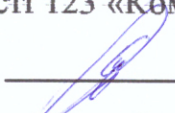


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект файлового менеджера для Linux-
систем на базі протоколу SSH»

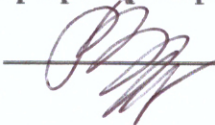
Виконав: студент 4-го курсу групи КІ-21-2
спеціальності 123 «Комп'ютерна інженерія»

 Нікіта ДРАЧЕВСЬКИЙ

Керівник: доцент кафедри КІ та КБ, к.т.н., доцент

 Юрій БРОДСЬКИЙ

Рецензент: професор кафедри КІ та КБ, д.т.н., доцент

 Володимир ВОРОТНИКОВ

Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко

3 березня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Драчевський Нікіта Володимирович

Тема роботи: Проєкт файлового менеджера для Linux-систем на базі протоколу
SSH

затверджена Наказом університету від 3 березня 2025 р. №79/с

Термін здачі студентом закінченої роботи 25 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

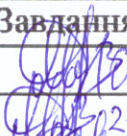
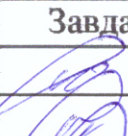
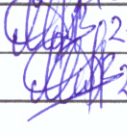
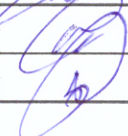
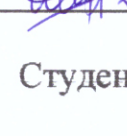
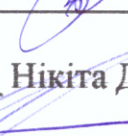
вихідні дані для роботи;

1. Мобільний файловий менеджеру для Android

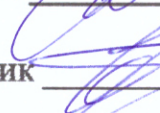
Обладнання та програмне забезпечення:

1. Windows 11
2. Android Studio
3. Java
4. Емулятор Android
5. VirtualBox
6. Ubuntu Server 22.04 LTS, Debian 12
7. OpenSSH Server
8. SSHJ (бібліотека)

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

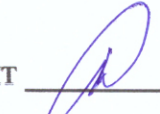
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Маєвський О.В.	 02.04.2025	 02.04.2025
2	Маєвський О.В.	 21.04.2025	 21.04.2025
3	Маєвський О.В.	 24.05.2025	 24.05.2025

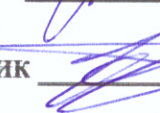
Студент  Нікіта ДРАЧЕВСЬКИЙ

Керівник  Юрій БРОДСЬКИЙ

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	2.04.2025-2.04.2025	Виконано
2	Аналіз літератури та подібних програм	3.04.2025-10.04.2025	Виконано
3	Проектування архітектури додатку	11.04.2025-20.04.2025	Виконано
4	Підготовка серверного середовища	21.04.2025-3.05.2025	Виконано
5	Реалізація підключення та роботи з файлами	4.05.2025-11.05.2025	Виконано
6	Реалізація функцій	12.05.2025-20.05.2025	Виконано
7	Тестування додатку	21.05.2025-23.05.2025	Виконано
8	Відлагодження	24.05.2025-10.06.2025	Виконано
9	Написання пояснювальної записки	11.06.2025-21.06.2025	Виконано
10	Захист	25.06.2025	

Студент  Нікіта ДРАЧЕВСЬКИЙ

Керівник  Юрій БРОДСЬКИЙ

АНОТАЦІЯ

У межах кваліфікаційної роботи бакалавра було розроблено мобільний застосунок для керування файлами на віддалених Linux-серверах через протокол SSH. Метою випускної кваліфікаційної роботи є створення зручного та функціонального інструменту для базового адміністрування серверного середовища з мобільного пристрою на базі операційної системи Android.

Пояснювальна записка містить 106 сторінок, включає 71 рисуноків, 4 таблиць. Список використаних джерел налічує 21 найменування і займає 2 сторінки.

У роботі визначено основні завдання для побудови клієнт-серверної взаємодії через SSH на мобільних пристроях; проаналізовано базові протоколи передачі даних та архітектурні рішення для Android-додатків; розглянуто особливості реалізації SFTP-доступу до файлової системи сервера; показано розробку функціоналу файлового менеджера з можливістю перегляду, створення, копіювання, перейменування та видалення файлів; наведено результати тестування застосунку в симульованому середовищі з використанням VirtualBox та Ubuntu Server.

Роботу виконано з дотриманням архітектурного патерну MVVM. Було реалізовано централізовану логіку з'єднання, обробки подій та збереження параметрів підключення. У результаті отримано стабільно працюючий мобільний застосунок, що може бути використаний для навчальних або практичних цілей у сфері системного адміністрування.

Ключові слова: ANDROID, SSH, МОБІЛЬНИЙ ЗАСТОСУНОК, SFTP, АДМІНІСТРУВАННЯ, СЕРВЕР, ФАЙЛОВИЙ МЕНЕДЖЕР, MVVM, VIRTUALBOX, UBUNTU.

					ККІКБ.КР.Б - 123 - 03 - ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Проект файлового менеджера для Linux-систем на базі протоколу SSH	Літ.	Арк.	Аркушів
Розроб.		Драчевський Н.В.		21.06				
Керівник		Бродський Ю.Б.		21.06			4	106
Реценз.		Воротніков В.В.		21.06		ФІКТ, Гр. КІ-21-2		
Н. контр.		Бродський Ю.Б.		21.06				
Затверд.		Єфіменко А.А.		21.06				

ANNOTATION

As part of the bachelor's qualification thesis, a mobile application was developed to manage files on remote Linux servers via the SSH protocol. The objective of the thesis is to design and implement a convenient and functional tool for basic server administration from an Android-based mobile device.

The explanatory report includes 106 pages, contains 71 figures, 4 tables. The list of references consists of 21 sources and is placed on 2 pages.

The thesis defines key tasks for building client-server interaction using SSH on mobile platforms; analyzes core data transfer protocols and architectural solutions for Android applications; examines the implementation of SFTP access to a server's file system; demonstrates the development of core file manager functionality including viewing, creating, copying, renaming, and deleting files; presents the results of testing the application in a simulated environment using VirtualBox and Ubuntu Server.

The application was built using the MVVM architecture pattern. A centralized logic for connection handling, user input processing, and preference storage was implemented. As a result, a stable and functional mobile application was developed, suitable for use in educational or practical scenarios related to system administration.

Keywords: ANDROID, SSH, MOBILE APPLICATION, SFTP, ADMINISTRATION, SERVER, FILE MANAGER, MVVM, VIRTUALBOX, UBUNTU.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ МОБІЛЬНОГО ФАЙЛОВОГО МЕНЕДЖЕРА З ВИКОРИСТАННЯМ SSH.....	11
1.1 Протокол SSH: принципи роботи, безпека та застосування	11
1.1.1 Основні властивості SSH та його роль у віддаленому адмініструванні..	11
1.1.2 Відмінність між SSH і SFTP: передача команд і файлів	13
1.1.3 Аналоги та переваги SSH у порівнянні з іншими технологіями.....	16
1.2 Android як платформа для системного адміністрування.....	21
1.3. Технічне завдання та функціональні вимоги до мобільного файлового менеджера.....	25
РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ	28
2.1 Структура застосунку: архітектура, активності, ViewModel, класи з'єднання	28
2.1.2 Екран авторизації: збереження доменного імені, логіна, пароля та порту	31
2.1.2 Екран перегляду файлів: завантаження директорій, адаптер списку	35
2.1.3 Екран вибору користувача: база даних Room, RecyclerView	40
2.2 Розробка класів керування з'єднанням: SSHManager, RemoteUser.....	43
2.3 Реалізація функціоналу файлового менеджера	46
2.3.1 Основні дії з файлами та папками: створення, видалення, перейменування, копіювання, переміщення	46
2.3.2 Реалізація механізмів передачі файлів: завантаження та вивантаження.	51
2.4 Збереження даних користувачів через Room.....	55
2.5 Збереження даних користувачів через Room.....	57
2.6 Підготовка серверного середовища для тестування	59
РОЗДІЛ 3 ТЕСТУВАННЯ ТА ВДОСКОНАЛЕННЯ ФАЙЛОВОГО МЕНЕДЖЕРА	66
3.1 Перевірка з'єднання з сервером, передача команд	66
3.2 Тестування роботи з файлами та обробка помилок	70

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

3.3 Оцінка продуктивності застосунку та UX-досвід користувача	81
ВИСНОВКИ.....	85
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	87
ДОДАТКИ.....	89

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система

ПЗ – програмне забезпечення

SSH – Secure Shell

SFTP – SSH File Transfer Protocol

IP – Internet Protocol

UI – User Interface

UX – User Experience

VM – Virtual Machine

DB – Database

MVVM – Model-View-ViewModel

DAO – Data Access Object

URI – Uniform Resource Identifier

IDE – Integrated Development Environment

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. У сучасних умовах динамічного розвитку інформаційних технологій усе більшої популярності набувають мобільні рішення для адміністрування серверної інфраструктури. Зважаючи на постійну потребу у швидкому доступі до віддалених ресурсів, зокрема з боку системних адміністраторів та DevOps-фахівців, розробка мобільного файлового менеджера з підтримкою SSH-протоколу є актуальною та доцільною. Такий інструмент дозволяє забезпечити базову взаємодію з сервером незалежно від місцезнаходження та наявності повноцінного комп'ютера, що підвищує ефективність управління IT-інфраструктурою.

Метою кваліфікаційної роботи є створення мобільного застосунку для платформи Android, який дозволяє здійснювати базове адміністрування файлової системи віддаленого Linux-сервера через протокол SSH із використанням інтуїтивно зрозумілого графічного інтерфейсу.

Визначена мета обумовлює наступні завдання:

- проаналізувати можливості використання Android-пристроїв для віддаленого адміністрування;
- дослідити принципи роботи протоколу SSH та технології захищеного доступу;
- спроектувати архітектуру застосунку з використанням патерну MVVM;
- реалізувати функціонал підключення до сервера, перегляду та керування файлами (копіювання, перейменування, видалення, передача);
- забезпечити збереження користувацьких параметрів та історії з'єднань через Room;
- протестувати застосунок у віртуальному середовищі на базі Ubuntu Server.

Об'єктом дослідження є процес мобільного віддаленого адміністрування серверів у Linux-середовищі.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є методи та інструменти реалізації мобільного файлового менеджера для Android, що використовує протокол SSH та SFTP для взаємодії з віддаленим сервером.

У процесі виконання кваліфікаційної роботи було розроблено проєкт мобільного застосунку "SshFileExplorer", який забезпечує авторизацію, перегляд файлової системи сервера, виконання базових операцій над файлами та збереження облікових даних користувачів з можливістю повторного підключення.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ МОБІЛЬНОГО ФАЙЛОВОГО МЕНЕДЖЕРА З ВИКОРИСТАННЯМ SSH

1.1 Протокол SSH: принципи роботи, безпека та застосування

Протокол SSH (Secure Shell) — це мережевий протокол, який забезпечує безпечний обмін даними, віддалене адміністрування серверів та тунелювання мережевого трафіку. Його основними особливостями є використання криптографії, аутентифікація користувача, шифрування з'єднання та перевірка цілісності переданих даних [1].

SSH став стандартом для безпечного адміністрування Unix-подібних систем. Завдяки використанню асиметричної криптографії (ключі RSA, DSA, ED25519) забезпечується висока безпека. Передача паролів у відкритому вигляді відсутня, і можливе застосування ключової авторизації, що ще більше знижує ризик компрометації доступу.

Протокол працює поверх TCP та зазвичай використовує порт 22. Підключення здійснюється за допомогою SSH-клієнта (наприклад, `ssh user@host`) до SSH-сервера на віддаленій машині. В рамках однієї сесії можна виконувати команди, передавати файли, а також створювати зашифровані тунелі (рис.1.1).

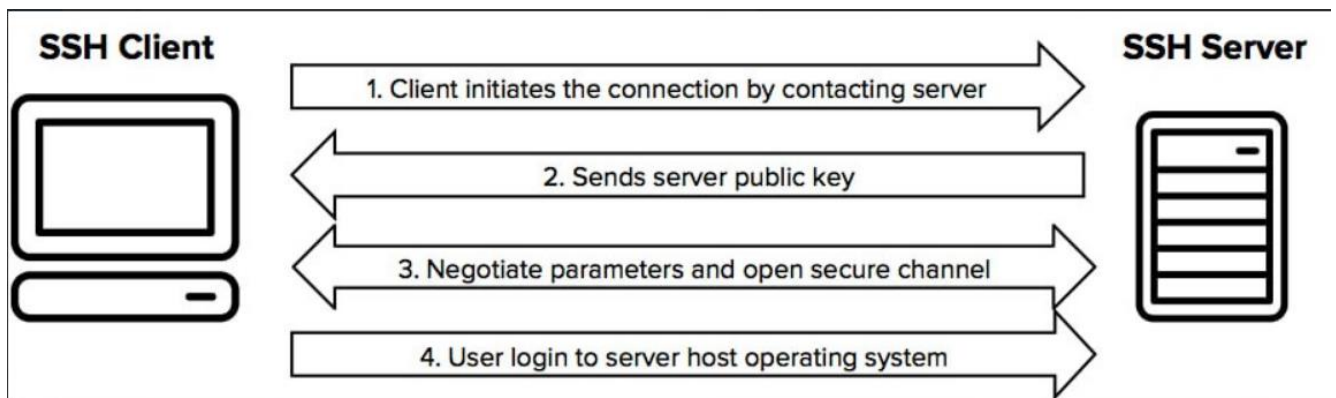


Рисунок 1.1 – Принцип роботи SSH-з'єднання [2]

1.1.1 Основні властивості SSH та його роль у віддаленому адмініструванні

Протокол SSH (Secure Shell) є фундаментальним інструментом у сфері мережевої безпеки, що забезпечує захищений канал зв'язку між двома віддаленими

комп'ютерами. Його функціональні властивості роблять його незамінним для адміністрування систем, передачі даних та доступу до ресурсів у недовірених мережах. До основних можливостей SSH належать [3]:

- 1. Шифрування інформації.** SSH забезпечує конфіденційність та цілісність даних завдяки застосуванню сучасних криптографічних алгоритмів. Зокрема, для симетричного шифрування використовуються такі алгоритми, як AES (Advanced Encryption Standard), а для асиметричного (публічного) шифрування та обміну ключами – RSA, ECDSA (Elliptic Curve Digital Signature Algorithm) та Ed25519. Це дозволяє ефективно захищати весь трафік, що передається між клієнтом і сервером, запобігаючи несанкціонованому перехопленню, прослуховуванню та атакам типу "людина посередині" (Man-in-the-Middle, MITM).
- 2. Аутентифікація користувача.** Для забезпечення доступу до віддалених систем SSH використовує надійні механізми аутентифікації. Окрім традиційної аутентифікації за допомогою паролів, протокол підтримує, і це є кращою практикою, аутентифікацію на основі пари криптографічних ключів (приватного та публічного). Цей метод значно підвищує рівень безпеки, усуваючи ризики, пов'язані з компрометацією слабких або скомпрометованих паролів, а також дозволяє реалізувати безпарольний доступ для автоматизованих систем.
- 3. Захищена передача файлів.** SSH інтегрує в собі функціонал для безпечного копіювання файлів. Для цього використовуються протоколи scp (Secure Copy Protocol) та sftp (Secure File Transfer Protocol), які працюють поверх того ж шифрованого SSH-каналу. Це гарантує конфіденційність та цілісність даних під час їх передачі між локальною та віддаленою системами, що є критично важливим для пересилання конфіденційної інформації або великих обсягів даних.
- 4. Порт-форвардинг та тунелювання.** Однією з найпотужніших функцій SSH є можливість переадресації портів (port forwarding), або тунелювання. Ця функція дозволяє створювати захищені тунелі для перенаправлення

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

мережевого трафіку через шифроване SSH-з'єднання. Це дає змогу обходити обмеження фаєрволів, забезпечувати безпечний доступ до внутрішніх ресурсів приватної мережі ззовні, або тунелювати незашифрований трафік інших протоколів (наприклад, HTTP, POP3) через захищений SSH-канал, підвищуючи їхню безпеку.

5. Підтримка автоматизації. Завдяки своїй архітектурі та можливості безпарольної аутентифікації за ключами, SSH є ідеальним інструментом для автоматизації рутинних завдань адміністрування. Його можна легко інтегрувати в скрипти (bash, Python тощо) для виконання таких операцій, як оновлення програмного забезпечення на віддалених серверах, автоматизоване резервне копіювання даних, розгортання програмних продуктів або моніторинг систем. Це суттєво підвищує ефективність управління IT-інфраструктурою.

6. Кросплатформеність. Важливою перевагою SSH є його кросплатформенність. Протокол та відповідні клієнтські й серверні реалізації доступні та функціонують на більшості сучасних операційних систем, включаючи Linux, UNIX-подібні системи, macOS та Windows. На Windows доступність SSH забезпечується як через інтегрований OpenSSH, так і через сторонні клієнти, такі як PuTTY. Це гарантує універсальність застосування SSH у різноманітних мережевих середовищах.

1.1.2 Відмінність між SSH і SFTP: передача команд і файлів

Протоколи SSH (Secure Shell) та SFTP (SSH File Transfer Protocol) є невід'ємними компонентами сучасної інформаційної безпеки, що забезпечують захищену взаємодію з віддаленими системами. Хоча вони тісно пов'язані, їхні функціональні призначення та сфера застосування суттєво розрізняються. Обидва протоколи базуються на механізмах безпечного з'єднання, але реалізують різні підходи до передачі даних та взаємодії з віддаленою системою [4].

SSH (Secure Shell) є багатофункціональним мережевим протоколом прикладного рівня, що призначений для забезпечення безпечного віддаленого

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

керування операційними системами. Його основна функція полягає у створенні криптографічно захищеного каналу для передачі текстових команд. Це дозволяє адміністраторам та користувачам авторизуватися на віддаленому сервері та взаємодіяти з ним через командний інтерфейс (shell) так, ніби вони працюють локально.

Серед ключових можливостей SSH варто виділити:

1. Виконання команд у реальному часі: користувач може надсилати команди операційній системі віддаленого сервера та миттєво отримувати результати їх виконання.
2. Керування процесами та службами: можливість запускати, зупиняти, перезапускати та моніторити системні процеси і служби.
3. Редагування конфігураційних файлів: безпечний доступ до системних файлів для їх редагування, що є критично важливим для налаштування програмного забезпечення та служб.
4. Доступ до системних журналів: перегляд та аналіз лог-файлів для діагностики проблем та моніторингу подій.
5. Можливість автоматизації через сценарії: інтеграція SSH у скрипти (написані на bash, PowerShell, Python тощо) для автоматизації рутинних завдань, що значно підвищує ефективність системного адміністрування.

Натомість SFTP (SSH File Transfer Protocol) – це окремий підпротокол, який функціонує поверх SSH і спеціально розроблений для безпечної та надійної передачі файлів між клієнтом та сервером. На відміну від застарілого FTP (File Transfer Protocol), який передає дані у відкритому, незашифрованому вигляді, SFTP успадковує всі переваги безпеки від базового SSH-каналу, забезпечуючи повне шифрування та аутентифікацію.

Основні функціональні можливості SFTP включають:

1. Двостороння передача файлів: безпечне завантаження (upload) файлів з локального комп'ютера на віддалений сервер та їх завантаження (download) з сервера на локальний пристрій.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

2. Управління файловою системою: можливість створювати, видаляти, перейменовувати файли та директорії на віддаленій системі.
3. Зміна прав доступу до файлів: управління дозволами (permissions) для файлів та папок на сервері, що є важливим аспектом безпеки
4. Перегляд вмісту директорій: навігація по файловій системі віддаленого сервера, перегляд списку файлів та піддиректорій.

Для наочнішого розмежування функцій SSH та SFTP наведено порівняльну таблицю 1.1.

Таблиця 1.1 – Розмежування функцій SSH та SFTP

	SSH (Secure Shell)	SFTP (SSH File Transfer Protocol)
Призначення	Віддалене керування системою, виконання команд	Безпечна передача файлів, управління файлами
Тип взаємодії	Текстові команди, виконання скриптів, інтерактивний сеанс	Файлова навігація, завантаження/вивантаження даних, управління директоріями
Протокол	Незалежний протокол прикладного рівня	Працює як підпротокол поверх SSH
Передача файлів	Можлива через допоміжні утиліти (наприклад, scp) або ручними командами	Через спеціалізований інтерфейс для файлових операцій
Гнучкість	Підходить для широкого спектра завдань адміністрування та автоматизації	Спеціалізований для роботи з файлами та каталогами

Таким чином, протоколи SSH та SFTP не є взаємозамінними, а, навпаки, взаємодоповнюють один одного. SSH надає потужний засіб для віддаленого керування та контролю операційною системою, тоді як SFTP спеціалізується на безпечній та ефективній передачі файлів. У практиці системного адміністрування

та розробки обидва протоколи часто використовуються спільно: SSH – для конфігурування серверів, виконання команд та діагностики, а SFTP – для безпечного переміщення файлів конфігурації, системних журналів, даних користувачів або розгортання програмного забезпечення.

1.1.3 Аналоги та переваги SSH у порівнянні з іншими технологіями

Порівняння SSH з альтернативними технологіями віддаленого доступу

У сфері забезпечення віддаленого доступу до комп'ютерних систем існує низка технологій, які певною мірою виконують схожі функції з протоколом SSH. Однак, більшість з них мають суттєві недоліки, особливо у питаннях безпеки, зручності використання та функціональності. Детальний розгляд цих альтернатив дозволяє краще зрозуміти переваги SSH.

1. Telnet

Telnet є одним з найстаріших протоколів віддаленого доступу, що активно використовувався з 1970-х років [5]. Він дозволяє встановлювати з'єднання з віддаленою системою та передавати текстові команди. Проте, ключовим і фатальним недоліком Telnet є повна відсутність шифрування даних. Весь трафік, включаючи імена користувачів, паролі та команди, передається у відкритому вигляді, що робить його надзвичайно вразливим до перехоплення даних та атак типу "людина посередині" (Man-in-the-Middle, MITM).

Порівняння з SSH

- Шифрування: Telnet передає дані у відкритому вигляді, тоді як SSH шифрує весь трафік, забезпечуючи конфіденційність.
- Аутентифікація: Telnet не підтримує надійні методи аутентифікації, такі як ключова аутентифікація, які є стандартними для SSH.
- Функціональність: Telnet обмежений передачею команд і не підтримує такі можливості SSH, як тунелювання, переадресація портів чи захищена передача файлів.

2. FTP (File Transfer Protocol)

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

FTP є класичним протоколом для передачі файлів між комп'ютерами [6]. Подібно до Telnet, FTP передає як дані, так і облікові дані (імена користувачів, паролі) у незашифрованому вигляді, що створює значні ризики для конфіденційності та цілісності інформації. У відповідь на ці проблеми були розроблені захищені альтернативи, такі як SFTP (SSH File Transfer Protocol), який працює поверх SSH, та FTPS (FTP Secure), що використовує протоколи SSL/TLS для шифрування.

Порівняння з SSH

- Шифрування: FTP не шифрує дані, тоді як SFTP (частина екосистеми SSH) є повністю захищеним протоколом передачі файлів.
- Виконання команд: FTP призначений виключно для передачі файлів і не надає можливості виконувати команди або сценарії на віддаленій системі, на відміну від SSH.
- Канали зв'язку: FTP вимагає окремих каналів для керування та передачі даних, тоді як SSH використовує єдиний, мультиплексований та шифрований канал.

3. RDP (Remote Desktop Protocol)

RDP – це протокол, розроблений корпорацією Microsoft, що забезпечує графічний доступ до віддалених робочих столів, переважно на базі операційних систем Windows [7]. Він дозволяє користувачеві бачити графічний інтерфейс віддаленої системи та взаємодіяти з нею, ніби він фізично знаходиться перед нею. Хоча RDP підтримує шифрування для захисту сесії, його основна орієнтація – це графічний інтерфейс користувача (GUI), а не автоматизація чи текстове адміністрування серверів, що є ключовими для SSH.

Порівняння з SSH

- **Тип інтерфейсу:** RDP орієнтований на графічний інтерфейс (GUI), тоді як SSH призначений для роботи з консоллю (CLI), що є більш ефективним для серверного адміністрування та автоматизації.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

- **Швидкодія:** SSH значно легший і швидший, особливо на низькошвидкісних або нестабільних мережових каналах, завдяки мінімальному обсягу переданих даних (текстові команди). RDP, через передачу графічних даних, вимагає більшої пропускної здатності.
- **Кросплатформеність:** SSH доступний та широко використовується на більшості операційних систем (Linux, UNIX, macOS, Windows), тоді як RDP переважно асоціюється з екосистемою Windows.

4. VPN (Virtual Private Network)

VPN – це технологія, що створює захищене з'єднання між пристроєм користувача та приватною мережею, забезпечуючи тунелювання та шифрування всього мережевого трафіку [8]. На відміну від SSH, який захищає окремі сесії або програми, VPN створює загальний зашифрований тунель, через який проходить весь трафік користувача. Таким чином, SSH і VPN не є прямими конкурентами, а швидше взаємодоповнювальними інструментами, де VPN може забезпечити загальну безпеку мережевого підключення, а SSH – безпечний доступ до конкретних серверних ресурсів або виконання команд. SSH також може бути використаний для створення простих тунелів, які в певних сценаріях можуть частково замінити функціонал VPN.

Порівняння з SSH

- **Область шифрування:** VPN забезпечує шифрування всього мережевого трафіку від пристрою користувача до VPN-сервера, тоді як SSH шифрує лише трафік конкретних SSH-сесій або тунелів.
- **Складність налаштування:** SSH, як правило, простіший у налаштуванні для віддаленого доступу до окремих серверів, ніж повноцінна VPN-інфраструктура.
- **Вимоги до програмного забезпечення:** VPN часто вимагає окремого клієнтського програмного забезпечення, тоді як SSH-клієнт часто вже вбудований у більшість операційних систем або легко доступний.

