ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО

Приглашенный преподаватель департамента программной инженерии факультета компьютерных наук

_____ А. Н. Степанов

«2» апреля 2023 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», кандидат технических наук

_____ В. В. Шилов

«2» апреля 2023 г.

Подп. и дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. и дата	
Инв. № подл.	

КЛИЕНТСКАЯ ЧАСТЬ МОБИЛЬНОГО ПРИЛОЖЕНИЯ «SHELFIE» ДЛЯ ТРЕКИНГА И РЕКОМЕНДАЦИИ КНИГ

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.02.02-01 ПЗ 01-1-ЛУ

Исполнитель:

Студент группы БПИ-206

_____ / А. С. Сибирцева /

«2» апреля 2023 г.

Москва 2023

УТВЕРЖДЕН

RU.17701729.02.02-01 ПЗ 01-1-ЛУ

КЛИЕНТСКАЯ ЧАСТЬ МОБИЛЬНОГО ПРИЛОЖЕНИЯ «SHELFIE» ДЛЯ ТРЕКИНГА И РЕКОМЕНДАЦИИ КНИГ

Пояснительная записка

RU.17701729.02.02-01 ПЗ 01-1

Листов 97

Подп. и дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. и дата	
Инв. № подл.	

Москва 2023

АННОТАЦИЯ

Пояснительная записка на «Клиентская часть мобильного приложения “Shelfie” – для трекинга и рекомендации книг» содержит следующие разделы: «Глоссарий», «Введение», «Назначение разработки», «Технические характеристики», «Технико-экономические показатели».

В разделе «Глоссарий» содержатся определения терминов и понятий, используемых в пояснительной записке.

В разделе «Введение» указано наименование и документ, на основе которого ведется разработка.

В разделе «Назначение разработки» указано функциональное и эксплуатационное назначение программного продукта, а также краткая характеристика области применения приложения.

Раздел «Технические характеристики» содержит следующие подразделы: «Постановка задачи на разработку программы», «Описание архитектуры», «Описание алгоритмов работы», «Описание функционирования программы», где разбирается состав выполняемых функций и описание интерфейса и его работы, «Описание и обоснования выбора метода организации входных и выходных данных», «Описание выбора состава технических и программных средств».

Раздел «Технико-экономические показатели» содержит ориентировочную экономическую эффективность, предполагаемую годовую потребность, экономические преимущества разработки приложения.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];
2) ГОСТ 19.102-77 Стадии разработки [2];
3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];
4) ГОСТ 19.104-78 Основные надписи [4];
5) ГОСТ 19.105-78 Общие требования к программным документам [5];
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];

7) ГОСТ 19.201-78 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к данной Пояснительной записке оформляются согласно ГОСТ 19.603-78 [8].

СОДЕРЖАНИЕ

ГЛОССАРИЙ

1) *Пользователь* – человек, который использует приложение «Shelfie».
2) *Подборка* (также, *коллекция*) – список книг, предлагаемых пользователю, объединенных по какому-то признаку.
3) *Сборник* – набор книг, создаваемый пользователем.
4) *Список* – формат отображения книг, когда в одной строке показывается только один элемент списка, а информация представлена в виде обложки произведения и краткой информацией справа от нее.
5) *ISBN* – уникальный номер книги.
6) *Время чтения* – среднее время, которое потратили пользователи на чтение книги.
7) *Чек-лист* – перечень пунктов, напротив которых ставится галочка, если пользователь хочет его выбрать.
8) *Скроллбар* – элемент графического интерфейса, представляющий из себя полосу прокрутки.
9) *Рейтинг книги* – средняя оценка книги, полученная на основе оценок пользователей приложения (число от нуля до десяти).
10) *Рецензия* (так же *отзыв*) – текст, написанный пользователем в качестве его оценки на прочитанное произведение.
11) *Нижняя панель навигации* – элемент графического интерфейса, расположенный в нижней части экрана, который используется для навигации между различными страницами в приложении.
12) *Баннер* – длинное изображение в профиле пользователя, которое выступает своего рода обложкой.
13) *Статус книги* – состояние, в котором находится книга. Может принимать пять значений: «Буду читать», «Читаю», «Перестал», «Прочитал», «Не читаю».
14) *Библиотека пользователя* – набор из всех книг пользователя, которые имеют статус: «Буду читать», «Читаю», «Перестал», «Прочитал».
15) *API (Application Programming Interface)* – описание способов (набор классов, методов и т. п.), которыми одна компьютерная программа (в данном случае, клиентская часть приложения) может взаимодействовать с другой (в данном случае, с сервером).

