МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «УНІВЕРСИТЕТ «КРОК»»
Фаховий коледж Університету «КРОК»

ДИПЛОМНА РОБОТА

За темою

«Створення програмного додатку для синхронного перегляду відео мовою C#»

Студент 4 курсу групи ІПЗ-20к-1

Нагребельний Олександр Олександрович

(прізвище, ім'я та бо батькові)

Керівник дипломної роботи

к.т.н., доцент

(посада керівника)

Добришин Ю. Є.

(прізвище, ім'я та бо батькові керівника)

До захисту .

(резолюція «До захисту»)

(підпис студента)

10.06.2024

(дата)

(підпис викладача)

Київ, 2024 рік

Скорочення

TCP (Transmission Control Protocol) - протокол керування передачею даних

TCP - це один з основоположних протоколів Інтернету, який використовується для надійної передачі даних між двома пристроями. Він працює на транспортному рівні моделі OSI, гарантуючи, що дані доставляються без помилок і в правильному порядку.

Хост (від англ. *host* — господар, який приймає гостей) - Комп'ютерний сервер, тобто пристрій, що містить ресурс і надає до нього доступ у форматі клієнт-сервер.

Клієнт — апаратний або програмний компонент обчислювальної системи, який надсилає запити серверу. Програма-клієнт взаємодіє з сервером, використовуючи певний протокол. Вона може запитувати з сервера будь-які дані, маніпулювати даними безпосередньо на сервері, запускати на сервері нові процеси і т. п. Отримані від сервера дані клієнтська програма може надавати користувачеві або використовувати як-небудь інакше, в залежності від призначення програми.

Зміст

Вступ

В останні роки все більше людей виявляють зацікавленість у використанні програмного забезпечення для синхронного перегляду відео. Створення програмного додатку для таких потреб може починатися з базових функцій синхронізації відтворення відео та може розвиватися до комплексних систем з можливістю обговорення, чату, анотацій та інших інтерактивних функцій.

Однак, незважаючи на розмаїття доступних варіантів, багато користувачів стикаються з обмеженнями або невдоволені рівнем налаштування, пропонованим цими платформами. Деякі платформи мають обмежений вибір відео, інші ж не пропонують гнучкість у налаштуванні відтворення або спілкування.

Спираючись на ситуацію в сервісах синхронного перегляду відео , було прийнято рішення створити додаток який би допоміг вирішити ситуацію потрібності спеціалізованого додатку , для вирішення проблеми було створено додаток який допомагає переглядати відео синхронно в локальній мережі , або будь яким емулятором локальної мережі .

Розділ 1. Аналіз існуючої інформації щодо теми дипломної роботи

1.1. Порівняльний аналіз існуючої інформації та рішень

Провівши аналіз існуючої інформації про готові рішення в сфері синхронного перегляду відео було знайдено безліч різних сервісів різного призначення , всі сервіси мають різні рівні налаштування , деякі сервіси мають ну дуже вузьку підтримку контенту , деякі мають широку підтримку , але в них може не бути функціоналу який є у якомусь іншому рішенні , поміж всіх цих сервісів було обрано декілька для прикладу:

1. **Netflix Party**:
    - *Функціональність*: Дозволяє користувачам синхронно дивитися фільми та серіали на Netflix, обмінюватися повідомленнями у чаті.
    - *Підтримка*: Працює лише з Netflix.
    - *Безпека*: Користувачі мають можливість створювати приватні сесії зі своїми друзями за допомогою унікального посилання, що зменшує ризик несанкціонованого доступу.
2. **Watch2Gether**:
    - *Функціональність*: Дозволяє користувачам синхронно дивитися відео з YouTube, Vimeo, SoundCloud та інших джерел, а також спілкуватися у чаті або через голосовий зв'язок.
    - *Підтримка*: Підтримує різні джерела відео та аудіо контенту з Інтернету.
    - *Безпека*: Залежить від приватності кожного конкретного випадку; користувачі можуть створювати приватні кімнати.

3. **Rave**:

   - *Функціональність*: Дозволяє синхронно дивитися відео з YouTube, Netflix, Vimeo, та інших джерел, а також слухати музику з Spotify. Має можливість голосового чату.
   - *Підтримка*: Підтримує різні джерела відео та музики з Інтернету.
   - *Безпека*: Приватність залежить від того, як користувачі налаштують доступ до своїх сесій.

4. **Discord** (з функцією "Go Live"):

   - *Функціональність*: Користувачі можуть транслювати свій екран та звук у реальному часі, що дозволяє синхронно дивитися відео або грати в ігри разом з іншими.
   - *Підтримка*: Широкі можливості для синхронного спілкування та перегляду контенту.
   - *Безпека*: Доступно налаштування приватності серверів та каналів, що забезпечує певний рівень захисту.

5. **Zoom** (з функцією демонстрації екрану):

   - *Функціональність*: Крім відеоконференцій, користувачі можуть демонструвати свій екран, що дозволяє синхронно дивитися відео, презентації та інші файли.
   - *Підтримка*: Популярний і зручний для відеоконференцій та синхронного перегляду контенту.
   - *Безпека*: Існують можливості налаштування безпеки конференцій та обміну контентом.

В цьому переліку є сервіси які дають змогу синхронно переглядати відео на різних платформах для відеохостингу , стрімінгових сервісах , також є сервіси які дають змогу демонструвати свій екран.

Відеохостинг є онлайн-платформою, яка надає функціонал завантажувати, зберігати, відтворювати. Процес роботи відеохостингу зазвичай включає наступні етапи:

1. **Завантаження відео**: Користувач завантажує відеофайл на платформу відеохостингу. Це може бути як відео, створене користувачем, так і контент, що був створений іншими користувачами чи виробниками контенту.
2. **Обробка та зберігання**: Після завантаження відео буде оброблене платформою в залежності від потреб платформи. Відео також буде зберігатися на серверах відеохостингу для подальшого відтворення.
3. **Відтворення**: Після обробки відео користувачі можуть переглядати відео через веб-браузери чи додатки, які надає відеохостинг. Це може бути як індивідуальний перегляд, так і синхронний перегляд з друзями чи спільнотою.

Сервіси для спільного перегляду відео, такі як Netflix Party, Watch2Gether або Rave, реалізують можливість синхронного перегляду через інтеграцію з відеохостингами або іншими платформами для відео. Вони надають інтерфейс, який дозволяє користувачам синхронно переглядати вміст та взаємодіяти під час перегляду, незважаючи на те, що вони можуть знаходитися в різних місцях.

Стрімінговий сервіс працює, використовуючи технології мережі Інтернет для передачі відео-, аудіо- та іншого мультимедійного контенту до користувача в режимі реального часу. Ось основні кроки, які здійснює стрімінговий сервіс:

1. **Надання контенту**: Стрімінгові сервіси мають великі серверні ферми, на яких зберігається контент у відповідних форматах, готових для передачі.
2. **Кодування контенту**: Вміст кодується у спеціальні формати, які оптимізовані для швидкої передачі через Інтернет, такі як H.264 або новіші стандарти, наприклад, HEVC (H.265).
3. **Передача контенту**: Коли користувач вибирає відео для перегляду, він надсилає запит до серверів стрімінгового сервісу. Сервери починають передавати відео у фрагментах, які називаються потоковими пакетами, до пристрою користувача через Інтернет.
4. **Буферизація і відтворення**: Пристрій користувача отримує потокові пакети , потім зберігає їх у буфері перед відтворенням. Відтворення розпочинається, коли в буфері накопичується достатньо даних для безперебійного відтворення.

Сервіси для синхронного перегляду використовують подібний принцип, але додають функціональність спільного доступу до контенту між декількома користувачами. Наприклад, коли користувачі дивляться відео разом через Netflix Party, Watch2Gether або інші аналогічні сервіси, кожен з них підключається до одного джерела контенту (наприклад, YouTube або Netflix) і синхронно отримує той самий потік відео.

Демонстрація екрану це функціональність, яка дозволяє одному користувачеві транслювати свій екран у реальному часі для інших учасників сесії. Це є потужний інструмент для синхронного перегляду відео , оскільки один користувач може відтворювати відео на своєму екрані, а інші учасники можуть переглядати те, що відтворюється на екрані першого користувача.

Наприклад, у Discord функція "Go Live" дозволяє користувачам транслювати свій екран разом зі звуком. Інші учасники можуть приєднатися до трансляції і переглядати вміст, який показується на екрані транслюючого користувача.

1.2. Постановка завдання на проектування

Постановка завдання

Назва проекту: Створення програмного додатку для синхронного перегляду відео

Мета проекту: Розробити програмне забезпечення, що дозволить користувачам синхронно переглядати відео в локальній мережі , або за допомогою емуляторів локальної мережі.

Опис проекту:

- Програмний додаток буде забезпечувати можливість одночасного перегляду відео для користувачів, які знаходяться в різних місцях.
- Додаток буде забезпечувати можливість керування відтворенням відео, включаючи паузу, перемотування та інші функції.
- Для зручності користувачів, програмний додаток буде підтримувати різні формати відео.

Основні функціональні вимоги:

1. Синхронне відтворення відео для усіх учасників сеансу.
2. Керування відтворенням відео (пауза, перемотування, регулювання гучності тощо).
3. Підтримка різних форматів відео.

Технічні вимоги:

1. Мова програмування: C#.
2. Фреймворк WPF з середовищем .Net framework .
3. Використання технологій для мережевих взаємодії.
4. Інтерфейс користувача повинен бути зрозумілим та зручним у використанні.

Очікуваний результат: Створення програмного додатку, який забезпечує синхронний перегляд відео у реальному часі для користувачів з різних місць розташування. Додаток повинен бути стабільним, ефективним у використанні та забезпечувати задекларовані функції.

Розділ 2. Проектні і технічні рішення. Види забезпечення

2.1. Інформаційне забезпечення

2.1.1. Аналіз інформаційних потоків

У рамках дипломної роботи проведено детальний аналіз інформаційних потоків, що виникають у процесі синхронного перегляду відео.

Відеопотік

Відеопотік є основним потоком даних у системі синхронного перегляду відео. Цей потік містить відеоконтент, який одночасно переглядається всіма учасниками. Відеодані передаються в реальному часі з одного комп'ютера до іншого, забезпечуючи синхронний перегляд вмісту.