Нижче на таблиці 1.2 наведені ключові переваги використання SSH. А на рис.1.2 основні відмінності між SSH, SFTP, Telnet, FTP, RDP та VPN.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 1.2 – Переваги протоколу SSH

Перевага SSH	Детальний опис
Комплексне шифрування	SSH забезпечує повне та надійне шифрування всього переданого трафіку, включаючи як дані, так і аутентифікаційну інформацію (логіни, паролі, ключі). Це усуває потребу у використанні додаткових засобів захисту та мінімізує ризики перехоплення або компрометації даних злоумисниками
Надійна аутентифікація	Протокол SSH підтримує різноманітні механізми аутентифікації, включаючи, але не обмежуючись, аутентифікацією за допомогою пари криптографічних ключів (приватний/публічний). Цей метод значно перевершує традиційну парольну аутентифікацію за рівнем безпеки, запобігаючи багатьом видам атак (наприклад, брутфорс) та забезпечуючи більш надійний доступ.
Розширена функціональність	Окрім базової функції віддаленого виконання команд, SSH надає широкий спектр додаткових можливостей. До них належать захищена передача файлів (за допомогою SFTP/SCP), тунелювання (переадресація портів), що дозволяє створювати захищені канали для іншого мережевого трафіку, а також ефективна автоматизація рутинних завдань через виконання сценаріїв.

Продовження таблиця 1.2 – Переваги протоколу SSH

Висока кросплатформеність	SSH є універсальним рішенням, що широко підтримується та інтегрований у більшість операційних систем. Він є нативним для UNIX-подібних систем (Linux, macOS, BSD) та повноцінно доступний для Windows як через вбудований клієнт OpenSSH, так і через популярні сторонні реалізації (наприклад, PuTTY), забезпечуючи єдиний стандарт доступу в гетерогенних середовищах.
Ефективність та стабільність	Протокол розроблений для ефективної роботи навіть в умовах низької пропускну здатності мережі або нестабільних з'єднань. Завдяки мінімалістичній передачі даних (в основному текстові команди), SSH забезпечує високу швидкість відгуку та стабільність роботи, що є критично важливим для віддаленого адміністрування у різноманітних мережевих умовах.
Низьке споживання ресурсів	У порівнянні з протоколами, що орієнтовані на графічний інтерфейс (наприклад, RDP), SSH вимагає значно менше системних ресурсів як на клієнтській, так і на серверній стороні. Це робить його ідеальним вибором для керування серверами, які часто працюють без графічної оболонки, а також для використання на пристроях з обмеженими обчислювальними можливостями.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

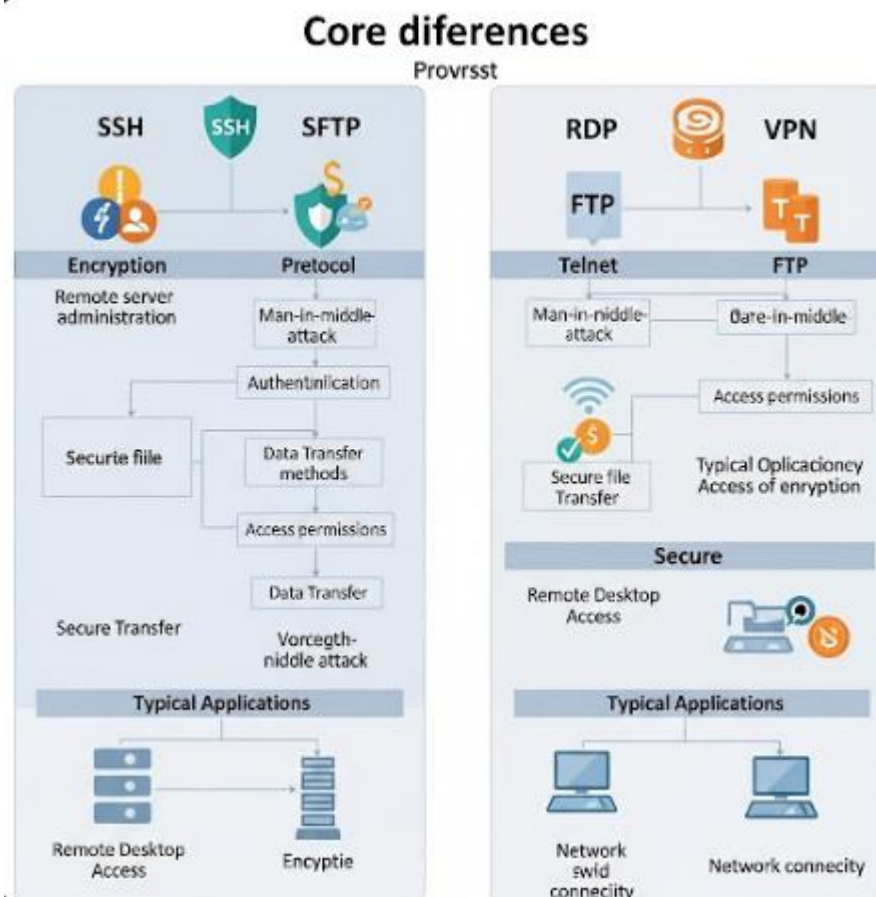


Рисунок 1.2 – Відмінності між SSH, SFTP, Telnet, FTP, RDP та VPN [9]

У сукупності своїх можливостей, SSH на сьогоднішній день визнаний де-факто стандартом у галузі безпечного віддаленого адміністрування та доступу до систем, значно випереджаючи за своїми можливостями та рівнем безпеки більшість розглянутих альтернативних рішень.

1.2 Android як платформа для системного адміністрування.

Операційна система Android, яка традиційно асоціюється переважно з мобільними телефонами, споживчими додатками та соціальними мережами, насправді володіє значним потенціалом для вирішення задач системного адміністрування. Оскільки Android базується на ядрі Linux, він успадковує багато фундаментальних можливостей та інструментів, що притаманні повноцінним Linux-системам, хоча й з певними архітектурними та безпековими обмеженнями. Для системних адміністраторів, яким необхідний оперативний доступ до серверної інфраструктури в будь-який момент, Android може слугувати надзвичайно зручним

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

та ефективним інструментом. У випадках, коли виникає потреба швидко підключитися до віддаленого сервера через протокол SSH, провести перевірку системних журналів (логів), перезапустити критично важливу службу, або ж перевірити обсяг вільного дискового простору – усі ці дії можуть бути виконані безпосередньо з мобільного пристрою, навіть за відсутності доступу до повноцінного комп'ютера.

На платформі Android існує низка бібліотек та методів для реалізації функціоналу віддаленого доступу до серверів. У рамках даної роботи для встановлення захищеного з'єднання з Linux-сервером використовувалася бібліотека **SSHJ** [10]. Ця бібліотека, призначена для середовища Java (що є основою розробки під Android), надає весь необхідний інструментарій для ініціації, підтримки та управління захищеними SSH-сесіями. Для підвищення зручності використання та уникнення постійного ручного введення аутентифікаційних даних (IP-адреси, логіни, паролі або шляхи до ключів) їхнє безпечне зберігання може бути реалізовано за допомогою механізму **SharedPreferences** або інших більш захищених сховищ даних Android (рис.1.3).

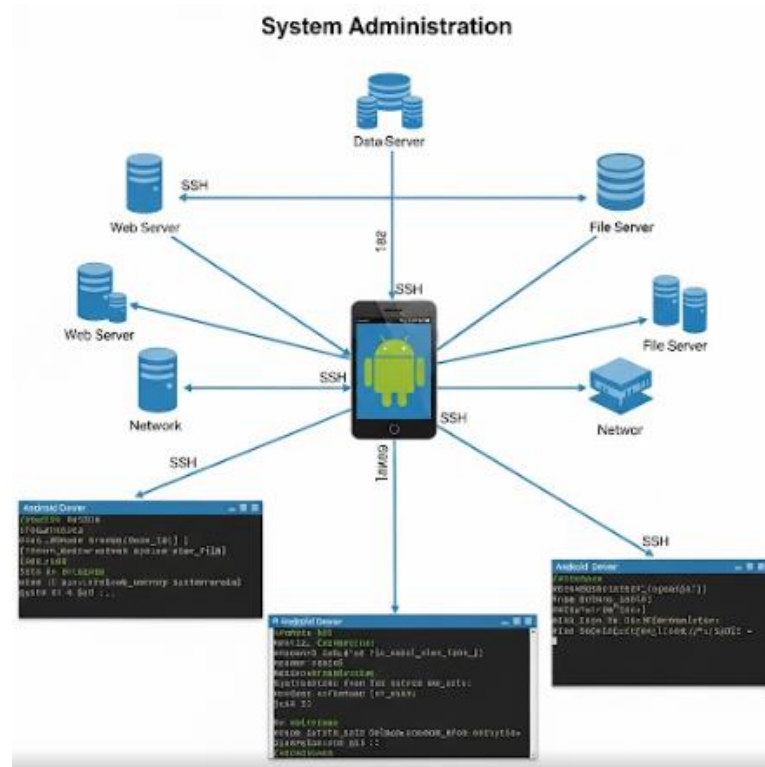


Рисунок 1.3 – Схема застосування Android-пристрою для віддаленого системного адміністрування через SSH

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

Безумовно, варто зазначити, що Android-пристрій не може забезпечити такий самий рівень глибокого контролю та гнучкості, як повноцінна операційна система на ноутбуці чи сервері. Наприклад, виконання багатьох низькорівневих операцій на системному рівні, які вимагають модифікації ядра або доступу до чутливих системних областей, може бути неможливим без отримання root-доступу, що часто не рекомендовано з міркувань безпеки. Однак, для виконання базових команд (таких як ls, mkdir, rm, cat, grep), запуску простих скриптів та перегляду структури файлової системи, функціоналу Android цілком достатньо [11].

Таким чином, операційна система Android може бути успішно використана як мобільна платформа для виконання завдань системного адміністрування, особливо коли необхідний швидкий доступ, оперативний моніторинг або відсутня можливість використовувати ноутбук. Крім того, гнучкість платформи дозволяє розробляти кастомізовані додатки під конкретні потреби адміністратора, що і було продемонстровано у даній кваліфікаційній роботі.

Для забезпечення зручності в підтримці, розширенні та масштабуванні Android-додатку критично важливим є його побудова на основі продуманої архітектури. Серед сучасних архітектурних шаблонів найбільш поширеним та рекомендованим для Android-розробки є MVVM (Model-View-ViewModel) [12]. Цей шаблон, застосований у даному проєкті, забезпечує чітке розділення відповідальності між логікою, даними та користувацьким інтерфейсом, що сприяє підвищенню якості коду та полегшує його модифікацію.

Основні компоненти архітектури MVVM у контексті Android-додатку:

1. **View (Вид).** Цей компонент відповідає за відображення користувацького інтерфейсу та взаємодію з користувачем. У контексті Android-розробки до View належать такі елементи, як **Activity** та **Fragment**. Вони отримують дані від ViewModel та відображають їх, а також передають дії користувача (наприклад, натискання кнопок, введення тексту) до ViewModel для обробки. У даному додатку реалізовано окремі екрани (View) для підключення до сервера та для перегляду файлів, що забезпечує логічне розмежування функціоналу.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

2. **ViewModel (Модель-Вид).** ViewModel слугує мостом між View та Model, містить у собі бізнес-логіку та логіку представлення даних, необхідних для відображення у View. Вона отримує запити від View, обробляє їх, взаємодіє з Model для отримання або збереження даних, а потім надає дані у форматі, зручному для відображення View. Наприклад, коли користувач ініціює підключення до сервера, саме ViewModel відповідає за валідацію введених даних та ініціацію запиту до Model. ViewModel також відповідає за збереження стану даних під час зміни конфігурації (наприклад, повороту екрану), щоб View могла відновити своє відображення без втрати інформації [13].

3. **Model (Модель).** Компонент Model інкапсулює логіку роботи з даними та джерелами даних. Це включає взаємодію з віддаленими серверами (наприклад, встановлення SSH-з'єднання, виконання команд), роботу з базами даних, мережеві операції, а також обробку помилок, пов'язаних з цими процесами. У цьому проєкті Model відповідає за всі аспекти, пов'язані з SSH-з'єднанням, виконанням команд на віддаленому сервері та обробкою відповідей.

Крім основних можливостей платформи Android, до переваг також належить підтримка фонові обробки даних, яка дозволяє виконувати мережеві або ресурсомісткі операції без блокування головного потоку інтерфейсу користувача. У межах цієї кваліфікаційної роботи фонову логіку реалізовано з використанням ExecutorService, що забезпечує асинхронне виконання дій, зокрема підключення до сервера через SSH. Це дозволяє уникнути "зависань" або некоректної роботи інтерфейсу при встановленні з'єднання.

Ще однією важливою особливістю Android є можливість зберігання стану та даних користувача між сесіями. У реалізованому застосунку для збереження параметрів підключення (IP-адреса, логін, пароль, порт) використано механізм SharedPreferences, який надає просту та зручну реалізацію зберігання у форматі ключ–значення. Це значно покращує досвід користувача, оскільки немає потреби повторно вводити дані при кожному запуску програми.

Поєднання архітектурного патерну MVVM, асинхронної обробки даних та локального збереження параметрів забезпечує гнучкість та масштабованість

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

програмного рішення. Такий підхід дає змогу в майбутньому розширити функціональність застосунку — наприклад, додати підтримку авторизації за допомогою SSH-ключів або керування кількома серверами — без суттєвої перебудови архітектури та без втрати стабільності застосунку.

1.3. Технічне завдання та функціональні вимоги до мобільного файлового менеджера

Підстава для розробки проєкту: Підставою для розробки даного програмного забезпечення стало завдання кваліфікаційної роботи бакалавра, що передбачає створення мобільного інструменту для базового адміністрування Linux-серверів через протокол SSH. Ідея базується на потребі у зручному доступі до серверної файлової системи за допомогою Android-пристрою без використання терміналу.

Призначення розробки проєкту: Мобільний застосунок призначений для адміністраторів, розробників або ІТ-фахівців, які мають потребу у дистанційному доступі до файлової системи Linux-серверів. Застосунок дозволяє підключатися через SSH, здійснювати навігацію по каталогах, виконувати базові дії з файлами (створення, видалення, копіювання, перейменування, передача), керувати декількома серверами, зберігати дані користувачів у локальній базі.

Вимоги до проєкту:
Застосунок повинен реалізовувати наступні функції:

1. Здійснювати підключення до віддаленого Linux-сервера через SSH-протокол.
2. Встановлювати SFTP-з'єднання для файлових операцій.
3. Забезпечувати перегляд вмісту директорій у вигляді списку з іконками.
4. Реалізовувати функціонал:
 - створення нової папки;
 - копіювання, перейменування, видалення файлів і папок;
 - завантаження файлів на сервер (upload);
 - завантаження файлів із сервера (download).

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

5. Зберігати IP, логін, пароль, порт для кожного сервера окремо у базі даних Room.
6. Надавати можливість вибору збережених серверів та їх редагування.
7. Підтримувати очікувані дії (чергу операцій), що дозволяє користувачу спочатку обрати операцію, а потім – директорію призначення.
8. Виводити статус дій (успішно/невдало) через повідомлення та інтерфейсні реакції.
9. Виконувати мережеві дії асинхронно, не блокуючи інтерфейс (через ExecutorService).
10. Забезпечувати адаптивний інтерфейс з урахуванням принципів UX.

Мінімальні системні вимоги:

На мобільному пристрої:

- Операційна система: Android 8.0 або вище
- Процесор: ARM, 4 ядра (Snapdragon/MediaTek)
- ОЗП: 2 ГБ і більше
- Пам'ять: не менше 50 МБ вільного місця

На сервері:

- ОС: Ubuntu 20.04/22.04, Debian 11/12, CentOS 7/8
- Працюючий OpenSSH-сервер
- Доступ по локальній мережі або VPN/публічний IP

Вимоги до інтерфейсу:

- Мінімум два основних екрани: підключення (ConnectionActivity) та файловий менеджер (FilesActivity).
- Додатковий екран керування користувачами (UsersActivity).
- Підтримка плаваючих кнопок для виклику дій.
- Екран має відображати шлях поточного каталогу, статус з'єднання та дії.
- Інтерфейс повинен бути зручним, чистим та інтуїтивно зрозумілим.
- Реалізація всіх елементів із дотриманням гайдлайнів Material Design.

Вимоги до інформаційної та програмної сумісності:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

- Застосунок повинен підтримуватись на більшості сучасних Android-пристроїв.
- Сумісність із серверними ОС: Ubuntu, Debian, CentOS.
- Підтримка нестандартного порту SSH.
- Сумісність із бібліотекою SSHJ (для SSH/SFTP-з'єднання).
- Зберігання даних у Room (внутрішня SQLite-база).

Висновок першого розділу

У першому розділі було розглянуто ключові теоретичні засади, необхідні для розробки мобільного застосунку адміністрування серверів на базі Android. Проведено детальний аналіз принципів роботи протоколу SSH, його архітектури, переваг безпечного з'єднання та можливостей взаємодії з файловою системою через підпротокол SFTP. Особлива увага приділена перевагам SSH у порівнянні з іншими протоколами, а також його ролі у сучасному IT-середовищі.

Розглянуто платформу Android як середовище для створення мобільних інструментів адміністрування. Незважаючи на обмеження мобільного форм-фактора, Android надає широкий набір інструментів — як для роботи з мережею, так і для зберігання стану, виконання фонових задач та створення інтерфейсів. Особливо підкреслено ефективність використання архітектури MVVM, ViewModel-компонентів, локального зберігання через SharedPreferences та фонові обробки за допомогою ExecutorService.

Крім того, у розділі було сформульовано технічне завдання, у якому визначено цілі, функціональні та нефункціональні вимоги до застосунку, а також окреслено середовище його використання. Уточнено вимоги до сумісності, інтерфейсу та програмної архітектури, що дозволяє забезпечити стабільність, масштабованість та подальший розвиток проєкту.

Отримані теоретичні положення, а також сформульоване технічне завдання стали основою для реалізації програмного рішення, описаного у другому розділі роботи.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Структура застосунку: архітектура, активності, ViewModel, класи з'єднання

Розроблений проєкт реалізовано у вигляді Android-додатку з використанням мови програмування Java, з дотриманням архітектурного шаблону **MVVM (Model-View-ViewModel)**. Цей підхід дозволяє чітко розділити відповідальність між рівнями інтерфейсу користувача (UI), бізнес-логіки та обробки даних, що значно спрощує процес розробки, полегшує налагодження та подальшу підтримку програмного продукту, а також забезпечує його масштабованість для майбутніх доопрацювань [14].

Загальна архітектура додатку умовно поділяється на чотири основні рівні:

1. Інтерфейс користувача (UI): компоненти, відповідальні за візуальне представлення даних та взаємодію з користувачем.
2. ViewModel: прошарок, що містить бізнес-логіку та слугує посередником між UI та моделями даних.
3. Класи з'єднання та логіки: компоненти, що інкапсулюють взаємодію з віддаленими серверами та базові операції.
4. Система збереження даних: механізми для персистентного зберігання даних на пристрої.

Детальний опис кожного рівня і його компонентів:

1. Інтерфейс користувача (View)

Цей рівень включає всі візуальні елементи, з якими взаємодіє користувач, та компоненти, що відповідають за їх відображення:

- ConnectionActivity – є початковим екраном додатку. Він надає інтерфейс для введення параметрів підключення до віддаленого сервера (IP-адреса, порт, логін, пароль) та ініціює передачу цих даних до відповідного ViewModel для встановлення з'єднання.
- FilesActivity – представляє основний інтерфейс файлового менеджера, який стає доступним після успішного підключення до сервера. Він відповідає за відображення вмісту віддалених директорій та надання

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

функціоналу для виконання файлових операцій (видалення, копіювання, створення тощо).

- UsersActivity – екран, призначений для керування збереженими обліковими записами користувачів/серверів. Дозволяє обирати, редагувати або видаляти параметри підключення до серверів, до яких раніше здійснювалося підключення.

2. ViewModel

ViewModel-и є центральними елементами архітектури MVVM, що містять бізнес-логіку та управляють станом даних для UI:

- ConnectionViewModel – відповідає за опрацювання логіки авторизації користувача та взаємодію з SSHManager для встановлення з'єднання. Він транслює результати процесу підключення у вигляді подій (ConnectionScreenEvent), які потім можуть бути оброблені UI.
- FilesViewModel – управляє всіма аспектами файлового менеджера, включаючи завантаження вмісту директорій, ініціацію операцій копіювання, видалення, створення папок та інше. Взаємодія з віддаленою файловою системою здійснюється через RemoteFileRepository
- UsersViewModel – забезпечує взаємодію з локальною базою даних Room, що дозволяє керувати списком збережених серверів та облікових записів (через RemoteUserDao та RemoteUserRepository).

3. Класи з'єднання та логіки (Model)

Цей рівень містить ядро функціоналу, що забезпечує взаємодію з віддаленими системами та обробку даних:

- SSHManager – є ключовим класом для управління SSH-з'єднанням. Він відповідає за встановлення та підтримку захищеного каналу, механізми автентифікації, виконання довільних команд на віддаленому сервері та ініціалізацію SFTP-клієнта.
- RemoteFileRepository – реалізує основні файлові операції на віддаленому сервері: перегляд вмісту директорій, копіювання, перейменування, завантаження (upload) та вивантаження (download) файлів, а також їх

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

видалення. Для деяких специфічних дій використовує виконання Linux-команд через SSH.

- InternalFileRepository – компонент, призначений для взаємодії з файловою системою самого Android-пристрою, зокрема для тимчасового зберігання файлів під час передачі або для очищення тимчасових даних.
- RemoteFileProvider, RemoteFileDescriptorCallback – реалізують механізм доступу до віддалених файлів через стандартну систему Android "поділитися/відкрити", що дозволяє інтегрувати файлові операції з іншими додатками.

4. Збереження та обробка даних

Цей рівень відповідає за персистентне зберігання даних та управління станом операцій:

- RemoteUser – модель даних, що описує структуру облікового запису користувача або сервера, включаючи IP-адресу, логін та пароль.
- AppDatabase, RemoteUserDao, RemoteUserRepository – компоненти бібліотеки Room, що надають функціонал для локального зберігання облікових записів серверів у базі даних на пристрої.
- RemoteUserPreferences – використовується для тимчасового або більш легкого зберігання налаштувань, таких як інформація про активного користувача між сесіями.
- PendingOperationData, PendingOperationType – структури даних, що використовуються для реалізації черги очікуваних операцій (наприклад, вибір цільового каталогу перед завершенням операції копіювання).

5. Допоміжні компоненти

Додаткові класи та утиліти, що забезпечують функціональність UI та обробку подій:

- FilesAdapter – адаптер для компонента RecyclerView, який відповідає за ефективне відображення списку файлів у файловому менеджері, включаючи відображення відповідних іконок, розмірів та дат модифікації.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

- FileActionsBottomSheet – реалізує контекстне меню у вигляді "нижнього листа" (bottom sheet) з основними діями, доступними для файлів (завантажити, перейменувати, перемістити, видалити тощо).
- ThrowableAction, OnSuccess, OnFailure – допоміжні інтерфейси та обгортки для стандартизованої обробки результатів асинхронних операцій, що можуть завершитися успішно або з помилкою

Таким чином, структура розробленого додатку побудована за модульним принципом з чітким розмежуванням відповідальностей між окремими компонентами. Це забезпечує високу зручність підтримки коду, гнучкість у розширенні функціональності (наприклад, додавання підтримки авторизації по ключу або кількох одночасних SSH-сесій) та легкість адаптації до нових вимог. Така архітектура сприяє надійній роботі та забезпечує стабільний досвід користувача.

2.1.2 Екран авторизації: збереження доменного імені, логіна, пароля та порту

Початковим етапом взаємодії користувача з мобільним застосунком є екран авторизації, що реалізований в активності ConnectionActivity. Цей інтерфейс призначений для введення ключових параметрів, необхідних для встановлення захищеного з'єднання з віддаленим Linux-сервером: логіна, пароля, доменного імені сервера (IP-адреси або доменного імені) та порту.

На рисунку 2.1 зображено схему підключення:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

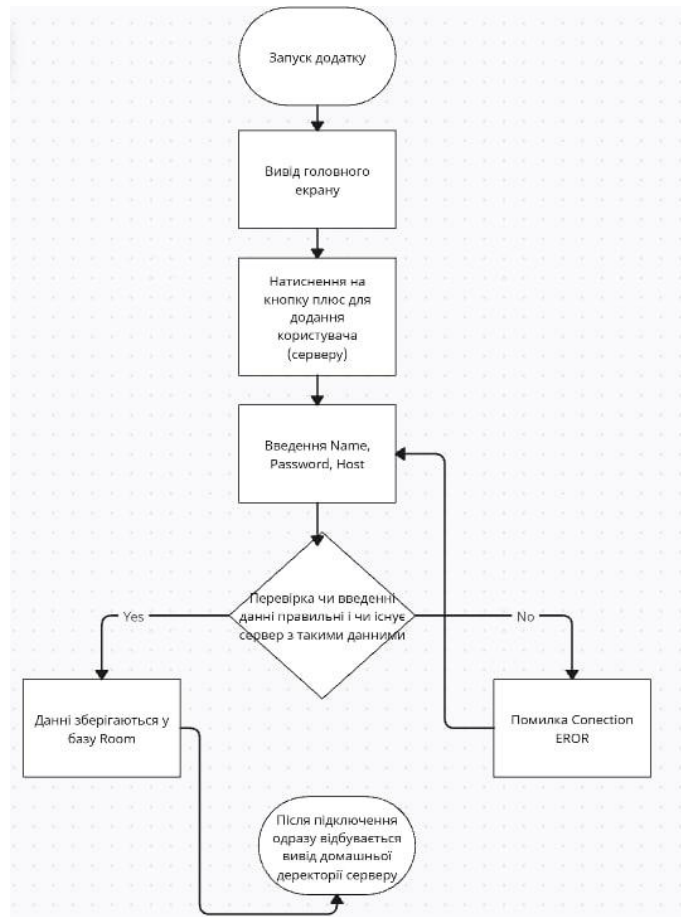


Рисунок 2.1 – Схема авторизації та підключення до серверу

Однією з ключових особливостей поточної реалізації, є впровадження підтримки доменних імен (наприклад, ubuntu.local). Ця функціональність досягається завдяки використанню Avahi daemon на стороні сервера, що забезпечує підтримку протоколу mDNS (multicast DNS) для локального іменування пристроїв [15]. Такий підхід усуває потребу ручного введення IP-адреси, спрощуючи процес авторизації та підвищуючи зручність використання застосунку.

