МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ГРАФІЧНИЙ КЛІЄНТ ОБМІНУ ІНФОРМАЦІЄЮ У МЕРЕЖАХ ТИПУ PEER-TO-PEER ЗА ПРОТОКОЛОМ BITTORRENT

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 401.22010308

Виконав студент 4-го курсу, групи 401
_____ В. В. Карвацький
«18» червня 2024 р.

Керівник: д-р техн. наук, проф.
_____І. М. Журавська
«18» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Карвацькому Віктору В'ячеславовичу.

1. Тема кваліфікаційної роботи «Графічний клієнт обміну інформацією у мережах типу peer-to-peer за протоколом BitTorrent».

Керівник роботи Журавська Ірина Миколаївна, д-р техн. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «18» червня 2024 р.

3. Вхідні (початкові) дані до роботи: специфікація протоколу BitTorrent v1.0.

Очікуваний результат: застосунок для передачі даних у мережах типу peer-to-peer за протоколом BitTorrent з графічним інтерфейсом.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- дослідження специфікації протоколу BitTorrent, принципів його роботи;
- огляд та вибір засобів для розробки клієнта, шаблонів проєктування;
- моделювання та проєктування програмного забезпечення;
- реалізація програмного забезпечення;
- здійснення тестування роботи кінцевого продукту.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Питання охорони праці для ІТ-спеціалістів»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	канд. техн. наук, доцент Алексєєва А. О.	

Керівник роботи __д-р техн. наук, проф. Журавська І. М._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Карвацький В. В._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_14_» _____січня____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Графічний клієнт обміну інформацією у мережах типу peer-to-peer за протоколом BitTorrent

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз специфікації протоколу BitTorrent, огляд існуючих засобів, розробка ПЗ	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	
12	Захист КРБ перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	

Розробив студент Карвацький В. В. _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи д-р техн. наук, проф. Журавська І. М. _____
(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Графічний клієнт обміну інформацією у мережах типу peer-to-peer за протоколом BitTorrent»

Студент: Карвацький Віктор В’ячеславович

Керівник: зав. каф. комп'ютерної інженерії, д-р техн. наук, проф. Журавська І. М.

Актуальність проєкту обумовлена популярністю та ефективністю мереж типу Peer-to-Peer (P2P) для передачі великого обсягу інформації без затрат на серверну інфраструктуру.

Об'єкт роботи – процеси обміну інформацією між комп’ютерними мережами.

Предмет роботи – програмні засоби отримання та вивантаження файлів у децентралізованих мережах.

Метою кваліфікаційної роботи є забезпечення обміну інформацією великих обсягів у мережах типу P2P за протоколом BitTorrent з дотриманням і перевіркою цілісності.

Пояснювальна записка складається зі вступу, трьох розділів та висновків.

У першому розділі розглядаються особливості P2P-мереж та принцип роботи протоколу BitTorrent.

У другому розділі описано процес вибору інструментів та підходів до програмування.

У третьому розділі описано реалізацію логіки застосунку та створення графічного інтерфейсу користувача.

В результаті розроблено застосунок для обміну інформацію в мережах типу P2P за протоколом BitTorrent, що управляється за допомогою графічного інтерфейсу.

Кваліфікаційна робота містить 53 сторінки (без додатків), 40 рисунків, 25 використаних джерел та 1 додаток.

***Ключові слова**: передача файлів, децентралізована мережа, TCP, peer-to-peer, BitTorrent.*

ABSTRACT

of the Bachelor's thesis

"GUI peer-to-peer network file sharing client using the BitTorrent protocol"

Student: Karvatskyi Viktor

Supervisor: Doctor of Technical Sciences, Professor Zhuravska Iryna

The relevance of the project lies in the popularity and efficiency of Peer-to-Peer (P2P) networks for transmitting large volumes of information without the costs associated with server infrastructure.

The object of the work is the process of exchanging information over the computer networks.

The subject of the work is the software for uploading and downloading files in decentralized networks.

The purpose of the work is to ensure the exchange of large volumes of information in peer-to-peer networks using the BitTorrent protocol with integrity checks.

The work consists of an introduction, 3 chapters and conclusions.

The first chapter is dedicated to the characteristics of P2P networks and the BitTorrent protocol.

The second chapter discusses the process of selecting appropriate tools and approaches to programming.

The third chapter is about the implementation of the application's logic and the creation of the graphical user interface.

As a result, a GUI application for exchanging information in P2P networks using the BitTorrent protocol was developed.

The overall scope of the work is 53 pages (without appendices). The thesis contains 40 figures, 25 references and an appendix.

***Keywords**: file transfer, decentralized network, TCP, Peer-to-Peer, BitTorrent.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CLR – Common Language Runtime
GUI – Graphical User Interface
HTTP – Hypertext Transfer Protocol
IDE – Integrated Development Environment
IL – Intermediate Language
IP – Internet Protocol
MVVM – Model–View–ViewModel
P2P – Peer-to-Peer
TCP – Transmission Control Protocol
SHA – Secure Hash Algorithms
URL – Uniform Resource Locator
UTF – Unicode Transformation Format
WPF – Windows Presentation Foundation

ВСТУП

Мережі типу Peer-to-Peer (P2P) є дуже ефективним рішенням для передачі великого обсягу інформації без затрат на серверну інфраструктуру. На противагу клієнт-серверній архітектурі, вони складаються з рівноправних учасників, які замість завантаження з одного визначеного місця, діляться частинами файлів між собою. В результаті, це надає великі можливості для масштабування і мережа може підтримувати будь-яку кількість учасників без внесення в неї змін.

Найпоширенішим протоколом, що використовує мережі типу P2P є BitTorrent. До росту популярності стрімінгових сервісів його частка складала майже третину з усього трафіку в мережі Інтернет [1].

Об’єкт кваліфікаційної роботи: процеси обміну інформацією між комп’ютерними мережами.

Предмет кваліфікаційної роботи: програмні засоби отримання та вивантаження файлів у децентралізованих мережах.

Мета кваліфікаційної роботи: забезпечення обміну інформацією великих обсягів у мережах типу P2P за протоколом BitTorrent з дотриманням і перевіркою цілісності.

Для досягнення визначеної мети необхідно вирішити наступні **завдання**:

– аналіз структури формату файлу .torrent (Bencode);
– взаємодія з трекером та пірами;
– алгоритм обміну даними;
– перевірка цілісності отриманих даних;
– огляд та вибір засобів для розробки клієнта;
– реалізація застосунку.

Сфера застосування: графічний клієнт обміну інформацією у мережах типу P2P за протоколом BitTorrent можна використовувати в усіх сферах життя та у різних галузях, оскільки він дозволяє обмінюватись файлами будь-якого типу та розміру між пристроями, що підтримують з’єднання з мережею Інтернет.

1 АНАЛІЗ ПРОЦЕСУ ОБМІНУ ДАНИМИ ТА КЛІЄНТІВ ЗА ПРОТОКОЛОМ BITTORRENT

1.1 Принцип роботи та переваги протоколу BitTorrent над іншими способами передачі даних

BitTorrent – це мережевий протокол передачі даних у мережах типу P2P, створений американським програмістом Бремом Коеном у 2001 році.

Його призначення – запобігання великого навантаження на сервер і мережу при розповсюдженні великих файлів. Замість того, щоб завантажувати файл з одного сервера, протокол BitTorrent дозволяє користувачам приєднуватися до мережі хостів для обміну між собою даними (рис. 1.1).

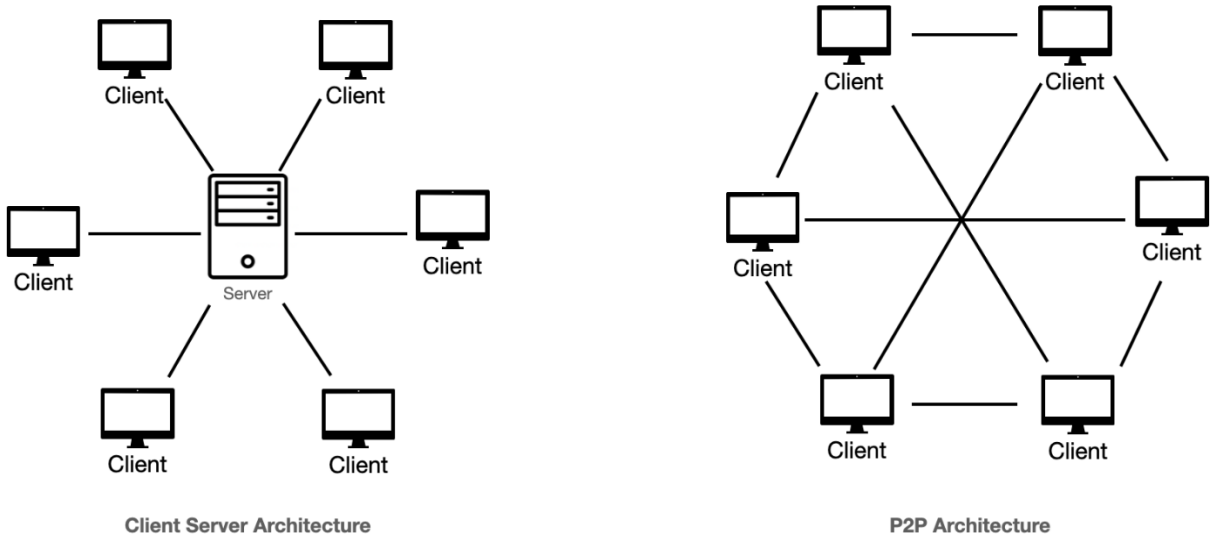


Рисунок 1.1 – Порівняльна схема клієнт-серверної архітектури та мережі типу P2P

Протокол є альтернативою старішому методу розповсюдження даних з одним джерелом і декількома дзеркальними джерелами і може ефективно працювати в мережах з низькою пропускною здатністю [18; 19]. Використовуючи протокол BitTorrent, кілька звичайних комп'ютерів можуть замінити великі сервери, ефективно роздаючи файли багатьом одержувачам.

Загальні особливості протоколу BitTorrent та переваги над іншими способами передачі даних:

– **децентралізація**, відсутність єдиної точки для отримання даних [23];

– рівномірний **розподіл навантаження** між учасниками мережі [20];

– файли діляться на **фрагменти**, для передачі іншим не потрібно мати повністю завантажений файл [24];

– гарантія **цілісності** кожного фрагмента перевіркою контрольних сум [9];

– можливість завантажувати **кілька файлів** одночасно та обирати тільки необхідні з них [22].

Для досягнення поставлених цілей, всі файли для передачі розбиваються на фрагменти (pieces) строго однакового розміру, зазвичай від 256 кбайт до 1 Мбайт [17]. В результаті, один фрагмент може містити частини різних файлів (рис. 1.2).

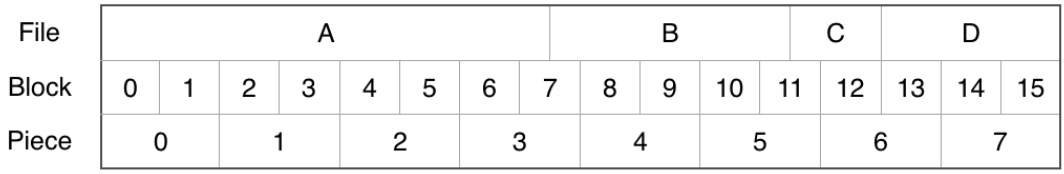


Рисунок 1.2 – Поділ файлів на фрагменти та блоки

Крім цього, фрагменти можуть поділятись на блоки, якщо розмір фрагменту є занадто великим для передачі за один раз. Розмір блоку встановлюється клієнтом.

Кожен фрагмент супроводжується хешем SHA-1 (або SHA-256 в новому стандарті v2), що дозволяє одразу перевірити його цілісність після отримання від іншого учасника мережі.

Для початку роботи з мережею, першим кроком є завантаження файлу з розширенням *.torrent*, що містить усю необхідну інформацію: URL-адресу трекеру та дані про всі файли і фрагменти, на які вони діляться [21].

Трекер (tracker) – це сервер, що контролює усіх активних учасників мережі (рис. 1.3). При зверненні до нього, він повертає інформацію про всіх доступних пірів: IP-адресу та порт кожного з них.

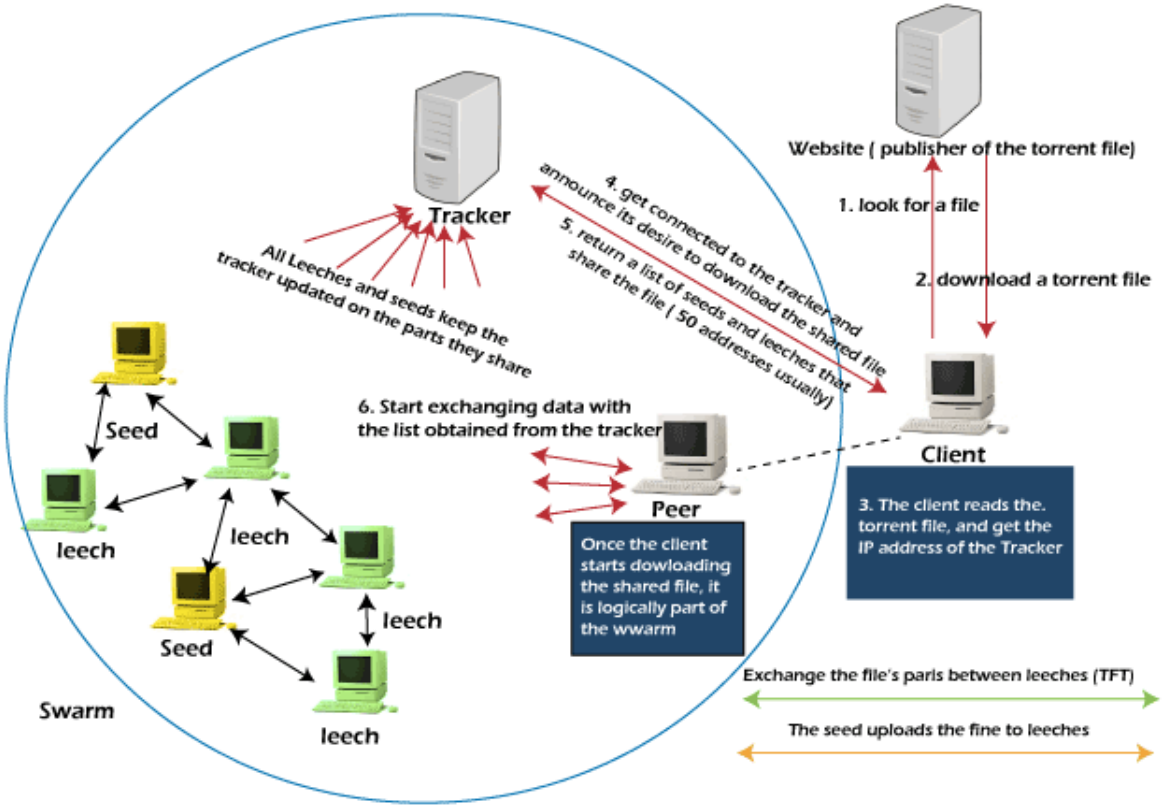


Рисунок 1.3 – Взаємодія між трекером, клієнтами та пірами [11]

Взаємодія з кожним піром відбувається за протоколом TCP [11]. Всі учасники мережі підтримують певні типи повідомлень (message) для обміну фрагментами між собою. Пір, що успішно завантажив усі файли, називається **сідом (seed)**, а той, хто ще в процесі завантаження – **ліч (leech)**. Особливістю BitTorrent, що забезпечує велику швидкість та ефективність, є те, ще необов'язково бути сідом, щоб ділитися даними з іншими. Лічі також можуть відправляти фрагменти іншим одразу після того, як вони у них з'явились. Це робить усіх учасників корисними, навіть якщо вони від'єднуються від мережі після завантаження файлу.

1.2 Аналіз відомих реалізацій P2P-застосунків на основі протоколу BitTorrent

Для передачі файлів за протоколом BitTorrent необхідний спеціальний застосунок з його підтримкою. Найпоширенішим клієнтом є **µTorrent** з ринковою долею 68,6 % [15]. Він має ряд недоліків, таких як вбудована реклама та проблеми

з надмірним використанням процесора та пам’яті, що досить негативно впливає на користувацький досвід та продуктивність (рис. 1.4).

