

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики
Кафедра программного обеспечения и администрирования информационных
систем


Разработка мобильного приложения «Помощник читателя»

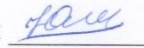
Бакалаврская работа

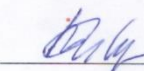
Направление 09.03.03 Прикладная информатика

Профиль Прикладная информатика в юриспруденции



Зав. кафедрой  д. ф.-м. н., проф. Артёмов М. А. __. __. 2023 г.

Обучающийся  4 курс 10 группа Собакарь Ю. Н.

Руководитель  к.т.н., доц. Курченкова Т. В.

Воронеж 2023

Аннотация

В работе представлен процесс разработки мобильного приложения «Помощник читателя». В работе рассмотрены аналогичные программные решения, требования к программному и аппаратному обеспечению, описан интерфейс и программная реализация приложения.

Содержание

| | |
|---|----|
| Введение..... | 5 |
| 1. Постановка задачи..... | 7 |
| 2. Обзор и анализ существующих решений | 8 |
| 3. Средства реализации..... | 10 |
| 4. Требования к программному и аппаратному обеспечению | 11 |
| 5. Реализация | 12 |
| 5.1 Диаграмма вариантов использования | 12 |
| 5.2 Модель базы данных..... | 13 |
| 5.3. Реализация базы данных..... | 15 |
| 5.4. Архитектура приложения..... | 15 |
| 6. Интерфейс | 18 |
| 7. Тестирование | 29 |
| 7.1. Тест 1. Добавление произведения | 29 |
| 7.2. Тест 2. Редактирование существующего произведения. | 29 |
| 7.3 Тест 3. Удаление произведения..... | 29 |
| 7.4 Тест 4. Добавление события в календарь. | 30 |
| 7.5 Тест 5. Добавление дополнительной информации о произведении | 30 |
| 7.6 Тест 6. Добавление персонажа..... | 30 |
| 7.7 Тест 7. Добавление данных о персонаже..... | 31 |
| 7.8 Тест 8. Добавление данных об отношениях персонажа..... | 31 |
| 7.9. Тест 9. Добавление цитаты | 32 |
| 7.10 Тест 10. Проверка графа взаимоотношений персонажей. | 32 |
| Заключение | 33 |
| Список литературы | 34 |
| Приложение 1. Класс базы данных и таблиц | 36 |
| Приложение 2. Листинг MainActivity.kt..... | 39 |

| | |
|---|----|
| Приложение 3. Листинг BookDao.kt | 40 |
| Приложение 4. Экран графа связей между персонажами..... | 43 |

Введение

Чтение художественных произведений является не только любимым занятием многих людей, но и может быть необходимым при изучении предмета литература в школе или вузе. Достижению более осознанного понимания, запоминания и анализирования прочитанного может способствовать ведение записей. Конспектирование помогает при необходимости вернуться к тексту, освежить в памяти основные мысли или даже переосмыслить те или иные моменты произведения. Кроме того, ведение записей может сэкономить время при подготовке к экзамену или написанию реферата.

Художественная литература характеризуется наличием персонажей, особенно если это роман, где героев много. Это означает, что ключевым компонентом анализа текста является записывание информации о каждом герое в отдельности и составление системы персонажей. Такая работа с произведением способствует систематизированному и глубокому пониманию прочитанного и помогает собрать более детальную и полную картину произведения.

В настоящее время мобильный телефон стал неотъемлемой частью жизни любого человека и заменяет множество предметов и устройств, таких как записные книжки, фотоаппараты, музыкальные плееры и т. д. Если говорить о чтении, то электронные и аудио книги становятся популярнее, чем их бумажные предшественники за счет своей портативности. Мобильное приложение для читателя может помочь записывать и систематизировать информацию о прочитанной книге и её персонажах в любой момент времени.

Сейчас Android-устройства наиболее распространены на территории Российской Федерации, поэтому приложение, разработанное для этой операционной системы, получит большую аудиторию и будет востребованным.

Целью работы является разработка мобильного приложения «Помощник читателя» для операционной системы Android [1], которое позволит пользователю вести литературные заметки о произведении и его персонажах в удобном формате.

1. Постановка задачи

Для достижения цели необходимо создать мобильное приложение [2] «Помощник читателя», которое позволит конечному пользователю добавлять новое произведение, редактировать информацию о произведении, добавлять персонажей и информацию о них, визуализировать связи между персонажами и вести календарь.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать доступные существующие решения;
- выбрать средства реализации;
- спроектировать диаграмму сценариев для разрабатываемого приложения;
- разработать базу данных приложения;
- разработать интерфейс;
- реализовать приложение, решающее поставленную задачу;
- протестировать приложение.

2. Обзор и анализ существующих решений

На данный момент нет полных аналогов для решения данной задачи, но существуют способы, которые можно использовать для достижения цели. Для ведения записей о произведении используются рукописные заметки, текстовые редакторы или мобильные приложения для электронного читательского дневника, такие как «Book Diary» [3]. Кратко опишем представленные способы.

Рукописные заметки: информация записывается от руки в тетради или на листах бумаги. Из плюсов можно выделить, что каждый может выбрать удобный и понятный для себя формат записи. Однако этот подход требует больше затрат по времени, так как нет возможности скопировать уже готовый текст, например, цитату.

Текстовые редакторы: этот подход устраняет недостаток с большим количеством временных затрат на переписывание информации, которые уже есть в готовом электронном виде, но данные также приходится структурировать вручную. Есть возможность создать таблицу, но таблицы обычно трудно читаются с мобильных устройств.

Специализированные мобильные приложения для ведение читательского дневника: к таким программам можно отнести Book Diary. Эта программа позволяет писать собственные описания к произведениям, вести календарь и статистику прочитанных книг. Однако здесь нет функций для добавления записей по каждому персонажу в отдельности.

Таким образом можно заметить, что ни один из методов не решает поставленную задачу полностью, и пользователю необходимо либо самостоятельно организовывать структуру записей о произведении, либо ограничиваться функциональностью приложения. Рассмотрим плюсы и минусы каждого подхода в таблице 1.

Таблица 1. Плюсы и минусы методов ведения записей о произведении

| | Рукописные заметки | Текстовые редакторы | Приложение для электронного читательского дневника |
|---|-----------------------|------------------------|---|
| Наличие шаблона оформления записей | — | + / — | + |
| Возможность повторного использования текстов | — | + | + |
| Использование на мобильном устройстве | — | + | + |
| Ведение записей о персонажах | + | + | — |
| Структурирование данных | вручную | вручную | полуавтоматически |
| Ведение календаря | вручную | вручную | полуавтоматически |
| Составление графа связей между персонажами | вручную | вручную | — |

3. Средства реализации

При реализации проекта использовались следующие программные средства:

- интегрированная среда разработки Android Studio [4];
- язык программирования Kotlin [5];
- язык разметки XML;
- СУБД SQLite [6];
- программный инструмент моделирования WhiteStarUML [7];
- библиотека Room [8];
- библиотека GraphView [9];
- библиотека Material CalendarView [10]

Библиотека Room является частью Android Architecture Components [12], библиотек, которые соответствуют рекомендациями построения архитектуры приложений Google, и предназначена для работы с базами данных SQLite. Она упрощает процесс создания запросов и работы с данными. Среди преимуществ можно выделить проверку SQL-запросов во время компиляции и удобные аннотации, которые минимизируют повторяющийся шаблонный код.

Библиотека Android GraphView используется для построения графов. С её помощью можно отображать данные в виде неориентированного графа, что решает поставленную задачу визуализации связей персонажей между собой.

Библиотека Material CalendarView использована для создания пользовательского календаря. Она позволяет настраивать различные аспекты календаря, такие как цвета, шрифты, виды выделения и др. Это помогает адаптировать календарь под дизайн приложения. Библиотека позволяет отображать даты в календаре и добавлять события. Также её преимуществом является поддержка всех языков, которые поддерживаются операционной системой Android, что легко позволяет настроить календарь для русскоязычных пользователей.

