Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 – «Программное обеспечение информационной безопасности мобильных систем»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

КУРСОВОГО ПРОЕКТА

по дисциплине Программное обеспечение безопасных мобильных систем

Тема: «MangaBook»

Исполнитель Николайчук Александра Николаевна

(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

Минск 2020

**Содержание**

[1. Обзор аналогов приложения 4](#_Toc59400056)

[1.1. WEBTOON 4](#_Toc59400057)

[1.2. Manga Viewer Rebirth 5](#_Toc59400058)

[1.3 Выводы по главе «Обзор аналогов приложения» 6](#_Toc59400059)

[2. Проектирование программного продукта 7](#_Toc59400060)

[2.1 Проектирование мобильного приложения 7](#_Toc59400061)

[2.2 Проектирование базы данных 8](#_Toc59400062)

[2.3 Выводы по главе «Проектирование программного продукта» 9](#_Toc59400063)

[3. Программная реализация приложений 10](#_Toc59400064)

[3.1 Технические средства для разработки 10](#_Toc59400065)

[3.2 Разработка баз данных 10](#_Toc59400066)

[3.3 Разработка мобильного приложения 10](#_Toc59400067)

[3.4 Вывод по главе «Программная реализация приложений» 12](#_Toc59400068)

[4. Обеспечение безопасности приложения 13](#_Toc59400069)

[4.1 Защита базы данных 13](#_Toc59400070)

[4.2 Запрет на запись и снимки экрана 13](#_Toc59400071)

[4.3 Биометрическая аутентификация 13](#_Toc59400072)

[4.4 Обфускация 13](#_Toc59400073)

[4.5 Выводы по главе «Обеспечение безопасности приложения» 13](#_Toc59400074)

[5. Руководство пользователя 14](#_Toc59400075)

[5.1 Выводы по главе «Руководство пользователя» 18](#_Toc59400076)

[Заключение 19](#_Toc59400077)

[Список использованных источников 20](#_Toc59400078)

[ПРИЛОЖЕНИЕ А 21](#_Toc59400079)

[ПРИЛОЖЕНИЕ Б 30](#_Toc59400080)

**Введение**

Цель курсового проекта: предоставление возможности просмотра книг.

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 с возможностью просмотра книг. 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 приложения
   1. WEBTOON

Приложение предназначено для чтения комиксов онлайн. Оно является бесплатным для использования. Содержит большую базу онлайн комиксов по различным жанрам. Данное приложение предоставляет удобный интерфейс и приятный дизайн. Экран приложения представлен на скриншоте ниже (Рисунок 1.1.1).

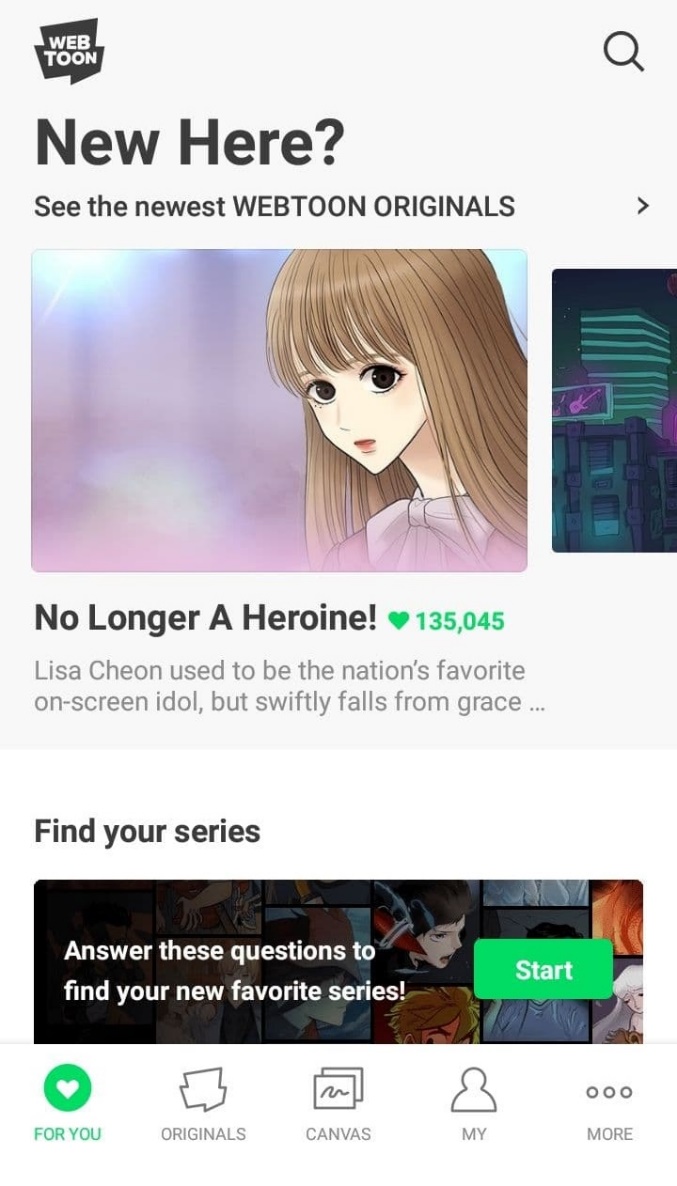


Рисунок 1.1.1 – Главный экран приложения «**WEBTOON**»

Данное приложение является уникальным среди аналогов, т.к. комиксы которые в нем размещаются читаются только прокручиванием экрана вниз из-за особенности создания глав: глава состоит из одной очень длинной страницы. Материалы данного приложения написаны преимущественно на английском и корейском языках, однако некоторые комиксы переводятся на русский язык другими пользователями и с данными переводами могут ознакомится все остальные пользователи. Данное приложение не имеет русской локализации, что является недостатком для пользователей, которые не владеют английским языком. Так же контект скачиваемый внутри данного приложения не доступен со сторонних приложений и все скачанные изображения являются зашифрованными.

* 1. Manga Viewer Rebirth

Приложение предназначено для чтения комиксов. Оно является бесплатным для использования, но в любой момент может показаться реклама, что может раздражать пользователя. Содержит большую базу книг по различным жанрам. Экран приложения представлен на скриншоте ниже (Рисунок 1.2.1)

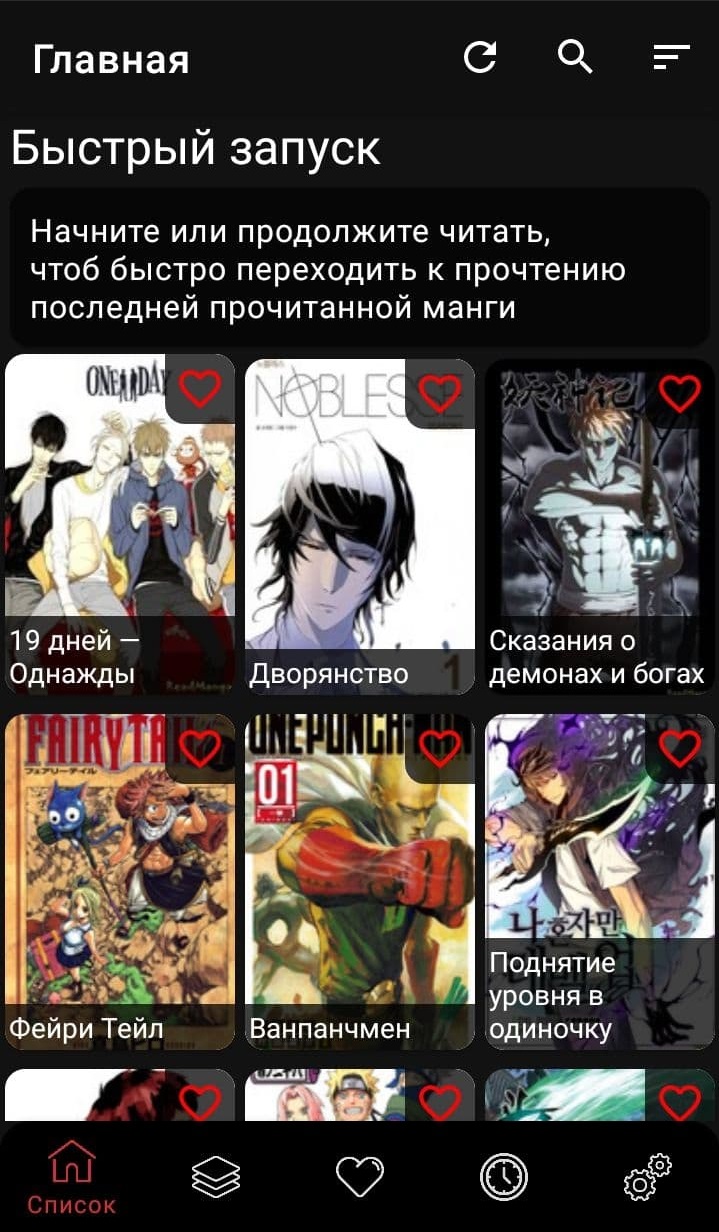


Рисунок 1.2.1 – Главный экран приложения «**Manga Viewer Rebirth**»