Аудіопотік

Аудіопотік також є важливим елементом у системі синхронного перегляду відео. Він містить аудіоконтент, який супроводжує відеоряд. Подібно до відеопотоку, аудіодані передаються в реальному часі, забезпечуючи синхронний звуковий супровід під час перегляду.

Потік керування

Потік керування відіграє важливу роль у забезпеченні синхронності дій між учасниками. Цей потік містить команди керування відтворенням, такі як пауза, відновлення, перемотування тощо.

Потік інформації про стан

Потік інформації про стан відтворення відео відіграє ключову роль у забезпеченні синхронності між учасниками. Цей потік містить інформацію про позицію у відео, статус відтворення та інші параметри, які дозволяють учасникам узгоджувати свої дії та забезпечувати однаковий стан відтворення на всіх пристроях.

2.1.2 Діаграми прецедентів , дій та послідовностей

2.1.2.1 Діаграма прецедентів

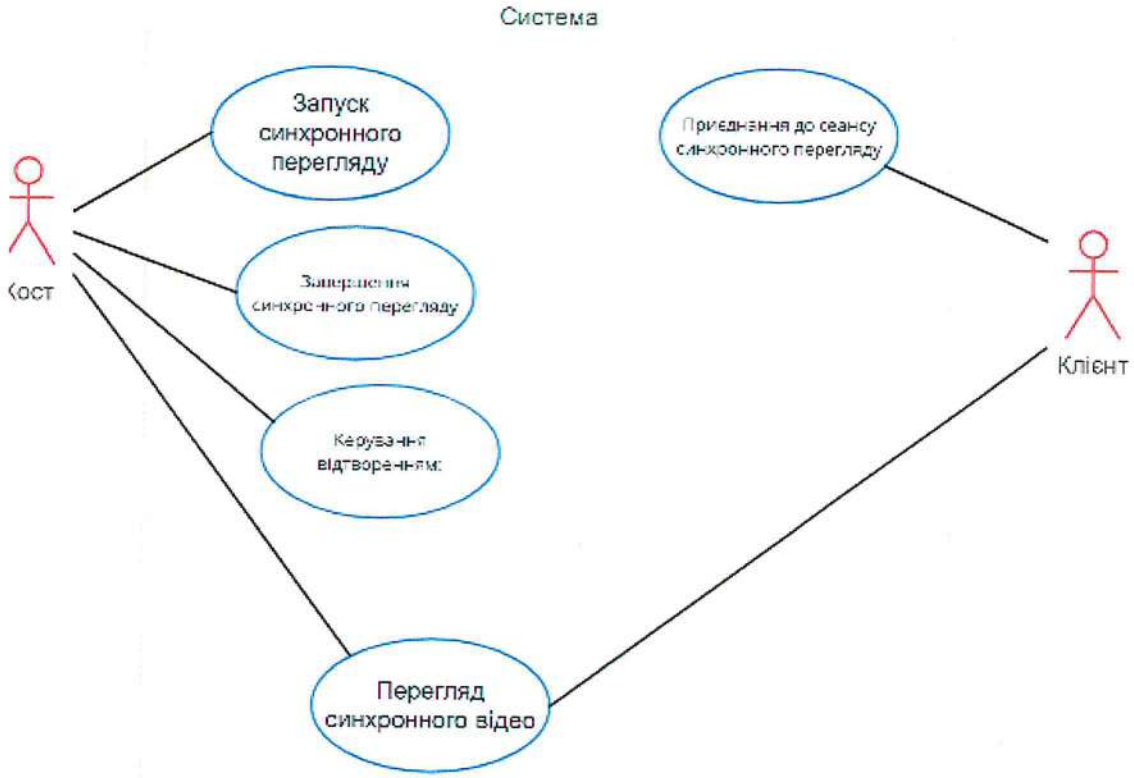

Рис. 2.1 Діаграма прецедентів

2.1.2.2 Діаграма дій Хоста

Рис. 2.2 Діаграма дій Хоста

2.1.2.3 Діаграма дій Клієнта

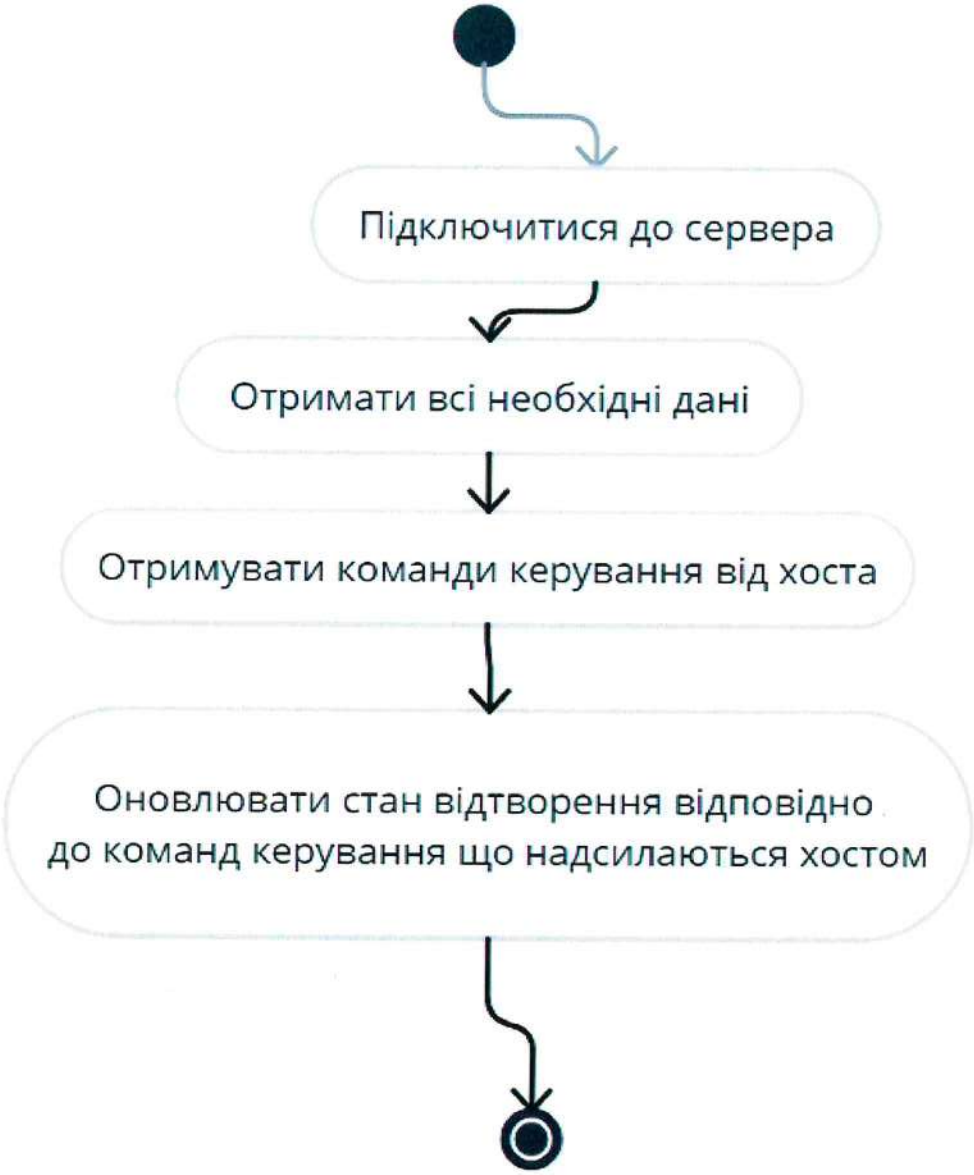

Рис. 2.3 Діаграма дій Клієнта

2.1.2.4 Діаграма Послідовностей

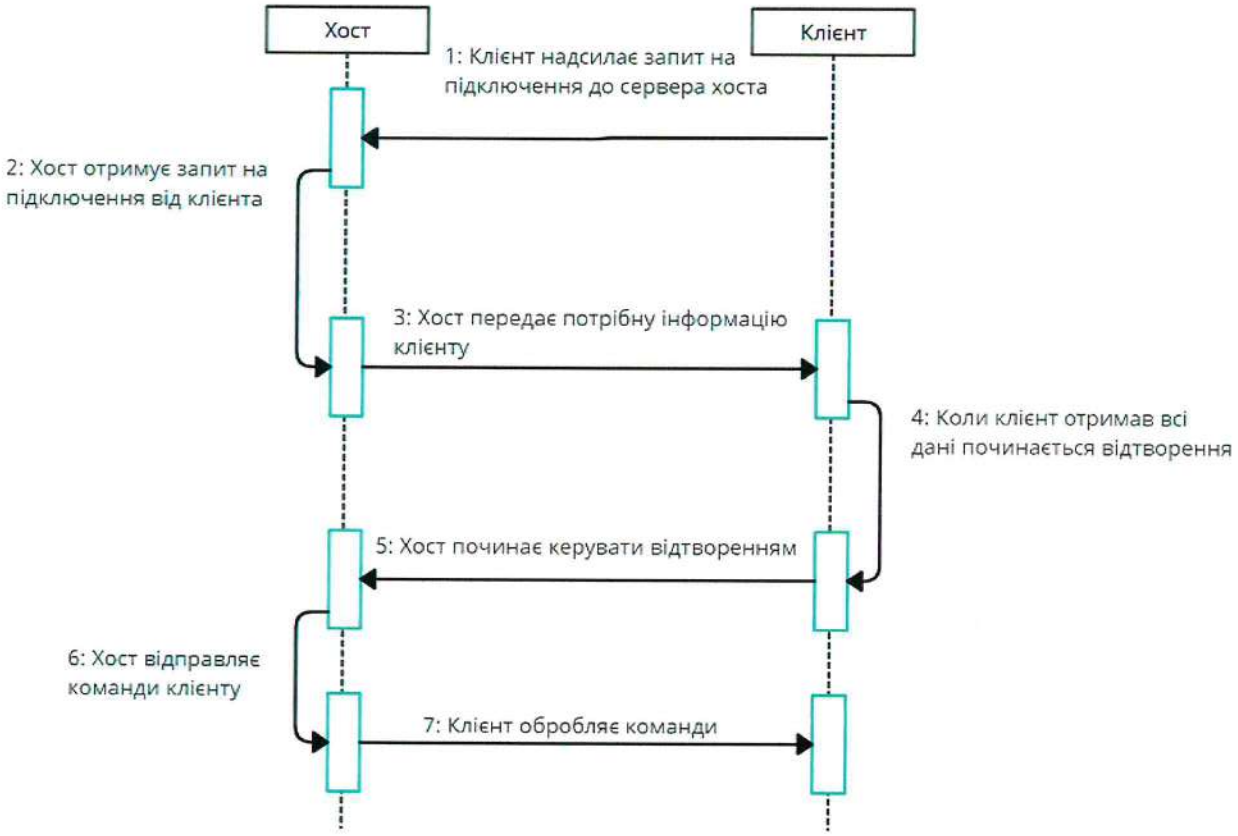

Рис. 2.4 Діаграма дій Послідовностей

2.1.3. Система інформації

У рамках розділу про систему інформації програмного додатку для синхронного перегляду відео мовою C# були розглянуті компоненти системи та їх взаємодія, інформаційні потоки та дані, що використовуються, а також вимоги до системи та технології її реалізації.