4. Требования к программному и аппаратному обеспечению

Поскольку Android [12] имеет открытый исходный код, фиксированных конфигураций аппаратного и программного обеспечения не существует.

Для совместимости с мобильным устройством требуется:

- версия операционной системы — Android 7.0 и более поздние версии;
- 43 МБ свободного места на устройстве.

5. Реализация

Приложение является клиентским, в котором обеспечено интерактивное взаимодействие пользователя с системой. Все данные пользователя хранятся в локальной базе данных.

5.1 Диаграмма вариантов использования

Основная функциональность приложения представлена на диаграмме вариантов использования (рис 5.1).

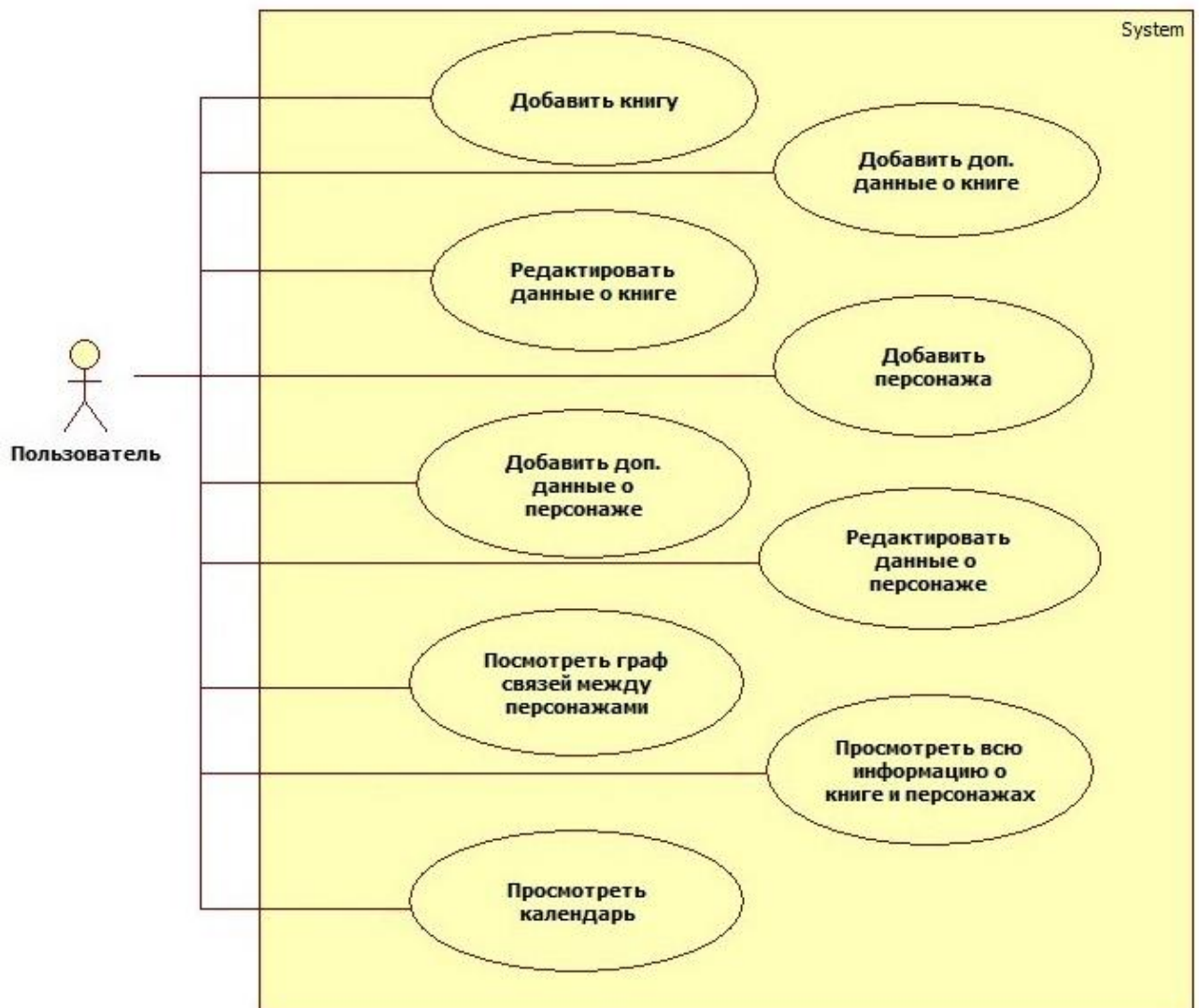


Рис. 5.1 Диаграмма вариантов использования

5.2 Модель базы данных

Подробное описание структуры базы данных приведено в табл. 2.

Таблица 2. Описание таблиц базы данных приложения «Помощник читателя»

| Таблица | Столбец | Описание |
|--|------------------|---|
| Book (Книга) | book_id | Идентификатор книги |
| | book_name | Название книги |
| | book_author | Автор книги |
| | year | Год создания |
| | calendar_date | Запланированная дата прочтения |
| | is_read | Флаг — отмечена ли книга как прочитанная |
| OtherInfBook (Дополнительная информация о книге) | otherInf_id | Идентификатор дополнительной информации |
| | information | Текст дополнительной информации |
| | book_id | Идентификатор книги, к которой принадлежит дополнительная информация |
| Character (Персонаж) | character_id | Идентификатор персонажа |
| | character_name | Имя персонажа |
| | biography | Биографическое описание персонажа |
| | portrait | Портретное описание персонажа |
| | book_id | Идентификатор книги, к которой относится персонаж |
| OtherInfCharacter (Дополнительная информация о персонаже) | otherInf_char_id | Идентификатор дополнительной информации о персонаже |
| | otherInf_text | Текст дополнительной информации о персонаже |
| | character_id | Идентификатор персонажа, которому принадлежит дополнительная информация |
| Quote | quote_id | Идентификатор цитаты |

| Таблица | Столбец | Описание |
|--|----------------------|--|
| (Цитата персонажа) | quote_text | Текст цитаты |
| | character_id | Идентификатор персонажа, которому принадлежит цитата |
| CharacterRelation (Связь между персонажами) | character_id_1 | Идентификатор персонажа |
| | character_id_2 | Идентификатор персонажа, с которым добавляется связь |
| | description_relation | Описание связи между персонажами |

На рисунке 5.2. представлена логическая модель базы данных.

