Фрілансери та незалежні розробники** — можливість працювати з різними фреймворками та мовами.

- **Освіта та навчання програмуванню** — використовується в університетах та школах.
- **Корпоративне середовище** — забезпечує стандартизацію розробки в компаніях.
- **Аналіз веб-коду** — корисний для маркетологів та SEO-фахівців.
- **Підтримка Open Source-проєктів** — зручний для розробників, що працюють над спільними проєктами.
- **Віддалена робота та рев'ю коду** — спрощує код-рев'ю завдяки вбудованим інструментам.
- **Розробка ігор** — підтримує роботу з різними парадигмами програмування. [15]

3.2. Архітектура розробленого web-додатку

Розроблений веб-додаток реалізовано за класичною архітектурною моделлю клієнт–сервер. Серверна частина написана мовою C++ із використанням легкого веб-фреймворку Crow, який забезпечує обробку HTTP-запитів, маршрутизацію та підтримку REST API. Crow вирізняється високою продуктивністю та ефективною роботою з багатопоточними запитами, що дозволяє створювати швидкі та надійні вебсервери. Завдяки простому синтаксису та підтримці middleware, Crow спрощує інтеграцію додаткових бібліотек і розширень. [16]

Клієнтська частина (frontend) реалізована з використанням Angular, одного з провідних сучасних фреймворків для створення односторінкових вебдодатків (SPA — Single Page Application). Angular базується на модульній архітектурі та забезпечує двосторонній зв'язок даних (two-way data binding) і реактивне програмування, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних.

Крім того, Angular має потужну екосистему вбудованих бібліотек і інструментів для маршрутизації, управління станом додатку, інтеграції з REST API та WebSocket, що дозволяє створювати масштабовані та надійні вебзастосунки. [17]

Для взаємодії між клієнтом і сервером використовується **REST API**, що дозволяє передавати дані у форматі JSON.

Архітектура передбачає такі основні компоненти (рис. 3.2.1):

1. **Серверна логіка (Backend)** — відповідає за прийом запитів, обробку даних, взаємодію з базою даних і формування відповідей у форматі JSON.
2. **База даних** — для зберігання даних використовується Postgresql, що забезпечує простоту розгортання.
3. **Клієнтська частина (Frontend)** — реалізує інтерфейс користувача, відправляє запити до API та відображає отримані дані.

Така архітектура дозволяє легко масштабувати проєкт, додаючи нові модулі та функціональність, забезпечує простоту підтримки коду і інтеграції з іншими системами, а також гарантує високу продуктивність і зручність користувацького інтерфейсу завдяки використанню сучасних фреймворків і технологій.

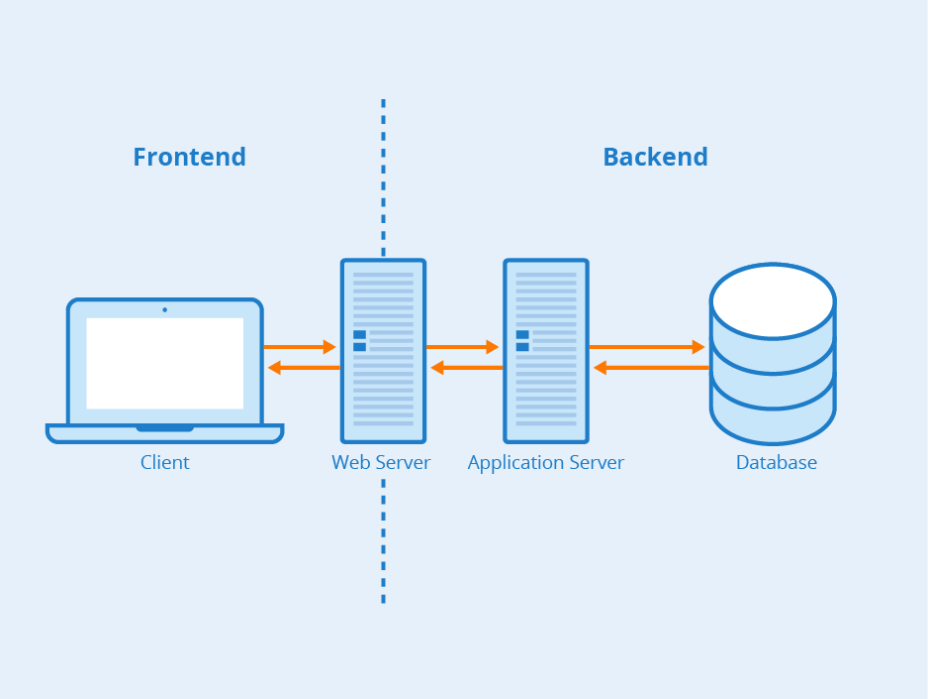

Рис. 3.2.1 Архітектура web-додатку

3.3. Реалізація серверної частини за допомогою C++

Реалізація серверної частини веб-додатку здійснювалася мовою програмування C++ із використанням легкого веб-фреймворку Crow, який забезпечує обробку HTTP-запитів, маршрутизацію, підтримку REST API та багатопоточну роботу з клієнтськими запитами. Обраний фреймворк дозволяє створювати високопродуктивні сервери із мінімальним накладним кодом, що є важливим для забезпечення масштабованості та ефективного використання апаратних ресурсів.

Серверна частина виконує такі основні функції:

- обробка HTTP-запитів від клієнта;
- маршрутизація запитів до відповідних обробників;
- взаємодія з базою даних Postgresql;
- формування відповідей у форматі JSON;
- асинхронна обробка багатопоточних запитів для забезпечення високої продуктивності.

3.3.1. Ініціалізація серверного додатку

Ініціалізація серверного додатку є першим і ключовим етапом реалізації серверної частини веб-додатку. На цьому етапі створюється базова структура сервера, визначаються порти для прийому запитів, налаштовуються параметри багатопоточності та підключення необхідних бібліотек.

