Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка веб-застосунку для автоматизації обміну даними між користувачами»

Виконав: студент групи К23-1м

Спеціальність 122 Комп'ютерні науки

Моісеєнко О.О.
(прізвище та ініціали)

Керівник д.е.н., проф. Корнєєв М.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Дніпровський державний технічний університет
(місце роботи)

в.о. завідувача кафедри програмного забезпечення систем
(посада)

к.т.н., доц. Жульковський О.О.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Моісеєнко О.О. Розробка веб-застосунку для автоматизації обміну даними між користувачами.

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

Об'єктом дослідження є процес обміну даними в корпоративному середовищі.

Предмет дослідження – методи та засоби створення веб-застосунків для автоматизації інформаційних процесів.

Метою роботи є автоматизація обміну даними між користувачами з використанням клієнт-серверної архітектури для підвищення ефективності взаємодії та забезпечення безпеки даних.

Кваліфікаійна робота присвячена створенню веб-застосунку для автоматизації обміну даними між користувачами в корпоративному середовищі. У роботі проведено аналіз сучасних інструментів для розробки клієнт-серверних систем, обґрунтовано вибір технологій, зокрема використання C#, ASP.NET Core. Запропонований веб-застосунок забезпечує можливість зберігання, редагування та передачі даних з контролем доступу. Було проведено тестування програмного продукту, яке підтвердило стабільність роботи системи та відповідність вимогам безпеки.

Практична цінність роботи полягає у створенні веб-застосунку, який може бути використаний у корпоративних середовищах для оптимізації процесів обміну інформацією, підвищення безпеки та ефективності управління даними.

Ключові слова: веб-застосунок, обмін даними, клієнт-серверна архітектура, ASP.NET Core, C#, автоматизація, корпоративне середовище.

ABSTRACT

Moiseenko O.O. Development of a Web Application for Automating Data Exchange Between Users

This project for obtaining a master's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2025.

The object of the study is the process of data exchange in a corporate environment.

The subject of the study is the methods and tools for developing web applications for automating information processes.

The aim of the work is to develop a web application for automating data exchange between users using a client-server architecture to improve interaction efficiency and ensure data security.

The thesis is devoted to the development of a web application for automating data exchange between users in a corporate environment. The study includes an analysis of modern tools for developing client-server systems and justifies the choice of technologies such as C#, ASP.NET Core, and SQL Server. The proposed web application provides data storage, editing, and transfer capabilities with access control. The developed software was tested, confirming the system's stability and compliance with security requirements.

The practical significance of the work lies in creating a web application that can be used in corporate environments to optimize information exchange processes, enhance security, and improve data management efficiency.

Keywords: web application, data exchange, client-server architecture, ASP.NET Core, C#, SQL Server, automation, corporate environment.

ЗМІСТ

ВСТУП

Сучасне суспільство неможливо уявити без різноманітних засобів комунікації, серед яких особливе місце займають цифрові технології обміну даними. Відправка електронних листів, обмін текстовими повідомленнями, відеозв'язок, аудіоконференції та онлайн-чати стали невід'ємною частиною повсякденного життя. Завдяки розвитку інтернет-технологій спілкування стало доступним як у синхронному режимі в реальному часі, так і в асинхронному форматі, що дозволяє користувачам взаємодіяти у зручний для них час.

Особливо актуальним питанням сьогодення є організація централізованого зберігання даних та обмін інформацією в корпоративному сегменті. Централізоване сховище даних дозволяє користувачам додавати, видаляти, змінювати та запитувати інформацію через різноманітні клієнтські додатки, забезпечуючи контрольований та безпечний доступ до ресурсів компанії.

Метою дослідження є створення веб-застосунку з використанням клієнт-серверної архітектури для автоматизації обміну даними між користувачами. Запропоноване рішення має забезпечити швидкий та зручний обмін інформацією, підтримку базових функцій управління даними, а також гарантувати захист даних під час передачі.

Методи дослідження – аналіз інформаційних технологій для створення клієнт-серверних систем, проектування архітектури веб-застосунку, методи програмування та розробки клієнт-серверних веб-додатків.

Основні завдання дослідження:

1. Проаналізувати сучасні технічні засоби для розробки веб-застосунків.
2. Обґрунтувати вибір клієнт-серверної архітектури.
3. Спроєктувати та реалізувати веб-застосунок з використанням сучасних інструментів.

4. Провести тестування створеного програмного забезпечення.

Об'єктом дослідження є розробка програмного забезпечення для обміну даними в корпоративному середовищі.

Предметом дослідження є апаратно-програмне забезпечення для створення веб-застосунків.

Практична значимість роботи полягає у створенні веб-застосунку, що дозволить автоматизувати процес обміну даними між користувачами, підвищити ефективність взаємодії в корпоративному середовищі та забезпечити безпечний доступ до інформаційних ресурсів.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 77 сторінок тексту, 46 рисунків та переліку літературних джерел з 15 найменувань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.

1.1 Аналіз предметної області

Необхідність прискорення обміну значними обсягами даних та різноманітною інформацією між користувачами, які знаходяться на значній відстані один від одного, спричинила появу та стрімкий розвиток мережевих технологій. Однією з ключових сучасних технологій у цій галузі є мережі типу *peer-to-peer* (P2P), що в українській термінології визначаються як пірингові мережі.

Визначення та принцип функціонування

Поняття *peer* у контексті комп’ютерних мереж означає вузол або клієнта, який, взаємодіючи з іншими вузлами, формує колективну мережу. Така мережа, хоча й може здаватися нестабільною та тимчасовою, насправді здатна забезпечити високий рівень стійкості та надійності за умови залучення великої кількості користувачів. Це досягається завдяки прямій взаємодії між пристроями без необхідності використання централізованого сервера.

Класифікація комп'ютерних мереж

Комп’ютерні мережі класифікуються на дві основні категорії:

- Однорангові мережі (peer-to-peer networks)
- Мережі з виділеним сервером (client-server networks)

Цей поділ має принципове значення, оскільки структура мережі безпосередньо впливає на її функціональність, ефективність та сфери застосування.

Однорангові мережі (P2P) характеризуються відсутністю централізованого керування мережею. Кожен пристрій (вузол) одночасно виконує функції як клієнта, так і сервера. Це означає, що користувачі самостійно надають свої ресурси (файли, обчислювальні потужності тощо) для спільного доступу та отримують доступ до ресурсів інших учасників мережі [1].

Мережі з виділеним сервером функціонують за принципом централізованого зберігання даних та керування доступом. Сервер виконує роль центрального вузла, який обслуговує запити клієнтів, що з'єднуються з ним для отримання ресурсів або послуг.

Переваги однорангових мереж

Протягом останніх десятиліть однорангові мережі здобули значну популярність завдяки низці переваг порівняно з архітектурою клієнт-сервер. Основними перевагами є:

1. Масштабованість:

Однорангові мережі уникають створення так званого «вузького місця» у вигляді центрального сервера, оскільки передача даних здійснюється безпосередньо між кінцевими вузлами. Це забезпечує можливість ефективного обміну великими обсягами даних, навіть у масштабних мережах.

2. Стійкість до відмов:

Завдяки відсутності єдиного центрального сервера працездатність мережі не залежить від доступності одного вузла. У разі виходу з ладу окремих пристроїв функціонування мережі загалом зберігається.

3. Приватність:

Дані зберігаються локально на пристроях користувачів, а обчислювальні операції також можуть виконуватися без залучення сторонніх серверів. Це знижує ризик витоку конфіденційної інформації та підвищує контроль над власними даними.

4. Простота розгортання:

Однорангові мережі не потребують складних налаштувань або спеціалізованих серверів, що робить їх зручними для самостійного розгортання та експлуатації.

5. Економічність:

Відсутність потреби у централізованому сервері суттєво знижує витрати на технічне обладнання та його обслуговування [3].

Сфери застосування однорангових мереж

Враховуючи наведені переваги, однорангові мережі знаходять застосування у різних сферах:

- Децентралізований обмін файлами:

P2P-мережі широко використовуються для обміну великими обсягами даних без залучення централізованих серверів. Прикладом є протокол BitTorrent, що дозволяє завантажувати файли частинами від різних користувачів одночасно.

- Розподілені обчислення:

Однорангові мережі можуть використовуватися для об'єднання обчислювальних ресурсів великої кількості пристроїв для вирішення складних задач, таких як наукові обчислення або обробка великих масивів даних [2].

- Криптовалютні системи:

Блокчейн-технології, зокрема Bitcoin та Ethereum, функціонують за принципом однорангових мереж, де кожен вузол зберігає копію реєстру транзакцій, забезпечуючи децентралізовану перевірку даних.

Недоліки однорангових мереж

Попри численні переваги, однорангові мережі мають також низку недоліків, які обмежують їх використання у певних сценаріях:

1. Захист персональних даних:

Відсутність централізованого управління ускладнює забезпечення належного рівня захисту персональних даних відповідно до чинного законодавства.

2. Складність адміністрування:

Оскільки кожен вузол є одночасно сервером та клієнтом, адміністрування мережі вимагає окремого налаштування кожного пристрою, що може бути складним для великих систем.

3. Зниження продуктивності:

При спільному використанні обчислювальних ресурсів та пропускної здатності мережі можливе зниження продуктивності окремих вузлів.

4. Безпекові ризики:

Відсутність централізованого контролю призводить до того, що будь-який учасник мережі може потенційно стати загрозою безпеці. Забезпечення автентифікації, шифрування даних та захисту від атак у такій мережі є складним завданням [4].

1.2 Аналіз аналогів представлених на ринку

Сьогодні існує значний вибір програмного забезпечення для мережевої комунікації, які відрізняються функціональними можливостями та сферами застосування. Однак на сучасному ринку все ще відчувається нестача якісних і зручних рішень, здатних задовольнити потреби більшості користувачів. Далі буде розглянуто найпопулярніші програми, що підтримують функції спільного спілкування, включаючи як текстовий, так і графічний обмін інформацією.

1.2.1 Network Assistant

Network Assistant є програмним забезпеченням для мережевої комунікації, розробленим для використання у домашніх та невеликих офісних локальних мережах. Програма функціонує за принципом формування єдиного мережевого середовища, створеного всіма пристроями, на яких вона запущена. У межах цього середовища комп’ютери можуть обмінюватися інформацією, використовуючи такі засоби комунікації, як багатоканальний чат, спільна дошка для малювання, обмін текстовими повідомленнями, система оголошень тощо.

Програмне забезпечення не потребує підключення до глобальної мережі Інтернет або використання виділеного сервера, оскільки передача даних відбувається за допомогою протоколу UDP, зокрема через широкомовні та мультикаст-повідомлення. Проте лише користувачі з адміністративними правами можуть здійснювати конфігурацію параметрів безпеки, включаючи

налаштування міжмережевого екрану (firewall). Це може створювати певні обмеження для користувачів без відповідних прав доступу, які не зможуть повноцінно використовувати програму [5].

Network Assistant підтримує кілька основних функцій, що забезпечують комплексне мережеве спілкування:

1. Багатокористувацький чат

Програма дозволяє створювати різноманітні типи каналів для спілкування:

- Загальні канали — доступні для всіх користувачів мережі.
- Приватні канали — призначені для особистих діалогів між двома користувачами.
- Парольовані канали — обмежують доступ до каналу за допомогою пароля, створюючи можливість для обговорень у вузькому колі осіб.

2. Спільна дошка для малювання

Однією з функціональних можливостей програми є багатокористувацька дошка для малювання, яка є спільною для всіх підключених користувачів мережі. Вона оснащена базовим набором інструментів, що включає:

- Олівець.
- Інструмент заливки.
- Текстові вставки.
- Геометричні фігури (лінія, прямокутник, еліпс).

Для підвищення точності малювання передбачена функція прив’язки до сітки. Однак функціональні можливості цієї дошки є обмеженими, оскільки відсутні такі базові інструменти, як гумка та функція скасування останньої дії, що ускладнює виправлення помилок під час створення зображень.

3. Управління статусами користувачів

Кожен користувач Network Assistant може налаштувати власний статус, що визначає спосіб отримання повідомлень, звукові сповіщення та реакції програми. Серед доступних налаштувань є можливість додати персональні

дані, включаючи ім’я, прізвище, контактні дані та зображення профілю. Програма підтримує як ручну, так і автоматичну зміну статусу залежно від певних подій, зокрема:

- Відключення пристрою від мережі.
- Проста роботи пристрою.
- Запуск зберігача екрана.

4. Додаткові можливості

Програмне забезпечення також підтримує такі функції:

- Передача файлів між користувачами мережі.
- Перегляд знімків екрана інших пристроїв.
- Обмін даними з буфера обміну.
- Моніторинг активності локальних або віддалених пристроїв.
- Можливість вибору мови інтерфейсу.

Network Assistant має кілька ключових переваг, які роблять його ефективним рішенням для комунікації в локальних мережах:

- Комплексний набір функцій: програма підтримує широкий спектр можливостей для спілкування, включаючи текстові повідомлення, передачу файлів, спільну дошку для малювання та управління статусами.
- Гнучкість у спілкуванні: можливість створення різних типів чат-каналів дозволяє користувачам спілкуватися у відкритих або приватних групах відповідно до потреб.
- Локальна автономність: програма не потребує підключення до Інтернету, що дозволяє використовувати її в закритих мережах, де глобальний доступ недоступний або небажаний.
- Простота використання: програмне забезпечення не вимагає складної конфігурації для базового використання, що знижує технічні бар'єри для користувачів.

Попри значний функціонал, Network Assistant має низку недоліків, які можуть обмежувати його ефективність:

- Обмежений функціонал інструментів малювання –відсутність базових інструментів, таких як гумка та функція скасування останньої дії, значно знижує зручність користування спільною дошкою для малювання.
- Використання лише в локальних мережах – програма не підтримує комунікацію через Інтернет, що обмежує її застосування для віддаленого спілкування між користувачами.
- Адміністративні обмеження – лише користувачі з правами адміністратора можуть налаштовувати мережеві параметри, включаючи налаштування firewall, що може бути незручним у деяких організаціях.

Network Assistant є зручним та функціональним програмним забезпеченням для комунікації у локальних мережах, що пропонує широкий спектр можливостей для обміну даними, включаючи текстові повідомлення, спільну дошку для малювання, передачу файлів та управління статусами користувачів. Завдяки простоті налаштування та автономній роботі без підключення до Інтернету, програма може бути ефективним інструментом для комунікації у закритих мережах, таких як офісні або навчальні заклади.

1.2.2 Imagination Cubed

Imagination Cubed є альтернативним програмним продуктом для спільного малювання, реалізованим як flash-додаток, інтегрований у веб-сторінку. Це забезпечує можливість доступу до інструменту з будь-якого пристрою, підключеного до Інтернету, за умови наявності браузера із підтримкою Flash. Щоб розпочати спільну роботу з іншим користувачем, необхідно надіслати йому запрошення через електронну пошту або службу миттєвих повідомлень AOL. Це ускладнює використання програми для швидких комунікацій, оскільки процес встановлення зв'язку займає додатковий час. Доступ до спільної дошки для малювання можливий лише за умови отримання правильного посилання від іншого користувача.

Imagination Cubed пропонує значно ширший набір інструментів для малювання, ніж Network Assistant, включаючи:

- Олівець для вільного малювання.
- Геометричні фігури: лінія, прямокутник, еліпс тощо.
- Штампи різних форм.
- Текстові вставки.

Кожен інструмент має можливості налаштування, зокрема вибір розміру, кольору та форми, що забезпечує більше гнучкості під час створення графічного контенту. Інтерфейс програми орієнтований на спрощення процесу малювання, однак текстовий чат реалізований незручно — він розташований у маленькому вікні, що ускладнює текстове спілкування.

Програма передбачає кілька варіантів збереження та поширення створених зображень:

- Надсилання готового малюнка електронною поштою.
- Друк безпосередньо з веб-інтерфейсу.

Цікавим функціональним доповненням є можливість перегляду покадрового створення зображення. Ця функція дозволяє крок за кроком відтворити процес формування малюнка, що може бути корисним для навчання або демонстрації послідовності роботи над зображенням.

Imagination Cubed підтримує одночасне використання до чотирьох користувачів, проте вже при підключенні трьох учасників спостерігається значне зниження продуктивності, навіть за умов високошвидкісного підключення до Інтернету. До проблем, що виникають при збільшенні кількості учасників, належать:

- Зникнення фрагментів ліній.
- Деформація кривих.
- Затримка у відображенні графічних об'єктів.

Приміром, під час 10-хвилинного сеансу з помірною активністю трьох користувачів програма генерує близько 100 КБ мережевого трафіку на кожного учасника, що може негативно впливати на стабільність підключення.

Програма має низку ключових переваг, зокрема:

- Широкий набір інструментів для малювання.
- Легкий доступ через веб-браузер без потреби встановлення додаткового програмного забезпечення.
- Відсутність необхідності у налаштуванні міжмережевого екрану (firewall).

Попри численні переваги, програмний продукт має й значні недоліки:

- Складний процес підключення: для початку сеансу необхідно отримати правильне посилання через електронну пошту або службу миттєвих повідомлень, що ускладнює швидке підключення.
- Обмежена кількість користувачів: підтримується лише до чотирьох учасників, що обмежує можливість використання для масштабних проєктів.
- Незручний текстовий чат: малий розмір вікна та його неінтуїтивне розташування ускладнюють текстову комунікацію.
- Проблеми з продуктивністю: при підключенні декількох користувачів можливі затримки у відображенні об'єктів та зниження якості малювання.

Imagination Cubed є зручним інструментом для спільного малювання в онлайн-середовищі, орієнтованим на творчі та розважальні цілі. Широкий вибір графічних інструментів та простота доступу через веб-браузер роблять його ефективним для невеликих творчих сесій або спільної роботи над простими графічними проєктами. Водночас обмеження щодо кількості користувачів, продуктивності та складності підключення унеможливлюють використання програми для професійних потреб або масштабних колективних сесій.

