МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

КАЛИНИНГРАДСКОЙ ОБЛАСТИ

ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ

«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СТРОИТЕЛЬСТВА»

(ГБУ КО ПОО «КИТиС»)

|  |  |  |
| --- | --- | --- |
| Курсовой проект  допущен к защите  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата)  Зам.директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_\_Павленко Г.Я.  (подпись) |  | Курсовой проект  защищен с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Пояснительная записка к курсовому проекту

по дисциплине: МДК 01.01 Разработка программных модулей

Тема: Разработка мобильного приложения управления личной библиотекой

Специальность 09.02.07 «Информационные системы и программирование»

|  |  |
| --- | --- |
| Выполнил: | студент 3 курса,  группы ИСп 22-1  Ткач Валерия Евгеньевна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель: | Большакова-Стрекалова Анна Викторовна, преподаватель ГБУ КО ПОО КИТиС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Калининград

2025

Аннотация

Данный курсовой проект посвящен разработке мобильного приложения «MyLib», предназначенного для учета и управления личной библиотекой пользователя. Актуальность проекта обусловлена ростом интереса к электронным библиотекам и необходимостью систематизации информации о прочитанных и запланированных книгах.

В качестве инструментария для реализации приложения использованы среда разработки Android Studio и язык программирования Kotlin. Приложение позволяет пользователю осуществлять поиск книг по названию и автору через API Google Books, добавлять выбранные книги в личную библиотеку, просматривать информацию о них, а также управлять содержимым своей коллекции.

В работе рассмотрены этапы проектирования архитектуры приложения, создание пользовательского интерфейса, реализация сетевых запросов и хранения данных. Приложение поддерживает динамическую загрузку информации, адаптацию интерфейса под разные устройства, а также выбор пользовательской темы оформления.

Ключевые слова: Android, Kotlin, библиотека, мобильное приложение, Google Books API, разработка.

Объём работы: 67 страниц.

Приложение: «MyLib» (APK-файл).

Среда разработки: Android Studio.

GitHub: https://github.com/Valeriuaa/MyLib

Содержание

[Введение 4](#_Toc196676795)

[1 Анализ предметной области 6](#_Toc196676796)

[1.1 Аналоги разрабатываемого приложения 6](#_Toc196676797)

[1.2 Техническое задание 10](#_Toc196676798)

[1.3 Описание структуры приложения 12](#_Toc196676799)

[2 Описание разработки приложения 14](#_Toc196676800)

[2.1 Обоснование средств разработки 14](#_Toc196676801)

[2.2 Разработка интерфейса 18](#_Toc196676802)

[2.3 Разработка логики работы программы. Схема взаимодействия компонентов проекта 23](#_Toc196676803)

[2.3.1 Логика работы приложения 23](#_Toc196676804)

[2.3.2 Схема взаимодействия компонентов 24](#_Toc196676805)

[2.4 Описание переменных, компонентов, классов и подпрограмм 25](#_Toc196676806)

[3 Тестирование и установка программы 29](#_Toc196676807)

[Заключение 34](#_Toc196676808)

[Список использованной литературы 35](#_Toc196676809)

Приложение А. [Листинг программы 36](#_Toc196676811)

Приложение Б. [Руководство пользователя 61](#_Toc196676813)

# Введение

В современном мире, где информация становится все более доступной и многообразной, управление личной библиотекой представляет собой важную задачу как для любителей чтения, так и для профессионалов. Сегодня мы наблюдаем неуклонный рост количества книг, статей, научных работ и других материалов, которые мы накапливаем в течение жизни. Этот процесс, хотя и увлекательный, может стать источником значительных трудностей, связанных с поиском нужной информации, сортировкой материалов и учётом прочитанного. Учитывая данные вызовы, разработка удобного и функционального приложения для управления личной библиотекой становится необходимостью.

Технологический прогресс делает возможным создание программных решений, которые могут помочь пользователям эффективно организовать свои библиотеки, управлять коллекцией книг и получать доступ к информации о них в любое время и в любом месте. Приложения для управления библиотеками позволяют не только вести учет книг, но и расширять возможности их поиска, сортировки. На современном рынке существует множество решений, однако каждое из них имеет свои преимущества и недостатки, что указывает на необходимость разработки нового, более удобного и функционального инструмента.

В данном курсовом проекте будет представлена разработка мобильного приложения «MyLib», которое призвано упростить процесс управления личной библиотекой. Приложение будет включать в себя функции, позволяющие пользователям добавлять книги, отслеживать их прочтение. Основное внимание будет уделено интуитивно понятному интерфейсу, который обеспечит простоту в использовании для людей разных возрастов и уровня подготовки.

Кроме того, приложение «MyLib» будет интегрировано с библиотечными базами данных, что позволит пользователям быстрее находить информацию о книгах, получать доступ к рецензиям и рекомендациям, а также находить другие ресурсы по интересующим темам. Одна из ключевых задач разработки заключается в создании системы категории и тегирования, что позволит быстро ориентироваться в большом количестве материалов.

Таким образом, цель данного курсового проекта заключается в создании мобильного приложения, которое не только упростит управление личной библиотекой, но и сделает процесс чтения более увлекательным и доступным. В рамках работы будут рассмотрены этапы разработки, определены требования к функционалу, проведены тестирования приложения. Это даст возможность понять всю важность и актуальность разработки современных решений для организации личных библиотек в условиях стремительного роста информационного потока.

1. Анализ предметной области

Предметная область управления личной библиотекой охватывает множество аспектов, включая учет книг, их классификацию, поиск и обмен информацией.

Важными функциями такого приложения являются:

1. Добавление книг: Пользователь должен иметь возможность легко добавлять новые книги в свою библиотеку с указанием основных данных (название, автор, жанр, год издания и т.д.).

2. Поиск и фильтрация: Необходимость быстрого поиска книг по различным критериям (автору, названию или жанру) позволяет пользователю эффективно управлять своей коллекцией

3. Учет прочитанных книг: Возможность отслеживания прочитанных книг и планирования будущих чтений помогает пользователю организовать свое время.

4. Интеграция с внешними источниками: Подключение к базам данных (например, ISBN) может значительно упростить процесс добавления книг.

1.1 Аналоги разрабатываемого приложения

ЛитРес — это одна из крупнейших российских платформ для покупки и чтения электронных книг и аудиокниг. Платформа предлагает пользователям доступ к обширной библиотеке, включающей как популярные новинки, так и классические произведения.

Преимущества: Широкий ассортимент, мобильное приложение с удобным интерфейсом, синхронизация между устройствами.

Недостатки: Платный контент, ограниченные функции для управления личной библиотекой, региональные ограничения.



Рисунок 1 – Логотип LitRes

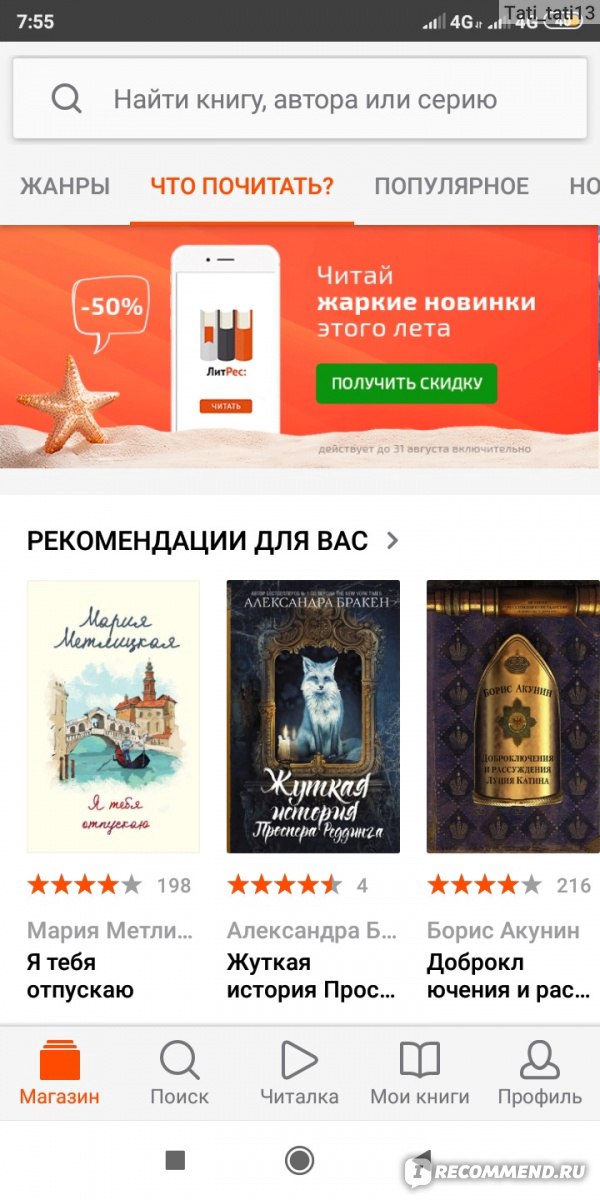


Рисунок 2 – Интерфейс LitRes

Goodreads – это одна из самых популярных платформ для любителей книг, позволяющая пользователям отслеживать свои прочитанные книги, делиться отзывами и находить новые произведения на основе рекомендаций.

Преимущества: Широкая база данных книг; возможность оставлять отзывы; социальные функции.

Недостатки: Ограниченные возможности по настройке интерфейса; отсутствие оффлайн-доступа.



Рисунок 3 – Логотип Goodreads



Рисунок 4 – Интерфейс Goodreads

Еще одним аналогом является BookBuddy – это мобильное приложение для управления личной библиотекой, которое позволяет пользователям удобно отслеживать свои книги на смартфонах или планшетах.

Преимущества: Удобный интерфейс; возможность сканирования штрих-кодов для добавления книг.

Недостатки: Платное приложение с ограниченным функционалом в бесплатной версии.



Рисунок 5 – Логотип BookBuddy

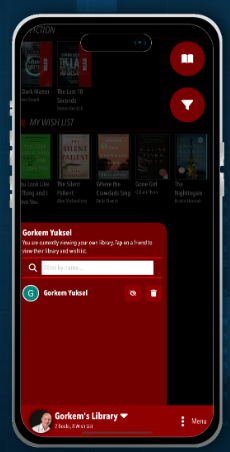


Рисунок 6 – Интерфейс BookBuddy

1.2 Техническое задание

1. Общая информация:

Наименование: приложение для управления личной библиотекой «MyLib»;

Назначение и область применения: мобильное приложение для управления личной библиотекой читателям.

1. Требования к программе

Требования к функциональным характеристикам

Пользовательские функции:

* Добавление книг в библиотеку (сканирование штрих-кода, ручной ввод).
* Удаление книг из библиотеки.
* Поиск книг по различным критериям (название, автор, жанр).
* Учет прочитанных книг (ввод даты прочтения).

Требования к надежности

* Устойчивость к сбоям: приложение должно корректно обрабатывать ошибки без потери данных.

Требования к производительности

* Время загрузки приложения не должно превышать 3 секунд на современных устройствах.
* Поиск книг должен выполняться за 2 секунды при наличии 1000+ записей в библиотеке.
* Минимальные системные требования для установки приложения на устройства:
* Android: версия 8.0 и выше

Требования к тестированию

* Проведение юнит-тестирования всех функций.
* Тестирование пользовательского интерфейса на различных устройствах и разрешениях экранов.
* Тестирование производительности на различных этапах разработки.
* Проведение нагрузочного тестирования для оценки стабильности работы приложения при высокой активности пользователей.
* Тестирование безопасности данных (проверка уязвимостей).

Требования к сопровождению

* Обеспечение технической поддержки пользователей.
* Регулярное обновление приложения с исправлениями ошибок и новыми функциями.
* Создание документации для пользователей и для разработчиков (API, описание функций и возможностей приложения).

3. Стадии разработки

Подготовительный этап:

* Сбор и анализ требований.
* Определение целевой аудитории.

Проектирование:

* Создание макетов пользовательского интерфейса.
* Описание структуры базы данных.

Разработка:

* Имплементация функциональности приложения согласно требованиям.
* Разработка API для взаимодействия с базой данных.

Тестирование:

* Проведение всех видов тестирования.
* Исправление выявленных ошибок.

Поддержка и сопровождение:

* Регулярные обновления и поддержки приложения.

