

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМИ ФАЙЛОВОГО МЕНЕДЖЕРА ДЛЯ
ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID. FILE MANAGER DEVELOPMENT
FOR ANDROID OS.

Виконав: студент(ка) 4 курсу, групи КНТ-227
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

Бурдакова О.С.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” _____ 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Бурдакової Олександри Сергіївни (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програми файлового менеджера для операційної системи Android. File Manager Development for Android OS.
 керівник проєкту (роботи) доц., к.т.н. Сердюк Сергій Микитович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “30” березня 2021 року
 № 103

2. Строк подання студентом проєкту (роботи) травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань роботи. 2. Матеріали та методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. Додатки.

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

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання 28 лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8–9 тижні	Розділ 5, Додатки
7	Захист роботи.	10 тиждень	

Студент(ка)

(підпис)

Бурдакова О.С.

(прізвище та ініціали)

Керівник проекту (роботи)

(підпис)

Сердюк С.М.

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
96 с., 2 табл., 30 рис., 3 дод., 20 джерел.

ОПЕРАЦІЙНА СИСТЕМА, ANDROID, ПРИСТРІЙ, ФАЙЛ, ФАЙЛОВА СИСТЕМА, ФАЙЛОВИЙ МЕНЕДЖЕР, ANDROID STUDIO, JAVA.

Об'єкт дослідження – процеси, що забезпечують роботу з файловою системою операційної системи (ОС) Android.

Предмет дослідження – методи розробки файлових менеджерів в ОС Android.

Мета роботи – розробка програмного забезпечення утиліти «Файловий менеджер» для основних файлових систем ОС Android.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування; файлові системи Ext2, Ext3, Ext4; мова програмування Java та середовище розробки AndroidStudio; мобільний пристрій під управлінням ОС Android.

Результати. Розроблено застосунок, що виконує наступні функції:

- приймає вхідні дані, які являють собою натискання на іконки каталогів або файлів, з якими працює користувач;
- видає вихідні дані у вигляді інформації про файл;
- виконує зміну імені файлу та його місця розташування, а також операцію вилучення файлу.

Програма створена, для користувачів мобільних пристроїв з ОС Android, яким потрібен простий та зручний файловий менеджер.

Галузь використання – файлова система ОС Android.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 96 pages, 30 figures, 2 tables, 3 appendixes, 20 sources.

OPERATING SYSTEM, ANDROID, DEVICE, FILE, FILE SYSTEM, FILE MANAGER, ANDROID STUDIO, JAVA.

Object of study is a processes providing work with the file system of the Android operating system (OS).

Subject of study are methods of developing of file managers in the Android OS.

The purpose of the work is development of software for the "File Manager" utility for the main file systems of the Android OS.

Materials, methods and technical means: object-oriented programming; file systems Ext2, Ext3, Ext4; Java programming language and AndroidStudio development environment; mobile device running Android OS.

Results. An application has been developed that performs the following functions:

- accepts input, which are clicks on the icons of directories or files that the user is working with;
- gives out initial data in the form of information about the file;
- performs the change of the file name and location, as well as the operation of deleting the file.

The program was created for users of mobile devices with Android OS who need a simple and convenient file manager.

The field of use is the Android OS file system.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз проблеми та постановка завдань роботи	11
1.1 Огляд особливостей операційної системи Android.....	11
1.2 Огляд основних файлових систем, що використовуються в операційній системі Android.....	15
1.3 Огляд програмних аналогів.....	20
2 Матеріали та методи.....	28
2.1 Обґрунтування вибору мови програмування та середовища розробки	28
2.2 Аналіз технічного завдання.....	35
3 Основні рішення щодо реалізації компонентів системи	40
3.1 Розробка загального алгоритму роботи програми	40
3.2 Розробка програми застосунку	42
4 Керівництво програміста.....	49
4.1 Призначення і умови застосування розробленої програми	49
4.2 Характеристики програми.....	49
4.3 Звернення до програми.....	52
4.4 Вхідні та вихідні дані	53
4.5 Повідомлення	53
5 Керівництво оператора.....	54
5.1 Призначення програми	54
5.2 Умови використання програми	54
5.3 Виконання програми	54
5.4 Повідомлення оператору	61
Висновки	63
Перелік джерел посилання.....	64
Додаток А Технічне завдання	66

Додаток Б Текст програми	71
Додаток В Копії слайдів презентації	88

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ГГц	– гігагерц;
Кб	– кілобайт;
Мб	– мегабайт;
ОС	– операційна система;
ПК	– персональний комп'ютер;
ФС	– файлова система.

ВСТУП

Файлові менеджери це інтерфейсні програми, що спрощують і прискорюють роботу користувача з файловими системами ОС. Вони дозволяють як створювати, так і видаляти файли, редагувати, копіювати, переміщати і виконувати багато інших операцій з файлами та каталогами.

Жодна операційна система не може обійтися без зручного та надійного файлового менеджера. На персональному комп'ютері (ПК) все досить просто – для цього є застосунок «Провідник», який зрозумілий на інтуїтивному рівні, або інші популярні програми, наприклад – Total Commander. З Android-системою ситуація дещо складніше: універсального "Провідника" в ОС Android немає.

За замовчуванням Google не оснащує прилади під управлінням Android повноцінним файловим менеджером, оскільки це суперечить політиці компанії [1]. Деякі виробники попередньо встановлюють свої власні більш потужні файлові менеджери на пристрої Android. В інших випадках користувачеві може знадобитися сторонній застосунок, щоб дійсно забезпечити роботу з файлами на своєму телефоні.

Також, слід зауважити, що до деяких системних файлів та папок звичайний користувач не має доступу: для цього необхідні su-права, тобто права адміністратора. Без цих прав користувачі Android мають доступ тільки до розділу даних, і це те, що вони будуть бачити при підключенні телефону до ПК або перегляді через файловий менеджер. Карта пам'яті microSD, якщо вона використовується, також буде відображатися змонтованою в цьому доступному користувачеві розділі даних, до якого він також може отримати доступ через ПК чи файловий менеджер. Саме тому, створення файлового менеджера для ОС Android є актуальним завданням.

Файловий менеджер Android може бути потужним інструментом для впорядкування інформації у мобільному телефоні користувача, але організаційна структура Android або її позірна відсутність можуть бути трохи

складними для звичайного браузера. Доступ до даних застосунку, зображень, музики і багато чому іншому здійснюється через єдину початкову папку, і це, безумовно, інший підхід до ієрархії, до якої будуть звикати користувачі персональних і Mac комп'ютерів, і набагато більш потужний, ніж той, що є у користувачів iOS.

Виходячи з вищезазначеного, у даній дипломній кваліфікаційній роботі бакалавра розробляється кросплатформна програма, що буде виконувати функції файлового менеджера.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Огляд особливостей операційної системи Android

Android – це операційна система для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux. Це порівняно молода операційна система, яка використовується в широкому колі мобільних пристроїв.

Згідно даних [2], станом на січень 2021 року ОС Android є найбільш поширеною у світі, випереджаючи ОС Windows майже на 10% (рис.1.1).

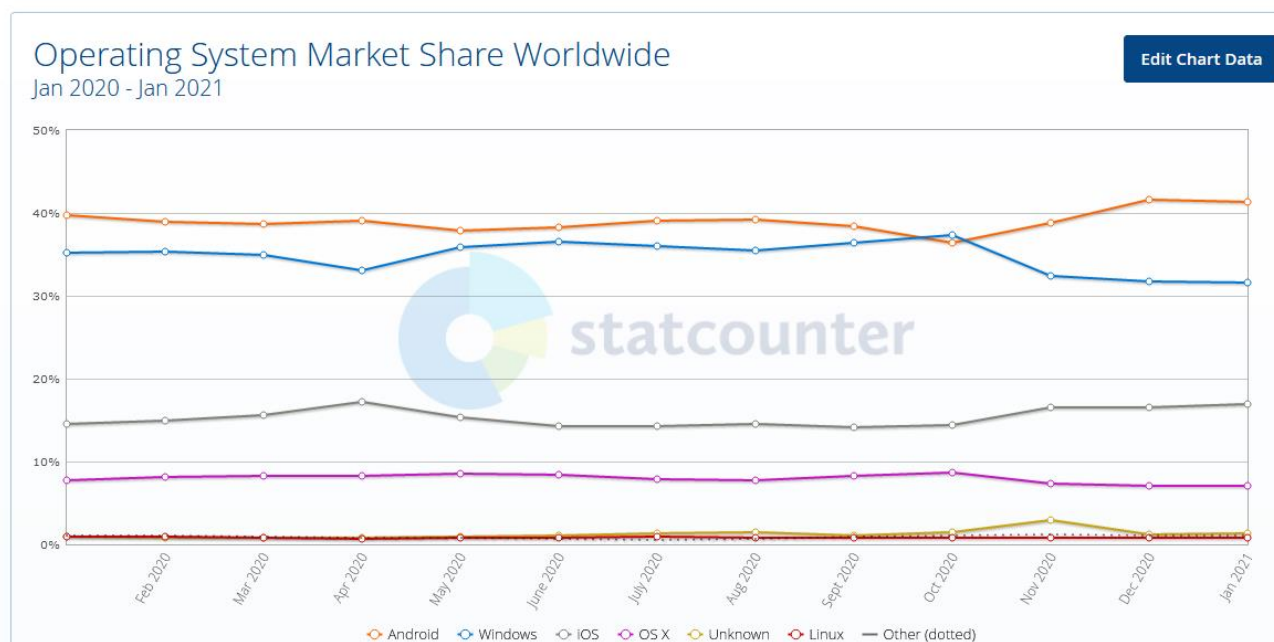


Рисунок 1.1 – Ринок операційних систем

Android дозволяє створювати Java-застосунки, які керують пристроєм через посередництво розроблених Google бібліотек. За допомогою Android Native Development Kit можна створювати застосунки, які написані на C, C++ або на мові Assembler. Станом на січень 2021 року у 71,93% смартфонів у

всьому світі встановлена ОС Android (рис. 1.2) [3].

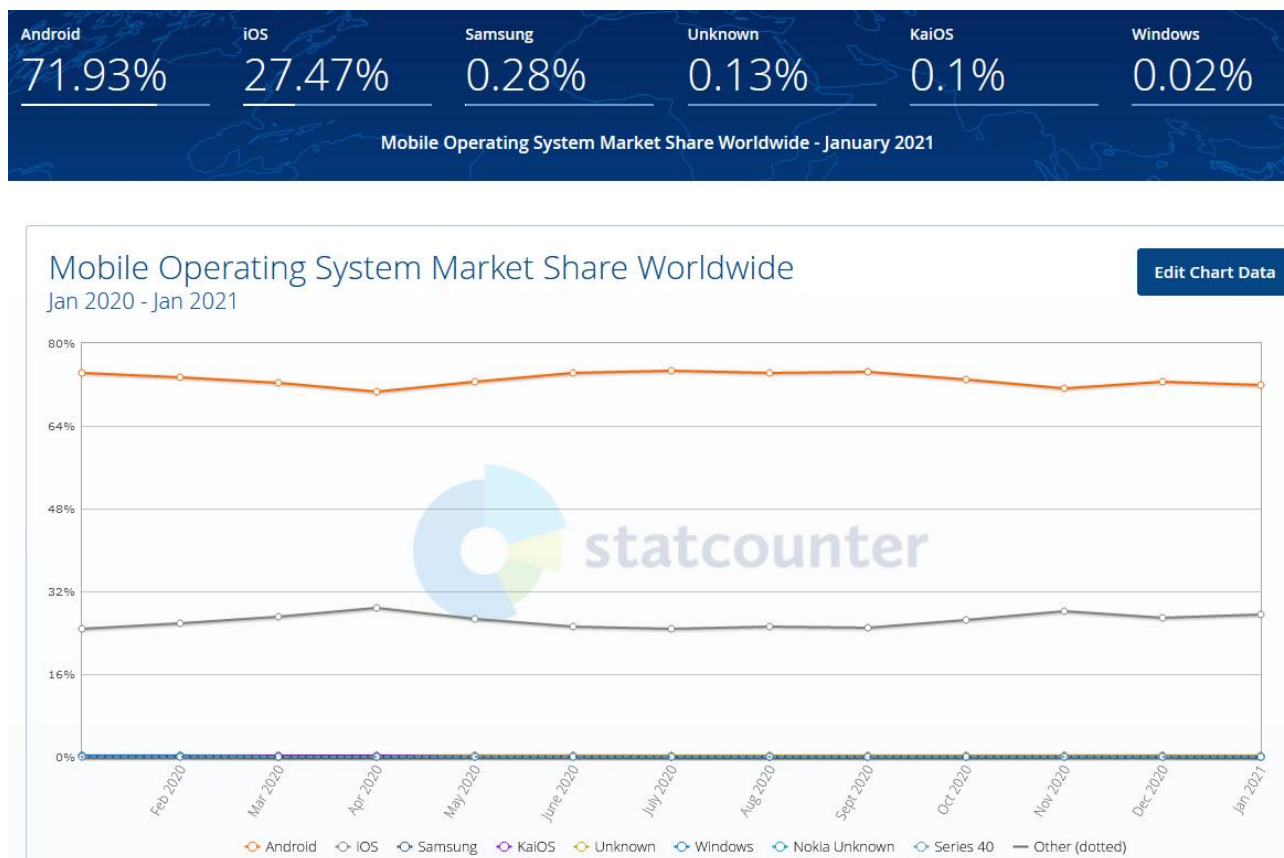


Рисунок 1.2 – Використання операційних систем у смартфонах

Під управлінням Android працюють як мобільні пристрої (найбільш розповсюджений клас), так і планшетні ПК, нетбуки або смартбуки. Також виробники не припиняють експериментувати та створюють інші гаджети під керуванням ОС Android: наприклад годинники або TV-приставки.

Операційну систему розробила компанія Android Inc., яка потім була придбана Google і переведена в ОНА – Open Handset Alliance, об'єднання, що займається розробкою і впровадженням відкритих мобільних стандартів. Крім Google, в ОНА входять такі гіганти як HTC, Intel, Samsung, Nvidia та інші.

Незважаючи на те що ОС Android базується на ядрі Linux, у ній використовуються далеко не всі можливості цієї операційної системи. Причиною тому є використання віртуальної машини Dalvik, в якій працює все програмне забезпечення ОС Android. Але з релізом Android Native Development

Kit у розробників з'явилась можливість створювати native-додатки на мові C та інших мовах програмування.

Перша версія Android була представлена в вересні 2008 року і тільки для смартфона T-Mobile, до листопада 2018 було випущено 40 оновлень версій цієї системи.

Операційна система Android використовує ядро ОС Linux тому вона, як і Linux, в тій чи іншій мірі підтримує велику кількість різноманітних файлових систем – від тих, що використовувались в Windows (тобто FAT і NTFS) і у використовуваних в відкритій POSIX-сумісній ОС Darwin файловій системі HFS+ і сучасній APFS, до мережевої 9pfs з Plan 9. Також є багато файлових систем притаманних суто ОС Linux, наприклад, Btrfs і сімейство ext (Ext2, Ext3, Ext4).

Як і в інших Linux-системах, ядро Linux забезпечує такі низькорівневі речі, як управління пам'яттю, захист даних, підтримку мультипроцесорності і багатопоточності. Але (за кількома винятками) користувач не знайде в ОС Android інших звичних компонентів GNU/Linux-систем. В Android відсутні компоненти проєкту GNU (вільної UNIX-подібної операційної системи), не використовується X.Org Server, а також системний менеджер Linux – systemd. Всі ці компоненти замінені аналогами, більш пристосованими для використання в умовах обмеженої пам'яті, низькій швидкості процесора і мінімального споживання енергії. Таким чином, Android більше схожий на вбудовану Linux-систему, ніж на GNU/Linux [4].

Таким чином, оскільки Android – це операційна система на основі Linux, мобільний телефон має структуру файлової системи в стилі Linux. У цій системі на кожному пристрої є шість основних розділів: завантажувальний (boot); системний (system); розділ для відновлення (recovery); розділ даних (data); кеша (cache) та іншого (misc). Карти пам'яті MicroSD також вважаються окремим розділом пам'яті. Пристрої, які поставляються з Android 7.0 Nougat, отримали можливість безперервного оновлення за рахунок того, що одночасно з системним розділом створюється другий і один з них оновлюється в

фоновому режимі, а при перезавантаженні відбувається перемикання, що дозволяє запрацювати оновленій системі.

Наведемо короткий опис того, що міститься в кожному розділі [5]:

- boot, цей розділ містить ядро, віртуальний накопичувач ramdisk і т. д., які необхідні для завантаження телефону при включенні;
- system, системний розділ містить файли операційної системи (також відомі як образ системи), які включають призначений для користувача інтерфейс Android і встановлені додатки. Цей розділ за замовчуванням монтується в режимі read-only (рис.1.3), а дані на ньому зазвичай змінюються тільки при оновленні системи;

```
# mount | grep /system
/dev/block/mmcblk0p14 on /system type ext4 (ro,seclabel,relatime,data=ordered)
```

Рисунок 1.3 – Приклад монтування розділу system

- recovery, альтернативний варіант завантаження ОС, програма відновлення дозволяє користувачеві створювати резервні копії (бекапи інших папок) та відновлювати їх;
- data, розділ даних зберігає призначені для користувача дані, починаючи від контактів і повідомлень до додатків і музики, і є розділом, до якого користувач може отримати доступ через посередництво файлового менеджера. Цей розділ очищується при виконанні скидання до заводських налаштувань;
- cache, ОС зберігає тут дані, що часто використовуються та компоненти застосунків. Цей розділ можна стерти, щоб виправити певні проблеми, він автоматично відновлюється і оновлюється з часом.
- misc, цей розділ містить іншу важливу інформацію про системні налаштуваннях, таку як конфігурація USB, ідентифікатор оператора зв'язку і інші параметри обладнання, які зазвичай в графічному інтерфейсі зберігаються у вигляді перемикачів включення/вимикання.

Якщо користувач не має прав адміністратора, то він в ОС Android може мати доступ тільки до розділу з даними, який відкривається, коли він підключає пристрій до свого ПК або використовує файловий менеджер. Якщо пам'ять телефону розширена за допомогою карти microSD, то пам'ять карти також включається в цей розділ з даними, доступними через ПК або файловий менеджер.

Для того щоб продивитися розділ даних телефону, телефони з встановленими версіями мобільної операційної системи Android Marshmallow або Nougat, мають власний файловий менеджер, який може отримати доступ до цього повного розділу. Ця опція знаходиться по шляху "Налаштування">"Зберігання">"Інше" (Settings>Storage>Other). Деякі телефони на старих версіях Android можуть включати чи не включати власний файловий менеджер, залежно від виробника.

Крім того, у Play Store доступні сторонні програми, які можуть виконувати функції файлових менеджерів. Найбільш розповсюдженими менеджерами є утиліти FX File Explorer і Solid Explorer.

Також можна переглядати файли з розділу data на ПК, підключившись через USB порт. При цьому телефон, що підключається має працювати у режимі MTP (Media Transfer), щоб користувач міг бачити повний розділ даних.

1.2 Огляд основних файлових систем, що використовуються в операційній системі Android

Файлова система (ФС) представляє собою структуру того чи іншого накопичувача інформації. Вона відповідає за те, який максимальний обсяг файлу можна зберегти в пам'яті, довжину їх назв, а також за організацію циклів запису або читання. Файлові системи виступають в ролі посередників між програмним кодом і фізичною пам'яттю.

Розглянемо основні ФС, що використовуються в ОС Android [6].

Файлова система VFAT (віртуальна FAT) вперше з'явилася в Windows 3.11 і була призначена для файлового введення/виводу в захищеному режимі. Ця файлова система підтримується Win9x та WinNT 4.0. Наприклад, VFAT – це «рідна» 32-розрядна файлова система для Windows95. Її контролює драйвер VFAT.VXD. VFAT використовує 32-розрядний код для всіх файлових операцій, а також може використовувати 32-розрядні драйвери захищеного режиму. При цьому, елементи таблиці розміщення файлів залишаються 12-ти або 16-розрядними, тому на диску використовується та ж структура даних (FAT). Тобто формат таблиці VFAT такий же, як і формат FAT.