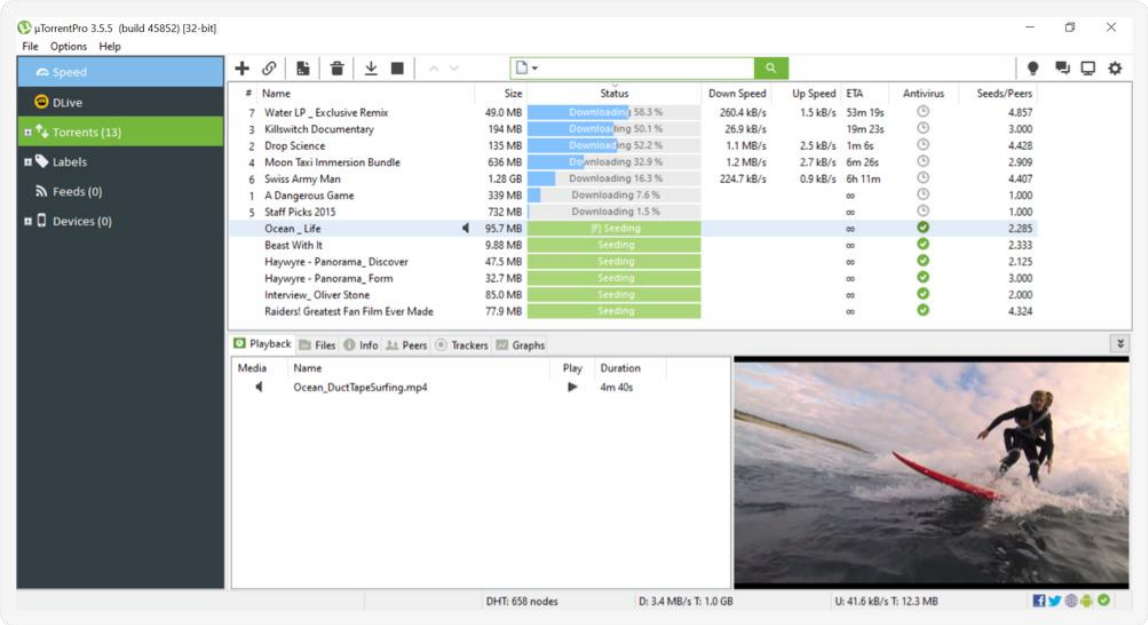


Рисунок 1.4 – Користувацький інтерфейс BitTorrent-клієнта µTorrent

Також є сумнозвісним випадок, коли клієнт є µTorrent поширювався з вбудованим майнером криптовалюти [16].

Другим після нього є офіційний клієнт **BitTorrent**, що був написаний самим Бремом Коеном. Він має ті самі проблеми з рекламою, що і µTorrent (рис. 1.5). Деякі його корисні функції, такі як відсутність реклами, швидша швидкість завантаження та підтримка антивірусу, доступні лише у платній версії *BitTorrent Pro*, що може бути недоліком для користувачів, які не бажають платити за додаткові можливості.

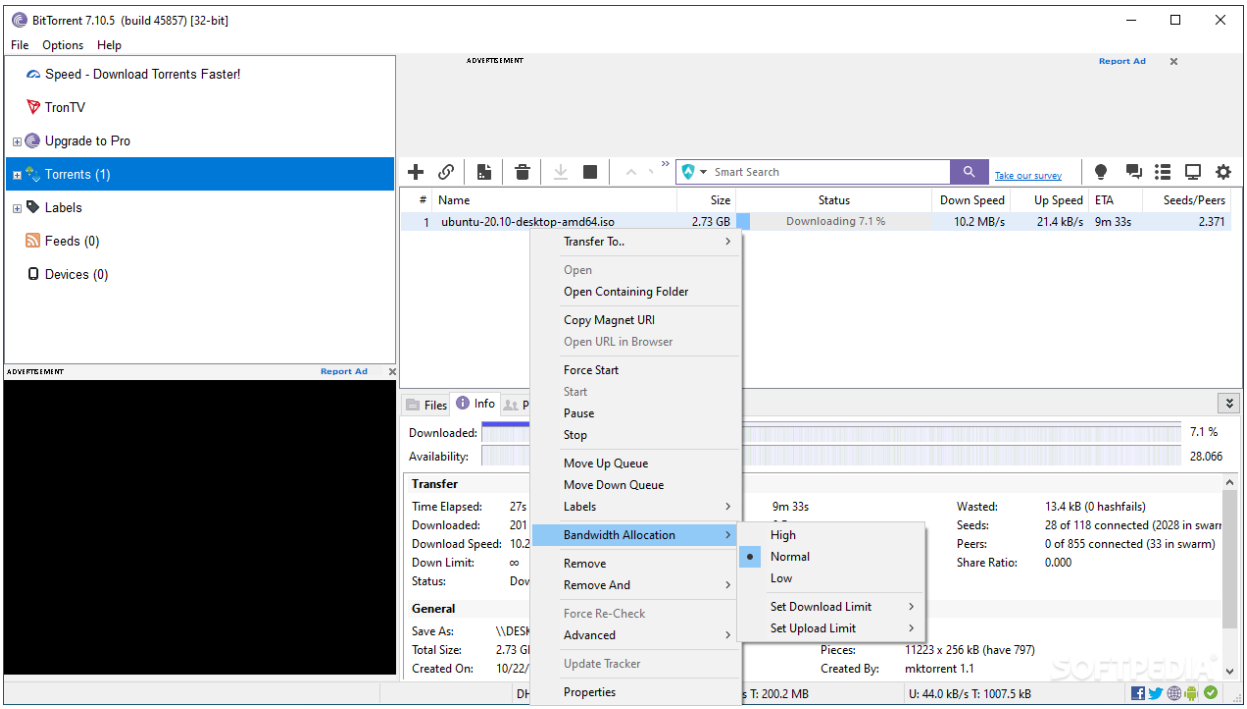


Рисунок 1.5 – Користувацький інтерфейс BitTorrent-клієнта BitTorrent Classic

Ще одним популярним клієнтом є **MediaGet**. Під час встановлення MediaGet часто пропонує додаткові програми або зміни в налаштуваннях браузера, що може бути небажаним (рис. 1.6).

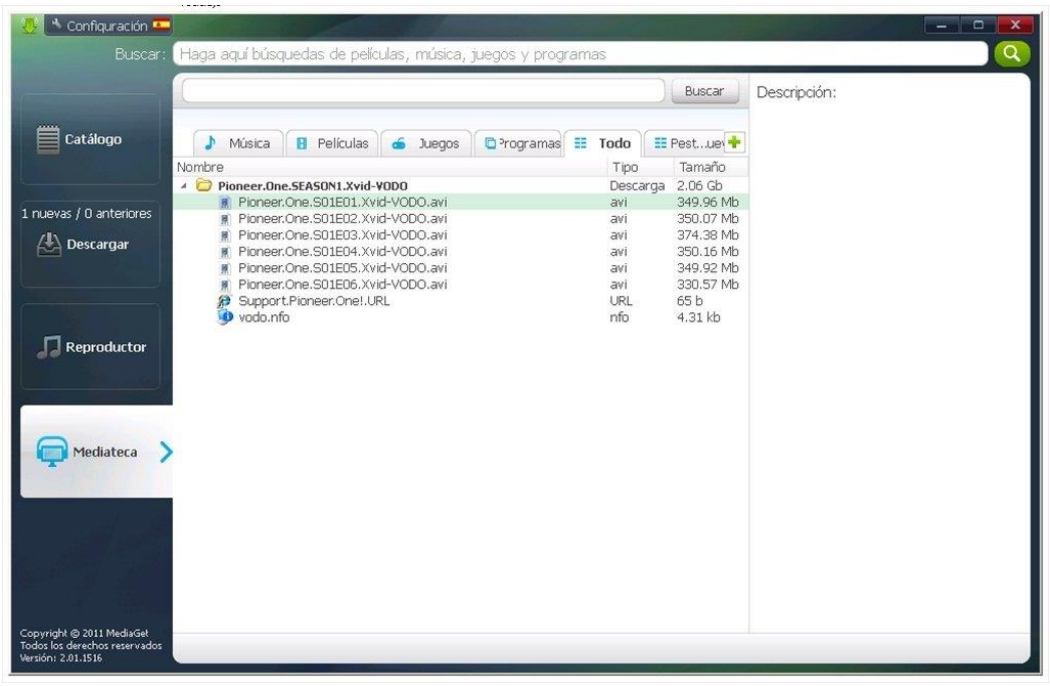


Рисунок 1.6 – Користувацький інтерфейс BitTorrent-клієнта MediaGet

Крім цього, оскільки усі ці варіанти орієнтовані на простоту та зручність для звичайного користувача, вони мають обмежену кількість інструментів для відлагодження. Додавання різних інструментів для розробників може допомогти тим, хто хотів би ознайомитись із принципами роботи протоколу BitTorrent.

Отже, BitTorrent є гнучким та ефективним рішенням для передачі даних мережею Інтернет і має низку переваг над традиційними методами. Найбільш розповсюджені BitTorrent-клієнти мають свої недоліки, що можуть бути виправлені створенням власного рішення.

1.3 Структура .torrent-файлу у форматі Bencode

Torrent-файл складається з метаданих про файли та папки для передачі.

За стандартом BEP-0003 [2], він є словником, що закодований у форматі *Bencode* і містить наступні ключі:

– *announce* – URL-адреса трекеру;

– *encoding* – кодування, що використовується (зазвичай UTF-8);

– *created by* – назва програми, за допомогою якої було створено файл;

– *creation date* – дата створення файлу у форматі Unix timestamp (кількість секунд, що пройшли з 1 січня 1970 року);

– *comment* – будь-який коментарій від автору файлу;

– *info* – словник, що містить інформацію про файли;

Словник *info* складається з таких даних:

– *length* – розмір файлу в байтах;

– *name* – назва файлу;

– *piece length* – розмір одного фрагменту файлу в байтах. Зазвичай це 256 кілобайт (262’144 байт);

– *pieces* – список з SHA-1 хешем кожного фрагменту.

Якщо є більше одного файлу, до цього додається список *files*, що містить словники з ключами *length* (розмір файлу в байтах, як зазначено вище) та *path* (відносний шлях до файлу).

Отже, отримані з файлу метаданих дані допоможуть встановити з'єднання з трекером, щоб дізнатись про інших пірів і валідувати усі фрагменти в майбутньому.

Torrent-файли використовують формат **Bencode**, що дозволяє кодувати різні типи даних у компактному вигляді [12].

Формат підтримує наступні значення: числа, байтові рядки, списки та словники. Використовує символи ASCII для службових символів.

Числа (integer) починаються з літери «i» та кодуються у десятичному форматі. Як і всі інші значення (крім рядків) закінчуються символом «e».

Приклади чисел:

– нуль – «i0e»;
– тринадцять – «i13e»;
– мінус двадцять шість – «i-26e».

Байтовий рядок (byte string) – це послідовність будь-яких байтів. В основному використовується для звичайних рядків, що складаються з символів ASCII. Кодується у вигляді «довжина:зміст». Наприклад, строка «example» кодується як «7:example».

Список (list) починається з літери «l». Не має розділювачів між елементами. Наприкад, список з числа 42 і рядку «test» виглядає як: «l i42e 4:test e» (пробіли додані для зручності читання, вони не підтримуються форматом).

Словник (dictionary) починається з літери «d» і складається з пар «ключ-значення». Аналогічно зі списком, не має розділювачів між елементами та парами. Кожен ключ має бути байтовим рядком та відсортованим за алфавітом. Наприклад, словник, що складається з двух елементів – «foo» зі значенням рядку «test» та «bar» зі значенням числа 42 виглядає наступним чином: «d 3:bar i:42:e 3:foo 4:test e».

Елементами списку, а також ключі та значення словників можуть бути будь-яких типів. Дозволяються вкладені списки та словники, що забезпечує можливість створення складних структур даних.

Отже, розуміючи структуру Bencode, можна розшифрувати дані з файлу метаданих та відповіді від трекеру.

1.4 Отримання даних з трекеру та комунікація з пірами

Трекер – це звичайний HTTP-сервер, що містить лише одну кінцеву точку – *announce*. Наприклад, адреса трекеру відомого Linux-дистрибутиву Debian: http://bttracker.debian.org:6969/announce. Для того, щоб сповістити трекер про себе і отримати адреси інших учасників мережі, потрібно виконати GET-запит на цю адресу з наступними параметрами:

– *info_hash* – SHA-1 хеш секції *info* з торрент-файлу;
– *peer_id* – ідентифікатор піру, рядок з 20 байтів;
– *port* – порт для вхідних з’єднань від інших пірів;
– *uploaded* – кількість байтів, що були відправлені іншим;
– *downloaded* – кількість байтів, що були отримані;
– *left* – кількість байтів, що залишилось отримати до кінця;
– *event* – теперішній стан клієнта. Допустимі значення: «started», «paused» та «stopped». При останніх двух значеннях трекер видаляє клієнта зі свого списку пірів;
– *compact* – булеве значення, що вказує на компактний режим списку пірів, зазвичай «1».

Відповідь трекер знову надсилає у форматі **Bencode**. Вона містить наступні дані [13]:

– *complete* – кількість сідів (пірів з повністю завантаженими фалйами);
– *incomplete* – кількість лічів (пірів без повністю завантажених файлів);
– *interval* – частота, з якою клієнт повинен повтрно звертатись до трекеру;
– *min interval* – мінімальний інтервал;

– *peers* – список пірів. Перші 4 байти кожного – адреса IPv4, далі йдуть 2 байти у порядку Big Endian, що означають порт;

Отже, отримані дані можна використати для встановлення з'єдання з іншими пірами. Також, після сповіщення трекеру, інші учасники мережі почнуть самостійно звертатись до клієнта.

В загальному виді алгоритм взаємодії з іншими пірами виглядає наступним чином (рис. 1.7):

1) встановлення TCP-з'єднання з піром за IP-адресою та портом, отриманими від трекеру;

2) виконання handshake;

3) етап обміну повідомленнями.

Отже, першою частиною є **handshake**. Він потрібен для того, щоб переконатись в тому, що у піра дійсно запущений клієнт, що підтримує протокол BitTorrent і він має необхідні фрагменти для завантаження. Handshake складається з наступних частин:

– довжина ідентифікатору протоколу, завжди дорівнює 19 (або 0 × 13 в шістнадцятковому вигляді);

– ідентифікатор протоколу, рядок «BitTorrent protocol»;

– вісім зарезервованих байтів, використовуються для розширень, описаних в стандарті BEP-0010 [3];

– infohash файлу для завантаження;

– власний ідентифікатор клієнта.

Пір, отримавши ці дані, має надіслати відповідь у тому самому форматі.

Успішно закінчивши handshake, можна приступати до основної частини – обміну повідомленнями.

