МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «**Проектування та розробка мобільного додатку читання електронних книг з розширеним функціоналом»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»

Максимчук Даниїл Сергійович

Керівник: викладач кафедри інформаційних технологій та аналітики даних
Місай Володимир Віталійович

Рецензент: кандидат технічних наук, доцент,
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних

професор, доктор економічних наук, Кривицька Ольга Романівна

Протокол № 11 від 20 травня 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: *Проектування та розробка мобільного додатку читання електронних книг з розширеним функціоналом*

Автор: *Максимчук Даниїл Сергійович*

Керівник: *Місай Володимир Віталійович*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 50 с., 23 рис., 1 табл., 5 додатків, 19 джерел.

Ключові слова: *мобільний застосунок, React Native, електронна книга, PDF, TXT, аналітика читання, персоналізація інтерфейсу, JavaScript, мобільна розробка.*

Короткий зміст праці: Кваліфікаційна робота присвячена розробці кросплатформного мобільного застосунку для читання електронних книг у форматах TXT та PDF. Метою роботи є створення програмного продукту, що поєднує базовий функціонал читання з можливостями ведення особистих нотаток, відстеження прогресу та гнучкого налаштування інтерфейсу. Об’єкт дослідження: процес розробки мобільного програмного забезпечення. Предмет дослідження: методи, архітектурні рішення та інструментальні засоби (React Native, Node.js) створення мобільного застосунку для роботи з текстовими даними. У процесі роботи проаналізовано існуючі аналоги, визначено вимоги до системи, спроектовано архітектуру та структуру бази даних. Практичним результатом є готовий застосунок із реалізованими модулями парсингу файлів, кастомізації відображення (світла/темна теми, зміна шрифтів) та збору статистики читацької активності. Результати тестування підтверджують стабільність та ефективність розробленого продукту.

ABSTRACT

of the qualification

work for a Bachelor's degree

Topic: *Design and Development of a Mobile E-book Reader Application with Advanced Features*

Author: *Danyil Serhiyovych Maksymchuk*

Advisor: *Volodymyr Vitaliyovych Misay*

Defended on "……"…… 20__.

Explanatory note to the thesis: 50 p., 23 figs., 1 table, 5 appendices, 19 sources.

Keywords: *mobile application, React Native, e-book, PDF, TXT, reading analytics, interface personalization, JavaScript, mobile development.*

Abstract: This thesis is dedicated to the development of a cross-platform mobile application for reading e-books in TXT and PDF formats. The goal of this work is to create a software product that combines basic reading functionality with the ability to take personal notes, track progress, and flexibly customize the interface. Object of study: the mobile software development process. Subject of study: methods, architectural solutions, and tools (React Native, Node.js) for creating a mobile application to work with text data. During the project, existing analogues were analyzed, system requirements were defined, and the architecture and database structure were designed. The practical result is a finished application with implemented modules for file parsing, display customization (light/dark themes, font changes), and collection of reader activity statistics. Testing results confirm the stability and effectiveness of the developed product.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі мобільні пристрої стали основним інструментом для споживання цифрового контенту. Популярність електронних книг постійно зростає завдяки їхній доступності та зручності. Проте більшість існуючих мобільних читалок пропонують лише базовий функціонал відображення тексту. Користувачі все частіше потребують інструментів, які дозволяють не лише читати, а й аналізувати власну активність, фіксувати важливі думки через нотатки та повністю адаптувати інтерфейс під власні фізіологічні потреби (зір, умови освітлення). Таким чином, розробка мобільного застосунку з розширеним функціоналом персоналізації та аналітики є актуальним завданням для покращення читацького досвіду.

Мета роботи — проектування та програмна реалізація кросплатформного мобільного застосунку для читання книг у форматах TXT та PDF, що забезпечує користувача розширеними можливостями аналітики та кастомізації.

Об’єкт дослідження — процес розробки мобільного програмного забезпечення для операційних систем iOS та Android.

Предмет дослідження — методи, архітектурні рішення та програмні засоби розробки мобільного застосунку для роботи з електронними книгами.

Методи дослідження. Для розв’язання поставлених завдань використано методи системного аналізу (для дослідження аналогів), об’єктно-орієнтованого проектування (для побудови архітектури) та методи гнучкої розробки програмного забезпечення (для реалізації продукту).

Практичне значення отриманих результатів полягає у створенні функціонального мобільного застосунку, який може бути використаний як для особистого читання, так і в освітніх цілях для підвищення ефективності роботи з текстовою інформацією.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ СТВОРЕННЯ ДОДАТКУ ЧИТАННЯ КНИГ

1.1. Аналіз предметної області та актуальність розробки

Процес цифровізації сучасної освіти та дозвілля призвів до того, що паперові носії інформації поступово витісняються електронними форматами. Предметна область даного дослідження охоплює мобільні технології обробки текстової та графічної інформації, що зберігається у форматах електронних книг.

Актуальність розробки зумовлена декількома ключовими факторами:

1. **Зміна парадигми читання:** Сучасний користувач перейшов від лінійного читання до "скануючого", що потребує інструментів для швидкого пошуку, створення закладок та миттєвого доступу до нотаток.
2. **Проблема концентрації:** В умовах надлишку інформації та постійних сповіщень смартфонів, застосунки для читання мають створювати ізольоване середовище («режим занурення»), де ніщо не відволікає від тексту.
3. **Потреба у самоконтролі:** Концепція "Quantified Self" (вимірювання власних показників) стала популярною в усіх сферах життя. Користувачі хочуть бачити об’єктивні дані: скільки сторінок прочитано, скільки часу витрачено на навчання, яка швидкість засвоєння матеріалу.

У межах даної роботи під "розширеним функціоналом" мається на увазі не лише рендеринг тексту, а й інтелектуальна надбудова, що аналізує взаємодію користувача з контентом. Це дозволяє перетворити звичайний ридер на персональний аналітичний інструмент для саморозвитку.

1.2. Порівняльний аналіз програм-аналогів

1.2.1. Застосунок eBoox.

Це один із найпопулярніших ридерів завдяки підтримці форматів fb2, epub, pdf. Плюси: простий інтерфейс, відсутність реклами. Мінуси: аналітика читання представлена дуже поверхнево, неможливо відстежити, скільки часу було витрачено на конкретну главу або книгу в розрізі тижня/місяця.

1.2.2. Застосунок PocketBook Reader.

Професійний інструмент із хмарною синхронізацією. Плюси: величезна кількість налаштувань рендерингу тексту. Мінуси: інтерфейс перевантажений кнопками та меню, що створює високий поріг входження для звичайного користувача.

1.2.3. Платформа RORK.

Це скоріше соціальна мережа для читачів. Плюси: чудові таймери читання та статистика. Мінуси: це не повноцінний ридер, він часто вимагає ручного введення прочитаних сторінок, якщо ви читаєте фізичну книгу або файл в іншому додатку.

Для обґрунтування доцільності розробки нового програмного продукту було проведено порівняльний аналіз його функціональних можливостей із найближчими ринковими аналогами. Результати аналізу за основними критеріями наведено нижче.

Таблиця 1.1

Порівняльна характеристика застосунків для читання

Критерій	eBoox	PocketBook	RORK	Проектний застосунок
Підтримка PDF/TXT	+	+	—	+
Темна/світла теми	+	+	+	+
Статистика часу	частково	—	+	+
Ведення нотаток	+	+	+	+

Як видно з таблиці 1.1, розроблюваний проєктний застосунок повністю задовольняє всім визначеним критеріям порівняння, демонструючи конкурентні переваги над існуючими аналогами, зокрема в частині ведення статистики часу та універсальності підтримки форматів.

Висновок за підрозділом: Існує потреба в «гібридному» рішенні, яке б поєднувало потужний двигун для відкриття PDF/TXT файлів (як у PocketBook) із візуально приємною аналітикою (як у RORK).

1.3. Обґрунтування вибору технологій розробки

Вибір програмно-технологічного стеку є критичним етапом, оскільки він визначає не лише швидкість розробки, а й здатність застосунку коректно працювати з великими файлами (наприклад, PDF-підручниками обсягом понад 500 МБ).

1.3.1. Кросплатформний підхід із React Native.

Для реалізації проекту було обрано фреймворк React Native від компанії Meta. На відміну від суто нативної розробки (Swift для iOS чи Kotlin для Android), цей підхід дозволяє використовувати єдину бізнес-логіку для обох платформ. Це критично важливо для індивідуальної розробки, оскільки забезпечує:

- Економію часу на написання та тестування коду.
- Використання потужної екосистеми JavaScript-бібліотек.
- Високу продуктивність завдяки використанню нативних компонентів через Bridge-інтерфейс.

1.3.2. Середовище виконання Node.js.

Node.js виступає фундаментом для інструментарію розробки. Використання пакетного менеджера (npm/yarn) дозволяє інтегрувати в проект стабільні модулі для роботи з файловою системою пристрою, що є ключовим для застосунку-бібліотеки.

1.3.3. Робота з форматами даних.

Для роботи з PDF обрано спеціалізовані модулі, що використовують нативні двигуни рендерингу (наприклад, Pdfium для Android та CGPDFContext для iOS). Це гарантує, що застосунок не буде "гальмувати" при масштабуванні складних схем чи графіків у книгах. Для формату TXT реалізовано власні алгоритми розбиття тексту на сторінки (пагінації) залежно від обраного користувачем розміру шрифту.

1.3.4. Локальне збереження даних.

Для реалізації функцій аналітики та збереження нотаток без використання зовнішніх серверів (offline-first approach) обрано механізми локального кешування. Це забезпечує конфіденційність користувацьких даних та можливість роботи із застосунком у місцях без доступу до мережі Інтернет.

1.4. Формування функціональних та нефункціональних вимог

Функціональні вимоги:

1. **Модуль бібліотеки:** імпорт та відображення електронних книг у форматах TXT та PDF.
2. **Модуль читання:** підтримка пагінації, масштабування тексту, плавної прокрутки та переходу між сторінками.
3. **Модуль кастомізації:** вибір між світлою, темною та сепія-темами, зміна розміру та гарнітури шрифту, зміна кольору тексту та фону.
4. **Модуль статистики:** відображення часу читання, відсотка прогресу книги, кількості завершених книг та приблизного часу до завершення читання.
5. **Модуль календаря:** відображення читацької активності за днями з можливістю перегляду часу читання та списку книг за конкретну дату.
6. **Система закладок:** збереження точок читання з можливістю швидкого

переходу до потрібного фрагмента книги.