Допрацьована версія VFAT дозволяє зберігати файли з довгими іменами, але по всіх інших характеристиках є морально застарілою. VFAT може використовуватися на картах пам'яті, а операційні системи на базі ОС Android повністю їх підтримують. В основному, це файлова система flash-накопичувачів Android.

Основний недолік VFAT – великі втрати на кластеризацію при великих розмірах логічного диска і обмеження на сам розмір логічного диска.

Файлова система YAFFS (Yet Another Flash File System) була розроблена для флеш-пам'яті NAND-типу Чарльзом Меннінгом з Нової Зеландії. Перша реалізація системи (YAFFS1) підтримує пристрої, що мають розмір сторінки 512 байт, а новіша (YAFFS2) – сучасні носії з великим розміром сторінки і більш серйозними обмеженнями записи. YAFFS2 використовувалася для організації роботи внутрішньої пам'яті гаджетів Android версії 2.2 і 2.3.

У більшості файлових систем для носіїв у вигляді флеш-пам'яті, блоки з неактуальною інформацією просто позначаються як такі. В YAFFS2 ці блоки додатково маркуються числами зростаючої арифметичної прогресії. Під час монтування відбувається сканування файлової системи, при якому пакети з актуальними даними легко ідентифікуються. Розташування блоків зберігається у вигляді дерев в оперативній пам'яті, що дозволяє значно прискорити монтування за рахунок знімків. Знімок має на увазі збереження цих дерев у флеш-пам'ять при розмонтуванні. Тоді при подальшому монтуванні ці

структури швидко зчитуються і відновлюються в пам'яті. Швидке монтування є одною з головних переваг YAFFS2 перед аналогами.

Файлові системи Ext2-Ext4 – це основні файлові системи ОС Android. На більшості сучасних гаджетів саме ці ФС використовуються для організації роботи внутрішнього сховища. Історично перші пристрої працювали під версіями Ext2. Але починаючи з версії Android 4, основними стали ФС Ext3, а потім і Ext4. Наявність функції журналювання є основною відмінністю між цими ФС. Це означає, що якщо в процесі запису або читання даних відбувається системний збій, наприклад, помилки пам'яті, то втрати або пошкодження даних користувача не відбудеться.

Незважаючи на те, що ФС формату Ext використовується в основному в блокових накопичувачах (жорсткі диски), користувачі можуть встановити даний тип і для карт пам'яті, але без сторонніх утиліт отримати до них доступ з операційних систем, крім ОС Linux, буде неможливо. Файлова система flash-накопичувачів Android зазвичай форматується в FAT (VFAT) або NTFS, а внутрішня пам'ять – в Ext3 або Ext4.

Файлова система F2FS (Flash Friendly File System) – це система, яка призначена в першу чергу для роботи з флеш-пам'яттю і SSD-накопичувачами. F2FS може використовуватися на картах пам'яті SD/MMC, а також з множиною інших типів пам'яті. Для повноцінної кастомізації існує цілий набір утиліт. До переваг цієї ФС можна віднести хорошу гнучкість в налаштуванні, високі показники збереження життєвого циклу блоків пам'яті, а також зберігання даних у вигляді журналу. При використанні системи F2FS відзначається [7] збільшення продуктивності при роботі з великою кількістю файлів. В інших сценаріях використання швидкість практично однакова в порівнянні з системою Ext4. Високу швидкість роботи забезпечує те, що індекси даних зберігаються в оперативній пам'яті, а підтримка F2FS включена в ядро Linux, починаючи з версії 3.8.

Іншою перевагою можна вважати дбайливе ставлення до вбудованого накопичувача інформації, так як враховуються особливості пам'яті, і

інформація об'єднується в секції. В результаті вбудована пам'ять повинна прослужити довше, ніж у випадку використання файлової системи Ext4.

Файлова система Samsung RFS (Robust File System) була розроблена корейською компанією Samsung для пристроїв на базі ОС Linux, різновидом якої є Android. Ця ФС призначається для флеш-пам'яті NAND (флеш-накопичувачі, що застосовуються у мобільних пристроях) і використовується в багатьох гаджетах власного виробництва. Для полегшення роботи з файлами використовується таблиця формату FAT, що дозволяє максимально просто зробити запис файлів на флеш-пам'ять і їх читання. Специфіка RFS враховує особливості NAND накопичувачів, що дозволяє збільшити тривалість їх експлуатації, а також знизити ймовірність втрати даних при системних збоях або випадкових відключеннях живлення.

Файлова система UBIFS, також як і Samsung RFS, призначена виключно для роботи з пам'яттю типу NAND. Основною перевагою даного типу ФС є зниження зносу носіїв даних. ФС UBIFS складається з двох шарів – UBI, який відповідає за роботу і зв'язок з фізичним носієм, і UBIFS, тобто самої файлової системи. Хоча розробником UBIFS є компанія Nokia, зустріти подібну ФС можна не тільки на оригінальних пристроях від даного виробника, а й на гаджетах інших виробників.

Файлова система SDCardFS прийшла на заміну файловій системі FUSE, яку довгий час використовувала в своїх пристроях компанія Google. Але ця ФС мала ряд недоліків, серед яких необхідно відзначити малу кількість пам'яті, що виділяється під застосунки, а також велике споживання сховища при кешуванні даних. ФС SDCardFS вирішувала ці проблеми, та представляла собою свого роду оболонку, що емулює ФС FAT32. Вона відправляє дані файловим системам нижнього рівня, що дозволяє оптимізувати роботу смартфонів, а також виправляє помилки, пов'язані з передачею даних на комп'ютер і назад. SDCardFS за своєю суттю не є класичною файловою системою, але її використання значно підвищує швидкість роботи смартфонів або планшетів на ОС Android.

Важливо відзначити, що в операційних системах Unix/Linux відсутнє поняття дисків. Файлова система Android має деревоподібну структуру. Також слід відзначити, що ОС Android як і ОС Linux, на відміну від Windows дуже чутливі до регістру. Наприклад папка Home, HoMe, home, HOME – це чотири різні файли (або папки), тоді як в Windows можна створити тільки одну папку з таким ім'ям. Символ "/" (слеш) означає кореневий каталог, з якого є доступ в усі інші папки.

Розглянемо детальну структуру файлової системи ОС Android (за винятком розділів перелічених у п. 1.1) [8]:

- а) / – кореневий каталог;
- б) /dev/ – включає у себе інформацію про систему і системні файли;

кореневий каталог для вузлів пристрів. Зазвичай цей каталог монтується як екземпляр «devtmpfs», але може бути іншого типу в sandboxed/containerized установках. Цей каталог управляється спільно ядром та systemd-udev (демон управління подіями пристрою) і не може записуватися іншими компонентами. Під цим каталогом може бути змонтований ряд віртуальних файлових систем спеціального призначення;

- в) /data/app – це місце, де зберігаються дані застосунків користувача (apk-файли встановлених застосунків та ігор) ;

- г) /data/app-lib – тут можна знайти додаткові бібліотеки файлів і конфігурацій, які призначені для роботи того чи іншого окремого застосунка;

- д) /data/dalvik-cache – файли кеш для Java-машини Dalvik, яка запускає та стежить за роботою застосунків);

- е) /data/data – файли застосунків (особисті налаштування, збереження, дані тощо);

- є) /data/system – папка з налаштуванням системних застосунків;

- ж) /mnt – в цю папку монтуються карти пам'яті, внутрішня пам'ять та інші віртуальні розділи;

з) / Preload / - додаткові файли і папки, які віддзеркалюються в папку /system / (є не у всіх Android);

и) /proc – віртуальна папка, що містить всі деталі Android системи, включаючи ядро, процеси, і параметри конфігурації

і) /proc/mount – містить список всіх файлових систем, що примонтовані;

ї) /system/app – папка для системних застосунків;

й) /system/bin – містить файли і посилання на бінарні файли, що виконуються;

к) /system/camerdata – файли, необхідні для забезпечення роботи відео-камери;

л) /system/etc – містить конфігураційні файли, які використовуються при завантаженні ОС і програм;

м) /system/framework – поточні процеси;

н) /system/fonts – файли, що зберігають системні шрифти;

о) /system/lib/ – сховище бібліотек системних застосунків і сервісів;

п) /system/lib/modules – папка з драйверами;

р) /system/build.prop – конфігураційний файл, який використовується для змін у налаштуваннях системи;

с) /storage – розділ для додавання не віртуальних карток пам'яті.

Знаючи цю інформацію про ФС, користувач може вирішувати різні завдання, наприклад видалити файли, що залишилися від деінстальованих застосунків, почистити Android від тимчасового сміття, змінити деякі налаштування тощо.

1.3 Огляд програмних аналогів

Зараз функції файлового менеджера можна знайти в усіх застосунках, які хоча б віддалено мають відношення до роботи з файлами. Але подібні програми не завжди зручні, наприклад, іноді треба дістатися до арк-файлу або звільнити місце у карті пам'яті, не "прив'язуючи" мобільний пристрій до

комп'ютеру. Тому важливо знайти той файловий менеджер, який не тільки впорається зі своїми прямими обов'язками, але і зможе порадувати користувача зручним інтерфейсом.

Розглянемо найбільш поширені існуючі файлові менеджери.

Файловий менеджер Total Commander (рис. 1.4) – безкоштовний, розповсюджений ФМ для ОС Android.

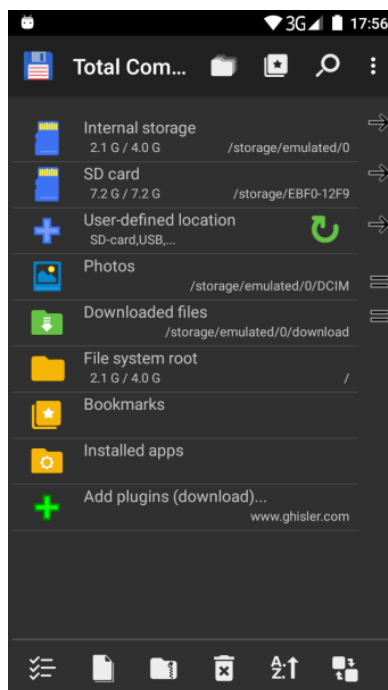


Рисунок 1.4 – Інтерфейс Total Commander

Це зручний і функціональний файловий менеджер з приємним інтерфейсом. Гарний варіант для звичайного користувача. Даний файловий менеджер має усі необхідні функції для зручного користування [9]:

- двохпанельний інтерфейс;
- root-доступ до файлової системи (при наявності прав);
- управління застосунками Android;
- вбудований архіватор, медіапрогравач;
- підтримка плагінів для доступу до USB флеш-носіїв, мережевого сервісу, клієнта FTP, протоколу WebDAV (він дозволяє веб серверу вести себе також як файловий сервер, підтримуючи спільну розробку веб контенту);

- текстовий редактор, закладки та інші інструменти;
- відправка файлів через Bluetooth.

У цьому ФМ передбачено багато зручних жестів: перетягування, виділення окремого файлу утриманням або декількох – протяжкою. Але зайвим є те, що у стартовому вікні винесені папки, в які збираються усі зображення, музика, завантаження тощо, адже такі самі папки є у робочому просторі майже кожного Android-пристрою.

Файловий менеджер Root Explorer – професійний багатофункціональний ФМ для користувачів Android з root-правами (рис. 1.5). Це не найсучасніший застосунок, але він часто оновлюється. У цих оновленнях присутній новий інтерфейс, нові функції і поліпшення стабільності [10].

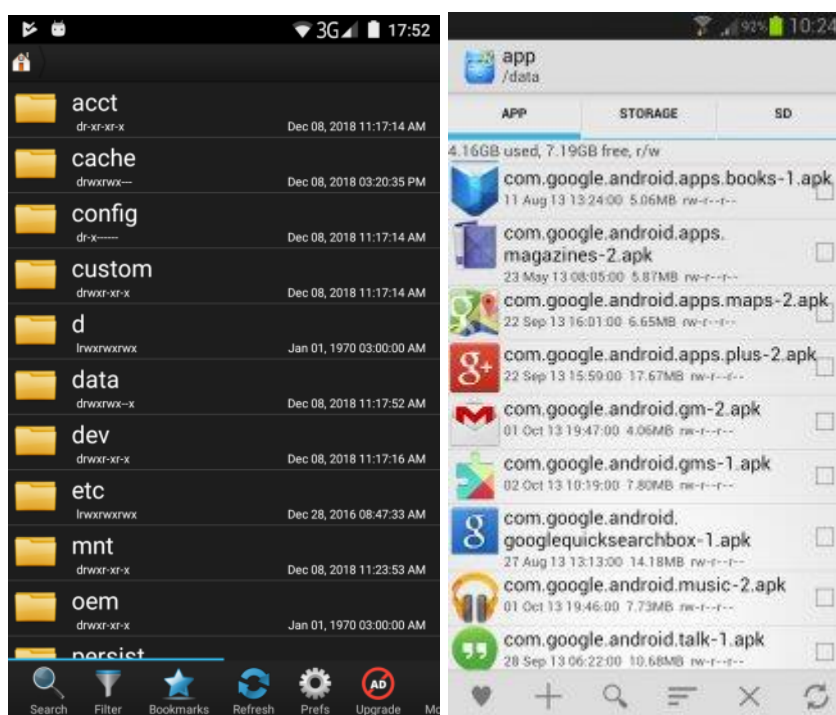


Рисунок 1.5 – Інтерфейс Root Explorer

Root Explorer надає доступ до всіх каталогів файлової системи без винятку (доступ до будь-яких системних прихованих файлів). У цьому застосунку є такі інструменти, як двійковий переглядач XML, можливість змінити власника і групи файлу, підтримка MD5 і багато іншого. ФМ має

складний інтерфейс та багато функцій. Це не дуже зручний варіант для звичайного користувача, адже дуже просто випадково видалити якісь системні папки або файли. Таким чином, цей файловий менеджер краще використовувати досвідченим користувачам або програмістам. Root Explorer – платна програма, яка коштує \$3.99 [11].

Файловий менеджер ES File Explorer – найбільш розповсюджений на даний час ФМ [9], що має всі потрібні для користувача ярлики і вкладки: коренева папка SD-карти; завантаження; фотографії; підключені хмарні сервіси (Google Диск, Dropbox, OneDrive та інші), підтримує FTP і підключення по локальній мережі); відсортовані за типами медіафайли (рис. 1.6). Тут же є присутнім і менеджер застосунків Android.

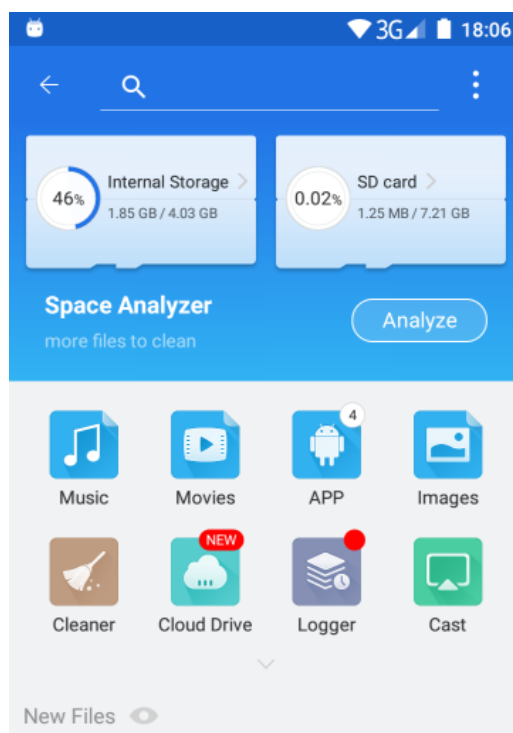


Рисунок 1.6 – Інтерфейс ES File Explorer

Для звичайних користувачів це дуже зручно і позбавляє від зайвої навігації по провіднику. Основним недоліком є надлишкова функціональність застосунку, яка не характерна для файлових менеджерів. Крім того, варто зазначити, що останні його версії стали сприйматися користувачами вже не

дуже однозначно: повідомляється про спливаючі повідомлення, погіршення інтерфейсу (з точки зору деяких користувачів) і інші зміни, що говорять на користь пошуку іншої програми для цих цілей [9].

Файловий менеджер X-plore (рис. 1.7) – як і попередній застосунок, є популярним ФМ, який тепер доступний і для мобільних пристроїв, що функціонують на базі ОС Android.

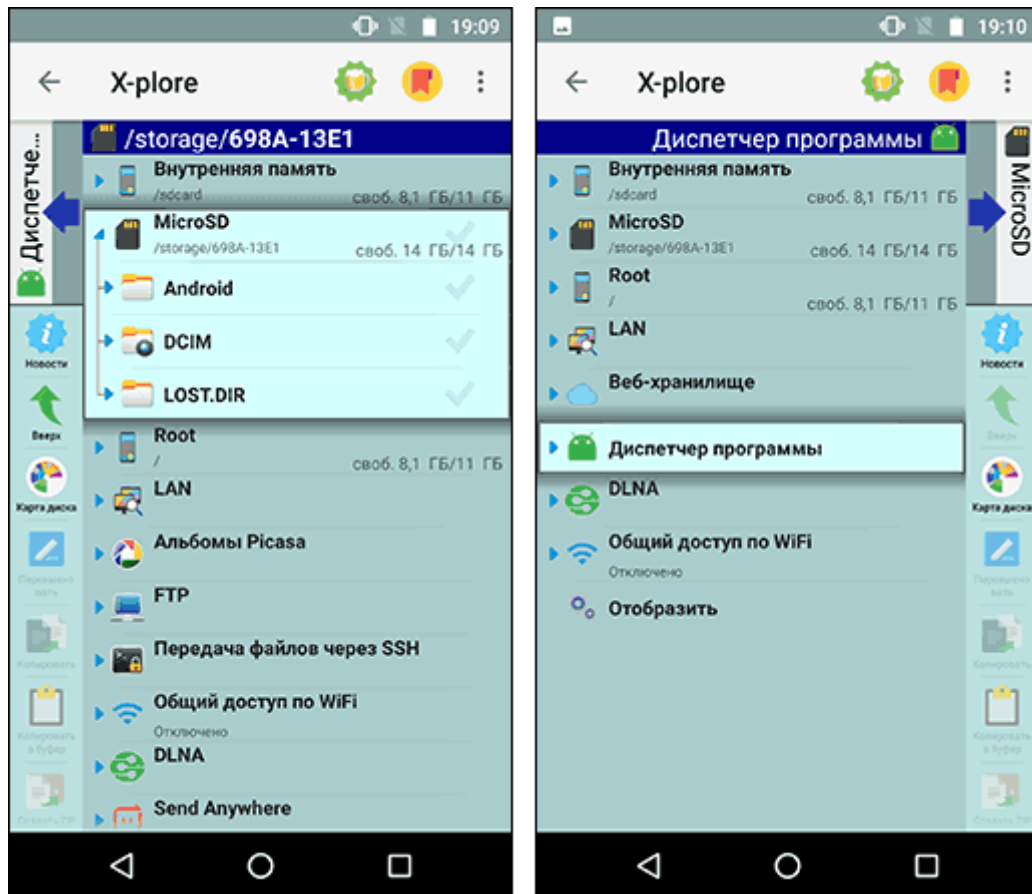


Рисунок 1.7 – Інтерфейс X-plore

Цей застосунок характеризується стабільною роботою та дозволяє забезпечити можливість роботи з документами, файлами і папками які розташовані на мобільному пристрої. Необхідно відзначити, що користувач може досить комфортно працювати з усіма дисками мобільного пристрою та, при необхідності, здійснювати відкриття фото і текстових документів. Також програма надає можливість доступу до властивостей файлів [12].

Найчастіше, цей застосунок встановлюють на смартфони та планшети з метою редагування файлів, їх перейменування, а також видалення. Завдяки створенню і редагуванню текстових файлів, створення папок, а також їх копіювання та переміщення, існує можливість забезпечити максимальний комфорт при роботі з мобільним пристроєм.

Файловий менеджер ASTRO File Manager є одним з найкращих та розповсюджених файлових менеджерів для мобільних Android-пристроїв, про що свідчить більш ніж 500 мільйонів установок з Play Market [13]. Базові та розширені можливості програми дозволяють виконувати стандартні операції з файлами, отримати контроль над усіма папками і процесами мобільної системи, управляти пам'яттю та встановленими застосунками.