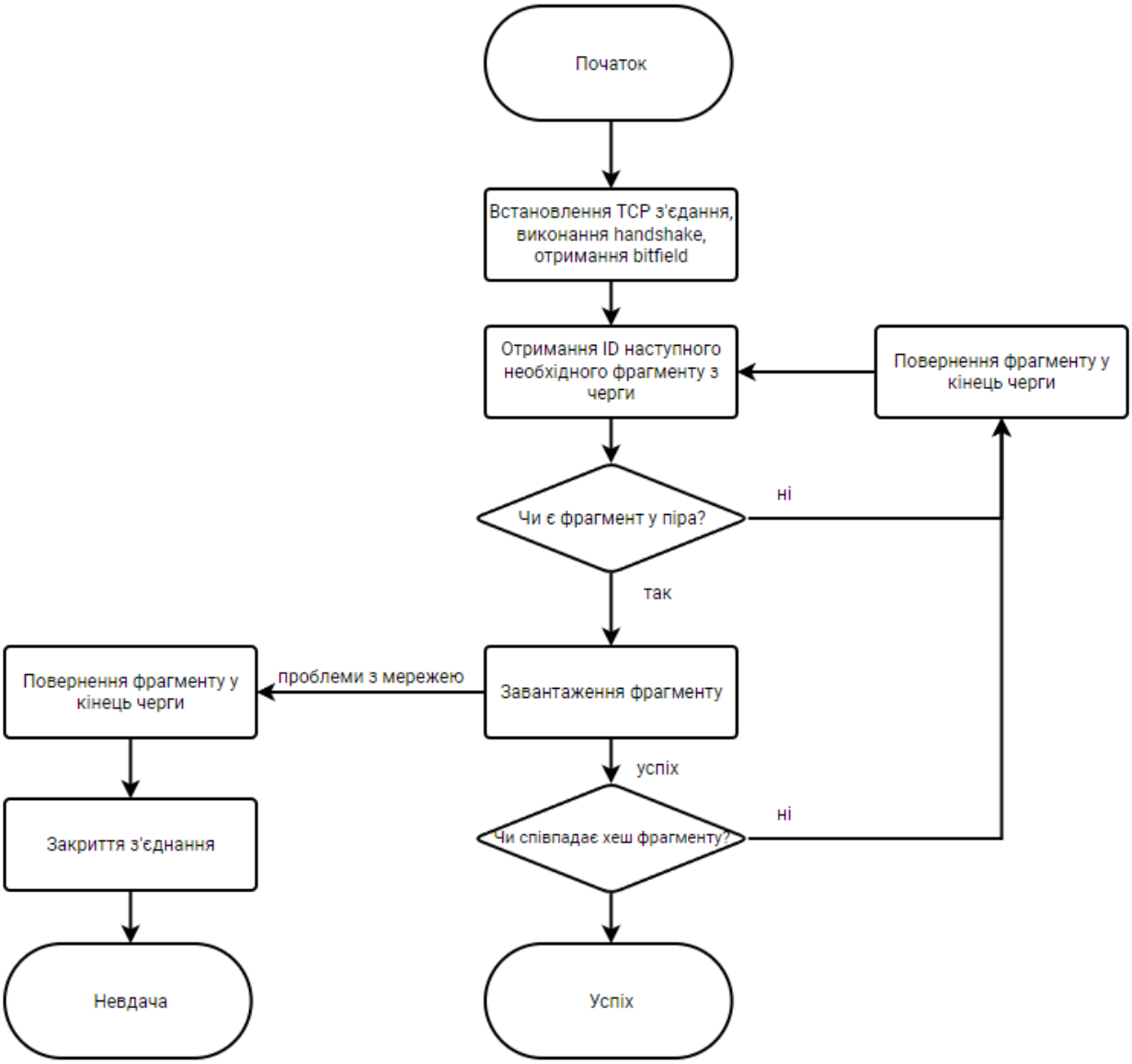


Рисунок 1.7 – Загальна схема комунікації між пірами

Загальна структура повідомлення: його довжина в байтах (32 біти), тип (2 байти) та сам зміст повідомлення. Існують наступні типи повідомлень в BitTorrent [4–6]:

– *choke* – блокування віддачі;
– *unchoke* – розблокування віддачі;
– *interested* – піру потрібний вказаний фрагмент;
– *not interested* – піру не потрібний вказаний фрагмент;
– *have* – у піра є вказаний фрагмент;

– *bitfield* – бітове поле з усіма наявними фрагментами;
– *request* – прохання надіслати вказаний фрагмент;
– *piece* – передача фрагменту;
– *cancel* – відміна передачі.

Для того, щоб позначити, які фрагменти є у піра, використовуються **бітові поля (bitfield),** в яких «1» означає *наявний*, «0» – *ненаявний*. Це дозволяє досягнути максимальної компактності, оскільки фрагментів може бути необмежена кількість.

Для ефективного використання трафіку в BitTorrent існує таке поняття, як **choke** (блокування передачі). За замовченням усі піри є заблокованими при першій взаємодії і розблоковуються після отримання повідомлення *unchoke* [7]. При виборі пірів для розблокування віддається перевага у випадках, коли обидві сторони у цьому зацікавлені – кожна зі сторін має сегменти, яких немає в іншої.

Підсумовуючи, описаний алгоритм може продовжуватись будь-яку кількість часу. Завершивши завантаження файлів, пір становиться сідом і може скільки завгодно залишатись у мережі, щоб допомагати іншим.

Комунікація між учасниками мережі в BitTorrent будується на базі протоколу **TCP (Transmission Control Protocol)**, що є одним з основних протоколів передачі даних на транспортному рівні моделі OSI. Він забезпечує надійний та послідовний обмін даними між двома пристроями через мережу. Це означає, що дані, надіслані одним пристроєм, приходять до іншого пристрою в тому ж порядку, в якому вони були надіслані без помилок передачі даних, таких як втрата пакетів або дублювання.

Для встановлення TCP-з'єднання з іншим пристроєм, необхідно знати його IP-адресу, що ідентифікує його у мережі та номер порту, на якому запущений TCP-сервер.

На відміну від протоколу UDP, для початку передачі даних у TCP необхідно виконати процес встановлення з'єднання за допомогою механізму «three-way handshake» (триходове рукостискання). Його ціль – гарантувати, що обидва пристрої готові до обміну даними. Він складається з трьох частин (рис. 1.8):

1) Клієнт відправляє серверу пакет із запитом на з'єднання (SYN);

2) Сервер підтверджує запит та вказує своє власне бажання встановити з'єднання (SYN-ACK);

3) Клієнт підтверджує прийом підтвердження від сервера (ACK).

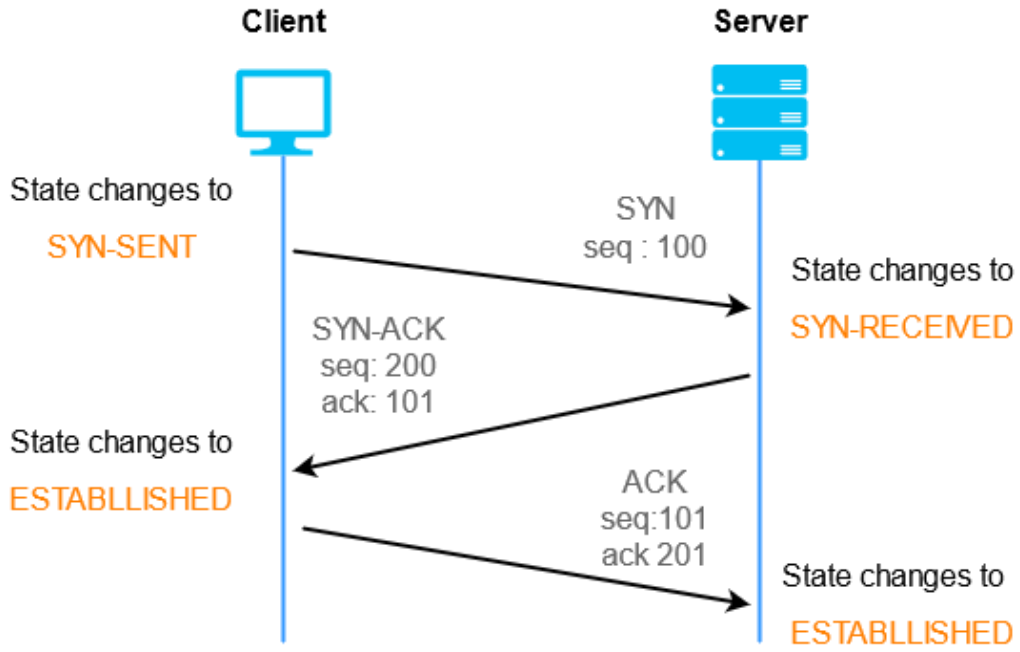


Рисунок 1.8 – Схема встановлення «TCP three-way handshake»

Описані характеристики роблять TCP ідеальним вибором для випадків, де важлива надійність та точність передачі даних, таких як вебсервери, електронна пошта, передача файлів, через що і використовується у BitTorrent [24]. Однак, через додаткове навантаження на керування з'єднанням та перевірку цілісності, TCP може бути менш ефективним за протокол UDP в деяких застосунках, що працюють у режимі реального часу, таких як трансляції або відеоконференції.

Висновки до розділу 1

В першому розділі було розглянуто особливості передачі даних у мережах типу peer-to-peer, їх переваги над традиційною клієнт-серверною архітектурою. Також розглянуто популярні рішення на ринку BitTorrent-клієнтів та недоліки, що

можливо виправити. Опрацьовано специфікацію протоколу BitTorrent, алгоритми комунікації між пірами, взаємодії з трекером для пошуку інших учасників мережі.

Отримані теоретичні відомості про протокол допоможуть ефективно реалізувати свій клієнт будь-якою мовою програмування.

2 ІНСТРУМЕНТИ ТА ПІДХОДИ ДО РОЗРОБКИ. ШАБЛОНИ ПРОЄКТУВАННЯ

2.1 Вибір мови програмування та фреймворків. Мова C# та платформа .NET

Першим етапом підготовки до розробки застосунку є вибір мови програмування та фреймворків. Кожна мова має свої особливості, що можуть як спростити, так і значно уповільнити процес розробки та якість виконання програми. Для написання клієнта обміну інформацією у мережах типу P2P слід звернути увагу на наступні можливості [8–10]:

– криптографічні хеш-функції, такі як SHA-1 та SHA-2;

– робота з мережею, протоколом TCP;

– засоби асинхронного програмування;

– бібліотеки для створення графічного інтерфейсу користувача (англ. Graphical User Interface, GUI).

Заданим критеріям відповідає значна кількість розповсюджених мов програмування, таких як Python, Java, C++ та інші. Для виконання задачі було вирішено використовувати мову C#, оскільки вона поєднує зручний синтаксис, що містить велику кількість синтаксичного цукру, з високою швидкістю виконання.

C# – це сучасна об'єктно-орієнтована мова програмування, що розробляється компанією Microsoft. Вона є основною мовою програмування для платформи .NET, що надає доступ до різноманітних бібліотек та інструментів для розробки застосунків (рис. 2.1).

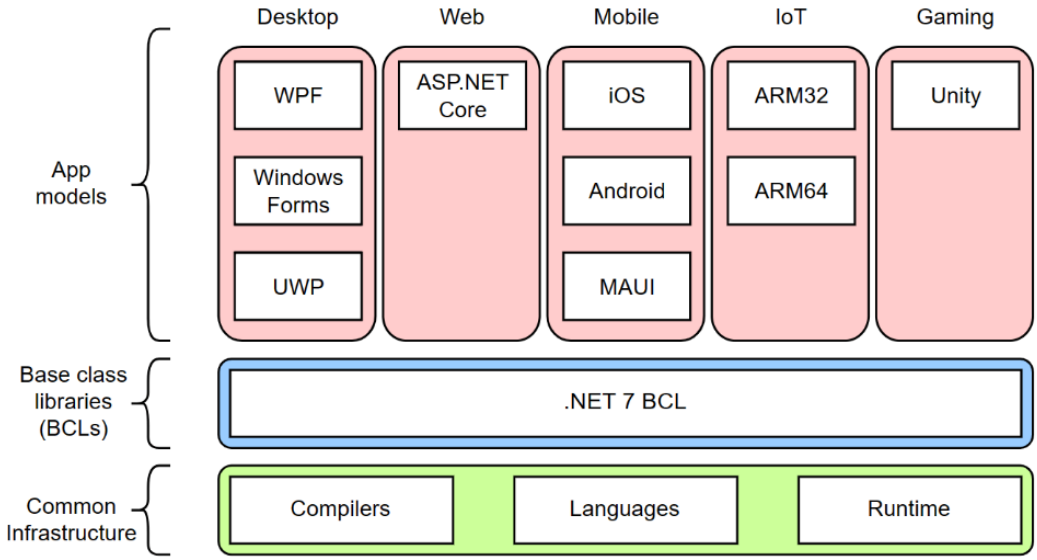


Рисунок 2.1 – Загальна схема компонентів платформи .NET

В основі .NET стоїть віртуальна машина CLR (Common Language Runtime), що виконує проміжний код IL (Intermediate Language). Ці частини є спільними для кожної мови програмування, що підтримується платформою .NET, наприклад C#, F# та Visual Basic. Над ними побудована стандартна бібліотека BCL (Base Class Library), що містить вбудовані типи, колекції, класи для маніпуляції з файловою системою, рядками тощо. Платформа пропонує фреймворки для написання різноманітних видів програм: *Windows Forms, WPF, UWP* для Desktop-застосунків; *ASP.NET Core* для вебсервісів; *Xamarin* для мобільних застосунків.

Код, написаний на C#, є **кросплатформним**, а отже може бути виконаний на різних платформах, оскільки він компілюється в проміжний код IL, який потім інтерпретується або компілюється JIT-компілятором в машинний код під час виконання програми. Це також дозволяє значно покращити швидкість застосунку.

Отже, враховуючи усі перелічені переваги, мова програмування C# та платформа .NET є вдалим вибором для поставленої задачі.

2.2 Технології створення графічного інтерфейсу користувача. Фреймворк WPF

Екосистема .NET пропонує три основні фреймворки для написання Desktop-застосунків: Windows Forms, WPF та UWP.

Windows Forms – технологія, що була створена в 2002 році разом з першою версією .NET Framework. Вона є найстарішою з перелічених, тому має обмежені можливості стилізації та анімації (рис. 2.2). Інтерфейс будується у ній на базі низки стандартних елементів, таких як кнопки, тексти, списки тощо. Для будування логіки вона використовує просту та легку у використанні модель програмування подій.

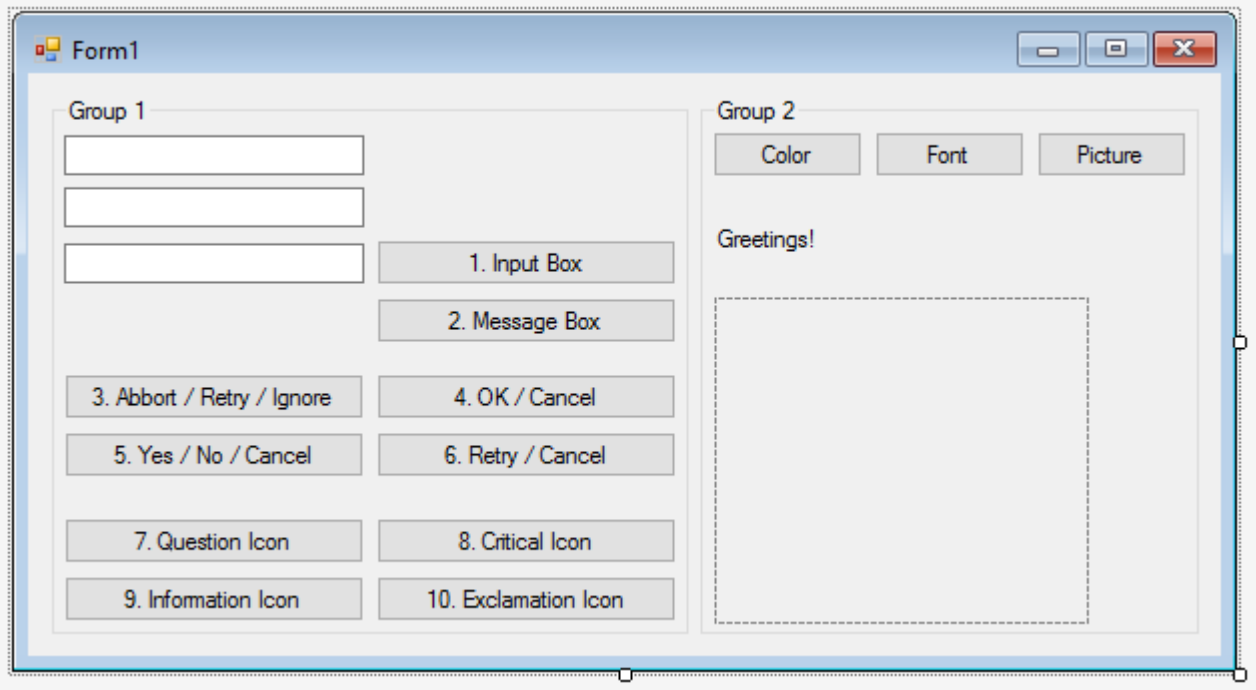


Рисунок 2.2 – Приклад форми, створеної засобами Windows Forms

WPF (Windows Presentation Foundation) – більш сучасна технологія, створена в 2006 році на базі .NET Framework 3.0. Використовує декларативну модель програмування, що дозволяє описувати інтерфейс користувача у файлах розмітки з форматом XAML, що базується на мові XML. Має більш багаті можливості стилізації та анімації, що робить її більш гнучкою для створення

сучасних та зручних інтерфейсів. Крім того, підтримує тривимірну графіку та графічні ефекти.