Основні кроки ініціалізації серверу включають:

1. **Імпорт необхідних бібліотек.**

Для роботи з Crow підключається заголовочний файл crow_all.h, що містить всі необхідні компоненти фреймворку. Це дозволяє уникнути ручного підключення окремих модулів Crow та спрощує компіляцію.

2. **Створення об’єкту додатку.**

   Центральним елементом серверної частини є об’єкт crow::SimpleApp, який відповідає за маршрутизацію запитів, управління потоками та формування відповідей. Створення цього об’єкту є обов’язковим для подальшого визначення маршрутів і запуску серверу.
3. **Визначення базових маршрутів.**

   На етапі ініціалізації рекомендується створити базовий маршрут, наприклад /, для перевірки працездатності серверу. Це дозволяє швидко переконатися, що сервер успішно запущений і готовий приймати запити.
4. **Налаштування порту та багатопоточності.**

   Сервер запускається на конкретному порту (у розробленому додатку використовується порт **18080**) і може обробляти запити у кількох потоках. Багатопоточність забезпечує можливість одночасної обробки численних запитів від клієнтів, що критично для масштабованих веб-додатків.
5. **Запуск серверного циклу.**

   Після налаштування всіх параметрів виконується запуск сервера методом run(). Після цього сервер переходить у робочий режим і готовий обробляти HTTP-запити від клієнтської частини.

3.3.2. Реалізація маршрутів

Маршрути (routes) визначають, як серверна частина обробляє різні HTTP-запити. У розробленому веб-додатку маршрути поділені на логічні блоки для кращої структурованості коду та простоти підтримки.

Для організації маршрутів у додатку використовується функція setup_routes, яка виконує ініціалізацію всіх маршрутів, що забезпечує централізоване керування логікою обробки запитів

Основні блоки маршрутів включають:

1. **Маршрути авторизації (setup_routes_for_auth)**
   Цей блок забезпечує обробку запитів, пов’язаних з аутентифікацією та авторизацією користувачів. Сюди входять функції для входу в систему, реєстрації нових користувачів та перевірки токенів доступу. Така ізоляція підвищує безпеку і дозволяє централізовано керувати процесом аутентифікації.
2. **Маршрути роботи з базою даних (setup_routes_for_db)**
   Даний блок відповідає за маршрути, які обробляють операції над даними в базі Postgresql. Це включає додавання нових записів, отримання даних, їх оновлення та видалення. Модульна структура маршрутів забезпечує простоту підтримки і дозволяє швидко інтегрувати нові функціональні можливості без змін у базовій логіці серверу.
3. **Маршрути інтеграції з сервісами Google (setup_routes_for_google)**
   Цей блок реалізує маршрути для взаємодії із зовнішніми сервісами, такими як Google OAuth або Google Sheets. Завдяки виділеному модулю сторонні інтеграції не впливають на основну логіку серверу, що спрощує тестування та підвищує стабільність роботи.

3.3.3. Асинхронна обробка запитів

Асинхронна обробка запитів є важливою складовою серверної частини веб-додатку, оскільки дозволяє ефективно працювати з великою кількістю одночасних підключень, не блокуючи основний потік виконання. Такий підхід

критично важливий для забезпечення високої продуктивності та стабільності системи при навантаженні.

У розробленому веб-додатку серверна частина підтримує багатопоточну обробку запитів, що дозволяє одночасно обробляти декілька запитів від клієнтів. Кожен HTTP-запит може виконуватися незалежно від інших, що знижує затримки при обробці операцій, пов’язаних із базою даних, зовнішніми API або складними обчисленнями.

Асинхронна обробка забезпечує:

1. **Незалежність потоків:** Кожен запит обробляється окремим потоком або асинхронною задачею, що дозволяє серверу обробляти кілька запитів одночасно.
2. **Швидкий відгук клієнту:** Основний потік серверу не блокується при виконанні ресурсомістких операцій, що дозволяє миттєво відповідати на інші запити.
3. **Масштабованість:** Завдяки асинхронній обробці сервер може ефективно працювати з великою кількістю одночасних клієнтських підключень, що особливо важливо для веб-додатків із високим навантаженням.
4. **Покращена взаємодія з базою даних та сторонніми сервісами:** Асинхронні запити дозволяють паралельно виконувати операції над базою даних або роботу з зовнішніми API, мінімізуючи час очікування для користувача.

Організація асинхронної обробки у проекті здійснюється через багатопоточність та розділення маршрутів на модулі. Кожен маршрут може незалежно обробляти запити, викликати відповідні функції логіки та формувати відповіді у форматі JSON.

Завдяки такій структурі сервер здатний швидко масштабуватися і обслуговувати сотні та тисячі одночасних запитів без суттєвого падіння продуктивності.

3.3.4. Взаємодія з базою даних

Взаємодія серверної частини з базою даних є ключовою складовою функціонування веб-додатку, оскільки від цього залежить збереження, обробка та отримання інформації, необхідної для роботи клієнтської частини.

У розробленому додатку для зберігання даних використовується реляційна база Postgresql, яка забезпечує надійність, масштабованість та високу продуктивність при роботі з великими обсягами інформації.

Основні принципи організації взаємодії з базою даних:

1. **Модульна структура роботи з базою даних**

   Всі операції з базою даних реалізовані через окремі модулі та функції, які відповідають за конкретні задачі: додавання записів, отримання даних, оновлення інформації та видалення записів. Такий підхід спрощує підтримку коду і дозволяє швидко додавати нові функціональні можливості без зміни базової логіки серверу.

2. **Безпека доступу до даних**

   Серверна частина обмежує прямий доступ до бази даних лише необхідними запитами через маршрути API. Для авторизації користувачів та контролю доступу використовується окремий модуль маршрутів авторизації. Це дозволяє уникати несанкціонованого доступу та забезпечує конфіденційність даних.

3. **Підтримка асинхронних операцій**

   Для підвищення продуктивності сервер підтримує асинхронну обробку запитів до бази даних. Це дозволяє одночасно виконувати кілька операцій без блокування основного потоку серверу, що особливо важливо при високому навантаженні та роботі з великими масивами даних.