4. Описание структуры программы

* Модуль управления книгами: Добавление, удаление, поиск.
* Модуль учета прочитанных книг: Функции внесения данных о прочитанных книгах.
* База данных: Хранение информации о книгах.
* Интерфейс: Удобное навигационное меню, дружественный пользовательский интерфейс.

## 1.3 Описание структуры приложения

Приложение «MyLib» предназначено для удобного поиска и сохранения информации о книгах. Оно разработано на языке Kotlin и использует GoogleBooks API для получения данных о книгах из внешнего источника.

Основная задача приложения — предоставить пользователю возможность находить интересующие книги, по ключевым словам, (название, автор и т.д.) и сохранять их в личную коллекцию внутри приложения. Интерфейс построен таким образом, чтобы обеспечить простую и быструю навигацию между результатами поиска и сохранёнными книгами.

Пользователь может:

* выполнять поиск книг через Google Books API;
* просматривать подробную информацию о найденной книге;
* добавлять понравившиеся книги в свою локальную библиотеку;
* удалять книги из списка сохранённых;
* использовать сохранённую коллекцию для чтения описаний, повторного поиска или планирования чтения.

Вся информация о сохранённых книгах хранится локально в памяти приложения, что делает работу быстрой и независимой от постоянного подключения к интернету (для уже сохранённых данных).

# 2 Описание разработки приложения

## 2.1 Обоснование средств разработки

При разработке приложения MyLib было принято решение использовать Google Books API в качестве основного инструмента для получения информации о книгах. Данный API предоставляет доступ к обширной базе данных, содержащей миллионы наименований книг с подробной информацией — включая название, автора, обложку, описание, дату публикации и другие метаданные.

Преимущества использования Google Books API:

Широкий охват: база данных Google Books охватывает книги на различных языках и из разных источников, включая коммерческие и свободные издания;

Бесплатность и доступность: API предоставляется бесплатно для разработчиков и легко интегрируется в Android-приложения;

Гибкий поиск: поддерживаются различные параметры поиска по названию, автору, издателю, ISBN и т.д.;

Поддержка формата JSON: возвращаемые данные удобно обрабатывать в Kotlin с помощью популярных библиотек для работы с JSON;

Обновляемость данных: поскольку информация берется напрямую из Google, она всегда актуальна и не требует локального хранения или ручного обновления.

Использование Google Books API позволило сократить время разработки, а также обеспечить пользователя надёжным и качественным контентом без необходимости создавать собственную базу данных книг.

Язык программирования:

Kotlin: выбор языка Kotlin для разработки приложения «MyLib» обусловлен несколькими причинами:

Совместимость с Java: Kotlin полностью совместим с Java, что позволяет использовать существующие библиотеки и инструменты, а также облегчает интеграцию с уже существующими проектами на Java.

Современный синтаксис: Kotlin предлагает более лаконичный и безопасный синтаксис по сравнению с Java, что позволяет писать код быстрее и уменьшает количество ошибок.[1]

Поддержка сообществом: Kotlin активно поддерживается Google, что делает его предпочтительным выбором для Android-разработки.[4]



Рисунок 7 – Логотип Kotlin

Среда разработки:

Android Studio: это официальная интегрированная среда разработки (IDE) для Android, предоставляющая широкий набор инструментов для разработки, тестирования и отладки приложений.



Рисунок 8 – Логотип Android Studio

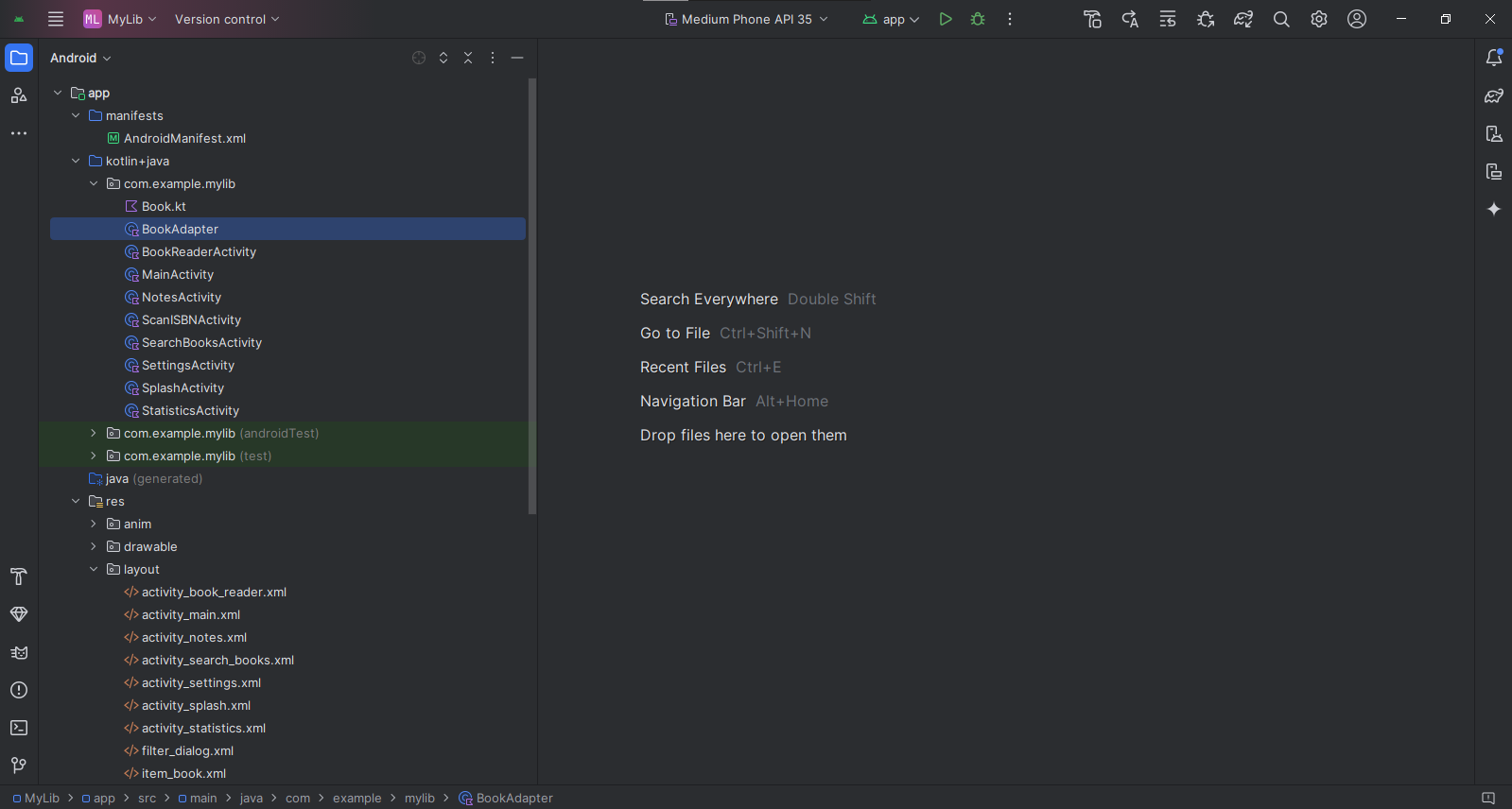


Рисунок 9 – Интерфейс Android Studio

Преимущества:

* Интуитивно понятный интерфейс и мощные возможности по редактированию кода.
* Встроенные инструменты для профилирования производительности приложения.
* Поддержка эмуляторов и реальных устройств для тестирования.

Для разработки макета дизайна интерфейса и общего вида программы будет использована программа Figma (логотип представлен на рисунке 10), которая позволяет быстро создавать визуальный макет приложения, путем размещение простых фигур, а также использования внутренних инструментов приложения.



Рисунок 10 – Логотип Figma

Также Figma позволяет подключать сторонние плагины, разработанные комьюнити дизайнеров, что позволяет упростить процесс разработки визуальных макетов в десятки раз, а следовательно, и понижает порог входа в профессиональное использование приложения. Некоторые из них представлены на рисунках 11 и 12.

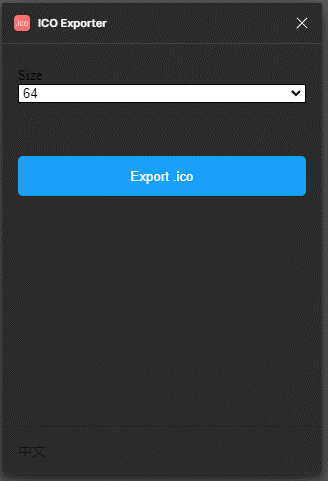


Рисунок 11 – Плагин в Figma ICO Exporter

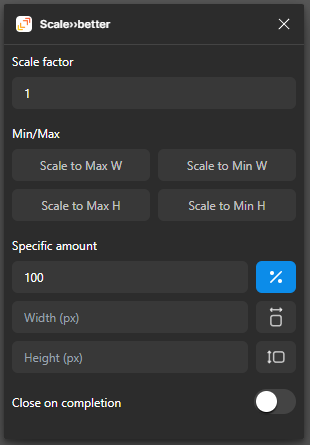


Рисунок 12 – Плагин Figma Scale better

## 2.2 Разработка интерфейса

Интерфейс программы был начат с разработки макета в Figma. Макет представлен на рисунках 13 и 14.

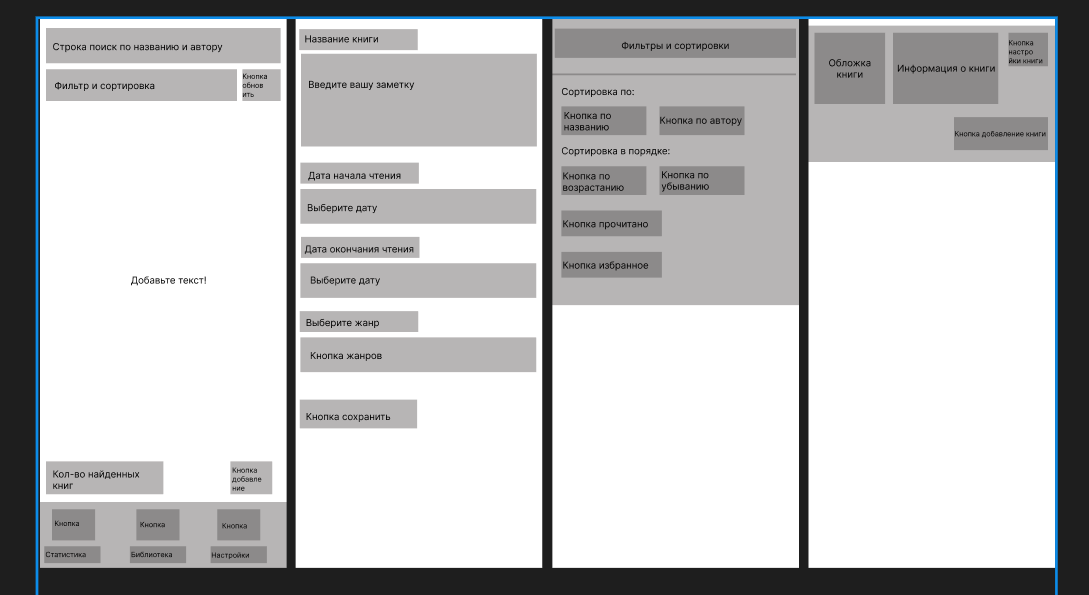


Рисунок 13 – Макет интерфейса

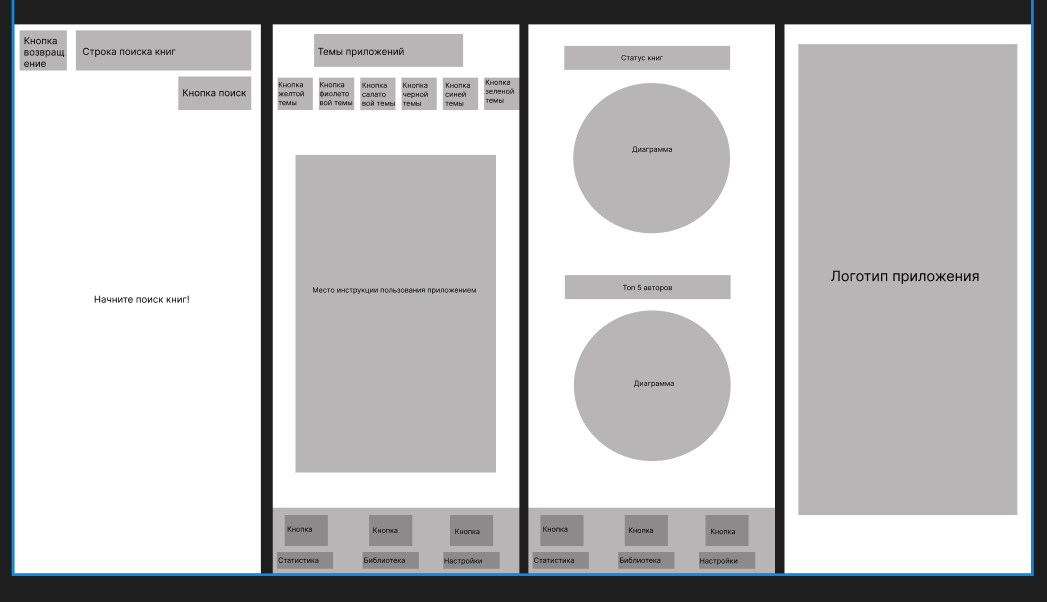


Рисунок 14 – Макет интерфейса

Далее макеты были доработаны уже в Android Studio и перенесены в код приложения. Итоговый интерфейс представлен на рисунках с 15 по 22.

