КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп’ютерні науки

на тему:

АРХІТЕКТУРА “КЛІЄНТ-СЕРВЕР” НА ПРИКЛАДІ ПРОГРАМНИХ ПРОДУКТІВ

Виконав студент 4-го курсу бакалаврату

Назар ЛАВРЕНТЮК

Науковий керівник:

доцент, кандидат фіз.-мат. наук

_____ Віктор ВОЛОХОВ

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент  Назар ЛАВРЕНТЮК

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування «____» _____ 20__ р., протокол № ___

Завідувач кафедри ТТП, проф.

_____ Микола НІКІТЧЕНКО

Київ – 2023

РЕФЕРАТ

У роботі було проведено дослідження на тему "Клієнт-серверна архітектура" і вона складається з 60 сторінок, 20 ілюстрацій, 2 таблиці, 11 джерел посилань.

АРХІТЕКТУРА, ВИКОРИСТАННЯ ТЕХНОЛОГІЙ, ДОСЛІДЖЕННЯ, ЗАСТОСУВАННЯ, ЗНАЧУЩІСТЬ, КЛІЄНТ, МЕТОДИ, ПЕРЕВАГИ, ПОРІВНЯННЯ, РЕЗУЛЬТАТИ, СЕРВЕР, СТРУКТУРА, СФЕРА, ТЕХНОЛОГІЇ.

В роботі були досліджені основні аспекти клієнт-серверної архітектури, такі як її структура, типи, переваги та недоліки. Дослідження включало вивчення літературних джерел, аналіз систем, що існує, та проведення експериментів.

Мета: виявити переваги та обмеження клієнт-серверної архітектури, розібрати можливості, які надають різні мови програмування, порівняти різні реалізації моделі.

Об’єкт: застосунки на базі різних архітектур.

Актуальність: з появою нових технологій та трендів, таких як хмарні обчислення, IoT, мобільність та віртуалізація, архітектура клієнт-сервер стикається з новими викликами. Дослідження цих викликів та розробка передових рішень можуть бути актуальними для покращення роботи програмних продуктів у вимогливих середовищах.

Результати дослідження підтвердили ефективність клієнт-серверної архітектури для розробки різноманітних додатків. Виявлені переваги цієї архітектури, що включають модульність, масштабованість, повторне використання, відокремлення завдань та покращену безпеку. Було також виявлено недоліки, такі як складність розробки та підтримки системи.

Результати дослідження можуть бути використані в розробці програмного забезпечення з використанням клієнт-серверної архітектури. Впровадження цієї архітектури може знайти застосування в різних галузях, включаючи мережеві додатки, вебсервіси, хмарні системи та багато інших.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

БД – база даних;

GC – Garbage Collector, збирач сміття;

IDE – Integrated Design Environment, інтегроване середовище розробки;

ПК – персональний комп'ютер.

API – Application Programming Interface, програмний інтерфейс програми.

JIT – Just-in-Time, точно в потрібний час.

HTML - HyperText Markup Language, мова гіпертекстової розмітки.

CSS - Cascading Style Sheets, каскадні таблиці стилів.

IoT - Internet of Things, Інтернет речей.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Сучасний бізнес для подальшого зростання вимагає відповідати розвитку інформаційних технологій, а клієнт-серверна архітектура є однією з ключових складових сучасних та успішних програмних продуктів. У цьому контексті важко уявити навіть малий бізнес без наявності онлайн ресурсу, який надає клієнтам можливість переглядати товари, дізнаватись ціни, здійснювати покупки та отримувати необхідну інформацію. Розвиток програмування ніколи не стоїть на місці, постійно впроваджуються нові мови, бібліотеки, інструменти та технології, спрямовані на покращення та розширення функціонала програмних забезпечень, в тому числі клієнт-серверних.

Сучасний технологічний прогрес постійно змінює кожну галузь нашого життя і спосіб отримання інформації. У новітньому цифровому світі, клієнт-серверна архітектура стає невіддільною складовою нашого існування та щоденного функціонування.

Клієнт-серверна архітектура допомагає визначати взаємозв'язок між комп'ютерами, де пристрої клієнтів надсилають запити та отримують ресурси від сервера. Ця модель розподіленого обчислення дає змогу ефективно організувати і обробити інформацію, забезпечуючи надійну та швидку комунікацію між складовими архітектури.

Актуальність роботи та підстави для її виконання. Архітектура клієнт-сервер є однією з найпоширеніших архітектур у розробці програмного забезпечення. Багато програмних продуктів, включаючи вебзастосунок, мобільні додатки та хмарні сервіси, базуються на принципах архітектури клієнт-сервер. Дослідження та розуміння цієї архітектури є важливим для розробників та архітекторів програмного забезпечення.

З появою нових технологій та трендів, таких як хмарні обчислення, IoT, мобільність та віртуалізація, архітектура клієнт-сервер стикається з новими викликами. Дослідження цих викликів та розробка передових рішень можуть бути

актуальними для покращення роботи програмних продуктів у вимогливих середовищах.

Мета й завдання роботи. Мета кваліфікаційної роботи - розкрити основні принципи функціонування цієї моделі та визначити найкращі практики її реалізації. Я планую дослідити сучасні мови програмування та популярні фреймворки, які дозволяють ефективно розробляти клієнт-серверні системи, і використовувати їх у практичних прикладах для демонстрації роботи архітектури. В результаті дипломної роботи я очікую виявити переваги та обмеження клієнт-серверної архітектури і запропонувати рекомендації для її оптимального використання в розробці програмного забезпечення, а також розібрати можливості, які надають різні мови програмування.

Об'єкт, методи й засоби розроблення. Об’єктом розроблення програмного засобу є реалізація чатів різних архітектур, а також їх дослідження та порівняння.

Розробці програмного засобу передували теоретичні відомості про різні архітектури, їх переваги та недоліки, а також порівняння та аналіз різних мов програмування для додаткового дослідження інструментів, які надають різні мови.

Можливі сфери застосування. Програмний продукт може слугувати внутрішнім безпечним зв’язком всередині невеликої компанії.

РОЗДІЛ 1 ВСТУП ДО КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

1.1 Основні компоненти клієнт-серверної архітектури

1.1.1 Сервер

Сервер – це комп’ютер або програмне забезпечення, призначене для обробки запитів і передачі даних комп’ютерам або пристроям клієнтів через мережу.

Сервер може бути фізичним комп'ютером або програмним забезпеченням.

Він здатний паралельно обслуговувати велику кількість клієнтів, виконуючи їх запити та надаючи доступ до різних ресурсів або функцій програми, що розташоване на віртуальному або хмарному середовищі.

Сервер може забезпечувати різноманітні послуги, такі як:

- зберігання даних;
- обмін різними файлами;
- видалення файлів або даних;
- надання доступу до додатків, що потребує клієнт;
- виконання важких математичних обчислень, які вимагають значних ресурсів.

Сервер є головною складовою даної архітектури і виконує найважливіші функції у системі. Одна з важливих характеристик сервера - це його доступність та надійність.

Ми можемо налаштувати сервери з такими параметрами:

- резервування;
- дублювання;
- механізми відновлення;
- механізми аутентифікації;
- механізми авторизації;
- шифрування даних;
- механізми аудиту.

Такі додаткові налаштування забезпечують безпечну роботу системи, повну конфіденційність, цілісність та доступність, уникнення втрати даних в разі виникнення непередбачених ситуацій, а також контроль доступу клієнтів до ресурсів.

Загалом, сервер є ключовим компонентом архітектури, адже він забезпечує ефективну роботу системи, надає послуги клієнтам та обробляє запити у високопродуктивному режимі.

Рисунок 1 – Приклади клієнтів та серверів

1.1.2 Клієнт

Клієнт – це поняття, яке може мати декілька значень, а саме, це може означати кінцевого користувача, що надсилає запити серверу або програму, яка встановлює зв'язок із сервером через мережу з метою отримання певних даних, виконання обчислень або звернення до певного сервісу.

Кінцеві користувачі можуть бути як справжніми людьми, так і цілими організаціями, які користуються послугами, які може надати сервер або інші клієнти. Щодо клієнтських програм, то вони можуть бути залучені на різних пристроях, таких як комп'ютери, смартфони, планшети й становлять собою застосунки, які є проміжним етапом зв’язку кінцевого клієнта з сервером.

Відповідно до прописаного протоколу взаємодії, сервер отримує запити клієнтів і обробляє їх, надаючи користувачам доступ до потрібних ресурсів, формулюючи результати, які потім можуть бути додатково оброблені, відображені, видалені або збережені самим клієнтом.

Клієнт в залежності від наданого функціоналу може мати різні функції та можливості. Наприклад, месенджер є типовим прикладом клієнтської програми, яка дозволяє користувачеві переглядати діалоги, відправляти файли іншим клієнтам, переглядати надану інформацію про користувачів тощо.

1.1.3 Клієнт-серверна архітектура

Клієнт-серверна структура – це програмна архітектура, що містить два основних компоненти – клієнт та сервер. За допомогою неї користувачі надсилають запити після взаємодії з клієнтською частиною, тоді як сервер відповідає на надіслані запити.

Така архітектура забезпечує міжпроцесовий зв’язок. Таким чином обмін даними відбувається як на стороні клієнта, так і на стороні сервера, завдяки чому кожен із них виконує свої функції, що закладені у коді.

Ключове завдання клієнт-серверної структури полягає в розподілі обов'язків між клієнтами та сервером. З п. 1.1.1 та п. 1.1.2 бачимо, що сервери, зазвичай, забезпечують високопродуктивне обслуговування клієнтів, виконуючи основні обчислення, зберігаючи або видаляючи дані та надаючи потрібні ресурси. Клієнти ж мають менш потужні обчислювальні технології, проте їх вистачає для опрацювання даних від сервера та висвітлення їх користувачам.

Мережеве навантаження залежить від запитів з маніпуляцією даними, надісланих з ПК на сервер БД, і попередньо необроблених даних, що повертаються клієнту. Результатом є значно менше використання мережевого трафіку і краща продуктивність, а тому і швидкість.