4. **Уніфікація запитів через централізовані функції**

Всі SQL-запити реалізовані через централізовані функції або модулі, що відповідають за конкретні операції з базою даних. Це забезпечує єдину логіку обробки даних, зменшує дублювання коду та полегшує тестування.

5. **Взаємодія з фронтендом через JSON**

   Дані, отримані з бази, перетворюються у формат JSON і передаються клієнтській частині через REST API. Це дозволяє фронтенду на Angular легко інтегрувати та відображати інформацію без додаткових конвертацій.

3.3.5. Формат передачі даних

Для забезпечення ефективної взаємодії між серверною та клієнтською частинами веб-додатку використовується формат JSON. JSON є легким, зрозумілим і широко підтримуваним стандартом обміну даними, який дозволяє фронтенду на Angular отримувати структуровану інформацію у зручному для обробки вигляді.

Основні аспекти використання JSON у серверній частині:

1. **Уніфікація даних**

   Дані, що надходять від бази даних або формуються сервером, перетворюються у єдиний формат JSON перед відправкою клієнту. Це дозволяє забезпечити стандартизовану структуру відповіді незалежно від типу запиту та типу даних.

2. **Зручність обробки на клієнтській стороні**

   JSON легко інтегрується з фронтенд-фреймворком Angular. Структуровані об’єкти JSON можуть бути безпосередньо використані для відображення інформації, двостороннього зв’язку даних (two-way data binding) та реактивного оновлення інтерфейсу користувача.

3. **Підтримка REST API**

   Формат JSON є стандартом обміну даними для REST API, що дозволяє серверу і фронтенду взаємодіяти незалежно від платформи чи мови

програмування. Кожен маршрут серверної частини повертає дані у вигляді JSON, що забезпечує узгодженість запитів і відповідей.

4. **Гнучкість структури даних**

   JSON дозволяє передавати складні структури даних, включаючи вкладені об’єкти та масиви. Це спрощує реалізацію більш складних сценаріїв, наприклад отримання інформації про користувача разом із пов’язаними об’єктами (роль, налаштування, історія дій).

5. **Валідація та безпека**

   Серверна частина забезпечує перевірку формату даних перед відправкою або обробкою отриманих запитів. Це дозволяє уникнути помилок при обробці запитів і забезпечує безпечний обмін інформацією між клієнтом і сервером.

3.4. Клієнтська частина веб-додатку

Фронтенд веб-додатку реалізовано з використанням сучасного фреймворку Angular, який є одним із провідних рішень для створення односторінкових веб-додатків.

Angular забезпечує модульну архітектуру, двосторонній зв’язок даних (two-way data binding), реактивне програмування та гнучку інтеграцію з REST API, що робить його ідеальним для побудови інтерфейсу користувача, який динамічно оновлюється у відповідь на зміни даних.

Архітектура фронтенд-додатку ґрунтується на модульній структурі Angular, яка включає компоненти, сервіси, модулі та маршрутизацію:

1. **Компоненти**

   Компоненти відповідають за відображення окремих елементів інтерфейсу користувача. Кожен компонент ізольований і містить власний шаблон, стилі та логіку, що забезпечує повторне використання та легку підтримку коду.

2. **Сервіси**

   Сервіси відповідають за роботу з даними та бізнес-логіку, включаючи взаємодію з серверною частиною через REST API. Це дозволяє розділяти логіку обробки даних та презентацію, підвищуючи масштабованість і тестованість додатку.

3. **Модулі**

   Модулі об’єднують компоненти та сервіси за функціональною ознакою, що дозволяє організувати код у логічні блоки та забезпечує зручне масштабування додатку.

4. **Маршрутизація (Routing)**

   Використання маршрутизації Angular дозволяє створювати SPA, де користувач може переміщатися між різними сторінками та компонентами без перезавантаження сторінки. Це підвищує швидкість роботи додатку та зручність користувача.

Взаємодія фронтенду з серверною частиною є ключовою для функціонування веб-додатку, оскільки забезпечує обмін даними між користувачем та сервером, а також підтримує актуальність інформації на інтерфейсі.

У розробленому додатку обмін здійснюється через REST API, який дозволяє фронтенду надсилати запити та отримувати дані у форматі JSON, стандартизованому для взаємодії між різними платформами.

Основні аспекти взаємодії фронтенду з сервером:

1. **Отримання даних**

   Коли користувач взаємодіє з інтерфейсом, фронтенд надсилає запити до серверної частини для отримання необхідної інформації. Сервер обробляє запит, отримує дані з бази та повертає їх у структурованому вигляді, що дозволяє Angular швидко відобразити їх на сторінці користувача.

2. **Передача даних на сервер**

   Дані, введені користувачем, такі як форма реєстрації, заповнення профілю або внесення змін у налаштування, передаються на сервер для обробки та

збереження у базі даних. Це забезпечує актуальність даних і узгодженість між клієнтом та сервером.

3. **Асинхронна обробка запитів**

   Для забезпечення швидкого відгуку інтерфейсу взаємодія з сервером здійснюється асинхронно. Це дозволяє фронтенду продовжувати працювати та оновлювати інтерфейс, не чекаючи завершення обробки запиту на сервері, що підвищує продуктивність та комфорт користувача.

4. **Обробка помилок та винятків**

   Взаємодія з серверною частиною передбачає контроль помилок, наприклад при недоступності серверу або некоректних даних. Фронтенд обробляє такі ситуації, відображаючи повідомлення користувачу або повторно надсилаючи запит, що забезпечує стабільну роботу додатку.

5. **Безпека та авторизація**

   Запити до серверної частини проходять через механізми авторизації та аутентифікації. Фронтенд передає токени доступу або ключі авторизації, які сервер перевіряє перед обробкою запиту, що гарантує безпеку даних та обмежує доступ сторонніх користувачів.

