Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP”

Виконав здобувач вищої освіти
IV курсу, групи КМ-20
ОПП «Комп'ютерна інженерія»
спеціальності 123 «Комп'ютерна інженерія»
_____ Калиновський В. М.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Калиновському Валентину Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№___ від __. .2024 року*

2. Строк подання студентом роботи до захисту *__. .2024 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи передачі файлів між комп'ютером та сервером у локальній мережі за протоколом FTP*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « » 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	22.05.2024 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Калиновський В.М. Програмне забезпечення кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP.

Метою роботи є створення системи передачі файлів між комп'ютером та сервером у локальній мережі за протоколом FTP.

Результат роботи – програмна реалізація кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Клієнт може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows, Linux, Android.

Програму розроблено на мовах програмування Python та Kotlin.

Ключові слова: локальна мережа, передача файлів, клієнт, сервер, FTP

ABSTRACT

Kalynovskyi V.M. Software of a cross-platform client-server system for transferring files using the FTP. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2024.

In this bachelor's qualification work, software is developed, which is intended for a cross-platform client-server system for transferring files using the FTP.

The purpose of the development is to create a file transfer system between a computer and a server in a local network using the FTP.

The result of the work is the software implementation of a cross-platform client-server system for transferring files using the FTP.

In the process of working on the implementation of the system, a study of existing methods, algorithms and software tools was performed. Our own software was developed and implemented, and all its components were described.

A convenient user interface has been developed. The instructions for working with software tools are provided.

The client can be used on PCs of IBM PC architecture with Windows, Linux, Android OS.

The software is developed in Python and Kotlin programming languages.

Keywords: local network, file transfer, client, server, FTP

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

FTP (File Transfer Protocol) – Протокол передачі файлів;

PI (Protocol Interpreter) – Інтерпретатор протоколу;

DTP (Data Transfer Process) – Процес передачі файлів;

HTTP (HyperText Transfer Protocol) – Протокол передачі гіпер-тексту;

SSL (Secure Sockets Layer) – Рівень захищених сокетів;

TLS (Transport Layer Security) – Захист на транспортному рівні;

OSI (Open Systems Interconnection) – Взаємозв'язок відкритих систем – абстрактна мережева модель для комунікацій і розроблення мережевих протоколів;

TCP (Transmission Control Protocol) – Протокол керування передачею – один із головних протоколів передачі даних в інтернеті; призначений для керування передачею даних в інтернеті;

UDP (User Datagram Protocol) – Протокол датаграм користувача – один із протоколів в стеку TCP/IP; від протоколу TCP він відрізняється тим, що працює без встановлення з'єднання; протокол є одним із найпростіших протоколів транспортного рівня моделі OSI, котрий виконує обмін повідомленнями (датаграмами від англійського «datagram») без підтвердження та гарантії доставки;

IP (Internet Protocol) – Інтернет протокол – маршрутизований протокол мережевого рівня стеку TCP/IP; невід'ємною частиною протоколу є адресація мережі;

URI (Uniform Resource Identifier) – Уніфікований ідентифікатор ресурсів – компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс; головним призначенням таких ідентифікаторів є створення можливі взаємодії з поданнями ресурсів через мережу, використовуючи спеціальні протоколи;

HDD (Hard disk drive) – Жорсткий Диск;

drag&drop (в перекладі з англійської «тягний-й-кидай») – технологія, спосіб оперування елементами інтерфейсу в інтерфейсах користувача (як графічних, так і текстових, де елементи графічного інтерфейсу реалізовані за допомогою псевдографіки) за допомогою маніпулятора «миша» чи сенсорного екрану;

API (application programming interface) – Інтерфейс програмування застосунків – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення;

СУБД/СКБД – Система управління/керування базами даних (від англійського DataBase Management System, DBMS) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних; надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних;

БД – База даних;

ПК – Персональний комп’ютер;

ОС – Операційна система;

ОЗУ – Оперативний запам’ятовуючий пристрій;

ПЗУ – Постійний запам’ятовуючий пристрій;

Лог (від англійського log – журнал) – запис у спеціальному файлів, який зберігає службову, технічну чи статистичну інформацію про події на сервері чи в системі; частіше це слово відноситься не до конкретного запису, а до цілого файлу записів і означає «файл записів»; даний термін може трактуватись як обидва із наведених понять;

Хендлер (від англійського handler – обробник) – функція, яка викликається якоюсь програмною системою у відповідь на настання будь-якої події;

Віджет (від англійського widget – штучка) – у графічному інтерфейсі це елемент взаємодії; призначені, як інструмент для швидкого доступу до певної інформації чи сервісів.

ВСТУП

На початку конструювання перших комп'ютерів з'явилась потреба у зберіганні розрахунків тривалий час. А з розвитком комп'ютерів з'явилась потреба у їх передачі між комп'ютерами. Підтримувати провідну мережу з часом становилось все накладніше, так як зростання загальної довжини проводів та комутуючого обладнання збільшувалась експоненційно і утримувати багато персоналу стало обтяжливо. З кожним роком витрати на обслуговування росли. Перехід не безпровідну передачу файлів/даних став необхідною мірою для розвитку технологій у технологічному плані та для економії грошових ресурсів у економічному плані.

Передача файлів безпровідним зв'язком є актуальною проблемою в наш час, оскільки дозволяє прискорити передачу даних і зекономити на обслуговуванні обладнання та виробництві нових проводів.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи передачі файлів між комп'ютером та сервером у локальній мережі за протоколом FTP.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих методів передачі файлів;

– Розробка алгоритмів системи передачі файлів;

– Програмна реалізація системи передачі файлів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі передачі файлів безпровідним шляхом між різними пристроями.

Отже, розробка та впровадження програмного забезпечення системи передачі файлів безпровідним шляхом є актуальною задачею, яка вимагає постійного вдосконалення та вирішувалася у цій кваліфікаційній роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

FTP є протоколом передачі файлів, розробленим для ефективного обміну даними між комп'ютерами у мережі (локальній/інтернет). Головна мета FTP полягає в забезпеченні надійного, швидкого та безпечного способу передачі файлів будь-якого типу та розміру. Цей протокол реалізує завантаження та вивантаження файлів на FTP-сервер, видалення файлів, видалення/створення директорій, перейменування файлів, тощо. Використання FTP сприяє оптимізації робочих процесів, полегшує обмін файлами між комп’ютерами та дозволяє забезпечити безпеку та конфіденційність даних під час їх передачі.

Основне призначення FTP полягає в тому, щоб дозволити користувачам здійснювати зв'язок з сервером і передавати файли туди або з нього. Він працює на принципах клієнт-серверної архітектури, де клієнт (наприклад, ваш комп'ютер) ініціює з'єднання з сервером (наприклад, веб-сайтом або сервером файлів), щоб виконати різні операції з файлами.

Головні функції FTP включають:

– Передача файлів: Користувачі можуть вивантажувати файли на сервер або завантажувати їх з нього за допомогою FTP. Це особливо корисно для веб-розробників, які розміщають свої веб-сайти на серверах.

– Керування файлами і каталогами: FTP дозволяє користувачам створювати, видаляти, переміщати та копіювати файли і каталоги на сервері. Це допомагає в організації файлової структури і керуванні контентом.

– Аутентифікація і авторизація: FTP забезпечує механізми для перевірки ідентичності користувачів і надання доступу до відповідних ресурсів на сервері. Це гарантує безпеку файлів і обмежує доступ лише для авторизованих користувачів.

– Перегляд і керування правами доступу: FTP дозволяє налаштовувати права

доступу до файлів і каталогів на сервері, визначаючи, хто може переглядати, редагувати або видаляти їх.

– Підтримка багатьох платформ: FTP є кросплатформовим протоколом, що означає, що він працює на різних операційних системах, таких як Windows, macOS і Linux.

Загалом, FTP використовується для широкого спектру завдань, від керування веб-сайтами до обміну файлами між користувачами. Його простота встановлення та використання робить його популярним інструментом для обміну файлами у мережах.

1.2 Область застосування

У сучасному бізнес-середовищі FTP використовується для обміну різноманітною інформацією, починаючи від фінансових звітів і конфіденційної корпоративної документації до медіа-файлів та БД клієнтів. Великі корпорації часто використовують FTP для автоматизації процесів обміну даними між різними відділами та підрозділами, що дозволяє ефективно координувати роботу та забезпечує швидкий доступ до актуальної інформації.

Також варто зазначити, що FTP забезпечує можливість взаємодії з клієнтами та партнерами, дозволяючи легко обмінюватися великими обсягами даних у безпечному середовищі. Наприклад, роздрібні компанії можуть використовувати FTP для отримання оновлень товарів та партнерських договорів від своїх постачальників, тим самим підтримуючи актуальність асортименту та узгодженість умов співпраці.

У науковій сфері FTP відіграє ключову роль у забезпеченні обміну даними та документами між дослідницькими групами та академічними установами. Він дозволяє вченим спільно працювати над проектами, обмінюватися результатами досліджень та даними, що сприяє прискоренню наукового прогресу та співпраці у межах міжнародних наукових спільнот.

В медичній сфері FTP є незамінним інструментом для обміну медичною інформацією, зображеннями та даними пацієнтів між лікарнями, клініками та

діагностичними центрами. Це дозволяє медичним працівникам отримувати швидкий доступ до необхідної інформації про пацієнтів та діагностичні дані, що може врятувати час у критичних ситуаціях та покращити якість медичного обслуговування.

Крім сфер бізнесу, науки та медицини, FTP відіграє критичну роль у таких галузях як правоохоронна діяльність та управління кризовими ситуаціями. Поліція, спецслужби та інші установи забезпечення безпеки використовують FTP для обміну важливою інформацією, документами та доказами в рамках розслідувань кримінальних справ та забезпечення громадського порядку. Також, управління кризовими ситуаціями, такими як природні лиха чи техногенні аварії, використовують FTP для координації дій, обміну інформацією та забезпечення швидкого реагування на надзвичайні ситуації.

Особиста та побутова сфера також відчуває вплив FTP. Для звичайних користувачів він може стати незамінним інструментом для обміну файлами між різними пристроями, такими як комп'ютери, смартфони та планшети. Наприклад, використовуючи FTP, люди можуть легко передавати фотографії, відео, музику та інші файли між своїми пристроями, незалежно від їх марки або операційної системи. Крім того, FTP може використовуватися для резервного копіювання важливих файлів та даних, що дозволяє зберігати їх у безпечному місці та відновлювати у випадку втрати або пошкодження основних пристроїв.

Таким чином, FTP є важливим інструментом в багатьох критичних сферах діяльності та є невід'ємною частиною повсякденного життя для багатьох користувачів, що підкреслює його широкий спектр застосувань та важливість у сучасному світі інформаційних технологій.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Протокол FTP має багато аналогів:

– SFTP (SSH File Transfer Protocol) – протокол передачі файлів, який використовує шифрування за допомогою протоколу SSH для забезпечення безпеки під час передачі даних. SFTP забезпечує аутентифікацію, конфіденційність та цілісність даних, що робить його популярним в сфері інформаційної безпеки та обміну даними через інтернет.

– FTPS (FTP over SSL/TLS) – цей протокол поєднує FTP з шифруванням SSL/TLS для забезпечення безпеки передачі даних. FTPS використовує сертифікати SSL/TLS для аутентифікації сервера та шифрування даних, що передаються між клієнтом і сервером. Він забезпечує захищений канал для передачі файлів і є популярним у банківській та фінансовій сферах та для забезпечення безпеки під час обміну конфіденційною інформацією.

– HTTP/HTTPS – хоча HTTP протокол призначений для передачі гіпертекстових документів, він також може використовуватися для передачі файлів за допомогою HTTP-запитів, таких як PUT та GET. HTTPS надає шифрування та аутентифікацію за допомогою протоколу SSL/TLS, що робить його популярним в сфері безпеки.

– SCP (Secure Copy Protocol) – протокол, який використовує SSH для безпечного копіювання файлів між комп'ютерами у мережі. SCP забезпечує шифрування та аутентифікацію за допомогою SSH, але відрізняється від SFTP своїм інтерфейсом командного рядка та використанням синтаксису, схожого на команди UNIX.

– TFTP (Trivial File Transfer Protocol) – простий протокол передачі файлів, який використовує UDP для передачі даних. TFTP зазвичай використовується для завантаження файлів на мережеві пристрої, такі як маршрутизатори або комутатори, а також для завантаження програмного забезпечення вбудованих пристроїв.

– AS2 (Applicability Statement 2) – протокол обміну даними, який використовується для безпечної передачі структурованих даних між компаніями через інтернет. AS2 забезпечує шифрування та цифровий підпис даних для забезпечення конфіденційності та цілісності інформації.

– WebDAV (Web Distributed Authoring and Versioning) – набір розширень протоколу HTTP, який дозволяє клієнтам здійснювати розподілене створення, редагування та управління веб-ресурсами, включаючи файли. WebDAV дозволяє працювати з файлами на веб-сервері, ніби вони знаходяться на локальному комп'ютері.

– BitTorrent Protocol – протокол для обміну та розповсюдження файлів у великому масштабі через інтернет. BitTorrent використовує розподілену архітектуру, де користувачі, які завантажують файли (піри), одночасно завантажують та роздають їх іншим користувачам.

– AFP (Apple Filing Protocol) – протокол, який використовується для обміну файлами між комп’ютерами у мережі, які працюють під управлінням операційної системи macOS. AFP дозволяє здійснювати спільний доступ до файлів та жиректорій на комп'ютерах Mac із використанням мережі.

FTP має кілька переваг, які роблять його найпопулярнішим серед усіх інших протоколів:

– Простота використання: FTP – дуже простий протокол, що робить його легким у використанні для користувачів будь-якого рівня навичок.

– Широка підтримка: FTP підтримується багатьма операційними системами та мережевими пристроями, що робить його універсальним рішенням для обміну файлами між різними платформами.

– Ефективність: У порівнянні з деякими іншими протоколами, такими як HTTP, FTP може бути більш ефективним для передачі великих файлів, оскільки він оптимізований саме для цього типу завдань.

– Розширення функціональності: FTP дозволяє створювати складніші структури каталогів та керувати правами доступу до файлів і папок, що робить його корисним для організації великих обсягів даних.

Однак FTP також має свої недоліки, зокрема відсутність шифрування за замовчуванням, що може призвести до проблем з безпекою при передачі конфіденційних даних. Тому у випадках, коли безпека є пріоритетом, можуть використовуватися інші протоколи, такі як SFTP або FTPS.

Протоколу FTP для реалізації системи програм буде достатньо, оскільки програми розроблювались для передачі файлів у локальній мережі, а тому шифрування не є пріоритетною задачею, а час, який буде потрібний на шифрування та дешифрування, буде зекономлено, що прискорить передачу файлів, а всі інші альтернативні протоколи не реалізують у повній мірі передачу файлів, як це реалізовано у FTP.

FTP-сервер – програма, яка приймає та обробляє вхідні запити за протоколом FTP. Список популярних FTP-серверів:

– FileZilla Server – безкоштовний FTP сервер, який підтримується для операційних систем Windows, Mac та Linux, має простий інтерфейс та надійні функції безпеки, такі як шифрування та аутентифікація;

– Xlight FTP Server – FTP сервер, який розроблений для платформи Windows, має гнучкі конфігураційні можливості та підтримує різні функції, включаючи віддалене керування та журналювання подій;

– CompleteFTP – комерційний FTP сервер для платформи Windows, має ряд розширених функцій, включаючи підтримку SFTP, FTPS та HTTPS, а також інтеграцію з Active Directory та LDAP для управління доступом;

– Core FTP Server – FTP сервер для платформи Windows, який відомий своєю швидкістю та ефективністю, має простий у використанні інтерфейс та підтримує різні функції, включаючи SSL-шифрування та журналювання подій;

– ProFTPD – відкритий FTP сервер, який підтримується для різних UNIX-подібних операційних систем, зокрема Linux, мін має гнучкі конфігураційні можливості та підтримує різні розширення, що робить його популярним в середовищах Linux.

FTP-клієнт – програма для підключення до FTP-сервера та віддаленої роботи з файлами. Це може бути спеціальний додаток для роботи по FTP, командний рядок і навіть стандартний файловий менеджер Windows/Linux. Список популярних FTP-клієнтів:

– FileZilla – безкоштовний FTP клієнт для Windows, Mac та Linux, має простий інтерфейс та підтримує багато функцій, включаючи керування чергами завантаження та розрив з'єднання;

– Cyberduck – безкоштовний FTP клієнт для Windows та Mac, який підтримує різні протоколи передачі файлів, включаючи FTP, SFTP та WebDAV;

– WinSCP – безкоштовний FTP та SFTP клієнт для Windows з підтримкою шифрування та інших безпекових функцій;

– Documents by Readdle – безкоштовний файловий менеджер для платформи Mac та iOS, який підтримує різні протоколи передачі файлів, включаючи FTP;

– X-plore File Manager – файловий менеджер для платформи Android, який підтримує різні протоколи передачі файлів, включаючи FTP.

Серед оглянутих поширених програм комплексом (системою) є лише FileZilla, так як тільки вона надає у використання як сервер, так і клієнт. Інші розробники надають лише половину комплексу програм, що для кінцевого користувача може бути не зручним шукати відразу декілька джерел необхідних програм і у кожній розбиратись окремо.

Кінцевому користувачу буде зручно зайти на сайт розробника і завантажити

готовий комплекс усіх необхідних програм та утиліт для зручної роботи. Розглянемо детальніше FileZilla.

FileZilla Client – вільна програма, багатоплатформний клієнт FTP з відкритим кодом. Бінарні коди доступні для Windows, Linux і Mac OS X. Підтримує FTP, SFTP та FTPS. На 2017 рік посідає шосту сходинку серед найпопулярніших завантажень всіх часів на SourceForge.net.

FileZilla Server – FTP-сервер, що підтримується тим самим проектом. Він підтримує FTP та FTPS.

Початковий код FileZilla та файли для завантаження розміщені на SourceForge. Незважаючи на певну схожість назви до Mozilla, FileZilla не має жодного стосунку до Mozilla Project.

FTP-підключення до сервера в FileZilla буває двох типів: швидке та збережене.

При швидкому підключенні потрібно вводити менше даних, але з'єднання не зберігається. Потрібно вводити дані знову щоразу, коли відбувається вхід в програму. Цей спосіб використовується, якщо необхідність підключитись до хосту виникає рідко.

При збереженому підключенні треба вводити більше даних, але це треба зробити лише один раз. Коли наступного разу буде відбуватися вхід до програми, знадобиться лише натиснути одну кнопку і програма сама підключиться до сервера. Цей спосіб зручний, коли робота з файлами на хості є регулярною.

Для підключення потрібно знати логін та пароль від FTP-акаунта на сервері, номер порту і адресу, до якої буде відбуватись підключення – або текстове ім'я сервера (хост), або його IP-адресу або доменне ім'я.

Інтерфейс програми складається із 5 частин. Зверху знаходиться меню програми. Під ними розташовані кнопки керування та налаштування. Далі розташовані поля вводу інформації для швидкого з'єднання із сервером. Після цих полів розташовується поле логера. Вся ця частина інтерфейсу займає не більше 25% інтерфейсу. Увесь інший простір зайнято робочим місцем.

Робочий простір можна поділити умовно на 2 частини: ліву та праву. У лівій частині відображається зміст клієнта, що під'єднався до сервера, а у правій – зміст

сервера.

Зміст умовно ділиться на дві частини: верхній та нижній. У верхній відображається деревовидна структура усього диску (без файлів), у якій відображається поточна відкрита директорія. У нижній відображається увесь зміст поточної відкритої директорії (разом із файлами).

Система розподілення інтерфейсу на дві половини є дуже зручною, особливо коли потребується керування декількома сховищами файлів.

Програма підтримує технологію «drag&drop», а тому достатньо просто перетягнути потрібні файли з лівої половини у праву (для вивантаження на сервер) чи навпаки (для завантаження на клієнт). Програма підтримує створення нових директорій на сервері, перейменування, видалення файлів/директорій тощо.

Інтерфейс програми зображено на рисунку 2.1.

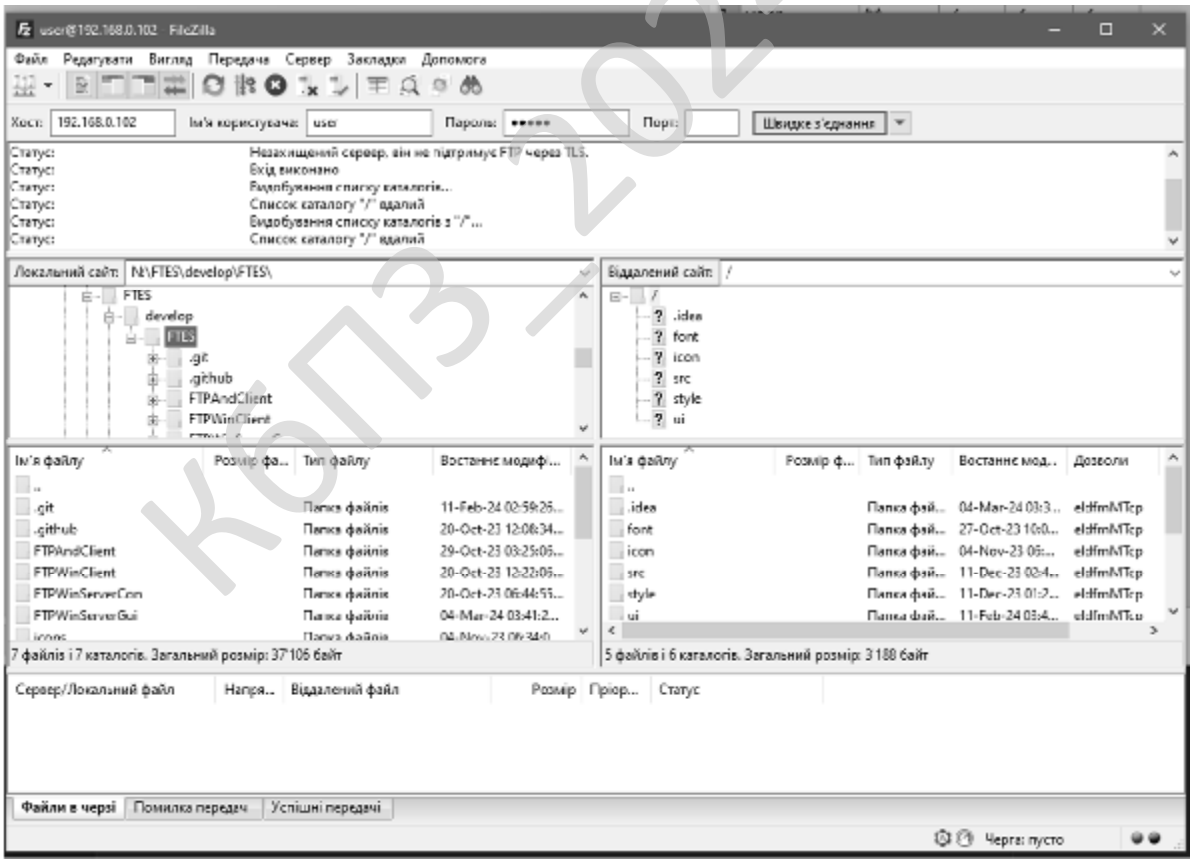

Рисунок 2.1 – Інтерфейс програми FileZilla

Ця програма підтримує передачу великих файлів, але на моїй клієнтській системі при передачі файлів більше 700МБ, коли процес передачі файлу, незалежно від

напрямку передачі файлу (завантаження чи вивантаження), досягає приблизно 700МБ – передача файлу просто зупиняється без повідомлення, яка відбулась помилка. Процес просто завершається і я бачу неповний файл (на клієнті чи сервері).