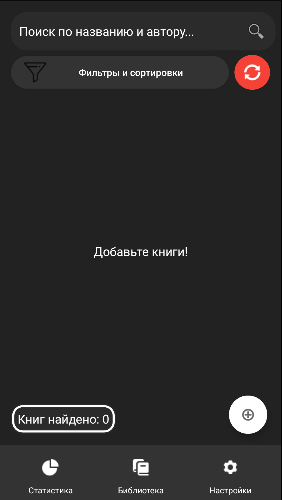


Рисунок 15 – Главный экран приложения

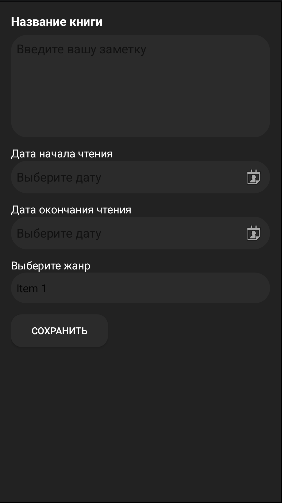


Рисунок 16 – Экран заметок

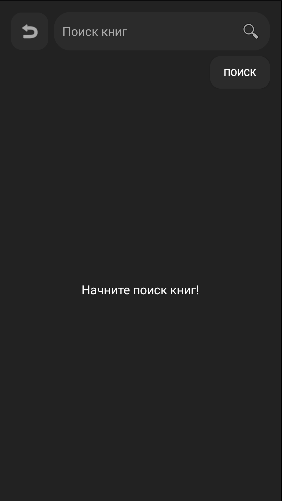


Рисунок 17 – Экран поиска книг

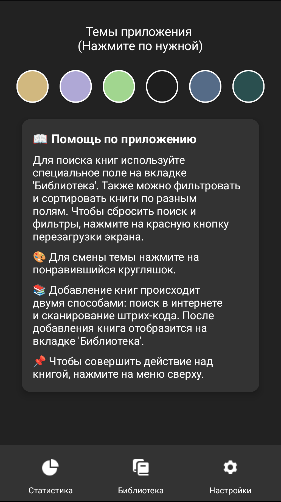


Рисунок 18 – Экран настроек



Рисунок 19 – Загрузочный экран

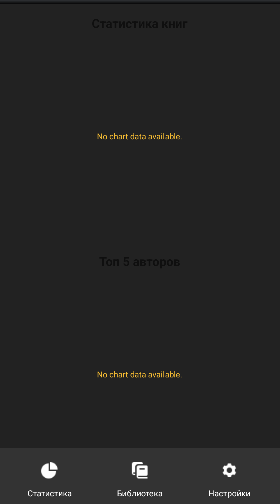


Рисунок 20 – Экран статистики

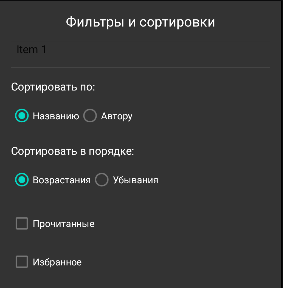


Рисунок 21 – Меню фильтров

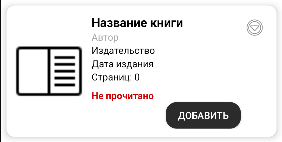


Рисунок 22 – Карточка книги

## 2.3 Разработка логики работы программы. Схема взаимодействия компонентов проекта

### 2.3.1 Логика работы приложения

Логика работы мобильного приложения «MyLib» основана на модульном подходе, где каждый компонент отвечает за выполнение отдельной задачи. Приложение реализует основные функции управления личной библиотекой: поиск, добавление, удаление и учет прочитанных книг. Все данные сохраняются локально, с возможностью обращения к внешнему API (Google Books) для получения информации о книгах.[5]

Основные этапы взаимодействия пользователя с приложением:

* Поиск книги. Пользователь вводит ключевые слова (название, автор и т.д.). Приложение отправляет запрос к Google Books API и отображает список найденных книг.[5]
* Добавление книги в библиотеку. Пользователь выбирает книгу из результатов поиска и сохраняет её в локальную базу данных приложения.
* Управление библиотекой. В интерфейсе отображается список сохранённых книг. Пользователь может удалить книгу или пометить её как прочитанную, указав дату прочтения.
* Работа с локальной базой данных. Вся информация о книгах сохраняется в локальной базе Room, что обеспечивает быстрый доступ к данным без подключения к интернету.
* Обработка ошибок и отказоустойчивость. Приложение обрабатывает возможные ошибки соединения с интернетом и API, уведомляя пользователя и позволяя повторить операцию.

### 2.3.2 Схема взаимодействия компонентов

Ниже представлена схема взаимодействия основных компонентов приложения:

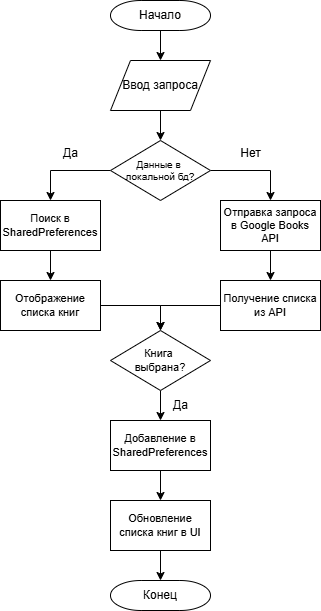


Рисунок 18 – Блок схема взаимодействия основных компонентов приложения

## 2.4 Описание переменных, компонентов, классов и подпрограмм

Таблица 1 – Описание переменных, компонентов, классов и подпрограмм

|  |  |  |
| --- | --- | --- |
| Тип | Название | Описание |
| Класс | BookViewHolder | Внутренний класс ViewHolder, который содержит ссылки на элементы интерфейса каждой карточки книги. |
| SearchBooksActivity | Активность, отвечающая за поиск книг через Google Books API и отображение результатов. |
| MainActivity | Главная activity приложения. Управляет отображением списка книг, фильтрацией, сортировкой и UI. |
| BookAdapter | Адаптер для отображения списка книг в RecyclerView. Обрабатывает отображение, взаимодействие и сохранение данных. |
| Переменные | isSearchMode | Флаг, указывающий, включен ли режим поиска. |
| isMainMode | Флаг, указывающий, является ли текущий режим основным. |
| books | Список книг, отображаемых в адаптере. |
| allBooks | Список всех книг (до фильтрации). |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
|  | context | Контекст приложения для доступа к ресурсам и UI. |
| filteredBooks | Список книг после применения фильтров. |
| currentSortBy | Текущий критерий сортировки (title или author). |
| currentSortOrder | Порядок сортировки (asc или desc). |
| sharedPreferences | SharedPreferences для сохранения данных пользователя. |
| bookAdapter | Адаптер для отображения книг в RecyclerView. |
| books | Локальный список книг. |
| currentGenre | Текущий выбранный жанр книг. |
| isReadable | Флаг: отображать читаемые книги. |
| isRead | Флаг: отображать прочитанные книги. |
| isFavorite | Флаг: отображать избранные книги. |
| isMenuVisible | Отвечает за отображение меню (меняется через Delegates.observable). |
| binding | ViewBinding для активити. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
|  | searchEditText | Поле ввода текста для поиска книг. |
| searchButton | Кнопка запуска поиска. |
| btnBack | Кнопка возврата назад из активности. |
| booksRecyclerView | RecyclerView, отображающий список найденных книг. |
| emptyTextView | Текстовое поле, отображающее сообщение при отсутствии результатов. |
| progressBar | Индикатор загрузки данных. |
| bookAdapter | Адаптер для списка книг. |
| requestQueue | Очередь запросов Volley для работы с сетью. |
| isLoading | Флаг, указывающий, происходит ли сейчас загрузка данных. |
| currentPage | Текущая страница пагинации для загрузки данных. |
| pageSize | Количество книг, загружаемых за один запрос. |
| query | Строка запроса, введённая пользователем. |
| Компаньон объект | companion object | Содержит статические значения и методы. |
| Переменные (в companion) | REQUEST\_CODE\_NOTES | Код запроса для заметок. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
|  | SCAN\_ISBN\_REQUEST\_CODE | Код запроса для сканирования ISBN. |
|  | CAMERA\_PERMISSION\_REQUEST\_CODE | Код запроса разрешения камеры. |
| instance | Статическая ссылка на экземпляр MainActivity. |
| Компоненты | titleTextView | TextView для отображения названия книги. |
| authorsTextView | TextView для отображения авторов книги. |
| thumbnailImageView | ImageView для отображения обложки книги. |
| publisherTextView | TextView для отображения издательства. |
| publishedDateTextView | TextView для отображения даты издания. |
| pageCountTextView | TextView для отображения количества страниц |
| moreOptionsButton | ImageButton для открытия меню с дополнительными действиями. |
| addBookButton | Button для добавления книги в избранное. |
| imgBtnBack | Кнопка "назад" с возможностью смены стиля через SharedPreferences. |
| edtSearch | Поле ввода поиска с возможностью изменения фона. |
| btnSearch | Кнопка "поиск", стилизуемая через SharedPreferences. |
| mainLayout | Основной layout активности, фон которого можно изменить. |

# 3 Тестирование и установка программы

* Установка приложения

Для установки программы необходимо скачать .apk файл приложения.

После установки приложение сразу готово к использованию и может быть протестировано.

* Тестирование программы

Таблица 2 – Тест-кейсы к программе

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название тест-кейса | Описание | Ожидаемый результат |
| 1 | Поиск книги по названию | Ввести в поле поиска корректное название книги и нажать кнопку поиска. | Отображается список книг, соответствующих запросу. |
| 2 | Поиск по пустому полю | Оставить поле поиска пустым и нажать кнопку. | Появляется предупреждение или ничего не происходит. |
| 3 | Нажатие на кнопку "Назад" | Нажать на кнопку назад в верхней части экрана. | Приложение закрывает текущую активность и возвращается назад. |
| 4 | Прокрутка списка книг | Прокрутить список найденных книг до конца. | Подгружается следующая порция данных (пагинация работает). |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | Клик по элементу книги в списке | Нажать на найденную книгу в списке. | Открывается подробная информация или предпросмотр книги. |