16) *Snackbar* – элемент, который позволяет выводить всплывающие сообщения, растягивая их по ширине экрана.

17) *Виджет* – элемент графического интерфейса, выполняющий како-то действие.

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование программы – «Клиентская часть мобильного приложения «Shelfie» для трекинга и рекомендации книг».

Наименование программы на английском языке – «Client Side of the "Shelfie" – Mobile Application for Recommending and Tracking Books».

Наименование программы для пользователя – «Shelfie».

1.2. Документы, на основании которых ведётся разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ РАЗРАБОТКИ

2.1. Функциональное назначение

Мобильное приложение «Shelfie» представляет собой сервис для поиска и трекинга книг. Сервис реализует в себе функциональность, позволяющую пользователям искать интересующие их книги, а также отслеживать свой прогресс в чтении: сколько книг было прочитано и какие планируется к прочтению.

Каждый пользователь может просматривать рецензии других людей на интересующие его произведения, а также писать собственные. Помимо этого, в приложении присутствует возможность сохранять понравившиеся цитаты из прочитанных книг. Благодаря тому, что приложение сохраняет историю чтения литературных произведений, пользователь с легкостью сможет определить, какие книги он еще не прочитал.

Клиентская часть мобильного приложения предназначена для взаимодействия пользователя с серверной частью и данными посредством графического интерфейса.

2.2. Эксплуатационное назначение

Мобильное приложение «Shelfie» будет применяться в сфере хобби и интересов. Так как вся информация о прочитанных произведения будет агрегирована в одном приложении, читателям не придется постоянно воспоминать, какие книги они еще не успели прочитать или на какой книге серии они остановились, вся необходимая им информация будет храниться в одном приложении. Это улучшит пользовательский опыт любителей литературы в отслеживании прочитанных произведений. Пользователь сможет находить подборки по своим любимым жанрам, что значительно упростит поиск новых книг для чтения.

Целевая аудитория приложения – люди, которые увлекаются чтением и хотят организовать свою виртуальную библиотеку, а также те, кто хочет систематизировать процесс чтения

2.3. Краткая характеристика области применения

Целевая аудитория мобильного приложения «Shelfie» – любители литературы, а также люди, которые хотят сделать чтение своей привычкой.

Пользователи смогут подобрать книгу по интересующему их жанру, автору и т. д., оставлять рецензии и отмечать запомнившиеся цитаты, а также создавать сборники для тематической организации книг.

Помимо основной информации о произведении читателям будет доступна статистика, собранная по другим пользователям, например, время чтения, рейтинг, количество людей, которые прочитали книгу, оставили рецензию или цитату. Это поможет пользователю оценить, насколько книга подходит под его потребности

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Разрабатываемая программа должна соответствовать описанным в техническом задании функциональным характеристикам, требованиям к интерфейсу и надежности, а также временным характеристикам.