UWP (Universal Windows Platform) – платформа для створення застосунків для операційної системи Windows 10 (рис. 2.3). У ній використовується декларативна модель програмування, аналогічна WPF, з можливістю розробки інтерфейсу за допомогою XAML. Має вбудовану підтримку адаптивного дизайну та реактивного програмування. Підтримує роботу на різних типах пристроїв, таких як комп'ютери, планшети, смартфони та інші.

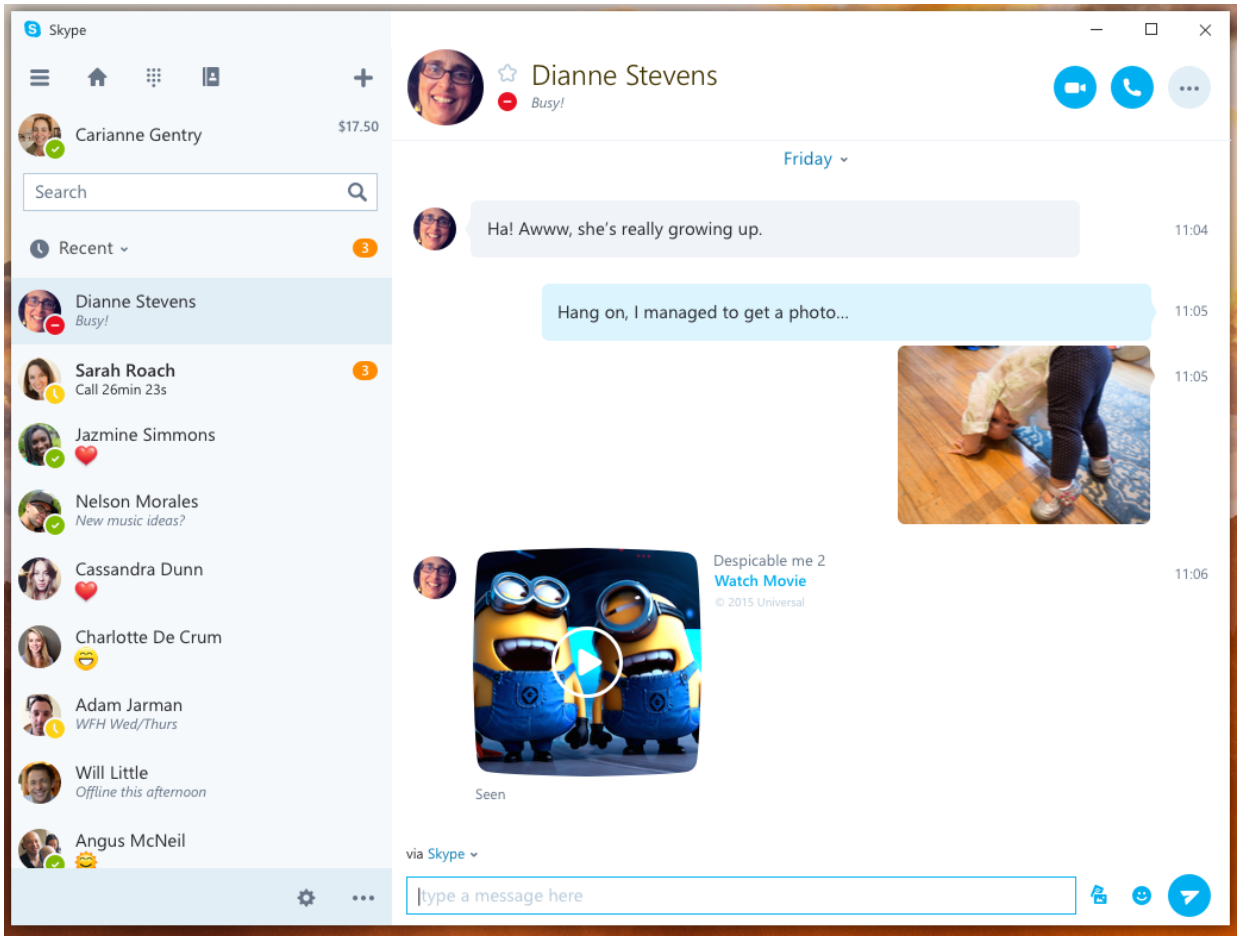


Рисунок 2.3 – Приклад UWP-застосунку – Skype

Незважаючи на те, що UWP є більш новою технологією, вона має низку недоліків, порівняно з іншими:

– більш обмежена підтримка сторонніх бібліотек та елементів управління порівняно з WPF;

– відсутня сумісність зі старішими версіями Windows;

– залежність від Магазину Windows, що є основною платформою для розповсюдження застосунків UWP;

– неповноцінна підтримка деяких функцій (зокрема доступ до системних ресурсів) через орієнтованість на універсальність та підтримку інших пристроїв, такі як планшети, сматфони та інші.

Отже, враховуючи описані переваги та недоліки різних платформ, найбільш вдалим вибором для поставленої задачі є WPF, що забезпечує найкращу підтримку різноманітних UI-бібліотек, елементів управління та версій ОС Windows.

2.3 Шаблон проєктування MVVM

MVVM (Model–View–ViewModel) – шаблон проєктування, в основі якого лежить розділення коду представлення від логіки застосунку. Рекомендується компанією Microsoft для використання у застосунках на платформі WPF.

Складається з трьох частин (рис. 2.4):

1) **Model (модель)** – частина, що містить бізнес-логіку застосунку, дані та операції над ними. Не залежить від інших частин;

2) **View (представлення)** – графічний інтерфейс, з яким взаємодіє користувач. Відображає дані, які надаються ViewModel, і відправляє користувацькі дії (наприклад, кліки мишею або натискання клавіш) ViewModel для обробки;

3) **ViewModel** – посередник між Model та View. Містить дані, що мають відображені на View та логіку, яка виконується при взаємодії з користувачем. ViewModel не повинна містити прямої залежності від WPF-специфічних класів.

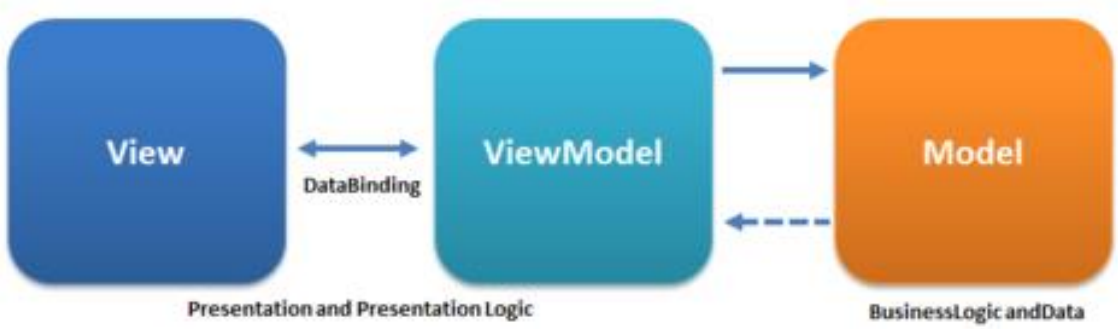


Рисунок 2.4 – Схема шаблону MVVM

Використання MVVM у застосунках на WPF сприяє кращому розділенню відповідальностей коду (Separation of Concerns), полегшенню тестування та можливістю повторного використання коду.

2.4 Середовище розробки. IDE JetBrains Rider

Вибір середовища розробки є важливою частиною підготовки до створення застосунку, оскільки різноманітні інструменти, що вони надають, можуть значно спростити та пришвидшити цей процес.

Компанія Microsoft пропонує два основних інструменти – **Visual Studio та Visual Studio Code** (рис. 2.5). Незважаючи на схожі назви, це два різних продукти, що значно відрізняються своєю функціональністю та призначенням.

Visual Studio – повноцінна інтегрована середа розробки, що надає широкий набір інструментів для розробки програмного забезпечення. Доступна тільки для ОС Windows. Має як і платну версію для професійних розробників, так і безкоштовну для програмістів-любителів.

Visual Studio Code – текстовий редактор, що має велику кількість розширень для гнучкого налаштування під потреби конкретного розробника. Доступний для платформ Windows, MacOS та Linux, але поступається Visual Studio по кількості інструментів для відлагодження коду, що робить його менш зручним варіантом для проєктів високої складності.

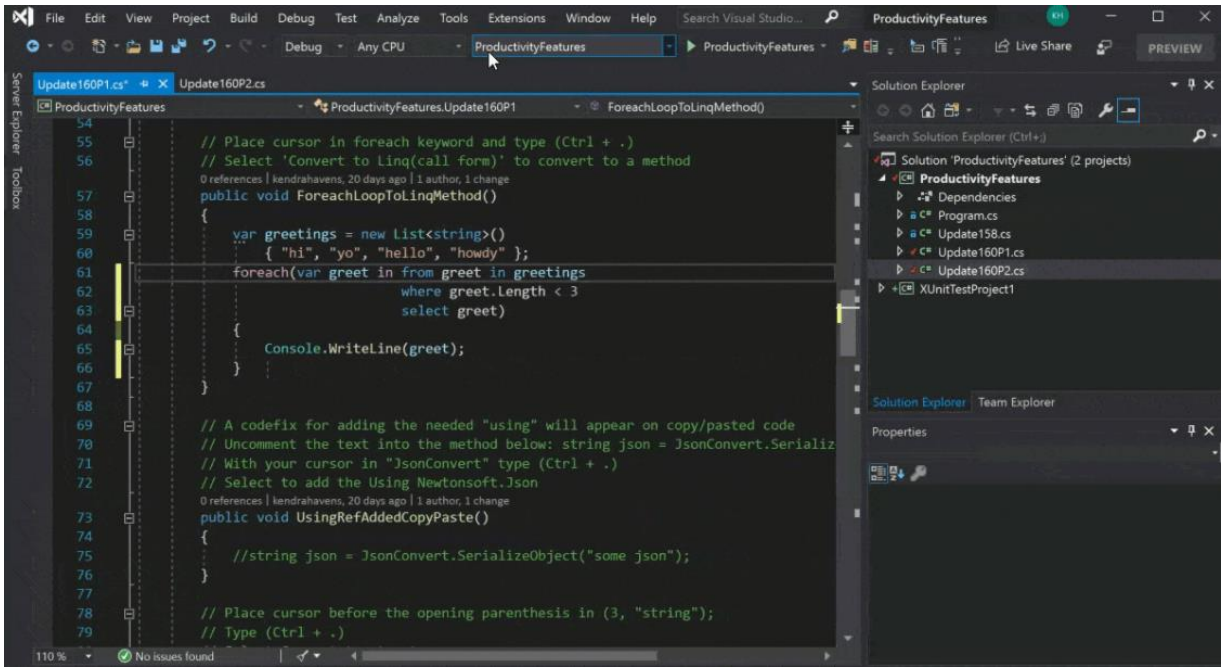


Рисунок 2.5 – Інтерфейс середовища Microsoft Visual Studio

Крім цього, ще одним популярним рішенням є середовище **Rider** від компанії JetBrains. На відміну від Visual Studio, воно є кросплатформним і дозволяє користувачам Linux та MacOS також писати застосунки на платформі .NET, використовуючи повноцінний пакет інструментів для рефакторингу та відлагодження (рис. 2.6).

Основна перевага Rider над іншими варіантами – потужні засоби аналізу коду та перевірки, що допомагають виявляти потенційні проблеми, оптимізувати код та покращувати загальну якість коду.

На жаль, Rider не має безкоштовної версії, але JetBrains надають безкоштовні ліцензії для студентів, чим можна скористатись для створення цього проєкту.

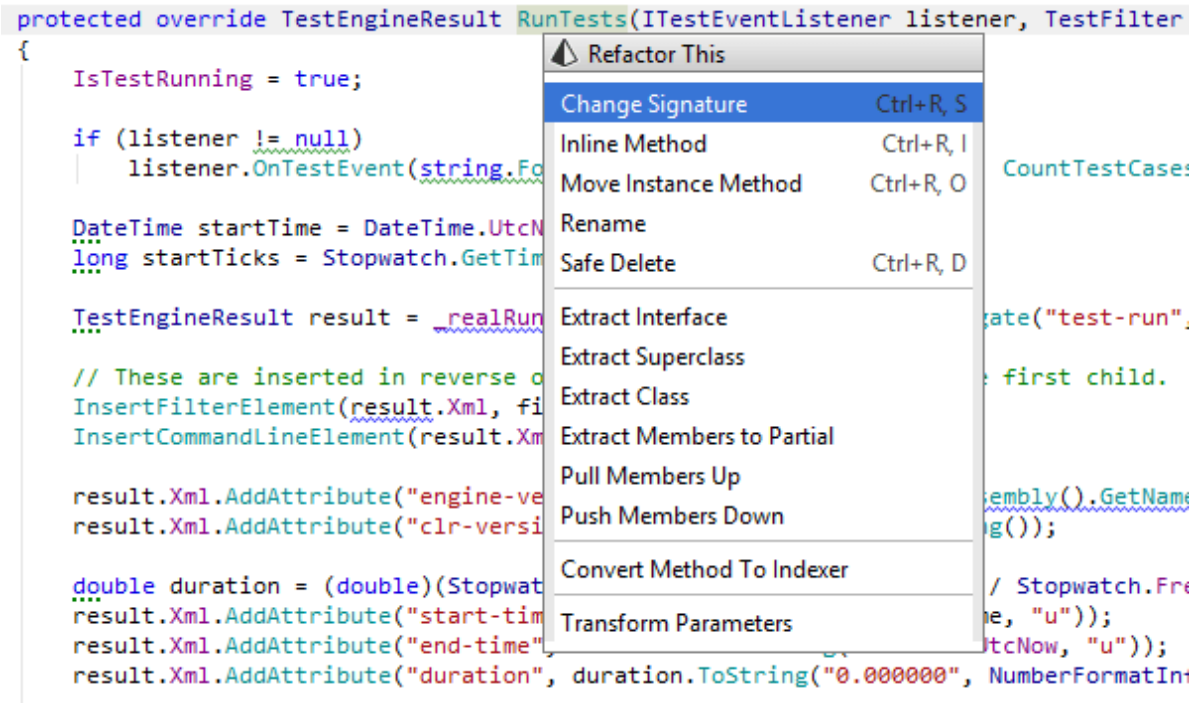


Рисунок 2.6 – Приклад засобів рефакторингу у JetBrains Rider

Підсумовуючи, середовище JetBrains Rider є найкращим варіантом для написання застосунків мовою програмування C# через наявність численних інструментів, що значно підвищують швидкість написання коду та допомагають виявляти потенційні проблеми.

2.5 Асинхронне програмування. Синтаксис async/await

Асинхронне програмування – це підхід до розробки програм, при якому код не блокує потік під час виконання деяких операцій, таких як введення/виведення даних або мережеві запити. Замість того, щоб чекати на завершення цих операцій, програма може продовжувати виконувати інші завдання, поки чекається результат асинхронної операції (рис. 2.7).

