МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 “Программное обеспечение информационной

безопасности мобильных систем”

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«SharedBooks»

Выполнил студент Комарова Елизавета Ивановна

(Ф.И.О.)

Руководитель проекта ассистент Северенчик Н.А. (учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: ассистент Северенчик Н.А. (учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: ассистент Северенчик Н.А. (учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc59034916)

[1 Обзор аналогов приложений «Библиотека программиста» 4](#_Toc59034917)

[1.1 Приложение «Моя библиотека» 4](#_Toc59034918)

[1.2 Приложение «Handy Library» 6](#_Toc59034919)

[1.3 Выводы по главе «Обзор аналогов приложений «Библиотека программиста» 8](#_Toc59034920)

[2 Проектирование программного продукта 9](#_Toc59034921)

[2.1 Проектирование мобильного приложения 9](#_Toc59034922)

[2.2 Проектирование базы данных 12](#_Toc59034923)

[2.3 Выводы по главе «Проектирование программного продукта» 13](#_Toc59034924)

[3 Программная реализация приложений 14](#_Toc59034925)

[3.1 Технические средства для разработки 14](#_Toc59034926)

[3.2 Разработка баз данных 14](#_Toc59034927)

[3.3 Разработка мобильного приложения 14](#_Toc59034928)

[3.4 Вывод по главе «Программная реализация приложений» 22](#_Toc59034929)

[4 Руководство пользователя 23](#_Toc59034930)

[5.1 Выводы по главе «Руководство пользователя» 28](#_Toc59034931)

[ЗАКЛЮЧЕНИЕ 29](#_Toc59034932)

[Список использованных источников 30](#_Toc59034933)

[**ПРИЛОЖЕНИЕ А** 31](#_Toc59034934)

# ВВЕДЕНИЕ

Цель курсового проекта: предоставление возможности учета выданных в долг книг для любителей чтения.

Задачи курсового проекта:

1. Изучить аналогичные программные средства по теме курсового проекта.

Выполнение данной задачи поможет нам сформировать требования к разрабатываемому продукту, определить, какие функциональные возможности предоставит пользователям мобильное приложение, а также выявить недостатки конкурентов, которые мы можем исправить.

1. Спроектировать приложение.

Проектирование приложения позволит получить более качественный, продуманный и проработанный в комплексе продукт и сэкономить время, которое уходило бы на внесение правок в уже готовом приложении.

1. Реализовать приложение.

В курсовом проекте будет реализовано мобильное приложение для с возможностью выдачи книг и учетом их возврата. В век развития информационных технологий этот вариант будет более оптимальный и удобный, так как вся информация будет хранится в одном приложении на телефоне, и пользователь легко сможет им воспользоваться, где бы он не был.

1. Разработать руководство пользователя.

Цель создания документа заключается в том, чтобы предоставить пользователю возможность самостоятельно решать свои прикладные задачи с помощью системы. Этой цели может служить и введение в предметную область, и ознакомление со всеми возможностями программы, и описание конкретных процедур решения задач, и приведение различных инструкций. Иногда Руководство пользователя больше похоже на справочник, к которому можно обращаться в процессе работы, а иногда – на учебник, который позволяет изучить принципы работы с программой и ее возможности, а затем применять их на практике

# 1 Обзор аналогов приложений «Библиотека программиста»

# Приложение «Моя библиотека»

"Моя библиотека" – бесплатное и свободное от рекламы приложение для учета и создания каталога домашней библиотеки (рисунок 1.1.1).

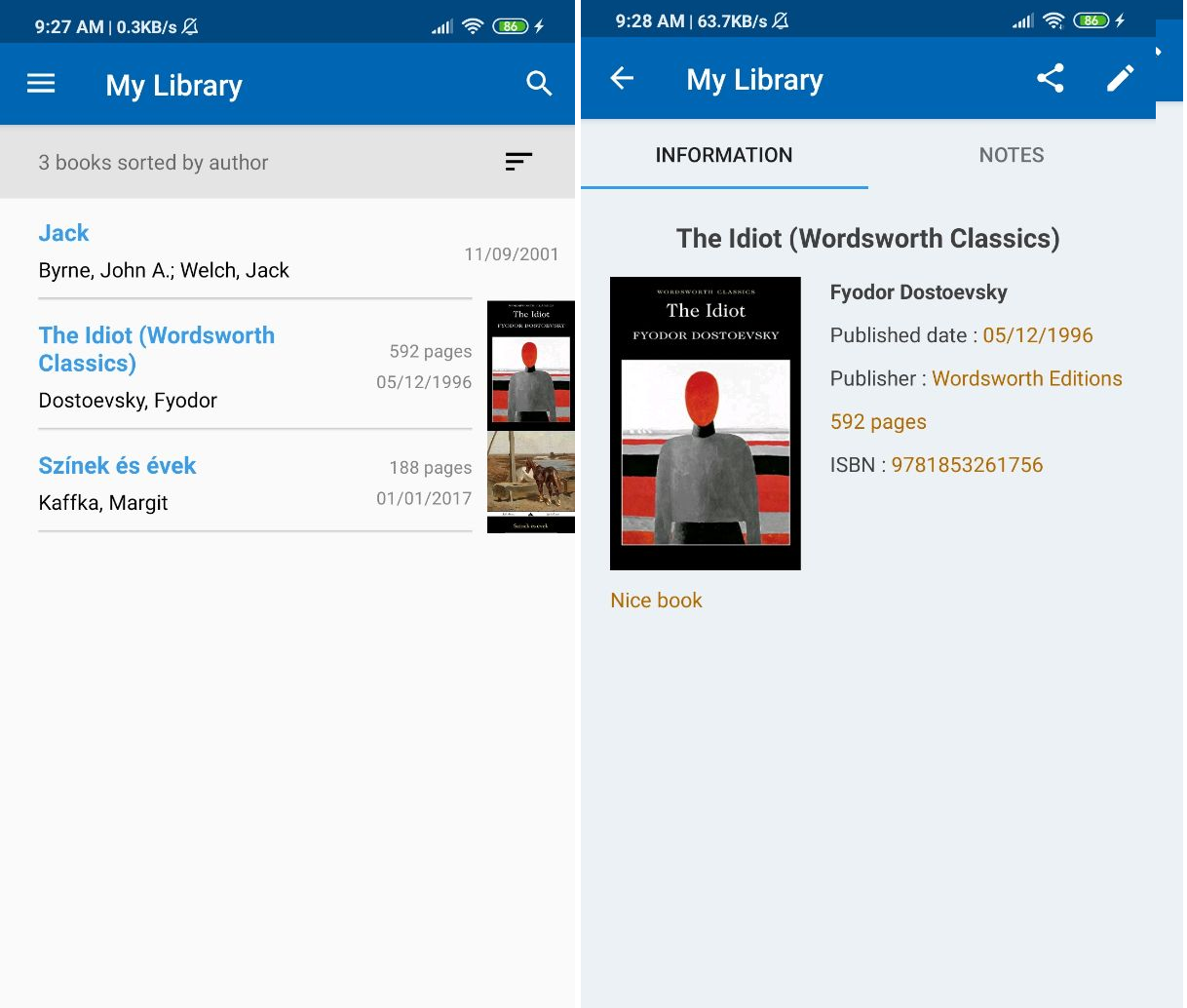


Рисунок 1.1.1 – Приложение «Моя библиотека»

Возможности приложения:

* Добавление книг в библиотеку, сканируя штрих-коды (можно указать название, автора, краткое содержание, дату публикации, издательство, добавить обложку и т. д.)
* Добавление книг в библиотеку по номеру ISBN или ключевым словам
* Добавление книг вручную
* Поиск книг в библиотеке
* Сортировка книг по названию, именам
* Экспорт всей библиотеки в один файл Excel
* Управление списком пожеланий

Для поиска книг по номеру ISBN "Моя библиотека" использует сервисы Google, Amazon, Open Library, Worldcat, IsbnPlus, Moly, data.bn, Libris, ISFDB и Biblio. Если приложение не находит номер ISBN, это значит, что его нет в этих сервисах.

Данные характеристики безусловно являются плюсами, однако здесь отсутствует возможность выдачи книг в долг, что не позволит пользователям контролировать, кто отдал им назад отданные в долг книги, а кто нет.

# 1.2 Приложение «Handy Library»

Это приложение поможет вам организовать вашу коллекцию книг (или небольшую библиотеку) удобным способом (рисунок 1.1.2).