7. **Модуль нотаток:** створення та локальне збереження користувацьких нотаток для окремих книг.
8. Механізм автоматичного збереження прогресу читання та його відновлення після повторного відкриття книги.
9. Підтримка локального збереження даних без використання зовнішніх серверів.

Нефункціональні вимоги:

1. Швидкість відгуку інтерфейсу (не більше 200 мс на дію).
2. Економне споживання заряду акумулятора.
3. Інтуїтивно зрозумілий інтерфейс (правило трьох кліків).

ВИСНОВКИ ДО РОЗДІЛУ

У першому розділі проведено аналіз предметної області та обґрунтовано розробку мобільного застосунку для читання. За результатами дослідження сформовано такі висновки:

1. **Актуальність:** Встановлено, що сучасні користувачі потребують не просто програм для читання, а персоналізованих інструментів з елементами аналізу читацької активності, що підтверджує доцільність розробки.
2. **Аналіз аналогів:** Огляд існуючих рішень (eBoox, PocketBook Reader, RORK) виявив дефіцит продуктів, які б поєднували мінімалістичний дизайн, стабільну роботу з форматами PDF/TXT та розширену статистику без нав’язливої монетизації. Це дозволило визначити функціональну нішу проєкту.
3. **Технологічний стек:** Для реалізації обрано фреймворк React Native та платформу Node.js. Це забезпечить кросплатформність, високу швидкість роботи застосунку та легкість його подальшого масштабування.
4. **Вимоги до системи:** Визначено ключові функціональні вимоги: модулі для ведення нотаток, автоматичне відстеження прогресу читання та гнучкі налаштування візуального відображення тексту.

Таким чином, результати першого розділу стали теоретичним та технічним фундаментом для подальшого проектування й програмної реалізації системи.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ДОДАТКУ ЧИТАННЯ КНИГ

2.1. Аналіз предметної області

Виділення об’єктів дослідження. Програмна реалізація системи базується на взаємодії декількох ключових об'єктів, що складають основу інформаційного середовища застосунку:

1. **Об’єкт «Електронна книга»:** характеризується системним шляхом (**uri**) та метаданими. Це динамічний об'єкт, стан якого змінюється залежно від вибору користувача в інтерфейсі бібліотеки.
2. **Об’єкт «Стан інтерфейсу»:** агрегований об'єкт, що описує візуальні параметри додатка (тема, фонове зображення). Його життєвий цикл керується через **ThemeContext**.
3. **Об’єкт «Персональна нотатка»:** текстова сутність, що має жорстку прив’язку до ідентифікатора книги.
4. **Об’єкт «Сховище»:** локальна база даних на основі **AsyncStorage**, що забезпечує персистентність (збереження після закриття) усіх налаштувань.
5. **Об’єкт «Закладка»:** структура даних, що містить інформацію про позицію читання користувача в конкретній книзі. Закладка включає номер сторінки або текстовий фрагмент та використовується для швидкого переходу до місця.
6. **Об’єкт «Статистика читання»:** містить інформацію про тривалість сесій читання, загальний час роботи із книгою, прогрес читання та історію активності користувача.
7. **Об’єкт «Календар активності»:** забезпечує групування статистичних даних за датами та використовується для відображення історії читання у календарному форматі.

Побудова інформаційної моделі. Інформаційна модель побудована на реактивному управлінні станом. Вхідний потік даних починається з вибору файлу, після чого **URI** передається через систему маршрутизації (**expo-router**) до екрана рендерингу. Одночасно з цим система ініціює запит до локального сховища для отримання раніше збережених нотаток, що відповідають цьому **URI**. Модель забезпечує цілісність даних: будь-яка зміна в налаштуваннях теми моментально відображається на всіх активних екранах без переривання процесу читання.

Опис існуючих обмежень на вхідні та вихідні дані.

- **Вхідні дані:** Система підтримує PDF та TXT. Основним обмеженням для PDF є необхідність валідного заголовка файлу для коректної ініціалізації **WebView**. Для TXT-файлів існує обмеження на кодування (рекомендовано UTF-8), оскільки використання застарілих кодувань може призвести до некоректного відображення символів кирилиці.
- **Вихідні дані:** Дані зберігаються у форматі JSON-рядків. Існує апаратне обмеження обсягу **AsyncStorage** (зазвичай до 6 МБ), що диктує необхідність оптимізації зберігання великої кількості нотаток.

2.2. Проектування системи

Концептуальне проектування системи.

Концепція системи базується на патерні «Provider», що дозволяє централізовано керувати станом застосунку. Головною ідеєю є відділення контенту від форми: логіка відкриття файлу абсолютно незалежна від того, яка візуальна тема обрана користувачем. Це забезпечує високу гнучкість системи та спрощує додавання нових функцій (наприклад, підтримку нових форматів книг) без зміни глобальної архітектури.

Архітектура застосунку реалізована за модульним принципом із

використанням компонентного підходу React Native. Система складається з трьох основних рівнів:

• рівень представлення (екрани та UI-компоненти);

• рівень бізнес-логіки (обробка даних, статистики та навігації);

• рівень локального збереження даних (AsyncStorage).

Для централізованого управління станом застосунку використовується Context API, а навігація між екранами реалізована за допомогою Expo Router. Такий підхід дозволяє спростити масштабування системи та забезпечити повторне використання компонентів.

Побудова концептуальної моделі.

Абстрактна модель системи демонструє наступні причинно-наслідкові зв’язки:

- Дія користувача (натискання на книгу) → Генерація маршруту з параметром **bookUri** → Активація компонента **PdfReader** або **TxtReader**.
- Зміна перемикача в **Settings** → Оновлення об’єкта в **ThemeContext** → Автоматичний перерахунок стилів усіх компонентів через **StyleSheet.create**.
- Введення тексту в модальному вікні нотаток → Серіалізація даних → Асинхронний запис у пам'ять пристрою.

2.3. Математичне та алгоритмічне забезпечення

Моделі реалізації методів стилізації.

Математично процес вибору кольорової гами описується функцією

вибору параметрів:

$$S = \{c_1, + \ c_2, \ldots, c_n\} \quad (1)$$

де S — набір стилів;

$\{c_1, + \ c_2, \ldots, c_n\}$ — компоненти (властивості) стилю;

n — кількість властивостей.

Алгоритм працює за логікою:

$$Color = isDark \, ? \, DarkValue : LightValue \quad (2)$$

де $isDark$ — умова перевірки темної теми;

$DarkValue$ — значення для темної теми;

$LightValue$ — значення для світлої теми.

Це мінімізує обчислювальні витрати при перемиканні інтерфейсу.

Алгоритм обробки шляхів (URI).

Для забезпечення кросплатформної роботи з файлами реалізовано математичну модель трансформації рядків. Оскільки системні шляхи до файлів часто містять пробіли та специфічні символи, які не сприймаються стандартними вебресурсами, застосовується алгоритм декодування:

$$URIdecoded = decode(URIencoded) \quad (3)$$

де:

$URIdecoded$ — декодований URI;

$URIencoded$ — закодований URI;

$Decode()$ — функція декодування URI.

Це гарантує, що модуль **WebView** отримає коректне посилання для відображення документа.

Алгоритм управління локальною базою нотаток.

Логіка збереження нотаток базується на транзакційній моделі:

1. **Читання:**

$$Datacurrent = ReadStorage(Key) \quad (4)$$

де: $Datacurrent$ — поточні дані, отримані зі сховища; $ReadStorage()$ — функція читання даних; Key — ключ доступу до даних.

2. **Трансформація:**

$$List = Parse(Datacurrent) \quad (5)$$

де: $List$ — сформований список даних; $Parse()$ — функція обробки та перетворення даних; $Datacurrent$ — поточні дані, отримані зі сховища.

3. **Оновлення:**

$$Listnew = List \cup \{NewNote\} \quad (6)$$

де: $Listnew$ — оновлений список; $List$ — поточний список даних; $NewNote$ — нова нотатка; $\cup$ — операція додавання елемента до множини.

4. **Запис:**

$$WriteStorage(Key,\ Stringify(Listnew)) \qquad (7)$$

де: $WriteStorage()$ — функція запису даних у сховище; Key — ключ доступу до сховища; $Stringify()$ — функція перетворення даних у рядковий формат; $Listnew$ — оновлений список даних.

Цей цикл гарантує, що нові дані не перезаписують старі, а додаються до загального масиву інформації про книгу.

Алгоритм підрахунку часу читання

Для збору статистики читацької активності використовується механізм обчислення тривалості сесії читання.

Після відкриття книги система фіксує час початку сесії:

$StartTime = t_1$

Після завершення читання або виходу з екрана книги визначається час завершення:

$EndTime = t_2$

Загальна тривалість сесії обчислюється як:

$SessionTime = t2 - t1$

Отримані дані використовуються для побудови статистики активності користувача та відображення інформації у календарі читання.

Алгоритм збереження прогресу читання

Для забезпечення безперервності процесу читання реалізовано механізм

автоматичного збереження останньої позиції користувача. Після зміни сторінки або завершення сесії читання система записує поточний прогрес до локального сховища.

При повторному відкритті книги застосунок автоматично виконує зчитування збереженого прогресу та повертає користувача до останньої позиції.

Відсоток прочитаної книги визначається за формулою:

$$Progress = (CurrentPage \div TotalPages) \times 100\% \quad (8)$$

де:

$Progress$ — відсоток завершення читання;

$CurrentPage$ — поточна сторінка;

$TotalPages$ — загальна кількість сторінок книги.

ВИСНОВКИ ДО РОЗДІЛУ

У другому розділі було проведено детальне проєктування інформаційного та математичного забезпечення мобільного застосунку. За результатами етапу проєктування можна зробити наступні висновки:

1. Визначено основні об'єкти системи та побудовано інформаційну модель, яка базується на реактивному управлінні станом через Context API. Це дозволяє забезпечити цілісність даних при взаємодії користувача з інтерфейсом.
2. Проведено аналіз вхідних та вихідних даних, встановлено необхідні обмеження щодо форматів файлів (PDF, TXT) та методів їх серіалізації для локального збереження.
3. Розроблено концептуальну модель системи, яка демонструє чіткий розподіл функціональних обов'язків між модулями бібліотеки, читання та управління налаштуваннями.
4. Описано математичні та алгоритмічні основи функціонування додатка, зокрема алгоритми динамічної стилізації інтерфейсу, декодування ресурсів та механізм транзакційного збереження нотаток у AsyncStorage.