До того ж ця клієнтська програма має інтерфейс лише для систем на ПК, а для системи Android програма не розроблена. Необхідно шукати іншу клієнтську програму, який для системи Android мало. Легше знайти FTP-сервер для Android. При спробі під'єднатись до серверу на Android, FileZilla не реалізує повну пропускну здатність передачі файлів – швидкість передачі не перевищує 5.6МБ/сек при максимально допустимій швидкості 10МБ/сек.

Цей комплекс програм не надає повної кросплатформеності. Це не є зручно для кінцевого користувача. Та й сама якість роботи програми є посередньою і має баги при передачі даних.

Програма WinSCP є більш функціональною, ніж FileZilla, але він не надає сервер і розроблено лише для ОС Windows. Інтерфейс програми зображено на рисунку 2.2.

Програма надає можливість відкривати декілька з'єднань із декількома серверами одночасно у різних вкладках. У програмі можна відобразити, а можна сховати деревовидну структуру файлової системи, тоді як у FileZilla вона завжди відображається.

Програма реалізує функціонал відображення властивостей файлів/директорій, як у стандартному файловому менеджері Windows. Також через WinSCP можна фільтрувати файли й відображати тільки потрібні.

Обидва менеджери реалізують «чергу». У порівнянні із FileZilla, WinSCP працює більш якісно та швидше. Але його недоліком є відсутня можливість запустити сервер. Якщо користувач отримає програмне забезпечення FTP серверу поганої якості – можуть відбуватись перебої при передачі файлів (передача файлів може просто зупинятись у процесі). Чи, можливо, користувачу потрібен клієнт для іншої операційної системи. Доволі часто виникає ситуація, коли потрібно запустити сервер на комп'ютері, а клієнт на телефоні. У такому випадку жодна із

програм не вирішать проблеми, оскільки не мають клієнту під систему Android.

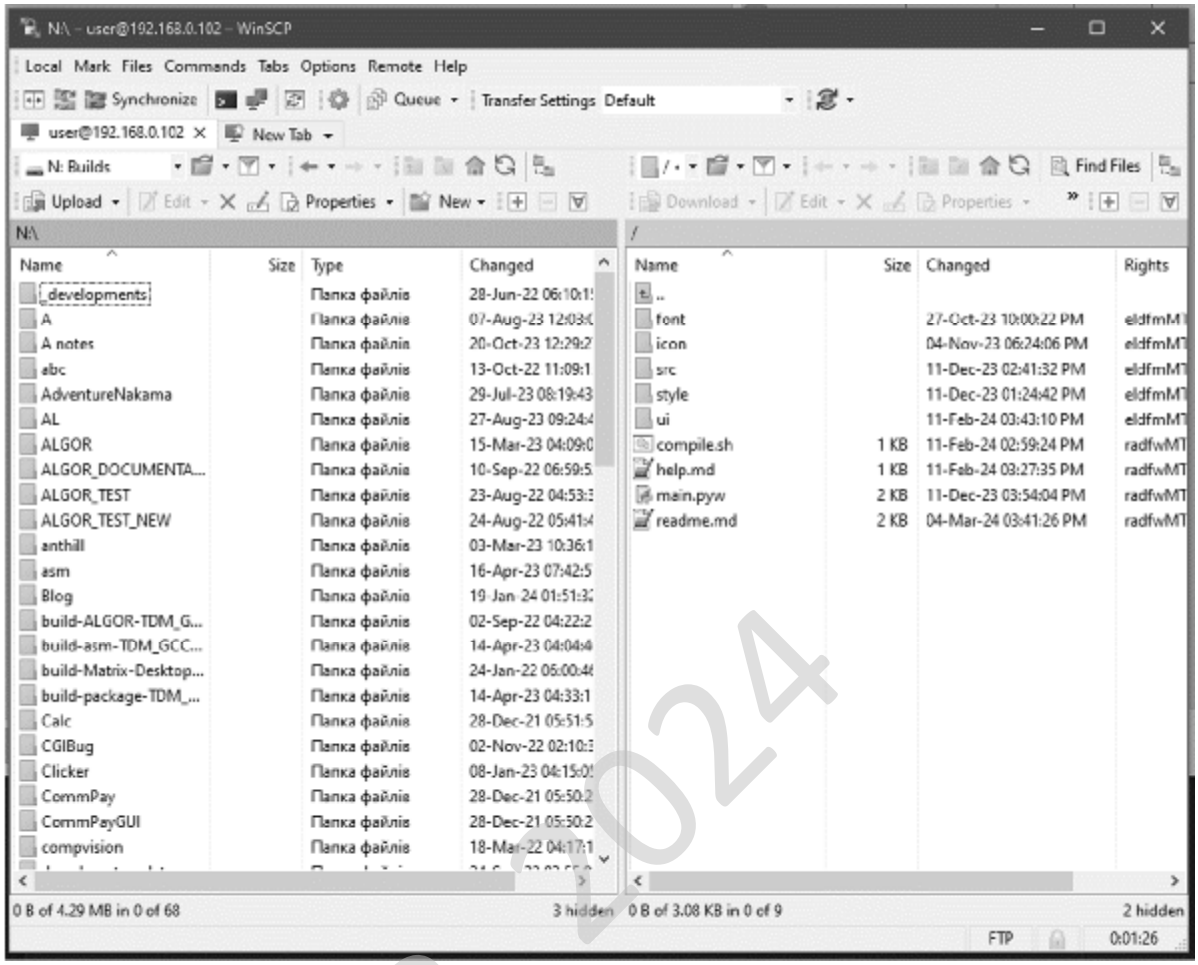

Рисунок 2.2 – Інтерфейс програми WinSCP

І взагалі, у Play Market, є достатньо програм-серверів і нема програм-клієнтів. А більшість серверів дуже поганої якості. У користувача може не бути можливості сісти за комп'ютер (навіть якщо він знаходиться у сусідній кімнаті).

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для системи (комплексу) програм передачі даних потрібно мати доступ із будь-якого пристрою. А тому клієнти мають бути розроблені під усі популярні системи: Linux, Windows, Android. Система Android сильно вирізняється від комп'ютерних систем, а тому інтерфейс буде розроблюватись окремо. Незалежно від інтерфейсу,

ядром клієнтської частини системи буде слугувати модуль реалізації FTP. Щоб не переписувати його під різні мови й не витрачати час, вигідно буде написати його мовою, доступною на усіх трьох системах. Під Android можна писати програми на Kotlin. Мова Java вже давно застаріла і зараз використовується тільки у проектах, які сильно розростили кодову базу й переписувати проект на Kotlin буде економічно не вигідно. Для таких проектів зараз використовується Java. Під системи Linux та Windows доступна для встановлення JVM (Java Virtual Machine), у якій запускаються програми, написані на Kotlin. Тому достатньо написати один модуль та два інтерфейси мовою Kotlin для клієнтів системи.

Реалізація серверу для мобільних застосунків не є оптимальним рішенням, так як телефони не мають достатньо пам'яті. Навіть сучасні моделі наділені 128ГБ та 256ГБ. А у змозі встановити 512ГБ більшість розробників зараз відмовляються від роз'єму карт пам'яті. На вже застарілих моделях можливе встановлення карт пам'яті до 512ГБ і у деяких випадках до 1ТБ. Більше 1ТБ карт пам'яті не існує. А до комп'ютеру може буде під'єднано до чотирьох HDD, об'єми яких зараз складають до 16ТБ у масовому продажі та до 32ТБ експериментальні зразки. Не виключено, що система (комплекс) буде використовуватись на обладнані домашнього файлового серверу, логікою якого буде виступати невеликий комп'ютер типу Raspberry PI, на якому буде встановлено Linux та до якого може бути під'єднано від 4 до 32 дисків у рейд-масиві. У будь-якому випадку, будь-то домашній комп'ютер чи домашній сервер, вигідно підтримувати лише Linux та Windows для сервера системи.

Ще одним аргументом є мобільність телефонів: сервер має бути запущений саме у локальній мережі, а тому якщо телефон під'єднано до мобільного інтернету чи користувач піде далеко від дому – то доступу до FTP-серверу у локальній мережі не буде.

Комп'ютерні системі не обмежені у виборі мови. Для розробки інтерфейсу було обрано найпопулярніший та найпотужніший фреймворк – Qt. Він підтримує дві мови програмування: C++ та Python. На C++ окрім стандартних та фреймворковських бібліотек інших нема. Фреймворк Qt підтримує бібліотеки для роботи з мережею та

мережевим обладнанням, але не підтримує протокол FTP, через що буде витрачено багато обмеженого часу на розробку протоколу. У мові Python вже є готова бібліотека для розробки сервера по протоколу FTP.

Для розробки FTP-серверу було обрано мову програмування Python, графічний фреймворк PyQt та бібліотеку pyftpdlib.

Для розробки FTP-клієнту було обрано мову програмування Kotlin, графічний фреймворк Javafx та бібліотеку org.apache.commons.net.

2.3 Розгорнута постановка завдання

Система (комплекс) програм має складатися із трьох програм: комп'ютерного сервера, комп'ютерного клієнта та мобільного клієнта.

Сервер повинен журналювати всі дії на сервері (хто під'єднався/від'єднався, які файли завантажуються/вивантажуються, які файли видаляються та ким тощо). У сервері має бути реалізована БД користувачів, які можуть бути під'єднані до серверу. На випадок відсутності можливості керувати програмою координатним маніпулятором («мишою») має бути реалізована консоль для вводу команд. Сервер має відображати таблицю користувачів із головними атрибутами та статусом (чи підключений користувач до серверу). Для таблиці мають бути реалізовані методи сортування та пошуку користувачів. Для БД потрібно реалізувати методи (та інтерфейс) додавання, редагування та видалення користувачів. Додатково потрібно реалізувати інтерфейс для перегляду усієї інформації про користувача: усі дозволи, час останнього підключення, останньої сесії, кількість використаного трафіку, особистий журнал користувача тощо. Сервер має надавати інструкцію по роботі із сервером.

Обидва клієнти мають реалізувати файлові менеджери для керуванням файловими системами як клієнта, так і сервера. Для підключення до сервера потрібно реалізувати поля для вводу IP-адреси локальної мережі, на якій із машин запущено сервер та логін з паролем користувача (який є у БД сервера). Якщо вхід до акаунту буде вдалим – клієнти мають дати доступ до файлових менеджерів. Вони мають реалізувати

завантаження/вивантаження файлів, перейменування файлів та їх видалення. Клієнти мають надавати у налаштуваннях перед підключенням вибір кореневої директорії клієнта (на Android – внутрішнє та зовнішнє сховище, а на ПК – буква логічного диску). Клієнти мають відображати процес передачі даних.

Система (комплекс) має вирішити недоліки системи FileZilla: незалежно від розміру, файл має передаватись у повному розмірі та без обмежень по швидкості. Сервер має встановлюватись на ПК чи сервер, а клієнт на будь-який пристрій будь то ПК, сервер, міні-комп'ютер, мобільний пристрій тощо.

Програми мають підтримувати англійську та українську мови.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

FTP – File Transfer Protocol – протокол передачі даних, призначений для організації файлового трафіку між клієнтською файловою системною, та файловою системою серверу.

Останню версію протоколу ще було затверджено у жовтні 1985 року, яка використовується й по нині.

Стандарт протоколу задокументовано у “Request for Comments: 959” (або RFC 959[1]). Тип документу: “RFC - Internet Standard”. FTP складається із двох каналів: керівного та інформаційного. Схема зображена на рисунку 3.1.

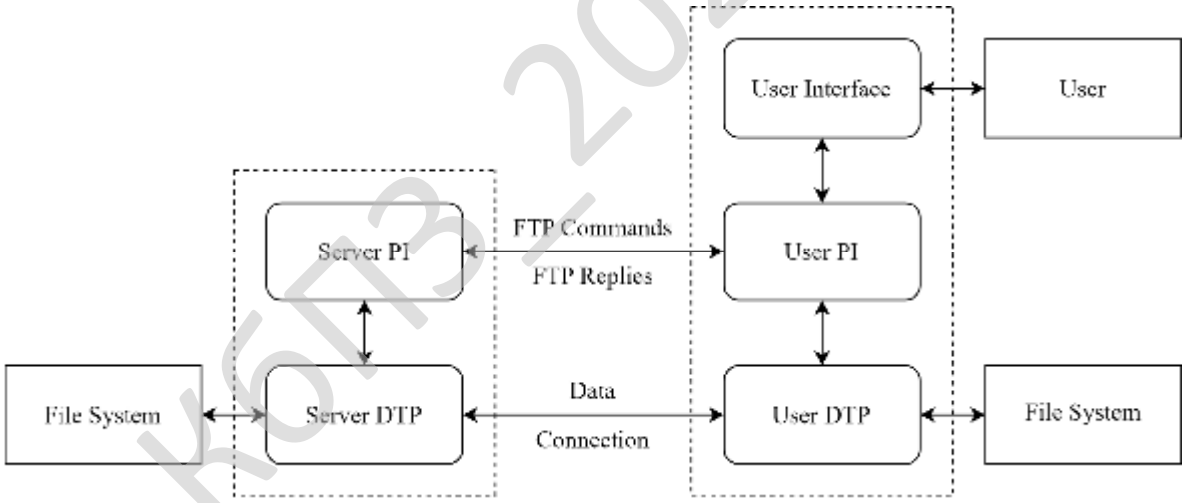

Рисунок 3.1 – Схема FTP (узято із документу RFC 959[1])

Керівний канал використовує порт 21, інформаційний вибирається випадково після 40000 номеру порту. Без встановлення зв’язку у керівному каналі не можливе керування інформаційним каналом.

Для встановлення з’єднання після встановлення контакту на Транспортному рівні моделі OSI по протоколу TCP клієнт має відправити команду на вхід до сервера під певним логіном, список яких встановлено у базі даних сервера. Після логіну клієнт

відправляє пароль і якщо дані вірні – клієнт входить до одного із акаунтів серверу та може керувати інформаційним каналом: створювати та видаляти директорії, дізнатись список файлів сервера, їх характеристику (розмір, дата створення тощо), завантажувати файли на сервер, із серверу, тощо. Для завершення зв'язку із сервером клієнт має відправити команду відключення від серверу.

Для реалізації БД акаунтів було обрано СУБД sqlite і бібліотеку sqlalchemy. Сервер FTP надає 10 дозволів: зміна поточної директорії серверу, отримання списку файлів, отримання файлів із серверу, додавання даних до файлу (якщо існує), видалення файлів/директорій, перейменування файлів/директорій, створення директорій, завантаження файлів на сервер, зміна режиму/дозволів файлу та зміна часу модифікації.

На сервері може бути зареєстровано багато користувачів і щоб заборонити користувачам доступ до чужих середовищ для кожного користувача можна зазначити власну домашню директорію, яка для користувача буде кореневою, за яку користувач не зможе вийти та потрапити до файлів серверу чи файлів інших користувачів.

Не всі користувачі можуть бути відповідальними й хтось може надмірно навантажувати пропускний канал серверу. Для цього за кожним користувачем буде відстежуватись кого коли було створено, хто коли останній раз заходив і чи залогінений користувач зараз на сервері. Також буде відстежуватись загальний час навантаження користувача на сервер (скільки часу загалом користувач завантажував файли на сервер і скільки із серверу), скільки було за цей час передано даних в гігабайтах та скільки було вдалих завантажень/вивантажень.

Якщо користувач має багато прав чи може являтись модератором, адміністратором серверу тощо і йому у домашню директорію користувача кореневу директорію серверу – користувач може випадково видалити/змінити важливий файл. Для відстеження, хто і як змінює файли на сервері, буде логуватись вся діяльність кожного користувача серверу. Логи будуть копіюватись із логів серверу. Кожного разу, як буде виконуватись якась дія над сервером, імітатор консолі буде їх перехвачувати і аналізувати. Якщо запис виконується через дії користувача – запис буде копіюватись

до БД і закріплюватись за користувачем. Далі оригінальний запис виводиться у консоль серверу та у файл. У консолі серверу будуть відображатись логи тільки за поточний запуск програми; у файлі будуть зберігатись усі логи. Для перегляду логів достатньо ввести команду «open APPDATA».

Для виводу логів і вводу команд потрібно реалізувати програмний імітатор консолі. Для його роботи треба реалізувати «Interceptor» (Перехоплювач). Річ у тім, що бібліотека pyftpdlib для логування використовує стандартну бібліотеку мови Python під назвою «logging». Можна створити власний хендлер (обробник) і встановити його у логер. Він є глобальним у цілому для інтерпретатора, тому достатньо встановити налаштований хендлер і все буде працювати, як потрібно. Бібліотека буде робити запис в логер, який є глобальним і відносно встановленого хендлера буде робити запис.

У хендлері буде встановлено власний потік текстових даних. Це і буде Перехоплювач. Він має реалізувати власні методи, але так як його задачею є заміна стандартного stdout, ці методи мають називатись так само. Для звичайного виводу у консоль достатньо визначити методи write та flush.

Кожного разу, коли буде викликатись запис логів у консоль (якщо казати про базовий хендлер, який встановлюється за замовчуванням), операція буде виконуватись з викликом системних команд write та flush, емуляція яких є у ядрі інтерпретатора Python. Замість стандартної логіки метод має лише випустити сигнал, який буде містити переданий у метод готовий запис.

Випущений сигнал має перехвачуватись у імітаторі консолі. Після цього строка додається у програмну консоль, а її копія аналізується і додається у БД. Таким чином бібліотека буде думати, що вона виводить логи у системну консоль, тоді як насправді логи будуть виводитись у програмі у текстовому полі, яке імітує консоль.

На головному вікні програми серверу буде розташовано імітатор консолі, інтерфейс запуску серверу, таблиця користувачів (зі скороченим списком даних) та кнопки керування сервером та БД.

Інтерфейс запуску команди буде містити поле вводу IP адреси у локальній мережі, на якій буде запускатись сервер, та кнопку запуску/зупинки серверу.

По таблиці можна виконувати пошук за іменем та сортувати в прямому та зворотному алфавітному порядку.

Кнопки керування БД запускають інтерфейси додавання, редагування, видалення, та огляду користувачів у БД серверу.

Для створення користувача достатньо ввести логін, пароль, домашню директорію та дозволи. Так як більшість функцій серверу розраховані саме на ім'я акаунту (пошук, сортування тощо) – ім'я буде заборонено до коригування. Видалення акаунту призведе до втрати всіх даних про акаунт у БД; директорія користувача видалена не буде автоматично. Для зміни імені користувача потрібно спочатку створити новий акаунт з новим іменем, потім вручну перенести всі записи у БД зі старого акаунту у новий і видалити старий акаунт. Для автоматизації цього процесу реалізовано «міграції», які додатково ще реалізують нове поле у БД, у якому будуть зберігатись усі старі імена акаунту. У вікні огляду акаунту можна переглянути усі дані із БД про користувача, включаючи усі логи за увесь час.

Створювати акаунти з однаковими іменами заборонено. Це ще одна причина, чому не можна змінювати ім'я, а тільки перестворювати акаунт.

Перейдемо до огляду клієнту. Клієнт має реалізувати два файлові менеджери – для серверної файлової системи та для клієнтської. Складатись інтерфейс буде із трьох частин: віджет підключення, віджет менеджера серверу та віджет менеджера клієнту.

На віджеті підключення розташовано combo box для вибору мови програми (англійська та українська), перемикач між світлою та темною темою, combo box для вибору кореневою директорію клієнтського менеджеру (внутрішня та зовнішня пам'ять), поля вводу IP адреси серверу, імені та пароля акаунту, та кнопка під'єднання.

Менеджери реалізують перемикач «копіювання». При завантаженні/вивантаженні файлу він копіюється і оригінал не видаляється. Якщо світчер вимкнути – після копіювання оригінальний файл буде видалятись.

Якщо під час роботи клієнту файли змінювались із інших програм – може знадобитись періодично оновлювати зміст менеджеру. За це відповідає друга кнопка «синхронізації».

Також менеджери реалізують кнопки створення директорій та видалення файлів/директорій.

Далі реалізується головний функціонал менеджерів. Кожен із менеджерів реалізує список файлів у поточній відкритій директорії, шлях до поточної директорії та кнопку завантаження/вивантаження.

Всі алгоритми клієнту описані у модулі ядра FTP клієнту. Ядро реалізує базовий функціонал клієнту: підключення до серверу, запит змісту поточної відкритої директорії серверу (за замовчуванням після під'єднання це коренева директорія користувача серверу), відкриття вкладеної директорії користувача серверу, повернення до батьківської директорії поточної директорії користувача серверу (вихід до попередньої директорії), перезапит змісту поточної директорії користувача серверу, аналогічні функції для клієнту (працює напряму), методи завантаження та вивантаження файлів і їх рекурсивні оболонки тощо.

Цей базовий функціонал по роботі із FTP сервером та файловою системою клієнту оформлено у окремий модуль, щоб його було можливо легко виокремити й спокійно копіювати у іншу програму – клієнт для іншої платформи. Від початку клієнт мав запускатись на будь-якому пристрої, саме тому було обрано середовище виконання програм JVM та графічний фреймворк Javafx. Клієнт має запускатись як на Android, так і на будь-якій іншій платформі. Система Android підтримує власний інструментарій для розробки додатків під свою систему, який включає багато фреймворків, включаючи графічний. Додатки під Android переписуються для інтеграції нового інтерфейсу. Саме тому розроблюється два додатки, а не один. Головним буде додаток під Android, а другорядним Javafx. Другорядний фрейморк простіше і використовується для розробки програм під комп'ютерні системи (Windows, Linux, MacOS) і взагалі під будь-яку іншу системи (навіть на холодильники та кофемашини, якщо там є встановлена Javafx).

