Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка застосунку Android Reader з можливістю читати книги різних форматів
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-2
Микитчук В.М.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Кіношенко Д.К.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Микитчуку Владиславу Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку Android Reader з можливістю читати книги різних форматів

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 травня 2024 р

3. Вихідні дані до роботи статті в інтернеті стосовно розробки під Android, офіційна документація Kotlin, офіційна документація Android API

4. Перелік питань, що потрібно опрацювати в роботі

1. Здатність застосунку пошуку файлів підтримуваних форматів на пристрої користувача.
2. Можливість працювати з PDF-форматом.
3. Можливість працювати із TXT форматом.
4. Можливість візуального розрізнення знайдених книг.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми читання книг різних форматів, постановка задачі, тестові книги.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-13.04.24	
3	Аналіз літератури з досліджуваної проблеми	14.04.24-22.04.24	
4	Аналіз бібліотек для розробки	23.04.24-27.04.24	
5	Створення візуальної складової застосунку	28.04.24-04.05.24	
6	Програмна реалізація	05.05.24-15.05.24	
7	Оформлення пояснювальної записки	16.05.24-18.05.24	
8	Перевірка на плагіат	27.05.24	
9	Рецензування	28.05.24	
10	Підготовка презентації та доповіді	29.05.24-04.06.24	
11	Занесення роботи в електронний архів	06.06.24	
12	Попередній захист кваліфікаційної роботи	06.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Кіношенко Д.К.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 63 с., 40 рис., 30 джерел.

ANDROID, ANDROID STUDIO, KOTLIN, PDF, TXT, РІДЕР, ПАТЕРНИ, АРХІТЕКТУРА, БІНАРНИЙ ПОШУК, ЕЛЕКТРОННА КНИГА, ФАЙЛ, МЕТАДАНІ.

Об’єктом роботи є відображення електронних книг різних форматів.

Метою роботи є розробка android-застосунку для читання електронних книг та файлів різних форматів.

Використано засоби сучасної розробки android-застосунків, а також бібліотеки для взаємодії з різними форматами електронних книг. Проведено аналіз застосування метаданих файлів різних форматів, а також взаємодії з ними. Обгорнуто в сучасний застосунок для пошуку файлів різних розширень та відкриття файлів в самому застосунку. Розроблено архітектуру для швидкого розширення функціоналу застосунку.

У результаті роботи здійснена програмна реалізація застосунку для читання книг різних форматів.

ANDROID, ANDROID STUDIO, KOTLIN, PDF, TXT, READER, PATTERNS, ARCHITECTURE, BINARY SEARCH, E-BOOK, FILE, METADATA.

The object of work is to display e-books of different formats.

The purpose of the work is to develop an android application for reading e-books and files of various formats.

The tools of modern android application development, as well as libraries for interaction with different e-book formats, are used. An analysis of the use of metadata of files of different formats, as well as interaction with them, is carried out. Wrapped in a modern application for searching for files of various extensions and opening files in the application itself. An architecture for rapid expansion of the application's functionality has been developed.

As a result of the work, the software implementation of the application for reading books of various formats was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

PDF – Portable Document Format, найвідоміший та найрозповсюдженіший формат документів

TXT – Text File Format, найвідоміший формат для зберігання необроблених даних

DOC – розширення файлу, що використовується для зберігання файлів, оброблюваних засобами Microsoft Word

MOBI – розширення файлу, створене для електронних книг, яке в основному використовуються пристроями Amazon

FB2 – розширення файлу, створене для сучасних електронних книг з додатковими можливостями зберігання форматів книги

EPUB – розширення файлу, що містить електронну книгу з певними особливостями, створена здебільшого для платформи Android

ПЗ – програмне забезпечення

ВСТУП

Читалки електронних книг, безсумнівно, є одними з найважливіших інструментів у цифровому світі, де доступ до літератури може здійснюватися швидко і зручно.

Завдяки розмаїттю форматів електронних книг, від PDF до EPUB, потреба в універсальних читалках стає доречною. Аналогічно до комп’ютерної графіки, яка розділяється на візуалізацію, обробку зображень і розпізнавання образів, читалки вирізняються за їхньою спроможністю працювати з різними форматами та функціональністю. Візуалізація в цьому контексті означає представлення текстової інформації у зручному для читання вигляді, тоді як розпізнавання форматів та підтримка різних стандартів є ключовими завданнями для забезпечення універсальності читалки.

Обробка зображень у цьому контексті може означати відображення обкладинок книг, показ вмісту сторінок або навіть можливість адаптувати шрифти та налаштовувати кольорову схему для комфортного читання.

Актуальність роботи цього питання полягає в пошуку оптимальних рішень для забезпечення зручності, універсальності та ефективності використання читалок електронних книг в сучасному світі.

1 ОГЛЯД ОСНОВНИХ ТЕХНОЛОГІЙ ШИФРУВАННЯ ТЕКСТУ ТА ЇХ ОСОБЛИВОСТЕЙ

1.1 Особливості PDF-формату

PDF — це формат, який використовується для представлення документів у вигляді електронного документа. Акронім «PDF» розшифровується як «Portable Document Format», що в перекладі з англійської означає «портативний формат документа». Основною метою PDF є збереження і передача документів, щоб вони виглядали однаково на будь-якому пристрої та зберігали свою структуру та форматування незмінними.

Окрім того, PDF-файли мають кілька ключових особливостей, що роблять їх важливим інструментом для обміну та збереження інформації. Однією з головних переваг PDF є їхня незалежність від платформи. Вони можуть бути відкриті та переглянуті на будь-якому пристрої, незалежно від операційної системи чи програмного забезпечення. Це забезпечує універсальність та зручність у використанні.

Ще однією важливою характеристикою PDF-файлів є збереження форматування. Під час створення PDF всі шрифти, розміри тексту та графічні елементи зберігаються без змін, що гарантує однаковий вигляд документа для всіх користувачів. Крім того, PDF-файли забезпечують високий рівень безпеки даних, оскільки можуть бути захищені паролем або шифруванням. Це робить їх ідеальними для передачі конфіденційної інформації.

Також PDF-файли підтримують вставку різноманітних мультимедійних елементів, таких як зображення, відео та аудіо. Це робить їх універсальним засобом для створення різних типів контенту. Більше того, PDF-файли можуть містити інтерактивні елементи, такі як гіперпосилання, форми для заповнення та анімації, що дозволяє користувачам взаємодіяти з документом.

Нарешті, PDF-файли можуть бути стиснуті для зменшення їх розміру. Це особливо корисно для передачі файлів через Інтернет чи зберігання на

обмеженому просторі. Усі ці особливості роблять PDF-файли незамінним інструментом для сучасного ділового та особистого використання.

Розуміння того, як працюють PDF-файли, допомагає ефективно використовувати їх для створення, редагування та обміну документами у різних сферах життя та роботи.

Метадані у PDF-файлах відіграють важливу роль у забезпеченні інформаційної структури та організації документів. Вони включають загальну інформацію про документ, таку як його назву, автора, опис та ключові слова. Ця інформація допомагає користувачам швидше знаходити та ідентифікувати документи.

Крім того, метадані містять інформацію про дату та час створення та останньої модифікації документа. Це дозволяє встановлювати часові межі для документів та відслідковувати їхню історію змін. Метадані також можуть включати інформацію про автора документа, включаючи ім'я, адресу електронної пошти та інші контактні дані, а також дані про організацію, до якої належить автор.

Додатково, метадані можуть містити інформацію про розмір документа та кількість його сторінок. Це допомагає користувачам оцінити обсяг та складність документа перед його відкриттям. Вони також можуть включати права доступу до документа, такі як рівень доступу для читання, редагування, друку та копіювання, а також вказівки про захист документа паролем чи шифруванням.

Нарешті, метадані можуть містити інформацію про використані шрифти та колірні простори у документі. Це важливо для забезпечення правильного відображення тексту та графіки на різних пристроях та програмах перегляду PDF. Усі ці аспекти роблять метадані невід'ємною частиною PDF-документів, забезпечуючи їхню організацію та зручність використання.

Метадані у PDF-файлах допомагають управляти та організовувати інформацію, забезпечуючи її ефективну обробку, пошук та використання.

1.2 Альтернативи PDF-формату та їх особливості

Спочатку здається, що з такими численними перевагами PDF-формату не варто використовувати інші формати документів. Однак, щоб зрозуміти сучасне становище, корисно згадати історію розвитку електронних документів.

У кінці 20-го століття, коли комп'ютеризація почала широко розвиватися, з'явилося багато різних форматів документів. Наприклад, DOC, XLS, і PPT від Microsoft, а також формати OpenDocument, HTML, і TXT. Кожен з цих форматів мав свої переваги і обмеження, що зумовило потребу в універсальному форматі, який би забезпечував однакове відображення документів на будь-якому пристрої.

У 1993 році Adobe Systems представила PDF-формат, який став відповіддю на цю потребу. PDF швидко став популярним через свою здатність зберігати форматування, графіку та інші елементи документа без змін, незалежно від платформи чи програмного забезпечення. Це робить PDF ідеальним для обміну документами та забезпечення їх однакового вигляду для всіх користувачів.

Проте, не всі формати стали застарілими чи непотрібними. Наприклад, формати DOCX та XLSX використовуються для редагування документів, оскільки вони надають більше можливостей для роботи з текстом та даними. HTML залишається важливим для веб-розробки, а TXT зручним для простих текстових даних.

Отже, хоча PDF-формат домінує у сфері електронних документів, інші формати все ще мають свої унікальні використання та переваги.

1.2.1 Формати для відтворення художньої літератури

В сучасному світі за наявності неймовірної кількості художньої літератури, від поем та віршів і до детективних романів, більшість сучасних книжок перейшла в електронний формат. Відповідно до цього безліч книг потребувала підтримки відображення, що для книг довжиною за тисячу сторінок стало проблемою. Якщо такі книги ми триматимемо у вигляді PDF, то читач буде вимушений читати усе в одному незмінюваному вигляді, без можливості зміни шрифту та його розміру, залишання закладок для продовження читання без додаткових дій тощо. У зв’язку з цим ми отримали ряд нових форматів та ПО для роботи з ними.

MOBI (Mobipocket) - це формат електронних книг, який спеціально розроблений для використання на пристроях компанії Amazon, таких як Kindle. Основні особливості цього формату включають підтримку тексту, зображень та відформатованого контенту. Файли MOBI можуть також містити інтерактивні елементи, такі як гіперпосилання. Частіше за все, вони мають розширення «.mobi» або «.azw» і сумісні з більшістю пристроїв Kindle, а також деякими програмами читання.

FB2 (FictionBook) - це відкритий формат для електронних книг, який часто використовується для літературних творів, зокрема фантастики та прози. Він відзначається високим рівнем структуризації, що дозволяє включати розділи, підзаголовки, вірші та інші елементи. Формат FB2 має просту структуру, що полегшує створення та редагування книг. Файли FB2 мають розширення «.fb2» і зазвичай сумісні з більшістю програм читання електронних книг.

EPUB (Electronic Publication) - це стандартний формат для електронних книг, який підтримується більшістю електронних книг-читалок, за винятком пристроїв Kindle. Основні особливості EPUB включають динамічне змінення розміру тексту, оптимізацію для різних пристроїв та підтримку вбудованих зображень, гіперпосилань та інших мультимедійних елементів. Файли EPUB

мають розширення .EPUB і є одними з найбільш універсальних форматів для електронних книг, сумісних з більшістю програм читання та пристроїв.

1.2.2 Сучасне ПЗ для відображення документів

Сучасне програмне забезпечення (ПЗ) для відображення документів є динамічним і різноманітним, пропонуючи рішення для різних потреб і платформ. Найпопулярніші програми включають у себе такі можливості, як перегляд, редагування, анотування та спільна робота над документами, що робить їх незамінними інструментами для бізнесу, освіти, наукових досліджень та особистого використання.

Одні з найвідоміших програм для відображення документів включають у себе Adobe Acrobat Reader, Microsoft Word, Google Docs, LibreOffice, Foxit Reader та Sumatra PDF. Adobe Acrobat Reader є стандартом для перегляду та друку файлів у форматі PDF та має розширені функції для редагування та коментування. Microsoft Word входить до складу пакету Microsoft Office та має широкий набір інструментів для створення та редагування текстових документів. Google Docs є веб-прикладанням, що дозволяє створювати та редагувати документи у реальному часі та спільно працювати з ними.

Нові технології, такі як розпізнавання тексту за допомогою штучного інтелекту, інтеграція з хмарними сервісами для зручного зберігання та обміну документами, а також підтримка мультимедійних елементів, дозволяють сучасним програмам для відображення документів працювати більш ефективно та забезпечувати різноманітні можливості для користувачів. Такі програми не лише спрощують роботу з документами, а й роблять її більш продуктивною та зручною для кожного користувача.

Окрім цього існує ряд платних та безкоштовних програм для читання на Android та IOS пристроях, в тому числі для електронних книг-планшетів. Подібні пристрої були б непотрібними без гарного ПЗ, з яким вони будуть

відображувати в красивому та зручному форматі усі книжки. До безкоштовних програм-читалок можна віднести одні з найпопулярніших:

ReadEra – це безкоштовний і сучасний застосунок для Android, спеціально розроблений для відображення електронних книг та інших видів документів в одному місці. Завдяки цьому застосунку, користувачі можуть зручно читати будь-які формати документів, залишати закладки, змінювати шрифт і розмір тексту, а також фон сторінок. Всі ці функції доступні безкоштовно, але з преміум версією з'являється додаткова можливість зберігання книжок в хмарі. Це дозволяє користувачам мати постійний доступ до своєї бібліотеки навіть при зміні пристрою, що робить ReadEra надзвичайно зручним інструментом для читання.

Librera – це безкоштовний open-source проєкт, який пропонує всі необхідні функції для читання книг. Незважаючи на те, що дизайн програми виглядає більш застарілим порівняно з іншими сучасними читалками, Librera чудово виконує свою основну функцію – швидке та зручне переглядання документів у дорозі. Відкритий вихідний код дозволяє іншим розробникам вивчати технології, використані у Librera, і створювати на їх основі власні, більш сучасні читалки. Таким чином, Librera не лише корисна для кінцевих користувачів, але й слугує навчальним ресурсом для програмістів.