Збереження облікових записів за допомогою Room

Для персистентного зберігання параметрів підключення та забезпечення зручності повторного доступу до серверів використовується внутрішня база даних Android, імплементована за допомогою бібліотеки **Room**. Дані зберігаються у вигляді об'єктів-сутностей RemoteUser, що інкапсулюють логін, пароль, домен (або IP-адресу) та унікальний ідентифікатор (ID).

Цей механізм надає наступні переваги:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

- Запобігання дублюванню: система забезпечує унікальність збережених облікових записів.
- Швидкий доступ: користувачі можуть швидко обирати та активувати попередні підключення без необхідності повторного введення даних.
- Актуалізація даних: можливість оновлювати існуючі облікові записи у випадку зміни параметрів доступу.

Структура сутності RemoteUser для Room визначена наступним чином:

```
Entity(tableName = RemoteUser.TABLE_NAME)
public class RemoteUser {
    public static final String TABLE_NAME = "remote_users";
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    private final long id;
    @ColumnInfo(name = "username")
    private final String username;
    @ColumnInfo(name = "password")
    private final String password;
    @ColumnInfo(name = "host")
    private final String host;

    @Ignore
    public RemoteUser(String username, String password, String host) {
        this(0, username, password, host);
    }

    public RemoteUser(long id, String username, String password, String host) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.host = host;
    }
}
```

Логіка обробки підключення у ViewModel

Бізнес-логіка, пов'язана з обробкою підключення, інкапсульована у класі ConnectionViewModel. Коли користувач ініціює з'єднання натисканням кнопки "Підключитися", відбувається валідація введених даних. У випадку успішної валідації полів створюється або оновлюється об'єкт RemoteUser, і застосунок намагається встановити захищене з'єднання через SSHManager в асинхронному режимі. Це запобігає блокуванню основного потоку інтерфейсу користувача.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Приклад реалізації методу onConnectButtonClick:

```
public void onConnectButtonClick(String username, String password, String host) {
    if (username.isEmpty() || password.isEmpty() || host.isEmpty()) {
        // Оновлення стану LiveData для відображення помилки на UI
        stateLiveData.setValue(ConnectionScreenState.event(new
ShowErrorMessage("Усі поля є обов'язковими")));
    } else {
        // Створення або оновлення об'єкта RemoteUser
        RemoteUser user = (savedUser != null)
            ? new RemoteUser(savedUser.getId(), username, password, host)
            : new RemoteUser(username, password, host);

        // Зміна стану LiveData на "Підключення..."
        stateLiveData.setValue(ConnectionScreenState.connecting());

        // Асинхронне виконання операції підключення
        async(
            () -> sshManager.connect(user), // Виконання в фоновому потоці
            () -> onConnectionSuccess(user), // Колбек у разі успіху
            this::onConnectionError         // Колбек у разі помилки
        );
    }
}
```

Автоматичне завантаження попереднього користувача

Для підвищення зручності користувача, після запуску активності ConnectionActivity, застосунок автоматично завантажує параметри останнього збереженого користувача. Ця функціональність реалізована методом onCreate() у ConnectionViewModel, де виконується асинхронний запит до RemoteUserRepository для отримання поточного користувача:

```
public void onCreate() {
    if (shouldShowSavedUser) {
        shouldShowSavedUser = false;
        async(
            remoteUserRepository::getCurrentUser,
            this::onCurrentUserLoaded,
            ___ -> { /*do nothing*/ }
        );
    }
}
```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Метод `onCurrentUserLoaded` відповідає за заповнення полів введення на екрані отриманими даними.

Стан та події інтерфейсу (LiveData)

Для ефективної синхронізації стану UI з логікою застосунку використовується клас `ConnectionScreenState` у поєднанні з `LiveData`. Цей клас визначає поточний стан екрану (наприклад, стан підключення, необхідність відображення повідомлень) та обробляє події, що виникають під час взаємодії (такі як помилка підключення або перехід до наступного екрану, `FilesActivity`). Використання `LiveData` забезпечує реактивність інтерфейсу та автоматичне оновлення UI при зміні стану без необхідності ручного управління життєвим циклом компонентів.

Таким чином, екран авторизації реалізовано відповідно до сучасних архітектурних підходів Android-розробки. Застосування `Room` для надійного зберігання облікових записів, `LiveData` для реактивного відображення стану та підтримка доменних імен серверів через протокол `Avahi/mDNS` забезпечують гнучкість, безпеку та високу зручність користувача, мінімізуючи потребу повторного введення даних та прискорюючи доступ до віддалених ресурсів.

2.1.2 Екран перегляду файлів: завантаження директорій, адаптер списку

Після успішного встановлення SSH-з'єднання користувач потрапляє на головний екран додатку – `FilesActivity`, який реалізує функціонал мобільного файлового менеджера. Основна логіка взаємодії з віддаленою файловою системою та управління станом цього екрану розміщена у `FilesViewModel`, тоді як безпосередній доступ до файлових операцій забезпечується через `RemoteFileRepository`.

Завантаження вмісту директорій через SFTP

На рисунку 2.2 зображено схему завантаження списку файлів (директорій):

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

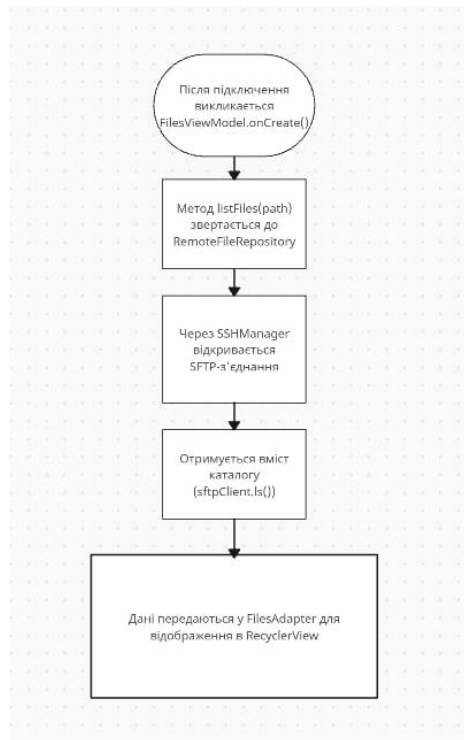


Рисунок 2.2 – Схема завантаження списку файлів

Механізм завантаження вмісту директорій реалізовано за допомогою методу `listFiles(path)` у класі `RemoteFileRepository`.

Цей метод виконує наступні кроки:

- Отримує активний SFTP-клієнт від `SSHManager`, який є точкою доступу до SSH-з'єднання.
- Визначає повний (канонічний) шлях до віддаленої директорії, використовуючи метод `canonicalize()` SFTP-клієнта для обробки відносних шляхів та символу `~` (домашня директорія користувача).
- Виконує команду `ls` (list directory) через SFTP-клієнта для отримання списку `RemoteResourceInfo` – об'єктів, що інкапсулюють інформацію про кожен файл або директорію.
- Відфільтровує службові записи (`.` та `..`) та сортує список за визначеним критерієм (наприклад, директорії спочатку, потім файли, за алфавітом).

Приклад реалізації методу `listFiles`:

```

public List<RemoteResourceInfo> listFiles(String path) throws Exception {
    RemoteUser user = getCurrentUser(); // Отримання інформації про поточного користувача
  
```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

```

SFTPClient sftpClient = sshManager.getSFTPClient(user); // Отримання SFTP-
клієнта
String fullRemotePath = getFullRemotePath(sftpClient, path); // Формування
повного шляху
return sftpClient.ls(fullRemotePath) // Виконання команди list
    .stream()
    .filter(file -> !file.getName().equals(".") && !file.getName().equals("..")) //
Фільтрація службових записів
    .sorted(RemoteFileComparator()) // Сортуння файлів та директорій
    .collect(Collectors.toList()); // Збір результатів у список
}

```

Цей метод викликається асинхронно з FilesViewModel кожного разу при переході користувача в іншу папку або при ініціації оновлення вмісту (наприклад, через жест onSwipeRefresh()), забезпечуючи актуальність відображуваних даних.

Навігація між папками

На рисунку 2.3 зображено схему навігації:

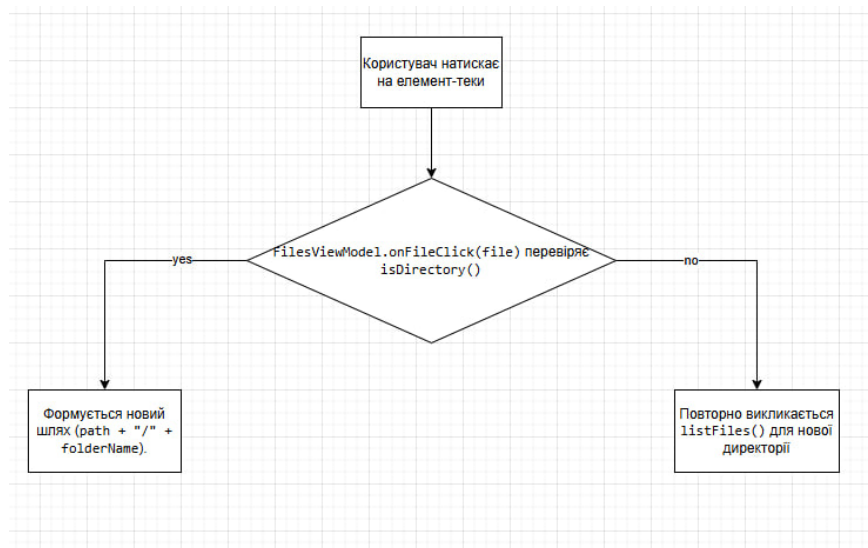


Рисунок 2.3 – Схема навігації між папками

Користувач може здійснювати навігацію по віддаленій файловій системі, натискаючи на назви директорій. Обробка цієї події відбувається у методі onFileClick() в FilesViewModel:

```

public void onFileClick(RemoteResourceInfo file) {
    if (file.isDirectory()) {
        // Якщо це директорія, формуємо новий шлях і завантажуюмо її вміст
        String path = new File(getCurrentState().getPath(), file.getName()).getPath();
        listFiles(path);
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

```

    } else {
        // Якщо це файл, ініціюємо подію відкриття файлу
        stateLiveData.setValue(getCurrentState().copyWithEvent(new OpenFile(file)));
    }
}

```

Також реалізовано функціонал повернення назад по ієрархії директорій (метод `onBackPressed()`), що дозволяє користувачеві підніматися на один рівень вище у файловій системі.

Адаптер `FilesAdapter`: форматування та обробка подій

Відображення списку файлів та директорій у `RecyclerView` здійснюється за допомогою `FilesAdapter`.

Цей адаптер відповідає за:

- Відображення іконок: призначення відповідних іконок для папок та різних типів файлів.
- Форматування інформації: відображення імені, дати останньої модифікації та, для файлів, їхнього розміру. Форматування розміру файлу здійснюється за допомогою `Formatter.formatFileSize()`, а дати — за допомогою `DateUtils.formatDateTime()` для забезпечення локалізованого та зручного для читання формату.

```

String size = Formatter.formatFileSize(context, item.getAttributes().getSize());
String      modificationDate      =      DateUtils.formatDateTime(context,
item.getAttributes().getMtime() * 1000, DateUtils.FORMAT_SHOW_DATE);
holder.binding.details.setText(item.isDirectory() ? modificationDate : String.format("%s
• %s", size, modificationDate));

```

- Обробка кліків: реєстрація кліків користувача по елементах списку для ініціації відповідних подій (наприклад, вхід у папку, відкриття контекстного меню для файлу). Кожен клік делегується відповідному методу у `FilesViewModel` для подальшої обробки логіки.

Додаткові функції файлового менеджера

Додаток підтримує широкий спектр файлових операцій:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

- `onCopyButtonClick()`, `onMoveButtonClick()`: зберігають тип майбутньої операції (копіювання/переміщення) та ініціюють діалог вибору цільової папки.
- `onFolderChosen()`: виконує безпосередньо операцію копіювання або переміщення після вибору цільової директорії.
- `onRenameFileConfirmed()`: змінює назву вибраного файлу або папки на віддаленому сервері.
- `onDeleteFileConfirmed()`: видаляє об'єкт (файл або директорію з її вмістом) на віддаленому сервері, використовуючи SSH-команду `rm -r`.
- `onCreateFolderConfirmed()`: створює нову директорію у поточній локації.
- `onUploadFilePicked()`: запускає процес передачі вибраного файлу з локального пристрою на віддалений сервер.
- `onDownloadButtonClick()`: завантажує вибраний віддалений файл у системну теку "Downloads" на Android-пристрої.

Обробка станів та подій у ViewModel

Для забезпечення реактивності та керованого відображення інтерфейсу, кожна взаємодія користувача та результат операції супроводжуються оновленням об'єкта `FilesScreenState`. Цей клас інкапсулює повний стан файлового менеджера, включаючи: поточний список файлів та директорій, шлях до поточної робочої директорії, статус завантаження даних (індикатор прогресу), одноразові події (наприклад, `ShowRenameFileDialog` для відображення діалогу перейменування, або `ShowErrorMessage` для виведення повідомлення про помилку). Цей підхід дозволяє чітко контролювати, які дії ініціюються, та як інтерфейс користувача має на них реагувати, забезпечуючи стабільну та передбачувану поведінку додатку.

Таким чином, екран перегляду файлів реалізує повноцінний SFTP-клієнт з розширеними можливостями навігації по директоріях, керування файлами та підтримкою всіх необхідних операцій – від копіювання до завантаження та видалення. Такий рівень деталізації та модульність архітектури забезпечує високу

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

масштабованість, стабільність роботи та готовність до подальшого розвитку в рамках дипломного проєкту.

2.1.3 Екран вибору користувача: база даних Room, RecyclerView

Для підвищення зручності користувача та оптимізації процесу взаємодії з віддаленими серверами, у застосунку реалізовано окремий екран, призначений для вибору та керування раніше збереженими підключеннями. Ця функціональність дозволяє користувачам швидко перемикатися між різними серверами без необхідності повторного введення аутентифікаційних даних для кожного з'єднання.

Персистентне зберігання облікових записів за допомогою Room

Усі облікові записи серверів персистентно зберігаються у локальній базі даних SQLite. Для взаємодії з цією базою даних використовується бібліотека Room Persistence Library, яка є абстракційним шаром над SQLite, що значно спрощує роботу з даними. Клас RemoteUser позначено як сутність @Entity для Room, що дозволяє відображати його об'єкти на таблицю в базі даних. Збереження, отримання та видалення даних виконується через інтерфейс RemoteUserDao (Data Access Object), який визначає абстрактні методи для роботи з базою даних, а Room автоматично генерує їхні реалізації під час компіляції.

Приклад визначення RemoteUserDao:

```
public interface RemoteUserDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    long save(RemoteUser user);

    @Query("SELECT * FROM " + RemoteUser.TABLE_NAME + " ORDER BY
    username ASC")
    List<RemoteUser> getAll(); // Метод для отримання всіх збережених користувачів

    @Query("SELECT * FROM " + RemoteUser.TABLE_NAME + " WHERE id = :id
    LIMIT 1")
    RemoteUser get(long id); // метод отримання ID

    @Delete
    void delete(RemoteUser user); // Метод для видалення користувача
```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

}

Такий підхід дозволяє ефективно керувати кількома профілями підключення, розглядаючи їх як записи у таблиці, а також забезпечує цілісність та консистентність даних.

Відображення списку користувачів за допомогою RecyclerView + UserAdapter

Екран керування користувачами реалізований за допомогою компонента RecyclerView, який оптимізований для відображення великих списків даних. Для візуалізації збережених облікових записів створено спеціалізований адаптер UsersAdapter. Цей адаптер відповідає за:

- Відображення інформації: кожен елемент списку відображає ключові дані облікового запису: логін (username), хост (host) та, за необхідності, порт.
- Обробка взаємодії: реалізована обробка натискань на окремі елементи списку (вибір користувача), а також наявність окремої кнопки для видалення облікового запису.

Приклад фрагментів коду з UsersAdapter:

@Override

```
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {  
    RemoteUser item = items.get(position);  
    holder.binding.username.setText(item.getUsername());  
    holder.binding.host.setText(item.getHost());  
    holder.binding.getRoot().setOnClickListener(__ ->  
viewModel.onUserClick(item));  
    holder.binding.deleteButton.setOnClickListener(__ ->  
viewModel.onDeleteButtonClick(item));  
}
```

Така реалізація забезпечує не лише простий перегляд збережених профілів, але й надає прямі засоби керування ними, включаючи їх видалення та вибір для подальшого підключення.

Взаємодія з ViewModel

На відміну від традиційного підходу, де адаптер міг би самостійно обробляти логіку взаємодії, у даному проєкті реалізовано принцип делегування. Адаптер

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

UsersAdapter не обробляє події (такі як вибір або видалення) безпосередньо, а викликає відповідні методи у UsersViewModel. Такий підхід забезпечує: дотримання архітектурного шаблону MVVM. Чітке розділення відповідальності, де View (адаптер) лише відображає дані та передає дії, а ViewModel містить логіку. Також чистота логіки ViewModel, тобто вся бізнес-логіка, пов'язана з управлінням користувачами та взаємодією з репозиторієм, зосереджена у ViewModel. І звичайно, легкість реагування на зміни в UI. ViewModel може ініціювати відповідні оновлення стану UI (наприклад, перехід до нового екрану) через LiveData, забезпечуючи реактивність застосунку

Таблиця 2.1 – Переваги реалізації екрану вибору користувача

Перевага	Опис
Збереження даних у Room	Використання бібліотеки Room забезпечує персистентне зберігання профілів підключення, завдяки чому дані користувачів не втрачаються після закриття або перезапуску додатку, підвищуючи надійність та зручність використання.
Реактивна логіка адаптера	Реалізація логіки обробки подій у ViewModel (через LiveData) та взаємодія з адаптером (RecyclerView) забезпечує реактивний зв'язок між інтерфейсом користувача та бізнес-логікою. Це дозволяє UI динамічно оновлюватися відповідно до змін стану, покращуючи користувацький досвід.

Продовження таблиці 2.1 – Переваги реалізації екрану вибору користувача

Можливість розширення	Модульна архітектура, побудована на принципах MVVM та чіткому розділенні відповідальностей, надає значну гнучкість для майбутнього розширення функціоналу. Це може включати додавання таких можливостей, як пошук, сортування,
-----------------------	--

	редагування профілів або інтеграція з авторизацією через SSH-ключі.
--	---

Отже, реалізація екрану вибору користувача є ключовою функцією для забезпечення зручності повторного підключення без повторного введення даних. Вона базується на сучасних підходах до архітектури Android-додатків, що робить застосунок гнучким, зручним та ефективним інструментом для системного адміністратора.

2.2 Розробка класів керування з'єднанням: SSHManager, RemoteUser

Для забезпечення стабільного та захищеного підключення до віддаленого сервера з мобільного додатку було розроблено два ключові компоненти, що відіграють центральну роль у системі: класи SSHManager та RemoteUser. Їхнє призначення полягає в інкапсуляції даних користувача, встановленні SSH-з'єднання, ініціалізації каналу передачі файлів за протоколом SFTP, виконанні команд та реалізації механізмів повторного з'єднання у випадку зміни активного користувача або тимчасової втрати зв'язку.

Клас RemoteUser є фундаментальною сутністю системи, реалізованою як повноцінна @Entity для бази даних Room. Цей клас призначений для персистентного зберігання облікових даних користувача, що включають його унікальний ідентифікатор (id), логін (username), пароль (password) та хост (host), який може бути представлений як IP-адресою, так і доменним ім'ям. Зокрема, підтримка доменних імен, таких як ubuntu.local, стала можливою завдяки використанню технології Avahi на серверній стороні, що надає функціонал multicast DNS (mDNS) для локального іменування пристроїв. Це значно спрощує процес підключення, усуваючи необхідність ручного введення IP-адреси.

@Entity(tableName = RemoteUser.TABLE_NAME)

```
public class RemoteUser {
    @PrimaryKey(autoGenerate = true)
    private final long id;
    private final String username;
    private final String password;
    private final String host;
```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

}

Клас SSHManager реалізовано за патерном Singleton, що гарантує існування лише одного його екземпляра протягом життєвого циклу застосунку. Його головним завданням є підтримка стабільного з'єднання з віддаленим сервером за допомогою бібліотеки SSHJ. Внутрішньо клас оперує такими ключовими полями, як sshClient для управління SSH-сесіями, sftpClient для роботи з файлами через SFTP, та currentUser, що зберігає об'єкт користувача, з яким поточне з'єднання встановлено.

Процес встановлення з'єднання інкапсульований у методі connect(RemoteUser user). Цей метод відповідає за ініціалізацію нового SSHClient, додавання PromiscuousVerifier для спрощеної (в контексті розробки) перевірки ключів хоста, встановлення з'єднання з вказаним хостом та стандартним портом SSH (22), здійснення автентифікації за логіном та паролем, а також створення SFTPClient для подальшої взаємодії з файловою системою. Після успішного встановлення з'єднання, currentUser оновлюється.

```
public void connect(RemoteUser user) throws IOException {  
    Security.removeProvider(BouncyCastleProvider.PROVIDER_NAME);  
    Security.insertProviderAt(new BouncyCastleProvider(), 1);  
  
    SSHClient sshClient = new SSHClient();  
    sshClient.addHostKeyVerifier(new PromiscuousVerifier());  
    sshClient.getConnection().getKeepAlive().setKeepAliveInterval(5);  
    sshClient.setConnectTimeout(5000);  
    sshClient.connect(user.getHost(), DEFAULT_PORT);  
    sshClient.authPassword(user.getUsername(), user.getPassword());  
}
```

Доступ до SFTPClient надається методом getSFTPClient(RemoteUser user). Перед тим, як повернути об'єкт SFTPClient, виконується перевірка connectIfRequired(user). Ця функція є критично важливою для забезпечення стійкої роботи системи, оскільки вона гарантує, що SFTP-операції завжди виконуються через активне та коректно встановлене з'єднання, навіть при зміні поточного користувача.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Можливість виконання довільних команд на віддаленому сервері реалізована методом `executeCommand(RemoteUser user, String command)`. Цей метод також передувє виконанню команди перевіркою `connectIfRequired(user)`. Після успішного встановлення або перепідключення сесії, команда передається на виконання, а її завершення очікується протягом заданого таймауту. Цей функціонал є базовим для широкого спектра операцій, таких як копіювання, видалення файлів чи створення директорій.

За стабільність та коректне функціонування з'єднання відповідає приватний метод `connectIfRequired(RemoteUser user)`. Він реалізує механізм повторного з'єднання, аналізуючи поточний стан `sshClient`. Якщо `sshClient` не ініціалізований або з'єднання відсутнє, метод ініціює повне встановлення з'єднання з параметрами вказаного користувача. У випадку, якщо з'єднання вже встановлено, але з іншим обліковим записом (`!user.equals(currentUser)`), поточне з'єднання спочатку коректно розривається, а потім встановлюється нове з'єднання з параметрами нового користувача.

```
private void connectIfRequired(RemoteUser user) throws IOException {
    if (sshClient == null || !sshClient.isConnected()) {
        connect(user);
    } else if (!user.equals(currentUser)) {
        disconnect();
        connect(user);
    }
}
```

Реалізована структура забезпечує низку значних переваг. По-перше, досягається повна ізоляція логіки з'єднання: клас `SSHManager` функціонує незалежно від компонентів інтерфейсу користувача, що значно підвищує модульність коду та спрощує його тестування. По-друге, використання патерну `Singleton` для `SSHManager` гарантує безпечне та централізоване управління SSH-сесіями. По-третє, архітектура є гнучкою та масштабованою, дозволяючи легко інтегрувати додаткові функції, такі як аутентифікація за допомогою SSH-ключів або розширене логування сесій. Нарешті, сумісність класу `RemoteUser` з

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

бібліотекою Room забезпечує надійне та ефективне зберігання облікових записів у локальній базі даних. Завдяки цій ретельно продуманій структурі, мобільний додаток отримує потужний та гнучкий механізм для взаємодії з віддаленими серверами, що забезпечує високий рівень зручності використання та є солідною основою для подальшого розвитку функціоналу у дипломному проєкті.

2.3 Реалізація функціоналу файлового менеджера

2.3.1 Основні дії з файлами та папками: створення, видалення, перейменування, копіювання, переміщення

Після успішного підключення до віддаленого сервера користувач отримує можливість виконувати базові операції з файлами та папками, що є ключовим функціоналом мобільного файлового менеджера. До цих операцій належать створення нової папки, перейменування існуючих об'єктів, копіювання, переміщення та видалення файлів і директорій. Усі ці дії реалізовані відповідно до архітектурного шаблону MVVM (Model-View-ViewModel): логіка взаємодії інкапсульована у FilesViewModel, тоді як безпосередня робота з віддаленим сервером здійснюється через RemoteFileRepository. Інтерфейс користувача реагує на події, що генеруються ViewModel, використовуючи механізми LiveData та спеціалізовані Event-класи.

Створення папки.

На рисунку 2.4 схема створення папки:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

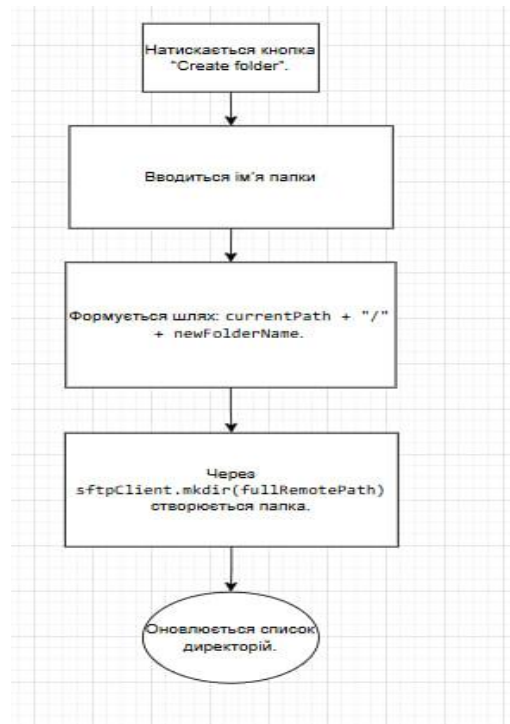


Рисунок 2.4 – Схема створення нової папки

Процес створення нової папки ініціюється після натискання відповідної кнопки в інтерфейсі, що викликає діалогове вікно для введення бажаної назви. Після підтвердження користувачем, ця подія обробляється методом onCreateFolderConfirmed() у FilesViewModel, який, у свою чергу, делегує виклик методу createFolder() з RemoteFileRepository. На рівні репозиторію відбувається отримання SFTP-клієнта від SSHManager, формування повного віддаленого шляху до нової папки, і виконання SFTP-команди mkdir для її створення. Папка створюється у поточній директорії, і після успішного завершення операції користувач бачить оновлений список вмісту директорії.