На сьогодення клієнт-серверні архітектури здійснюють обмін повідомленнями через локальні мережі, що базуються на стандартах Ethernet.

Проте дуже рідко зустрічаються винятки, які досі використовують старі локальні мережі, такі як Token Ring.

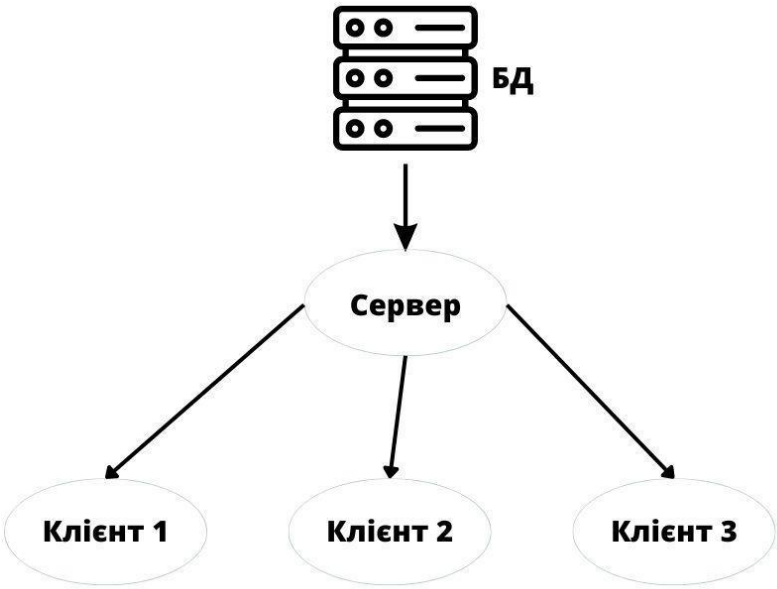

Рисунок 2 – Клієнт-серверна архітектура

1.2 Переваги та недоліки архітектури

Почнемо з переваг:

- Покращений обмін даними - дозволяє знизити навантаження на мережу і забезпечити швидку передачу даних.
- Спільні ресурси між різними платформами - сприяє оптимізації використання ресурсів, покращення продуктивності системи. Також це надає сумісність між різними системами, здатність легко розширити функціональність та можливості системи.
- Інтеграція служб - дозволяє оптимізувати робочі процеси, забезпечити ефективну обробку даних та зменшити витрати на розробку окремих систем.

- Безпека – надає можливість контролю доступу, шифрування даних та забезпечує захист від атак.
- Масштабованість – клієнт-серверна архітектура дає можливість легко розширяти систему.
- Простота обслуговування – архітектура надає всі інструменти для спрощення обслуговування системи.
- Можливість обробки даних, попри їх місцеперебування – дані можуть зберігатися на різних серверах або в різних місцях, але все одно вони будуть доступні для обробки і використання.
- Централізоване управління – сервер, виконуючи роль управління, спрощує керування системи.

Далі розберемо основні недоліки:

- Перевантаження серверів – призводить до неефективної обробки запитів, зменшення продуктивності та виникнення непередбачених результатів.
- Вплив централізованої архітектури – впливає на розширення, робить набагато складнішим питання масштабування.
- Початкові умови для праці – оскільки взаємодія відбувається через мережу, слабка швидкість може викликати затримки та збої.

Взявши до уваги всі вищезазначені факти, можна стверджувати, що переваг більше ніж недоліків, а це свідчить про те, що клієнт-серверна архітектура є дуже вдалою розробкою і з часом вона буде надалі покращуватись та збільшувати число користувачів.

1.3 Класифікація клієнт-серверної архітектури

Клієнт-серверну архітектуру класифікують на:

- однорівнева архітектура;
- дворівнева архітектура;
- трирівнева архітектура;
- багаторівнева архітектура;
- мікросервісна архітектура;
- розподілена архітектура.

Розберемо їх більш детально.

1.3.1 Однорівнева архітектура

Однорівнева клієнт-серверна архітектура – це найпростіша форма структури, де вся бізнес-логіка та обробка даних відбуваються на сервері, а клієнти виконують роль лише простих інтерфейсів для взаємодії з сервером. У цій архітектурі клієнти не мають власної логіки та обробки даних, а лише відправляють запити до сервера та отримують від нього відповіді.

Такий тип структури використовується для задоволення простих систем з центральним механізмом управління, які в найближчому часі не планують масштабуватись. У такого підходу є свої переваги та недоліки.

Рисунок 3 – Однорівнева клієнт-серверна архітектура

Переваги:

– Простота розробки - через те, що вся логіка розташована на сервері, клієнти це просто інтерфейси для взаємодії, це значно спрощує процес розробки клієнтської частини.

– Актуальність інформації – оскільки вся інформація знаходиться на сервері, то при зміні даних у БД, не виникне ситуації несумісності даних у різних клієнтів.

– Легша реалізація безпеки даних - оскільки вся логіка зберігається на сервері, то для захисту даних достатньо зробити механізм захисту лише на стороні серверу.

– Мале навантаження клієнтів - оскільки основна частина обробки інформації знаходиться на сервері - це значною мірою зменшує навантаження на клієнтську частину.

Недоліки:

– Обмеженість у масштабуванні - зі зростанням складності реалізації логіки та кількості клієнтів, швидко буде зростати навантаження на сервер, що може стати критичним для коректної роботи.

- Повна залежність від сервера - при проблемах з сервером, клієнти також зупинять свій функціонал.
- Залежність від мережі - оскільки при однорівневій архітектурі відбувається активна комунікація мережею, то при виникненні навіть малих проблем з мережею, система не буде працювати.
- Висока вимога до потужності сервера - оскільки за все відповідає сервер, то для нього потрібні більш потужні сервери ніж, коли логіка розподілена між компонентами.

Отже, легко зрозуміти, що така структура ефективна, при простій бізнес-логіці та обмеженій кількості користувачів.

1.3.2 Дворівнева архітектура

Дворівнева архітектура це одна з форм розподіленої архітектури, яка включає два основних рівні: клієнтський і серверний.

На клієнтському рівні розташовані клієнтські програми або пристрої, які взаємодіють з користувачем через інтерфейс та сервером, надсилаючи різні запити та обробляючи отримані відповіді на них для отримання даних або виконання певних операцій.

Серверний рівень складається з серверів, які отримують запити та обробляють їх, виконують необхідні операції та забезпечують доступ до даних чи послуг, що необхідні користувачу.

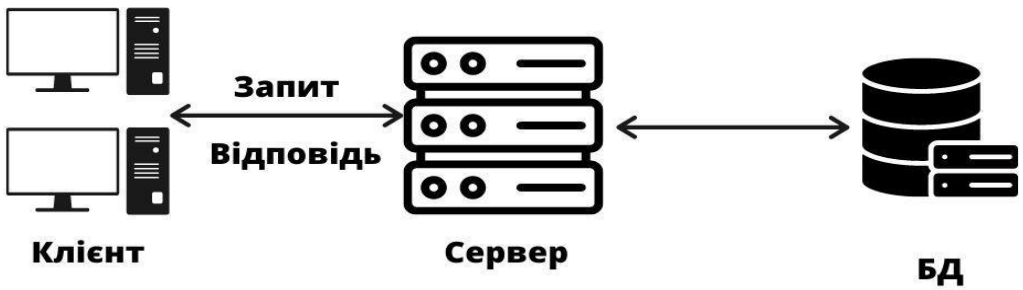

Рисунок 4 - Дворівнева клієнт-серверна архітектура

Переваги:

– Розширюваність - на відміну від однорівневої архітектури, сприяє розширенню шляхом додавання нових серверів та клієнтів.

– Висока продуктивність - завдяки розділенню обов’язків між компонентами архітектури, надає можливість забезпечити високу продуктивність та швидку відповідь на запити користувачів.

– Легкість в обслуговуванні окремих компонентів - оскільки обов’язки розподілені, окремі моделі можуть бути модифіковані чи розширенні без залежності від інших модулів.

– Розповсюдженість - така архітектура є дуже популярною, тому багато питань, які можуть виникнути в процесі розробки можна знайти розібраними у просторах Інтернету, а також наявні багато фреймворків та додаткових бібліотек для спрощення та покращення роботи з різними технологіями.

Недоліки:

- Перевантаження сервера - зі збільшенням кількості клієнтів призводить до великої навантаженості сервера та виникнення можливих проблем з обробкою та зберіганням даних.
- Залежність від мережі - як і при однорівневій архітектурі у дворівневій архітектурі відбувається активна комунікація мережею, тому при виникненні навіть малих проблем з мережею, система може працювати.
- Централізований контроль - велика залежність від сервера, що може призвести до виникнення затримок або втрати даних.

Дворівнева архітектура показує кращі результати на більш масштабованих застосунках, проте все ще доволі великі недоліки виходячи з простоти розробки такої системи, такий тип структури найкраще підійде для продуктів де, бізнес-логіку можна чітко поділити на 2 частини - клієнтську та серверну. Наприклад, така архітектура чудово підійде для простих веб застосунків, мобільних програм тощо.

1.3.3 Трирівнева архітектура

Трирівнева архітектура - це структура, що передбачає відділення прикладного рівня від маніпуляції даними. Як бачимо, виокремлюється деякий додатковий програмний рівень, який зосереджується на вирішені прикладної логіки застосування.

Програми такого проміжного рівня часто функціонують під керуванням спеціальних серверів застосунків, але запуск програм також може здійснюватися і під керуванням звичайного вебсервер.

Частіше всього сервер даних і серверні модулі третього рівня розміщуються на одному пристрої, хоч і являють собою зовсім окремі й логічно незалежні програмні рівні, що допомагають розділити обов’язки.

Основні рівні в трирівневій архітектурі містять:

1. Рівень клієнта - також називають верхнім шаром архітектури, головне завдання якого забезпечити взаємодію застосунку з користувачем, а також збір та валідацію даних, введених клієнтом, та відображає дані або ресурси отримані у відповідь на запит клієнта.

