Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ЗАСТОСУНКУ НА C++

**Текстова частина до курсової роботи
за спеціальністю „Комп'ютерні науки"**

Керівник курсової роботи
ст. викладач Вовк Н.Є.
(прізвище та ініціали)

_____
(підпис)

"____" _____ 2021 р.

Виконав студент Вакуленко С.С.
(прізвище та ініціали)

"____" _____ 2021 р.

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
Доцент., к. ф.-м. н. О.П.Жижерун
_____
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Вакуленко Сергію Сергійовичу факультету інформатики 3 курсу

ТЕМА: Розробка клієнт-серверного застосунку на C++

Зміст ТЧ
1.Індивідуальне завдання
2.Календарний план
3.Перелік умовних позначень
4.Вступ
5.Постановка завдання курсової роботи
6.Теоретичні відомості
7.Огляд створеного додатку та аналіз результатів
8.Висновки
9.Список використаний джерел

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримала _____
(підпис)

Тема: Розробка клієнт-серверного застосунку на C++

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	25.10.2020	
2.	Огляд літератури за темою роботи.	28.12.2020	
3.	Виконання аналізу актуальності теми	18.01.2021	
4.	Розробка програмного застосунку	05.04.2021	
5.	Написання пояснювальної роботи.	03.05.2021	
6.	Попередня демонстрація проекту науковому керівнику	03.05.2021	
7.	Створення презентації	15.05.2021	
8.	Захист курсової роботи	З 24.05.2021	

Студент _____Вакуленко С.С,_____ _____

Керівник _____Вовк Н.Є._____ _____

“_____”_____

Зміст

Перелік умовних позначень

1. БД – база даних.
2. СКБД - система керування базами даних.
3. Token Ring - це топологія локальної мережі (LAN), яка надсилає дані в одному напрямку за певною кількістю місць за допомогою маркера.
4. Мейнфрейми - це тип комп’ютерів, який, як правило, відомий своїми великими розмірами, обсягом пам’яті, обробною потужністю та високим рівнем надійності. Вони в основному використовуються великими організаціями для критично важливих програм, що вимагають великих обсягів обробки даних.
5. Windows Forms - технологія інтелектуальних клієнтів для NET Framework, яка являє собою набір керованих бібліотек, що спрощують виконання стандартних завдань. [6]

ВСТУП

Постійно зростаючий вплив клієнт-серверної архітектури для ведення та розширення бізнесу в Інтернеті створив високий попит на клієнт-серверні програми.
Якщо 20 років тому, клієнт-серверні застосунки створювали для великих корпоративних бізнесів, то зараз важко уявити навіть маленький магазин без власного сайту, де б клієнт міг придбати товар або послугу.
Як і бізнес процеси, програмування не стоїть на місці, тому з'являються нові мови та технології для розробки, покращення та розширення функціоналу застосунків.

На сьогоднішній день існує безліч мов програмування, які є популярнішими, дешевшими в розробці та простішими в розумінні за мову C++. Я вирішив дослідити актуальність цієї мови, пройти процес написання клієнт-серверного застосунку, використовуючи її, та дізнатися чому велика кількість програмістів надає перевагу іншим технологіям розробки, а для виконання яких задач, C++ досі не має конкурентів.

Мета курсової роботи - розібратись у перевагах та недоліках C++ при створенні клієнт-серверного застосунку. Оцінити складність написання серверу, проблеми з якими можуть зіткнутись програмісти під час реалізації, ефективність коду у порівнянні з іншими мовами програмування.

Завданням курсової роботи є розробка застосунку, за допомогою якого можна порівнювати швидкість роботи серверів написаних на різних мовах і в подальшому аналізувати їх ефективність.

Застосунок розроблений на мовах C++, C#, Java та .NET Framework. Робота з базою даних відбувалась із MS SQL Server 2019. Взаємодія клієнта та сервера відбувається на основі протоколу TCP. Написання коду на C++, C# та .NET здійснювалось у Microsoft Visual Studio 2019, а на Java – в Intellij Idea ultimate edition.

Робота складається зі вступу, трьох основних розділів, кожен з яких містить підрозділи, висновків та списку використаної літератури.

Перший розділ присвячений аналізу вибраної теми, формулюванню та поставці задачі.

У другому розділі описуються принцип роботи клієнт-серверної архітектури, переваги\недоліки та її альтернативи. Багато уваги приділено відмінностям мови, перевагам\недолікам та актуальності.

У третьому розділі описується реалізація проекту. Опис користувацького інтерфейсу, використання бібліотек та детальний опис функціоналу.

РОЗДІЛ 1: ПОСТАНОВКА ЗАВДАННЯ КУРСОВОЇ РОБОТИ

1.1. Постановка завдання

Створити клієнт-серверний додаток з сервером на C++, та серверами, написаними на сучасніших мовах програмування, сімейства «C», а саме Java та C#.

В процесі створення необхідно:
- порівняти витрачений на розробку час
- проаналізувати проблеми та вади мови
- визначити ефективність (швидкидкість обробки запитів)
- оцінити можливості та складності розширення

Усі три сервери, повинні мати однаковий функціонал для справедливого порівняння та можливості оцінювання швидкості їх роботи. Також необхідно розробити клієнтську частину, на одній з мов програмування, для формування пакетів, які будуть обробляти сервери. На клієнті також необхідно заміряти час обробки запиту.

РОЗДІЛ 2: ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Клієнт-серверна архітектура

У сучасному обчислювальному світі система клієнт-сервер стала популярною, оскільки вона використовується практично щодня для різних додатків. Деякими стандартизованими протоколами, які використовують клієнти та сервери для взаємодії між собою, є:

Протокол передачі файлів (**FTP**)

Простий протокол передачі пошти (**SMTP**)

Протокол передачі гіпертексту (**HTTP**).

Таким чином, клієнт-серверну систему можна визначити, як програмну архітектуру, що складається з *клієнта* і *сервера*, за допомогою якої клієнти надсилають запити, тоді як сервер відповідає на надіслані запити.

Клієнт-сервер забезпечує міжпроцесовий зв’язок, оскільки він передбачає обмін даними як від клієнта, так і від сервера, завдяки чому кожен із них виконує різні функції.

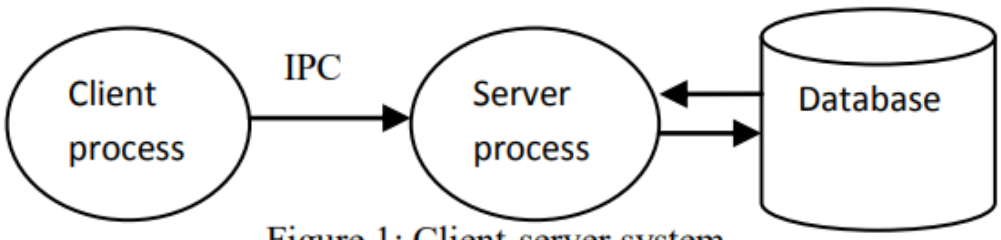


Figure 1: Client-server system

Клієнт-серверна архітектура розподіляє обов'язки з обробки даних між клієнтами, які, як правило, є ПК та сервером, робочою станцією високого класу або «мейнфреймом». ПК мають значну обчислювальну потужність і тому здатні брати необроблені дані, що повертаються сервером, і

форматувати результат для виведення. Прикладні програми та процесори запитів можна зберігати та виконувати на ПК.

Мережевий трафік зводиться до запитів на маніпуляції з даними, надісланих з ПК на сервер БД, і необроблених даних, що повертаються в результаті цього запиту. Результатом є значно менший мережевий трафік і теоретично краща продуктивність.

Сучасні архітектури клієнт/сервер обмінюються повідомленнями через локальні мережі. Хоча кілька старих локальних мереж Token Ring все ще використовуються, більшість сучасних локальних мереж базуються на стандартах Ethernet.

Клієнт-серверна архітектура схожа на традиційну централізовану архітектуру тим, що СУБД знаходиться на одному комп'ютері. Багато сучасних «мейнфреймів» насправді функціонують як великі швидкі сервери тому що необхідність обробляти великі масиви даних все ще існує, проте місце деяких обробок змінилося.

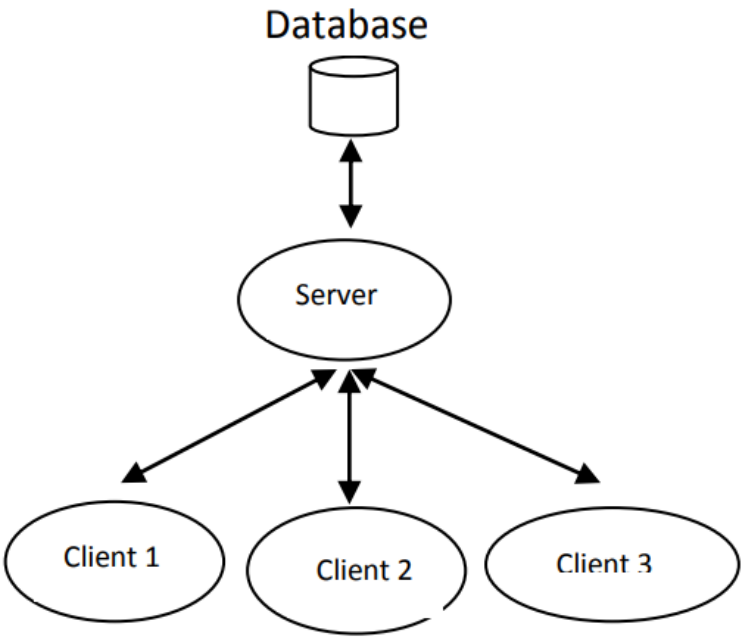


Figure 2: Interprocess communication among client and server

2.1.1 Переваги \ недоліки клієнт-серверної архітектури

Переваги:

- Покращений обмін даними.

Дані зберігаються за допомогою звичайних бізнес-процесів і маніпулювання на сервері доступне для призначених користувачів (клієнтів) через авторизований доступ, тобто дозволяє спростити обмін ресурсами від клієнта до серверів.