Рисунок 19 – Результат тест-кейса 1

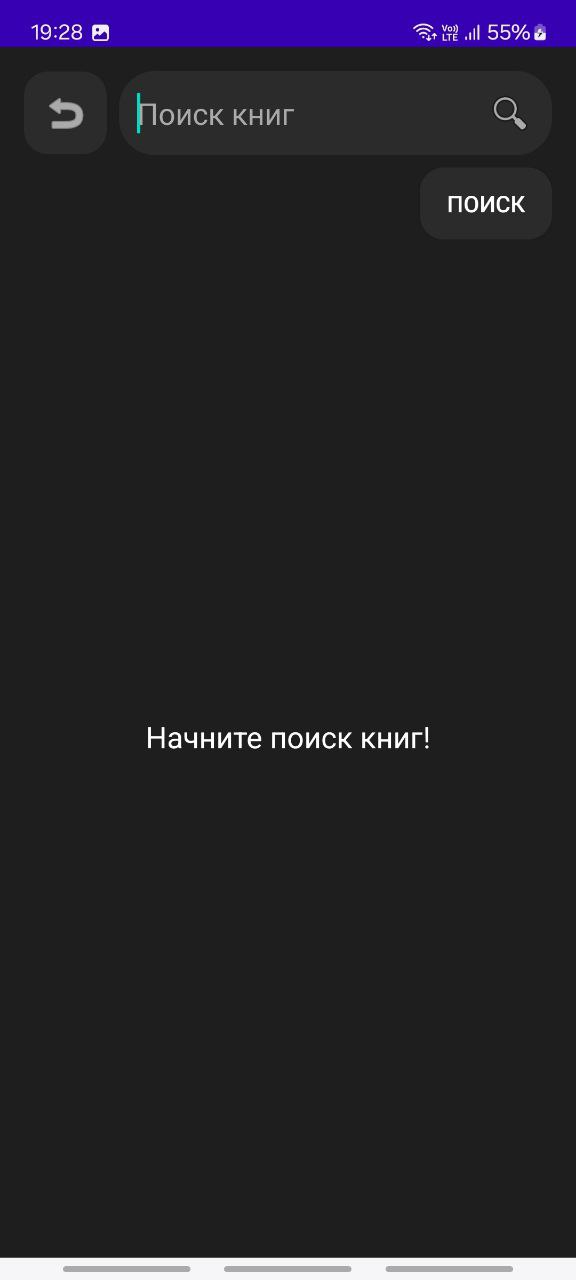


Рисунок 20 – Результат тест-кейса 2

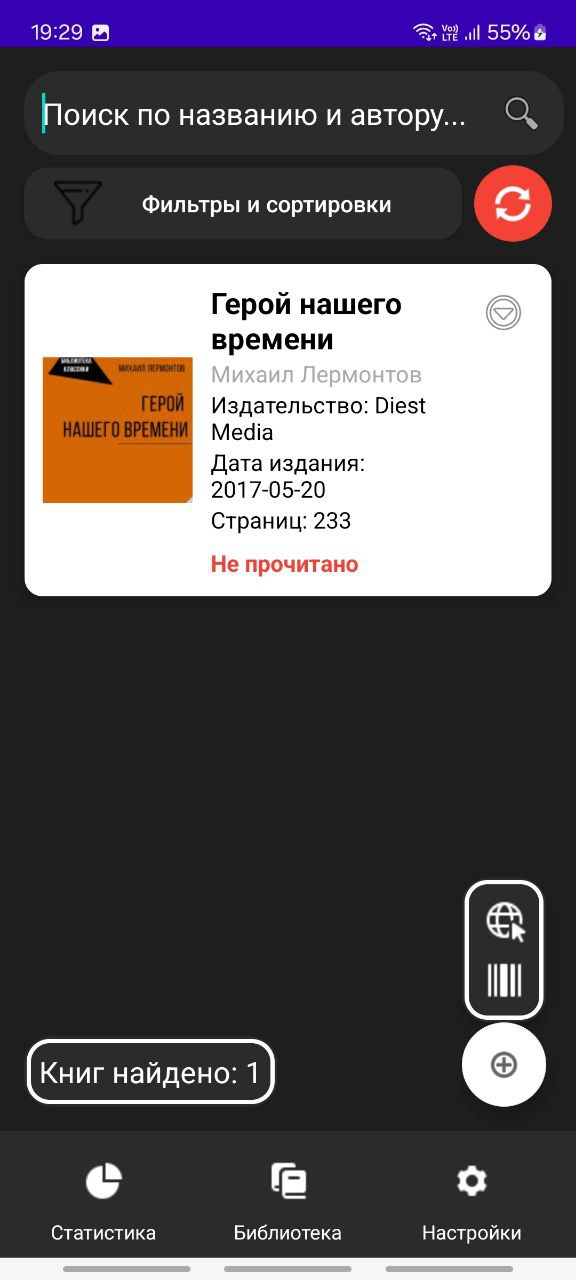


Рисунок 21 – Результат тест-кейса 3

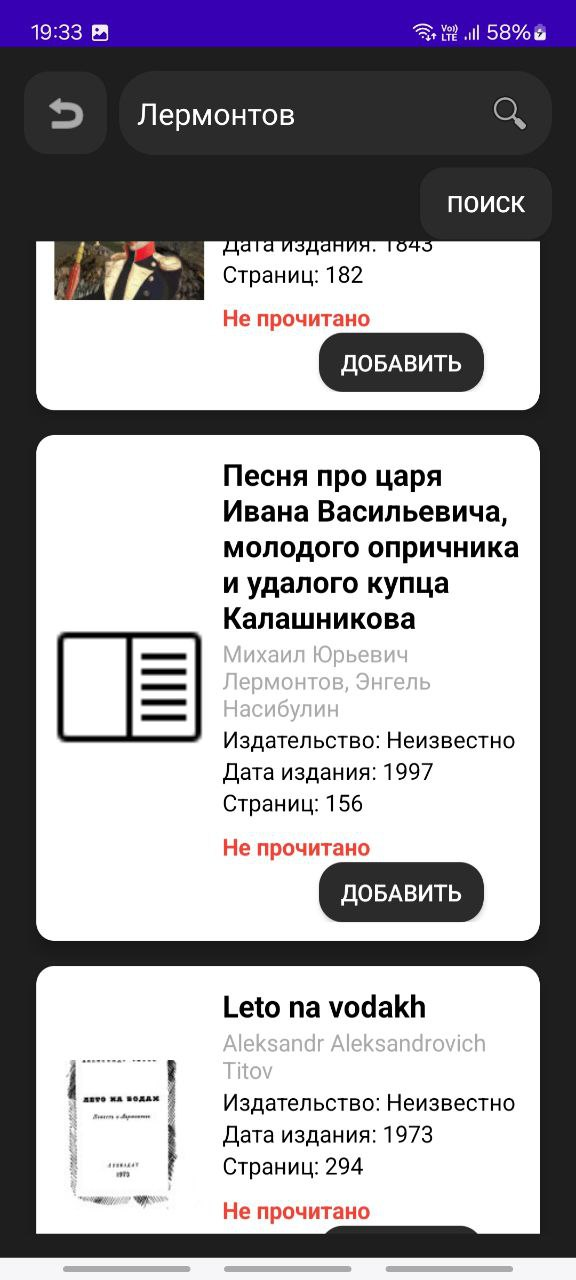


Рисунок 22 – Результат тест-кейса 4

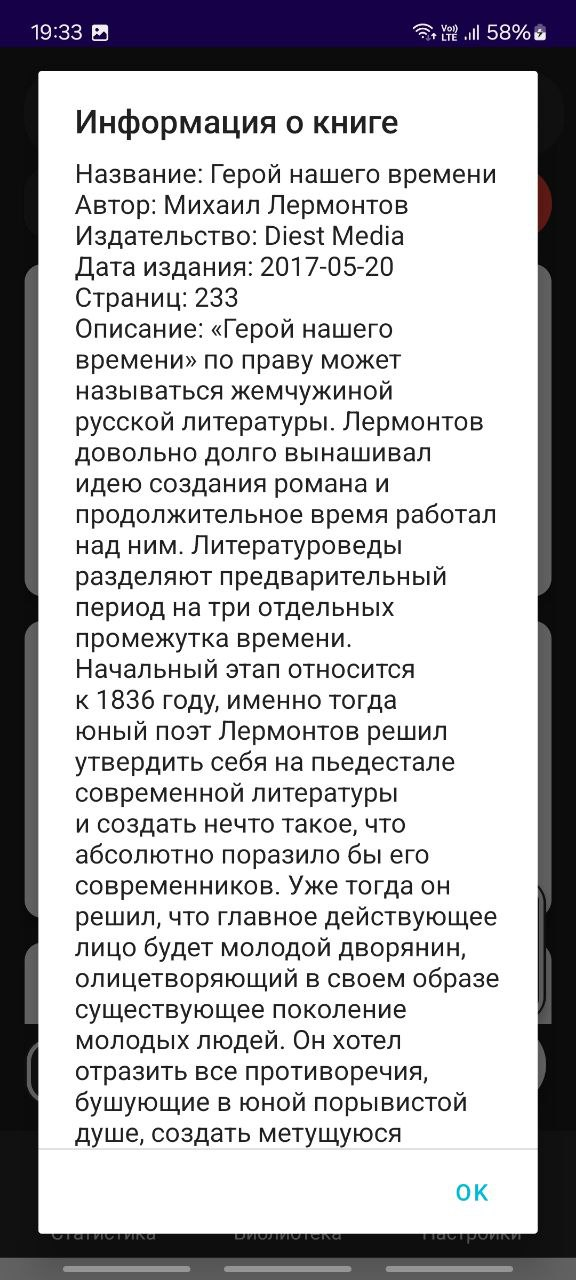


Рисунок 23 – Результат тест-кейса 5

После тестирования, можно сделать вывод, что все тест кейсы были успешны и программа работает правильно.

# Заключение

В рамках курсового проекта было успешно разработано и протестировано мобильное приложение управления личной библиотекой «MyLib», созданное с использованием среды Android Studio и языка программирования Kotlin.

Разработка охватывала все ключевые этапы жизненного цикла программного обеспечения: анализ требований, проектирование архитектуры, реализация пользовательского интерфейса и логики приложения, подключение к внешним API (Google Books API), а также реализация локального хранения данных с помощью SharedPreferences.

Приложение предоставляет пользователю удобный способ поиска, добавления и хранения информации о прочитанных или планируемых к прочтению книгах. Реализованы функции фильтрации, сортировки, ведения заметок, добавления в избранное и отображения детальной информации о книгах. Также предусмотрены визуальные настройки темы оформления.

Проведённое тестирование показало корректную работу основных функций приложения, стабильную интеграцию с внешним API и положительный пользовательский опыт.

# Список использованной литературы

1. Android Jetpack Documentation [Электронный ресурс]. — Режим доступа: https://developer.android.com/jetpack, свободный. — Дата обращения: 20.04.2025.
2. Брашнаров А., Миленковски В. Kotlin. Программирование для Android. — М.: ДМК Пресс, 2020. — 480 с.
3. GSON: JSON-парсер от Google [Электронный ресурс]. — Режим доступа: https://github.com/google/gson, свободный. — Дата обращения: 20.04.2025.
4. GitHub. Примеры реализации RecyclerView и Adapter [Электронный ресурс]. — Режим доступа: https://github.com, свободный. — Дата обращения: 20.04.2025.
5. Google Books API Documentation [Электронный ресурс]. — Режим доступа: https://developers.google.com/books, свободный. — Дата обращения: 20.04.2025.
6. Официальная документация Volley [Электронный ресурс]. — Режим доступа: https://developer.android.com/training/volley, свободный. — Дата обращения: 20.04.2025.
7. Официальная документация разработчика Android [Электронный ресурс]. — Режим доступа: https://developer.android.com, свободный. — Дата обращения: 20.04.2025.
8. Официальный сайт Kotlin для Android [Электронный ресурс]. — Режим доступа: https://developer.android.com/kotlin, свободный. — Дата обращения: 20.04.2025.
9. Stack Overflow — ответы на вопросы по Android и Kotlin [Электронный ресурс]. — Режим доступа: https://stackoverflow.com, свободный. — Дата обращения: 20.04.2025.