Moon+ Reader – це безкоштовний застосунок для читання книг, який підходить як для читачів, так і для дизайнерів книг. Цей застосунок демонструє, як можна оформити книгу на високому дизайнерському рівні, роблячи її привабливішою для читачів. Окрім стандартних можливостей, таких як зміна шрифтів та кольорів тексту, Moon+ Reader надає користувачам спеціальну лінійку для оптимального налаштування розмірів тексту та інших елементів книги. Ці додаткові можливості можуть значно підвищити естетичну цінність книги, що особливо важливо для людей, які цінують візуальну привабливість написаного.

Що стосується IOS, тут також є кілька популярних застосунків для читання книг.

FB Reader – одна з найпопулярніших читалок електронних книг для платформ Android та iOS. Ця програма вирізняється своєю універсальністю, адже вона працює як на смартфонах, так і на планшетах, підтримуючи різні версії операційних систем. Основною перевагою FB Reader є сучасний та зручний дизайн, який забезпечує комфортне використання. Хоча ця читалка може не мати багатьох вишуканих функцій, вона надійно виконує основні завдання, такі як читання електронних книг у різних форматах, відображення таблиць і підтримка різних шрифтів. Користувачі можуть налаштовувати режим читання відповідно до своїх уподобань, що робить FB Reader зручним інструментом для щоденного використання.

Kindle – це один з найвідоміших читальних застосунків, створений компанією Amazon. Цей застосунок відомий своєю високою оптимізацією та ефективністю у роботі. Kindle підтримує багато форматів електронних книг, включаючи власний Amazon Kindle Format (AZW), TXT, PDF та MOBI. Однак одним з недоліків Kindle є відсутність підтримки формату EPUB, що може бути незручним для користувачів, які мають книги у цьому форматі. Незважаючи на це, Kindle відзначається своєю простотою у використанні, ефективністю та доступністю великого асортименту книг у магазині Amazon, що робить його незамінним інструментом для багатьох читачів електронних книг.

Pocketbook – це ще один універсальний інструмент для читання електронних книг, який доступний як на платформі Android, так і на IOS. Ця програма дозволяє користувачам зручно читати книги на своїх пристроях і забезпечує можливість синхронізації книг між різними пристроями. Це робить Pocketbook ідеальним варіантом для тих, хто часто переміщується між різними гаджетами. Дизайн програми відповідає сучасним стандартам, що забезпечує комфортний інтерфейс для читання без зайвих відволікань. Pocketbook рекомендується користувачам, які шукають зручний та універсальний спосіб читання електронних книг на своїх мобільних

пристроях, адже він пропонує багатий функціонал та високу зручність у використанні.

Існує безліч інших схожих застосунків, і, як і майже з усім, з часом їх ставатиме все більше.

1.3 Інтеграція різних форматів електронних книг з Android платформою

Взаємодія платформи Android з файлами є ключовим аспектом розробки будь-якого застосунку, в тому числі і читальної програми. Android надає розробникам широкі можливості для роботи з файловою системою, що дозволяє ефективно керувати різними типами файлів, включаючи електронні книги у різних форматах.

Перш за все, важливо зрозуміти, що Android підтримує різноманітні системи файлових систем, такі як FAT, ext3, ext4, а також власну віртуальну файлову систему, яка дозволяє доступатися до різних директорій та файлів на пристрої. Це створює можливості для розробників створювати застосунки, які можуть легко читати та записувати дані з різних джерел, включаючи внутрішню пам'ять пристрою, SD-карти, а також зовнішні сховища хмарних сервісів.

Крім того, Android надає API для роботи з файлами, яке дозволяє розробникам здійснювати різноманітні операції з файлами, такі як читання, запис, копіювання, переміщення та видалення. Це API дозволяє ефективно взаємодіяти з файлами безпосередньо з програмного коду застосунку, що є важливим для розробки читальної програми, оскільки користувачі очікують зручного інтерфейсу для роботи з електронними книгами.

Крім того, Android пропонує різні інструменти для роботи з файлами, такі як Content Providers, які дозволяють застосункам спільно використовувати дані, а також Document Providers, які дозволяють

розробникам доступатися до різних джерел файлів, включаючи хмарні сховища.

Загальна гнучкість та можливості, які надає платформа Android для роботи з файлами, роблять її ідеальною платформою для розробки читальної програми, яка може ефективно працювати з електронними книгами у різних форматах. Враховуючи ці особливості, розробники можуть створювати застосунки, які забезпечують зручний та надійний досвід читання для користувачів Android пристроїв.

1.4 Постановка задачі

Таким чином, відтворення візуально електронних книг та інших видів документів різного формату є актуальною проблемою. Тому ставиться завдання створити один з таких, що зможе працювати одночасно з багатьма різними форматами.

Об’єктом роботи є файли форматів PDF, TXT, XML.

Метою роботи є розробка застосунку, що базується на використанні сучасних технологій та бібліотек, які дозволяють читати різні формати книг а також форматувати текст задля збільшення читабельності під різні потреби користувачів.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих бібліотек та технологій, на достатньому рівні щоб мати змогу з ними працювати;

– розробити алгоритм пошуку файлів на пристрої, щоб при натисканні на них ми могли одразу відкрити їх;

– реалізувати змогу працювати з різними форматами.

2 АЛГОРИТМИ ВЗАЄМОДІЇ ЗАСТОСУНКУ В СЕРЕДОВИЩІ ANDROID

2.1 Пошук усіх оброблюваних файлів на пристрої

Оскільки ми прагнемо створити застосунок, який повноцінно замінюватиме програму-читалку, нам необхідно забезпечити користувачам можливість переглядати файли в зручному вигляді безпосередньо у самому застосунку. Для цього потрібно знайти відповідні файли на пристрої та відобразити їх у доступній і зрозумілій формі. Процес пошуку і відображення файлів потребує ретельної підготовки та врахування особливостей платформи.

Першим кроком є отримання доступу до файлів на пристрої. Це завдання ускладнюється залежно від рівня Android API, який використовується на конкретному пристрої. У новіших версіях Android система безпеки вимагає, щоб застосунок запитував дозволи на доступ до певних ресурсів, таких як зовнішня пам’ять. Без належного дозволу застосунок не зможе «читати» або використовувати ці файли. Тому критично важливо правильно запитати доступ, щоб уникнути помилок і забезпечити коректну роботу застосунку.

Особливу увагу слід звернути на різні версії Android API. Наприклад, для пристроїв із версією Android 9 (API level 28) необхідно запросити дозвіл на читання зовнішньої пам’яті (READ_EXTERNAL_STORAGE). Це дозволить застосунку отримати доступ до файлів, збережених на зовнішній пам’яті пристрою.

Важливо не запитувати цей дозвіл на версіях Android, нижчих за API level 28, оскільки це може призвести до запиту невідомого виду доступу, що не тільки спричинить помилку, але й може негативно вплинути на роботу застосунку. Для правильного запиту доступу необхідно враховувати версію

системи і дотримуватися відповідних рекомендацій. Зазвичай цей процес виглядає наступним чином (рис 2.1).

Таким чином, для створення повноцінного застосунку-читалки, потрібно не лише забезпечити можливість перегляду файлів, але й правильно організувати процес отримання доступу до цих файлів, враховуючи всі нюанси роботи з різними версіями Android. Це дозволить зробити використання застосунку максимально комфортним і безпечним для користувачів.

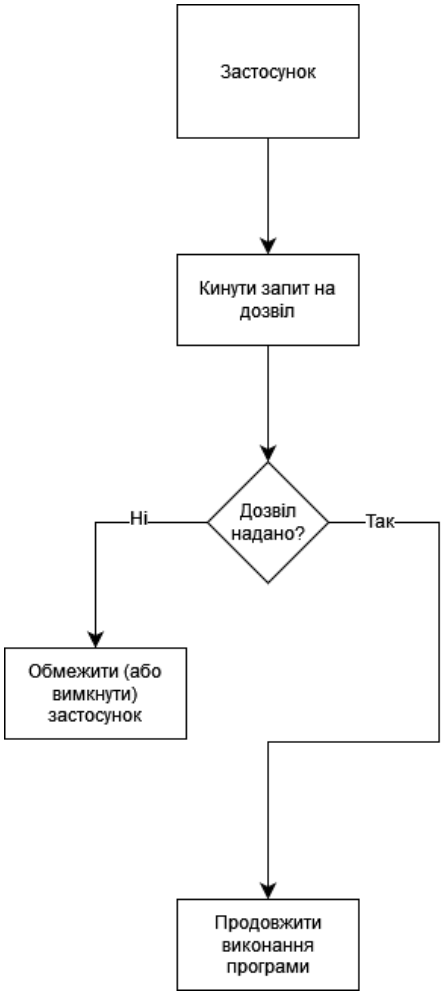


Рисунок 2.1 – Схема логіки посилання запитів на доступ

На даній схемі (рис. 2.1) видно, що в разі ненадання доступу написано, що застосунок в разі відсутності доступу до необхідних привілеїв, нам необхідно його вимкнути або обмежити. В нашому випадку, під час розробки

застосунку для читання книг, потрібно звернути увагу, що, в разі відсутності доступу до файлів, наш проєкт фактично не здатен нічого зробити. Тому застосунок бажано відключати, щоб не породжувати ніяких небажаних ситуацій.

Тепер, коли доступ надано, нам потрібно певним чином знаходити файли в каталозі папок пристрою. Щоб цього добитись, в особливості для нижчих версій Android (на вищих версіях були розроблені значно продуманіші методи), нам необхідно написати певний алгоритм, який буде це робити для нас. Виглядає такий алгоритм наступним чином (рис. 2.2).

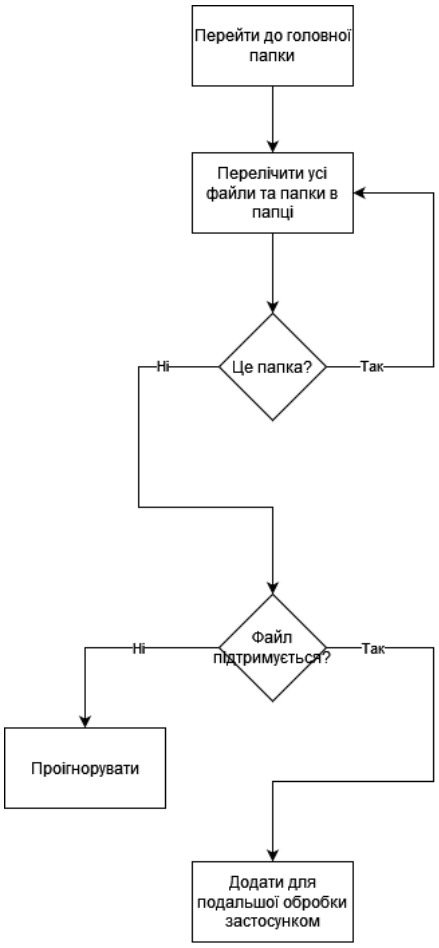


Рисунок 2.2 – Логіка пошуку файлів, які потрібно відобразити в застосунку

Таким чином усі файли будуть знайдені та оброблені таким чином, щоб програмно з ними можна було взаємодіяти (тобто згенерувати усі візуальні елементи для файлів, а також перевірити, чи валідні вони для читання).

2.2 Схематична структура проєкту

Для забезпечення ефективного функціонування нашого проєкту, необхідно ретельно продумати його архітектуру. Створення чіткої архітектури дозволить нам забезпечити стабільну роботу застосунку, полегшить майбутнє масштабування та обслуговування, а також сприятиме зручності роботи команди розробників. Наш застосунок має кілька ключових аспектів, які можуть або відрізнятися, або бути спільними для багатьох частин функціоналу. Враховуючи ці особливості, ми розділимо функціональні складові на основні частини, що дозволить нам більш ефективно організувати процес розробки та тестування.

Отже, розподілимо наш застосунок на такі основні функціональні блоки.

Перший блок – обробка доступу застосунку. Цей компонент відповідає за всі аспекти аутентифікації та авторизації користувачів. Він включає механізми реєстрації нових користувачів, вхід до системи, управління сесіями та перевірку прав доступу до різних частин застосунку. Надійність та безпека цього блоку визначають захищеність даних користувачів та стабільність роботи всього застосунку.

Другий блок – початкова обробка файлів. Цей компонент займається завантаженням, збереженням та початковою обробкою файлів, які завантажують користувачі. Тут реалізовані механізми завантаження файлів на сервер, їх попередньої обробки, наприклад, перевірка на віруси чи конвертація в потрібний формат, а також збереження у базі даних чи

файловій системі. Це дозволяє забезпечити стабільність і швидкодію під час роботи з великою кількістю файлів.

Третій блок – візуальні елементи. Цей компонент відповідає за відображення інформації користувачам та взаємодію з ними. До цього блоку належать усі частини інтерфейсу користувача (UI), включаючи форми, кнопки, графіки, таблиці та інші візуальні компоненти. Ретельно продуманий дизайн та зручний інтерфейс значно покращують користувацький досвід, що позитивно впливає на загальне враження від застосунку.

Кожен з цих блоків має свої специфічні задачі та вимоги, тому їх реалізація потребує окремої уваги та детального планування. Зосереджуючись на цих основних частинах, ми зможемо створити добре структурований, ефективний і зручний у використанні застосунок, який задовольнить потреби наших користувачів.

2.2.1 Обробка доступу застосунку

Наш проєкт планується як такий, що буде підтримуватись в тому числі на старих телефонах. Це плани на майбутнє, але оскільки це в будь-якому разі покращить структуризацію, то бажано про це подумати заздалегідь. Відповідно до цього нам потрібен певний механізм, що надаватиме певні види доступу застосунку, посилаючи запити користувачу. Оскільки в середовищі Android при базовій розробці кожному окремому елементу (певній «сцені») цей доступ надається специфічним для кожної сцени шляхом, нам бажано це перевизначити однаковим шляхом для усіх місць, де доступ може бути необхідним. Для цього ми створимо певний інтерфейс, який міститиме в собі доволі просту структуру (рис. 2.3).