Данное приложение предоставляет своеобразный яркий дизайн, в котором используются цвета совершенно не подходящие друг другу. Интерфейс приложения не является интуитивно понятным и может запутать пользователя. Специальные значки, для перехода в разделы являются слишком большими, что занимает много места на экране. При выборе определенной книги через день в ней могут пропасть все главы, а также информация об ошибке накладывается на страницу, таким образом, что саму информацию сложно прочитать. При загрузке глав книг, они сохраняются во внешнюю память, поэтому они могут просматриваться другими приложениями.

**1.3 Выводы по главе «Обзор аналогов приложения»**

В данной главе были рассмотрены два приложения, являющихся аналогами для темы курсового проекта: «WEBTOON» и «Manga Viewer Rebirth». В результате были сделаны следующие выводы.

Приложение «WEBTOON» имеет более приятый глазу и удобный дизайн, чем «Manga Viewer Rebirth», а также интерфейс у него интуитивно более понятный. Дизайн у приложения «WEBTOON» почти везде имеет оттенки зеленого, в то время как дизайн приложения «Manga Viewer Rebirth» для различных деталей имеет совершенно разнообразные цвета, что мешает общему восприятию.

Сохранение глав в приложении «WEBTOON» реализовано в папку приложения с шифрованием данных, в то время как второго приложения они сохраняются в папку во внешней памяти, что является небезопасным.

Также явным минусом приложения «Manga Viewer Rebirth» является навязчивая реклама, которая демонстрируется слишком часто. Нестабильная работа приложения делает его на время бесполезным, т.к. становится невозможным читать книги.

**2.** **Проектирование программного продукта**

Задачами приложения являются:

* предоставление пользователю возможности ознакомится с интересующими его книгами
* предоставление пользователю возможности добавлять книги
* предоставление пользователям возможности общаться друг с другом

Основной целевой аудиторией приложения являются мужчины и женщины в возрасте от 14 лет до 30 лет, знающие русский язык, которые интересуются комиксами.

В данном приложении пользователь может иметь одну из следующих ролей – «пользователь» и «администратор».

Пользователь может добавлять книги, добавлять книги, добавлять переводы, комментировать страницы, главы, книги.

**2.1 Проектирование мобильного приложения**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций.

Для неавторизованных пользователей: зарегистрироваться.

Для авторизованных пользователей:

* добавлять книги, новые главы
* добавлять переводчика, автора
* поиска книг
* просмотр каталога книг

Мобильное приложение должно быть реализовано в среде разработки – Android Studio 3.0. Для написания приложения использован язык программирования Java.

Для пользователя предусмотрены следующие возможности интерфейса:

На начальном экране приложение должно представлять каталог книг.

Если пользователь захочет использовать функции, предоставляемые только зарегистрированными пользователям, предоставить возможность перейти на экран регистрации.

На экране регистрации приложение должно позволить пользователю предоставить форму для заполнения полей: пароль, почту.

После регистрации приложение отображает раздел с профилем пользователя, чтобы он мог заполнить оставшиеся данные, и затем нажав кнопку подтверждения должен быть перенаправлен на тот раздел, откуда был произведен переход на экран регистрации.

В разделе с профилем пользователя приложение должно предоставить ему возможность изменения всех параметров пользователя.

В разделе с каталогом должны быть отображены последние добавленные книги, либо книги, с последними добавленными главами.

В разделе книги должна быть предусмотрена навигация, с возможностью сортировки глав по убыванию и по возрастанию.

**2.2 Проектирование базы данных**

В базе данных данного приложения должны быть реализованы 3 таблицы.

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

* author\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* author\_name\_rus – строковый тип данных, максимальной длинной;
* author\_name\_eng – строковый тип данных, максимальной длинной;
* author\_description – строковый тип данных, максимальной длинной;

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

* book\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* book\_name\_rus – строковый тип данных, максимальной длинной;
* book\_name\_eng – строковый тип данных, максимальной длинной;
* book\_name\_orig – строковый тип данных, максимальной длинной;
* book\_name\_alt – строковый тип данных, максимальной длинной;
* book\_type – строковый тип данных, максимальной длинной;
* book\_year – строковый тип данных, максимальной длинной;
* book\_cover – строковый тип данных, максимальной длинной;
* book\_description – строковый тип данных, максимальной длинной;
* author\_id – целочисленный тип данных,
* translator\_id – целочисленный тип данных,
* user\_id – целочисленный тип данных

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

* translator\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* translator\_name – строковый тип данных, максимальной длинной;
* translator\_site – строковый тип данных, максимальной длинной;
* translator\_description – строковый тип данных, максимальной длинной;
* translator\_id – целочисленный тип данных

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

* user\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* user\_login – строковый тип данных, максимальной длинной;
* user\_password – строковый тип данных, максимальной длинной;
* user\_mail – строковый тип данных, максимальной длинной;
* user\_isadmin – целочисленный тип данных

В таблице «User\_Info» содержится информация о продуктах. Столбцы данной таблицы:

* user\_info\_id – целочисленный тип данных, не должен содержать значений null, является первичным ключом;
* user\_info\_name – строковый тип данных, максимальной длинной;
* user\_info\_lastname – строковый тип данных, максимальной длинной;
* user\_info\_sex – строковый тип данных, максимальной длинной;
* user\_info\_date\_of\_birth – строковый тип данных, максимальной длинной;
* user\_info\_country – строковый тип данных, максимальной длинной;
* user\_info\_about – строковый тип данных, максимальной длинной;
* user\_info\_pic – строковый тип данных, максимальной длинной;
* user\_id – целочисленный тип данных

**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.

**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.

Java – строго типизированный объектно-ориентированный язык программирования.

SQLite – компактная встраиваемая СУБД.

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

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

**3.2 Разработка баз данных**

Код создания структуры базы данных представлен в Приложении A.

Серверное приложение имеет следующие структурные объекты:

* Book – таблица продуктов;
* Author – таблица авторов;
* Translator – таблица рецептов.
* User – таблица пользователя

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. Приложение имеет следующую структуру.

**User:**

* Атрибуты: id, login, password, isAdmin, name, lastname, sex, date\_of\_birth, country, about, pic pic\_wallpaper.

**Book:**

* Атрибуты: id, name\_rus, name\_original, name\_eng, name\_alt, type, year\_relise, author, painter, translator, category, pic, description.

**Author:**

* Атрибуты: id, name\_rus, name\_eng, name\_orig, description.

**Translator:**

* Атрибуты: id, name, site, vkgroup, description.

**AuthorFragment:**

* Атрибуты: author, name\_rusTextView, name\_engTextView, name\_origTextView, descriptionTextView, db.
* Методы: onCreateView, onClick\_back.

**TranslatorFragment:**

* Атрибуты: translator, nameTextView, siteTextView, vkgroupTextView, descriptionTextVie, db.
* Методы: onCreateView, onClick\_back.

**AddBookFragment:**

* Атрибуты: name\_rusEditText, name\_originalEditText, name\_engEditText, name\_altEditText, typeEditText, year\_reliseEditText, authorEditText, painterEditText, translatorEditText, category, pic, description, db.
* Методы: onClick\_save, onCreateView

**AddAuthorFragment:**

* Атрибуты: name\_rusEditText, name\_engEditText, name\_origEditText, descriptionEditText, save\_button, db.
* Методы: onCreateView, onClick\_save.

**AddTranslatorFragment:**

* Атрибуты: nameEditText, siteEditText, vkgroupEditText, descriptionEditText, save\_button, db.
* Методы: onCreateView, onClick\_save.\

**BookAdapter:**

* Атрибуты: inflater, booklist.
* Методы: onCreateViewHolder, onBindViewHolder.