   Переваги такої організації взаємодії

- **Швидкий відгук інтерфейсу:** Асинхронні запити дозволяють користувачу отримувати актуальну інформацію без затримок.
- **Стандартизований обмін даними:** Формат JSON забезпечує зрозумілу і структуровану передачу даних між сервером і фронтендом.
- **Масштабованість:** Додавання нових функціональних запитів до API або нових компонентів на фронтенді здійснюється без змін у базовій логіці.
- **Безпека:** Використання токенів доступу і централізованої авторизації гарантує захист даних від несанкціонованого доступу.
- **Сумісність:** REST API дозволяє фронтенду і серверу взаємодіяти незалежно від платформ і технологій, що підвищує гнучкість системи.

3.5. Функціональні модулі системи

Розроблений веб-додаток складається з окремих функціональних модулів, кожен з яких виконує певні завдання та забезпечує зручну взаємодію користувача з платформою. Модульна архітектура дозволяє легко розширювати функціональність системи, додаючи нові можливості без необхідності переробки існуючого коду.

Модулі взаємопов'язані між собою і дозволяють ефективно організовувати робочі процеси, планувати події та зберігати необхідну інформацію. Кожен модуль реалізований як окремий Angular компонент із власною логікою, шаблоном та стилями, що забезпечує повторне використання коду та полегшує підтримку системи.

Основними модулями системи є модуль автентифікації, модуль управління даними, модуль візуалізації інформації та модуль нотаток. Кожен з них виконує специфічні функції для підвищення продуктивності, безпеки та зручності роботи користувача.

3.5.1. Модуль автентифікації

Модуль автентифікації є критично важливим компонентом системи, що забезпечує безпечний доступ користувачів до платформи та захист конфіденційної інформації.

Він реалізує механізми ідентифікації та авторизації, контролюючи доступ до функціональних можливостей додатку на основі ролей та прав користувачів (рис. 3.5.1).

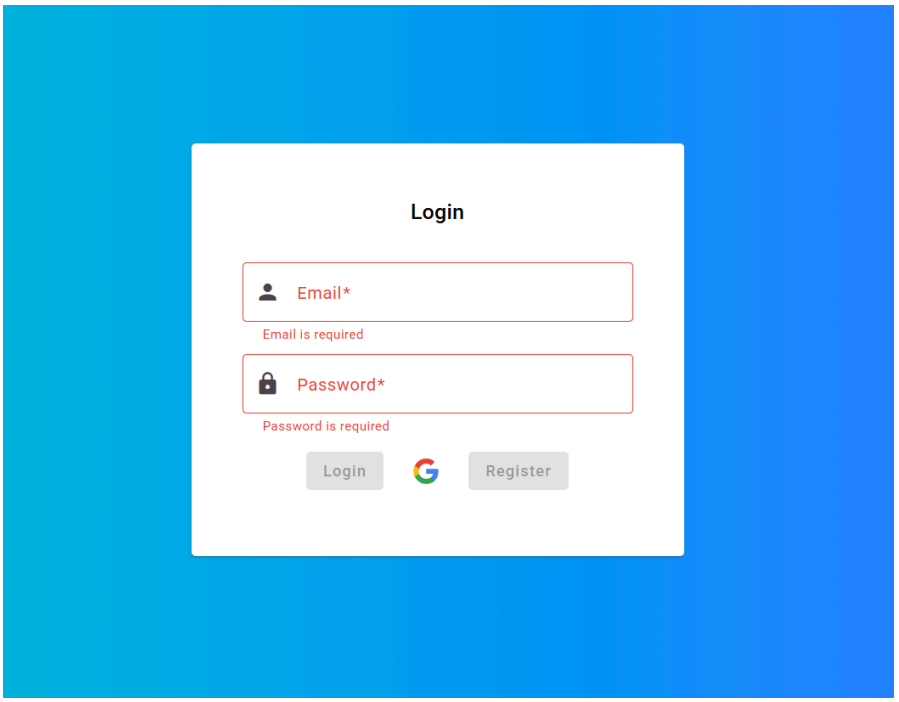

Рис. 3.5.1 Модуль автентифікації

Основні функції модуля автентифікації включають:

1. **Введення даних користувача**
    - Користувач вводить логін (ім’я користувача або електронну адресу) та пароль.
    - Система перевіряє правильність формату введених даних, наприклад довжину пароля або наявність обов’язкових символів.
2. **Перевірка автентичності**
    - Введені дані порівнюються з інформацією у базі даних користувачів.
    - Використовується хешування паролів для безпечного зберігання та порівняння.
    - У разі невірних даних користувач отримує повідомлення про помилку.
3. **Успішний вхід**
    - При правильних даних користувач отримує доступ до системи.
    - Передбачене збереження сесії для спрощення повторного входу.

3.5.2. Модуль управління даними

Модуль управління даними є центральним компонентом системи, що забезпечує відображення, організацію інформації. Цей модуль дозволяє користувачам ефективно працювати з великими обсягами структурованих даних та здійснювати аналітичну обробку інформації (рис. 3.5.2).

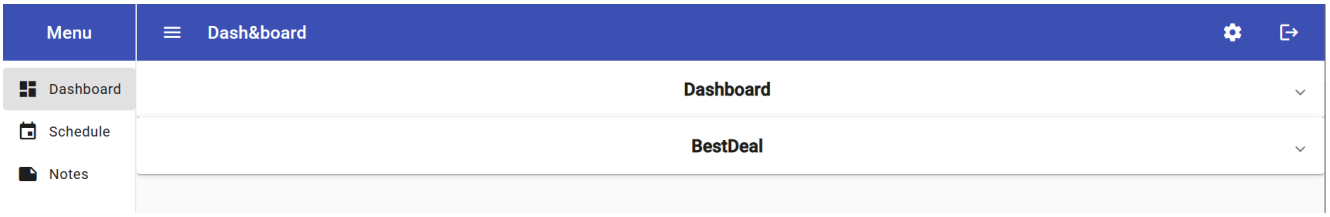

Рис. 3.5.2 Модуль управління даними

3.5.3. Модуль графіку

Модуль графіку відображає події та завдання у вигляді календаря або діаграми, що дозволяє користувачу переглядати планування робочого часу та відстежувати виконання завдань.