Побудовані моделі є достатніми та вичерпними для подальшої програмної реалізації системи, яка буде детально розглянута у наступному розділі.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ ДОДАТКУ ЧИТАННЯ КНИГ

3.1. Засоби розробки

Для програмної реалізації кваліфікаційної роботи було обрано стек технологій, що базується на середовищі **Node.js** та фреймворку **React Native**. Такий вибір зумовлений необхідністю створення кросплатформного продукту з єдиною кодовою базою.

Обґрунтування інструментарію:

React Native: дозволяє створювати мобільні інтерфейси, що використовують нативні компоненти ОС, забезпечуючи високу швидкість відгуку (60 FPS).

Expo SDK: обрано як платформу для прискореної розробки, що надає доступ до системних функцій (файлова система, іконки, збереження даних) без необхідності написання складного нативного коду на Java/Swift.

Expo Router: використовується для реалізації навігації на основі файлової структури, що є найбільш сучасним підходом у середовищі React Native. Навігація в застосунку реалізована за допомогою файлової маршрутизації, де кожен екран відповідає окремому файлу в структурі проєкту. Такий підхід спрощує підтримку застосунку та пришвидшує процес масштабування системи при додаванні нових модулів.

React Native WebView: обрано як основний рушій для відображення PDF-файлів, оскільки він дозволяє використовувати вбудовані можливості системних браузерів для рендерингу складних документів.

AsyncStorage: локальне нереляційне сховище типу «ключ-значення», обране для миттєвого доступу до налаштувань теми та списку нотаток.

AsyncStorage використовується для локального збереження:

• налаштувань теми;

• прогресу читання;

• списку закладок;

• нотаток користувача;

• статистики читацької активності.

Для подальшого масштабування проєкту передбачена можливість переходу до використання SQLite як локальної бази даних.

3.2. Вимоги до технічного та програмного забезпечення

Відповідно до стандарту OSI та принципів відкритих систем, програмне забезпечення застосунку має наступну структуру:

Системне програмне забезпечення: додаток розрахований на роботу в середовищі ОС Android (версії 7.0 "Nougat" і вище) або iOS (версії 12.0 і вище).

Інструментальне (операційне) ПЗ: для розгортання проєкту необхідне середовище Node.js (LTS версія) та встановлений клієнт Expo Go для тестування.

Функціональне (прикладне) ПЗ: розроблений пакет компонентів, що взаємодіють між собою через внутрішню шину подій React.

Технічні вимоги до пристрою:

Процесор із тактовою частотою від 1.5 ГГц.

Мінімум 2 ГБ оперативної пам'яті (для коректного рендерингу важких PDF-документів обсягом понад 100 МБ).

Наявність графічного прискорювача з підтримкою OpenGL ES 2.0.

3.3. Опис програмної реалізації

Програмна частина застосунку має чітку модульну структуру, що відповідає архітектурі Expo Router.

Основні компоненти та модулі:

ThemeContext.tsx: реалізує логіку управління станом за допомогою **createContext**. Містить методи **toggleTheme** (зміна кольорової схеми) та **updateBackgroundImage** (встановлення фону). Дані синхронізуються з **AsyncStorage**.

_layout.tsx: центральний вузол (Root Layout), який огортає всю програму провайдером теми. Тут реалізовано динамічну стилізацію верхньої панелі навігації (**Stack.Screen**) залежно від активної теми.

(tabs)/library.tsx: модуль інтерфейсу бібліотеки. Він реалізує відображення списку доступних книг. Вхідними даними є масив об'єктів книг, вихідними — подія переходу до екрана читання з передачею **bookUri**.

reader/pdf.tsx: клас-контролер для PDF. Виконує декодування шляху через **decodeURIComponent** та ініціалізує **WebView**.

reader/index.tsx: модуль для роботи з текстовими даними. Використовує компонент **ScrollView** для забезпечення плавної прокрутки великих обсягів тексту.

Для TXT-файлів реалізовано власний механізм обробки тексту, який включає:

• зчитування текстового вмісту файлу;

• розбиття тексту на сторінки;

• адаптацію відображення залежно від розміру шрифту;

• автоматичне збереження поточної позиції читання.

Це дозволяє забезпечити стабільну роботу навіть із великими текстовими файлами.

(tabs)/stats.tsx: модуль збору та відображення статистики читацької активності. Реалізує підрахунок часу читання, визначення прогресу книги та формування аналітичних показників користувача.

(tabs)/calendar.tsx: модуль календаря активності користувача. Забезпечує відображення історії читання за днями та відкриття детальної інформації про активність через модальні вікна.

Принципи функціонування методів: Кожен метод у системі є асинхронним (async/await), що запобігає блокуванню інтерфейсу (UI-потоку) під час запису нотаток або зчитування великих конфігураційних файлів із пам'яті пристрою.

3.4. Інтерфейс застосунку та керівництво користувача

Інтерфейс застосунку:

Головний екран «Моя бібліотека» (компонент LibraryScreen) призначений для візуалізації та керування локальною базою даних книг(Рис 3.1). Інтерфейс реалізовано за допомогою контейнера ImageBackground із підтримкою динамічної зміни колірних схем через ThemeContext. Основний контент виводиться оптимізованим компонентом FlatList у вигляді карток, кожна з яких відображає назву файлу, поточний прогрес читання у відсотках та інтерактивні елементи для додавання книги в улюблені (toggleFavorite), оновлення стану або її видалення(Рис 3.3).

Персистентне збереження інформації реалізовано через AsyncStorage.

Імпорт нових документів форматів TXT та PDF виконується асинхронно за допомогою expo-document-picker із обов'язковою перевіркою на дублікати(Рис 3.2). Навігаційні переходи до відповідних читалок здійснює expo-router на основі аналізу розширення файлу. Для зручності користувача на екрані також інтегровано функції сортування(Рис 3.5) за прогресом та фільтрації книг(Рис 3.4), які реалізовані через обробку масивів даних у реальному часі та викликаються за допомогою модульних вікон Modal.