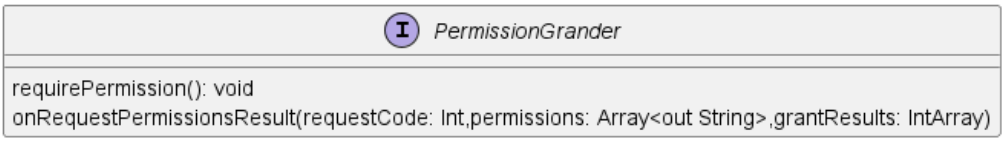


Рисунок 2.3 – Головний інтерфейс обробки доступу в застосунку

Даний інтерфейс здатен при правильному перевизначенні надавати доступ зручним шляхом не лише до файлів, але й до будь-якого іншого виду доступу. Це буде корисно, наприклад, якщо ми при розвитку застосунку захочемо також отримувати доступ до галереї/гугл сервісів тощо. Тепер поговоримо саме про доступ до файлів. Як можна було здогадатись, в наш механізм потрібні нові частини, які будуть взаємодіяти з вже згаданим раніше (рис. 2.3) інтерфейсом. Він буде слугувати конкретно для надання доступу до файлів на пристрої, і називатиметься «ExternalStorageManagementPermissionGrander», що також буде інтерфейсом зі специфічними реалізаціями під конкретні версії Android API. Виглядатиме в загальній картині це наступним чином (рис. 2.4).

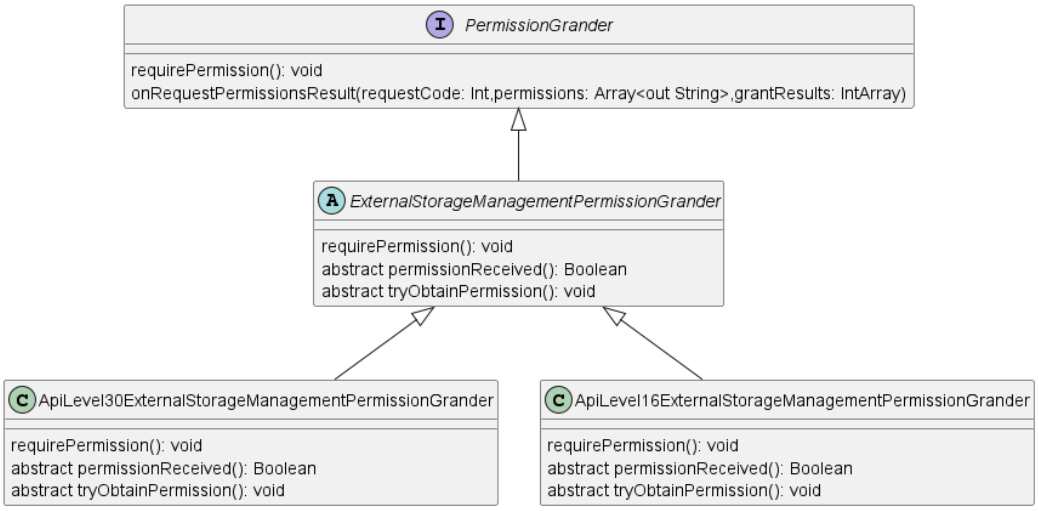


Рисунок 2.4 – Схема класів для отримання доступу до файлів на пристроїв для 16+ та 30+ версій Android API

Останнім елементом, якого не вистачає в нашій архітектурі, є фабрика (Factory). Цей компонент відіграє вирішальну роль, оскільки саме він буде

відповідати за стабільне створення конкретних реалізацій отримувачів доступу, або, за можливості та в разі ефективної оптимізації, за перевикористання вже існуючих екземплярів. Основною метою цієї фабрики є забезпечення того, щоб залежно від версії операційної системи та пристрою користувача була надана відповідна реалізація отримувача доступу.

Назвемо клас «ExternalStorageManagementPermissionGranderFactory». Така назва вибрана невипадково, адже вона чітко відображає його основну функцію — генерувати реалізації для доступу до зовнішнього сховища (external storage).

Фабрика повинна враховувати різноманітні умови та специфікації, характерні для різних версій операційної системи. Наприклад, доступ до зовнішнього сховища може відрізнятися в Android версії 6.0 (Marshmallow) від версії 11 (R), і тому для кожної з цих версій потрібні окремі реалізації отримувачів доступу.

Основна задача «ExternalStorageManagementPermissionGranderFactory» полягає в тому, щоб забезпечити правильний механізм запиту дозволів та доступу до файлів у зовнішньому сховищі для різних версій пристроїв користувачів. Це дозволить нам уникнути помилок та забезпечити стабільну роботу програми незалежно від версії операційної системи.

Такий підхід забезпечує високу гнучкість та масштабованість нашого рішення. Завдяки використанню фабричного методу ми можемо легко додавати підтримку нових версій операційної системи або модифікувати існуючі реалізації без значних змін в основному коді програми. Крім того, перевикористання існуючих екземплярів реалізацій, коли це можливо, допоможе оптимізувати споживання ресурсів і підвищити ефективність роботи застосунку.

2.2.2 Початкова обробка файлів

Щоб ефективно взаємодіяти зі знайденими файлами, необхідно привести їх у такий формат, який дозволяє легко розрізняти ці файли після обробки. Це особливо важливо для коректного відображення візуальних елементів, які мають різну логіку і потребують чіткого розмежування. Для досягнення цієї мети ми вирішили використовувати так званий TypedUri. Цей клас створений для того, щоб обробляти різні типи файлів, наприклад PDF та TXT документи, з урахуванням їх специфічних особливостей.

Наприклад, інформація про назву документа може зберігатися по-різному в залежності від типу файлу. У випадку з PDF-документами, назва може бути взята з метаданих файлу, тоді як для TXT-документів, які зазвичай не мають метаданих, достатньо використовувати ім'я файлу. Завдяки такій абстракції, ми можемо забезпечити правильне зчитування та обробку назв файлів незалежно від їх типу.

TypedUri дозволяє з легкістю додавати підтримку нових типів файлів у майбутньому. На даний момент реалізовано обробку лише двох типів файлів: PDF і TXT, але структура цього класу передбачає можливість розширення функціоналу без значних зусиль. Це робить систему більш гнучкою та масштабованою, готовою до інтеграції нових форматів файлів по мірі необхідності.

Загальний вигляд класу TypedUri представлений на рисунку 2.5, де видно його структуру та основні компоненти. Це включає в себе визначення типу файлу, методи для зчитування назв документів з метаданих або імен файлів та можливість додавання нових типів файлів з мінімальними змінами в коді.

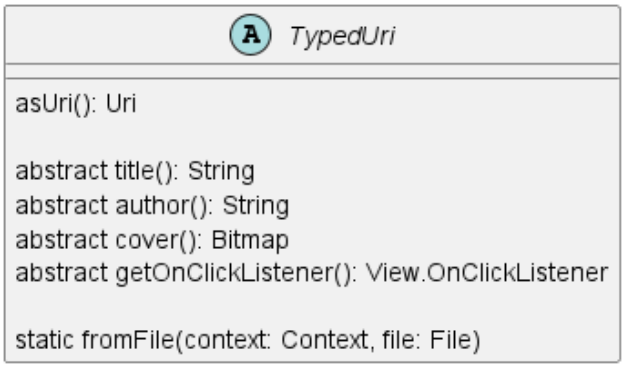


Рисунок 2.5 – Структура TypedUri

Як було вказано раніше, усі ці методи – це ті частини, що відрізняються від типу до типу. І відповідно до цього ми можемо створити перевизначені підтипи. Виглядатиме це наступним чином (рис. 2.6).

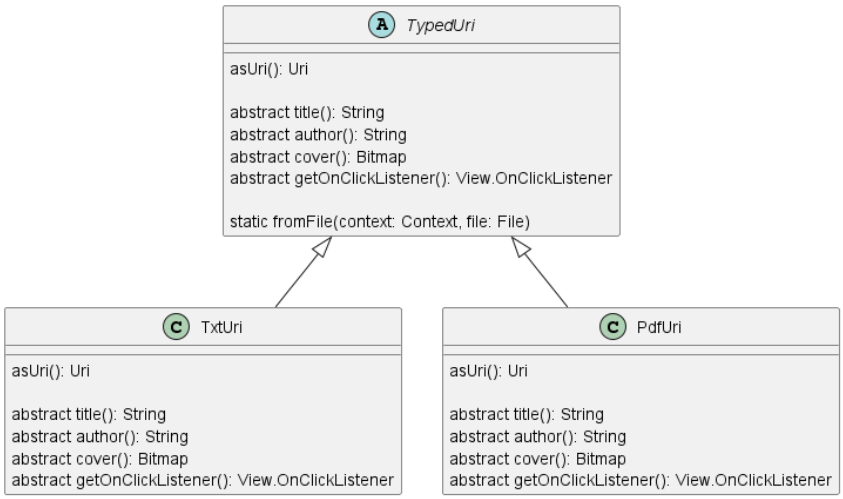


Рисунок 2.6 – Ієрархія TypedUri

2.2.3 Візуальні елементи

Як і всі інші програми для читання, наша система потребує наявності візуальних елементів, з якими користувач може взаємодіяти для комфортного і ефективного використання. У нашому випадку, подібно до більшості інших подібних програм, ми реалізуємо спеціальні «картки» книг, які служать для

вибору та перегляду конкретної книги. Ці картки є ключовим елементом інтерфейсу, що дозволяє користувачеві легко знайти потрібну книгу серед інших.

Кожна книга в нашій системі, залежно від її типу файлу, буде відображатися у відповідній обгортці, яка створена спеціально для цього типу файлу. Це означає, що ми маємо різні візуальні представлення для PDF, ePub, MOBI та інших форматів книг. Однак, незважаючи на різноманітність форматів, структура самої картки залишається незмінною. Змінюються лише конкретні параметри, такі як обкладинка книги, заголовок, автор, короткий опис та інші метадані, які забезпечують користувача необхідною інформацією про книгу.

Важливо, що всі ці елементи мають бути організовані таким чином, щоб забезпечити максимальну зручність для користувача. Картка повинна бути інтуїтивно зрозумілою, легкою для сприйняття та привабливою з візуальної точки зору.

Загальна схема вигляду картки книги представлена на рисунку 2.7. Цей рисунок демонструє, як саме організовані всі елементи на картці, забезпечуючи гармонійний і зрозумілий інтерфейс. Завдяки чіткій структурі і візуальній привабливості карток, користувач зможе легко знаходити і вибирати книги для читання, що значно покращує загальний досвід використання нашої програми.

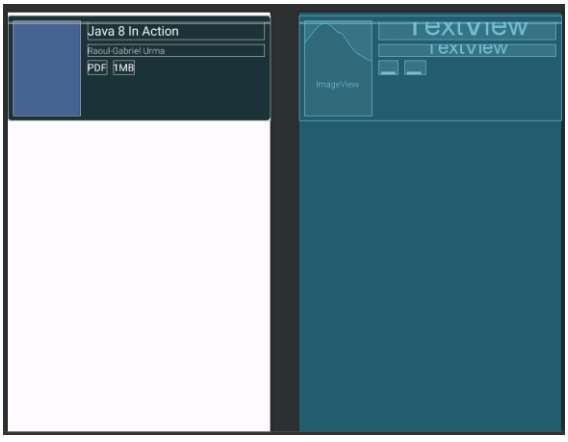


Рисунок 2.7 – Візуальна складова картки книги

Як видно на схемі (рис. 2.7), кожна книга, якщо має певні параметри, містить єдину структуру, але з різною логікою отримання тих чи інших даних, якими заповнюється сама картка. Це означає, що хоча кожна книга підпорядковується певній базовій структурі, спосіб, у який інформація витягується та відображається, може відрізнятися в залежності від конкретної книги.

Коли користувач натискає на картку, яка представляє книгу, до конкретного Uri (шляху до файлу) необхідно застосувати відповідну Activity. Activity в Android-застосунках - це спеціальний елемент, який можна охарактеризувати як «сцену», на якій відбуваються поточні дії і з якою користувач може взаємодіяти у цей момент. Інакше кажучи, кожна Activity визначає певний інтерфейс користувача та логіку роботи з цими інтерфейсами. Це дозволяє структурувати застосунок таким чином, щоб кожна окрема дія мала свій власний простір для відображення і взаємодії з користувачем.

Така структура передбачає певну залежність між Activity та типом файлу, з яким вона працює. Це означає, що для кожного типу файлу існує своя Activity, яка забезпечує правильне відображення та обробку даних. Наприклад, якщо потрібно відобразити текстову інформацію, буде використовуватися одна Activity, яка має всі необхідні елементи для роботи з текстом, такі як можливість змінювати розмір шрифту, залишати закладки чи змінювати фон сторінок.

З іншого боку, для роботи з графічними даними застосовується інша Activity, спеціально оптимізована для відображення зображень. Вона може включати інструменти для масштабування, перегляду в повноекранному режимі та інші функції, які забезпечують комфортне переглядання графічного контенту. Такий підхід дозволяє максимально ефективно використовувати можливості пристрою і забезпечувати найкращий користувацький досвід, оскільки кожна Activity адаптована для оптимальної роботи з конкретним типом контенту.

Таким чином, натискання на картку книги призводить до активації відповідної Activity, яка зчитує Uri файлу, забезпечує його правильне відображення та надає всі необхідні інструменти для взаємодії з цим файлом. Це дозволяє користувачам зручно і ефективно працювати з різними типами файлів, використовуючи інтерфейси, спеціально створені для кожного типу даних.Таким чином, загальну схему взаємодії можна відобразити наступним чином (рис. 2.8), де кожна Activity асоціюється з конкретним типом файлу і забезпечує відповідну логіку обробки і відображення даних. Це дозволяє створити гнучку і розширювану систему, яка може легко адаптуватися до різних типів контенту, зберігаючи при цьому єдину структуру даних для всіх книг.