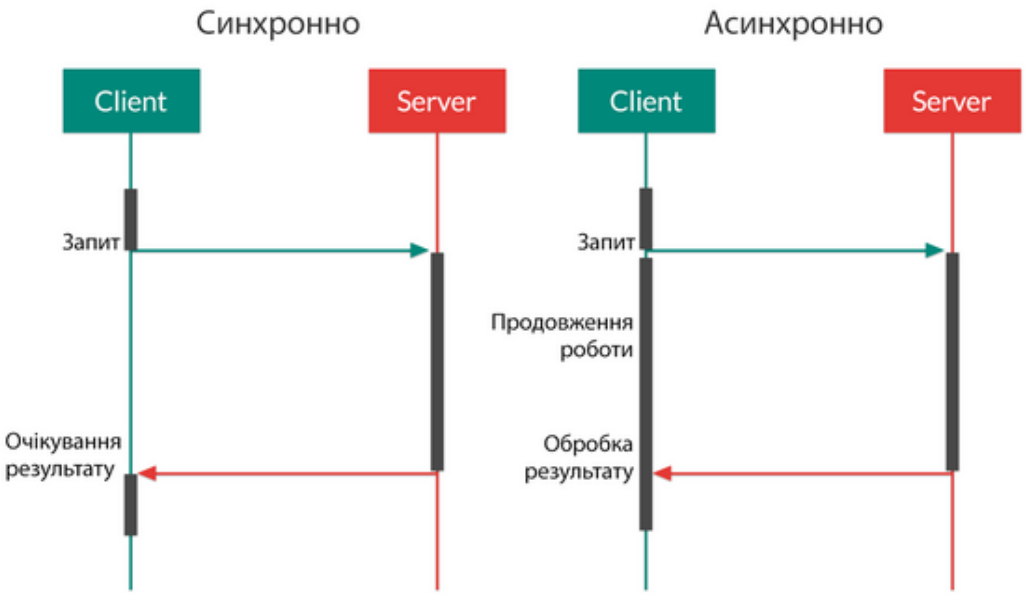


Рисунок 2.7 – Порівняння синхронного та асинхронного підходів

Цей підхід є дуже актуальним для створення BitTorrent-клієнта, оскільки дозволяє працювати з великою кількістю клієнтів без потреби створювати по потоку операційної системи на кожного з них. Це дозволяє працювати з різними клієнтами навіть у всього одному потоці, але для досягнення максимальної ефективності, задачі розподіляються по набору заздалегідь ініціалізованих потоків, відомому як пул потоків (thread pool).

Одною з переваг мови програмування C# є те, що у ній вперше був введений синтаксис **async/await**, спеціально призначений для асинхронного програмування і значно спрощуючий керування операціями такого типу.

Асинхронні методи помічаються ключовим словом **async** у своєму оголошенні. Такі методи можуть містити виклики інших асинхронних методів і використовувати ключове слово **await** для очікування завершення асинхронних операцій без блокування потоку виконання.

Отже, парадигма асинхронного програмування, в поєднанні з мовою програмування C#, надає можливості для ефективної обробки великої кількості мережевих запитів через зручний синтаксис.

Висновки до розділу 2

В другому розділі було розглянуто усі наявні варіанти щодо мови програмування, середи розробки, фреймворків для написання Desktop-застосунків та обрано найбільш актуальні для поставленої задачі.

Було також описано шаблони проєктування та підходи програмування, що стануть основою для створюваного BitTorrent-клієнта, і принцип використання протоколу TCP, на якому базується комунікація між учасниками мережі.

Використовуючи результати, отримані з цього розділу, можна приступити до безпосередньо етапу реалізації застосунку.

3 РЕАЛІЗАЦІЯ ЛОГІКИ ЗАСТОСУНКУ ТА ПОБУДУВАННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Створення проєкту. Обробка вхідних даних

Для початку етапу реалізації застосунку мовою програмування C# у середовищі JetBrains Rider, необхідно ініціалізувати проєкт (рис. 3.1).

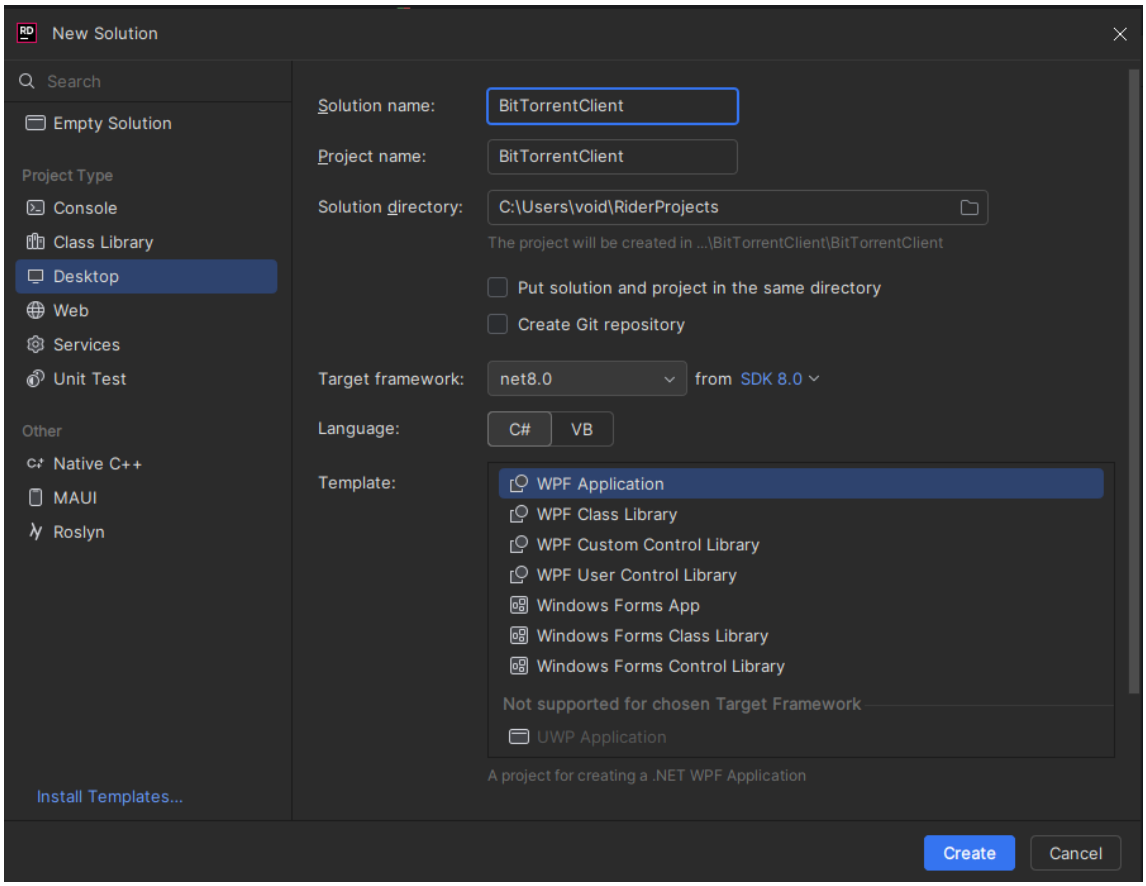


Рисунок 3.1 – Вікно створення проєкту у JetBrains Rider

Усього застосунок можна розділити на три проєкти:

1) **BitTorrentClient** – бібліотека, що містить логіку застосунку, реалізацію протоколу BitTorrent, спільну для інших проєктів;

2) **BitTorrentClient.CLI** – консольний застосунок, проста реалізація функціоналу, наданого у бібліотеці з детальним логуванням для простого відлагодження;

3) **BitTorrentClient.GUI** – WPF-застосунок з повною реалізацією функціоналу. Має графічний інтерфейс, доступний для усіх користувачів.

Створивши проєкт, можна приступати до створення вхідної точки в бібліотеку – клас *TorrentClient*, що буде містити усі необхідні методи для управління клієнтом.

Першим кроком для ініціалізації клієнта є завантаження метаданих про торрент через метод *LoadFromStream* (рис. 3.2).

```
public void LoadFromStream(Stream stream)
{
    var reader = new BencodeReader(stream);
    var parser = new BencodeParser();
    var torrentBencode = (BDictionary)parser.Parse(reader);
    var info = (BDictionary)torrentBencode["info"];
    var pieces = ((BString)info["pieces"]).Value;
    var pieceHashes = new List<byte[]>();
    for (var i = 0; i < pieces.Length; i += 20)
    {
        var piece = pieces[i..(i + 20)].ToArray();
        pieceHashes.Add(piece);
    }
    var infoHash = SHA1.HashData(info.EncodeAsBytes());
    Torrent = new TorrentFile
    {
        Announce = torrentBencode["announce"].ToString()!,
        InfoHash = infoHash,
        PieceHashes = pieceHashes,
        PieceLength = ((BNumber)info["piece length"]).Value,
        Length = (int)((BNumber)info["length"]).Value,
        Name = info["name"].ToString()!,
        Comment = torrentBencode["comment"].ToString()
    };
```

Рисунок 3.2 – Метод *LoadFromStream*. Завантаження метаданих

Метод приймає об’єкт типу Stream, що узагальнює його та дозволяє отримувати дані з найрізноманітніших джерел: файлів (FileStream), мережі (NetworkStream), пам’яті (MemoryStream) тощо.

Для парсингу файлу типу Bencode використовується бібліотека BencodeNET. Для того, щоб додати залежність до проєкту необхідно знайти її у каталозі пакетного менеджеру NuGet та встановити (рис. 3.3).

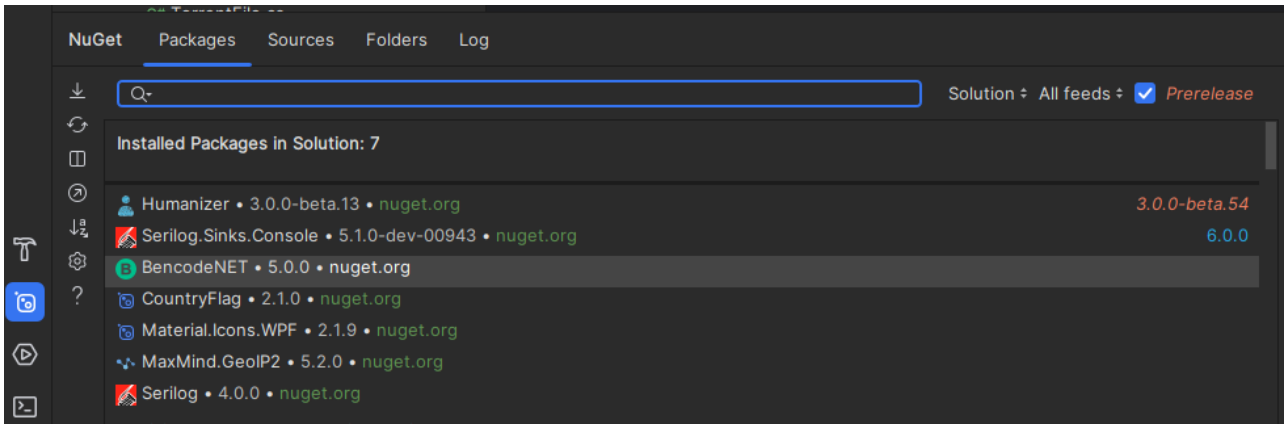


Рисунок 3.3 – Пакет BencodeNET у репозиторії NuGet

Згідно зі специфікацією, описаною у розділі 1, поле *pieces* розділяється на список, кожен елемент якого є масивом з 20 байтів, що відповідає за хеш відповідного фрагменту файлу. З тексту секції *info* розраховується SHA1-хеш, що далі носить назву InfoHash. Інші поля записуються без змін у об’єкт типу TorrentFile, що відповідає за метадані завантаження.

Далі створюється черга з задач, що наповнюється фрагментами торренту та використовується далі для організації процесу завантаження (рис. 3.4).

```
        Tracker = new Tracker(Torrent.Announce);

        var blocksPerPiece = (int)Torrent.PieceLength / BlockSize;
        for (var i = 0; i < Torrent.PieceHashes.Count; i++)
        {
            for (var j = 0; j < blocksPerPiece; j++)
            {
                WorkQueue.Enqueue(new BlockRequest(i, j));
            }
        }
```

Рисунок 3.4 – Метод *LoadFromStream*. Створення задач

Отже, отримані дані дозволяють перейти до наступного процесу – встановлення з’єднання з пірами.

3.2 Реєстрація клієнта у трекері. Встановлення з'єднання з пірами

Для того, щоб зареєструватись у мережі та отримати дані про інші піри, необхідно надіслати йому HTTP-запит з власно згенерованим ідентифікатором клієнта та InfoHash, що ідентифікує бажаний торрент (рис. 3.5).

```
    public async Task<IEnumerable<IPEndPoint>> Announce(byte[] peerId, byte[]
infoHash, long left)
    {
        var parser = new BencodeParser();
        var trackerResponseStream = await GetTrackerResponse(peerId, infoHash,
left);
        var trackerResponseReader = new BencodeReader(trackerResponseStream);
        var trackerResponse = (BDictionary)parser.Parse(trackerResponseReader);
        Interval = (BNumber)trackerResponse["interval"];
        var peerAddresses = ((BString)trackerResponse["peers"]).Value;
        var peers = new List<IPEndPoint>();
        for (var i = 0; i < peerAddresses.Length; i += 6)
        {
            var ip = peerAddresses[i..(i + 4)];
            var port = peerAddresses[(i + 4)..(i + 6)];
            var peer =
                new IPEndPoint(new IPAddress(ip.Span),
BinaryPrimitives.ReadUInt16BigEndian(port.Span));
            peers.Add(peer);
        }
        return peers;
    }
```

Рисунок 3.5 – Надсилання HTTP-запиту

Відповідь також приходить у форматі Bencode, тому знову використовується клас *BencodeReader* для отримання списку пірів. Для кожного з них створюється об'єкт класу *IPEndPoint*, що містить IP-адресу (перші 4 байти) та порт (2 байти) кожного піру.

Логіка підтримки з'єднання з пірами знаходиться в класі *PeerHandler*. Для його створення використовується метод *Connect*, що встановлює TCP-з'єднання та здійснює handshake (рис. 3.6).

Робота з кожним пакетом (*handshake, piece, message* та ін.) інкапсулюється у відповідний клас, що містить два методи – *ReadFromStreamAsync* та *WriteFromStreamAsync* для їх читання та писання у потік. Приклад читання Handshake, що зчитує поля по байтам та виконує необідні перевірки, щоб переконатись у валідності з'єднання, наведено на рис. 3.7.

```
  public static async Task<PeerHandler?> Connect(IPEndPoint endPoint, byte[]
infoHash, byte[] peerId)
    {
        try
        {
            var tcpClient = new TcpClient();
            tcpClient.ReceiveTimeout = 1000;
            tcpClient.SendTimeout = 1000;
            await tcpClient.ConnectAsync(endPoint);
            var stream = tcpClient.GetStream();
            var handshake = new Handshake(infoHash, peerId);
            await handshake.WriteToStreamAsync(stream);
            var response = await Handshake.ReadFromStreamAsync(stream);
            var peer = new Peer(endPoint, response.PeerId);
            var isConnected = response.InfoHash.SequenceEqual(infoHash);
            if (isConnected)
            {
                Log.Information("Handshake successful with {peer}",
peer.DisplayName);
            }
            return isConnected ? new PeerHandler(peer, tcpClient) : null;
        }
        catch (SocketException e)
        {
            Log.Error("Failed to connect to {endpoint}", endPoint);
        }
```

Рисунок 3.6 – Метод *Connect*

```
    public static async Task<Handshake> ReadFromStreamAsync(Stream stream)
    {
        var identifierLength = stream.ReadByte();
        if (identifierLength != IdentifierLength)
            throw new InvalidHandshakeException("Invalid protocol identifier
length.");
        var identifier = new byte[19];
        await stream.ReadExactlyAsync(identifier, 0, 19);
        if (!identifier.SequenceEqual(Encoding.UTF8.GetBytes(Identifier)))
            throw new InvalidHandshakeException("Invalid protocol identifier
header.");
        var extensions = new byte[8];
        var infoHash = new byte[20];
        var peerId = new byte[20];

        await stream.ReadExactlyAsync(extensions, 0, 8);
        await stream.ReadExactlyAsync(infoHash, 0, 20);
        await stream.ReadExactlyAsync(peerId, 0, 20);
        return new Handshake(infoHash, peerId);
    }
```

Рисунок 3.7 – Метод *ReadFromStreamAsync*

Зворотній процес для передачі своєї частини handshake іншому піру описаний у методі *WriteToStreamAsync* (рис. 3.8).

```
   public async Task WriteToStreamAsync(Stream stream)
   {
```

```
     stream.WriteByte(IdentifierLength);
     await stream.WriteAsync(Encoding.UTF8.GetBytes(Identifier).AsMemory(0, 19));
     await stream.WriteAsync(new byte[8]);
     await stream.WriteAsync(InfoHash);
     await stream.WriteAsync(PeerId);
    }
```

Рисунок 3.8 – Метод *WriteToStreamAsync*

Класи для інших типів повідомлень працюють за таким самим принципом і відрізняються лише полями, що відправляються.

Здійснивши успішне рукостискання з піром, можна починати обмінюватись з ним повідомленнями через встановлене TCP-з'єднання.