**DatabaseUtils:**

* Методы: onCreate, onUpgrade, onConfigure, selectBook, selectTranslator, selectAuthor, insertBook, insertAuthor, insertTranslator, deleteBook, deleteAuthor, deleteTranslator, updateBook, update, getAll, like.

**LoginActivity:**

* Атрибуты: context, loginEditText, passwordEditText, loginButton, registerButton, db.
* Методы: onCreate, onResume, onClick\_loginButton, onClick\_registerButton.

**RegisterActivity:**

* Атрибуты: context, loginEditText, passwordEditText, nameEditText, lastnameEditText, sexEditText, date\_of\_birthEditText, countryEditText, aboutEditText, picEditText, pic\_wallpaperEditText, registerButton, db.
* Методы: onCreate, onResume, onClick\_registerButton.

**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.

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

**4. Обеспечение безопасности приложения**

**4.1 Защита базы данных**

Для защиты базы данных была использована библиотека SQLCipher, которая используется для шифрования баз данных SQLite. Для того, чтобы воспользоваться данной библиотекой были проведены следующие шаги:

1. Добавление в файл «build.gragle» проекта зависимости «net.zetetic:android-database-sqlcipher:4.4.2@aar».
2. Замена импортированных классов в местах обращения к базе данных на классы, предоставляемые библиотекой.
3. В начало каждого метода обращения к базе данных добавлена строка «SQliteDatabase.loadlibs(context)», необходимая для работы библиотеки.

После выполнения описанных шагов база данных начинает шифроваться с помощью библиотеки SQLCipher,

**4.2 Запрет на запись и снимки экрана**

Для предотвращения возможных ошибок самого пользователя, а именно записи или снимков экранов приложения, содержащие конфиденциальные данные был реализован запрет на эти действия.

**4.3 Биометрическая аутентификация**

Для удобства пользователя реализована возможность входа в приложение при помощи отпечатка пальца. Код для реализации биометрической аутентификации был предоставлен документацией по платформе android. Для работы биометрической аутентификации необходимо иметь версию Android API не ниже 16.

**4.4 Обфускация**

В приложении реализована обфускация — приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

**4.5 Выводы по главе «Обеспечение безопасности приложения»**

В приложении были реализованы шифрование базы данных, запрет на снимок и запись экрана, биометрическая аутентификация, обфускация.

Этих мер достаточно для защиты данных пользователя и если он не будет совершать ошибок, основанных на человеческом факторе, то его данные, включая пароли, будут в безопасности.

**5. 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. При открытии приложения будет предоставлен экран для входа в приложение. Если у пользователя существует аккаунт, то ему необходимо ввести электронную почту и пароль и войти в приложение нажав кнопку «sign in». Если пользователь не зарегистрирован, то ему необходимо пройти регистрацию нажав кнопку «sign up».

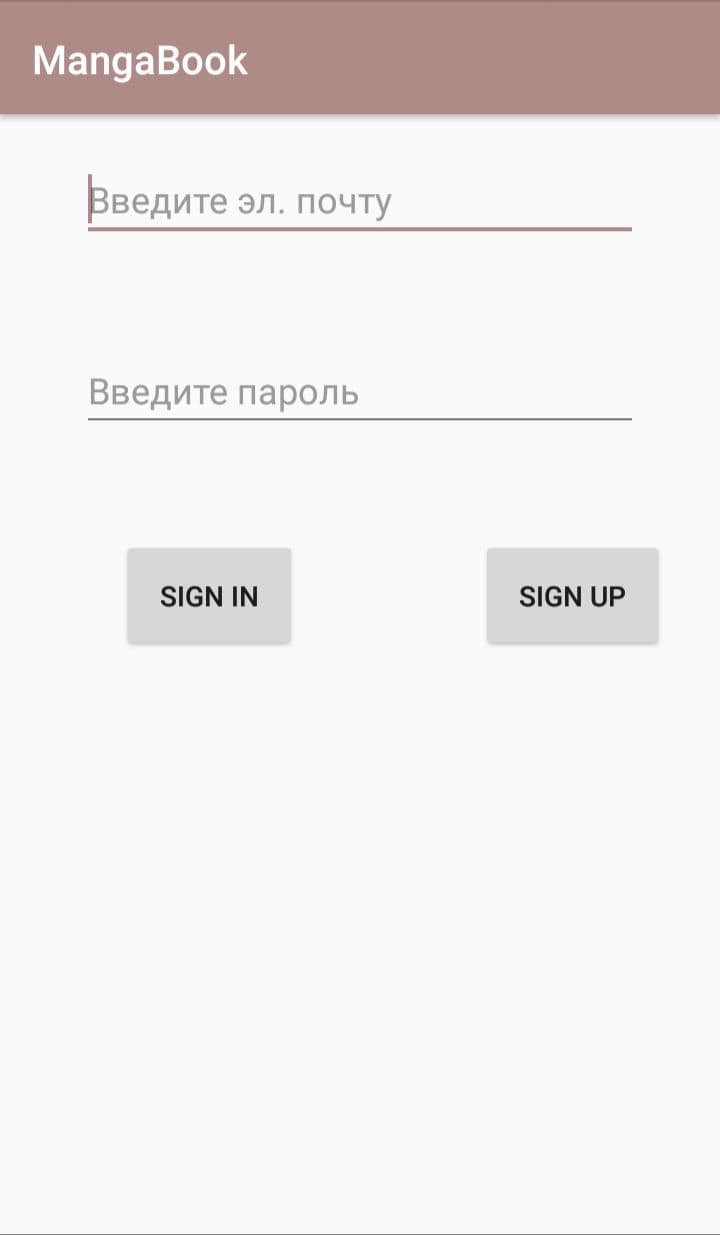


Рисунок 5.1 – Экран при открытии приложения

Пользователю будет предоставлен экран (Рисунок 5.2), в котором ему необходимо будет ввести свои данные для входа в приложение.

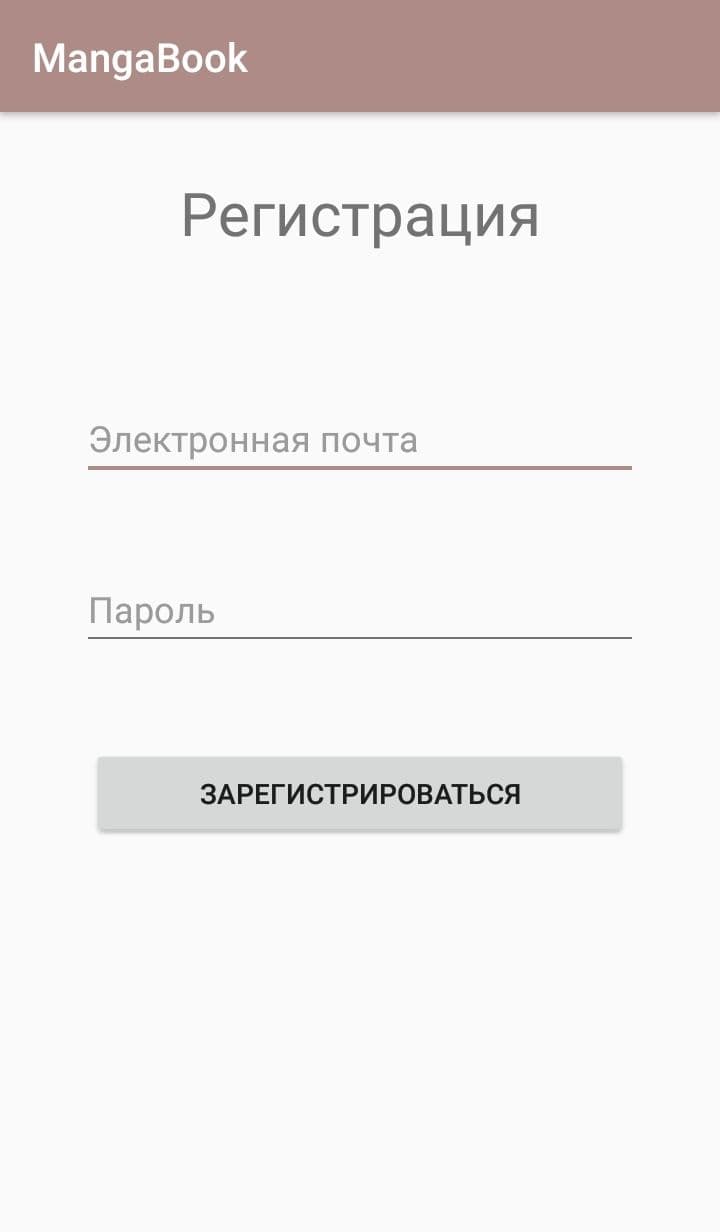


Рисунок 5.2 – Экран регистрации пользователя

После входа в приложение реализована биометрическая аутентификация по отпечатку пальца.