Компоненти системи інформації:

1. Модуль хоста:

- Відповідає за запуск синхронного перегляду.
- Обробляє команди керування відтворенням та оновлює стан відтворення на всіх комп'ютерах учасників.

2. **Модуль клієнта**:

- Підключається до хоста та отримує відео та інформацію про стан.
- Надсилає команди керування відтворенням та оновлює стан відтворення.

3. Модуль керування мережею:

- Встановлює мережеве з'єднання між хостом та клієнтами.

Взаємодія компонентів системи інформації:

Хост:

- Вибирає відео та запускає сервер.
- Очікує на підключення клієнтів та надсилає їм потрібні дані.
- Обробляє команди керування відтворенням та оновлює стан відтворення.

Клієнт:

- Підключається до сервера хоста та отримує необхідні дані.
- Відтворює відео та отримує команди керування.
- Оновлює стан відтворення відповідно до отриманих команд.

Інформаційні потоки:

1. Відеопотік: від хоста до клієнтів.
2. Потік керування: від хоста до клієнтів.
3. Потік інформації про стан: від хоста до клієнтів.

Дані, що використовуються в системі інформації:

1. Відеофайли
2. IP-адреси учасників
3. Позиція у відео
4. Статус відтворення (пауза, відновлення, перемотування)

Технології реалізації системи інформації:

1. .NET Framework
2. C#
3. WPF
4. TCP

Що таке .NET Framework?

.NET Framework - це платформа програмного забезпечення, розроблена компанією Microsoft, яка використовується для створення різних типів програм, таких як:

- **Клієнтські програми:** програми, які запускаються на комп'ютері користувача, такі як десктопні програми та ігри.
- **Веб-застосунки:** програми, які запускаються в браузері, такі як веб-сайти.
- **Веб-служби:** програми, які надають дані та функціональні можливості іншим програмам через Інтернет.
- **Мобільні програми:** програми, які запускаються на мобільних пристроях, таких як смартфони та планшети.

Основні характеристики .NET Framework:

• **Загальна мовна середа (CLI):** CLI - це середовище виконання, яке дозволяє різним мовам програмування працювати разом. Це означає, що ви можете використовувати будь-яку мову, що підтримується CLI, для розробки програм .NET, наприклад C#, Visual Basic .NET, F# та інші.

• **Класова бібліотека .NET Framework (FCL):** FCL - це великий набір класів і компонентів, які надають загальні функціональні можливості, такі як доступ до даних, мережеві комунікації та графічний інтерфейс користувача. Це економить час розробників, оскільки їм не потрібно писати код з нуля для цих поширених завдань.

• **Common Language Runtime (CLR):** CLR - це віртуальна машина, яка виконує код .NET. CLR керує пам'яттю, безпекою та іншими важливими аспектами виконання програми.

• **ASP.NET:** ASP.NET - це платформа для розробки веб-застосунків .NET. Вона включає в себе серверні компоненти, які дозволяють розробникам створювати динамічні та масштабовані веб-сайти.

• **Windows Presentation Foundation (WPF):** WPF - це фреймворк для розробки клієнтських програм .NET. Він дозволяє розробникам створювати візуально привабливі та зручні для користувачів програми з графічним інтерфейсом користувача.

Переваги використання .NET Framework:

• **Широкий спектр можливостей:** .NET Framework може використовуватися для створення різних типів програм, що робить його універсальною платформою розробки.

• **Простота використання:** CLI та FCL роблять .NET Framework відносно простою у вивченні та використанні, порівняно з іншими платформами розробки.

• **Продуктивність:** CLR та інші компоненти .NET Framework оптимізовані для забезпечення високої продуктивності програм.

C#: Огляд та основні характеристики

Що таке C#?

C# - це об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Microsoft. Вона схожа на інші мови програмування, такі як C++ і Java, але має деякі унікальні функції, які роблять її популярним вибором для розробки програм .NET.

Основні характеристики C#:

• **Об'єктно-орієнтована:** C# підтримує об'єктно-орієнтоване програмування (ООР), що використовує концепції, такі як класи, об'єкти, спадщина та поліморфізм. ООР робить код більш модульним, повторно використовуваним і простішим у підтримці.

• **Безпечна:** C# - це керована мова, що означає, що пам'ять автоматично керується віртуальною машиною. Це допомагає запобігти поширеним помилкам програмування, таким як витоки пам'яті та недійсні посилання.

• **Компільована:** C# - це компільована мова, це означає, що код C# перекладається в машинний код перед запуском.

• **Сучасна:** C# постійно оновлюється новими функціями та можливостями. Це робить його потужною та гнучкою мовою, яка може використовуватися для створення широкого спектру програм.

Переваги використання C#:

• **Простота використання:** C# - це відносно проста мова для вивчення та використання, навіть для початківців.

• **Продуктивність:** C# - це компільована мова, яка робить програми C# швидкими та ефективними.

• **Безпека:** C# - це керована мова, яка допомагає запобігти поширеним помилкам програмування.

• **Універсальність:** C# можна використовувати для створення різних типів програм, таких як веб-застосунки, десктопні програми, мобільні програми та ігри.

Що таке WPF?

WPF (Windows Presentation Foundation) - це фреймворк для розробки графічного інтерфейсу користувача (GUI) для Windows, розроблений компанією Microsoft. Він використовується для створення комп'ютерних програм з багатим інтерфейсом користувача.

Основні характеристики WPF:

• **Декларативний:** WPF використовує декларативний підхід до розробки GUI, де інтерфейс користувача описується за допомогою XAML, мови розмітки на основі XML. Це робить код WPF більш читабельним і лаконічним, порівняно з імперативним кодом.

• **Векторна графіка:** WPF використовує векторну графіку для рендерингу елементів інтерфейсу користувача. Це робить графічні елементи WPF масштабованими та чіткими на будь-якому екрані.

• **Зв'язування даних:** WPF підтримує потужне зв'язування даних, яке дозволяє синхронізувати дані програми з елементами інтерфейсу користувача. Це робить код WPF більш динамічним і гнучким.

• **Анімація:** WPF підтримує багаті можливості анімації, які дозволяють розробникам створювати динамічні та візуально привабливі інтерфейси користувача.

• **Стилі:** WPF використовує систему стилів, яка дозволяє розробникам легко налаштовувати зовнішній вигляд елементів інтерфейсу користувача.

Переваги використання WPF:

• **Продуктивність:** WPF використовує апаратне прискорення графіки, що робить програми WPF швидкими та чутливими.

• **Гнучкість:** WPF - це дуже гнучкий фреймворк, який можна використовувати для створення широкого спектру інтерфейсів користувача.

• **Простота використання:** WPF - це відносно простий у використанні фреймворк, навіть для початківців.

TCP: Протокол керування передачею даних

Що таке TCP?

TCP (Transmission Control Protocol) - це протокол мережевого рівня, який використовується для надійної передачі даних між двома комп'ютерами. Він є одним з основних протоколів в Інтернеті, і використовується для широкого спектру програм, таких як веб-браузери, електронна пошта та файловий обмін.

Історія створення TCP:

TCP був розроблений в 1970-х роках Робертом Каном та Вінтоном Серфом як частина набору протоколів TCP/IP. TCP/IP був розроблений як альтернатива більш ранньому протоколу ARPANET, який був ненадійним і не масштабованим.

Перша специфікація TCP була опублікована в 1981 році, і протокол з того часу постійно доопрацьовується. Сьогодні TCP є одним з найпоширеніших протоколів в Інтернеті, і він використовується для мільярдів з'єднань щодня.

Рис. 2.5 Вінтон Серф та Роберт Кан

Основні характеристики TCP:

- **Надійність:** TCP гарантує, що всі дані, які надсилаються одним комп'ютером, будуть доставлені іншому комп'ютеру в правильному порядку. Це робиться за допомогою механізмів підтвердження та повторної передачі.

- **Контроль потоку:** TCP регулює швидкість передачі даних між двома комп'ютерами, щоб запобігти перевантаженню мережі. Це робиться за допомогою механізмів вікна ковзання.

- **Дуплексне з'єднання:** TCP встановлює дуплексне з'єднання між двома комп'ютерами, що дозволяє їм надсилати та отримувати дані одночасно.

- **Гнучкість:** TCP це дуже гнучкий протокол, який використовується для різних типів програм.

Переваги використання TCP:

- **Надійність:** TCP - це дуже надійний протокол, який гарантує, що всі дані будуть доставлені.
- **Контроль потоку:** TCP допомагає запобігти перевантаженню мережі.
- **Дуплексне з'єднання:** TCP дозволяє комп'ютерам надсилати та отримувати дані одночасно.
- **Гнучкість:** TCP може використовуватися для різних типів програм.

2.1.4. Алгоритми функціонування системи

Алгоритм роботи хоста:

1. Хост вибирає відео на своєму комп'ютері.
2. Хост створює сервер по локальному IP.
3. Хост очікує на підключення клієнтів.
4. При підключенні клієнта хост надсилає йому відео , коли відправка пройде почнеться відтворення.
5. Хост обробляє команди керування відтворенням, що надсилаються клієнтам, та оновлює стан відтворення на всіх комп'ютерах учасників.