У модулі реалізовано:

- Перегляд подій у різних режимах (щоденний, тижневий, місячний);
- Автоматичне позначення конфліктів у розкладі;
- Відображення завдань за пріоритетом, категорією або терміном виконання;
- Інтерактивну синхронізацію з іншими модулями системи, наприклад з дошкою завдань та нотатками;
- Відображення нагадувань про події для зручного планування.

Модуль графіку призначений виключно для перегляду, що забезпечує користувачу можливість швидко орієнтуватися у запланованих подіях та ефективно контролювати свій час (рис. 3.5.3).

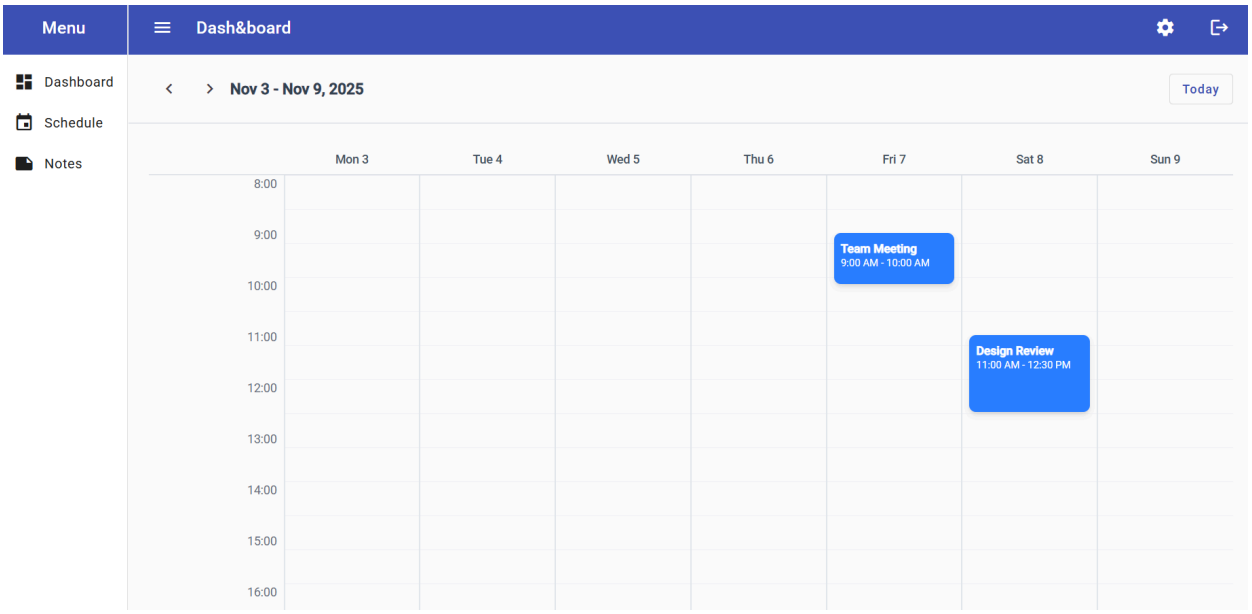

Рис. 3.5.3 Модуль графіку

3.5.4. Модуль нотаток

Модуль нотаток призначений для створення, збереження та організації текстових заміток користувача. Він дозволяє ефективно фіксувати ідеї, важливі деталі або завдання та швидко отримувати до них доступ (рис 3.5.4).

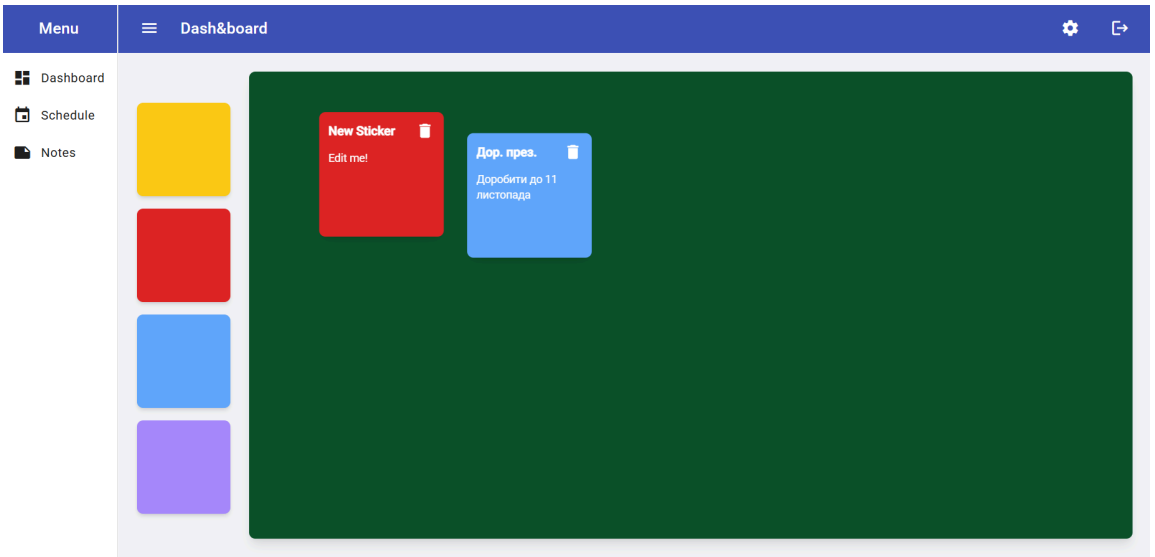

Рис. 3.5.4 Модуль нотаток

Основні функції модуля включають:

- Створення та редагування нотаток;
- Видалення непотрібних нотаток;
- Сортування та групування нотаток за категоріями або тегами;
- Пошук нотаток за ключовими словами для швидкого доступу до потрібної інформації;
- Можливість інтеграції з іншими модулями системи, наприклад, з графіком або дошкою завдань;
- Збереження нотаток у структурованому вигляді для подальшого використання.