Дизайн застосунку доволі простий (рис. 1.8), весь інтерфейс зроблений в приємних тонах. Майже всі іконки в ASTRO File Manager нагадують аналогічні для Windows, що спрощує для користувача розуміння принципів роботи.

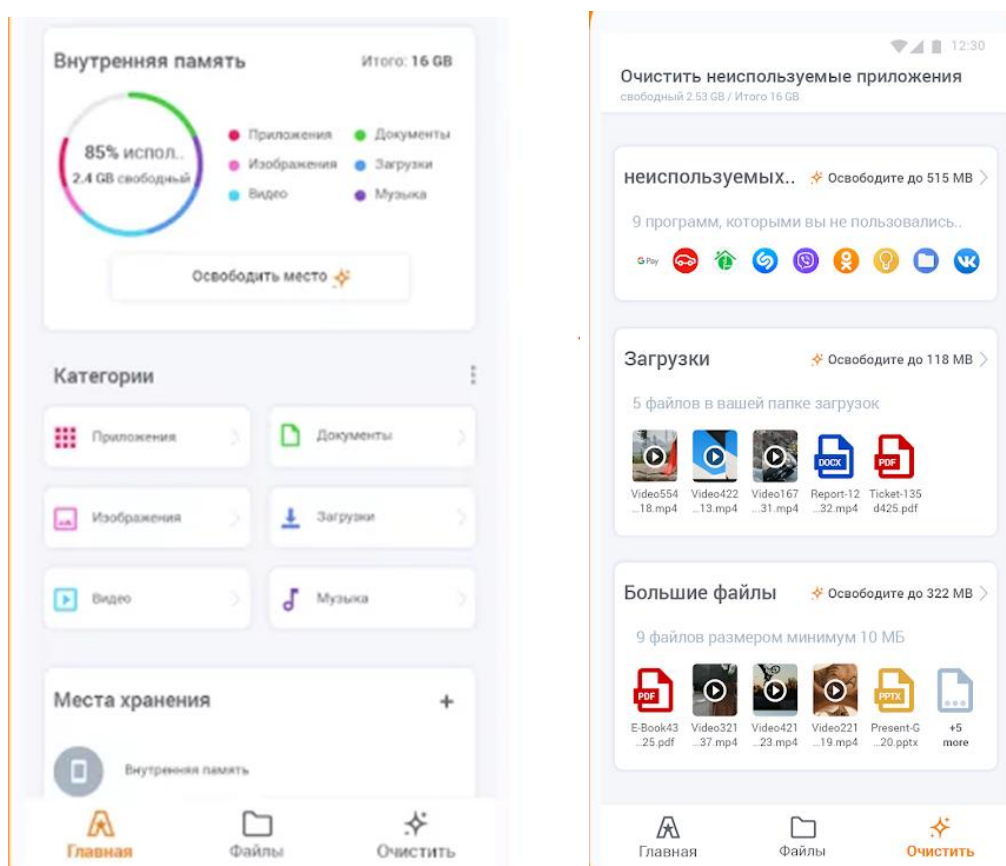


Рисунок 1.8 – Інтерфейс ASTRO File Manager

Зверху знаходиться панель з 15 кнопками для управління, що прокручують: Сеть, Отображение, Настройки, Закладки, Новая папка, Обновить, Сортировка, Вид, Вперед, Назад, Поиск, Редактировать, Мультивыбор, Переход в попередню директорію та Домой. Окремо слід відзначити можливість налаштовувати роботу з файлами під свої потреби для більш зручної і швидкої роботи. Можна налаштувати відображення папок на самому початку, відображення прихованих файлів, розширень файлів, інформації про файлах, превью для картинок та інше, а у кожного типу файлів є власна іконка. За допомогою ASTRO File Manager можна виконувати всі стандартні операції з файлами і папками, а також передавати файли по Bluetooth.

ASTRO File Manager може працювати з архівами Tar і ZIP. Також є присутнім менеджер процесів і утиліта для створення резервної копії apk-файлу з уже встановлених застосунків. Цей ФМ забезпечує можливість пошуку необхідних файлів, як на пристрої, так і на SD карті. Підтримується робота з Wi-Fi і Bluetooth мережами, але тільки після установки відповідних розширень. Особливістю ФМ є те, що для використання всіх цих функцій не потрібно root прав користувача.

Результати огляду програм-аналогів ФМ наведено у табл. 1.1.

Таблиця 1.1 – Результати огляду програм-аналогів ФМ

Назва ФМ	Розробник	Оновлення	Версія Android	Підтримка української мови	Ціна, \$
1	2	3	4	5	6
Total Commander	C. Ghisler	2020-11-02	від 1.5	—	—
Root Explorer	Speed Software	2020-10-28	від 2.3	—	3.99
ES File Explorer	ES Global	2021-01-30	від 4.0	—	—
X-plore	Lonely Cat Games	2020-12-21	від 4.1	—	—

Продовження табл. 1.1

1	2	3	4	5	6
ASTRO File Manager	Metago Mobile	2020-12-17	від 2.3	—	—

За результатами проведеного огляду програм аналогів ФМ, розроблено технічне завдання на розробку власного файлового менеджера, наведене в додатку А.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Обґрунтування вибору мови програмування та середовища розробки

Програмні застосунки під ОС Android можна створити практично на будь-якій популярній мові та знайти відповідні фреймворки або утиліти. Наприклад, мова BASIC не підтримується середовищем Android Studio і не підходить для середовищ Unity і Xamarin. Для неї є спеціальне середовище розробки B4A, в якому можна створювати Android- застосунки. B4A втілює концепцію візуального прототипування RAD, яка забезпечує швидкість і зручність програмування [14].

Відповідно до вимог технічного завдання (див. п. А.4), для створення програми має бути використана мова програмування Java. Тому, необхідність у виборі мови, як такої, практично відсутня. Але доцільно обґрунтувати цей вибір мови та розглянути можливості Java.

За даними щорічних опитувань спеціалізованих вітчизняних та міжнародних ресурсів, наприклад українського ресурсу dou.ua [14], Java та JavaScript останніми роками традиційно очолюють рейтинг найпопулярніших мов програмування. На рис.2.1 представлені результати щорічного опитування щодо мов програмування. Цього разу ресурсом dou.ua було зібрано 7211 анкет, з них 92% респондентів перебувають в Україні.

Але, як показують дані рис. 2.2, для розробки мобільних застосунків в Україні у 2021 році основними мовами є Kotlin та Swift, а частка кросплатформових фреймворків менша, ніж здавалася.

Незважаючи на це, для будь-якого розробника мобільних застосунків на ОС Android, першою і найкращою мовою програмування поки залишається Java, так як вона підтримується компанією Google. Крім того, більшість застосунків в Google Play побудовані саме на Java [15]. Саме з цих міркувань, а також з огляду на свій досвід розробки, в якості мови програмування було обрано мову Java.

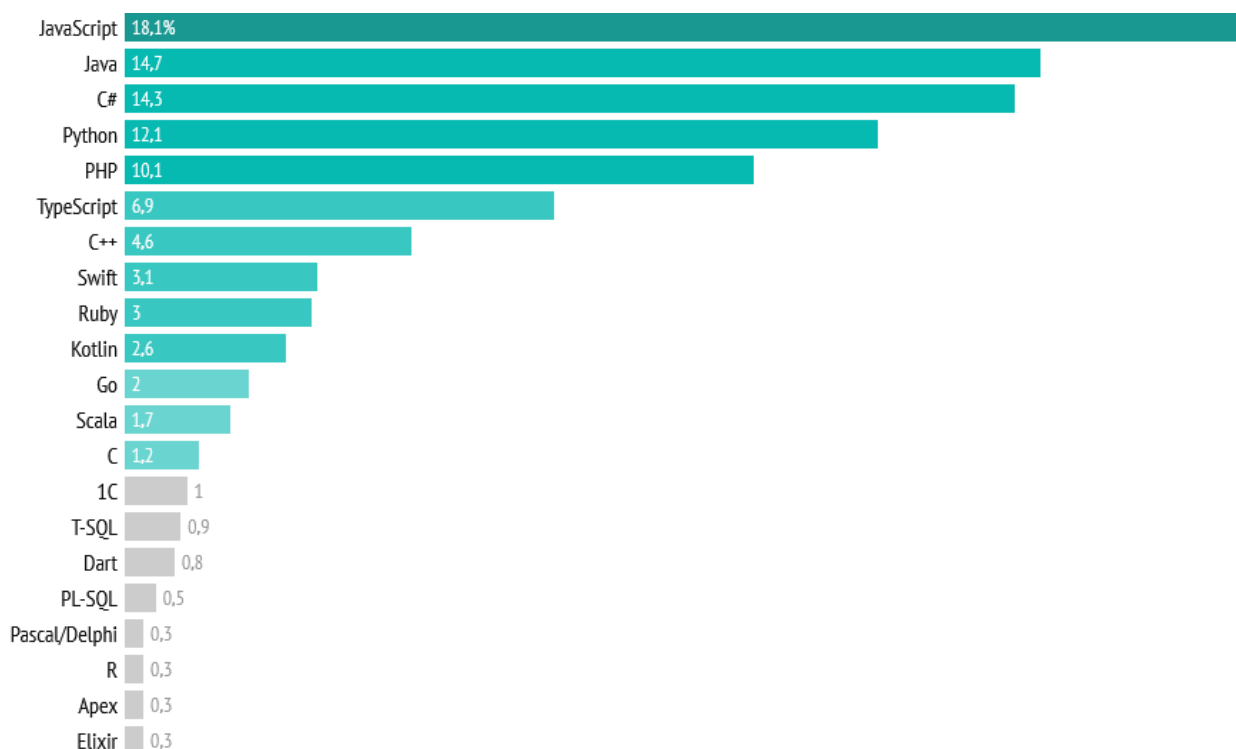


Рисунок 2.1 – Використання мов для роботи у поточний час (у 2021 році)

[Mobile](#) [Back-end](#) [Front-end](#) [Data processing](#) [Desktop](#) [System](#) [QA automation](#) [Full Stack](#) [IoT](#) [GameDev](#)



Рисунок 2.2 – Використання мов для мобільної розробки

Мова програмування Java була спочатку розроблена компанією Sun Microsystems, з ініціативи Джеймса Гослінга. Вона була випущена в 1995 році в якості основного компонента Java-платформи Sun Microsystems (Java 1.0 [J2SE]).

Останнім випуском Java Standard Edition є Java SE 11. З розвитком Java і її широкою популярністю було створено безліч конфігурацій для різних типів платформ. Наприклад, J2EE для корпоративних застосунків, а J2ME для мобільних застосунків.

Особливостями Java є [16]:

- об'єктно-орієнтованість. На Java всі сутності є об'єктами. Java можна легко розширити, оскільки вона базується на об'єктній моделі;
- незалежність від платформи. На відміну від багатьох інших мов програмування, включаючи C та C++, коли Java компілюється, вона не компілюється під певну машинну платформу, а в незалежний від платформи байт-код. Цей байт-код поширюється через Інтернет і інтерпретується віртуальною машиною Java (JVM) на будь-якій платформі, на якій він запущений;
- простота. Java розроблена для того, щоб її було легко вивчити. Якщо користувач розуміє основні концепції об'єктно-орієнтованого програмування, то Java легко засвоїти;
- безпека. Методи автентифікації Java засновані на шифруванні з відкритим ключем;
- незалежність від архітектури. Компілятор Java генерує нейтральний до архітектури формат об'єктного файлу. Це робить скомпільований код таким, що виконується на багатьох процесорах, за наявності системи середовища виконання Java;
- переносимість. Незалежність від архітектури та відсутність аспектів специфікації, що залежать від реалізації, робить мову Java переносимою. Компілятор на Java написаний на ANSI C з чистою межею переносимості, яка є підмножиною стандарту POSIX;
- надійність. Java докладно зусиль для усунення ситуацій, схильних до помилок, приділяючи основну увагу перевірці помилок під час компіляції і перевірці під час виконання;
- багатопоточність. Завдяки багатопоточності Java можна писати програми, які можуть виконувати множину завдань одночасно. Ця особливість проєктування дозволяє розробникам створювати інтерактивні застосунки, які можуть працювати без збоїв;

- інтерпретованість мови. Байтовий код Java перекладається «на льоту» (динамічна компіляція – Just In Time) в машинні інструкції та ніде не зберігається. Процес розробки більш швидкий і аналітичний, так як зв'язування - це поступовий і легкий процес;

- висока продуктивність. Завдяки використанню динамічних компіляторів Java забезпечує високу продуктивність;

- розподілення. Мова Java розроблена для розподіленого середовища Інтернету;

- динамічність. Java вважається більш динамічним, ніж мови C або C++, оскільки вона призначена для адаптації до я середовищ, що розвиваються. Програми Java можуть містити великий обсяг run-time інформації, яку можна використовувати для перевірки та дозволу доступу до об'єктів під час виконання.

Java – як мова, яка має об'єктно-орієнтовану функцію, підтримує наступні основні концепції: поліморфізм; спадкування; інкапсуляція; абстракція; класи; об'єкти; екземпляри; методи тощо.

Завдяки цьому, мова Java використовується у різноманітних напрямках діяльності [17]:

- для розробки програм для ОС Android;
- створення корпоративного програмного забезпечення;
- широкого кола мобільних програм;
- у наукових обчисленнях;
- для аналізу великих обсягів даних;
- програмування Java на апаратних пристроях;
- для серверних технологій, таких як Apache, JBoss, GlassFish тощо.

Найбільш розповсюдженими інтегрованими середовищами розробки для Java є наступні: NetBeans; IntelliJ IDEA; Eclipse; JDeveloper; Android Studio; DrJava та MyEclipse [18].

Серед цих середовищ, профільним середовищем Java розробки для ОС Android є Android Studio (це середовище розробки згідно технічного завдання). В його основі лежить середовище IntelliJ IDEA, оптимізоване під рішення конкретних мобільних завдань: рефакторінга; збірки; шаблонізації; пошуку причин низької продуктивності та її збільшення. Також це середовище працює на Apache License 2.0, має відкритий вихідний код і тому є безкоштовним. Android Studio багато запозичив з Eclipse в якості основної IDE для розробки власних застосунків для Android.

Версії Android Studio сумісні з деякими операційними системами Apple, Windows і Linux. Завдяки підтримці інтеграції Google Cloud Platform і застосунків Google, Android Studio пропонує розробникам добре оснащений інструментарій для створення застосунків Android або інших проєктів і є невід'ємною частиною розробки Android з 2013 року.

Програміст може почати розробку свого застосунку для Android в будь-який з наступних операційних систем:

- Microsoft® Windows® 10/8/7 / Vista / 2003 (32- або 64-розрядна);
- Mac® OS X® 10.8.5 або вище, до 10.9 (Mavericks);
- середовище робочого столу GNOME або KDE.

Всі необхідні інструменти для розробки застосунків Android можуть бути завантажені з Інтернету.

Android Studio постійно розвивається, але наразі можна виділити такі важливі функції:

- живі макети (layout), тобто редагувальник WYSIWYG який забезпечує живе кодування – подання (rendering) програми в реальному часі;
- консоль розробника, тобто підказки по оптимізації, допомога по перекладу, стеження за напрямком, агітації та акції – метрики Google аналітики;
- резерви бета релізів та покрокові релізи;
- базування на системі автоматичного збирання проєктів Gradle;
- Android-орієнтований рефакторинг (зміна внутрішньої структури

програмного забезпечення) та швидкі виправлення;

- Lint-інструменти для дослідження й вивірення програмного коду, що охоплюють продуктивності, юзабіліті, сумісності версій та інші проблеми;
- використання можливостей ProGuard (утиліти, яка видаляє з готового коду невикористовувані фрагменти та змінює імена змінних і методів для ускладнення реверс-інжинірингу застосунку) та підписів до програм;
- шаблони для створення поширених Android дизайнів та компонентів;
- багатий редактор макетів (layouts) що дозволяє користувачам перетягнути і покласти (drag-and-drop) компоненти користувацького інтерфейсу, як варіант, переглянути одночасно макети (layouts) на різних конфігураціях екранів.

На додаток до набору стандартних бібліотек розробки Java (які забезпечують підтримку таких загальних завдань, як обробка рядків, робота в мережі і управління файлами), середовище розробки Android також включає бібліотеки Android. Це набір бібліотек на основі мови Java, специфічних для розробки під Android. Приклади бібліотек в цій категорії включають бібліотеки каркаса застосунків на додаток до тих, які полегшують створення призначеного для користувача інтерфейсу, малювання графіки і доступ до бази даних.

Розглянемо основні бібліотеки Android, які доступні розробникам Android-пристроїв:

- android.app надає доступ до моделі програми та є основою для усіх застосунків Android;
- android.content полегшує доступ до вмісту, забезпечує публікацію та обмін повідомленнями між програмами та компонентами програми;
- android.database використовується для доступу до даних, опублікованих постачальниками вмісту, і включає класи управління базами даних SQLite;
- android.graphics є API низькорівневого 2D графічного малювання, що включає кольори, точки, фільтри, прямокутники та полотна;

- `android.hardware` представляє API, що забезпечує доступ до апаратних засобів, таких як акселерометр та датчик освітленості;
- `android.opengl` є інтерфейсом Java для API візуалізації 3D-графіки OpenGL ES;
- `android.os` надає програмам доступ до стандартних служб операційної системи, включаючи повідомлення, системні послуги та зв'язок між процесами;
- `android.media` пропонує класи для відтворення аудіо та відео інформації;
- `android.net` є набором API, що забезпечують доступ до мережевого стеку. Включає `android.net.wifi`, який забезпечує доступ до бездротового стеку пристрою;
- `android.print` включає набір класів, що дозволяє надсилати вміст на налаштовані принтери з застосунків Android;
- `android.provider` набір класів, що забезпечують доступ до стандартних баз даних постачальника вмісту Android, таких як ті, що підтримуються календарем та програмами контактів;
- `android.text` використовується для візуалізації та обробки тексту на дисплеї пристрою;
- `android.util` це набір службових класів для виконання таких завдань, як перетворення рядків і чисел, обробка XML та маніпулювання датою та часом;
- `android.view` є основними будівельними блоками користувальницьких інтерфейсів застосунків;
- `android.widget` це багата колекція вбудованих компонентів користувацького інтерфейсу, таких як кнопки, мітки, спискові презентації, менеджери компоновки, перемикачі тощо;
- `android.webkit` - набір класів, призначених для того, щоб дозволити будувати можливості веб перегляду у застосунках.

Таким чином, Android реалізований у вигляді архітектури стека програмного забезпечення, що складається з ядра Linux, середовища виконання програмного продукту та відповідних бібліотек, інфраструктури застосунків і набору застосунків. Застосунки переважно написані на Java і скомпільовані до формату байт-коду в середовищі збірки Android Studio. Коли застосунок згодом встановлюється на пристрій, цей байт-код компілюється середовищем виконання Android (ART) у власний формат, який використовується центральним процесором. Ключовими цілями архітектури Android є продуктивність і ефективність як при виконанні програми, так і при реалізації повторного використання в дизайні застосунків.

2.2 Аналіз технічного завдання

Згідно з технічним завданням, повний текст якого представлений в Додатку А, програма призначена для користувачів мобільних пристроїв з ОС Android, яким потрібен простий та зручний файловий менеджер. Тому система повинна виконувати наступні функції:

- мати інтуїтивно-зрозумілий інтерфейс;
- надавати користувачу зручні можливості для вибору файлу або каталогу;
- відображати всі логічних дисків на мобільному пристрої;
- забезпечувати перегляд вмісту обраного каталогу;
- відображення типів файлів;
- відображення інформації про файл (назва файлу, тип файлу, розмір, дата створення, дата зміни);
- створення нових каталогів;
- переміщення, копіювання, видалення файлів.

Крім того було зазначено, що робота програми повинна розпочинатися з запуску програми шляхом натискання іконки запуску. Далі користувач обирає

потрібний логічний диск, а потім – каталог. Для відкриття потрібного каталогу натискає на його іконку.

Для копіювання, переміщення, вилучення та перейменування файлів користувач може використовувати відповідні команди з контекстного меню, яке з'являється при натисканні на іконку файлу.