```

public void createFolder(String path) throws Exception {
    RemoteUser user = getCurrentUser();
    SFTPClient sftpClient = sshManager.getSFTPClient(user);
    String fullRemotePath = getFullRemotePath(sftpClient, path);
    sftpClient.mkdir(fullRemotePath);
}
  
```

Видалення файлів або папок.

На рисунку 2.5 зображено схему видалення файлів і директорій:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

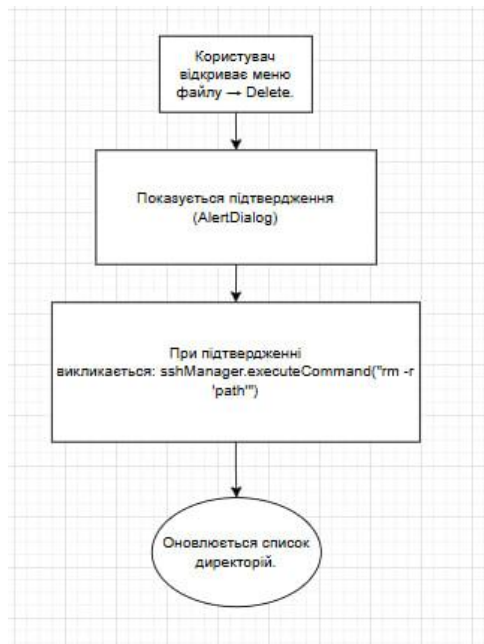


Рисунок 2.5 – Схема видалення файлів або директорій

Для видалення файлів або директорій у додатку передбачено окрему кнопку. Її натискання ініціює показ діалогового вікна підтвердження (ShowDeleteFileDialog) для запобігання випадковому видаленню. Після отримання підтвердження від користувача, FilesViewModel викликає метод onDeleteFileConfirmed(), який делегує виконання операції методу deleteFile() у RemoteFileRepository. Видалення на віддаленому сервері виконується шляхом відправлення SSH-команди `rm -r`. Ця команда ефективно обробляє як окремі файли, так і директорії з усім їхнім вмістом, забезпечуючи гнучкість операції.

```

public void deleteFile(RemoteResourceInfo file) throws Exception
{
    RemoteUser user = getCurrentUser(); // Використання rm -r для рекурсивного
    видалення файлів та папок
    sshManager.executeCommand(user, "rm -r " + file.getPath() + "");
}
  
```

Перейменування.

На рисунку 2.6 зображено схему перейменування файлів і директорій:

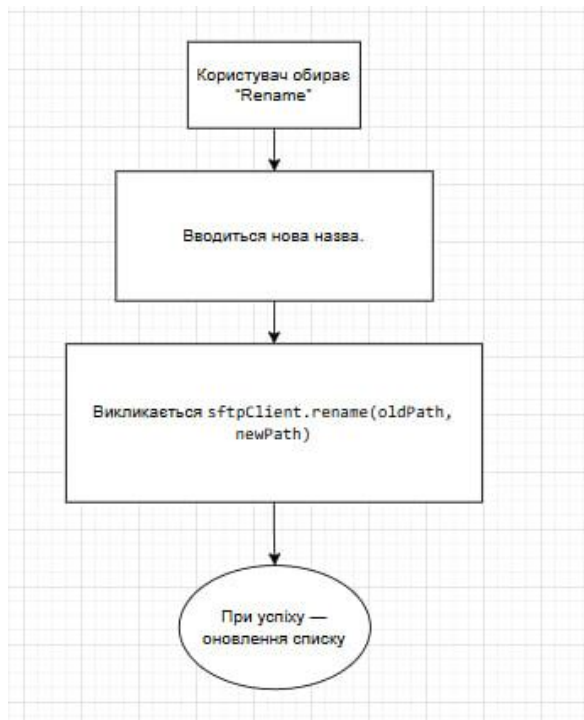


Рисунок 2.6 схема переіменування файлів і директорій:

Операція перейменування ініціюється викликом методу `onRenameButtonClick()`, що призводить до появи діалогового вікна, де користувач може ввести нове ім'я для файлу або папки. Після підтвердження, логіка перейменування обробляється у `RemoteFileRepository` за допомогою методу `renameFile()`. Цей метод отримує SFTP-клієнт, формує повні шляхи для старого та нового імен, а потім використовує SFTP-команду `rename` для виконання дії. Ця операція є безпечною, і у випадку успішного завершення список файлів у поточному каталозі автоматично оновлюється.

```

public void renameFile(String oldPath, String newPath) throws Exception {
    RemoteUser user = getCurrentUser();
    SFTPClient sftpClient = sshManager.getSFTPClient(user);
    String fullOldPath = getFullRemotePath(sftpClient, oldPath);
    String fullNewPath = getFullRemotePath(sftpClient, newPath);
    sftpClient.rename(fullOldPath, fullNewPath);
}
  
```

Копіювання та переміщення.

На рисунках 2.7-2.8 зображені схеми копіювання і переміщення файлів:

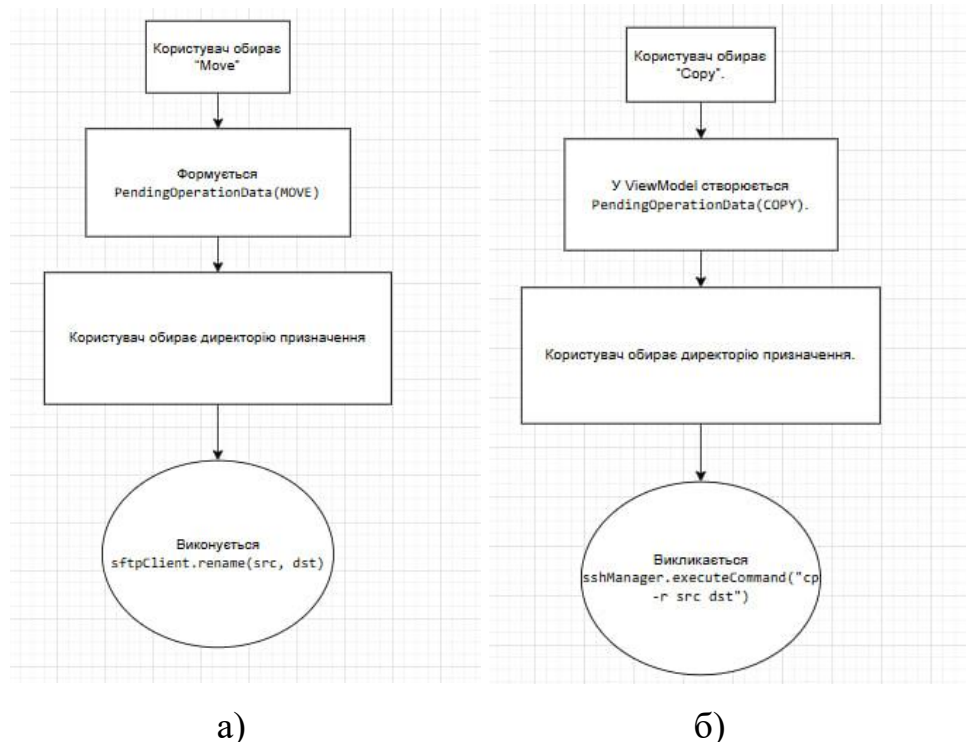


Рисунок 2.7 – Схема копіювання і переміщення файлів
а – схема копіювання; б – схема переміщення

Для реалізації функцій копіювання та переміщення файлів використовується подвійна логіка взаємодії. Спершу користувач натискає відповідну кнопку (наприклад, "Копіювати" або "Перемістити"). Після цього, застосунок переходить у режим вибору цільової папки, куди буде скопійовано або переміщено об'єкт. Коли користувач обирає цільову директорію, FilesViewModel викликає відповідний метод — copyFile() або moveFile() у RemoteFileRepository.

Операція копіювання (copyFile()) виконується шляхом формування та відправлення SSH-команди `cp -r` на віддалений сервер. Ця команда дозволяє рекурсивно копіювати як файли, так і директорії з усім їхнім вмістом до вказаного місця призначення

```

public void copyFile(RemoteResourceInfo file, String destinationPath) throws
Exception
    RemoteUser      user      =      getCurrentUser();
    SFTPClient      sftpClient =      sshManager.getSFTPClient(user);
    String fullDestinationPath = getFullRemotePath(sftpClient, destinationPath);
    String command = "cp -r " + file.getPath() + " " + fullDestinationPath + "/" +

```

```

file.getName()
sshManager.executeCommand(user,
}

```

Операція переміщення (moveFile()) реалізована як аналог перейменування, але з використанням нового шляху призначення. SFTP-клієнт використовує команду rename для зміни місця розташування файлу або папки.

```

public void moveFile(RemoteResourceInfo file, String destinationPath) throws
Exception
{
    RemoteUser user = getCurrentUser();
    SFTPCClient sftpClient = sshManager.getSFTPCClient(user);
    String fullDestinationPath = getFullRemotePath(sftpClient, destinationPath);
    sftpClient.rename(file.getPath(), fullDestinationPath + "/" + file.getName());
}

```

Важливою особливістю реалізації всіх цих дій є їхнє асинхронне виконання, що запобігає блокуванню головного потоку інтерфейсу користувача та забезпечує плавність роботи застосунку. Після завершення кожної операції, користувач отримує зворотний зв'язок у вигляді оновленого списку файлів або повідомлення про помилку, якщо така виникла. Завдяки такому підходу, забезпечується стабільність, зручність та логічна послідовність взаємодії з файловою системою віддаленого сервера.

2.3.2 Реалізація механізмів передачі файлів: завантаження та вивантаження

Ключовим елементом функціональності розробленого файлового менеджера є можливість обміну даними між Android-пристроєм та віддаленим сервером. Застосунок підтримує дві основні операції: завантаження файлів із сервера на локальний пристрій (download) та відправлення файлів з пристрою на сервер (upload). Обидві операції реалізовані з використанням протоколу SFTP (SSH File Transfer Protocol), що забезпечує захищену та надійну передачу даних.

Завантаження файлів (download).

На рисунку 2.9 зображено схему завантаження файлів на телефон:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

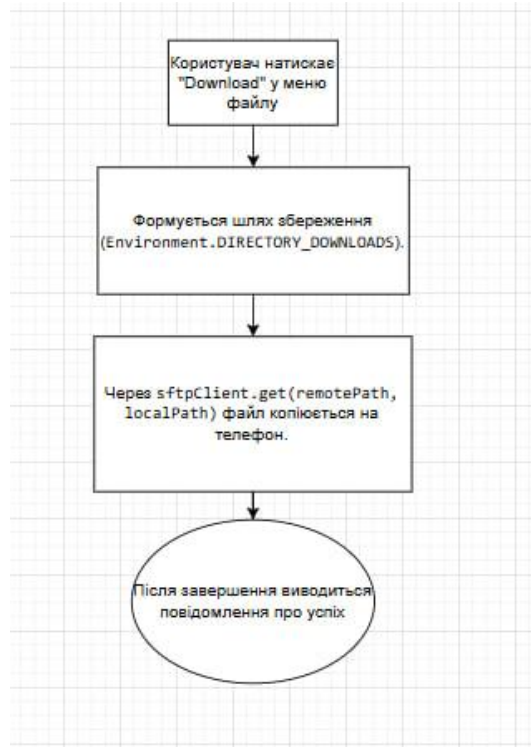


Рисунок 2.8 – Схема завантаження файлів з сервера на телефон

Процес завантаження файлу ініціюється натисканням на відповідну кнопку "Завантажити" (Download) для обраного віддаленого ресурсу. Ця дія викликає метод `onDownloadButtonClick()` у `FilesViewModel`. У цьому методі формується повний шлях до локальної директорії завантажень (зазвичай `Environment.DIRECTORY_DOWNLOADS`), куди буде збережено файл. Далі, асинхронно викликається метод `remoteFileRepository.downloadFile()`. На рівні репозиторію відбувається отримання SFTP-клієнта від `SSHManager`, формування повного віддаленого шляху до файлу, і використання методу `sftpClient.get()` для перенесення даних. Після успішного завершення операції файл зберігається на пристрої користувача, а у інтерфейсі відображається повідомлення про успішне завантаження. Логіка обробки подій та повідомлень про стан завантаження реалізована через `LiveData` та спеціальний клас `ShowSuccessDownloadMessage`.

```

public void downloadFile(String remotePath, String localPath) throws Exception {
    RemoteUser      user      =      getCurrentUser();
    SFTPClient      sftpClient =      sshManager.getSFTPClient(user);
    String          fullRemotePath =      getFullRemotePath(sftpClient, remotePath);
  
```

```
sftpClient.get(fullRemotePath, localPath);
}
```

Завантаження файлів на сервер (upload).

На рисунку 2.10 зображено схему завантаження файлів з телефону на сервер:

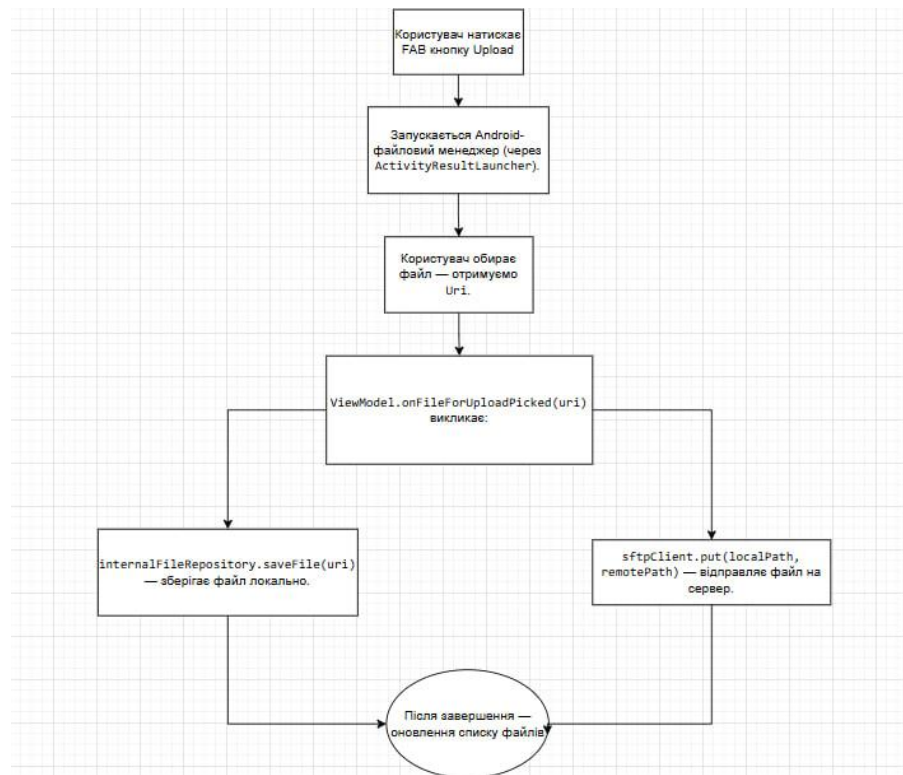


Рисунок 2.9 – Схема завантаження файлів з телефону на сервер

Процес завантаження файлу на віддалений сервер починається після натискання на Floating Action Button (FAB) з іконкою завантаження. Це дія викликає системний вибірник файлів (Android's file picker), що дозволяє користувачеві обрати файл з локального пристрою. Обраний файл передається у вигляді Uri до методу onFileForUploadPicked(Uri uri) у FilesViewModel. Процес завантаження складається з кількох ключових етапів. Спершу, використовуючи внутрішній репозиторій InternalFileRepository, створюється тимчасова копія файлу на Android-пристрої. InternalFileRepository читає байти з наданого Uri та зберігає їх у тимчасовій директорії. Після цього, ініціюється безпосереднє виконання завантаження методом uploadFile() у RemoteFileRepository. У цьому методі, після отримання SFTP-клієнта та формування повного віддаленого шляху, використовується метод sftpClient.put() для передачі локального файлу на

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

віддалений сервер. Важливою особливістю є автоматичне очищення тимчасового файлу з локального пристрою у блоці finally після завершення операції передачі, незалежно від її успішності чи невдачі.

```
public void uploadFile(Uri uri, String remotePath) throws Exception {
    String localPath = null;
    try {
        RemoteUser user = getCurrentUser();
        localPath = internalFileRepository.saveFile(uri);
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullRemotePath = getFullRemotePath(sftpClient, remotePath);
        sftpClient.put(localPath, fullRemotePath);
    } finally {
        if (localPath != null) {
            internalFileRepository.deleteFile(localPath);
        }
    }
}
```

Особливості реалізації. Обидві операції передачі файлів виконуються у фоновому потоці за допомогою асинхронного методу `async(...)` з `BaseViewModel`. Цей підхід гарантує, що інтерфейс користувача залишається чутливим та не блокується під час виконання тривалих операцій передачі даних. Усі потенційні винятки, що виникають під час цих операцій, централізовано обробляються у методі `onError()`, і користувач отримує відповідні повідомлення про успіх або помилку через Toast або інші екранні елементи. Важливо також зазначити, що передача Uri до файлу виконується без втрати прав доступу, завдяки коректній реєстрації `ActivityResultLauncher` для взаємодії із системним вибірником файлів.

Таким чином, реалізація механізмів передачі файлів у застосунку забезпечує інтуїтивну взаємодію з користувачем, гарантує безпечний та надійний обмін даними, підтримує роботу з усіма типами файлів та забезпечує просту інтеграцію з файловою системою Android. У майбутньому передбачається можливість розширення цієї функції, зокрема, шляхом додавання підтримки механізму drag-and-drop або можливості одночасного завантаження кількох файлів.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

2.4 Збереження даних користувачів через Room

Для ефективного управління даними про підключення до віддалених серверів у розробленому застосунку інтегровано вбудовану базу даних Room. Це рішення дозволяє зручно керувати декількома профілями користувачів та персистентно зберігати їхні облікові дані, що усуває потребу повторного введення інформації при кожному новому підключенні та значно підвищує зручність використання.

Загальна архітектура реалізації зберігання даних у Room складається з трьох ключових елементів: моделі сутності `RemoteUser`, інтерфейсу доступу до даних `RemoteUserDao` та самої абстрактної бази даних `AppDatabase`. Крім того, для абстрагування роботи з джерелом даних та спрощення взаємодії з `ViewModel`, було створено репозиторій `RemoteUserRepository`.

Модель користувача. Модель `RemoteUser` є сутністю, яка безпосередньо відображається на таблицю в базі даних. Вона інкапсулює облікові дані користувача та системні ідентифікатори: `id`, що є унікальним первинним ключем з автоматичною генерацією; `username` (логін користувача); `password` (пароль); та `host` (IP-адреса або доменне ім'я сервера).

```
@Entity(tableName = RemoteUser.TABLE_NAME)
public class RemoteUser {
    public static final String TABLE_NAME = "remote_users";
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    private final long id;
    @ColumnInfo(name = "username")
    private final String username;
    @ColumnInfo(name = "password")
    private final String password;
    @ColumnInfo(name = "host")
    private final String host;
```

Інтерфейс доступу до даних (DAO). `RemoteUserDao` є DAO-інтерфейсом, який визначає набір методів для виконання основних операцій CRUD (Create, Read, Update, Delete) з таблицею `remote_users`. Завдяки анотаціям Room, цей інтерфейс дозволяє безпосередньо взаємодіяти з базою даних, надаючи методи для

збереження нових або оновлення існуючих записів, отримання всіх користувачів або конкретного за id, а також видалення облікових записів.

```
@Dao
public interface RemoteUserDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    long save(RemoteUser user); // Returns the ID of the saved user

    @Query("SELECT * FROM " + RemoteUser.TABLE_NAME + " ORDER BY username ASC")
    List<RemoteUser> getAll(); // Returns all users ordered by username

    @Query("SELECT * FROM " + RemoteUser.TABLE_NAME + " WHERE id = :id LIMIT 1")
    RemoteUser get(long id); // Returns a user by ID

    @Delete
    void delete(RemoteUser user); // Deletes a user from the database
}
```

База даних Room. Клас AppDatabase є центральною точкою доступу до бази даних Room. Він оголошений як абстрактний клас, що розширює RoomDatabase, та містить абстрактний метод remoteUserDao(), який повертає екземпляр RemoteUserDao. AppDatabase реалізована за патерном Singleton, щоб гарантувати наявність лише одного екземпляра бази даних протягом життєвого циклу застосунку, запобігаючи ресурсоємним операціям створення та закриття.

```
@Database(entities = {RemoteUser.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {

    public abstract RemoteUserDao remoteUserDao();

}
```

Репозиторій RemoteUserRepository. Репозиторій RemoteUserRepository є проміжною ланкою, що абстрагує логіку взаємодії з RemoteUserDao від ViewModel. Він надає зручний API для виконання операцій з даними, таких як отримання поточного користувача (getCurrentUser()), збереження поточного користувача (saveCurrentUser()), отримання списку всіх користувачів (getAllUsers()), та

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

видалення користувача (`deleteUser()`). Важливою особливістю репозиторію є можливість зберігати ID поточного активного користувача, що дозволяє легко відновити останню робочу сесію навіть після перезапуску програми.

Взаємодія з `ViewModel` та екраном користувачів. Екран вибору збережених користувачів реалізовано у класі `UsersActivity`. Уся логіка взаємодії з цим екраном, включаючи отримання та обробку даних, зосереджена у `UsersViewModel`. Відображення списку користувачів здійснюється за допомогою `RecyclerView` та адаптера `UsersAdapter`. При натисканні на елемент списку (вибір користувача), адаптер викликає відповідний метод у `ViewModel`, який, у свою чергу, використовує `remoteUserRepository` для встановлення обраного користувача як поточного.

```
public void onClick(RemoteUser user) {
    remoteUserRepository.setCurrentUserId(user.getId()); // Далі слідує логіка переходу
    до екрану авторизації з даними вибраного користувача
}
```

Також реалізовано функціонал для видалення облікових записів безпосередньо з цього екрану, що підвищує зручність управління профілями.

Реалізація механізму збереження користувачів через `Room` забезпечує високу гнучкість у керуванні підключеннями та відкриває можливості для подальшого масштабування застосунку. У майбутньому це дозволить додати такі функції, як історія з'єднань, більш складні методи авторизації (наприклад, через SSH-ключі), або синхронізацію профілів. Таке архітектурне рішення є ефективним, надійним та повністю відповідає сучасним стандартам `Android`-розробки, забезпечуючи високий рівень користувацького досвіду.

2.5 Збереження даних користувачів через `Room`

У мобільному файловому менеджері реалізовано механізм підтримки відкладених дій, таких як копіювання або переміщення файлів, що підвищує зручність користувацької взаємодії. Цей підхід дозволяє користувачеві спочатку

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

ініціювати дію над певним файлом або папкою, а вже потім окремо вибрати цільову директорію для виконання операції, уникнувши передчасних або помилкових дій.

Центральним елементом цього механізму є клас `PendingOperationData`, який слугує простою структурою для збереження двох ключових параметрів: об'єкта типу `RemoteResourceInfo`, що описує цільовий файл або папку, над якою виконується дія, та типу операції (`PendingOperationType`), що визначає характер дії (наприклад, `COPY` або `MOVE`). Перерахування `PendingOperationType` чітко розмежовує підтримувані операції, у поточній реалізації це `COPY` для копіювання та `MOVE` для переміщення.