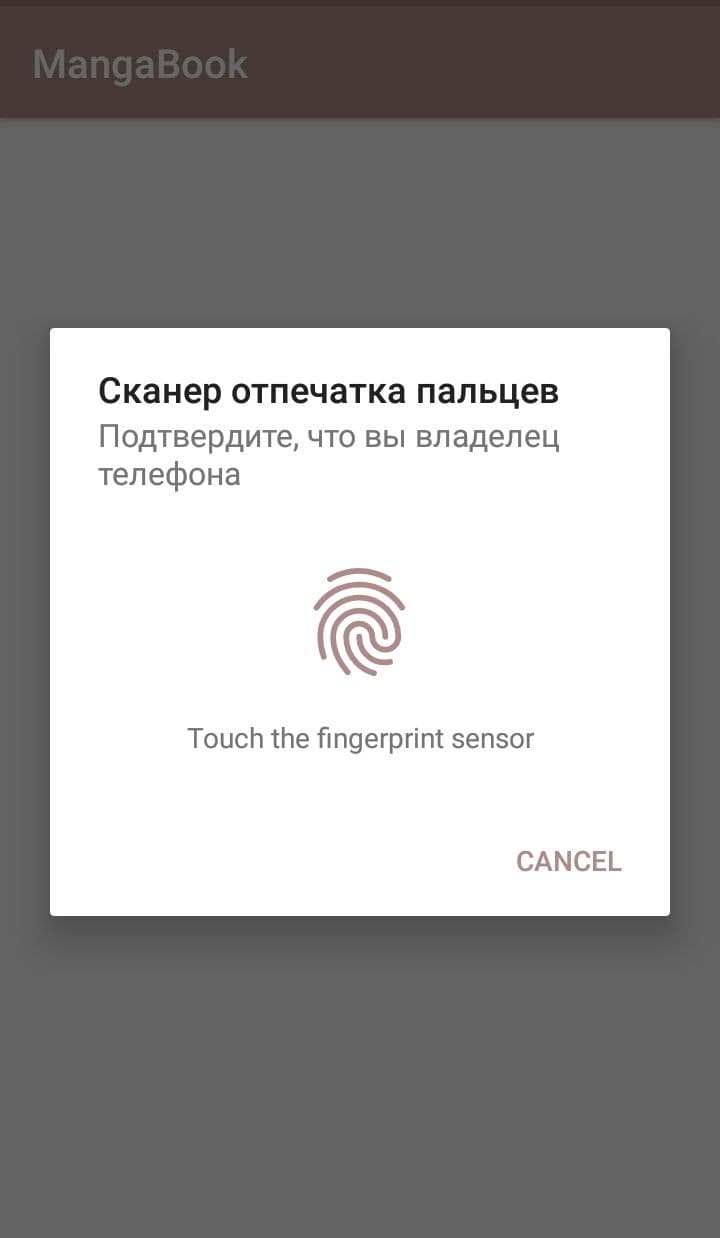


Рисунок 5.3 – Экран подтверждения биометрических данных

После того как устройство выявит сходство с владельцем, экран будет перенаправлен на каталог книг.

На скриншоте, который показан на рисунке ниже (Рисунок 4.4) можно увидеть меню приложения. С помощью него можно перейти на 4 экрана: каталог, авторы, переводчики, экран добавления книги.

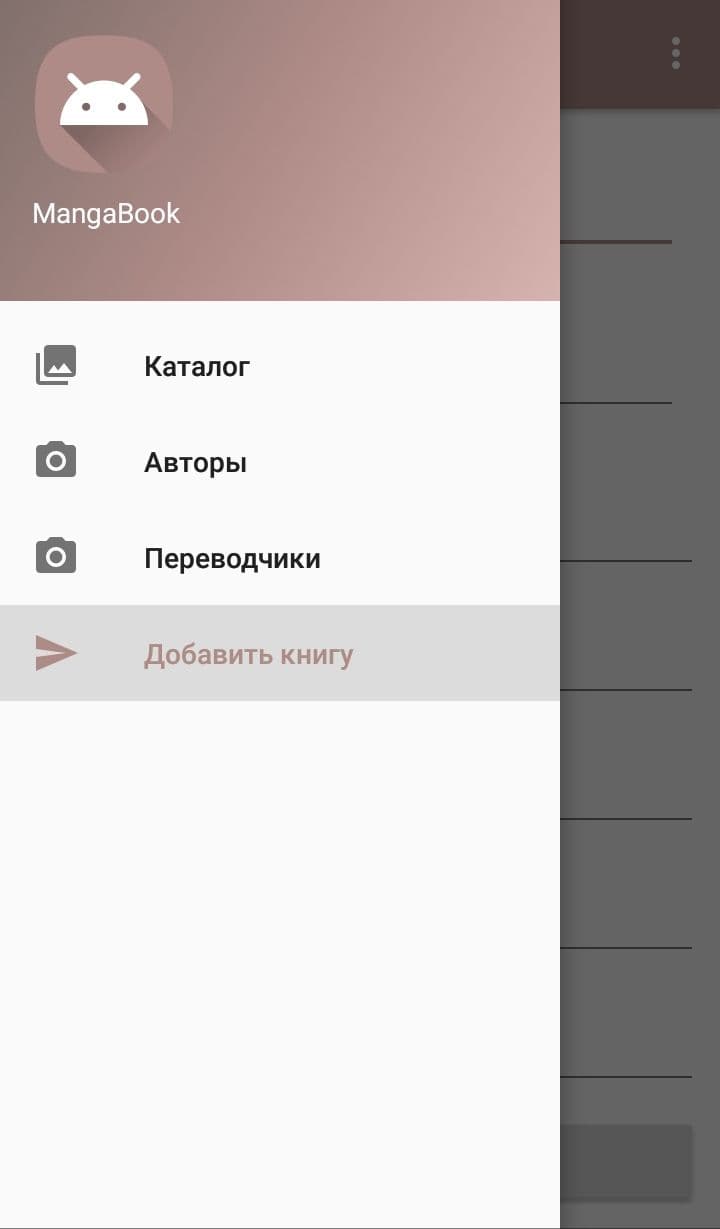


Рисунок 5.4. – Меню приложения

С помощью данного меню пользователь может зайти на страницу списка авторов книг и с помощью кнопки добавить автора (Рисунок 4.5).