Програма повинна також надавати можливість користувачу зручно працювати з сенсорним екраном.

Важливо відзначити системні вимоги до смартфона, на якому можна експлуатувати дану програму:

- RAM 1024 Мб;
- частота процесора не менше 1300МГц;
- наявність вільного дискового простору – 2,5 Мб;
- версія Android вище ніж 4.1 JellyBean (API level– 16).

Для виконання основної функції мобільного додатку – доступу до файлу (каталогу), його перейменування та видалення мають бути використані клас `java.io.File` та його методи (наприклад: `renameTo(String newName)`, `delete()`).

Для копіювання та переміщення файлів або каталогів мають бути використані класи та їхні методи для роботи з потоками: `BufferedOutputStream` (клас цепного потоку), `FileOutputStream` (клас потоку для з'єднання), `BufferedInputStream`, `FileInputStream`.

Для забезпечення доступу до файлів та каталогів на зовнішньому носії інформації (SD карта), та зміни їх вмісту у маніфесті мобільного застосунку необхідно прописати: `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>` та `<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>`.

Для виконання основної функції мобільного застосунку – доступу до файлу (каталогу), його перейменування та вилучення використовують клас `java.io.File` та його методи [12]:

– `boolean createNewFile()`: створює новий файл, який переданий в конструктор. У разі вдалого створення повертає `true`;

- `boolean delete()`: видаляє каталог або файл шляхом, який переданий в конструктор. При вдалому вилученні повертає `true`;
- `boolean exists()`: перевіряє, чи існує за вказаною в конструкторі шляху файл або каталог. І якщо файл або каталог існує, то повертає `true`;
- `String getAbsolutePath()`: повертає абсолютний шлях, переданого в конструктор об'єкта;
- `String getName()`: повертає коротке ім'я файлу або каталогу;
- `String getParent()`: повертає ім'я батьківського каталогу;
- `boolean isDirectory()`: повертає значення `true`, якщо за вказаним шляхом розташовується каталог;
- `boolean isFile()`: повертає значення `true`, якщо за вказаним шляхом знаходиться файл;
- `boolean isHidden ()`: повертає значення `true`, якщо каталог чи файл є прихованими;
- `canExecute()`: перевіряє чи може програма виконати файл за шляхом, вказаним у поточному екземплярі класу `Path`;
- `canRead ()`: файл є доступним для читання;
- `canWrite()`: файл є доступним для запису;
- `long length()`: повертає розмір файлу в байтах;
- `long lastModified()`: повертає час останньої зміни файлу або каталогу. Повертає значення, що дорівнює кількості мілісекунд, що пройшли з початку епохи Unix;
- `String[] list()`: повертає масив файлів і підкаталогів, які знаходяться в певному каталозі;
- `File[] listFiles()`: повертає масив файлів і підкаталогів, які знаходяться в певному каталозі;
- `boolean mkdir()`: створює новий каталог і при вдалому створенні повертає значення `true`;
- `boolean renameTo(File dest)`: перейменовує файл або каталог. Також

може виконувати функції переміщення.

Для копіювання файлів та каталогів використовують класи `java.nio.channels.FileChannel`, `java.io.FileOutputStream` та `java.io.FileInputStream`.

Найбільш використовуваним для копіювання файлу є спосіб з використанням потоків. Алгоритм виконання цього способу наступний. Створюються два файли – вихідний та той, що буде його копією. Після цього створюється `InputStream` від вихідного файлу, а далі він записується в результуючий файл-копію з використанням `OutputStream`.

Нижче представлений метод на мові Java, що виконує копіювання файлу з використанням потоків [19]:

```
private static void copyFileUsingStream(File source, File
dest) throws IOException {
    InputStream is = null;
    OutputStream os = null;
    try {
        is = new FileInputStream(source);
        os = new FileOutputStream(dest);
        byte[] buffer = new byte[1024];
        int length;
        while ((length = is.read(buffer)) > 0) {
            os.write(buffer, 0, length);
        }
    } finally {
        is.close();
        os.close();
    }
}
```

Щодо способу копіювання файлів з використанням каналів, то в класі `java.nio.channels.FileChannel` є корисний метод `transferFrom()`. Згідно з документацією, цей спосіб копіювання файлу працює швидше, ніж попередній (з використанням потоків).

Наведемо код, що демонструє використання цього методу [19]:

```
private static void copyFileUsingChannel(File source, File
```

```

dest) throws IOException {
    FileChannel sourceChannel = null;
    FileChannel destChannel = null;
    try {
        sourceChannel = new
FileInputStream(source).getChannel();
        destChannel = new
FileOutputStream(dest).getChannel();
        destChannel.transferFrom(sourceChannel, 0,
sourceChannel.size());
    }finally{
        sourceChannel.close();
        destChannel.close();
    }
}

```

Починаючи з Java 7 було введено в клас Files, що надає метод copy() для копіювання файлу. Цей клас використовує вбудовані можливості системи, щоб скопіювати файл.

Проведене тестування розглянутих вище методів копіювання на прикладі файлів розміром 1 Gb, показало перевагу способу копіювання файлу з використанням потоків [19].

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка загального алгоритму роботи програми

Алгоритм роботи програми, що розробляється має забезпечити виконання функцій передбачених технічним завданням і перелічених у п. А. 3.

Згідно з завданням, вхідними даними для застосунку можуть бути наступні керуючі дії користувача:

- подія натискання іконки запуску програми файлового менеджера;
- подія натискання на іконки каталогів та файлів, з яким буде працювати користувач;
- події натискання на пункти контекстного меню (з'являються коли користувач натискає на елемент списку) для виконання функцій вилучення, перейменування, переміщення або запуску файлу.

Згідно з завданням, вихідними даними для застосунку можуть бути наступні зміни стану обраних користувачем об'єктів або інформація щодо цих об'єктів:

- детальна інформація про файл або каталог;
- відкриття діалогового вікна для перейменування файлу або каталогу;
- відкриття діалогового вікна для підтвердження вилучення файлу або каталогу;
- відкриття діалогового вікна для переміщення або копіювання файлу або каталогу;
- запуск файлу, якщо система має програмний засіб за допомогою якого можна відкрити даний файл, та відкриває вікно для вибору застосунку, якщо система має декілька таких програмних засобів.

Блок-схема алгоритму програми, що забезпечує виконання передбачених технічним завданням функцій наведена на рис. 3.1.

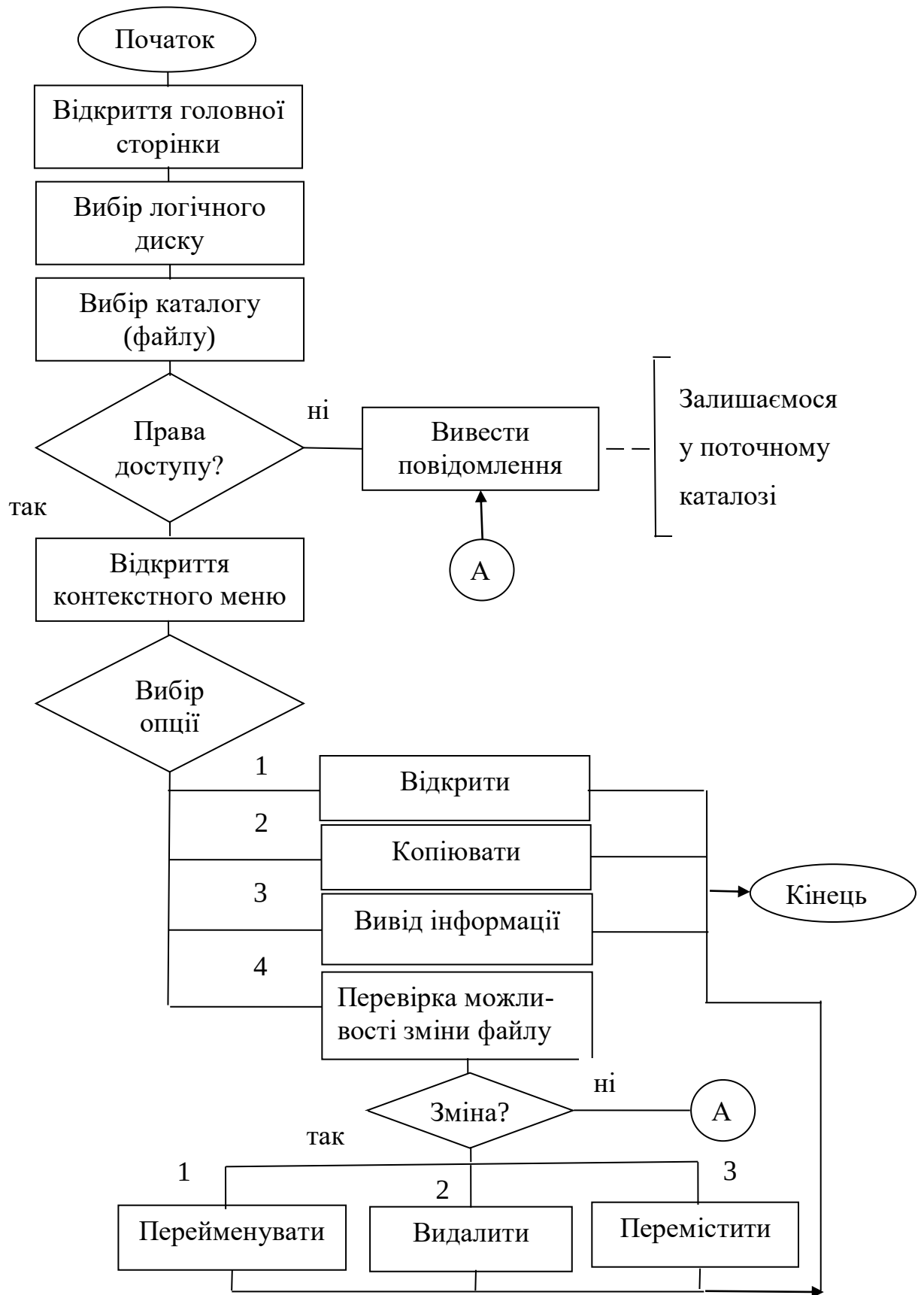


Рисунок 3.1 – Блок-схема алгоритму програми

3.2 Розробка програми застосунку

Програма, як будь-яка традиційна програма на Java, складається з файлів з вихідним кодом, що мають розширення java, та файлів розмітки інтерфейсу з розширенням xml.

Розпочинати створення програми зручно зі створення інтерфейсу, хоча тим часом можна паралельно розробляти файли Java. Інтерфейс створюється за допомогою мови xml, що розшифровується як розширювана мова розмітки (eXtensible Markup Language).

Ключовим компонентом для створення візуального інтерфейсу в Android-застосунках є activity (активність). Часто activity асоціюється з окремим екраном або вікном застосунку, а перемикання між вікнами буде відбуватися як переміщення від однієї activity до іншої. Застосунок може мати одну або кілька activity.

Активність має свій життєвий цикл – від початку, коли Android створює екземпляр активності, проміжний стан, і до кінця, коли екземпляр знищується системою і звільняє ресурси, що використовувались. Активність може перебувати в трьох наступних станах [20]:

- активна (active або running), якщо активність знаходиться на передньому плані екрана. Користувач при цьому може взаємодіяти з активним вікном;
- призупинена (paused), коли активність втратила фокус, але все ще видима користувачу. Тобто поточна активність знаходиться зверху і частково перекриває цю активність. Призупинена активність може бути знищена системою в критичних ситуаціях при нестачі пам'яті;
- зупинена (stopped), у випадку якщо ця діяльність повністю закрита іншою активністю. Вона більше не видима користувачу і може бути знищена системою, якщо пам'ять необхідна для більш важливого процесу.

Якщо активність, яка була знищена системою, потрібно знову показати на екрані, вона повинна бути повністю перезапущена і відновлена в своєму

попередньому стані.

Всі об'єкти активності є об'єктами класу `android.app.Activity`, який містить базову функціональність для всіх `activity`.

Головною активністю застосунку є `MainActivity`. Для швидкого відгуку системи на дії користувача було створено клас фрагменту `FileManagerFragment`, де знаходиться опис майже всієї логіки програми. Для взаємодії `MainActivity` та `FileManagerFragment` було використано клас `android.support.v4.app.FragmentManager`, який необхідний для роботи з фрагментами активності, та інтерфейс `DocumentSelectListener`. Також, завдяки цьому, код має досить компактний вигляд та легко читається. Частина цього коду представлена на рисунку 3.2.

```
public class MainActivity extends AppCompatActivity implements DocumentSelectListener {
    private ActionBar mActionBar;
    private FragmentManager mFragmentManager;
    private FragmentTransaction mFragmentTransaction;
    private FileManagerFragment mDirectoryFragment;
    private boolean mIsSelectionMode;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mActionBar = getSupportActionBar();
        if (mActionBar != null) {
            mActionBar.setTitle("Simple file manager");
        }
        mFragmentManager = getSupportFragmentManager();
        mFragmentTransaction = mFragmentManager.beginTransaction();
        mDirectoryFragment = new FileManagerFragment();
        mFragmentTransaction.replace(R.id.fragment_container, mDirectoryFragment, mDirectoryFragment.toString());
        mFragmentTransaction.commit();
    }
}
```

Рисунок 3.2 – Функція `onCreate()` `MainActivity`

Основна логіка програми базується на "звертанні" до файлів та каталогів. Реалізується це за допомогою класу `java.io.File`. Його завданням є управління інформацією про файли та каталоги. Наприклад, для отримання логічних дисків та кореневого каталогу було написано функцію `listRootFolders()`, її код наведено на рисунку 3.3 де доступ до внутрішнього та

зовнішнього сховища було реалізовано методами `initInternalStore(extStorageAbsPath)` та `initExternalStore(extStorageAbsPath)`.

```
private void listRootFolders() {
    mCurrentDir = null;
    mItems.clear();

    String extStorageAbsPath = Environment.getExternalStorageDirectory().getAbsolutePath();
    //intStore
    initInternalStore(extStorageAbsPath);
    //extStore
    initExternalStore(extStorageAbsPath);
    //root
    ListItem fs = new ListItem();
    fs.setTitle("/");
    fs.setSubtitle("SystemRoot");
    fs.setIcon(R.drawable.ic_directory);
    fs.setFile(new File("/"));
    mItems.add(fs);
    mListAdapter.notifyDataSetChanged();
}
```

Рисунок 3.3 – Функція `listRootFolders()`

Слід звернути увагу на метод `initExternalStore(extStorageAbsPath)` (див. рисунок 3.4), адже щоб отримати шлях у кореневому каталозі до зовнішнього сховища, довелося звертатися до файлу `proc/mounts` який зберігає інформацію про пристрої, що примонтовані до смартфона. Файл був зчитаний за допомогою класів `BufferedReader` и `FileReader`. Шляхом виключення, було отримано масив `String`-елементів який зберігає шляхи до усіх примонтованих `storage`-пристроїв.

Також важливою і основною є функція `listFiles(File dir)` яка знаходить усі файли й каталоги, що знаходяться у папці `dir`. Тут здійснюється перевірка чи є доступ до файлу або каталогу і якщо ні, то користувач отримує відповідне повідомлення.

Якщо доступ є, то отримується масив `File`-елементів що знаходяться у відповідній папці, проводиться сортування `File`-елементів (спочатку каталогів, а

потім файлів) за ім'ям та заповнюється ListView на основі отриманих даних. Код цієї функції наведено у Додатку А – клас `com.alburd.sfmapp.ui.fragments.FileManagerFragment`, метод `listFiles(File dir)`.

```
private void initExternalStore(String extStorageAbsPath) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader("/proc/mounts"));
        String infoAboutMountDevices;
        ArrayList<String> result = new ArrayList<>();
        while ((infoAboutMountDevices = reader.readLine()) != null) {
            if (!infoAboutMountDevices.contains("/storage"))
                || infoAboutMountDevices.contains("asec")
                || infoAboutMountDevices.contains("tmpfs")
                || infoAboutMountDevices.contains("none")) {
                continue;
            }
            String[] info = infoAboutMountDevices.split(" ");
            if (!extStorageAbsPath.contains(info[1])) {
                result.add(info[1]);
            }
        }
        reader.close();
        for (String path : result) {
            ListItem item = new ListItem();
            item.setTitle("ExternalStorage");
            item.setIcon(R.drawable.ic_external_storage);
            item.setSubtitle(getInfoAboutFileSystemSpace(path));
            item.setFile(new File(path));
            mItems.add(item);
        }
    } catch (Exception e) {
        Log.e(LOG_FILE_MANAGER_FRAGMENT, e.getMessage());
    }
}
```

Рисунок 3.4 – Функція `initExternalStore()`

Для створення каталогу реалізується проста функція `mkdir()` класу `File` у методі `mkDir()` у класі `FileManagerFragment`. Її код також представлений у Додатку А.

Далі слід розглянути функції відкриття, перейменування, вилучення, переміщення, копіювання файлів та отримання інформації про файл. Ці дії пропонує зробити контекстне меню, яке реалізовується за допомогою класу `android.support.v7.widget.PopupMenu`. Він дозволяє створити спливаюче меню, що прив'язане до елементу `View`. Меню реалізовано у вигляді модального вікна, яке відображається нижче від батьківського меню або в іншому місці, якщо місця знизу недостатньо.

В ОС Андроїд для переходу з однієї активності в іншу і назад використовується об'єкт `Intent` (намір). `Intent` є об'єктом опису операції, яку необхідно виконати через систему `Android`. Тобто необхідно спочатку описати деяку операцію у вигляді об'єкта `Intent`, після чого відправити її на виконання в систему викликом одного з методів активності `Activity`.

`Android` підтримує два види `Intent` – це явні (`explicit`) наміри та неявні (`implicit`) наміри. Неявні наміри – це наміри, які не дають певні цільові компоненти, але включають достатньо інформації для системи щоб визначити, який компонент підходить для запуску цього наміру.

Коли `Android` отримує неявний об'єкт для виконання, то система шукає відповідні компоненти шляхом порівняння вмісту `Intent` з фільтрами `Intent` інших застосунків, зареєстрованих в файлах маніфестів. Якщо параметри об'єкта `Intent` збігаються з параметрами одного з фільтрів `Intent`, то система запускає цей компонент і передає йому об'єкт `Intent`. При наявності декількох відповідних фільтрів система відкриває діалогове вікно, де користувач може вибрати відповідний застосунок.

Фільтр `Intent` є секцією в файлі маніфесту застосунку, що описує типи об'єктів `Intent`, які компонент міг би виконати. Таким чином, наявність фільтра `Intent` в описі активності в маніфесті дозволяє іншим програмам безпосередньо запускати дану операцію за допомогою деякого об'єкту `Intent`. Якщо фільтр `Intent` не описаний, то операцію можна буде запустити тільки за допомогою явного наміру.

У програмі, що розробляється відкриття файлу здійснюється завдяки

неявному наміру (клас `android.content.Intent`), який дозволяє передати намір здійснити дію з файлом за допомогою іншого застосунку.

Перейменування та переміщення файлу забезпечує метод `renameTo()` класу `File`, а також клас діалогу `android.app.AlertDialog`. Реалізація методу `showRenameDialog()` наведена на рис 3.5. Схожий вигляд має функція `moveFile()` класу `FileManagerFragment`, яка виконує функцію переміщення. Її код наведено у Додатку А.

```
private void showRenameDialog(int position) {
    final File file = mItems.get(position).getFile();
    final EditText inputNewName = AndroidUtil.initEditText(mContext, file.getName());
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage("Do you want to rename file?")
        .setView(inputNewName)
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String newName = inputNewName.getText().toString();
                if (!TextUtils.isEmpty(newName)) {
                    String path = file.getAbsolutePath()
                        .substring(0, file.getAbsolutePath().lastIndexOf("/"));
                    if (file.renameTo(new File(path, newName))) {
                        listFiles(mCurrentDir);
                    } else {
                        // TODO: 12/11/18 doesn't want work with SDCard
                        Toast.makeText(mContext, "You can't rename the file...", Toast.LENGTH_SHORT).show();
                    }
                }
            }
        })
        .setNegativeButton("CANCEL", null)
        .show();
}
```

Рисунок 3.5 – Функція `showRenameDialog ()`

Вилучення файлу забезпечує метод `showDeleteDialog ()` за допомогою функції `delete()` класу `File`.