Алгоритм роботи клієнта:

1. Клієнт підключається до сервера хоста по локальному IP.
2. Клієнт отримує від хоста відео.
3. Клієнт відтворює відео.
4. Клієнт отримує від хоста команди керування відтворенням.
5. Клієнт оновлює стан відтворення відповідно до команд керування, що надсилаються хостом.

2.2. Математичне забезпечення

2.2.1. Особливості розробки алгоритму

Метод низхідного проектування

Для розробки алгоритму використовувався метод низхідного проектування. Цей метод передбачає розбиття задачі на менші, більш прості підзадачі. Кожна підзадача описується своїм алгоритмом.

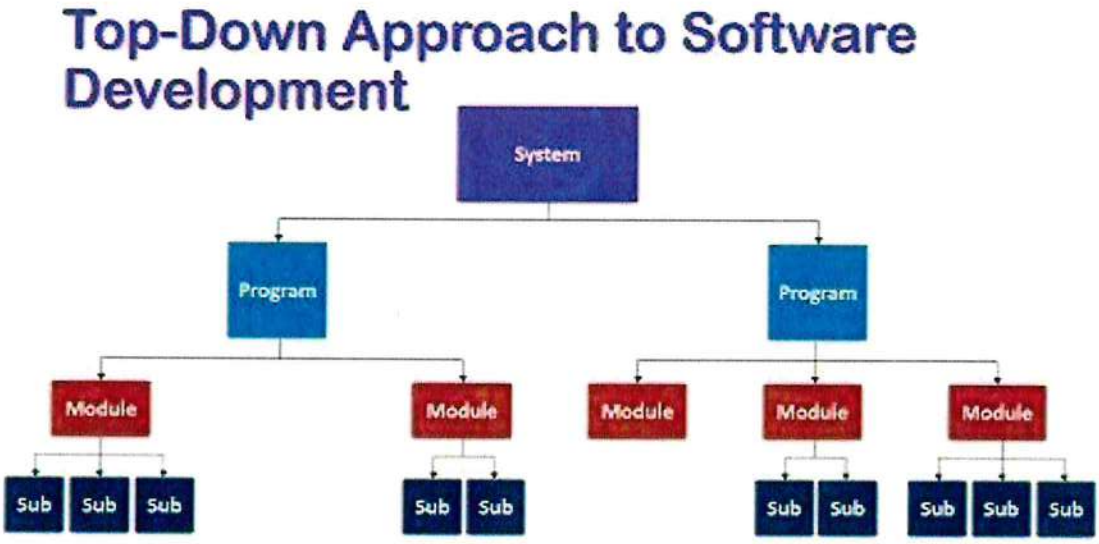

Рис. 2.6 Метод низхідного проектування

Етапи розробки алгоритму

1. **Визначення загального підходу до вирішення задачі.** На цьому етапі було визначено основні етапи синхронного перегляду відео, а також дані, які будуть використовуватися при цьому.

2. **Виділення окремих блоків, які виконують закінчений процес обробки даних.** На цьому етапі задача було розбита на менші підзадачі, кожна з яких буде відповідати за виконання певної функції.

3. **Конкретизація блоків, які були виділені раніше.** На цьому етапі було описано алгоритми роботи кожного з виділених блоків.

4. **Деталізація алгоритмів роботи блоків.** На цьому етапі було описано алгоритми роботи кожного з операторів, які використовуються в алгоритмах роботи блоків.

Приклад алгоритму

Алгоритм роботи хоста:

1. Хост вибирає відео на своєму комп'ютері.

2. Хост створює сервер по локальному IP.

3. Хост очікує на підключення клієнтів.

4. При підключенні клієнта хост надсилає йому відео та інформацію про стан.

5. Запускає відтворення відео.

6. Хост обробляє команди керування відтворенням, що надсилаються клієнтам, та оновлює стан відтворення на всіх комп'ютерах учасників.

Алгоритм роботи клієнта:

1. Клієнт підключається до сервера хоста по локальному IP.

2. Клієнт отримує від хоста відео та інформацію про стан.

3. Клієнт після отримання відео починає відтворення.

4. Клієнт отримує від хоста команди керування відтворенням.

5. Клієнт оновлює стан відтворення відповідно до команд керування, що надсилаються хостом.

2.2.2. Математичні моделі

В рамках дипломної роботи не використовувалися математичні моделі.

2.2.3. Методи розрахунку

В рамках дипломної роботи використовувалися наступні методи розрахунку:

- **Розрахунок пропускної здатності мережі:**

  - Пропускна здатність мережі визначається як максимальна швидкість передачі даних, яка може бути досягнута в даній мережі.

  - Для розрахунку пропускної здатності мережі можна використовувати наступну формулу:

Пропускна здатність = Місткість каналу * Швидкість передачі даних

де:

- **Місткість каналу:** визначається в бітах в секунду (біт/с)
- **Швидкість передачі даних:** визначається в бітах в секунду (біт/с)

- **Розрахунок затримки:**

  - Затримка - це час, який потрібен даним для переміщення з одного вузла мережі до іншого.

  - Для розрахунку затримки можна використовувати наступну формулу:

Затримка = Час передачі даних + Час обробки даних

де:

- **Час передачі даних:** визначається в секундах (с)
- **Час обробки даних:** визначається в секундах (с)

2.3. Програмне забезпечення

2.3.1. Мова програмування

Програмний додаток для синхронного перегляду відео мовою C# розроблений з використанням мови програмування C#. C# є об'єктно-орієнтованою мовою програмування, яка розроблена компанією Microsoft. Вона є однією з найпопулярніших мов програмування у світі завдяки своїй простоті, гнучкості та потужності.

Переваги використання C#:

- **Простота:** C# має просту та зрозумілу синтаксичну структуру, що робить її легкою для вивчення.
- **Гнучкість:** C# підтримує широкий спектр парадигм програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне програмування.
- **Потужність:** C# є потужною мовою програмування, яка може використовуватися для розробки широкого спектру програмного забезпечення, включаючи веб-додатки, десктопні програми, мобільні програми та ігри.

2.3.2. Інструменти розробки

Для розробки програмного додатку для синхронного перегляду відео мовою C# використовувалися наступні інструменти розробки:

- **Visual Studio:** Visual Studio є інтегрованим середовищем розробки (IDE) від компанії Microsoft, яке використовується для розробки програмного забезпечення на мові C#. Visual Studio пропонує широкий спектр функцій, які допомагають розробникам писати код, тестувати його та налагоджувати.
- **.NET Framework:** .NET Framework є платформою для розробки програмного забезпечення від компанії Microsoft, яка використовується для розробки програмного забезпечення мовою C#. .NET Framework пропонує

широкий спектр бібліотек та інструментів, які допомагають розробникам писати код, тестувати його та налагоджувати.

2.3.3. Бібліотеки

В програмному додатку для синхронного перегляду відео мовою C# використовувалась бібліотека XamlAnimatedGif

2.3.4. Архітектура програмного забезпечення

Програмний додаток для синхронного перегляду відео мовою C# має трирівневу архітектуру:

- **Презентаційний рівень:** Презентаційний рівень відповідає за взаємодію з користувачем. Він реалізований за допомогою технології WPF (Windows Presentation Foundation).

- **Рівень бізнес-логіки:** Рівень бізнес-логіки відповідає за виконання основних функцій програми. Він реалізований за допомогою мови програмування C#.

- **Рівень доступу до даних:** Рівень доступу до даних відповідає за доступ до даних, які використовуються програмою. Рівень доступу до даних у програмі забезпечує контроль над доступом до відеоданих, що передаються між хостом та клієнтом. Не зберігаючи дані в базі даних, цей рівень відповідає за безпечну передачу та збереження відео на локальних пристроях.

2.3.5. Алгоритми

Алгоритм роботи хоста:

1. Хост вибирає відео на своєму комп'ютері.

2. Хост створює сервер по локальному IP.

3. Хост очікує на підключення клієнтів.

4. При підключенні клієнта хост надсилає йому відео та інформацію про стан.

5. Запускає відтворення відео.

6. Хост обробляє команди керування відтворенням, що надсилаються клієнтам, та оновлює стан відтворення на всіх комп'ютерах учасників.

Алгоритм роботи клієнта:

1. Клієнт підключається до хоста по локальному IP.

2. Клієнт отримує від хоста відео та інформацію про стан.

3. Клієнт відтворює відео.

4. Клієнт отримує від хоста команди керування відтворенням.

5. Клієнт оновлює стан відтворення відповідно до команд керування, що надсилаються хостом.

Розділ 3. Інструкція користувача щодо роботи з програмним забезпеченням

3.1 Вигляд головного вікна програми

Рис. 3.1 Головне вікно

Спочатку розглянемо головне вікно , тут є 5 елементів , елементи 1 та 2 відповідають за створення сервера , елементи 3 , 4 та 5 відповідають за приєднання до серверу.

Як створити сервер?

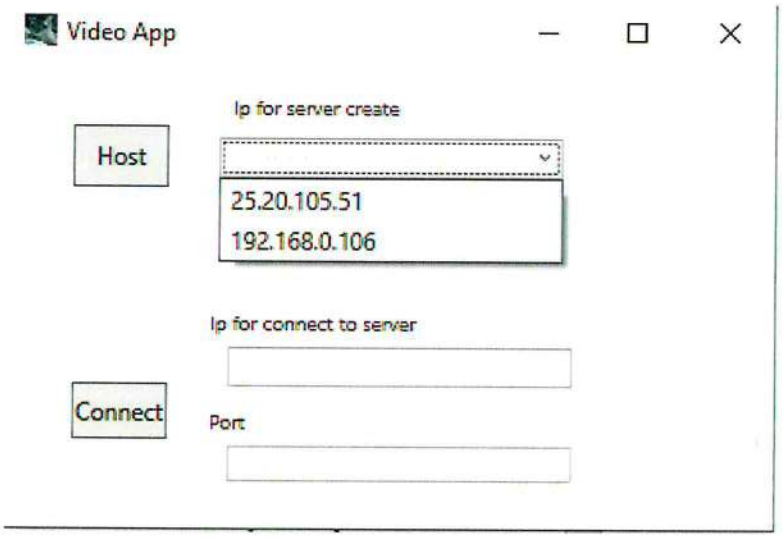

Рис. 3.2 Вибір адреси

Користувач натискає на елемент 1 (рис. 3.1) і вибирає там адресу , в цьому списку будуть присутні локальні адреси , сама локальна адреса комп'ютера та якщо є встановлені емулятори локальної адреси , в даному випадку встановлений емулятор Hamachi і адреса 25.20.105.51 є локальною адресою Hamachi.