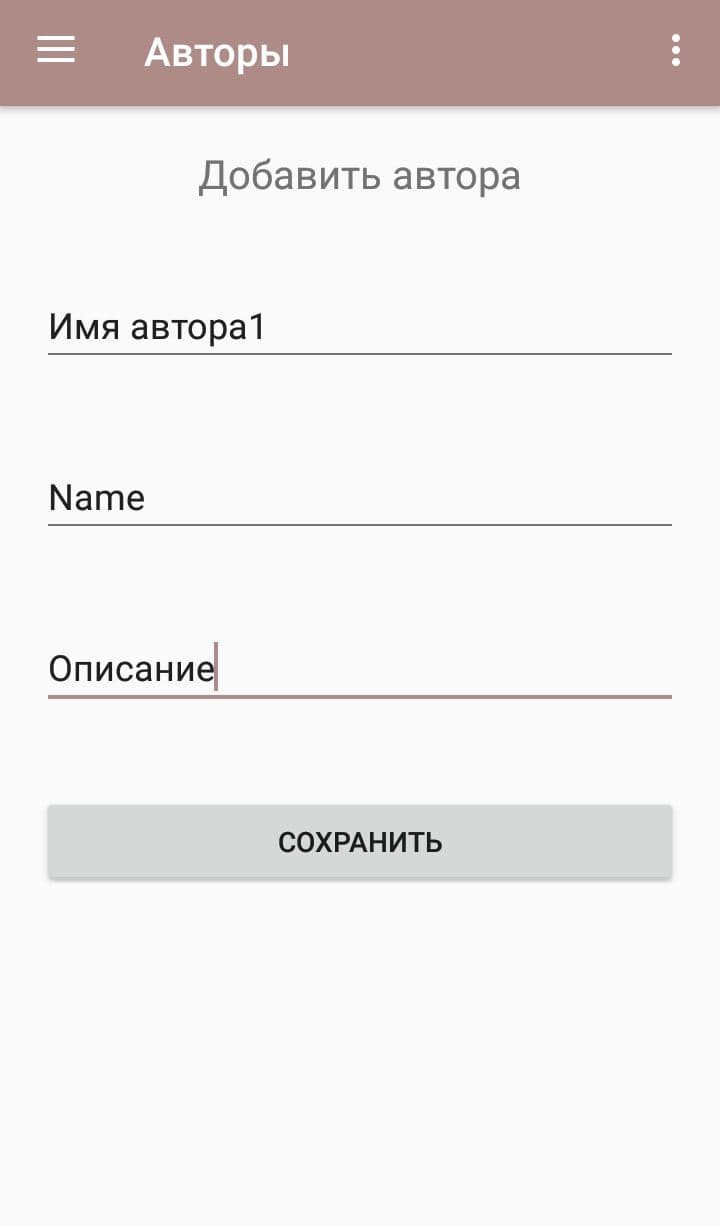
 

Рисунок 5.4 – Страницы для работы с авторами

Аналогичные действия проводятся и с переводчиком. При нажатии на элемент списка приложение перенаправит на страницу с данными, которые можно изменить, либо удалить выбранный элемент.

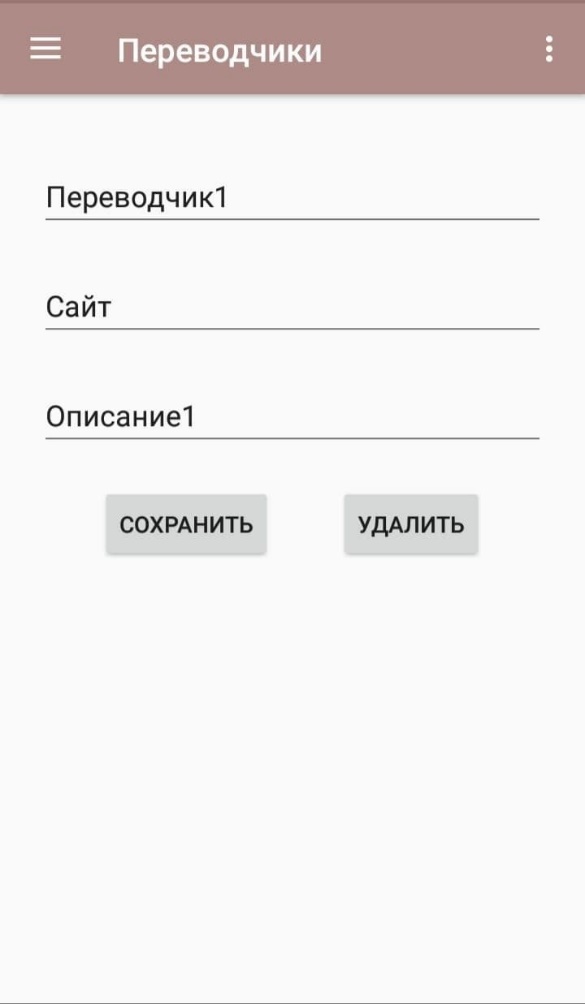


Рисунок 5.5 – Экран для изменения выбранного элемента

С помощью пункта меню создать книгу пользователь переходит на страницу создания книги, продемонстрированной на рисунке ниже (Рисунок 5.6)

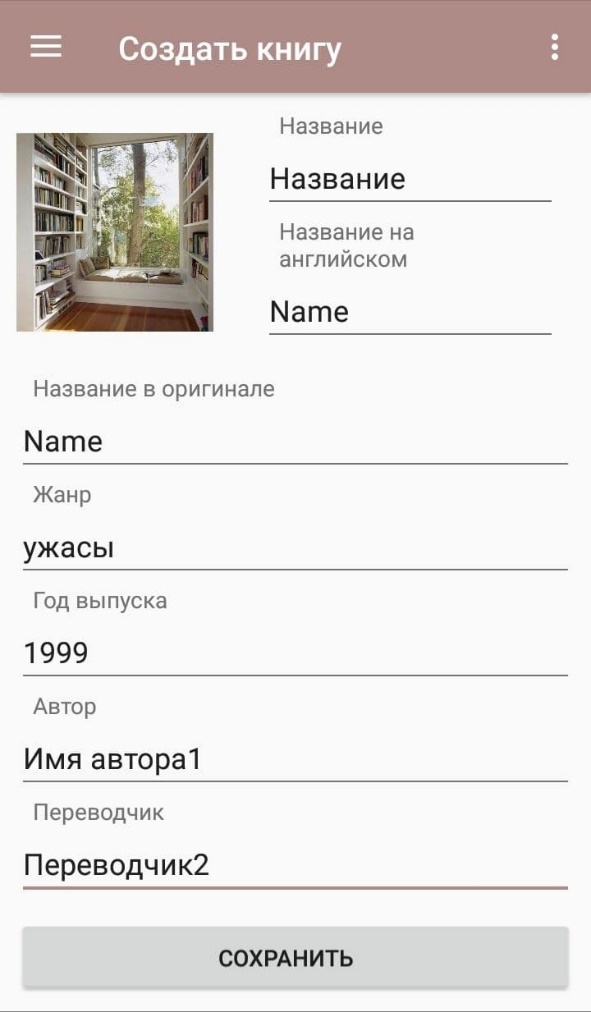


Рисунок 5.6 – Просмотр книг и создание новой книги

После нажатия кнопки «Сохранить» пользователь переходит на страницу каталога, на которой появляется новая книга.