```
public class PendingOperationData {
    private final RemoteResourceInfo file;
    private final PendingOperationType type;
    public PendingOperationData(RemoteResourceInfo file, PendingOperationType
type) {
        this.file = file;
        this.type = type;
    }
    public RemoteResourceInfo getFile() {
        return file;
    }
    public PendingOperationType getType() {
        return type;
    }
}
```

```
public enum PendingOperationType {
    COPY,
    MOVE,
}
```

Алгоритм виконання відкладених операцій працює наступним чином: коли користувач натискає кнопку "Копіювати" або "Перемістити", у `FilesViewModel` створюється об'єкт `PendingOperationData`, який зберігається у поточному стані `FilesScreenState`. Після цього додаток ініціює перехід до екрану вибору цільової директорії за допомогою відповідної події (`ShowFolderChooser`). Коли користувач обирає директорію, викликається метод `onFolderChosen(String folderPath)`, який

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

перевіряє збережену операцію у PendingOperationData та виконує відповідну дію (копіювання або переміщення) через RemoteFileRepository.

Такий підхід забезпечує повне розділення логіки вибору файлу та його цільової директорії, підвищуючи зручність для користувача. Крім того, він зберігає гнучкість системи, дозволяючи легко розширити функціонал для підтримки інших багатоетапних операцій у майбутньому. Ця реалізація також спрощує підтримку коду, оскільки операції не виконуються миттєво, а плануються та підтверджуються, роблячи файловий менеджер більш динамічним та адаптивним для реального використання.

2.6 Підготовка серверного середовища для тестування

Для комплексного тестування мобільного файлового менеджера було створено локальне серверне середовище на базі двох популярних дистрибутивів Linux — Ubuntu Server та Debian, обидва розгорнуті у середовищі віртуалізації Oracle VirtualBox. Така конфігурація дозволила не лише перевірити роботу додатку з різними ОС, а й протестувати підключення як по IP-адресі, так і по доменному імені, що стало можливим завдяки налаштуванню демона Avahi (служби mDNS/Bonjour).

Вибір програмного та апаратного забезпечення:

- Oracle VirtualBox — створення та керування віртуальними машинами [16];
- Ubuntu Server 22.04 — перший сервер для початкового налаштування [17];
- Debian 12 — для перевірки сумісності з іншими популярними дистрибутивами [18];
- OpenSSH Server — сервер SSH для забезпечення з'єднання [19];
- Avahi Daemon — служба автоматичного оголошення доменних імен у локальній мережі [20].

Інсталяція та конфігурація серверів

На кожній ВМ було виконано:

- Інсталяцію ОС з ISO-образу;
- Встановлення OpenSSH

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

- Створення користувача з правами sudo;
- Інсталяція Avahi (sudo apt install avahi-daemon) для доступу по доменному імені типу hostname.local.

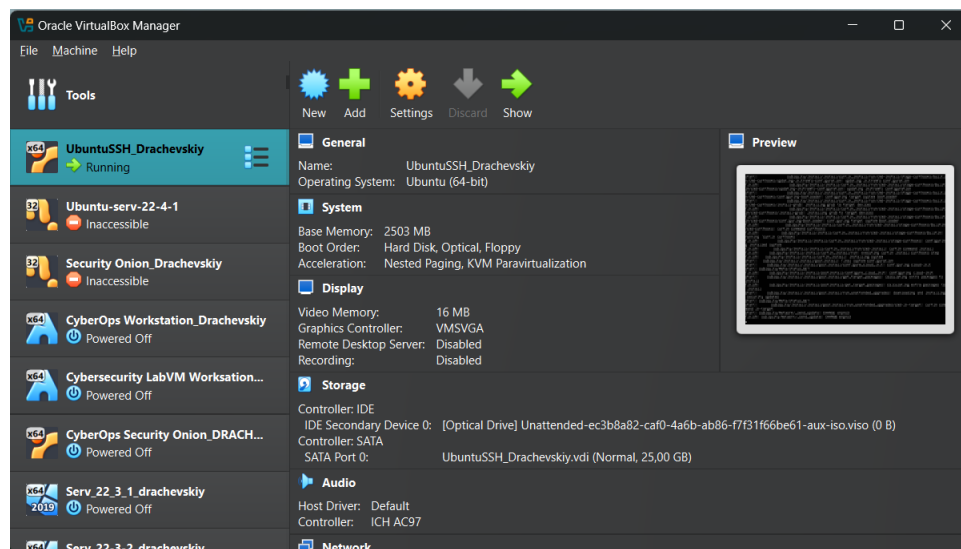


Рисунок 2.10 – Вигляд робочої станції Ubuntu

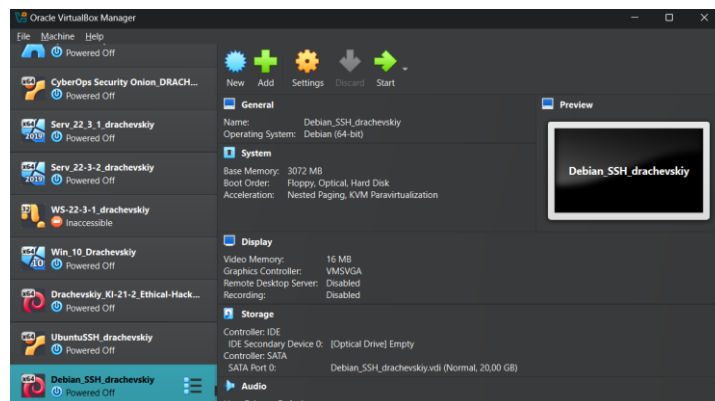


Рисунок 2.11 – Вигляд робочої станції Debian

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

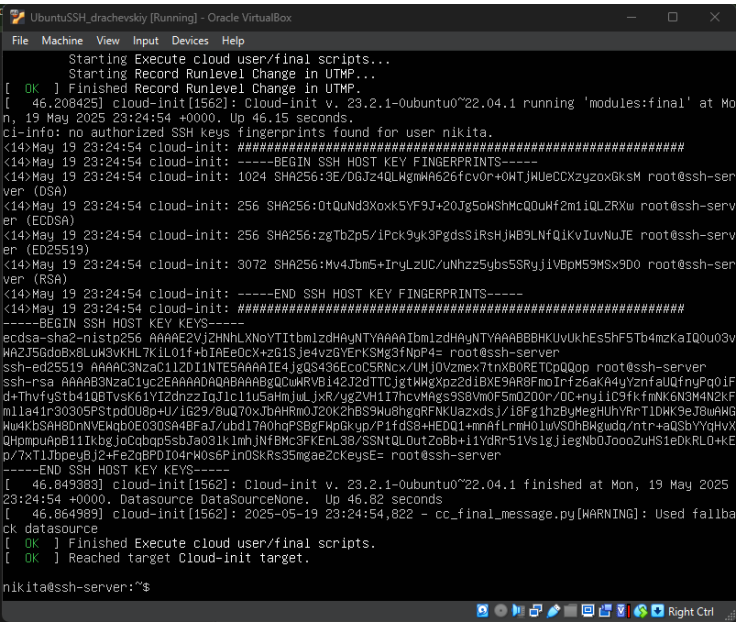


Рисунок 2.12 – Успішний запуск Ubuntu

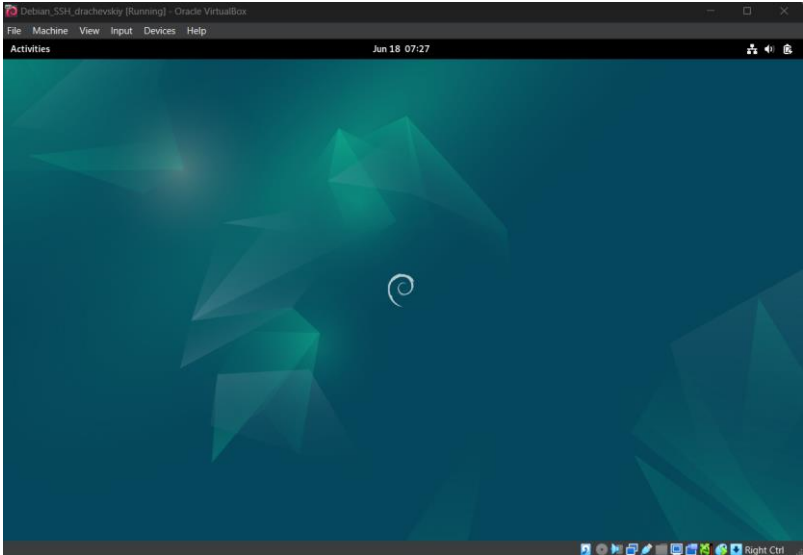


Рисунок 2.13 – Успішний запуск Debian

```
nikita@ssh-server:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d7:bd:0b brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.110/24 metric 100 brd 192.168.0.255 scope global dynamic enp0s3
        valid_lft 7164sec preferred_lft 7164sec
    inet6 fe80::a00:27ff:fed7:bd0b/64 scope link
        valid_lft forever preferred_lft forever
nikita@ssh-server:~$ _
```

Рисунок 2.14 – ip a на Ubuntu

```
nikita@debian:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:4b:de:1c brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.113/24 brd 192.168.0.255 scope global dynamic noprefixroute enp0s3
        valid_lft 6981sec preferred_lft 6981sec
    inet6 fe80::a00:27ff:fe4b:de1c/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
nikita@debian:~$
```

Рисунок 2.15 – ip a на Debian

```
nikita@debian:~$ su -
Password:
root@debian:~# sudo apt update
Hit:1 http://security.debian.org/debian-security bookworm-security InRelease
Hit:2 http://deb.debian.org/debian bookworm InRelease
Hit:3 http://deb.debian.org/debian bookworm-updates InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
27 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: Skipping acquire of configured file 'non-free-firmware/source/Sources' as repository 'http://security.debian.org/debian-security bookworm-security InRelease' doesn't have the component 'non-free-firmware' (component misspelt in sources.list?)
root@debian:~# sudo apt install avahi-daemon
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
avahi-daemon is already the newest version (0.8-10+deb12u1).
avahi-daemon set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 27 not upgraded.
root@debian:~# sudo systemctl enable avahi-daemon
root@debian:~# sudo systemctl start avahi-daemon
root@debian:~#
```

Рисунок 2.16 – Встановлення avahi-daemon на Debian

```
nikita@ubuntu:~$ sudo systemctl enable avahi-daemon
Synchronizing state of avahi-daemon.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable avahi-daemon
nikita@ubuntu:~$ sudo systemctl start avahi-daemon
nikita@ubuntu:~$ hostname
ubuntu
nikita@ubuntu:~$
```

Рисунок 2.17 – Встановлення avahi-daemon на Ubuntu

Перевірка роботи SSH

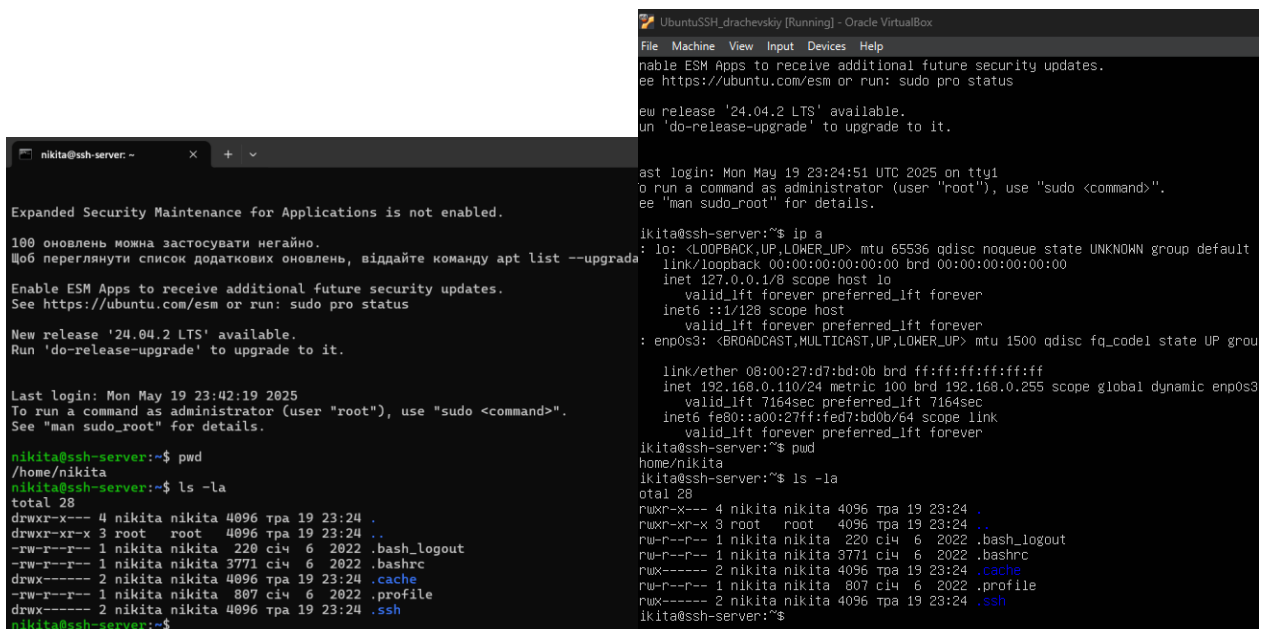
		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

Після інсталяції було перевірено доступність сервера через SSH. З хост-машини

У випадку успішного підключення система запитує пароль та надає доступ до командного рядка Ubuntu.

```
nikita@ssh-server: ~  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
System information as of понеділок, 19 травня 2025 23:44:30 +0000  
  
System load:  0.09375      Processes:            108  
Usage of /:   45.0% of 11.21GB Users logged in:      1  
Memory usage: 10%          IPv4 address for enp0s3: 192.168.0.110  
Swap usage:   0%  
  
Expanded Security Maintenance for Applications is not enabled.  
  