1.2.3 CollabEdit

CollabEdit є веб-застосунком, орієнтованим на організацію ефективної спільної роботи над текстовими документами в режимі реального часу. Завдяки можливості одночасного редагування одним документом кількома учасниками, цей інструмент стає незамінним для командного співробітництва, навчальних проєктів, колективного написання коду та інших форм взаємодії, де важливий синхронний обмін ідеями. Головною метою цього аналізу є дослідження функціональних можливостей CollabEdit з урахуванням його застосування у сфері колективної діяльності.

Ключовою особливістю CollabEdit є функція синхронного редагування тексту, яка дозволяє всім учасникам бачити зміни в документі одразу після їх внесення. Це сприяє підвищенню ефективності співпраці, оскільки зникає необхідність у багаторазовій передачі файлів між учасниками або синхронізації правок вручну. Така технологія базується на використанні алгоритмів оптимальної синхронізації, що мінімізують ризик конфліктів під час внесення змін кількома користувачами одночасно. У випадку, коли конфлікти все ж виникають, програма автоматично застосовує алгоритми об’єднання правок, зберігаючи логічну цілісність документа та точність внесених змін. Цей підхід особливо корисний для середовищ, де важливе динамічне оновлення інформації, як-от під час технічних обговорень або роботи над програмним кодом.

Програма підтримує широкий спектр текстових форматів, що включає як звичайний текст, так і програмний код з підсвічуванням синтаксису для різних мов програмування, зокрема Python, JavaScript, C++. Це дозволяє використовувати CollabEdit не лише для стандартного документування, а й для технічних завдань, таких як спільне програмування, перевірка коду або підготовка технічної документації.

Важливою перевагою є інтеграція CollabEdit з популярними інструментами для спільної роботи, такими як Slack, Trello, GitHub, що робить

його частиною цілісного робочого середовища для управління проєктами. Завдяки цьому учасники можуть взаємодіяти не лише через текстове поле редактора, а й у межах інших платформ для обговорення задач та контролю за їхнім виконанням. Вбудований текстовий чат у CollabEdit забезпечує безпосередню комунікацію між учасниками, дозволяючи обговорювати внесені зміни або погоджувати наступні етапи роботи прямо в інтерфейсі редактора. Додаткова функція коментування тексту дає можливість залишати текстові примітки до конкретних фрагментів документа, що полегшує рецензування та обговорення правок.

Функція збереження історії версій дозволяє відстежувати кожну зміну, зроблену в документі, а також повертатися до попередніх редакцій, якщо виникає потреба відновити певний етап роботи. Це особливо корисно в умовах довгострокових проєктів або під час редагування складних текстів, де необхідно фіксувати кожний внесений етап змін. Крім того, адміністратори документів можуть призначати різні рівні доступу для користувачів, обмежуючи або дозволяючи редагування або лише перегляд документа, що гарантує додатковий контроль за збереженням інформації.

CollabEdit демонструє універсальність і адаптивність у використанні. Його активно застосовують розробники програмного забезпечення для спільної розробки коду, студенти та викладачі для групових навчальних проєктів, письменники та редактори для колективного редагування текстів, а також бізнес-команди для створення звітів, презентацій та інших робочих документів.

Таким чином, CollabEdit — це не просто текстовий редактор для спільної роботи, а комплексний інструмент для ефективної координації колективної діяльності, що поєднує в собі зручність редагування в реальному часі, багатофункціональність для різних типів документів та інтеграцію з іншими популярними платформами для управління робочим процесом. Завдяки таким можливостям програма стає потужним рішенням для команд, які прагнуть

покращити продуктивність та ефективність у роботі з текстовими матеріалами [6].

1.2.4 Confluence

Confluence — це інноваційний веб-застосунок для організації колективної роботи, розроблений компанією *Atlassian*. Його основне призначення полягає у створенні, редагуванні та управлінні документацією в командному середовищі, що робить його ефективним інструментом для координації спільної діяльності як у малих групах, так і у великих корпоративних структурах. Програмний продукт сприяє прозорій комунікації, покращенню управління знаннями та оптимізації робочих процесів, забезпечуючи централізовану платформу для взаємодії.

Основною функцією Confluence є створення та спільне редагування сторінок у режимі реального часу. Це дозволяє користувачам одночасно працювати над документами, додаючи текст, графіку, відеофайли та інші мультимедійні елементи, створюючи інтерактивний контент для всебічного висвітлення проектних задач. Всі внесені зміни синхронізуються миттєво, що дозволяє уникнути дублювання даних або втрати важливої інформації. Автоматичне збереження правок підвищує надійність роботи з документами, знижуючи ризики технічних збоїв або втрат даних.

Зручна система управління контентом реалізована через концепцію *просторів* (spaces). Простори — це структуровані інформаційні блоки, які можуть відповідати конкретним проєктам, відділам або тематичним напрямкам діяльності компанії. У межах кожного простору можна створювати ієрархічну структуру сторінок, що забезпечує зручний пошук та доступ до потрібної інформації. Цей підхід сприяє систематизації даних та ефективній організації корпоративної бази знань.

Однією з ключових переваг Confluence є його інтеграція з іншими продуктами компанії *Atlassian*, зокрема Jira та Trello. Завдяки цьому

платформа забезпечує безперервний робочий процес, дозволяючи користувачам одночасно керувати завданнями, документами та проектами в єдиному інформаційному просторі. Крім того, система підтримує інтеграцію з зовнішніми сервісами через API, що робить її гнучким рішенням для організацій із комплексною IT-інфраструктурою.

Confluence також передбачає функціональність коментування та обговорення змін безпосередньо на сторінках. Користувачі можуть залишати коментарі до конкретних фрагментів документа або до всього документа загалом, що сприяє детальному обговоренню внесених правок і покращує взаєморозуміння між членами команди. Сповіщення про нові коментарі та правки сприяють своєчасному інформуванню учасників про зміни в проекті, що робить комунікацію більш оперативною та прозорою.

Важливою складовою функціоналу Confluence є система збереження версій документів. Вона дозволяє фіксувати кожну зміну, здійснену в документі, та надає можливість повернутися до попередніх редакцій за необхідності. Це особливо корисно для довгострокових проєктів, де важливо зберігати історію правок, аналізувати етапи роботи та контролювати розвиток документації на різних стадіях її створення.

Безпека даних у Confluence реалізована через гнучку систему налаштувань доступу. Адміністратори можуть призначати різні рівні прав доступу до документів: від повного редагування до обмеженого перегляду. Це дозволяє контролювати доступ до конфіденційної інформації, обмежуючи її використання лише авторизованими користувачами або командами. Такий підхід забезпечує високий рівень інформаційної безпеки в корпоративному середовищі.

Confluence є універсальним інструментом, який знаходить застосування у широкому спектрі сфер. Його активно використовують:

- Розробники програмного забезпечення для документування технічних специфікацій, API та результатів тестування.

- Бізнес-команди для управління проектною документацією, створення внутрішніх політик та зберігання звітів.
- Навчальні заклади для створення освітніх матеріалів, організації дистанційного навчання та обміну знаннями між викладачами та студентами.
- Креативні агентства та дизайнери для організації спільної роботи над концепціями, творчими проєктами та візуальними матеріалами.

Завдяки широкому набору функцій, зокрема інтеграції з популярними інструментами, гнучкому управлінню доступом та можливості коментування в реальному часі, Confluence є одним із найбільш потужних рішень для організації колективної роботи. Його здатність забезпечувати централізовану платформу для спільного створення, зберігання та управління інформацією робить його ідеальним вибором для компаній, які прагнуть підвищити ефективність командної співпраці, підтримувати прозору комунікацію та покращити управління знаннями [7].

1.3 Висновок до першого розділу

У даному розділі було проведено дослідження принципів та архітектур однорангових (P2P) мереж, зокрема їх функціонування, класифікації та основних характеристик. Проаналізовано ключові підходи до побудови таких мереж, включаючи концепцію децентралізованого обміну даними, масштабованості та стійкості до відмов. Окрему увагу приділено перевагам однорангових мереж, таким як відсутність єдиного точки відмови, підвищена приватність даних та зниження витрат на інфраструктуру.

Особливу увагу було приділено аналізу програмних аналогів, включаючи Network Assistant, Imagination Cubed, CollabEdit та Confluence. Було розглянуто функціональні можливості цих рішень, такі як спільне редагування даних, текстова комунікація, обмін файлами та підтримка командної роботи, а також їх обмеження, зокрема складність адміністрування та питання безпеки.

Метою дослідження є визначення ключових принципів та вимог до створення системи мережевої взаємодії на основі однорангових мереж, здатної забезпечити стабільний обмін даними та зручність у використанні.

Для досягнення поставленої мети було визначено та виконано такі завдання дослідження:

1. Проведено аналіз однорангових мереж та їх принципів функціонування.
2. Визначено переваги та обмеження P2P-архітектур у порівнянні з мережею клієнт-сервер.
3. Виконано порівняння програмних рішень для мережевої взаємодії.
4. Окреслено критерії вибору архітектури для розробки програмного забезпечення.

Вхідними даними для аналізу стали принципи роботи однорангових мереж, переваги використання P2P-архітектури та можливості їх застосування для побудови ефективних систем комунікації.

РОЗДІЛ 2. ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ.

2.1. Клієнт-серверна архітектура.

Додаток, призначений для багатокористувацького використання в мережі, може реалізовувати два основні підходи до передачі повідомлень між користувачами: централізоване управління або розподілену синхронізацію.

У першому випадку, що реалізується за принципом «клієнт-сервер», використовується спеціальний сервер, який відповідає за обробку клієнтських запитів, збереження даних про стан користувачів та синхронізацію між ними. Сервер приймає інформацію від кожного клієнта та передає її іншим учасникам, забезпечуючи рівномірний розподіл навантаження і контроль над обміном даними.

Другий підхід передбачає взаємодію в мережі без централізованого сервера, де кожен учасник самостійно обмінюється даними з іншими, використовуючи методи, подібні до широкомовлення. Це дозволяє скоротити витрати ресурсів, необхідних для підтримки сервера, але ускладнює реалізацію клієнтської частини через необхідність обробки більш складних мережевих процесів. Комбінація функцій клієнта і сервера може бути виправданою у випадках підключення за принципом «точка-точка», де обидві сторони взаємодіють напряму, динамічно змінюючи ролі для більшої гнучкості у процесі обміну даними.

Розроблювана система повинна забезпечувати зручний обмін повідомленнями між користувачами через Інтернет.

Застосування клієнт-серверної архітектури для реалізації цієї системи пропонує низку ключових переваг.

- По-перше, знижується навантаження на клієнтські пристрої, оскільки основні обчислювальні процеси та обробка даних виконуються на сервері.

- По-друге, усувається потреба у встановленні прямих з'єднань між клієнтами, що підвищує безпеку та спрощує мережеву взаємодію.
- По-третє, централізоване управління дозволяє ефективно контролювати роботу системи, забезпечувати її стабільність, а також полегшує адміністрування та технічну підтримку [10].

2.2. Взаємодія клієнта і сервера

Обмін даними між клієнтом і сервером здійснюється через виклики серверних методів та передавання інформаційних повідомлень, які організовуються у черги. Для зручності спілкування користувачі об'єднуються у тематичні чат-кімнати, а сервер створює для кожної з них окрему чергу повідомлень. Це дозволяє підтримувати узгодженість даних між усіма учасниками сесії, синхронізуючи інформацію в межах конкретної кімнати та забезпечуючи ефективний обмін даними.

Така структура дає можливість обробляти повідомлення незалежно для кожної кімнати, оскільки вони функціонують як окремі модулі системи. Це сприяє високій продуктивності сервера, мінімізує затримки під час передачі даних та покращує масштабованість системи. Завдяки використанню оптимізованих черг повідомлення надходять користувачам у реальному часі, що дозволяє досягти швидкого обміну інформацією та комфортної взаємодії між учасниками [8].

Організація обміну повідомленнями таким способом сприяє покращенню масштабованості додатка, оскільки дає змогу без зниження продуктивності додавати нові кімнати та обслуговувати велику кількість користувачів одночасно. Це забезпечує стабільність і ефективність комунікаційної платформи, де кожен учасник може оперативно отримувати актуальну інформацію без затримок.

Рисунок 2.1 – Етапи взаємодії клієнта та серверу

2.3 Програмні засоби для розробки серверної частини

Оскільки основний фокус проєкту спрямований на веб-програмування, мова C++ не підходить через високий рівень складності для створення веб-додатків. Python, хоча і відомий своєю простотою та швидкістю розробки, може виявитися менш гнучким для потреб цього проєкту. Тому вибір зводиться до Java та C#. Для реалізації системи необхідна мова програмування, що підтримує об'єктно-орієнтовану парадигму, серед яких найпоширенішими на ринку є Java, C++, C# та Python.

C# був обраний для реалізації цього проєкту завдяки кільком важливим перевагам:

1. Розширені можливості для створення складних додатків

Мова C# підтримує ключові принципи об'єктно-орієнтованого програмування, зокрема інтерфейси, абстрактні класи, поліморфізм та наслідування. Це робить її оптимальним вибором для розробки масштабованих і складних веб-додатків.

2. Інтеграція з хмарними платформами

Технологія .NET Core, на якій базується C#, легко інтегрується з провідними хмарними сервісами, зокрема Microsoft Azure. Це забезпечує високу гнучкість та масштабованість додатка, спрощуючи управління ресурсами та розгортання у хмарному середовищі.

3. Високий рівень безпеки

C# забезпечує надійний захист даних завдяки суворій типізації та вбудованим механізмам управління пам'яттю, включаючи автоматичний збирач сміття. Це значно знижує ймовірність витоків пам'яті та помилок, пов'язаних із її ручним керуванням.

4. Широка підтримка спільноти та доступність ресурсів

Мова C# має велику активну спільноту розробників, а її розвиток підтримується Microsoft. Це гарантує доступ до великої кількості технічної документації, навчальних матеріалів, форумів та готових рішень для швидкого усунення помилок під час розробки.

5. Зручне середовище розробки Visual Studio

Розробка додатків на C# відбувається у Visual Studio — одному з найпотужніших середовищ розробки, яке пропонує розширені інструменти для дебагінгу, автоматичного тестування, рефакторингу та розгортання програм.

C# використовує проміжну мову CIL (Common Intermediate Language), яка є результатом компіляції всіх класів перед їх виконанням. Обробка цього коду здійснюється за допомогою віртуальної машини CLR (Common Language Runtime), що входить до складу платформ .NET Framework та .NET Core. Такий підхід дозволяє C#-додаткам працювати на різних операційних системах, використовуючи єдину кодову базу, що сприяє кросплатформеності та спрощенню розробки.

Основними перевагами мови програмування C# є:

1. Висока безпека

C# забезпечує надійний захист даних завдяки суворій типізації та автоматизованому управлінню пам'яттю за допомогою вбудованого збирача сміття (garbage collector), що знижує ризик помилок, пов’язаних із використанням пам'яті.

2. Ефективність виконання

Підтримка JIT-компіляції (Just-In-Time) дозволяє додаткам на C# виконуватися з високою продуктивністю та оптимізованою швидкістю на різних операційних системах.

3. Об'єктно-орієнтована парадигма

Мова C# повністю відповідає принципам об'єктно-орієнтованого програмування, підтримуючи ключові концепції, такі як інтерфейси, абстракції, наслідування та поліморфізм, що сприяє створенню структурованих та масштабованих додатків.

4. Захист від помилок

Завдяки статичній типізації та вбудованим механізмам обробки винятків, C# дозволяє знизити кількість потенційних помилок у коді ще на етапі компіляції.

5. Підтримка асинхронності та багатозадачності

C# має вбудовані інструменти для роботи з багатопотоковістю, включаючи підтримку асинхронного програмування за допомогою ключових слів async та await, що дозволяє створювати ефективні додатки з обробкою декількох завдань одночасно.

6. Кросплатформеність

Завдяки використанню проміжної мови CIL (Common Intermediate Language) та віртуальної машини CLR (Common Language Runtime), додатки на C# можуть працювати на різних апаратних платформах без необхідності зміни коду.

7. Широкий вибір інструментів та ресурсів

C# надає розробникам доступ до потужних інструментів, таких як Visual Studio, а також має обширну документацію та навчальні ресурси для спрощення процесу розробки.

8. Продуктивність розробки

Завдяки розвиненій екосистемі, яка включає інструменти для тестування, налагодження та автоматизованого розгортання, процес створення додатків на C# є швидким та зручним.

9. Активна спільнота

C# підтримується великою спільнотою розробників, яка забезпечує постійну технічну підтримку, обмін знаннями та регулярне оновлення мови.

Основною метою мови C# є забезпечення високого рівня безпеки додатків, запобігаючи несанкціонованому доступу до системних ресурсів та обмежуючи використання глобальних методів. Це сприяє зниженню ризиків неконтрольованого виконання коду та підвищує загальну захищеність програмного забезпечення. Крім того, C# має широкий вибір безкоштовних навчальних матеріалів і технічних ресурсів, доступних онлайн, що полегшує як процес освоєння мови, так і вирішення складних завдань під час розробки.

Платформа .NET, розроблена компанією Microsoft, є потужним середовищем для створення програмного забезпечення, яке охоплює широкий спектр інструментів та бібліотек. Вона дозволяє створювати веб-додатки, десктопні програми, мобільні застосунки, хмарні сервіси, ігри та інші типи програмних продуктів. Основними компонентами платформи є .NET Framework та .NET Core, які у новітніх версіях, починаючи з .NET 5, були об'єднані в єдину кросплатформенну екосистему, що спрощує розробку додатків для різних операційних систем.

.NET Framework — це початкова версія платформи .NET, створена для розробки та виконання програм на Windows. Вона пропонує розробникам великий набір бібліотек, інструментів та служб, які полегшують створення додатків. У свою чергу, .NET Core — це сучасна кросплатформенна версія .NET, що дозволяє розробляти програми для Windows, macOS та Linux. Вона відзначається високою продуктивністю та підтримкою сучасних технологій, що робить її популярною серед розробників, які створюють кросплатформенні рішення.