2. Додатковий рівень (бізнес або логічний) - також називають проміжним шаром клієнт-серверної архітектури цей рівень відокремлює всю бізнес-логіку та обробку даних системи від інших рівнів. Він відповідає за обробку наявної логіки, виконання різних розрахунків, доступ до бази даних та інші операції над наданою інформацією. На цьому рівні зазвичай реалізовані допоміжні сервіси, компоненти та модулі, які забезпечують ефективну функціональність системи на всіх рівнях.

3. Рівень сервера - також називають нижнім шаром архітектури, це рівень, що забезпечує доступ до БД, а також здійснює збереження, видалення та обробку даних, що надійшли від клієнта або проміжного рівня. Серверний рівень може включати БД, серверні компоненти та різноманітні сервіси, які виконують операції з наданою інформацією, забезпечують надійну безпеку, масштабованість та інші аспекти серверної частини системи, для оптимізації роботи всієї системи.

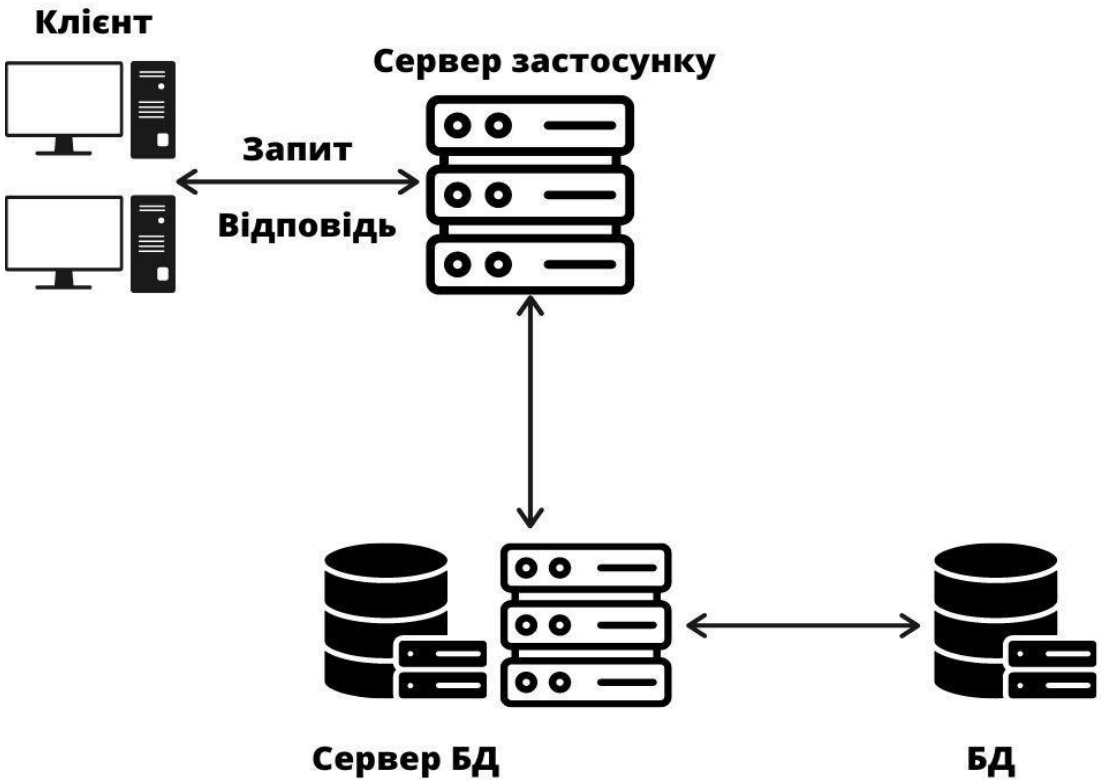

Рисунок 5 - Модель трирівневої архітектури

Розглянемо переваги такого типу архітектури:

– Розділення відповідальності між різними шарами системи - одне з головних переваг трирівневої архітектури над вище розглянутими, адже це дозволяє легше керувати даними та розвивати систему.

– Покращена безпека - надає можливість реалізувати надійний захист даних на різних рівнях архітектури, що значно покращує збереження даних та безпеку усього продукту, хоч і важча у реалізації.

– Модульність - властивість, яка не змінює інші рівні при модифікації одного з рівнів, що дає гнучкість коду та здатність легко вносити зміни в окремі рівні. Наприклад, зміна в інтерфейсі рівня клієнта ніяк не впливає на бізнес-логіку та обробку даних на проміжному рівні або на результат звернення до БД на рівні сервера. Також дає можливість надати розробку окремих рівнів різним командам розробників, що надає легкості в розробці.

− Гнучке тестування - завдяки модульності, кожен з рівнів легко тестувати, незалежно від інших компонентів чи сервісів даної архітектури.

Недоліки:

− Підвищення рівня складності розробки - одна з головних проблем полягає у збільшенні кількості рівнів. Це сприяє наростанню складності розробки архітектури та забезпечення надійного захисту кожного з компонентів.

− Зростання часу очікування відповіді - зі збільшенням кількості рівнів збільшується кількість передач даних між компонентами, що значно впливає на час обробки запиту.

− Питання стабільності - при виході з ладу одного з компонентів, вся система застигне на місці, адже без одного з рівнів коректна обробка запиту неможлива.

− Більші вимоги до продукту та витрати на управління - кожен елемент вимагає окремі правила керування компонентом, що може ускладнити загальну взаємодію.

Найкращим прикладом застосунку трирівневої клієнт-серверної архітектури може слугувати будь-який веб сервіс малого бізнесу, адже він містить інтерфейс, тобто клієнтський рівень, бізнес-логіку ресурсу й обробку переказів (при онлайн замовленнях) та рівень доступу до БД з клієнтською інформацією та даними по товарах, тобто рівень сервера.

1.3.4 Багаторівнева архітектура

Багаторівнева клієнт-серверна архітектура - це розширена форма трирівневої архітектури. У даній архітектурі обробка різних додатків, маніпуляції та функції управління даними - усі ці параметри ізольовані один від одного, що доволі

ефективно. Фактично багаторівнева структура - це архітектура у яких виділено більш ніж три основних компоненти.

Кожен рівень виконує свої окремі завдання і взаємодіє з іншими рівнями для обробки запитів та передачі даних. Основна ідея багаторівневої архітектури полягає у повному поділі функцій на класи й присвоєнні відповідальності між різними рівнями для забезпечення більшої модульності та масштабованості системи.

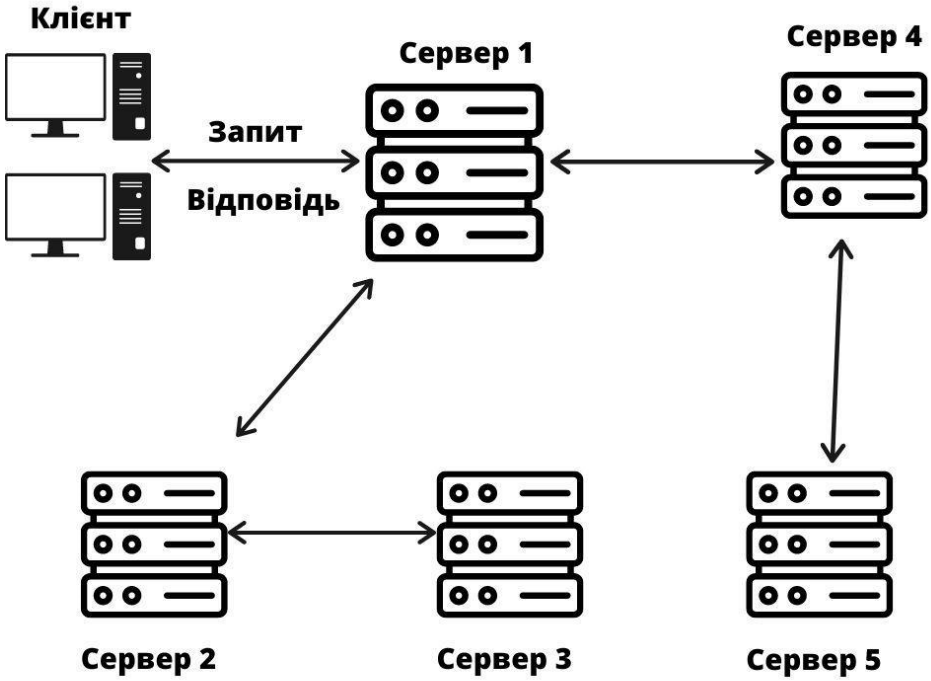

Рисунок 6 - модель багаторівневої клієнт-серверної моделі

Переваги:

– Краща модульність - система розбивається на менші рівні з чітко визначеними функціональними блоками, що полегшує розробку, тестування та підтримку системи.

– Легка масштабованість - кожен рівень можна розширити без зміни інших рівнів, що дозволяє швидко міняти структуру коду окремих модулів.

- Повторне використання - розбиття на менші функціональні блоки дає змогу використовувати їх в інших програмах або на інших рівнях, що спрощує розробку.
- Легший процес тестування - оскільки архітектура розбита на менші блоки за темами, їх доволі легко тестувати, адже вони не залежні від інших.
- Сумісність - сприяє інтеграції з наявними системами та розширенню функціональності шляхом інших архітектур та готових функціональних блоків.

Недоліки:

- Підвищення рівня складності розробки - як і для трирівневої архітектури, одна з головних проблем полягає у збільшенні кількості рівнів. Це сприяє наростанню складності розробки архітектури та забезпечення надійного захисту кожного з компонентів.
- Зростання часу затримок - комунікація між рівнями може призвести до довгих затримок, через те що, кількість відправлення між рівнями збільшується, а запит від клієнта розбивається на декілька розподілених запитів на функціональні блоки.
- Питання стабільності - при виході з ладу одного з компонентів, вся система застигне на місці, адже без одного з рівнів коректна обробка запиту може дати збій.
- Залежність від потужності мережі - при малій швидкості, запити можуть довго очікувати на відповідь, застрягати між рівнями, втрачати дані тощо.
- Побудова взаємодії між компонентами потребує більше ресурсів - через складність розробки такої системи і додаткових затрат на нові команди розробників збільшує використання ресурсів.