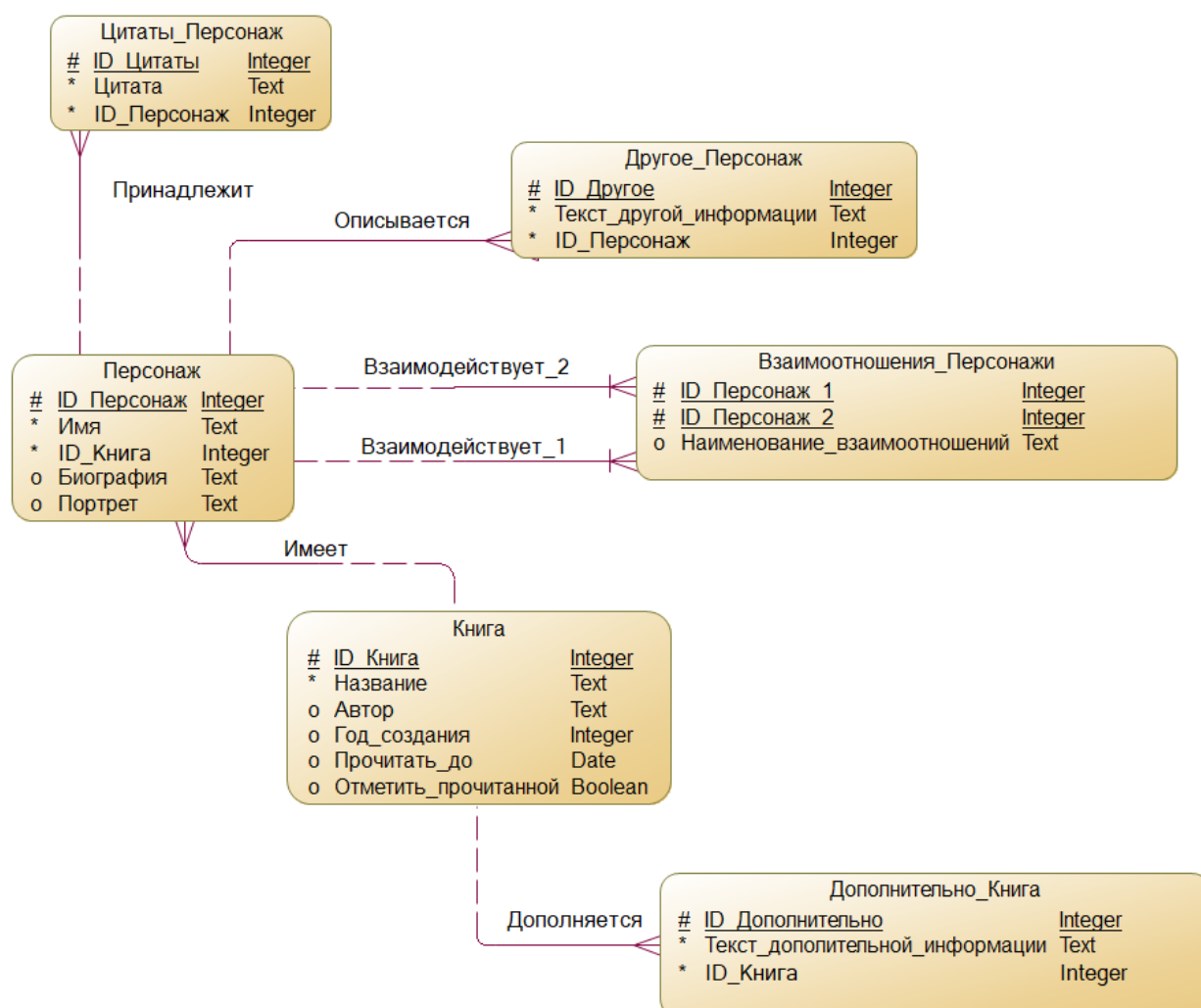


Рис. 5.2. Логическая модель базы данных

5.3. Реализация базы данных

База данных реализована с помощью библиотеки Room, которая обеспечивает уровень абстракции над СУБД SQLite.

Листинг кода класса базы данных и классов, содержащихся в ней таблиц, представлен в Приложении 1.

5.4. Архитектура приложения

В начале работы над приложением создан новый проект в интегрированной среде разработке Android Studio.

При разработке мобильного приложения была создана архитектура, основанная на паттерне Model-View-ViewModel (MVVM) [13] для разделения логики приложения на три компонента: модель данных (Model), представление (View) и модель представления (ViewModel).

Проект, основанный на паттерне MVVM, содержит следующую структуру:

- директория manifests хранит файл манифеста, который описывает конфигурацию приложения и определяет каждый из компонентов приложения;
- директория database содержит данные приложения и хранит класс базы данных приложения, директории entity и dao;
- директория entity содержит классы, представляющие собой таблицы базы данных приложения;
- директория dao содержит класс, предоставляющий методы, которые приложение может использовать для работы с данными в базе данных;
- директория screen хранит директории, содержащие отдельные экраны приложения. Каждая директория экрана содержит класс фрагмента (View), который отвечает за представление данных,

и класс `ViewModel`, который является связующим звеном между представлением и данными;

- директория `adapter` содержит классы адаптеров, необходимые для представления и работы с набором данных;
- `MainActivity` является основным классом для запуска приложения;
- директория `res` содержит ресурсы, используемые в приложении, разбитые на директории;
- директория `drawable` содержит изображения;
- директория `font` содержит файлы шрифтов;
- директория `layout` содержит XML-файл, определяющие макеты пользовательского интерфейса
- директория `menu` содержит файлы XML, определяющие меню приложения;
- директория `navigation` содержит XML-файл навигационного графа (`navigation graph`), который определяет структуру экранов приложения и связи между ними;
- директория `values` хранит XML-файлы, содержащие простые значения, такие как строки, целые числа и цвета.

Структура проекта при работе в среде разработки Android Studio представлена на рис. 5.3.

При разработке приложения был также использована `Single Activity` архитектура. Это подход к проектированию мобильного приложения, в котором все экраны и функциональность размещаются в одном основном компоненте, который называется `Activity`. Вместо создания новых `Activity` для каждого экрана приложения, используются фрагменты, которые отображают содержимое отдельных экранов и с помощью которых происходит управление навигацией. Это позволяет уменьшить количество кода и упростить управление жизненным циклом приложения.

Листинг кода основной `Activity` представлен в Приложении 2.

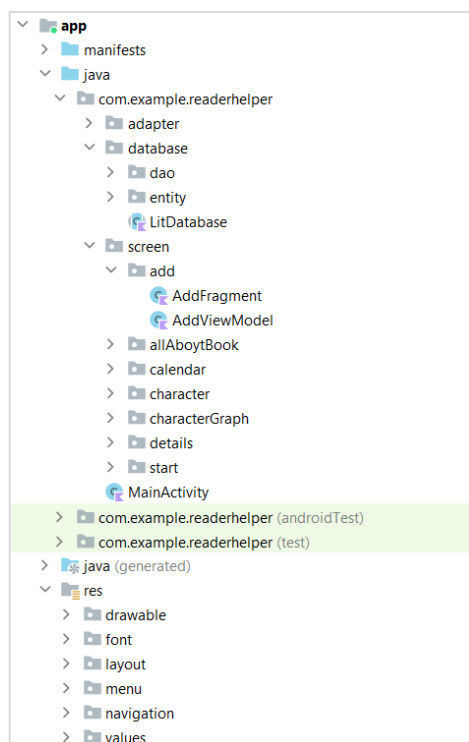


Рис. 5.3. Структура проекта

5.5. Реализация графа связей между персонажами

Для визуализации существования или отсутствия связи между персонажами был построен неориентированный граф. В вершины графа записываются имена персонажей, а ребрами демонстрируется наличие связи между героями. Для решения этой задачи использована библиотека GraphView.

Чтобы добавить персонажей в качестве вершины, необходимо получить список всех героев книги, для которой строится граф. Это происходит с помощью запроса в базу данных. Класс, реализующий методы доступа к базе данных представлен в Приложении 3. После этого на граф добавляются вершины, которые содержат уникальный идентификатор персонажа и имя. Затем делается еще один запрос в базу данных для получения списка наличия связей между персонажами. Связь между персонажами представляет собой пару их идентификаторов. Если между персонажами есть связь, то между соответствующими вершинами графа добавляется ребро.

Листинг кода экрана графа связей представлен в Приложении 4.

6. Интерфейс

Интерфейс [14] приложения состоит из нескольких отдельных экранов:

- главный экран;
- экран добавления/редактирования основной информации о произведении;
- экран календаря;
- экран дополнительной информацией о произведении;
- экран с полной информацией о произведении и его персонажах для чтения.
- экран персонажа;
- экран графа связей между персонажами.

На главном экране расположены кнопка календаря, кнопка поиска, список всех добавленных произведений и кнопка для добавления нового (рис. 6.1.).

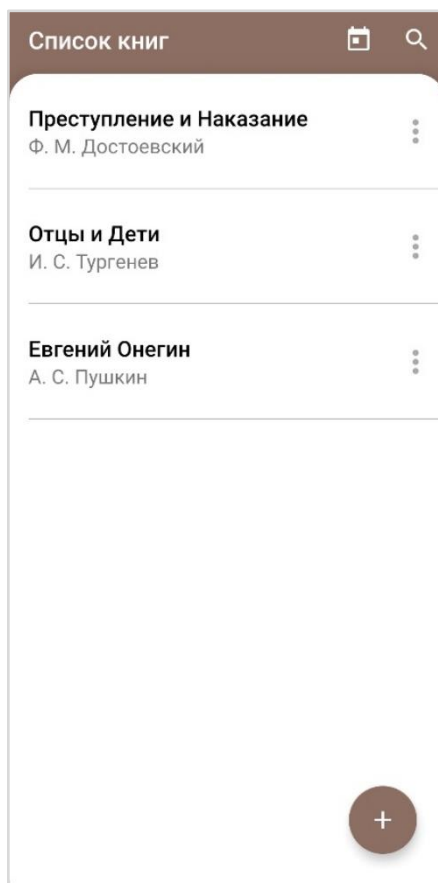


Рис. 6.1. Главный экран

При нажатии на круглую кнопку (floating action button) происходит переход на экран добавления основной информации о новом произведении (рис. 6.2.).