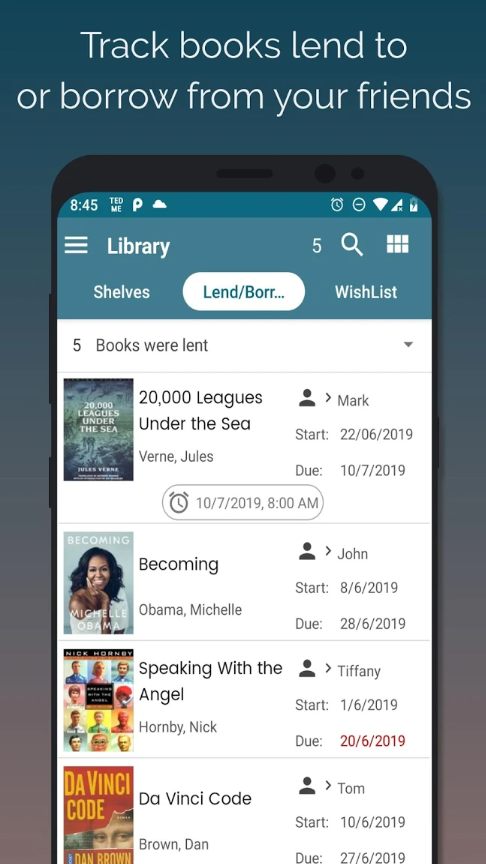


Рисунок 1.1.2 – Приложение «Handy Library»

Основные характеристики:

* Сканирование ISBN-кода (штрих-кода), чтобы получить такие сведения о книге, как название, автор, номер страницы, издатель, дата публикации, рейтинг, описание и сохранить книгу в библиотеке;
* Поиск книги в Интернете по названию, автору или ISBN-коду, чтобы получить детали книги и сохранить ее;
* Добавить новую книгу в свою библиотеку вручную, когда службы приложения не могут соответствовать вашему поиску;
* Возможность следить за книгами, которые вы одалживаете или одалживаете у своих друзей, и уведомляете, когда книга просрочена;
* Управление списком пожеланий;
* Написать заметку для книги;
* Поделитесь ссылкой на книгу со своими друзьями;
* Найти книгу, которая уже есть в библиотеке, отсканировав ее код ISBN или по ключевому слову (название, автор, ISBN);
* Сортировка книг в библиотеке по названию, автору или добавленной дате;
* Фильтровать книги, которые отмечены как избранные, прочитанные или непрочитанные;
* Организовать книги по полкам или категориям,
* Экспорт или импорт библиотеки с файлами Excel;

# 1.3 Выводы по главе «Обзор аналогов приложений «Библиотека программиста»

В данной главе были изучены два приложения, которые являются аналогами приложения, – это «Моя библиотека», «Handy Library». В конечном результате были подведены выводы:

Дизайн у приложений имеет большое количество кнопок и текстового контента, который затрудняет восприятие информации. Из этого можно сделать вывод, что дизайн для будущего приложения необходимо делать простой и лаконичный.

Плюсы функционала приложения «Моя библиотека»: сортировка книг по названию, именам, категориям.

Минусы функционала приложения «Моя библиотека»: нет возможности отслеживания выданных на руки книг и возможности самой выдачи.

Плюсы функционала приложения «Handy Library»: отслеживание книг, которые вы даете в долг или берете в долг у своих друзей.

Минусы функционала приложения «Handy Library»: большой размер приложения.

# 2 Проектирование программного продукта

# 2.1 Проектирование мобильного приложения

В рамках курсового проекта необходимо разработать мобильное приложение, предоставляющее пользователю возможность создания, просмотра, удаления и редактирования книг; возможность создания, просмотра, удаления и редактирования читателей; возможность выдачи книг читателю; учет возврата книг в библиотеку; поиск книг по наименованию, году публикации, жанру и автору.

Первым этапом в использовании приложения будет регистрация, для этой цели будет создана активность с полями для ввода электронной почты и пароля, а также двумя кнопками: sign up (зарегистрироваться) и sign in (войти).

При заполнении необходимой информации и нажатии кнопки sign up, пользователь должен быть зарегистрирован в системе либо необходимо информировать его о том, что такой пользователь уже существует, в случае ввода электронной почты, ранее зарегистрированной, и о том, что электронная почта и пароль не валидные, в случае ввода данных не соответствующих требованиям.

При заполнении необходимой информации и нажатии кнопки нажатии sign in, пользователь должен попасть на главную активность либо необходимо информировать его о том, что введенные логин или пароль неправильные.

В целях безопасности каждый вход в приложения защищен биометрической аутентификацией и должна быть предусмотрена невозможность создания скриншота.

На главной странице приложение должно содержать нижнее навигационное меню, с вкладками: книги, читатели, выдачи. Каждой вкладке должен соответствовать фрагмент.

Во фрагменте с читателями должен отображаться список читателей, необходима кнопка добавления. Аналогично для книг. Активность выдач будет только список.

Можно ознакомится с UML-диаграммой последовательности для фрагмента «Читатели» на рисунке 2.1.1.

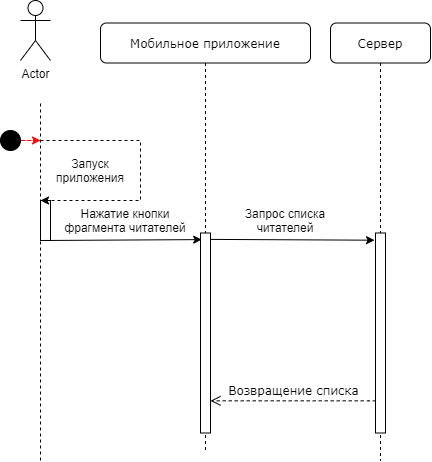


Рисунок 2.1.1 – Диаграмма последовательности

Должно быть реализовано контекстное меню для списка книг и читателей.

Контекстно меню для книг в соответствии с их статусом должно включать:

1. AVAILABLE
   * View (просмотр книги)
   * Delete (удаление книги)
   * Edit (редактирование книги)
   * Issue book (выдача книги)
   * Dispose book (списание книги)
2. LOANED
   * View (просмотр книги)
   * Return book (возврат книги)
3. DISPOSED
   * View (просмотр книги)
   * Delete (удаление книги)
   * Edit (редактирование книги)

Контекстно меню для читателей должно включать в себя возможность редактирования и удаления читателя, а также возможность отправки электронного письма определенным читателям.

Также для каждого фрагментов читатели и книги должно присутствовать меню в верхней панели, включающее следующие возможности:

1. Для книг:
   1. Поиск;
   2. Выход;
   3. Сортировка по статусу книги.
2. Для читателей:
   1. Выход;
   2. Сортировка по факту выдачи.

В случае выбора пользователем в контекстном меню пункта просмотр должен появится новый фрагмент с информацией о книге, читателе.

В случае выбора редактирования должен появится похожий фрагмент с возможность сохранения фрагмента.

При удалении, элемент из списка должен удалиться, а список обновиться.

При выдаче книги должен открываться новый фрагмент с выбранной книгой и возможностью выбора читателя, не имеющего на руках книгу, кнопка выдать и отмена.

Можно ознакомиться с UML-диаграммой, а именно диаграммой вариантов использования (рис. 2.1.2).

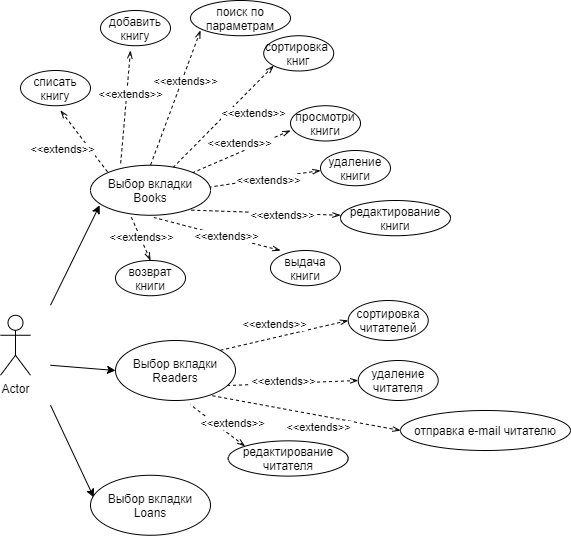


Рисунок 2.1.2 – Диаграмма вариантов использования

Данная диаграмма отображает функциональность программного продукта с точки зрения получения значимого результата для пользователя.

Таким образом приложение должно выполнять следующие функции:

* предоставление списка книг, читателей, выдач;
* возможность создания и редактирования книг и читателей;
* сортировка книг по статусу;
* сортировка читателей по факту наличия книг на руках (те, у кого есть долг, и те, у кого его нет);
* поиск книг;
* возможность выдачи книг читателям;
* возможность отправки e-mail определенным пользователяи;
* учет возврата книг для добавленных читателей;

# 2.2 Проектирование базы данных

В базе данных данного приложения должны быть реализованы 3 таблицы.

В таблице «Book» содержится информация о продуктах. Столбцы данной таблицы:

* book\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* userid – строковый тип данных, максимальной длинной, не должен содержать значений null;
* title – строковый тип данных, максимальной длинной, не должен содержать значений null;
* publication\_year – целочисленный тип данных, не должен содержать значений null;
* authors – строковый тип данных, c максимальной длинной;
* book\_status – символьный тип данных, максимальной длинной 50 символов, принимающий значения «AVAILABLE», «LOANED», «DISPOSED»;
* cover – строковый тип данных, c максимальной длинной;

В таблице «Reader» хранится информация о читателях:

* reader\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* userid – строковый тип данных, максимальной длинной, не должен содержать значений null;
* first\_name – целочисленный тип данных, не должен содержать значений null;
* last\_name – целочисленный тип данных, не должен содержать значений null;
* email – символьный тип данных, максимальной длинной 1 символ.
* has\_book – целочисленный тип данных, который может принимать значение 1, 0 или NULL.

В таблице «Loan» содержится информация о выдачах книг. Столбцы данной таблицы:

* loan\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* userid – строковый тип данных, максимальной длинной, не должен содержать значений null;
* reader\_id – внешний ключ, представленный целочисленным значением;
* book\_id – внешний ключ, представленный целочисленным значением;
* email – целочисленный тип данных, внешний ключ;
* id\_closed – целочисленный тип данных, не должен содержать значений null.

Схема базы данных представлена на рисунке 2.2.1.

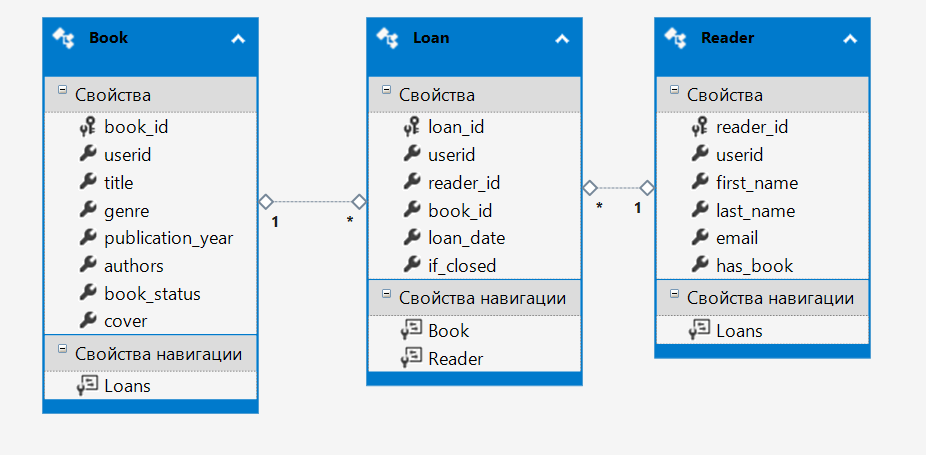


Рисунок 2.2.1 – Схема базы данных

Из данной схемы видно, что таблицы Book и Loan соединены связью многий-ко-многим, аналогично Reader и Loan.