Модуль нотаток сприяє підвищенню продуктивності користувача, дозволяє підтримувати порядок у робочих процесах і забезпечує швидкий доступ до важливої інформації.

Висновки до розділу 3

У третьому розділі було розглянуто практичну реалізацію веб-додатку на мові C++ із використанням веб-фреймворку Crow для серверної частини та Angular для фронтенду. Запропонована архітектура клієнт-сервер забезпечує ефективну обробку HTTP-запитів, масштабованість та високу продуктивність системи. Завдяки використанню Crow було реалізовано маршрути для авторизації, роботи з базою даних PostgreSQL, а також інтеграції з зовнішніми сервісами, що гарантує модульність і гнучкість рішення.

Асинхронна обробка запитів та багатопоточність дозволяють системі ефективно працювати з великою кількістю одночасних клієнтських підключень, підвищуючи швидкість відгуку та стабільність роботи. Використання формату обміну даними JSON у поєднанні з REST API створює простий та стандартизований інтерфейс для взаємодії між сервером і клієнтською частиною.

Фронтенд, реалізований на Angular, забезпечує зручний і сучасний користувацький інтерфейс з підтримкою двостороннього зв’язку даних та

реактивного програмування, що підвищує інтерактивність та користувацький досвід.

Отже, практична реалізація показала, що використання C++ у web-розробці, незважаючи на складнощі, дозволяє створювати високопродуктивні, масштабовані та надійні веб-додатки, які відповідають сучасним вимогам. Такий підхід є ефективним вибором для проектів із критичними вимогами до продуктивності та оптимізації ресурсів.

ВИСНОВКИ

У результаті проведеного дослідження було проаналізовано сучасні підходи до веб-розробки, особливості застосування мови C++ у цій сфері, а також порівняно доступні інструменти та фреймворки для створення веб-додатків на основі C++. Було виявлено, що незважаючи на вищу складність розробки, C++ має значні переваги у вигляді високої продуктивності, точного контролю над пам'яттю та ресурсами, що робить його привабливим для розробки високонавантажених, критично важливих систем.

Практична реалізація веб-додатку із серверною частиною на C++ (з використанням фреймворку Crow) та фронтендом на Angular продемонструвала здатність цієї мовної платформи забезпечувати сучасний функціонал, масштабованість і стабільність роботи системи. Запропоновані архітектурні рішення дозволили ефективно обробляти запити, інтегрувати зовнішні сервіси, підтримувати безпечну авторизацію та зручний інтерфейс користувача.

Таким чином, C++ може бути розглянутий як ефективний інструмент для розробки веб-додатків, які потребують високої продуктивності та оптимізації ресурсів, особливо у сферах, де критичним є час відгуку і масштабованість. Однак для більшості традиційних веб-проектів розробникам варто враховувати

складність мови і можливі труднощі інтеграції, зважуючи на ці фактори переваги інших, більш поширених мов і фреймворків.

Загалом, застосування C++ у веб-розробці є перспективним напрямком, що може доповнювати існуючі технології, особливо у специфічних, продуктивно-критичних сценаріях, забезпечуючи баланс між ефективністю та функціональністю сучасних веб-систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stud.com.ua – «ПІДХОДИ ДО РОЗРОБКИ WEB-ДОДАТКІВ» [Електронний ресурс]. URL: https://stud.com.ua/… (дата звернення: 01.07.2025).
2. Stud.com.ua – «WEB-ДОДАТКИ» [Електронний ресурс]. URL: https://stud.com.ua/… (дата звернення: 10.07.2025).
3. itProger – «Порівняння популярних мов програмування для веб-розробки» [Електронний ресурс]. URL: https://itproger.com/… (дата звернення: 20.07.2025).
4. ReachAboveMedia – «Role of C++ in Web Development: A Comprehensive Guide» [Електронний ресурс]. URL: https://reachabove.media/… (дата звернення: 22.07.2025).
5. Nexus Software Systems – «How C++ is Used for Better Performance Web Development Now» [Електронний ресурс]. URL: https://nexussoft.com/… (дата звернення: 25.07.2025).
6. Codebranch – «Unlocking the Power of C++ in Web Development» [Електронний ресурс]. URL: https://codebranch.com/… (дата звернення: 28.07.2025).

7. freeCodeCamp – «The Difference Between a Framework and a Library» [Електронний ресурс]. URL: https://freecodecamp.org/… (дата звернення: 02.08.2025).
8. Analytics Insight – «Top Libraries for Building Web Apps in C++» [Електронний ресурс]. URL: https://analyticsinsight.net/… (дата звернення: 03.08.2025).
9. Code Synthesis – «ODB: C++ Object-Relational Mapping (ORM)» [Електронний ресурс]. URL: https://codesynthesis.com/… (дата звернення: 07.08.2025).
10. Source Forge – «SOCI - The C++ Database Access Library» [Електронний ресурс]. URL: https://sourceforge.net/… (дата звернення: 12.08.2025).
11. Gyata AI – «Overview of Using Native Database Drivers» [Електронний ресурс]. URL: https://gyata.ai/… (дата звернення: 27.08.2025).
12. Geeks For Geek – «How to Connect Front End and Backend» [Електронний ресурс]. URL: https://geeksforgeeks.org/… (дата звернення: 04.09.2025).
13. DevOps – «DevOps Challenges in C/C++ Projects» [Електронний ресурс]. URL: https://devops.com/… (дата звернення: 10.09.2025).
14. Medium – «Implementing DevOps For C and C++ Projects» [Електронний ресурс]. URL: https://medium.com/… (дата звернення: 15.09.2025).
15. Jobicy – «Visual Studio Code Reviews, Features, Pros & Cons in 2025» [Електронний ресурс]. URL: https://jobicy.com/… (дата звернення: 02.10.2025).
16. Crow – «Home» [Електронний ресурс]. URL: https://crowcpp.org/1.2.1/ (дата звернення: 07.10.2025).
17. Angular – «Introduction: What is Angular?» [Електронний ресурс]. URL: https://angular.dev/overview (дата звернення: 13.10.2025).

