МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

До захисту допустити:
В.о. зав. кафедри

Ганна МАРТИНЮК

«04» червня 2025 р.

«ФАЙЛОВИЙ СЕРВЕР НА .NET»

Кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки»
Воліка Дмитра Олексійовича
Науковий керівник:
Мнацаканян Марія Сергіївна,
кандидат технічних наук, доцент кафедри системного аналізу та інформаційних технологій
Рецензент:
Охріменко Тетяна Олександрівна,
кандидат технічних наук, старший дослідник, заступник декана з наукової роботи факультету комп'ютерних наук та технологій Державного університету «Київського авіаційного університету»

Кваліфікаційна робота захищена
з оцінкою відмінно 90 (А)
Секретар ЕК

«11» червня 2025 р.

Київ– 2025

АНОТАЦІЯ

Дипломна бакалаврська робота за спеціальністю 122 – “Комп’ютерні науки” – Маріупольський державний університет, Київ 2025 рік.

В роботі досліджено та проаналізовано сучасні файлові сервери та їх аналоги, розглянуто технології та принцип роботи. Було проведено аналіз сучасного ринку та статистичне дослідження. Також було створено концепцію власного рішення та досліджено технології, необхідні для розробки.

У результаті даної роботи був спроектований та розроблений власний файловий сервер, призначений для корпоративних вимог.

Ключові слова: файловий сервер, хмаровий сервіс, база даних, інтеграція, адміністрування, синхронізація, адміністративна консоль.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології мають вирішальну роль у забезпеченні ефективної роботи підприємств та організацій. Одним із найважливіших елементів інформаційної інфраструктури будь-якої компанії є сервер – централізована система для зберігання, управління, обміну та захисту даних. Через постійне зростання обсягів цифрової інформації, питання організації ефективного та безпечного доступу до файлів стає дедалі актуальнішим.

Файловий сервер є важливим компонентом сучасних мереж, оскільки забезпечує централізоване зберігання даних, що дозволяє уникнути дублювання інформації, оптимізувати використання ресурсів та забезпечити контроль доступу до файлів. Використання файлового сервера дозволяє значно підвищити продуктивність співробітників, оскільки вони можуть швидко та зручно отримувати доступ до необхідних документів, незалежно від їх фізичного розташування.

Одним із ключових аспектів файлового сервера є забезпечення безпеки та конфіденційності даних. У сучасних умовах кіберзагроз питання захисту інформації набуває особливого значення. Важливою складовою функціонування файлового сервера є впровадження механізмів автентифікації користувачів, розмежування прав доступу, використання методів шифрування та резервного копіювання. Завдяки цим заходам забезпечується захист від несанкціонованого доступу, втрати чи пошкодження даних.

Файлові сервери можуть бути реалізовані на основі різних операційних систем, таких як Windows Server, Linux (наприклад, на основі Samba) та спеціалізованих NAS-пристроїв. Кожен із цих варіантів має переваги та недоліки, зумовлені потребами конкретної організації. Вибір файлового сервера залежить від таких факторів, як масштабність мережі, кількість користувачів, вимоги до безпеки та рівня доступності даних.

Окрім технічних аспектів, велику увагу слід приділити адміністративному управлінню файловим сервером. Це включає управління правами доступу, налаштування політик безпеки та забезпечення безперервності роботи сервера. Ефективне адміністрування дозволяє знизити ризик збоїв та забезпечити стабільну роботу системи.

Важливим аспектом використання файлових серверів є їх роль корпоративного документообігу. Завдяки централізованому зберігання файлів співробітники можуть працювати з документами в режимі реального часу, що спрощує процеси спільної роботи. Це особливо актуально для великих компаній, де доступ до загальних ресурсів є ключовою умовою ефективного виконання завдань.

Таким чином, дослідження теми файлових серверів є актуальним у контексті сучасних вимог до управління даними. У даній дипломній роботі буде розглянуто принципи побудови файлових серверів, їх функціональні можливості, основні методи забезпечення безпеки та адміністрування. Особлива увага приділятиметься вибору оптимального рішення для різних сценаріїв використання, а також порівняльному аналізу доступних технологій. Також, буде описаний поетапний процес проектування та розробки власного файлового серверу відповідно до сучасних вимог ринку.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд технології

Файловий сервер – це, як правило, центральний сервер у комп'ютерній мережі, який забезпечує підключення користувачів до мережевої системи зберігання даних. Цей термін може означати як устаткування, і програмне забезпечення, необхідне виконання функцій файлового сервера.

Користувачі, отримавши необхідні права доступу до певних файлів в мережевий системі зберігання даних, можуть їх відкривати і редагувати, і навіть видаляти файли і папки так само, якби вони працювали на власному комп'ютері.

На файловому сервері кожному авторизованому користувачеві надається певний простір для зберігання робочих файлів. Інші користувачі можуть також їх відкривати, читати та редагувати відповідно до їх прав доступу. Ці права встановлюються адміністратором файлового сервера. Саме він визначає, хто які файли та в яких папках може відкривати та переглядати, а також (якщо це дозволено) редагувати, видаляти чи додавати нові файли.

Крім того, файловий сервер може мати підключення до інтернету, і, при відповідній конфігурації прав доступу, користувачі можуть отримувати доступ до інтернет-ресурсів, якщо доступ до них дозволений мережевим адміністратором. У деяких організаціях може адміністративно встановлюватися заборона доступу до певних ресурсів за тими чи іншим критеріям. Наприклад, може бути закритий доступ до відеохостингу Youtube, сайтів з розважальним контентом тощо. Крім того, підключення файлового сервера до інтернету забезпечує віддалений доступ користувачів до своїх папок на файловому сервері, якщо вони знаходяться не на робочому місці.

Для файлового сервера можуть підійти будь-які сучасні операційні системи Windows, Linux або MacOS, хоча треба мати на увазі, що мережні пристрої повинні бути з ними сумісні.

Також треба взяти до уваги, що файлові сервери часто використовуються не тільки для зберігання та обробки файлів, але також як репозиторій для програм, які доступні користувачам корпоративної мережі, а також як сервер резервування.

1.2 Принцип роботи

Файловий сервер надає спільні диски, до яких можуть отримати доступ робочі станції в мережі. Диск містить комп’ютерні файли, такі як текстові документи, зображення, аудіо- та відеофайли тощо. Файлові сервери зазвичай використовуються для зберігання. Отже, вони не запускають комп’ютерних програм і не виконують обчислювальних завдань [1].

Файлові сервери використовують різні протоколи для доступу до файлів залежно від обчислювального середовища. Наприклад, файлові сервери в Інтернеті доступні через протоколи FTP і HTTP. Доступ до файлових серверів у локальній мережі, наприклад тих, що використовуються в офісах і школах, здійснюється за допомогою протоколу SMB або NFS [1].

Для надійної роботи файлового сервера необхідно вибрати відповідне обладнання. Це перш за все процесор достатньої потужності для обслуговування заданої кількості користувачів, а також дискові накопичувачі, які мають ємність, достатню для розміщення необхідних програм та операційної системи та іншого програмного забезпечення для обслуговування користувачів корпоративної мережі. Важливе значення для швидкодії системи має обсяг оперативної пам'яті (ОЗП), де розміщуються модулі запущених до роботи програм. Якщо обсяг ОЗП буде недостатній, то робота системи сильно уповільниться, і не допоможе навіть найпотужніший і найшвидший процесор.

Визначальним чинником вибору параметрів файлового сервера є кількість користувачів корпоративної мережі. Для зв'язку користувачів з файловим сервером використовуються спеціальні протоколи, наприклад протокол SMB

(Server Message Block) розроблений IBM. Він може використовуватися в локальних мережах як на пристрої Windows, так і macOS. Як протокол мережної операційної системи часто використовується NFS (Network File System). Якщо файловий сервер працює під ОС Unix, то щоб поєднати обидва типи протоколів в одній мережі як клієнти, так і файлові сервери, повинні бути оснащені програмами, які дозволяють виконувати протокол SMB в цих системах. Це може бути, наприклад, програмна платформа Samba.

Для отримання віддаленого доступу до файлового сервера зазвичай використовується традиційний протокол File Transfer Protocol (FTP) або його захищений варіант SFTP (Secure FTP). Крім того, може використовуватися протокол безпечного копіювання SCP (Secure Copy) та WebDAV (Web Distributed Authoring and Versioning) – набір розширень та додатків до протоколу HTTP. WebDAV дозволяє змінювати властивості об'єктів, що зберігаються на сервері, шукати файл за властивостями, блокувати файл для редагування одним користувачем, керувати версіями файлів, а також керувати доступом до файлів на основі списків.

1.3 Переваги використання файлового серверу

Для багатьох компаній вирішальним критерієм під час використання файлового сервера у корпоративній мережі є можливість централізованого управління та розмежування прав доступу між користувачами різних підрозділів. Крім того, легко можна забезпечити можливість колективної роботи над документами, виключивши проблему використання різних версій одного документа різними користувачами.

Інша перевага файлового сервера – це усунення ресурсних обмежень для користувачів. За винятком особистих файлів, всі робочі документи та їх резервні копії можуть бути розміщені на загальному сервері. При правильній організації

структури папок та директорій користувачі отримують однакове подання всіх доступних документів в організації відповідно до своїх прав доступу.

Якщо файловий сервер налаштований для роботи через інтернет, то файли також доступні для віддаленої роботи, як і при роботі в локальній мережі. Але, на відміну від хмарного рішення, компанія продовжує зберігати контроль за файлами та їх безпекою. Це очевидна перевага перед сторонніми рішеннями щодо зберігання корпоративної інформації.

1.4 Типи файлових серверів

Файлові сервери класифікуються за різними ознаками, зокрема за типом операційної системи, що використовується, та за протоколами доступу до файлів.

Сервери на базі Windows.

Файлові сервери, що працюють на операційній системі *Windows Server*, широко використовуються в корпоративних середовищах. Вони підтримують файлову систему *NTFS* (*New Technology File System*), яка забезпечує розширені можливості, такі як:

- Безпека та дозволи: *NTFS* надає можливість встановлювати детальні дозволи на рівні файлів і папок, що забезпечує контроль доступу для користувачів та груп.

- Журналювання: Функція журналювання допомагає відстежувати зміни у файловій системі, що сприяє швидкому відновленню після збоїв.

- Підтримка великих файлів та розділів: *NTFS* підтримує роботу з великими файлами та розділами, що робить її придатною для сучасних потреб зберігання даних.

Недоліком *NTFS* є обмежена сумісність з іншими операційними системами, такими як *Linux* та *macOS*, без використання додаткового програмного забезпечення.

Сервери на базі Linux.

Файлові сервери на базі *Linux* часто використовують файлові системи, такі як *ext4, XFS* або *Btrfs*. Ці файлові системи мають свої особливості:

- *ext4*: Це одна з найпоширеніших файлових систем у *Linux*, відома своєю стабільністю та продуктивністю. Вона підтримує великі обсяги даних та має функцію журналювання, що зменшує ризик пошкодження даних під час збоїв.

- *XFS*: Файлова система, оптимізована для роботи з великими файлами та високою швидкістю введення-виведення. Вона підходить для серверів, які обробляють великі обсяги даних.

- *Btrfs*: Нова файлова система, яка пропонує розширені можливості, такі як знімки (snapshots), стиснення даних та самовідновлення.

Перевагою файлових серверів на базі *Linux* є їхня гнучкість, масштабованість та відкритий вихідний код, що дозволяє налаштовувати систему відповідно до специфічних потреб організації.

NAS (Network Attached Storage).

NAS — це спеціалізовані пристрої, призначені для зберігання даних та надання доступу до них через мережу. Вони часто використовуються в малих та середніх підприємствах, а також у домашніх умовах для централізованого зберігання даних. *NAS*-пристрої зазвичай підтримують різні протоколи доступу, такі як SMB/CIFS для Windows та NFS для Unix/Linux систем.

Переваги NAS:

- простота використання: NAS-пристрої зазвичай мають зручний веб-інтерфейс для налаштування та управління, що робить їх доступними навіть для користувачів без глибоких технічних знань.

- масштабованість: Багато NAS-пристроїв дозволяють додавати додаткові жорсткі диски або розширювати обсяг зберігання за потреби.

- енергоефективність: NAS споживають менше енергії порівняно з повноцінними серверами, що робить їх економічно вигідними для тривалого використання.

Недоліки NAS:

- обмежена продуктивність: Через використання спільних мережевих ресурсів швидкість доступу до даних може бути нижчою порівняно з локальними дисками або спеціалізованими серверами.

- обмежена функціональність: NAS-пристрої зазвичай мають фіксований набір функцій, що може бути недостатнім для великих організацій з комплексними вимогами.

1.5 Аналоги файлових серверів

У сучасному світі існує безліч альтернатив традиційним файловим серверам, які забезпечують зберігання та доступ до даних.

1. Хмарні сховища.

Хмарні сервіси надають можливість зберігати дані на віддалених серверах, доступ до яких здійснюється через інтернет. Це дозволяє користувачам отримувати доступ до своїх файлів з будь-якого пристрою та місця.

- *Google Drive*: надає 15 ГБ безкоштовного простору з можливістю розширення. Інтегрується з іншими сервісами Google, такими як Docs, Sheets та Slides, що спрощує спільну роботу над документами.

- *Dropbox*: відомий своєю простотою використання та швидкою синхронізацією файлів між пристроями. Підтримує функції спільного доступу та коментування файлів.

- *Microsoft OneDrive*: інтегрується з пакетом Office 365, що дозволяє редагувати документи Word, Excel та PowerPoint безпосередньо в хмарі. Надає 5 ГБ безкоштовного простору з можливістю розширення.

2. Мережеві сховища даних (NAS).

- *NAS-пристрої* підключаються до локальної мережі та надають спільний доступ до даних для всіх підключених користувачів. Вони часто використовуються в малих та середніх підприємствах для централізованого зберігання даних.

- *Synology*: Відомий виробник NAS-пристроїв, які підтримують різні функції, такі як резервне копіювання, медіа-сервер та навіть відеоспостереження.

- *QNAP*: Пропонує потужні NAS-рішення з підтримкою віртуалізації, додатків та розширених функцій безпеки.

3. Децентралізовані системи зберігання.

Ці системи використовують технологію блокчейн для розподіленого зберігання даних, забезпечуючи безпеку та стійкість до цензури.

- *BlockHouse*: Децентралізована система зберігання, яка базується на приватних блокчейнах. Кожен учасник може надавати свій невикористаний простір для зберігання даних інших користувачів, використовуючи смарт-контракти для автоматичної перевірки доступності файлів.

4. Спеціалізовані файлові системи.

Деякі організації використовують спеціалізовані файлові системи для управління великими обсягами даних або специфічними типами даних.

- *Ceph*: Розподілена файловая система, яка забезпечує високу доступність та масштабованість. Використовується в великих дата-центрах та хмарних середовищах для зберігання об'єктів, блоків та файлових даних.

- *GlusterFS*: Масштабована розподілена файловая система, яка дозволяє об'єднувати сховища з різних серверів в єдину файлову систему. Підходить для обробки великих обсягів даних та забезпечує високу доступність.

5. Програмні рішення для спільного доступу до файлів.

Існують програмні платформи, які дозволяють організувати спільний доступ до файлів без необхідності використання окремого файлового сервера:

- *Nextcloud*: відкрите програмне забезпечення для створення власного хмарного сховища. Забезпечує функції спільного доступу до файлів, календарів, контактів та інших даних.

- *ownCloud*: схоже на *Nextcloud* рішення, яке дозволяє створювати приватне хмарне сховище з можливістю синхронізації та спільного доступу до файлів.