# 2.3 Выводы по главе «Проектирование программного продукта»

В данной главе было рассмотрено проектирование всех компонентов приложения.

По части мобильного приложения был определен основной функционал приложения, также были разработаны диаграмма последовательности и диаграмма вариантов использования приложения.

Была спроектирована база данных, где были определены необходимые таблицы, поля для каждой из них. Также были установлены связи между таблицами, которые описаны на схеме базы данных.

**3 Программная реализация приложений**

**3.1 Технические средства для разработки**

Для разработки приложений курсового проекта были использованы следующие технические средства:

Android Studio – интегрированная среда разработки для работы с платформой Android [1].

Java – строго типизированный объектно-ориентированный язык программирования [2].

SQLite – компактная встраиваемая СУБД. [3].

SQLCipher – это библиотека с открытым исходным кодом, созданная компанией Zetetic, для прозрачного 256-битного AES шифрования баз данных SQLite [4].

Firebase Auth – предоставляет бэкенд, простой в использовании SDK и готовые библиотеки пользовательского интерфейса для реализации аутентификации пользователей в приложении [5].

**3.2 Разработка баз данных**

Код создания структуры базы данных представлен в Приложении A.

Серверное приложение имеет следующие структурные объекты:

* Book – таблица продуктов;
* Reader– таблица категорий;
* Loan – таблица рецептов.

Доступ к данным базы данных из веб-приложения используя библиотеку SQLCipher.

**3.3 Разработка мобильного приложения**

Мобильное приложение было реализовано в среде разработки – Android Studio. Для написания приложения был использован язык программирования Java.

Графическая структура проекта представлена на рисунке 3.3.1.

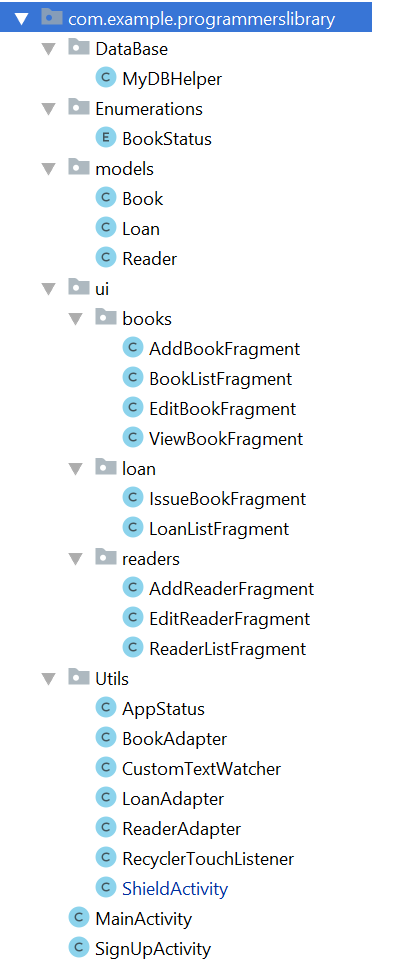


Рисунок 3.3.1 – Графическая структура проекта

Пакет DataBase хранит в себе классы для работы с базой данных (см. таблицу 3.3.1).

**Таблица 3.3.1 – Содержание пакета DataBase**

|  |  |
| --- | --- |
| Класс MyDBHelper – | |
| insertBook | Метод для вставки книги |
| insertReader | Метод для вставки читателя |
| insertLoan | Метод для вставки выдачи |
| getBook | Метод для получения книги из базы данных |
| getReader | Метод для получения читателя из базы данных |
| getLoan | Метод для получения выдачи |
| getAllBooks | Метод для получения списка всех книг |
| getAllReaders | Метод для получения списка всех читателей |
| getAllLoans | Метод для получения всех выдачи |
| getBooksCount | Метод для получения количества книг у пользователя |
| getReadersCount | Метод для получения количества читателей у пользователя |
| getLoansCount | Метод для получения количества выдач у пользователя |
| updateBook | Метод для обновления информации о книге |
| updateReader | Метод для обновления информации о читателе |
| updateLoan | Метод для обновления информации о выдаче |
| deleteBook | Метод для удаления книги |
| deleteReader | Метод для удаления читателя |
| deleteLoan | Метод для удаления выдачи |
| getLoansByBookID | Метод для получения выдачи по идентификатору книги |
| getReaderNameByReaderID | Метод для получения имени читателя по его идентификатору |
| getBookTitleByBookID | Метод для получения названия книги ее его идентификатору |
| Validate | Метод для валидации JSON документа |

Пакет Enumeration хранит в себе перечисления (см. таблицу 3.3.2).

**Таблица 3.3.2 – Содержание пакета Enumeration**

|  |  |
| --- | --- |
| Перечисление BookStatus – доступные списки книг | |
| AVAILABLE | Книга доступна |
| LOANED | Книга выдана |
| DISPOSED | Книга списана |

Пакет models хранит в себе классы для описания сущностей (см. таблицу 3.3.3).

**Таблица 3.3.3 – Содержание пакета models**

|  |
| --- |
| Класс Book – класс для описания объекта книга |
| Класс Loan – класс для описания выдачи |
| Класс Reader – класс для описания объекта читатель |

Пакет ui хранит в себе классы работы с фрагментами и интерфейсом в целом (см. таблицу 3.3.4).

**Таблица 3.3.4 – Содержание пакета ui**

|  |  |
| --- | --- |
| Класс AddBookFragment – класс для работы с фрагментом для добавления книги | |
| onCreateView | Метод для инициализации и заполнения элементов управления фрагмента |
| onItemSelected | Метод, описывающий обработчик нажатия |
| onNothingSelected | Метод, обрабатывающий случай, когда ничего не выбрано |
| onActivityResult | Метод для обработки результата выбора обложки |
| createBook | Метод для вставки новой книги в базу данных |
| Класс BookListFragment – работа с каталогом книг | |
| onCreateView | Метод для инициализации и заполнения элементов управления фрагмента |
| deleteBook | Метод для удаления книги из базы данных |
| showActionsDialog | Метод для открытия меню с выбором |
| returnBook | Метод для возврата книги в базу данных |
| editBook | Метод для редактирования книги |
| onOptionsItemSelected | Метод для обработки выбранного пункта меню |
| Класс EditBookFragment – метод для редактирования книги, во многом аналогичен классу AddBookFragment | |
| Класс ViewBookFragment – | |
| onCreateView | Метод для инициализации элементов упраления фрагмента |
| onCreate |  |
| Класс IssueBookFragment – класс для реализации выдачи книги | |
| onCreateView | Метод для инициализации элементов управления |
| SpinnerBookClass | Класс для создания выпадающего меню |
| SpinnerReaderClass | Класс для создания выпадающего меню |
| Класс LoanListFragment – класс для управления фрагментом со списком займов | |
| onCreateView | Метод для инициализации элементов упраления фрагмента |
| toggleEmptyNotes | Метод для установления сообщения в случае отсутствия |
| Класс AddReaderFragment – класс для управления фрагментом для добавления читателя | |
| onCreateView | Метод для инициализации элементов управления фрагмента |
| createReader | Метод для добавления читателя в базу данных |
| Класс EditReaderFragment – класс для управления фрагментом для редактирования читателя | |
| onCreateView | Метод для инициализации элементов управления фрагмента |
| updateReader | Метод для вставки читателя в базу данных |
| Класс ReaderListFragment – класс для управления фрагментом со списком всех книг | |
| onCreateView | Метод для инициализации элементов управления фрагмента |
| deleteReader | Метод для удаления читателя из базы данных |
| showActionsDialog | Метод для открытия меню выбора |
| onOptionsItemSelected | Метод для обработки выбранного пункта меню |

Пакет Utils хранит в себе вспомогательные классы (см. таблицу 3.3.5).

**Таблица 3.3.5 – Содержание пакета Utils**

|  |  |
| --- | --- |
| Класс AppStatus – класс для проверки наличия интернета | |
| isOnline | Метод, проверяющий наличие интернета |
| Класс BookAdapter – класс, реализующий адаптер для наполнения данными списка книг | |
| MyViewHolder | Класс, реализующий ViewHolder |
| onCreateViewHolder | Метод для создания объекта ViewHolder |
| onBindViewHolder | Метод отвечает за связь Java объекта и View |
| getItemCount | Метод возвращает количество элементов в списке |
| getFilter | Метод для реализации поиска |
| Класс CustomTextWatcher – класс для проверки ввода символов в поле | |
| beforeTextChanged | Метод работающий перед тем, как текст изменится |
| onTextChanged | Метод работающий во время того, как текст изменяется |
| afterTextChanged | Метод работающий после того, как текст изменится |
| Класс LoanAdapter – класс аналогичный BookAdapter | |
| Класс ReaderAdapter – класс аналогичный BookAdapter | |
| Класс RecycleViewTouchListener – класс, реализующий интерфейс для обработки нажатия на экран | |
| ClickListener | Объект интерфейса ClickListener |
| GestureDetector | Объект для отслеживания действий |
| RecyclerTouchListener | Конструктор класса |
| onInterceptTouchEvent | Метод, передающий управление для обработки нажатия расширяемому классу |
| onTouchEvent | Метод, обрабатывающий событие нажатия |
| onRequestDisallowInterceptTouchEvent | Метод прерывающий управление расширяемого класса при нажатии |
| ClickListener | Интерфейс с методами onClick(), onLongClick() |
| ShieldActivity | Активность для работы с FingerPrint |

Ознакомится с диаграммой классов разработанного приложения можно на рисунке 3.3.2.