Починаючи з .NET 5, Microsoft об'єднала можливості .NET Framework та .NET Core, створивши єдину кросплатформенну платформу. Це усунуло необхідність вибору між двома середовищами, спростивши процес розробки. Нові версії, починаючи з .NET 5, забезпечують відмінну продуктивність,

гнучкість у масштабуванні та підтримку кількох операційних систем, що робить платформу універсальним рішенням для створення сучасних додатків.

ASP.NET — це фреймворк для створення динамічних веб-додатків і веб-сервісів, який забезпечує гнучкість та простоту у розробці веб-рішень. Його кросплатформенна версія, ASP.NET Core, відзначається високою продуктивністю та масштабованістю, що робить її оптимальним вибором для створення сучасних веб-додатків з підтримкою різних операційних систем.

Xamarin, один із компонентів платформи .NET, дозволяє створювати мобільні додатки для iOS та Android, використовуючи мову програмування C#. Це дає можливість розробникам писати основний код один раз та застосовувати його на різних мобільних платформах, що значно скорочує витрати на розробку та подальшу підтримку програм [9].

Хмарна платформа Azure, яка тісно інтегрована з .NET, надає розробникам широкий спектр можливостей для створення, розгортання та управління додатками у хмарному середовищі. Серед доступних функцій — розробка масштабованих хмарних додатків, управління базами даних, аналітика та засоби для автоматизації робочих процесів. Завдяки тісній інтеграції з платформою .NET, розробники можуть легко використовувати сервіси Azure для побудови потужних, продуктивних та надійних хмарних рішень.

.NET — це платформа, яка завдяки своїй гнучкості, масштабованості та багатому набору інструментів широко використовується для розробки програмного забезпечення в різних галузях. Вона забезпечує розробників усіма необхідними засобами для створення безпечних, продуктивних та сучасних додатків, що відповідають сучасним стандартам якості.

Однією з ключових переваг .NET є підтримка об'єктно-орієнтованого програмування, що дозволяє створювати масштабовані та модульні додатки зі зручним управлінням кодовою базою. Вбудовані механізми безпеки, включаючи типобезпеку та автоматизоване управління пам'яттю,

допомагають захищати програми від виконання шкідливого коду та помилок, пов'язаних із некоректною роботою з пам'яттю.

.NET також відзначається кросплатформеністю, що дозволяє створювати програми для Windows, macOS та Linux з однієї кодової бази. Інструменти розробки, такі як Visual Studio та Visual Studio Code, забезпечують розширені можливості для написання, тестування, налагодження та розгортання коду, що робить процес розробки більш зручним, швидким та ефективним.

Windows Communication Foundation (WCF) — це платформа для створення сервісів у середовищі .NET Framework, яка дозволяє розробникам будувати безпечні, масштабовані та надійні додатки для обміну даними між клієнтами та серверами. WCF залишається ефективним інструментом, особливо для корпоративних систем та внутрішніх додатків, де критично важливі висока безпека та стабільність [11].

Основними компонентами WCF є:

1. Service (Сервіс) — головний елемент виконання логіки додатка, що приймає запити від клієнтів і відправляє відповіді.
2. Endpoint (Кінцева точка доступу) — точка взаємодії з сервісом, яка складається з адреси та контракту.
3. Contract (Контракт) — описує операції, які може виконувати сервіс, і включає кілька типів:

- Service Contract — визначає основні методи, доступні для клієнтів.
- Data Contract — описує структуру даних, які передаються між клієнтом і сервером.
- Message Contract — забезпечує контроль над SOAP-повідомленнями.
- Fault Contract — відповідає за обробку помилок у сервісі.

4. Binding (Прив'язка) — визначає способи взаємодії між клієнтом і сервісом, включаючи протоколи, безпеку та інші параметри комунікації.

5. Behavior (Поведінка) — налаштовує додаткові параметри сервісу, такі як тайм-аути, управління потоками, інстанціями та інші властивості.

Переваги WCF:

- Уніфікація — підтримує створення сервісів для різних комунікаційних протоколів у межах однієї платформи.
- Масштабованість та продуктивність — здатність обробляти велику кількість запитів із високою швидкістю.
- Інтеграція з іншими технологіями Microsoft — легке поєднання з платформами .NET та хмарними сервісами Azure.

Недоліки WCF:

- Складність використання — потребує глибоких знань для правильної конфігурації та налаштування.
- Припинення розвитку — Microsoft більше не оновлює WCF, фокусуючись на більш сучасних технологіях, таких як ASP.NET Core.

Попри це, WCF залишається актуальним рішенням для розробки корпоративних сервісів, особливо у сценаріях, де важливі безпека, контроль над передачею даних та надійність роботи у внутрішніх мережах.

JSON (JavaScript Object Notation) — це текстовий формат для представлення та обміну структурованими даними, який вирізняється своєю простотою та легкістю для читання як людиною, так і машиною. Він базується на синтаксисі об'єктів JavaScript, проте залишається незалежним від конкретної мови програмування, що робить його універсальним засобом обміну даними між різними системами.

Основні характеристики JSON:

1. Простота та читабельність

JSON використовує пари "ключ-значення" та впорядковані списки, що забезпечує простий синтаксис, зрозумілий як для розробників, так і для машинного парсингу.

2. Компактність та ефективність

Завдяки мінімалістичному синтаксису JSON займає менше місця в порівнянні з іншими форматами даних, що зменшує обсяг передаваних даних через мережу та підвищує ефективність обміну інформацією.

3. Універсальність використання

JSON може бути використаний у більшості сучасних мов програмування, таких як JavaScript, Python, Java, C# тощо, завдяки наявності вбудованих або сторонніх бібліотек для парсингу та генерації JSON-даних.

4. Гнучка структура даних

Формат підтримує вкладені об'єкти та масиви, що дозволяє представляти складні структури даних, включаючи багаторівневі об'єкти та колекції значень.

Завдяки цим характеристикам JSON став стандартом для обміну даними у веб-розробці, API, хмарних сервісах та інших системах, де важлива швидкість передачі та простота інтеграції даних [12].

2.3. Загальна структура програми

Щоб детально описати загальну структуру програми, варто звернути увагу на платформу Microsoft .NET, яка відома своєю гнучкістю та широкими можливостями для створення розподілених програм. Основою цієї платформи є середовище виконання CLR (Common Language Runtime), яке забезпечує універсальність виконання коду, написаного на різних мовах програмування. Під час компіляції вихідний код перетворюється на платформонезалежний байт-код MSIL (Microsoft Intermediate Language), який є проміжною мовою. Остаточна компіляція у машинний код, оптимізований для конкретної операційної системи та апаратної архітектури, виконується безпосередньо під час запуску програми за допомогою технології JIT-компіляції (Just-In-Time). Такий підхід дозволяє створювати кросплатформені рішення, зберігаючи стабільність, продуктивність та високу сумісність програмного забезпечення з різними середовищами виконання.

Варто відзначити, що бібліотека класів .NET Framework пропонує широкий набір функцій для створення додатків різної складності — від базових до масштабованих корпоративних систем. Архітектура платформи включає потужні інструменти для мережевої взаємодії, що сприяє розробці високопродуктивних серверних рішень.

У середовищі .NET серверна частина може бути реалізована як віддалений об'єкт, здатний обслуговувати кілька клієнтів одночасно, забезпечуючи синхронізацію даних та стабільну роботу додатка навіть при високих навантаженнях.

Технологія .NET Remoting дозволяє організовувати віддалену взаємодію між об'єктами, розташованими як у різних доменах додатків, так і на окремих пристроях. Віддалені об'єкти можуть передаватися клієнту за значенням (повна копія об'єкта) або за посиланням, де клієнт працює через об'єкт-проксі, який прозоро виконує взаємодію з серверною частиною, приховуючи технічні деталі підключення [12].

Web-сервіси (XML Web Services) у .NET можна розглядати як віддалені об'єкти, які надають доступ до своїх методів через HTTP для передачі даних та XML як формат обміну. Web-сервіс є класом, розміщеним на веб-сервері, а клієнт взаємодіє з ним за допомогою URL через проксі-об'єкт, який містить методи оригінального класу та переспрямовує виклики до сервісу.

Для динамічного створення проксі-об'єктів у .NET використовуються метадані, а для опису можливостей сервісу застосовується стандарт WSDL (Web Services Description Language). Це дозволяє будь-якому клієнту, що підтримує відправку SOAP-запитів, взаємодіяти із сервісом. Альтернативно, можна статично згенерувати проксі-клас, додавши його до клієнтського додатка.

Web-сервіси у .NET часто розміщуються на веб-сервері IIS (Internet Information Services), де клієнт отримує WSDL-опис для коректного формування SOAP-запитів, а IIS обробляє ці запити через ISAPI-розширення, перенаправляючи їх до відповідного екземпляра сервісу.

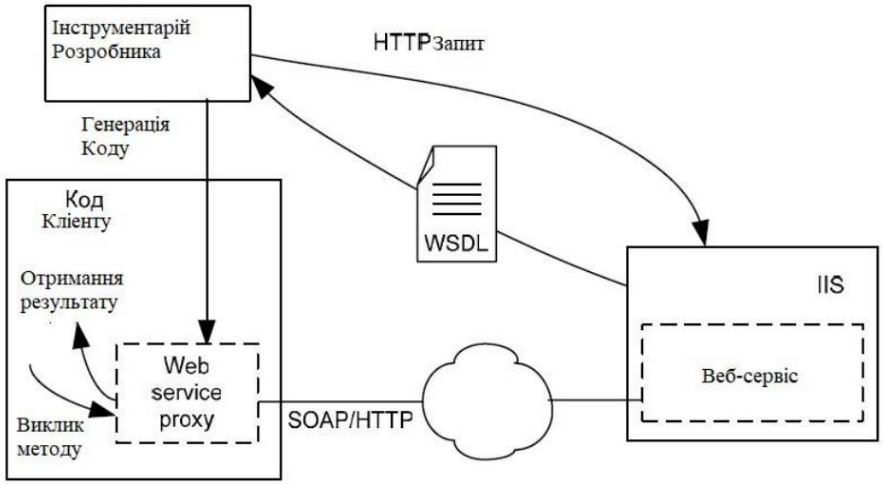


Рисунок 2.2 – Статична генерація проксі-класу web-сервісу

Використання web-служб для реалізації серверної частини забезпечує низку переваг, зокрема усуває залежність від специфіки клієнтських додатків та операційної системи користувача. Для взаємодії з сервісом клієнту достатньо підтримки протоколу HTTP та можливості обробляти дані у форматі XML.

Ключова перевага web-служб полягає у спрощенні клієнт-серверної взаємодії, оскільки методи сервісу викликаються так само, як локальні методи класу. Це дозволяє розробникам зосередитися на створенні бізнес-логіки, не заглиблюючись у деталі низькорівневої мережевої комунікації.

Завдяки використанню стандартних протоколів, таких як HTTP, та форматів обміну даними на кшталт XML, web-служби легко масштабуються та сумісні з різними клієнтськими платформами, забезпечуючи прозорий і стабільний обмін даними незалежно від середовища виконання.

Клієнтську частину реалізовано як Windows-додаток на платформі .NET, що забезпечує доступ до потужних інструментів розробки, бібліотек та автоматизованих можливостей, таких як генерація проксі-класів для web-служб та управління пам'яттю через збирач сміття.

Використання .NET прискорює процес розробки завдяки широкому набору інструментів, зокрема середовищу Visual Studio, яке спрощує

налагодження, тестування, збірку та розгортання додатків. Це також сприяє підвищенню безпеки та надійності програмного забезпечення.

Крім того, завдяки технології Windows Presentation Foundation (WPF), клієнтський додаток отримує можливість створення сучасних, естетично привабливих та зручних інтерфейсів, що покращує загальний користувацький досвід і відповідає сучасним стандартам якості.

2.4. Структура серверної частини

Серверну частину системи реалізовано у вигляді XML web-служби, розгорнутої на Internet Information Services (IIS) як частину веб-додатка. Основні .функції сервісу включають управління чат-кімнатами, підключенням клієнтів, синхронізацію їхнього стану, а також надання додаткової інформації для інтерактивної взаємодії.

Web-служба представлена класом із набором методів, деякі з яких доступні для віддаленого виклику. Вона може бути інтегрована як файл з вихідним кодом або .NET-збірка. Веб-додаток, у складі якого функціонує сервіс, є інтерактивним набором файлів, що включає логіку обробки клієнтських запитів, конфігураційні налаштування та механізми обробки подій.

Усі компоненти веб-додатка розміщені у віртуальному каталозі IIS, що забезпечує централізоване управління, легкий доступ до ресурсу та ефективну обробку клієнтських запитів.

ASP.NET — це сучасна технологія Microsoft для створення веб-додатків, яка розширює функціональність IIS (Internet Information Services), дозволяючи серверу обробляти запити до ресурсів додатка. Клієнти взаємодіють із веб-додатком через URL, який сервер трансформує у фізичний шлях до файлу на диску. ASP.NET функціонує як ISAPI-розширення, забезпечуючи потужний механізм для обробки HTTP-запитів і формування відповідей.

Життєвий цикл додатка ASP.NET починається з моменту, коли клієнт надсилає запит до веб-сервера. IIS перевіряє розширення файлу, визначає, який обробник має його обробити, і передає запит відповідному модулю. Наприклад, ASP.NET обробляє файли з розширеннями `.aspx`, `.asmx`, `.ashx` і `.ascx`. Можна додати обробку й інших типів файлів, зареєструвавши відповідний обробник в IIS. Такий підхід забезпечує контрольований процес обробки запитів та управління станом веб-додатка.

При надходженні першого запиту до ресурсу веб-додатка ASP.NET, клас ApplicationManager створює домен додатка, який ізолює додаток на рівні глобальних змінних, забезпечуючи безпеку даних і контроль над ресурсами. У межах цього домену ініціалізується об'єкт HostingEnvironment, який зберігає ключову інформацію про додаток, зокрема каталог, у якому він розміщений.

Після цього ASP.NET створює об'єкти HTTPContext, HTTPRequest та HTTPResponse для кожного запиту. HTTPRequest містить дані про запит, включаючи заголовки та cookies, HTTPResponse відповідає за формування відповіді клієнту, а HTTPContext об'єднує їх, забезпечуючи повний контекст для обробки HTTP-комунікації.

При запуску програми створюється об'єкт класу HTTPApplication, який обробляє події життєвого циклу додатка. Якщо у проєкті присутній файл Global.asax, у ньому може бути описаний похідний клас від HTTPApplication з кастомною логікою обробки подій, який і буде використаний для управління життєвим циклом замість базового класу.

Для обробки кожного нового запиту в ASP.NET спочатку створюється новий екземпляр класу HTTPApplication, однак, щоб підвищити продуктивність, система може повторно використовувати ці екземпляри. Це дозволяє оптимізувати роботу веб-додатка, забезпечуючи швидке та ефективне обслуговування клієнтських запитів, що показано на рисунку 2.3.

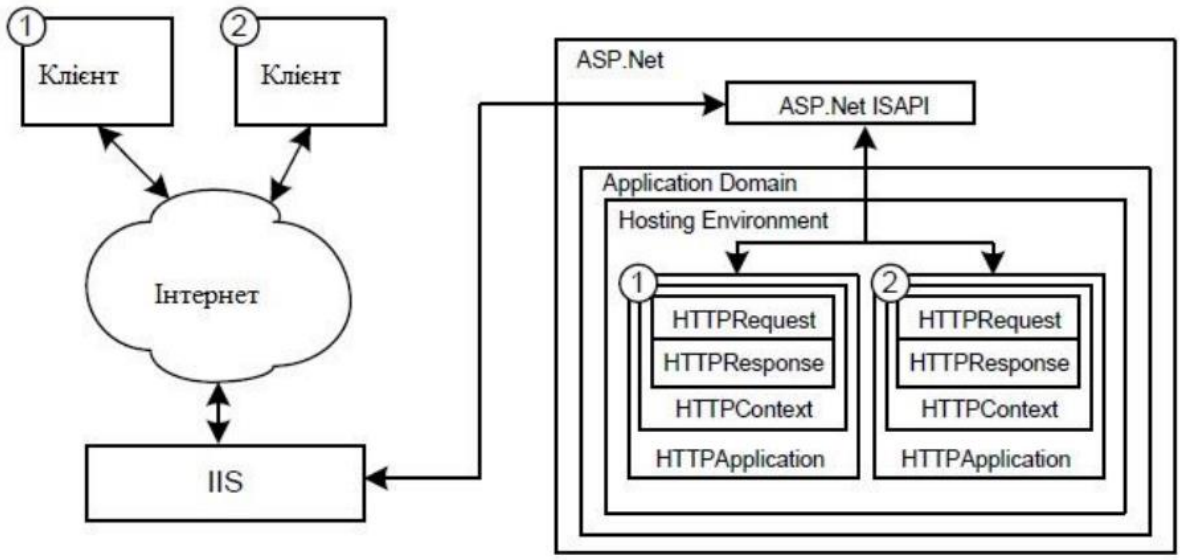


Рисунок 2.3 – Схема відносини примірника HTTPApplication і запиту клієнта

На етапі обробки запиту додаток ASP.NET породжує події, які можна розширити, реалізувавши обробники в похідному класі від HTTPApplication. Кожен екземпляр цього класу обробляє лише один запит одночасно, що дозволяє зберігати параметри запиту у нестатичних членах без необхідності блокування потоків.

Процес обробки можна доповнити за допомогою HTTP модулів, зокрема SessionStateModule, який відповідає за керування сесіями користувачів і генерує події, такі як Session_Start та Session_End. Це дає змогу додатку реагувати на зміну стану сесії в реальному часі.

При зверненні до web-служби додаток створює екземпляр класу web-служби, похідного від System.Web.Services.WebService, і викликає відповідний метод. Об'єкти Application та Session, що представляють колекції пар "ключ-значення", використовуються для збереження даних сесії користувача та глобального стану додатка, який розділяється між усіма активними сесіями.