- Інтеграція служб.

Кожен клієнт отримує можливість отримати доступ до корпоративної інформації через інтерфейс робочого столу, усуваючи необхідність входу в термінальний режим або процесор.

- Спільні ресурси між різними платформами.

Додаток, що використовується для моделі клієнт-сервер, створюється незалежно від апаратної платформи або технічного досвіду відповідного програмного забезпечення (програмного забезпечення операційної системи), що забезпечує відкрите обчислювальне середовище, змушуючи користувачів отримувати послуги клієнтів та серверів (бази даних, програми та послуги зв'язку).

- Можливість обробки даних, незважаючи на місцезнаходження.

Користувачі клієнт-сервера можуть безпосередньо входити в систему, незважаючи на розташування або технологію процесорів.

- Простота обслуговування.

Клієнт-серверна архітектура - це розподілена модель, що представляє розподілені обов'язки між незалежними комп'ютерами, інтегрованими в мережу. Тому його легко замінити, відремонтувати, оновити та перенести сервер, тоді як клієнт залишається незмінним. Ця несвідома зміна називається інкапсуляцією.

- Безпека.

Сервери мають кращий контроль доступу та ресурсів, щоб гарантувати, що лише авторизовані клієнти можуть отримувати доступ до даних та керувати ними, а оновлення серверів ефективно адмініструються.

Недоліки:

- Перевантажені сервери.

Коли на сервер надходять часті одночасні клієнтські запити, сервер сильно перевантажується, утворюючи перевантаження трафіку.

- Вплив централізованої архітектури.

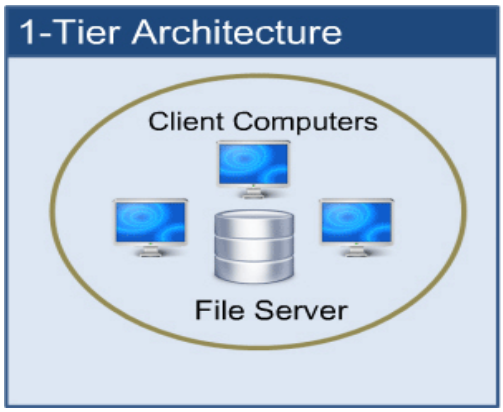
У випадку критичної відмови сервера, клієнтські запити не виконуються. Таким чином технологія клієнт-сервер втрачає надійність мережі.

Взявши до уваги всі вище зазначені факти, можемо стверджувати, що переваг більше ніж недоліків, а це свідчить проте наскільки важливою та необхідною є технологія клієнт-сервер.

2.1.2 Класифікація серверів

Однорівнева архітектура.

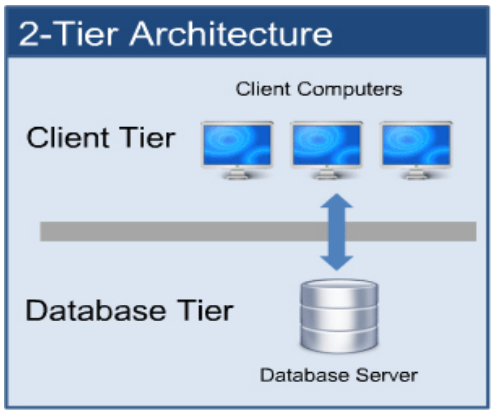
В однорівневій архітектурі всі параметри конфігурації клієнт/сервер: середовище користувацького інтерфейсу, логіка даних та логічна система маркетингу - існують в одній системі. Ці типи сервісів надійні, але з ними дуже складно впоратися, тому, що вони містять всі дані з різною дисперсією, які призначені для реплікації всієї роботи. Ця архітектура також містить різні рівні.



Наприклад - Рівень презентації, бізнесу, доступу до даних з використанням єдиного програмного пакета. Всі дані зберігаються на локальному комп'ютері. Деякі додатки, керують усіма трьома рівнями, наприклад MP3-плеєр, MS Office; але ці типи додатків представлені у вигляді додатків з однорівневої архітектурою.

Дворівнева архітектура.

Дворівнева архітектура забезпечує найкраще середовище клієнт/сервер, яке допомагає зберігати інтерфейс користувача в клієнтській системі, а вся база даних зберігається на серверній машині. Бізнес-логіка та логіка бази даних існують на клієнтському сервері, але їх потрібно підтримувати. Коли логіка даних та бізнес збираються на клієнтському боці, це називається "архітектура тонкого серверного клієнта". Але якщо бізнес-логіка та логіка даних управляються на серверній машині, то це відомо як "архітектура товстого клієнтського сервера".



У цій архітектурі клієнтські та серверні машини пов'язані безпосередньо інтеграцією, тому що якщо клієнт запускає будь-який вхід для серверного терміналу, тоді між ними не повинно бути проміжних. Отже, він забезпечує вихід з найшвидшими темпами та ігнорує непорозуміння між іншими клієнтами. Прикладом, де використовується дворівнева архітектура, може слугувати програма онлайн-бронювання квитків.

Переваги дворівневої архітектури:

- Просте проектування всіх додатків;
- Максимальне задоволення користувачів;
- Впровадження однорідного середовища;
- Найкраща продуктивність;

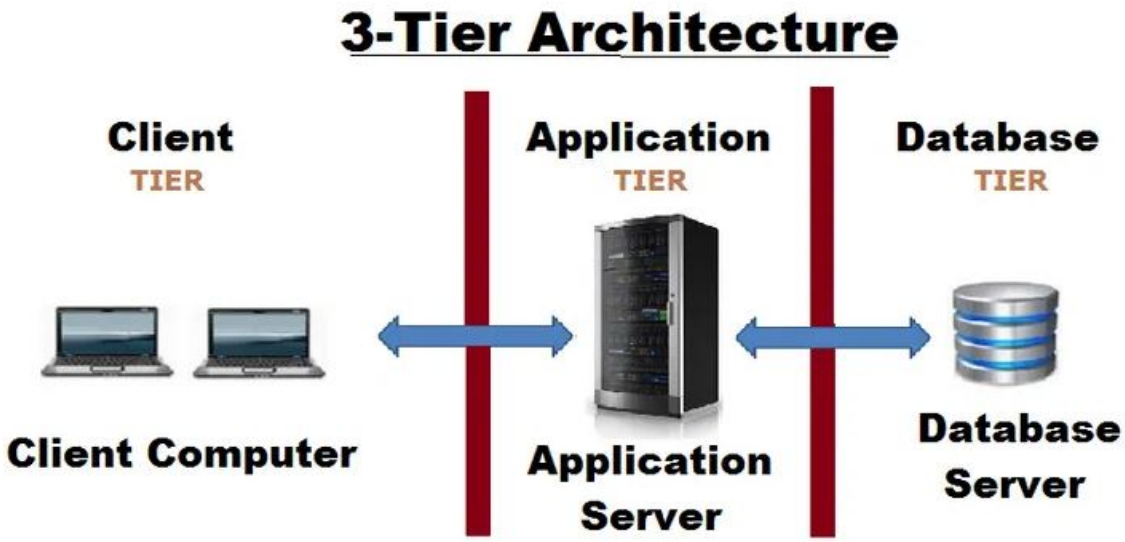
Обмеження дворівневої архітектури:

- Низька продуктивність, через збільшення кількості підключень кожного користувача;
- Менша безпека;
- Усі клієнти повністю залежать від бази даних виробника;
- Менша портативність, що означає, що ця архітектура повністю залежить від конкретної бази даних;

Трирівнева архітектура.

У 3-рівневій архітектурі необхідне проміжне програмне забезпечення тому, що, якщо клієнтська машина відправляє запит на сервер, то спочатку цей запит приймається на проміжному рівні, а тільки потім цей запит надходить на сервер. Таким чином, спочатку відповідь сервера надходить на середній рівень, потім він надходить на клієнтську машину. Вся логіка даних і бізнес-логіка зберігаються в проміжному програмному забезпеченні. За рахунок

використання проміжного програмного забезпечення для підвищення його гнучкості і забезпечення відмінної продуктивності.



Трирівнева архітектура поділяється на 3 рівні, такі як рівень презентації (рівень клієнта), рівень додатків (рівень бізнесу) та рівень бази даних (рівень даних). Клієнтська машина обробляє рівень презентації, рівень додатків контролює рівень додатків, і нарешті серверна машина піклується про рівень бази даних.

Переваги трирівневої архітектури:

- Цілісність даних;
- Покращена безпека дворівневої архітектури;
- Структура бази даних захована;

Обмеження трирівневої архітектури:

- Складніше спілкування між клієнтом та сервером, оскільки використовується проміжне програмне забезпечення;

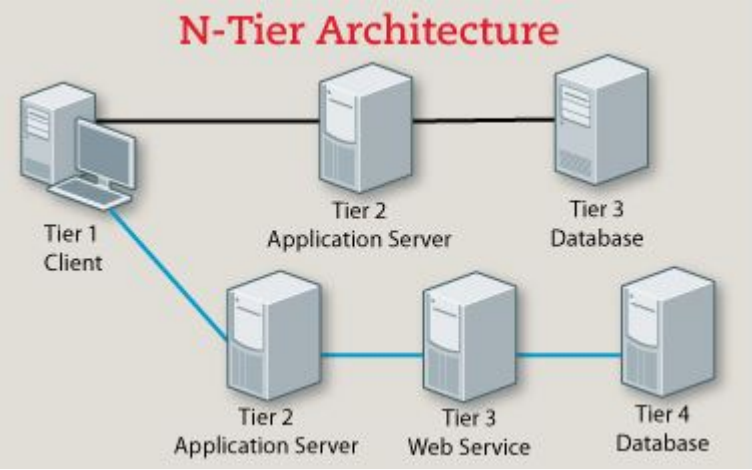
Багаторівнева архітектура.

Багаторівнева архітектура - це масштабована форма трирівневої архітектури. У цій архітектурі цілі презентації, обробка додатків та функції управління даними ізольовані один від одного.