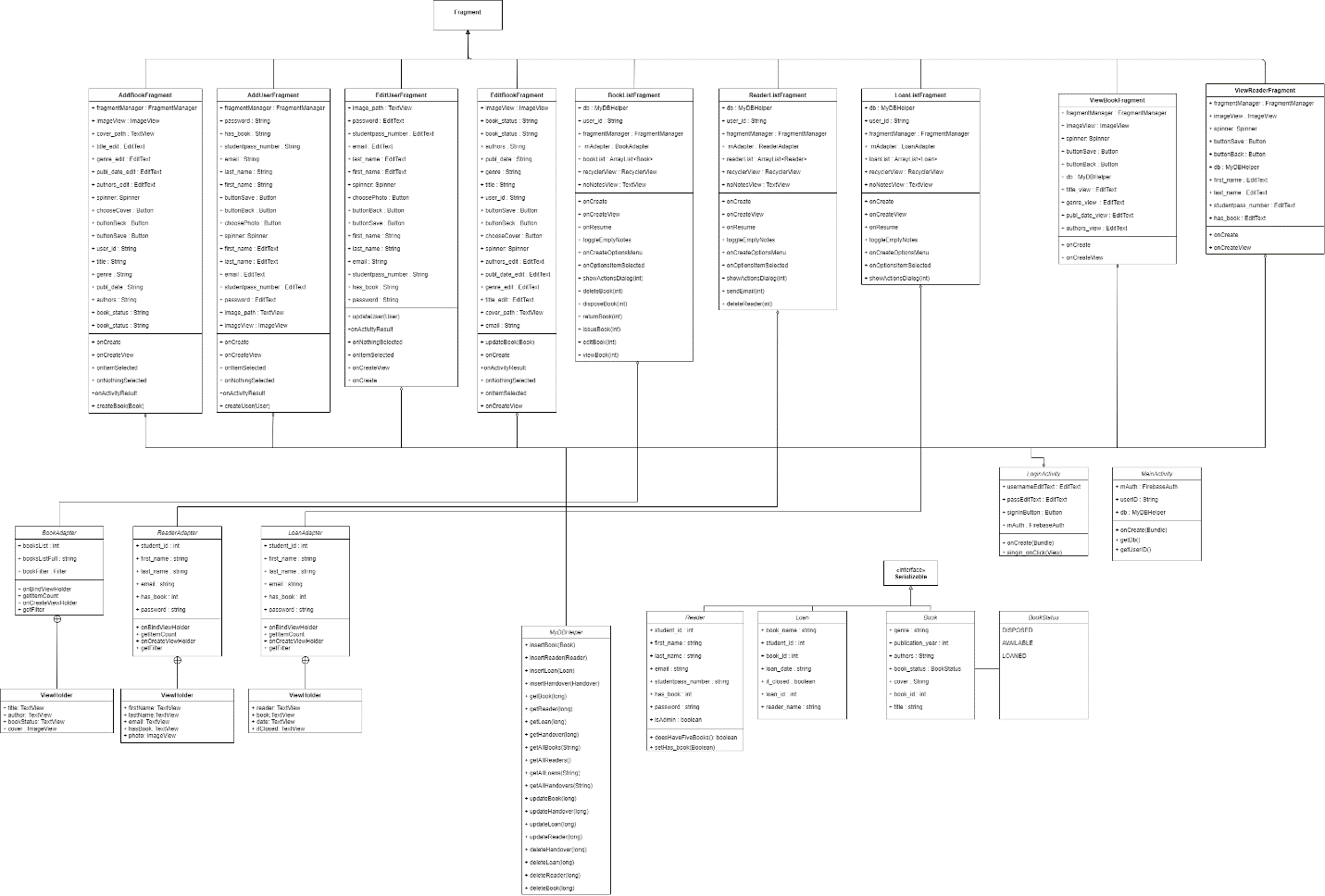


Рисунок 3.3.2 – Диаграмма классов

Классы AddBookFragment, AddUserFragment, EditUserFragment, EditBookFragment, BookListFragment, ReaderListFragment, LoanListFragment, ViewBookFragment, ViewReaderFragment имеют отношение c классом Fragment – обобщение.

Обобщение – выражает специализацию или наследование, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя.

Классы AddBookFragment, AddUserFragment, EditUserFragment, EditBookFragment, BookListFragment, ReaderListFragment, LoanListFragment, ViewBookFragment, ViewReaderFragment, LoginActivity имееют отношение агрегации с классом MyDBHelper. Такое же отношение у классов BookListFragment, ReaderListFragment, LoanListFragment, с BookAdapter.ViewHolder, LoanAdapter.ViewHolder.

Агрегация — это разновидность ассоциации при отношении между целым и его частями. Агрегация встречается, когда один класс является коллекцией или контейнером других.

Классы Reader, Loan, Book имплементируют интерфейс Serializable, то есть находятся в отношении реализация.

Реализация — отношение между двумя элементами модели, в котором один элемент (клиент) реализует поведение, заданное другим (поставщиком).

Класс Book связан с BookStatus связью ассоциации.

Класс ViewHolder является вложенным в BookAdapter.ViewHolder, LoanAdapter.ViewHolder, ReaderAdapter.ViewHolder. Классы ViewHolder являются вложенными и имеют отношение ассоциации вложенного класса.

После реализации мобильного приложения была выполнена обфускация кода Android с помощью ProGuard [6].

# 3.4 Вывод по главе «Программная реализация приложений»

В данной главе было рассмотрена разработка каждого отдельного компонента приложения.

На первоначальном этапе были рассмотрены технологии, которые применены для разработки курсового проекта.

Вторым этапом была разработка структуры базы данных на стороне сервера.

Третий этапом стала разработка мобильного приложения, где была представлена графическая структура приложения, рассмотрен каждый класс, их методы и поля, в частности.

# 4 Руководство пользователя

Для того чтобы установить приложения на телефон необходимо иметь версию Android минимум 8.0 и версию API не ниже 26. Версию Android и API вашего телефона можно узнать в настройках телефона.

Если версия Android подходит, тогда можно устанавливать приложение на мобильный телефон. В меню ваших приложений или на главном экране смартфона появится новая иконка с именем «Programmers Library». Запустим приложение нажатием на иконку приложения. Откроется страница входа или регистрации. Для того, чтобы зайти в приложении необходимо зарегистрироваться или войти в уже созданный профиль. Чтобы зарегистрироваться необходимо иметь интернет-соединение, чтобы отправить данные на сервер.

После того, как вы войдете или зарегистрируетесь как обычный пользователь, вам откроется страница с биометрической аутентификацией, после успешной аутентификации откроется страница со списком книг (рис. 5.1).

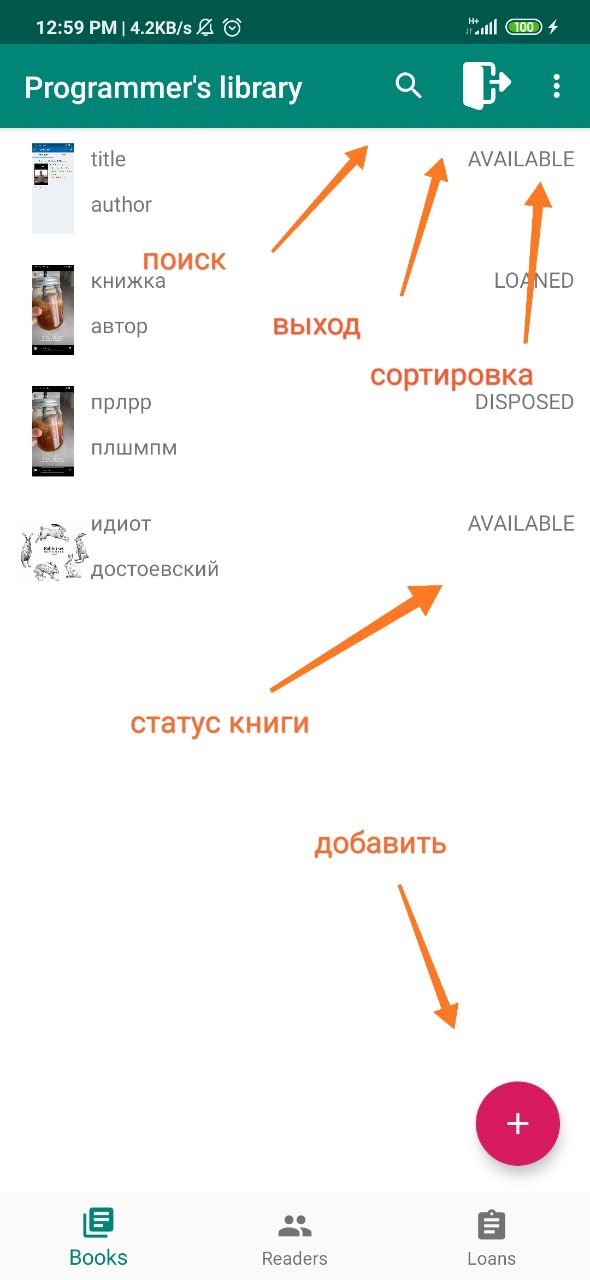
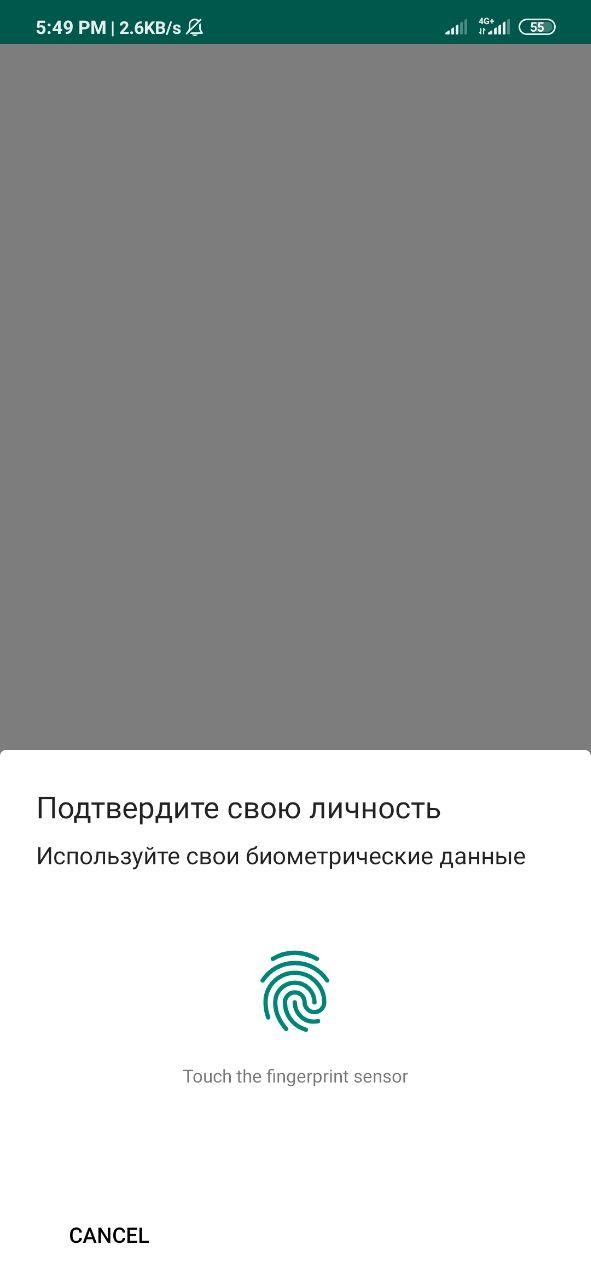