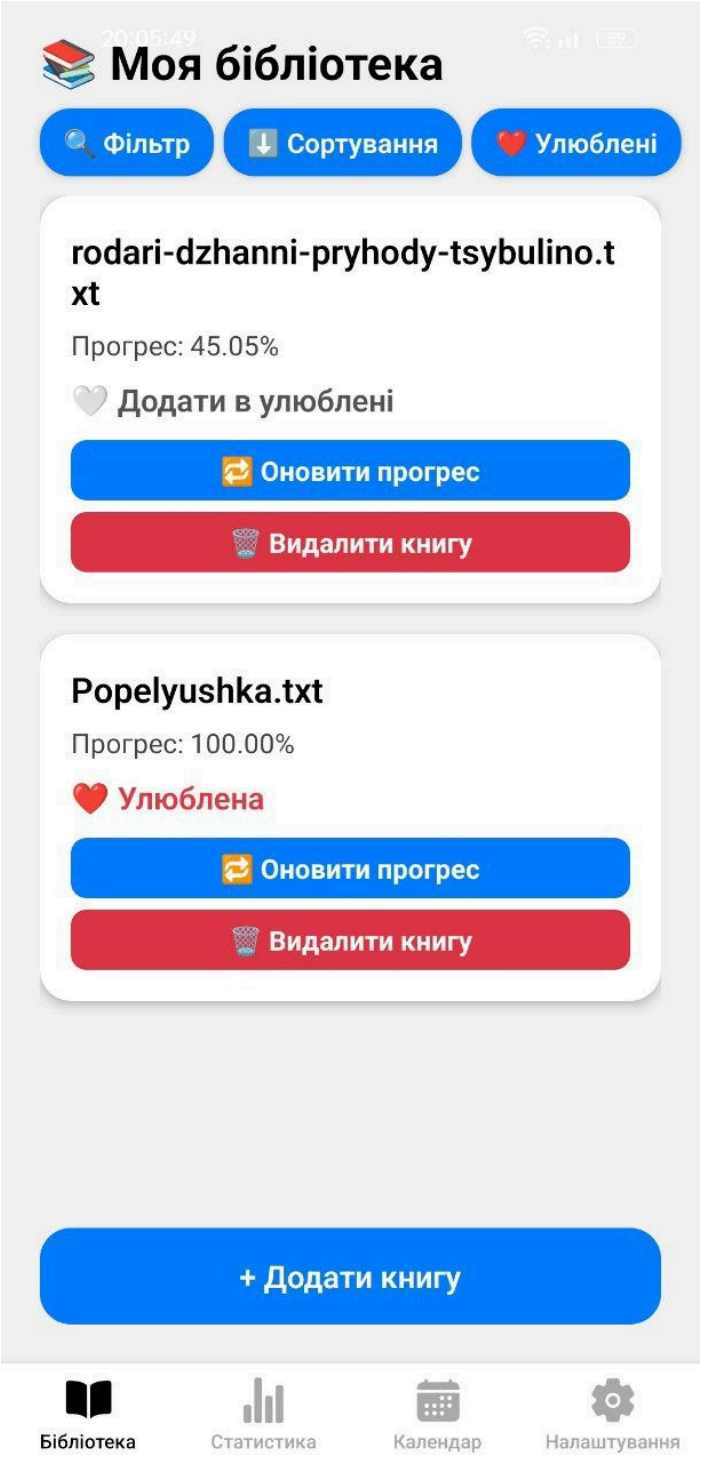


Рис. 3.1. Головний екран бібліотеки та список книг

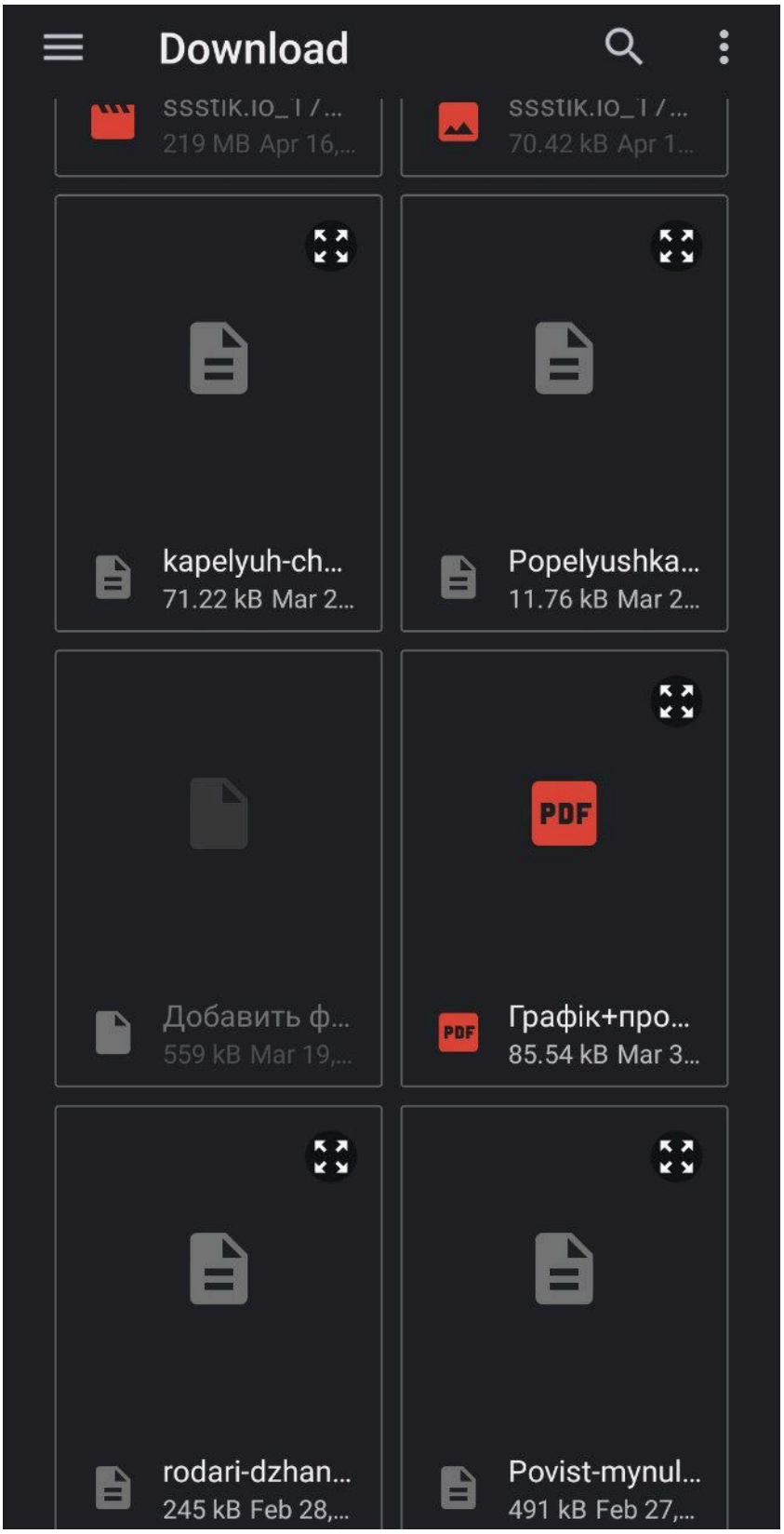


Рис. 3.2. Екран додавання книги за допомогою системи Document Picker.

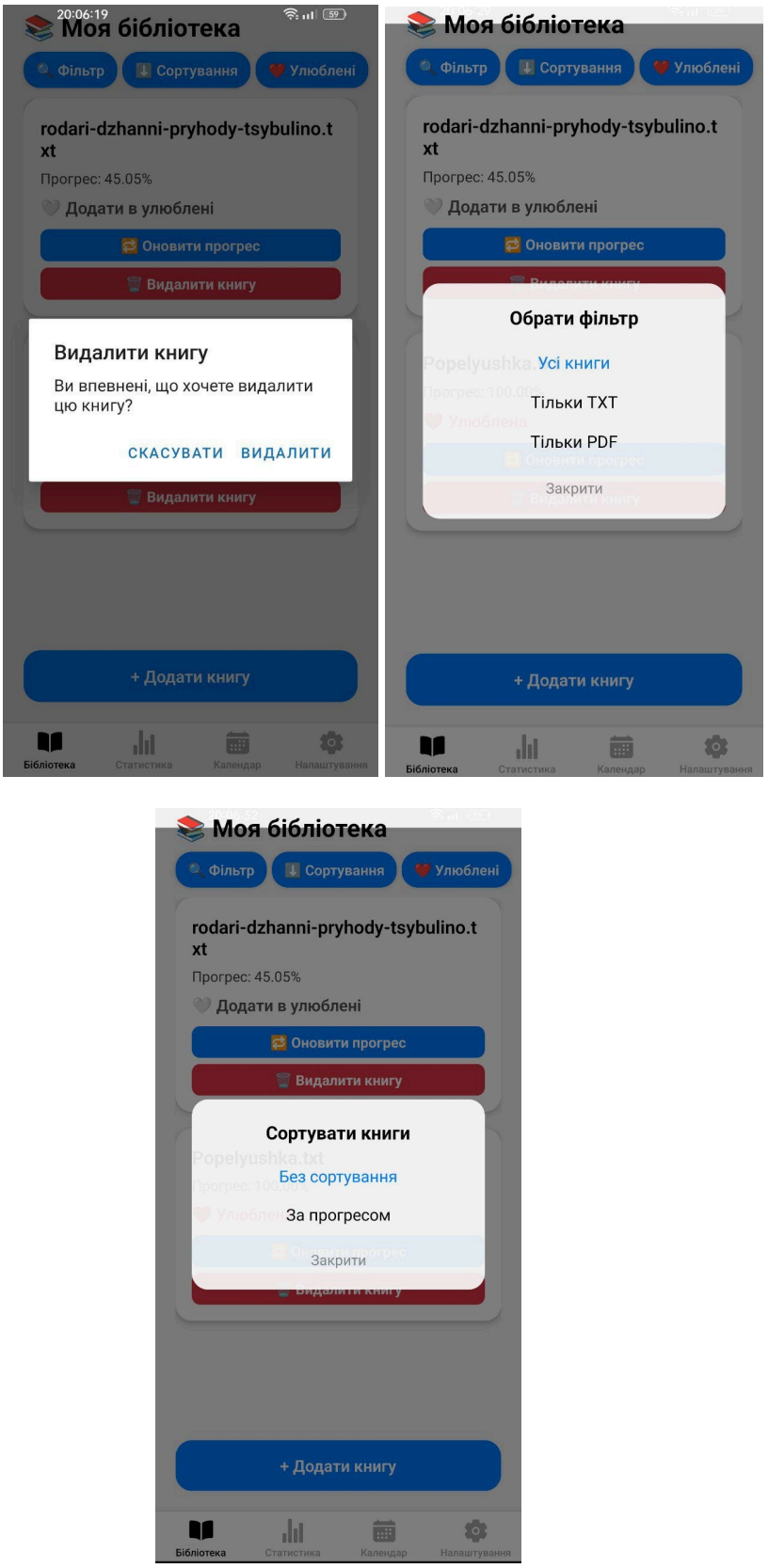
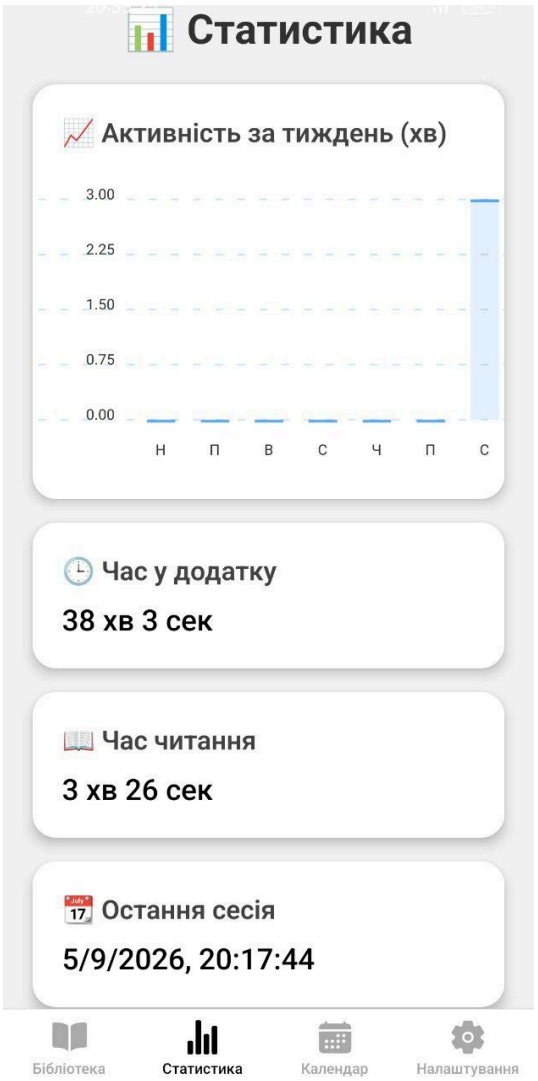


Рис. 3.3, 3.4, 3.5. Модульні вікна Видалення, Фільтрації та Сортування книг

Екран аналітики та статистики, реалізований у модулі StatsScreen,

призначений для збору, обробки та візуалізації метрик користувацької активності в реальному часі. Програмна логіка компонента побудована на взаємодії з асинхронним сховищем AsyncStorage та бібліотекою форматування часу date-fns з використанням української локалізації uk. Компонент використовує хук useFocusEffect для гарантування актуалізації даних кожного разу, коли вікно переходить у стан активного фокусу користувача. Архітектура графічного інтерфейсу підтримує адаптацію колірних тем через ThemeContext та виводить інформаційні блоки поверх накладеного кастомного фону ImageBackground.

Візуальну структуру інтерфейсу формує вертикально орієнтований контейнер ScrollView, що містить кастомні інформаційні картки (Card), які забезпечені ефектами тіней (shadowOpacity, elevation) для створення візуальної глибини. Головним елементом аналітичної панелі є віджет тижневої активності, побудований на базі графічного компонента BarChart з бібліотеки react-native-chart-kit. Графік динамічно обчислює ширину робочої області через API Dimensions операційної системи та відображає стовпчикову діаграму розподілу часу читання за останні сім днів у розрізі хвилин. Конфігурація діаграми (chartConfig) автоматично підлаштовує колір градієнтів, підписів осей та стовпчиків під світлу тему оформлення. Загальний вигляд реалізованого екрана аналітики та метрик активності наведено на рисунку 3.6 та 3.7.