3.3 Створення обробників PeerHandler та DownloadWorker

Підключення до пірів та початок обробки повідомлень здійснюються у методі *StartWorkers* класу *TorrentClient* (рис. 3.9).

```
 public void StartWorkers()
  {
    var peerEndpoints = Tracker.Announce(PeerId, Torrent.InfoHash,
Torrent.Length).Result;
    PeerHandlers = new ConcurrentDictionary<IPEndPoint, PeerHandler>();
    _workerCancellationTokenSource = new CancellationTokenSource();
    foreach (var endpoint in peerEndpoints)
    {
      Log.Information("Connecting to the peer: {endpoint}", endpoint);
      var ct = _workerCancellationTokenSource.Token;
      Task.Run(async () =>
      {
        var handler = await PeerHandler.Connect(endpoint, Torrent.InfoHash,
PeerId);
        if (handler == null) return;
        handler.BlockReceived += payload => { ReceivedBlocks.Add(payload); };
        PeerHandlers[endpoint] = handler;
        DownloadWorkers[endpoint] = new DownloadWorker(handler, WorkQueue);
        DownloadWorkers[endpoint].Start(BlockSize, ct);

        while (handler.IsConnected && !ct.IsCancellationRequested)
        {
          await handler.ReadMessageAsync();
        }
        PeerHandlers.Remove(endpoint, out _);
        DownloadWorkers.Remove(endpoint, out _);
      }, ct);
    }
  }
```

Рисунок 3.9 – Метод *StartWorkers*

У ньому, після отримання даних про пірів з трекеру та встановлення рукостискання з кожним, створюються об'єкти-обробники типу *PeerHandler, що* отримують та опрацьовують вхідні повідомлення від пірів. Одночасно з ними

працюють обробники типу *DownloadWorker*. Вони відповідають за надсилання запитів на відправку фрагментів та їх отримання.

Усі обробники додаються у колекції типу *ConcurrentDictionary*. Це забезпечує безпечний доступ до їх елементів між різними потоками без ризиків станів гонитви (англ. race condition), блокувань та іншим проблем, пов'язаних з багатопотоковістю. Для можливості без проблем зупинити роботу будь-якого обробника, кожному з них надається *CancellationToken*. Він використовуються для паузи або відмінення завантаження.

3.4 Надсилання запитів на фрагменти та обробка вхідних повідомлень

Основою обробки повідомлень є метод *ReadMessageAsync* обробника *PeerHandler* (рис. 3.10).

```
    public async Task ReadMessageAsync()
    {
        try
        {
            var message = await Message.ReadFromStreamAsync(Stream);

            Log.Verbose("Message from {peer}, type: {type}, payload length:
{length}",
                Peer.DisplayName, message.Type, message.Payload.Length);

            switch (message.Type)
            {
                case MessageType.KeepAlive:
                    break;
                case MessageType.Unchoke:
                    Choked = false;
                    break;
                case MessageType.Bitfield:
                    Peer.Bitfield = new Bitfield(message.Payload[1..]);
                    break;
                case MessageType.Piece:
                    var payload = new PiecePayload(message.Payload);
                    BlockReceived.Invoke(payload);
                    break;
                default:
                    throw new ArgumentOutOfRangeException();
            }
        }
        catch (SocketException e)
        {
            Log.Error("Disconnected from {endpoint}", Peer.EndPoint);
            IsConnected = false;
        }
        catch (IOException e)
        {
```

```
            Log.Error("Failed to read from socket {endpoint}", Peer.EndPoint);
            IsConnected = false;
        }
    }
```

Рисунок 3.10 – Метод *ReadMessageAsync*

Спочатку, він читає з сокету повідомлення фіксованої структури – 4 байти, що означають довжину тіла, та його зміст. Далі, в залежності від типу повідомлення, відбувається його обробка.

Повідомлення типу **KeepAlive** існують для підтримки з'єднання, тому додаткової обробки не потребують.

При отриманні повідомлень **Choke** та **Unchoke** ставиться відповідний флаг у об'єкті обробника. Він означає, чи готовий пір приймати запити на отримання, та враховується перед відправкою запитів на завантаження фрагментів.

Тип **Bitfield** сповіщує про те, які фрагменти наявні у піра, і аналогічно відображає цей стан у об'єкті обробника.

Для повідомлення типу **Piece** використовується подія BlockReceiver, що сповіщує обробник завантаження про отримані байти фрагменту та посилає їх.

Обробник завантаження починає свою роботу за допомогою методу Start (рис. 3.11).

```
  public void Start(uint blockSize, CancellationToken cancellationToken)
    {
        Task.Run(async () =>
        {
            while (Handler.IsConnected
&& !cancellationToken.IsCancellationRequested)
            {
                if (WorkQueue.TryDequeue(out var blockRequest))
                {
                    if (Handler.Peer.Bitfield != null &&
                        !Handler.Peer.Bitfield.HasPiece(blockRequest.Piece) ||
Handler.Choked)
                    {
                        WorkQueue.Enqueue(blockRequest);
                        continue;
                    }
                    Log.Verbose("Requesting block {piece}-{block} from {peer}",
                        blockRequest.Piece, blockRequest.BlockIndex,
                        Handler.Peer.DisplayName);
                    var requestPayload = new
RequestPayload((uint)blockRequest.Piece,
                        (uint)(blockRequest.BlockIndex * blockSize), blockSize);
                    var request = new Message(requestPayload.AsBytes());
```

```
                    await request.WriteToStreamAsync(Handler.Stream);
                    try
                    {
                        var tcs = new TaskCompletionSource();
                        var ct = new CancellationTokenSource(5000);
                        ct.Token.Register(() => tcs.TrySetCanceled(),
useSynchronizationContext: false);
                        PieceNotify cb = _ => tcs.TrySetResult();
                        Handler.BlockReceived += cb;
                        await tcs.Task;
                        Handler.BlockReceived -= cb;
                    }
                    catch (Exception e)
                    {
                        WorkQueue.Enqueue(blockRequest);
                        Handler.IsConnected = false;
                        break;
                    }
                }
            }
        }, cancellationToken);
    }
```

Рисунок 3.11 – Метод *Start*

Обробник працює, поки не закінчились фрагменти для завантаження або не було подано сигнал про зупинення. Взявши задачу з черги, він перевіряє за допомогою бітового поля, чи є у піра заданий фрагмент. Якщо немає, він повертає його в кінець черги для опрацювання іншим обробником і береться за інший.

Далі створюється запит – повідомлення *request* та відправляється до піра. Після чого чекає на отримання від нього повідомлення *piece* з самим фрагментом за допомогою події *BlockReceived*.

Після того, як у черзі не залишилось роботи, файл можна вважати завантаженим. Для завершення процесу, метод *WriteToFile* записує усі фрагменти з пам’яті у файл (рис. 3.12).

```
    public void WriteToFile(string filePath)
    {
        var resultBytes = new byte[BlockSize * ReceivedBlocks.Count];
        foreach (var block in ReceivedBlocks)
        {
            var pos = block.Index * Torrent.PieceLength + block.Begin;
            Buffer.BlockCopy(block.Data, 0, resultBytes, (int)pos, BlockSize);
        }
        File.WriteAllBytes(filePath, resultBytes);
    }
```

Рисунок 3.12 – Метод *WriteToFile*

Отже, вся необхідна логіка застосунку була реалізована у бібліотеці *BitTorrentClient*. Далі, на базі його функціоналу будуть побудовані реалізації консольного та графічного клієнтів.

3.5 Реалізація консольного клієнта

Найпростішою версією застосунку є консольний клієнт, що приймає шлях до файлу в аргументах командного рядка та запускає методи, реалізовані у бібліотеці (рис. 3.13).

```
  static void Main(string[] args)
  {
    Log.Logger = new LoggerConfiguration()
      .MinimumLevel.Verbose()
      .WriteTo.Console()
      .CreateLogger();

    using var fs = File.OpenRead(args[0]);
    var client = new TorrentClient();
    client.LoadFromStream(fs);
    Log.Information("Announce URL: {announce}", client.Torrent.Announce);
    Log.Information("Comment: {comment}", client.Torrent.Comment);
    Log.Information("Length: {length}", client.Torrent.Length.Bytes());
    Log.Information("Piece length: {piece_length}",
client.Torrent.PieceLength.Bytes());
    Log.Information("Name: {name}", client.Torrent.Name);

    client.StartWorkers();
    while (client.PeersConnected != 0 && client.BlocksPending != 0)
    {
      Thread.Sleep(5000);
      Log.Information("Downloaded {downloaded_count} blocks, {total_count} left",
        client.BlocksReceived,
        client.BlocksPending);
    }

    client.WriteToFile(args[1]);
  }
```

Рисунок 3.13 – Метод *Program* у *BitTorrentClient.CLI*

Спочатку, налаштовується логування у консоль за допомогою бібліотеки *Serilog*. Далі, відкривається торрент-файл і його метадані виводяться на екран (рис. 3.14).

```
[23:52:03 INF] Announce URL: http://bttracker.debian.org:6969/announce
[23:52:03 INF] Comment: "Debian CD from cdimage.debian.org"
[23:52:03 INF] Length: 629 MB
[23:52:03 INF] Piece length: 256 KB
[23:52:03 INF] Name: debian-12.5.0-amd64-netinst.iso
```

Рисунок 3.14 – Метадані торрент-файлу

Кожне вхідне повідомлення та запит на відправку фрагмент також логуються (рис. 3.15).

```
[03:59:22 VRB] Message from -TR3000-v3thaisgkgai@188.212.112.163:32767, type: Piece, payload length: 16393
[03:59:22 VRB] Requesting block 1589-3 from -TR3000-v3thaisgkgai@188.212.112.163:32767
[03:59:22 VRB] Message from -qB4650-JgLDoxRr((Cl@87.227.189.89:55000, type: Piece, payload length: 16393
[03:59:22 VRB] Requesting block 899-4 from -qB4650-JgLDoxRr((Cl@87.227.189.89:55000
[03:59:22 VRB] Message from -DE13F0-l.Ajw5zU*1sI@76.193.127.96:62664, type: Piece, payload length: 16393
[03:59:22 VRB] Requesting block 1796-6 from -DE13F0-l.Ajw5zU*1sI@76.193.127.96:62664
[03:59:22 VRB] Message from -qB4650-eCIkNhgWot6L@45.13.104.37:7891, type: Piece, payload length: 16393
[03:59:22 VRB] Requesting block 189-8 from -qB4650-eCIkNhgWot6L@45.13.104.37:7891
[03:59:22 VRB] Message from -TR3000-hne5j5pncgl0@162.238.119.27:51413, type: Piece, payload length: 16393
[03:59:22 VRB] Requesting block 2394-6 from -TR3000-hne5j5pncgl0@162.238.119.27:51413
[03:59:22 VRB] Message from -qB4640--xL8inp-eNPP@70.175.76.239:22222, type: Piece, payload length: 16393
[03:59:22 VRB] Requesting block 509-7 from -qB4640--xL8inp-eNPP@70.175.76.239:22222
[03:59:22 VRB] Message from -DE211s-ADp8-TngOKti@46.232.210.225:58048, type: Piece, payload length: 16393
```

Рисунок 3.15 – Логування повідомлень

Створена консольна реалізація достатня для відлагодження коду, оскільки реалізує весь функціонал бібліотеки та виводить інформацію у консоль. Але вона не є зручною для кінцевого користувача, оскільки він більше потребує зручного та простого для використання інтерфейсу, ніж детальні дані про кожне отримане та надіслане повідомлення.

3.6 Реалізація графічного клієнта

3.6.1 Список завантажень

Для початку реалізації графічного клієнта необхідно створити проєкт типу *WPF Application* з назвою *BitTorrentClient.WPF*.

Графічний клієнт складається з наступних екранів:

– список завантажень;

– деталі про завантаження;

– схема з пірами, з якими встановлено з’єднання.

Активний екран залежить від обраної ViewModel, за що відповідає сервіс NavigationService. Він містить один метод – *Navigate*, що здійснює перехід застосунку до іншого екрану.

Основою застосунку є можливість завантажувати декілька файлів одночасно, що досягається створенням **списку завантажень**. За нього відповідає сервіс *DownloadManager* (рис. 3.16).

```
public class DownloadManager
{
    public ObservableCollection<TorrentClient> TorrentClients { get; } = [];
    public ObservableCollection<DownloadViewModel> DownloadViewModels { get; } = [];

    public void AddClient(string torrentFilePath)
    {
        using var fs = File.OpenRead(torrentFilePath);
        var client = new TorrentClient();
        client.LoadFromStream(fs);

        TorrentClients.Add(client);
        DownloadViewModels.Add(new DownloadViewModel(client)
        {
            FileName = client.Torrent.Name,
            TotalMb = client.BlocksPending * TorrentClient.BlockSize / 1_048_576
        });
        new Thread(client.StartWorkers).Start();
    }

    public void Stop(TorrentClient client)
    {
        client.Pause();
        var vm = DownloadViewModels.First(vm => vm.PeerId == client.PeerId);
        TorrentClients.Remove(client);
        DownloadViewModels.Remove(vm);
    }

    public void RefreshViewModels(Dictionary<byte[], int> lastBlocks)
    {
        foreach (var client in TorrentClients)
        {
            var downloadVm = DownloadViewModels.First(vm =>
                vm.PeerId == client.PeerId);

            downloadVm.BlocksReceived = client.BlocksReceived;
            downloadVm.Progress = (float)client.BlocksReceived /
                (client.BlocksReceived + client.BlocksPending) * 100;
```

```
            downloadVm.SpeedKbps         =         (client.BlocksReceived         -
lastBlocks.GetValueOrDefault(client.PeerId, 0)) *
                TorrentClient.BlockSize / 1024;
            downloadVm.ReceivedMb        =        downloadVm.BlocksReceived        *
TorrentClient.BlockSize / 1_048_576;

            lastBlocks[client.PeerId] = downloadVm.BlocksReceived;
        }
    }
}
```

Рисунок 3.16 – Клас *DownloadManager*

Для кожного завантаження створюється окремий об’єкт класу TorrentClient. Метод *AddClient* додає нове завантаження, *Stop* – зупиняє клієнт і видаляє його зі списку завантажень.

Клас *DownloadViewModel* представляє дані, що відображаються на екрані. Для того, щоб дані відновлювались в інтерфейсі після їх зміни, потрібно в сеттері викликати метод **OnPropertyChanged** (рис. 3.17).

```
    private string _fileName;
    public string FileName
    {

        get => _fileName;
        set { _fileName = value; OnPropertyChanged(); }
    }
```

Рисунок 3.17 – Приклад використання *OnPropertyChanged*

Клас *DownloadViewModel* містить такі властивості:

– *FileName* – назва файлу;
– *Progress* – кількість відсотків, на скільки завантажено файл;
– *BlocksReceived* – кількість отриманих блоків;
– *ReceivedMb* – обсяг отриманого файлу, Мбайт;
– *SpeedKbps* – швидкість завантаження, кбайт/с;
– *TotalMb* – розмір файлу, Мбайт.

Представивши дані про кожне завантаження у своєму ViewModel, можна легко відобразити їх список за допомогою елементу *ListView* (рис. 3.18).

```
        <ListView ItemsSource="{Binding Downloads}" SelectedItem="{Binding
SelectedDownload}">
            <ListView.ItemTemplate>
                <DataTemplate DataType="{x:Type viewModels:DownloadViewModel}">

                    <StackPanel>
                        <TextBlock Text="{Binding FileName}" />

                        <ProgressBar Height="20" Width="770" Value="{Binding
Progress}" />
                        <TextBlock Text="{Binding Progress,
StringFormat={}{0:F2}%}" />

                        <TextBlock>
                            <Run Text="{Binding ReceivedMb}"/>
                            <Run Text="/"/>
                            <Run Text="{Binding TotalMb}"/>
                            <Run Text="MB"/>
                        </TextBlock>
                        <TextBlock>
                            <Run Text="Speed:"/>
                            <Run Text="{Binding SpeedKbps}"/>
                            <Run Text="Kb/s"/>
                        </TextBlock>
                    </StackPanel>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
```

Рисунок 3.18 – Вид *DownloadListView.xaml*

Результатом цього є готовий екран списку завантажень (рис. 3.19), що зручно відображає інформацію про прогрес кожного з них.