Після вибору адреси користувач натискає кнопку Host (Рис. 3.1 елемент 2), після натискання кнопки якщо все було зроблено правильно буде відкрито вікно в якому потрібно вибрати відео файл(Рис. 3.3).

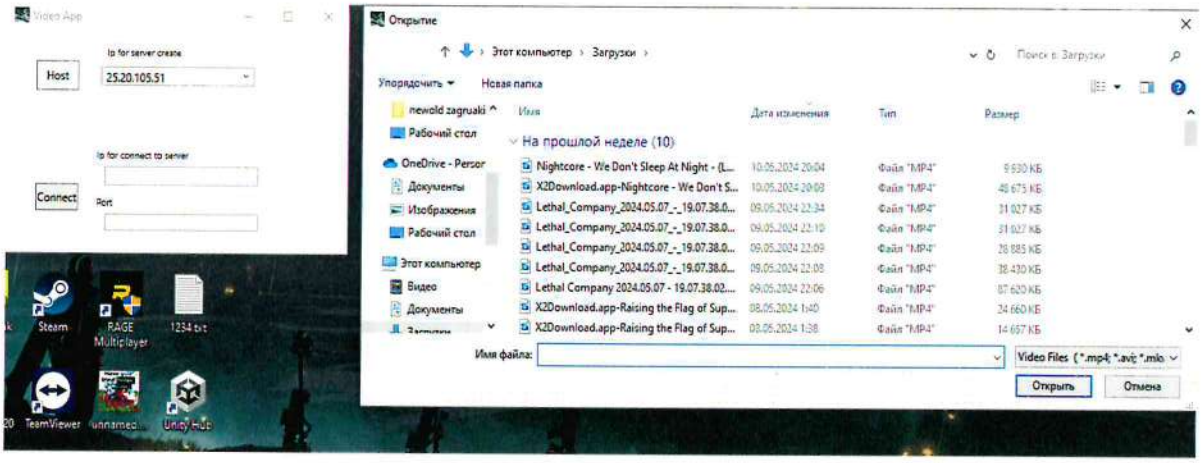

Рис. 3.3 Вікно вибору файлу

Коли користувач вибрав відео для хосту він зможе побачити таке вікно в якому будуть прописані данні які потрібні іншому користувачу клієнту для підключення , також вони будуть скопійовані в буфер обміну(Рис. 3.4).

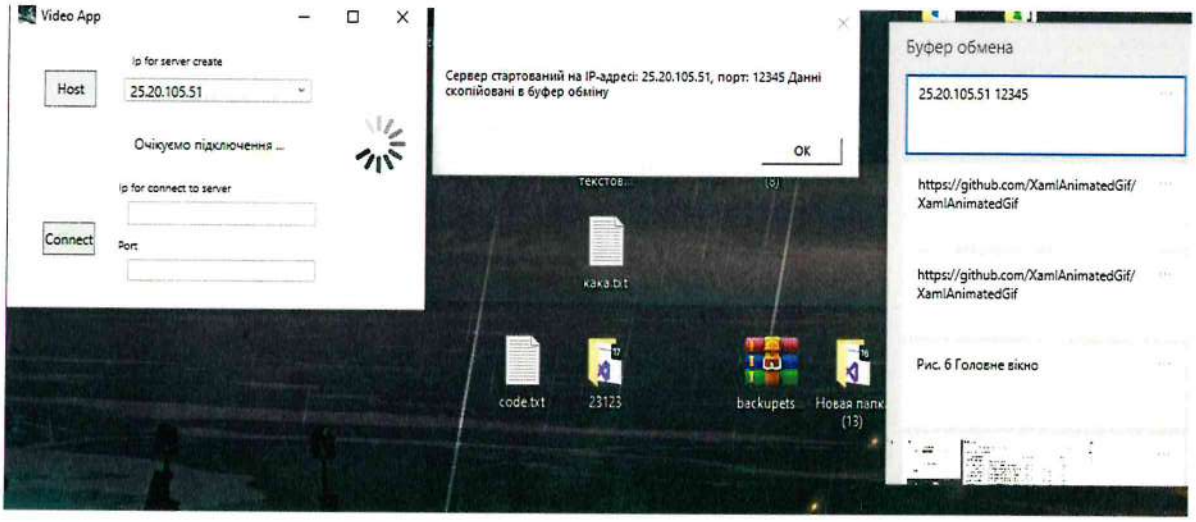

Рис. 3.4 Вікно з інформацією про сервер

Після цього сервер запущений , тепер клієнту потрібно ввести IP адресу та порт яка була отримана від хоста , IP адреса вводиться в поле для IP адреси(рис. 3.1 елемент 3), порт вводиться в поле для порту (рис.6 елемент 4) після цього треба натиснути на кнопку Connect (рис. 3.1 елемент 5) після цього ми будемо мати таку картину(Рис 3.5) .

Рис. 3.5 Вікно з інформацією про сервер

Коли файл відправиться клієнту хост отримає (Рис. 3.6) буде вікно я кому буде сказано що все пройшло успішно.

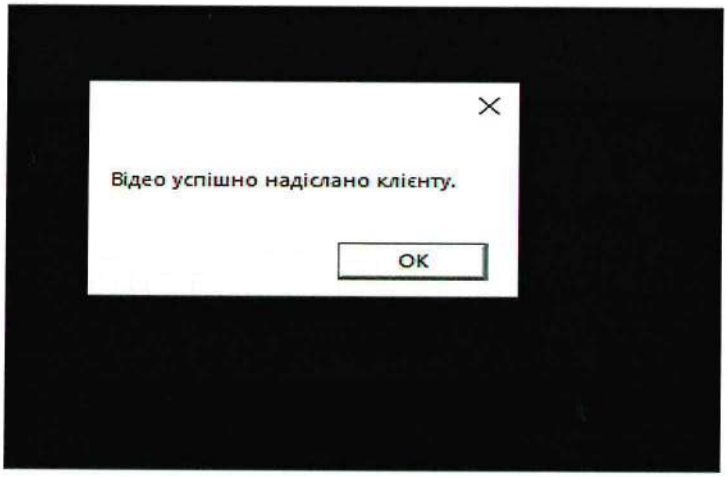

Рис. 3.6 Вікно з підтвердженням

Після натискання кнопки ок (Рис. 3.6) буде нове вікно (Рис. 3.7) в якому потрібно буде вибрати чи потрібно приймати нових клієнтів якщо так то залишається можливість підключення інших клієнтів , якщо ні то залишаються клієнти які підключилися і всі операції контролю відтворення проходять з ними.

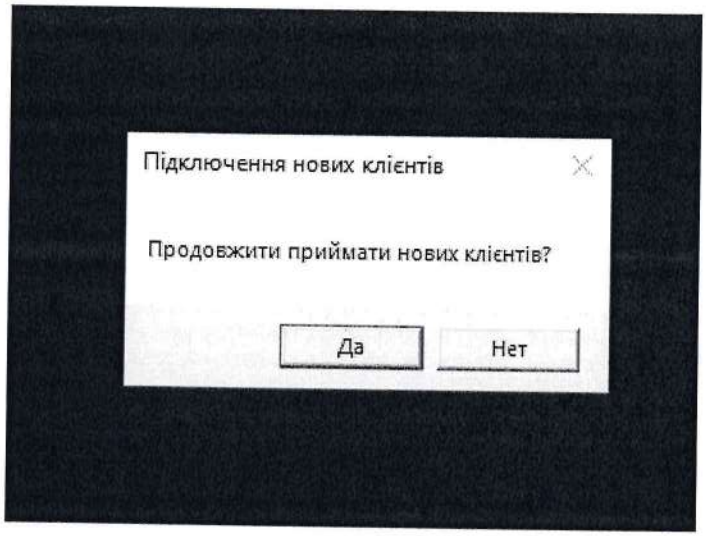

Рис. 3.7 Вікно Підключення нових клієнтів

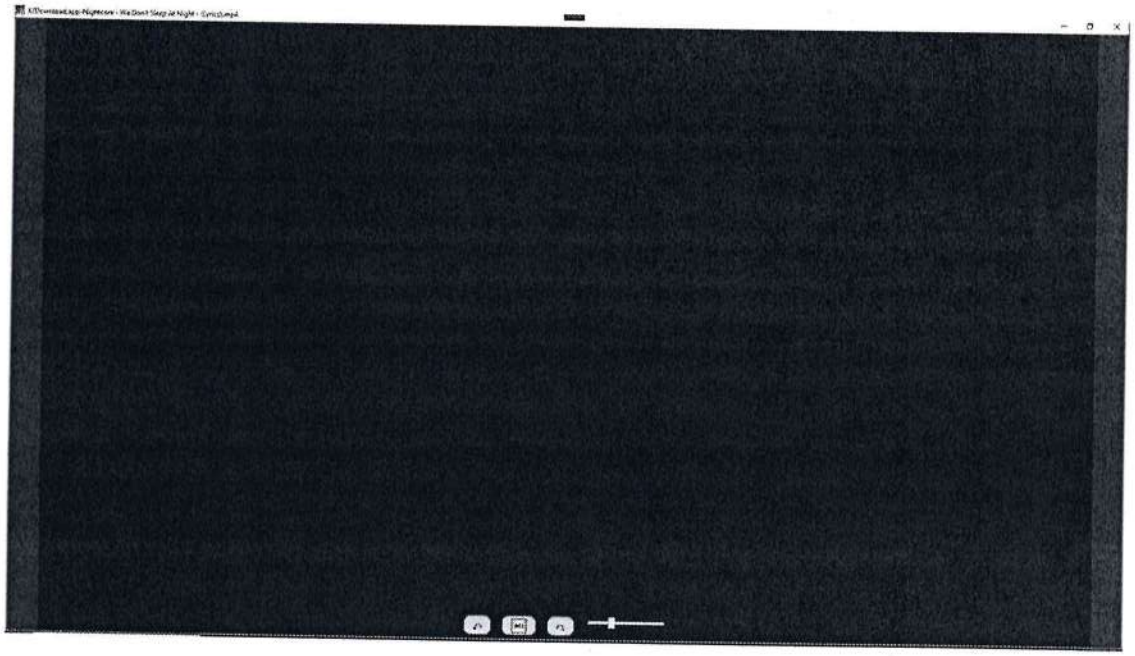

Рис. 3.8 Вікно з відео хоста

Далі ми маємо такий вигляд вікна з відео для хоста (рис. 3.8) тут є такі елементи контролю відтворення як пауза/плей , на 10 секунд раніше , пізніше , звук для кожного вікна унікальний для гнучкого налаштування, також є система приховування елементів контролю , для цього лише потрібно клацнути на будь яку точку вікна , при повторному кліку елементи знову будуть видимі, при натисканні клавіші F вікно увійде у повний екран(Рис. 3.9) , за допомогою клавіші ESC можна вийти з повного екрану до звичайного представлення , за допомогою клавіші F5 можна синхронізувати відео між хостом.