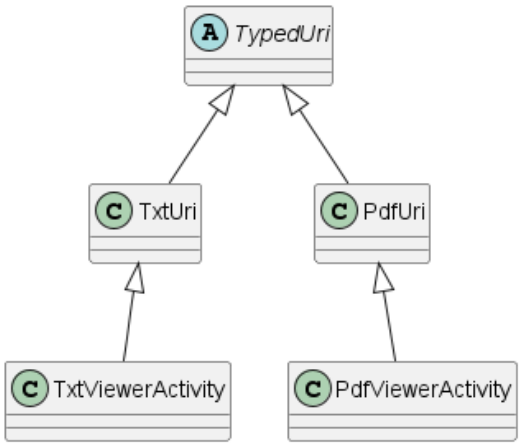


Рисунок 2.8 – Схематичне розподілення залежностей ViewerActivity від типу Uri

Тепер звернемо увагу на головну сторінку, де ми будемо переглядати різноманітні книги, щоб вибрати ту, яку хочемо прочитати в даний момент. Основна структура цієї сторінки оформлена у вигляді списку карток, розташованих вертикально одна під одною з невеликим проміжком між ними (рис. 2.9). Кожна картка представляє окрему книгу і містить її обкладинку, назву, автора та короткий опис або анотацію. Така схема розміщення дозволяє легко і швидко ознайомитися з доступними варіантами і прийняти

рішення про вибір наступної книги для читання. Візуальна презентація карток зроблена таким чином, щоб кожен елемент був чітко відокремлений від інших, але водночас не займав надто багато простору, що дозволяє користувачеві комфортно переглядати декілька опцій одночасно без необхідності постійно прокручувати сторінку.

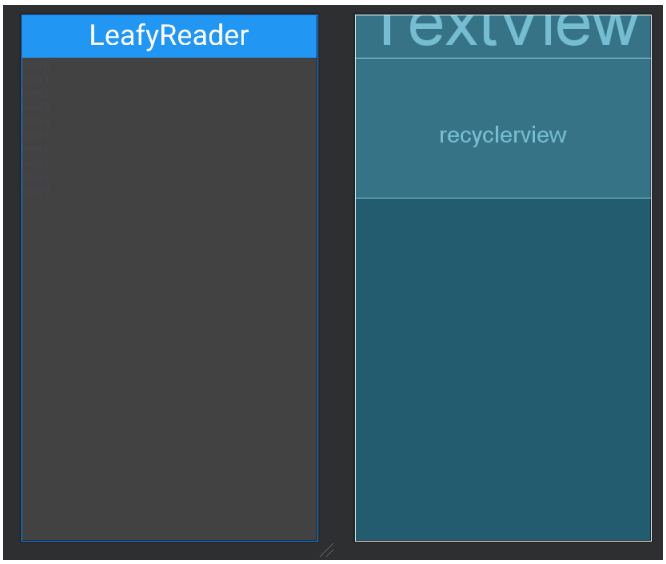


Рисунок 2.9 – Візуальна складова нашої головної сторінки застосунку

Завдяки нескладному візуальному оформленню, реалізація усіх необхідних маніпуляцій із зображеннями та елементами інтерфейсу стає досить простою. RecyclerView в Android надає нам можливість не турбуватися про оптимізацію роботи нашого застосунку, оскільки він автоматично управляє динамічним завантаженням елементів під час прокрутки списку. Це означає, що елементи інтерфейсу завантажуються поступово, по мірі їхнього з'явлення на екрані, що значно знижує навантаження на систему.

З урахуванням розміру екрану нашого пристрою, одночасно на ньому може відображатися максимум чотири елементи. Це означає, що навіть при інтенсивному використанні застосунку, коли користувач швидко перегортає список, система повинна обробляти лише невелику кількість елементів

одночасно. Таким чином, на початкових етапах розробки не повинно виникати жодних проблем із продуктивністю та завантаженням даних.

У підсумку, використання RecyclerView у нашому застосункові забезпечує ефективне управління пам'яттю та ресурсоємність, завдяки чому ми можемо сконцентруватися на функціональності та зручності користування, а не на технічних деталях оптимізації відображення елементів. Це дозволяє створити більш плавний та приємний для користувача інтерфейс, що є важливим аспектом успіху будь-якого мобільного застосунку.

Також було вдосконалено роботу із View карток книг. Щоб легше взаємодіяти з їх полями, можна використати так званий «RecyclerView.ViewHolder», перевизначивши який, і створивши в ньому правильні поля, ми зможемо напряму встановлювати ним необхідні нам значення. Структура такого ViewHolder-а на прикладі BookViewHolder схематично виглядає наступним чином (рис. 2.10).

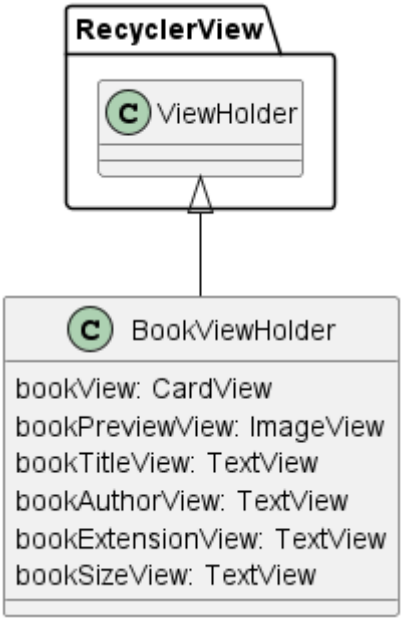


Рисунок 2.10 – Схематичне зображення BookViewHolder

2.3 Додаткова структуризація книг

Як можна здогадатись, не всі книги можуть задовольнити нашу структуру «карток», яку ми для них підготували. Деякі книги мають настільки довгі назви, що вони б з легкістю порушили весь візуальний стиль і формат нашого макета. Наприклад, назви, що складаються з багатьох слів або включають довгі фрази, не вписуються у відведені для них місця на картках. Це створює необхідність застосування підходу до скорочення назв.

Для вирішення цієї проблеми ми розробили спеціальний алгоритм, що автоматично скорочує назви залежно від їхньої довжини. Цей алгоритм працює таким чином, що довгі назви обрізаються до певного допустимого розміру, при цьому в кінці скороченої назви додаються три крапки («…»), які вказують на те, що назва була урізана. Алгоритм налаштований так, щоб враховувати різні фактори, включаючи кількість символів, слів і навіть змістовне навантаження заголовків. Завдяки цьому він ефективно визначає оптимальний момент для вставки трикрапок, зберігаючи сутність назви і основну інформацію, яку вона передає.

Застосування цього алгоритму дозволяє зберегти візуальну цілісність нашої структури «карток», забезпечуючи водночас зрозумілість та читабельність інформації. Наприклад, якщо книга має назву «Фантастичні пригоди у вигаданому світі: історія, яка змінює життя», наш алгоритм може скоротити її до «Фантастичні пригоди у вигаданому світі…», що зберігає основну суть і ключові слова, дозволяючи водночас уникнути перевантаження картки текстом. Цей підхід забезпечує користувачам приємний і зручний досвід взаємодії з інтерфейсом, дозволяючи легко орієнтуватися серед великої кількості книг.

Отже, наданий алгоритм є важливою складовою нашої системи відображення інформації про книги, яка забезпечує естетичний вигляд і функціональність нашого візуального інтерфейсу. Завдяки цьому рішенню ми можемо підтримувати акуратний та привабливий вигляд наших карток,

навіть коли маємо справу з дуже довгими назвами. Нижче наведено схематичне зображення цього алгоритму (рис. 2.11), що ілюструє основні принципи його роботи та процес скорочення назв до прийнятної довжини.

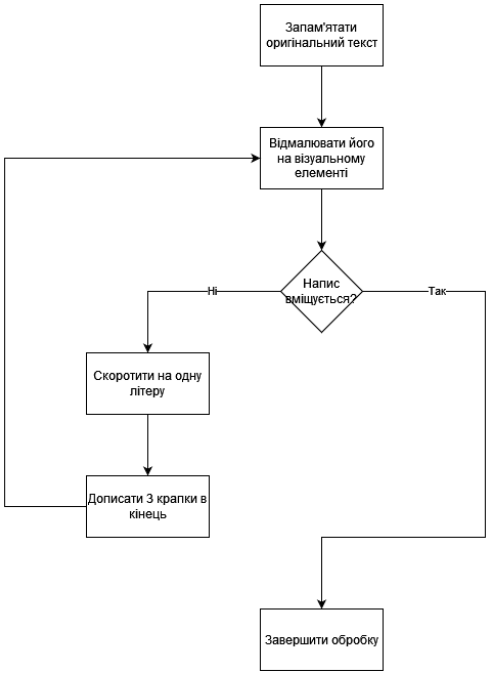


Рисунок 2.11 – Алгоритм скорочення назв книг

Наведений алгоритм, представлений на рисунку 2.11, має велике значення, незважаючи на його на перший погляд простий характер. Цей алгоритм впливає на продуктивність інтерфейсу шляхом оптимізації рендерингу елементів. Використання цього методу має ряд переваг. По-перше, відпадає необхідність у постійній перевірці просторового розміщення тексту, що запобігає його перетину з іншими елементами інтерфейсу. Це сприяє уникненню конфліктів та покращує зовнішній вигляд інтерфейсу, забезпечуючи при цьому стабільність його функціоналу.

Додатково, застосування вказаного алгоритму знижує обчислювальну навантаженість системи. Це особливо актуально для мобільних пристроїв з обмеженими ресурсами, де оптимізація продуктивності має велике значення. Мінімізація кількості рендерингу дозволяє підвищити швидкість роботи додатку і покращити загальний користувацький досвід.

Крім того, зменшення кількості рендерингу сприяє уникненню потенційних помилок при взаємодії з інтерфейсом, забезпечуючи стабільність та передбачуваність роботи додатку. Користувачі отримують зручний та надійний інструмент для взаємодії з контентом без втручання в технічні деталі. Такий підхід сприяє підвищенню ефективності використання додатку та позитивно впливає на загальне враження від його роботи.

Додатковою перевагою даного алгоритму є його вплив на оптимізацію використання ресурсів пристрою. За допомогою ефективного управління пам'яттю та обчислювальними ресурсами, цей алгоритм дозволяє забезпечити оптимальну продуктивність додатку при мінімальному споживанні енергії. Це особливо актуально для мобільних пристроїв, де важливо зберігати заряд акумулятора для тривалого використання пристрою.

Ще однією перевагою є можливість адаптації алгоритму до змінюваних умов інтерфейсу. Завдяки гнучким налаштуванням, розробники можуть вибирати оптимальні параметри для конкретного додатку, що дозволяє досягти найкращої продуктивності та ефективності. Такий підхід сприяє максимальному використанню можливостей пристрою та забезпечує плавну та приємну роботу інтерфейсу для кінцевого користувача.

Крім того, дана технологія може бути легко інтегрована з іншими модулями та компонентами системи. Це відкриває широкі можливості для розширення функціональності додатку та вдосконалення його властивостей. Інтеграція з іншими інструментами оптимізації та управління ресурсами дозволяє створювати комплексні рішення для досягнення найвищих стандартів якості та продуктивності.

Додатковою перевагою цього алгоритму є його здатність до автоматизації процесів управління контентом, що сприяє підвищенню продуктивності та забезпечує зручний користувацький досвід.

3 КОМП’ЮТЕРНА МОДЕЛЬ ЗАСТОСУНКУ ДЛЯ ЧИТАННЯ ФАЙЛІВ

3.1 Обґрунтування вибору середовища програмної реалізації

Під час виконання моєї дипломної роботи я звернувся до розробки мобільного додатка Android Reader, що спеціалізується на роботі з PDF-файлами. Обравши середовище розробки Android Studio та мову програмування Kotlin, я був готовий використати їх потужність для створення інноваційних та ефективних рішень у мобільному програмуванні. Проект Android Reader став для мене відмінною можливістю розширити мої знання та навички в галузі мобільної розробки, а також поглибити моє розуміння архітектури Android.

Моє завдання полягало в створенні функціоналу, який забезпечував зручність і простоту взаємодії з PDF-файлами, включаючи їх завантаження, відображення та роботу з контентом. У процесі розробки я створив інтуїтивно зрозумілий інтерфейс для користувачів, що дозволяв зручно та ефективно використовувати функціонал додатка.

Мобільні застосунки, такі як Android Reader, мають специфічні вимоги до ефективності використання ресурсів пристрою та оптимізації відображення контенту. Використання Android Studio та Kotlin дозволило мені оптимізувати роботу додатка, забезпечуючи його високу продуктивність та якість.

Отже, розробка Android Reader стала для мене значним кроком у професійному розвитку, дозволяючи отримати важливий практичний досвід у галузі мобільної розробки та створити корисний та практичний продукт для користувачів.

3.2 Програмна реалізація

3.2.1 Обробка доступу

В коді на даний момент створено інтерфейс, який дозволяє перевизначити різну поведінку для різних версії Android. Інтерфейс має наступний вигляд (рис. 3.1).

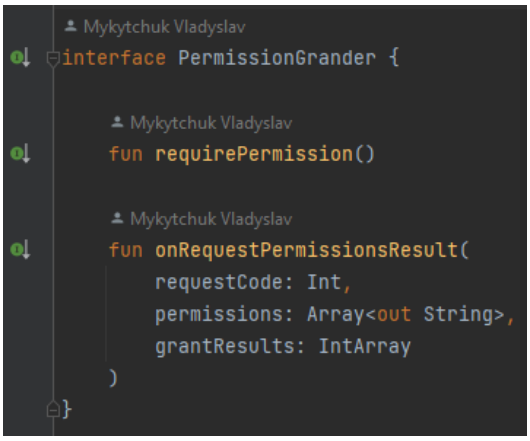


Рисунок 3.1 – Інтерфейс PermissionGrander, що дозволяє давати запит на різні види доступів

На даний момент наш застосунок потребує лише один вид доступу, тому створено «ExternalStorageManagementPermissionGrander», що кидає запит на доступ до файлів на пристрої. Має цей абстрактний клас наступний вигляд (рис. 3.2).