ДОДАТОК А

Main.cpp

```
#include "crow_all.h"
#include <pqxx/pqxx>
#include "routes.hpp"
#include "bcrypt.hpp"

int main() {
        BCrypt::init();
        crow::SimpleApp app;
                pqxx::connection   c("dbname=dyplomna   user=postgres
password=password host=localhost port=5432");
        setup_routes(app, c);
        CROW_ROUTE(app, "/")([] {
            return "Hello, Crow!";
        });
        std::cout << "Server starting on http://localhost:18080\n";
        app.port(18080).multithreaded().run();
        return 0;
}
```

Auth.cpp

```
#include "bcrypt.hpp"
#include "auth.hpp"
void  setup_routes_for_auth(crow::SimpleApp&  app,  pqxx::connection&
c) {
    //REGISTRATION
CROW_ROUTE(app, "/api/register").methods(crow::HTTPMethod::Post)
([&c](const crow::request& req){
    crow::response res;
    res.set_header("Access-Control-Allow-Origin", "*");
    res.set_header("Content-Type", "application/json");
```

```
    auto body = crow::json::load(req.body);
    if (!body) {
        res.code = 400;
        res.body = "{\"message\":\"Invalid JSON\"}";
        return res;
    }

    std::string username = body["username"].s();
    std::string password = body["password"].s();
    std::string hashed = BCrypt::hash(password);

    try {
        pqxx::work txn(c);

        // Insert user and return its id
        pqxx::result r = txn.exec_params(
             "INSERT INTO users (username, password_hash) VALUES ($1,
$2) RETURNING id",
            username, hashed
        );
        txn.commit();

        int user_id = r[0]["id"].as<int>();
        res.code = 200;
             res.body  =  "{\"message\":\"Registration  successful\",
\"user_id\":" + std::to_string(user_id) + "}";
    } catch (const std::exception& e) {
        res.code = 500;
           res.body  =  std::string("{\"message\":\"")  +  e.what()  +
"\"}";
    }
    return res;
});

//VALIDATION
```

```
CROW_ROUTE(app, "/api/validate").methods(crow::HTTPMethod::Options)
([](const crow::request& req){
    crow::response res;
    res.set_header("Access-Control-Allow-Origin", "*");
    res.set_header("Access-Control-Allow-Methods", "POST, OPTIONS");
    res.set_header("Access-Control-Allow-Headers", "Content-Type");
    res.set_header("Content-Type", "application/json");
    res.code = 204;
    res.end();
    return res;
});

CROW_ROUTE(app, "/api/validate").methods(crow::HTTPMethod::Post)
([&c](const crow::request& req){
    crow::response res;
    res.set_header("Access-Control-Allow-Origin", "*");
    res.set_header("Access-Control-Allow-Headers", "Content-Type");
    res.set_header("Access-Control-Allow-Methods", "POST, OPTIONS");
    res.set_header("Content-Type", "application/json");

    auto body = crow::json::load(req.body);
    if (!body) {
        res.code = 400;
        res.body = "{\"message\":\"Invalid JSON\"}";
        return res;
    }
    std::cout << "[DEBUG] Username: " << body << std::endl;

    std::string username = body["username"].s();
    std::string password = body["password"].s();

    try {
        static bool prepared = false;
        if (!prepared) {
```

```
            c.prepare("get_user", "SELECT id,username, password_hash
FROM users WHERE username = $1");
        prepared = true;
    }
            pqxx::work txn(c);
                   pqxx::result r = txn.exec_prepared("get_user",
username);
        for (const auto& row : r) {
                           std::cout << "[DEBUG] Username: " <<
row["username"].c_str() << std::endl;
                       std::cout << "[DEBUG] Password hash: " <<
row["password_hash"].c_str() << std::endl;
}
        if (r.size() != 1) {
            res.code = 401;
                  res.body = "{\"message\":\"Invalid username or
password\"}";
            return res;
        }

                                std::string stored_hash =
r[0]["password_hash"].as<std::string>();

        if (BCrypt::verify(password, stored_hash)) {
            crow::json::wvalue response_json;
            response_json["message"] = "Login successful";
            response_json["id"] = r[0]["id"].as<int>();

            res.code = 200;
            res.body = res.body = response_json.dump();
        } else {
            res.code = 401;
                  res.body = "{\"message\":\"Invalid username or
password\"}";
        }
```

```
    } catch (const std::exception &e) {
        res.code = 500;
           res.body  =  std::string("{\"message\":\"")  +  e.what()  +
"\"}";
    }

    return res;
});
}
```