Рисунок 5.1 – Страница списка книг

При долгом нажатии на элемент списка откроется контекстное меню, которое можно посмотреть на рисунке 5.2.

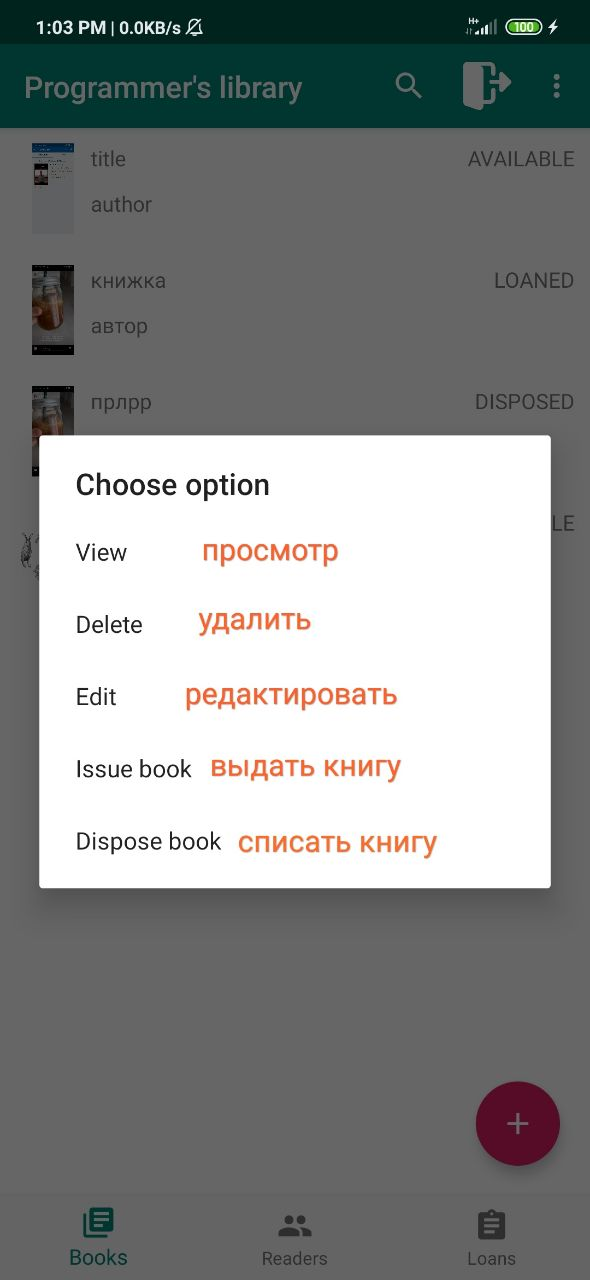


Рисунок 5.2 – Контекстное меню книг

При нажатии на кнопку добавить откроется страница добавления книги (см. рис. 5.3).

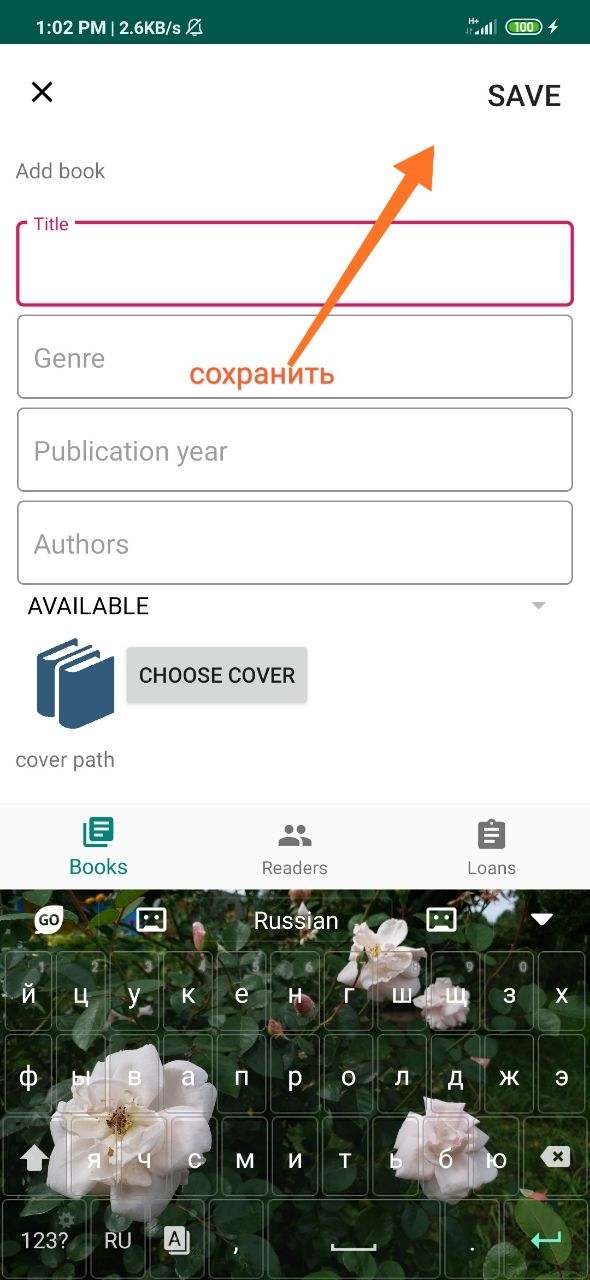


Рисунок 5.3 – Страница добавления

После нажатия на Readers откроется фрагмент с со списком читателей (рис. 5.4). На экране показаны кружки, которые являются индикатором, есть ли у читателя долг.

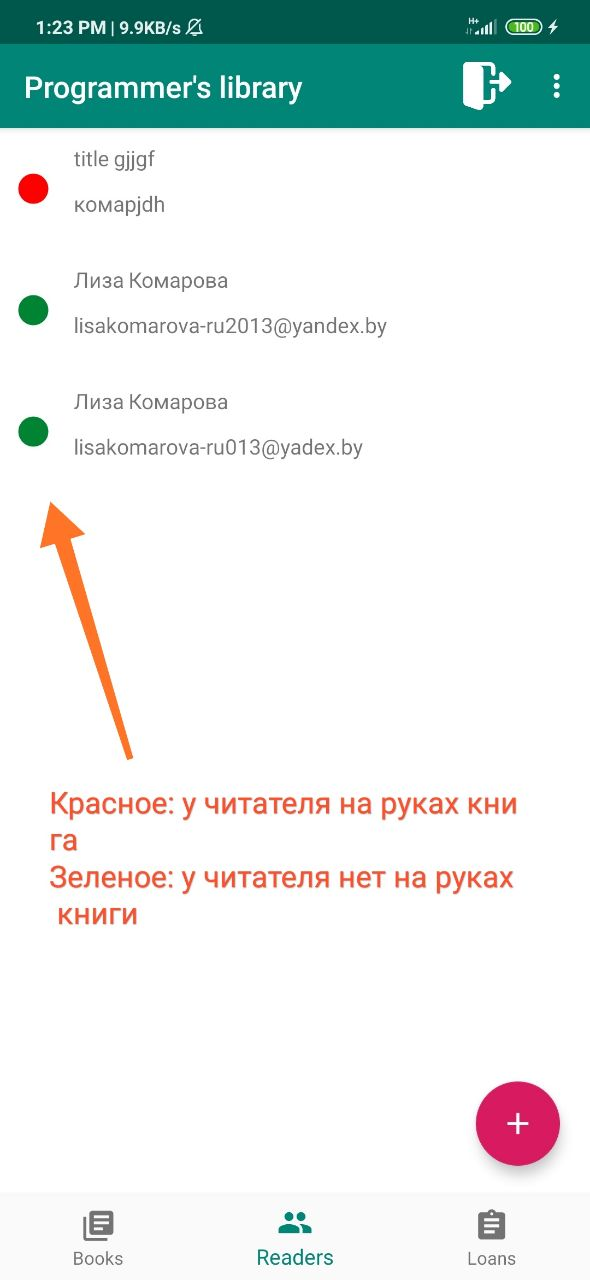


Рисунок 5.4 – Список читателей

Добавление читателя представлено на рисунке 5.5). Поле адреса электронной почты почты должно быть уникальным.