Як приклад можна навести вебзастосунок великої компанії, такої як Amazon. Такий вебресурс, крім попередніх 3 рівнів містить допоміжні рівні інформації або

додаткові сховища, які відповідають за збереження та керування даними, такими як бази даних або файлові системи.

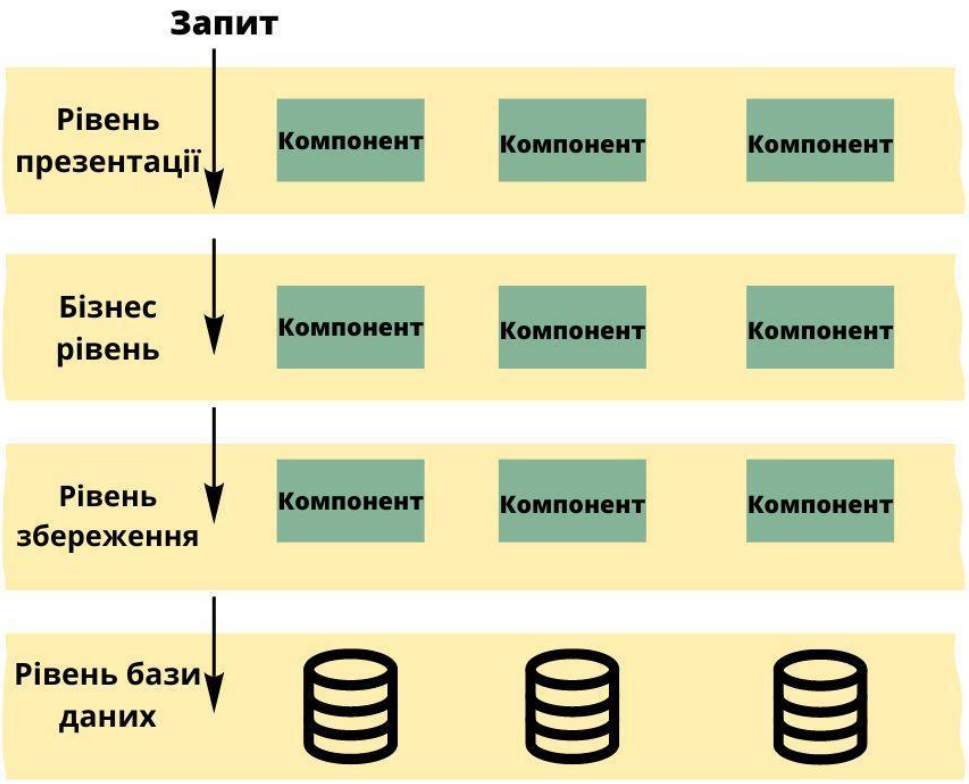

Рисунок 7 - Багаторівнева клієнт-серверна архітектура

1.3.5 Мікросервісна архітектура

Мікросервісна архітектура є ще одним з основних типів архітектури, в якому програмна система розбивається на багато невеликих та незалежних компонентів, відомі як мікросервіси. Кожен мікросервіс виконує якусь свою роботу, функцію або набір функцій та працює незалежно від інших мікросервісів. Комунікація між мікросервісами зазвичай здійснюється за допомогою протоколів. Найчастіше використовується HTTP.

Одними з головних особливостей мікросервісної архітектури є її масштабованість та гнучкість, адже для того, щоб, збільшити архітектуру, достатньо додати нові мікросервіси або масштабувати уже чинні, але, оскільки, усі

компоненти такої структури незалежні одна від іншої, то розширення системи вважається не важкою дією.

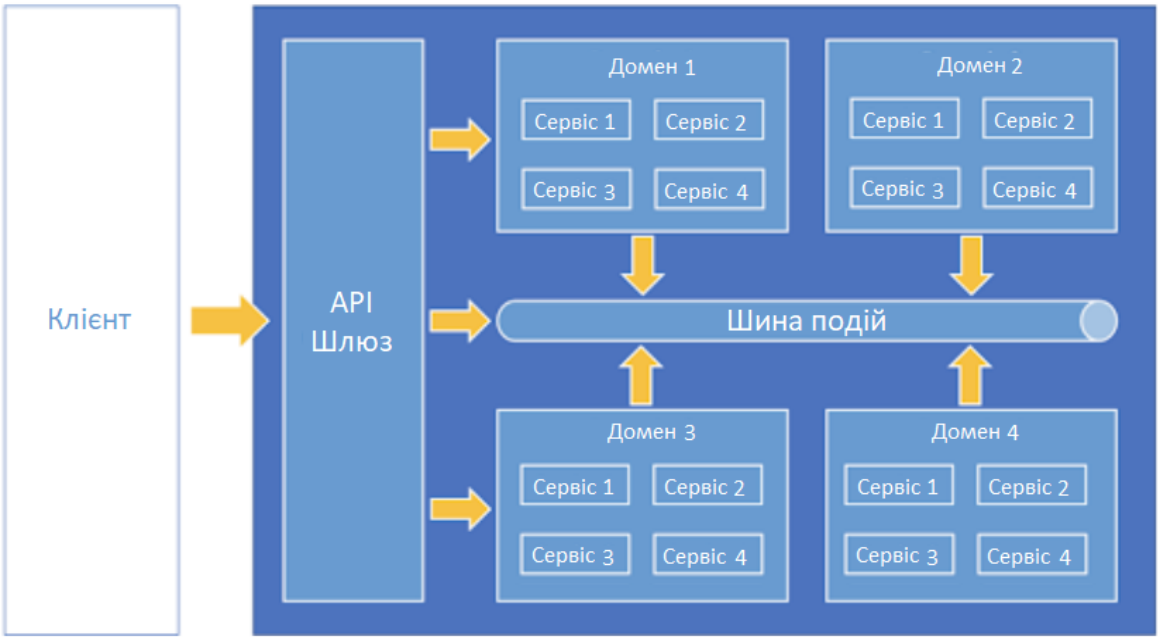

Рисунок 8 - Мікросервісна клієнт-серверна архітектура

Переваги:

– Гнучкість та масштабованість - як було сказано вище це і є одними з особливостей такої архітектури.

– Зменшення ризиків - це допомагає уникнути однієї з головних проблем інших архітектур, а саме залежність одного від інших, якщо один з компонентів у попередніх структурах виходив з ладу, то ставилось питання, щодо коректної роботи всієї системи. В такому випадку, при виникненні помилки в одному з мікросервісів всі інші компоненти продовжують працювати.

– Відокремленість - кожен з мікросервісів може бути написаний мовою, яка найкраще підходить під конкретну задачу, адже мікросервіси між собою не залежать, а також це допомагає при оновленні окремих компонентів, адже це ніяк не вплине на роботу інших частин.

– Легка розробка - така архітектура дозволяє розробляти кожен компонент окремо і незалежну і навіть різними групами розробників, що дозволяє легко виконувати розробку, тестування, модифікації тощо.

Недоліки:

– Проблеми з керуванням - у всього є свої мінуси, для такої архітектури з легкістю масштабування приходить проблема складності координування мікросервісів як одного механізму.

– Казуси з оновленнями - при модифікації якогось мікросервісу, треба забезпечити, що оновлення буде сумісне з попередньою версією і що взаємодія між сервісами матиме можливість продовжувати коректну працю.

– Складність тестування взаємодій - якщо в такій архітектурі легко тестувати окремі компоненти, то складно перевірити коректність взаємодії, тому віднестись до перевірки треба відповідально.

– Навантаження на мережу - чим більше компонентів у системі, тим більше навантаження на роботу продукту, можливі довгі затримки тощо.

– Більші витрати - підтримувати одночасно роботу усіх мікросервісів, обслуговувати їх, вдарить по затратах.

Як бачимо, при збільшенні кількості компонентів, зростає і складність системи, що несе за собою наслідки перевірки безпеки, досконалої розробки взаємодії між компонентами та збільшення використання системних ресурсів.

Прикладом мікросервісної архітектури може слугувати платформа Netflix, яка є провайдером відеопотокового сервісу, що доволі широко використовується у світі.

1.3.6 Розподілена архітектура

Розподілена клієнт-серверна архітектура - це тип структури, в якій функції та модулі системи розподілені між клієнтами та серверами, які взаємодіють за допомогою мережі.

У цій архітектурі клієнти є користувачами, які надсилають запити серверам для отримання ресурсів або сервісів, модифікації даних або виконання певних операцій. Сервери надають клієнтам потрібну відповідь на запит, виконують операції над даними та забезпечують доступ до системних функцій.