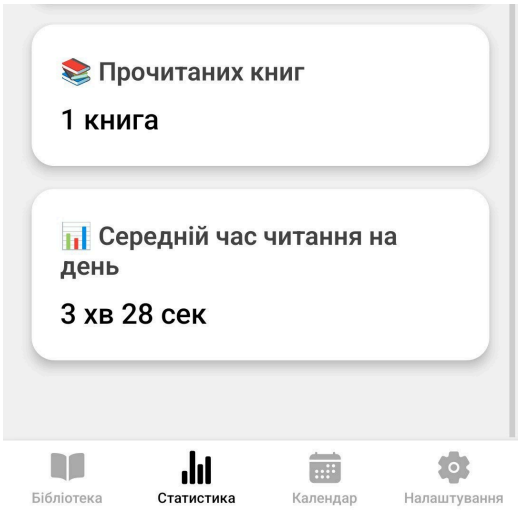


Рис. 3.6 та 3.7. Екран Статистики

Модуль інтерактивного календаря читання, реалізований у компоненті CalendarScreen, виконує функцію системного трекера та інструменту щоденного моніторингу залученості користувача до процесу взаємодії з літературою. Програмна логіка екрана тісно інтегрована з утилітами бібліотеки date-fns, зокрема методами формування часових інтервалів startOfMonth, endOfMonth та eachDayOfInterval, що дозволяє гнучко генерувати динамічну сітку днів для будь-якого обраного місяця. Керування станом синхронізації та вибірки даних покладається на хук useFocusEffect, який запускає каскад асинхронних операцій читання локальних сховищ при кожному отриманні компонентом фокусу візуалізації. Візуальне оформлення екрана наслідує загальну концепцію додатка, використовуючи ThemeContext для підтримки глобальної колірної схеми та шар ImageBackground для накладання декоративного фону.

Елементи графічного інтерфейсу верхнього рівня представлені блоком перемикання місяців monthSwitcher, що складається з навігаційних кнопок TouchableOpacity та текстового поля, де назва місяця автоматично форматується з великої літери за локаллю uk. Головна робоча область організована у вигляді гнучкого контейнера з властивостями flexDirection: "row" та flexWrap. Програмна логіка рендерингу оперує умовним стилюванням елементів: поточна дата автоматично виділяється заливкою фірмового синього кольору, а дні, що мають збережену історію активності (час читання, записи або завершені книги), маркуються акцентною зеленою рамкою (#28a745), створюючи для користувача наочний ефект теплової карти (heat map) особистих досягнень. Графічний вигляд сформованої сітки календаря активності представлено на рисунку 3.8.

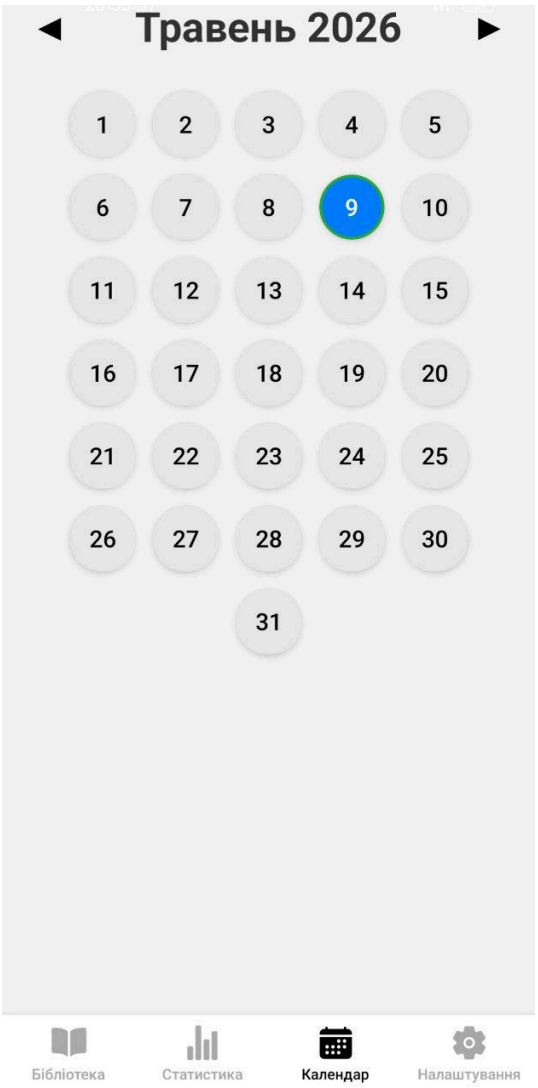


Рис. 3.8. Екран Календаря

Екран конфігурування та налаштування інтерфейсу користувача, реалізований у модулі SettingsScreen, виконує функцію панелі адміністрування параметрів кастомізації мобільного застосунку. Програмна логіка компонента забезпечує персистентне керування станом глобального оформлення через взаємодію з хуками useContext (інтеграція з ThemeContext) та механізмами асинхронного збереження AsyncStorage. Організація призначеного для користувача інтерфейсу базується на використанні прокручуваного контейнера ScrollView та кастомних карток параметрів, що динамічно змінюють свої стилі (darkContainer, darkCard, darkText) у відповідь на перемикання прапорця системної теми. Для відображення декоративного підкладкового шару на екрані розгорнуто універсальний компонент ImageBackground.

Усі доступні параметри адміністрування згруповані за функціональним призначенням у вигляді карток-модулів. Перший блок відповідає за інверсію колірної схеми додатка і реалізований через інтерактивний елемент керування Switch, який змінює стан прапорця isDark та ініціює виклик глобального методу toggleTheme. Другий інформаційний блок містить інструменти кастомізації графічного тла. Тут інтегровано горизонтальну стрічку ScrollView для швидкого вибору одного зі статичних пресетів високоякісних зображень із віддаленого репозиторію Unsplash, визначених у масиві даних backgroundOptions. Поточний активний вибір підсвічується акцентною рамкою синього кольору із товщиною лінії у 3 пікселі. Загальний вигляд екрана налаштувань у темній колірній схемі із горизонтальним селектором зображень наведено на рисунку 3.10.

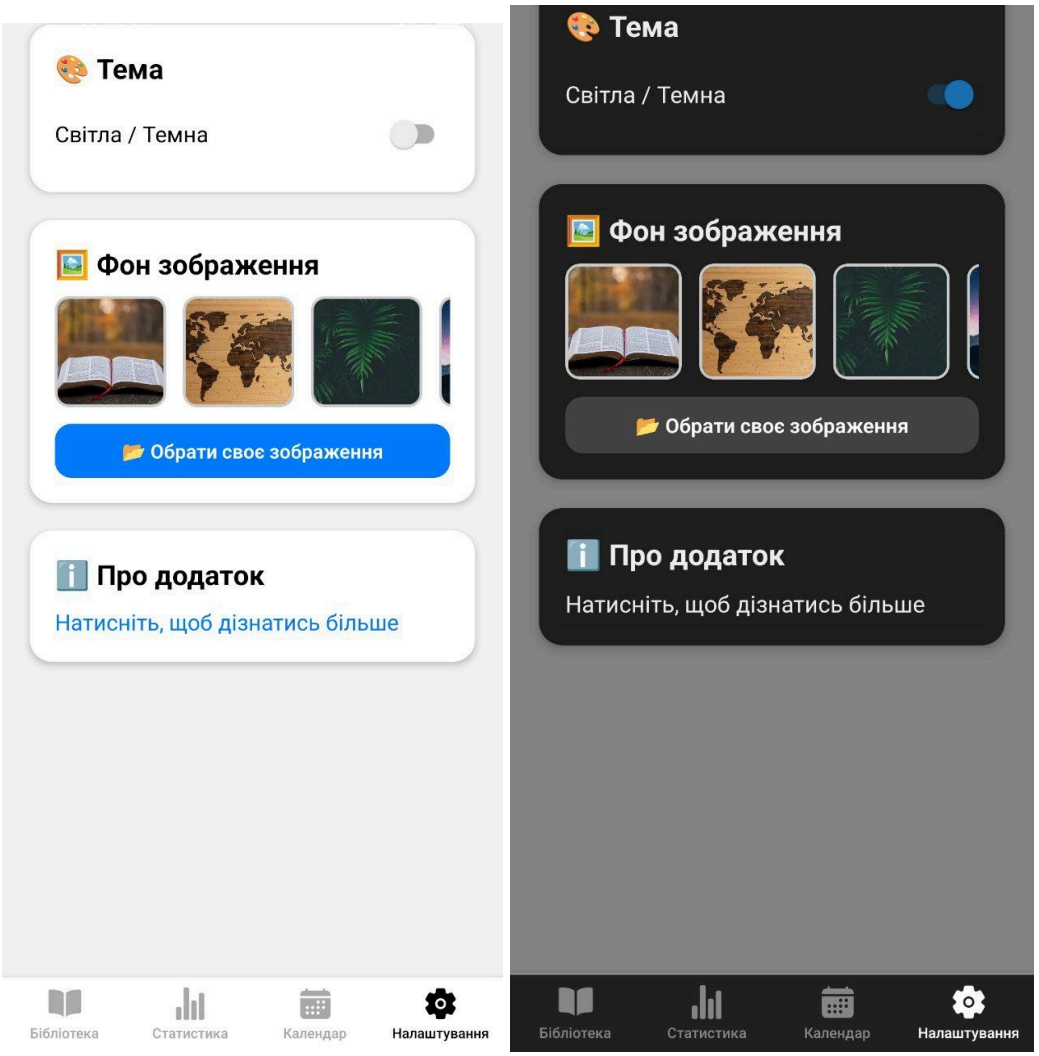


Рис. 3.9 та 3.10. Екран Налаштувань застосунку(світла та темна варіації)

Керівництво користувача:

Запуск: Відкрийте застосунок. На головному екрані відобразиться ваша бібліотека книг.

Навігація: Використовуйте нижню панель для перемикання між бібліотекою та налаштуваннями.

Бібліотека: У меню бібліотеки ви можете побачити поточний список доданих книг, додати книгу, видалити її, оновити ваш прогрес читання та перейти до самого читання. Також є функції сортування та фільтрації за певними критеріями.

Читання: Натисніть на обкладинку книги. Якщо це PDF — відкриється вбудований переглядач із підтримкою масштабування.

Статистика: У меню статистики ви можете переглянути деякі дані про ваш час читання, час проведений у застосунку, кількість прочитаних книг і тд.

Календар: У меню календаря ви можете побачити календар поточного місяця та дні вашої активності, при кліку на певний день можна побачити коротку довідку про ваше активність в цей день та залишити нотатку.

Персоналізація: У меню налаштувань оберіть "Темна тема" для комфортного читання вночі або встановіть власне зображення на фон додатку.

3.5. Тестування програмного продукту

Тестування: Для перевірки працездатності було використано метод **«чорної скриньки»**.