Висновки

У розділі розглянуто принципи роботи файлових серверів, їх основні функції та роль у корпоративних мережах. Визначено, що файловий сервер забезпечує централізоване зберігання та керування даними, розмежування прав доступу та можливість спільної роботи над документами. Проаналізовано різні типи серверів — на базі Windows, Linux та NAS-пристроїв, кожен із яких має свої переваги та обмеження. Також розглянуто альтернативи, серед яких основними є хмарні рішення. Приділена увага правильному вибору обладнання та протоколів для забезпечення надійності, безпеки та ефективності роботи серверу в корпоративному середовищі.

2. ДОСЛІДЖЕННЯ ФАЙЛОВИХ СЕРВЕРІВ

2.1. Дослідження ринку

У сучасному світі управління даними є критично важливим аспектом діяльності будь-якої організації. Файлові сервери та їх аналоги, такі як хмарні сховища, забезпечують зберігання, обробку та доступ до інформації. У цьому розділі ми дослідимо поточний стан ринку файлових серверів та їх альтернатив, проаналізуємо попит на ці рішення та розглянемо, які продукти використовуються різними компаніями для виконання специфічних завдань.

Ринок файлових серверів та хмарних сховищ продовжує активно розвиватися, адаптуючись до змінних потреб бізнесу та технологічних трендів.

Згідно з даними, у 2024 році спостерігалося зростання впровадження хмарних рішень, штучного інтелекту та обробки великих даних в Україні. Українські компанії активно впроваджують AI-рішення для автоматизації бізнес-процесів, аналізу великих обсягів даних і створення нових продуктів, що свідчить про динамічний розвиток IT-галузі в країні [18].

Попит на файлові сервери та хмарні сховища залежить від розміру компанії, галузі та специфічних потреб бізнесу. Малі та середні підприємства часто обирають хмарні сховища через їхню гнучкість, масштабованість та відсутність необхідності в значних початкових інвестиціях. Великі корпорації, зокрема ті, що працюють з конфіденційними даними, можуть віддавати перевагу власним файловим серверам або приватним хмарним рішенням для забезпечення більшого контролю та безпеки.

За даними звіту Cion, близько 60% корпоративних даних зберігається у хмарі. При цьому під хмарою розуміються як SaaS-рішення, що перераховані вище, так і системи, які початково орієнтовані на корпоративний сектор. Останньому часто потрібні надійніші інструменти, ніж може запропонувати масовий сервіс зберігання, наприклад, коли йдеться про зберігання резервних копій. Регулярне створення бекапів за розкладом дозволяє компанії завжди мати під рукою

необхідні дані на той випадок, якщо їх вкрадуть, пошкодять, не зможуть отримати доступ або на поточні ІТ-системи буде здійснена кібератака [8].

2.2. Популярні рішення на ринку

1. Dropbox.

Історія та розвиток:

Dropbox був заснований у 2007 році і швидко здобув популярність завдяки простоті використання та інтуїтивному інтерфейсу. У своїй ранній стадії сервіс позиціонувався як інноваційне рішення для синхронізації файлів між пристроями, що стало вирішальним чинником для популярності серед приватних користувачів та професіоналів.

Технічна реалізація та функціонал:

Dropbox забезпечує автоматичну синхронізацію файлів, ведення історії версій, спільну роботу в режимі реального часу, а також інтеграцію з численними сторонніми додатками й сервісами (наприклад, з пакетом Microsoft Office). Докладна реалізація механізму синхронізації дозволяє ефективно вирішувати конфлікти файлів і забезпечувати безперебійний доступ до даних з різних платформ (Windows, macOS, Linux, мобільні ОС).

Безпека та шифрування:

Dropbox використовує сучасні методи шифрування даних – як під час передачі (TLS/SSL), так і для зберігання (AES-256), що забезпечує високий рівень захищеності інформації. Проте певна критика стосується архітектури безпеки, оскільки ключі розшифрування керуються сервісом, що може становити ризик для користувачів з особливою потребою в конфіденційності.

Бізнес-модель та ринкова орієнтація:

З точки зору економічної моделі, *Dropbox* застосовує модель freemium – безкоштовний план з обмеженим простором (зазвичай 2 ГБ, що легко можна збільшити за рахунок запрошення знайомих) та низку платних тарифів для індивідуальних та корпоративних клієнтів. Такий підхід дозволяє залучати

широку аудиторію, при цьому забезпечуючи ефективну монетизацію через корпоративні рішення та преміум-функції.

2. Google Диск

Екосистема та інтеграція:

Google Диск є частиною величезної екосистеми *Google*, що включає Gmail, Google Документи, Таблиці, Презентації, тощо. Він значно полегшує спільну роботу завдяки можливості миттєвої редагування документів у реальному часі. Глибока інтеграція стимулює використання сервісу як для навчальних закладів, так і для бізнесу.

Функціональні можливості:

Окрім стандартного зберігання, *Google Диск* надає можливості для автоматичного резервного копіювання фотографій і важливих файлів, а також підтримує пошук за допомогою потужних алгоритмів *Google*, що значно полегшує орієнтування і доступ до потрібного контенту. Розширені функції управління доступом дозволяють налаштовувати права для окремих користувачів, що є важливим для корпоративних користувачів.

Безкоштовний доступ та платні опції:

Безкоштовний обсяг пам’яті, який складає 15 ГБ, а також можливість придбання додаткового простору в рамках сервісу *Google* One, забезпечують гнучке масштабування — від приватного використання до великих корпоративних конфігурацій. Такий підхід дозволяє *Google* Диску бути конкурентоспроможним як серед приватних користувачів, так і на ринку корпоративних рішень.

Аналітична оцінка:

Завдяки інтеграції з іншими продуктами *Google*, сервіс є прикладом ефективного використання синергій в екосистемі, що сприяє зниженню витрат на управління даними та підвищенню продуктивності роботи.

3. Microsoft OneDrive

Інтеграція з продуктами Microsoft:

OneDrive є невід'ємною частиною екосистеми Microsoft, інтегрованою як з Windows, так і з пакетом Microsoft 365. Це дозволяє забезпечити широкі можливості для спільної роботи, інтегруючи робочі документи, електронну пошту та календар у єдиному середовищі.

Технічний аспект та зручність використання:

Функціонал *OneDrive* включає синхронізацію даних між пристроями, спільне редагування документів через Office Online, а також можливість автономного доступу до файлів. Завдяки усуненню перешкод між різними платформами, *OneDrive* підходить як для приватних осіб, так і для великих організацій, які користуються комплексними рішеннями від Microsoft.

Безпека та управління доступом:

Сервіс використовує сучасні алгоритми шифрування та дотримується стандартів корпоративної безпеки. Особлива увага приділяється інтеграції з Active Directory, що дозволяє корпоративним клієнтам ефективно управляти правами доступу і забезпечувати відповідність стандартам безпеки.

Комерційна орієнтація:

Однією з ключових переваг *OneDrive* є його інтеграція з бізнес-інструментами, що дозволяє організаціям централізовано управляти файлами, робочими процесами та співпрацею. Це робить сервіс надзвичайно привабливим для компаній, які вже інвестували в інфраструктуру Microsoft.

4. iCloud Drive

Інтеграція в екосистему Apple:

iCloud Drive є основним рішенням для користувачів пристроїв Apple, оскільки інтегрується з macOS, iOS та іншими продуктами Apple. Завдяки цій інтеграції доступ до даних здійснюється з максимальною зручністю, що дозволяє безперебійно синхронізувати фото, документи та налаштування між пристроями.

Підхід до безпеки:

Apple приділяє значну увагу приватності користувачів, використовуючи як при передачі, так і при зберіганні даних високотехнологічні методи шифрування.

Це особливо важливо з огляду на сучасні вимоги до захисту персональної інформації. Також сервіс підтримує функції резервного копіювання, що допомагає користувачам відновлювати дані у разі втрати.

Функціональні можливості:

iCloud Drive надзвичайно зручний при роботі з файлами, що забезпечено оптимізованими алгоритмами синхронізації та доступом до файлів безпосередньо з Finder або iOS-файлової системи. Така функціональна інтеграція сприяє створенню безшовного користувацького досвіду.

Ринкова орієнтація:

Призначений переважно для екосистеми Apple, *iCloud Drive* задовольняє потреби як роздрібних користувачів, так і професіоналів, що працюють у творчих або креативних сферах. Платіжна система, заснована на додаткових планах (від 50 ГБ до 2 ТБ), дозволяє гнучко масштабувати простір зберігання відповідно до потреб користувача.

5. Mega

Позиціонування на ринку:

Mega відомий як сервіс, орієнтований на забезпечення високого рівня конфіденційності та безпеки. У порівнянні з традиційними платформами, *Mega* пропонує великий обсяг безкоштовного простору з 20 ГБ, що є привабливим чинником для користувачів, які шукають альтернативні рішення.

Безпека та шифрування:

Основною відмінністю *Mega* є акцент на шифрування даних на стороні клієнта. Завдяки цьому навіть адміністратори платформи не мають доступу до розшифрованих даних, що забезпечує принцип «zero-knowledge». Така архітектурна модель стає особливо актуальною для осіб та організацій, що потребують високого рівня конфіденційності.

Функціональні можливості та недоліки:

Mega дозволяє користувачам зберігати, синхронізувати та обмінюватися великими файлами. Попри це, інтерфейс сервісу може здатися менш інтуїтивно

зрозумілим порівняно з іншими ринковими лідерами, що може стати перешкодою для деяких категорій користувачів. Крім того, критики відзначають певні питання сумісності та швидкості роботи при передачі файлів.

Бізнес-модель:

Платні варіанти *Mega* надають можливість значного розширення обсягу даних до 16 ТБ, що робить сервіс конкурентним вибором для професійних користувачів, які потребують великих обсягів сховища при високих стандартах безпеки.

6. pCloud

Інноваційний підхід до оплати:

pCloud відзначається серед конкурентів завдяки впровадженню довічних тарифних планів, що дозволяють користувачам здійснювати одноразову покупку простору для зберігання даних, уникаючи повторних платежів. Це робить сервіс привабливим для тих, хто шукає довгострокові рішення.

Архітектура та функціональні особливості:

Сервіс має сучасний інтерфейс, оптимізований для різних пристроїв, та підтримує функцію автоматичної синхронізації. *pCloud* також дозволяє інтеграцію з соціальними мережами й іншими хмарними сервісами, що розширює можливості обміну інформацією. Окремо слід відзначити функцію *pCloud* Crypto, яка забезпечує клієнтське шифрування папок із критичними даними.

Безпека:

pCloud використовує надійні методи шифрування, гарантуючи, що дані користувачів захищені як при передачі, так і на сервері. Доповненням до цього є можливість зберігати файли в зашифрованому вигляді, що є суттєвою перевагою при роботі з чутливою інформацією.

Цільова аудиторія та перспективи:

Завдяки довічній оплаті та сучасним функціям, *pCloud* орієнтується як на окремих користувачів, так і на малі підприємства. Це створює перспективи

гармонійного поєднання економічної ефективності та високих стандартів безпеки.

7. iDrive

Орієнтація на резервне копіювання:

iDrive позиціонується не лише як засіб зберігання даних, а і як комплексне рішення для резервного копіювання. Сервіс підтримує резервне копіювання даних з різних пристроїв, включаючи комп'ютери, мобільні пристрої та сервери, що робить його привабливим для користувачів із різних сегментів ринку.

Функціональні можливості:

Основними перевагами *iDrive* є можливості автоматичного резервного копіювання, підтримка версіонування даних та комплексне управління кількома пристроями з однієї консолі. Ці функції допомагають зменшити ризик втрати важливої інформації та забезпечують зручність у відновленні даних у разі збою обладнання або помилок користувача.

Безпека:

iDrive використовує високоякісні методи шифрування для передачі і зберігання даних. На додаток до цього, платформа пропонує можливості для налаштування параметрів резервного копіювання, що дозволяє генерувати специфічні політики безпеки, орієнтовані на потреби конкретних користувачів чи організацій.

Ринкова орієнтація та бізнес-модель:

Завдяки своїй спеціалізації на резервному копіюванні, *iDrive* приваблює особистих користувачів з високим рівнем потреб у захисті даних та малий бізнес, де важливо забезпечити архівування корпоративної інформації. Модель платних підписок з різними рівнями доступу дозволяє каналізувати інвестиції в додаткові функції для корпоративних клієнтів.

8. SpiderOak

Фокус на конфіденційність:

SpiderOak відомий своєю принциповою «No Knowledge» (без знань) політикою. Це означає, що компанія не має доступу до розшифровуючих ключів користувачів. Такий підхід є критично важливим для організацій та приватних осіб, що надають пріоритет максимальній конфіденційності інформації.

Технічні характеристика та функції:

Сервіс забезпечує автоматичне резервне копіювання, синхронізацію файлів і можливості спільного використання даних, при цьому застосовуючи високорівневе шифрування. Особливу увагу варто приділити тому, що навіть у випадку компрометації серверів, користувачі залишаються єдиними власниками ключів розшифровки, що мінімізує ризики витоку даних.

Переваги та обмеження:

Незважаючи на високу ступінь безпеки, інтерфейс *SpiderOak* може здаватися менш інтуїтивно зрозумілим у порівнянні з більш масовими рішеннями, такими як *Dropbox* чи *Google* Диск. Це часто обумовлює орієнтацію сервісу на певну нішу – тих користувачів, які готові поступитися зручністю заради підвищеної приватності.

Цільова аудиторія:

SpiderOak знаходить застосування переважно серед користувачів, які працюють з конфіденційними даними, юристів, журналістів та інших професіоналів, для яких безпека та конфіденційність мають першорядне значення.

9. MediaFire

Орієнтація на спрощений обмін файлами:

MediaFire розроблений з акцентом на простоті використання та швидкому обміні даними між користувачами. Сервіс став популярним завдяки можливості миттєво завантажувати та поширювати файли за допомогою прямого посилання.

Функціональні можливості:

Основний інструментарій *MediaFire* полягає в забезпеченні простого інтерфейсу для завантаження, організації та обміну файлами. Невелика кількість

додаткових функцій порівняно з іншими хмарними сервісами може бути перевагою для користувачів які цінують швидкість та зручність при обміні великими файлами.

Бізнес-модель та безпека:

Безкоштовний план надає обмежений обсяг пам’яті (близько 10 ГБ), що достатній для легкого обміну медіа та документами, проте деякі питання стосовно безпеки зберігання даних можуть залишатися відкритими. Платні варіанти розширюють можливості зберігання і забезпечують кращий контроль доступу, що розглядається як необхідність для корпоративного використання.

Потенціал розвитку:

Враховуючи фокус на швидкому обміні контентом, *MediaFire* продовжує залишатися привабливим рішенням для користувачів, для яких зручність розповсюдження файлів переважує над комплексними функціями управління даними.

10. Box

Позиціонування для корпоративного сегмента:

Box є рішенням, орієнтованим на корпоративний ринок, де основна увага приділяється не лише зберіганню даних, а й інтеграції в бізнес-процеси, управлінню співпрацею та забезпеченню відповідності нормативним вимогам.

Технічка інтеграція та функціональні можливості:

Box пропонує потужні інструменти для управління документами, розширені можливості спільної роботи, а також інтеграцію з різноманітними корпоративними додатками, такими як *Salesforce, Slack, Microsoft Office* тощо. Завдяки цьому сервіс адаптований до умов сучасного бізнес-середовища, де важливо мати універсальний інструмент для організації робочих процесів.

Безпека та управління доступом:

Box підтримує багаторівневі механізми безпеки, включаючи шифрування даних, аутентифікацію двофакторного доступу та детальний аудит активності користувачів. Для великих організацій окремо розроблені політики управління

даними, що відповідають вимогам щодо відповідності (compliance) з нормативними стандартами.

Бізнес-модель:

Більш розширені платформи та корпоративні рішення стимулюють підвищений інтерес серед корпоративних клієнтів, для яких важливо мати централізований контроль та цілісний підхід до управління інформацією. *Box* демонструє приклад успішного синтезу корпоративних стандартів і сучасних тенденцій ринку.