Рис 6.2. Экран добавления нового произведения

Экран добавления содержит в себе кнопку сохранения в нижнем правом углу экрана и четыре основных поля для заполнения:

- название произведения;
- автор;
- год создания;
- дата, до которой запланировано прочитать произведение.

После сохранения новой книги будет вновь открыт главный экран, а в списке отобразится добавленная книга.

Каждый элемент списка произведений имеет меню действий с правой стороны, которое отображается при нажатии (рис. 6.3.).

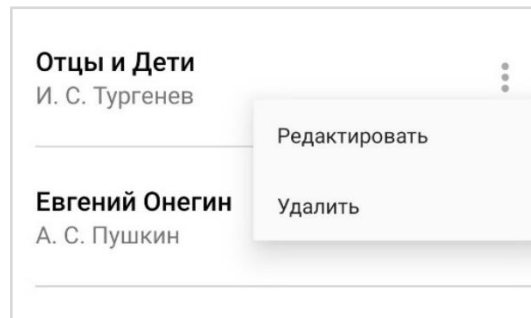


Рис 6.3. Меню элемента списка книг

При нажатии на пункт «Редактировать» происходит переход на экран, где можно изменить основную информацию о книге (рис 6.4.).

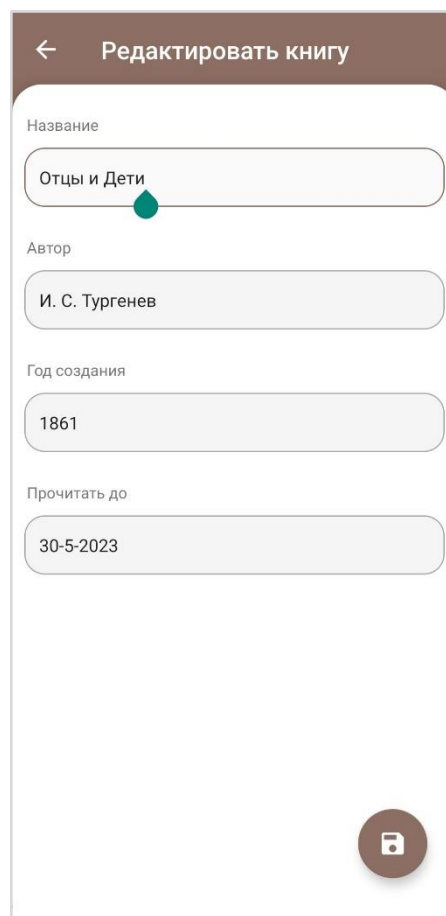


Рис. 6.4. Экран редактирования

При нажатии на пункт «Удалить» появляется всплывающее уведомление с предупреждением об удалении книги и всей информацией о ней (рис. 6.5.). После подтверждения удаления книга пропадает из списка на главном экране.

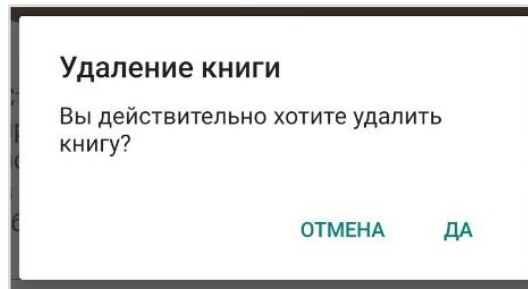


Рис. 6.5. Предупреждение об удалении

Для того чтобы осуществить поиск достаточно нажать на иконку поиска в верхнем углу экрана, чтобы появилось поле для ввода текста (рис. 6.6.). Поиск осуществляется по названию или автору.

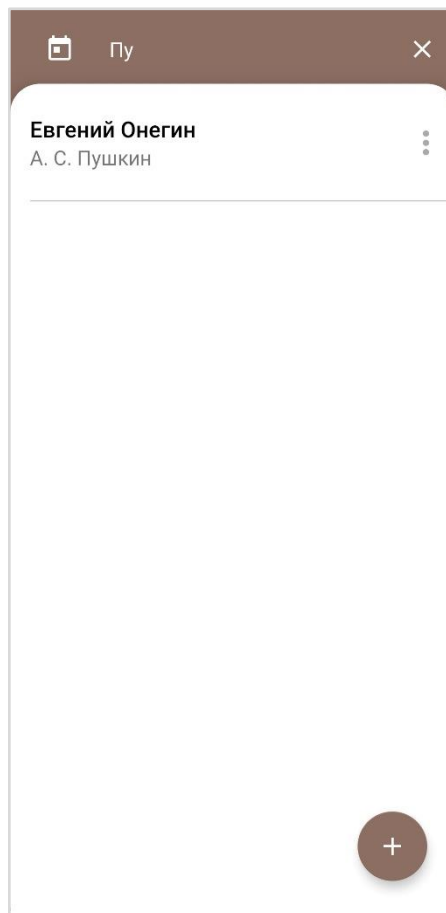


Рис. 6.6 Поиск книги

В главном экране, если нажать на иконку календаря, будет совершен переход на другой экран, который представлен на рис. 6.7.

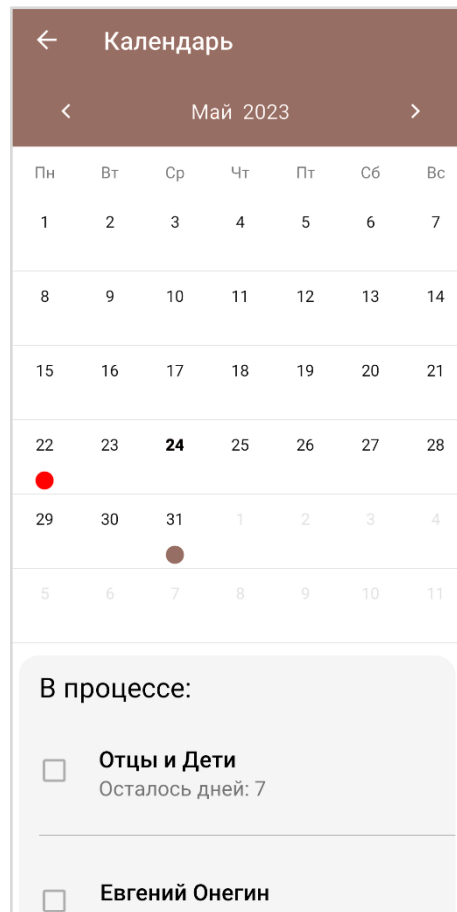


Рис. 6.7. Экран календаря

Точками в календаре отмечены те произведения, который еще находятся в процессе чтения. Красная точка означает, что срок чтения уже просрочен. Внизу экрана находятся два списка: «в процессе» и «прочитанные» (рис. 6.8.).