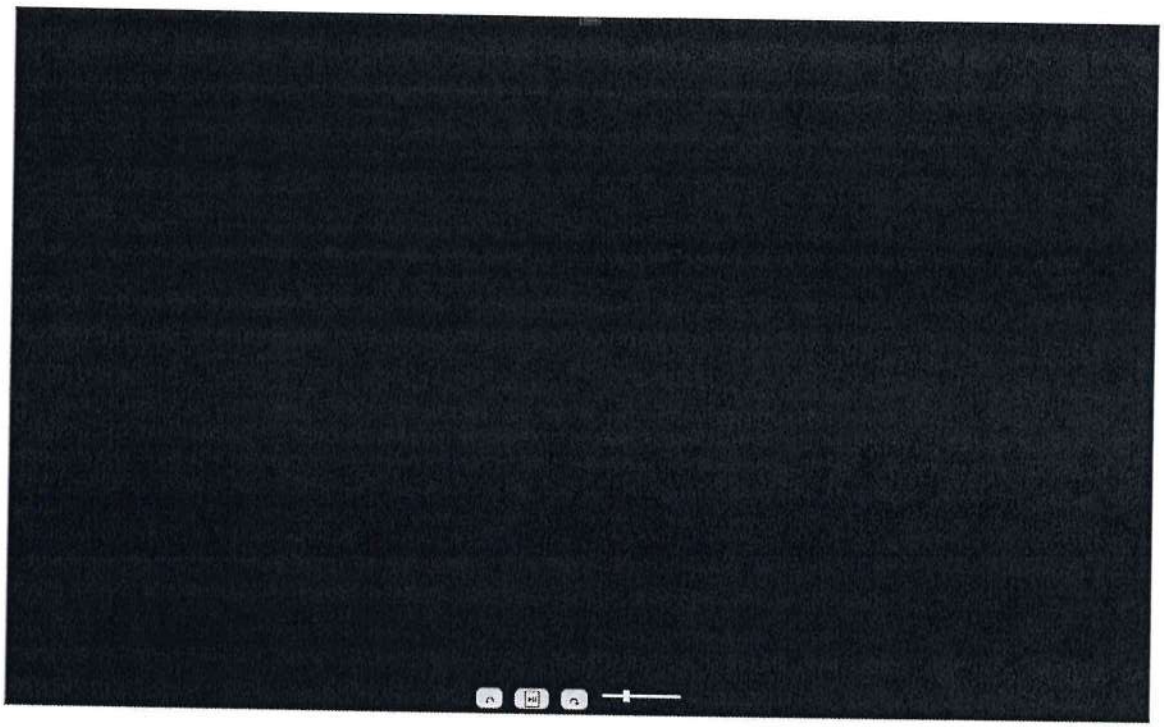

Рис. 3.9 Вікно в повному екрані

Вікно клієнта має налаштування звуку , вхід в повний екран та вихід з нього (Рис. 3.10)

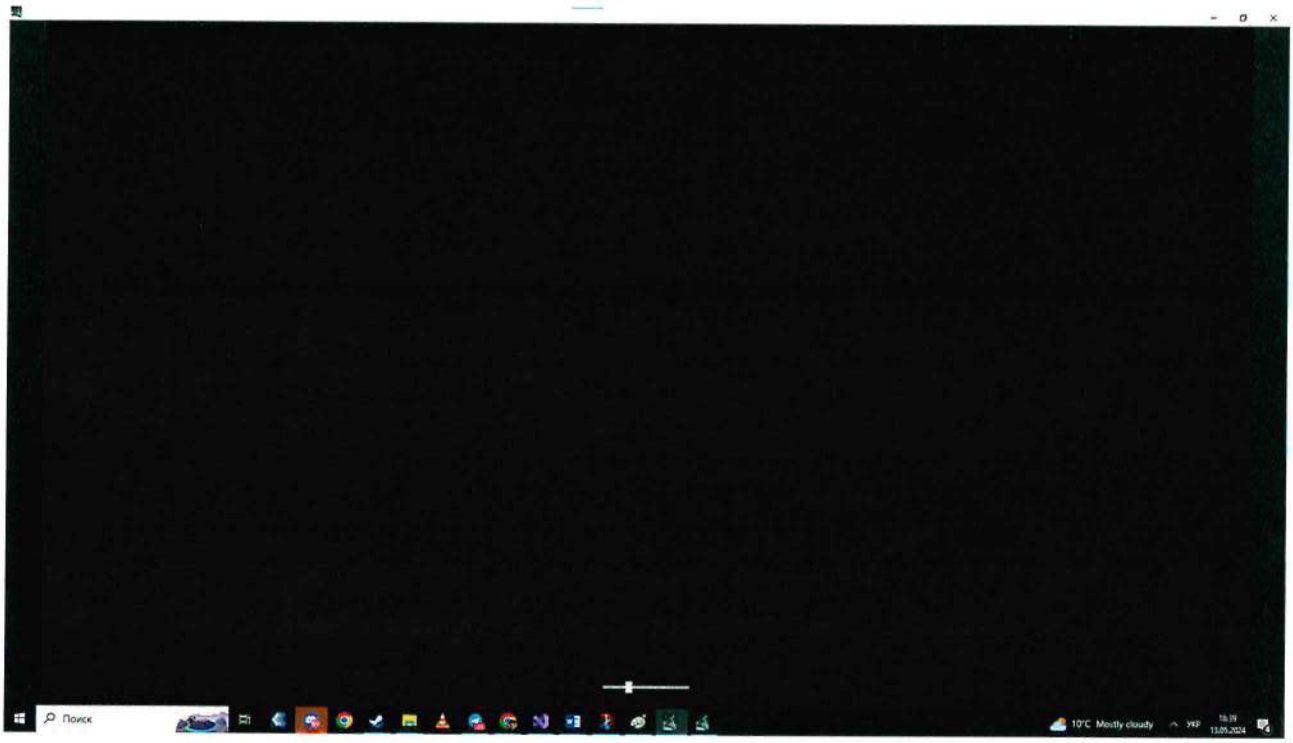

Рис. 3.10 Вікно клієнта

Для того щоб розпочати перегляд треба щоб хост натиснув на паузу/плей , так як відео при відкритті стоїть на паузі , коли це відбудеться запуститься відео , приклад (Рис. 3.11)

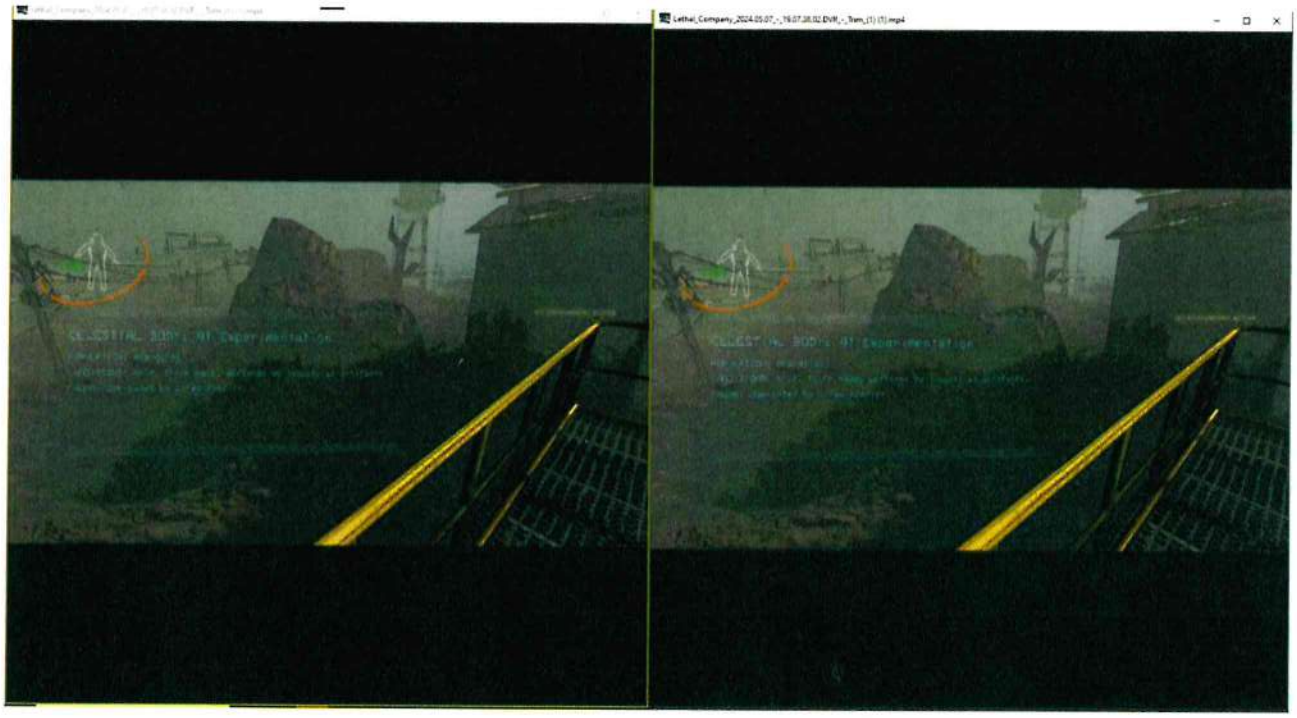

Рис. 3.11 Вікно хоста та клієнта

Якщо ви почали перегляд та дозволили новим клієнтам під'єднуватися , при новому підключенні треба буде натиснути паузу і потім F5 для того щоб синхронізувати відео.

