

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Мобільний застосунок для книжкового обліку»

Виконав студент групи ПЗп-21-3
ЗАДНІПРЯНЕЦЬ Артур Артурович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«12» травня 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗАДНІПРЯНЕЦЬ Артур Артурович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Мобільний застосунок для книжкового обліку»**,
затверджена Наказом закладу вищої освіти від **«12» травня 2026 р., №261/с**

Вихідні дані для роботи: управління процесами книжкового обліку та організації книжкової полиці із використанням мобільних технологій для покращення досвіду читання.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю. М.	12.05.2026р.	12.05.2026р.
2	Єфремов Ю. М.	12.05.2026р.	12.05.2026р.
3	Єфремов Ю. М.	12.05.2026р.	12.05.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з пояснювальної записки обсягом 66 сторінок, що містить 26 рисунків, 11 таблиць, 30 використаних джерел, 5 додатків та мобільного додатку для книжкового обліку.

Об'єктом дослідження у цій роботі виступають процеси автоматизації обліку персональних книжкових колекцій та систематизації метаданих літературних видань у цифровому середовищі. Предметом дослідження є сучасні архітектурні підходи, програмні інструменти розробки мобільних систем під управлінням операційної системи Android та методи інтеграції прикладних рішень із глобальними бібліографічними сервісами.

Метою виконання кваліфікаційної роботи є проектування та програмна імплементація мобільного застосунку «LiteraList», який забезпечує користувачу зручний інструментарій для управління домашньою бібліотекою, включаючи автоматизацію заповнення даних та моніторинг читацької активності.

Основним результатом роботи є функціональний мобільний додаток, який дозволяє здійснювати керування книжковим каталогом, редагувати інформацію про видання, керувати графічними обкладинками та візуалізувати прогрес читання.

Практична цінність отриманих результатів полягає у створенні готового до використання програмного продукту, який вирішує проблему ефективної організації персонального інтелектуального простору та мінімізує ручну працю під час каталогізації великих обсягів літератури.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, KOTLIN, КНИЖКОВИЙ ОБЛІК

					ІПЗ.КР.Б – 121 – 26 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Задніпрянець А. А.			Мобільний застосунок для книжкового обліку Пояснювальна записка	Літ.	Арк.
Керівник		Єфремов Ю. М.					3
						Аркушів	66
Рецензент		Власенко О. В.				Житомирська політехніка, група ІПЗп-21-3	
Зав. каф.		Вакалюк Т. А.					

ABSTRACT

The bachelor's thesis consists of a 66-page explanatory note containing 26 figures, 11 tables, 30 references, and 5 appendices, as well as a mobile app for bookkeeping.

The object of the study is the processes of automating the management of personal book collections and the systematization of metadata for literary publications within a digital environment. The subject of the research encompasses modern architectural approaches, software development tools for mobile systems running on the Android operating system, and methods for integrating application solutions with global bibliographic services.

The goal of this qualification work is the design and software implementation of the "LiteraList" mobile application, which provides users with a convenient toolkit for home library management, including automated data entry and reading activity monitoring.

The main result of the work is a functional mobile application that enables book catalog management, editing of publication details, management of cover art, and visualization of reading progress.

The practical value of the results lies in the creation of a ready-to-use software product that addresses the challenge of efficiently organizing a personal intellectual space and minimizes manual labor when cataloging large volumes of literature.

KEYWORDS: MOBILE APPLICATION, ANDROID, KOTLIN, BOOK TRACKING.

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ, АРХІТЕКТУРИ ТА ФУНКЦІОНАЛЬНИХ ВИМОГ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури мобільного додатку.....	16
1.4 Обґрунтування вибору інструментальних засобів	22
Висновки до розділу 1	25
РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ КНИЖКОВОГО ОБЛІКУ	26
2.1 Визначення варіантів використання	26
2.2 Проєктування шару даних.....	30
2.3 Проєктування шару бізнес-логіки	39
2.4 Проєктування шару представлення.....	45
Висновки до розділу 2	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МОБІЛЬНИМ ЗАСТОСУНКОМ ДЛЯ КНИЖКОВОГО ОБЛІКУ	50
3.1 Порядок встановлення та налаштування параметрів системи	50
3.2 Структура інтерфейсу мобільного додатку	53
3.3 Тестування роботи мобільного додатку	57
Висновки до розділу 3	61
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64
ДОДАТКИ.....	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – користувацький інтерфейс

UX – досвід користувача

API – прикладний програмний інтерфейс

IDE – інтегроване середовище розробки

ISBN – міжнародний стандартний номер книги

ПЗ – програмне забезпечення

БД – база даних

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми полягає у сучасному інформаційному суспільстві спостерігається стрімка цифровізація всіх сфер життя, включаючи культуру споживання контенту та організацію особистого інтелектуального простору. Проте значна частина користувачів все ще надають перевагу паперовим варіантам літератури. Про це свідчать результати всеукраїнського соціологічного дослідження рівня читання дітей та дорослих у 2024 році «Зміцнення позиції української книги», яке провела дослідницька агенція Info Sapiens на замовлення Українського інституту книги у вересні 2024 року [1]. Проаналізувавши практику читання чи слухання книжок на дозвіллі (зауважимо, що при цьому виключається читання довідників, підручників, книг рецептів), найчастішою залишається практика читання саме друкованих книжок. 10% респондентів щоденно читають такі книжки, 12% – кілька разів на тиждень і 14% – кілька разів на місяць. Читання електронних книг залишається менш уживаною практикою серед респондентів – 73% вказали, що ніколи не читають таких книг. Подібна ситуація з аудіокнигами – 72% респондентів ніколи не слухали їх.

Проблема ефективного управління домашніми бібліотеками стає дедалі гострішою через постійне зростання обсягів друкованої продукції та потребу у миттєвому доступі до метаданих видань. Традиційні методи обліку книг, такі як паперові каталоги або прості текстові таблиці, вже не забезпечують необхідного рівня мобільності, швидкості пошуку та автоматизації, що зумовлює актуальність розробки спеціалізованого програмного забезпечення для мобільних платформ.

Мета випускної кваліфікаційної роботи бакалавра полягає в комплексному інженерному проектуванні, розробці, тестуванні та впровадженні автономного мобільного застосунку для автоматизованого обліку, каталогізації та інтерактивного супроводу домашньої бібліотеки.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

Завдання роботи включають аналіз предметної області та аналогів мобільних систем обліку літератури для формування технічних вимог до проєкту, обґрунтування вибору архітектури та стеку розробки, проєктування локальної бази даних для забезпечення офлайн-режиму, інтеграція мобільного застосунку із зовнішнім бібліографічним сервісом для автоматичного заповнення інформації про книги за номером ISBN, розробку якісного та зручного інтерфейсу користувача, реалізацію алгоритмів моніторингу читацької діяльності та керування графічними обкладинками у файловій системі пристрою, а також проведення підсумкового тестування створеного прототипу на відповідність усім функціональним вимогам.

Об'єктом дослідження виступають процеси автоматизації управління, каталогізації та інтерактивного супроводу персональних і домашніх книжкових колекцій, що охоплюють життєвий цикл взаємодії користувача з друкованими виданнями від моменту їх отримання до систематичного аналізу прогресу читання.

Предметом дослідження є архітектурні підходи, програмні технології, алгоритми локального збереження даних та методи мережевої інтеграції із зовнішніми бібліографічними сервісами, що застосовуються для проєктування та розробки автономного мобільного застосунку на платформі Android, здатного забезпечити ефективний книжковий облік і відстеження читання.

Практичне значення розробки мобільного застосунку полягає у створенні готового до впровадження програмного продукту, який безпосередньо вирішує завдання автоматизації обліку, систематизації та інтерактивного супроводу домашніх книжкових колекцій. Створене інженерне рішення може бути успішно використане індивідуальними користувачами для оптимізації управління власним інтелектуальним простором, а також приватними книжковими клубами чи невеликими локальними бібліотеками для ведення оперативного каталожного обліку без залучення високовартісних стаціонарних систем.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ, АРХІТЕКТУРИ ТА ФУНКЦІОНАЛЬНИХ ВИМОГ

1.1 Постановка задачі

Задача випускної кваліфікаційної роботи бакалавра полягає в комплексному інженерному проектуванні, розробці, тестуванні та впровадженні автономного мобільного застосунку для автоматизованого обліку, каталогізації та інтерактивного супроводу домашньої бібліотеки.

На аналітичному етапі необхідно виконати системний аналіз предметної області, дослідити актуальні потреби користувачів у сфері систематизації літературних фондів та сформулювати вичерпний перелік функціональних і нефункціональних вимог до системи. Також цей етап передбачає проведення порівняльного аналізу існуючих ринкових аналогів та конкурентних програмних рішень для визначення оптимальних шляхів підвищення якості користувацького досвіду та автоматизації рутинних операцій із введення даних.

На етапі архітектурного проектування задача вимагає побудови надійної структури мобільного програмного продукту на основі сучасних патернів проектування та підходів, що забезпечують ізоляцію джерела даних, бізнес-логіки та компонентів інтерфейсу користувача завдяки використанню абстрактних шарів.

У межах розробки інформаційного забезпечення необхідно спроектувати оптимальну структуру локальної бази даних для постійного, безпечного та автономного збереження відомостей про книжкові колекції безпосередньо на пристрої користувача. Такий підхід має гарантувати безперервний доступ до персональної бібліотеки та можливість перегляду чи редагування інформації навіть за умов повної відсутності активного підключення до мережі інтернет.

Практична реалізація системи передбачає створення програмного ядра для виконання базових операцій додавання, перегляду, редагування та

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

видалення бібліографічних записів. Програма повинна оперувати набором характеристик кожної книги, включаючи її назву, автора, рік видання, унікальний міжнародний номер, загальну кількість сторінок, персональну оцінку та рецензію.

Для розширення стандартного функціоналу каталогу задача включає створення інтерактивних інструментів моніторингу читацької діяльності шляхом обліку прочитаних сторінок та динамічної візуалізації прогресу читання.

Важливою частиною інженерного завдання є розробка надійних алгоритмів керування графічними обкладинками, що передбачає вибір фотографій із пам'яті пристрою, їх копіювання до ізольованої внутрішньої файлової системи застосунку та реалізацію механізму автоматичного знищення застарілих або замінених файлів для забезпечення оптимального використання апаратних ресурсів мобільного пристрою.

З метою покращення досвіду користувача та прискорення процесу наповнення каталогу задача вимагає налаштування взаємодії мобільного застосунку із зовнішньою мережевою інфраструктурою глобального бібліографічного сервісу. Програмний модуль інтеграції повинен забезпечувати автоматичний пошук, розпізнавання та заповнення відповідних інформаційних полів книги за її міжнародним стандартним номером.

Для забезпечення стійкості мережевих запитів та дотримання вимог кібербезпеки необхідно реалізувати механізм ізольованого збереження конфіденційних ключів доступу до зовнішніх сервісів поза основним репозиторієм вихідного коду.

Завершальний етап передбачає процес розгортання та проведення комплексного мануального тестування розробленого програмного продукту за визначеними варіантами використання, оцінку стабільності функціонування всіх його компонентів та документальне оформлення результатів розробки відповідно до встановлених стандартів якості.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

1.2 Аналіз аналогів програмного продукту

Для розробки якісного програмного продукту потрібно проаналізувати популярних конкурентів на ринку. При цьому береться до уваги користувацький інтерфейс систем, їх основний функціонал, переваги та недоліки, а також їх порівняння між собою та власним програмним продуктом.

Платформа Goodreads (рис. 1.1) є однією з найбільших і найвідоміших соціальних мереж для книголюбів у світі, яка поєднує в собі функції масштабного бібліографічного каталогу та інструментів для інтерактивної взаємодії читачів [2]. Створений як глобальний віртуальний простір для систематизації літератури, цей сервіс дозволяє користувачам керувати власними цифровими полицями, ділитися читацьким досвідом та шукати нові твори на основі персоналізованих рекомендацій і колективного розуму багатомільйонної аудиторії.

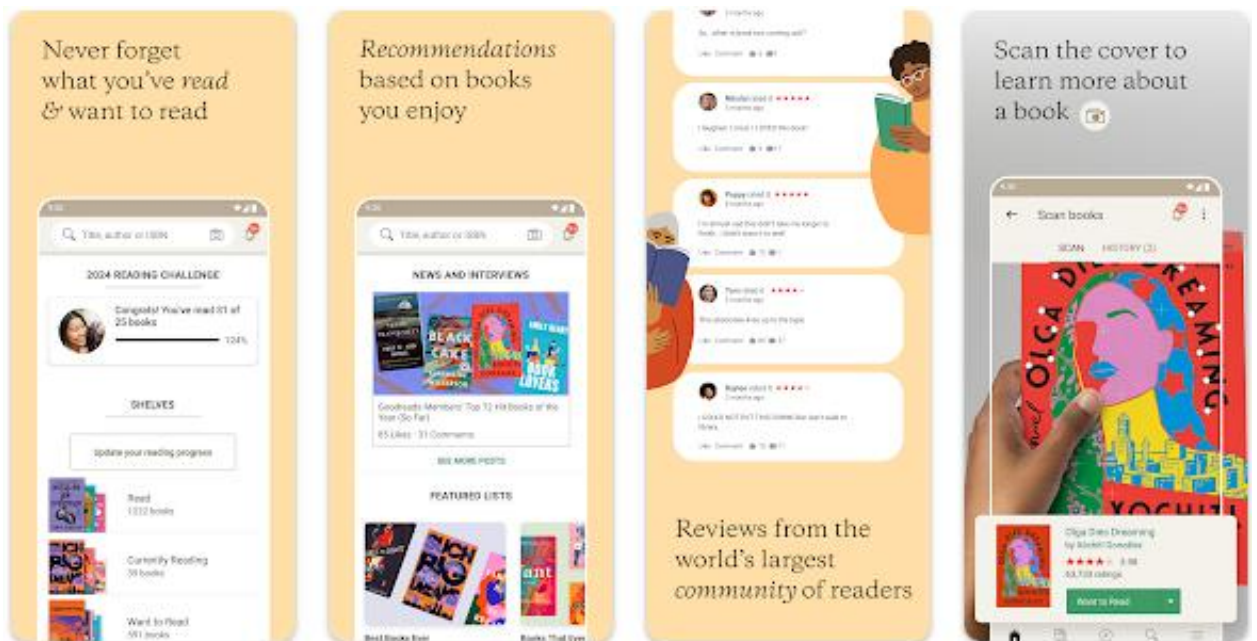


Рис. 1.1 - Мобільний додаток Goodreads

Основне ядро функціонала Goodreads зосереджене навколо концепції віртуальних книжкових полиць, де кожен користувач може розподіляти літературу за категоріями прочитаного, поточного читання та планів на майбутнє, а також створювати власні специфічні добірки. Автоматизація наповнення профілю забезпечується за допомогою розширеної пошукової

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

системи, яка підтримує розпізнавання книг за назвою, автором або міжнародним стандартним номером ISBN, а мобільна версія додатку додатково містить інструмент для сканування штрих-кодів безпосередньо з паперових обкладинок. Соціальна складова платформи дозволяє користувачам публікувати детальні текстові рецензії, виставляти оцінки за п'ятибальною шкалою, коментувати відгуки друзів, об'єднуватися в тематичні книжкові клуби та брати участь у щорічних інтерактивних марафонах читання, самостійно встановлюючи планку кількості книг на рік. Окреме місце займає рекомендаційна система, яка аналізує історію оцінок читача і пропонує нові релевантні видання, а також розвинений інструментарій для авторів, які можуть вести офіційні блоги, влаштовувати безкоштовні розіграші своїх друкованих примірників та напряду спілкуватися з аудиторією.