Дані сесії в ASP.NET можуть зберігатися у пам'яті процесу, на State Server або у базі даних. Зберігання у пам'яті процесу забезпечує найшвидший

доступ, проте підходить лише для невеликих обсягів даних, щоб уникнути перевантаження оперативної пам'яті сервера.

Серверна частина має управляти кімнатами, обробляти запити користувачів на підключення та синхронізувати повідомлення між учасниками. Для цього зберігається список кімнат та користувачів, а кожна кімната має власну чергу повідомлень.

Web-служба використовує об'єкт Session, який через HttpSessionState надає доступ до інформації про користувача, поточну кімнату та час останньої синхронізації. Це дозволяє ідентифікувати клієнта та отримувати доступ до відповідної черги повідомлень для коректної обробки даних.

Основні особливості синхронізації та роботи системи:

1. Синхронізація повідомлень:

Серверна частина підтримує чергу повідомлень, впорядкованих за часом надходження. Під час кожного запиту клієнта сервер порівнює час останньої синхронізації та передає всі нові повідомлення, додаючи до черги нові повідомлення, отримані від клієнта.

2. Обробка запитів та управління станом:

Клас HTTPApplication обробляє лише один запит одночасно, що дозволяє зберігати параметри запиту у нестатичних членах без блокувань. Для збереження даних між викликами використовується HttpSessionState, що дозволяє зберігати стан сесії та інформацію про користувача.

3. Гнучкість зберігання даних:

Дані сесії можуть зберігатися у пам'яті процесу, на State Server або у базі даних, що забезпечує баланс між швидкістю доступу та ефективним використанням ресурсів.

4. Масштабованість і продуктивність:

Завдяки використанню ASP.NET у поєднанні з IIS, система підтримує обробку великої кількості одночасних запитів, що підвищує її масштабованість та стабільність при високих навантаженнях [13].

5. Інтеграція та безпека:

Платформа підтримує інтеграцію з іншими системами через стандартні протоколи та формати, такі як HTTP та XML, а також забезпечує захист даних через вбудовані механізми аутентифікації та авторизації.

2.5. Структура клієнтської частини

Клієнтська частина системи реалізована як WPF-додаток на платформі .NET, що забезпечує зручний та інтуїтивний інтерфейс для користувачів. Вона складається з кількох модулів, що відповідають за взаємодію з користувачем, введення текстових повідомлень, обмін файлами та виклик методів web-служби для обміну даними з сервером.

Основні особливості клієнтської частини:

1. Модульна структура:

Клієнтський додаток побудований із кількох модулів, кожен з яких виконує окрему функцію, забезпечуючи масштабованість, легкість у підтримці та розширенні функціоналу.

2. Інтерактивний інтерфейс:

Додаток надає зручний графічний інтерфейс, що підтримує надсилання текстових повідомлень, обмін файлами та швидку взаємодію з сервером.

3. Інтеграція з сервером:

Клієнтська частина напряму викликає методи web-служби для обміну даними, забезпечуючи стабільний та швидкий зв’язок із серверною частиною.

4. Продуктивність та кросплатформеність:

Додаток оптимізований для Windows, але його модульна архітектура дозволяє адаптувати рішення для інших платформ у майбутньому [15].

2.6 .Висновок до другого розділу

У другому розділі було проведено ґрунтовний аналіз підходів до реалізації клієнт-серверної архітектури для багатокористувацького

програмного забезпечення. Основну увагу було зосереджено на централізованій моделі передачі даних, яка передбачає використання сервера для обробки запитів клієнтів, управління доступом до ресурсів та забезпечення синхронізації між усіма учасниками мережі. Такий підхід дозволяє ефективно контролювати процеси обміну даними, підвищує рівень безпеки інформації та спрощує адміністрування системи.

Було обґрунтовано вибір мови програмування C# та платформи .NET для реалізації проєкту. Основними причинами вибору стали:

- Кросплатформеність: Можливість запуску додатків на різних операційних системах завдяки використанню проміжної мови CIL (Common Intermediate Language) та середовища виконання CLR (Common Language Runtime).
- Об'єктно-орієнтоване програмування (ООП): Підтримка основних принципів ООП, таких як інкапсуляція, наслідування, поліморфізм, що сприяє створенню структурованого та масштабованого коду.
- Інструменти для розробки: Потужне середовище Visual Studio, що спрощує процес створення, тестування та налагодження програм.
- Безпека: Вбудовані механізми управління пам’яттю, зокрема автоматичний збирач сміття, а також статична типізація коду, що знижує ймовірність помилок.

Також було розглянуто принципи взаємодії між клієнтською та серверною частинами програми, де кожен клієнт взаємодіє із сервером через web-служби, зокрема за допомогою технології ASP.NET. Обмін даними організовано за допомогою черг повідомлень, що забезпечує ефективний розподіл інформації між користувачами в реальному часі.

Загальна структура програмного забезпечення включає:

1. Серверна частина – відповідає за обробку запитів клієнтів, управління чат-кімнатами та синхронізацію повідомлень між учасниками.

2. Клієнтська частина – WPF-додаток, який забезпечує зручний графічний інтерфейс для користувачів, включаючи можливість надсилання текстових повідомлень, обміну файлами та підключення до сервера.

3. Програмні засоби – використання .NET Core для серверної частини та WPF для клієнтської частини, що забезпечує стабільність, продуктивність та можливість кросплатформенного використання.

РОЗДІЛ 3. РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКУ ДЛЯ ЛОКАЛЬНОЇ ПЕРЕДАЧІ ІНФОРМАЦІЇ

3.1 Актуальність розробки додатку для локальної передачі інформації

Розробка клієнт-серверних додатків для обміну інформацією на локальному рівні набуває значної актуальності, особливо серед приватних компаній, що прагнуть забезпечити високий рівень захисту конфіденційних даних. Використання локальних серверів дозволяє зберігати інформацію в межах корпоративної мережі, що мінімізує ризики витоку даних та корпоративного шпигунства.

Інноваційність підходу полягає у створенні власного програмного рішення з використанням клієнт-серверної архітектури, яке функціонує виключно в локальному середовищі компанії. Це забезпечує повний контроль над процесами обміну даними та усуває залежність від сторонніх програмних продуктів, які потенційно можуть збирати дані про діяльність компанії, включаючи кількість персоналу та продуктивність.

Окрему цінність представляє підхід до організації мережевої взаємодії без використання централізованого сервера. У цьому сценарії кожен учасник мережі самостійно передає дані всім іншим, застосовуючи методи широкомовлення. Такий підхід дозволяє значно зекономити на ресурсах, зазвичай витрачених на підтримку сервера. Проте, він ускладнює клієнтську частину додатка через необхідність обробки більш складної логіки передачі даних. Водночас, поєднання функцій клієнта і сервера може бути виправданим у сценаріях підключення «точка-точка», де обидві сторони здатні динамічно змінювати ролі, забезпечуючи гнучкість та ефективність у взаємодії.

Запропонований підхід демонструє практичне значення та інноваційність у сфері інформаційної безпеки, оскільки забезпечує локалізацію даних, мінімізацію ризиків зовнішнього впливу на критичні бізнес-процеси та ресурсоефективність архітектури мережевої взаємодії.

3.2. Структура проекту

Загальна структура проекту передбачає взаємодію клієнтської частини з трьома серверами, кожен з яких виконує окрему функціональну роль, забезпечуючи комплексне управління інформаційними потоками та підвищену безпеку даних (рис. 3.1).

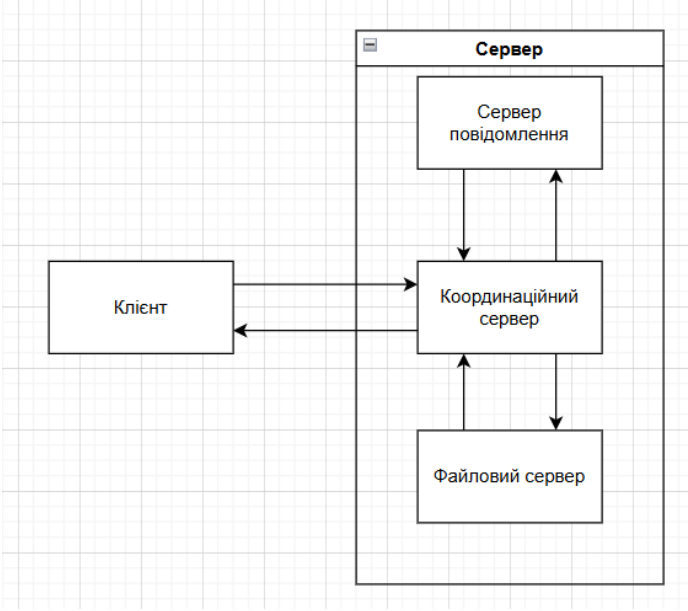


Рисунок 3.1 – Структура проекту

Організаційний сервер виконує ключову роль контролера та координатора системи. Він відповідає за автентифікацію та авторизацію користувачів, керує запитами клієнтів і визначає порядок доступу до інших серверів. Його основне завдання – забезпечення централізованого контролю над комунікаціями між клієнтами та сервісами системи. Організаційний сервер діє як перший рівень захисту, перевіряючи права доступу перед передачею запитів до інших серверів.

Файловий сервер відповідає за зберігання, управління та надання доступу до файлів. Він обробляє запити на завантаження, збереження, редагування та видалення файлів. Доступ до файлових ресурсів можливий

лише після підтвердження прав через організаційний сервер, що забезпечує контроль доступу до конфіденційних даних.

Сервер повідомлень використовується для обміну текстовими та системними повідомленнями між користувачами. Підтримує також пересилання конфіденційними повідомленнями [13].

Взаємодія між клієнтом та серверами відбувається за чітко визначеними протоколами, де організаційний сервер виступає центральним вузлом управління запитами, а файловий та сервер повідомлень взаємодіють безпосередньо лише після успішної авторизації. Така архітектура сприяє підвищенню рівня безпеки, розподілу навантаження між серверами та спрощенню масштабування системи.

3.3. Розробка архітектури

Для реалізації функціональних можливостей клієнт-серверного додатку використовується наступні класи (рис. 3.2):

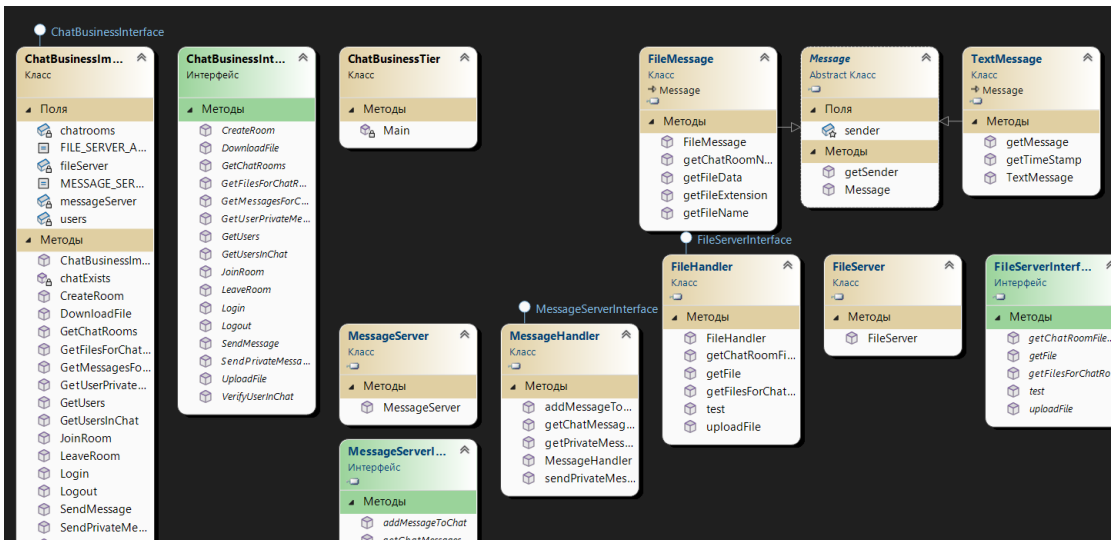


Рисунок 3.2 – Діаграма класів

Данні класи поділяються на декілька груп. Для контролюючого сервера використовуються класи ChatBusinessImplementation, ChatBusinessInterface,

ChatBusinessTier Для файлового серверу використовуються класи FileHandler, FileServer, FileServerInterface. Для функціоналу пересилання повідомлення використовуються MessageHandler, MessageServer, MessageServerInterface

Розглянемо функціонал пов’язаний з роботою повідомлень (рис. 3.3):

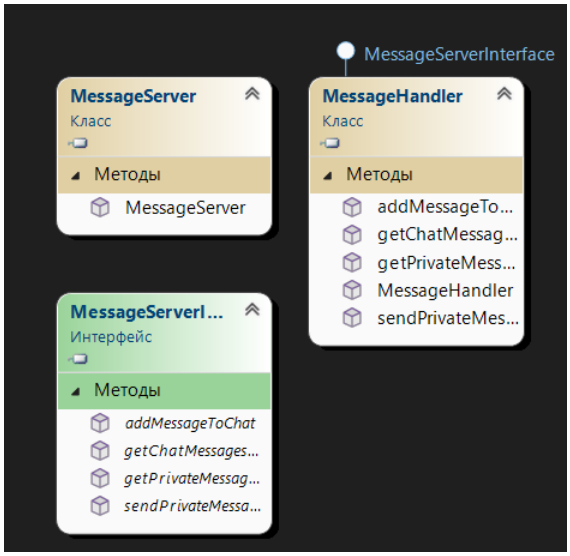


Рисунок 3.3 – Класи групи Message

Клас MessageServer у просторі імен MessageServer є основною точкою запуску сервера обміну повідомленнями, реалізованого за допомогою технології Windows Communication Foundation (WCF).

Цей сервер налаштований для прослуховування вхідних TCP-з'єднань, забезпечуючи комунікацію між клієнтами та сервером у локальній мережі.

Метод Main (рис. 3.4) є точкою входу в додаток, і при його запуску виводиться повідомлення "Starting up Message server...", що інформує про початок роботи сервера. Далі створюється об'єкт ServiceHost під назвою host, який відповідає за розгортання та управління екземплярами служби WCF. Для конфігурації передачі даних використовується NetTcpBinding — це біндінг, що забезпечує двосторонню комунікацію через протокол TCP, який підходить для внутрішньо-мережевих додатків з високою продуктивністю.

Об'єкт host ініціалізується для класу MessageHandler, який, містить логіку обробки повідомлень. Далі до хоста додається кінцева точка служби за

допомогою методу AddServiceEndpoint, де вказано інтерфейс MessageServerInterface, біндінг NetTcpBinding та URL-адресу net.tcp://0.0.0.0:8101/MessageServer, яка дозволяє підключатися до сервера через порт 8101 з використанням усіх доступних мережевих інтерфейсів (рис. 3.4).

```
 * **/
namespace MessageServer
{
    public class MessageServer
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Starting up Message server...");
            ServiceHost host;
            NetTcpBinding binding = new NetTcpBinding();
            host = new ServiceHost(typeof(MessageHandler));

            host.AddServiceEndpoint(typeof(MessageServerInterface), binding, "net.tcp://0.0.0.0:8101/MessageServer");

            host.Open();
            Console.WriteLine("Message Service online.");
            Console.ReadLine();
            host.Close();
        }
    }
}
```

Рисунок 3.4 – Клас MessageServer

Після цього сервер відкриває з'єднання через метод host.Open() і виводить повідомлення "Message Service online.", сигналізуючи про успішний запуск. Програма чекає на введення користувача (Console.ReadLine()), що утримує сервер у робочому стані. Після отримання введення сервер закриває з'єднання методом host.Close() і завершує роботу. Така архітектура забезпечує базовий сервер обміну повідомленнями з простою конфігурацією через WCF, орієнтовану на TCP-з'єднання.

Інтерфейс MessageServerInterface (рис. 3.5) є контрактом служби Windows Communication Foundation (WCF), який визначає набір методів для взаємодії з сервером обміну повідомленнями. Цей інтерфейс помічений атрибутом [ServiceContract], що вказує, що даний інтерфейс використовується для опису контракту служби в WCF. Контракт служби визначає, які методи доступні для клієнтів та яким чином вони можуть взаємодіяти із сервером.

Інтерфейс містить декілька методів, кожен з яких позначений атрибутом [OperationContract].

```
namespace MessageServer
{
    [ServiceContract]
    public interface MessageServerInterface
    {
        [OperationContract]
        void addMessageToChat(string chatroom,TextMessage textMessage);
        [OperationContract]
        List<TextMessage> getChatMessagesFor(string chatroom);
        [OperationContract]
        void sendPrivateMessage(string user,TextMessage textMessage);
        [OperationContract]
        List<TextMessage> getPrivateMessageFor(string receiver);

    }
}
```

Рисунок 3.5 – Інтерфейс MessageServerInterface

Void addMessageToChat(string chatroom, TextMessage textMessage). Цей метод призначений для додавання нового текстового повідомлення. List<TextMessage> getChatMessagesFor(string chatroom) - метод повертає список повідомлень, збережених у вказаній чат-кімнаті. Void sendPrivateMessage(string user, TextMessage textMessage) - метод дозволяє відправити приватне повідомлення.

Клас MessageHandler (рис. 3.5) є реалізацією інтерфейсу MessageServerInterface та відповідає за обробку як публічних, так і приватних повідомлень у системі. Основне його призначення — управління зберіганням та обробкою повідомлень у чат-кімнатах та приватних повідомленнях користувачів у локальному середовищі.

Клас містить два основних поля: chatMessages та privateMessages, які є словниками (Dictionary<string, List<TextMessage>>). Поле chatMessages зберігає всі повідомлення для кожної окремої чат-кімнати, де ключем є назва кімнати, а значенням — список об'єктів TextMessage. Поле privateMessages працює аналогічно, але зберігає приватні повідомлення для кожного користувача, де ключем є ім'я користувача, а значенням — список його

повідомлень. Обидва поля позначені атрибутом [DataMember], що дозволяє серіалізувати їх для передачі через WCF-служби, забезпечуючи збереження даних між сесіями.