3.2. Описание архитектуры

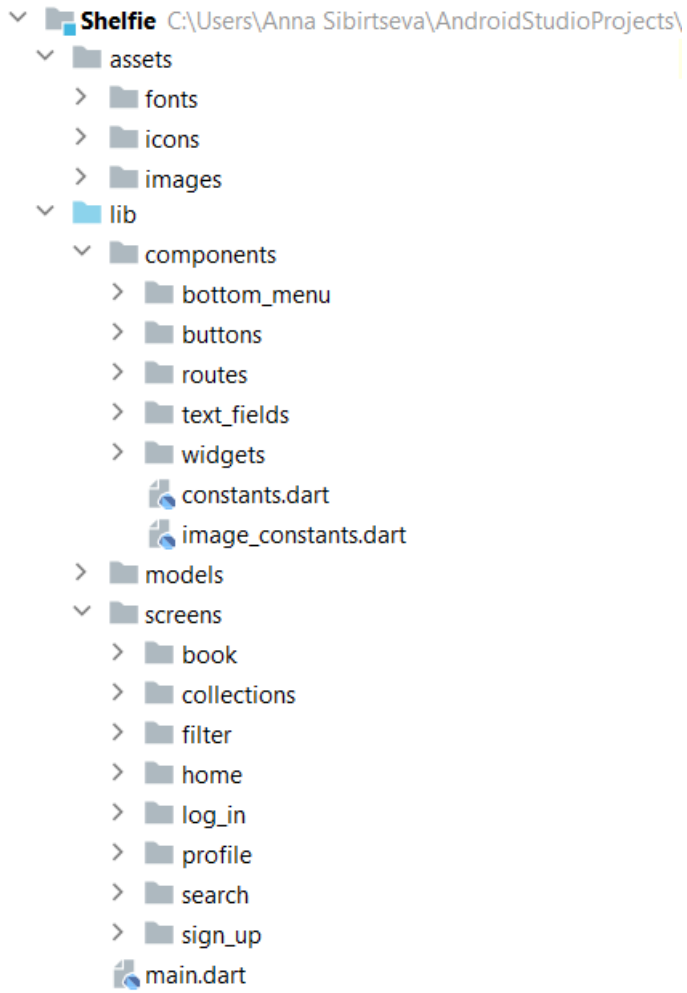

Рисунок 1. Общая архитектура проекта

На рисунке 1 представлена общая архитектура проекта. В папке “assets” хранятся различные ресурсы для интерфейса приложения, как, например:

1. В папке “fonts” содержаться основные шрифты приложения.

2. Необходимые изображения, такие как логотип или картинки, выводящиеся при ошибках или потерянном интернет-соединении хранятся в папке “images”.
3. Папка “icons” содержит уникальные иконки приложения, как, например, для отображения статуса книг.

В папке “components” хранятся все необходимые интерфейсные компоненты приложения, такие как:

1. Созданные дополнительно виджеты: поля для ввода, кнопки, различные диалоговые окна и т. д.
2. Нижняя панель навигации и ее анимация.
3. Граф переходов между экранами приложения. На рисунке 2 представлена папка

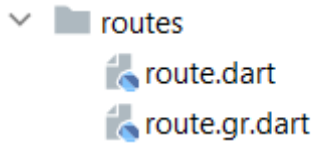

Рисунок 2. Архитектура графа переходов.

“routes”, состоящая из двух файлов:

3.1. routes.dart – содержит в себе описание всех возможных путей приложения

3.2. routes.gr.dart – сгенерированный библиотекой AutoRoute [9] граф переходов.

В папке “models” можно найти описание всех классов и сущностей, используемых в приложении (подробнее смотрите в приложении 1. «Описание и функциональное назначение классов» и приложении 2. «Описание и функциональное назначение полей, методов и свойств»).

Папка “pages” содержит в себе код описания для всех экранов, где каждый выделен в отдельную папку с одноименным названием. Структура каждой папки-страницы представлена на рисунке 3 на примере экрана авторизации. У каждого экрана есть основной файл с кодом, где происходит его создание, а также папка “components”, в которой хранятся описания переднего и заднего плана. Чаще всего под задним планом подразумеваются элементы интерфейса, не имеющие смысловой нагрузки и созданы только ради дизайна, например: лейблы, изображение и т. д. Передний план в свою очередь отвечает за все

элементы интерфейса, с которыми пользователь может как-то взаимодействовать. Данная архитектура помогает разбивать экраны на слои, для упрощения работы с ними.

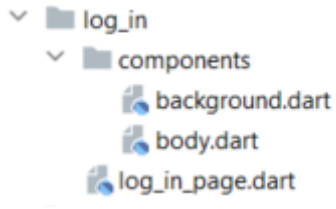

Рисунок 3. Структура папок с созданием экранов.