Отримання інформації про файл реалізує метод `getDescriptionFile(File file)`. Щоб отримати детальну інформацію використовуються такі методи класу `File`: `getName()`, `getAbsolutePath()`, `length()`, `canRead()`, `canWrite()`, `canExecute()`, `lastModified()`. Опис цих методів був наведений у п. 2.2.

Реалізація методів `showDeleteDialog()` та `getDescriptionFile(File file)` досить проста, їхній код наведено у Додатку А, клас `com.alburd.sfmapp.ui.fragments.FileManagerFragment`.

Цікавою також є функція копіювання `copyFile(int position)`. Щоб при копіюванні користувальницький інтерфейс не "гальмував", було прийняте рішення винести функціонал копіювання в інший потік за допомогою спадкування від класу `AsyncTask` (див. рис. 3.6). Цей клас дозволяє виконувати фонові операції в окремому потоці і публікувати результати цих операцій для `UI thread`, без необхідності вручну маніпулювати потоками або обробниками.

```
public class CopyFileAsyncTask extends AsyncTask<Void, Void, Boolean> {
    public static final String TAG = "LogCopyFileAsyncTask";
    private String mCopyTo;
    private File mFileForCopy;
    @SuppressWarnings("StaticFieldLeak") private Context mContext;
    public CopyFileAsyncTask(String copyTo, File fileForCopy, Context context) {
        this.mCopyTo = copyTo;
        this.mFileForCopy = fileForCopy;
        this.mContext = context;
    }
    @Override
    protected Boolean doInBackground(Void... files) {
        try {
            File fileTargetLocation = new File(mCopyTo, mFileForCopy.getName());
            Log.d(TAG, "doInBackground: " + mCopyTo + " " + mFileForCopy.getName());
            InputStream in = new FileInputStream(mFileForCopy);
            OutputStream out = new FileOutputStream(fileTargetLocation);
            byte[] buf = new byte[1024];
            int len;
            while ((len = in.read(buf)) > 0) {
                out.write(buf, 0, len);
            }
            in.close();
            out.close();
            return true;
        } catch (IOException e) {
            Log.d(TAG, "doInBackground: " + e.getMessage());
        }
        return false;
    }
}
```

Рисунок 3.6 – Клас `CopyFileAsyncTask` який успадковується від `AsyncTask`

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови застосування розробленої програми

Розроблений застосунок файлового менеджера має назву "Simple File Manager". Програма призначена для користувачів мобільних пристроїв під керуванням ОС Android, яким потрібен простий та зручний файловий менеджер. Застосунок виконує стандартні функції файлових менеджерів: копіювання; перейменування; переміщення файлів та каталогів. Ця утиліта розроблена на мові Java у середовищі розробки AndroidStudio.

Галузь використання – файлова система ОС Android.

Системні вимоги до мобільного пристрою, на якому буде інстальоване розроблене програмне забезпечення, наступні:

- RAM 1024 Мб;
- частота процесору не менше 1300МГц;
- наявність вільного дискового простору – 2,5 Мб;
- версія Android вище ніж 4.1 JellyBean (API level– 16).

Функції, виконання яких забезпечує розроблений застосунок, наведено у п. А.3 Додатку А.

4.2 Характеристики програми

Програма застосунку складається з класів та методів наведених у таблиці 4.1.

Для модернізації даного програмного продукту потрібно використати фреймворк AndroidStudio, тому що графічний інтерфейс та структура зберігання даних побудовані виключно на бібліотеках Android SDK.

Для запуску проєкту необхідно запустити середовище AndroidStudio та обрати каталог проєкту SFMAApp. Далі потрібно буде налаштувати збірку проєкту і відкрити потрібний файл проєкту для редагування.

Таблиця 4.1 – Основні класи та методи програми

Назва класу, метода або поля	Опис
1	2
SFMAApp.java	Клас, який успадковується від Application, прописується у маніфесті, як клас "точки входу" до застосунку. Реалізує синглтон класу SFMAApp, та геттер для нього
MainActivity.java	Клас головної активності. Вона має властиві усім активностям універсальні методи. Також реалізує інтерфейс DocumentSelectListener для зв'язку з фрагментом FileManagerFragment
FileListAdapter.java	Клас-адаптер для ListView, який складається з масиву елементів ListItem та успадковується від BaseFragmentAdapter
ListItem.java	Клас-модель, поля якого описують файл або папку
DocumentSelectListener.java	Інтерфейс який містить методи для роботи з елементами списку ListView, логіка якого реалізується в FileManagerFragment
FileManagerFragment.java	Клас, де знаходиться основна логіка застосунку. Містить як загальні методи для фрагментів, так і додаткові, що описані далі
void listRootFolders()	Метод для отримання логічних дисків та кореневого каталогу, який містить методи initInternalStore(extStorageAbsPath) та initExternalStore(extStorageAbsPath для доступу до внутрішнього та зовнішнього сховищ

Продовження таблиці 4.1

1	2
<code>void initInternalStore(String extStorageAbsPath)</code>	Знаходить внутрішнє сховище за допомогою статичного метода класу <code>Environment.getExternalStorageDirectory()</code> .
<code>void initExternalStore(String extStorageAbsPath)</code>	Метод, що знаходить шлях до зовнішнього сховища. Він звертатиметься до файлу <code>proc/mounts</code> . Файл був зчитаний за допомогою класів <code>BufferedReader</code> та <code>FileReader</code> . Шляхом виключення, було отримано масив <code>String</code> -елементів який зберігає шляхи до усіх вмонтованих <code>storage</code> -пристроїв
<code>boolean listFiles(File dir)</code>	Знаходить усі файли й каталоги, що знаходяться у папці <code>dir</code> . Якщо доступу до каталогу немає – повертає <code>false</code> .
<code>boolean openFile(int position)</code>	Відкриває файли за допомогою застосунків встановлених на мобільному пристрою. Повертає <code>false</code> якщо немає доступу для читання, або система не має застосунку для відкриття файлу
<code>void showRenameDialog (int position)</code>	Відкриває діалог для перейменування файлу. Якщо перейменувати неможливо, виводиться відповідне повідомлення
<code>void showDeleteDialog(int position)</code>	Відкриває діалог для підтвердження видалення файлу. Якщо видалити неможливо, виводиться відповідне повідомлення

Продовження таблиці 4.1

1	2
List<String> getDescriptionFile(File file)	Повертає масив String, який містить інформацію про файл. Використовується методом showInfoBox(List <String> message) для відображення інформації про файл в діалоговому вікні
void showInfoBox(List <String> message)	Відкриває діалогове вікно, що показує детальну інформацію про файл
void copyFile(int position)	Відкриває діалог для копіювання файлу за допомогою класу CopyFileAsyncTask. Його опис наведено нижче.
void moveFile(final int position)	Має схожий функціонал з методом showRenameDialog ()
void mkdir()	Відкриває діалог для створення нового каталогу, використовує функцію mkdir() класу File
CopyFileAsyncTask.java	Клас для виконання функції копіювання в іншому потоці. Успадковується від класу AsyncTask. Використовує класи FileInputStream та FileOutputStream для запису та зчитування потоку байтів файлу

4.3 Звернення до програми

Для запуску застосунку необхідно, спочатку авантажити інсталятор програми у вигляді арк-файл "sfm-release.apk", а потім виконати запуск програми шляхом натискання на відповідну іконку застосунку.

4.4 Вхідні та вихідні дані

Вхідними даними застосунку є:

- подія натискання іконки запуску програми файлового менеджера;
- події натискання на іконки каталогів та файлів, з яким буде працювати користувач;
- події натискання на пункти контекстного меню (з'являються коли користувач натискає на елемент списку) для виконання функцій вилучення, перейменування, переміщення або запуску файлу.

Вихідні дані застосунку:

- детальна інформація про файл або каталог;
- відкриття діалогового вікна для перейменування файлу або каталогу;
- відкриття діалогового вікна для підтвердження вилучення файлу або каталогу;
- відкриття діалогового вікна для перейменування та копіювання файлу або каталогу;
- запуск файлу у тому випадку, коли система має застосунок, що може відкрити даний файл, та відкриває вікно для вибору застосунку якщо система має декілька таких застосунків.

4.5 Повідомлення

У програмі не було передбачено повідомлень для програміста.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Файловий менеджер "Simple File Manager" призначений для забезпечення зручної роботи з файлами та надає можливість користувачу копіювати, перемішувати, перейменовувати файли або каталоги, а також отримувати інформацію про файл або каталог. Програма працює майже з усіма типами файлів, які можуть знаходитись у смартфоні та працює на усіх пристроях з версією Android вище 4.1.

Даний застосунок надає можливість користувачу виконувати функції, що наведені у п. А.3 Додатку А.:

5.2 Умови використання програми

Завдяки тому, що утиліта "Simple File Manager" розроблена на мові Java, вона є кросплатформовою, тобто може виконуватися на пристроях під керуванням операційних систем Android, ОС сімейства Linux та Mac OS.

Системні вимоги до мобільного пристрою, на якому можна безперешкодно експлуатувати дану програму наступні:

- RAM 1024 Мб;
- частота процесору не менше 1300 МГц;
- наявність вільного дискового простору – 2,5 Мб.

5.3 Виконання програми

Для першого запуску програми потрібно виконати наступні дії:

- упевнитися у відповідності характеристик смартфона, на яку встановлюється програма, мінімальним системним вимогам, наведеним у попередньому пункті;
- упевнитися у справності всіх компонентів пристрою;

- завантажити інсталятор програми у вигляді арк-файлу "sfm-release.apk";
- виконати запуск програми шляхом запуску через натискання на іконку "Simple File Manager" у меню застосунків (див рис.5.1).

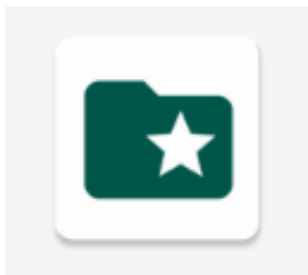


Рисунок 5.1 – Іконка застосунку

Якщо застосунок вперше запускається на пристрої з ОС Android 6 та вище, з'являється діалогове вікно для підтвердження доступу до файлової системи (див рис. 5.2).

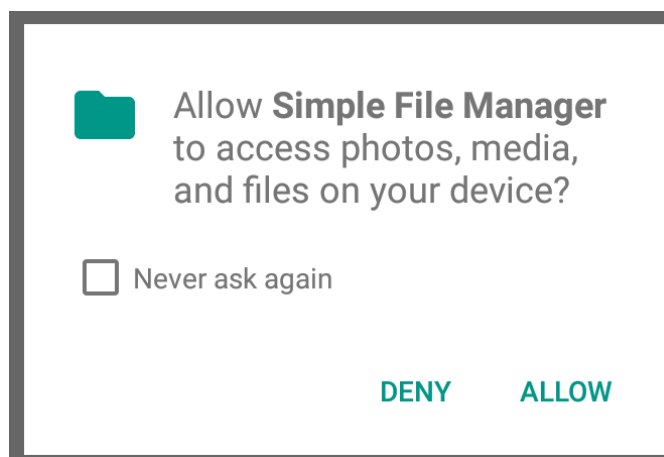


Рисунок 5.2 – Діалогове вікно для підтвердження доступу до ФС

Якщо користувач натискає кнопку "DENY", програма завершується. У цьому випадку переходу до головного вікна застосунку не буде. Щоб все ж таки дати доступ до ФС, слід виконати наступні дії: Settings -> Apps -> Simple File Manager -> Permissions -> Поставити галочку напроти "Storage".

Після запуску програми з'являється головна форма, що наведена на рис. 5.3. Вона ж є домашньою сторінкою.

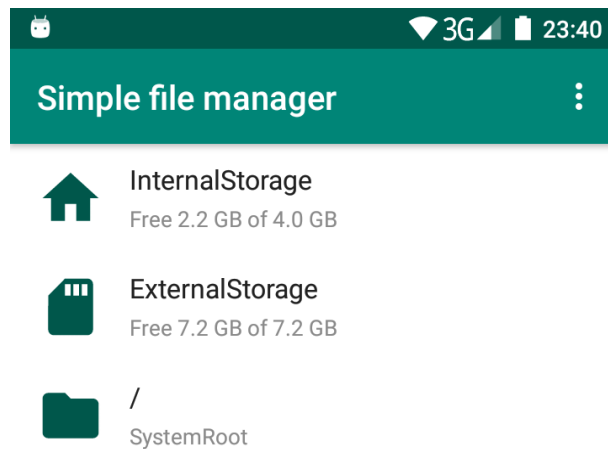


Рисунок 5.3 – Головна форма застосунку

Після натискання користувачем на іконці потрібного каталогу, відкриється його вміст (див. рисунок 5.4).

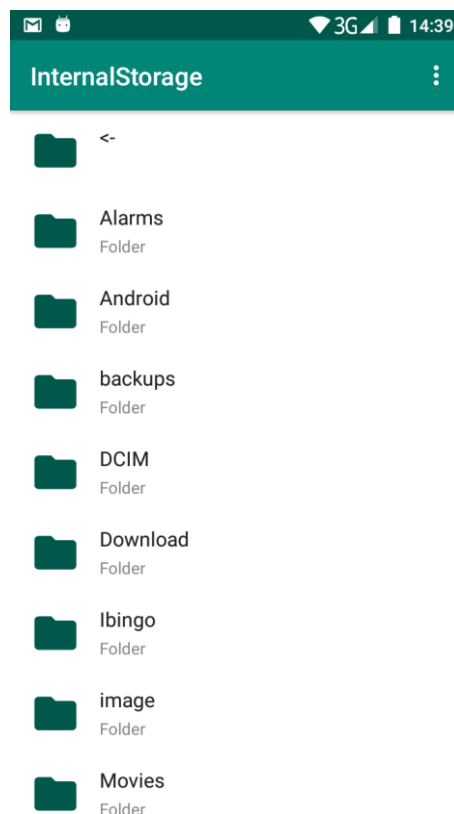


Рисунок 5.4– Приклад відкриття обраного каталогу

Засобами контекстного меню можна видаляти, перейменовувати, копіювати, запускати файл або каталог та переглядати інформацію про них. Приклад роботи з цими функціями наведено на рис 5.5 – 5.8.

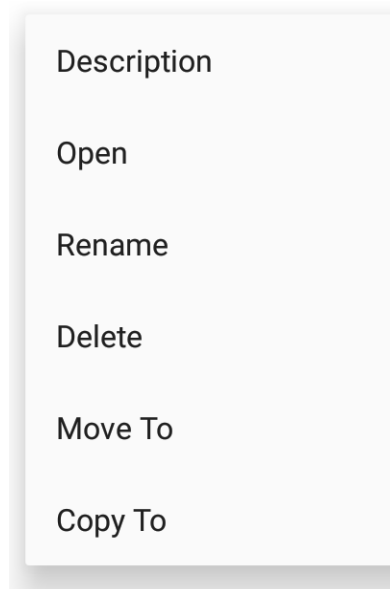


Рисунок 5.5 – Контекстне меню для роботи з файлом або каталогом

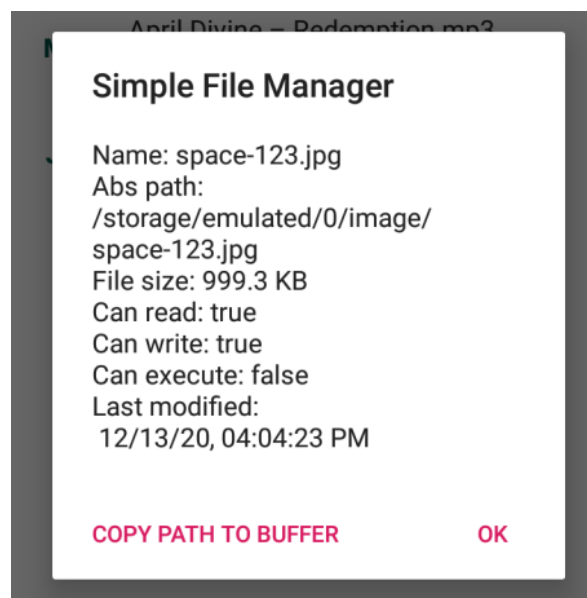


Рисунок 5.6 – Інформація про файл, яка з'являється після натискання пункту "Description"

Також вікно "Description" можна відкрити шляхом тривалого натискання (впродовж 2-3 секунди) на іконку файлу або каталогу. Як можна побачити,

вікно має кнопку "Copy path to buffer", що зручно при використанні функцій переміщення та копіювання.

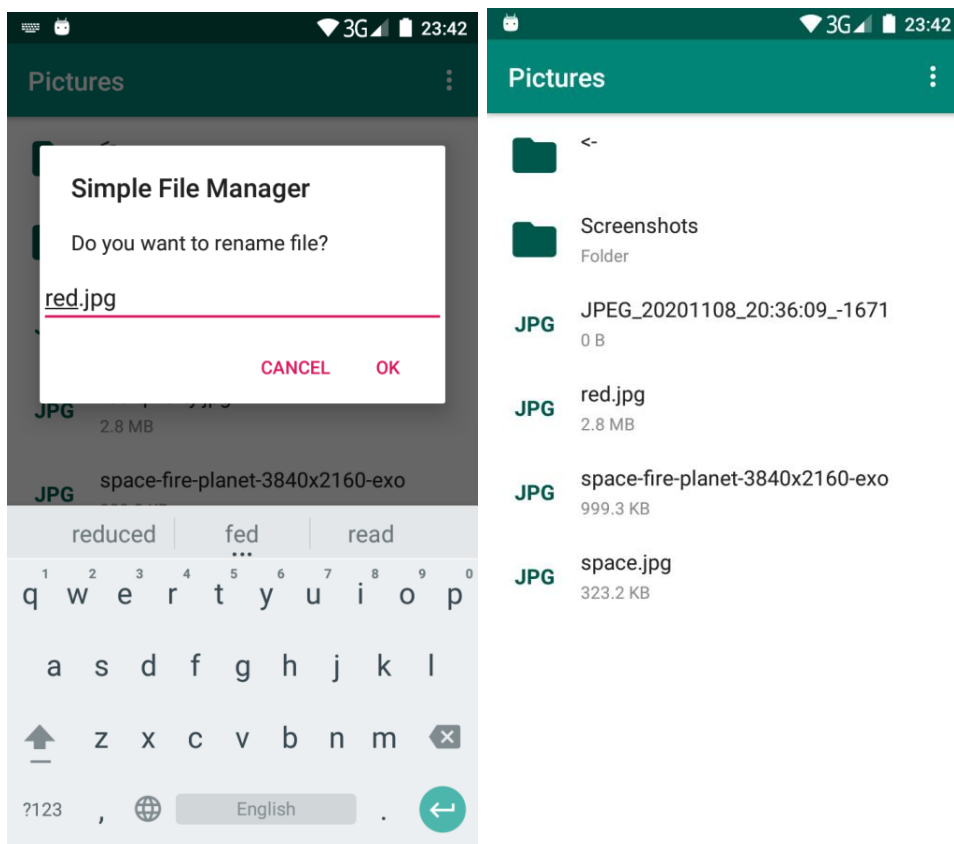


Рисунок 5.7 – Перейменування файлу, після натискання пункту "Rename"

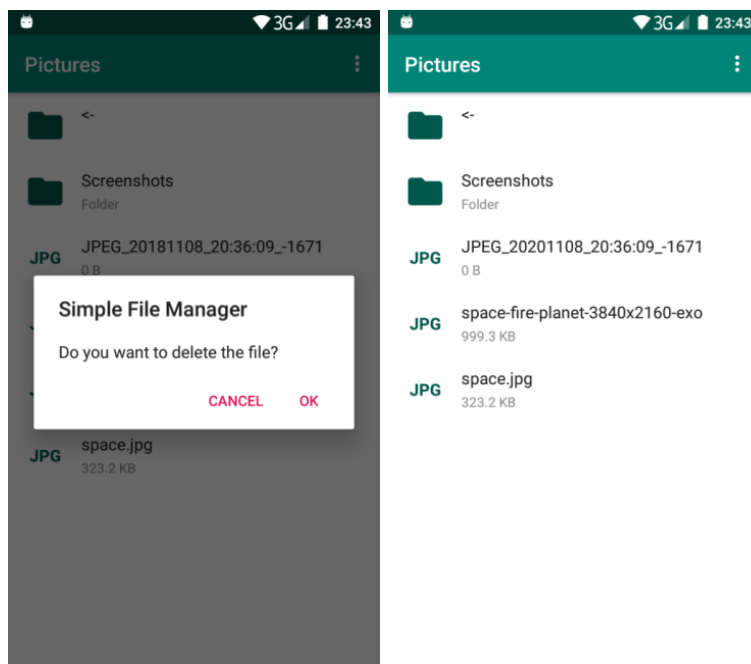


Рисунок 5.8 – Видалення файлу, після натискання пункту "Delete"

Діалогові вікна для переміщення та копіювання файлів або каталогів наведені на рис. 5.9. Як можна побачити, вони мають однаковий вигляд, але виконують різні функції. Використовуючи функцію переміщення, якщо переміщення вдале, користувальницький інтерфейс здійснює перехід до каталогу куди було здійснене переміщення файлу. У випадку з функцією копіювання, користувач отримує Toast-повідомлення або про шлях куди було здійснене копіювання, або про неможливість копіювання до вказаного каталогу (див. рис. 5.10).