Рисунок 3.2 – Код абстрактного класу «ExternalStorageManagementPermissionGrander»

Він має на даний момент одну реалізацію, для версії Android 11 (Android API 30). У цієї версії з’являються і відповідні їй особливі способи звертання, не характерні версіям нижче. Перш за все, в нас є змога запросити цей доступ напряму з застосунку, не змушуючи користувача йти до налаштувань самостійно. Виглядає реалізація наступним чином (рис. 3.3).

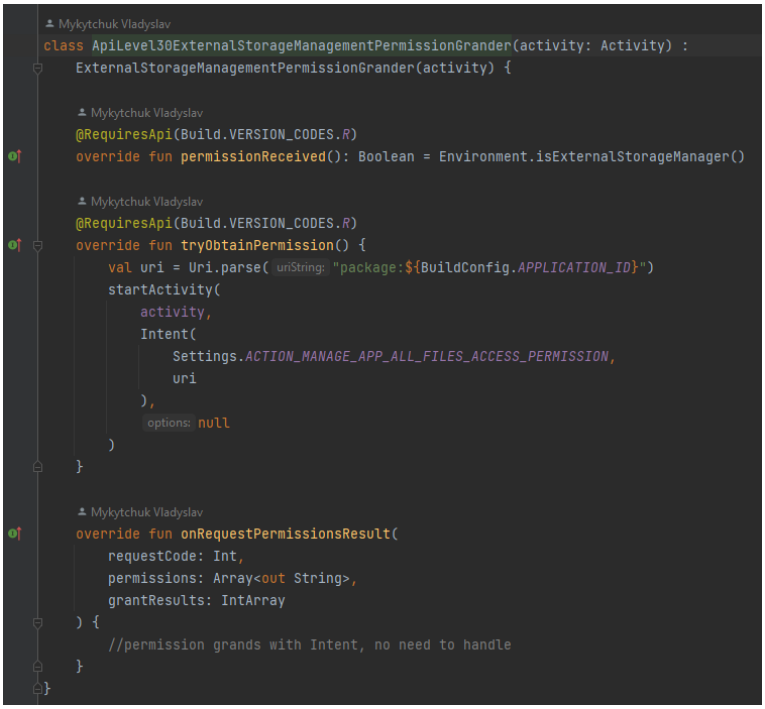


Рисунок 3.3 – Реалізація ExternalStorageManagementPermissionGrander для Android 11 та вище

Тепер, коли з цим питанням розібрались, як саме ми будемо шукати файли?

Алгоритм дій, якого слід дотримуватись, включає наступні етапи. Спочатку потрібно визначити кореневий каталог, з якого розпочнеться пошук. Важливо врахувати, що кореневий каталог може відрізнятись в залежності від версії Android. Після цього програма рекурсивно переглядає всі підкаталоги та файли у визначеному кореневому каталозі. Для кожного елементу, тобто каталогу чи файлу, проводиться перевірка на відповідність критеріям пошуку, які можуть включати, наприклад, ім'я, розмір або тип файлу. При знаходженні відповідного елементу його можна включити в

список знайдених файлів або відповідно обробити. Якщо поточний елемент є каталогом, процес рекурсивно повторюється для цього каталогу. Пошук завершується, коли всі каталоги та файли переглянуті або досягнуто заданий обсяг пошуку.Важливо враховувати ефективність алгоритму, особливо при пошуку великої кількості файлів. Наприклад, можна оптимізувати алгоритм, використовуючи паралельні потоки для пошуку на багатоядерних системах або використовуючи індексацію файлів для прискорення пошуку.

Кодом це реалізовано наступним чином (рис. 3.4).

Рисунок 3.4 – Реалізація пошуку файлів на пристрої

Як можна було помітити, в нас є дивний метод TypedUri.fromFile(context, file). Як і повертаємо ми конкретно TypedUri. Навіщо це потрібно? Справа у тому, що під час відображення файлів ми маємо певним, відмінним для різних форматів, чином отримувати метадані файлів задля зручного перебігу по ним. Задля цього ми зробили доволі цікаву «обгортку», в яку поміщаємо Uri знайдених файлів. Подібні структурні рішення дозволяють легше орієнтуватись по коду, писати його чистіше, а також пришвидшують розробку застосунку у випадках, коли необхідно додати щось нове до застосунку. Більш того, це дозволяє писати надійний код, який легше тестувати і згодом використовувати.

3.2.2 Обробка особливостей файлів

Розглядаючи необхідність розрізнення Uri для різних форматів файлів у нашому додатку, ми вирішуємо завдання з більш глибокої та комплексної перспективи. Кожен файл, будучи унікальним екземпляром даних, має властивості, які вимагають індивідуального та гнучкого підходу до його обробки. Ця унікальність може виявитися в різноманітних аспектах, таких як формат даних, кодування, структура, метадані та інші характеристики. Таким чином, для ефективної та коректної обробки кожного типу файлу потрібно враховувати всі ці нюанси.

Основним завданням стає розробка адаптивних та універсальних механізмів обробки файлів, які забезпечать оптимальну взаємодію з різноманітними форматами. Це передбачає не лише створення спеціалізованих алгоритмів для кожного типу файлу, але й їх інтеграцію та взаємодію з іншими складовими додатку. Такий підхід дозволить забезпечити максимальну сумісність з різними форматами файлів та забезпечить їхню оптимальну обробку та відображення.

Додатково, важливою є імплементація механізмів управління ресурсами та оптимізації обробки файлів для забезпечення високої продуктивності та стабільності роботи додатку. Це включає в себе ефективне використання пам'яті, оптимізацію часу завантаження та обробки файлів, а також управління конкурентним доступом до ресурсів. Такий комплексний підхід дозволить створити додаток, який буде ефективно працювати з будь-якими типами файлів та забезпечить високу якість користувацького досвіду.

Крім того, важливою складовою розробки додатку є врахування потенційних сценаріїв взаємодії користувача з різними типами файлів та відповідне адаптування інтерфейсу для забезпечення зручності та інтуїтивності користування. Наприклад, для файлів різних форматів можуть бути використані різні елементи управління або режими перегляду, що відповідають їхньому змісту та функціональності. Такий підхід дозволить

максимально врахувати потреби користувачів та забезпечити їм зручний та ефективний інструмент для роботи з файлами.

Рисунок 3.5 – Обгортка TypedUri

В даній обгортці ми маємо наступні перевизначені для різних типів файлів методи:

– title() – даний метод повертає назву файлу;

– author() – даний метод повертає автора файлу (якщо його можна знайти в метаданих відповідного файлу);

– cover() – даний метод повертає обгортку для книжки. Ця обгортка може бути як в метаданих, так і згенерована певним чином;

– getOnClickListener() – даний метод потрібен, щоб при натисканні конкретного файлу виконувалась конкретна послідовність дій задля правильного відображення файлу на пристрої. Різні файли мають різні реалізації відображення.

Окрім цього ми також маємо мати змогу звернутись до обгортуваного Uri, задля чого використовуємо метод «asUri()». Ну і відповідно статичний фабричний метод, щоб ми могли отримати відповідний «типізований Uri» (TypedUri) на основі розширення файлу. На даний момент реалізовано розрізнення файлів типів PDF, DOC, TXT.

Тепер розглянемо особливості PDF файлів з точки зору коду. Оскільки його багато, розподілимо на першу частину (рис. 3.6), та другу (рис. 3.7).

Рисунки 3.6– Перша частина реалізації TypedUri на основі PDF

Рисунки 3.7 – Друга частина реалізації TypedUri на основі PDF

У коді нашого застосунку ми ініціалізуємо «ліниве» поле, яке отримує метадані файлу. Це означає, що ініціалізація цього поля відбудеться лише один раз - пізніше, і отримане значення буде використовуватися для всіх подальших звертань до цього поля. Ми отримуємо ці метадані з файлу лише один раз і тут же закриваємо файл, щоб уникнути проблем з витратою ресурсів пристрою, особливо коли цей файл може бути використаний іншими застосунками. Використовуємо для цього бібліотеки «com.shockwave.pdfium.PdfDocument» і «com.shockwave.pdfium.PdfiumCore».

На щастя, на сьогоднішній день існує чимало відмінних інструментів для роботи з PDF-файлами, що дозволяє нам уникнути зайвих зусиль на створення власної бібліотеки. З отриманих метаданих ми отримуємо назву книги, яка, можливо, не співпадає з ім'ям файлу, та автора книги. Але іноді при завантаженні будь-якого файлу з PDF автора замінюють на сайт, з якого було скачано файл. Тому, в такому випадку, краще нічого не відображати. Щодо обкладинки книги, у PDF-файлах вона також не завжди міститься в метаданих. Тут деяка дотриманість полягає у тому, що перша сторінка PDF-файлу вважається обкладинкою. Це може бути навіть і не книга, а, наприклад, виписка з банку тощо, і тоді обкладинка буде простою першою сторінкою. Так чи інакше, для зручності користувача ми відображаємо приблизний зміст книги на обкладинці.

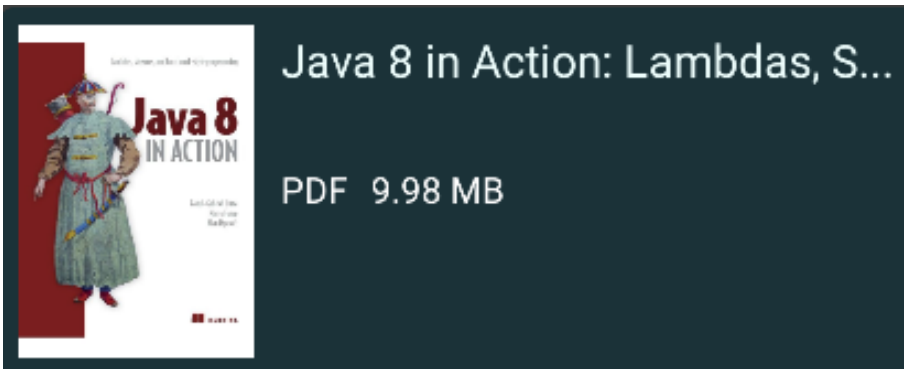


Рисунок 3.8 – Картка PDF-файлу на прикладі книги з мови програмування Java «Java 8 in Action: Lambdas, Streams, and Functional-Style Programming»

Далі розглянемо текстові файли (з розширенням TXT). Дані файли не потребують особливої обробки, як і не містять будь-яких метаданих самих по собі, через що реалізація даного типу файлів є доволі простою (рис. 3.9).

Рисунок 3.9 – Код реалізації TypedUri для файлів txt-формату

Оскільки інформація про назву книги та автора відсутня в метаданих текстового файлу, нам доводиться користуватися іншими джерелами для отримання цих даних. Так, для назви книги ми використовуємо просто назву файлу без розширення. Щодо автора, його також не вказано в метаданих, тому ми вирішили приховати цю інформацію. Щодо обкладинки книги, ми вирішили спростити процес для користувача, використовуючи першу літеру назви файлу як обкладинку. Проте, якщо перша літера відсутня (наприклад, у випадку файлу з назвою «.txt»), ми розміщуємо напис «TXT» на обкладинці. Це допомагає зробити вміст книги більш зрозумілим для користувача, сприяючи швидшому орієнтуванню серед файлів. Ця практика також допомагає уникнути можливих проблем з функціонуванням застосунку, оскільки недостатньо коректно оброблені файли можуть спричинити непередбачені помилки. Давайте розглянемо приклад для файлу з назвою «sample3.txt» (див. рис. 3.10).

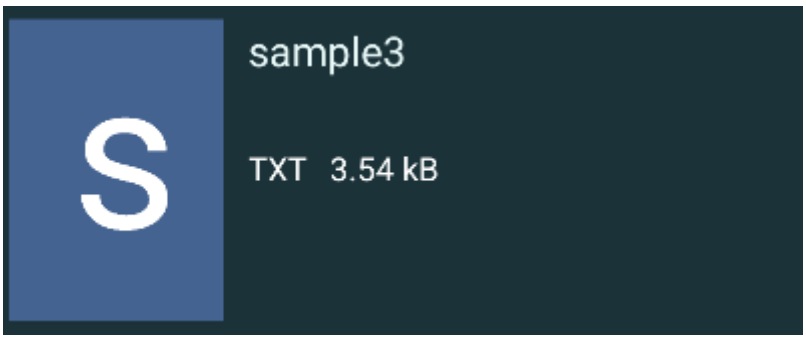


Рисунок 3.10 – Картка txt-файлу на прикладі текстовика-прикладу «sample3»

В майбутньому можна придумати нові способи обробки подібних ситуацій, задля покращення користувацького досвіду, але поки що зупинимось на цьому.

Окрім того, якщо назва книги занадто велика, щоб відобразити її на карточці, ми використовуємо її скорочення. Це зроблено задля того, щоб усі книжні картки мали однаковий формат, і не псували структуру в нашому застосунку (не зміщувалася назва нижче ніж їй відведено, самі картки були одного розміру тощо). Тому в нашому коді застосовано на такий випадок так званий «TextShortening», про який поговоримо далі.

3.2.3 Стискання тексту

Стискання тексту є дуже важливою частиною створення книго-карток, щоб вони виглядали в однаковому стилі, та не кидались в очі, якщо назва файлу занадто довга. Ця ситуація є доволі частою, оскільки безліч назв книг не вміщаються в умовні 15 символів. Для таких випадків і було реалізовано «TextShortening».

Для прикладу візьмемо PDF-книгу «Kotlin in Action, Second Edition». Її картка має наступний вигляд (рис. 3.11).