Винятком все ж таки будуть старі мобільні пристрої років так 2004-2009. На них встановлювалась J2ME – Java Platform, Micro Edition (Java ME). Java ME була платформою для мобільних пристроїв, призначеною для запуску програм, написаних

мовою Java. Java ME можна розглядати і як операційну систему, і як платформу для встановлення та запуску програм. Вона являє собою спеціальну платформу для мобільних пристроїв, яка включає JVM і набір API для розробки додатків. Тому її можна охарактеризувати як операційну систему, але у вужчому сенсі – як операційна система для запуску Java-додатків на мобільних пристроях.

Для запуску клієнту на такій платформі потрібно використовувати API, інший графічний фреймворк та набір бібліотек. Але у наш час такі пристрої вже не виробляються і не продаються. Старі телефони продаються на деталі, оскільки жодна модель вже не працює. А сучасні кнопкові мобільні телефони виробляються із встановленням кастомізованого Android.

Інтерфейси будуть із однаковим функціоналом. Відрізнятись вони будуть лише дизайном, так як кожен фреймворк оптимізовано під власну платформу. Розробка під Android складніше, а тому першим буде спроектовано Javafx інтерфейс. Програма складається із невеликої кількості модулів. Це лише ініціалізатор програми, графічний інтерфейс та ядро FTP клієнту. Інтерфейс описується у окремому «.xml» файлі і імпортується модулем логіки графічного інтерфейсу.

Інтерфейс тісно пов’язаний із модулем ядра клієнту. Після запуску програма ініціалізатор запускає інтерфейс, через який користувач може підключитись та використовувати FTP сервер. Кожна кнопка, кожен віджет пов’язано з тім чи іншим методом ядра клієнту.

Приклад. Після підключення менеджери запускають процес запиту файлових систем серверу та клієнту, для чого використовується модуль ядра клієнту, точніше його методи, один з яких звертається до серверу і запитує файли поточної відкритої директорії (так як зараз відбувся процес відключення – це завжди буде коренева директорія домашньої директорії користувача серверу), а другий зчитує файли з корню файлової системи клієнта, який вказується перед підключенням. Результат роботи методів ядра формується у список об’єктів, яких вже оброблюється модулем логіки інтерфейсу, формується список для віджетів менеджерів, встановлюється та оновлюється інтерфейс. Після цього користувач може спостерігати у зручному

графічному вигляді зміст серверу та клієнту.

Аналогічно і з усім іншим функціоналом. Перехід по директоріям файлових систем, перейменування файлів/директорій, їх видалення, переміщення, завантаження, вивантаження тощо. Завжди спочатку відправляється команда на сервер через модуль ядра клієнту і вже залежно від результату оновлюється інтерфейс користувача.

Так як програми, що розроблюються під JVM є, умовно кажучи, кросплатформовими, то і код під Android можна спокійно просто скопіювати. Але так як на цих двох платформах різні фреймворки, то й код графічного інтерфейсу буде відрізнятись. Єдиним незмінним буде ядро клієнту. Цей модуль спокійно копіюється у другу програму і починається розробка другого інтерфейсу. У головному ініціалізаторі вже потрібно отримати де-які дозволи. Система Android старається бути більш захищеною і потребує, щоб програма запитувала дозволів у користувача. У поточному випадку потрібні дозволи на читання пам’яті, запис пам’яті та доступ до пам’яті. При першому запусці ці дозволи не будуть надані, а тому вони будуть запитуватись у користувача. Якщо вони будуть надані – програма буде працювати коректно, інакше – ні. Якщо дозвіл не буде надано, чи з часом його буде знято, то при запуску програми дозволи знову будуть запитуватись.

Одночасно із запитом дозволів ініціалізується інтерфейс. Він дуже схожий на комп’ютерний інтерфейс, але кожен віджет поділено на свої фрагменти. Річ у тім, що екран телефону не такий великий, як монітор комп’ютеру і якщо увесь інтерфейс вмістити на один екран – він буде дуже малим. Тому створюється інтерфейс із навігаційною панеллю знизу. Кожна вкладка буде відповідати за відповідний фрагмент. Ініціалізуються усі фрагменти, але потім усі приховуються окрім головного – підключення до серверу. На цьому фрагменті можна налаштувати тему програми, мову та кореневу директорію клієнту (внутрішня пам’ять чи зовнішня). Для підключення до серверу потрібно ввести IP адресу серверу, логін та пароль користувача та натиснути кнопку «Під’єднатись».

Після успішного підключення до серверу відображається інші фрагменти.

Усього фрагментів не 3 (як очікувалось), а 4. Другий та третій – файлові

менеджери серверу та клієнту відповідно (як у комп'ютерній версії програми). Ці фрагменти повністю відповідають інтерфейсу комп'ютерної програми і виконують один і той самий функціонал.

Четвертий фрагмент, по суті, є другим фрагментом із додаванням лише однієї кнопки, тобто файловим менеджером серверу не тільки із підтримкою вивантаження файлів на сервер, а й із завантаженням файлів із серверу. Цей менеджер підтримує андроідівський SAF менеджер.

SAF – Storage Access Framework (у перекладі: Платформа Доступу до Сховища) – платформа, що дозволяє користувачам передивлятись та відкривати документи, зображення та інші файли. Це вмонтоване у систему (з версії Android v4.4 API 19) ізольоване файлове середовище. Воно захищає файлову систему клієнта. Коли користувачу потрібно вибрати потрібні файли у файловій системі клієнту відкривається SAF – програма, дуже схожа на стандартний андроідівський файловий менеджер, де користувач може вибирати файли (для вивантаження на сервер) чи поточну директорію (для завантаження із серверу). Ця платформа є безпечною, так як вона ізольована він програми і спілкується з нею за певними протоколами. Сама програма не отримує доступу до файлового середовища клієнту, а тому користувач може не хвилюватися, що якісь важливі данні чи сховані файли можуть відправлятись на сервери зловмисникам тощо.

Програма не збирає ніяких даних, файлів користувачів і не відправляє їх мені. Але якщо користувачу не потрібен повноцінний менеджер, а всього-то вибрати швидко декілька файлів для завантаження на сервер, то користувачу буде легше використати саме SAF. За звичайним сценарієм для вивантаження файлів на сервер користувач має спочатку вибрати потрібну директорію на другому фрагменті (файловий менеджер серверу), а потім на третьому фрагменті (файлова система клієнту) вибрати файли/директорії і натиснути кнопку вивантаження, після чого файли почнуть вивантажуватись у ту директорію, яка відкрита на сервері. Для завантаження файлів із серверу аналогічно потрібно спочатку відкрити потрібну директорію у третьому фрагменті, повернутись на другий фрагмент, вибрати файли/директорії і

натиснути кнопку завантаження із серверу. Це не дуже зручно. Але це не всі причини.

Якщо користувач не надасть доступ до файлової системи програмі – третій фрагмент не буде працювати. У цьому випадку, знову ж, допоможе SAF. Важливо відзначити, що SAF не підтримує рекурсію, а тому там не можна виділяти директорії і не важливо, чи є у директорії іще директорії. У SAF можна виділяти тільки файли. Тому такий метод підходить лише для передачі невеликої кількості невеликих файлів! Якщо потрібно завантажити пару файликів, достатньо вибрати їх у четвертому фрагменті, натиснути кнопку завантаження й у SAF, що відкрився, вибрати директорію, куди завантажити файли на телефон. Якщо файли потрібно вивантажити, файли у четвертому фрагменті вибирати не треба. Достатньо відкрити директорію у четвертому фрагменті, натиснути кнопку вивантаження й вибрати пару-трійку файлів на телефоні і вони будуть вивантажені на сервер. Якщо потрібне рекурсивне завантаження/вивантаження великих файлів, то треба надавати доступ до файлової системи клієнту і використовувати програмні менеджери із другого та третього фрагменту.

Для забезпечення безпеки SAF використовує власні андроідівські стандарти кодування URI файлових шляхів. SAF поверне ці URI-шляхи до вибраних файлів/директоії і у цьому випадку програмі прийдеться за цим шляхом відкривати зміст файлу в бінарному форматі й передавати файли потоком у мережі. Насправді, SAF було розроблено, як файловий менеджер для програм типу нотаток, редакторів тощо, де файли не передаються мережею. Наприклад, у блокноті користувач через SAF обирає нотатку, менеджер повертає закодований URI шлях і за ним програма може відкрити зміст файлу для редагування чи читання. Таким чином програма не отримує доступу до файлової системи та важливих системних файлів, що ускладнює потрапляння вірусів, але залишається доступ до змісту файлів, через який користувач може їх передивлятись та редагувати. Тобто за користувачем залишається контроль за усім процесом. Саме тому андроідівський клієнт підтримує як файлову передачу, так і потокову.

У файловій передачі обмежень нема, а у потоковій через можливі помилки у

передачі бажано передавати невеликі файли.

Робота серверу та клієнтів описана. Але не описана їх взаємодія. Як було наведено на схемі в початку розділу, недостатньо встановити з'єднання між сервером та клієнтом. Інтерфейс користувача встановлює з'єднання із клієнтським PI (інтерпретатору протоколу), який встановлює контроль над клієнтським DTP (процесом передачі даних), який у свою чергу має доступ до клієнтської файлової системи. Коли клієнтський PI отримує команду від інтерфейсу користувача на встановлення з'єднання із сервером, клієнтський PI встановлює з'єднання із серверним PI по 21 порту. Серверний PI має контроль над серверним DTP, який має доступ до серверної файлової системи. Якщо встановлення з'єднання успішне і користувач залогінився успішно – встановлюється зв'язок між серверним та клієнтським DTP на порті 40000+. Залежно від того, який встановлено режим передачі даних (активний чи пасивний), каналом DTP буде керувати серверний чи клієнтський PI відповідно. Якщо режим активний – команда від інтерфейсу клієнту відправляється в клієнтський PI, який відправляє запит на завантаження/вивантаження файлів. Якщо запит виконати можливо, то серверний PI відправляє команду на серверний DTP завантажити/вивантажити файли, а клієнтський PI відправляє клієнтському DTP прослуховувати канал. Якщо файли завантажуються – із серверного DTP відправляється команда на клієнтський DTP передати файли, після чого починається передача файлів із клієнтської файлової системи у серверну; якщо вивантажуються – серверний DTP передає команду клієнтському DTP приймати файли після чого починається передача файлів із серверу на клієнт. У випадку із пасивним режимом все відбувається навпаки – каналом даних керує клієнтський PI, а серверний PI встановлює у серверний DTP режим прослуховування. Якщо файли завантажуються – із клієнтського DTP відправляється команда передати файли і починається передача файлів із серверу на клієнт; якщо вивантажуються – відправляється команда із клієнтського DTP на серверний DTP команда приймати файли і починається передача файлів із клієнту на сервер.

Підведемо підсумок: якщо встановлюється активний режим, то команди

користувача передаються на клієнтський PI, а потім на серверний PI, який і керує каналом передачі файлів, а якщо пасивний – команди користувача відправляються на клієнтський PI, який і керує каналом передачі даних напряму. У випадку пасивного режиму серверний PI потрібен лише для встановлення та розірвання зв'язку, перевірки якості зв'язку, визначення процесу передачі завантаження/вивантаження тощо.

3.2 Розробка структурної схеми

Система працює у локальній мережі. До серверу можна отримати доступ, якщо клієнт буде під'єднано до однієї локальної мережі. Клієнт із свого боку має доступ до клієнтської файлової системи, а сервер до серверної та до БД. Між сервером та клієнтом встановлюється керуючий канал, а між файловими системи канал передачі даних. Структурну схему зображено на рисунку 3.2.

Рисунок 3.2 – Структурна схема системи

Так як система розгортається у локальній мережі, шифрування не є необхідним,

особливо коли локальна мережа не під'єднана до інтернету. Це дозволить передавати файли між клієнтом та сервером швидше.

Для під'єднання до серверу клієнт має знати IP адресу та порт серверу, а також ім'я та пароль акаунту. Спочатку встановлюється канал між сервером і клієнтом і якщо IP адреса та порт введені правильно і сервер працює коректно – канал між сервером буде відкрито. Але канал між файловими системами (точніше, між DTP, які ними керують) буде закрито. Для відкриття другого каналу клієнт відразу після встановлення з'єднання входить до акаунту. Для цього він відправляє на сервер команди USER та PASS із вказанням імені та паролю користувача відповідно. Якщо користувач є у БД із таким іменем і паролем – користувач вважається підключеним, а між його файловою системою та файловою системою сервера (точніше, між їх DTP) відкривається канал передачі даних. Залежно від режиму роботи каналом керує чи серверний, чи клієнтський DTP. Якщо ім'я чи пароль неправильні – другий канал не відкривається, а клієнт отримує повідомлення від серверу із кодом помилки. Коли клієнт отримує помилку входу до акаунту – керуючий канал закривається, навіть якщо його було успішно встановлено. Користувач має перевірити введені дані та виправити їх на правильні. Після чого процес підключення повторюється.

Коли підключення відбувається успішно – у БД за користувачем робиться запис, про час його підключення. Кожна дія користувача записується у БД – перейменування, переміщення, видалення, завантаження/вивантаження тощо. Додатково ведеться статистика, скільки часу та трафіку було витрачено на завантаження/вивантаження, скільки було успішно передано файлів; коли користувача було зареєстровано, коли він останній раз підключався та скільки тривала його остання сесія.

Канал передачі даних використовується тільки при завантаженні/вивантаженні файлів. При пасивному режимі серверний DTP встановлюється у режим прослуховування каналу і клієнтський DTP відправляє команду на прийом файлів і самі файли при вивантаженні файлів чи команду на віддачу файлів і отримує файли від серверу при завантаженні. При активному режимі клієнтський DTP встановлюється у

режим прослуховування і сервер команда від клієнту відправляється через сервер на серверний DTP, який відправляє команду клієнтському DTP на прийом файлів і починає відправляти файли при вивантажені та навпаки.

Можна помітити, що операції завантаження та вивантаження для серверу та клієнту є протилежними. Коли файли передаються від клієнту до серверу – файли вивантажуються із клієнту і завантажуються на сервер. Якщо файли передаються від серверу до клієнту – файли завантажуються на клієнт і вивантажуються із серверу. Коли розмова йде лише про користувача – він може спостерігати й керувати лише клієнтом, тому з його точки зору завантаження, це завжди переміщення файлів із серверу на клієнт і вивантаження, це завжди переміщення файлів із клієнту на сервер. Але якщо розглядати схему в цілому, коли річ йде як про клієнт, так і про сервер, ще й розглядаються різні режими роботи, можна легко заплутатись, звідки й куди переміщуються файли при завантаженні та вивантаженні. Тому треба обов'язково завжди уточнювати тип операції, звідки куди передаються дані та в якому режимі (наприклад, файли завантажуються із серверу до клієнту у пасивному режимі).

3.3 Розробка функціональної схеми

Система складається із серверу та клієнту. У сервері головними модулями є ініціалізатор інтерфейсу, центральний модуль інтерфейсу, модуль БД, модуль керування БД, модуль імітатору консолі та модуль ядра серверу. У клієнті головними модулями є ініціалізатор інтерфейсу центральний модуль інтерфейсу, модуль моделі з'єднання, модуль файлової системи, модуль ядра клієнту.

Модуль ядра клієнту виконує ролько клієнтського PI, а модуль файлової системи виконує роль клієнтського DTP. У свою чергу модуль ядра серверу є як і серверним PI, так і серверним DTP. Від початку сервер був консольною програмою і не мав БД та інтерфейсу. Сервер просто запускався та мав одного головного користувача. Коли було розроблено клієнти – було дописано інтерфейс та БД для серверу.

Функціональну схему зображено на рисунку 3.3.

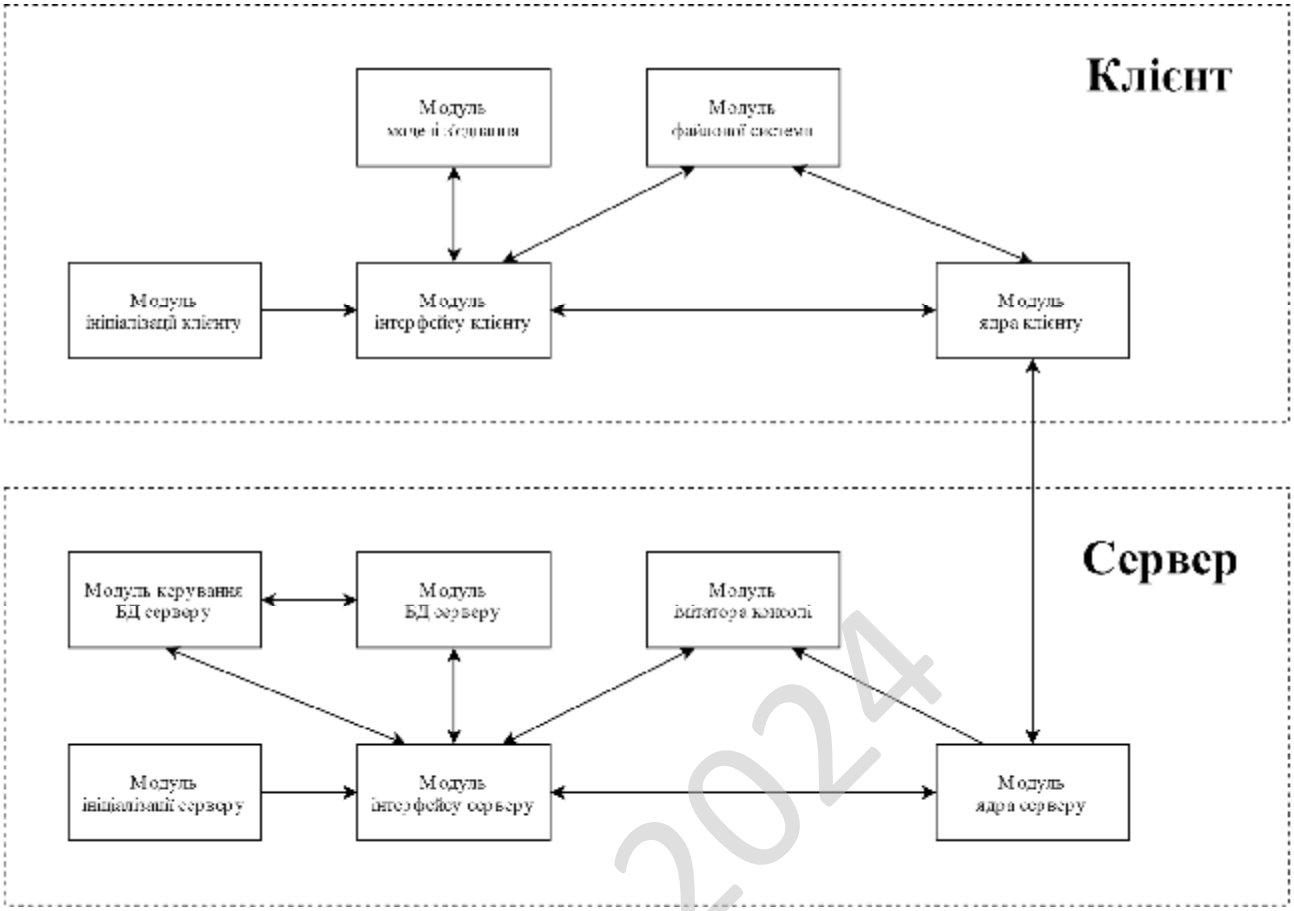

Рисунок 3.3 – Функціональна схема системи

Сервер є програмою, яка лише очікує підключення клієнтів та виконує їх команди. Адміністратор серверу може його запустити/зупинити та керувати користувачами (додавати, змінювати, видаляти та переглядати акаунти). Після запуску інтерфейсу потрібно запустити саме ядро серверу, для чого Адміністратору достатньо лише ввести IP адресу машини в локальній мережі, на якій запускається сервер та натиснути кнопку запуску. Після цього користувачі у локальній мережі зможуть під'єднуватись до серверу. Після запуску інтерфейсу клієнту доступний фрагмент із вводом IP адреси серверу, імені та пароля користувача. Після успішного встановлення з'єднання та підключення клієнт ініціалізує модель з'єднання – модуль, який зберігає логічні змінні, які данні потрібно зберігати – файлову систему серверу, клієнту та з'єднання. Також ініціалізується файлова система клієнту (якщо надано доступ), та файлова система серверу (запитується через ядро клієнту).

Сервер, після відкриття каналу і отримання команди на запит змісту поточної відкритої директорії (при підключенні це завжди коренева директорія користувача), виконує команди напряму із модуля ядра серверу і відправляє запис в імітатор консолі, який перехвачується інтерфейсом і заноситься в БД. Кожна дія, яка виконується сервером, проходить етап логування в імітаторі консолі, записи з якого відправляються в БД.

При відключенні клієнту від серверу модуль файлової системи заморажується у пам’яті, а модуль моделі з’єднання оновлюється так, що у ньому тепер зберігається інформація, що з’єднання із сервером немає і оновлювати файлову систему серверу тепер неможливо, але оновлення файлової системи клієнту можливе, не дивлячись на те, що доступу до неї немає. Неможливе оновлення файлової системи клієнту тільки при запуску клієнту. Після повторного підключення до серверу (того самого чи іншого в цій же локальній мережі) знову всі змінні приймають значення «істини», а модуль файлової системи розморожується. Таким чином при повторному під’єднані до серверу клієнтський менеджер буде зберігати ту директорію, у якій він знаходився у момент відключення і її зміст, а серверний менеджер почнеться знову із домашньої кореневої директорії користувача.

3.4 Розробка діаграми процесів

Запуск серверу ініціалізує інтерфейс, через який Адміністратор може керувати БД серверу та запускати/зупиняти сервер. Після запуску серверного ПЗ ініціалізується консольний буфер – перехоплювач, через який буде працювати програмна консоль. Результат ініціалізації відправляється назад у серверне ПЗ.

Після цього зчитується БД серверу і всі її користувачі додаються в облік ядра серверу. Коли ядро серверу отримує список користувачів, які можуть до нього підключитись, тоді стає можливим підключення клієнтів, оскільки серверу стає можливим не тільки встановлення зв’язку, а й аутентифікації й надання дозволів.

Результат зчитування файлів повертається у серверне ПЗ.