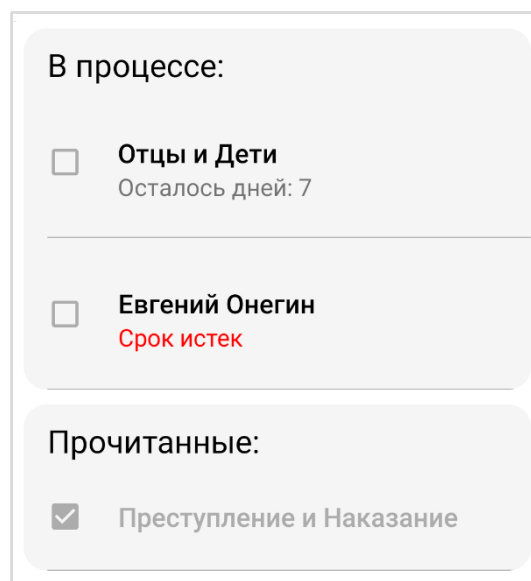


Рис 6.8. Списки «в процессе» и «прочитанные»

Список «в процессе» отображает те произведения, которые пользователь еще не отметил, как прочитанные. Элемент списка состоит из названия книги и количества дней до конца установленного срока прочтения. Если срок истек, это будет указано красным цветом текста. Для того, чтобы отметить произведение как прочитанное, достаточно поставить галочку в левой части элемента списка, и тогда произведение переместится в список «прочитанные».

Для того, чтобы добавить или просмотреть дополнительную информацию о книге, необходимо нажать на книгу из списка, находящегося на главном экране. После нажатия открывается экран дополнительной информации (рис. 6.9.).

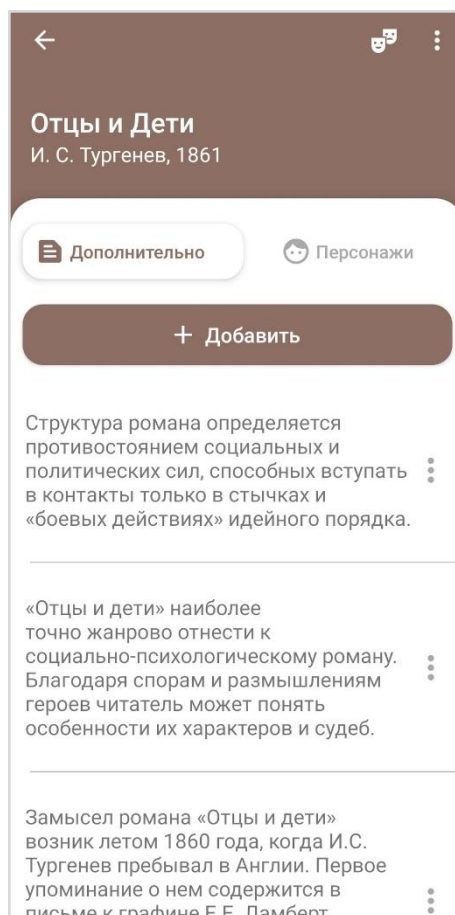


Рис 6.9. Экран дополнительной информации

С помощью этого экрана можно добавить дополнительную информацию о книге или персонажа. Для этого необходимо выбрать одну из двух радиокнопок «Дополнительно» и «Персонажи» и нажать кнопку

добавить. После нажатия откроется всплывающее окно, в которое можно внести текст (рис 6.10.).

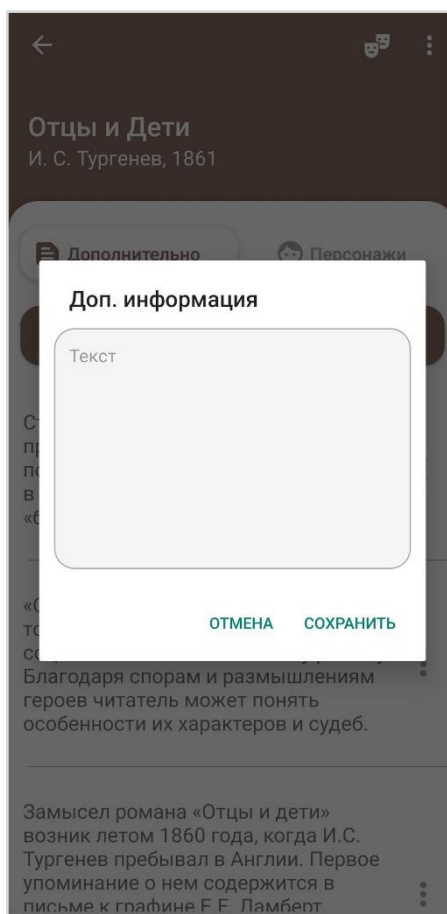


Рис. 6.10. Всплывающее окно

При сохранении введенный текст добавляется как новый элемент списка, который содержит сам текст и меню действий справа, с помощью которого можно редактировать или удалить информацию (рис. 6.11.).

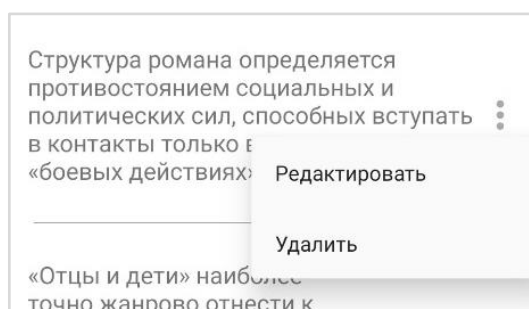


Рис. 6.11. Меню элемента списка «Дополнительно»

В верхней части экрана дополнительной информации расположена кнопка для перехода на экран графа связей между персонажами и меню действий (рис 6.12.).

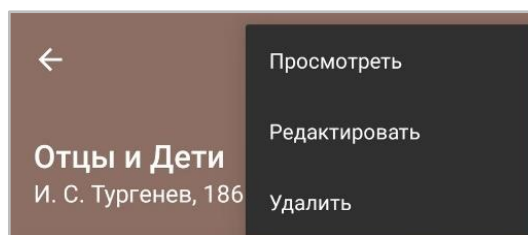


Рис 6.12. Меню действий

При нажатии на пункт «Просмотреть» открывается новый экран, который содержит в себе всю информацию о произведении и его персонажах, доступной для чтения на одном экране (рис 6.13.).



Рис. 6.13. Экран с полной информацией о произведении

Чтобы открыть полную информацию о персонаже (рис. 6.14), достаточно выбрать нужного персонажа из списка.



Рис. 6.14. Экран персонажа

Экран содержит пять радиокнопок, отвечающие за различную информацию о персонаже, которую можно добавлять, редактировать и удалять:

- биография;
- портрет;
- отношения;
- цитаты;
- другое.

Особого внимания заслуживает информация об отношениях персонажей, которая представлена на рис. 6.15.

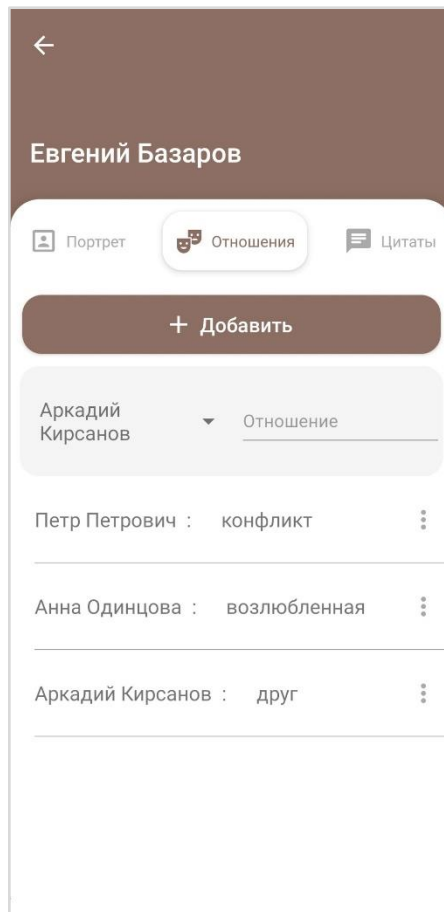


Рис. 6.15. Радиокнопка «отношения»

Для того чтобы добавить взаимоотношения между персонажами, необходимо выбрать другого персонажа из выпадающего списка на сером фоне (рис 6.16), ввести описание отношений и нажать кнопку добавить.