Перевага полягає в тому, що така архітектура забезпечує гнучкі та багаторазові додатки;

Обмеження:

Оскільки така архітектура використовує складну структуру, компонетування рівнів, - її важче реалізувати;



2.1.3 Приклади клієнт-серверної архітектури

- Веб-сервери.

Веб-сервер, як високопродуктивна комп'ютерна система, яка може розміщувати кілька веб-сайтів. На цьому сервері встановлюються різні типи програмного забезпечення веб-сервера. Наприклад, як Apache або Microsoft IIS, що забезпечує доступ до розміщених кількох веб-сайтів в Інтернеті, і ці сервери пов’язані з Інтернетом через швидкісне з’єднання, що забезпечує надвисоку швидкість передачі даних.

- Поштові сервери.

Сервери електронної пошти допомагають надсилати та отримувати всі електронні листи. Деякі програмні засоби запускаються на поштовому сервері, що дозволяє адміністратору створювати та обробляти всі облікові записи електронної пошти для будь-якого домену, розміщеного на сервері. Поштові сервери використовують деякі протоколи для надсилання та отримання електронних листів, таких як SMTP, IMAP та POP3. Протокол SMTP допомагає запускати повідомлення та керує усіма вихідними запитами електронної пошти. IMAP та POP3 допомагають отримувати всі повідомлення та обробляти всі вхідні листи.

- Файлові сервери.

Файловий сервер – це система, що дозволяє користувачам отримувати доступ до всіх файлів. Він працює як централізоване місце зберігання файлів, і до нього можуть отримати доступ кілька термінальних систем.

- DNS.

DNS розшифровується як «Сервер доменних імен» (“Domain Name Server“), і він має величезну базу даних різних типів загальнодоступних IP-адрес, і вони зв’язуються зі своїми хост іменами.
Ці типи серверів допомагають доставляти всі ресурси (наприклад, файли, каталоги, спільні пристрої, такі як програми та принтери) на клієнтський термінал, такі як ПК, смартфони, КПК, ноутбуки, планшети тощо.

2.1.4 Інші принципи взаємодії

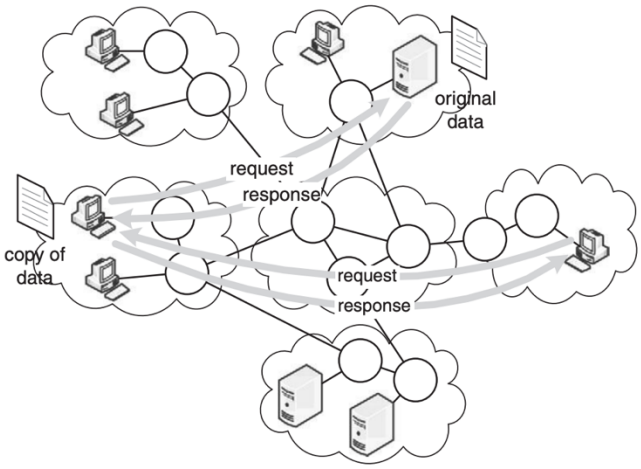
Окрім клієнт-серверної архітектури також існує кілька інших підходів щодо розподілу інформації по мережі.

Peer-to-peer networking (P2P)

Це розподілена архітектура додатків, яка розділяє завдання або робочі навантаження між вузлами.

P2P поєднують в собі ролі клієнта та сервера в кожному з вузлів однорангової мережі. Замість розповсюдження інформації з одного централізованого сервера, всі вузли працюють в якості серверів, до яких інші вузли можуть підключатися.

Використовуючи відповідну контрольну інформацію, вузол визначає, до якого іншого вузла підключитися, щоб отримати певну інформацію.[2]

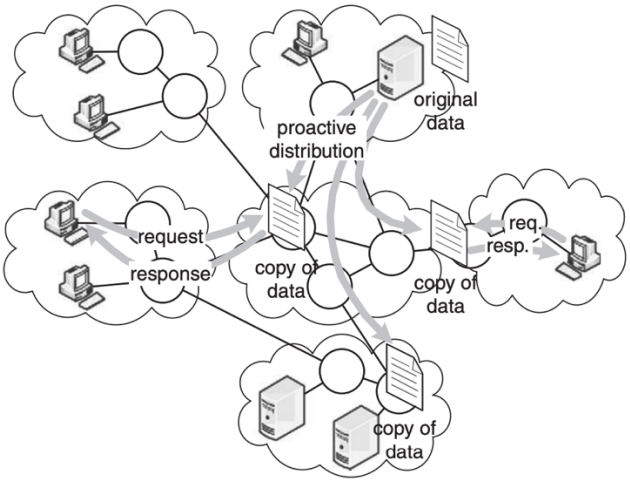


Content delivery networks (CDN)

Мережа доправлення контенту – це географічно розподілена мережа проксі-серверів та їх центрів обробки даних.

Ініціативний розподіл вмісту дозволяє клієнтам отримати доступ до копій вмісту, розташованих ближче географічно. Таким чином, можна досягти кращої продуктивності доступу, ніж при доступі до оригіналу сервера.

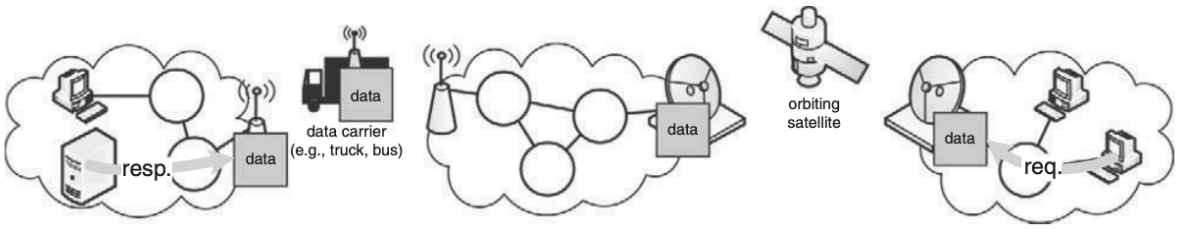
Використання розповсюдження контенту вимагає, щоб мережа підтримувала механізми, що дозволяють перенаправляти запит на локальну копію. [2]



Delay-tolerant networking (DTN)

Мережі, стійкі до затримки, складаються з мережевих систем які не можуть забезпечити постійне з'єднання. Цей підхід прагне вирішити технічні проблеми в неоднорідних мережах, які можуть мати відсутність постійного мережевого зв’язку. Домени таких додатків включають транспортні мережі, де транспортні засоби можуть бути відключені на деякий час. Необхідні принципові зміни у функціональності мережевих систем для того, щоб вузли зберігали дані протягом потенційно значної кількості часу.

Прикладами таких мереж є мережі, що працюють у мобільному або екстремальному наземному середовищі, або мережі в космосі. [2]



2.2 Вибір та аналіз мови розробки застосунків

Вибір мови програмування є важливим фактором під час створення додатків. Існує безліч різних мов програмування, кожна з яких є унікальною за своєю сферою використання та має ряд переваг та недоліків. Вибір мови буде залежати від предметної області, можливостей розробників, фінансів, кінцевої цілі застосунку та багатьох інших факторів. Вивчення сильних і слабких сторін мови є ключовим фактором для забезпечення ефективності та масштабування програми. Не існує однозначної відповіді, яка мова найкраща або, яка найкраще підходить для розробки серверу або клієнтської частини.

Визначивши переваги та недоліки можна вибрати мову та технологію, яка буде підходити найбільше.

2.2.1 Порівняння C++, Java та C#

Зазвичай додатки створені на C ++ більш складні.

У 1999 році Phipps, G. [1], дослідив, що код на C ++ є більш схильним до помилок, час розробки триваліший та більше помилок на хвилину, ніж Java.

Критерій порівняння	Java	C++	C#
Синтаксис	Зберігає візуальну сумісність з мовами C і C ++. Проте сильно відрізняється від них. В синтаксисі Java велику кількість засобів визначено необов’язковими, в результаті чого, їх було видалено.	Зберігає максимально можливу сумісність із синтаксисом мови C.	Зберігає максимально можливу сумісність із синтаксисом мови C.

Виконання програми	Спочатку відбувається компіляція коду в проміжний, а вже потім його буде інтерпретовано в машинний.	Код одразу перетворюється в машинний.	Код компілюється в проміжний, а потім JIT-компілятор перетворює його на машинний.
Управління ресурсами	У цій мові присутня функція прибиральника сміття, який стежить лише за пам’яттю. Йому для роботи потрібні системні ресурси, що значно знижує ефективність виконання програм.	Працює за принципом «захоплення ресурсів шляхом ініціалізації». Цей принцим дозволяє ресурсам асоційоватись із потрібним об'єктом і автоматично видалятися із пам’яті, якщо цей об'єкт буде знищено.	Прибиральнимк сміття керує виділенням та звільненням пам’яті в проекті. Окрім автоматичного прибирання сміття, C# містить клас System.GC, що надає програмісту декілька функцій роботи з пам’яттю вручну.
Вказівники	Покажчики відсутні.	Присутня можливість працювати із вказівниками на низькому рівні.	Вказівники використовуються розробниками лише в крайньому випадку. Зазвичай, це відбувається задля оптимізації програм.
Парадигма програмування	Об’єктно-орієнтована	Об’єктно-орієнтована та процедурна	Об’єктно-орієнтована
Препроцесор	Оскільки програмісти, що користувалися в своїй роботі	Препроцесор використовується у випадках, коли потрібне	Визначено декілька директив препроцесора,

	препроцесором мали багато проблем з цим, розробники мови вирішили прибрати його з мови повністю. Проте таким чином вони зазнали втрат у можливостях текстових замін в коді мовними засобами і всі можливості метапрограмуванн я.	ввімкнення визначень класів і функцій. Також, для підключення різноманітних бібліотек вихідного коду. І наостанок надає можливості метапрограмуван ня з використанням препроцесора.	що впливають на процес перетворення коду в машинний.