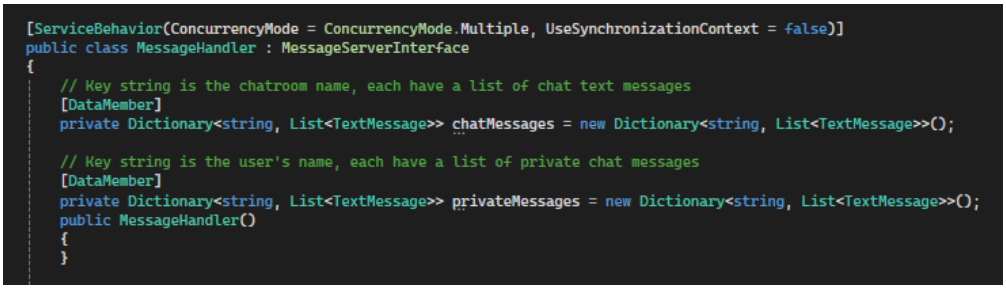


Рисунок 3.5 – Клас MessageHandler

Метод addMessageToChat (public void addMessageToChat(string chatroom, TextMessage textMessage)) відповідає за додавання нового повідомлення до певної чат-кімнати (рис. 3.6). Спочатку метод виводить у консоль інформацію про надходження нового повідомлення. Потім, використовуючи метод TryGetValue, перевіряється наявність кімнати у словнику chatMessages. Якщо чат-кімната вже існує, повідомлення додається до її списку повідомлень. Якщо ні, створюється новий запис у словнику, і повідомлення додається до нового списку. Таким чином, метод гарантує, що нові чат-кімнати створюються автоматично під час першого повідомлення.

Рисунок 3.6 – Метод addMessageToChat

Метод getChatMessagesFor (public List<TextMessage> getChatMessagesFor(string chatroom)) використовується для отримання списку

повідомлень із певної чат-кімнати (рис. 3.7). Якщо кімната існує у словнику chatMessages, метод повертає список повідомлень. Якщо кімната відсутня, метод повертає null і виводить у консоль повідомлення про помилку. Цей метод дозволяє клієнтам отримувати історію чату або оновлювати список повідомлень під час сесії.

```
public List<TextMessage> getChatMessagesFor(string chatroom)
{
    if (chatMessages.ContainsKey(chatroom))
    {
        Console.WriteLine("Requested to get messages for chatroom: " + chatroom);
        return chatMessages[chatroom];
    }
    else
    {
        Console.Write("Failed to find chat message for chatroom: " + chatroom);
    }
    return null;
}
```

Рисунок 3.7 – Метод getChatMessagesFor

Метод sendPrivateMessage (public void sendPrivateMessage(string receiver, TextMessage textMessage)) використовується для надсилання приватного повідомлення конкретному користувачу (рис. 3.8). Він приймає ім'я отримувача (receiver) та об'єкт повідомлення (textMessage). Якщо в словнику privateMessages вже існує запис для цього користувача, повідомлення додається до списку. Якщо користувач ще не має жодного повідомлення, створюється новий запис зі списком повідомлень, що включає поточне. Метод також виводить інформацію про відправника, отримувача та вміст повідомлення в консоль для моніторингу роботи системи.

```
public void sendPrivateMessage(string receiver, TextMessage textMessage)
{
    Console.WriteLine("Incoming private message request from sender: " + textMessage.getSender()+ "\nMessage: " +
    if (privateMessages.ContainsKey(receiver))
    {
        privateMessages[receiver].Add(textMessage);
    }
    else
    {
        privateMessages[receiver] = new List<TextMessage>();
        privateMessages[receiver].Add(textMessage);
    }
}
```

Рисунок 3.8 – Метод sendPrivateMessage

Метод getPrivateMessageFor (public List<TextMessage> getPrivateMessageFor(string receiver)) використовується для отримання приватних повідомлень конкретного користувача. Якщо користувач має повідомлення у словнику privateMessages, метод повертає список цих повідомлень. Якщо повідомлень немає, метод повертає пустий список (new List<TextMessage>()), гарантуючи, що клієнт завжди отримує результат, навіть якщо для цього користувача немає повідомлень.

```
public List<TextMessage> getPrivateMessageFor(string receiver)
{
    List<TextMessage> pm = null;
    if(privateMessages.ContainsKey(receiver))
    {
        pm = privateMessages[receiver];
    }
    pm = pm == null ? new List<TextMessage>() : pm;
    return pm;
}
```

Рисунок 3.9 – Метод getPrivateMessageFor

Клас MessageHandler забезпечує централізоване управління обміном повідомленнями, включаючи як групові чати, так і приватні повідомлення. Логіка класу підтримує автоматичне створення чат-кімнат, зберігання історії повідомлень та просту взаємодію з клієнтами через WCF.

Для взаємодії з файлами використовується наступна група класів (рис. 3.10):

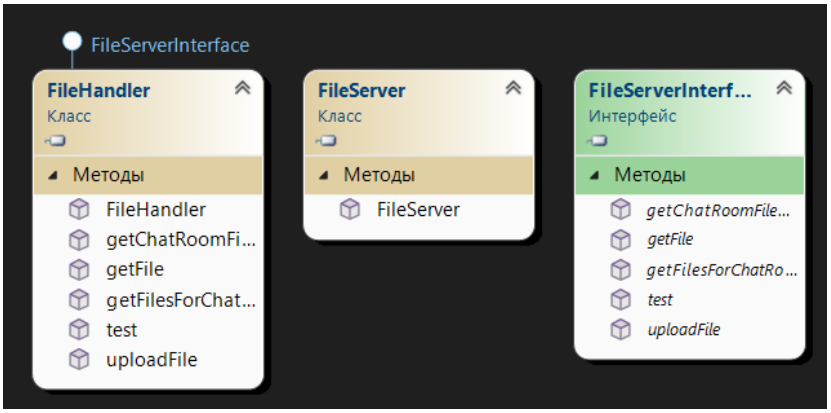


Рисунок 3.10 – Група класів Files

Клас FileServer (рис. 3.11) є точкою запуску для файлового сервера для обробки та обміну файлами через мережевий протокол TCP. Головний метод Main ініціалізує сервер. Для мережевої комунікації створюється аналогічно об'єкт NetTcpBinding, який забезпечує ефективну передачу даних у локальній мережі. Метод AddServiceEndpoint додає кінцеву точку служби за адресою net.tcp://0.0.0.0:8100/FileServer, що дозволяє серверу приймати вхідні з'єднання на порту 8100 через усі доступні мережеві інтерфейси. Після цього сервер активується за допомогою host.Open(). Сервер залишається активним, очікуючи введення користувача (Console.ReadLine()), після чого з'єднання закривається командою host.Close().

```
namespace FileServer
{
    public class FileServer
    {

        static void Main(string[] args)
        {
            Console.WriteLine("Starting up File server...");
            ServiceHost host;
            NetTcpBinding binding = new NetTcpBinding();
            host = new ServiceHost(typeof(FileHandler));

            host.AddServiceEndpoint(typeof(FileServerInterface), binding, "net.tcp://0.0.0.0:8100/FileServer");

            host.Open();
            Console.WriteLine("File Service online.");
            Console.ReadLine();
            host.Close();
        }
    }
}
```

Рисунок 3.11 – Клас FileServer

Інтерфейс FileServerInterface (рис. 3.12) визначає контракт WCF-служби для обміну файлами між клієнтами та файловим сервером. Він містить п'ять методів: uploadFile(FileMessage fileMessage), getFilesForChatRoom(string chatRoom), getChatRoomFileCount(string chatroom), getFile(string chatroom, string fileName), а також test() для повернення тестового списку файлів. Кожен метод позначений атрибутом [OperationContract], що дозволяє викликати їх через WCF-з'єднання, забезпечуючи базові операції управління файлами у мережевому середовищі.

```
[ServiceContract]
public interface FileServerInterface
{
    [OperationContract]
    void uploadFile(FileMessage fileMessage);

    [OperationContract]
    List<FileMessage> getFilesForChatRoom(string chatRoom);


    [OperationContract]
    int getChatRoomFileCount(string chatroom);

    [OperationContract]
    FileMessage getFile(string chatroom, string fileName);

    [OperationContract]
    List<FileMessage> test();

}
```

Рисунок 3.12 – Інтерфейс FileServerInterface

Клас FileHandler є реалізацією інтерфейсу FileServerInterface і використовується для управління файлами у WCF-службі файлового сервера, забезпечуючи зберігання, пошук та отримання файлів, а також тестування функціональності. Він позначений наступним атрибутом серверу [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, UseSynchronizationContext = false)], що дозволяє обробляти кілька запитів одночасно без синхронізації контексту, підвищуючи продуктивність сервера при обробці паралельних запитів. Основним сховищем даних у цьому класі є поле files, яке представляє собою список об'єктів FileMessage (List<FileMessage>), де кожен об'єкт зберігає інформацію про файл, включаючи ім'я файлу, чат-кімнату, формат, відправника та власне бінарні дані файлу (рис. 3.13).

Конструктор FileHandler (рис. 3.13) ініціалізує порожній список files для зберігання файлів, готовий до наповнення під час роботи сервера. Метод uploadFile(FileMessage fileMessage) використовується для завантаження нового файлу на сервер, додаючи об'єкт FileMessage до списку files та виводячи інформацію про завантажений файл у консоль для моніторингу.

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, UseSynchronizationContext = false)]
public class FileHandler : FileServerInterface
{

    // File data is stored in a List of FileMessage instances.
    private List<FileMessage> files;
    public FileHandler()
    {
        files = new List<FileMessage>();
    }
```

Рисунок 3.13 – Клас FileHandler

Метод getFile(string chatroom, string fileName) шукає у колекції files файл за назвою чат-кімнати та іменем файлу, повертаючи перший знайдений об'єкт FileMessage або null, якщо файл відсутній (рис. 3.14).

Для отримання всіх файлів у вказаній чат-кімнаті використовується метод getFilesForChatRoom(string chatRoomName), який створює новий список, перебираючи всі файли у сховищі та додаючи до результату лише ті, що відповідають заданій кімнаті, при цьому в консоль виводиться інформація про запит (рис. 3.14).

```
public FileMessage getFile(string chatroom, string fileName)
{
    foreach (FileMessage fm in files)
    {
        if(fm.getChatRoomName().Equals(chatroom) && fm.getFileName().Equals(fileName))
        {
            return fm;
        }
    }
    return null;
}

public List<FileMessage> getFilesForChatRoom(string chatRoomName)
{
    Console.WriteLine("Request to get files for chatroom: " + chatRoomName);
    List<FileMessage> chatRoomFiles = new List<FileMessage>();
    foreach (FileMessage fm in files)
    {
        if (fm.getChatRoomName().Equals(chatRoomName))
        {
            chatRoomFiles.Add(fm);
        }
    }
    return chatRoomFiles;
}
```

Рисунок 3.14 - Методи getFile та getFilesForChatRoom

Метод getChatRoomFileCount(string chatroom) рахує кількість файлів у заданій чат-кімнаті, використовуючи цикл для підрахунку відповідних файлів у колекції files (рис. 3.15).

```
public int getChatRoomFileCount(string chatroom)
{
    int count = 0;
    foreach (FileMessage fm in files)
    {
        if (fm.getChatRoomName().Equals(chatroom))
        {
            count = count + 1;
        }
    }
    return count;
}
```

Рисунок 3.15 - Метод getChatRoomFileCount

Клас FileHandler демонструє просту, але ефективну обробку файлів у мережевому середовищі, з базовими механізмами для зберігання, фільтрації та пошуку файлів за вказаними критеріями.

Останні класи, що відповідають за серверну частину, в основному за контролюючий функціонал:

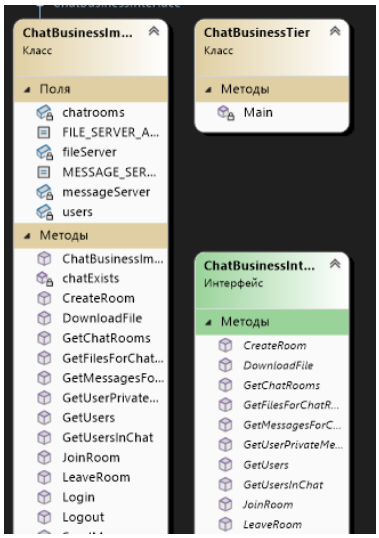


Рисунок 3.16 – Група класів, сервісу-контролера

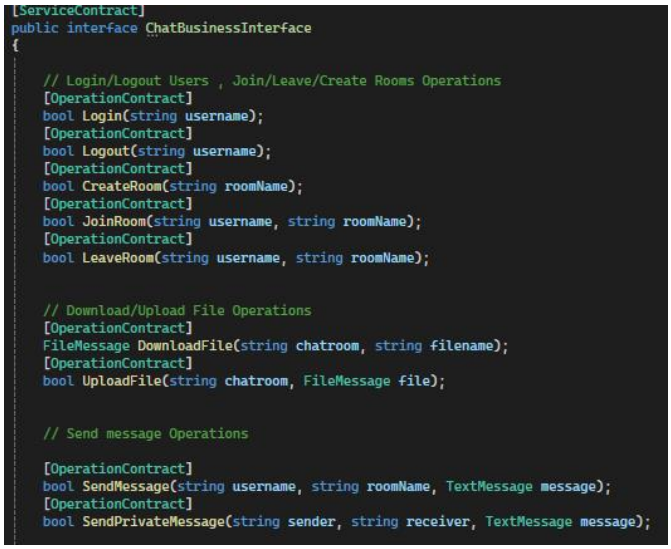
Клас ChatBusinessTier (рис. 3.17) є точкою запуску бізнес-логіки серверного рівня в архітектурі WCF і за своєю структурою аналогічний класам FileServer та MessageServer. Головний метод Main ініціалізує сервер бізнес-логіки Взаємодія з клієнтами налаштована через метод AddServiceEndpoint,

який додає кінцеву точку служби з URL-адресою net.tcp://0.0.0.0:8102/BusinessTier, що дозволяє серверу приймати підключення на порту 8102 через усі доступні мережеві інтерфейси. Архітектурно цей клас аналогічний файловому (FileServer) та серверу повідомлень (MessageServer), оскільки використовує однаковий підхід до розгортання WCF-служби з мінімальними налаштуваннями.

```
namespace ChatBusinessTier
{
    public class ChatBusinessTier
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Starting up Business Tier server...");
            ServiceHost host;
            NetTcpBinding binding = new NetTcpBinding();
            host = new ServiceHost(typeof(ChatBusinessImplementation));

            host.AddServiceEndpoint(typeof(ChatBusinessInterface), binding, "net.tcp://0.0.0.0:8102/BusinessTier");

            host.Open();
            Console.WriteLine("Business Tier Service online.");
            Console.ReadLine();
            host.Close();
        }
    }
}
```

Рисунок 3.17 – Клас ChatBusinessTier

Інтерфейс ChatBusinessInterface визначає контракт для бізнес-логіки WCF-сервісу, відповідального за управління користувачами, чат-кімнатами, обміном повідомленнями та файлами у системі. Він забезпечує базові операції для управління користувачами для автентифікації користувачів, управління чат-кімнатами. Функції обміну файлами

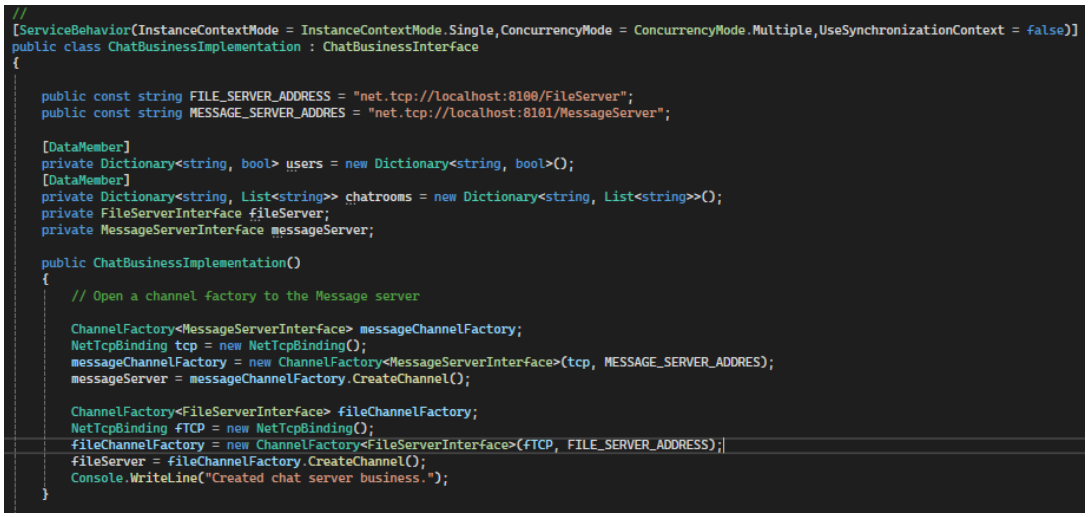


```
[ServiceContract]
public interface ChatBusinessInterface
{

    // Login/Logout Users , Join/Leave/Create Rooms Operations
    [OperationContract]
    bool Login(string username);
    [OperationContract]
    bool Logout(string username);
    [OperationContract]
    bool CreateRoom(string roomName);
    [OperationContract]
    bool JoinRoom(string username, string roomName);
    [OperationContract]
    bool LeaveRoom(string username, string roomName);


    // Download/Upload File Operations
    [OperationContract]
    FileMessage DownloadFile(string chatroom, string filename);
    [OperationContract]
    bool UploadFile(string chatroom, FileMessage file);


    // Send message Operations

    [OperationContract]
    bool SendMessage(string username, string roomName, TextMessage message);
    [OperationContract]
    bool SendPrivateMessage(string sender, string receiver, TextMessage message);
```

Рисунок 3.18 – Інтерфейс ChatBusinessInterface

Клас ChatBusinessImplementation (рис. 3.19) є серверним компонентом бізнес-логіки в архітектурі WCF, який реалізує інтерфейс ChatBusinessInterface для управління користувачами, чат-кімнатами, повідомленнями та файлами у системі. Він позначений атрибутом [ServiceBehavior] із параметрами InstanceContextMode = Single, що забезпечує використання одного екземпляра сервісу для обробки всіх клієнтських запитів, та ConcurrencyMode = Multiple, який дозволяє одночасне обслуговування декількох клієнтських запитів без блокування потоку.

Конструктор ChatBusinessImplementation (рис. 3.19) відповідає за ініціалізацію сервісу, зокрема, він створює канали зв'язку з файловим (FileServer) та сервером повідомлень (MessageServer). Для цього використовуються об'єкти ChannelFactory, які створюють проксі-клієнти для взаємодії з віддаленими сервісами через TCP-з'єднання (NetTcpBinding). Константи FILE_SERVER_ADDRESS та MESSAGE_SERVER_ADDRES визначають адреси відповідних серверів.