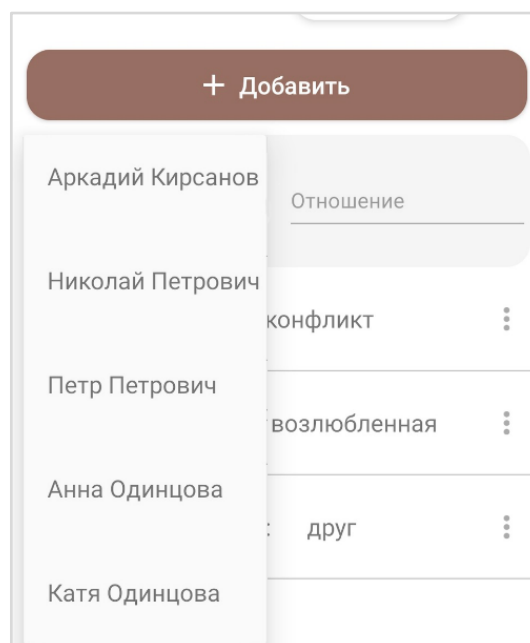


Рис. 6.16. Выпадающий список персонажей

На основе данных, которые заполняются в разделе отношений, будет построен неориентированный граф между всеми персонажами книги, который наглядно демонстрирует связь героев между собой (рис. 6.17).



Рис. 6.17. Экран графа связей между персонажами

7. Тестирование

Для проверки корректности работы приложения было осуществлено ручное тестирование.

7.1. Тест 1. Добавление произведения

Цель теста: проверить корректность добавления нового произведения.

Порядок выполнения.

1. Перейти в главное меню.
2. Нажать на круглую кнопку «Добавить произведение».
3. Заполнить поля «Название», «Автор», «Год», «Прочитать до» данными в новом экране «Добавить книгу».
4. Сохранить результат, нажав на круглую кнопку «Сохранить».

Результат: произойдет переход на главный экран, в котором отображено добавленное произведение и информация о нём, соответствующая тексту в заполненных полях.

7.2. Тест 2. Редактирование существующего произведения.

Цель теста: проверить возможность редактирования информации о произведении.

Порядок выполнения.

1. Нажать на меню произведения в правой части элемента списка.
2. Выбрать в меню пункт «Редактировать».
3. Изменить поля информации о произведении в новом экране.
4. Нажать на кнопку «Сохранить».

Результат: основная информация о произведении будет обновлена в соответствии с введенными значениями.

7.3 Тест 3. Удаление произведения

Цель теста: проверить корректность удаления произведения.

Порядок выполнения.

1. Нажать на меню произведения в правой части элемента списка.
2. Выбрать в меню пункт «Удалить».
3. Подтвердить удаление в всплывающем окне.

Результат: произведение удалено из списка произведений и базы данных.

7.4 Тест 4. Добавление события в календарь.

Цель теста: проверить корректность добавления события о дате прочтения произведения в календарь.

Порядок выполнения.

1. Добавить новое произведение, заполнив поле «прочитать до».
2. Зайти в экран календаря.

Результат: точкой в календаре отображается дата прочтения произведения. Произведение добавлено в список «В процессе», указано оставшиеся количество дней до конца срока прочтения или надпись «Срок истек», если дата меньше сегодняшней.

7.5 Тест 5. Добавление дополнительной информации о произведении

Цель теста: проверить корректность добавления дополнительной информации.

Порядок выполнения.

1. Перейти на экран дополнительной информацией о произведении, выбрав в главном меню произведение из списка.
2. Нажать кнопку «Добавить».
3. Заполнить текстовое поле в всплывающем окне.
4. Сохранить введённый текст.

Результат: в списке дополнительной информации появляется новый элемент, который содержит введённый текст.

7.6 Тест 6. Добавление персонажа

Цель теста: проверить корректность добавления персонажа.

Порядок выполнения.

1. Перейти на экран дополнительной информацией о произведении, выбрав в главном меню произведение из списка.
2. Нажать радиокнопку «Персонажи».
3. Нажать кнопку «Добавить».
4. Заполнить текстовое поле в всплывающем окне.
5. Сохранить введённый текст.

Результат: в списке персонажей появится новый элемент, который содержит введённый текст.

7.7 Тест 7. Добавление данных о персонаже.

Цель теста: проверить корректность добавления информации о персонаже в графу «Биография»

Порядок выполнения.

1. Выбрать персонажа из списка персонажей.
2. Нажать радиокнопку «Биография».
3. Нажать кнопку «Редактировать».
4. Заполнить текстовое поле.
5. Нажать кнопку «Сохранить».

Результат: добавлен текст в поле «Биография» персонажа.

7.8 Тест 8. Добавление данных об отношениях персонажа.

Цель теста: проверить корректность добавления информации о персонаже в графу «Отношения»

Порядок выполнения.

1. Выбрать персонажа из списка персонажей.
2. Нажать радиокнопку «Отношения».
3. Выбрать другого персонажа из выпадающего списка персонажей.
4. Заполнить текстовое поле об отношениях между персонажами.
5. Нажать кнопку «Сохранить».

Результат: в списке отношений появляется новый элемент, который содержит имя выбранного персонажа и введённый текст.

7.9. Тест 9. Добавление цитаты

Цель теста: проверить корректность добавления цитаты.

Порядок выполнения.

1. Выбрать персонажа из списка персонажей.
2. Нажать радиокнопку «Цитаты».
3. Нажать кнопку «Добавить».
4. Заполнить текстовое поле в всплывающем окне.
5. Сохранить введённый текст.

Результат: в списке цитат появляется новый элемент, который содержит введённый текст.

7.10 Тест 10. Проверка графа взаимоотношений персонажей.

Цель теста: проверить корректность отображение графа взаимоотношений персонажей

Порядок выполнения.

1. Добавить данные об отношениях между персонажами.
2. Перейти на экран графа взаимоотношений.

Результат: отображен граф взаимоотношений персонажами. В вершинах указаны имена персонажей, наличие отношений между персонажами показаны линиями. Если у персонажа нет отношений с другими персонажами, он также обозначен на графе.

Заключение

В результате выполнения работы было разработано мобильное приложение для операционной системы Android, позволяющее записывать и сохранять заметки о художественных произведениях и их персонажах.

Были решены следующие задачи:

- проанализированы доступные существующие решения;
- выбраны средства реализации;
- спроектирована диаграмма сценариев;
- разработана база данных приложения;
- разработан интерфейс;
- реализовано приложение, решающее поставленную задачу;
- произведено тестирование приложения.

Приложение «Помощник читателя» будет полезно ученикам средней и старшей школы и студентам, изучающим предмет литература, а также любителям чтения художественных произведения, которые хотят вести собственные записи о прочитанной книге.

Материалы работы были опубликованы в научно-техническом журнале «Информационные технологии в строительных, социальных и экономических системах» [15].

Список литературы

1. Колисниченко Д. Н. Программирование для Android / Самоучитель, СПб.: БХВ-Петербург, 2015. — 303 с.
2. Дейтел П. Android для разработчиков. 3-е изд. / П. Дейтел, Х. Дейтел, А. Уолд ; пер. с англ. Е. Матвеев. — СПб.: Питер, 2016. — 512 с.
3. Читательский дневник BookDiary – URL: <https://bookdiary.ru/> (Дата обращения: 29.05.2023)
4. Документация Android Developers. – URL: <https://developer.android.com/docs> (Дата обращения: 29.05.2023)
5. Документация Kotlin – URL: <https://kotlinlang.org/docs/home.html> (Дата обращения: 29.05.2023)
6. Документация SQLite – URL: <https://www.sqlite.org/docs.html> (Дата обращения: 29.05.2023)
7. Матвеева М. В. Унифицированный язык моделирования UML / М. В. Матвеева, А. Ш. Исламов, Учебно-методическое пособие, Изд-во ВГУ, 2016 г. — 52 с.
8. Документация Room – URL: <https://developer.android.com/training/data-storage/room> (Дата обращения: 29.05.2023)
9. Документация GraphView – URL: <https://www.mobintouch.com/ui/graphview/> (Дата обращения: 29.05.2023)
10. Документация Material CalendarView – URL: <https://applandeo.com/blog/materialcalendarview-customizable-calendar-widget-for-android-updated-2022/> (Дата обращения: 29.05.2023)
11. Компоненты архитектуры Android – URL: <https://developer.android.com/jetpack> (Дата обращения: 29.05.2023)
12. Android. Программирование для профессионалов. 2-е изд. / Б. Харди, Б. Филлипс, К. Стюарт, К. Марсикано ; пер. с англ. Е. Матвеев. — СПб.: Питер, 2016. — 640 с.