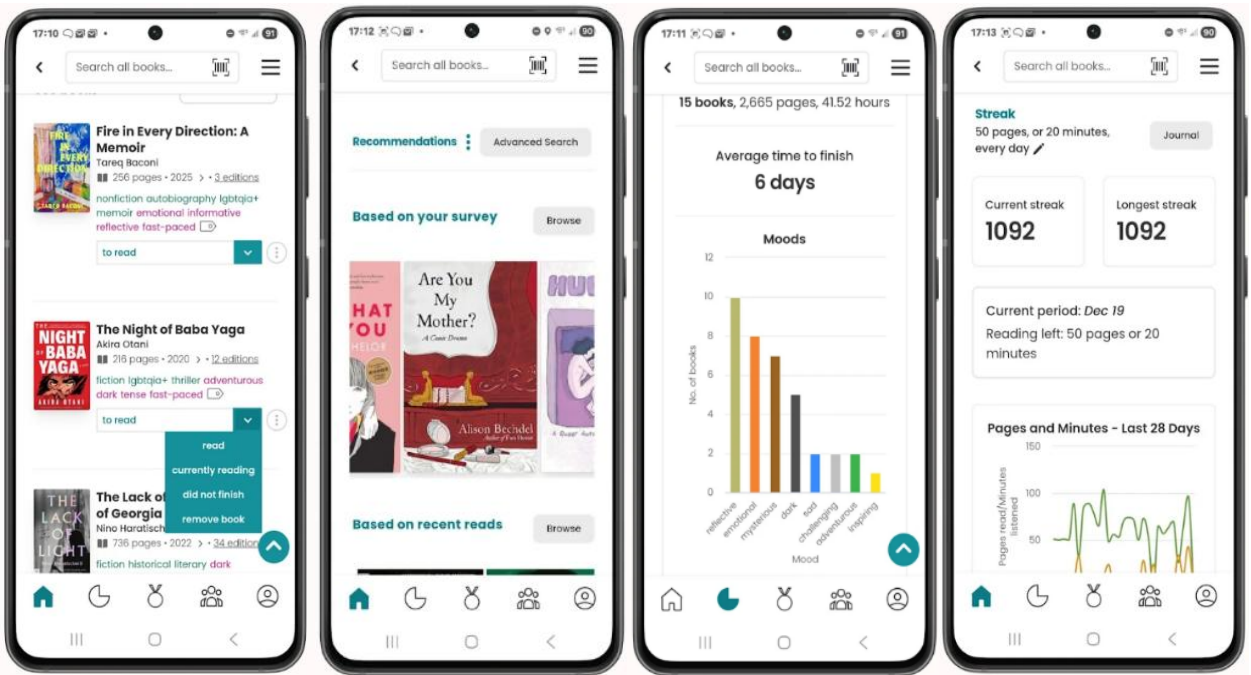


Рис. 1.2 – Мобільний додаток StoryGraph

Платформа StoryGraph (рис. 1.2) є сучасним цифровим інструментом, створеним спеціально для детального обліку, каталогізації та глибокого аналізу читацької діяльності [3]. Цей веб сервіс та однойменний мобільний застосунок пропонують користувачам принципово новий підхід до взаємодії з власною бібліотекою, фокусуючись не просто на фіксації списку прочитаних книг, а на дослідженні унікальних читацьких звичок і вподобань за допомогою інтелектуальних аналітичних алгоритмів.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Основне ядро функціоналу платформи побудоване навколо гнучкої системи ведення особистого профілю читача. Користувачі мають можливість додавати літературні твори до персональних списків, відстежувати поточний прогрес читання у сторінках або відсотках, фіксувати точні дати початку та завершення роботи над книгою. Унікальною особливістю сервісу є детальна система тегів та класифікації книг за емоційним забарвленням, темпом оповіді, типом персонажів та деталізацією сюжету. Замість стандартних текстових відгуків сервіс пропонує користувачам пройти структуроване опитування після прочитання, результати якого формують унікальну матрицю характеристик твору. На основі цих даних функціонує потужна система персональних рекомендацій, яка підбирає нові книги не лише за жанрами чи авторами, а й за специфічним настроєм чи бажаною динамікою сюжету. Крім того, платформа містить розвинений аналітичний блок, який у реальному часі візуалізує статистику читання за допомогою різноманітних діаграм, відображаючи розподіл книг за настроєм, жанрами, обсягом, форматом та оцінками користувача. Також у застосунку реалізовано інструменти соціальної взаємодії у форматі спільних читань, створення користувацьких книжкових викликів та відслідковування щоденного читання, що стимулює регулярність.

Мобільний застосунок Bookly (рис. 1.3) є одним із провідних комерційних рішень на ринку програмного забезпечення для любителів читання, оскільки він позиціонується не просто як статичний каталог літератури, а як інтерактивний асистент і трекер читацької діяльності [4]. Основний фокус цього продукту спрямований на мотивацію користувача, керування часом та детальну аналітику особистого прогресу. Центральною функцією Bookly є інтерактивний таймер читання, який користувач запускає під час безпосереднього опрацювання книги. Цей інструмент дозволяє фіксувати точний час, витрачений на кожну читацьку сесію, а по її завершенні система пропонує ввести номер останньої прочитаної сторінки, що автоматично вираховує швидкість читання у сторінках за годину. На основі

		Задніпрянець А. А.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

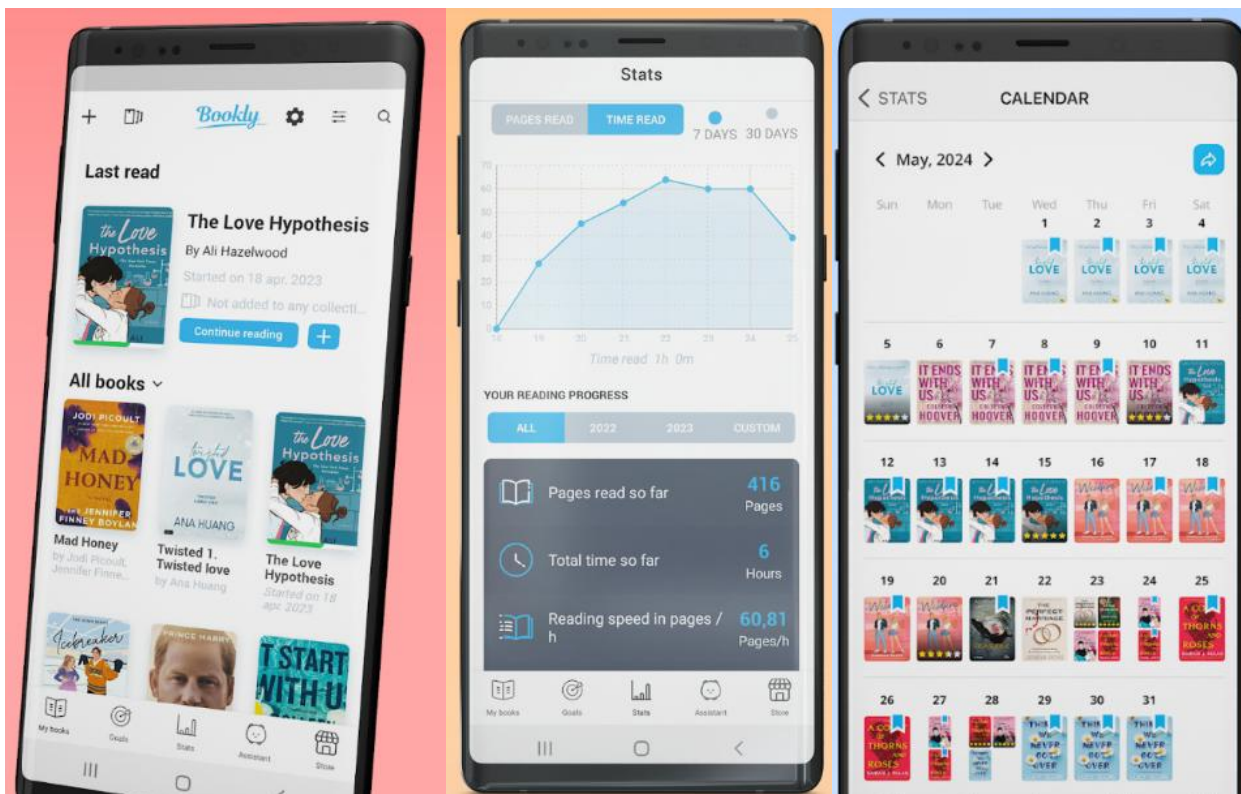


Рис. 1.3 – Мобільний додаток Bookly

цих даних застосунок будує прогнози, повідомляючи, скільки приблизно часу знадобиться для повного завершення поточної книги або розділу.

Для наповнення бази даних у Bookly реалізовано кілька альтернативних способів. Користувач може додавати книги вручну, шукати їх у глобальній онлайнівій базі даних за назвою чи автором, а також використовувати модуль сканування штрих-кодів ISBN за допомогою камери мобільного пристрою, що забезпечує миттєве завантаження бібліографічних метаданих та офіційних обкладинок. Кожна картка книги містить розширений набір атрибутів, включаючи назву, автора, загальну кількість сторінок, поточний статус читання та категорію.

Інтерактивна складова представлена інструментами для фіксації думок безпосередньо в процесі читання. Застосунок дозволяє зберігати текстові цитати, створювати персональні нотатки, виставляти книгам рейтинги та писати розгорнуті рецензії. Окремим функціональним блоком є глибока система статистики та досягнень. Програма генерує щоденні, щотижневі та щомісячні звіти, які візуалізують загальний час читання, кількість освоєних сторінок, тривалість найдовшої серії регулярного читання та динаміку

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

прогресу у вигляді інфографіки. Інтерактивні елементи включають систему рівнів, досягнень, віртуальних нагород та можливість встановлювати персональні цілі, наприклад, прочитати певну кількість книг за рік або присвячувати читанню не менше тридцяти хвилин щодня. Крім того, у систему інтегровано звукові асистенти, такі як генератори білого шуму чи мелодій, що покликані допомогти користувачеві краще сконцентруватися на тексті.

Таблиця 1.1

Таблиця порівняння книжкових додатків

Характеристика	Goodreads	StoryGraph	Bookly	Власний додаток
Тип платформи	Соціальна мережа	Платформа для аналізу читацьких звичок	Таймер читання та відстеження звичок	Додаток для обліку та відстеження читання
Автономність	Робота без доступу до мережі інтернет є неможливою	Робота без доступу до мережі інтернет є неможливою	Хмарна синхронізація з кешуванням даних.	Повністю автономний з можливістю отримувати дані з інтернету
Відстеження читання	Річний звіт про кількість прочитаних книжок	Візуальна статистика та графіки	Детальні звіти за днями	Відстеження кількості прочитаних сторінок
Інтерфейс користувача	Застарілий, перевантажений	Сучасний, мінімалістичний	Яскравий, сучасний	Сучасний, мінімалістичний

Завдяки зіставленню популярних аналогів (табл. 1.1) було визначено основні характеристики, які користувачі очікують від книжкового додатка. Розроблюване ПЗ не повинно вимагати постійного підключення до інтернету, дозволяє відстежувати базову інформацію про читання та має якісний, сучасний та неперевантажений інтерфейс користувача.

1.3 Вибір архітектури мобільного додатку

Серед поширених патернів проектування архітектури мобільного додатка можна виділити такі шаблони як MVC, MVP, MVVM, MVI, Clean Architecture.

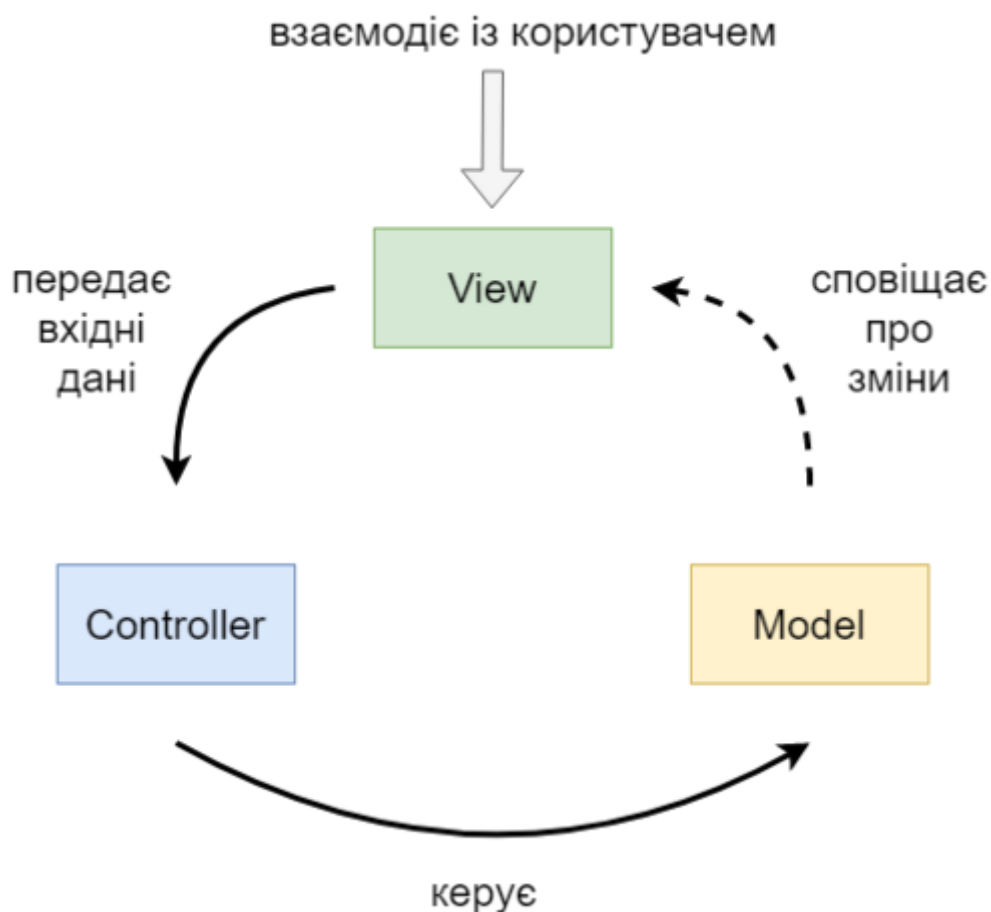


Рис. 1.4 – Схема патерна MVC

Архітектурний патерн MVC (Model-View-Controller) (рис. 1.4) є одним із найбільш фундаментальних підходів до проектування програмного забезпечення, який базується на принципі розділення відповідальності між трьома основними компонентами [5]. Першим компонентом є Модель (Model), яка виконує роль центрального елемента системи, оскільки вона відповідає за керування даними, бізнес-логіку та правила обробки інформації. Модель не має прямого зв'язку з візуальним відображенням і лише надає дані іншим частинам системи за запитом або сповіщає про їх зміну. Другий компонент під назвою Представлення (View) призначений виключно для візуалізації інформації, отриманої від моделі, та побудови графічного інтерфейсу

користувача. Його головна задача полягає в тому, щоб перетворити дані у зручний для сприйняття формат, наприклад, у вигляді кнопок, текстових полів чи списків. Третім компонентом виступає Контролер (Controller), який бере на себе роль посередника та координатора взаємодії між користувачем і системою. Він перехоплює вхідні сигнали, такі як натискання клавіш чи жести, інтерпретує їх та ініціює відповідні зміни в моделі або оновлює представлення.

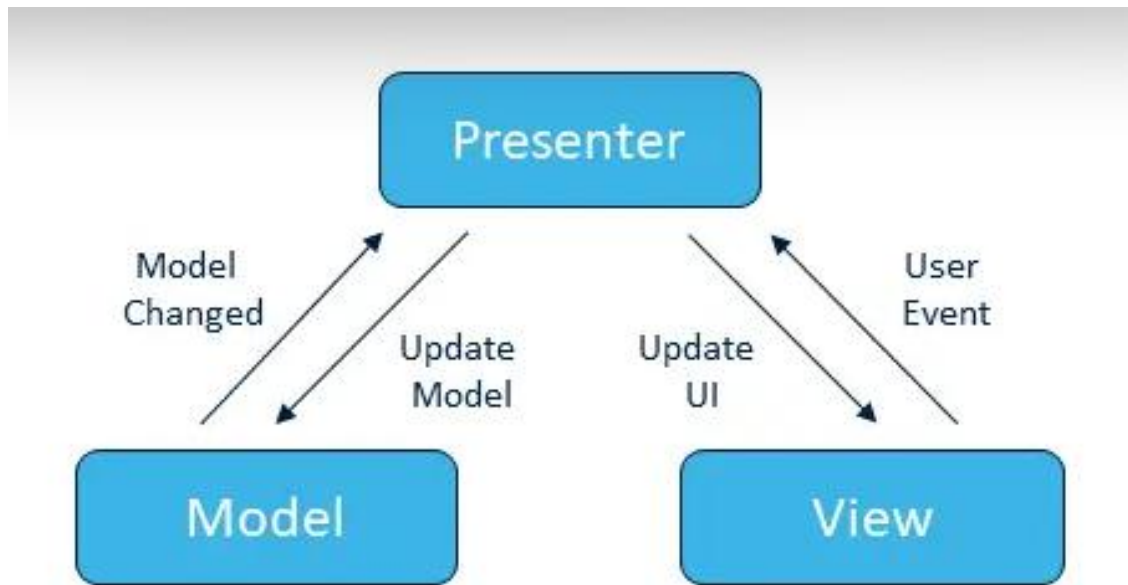
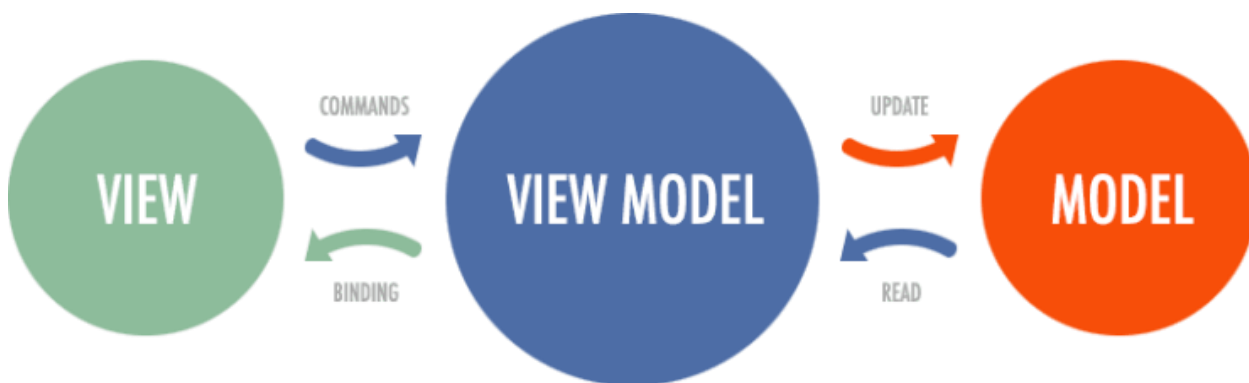


Рис. 1.5 – Схема патерна MVP

Архітектурний патерн MVP (Model-View-Presenter) (рис. 1.5) виник як еволюційний розвиток класичної моделі MVC з метою подолання її недоліків у контексті розробки графічних інтерфейсів, зокрема для мобільних платформ [6]. Основна ідея цього підходу полягає у винесенні всієї логіки взаємодії з даними та управління станом інтерфейсу в окремий компонент, що дозволяє зробити представлення максимально пасивним та незалежним.

Компонент Модель (Model) у цій структурі продовжує відповідати за роботу з джерелами даних, такими як локальні бази даних або мережеві запити, та виконання бізнес-правил застосунку. Представлення (View) у патерні MVP стає гранично простим рівнем, який лише відображає дані та передає події від користувача до презентера, не маючи при цьому жодних знань про логіку обробки цих подій. Ключовою фігурою стає Презентер

(Presenter), який виступає в ролі інтелектуального посередника. Він отримує дані від моделі, перетворює їх у формат, придатний для відображення, та надає



команди представленню через чітко визначений інтерфейс.