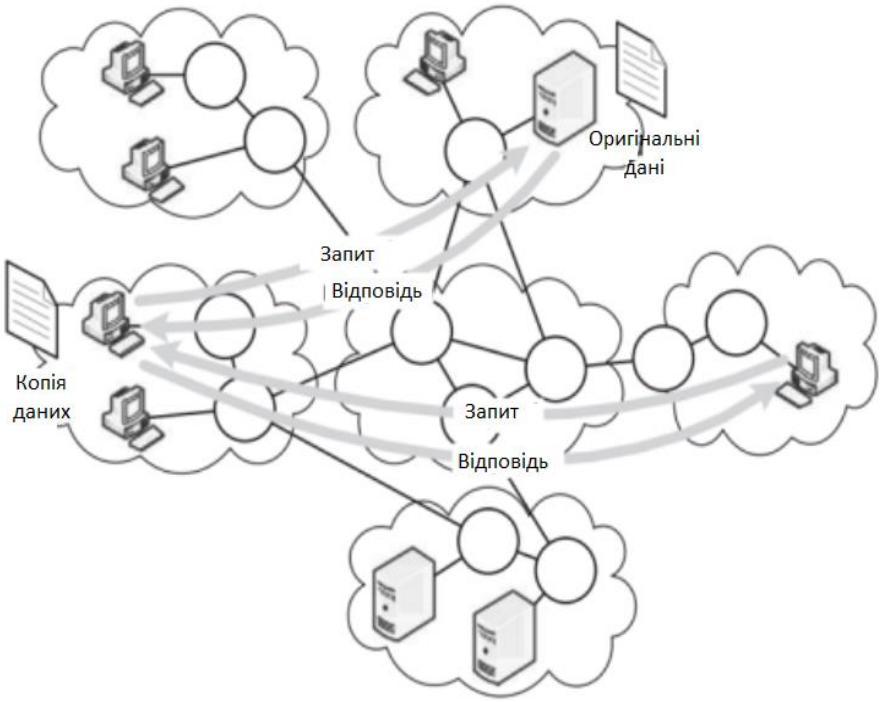

Рисунок 9 - Розподілена клієнт-серверна архітектура

Переваги:

– Стійка безпека - через можливість захистити доступ до рівня серверів, безпека реалізується неважко та надає надійний захист даних.

- Швидкість обробки запитів - клієнти та сервери взаємодіють напряму, не завантажуючи мережу, що впливає на швидкість надання відповіді клієнту.
- Поділ завдань за призначенням - клієнтські та серверні рівні чітко розмежовують границі між функціоналом на різних рівнях, що забезпечує легке модульне тестування.

Недоліки:

- Важкість планування розробки - така архітектура вимагає чіткого плану розробки, для коректної роботи системи.
- Складність керування системою - як і розробка, керування буде викликом для розробників, потрібно створити чіткий протокол взаємодії для правильної роботи структури.
- Важливість стабільної мережі - від мережі залежить багато факторів, при низькій пропускній здатності мережі або тривалих затримок, можуть виникнути непередбачені ситуації.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ НА ПРИКЛАДІ ЧАТУ

2.1 Постановка практичної задачі

Створити клієнт-серверний додаток, а саме чат, з однорівневою архітектурою, використовуючи сучасну мову сім'ї "С" Java, використовуючи технологію Socket, розібравши особливості побудови програми та оцінивши використані бібліотеки та пояснити вибір саме цієї мови програмування.

Як другий етап створити клієнт-серверний додаток на прикладі чату, проте з використанням дворівневої архітектури та іншої мови програмування, а саме Python з використанням фреймворку Flask, розмістити його на безкоштовну платформу та перевірити коректну роботу на різних пристроях (різні комп'ютери, ноутбуки та мобільні пристрої).

2.2 Актуальність мов програмування

2.2.1 Порівняння мов програмування C++, C# та Java

Для початку зробимо порівняння мов програмування сім'ї "С", а саме C++, C# та Java.

Ціллю створення C++ було бажання розширити функціонал мови С, найбільш розповсюджену мову на той час. Оскільки упор був на ті ж аспекти, що і мова С, то і деякі не найкращі особливості C++ потягнув за собою. Щодо мови Java, розробники пішли декілька іншим шляхом, вони зосередили зусилля не стільки на прив'язку до батьківської мови, як на збільшення ефективності та створення цілісної мови програмування. Пізніше розробниками Microsoft була запропонована ще одна переробка мови C++, яка пішла шляхом близьким до Java. Побудуємо таблицю порівняння даних мов по різним критеріям:

Таблиця 2.1 - Порівняння мов програмування

Критерій	C++	C#	Java
Синтаксис	Максимально зберегли сумісність з батьківською мовою С.	Максимально зберегли сумісність з батьківською мовою С, з виправленням не найкращих рішень та додаванням додаткових інструментів збільшення ефективності.	Зберігає візуальну сумісність з мовами С і С ++. Проте через те, велику кількість синтаксичних інструментів розробники прибрали, заявивши, що вони тепер не обов’язкові, а пізніше і зовсім відмовившись від багатьох з них, мова Java сильно відрізняється від них. Тому хоч програми, написані на Java здаються більш громіздкими, вони стали легші у вивченні та розумінні.

Управління ресурсами та GC	В основі покладено принцип захоплення ресурсів при їх ініціалізації, який дає можливість автоматично очищати пам’ять від цього ресурсу при знищенні об’єкта.	C# пішли трохи та розробили як автоматичний, так і надали деякі можливості розробникам вручну викликати GC та використовувати його функції. З ними можна ознайомитись у класі System.GC. Автоматичний GC виділяє та звільняє пам’ять у проекті при появі або закінчення життя об’єктів. Така реалізація дає більше інструментів для роботи.	У цій мові наявний прибиральник сміття, проте він стежить лише за пам’яттю. Також, йому для роботи потрібні значні системні ресурси, що значно знижує ефективність виконання програм.
Стандарти оточення	C++ доволі вільно ставиться до стандартів на введення або виведення,	Аналогічно до C++.	Java має чітко визначені стандарти на багато речей, таких як:

	графіку, доступ до БД тощо.		введення та виведення, графіку, геометричну інтерпретацію, діалоги, отримання доступу до БД та інших подібних додатків. Стандарти на графіку, отримання доступ до баз даних тощо інколи буває суттєвим недоліком, якщо розробник має завдання визначити власні стандарти.
Вказівники	Низький рівень можливості використання вказівників.	Вказівники реалізовано на дещо вищому рівні ніж на C++, проте все одно розробниками	Вказівники відсутні, оскільки Java-розробники визнали вказівники як не обов’язкові, та

		використовується дуже рідко, лише задня мінімальної оптимізації програм по часу, при прямому зверненню до об’єкта.	згодом вирішили зовсім прибрати їх.
Етап компіляції	Після запуску процесу компіляції, код одразу перетворюється на машинний.	Спочатку код компілюється в проміжний і лише згодом він інтерпретується в машинний.	Аналогічно до C# засобами компіляції, спочатку код перетворюються на проміжний між початковим та машинним, а потім з використанням технологій компілятора JIT код перетворюється його на машинний.
Парадигми програмування	Об’єктно - орієнтована та процедурна.	Об’єктно - орієнтована.	Об’єктно - орієнтована.

Функції препроцесора.	C++ реалізовує функції препроцесора для визначень класів функцій для роботи з бібліотеками, повністю зроблених у вихідному коді, а також дозволяє метапрограмувати з використанням функцій препроцесора, яке вирішує доволі складні проблеми пов'язані з дублюванням коду. Кажуть, що такий механізм не може бути безпечним, оскільки імена макросів препроцесора доступні, а самі макроси не пов'язані з	Визначено декілька директив препроцесора, що можуть впливати перетворення коду з вихідного у машинний.	Все більше програмістів, що намагались застосувати у своїй роботі інструменти препроцесора зіштовхнулись з низкою проблем з ним, тому розробники мови прийняли рішення назовсім прибрати його з Java. Розробники мови виключили функції препроцесора, позбавившись від всіх проблем з його використанням, проте при цьому втративши всі можливості метапрограмування препроцесора і заміну тексту

	конструкціями C++. З іншого погляду, мова надає достатньо інших зручних інструментів такі як константи, шаблони, вбудовані функції для того, щоб набагато зменшити використання функцій препроцесора.		всередині коду різними засобами мови.

2.2.2 Статистика та популярність мов програмування

Беручи до уваги рейтинг, наданий українською компанією Dou, можемо побачити популярність мови C++ потихеньку згасає, за останні два роки її рейтинг на ринку знизився більш ніж на 35%. Щодо Java та C#, то останні роки рейтинг Java падає, проте незначно, приблизно на 3%, C# до минулого року показував непоганий ріст на ринку, проте останній рік він втратив 8% рейтингу. Популярність серед додаткових мов програмування, яка потроху набирає рейтинг набула мова Python, яка за останній рік набрала понад 20% рейтингу. Скоріш за все Python скоро викине PHP з ринку, саме тому він набирає популярність.

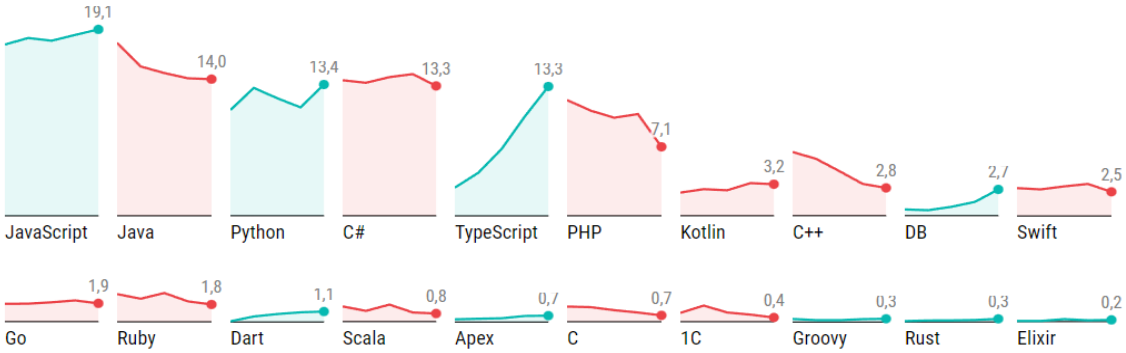

Рисунок 10 - Рейтинг мов за 2023 рік виходячи з рейтингу Dou

Якщо порівнювати по медіанним зарплатам на момент зими 2022 року, то бачимо, що Java, Python та C# мають приблизно однакові числа, тоді як C++ дійсно відстає.

Рисунок 11 - Медіанні зарплати

2.2.3 Висновок

Виходячи з усіх порівнянь для першої клієнт-серверної архітектури я обрав Java, оскільки все більше програмістів надають перевагу саме цій мові

програмування і вона не один рік доводить свою стабільність. Щодо середовища я обрав IntelliJ IDEA Ultimate, оскільки вона на голову вище інших середовищ завдяки зручному інтерфейсу, допоміжних інструментів тощо.