Висновок

Під час написання дипломної роботи було отримано цінний досвід, який допоміг покращити свої навички у програмуванні на мові C# та в розробці графічного інтерфейсу з використанням мови розмітки XAML , також отримав цінний досвід в мережевому програмуванні. Це дозволило більш ефективно виконувати завдання, пов'язані з розробкою програмного забезпечення.

Крім того, було освоєно навички управління часом та організації роботи для досягнення поставлених цілей у визначені терміни. Це допомогло краще розподіляти свій час між різними завданнями та пріоритетами, забезпечуючи результативну роботу і вчасне виконання проектів.

Загалом, дипломна робота стала важливим кроком у моєму професійному розвитку, знання та навички що були набуті в ході практики будуть використані в майбутніх проектах та задачах.

Перелік посилань

1. CLR via C# (4th Edition) (Developer Reference) / Jeffrey Richter США, 2012 р.

2. Professional WPF and C# Programming: Practical Software Development Using WPF and C# / Alex Khang США, 2019 р.

3. Universal Windows Apps with XAML and C# Unleashed / Adam Nathan США, 2015 р.

4. WPF Documentation // Режим доступу: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-8.0

5. XAML Documentation //Режим доступу: https://learn.microsoft.com/ru-ru/visualstudio/xaml-tools/?view=vs-2022

6. C# Documentation //Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/

7. .Net Framework Documentation// Режим доступу: https://learn.microsoft.com/en-us/dotnet/framework/

8. TCP Documentation // Режим доступу : https://datatracker.ietf.org/doc/html/rfc9293

9. Git Documentation //Режим доступу : https://git-scm.com/doc

10. XamlAnimatedGif // Режим доступу : https://github.com/XamlAnimatedGif/XamlAnimatedGif

Додатки

Додаток А

Посилання на Github

https://github.com/Gigakaban/LocalVideoSharingApp

Код мовою C#

Код головного вікна

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Windows;
using Microsoft.Win32;

namespace Video_App
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public event EventHandler<List<string>> ClientIPsUpdated;
        public event EventHandler<string> Name;
        private List<string> clientIPs = new List<string>();
        private TcpListener server;
        private TcpClient client;
        private NetworkStream stream;
        private bool isServer = false;
        private bool off = true;
        public MainWindow()
        {
            InitializeComponent();
            Closing += MainWindow_Closing;
            string hostName = Dns.GetHostName();
            IPAddress[] addresses = Dns.GetHostAddresses(hostName);

            foreach(var item in addresses)
            {
                if(item.AddressFamily ==
System.Net.Sockets.AddressFamily.InterNetwork)
                {
                    cboxip.Items.Add(item);
                }
            }


        }
        public void videoWindowClosed1()
        {
            mwindow.Visibility = Visibility.Visible;
        }

        private void MainWindow_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
```

```
{
    if(File.Exists(Path.Combine(Path.GetTempPath(), "tempvideo.mp4")))
    {
        File.Delete(Path.Combine(Path.GetTempPath(), "tempvideo.mp4"));
    }
    Process[] processes = Process.GetProcesses();
    foreach(var item in processes)
    {
        Console.WriteLine(item.ProcessName);
        if(item.ProcessName== "Video App")
        {
           item.Kill();
        }
    }

}

private void ConnectButton_CLick(object sender, RoutedEventArgs e)
{
    if(IPAddress.TryParse(ServerIpTextBox.Text, out IPAddress a))
    {
        string serverIp = ServerIpTextBox.Text;
        int serverPort;

        if (int.TryParse(ServerPortTextBox.Text, out serverPort))
        {
            if(serverPort>0&&serverPort<65534)
            {

                Thread clientThread = new Thread(() =>
                {
                    isServer = false;
                    ConnectToServer(serverIp, serverPort);
                });
                clientThread.Start();
            }
            else
            {
                MessageBox.Show("Введіть коректний номер порту.");
            }

        }
        else
        {
            MessageBox.Show("Введіть номер порту (цифри).");
        }
    }
    else
    {
        MessageBox.Show("Введіть ipv4 правильно.");
    }
}

private void HostButton_Click(object sender, RoutedEventArgs e)
{
    if (cboxip.SelectedItem==null)
    {
        MessageBox.Show("Виберіть ip в випадаючому списці");
    }
    else
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Video Files | *.mp4; *.avi; *.mkv";
        if (openFileDialog.ShowDialog() == true)
```

```
                {
                    string videoFilePath = openFileDialog.FileName;

                    string[] tstr = videoFilePath.Split('\\');

                    Thread serverThread = new Thread(() =>
                    {
                        isServer = true;
                        StartServer(videoFilePath, tstr[tstr.Length - 1]);

                    });

                    serverThread.Start();
                    lgif.Visibility = Visibility.Visible;
                    lstatus.Visibility = Visibility.Visible;
                }
            }

        }

        private async void StartServer(string videoFilePath, string name)
        {


            try
            {
                int port = 12345;
                string cbox = "";
                await Dispatcher.InvokeAsync(() =>
                {
                    cbox = cboxip.SelectedItem.ToString();
                }, System.Windows.Threading.DispatcherPriority.Normal);
                IPAddress myIp = IPAddress.Parse(cbox);
                await Dispatcher.InvokeAsync(() =>
                {
                Clipboard.SetText($"{myIp} {port}");
                }, System.Windows.Threading.DispatcherPriority.Normal);



                server = new TcpListener(myIp, port);
                server.Start();
                MessageBox.Show("Сервер стартований на IP-адресі: " + myIp.ToString()
+ ", порт: " + port.ToString()+" Данні скопійовані в буфер обміну");

                bool isFirstClient = true;

                while (off)
                {

                    client = await server.AcceptTcpClientAsync();
                    await Dispatcher.InvokeAsync(() =>
                    {
                        lstatus.Content = "Передача данних...";
                    }, System.Windows.Threading.DispatcherPriority.Normal);


                    stream = client.GetStream();

UpdateClientIPs(((IPEndPoint)client.Client.RemoteEndPoint).Address.ToString());

                    using (FileStream fileStream = File.OpenRead(videoFilePath))
                    {
```

```
                    byte[] buffer = new byte[4096];
                    int bytesRead;

                    while ((bytesRead = await fileStream.ReadAsync(buffer, 0,
buffer.Length)) > 0)
                    {
                        await stream.WriteAsync(buffer, 0, bytesRead);
                    }
                }


                if (isFirstClient)
                {
                    isFirstClient = false;
                    await Dispatcher.InvokeAsync(() =>
                    {
                        Uri videoUri = new Uri(videoFilePath);
                        video vid = new video(videoUri,clientIPs,name);
                        vid.Show();
                        mwindow.Visibility = Visibility.Hidden;
                    }, System.Windows.Threading.DispatcherPriority.Normal);
                }
                await Dispatcher.InvokeAsync(() =>
                {
                    Name?.Invoke(this, name);
                }, System.Windows.Threading.DispatcherPriority.Normal);


                stream.Close();
                client.Close();
                MessageBox.Show("Відео успішно надіслано клієнту.");


                MessageBoxResult result = MessageBox.Show("Продовжити приймати
нових клієнтів?", "Підключення нових клієнтів", MessageBoxButton.YesNo);
                if (result == MessageBoxResult.No)
                {
                    break;
                }

            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка сервера: " + ex.Message);
        }
        finally
        {

            if (stream != null)
                stream.Close();
            if (client != null)
                client.Close();
            if (server != null)
                server.Stop();
        }
    }

    private void UpdateClientIPs(string ipAddress)
    {
        clientIPs.Add(ipAddress);
        ClientIPsUpdated?.Invoke(this, clientIPs);

    }
```

```
        private async void ConnectToServer(string serverIp, int serverPort)
        {
            try
            {
                client = new TcpClient();
                await client.ConnectAsync(serverIp, serverPort);
                stream = client.GetStream();
                Application.Current.Dispatcher.Invoke(() =>
                {

                    lgif.Visibility = Visibility.Visible;
                    lstatus.Content = "Триває отримання данних...";
                    lstatus.Visibility = Visibility.Visible;
                });


                using (MemoryStream memoryStream = new MemoryStream())
                {
                    byte[] buffer = new byte[4096];
                    int bytesRead;
                    while ((bytesRead = await stream.ReadAsync(buffer, 0,
buffer.Length)) > 0)
                    {
                        memoryStream.Write(buffer, 0, bytesRead);
                    }

                    byte[] receivedData = memoryStream.ToArray();

                    Application.Current.Dispatcher.Invoke(() =>
                    {

                        videoclient videoclient= new
videoclient(Convert.ToBase64String(receivedData));
                        videoclient.Show();

                        mwindow.Visibility = Visibility.Hidden;
                    });
                }

            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка підключення: " + ex.Message);
            }
            finally
            {
                stream.Close();
                client.Close();
            }