Рис. 1.6 – Схема патерна MVVM

Архітектурний патерн MVVM (рис. 1.6), назва якого є аббревіатурою від Model-View-ViewModel, на сьогодні є сучасним стандартом та рекомендованим підходом від компанії Google для розробки мобільних застосунків на платформі Android [7]. Основна ідея цієї структури полягає у забезпеченні чіткого розділення відповідальності між різними компонентами системи, що значно полегшує тестування, підтримку та подальше оновлення програмного коду.

У цій архітектурі рівень Model відповідає за управління даними та бізнес-логіку, часто включаючи роботу з локальними базами даних, такими як Room, або взаємодію з мережевими сервісами. Рівень View представляє графічний інтерфейс користувача, який у сучасних проектах зазвичай створюється за допомогою декларативного фреймворку Jetpack Compose та стандартів Material Design 3. Цей компонент відповідає виключно за візуалізацію станів та збір дій користувача, не містячи складної логіки обробки даних.

Ключовою ланкою в цій структурі є ViewModel, яка виступає інтелектуальним посередником між даними та інтерфейсом. Вона отримує необхідну інформацію з репозиторію, перетворює її у зручний для відображення стан і надає View доступ до цих даних. Важливою особливістю є те, що ViewModel не має прямого посилання на компоненти інтерфейсу, що

робить логіку обробки даних повністю ізольованою від візуальної частини. Такий підхід дозволяє уникнути перевантаження коду та забезпечує стабільність роботи застосунку навіть при зміні конфігурації пристрою або оновленні елементів дизайну.

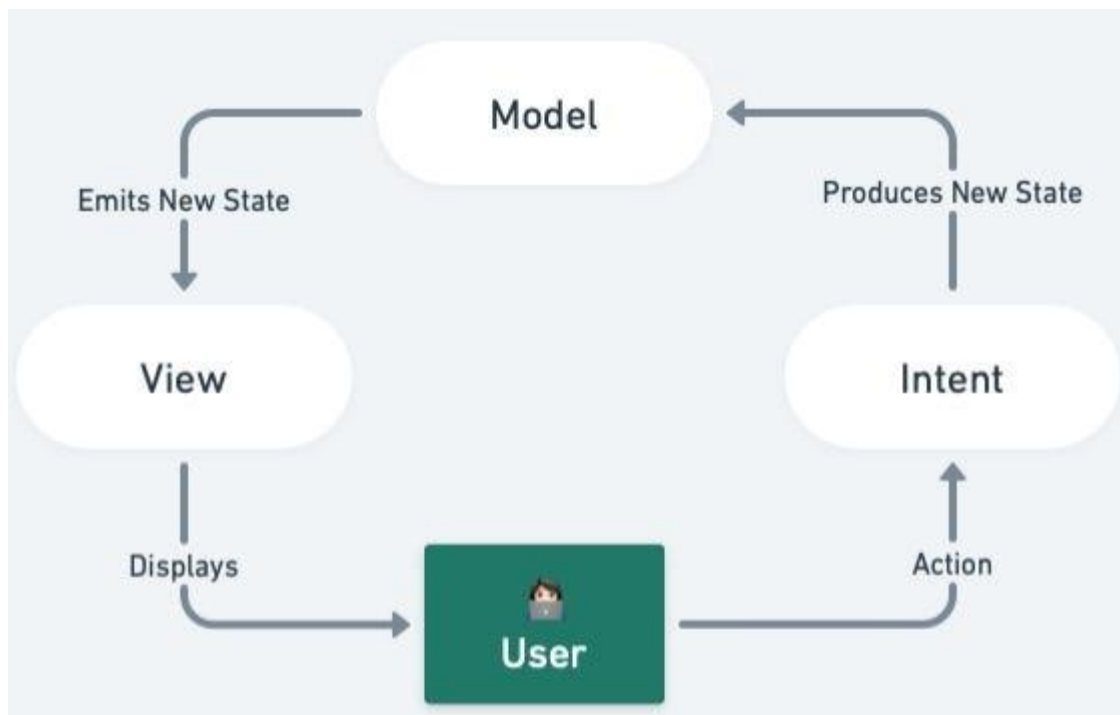


Рис. 1.7 – Схема патерна MVI

Архітектурний патерн MVI (рис. 1.7), що розшифровується як Model-View-Intent, є одним із найсучасніших підходів до проектування мобільних застосунків, який базується на принципах реактивного програмування та односпрямованого потоку даних [8]. На відміну від традиційних архітектур, де стан екрана може змінюватися з різних джерел, MVI пропонує суворий циклічний процес взаємодії компонентів, що робить поведінку інтерфейсу максимально передбачуваною та легкою для налагодження.

Центральним поняттям у цій архітектурі є Intent, який представляє конкретну дію користувача або подію, що має змінити стан програми. Коли користувач натискає кнопку або вводить текст, створюється об'єкт Intent, який передається до бізнес-логіки. Цей вхідний сигнал опрацьовується компонентом Model, який у контексті MVI розглядається як єдине джерело істини, що зберігає стан усього екрана в незмінному вигляді. Після обробки

Intent модель генерує новий стан, який передається назад у View для відображення.

Рівень View у цій структурі є максимально пасивним і просто реагує на кожну зміну стану, оновлюючи графічні елементи відповідно до отриманих даних. Завдяки використанню незмінних об'єктів стану розробники уникають поширених помилок, пов'язаних із неочікуваними мутаціями даних у різних частинах коду. Такий підхід ідеально поєднується з декларативними інструментами розробки, як-от Jetpack Compose, де інтерфейс напряму перемальовується у відповідь на оновлення єдиного об'єкта стану, забезпечуючи високу стабільність та легкість тестування логіки застосунку.

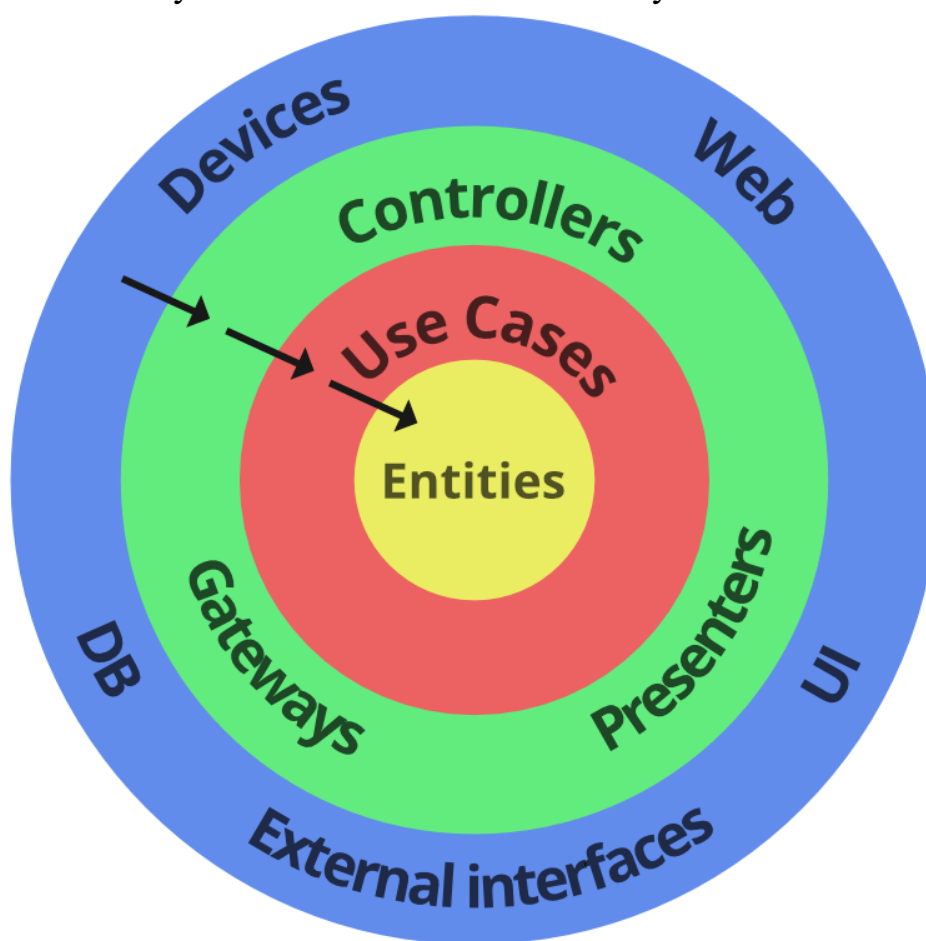


Рис. 1.8 – Схема патерна Clean Architecture

Чиста архітектура, або Clean Architecture (рис. 1.8), запропонована Робертом Мартіном, є концептуальним підходом до розробки програмного забезпечення, що базується на принципі незалежності бізнес-логіки від зовнішніх факторів, таких як бази даних, фреймворки або інтерфейс користувача [9]. Основна мета цієї архітектури полягає у створенні системи,

яка є стійкою до змін, легко тестується та підтримується протягом тривалого часу.

Фундаментом архітектури є правило залежностей, яке стверджує, що залежності в коді можуть бути спрямовані лише всередину, до центру системи, де знаходиться найбільш абстрактна та стабільна логіка. У самому серці розташовані сутності (Entities), які описують базові об'єкти бізнес-області та правила, що залишаються незмінними незалежно від того, як саме працює застосунок. Наступним шаром є випадки використання (Use Cases), які містять специфічну для конкретного продукту логіку та оркеструють потік даних між сутностями та зовнішніми компонентами.

На периферії системи знаходяться адаптери інтерфейсів та зовнішні деталі реалізації, як-от контролери, репозиторії, графічний інтерфейс та механізми збереження даних. Завдяки такому розподілу, розробник може повністю замінити локальну базу даних на хмарне сховище або перейти з однієї бібліотеки візуалізації на іншу, не вносячи жодних змін у ядро системи з бізнес-правилами. Це забезпечує високу гнучкість розробки, дозволяє легко писати модульні тести для кожного шару окремо та гарантує, що технічні рішення щодо інфраструктури не будуть диктувати структуру логіки самого застосунку.

Традиційні підходи MVC та MVP в сучасній Android-розробці часто призводить до перевантаження контролера або презентера надлишковим кодом, що суттєво ускладнює тестування та подальшу підтримку. Патерн MVI базується на принципах реактивного програмування та односпрямованого потоку даних, що забезпечує передбачуваність роботи програми, але вимагає складнішої реалізації. Clean Architecture найкраще підходить для великих корпоративних додатків і наявність багатьох абстрактних шарів сильно ускладнює розробку невеликих додатків. Тому найбільш оптимальним і рекомендованим з боку Google вибором є архітектура MVVM, яка забезпечує чітке розділення відповідальності, що запобігає витокам пам'яті і робить код значно легшим для автоматизованого тестування та підтримки.

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

1.4 Обґрунтування вибору інструментальних засобів

Під час проектування та безпосередньої програмної імплементації мобільного застосунку «LiteraList» буде використано комплексний інструментарій та стек сучасних технологій, що повністю відповідають галузевим стандартам розробки під операційну систему Android.

В якості IDE найактуальнішим варіантом є Android Studio. Android Studio — це офіційне інтегроване середовище розробки для створення додатків на базі операційної системи Android, розроблене корпорацією Google на основі платформи IntelliJ IDEA від компанії JetBrains [10]. Це потужний інструмент, який надає розробникам повний цикл засобів для написання коду, тестування та налагодження програмного забезпечення для смартфонів, планшетів, годинників, телевізорів та автомобілів. Однією з ключових особливостей середовища є інтелектуальний редактор коду, який підтримує мови Java, Kotlin та C++, пропонуючи розширене автодоповнення, аналіз помилок у реальному часі та автоматичний рефакторинг. Система збірки базується на Gradle, що дозволяє гнучко налаштовувати процес компіляції, створювати кілька варіантів додатків для різних пристроїв з одного вихідного коду та ефективно керувати залежностями. Для перевірки роботи програм передбачений швидкий і функціональний емулятор Android, який імітує різні апаратні характеристики, версії системи та геопозицію. Важливою складовою є набір інструментів аналізу продуктивності (Profilers), які в реальному часі відстежують споживання пам'яті, використання процесора, мережеву активність та енергоспоживання, допомагаючи оптимізувати додаток. Також середовище інтегроване з Google Play Console та інструментами Firebase, що спрощує монетизацію, хмарне тестування та публікацію готового продукту в магазині. Завдяки модульній структурі та постійним оновленням, Android Studio залишається головним стандартом у світі мобільної розробки, забезпечуючи високу швидкість роботи та зручність для створення як простих проектів, так і складних корпоративних систем.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

В якості мови програмування було обрано Kotlin, адже це сучасна статично типізована мова програмування, розроблена компанією JetBrains, яка стала офіційною та пріоритетною мовою для розробки під Android за рішенням Google у 2017 році [11]. Головною причиною її популярності є повна сумісність із Java, що дозволяє розробникам поступово переходити на нову мову, використовуючи існуючі бібліотеки та код у межах одного проекту. Kotlin було створено з метою усунення основних недоліків Java, зокрема він пропонує набагато лаконічніший синтаксис, що дозволяє зменшити обсяг шаблонного коду на 40% і більше, роблячи розробку швидшою, а підтримку проектів простішою. Однією з найважливіших функцій мови є вбудована безпека щодо порожніх посилань (Null Safety), яка на етапі компіляції запобігає виникненню найпоширенішої помилки в програмуванні — NullPointerException, що суттєво підвищує стабільність мобільних додатків. Окрім цього, Kotlin впроваджує потужний механізм корутин (Coroutines), який дозволяє легко керувати асинхронними операціями, такими як запити до мережі або бази даних, не блокуючи головний потік інтерфейсу та забезпечуючи плавну роботу програми. Сучасна розробка на Android також базується на декларативному підході до створення інтерфейсів через Jetpack Compose, який повністю побудований на Kotlin і використовує його функціональні можливості для опису UI-компонентів. Завдяки підтримці мультиплатформенності, мова дозволяє ділити логіку між Android та iOS, що економить ресурси бізнесу. Високий рівень підтримки з боку спільноти та Google, наявність сучасних конструкцій на кшталт функцій-розширень, лямбда-виразів та розумного приведення типів роблять Kotlin стандартом індустрії, який забезпечує розробникам продуктивне написання коду, а користувачам — надійний та швидкий продукт.

Для побудови користувацького інтерфейсу будуть використані Jetpack Compose та Material Design. Jetpack Compose — це сучасний декларативний інструментарій від Google, який повністю змінив підхід до створення користувацьких інтерфейсів в Android, відмовившись від застарілої системи

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

на основі XML на користь чистого коду мовою Kotlin [12]. На відміну від традиційного імперативного підходу, де розробник мав вручну описувати кожен крок зміни стану UI, у Compose інтерфейс описується як функція від поточних даних: коли дані змінюються, система автоматично перемальовує лише необхідні частини екрана. Це забезпечує безпрецедентну швидкість розробки та легкість у підтримці коду, оскільки логіка та зовнішній вигляд тепер знаходяться в одному місці. Найтісніша інтеграція Jetpack Compose з Material Design — відкритою системою дизайну від Google — дозволяє створювати візуально привабливі та інтуїтивно зрозумілі додатки з мінімальними зусиллями [13]. Material Design надає розробникам готові компоненти, такі як кнопки, картки, панелі навігації та текстові поля, які вже містять вбудовану анімацію, тіні та логіку взаємодії, що відповідає сучасним стандартам естетики. Використання цієї зв'язки в сучасній розробці обумовлене необхідністю створення складних, адаптивних інтерфейсів, які коректно працюють на різних форм-факторах, від смартфонів до складних пристроїв і планшетів.

Керування збереженням даних буде відбуватися за допомогою бібліотеки Room, яка є обгорткою над SQLite, що входить до складу набору Android Jetpack і є офіційно рекомендованим інструментом для збереження структурованих даних у локальній пам'яті пристрою [14]. Хоча в основі лежить SQLite, перевірена часом реляційна система керування базами даних, пряма робота з нею через стандартні API є складною та ризикованою через необхідність писати велику кількість шаблонного коду для створення таблиць, виконання запитів та ручного перетворення результатів у об'єкти Kotlin [15]. Room вирішує ці проблеми, пропонуючи рівень абстракції, який дозволяє розробникам взаємодіяти з базою даних через зручні анотації, такі як @Entity для опису таблиць та @Dao для визначення методів доступу до даних. Однією з найважливіших переваг Room є перевірка SQL-запитів на етапі компіляції: якщо розробник помилиться в назві колонки або синтаксисі запиту, програма

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

просто не запуститься, що дозволяє виявити помилки задовго до запуску додатка.

У сучасній розробці Room та SQLite відіграють критичну роль у реалізації стратегії "offline-first", яка дозволяє додаткам повноцінно працювати навіть за відсутності інтернет-з'єднання. Завдяки Room дані, отримані з мережі, кешуються в локальній базі, забезпечуючи миттєвий доступ до інформації та плавну роботу інтерфейсу. Бібліотека ідеально інтегрована з сучасними реактивними інструментами, такими як Kotlin Coroutines та Flow, що дозволяє UI автоматично оновлюватися, щойно в базі даних відбуваються будь-які зміни. Таке поєднання реляційної бази SQLite та зручності високорівневої абстракції Room робить локальне зберігання надійним, безпечним та легким у підтримці для будь-якого сучасного Android-проекту.

Висновки до розділу 1

Було поставлено задачу для виконання кваліфікаційної роботи та визначено план розробки програмного рішення, проведено системний аналіз вимог.

Проаналізовано аналоги розроблюваного програмного продукту, якими являються Goodreads, StoryGraph та Bookly. Було розглянуто їх функціонал та виділено найважливіші для користувачів елементи додатків.

Розглянуто популярні патерни для Android додатків такі як MVC, MVP, MVVM, MVI та Clean Architecture. В результаті для розробки було обрано MVVM, який є рекомендованим Google та значно легшим для автоматизованого тестування і підтримки.