Щодо другого застосунку, я обрав Python, як мову яка розвивається і займає значне місце у рейтингу мов. Середовище я також обрав від компанії JetBrains, а саме PyCharm.

2.3 Реалізація першого клієнт-серверного застосунку

Чат був реалізований з однорівневою архітектурою, тобто і основна логіка та обробка запитів відбувається на сервері, а клієнт лише надає базовий інтерфейс для відображення даних, і сервер і клієнт працюють в одній локальній мережі.

2.3.1 Class Server

Були використані такі бібліотеки:

- java.io.FileWriter,
- java.io.IOException,
- java.net.ServerSocket,
- java.util.ArrayList,
- java.util.concurrent.LinkedBlockingQueue

З’єднання клієнтів з сервером відбувається за допомогою сокета. Коли клієнт підключається до сервера, вказавши свій нікнейм, сервер відповідно пише у чат, що відповідний користувач приєднався.

```
A new client connected!
Server: asdasd has entered the chat!
```

Рисунок 12 - Встановлено з’єднання між клієнтом та сервером.

Коли клієнт відправляє повідомлення, сервер, отримуючи запит починає обробляти його та після обробки відправляє його всім клієнтам. Для відправника

буде висвітлено лише повідомлення, а для інших користувачів повідомлення буде у форматі “Username” + “: ” + “Message”

Рисунок 13 - Інтерфейс чату при відправленні повідомлення.

Також наявні допоміжні команди, для допомоги користувачу, а саме “/help”, “/userlist”, “/msg”.

“/help” – команда допомоги, підказує які команди наявні;

“/userlist” – команда, яка показує які користувачі в мережі;

“/msg” – команда надсилання приватного повідомлення конкретному користувачу.

Всі звернення клієнтів записуються на сервері та в окремий файл в кореневій теці.

2.3.2 Class Client

Оскільки клієнт в однорівневій архітектурі не містить ніякої логіки, то сам клас складається лише з опису інтерфейсу та опису посилання запиту на сервер.

Були використані бібліотеки:

- import javax.swing.*;
- import java.awt.*;
- import java.awt.event.*;
- import java.io.*;
- import java.net.Socket;

Java Swing - це бібліотека для створення графічного інтерфейсу мовою програмування Java.

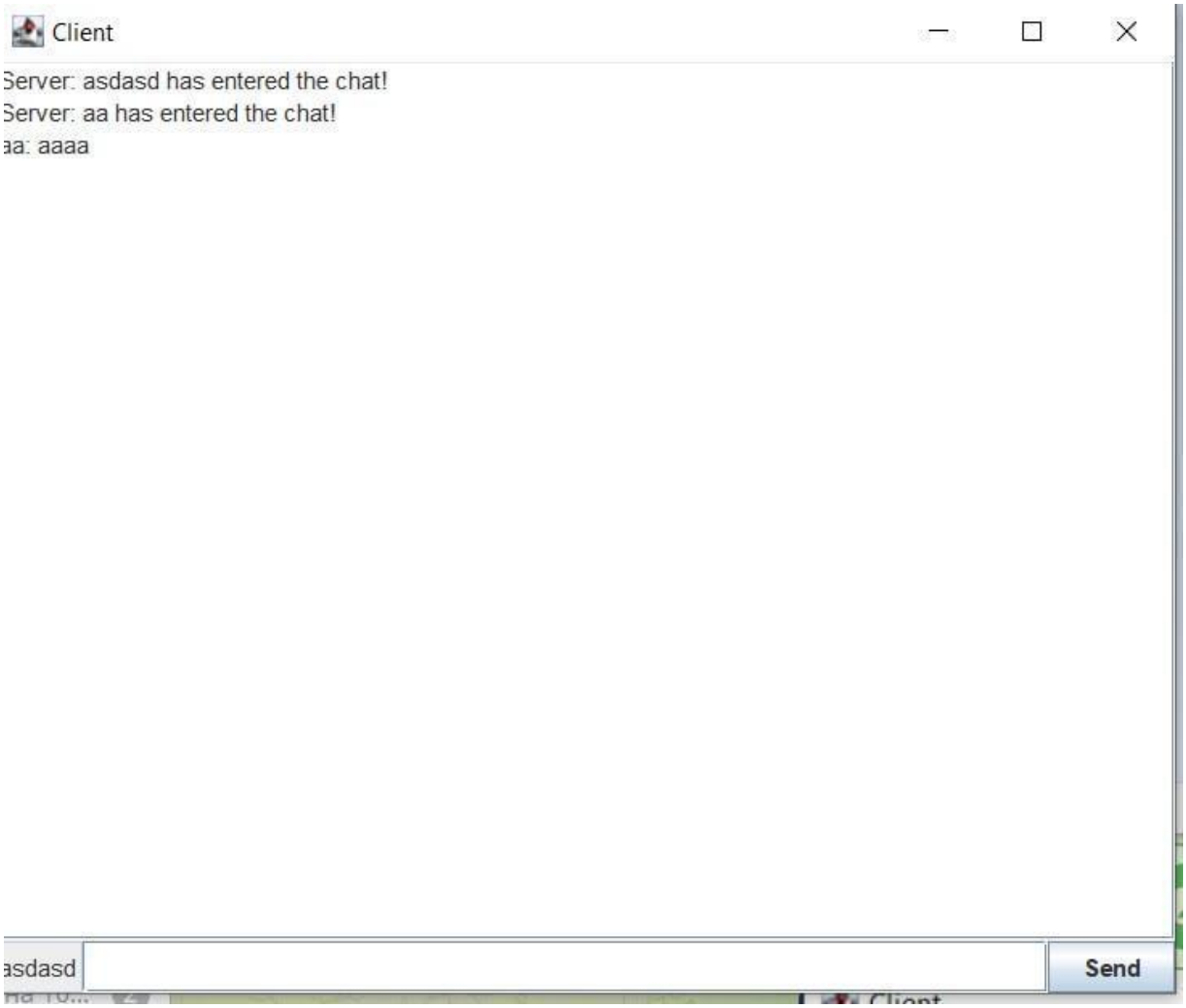

Рисунок 14 - Інтерфейс клієнтського рівня

Зліва знизу пишеться бачимо текст, який відображає username користувача, посередині поле самого чату, де відображаються дані про приєднання або від’єднання користувачів та самі повідомлення, знизу ще одне поле для введення самого повідомлення і кнопка “Send”, для посиланні запиту на сторону сервера.

2.3.3 Class ClientHandler

Даний клас є допоміжним класом Сервера, який при отриманні повідомлення від користувача, перетворює повідомлення у формат, який приймається сервером та відправляє відповідно далі на сервер.

```
@Override
public void run()
{
    String messageFromClient;

    while (socket.isConnected()) {
        try {
            messageFromClient = bufferedReader.readLine();
            this.server.messageQueue.add(this.clientUsername + ": " + messageFromClient);
        } catch (IOException e) {
            this.closeEverything();
            break;
        }
    }
}

public void sendMessage(String messageToSend)
{
    try {
        this.bufferedWriter.write(messageToSend);
        this.bufferedWriter.newLine();
        this.bufferedWriter.flush();
    }catch (IOException e) {
        e.printStackTrace();
    }
}
```

Рисунок 15 - методи обробки запиту від клієнта.

2.3.4 Class ClientConnector

Допоміжний клас Сервера, який відловлює приєднання нових користувачів та присвоює їм об’єкт класу ClientHandler див п. 3.3.3 для подальшого відстежування надсилання запитів.

```
public void run()
{
    try {
        while (!this.server.serverSocket.isClosed()){
            Socket socket = this.server.serverSocket.accept();
            System.out.println("A new client connected!");
            ClientHandler clientHandler =  new ClientHandler(socket, this.server);
            this.server.clientHandlers.add(clientHandler);

            Thread thread = new Thread(clientHandler);
            thread.start();
        }

    } catch(IOException e){
        System.out.println(e.getMessage());
    }
}
```

Рисунок 16 - Функція обробки з’єднання клієнта з сервером

2.4 Реалізація другої структури

2.4.1 Інтерфейс

Інтерфейс був створений за допомогою фреймворку Flask. Flask - це фреймворк, який допомагає у створенні різних вебзастосунків мовою програмування Python, за допомогою набору інструментів Werkzeug та Jinja2. Чат був реалізований з дворівневою архітектурою, тобто, більшість логіки реалізовано на сервері, проте клієнтська частина, отримує результат запиту, обробляє його і вже після виводить на екран для користувача.

Інтерфейс та взаємодію реалізовано з використанням HTML та CSS. Для кращого функціонування, програму було викладено на сайт з хостингом для перевірки коректної роботи на різних пристроях та більшої мобільності, при запуску, бачимо інтерфейс з двома полями, а саме: username та roomID. Перше поле використовується для задання ім’я користувачу, яке будуть бачити інші користувачі, а інше поле відповідає за кімнату, в яку користувач хоче приєднатись.

Рисунок 17 - Інтерфейс при запуску

Кімнати можуть використовуватись, як діалоги або групові чати, при приєднанні користувачів, у чаті з’являється повідомлення про приєднання. При надсиланні повідомлень, відображається спочатку ім’я, потім сам зміст повідомлення.