2.3. Загальні спостереження та висновки

У сучасному ринковому просторі хмарного зберігання даних спостерігається тенденція до спеціалізації:

- *Dropbox, Google* Диск та *OneDrive* захоплюють масовий ринок завдяки інтуїтивності, широкій функціональності та інтеграції з іншими популярними продуктами.

- *iCloud Drive* зосереджений на користувачах екосистеми *Apple*, де зручність та інтегрованість мають головне значення.

- *Mega* та *SpiderOak* адресують сегмент користувачів із підвищеними вимогами до безпеки та конфіденційності, що є критично важливими в епоху зростаючих кібератак.

- *pCloud* та *iDrive* комбінують зручність синхронізації із специфічними рішеннями для резервного копіювання, пропонуючи гнучкі тарифи та інноваційні платіжні моделі (наприклад, довічні підписки в *pCloud*).

- *MediaFire* обирається за простоту та економію часу при спільному доступі до файлів, а *Box* залишається популярним серед корпоративних клієнтів, для яких важливі інтеграція з бізнес-процесами, безпека та відповідність законодавчим нормам.

Вибір конкретного сервісу залежить від багатьох факторів: обсягу зберігання, рівня безпеки, цільової аудиторії, інтерфейсу користувача та

інтеграції з іншими інструментами. При порівнянні цих сервісів важливо враховувати як технічні характеристики, так і економічні аспекти, враховуючи постійно змінюваний ландшафт ринку хмарних технологій. Кожне з рішень має свої унікальні переваги, що дозволяють користувачам вибирати оптимальний варіант відповідно до індивідуальних чи корпоративних потреб.

2.4. Статистичне дослідження

Кількість даних збережена у хмарових сховищах.

До 2025 року в хмарі буде зберігатися понад 100 зетабайт даних. Цей факт свідчить про зростання хмарних обчислень. Для чіткішої картини один зетабайт дорівнює мільярду терабайтів і трильйону гігабайтів. Це досить приголомшлива сума. Щоб поглянути на ситуацію в перспективі, до 2025 року обсяг збережених даних у всьому світі досягне 200 зетабайт. Це означає, що половина загальної кількості даних у всьому світі буде в хмарі. Для порівняння, у 2015 році 25% обчислювальних даних було в хмарі [9].

Міграція робочого навантаження в хмару в європейських організаціях.

Згідно зі статистикою переходу на хмару, це ініціатива номер один у порядку денному європейських організацій, які ще не перенесли свою діяльність у хмару. Далі йде оптимізація використання хмари для скорочення витрат (59%) і сприяння стратегії використання хмари (50%) [9]. Візуальний графік на рисунку 2.3.1

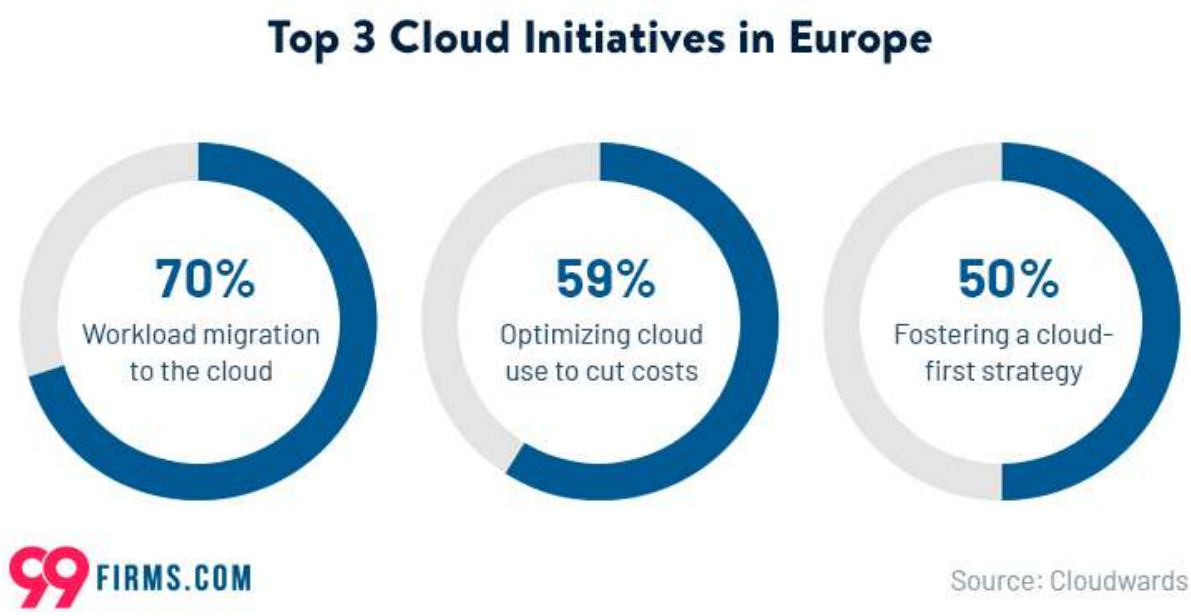

Рисунок 2.3.2. Візуальний графік [9]

Робочі навантаження у хмаровому середовищі.

Мережевий гігант Cisco повідомив, що до 2021 року більшість виділених серверів буде меншістю завдяки зростанню хмарних обчислень. Статистика їх Global Cloud Index показує, що 94% робочих навантажень відбуватимуться в публічних і приватних хмарах [9].

Найбільш популярні сервіси.

Google Drive домінує на світовому ринку хмарних сховищ за кількістю користувачів. Статистика хмарних обчислень показує, що Dropbox є наступним у списку з 66,2%. OneDrive займає третє місце (39,35%), за ним іде iCloud (38,89%). Деякі менш відомі сервіси MEGA (5,09%), Box (4,17%) і pCloud (1,39%) [9]. Візуальний графік на рисунку 2.3.2

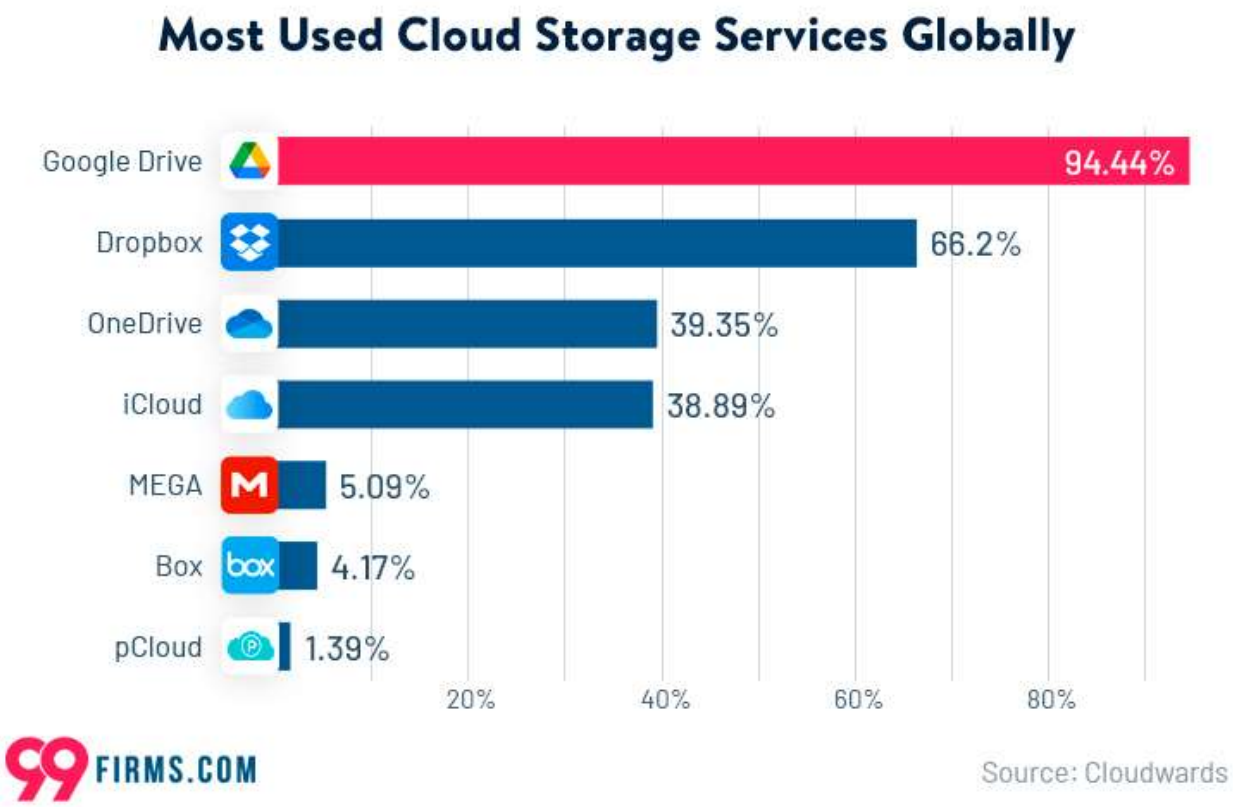

Рисунок 2.3.2. Візуальний графік [9]

Причини високої популярності хмарових рішень.

Є багато причин, чому компанії тяжіють до хмари. Серед багатьох головним є, безсумнівно, підвищення швидкості. Далі йдуть більша гнучкість (63%) і покращена підтримка клієнтів (57%). Як правило, організації з понад 1000 співробітників прагнуть до гнучкості та зниження витрат, тоді як менші організації шукають безперервності бізнесу [9]. Візуальний графік на рисунку 2.3.3

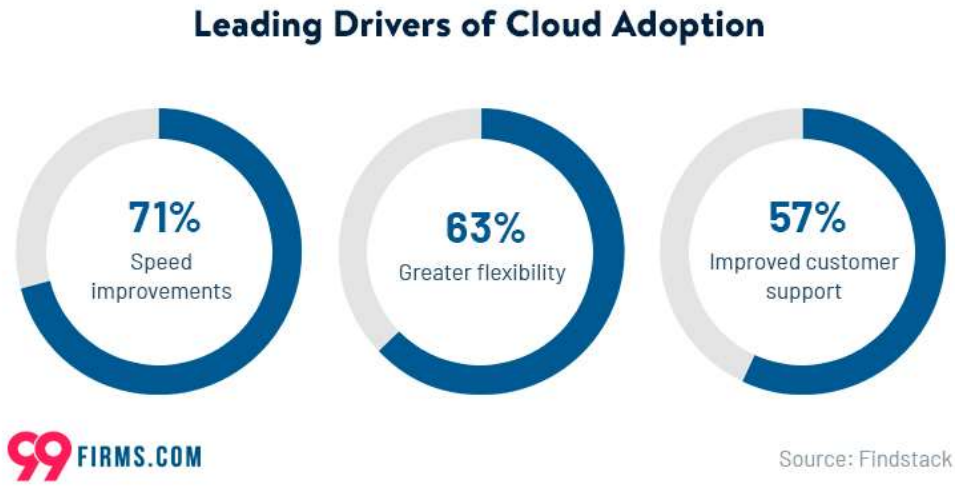

Рисунок 2.3.3. Візуальний графік [9]

2.5. Концепція власного серверного рішення

У ході проведеного дослідження було детально проаналізовано сучасний ринок хмарних сховищ та файлових серверів, їх переваги, недоліки та особливості використання різними компаніями. Зібрані дані свідчать про те, що, хоча існуючі хмарні сервіси пропонують широкий функціонал і гнучкість у зберіганні даних, вони не завжди відповідають специфічним вимогам корпоративного сектору.

Ключові висновки дослідження:

- Проблеми конфіденційності та безпеки:

Багато компаній відмовляються від використання комерційних хмарних рішень через ризик витоку інформації або недостатній рівень контролю над даними. Великі корпорації та державні установи потребують локальних рішень або приватних хмар, що дозволяють зберігати конфіденційні дані у межах власної інфраструктури.

- Обмеження у кастомізації та інтеграції:

Хоча такі сервіси, як Dropbox, Google Диск або Microsoft OneDrive, мають високу популярність, вони не завжди дозволяють повноцінно інтегруватися з корпоративними інформаційними системами. Вбудовані інструменти часто не

враховують специфічні потреби підприємств, що працюють у різних сферах (фінанси, медицина, виробництво тощо).

- Висока вартість масштабування:

Використання комерційних рішень для великих компаній може бути дорогим через необхідність оплачувати місце у хмарному сховищі за кожного користувача. У довгостроковій перспективі створення власного серверного рішення є фінансово вигіднішим варіантом.

- Проблеми продуктивності при роботі з великими обсягами даних

Деякі компанії працюють із великими файлами (відеопродукція, архітектурні проекти, бази даних), і робота з такими даними через публічні хмарні сервіси може викликати затримки та зниження швидкості доступу.

- Юридичні та регуляторні обмеження:

У певних країнах існують законодавчі вимоги щодо зберігання даних лише на локальних серверах або у визначених дата-центрах. Це унеможливлює використання деяких міжнародних хмарних сервісів.

Рішення:

На основі отриманих висновків було вирішено розробити власний файловий сервер, орієнтований на корпоративні вимоги. Це рішення дозволить:

- забезпечити повний контроль над даними без ризику передачі третім сторонам.
- оптимізувати витрати на зберігання шляхом використання локальної або гібридної інфраструктури.
- налаштувати сервер відповідно до потреб компанії, інтегруючи його з існуючими ERP- та CRM-системами.
- забезпечити високу швидкість обробки даних та зручність доступу для співробітників без необхідності залежності від зовнішніх сервісів.
- виконати всі законодавчі вимоги щодо зберігання конфіденційної інформації.

2.6. Приклади використання корпоративного файлового серверу

- Інтеграція в корпоративну мережу великих підприємств.

- Контроль доступу: Файловий сервер дозволяє інтегруватися з існуючими системами аутентифікації, що забезпечує засоби тонкого регулювання прав доступу до документів на основі ролей співробітників.

- Корпоративний документообіг: Організації з великою кількістю секретної або внутрішньої документації потребують централізованої системи, що дозволяє не лише зберігати файли, а й контролювати їх редагування, перегляд і архівування.

- Проблеми конфіденційності та безпеки**:** Віддалений доступ і робота в умовах «home-office»: В умовах глобальної тенденції до віддаленої роботи, можливість стати центральним сховищем документів із підтримкою розширеного аудиту та журналювання дій користувачів гарантує безпеку корпоративної інформації, незалежно від місця знаходження співробітників.

- Спільна робота в команді: ваш файловий сервер може забезпечити інтеграцію з корпоративними системами комунікацій та інструментами спільної роботи, що дозволяє більш ефективно організувати робочі процеси в команді.

- Сервіс для компаній з високими вимогами до безпеки: Контроль внутрішнього обміну даними: організації, що працюють у сферах оборони, фінансів або охорони здоров’я, часто мають суворі вимоги щодо зберігання та передачі даних. Власний файловий сервер дозволяє організувати «sandbox»-середовище, де контроль доступу здійснюється централізовано і в повній відповідності до внутрішніх політик безпеки.

- Аудит та відповідність стандартам: застосування розгорнутого аудиту активності на сервері, із можливістю генерувати детальні звіти для відповідних контролюючих органів, є суттєвою перевагою в умовах підвищеного рівня регуляції.

- Проблеми конфіденційності та безпеки: можливість централізованої адміністрації дозволяє встановлювати спеціальні політики для різних підрозділів компанії.

- Індивідуальна інтеграція: крім стандартних функцій зберігання файлів, сервер може бути адаптований для обробки специфічних типів даних (наприклад, великі бази даних з медичними записами або технічною документацією), забезпечуючи необхідну продуктивність і масштабованість.

2.7. Переваги вашого файлового серверу над ринковими рішеннями

- Централізоване управління користувачами та доступом: адаптація під вимоги організації, на відміну від більшості публічних сервісів, ваш сервер дозволяє створювати та адмініструвати детальні політики доступу, адаптовані до специфічних вимог організації.