Наприкінці розглянуті обрані інструментальні засоби для розробки програмного рішення: Android Studio, Kotlin, Jetpack Compose, Material Design та бібліотеку Room для зручного керування SQLite.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ КНИЖКОВОГО ОБЛІКУ

2.1 Визначення варіантів використання

Для розроблюваного мобільного застосунку «LiteraList» можна виділити кілька ключових варіантів використання, які описують взаємодію користувача з системою та повністю відображають закладений функціонал. Оскільки основним актором у системі виступає безпосередньо читач, усі прецеденти зосереджені навколо керування персональною бібліотекою та відстеження читацької активності.

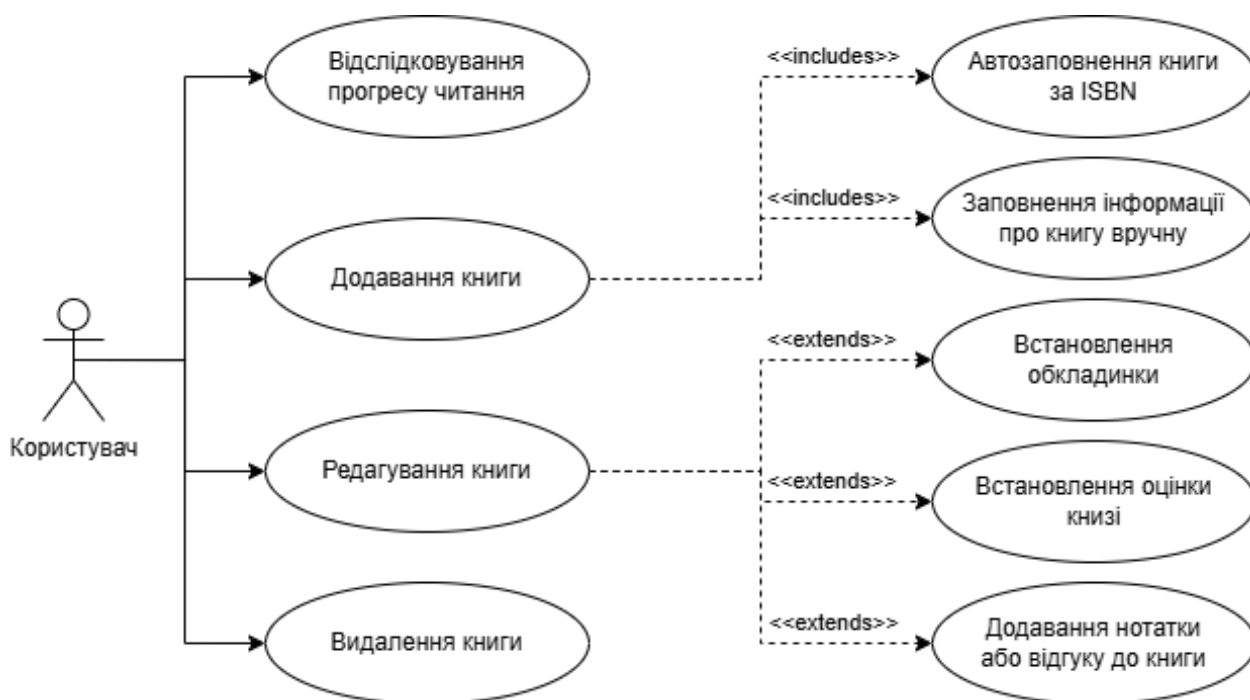


Рис. 2.1. Діаграма варіантів використання

Для наочної демонстрації було побудовано діаграму варіантів використання системи за стандартом UML 2.0 (Рис. 2.1) [16]. Першим і базовим варіантом використання є локальне керування каталогом книг, що включає додавання нових видань, редагування поточної інформації та видалення застарілих карток. Під час додавання книги користувач має змогу самостійно заповнити всі поля необхідною інформацією, але також присутній важливий супутній прецедент — автоматичне наповнення метаданих за допомогою міжнародного стандартного номера ISBN [17]. Користувач

вводить унікальний код, після чого застосунок надсилає запит до зовнішньої мережевої інфраструктури, та самостійно заповнює текстові поля, такі як назва, автори, кількість сторінок та опис. Таким чином користувач мінімізує кількість дій необхідних для додавання книги, пришвидшує додавання великого об'єму літератури та покращує загальний досвід користування додатком.

Іншим варіантом використання є керування графічними обкладинками літературних творів. Цей процес передбачає, що користувач обирає фотографію з галереї пристрою, а система копіює графічний файл в ізольовану внутрішню файлову систему застосунку. При цьому автоматично спрацьовує алгоритм знищення застарілих або замінених зображень, що дозволяє уникнути надлишкового засмічення пам'яті смартфона.

Також наявний варіант використання, пов'язаний із моніторингом читацької діяльності та інтерактивним відстеженням прогресу. Користувач має можливість фіксувати кількість фактично опрацьованих сторінок, на основі чого система здійснює динамічну візуалізацію. Застосунок відображає прогрес у вигляді графічних елементів користувацького інтерфейсу, що допомагає читачу наочно оцінювати власні досягнення. Також користувач може залишати персональні відгуки та рецензії на опрацьовані книги, завершуючи цим повний цикл взаємодії з кожним виданням у своїй домашній бібліотеці. Розглянуті варіанти описані в таблицях 2.1-2.8, де детально описані сценарії використання системи:

Таблиця 2.1

Прецедент «Додавання книги вручну»

Назва	Додавання книги вручну
Актори	Користувач
Опис	Користувач додає книжку до системи заповнюючи необхідні текстові поля

Таблиця 2.2

Прецедент «Додавання книги за ISBN»

Назва	Додавання книги за ISBN
Актори	Користувач
Опис	Користувач при додаванні книги заповнює лише ISBN, а додаток самостійно заповнює решту інформації наявної на зовнішньому ресурсі

Таблиця 2.3

Прецедент «Редагування книги»

Назва	Редагування книги
Актори	Користувач
Опис	Користувач може змінювати основні відомості про книгу або доповнювати відсутні дані

Таблиця 2.4

Прецедент «Встановлення обкладинки»

Назва	Встановлення обкладинки
Актори	Користувач
Опис	Користувач може самостійно встановити обкладинку книги обравши відповідну фотографію на мобільному пристрої

Таблиця 2.5

Прецедент «Встановлення оцінки»

Назва	Встановлення оцінки
Актори	Користувач

продовж. табл. 2.5

Опис	Користувач за допомогою зручного повзунка може поставити книзі оцінку від 1 до 10 балів
------	---

Таблиця 2.6

Прецедент «Додавання нотатки або відгуку до книги»

Назва	Додавання нотатки або відгуку до книги
Актори	Користувач
Опис	Користувач може закріпити за книгою свої думки, нотатки, відгук чи рецензію

Таблиця 2.7

Прецедент «Відслідковування прогресу читання»

Назва	Відслідковування прогресу читання
Актори	Користувач
Опис	Користувач вносить в систему кількість прочитаних сторінок книги та завжди може подивитися на якій сторінці він зупинився

Таблиця 2.8

Прецедент «Видалення книги»

Назва	Видалення книги
Актори	Користувач
Опис	Користувач може видалити будь-яку книгу очистивши всю інформацію про неї

2.2 Проектування шару даних

Модульна система мобільного застосунку «LiteraList» побудована з урахуванням сучасних стандартів інженерії програмного забезпечення та базується на принципах архітектури MVVM. Розподіл кодової бази на ізольовані логічні блоки дозволяє досягти високого рівня гнучкості, масштабованості та спрощує процес модульного тестування компонентів. Замість монолітної структури застосунків використовує чітко розмежовані рівні, кожен з яких виконує свою суворо визначену роль і взаємодіє з іншими рівнями лише через відкриті інтерфейси та правила односпрямованого потоку залежностей.

Шар даних являється елементом Model у архітектурі MVVM та виступає реалізацією принципу single source of true, тобто єдиним джерелом правди [18]. Всі інші шари додатку напряму залежать від цієї структури. Для зберігання даних було обрано бібліотеку Room, і її спочатку потрібно ініціалізувати. Для цього в пакеті data було створено абстрактний клас LiteraListDatabase, який наслідується від RoomDatabase:

```
@Database(entities = [Book::class], version = 2, exportSchema =
false)
abstract class LiteraListDatabase : RoomDatabase() {

    abstract val dao: BookDao

    companion object {
        @Volatile
        private var INSTANCE: LiteraListDatabase? = null

        fun getDatabase(context: Context): LiteraListDatabase {
            return INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context, LiteraListDatabase::class.java,
                    "literalist_database"
                )
            }
        }
    }
}
```

```

).fallbackToDestructiveMigration(true).build().also { INSTANCE =
it }

    }

}

}

```

За допомогою анотації `@Database` зазначаються сутності, які входять в базу даних, версія бази даних та опціональне збереження історії схем бази даних. До класу передається об'єкт доступу до даних `dao`, який буде розглянутий далі. Змінна `INSTANCE`, яка представляє екземпляр бази даних, повинна бути анотована `@Volatile`, оскільки всі звернення до бази даних мають бути атомарними для забезпечення цілісності та виключення стану гонитви. Доступ до цього екземпляру відбувається за допомогою функції `getDatabase()`, яка спочатку перевіряє чи він існує, оскільки процес ініціалізації бази даних дуже ресурсоємний. Також для створення екземпляру обов'язково надається контекст `Context`, який дає дозвіл на використання внутрішнього файлового простору додатку. Для зручності тестування, було використано правило деструктивної міграції, щоб при зміні схеми БД, існуючі тестові дані знищувались і не викликали конфлікт з новою схемою. В подальшому `Room` дозволяє налаштовувати алгоритми міграції, якщо потрібно буде зберігати записи в таблиці. Головною та єдиною сутністю додатку, яка буде зберігатися в базі даних є книга, представлена класом даних `Book`.

```

@Entity(tableName = "books")
data class Book(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val title: String,
    val author: String,
    val year: Int? = null,
    val description: String = "",
    val isbn: String = "",
    val coverUri: String? = null,

```

```

    val totalPages: Int = 0,
    val readPages: Int = 0,
    val rating: Int = 0,
    val review: String = "",
    val isFavorite: Boolean = false,
)

```

Анотація @Entity вказує бібліотеці Room, що цей клас потрібно конвертувати в сутність і вказати назву таблиці «books». Розглянемо поля сутності в таблиці 2.9.

Таблиця 2.9

Таблиця полів сутності Book

Поле	Тип	Обов'язковість	Опис
id	INTEGER	-	Ідентифікатор книги, який є первинним ключем з автоінкрементом
title	TEXT	+	Назва книги
author	TEXT	+	Автор або автори книги
year	INTEGER	-	Рік видання
description	TEXT	-	Опис книги
isbn	TEXT	-	ISBN код книги
coverUri	TEXT	-	Посилання на розташування обкладинки книги
totalPages	INTEGER	-	Загальна кількість сторінок книги
readPages	INTEGER	-	Кількість прочитаних сторінок книги
rating	INTEGER	-	Оцінка книги
review	TEXT	-	Рецензія або нотатки на книгу
isFavourite	INTEGER	-	Чи є книга улюбленою

Варто зауважити, що в SQLite немає явного типу даних Boolean, тому для зберігання булевих значень використовується INTEGER тип зі значеннями 0 (False) та 1(True), які бібліотека Room автоматично конвертує.

Для взаємодії із базою даних використовується DAO. DAO (Data Access Object) — це спеціальний компонент, який виступає інтерфейсом для взаємодії з базою даних [19]. Замість написання складних низькорівневих SQL-запитів вручну, розробник створює DAO-інтерфейс, де за допомогою простих функцій та анотацій (таких як `@Insert`, `@Update`, `@Delete` або `@Query`) чітко описує всі необхідні операції: додавання, оновлення, видалення чи вибірку даних. Головна перевага DAO полягає в тому, що Room автоматично генерує реалізацію цих методів, перевіряє правильність SQL-запитів ще на етапі компіляції додатка.

```
@Dao
interface BookDao {
    @Query("SELECT * FROM books ORDER BY title ASC")
    fun getAllBooks(): Flow<List<Book>>

    @Query("SELECT * FROM books WHERE id = :id")
    fun getBookById(id: Int): Flow<Book?>

    @Upsert
    suspend fun upsertBook(book: Book)

    @Delete
    suspend fun deleteBook(book: Book)
}
```

`BookDao` містить 4 асинхронні функції які реалізують CRUD операції. Функція `upsertBook(book: Book)` відповідає за додавання, або оновлення книги, `deleteBook(book: Book)` видаляє книгу. Функція `getBookById(id: Int)` знаходить книгу по її ідентифікатору, а `getAllBooks()` повертає список усіх книжок в базі даних. Проте слід зазначити, що обгортаються в так звані потоки (Flows), які є основою реактивного програмування Android додатків.

Kotlin Flow — це сучасний інструмент для роботи з асинхронними потоками даних в Android, який побудований на базі корутин і повністю інтегрований у мову Kotlin [20]. На відміну від звичайних функцій, які

повертають лише одне значення, Flow дозволяє послідовно "випромінювати" (emit) кілька значень протягом певного часу, що робить його ідеальним для отримання актуальних оновлень із бази даних, завантаження файлів частинами або обробки мережових запитів у реальному часі. Це означає, що відображуваний список книжок буде автоматично оновлюватись, якщо відбулися зміни в базі даних, навіть не викликаючи функцію ще раз.

Для отримання інформації про книги з мережі, було використано Google Books API [21]. Він містить надзвичайно велику базу книжок та пропонує безкоштовний обмежений доступ для розробників. Базовий план дозволяє здійснювати до 1000 запитів на день, чого достатньо для цілей додатка. Щоб працювати з цим API, необхідно створити мережовий модуль який буде дерево залежностей і використовує патерн ін'єкції залежностей (Dependency Injection). Для спрощення побудови дерева залежностей була використана бібліотека Hilt.

Hilt — це офіційно рекомендована компанією Google бібліотека для впровадження залежностей в Android-додатках [22]. Вона автоматизує процес створення, зв'язування та керування життєвим циклом об'єктів у додатку, позбавляючи розробника необхідності писати фабрики вручну. Завдяки готовій інтеграції з основними компонентами Android, Hilt автоматично знищує об'єкти, коли вони більше не потрібні, що запобігає витокам пам'яті. Головними перевагами застосування Hilt є суттєве підвищення читабельності та чистоти коду, легка масштабованість проекту, просте налаштування модульних тестів завдяки швидкій заміні реальних об'єктів на тестові.

Для виконання запитів до API також була використана бібліотека Retrofit, яка перетворює складні HTTP-запити до REST API у звичайні інтерфейси мови Kotlin або Java [23]. Замість написання низькорівневого коду для відкриття мережових з'єднань, обробки потоків даних та ручного парсингу відповідей, розробнику достатньо описати структуру запитів (наприклад, @GET, @POST) за допомогою простих анотацій прямо у функціях інтерфейсу. В якості серіалізатора/десеріалізатора було використано Kotlinx

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Serialization. Побудова мережевого модуля починається з конфігурації провайдера серіалізатора provideJson().

```
@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {
    @Provides
    @Singleton
    fun provideJson(): Json {
        return Json {
            ignoreUnknownKeys = true
            coerceInputValues = true
        }
    }
}
```

Роль серіалізатора полягає в перетворенні складних JSON класів у Kotlin класи, які міститимуть лише потрібну інформацію. Для цього використані два прапори, ignoreUnknownKeys дозволяє пропустити ключі, які не вказані в цільовому класі даних, а coerceInputValues дозволяє обробляти некоректні значення, замінюючи їх на значення за замовчуванням або null.

Далі було створено провайдер provideOkHttpClient(), який перехоплює всі вихідні запити, які виконує Retrofit, та додає в кінці API ключ, як того вимагає Google Books API. Сам API ключ зберігається окремо в конфігурації збірки.

```
@Provides
@Singleton
fun provideOkHttpClient(): OkHttpClient {
    return OkHttpClient.Builder()
        .addInterceptor { chain ->
            val originalRequest = chain.request()
            val urlWithKey = originalRequest.url.newBuilder()
                .addQueryParameter("key",
BuildConfig.GOOGLE_BOOKS_API_KEY).build()
            val request =
originalRequest.newBuilder().url(urlWithKey).build()
        }
}
```

```

        chain.proceed(request)
    }
    .build()
}

```

Для побудови екземпляра Retrofit використовуються наведені вище залежності. `baseUrl()` встановлює базовий URL за яким створюються запити, `addConverterFactory()` в якості конвертера вказує JSON серіалізатор, а `.client()` підключає `Interceptor`.

```

@Provides
@Singleton
fun provideRetrofit(json: Json, okHttpClient: OkHttpClient):
Retrofit {
    return Retrofit.Builder()
        .baseUrl("https://www.googleapis.com/books/v1/")

        .addConverterFactory(json.asConverterFactory("application/json".
toMediaType()))
        .client(okHttpClient)
        .build()
}

```

Сервіс `GoogleBooksApiService` містить в собі функцію `searchByIsbn()` яка створює запит до API.

```

interface GoogleBooksApiService {
    @GET("volumes")
    suspend fun searchByIsbn(@Query("q") query: String):
GoogleBooksResponse
}

```

Результат запиту десеріалізується в декілька вкладених Kotlin data class. Серед них в подальшому буде використаний `VolumeInfo`.

```

@Serializable
data class GoogleBooksResponse(
    val items: List<BookItem>? = null
)
@Serializable

```

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

```

data class BookItem(
    val volumeInfo: VolumeInfo
)
@Serializable
data class VolumeInfo(
    val title: String? = null,
    val authors: List<String>? = null,
    val publishedDate: String? = null,
    val description: String? = null,
    val pageCount: Int? = null,
    val imageLinks: ImageLinks? = null
)
@Serializable
data class ImageLinks(
    val thumbnail: String? = null
)

```

Щоб завершити створення мережевого модуля, було імплементовано **GoogleBooksApiService**, за допомогою `retrofit.create`.

```

@Provides
@Singleton
fun provideGoogleBooksApiService(retrofit: Retrofit):
GoogleBooksApiService {
    return retrofit.create(GoogleBooksApiService::class.java)
}

```

Для об'єднання обох джерел даних та створення єдиного інтерфейсу взаємодії шару бізнес-логіки та шару даних було використано патерн репозиторій, який містить всі реалізовані раніше функції. В нашому випадку репозиторій представляє **BookRepository**.

```

class BookRepository(
    private val bookDao: BookDao,
    private val apiService: GoogleBooksApiService
) {

    val allBooks: Flow<List<Book>> = bookDao.getAllBooks()
}

```

```

fun getBookById(id: Int): Flow<Book?> {
    return bookDao.getBookById(id)
}

suspend fun upsert(book: Book) {
    bookDao.upsertBook(book)
}

suspend fun delete(book: Book) {
    bookDao.deleteBook(book)
}

suspend fun getBookFromApi(isbn: String): VolumeInfo? {
    val response = apiService.searchByIsbn("isbn:$isbn")
    return response.items?.firstOrNull()?.volumeInfo
}
}

```

BookRepository необхідно надати екземпляри **BookDao** та **GoogleBooksApiService**, а **BookDao** повинен мати доступ до екземпляра бази даних, тому було створено ще один модуль залежностей **DataModule**.

```

@Module
@InstallIn(SingletonComponent::class)
object DataModule {

    @Provides
    @Singleton
    fun provideDatabase(@ApplicationContext context: Context):
LiteraListDatabase {
        return LiteraListDatabase.getDatabase(context)
    }

    @Provides
    fun provideBookDao(database: LiteraListDatabase): BookDao {
        return database.dao
    }
}

```

```

@Provides
@Singleton
fun provideBookRepository(dao: BookDao, apiService:
GoogleBooksApiService): BookRepository {
    return BookRepository(dao, apiService)
}
}

```

Для демонстрації утвореного дерева залежностей шару даних було побудовано діаграму залежностей (рис. 2.2).