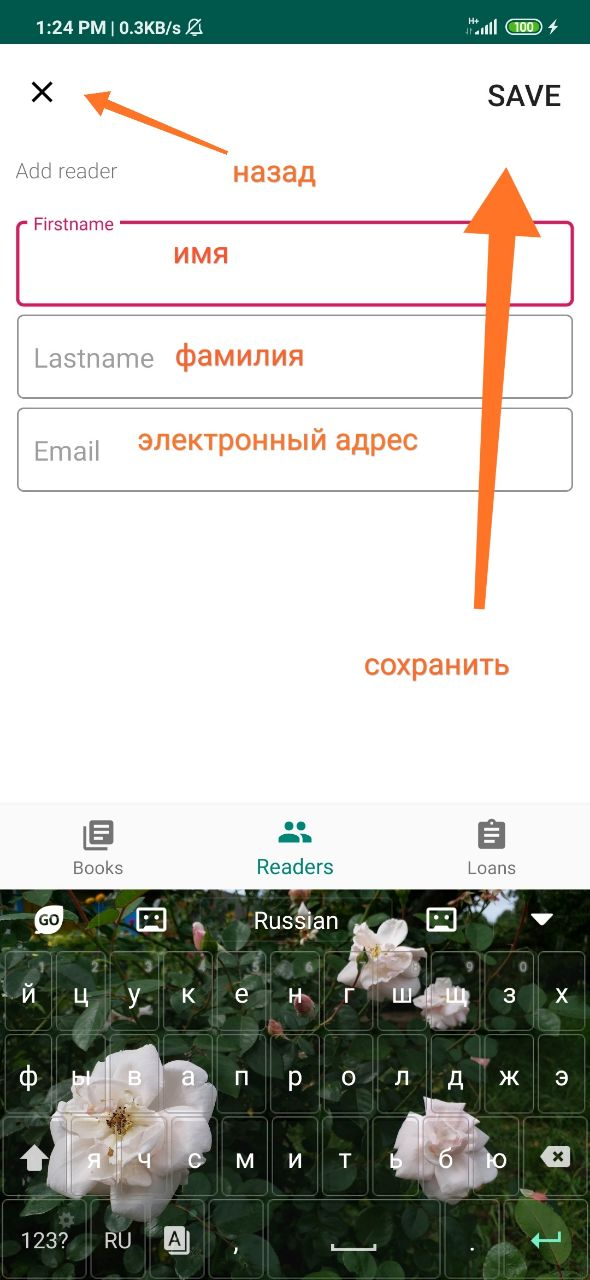


Рисунок 5.5 – Добавление читателя

При нажатии на вкладку Loans, появится список выдач книг (рисунок 5.6).

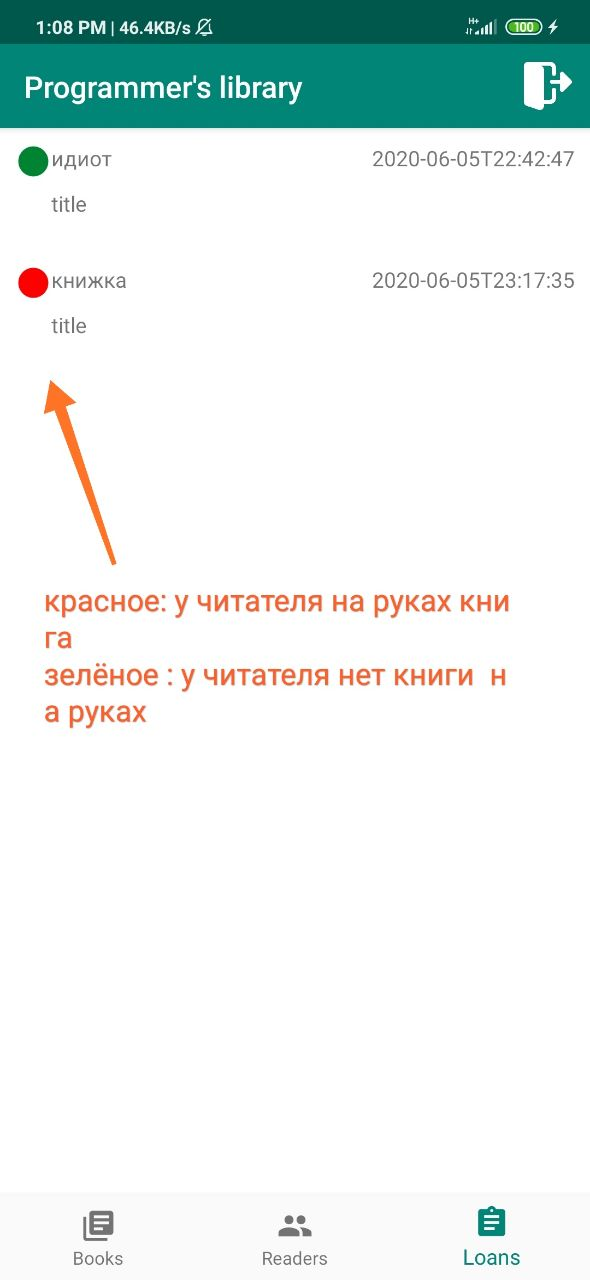


Рисунок 5.6 – Список выдач книг

# 5.1 Выводы по главе «Руководство пользователя»

В данной главе было описано руководство пользователя.

Данное руководство позволяет пользователю узнать требования для установки приложения на телефон, а также как использовать все функции приложения. Были предоставлены скриншоты самого приложения для большей наглядности использования.

# ЗАКЛЮЧЕНИЕ

Целью моего курсового проекта было предоставление возможности учета выданных в долг книг для любителей чтения.

Были решены следующие поставленные задачи:

* обзор аналогов;

В результате было рассмотрено 2 приложения и были выявлены их положительные и отрицательные стороны. Эта информация была использована для улучшения функционала приложения и пользовательского интерфейса.

* проектирование приложения;

Был спроектирован каждый отдельный компонент приложения. В мобильном приложении был определен основной функционал, также были разработаны необходимые диаграммы. При проектировании базы данных, были определены необходимые таблицы и разработана схема базы данных.

* реализация приложения;

Данная задача была реализована с помощью следующих технических средств: SQLite, среда разработки Android Studio, языков программирования Java, библиотек SQLCipher, Firebase Auth.

Была реализована локальная база данных.

При решении данной задачи была достигнута поставленная цель, и был создан проект «Библиотека программиста», который включает в себя android-клиент и сервер.

* разработка руководства пользователя.

Разработано руководство, позволяющее пользователю узнать требования для установки приложения на телефон, а также как использовать все функции мобильного приложения.

Мобильное приложение имеет простой интерфейс и обладает следующими особенностями:

* предоставление списка книг, читателей, выдач;
* возможность создания и редактирования книг и читателей;
* сортировка книг по статусу;
* сортировка читателей по факту наличия книг на руках (те, у кого есть долг и те у кого его нет);
* поиск книг;
* возможность выдачи книг читателям;
* учет возврата книг для добавленных читателей;

В соответствии с поставленными задачами и полученным результатом можно сделать вывод, что задачи были выполнены в полном объеме.

# Список использованных источников

1. Android Studio [Электронный ресурс]. / Android Developers – Режим доступа:<https://developer.android.com/studio> – Дата доступа: 29.04.2020
2. \The Java Language Specification [Электронный ресурс]. / James Gosling; Bill Joy; Guy Steele; Gilad Bracha; Alex Buckley; Daniel Smith – Режим доступа: https://docs.oracle.com/javase/specs/jls/se12/html/index.html – Дата доступа: 29.04.2020
3. SQLite [Электронный ресурс] / [sqlite.org](https://www.sqlite.org/) – Режим доступа: <https://www.sqlite.org/2020/sqlite-android-3340000.aar> Дата доступа: 10.12.2020
4. SQLCipher [Электронный ресурс] / newtonsoft.com – Режим доступа: https://github.com/sqlcipher/android-database-sqlcipher.git/ Дата доступа: 10.12.2020
5. Firebase Authentification [Электронный ресурс] / google developers – Режим доступа: https://firebase.google.com/docs/auth / Дата доступа: 09.12.2020
6. How To Obfuscate In Android With ProGuard [Электронный ресурс]. / https://medium.com/ – Режим доступа: https://medium.com/@angelhiadefiesta/how-to-obfuscate-in-android-with-proguard-acab47701577 – Дата доступа: 14.12.2020

**ПРИЛОЖЕНИЕ А**

Код создания структуры серверной базы данных

**package** com.example.programmerslibrary.DataBase;  
  
**import** android.content.ContentValues;  
**import** android.content.Context;  
**import** android.database.Cursor;  
  
**import** com.example.programmerslibrary.Enumerations.BookStatus;  
**import** com.example.programmerslibrary.models.Book;  
**import** com.example.programmerslibrary.models.Loan;  
**import** com.example.programmerslibrary.models.Reader;  
  
**import** net.sqlcipher.database.SQLiteDatabase;  
**import** net.sqlcipher.database.SQLiteOpenHelper;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** MyDBHelper **extends** SQLiteOpenHelper {  
  
 **private static final** String ***password*** = **"password12345678"**;  
 *// Database Version* **private static final int *DATABASE\_VERSION*** = 1;  
  
 *// Database Name* **private static final** String ***DATABASE\_NAME*** = **"library\_db"**;  
  