Всі прочитанні дані БД можуть бути передані у віджет відображення даних, звідки Адміністратор, може дізнатись домашню директорію користувача, час та трафік навантаження, дозволи, всі дії над сервером (завантаження/вивантаження, перейменування, видалення тощо). Результат роботи із оглядачем файлів відправляється у серверне ПЗ.

Діаграма процесів зображена на рисунку 3.4.

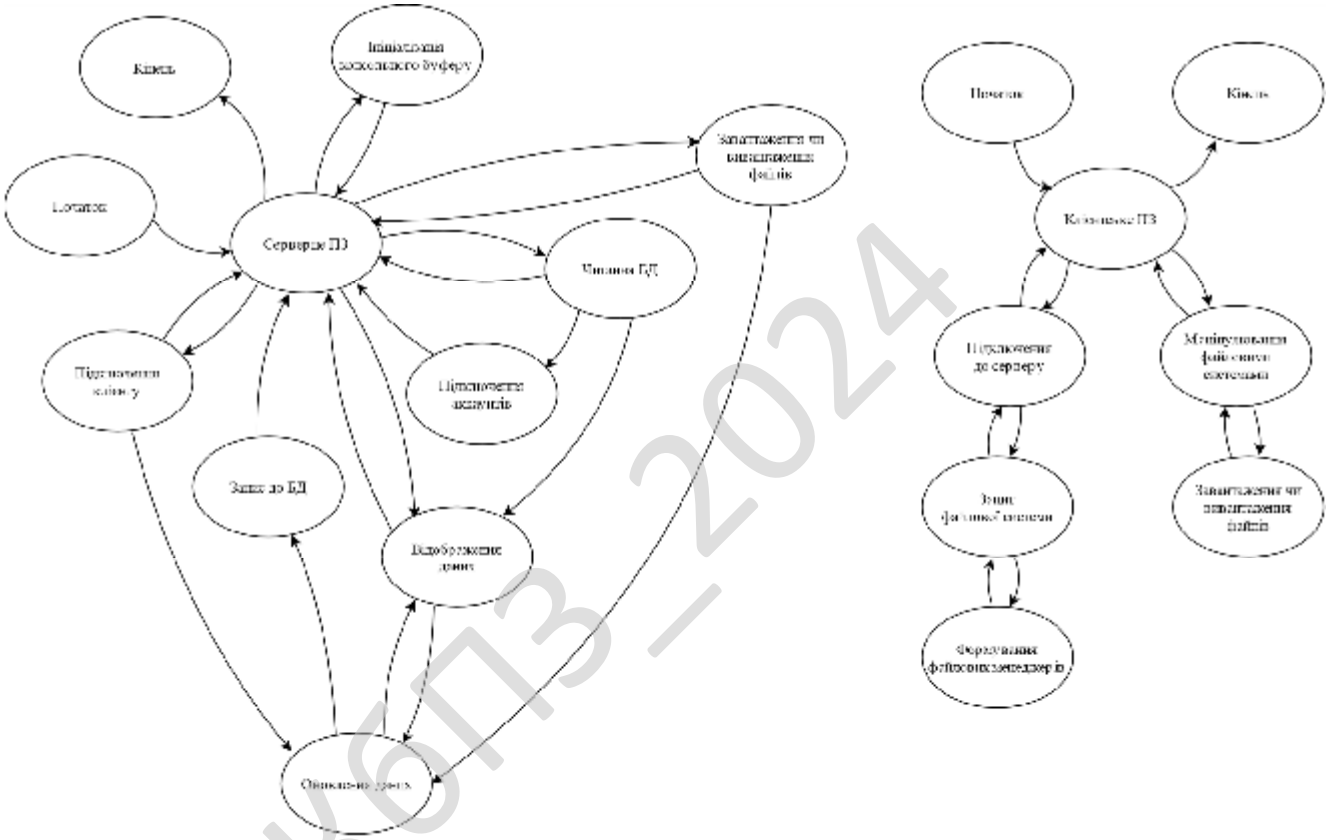

Рисунок 3.4 – Діаграма взаємодії процесів

Адміністратор може змінити деякі дані користувача – пароль, домашню директорію та дозволи. Ім’я, статистику та логи змінити неможливо. Після зміни деяких даних відбувається запис до БД.

Після успішного під’єднання клієнту також відбувається запис до БД. Результат під’єднання відправляється у серверне ПЗ.

Клієнт може запустити завантаження/вивантаження файлів, які також записуються до БД. Кожному запису до БД передує їх оновлення. Результат оновлення

впливає на віджет відображення даних БД. Результат передачі даних передається у серверне ПЗ.

Серверне ПЗ (центром якого є головний інтерфейс) постійно перевіряє усі результати, які відправляють йому процеси. Якщо консольний буфер не вдалось ініціалізувати – надається звіт із результатом, після чого потрібно виправити помилку у програмі. Якщо файл завантажено/вивантажено безуспішно – відправляється повідомлення клієнту про помилку після чого клієнт має перевірити, чи не відправляє він поламані файли та заново запустити процес передачі файлів. Помилка могла виникнути як у процесі передачі, так і через зломані файли. Якщо підключити клієнт не вдалось – канал даних не відкривається і з'єднання розривається. Якщо прочитати чи оновити дані в БД не вдається – відправляється звіт Адміністратору з помилкою, після чого її потрібно виправити.

Для завершення програми потрібно зупинити сервер. Якщо до серверу під'єднано клієнти – всі з'єднання розриваються примусово зі сторони серверу. Коли сервер зупинено – можна завершати програму. При завершені всі акаунти із ядра серверу видаляються і програма завершає свою роботу.

Запуск клієнту ініціалізує інтерфейс, що і є клієнтським ПЗ. Спочатку користувачу доступні лише елементи налаштування програми та підключення до серверу. Після успішного підключення клієнт запитує зміст поточної директорії серверу (якою буде домашня коренева директорія користувача) та ініціалізує файлові менеджери.

Після отримання доступу до файлових менеджерів серверу та клієнту користувач може ними керувати – переміщати, перейменовувати видаляти файли тощо. Також клієнт може запустити передачу файлів між сервером та клієнтом.

Коли роботу із сервером завершено – користувач може відключитись від серверу та завершити програму.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Для реалізації серверу було використано бібліотеку pyftpdlib. Вона реалізує FTP, але не являє собою повноцінний додаток. Це лише каркас, на якому можна побудувати власний додаток. Можна реалізувати свій клас, який буде успадкований від класу FTPHandler та реалізує власні команди. Так можна розширити протокол, але у такому випадку із сервером буде працювати повноцінно лише клієнт, розроблений під цей сервер. Інші клієнти будуть не повністю використовувати сервер. А клієнт, розроблений під цей сервер буде несумісним із іншими серверами.

Для реалізації БД було використано бібліотеку sqlalchemy. За СУБД було обрано sqlite. БД складається із наступних колонок:

```
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True)
    online = Column(Boolean)
    username = Column(String)
    password = Column(String)
    home_dir = Column(String)
    date_of_creation = Column(DateTime)
    date_of_change = Column(DateTime)
    last_login_date = Column(DateTime)
    login_time = Column(Interval)
    upload_count_successful = Column(Integer)  # STOR
    upload_size = Column(Integer)  # STOR
    upload_time = Column(Float)  # STOR
    download_count_successful = Column(Integer)  # RETR
    download_size = Column(Integer)  # RETR
    download_time = Column(Float)  # RETR
    permission_CWD = Column(Boolean)
    permission_LIST = Column(Boolean)
```

```
    permission_RETR = Column(Boolean)
    permission_APPE = Column(Boolean)
    permission_DELE = Column(Boolean)
    permission_RNFR = Column(Boolean)
    permission_MKD = Column(Boolean)
    permission_STOR = Column(Boolean)
    permission_CHMOD = Column(Boolean)
    permission_MFMT = Column(Boolean)
    user_logs = Column(Text)
```

Модуль керування БД реалізує клас, що ініціалізує та читає БД:

```
self.engine = create_engine(f"sqlite:///{program_dir}\\users_database.db")
Base.metadata.create_all(self.engine)
self.Session = sessionmaker(bind=self.engine)
self.session = self.Session()
```

Методи цього класу реалізують зчитування даних конкретного користувача, чи усіх користувачів, перевірку, чи існує користувач у БД (перевірка відбувається за іменем), створення користувача, встановлення статусу користувача (чи під'єднано його до серверу, чи ні), оновлення даних користувача, перерахування статистики, встановлення дати та часу, видалення користувача та закриття БД. Так як БД постійно використовується інтерфейсом, то й відкрита вона постійно й закривається лише при завершенні програми.

Клас перехоплювача консольного виводу являє собою звичайний об'єкт, який випускає сигнали:

```
class Interceptor(QObject):
    writing = pyqtSignal(str)

    def __init__(self):
        super().__init__()

    def write(self, text):
        self.writing.emit(text)

    def flush(self):
        pass
```

Як тільки має відбутись запис до консолі, якої не існує, «перехоплювачем» випускається сигнал, який перехвачується в інтерфейсом серверу. Алгоритм аналізу запису зображено на рисунку 4.1.

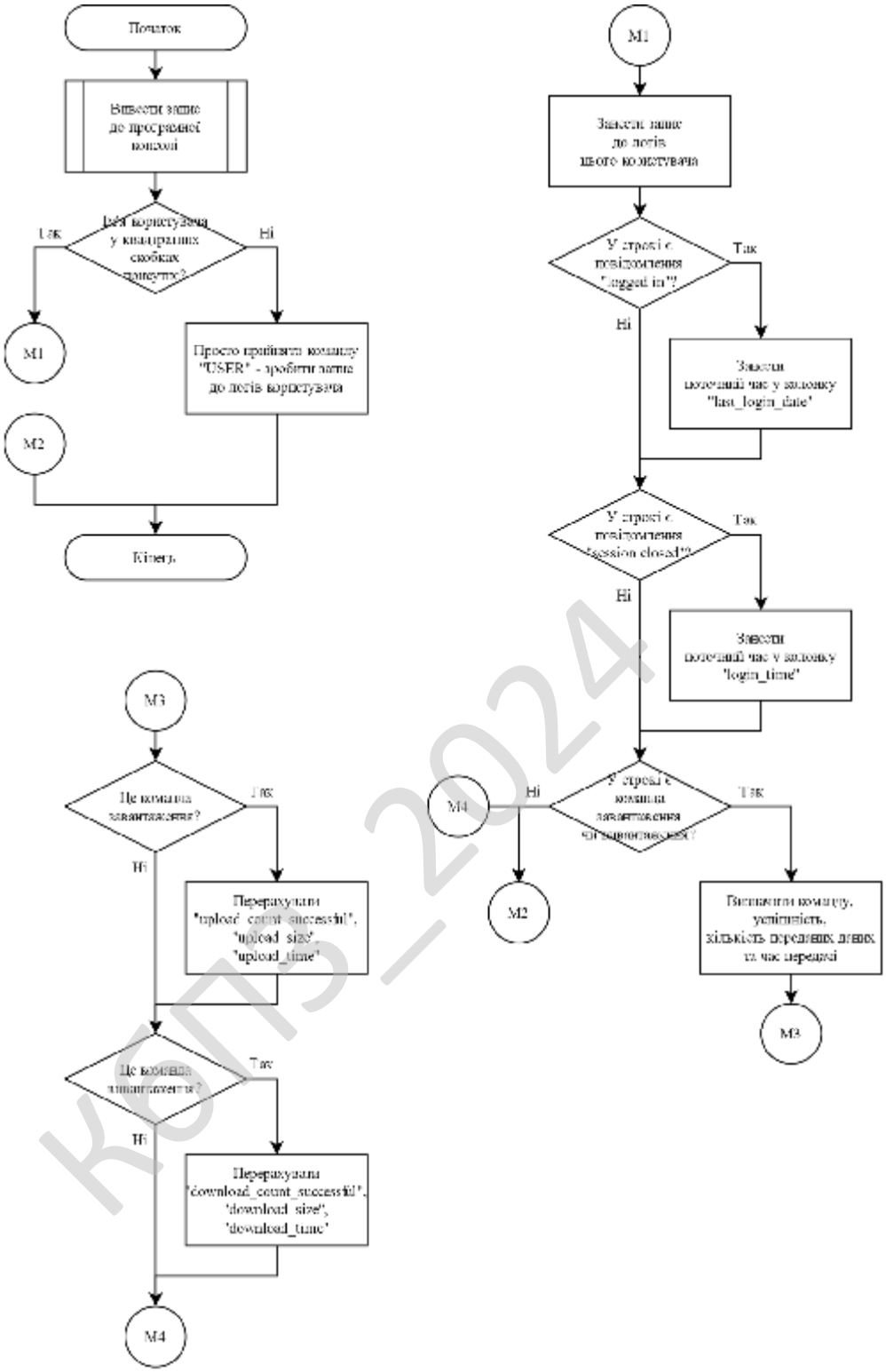

Рисунок 4.1 – Блок-схема аналізу запису

Перевіримо роботу серверу. Я запускаю сервер, відкриваю клієнт на телефоні, підключаюсь до серверу, обираю файл «сюжет.txt», завантажую його на сервер, відключаюсь від серверу та зупиняю сервер. Логи, які я отримав у консолі:

```
[INFO 2024-03-10 14:17:51] concurrency model: multi-thread
[INFO 2024-03-10 14:17:51] masquerade (NAT) address: None
[INFO 2024-03-10 14:17:51] passive ports: None
[INFO 2024-03-10 14:18:34] 192.168.0.103:39706-[]
FTP session opened (connect)
[INFO 2024-03-10 14:18:34] 192.168.0.103:39706-[user]
USER 'user' logged in.
[INFO 2024-03-10 14:19:29] 192.168.0.103:39706-[user]
STOR N:\FTES\develop\FTES\FTPWinServerGui\Сюжет.txt
completed=1 bytes=3906 seconds=0.016
[INFO 2024-03-10 14:19:37] 192.168.0.103:39706-[user]
CWD N:\FTES\develop\FTES\FTPWinServerGui 250
[INFO 2024-03-10 14:20:38] 192.168.0.103:39706-[user]
FTP session closed (disconnect).
[INFO 2024-03-10 14:20:46]
>>> shutting down FTP server (0 active workers) <<<
```

Результат аналізу логів зображено на рисунку 4.2.

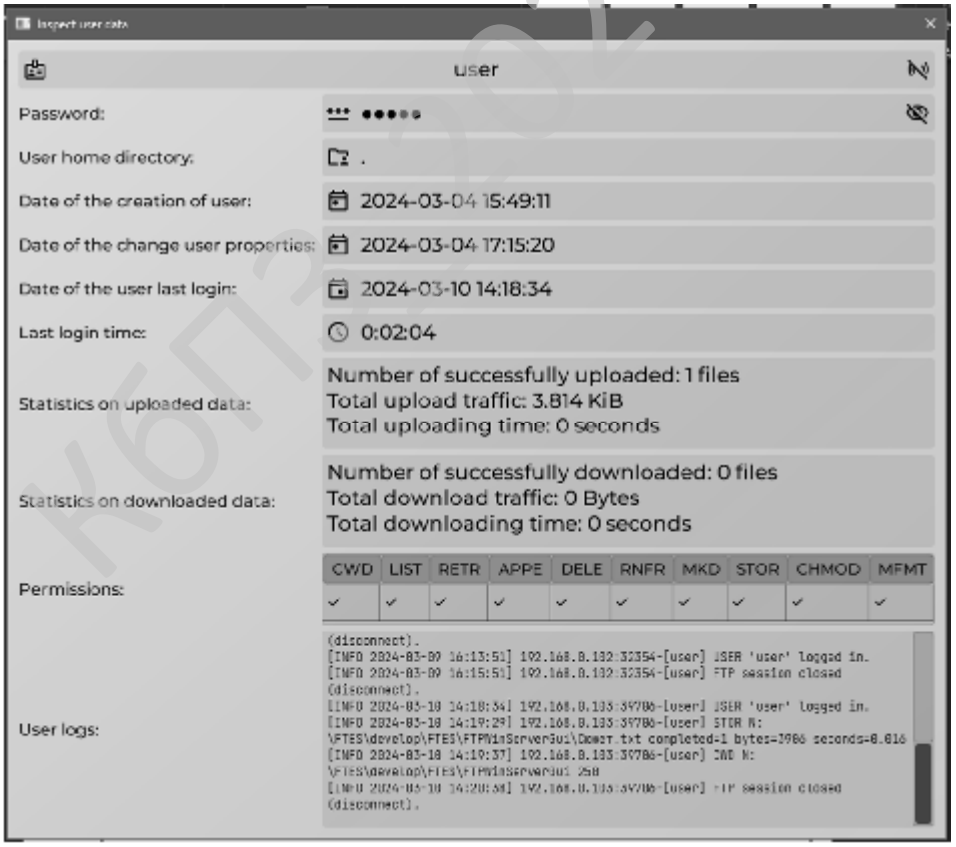

Рисунок 4.2 – Скріншот даних користувача

Проаналізуємо результат. До логів користувача заносяться усі строки, які у консолі мають ім’я користувача у квадратних скобках. Із запису про підключення

користувача виокремлюється час та заноситься у комірку «Дати останнього підключення користувача». Із запису про вивантаженням файлу виокремлюється вся важлива інформація: кількість успішно переданих файлів, їх загальний трафік та час завантаження. Статистика відображається з погляду користувача, тобто передача із пристрою на сервер – вивантаження (upload), а із серверу на пристрій – завантаження (download). Після відключення клієнту аналізатор визначає різницю між часом відключення та підключення та заносить її до комірки «тривалості останнього сеансу». Алгоритм описано правильно і це підтверджено на практиці.

Запис виконується відразу після виконання команди користувача. Якщо команду було неможливо виконати й виникла помилка – користувачу відправлявся результат, а у логи записувався код помилки та можлива причина. Коли ж помилка не виникає й операція виконується успішно – у логи та в БД записується, яка дія й ким була успішно виконана. Підтверджено.

Далі буде протестовано рекурсивне переміщення із серверу на клієнт (рекурсивне завантаження). Для цього я використаю директорію із властивостями, зображеними на рисунку 4.3 та один додатковий файл.

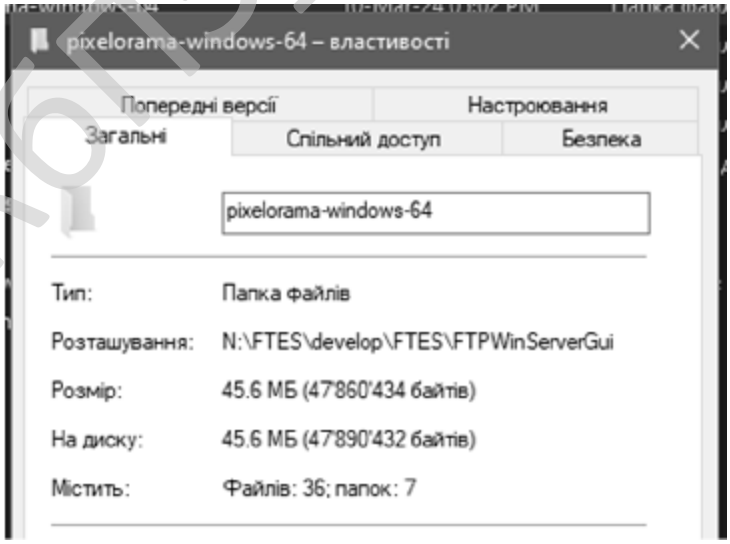

Рисунок 4.3 – Скріншот властивостей завантажуваної директорії

Увесь процес завантаження файлів логується. У консолі серверу, у логах серверу та у логах користувача буде збережено наступні записи:

```
[INFO 2024-03-10 15:15:13] 192.168.0.103:39902-[]
FTP session opened (connect)
[INFO 2024-03-10 15:15:13] 192.168.0.103:39902-[user]
USER 'user' logged in.
[INFO 2024-03-10 15:15:26] 192.168.0.103:39902-[user]
CWD N:\FTES\develop\FTES\FTPWinServerGui 250
[INFO 2024-03-10 15:15:38] 192.168.0.103:39902-[user]
CWD N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64 250
[INFO 2024-03-10 15:15:43] 192.168.0.103:39902-[user]
RETR
N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64\Pixelorama.exe
completed=1 bytes=37732352 seconds=4.671
[INFO 2024-03-10 15:15:44] 192.168.0.103:39902-[user]
RETR
N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64\Pixelorama.pck
completed=1 bytes=10064720 seconds=1.219
[INFO 2024-03-10 15:15:44] 192.168.0.103:39902-[user]
CWD
N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64\pixelorama_data
250
[INFO 2024-03-10 15:15:44] 192.168.0.103:39902-[user]
CWD
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes 250
[INFO 2024-03-10 15:15:44] 192.168.0.103:39902-[user]
CWD
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes\Grass 250
[INFO 2024-03-10 15:15:44] 192.168.0.103:39902-[user]
RETR
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes\Grass\~Grass1.png
completed=1 bytes=567 seconds=0.0
[INFO 2024-03-10 15:15:44] 192.168.0.103:39902-[user]
RETR
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes\Grass\~Grass2.png
completed=1 bytes=573 seconds=0.0
…
[INFO 2024-03-10 15:15:45] 192.168.0.103:39902-[user]
CWD N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64 250
[INFO 2024-03-10 15:15:45] 192.168.0.103:39902-[user]
```

```
DELE
N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64\Pixelorama.exe
250
[INFO 2024-03-10 15:15:45] 192.168.0.103:39902-[user]
DELE
N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64\Pixelorama.pck
250
[INFO 2024-03-10 15:15:45] 192.168.0.103:39902-[user]
CWD
N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64\pixelorama_data
250
[INFO 2024-03-10 15:15:45] 192.168.0.103:39902-[user]
CWD
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes 250
[INFO 2024-03-10 15:15:45] 192.168.0.103:39902-[user]
CWD
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes\Grass 250
[INFO 2024-03-10 15:15:46] 192.168.0.103:39902-[user]
DELE
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes\Grass\~Grass1.png 250
[INFO 2024-03-10 15:15:46] 192.168.0.103:39902-[user]
DELE
N:\FTES\develop\FTES\FTPWinServerGui\
pixelorama-windows-64\pixelorama_data\Brushes\Grass\~Grass2.png 250
…
[INFO 2024-03-10 15:15:46] 192.168.0.103:39902-[user]
RMD N:\FTES\develop\FTES\FTPWinServerGui\pixelorama-windows-64 250
[INFO 2024-03-10 15:15:46] 192.168.0.103:39902-[user]
RETR N:\FTES\develop\FTES\FTPWinServerGui\Сюжет12345.txt
completed=1 bytes=3906 seconds=0.015
[INFO 2024-03-10 15:15:46] 192.168.0.103:39902-[user]
DELE N:\FTES\develop\FTES\FTPWinServerGui\Сюжет12345.txt 250
[INFO 2024-03-10 15:15:55] 192.168.0.103:39902-[user]
FTP session closed (disconnect).
```

Обов’язково потрібно перевірити статистику. Якщо відкрити перегляд даних користувача, то в категорії «Statistics on downloaded data» буде відображено

нову інформацію. Раніше всі характеристики були по нулям, так як користувач нічого не завантажував із серверу. Тепер дані змінились на:

```
Number of successfully downloaded: 37 files
Total download traffic: 45.647 MiB
Total downloading time: 5 seconds
```

У статистиці 37 файлів, тому що разом із програмою, що складається із 36 файлів було завантажено іще один файл. Увесь цей функціонал можливий завдяки реалізації методів «Процес завантаження списку файлів» (зображено на рисунку 4.4; далі буде називатись «Процес 1»), «Процес завантаження елементу» (зображено на рисунку 4.5; далі буде називатись «Процес 2») та «Процес рекурсивного видалення елементів списку» (зображено на рисунку 4.6; далі буде називатись «Процес 3»). Усі ці Процеси можна використати окремо один від одного. Наприклад, якщо користувач хоче просто видалити якісь файли чи директорії на сервері чи клієнті, то для цього застосовується лише «Процес 3» для виділених файлів/директорій через натискання відповідної клавіші, на якій зображено смітник. А для завантаження одного файлу чи рекурсивного завантаження однієї директорії вистачить «Процесу 2». Але у списку менеджеру містяться не тільки файли й директорії, а й службові посилання, наприклад «..», яка повертає у батьківську директорію. До того ж завантаження та вивантаження – процес копіювання, а тому якщо користувачу потрібно перемістити файл із серверу на клієнт (чи навпаки), то після копіювання файл (директорію) потрібно вручну видалити, а завантаження батьківської директорії може призвести до неочікуваного результату. Для цього й було написано обгортку «Процес 1», яка враховує можливість вибору батьківської директорії та потреби видалення файлів після копіювання.