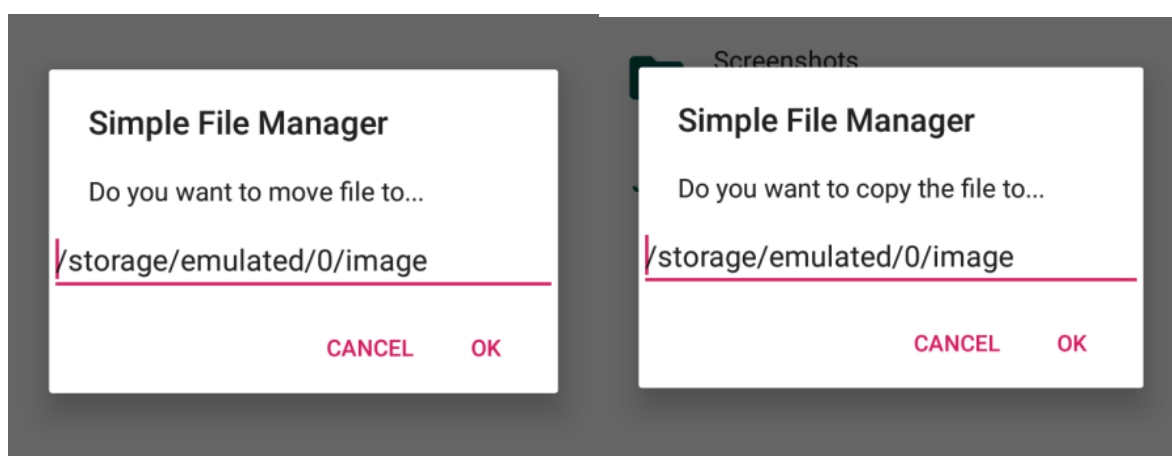


Рисунок 5.9 – Діалогові вікна копіювання та переміщення файлу

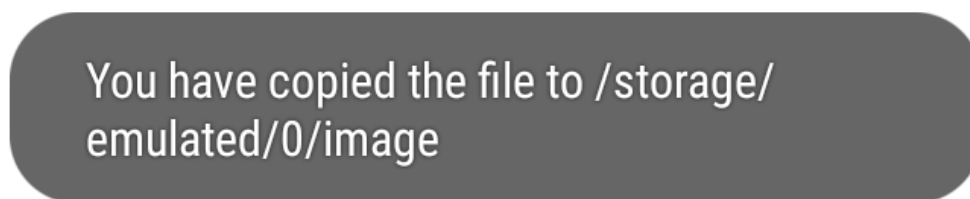


Рисунок 5.10 – Приклад Toast-повідомлення після копіювання файлу

Для створення нового каталогу, слід перейти до каталогу, де необхідно створити новий, далі натиснути на OptionMenu (являє собою три крапки) у Toolbar, далі – на пункт меню "Create new folder" та ввести ім'я нового файлу (див. рис. 5.11 та 5.12).

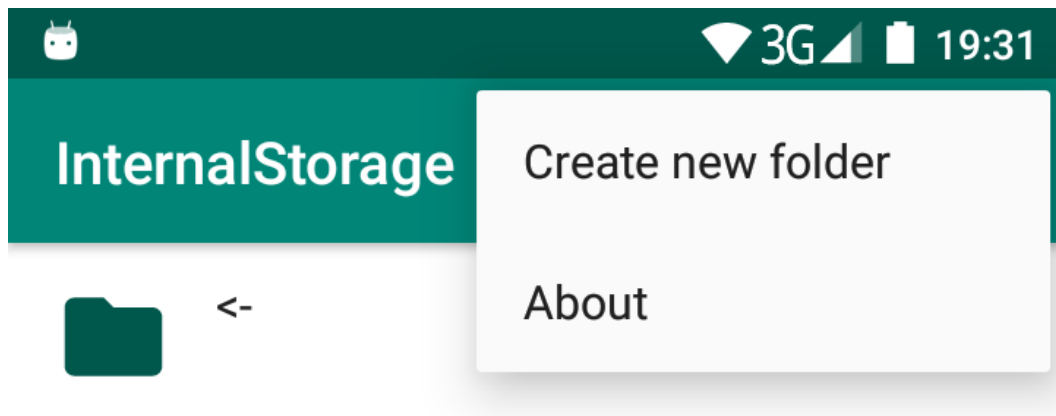


Рисунок 5.11 – OptionMenu у Toolbar

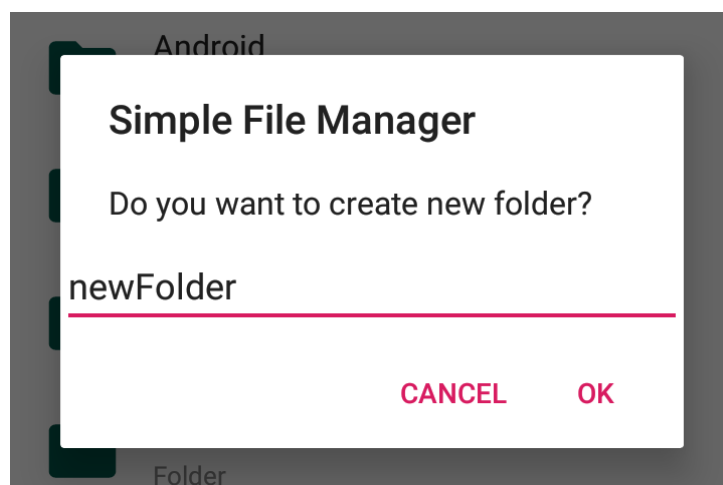


Рисунок 5.12 – Створення нового каталогу

При натисканні на пункт меню "About" відкривається нова форма зі стислою інформацією про застосунок (див. рис. 5.13).

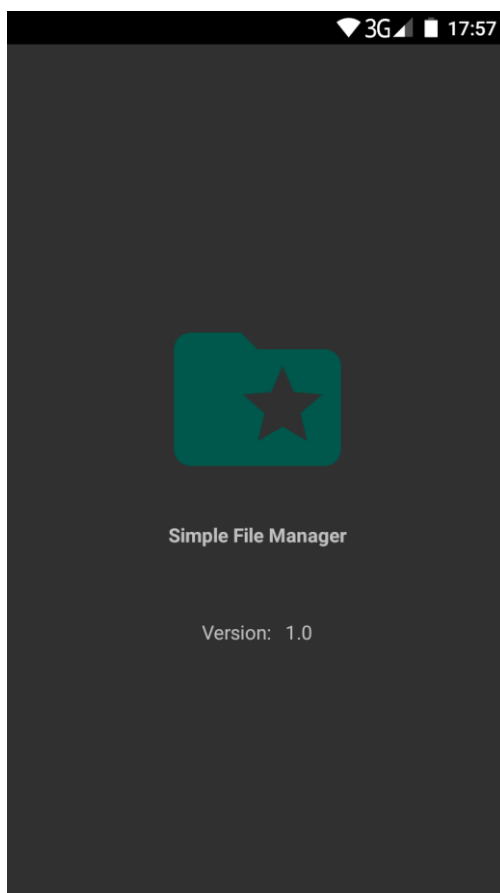


Рисунок 5.13 – Стисла інформація про застосунок

5.4 Повідомлення оператору

Під час роботи програми користувач-оператор може отримувати повідомлення. У випадку, якщо доступу до файлу або каталогу немає, користувач отримує повідомлення, які зображені на рис 5.14.

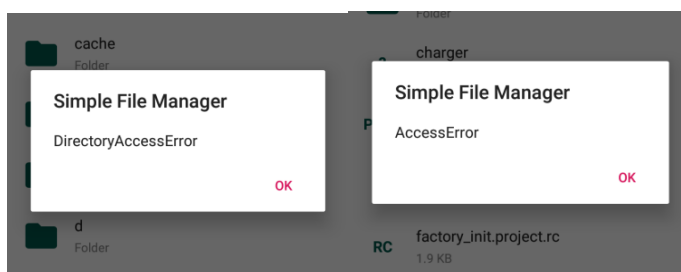


Рисунок 5.14 – Повідомлення про відсутність можливості читати файли або каталоги

Це може трапитися через те, що користувач не має прав адміністратора та намагається потрапити до системних каталогів або файлів які не рекомендується читати (для захисту персональних даних) та редагувати (щоб не зіпсувати роботу системи).

ВИСНОВКИ

Для постановки завдань роботи для даної дипломної кваліфікаційної роботи бакалавра було проведено огляд особливостей ОС Android, основних файлових систем, що використовуються в ОС Android та програмних аналогів.

Виходячі з результатів проведених оглядів, було розроблено технічне завдання на розробку програми ФМ. Мовою програмування застосунку було обрано мову Java, адже саме вона підтримується компанією Google, а більшість застосунків в Google Play побудовані теж на Java.

У якості середовища розробки було обрано профільне середовище Java розробки для ОС Android – Android Studio, яке має відкритий вихідний код і тому є безкоштовним.

Створений програмний продукт "Simple File Manager", надає можливість проводити прості операції з файлами та каталогами мобільних Android-пристроїв: копіювання; переміщення; видалення; перейменування, запуск (перегляд вмісту для каталогів) та перегляд детальної інформації про файл або каталог.

Розроблено комплект програмної документації, що включає в себе, опис програми, керівництво програміст та керівництво оператора.

Застосунок "Simple File Manager" може експлуатуватись у якості файлового менеджера в операційній системі Android вище ніж 4.1 JellyBean.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Иерархия папок в Android – что в них есть и где мои файлы [Электрон. ресурс]. – Режим доступа: <https://protabletpc.ru/advice/struktura-naznachenie-faylov-papok-v-android.html>.
2. Operating System Market Share Worldwide - January 2021 [Electronic resource]. – Access mode: <https://gs.statcounter.com/os-market-share>.
3. Mobile Operating System Market Share Worldwide - January 2021 [Electronic resource]. – Access mode: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
4. Как работает Android, часть 1 [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/company/solarsecurity/blog/334796/>.
5. Triggs, R. Making sense of your Android phone's storage folders [Electronic resource] / R. Triggs. – Access mode: <https://www.androidauthority.com/author/robtriggs/>.
6. Файловые системы Android [Электрон. ресурс]. – Режим доступа: <https://losst.ru/fajlovye-sistemy-android>.
7. Нестеренко, В. Redmi Note 5 на Android 9 с файловой системой f2fs. Стоит ли переходить? [Электрон. ресурс] / В. Нестеренко. – Режим доступа: https://ru-mi.com/blog/redmi_note_5_na_android_9_s_faylovoy_sistemoy_f2fs.html.
8. Структура и назначение папок и файлов в Android [Электрон. ресурс]. – Режим доступа: <https://www.infoconnector.ru/struktura-i-naznachenie-papok-i-fajlov-v-android>.
9. Лучшие файловые менеджеры для Android [Электрон. ресурс]. – Режим доступа: <https://remontka.pro/android-file-managers/>.
10. ТОП-10 лучших файловых менеджера для Андроид в 2020 году [Электрон. ресурс]. – Режим доступа: <https://xpc.com.ru/luchshiy-faylovyy-menedzher-dlya-android.html#Total-Commander>.
11. Root Explorer 4.8.4 [Электрон. ресурс]. – Режим доступа: <https://android.biblprog.org.ua/ru/root-explorer/>.

12. Класс File. Работа с файлами и каталогами [Электрон. ресурс]. – Режим доступа: <https://metanit.com/java/tutorial/6.11.php>.

13. Менеджер Файлов и Проводник АСТРО [Электрон. ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=com.metago.astro&hl=ru&gl=US>.

14. Якимова, Е. На чём пишут приложения для Android [Электрон. ресурс] / Е. Якимова. – Режим доступа: <https://livotyping.com/ru/blog/na-chem-pishut-prilozhenija-pod-android>.

15. Как выбрать язык программирования для создания Андроид – приложения [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/post/477578/>.

16. Java – Overview [Electronic resource]. – Access mode: https://www.tutorialspoint.com/java/java_overview.htm.

17. What is Java? Definition, Meaning & Features of Java Platforms [Electronic resource]. – Access mode: <https://www.guru99.com/java-platform.html>.

18. Лучшие IDE для Java [Электрон. ресурс]. – Режим доступа: https://geekbrains.ru/posts/java_ides.

19. Как скопировать файл в Java? [Электрон. ресурс]. – Режим доступа: <https://javadevblog.com/kak-skopirovat-fajl-v-java-4-sposoba-primery-i-kod.html>.

20. Activity (Активность, Деятельность) [Электрон. ресурс]. – Режим доступа: <http://developer.alexanderklimov.ru/android/theory/activity-theory.php>.

ДОДАТОК А
Технічне завдання

A.1 Підстава для розробки

Підставою для розробки служить завдання на дипломну кваліфікаційну роботу бакалавра на тему "Розробка програми файлового менеджера для операційної системи Android", затверджене наказом № 103 від 30 березня 2021 р. по НУ «Запорізька політехніка».

A.2 Мета розробки та її призначення

Метою розробки є створення файлового менеджера, який функціонує під керуванням ОС Android. Застосунок повинен виконувати стандартні функції файлових менеджерів: копіювання, перейменування, переміщення файлів та каталогів.

Програма призначена для користувачів мобільних пристроїв з ОС Android, яким потрібен простий та зручний ФМ.

A.3 Основні вимоги до програмного продукту

Відповідно з проведеним у п. 1.3 оглядом, можна сформулювати вимоги до застосунку «Simple File Manager», що розробляється.

Застосунок повинен забезпечувати виконання наступних функцій та мати такі властивості:

- мати інтуїтивно-зрозумілий інтерфейс;
- надавати користувачу зручні можливості для вибору файлу або каталогу;
- відображати всі логічні диски мобільного пристрою;
- перегляд вмісту обраного логічного диску;
- відображати типи файлів;
- відображати інформацію щодо файлу (назва файлу, тип файлу, розмір, дата створення, дата зміни);

- створення нових каталогів;
- створення простих текстових файлів;
- переміщення, копіювання, видалення файлів та каталогів.

Вхідними даними для застосунку є керуючі дії користувача: події натискання іконки запуску програми файлового менеджера; натискання на іконки каталогів та файлів, з яким буде працювати користувач.

Вихідними даними застосунку є зміни у стані файлу або каталогу (запуск файлу/відкриття каталогу, зміни місця розташування файлу/каталогу, перейменування, копіювання та зміни властивостей файлу/каталогу).

А.4 Вимоги до видів забезпечення та компонентів

Застосунок повинен працювати під керуванням операційної системи Android вище ніж версія 4.1 JellyBean (API level – 16).

Для створення програми доцільно використовувати мову програмування Java та середовище розробки AndroidStudio (офіційне середовище для Android пристроїв).

Системні вимоги до мобільного пристрою, на якому можна безперешкодно експлуатувати дану програму наступні:

- RAM 1024 Мб;
- частота процесору не менше 1300 МГц;
- наявність вільного дискового простору – 2,5 Мб.

А.5 Визначені системні функції, що мають бути використані у застосунку

Робота програми повинна розпочинатися з запуску програми шляхом натискання іконки запуску. Далі користувач обирає потрібний логічний диск, а потім – каталог. Для відкриття потрібного каталогу натискає на його іконку.

Для копіювання, переміщення, видалення та перейменування файлів

користувач використовує відповідні команди з головного вікна.

Програма повинна також надавати можливість користувачу зручно працювати з сенсорним екраном.

A.6 Вимоги до надійності

Надійне функціонування програми має бути забезпечене виконанням сукупності організаційно-технічних заходів, перелік яких наведено нижче:

- використання мобільного пристрою, зі встановленою операційною системою Android не нижче версія 4.1;
- наявність надійного акумулятора;
- наявність коректно працюючого сенсору;
- наявність анти-вірусної програми на мобільному пристрої.

A.7 Стадії й етапи розробки

При розробці програмного продукту будуть проведені наступні стадії, які перелічені нижче.

Розробка технічного завдання, що включає в себе виконання наступних етапів: розроблення, погодження та затвердження технічного завдання.

Робоче проєктування включає в себе виконання наступних етапів:

- розробка програми (програмування і налагодження програми);
- розробка програмної документації (розробка документів «Керівництво програміста» та «Керівництво оператора»);
- випробування програми;
- коригування програми та програмної документації за результатами випробувань.

Впровадження включає в себе виконання наступного етапу: підготовка і передача програми (повинна бути виконана робота по підготовці і передачі програми і програмної документації в експлуатацію)..

A.8 Вимоги до програмної документації

Програмна документація повинна включати: керівництво програміста, керівництво оператора, вихідний код програми.