100 оновлень можна застосувати негайно.  
Щоб переглянути список додаткових оновлень, віддайте команду apt list --upgradable  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
New release '24.04.2 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Mon May 19 23:42:19 2025  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
nikita@ssh-server:~$ |
```

Рисунок 2.18 – Перевірка з хоста а саме з Windows



а)

б)

Рисунок 2.19 – Тестування одних і тих самих команд на сервері і на хості (команда ls -la)

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

а - тестування з хосту; б – тестування на сервері

Також було дозволено підключення з мобільного пристрою в локальній мережі — це стало основою для роботи мобільного додатку.

Підготовка файлової структури

На сервері вручну створено кілька папок для тестування дій:

- /home/user/test-folder — для перевірки створення і видалення;
- інші каталоги для перевірки глибокої навігації та доступу.

Мережеві аспекти

Усі сервери проходили перевірку:

- Пінг з Android-пристрою;
- Відкритість порту 22 (SSH);
- Доступність *.local доменів через Avahi.

Завдяки використанню двох серверів (Ubuntu і Debian), а також налаштуванню Avahi для роботи з локальними доменами, середовище стало наближеним до реального корпоративного або освітнього використання. Тестування на таких умовах дозволяє гарантувати, що додаток працює стабільно, гнучко та підтримує сучасні мережеві практики, включно з mDNS.

Висновок до другого розділу

У другому розділі було детально розглянуто процес проектування та реалізації мобільного застосунку для адміністрування Linux-серверів через SSH. Основна увага приділялася побудові структури застосунку за архітектурною моделлю MVVM, що забезпечує чітке розділення відповідальностей між інтерфейсом, бізнес-логікою та доступом до даних.

Окремо проаналізовано екрани авторизації, перегляду файлів і вибору користувача, що забезпечують ключову функціональність для зручного керування віддаленими серверами. Реалізовано збереження облікових даних користувачів за допомогою бази даних Room, а також повноцінну роботу з файловою системою:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

перегляд, копіювання, перейменування, переміщення, видалення, завантаження та вивантаження файлів.

Важливою складовою також стала обробка черги операцій (copy/move) через об'єкт PendingOperationData, що забезпечує послідовну взаємодію з файловою системою. Було налаштовано повноцінне тестове середовище на базі Ubuntu і Debian із підтримкою доменних імен (через Avahi), що дозволило перевірити роботу застосунку не лише по IP, а й по локальному імені сервера.

Таким чином, реалізація програмної частини довела життєздатність запропонованої архітектури, гнучкість Android-платформи для задач адміністрування, а також ефективність застосування сучасних бібліотек і підходів для створення мобільного SSH-клієнта.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

РОЗДІЛ 3 ТЕСТУВАННЯ ТА ВДОСКОНАЛЕННЯ ФАЙЛОВОГО МЕНЕДЖЕРА

3.1 Перевірка з'єднання з сервером, передача команд

Після завершення реалізації функціоналу мобільного застосунку було проведено ретельне тестування підключення до серверів по SSH. Метою тестування було підтвердити стабільність з'єднання, правильність обробки логіки авторизації, а також здатність виконувати базові адміністративні команди.

Для проведення тестування функціональності застосунку було розгорнуто два віртуальні сервери. Перший — це Ubuntu Server 22.04, а другий — Debian 12, обидва працювали у середовищі VirtualBox. На обох віртуальних машинах було успішно встановлено openssh-server для забезпечення SSH-доступу. Крім того, на кожному сервері було запущено сервіс Avahi Daemon, який відповідає за резолвінг доменних імен у локальній мережі за схемою hostname.local. Це дозволило тестувати підключення до серверів не лише за їхніми IP-адресами, але й за символьними іменами, такими як ubuntu.local та debian.local, значно спрощуючи процес ідентифікації та підключення у тестовому середовищі.

Процес підключення до віддаленого сервера починався на екрані авторизації, де користувач вводив необхідні параметри: логін, пароль та адресу сервера (IP-адресу або DNS-ім'я), після чого натискав кнопку "Підключитися". Застосунок ініціював встановлення з'єднання через SSHManager. У випадку успішного підключення, здійснювався автоматичний перехід до інтерфейсу файлового менеджера. Візуалізація етапів підключення представлена на рис.3.1-3.3.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		66



Рисунок 3.1 – Стартовий екран де видно усі сервери до яких підключався



Рисунок 3.2 – Виділення з бази даних одного з серверів

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				67
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 3.3 – При натисненні на сервер відкривється ConnectionActivity після чого поля вже заповненні і можна натискати кнопку Connect для підключення до серверу

Після успішного з'єднання додаток переходить до Home директорії, (рис.3.4).

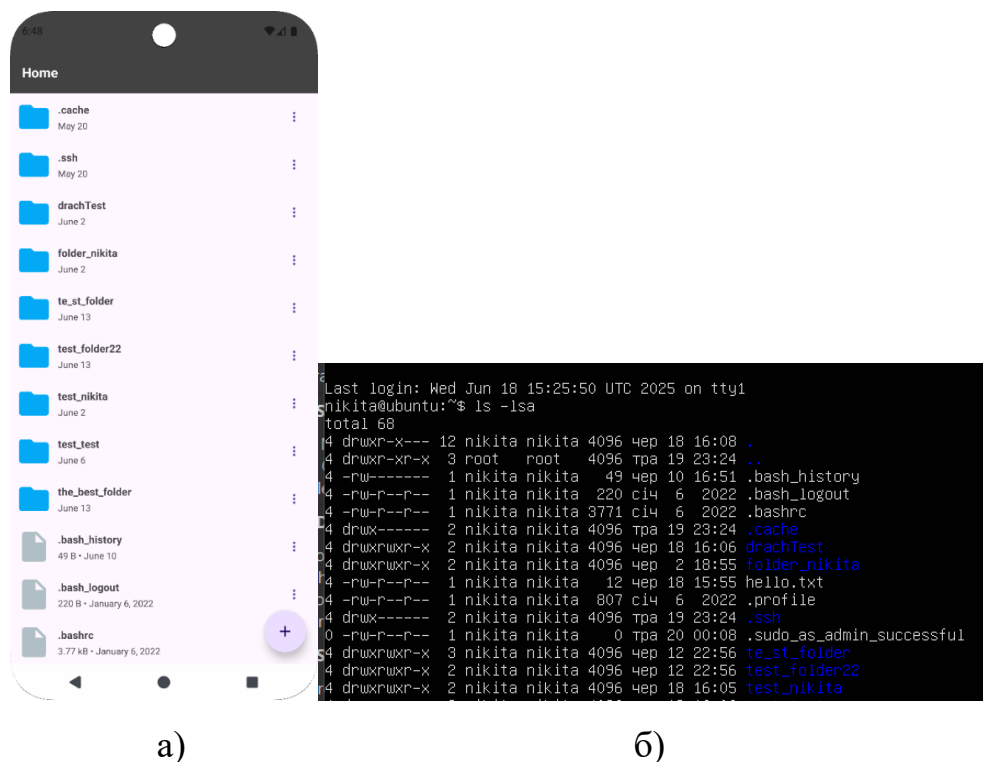


Рисунок 3.4 – Вивід домашньої деректорії сервера Ubuntu і підтвердження правильності на сервері

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

а – тестування з додатку; б – тестування з серверу

Те ж саме відбувається і на іншому сервері (рис.3.5).

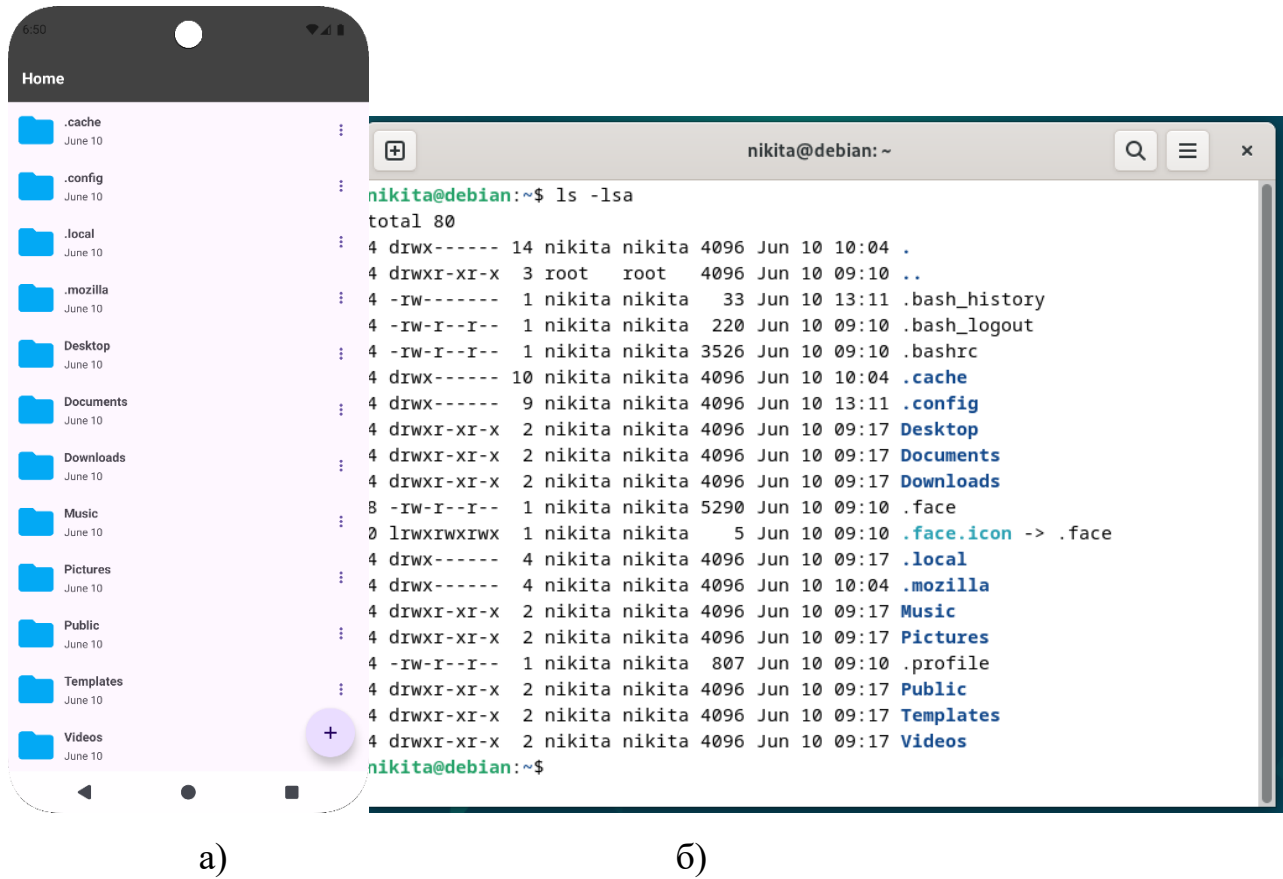


Рисунок 3.5 – Номе деректорія серверу Debian і підтвердження правильності на сервері

а – тестування з додатку; б – тестування з серверу

Особлива увага під час тестування приділялась негативним сценаріям, що передбачали некоректне введення даних або відсутність доступу до сервера. Було протестовано такі ситуації: використання неправильного пароля при спробі авторизації; спроба підключення до неіснуючого хоста (наприклад, wronghost.local); та ситуація, коли віддалений сервер був недоступний або не підключений до мережі. У кожному з цих випадків система успішно обробляла відповідні винятки та виводила інформативні повідомлення користувачеві через системні сповіщення (Toast) або безпосередньо в інтерфейсі користувача.

Рисунок 3.6 демонструє обробку помилки.



Рисунок 3.6 – Спроба підключитись до неіснуючого серверу

Результати проведеного тестування підтвердили високу стабільність з'єднання та коректність виконання широкого спектра команд на віддалених серверах під керуванням Ubuntu та Debian. Архітектура розробленого застосунку демонструє значну гнучкість, дозволяючи легко розширювати набір підтримуваних команд та функціоналу. Підтримка доменних імен (*.local) завдяки сервісу Avahi Daemon значно підвищує зручність використання у локальних мережах. Окрім цього, реакція на помилки реалізована відповідно до сучасних стандартів User Experience (UX), що забезпечує зрозумілу та передбачувану поведінку застосунку навіть у разі виникнення збоїв або некоректних вхідних даних.

3.2 Тестування роботи з файлами та обробка помилок

Цей розділ присвячений тестуванню ключових функцій файлового менеджера, які забезпечують взаємодію користувача з віддаленою файловою

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

системою. Тестуванню підлягали операції перегляду директорій, створення нових елементів, перейменування, копіювання, переміщення, видалення та відкриття файлів. Особливу увагу було зосереджено на перевірці стійкості застосунку до помилок та його реакції на нестандартні або некоректні ситуації.

Перегляд директорій.

Після успішного встановлення SSH-з'єднання мобільний застосунок автоматично виконує команду `ls` для отримання вмісту поточної директорії. Отримані дані відображаються у компоненті `RecyclerView`, який підтримує навігацію та переходи у вкладені каталоги. Проведене тестування підтвердило успішне відображення як великої кількості елементів у папці, так і коректний вивід повідомлення «Папка порожня» у випадку відсутності вмісту в директорії, рис.3.7, рис.3.8.

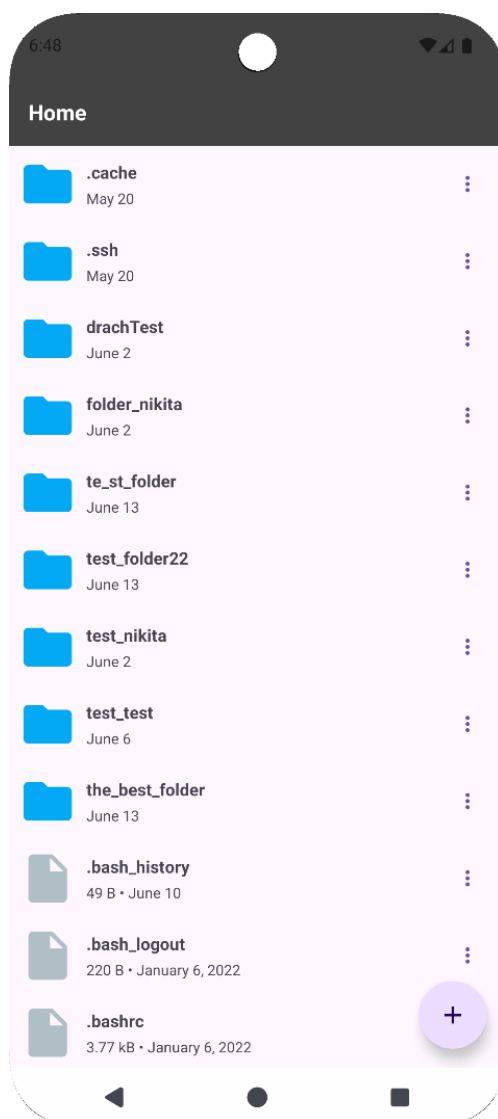


Рисунок 3.7 – Повна директорія

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

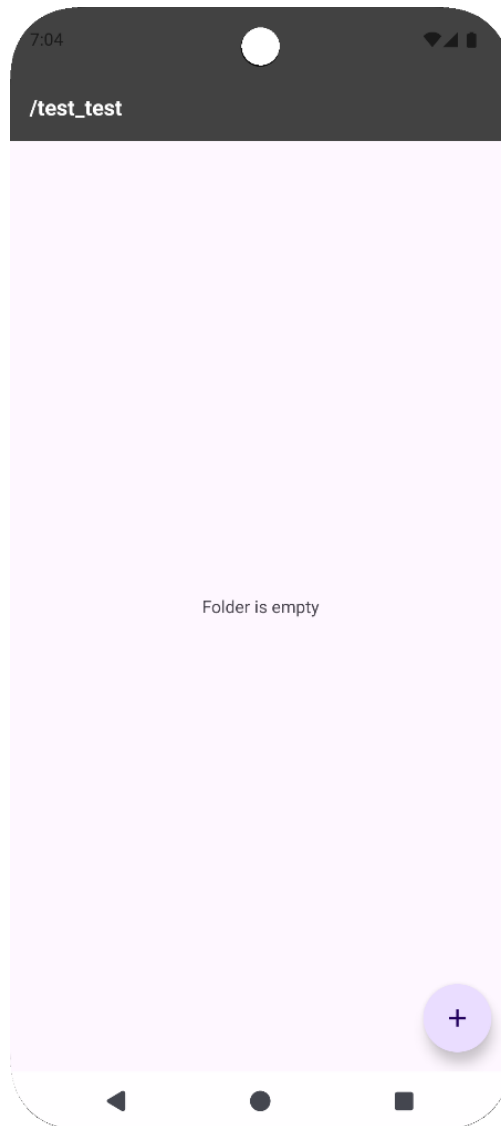


Рисунок 3.8 – Порожня директорія

Створення директорій.

Функція створення нової папки активується через Floating Action Button (FAB). Після введення бажаної назви папки, викликається метод `createFolder()` у відповідному ViewModel, який ініціює виконання команди `mkdir` на віддаленому сервері. Результати тестування засвідчили стабільну роботу функції при створенні унікальних папок. Було також підтверджено коректне відображення повідомлень про помилки у випадках, коли користувач намагався створити папку з порожньою назвою або назвою, що вже існує (дублювання). Аналогічно, система успішно

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

блокує використання некоректних символів у назві папки, супроводжуючи це відповідним повідомленням, рис.3.9, 3.10, 3.11.

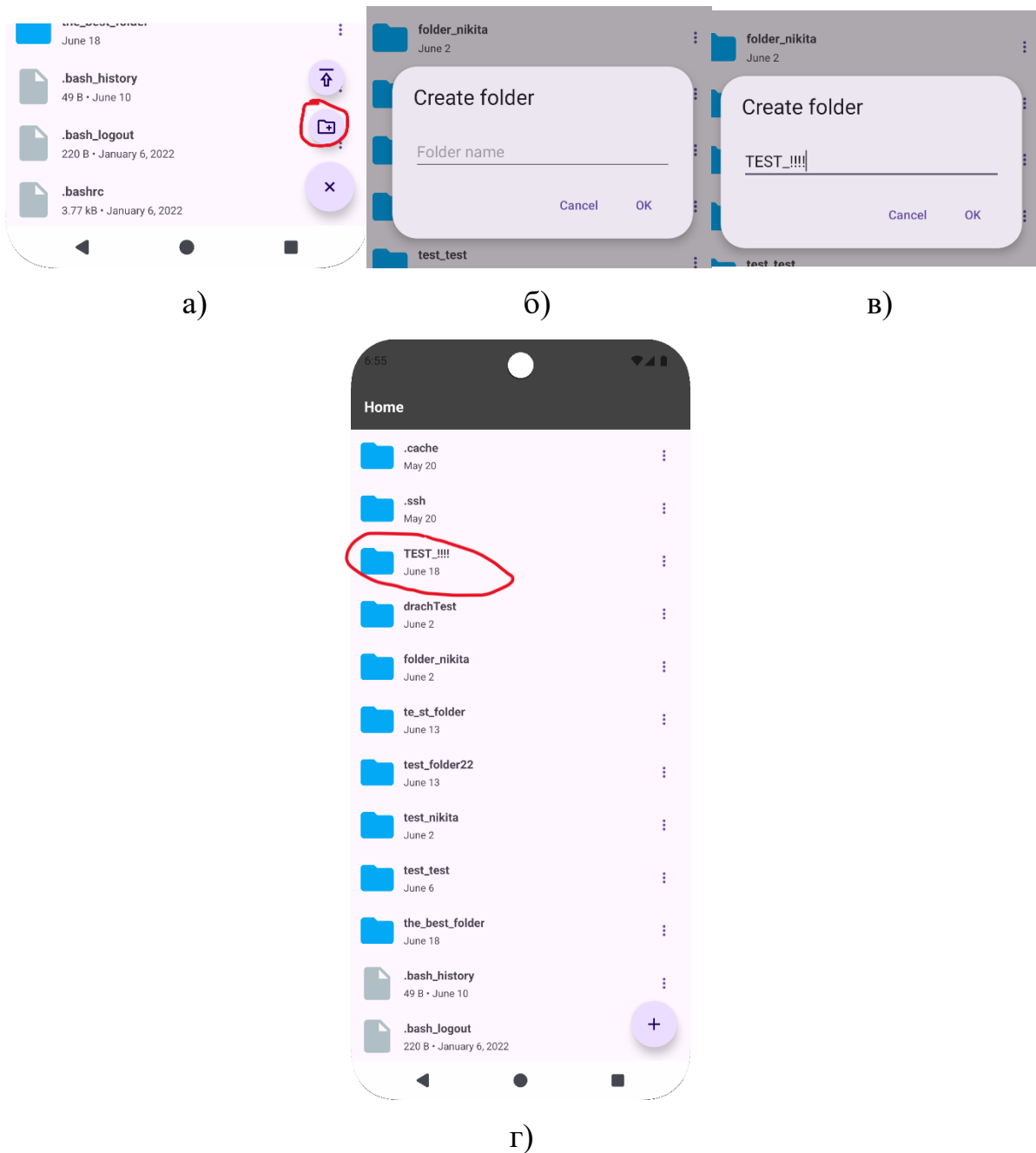
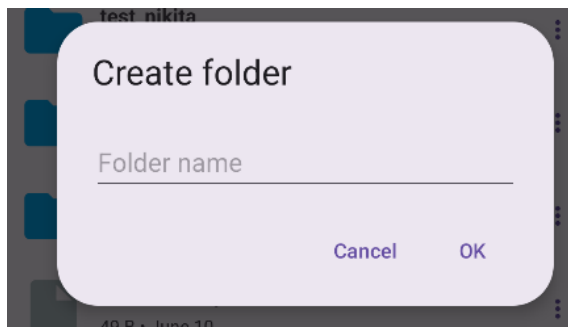
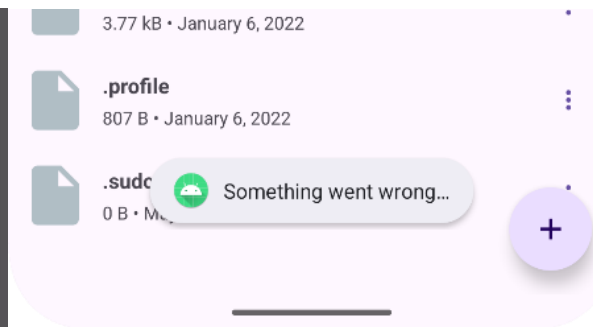


Рисунок 3.9 – Створення папки з унікальним ім’ям

а – натиснення кнопки; б – діалогове вікно; в – ввід унікального імені; г – демонстрація створеної папки



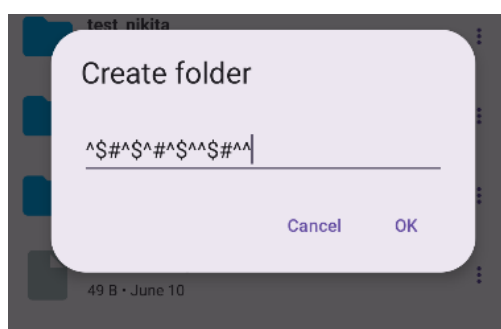
а)



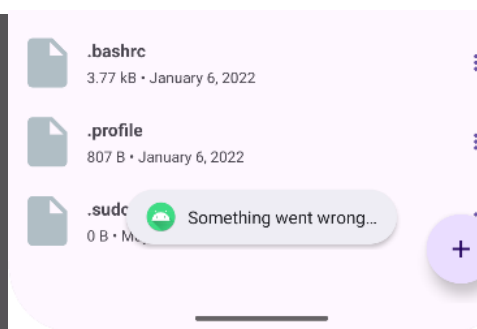
б)

Рисунок 3.10 – Створення папки без імені

а – спроба натиснути ок з пустим полем; б - помилка



а)



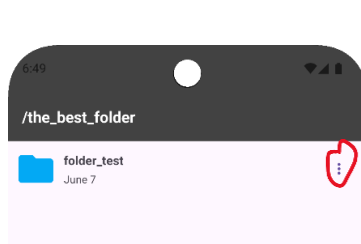
б)

Рисунок 3.11 – Створення папки без імені

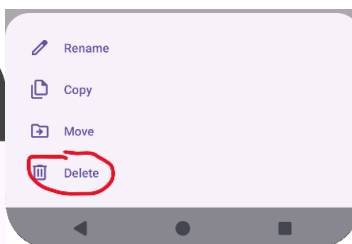
а – заповнення поля символами; б – помилка

Видалення елементів.

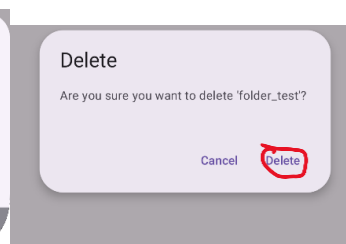
Механізм видалення файлів та папок реалізовано з використанням команди `rm -r` на віддаленому сервері. Користувач обирає файл або папку, після чого операція підтверджується у спеціальному діалоговому вікні. Сценарії тестування включали успішне видалення звичайних файлів, а також рекурсивне видалення папок з усім їхнім вмістом, рис.3.12.



а)



б)



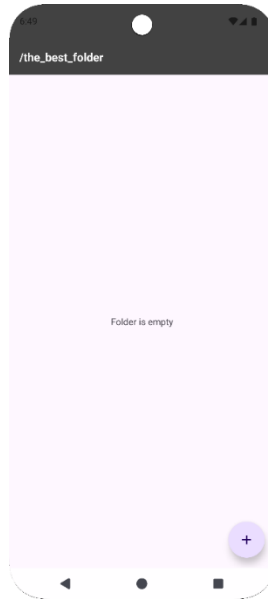
в)

		Драчевський Н.В.		
		Бродський Ю.Б.		
Змн.	Арк.	№ докум.	Підпис	Дата

ККІКБ.КР.Б – 123 – 03 – ПЗ

Арк.

74



г)

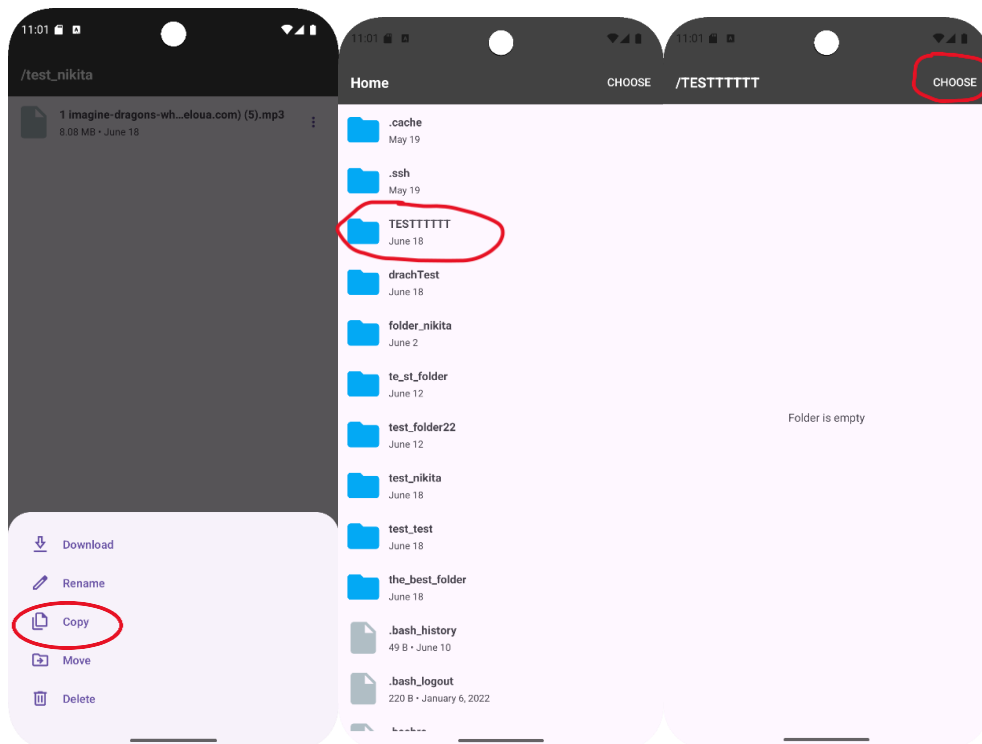
Рисунок 3.12 – Процес видалення

а – натиснення на три крапки; б – меню вибору і натиснення на видалити; в – вивід діалогового вікна підтвердження; г – демонстрація папки

Копіювання, переміщення, перейменування.

Ці операції, що є основними для управління файлами, доступні через окреме контекстне меню (символ "три крапки") для кожного елемента. Користувач може обрати потрібну дію: копіювання, що дублює об'єкт у вказану цільову директорію (рис.3.13); переміщення, що переносить об'єкт у нове місце розташування (рис.3.14); або перейменування, що змінює ім'я ресурсу (рис.3.15);. Передача параметрів для цих багатоетапних операцій здійснюється за допомогою класу PendingOperationData, який інкапсулює тип операції (COPY або MOVE) та інформацію про файл.

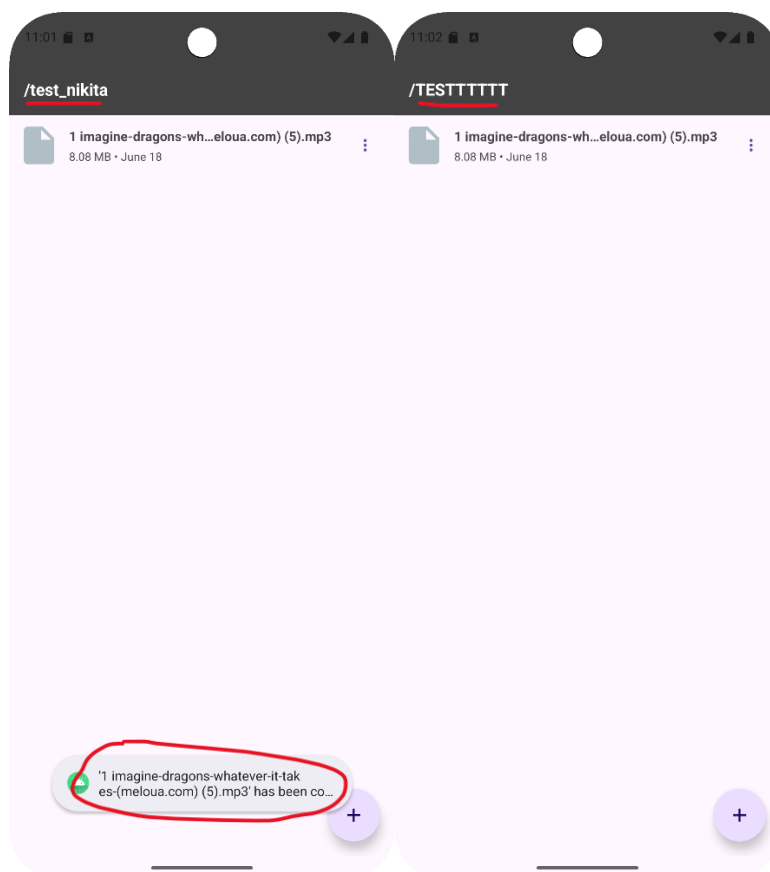
		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		75



а)

б)

в)



г)

д)

Рисунок 3.13 – Процес копіювання

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

а – меню вибору і натиснення на копіювати; б – вікно вибору папки; в – вікно з вибраною папкою; г – вікно папки з якої копіювали і повідомлення про успішність; д – вікно папки в яку копіювали

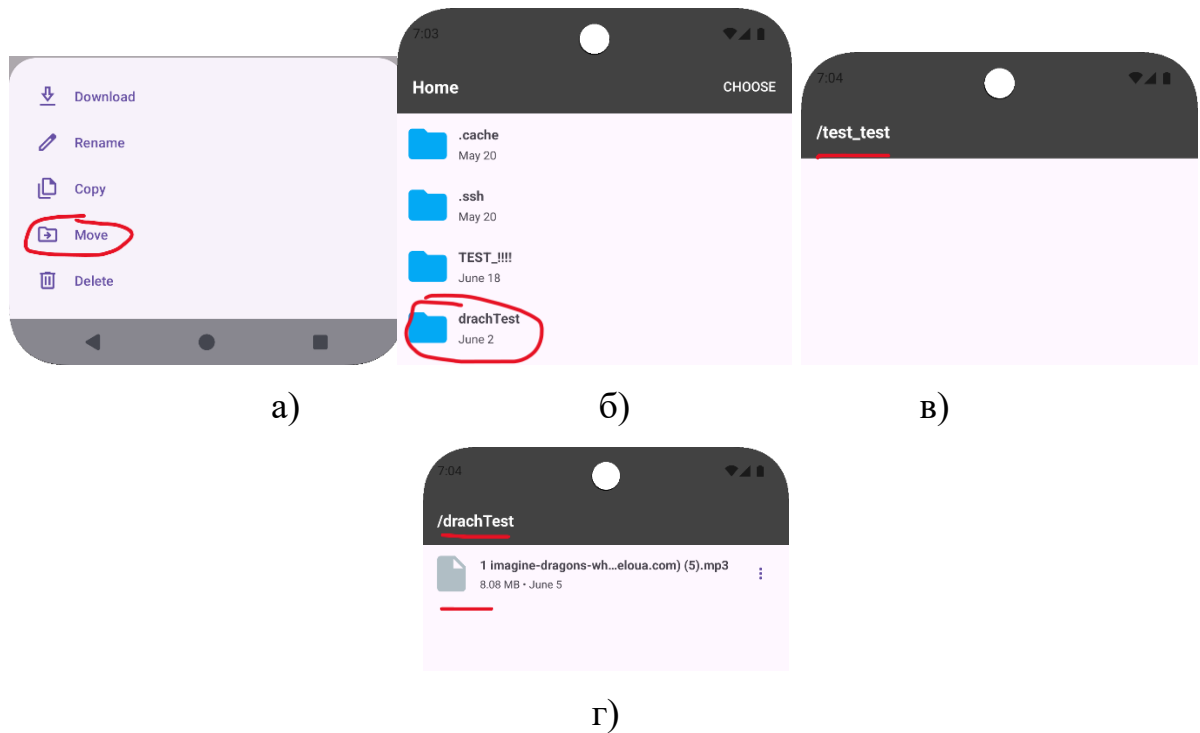


Рисунок 3.14 – Процес переносу файлу

а – меню вибору і натиснення на переміщення; б – вікно вибору папки; в – вікно з папкою з якої переміщали; г – вікно папки в яку переміщували;

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

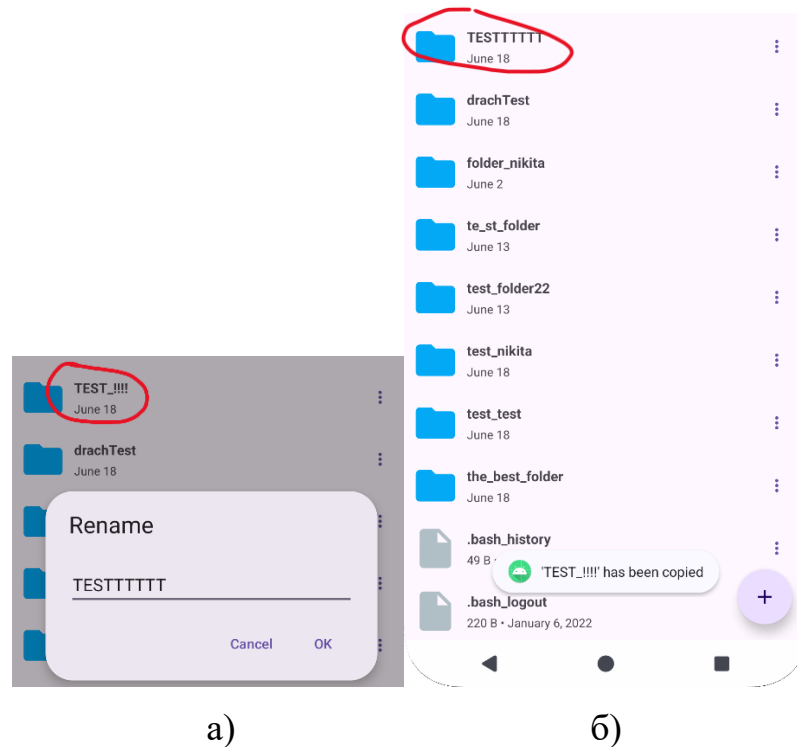


Рисунок 3.15 – Процес перейменування

а – заповнення поля для перейменування папки TEST_!!!; б – підтвердження про успішне перейменування

Відкриття файлів. Функція відкриття файлів реалізована за натисканням на відповідний елемент у списку. Система визначає MIME-тип файлу та створює Intent.ACTION_VIEW, який ініціює відкриття файлу у відповідному зовнішньому застосунку, встановленому на Android-пристрої.