- Підвищена безпека та контроль над даними: використання власного серверу дозволяє зберігати дані на внутрішніх серверах компанії або у приватному хмарному середовищі, що зменшує ризики, пов’язані з передачею даних стороннім сервісам.

- Аудит активності та контроль версій: можливість вести детальний аудит всіх дій користувачів і дозволяти відновлення попередніх версій файлу стає незамінною функцією в умовах високої відповідальності за дані.

- Модульність функціоналу: розроблений сервер може бути розширений за потребою, впроваджуючи нові алгоритми шифрування, засоби резервного копіювання, інтеграцію з аналітичними інструментами або навіть штучним інтелектом для аналізу даних.

- Низька залежність від сторонніх сервісів: власне рішення зменшує ризики, пов’язані із зовнішніми обмеженнями, змінами цінової політики чи умов використання сторонніх платформ.

- Оптимізація витрат для великих організацій: розгортанні власного файлового серверу компанія інвестує один раз у інфраструктуру та підтримку,

що може бути економічно вигідним у довгостроковій перспективі порівняно з постійними витратами на підписку публічних сервісів.

Висновки

У розділі проаналізовано сучасний стан ринку файлових серверів та хмарних сховищ. Встановлено, що через питання безпеки, вартості та регуляторних обмежень великі компанії все частіше обирають власні рішення. Розробка корпоративного файлового серверу дозволяє забезпечити повний контроль над даними, гнучку інтеграцію та відповідність вимогам безпеки. Також підкреслено переваги власної інфраструктури над комерційними сервісами, особливо для бізнесу з високими стандартами конфіденційності. Власний файловий сервер є економічно доцільним в довгостроковій перспективі.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ФАЙЛОВОГО СЕРВЕРУ

3.1 Вибір технологій

Основна технологічна платформа – .NET 8

Для розробки серверної та клієнтської частини було обрано .NET 8, що є однією з найновіших версій .NET від Microsoft. Це універсальна кросплатформна технологія, яка підтримує Windows, Linux та macOS, що відкриває можливість подальшого розширення та масштабування проекту.

Переваги використання .NET 8:

- Висока продуктивність та оптимізована робота з пам’яттю.
- Покращена підтримка багатопотоковості та асинхронних операцій, що є критично важливим для роботи серверів.
- Інтегрований механізм роботи з API та мережею, що спрощує реалізацію клієнт-серверної взаємодії.
- Підтримка Entity Framework Core для зручної роботи з базою даних.
- Вбудована підтримка JSON-серіалізації, що є ключовою для обміну даними між клієнтом і сервером.

.NET 8 також забезпечує підтримку довгострокових оновлень (LTS), що гарантує стабільність платформи в майбутньому.

Система управління базами даних – PostgreSQL

Для збереження даних у проекті використовується PostgreSQL, що є однією з найпотужніших реляційних баз даних із відкритим вихідним кодом.

***Основні причини вибору PostgreSQL*:**

- Висока продуктивність при роботі з великими обсягами даних.
- Надійність та відмовостійкість (наприклад, підтримка ACID-транзакцій).
- Гнучкість у розширенні завдяки можливості написання власних функцій та розширень.
- Підтримка JSONB, що є важливим для зберігання структурованих даних у форматі JSON.

- Потужні можливості роботи з правами доступу, що необхідно для реалізації адміністративної панелі керування користувачами.

Завдяки PostgreSQL можна ефективно реалізувати зберігання даних користувачів, логування змін файлів, а також керування синхронізацією даних між сервером і клієнтами.

Робота з базою даних – Entity Framework Core

Для взаємодії сервера з базою даних використовується Entity Framework Core (EF Core) – ORM (Object-Relational Mapping) для .NET.

Переваги EF Core:

- Спрощує роботу з базою, дозволяючи використовувати C#-класи замість SQL-запитів.
- Підтримує Migrations, що полегшує оновлення структури бази без втрати даних.
- Автоматично оптимізує запити для ефективного використання ресурсів.
- Інтеграція з PostgreSQL через офіційний провайдер Npgsql.

Entity Framework Core дозволяє скоротити кількість коду, пов'язаного з роботою з базою даних, та підвищує безпеку шляхом захисту від SQL-ін'єкцій.

Протокол обміну даними – TCP-сокети

Для забезпечення швидкого та ефективного обміну даними між сервером та клієнтами використовується протокол TCP (Transmission Control Protocol).

Причини вибору TCP:

- Надійна передача даних без втрати пакетів.
- Підтримка постійного з'єднання між клієнтом і сервером, що зменшує затримки при синхронізації файлів.
- Гнучкість у створенні власних механізмів авторизації та обробки запитів.
- Вища продуктивність у порівнянні з HTTP у випадках постійного трафіку.

Використання сокетів TCP дозволяє клієнтам миттєво отримувати оновлення про зміни на сервері та забезпечує ефективну синхронізацію файлів.

Формат обміну даними – JSON

Більшість запитів і відповідей між клієнтом і сервером використовує формат JSON**.**

Переваги JSON:

- Легкий для читання та зручний у використанні.
- Швидка серіалізація та десеріалізація в .NET.
- Компактність у порівнянні з XML, що зменшує навантаження на мережу.
- Підтримка в PostgreSQL через JSONB-тип даних.

JSON ідеально підходить для передавання структурованих даних між сервером і клієнтом, наприклад, налаштувань користувача, параметрів синхронізації або мета-інформації файлів.

Графічний інтерфейс – WPF

Для створення клієнтського графічного інтерфейсу використовується Windows Presentation Foundation (WPF)**.**

Причини вибору WPF:

- Гнучкість у створенні сучасних UI з використанням XAML.
- Підтримка MVVM (Model-View-ViewModel) для спрощення структури коду.
- Підтримка апаратного прискорення (GPU), що забезпечує плавну анімацію.
- Великий набір готових контролів та можливість їхнього кастомного стилізування.
- Інтеграція з .NET та можливість використання бібліотек, таких як MahApps.Metro для покращеного UI.

Завдяки WPF клієнтська частина файлового сервера має зручний, швидкий та сучасний інтерфейс, що забезпечує просте управління файлами та налаштуваннями.

Середовище розробки

Для написання коду та роботи з базою даних використовувалися продукти компанії JetBrains – Rider та DataGrip**.**

JetBrains Rider – середовище розробки для .NET

Основним інструментом для написання коду був JetBrains Rider – кросплатформне середовище розробки для .NET, яке поєднує в собі продуктивність ReSharper та швидкодію IntelliJ-платформи.

Причини вибору Rider:

- Висока продуктивність і швидке індексування проєкту.
- Підтримка .NET 8, що є основною платформою проєкту.
- Інтелектуальні підказки та автодоповнення коду, що прискорює розробку.
- Гнучкі інструменти для роботи з Entity Framework Core (наприклад, перегляд і редагування міграцій).
- Вбудована підтримка роботи з Git та іншими системами контролю версій.

Rider також має потужний рефакторинг-код та інструменти аналізу, що допомагають писати код чистішим і більш оптимізованим.

DataGrip – інструмент для роботи з базою даних

Для адміністрування бази даних PostgreSQL використовувався JetBrains, DataGrip – один із найкращих клієнтів для роботи з реляційними БД.

Основні переваги DataGrip:

- Підтримка PostgreSQL та можливість роботи з іншими СУБД.
- Зручний редактор SQL-запитів з підсвічуванням синтаксису.
- Інструменти аналізу продуктивності запитів, що дозволяє оптимізувати взаємодію серверної частини з БД.
- Вбудований перегляд структури бази та можливість редагування даних без написання SQL-запитів вручну.
- Інтеграція з системами контролю версій (Git), що дозволяє зберігати та відстежувати зміни в схемі БД.

Використання DataGrip значно спростило роботу з PostgreSQL, забезпечивши гнучке керування структурою бази, аналіз продуктивності запитів та ефективне адміністрування даних.

Таким чином вибір зазначених технологій та інструментів дозволив реалізувати ефективну, продуктивну та масштабовану систему.

- ***.NET 8*** – потужна основа для розробки серверної та клієнтської частини.
- ***PostgreSQL + Entity Framework Core*** – стабільне та швидке збереження даних.
- ***TCP-сокети*** – ефективний спосіб обміну даними в реальному часі.
- ***JSON*** – простий і гнучкий формат передачі інформації.
- ***WPF*** – створення зручного та стильного інтерфейсу для клієнтської частини.
- ***JetBrains Rider та DataGrip*** – потужні інструменти, що забезпечили продуктивну розробку та адміністрування БД.

Такий вибір технологій гарантує стабільність роботи системи, зручність її використання та можливість подальшого розширення.

3.2. Проектування розробки

Архітектура:

За основу була взята модель клієнт серверної архітектури. Клієнт-серверна архітектура (рисунок 3.2.1) — це модель взаємодії між комп'ютерними системами, де одна сторона (клієнт) запитує сервіси чи ресурси, а інша сторона (сервер) їх надає.

Основні компоненти

- Клієнт – це пристрій або програма, яка надсилає запити до сервера. Наприклад, веб-браузер, мобільний додаток чи настільна програма.
- Сервер – це пристрій або програма, яка обробляє запити клієнтів і надає необхідні дані або послуги.

- Мережа – канал зв’язку між клієнтом і сервером, зазвичай через Інтернет або локальну мережу (LAN).

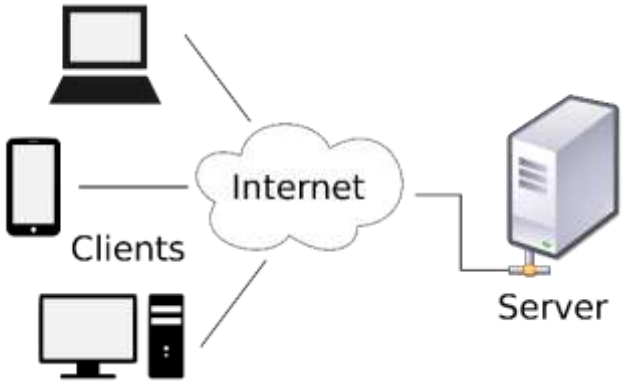

Рисунок 3.2.1. Схема

Складові проекту:

Проект було вирішено розділити на 3 окремі частини:

- Сервер – основна логіка проекту. Забезпечує взаємодію клієнтів з файлами на сервері та базою даних
- Адміністративна консоль - інструмент з графічним інтерфейсом для керування користувачами, їх рівнями доступу та розділення по групах.
- Клієнтський додаток – програма яка встановлюється на комп’ютер клієнтів. Також має графічний інтерфейс для авторизації та зміни налаштувань

База даних:

Під час розробки бази даних для файлового сервера було використано підхід "Code First", що є частиною ORM (Object-Relational Mapping) підходу. Це означає, що структура бази даних формується на основі об'єктів коду, а самі таблиці створюються та оновлюються автоматично за допомогою механізму міграцій.

Основні сутності бази даних:

База даних містить три основні сутності, що представляються у вигляді таблиць:

- Users – зберігає інформацію про користувачів.

- Sessions – відповідає за управління сесіями користувачів.
- Groups – використовується для групування користувачів та налаштування їхніх прав доступу.

Структура таблиць та зв'язки:

1. Users (Користувачі)

Ця таблиця містить основну інформацію про зареєстрованих користувачів.

Поля:

- Id – унікальний ідентифікатор користувача.
- Login – ім'я користувача (логін).
- Password – збережений у хешованому вигляді пароль.
- GroupId – зовнішній ключ, що пов'язує користувача з певною групою.

2. Sessions (Сесії)

Таблиця зберігає активні сесії користувачів, що дозволяє відстежувати, хто зараз підключений до системи.

Поля:

- Id– унікальний ідентифікатор сесії.
- Token – унікальний токен сесії, що використовується для аутентифікації.
- UserId – зв'язок із користувачем, який має активну сесію.

3. Groups (Групи користувачів)

Ця таблиця використовується для організації користувачів у групи та керування їхніми правами доступу.

Поля:

- Id – унікальний ідентифікатор групи.
- Name – назва групи.
- FolderPath – шлях до папки на сервері, яка буде синхронізуватись у користувачів цієї групи.
- Зовнішні ключі та зв'язки.

- Users (Id) → Sessions (UserID) (Зв’язок "один до багатьох": один користувач може мати декілька активних сесій).
- Groups (Id) → Users (GroupID)(Зв’язок "один до багатьох": одна група може містити багато користувачів).

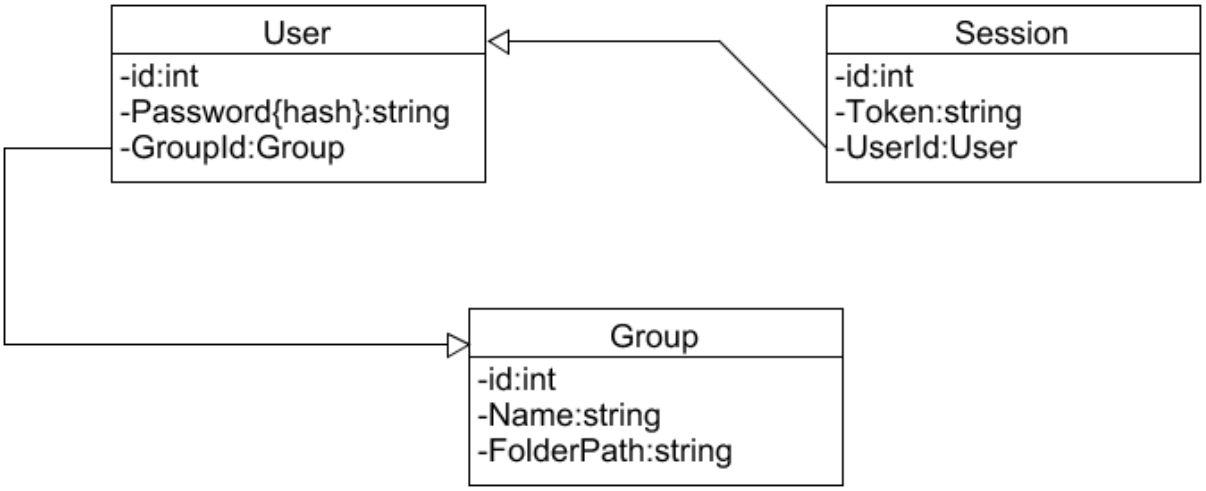

Рисунок 3.2.2. Блок-схема

Переваги підходу "Code First":

- Гнучкість у розробці – зміни в моделі автоматично відображаються в базі даних за допомогою механізму міграцій
- Менше залежності від конкретної СУБД – можна легко змінювати базу даних без необхідності переписувати SQL-запити.
- Автоматичне створення та оновлення структури бази – ORM самостійно генерує SQL-код для створення таблиць та їхніх зв’язків.
- Ця архітектура забезпечує надійність зберігання даних та ефективну взаємодію між користувачами, групами та сесіями.

3.3. Опис функціоналу

Файловий сервер має забезпечувати стабільний та безперебійний обмін даними між клієнтським пристроєм і сервером. Основна функція такої системи полягає у синхронізації файлів, що передбачає автоматичне оновлення та передачу даних між сервером та клієнтом. Це дозволяє користувачам отримувати

актуальні версії файлів незалежно від їхнього місцезнаходження та пристрою, з якого вони підключаються. Для досягнення ефективності та надійності обміну файлами програма має підтримувати механізми перевірки цілісності даних та безпечні протоколи передачі інформації.

Серверна частина:

Файловий сервер повинен забезпечувати ефективну взаємодію з клієнтами, включаючи механізми реєстрації, авторизації та аутентифікації користувачів. Це необхідно для контролю доступу та забезпечення безпеки переданих даних.