2.2.2 Актуальність мови

2.2.2.1 Статистика актуальних мов

Такі мови програмування, як C++ і C#, стабільно тримаються на своїй позиції вже більше року. Однак Java, в порівнянні з минулим роком, зменшилася в актуальності на 4.54%. Проте це не заважає їй лідирувати над C++ і C#. Нижче наведено таблицю рейтингу мов програмування, станом на травень 2021 року. Рейтинг відображає шанс на зменшення актуальності (у відсотках) та порівняння із травнем 2020 року. [8]

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	↑	Python	11.87%	+2.75%
3	2	↓	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%
6	6		Visual Basic	4.02%	-0.16%
7	7		JavaScript	2.45%	-0.23%
8	14	⇈	Assembly language	2.43%	+1.31%
9	8	↓	PHP	1.86%	-0.63%
10	9	↓	SQL	1.71%	-0.38%

2.2.2.2 Частота оновлення та масштабність їх нововведень

Java

Починаючи з 2018 року, Java стабільно оновлюється двічі на рік.

З кожним оновленням, Java додає великий список функцій. Наприклад одними із основних функцій, що було додано у Java 14, є:

- Відповідність шаблону для instanceof;
- Текстові блоки;
- Корисні виключення NullPointer;
- Розподіл пам'яті з урахуванням NUMA для G1;
- Видалено збирача сміття з паралельним знаком (CMS). [9]

C# за весь час його існування оновлювали 12 разів. В останній його версії було додано нововведення, які за об'ємом не поступаються Java. Ось деякі з них:

- Методи інтерфейсу за замовчуванням;
- Покращення відповідності шаблонів;
- Використання декларацій;
- Статичні локальні функції;
- Асинхронні потоки;

- Індекси та діапазони;
- Покращення інтерпольованих дослівних рядків.[10]

C++

Історію існування C ++ розділяють на два періоди:

- Ранній C ++ (1979-1991)
- Стандартний C ++

В останню версію C ++ було додано такі функції:

- Тестові макроси;
- Призначені ініціалізатори;
- Вилучено вимогу використовувати typename для усунення різниці між типами у багатьох контекстах;
- Сукупна ініціалізація за допомогою дужок;
- Модулі;
- Скорочені шаблони функцій[11]

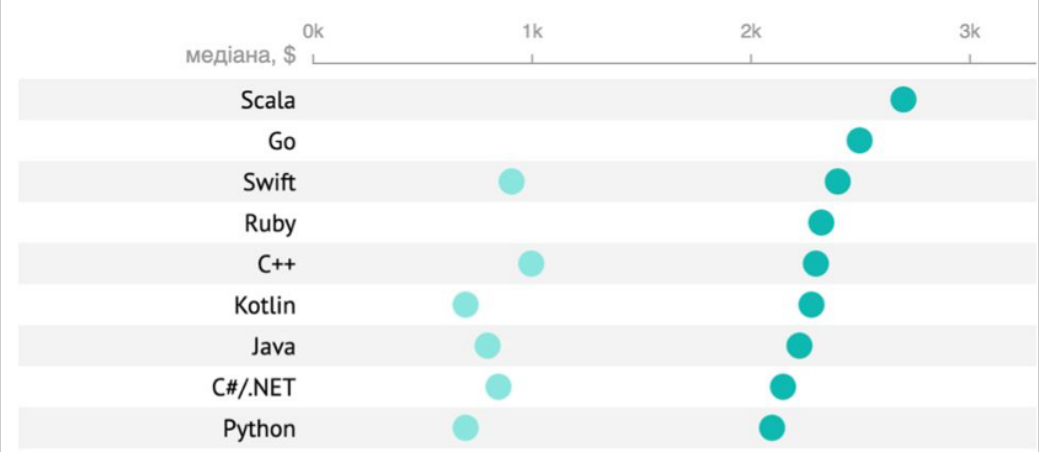
2.2.3 Вартість розробки та пошук спеціаліста

Важливим у розробці застосунку є пошук програмістів та вартість їх праці. За результатами досліджень стало відомо, що до п'ятірки найбільш затребуваних мов програмування виявився C++. Вона є важливою мовою програмування, що особливо корисна в контексті створення систем, які вимагають високого рівня продуктивності. Це пояснює те, чому ця мова знаходиться в списку найбільш високооплачуваних.

Також варто звернути увагу на той факт, що такі мови програмування як Java і C# оплачуються практично однаково, і цьому є логічне пояснення. І Java, і C# є об’єктно-орієнтованими, мають схожий синтаксис і складність для вивчення. Проте Java більше оплачується, все через її популярність

серед розробників. Статистка показує, що вона використовується приблизно на 10% частіше, ніж C#.[12]

Нище наведено діаграму, що описує медіану зарплат розробників станом на зиму 2021 року за версією платформи dou.



РОЗДІЛ 3: ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Обгрунтування вибору засобів розробки

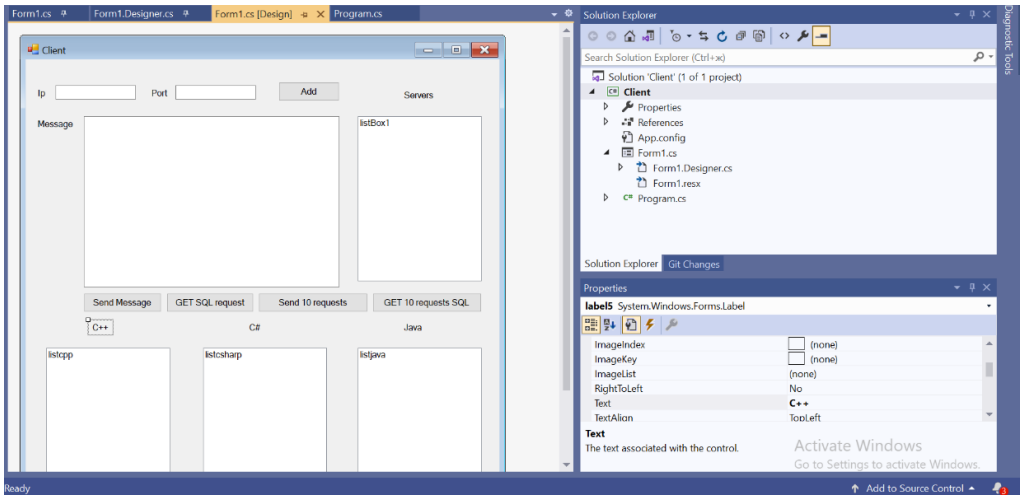
Оскільки Java виникла як альтернатива C++, а C# була запропонована корпорацією Microfost як мова, що поділяє принципи C++ та розширює її перевагами Java, то цікаво дослідити саме ці три мови й порівняти їхню ефективність при розробці серверної частини застосунку. При цьому процес розробки полегшується завдяки тому, що ці три мови мають подібний синтаксис.

MS SQL Server було обрано в якості СКБД, оскільки вона легко інтегрується до операційної системи Windows та середовища розробки Visual Studio.

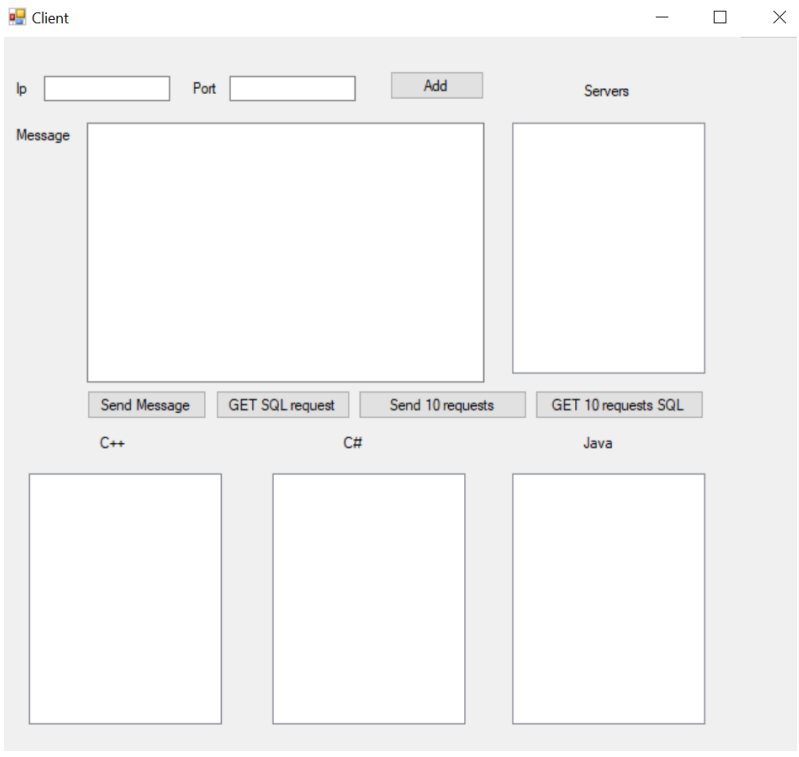
3.2 Створення клієнтської частини

3.2.1 Опис користувацького інтерфейсу

Користувацький інтерфейс було реалізовано за допомогою Windows Forms (.NET Framework). [6] Реалізація проста та зрозуміла. Полягає у переміщені елементів управління на форму і написання коду для реагування на дії користувача, такі як клацання миші або натискання клавіш. Після цього, автоматично генерується код інтерфейсу.




```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
1 reference
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.label2 = new System.Windows.Forms.Label();
    this.label1 = new System.Windows.Forms.Label();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.label3 = new System.Windows.Forms.Label();
    this.textBox3 = new System.Windows.Forms.TextBox();
    this.listBox1 = new System.Windows.Forms.ListBox();
    this.label4 = new System.Windows.Forms.Label();
```



Інтерфейс складається з:

- 3 текстових полів (System.Windows.Forms.TextBox) – для введення IP адреси, порту та тексту, який буде оброблятись серверами
- 4 елементів для відображення списку (System.Windows.Forms.TextBox) - відобрежння результатів обробки повідомлення\отримання даних з БД
- 5 кнопок:

“Send Message” – відправлення повідомлення введене у поле на сервер.