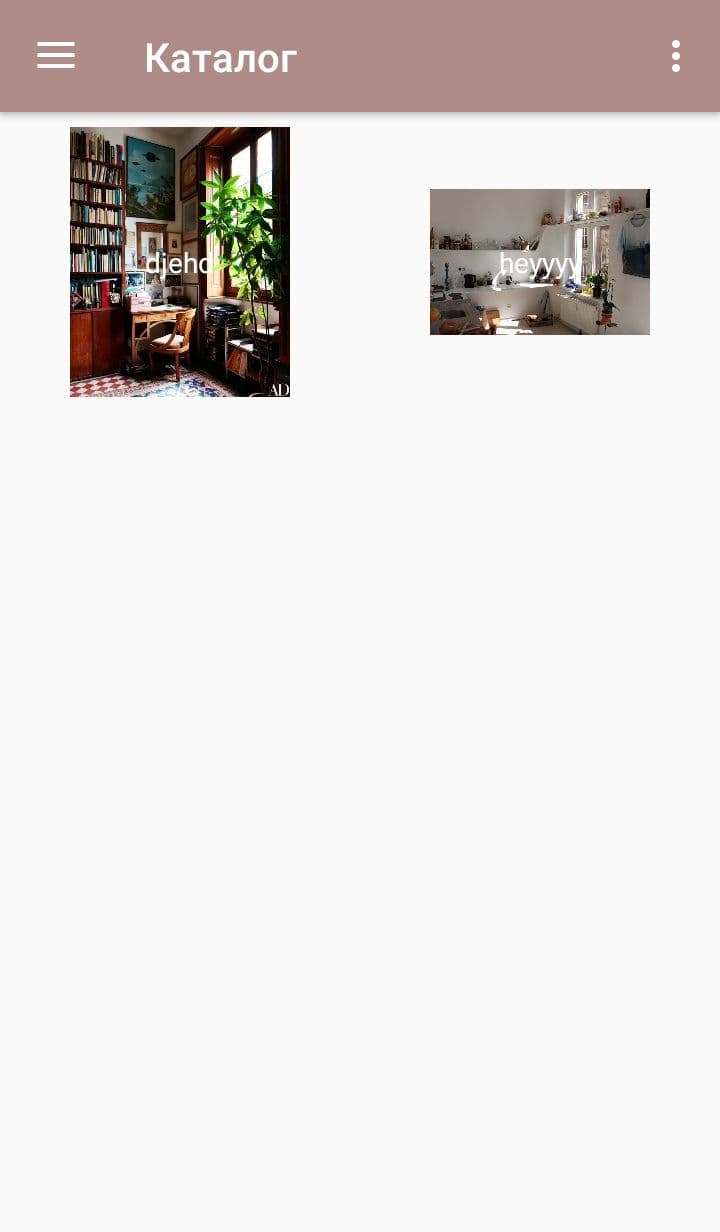
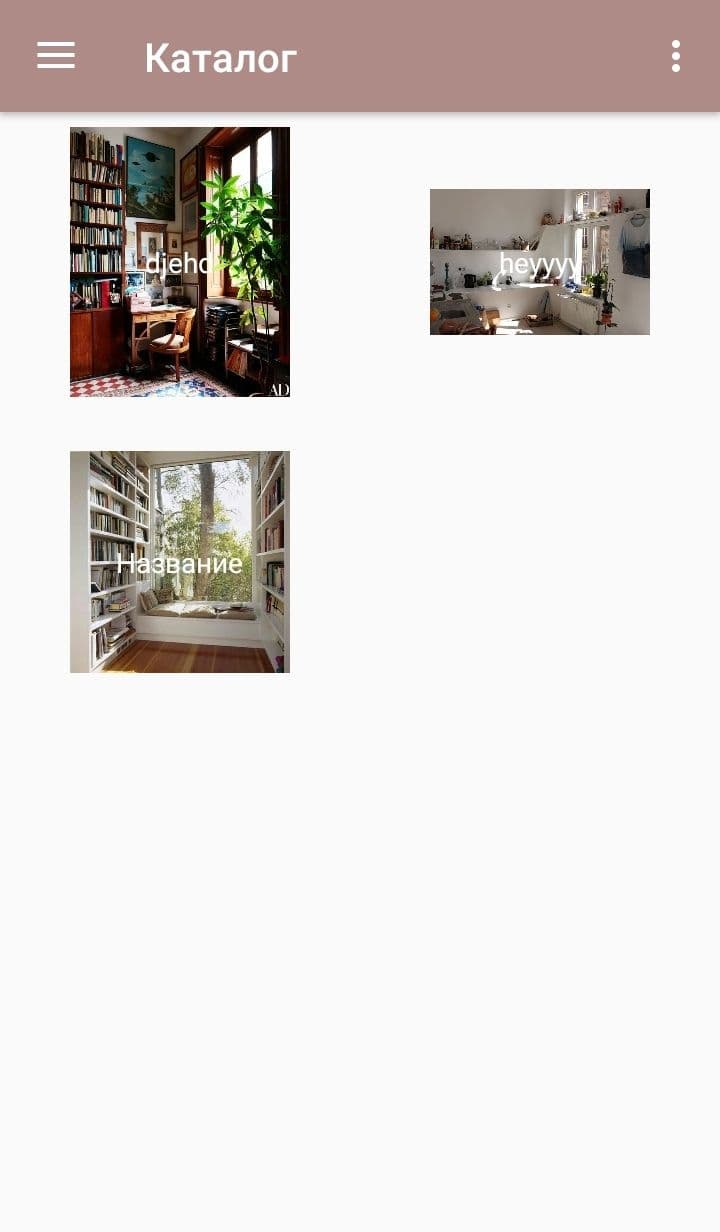
 

Рисунок 5.7 – Страницы каталога

При нажатии на элемент списка приложение перенаправит на страницу с данными, которые можно изменить, либо удалить выбранный элемент.

**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 промсотра книг.

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.

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

1. **Список использованных источников**
2. Android Studio [Электронный ресурс]. / Android Developers – Режим доступа:<https://developer.android.com/studio> – Дата доступа: 12.12.2020
3. \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: 12.12.2020
4. SQLite [Электронный ресурс] / [sqlite.org](https://www.sqlite.org/) – Режим доступа: <https://www.sqlite.org/2020/sqlite-android-3340000.aar> Дата доступа: 14.12.2020
5. SQLCipher [Электронный ресурс] / newtonsoft.com – Режим доступа: https://github.com/sqlcipher/android-database-sqlcipher.git/ Дата доступа: 10.12.2020
6. Firebase Authentification [Электронный ресурс] / google developers – Режим доступа: https://firebase.google.com/docs/auth / Дата доступа: 04.12.2020

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: 16.12.2020

**ПРИЛОЖЕНИЕ А**

Код создания базы данных

**package** belstu.diploma.mobile.mangabook.utils;  
  
**import** android.content.ContentValues;  
**import** android.content.Context;  
  
**import** net.sqlcipher.Cursor;  
**import** net.sqlcipher.database.SQLiteDatabase;  
**import** net.sqlcipher.database.SQLiteOpenHelper;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**import** belstu.diploma.mobile.mangabook.model.Author;  
**import** belstu.diploma.mobile.mangabook.model.Book;  
**import** belstu.diploma.mobile.mangabook.model.Bookmark;  
**import** belstu.diploma.mobile.mangabook.model.Chapter;  
**import** belstu.diploma.mobile.mangabook.model.Translator;  
**import** belstu.diploma.mobile.mangabook.model.User;  
  
**public class** DatabaseUtils **extends** SQLiteOpenHelper {  
  
 **private static** DatabaseUtils *instance*;  
  
 **private static final** String ***password*** = **"somepassword"**;  
  
 **private static final int *DATABASE\_VER*** = 1;  
 **public static final** String ***DATABASE\_NAME*** = **"manga.db"**;  
  
  
 **public** DatabaseUtils(Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VER***);  
 SQLiteDatabase.*loadLibs*(context);  
 }  
  
 @Override  
 **public void** onConfigure(SQLiteDatabase db) {  
 *//db.setForeignKeyConstraintsEnabled(true);* **super**.onConfigure(db);  
 db.execSQL(**"PRAGMA foreign\_keys = ON;"**);  
 }  
  
 @Override  
 **public void** onCreate(SQLiteDatabase sqLiteDatabase) {  
 sqLiteDatabase.execSQL(User.***CREATE\_TABLE\_USER***);  
 sqLiteDatabase.execSQL(Author.***CREATE\_TABLE\_AUTHOR***);  
 sqLiteDatabase.execSQL(Translator.***CREATE\_TABLE\_TRANSLATOR***);  
 sqLiteDatabase.execSQL(User.***CREATE\_TABLE\_USER\_INFO***);  
 sqLiteDatabase.execSQL(Book.***CREATE\_TABLE\_BOOK***);  
 sqLiteDatabase.execSQL(Chapter.***CREATE\_TABLE\_CHAPTER***);  
 sqLiteDatabase.execSQL(Bookmark.***CREATE\_TABLE\_BOOKMARK***);  
 }  
  
 @Override  
 **public void** onUpgrade(SQLiteDatabase sqLiteDatabase, **int** i, **int** i1) {  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + Bookmark.***TABLE\_BOOKMARK***);  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + Chapter.***TABLE\_CHAPTER***);  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + Book.***TABLE\_BOOK***);  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + User.***TABLE\_USER\_INFO***);  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + Translator.***TABLE\_TRANSLATOR***);  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + Author.***TABLE\_AUTHOR***);  
 sqLiteDatabase.execSQL(**"DROP TABLE IF EXISTS "** + User.***TABLE\_USER***);  
 onCreate(sqLiteDatabase);  
 }  
  
 **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\_BOOK\_NAME\_RUS***, book.getBookNameRus());  
 values.put(Book.***COLUMN\_BOOK\_NAME\_ENG***, book.getBookNameEng());  
 values.put(Book.***COLUMN\_BOOK\_NAME\_ORIG***, book.getBookNameOrig());  
 values.put(Book.***COLUMN\_BOOK\_NAME\_ALT***, book.getBookNameAlt());  
 values.put(Book.***COLUMN\_BOOK\_TYPE***, book.getBookType());  
 values.put(Book.***COLUMN\_BOOK\_YEAR***, book.getBookYear());  
 values.put(Book.***COLUMN\_BOOK\_COVER***, book.getCover());  
 values.put(Book.***COLUMN\_BOOK\_DESCRIPTION***, book.getBookDescription());  
 values.put(Author.***COLUMN\_AUTHOR\_ID***, book.getAuthorId());  
 values.put(Translator.***COLUMN\_TRANSLATOR\_ID***, book.getTranslatorId());  
 values.put(User.***COLUMN\_USER\_ID***, book.getUserId());  
  