```
//
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,ConcurrencyMode = ConcurrencyMode.Multiple,UseSynchronizationContext = false)]
public class ChatBusinessImplementation : ChatBusinessInterface
{

    public const string FILE_SERVER_ADDRESS = "net.tcp://localhost:8100/FileServer";
    public const string MESSAGE_SERVER_ADDRES = "net.tcp://localhost:8101/MessageServer";

    [DataMember]
    private Dictionary<string, bool> users = new Dictionary<string, bool>();
    [DataMember]
    private Dictionary<string, List<string>> chatrooms = new Dictionary<string, List<string>>();
    private FileServerInterface fileServer;
    private MessageServerInterface messageServer;

    public ChatBusinessImplementation()
    {
        // Open a channel factory to the Message server

        ChannelFactory<MessageServerInterface> messageChannelFactory;
        NetTcpBinding tcp = new NetTcpBinding();
        messageChannelFactory = new ChannelFactory<MessageServerInterface>(tcp, MESSAGE_SERVER_ADDRES);
        messageServer = messageChannelFactory.CreateChannel();

        ChannelFactory<FileServerInterface> fileChannelFactory;
        NetTcpBinding fTCP = new NetTcpBinding();
        fileChannelFactory = new ChannelFactory<FileServerInterface>(fTCP, FILE_SERVER_ADDRESS);
        fileServer = fileChannelFactory.CreateChannel();
        Console.WriteLine("Created chat server business.");
    }
```

Рисунок 3.19 – Клас ChatBusinessImplementation

Основні дані в класі зберігаються у вигляді двох словників: users (Dictionary<string, bool>), який містить імена користувачів та їхній статус онлайн (true або false), та chatrooms (Dictionary<string, List<string>>), що асоціює назви чат-кімнат зі списками користувачів, які до них приєдналися. Методи userExists, chatExists, userLoggedIn та userInChat (рис. 3.20) — це приватні методи для перевірки, чи існує користувач або кімната, чи залогінений користувач, а також чи знаходиться він у певній кімнаті. Всі ключові методи управління користувачами та чатами синхронізовані за допомогою атрибута [MethodImpl(MethodImplOptions.Synchronized)], що гарантує послідовність виконання у багатопотоковому середовищі.

```
// Check if user exists
private bool userExists(string username)
{
    return users.ContainsKey(username);
}
// Check if chat room exists
private bool chatExists(string chatroom)
{
    return chatrooms.ContainsKey(chatroom);
}
// Check if user is already logged in
private bool userLoggedIn(string username)
{
    // users[username] is true if logged in so check if one of the clients are logged in
    return userExists(username) && users[username] == true;
}
// Checks if user belongs to a chatroom specified
private bool userInChat(string username, string roomName)
{
    bool userInChat = false;
    if (chatExists(roomName) && chatrooms[roomName].Contains(username))
    {
        userInChat = true;
    }
    return userInChat;
}
```

Рисунок 3.20 – Методи userExists, chatExists, userLoggedIn, userInChat

Метод Login(string username) додає нового користувача або оновлює його статус, дозволяючи повторний вхід (рис. 3.21). Якщо користувач вже зареєстрований і активний, виводиться відповідне повідомлення. Метод Logout(string username) змінює статус користувача на false, якщо він був залогінений. Для управління чат-кімнатами передбачені методи CreateRoom(string roomName) (створює нову кімнату, якщо такої ще немає), JoinRoom(string username, string roomName) (додає користувача до існуючої кімнати) та LeaveRoom(string username, string roomName) (видаляє користувача з кімнати).

Методи для обміну файлами включають DownloadFile(string chatroom, string filename) — отримує файл з файлового сервера через проксі fileServer.getFile, та UploadFile(string chatroom, FileMessage fileMessage) — завантажує файл на сервер, якщо кімната існує.

```
[MethodImpl(MethodImplOptions.Synchronized)]
public bool Login(string username)
{
    bool loggedIn;
    // Check if user exists and is not logged in
    if (users.TryGetValue(username, out loggedIn))
    {
        Console.WriteLine("Existing user " + username + " logged in.");

        if(loggedIn)
        {
            Console.WriteLine("User " + username + " already logged in");
            loggedIn = false;
        }
        else
        {
            Console.WriteLine("User " + username + " added");
            users[username] = true;
            loggedIn = true;
        }
    }
    else
    {
        Console.WriteLine("New user " + username + " added.");
        loggedIn = true;
        users.Add(username, true);
    }
    Console.WriteLine("Current users: " + users.Count);
    return loggedIn;
}
```

Рисунок 3.21 – Метод Login

Функціональність обміну повідомленнями представлена методами SendMessage (рис. 3.22) для надсилання групових повідомлень та SendPrivateMessage(string sender, string receiver, TextMessage message) для приватних повідомлень, обидва з яких передають дані на сервер повідомлень через messageServer. Методи GetMessagesForChatRoom та GetUserPrivateMessages повертають історію повідомлень, збережену на сервері повідомлень.

```
[MethodImpl(MethodImplOptions.Synchronized)]
public bool SendMessage(string username, string roomName, TextMessage message)
{
    bool messageSent = false;
    if(chatExists(roomName) && userInChat(username, roomName))
    {
        if (message == null && messageServer == null)
        {
            Console.WriteLine("Message null");
        }
        else
        {
            messageServer.addMessageToChat(roomName, message);
        }
        messageSent = true;
    }
    else
    {
        Console.WriteLine("userinChat() or chatExists() returned false. User may not have access
    }

    if(messageSent == false)
    {
        Console.WriteLine("failed to send a message.");
```

Рисунок 3.22 – Метод SendMessage

Додатково, методи GetUsers, GetUsersInChat, GetChatRooms та VerifyUserInChat забезпечують можливість отримання списків користувачів та

перевірки доступу (рис. 3.23). Клас ChatBusinessImplementation забезпечує повноцінну бізнес-логіку для управління чатами, включаючи автентифікацію, авторизацію, роботу з файлами та повідомленнями, використовуючи проксі-клієнти для взаємодії із зовнішніми сервісами в архітектурі WCF.

```
[MethodImpl(MethodImplOptions.Synchronized)]
public List<string> GetUsers()
{
    return users.Keys.ToList();
}

[MethodImpl(MethodImplOptions.Synchronized)]
public List<string> GetUsersInChat(string roomname)
{
    return chatrooms[roomname];
}

[MethodImpl(MethodImplOptions.Synchronized)]
public Dictionary<string, List<string>> GetChatRooms()
{
    return chatrooms;
}

[MethodImpl(MethodImplOptions.Synchronized)]
public bool VerifyUserInChat(string user, string roomName)
{
    bool verified = false;

    if(userExists(user) && userInChat(user, roomName))
    {
        verified = true;
    }

    return verified;
}
```

Рисунок 3.23 – Методи для отримання списків

3.4. Проектування користувацького інтерфейсу

Для проектування візуального інтерфейсу у цьому проекті використовується інструмент Visual Studio Blend (рис. 3.24), який є потужним середовищем для створення сучасних графічних інтерфейсів на платформі WPF (Windows Presentation Foundation). Visual Studio Blend дозволяє створювати інтерактивні елементи управління, застосовувати стилі, анімації та шаблони для покращення користувацького досвіду. Завдяки підтримці XAML, інструмент надає можливість працювати з візуальними компонентами у режимі реального часу, а також редагувати як візуальну частину, так і код одночасно. Це робить його ефективним вибором для розробників та

дизайнерів, які прагнуть створити естетично привабливі та функціональні інтерфейси у WCF-додатках.

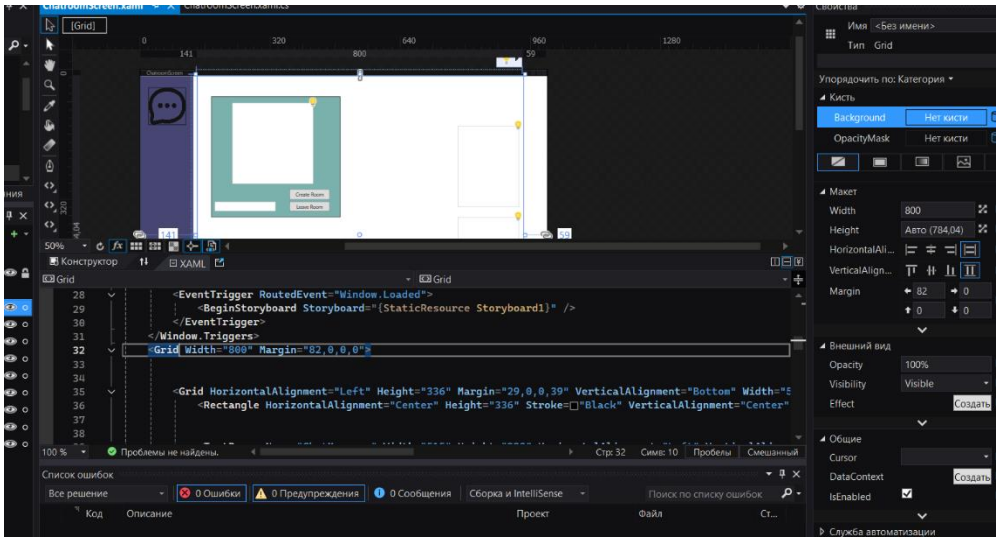


Рисунок 3.24 – Інтерфейс Visual Studio Blend

Окремо варто розглянути можливості створення власних анімацій за допомогою Visual Studio Blend, оскільки цей інструмент надає широкий функціонал для створення складних візуальних ефектів у WPF-додатках. За допомогою Blend можна створювати анімації, використовуючи редактор часової шкали (Timeline), що дозволяє покроково контролювати зміни властивостей елементів, таких як прозорість, розмір, обертання та положення.

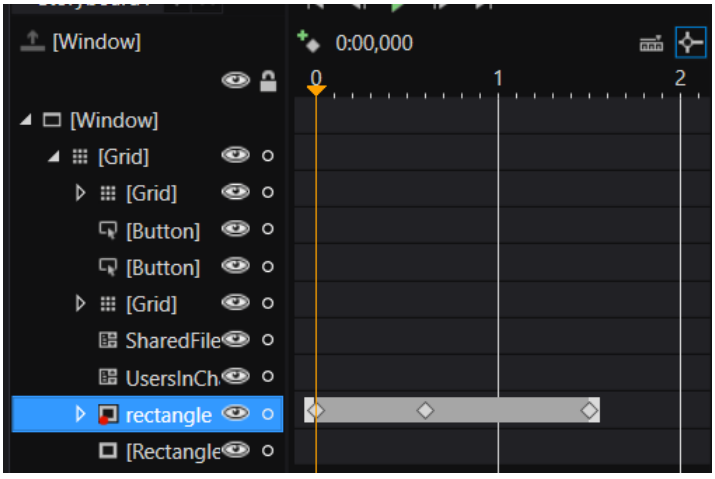


Рисунок 3.25 - Створювати анімації (Timeline)

Крім того, Blend підтримує створення тригерів (Triggers) та візуальних станів (Visual States), що дозволяє автоматизувати анімації на основі взаємодії користувача з інтерфейсом (рис. 3.26). Це відкриває широкі можливості для створення динамічних UI, які покращують користувацький досвід і роблять додаток більш інтерактивним та сучасним.