“Send 10 requests” – відправлення 10 повідомлень.

“Add” – додає IP та порт сервера в TextBox.

“GET SQL request” – SQL запит в базу.

“GET 10 request SQL” – 10 однакових запитів в базу.

3.2.2 Клієнт

В основі мережевих взаємодій по протоколам TCP і UDP лежать сокети. В .NET сокети представлені класом System.Net.Sockets, який надає низькорівневий інтерфейс для прийому і відправки повідомлень по мережі. [7]

Для надсилання даних, використовується метод Writer(), а для їх зчитування Read().

Надсилання даних на сервер.

```
//відправлення одного повідомлення
1 reference
private void button1_Click(object sender, EventArgs e){
    try
    {
        if (listBox1.SelectedIndex != -1)
        {
            TcpClient client = connects[listBox1.SelectedIndex]; //вибираємо підключення із існуючих, за допомогою listbox
            {
                var startTime = System.Diagnostics.Stopwatch.StartNew();  //починаємо відлік часу час

                NetworkStream stream = streams[listBox1.SelectedIndex]; // вибираємо потік із існуючих, за допомогою listbox
                byte[] data = Encoding.UTF8.GetBytes(textBox3.Text + "\r\n"); //конвертуємо строку повідомлення в байти

                stream.Write(data, 0, data.Length); //відправляємо на сервер
```

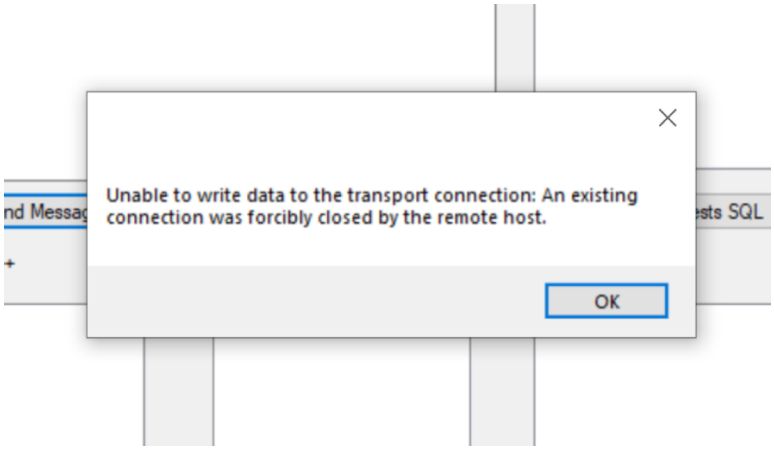
Отримання даних.

```
string[] str = listBox1.Items[listBox1.SelectedIndex].ToString().Split(':'); // Розділяємо рядок на IP та порт
switch (str[1]) //порівнюємо порти і додаємо в свій список
{
    case "6000":
        {
            byte[] data2 = new byte[256]; // масив байтів для отримання повідомлення з серверу

            int bytes = stream.Read(data2, 0, data2.Length); //отримання повідомлення з серверу

            string message = Encoding.UTF8.GetString(data2, 0, bytes); //перекодовуємо в строку

            var resultTime = startTime.Elapsed;  //закінчуємо відлік часу
            string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",
                resultTime.Hours, resultTime.Minutes, resultTime.Seconds, resultTime.Milliseconds);
            listcpp.Items.Add("MSG: " + elapsedTime);

            MessageBox.Show(message);
            break;
        }
    case "7000":
        {
            byte[] data2 = new byte[256];
```

Обробка винятків.

Якщо клієнт спробує відправити запит на сервер, до якого відсутнє підключення, то виводиться повідомлення, що з'єднання неможливе, а IP та порт видаляється зі списку “Servers”.

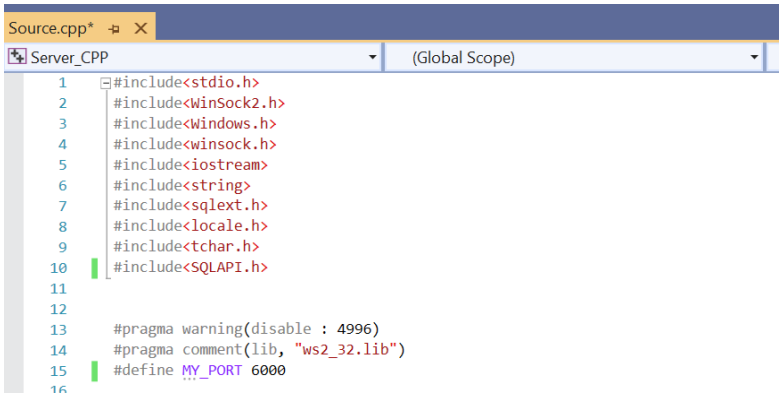
```
    }
    catch (Exception ex)
    {
        //якщо підключення недоступне, то виводимо повідомлення та видаляємо ip та порт
        MessageBox.Show(ex.Message);
        connects.RemoveAt(listBox1.SelectedIndex);
        streams.RemoveAt(listBox1.SelectedIndex);
        listBox1.Items.RemoveAt(listBox1.SelectedIndex);

    }

}
```



3.3 Сервер на C++

Для створення серверу, я використав інтерфейс Winsock, який спрощує розробку мережевих застосунків під Windows. Winsock являє собою інтерфейс між додатком та транспортним протоколом, що виконує передачу даних.

Для того, щоб працювати з Winsock необхідно підключити всі бібліотеки та .h файли.



Наступний крок – ініціалізація. Для ініціалізації Winsock викликаємо функцію WSAStartup().

```
int main()
{
    setlocale(0, ""); // для виводу кирилиці в консоль
    char buff[1024]; // виділення пам*яті для буфера

    printf("TCP SERVER START\n");

    // перевірка чи змогли ми ініціалізувати бібілотеку WinSock2 з нашим сокетом та буфером
    // якщо ні, то повертаємо помилку

    if (WSAStartup(0x0202, (WSADATA*)&buff[0]))
    {
        // Помилка!
        printf("Error WSAStartup %d\n", WSAGetLastError());
        return -1;
    }
```

Створення основного засобу комунікації в Winsock- сокета (socket).

```
// Ініціалізація сокету
SOCKET mysocket;

// AF_INET - сокет Інтернета
// SOCK_STREAM - потоковий соке
// 0 - за замовчування для TCP протоколу

if ((mysocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    // Помилка!
    printf("Error socket %d\n", WSAGetLastError());
    WSACleanup(); // звільнення/очищення від ресурсів зайнятих Winsock
    return -1;
}


sockaddr_in local_addr;                 // описує сокет для роботи з протоколом
local_addr.sin_family = AF_INET;       // вказуємо тип сокету
local_addr.sin_port = htons(MY_PORT); // порт
local_addr.sin_addr.s_addr = 0;      // адресу
```

Функція bind() пов'язує локальну адресу з сокетом. Потрібна на не підключений сокет, і викликається пере прослуховуванням сокета, функцією listen().

```
// викликаємо bind для зв*язування
if (bind(mysocket, (sockaddr*)&local_addr, sizeof(local_addr)))
{
    // Помилка
    printf("Error bind %d\n", WSAGetLastError());
    closesocket(mysocket); // закриваємо сокет!
    WSACleanup();
    return -1;
}

// підключення до сокету
// розмір черги - 0x100
if (listen(mysocket, 0x100))
{
    // Помилка
    printf("Error listen %d\n", WSAGetLastError());
    closesocket(mysocket);
    WSACleanup();
    return -1;
}

printf("ochikuvani pidkluchenya...\n");
```

```
// Отримуємо повідомлення з черги

SOCKET client_socket;     // сокет для клієнта
sockaddr_in client_addr; //  адреса клієнта - заповнюється системою
```

Витягуємо перше з'єднання, за допомогою функціїї accept()[13], з черги та створюємо новий потік, для роботи в ньому з клієнтом.

```
// Отримуємо повідомлення з черги

SOCKET client_socket;     // сокет для клієнта
sockaddr_in client_addr; //  адреса клієнта - заповнюється системою

//розмір адреси клієнта
int client_addr_size = sizeof(client_addr);


// https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-accept
// Функція accept витягує перше з'єднання в черзі очікуваних з'єднань на сокетах

while ((client_socket = accept(mysocket, (sockaddr*)&client_addr, \
    & client_addr_size)))
{
    nclients++; // збільшуємо кількість клієнтів

    // витягуємо ім*я хоста

    HOSTENT* hst;
    hst = gethostbyaddr((char*)&client_addr.sin_addr.s_addr, 4, AF_INET);

    // в термінал виводимо ім*я та інформацію по клієнту
    printf("+%s [%s] new connect!\n",
        (hst) ? hst->h_name : "", inet_ntoa(client_addr.sin_addr));
    PRINTNUSERS

     // через CreateThread створюємо новий потік для роботи в ньому з клієнтом
        DWORD thID;
    CreateThread(NULL, NULL, SexToClient, &client_socket, NULL, &thID);
}

return 0;
```

Далі реалізована функція, яка працює з сокетами та відправляє відповідь клієнту. Для того, щоб надіслати дані використовується функція send. Клієнту відправляємо повідомлення “Server OK”. У випадку одного SQL запиту виводимо результат запиту в консоль сервера (перший випадок), там само і у випадку роботи з повідомленням (третій випадок).