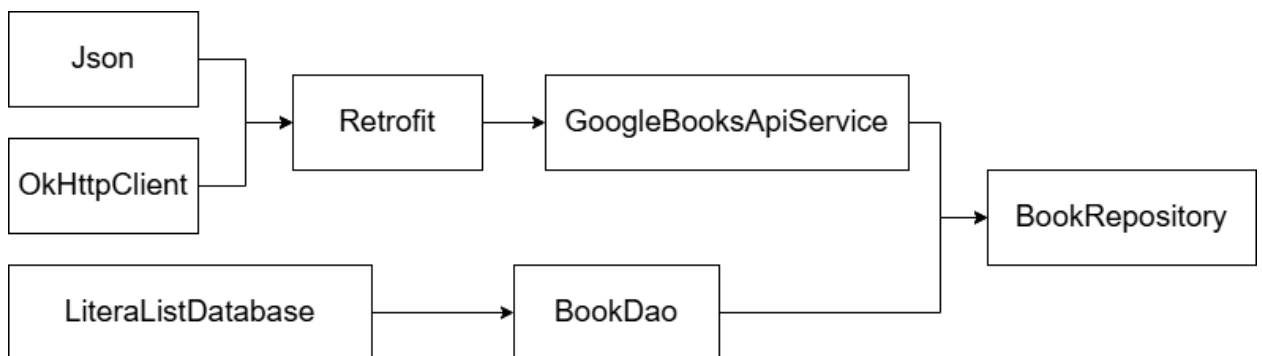


Рис. 2.2 – Діаграма залежностей шару даних

Таким чином вся взаємодія з шаром даних зводиться до викликів асинхронних функцій BookRepository. Використання такого підходу підвищує надійність додатку, полегшує його модернізацію, підтримку та тестування. Шару бізнес-логіки не потрібні деталі реалізації джерел даних. Якщо в майбутньому потрібно буде додати нові джерела даних, достатньо створити нову реалізацію та додати відповідні залежності до репозиторію.

2.3 Проєктування шару бізнес-логіки

У межах архітектурного патерну MVVM компонент ViewModel відіграє роль інтелектуального посередника та сполучної ланки між графічним інтерфейсом користувача та шаром даних системи. Головне призначення цього компонента полягає в акумуляції, збереженні та підготовці даних, отриманих з нижчих рівнів архітектури, для їх подальшого пасивного відображення на екрані. ViewModel повністю ізольована від конкретної

реалізації візуальних елементів або фреймворків макетування або посилань на контекст операційної системи, що запобігає витокам пам'яті під час рекомпозиції екранів та забезпечує повну незалежність логіки представлення від життєвого циклу графічної оболонки.

З іншого боку, ViewModel виступає точкою обробки подій взаємодії, приймаючи наміри користувача та транслюючи їх у відповідні виклики методів бізнес-правил чи репозиторіїв даних. Замість прямого маніпулювання елементами екрана, цей компонент оновлює реактивні стани або потоки даних, які відстежує інтерфейс користувача. Такий підхід забезпечує автоматичне та синхронне оновлення графічної оболонки при зміні внутрішнього стану, робить структуру програми гнучкою, а саму логіку представлення — максимально доступною для ізольованого автоматизованого тестування без залучення важковагових компонентів середовища виконання.

Кожна ViewModel зберігає в собі стан активного екрану користувацького інтерфейсу і відповідно її життєвий цикл повністю залежить від того, який екран зараз використовується користувачем. Після того, як користувач залишає певну сторінку додатку, ViewModel автоматично знищується, звільняючи ресурси пристрою. Щоб розкрити стан екрана для шару представлення використовуються потоки стану (StateFlow) які реалізують патерн Observer як постачальники [24]. Шар представлення не може напряму змінювати свій стан, він лише стежить за змінами стану і відповідно модифікує своє відображення, таким чином дотримуючись принципу інкапсуляції.

Розглянемо BookViewModel, яка відповідає за відображення інформації про книгу на екрані BookScreen. Дана ViewModel для функціонування потребує екземпляр BookRepository для взаємодії з базою даних, Context операційної системи для роботи з файлами обкладинок та SavedStateHandle для отримання аргументів навігації. Вони реалізують патерн Singleton, тобто існують в єдиному екземплярі протягом всього життєвого циклу додатку, а тому щоб ViewModel їх використовувала, застосовуються інструменти

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

бібліотеки Hilt. Вона позначається анотацією `@HiltViewModel`, а залежності надаються за допомогою `dependency injection` конструктора. Таким чином можна безпечно надавати `Context` операційної системи за допомогою анотації `@param:ApplicationContext`.

```
@HiltViewModel
class BookViewModel @Inject constructor(
    private val bookRepository: BookRepository,
    @param:ApplicationContext private val context: Context,
    savedStateHandle: SavedStateHandle
) : ViewModel() {
    private val _uiState = MutableStateFlow(BookUiState())
    val uiState: StateFlow<BookUiState> = _uiState.asStateFlow()
```

Інкапсуляція стану відбувається за допомогою двох полів: `_uiState` – приватне поле для зміни стану та публічне `read-only` поле `uiState`, яке відкриває стан для шару представлення. Стан зберігається за допомогою класу даних `BookUiState`, а для перетворення в сутність `Book` та навпаки, використовуються функції розширення `BookUiState.toBook` та `Book.toBookUiState` відповідно.

`BookViewModel` при створенні автоматично викликає функцію `loadBook`, за допомогою `init` блока, та передає їй ідентифікатор книги для завантаження, отриманий з аргументів навігації. Для цього створюється корутина, життєвий цикл якої прив'язаний до `ViewModel`. Це означає, що якщо користувач закриє екран, або перейде на інший, корутина автоматично відміняється. Також тут відбувається отримання останнього актуального значення з потоку `Flow<Book?>` за допомогою `.collectLatest`.

```
fun loadBook(id: Int) {
    viewModelScope.launch {
        bookRepository.getBookById(id).collectLatest
        { book ->
            book?.let {
                _uiState.value = it.toBookUiState()
            }
        }
    }
}
```

```

    }
  }
}

```

Окремо варто розглянути функцію `updateCoverUri`, яка відповідає за встановлення та оновлення обкладинки книги. Для роботи з файлами використовується спеціальний диспетчер вводу/виводу `Dispatchers.IO`, щоб вивантажити блокуючу операцію до пулу потоків на вимогу і не впливати на дієздатність додатку. Функція отримує `URI` фотографії на пристрої, та копіює її у внутрішню папку додатка під новим ім'ям.

```

fun updateCoverUri(sourceUri: Uri) {
    viewModelScope.launch(Dispatchers.IO) {
        try {
            val oldPath = _uiState.value.coverUri
            val fileName =
                "cover_${System.currentTimeMillis()}.jpg"
            val destFile = File(context.filesDir, fileName)

            context.contentResolver.openInputStream(sourceUri)?.use { input
->

                destFile.outputStream().use { output ->
                    input.copyTo(output)
                }
            }

            oldPath?.let { path ->
                val oldFile = File(path)
                if (oldFile.exists()) {
                    oldFile.delete()
                }
            }

            _uiState.value = _uiState.value.copy(coverUri =
destFile.absolutePath)

            bookRepository.upsert(uiState.value.toBook())

        } catch (e: Exception) {

```

```

        e.printStackTrace()
    }
}
}

```

Такий підхід використовується через те, що користувач може змінити назву фотографії, перемістити її і тоді URI фотографії стане не актуальним і додаток втратить доступ до обкладинки. Також URI файлів часто змінюються при перезавантаженні мобільного пристрою. Якщо база даних містила попередній ідентифікатор фотографії, функція автоматично видаляє стару фотографію, щоб внутрішня папка додатка не накопичувала сміття. В кінці функція оновлює актуальну адресу фотографії для стану та зберігає зміни до бази даних.

Також BookViewModel як і інші ViewModel містить багато функцій для оновлення полів інтерфейсу з валідацією. Розглянемо на прикладі updateReadPages, яка відповідає за оновлення кількості прочитаних сторінок. Вона перевіряє, щоб кожен уведений користувачем символ був цифрою, а уведена кількість сторінок була меншою за загальну. Якщо користувач намагається увести некоректні дані, інтерфейс не оновлюється. Інші ViewModel мають подібну логіку та функції, тому окремо розглядатися не будуть.

```

fun updateReadPages(pagesInput: String) {
    if (!pagesInput.all { char -> char.isDigit() }) return
    val totalPages = _uiState.value.totalPages
    val pages = pagesInput.toIntOrNull()
    if (pages != null && pages <= totalPages) {
        _uiState.value = _uiState.value.copy(readPages = pages)
    }
}

```

Навігація була реалізована за допомогою бібліотеки Navigation Compose у поєднанні з типізованими шляхами і представляє собою сучасний та надійний підхід до організації переходів між екранами в додатках на Jetpack Compose [25]. Раніше розробникам доводилося використовувати рядкові

константи для опису маршрутів та передачі аргументів, що часто призводило до помилок під час виконання програми через неправильний формат рядків чи невідповідність типів. Впровадження строгої типізації вирішує цю проблему на етапі компіляції, використовуючи можливості Kotlin-серіалізації.

В основі нової концепції лежить опис кожного екрана або маршруту у вигляді звичайного об'єкта або класу, позначеного анотацією серіалізації. Екран, який не потребує вхідних даних, зазвичай представляється як самостійний об'єкт, тоді як екран із параметрами описується за допомогою класу даних, де кожне поле відповідає за конкретний аргумент, наприклад, ідентифікатор книги чи рядок пошуку. Бібліотека навігації автоматично аналізує структуру цих класів, генерує відповідні внутрішні схеми маршрутів та бере на себе всю рутинну роботу з упакування, передачі та розпакування даних.

Головним координатором процесу являється NavHost, який зв'язується з контролером NavController. Замість колишніх текстових шаблонів, функція розширення для створення графів тепер приймає як загальний тип безпосередньо серіалізований клас екрана. Під час виклику методу навігації розробник передає екземпляр класу даних із заповненими полями, а система самостійно конвертує його у внутрішній безпечний формат. На цільовому екрані аргументи автоматично витягуються з BackStackEntry, що дозволяє отримати доступ до вихідного об'єкта

Важливою перевагою такого підходу є інтеграція з архітектурними компонентами, зокрема з ViewModel. Коли цільовий екран ініціалізує свою ViewModel, аргументи навігації автоматично стають доступними через SavedStateHandle завдяки вбудованим методам розширення, які працюють з тими ж серіалізованими класами. Це забезпечує повну ізоляцію логіки, захищає додаток від збоїв через помилки в іменах ключів та гарантує, що структура даних залишатиметься цілісною та передбачуваною на всіх рівнях архітектури мобільного застосунку.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Переходи між екранами відбуваються після виклику callback функцій з інтерфейсу користувача в які передається `NavController` та функція розширення `.navigate(/*цільовий екран*/).`

2.4 Проектування шару представлення

Побудова шару представлення (View) у Jetpack Compose базується на принципах декларативного інтерфейсу та реактивного керування станом, що кардинально відрізняється від старого імперативного підходу з використанням XML-макетів. Замість створення статичного дерева елементів інтерфейсу, які потім потрібно примусово змінювати в коді, розробник описує графічну оболонку за допомогою спеціальних Composable функцій, що приймають поточні дані та самостійно трансформуються під час їх оновлення.

Центральним архітектурним шаблоном для організації цього процесу є односпрямований потік даних (Unidirectional Data Flow) [26]. Шар представлення у цій системі займає повністю пасивну позицію: він лише підписується на поточний стан інтерфейсу, що надходить зверху від архітектурного компонента ViewModel, і перетворює ці сирі дані на візуальні елементи. Будь-яка дія користувача, наприклад натискання кнопки, введення тексту чи гортання екрана, не змінює інтерфейс безпосередньо, а генерує подію (Event), яка надсилається назад до ViewModel для обробки та формування нового стану.

Для збереження реактивності та коректного відображення змін використовується концепція рекомпозиції. Коли потік даних, наприклад StateFlow, передає нове значення стану інтерфейсу, фреймворк Jetpack Compose автоматично фіксує, які саме компоновані функції залежать від цих даних, і перезбирає лише їх, мінімізуючи навантаження на систему. Доступ до бізнес-станів усередині екранів реалізується за допомогою інструментів життєвого циклу Android, які дозволяють автоматично припиняти спостереження за даними, коли екран стає неактивним, що повністю унеможливорює витoki пам'яті та оптимізує використання ресурсів пристрою.

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

Візуальне оформлення шару представлення спирається на специфікації дизайну Material Design 3, що забезпечує готову систему колірних палітр, типографіки, відступів та інтерактивних компонентів. Екрани проектуються за принципом модульності, де складні інтерфейси розбиваються на дрібні, незалежні та легкі для тестування компоновані функції, які повторно використовуються у різних частинах додатку. Таке чітке розділення відповідальності гарантує, що графічна оболонка залишатиметься гнучкою, легкою для підтримки та повністю ізольованою від внутрішньої бізнес-логіки застосунку.

Розглянемо для прикладу екран книги BookScreen, який містить інформацію про обрану книгу. Всі функції, які безпосередньо відповідають за інтерфейс позначаються анотацією @Composable. В параметри передаються дві callback функції які викликають відповідні методи при навігації та передають параметри навігації. Екран також отримує відповідну ViewModel для керування своєю логікою. Modifier – це спеціальний об'єкт, який використовується для налаштування, декорування та зміни поведінки елементів інтерфейсу, дозволяючи обробляти кліки, тривалі натискання або жести свайпу.

```
@Composable
fun BookScreen(
    onBookEdit: (Int) -> Unit,
    onChangeSave: () -> Unit,
    modifier: Modifier = Modifier,
    viewModel: BookViewModel = hiltViewModel()
) {
    val uiState by
viewModel.uiState.collectAsStateWithLifecycle()
    var selectedTabIndex by rememberSaveable {
mutableIntStateOf(0) }
    val titles = listOf(stringResource(R.string.notes),
stringResource(R.string.info))
```

Доступ до стану інтерфейсу відбувається за допомогою посилання на `uiState ViewModel.collectAsStateWithLifecycle()` збирає значення зі `StateFlow` до тих пір, поки екран активний. Як тільки користувач згортає додаток, збір значень зупиняється щоб звільнити ресурси пристрою.

Екран містить дві вкладки: вкладку нотаток та вкладку інформації про книгу і для того щоб зберігати стан активності вкладок, використовується `rememberSaveable` в поєднанні з `mutableStateOf`. Як було сказано раніше, екран не повинен зберігати стан, чи містити бізнес-логіку, але в подібних випадках це є прийнятним, бо збережене значення не представляє корисних даних і використовується суто для модифікації зовнішнього вигляду. `titles` представляє список із заголовків вкладок, але замість того щоб прямо в коді писати назви вкладок, було використано посилання на рядковий ресурс, який є частиною додатка. Це робиться для того, щоб полегшити підтримку додатком різних мов. Для кожної мови існує окремий `strings.xml` з якого підтягуються необхідні рядки.