# Приложение А

# Листинг программы

MainActivity.kt

package com.example.mylib

import android.Manifest

import android.app.Activity

import android.content.Context

import android.content.Intent

import android.content.SharedPreferences

import android.content.pm.PackageManager

import android.graphics.Color

import android.os.Bundle

import android.text.Editable

import android.text.TextWatcher

import android.util.Log

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.view.animation.Animation

import android.view.animation.AnimationSet

import android.view.animation.RotateAnimation

import android.view.animation.ScaleAnimation

import android.widget.CheckBox

import android.widget.ImageButton

import android.widget.RadioButton

import android.widget.Spinner

import android.widget.Toast

import androidx.appcompat.app.AlertDialog

import androidx.appcompat.app.AppCompatActivity

import androidx.core.app.ActivityCompat

import androidx.core.content.ContextCompat

import androidx.recyclerview.widget.LinearLayoutManager

import com.example.mylib.databinding.ActivityMainBinding

import com.google.android.material.floatingactionbutton.FloatingActionButton

import com.google.gson.Gson

import com.google.gson.reflect.TypeToken

import kotlin.properties.Delegates

class MainActivity : AppCompatActivity() {

private lateinit var allBooks: MutableList<Book>

private lateinit var filteredBooks: MutableList<Book>

private var currentSortBy: String = "title"

private var currentSortOrder: String = "asc"

private lateinit var sharedPreferences: SharedPreferences

private lateinit var bookAdapter: BookAdapter

private var books: MutableList<Book> = mutableListOf()

private var currentGenre: String = "Все жанры" // По умолчанию

private var isReadable: Boolean = false

private var isRead: Boolean = false

private var isFavorite: Boolean = false

private var isMenuVisible by Delegates.observable(false) { \_, \_, newValue ->

binding.menuContainer.visibility = if (newValue) View.VISIBLE else View.GONE

if (newValue) {

showMenu()

} else {

hideMenu()

}

}

private lateinit var binding: ActivityMainBinding

companion object {

private val REQUEST\_CODE\_NOTES = 1001

private const val SCAN\_ISBN\_REQUEST\_CODE = 1001

private const val CAMERA\_PERMISSION\_REQUEST\_CODE = 1002

var instance: MainActivity? = null

fun updateUiStatic(bookTitle: String) {

instance?.updateUI()

}

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

binding = ActivityMainBinding.inflate(layoutInflater)

setContentView(binding.root)

instance = this

// Загружаем состояние фильтров

loadFiltersState()

binding.searchEditText.addTextChangedListener(object : TextWatcher {

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

filterBooks(s.toString()) // Фильтруем книги при изменении текста

}

override fun afterTextChanged(s: Editable?) {}

})

sharedPreferences = getSharedPreferences("saved\_books", Context.MODE\_PRIVATE)

loadSavedBooks()

allBooks = loadSavedBooks().toMutableList()

filteredBooks = allBooks

setupSortAndFilter()

binding.booksRecyclerView.layoutManager = LinearLayoutManager(this)

bookAdapter = BookAdapter(

loadSavedBooks().toMutableList(), // Список книг

false, // Не в режиме поиска (сокрытие кнопки)

true, // В главном режиме (кнопка добавления скрыта)

this

)

binding.booksRecyclerView.adapter = bookAdapter

updateUI()

val imageButton1 = findViewById<ImageButton>(R.id.imageButton1)

imageButton1.setOnClickListener {

val intent = Intent(this, SearchBooksActivity::class.java)

startActivity(intent)

}

val imageButton2 = findViewById<ImageButton>(R.id.imageButton2)

imageButton2.setOnClickListener {

if (ContextCompat.checkSelfPermission(

this,

Manifest.permission.CAMERA

) == PackageManager.PERMISSION\_GRANTED

) {

// Разрешение есть, запускаем сканирование

startScanISBN()

} else {

// Запрашиваем разрешение

ActivityCompat.requestPermissions(

this,

arrayOf(Manifest.permission.CAMERA),

CAMERA\_PERMISSION\_REQUEST\_CODE

)

}

}

// Загружаем тему

val sharedPref = getSharedPreferences("app\_theme", Context.MODE\_PRIVATE)

val bgColor = sharedPref.getInt("bg\_color", Color.WHITE)

val bottomNavColor = sharedPref.getInt("bottom\_nav\_color", Color.LTGRAY)

val textViewBack = sharedPref.getString("txt\_book", "txt\_dark\_gray")

val txtBook = resources.getIdentifier(textViewBack, "drawable", packageName)

val edtTextBack = sharedPref.getString("edt\_text", "rounded\_edittext")

val edtText = resources.getIdentifier(edtTextBack, "drawable", packageName)

val fltBtnBack = sharedPref.getString("flt\_btn", "rounded\_button")

val fltBtn = resources.getIdentifier(fltBtnBack, "drawable", packageName)

if (fltBtn != 0) {

binding.filterButton.setBackgroundResource(fltBtn)

binding.imageButton1.setBackgroundResource(fltBtn)

binding.imageButton2.setBackgroundResource(fltBtn)

}

if (edtText != 0) {

binding.searchEditText.setBackgroundResource(edtText)

}

if (txtBook != 0) {

binding.booksFoundTextView.setBackgroundResource(txtBook)

binding.menuContainer.setBackgroundResource(txtBook)

}

// Применяем цвета

binding.imageButton1.setBackgroundResource(fltBtn)

binding.imageButton2.setBackgroundResource(fltBtn)

binding.main.setBackgroundColor(bgColor)

binding.bottomNavigation.setBackgroundColor(bottomNavColor)

binding.booksFoundTextView.setBackgroundResource(txtBook)

binding.searchEditText.setBackgroundResource(edtText)

binding.filterButton.setBackgroundResource(fltBtn)

val addButton = findViewById<FloatingActionButton>(R.id.addButton)

binding.filterButton.setOnClickListener {

showFilterDialog()

}

binding.refreshButton.setOnClickListener {

animateRefreshButton()

}

addButton.setOnClickListener {

isMenuVisible = !isMenuVisible // Переключение состояния видимости меню

}

}

private fun showMenu() {

binding.menuContainer.alpha = 0f // Начальная прозрачность контейнера

binding.menuContainer.animate().alpha(1f).setDuration(300)

.start() // Анимация появления контейнера

}

private fun hideMenu() {

binding.menuContainer.animate().alpha(0f).setDuration(300).withEndAction {

binding.menuContainer.visibility = View.GONE // Скрываем контейнер после анимации

}.start()

}

private fun filterBooks(query: String) {

// Применяем фильтрацию по запросу

val filteredList = allBooks.filter { book ->

book.title.contains(query, ignoreCase = true) ||

book.authors.contains(query, ignoreCase = true) // Проверка по строке авторов

}.toMutableList() // Преобразуем результат в MutableList

// Обновляем адаптер с отфильтрованными книгами

bookAdapter.updateBooks(filteredList)

// Обновляем текст с количеством найденных книг

binding.booksFoundTextView.text = if (filteredList.isEmpty()) {

"Ничего не найдено"

} else {

"Книг найдено: ${filteredList.size}"

}

}

private fun animateRefreshButton() {

updateUI()

// 1. Rotate Animation

val rotateAnimation = RotateAnimation(

0f, 360f,

Animation.RELATIVE\_TO\_SELF, 0.5f,

Animation.RELATIVE\_TO\_SELF, 0.5f

)

rotateAnimation.duration = 500 // Half a second for rotation

rotateAnimation.repeatCount = 0

rotateAnimation.fillAfter = false // Don't keep the final rotation state

// 2. Scale Animation (Increase and then decrease)

val scaleAnimation = ScaleAnimation(

1f, 1.2f, // fromX, toX (1f = original size, 1.2f = 120% size)

1f, 1.2f, // fromY, toY

Animation.RELATIVE\_TO\_SELF, 0.5f, // Pivot X

Animation.RELATIVE\_TO\_SELF, 0.5f // Pivot Y

)

scaleAnimation.duration = 200 // Quarter second to scale up

scaleAnimation.repeatMode = Animation.REVERSE // Scale back down

scaleAnimation.repeatCount = 1 // Only scale up and then back down once

// 3. Combine Animations

val animationSet = AnimationSet(true) // true = share interpolator

animationSet.addAnimation(rotateAnimation)

animationSet.addAnimation(scaleAnimation)

binding.refreshButton.startAnimation(animationSet)

}

private fun showFilterDialog() {

// Восстановление состояния фильтров

loadFiltersState()

val builder = AlertDialog.Builder(this, R.style.CustomAlertDialog)

val inflater = LayoutInflater.from(this)

val dialogView: View = inflater.inflate(R.layout.filter\_dialog, null)

builder.setView(dialogView)

// Инициализация компонентов диалога

val genreSpinner: Spinner = dialogView.findViewById(R.id.genreSpinner)

val sortByTitleRadioButton: RadioButton = dialogView.findViewById(R.id.sortByTitleRadioButton)

val sortByAuthorRadioButton: RadioButton = dialogView.findViewById(R.id.sortByAuthorRadioButton)

val sortOrderAscRadioButton: RadioButton = dialogView.findViewById(R.id.sortOrderAscRadioButton)

val sortOrderDescRadioButton: RadioButton = dialogView.findViewById(R.id.sortOrderDescRadioButton)

val readCheckBox: CheckBox = dialogView.findViewById(R.id.readCheckBox)

val favoriteCheckBox: CheckBox = dialogView.findViewById(R.id.favoriteCheckBox)

// Устанавливаем сохраненное состояние кнопок

sortByTitleRadioButton.isChecked = currentSortBy == "title"

sortByAuthorRadioButton.isChecked = currentSortBy == "author"

sortOrderAscRadioButton.isChecked = currentSortOrder == "asc"

sortOrderDescRadioButton.isChecked = currentSortOrder == "desc"

readCheckBox.isChecked = isRead

favoriteCheckBox.isChecked = isFavorite

builder.setPositiveButton("Применить") { dialog, \_ ->

// Получаем выбранный жанр

val selectedGenre = genreSpinner.selectedItem?.toString() ?: "Все жанры"

// Получаем выбранное значение сортировки

val sortBy = when {

sortByTitleRadioButton.isChecked -> "title"

sortByAuthorRadioButton.isChecked -> "author"

else -> "title"

}

val sortOrder = if (sortOrderAscRadioButton.isChecked) "asc" else "desc"

// Сохраняем состояние фильтров

currentSortBy = sortBy

currentSortOrder = sortOrder

currentGenre = selectedGenre

isRead = readCheckBox.isChecked

isFavorite = favoriteCheckBox.isChecked

// Сохраняем состояние в SharedPreferences

saveFiltersState()

// Применяем фильтры

applyFilters(selectedGenre, sortBy, sortOrder, isReadable, isRead, isFavorite)

// Закрытие диалога

dialog.dismiss()

}

builder.setNegativeButton("Отмена") { dialog, \_ -> dialog.dismiss() }

// Отображаем диалог

val alertDialog = builder.create()

alertDialog.show()

// Устанавливаем ширину диалога на 90% от ширины экрана

val screenWidth = resources.displayMetrics.widthPixels

alertDialog.window?.setLayout((screenWidth \* 0.9).toInt(), ViewGroup.LayoutParams.WRAP\_CONTENT)

}

// Обновляем адаптер с фильтрацией по состоянию `isRead`

private fun applyFilters(

genre: String,

sortBy: String,

sortOrder: String,

isReadable: Boolean,

isRead: Boolean,

isFavorite: Boolean

) {

var filteredBooks: MutableList<Book> = allBooks.toMutableList()

// Фильтрация по жанру

if (genre != "Все жанры") {

filteredBooks = filteredBooks.filter { it.genre == genre }.toMutableList()

}

// Фильтрация по статусу читаемости (книги, которые читаются)

if (isReadable) {

filteredBooks = filteredBooks.filter { it.startDate != null && it.endDate == null }.toMutableList()

}

// Фильтрация по состоянию прочтения

if (isRead) {

filteredBooks = filteredBooks.filter { it.isRead }.toMutableList()

}

// Фильтрация по избранному

if (isFavorite) {

filteredBooks = filteredBooks.filter { book -> book.isFavorite }.toMutableList()

}

// Сортировка

filteredBooks = when (sortBy) {

"title" -> filteredBooks.sortedBy { it.title }.toMutableList()

"author" -> filteredBooks.sortedBy { it.authors }.toMutableList()

else -> filteredBooks.sortedBy { it.title }.toMutableList()

}

// Применяем сортировку по порядку

if (sortOrder == "desc") {

filteredBooks.reverse()

}

// Дополнительно сортируем, чтобы избранные были первыми

filteredBooks = filteredBooks.sortedByDescending { it.isFavorite }.toMutableList()

// Обновляем UI

bookAdapter.updateBooks(filteredBooks)

Log.d(

"ApplyFilters",

"Applied filters: genre=$genre, sortBy=$sortBy, sortOrder=$sortOrder, " +

"isReadable=$isReadable, isRead=$isRead, isFavorite=$isFavorite"

)

}

private fun setupSortAndFilter() {

// Устанавливаем текущие значения сортировки и фильтров при открытии фильтра

binding.filterButton.setOnClickListener {

showFilterDialog()

}

}

private fun loadFiltersState() {

val sharedPreferences = getSharedPreferences("filters", MODE\_PRIVATE)