Функціональне тестування: перевірено коректність відкриття файлів із різними назвами (включаючи кирилицю та пробіли). Завдяки **decodeURIComponent** помилок доступу до файлів не виявлено.

Тестування виключних ситуацій: при відсутності файлу за вказаним шляхом додаток не аварійно завершує роботу, а повертає користувача до бібліотеки або показує індикатор завантаження.

UX-тестування: підтверджено миттєве застосування тем до всіх екранів завдяки використанню **Context.Consumer**.

Тестування продуктивності: Було проведено перевірку стабільності роботи застосунку під час відкриття великих PDF-файлів та швидкого перемикання між екранами. Особлива увага приділялась використанню оперативної пам’яті та плавності інтерфейсу під час прокрутки великих документів. Результати тестування підтвердили відсутність критичних помилок, аварійного завершення роботи та витоків пам’яті.

Також було перевірено:

• коректність роботи механізму збереження прогресу;

• синхронізацію тем оформлення між екранами;

• правильність роботи системи закладок;

• відображення статистики після повторного запуску застосунку.

ВИСНОВКИ ДО РОЗДІЛУ

У третьому розділі проведено програмну реалізацію та тестування мобільного застосунку. За результатами роботи сформовано такі висновки:

1. **Технології:** Використання React Native та Expo дозволило створити кросплатформне рішення з високою продуктивністю та адаптивним інтерфейсом для Android та iOS.
2. **Архітектура:** Реалізовано модульну структуру застосунку. Завдяки інтеграції AsyncStorage забезпечено надійне збереження прогресу читання, бібліотеки книг та персональних налаштувань користувача.
3. **Функціонал:** Розроблено модулі для роботи з форматами TXT і PDF, систему контекстних нотаток та блок аналітики. Оптимізація рендерингу дозволила досягти швидкої роботи застосунку при читанні великих файлів.
4. **Валідація:** Тестування підтвердило стабільність системи та коректність роботи інтерфейсу в різних колірних темах. Продукт повністю відповідає вимогам зручності (UI/UX) та технічному завданню.

Загалом, розроблений програмний комплекс є технічно завершеним, протестованим і готовим до практичної експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі було розв’язано актуальну задачу проектування та розробки кросплатформного мобільного застосунку для читання електронних книг у форматах TXT та PDF із розширеним функціоналом персоналізації, ведення статистики та організації читацької активності користувача. За результатами виконаного дослідження та програмної реалізації можна зробити такі висновки:

Аналіз предметної області показав, що сучасні мобільні рідери, попри широкий функціонал, часто не забезпечують достатнього рівня персоналізації та інструментів для аналізу читацької активності. Це підтверджує актуальність розробки легкого кросплатформного застосунку, який поєднує базові можливості читання з додатковими функціями статистики, нотаток і закладок при збереженні простоти інтерфейсу та локальної роботи без залежності від серверної інфраструктури.

У процесі проектування було обрано компонентно-орієнтовану архітектуру на основі React Native із використанням Expo Router для файлової навігації. Такий підхід дозволив логічно розділити функціональні модулі системи (бібліотека, читалка, налаштування, статистика, календар) та забезпечити масштабованість застосунку. Використання Context API дозволило реалізувати централізоване керування глобальним станом (тема, налаштування інтерфейсу), що забезпечує миттєве оновлення UI без перезавантаження компонентів.

Програмна реалізація виконана із застосуванням сучасного стеку технологій, зокрема Expo, React Native WebView та AsyncStorage. Реалізовано модулі для відображення PDF-файлів через вбудовані системні рушії, а також для обробки TXT-файлів із власною логікою пагінації та адаптації тексту. Додатково розроблено системи закладок, нотаток та автоматичного збереження прогресу читання, що дозволяє користувачу швидко повертатися до потрібних

фрагментів тексту. Усі дані зберігаються локально на пристрої, що забезпечує конфіденційність та автономність роботи застосунку.

Практична апробація підтвердила стабільність роботи застосунку при відкритті великих текстових і PDF-файлів, а також коректність функціонування всіх основних модулів: читання, закладок, нотаток, статистики та календаря активності. Завдяки асинхронній моделі обробки даних та оптимізованій роботі з локальним сховищем вдалося забезпечити плавність інтерфейсу та відсутність критичних затримок при взаємодії з користувачем.

Мета кваліфікаційної роботи досягнута, усі поставлені завдання виконані в повному обсязі. Розроблений програмний продукт є готовим до використання та має потенціал для подальшого розвитку, зокрема шляхом додавання підтримки нових форматів електронних книг, хмарної синхронізації та розширеної аналітики читацької поведінки користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сяо Б. *React. Швидкий старт. Розробка односторінкових вебдодатків*. Київ: Видавнича група BHV, 2021. 288 с.

2. React Native Documentation. URL: https://reactnative.dev/docs/getting-started (дата звернення: 09.05.2026).

3. Expo Documentation. URL: https://docs.expo.dev (дата звернення: 09.05.2026).

4. Expo Router Documentation. URL: https://docs.expo.dev/router/introduction/ (дата звернення: 09.05.2026).

5. Node.js v20.x Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 09.05.2026).

6. AsyncStorage Reference (React Native Async Storage). URL: https://react-native-async-storage.github.io/async-storage/docs/usage/ (дата звернення: 09.05.2026).

7. SQLite in React Native (expo-sqlite). URL: https://docs.expo.dev/versions/latest/sdk/sqlite/ (дата звернення: 09.05.2026).

8. Expo File System API. URL: https://docs.expo.dev/versions/latest/sdk/filesystem/ (дата звернення: 09.05.2026).

9. Expo Document Picker (робота з файлами TXT/PDF). URL: https://docs.expo.dev/versions/latest/sdk/document-picker/ (дата звернення: 09.05.2026).

10. React Native WebView Guide. URL: https://github.com/react-native-webview/react-native-webview (дата звернення: 09.05.2026).

11. React Navigation (основи навігації мобільних застосунків). URL: https://reactnavigation.org/docs/getting-started (дата звернення: 09.05.2026).

12. React Native PDF Library (react-native-pdf). URL: https://github.com/wonday/react-native-pdf (дата звернення: 09.05.2026).

13. React Native Calendars (календар для відображення статистики читання). URL: https://github.com/wix/react-native-calendars (дата звернення: 09.05.2026).

14. Zustand State Management (керування станом застосунку). URL: https://docs.pmnd.rs/zustand/getting-started/introduction (дата звернення: 09.05.2026).

15. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/introduction/getting-started (дата звернення: 09.05.2026).

16. TypeScript Handbook. URL: https://www.typescriptlang.org/docs/ (дата звернення: 09.05.2026).

17. MDN Web Docs (JavaScript, Async/Await, LocalStorage концепції). URL: https://developer.mozilla.org/ (дата звернення: 09.05.2026).

18. eBoox — застосунок для читання книг. Google Play Store. URL: https://play.google.com/store/apps/details?id=com.reader.books (дата звернення: 20.04.2025).

19. RORK — сервіс для читання та нотаток. URL: https://rork.ua (дата звернення: 20.04.2025).

ДОДАТКИ

Додаток А

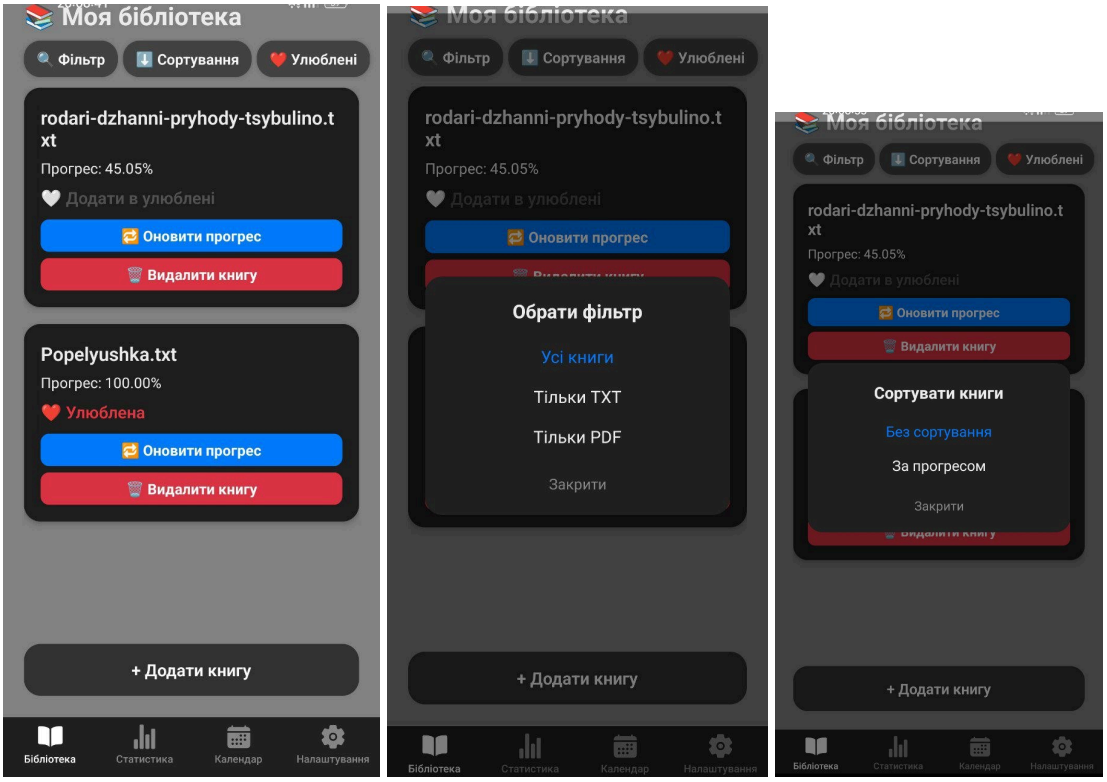


Рис. А.1, А.2 та А.3. Головний екран та модульні вікна Фільтрації і Сортування книг у темній темі застосунку