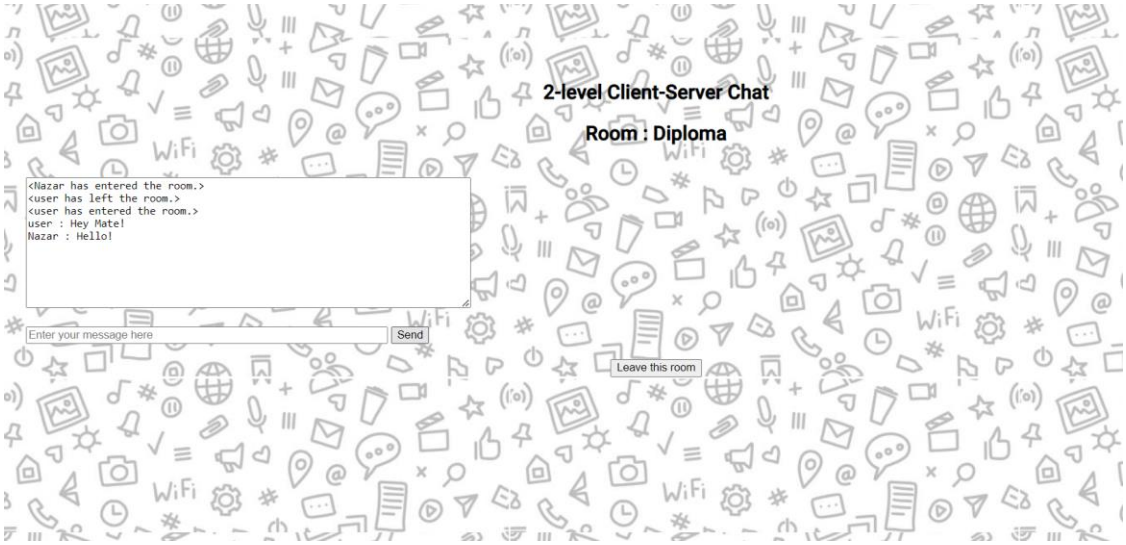

Рисунок 18 - Інтерфейс чату всередині кімнати

2.4.2 Розгортання

Для розгортання застосунку було використано інструменти платформи “pythonanywhere”.

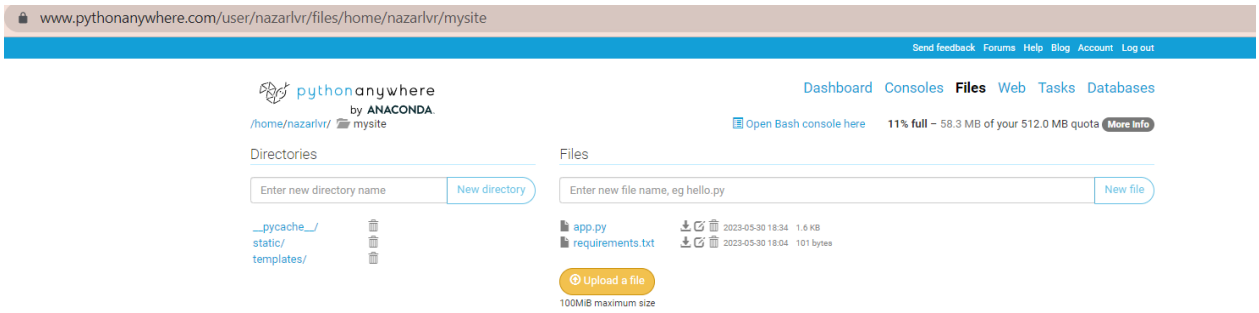

Рисунок 19 - Структура проєкту

Для цього, були виконанні наступні пункти:

- Створення акаунту на платформі.
- Створення нового веб застосунку з вибором Flask та версії Python 3.10.
- Завантаження файлів проєкту та створення тек під них, а саме: static та templates.
- Створення та завантаження нового файлу “requirements.txt”, де описано потрібні бібліотеки та їх версії.
- В консолі за допомогою команди встановлено усі вимоги, щодо бібліотек.
- Зміна конфігураційного файлу для коректної роботи.

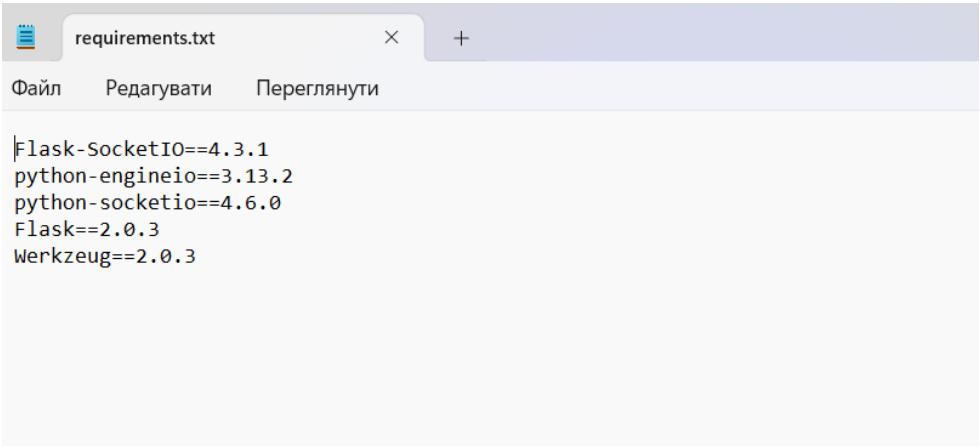

Рисунок 20 - Версії встановлених бібліотек

Після пророблених дій веб додаток почав працювати за посиланням - http://nazarlvr.pythonanywhere.com/.

2.5 Аналіз роботи

В ході дипломної роботи, було створено два додатки з різними архітектурами, а саме однорівневий додаток з використанням Java та складової бібліотеки Java Swing та дворівневий веб додаток з використанням Python та фреймворку Flask.

2.5.1 Аналіз використаних мов

Таблиця 2.5 - Порівняння мов програмування

Характеристика/Мова	Python + Flask	Java + JFrame
Інтерфейс	Створення інтерфейсу доволі легке та інтуїтивне, не потребує додаткових, глибоких знань, легко змінювати в процесі розробки.	Потребує глибшого розуміння для створення навіть базових речей.
Час відповіді	Додатки показали однакові результати в межах статистичної	Оскільки навантаження було дуже мале результати майже не

	похибки.	відрізняються.
Написання коду	На базовому рівні Python дуже легких у написанні коду, а Flask допомагає легко налаштувати веб додаток. В середньому при однакових вебзастосунках, Python потребує менше строк коду.	Java складніша у розумінні та написанні коду, проте є більш актуально для написання великих вебзастосунків, оскільки такий додаток інтегрується з більшою кількістю інших додатків, має велику базу зовнішніх бібліотек.
Продуктивність	Менше продуктивна при великій кількості запитів.	Java вважається більш продуктивною мовою. Якщо продукт буде містити велику кількість запитів, краще підійде дана мова.
Розгортання	Python вважається більш гнучким до розгортання. Для цього вебзастосунку потрібні лише інтерпретатор Python та деякі залежності.	Java потребує наявність JRE, що потребує додаткових зусиль для розгортання.

З таблиці 2.5 видно, що мову потрібно підбирати під кожен проєкт окремо, десь потрібна швидкість і простота, тоді краще реалізувати на Python, десь потрібна стабільність і потужність, тоді підійде Java.

2.5.2 Аналіз використаних архітектур

На Java була реалізована однорівнева архітектура, така структура підійде для внутрішніх обмінів інформації в локальній мережі з не дуже великим обсягом клієнтів, хоч запитів Java дозволяє обробити більше і швидкість буде краща, така архітектура дуже рідко використовується в силу слабкої масштабованості.

На Python була реалізована дворівнева архітектура, яка є більш популярною ніж попередня. Як видно з п. 2.4.2 проєкт доволі просто розгорнувся і надає

можливість приєднання клієнтів з різних місць та пристроїв, що дає більшу масштабованість ніж однорівнева, до того ж проєкт доволі легкий і лаконічний, що зберегло час його виконання та розгортання. Проте при великій кількості запитів, можуть бути затримки або некоректна робота, до того ж програми на Python менш інтегровані, що надалі може стати проблемою.

ВИСНОВКИ

У рамках дослідження клієнт-серверної архітектури було проведено аналіз різних типів архітектур та їх впливу на розробку програмного забезпечення. Отримані результати вказують на те, що вибір конкретної архітектурної моделі залежить від потреб проєкту, його масштабу та вимог до продуктивності, масштабованості, безпеки та розширюваності системи.

Дослідження показали, що клієнт-серверна архітектура є ефективним рішенням для багатьох типів додатків. Різні типи клієнт-серверних архітектур, такі як одноріvнева, дворівнева, трирівнева, багаторівнева, мікросервісна та розподілена, надають різні рівні функціональності, модульності та масштабованості.

Впровадження клієнт-серверної архітектури може принести значні переваги, включаючи полегшення розробки, тестування та підтримки системи, можливість масштабування окремих компонентів, повторне використання функціональних блоків, відокремлення завдань та покращену безпеку.

На основі отриманих результатів можна зробити висновок, що вибір певного типу клієнт-серверної архітектури повинен здійснюватись з урахуванням специфіки проєкту та його вимог. Кожен тип архітектури має свої переваги та недоліки, тому важливо здійснити обґрунтований вибір з урахуванням потреб та характеристик проєкту.

Дослідження вказують на доцільність подальшого вдосконалення клієнт-серверних архітектур з метою постійного пристосування до наростальних вимог і технологій. Розробники мають можливість використовувати різні типи архітектур для досягнення найкращих результатів в розробці програмного забезпечення.

Було розроблено та проаналізовано два проєкта на різних мовах програмування та з різними архітектурами. Також проведений аналіз, щодо доцільності застосування кожної з мов, саме до проєктів з клієнт-серверною архітектурою, було досліджено переваги та недоліки реалізованих продуктів та оптимальні застосування їх у реальних проєктах. Також були дослідженні

інструменти та їх доцільні застосування. Кожна з реалізацій є кращою, під конкретні умови, тому важливим кроком у виборі мови, фреймворків та архітектури є правильний аналіз вимог та умов для проєкту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Berson A. Client/server architecture / Alex Berson. – Ann Arbor: McGraw-Hill, 1996. – 569 с.
2. Goodyear M. Enterprise System Architectures / Mark Goodyear., 2017. – 978 с.
3. Nieh J. A Comparison of Thin-Client Computing Architectures / J. Nieh, J. Yang, N. Novik. – New York: Department of Computer Science, Columbia University, 2000.
4. Client–server model [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/client-server-model/.
5. Request–response [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@jen_strong/the-request-response-cycle-of-the-web-1b7e206e9047.
6. C# Vs C++ And C# Vs Java – Explore The Key Differences [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/csharp-vs-cpp-vs-java/.
7. Features of Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/features-of-java.
8. IntelliJ IDEA overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html.
9. How to Make Frames (Main Windows) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html.
10. Tutorial — Flask Documentation (2.3.x) [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/2.3.x/tutorial/.
11. Token ring [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchnetworking/definition/Token-Ring.