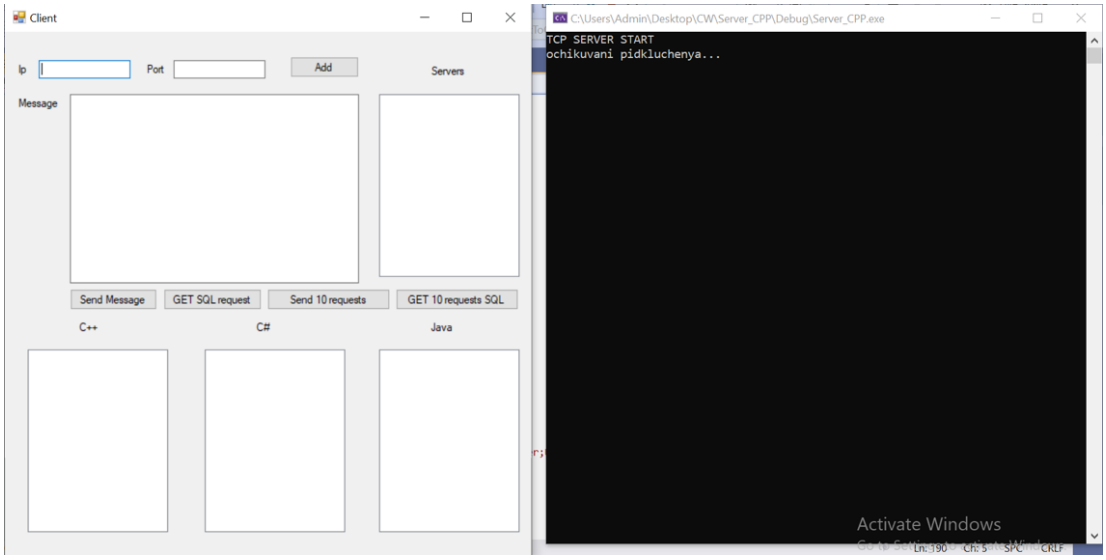
```
// ця функція буде викликатись лише в новому потоці для

DWORD WINAPI SexToClient(LPVOID client_socket){

    SOCKET my_sock;
    my_sock = ((SOCKET*)client_socket)[0];
    char buff[20 * 1024];


        // приймання рядка від клієнта та повернення її клієнтові
    int bytes_recv;
    // функція recv отримує дані з підключеного сокета
    while ((bytes_recv = recv(my_sock, &buff[0], sizeof(buff), 0)) &&
        bytes_recv != SOCKET_ERROR){

        buff[bytes_recv] = '\0';

        std::string test(buff, bytes_recv);
         if (test._Equal("sql\r\n"))
        {
            SAConnection con;
            con.Connect("User", "", "", SA_SQLServer_Client);
            SACommand cmd(
                &con,
                _TSA("SELECT * FROM person"));
            cmd.Execute();
            while (cmd.FetchNext())
            {
                SAString sName = cmd.Field(_TSA("name"));
                long nId = cmd.Field(_TSA("id"));
                printf("Name: %s, Id: %d \n", sName.GetMultiByteChars(), nId);

            }
            std::string send_arr = "Server OK";
            send(my_sock, send_arr.c_str(), sizeof(send_arr), 0);
        }

        else if (test._Equal("sql10\r\n"))
        {
            SAConnection con;
            con.Connect("User", "", "", SA_SQLServer_Client);
            SACommand cmd(
                &con,
                _TSA("SELECT * FROM person"));
            cmd.Execute();
            while (cmd.FetchNext())
            {
                SAString sName = cmd.Field(_TSA("name"));
                long nId = cmd.Field(_TSA("id"));

            }
            std::cout << "SQL 10 REQUESTS" << std::endl;
            std::string send_arr = "Server OK";
            send(my_sock, send_arr.c_str(), sizeof(send_arr), 0);
        }
        else
        {
            std::cout << test << std::endl;
            std::string send_arr = "Server OK";
            send(my_sock, send_arr.c_str(), sizeof(send_arr), 0);
        }

    }
```

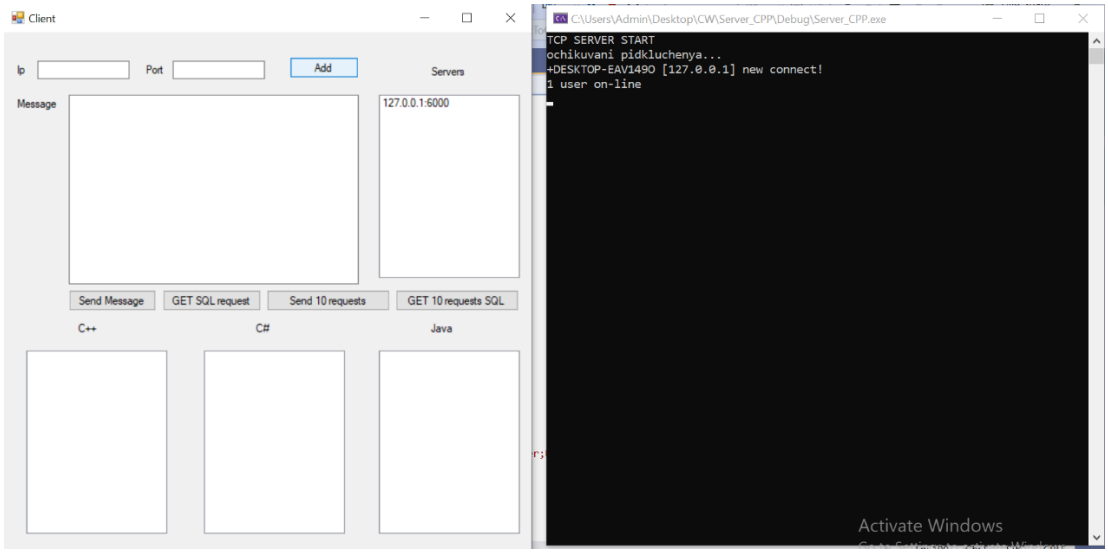
Останнім кроком є закриття з'єднання. Процедура закриття активного з'єднання відбувається за допомогою функції closesocket().

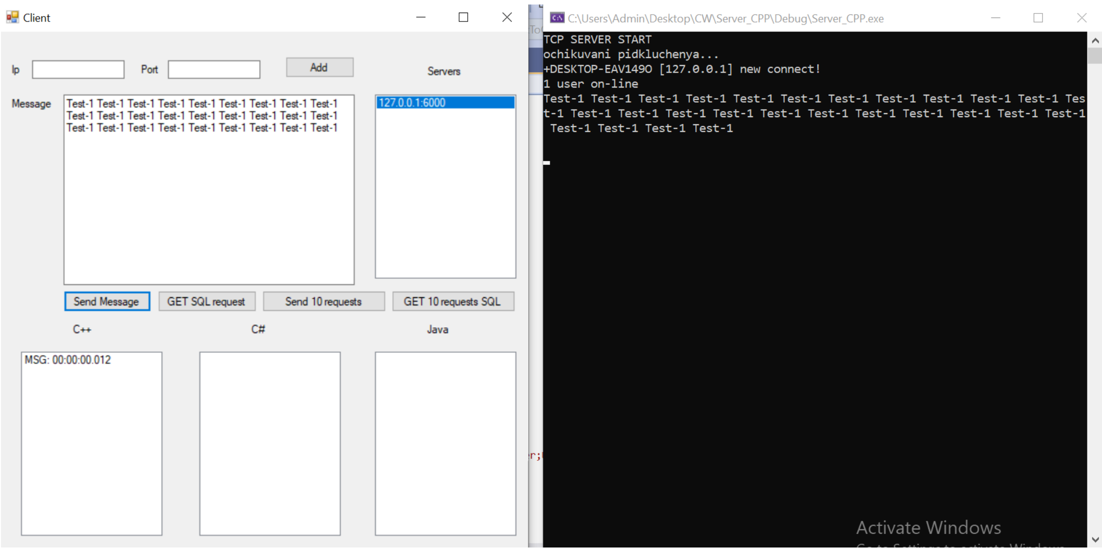
```
    }
    nclients--; // зменшуємо кількість активних клієнтів
    printf("-disconnect\n"); PRINTNUSERS

        // закриваємо сокет
        closesocket(my_sock);
    return 0;
```

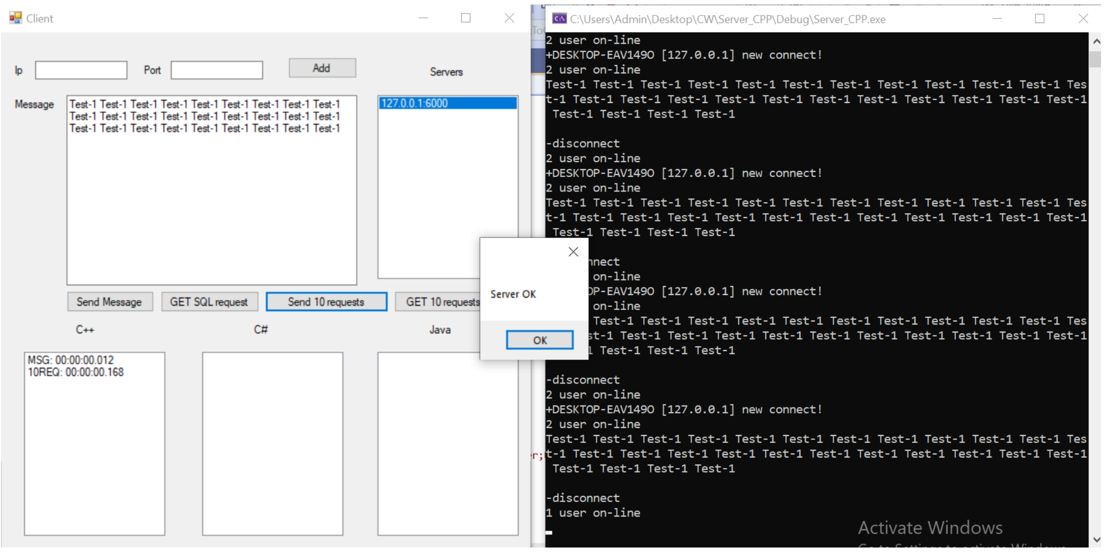
Підключення до БД відбувалось за допомогою класу підключення до СКБД SQLAPI++. [14]

Демонстрація роботи сервера









3.4 Сервер на C#

По-перше, у програмі відбувається створення tcp – клієнту, а також ініціалізація об’єкту клієнта:

```
3 references
public class ClientObject
{
    public TcpClient client;
    1 reference
    public ClientObject(TcpClient tcpClient)
    {
        client = tcpClient;
    }
```

По-друге, створюємо потрібний нам потік та ініціалізуємо його отриманим потоком нашого клієнта.
Наступним кроком є тримання повідомлення від стріму клієнта – створюється об’єкт StringBuilder та формується рядок.
Далі відбувається перевірка отриманого рядка, і у випадку, якщо він рівний «sql\r\n» або «sql10\r\n» відбувається формування connectionString – рядка, що містить усі дані, необхідні для підключення до бази даних «User», як-от назва серверу, бази даних та рядок Trusted_Connection, що означає Windows Authentication.