```
// Приклад методу для відкриття файлу на рівні репозиторію
RemoteFile openFile(String fullRemotePath) throws Exception {
    RemoteUser user = getCurrentUser();
    SFTPClient sftpClient = sshManager.getSFTPClient(user);
    return sftpClient.open(fullRemotePath);
}
```

Логіка у ViewModel для обробки натискання на файл виглядає наступним чином:

```
public void onFileClick(RemoteResourceInfo file) {
    if (file.isDirectory()) {
        // Якщо це директорія, перейти до неї
        listFiles(new File(currentPath, file.getName()).getPath());
    } else {
```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

```
// Якщо це файл, ініціювати подію відкриття
stateLiveData.setValue(currentState.copyWithEvent(new OpenFile(file)));
}
}
```

Обробка події у FilesActivity включає визначення MIME-типу файлу за його розширенням та створення Intent для запуску зовнішнього застосунку:

```
// Фрагмент коду в FilesActivity для обробки події OpenFile
String mimeType =
MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension);
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(uri, mimeType); // uri - content:// URI до тимчасового файлу
startActivity(intent);
```

Важливою особливістю є те, що файли завантажуються у тимчасову директорію на локальному пристрої та передаються до зовнішніх застосунків через content:// URI з обмеженими правами доступу, що гарантує безпеку даних (рис.3.16).

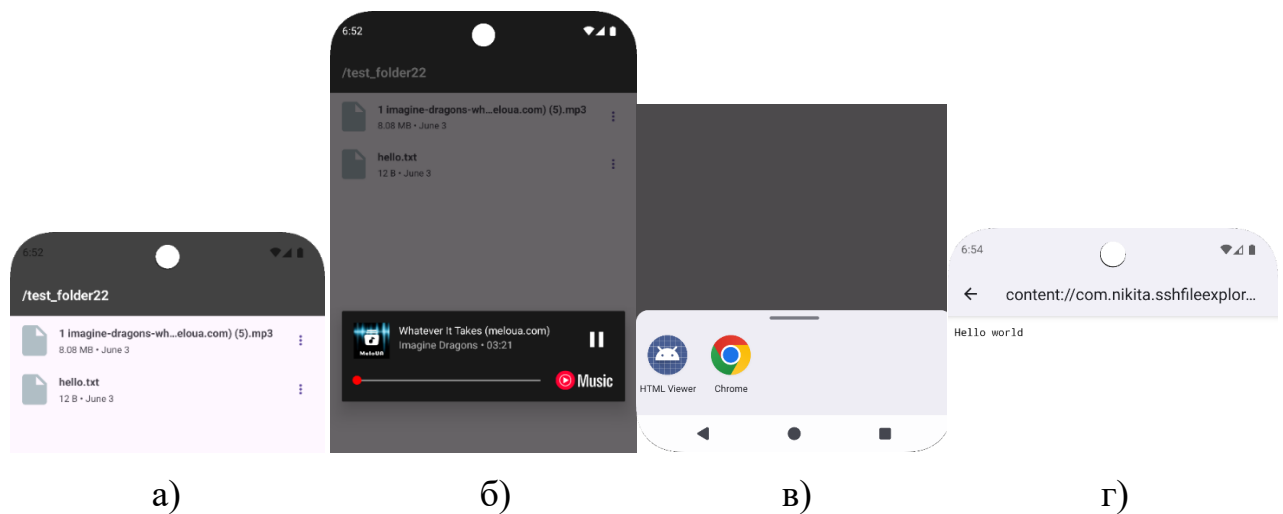


Рисунок 3.16 – Процес відкриття файлів (текстовий файл, файл музики)
а – вікно з файлами і натиснення на них; б – вікно прослуховування файлу
музики; в – вікно вибору додатка для відкриття текстового файлу; г –
підтвердження відкриття текст. файлу;

Обробка завантаження з додатку і вивантаження на телефон.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

Користувач може завантажити файл із сервера у локальну директорію пристрою. Функція активується через меню додаткових дій у файлі. Після підтвердження файл копіюється у папку Downloads. Також користувач може вибрати файл з локальної пам'яті телефону та надіслати його на віддалений сервер. Дія активується через FAB («Upload») (рис.3.17).

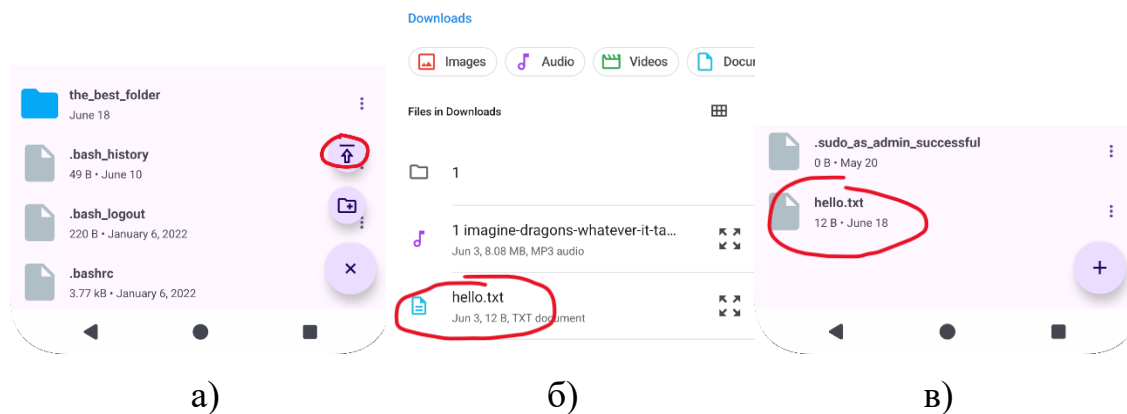


Рисунок 3.17 – Процес upload

а – натиснення на кнопку Upload; б – вікно вибору файлу з телефону; в – вікно додатку з завантаженим файлом;

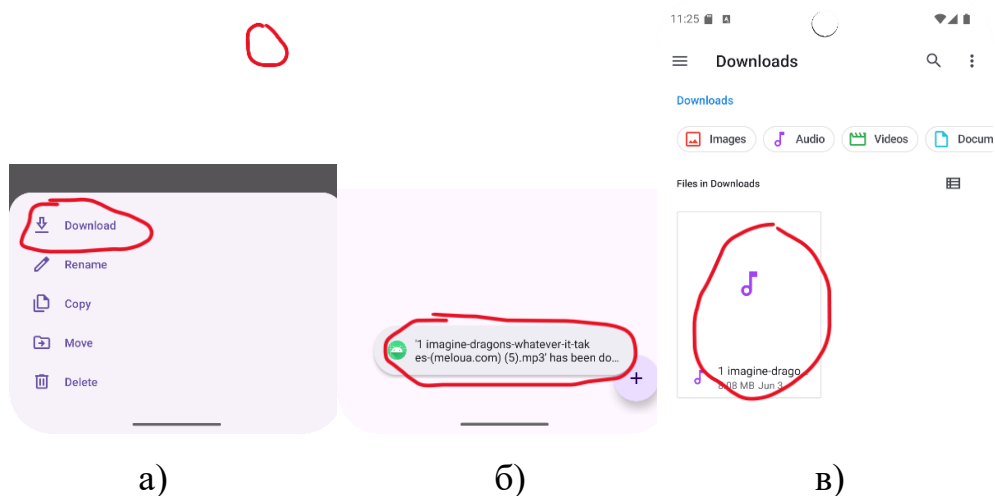


Рисунок 3.18 – Процес download

а – меню вибору і натиснення на Download; б – повідомлення про успішне встановлення; в – вікно файлів телефону з завантаженим файлом з сервера;

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

Також на наступних скріншотах видно результат відключення інтернету (рис.3.19).

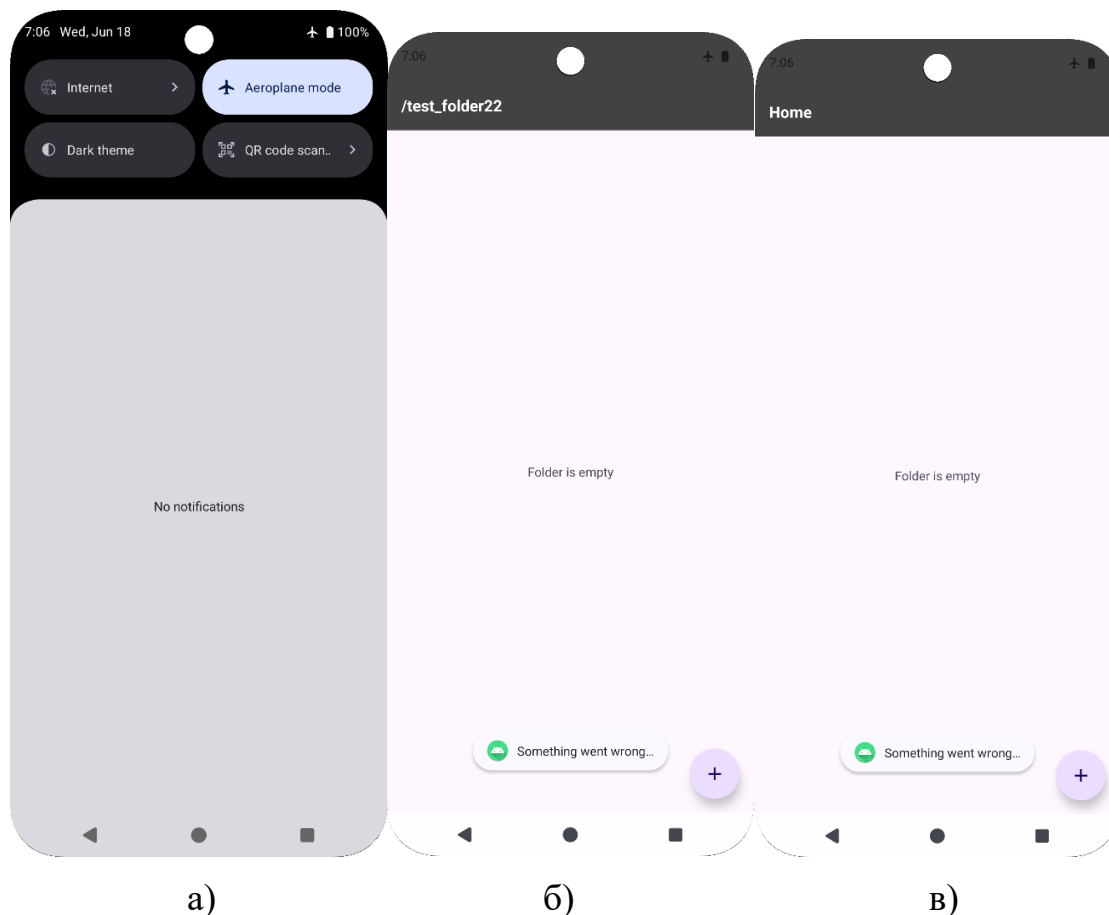


Рисунок 3.19 – Процес відключення інтернету

а – вимкнення інтернету на телефоні; б – вікно поточної папки з помилкою; в –
Номер директорія з помилкою;

Проведене тестування підтвердило стабільну роботу усіх реалізованих функцій файлового менеджера. Усі виняткові ситуації та помилки обробляються грамотно, а інтерфейс користувача залишається чутливим та адаптивним до дій користувача. Завдяки послідовному використанню архітектури MVVM та відокремленим репозиторіям, додаток демонструє високу розширюваність та надійність у роботі, що є критично важливим для дипломного проєкту.

3.3 Оцінка продуктивності застосунку та UX-досвід користувача

Проведено комплексну оцінку продуктивності розробленого Android-застосунку, а також детальний аналіз користувацького досвіду (UX). Основна увага приділялася ключовим аспектам, таким як швидкодія системи, стабільність

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

функціонування при взаємодії з віддаленим сервером, інтуїтивність та зрозумілість інтерфейсу, а також адаптація застосунку до різноманітних сценаріїв використання, що є критично важливим для його ефективності та зручності.

Продуктивність (Performance). Аналіз продуктивності застосунку виявив оптимальні показники швидкодії на всіх етапах взаємодії. Час встановлення SSH-з'єднання з віддаленим сервером в середньому становить від 1.2 до 1.6 секунди, причому цей показник може незначно змінюватися залежно від поточного навантаження на сервер. Завдяки реалізації асинхронного виконання операцій за допомогою ExecutorService, інтерфейс користувача залишається повністю чутливим, що виключає відчуття "зависання" навіть під час мережевих затримок.

Операції з файлами також демонструють високу швидкодію. Завантаження списку директорій, що містять понад 100 елементів, відбувається в діапазоні від 500 до 800 мілісекунд. Такі операції, як копіювання та переміщення файлів, які виконуються через SSH-команди та протокол SFTP, не спричиняють помітних затримок, зазвичай завершуючись менш ніж за 1 секунду. Час, необхідний для завантаження або вивантаження файлів розміром до 5 мегабайт, становить до 2-3 секунд, при цьому цей показник безпосередньо залежить від пропускної здатності та стабільності мережевого з'єднання.

Щодо споживання ресурсів, застосунок демонструє високу ефективність. Він стабільно функціонує на пристроях з обсягом оперативної пам'яті від 2 ГБ та операційною системою Android 8.0 або новішою. Середнє споживання оперативної пам'яті застосунком не перевищує 100 МБ, що свідчить про його оптимізацію та мінімальний вплив на загальну продуктивність пристрою.

Стабільність. Проведене тестування стабільності застосунку охоплювало різноманітні умови експлуатації для імітації реальних сценаріїв використання. Тести проводилися за умов зниженого сигналу Wi-Fi, при імітації відключення віддаленого сервера, а також при повторному відкритті застосунку після його переведення у фоновий режим. У всіх вищезазначених випадках застосунок демонстрував високу стійкість: він коректно обробляв усі виняткові ситуації,

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

виводив інформативні повідомлення для користувача та не допускав аварійного завершення роботи, що підтверджує його надійність.

UX-досвід (User Experience). Інтерфейс користувача розроблений у суворій відповідності до сучасних рекомендацій та принципів Material Design [21], що забезпечує його інтуїтивність та привабливість. Дизайн характеризується мінімальною кількістю елементів на екрані, що сприяє його чистоті та легкості сприйняття. Використання Floating Action Button (FAB) забезпечує швидкий доступ до ключових дій. Застосунок демонструє високу адаптивність до різних розмірів екранів, коректно відображаючись як на маленьких смартфонах, так і на великих планшетах. Візуальний поділ директорій та файлів реалізований за допомогою іконок, відображення розміру та дати модифікації, що значно покращує навігацію та візуальне сприйняття інформації.

До основних UX-переваг застосунку можна віднести функцію автозаповнення збережених користувачів, яка реалізована через інтеграцію з базою даних Room, що значно економить час при повторних підключеннях. Забезпечено швидко та безперешкодну навігацію між папками, а також миттєву реакцію інтерфейсу на дії користувача. Система сповіщень реалізована за допомогою Toast-повідомлень та діалогових вікон помилок, що забезпечує зрозумілий та своєчасний зворотний зв'язок з користувачем.

Загальна оцінка продуктивності застосунку показала, що він функціонує швидко та ефективно, не створюючи надмірного навантаження на системні ресурси мобільного пристрою. Висока стабільність роботи, особливо при обробці мережеских збоїв та несприятливих умов, підтверджує його надійність. Інтерфейс користувача розроблений з урахуванням специфічних сценаріїв використання системного адміністратора, забезпечуючи легкий доступ до файлів, можливість зміни структури директорій, а також зручні функції перегляду та завантаження. Таким чином, проведений аналіз UX-досвіду підтверджує, що навіть у початковій версії застосунку він є зручним та придатним для повсякденної роботи з віддаленими серверами.

Висновок до третього розділу

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

У межах третього розділу було проведено комплексне тестування функціональних можливостей мобільного файлового менеджера, зосереджене на перевірці з'єднання, передачі SSH-команд, взаємодії з файловою системою віддаленого сервера та стійкості додатку до помилок і нестабільних умов роботи.

Було підтверджено стабільну роботу з'єднання через протокол SSH: з'єднання встановлюється швидко, надійно підтримується протягом усієї сесії та коректно завершується. Передача команд, таких як `ls`, `mkdir`, `rm -r`, `hostname`, виконується з передбачуваними результатами, що свідчить про надійність реалізованої логіки в класах `SSHManager` та `RemoteFileRepository`.

Функціональні операції з файлами (створення папок, видалення, копіювання, переміщення, перейменування) були протестовані у різних сценаріях, включаючи обробку типових помилок — відсутність з'єднання, неправильні команди, заборонені дії. Додаток виявив стійкість до виняткових ситуацій і коректно інформував користувача про проблеми.

Оцінка UX-параметрів підтвердила інтуїтивність інтерфейсу, відповідність стилю `Material Design` та зручність роботи навіть на пристроях із середніми характеристиками. Система подій, станів та `ViewModel` забезпечує чіткий зв'язок між діями користувача і реакцією інтерфейсу, що покращує загальний досвід використання.

Таким чином, реалізований застосунок демонструє відповідність поставленим функціональним вимогам, є стабільним, продуктивним та зручним у щоденному використанні.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

ВИСНОВКИ

У рамках цієї кваліфікаційної роботи було ефективно реалізовано повнофункціональний мобільний застосунок, призначений для системного адміністрування віддалених Linux-серверів за допомогою захищеного протоколу SSH. Проєкт охопив усі ключові етапи життєвого циклу розробки: від чіткого визначення технічного завдання до реалізації архітектурних рішень, безпосереднього впровадження заявленого функціоналу та подальшого всебічного тестування застосунку в реальному тестовому середовищі.

Під час виконання роботи було проведено комплексний аналіз переваг, які надає використання протоколу SSH для завдань мобільного адміністрування, підкреслюючи його безпеку та гнучкість. Досконало досліджено можливості платформи Android, зокрема, аспекти, що стосуються розробки користувацьких інтерфейсів та оптимізації продуктивності. Значну увагу було приділено застосуванню архітектурного шаблону MVVM (Model-View-ViewModel), який забезпечив чітке розділення відповідальності компонентів та сприяв підвищенню модульності та підтримуваності коду. Практична частина роботи включала успішну реалізацію механізмів підключення до віддалених серверів, повноцінної взаємодії з файловою системою, виконання довільних SSH-команд та персистентного збереження даних користувача за допомогою бібліотеки Room.

Для проведення тестування було розгорнуто спеціалізоване серверне середовище у віртуальних машинах VirtualBox, використовуючи операційні системи Ubuntu та Debian. Важливою складовою цього середовища стала інсталяція та налаштування сервісу avahi-daemon, що забезпечило коректний резолвінг доменних імен у локальній мережі. Це рішення дозволило проводити комфортне та ефективне тестування без необхідності використання прямих IP-адрес, що значно спростило процес взаємодії.

Всі цілі, поставлені на початковому етапі кваліфікаційної роботи, були повністю досягнуті. Розроблений мобільний файловий менеджер відповідає сучасним вимогам, що висуваються до рішень для системного адміністрування, забезпечуючи високий рівень безпечного доступу, стабільність функціонування,

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

зручність використання та архітектурну масштабованість. Запропоноване рішення є надійною основою, яка може бути використана для подальшого розширення функціоналу. Перспективи розвитку включають, зокрема, реалізацію авторизації за допомогою SSH-ключів, підтримку одночасної роботи з кількома віддаленими серверами та інтеграцію з іншими системами адміністрування, що дозволить створити більш комплексний інструмент для адміністраторів.

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. SSH Protocol Explained [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.ssh.com/academy/ssh/protocol>
2. Linux Server SSH Connection [Електронний ресурс] – Режим доступу до ресурсу:
<https://medium.com/@sari.mmusab/linux-server-ssh-connection-1cd3f19196e>
3. OpenSSH [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.openssh.com>
4. Are SSH and SFTP the Same? [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.goanywhere.com/blog/are-ssh-and-sftp-the-same>
5. Definition What is Telnet? [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.techtarget.com/searchnetworking/definition/Telnet>
6. What is FTP? [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.techtarget.com/searchnetworking/definition/File-Transfer-Protocol-FTP>
7. What is Remote Desktop Protocol (RDP)? [Електронний ресурс] – Режим доступу до ресурсу:
<https://segura.security/post/what-is-remote-desktop-protocol>
8. What Is a VPN? [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.paloaltonetworks.com/cyberpedia/what-is-a-vpn>
9. VDI vs VPN vs RDP: Choosing a Secure Remote Access Solution [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.esecurityplanet.com/endpoint/secure-access-for-remote-workers-rdp-vpn-vdi/>
10. SSHJ – Java SSH Library [Електронний ресурс] – Режим доступу до ресурсу:
<https://github.com/hierynomus/sshj>
11. Android Developers – Architecture Components [Електронний ресурс] – Режим доступу до ресурсу:
<https://developer.android.com/topic/libraries/architecture>
12. MVVM (Model View ViewModel) Architecture Pattern in Android [Електронний ресурс] – Режим доступу до ресурсу:

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

<https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/>

13. Android Developers – ViewModel [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.android.com/topic/libraries/architecture/viewmodel>
14. MVVM (Model View ViewModel) Architecture Pattern in Android [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.android.com/topic/libraries/architecture/viewmodel>
15. Enable mDNS to Ubuntu Server [Электронный ресурс] – Режим доступа до ресурсу: <https://www.thefanatic.com/enable-mdns-to-ubuntu-server/>
16. Oracle VirtualBox [Электронный ресурс] – Режим доступа до ресурсу: <https://www.virtualbox.org/>
17. Ubuntu 22.04 LTS [Электронный ресурс] – Режим доступа до ресурсу: <https://ubuntu.com/blog/tag/22-04-lts>
18. Downloading Debian [Электронный ресурс] – Режим доступа до ресурсу: <https://www.debian.org/download>
19. OpenSSH [Электронный ресурс] – Режим доступа до ресурсу: <https://www.openssh.com/>
20. Avahi Daemon Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://www.avahi.org>
21. Material Design [Электронный ресурс] – Режим доступа до ресурсу: <https://m3.material.io/>

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

ДОДАТКИ

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

Схема мережі

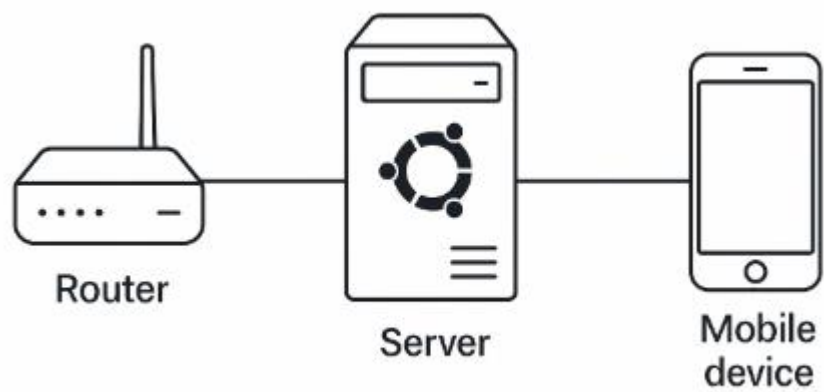
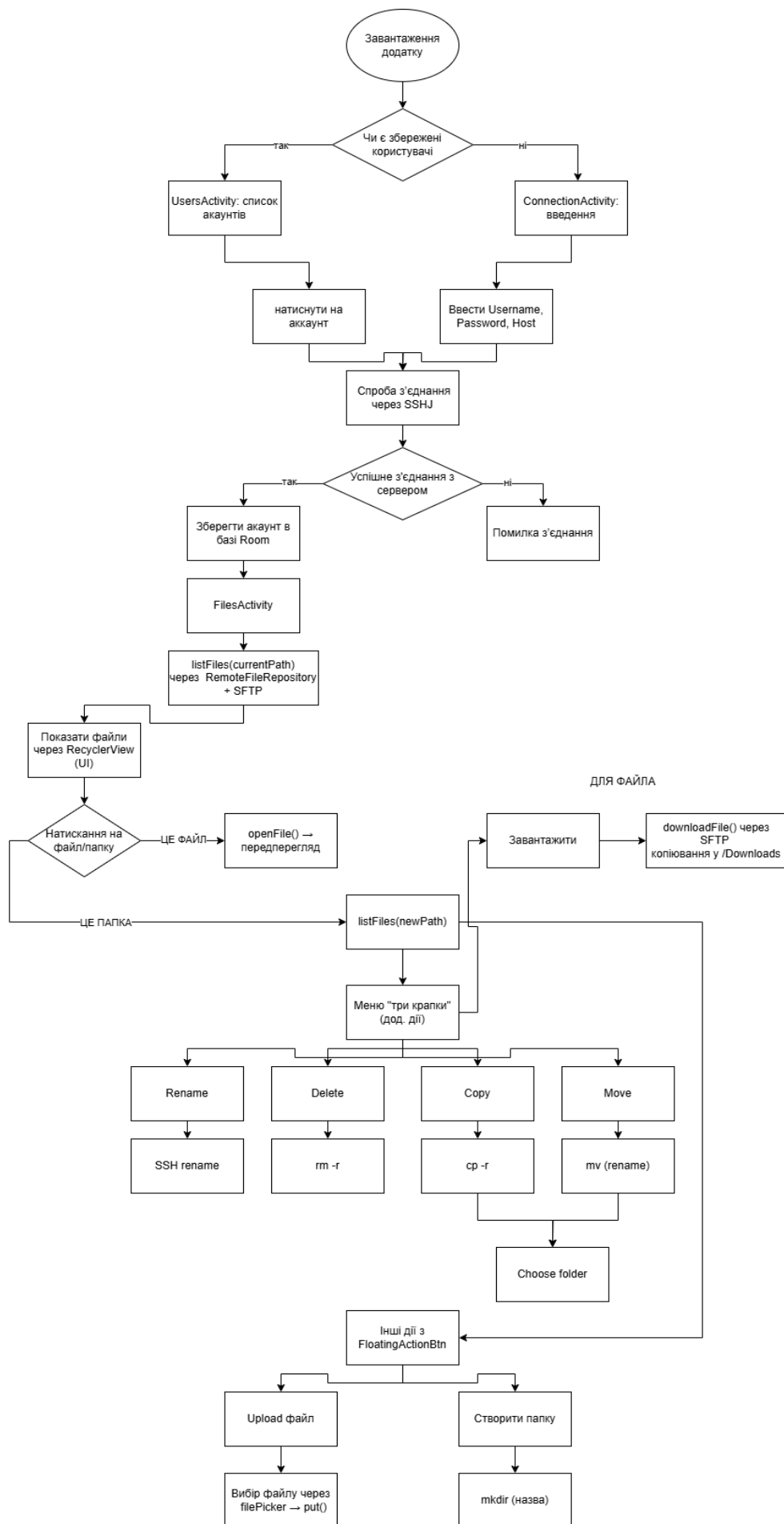


Схема роботи додатку:



Лістинги основних класів

З'єднання з сервером:

```

public class SSHManager {
    private static final int DEFAULT_PORT = 22;
    private static SSHManager instance;

    private SSHClient sshClient;
    private SFTPClient sftpClient;
    private RemoteUser currentUser;

    private SSHManager() {
    }

    public static SSHManager getInstance() {
        if (instance == null) {
            instance = new SSHManager();
        }
        return instance;
    }

    public void connect(RemoteUser user) throws IOException {
        Security.removeProvider(BouncyCastleProvider.PROVIDER_NAME);
        Security.insertProviderAt(new BouncyCastleProvider(), 1);

        SSHClient sshClient = new SSHClient();
        sshClient.addHostKeyVerifier(new PromiscuousVerifier());
        sshClient.getConnection().getKeepAlive().setKeepAliveInterval(5);
        sshClient.setConnectTimeout(5000);
        sshClient.connect(user.getHost(), DEFAULT_PORT);
        sshClient.authPassword(user.getUsername(), user.getPassword());

        this.sshClient = sshClient;
        this.sftpClient = sshClient.newSFTPClient();
        this.currentUser = user;
    }

    public void disconnect() throws IOException {
        try {
            if (sftpClient != null) {
                sftpClient.close();
            }
            if (sshClient != null && sshClient.isConnected()) {
                sshClient.disconnect();
            }
        } finally {
            sftpClient = null;
            sshClient = null;
        }
    }

    public SFTPClient getSFTPClient(RemoteUser user) throws IOException {
        connectIfRequired(user);
        return sftpClient;
    }

    public void executeCommand(RemoteUser user, String command) throws IOException {
        connectIfRequired(user);
        try (Session session = sshClient.startSession()) {
            Session.Command cmd = session.exec(command);
            cmd.join(5, TimeUnit.SECONDS);
        }
    }

    private void connectIfRequired(RemoteUser user) throws IOException {
        if (sshClient == null || !sshClient.isConnected()) {

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        connect(user);
    } else if (!user.equals(currentUser)) {
        disconnect();
        connect(user);
    }
}

@Entity(tableName = RemoteUser.TABLE_NAME)
public class RemoteUser {
    public static final String TABLE_NAME = "remote_users";
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    private final long id;
    @ColumnInfo(name = "username")
    private final String username;
    @ColumnInfo(name = "password")
    private final String password;
    @ColumnInfo(name = "host")
    private final String host;

    @Ignore
    public RemoteUser(String username, String password, String host) {
        this(0, username, password, host);
    }

    public RemoteUser(long id, String username, String password, String host) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.host = host;
    }

    public long getId() {
        return id;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getHost() {
        return host;
    }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof RemoteUser)) return false;
        RemoteUser that = (RemoteUser) o;
        return Objects.equals(username, that.username) && Objects.equals(password, that.password) &&
        Objects.equals(host, that.host);
    }