Прикладом інтерактивного елемента додатка є повзунок для виставлення оцінки книзі. Для цього використовується вбудований елемент `Slider`. Поточне значення він отримує з `uiState`, а при перетягуванні він щоразу викликає лямбда вираз де передає функції `onRatingUpdate` поточне значення. Щоб користувач міг обрати значення від 1 до 10, а між ними було 8 кроків були використані відповідні параметри `valueRange` та `steps`. Текст після слайдера одразу відображає поточне значення.

```
Text(text = stringResource(R.string.rating), style =
MaterialTheme.typography.titleLarge)
Row(verticalAlignment = Alignment.CenterVertically, modifier =
Modifier.padding(5.dp)) {
    Slider(
        value = uiState.rating.toFloat(),
        onChange = { onRatingUpdate(it.toInt()) },
        steps = 8,
        valueRange = 1f..10f,
        modifier = Modifier.weight(1f)
```

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    )
    Spacer(modifier = Modifier.size(8.dp))
    Text(text = uiState.rating.toString())
}

```

Під час розробки, для відображення поточного вигляду та коригування елементів інтерфейсу були використані окремі функції попереднього перегляду, анотовані `@Preview`, які не потрапляють у зібраний проєкт, а слугують лише для налагодження. В них розробник передає об'єкт стану заповнений тестовими даними. Preview функції також дозволяють побачити вигляд елементів за світлої та темної теми пристрою.

Стилі, кольори, шрифти визначаються у файлах `Theme.kt`, `Color.kt` та `Type.kt`, які були згенеровані за допомогою ресурса Material Theme Builder [27]. Для того щоб ці стилі застосовувались до Composable функцій, достатньо елемент найвищого рівня обгорнути в `LiteraListTheme { ... }`.

Висновки до розділу 2

На етапі аналізу потреб користувача було виділено варіанти використання мобільного застосунку «LiteraList». Ці сценарії охоплюють повний життєвий цикл взаємодії з книгою і забезпечують комплексний підхід до управління домашньою бібліотекою, включаючи елементи мотивації та персоналізації досвіду користувача.

Шар даних дотримується підходу single source of truth та реалізується через локальну базу даних SQLite та зовнішній ресурс Google Books API. Було визначено основну сутність Book з відповідним набором атрибутів для зберігання бібліографічних даних і персоналізованої інформації. Для оновлення даних у реальному часі використано Kotlin Flow, що гарантує миттєве відображення змін. Інтеграція зовнішнього ресурсу передбачає підключення до Google Books API через бібліотеки Retrofit та Hilt, дозволяючи застосунку автоматично заповнювати потрібні поля інформації про книгу.

Шар бізнес-логіки представлений ViewModel та відповідає за обробку подій користувача та формування чистого, незмінного стану інтерфейсу, що повністю ізолювано від шару представлення. Ключові функції логіки включають асинхронне завантаження даних книги з використанням корутин та реалізують механізми взаємодії із файловою системою для встановлення обкладинки. Таким чином, бізнес-логіка не лише кешує дані, але й забезпечує валідацію даних введених користувачем, гарантуючи цілісність даних.

Шар представлення реалізовано за допомогою Jetpack Compose, що відображає декларативний та реактивний підхід до розробки UI. View виступає пасивним спостерігачем і відображає стан, який отримує від ViewModel. Для цього використовуються StateFlow, які гарантують, що UI оновлюється виключно при зміні стану. Візуальне оформлення відповідає сучасним стандартам Material Design 3, а навігація реалізована за допомогою Navigation Compose. Це забезпечує високий рівень модульності та стійкості системи до помилок на етапі компіляції.

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МОБІЛЬНИМ ЗАСТОСУНКОМ ДЛЯ КНИЖКОВОГО ОБЛІКУ

3.1 Порядок встановлення та налаштування параметрів системи

Для коректної роботи додатка, були визначені системні вимоги, які наведені в таблиці 3.1. Представлені у таблиці системні вимоги забезпечують коректне розгортання мобільного застосунку «LiteraList» на цільових пристроях користувачів. Мінімальний рівень обмежений архітектурними особливостями використаних технологій та гарантує базовий функціонал програми, тоді як рекомендовані параметри орієнтовані на досягнення максимальної швидкодії та доступу до зовнішнього джерела інформації про книги [28].

Таблиця 3.1

Таблиця системних вимог мобільного додатка

Параметр системи	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Android 8.0 Oreo (API рівень 26)	Android 10.0 Q (API рівень 29) або новіша версія
Оперативна пам'ять	2 ГБ RAM	4 ГБ RAM або більше
Мережеві інтерфейси	—	Стабільне підключення до мережі інтернет
Апаратні компоненти	Сенсорний дисплей середньої роздільної здатності	Сенсорний дисплей високої роздільної здатності

Додаток розповсюджуватиметься за допомогою самостійного APK файла. APK (Android Package) — це формат, який операційна система Android використовує для розповсюдження та встановлення мобільних додатків [29]. Його можна порівняти з архівом ZIP, оскільки він містить у собі абсолютно всі

компоненти, необхідні для роботи програми на пристрої. Усередині APK-файлу знаходяться скопійовані файли коду (вони мають розширення .dex і містять інструкції для віртуальної машини Android), файли ресурсів (такі як зображення, аудіо та інтерфейси користувача), а також файли сертифікатів, які підтверджують безпеку та авторство додатка.

Процес інсталяції мобільного застосунку «LiteraList» на пристрій під управлінням операційної системи Android за допомогою установчого інсталяційного файлу у форматі APK передбачає виконання кількох послідовних кроків. Початковим етапом є підготовка мобільного телефону або планшета, оскільки за замовчуванням система безпеки Android блокує інсталяцію програмного забезпечення, отриманого не з офіційного магазину Google Play Market. Для зняття цього обмеження користувачеві необхідно перейти в загальні налаштування операційної системи, знайти розділ безпеки або конфіденційності та активувати дозвіл на встановлення додатків із невідомих джерел (рис. 3.1):

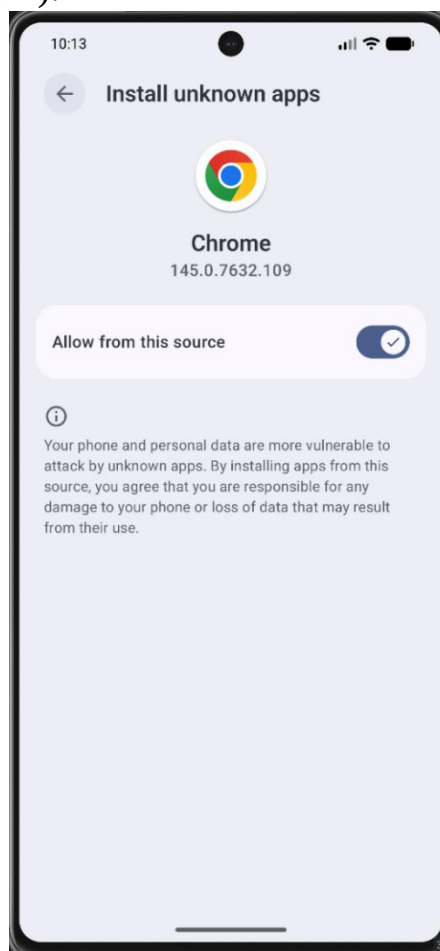


Рис. 3.1 – Надання дозволу на встановлення додатка

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Залежно від версії Android, цей дозвіл може надаватися як глобально для всієї системи, так і локально для конкретного інструменту, через який виконуватиметься завантаження, наприклад, для використовуваного веб-браузера чи стандартного провідника файлів.

Наступним етапом є безпосереднє перенесення інсталяційного пакета на цільовий пристрій, що досягається шляхом прямого завантаження інсталяційного файлу «LiteraList.apk» через браузер, отримання його електронною поштою чи месенджером, або ж копіюванням із персонального комп'ютера за допомогою USB-кабелю у внутрішню пам'ять телефону. Після успішного збереження APK-файлу користувач має відкрити будь-який доступний файловий менеджер, перейти до каталогу завантажень або тієї папки, куди було поміщено інсталятор, та натиснути на файл додатка для запуску інтегрованого системного майстра встановлення.

У цей момент на екрані з'явиться стандартне діалогове вікно операційної системи, яке відобразить назву програми «LiteraList» та запитає підтвердження наміру провести інсталяцію (рис. 3.2).

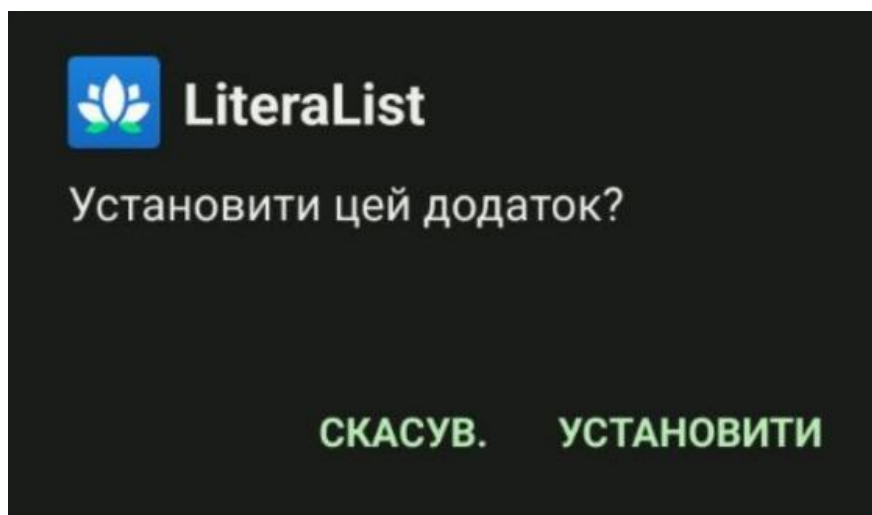


Рис. 3.2 – Запит на встановлення додатка

Після натискання кнопки підтвердження система розпакує ресурси програми та виконає її конфігурацію, а в разі додаткових запитів безпеки Play Захисту щодо перевірки невідомого розробника користувачеві слід обрати варіант ігнорування попередження та продовження процедури (рис. 3.3).

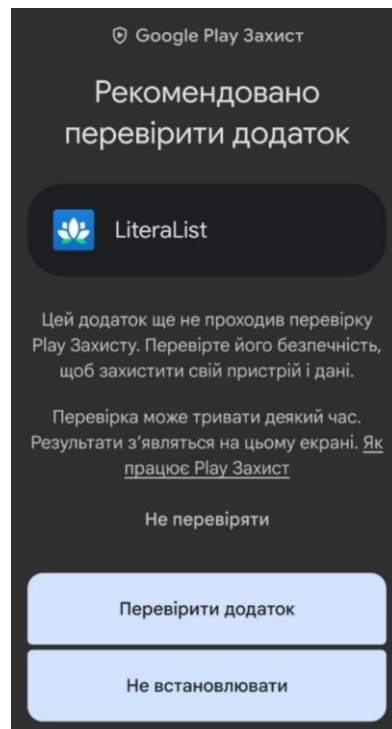


Рис. 3.3 – Запит на перевірку додатка Play Захист

Завершальний етап супроводжується повідомленням про успішне встановлення програмного продукту (рис. 3.4), після чого на робочому столі пристрою та в загальному меню з'явиться відповідна графічна іконка «LiteraList», готова до першого запуску та повноцінного використання для обліку домашньої бібліотеки.

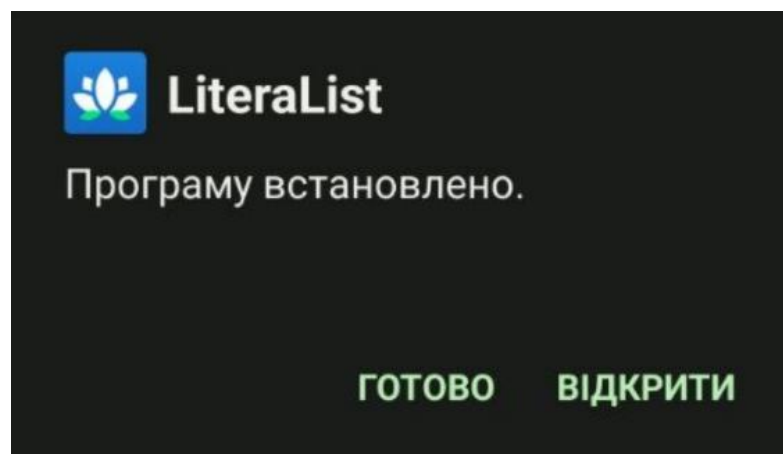


Рис. 3.4 – Підтвердження встановлення додатка

3.2 Структура інтерфейсу мобільного додатку

Інтерфейс «LiteraList» складається з трьох екранів. Коли користувач вперше запускає додаток, перше що він бачить, це екран порожньої книжкової

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

полиці та повідомлення, яке пропонує додати першу книгу. Для цього на екрані присутня відповідна кнопка (рис. 3.5).



Рис. 3.5 – Екран порожньої книжкової полиці

Рис. 3.6 – Екран редагування інформації про книгу

Коли користувач натискає на кнопку, перед ним з’являється екран редагування книги(рис. 3.6). Обов’язкові поля підсвічені червоним до тих пір,

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

поки вони не будуть заповнені. Також кнопка «Зберегти» залишатиметься неактивною, доки не будуть заповнені обов'язкові поля. В кінці поля ISBN присутня кнопка пошуку, яка дозволяє заповнити решту полів автоматично, якщо запис про відповідну книгу присутній в Google Books.

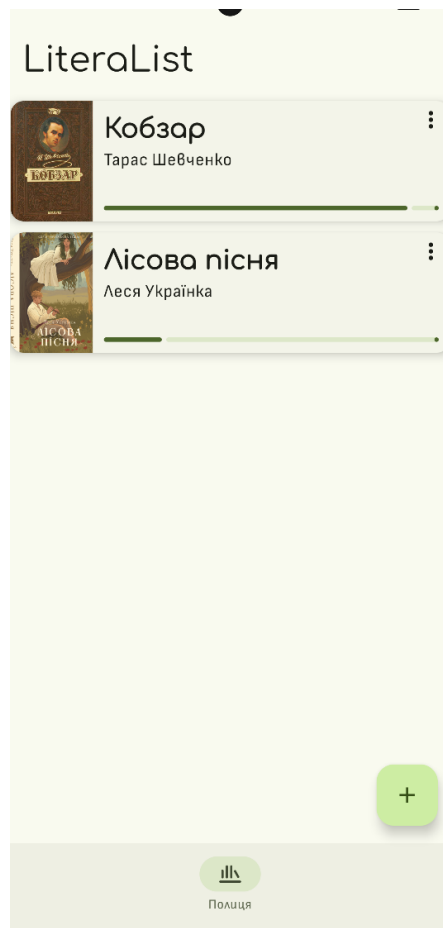


Рис. 3.7 – Екран книжкової полиці з книгами

Після додавання книжок, на екрані книжкової полиці (рис. 3.7) відображаються картки книжок, які містять обкладинку, назву книжки та їх автори. Також на картці присутня смужка прогресу, яка відображає кількість сторінок, які прочитав користувач. Вгорі справа картки знаходиться кнопка меню, яка відкриває опцію видалення книги.

Екран книги складається із двох вкладок. На вкладці нотаток (рис. 3.8) користувач за допомогою слайдера може поставити оцінку від 1 до 10. Нижче відслідковується кількість прочитаних сторінок, користувач може увести потрібне число за допомогою клавіатури, або використовувати кнопки «+» та «-» для відповідно збільшення та зменшення значення на 1. Також користувачу

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

надається вільне поле для нотаток, яке користувач може використовувати як забажає: написання рецензії, відгуку, закладок, цитат. Зміни не будуть збережені поки користувач не натисне кнопку «Зберегти».

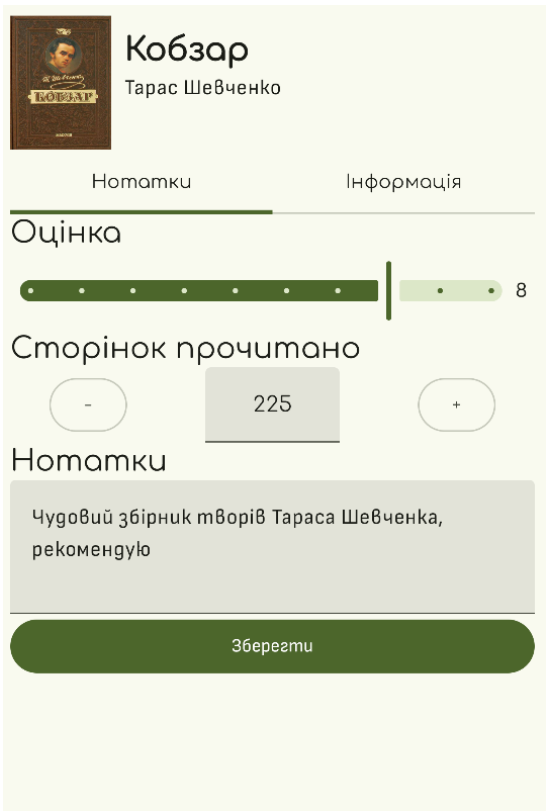


Рис. 3.8 – Вкладка нотаток екрану книги

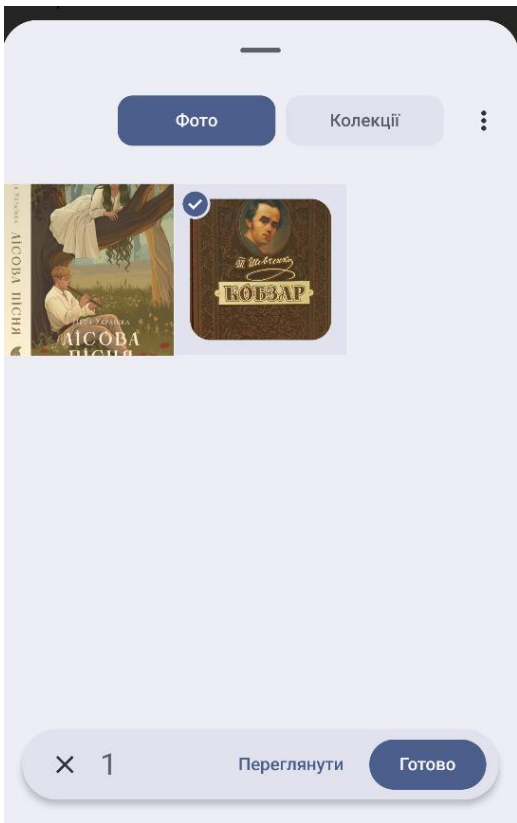
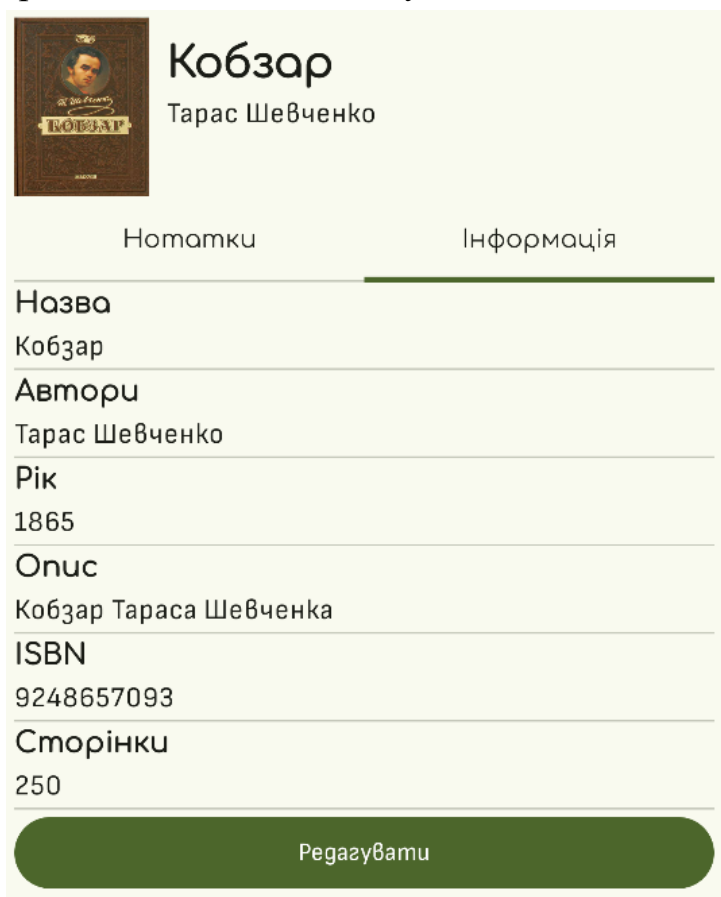
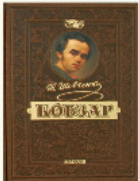


Рис. 3.9 – Вікно вибору фотографії для обкладинки

При натисканні на обкладинку книги, відкривається вікно вибору фотографії з пам'яті телефона (рис. 3.9). Для обкладинки дозволяється обрати лише одну фотографію. Коли користувач натискає на фотографію, в нього є можливість попереднього перегляду фотографії. Кнопка «Готово» підтверджує вибір та оновлює обкладинку книги.





Кобзар

Тарас Шевченко

Нотатки

Інформація

Назва

Кобзар

Автори

Тарас Шевченко

Рік

1865

Опис

Кобзар Тараса Шевченка

ISBN

9248657093

Сторінки

250

Редагувати

Рис. 3.10 – Вкладка інформації про книгу