13. Паттерны для новичков: MVC vs MVP vs MVVM – URL: <https://habr.com/ru/articles/215605/> (Дата обращения: 29.05.2023)
14. Система дизайна интерфейсов программного обеспечения и приложений – URL: <https://m3.material.io/> (Дата обращения: 29.05.2023)
15. Собакарь Ю. Н. Разработка мобильного приложения для помощи читателю под операционную систему Android / Ю. Н. Собакарь // Информационные технологии в строительных, социальных и экономических системах. — 2022. — №4 (30). — С. 141–144

Приложение 1. Класс базы данных и таблиц

Класс базы данных.

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import com.example.readerhelper.database.dao.BookDao
import com.example.readerhelper.database.entity.*

@Database(entities = [Book::class, OtherInfBook::class, BookCharacter::class,
BookCharacterCrossRef::class,
    Quote::class, OtherInfCharacter::class, ReadingBook::class], version = 1)
abstract class LitDatabase: RoomDatabase() {

    abstract fun getBookDao(): BookDao

    companion object {
        // Singleton prevents multiple
        // instances of database opening at the
        // same time.
        @Volatile
        private var INSTANCE: LitDatabase? = null

        fun getDatabase(context: Context): LitDatabase {
            // if the INSTANCE is not null, then return it,
            // if it is, then create the database
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    LitDatabase::class.java,
                    "lit_database"
                ).build()
                INSTANCE = instance
                // return instance
                instance
            }
        }
    }
}
```

Классы таблиц базы данных.

```
//сущность Книга
@Entity
data class Book(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "book_id")
    val bookId: Int = 0,

    @ColumnInfo(name = "book_name")
    val bookName:String?,

    @ColumnInfo(name = "book_author")
    val bookAuthor:String?,

    @ColumnInfo(name = "year")
    val year: String?,

    @ColumnInfo(name = "calendar_date")
    val calendarDate: String?,

    var isRead: Boolean,
```

```

)
//сущность Дополнительно_книга
@Entity
data class OtherInfBook(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "otherInf_id")
    val bookId: Int = 0,

    @ColumnInfo(name = "information")
    val information:String?,

    @ColumnInfo(name = "parent_book_id")
    val parentBookIdOther: Int
)
//сущность Персонаж
@Entity
data class BookCharacter(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "character_id")
    val characterId: Int = 0,

    @ColumnInfo(name = "character_name")
    var characterName:String?,

    @ColumnInfo(name = "parent_book_id_to_character")
    val parentBookIdChar: Int,

    @ColumnInfo(name = "biography")
    var biography: String? = "",

    @ColumnInfo(name = "portrait")
    var portrait: String? = ""
)
//сущность Взаимоотношения_персонаж
@Entity
data class BookCharacterCrossRef(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "relation_id")
    val relationId: Int = 0,

    @ColumnInfo(name = "character_id")
    var characterId1: Int,

    @ColumnInfo(name = "character_name1")
    var characterName1: String?,

    @ColumnInfo(name = "character_id2")
    var characterId2: Int,

    @ColumnInfo(name = "character_name2")
    var characterName2: String?,

    var descriptionRelationship:String?
)
//сущность Другое_персонаж
@Entity
data class OtherInfCharacter(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "other_inf_char_id")
    val otherInfCharId: Int = 0,

    @ColumnInfo(name = "text_other_inf_char")
    val text:String?,

```

```

        @ColumnInfo(name = "parent_character_id")
        val parentCharacter: Int
    )
    //сущность Цитата_Персонаж
    @Entity
    data class Quote(
        @PrimaryKey(autoGenerate = true)
        @ColumnInfo(name = "quote_id")
        val quoteId: Int = 0,

        @ColumnInfo(name = "text_quote")
        val text:String?,

        @ColumnInfo(name = "parent_character_id")
        val parentCharacter: Int
    )

```

Приложение 2. Листинг MainActivity.kt

```
import android.os.Bundle
import android.view.MenuItem
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.NavController
import androidx.navigation.findNavController
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.onNavDestinationSelected
import com.example.readerhelper.adapter.APP
import com.example.readerhelper.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    lateinit var binding: ActivityMainBinding
    lateinit var navController: NavController
    lateinit var navHostFragment: NavHostFragment

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        APP = this
        navHostFragment =
supportFragmentManager.findFragmentById(R.id.nav_fragment) as NavHostFragment
        navController = navHostFragment.navController
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        val navController = findNavController(R.id.nav_fragment)

        return item.onNavDestinationSelected(navController) ||
super.onOptionsItemSelected(item)
    }
}
```

Приложение 3. Листинг BookDao.kt

```
import androidx.lifecycle.LiveData
import androidx.room.*
import com.example.readerhelper.database.entity.*

@Dao
interface BookDao {

    //получить все книги
    @Query("SELECT * FROM Book")
    fun getAllBooks(): LiveData<List<Book>>

    //получить книгу по id
    @Query("SELECT * FROM Book WHERE book_id=:bookId")
    fun getBook(bookId: Int): Book

    //добавить книгу
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insert(book: Book)
    //получить список книг с датой прочтения
    @Query("SELECT * FROM ReadingBook")
    fun getReadingBook(): LiveData<List<ReadingBook>>

    //получить прочитанные книги
    @Query("SELECT * FROM ReadingBook where isRead=:ok")
    fun getIsReadingBook(ok: Boolean): LiveData<List<ReadingBook>>
    //добавить книгу с датой прочтения
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insertReadingBook(readingBook: ReadingBook)

    //обновить основную информацию о книге
    @Update
    fun updateBook(book: Book)

    //удалить книгу
    @Delete
    fun delete(book: Book)

    //получить список доп. информации о книге
    @Transaction
    @Query("SELECT * FROM Book where book_id =:bookId")
    fun getBookOtherInformation (bookId: Int): LiveData<BookAndInf>

    //добавить доп.информацию о книге
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insertOtherInfo(otherInfBook: OtherInfBook)

    //обновить доп.информацию о книге
    @Update
    fun updateOtherInfo(otherInfBook: OtherInfBook)

    //добавить персонажа
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insertCharacter(character: BookCharacter)

    //получить список персонажей
    @Transaction
    @Query("SELECT * FROM Book where book_id =:bookId")
    fun getCharacters (bookId: Int): LiveData<BookAndCharacter>

    //добавить отношения между персонажами
```



```

@Insert(onConflict = OnConflictStrategy.IGNORE)
fun insertBookCharacterCrossRef(characterCrossRef: BookCharacterCrossRef)

//получить список взаимоотношений определенного персонажа
@Transaction
@Query("SELECT * FROM BookCharacter where character_id =:characterId")
fun getCharacterRelations (characterId: Int):
LiveData<CharacterWithOther>

//обновить персонажа
@Update
fun updateCharacter(bookCharacter: BookCharacter)

//получить список доп. информации о персонаже
@Transaction
@Query("SELECT * FROM BookCharacter where character_id =:characterId")
fun getCharacterOtherInf (characterId: Int):
LiveData<CharacterAndOtherInf>

//получить список цитат персонажа
@Transaction
@Query("SELECT * FROM BookCharacter where character_id =:characterId")
fun getCharacterQuotes (characterId: Int): LiveData<CharacterAndQuote>

//добавить цитату персонажу
@Insert(onConflict = OnConflictStrategy.IGNORE)
fun insertCharacterQuote(quote: Quote)

//обновить цитату
@Update
fun updateQuote(quote: Quote)

//обновить доп. инф-цию о персонаже
@Update
fun updateCharacterOtherInf(characterOtherInf: OtherInfCharacter)