```
<Window.Resources>
    <Storyboard x:Key="Storyboard1">
        <DoubleAnimationUsingKeyFrames Storyboard.TargetName="rectangle" Storyboard.TargetProperty
            <EasingDoubleKeyFrame KeyTime="00:00:00" Value="7.581"/>
            <EasingDoubleKeyFrame KeyTime="00:00:00.6000000" Value="7.603"/>
            <EasingDoubleKeyFrame KeyTime="00:00:01.5000000" Value="1.079"/>
        </DoubleAnimationUsingKeyFrames>
        <DoubleAnimationUsingKeyFrames Storyboard.TargetName="rectangle" Storyboard.TargetProperty
            <EasingDoubleKeyFrame KeyTime="00:00:00" Value="436.2"/>
            <EasingDoubleKeyFrame KeyTime="00:00:00.6000000" Value="437.4"/>
            <EasingDoubleKeyFrame KeyTime="00:00:01.5000000" Value="4.4"/>
```

Рисунок 3.26 – Використання тригерів

3.5 Тестування додатку

Важливо провести комплексне тестування всього існуючого функціоналу додатку, починаючи з його завантаження та закінчуючи відправкою повідомлень і файлів між користувачами. Для цього спочатку необхідно активувати 3 мікросервіси (рис. 3.27), кожен з яких відповідає за окремий аспект функціональності системи. Першим етапом є запуск файлового сервера (FileServer), який обробляє завантаження та отримання файлів, забезпечуючи їхнє збереження та доступність для користувачів у межах чат-кімнат. Наступним кроком є активація сервера повідомлень (MessageServer), який відповідає за збереження історії повідомлень, передачу текстових даних у чатах та обробку приватних повідомлень між користувачами. Третім етапом є запуск бізнес-логіки (ChatBusinessTier), яка координує роботу файлового та сервера повідомлень, забезпечує автентифікацію користувачів, управління чат-кімнатами та перевірку прав доступу.

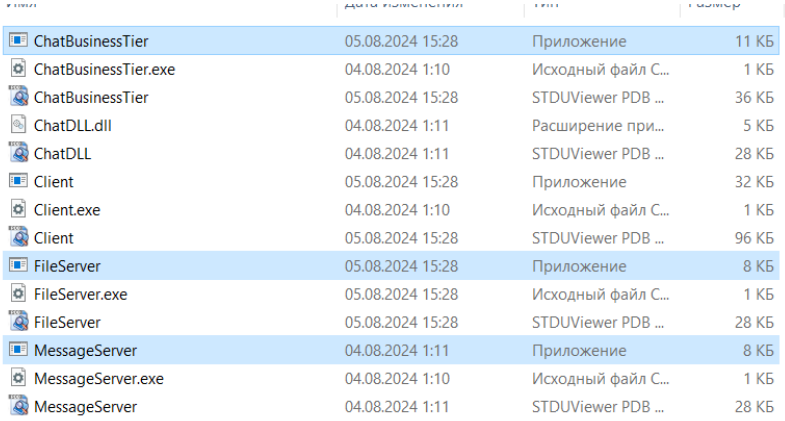


Рисунок 3.27 – Виділені мікросервіси для завантаження

У процесі запуску додатка всі три мікросервіси повинні вивести повідомлення, що свідчать про успішну ініціалізацію та готовність до обробки запитів. Це важливо для перевірки правильності їхнього розгортання та коректності мережевих з'єднань між сервісами.

Першим запускається файловий сервер (FileServer), який після успішної активації має вивести повідомлення: "File Service online.". Це підтверджує, що сервер успішно відкрив мережевий порт для прийому файлів.

Другим мікросервісом є сервер повідомлень (MessageServer), який після запуску повинен відобразити: "Message Service online.".

Третім компонентом є сервер бізнес-логіки (ChatBusinessTier), який після успішної ініціалізації, у консолі має з'явитися повідомлення: "Business Tier Service online.".

Наявність цих повідомлень у кожному мікросервісі вказує на успішний запуск системи, правильне відкриття мережевих портів та готовність програми до обробки клієнтських запитів. Якщо хоча б одне з цих повідомлень відсутнє, це може свідчити про помилки у конфігурації, мережевих підключеннях або ініціалізації компонентів. (рис. 3.28):

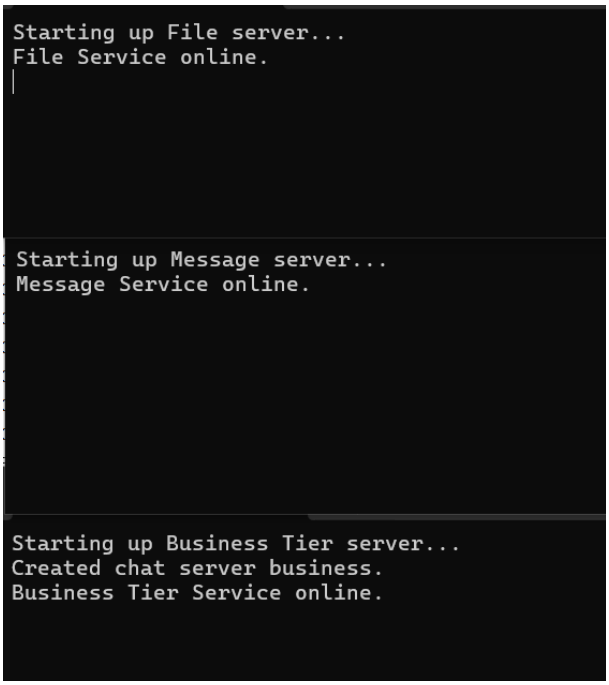


Рисунок 3.28 – Запуск мікросервісів

Далі можна завантажити додаток із користувацьким інтерфейсом за допомогою виконуваного .exe файлу Client, який є графічною оболонкою для взаємодії з усіма мікросервісами системи. Цей файл представляє собою самостійний клієнтський застосунок (рис. 3.29).

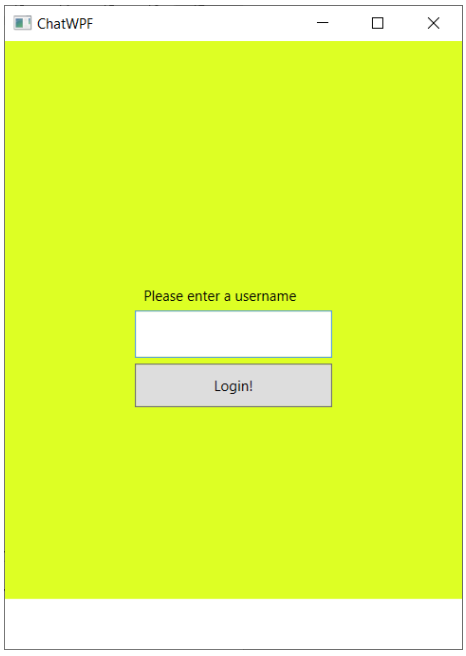


Рисунок 3.29 – Вікно авторизації

У головному вікні додатка користувачу буде запропоновано авторизуватися через введення імені користувача, після чого він зможе створювати нові чат-кімнати, приєднуватися до існуючих або надсилати приватні повідомлення іншим користувачам (рис. 3.30).

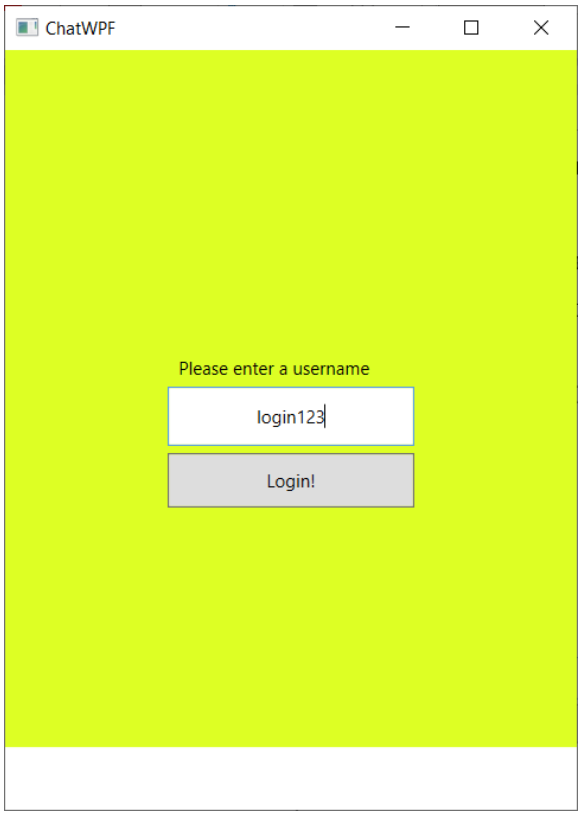


Рисунок 3.30 – Введення логіну користувача

Після введення імені користувача в поле для автентифікації необхідно натиснути на кнопку «Login», щоб ініціювати процес входу в систему. Натискання цієї кнопки надсилає запит до сервера бізнес-логіки, де відбувається перевірка наявності користувача в системі та його статусу. Якщо користувача ще немає у базі або він був раніше відключений, сервер реєструє нового користувача або змінює його статус на «активний». (рис. 3.31):

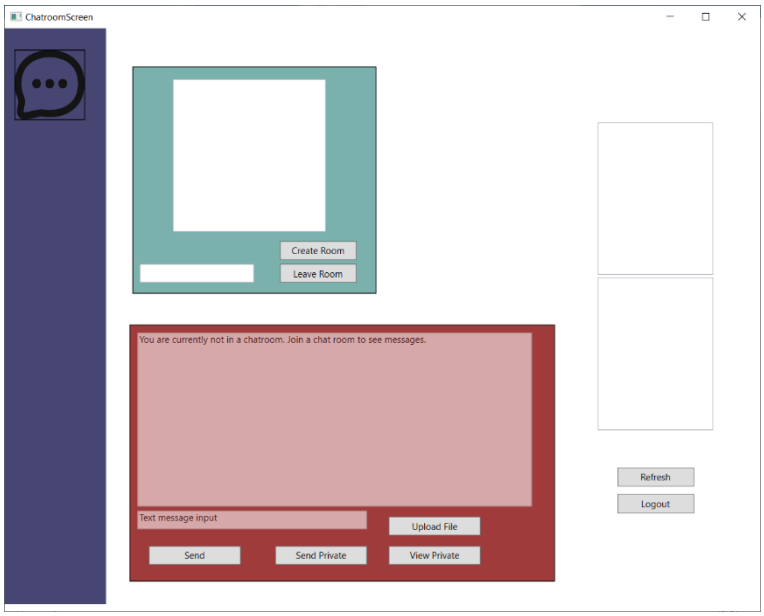


Рисунок 3.31 – Вхід в систему

Для тестування відкриємо, ще одну клієнтську програму користувача, та введемо логін однаковий з попереднім. Повідомлення відображає, що користувач вже зареєстрований на іншому пристрої, система виводить повідомлення про помилку і вимагає повторного введення даних для входу (рис. 3.32):

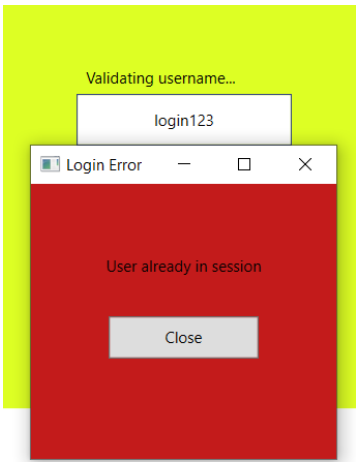


Рисунок 3.32 – Виведення помилки авторизації

Для успішної авторизації, у разі помилки або некоректного введення даних, необхідно змінити логін і повторно натиснути на кнопку «Login». Після

цього слід створити активну сесію для взаємодії користувачів. Для цього потрібно ввести бажану назву сесії у відповідне текстове поле та натиснути кнопку «CreateRoom». У результаті сервер створює нову чат-кімнату та додає її до списку доступних сесій (рис. 3.33):

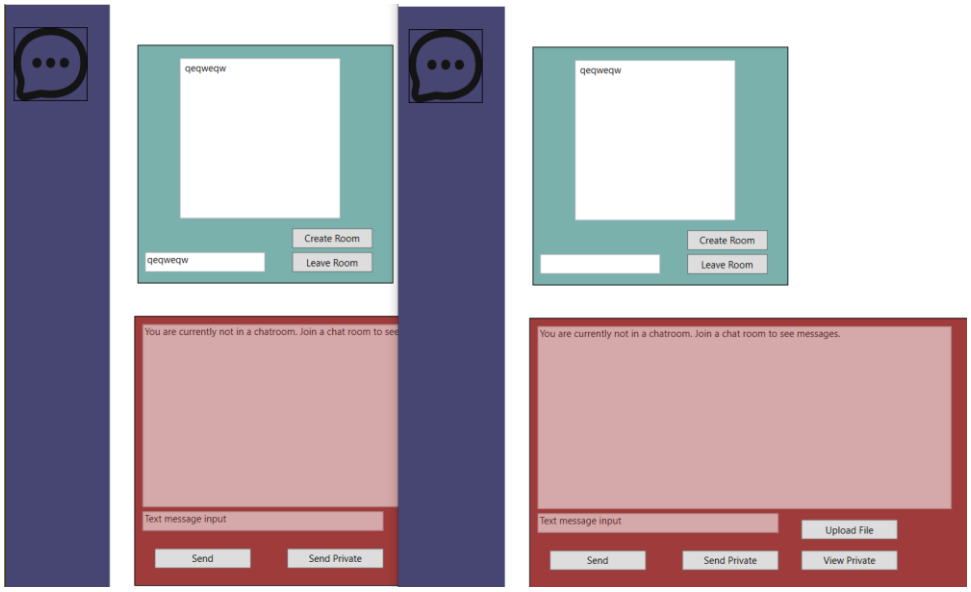


Рисунок 3.33 – Створення активної сесії

Після попередніх дій є можливість підключити користувачів до створеної сесії. Для цього необхідно двічі клацнути по назві сесії у списку доступних чат-кімнат. У відповідь відкриється нове вікно, в якому потрібно натиснути кнопку «Join», що ініціює запит до сервера бізнес-логіки для додавання користувача до обраної сесії. (рис. 3.34):

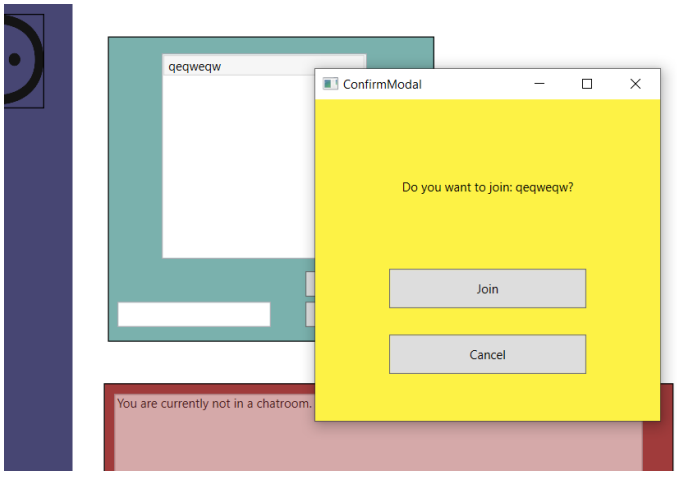


Рисунок 3.34 – Виведення повідомлення про приєднання

Після успішного приєднання обох користувачів до створеної сесії, у правій частині інтерфейсу додатка відображається кількість активних користувачів, підключених до цієї сесії. Цей показник оновлюється в реальному часі, відображаючи актуальну кількість учасників, які приєдналися до чат-кімнати. (рис. 3.35):

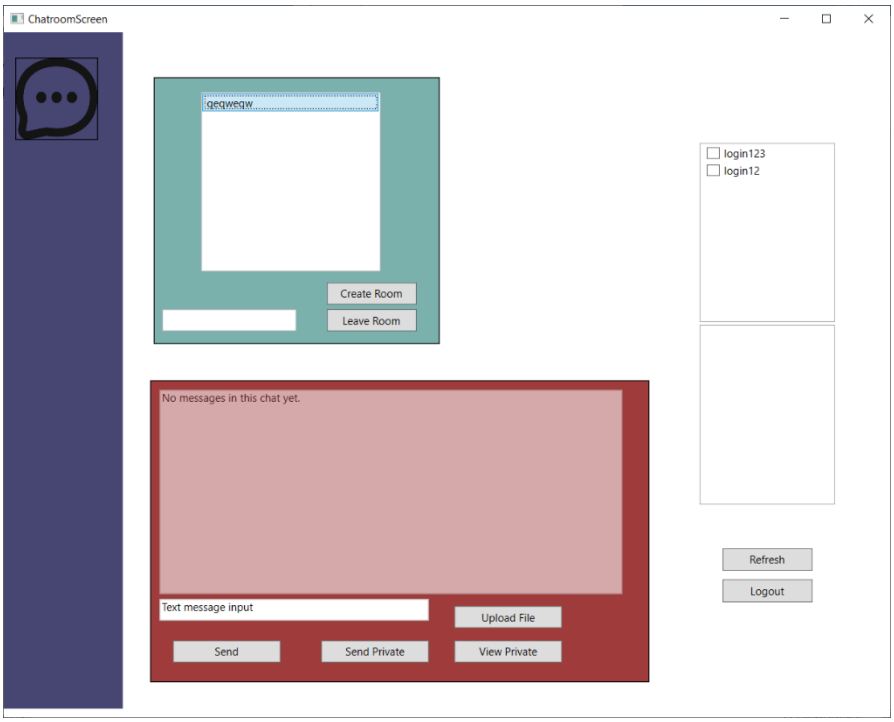


Рисунок 3.35 – Виведення кількості всіх користувачів сесії

Від імені обох користувачів спробуємо надіслати текстове повідомлення до створеної сесії. Для цього кожен користувач має ввести текст повідомлення у відповідне поле вводу в клієнтському інтерфейсі додатка та натиснути кнопку «Send» (3.36):

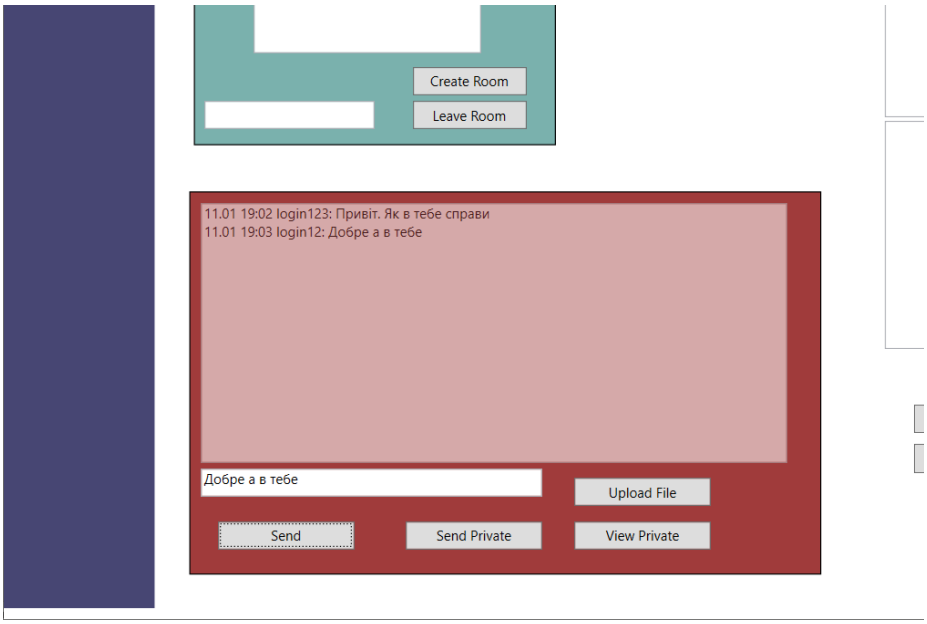


Рисунок 3.36 – Відправлення текстового повідомлення

Після натискання запит надійде до сервера бізнес-логіки, який перевірить, чи користувач має доступ до вказаної сесії, і у разі успішної перевірки передасть повідомлення на сервер повідомлень. Якщо один із користувачів надішле нове повідомлення, воно також відобразиться у вікні обох клієнтів (рис. 3.37). Далі створимо нового 3 користувача:

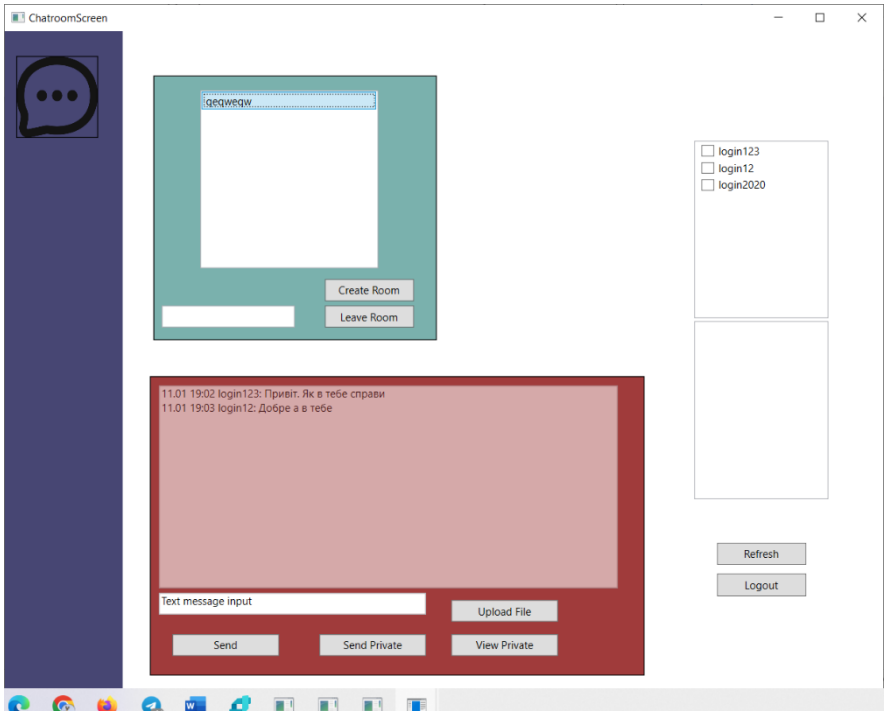


Рисунок 3.37 – Приєднання 3 користувача

Навіть якщо 3 користувач приєднався пізніше ніж інші, до чат групи, він зможе бачити всі попередні повідомлення інших користувачів. Далі спробуємо відправити приватне повідомлення. Для таких дій потрібно відмітивши конкретного користувача - login123 (рис. 3.38):

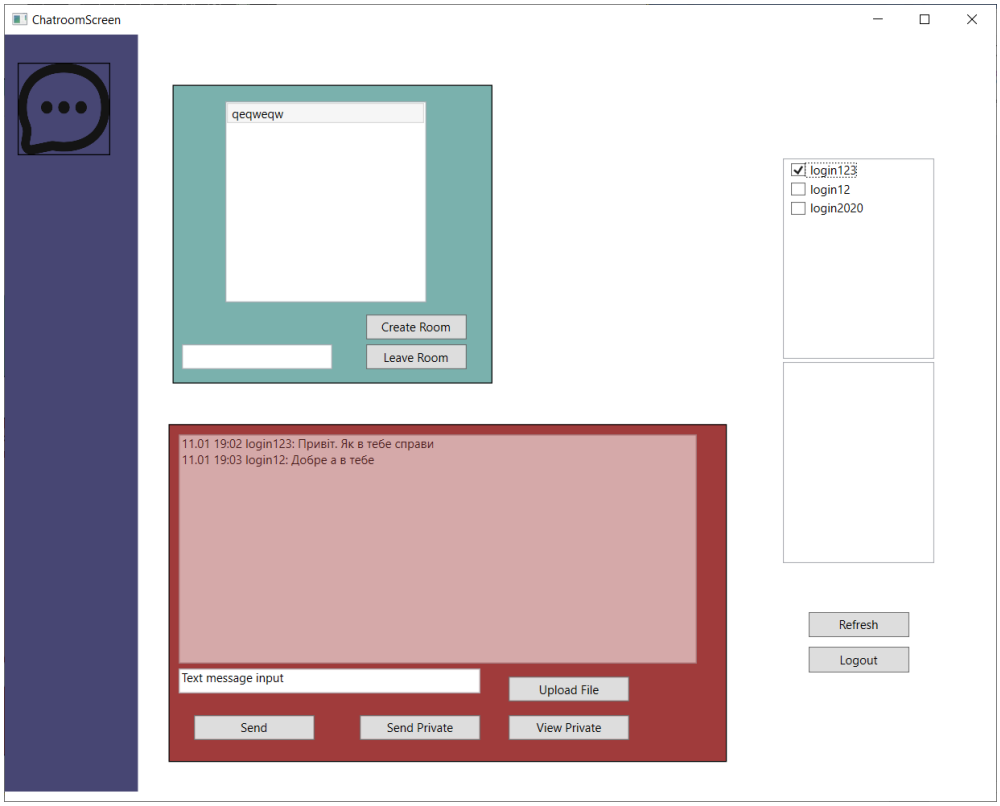


Рисунок 3.38 – Вибір користувача для приватного листування

Наступне одному з користувачів необхідно натиснути кнопку «Send Private», щоб надіслати приватне повідомлення іншому користувачу. Після цього інший користувач повинен відкрити власне приватне вікно для перегляду повідомлень, натиснувши на кнопку «ViewPrivate» у клієнтському інтерфейсі. У відкритому вікні відобразяться всі отримані приватні повідомлення, включаючи повідомлення, надіслані від імені користувачів login2020 та login123. Це дозволяє переглядати історію особистої переписки між користувачами, підтверджуючи коректність обробки приватних повідомлень у системі. (рис. 3.39):

Рисунок 3.39 – Перегляд приватних повідомлень