// Восстанавливаем сохраненное состояние фильтров

currentSortBy = sharedPreferences.getString("currentSortBy", "title") ?: "title"

currentSortOrder = sharedPreferences.getString("currentSortOrder", "asc") ?: "asc"

currentGenre = sharedPreferences.getString("currentGenre", "Все жанры") ?: "Все жанры"

isReadable = sharedPreferences.getBoolean("isReadable", false)

isRead = sharedPreferences.getBoolean("isRead", false)

isFavorite = sharedPreferences.getBoolean("isFavorite", false)

}

private fun saveFiltersState() {

val sharedPreferences = getSharedPreferences("filters", MODE\_PRIVATE)

val editor = sharedPreferences.edit()

// Сохраняем состояние фильтров и чекбоксов

editor.putString("currentSortBy", currentSortBy)

editor.putString("currentSortOrder", currentSortOrder)

editor.putString("currentGenre", currentGenre)

editor.putBoolean("isReadable", isReadable)

editor.putBoolean("isRead", isRead)

editor.putBoolean("isFavorite", isFavorite)

editor.apply()

}

fun onNavButtonClick(view: View) {

when (view.id) {

R.id.nav\_home -> {

startActivity(Intent(this, StatisticsActivity::class.java))

overridePendingTransition(R.anim.slide\_fade\_in\_left, R.anim.slide\_fade\_out\_left)

}

R.id.nav\_profile -> {

startActivity(Intent(this, SettingsActivity::class.java))

overridePendingTransition(R.anim.slide\_fade\_in\_right, R.anim.slide\_fade\_out\_right)

}

}

}

override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults)

if (requestCode == CAMERA\_PERMISSION\_REQUEST\_CODE) {

if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

// Разрешение получено, запускаем сканирование

startScanISBN()

} else {

Log.e("MainActivity", "Разрешение на камеру отклонено")

}

}

}

private fun startScanISBN() {

val intent = Intent(this, ScanISBNActivity::class.java)

startActivityForResult(intent, SCAN\_ISBN\_REQUEST\_CODE)

}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

super.onActivityResult(requestCode, resultCode, data)

if (requestCode == SCAN\_ISBN\_REQUEST\_CODE && resultCode == Activity.RESULT\_OK) {

val scannedISBN = data?.getStringExtra("ISBN\_RESULT")

Log.d("MainActivity", "Получен ISBN: $scannedISBN")

if (!scannedISBN.isNullOrEmpty()) {

Toast.makeText(this, "Книга успешно найдена!", Toast.LENGTH\_SHORT).show()

val intent = Intent(this, SearchBooksActivity::class.java)

intent.putExtra("ISBN", scannedISBN)

startActivity(intent)

} else {

Toast.makeText(this, "Книга не найдена в базе данных!", Toast.LENGTH\_SHORT).show()

}

}

}

override fun onResume() {

super.onResume()

updateUI() // Обновляем список книг при возвращении в активити

}

private fun updateUI() {

// Получаем список сохраненных книг

val bookList = loadSavedBooks()

// Обновляем количество найденных книг

val sharedPreferences = getSharedPreferences("book\_count", Context.MODE\_PRIVATE)

val bookCount = sharedPreferences.getInt("count", 0)

binding.booksFoundTextView.text = "Книг найдено: $bookCount"

// Обновляем видимость элементов UI в зависимости от наличия книг

if (bookList.isEmpty()) {

binding.booksRecyclerView.visibility = View.GONE

binding.emptyTextView.visibility = View.VISIBLE

} else {

binding.booksRecyclerView.visibility = View.VISIBLE

binding.emptyTextView.visibility = View.GONE

bookAdapter.updateBooks(bookList)

}

}

private fun loadSavedBooks(): MutableList<Book> {

val json = sharedPreferences.getString("books", null)

Log.d("BookLoading", "Загружаем книги из SharedPreferences: $json")

return if (json != null) {

try {

val type = object : TypeToken<List<Book>>() {}.type

val books: List<Book> = Gson().fromJson(json, type) ?: emptyList()

books.sortedByDescending { it.isFavorite }.toMutableList()

Log.d("BookLoading", "Загруженные книги: $books")

// Convert List<Book> to MutableList<Book>

books.map { book ->

// Логируем поля книги

Log.d("BookLoading", "Книга: ${book.title}, Издательство: ${book.publisher}, Дата издания: ${book.publishedDate}, Страниц: ${book.pageCount}")

val publisher = book.publisher ?: "Неизвестно"

val publishedDate = book.publishedDate ?: "Не указано"

val pageCount = book.pageCount.takeIf { it > 0 } ?: 0

// Передаем параметр isRead, возможно с дефолтным значением

Book(

title = book.title,

authors = book.authors,

thumbnail = book.thumbnail,

description = book.description ?: "",

note = book.note,

genre = book.genre,

startDate = book.startDate,

endDate = book.endDate,

publisher = publisher,

publishedDate = publishedDate,

pageCount = pageCount,

previewLink = "",

webReaderLink = "",

isFavorite = book.isFavorite

)

}.toMutableList()

} catch (e: Exception) {

Log.e("BookLoading", "Ошибка при загрузке книг: ${e.message}")

mutableListOf()

}

} else {

Log.d("BookLoading", "Нет данных в SharedPreferences, возвращаем пустой список.")

mutableListOf()

}

}

}

BookAdapter.kt

package com.example.mylib

import android.content.Context

import android.content.Intent

import android.graphics.Color

import android.net.Uri

import android.util.Log

import android.view.\*

import android.widget.\*

import androidx.activity.result.ActivityResultLauncher

import androidx.appcompat.app.AlertDialog

import androidx.core.content.ContextCompat

import androidx.recyclerview.widget.RecyclerView

import com.bumptech.glide.Glide

import com.google.gson.Gson

import com.google.gson.reflect.TypeToken

import org.json.JSONObject

import java.io.BufferedReader

import java.io.InputStreamReader

import java.net.HttpURLConnection

import java.net.URL

import java.time.LocalDate

import kotlin.concurrent.thread

class BookAdapter(

private var books: MutableList<Book>,

private val isSearchMode: Boolean,

private val isMainMode: Boolean,

private val context: Context

) : RecyclerView.Adapter<BookAdapter.BookViewHolder>() {

class BookViewHolder(view: View) : RecyclerView.ViewHolder(view) {

val titleTextView: TextView = view.findViewById(R.id.bookTitle)

val authorsTextView: TextView = view.findViewById(R.id.bookAuthors)

val thumbnailImageView: ImageView = view.findViewById(R.id.bookThumbnail)

val publisherTextView: TextView = view.findViewById(R.id.bookPublisher)

val publishedDateTextView: TextView = view.findViewById(R.id.bookPublishedDate)

val pageCountTextView: TextView = view.findViewById(R.id.bookPageCount)

val moreOptionsButton: ImageButton = view.findViewById(R.id.moreOptionsButton)

val addBookButton: Button = view.findViewById(R.id.addBookButton)

val bookReadStatus: TextView = itemView.findViewById(R.id.bookReadStatus)

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BookViewHolder {

val view = LayoutInflater.from(parent.context).inflate(R.layout.item\_book, parent, false)

return BookViewHolder(view)

}

override fun onBindViewHolder(holder: BookViewHolder, position: Int) {

val book = books[position]

book.updateReadStatus()

holder.titleTextView.text = book.title

holder.authorsTextView.text = book.authors

holder.publisherTextView.text = "Издательство: ${book.publisher}"

holder.publishedDateTextView.text = "Дата издания: ${book.publishedDate}"

holder.pageCountTextView.text = "Страниц: ${book.pageCount}"

holder.itemView.setOnClickListener {

showBookDetails(context, book)

}

// Используем `isRead` для статуса прочитанности

holder.bookReadStatus.text = if (book.isRead) "Прочитано" else "Не прочитано"

holder.bookReadStatus.setTextColor(

ContextCompat.getColor(

holder.itemView.context,

if (book.isRead) R.color.colorGreenPrimary else R.color.colorRedPrimary

)

)

Log.d("BookAdapter", "Книга: ${book.title}, endDate: ${book.endDate}, isRead: ${book.isRead}")

// Загружаем изображение книги через Glide

Glide.with(holder.thumbnailImageView.context)

.load(book.thumbnail.replace("http://", "https://"))

.placeholder(R.drawable.book\_placeholder)

.error(R.drawable.book\_placeholder)

.into(holder.thumbnailImageView)

holder.addBookButton.setOnClickListener {

val context = holder.itemView.context

if (saveBookToPreferences(context, book)) {

Toast.makeText(context, "Книга добавлена в избранное!", Toast.LENGTH\_SHORT).show()

} else {

Toast.makeText(context, "Эта книга уже добавлена!", Toast.LENGTH\_SHORT).show()

}

}

// В адаптере для каждого элемента мы добавляем обработку состояния чекбокса

Log.d(

"BookAdapter",

"WebReaderLink: ${book.webReaderLink}, PreviewLink: ${book.previewLink}"

)

if (isSearchMode) {

holder.moreOptionsButton.visibility = View.GONE

}

if (isMainMode) {

holder.addBookButton.visibility = View.GONE

}

holder.moreOptionsButton.setOnClickListener {

val context = holder.itemView.context

val popupMenu = PopupMenu(context, it)

popupMenu.inflate(R.menu.book\_options\_menu)

popupMenu.setOnMenuItemClickListener { menuItem ->

when (menuItem.itemId) {

R.id.menu\_add\_note -> {

val intent = Intent(context, NotesActivity::class.java)

intent.putExtra("BOOK\_TITLE", book.title)

intent.putExtra("IS\_READ", book.isRead)

context.startActivity(intent) // Добавляем запуск активности

true

}

R.id.menu\_delete -> {

removeBook(position, context)

true

}

R.id.menu\_favorite -> {

pinBookToTop(position)

true

}

R.id.menu\_read\_on\_google\_books -> {

fetchBookLink(context, book.title) { webReaderLink ->

if (!webReaderLink.isNullOrEmpty()) {

val intent = Intent(Intent.ACTION\_VIEW, Uri.parse(webReaderLink))

context.startActivity(intent)

} else {

Toast.makeText(

context,

"Ссылка для чтения отсутствует",

Toast.LENGTH\_SHORT

).show()

}

}

true

}

else -> false

}

}

popupMenu.show()

}

}

override fun getItemCount() = books.size

fun clearBooks() {

books.clear()

notifyDataSetChanged()

}

fun updateBooks(newBooks: MutableList<Book>) {

val sharedPreferences = context.getSharedPreferences("book\_notes", Context.MODE\_PRIVATE)

newBooks.forEach { book ->

val savedEndDate = sharedPreferences.getString("${book.title}\_end\_date\_parsed", null)

if (!savedEndDate.isNullOrEmpty()) {

book.endDate = LocalDate.parse(savedEndDate)

}

book.updateReadStatus()

}

books = newBooks

notifyDataSetChanged()

}

fun addBooks(newBooks: List<Book>) {

val sharedPreferences = context.getSharedPreferences("book\_notes", Context.MODE\_PRIVATE)

newBooks.forEach { book ->

val savedEndDate = sharedPreferences.getString("${book.title}\_end\_date\_parsed", null)

if (!savedEndDate.isNullOrEmpty()) {

book.endDate = LocalDate.parse(savedEndDate)

}

book.updateReadStatus()

}

val startPosition = books.size

books.addAll(newBooks)

notifyItemRangeInserted(startPosition, newBooks.size)

}

fun removeBook(position: Int, context: Context) {

val book = books[position]

books.removeAt(position)

notifyItemRemoved(position)

removeBookFromPreferences(context, book)

decrementBookCount(context)

}

fun pinBookToTop(position: Int) {

val book = books[position]

book.isFavorite = true // Отмечаем книгу как избранную

books.removeAt(position)

books.add(0, book) // Перемещаем в начало

books.sortByDescending { it.isFavorite } // Убедимся, что избранные сверху

notifyDataSetChanged()

saveBookListToPreferences()

}

private fun saveBookListToPreferences() {

val sharedPreferences = context.getSharedPreferences("saved\_books", Context.MODE\_PRIVATE)

val editor = sharedPreferences.edit()

val gson = Gson()

editor.putString("books", gson.toJson(books))

editor.apply()

}

fun updateBookStatus(bookTitle: String, isRead: Boolean) {

// Ищем книгу по её названию

val book = books.find { it.title == bookTitle }

book?.let {

it.isRead = isRead // Обновляем статус

notifyDataSetChanged() // Уведомляем адаптер, что данные изменились

}

}

private fun showBookDetails(context: Context, book: Book) {

val message = """