 **public** MyDBHelper(Context context) {  
  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
 SQLiteDatabase.*loadLibs*(context);  
 }  
  
 @Override  
 **public void** onConfigure(SQLiteDatabase db) {  
 *//db.setForeignKeyConstraintsEnabled(true);* **super**.onConfigure(db);  
 db.execSQL(**"PRAGMA foreign\_keys = ON;"**);  
 }  
  
 *// Creating Tables* @Override  
 **public void** onCreate(SQLiteDatabase db) {  
  
 *// create book table* db.execSQL(Book.***CREATE\_TABLE***);  
 *// create reader table* db.execSQL(Reader.***CREATE\_TABLE***);  
 *// create loans table* db.execSQL(Loan.***CREATE\_TABLE***);  
 }  
  
 *// Upgrading database* @Override  
 **public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  
 *// Drop older table if existed* db.execSQL(**"DROP TABLE IF EXISTS "** + Book.***TABLE\_NAME***);  
 db.execSQL(**"DROP TABLE IF EXISTS "** + Reader.***TABLE\_NAME***);  
 db.execSQL(**"DROP TABLE IF EXISTS "** + Loan.***TABLE\_NAME***);  
  
 *// Create tables again* onCreate(db);  
 }  
  
 **public long** insertBook(Book book) {  
 *// get writable database as we want to write data* SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 *// `id` and `book status` will be inserted automatically.  
 // no need to add them* values.put(Book.***COLUMN\_USER\_ID***, book.getUserid());  
 values.put(Book.***COLUMN\_TITLE***, book.getTitle());  
 values.put(Book.***COLUMN\_GENRE***, book.getGenre());  
 values.put(Book.***COLUMN\_PUBLICATION\_YEAR***, book.getPublication\_year());  
 values.put(Book.***COLUMN\_AUTHORS***, book.getAuthors());  
 values.put(Book.***COLUMN\_BOOK\_STATUS***, book.getBook\_status().toString());  
 values.put(Book.***COLUMN\_COVER***, book.getCover());  
  
 *// insert row* **long** id = db.insert(Book.***TABLE\_NAME***, **null**, values);  
  
 *// close db connection* db.close();  
  
 *// return newly inserted row id* **return** id;  
 }  
  
 **public long** insertReader(Reader reader) {  
 *// get writable database as we want to write data* SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 *// `id` and `book status` will be inserted automatically.  
 // no need to add them* values.put(Reader.***COLUMN\_FIRSTNAME***, reader.getFirst\_name());  
 values.put(Reader.***COLUMN\_USER\_ID***, reader.getUserid());  
 values.put(Reader.***COLUMN\_LASTNAME***, reader.getLast\_name());  
 values.put(Reader.***COLUMN\_EMAIL***, reader.getEmail());  
 **if** (reader.doesHaveBook())  
 values.put(Reader.***COLUMN\_HASBOOK***, **"true"**);  
 **else** values.put(Reader.***COLUMN\_HASBOOK***, **"false"**);  
  
 *// insert row* **long** id = db.insert(Reader.***TABLE\_NAME***, **null**, values);  
  
 *// close db connection* db.close();  
  
 *// return newly inserted row id* **return** id;  
 }  
  
 **public long** insertLoan(Loan loan) {  
 *// get writable database as we want to write data* SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 *// `id` and `book status` will be inserted automatically.  
 // no need to add them* values.put(Loan.***COLUMN\_USER\_ID***, loan.getUserID());  
 values.put(Loan.***COLUMN\_READER\_ID***, loan.getReader\_id());  
 values.put(Loan.***COLUMN\_BOOK\_ID***, loan.getBook\_id());  
 **if** (loan.getIf\_closed())  
 values.put(Loan.***COLUMN\_IF\_CLOSED***, **"true"**);  
 **else** values.put(Loan.***COLUMN\_IF\_CLOSED***, **"false"**);  
  
 *// insert row* **long** id = db.insert(Loan.***TABLE\_NAME***, **null**, values);  
  
 *// close db connection* db.close();  
  
 *// return newly inserted row id* **return** id;  
 }  
  
 **public** Book getBook(**long** id) {  
 *// get readable database* SQLiteDatabase db = **this**.getReadableDatabase();  
  
 Cursor cursor = db.query(Book.***TABLE\_NAME***,  
 **new** String[]{Book.***COLUMN\_ID***, Book.***COLUMN\_USER\_ID***, Book.***COLUMN\_TITLE***, Book.***COLUMN\_GENRE***, Book.***COLUMN\_PUBLICATION\_YEAR***,  
 Book.***COLUMN\_AUTHORS***, Book.***COLUMN\_BOOK\_STATUS***,  
 Book.***COLUMN\_COVER***},  
 Book.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(id)}, **null**, **null**, **null**, **null**);  
  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
  
 *// prepare note object* Book note = **new** Book(  
 cursor.getInt(cursor.getColumnIndex(Book.***COLUMN\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_USER\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_TITLE***)),  
 cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_GENRE***)),  
 cursor.getInt(cursor.getColumnIndex(Book.***COLUMN\_PUBLICATION\_YEAR***)),  
 cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_AUTHORS***)),  
 BookStatus.*valueOf*(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_STATUS***))),  
 cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_COVER***)));  
  
 *// close the db connection* cursor.close();  
  
 **return** note;  
 }  
  
 **public** Reader getReader(**long** id) {  
 *// get readable database* SQLiteDatabase db = **this**.getReadableDatabase();  
  
 Cursor cursor = db.query(Reader.***TABLE\_NAME***,  
 **new** String[]{Reader.***COLUMN\_ID***, Reader.***COLUMN\_USER\_ID***, Reader.***COLUMN\_FIRSTNAME***, Reader.***COLUMN\_LASTNAME***, Reader.***COLUMN\_EMAIL***,  
 Reader.***COLUMN\_HASBOOK***},  
 Reader.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(id)}, **null**, **null**, **null**, **null**);  
  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
  
 *// prepare reader object* Reader reader = **new** Reader(  
 cursor.getInt(cursor.getColumnIndex(Reader.***COLUMN\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_USER\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_FIRSTNAME***)),  
 cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_LASTNAME***)),  
 cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_EMAIL***)),  
 Boolean.*parseBoolean*(cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_HASBOOK***))));  
  
 *// close the db connection* cursor.close();  
  
 **return** reader;  
 }  
 **public** Loan getLoan(**long** id) {  
 *// get readable database* SQLiteDatabase db = **this**.getReadableDatabase();  
  
 Cursor cursor = db.query(Loan.***TABLE\_NAME***,  
 **new** String[]{Loan.***COLUMN\_ID***, Loan.***COLUMN\_USER\_ID***, Loan.***COLUMN\_READER\_ID***, Loan.***COLUMN\_BOOK\_ID***, Loan.***COLUMN\_LOAN\_DATE***,  
 Loan.***COLUMN\_IF\_CLOSED***},  
 Loan.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(id)}, **null**, **null**, **null**, **null**);  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
 Cursor cursorReader = db.query(Reader.***TABLE\_NAME***,  
 **new** String[]{Reader.***COLUMN\_FIRSTNAME***, Reader.***COLUMN\_LASTNAME***},  
 Reader.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)))}, **null**, **null**,  
 **null**, **null**);  
 Cursor cursorBook = db.query(Book.***TABLE\_NAME***,  
 **new** String[]{Book.***COLUMN\_TITLE***},  
 Book.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_BOOK\_ID***)))}, **null**, **null**,  
 **null**, **null**);  
 **if** (cursorBook != **null**)  
 cursorBook.moveToFirst();  
 **if** (cursorReader != **null**)  
 cursorReader.moveToFirst();  
  
 String bookTitle = cursorBook.getString(0);  
 String readerFullname = cursorReader.getString(0) + **" "** + cursorReader.getString(1);  
  
 *// prepare loan object* Loan loan = **new** Loan(  
 cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)),  
 bookTitle,  
 readerFullname,  
 cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_LOAN\_DATE***)),  
 Boolean.*parseBoolean*(cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_IF\_CLOSED***))));  
 loan.setReader\_id(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)));  
 loan.setBook\_id(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_BOOK\_ID***)));  
 *// close the db connection* cursor.close();  
  
 **return** loan;  
 }  
  
 **public** List<Book> getAllBooks(String user\_id) {  
 List<Book> books = **new** ArrayList<>();  
  
 *// Select All Query* String selectQuery = **"SELECT \* FROM "** + Book.***TABLE\_NAME*** +  
 **" WHERE "** + Book.***COLUMN\_USER\_ID*** + **"= "** + user\_id;  
  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.query(Book.***TABLE\_NAME***,  
 **new** String[]{Book.***COLUMN\_ID***, Book.***COLUMN\_USER\_ID***, Book.***COLUMN\_TITLE***, Book.***COLUMN\_GENRE***, Book.***COLUMN\_PUBLICATION\_YEAR***,  
 Book.***COLUMN\_AUTHORS***, Book.***COLUMN\_BOOK\_STATUS***, Book.***COLUMN\_COVER***},  
 **null**, **null**, **null**, **null**,  
 **null**, **null**);  
  
 *// looping through all rows and adding to list* **if** (cursor.moveToFirst()) {  
 **do** {  
 Book book = **new** Book();  
 book.setBook\_id(cursor.getInt(cursor.getColumnIndex(Book.***COLUMN\_ID***)));  
 book.setUserid(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_USER\_ID***)));  
 book.setTitle(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_TITLE***)));  
 book.setGenre(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_GENRE***)));  
 book.setPublication\_year(cursor.getInt(cursor.getColumnIndex(Book.***COLUMN\_PUBLICATION\_YEAR***)));  
 book.setAuthors(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_AUTHORS***)));  
 book.setBook\_status(BookStatus.*valueOf*(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_STATUS***))));  
 book.setCover(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_COVER***)));  
  
 books.add(book);  
 } **while** (cursor.moveToNext());  
 }  
  
 *//* cursor.close();  
 *// close db connection* db.close();  
  