Основні етапи взаємодії клієнта з сервером:

1. Реєстрація – новий користувач створює обліковий запис, надаючи необхідні дані (логін, пароль). Сервер зберігає цю інформацію в базі даних із застосуванням механізмів шифрування для захисту паролів.
2. Аутентифікація – під час підключення клієнт вводить свої облікові дані, які сервер перевіряє.
3. Авторизація – після успішної аутентифікації сервер визначає рівень доступу користувача до ресурсів (файлів, директорій).
4. Передача файлів – здійснюється за допомогою TCP-протоколу, який забезпечує надійну та безпомилкову доставку даних. TCP використовує механізми підтвердження отримання пакунків та повторної передачі у разі втрат, що гарантує цілісність файлів під час обміну між клієнтом і сервером.

Завдяки використанню TCP-протоколу передача файлів відбувається стабільно навіть за нестабільного з’єднання, а вбудовані механізми контролю потоків запобігають перевантаженню мережі.

Клієнтська частина:

Після встановлення та реєстрації в додатку клієнт має отримати можливість налаштувати параметри синхронізації, зокрема обрати кінцеву папку на своєму ПК, куди будуть завантажуватися файли із сервера. Це забезпечить

гнучкість у використанні та дозволить користувачеві інтегрувати серверні дані у власну файлову систему без додаткових маніпуляцій.

Для зручності роботи з сервісом необхідно реалізувати механізм автоматичної синхронізації файлів без безпосередньої участі клієнта. Цей механізм має працювати у фоновому режимі, забезпечуючи своєчасне оновлення даних на клієнтському пристрої.

Основні етапи роботи автоматичної синхронізації:

1. Запит на сервер – клієнтський додаток періодично надсилає запит до сервера для перевірки змін у файловій структурі.

2. Порівняння версій файлів – сервер аналізує файли, що зберігаються у клієнта, та порівнює їх із останніми версіями на сервері.

3. Передача змінених файлів – якщо виявлено оновлення, сервер передає необхідні файли клієнту або, у випадку двосторонньої синхронізації, отримує оновлення від клієнта.

4. Збереження файлів у вибраній директорії – отримані файли автоматично завантажуються до папки, обраної клієнтом.

Налаштування періодичності синхронізації:

Оскільки частота оновлення файлів може залежати від потреб користувача, необхідно реалізувати можливість налаштування затримки між кожною синхронізацією. Це дасть змогу оптимізувати навантаження на мережу та сервер, а також уникати зайвого трафіку. Користувач зможе обирати між:

- Періодичною синхронізацією (з інтервалами, заданими користувачем, наприклад, кожну годину).
- Ручним оновленням (синхронізація запускається за запитом користувача).

Цей підхід забезпечить гнучке налаштування роботи додатка відповідно до потреб та можливостей клієнта, а також оптимізує використання ресурсів мережі та серверної інфраструктури.

Адміністративна консоль:

Адміністративна консоль є ключовим інструментом для управління користувачами та їхніми правами доступу на файловому сервері. Вона надає можливість виконувати основні операції з користувачами та групами, а також забезпечує зручний інтерфейс для швидкого отримання та обробки даних.

Керування користувачами:

- Видалення облікових записів користувачів у разі необхідності (наприклад, якщо обліковий запис більше не активний або порушує правила використання системи).
- Пошук конкретних користувачів за допомогою інтегрованої пошукової системи.

Робота з групами користувачів:

- Створення нових груп для організації користувачів за певними критеріями (наприклад, відділи компанії, рівні доступу, команди проєктів тощо).
- Редагування існуючих груп (зміна назви, опису, додавання або видалення користувачів).
- Призначення груп користувачам для спрощеного керування їхніми правами доступу.

Отримання та оновлення даних:

- Адміністратор має можливість вручну отримувати оновлені дані з бази для перегляду актуального стану користувачів, груп та їхніх налаштувань.
- Адміністративна консоль має адаптивний дизайн, що забезпечує її коректне відображення на різних екранах.

Таким чином, адміністративна консоль забезпечує гнучке управління користувачами та їхніми правами доступу, а також має зручний, адаптивний інтерфейс, що гарантує ефективну взаємодію адміністратора із системою.

3.4. Принцип роботи

Авторизація та аутентифікація:

Під час реєстрації у клієнтському додатку користувач вводить необхідні дані, які у вигляді JSON-запиту надсилаються на сервер. Сервер обробляє отриману інформацію, перевіряє її коректність та унікальність, після чого створює новий обліковий запис. Для безпечного збереження паролю використовується технологія хешування за допомогою технології PBKDF2 та алгоритму SHA-1. Якщо всі перевірки успішні, система генерує унікальний сесійний токен, який відправляється у відповідь клієнту. Токен виступає в ролі цифрового ключа, що підтверджує особу користувача під час подальших запитів.

Вхід до системи працює за аналогічним принципом: клієнт надсилає облікові дані на сервер, де вони перевіряються на відповідність збереженим у базі даних записам. У разі успішної аутентифікації сервер генерує новий токен або підтверджує чинність наявного, після чого повертає його клієнту. Надалі цей токен використовується для доступу до захищених ресурсів, що дозволяє уникнути повторного введення облікових даних.

Застосування токенів значно підвищує безпеку системи, оскільки унеможливлює зберігання паролів на клієнтській стороні та зменшує ризик компрометації облікових записів.

Загальна схема взаємодії компонентів системи на рисунку 3.2.3

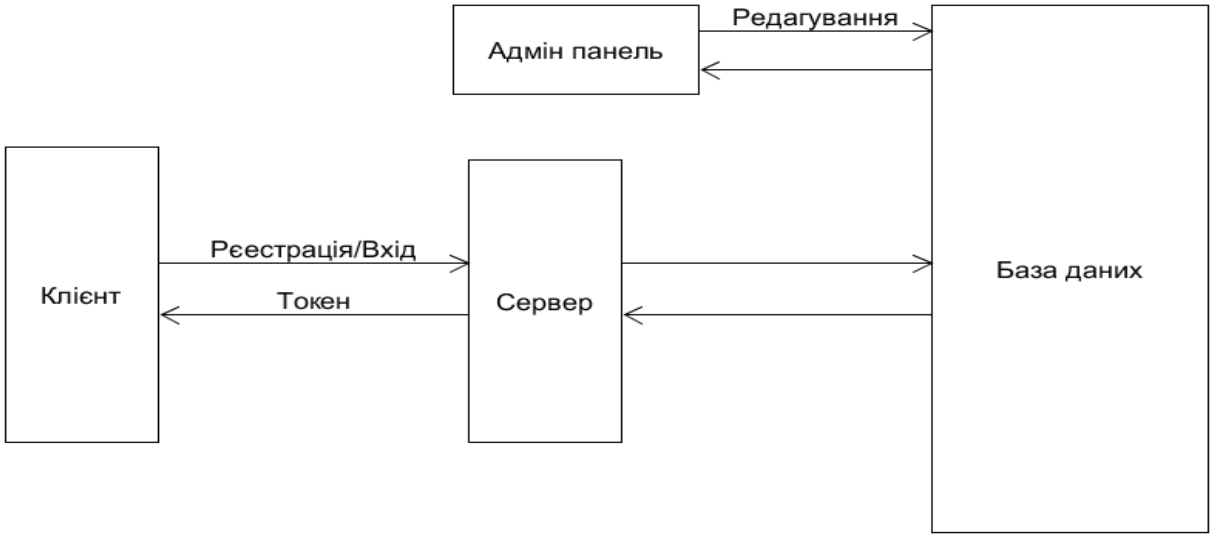

Рисунок 3.2.3. Блок-схема

Механізм синхронізації файлів між клієнтом і сервером:

Система синхронізації файлів передбачає обмін інформацією між клієнтом і сервером для забезпечення актуальності даних. Основним принципом роботи є порівняння списку файлів, що зберігаються на клієнтському пристрої, із файлами, доступними на сервері. Такий підхід дозволяє мінімізувати обсяг переданих даних і забезпечити ефективне оновлення файлів. Етапи синхронізації складаються з:

1. Формування списку файлів клієнтом.

Перший етап синхронізації передбачає створення клієнтським додатком списку файлів у вибраній папці. Для цього застосовується алгоритм обходу файлової системи, який записує всі файли разом із їхніми характеристиками.

Основними параметрами, що фіксуються, є:

- Хеш-функція файлу – унікальне значення, згенероване на основі вмісту файлу (MD5). Використання хешування дозволяє швидко ідентифікувати зміни у файлах.
- Розмір файлу – кількість байтів у файлі, що також є додатковим критерієм перевірки.

- Назва файлу – збереження ідентифікатора, що дозволяє визначити його призначення.
- Відносний шлях – шлях до файлу в межах синхронізованої директорії, що дозволяє структурувати дані після отримання нових файлів.

Ця інформація збирається у вигляді структури даних та використовується для порівняння із серверною версією.

2. Перевірка серверних файлів та їх доступність.

На сервері файли організовані за групами, доступ до яких визначається адміністратором у адміністративній консолі. Це забезпечує розмежування прав користувачів і гарантує, що клієнт отримає лише ті файли, які входять до дозволеної групи.

Після отримання списку файлів від клієнта сервер виконує наступні дії:

1. Визначає, до якої групи належить клієнт.

2. Вивантажує список файлів, доступних для цієї групи.

3. Відправляє клієнту лише той перелік файлів, які відповідають групі користувача.

Таким чином, якщо користувач не має доступу до певної групи, він не зможе отримати відповідні файли навіть у разі прямого запиту.

3. Порівняння списків та запит відсутніх або змінених файлів.

Після отримання серверного списку клієнт починає процес порівняння:

1. Якщо файл з однаковим шляхом і назвою присутній на клієнті, але його хеш не співпадає із серверним, це означає, що файл був змінений, і його необхідно оновити.

2. Якщо файлу взагалі немає у клієнтській директорії, він також підлягає завантаженню.

3. Якщо файл є у клієнта, але відсутній у серверному списку (наприклад, він був видалений адміністратором), система видалить його.

Для кожного такого випадку клієнт відправляє запит до сервера для отримання відповідних файлів. Схема на рисунку 3.2.4.

4. Завантаження файлів на клієнт.

Після обробки запитів сервер передає клієнту потрібні файли, які зберігаються у папці, обраній для синхронізації. Перед відправкою кожен файл проходить перевірку, чи належить він до групи користувача, який зробив запит. Це відбувається для того щоб уникнути несанкціонованого доступу. Схема на рисунку 3.2.5


Запит списку файлів
Порівняння локальних файлів з серверними
Файл співпадає
Так
Ні
Так
Запит файлу
Отримання
Запис файлу
Серверні файли ще лишились
Ні
Видалення непотрібних файлів

Рисунок 3.2.4. Блок-схема

Рисунок 3.2.5. Блок-схема

Запропонований механізм дозволяє забезпечити:

- Ефективне управління файлами завдяки використанню хешів та розмежуванню доступу.
- Безпеку даних, оскільки користувач отримує доступ лише до дозволених файлів.
- Мінімізацію мережевого навантаження, оскільки передаються лише змінені або відсутні файли.

Даний підхід є оптимальним для корпоративних систем управління файлами, сервісів резервного копіювання, а також для систем колективної роботи з документами.

Функціонування серверної частини:

Серверна частина програмного комплексу побудована за асинхронним принципом, що забезпечує високу продуктивність і безперервну обробку запитів користувачів. Сервер функціонує у режимі постійного приймання запитів та

надання відповідей, що дозволяє мінімізувати час очікування з боку клієнта та забезпечити стабільну роботу системи.

Окрім обробки запитів у реальному часі, сервер також виконує періодичні операції фонової перевірки. Зокрема, кожні 5 хвилин він здійснює автоматичне сканування всіх папок наявних груп. У ході цього процесу сервер отримує інформацію про всі файли, що містяться в цих папках, і зберігає їхні параметри у спеціальну структуру даних у оперативній пам’яті. Такий підхід дозволяє значно пришвидшити подальшу взаємодію з користувачами, оскільки сервер вже має актуальні метадані файлів і не потребує повторного сканування у момент запиту. Це сприяє оптимізації ресурсів та зменшенню навантаження на файлову систему.

Щодо обмежень на зберігання даних, система дозволяє зберігати файли розміром до 2 Гб кожен, при цьому загальний обсяг папок користувачів не обмежений. Такий підхід забезпечує баланс між продуктивністю сервера та можливостями збереження даних, запобігаючи перевантаженню системи великими окремими файлами.

Загалом, асинхронна архітектура та впроваджені механізми безпеки гарантують стабільну, ефективну та швидкодіючу роботу сервера, що є критично важливим для забезпечення безперебійного обслуговування користувачів.

3.5. Взаємодія з інтерфейсом

Клієнтська програма:

Робота користувача з програмою розпочинається з вікна авторизації (рисунок 3.5.1), яке забезпечує контроль доступу до системи. Авторизація є необхідною для ідентифікації користувача та перевірки його прав доступу до сервісу. Вікно містить поля для введення облікових даних (логіна та пароля), а також кнопки для входу та реєстрації нового користувача.

Перед виконанням авторизації програма встановлює з'єднання з сервером для перевірки коректності введених даних. Якщо сервер тимчасово недоступний

через технічні причини або мережеві проблеми, користувач отримує відповідне повідомлення про помилку. Це дозволяє йому зрозуміти причину збою та повторити спробу пізніше.

Якщо користувач здійснює некоректну реєстрацію (наприклад, вводить пароль, що не відповідає вимогам безпеки, або використовує вже зареєстрований логін), система відображає відповідне повідомлення з поясненням помилки (рисунок 3.5.2). Аналогічно, у разі некоректного введення облікових даних при вході (наприклад, неправильний пароль або відсутність облікового запису в базі даних), користувач отримує конкретне повідомлення про причину відмови у доступі (рисунок 3.5.3).

Такий підхід підвищує зручність використання програми, оскільки користувач не залишається в невідомості щодо причини невдачі. Завдяки чіткій обробці помилок і детальним повідомленням, мінімізується кількість запитів до технічної підтримки, а також підвищується загальний рівень користувацького досвіду.

Рисунок 3.5.1

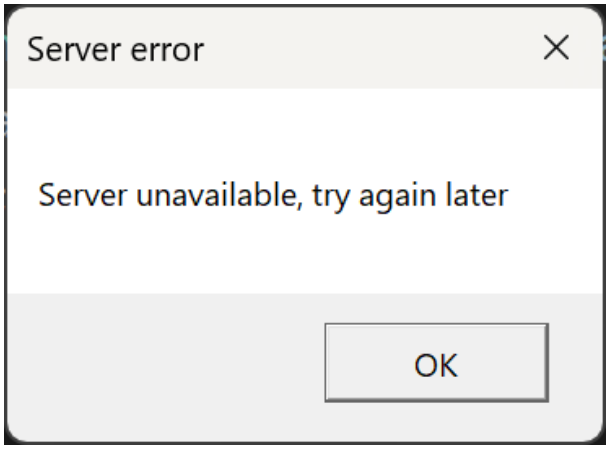

Рисунок 3.5.2

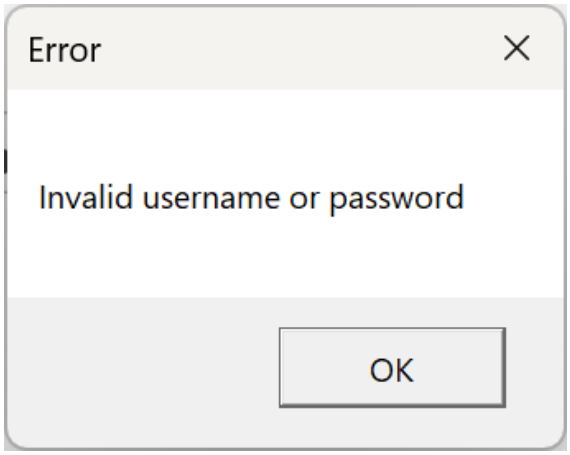

Рисунок 3.5.3

Після успішного проходження авторизації користувач отримує доступ до меню налаштувань. У цьому розділі передбачено низку функціональних можливостей, спрямованих на керування процесом синхронізації файлів (рисунок 3.5.4). Користувач може використовувати наступні кнопки для взаємодії з функціоналом:

- *Select directory* – визначити каталог, вміст якого буде автоматично або вручну синхронізуватися із сервером.
- *Sync files* – ініціювати процес передачі та оновлення файлів між локальним пристроєм і сервером.
- *Add to scheduler* – налаштувати автоматичний запуск процесу синхронізації відповідно до заданого розкладу за допомогою “Планувальнику Windows”.
- *Add to scheduler* – налаштувати автоматичний запуск процесу синхронізації відповідно до заданого розкладу за допомогою “Планувальнику Windows”.
- *Delete from scheduler* – видалити раніше додане налаштування автоматичної синхронізації у “Планувальнику Windows”.
- *Logout* – завершити поточний сеанс роботи.