ДОДАТОК А

Код до першого продукту:

Client.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
public class Client extends JFrame {
   private static final String SERVER_HOST = "localhost";
   private static final int SERVER_PORT = 1234;
   private Socket socket;
   private BufferedReader bufferedReader;
   private BufferedWriter bufferedWriter;
   private JTextField jtfMessage;
   private JTextField jtfName;
   private JTextArea jtaTextAreaMessage;
   private String clientName;
   private boolean f;
   public Client(){
      f = false;
      try
      {
         this.socket = new Socket(SERVER_HOST, SERVER_PORT);
         bufferedReader = new BufferedReader(new
InputStreamReader(this.socket.getInputStream()));
         bufferedWriter = new BufferedWriter(new
OutputStreamWriter(this.socket.getOutputStream()));
      } catch (IOException e) {
         e.printStackTrace();
      }
```

```
setBounds(600, 300, 600, 500);
setTitle("Client");
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
jtaTextAreaMessage = new JTextArea();
jtaTextAreaMessage.setEditable(false);
jtaTextAreaMessage.setLineWrap(true);
JScrollPane jsp = new JScrollPane(jtaTextAreaMessage);
add(jsp, BorderLayout.CENTER);
JPanel bottomPanel = new JPanel(new BorderLayout());
add(bottomPanel, BorderLayout.SOUTH);
JButton jbSendMessage = new JButton("Set username");
bottomPanel.add(jbSendMessage, BorderLayout.EAST);
jtfMessage = new JTextField("");
bottomPanel.add(jtfMessage, BorderLayout.CENTER);
jtfName = new JTextField("Username:");
jtfName.setEditable(false);
bottomPanel.add(jtfName, BorderLayout.WEST);
jbSendMessage.addActionListener(e -> {
   try {
      if (!f && !jtfMessage.getText().isEmpty())
      {
         String messageStr = jtfMessage.getText();
         this.clientName = messageStr;
         bufferedWriter.write(messageStr);
         bufferedWriter.newLine();

         bufferedWriter.flush();
         jtfMessage.setText("");
         jtfName.setText(messageStr);
         jtfMessage.grabFocus();
         jbSendMessage.setText("Send");
         f = true;
      }
      else if (!jtfMessage.getText().trim().isEmpty())
```

```
            {
                String messageStr = jtfMessage.getText();
                jtaTextAreaMessage.append(messageStr + '\n');
                bufferedWriter.write(messageStr);
                bufferedWriter.newLine();
                bufferedWriter.flush();
                jtfMessage.setText("");
                jtfMessage.grabFocus();
            }
        } catch (IOException ex) {
            closeEverything();
        }
    });
    jtfMessage.addFocusListener(new FocusAdapter() {
        @Override
        public void focusGained(FocusEvent e) {
            jtfMessage.setText("");
        }
    });
    new Thread(() -> {
        try {
            while (socket.isConnected())
            {
                String inMes = bufferedReader.readLine();
                jtaTextAreaMessage.append(inMes);
                jtaTextAreaMessage.append("\n");
            }
        } catch (Exception e)
        {
            closeEverything();
        }
    }).start();
    setVisible(true);
}
```

```
    public void closeEverything() {
        try {
            if (bufferedReader != null) {
                bufferedReader.close();
            }
            if (bufferedWriter != null) {
                bufferedWriter.close();
            }
            if (socket != null) {
                socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args)
    {
        Client client = new Client ();
    }
}
```

Server.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.net.ServerSocket;
import java.util.ArrayList;
import java.util.concurrent.LinkedBlockingQueue;

public class Server{
    public boolean isRunning;
    public ServerSocket serverSocket;
    public ArrayList<ClientHandler> clientHandlers = new ArrayList<>();
    public Server(ServerSocket serverSocket)
```

```
{
    this.serverSocket = serverSocket;
}
public LinkedBlockingQueue<String> messageQueue;

public void startServer() throws InterruptedException {

    this.isRunning = true;
    this.messageQueue = new LinkedBlockingQueue<>();
    ClientConnector clientConnector = new ClientConnector(this);
    Thread connectorThread = new Thread(clientConnector);
    connectorThread.start();
    System.out.println("Server Started!");

    while (this.isRunning)
    {
        if (!this.messageQueue.isEmpty())
        {
            this.processMessage(this.messageQueue.take());
        }

        Thread.sleep(500);
    }
}

public void processMessage(String msg)
{
    try {
        String mes = "User ";
        FileWriter out = new FileWriter("D:\\parallel\\paralelproject2\\log.txt", true);
        for (int i = 0; i < msg.length(); i++){
            if (msg.charAt(i) != ':'){
                mes = mes + msg.charAt(i);
            } else{
```

```
                    mes = mes + " send a message: \"";
                }
            }
            mes +="\"";
            out.write(mes + '\n');
            out.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        System.out.println(msg);
        msg = msg.trim();
        String[] ss = msg.split(": ", 2);
        String username = ss[0], text = ss[1];
        ClientHandler clientHandler = findClient(username);

        if (text.charAt(0) == '/')
        {
            ss = text.split(" ", 2);
            String command = ss[0];
            if (ss.length > 1)
                text = ss[1];

            if (clientHandler != null)
            {
                if (command.equals("/help"))
                {
                    clientHandler.sendMessage("Server: \"/help\" \"/userlist\" \"/msg username
message\"");
                }
                else if(command.equals("/userlist"))
                {
                    StringBuilder sb = new StringBuilder();

                    for (ClientHandler ch : this.clientHandlers)
```

```
                sb.append(", " + ch.clientUsername);

            sb.delete(0,2);

            clientHandler.sendMessage(sb.toString());
        }
        else if(command.equals("/msg"))
        {
            ss = text.split(" ", 2);
            String address = ss[0];
            text = ss[1];
            ClientHandler clientHandler1 = findClient(address);

            if (clientHandler1 != null)
            {
                clientHandler1.sendMessage(username + "(Private message): " + text);
            }
            else
            {
                clientHandler.sendMessage("Server: No user with name \"" +  address + "\"");
            }
        }
        else
        {
            clientHandler.sendMessage("Server: Unknown command \"" +  command + "\"");
        }
      }
   }
   else
   {
      broadcastMessage(msg, username);
   }
}
```

```
public void broadcastMessage(String messageToSend, String clientUsername) {
    for (ClientHandler clientHandler : this.clientHandlers) {
        try {
            if (!clientHandler.clientUsername.equals(clientUsername)) {
                clientHandler.bufferedWriter.write(messageToSend);
                clientHandler.bufferedWriter.newLine();
                clientHandler.bufferedWriter.flush();
            }
        }catch (IOException e) {
            clientHandler.closeEverything();
        }
    }
}

public void broadcastMessage(String messageToSend) {
    this.broadcastMessage(messageToSend, null);
}

public ClientHandler findClient(String username)
{
    for (ClientHandler clientHandler : this.clientHandlers)
    {
        if (clientHandler.clientUsername.equals(username))
            return clientHandler;
    }

    return null;
}

public void closeServerSocket()
{
    try {
        if (serverSocket != null) {
            serverSocket.close();
```

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

    public static void main(String[] args) throws IOException{
        ServerSocket serverSocket = new ServerSocket(1234);
        Server server = new Server(serverSocket);
        try
        {
            server.startServer();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            server.isRunning = false;
        }
    }
}
```

ServerHandler

```
import java.io.*;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
public class ClientHandler implements Runnable {
    private Server server;
    private Socket socket;
    private BufferedReader bufferedReader;
```

```
        public BufferedWriter bufferedWriter;
        public String clientUsername;
        public ClientHandler(Socket socket, Server srv) {
            try {
                this.socket = socket;
                this.server = srv;
                this.bufferedWriter = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
                this.bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                this.clientUsername = bufferedReader.readLine();
                this.server.messageQueue.add("Server: " + clientUsername + " has entered the chat!");
            } catch (IOException e) {
                this.closeEverything();
            }
        }
        @Override
        public void run()
        {
            String messageFromClient;
            while (socket.isConnected()) {
                try {
                    messageFromClient = bufferedReader.readLine();
                    this.server.messageQueue.add(this.clientUsername + ": " + messageFromClient);
                } catch (IOException e) {
                    this.closeEverything();
                    break;
                }
            }
        }
        public void sendMessage(String messageToSend)
        {
            try {
                this.bufferedWriter.write(messageToSend);
```

```
            this.bufferedWriter.newLine();
            this.bufferedWriter.flush();
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void removeClientHandler() {
        this.server.clientHandlers.remove(this);
        this.server.broadcastMessage("Server: " + clientUsername + " has left the chat!");
    }
    public void closeEverything() {
        removeClientHandler();
        try {
            if (this.bufferedReader != null) {
                this.bufferedReader.close();
            }
            if (this.bufferedWriter != null) {
                this.bufferedWriter.close();
            }
            if (this.socket != null) {
                this.socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ClientConnector.java

```
import java.io.IOException;
import java.net.Socket;

public class ClientConnector implements Runnable
```

```
{
    public Server server;

    public ClientConnector(Server srv)
    {
        this.server = srv;
    }

    public void run()
    {
        try {
            while (!this.server.serverSocket.isClosed()){
                Socket socket = this.server.serverSocket.accept();
                System.out.println("A new client connected!");
                ClientHandler clientHandler =  new ClientHandler(socket, this.server);
                this.server.clientHandlers.add(clientHandler);

                Thread thread = new Thread(clientHandler);
                thread.start();
            }

        } catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
}
```