«Процес 1» рекурсивно проходить по всім обраним файлам. На початку він перевіряє, чи є серед обраних батьківська директорія і якщо вона знаходиться – вона просто ігнорується. Після цього викликається «Процес 2». Після того, як цей процес відпрацює, перевіряється, чи увімкнуто режим копіювання. Якщо ні – поточний завантажений файл/директорія видаляються. Після цього цикл

переходить до наступного файлу і так до тих пір, поки не буде перебрано усі елементи. Після виконання усіх Процесів над усіма обраними файлами завантаження/вивантаження завершується.

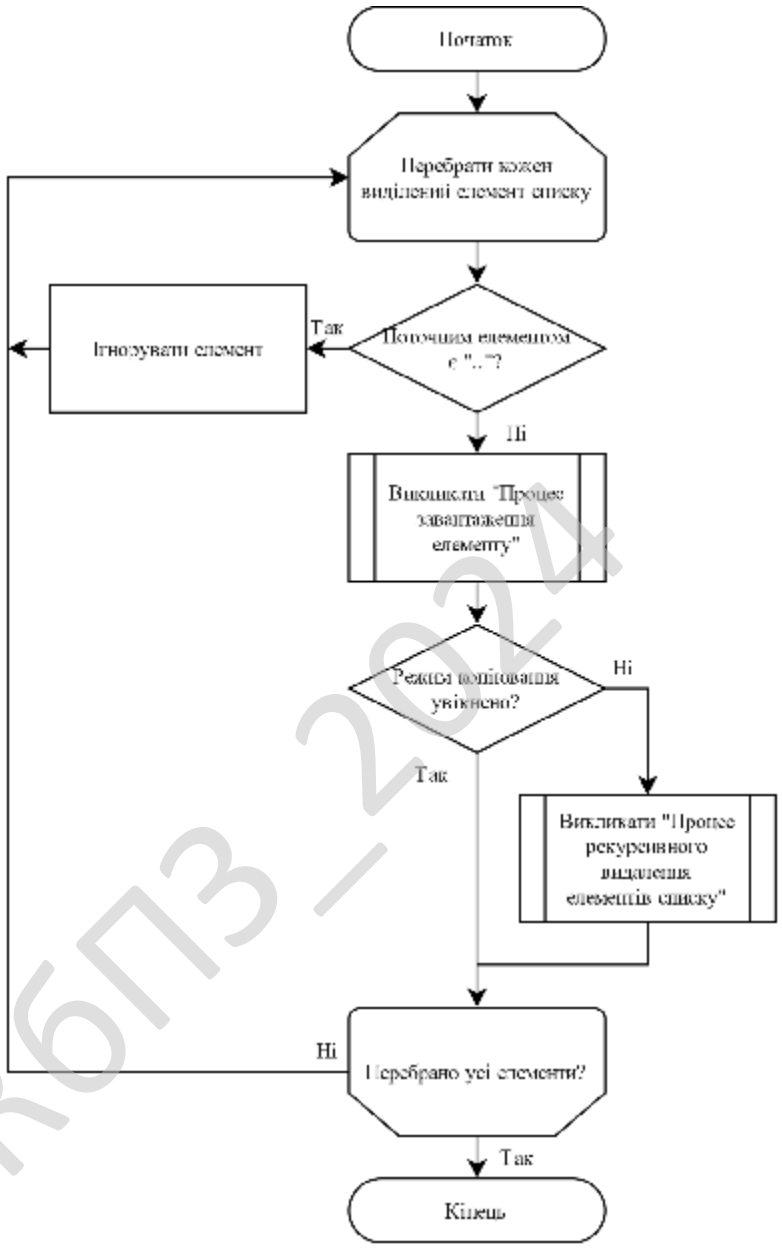

Рисунок 4.4 – Блок-схема «Процесу завантаження списку файлів»

«Процес 2» зберігає до пам’яті шлях до файлу/директорії на сервері та шлях, де файл/директорія має розташовуватись на клієнті. Після чого робиться перевірка, чи є цей елемент файлом. Якщо так – він просто копіює файл із серверу на клієнт. Інакше – створюється відповідна директорія на клієнті та викликається «Процес 2» для кожного вкладеного елементу.

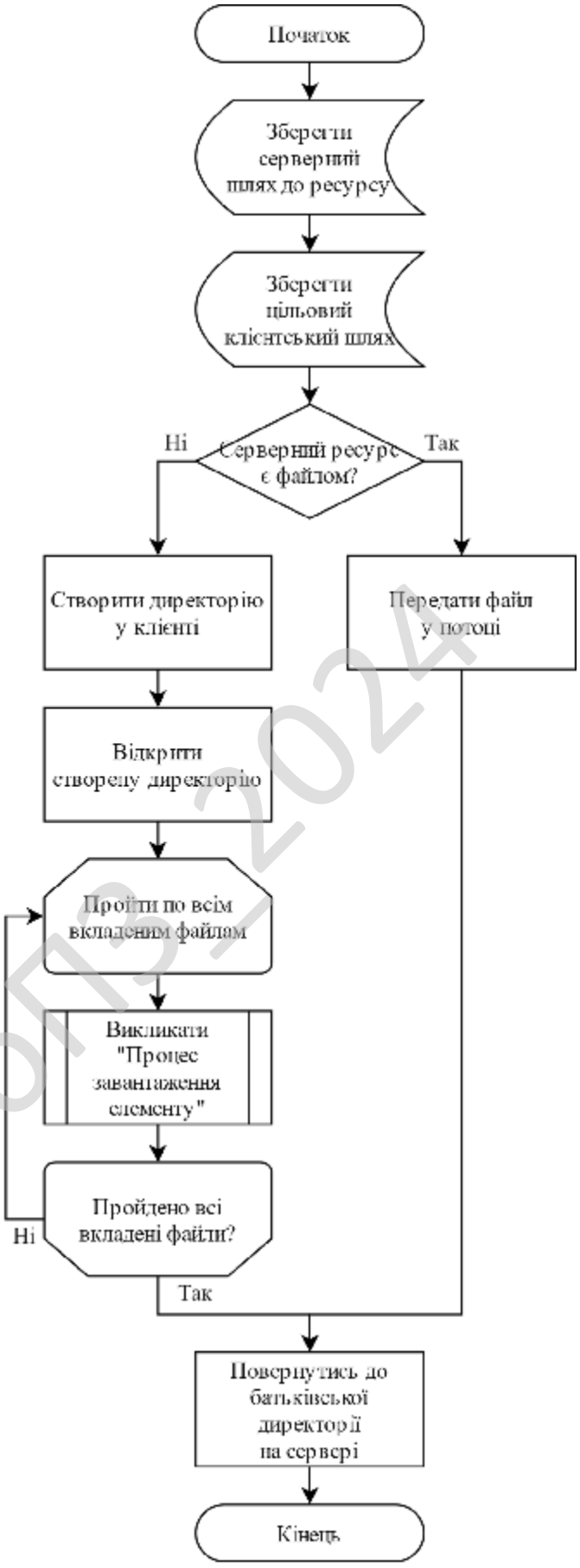

Рисунок 4.5 – Блок-схема «Процесу завантаження елементу»

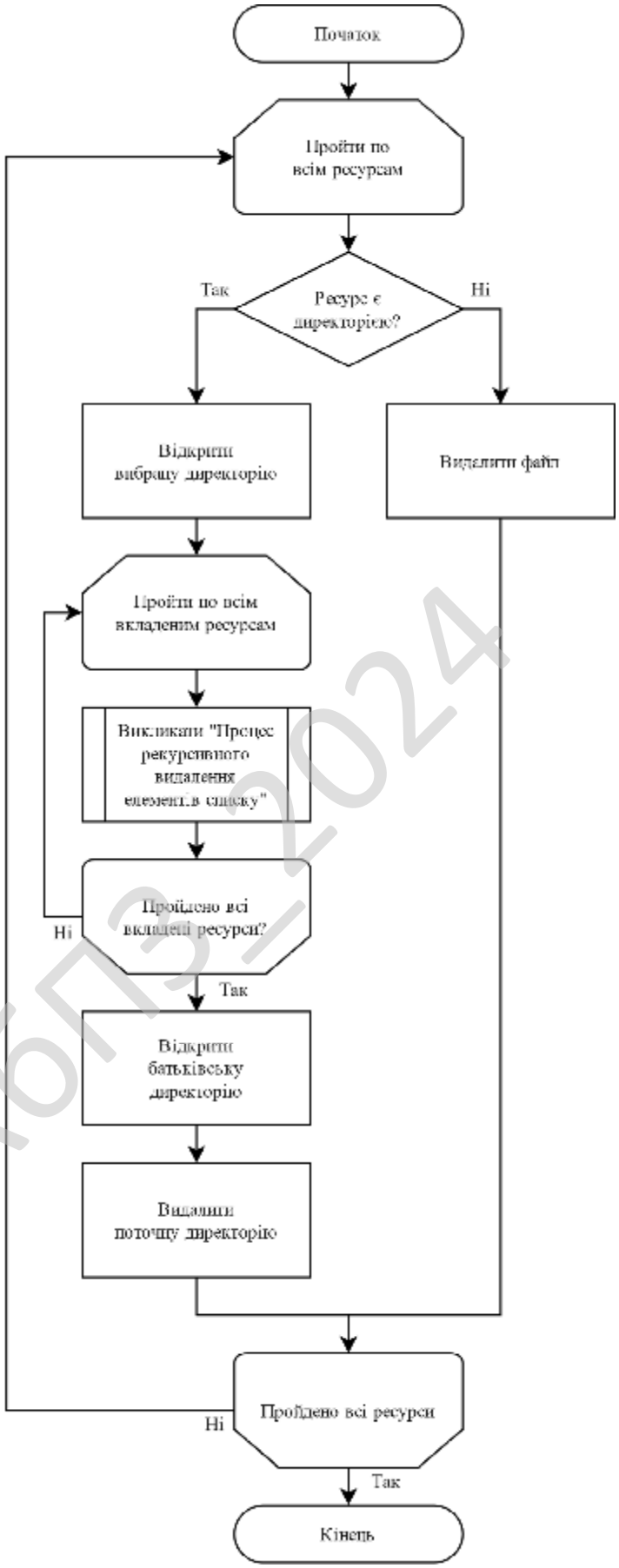

Рисунок 4.6 – Блок-схема «Процесу видалення елементів списку»

«Процес 2» зберігає до пам’яті шлях до файлу/директорії на сервері та шлях, де файл/директорія має розташовуватись на клієнті. Після чого робиться перевірка, чи є цей елемент файлом. Якщо так – він просто копіює файл із серверу на клієнт. Інакше – створюється відповідна директорія на клієнті та викликається «Процес 2» для кожного вкладеного елементу.

«Процес 3» є рекурсивним сам по собі і приймає список файлів/директорій. Але в межах даного алгоритму він приймає список лише із одного файлу/директорії, яку тільки що було завантажено. Повноцінно він використовується, коли користувач видаляє декілька файлів/директорій. Цей алгоритм проходить по всім вибраним елементам і перевіряє, чи є вони файлами. Якщо так – просто видаляє їх. Інакше – відкриває вибрану директорію і проходиться по всім вкладеним елементам. Після видалення всіх вкладених елементів алгоритм повертається до початкової директорії і видаляє її. Чому алгоритм саме такий? Не можна на сервері видаляти не порожні директорії. Якщо у директорії є якийсь зміст – потрібно спочатку його видалити.

За аналогічним принципом побудовано і вивантаження файлів, тільки у такому випадку сервер та клієнт змінюються місцями. Цільовим сховищем становиться сервер, а ресурсним – клієнт.

Як можна було визначити із результату роботи аналізатора, завантаження і вивантаження працює успішно. Можна вимкнути режим копіювання і файли будуть видалятись після передачі правильно. Алгоритм реалізовано правильно.

Для завантаження/вивантаження файлів потрібно їх якось вибрати. Але як вказати програмі, які файли завантажити чи куди файли вивантажити, якщо клієнтська програма нічого не знає про серверну файлову систему? Для цього реалізуються файлові менеджери. Вони мають відображати зміст поточної відкритої директорії серверу та клієнту. Для роботи серверного менеджера потрібно реалізувати запит алгоритм запиту змісту серверу. Даний метод викликається завжди, коли клієнт підключається до серверу, коли відкривається вкладена директорія чи оновлюється зміст поточної. Запускається метод у потоці:

```
// Loading the Server File System
```

```
var serverFiles: List<String>? = null
val mainServingThread = Thread {
    serverFiles = serv.openServerDirectory(filePath.text.toString())
}
mainServingThread.start()
mainServingThread.join()
```

Отриманий список адаптується у правильний формат, із якого можна буде сформувати й відобразити графічний віджет:

```
// Converting a list of file names to a list of file objects
var fileItems = serverFiles!!.map { fileName ->
    val imageUri: Uri
    var info = ""
    val isDir: Boolean
    if (fileName == "..") {
        imageUri = Uri.parse(
            "android.resource://" + requireContext().packageName +
            "/" + R.drawable.undo
        )
        info = "Return"
        isDir = true
    } else {
        isDir = fileName.contains("/")
        imageUri = if (isDir) {
            Uri.parse(
                "android.resource://" +
                requireContext().packageName +
                "/" + R.drawable.folder
            )
        } else {
            Uri.parse(
                "android.resource://" +
                requireContext().packageName +
                "/" + R.drawable.file
            )
        }
        val additionalServingThread = Thread {
            info = serv.getServerFileSize(
                ImportantData.serverRoot +
                ImportantData.serverPath +
                fileName
            ).toString()
```

```
            }
            additionalServingThread.start()
            additionalServingThread.join()
        }
        FileItem(imageUri, fileName, info, isDir, false)
    }
```

Для роботи віджету потрібно знати 5 характеристик: шлях до іконки елементу, назва елементу, властивості елементу, чи є елемент директорією та чи є елемент вибраним.

Список імен переформатовується в список списків даних файлу. Знаючи ім’я файлу, можна отримати інші дані. Тому по стандарту ім’я відоме завчасно перед формування віджету. Іконок існує всього дві – для файлу та директорії. Якщо це батьківська директорія – встановлюється іконка директорія, інформація «Повернення» та логічна змінна зберігає «true», вказуючи, що це директорія. Інакше – визначається тип (якщо це директорія – вкінці буде символ «/»), іконка (залежить від визначеного типу) та інформація (запитується з серверу; відображає інформацію, скільки займає файл/директорія пам’яті). Остання змінна завжди встановлюється у «false», так як на початку не може бути жодної вибраної директорії.

Сформувавши детальний список із усією важливою інформацією можна підготувати віджет до відображення користувачу.

Із сформованим віджетом вже можна працювати. Аналізується час натискання та тип обраного елементу списку. Якщо це файл – незалежно від тривалості елемент обирається. Але якщо це директорія, то короткострокове натискання на неї відкриває її зміст, а довгострокове – виділяє, як файл.

Якщо виділити декілька файлів, а потім відкрити директорію – виділення скинеться. Так само і з виходом. Якщо вибрати декілька файлів та повернутись у батьківську директорію – виділення буде скинуто. Можна обрати чи увесь вміст вибраної директорії, чи нічого. Не можна обрати декілька файлів у декількох директоріях, що розташовані на одному рівні відносно кореня файлової системи.

Із інформації, що файли завантажуються та вивантажуються, можна зрозуміти, що користувач може обирати файли, а тому менеджери працюють справно. Але для їх роботи потрібен список файлів, який запитується у потоці. Робота методу «openServerDirectory()» зображена на рисунку 4.7. У потоці запускається метод, який запитує список файлів від серверу та оброблює його, переводячи в список, з яким зможе працювати програма. Саме такий список і повертається на форматування в список даних для віджету.

Рисунок 4.7 Блок-схема отримання змісту поточної відкритої директорії

Із результату передачі даних видно, що алгоритми передачі змісту серверу, завантаження/вивантаження та аналізу реалізовані правильно та успішно.

4.2 Захист розробленого програмного забезпечення

Для захисту проекту було використано ліцензію Apache License 2.0. Це ліцензія відкритого програмного забезпечення, яка є безкоштовною і єдина, яка надає право патентувати ліцензовану роботу. Ліцензія дозволяє третім сторонам комерційне використання, модифікацію, поширення, використання у патентах та приватне користування із обов'язковим зазначенням авторства.

Кожен файл проекту має починатись із ліцензійної вставки, яка визначає, хто є автором. Будь хто інший може змінювати мою програму, поширювати чи продавати, але він має обов'язково зазначити, що початковим автором є саме я. Якщо хтось внесе зміни до мого проекту, він має зазначити, які саме він зробив зміни.

Ліцензія не дозволяє використання проекту у торгових марках. ПЗ поширюється «як є». Ліцензія не надає гарантій щодо придатності до використання у певній меті. Повну відповідальність за визначення доцільності використання чи поширення несе користувач ПЗ. Ні в якому випадку та ні в якому правовому полі автор ПЗ не несе відповідальності за спричинені збитки ліцензією чи через неможливість використання ПЗ.

При поширенні роботи будь-хто може стягувати плату за гарантії, підтримку, поруку, компенсації та інші зобов'язання. Проте, при прийнятті таких зобов'язань, користувач діє лише від свого імені, а не від імені будь-кого із авторів і тільки тоді, коли користувач сам готовий компенсувати збитки та захищати будь-кого із авторів.

Видавцем ліцензії є Apache Software Foundation. Опубліковано ліцензію було у січні 2004 року. Ліцензію було схвалено в OSI (Open Sourse Initiative – некомерційна організація, створена для просування відкритого ПЗ) та вона є сумісною із GPLv3. Якщо команда розробників вирішить перейти на ліцензію GPLv3 – усі форки (похідні їх розробок) можуть залишитись під ліцензією Apache License 2.0.

Ліцензія потребує поширення ПЗ разом із початковим кодом. Якщо ж ПЗ не надає початкового коду, автор зобов'язаний надати його за запитом користувача програми.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Система складається із трьох програм: серверу FTPWinServer та клієнтів FTPAndClient і FTPWinClient.

Для вводу в експлуатацію системи потрібно встановити та запустити сервер у локальній мережі. Підійде будь-який пристрій, на який встановлено Python. Для запуску інтерфейсу серверу потрібно встановити бібліотеки:

```
pip install pyftpdlib
pip install sqlalchemy
pip install pyqt6
pip install humanize
```

Після встановлення бібліотек потрібно завантажити репозиторій. Для цього потрібно ввести команду:

```
git clone https://github.com/Nakama3942/FTES.git
```

Далі можна запускати програму: відкриваємо репозиторій, заходимо у директорію серверу та запускаємо «main.pyw». Для коректної роботи серверу треба в БД додати хоча б одного користувача. Інтерфейс програми поділено на дві частини. Зліва керування (запуск та зупинка) сервером. Справа керування БД. Під таблицею розташовані кнопки керування БД – сортування БД, додавання, редагування, видалення, огляд користувача. Після натискання кнопки додавання потрібно ввести ім’я користувача, його пароль, домашню директорію та дозволи. Скріншот запущеного серверу зображено на рисунку 5.1.

Для підключення до серверу й керування файловими системами серверу та клієнту потрібно використати клієнтський додаток. На мобільний пристрій програму потрібно попередньо встановити із зібраного «.apk» файлу. На комп’ютері достатньо відразу запустити «.jar» файл. Програма не потребує встановлення. Для підключення до серверу треба ввести IP адресу серверу, ім’я та пароль користувача.

Інтерфейс мобільного клієнту зображено на рисунку 5.2, а комп’ютерного клієнту – на рисунку 5.3.