Название: ${book.title}

Автор: ${book.authors}

Издательство: ${book.publisher}

Дата издания: ${book.publishedDate}

Страниц: ${book.pageCount}

Описание: ${book.description}

""".trimIndent()

AlertDialog.Builder(context)

.setTitle("Информация о книге")

.setMessage(message)

.setPositiveButton("OK", null)

.show()

}

// Метод для сохранения книги в SharedPreferences

private fun saveBookToPreferences(context: Context, book: Book): Boolean {

val sharedPreferences = context.getSharedPreferences("saved\_books", Context.MODE\_PRIVATE)

val editor = sharedPreferences.edit()

val gson = Gson()

// Устанавливаем флаг isPinned как true (книга закреплена)

book.isFavorite = true

// Читаем текущий список книг из SharedPreferences

val json = sharedPreferences.getString("books", "[]")

val type = object : TypeToken<MutableList<Book>>() {}.type

val bookList: MutableList<Book> = gson.fromJson(json, type) ?: mutableListOf()

// Проверяем, есть ли книга в списке

if (bookList.any { it.title == book.title }) return false

// Добавляем книгу в список

bookList.add(book)

// Сортируем книги так, чтобы закрепленные были сверху

bookList.sortByDescending { it.isFavorite }

// Сохраняем обновленный список в SharedPreferences

editor.putString("books", gson.toJson(bookList))

editor.apply()

// Увеличиваем счетчик книг

incrementBookCount(context)

return true

}

// Изменение в методе на удаление книги

private fun removeBookFromPreferences(context: Context, book: Book) {

val sharedPreferences = context.getSharedPreferences("saved\_books", Context.MODE\_PRIVATE)

val editor = sharedPreferences.edit()

val gson = Gson()

val json = sharedPreferences.getString("books", "[]")

val type = object : TypeToken<MutableList<Book>>() {}.type

val bookList: MutableList<Book> = gson.fromJson(json, type) ?: mutableListOf()

// Если книга закреплена, просто снимаем флаг

bookList.find { it.title == book.title }?.let { it.isFavorite = false }

// Фильтруем список (если книга не закреплена, удаляем)

val updatedBookList = bookList.filter { it.isFavorite || it.title != book.title }

// Сохраняем обновленный список

editor.putString("books", gson.toJson(updatedBookList))

editor.apply()

// Уменьшаем счетчик книг

decrementBookCount(context)

}

// Метод для увеличения счетчика книг

private fun incrementBookCount(context: Context) {

val sharedPreferences = context.getSharedPreferences("book\_count", Context.MODE\_PRIVATE)

val currentCount = sharedPreferences.getInt("count", 0)

val editor = sharedPreferences.edit()

editor.putInt("count", currentCount + 1)

editor.apply()

}

// Метод для уменьшения счетчика книг

private fun decrementBookCount(context: Context) {

val sharedPreferences = context.getSharedPreferences("book\_count", Context.MODE\_PRIVATE)

val currentCount = sharedPreferences.getInt("count", 0)

val editor = sharedPreferences.edit()

if (currentCount > 0) {

editor.putInt("count", currentCount - 1)

}

editor.apply()

}

private fun fetchBookLink(context: Context, title: String, callback: (String?) -> Unit) {

thread {

try {

val apiKey = "AIzaSyDjkmHuXtJRPtNC6ZDDl\_V1FoOXJVkSNy8"

val query = title.replace(" ", "+")

val searchUrl = "https://www.googleapis.com/books/v1/volumes?q=$query&key=$apiKey"

val url = URL(searchUrl)

val connection = url.openConnection() as HttpURLConnection

connection.requestMethod = "GET"

connection.connect()

val reader = BufferedReader(InputStreamReader(connection.inputStream))

val responseText = reader.use { it.readText() }

val jsonObject = JSONObject(responseText)

val items = jsonObject.optJSONArray("items")

var webReaderLink: String? = null

if (items != null && items.length() > 0) {

val firstBook = items.getJSONObject(0)

val accessInfo = firstBook.optJSONObject("accessInfo")

webReaderLink =

accessInfo?.optString("webReaderLink", "")?.replace("http://", "https://")

}

(context as? android.app.Activity)?.runOnUiThread {

callback(webReaderLink)

}

} catch (e: Exception) {

Log.e("BookAdapter", "Ошибка загрузки книги: ${e.message}")

(context as? android.app.Activity)?.runOnUiThread {

callback(null)

}

}

}

}

}

SearchBooksActivity.kt

package com.example.mylib

import android.content.Context

import android.graphics.Color

import android.os.Bundle

import android.util.Log

import android.view.View

import android.widget.Button

import android.widget.EditText

import android.widget.ImageButton

import android.widget.LinearLayout

import android.widget.ProgressBar

import android.widget.TextView

import android.widget.Toast

import androidx.appcompat.app.AppCompatActivity

import androidx.recyclerview.widget.LinearLayoutManager

import androidx.recyclerview.widget.RecyclerView

import com.android.volley.Request

import com.android.volley.RequestQueue

import com.android.volley.Response

import com.android.volley.toolbox.JsonObjectRequest

import com.android.volley.toolbox.Volley

import com.google.gson.Gson

import com.google.gson.reflect.TypeToken

class SearchBooksActivity : AppCompatActivity() {

private lateinit var searchEditText: EditText

private lateinit var searchButton: Button

private lateinit var btnBack: ImageButton

private lateinit var booksRecyclerView: RecyclerView

private lateinit var emptyTextView: TextView

private lateinit var progressBar: ProgressBar

private lateinit var bookAdapter: BookAdapter

private lateinit var requestQueue: RequestQueue

private var isLoading = false

private var currentPage = 0

private val pageSize = 10

private var query = ""

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_search\_books)

val imgBtnBack = findViewById<ImageButton>(R.id.btnBack)

val edtSearch = findViewById<EditText>(R.id.searchEditText)

val btnSearch = findViewById<Button>(R.id.searchButton)

val mainLayout = findViewById<LinearLayout>(R.id.main)

val txt1 = findViewById<TextView>(R.id.emptyTextView)

// Загружаем тему

val sharedPref = getSharedPreferences("app\_theme", Context.MODE\_PRIVATE)

val bgColor = sharedPref.getInt("bg\_color", Color.WHITE)

val textColor = sharedPref.getInt("text\_color", Color.BLACK)

val fltBtnBack = sharedPref.getString("flt\_btn", "rounded\_button")

val fltBtn = resources.getIdentifier(fltBtnBack, "drawable", packageName)

val edtTextBack = sharedPref.getString("edt\_text", "rounded\_edittext")

val edtText = resources.getIdentifier(edtTextBack, "drawable", packageName)

if (fltBtn != 0) {

imgBtnBack?.setBackgroundResource(fltBtn)

btnSearch?.setBackgroundResource(fltBtn)

}

if (edtText != 0) {

edtSearch?.setBackgroundResource(edtText)

}

edtSearch?.setBackgroundResource(edtText)

mainLayout?.setBackgroundColor(bgColor)

txt1?.setTextColor(textColor)

imgBtnBack?.setBackgroundResource(fltBtn)

btnSearch?.setBackgroundResource(fltBtn)

searchEditText = findViewById(R.id.searchEditText)

searchButton = findViewById(R.id.searchButton)

btnBack = findViewById(R.id.btnBack)

booksRecyclerView = findViewById(R.id.recyclerView)

emptyTextView = findViewById(R.id.emptyTextView)

progressBar = findViewById(R.id.progressBar)

booksRecyclerView.layoutManager = LinearLayoutManager(this)

requestQueue = Volley.newRequestQueue(this)

bookAdapter = BookAdapter(mutableListOf(), true, false, this) // Передаем true для режима поиска

booksRecyclerView.adapter = bookAdapter

btnBack.setOnClickListener {

finish()

}

searchButton.setOnClickListener {

query = searchEditText.text.toString().trim()

if (query.isNotEmpty()) {

resetSearch()

searchBooks(query, currentPage)

}

}

booksRecyclerView.addOnScrollListener(object : RecyclerView.OnScrollListener() {

override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {

val layoutManager = recyclerView.layoutManager as LinearLayoutManager

val totalItemCount = layoutManager.itemCount

val lastVisibleItem = layoutManager.findLastVisibleItemPosition()

if (!isLoading && totalItemCount <= lastVisibleItem + pageSize) {

searchBooks(query, currentPage)

}

}

})

}

private fun resetSearch() {

emptyTextView.visibility = View.GONE

booksRecyclerView.visibility = View.VISIBLE

progressBar.visibility = View.VISIBLE

currentPage = 0

bookAdapter.clearBooks()

}

private fun searchBooks(query: String, page: Int) {

if (isLoading) return // Предотвращаем многократные запросы

isLoading = true

val url = "https://www.googleapis.com/books/v1/volumes?q=$query&startIndex=$page&maxResults=$pageSize&projection=full"

val request = JsonObjectRequest(Request.Method.GET, url, null,

Response.Listener { response ->

progressBar.visibility = View.GONE

isLoading = false

Log.d("SearchBooksActivity", "API Response: $response") // Логируем весь ответ

val items = response.optJSONArray("items")

if (items != null && items.length() > 0) {

val bookList = ArrayList<Book>()

for (i in 0 until items.length()) {

val bookJson = items.getJSONObject(i)

val volumeInfo = bookJson.optJSONObject("volumeInfo") ?: continue

val accessInfo = bookJson.optJSONObject("accessInfo") // Доступ к webReaderLink

val title = volumeInfo.optString("title", "Без названия")

val authorsArray = volumeInfo.optJSONArray("authors")

val authors = if (authorsArray != null && authorsArray.length() > 0) {

val authorsList = mutableListOf<String>()

for (j in 0 until authorsArray.length()) {

authorsList.add(authorsArray.getString(j))

}

authorsList.joinToString(", ")

} else {

"Неизвестный автор"

}

val imageLinks = volumeInfo.optJSONObject("imageLinks")

val thumbnail = imageLinks?.optString("thumbnail", "") ?: ""

val description = volumeInfo.optString("description", "")

val publisher = volumeInfo.optString("publisher", "Неизвестно")

val publishedDate = volumeInfo.optString("publishedDate", "Не указано")

val pageCount = volumeInfo.optInt("pageCount", 0)

val previewLink = volumeInfo.optString("previewLink", "")

// Получаем `webReaderLink`

var webReaderLink = accessInfo?.optString("webReaderLink") ?: ""

// Если `webReaderLink` отсутствует, пробуем сгенерировать его из `selfLink`

if (webReaderLink.isNullOrEmpty()) {

val selfLink = bookJson.optString("selfLink", "")

if (selfLink.isNotEmpty()) {

webReaderLink = "https://play.google.com/store/books/details?id=${selfLink.substringAfterLast("/")}"

}

}

Log.d("SearchBooksActivity", "webReaderLink: $webReaderLink") // Логируем webReaderLink

val book = Book(

title = title,

authors = authors,

thumbnail = thumbnail,

description = description,

publisher = publisher,

publishedDate = publishedDate,

pageCount = pageCount,

previewLink = previewLink,

webReaderLink = webReaderLink

)

bookList.add(book)

}

if (bookList.isEmpty()) {

Toast.makeText(this, "Книга не найдена в базе данных!", Toast.LENGTH\_SHORT).show()

} else {

bookAdapter.addBooks(bookList)

currentPage += pageSize

}

} else {

Toast.makeText(this, "Нет результатов по запросу!", Toast.LENGTH\_SHORT).show()

}

},

Response.ErrorListener { error ->

Log.e("SearchBooksActivity", "Ошибка загрузки данных: ${error.message}")

progressBar.visibility = View.GONE

isLoading = false

Toast.makeText(this, "Ошибка при поиске книги!", Toast.LENGTH\_SHORT).show()

})

requestQueue.add(request)

}

}

Приложение Б

Руководство пользователя

Для начала работы с приложением MyLib необходимо установить его на вашем смартфоне. Приложение предоставляется в виде исполняемого файла .apk и может быть запущено на смартфонах под управлением операционной системы Android.

После загрузки исполняемого файла MyLib.apk, запустите его, и вы попадёте на главный экран.