ДОДАТОК Б
Текст програми

Б.1 Клас com.alburd.sfmapp.ui.activities.MainActivity

```

package com.alburd.sfmapp.ui.activities;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import com.alburd.sfmapp.R;
import com.alburd.sfmapp.ui.fragments.FileManagerFragment;
import com.alburd.sfmapp.ui.fragments.interfaces.DocumentSelectListener;
import java.util.ArrayList;
public class MainActivity extends AppCompatActivity implements
DocumentSelectListener {
    private ActionBar mActionBar;
    private FragmentManager mFragmentManager;
    private FragmentTransaction mFragmentTransaction;
    private FileManagerFragment mDirectoryFragment;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mActionBar = getSupportActionBar();
        if (mActionBar != null) {
            mActionBar.setTitle("Simple file manager");
        }
        mFragmentManager = getSupportFragmentManager();
        mFragmentTransaction = mFragmentManager.beginTransaction();
        mDirectoryFragment = new FileManagerFragment();
        mFragmentTransaction.replace(R.id.fragment_container,
mDirectoryFragment, mDirectoryFragment.toString());
        mFragmentTransaction.commit();
    }
    @Override
    public void onBackPressed() {
        if (mDirectoryFragment.onBackPressed()) {
            super.onBackPressed();
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater menuInflater = getMenuInflater();
        menuInflater.inflate(R.menu.main, menu);
        return super.onCreateOptionsMenu(menu);
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.file_manager_menu_create_new_folder:
                makeDir();
                break;
        }
    }
}

```

```

        case R.id.file_manager_menu_about:
            Intent intent = new Intent(this, AboutActivity.class);
            startActivity(intent);
            break;
    }
    return super.onOptionsItemSelected(item);
}
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    MenuItem createNewFolder =
menu.findItem(R.id.file_manager_menu_create_new_folder);
    if (mDirectoryFragment.getCurrentDir() != null) {
        createNewFolder.setVisible(true);
    } else {
        createNewFolder.setVisible(false);
    }
    return super.onPrepareOptionsMenu(menu);
}
@Override
public void startDocumentSelectActivity(View view, int position) {
    mDirectoryFragment.openPopupMenu(view, position);
}
@Override
public void didSelectFiles(FileManagerFragment activity,
                           ArrayList<String> files) {
    mDirectoryFragment.showInfoDialog(files);
}
@Override
public void updateAppBarName(String name) {
    if (name.isEmpty()) {
        mActionBar.setTitle("Simple file manager");
    } else {
        mActionBar.setTitle(name);
    }
}
@Override
public void makeDir() {
    mDirectoryFragment.mkDir();
}
}

```

Б.2 Клас com.alburd.sfmapp.ui.fragments.FileManagerFragment

```

package com.alburd.sfmapp.ui.fragments;
import android.app.AlertDialog;
import android.content.BroadcastReceiver;
import android.content.ClipData;
import android.content.ClipboardManager;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Environment;
import android.os.StatFs;

```

```

import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.view.GravityCompat;
import android.support.v7.widget.PopupMenu;
import android.text.TextUtils;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;
import com.alburd.sfmapp.R;
import com.alburd.sfmapp.data.asynctask.CopyFileAsyncTask;
import com.alburd.sfmapp.ui.adapters.FileListAdapter;
import com.alburd.sfmapp.ui.adapters.adaptermodels.ListItem;
import com.alburd.sfmapp.ui.fragments.helpers.FormatString;
import com.alburd.sfmapp.ui.fragments.helpers.HistoryEntry;
import com.alburd.sfmapp.ui.fragments.interfaces.DocumentSelectListener;
import com.alburd.sfmapp.ui.fragments.layouts.AndroidUtil;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
public class FileManagerFragment extends Fragment {
    private static final String LOG_FILE_MANAGER_FRAGMENT = "LogFileManagerFragment";
    private static String sTitleForUpdate = "";
    private Context mContext;
    private View mFileManagerView;
    private ArrayList<HistoryEntry> mHistory = new ArrayList<>();
    private ListView mListView;
    private ArrayList<ListItem> mItems = new ArrayList<>();
    private FileListAdapter mListAdapter;
    private DocumentSelectListener mListener;
    private boolean isBroadcastReceiverRegistered;
    private File mCurrentDir;
    private long mSizeLimit = 1024 * 1024 * 1024;
    private BroadcastReceiver mBroadcastReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context arg, Intent intent) {
            Runnable r = new Runnable() {
                public void run() {
                    try {
                        if (mCurrentDir == null) {
                            listRootFolders();
                        } else {
                            listFiles(mCurrentDir);
                        }
                    } catch (Exception e) {

```

```

        Log.e(LOG_FILE_MANAGER_FRAGMENT, e.getMessage());
    }
}
};
if (Intent.ACTION_MEDIA_UNMOUNTED.equals(intent.getAction())) {
    mListView.postDelayed(r, 1000);
} else {
    r.run();
}
}
};
public File getCurrentDir() {
    return mCurrentDir;
}
private void updateTitleName(String title) {
    if (mListener != null) {
        mListener.updateAppBarName(title);
    }
}
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    mContext = context;
    if (context instanceof DocumentSelectListener) {
        mListener = (DocumentSelectListener) context;
    } else {
        throw new RuntimeException(context.toString()
            + " must implement DocumentSelectListener");
    }
}
@Override
public void onDestroy() {
    super.onDestroy();
    if (isBroadcastReceiverRegistered) {
        mContext.unregisterReceiver(mBroadcastReceiver);
    }
    mListener = null;
}
@Override
public View onCreateView(@NonNull LayoutInflater inflater,
    ViewGroup container,
    Bundle savedInstanceState) {
    if (!isBroadcastReceiverRegistered) {
        isBroadcastReceiverRegistered = true;
        IntentFilter filter = getIntentFilter();
        mContext.registerReceiver(mBroadcastReceiver, filter);
    }
    if (mFileManagerView == null) {
        mFileManagerView = inflater.inflate(
            R.layout.document_select_layout,
            container,
            false);
        mAdapter = new FileListAdapter(mContext, mItems);
        mListView = mFileManagerView.findViewById(R.id.listView);
        mListView.setAdapter(mAdapter);
        mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

```

```

        @Override
        public void onItemClick(AdapterView<?> adapterView, View view, int i,
long l) {
            chooseItem(view, i);
        }
    });
    mListView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
        @Override
        public boolean onItemLongClick(AdapterView<?> adapterView, View view,
int i, long l) {
            return showInfoAbout(i);
        }
    });
    listRootFolders();
} else {
    ViewGroup parent = (ViewGroup) mFileManagerView.getParent();
    if (parent != null) {
        parent.removeView(mFileManagerView);
    }
}
return mFileManagerView;
}
private IntentFilter getIntentFilter() {
    IntentFilter filter = new IntentFilter();
    filter.addAction(Intent.ACTION_MEDIA_BAD_REMOVAL);
    filter.addAction(Intent.ACTION_MEDIA_CHECKING);
    filter.addAction(Intent.ACTION_MEDIA_EJECT);
    filter.addAction(Intent.ACTION_MEDIA_MOUNTED);
    filter.addAction(Intent.ACTION_MEDIA_NOFS);
    filter.addAction(Intent.ACTION_MEDIA_REMOVED);
    filter.addAction(Intent.ACTION_MEDIA_SHARED);
    filter.addAction(Intent.ACTION_MEDIA_UNMOUNTABLE);
    filter.addAction(Intent.ACTION_MEDIA_UNMOUNTED);
    filter.addDataScheme("file");
    return filter;
}
private void chooseItem(View view, int i) {
    if (i < 0 || i >= mItems.size()) {
        return;
    }
    HistoryEntry history;
    ListItem item = mItems.get(i);
    File file = item.getFile();
    if (file == null) {
        history = mHistory.remove(mHistory.size() - 1);
        sTitleForUpdate = history.getTitle();
        updateTitleName(sTitleForUpdate);
        if (history.getDirectory() != null) {
            listFiles(history.getDirectory());
        } else {
            listRootFolders();
        }
        mListView.setSelectionFromTop(history.getScrollItem(),
history.getScrollOffset());
    } else if (file.isDirectory()) {

```

```

        history = new HistoryEntry();
        history.setScrollItem(mListView.getFirstVisiblePosition());
        history.setScrollOffset(mListView.getChildAt(0).getTop());
        history.setDirectory(mCurrentDir);
        history.setTitle(sTitleForUpdate);
        updateTitleName(sTitleForUpdate);
        if (!listFiles(file)) {
            return;
        }
        mHistory.add(history);
        sTitleForUpdate = item.getTitle();
        updateTitleName(sTitleForUpdate);
        mListView.setSelection(0);
    } else {
        if (!file.canRead()) {
            showErrorInfoBox("AccessError");
        } else if (file.length() > mSizeLimit) {
            showErrorInfoBox("FileUploadLimit");
        } else if (!file.isDirectory()) {
            if (mListener != null) {
                mListener.startDocumentSelectActivity(view, i);
            }
        } else {
            showErrorInfoBox("Choose correct file");
        }
    }
}

private boolean showInfoAbout(int i) {
    if (i < 0 || i >= mItems.size()) {
        return false;
    }
    ListItem item = mItems.get(i);
    File file = item.getFile();
    if (file != null) {
        if (!file.canRead()) {
            showErrorInfoBox("AccessError");
        } else if (file.length() > mSizeLimit) {
            showErrorInfoBox("FileUploadLimit");
        } else if (mListener != null) {
            mListener.didSelectFiles(FileManagerFragment.this,
getDescriptionFile(file));
            return true;
        } else {
            showErrorInfoBox("Choose correct file.");
            return false;
        }
    }
    return false;
}

private void showErrorInfoBox(String error) {
    if (mContext == null) {
        return;
    }
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage(error).setPositiveButton("OK", null).show();
}

```

```

    }
    private ArrayList<String> getDescriptionFile(File file) {
        ArrayList<String> data = new ArrayList<>();
        data.add("Name: " + file.getName());
        data.add("Abs path:\n" + file.getAbsolutePath());
        data.add("File size: " + FormatString.formatFileSize(file.length()));
        data.add("Can read: " + String.valueOf(file.canRead()));
        data.add("Can write: " + String.valueOf(file.canWrite()));
        data.add("Can execute: " + String.valueOf(file.canExecute()));
        data.add(String.format("Last modified:\n %tD, %<tr", new
Date(file.lastModified())));
        return data;
    }
    private void listRootFolders() {
        mCurrentDir = null;
        mItems.clear();
        String extStorageAbsPath =
Environment.getExternalStorageDirectory().getAbsolutePath();
        initInternalStore(extStorageAbsPath);
        initExternalStore(extStorageAbsPath);
        ListItem fs = new ListItem();
        fs.setTitle("/");
        fs.setSubtitle("SystemRoot");
        fs.setIcon(R.drawable.ic_directory);
        fs.setFile(new File("/"));
        mItems.add(fs);
        mAdapter.notifyDataSetChanged();
    }
    private void initInternalStore(String extStorageAbsPath) {
        ListItem internalStorageItem = new ListItem();
        internalStorageItem.setTitle("InternalStorage");
        internalStorageItem.setIcon(R.drawable.ic_home_storage);
        internalStorageItem.setSubtitle(getInfoAboutFileSystemSpace(extStorageAbsPath));
        internalStorageItem.setFile(Environment.getExternalStorageDirectory());
        mItems.add(internalStorageItem);
    }
    private void initExternalStore(String extStorageAbsPath) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader("/proc/mounts"));
            String infoAboutMountDevices;
            ArrayList<String> result = new ArrayList<>();
            while ((infoAboutMountDevices = reader.readLine()) != null) {
                if ((!infoAboutMountDevices.contains("/storage"))
                    || infoAboutMountDevices.contains("asec")
                    || infoAboutMountDevices.contains("tmpfs")
                    || infoAboutMountDevices.contains("none")) {
                    continue;
                }
                String[] info = infoAboutMountDevices.split(" ");
                if (!extStorageAbsPath.contains(info[1])) {
                    result.add(info[1]);
                }
            }
            reader.close();
            for (String path : result) {

```

```

        ListItem item = new ListItem();
        item.setTitle("ExternalStorage");
        item.setIcon(R.drawable.ic_external_storage);
        item.setSubtitle(getInfoAboutFileSystemSpace(path));
        File extStorage = new File(path);
        item.setFile(extStorage);
        mItems.add(item);
    }
} catch (Exception e) {
    Log.e(LOG_FILE_MANAGER_FRAGMENT, e.getMessage());
}
}

private boolean listFiles(File dir) {
    if (!dir.canRead()) {
        if
        (dir.getAbsolutePath().startsWith(Environment.getExternalStorageDirectory().toString())
            &&
            !Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)
            &&
            !Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
            mCurrentDir = dir;
            mItems.clear();
            AndroidUtil.clearDrawableAnimation(mListView);
            mListAdapter.notifyDataSetChanged();
            return true;
        }
        showErrorInfoBox("DirectoryAccessError");
        return false;
    }
    File[] files;
    try {
        files = dir.listFiles();
    } catch (Exception e) {
        showErrorInfoBox(e.getLocalizedMessage());
        return false;
    }
    if (files == null) {
        showErrorInfoBox("UnknownError");
        return false;
    }
    mCurrentDir = dir;
    mItems.clear();
    sortFiles(files);
    initListItems(files);
    initFolderBack();
    mListAdapter.notifyDataSetChanged();
    return true;
}

private void sortFiles(File[] files) {
    Arrays.sort(files, new Comparator<File>() {
        @Override
        public int compare(File lhs, File rhs) {
            if (lhs.isDirectory() != rhs.isDirectory()) {
                return lhs.isDirectory() ? -1 : 1;
            }
            return lhs.getName().compareToIgnoreCase(rhs.getName());
        }
    });
}

```

```

    });
}
private void initListItems(File[] files) {
    for (File file : files) {
        if (file.getName().startsWith(".")) {
            continue;
        }
        ListItem item = new ListItem();
        item.setTitle(file.getName());
        item.setFile(file);
        if (file.isDirectory()) {
            item.setIcon(R.drawable.ic_directory);
            item.setSubtitle("Folder");
        } else {
            String fileName = file.getName();
            String[] sp = fileName.split("\\.");
            String extension = sp.length > 1 ? sp[sp.length - 1] : "?";
            item.setExtension(extension);
            item.setSubtitle(FormatString.formatFileSize(file.length()));
        }
        mItems.add(item);
    }
}
private void initFolderBack() {
    ListItem item = new ListItem();
    item.setTitle("<-");
    item.setSubtitle("");
    item.setIcon(R.drawable.ic_directory);
    item.setFile(null);
    mItems.add(0, item);
    AndroidUtil.clearDrawableAnimation(mListView);
}
private String getInfoAboutFileSystemSpace(String path) {
    StatFs stat = new StatFs(path);
    long total = (long) stat.getBlockCount() * (long) stat.getBlockSize();
    long free = (long) stat.getAvailableBlocks() * (long) stat.getBlockSize();
    if (total == 0) {
        return null;
    }
    return "Free " + FormatString.formatFileSize(free) + " of " +
FormatString.formatFileSize(total);
}
public static final String PATH = "path to file";
public void showInfoDialog(final List<String> message) {
    if (mContext == null) {
        return;
    }
    StringBuilder stringBuilder = new StringBuilder();
    for (String s : message) {
        stringBuilder.append(s);
        stringBuilder.append("\n");
    }
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage(stringBuilder.toString())

```

```

        .setNeutralButton("Copy path to buffer", new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        ClipboardManager clipboardManager =
            (ClipboardManager)
mContext.getSystemService(Context.CLIPBOARD_SERVICE);
        ClipData clipData;
        String path =
message.get(1).substring(message.get(1).indexOf("/"));
        clipData = ClipData.newPlainText(PATH, path);
        clipboardManager.setPrimaryClip(clipData);
        Toast.makeText(mContext, "AbsPath has been copied to buffer",
Toast.LENGTH_SHORT).show();
    }
})
        .setPositiveButton("OK", null).show();
    }
    public void openPopupMenu(View view, final int position) {
        PopupMenu popupMenu = new PopupMenu(mContext, view, GravityCompat.END);
        popupMenu.getMenuInflater().inflate(R.menu.menu_item, popupMenu.getMenu());
        popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.file_item_info:
                        showInfoAbout(position);
                        return true;
                    case R.id.file_item_rename:
                        showRenameDialog(position);
                        return true;
                    case R.id.file_item_delete:
                        showDeleteDialog(position);
                        return true;
                    case R.id.file_item_move:
                        moveFile(position);
                        return true;
                    case R.id.file_item_copy:
                        copyFile(position);
                        return true;
                }
                return false;
            }
        });
        popupMenu.show();
    }
    private void showRenameDialog(int position) {
        final File file = mItems.get(position).getFile();
        final EditText inputNewName = AndroidUtil.initEditText(mContext,
file.getName());
        new AlertDialog.Builder(mContext)
            .setTitle(mContext.getString(R.string.app_name))
            .setMessage("Do you want to rename file?")
            .setView(inputNewName)
            .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                @Override

```

```

        public void onClick(DialogInterface dialog, int which) {
            String newName = inputNewName.getText().toString();
            if (!TextUtils.isEmpty(newName)) {
                String path = file.getAbsolutePath()
                    .substring(0,
file.getAbsolutePath().lastIndexOf("/"));
                if (file.renameTo(new File(path, newName))) {
                    listFiles(mCurrentDir);
                } else {
                    // TODO: 12/11/18 doesn't want work with SDCard
                    Toast.makeText(mContext, "You can't rename the
file...", Toast.LENGTH_SHORT).show();
                }
            }
        }
    })
    .setNegativeButton("CANCEL", null)
    .show();
}

private void showDeleteDialog(final int position) {
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage("Do you want to delete the file?")
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                File file = mItems.get(position).getFile();
                if (file.delete()) {
                    listFiles(mCurrentDir);
                } else {
                    Toast.makeText(mContext, "You can't delete the file...",
Toast.LENGTH_SHORT).show();
                }
            }
        })
        .setNegativeButton("CANCEL", null)
        .show();
}

private void moveFile(final int position) {
    final File file = mItems.get(position).getFile();
    final EditText inputNewLocation = AndroidUtil.initEditText(mContext, null);
    fillEditBox(inputNewLocation);
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage("Do you want to move file to?")
        .setView(inputNewLocation)
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String nameFile = file.getName();
                if (!TextUtils.isEmpty(nameFile)) {
                    String path = inputNewLocation.getText().toString();
                    if (file.renameTo(new File(path, nameFile))) {
                        mCurrentDir = new File(path);
                        updateTitleName(mCurrentDir.getName());
                        listFiles(mCurrentDir);
                    }
                }
            }
        })
        .setNegativeButton("CANCEL", null)
        .show();
}

```

```

        } else {
            Toast.makeText(mContext, "You can't move the file...",
Toast.LENGTH_SHORT).show();
        }
    }
    })
    .setNegativeButton("CANCEL", null)
    .show();
}
private void copyFile(int position) {
    final File file = mItems.get(position).getFile();
    final EditText inputLocationForCopy = AndroidUtil.initEditText(mContext, null);
    fillEditBox(inputLocationForCopy);
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage("Do you want to copy file to?")
        .setView(inputLocationForCopy)
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String nameTargetLocation =
inputLocationForCopy.getText().toString();
                CopyFileAsyncTask asyncTask = new
CopyFileAsyncTask(nameTargetLocation, file, mContext);
                asyncTask.execute();
            }
        })
        .setNegativeButton("CANCEL", null)
        .show();
}
private void fillEditBox(EditText inputLoc) {
    ClipboardManager clipboardManager =
        (ClipboardManager)
mContext.getSystemService(Context.CLIPBOARD_SERVICE);
    ClipData data = clipboardManager.getPrimaryClip();
    if (data != null) {
        ClipData.Item item = data.getItemAt(0);
        String text = item.getText().toString();
        inputLoc.setText(text);
        Toast.makeText(mContext, text, Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(mContext, "Buffer is empty", Toast.LENGTH_LONG).show();
    }
}
public boolean onBackPressed() {
    if (mHistory.size() > 0) {
        HistoryEntry histEntry = mHistory.remove(mHistory.size() - 1);
        sTitleForUpdate = histEntry.getTitle();
        updateTitleName(sTitleForUpdate);
        if (histEntry.getDirectory() != null) {
            listFiles(histEntry.getDirectory());
        } else {
            listRootFolders();
        }
        mListView.setSelectionFromTop(histEntry.getScrollItem(),

```

```

histEntry.getScrollOffset());
        return false;
    } else {
        return true;
    }
}
public void mkDir() {
    final EditText inputNewDir = AndroidUtil.initEditText(mContext, null);
    new AlertDialog.Builder(mContext)
        .setTitle(mContext.getString(R.string.app_name))
        .setMessage("Do you want to create new folder?")
        .setView(inputNewDir)
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String nameFile = inputNewDir.getText().toString();
                if (!TextUtils.isEmpty(nameFile) && mCurrentDir != null) {
                    if (new File(mCurrentDir.getAbsolutePath(),
inputNewDir.getText().toString()).mkdir()) {
                        Toast.makeText(mContext, "You have created new
folder.", Toast.LENGTH_SHORT).show();
                        listFiles(mCurrentDir);
                    } else {
                        Toast.makeText(mContext, "You can't create folder
here...", Toast.LENGTH_SHORT).show();
                    }
                }
            }
        })
        .setNegativeButton("CANCEL", null)
        .show();
}
}

```

Б.3 Клас com.alburd.sfmapp.ui.adapters.FileListAdapter

```

package com.alburd.sfmapp.ui.adapters;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import com.alburd.sfmapp.ui.adapters.adaptermodels.ListItem;
import com.alburd.sfmapp.ui.fragments.layouts.DetailCellLayout;
import java.util.List;
public class FileListAdapter extends BaseFragmentAdapter {
    private Context mContext;
    private List<ListItem> mItems;
    public FileListAdapter(Context context, List<ListItem> items) {
        mContext = context;
        mItems = items;
    }
    @Override
    public int getCount() {
        return mItems.size();
    }
    @Override
    public Object getItem(int position) {

```

```

        return mItems.get(position);
    }
    @Override
    public long getItemId(int position) {
        return 0;
    }
    public int getViewTypeCount() {
        return 2;
    }
    public int getItemViewType(int pos) {
        return mItems.get(pos).getSubtitle().length() > 0 ? 0 : 1;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = new DetailCellLayout(mContext);
        }
        DetailCellLayout textDetailCell = (DetailCellLayout) convertView;
        ListItem item = mItems.get(position);
        if (item.getIcon() != 0) {
            textDetailCell.setDataItem(item.getTitle(),
                                      item.getSubtitle(), null, item.getIcon());
        } else {
            String type = item.getExtension().toUpperCase().substring(0,
                               Math.min(item.getExtension().length(), 4));
            textDetailCell.setDataItem(item.getTitle(),
                                      item.getSubtitle(), type, 0);
        }
        return convertView;
    }
}