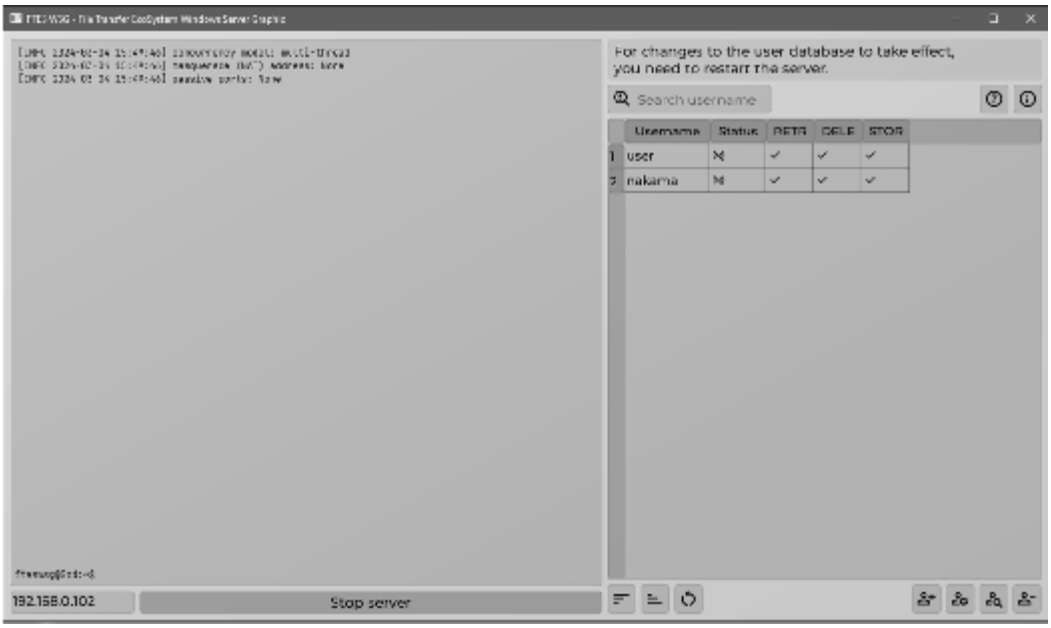

Рисунок 5.1 – Скріншот серверу

Необхідні системні характеристиками для запуску серверу:

Процесор: любий з тактовою частотою більше 1 ГГц

ОЗУ: 256 МБ

ПЗУ: 38 МБ

ОС: Будь-яка із архітектурою x64

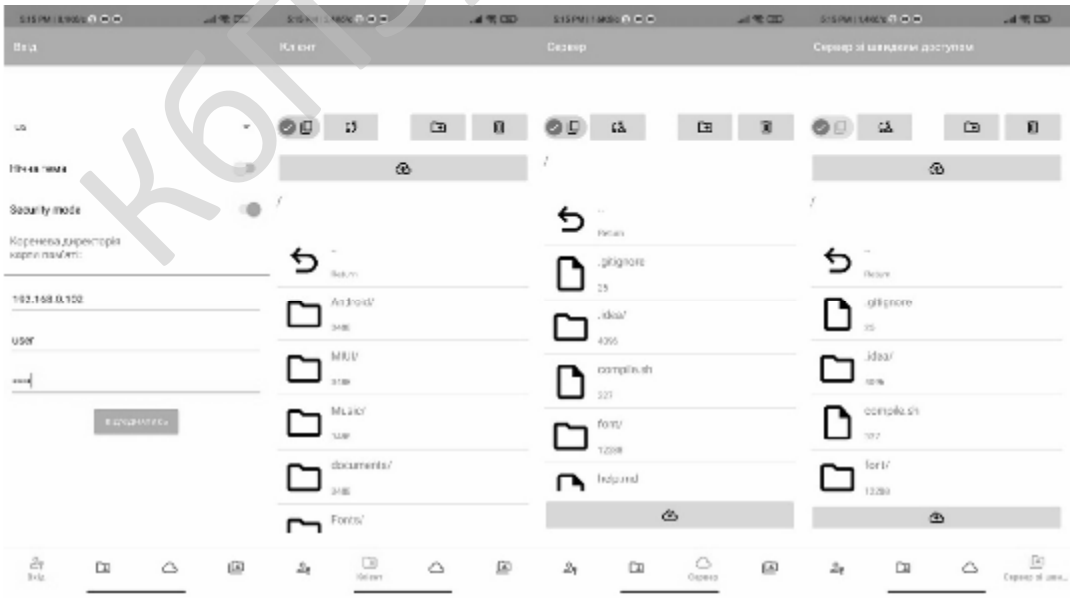

Рисунок 5.2 – Скріншоти клієнту телефону

Необхідні системні характеристиками для запуску мобільного клієнту:

Процесор: любий з тактовою частотою більше 1 ГГц

ОЗУ: 1 ГБ

ПЗУ: 8 МБ

ОС: Android 12 (Android API 31) та старше

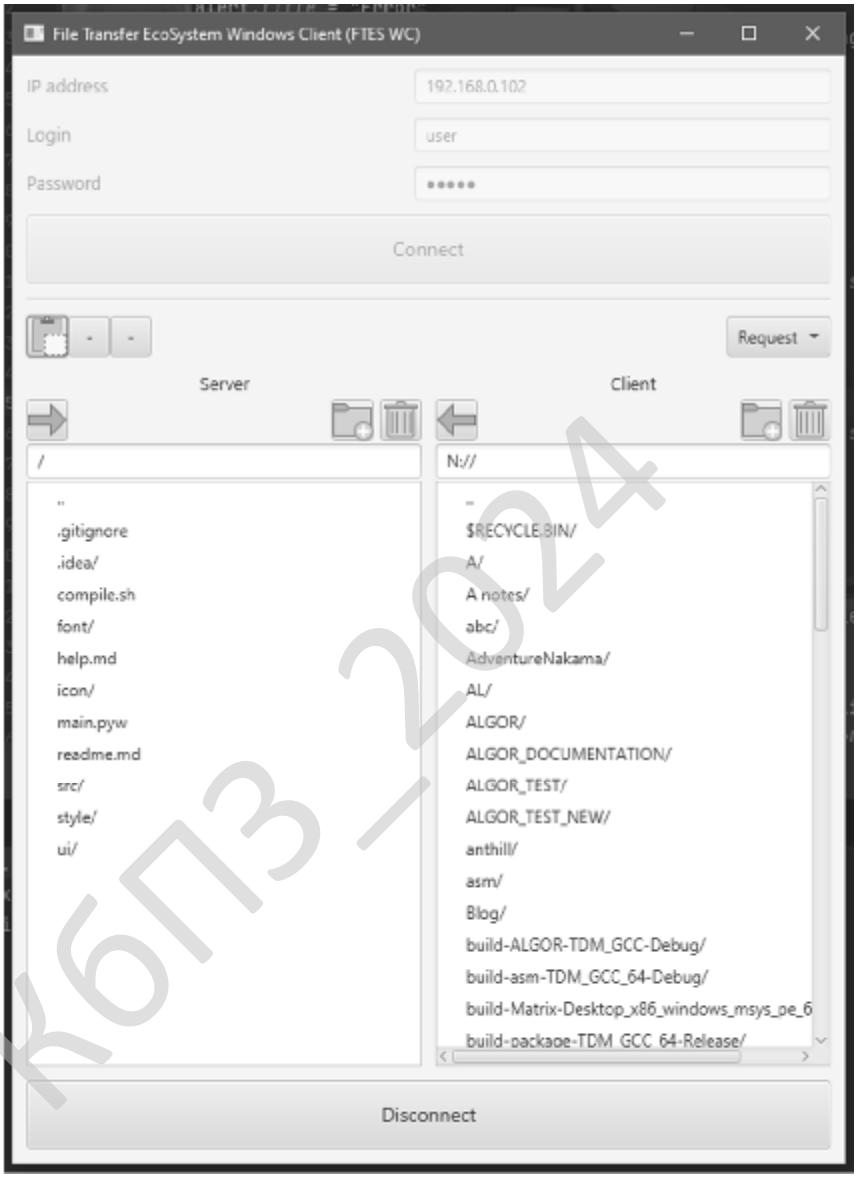

Рисунок 5.3 – Скріншот клієнту комп'ютеру

Необхідні системні характеристиками для запуску мобільного клієнту:

Процесор: любий з тактовою частотою більше 1 ГГц

ОЗУ: 256 МБ

ПЗУ: 12 МБ

ОС: Будь-яка із JVM x64

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для системи передачі файлів FTP протоколом у локальній мережі.

Для вирішення поставленої мети було проведено:

– дослідження існуючих систем передачі файлів FTP протоколом у локальній мережі.

– розробку алгоритмів системи передачі файлів FTP протоколом у локальній мережі.

– реалізацію програмного забезпечення системи передачі файлів FTP протоколом у локальній мережі.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішувати завдання передачі файлів між сервером та клієнтом у локальній мережі.

Розроблене програмне забезпечення представляє собою комплекс програм, що можна використовувати практично на будь-якому пристрої, підключеному до однієї локальної мережі.

Програмне забезпечення розроблялося у об’єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки програмного забезпечення системи передачі файлів FTP протоколом у локальній мережі використовувалися мови програмування Python та Kotlin. Дані мови програмування дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

Шифрування даних у розробленому програмному забезпеченні не використовувалось, так як передбачалось використання системи у локальній мережі, доступ до якої надано лише надійним пристроям.

Запропоноване програмне забезпечення складається з клієнтського та серверного додатків. У п’ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення серверного програмного забезпечення та інструкція для використання клієнтського програмного забезпечення.

Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Apache License 2.0.

Програмне забезпечення системи передачі файлів FTP протоколом у локальній мережі є важливим елементом в будь-якому сучасному діловому середовищі, а також для особистого використання з метою швидкої безпровідної передачі файлів між пристроями локальної мережі. Так як у наш час все більше створюється контенту, який треба поширювати між різними пристроями, кількість яких теж постійно зростає актуальність мережевих систем передачі файлів становиться все більше. Тож, розроблене у цій кваліфікаційній роботі програмне забезпечення має важливе значення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документ стандарту протоколу “RFC 959”: https://datatracker.ietf.org/doc/html/rfc959
2. Документація по мові Python: https://docs.python.org/uk/3/
3. Документація по фреймворку PyQt: https://doc.qt.io/qtforpython-6/
4. Документація по бібліотеці pyftpdlib: https://pyftpdlib.readthedocs.io/en/latest/
5. Документація по бібліотеці sqlalchemy: https://docs.sqlalchemy.org/en/20/
6. Документація по системним параметрам та функціям мови Python: https://docs.python.org/uk/3/library/sys.html
7. Документація по сокетам: https://docs.python.org/uk/3/howto/sockets.html
8. Документація по вебсокетам: https://websockets.readthedocs.io/en/stable/
9. Документація по мові Kotlin: https://kotlinlang.org/docs/home.html
10. Документація по Android: https://developer.android.com/guide
11. Документація по фреймворку Javafx: https://fxdocs.github.io/docs/html5/
12. Документація по бібліотеці org.apache.commons.net: https://cwiki.apache.org/confluence/display/commons/FrontPage
13. Документація про SAF: https://developer.android.com/guide/topics/providers/document-provider
14. Стаття «Що таке FTP і для чого він потрібен» на сайті hostiq: https://hostiq.ua/wiki/ukr/ftp/
15. Стаття «ТОП-15 FTP-клієнтів» на сайті plerdy: https://www.plerdy.com/ua/blog/top-15-ftp-clients/
16. Стаття «Що таке stdin, stdout і stderr у Linux?» на сайті techukraine: https://techukraine.net/що-таке-stdin-stdout-i-stderr-y-linux/
17. Стаття «8 регулярних виразів, які ви повинні знати» на сайті tutsplus: https://code.tutsplus.com/uk/8-regular-expressions-you-should-know--net-6149t
18. Стаття «47 важливих порад для UI та UX дизайнерів» на сайті uxpub:

https://ux.pub/editorial/47-vazhlivikh-porad-dlia-ui-ta-ux-dizainieriv-4h0j

19. Стаття «The ubiquitous digital file: A review of file management research» на сайті researchgate: https://www.researchgate.net/publication/354723356_The_ubiquitous_digital_file_A_review_of_file_management_research

20. Стаття «File system» на сайті techtarget: https://www.techtarget.com/searchstorage/definition/file-system

21. Стаття «A File Transfer Protocol (FTP)» на сайті sciencedirect: https://www.sciencedirect.com/science/article/abs/pii/0376507578900090

22. Стаття «What Is FTP» на сайті spiceworks: https://www.spiceworks.com/tech/networking/articles/what-is-ftp/

23. Стаття «A Complete Guide to Socket Programming in Python» на сайті datacamp: https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python

24. Стаття у Вікіпедії «FTP»: https://uk.wikipedia.org/wiki/FTP

25. Стаття у Вікіпедії «Регулярний вираз»: https://uk.wikipedia.org/wiki/Регулярний_вираз

26. Стаття у Вікіпедії «Файлова система»: https://uk.wikipedia.org/wiki/Файлова_система

27. Стаття у Вікіпедії «Сокет»: https://uk.wikipedia.org/wiki/Сокет_(програмний_інтерфейс)

28. Стаття у Вікіпедії «WebSocket»: https://uk.wikipedia.org/wiki/WebSocket

29. Стаття у Вікіпедії «FileZilla»: https://uk.wikipedia.org/wiki/FileZilla

30. Документація програми «FileZilla»: https://wiki.filezilla-project.org/Documentation

31. Стаття у Вікіпедії «WinSCP»: https://uk.wikipedia.org/wiki/WinSCP

32. Документація програми «WinSCP»: https://winscp.net/eng/docs/lang:uk

33. Деніел Б. Linux Pocket Guide: Essential Commands / Деніел Б. – Себастопол : O'Reilly Media, 2016 – 274с.

34. Томас Х. File Systems: Structures and Algorithms / Томас Х. – Нью Джерсі : Prentice Hall, 1988 – 280с.

35. Рене Ш. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions / Рене Ш. – Берлінгтон : Digital Press, 2004 – 784с.

36. Димитрос С. Architecture of Network Systems (The Morgan Kaufmann Series in Computer Architecture and Design) / Димитрос С., Тільман В. – Берлінгтон : Morgan Kaufmann, 2011 – 344с.

37. Олексій В. Програмування в PYTHON. Теорія і практика / Олексій В. – Київ : Ліра-К, 2023 – 462с.

38. Марк Л. Python. Довідник програміста / Марк Л. – Київ : Діалектика, 2023 – 294с.

39. Роберт М. Чистий код: створення, аналіз, рефакторинг / Роберт М. – Харків : Фабула, 2019 – 416с.

40. Роберт М. Чиста архітектура: мистецтво розробки програмного забезпечення / Роберт М. – Харків : Фабула, 2019 – 416с.

41. Кліффорд Ш. Алгоритми: Побудова та аналіз / Томас К., Чарльз Л., Рональд Р., Кліффорд Ш. – Київ : Діалектика, 2022 – 1328с.

42. Алан К. About Face: The Essentials of Interaction Design / Алан К., Роберт Р., Давід К., Крістофре Н. – Індіанаполіс : John Wiley & Sons, 2014 – 720с.

43. Стів К. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability / Стів К. – Сан-Франциско : New Riders, 2014 – 216с.

44. Грег Н. Android Design Patterns: Interaction Design Solutions for Developers / Грег Н. – Індіанаполіс : John Wiley & Sons, 2013 – 456с.

45. П’єр-Олівер Л. Programming Android with Kotlin: Achieving Structured Concurrency with Coroutines / П’єр-Олівер Л., Аманда Х.-Д., Блейк М., Майк Д. – Себастопол : O’Reilly Media, 2022 – 352с.

46. Джош С. Kotlin Programming: The Big Nerd Ranch Guide / Джош С., Девід Г. – Індіанаполіс : Big Nerd Ranch, 2018 – 480с.

47. Білл Ф. Android Programming: The Big Nerd Ranch Guide / Білл Ф., Кріс

С., Браян Х., Крістін М. – Індіанаполіс : Big Nerd Ranch, 2015 – 618с.

48. Марцін М. Android Development with Kotlin: Enhance your skills for Android development using Kotlin / Марцін М., Ігор В. – Бірмінгем : Packt Publishing, 2017 – 440с.

49. Нібедіт Д. ross-Platform Development with Qt 6 and Modern C++: Design and build applications with modern graphical user interfaces without worrying about platform dependency / Нібедіт Д. – Бірмінгем : Packt Publishing, 2021 – 442с.

50. Пітер С. Frontend Development with JavaFX and Kotlin: Build State-of-the-Art Kotlin GUI Applications / Пітер С. – Нью Йорк : Apress Media, 2023 – 152с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №____ від __.__.2024 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– систему забезпечення безпеки файлів та ресурсів серверу;
– цілісність файлів у процесі завантаження/вивантаження;
– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень щодо підтримки різних операційних систем.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows, Linux, Android і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows, Linux, Android.

5.8.1 Обладнання

Комп’ютер для серверу: процесор любий із тактовою частотою 1 ГГц; ОЗУ 256 МБ; ПЗУ 38 МБ; ОС будь-яка із архітектурою x64.

Комп’ютер для клієнту: процесор любий із тактовою частотою 1 ГГц; ОЗУ 1 ГБ; ПЗУ 8 МБ; ОС Android 12 (Android API 31) та старше чи ОЗУ 256 МБ; ПЗУ 12 МБ; ОС будь-яка із JVM x64.

5.8.2 Мова програмування

Програму розроблено на мовах програмування Python та Kotlin.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

- Структурна схема системи.
- Функціональна схема системи.
- Діаграма процесів.
- Блок-схема алгоритму роботи програми.
- Пояснювальна записка.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок-схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Відлагодження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання кваліфікаційної бакалаврської роботи на попередній захист __.__.2024 р.

9.2 Подання кваліфікаційної бакалаврської роботи на захист __.__.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Є.В. Мелешко

Програмне забезпечення кросплатформної клієнт-серверної системи для передачі файлів за протоколом FTP

Лістинг програми

Код документу 12

Носій: Хмарне сховище

Загальна кількість аркушів: 33

Літера: РП

Кропивницький – 2024 року

// ClientLogic.kt – Кросплатформовий модуль: ядро FTP клієнту

```
package com.fac.ftpandclient

import org.apache.commons.net.ftp.FTP
import org.apache.commons.net.ftp.FTPClient
import java.io.File
import java.io.FileInputStream
import java.io.FileOutputStream
import java.io.InputStream
import java.io.OutputStream

class LoginException(message: String) : Exception(message)

class ClientLogic(
      private val username: String = "user",
      private val password: String = "12345",
      private val server: String = "192.168.0.102"
) {
      private val port: Int = 21
      private val ftpClient = FTPClient()

      fun connect() {
            ftpClient.controlEncoding = "UTF-8"
            ftpClient.connect(server, port)
            if (!ftpClient.login(username, password)) {
                  disconnect()
                  throw LoginException("USER '$username' failed login.")
            }
            ftpClient.setFileType(FTP.BINARY_FILE_TYPE)
      }

      fun disconnect() {
            ftpClient.logout()
            ftpClient.disconnect()
      }

      private fun getServerSystem(): List<String> {
            val directoryContent = mutableListOf("..")
            val subFiles = ftpClient.listFiles(
                  ftpClient.printWorkingDirectory()
            )
            if (subFiles != null) {
                  for (subFile in subFiles) {
                        if (subFile.isDirectory) {
                              directoryContent.add("${subFile.name}/")
                        } else {
                              directoryContent.add(subFile.name)
                        }
                  }
            }
            return directoryContent
      }

      fun openServerDirectory(serverDirectoryName: String): List<String>? {
            if (ftpClient.changeWorkingDirectory(serverDirectoryName)) {
                  return getServerSystem()
            }
            return null
      }

      fun openParentServerDirectory(): List<String>? {
            if (ftpClient.changeToParentDirectory()) {
                  return getServerSystem()
            }
            return null
      }
```

```
fun updateServerDirectory(): List<String> {
      return getServerSystem()
}

private fun getClientSystem(clientDirectoryName: String): List<String>
{
      val directory = File(clientDirectoryName)
      val directoryContent = mutableListOf("..")
      val subFiles = directory.listFiles()
      if (subFiles != null) {
            for (subFile in subFiles) {
                  if (subFile.isDirectory) {
                        directoryContent.add("${subFile.name}/")
                  } else {
                        directoryContent.add(subFile.name)
                  }
            }
      }
      return directoryContent
}

fun openClientDirectory(clientDirectoryName: String): List<String>? {
      if (File(clientDirectoryName).isDirectory) {
            return getClientSystem(clientDirectoryName)
      }
      return null
}

fun openParentClientDirectory(
      clientDirectoryName: String
): List<String>? {
      if (File(clientDirectoryName).isDirectory) {
            return getClientSystem(clientDirectoryName)
      }
      return null
}

fun updateClientDirectory(clientDirectoryName: String): List<String> {
      return getClientSystem(clientDirectoryName)
}

fun getServerFileSize(filePath: String): Long {
      return ftpClient.mlistFile(filePath).size
}

fun getClientFileSize(filePath: String): Long {
      return File(filePath).length()
}

fun downloadAll(
      clientPath: String,
      serverFiles: List<String>,
      clear: Boolean
) {
      for (serverFile in serverFiles) {
            if (serverFile == "..") {
                  continue
            }
            download(clientPath, serverFile)
            if (clear) {
                  val clearingPath = mutableListOf<String>()
                  clearingPath.add(
                        ftpClient.printWorkingDirectory()
                        + "/" + serverFile
                  )
                  removeServerPath(clearingPath)
            }
      }
}
```

```
private fun download(clientPath: String, serverFile: String) {
      val sourcePath = if (ftpClient.printWorkingDirectory() == "/") {
            ftpClient.printWorkingDirectory() + serverFile
      }
      else {
            ftpClient.printWorkingDirectory() + "/" + serverFile
      }
      val targetPath = File(clientPath, serverFile)

      if (ftpClient.mlistFile(sourcePath).isFile) {
            // This element a file
            val localFileOutputStream = FileOutputStream(targetPath.path)
            ftpClient.retrieveFile(sourcePath, localFileOutputStream)
            localFileOutputStream.close()
      } else {
            // This element a directory
            createClientDirectory(targetPath.path)
            ftpClient.changeWorkingDirectory(sourcePath)
            for (file in ftpClient.listFiles()) {
                  download(targetPath.path, file.name)
            }
            openParentServerDirectory()
      }
}

fun uploadAll(
      clientPath: String,
      clientFiles: List<String>,
      clear: Boolean
) {
      for (clientFile in clientFiles) {
            if (clientFile == "..") {
                  continue
            }
            upload(clientPath, clientFile)
            if (clear) {
                  val clearingPath = mutableListOf<String>()
                  clearingPath.add(File(clientPath, clientFile).path)
                  removeClientPath(clearingPath)
            }
      }
}

private fun upload(clientPath: String, clientFile: String) {
      val sourcePath = File(clientPath, clientFile)
      val targetPath = if (ftpClient.printWorkingDirectory() == "/") {
            ftpClient.printWorkingDirectory() + clientFile
      }
      else {
            ftpClient.printWorkingDirectory() + "/" + clientFile
      }

      if (sourcePath.isFile) {
            // This element a file
            val localFileInputStream = FileInputStream(sourcePath.path)
            ftpClient.storeFile(targetPath, localFileInputStream)
            localFileInputStream.close()
      } else {
            // This element a directory
            createServerDirectory(targetPath)
            ftpClient.changeWorkingDirectory(targetPath)
            for (file in sourcePath.listFiles()!!) {
                  upload(sourcePath.path, file.name)
            }
            openParentServerDirectory()
      }
}
```

```
        fun isServerFile(fileName: String): Boolean {
              if (ftpClient.mlistFile(fileName).isFile) {
                    return true
              }
              return false
        }

        fun currentServerDir(): String {
              return ftpClient.printWorkingDirectory()
        }

        fun downloadStream(fileName: String, fileContentStream: OutputStream) {
              // Download the contents of the file from the server
              ftpClient.retrieveFile(fileName, fileContentStream)
        }

        fun uploadStream(fileName: String, fileContentStream: InputStream) {
              // Upload the contents of the file to the server
              ftpClient.storeFile(fileName, fileContentStream)
        }

        fun createServerDirectory(newServerDirectory: String) {
              ftpClient.makeDirectory(newServerDirectory)
        }

        fun createClientDirectory(newClientDirectory: String) {
              File(newClientDirectory).mkdirs()
        }

        fun removeServerPath(serverFilePaths: List<String>) {
              for (serverFilePath in serverFilePaths) {
                    if (ftpClient.mlistFile(serverFilePath).isDirectory) {
                          // This element a directory
                          ftpClient.changeWorkingDirectory(serverFilePath)
                          for (file in ftpClient.listFiles()) {
                                val files = mutableListOf<String>()
                                files.add(file.name)
                                removeServerPath(files)
                          }
                          openParentServerDirectory()
                          ftpClient.removeDirectory(serverFilePath)
                    } else {
                          // This element a file
                          ftpClient.deleteFile(
                                ftpClient.mlistFile(serverFilePath).name
                          )
                    }
              }
        }

        fun removeClientPath(clientFilePaths: List<String>) {
              for (clientFilePath in clientFilePaths) {
                    File(clientFilePath).deleteRecursively()
              }
        }
}
```