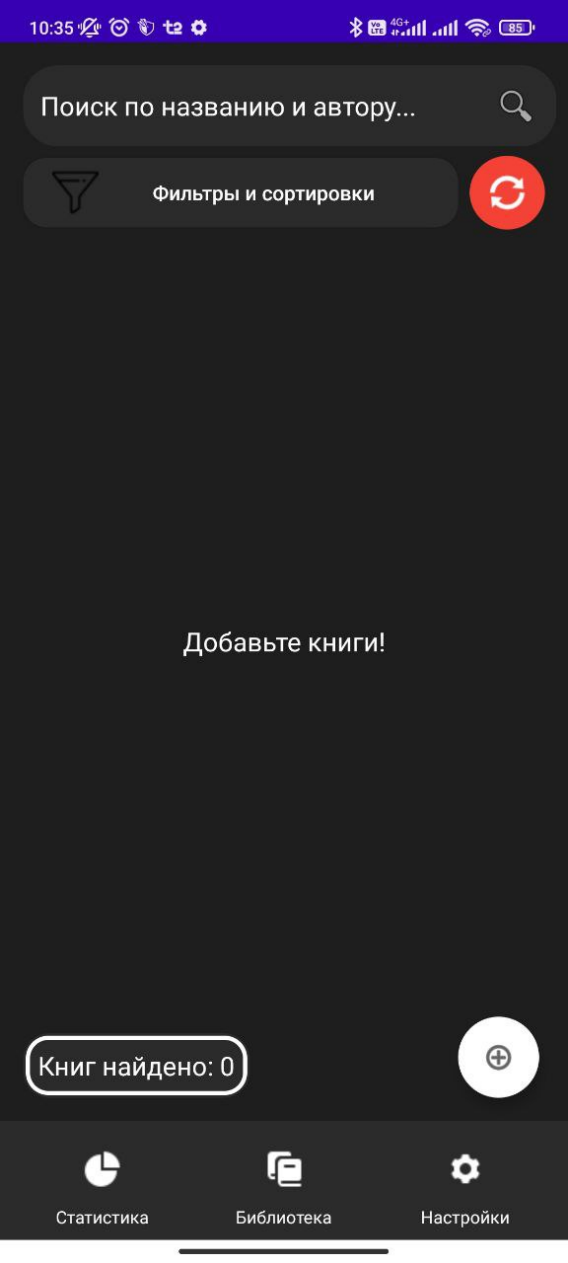


Рисунок 1 – Главный экран

Главный экран отображает список добавленных книг, кнопки:

* Фильтры и сортировки
* Обновление списка
* Меню добавления книг
* Поиск добавленных книг

Добавление книги: для добавления книги, нажмите на «+» и выберите первый пункт, после Вы перейдете на экран поиска книг.



Рисунок 2 – Экран поиска книг

Поиск книг: для поиска книг, в поле «Поиск книг» введите автора или название книги и нажмите кнопку «Поиск». Появится список книг по запросу, со всей информацией, и кнопкой «Добавить».

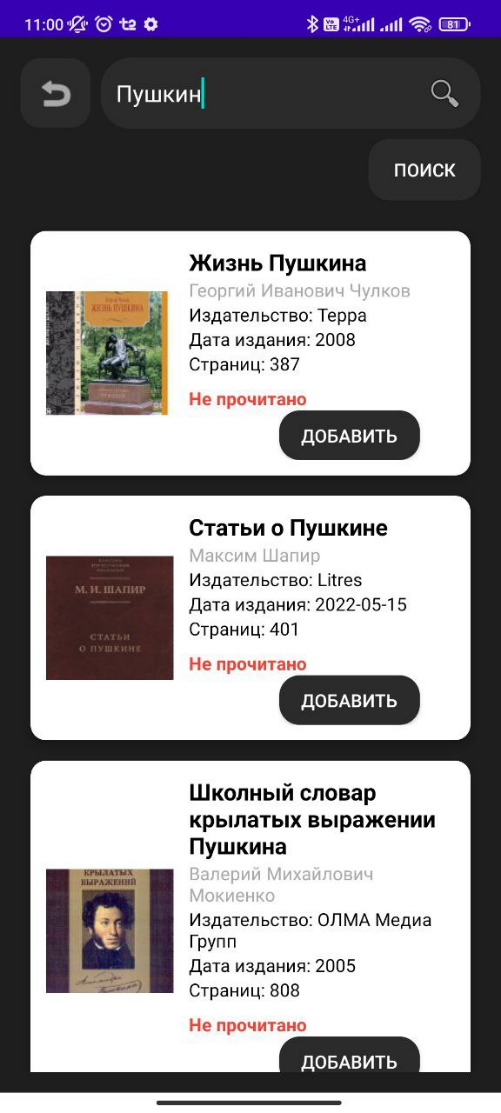


Рисунок 3 – Экран поиска книг со списком по запросу «Пушкин»

Добавление книг: для добавления книги нужно нажать кнопку «Добавить» на карточке книги, после книга будет добавлена на главный экран приложения.

Чтобы вернутся на главный экран, нажмите кнопку возврата в верхнем левом углу экрана.

Теперь на главном экране есть добавленная книга.

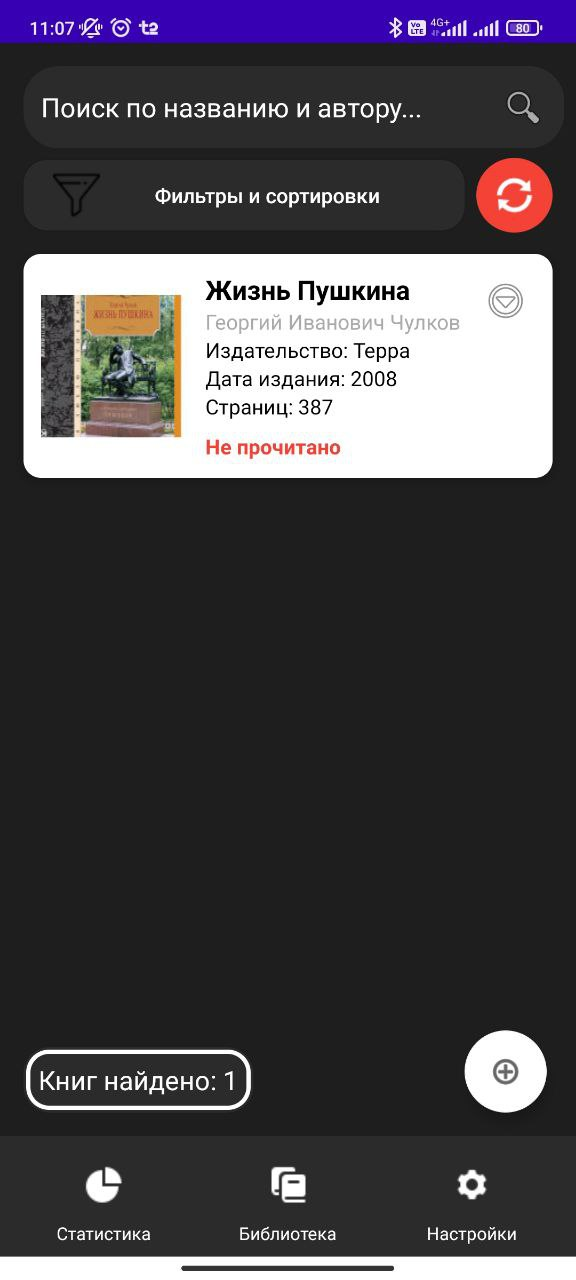


Рисунок 4 – Главный экран с добавленной книгой

Меню книги: для открытия меню книги, нажмите на кнопку в верхнем правом углу карточки книги. В меню книги можно удалить книгу, добавить заметку, сделать ее избранной и прочитать на Google Books.

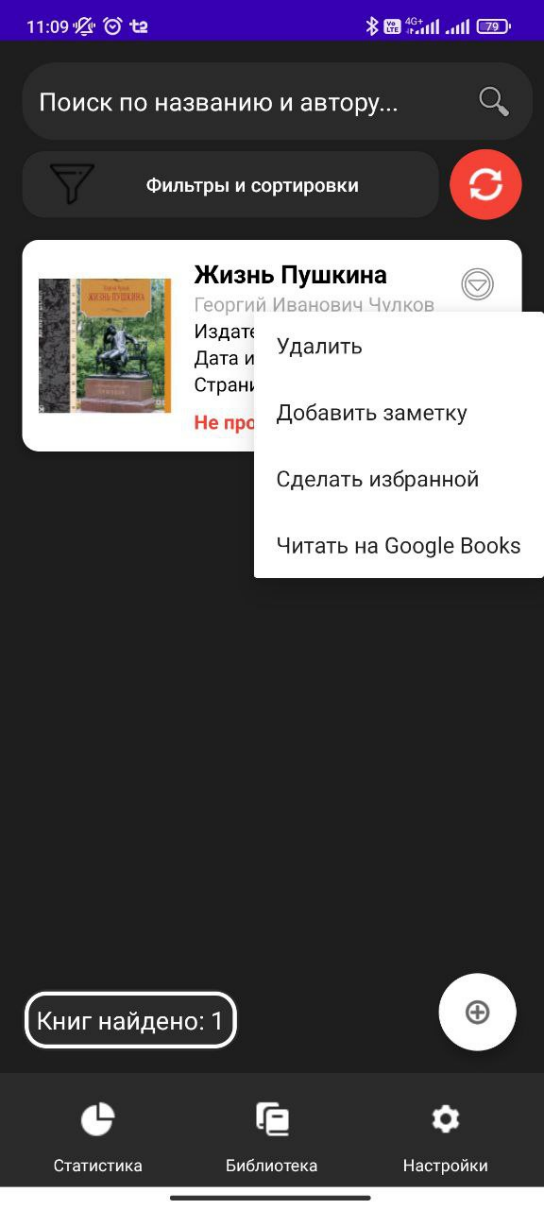


Рисунок 5 – Главный экран с открытым меню книги

Экран статистики: на экране статистики можно просмотреть статистику по добавленным книгам.



Рисунок 6 – Экран статистики

Экран настройки: на экране настройки можно поменять тему приложения и ознакомиться с «Помощь по приложению».

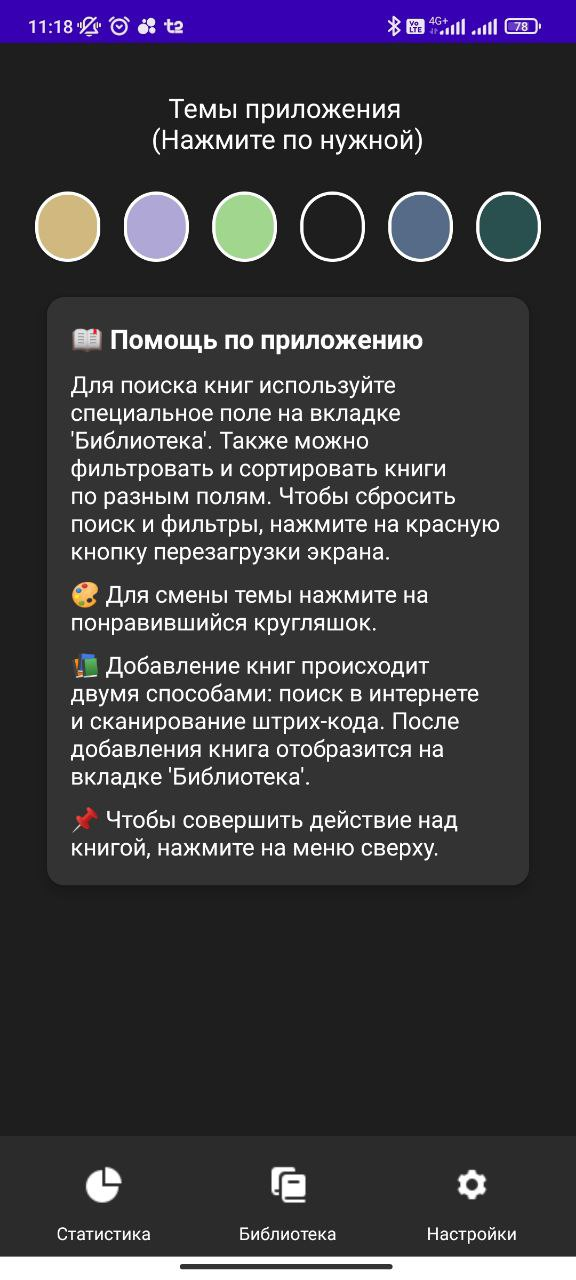


Рисунок 7 – Экран настройки