Додаток Б

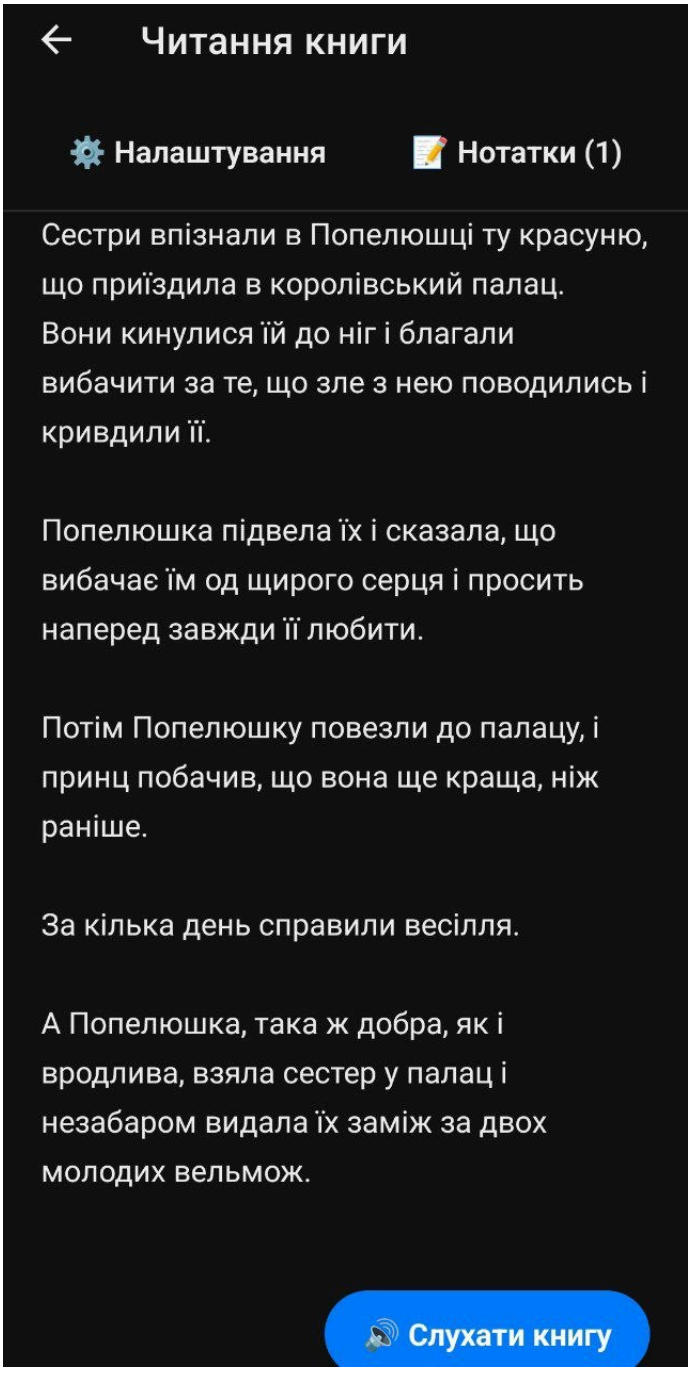


Рис. Б. 1. Екран читання книг TXT

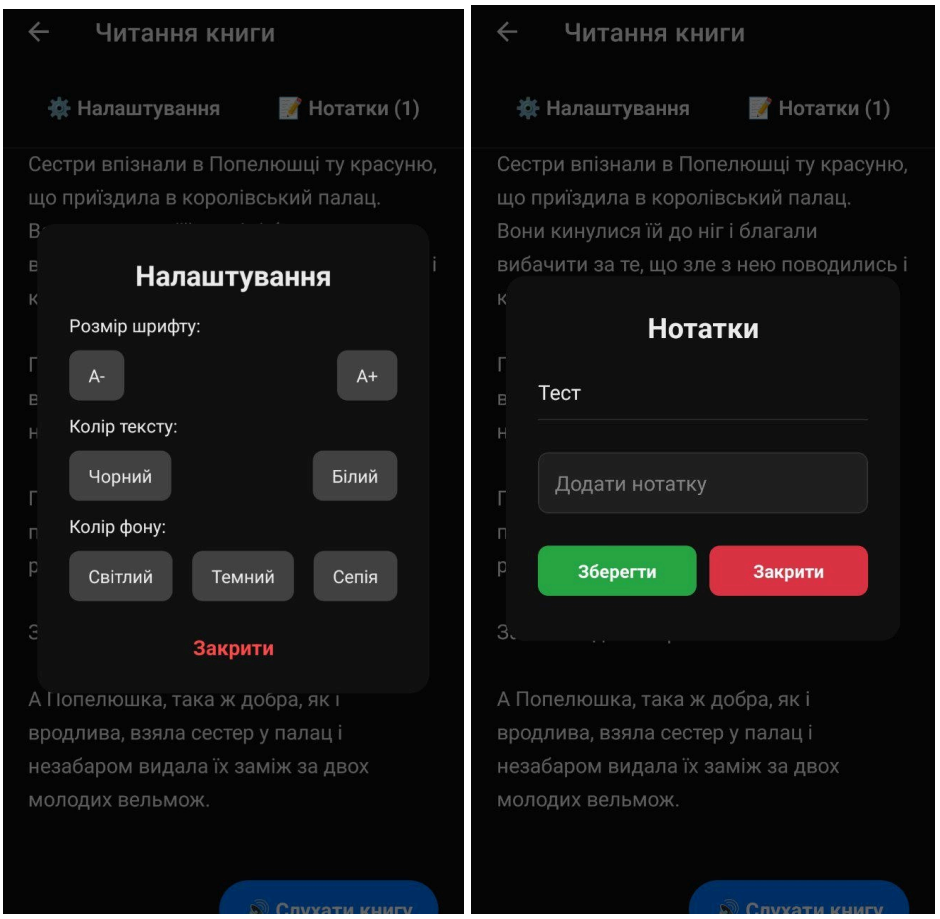
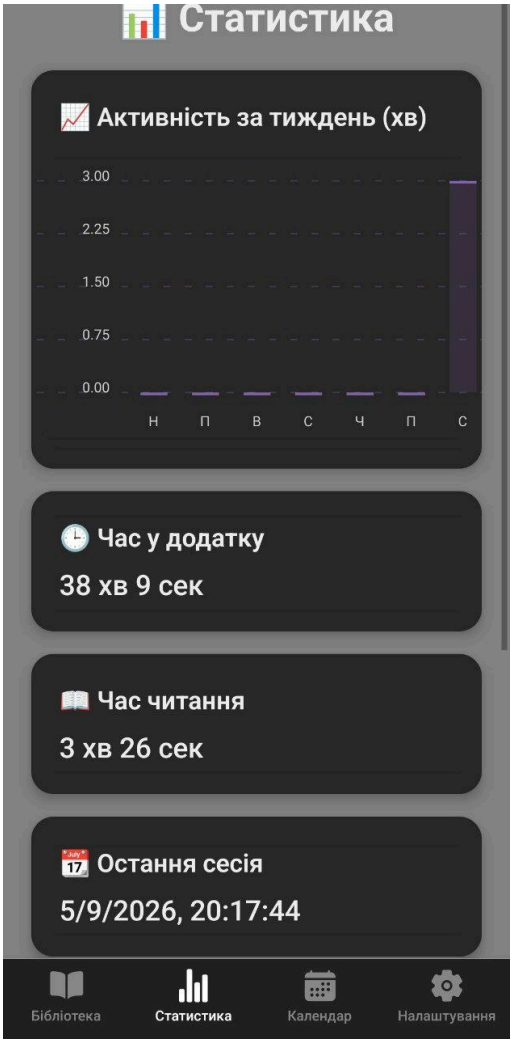


Рис. Б.2 та Б.3. Модульні вікна Налаштувань та Нотатків екрану читання

Додаток В



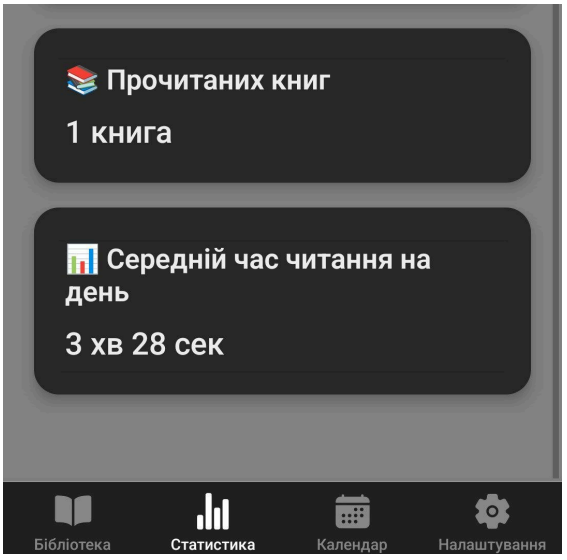


Рис. В.1 та В.2. Екран Статистики(темна тема)

Додаток Г

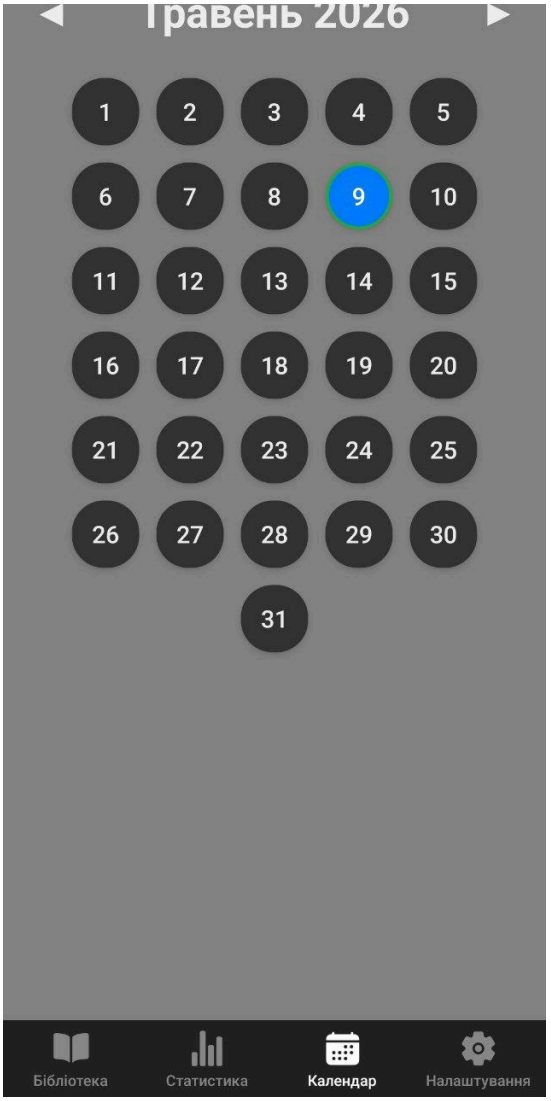
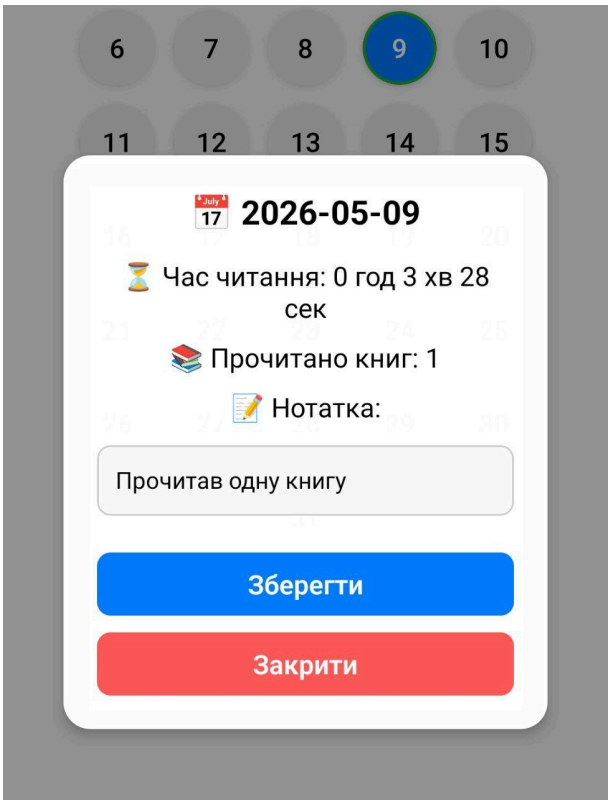