Наступним кроком, разом із формуванням рядка запиту в базу даних мовою SQL, відбувається сам процес підключення за допомогою створеної connectionString.

```
1 reference
public void Process()
{
    NetworkStream stream = null;
    try
    {
        stream = client.GetStream();
        byte[] data = new byte[64];
        while (true)
        {

            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            do
            {
                bytes = stream.Read(data, 0, data.Length);
                builder.Append(Encoding.UTF8.GetString(data, 0, bytes));
            }
            while (stream.DataAvailable);

            string message = builder.ToString();

            if(message.Equals("sql\r\n")|| message.Equals("sql10\r\n"))
            {
                string connectionString = @"Server=localhost;Database=User;Trusted_Connection=True;";
                string sql = "select * from [dbo].[person]";

                SqlConnection connection = new SqlConnection(connectionString);
```

По-друге, створюється блок using, що при завершенні коду, який в ньому знаходиться, звільняє область пам’яті від створених в ньому даних. В цьому блоці відбувається відкриття з’єднання із базою даних, зчитування даних з неї та відображення інформації в консоль користувача.
Наступним кроком є запис нашої змінної data у раніше створений потік.

```
            using (SqlCommand command = new SqlCommand(sql, connection))
            {
                connection.Open();
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        Console.WriteLine("id: {0} name: {1}", reader.GetInt32(0), reader.GetString(1));
                    }
                }
            }
        }
        else Console.WriteLine(message);

        data = Encoding.UTF8.GetBytes("Server OK");
        stream.Write(data, 0, data.Length);
    }
}
```

По-третє, у програмі відбувається прослуховування клієнта в новому потоці. Створення об’єкту TcpListener на порті "127.0.0.1", та, викликаючи у нього метод Start() починаємо слухати пакети на даному порті. Наступним кроком є створення нового потоку для обслуговування нового клієнту. Врешті-решт, у блоці finally, який викликається навіть у випадку виникнення виключення в результаті роботи програми, відбувається зупинка прослуховування.

```
0 references
class Program
{

    const int port = 7000;
    static TcpListener listener;
    0 references
    static void Main(string[] args)
    {
        try
        {
            listener = new TcpListener(IPAddress.Parse("127.0.0.1"), port);
            listener.Start();
            Console.WriteLine("Ochikuvania pidkluchenya...");

            while (true)
            {
                TcpClient client = listener.AcceptTcpClient();
                ClientObject clientObject = new ClientObject(client);

                // створюємо новий потік для роботи з клієнтом
                Thread clientThread = new Thread(new ThreadStart(clientObject.Process));
                clientThread.Start();
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        finally
        {
            if (listener != null)
                listener.Stop();
        }


    }
}
```

3.5 Сервер на Java

Для реалізації роботи із сокетами на Java, було використано такі бібліотеки:

java.net.ServerSocket - реалізує серверний сокет, що очікує запити, що відправляються. Також може відправляти у відповідь різні дані;

java.net.Socket – сокет, який використовується для зв’язку сервера та клієнта між собою;

Для здійснення роботи з базами даних використано microsoft.sqlserver.jdbc.SQLServerDataSource, що дає змогу встановлювати зв’язок зі сховищем даних SQL Server.

```
class Main {

    static ExecutorService executeIt = Executors.newFixedThreadPool( nThreads: 10);

    public static void main(String[] args) throws IOException {

        // стартуємо сервер на порту 8000 та ініціалізуємо змінну для обробки консольних команд з самого сервера
        try (ServerSocket server = new ServerSocket( port: 8000);
             BufferedReader br = new BufferedReader(new InputStreamReader(System.in))) {
            System.out.println("Server socket created, command console reader for listen to server commands");

            // заходимо в цикл, за умови що серверний сокет не закритий
            while (!server.isClosed()) {

                Socket client = server.accept();


                executeIt.execute(new MonoThreadClientHandler(client));
                System.out.print("Connection accepted.");
            }

            executeIt.shutdown();
        } catch (IOException e) {
            e.printStackTrace();
```

```
class MonoThreadClientHandler implements Runnable {

    public Socket clientDialog;
    private static String URL = "jdbc:sqlserver://localhost\\SQLEXPRESS:1434;DatabaseName=User;authenticationScheme=JavaKerberos;use
    public MonoThreadClientHandler(Socket client) {
        clientDialog = client;
    }

    @Override
    public void run() {

        try {
            // ініціалізуємо канали для спілкування в сокеті для сервера

            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(clientDialog.getOutputStream()));
            // читаємо з сокета
            DataInputStream in = new DataInputStream(clientDialog.getInputStream());
            BufferedReader d = new BufferedReader(new InputStreamReader(in));
            System.out.println("DataInputStream created");
            System.out.println("DataOutputStream  created");
```

```
        // починаємо діалог з підключений клієнтом в циклі, поки сокет не закрит клієнтом
        while (true) {
            try {
                System.out.println("Server reading from channel");

                String entry = null;
                entry = d.readLine();
                String finalEntry = entry;


                    System.out.println(finalEntry);


                    if (finalEntry != null) {
                        if (finalEntry.equalsIgnoreCase( anotherString: "sql") || finalEntry.equalsIgnoreCase( anotherString: "sql10")) {
                            try {

                                Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
                                SQLServerDataSource ds = new SQLServerDataSource();
                                ds.setServerName("localhost\\SQLEXPRESS");
                                ds.setPortNumber(1434);
                                ds.setIntegratedSecurity(true);
```

3.6 Недоліки та проблеми, що виникли під час розробки

Розробка C++ зайняла найбільше часу. Було багато проблем з зі зверненнями в БД. Довелось окремо скачувати та налаштовувати бібілотеку SQLAPI++, що також зайняло багато часу. Реалізація серверу на C++ найважка і необхідно написати на 30% більше коду ніж на C# або на 15% більше ніж на Java, щоб реалізувати один і той самий функціонал. При розробці серверу на Java, також виникла проблема з підключення до БД, адже з використанням бібліотеки, авторизація відбувається через акаунт SQL Sever.

3.7 Тестування програми і результати її виконання

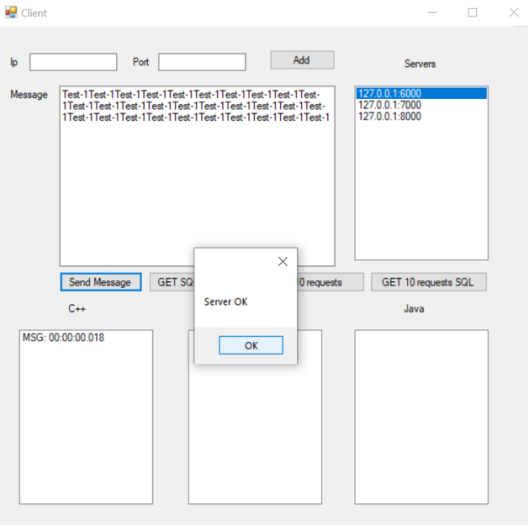
Для тестування необхідно додати сервери в список «Servers».

127.0.0.1:6000 – C++

127.0.0.1:7000 – C#

127.0.0.1:8000 – Java

Після успішної обробки запиту, сервер повертає повідомлення "Server OK", з клієнтського боку зупиняється таймер, а час записується у відповідну таблицю.



Перший тест – передача текстового повідомлення довжиною в 840 символів.

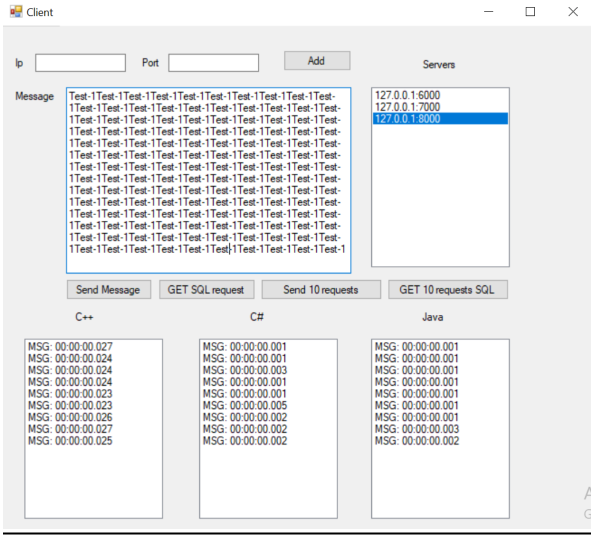
Кількість повторень: 9

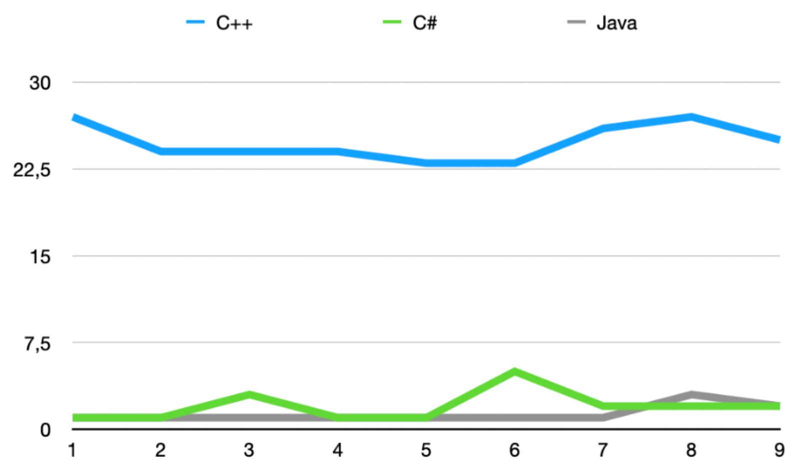
Результат:

*якщо результат 0, то виконання відбулось за менш ніж 1 Мілі секунду

Результати всіх трьох мов стабільні, без різкого зменшення або збільшення часу.