//добавить доп. инф-цию о персонаже
@Insert(onConflict = OnConflictStrategy.IGNORE)
fun insertCharacterOtherInf(characterOtherInf: OtherInfCharacter)

//получить список всех взаимоотношений по книге
@Transaction
@Query("SELECT CR.* FROM BookCharacter BC JOIN BookCharacterCrossRef CR
ON " +
        "BC.character_id = CR.character_id where
BC.parent_book_id_to_character =:bookId")
fun getCharacterRelationFromBook(bookId: Int):
LiveData<List<BookCharacterCrossRef>>

//удалить доп. информацию о книге
@Delete
fun deleteOtherInfBook(otherInfBook: OtherInfBook)

//удалить доп. информацию о персонаже
@Delete
fun deleteOtherInfChar(otherInfCharacter: OtherInfCharacter)

//удалить цитату
@Delete
fun deleteOtherInfQuote(quote: Quote)

//удалить персонажа
@Delete

```

```

fun deleteBookCharacter(character: BookCharacter)

//удалить взаимоотношения персонажей
@Delete
fun deleteRelation(bookCharacterCrossRef: BookCharacterCrossRef)

//удалить книгу из календаря
@Delete
fun deleteReadingBook(readingBook: ReadingBook)

//обновить книгу из календаря
@Update
fun updateReadingBook(readingBook: ReadingBook)
}
//поиск книге по названию и автору
@Query("SELECT * FROM Book WHERE book_name LIKE:searchQuery OR
book_author LIKE :searchQuery ")
fun searchBook(searchQuery:String): Flow<List<Book>>

```

Приложение 4. Экран графа связей между персонажами

Макет вершины графа.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="20dp"
    android:background="@drawable/node"
    android:padding="10dp"
    >
    <TextView
        android:textSize="48sp"
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto_medium"
        android:textColor="@android:color/white"
        tools:text="Node text here" />
</LinearLayout>
```

Макет экрана графа связей персонажей.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".screen.characterGraph.CharacterGraphFragment">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar4"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:textAlignment="viewStart"
        android:theme="@style/ThemeOverlay.AppCompat"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.otaliastudios.zoom.ZoomLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:hasClickableChildren="true"
        android:scrollbars="vertical|horizontal"
        >

        <androidx.recyclerview.widget.RecyclerView
            android:layout_marginTop="?attr/actionBarSize"
            tools:listitem="@layout/node"
            app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
            android:id="@+id/recycler"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </com.otaliastudios.zoom.ZoomLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Класс представления.

```

import android.os.Bundle
import android.view.*
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.fragment.app.Fragment
import androidx.lifecycle.LiveData
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.fragment.findNavController
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.setupWithNavController
import androidx.recyclerview.widget.RecyclerView
import com.example.readerhelper.R
import com.example.readerhelper.database.entity.Book
import com.example.readerhelper.database.entity.BookAndCharacter
import com.example.readerhelper.database.entity.BookCharacterCrossRef
import com.example.readerhelper.databinding.FragmentCharacterGraphBinding
import com.google.android.material.snackbar.Snackbar
import dev.bandb.graphview.AbstractGraphAdapter
import dev.bandb.graphview.decoration.edge.StraightEdgeDecoration
import dev.bandb.graphview.graph.Graph
import dev.bandb.graphview.graph.Node
import dev.bandb.graphview.layouts.GraphLayoutManager
import dev.bandb.graphview.layouts.energy.FruchtermanReingoldLayoutManager
import java.util.*

class CharacterGraphFragment : Fragment() {

    private lateinit var viewModel: CharacterGraphViewModel
    private lateinit var binding: FragmentCharacterGraphBinding
    private lateinit var currentBook: Book
    private lateinit var recyclerView: RecyclerView
    private lateinit var adapter: AbstractGraphAdapter<NodeViewHolder>
    private var currentNode: Node? = null
    // private lateinit var graph: Graph

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentCharacterGraphBinding.inflate(layoutInflater,
container, false)
        currentBook = arguments?.getSerializable("book") as Book
        (activity as AppCompatActivity).setSupportActionBar(binding.toolbar4)
        recyclerView = binding.recycler
        val graphLayoutManager: GraphLayoutManager =
FruchtermanReingoldLayoutManager(activity as AppCompatActivity)
        recyclerView.layoutManager = graphLayoutManager
        recyclerView.addItemDecoration(StraightEdgeDecoration())

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val navController = findNavController()
        val appBarConfig = AppBarConfiguration(navController.graph)
        val toolbar = view.findViewById<Toolbar>(R.id.toolbar4)

        toolbar.setupWithNavController(navController, appBarConfig)
        init()
    }
}

```

```

        private fun init() {
            viewModel =
ViewModelProvider(this)[CharacterGraphViewModel::class.java]
            val graph = createGraph()
            recyclerView = binding.recycler
            recyclerView.layoutManager =
FruchtermanReingoldLayoutManager(requireActivity(), 1000)
            setLayoutManager()
            setEdgeDecoration()
            setupGraphView(graph)
        }

        private fun setLayoutManager() {
            recyclerView.layoutManager =
FruchtermanReingoldLayoutManager(requireActivity(), 1000)
        }

        private fun setEdgeDecoration() {
            recyclerView.addItemDecoration(StraightEdgeDecoration())
        }

        private fun createGraph(): Graph {
            val graph = Graph()

            val list2: LiveData<BookAndCharacter> =
viewModel.getCharacter(currentBook.bookId)

            list2.observe(viewLifecycleOwner) { characters ->
viewModel.addAllBooksToGraph(graph, characters.characterList)

            val list: LiveData<List<BookCharacterCrossRef>> =
viewModel.getCharacterRelationFromBook(currentBook.bookId)

            list.observe(viewLifecycleOwner) { listRel ->
viewModel.addEdgeToGraph(graph, listRel)
            }
        }
        return graph
    }

    private fun setupGraphView(graph: Graph) {
        adapter = object : AbstractGraphAdapter<NodeViewHolder>() {
            override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
NodeViewHolder {
                val view = LayoutInflater.from(parent.context)
                    .inflate(R.layout.node, parent, false)
                return NodeViewHolder(view)
            }

            override fun onBindViewHolder(holder: NodeViewHolder, position: Int)
{
                holder.textView.text =
Objects.requireNonNull(getNodeData(position)).toString()
            }

            }.apply {
                this.submitGraph(graph)
                recyclerView.adapter = this
            }
    }

    private inner class NodeViewHolder internal constructor(itemView: View) :

```

```

RecyclerView.ViewHolder(itemView) {
    var textView: TextView = itemView.findViewById(R.id.textView)
}
}

```

Класс ViewModel.

```

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import com.example.readerhelper.adapter.REPOSITORY
import com.example.readerhelper.database.entity.BookAndCharacter
import com.example.readerhelper.database.entity.BookCharacter
import com.example.readerhelper.database.entity.BookCharacterCrossRef
import dev.banb.graphview.graph.Graph
import dev.banb.graphview.graph.Node

class CharacterGraphViewModel (application: Application):
    AndroidViewModel(application)    {

        fun getCharacterRelationFromBook(bookId: Int):
        LiveData<List<BookCharacterCrossRef>> {
            return REPOSITORY.getCharacterRelationFromBook(bookId)
        }

        fun getCharacter(bookId: Int) : LiveData<BookAndCharacter> {
            return REPOSITORY.getCharacters(bookId)
        }

        fun addEdgeToGraph(graph: Graph, list: List<BookCharacterCrossRef>) {
            for(k in list.indices) {
                for (i in graph.nodes.indices) {
                    if(list[k].characterName1!!.equals(graph.nodes[i].data))
                        for (j in graph.nodes.indices)
                            if(list[k].characterName2!!.equals(graph.nodes[j].data))
                                graph.addEdge(graph.nodes[i], graph.nodes[j])
                }
            }
        }

        fun addAllBooksToGraph(graph: Graph, list: List<BookCharacter>){
            for (i in list.indices) {
                graph.addNode(Node(list[i].characterName!!))
            }
        }
    }
}

```