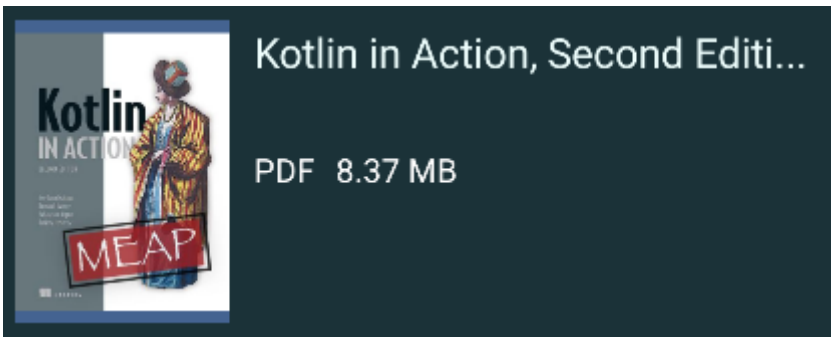


Рисунок 3.11 – Картка PDF-файлу книги «Kotlin in Action, Second Edition»

Можна помітити, що назва книги є трохи довшою за саму картку, через що довелось скоротити назву, використовуючи «…». Тепер поговоримо про сам алгоритм дії.

Працює усе наступним чином: Під час ініціалізації TextView (спеціальні текстові поля на картці, які можна в коді змінювати) необхідно заміряти, чи влізає текст без змін. Якщо він довше, ніж має бути (тобто його довжина, відобразивши її на картці, більше ніж довжина самого поля на картці), то ми зменшуємо розмір тексту до тих пір, поки скорочений текст з дописаними трьома крапками («…») не стане меншим за шириною, ніж довжина текстового поля. Кодом це має наступний вигляд (рис. 3.12).

```
Mykytchuk Vladyslav
private fun shortenTextDefault(textView: TextView) {
    val startTime = System.nanoTime()
    val originalText: CharSequence = textView.text
    val paint: Paint = textView.paint
    var width = paint.measureText(originalText,  start: 0, originalText.length)

    if (width > textView.width) {
        var maxLength = originalText.length
        val ellipsisWidth = paint.measureText( text: "...")
        while (width + ellipsisWidth > textView.width) {
            maxLength--
            width = paint.measureText(originalText,  start: 0, maxLength)
        }
        val shortenedText: CharSequence =
            originalText.subSequence(0, maxLength).toString() + "..."
        textView.text = shortenedText
    } else {
        textView.text = originalText
    }
    Log.i( tag: "speedNanos",  msg: "${System.nanoTime() - startTime}nanos")
}
```

Рисунок 3.12 – Код для скорочення тексту на TextView за необхідності

Також нам потрібно не забувати, що різні версії Android API мають різні специфіки ініціалізації TextView. Якщо бути точним, то починаючи з Android 4.1 (Android API 16) при перевизначенні OnGlobalLayoutListener-а (це той метод, який відповідає за взаємодію з TextView під час ініціалізації) нам також необхідно видалити цей зчитувач з глобального переліку зчитувачів. Виглядає обробка в залежності від версії наступним чином (рис. 3.13).

```
private val onGlobalLayout =
    when (Build.VERSION.SDK_INT) {
        in 1 ≤..<Build.VERSION_CODES.JELLY_BEAN -> { textView: TextView ->
            OnGlobalLayoutListener { shortenTextDefault(textView) }
        }

        else -> { textView: TextView ->
            object : OnGlobalLayoutListener {
                override fun onGlobalLayout() {
                    textView.viewTreeObserver.removeOnGlobalLayoutListener( victim: this)
                    shortenTextDefault(textView)
                }
            }
        }
    }
```

Рисунок 3.13 – Код скорочення тексту в залежності від версії Android поточного користувача застосунку

І тепер, коли в «onGlobalLayout» змінній знаходиться конкретна реалізація для поточного користувача, ми можемо її викликати напряму вже публічним методом (або функцією в випадку Kotlin) (рис. 3.14).

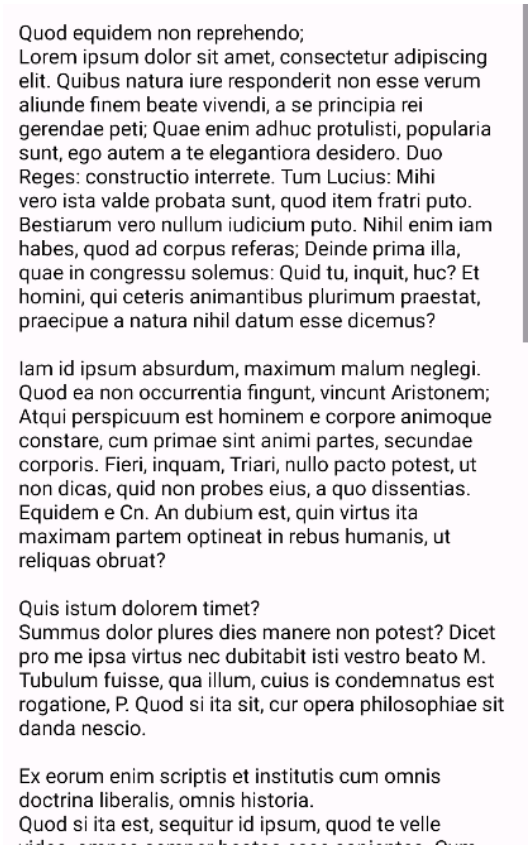
```
Mykytchuk Vladyslav
fun shortenText(textView: TextView) {
    textView.viewTreeObserver.addOnGlobalLayoutListener(onGlobalLayout.invoke(textView))
}
```

Рисунок 3.14 – Метод «shortenText», що використовується для «TextView» заголовків на картках книг

За рахунок такої дивної структури ми тим самим пришвидшуємо майбутні звертання до цієї функції, оскільки розподілили різні реалізації одним when statement-ом, що виконається при запуску застосунку.

3.2.4 Різні види Activity застосунку

Розглянемо більш детально процес відображення різних типів файлів у нашій програмі, оскільки це має значний вплив на користувацький досвід і функціональність застосунку. Перш за все, слід відзначити, що цей аспект є однією з ключових складових нашої розробки, яку можна безперервно вдосконалювати і вдосконалювати у майбутньому, незалежно від питань дизайну або оптимізації. Коли користувач натискає на картку книги чи іншого файлу, інтерфейс повинен відреагувати адекватно і відобразити вміст цього файлу. Наприклад, при розгляді текстових файлів користувач повинен мати можливість прокручувати текст та зручно читати його на екрані. Таким чином, важливо, щоб програма правильно інтерпретувала тип файлу та відображала його відповідно до його призначення. Давайте розглянемо приклад з текстовим файлом «sample3.txt», який був згаданий раніше, щоб краще зрозуміти цей процес (див. рис. 3.15).


Quod equidem non reprehendo;
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quibus natura iure responderit non esse verum aliunde finem beate vivendi, a se principia rei gerendae peti; Quae enim adhuc protulisti, popularia sunt, ego autem a te elegantiora desidero. Duo Reges: constructio interrete. Tum Lucius: Mihi vero ista valde probata sunt, quod item fratri puto. Bestiarum vero nullum iudicium puto. Nihil enim iam habes, quod ad corpus referas; Deinde prima illa, quae in congressu solemus: Quid tu, inquit, huc? Et homini, qui ceteris animantibus plurimum praestat, praecipue a natura nihil datum esse dicemus?

Iam id ipsum absurdum, maximum malum neglegi. Quod ea non occurrentia fingunt, vincunt Aristonem; Atqui perspicuum est hominem e corpore animoque constare, cum primae sint animi partes, secundae corporis. Fieri, inquam, Triari, nullo pacto potest, ut non dicas, quid non probes eius, a quo dissentias. Equidem e Cn. An dubium est, quin virtus ita maximam partem optineat in rebus humanis, ut reliquas obruat?

Quis istum dolorem timet?
Summus dolor plures dies manere non potest? Dicet pro me ipsa virtus nec dubitabit isti vestro beato M. Tubulum fuisse, qua illum, cuius is condemnatus est rogatione, P. Quod si ita sit, cur opera philosophiae sit danda nescio.

Ex eorum enim scriptis et institutis cum omnis doctrina liberalis, omnis historia.
Quod si ita est, sequitur id ipsum, quod te velle

Рисунок 3.15 – Відображення тексту з файлу sample3.txt

Оскільки файл «sample3.txt» є лише звичайним текстовим документом без будь-якої форматувальної розмітки чи складнішої вмістової структури, такої як різні кольори або розміри шрифтів, а також відсутності в ньому графічних елементів, виникає необхідність у простому методі його відображення. Тому ми вирішили звернутися до простого рішення, яке полягає у склеюванні текстового вмісту файлу та подальшому відображенні цього вмісту в одному великому TextView. Хоча такий підхід може не бути оптимальним з точки зору використання ресурсів пристрою, особливо у випадку великих обсягів даних, наразі ми зупинились на цьому варіанті, оскільки він задовольняє поточні вимоги. Для реалізації цього підходу наведено приблизний вигляд коду (див. рис. 3.16), який пропонує простий та зрозумілий механізм відображення текстового вмісту файлу у мобільному застосунку.

```
Mykytchuk Vladyslav *
class TxtViewerActivity: AppCompatActivity() {

    Mykytchuk Vladyslav *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_view_txt_file)

        val textView = findViewById<TextView>(R.id.textView)

        // Get the intent that started this activity
        val intent: Intent? = intent
        val uri: Uri = intent?.getParcelableExtra( name: "txtUri")!!

        uri.let { fileUri ->
            try {
                // Open the URI for reading
                contentResolver.openInputStream(fileUri)?.use { inputStream ->
                    val reader = BufferedReader(InputStreamReader(inputStream))
                    val stringBuilder = StringBuilder()
                    var line: String?
                    // Read lines from the file and append to the StringBuilder
                    while (reader.readLine().also { line = it } != null) {
                        stringBuilder.append(line).append("\n")
                    }
                    // Set the text of the TextView to the contents of the file
                    textView.text = stringBuilder.toString()
                } ^let
            } catch (e: Exception) {
                e.printStackTrace()
                textView.text = getString(R.string.something_went_wrong_during_file_opening)
            }
        }
    }
}
```

Рисунок 3.16 – Код реалізації TxtViewerActivity

Як можна помітити, також в цьому коді оброблено виключну ситуацію, при якій при виникненні будь-яких проблем під час відкриття файлу станеться помилка, про наявність помилки напише замість тексту самого файлу. Це також необхідно задля того, щоб наш застосунок був значно надійнішим і не виникало помилок, які б закривали проєкт тощо.

Тепер перейдемо до PDF-файлів. На відміну від txt-файлів, цей формат містить значно більше особливостей, через що робота з ним є важчою. На щастя, існує безліч бібліотек, які задовільняють наші потреби, тому ми використаємо одну з таких, підлаштувавши під наші потреби.

Перш за все, слід зауважити, що для відображення PDF-файлів було використано гітхаб-бібліотеку «com.github.barteksc.pdfviewer». Ця бібліотека надає можливості для відображення як цілого файлу, так і його конкретних сторінок, а також дозволяє налаштувати «listener-и», тобто реагування застосунку на певні дії користувача, такі як натискання, віджимання, зміну звуку тощо. В нашому випадку нам потрібно перевизначити поведінку застосунку при натисканні на праву та ліву частину екрану (в нашій реалізації це переносить нас на наступну/минулу сторінки відповідно). В випадку, що це остання сторінка при натисканні вправо нічого не відбудеться, як і при натисканні вліво при першій сторінці. Виглядає все налаштування наступним чином (рис. 3.17).

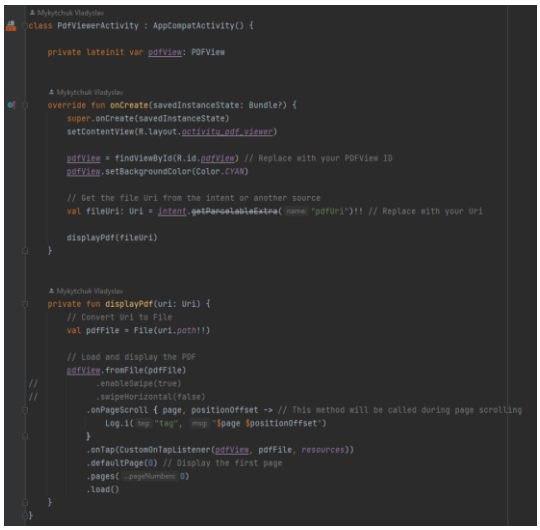


Рисунок 3.17 – Код реалізації PdfViewerActivity

Тепер варто зазначити нашу власну перевизначену логіку для «CustomOnTapListener». Він потрібен для того, щоб використовувати власну реалізацію перелистування сторінок, що окрім візуальних змін також є більш оптимальним, якщо користувач хоче переглянути не весь документ, а лише перші кілька сторінок, оскільки вони будуть завантажуватись при натисканнях на екран. Такий підхід також має свої мінуси, оскільки найчастішим сценарієм є той, коли користувач пролистує приблизно 5 сторінок поспіль, через що можна було б їх відкривати та зберігати у пам’яті одразу, але поки що зупинимось на цьому варіанті.

3.3 Інструкція користувача

При запуску програми-читалки користувач одразу бачить усі файли на пристрої, чиї розширення вже реалізовані для відображення. Виглядатиме це наступним чином (рис. 3.18).

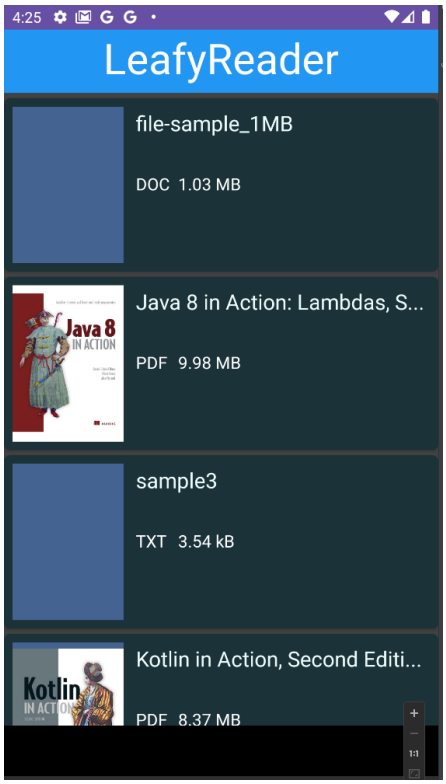


Рисунок 3.18 – Приклад відображення книг, знайдених на пристрої

Щоб переглянути обрану книгу зі списку доступних книг, користувачу просто необхідно клацнути будь-яку точку на «карточці» відповідної книги. У цьому контексті клацабельною є вся темно-синя зона, яка представляє собою графічний елемент карточки. Ця дія запускає відкриття обраного PDF-файлу. Для наочності, процес відкриття PDF-файлу показано на рисунку 3.19.