Після попередніх дій перевірка функціональності файлового менеджера додатка. Для цього необхідно додати новий файл до активної сесії. Щоб здійснити це, користувач має натиснути кнопку «UploadFile», після чого з’явиться діалогове вікно вибору файлу. Користувач може вказати шлях до необхідного файлу на своєму пристрої, після чого файл буде завантажений у вибрану чат-кімнату (рис. 3.40):

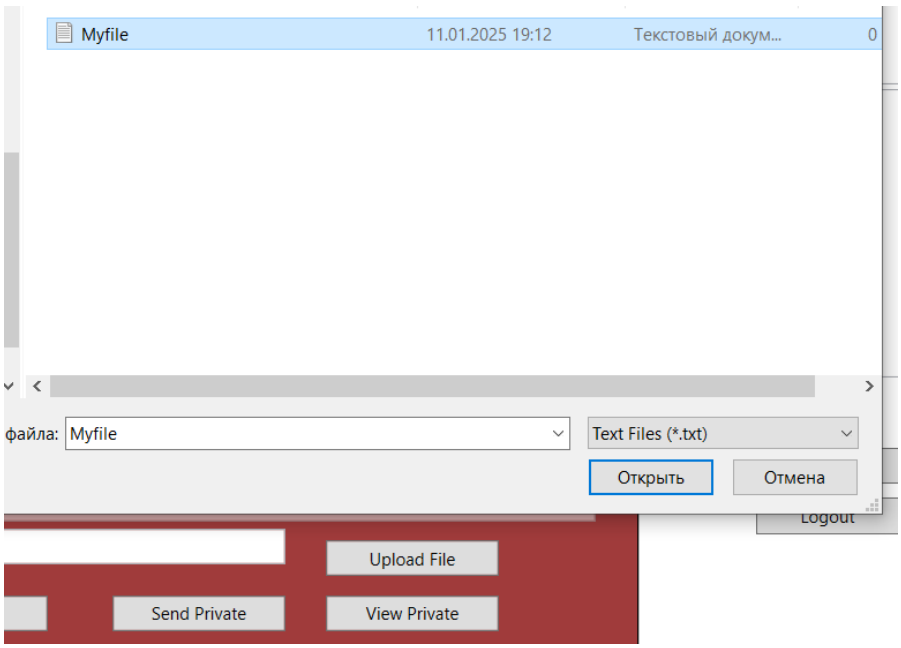


Рисунок 3.40 – Додавання файлу в мережу

Після успішного додавання файлу до активної сесії всі користувачі, підключені до цієї кімнати, зможуть переглядати завантажений файл у правій частині інтерфейсу програми. Файл автоматично відобразиться у списку доступних для цієї сесії (рис. 3.41):

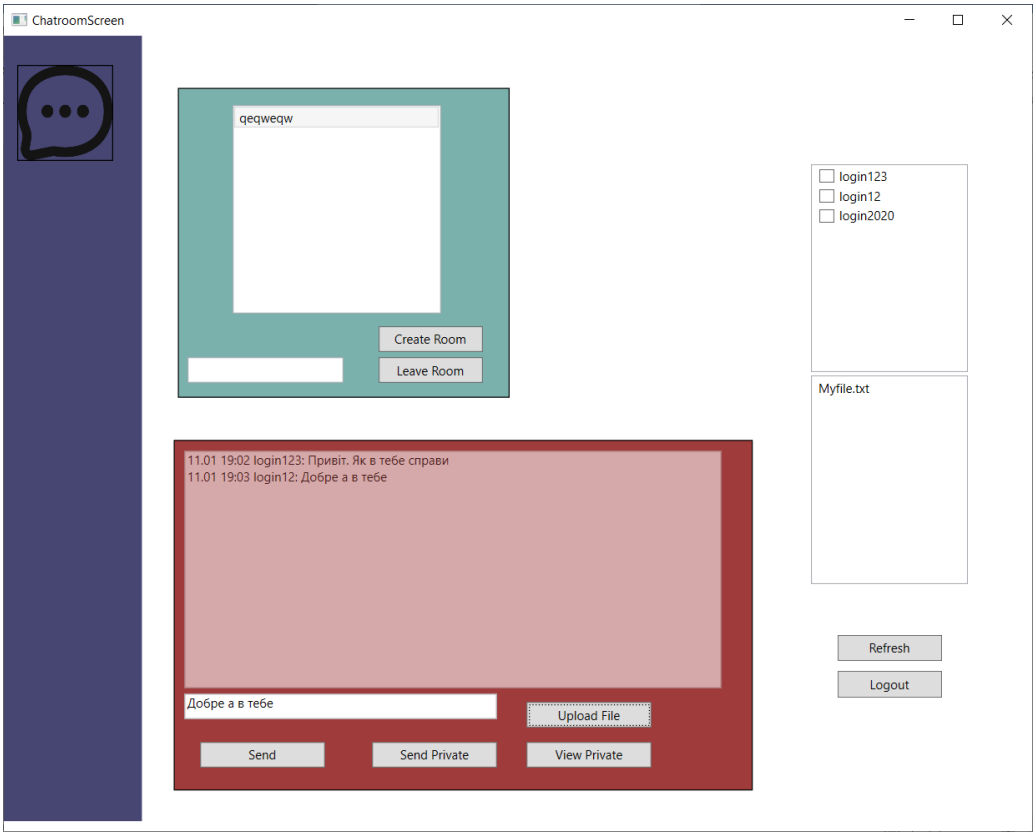


Рисунок 3.41 – Список доданих файлів користувачами

Щоб збереження файлу на своєму пристрої інший користувач може двічі натиснути на вибраний файл у списку доступних. Після цього з'явиться діалогове вікно, де користувач матиме можливість вибрати директорію та вказати шлях, за яким бажає зберегти файл. (рис. 3.42):

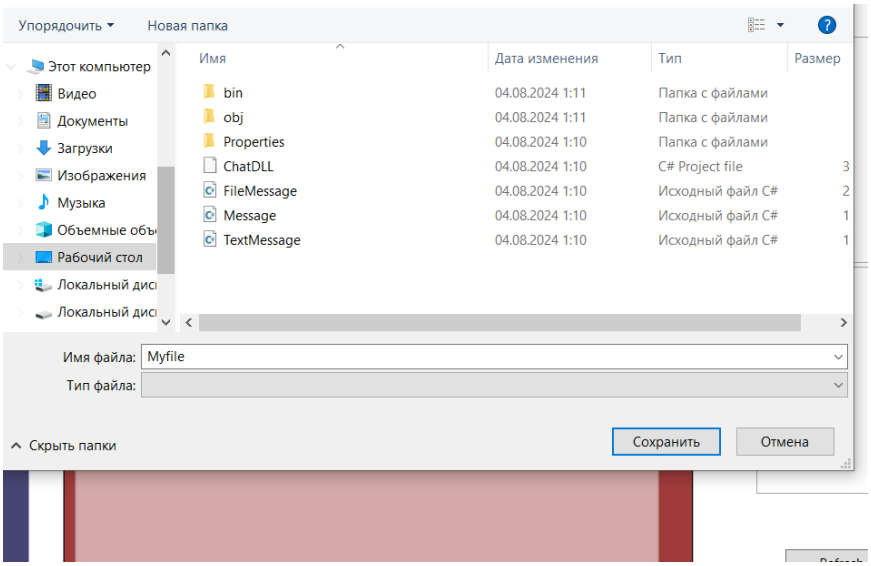


Рисунок 3.42 – Шлях для завантаження файлу

Для того щоб користувач залишив поточну сесію, йому необхідно натиснути кнопку «Leave Room». Після цього він буде видалений із списку учасників цієї чат-кімнати, і решта користувачів зможуть побачити оновлений список, в якому відображатиметься відсутність цього користувача. (рис. 3.43):

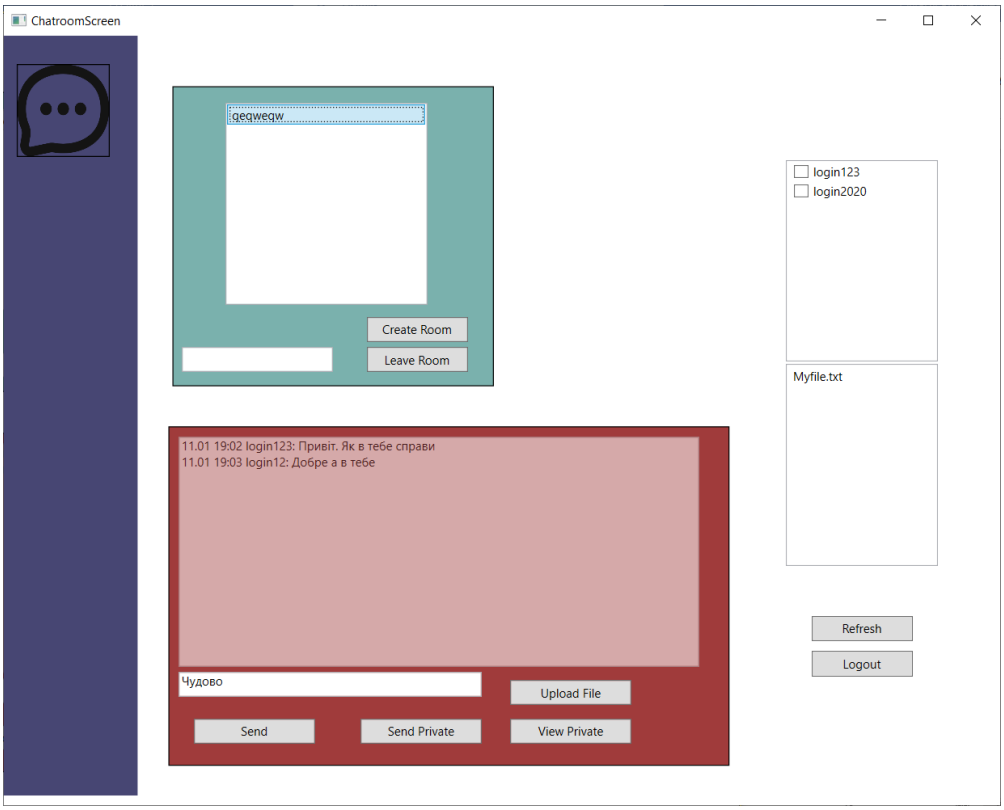


Рисунок 3.43 – Вихід з активної сесії

3.6 Висновок до третього розділу

В третьому розділі було розроблено клієнт-серверний додаток з архітектурою, що складається з трьох мікросервісів: сервера повідомлень, файлового сервера та сервера бізнес-логіки. Додаток забезпечує обмін текстовими повідомленнями, надсилання приватних повідомлень, завантаження та збереження файлів, а також керування чат-кімнатами через зручний WPF-інтерфейс. Мікросервісна структура гарантує стабільність, масштабованість та безпеку, що робить систему ефективною для використання у корпоративних мережах.

Інноваційність підходу полягає у створенні власного програмного рішення з використанням клієнт-серверної архітектури, яке функціонує виключно в локальному середовищі компанії. Це забезпечує повний контроль над процесами обміну даними та усуває залежність від сторонніх програмних продуктів, які потенційно можуть збирати дані про діяльність компанії, включаючи кількість персоналу та продуктивність.

Окрему цінність представляє підхід до організації мережевої взаємодії без використання централізованого сервера. У цьому сценарії кожен учасник мережі самостійно передає дані всім іншим, застосовуючи методи широкомовлення. Такий підхід дозволяє значно зекономити на ресурсах, зазвичай витрачених на підтримку сервера. Проте, він ускладнює клієнтську частину додатка через необхідність обробки більш складної логіки передачі даних. Водночас, поєднання функцій клієнта і сервера може бути виправданим у сценаріях підключення «точка-точка», де обидві сторони здатні динамічно змінювати ролі, забезпечуючи гнучкість та ефективність у взаємодії.

Запропонований підхід демонструє практичне значення та інноваційність у сфері інформаційної безпеки, оскільки забезпечує локалізацію даних, мінімізацію ризиків зовнішнього впливу на критичні бізнес-процеси та ресурсоефективність архітектури мережевої взаємодії.

ВИСНОВОК

У ході виконання цієї роботи було проведено комплексний аналіз та розробку клієнт-серверної системи для мережевої взаємодії. Основною метою проєкту було створення програмного забезпечення, яке забезпечує ефективний обмін інформацією між користувачами, зручний інтерфейс та високу продуктивність.

Було детально розглянуто принципи побудови клієнт-серверної архітектури, що дозволило обґрунтовано вибрати централізований підхід для реалізації системи. Такий підхід забезпечує стабільність роботи, контроль над обміном даними та підвищений рівень безпеки.

Вибір мови програмування C# та платформи .NET для реалізації системи був зумовлений їхніми перевагами, зокрема:

- Кросплатформеністю – можливістю розгортання додатків на різних операційних системах.
- Об’єктно-орієнтованим підходом – що сприяє структурованості та зручності підтримки коду.
- Високим рівнем безпеки – завдяки статичній типізації та вбудованим механізмам управління пам’яттю.
- Інструментами для розробки – використання Visual Studio та ASP.NET для ефективної розробки серверної частини та WPF для клієнтської частини.

Практичний результат роботи включає розробку функціонального клієнт-серверного додатка, який складається з:

1. Серверної частини, що обробляє запити клієнтів, керує обміном даними та синхронізує інформацію між учасниками мережі.
2. Клієнтської частини, реалізованої у вигляді WPF-додатка з інтуїтивно зрозумілим інтерфейсом для користувачів.

3. Інструментів взаємодії, що включають використання web-служб для обміну даними та черг повідомлень для синхронізації даних у реальному часі.

Розроблена система відповідає сучасним стандартам програмної інженерії, забезпечує стабільну роботу в умовах великої кількості користувачів, а також демонструє високу масштабованість та надійність.

Досягнуті результати підтверджують успішне виконання поставлених завдань, зокрема:

- Аналіз та вибір архітектурних підходів для розробки мережевої системи.
- Вибір оптимальних інструментів для реалізації серверної та клієнтської частин.
- Створення стабільного програмного продукту, здатного працювати у багатокористувацькому середовищі.

Таким чином, розроблене програмне забезпечення може бути використане для побудови ефективних систем комунікації, зокрема у корпоративному середовищі або для локального обміну даними.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Buford J.F. P2P Networking and Aplications: підручник / J.F. Buford, H. Yu, Eng Keong Lua; Elsevier Inc.– USA: 2009. – 396 p.

2. P. Mell, T. Grance, “The NIST Definition of Cloud Computing,” National Institute of Standards and Technology, U. S. Department of Commerce, September 2011.

3. Ghani & Naghmeh Niknejad & Seung Ryul Jeong, (2015), “Energy saving in green cloud computing data centers: a review”, Journal of Theoretical and Applied Information Technology, ISSN: 1992- 8645, pages 16-30.

4. Sidra Aslam , Munam Ali Shah (2015), “Load Balancing Algorithms in Cloud Computing: A Survey of Modern Techniques”, 2015 National Software Engineering Conference (NSEC 2015) on 17-17 Dec. 2015, DOI: 10.1109/NSEC.2015.7396341, IEEE 2015.

5. Network Assistant [Електронний ресурс]. – Режим доступу: http://www.gracebyte.com/nassi

6. Collabedit [Електронний ресурс]. – Режим доступу: http://collabedit.com/

7. Confluence [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/ru/software/confluence

8. Richards M., Software Architecture Patterns. O’Reilly Media, Inc., 2015. 54p.

9. ASP.NET documentation. Microsoft. – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/overview

10. Client-Server Architecture. Sciencedirect. – [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/topics/computer-science/client-server-architecture

11. C# Programming Language. Geekforgeeks. – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/csharp-programming-language/

12. MVC Framework Introduction – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-framework-introduction/

13. Introduction to MVVM Architecture – [Електронний ресурс] – Режим доступу: https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679

14. Understanding VIPER Architecture – [Електронний ресурс] – Режим доступу: https://medium.com/@pinarkocak/understanding-viper-pattern-619fa9a0b1f1

15. Офіційний документація WPF [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0