 *// return notes list* **return** books;  
 }  
  
 **public** List<Reader> getAllReaders(String user\_id) {  
 List<Reader> readers = **new** ArrayList<>();  
  
 *// Select All Query* String selectQuery = **"SELECT \* FROM "** + Reader.***TABLE\_NAME*** +  
 **" WHERE "** + Reader.***COLUMN\_USER\_ID*** + **"= "** + user\_id;  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.query(Reader.***TABLE\_NAME***,  
 **new** String[]{Reader.***COLUMN\_ID***, Reader.***COLUMN\_USER\_ID***, Reader.***COLUMN\_FIRSTNAME***, Reader.***COLUMN\_LASTNAME***, Reader.***COLUMN\_EMAIL***,  
 Reader.***COLUMN\_HASBOOK***},  
 **null**, **null**, **null**, **null**,  
 **null**, **null**);  
  
 *// looping through all rows and adding to list* **if** (cursor.moveToFirst()) {  
 **do** {  
 Reader reader = **new** Reader();  
 reader.setReader\_id(cursor.getInt(cursor.getColumnIndex(Reader.***COLUMN\_ID***)));  
 reader.setUserid(cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_USER\_ID***)));  
 reader.setFirst\_name(cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_FIRSTNAME***)));  
 reader.setLast\_name(cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_LASTNAME***)));  
 reader.setEmail(cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_EMAIL***)));  
 reader.setHas\_book(Boolean.*parseBoolean*(cursor.getString(cursor.getColumnIndex(Reader.***COLUMN\_HASBOOK***))));  
  
 readers.add(reader);  
 } **while** (cursor.moveToNext());  
 }  
  
 *//* cursor.close();  
 *// close db connection* db.close();  
  
 *// return notes list* **return** readers;  
 }  
  
 **public** List<Loan> getAllLoans(String user\_id) {  
 List<Loan> loans = **new** ArrayList<>();  
  
 *// Select All Query* String selectQuery = **"SELECT \* FROM "** + Loan.***TABLE\_NAME*** +  
 **" WHERE "** + Loan.***COLUMN\_USER\_ID*** + **"= "** + user\_id;  
  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.query(Loan.***TABLE\_NAME***,  
 **new** String[]{Loan.***COLUMN\_ID***, Loan.***COLUMN\_USER\_ID***, Loan.***COLUMN\_READER\_ID***, Loan.***COLUMN\_BOOK\_ID***, Loan.***COLUMN\_LOAN\_DATE***,  
 Loan.***COLUMN\_IF\_CLOSED***},  
 **null**, **null**, **null**, **null**,  
 **null**, **null**);  
  
  
 *// looping through all rows and adding to list* **if** (cursor.moveToFirst()) {  
 **do** {  
 *// prepare reader object* Cursor cursorReader = db.query(Reader.***TABLE\_NAME***,  
 **new** String[]{Reader.***COLUMN\_FIRSTNAME***, Reader.***COLUMN\_LASTNAME***},  
 Reader.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)))}, **null**, **null**,  
 **null**, **null**);  
 Cursor cursorBook = db.query(Book.***TABLE\_NAME***,  
 **new** String[]{Book.***COLUMN\_TITLE***},  
 Book.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_BOOK\_ID***)))}, **null**, **null**,  
 **null**, **null**);  
 **if** (cursorBook != **null**)  
 cursorBook.moveToFirst();  
 **if** (cursorReader != **null**)  
 cursorReader.moveToFirst();  
  
 String bookTitle = cursorBook.getString(0);  
 String readerFullname = cursorReader.getString(0) + **" "** + cursorReader.getString(1);  
  
 *// prepare loan object* Loan loan = **new** Loan(  
 cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)),  
 bookTitle,  
 readerFullname,  
 cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_LOAN\_DATE***)),  
 Boolean.*parseBoolean*(cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_IF\_CLOSED***))));  
 loan.setReader\_id(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)));  
 loan.setBook\_id(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_BOOK\_ID***)));  
 loans.add(loan);  
 } **while** (cursor.moveToNext());  
 }  
  
 *//* cursor.close();  
 *// close db connection* db.close();  
  
 *// return notes list* **return** loans;  
 }  
  
  
 **public int** getBooksCount(String user\_id) {  
 **return** getAllBooks(user\_id).size();  
 }  
  
 **public int** getReadersCount(String user\_id) {  
 **return** getAllReaders(user\_id).size();  
 }  
  
 **public int** getLoansCount(String user\_id) {  
 **return** getAllLoans(user\_id).size();  
 }  
  
 **public void** updateBook(Book book) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(Book.***COLUMN\_TITLE***, book.getTitle());  
 values.put(Book.***COLUMN\_GENRE***, book.getGenre());  
 values.put(Book.***COLUMN\_PUBLICATION\_YEAR***, book.getPublication\_year());  
 values.put(Book.***COLUMN\_AUTHORS***, book.getAuthors());  
 values.put(Book.***COLUMN\_BOOK\_STATUS***, book.getBook\_status().toString());  
 values.put(Book.***COLUMN\_COVER***, book.getCover());  
  
 *// updating row* db.update(Book.***TABLE\_NAME***, values, Book.***COLUMN\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(book.getBook\_id())});  
 }  
  
 **public void** updateReader(Reader reader) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(Reader.***COLUMN\_FIRSTNAME***, reader.getFirst\_name());  
 values.put(Reader.***COLUMN\_LASTNAME***, reader.getLast\_name());  
 values.put(Reader.***COLUMN\_EMAIL***, reader.getEmail());  
 **if** (reader.doesHaveBook())  
 values.put(Reader.***COLUMN\_HASBOOK***, **"true"**);  
 **else** values.put(Reader.***COLUMN\_HASBOOK***, **"false"**);  
  
  
 *// updating row* db.update(Reader.***TABLE\_NAME***, values, Reader.***COLUMN\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(reader.getReader\_id())});  
 }  
  
 **public void** updateLoan(Loan loan) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 **if** (loan.getIf\_closed())  
 values.put(Loan.***COLUMN\_IF\_CLOSED***, **"true"**);  
 **else** values.put(Loan.***COLUMN\_IF\_CLOSED***, **"false"**);  
  
  
 *// updating row* db.update(Loan.***TABLE\_NAME***, values, Loan.***COLUMN\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(loan.getLoan\_id())});  
 }  
  
 **public void** deleteBook(Book book) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 db.delete(Book.***TABLE\_NAME***, Book.***COLUMN\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(book.getBook\_id())});  
 db.close();  
 }  
  
 **public void** deleteReader(Reader reader) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 db.delete(Reader.***TABLE\_NAME***, Reader.***COLUMN\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(reader.getReader\_id())});  
 db.close();  
 }  
  
 **public void** deleteLoan(Loan loan) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 db.delete(Loan.***TABLE\_NAME***, Loan.***COLUMN\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(loan.getLoan\_id())});  
 db.close();  
 }  
  
 **public** Loan getLoansByBookID(**int** id) {  
 *// get readable database* SQLiteDatabase db = **this**.getReadableDatabase();  
  
 Cursor cursor = db.query(Loan.***TABLE\_NAME***,  
 **new** String[]{Loan.***COLUMN\_ID***, Loan.***COLUMN\_USER\_ID***, Loan.***COLUMN\_READER\_ID***, Loan.***COLUMN\_BOOK\_ID***, Loan.***COLUMN\_LOAN\_DATE***,  
 Loan.***COLUMN\_IF\_CLOSED***},  
 Loan.***COLUMN\_BOOK\_ID*** + **"=? and "** + Loan.***COLUMN\_IF\_CLOSED*** + **"=?"**,  
 **new** String[]{String.*valueOf*(id), **"false"**}, **null**, **null**, **null**, **null**);  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
 Cursor cursorReader = db.query(Reader.***TABLE\_NAME***,  
 **new** String[]{Reader.***COLUMN\_FIRSTNAME***, Reader.***COLUMN\_LASTNAME***},  
 Reader.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)))}, **null**, **null**,  
 **null**, **null**);  
 Cursor cursorBook = db.query(Book.***TABLE\_NAME***,  
 **new** String[]{Book.***COLUMN\_TITLE***},  
 Book.***COLUMN\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_BOOK\_ID***)))}, **null**, **null**,  
 **null**, **null**);  
 **if** (cursorBook != **null**)  
 cursorBook.moveToFirst();  
 **if** (cursorReader != **null**)  
 cursorReader.moveToFirst();  
  
 String bookTitle = cursorBook.getString(0);  
 String readerFullname = cursorReader.getString(0) + **" "** + cursorReader.getString(1);  
  
 *// prepare loan object* Loan loan = **new** Loan(  
 cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_ID***)),  
 cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_USER\_ID***)),  
 bookTitle,  
 readerFullname,  
 cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_LOAN\_DATE***)),  
 Boolean.*parseBoolean*(cursor.getString(cursor.getColumnIndex(Loan.***COLUMN\_IF\_CLOSED***))));  
 loan.setReader\_id(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_READER\_ID***)));  
 loan.setBook\_id(cursor.getInt(cursor.getColumnIndex(Loan.***COLUMN\_BOOK\_ID***)));  
  
 *// close the db connection* cursor.close();  
  
 **return** loan;  
 }  
  
 **public** String getReaderNameByReaderID(**long** id) {  
 Reader r = getReader(id);  
  
 **return** r.getFirst\_name() + r.getLast\_name();  
 }  
  
 **public** String getBookTitleByBookID(**long** id) {  
 **return** getBook(id).getTitle();  
 }  
  
 **public** SQLiteDatabase getReadableDatabase() {  
 **return** (**super**.getReadableDatabase(***password***));  
 }  
  
 **public** SQLiteDatabase getWritableDatabase() {  
 **return** (**super**.getWritableDatabase(***password***));  
 }  
}