Рисунок 3.19 – Відображення PDF-книги при відкритті в застосунку

Механіка перегляду сторінок у програмі впроваджена таким чином, щоб забезпечити максимальну зручність та ефективність користування. Користувач може перегортати сторінки, натискаючи на праву або ліву сторону екрану відповідно, щоб переходити на наступну або попередню сторінку відповідно до поточного контексту. Цей підхід забезпечує інтуїтивну навігацію, яка дозволяє швидко переходити між різними частинами документу, забезпечуючи при цьому мінімальні зусилля від користувача.

У застосунок до перегортання сторінок, програма також підтримує можливість масштабування вмісту. Користувач може збільшувати або

зменшувати масштаб сторінки, використовуючи жести двома пальцями. Завдяки цій функціональності, користувач може адаптувати розмір вмісту до своїх потреб і вподобань, забезпечуючи комфортне читання навіть на пристроях з різними розмірами екрану. Цей підхід дозволяє забезпечити максимальну гнучкість і зручність взаємодії з програмою, забезпечуючи задоволення від використання та оптимальне використання ресурсів пристрою.

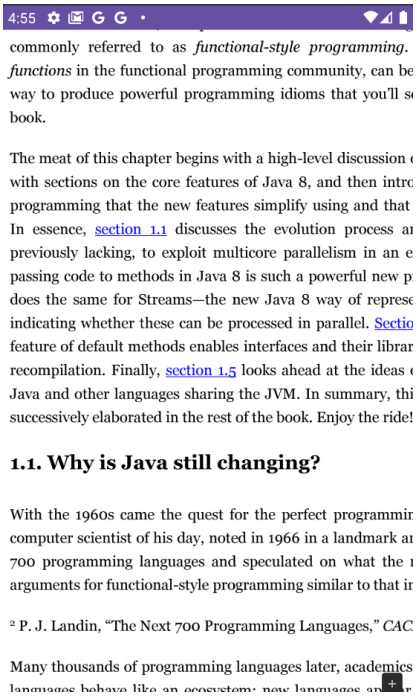


Рисунок 3.20 – Приклад приближення тексту на сторінці для PDF-файлу

Під час перегляду формату «карточки» відповідної книги, важливо враховувати додаткові елементи, що спрямовані на полегшення орієнтування користувача серед бібліотеки книг. Ми вважаємо за потрібне відображати частину метаданих, яка містить інформацію про книгу, щоб користувач міг швидше знайти та відкрити потрібний йому твір. Відповідно, у форматі «карточки» відображається інформація, яка допомагає ідентифікувати книгу, зокрема назва, автор та інші важливі атрибути (див. рис. 3.21). Це сприяє

більш ефективній організації бібліотеки книг та забезпечує користувачам зручнішу навігацію та доступ до необхідного контенту.

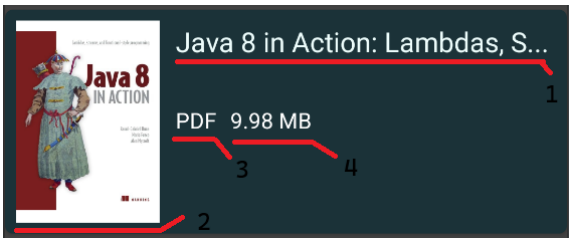
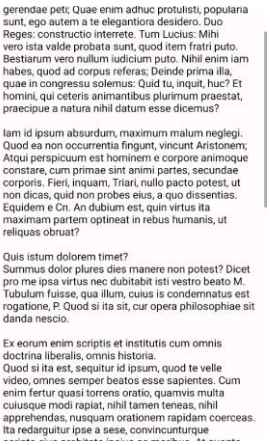


Рисунок 3.21 – Формат «карточки» книги з метаданими файлу:
1 – назва книги, 2- обкладинка, 3 – формат (розширення) файлу,
4– об’єм файлу на пристрої

Назва книги напряму знаходиться в метаданих PDF-файлу. Що стосується обкладинки, сам по собі PDF її не містить, тому ми для кращого орієнтування користувача по книгам використовуємо першу сторінку самого PDF. У випадку, коли книга не містить обкладинки першою сторінкою, даний функціонал виступає звичайним прев’ю книги.

Таку саму інформацію, якщо така є, ми побачимо в тому числі від усіх інших файлів, що ми вміємо опрацьовувати. Варто зазначити, що на відміну від PDF-файлів, наприклад, txt-файли не мають ніякого особливого візуалу, тому при відкритті текстового документу ми читаємо звичайним пролистуванням книги згори вниз. Виглядає це наступним чином (рис. 3.22).


gerendae peti; Quae enim adhuc protulisti, popularia sunt, ego autem a te elegantiora desidero. Duo Reges: constructio interrete. Tum Lucius: Mihi vero ista valde probata sunt, quod item fratri puto. Bestiarum vero nullum iudicium puto. Nihil enim iam habes, quod ad corpus referas; Deinde prima illa, quae in congressu solemus: Quid tu, inquit, huc? Et homini, qui ceteris animantibus plurimum praestat, praecipue a natura nihil datum esse dicemus?

Iam id ipsum absurdum, maximum malum neglegi. Quod ea non occurrentia fingunt, vincunt Aristonem; Atqui perspicuum est hominem e corpore animoque constare, cum primae sint animi partes, secundae corporis. Fieri, inquam, Triari, nullo pacto potest, ut non dicas, quid non probes eius, a quo dissentias. Equidem e Cn. An dubium est, quin virtus ita maximam partem optineat in rebus humanis, ut reliquas obruat?

Quis istum dolorem timet?
Summus dolor plures dies manere non potest? Dicet pro me ipsa virtus nec dubitabit isti vestro beato M. Tubulum fuisse, qua illum, cuius is condemnatus est rogatione, P. Quod si ita sit, cur opera philosophiae sit danda nescio.

Ex eorum enim scriptis et institutis cum omnis doctrina liberalis, omnis historia.
Quod si ita est, sequitur id ipsum, quod te velle video, omnes semper beatos esse sapientes. Cum enim fertur quasi torrens oratio, quamvis multa cuiusque modi rapiat, nihil tamen teneas, nihil apprehendas, nusquam orationem rapidam coerceas. Ita redarguitur ipse a sese, convincunturque

Рисунок 3.22 – Пролистування текстового документа

Полоска справа показує, наскільки далеко пролистнувся текст, в тому числі показуючи масштабом, скільки ще залишилось читати.

1.4 Тестування розробленого застосунку

Для проведення вичерпних тестів нашого застосунку ми маємо урахувати різноманітні сценарії та умови його використання. Важливо переконатися, що програма працює надійно та стабільно у різних умовах, а також що вона коректно обробляє різні вхідні дані та ситуації. У нашому випадку, ми плануємо провести тести з використанням файлів різних розширень, розмірів, та характеристик, щоб впевнитися, що застосунок працює ефективно та коректно у різних умовах.

По-перше, ми розглянемо сценарій, коли користувач намагається запустити застосунок без надання необхідних дозволів. Для цього ми спочатку видаляємо застосунок з віртуального пристрою, а потім встановлюємо його знову. Після цього ми спробуємо відкрити застосунок і проаналізуємо результати цієї спроби, спостерігаючи за виниклими помилками або повідомленнями про недостатні права доступу (рис. 3.23).

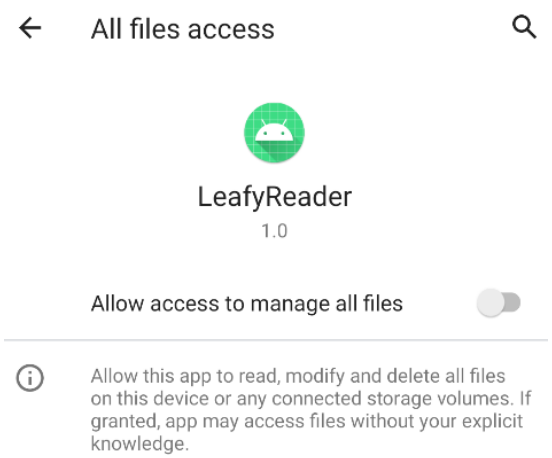


Рисунок 3.23 – Скріншот того, що ми вимкнули доступ до файлів для нашого застосунку

Тепер після надання доступу ми бачимо, що в застосунку ми отримали таку саму картинку, як в нас була до цього на головній сторінці (рис. 3.24).

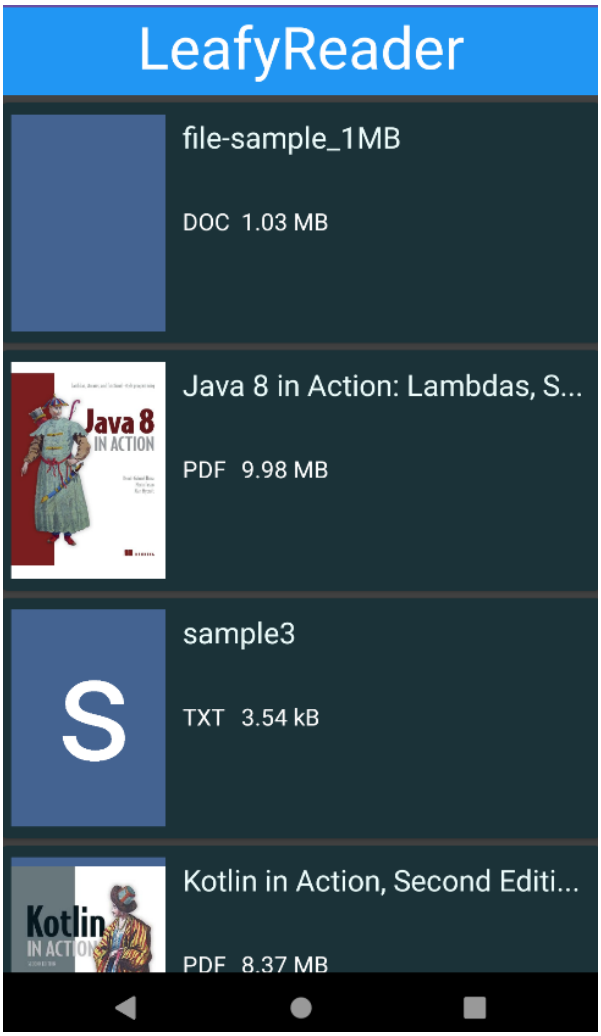


Рисунок 3.24 – Головна сторінка застосунку

Після цього варто спробувати відкрити різні файли. Оскільки DOC-файли поки що ми вміємо лише бачити, але не читати, то при натисканні на нього застосунок просто нічого не виконує. Тепер подивимось, що відбувається при роботі з PDF-файлами. Як було зазначено в інструкції користувача, обкладинкою книги виступає її перша сторінка. Це варто перевірити, і заодно чи можна узагалі читати книгу з нашого застосунку. Наша обкладинка має наступний вигляд (рис. 3.25).

Рисунок 3.25 – Обкладинка PDF-книги для тесту

Тепер відкриємо саму книгу та подивимось, чи справді перша сторінка є обкладинкою (рис. 3.26).

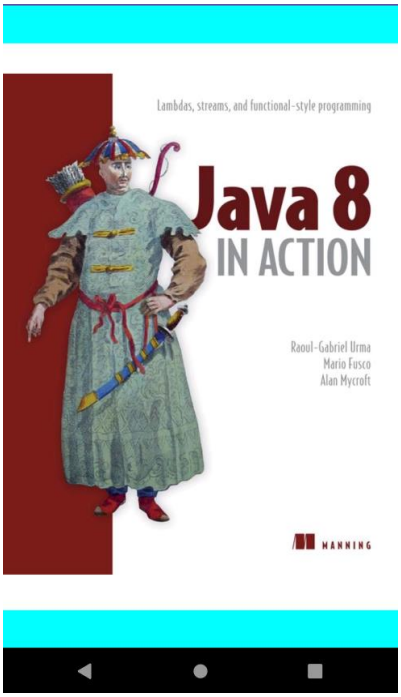
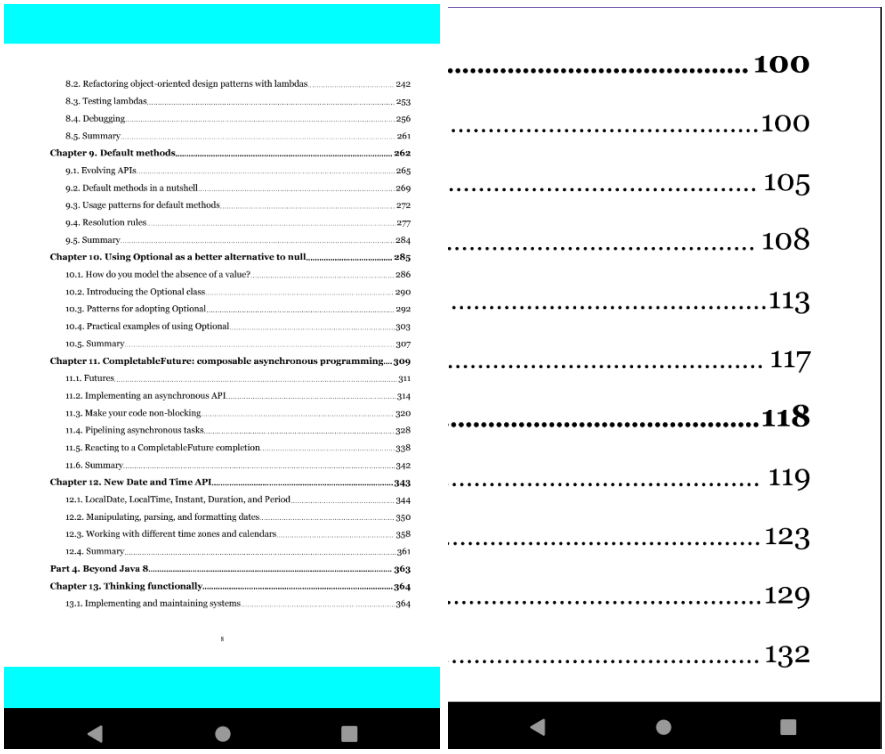


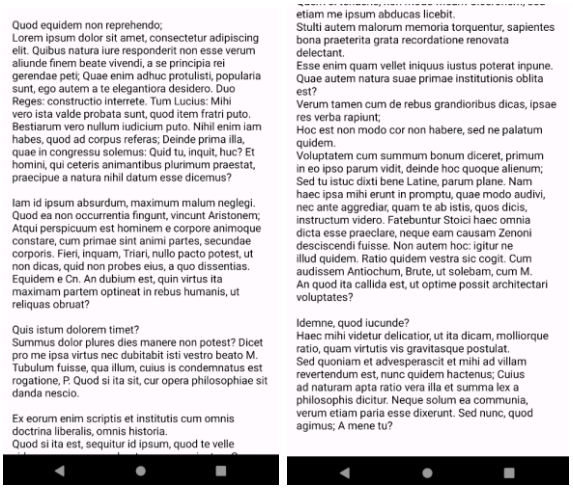
Рисунок 3.26 – Перша сторінка PDF-книги для тесту

Тепер спробуємо подивитись, що в сусідніх сторінках, та чи можна робити речі, зазначені в інструкції користувачу (рис. 3.27).