 *// insert row* **long** id = db.insert(Book.***TABLE\_BOOK***, **null**, values);  
  
 *// close db connection* db.close();  
  
 *// return newly inserted row id* **return** id;  
 }  
  
 **public long** insertAuthor(Author author) {  
 *// 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(Author.***COLUMN\_AUTHOR\_NAME\_RUS***, author.getAuthorNameRus());  
 values.put(Author.***COLUMN\_AUTHOR\_NAME\_ENG***, author.getAuthorNameEng());  
 values.put(Author.***COLUMN\_AUTHOR\_DESCRIPTION***, author.getAuthorDescription());  
  
 *// insert row* **long** id = db.insert(Author.***TABLE\_AUTHOR***, **null**, values);  
  
 *// close db connection* db.close();  
  
 *// return newly inserted row id* **return** id;  
 }  
  
 **public long** insertTranslator(Translator translator) {  
 *// 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(Translator.***COLUMN\_TRANSLATOR\_NAME***, translator.getTranslatorName());  
 values.put(Translator.***COLUMN\_TRANSLATOR\_SITE***, translator.getTranslatorSite());  
 values.put(Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION***, translator.getTranslatorDescription());  
  
 *// insert row* **long** id = db.insert(Translator.***TABLE\_TRANSLATOR***, **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\_BOOK***,  
 **new** String[]{  
 Book.***COLUMN\_BOOK\_ID***,  
 Book.***COLUMN\_BOOK\_NAME\_RUS***,  
 Book.***COLUMN\_BOOK\_NAME\_ENG***,  
 Book.***COLUMN\_BOOK\_NAME\_ORIG***,  
 Book.***COLUMN\_BOOK\_NAME\_ALT***,  
 Book.***COLUMN\_BOOK\_TYPE***,  
 Book.***COLUMN\_BOOK\_YEAR***,  
 Book.***COLUMN\_BOOK\_COVER***,  
 Book.***COLUMN\_BOOK\_DESCRIPTION***,  
 Author.***COLUMN\_AUTHOR\_ID***,  
 Translator.***COLUMN\_TRANSLATOR\_ID***,  
 User.***COLUMN\_USER\_ID*** },  
 Book.***COLUMN\_BOOK\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(id)}, **null**, **null**, **null**, **null**);  
  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
  
 *// prepare note object* Book book = **new** Book();  
 book.setBookId(cursor.getInt(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_ID***)));  
 book.setBookNameRus(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_RUS***)));  
 book.setBookNameEng(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_ENG***)));  
 book.setBookNameOrig(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_ORIG***)));  
 book.setBookNameAlt(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_ALT***)));  
 book.setBookType(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_TYPE***)));  
 book.setBookYear(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_YEAR***)));  
 book.setCover(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_COVER***)));  
 book.setBookDescription(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_DESCRIPTION***)));  
 book.setAuthorId(cursor.getInt(cursor.getColumnIndex(Author.***COLUMN\_AUTHOR\_ID***)));  
 book.setTranslatorId(cursor.getInt(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_ID***)));  
 book.setUserId(cursor.getInt(cursor.getColumnIndex(User.***COLUMN\_USER\_ID***)));  
  
 *// close the db connection* cursor.close();  
  
 **return** book;  
 }  
 **public** Translator getTranslator(**long** id) {  
 *// get readable database* SQLiteDatabase db = **this**.getReadableDatabase();  
  
 Cursor cursor = db.query(Translator.***TABLE\_TRANSLATOR***,  
 **new** String[]{  
 Translator.***COLUMN\_TRANSLATOR\_ID***,  
 Translator.***COLUMN\_TRANSLATOR\_NAME***,  
 Translator.***COLUMN\_TRANSLATOR\_SITE***,  
 Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION*** },  
 Translator.***COLUMN\_TRANSLATOR\_ID*** + **"=?"**,  
 **new** String[]{String.*valueOf*(id)}, **null**, **null**, **null**, **null**);  
  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
  
 *// prepare note object* Translator translator = **new** Translator();  
 translator.setTranslatorId(cursor.getInt(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_ID***)));  
 translator.setTranslatorName(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_NAME***)));  
 translator.setTranslatorSite(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_SITE***)));  
 translator.setTranslatorDescription(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION***)));  
  
 *// close the db connection* cursor.close();  
  
 **return** translator;  
 }  
 **public** Translator getTranslatorByName(String name) {  
 *// get readable database* SQLiteDatabase db = **this**.getReadableDatabase();  
  
 Cursor cursor = db.query(Translator.***TABLE\_TRANSLATOR***,  
 **new** String[]{  
 Translator.***COLUMN\_TRANSLATOR\_ID***,  
 Translator.***COLUMN\_TRANSLATOR\_NAME***,  
 Translator.***COLUMN\_TRANSLATOR\_SITE***,  
 Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION*** },  
 Translator.***COLUMN\_TRANSLATOR\_NAME*** + **"=?"**,  
 **new** String[]{String.*valueOf*(name)}, **null**, **null**, **null**, **null**);  
  
 **if** (cursor != **null**)  
 cursor.moveToFirst();  
  
 *// prepare note object* Translator translator = **new** Translator();  
 translator.setTranslatorId(cursor.getInt(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_ID***)));  
 translator.setTranslatorName(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_NAME***)));  
 translator.setTranslatorSite(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_SITE***)));  
 translator.setTranslatorDescription(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION***)));  
  
 *// close the db connection* cursor.close();  
  
 **return** translator;  
 }  
  
 **public** List<Book> getAllBooks() {  
 List<Book> books = **new** ArrayList<>();  
  
 *// Select All Query* String selectQuery = **"SELECT \* FROM "** + Book.***TABLE\_BOOK***;  
*// + " WHERE " + User.COLUMN\_USER\_ID + "= " + user\_id;* SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.query(Book.***TABLE\_BOOK***,  
 **new** String[]{  
 Book.***COLUMN\_BOOK\_ID***,  
 Book.***COLUMN\_BOOK\_NAME\_RUS***,  
 Book.***COLUMN\_BOOK\_NAME\_ENG***,  
 Book.***COLUMN\_BOOK\_NAME\_ORIG***,  
 Book.***COLUMN\_BOOK\_NAME\_ALT***,  
 Book.***COLUMN\_BOOK\_TYPE***,  
 Book.***COLUMN\_BOOK\_YEAR***,  
 Book.***COLUMN\_BOOK\_COVER***,  
 Book.***COLUMN\_BOOK\_DESCRIPTION***,  
 Author.***COLUMN\_AUTHOR\_ID***,  
 Translator.***COLUMN\_TRANSLATOR\_ID***,  
 User.***COLUMN\_USER\_ID*** },  
 **null**, **null**, **null**, **null**,  
 **null**, **null**);  
  
 *// looping through all rows and adding to list* **if** (cursor.moveToFirst()) {  
 **do** {  
 Book book = **new** Book();  
 book.setBookId(cursor.getInt(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_ID***)));  
 book.setBookNameRus(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_RUS***)));  
 book.setBookNameEng(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_ENG***)));  
 book.setBookNameOrig(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_ORIG***)));  
 book.setBookNameAlt(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_NAME\_ALT***)));  
 book.setBookType(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_TYPE***)));  
 book.setBookYear(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_YEAR***)));  
 book.setCover(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_COVER***)));  
 book.setBookDescription(cursor.getString(cursor.getColumnIndex(Book.***COLUMN\_BOOK\_DESCRIPTION***)));  
 book.setAuthorId(cursor.getInt(cursor.getColumnIndex(Author.***COLUMN\_AUTHOR\_ID***)));  
 book.setTranslatorId(cursor.getInt(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_ID***)));  
 book.setUserId(cursor.getInt(cursor.getColumnIndex(User.***COLUMN\_USER\_ID***)));  
  
 books.add(book);  
 } **while** (cursor.moveToNext());  
 }  
  
 *//* cursor.close();  
 *// close db connection* db.close();  
  