```

Б.4 Клас com.alburd.sfmapp.ui.adapters.adaptermodels.ListItem

```

package com.alburd.sfmapp.ui.adapters.adaptermodels;
import java.io.File;
public class ListItem {
    private int mIcon;
    private String mTitle;
    private String mSubtitle;
    private String mExtension;
    private File mFile;
    public ListItem(int icon, String title, String subtitle, String ext, File
file) {
        this.mIcon = icon;
        this.mTitle = title;
        this.mSubtitle = subtitle;
        this.mExtension = ext;
        this.mFile = file;
    }
    public int getIcon() {
        return mIcon;
    }
    public void setIcon(int icon) {
        this.mIcon = icon;
    }
}

```

```

public String getTitle() {
    return mTitle;
}
public void setTitle(String title) {
    this.mTitle = title;
}
public String getSubtitle() {
    return mSubtitle;
}
public void setSubtitle(String subtitle) {
    this.mSubtitle = subtitle;
}
public String getExtension() {
    return mExtension;
}
public void setExtension(String extension) {
    this.mExtension = extension;
}
public File getFile() {
    return mFile;
}
public void setFile(File file) {
    this.mFile = file;
}
}

```

Б.5 Интерфейс ...ui.fragments.interfaces.DocumentSelectListener

```

package com.alburd.sfmapp.ui.fragments.interfaces;
import android.view.View;
import com.alburd.sfmapp.ui.fragments.FileManagerFragment;
import java.io.File;
import java.util.ArrayList;
public interface DocumentSelectListener {
    void didSelectFiles(FileManagerFragment activity, ArrayList<String> files);
    void updateAppBarName(String name);
    void startDocumentSelectActivity(View view, int position);
    void makeDir();
}

```

Б.6 Класс com.alburd.sfmapp.data.asyncntask.CopyFileAsyncTask

```

package com.alburd.sfmapp.data.asyncntask;
import android.annotation.SuppressLint;
import android.content.Context;
import android.os.AsyncTask;
import android.util.Log;
import android.widget.Toast;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
public class CopyFileAsyncTask extends AsyncTask<Void, Void, Boolean> {

```

```

public static final String TAG = "LogCopyFileAsyncTask";
private String mCopyTo;
private File mFileForCopy;
@SuppressWarnings("StaticFieldLeak")
private Context mContext;
public CopyFileAsyncTask(String copyTo, File fileForCopy, Context context)
{
    this.mCopyTo = copyTo;
    this.mFileForCopy = fileForCopy;
    this.mContext = context;
}
@Override
protected Boolean doInBackground(Void... files) {
    try {
        File fileTargetLocation = new File(mCopyTo,
mFileForCopy.getName());
        Log.d(TAG, "doInBackground: " + mCopyTo + " " +
mFileForCopy.getName());
        InputStream in = new FileInputStream(mFileForCopy);
        OutputStream out = new FileOutputStream(fileTargetLocation);
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        in.close();
        out.close();
        return true;
    } catch (IOException e) {
        Log.d(TAG, "doInBackground: " + e.getMessage());
    }
    return false;
}
@Override
protected void onPostExecute(Boolean aVoid) {
    if (aVoid) {
        Toast.makeText(mContext, "You have copied the file to " + mCopyTo,
Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(mContext, "You can't copy the file",
Toast.LENGTH_SHORT).show();
    }
}
}

```

ДОДАТОК В
Копії слайдів презентації

ДИПЛОМНА РОБОТА

РОЗРОБКА ПРОГРАМИ ФАЙЛОВОГО МЕНЕДЖЕРА ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

Розробила: ст. групи КНТ-227
Керівник: доцент, к.т.н.

Бурдакова О.С.
Сердюк С.М.

Рисунок В.1 – Слайд №1

Мета роботи

розробка програмного забезпечення утиліти «Файловий менеджер» для основних файлових систем операційної системи (ОС) Android.

Об'єкт дослідження – процеси, що забезпечують роботу з файловою системою ОС Android.

Предмет дослідження – методи розробки файлових менеджерів в ОС Android



НУ "ЗП", Кафедра програмних засобів

Рисунок В.2 – Слайд №2

Задачі, що вирішуються

- огляд особливостей ОС Android;
- огляд основних файлових систем, що використовуються в ОС Android;
- огляд програм-аналогів;
- розробка технічного завдання;
- вибір і обґрунтування мови програмування та середовища розробки;
- розробка загального алгоритму роботи програми;
- програмна реалізація застосунку;
- розробка необхідної документації.

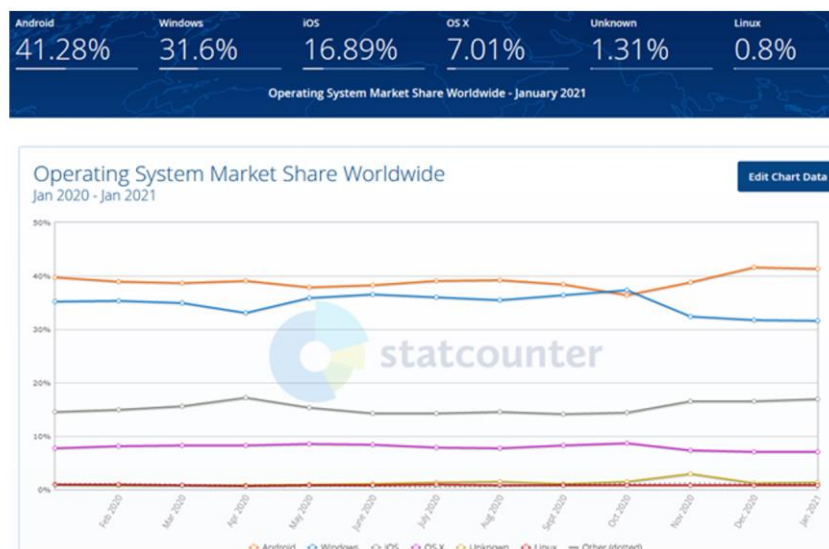


НУ "ЗП", Кафедра програмних засобів

3

Рисунок В.3 – Слайд №3

Ринок операційних систем

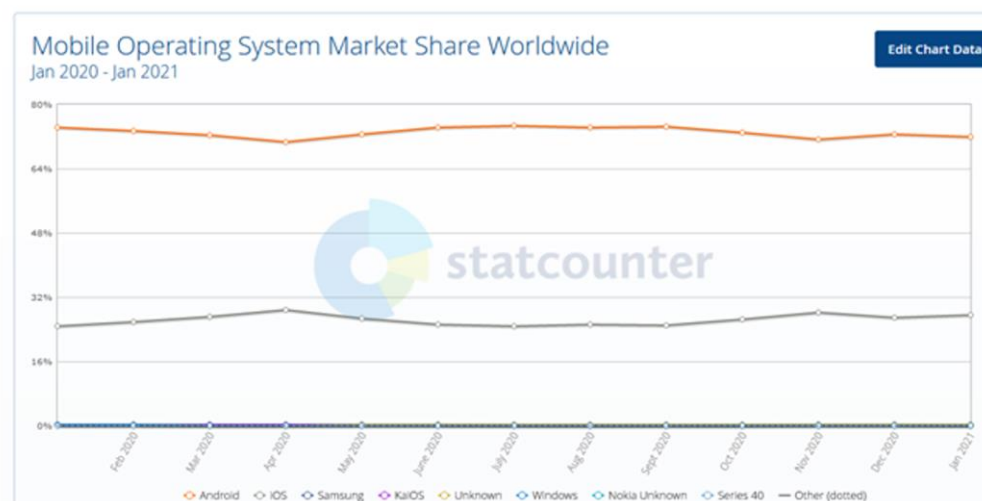


НУ "ЗП", Кафедра програмних засобів

4

Рисунок В.4 – Слайд №4

Використання операційних систем у смартфонах



НУ "ЗП", Кафедра програмних засобів

5

Рисунок В.5 – Слайд №5

Файлові системи, що використовуються в ОС Android

Найменування	Призначення
VFAT	файловий ввід/вивід у захищеному режимі
YAFFS	для флеш-пам'яті NAND-типу
Ext2-Ext4	основні файлові системи ОС Android
F2FS	для роботи з флеш-пам'яттю (SD/MMC) і SSD-накопичувачами
Samsung RFS	для пристроїв на базі ОС Linux
UBIFS	для роботи з пам'яттю типу NAND
SDCardFS	модуль для Unix-подібної ОС, що дозволяє користувачам без спеціальних прав створювати власні файлові системи



НУ "ЗП", Кафедра програмних засобів

6

Рисунок В.6 – Слайд №6

Огляд програм-аналогів

Назва ФМ	Розробник	Оновлення	Версія Android	Підтримка української мови	Ціна, \$
Total Commander	C. Ghisler	2020-11-02	від 1.5	—	—
Root Explorer	Speed Software	2020-10-28	від 2.3	—	3.99
ES File Explorer	ES Global	2021-01-30	від 4.0	—	—
X-plore	Lonely Cat Games	2020-12-21	від 4.1	—	—

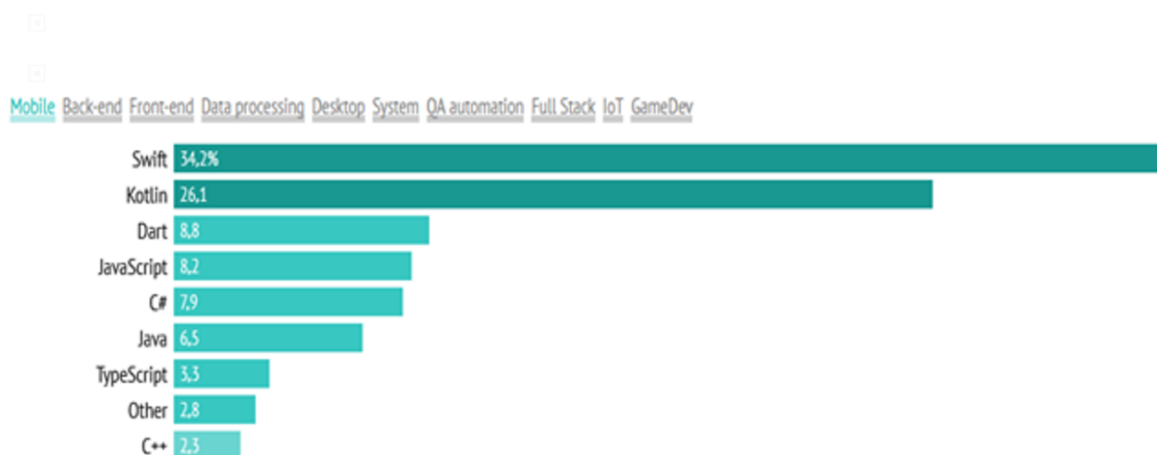


НУ "ЗП", Кафедра програмних засобів

7

Рисунок В.7 – Слайд №7

Використання мов програмування для мобільної розробки

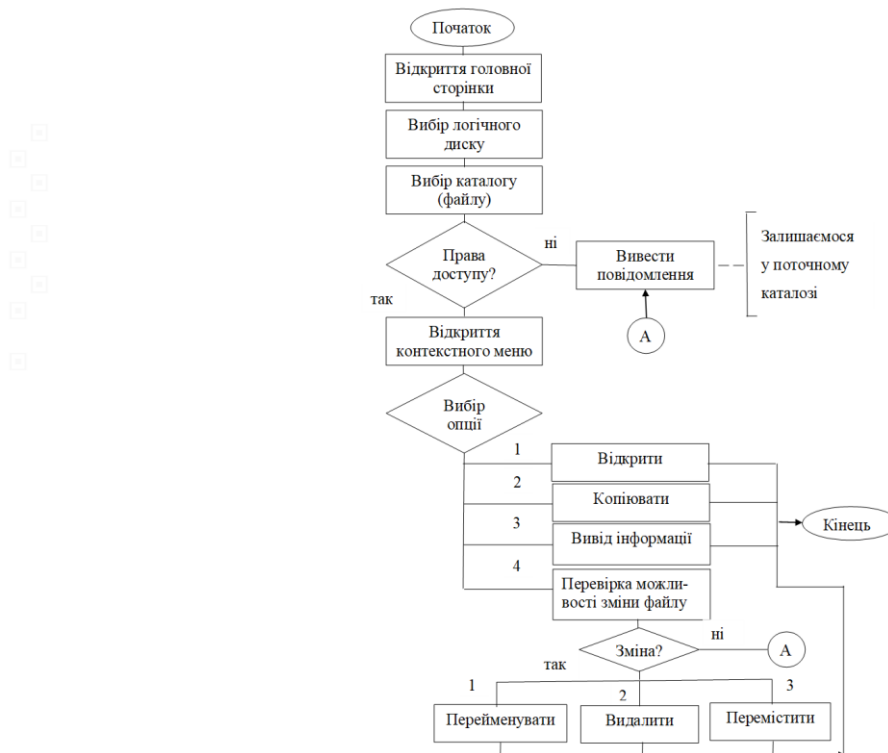


НУ "ЗП", Кафедра програмних засобів

8

Рисунок В.8 – Слайд №8

Блок-схема алгоритму програми



НУ "ЗП", Кафедра програмних засобів

Рисунок В.9 – Слайд №9

9

Головна активність застосунку

```

public class MainActivity extends AppCompatActivity implements DocumentSelectListener {
    private ActionBar mActionBar;
    private FragmentManager mFragmentManager;
    private FragmentTransaction mFragmentTransaction;
    private FileManagerFragment mDirectoryFragment;
    private boolean mIsSelectionMode;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mActionBar = getSupportActionBar();
        if (mActionBar != null) {
            mActionBar.setTitle("Simple file manager");
        }
        mFragmentManager = getSupportFragmentManager();
        mFragmentTransaction = mFragmentManager.beginTransaction();
        mDirectoryFragment = new FileManagerFragment();
        mFragmentTransaction.replace(R.id.fragment_container, mDirectoryFragment, mDirectoryFragment.toString());
        mFragmentTransaction.commit();
    }
}

```



НУ "ЗП", Кафедра програмних засобів

Рисунок В.10 – Слайд №10

10

Основні класи застосунку

Назва класу	Опис
SFMAApp.java	Клас, що успадковується від Application, прописується у маніфесті, як клас "точки входу" до застосунку. Реалізує синглтон класу SFMAApp та геттер для нього
MainActivity.java	Клас головної активності. Вона має властиві усім активностям універсальні методи. Також реалізує інтерфейс DocumentSelectListener для зв'язку з фрагментом FileManagerFragment
FileListAdapter.java	Клас-адаптер для ListView, який складається з масиву елементів ListItem та успадковується від BaseFragmentAdapter
ListItem.java	Клас-модель, поля якого описують файл або папку
DocumentSelectListener.java	Інтерфейс який містить методи для роботи з елементами списку ListView, логіка якого реалізується в FileManagerFragment
FileManagerFragment.java	Клас, де знаходиться основна логіка застосунку. Містить як загальні методи для фрагментів, так і додаткові

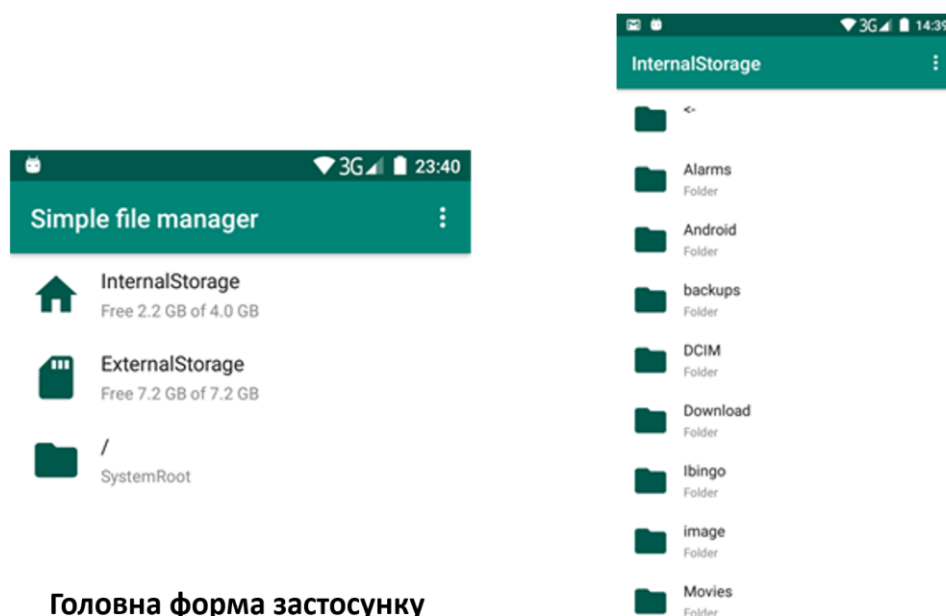


НУ "ЗП", Кафедра програмних засобів

11

Рисунок В.11 – Слайд №11

Приклади роботи з застосунком



Головна форма застосунку

Приклад відкриття обраного каталогу

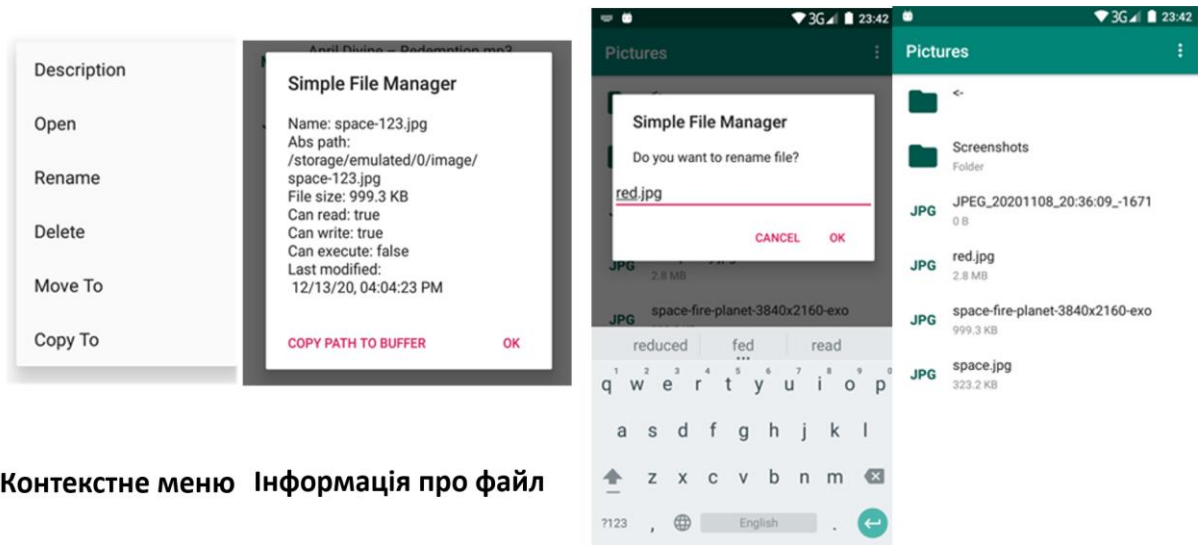


НУ "ЗП", Кафедра програмних засобів

12

Рисунок В.12 – Слайд №12

Приклади роботи з застосунком



Контекстне меню Інформація про файл

Перейменування файлу



НУ "ЗП", Кафедра програмних засобів

13

Рисунок В.13 – Слайд №13

Висновки

- проведено огляд особливостей ОС Android;
- проведено огляд файлових систем, що використовуються в ОС Android;
- проведено огляд програм-аналогів;
- розроблено технічне завдання;
- обрані засоби розробки;
- розроблено загальний алгоритм роботи програми;
- розроблено програмне забезпечення;
- розроблено необхідну документацію.



НУ "ЗП", Кафедра програмних засобів

14

Рисунок В.14 – Слайд №14

**Доклад закінчено,
дякую за увагу!**



НУ "ЗП", Кафедра програмних засобів

Рисунок В.15 – Слайд №15

15