Найкращі та найстабільніші показники у Java. C++ займає останнє місце.





Другий тест – передача текстового повідомлення довжиною в 350 символів 10 разів.

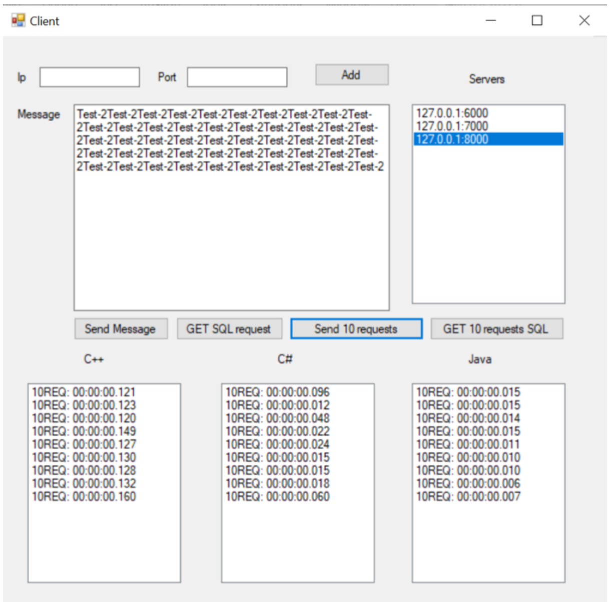
<u>Кількість повторень: 9</u>

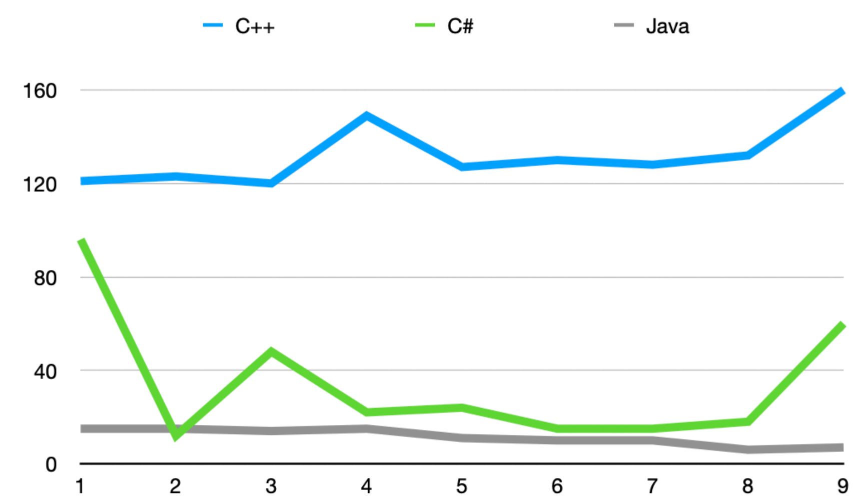
<u>Результат:</u>

В даному експерименті, Java також виявилась найстабільнішою та найшвидшою.

C# має схожі результати, але був не стабільний і декілька разів просідав по ефективності.

C++ займає останнє місце по швидкості, але його результати були стабільні, без значного просідання швидкості.





Третій тест – SQL запит

<u>База</u>: MS SQL Server

<u>Кількість повторень</u>: 5

<u>Заповнених рядків</u>: 9

<u>Результат:</u> на відміну від двох попередніх тестах, всі три мови показали майже однакові результати по швидкості.

Найменші коливання (найбільш стабільним) була мова C#.

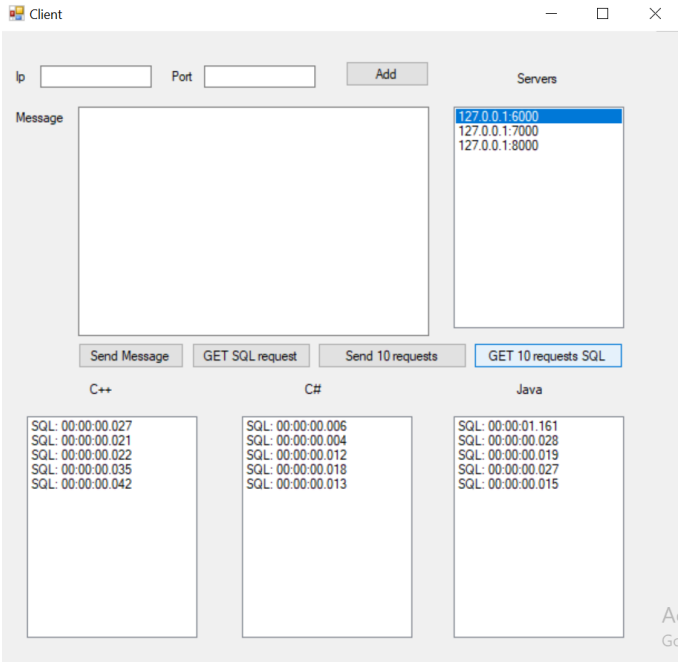
SQLQuery3.sql - DE...9O.User (new (63))*

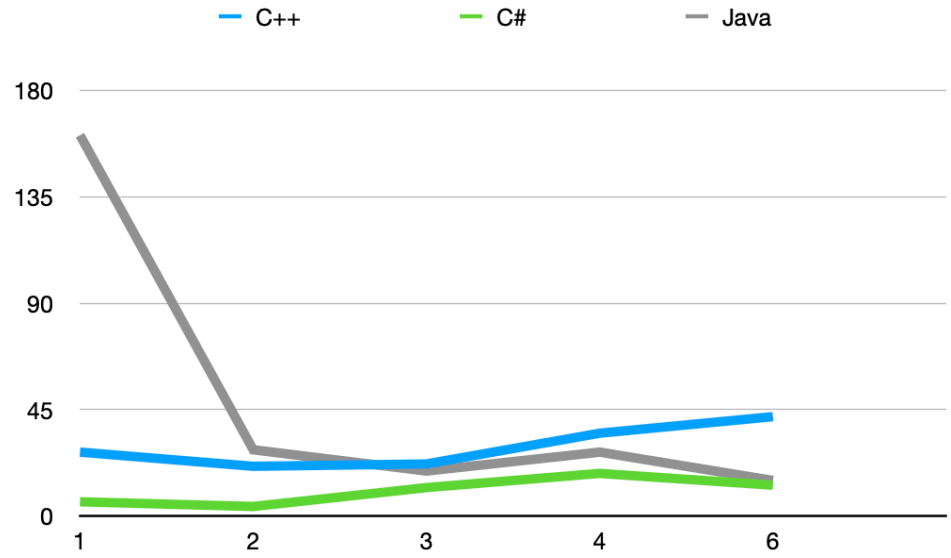
```
SELECT *
FROM [dbo].[person]
```

100 %

Results | Messages

	id	name	city
1	1	John	Kyiv
2	2	John	Kyiv
3	3	John	Kyiv
4	4	John	Kyiv
5	5	John	Kyiv
6	6	John	Kyiv
7	7	John	Kyiv
8	8	John	Kyiv
9	9	John	Kyiv





Висновки

В процесі розробки курсової роботи я проаналізував актуальність C++, Java та C#, їх популярність у сучасному світі, швидкість розробки клієнт-серверних застосунку, проблеми з якими стикаються програмісти і порівняв швидкість та стабільність їх роботи.

В результаті дослідження стало зрозуміло, що C++ - унікальна мова і Java або C#, не можуть замінити її. Досконале вивчення C++ та написання коду займає більше часу, а година програміста на C++ найдорожчка у порівнянні з Java та C#.

У програмній частині курсової роботи було створено клієнт-серверний застосунок, на основі якого зроблено висновки для дослідження. Сервери застосунку створенні на C++, C# та Java, а клієнт на .NET.

Серед проблем, з якими я зіткнувся під час роботи над курсовим проектом, стало те, що під час написання коду на C++ виникло багато проблем з помилками та підключенням бібліотек.

Написання частини проекту на C++ у мене зайняло на 25-30% більше, ніж на інших мовах. Також великим недоліком C++ в порівнянні з Java та C# стало те, що для кожної архітектури процесора (86x, 64x, ARM) та ОС (Windows, Mac, Linuks) треба білдити(to build) новий exe-файл. Що стосується ефективності, то показники C++ були найнижчі, а Java та C# мали приблизно однакові результати.

Застосунок можна продовжити розвивати шляхом оптимізації та покращення серверів, додавання інших мов програмування для порівняння, а також створення Peer-2-peer взаємодії і порівнювання її з клієнт-сервером.

Це дасть змогу кращим способом порівняти обрані мови програмування.

Список використаних джерел

1. Phipps, G. (1999). Comparing observed bug and productivity rates for java and C++.
2. Dimitrios Serpanos (2011), Tilman Wolf, in Architecture of Network Systems.
3. Dr. Edward Thomas (2014), IOSR Journal of Computer Engineering.
4. Jan L. Harrington(2016), in Relational Database Design and Implementation (Fourth Edition).
5. https://sites.google.com/site/clientserverarchitecture/advantages-of-client-server-architecture
6. https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8
7. https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets?view=net-5.0
8. https://www.tiobe.com/tiobe-index
9. www.oracle.com
10. https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history
11. https://en.cppreference.com/w/cpp/language/history

12. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

13. https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-accept

14. https://www.sqlapi.com/ApiDoc/class_s_a_connection.html