        }


    }
}
```

Код вікна відео хоста

```
using System;
using System.Collections.Generic;
using System.Net.Sockets;
using System.Text;
```

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;


namespace Video_App
{
    /// <summary>
    /// Логика взаимодействия для video.xaml
    /// </summary>
    public partial class video : Window
    {
        private List<string> clientIPs = new List<string>();
        public event EventHandler<bool> WindowClosed;
        private string _name;
        public video(Uri str, List<string> ClientIPs, string name)
        {
            InitializeComponent();

            vido.LoadedBehavior = MediaState.Manual;

            _name= name;

            vid.Title= name;

            vido.Volume = 0.5;

            vido.Source = str;

            vido.Pause();

            clientIPs = ClientIPs;

            ((MainWindow)Application.Current.MainWindow).ClientIPsUpdated +=
MainWindow_ClientIPsUpdated;

            ((MainWindow)Application.Current.MainWindow).Name += NameTransport;

            Closing += WindowClosed1;

            this.KeyDown += Window_KeyDown;

        }
        private void NameTransport(object sender, string name)
        {
            try
            {
                foreach (var item in clientIPs)
                {
                    SendName(item, 12346, name);
                }
            }
            catch
            {

            }

        }
        private void Window_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.F)
            {
                OnFullScreen();
            }
```

```
            if (e.Key == Key.Escape)
            {
                OffFullScreen();
            }
            if (e.Key == Key.F5)
            {
                try
                {
                    TimeSpan currentPosition = vido.Position;
                    foreach (var item in clientIPs)
                    {
                        SyncTime(item, 12346,currentPosition);
                    }
                }
                catch { }

            }
        }
        private void SyncTime(string ip, int port, TimeSpan currentPosition)
        {
            try
            {

                using (TcpClient client = new TcpClient(ip, port))
                {

                    using (NetworkStream stream = client.GetStream())
                    {

                        string request = $"time{currentPosition}";
                        byte[] data = Encoding.UTF8.GetBytes(request);
                        stream.Write(data, 0, data.Length);

                        Console.WriteLine("Запит відправлений успішно!");
                    }
                }
            }
            catch (Exception e)
            {

            }

        }
        private void OnFullScreen()
        {
            this.WindowStyle = WindowStyle.None;
        }
        private void OffFullScreen()
        {
            this.WindowStyle = WindowStyle.ToolWindow;
        }
        private void WindowClosed1(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            MainWindow v = new MainWindow();
            v.videoWindowClosed1();
            try
            {
                foreach (var item in clientIPs)
                {
                    SendExit(item, 12346);
                }
            }
            catch { }
```

```
}
private void Button_Click(object sender, RoutedEventArgs e)
{
    if(lessvid.Visibility==Visibility.Hidden)
    {
        lessvid.Visibility = Visibility.Visible;
        pause.Visibility = Visibility.Visible;
        incrvid.Visibility = Visibility.Visible;
        volum.Visibility = Visibility.Visible;
    }
    else
    {
        lessvid.Visibility = Visibility.Hidden;
        pause.Visibility = Visibility.Hidden;
        incrvid.Visibility = Visibility.Hidden;
        volum.Visibility = Visibility.Hidden;
    }
}

private void Minus10second_Click(object sender, RoutedEventArgs e)
{
    TimeSpan currentPosition = vido.Position;
    TimeSpan newPosition = currentPosition.Subtract(TimeSpan.FromSeconds(10));

    if (newPosition >= TimeSpan.Zero)
    {
        vido.Position = newPosition;
    }
    try
    {
        foreach (var item in clientIPs)
        {
            SendMinus10(item, 12346);
        }
    }
    catch
    { }

}
private void Plus10second_Click(object sender, RoutedEventArgs e)
{
    TimeSpan currentPosition = vido.Position;
    TimeSpan newPosition = currentPosition.Add(TimeSpan.FromSeconds(10));

    if (newPosition < vido.NaturalDuration.TimeSpan)
    {
        vido.Position = newPosition;
    }
    try
    {
        foreach (var item in clientIPs)
        {
            SendPlus10(item, 12346);
        }
    }
    catch { }

}
private bool isPaused = false;

private void Pause_Click(object sender, RoutedEventArgs e)
{
    if (isPaused)
```

```
            {
                vido.Play();
                try
                {
                    foreach (var item in clientIPs)
                    {
                        SendPlay(item, 12346);
                        SendName(item, 12346, _name);
                    }
                }
                catch { }

            }
            else
            {
                vido.Pause();
                try
                {
                    foreach (var item in clientIPs)
                    {
                        SendPause(item, 12346);

                    }
                }
                catch { }

            }

            isPaused = !isPaused;
        }
        private void Slider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            double sliderValue = volum.Value;
            vido.Volume=sliderValue;
        }


        private void MainWindow_ClientIPsUpdated(object sender, List<string> e)
        {
            clientIPs = e;

        }

        private void SendPause(string ip,int port)
        {
            try
            {

                using (TcpClient client = new TcpClient(ip, port))
                {
                    using (NetworkStream stream = client.GetStream())
                    {
                        string request = "pause";
                        byte[] data = Encoding.UTF8.GetBytes(request);
                        stream.Write(data, 0, data.Length);

                        Console.WriteLine("Запит відправлений успішно!");
                    }
                }
            }
            catch (Exception e)
            {
```

```
    }

}

private void SendPlay (string ip, int port)
{
    try
    {

        using (TcpClient client = new TcpClient(ip, port))
        {
            using (NetworkStream stream = client.GetStream())
            {
                string request = "play";
                byte[] data = Encoding.UTF8.GetBytes(request);
                stream.Write(data, 0, data.Length);

                Console.WriteLine("Запит відправлений успішно!");
            }
        }
    }
    catch (Exception e)
    {

    }
}

private void SendName(string ip, int port,string name)
{
    try
    {

        using (TcpClient client = new TcpClient(ip, port))
        {
            using (NetworkStream stream = client.GetStream())
            {
                string request = $"name{name}"; // Запрос на паузу
                byte[] data = Encoding.UTF8.GetBytes(request);
                stream.Write(data, 0, data.Length);

                Console.WriteLine("Запит відправлений успішно!");
            }
        }
    }
    catch (Exception e)
    {

    }
}

private void SendPlus10(string ip, int port)
{
    try
    {

        using (TcpClient client = new TcpClient(ip, port))
        {
            using (NetworkStream stream = client.GetStream())
            {
                string request = "+10";
                byte[] data = Encoding.UTF8.GetBytes(request);
                stream.Write(data, 0, data.Length);

                Console.WriteLine("Запит відправлений успішно!");
```

```
                }
            }
        }
        catch (Exception e)
        {

        }
    }

    private void SendMinus10(string ip, int port)
    {
        try
        {
            using (TcpClient client = new TcpClient(ip, port))
            {
                using (NetworkStream stream = client.GetStream())
                {
                    string request = "-10";
                    byte[] data = Encoding.UTF8.GetBytes(request);
                    stream.Write(data, 0, data.Length);

                    Console.WriteLine("Запит відправлений успішно!");
                }
            }
        }
        catch (Exception e)
        {

        }
    }

    private void SendExit(string ip, int port)
    {
        try
        {

            using (TcpClient client = new TcpClient(ip, port))
            {
                using (NetworkStream stream = client.GetStream())
                {
                    string request = "exit";
                    byte[] data = Encoding.UTF8.GetBytes(request);
                    stream.Write(data, 0, data.Length);

                    Console.WriteLine("Запит відправлений успішно!");
                }
            }
        }
        catch (Exception e)
        {

        }
    }
}
}
```

Код вікна клієнта

```
using System;
using System.Text;
using System.Windows;
using System.Windows.Controls;
```

```
using System.IO;
using System.Net.Sockets;
using System.Net;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Windows.Input;


namespace Video_App
{
    /// <summary>
    /// Логика взаимодействия для videoclient.xaml
    /// </summary>
    public partial class videoclient : Window
    {
        static private IPAddress ipAddress = IPAddress.Any;
        static private int port = 12346;
        static private TcpListener listener = new TcpListener(ipAddress, port);

        public videoclient(string str)
        {
            InitializeComponent();
            byte[] bytes = Convert.FromBase64String(str);

            string tempFilePath = Path.Combine(Path.GetTempPath(), "tempvideo.mp4");

            File.WriteAllBytes(tempFilePath, bytes);

            Uri uri = new Uri(tempFilePath);

            vido.LoadedBehavior = MediaState.Manual;

            vido.Pause();

            StartListener();

            Closing += videoclient_Closing;

            this.KeyDown += Window_KeyDown;

            vido.Source = uri;
        }
        private void Window_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.F)
            {
                OnFullScreen();
            }
            if (e.Key == Key.Escape)
            {
                OffFullScreen();
            }
        }
        private void OnFullScreen()
        {
            this.WindowStyle = WindowStyle.None;
        }
        private void OffFullScreen()
        {
            this.WindowStyle = WindowStyle.ToolWindow;
        }
        private void Button_Click(object sender, RoutedEventArgs e)
        {
```

```
            if (volum.Visibility == Visibility.Hidden)
            {
                volum.Visibility = Visibility.Visible;
            }
            else
            {
                volum.Visibility = Visibility.Hidden;
            }
        }
        private void Slider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            double sliderValue = volum.Value;

            vido.Volume = sliderValue;
        }
        private void videoclient_Closing(object sender, CancelEventArgs e)
        {
            listener.Stop();

            MainWindow v = new MainWindow();
            v.videoWindowClosed1();
        }

        async void StartListener()
        {
            try
            {
                listener.Start();
                Console.WriteLine("Очікування підключень...");

                while (true)
                {
                    TcpClient client = await listener.AcceptTcpClientAsync();
                    Console.WriteLine("Підключений клієнт!");

                    Task.Run(() => HandleClient(client));
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Помилка: " + e.Message);
            }
        }
        async void HandleClient(TcpClient client)
        {
            try
            {
                NetworkStream stream = client.GetStream();

                byte[] buffer = new byte[1024];
                int bytesRead;

                while ((bytesRead = await stream.ReadAsync(buffer, 0, buffer.Length))
> 0)
                {
                    string data = Encoding.UTF8.GetString(buffer, 0, bytesRead);
                    Console.WriteLine($"Отримано от клієнта: {data}");

                    if (data == "pause")
                    {
                        Application.Current.Dispatcher.Invoke(() =>
                        {
                            vido.Pause();
```

```
                    });
                }
                if (data.Contains("time"))
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        data = data.Remove(0, 4);

                        TimeSpan position = TimeSpan.Parse(data);
                        vido.Position = position;

                    });
                }
                if (data.Contains("name"))
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        data = data.Remove(0, 4);
                        vid.Title = data;

                    });
                }
                if (data == "play")
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        vido.Play();
                    });
                }
                if (data == "+10")
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        TimeSpan currentPosition = vido.Position;
                        TimeSpan newPosition =
currentPosition.Add(TimeSpan.FromSeconds(10));

                        if (newPosition < vido.NaturalDuration.TimeSpan)
                        {
                            vido.Position = newPosition;
                        }
                    });
                }
                if (data == "-10")
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        TimeSpan currentPosition = vido.Position;
                        TimeSpan newPosition =
currentPosition.Subtract(TimeSpan.FromSeconds(10));

                        if (newPosition >= TimeSpan.Zero)
                        {
                            vido.Position = newPosition;
                        }
                    });
                }
                if (data == "exit")
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        Close();
                    });
                }
```

```
                    byte[] response = Encoding.UTF8.GetBytes("Данні отримані
успішно!");
                    await stream.WriteAsync(response, 0, response.Length);
                }

                client.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("Помилка при обробці клієнта: " + e.Message);
            }
        }

    }
}
```