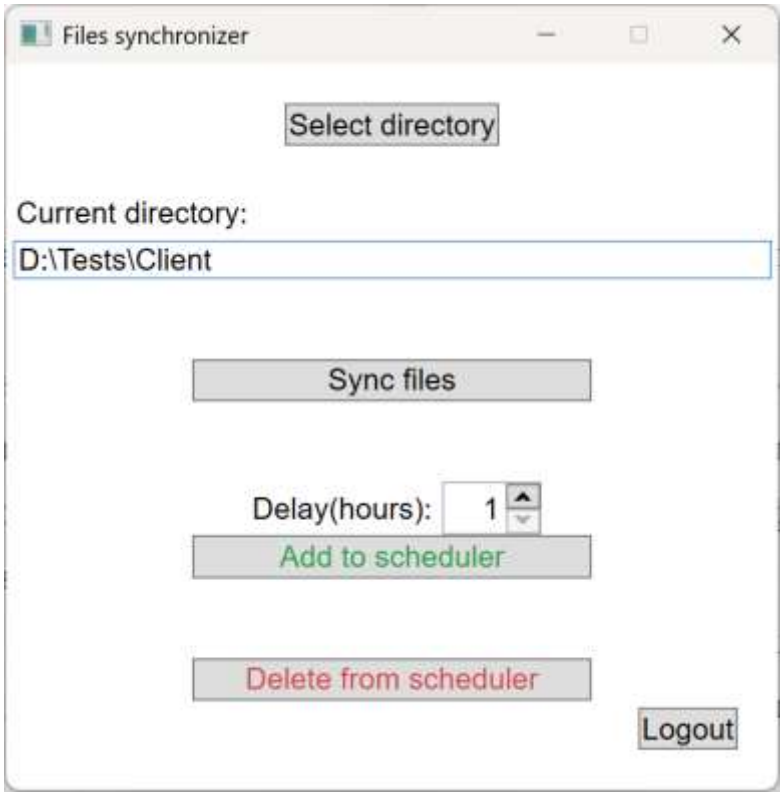

Рисунок 3.5.1

Адміністративна консоль:

Адміністраторська панель забезпечує централізоване керування користувачами та групами в системі файлового сервера (рисунок 3.5.2). Завдяки широкому функціоналу, адміністратор має можливість:

- Видаляти користувачів – у разі потреби прибирати акаунти зі списку активних користувачів. Для цього потрібно вибрати користувача за допомогою миші та натиснути клавішу “Delete” та кнопку “Save changes” для збереження змін.
- Створювати та редагувати групи – формувати нові групи користувачів, змінювати їхні параметри та склад за допомогою кнопок “Create” та “Edit” у розділі “Groups”. Після натиску кнопки відкриється додаткове вікно зі створенням (рисунок 3.5.3) або редагуванням (рисунок 3.5.4) групи. Також э можливість видаляти групи, Для цього потрібно вибрати групу за допомогою

миші та натиснути клавішу “Delete” та кнопку “Save changes” для збереження змін.

- Призначати групи користувачам – розподіляти права доступу та функціональні можливості, закріплюючи користувачів за відповідними групами, для цього потрібно натиснути на комірку у колонці “Groups” напроти необхідного юзера (рисунок 3.5.5).
- Шукати конкретних користувачів – здійснювати швидкий пошук за заданими критеріями, що значно полегшує адміністрування великої кількості акаунтів. Для пошуку потрібно ввести ім'я користувача у строку пошуку та натиснути кнопку “Search” (рисунок 3.5.6).
- Оновлювати дані з бази даних – отримувати актуальну інформацію, для цього потрібно натиснути кнопку “Load from database”.

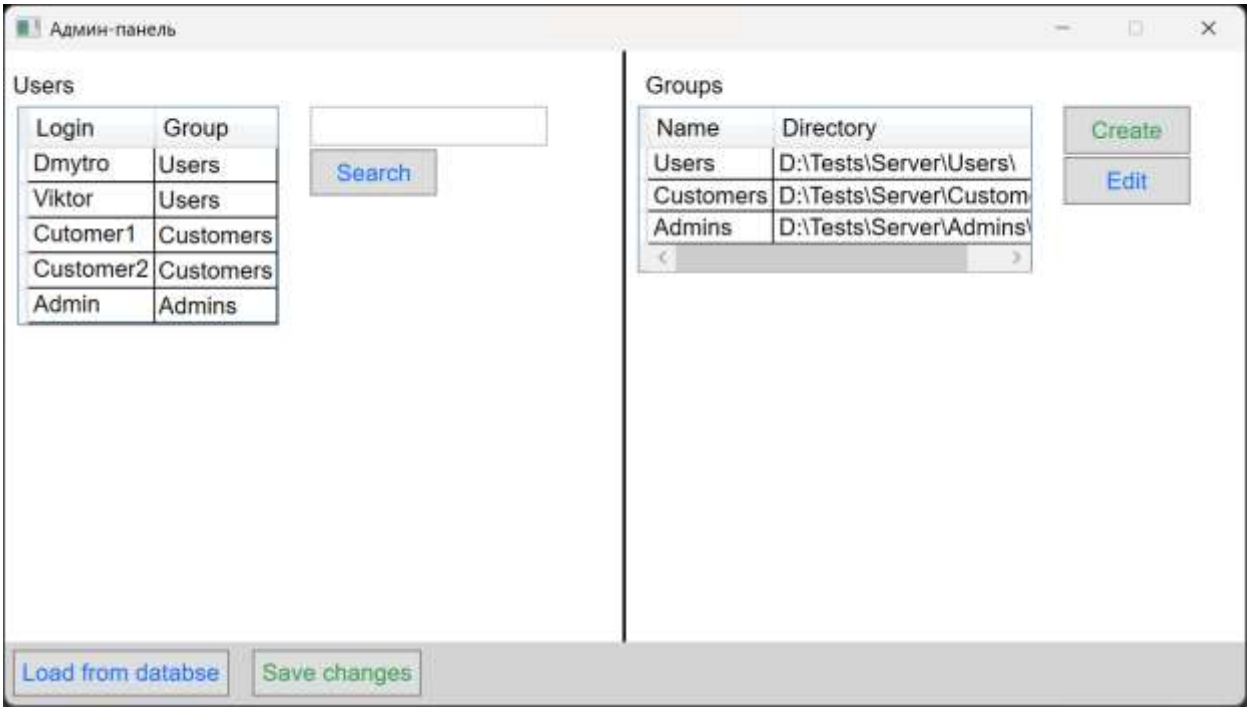

Рисунок 3.5.2

Рисунок 3.5.3

Рисунок 3.5.4

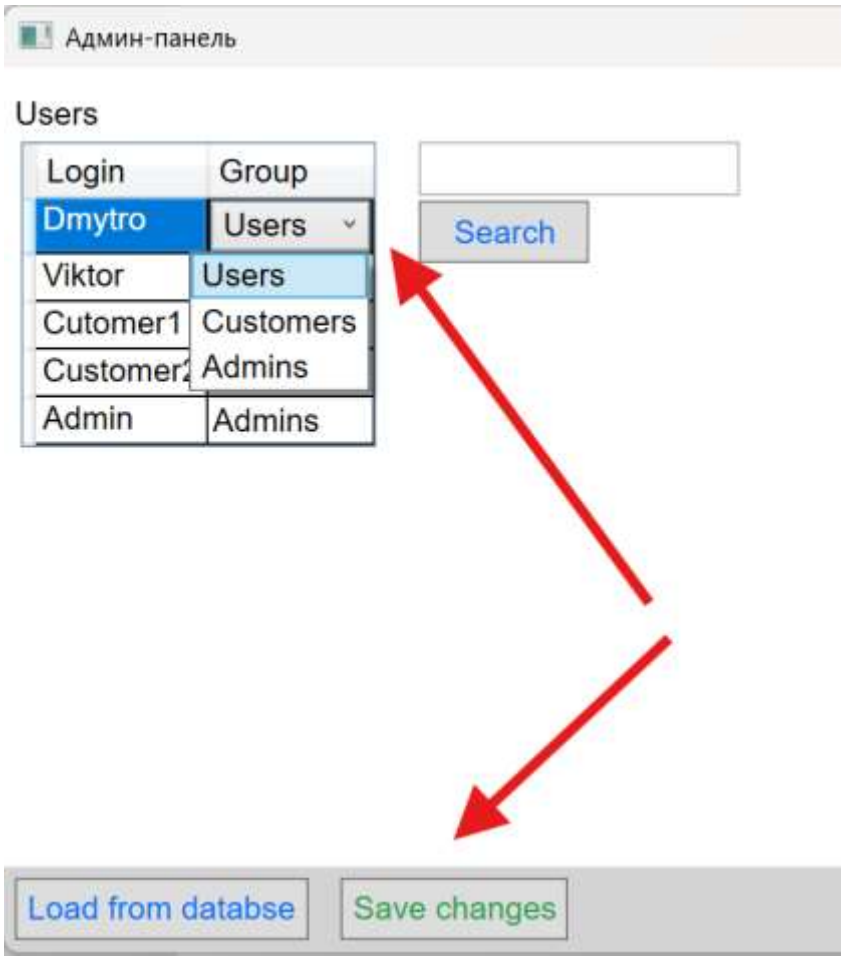

Рисунок 3.5.4

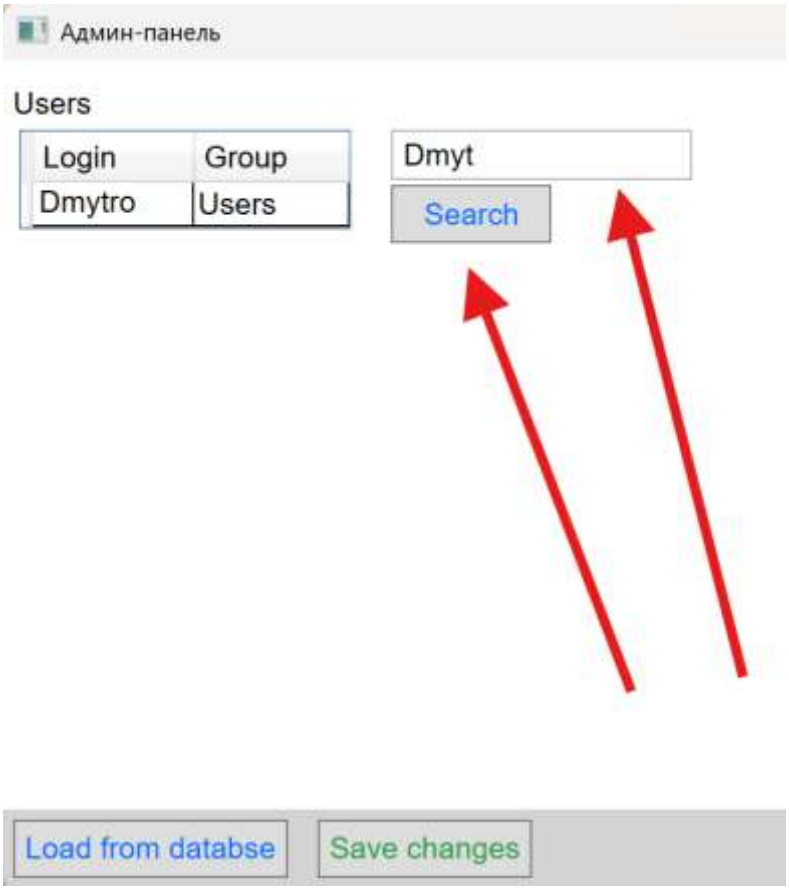

Рисунок 3.5.4

Інтерфейс адміністративної панелі виконано з використанням адаптивного дизайну, що забезпечує коректне та зручне відображення на пристроях із різними розмірами екрану та в різних режимах перегляду.

ВИСНОВОК

У рамках дипломної роботи було розроблено повноцінне клієнт-серверне програмне забезпечення, яке забезпечує безпечну та зручну синхронізацію файлів між сервером та клієнтом. Розроблена система спрямована на вирішення широкого спектра завдань – від корпоративного до комерційного використання, демонструючи високий рівень адаптивності та функціональності.

Реалізація проекту дозволила отримати суттєвий практичний досвід у роботі з різними компонентами програмного продукту. Особливу увагу приділено вдосконаленню навичок роботи з платформою ".Net", що дозволило не лише реалізувати проект за сучасними вимогами, але й значно розширити компетенції щодо використання сучасних технологій у розробці програмного забезпечення.

Незважаючи на вже реалізований великий обсяг функціоналу, система має значний потенціал для подальшого розвитку та інтеграції нових можливостей. Перспективними напрямками подальших досліджень є оптимізація алгоритмів синхронізації, підвищення рівня безпеки даних та розширення функціональних можливостей клієнтського та серверного модулів.