    @Override
    public int hashCode() {
        return Objects.hash(username, password, host);
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

Робота з файлами:

```

public class RemoteFileRepository {
    private final SSHManager sshManager;
    private final RemoteUserRepository remoteUserRepository;
    private final InternalFileRepository internalFileRepository;

    public RemoteFileRepository(Context applicationContext) {
        this.sshManager = SSHManager.getInstance();
        this.remoteUserRepository = RemoteUserRepository.getInstance(applicationContext);
        this.internalFileRepository = new InternalFileRepository(applicationContext);
    }

    public RemoteFile openFile(String fullRemotePath) throws Exception {
        RemoteUser user = getCurrentUser();
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        return sftpClient.open(fullRemotePath);
    }

    public List<RemoteResourceInfo> listFiles(String path) throws Exception {
        RemoteUser user = getCurrentUser();
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullRemotePath = getFullRemotePath(sftpClient, path);
        return sftpClient.ls(fullRemotePath)
            .stream()
            .filter(file -> !file.getName().equals(".") && !file.getName().equals(".."))
            .sorted(RemoteFileComparator())
            .collect(Collectors.toList());
    }

    public void renameFile(String oldPath, String newPath) throws Exception {
        RemoteUser user = getCurrentUser();
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullOldPath = getFullRemotePath(sftpClient, oldPath);
        String fullNewPath = getFullRemotePath(sftpClient, newPath);
        sftpClient.rename(fullOldPath, fullNewPath);
    }

    public void copyFile(RemoteResourceInfo file, String destinationPath) throws Exception {
        RemoteUser user = getCurrentUser();
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullDestinationPath = getFullRemotePath(sftpClient, destinationPath);
        String command = "cp -r '" + file.getPath() + "' '" + fullDestinationPath + "/" +
file.getName() + "'";
        sshManager.executeCommand(user, command);
    }

    public void moveFile(RemoteResourceInfo file, String destinationPath) throws Exception {
        RemoteUser user = getCurrentUser();
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullDestinationPath = getFullRemotePath(sftpClient, destinationPath);
        sftpClient.rename(file.getPath(), fullDestinationPath + "/" + file.getName());
    }

    public void deleteFile(RemoteResourceInfo file) throws Exception {
        RemoteUser user = getCurrentUser();
        sshManager.executeCommand(user, "rm -r '" + file.getPath() + "'");
    }

    public void createFolder(String path) throws Exception {
        RemoteUser user = getCurrentUser();
        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullRemotePath = getFullRemotePath(sftpClient, path);
        sftpClient.mkdir(fullRemotePath);
    }

    public void downloadFile(String remotePath, String localPath) throws Exception {
        RemoteUser user = getCurrentUser();
    }

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        SFTPClient sftpClient = sshManager.getSFTPClient(user);
        String fullRemotePath = getFullRemotePath(sftpClient, remotePath);
        sftpClient.get(fullRemotePath, localPath);
    }

    public void uploadFile(Uri uri, String remotePath) throws Exception {
        String localPath = null;
        try {
            RemoteUser user = getCurrentUser();
            localPath = internalFileRepository.saveFile(uri);
            SFTPClient sftpClient = sshManager.getSFTPClient(user);
            String fullRemotePath = getFullRemotePath(sftpClient, remotePath);
            sftpClient.put(localPath, fullRemotePath);
        } finally {
            if (localPath != null) {
                internalFileRepository.deleteFile(localPath);
            }
        }
    }

    private Comparator<RemoteResourceInfo> RemoteFileComparator() {
        return Comparator
            .comparing(RemoteResourceInfo::isDirectory)
            .reversed()
            .thenComparing(RemoteResourceInfo::getName);
    }

    private RemoteUser getCurrentUser() throws Exception {
        RemoteUser user = remoteUserRepository.getCurrentUser();
        if (user == null) {
            throw new Exception("No remote user configured");
        }
        return user;
    }

    private String getFullRemotePath(SFTPClient sftpClient, String path) throws IOException {
        String parent = sftpClient.canonicalize("~").replace("/~", "");
        return new File(parent, path).getPath();
    }
}

public class FilesViewModel extends BaseViewModel {
    private final RemoteFileRepository remoteFileRepository;
    private final MutableLiveData<FilesScreenState> stateLiveData;

    public FilesViewModel(RemoteFileRepository remoteFileRepository) {
        this.remoteFileRepository = remoteFileRepository;
        this.stateLiveData = new MutableLiveData<>(FilesScreenState.initial());
    }

    public LiveData<FilesScreenState> getStateLiveData() {
        return stateLiveData;
    }

    public void onCreate() {
        listFiles(getCurrentState().getPath());
    }

    public void onSwipeRefresh() {
        listFiles(getCurrentState().getPath());
    }

    public void onFileClick(RemoteResourceInfo file) {
        if (file.isDirectory()) {
            String path = new File(getCurrentState().getPath(), file.getName()).getPath();
            listFiles(path);
        } else {
            stateLiveData.setValue(getCurrentState().copyWithEvent(new OpenFile(file)));
        }
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    public void onMoreButtonClick(RemoteResourceInfo file) {
        stateLiveData.setValue(getCurrentState().copyWithEvent(new
ShowFileActionsBottomSheet(file)));
    }

    public void onDownloadButtonClick(RemoteResourceInfo file) {
        String path = getCurrentState().getPath();
        String remoteResourceInfoPath = new File(path, file.getName()).getPath();
        String localFilePath = new File(getDownloadsFolderPath(), file.getName()).getPath();
        async(
            () -> remoteFileRepository.downloadFile(remoteResourceInfoPath, localFilePath),
            () -> stateLiveData.setValue(getCurrentState().copyWithEvent(new
ShowSuccessDownloadMessage(file))),
            this::onError
        );
    }

    public void onRenameButtonClick(RemoteResourceInfo file) {
        stateLiveData.setValue(getCurrentState().copyWithEvent(new ShowRenameFileDialog(file)));
    }

    public void onRenameFileConfirmed(RemoteResourceInfo file, String newName) {
        String path = getCurrentState().getPath();
        String oldFilePath = new File(path, file.getName()).getPath();
        String newFilePath = new File(path, newName).getPath();
        stateLiveData.setValue(getCurrentState().copyWithLoading(true));
        async(
            () -> remoteFileRepository.renameFile(oldFilePath, newFilePath),
            () -> {
                FilesScreenState successState = getCurrentState()
                    .copyWithLoading(false)
                    .copyWithEvent(new ShowSuccessCopyMessage(file));
                listFiles(path);
                stateLiveData.setValue(successState);
            },
            this::onError
        );
    }

    public void onCopyButtonClick(RemoteResourceInfo file) {
        PendingOperationData pendingData = new PendingOperationData(file,
PendingOperationType.COPY);
        FilesScreenState newState = getCurrentState()
            .copyWithPendingOperationData(pendingData)
            .copyWithEvent(new ShowFolderChooser());
        stateLiveData.setValue(newState);
    }

    public void onMoveButtonClick(RemoteResourceInfo file) {
        PendingOperationData pendingData = new PendingOperationData(file,
PendingOperationType.MOVE);
        FilesScreenState newState = getCurrentState()
            .copyWithPendingOperationData(pendingData)
            .copyWithEvent(new ShowFolderChooser());
        stateLiveData.setValue(newState);
    }

    public void onFolderChosen(String folderPath) {
        FilesScreenState currentState = getCurrentState();
        PendingOperationData pendingData = currentState.getPendingOperationData();
        if (pendingData == null) {
            stateLiveData.setValue(currentState.copyWithEvent(new ShowErrorMessage()));
            return;
        }

        FilesScreenState startState = currentState
            .copyWithLoading(true)

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .copyWithPendingOperationData(null);
stateLiveData.setValue(startState);

if (pendingData.getType() == PendingOperationType.COPY) {
    async(
        () -> remoteFileRepository.copyFile(pendingData.getFile(), folderPath),
        () -> {
            FilesScreenState successState = getCurrentState()
                .copyWithLoading(false)
                .copyWithEvent(new ShowSuccessCopyMessage(pendingData.getFile()));
            stateLiveData.setValue(successState);
        },
        this::onError
    );

} else if (pendingData.getType() == PendingOperationType.MOVE) {
    async(
        () -> remoteFileRepository.moveFile(pendingData.getFile(), folderPath),
        () -> listFiles(currentState.getPath()),
        this::onError
    );

} else {
    throw new IllegalArgumentException("Unknown pending operation type: " +
pendingData.getType());
}

}

public void onDeleteButtonClick(RemoteResourceInfo file) {
    stateLiveData.setValue(getCurrentState().copyWithEvent(new ShowDeleteFileDialog(file)));
}

public void onDeleteFileConfirmed(RemoteResourceInfo file) {
    String path = getCurrentState().getPath();
    stateLiveData.setValue(getCurrentState().copyWithLoading(true));
    async(
        () -> remoteFileRepository.deleteFile(file),
        () -> listFiles(path),
        this::onError
    );
}

public void onCreateFolderButtonClick() {
    stateLiveData.setValue(getCurrentState().copyWithEvent(new ShowCreateFolderDialog()));
}

public void onCreateFolderConfirmed(String folderName) {
    String path = getCurrentState().getPath();
    String newFolderPath = new File(path, folderName).getPath();
    stateLiveData.setValue(getCurrentState().copyWithLoading(true));
    async(
        () -> remoteFileRepository.createFolder(newFolderPath),
        () -> listFiles(path),
        this::onError
    );
}

public void onUploadFileClick() {
    stateLiveData.setValue(getCurrentState().copyWithEvent(new ShowFilePicker()));
}

public void onFileForUploadPicked(Uri uri) {
    String path = getCurrentState().getPath();
    async(
        () -> remoteFileRepository.uploadFile(uri, path),
        () -> listFiles(path),
        this::onError
    );
}
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public void onChooseButtonClick() {
    String path = getCurrentState().getPath();
    stateLiveData.setValue(getCurrentState().copyWithEvent(new FinishWithChosenPath(path)));
}

public void onEventHandled() {
    stateLiveData.setValue(getCurrentState().copyWithEvent(null));
}

public void onBackPressed() {
    FilesScreenState currentState = getCurrentState();
    if (!currentState.isRoot()) {
        listFiles(new File(currentState.getPath()).getParent());
    }
}

private void listFiles(String path) {
    FilesScreenState currentState = getCurrentState();
    boolean clearFiles = !currentState.getPath().equals(path);
    FilesScreenState startState = getCurrentState()
        .copyWithPath(path)
        .copyWithLoading(true);
    if (clearFiles) {
        startState = startState.copyWithFiles(new ArrayList<>());
    }
    stateLiveData.setValue(startState);
    async(
        () -> remoteFileRepository.listFiles(path),
        files -> {
            FilesScreenState successState = getCurrentState()
                .copyWithLoading(false)
                .copyWithFiles(files);
            stateLiveData.setValue(successState);
        },
        this::onError
    );
}

private FilesScreenState getCurrentState() {
    return stateLiveData.getValue();
}

private void onError(Throwable error) {
    FilesScreenState newState = getCurrentState()
        .copyWithLoading(false)
        .copyWithEvent(new ShowErrorMessage());
    stateLiveData.setValue(newState);
}

private String getDownloadsFolderPath() {
    return
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS).getAbsolutePath();
}

public static class Factory implements ViewModelProvider.Factory {
    private final RemoteFileRepository remoteFileRepository;

    public Factory(Context context) {
        this.remoteFileRepository = new RemoteFileRepository(context.getApplicationContext());
    }

    @NonNull
    @Override
    @SuppressWarnings("unchecked")
    public <T extends ViewModel> T create(Class<T> modelClass) {
        if (modelClass.isAssignableFrom(FilesViewModel.class)) {
            return (T) new FilesViewModel(remoteFileRepository);
        }
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        throw new IllegalArgumentException("Unknown ViewModel class");
    }
}

public class FilesAdapter extends RecyclerView.Adapter<FilesAdapter.ViewHolder> {
    private final FilesViewModel viewModel;
    private final List<RemoteResourceInfo> items = new ArrayList<>();
    private final boolean showMoreButton;

    public FilesAdapter(FilesViewModel viewModel, boolean showMoreButton) {
        this.viewModel = viewModel;
        this.showMoreButton = showMoreButton;
    }

    public void setItems(List<RemoteResourceInfo> items) {
        this.items.clear();
        this.items.addAll(items);
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        ItemFileBinding binding = ItemFileBinding.inflate(inflater, parent, false);
        return new ViewHolder(binding);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        RemoteResourceInfo item = items.get(position);
        holder.binding.icon.setImageResource(item.isDirectory() ? R.drawable.ic_folder :
R.drawable.ic_file);
        holder.binding.name.setText(item.getName());
        holder.binding.moreButton.setVisibility(showMoreButton ? View.VISIBLE : View.GONE);

        Context context = holder.binding.getRoot().getContext();
        String modificationDate = DateUtils.formatDateTime(
            context,
            item.getAttributes().getMtime() * 1000,
            DateUtils.FORMAT_SHOW_DATE
        );
        if (item.isDirectory()) {
            holder.binding.details.setText(modificationDate);
        } else {
            String size = Formatter.formatFileSize(context, item.getAttributes().getSize());
            holder.binding.details.setText(String.format("%s • %s", size, modificationDate));
        }

        holder.binding.getRoot().setOnClickListener(__ -> viewModel.onFileClick(item));
        holder.binding.moreButton.setOnClickListener(__ -> viewModel.onMoreButtonClick(item));
    }

    @Override
    public int getItemCount() {
        return items.size();
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        private final ItemFileBinding binding;

        public ViewHolder(ItemFileBinding binding) {
            super(binding.getRoot());
            this.binding = binding;
        }
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

Користувачі:

```
@Dao
public interface RemoteUserDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    long save(RemoteUser user); // Returns the ID of the saved user

    @Query("SELECT * FROM " + RemoteUser.TABLE_NAME + " ORDER BY username ASC")
    List<RemoteUser> getAll(); // Returns all users ordered by username

    @Query("SELECT * FROM " + RemoteUser.TABLE_NAME + " WHERE id = :id LIMIT 1")
    RemoteUser get(long id); // Returns a user by ID

    @Delete
    void delete(RemoteUser user); // Deletes a user from the database
}

@Database(entities = {RemoteUser.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    private static final String DB_NAME = "app_database.db";
    private static AppDatabase instance;

    public static AppDatabase getInstance(Context applicationContext) {
        if (instance == null) {
            instance = Room
                .databaseBuilder(applicationContext, AppDatabase.class, DB_NAME)
                .build();
        }
        return instance;
    }

    public abstract RemoteUserDao remoteUserDao();
}

public class RemoteUserRepository {
    private static RemoteUserRepository instance;
    private final RemoteUserDao remoteUserDao;
    private long currentUserId = -1;

    public static RemoteUserRepository getInstance(Context applicationContext) {
        if (instance == null) {
            instance = new RemoteUserRepository(applicationContext);
        }
        return instance;
    }

    private RemoteUserRepository(Context applicationContext) {
        this.remoteUserDao = AppDatabase.getInstance(applicationContext).remoteUserDao();
    }

    public RemoteUser getUser(long id) {
        return remoteUserDao.get(id);
    }

    public void saveUser(RemoteUser user) {
        remoteUserDao.save(user);
    }

    public void deleteUser(RemoteUser user) {
        remoteUserDao.delete(user);
    }

    @Nullable
    public RemoteUser getCurrentUser() {
        if (currentUserId != -1) {
            return remoteUserDao.get(currentUserId);
        } else {
            return null;
        }
    }
}
```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    public void saveCurrentUser(RemoteUser user) {
        currentUserId = remoteUserDao.save(user);
    }

    public List<RemoteUser> getAllUsers() {
        return remoteUserDao.getAll();
    }

    public void setCurrentUserId(long userId) {
        this.currentUserId = userId;
    }

    public void clearCurrentUserId() {
        this.currentUserId = -1;
    }
}

public class UsersViewModel extends BaseViewModel {
    private final RemoteUserRepository remoteUserRepository;

    private final MutableLiveData<UsersScreenState> stateLiveData;

    public UsersViewModel(RemoteUserRepository remoteUserRepository) {
        this.remoteUserRepository = remoteUserRepository;
        this.stateLiveData = new MutableLiveData<>(UsersScreenState.initial());
    }

    public LiveData<UsersScreenState> getStateLiveData() {
        return stateLiveData;
    }

    public void onCreate() {
        loadUsers();
    }

    public void onUserClick(RemoteUser user) {
        remoteUserRepository.setCurrentUserId(user.getId());
        stateLiveData.setValue(getCurrentState().copyWithEvent(new ShowConnectionScreen()));
    }

    public void onDeleteButtonClick(RemoteUser user) {
        stateLiveData.setValue(getCurrentState().copyWithLoading(true));
        async(
            () -> remoteUserRepository.deleteUser(user),
            this::loadUsers,
            this::onUnknownError
        );
    }

    public void onAddButtonClick() {
        remoteUserRepository.clearCurrentUserId();
        stateLiveData.setValue(getCurrentState().copyWithEvent(new ShowConnectionScreen()));
    }

    public void onEventHandled() {
        stateLiveData.setValue(getCurrentState().copyWithEvent(null));
    }

    private void loadUsers() {
        stateLiveData.setValue(getCurrentState().copyWithLoading(true));
        async(
            remoteUserRepository::getAllUsers,
            this::onLoadUsersSuccess,
            this::onUnknownError
        );
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

private void onLoadUsersSuccess(List<RemoteUser> users) {
    UsersScreenState newState = getCurrentState()
        .copyWithLoading(false)
        .copyWithUsers(users);
    stateLiveData.setValue(newState);
}

private void onUnknownError(Throwable error) {
    UsersScreenState newState = getCurrentState()
        .copyWithLoading(false)
        .copyWithEvent(new ShowUnknownErrorMessage());
    stateLiveData.setValue(newState);
}

private UsersScreenState getCurrentState() {
    return stateLiveData.getValue();
}

public static class Factory implements ViewModelProvider.Factory {
    private final RemoteUserRepository remoteUserRepository;

    public Factory(Context context) {
        this.remoteUserRepository =
RemoteUserRepository.getInstance(context.getApplicationContext());
    }

    @NonNull
    @Override
    @SuppressWarnings("unchecked")
    public <T extends androidx.lifecycle.ViewModel> T create(Class<T> modelClass) {
        if (modelClass.isAssignableFrom(UsersViewModel.class)) {
            return (T) new UsersViewModel(remoteUserRepository);
        }
        throw new IllegalArgumentException("Unknown ViewModel class");
    }
}

public class UsersAdapter extends RecyclerView.Adapter<UsersAdapter.ViewHolder> {
    private final UsersViewModel viewModel;
    private final List<RemoteUser> items = new ArrayList<>();

    public UsersAdapter(UsersViewModel viewModel) {
        this.viewModel = viewModel;
    }

    public void setItems(List<RemoteUser> items) {
        this.items.clear();
        this.items.addAll(items);
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        ItemUserBinding binding = ItemUserBinding.inflate(inflater, parent, false);
        return new ViewHolder(binding);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        RemoteUser item = items.get(position);
        holder.binding.username.setText(item.getUsername());
        holder.binding.host.setText(item.getHost());
        holder.binding.getRoot().setOnClickListener(__ -> viewModel.onUserClick(item));
        holder.binding.deleteButton.setOnClickListener(__ -> viewModel.onDeleteButtonClick(item));
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@Override
public int getItemCount() {
    return items.size();
}

public static class ViewHolder extends RecyclerView.ViewHolder {
    private final ItemUserBinding binding;

    public ViewHolder(ItemUserBinding binding) {
        super(binding.getRoot());
        this.binding = binding;
    }
}
}

```

Інтерфейс і логіка підключення

```

public class ConnectionViewModel extends BaseViewModel {
    private final RemoteUserRepository remoteUserRepository;
    private final SSHManager sshManager;

    private final MutableLiveData<ConnectionScreenState> stateLiveData;
    private RemoteUser savedUser;

    private boolean shouldShowSavedUser = true;

    public ConnectionViewModel(RemoteUserRepository remoteUserRepository) {
        this.remoteUserRepository = remoteUserRepository;
        this.sshManager = SSHManager.getInstance();
        this.stateLiveData = new MutableLiveData<>(ConnectionScreenState.initial());
    }

    public LiveData<ConnectionScreenState> getStateLiveData() {
        return stateLiveData;
    }

    public void onCreate() {
        if (shouldShowSavedUser) {
            shouldShowSavedUser = false;
            async(
                remoteUserRepository::getCurrentUser,
                this::onCurrentUserLoaded,
                __ -> { /*do nothing*/ }
            );
        }
    }

    private void onCurrentUserLoaded(RemoteUser savedUser) {
        this.savedUser = savedUser;
        if (savedUser != null) {
            ConnectionScreenEvent event = new ShowSavedUser(savedUser);
            stateLiveData.setValue(ConnectionScreenState.event(event));
        }
    }

    public void onConnectButtonClick(String username, String password, String host) {
        if (stateLiveData.getValue().isConnecting()) {
            return;
        }
        if (username.isEmpty() || password.isEmpty() || host.isEmpty()) {
            ConnectionScreenEvent event = new ShowErrorMessage("All fields are required");
            stateLiveData.setValue(ConnectionScreenState.event(event));
        } else {
            RemoteUser user;
            if (savedUser != null) {
                user = new RemoteUser(savedUser.getId(), username, password, host);
            } else {
                user = new RemoteUser(username, password, host);
            }
        }
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    new RemoteUser(username, password, host);
    stateLiveData.setValue(ConnectionScreenState.connecting());
    async(
        () -> sshManager.connect(user),
        () -> onConnectionSuccess(user),
        this::onConnectionError
    );
}

private void onConnectionSuccess(RemoteUser user) {
    async(
        () -> remoteUserRepository.saveCurrentUser(user),
        this::onConnectionSuccess,
        this::onConnectionError
    );
}

private void onConnectionSuccess() {
    ConnectionScreenEvent event = new GoToFilesScreen();
    stateLiveData.setValue(ConnectionScreenState.event(event));
}

private void onConnectionError(Throwable error) {
    ConnectionScreenEvent event = new ShowErrorMessage("Connection error");
    stateLiveData.setValue(ConnectionScreenState.event(event));
}

public void onEventHandled() {
    stateLiveData.setValue(ConnectionScreenState.initial());
}

public static class Factory implements ViewModelProvider.Factory {
    private final RemoteUserRepository remoteUserRepository;

    public Factory(Context context) {
        this.remoteUserRepository =
RemoteUserRepository.getInstance(context.getApplicationContext());
    }

    @NonNull
    @Override
    @SuppressWarnings("unchecked")
    public <T extends androidx.lifecycle.ViewModel> T create(Class<T> modelClass) {
        if (modelClass.isAssignableFrom(ConnectionViewModel.class)) {
            return (T) new ConnectionViewModel(remoteUserRepository);
        }
        throw new IllegalArgumentException("Unknown ViewModel class");
    }
}

public class PendingOperationData {
    private final RemoteResourceInfo file;
    private final PendingOperationType type;

    public PendingOperationData(RemoteResourceInfo file, PendingOperationType type) {
        this.file = file;
        this.type = type;
    }

    public RemoteResourceInfo getFile() {
        return file;
    }

    public PendingOperationType getType() {
        return type;
    }
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@Override
public boolean equals(Object o) {
    if (!(o instanceof PendingOperationData)) return false;
    PendingOperationData that = (PendingOperationData) o;
    return Objects.equals(file, that.file) && type == that.type;
}

@Override
public int hashCode() {
    return Objects.hash(file, type);
}
}

public class ConnectionActivity extends AppCompatActivity {
    private ActivityConnectionBinding binding;
    private ConnectionViewModel viewModel;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityConnectionBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        ViewCompat.setOnApplyWindowInsetsListener(binding.getRoot(), (v, insets) -> {
            Insets systemBars = insets.getInsets(Type.systemBars() | Type.displayCutout());
            v.setPadding(systemBars.left, 0, systemBars.right, systemBars.bottom);
            int titlePadding = (int) getResources().getDimension(R.dimen.screen_title_padding);
            binding.title.setPadding(titlePadding, titlePadding + systemBars.top, titlePadding,
titlePadding);
            return insets;
        });

        ConnectionViewModel.Factory viewModelFactory = new ConnectionViewModel.Factory(this);
        viewModel = new ViewModelProvider(this, viewModelFactory).get(ConnectionViewModel.class);
        viewModel.getStateLiveData().observe(this, this::renderState);
        viewModel.onCreate();

        binding.connectButton.setOnClickListener(v -> {
            String username = binding.username.getText().toString();
            String password = binding.password.getText().toString();
            String host = binding.host.getText().toString();
            viewModel.onConnectButtonClick(username, password, host);
        });
    }

    private void renderState(ConnectionScreenState state) {
        binding.progress.setVisibility(state.isConnecting() ? View.VISIBLE : View.GONE);

        if (state.getEvent() != null) {
            handleEvent(state.getEvent());
            viewModel.onEventHandled();
        }
    }

    private void handleEvent(ConnectionScreenEvent event) {
        if (event instanceof ShowSavedUser) {
            RemoteUser user = ((ShowSavedUser) event).getUser();
            binding.username.setText(user.getUsername());
            binding.password.setText(user.getPassword());
            binding.host.setText(user.getHost());

        } else if (event instanceof ShowErrorMessage) {
            String message = ((ShowErrorMessage) event).getMessage();
            Toast.makeText(this, message, Toast.LENGTH_SHORT).show();

        } else if (event instanceof GoToFilesScreen) {
            Intent intent = FilesActivity.createIntent(this);

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        startActivity(intent);
        finishAffinity();

    } else {
        throw new IllegalArgumentException("Unknown event type: " +
event.getClass().getSimpleName());
    }
}
}

```

		Драчевський Н.В.			ККІКБ.КР.Б – 123 – 03 – ПЗ	Арк.
		Бродський Ю.Б.				106
Змн.	Арк.	№ докум.	Підпис	Дата		