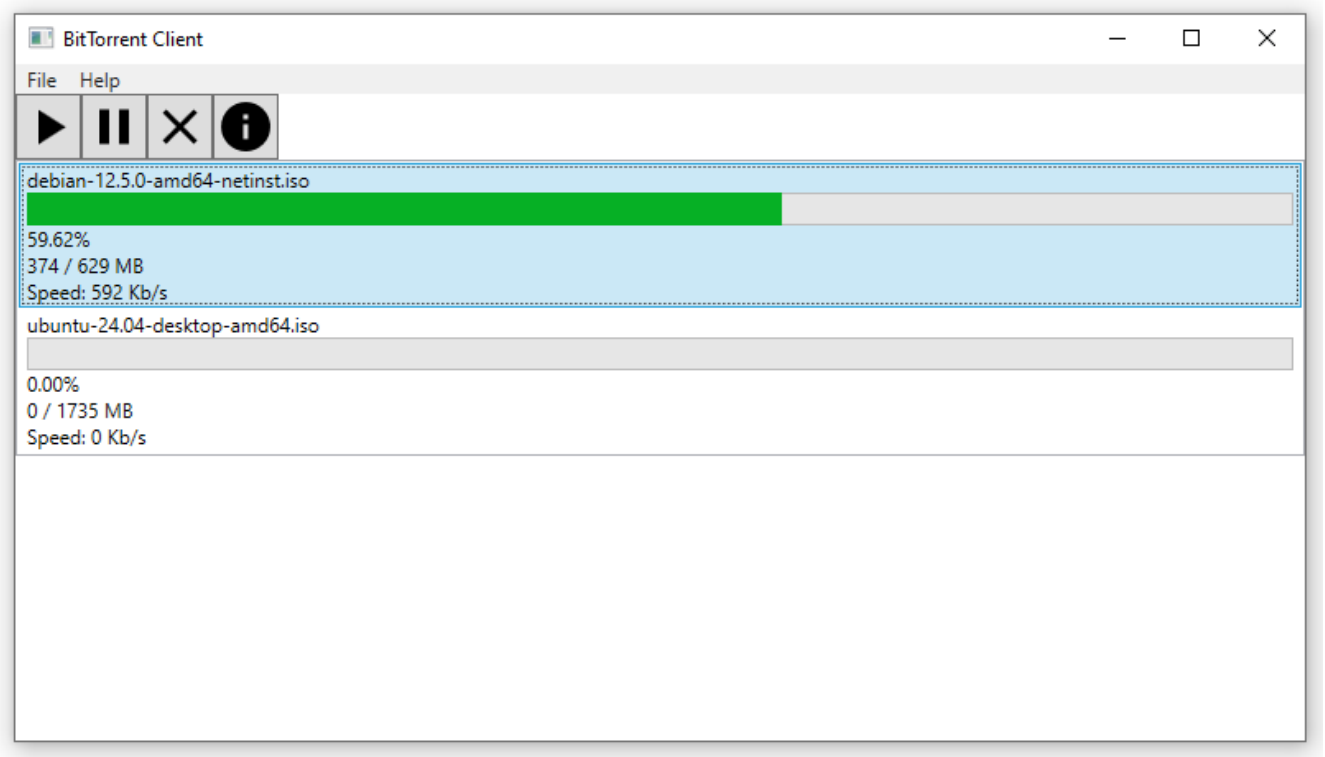


Рисунок 3.19 – Інтерфейс списку завантажень

Наступним кроком є реалізація дій, що представлені у меню вище – початок завантаження, пауза, стоп та перехід до інформації про торрент.

3.6.2 Патерн Command

Дії у фреймворку WPF представляються у вигляді патерну *Команда* (англ. Command). Він використовується для відділення логіки дії від її виклику, що сприяє кращій підтримуваності та тестованості коду. Цей патерн дозволяє створювати інтерфейси користувача, які можуть реагувати на дії користувача, такі як натискання кнопок, шляхом прив’язування команд до елементів інтерфейсу.

Наприклад, розглянемо команду **OpenCommand**, що відповідає за відкриття діалогового вікну вибору торрент-файлу і додавання його до списку завантажень (рис. 3.20).

```
public class OpenCommand : CommandBase
{
    private readonly DownloadManager _downloadManager;

    public OpenCommand(DownloadManager downloadManager)
    {
        _downloadManager = downloadManager;
    }

    public override void Execute(object? parameter)
    {
        var dialog = new Microsoft.Win32.OpenFileDialog();
        dialog.DefaultExt = ".torrent";
        dialog.Filter = "Torrent files (.torrent)|*.torrent";

        bool? result = dialog.ShowDialog();
        if (result == true)
        {
            var filename = dialog.FileName;
            _downloadManager.AddClient(filename);
        }
    }
}
```

Рисунок 3.20 – Клас *OpenCommand*

Тепер цю команду можна легко прив’язати до натискання кнопки або елементу меню (рис. 3.21).

```
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="_File">
                <MenuItem Header="_Open" Command="{Binding OpenCommand}"/>
                <MenuItem Header="_Exit" Command="{Binding ExitCommand}" />
            </MenuItem>
            <MenuItem Header="_Help">
                <MenuItem Header="_About" Command="{Binding AboutCommand}" />
            </MenuItem>
        </Menu>
```

Рисунок 3.21 – Прив’язка команд

Інші команди *(PauseCommand, ResumeCommand, StopCommand та ін.)* було створено аналогічним способом, відрізняється лише виклик інших методів *DownloadManager*. Також було створено команду *NavigateCommand*, що використовує *NavigationService* для переходу на інший екран.

3.6.3 Екран деталей завантаження у графічному застосунку

В екрані деталей до наявних в *DownloadViewModel* було додано властивості з метаданих про торрент та технічні деталі, такі як довжина фрагменту, кількість підключених пірів та кількість отриманих блоків (рис. 3.22).

```
<StackPanel>
    <TextBlock>
        <Run FontWeight="Bold" Text="Client Peer ID:" />
        <Run Text="{Binding PeerId, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="File name:" />
        <Run Text="{Binding FileName, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Announce URL:" />
        <Run Text="{Binding AnnounceUrl, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Comment:" />
        <Run Text="{Binding Comment, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Infohash:" />
        <Run Text="{Binding InfoHash, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Piece length:" />
        <Run Text="{Binding PieceLength, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Block size:" />
        <Run Text="{Binding BlockSize, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Peers connected:" />
        <Run Text="{Binding PeersConnected, Mode=OneWay}" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Received:" />
        <Run Text="{Binding ReceivedMb, Mode=OneWay}" />
        <Run Text="/" />
        <Run Text="{Binding TotalMb, Mode=OneWay}" />
        <Run Text="MB" />
    </TextBlock>
    <TextBlock>
        <Run FontWeight="Bold" Text="Blocks received/pending:" />
        <Run Text="{Binding BlocksReceived, Mode=OneWay}" />
        <Run Text="/" />
        <Run Text="{Binding BlocksPending, Mode=OneWay}" />
    </TextBlock>
</StackPanel>
```

Рисунок 3.22 – Вид *DownloadDetailsView.xaml*

Другою вкладкою є схема, що складається з усіх пірів у мережі та інформації про них (IP-адреса, ID та країна походження). У центрі цієї схеми зображено власний клієнт у вигляді піктограми, до якого лініями під'єднані інші учасники мережі. Координати кожного піра визначаються випадково з перевіркою на перетин (рис. 3.23).

```
    private (int, int) GetNextCoordinates()
    {
        var x = Random.Shared.Next(1024);
        var y = Random.Shared.Next(768);
        var rect = new Rectangle(x, y, 140, 140);
        foreach (var peer in Peers)
        {
            var other = new Rectangle(peer.X, peer.Y, 140, 140);
            if (rect.IntersectsWith(other))
            {
                return GetNextCoordinates();
            }
        }
        return (x, y);
    }
```

Рисунок 3.23 – Метод *GetNextCoordinates*

Останньою деталлю є визначення країни піра за його IP-адресою. Це здійснюється за допомогою бази даних MaxMind GeoIP [14]. Для цього було завантажено її та створено сервіс GeoService засобами бібліотеки для C# MaxMind.GeoIP2 (рис. 3.24).

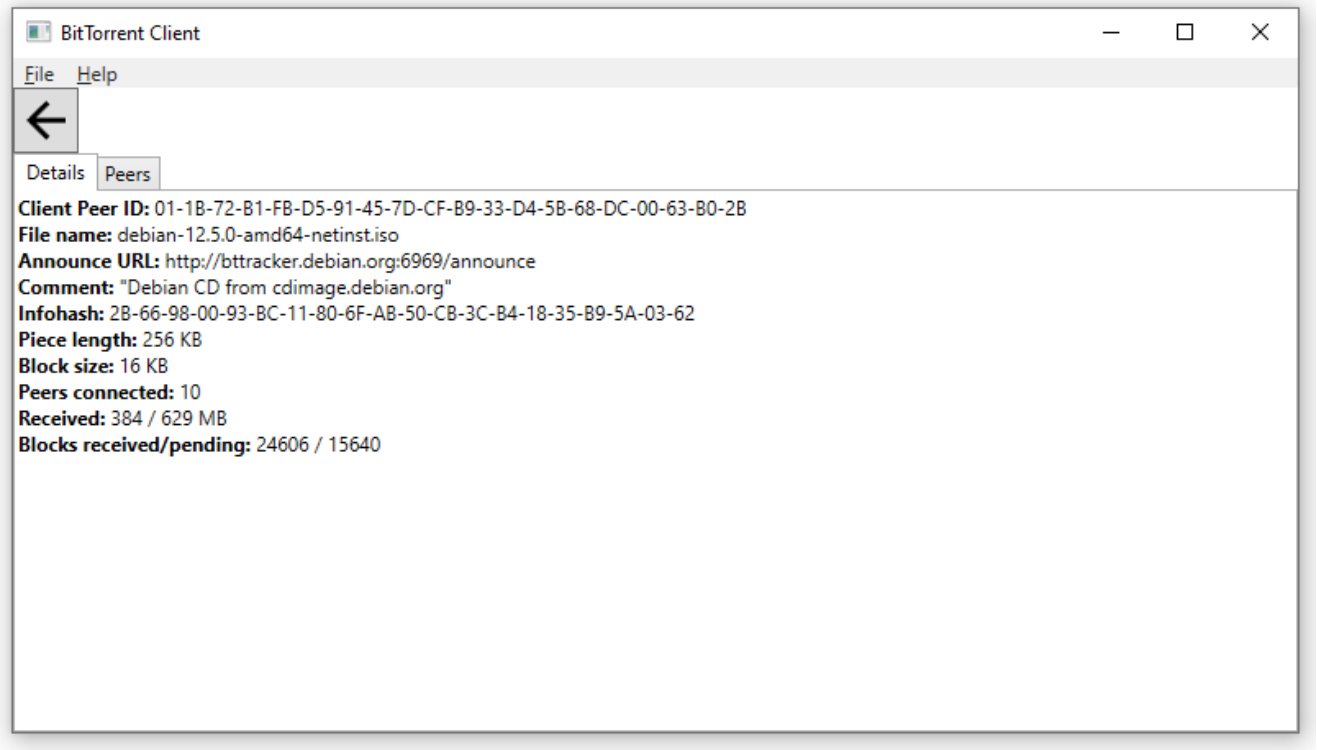
```
public class GeoService
{
    private readonly DatabaseReader _databaseReader;

    public GeoService()
    {
        _databaseReader = new DatabaseReader("GeoLite2-Country.mmdb");
    }
    public string GetCountryIsoCodeFromIp(string ipAddress)
    {
        try
        {
            var response = _databaseReader.Country(ipAddress);
            return response.Country.IsoCode;
        }
        catch (AddressNotFoundException)
        {
            return "UA";
        }
    }
}
```

Рисунок 3.24 – Клас GeoService

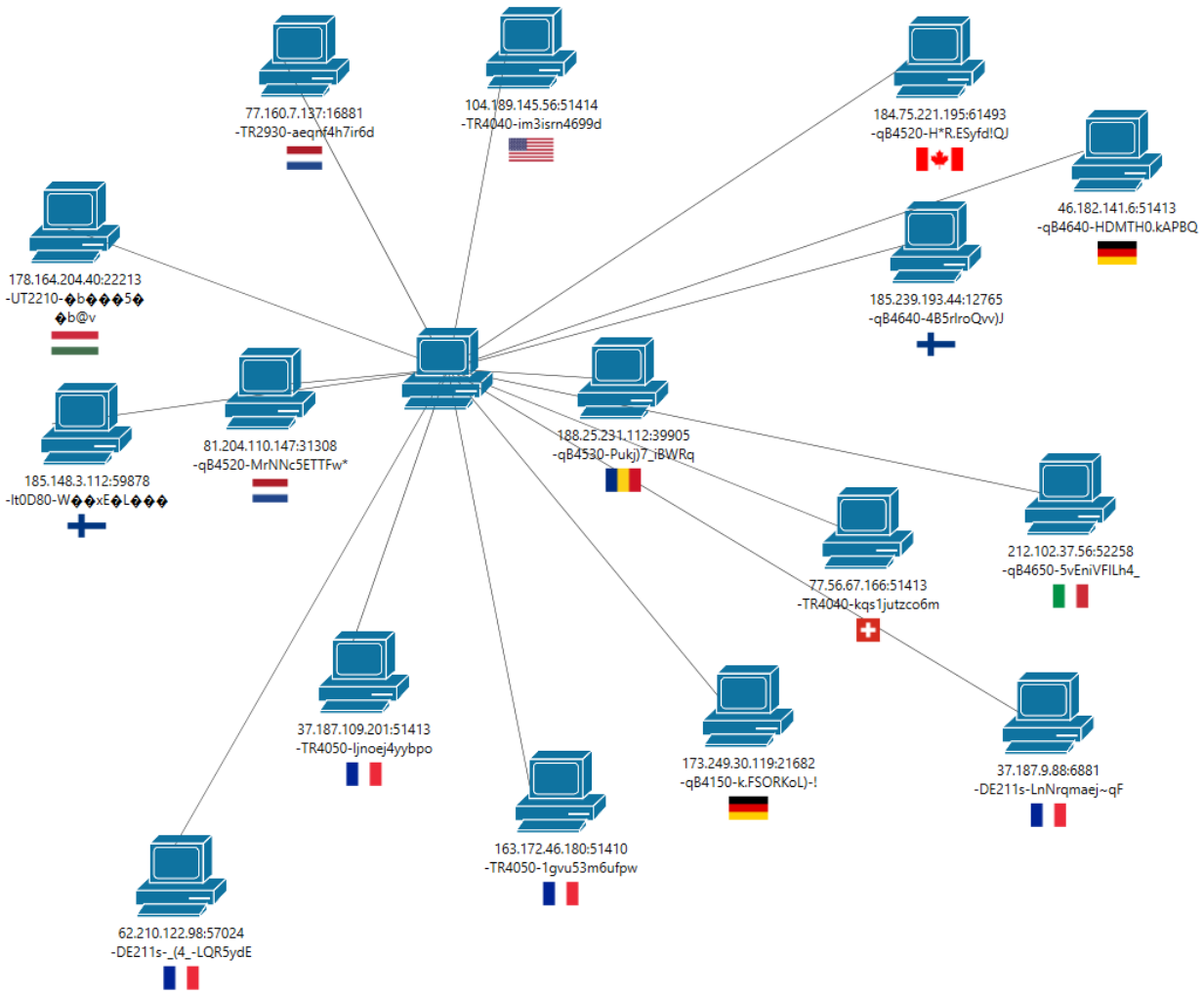
Код файлу *DownloadPeersView.xaml*, що представляє шаблон цього виду, наведено у додатку А. Його було реалізовано за допомогою елементу інтерфейсу *Canvas*, що дозволяє додавати елементи на будь-які координати (x, y) та *ScrollView*, що забезпечує можливість прокручення схеми, якщо вона повністю не вміщується на екран.

Відображення прапорів здійснюється за допомогою пакету *CountryFlag*, що містить елемент контролю WPF для цього.

Результат виглядає наступним чином (рис. 3.25).



а)



б)

Рисунок 3.25 – Інтерфейс розробленого графічного клієнта: а – вкладка Details; б – вкладка Peers

В результаті було створено дві вкладки – Details (метадані про торент та технічні деталі про завантаження) та Peers (схема пірів у мережі). На цьому весь функціонал екрану деталей завантаження є успішно реалізованим.