// MainActivity.kt – Модуль ініціалізації клієнту (Android)

```
package com.fac.ftpandclient

import android.os.Bundle
import androidx.activity.result.ActivityResultLauncher
import com.google.android.material.bottomnavigation.BottomNavigationView
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.findNavController
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.setupActionBarWithNavController
import androidx.navigation.ui.setupWithNavController
import com.fac.ftpandclient.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    private lateinit var connectionModel: ConnectionModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val navView: BottomNavigationView = binding.navView

        // Accessing the ConnectionViewModel
        connectionModel = ViewModelProvider(this)[ConnectionModel::class.java]

        val navController =
findNavController(R.id.nav_host_fragment_activity_main)
        // Passing each menu ID as a set of Ids because each
        // menu should be considered as top level destinations.
        val appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.navigation_login,
                R.id.navigation_client,
                R.id.navigation_server,
                R.id.navigation_server_with_saf
            )
        )

        // Subscribe to connection state changes
        connectionModel.isConnected().observe(this) { isConnected ->
            // Setting the click ability of tabs depending on isConnected
            navView.menu.findItem(
                R.id.navigation_client
            ).isVisible = isConnected
            navView.menu.findItem(
                R.id.navigation_server
            ).isVisible = isConnected
            navView.menu.findItem(
                R.id.navigation_server_with_saf
            ).isVisible = isConnected
        }

        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)
    }
}
```

// LoginFragment.kt – Один із чотирьох модулів інтерфейсу: підключення до серверу (Android)

```
package com.fac.ftpandclient.ui

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.AdapterView
import android.widget.ArrayAdapter
import android.widget.Button
import android.widget.EditText
import android.widget.Spinner
import android.widget.Toast
import androidx.appcompat.app.AppCompatDelegate
import androidx.appcompat.widget.SwitchCompat
import androidx.core.content.ContextCompat
import androidx.core.view.isVisible
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import com.fac.ftpandclient.ClientLogic
import com.fac.ftpandclient.ConnectionModel
import com.fac.ftpandclient.FileSystemDivulger
import com.fac.ftpandclient.R
import com.fac.ftpandclient.databinding.FragmentLoginBinding
import com.fac.ftpandclient.ImportantData
import java.io.File

class LoginFragment : Fragment() {

    private var _binding: FragmentLoginBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    private lateinit var serv: ClientLogic
    private lateinit var connectionModel: ConnectionModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentLoginBinding.inflate(inflater, container, false)
        val root: View = binding.root

        // Accessing the ConnectionViewModel
        connectionModel = ViewModelProvider(
            requireActivity()
        )[ConnectionModel::class.java]

        // Links to interface buttons
        val languageBox = root.findViewById<Spinner>(R.id.languageBox)
        val themeSwitch = root.findViewById<SwitchCompat>(R.id.themeSwitch)
        val rootDir = root.findViewById<Spinner>(R.id.rootDirBox)
        val serverIp = root.findViewById<EditText>(R.id.serverIpField)
        val login = root.findViewById<EditText>(R.id.loginField)
        val password = root.findViewById<EditText>(R.id.passwordField)
        val connecting = root.findViewById<Button>(R.id.connectButt)

        // Creating and installing an adapter for selecting a language
        val languages = arrayOf("En", "Ua")
        val languageAdapter = ArrayAdapter(
            requireContext(),
            android.R.layout.simple_spinner_item,
            languages
        )
```

```
languageBox.adapter = languageAdapter

themeSwitch.setOnCheckedChangeListener { _, isChecked ->
    // Changing the App Theme
    if (isChecked) {
        // Dark theme selected
        AppCompatDelegate.setDefaultNightMode(
            AppCompatDelegate.MODE_NIGHT_YES
        )
    } else {
        // Light theme selected
        AppCompatDelegate.setDefaultNightMode(
            AppCompatDelegate.MODE_NIGHT_NO
        )
    }
}

// Creating a list of available roots
val storageDirectories = ContextCompat.getExternalFilesDirs(
    requireContext(),
    null
)
val rootDirs = mutableListOf<String>()
for (dir in storageDirectories) {
    rootDirs.add(dir.absolutePath.substring(
        0,
        dir.absolutePath.indexOf("/Android/data")
    ))
}

// Creating and installing an adapter for selecting a root
val storageAdapter = ArrayAdapter(
    requireContext(),
    android.R.layout.simple_spinner_item,
    rootDirs
)
rootDir.adapter = storageAdapter

// Installing a root selection handler
rootDir.onItemSelectedListener =
    object : AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>?,
            view: View?,
            position: Int,
            id: Long
        ) {
            // Save selected item
            val selectedItem = rootDirs[position]
            ImportantData.clientRoot = selectedItem
    }
    override fun onNothingSelected(parent: AdapterView<*>?) {
        // Nothing selected
        val myToast = Toast.makeText(
            activity,
            "Nothing selected",
            Toast.LENGTH_LONG
        )
        myToast.show()
    }
}

if (!connectionModel.isConnected().value!!) {
    connecting.text = getString(R.string.connect)
}
else {
    connecting.text = getString(R.string.disconnect)
}
```

```
            connecting.setOnClickListener {
                // Connecting/disconnecting a connection
                if (!connectionModel.isConnected().value!!) {
                    Thread {
                        try {
                            ImportantData.server = ClientLogic(
                                login.text.toString(),
                                password.text.toString(),
                                serverIp.text.toString()
                            )
                            serv = ImportantData.server!!
                            serv.connect()
                            FileSystemDivulger.encodeFileSystem(
                                FileSystemDivulger.getAllFilesAndDirectories(
                                    File(ImportantData.clientRoot)
                                )
                            )
                            activity?.runOnUiThread {
                                connecting.text = getString(R.string.disconnect)
                                connectionModel.setConnected(true)
                                ImportantData.clientPath = "/"
                                ImportantData.serverRoot = "/"
                                ImportantData.serverPath = ""
                                ImportantData.rootOfHomeDirectoryIsVisible = false
                                rootDir.isVisible = false
                            }
                        } catch (e: Exception) {
                            val myToast = Toast.makeText(
                                activity,
                                e.message.toString(),
                                Toast.LENGTH_LONG
                            )
                            myToast.show()
                        }
                    }.start()
                }
                else {
                    Thread {
                        serv.disconnect()
                    }.start()
                    connecting.text = getString(R.string.connect)
                    connectionModel.setConnected(false)
                    connectionModel.setClientUpdateIsNeeded(true)
                    connectionModel.setServerUpdateIsNeeded(true)
                    ImportantData.clientPath = ""
                    ImportantData.serverPath = ""
                    ImportantData.rootOfHomeDirectoryIsVisible = true
                    rootDir.isVisible = true
                }
            }

            return root
        }

        override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
        }
    }
```

// FileListAdapter.kt – Модуль файлової системи (Android)

```
package com.fac.ftpandclient

import android.graphics.Color
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView

class FileListAdapter(
    private var fileList: List<FileItem>
) : RecyclerView.Adapter<FileListAdapter.FileViewHolder>() {

    // Listener interface
    interface OnItemClickListener {
        fun onItemClick(position: Int)
    }

    // Listener variable
    private var listener: OnItemClickListener? = null

    class FileViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val fileImage: ImageView = itemView.findViewById(R.id.fileImage)
        val fileName: TextView = itemView.findViewById(R.id.fileName)
        val fileInfo: TextView = itemView.findViewById(R.id.fileInfo)
    }

    fun updateFileList(newFileList: List<FileItem>) {
        fileList = newFileList
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int
    ): FileViewHolder {
        val view =
            LayoutInflater
            .from(parent.context)
            .inflate(R.layout.fragment_file, parent, false)
        return FileViewHolder(view)
    }

    override fun onBindViewHolder(holder: FileViewHolder, position: Int) {
        val fileItem = fileList[position]

        holder.fileImage.setImageURI(fileItem.image)
        holder.fileName.text = fileItem.name
        holder.fileInfo.text = fileItem.info

        // Handling a short click on a list item
        holder.itemView.setOnClickListener {
            if (!fileItem.isDirectory) {
                toggleSelection(position)
            }
            else {
                listener?.onItemClick(position)
            }
        }

        // Handling a long press on a list item
        holder.itemView.setOnLongClickListener {
            // If the selected element is a directory - select it
            if (fileItem.isDirectory) {
                toggleSelection(position)
                return@setOnLongClickListener true
```

```
            }
            false
        }

        val backgroundResId = if (fileItem.isSelected) {
            R.color.green_500_trans // Resource for the selected element
        } else {
            Color.TRANSPARENT // Resource for an unselected element
        }
        holder.itemView.setBackgroundResource(backgroundResId)
    }

    override fun getItemCount(): Int = fileList.size

    fun setOnItemClickListener(listener: OnItemClickListener) {
        this.listener = listener
    }

    private fun toggleSelection(position: Int) {
        // Selecting an element
        val fileItem = fileList[position]
        fileItem.isSelected = !fileItem.isSelected
        notifyItemChanged(position)
    }

    fun getSelectedFiles(): List<FileItem> {
        // Method for getting a list of selected elements
        return fileList.filter { it.isSelected }
    }
}
```

// ConnectionModel.kt – Модуль моделі з'єднання (Android)

```
package com.fac.ftpandclient

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class ConnectionModel : ViewModel() {
    private val isConnectedLiveData = MutableLiveData<Boolean>()
    private val clientUpdateIsNeededLiveData = MutableLiveData<Boolean>()
    private val serverUpdateIsNeededLiveData = MutableLiveData<Boolean>()

    init {
        isConnectedLiveData.value = false
        clientUpdateIsNeededLiveData.value = false
        serverUpdateIsNeededLiveData.value = false
    }

    fun setConnected(isConnected: Boolean) {
        isConnectedLiveData.value = isConnected
    }

    fun isConnected(): LiveData<Boolean> {
        return isConnectedLiveData
    }

    fun setClientUpdateIsNeeded(clientUpdateIsNeeded: Boolean) {
        clientUpdateIsNeededLiveData.value = clientUpdateIsNeeded
    }

    fun clientUpdateIsNeeded(): LiveData<Boolean> {
        return clientUpdateIsNeededLiveData
    }

    fun setServerUpdateIsNeeded(serverUpdateIsNeeded: Boolean) {
        serverUpdateIsNeededLiveData.value = serverUpdateIsNeeded
    }

    fun serverUpdateIsNeeded(): LiveData<Boolean> {
        return serverUpdateIsNeededLiveData
    }
}
```

// FTPWinClientApplication.kt – Модуль ініціалізації клієнту (ПК)

```
package com.fwc.ftpwinclient

import javafx.application.Application
import javafx.fxml.FXMLLoader
import javafx.scene.Scene
import javafx.stage.Stage

class FTPWinClientApplication : Application() {
    override fun start(stage: Stage) {
        val fxmlLoader = FXMLLoader(
            FTPWinClientApplication::class.java.getResource("main-window.fxml")
        )
        val scene = Scene(fxmlLoader.load(), 600.0, 800.0)
        stage.title = "File Transfer EcoSystem Windows Client (FTES WC)"
        stage.scene = scene
        stage.minWidth = 600.0
        stage.minHeight = 800.0
        stage.show()
    }
}

fun main() {
    Application.launch(FTPWinClientApplication::class.java)
}
```

// MainWindowController.kt – Модуль інтерфейсу клієнту (ПК)

```
package com.fwc.ftpwinclient

import javafx.fxml.FXML
import javafx.scene.control.*
import javafx.scene.layout.VBox
import java.io.File

class MainWindowController {
    @FXML
    private lateinit var connectGroup: VBox
    @FXML
    private lateinit var serverGroup: VBox
    @FXML
    private lateinit var ipField: TextField
    @FXML
    private lateinit var loginField: TextField
    @FXML
    private lateinit var passField: PasswordField
    @FXML
    private lateinit var serverDirectory: TextField
    @FXML
    private lateinit var clientDirectory: TextField
    @FXML
    private lateinit var serverFileSystems: TreeView<String>
    @FXML
    private lateinit var clientFileSystems: TreeView<String>
    @FXML
    private lateinit var connectButt: Button
    @FXML
    private lateinit var disconnectButt: Button
    @FXML
    private lateinit var downloadButt: Button
    @FXML
    private lateinit var uploadButt: Button
    @FXML
    private lateinit var createServerDirButt: Button
    @FXML
    private lateinit var createClientDirButt: Button
    @FXML
    private lateinit var removeServerDirButt: Button
    @FXML
    private lateinit var removeClientDirButt: Button
    @FXML
    private lateinit var dublicateToggle: ToggleButton

    private lateinit var serv: ClientLogic

    @FXML
    private fun onConnectClicked()
    {
        try {
            serv = ClientLogic(loginField.text, passField.text, ipField.text)
            serv.connect()
            connectGroup.isDisable = true
            serverGroup.isDisable = false
        } catch (e: Exception) {
            val error = e.toString().split(": ")
            val alert = Alert(Alert.AlertType.ERROR)
            alert.title = "Error"
            alert.headerText = error.subList(1, error.size).joinToString(": ")
            alert.contentText = error[0]
            alert.buttonTypes.setAll(ButtonType.OK)
            alert.showAndWait()
            return
        }

        serverDirectory.text = "/"
```

```
serverFileSystems.root = createTreeItems(
    serverDirectory.text,
    serv.openServerDirectory(serverDirectory.text)!!
)
val serverSelectionModel = serverFileSystems.selectionModel
serverSelectionModel.selectionMode = SelectionMode.MULTIPLE

clientDirectory.text = "N://"
clientFileSystems.root = createTreeItems(
    clientDirectory.text,
    serv.openClientDirectory(clientDirectory.text)!!
)
val clientSelectionModel = clientFileSystems.selectionModel
clientSelectionModel.selectionMode = SelectionMode.MULTIPLE

serverFileSystems.setOnMouseClicked { event ->
    if (event.clickCount == 2) {
        val selectedItem = serverFileSystems.selectionModel.selectedItem
        if (selectedItem != null) {
            if (selectedItem.value == "..") {
                serverDirectory.text =
                    serverDirectory
                    .text
                    .split("/")
                    .subList(
                        0,
                        serverDirectory.text.split("/").size - 2
                    ).joinToString("/")
                    + "/"
                serverFileSystems.root = createTreeItems(
                    serverDirectory.text,
                    serv.openParentServerDirectory()!!
                )
            } else {
                try {
                    serverFileSystems.root = createTreeItems(
                        serverDirectory.text,
                        serv.openServerDirectory(selectedItem.value)!!
                    )
                    serverDirectory.text += selectedItem.value
                } catch (e: Exception) {
                    // Analogue of the client "if (dir.isDirectory)"
                }
            }
        }
    }
}
clientFileSystems.setOnMouseClicked { event ->
    if (event.clickCount == 2) {
        val selectedItem = clientFileSystems.selectionModel.selectedItem
        if (selectedItem != null) {
            if (selectedItem.value == "..") {
                clientDirectory.text =
                    "N://" + clientDirectory
                    .text
                    .split("/")
                    .subList(
                        2,
                        clientDirectory.text.split("/").size - 2
                    ).joinToString("/")
                if (clientDirectory.text.last().toString() != "/") {
                    clientDirectory.text += "/"
                }
                clientFileSystems.root = createTreeItems(
                    clientDirectory.text,
                    serv.openParentClientDirectory(
                        clientDirectory.text
                    )!!
                )
```

```
                } else {
                    val dir = File(
                        clientDirectory.text + selectedItem.value
                    )
                    if (dir.isDirectory) {
                        clientFileSystems.root = createTreeItems(
                            clientDirectory.text,
                            serv.openClientDirectory(dir.path)!!
                        )
                        clientDirectory.text += selectedItem.value
                    }
                }
            }
        }
    }
}

@FXML
private fun onDisconnectClicked()
{
    serv.disconnect()
    connectGroup.isDisable = false
    serverGroup.isDisable = true

    serverDirectory.text = ""
    serverFileSystems.root = null
    clientDirectory.text = ""
    clientFileSystems.root = null
}

@FXML
private fun onDownloadClicked()
{
    val selectedItems = serverFileSystems.selectionModel.selectedItems
    serv.downloadAll(
        clientDirectory.text,
        createList(selectedItems),
        !dublicateToggle.isSelected
    )
    updateSystems()
}

@FXML
private fun onUploadClicked()
{
    val selectedItems = clientFileSystems.selectionModel.selectedItems
    serv.uploadAll(
        clientDirectory.text,
        createList(selectedItems),
        !dublicateToggle.isSelected
    )
    updateSystems()
}

@FXML
private fun onCreateServerDirClicked()
{
    val dialog = TextInputDialog()
    dialog.title = "Creating a new directory"
    dialog.headerText = "Enter the name of new directory"
    dialog.contentText = "Name of directory: "

    val dialogResult = dialog.showAndWait()
    dialogResult.ifPresent { directoryName ->
        if (directoryName.isNotBlank()) {
            serv.createServerDirectory(directoryName)
        } else {
            val alert = Alert(Alert.AlertType.ERROR)
            alert.title = "Error"
```

```
            alert.headerText = "The directory name cannot be empty"
            alert.contentText = "Enter the name of new directory"
            alert.buttonTypes.setAll(ButtonType.OK)
            alert.showAndWait()
            onCreateServerDirClicked()
        }
    }
    updateSystems()
}

@FXML
private fun onCreateClientDirClicked()
{
    val dialog = TextInputDialog()
    dialog.title = "Creating a new directory"
    dialog.headerText = "Enter the name of new directory"
    dialog.contentText = "Name of directory: "

    val dialogResult = dialog.showAndWait()
    dialogResult.ifPresent { directoryName ->
        if (directoryName.isNotBlank()) {
            serv.createClientDirectory(clientDirectory.text + directoryName)
        } else {
            val alert = Alert(Alert.AlertType.ERROR)
            alert.title = "Error"
            alert.headerText = "The directory name cannot be empty"
            alert.contentText = "Enter the name of new directory"
            alert.buttonTypes.setAll(ButtonType.OK)
            alert.showAndWait()
            onCreateClientDirClicked()
        }
    }
    updateSystems()
}

@FXML
private fun onRemoveServerDirClicked()
{
    val selectedItems = serverFileSystems.selectionModel.selectedItems
    for (selectedItem in selectedItems) {
        serv.removeServerPath(serverDirectory.text + selectedItem.value)
    }
    updateSystems()
}

@FXML
private fun onRemoveClientDirClicked()
{
    val selectedItems = clientFileSystems.selectionModel.selectedItems
    for (selectedItem in selectedItems) {
        serv.removeClientPath(clientDirectory.text + selectedItem.value)
    }
    updateSystems()
}

private fun updateSystems() {
    serverFileSystems.root = createTreeItems(
        serverDirectory.text,
        serv.updateServerDirectory()
    )
    clientFileSystems.root = createTreeItems(
        clientDirectory.text,
        serv.updateClientDirectory(clientDirectory.text)
    )
}

private fun createTreeItems(
    rootName: String,
    list: List<String>
```

```
    ): TreeItem<String> {
        val root = TreeItem(rootName)
        for (content in list) {
            root.children.add(TreeItem(content))
        }
        return root
    }

    private fun createList(list: List<TreeItem<String>>): List<String> {
        return list.map { it.value }
    }
}
```

// Server.py – Ядро FTP серверу

```
import logging
import threading

from pyftpdlib.authorizers import DummyAuthorizer
from pyftpdlib.handlers import FTPHandler
from pyftpdlib.servers import ThreadedFTPServer

from src.GlobalStates import GlobalStates

class Server:
      def __init__(self, ip, stdout, stderr):
            self.__ip = ip
            self.__stdout = stdout
            self.__stderr = stderr

            self.__authorizer = DummyAuthorizer()
            self.__handler = FTPHandler
            self.__server = None

            self.__server_thread = None

      def set_log(self):
            # Налаштування логування
            logger = logging.getLogger()
            logger.setLevel(logging.INFO)

            formatter = logging.Formatter(
                  '[%(levelname)s %(asctime)s] %(message)s',
                  datefmt='%Y-%m-%d %H:%M:%S'
            )

            # Створення обробника для виводу в консоль
            console_handler = logging.StreamHandler()
            console_handler.setFormatter(formatter)
            console_handler.encoding = 'utf-8'
            console_handler.setStream(self.__stdout)
            logger.addHandler(console_handler)

            # Створення обробника для запису до файлу
            file_handler = logging.FileHandler(
                  f"{GlobalStates.program_dir}\pyftpd.log", encoding="utf-8"
            )
            file_handler.setFormatter(formatter)
            logger.addHandler(file_handler)

      def set_ip(self, ip):
            self.__ip = ip

      def build(self):
            self.__handler.authorizer = self.__authorizer
            # self.__handler.banner = "pyftpdlib основанный на ftpd."

            self.__server = ThreadedFTPServer((self.__ip, 21), self.__handler)

            self.__server.max_cons = 256
            self.__server.max_cons_per_ip = 5

      def add_user(self, username, password, homedir, perm):
            self.__authorizer.add_user(username, password, homedir, perm)

      def remove_all_users(self):
            self.__authorizer.user_table.clear()

      def run(self):
            self.__server_thread = threading.Thread(
                  target=self.__server.serve_forever
            )
```

```
        self.__server_thread.start()

    def stop(self):
        logging.shutdown()
        self.__server.close_all()
```