Отриманий досвід та проведене дослідження свідчать про практичну значимість розробленого рішення та його потенціал для застосування в різноманітних галузях інформаційних технологій. Ця робота є вагомим внеском у сферу розробки високонадійних та ефективних файлових серверів, що відповідають сучасним стандартам безпеки та зручності використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке сервер? Визначення, типи та особливості – URL: https://itedu.center/ua/blog/comparisons/types_of_servers/

2. Wikipedia – URL: https://uk.wikipedia.org (дата звернення: 02.03.2025)

3. Як вибрати файловий сервер? – URL: https://server-shop.ua/ua/how-to-choose-a-file-server.html (дата звернення: 02.03.2025)

4. Kingston’s Knowledge Centre – URL: https://www.kingston.com/en/blog (дата звернення: 04.03.2025)

5. Cornell University. Arxiv – URL: https://arxiv.org (дата звернення: 05.03.2025)

6. Best Disk Drill Alternatives for Windows and Mac – URL: https://www.magicuneraser.com/press/best_disk_drill_alternatives.php (дата звернення: 08.03.2025)

7. 10 Найкращих хмарних сервісів зберігання інформації – URL: https://www.moyo.ua/ua/news/10-luchshikh-oblachnykh-servisov-khraneniya-informatsii.html (дата звернення: 08.03.2025)

8. Топ-5 хмарних сховищ для бізнесу та особистого використання у 2025 році – URL: https://blog.colobridge.net/uk/2024/03/top-cloud-storage-2024-ua/ (дата звернення: 16.03.2025)

9. Чим замінити Google Drive: 8 альтернатив для зберігання файлів – URL: https://spacelab.ua/articles/chim-zaminiti-google-drive-8-alternativ-dlya-zberigannya-fajliv/ (дата звернення: 16.03.2025)

10. Google Drive vs Dropbox vs OneDrive – URL: https://www.goodcloudstorage.net/google-drive-vs-dropbox-vs-onedrive/ (дата звернення: 16.03.2025)

11. Cloud Computing Statistics - URL: https://99firms.com/blog/cloud-computing-statistics (дата звернення: 16.03.2025)

12. File server – URL: https://www.techtarget.com/searchnetworking/definition/file-server (дата звернення 18.03.2025)

13. Microsoft documentation - URL: https://learn.microsoft.com/en-us/ (дата звернення: 20.03.2025)

14. Resources for Developers – URL: https://developer.mozilla.org/en-US/ (дата звернення: 20.03.2025)

15. Форум – URL: https://stackoverflow.com/questions (дата звернення: 20.03.2025)

16. Веб-сервіс для розробки програмного забезпечення – URL: https://github.com (дата звернення: 20.03.2025)

17. Mastering .Net – URL: https://medium.com (дата звернення 20.03.2025)

18. Галузеві тренди. Штучний інтелект в Україні: як розвивається галузь – URL: https://hub.kyivstar.ua/articles/galuzevi-trendi-shtuchnij-intelekt-v-ukrayini-yak-rozvivayetsya-galuz (дата звернення: 22.03.2025)

ДОДАТОК А. Код програми

Сервер

Program.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.Entity;
using System.Diagnostics.CodeAnalysis;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Security.Cryptography;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore.Diagnostics;
using Microsoft.Extensions.Configuration;
using Server.Database;
using Server.Files;
using Server.Helpers;
using Server.Models;
using Server.Workers;

namespace Server
{
    // State object for reading client data asynchronously
    public class StateObject
    {
```

```
// Size of receive buffer.
public const int BufferSize = 1024;

// Receive buffer.
public byte[] buffer = new byte[BufferSize];

// Received data string.
public StringBuilder sb = new StringBuilder();

// Client socket.
public Socket workSocket = null;

private long targetBytesCount = 0;
private long sentBytesCount = 0;

public long GetRest()
{
   return this.targetBytesCount - this.sentBytesCount;
}

public void SetGoal(long targetBytesCount)
{
   this.targetBytesCount = targetBytesCount;
   this.sentBytesCount = 0;
}

public long SetNextResultNGetRest(long nextSentBytesCount)
{
   this.sentBytesCount += nextSentBytesCount;
```

```
        return this.targetBytesCount - this.sentBytesCount;
    }

    public void ResetGoal()
    {
        this.targetBytesCount = 0;
        this.sentBytesCount = 0;
    }
}

public class AsynchronousSocketListener
{
    // Thread signal.
    public static ManualResetEvent allDone = new ManualResetEvent(false);

    public static List<FilesGroup>FilesGroups  = new List<FilesGroup>();

    public const int maxSize=2000000000;

    public static void RefreshFiles()
    {
        while (true)
        {

            using (Context db = new Context())
            {
                int currentCount = FilesGroups.Count;
                for (int i = 0; i < currentCount; i++)
                {
```

```
                if (db.Groups.FirstOrDefault(x=>
x.Id==FilesGroups[i].GroupId)==null)
                {
                    FilesGroups.Remove(FilesGroups[i]);
                    currentCount = FilesGroups.Count;
                }
            }

            foreach (var filesGroup in db.Groups)
            {
                var temp = new List<FileM>();
                try
                {
                    if (filesGroup.FolderPath!="")
                    {
                        ScanFiles.Start(filesGroup.FolderPath, temp);
                    }

                    var candidate = FilesGroups.FirstOrDefault(x => x.GroupId ==
filesGroup.Id);

                    if (candidate==null)
                    {
                        FilesGroups.Add(new FilesGroup(){GroupId = filesGroup.Id,
files = temp});
                    }
                    else
                    {
                        candidate.files = temp;
```

```
candidate.JsonList=JsonWorker<List<FileM>>.ObjToJson(temp);
                    }

                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }


            }

            Thread.Sleep(3000);
        }


    }
}

public static async void AsyncRefreshFiles()
{
    await Task.Run(RefreshFiles);
}
public static void StartListening()
{
    // Establish the local endpoint for the socket.
    // The DNS name of the computer
    IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
```

```
        var builder= new ConfigurationBuilder().AddJsonFile("appsettings.json",
true, true);
        var config = builder.Build();

        IPAddress ipAddress = IPAddress.Parse(config["LocalIp"]);
        IPEndPoint localEndPoint = new IPEndPoint(ipAddress,
int.Parse(config["LocalPort"]));

        // Create a TCP/IP socket.
        Socket listener = new Socket(ipAddress.AddressFamily,
            SocketType.Stream, ProtocolType.Tcp );

        int count = 0;

        AsyncRefreshFiles();

        // Bind the socket to the local endpoint and listen for incoming connections.
        try {
            listener.Bind(localEndPoint);
            listener.Listen(100);

            while (true) {
                // Set the event to nonsignaled state.
                allDone.Reset();

                // Start an asynchronous socket to listen for connections.
                listener.BeginAccept(
                    new AsyncCallback(AcceptCallback),
```

```
            listener );

            // Wait until a connection is made before continuing.
            allDone.WaitOne();

        }

    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }

    Console.WriteLine("\nPress ENTER to continue...");
    Console.Read();

}

public static void AcceptCallback(IAsyncResult ar)
{
    // Signal the main thread to continue.
    allDone.Set();

    // Get the socket that handles the client request.
    Socket listener = (Socket) ar.AsyncState;
    Socket handler = listener.EndAccept(ar);

    // Create the state object.
    StateObject state = new StateObject();
    state.workSocket = handler;
    handler.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
```

```
        ReadCallback, state);
}

public static void ReadCallback(IAsyncResult ar)
{
    String content = String.Empty;

    // Retrieve the state object and the handler socket
    // from the asynchronous state object.
    StateObject state = (StateObject) ar.AsyncState;
    Socket handler = state.workSocket;

    // Read data from the client socket.

    int bytesRead;


    try
    {
        bytesRead = handler.EndReceive(ar);
    }
    catch (Exception e)
    {
        return;
    }

    if (bytesRead > 0) {
        // There  might be more data, so store the data received so far.
        state.sb.Append(Encoding.Unicode.GetString(
```

```
            state.buffer, 0, bytesRead));

        // Check for end-of-file tag. If it is not there, read
        // more data.
        try
        {
            content = state.sb.ToString();

            if (content.Contains("GET:FILES:LIST:"))
            {
                var temp = content.Split(":");


                string token = temp.Last();

                Group group = DbHelper.GetGroupBySession(token);

                if (group!= null)
                {
                    var filesGroup = FilesGroups.FirstOrDefault(x => x.GroupId ==
group.Id);

                    if (filesGroup!=null)
                    {
                        string req = content.Replace(token, "");
                        if (req=="GET:FILES:LIST:LENGTH:")
                        {
                            Send(handler, filesGroup.JsonList.Length.ToString());
                        }
```

```
                else if (req=="GET:FILES:LIST:")
                {
                    Send(handler,filesGroup.JsonList);
                }
            }
        }
        else if (content.Contains("GET:FILE:"))
        {
            string temp = content.Replace("GET:FILE:", "");

            string path=temp.Substring(0, temp.IndexOf(':'));

            string token = temp.Replace(path+":", "");

            var groupDb = DbHelper.GetGroupBySession(token);

            var group = FilesGroups.FirstOrDefault(x => x.GroupId ==
groupDb.Id);

            if (group!=null)
            {
                var file = group.files.FirstOrDefault(x => x.Path == path);

                if (file != null && File.Exists(groupDb.FolderPath+file.Path))
                {
                    try
                    {
                        StateObject fileSendState = new StateObject();
```

```
    if (file.Size<=maxSize)
    {
        String filePath = groupDb.FolderPath + file.Path;
        byte[] fileByteArray = File.ReadAllBytes(filePath);
        state.SetGoal(fileByteArray.Length);
        handler.BeginSend(
            fileByteArray,
            0,
            fileByteArray.Length,
            SocketFlags.None,
            SendFileCallback,
            new object[]
            {
                handler,
                fileByteArray,
                fileSendState,
                file.Size,
                maxSize

            }
        );

    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    // ignored
```

```
                }
            }
            else
            {
                Send(handler, "0");
            }
        }
    }
    else if (content.Contains("ADD:USER:"))
    {
        string data = content.Replace("ADD:USER:", "");

        ClientUser tempUser = JsonWorker<ClientUser>.JsonToObj(data);

        using (Context db = new Context())
        {
            if (db.Users.FirstOrDefault(x => x.Login
==tempUser.Login)!=null)
            {
                Send(handler, "USER:EXISTS");
            }
            else
            {
                var user = new User()
                    {Login = tempUser.Login, Password =
PasswordHash.CreateHash(tempUser.Password)};

                db.Users.Add(user);
```

```
                    var bytes = new byte[16];
                    using (var rng = new RNGCryptoServiceProvider())
                    {
                       rng.GetBytes(bytes);
                    }

                    string token = BitConverter.ToString(bytes).Replace("-",
"").ToLower();

                    db.Sessions.Add(new Session() {Token = token, User = user});

                    Send(handler, "ADD:SUCCESS:"+ token);
                    db.SaveChanges();
                }
            }


        }
        else if (content.Contains("SIGN:IN:"))
        {
            string data = content.Replace("SIGN:IN:", "");

            ClientUser tempUser = JsonWorker<ClientUser>.JsonToObj(data);

            using (Context db = new Context())
            {
```

```
                var candidate = db.Users.FirstOrDefault(x => x.Login ==
tempUser.Login);

                if (candidate!= null &&
PasswordHash.ValidatePassword(tempUser.Password, candidate.Password))
                {
                    var bytes = new byte[16];
                    using (var rng = new RNGCryptoServiceProvider())
                    {
                        rng.GetBytes(bytes);
                    }

                    string token = BitConverter.ToString(bytes).Replace("-",
"").ToLower();

                    db.Sessions.Add(new Session() {Token = token, User =
candidate});

                    db.SaveChanges();

                    Send(handler, "SIGN:IN:SUCCESS:"+token);
                }
                else
                {
                    Send(handler, "SIGN:IN:INCORRECT");
                }
            }

        }
```

```
else if (content.Contains("SIGN:OUT:"))
{
   string data = content.Replace("SIGN:OUT:", "");

   using (Context db = new Context())
   {
      var candidate = db.Sessions.FirstOrDefault(x => x.Token == data);
      if (candidate!=null)
      {
         db.Sessions.Remove(candidate);
         db.SaveChanges();
      }
   }
}
else if (content.Contains("CHECK:SESSION:"))
{
   string token = content.Replace("CHECK:SESSION:", "");

   using (var db=new Context())
   {
      if (db.Sessions.FirstOrDefault(x => x.Token == token) != null)
      {
         Send(handler, "SESSION:CORRECT");
      }
      else
      {
         Send(handler, "SESSION:INCORRECT");
      }
```

```
            }
        }

        else {
            // Not all data received. Get more.
            handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                new AsyncCallback(ReadCallback), state);
        }
    }
    catch (Exception e)
    {
        // ignored
    }
  }
}

private static void Send(Socket handler, String data)
{
  StateObject fileSendState = new StateObject();



  // Convert the string data to byte data using ASCII encoding.
  byte[] byteData = Encoding.Unicode.GetBytes(data);

  fileSendState.SetGoal(byteData.Length);
  // Begin sending the data to the remote device.
  handler.BeginSend(byteData, 0, byteData.Length, 0,
    new AsyncCallback(SendCallback), new object[]
```

```
            {
                handler,
                byteData,
                fileSendState
            });
        }

        private static void SendCallback(IAsyncResult ar)
        {
            try
            {
                // Retrieve the socket from the state object.
                Socket handler = (Socket) ((object[])ar.AsyncState)[0];
                byte[] jsonByteArray = (byte[])((object[])ar.AsyncState)[1];
                StateObject jsonSendState = (StateObject)((object[])ar.AsyncState)[2];

                // Complete sending the data to the remote device.
                int bytesSent = handler.EndSend(ar);

                long total = jsonSendState.SetNextResultNGetRest(bytesSent);

                if (total > 0)
                {
                    handler.BeginSend(jsonByteArray, (int)total, jsonByteArray.Length -
(int)total,
                        SocketFlags.None, new AsyncCallback(SendCallback), handler);
                    return;
                }
```

```
        Console.WriteLine("Sent {0} bytes to client.", bytesSent);

        handler.Shutdown(SocketShutdown.Both);
        handler.Close();

    }
    catch (Exception e)
    {
        //ignored
    }
}

private static void SendFileCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the socket from the state object.
        Socket handler = (Socket) ((object[])ar.AsyncState)[0];
        byte[] fileByteArray = (byte[])((object[])ar.AsyncState)[1];
        StateObject fileSendState = (StateObject)((object[])ar.AsyncState)[2];

        long fileSize = (long)((object[]) ar.AsyncState)[3];
        int maxSize=(int)((object[]) ar.AsyncState)[4];

        // Complete sending the data to the remote device.
        // handler.EndSendFile(ar);
        int nextSentBytesCount = handler.EndSend(ar);
```

```
            long total = fileSendState.SetNextResultNGetRest(nextSentBytesCount);

            if (total > 0)
            {
               handler.BeginSend(fileByteArray, (int)total, fileByteArray.Length -
(int)total,
                  SocketFlags.None, new AsyncCallback(SendFileCallback), handler);
               return;
            }

            if (fileSize<maxSize)
            {
               handler.Shutdown(SocketShutdown.Both);
               handler.Close();

            }

         }
         catch (Exception e)
         {
            Console.WriteLine(e.ToString());
         }
      }

      public static void Main(String[] args)
      {
         Console.WriteLine("Start");
         StartListening();
      }
```

```
    }
}
```

Адміністративна консоль

MainWindowViewModel.cs:

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows;
using Server.Helpers;
using Server.Models;
using Server.Views;

namespace Server.ViewModels
{
    public class MainWindowViewModel:BaseViewModel
    {
        public ObservableCollection<User> Users { get; set; } = new
ObservableCollection<User>();

        public List<User> OldUsers;
        public ObservableCollection<Group> Groups { get; set;}= new
ObservableCollection<Group>();
        public Group SelectedGroup { get; set; }


        private string _prevFilter="";
        public string SearchFilter { get; set; }

        public User SelectedUser { get; set;}
```

```
private Window _window;

public MainWindowViewModel(Window window)
{
   SearchFilter = "";

   _window = window;

   OldUsers = new List<User>();

   _window.Closing += new CancelEventHandler(Window_Closing);

   DBHelper.LoadAll(Users, Groups, OldUsers);
}

public RelayCommand SaveBtn
{
   get
   {
      return new RelayCommand(
         obj =>
         {
            DBHelper.SaveGroupsDeleted(Groups, Users);
            DBHelper.SaveUsers(Users);

            OldUsers.Clear();

            foreach (var user in Users)
```

```
            {
                OldUsers.Add((User)user.Clone());
            }

            MessageBox.Show("Changes were saved!");
        }
    );
  }
}



public RelayCommand ResetBtn
{
  get
  {
    return new RelayCommand(
      obj =>
      {

        if (DBHelper.IsChanged(Users, OldUsers,  _prevFilter))
        {
          var result=MessageBox.Show("Unsaved changes. Save now?",
            "Warning", MessageBoxButton.YesNo);
          if (result==MessageBoxResult.Yes)
          {
            DBHelper.SaveGroupsDeleted(Groups, Users);
            DBHelper.SaveUsers(Users);
```

```
                }
            }
            SearchFilter = "";
            OnPropertyChanged("SearchFilter");
            _prevFilter = "";
            DBHelper.LoadAll(Users, Groups, OldUsers);
        }
    );
  }
}

public RelayCommand AddGroupBtn
{
  get
  {
    return new RelayCommand(
      obj =>
      {
        _window.IsEnabled = false;
        Window addWindow = new GroupWindow(this);
        addWindow.Owner = _window;
        addWindow.Show();

      }
    );
  }
}

public RelayCommand EditBtn
```

```
{
    get
    {
        return new RelayCommand(
            obj =>
            {
                if (SelectedGroup==null)
                {
                    MessageBox.Show("Group is not selected!");
                    return;
                }
                Window addWindow = new GroupWindow(this, true);
                _window.IsEnabled=false;
                addWindow.Owner = _window;
                addWindow.Show();
            }
        );
    }
}


public RelayCommand SearchBtn
{
    get
    {
        return new RelayCommand(
            obj =>
            {
                if (SearchFilter=="")
```

```
            {
                return;
            }
            if (DBHelper.IsChanged(Users,OldUsers ,_prevFilter))
            {
                var result=MessageBox.Show("Unsaved changes. Save now?",
                    "Warning", MessageBoxButton.YesNo);
                if (result==MessageBoxResult.Yes)
                {
                    DBHelper.SaveGroupsDeleted(Groups, Users);
                    DBHelper.SaveUsers(Users);

                }
            }

            DBHelper.LoadAll(Users, Groups, OldUsers,SearchFilter);
            _prevFilter = SearchFilter;
        }
    );
  }
}


void Window_Closing(object sender, CancelEventArgs e)
{
  if (DBHelper.IsChanged(Users, OldUsers,  _prevFilter))
  {
    var result=MessageBox.Show("Unsaved changes. Save now?",
      "Warning", MessageBoxButton.YesNo);
```

```
            if (result==MessageBoxResult.Yes)
            {
                DBHelper.SaveGroupsDeleted(Groups, Users);
                DBHelper.SaveUsers(Users);
            }
        }
    }
}
```

GroupWindowViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Windows;
using Microsoft.WindowsAPICodePack.Dialogs;
using Server.Database;
using Server.Helpers;
using Server.Models;

namespace Server.ViewModels
{
    public class GroupWindowViewModel:BaseViewModel
    {

        private Window _window;

        private MainWindowViewModel _mainWindow;
```

```
private bool _isEdit;

private string _name ="";

private string _folderPath="";

public string Name
{
   get { return _name;}
   set
   {
      _name = value;
      OnPropertyChanged("Name");
   }
}
public string FolderPath
{
   get { return _folderPath;}
   set
   {
      _folderPath = value;
      OnPropertyChanged("FolderPath");
   }
}

public GroupWindowViewModel(Window window, MainWindowViewModel
mainWindow, bool isEdit=false)
{
```

```
            _mainWindow = mainWindow;
            if (isEdit)
            {
                Name = _mainWindow.SelectedGroup.Name;
                FolderPath = _mainWindow.SelectedGroup.FolderPath;
            }

            _isEdit = isEdit;
            _window = window;
            _window.Closing += new CancelEventHandler(Window_Closing);
        }

        public RelayCommand SaveBtn
        {
            get
            {
                return new RelayCommand(
                    obj =>
                    {
                        _window.Owner.IsEnabled = true;
                        if (Name.Length<4)
                        {
                            MessageBox.Show("Minimum number of characters in name is 4",
"Error");
                            return;
                        }
                        if (_isEdit==false)
                        {
```

```
                var newGroup = new Group() {Name = Name, FolderPath =
FolderPath};
                _mainWindow.Groups.Add(newGroup);
                using (Context db = new Context())
                {
                    db.Groups.Add(newGroup);
                    db.SaveChanges();
                }
                _window.Close();
            }
            else
            {
                using (Context db = new Context())
                {
                    var candidate=db.Groups.FirstOrDefault(x => x.Id ==
                        _mainWindow.SelectedGroup.Id);

                    candidate.Name = Name;
                    candidate.FolderPath = FolderPath;
                    db.SaveChanges();

                }

                DBHelper.LoadAll(_mainWindow.Users, _mainWindow.Groups,
_mainWindow.OldUsers);
            }
            _window.Close();
        }
    );
```

```
        }
    }

    public RelayCommand SelectFolder
    {
        get
        {
            return new RelayCommand(
                obj =>
                {
                    var dlg = new CommonOpenFileDialog();
                    dlg.IsFolderPicker = true;
                    _window.Topmost = false;
                    if (dlg.ShowDialog() == CommonFileDialogResult.Ok)
                    {
                        FolderPath = dlg.FileName+"\\";
                    }
                    _window.Topmost = true;
                }
            );
        }
    }

    void Window_Closing(object sender, CancelEventArgs e)
    {
        _window.Owner.IsEnabled = true;

    }
}
```

```
}
```

Клієнтська програма. Графічний інтерфейс

AuthenticationWindowViewModel.cs

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Windows;
using System.Windows.Controls;
using Client.Helpers;
using Client.Models;
using Client.Packets;
using Client.Workers;
using TcpClient = System.Net.Sockets.TcpClient;

namespace Client.ViewModels
{
   public class AuthenticationWindowViewModel:BaseViewModel
   {
      private readonly Window _window;
      private UserData UserData { get; set; }
      public string CurrentLogin { get; set; }

      public AuthenticationWindowViewModel(Window window)
      {
         _window = window;
         CurrentLogin = "";

         try
         {
```

```
                using (var client = TcpHelper.GetClient())
                {
                    using (var stream = client.GetStream())
                    {

                    }

                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Server unavailable, try again later", "Server
error");
                Application.Current.Shutdown();
            }

            if (File.Exists(@"Sync\UserData.json"))
            {
                UserData =
JsonWorker<UserData>.JsonToObj(File.ReadAllText(@"Sync\UserData.json"));

                string answer;

                if (UserData.SessionToken!=null)
                {
                    using (var client = TcpHelper.GetClient())
                    {
                        using (var stream = client.GetStream())
                        {
```

```
                        PacketSender.SendJsonString(stream,
"CHECK:SESSION:"+Encryptor.EncodeDecrypt(UserData.SessionToken));

                        answer = PacketRecipient.GetJsonData(stream);
                    }
                }

                if (answer.Contains("SESSION:CORRECT"))
                {
                    var nextWindow = new MainWindow(UserData);
                    nextWindow.Show();
                    _window.Close();
                }
            }

        }
        else
        {
            UserData = new UserData() {SessionToken = null, FolderPath =
null};
        }
    }

    public RelayCommand RegistrationBtn
    {
        get
        {
            return new RelayCommand(
                obj =>
```

```
        {
            var pwBox = obj as PasswordBox;

            if (CurrentLogin.Length < 4 || pwBox.Password.Length < 4)
            {
                MessageBox.Show("Minimum number of characters in login
and password is 4", "Error");
                return;
            }

            var user = new User(){Login = CurrentLogin, Password =
pwBox.Password};

            try
            {
                string answer;
                using (var client = TcpHelper.GetClient())
                {
                    using (var stream = client.GetStream())
                    {
                        PacketSender.SendJsonString(stream, "ADD:USER:" +
JsonWorker<User>.ObjToJson(user));

                        answer = PacketRecipient.GetJsonData(stream);
                    }
                }

                if (answer.Contains("ADD:SUCCESS:"))
                {
```

```
                    string token =
Encryptor.EncodeDecrypt(answer.Replace("ADD:SUCCESS:", ""));

                    UserData.SessionToken = token;

                    File.WriteAllText(@"Sync\UserData.json",
JsonWorker<UserData>.ObjToJson(UserData));

                    var nextWindow = new MainWindow(UserData);
                    nextWindow.Show();
                    _window.Close();

                }
                else if(answer=="USER:EXISTS")
                {
                    MessageBox.Show("This login is already in use", "Error");
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Server unavailable, try again later",
"Error");
            }

        }
    );
  }
}
```

```
        public RelayCommand SignInBtn
        {
            get
            {
                return new RelayCommand(
                    obj =>
                    {
                        var pwBox = obj as PasswordBox;

                        var user = new User(){Login = CurrentLogin, Password =
pwBox.Password};

                        try
                        {
                            string answer;
                            using (var client = TcpHelper.GetClient())
                            {
                                using (var stream = client.GetStream())
                                {
                                    PacketSender.SendJsonString(stream, "SIGN:IN:" +
JsonWorker<User>.ObjToJson(user));

                                    answer = PacketRecipient.GetJsonData(stream);
                                }
                            }
```

```
                    if (answer.Contains("SIGN:IN:SUCCESS:"))
                    {
                        string token =
Encryptor.EncodeDecrypt(answer.Replace("SIGN:IN:SUCCESS:", ""));

                        UserData.SessionToken = token;

                        File.WriteAllText(@"Sync\UserData.json",
JsonWorker<UserData>.ObjToJson(UserData));

                        var nextWindow = new MainWindow(UserData);
                        nextWindow.Show();
                        _window.Close();

                    }
                    else if(answer=="SIGN:IN:INCORRECT")
                    {
                        MessageBox.Show("Invalid username or password",
"Error");
                    }
                }
                catch (Exception e)
                {
                    MessageBox.Show("Server unavailable, try again later",
"Error");
                }

            }
        );
```

```
            }
        }


    }
}
```

MainWindowViewModel.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Windows;
using Client.Helpers;
using Client.Models;
using Client.Packets;
using Client.Workers;
using Microsoft.Win32.TaskScheduler;
using Microsoft.WindowsAPICodePack.Dialogs;

namespace Client.ViewModels
{
    public class MainWindowViewModel : BaseViewModel
    {
        private readonly Window _window;

        private readonly UserData UserData;
```

```
        public int TimeDelay { get; set; }

        public MainWindowViewModel(UserData userData, Window window)
        {
            _window = window;
            UserData = userData;
            TimeDelay = 1;
            if (UserData.FolderPath == null) UserData.FolderPath = "Not selected
";
        }

        public string FolderPath
        {
            get => UserData.FolderPath.Remove(UserData.FolderPath.Length - 1);
            set
            {
                UserData.FolderPath = value;
                OnPropertyChanged();
            }
        }

        public RelayCommand SignOutBtn
        {
            get
            {
                return new RelayCommand(obj =>
                {
                    using (var client = TcpHelper.GetClient())
                    {
```

```
                    using (var stream = client.GetStream())
                    {
                        PacketSender.SendJsonString(stream,
                            "SIGN:OUT:" +
Encryptor.EncodeDecrypt(UserData.SessionToken));
                    }
                }

                UserData.SessionToken = null;
                File.WriteAllText(@"Sync\UserData.json",
JsonWorker<UserData>.ObjToJson(UserData));

                var nextWindow = new AuthenticationWindow();
                nextWindow.Show();
                _window.Close();
            });
        }
    }

    public RelayCommand SelectFolderBtn
    {
        get
        {
            return new RelayCommand(obj =>
            {
                var dlg = new CommonOpenFileDialog();
                dlg.IsFolderPicker = true;

                if (dlg.ShowDialog() == CommonFileDialogResult.Ok)
```

```
                {
                    FolderPath = dlg.FileName + "\\";
                    File.WriteAllText(@"Sync\UserData.json",
JsonWorker<UserData>.ObjToJson(UserData));
                }
            });
        }
    }

    public RelayCommand SyncFiles
    {
        get
        {
            return new RelayCommand(obj =>
            {
                if (FolderPath=="Not selected")
                {
                    MessageBox.Show("No directory selected!", "Error");
                    return;
                }
                try
                {
                    var currentDir =
Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

                    var process = new Process();
                    process.StartInfo.FileName = currentDir +
@"\Sync\Client.exe";
                    process.Start();
```

```
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message, "Error");
            }
        });
    }
}

public RelayCommand CreateTask
{
    get
    {
        return new RelayCommand(obj =>
        {
            if (FolderPath=="Not selected")
            {
                MessageBox.Show("No directory selected!", "Error");
                return;
            }
            using (TaskService ts = new())
            {
                TaskDefinition td = ts.NewTask();
                td.RegistrationInfo.Description = "Sync files with the server";

                var dt = new DailyTrigger();
                dt.StartBoundary = DateTime.Now;
                dt.Repetition = new
RepetitionPattern(TimeSpan.FromHours(TimeDelay), TimeSpan.FromDays(1));
```

```
                    td.Triggers.Add(dt);

                    var currentDir =
Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

                    td.Actions.Add(new ExecAction(currentDir +
@"Sync\Client.exe"));

                    ts.RootFolder.RegisterTaskDefinition(@"FilesSynchronizer",
td);
                }

                MessageBox.Show("The task was added!", "Success");
            });
        }
    }

    public RelayCommand DelTask
    {
        get
        {
            return new RelayCommand(obj =>
            {
                using (TaskService ts = new())
                {
                    try
                    {
                        ts.RootFolder.DeleteTask("FilesSynchronizer");
                    }
```

```
                catch (Exception e)
                {
                    MessageBox.Show("You haven't added a task yet!", "Error");
                    return;
                }

                MessageBox.Show("Task was deleted", "Success");
            }
        });
    }
  }
}
```

Клієнтська програма. Синхронізація файлів

Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Threading;
using Client.Files;
using Client.Helpers;
using Client.Models;
using Client.Packets;
using Client.Workers;

namespace Client
{
```

```
        internal class Program
        {
            public static void Main(string[] args)
            {
                var currentPath =
Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

                var userData =
JsonWorker<UserData>.JsonToObj(File.ReadAllText(currentPath + @"\" +
"UserData.json"));

                var mainDir = userData.FolderPath;

                var serverFiles = new List<FileM>();

                string data;

                int fileListLenght;

                using (var client = TcpHelper.GetClient())
                {
                    using (var stream = client.GetStream())
                    {
                        PacketSender.SendJsonString(stream,
                            "GET:FILES:LIST:LENGTH:" +
Encryptor.EncodeDecrypt(userData.SessionToken));
```

```
fileListLenght=Convert.ToInt32(PacketRecipient.GetJsonData(stream,
client.SendBufferSize));

                }

            }

            using (var client = TcpHelper.GetClient())
            {
                using (var stream = client.GetStream())
                {
                    PacketSender.SendJsonString(stream,
                        "GET:FILES:LIST:" +
Encryptor.EncodeDecrypt(userData.SessionToken));

                    data = PacketRecipient.GetJsonData(stream,
client.SendBufferSize, fileListLenght);
                }
            }

            serverFiles = JsonWorker<List<FileM>>.JsonToObj(data);

            if (serverFiles.Count == 0)
            {
                var folder = new DirectoryInfo(mainDir);

                foreach (var file in folder.GetFiles()) file.Delete();
```

```
            foreach (var dir in folder.GetDirectories()) dir.Delete(true);
        }


        var files = new List<FileM>();

        ScanFiles.Start(mainDir, files);

        long totalSize = 0;

        foreach (var item in serverFiles)
        {
            totalSize += item.Size;

            if (item.Size == 0)
            {
                if (!Directory.Exists(item.Path))
                    Directory.CreateDirectory(mainDir +
Path.GetDirectoryName(item.Path));
                File.Create(mainDir + item.Path);
                continue;
            }

            var selectedItem = files.FirstOrDefault(x => x.Path == item.Path);

            if (selectedItem == null)
            {
                PacketFile.GetFile(item.Path, serverFiles,
Encryptor.EncodeDecrypt(userData.SessionToken), mainDir);
```

```
                continue;
            }

            if (item.Size != selectedItem.Size)
            {
                File.Delete(mainDir + selectedItem.Path);
                PacketFile.GetFile(item.Path, serverFiles,
Encryptor.EncodeDecrypt(userData.SessionToken), mainDir);
            }
            else
            {
                string hash;

                try
                {
                    hash = CreateFileHash.CreateMD5(mainDir +
selectedItem.Path);
                }
                catch (IOException)
                {
                    continue;
                }

                if (hash != item.Hash)
                {
                    File.Delete(mainDir + selectedItem.Path);
                    PacketFile.GetFile(item.Path, serverFiles,
Encryptor.EncodeDecrypt(userData.SessionToken),
                        mainDir);
```

```
                }
            }
        }

        foreach (var i in files)
            if (!serverFiles.Exists(x => x.Path == i.Path))
                File.Delete(mainDir + i.Path);

        Dirs.ClearEmptyDirs(mainDir);

        long totalSizeClient = 0;

        var clientFiles = new List<FileM>();
    }
  }
}
```