Висновки до розділу 3

В третьому розділі було реалізовано застосунок засобами мови програмування C#.

Спочатку було створено бібліотеку для роботи з протоколом BitTorrent. Після цього, на базі цієї бібліотеки було побудовано два застосунки – простий з консольним інтерфейсом, що має детальне логування та підходить для відлагодження, та графічний з більш розширеним функціоналом.

Результатом розділу є розроблений графічний клієнт обміну інформацією у мережах типу P2P за протоколом BitTorrent.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було проаналізовано та досліджено властивості обміну інформацією у мережах типу Peer-to-Peer та розроблено графічний клієнт, що працює за протоколом BitTorrent.

Для досягнення визначеної мети вирішено поставлені завдання:

– проаналізовано структуру формату даних Bencode, в якому закодовані файли метаданих та відповіді від трекеру;

– опрацьовано алгоритми обміну даними і взаємодії з трекером та пірами;

– розглянуто способи перевірки цілісності отриманих даних за допомогою хеш-функцій;

– виконано огляд та вибір засобів для розробки програмного забезпечення;

– реалізовано та проведено тестування застосунку.

Завдяки проведеному дослідженню предметної області було визначено особливості P2P-мереж та їх переваги над клієнт-серверною архітектурою у контексті передачі великого обсягу даних. Було опрацьовано специфікацію протоколу BitTorrent: структуру файлу метаданих, роботу з трекером та алгоритми взаємодії між пірами.

Відповідно до технічного завдання обрано найбільш придатні для розробки застосунку технології, шаблони проєктування, реалізовано логіку та графічний інтерфейс користувача. Проведено тестування застосунку та усунено знайдені помилки.

Результатом проведеної роботи є забезпечення обміну інформацією великих обсягів у мережах типу P2P з дотриманням і перевіркою цілісності за допомогою розробленого графічного клієнта, що працює за протоколом BitTorrent. Кінцевий продукт має низку переваг над розповсюдженими аналогами: відсутність реклами та інсталяторів небажаного ПЗ, розширений функціонал для відлагодження та можливість створення схем мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cachelogic. The true picture of peer-to-peer file sharing. 2005. URL: https://web.archive.org/web/20050308160805/http://www.cachelogic.com/research/slide9.php (Last accessed: 10.05.2024).

2. Cohen B. The BitTorrent Protocol Specification (BEP-0003). 2008. URL: https://www.bittorrent.org/beps/bep_0003.html (Last accessed: 10.05.2024).

3. Norberg A., Strigeus L., Hazel G. The BitTorrent Extension Protocol (BEP-0010). 2008. URL: https://www.bittorrent.org/beps/bep_0010.html (Last accessed: 10.05.2024).

4. Cohen B. The BitTorrent Protocol Specification v2 (BEP-0052). 2017. URL: https://www.bittorrent.org/beps/bep_0052.html (Last accessed: 10.05.2024).

5. TheoryOrg. BitTorrent Protocol Specification v1.0. URL: https://wiki.theory.org/BitTorrentSpecification (Last accessed: 10.05.2024).

6. Міхав В. В., Мелешко Є. В. Метод роботи рекомендаційної системи у комп'ютерній мережі типу peer to peer. *Системи управління, навігації та зв'язку* : зб. наук. праць. Полтава : ПНТУ, 2023. Т. 1 (71). С. 112–117. DOI: https://doi.org/10.26906/SUNZ.2023.1.112.

7. Куперштейн Л. М., Кренцін М. Д., Дудатьєв А. В., Каплун В. А. Аналіз проблем безпеки пірингових мереж. *Інформаційні технології та комп'ютерна інженерія.* 2022. № 2. С. 5–14. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/36237/115315.pdf?sequence=2&isAllowed=y (дата звернення: 10.05.2024).

8. Міхав В. В., Мелешко Є. В., Дрєєв, О. М., Лавданський А. О. Модель рекомендаційної системи для комп'ютерних мереж типу P2P. *Вісник Черкаського державного технологічного університету*, 2023. № 1. С. 52–60. DOI: 10.24025/2306-4412.1.2023.273495.

9. Кренцін М.Д., Куперштейн Л. М., Маліновський В. І. Аналіз методів підвищення захищеності пірингових мереж. *Інформаційні технології та*

комп`ютерне моделювання : матеріали Міжнар. наук.-практ. конф., м. Івано-Франківськ, 6–8 липня 2023 р. С. 135–137.

10. BitTorrent Foundation. BitTorrent (BTT) White Paper. 2019. URL: https://bittorrent.com/btt/btt-docs/BitTorrent_(BTT)_White_Paper_v0.8.7_Feb_2019.pdf (Last accessed: 10.05.2024).

11. Chen X., Jarvis A. Analysing BitTorrent's seeding strategies. *Institute of Electrical and Electronics Engineers.* 2009. DOI: 10.1109/CSE.2009.140.

12. Antal E., Vinkó T. On maximum throughput in BitTorrent. *Kecskeméti Főiskola.* 2016. URL: https://core.ac.uk/reader/328818228 (Last accessed: 10.05.2024).

13. Lei G., Songqing C., Zhen X., Enhua T., Xiaoning D., Xiaodong Z. A performance study of BitTorrent-like peer-to-peer systems. *Institute of Electrical and Electronics Engineers.* 2007. DOI: 10.1109/JSAC.2007.070116

14. MaxMind GeoIP. Geolocate an IP address using Databases. URL: https://dev.maxmind.com/geoip/geolocate-an-ip/databases (Last accessed: 28.05.2024).

15. TorrentFreak. uTorrent is the Most Used BitTorrent Client By Far. URL: https://torrentfreak.com/utorrent-is-the-most-used-bittorrent-client-by-far-200405/ (Last accessed: 30.05.2024).

16. InfoWorld. uTorrent installs cryptocurrency miner on user computers. URL: https://www.infoworld.com/article/2893808/utorrent-installs-cryptocurrency-miner-on-user-computers.html (Last accessed: 30.05.2024).

17. Wu D., Dhungel P., Hei X., Zhang C., Ross K. Understanding Peer Exchange in BitTorrent Systems. *Institute of Electrical and Electronics Engineers*. 2010. DOI: 10.1109/P2P.2010.5569967

18. Sharma P., Bhakuni A., Kaushal R. Performance analysis of BitTorrent protocol. *Indira Gandhi Institute of Technology*. 2013. DOI: 10.1109/NCC.2013.6488040

19. Guan D., Wang J., Zhang Y., Dong J. Understanding BitTorrent Download Performance. *East China Jiao Tong University*. *2008. DOI: 10.1109/ICN.2008.16*

20. Rai V., Sivasubramanian S., Bhulai S., Garbacki P., Van Steen M. A Multiphased Approach for Modeling and Analysis of the BitTorrent Protocol. *Vrije Universiteit.* 2007. DOI: 10.1109/ICDCS.2007.81

21. Chen X., Chu X., Liu J. A Lightweight Emulator for BitTorrent-Like File Sharing Systems. *Hong Kong Baptist University*. 2010. DOI: 10.1109/ICC.2010.5502796

22. Tian Y., Wu D., Ng K. Analyzing Multiple File Downloading in BitTorrent. *The Chinese University of Hong Kong*. 2006. DOI: 10.1109/ICPP.2006.23

23. Markakis E., Skiannis C., Sideris A., Palis E. Improving BitTorrent content delivery over broadcast networks. *Panepistemio Aigaiou.* 2014. DOI: 10.1109/TEMU.2014.6917745

24. Mazurczyk W., Kopiczko P. Understanding BitTorrent through real measurements. *Politechnika Warszawska*. 2013. DOI: 10.1109/CC.2013.6674215

25. Qi J., Zhang H., Ji Z., Yun L. Analyzing BitTorrent Traffic Across Large Network. *Harbin Institute of Technology*. DOI: 10.1109/CW.2008.150

ДОДАТОК А

Лістинги коду мовою програмування C#

DownloadPeersView.xaml

```
<ScrollViewer VerticalScrollBarVisibility="Visible"
HorizontalScrollBarVisibility="Visible">
    <Grid>
        <ItemsControl ItemsSource="{Binding Peers}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <Canvas Width="1024" Height="768" VerticalAlignment="Top"
                            HorizontalAlignment="Left" />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <Line X1="{Binding LineX1}" Y1="{Binding LineY1}"
                          X2="{Binding LineX2}" Y2="{Binding LineY2}"
                          Stroke="Black" StrokeThickness="0.5" />
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>

        <ItemsControl ItemsSource="{Binding Peers}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <Canvas />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>

            <ItemsControl.ItemTemplate>
                <DataTemplate DataType="{x:Type viewModels:PeerViewModel}">
                    <Grid>
                        <StackPanel>
                            <Image Width="80" Height="80"

Source="pack://application:,,,/Resources/cisco_pc.png" />
                            <TextBlock Text="{Binding IpAddress}"
TextAlignment="Center" />
                            <TextBlock Text="{Binding PeerId}"
TextAlignment="Center" />
                            <countryFlag:CountryFlag Code="{Binding CountryCode}"
Width="30"
                                                     Height="20"
                                                     Margin="5" />
                        </StackPanel>
                    </Grid>
                </DataTemplate>
            </ItemsControl.ItemTemplate>

            <ItemsControl.ItemContainerStyle>
                <Style TargetType="ContentPresenter">
                    <Setter Property="Canvas.Left"
                            Value="{Binding X}" />
                    <Setter Property="Canvas.Top"
                            Value="{Binding Y}" />
                </Style>
```

```
            </ItemsControl.ItemContainerStyle>
        </ItemsControl>

        <Canvas
            VerticalAlignment="Top"
            HorizontalAlignment="Left">
            <Image Width="80" Height="80"
                   Canvas.Left="432" Canvas.Top="304"
                   Source="pack://application:,,,/Resources/cisco_pc.png" />
        </Canvas>
    </Grid>

</ScrollViewer>
```

Bitfield.cs

```
public class Bitfield
{
    private byte[] Bytes { get; set; }

    public Bitfield(int pieceCount)
    {
        Bytes = new byte[pieceCount / 8];
    }

    public Bitfield(byte[] bytes)
    {
        Bytes = bytes;
    }

    public bool HasPiece(int index)
    {
        var byteIndex = index / 8;
        var offset = index % 8;
        return (Bytes[byteIndex] >> (7 - offset) & 1) != 0;
    }

    public void SetPiece(int index)
    {
        var byteIndex = index / 8;
        var offset = index % 8;
        Bytes[byteIndex] |= (byte)(1 << (7 - offset));
    }
}
```

BlockRequest.cs

```
public struct BlockRequest
{
    public int Piece { get; set; }
    public int BlockIndex { get; set; }

    public BlockRequest(int piece, int blockIndex)
    {
        Piece = piece;
        BlockIndex = blockIndex;
    }
}
```

Handshake.cs

```
public class Handshake
{
    private const byte IdentifierLength = 19;
    private const string Identifier = "BitTorrent protocol";

    public byte[] InfoHash { get; }
    public byte[] PeerId { get; }

    public Handshake(byte[] infoHash, byte[] peerId)
    {
        InfoHash = infoHash;
        PeerId = peerId;
    }

    public async Task WriteToStreamAsync(Stream stream)
    {
        stream.WriteByte(IdentifierLength);
        await stream.WriteAsync(Encoding.UTF8.GetBytes(Identifier).AsMemory(0,
19));
        await stream.WriteAsync(new byte[8]);
        await stream.WriteAsync(InfoHash);
        await stream.WriteAsync(PeerId);
    }

    public static async Task<Handshake> ReadFromStreamAsync(Stream stream)
    {
        var identifierLength = stream.ReadByte();
        if (identifierLength != IdentifierLength)
            throw new InvalidHandshakeException("Invalid protocol identifier
length.");

        var identifier = new byte[19];
        await stream.ReadExactlyAsync(identifier, 0, 19);
        if (!identifier.SequenceEqual(Encoding.UTF8.GetBytes(Identifier)))
            throw new InvalidHandshakeException("Invalid protocol identifier
header.");

        var extensions = new byte[8];
        var infoHash = new byte[20];
        var peerId = new byte[20];

        await stream.ReadExactlyAsync(extensions, 0, 8);
        await stream.ReadExactlyAsync(infoHash, 0, 20);
        await stream.ReadExactlyAsync(peerId, 0, 20);

        return new Handshake(infoHash, peerId);
    }
}
```

Message.cs

```
public class Message
{
    public byte[] Payload { get; set; }
    public MessageType Type => Payload.Length != 0 ? (MessageType)Payload[0] :
MessageType.KeepAlive;

    public Message(byte[] payload)
    {
        Payload = payload;
    }

    public async Task WriteToStreamAsync(Stream stream)
    {
        var lengthBuffer = new byte[4];
        BinaryPrimitives.WriteUInt32BigEndian(lengthBuffer, (uint)Payload.Length);
        await stream.WriteAsync(lengthBuffer);

        await stream.WriteAsync(Payload);
    }

    public static async Task<Message> ReadFromStreamAsync(Stream stream)
    {
        var lengthBuffer = new byte[4];
        await stream.ReadExactlyAsync(lengthBuffer, 0, 4);

        var length = BinaryPrimitives.ReadUInt32BigEndian(lengthBuffer);
        var messageBuffer = new byte[length];
        await stream.ReadExactlyAsync(messageBuffer, 0, (int)length);

        return new Message(messageBuffer);
    }
}
```

MessageType.cs

```
public enum MessageType : sbyte
{
    KeepAlive = -1,
    Choke = 0,
    Unchoke = 1,
    Interested = 2,
    NotInterested = 3,
    Have = 4,
    Bitfield = 5,
    Request = 6,
    Piece = 7,
    Cancel = 8
}
```

Peer.cs

```
public class Peer
{
    public IPEndPoint EndPoint { get; }
    public byte[] PeerId { get; set; }
    public Bitfield? Bitfield { get; set; }
    public bool Choked { get; set; } = true;
    public string DisplayName => $"{Encoding.UTF8.GetString(PeerId)}@{EndPoint}";

    public Peer(IPEndPoint endPoint, byte[] peerId)
    {
        EndPoint = endPoint;
        PeerId = peerId;
    }
}
```

Piece.cs

```
public class Piece
{
    public int Index { get; set; }
    public byte[] Hash { get; set; }
    public bool Downloaded { get; set; }

    public Piece(int index, byte[] hash)
    {
        Index = index;
        Hash = hash;
    }
}
```

PiecePayload.cs

```
public struct PiecePayload
{
    public uint Index { get; set; }
    public uint Begin { get; set; }
    public byte[] Data { get; set; }

    public PiecePayload(byte[] payload)
    {
        Index = BinaryPrimitives.ReadUInt32BigEndian(payload.AsSpan(1, 4));
        Begin = BinaryPrimitives.ReadUInt32BigEndian(payload.AsSpan(5, 4));
        Data = payload.AsSpan(9).ToArray();
    }
}
```

RequestPayload.cs

```
public class RequestPayload
{
    public uint Index { get; set; }
    public uint Begin { get; set; }
    public uint Length { get; set; }

    public RequestPayload(byte[] payload)
    {
        Index = BinaryPrimitives.ReadUInt32BigEndian(payload.AsSpan(1, 4));
        Begin = BinaryPrimitives.ReadUInt32BigEndian(payload.AsSpan(5, 4));
        Length = BinaryPrimitives.ReadUInt32BigEndian(payload.AsSpan(9, 4));
    }

    public RequestPayload(uint index, uint begin, uint length)
    {
        Index = index;
        Begin = begin;
        Length = length;
    }

    public byte[] AsBytes()
    {
        var payload = new byte[13];
        payload[0] = (byte)MessageType.Request;

        BinaryPrimitives.WriteUInt32BigEndian(payload.AsSpan(1, 4), Index);
        BinaryPrimitives.WriteUInt32BigEndian(payload.AsSpan(5, 4), Begin);
        BinaryPrimitives.WriteUInt32BigEndian(payload.AsSpan(9, 4), Length);
        return payload;
    }
}
```

TorrentFile.cs

```
public struct TorrentFile
{
    public string Announce { get; set; }
    public byte[] InfoHash { get; set; }
    public IList<byte[]> PieceHashes { get; set; }
    public long PieceLength { get; set; }
    public int Length { get; set; }
    public string Name { get; set; }
    public string? Comment { get; set; }
}
```