 *// return notes list* **return** books;  
 }  
  
 **public** List<Author> getAllAuthors() {  
 List<Author> authors = **new** ArrayList<>();  
  
 *// Select All Query* String selectQuery = **"SELECT \* FROM "** + Author.***TABLE\_AUTHOR***;  
  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.query(Author.***TABLE\_AUTHOR***,  
 **new** String[]{  
 Author.***COLUMN\_AUTHOR\_ID***,  
 Author.***COLUMN\_AUTHOR\_NAME\_RUS***,  
 Author.***COLUMN\_AUTHOR\_NAME\_ENG***,  
 Author.***COLUMN\_AUTHOR\_DESCRIPTION*** },  
 **null**, **null**, **null**, **null**,  
 **null**, **null**);  
  
 *// looping through all rows and adding to list* **if** (cursor.moveToFirst()) {  
 **do** {  
 Author author = **new** Author();  
 author.setAuthorId(cursor.getInt(cursor.getColumnIndex(Author.***COLUMN\_AUTHOR\_ID***)));  
 author.setAuthorNameRus(cursor.getString(cursor.getColumnIndex(Author.***COLUMN\_AUTHOR\_NAME\_RUS***)));  
 author.setAuthorNameEng(cursor.getString(cursor.getColumnIndex(Author.***COLUMN\_AUTHOR\_NAME\_ENG***)));  
 author.setAuthorDescription(cursor.getString(cursor.getColumnIndex(Author.***COLUMN\_AUTHOR\_DESCRIPTION***)));  
 authors.add(author);  
 } **while** (cursor.moveToNext());  
 }  
  
 *//* cursor.close();  
 *// close db connection* db.close();  
  
 *// return notes list* **return** authors;  
 }  
  
 **public** List<Translator> getAllTranslators() {  
 List<Translator> translators = **new** ArrayList<>();  
  
 *// Select All Query* String selectQuery = **"SELECT \* FROM "** + Translator.***TABLE\_TRANSLATOR***;  
  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.query(Translator.***TABLE\_TRANSLATOR***,  
 **new** String[]{  
 Translator.***COLUMN\_TRANSLATOR\_ID***,  
 Translator.***COLUMN\_TRANSLATOR\_NAME***,  
 Translator.***COLUMN\_TRANSLATOR\_SITE***,  
 Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION*** },  
 **null**, **null**, **null**, **null**,  
 **null**, **null**);  
  
 *// looping through all rows and adding to list* **if** (cursor.moveToFirst()) {  
 **do** {  
 Translator translator = **new** Translator();  
 translator.setTranslatorId(cursor.getInt(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_ID***)));  
 translator.setTranslatorName(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_NAME***)));  
 translator.setTranslatorSite(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_SITE***)));  
 translator.setTranslatorDescription(cursor.getString(cursor.getColumnIndex(Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION***)));  
 translators.add(translator);  
 } **while** (cursor.moveToNext());  
 }  
  
 *//* cursor.close();  
 *// close db connection* db.close();  
  
 *// return notes list* **return** translators;  
 }  
  
 **public void** updateBook(Book book) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(Book.***COLUMN\_BOOK\_NAME\_RUS***, book.getBookNameRus());  
 values.put(Book.***COLUMN\_BOOK\_NAME\_ENG***, book.getBookNameEng());  
 values.put(Book.***COLUMN\_BOOK\_NAME\_ORIG***, book.getBookNameOrig());  
 values.put(Book.***COLUMN\_BOOK\_NAME\_ALT***, book.getBookNameAlt());  
 values.put(Book.***COLUMN\_BOOK\_TYPE***, book.getBookType());  
 values.put(Book.***COLUMN\_BOOK\_YEAR***, book.getBookYear());  
 values.put(Book.***COLUMN\_BOOK\_COVER***, book.getCover());  
 values.put(Book.***COLUMN\_BOOK\_DESCRIPTION***, book.getBookDescription());  
 values.put(Author.***COLUMN\_AUTHOR\_ID***, book.getAuthorId());  
 values.put(Translator.***COLUMN\_TRANSLATOR\_ID***, book.getTranslatorId());  
 values.put(User.***COLUMN\_USER\_ID***, book.getUserId());  
  
 *// updating row* db.update(Book.***TABLE\_BOOK***, values, Book.***COLUMN\_BOOK\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(book.getBookId())});  
 }  
  
 **public void** updateTranslator(Translator translator) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(Translator.***COLUMN\_TRANSLATOR\_NAME***, translator.getTranslatorName());  
 values.put(Translator.***COLUMN\_TRANSLATOR\_SITE***, translator.getTranslatorSite());  
 values.put(Translator.***COLUMN\_TRANSLATOR\_DESCRIPTION***, translator.getTranslatorDescription());  
  
 *// updating row* db.update(Translator.***TABLE\_TRANSLATOR***, values, Translator.***COLUMN\_TRANSLATOR\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(translator.getTranslatorId())});  
 }  
  
 **public void** deleteBook(Book book) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 db.delete(Book.***TABLE\_BOOK***, Book.***COLUMN\_BOOK\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(book.getBookId())});  
 db.close();  
 }  
 **public void** deleteTranslator(Translator translator) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 db.delete(Translator.***TABLE\_TRANSLATOR***, Translator.***COLUMN\_TRANSLATOR\_ID*** + **" = ?"**,  
 **new** String[]{String.*valueOf*(translator.getTranslatorId())});  
 db.close();  
 }  
 **public** SQLiteDatabase getReadableDatabase() {  
 **return** (**super**.getReadableDatabase(***password***));  
 }  
  
 **public** SQLiteDatabase getWritableDatabase() {  
 **return** (**super**.getWritableDatabase(***password***));  
 }  
  
}

**ПРИЛОЖЕНИЕ Б**

Код создания аутентификации по отпечатку пальца

**package** belstu.diploma.mobile.mangabook;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
**import** androidx.biometric.BiometricPrompt;  
  
**import** android.os.Bundle;  
  
**import** java.util.concurrent.Executor;  
**import** java.util.concurrent.Executors;  
  
**import** androidx.fragment.app.FragmentActivity;  
  
**import** android.util.Log;  
**import** android.view.View;  
  
**import** androidx.annotation.NonNull;  
  
**public class** FingerPrintController **extends** AppCompatActivity {  
  
 **private static final** String TAG = FingerPrintController.**class**.getName();  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 *//setContentView(R.layout.activity\_finger\_print\_controller);  
  
//Create a thread pool with a single thread//* Executor newExecutor = Executors.newSingleThreadExecutor();  
  
 FragmentActivity activity = **this**;  
  
*//Start listening for authentication events//* **final** BiometricPrompt myBiometricPrompt = **new** BiometricPrompt(activity, newExecutor, **new** BiometricPrompt.AuthenticationCallback() {  
 @Override  
  
*//onAuthenticationError is called when a fatal error occurrs//* **public void** onAuthenticationError(**int** errorCode, @NonNull CharSequence errString) {  
 **super**.onAuthenticationError(errorCode, errString);  
 **if** (errorCode == BiometricPrompt.ERROR\_NEGATIVE\_BUTTON) {  
 } **else** {  
  
*//Print a message to Logcat//* Log.d(TAG, **"An unrecoverable error occurred"**);  
 }  
 }  
  
*//onAuthenticationSucceeded is called when a fingerprint is matched successfully//* @Override  
 **public void** onAuthenticationSucceeded(@NonNull BiometricPrompt.AuthenticationResult result) {  
 **super**.onAuthenticationSucceeded(result);  
  
*//Print a message to Logcat//* Log.d(TAG, **"Fingerprint recognised successfully"**);  
 finish();  
 }  
  
*//onAuthenticationFailed is called when the fingerprint doesn’t match//* @Override  
 **public void** onAuthenticationFailed() {  
 **super**.onAuthenticationFailed();  
  
*//Print a message to Logcat//* Log.d(TAG, **"Fingerprint not recognised"**);  
 }  
 });  
  
*//Create the BiometricPrompt instance//* **final** BiometricPrompt.PromptInfo promptInfo = **new** BiometricPrompt.PromptInfo.Builder()  
  
*//Add some text to the dialog//* .setTitle(**"Сканер отпечатка пальцев"**)  
 .setSubtitle(**""**)  
 .setDescription(**"Подтвердите, что вы владелец телефона"**)  
 .setNegativeButtonText(**"Cancel"**)  
  
*//Build the dialog//* .build();  
  
*//Assign an onClickListener to the app’s “Authentication” button//* myBiometricPrompt.authenticate(promptInfo);  
  
 }  
}