Рис. Г. 1. Екран Календаря читання(темна тема)



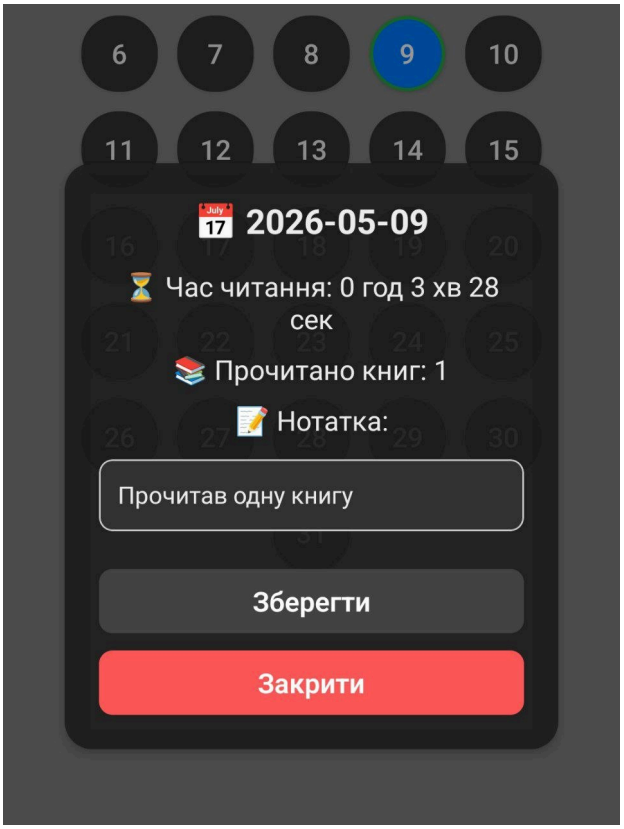


Рис. Г.2 та Г.3. Модальне вікно з даними за день(світла та темна варіації)

Додаток Д

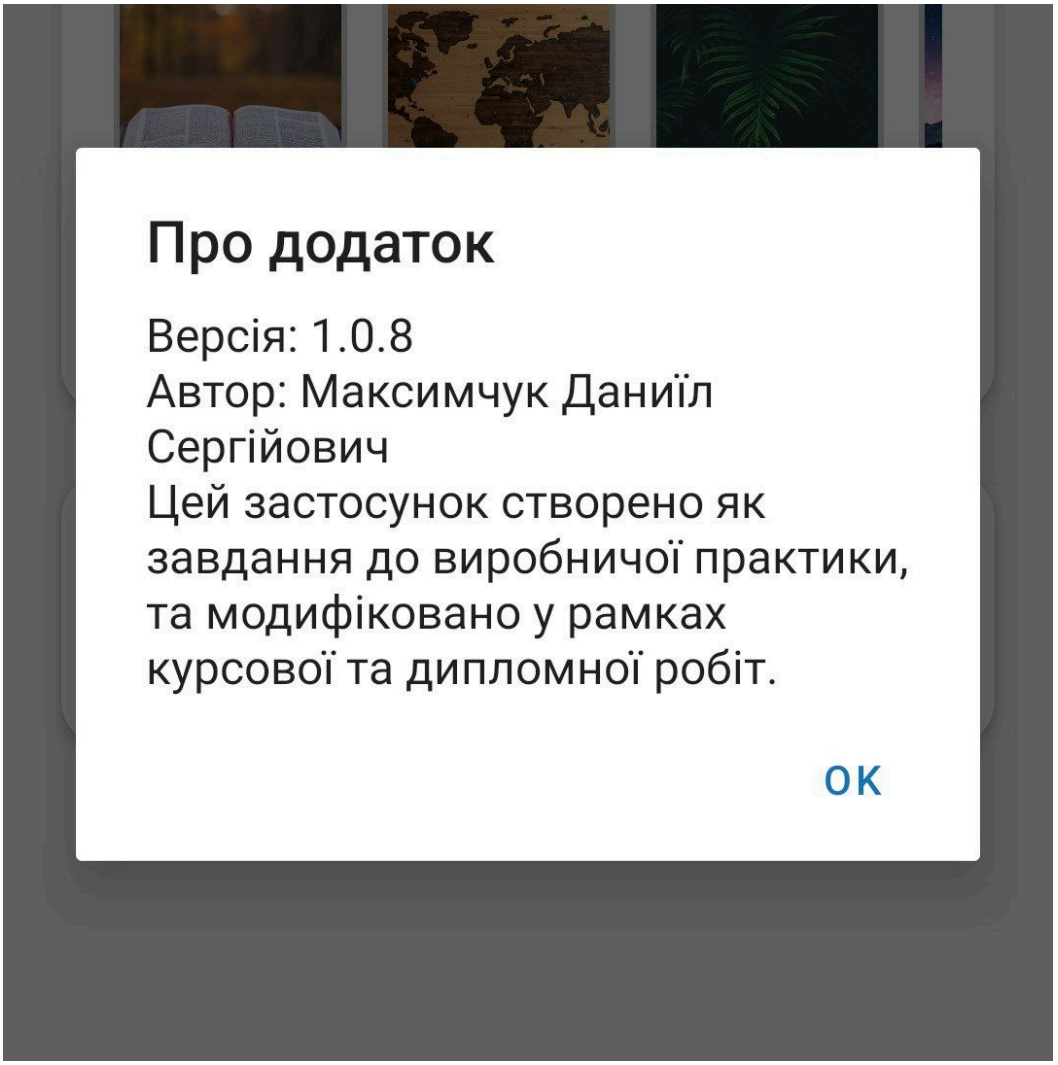


Рис. Д. 1. Модальне вікно “Про додаток”

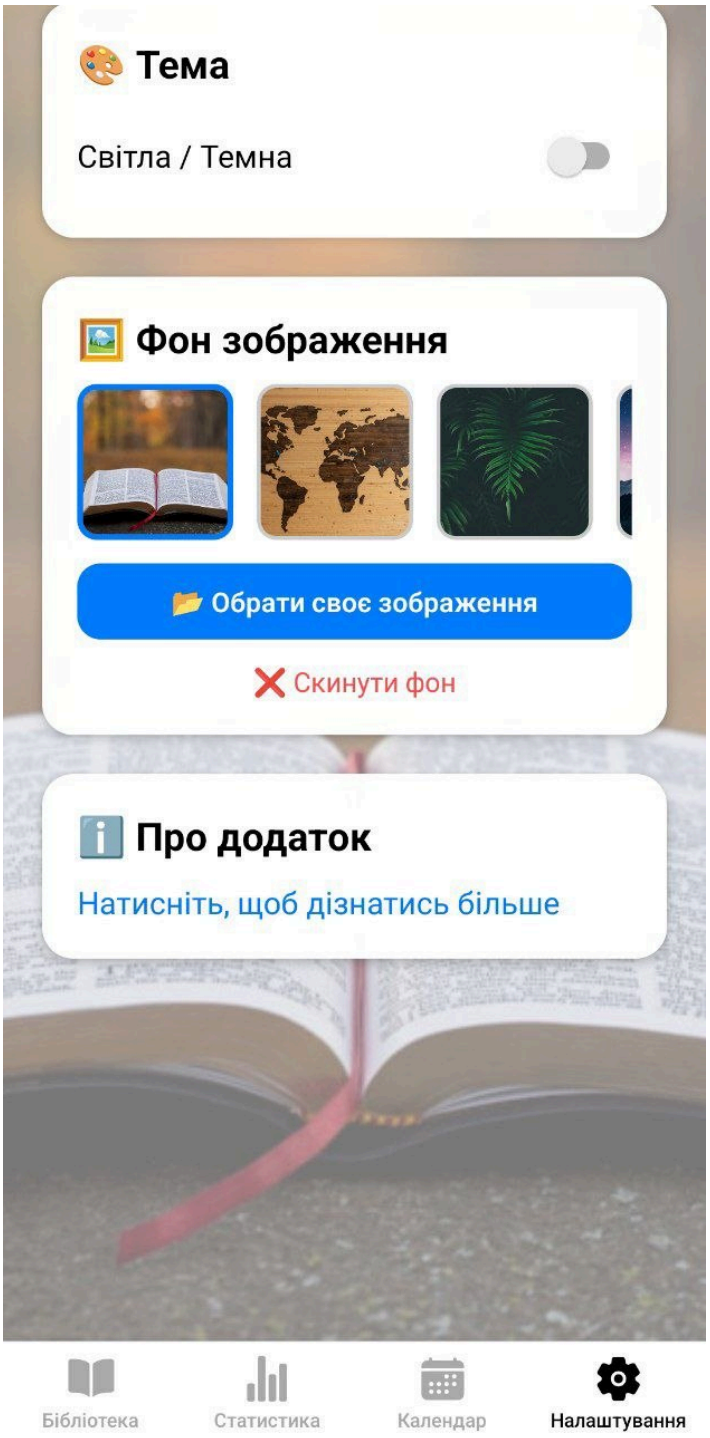


Рис. Д. 2. Екран Налаштувань застосунку з фоновим зображенням