Db.cpp

```
#include "db.hpp"
void  setup_routes_for_db(crow::SimpleApp&  app,  pqxx::connection&  c)
{
   //Projects retrieval route
                                                        CROW_ROUTE(app,
"/api/projects/<int>").methods(crow::HTTPMethod::Get)
    ([&c](int user_id){
        crow::response res;
        res.set_header("Access-Control-Allow-Origin", "*");
        res.set_header("Content-Type", "application/json");

        try {
            // Extract user_id from query string
            if (!user_id) {
                res.code = 400;
                         res.body  =  "{\"error\":\"Missing  user_id
parameter\"}";
                return res;
            }

            pqxx::work txn(c);
            pqxx::result r = txn.exec_params(
```

```
                    "SELECT id, name, sales, orders, revenue,
users,created_at "
                    "FROM projects WHERE user_id = $1 ORDER BY
created_at DESC",
                user_id
            );

            crow::json::wvalue projects_json;
            projects_json["projects"] = crow::json::wvalue::list();
            size_t index = 0;
            for (const auto& row : r) {
                crow::json::wvalue project;
                project["id"] = row["id"].as<int>();
                project["name"] = row["name"].as<std::string>();
                project["sales"] = row["sales"].as<double>();
                project["orders"] = row["orders"].as<int>();
                project["revenue"] = row["revenue"].as<double>();
                project["users"] = row["users"].as<int>();

                                projects_json["projects"][index++]   =
std::move(project);
            }
            res.code = 200;
            res.body = projects_json.dump();
            return res;
        }
        catch (const std::exception &e) {
            res.code = 500;
               res.body = std::string("{\"error\":\"") + e.what() +
"\"}";
            return res;
        }
    });

//Calendar
```

```
CROW_ROUTE(app,
"/api/calendar/<int>").methods(crow::HTTPMethod::Get)
([&c](int user_id) {
    crow::json::wvalue res_json;
    std::cout << "[DEBUG] USER_ID: " << user_id<< std::endl;
    try {
        pqxx::work txn(c);

        pqxx::result r = txn.exec_params(
              "SELECT id, title, description, start_time, end_time
FROM calendar WHERE user_id = $1",
            user_id
        );

        crow::json::wvalue::list events;

        for (const auto& row : r) {
            crow::json::wvalue ev;
            ev["id"] = row["id"].as<int>();
            ev["title"] = row["title"].as<std::string>();
             ev["description"] = row["description"].is_null() ? "" :
row["description"].as<std::string>();
            ev["start_time"] = row["start_time"].as<std::string>();
            ev["end_time"] = row["end_time"].as<std::string>();
            events.emplace_back(std::move(ev));
        }

        res_json["events"] = std::move(events);
    }
    catch (const std::exception& ex) {
        res_json["error"] = ex.what();
    }

    return crow::response{res_json};
});
```

```
}
```

GoogleAuth.cpp

```
#include "googleAuth.hpp"

   // Callback for libcurl
size_t curl_write_callback(char* ptr, size_t size, size_t nmemb,
void* userdata) {
    std::string* response = static_cast<std::string*>(userdata);
    response->append(ptr, size * nmemb);
    return size * nmemb;
}

// Function to perform POST request with libcurl and get JSON
response
nlohmann::json post_json(const std::string& url, const
nlohmann::json& body) {
    CURL* curl = curl_easy_init();
      if (!curl) throw std::runtime_error("Failed to initialize
CURL");

    std::string response_data;
       struct curl_slist* headers = curl_slist_append(nullptr,
"Content-Type: application/json");

    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, body.dump().c_str());
                curl_easy_setopt(curl,       CURLOPT_WRITEFUNCTION,
curl_write_callback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response_data);

    CURLcode res = curl_easy_perform(curl);
    curl_easy_cleanup(curl);
    curl_slist_free_all(headers);
```

```
    if (res != CURLE_OK)
        throw std::runtime_error("CURL request failed");

    return nlohmann::json::parse(response_data);
}

// Setup Crow routes for Google OAuth2
void setup_google_oauth_routes(crow::SimpleApp& app,
pqxx::connection& db) {
    const std::string client_id =
"839021723800-m5cadpqru72rr85mhv04gevj7ue3u500.apps.googleusercontent.com";
    const std::string client_secret =
"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";
    const std::string redirect_uri =
"http://localhost:4200/oauth2callback";

    // 🔹 Route 1: Redirect user to Google login
    CROW_ROUTE(app, "/api/auth/google")
    ([client_id, redirect_uri]() {
        std::string auth_url =
            "https://accounts.google.com/o/oauth2/v2/auth?"
            "client_id=" + client_id +
            "&redirect_uri=" + redirect_uri +
            "&response_type=code"
            "&scope=openid%20email%20profile";

        crow::response res;
        res.redirect(auth_url);
        return res;
    });

    // 🔹 Route 2: Handle Google callback
    CROW_ROUTE(app, "/oauth2callback")
```

```
        ([client_id,    client_secret,    redirect_uri,    &db](const
crow::request& req) {
        auto code = req.url_params.get("code");
        if (!code)
                return  crow::response(400,  "Missing  authorization
code");

        try {
            // Exchange code for access token
            nlohmann::json token_request = {
                {"code", code},
                {"client_id", client_id},
                {"client_secret", client_secret},
                {"redirect_uri", redirect_uri},
                {"grant_type", "authorization_code"}
            };
                                        auto     token_response     =
post_json("https://oauth2.googleapis.com/token", token_request);
                                    std::string    access_token    =
token_response.at("access_token");

            // Fetch user info from Google
            nlohmann::json user_info;
            {
                CURL* curl = curl_easy_init();
                std::string user_response;
                                    struct   curl_slist*   headers   =
curl_slist_append(nullptr,    ("Authorization:    Bearer    "    +
access_token).c_str());

                                curl_easy_setopt(curl,   CURLOPT_URL,
"https://www.googleapis.com/oauth2/v2/userinfo");
                curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
                        curl_easy_setopt(curl,  CURLOPT_WRITEFUNCTION,
curl_write_callback);
```

```
                        curl_easy_setopt(curl,  CURLOPT_WRITEDATA,
&user_response);
                CURLcode res = curl_easy_perform(curl);
                curl_easy_cleanup(curl);
                curl_slist_free_all(headers);

                if (res != CURLE_OK)
                     return crow::response(500, "Failed to fetch user
info");

                user_info = nlohmann::json::parse(user_response);
            }

            // Optionally save user to database
            {
                pqxx::work txn(db);
                txn.exec_params(
                     "INSERT INTO users (username, email) VALUES ($1,
$2) ON CONFLICT(email) DO NOTHING",
                    user_info["name"].get<std::string>(),
                    user_info["email"].get<std::string>()
                );
                txn.commit();
            }

            return crow::response{user_info.dump(4)};
        } catch (const std::exception& e) {
             return crow::response(500, std::string("OAuth error: ")
+ e.what());
        }
    });
}
```