// main.pyw – Модуль ініціалізації серверу

```
import sys
import os

from PyQt6.QtWidgets import QApplication

from src.GlobalStates import GlobalStates
from ui.ServerWindow import ServerWindow
from ui.UserDb import UserDb

if __name__ == '__main__':
      GlobalStates.program_dir = os.path.expandvars("%APPDATA%\FTPWinServerGui")

      if not os.path.exists(GlobalStates.program_dir):
            os.makedirs(GlobalStates.program_dir)

      GlobalStates.user_db = UserDb(GlobalStates.program_dir)

      app = QApplication(sys.argv)
      with open("./style/light.qss", "r") as style:
            app.setStyleSheet(style.read())
      ui = ServerWindow()
      ui.show()
      sys.exit(app.exec())
```

// ServerWindow.py – Модуль інтерфейсу серверу

```
from datetime import datetime
import re
from markdown_it import MarkdownIt

from PyQt6.QtWidgets import QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
QSizePolicy, QPlainTextEdit, QLineEdit, QToolButton, QPushButton, QTableView,
QSpacerItem, QHeaderView, QLabel, QMessageBox
from PyQt6.QtCore import QRegularExpression, Qt, QSortFilterProxyModel,
QSettings
from PyQt6.QtGui import QRegularExpressionValidator, QStandardItemModel,
QStandardItem, QIcon, QPixmap, QTransform, QFontDatabase, QCloseEvent

from src.GlobalStates import GlobalStates
from src.Server import Server
from ui.Interceptor import Interceptor
from ui.CreateUserFormDialog import CreateUserFormDialog
from ui.UpdateUserFormDialog import UpdateUserFormDialog
from ui.AboutUserFormDialog import AboutUserFormDialog
from ui.frames.ConsoleFrame import ConsoleFrame
from ui.frames.BaseLineFrame import BaseLineFrame

class ServerWindow(QMainWindow):
      def __init__(self):
            super(ServerWindow, self).__init__()

            ####################
            # Initialization
            ####################
            # Initialization of program font
            QFontDatabase.addApplicationFont("./font/Montserrat-Medium.ttf")
            QFontDatabase.addApplicationFont("./font/JetBrainsMono-Light.ttf")
            # print(QFontDatabase.applicationFontFamilies(id)[0])

            # Initialization of Logger
            self.STDOUT = Interceptor()
            self.STDERR = Interceptor()
            self.STDOUT.writing.connect(self.intercept_writing)
            self.STDERR.writing.connect(self.intercept_writing)

            # Initialization of dialog windows
            self.create_user_form_dialog = CreateUserFormDialog()
            self.update_user_form_dialog = UpdateUserFormDialog()
            self.about_user_form_dialog = AboutUserFormDialog()

            # Initialization of program_settings
            self.settings = QSettings("Kalynovsky Valentin", "FTES WSG")

            # Initialization of Server
            self.serv = Server("", self.STDOUT, self.STDERR)
            self.serv.set_log()

            ##############################
            #
            # Server layout
            #
            ##############################
            self.server_layout = QVBoxLayout()

            self.console_frame = ConsoleFrame()
            self.console_frame.query_string.setText("fteswsg@God:~$")
            self.server_layout.addWidget(self.console_frame)

            self.serving_layout = QHBoxLayout()

            self.ip_address = QLineEdit(self)
            self.ip_address.setSizePolicy(
                  QSizePolicy.Policy.Minimum,
```

```
      QSizePolicy.Policy.Minimum
)
self.ip_address.setPlaceholderText("Enter the IP")
self.ip_address.setValidator(
      QRegularExpressionValidator(
            QRegularExpression(
                  r"^192\.168\.
                    (?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
                  \.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$"
            )
      )
)
self.ip_address.setText(
      self.settings.value("ip_address", "192.168.")
)
self.serving_layout.addWidget(self.ip_address)

self.serving_butt = QPushButton("Start server", self)
self.serving_butt.setSizePolicy(
      QSizePolicy.Policy.Expanding,
      QSizePolicy.Policy.Minimum
)
self.serving_butt.setCheckable(True)
self.serving_butt.clicked.connect(self.serving_butt_clicked)
self.serving_layout.addWidget(self.serving_butt)

self.server_layout.addLayout(self.serving_layout)

##############################
#
# Users layout
#
##############################
self.users_layout = QVBoxLayout()

###################
# Search line
###################
self.attention_line = QLabel(self)
self.attention_line.setText("For changes to the user database to
      take effect,\nyou need to restart the server.")
self.attention_line.setObjectName("attention_line")
self.attention_line.setVisible(False)
self.users_layout.addWidget(self.attention_line)

self.search_layout = QHBoxLayout()

self.icon_frame = BaseLineFrame()
self.icon_frame.line_frame_icon.setPixmap(
      QPixmap("./icon/search_user_24.svg")
)
self.icon_frame.line_frame_field.setPlaceholderText(
      "Search username"
)
self.icon_frame.line_frame_field.setFixedWidth(160)
self.icon_frame.line_frame_field.setSizePolicy(
      QSizePolicy.Policy.Minimum,
      QSizePolicy.Policy.Minimum
)
self.icon_frame.line_frame_field.textChanged.connect(
      self.search_line_textChanged
)
self.search_layout.addWidget(self.icon_frame)

self.spacer = QSpacerItem(
      250,
      20,
      QSizePolicy.Policy.Preferred,
      QSizePolicy.Policy.Minimum
```

```
)
self.search_layout.addSpacerItem(self.spacer)

self.help_tool = QToolButton(self)
self.help_tool.setIcon(QIcon(QPixmap("./icon/help_24.svg")))
self.help_tool.clicked.connect(self.help_tool_clicked)
self.search_layout.addWidget(self.help_tool)

self.info_tool = QToolButton(self)
self.info_tool.setIcon(QIcon(QPixmap("./icon/info_24.svg")))
self.info_tool.clicked.connect(self.info_tool_clicked)
self.search_layout.addWidget(self.info_tool)

self.users_layout.addLayout(self.search_layout)

####################
# Creating a user table
####################
# Reading the entire database
user_list = GlobalStates.user_db.get_all_users()

# Creating a data model
self.user_model = QStandardItemModel(len(user_list), 4)
self.user_model.setHorizontalHeaderLabels([
      "Username",
      "Status",
      "RETR",
      "DELE",
      "STOR"
])

# Filling the model with data from the list of users
for row, user in enumerate(user_list):
      self.process_table_row(row, user)

# Adding the model to the filter shell
self.proxy_model = QSortFilterProxyModel()
self.proxy_model.setSourceModel(self.user_model)

# Displaying the data shell in a table
self.user_list = QTableView(self)
self.user_list.setSizePolicy(
      QSizePolicy.Policy.Minimum,
      QSizePolicy.Policy.Expanding
)
self.user_list.setModel(self.proxy_model)
self.user_list.horizontalHeader().setSectionResizeMode(
      QHeaderView.ResizeMode.ResizeToContents
)
GlobalStates.user_db.new.connect(self.user_db_new)
GlobalStates.user_db.dirty.connect(self.user_db_dirty)
GlobalStates.user_db.deleted.connect(self.user_db_deleted)
self.users_layout.addWidget(self.user_list)

####################
# Adding a tool buttons
####################
self.tool_layout = QHBoxLayout()

self.straight_sort_tool = QToolButton(self)
self.straight_sort_tool.setIcon(
      QIcon(QPixmap("./icon/sort_24.svg"))
)
self.straight_sort_tool.clicked.connect(
      self.straight_sort_tool_clicked
)
self.tool_layout.addWidget(self.straight_sort_tool)

self.reverse_sort_tool = QToolButton(self)
```

```
self.reverse_sort_tool.setIcon(
      QIcon(QPixmap("./icon/sort_24.svg")
      .transformed(QTransform().scale(1, -1)))
)
self.reverse_sort_tool.clicked.connect(
      self.reverse_sort_tool_clicked
)
self.tool_layout.addWidget(self.reverse_sort_tool)

self.reset_sort_tool = QToolButton(self)
self.reset_sort_tool.setIcon(QIcon(QPixmap("./icon/reset_24.svg")))
self.reset_sort_tool.clicked.connect(self.reset_sort_tool_clicked)
self.tool_layout.addWidget(self.reset_sort_tool)

self.spacer = QSpacerItem(
      250,
      20,
      QSizePolicy.Policy.Preferred,
      QSizePolicy.Policy.Minimum
)
self.tool_layout.addSpacerItem(self.spacer)

self.add_user_tool = QToolButton(self)
self.add_user_tool.setIcon(QIcon(QPixmap("./icon/user_add_24.svg")))
self.add_user_tool.clicked.connect(self.add_user_tool_clicked)
self.tool_layout.addWidget(self.add_user_tool)

self.update_user_tool = QToolButton(self)
self.update_user_tool.setIcon(
      QIcon(QPixmap("./icon/user_update_24.svg"))
)
self.update_user_tool.clicked.connect(self.update_user_tool_clicked)
self.tool_layout.addWidget(self.update_user_tool)

self.about_user_tool = QToolButton(self)
self.about_user_tool.setIcon(
      QIcon(QPixmap("./icon/user_about_24.svg"))
)
self.about_user_tool.clicked.connect(self.about_user_tool_clicked)
self.tool_layout.addWidget(self.about_user_tool)

self.remove_user_tool = QToolButton(self)
self.remove_user_tool.setIcon(
      QIcon(QPixmap("./icon/user_remove_24.svg"))
)
self.remove_user_tool.clicked.connect(self.remove_user_tool_clicked)
self.tool_layout.addWidget(self.remove_user_tool)

self.users_layout.addLayout(self.tool_layout)

##############################
#
# Main window customization
#
##############################
self.main_layout = QHBoxLayout()
# self.main_layout.setContentsMargins(20, 6, 6, 6)
self.main_layout.addLayout(self.server_layout)
self.main_layout.addLayout(self.users_layout)

self.central_widget = QWidget()
self.central_widget.setLayout(self.main_layout)
self.setCentralWidget(self.central_widget)
self.setWindowTitle(
      "FTES WSG - File Transfer EcoSystem Windows Server Graphic"
)
self.setMinimumSize(1280, 720)
self.setFocus()
```

```
##############################
#
# Implementations
#
##############################
###################
# Implementations of Server functional
###################
def intercept_writing(self, text):
      """Implementation of log writing"""
      self.console_frame.console_output.appendPlainText(text.strip())

      text_re = re.search(r"\[.*\] .*-\[(.*?)\]", text)
      if text_re:
            username = text_re.group(1)
            if username:
                  GlobalStates.user_db.silent_spy_update(
                        username, {"user_logs": text}
                  )
                  if re.search(r"logged in", text):
                        GlobalStates.user_db.set_user_status(
                              username, True
                        )
                        GlobalStates.user_db.set_user_date(
                              username,
                              {
                                    "last_login_date":
                                    datetime.now()
                                    .replace(microsecond=0)
                              }
                        )
                  if re.search(r"session closed", text):
                        GlobalStates.user_db.set_user_status(
                              username, False
                        )
                        GlobalStates.user_db.set_user_time(
                              username,
                              {
                                    "login_time":
                                    datetime.now()
                                    .replace(microsecond=0)
                              }
                        )

                  if command_match := re.search(
r"\[.*?\] (STOR|RETR) .*?completed=(\d+) bytes=(\d+) seconds=([\d.]+)",
                        text
                  ):
                        operation_type = command_match.groups()
                        completed = command_match.groups()
                        bytes_transferred = command_match.groups()
                        seconds_transferred = command_match.groups()
                        match operation_type:
                              case "STOR":
                                    GlobalStates
                                    .user_db
                                    .number_recalculate(
                                          username, {
                                                "upload_count_successful":
                                                int(completed),
                                                "upload_size":
                                                int(bytes_transferred),
                                                "upload_time":
                                                float(seconds_transferred)
                                          }
                                    )
                              case "RETR":
                                    GlobalStates
                                    .user_db
```

```
                                    .number_recalculate(
                                          username, {
                                                "down_count_successful":
                                                int(completed),
                                                "download_size":
                                                int(bytes_transferred),
                                                "download_time":
                                                float(seconds_transferred)
                                          }
                                    )
                  else:
                        text_re = re.search(r"USER '(.*?)'", text)
                        if text_re:
                              username = text_re.group(1)
                              if username:
                                    GlobalStates
                                    .user_db
                                    .silent_spy_update(
                                          username, {"user_logs": text}
                                    )

      def serving_butt_clicked(self) -> None:
            """Implementation of connect/disconnect button"""
            if self.serving_butt.isChecked():
                  self.serving_butt.setText("Stop server")
                  self.attention_line.setVisible(True)

                  if self.ip_address.text() != "":
                        self.serv.set_ip(self.ip_address.text())

                  self.serv.build()

                  users = GlobalStates.user_db.get_all_users()
                  for user in users:
                        permissions_dict = {
                              "e": user.permission_CWD,
                              "l": user.permission_LIST,
                              "r": user.permission_RETR,
                              "a": user.permission_APPE,
                              "d": user.permission_DELE,
                              "f": user.permission_RNFR,
                              "m": user.permission_MKD,
                              "w": user.permission_STOR,
                              "M": user.permission_CHMOD,
                              "T": user.permission_MFMT
                        }
                        permission = ''.join(
                              key for key,
                              value in permissions_dict.items() if value
                        )
                        self.serv.add_user(
                              user.username,
                              user.password,
                              user.home_dir,
                              permission
                        )

                  self.serv.run()

            else:
                  self.serving_butt.setText("Start server")
                  self.attention_line.setVisible(False)
                  self.serv.stop()
                  self.serv.remove_all_users()

      ###################
      # Implementations of Users functional
      ###################
      def search_line_textChanged(self):
```

```
            self.proxy_model.setFilterRegularExpression(
                  self.icon_frame.line_frame_field.text()
            )

      def help_tool_clicked(self):
            self.create_about_program("help.md")

      def info_tool_clicked(self):
            self.create_about_program("readme.md")

      def create_about_program(self, file_name):
            with open(file_name, "r") as f:
                  text = f.read()
                  md = MarkdownIt()
                  html = md.render(text)

            about_program_container = QMessageBox()
            about_program = QMessageBox()
            about_program.about(about_program_container, "About program", html)

      def user_db_new(self, user_obj):
            self.process_table_row(self.user_list.model().rowCount(), user_obj)

            self.user_list.update()

      def user_db_dirty(self, user_obj):
            self.process_table_row(
                  self
                  .user_model
                  .findItems(user_obj.username, column=0)[0]
                  .row(),
                  user_obj
            )

            self.user_list.update()

      def user_db_deleted(self, username):
            self.user_model.removeRow(
                  self.user_model.findItems(username, column=0)[0].row()
            )

            self.user_list.update()

      def straight_sort_tool_clicked(self):
            self.user_list.model().sort(0, Qt.SortOrder.AscendingOrder)

      def reverse_sort_tool_clicked(self):
            """Implementation of sort in reverse order a user table"""
            self.user_list.model().sort(0, Qt.SortOrder.DescendingOrder)

      def reset_sort_tool_clicked(self):
            self.user_list.model().sort(-1)

      def add_user_tool_clicked(self):
            self.create_user_form_dialog.show()

      def update_user_tool_clicked(self):
            selected_indexes = self.user_list.selectionModel().selectedRows()
            if len(selected_indexes) == 0:
                  return
            username_index = self.user_model.index(selected_indexes[0].row(), 0)
            username = self.user_model.data(username_index)
            self.update_user_form_dialog.set_username(username)
            self.update_user_form_dialog.show()

      def about_user_tool_clicked(self):
            selected_indexes = self.user_list.selectionModel().selectedRows()
            if len(selected_indexes) == 0:
                  return
```

```
        username_index = self.user_model.index(selected_indexes[0].row(), 0)
        username = self.user_model.data(username_index)
        user = GlobalStates.user_db.get_user(username)
        self.about_user_form_dialog.set_username(user)
        self.about_user_form_dialog.show()

    def remove_user_tool_clicked(self):
        selected_indexes = self.user_list.selectionModel().selectedRows()
        usernames = []

        for index in selected_indexes:
            username_index = self.user_model.index(index.row(), 0)
            username = self.user_model.data(username_index)
            usernames.append(username)

        for username in usernames:
            GlobalStates.user_db.remove_user(username)

    def closeEvent(self, a0: QCloseEvent) -> None:
        if self.serving_butt.isChecked():
            self.serv.stop()
            self.serv.remove_all_users()

        GlobalStates.user_db.close()

        self.settings.setValue("ip_address", self.ip_address.text())

        super().closeEvent(a0)

    def process_table_row(self, row, user):
        for col, data in enumerate([
            user.username,
            user.online,
            user.permission_RETR,
            user.permission_DELE,
            user.permission_STOR
        ]):
            if col == 0:
                item = QStandardItem(str(data))
            elif col == 1:
                if data:
                    item = QStandardItem(
                        QIcon(QPixmap("./icon/online_24.svg")), ""
                    )
                else:
                    item = QStandardItem(
                        QIcon(QPixmap("./icon/offline_24.svg")), ""
                    )
            else:
                if data:
                    item = QStandardItem(
                        QIcon(QPixmap("./icon/check_24.svg")), ""
                    )
                else:
                    item = QStandardItem(
                        QIcon(QPixmap("./icon/cancel_24.svg")), ""
                    )
            item.setEditable(False)
            if col != 0:
                item.setSelectable(False)
            self.user_model.setItem(row, col, item)
```

// Interceptor.py – Модуль імітатора консолі

```
from PyQt6.QtCore import QObject, pyqtSignal

class Interceptor(QObject):
      writing = pyqtSignal(str)

      def __init__(self):
            super().__init__()

      def write(self, text):
            self.writing.emit(text)

      def flush(self):
            pass
```

// UserDb.py – Модуль БД серверу та керування БД

```
from sqlalchemy import create_engine, update, Column, Integer, Float, String,
Text, Boolean, DateTime, Interval
from sqlalchemy.orm import declarative_base
from sqlalchemy.orm import sessionmaker

from PyQt6.QtCore import QObject, pyqtSignal

Base = declarative_base()

class User(Base):
      __tablename__ = "users"
      id = Column(Integer, primary_key=True)
      online = Column(Boolean)
      username = Column(String)
      password = Column(String)
      home_dir = Column(String)
      date_of_creation = Column(DateTime)
      date_of_change = Column(DateTime)
      last_login_date = Column(DateTime)
      login_time = Column(Interval)
      upload_count_successful = Column(Integer)  # STOR
      upload_size = Column(Integer)  # STOR
      upload_time = Column(Float)  # STOR
      download_count_successful = Column(Integer)  # RETR
      download_size = Column(Integer)  # RETR
      download_time = Column(Float)  # RETR
      permission_CWD = Column(Boolean)
      permission_LIST = Column(Boolean)
      permission_RETR = Column(Boolean)
      permission_APPE = Column(Boolean)
      permission_DELE = Column(Boolean)
      permission_RNFR = Column(Boolean)
      permission_MKD = Column(Boolean)
      permission_STOR = Column(Boolean)
      permission_CHMOD = Column(Boolean)
      permission_MFMT = Column(Boolean)
      user_logs = Column(Text)

class UserDb(QObject):
      new = pyqtSignal(User)
      dirty = pyqtSignal(User)
      deleted = pyqtSignal(str)

      def __init__(self, program_dir):
            super(UserDb, self).__init__()

            self.engine = create_engine(
                  f"sqlite:///{program_dir}\\users_database.db"
            )
            Base.metadata.create_all(self.engine)
            self.Session = sessionmaker(bind=self.engine)
            self.session = self.Session()

      def get_user(self, username):
            return self.session.query(User).filter_by(username=username).first()

      def get_all_users(self):
            return self.session.query(User).all()

      def check_username(self, username):
            existing_user = self
                  .session
                  .query(User.username)
                  .filter_by(username=username)
                  .scalar()
            return existing_user is None
```

```
def create_user(self, username, user_data):
      new_user = User(username=username, **user_data)
      self.session.add(new_user)
      self.session.commit()
      self.new.emit(new_user)

def set_user_status(self, username, online: bool):
      user = self.get_user(username)
      if user:
            # Оновлюємо статус користувача
            setattr(user, "online", online)

            # Фіксуємо значення в БД
            self.session.commit()
            self.dirty.emit(user)

def update_user(self, username, new_data):
      user = self.get_user(username)
      if user:
            # Оновлюємо поля користувача
            for key, value in new_data.items():
                  setattr(user, key, value)

            # Фіксуємо значення в БД
            self.session.commit()
            self.dirty.emit(user)

def silent_spy_update(self, username, new_data):
      user = self.get_user(username)
      if user:
            for key, value in new_data.items():
                  # Створюємо об'єкт запиту для оновлення поля
                  update_query = update(User)
                        .where(User.username == username)
                  # Оновлюємо поле, додаючи нове значення
                  update_query = update_query
                        .values({key: User.__dict__[key] + value})
                  # Виконуємо запит
                  self.session.execute(update_query)

            # Фіксуємо значення в БД
            self.session.commit()

def number_recalculate(self, username, new_data):
      user = self.get_user(username)
      if user:
            # Оновлюємо поля користувача
            for key, value in new_data.items():
                  setattr(user, key, getattr(user, key) + value)

            # Фіксуємо значення в БД
            self.session.commit()

def set_user_date(self, username, new_data):
      user = self.get_user(username)
      if user:
            # Оновлюємо поля користувача
            for key, value in new_data.items():
                  setattr(user, key, value)

            # Фіксуємо значення в БД
            self.session.commit()

def set_user_time(self, username, new_data):
      user = self.get_user(username)
      if user:
            # Оновлюємо поля користувача
            for key, value in new_data.items():
                  tim = value - getattr(user, "last_login_date")
```

```
                        setattr(user, key, tim)

                # Фіксуємо значення в БД
                self.session.commit()

    def remove_user(self, username):
          user = self.session.query(User).filter_by(username=username).first()
          if user:
                self.session.delete(user)
                self.deleted.emit(username)
                self.session.commit()

    def close(self):
          self.session.close()
```