Вкладка інформації про книгу (рис. 3.10) містить всі дані про книгу, таку як назва, автори, рік видання, опис, ISBN та загальну кількість сторінок. Кнопка «Редагувати» дозволяє перейти на екран редагування інформації про книгу.

3.3 Тестування роботи мобільного додатку

Для тестування додатку розглянемо основний сценарій використання додатку: користувач натискає кнопку для додавання книги, вводить ISBN код видання, натискає пошук, всі поля книги автоматично заповнюються,

користувач натискає «Зберегти», натискає на картку книги, змінює обкладинку, натискає меню картки та видаляє книгу.

Для спостереження змін стану додатка, використовуватимуться такі вбудовані інструменти Android Studio як App Inspector та Device Explorer [30]. Першим кроком натискаємо додати книгу, вводимо ISBN код та здійснюємо пошук. Через декілька секунд, додаток автоматично заповнив решту полів, що відповідає очікуваній поведінці (рис. 3.11).

Назва

Тарас Шевченко Кобзар

Автори

Тарас Шевченко

Рік

Опис

«Тарас Шевченко. Вибрана поезія» — видатна збірка творів, що промовляє до серця кожного українця. Це добірка безсмертних рядків із «Кобзаря», які об'єднують покоління, надихають боротися, любити, вірити. Шевченко — символ волі, правди й національного духу. Його слово — пророче, сильне, емоційне, воно актуальне сьогодні як ніколи. Його поезія — це не тільки література, а глибока розмова з народом, заклик до гії, до самопізнання й любові до Батьківщини. Книга входить до нової серії «Класика української літератури», створеної для учнів, студентів, педагогів, шанувальників української культури та всіх, хто хоче глибше зрозуміти силу рідного слова.

Сторінки

322

ISBN

9786178738013

Рис. 3.11 – Результат пошуку книги за ISBN

Якщо подивитися у вкладку Network Inspector, можна побачити, що додаток успішно здійснив запит до Google Books API та отримав відповідь за 899 мс. зі статусом кодом 200 (рис. 3.12).

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

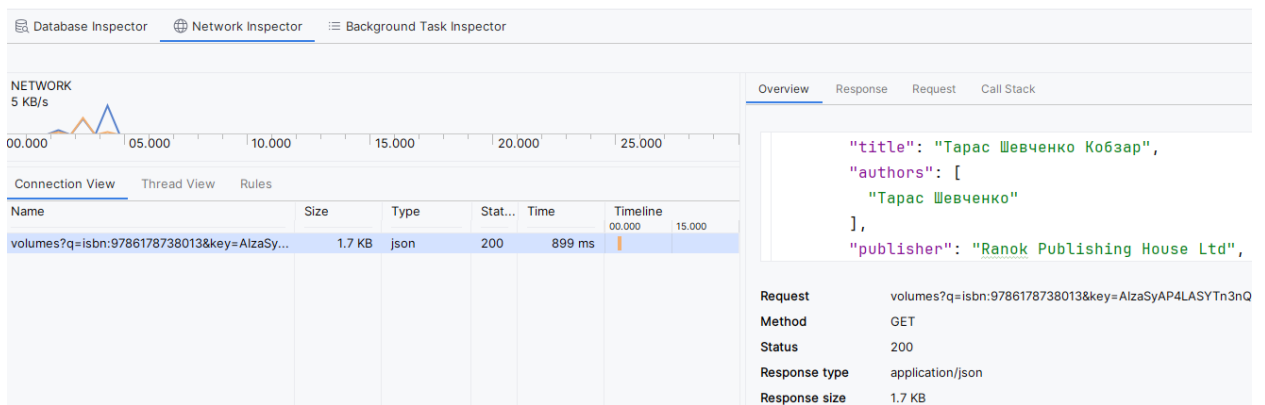


Рис. 3.12 – Результат запиту до Google Books API

Редагуємо некоректну назву книги, та зберігаємо зміни. Тепер перейдемо до екрану книги, щоб перевірити, чи правильно копіюються, замінюються та видаляються файли обкладинки (рис. 3.13).

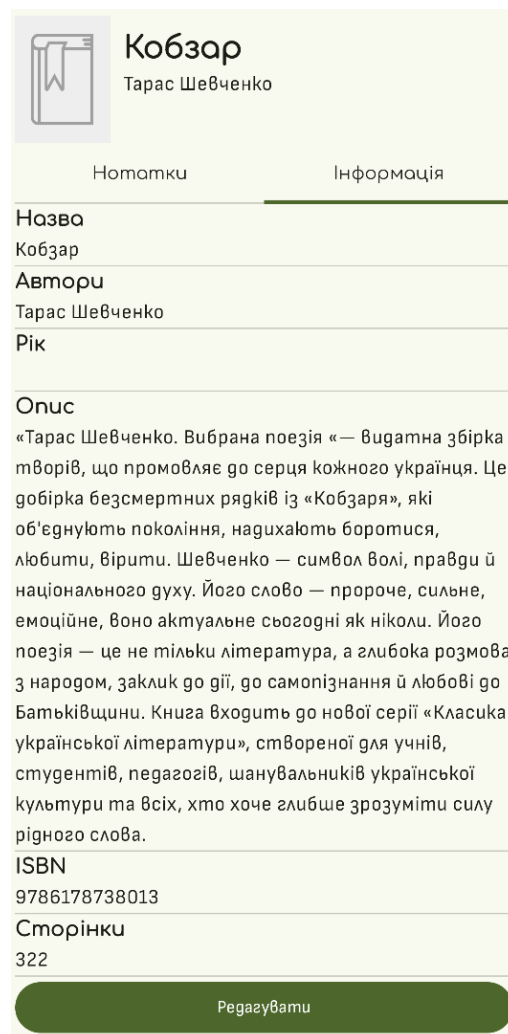


Рис. 3.13 – Додана нова книга без обкладинки

До додавання обкладинки, внутрішня папка додатка не містить фотографій (рис. 3.14).

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

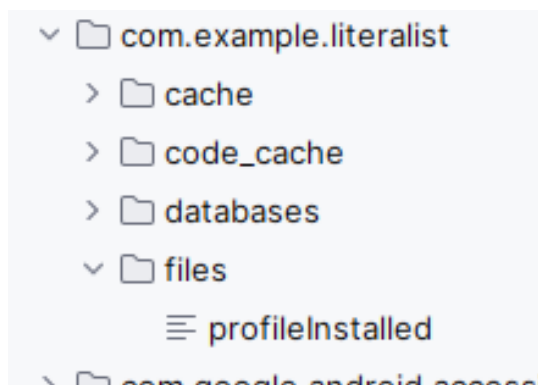


Рис. 3.14 – Внутрішня папка додатка

Після встановлення обкладинки бачимо, що файл успішно скопіювався та міститься в підпапці files (рис. 3.15).

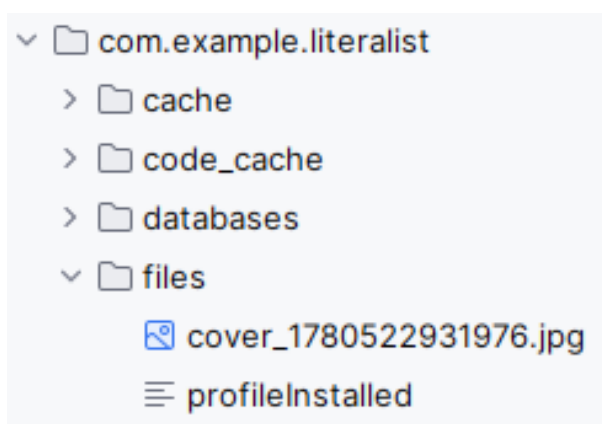


Рис. 3.15 – Створений файл обкладинки

Спробуємо змінити обкладинку на іншу. Нова обкладинка закріпилася за книгою, стара обкладинка видалилась і її замінила нова (рис. 3.16).

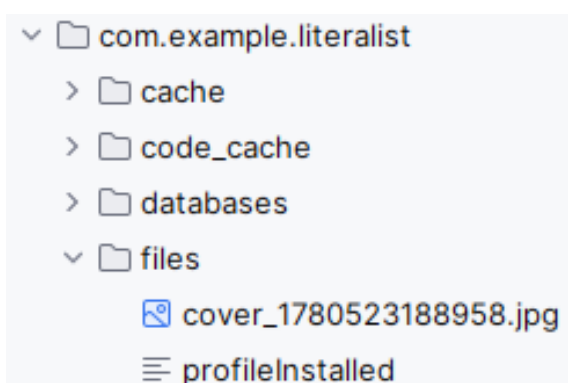


Рис. 3.16 – Нова обкладинка замінила стару

Після видалення книги, підпапка files більше не містить зображень, що підтверджує видалення обкладинки разом з книгою. Таким чином ми пересвідчилися, що додаток не буде непередбачувано збільшуватись в розмірі в процесі заміни обкладинок чи видалення книг.

Висновки до розділу 3

На початку були визначені мінімальні та рекомендовані системні вимоги, що гарантують коректну роботу додатку на великій різноманітності пристроїв Android. Зокрема, розглянуто процедуру надання дозволів на встановлення додатків із невідомих джерел для застосування інсталяційного APK пакета.

Далі було розглянуто структуру інтерфейсу мобільного додатка. Він складається з екрану книжкової полиці, екрану книги та екрану редагування інформації про книгу. На екрані книжкової полиці відображаються картки книг, які містять обкладинку книги, назву авторів та індикатор прогресу читання. Екран редагування містить поля для заповнення інформації про книгу та кнопку для пошуку книги за ISBN. Екран книги складається з двох вкладок: «нотатки» та «інформація». Також на цьому екрані можна змінити обкладинку книги натиснувши на неї, та обравши з фотографій на пристрої.

Було проведено тестування функціоналу додатка за допомогою вбудованих інструментів Android Studio. Підтверджено роботу інтеграції Google Books API, із задовільним часом відгуку. Додаток успішно реалізував процеси створення, заміни та видалення файлів обкладинок. Таким чином, комплексний аналіз та тестування дозволили переконатися у відповідності очікуваній поведінці та зручності використання розробленого програмного продукту «LiteraList».

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

ВИСНОВКИ

У ході виконання випускної кваліфікаційної роботи бакалавра було повністю досягнуто поставленої мети, що полягала у проектуванні та програмній імплементації мобільного застосунку для автоматизації процесів обліку персональної бібліотеки. За результатами розробки отримано програмний продукт, який реалізує повний цикл керування книжковим каталогом, забезпечуючи користувачеві можливість додавання нових видань, редагування існуючих записів та видалення застарілих карток книг. Ключовим технологічним результатом стала інтеграція із зовнішнім сервісом Google Books API, що дозволило впровадити функцію автоматичного наповнення метаданих за міжнародним номером ISBN, суттєво знижуючи обсяг ручного введення та мінімізуючи ймовірність помилок. Також у застосунку успішно реалізовано механізми моніторингу читацької активності через динамічну візуалізацію прогресу та надійну систему локального керування графічними обкладинками в ізольованій папці додатку.

Вибір сучасного технологічного стеку, що базується на мові програмування Kotlin, декларативному фреймворку Jetpack Compose та архітектурному патерні MVVM, забезпечив високу продуктивність, стабільність та гнучкість розробленої системи. Використання бібліотеки Room для локального збереження даних дозволило реалізувати повноцінне функціонування програми в офлайн-режимі, а впровадження фреймворку Hilt для ін'єкції залежностей суттєво підвищило якість коду та полегшило процес його тестування. Комплексне тестування функціоналу, проведене за допомогою інструментів Android Studio, підтвердило повну відповідність розробленого прототипу визначеним технічним вимогам, продемонструвавши стабільну роботу мережевої взаємодії та ефективність алгоритмів обробки даних.

Практична цінність отриманих результатів полягає у створенні готового мобільного інструменту, який вирішує систематизацію домашніх книжкових колекцій та надає зручний інтерфейс для управління персональним

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

інтелектуальним простором. Подальші напрямки вдосконалення програмного комплексу можуть бути пов'язані із впровадженням хмарної синхронізації даних між декількома пристроями користувача, розробкою рекомендаційної системи на основі аналізу читацьких вподобань та інтеграцією соціальних функцій для обміну відгуками про прочитану літературу. Розширення аналітичних можливостей застосунку в майбутньому дозволить забезпечити глибшу візуалізацію статистики читання та підвищити рівень залученості користувачів.

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Анастасія Шуренкова, Ольга Прочуханова «Зміцнення позиції української книги». Київ, 2024. – С.11. – URL: https://drive.google.com/file/d/1Jrd2Omb1QepBKiwnNy4-fQ_Q7_zt6QKz/view.
2. Goodreads. URL: <https://www.goodreads.com/> (дата звернення: 04.06.2026).
3. The StoryGraph. URL: <https://thestorygraph.com/> (дата звернення: 04.06.2026).
4. Google Play. Bookly. URL: <https://play.google.com/store/apps/details?id=com.twodoor.bookly&hl=uk> (дата звернення: 04.06.2026).
5. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
6. Burlikowski T. Architectural Patterns in Mobile Development: Implementing MVP, MVVM, and MVI. Birmingham : Packt Publishing, 2022. 312 p.
7. Smith J. Advanced MVVM Architecture in Modern Application Development. New York : Apress, 2023. 284 p.
8. Escoffier C. Reactive Systems in Java: Resilient, Event-Driven Architecture with Quarkus. Sebastopol : O'Reilly Media, 2021. 320 p.
9. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
10. Android Developers. Android Studio. URL: <https://developer.android.com/studio> (дата звернення: 04.06.2026).
11. Kotlin. URL: <https://kotlinlang.org/> (дата звернення: 04.06.2026).
12. Медведюк Д., Ерам. Використання Jetpack Compose у сучасній Android-розробці. Ерам. 30.05.2023. URL: <https://careers.epam.ua/blog/jetpack-compose-in-android-development> (дата звернення: 04.06.2026).
13. Material Design 3. URL: <https://m3.material.io/> (дата звернення: 04.06.2026).

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

14. Android Developers. Save data in a local database using Room. URL: <https://developer.android.com/training/data-storage/room> (дата звернення: 04.06.2026).
15. SQLite. URL: <https://sqlite.org/> (дата звернення: 04.06.2026).
16. Miles R., Hamilton K. Learning UML 2.0. Sebastopol : O'Reilly Media, 2006. 269 p.
17. Державна наукова установа "Книжкова палата України імені Івана Федорова". URL: <http://www.ukrbook.net/agentstvo.html> (дата звернення: 04.06.2026).
18. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 611 p.
19. Alur D., Crupi J., Malks D. Core J2EE Patterns: Best Practices and Design Strategies. 2nd ed. Upper Saddle River : Prentice Hall, 2003. 680 p.
20. Android Developers. Kotlin Flows. URL: <https://developer.android.com/kotlin/flow> (дата звернення: 04.06.2026).
21. Developers Google. Google Books API. URL: <https://developers.google.com/books/docs/overview> (дата звернення: 04.06.2026).
22. Developers Android. Dependency injection with Hilt. URL: <https://developer.android.com/training/dependency-injection/hilt-android> (дата звернення: 04.06.2026).
23. Retrofit. URL: <https://square.github.io/retrofit/> (дата звернення: 04.06.2026).
24. Refactoring Guru. Observer. URL: <https://refactoring.guru/uk/design-patterns/observer> (дата звернення: 04.06.2026).
25. Developers Android. Navigation. URL: <https://developer.android.com/guide/navigation> (дата звернення: 04.06.2026).

26. Android Developers. Compose UI Architecture. URL: <https://developer.android.com/develop/ui/compose/architecture> (дата звернення: 04.06.2026).
27. Material Foundation. Material Theme Builder. URL: <https://material-foundation.github.io/material-theme-builder/> (дата звернення: 04.06.2026).
28. Android Developer. API level. URL: <https://developer.android.com/guide/topics/manifest/uses-sdk-element> (дата звернення: 04.06.2026).
29. Fileformat. APK. URL: <https://docs.fileformat.com/uk/compression/apk/> (дата звернення: 04.06.2026).
30. Android Developer. Inspect network traffic with the Network Inspector. URL: <https://developer.android.com/studio/debug/network-profiler> (дата звернення: 04.06.2026).

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

ДОДАТКИ

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

GoogleBooksApiService.kt