Рисунки 3.27– Відкриття сусідніх сторінок PDF-файлу та зміна масштабування

Тепер відкриємо текстовий документ та перевіримо його працездатність з пролистуванням (рис. 3.28).



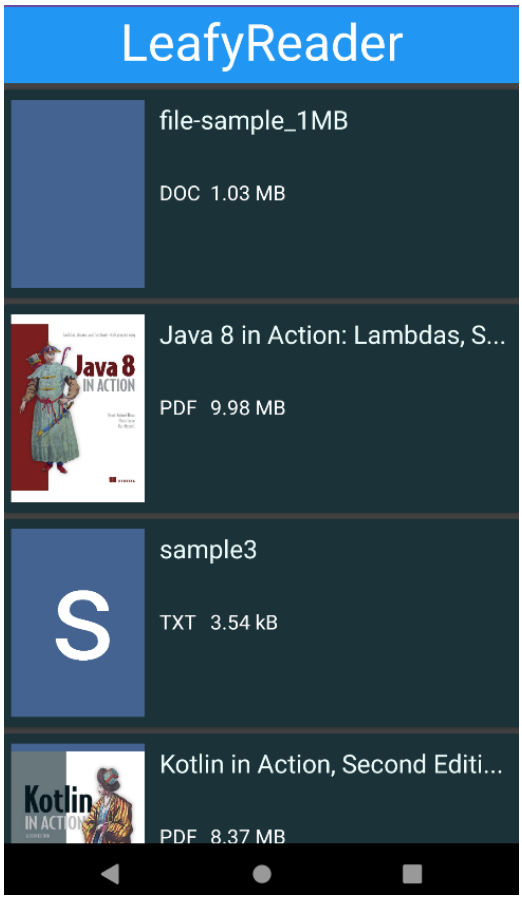
Рисунки 3.28 – Тестування текстового файлу в застосунку

Додатковим етапом нашого експерименту буде перевірка впливу переміщення файлів на продуктивність застосунку. Після переміщення

файлів ми будемо спостерігати за часом, який потрібно програмі для виявлення та завантаження їх відповідних даних. Це дозволить нам оцінити ефективність роботи застосунку у випадку, коли файли розташовані в різних директоріях на пристрої.

Крім того, ми можемо проаналізувати, чи вплине переміщення файлів на розмір самого застосунку або на його загальну продуктивність. Для цього ми проведемо порівняльний аналіз роботи застосунку до та після переміщення файлів, звертаючи увагу на час запуску програми, обсяг використаної оперативної пам'яті та використання процесорних ресурсів.

Цей комплексний підхід дозволить нам отримати більш детальне уявлення про вплив переміщення файлів на роботу нашого застосунку, що, у свою чергу, допоможе нам прийняти інформовані рішення щодо подальшого вдосконалення його функціоналу та продуктивності.



Рисунки 3.29 – Програма після переміщення книг до специфічних директорій

Однак, перед тим як зробити остаточний висновок щодо стабільності та продуктивності застосунку, ми повинні розглянути більш широкий спектр сценаріїв тестування. Це включає проведення навантажувальних тестів, тестів на межах, тестів відновлення, а також тестування сумісності з різними версіями операційної системи та пристроями.

Проведення навантажувальних тестів дозволить нам визначити, як застосунок працює під час інтенсивного використання, наприклад, коли багато користувачів одночасно використовують його або коли відбувається одночасний доступ до великої кількості файлів. Такі тести допоможуть нам виявити можливі межі продуктивності та визначити, чи необхідні додаткові оптимізації для оптимальної роботи застосунку в умовах високого навантаження.

Тести на межах допоможуть нам визначити, як застосунок поводиться в екстремальних умовах, наприклад, коли величина файлу перевищує максимально допустимий розмір або коли файлова система має складну структуру. Це дозволить нам ідентифікувати можливі проблеми з обробкою великих файлів або зі справжньою архітектурою файлової системи.

Тести відновлення будуть спрямовані на перевірку того, як застосунок реагує на непередбачені ситуації, такі як втрата з'єднання, некоректне закриття або аварійне завершення роботи. Вони допоможуть нам забезпечити, що застосунок правильно відновлюється після таких подій та зберігає консистентність даних.

Нарешті, тести сумісності будуть спрямовані на перевірку того, як застосунок працює на різних версіях операційної системи Android та на різних пристроях з різною апаратною конфігурацією. Це дозволить нам забезпечити, що застосунок працює стабільно та коректно на різних платформах та пристроях, що розширить його аудиторію користувачів та покращить загальний досвід використання.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений застосунок для читання багатьох форматів документів на Android-пристроях.

Завершуючи розробку нашого застосунку, можемо підвести підсумки та визначити основні досягнення, а також окреслити можливості для подальшого вдосконалення. На основі проведеної роботи та отриманих результатів, можна зробити кілька важливих висновків. Створення надійної системи аутентифікації та авторизації забезпечило високий рівень безпеки даних користувачів. Використання сучасних технологій дозволило запобігти більшості поширених загроз і забезпечити безпечний доступ до застосунку. Реалізовані механізми завантаження та обробки файлів працюють стабільно і швидко, що дозволяє користувачам легко завантажувати та зберігати різноманітні файли. Автоматична перевірка файлів на віруси та їх конвертація у потрібний формат значно спрощують роботу з великим обсягом даних. Інтерфейс користувача вийшов інтуїтивно зрозумілим і зручним, що сприяє позитивному користувацькому досвіду. Використання сучасних підходів до дизайну допомогло створити привабливий та функціональний інтерфейс.

Незважаючи на досягнуті результати, наш застосунок має потенціал для подальшого розвитку та вдосконалення. Розширення підтримуваних форматів файлів дозволить користувачам завантажувати та обробляти ще більше різноманітних даних. Наприклад, додавання підтримки форматів CAD, 3D моделей чи специфічних наукових даних розширить сферу застосування нашого продукту. Інтеграція з хмарними сервісами дозволить користувачам зберігати свої файли не тільки локально, але й у хмарі. Це забезпечить доступ до даних з будь-якого пристрою та підвищить надійність збереження інформації. Крім того, синхронізація в реальному часі сприятиме більш ефективній співпраці між користувачами. Оптимізація алгоритмів обробки даних і завантаження файлів може підвищити швидкодію

застосунку. Впровадження нових технологій та інструментів для аналізу продуктивності допоможе виявити й усунути вузькі місця в роботі системи. Додавання можливості інтеграції з іншими популярними сервісами та платформами, такими як CRM-системи, платформи для керування проєктами або аналітичні інструменти, значно розширить функціональні можливості нашого застосунку і зробить його більш привабливим для корпоративних користувачів. Регулярне оновлення системи безпеки та впровадження нових методів захисту даних допоможе забезпечити надійність і конфіденційність інформації, що зберігається в застосунку. Постійне збирання зворотного зв'язку від користувачів і впровадження їх пропозицій та зауважень допоможе зробити інтерфейс ще зручнішим і привабливішим.

Таким чином, наш проєкт уже досяг значного успіху, але має великий потенціал для подальшого розвитку. Впровадження зазначених покращень дозволить не тільки розширити функціональні можливості, але й підвищити задоволеність користувачів, що є ключовим фактором успіху будь-якого програмного продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кобилін О. А., & Творошенко І. С. (2021). Методи цифрової обробки зображень, с. 7-24.

2. Andrew Chupikov, Dmitry Kinoshenko, Vladimir Mashtalir, Konstantin Shcherbinin (2006). Image retrieval with segmentation-based query, с. 1-2.

3. Dmitry Kinoshenko, Vladimir Mashtalir, Alexander Orlov, Elena Yegorova (2003). Method of creating of functional invariants under one-parameter geometric image transformations, с.33-42.

4. Yevgeniy Bodyanskiy, Alina Shafronenko, Sergii Mashtalir (2020). Online robust fuzzy clustering of data with omissions using similarity measure of special type, с.224-228.

5. Особливості «permission handling» під конкретні версії Android. URL: https://developer.android.com/about/versions/13/behavior-changes-13 (дата звернення 12.04.2024).

6. Патерни проектування. URL: https://refactoring.guru/design-patterns (дата звернення 14.04.2024).

7. Bodyanskiy Ye, A Shafronenko, S Mashtalir (2020). Online robust fuzzy clustering of data with omissions using similarity measure of special type-Lecture Notes in Computational clutolligence and Decision Waking-Cham, с.321-328.

8. Бібліотека для зчитування метаданих з файлу PdfiumCore. URL: https://github.com/barteksc/PdfiumAndroid/blob/master/src/main/java/com/shockwave/pdfium/PdfiumCore.java (дата звернення 17.04.2024).

9. Valentin Lyubchenko, Rami Matarneh, Oleg Kobylin, Vyacheslav Lyashenko (2016). Digital image processing techniques for detection and diagnosis of fish diseases, с. 73-82.

10. Permissions on Android. URL: https://developer.android.com/guide/topics/permissions/overview (дата звернення 18.04.2024).

11. Vyacheslav Lyashenko, Oleg Kobylin, Yevhenii Baranchykov (2018). Ideology of Image Processing in Infocommunication Systems, с. 31-35.

12. Vinicius M Alves, Eugene N Muratov, Stephen J Capuzzi, Regina Politi, Yen Low, Rodolpho C Braga, Alexey V Zakharov, Alexander Sedykh, Elena Mokshyna, Sherif Farag, Carolina H Andrade, Victor E Kuz'Min, Denis Fourches, Alexander Tropsha (2016). Alarms about structural alerts, с. 24-28.

13. Android Runtime Permissions. URL: https://medium.com/@ominoblair/android-runtime-permissions-91b42d2fa0a3 (дата звернення 19.04.2024).

14. В.О. Гороховатський, І.С. Творошенко (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник. –Харків: ХНУРЕ, с. 7-15.

15. Functional Programming Paradigm. URL: https://www.geeksforgeeks.org/functional-programming-paradigm/ (дата звернення 20.04.2024).

16. Object-oriented programming. URL: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP (дата звернення 21.04.2024).

17. ВО Гороховатський, ІС Творошенко (2022). Аналіз багатовимірних даних за описом у формі множини компонент, с.38-46.

18. O. Kobylin, V. Gorokhovatskyi, I. Tvoroshenko, O. Peredrii (2020). The Application of Non-Parametric Statistics Methods in Image Classifiers Based on Structural Description Components, Telecommunications and Radio Engineering, 79 (10), pp. 855-863.

19. В.О. Гороховатський, С.В. Гадецька (2020). Статистичне оброблення та аналіз даних у структурних методах класифікації зображень (монографія), Харків, ФОП Панов А.Н., 128 с.

20. Kotlin Android Extensions. URL: https://antonioleiva.com/kotlin-android-extensions/ (дата звернення 23.04.2024).

21. Building with Gradle. URL: https://www.baeldung.com/gradle-fat-jar (дата звернення 24.04.2024).

22. Configure your build. URL: https://developer.android.com/build (дата звернення 25.04.2024)

23. Dmitry Kinoshenko, Sergey Mashtalir, Andreas Stephan, Vladimir Vinarski (1993). Neural Network Segmentation Of Video Via Time Series Analysis, INFORMATION THEORIES & APPLICATIONS, pp. 232.

24. Vyacheslav Lyashenko, Oleg Kobylin, M Ayaz Ahmad (2014). General methodology for implementation of image normalization procedure using its wavelet transform, IJSR, pp. 2877.

25. Iryna Tvoroshenko, Volodymyr Gorokhovatskyi, Oleg Kobylin, Arsenii Tvoroshenko (2023). Application of deep learning methods for recognizing and classifying culinary dishes in images, International Journal of Academic and Applied Research, 70 c.

26. Android Logging in Kotlin – the right way. URL: https://muthuraj57.medium.com/logging-in-kotlin-the-right-way-d7a357bb0343 (дата звернення 28.04.2024).

27. Android Custom intents. URL: https://developer.android.com/guide/app-actions/custom-intents (дата звернення 29.04.2024).

28. Boosting User Engagement. URL: https://medium.com/mset/boosting-user-engagement-improved-ranking-with-sharesheet-custom-actions-android-14-e97f79407f5b (дата звернення 30.04.2024).

29. Kotlin Inheritance. URL: https://kotlinlang.org/docs/inheritance.html (дата звернення 01.05.2024).

30. Vyacheslav Lyashenko, Oleg Kobylin, Oleksandr Selevko (2020). Wavelet analysis and contrast modification in the study of cell structures images, WARSE, 4706 c.