```

interface GoogleBooksApiService {
    @GET("volumes")
    suspend fun searchByIsbn(@Query("q") query: String): GoogleBooksResponse
}

@Serializable
data class GoogleBooksResponse(
    val items: List<BookItem>? = null
)

@Serializable
data class BookItem(
    val volumeInfo: VolumeInfo
)

@Serializable
data class VolumeInfo(
    val title: String? = null,
    val authors: List<String>? = null,
    val publishedDate: String? = null,
    val description: String? = null,
    val pageCount: Int? = null,
    val imageLinks: ImageLinks? = null
)

@Serializable
data class ImageLinks(
    val thumbnail: String? = null
)

```

BookScreen.kt

```

@Composable
fun BookScreen(
    onBookEdit: (Int) -> Unit,
    onChangesSave: () -> Unit,
    modifier: Modifier = Modifier,
    viewModel: BookViewModel = hiltViewModel()
) {
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()
    var selectedTabIndex by rememberSaveable { mutableIntStateOf(0) }
    val titles = listOf(stringResource(R.string.notes),
stringResource(R.string.info))
    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = 10.dp)
    ) {
        TopInformation(
            title = uiState.title,
            author = uiState.author,
            coverUri = uiState.coverUri,
            onCoverClick = { viewModel.updateCoverUri(it) },
            modifier = Modifier
                .height(100.dp)
                .fillMaxWidth()
        )
        SecondaryTabRow(selectedTabIndex = selectedTabIndex) {
            titles.forEachIndexed { index, title ->
                Tab(
                    selected = selectedTabIndex == index,
                    onClick = { selectedTabIndex = index },
                    text = { Text(title) }
                )
            }
        }

        when (selectedTabIndex) {
            0 -> NotesTab(
                uiState = uiState,
                onReadPagesUpdate = { viewModel.updateReadPages(it) },
                onRatingUpdate = { viewModel.updateRating(it) },
                onReviewUpdate = { viewModel.updateReview(it) },
                onSaveClick = { viewModel.saveNotes { onChangesSave() } },
                modifier = Modifier.fillMaxWidth()
            )
            1 -> InfoTab(
                uiState = uiState,
                onEditClick = { onBookEdit(it) }
            )
        }
    }
}

@Composable
fun NotesTab(
    uiState: BookUiState,
    onRatingUpdate: (Int) -> Unit,
    onReadPagesUpdate: (String) -> Unit,
    onReviewUpdate: (String) -> Unit,
    onSaveClick: () -> Unit,

```

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

```

        modifier: Modifier = Modifier
    ) {
        Column(modifier = modifier.verticalScroll(rememberScrollState())) {
            Text(text = stringResource(R.string.rating), style =
MaterialTheme.typography.titleLarge)
            Row(verticalAlignment = Alignment.CenterVertically, modifier =
Modifier.padding(5.dp)) {
                Slider(
                    value = uiState.rating.toFloat(),
                    onChange = { onRatingUpdate(it.toInt()) },
                    steps = 8,
                    valueRange = 1f..10f,
                    modifier = Modifier.weight(1f)
                )
                Spacer(modifier = Modifier.size(8.dp))
                Text(text = uiState.rating.toString())
            }

            Text(
                text = stringResource(R.string.read_pages),
                style = MaterialTheme.typography.titleLarge
            )
            Row(
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.SpaceAround,
                modifier = Modifier.fillMaxWidth()
            ) {
                OutlinedButton(
                    onClick = { onReadPagesUpdate((uiState.readPages -
1).toString()) },
                    enabled = uiState.readPages > 0
                ) {
                    Text(text = "-")
                }
                TextField(
                    value = uiState.readPages.toString(),
                    onChange = { onReadPagesUpdate(it) },
                    modifier = Modifier.width(100.dp),
                    keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
                    singleLine = true,
                    textStyle =
MaterialTheme.typography.titleMedium.copy(textAlign = TextAlign.Center),
                )
                OutlinedButton(
                    onClick = { onReadPagesUpdate((uiState.readPages +
1).toString()) },
                    enabled = uiState.readPages < uiState.totalPages
                ) {
                    Text(text = "+")
                }
            }

            Text(text = stringResource(R.string.notes), style =
MaterialTheme.typography.titleLarge)
            TextField(
                value = uiState.review,
                onChange = { onReviewUpdate(it) },
                modifier = Modifier
                    .fillMaxWidth()
                    .heightIn(min = 100.dp)
            )
            Button(onClick = { onSaveClick() }, modifier =
Modifier.fillMaxWidth()) {
                Text(text = stringResource(R.string.save))
            }
        }
    }

```

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Єфремов Ю. М.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

@Composable
fun TopInformation(
    title: String,
    author: String,
    modifier: Modifier = Modifier,
    coverUri: String? = null,
    onCoverClick: (Uri) -> Unit
) {
    val photoPickerLauncher =

rememberLauncherForActivityResult(ActivityResultContracts.PickVisualMedia())
{ uri ->
    uri?.let { onCoverClick(it) }
}
    Row(modifier = modifier) {
        if (coverUri == null) {
            Image(
                painter = painterResource(R.drawable.book_placeholder),
                contentDescription = null,
                modifier = Modifier
                    .width(75.dp)
                    .clickable {
                        photoPickerLauncher.launch(
                            PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageOnly)
                        )
                    },
                contentScale = ContentScale.FillBounds
            )
        } else {
            AsyncImage(
                model = coverUri,
                contentDescription = null,
                placeholder = painterResource(R.drawable.book_placeholder),
                modifier = Modifier
                    .width(75.dp)
                    .clickable {
                        photoPickerLauncher.launch(
                            PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageOnly)
                        )
                    },
                contentScale = ContentScale.FillBounds
            )
        }
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(10.dp),
            verticalArrangement = Arrangement.SpaceBetween,
        ) {
            Column {
                Text(
                    text = title,
                    style = MaterialTheme.typography.headlineSmall.copy(
                        fontWeight = FontWeight.Bold
                    ),
                    overflow = TextOverflow.Ellipsis,
                    maxLines = 1
                )
            }
        }
    }
}

```

```

        )
        Text(
            text = author,
            style = MaterialTheme.typography.bodyLarge,
            overflow = TextOverflow.Ellipsis,
            maxLines = 2
        )
    }
}

}

}

@Composable
fun InfoTab(
    uiState: BookUiState,
    onEditClick: (Int) -> Unit,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier
            .fillMaxWidth()
            .verticalScroll(rememberScrollState())
    ) {
        InfoTabItem(title = stringResource(R.string.title), text =
        uiState.title)
        InfoTabItem(title = stringResource(R.string.authors), text =
        uiState.author)
        InfoTabItem(title = stringResource(R.string.year), text =
        uiState.year)
        InfoTabItem(title = stringResource(R.string.description), text =
        uiState.description)
        InfoTabItem(title = stringResource(R.string.isbn), text =
        uiState.isbn)
        InfoTabItem(title = stringResource(R.string.pages), text =
        uiState.totalPages.toString())
        Button(onClick = { onEditClick(uiState.id) }, modifier =
        Modifier.fillMaxWidth()) {
            Text(text = stringResource(R.string.edit))
        }
    }
}

@Composable
fun InfoTabItem(title: String, text: String, modifier: Modifier = Modifier) {
    Column(modifier = modifier.fillMaxWidth()) {
        Text(
            text = title,
            style = MaterialTheme.typography.titleMedium.copy(fontWeight =
            FontWeight.Bold)
        )
        Text(text = text)
        HorizontalDivider()
    }
}

```

EditBookScreen.kt

```

@Composable
fun EditBookScreen(
    modifier: Modifier = Modifier,
    onBookSave: () -> Unit,
    viewModel: EditBookViewModel = hiltViewModel()
) {
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()

    EditBookBody(
        uiState,
        modifier = modifier,
        onTitleUpdate = { viewModel.updateTitle(it) },
        onAuthorUpdate = { viewModel.updateAuthor(it) },
        onYearUpdate = { viewModel.updateYear(it) },
        onDescriptionUpdate = { viewModel.updateDescription(it) },
        onPagesUpdate = { viewModel.updatePages(it) },
        onIsbnUpdate = { viewModel.updateIsbn(it) },
        onIsbnSearch = { viewModel.searchByIsbn() }
    ) { viewModel.saveBook { onBookSave() } }
}

@Composable
private fun EditBookBody(
    uiState: BookUiState,
    modifier: Modifier = Modifier,
    onTitleUpdate: (String) -> Unit = {},
    onAuthorUpdate: (String) -> Unit = {},
    onYearUpdate: (String) -> Unit = {},
    onDescriptionUpdate: (String) -> Unit = {},
    onPagesUpdate: (String) -> Unit = {},
    onIsbnUpdate: (String) -> Unit = {},
    onIsbnSearch: () -> Unit = {},
    onSaveClick: () -> Unit = {}
) {
    val scrollState = rememberScrollState()
    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(16.dp)
            .verticalScroll(scrollState)
    ) {
        OutlinedTextField(
            value = uiState.title,
            onValueChange = { onTitleUpdate(it) },
            label = { Text(stringResource(R.string.title)) },
            isError = uiState.title.isBlank(),
            singleLine = true,
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(12.dp))

        OutlinedTextField(
            value = uiState.author,
            onValueChange = { onAuthorUpdate(it) },
            label = { Text(stringResource(R.string.authors)) },
            isError = uiState.author.isBlank(),
            singleLine = true,
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(12.dp))
    }
}

```

		Задніпрянець А. А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

        OutlinedTextField(
            value = uiState.year,
            onChange = { onYearUpdate(it) },
            label = { Text(stringResource(R.string.year)) },
            singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(12.dp))

        OutlinedTextField(
            value = uiState.description,
            onChange = { onDescriptionUpdate(it) },
            label = { Text(stringResource(R.string.description)) },
            modifier = Modifier
                .fillMaxWidth()
                .heightIn(min = 100.dp)
        )
        Spacer(modifier = Modifier.height(12.dp))

        OutlinedTextField(
            value = uiState.totalPages.toString(),
            onChange = { onPagesUpdate(it) },
            label = { Text(stringResource(R.string.pages)) },
            keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
            singleLine = true,
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(12.dp))

        OutlinedTextField(
            value = uiState.isbn,
            onChange = { onIsbnUpdate(it) },
            label = { Text(stringResource(R.string.isbn)) },
            keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
            singleLine = true,
            trailingIcon = {
                IconButton(onClick = { onIsbnSearch() }) {
                    Icon(
                        painter =
painterResource(R.drawable.outline_search_24),
                        contentDescription = null
                    )
                }
            },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(12.dp))

        Button(
            onClick = { onSaveClick() },
            enabled = uiState.title.isNotBlank() &&
uiState.author.isNotBlank(),
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(text = stringResource(R.string.save))
        }
    }
}

```

BookShelfScreen.kt

```

@Composable
fun BookShelfScreen(
    onBookCardClick: (Int) -> Unit,
    modifier: Modifier = Modifier,
    viewModel: BookShelfViewModel = hiltViewModel()
) {
    val booksList by viewModel.books.collectAsStateWithLifecycle()
    if (booksList.isEmpty()) {
        Box(modifier = modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
            Text(
                text = stringResource(R.string.empty_library_message),
                style = MaterialTheme.typography.titleLarge,
            )
        }
    } else {
        LazyColumn(
            modifier = modifier
                .fillMaxSize()
        ) {
            items(booksList) { book ->
                BookCard(
                    book = book,
                    onBookCardClick = onBookCardClick,
                    onBookDeleteClick = { viewModel.deleteBook(book) },
                    modifier = Modifier
                        .height(120.dp)
                        .padding(5.dp)
                )
            }
        }
    }
}

@Composable
fun BookCard(
    book: Book,
    onBookCardClick: (Int) -> Unit,
    onBookDeleteClick: (Book) -> Unit,
    modifier: Modifier = Modifier
) {
    var isMenuExpanded by remember { mutableStateOf(false) }
    ElevatedCard(
        modifier = modifier,
        elevation = CardDefaults.cardElevation(5.dp)
    ) {
        Box(contentAlignment = Alignment.TopEnd) {
            Row(
                modifier = Modifier
                    .fillMaxSize()
                    .clickable(onClick = { onBookCardClick(book.id) })
            ) {
                if (book.coverUri != null) {
                    AsyncImage(
                        model = book.coverUri,
                        contentDescription = null,
                        placeholder =
painterResource(R.drawable.book_placeholder),
                        modifier = Modifier
                            .width(75.dp),

```

```

        contentScale = ContentScale.Crop
    )
} else {
    Image(
        painter = painterResource(id =
R.drawable.book_placeholder),
        contentDescription = null,
        modifier = Modifier.width(75.dp),
        contentScale = ContentScale.Crop
    )
}
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(10.dp),
    verticalArrangement = Arrangement.SpaceBetween,
) {
    Column {
        Text(
            text = book.title,
            style =
MaterialTheme.typography.headlineSmall.copy(
                fontWeight = FontWeight.Bold
            ),
            overflow = TextOverflow.Ellipsis,
            maxLines = 1
        )
        Text(
            text = book.author,
            style = MaterialTheme.typography.bodyLarge,
            overflow = TextOverflow.Ellipsis,
            maxLines = 2
        )
    }
    LinearProgressIndicator(
        progress = {
            if (book.totalPages > 0) {
                book.readPages.toFloat() / book.totalPages
            } else {
                0f
            }
        },
        modifier = Modifier.fillMaxWidth()
    )
}
}
Box(Modifier.padding(5.dp)) {
    Icon(
        painter =
painterResource(R.drawable.outline_more_vert_24),
        contentDescription = null,
        modifier = Modifier
            .clickable(onClick = { isMenuExpanded =
!isMenuExpanded })
    )
    DropdownMenu(
        expanded = isMenuExpanded,
        onDismissRequest = { isMenuExpanded = false }
    ) {
        DropdownMenuItem(
            { Text(text = stringResource(R.string.delete)) },
            onClick = {

```

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

                                onBookDeleteClick(book)
                                isMenuExpanded = false
                            }
                        )
                    }
                }
            }
        }

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TopAppBar(modifier: Modifier = Modifier) {
    TopAppBar(
        modifier = modifier,
        title = {
            Text(
                text = stringResource(R.string.app_name),
                style = MaterialTheme.typography.headlineLarge
            )
        },
    )
}

@Composable
fun BottomAppBar(onNavItemClick: () -> Unit, modifier: Modifier = Modifier) {
    NavigationBar(modifier = modifier) {
        NavigationBarItem(
            selected = true,
            onClick = onNavItemClick,
            icon = {
                Icon(
                    painterResource(R.drawable.ic_bookshelf),
                    contentDescription = null,
                )
            },
            label = { Text(stringResource(R.string.bookshelf)) }
        )
    }
}

@Composable
fun AddBookFab(onAddBookClick: () -> Unit) {
    FloatingActionButton(onClick = onAddBookClick) {
        Icon(
            painterResource(R.drawable.outline_add_24),
            contentDescription = null
        )
    }
}

```

Navigation.kt

```

@Composable
fun LiteralList(
    modifier: Modifier = Modifier,
    navController: NavHostController = rememberNavController(),
) {
    NavHost(
        navController = navController,
        startDestination = BookShelfScreen,
        modifier = modifier
    ) {
        composable<BookShelfScreen> {
            Scaffold(
                topBar = { TopAppBar() },
                bottomBar = {
                    BottomAppBar(onNavItemClick = {})
                },
                floatingActionButton = {
                    AddBookFab(onAddBookClick = {
                        navController.navigate(EditBookScreen()) })
                },
                modifier = Modifier.fillMaxSize()
            ) { innerPadding ->
                BookShelfScreen(
                    modifier = Modifier.padding(innerPadding),
                    onBookCardClick = {
                        navController.navigate(BookScreen(it)) }
                )
            }
        }

        composable<EditBookScreen> {
            Scaffold { innerPadding ->
                EditBookScreen(
                    onBookSave = { navController.popBackStack() },
                    modifier = Modifier.padding(innerPadding)
                )
            }
        }

        composable<BookScreen> {
            Scaffold { innerPadding ->
                BookScreen(
                    onBookEdit = { navController.navigate(EditBookScreen(it)) },
                    onChangesSave = { navController.popBackStack() },
                    modifier = Modifier.padding(innerPadding)
                )
            }
        }
    }
}

@Serializable
object BookShelfScreen

@Serializable
data class BookScreen(val id: Int)

@Serializable
data class EditBookScreen(val id: Int? = null)

```

		Задніпрянець А. А.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Єфремов Ю. М.				78
Змн.	Арк.	№ докум.	Підпис	Дата		