Некоторые экраны имеют другую структуру в силу того, что несут в себе большую смысловую нагрузку, как, например, страница профиля (подробнее смотрите в пункте 8. Профиль):

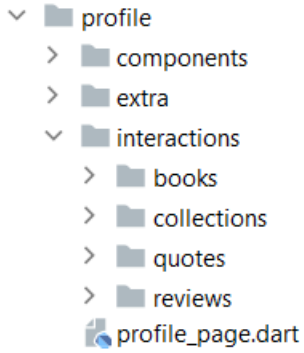

Рисунок 4. Структура экрана профиля.

Так как она содержит в себе статистику пользователя, помимо “components”, можно заметить папку “interactions”, состоящую из соответствующих экранов.

3.3. Описание алгоритма работы

Обмен данными между серверной и клиентской частями осуществляется посредством HTTP-методов (и библиотеки http.dart [12]). Было использовано 4 вида HTTP-запросов: POST, PUT, GET и DELETE:

- HTTP-запрос POST был применен для отправки новых данных на сервер: для создания нового пользователя, цитаты, отзыва, сборника. Так же для разного рода управления объектами, как, например: изменения статуса книги, добавления произведения в сборник и т. д.

- HTTP- запрос PUT был применен для обновления существующей информации на сервере. Для редактирования информации о пользователе: его аватара и баннера профиля, а также изменения данных сборника: обложка, название или описание.
- HTTP- запрос GET был применен для получения данных с сервера о ранее созданных объектах: пользователь и информации о нём, книги, хранящиеся в базе данных, отзывы и цитаты как пользователя, так и на произведение в целом и т. д.
- HTTP- запрос DELETE был применен для удаления существующих данных на сервере, например: цитат или пользовательских сборников.

Параметры могут передаваться тремя разными способами:

- Header – обычно в них передаются параметры, относящиеся к пользователю (имя, ID, флаги доступа, описание прав, страна и прочее).
- Path – параметры, по которым идентифицируются ресурсы (например, id коллекции, книги и т. д.).
- Body – остальное: данные для вставки и прочее (в GET-запросах нет тела).

3.4. Описание функционирования программы

3.4.1. Состав выполняемых функций

Список всех выполняемых функций можно найти в настоящем Техническом задании «Клиентская часть мобильного приложения «Shelfie» для трекинга и рекомендации книг».

3.4.2. Описание интерфейса и его работы

Для удобства «Описание интерфейса и его работы» имеет сокращенную нумерацию: ссылаясь на разделы, содержащееся в «Описание интерфейса и его работы», из других разделов или других документов, стоит использовать расширенную нумерацию, т. е. пункт «1. Авторизация» стоит нумеровать, как «3.4.2.1. Авторизация».

Так же описание запросов к серверу будет иметь следующий формат:

<ТИП ЗАПРОСА>: <КОНЕЧНАЯ ТОЧКА API [11]>

1. Авторизация

При открытии приложения первый экран – экран авторизации (Рисунок 6). Он содержит следующие элементы, для входа в аккаунт:

1.1. Поле для ввода почты.

1.2. Поле для ввода пароля, которое автоматически скрывает введенный пароль (рисунок 8).

1.2.1. На поле для ввода пароля есть иконка глаза, при нажатии на которую введенный пароль показывается (рисунок 9).

1.2.2. При нажатии иконка сменяется на другую в виде «зачеркнутого глаза», как показано на рисунке 9.

1.2.3. При повторном нажатии пароль снова скрывается, и иконка возвращается в свое первоначальное состояние рисунок 8.

1.3. Кнопка, для подтверждения введенных данных и входа в аккаунт («Войти»).

1.4. Кнопка для перехода на экран регистрации (подробнее в разделе «2. Регистрация»), обозначенная соответствующим подчеркнутым текстом «ЗАРЕГЕСТРИРОВАТЬСЯ»

Рисунок 8. Поле для пароля, скрывающее текст.

Рисунок 9. Поле для пароля, показывающее текст.

После ввода данных и нажатия на кнопку, описанную в пункте 1.3 выше, осуществляется запрос к серверу посредством POST: /users/user/login запроса, в который мы передаем введенную ранее почту и пароль. Пока приложение ожидает ответа, пользователь видит сообщение, как показано на рисунке 7.

Если ответом сервера был код ошибки, начинающийся с четырех, то программа понимает, что пользователь не был авторизован и на экране появляется соответствующее сообщение (Рисунок 6)

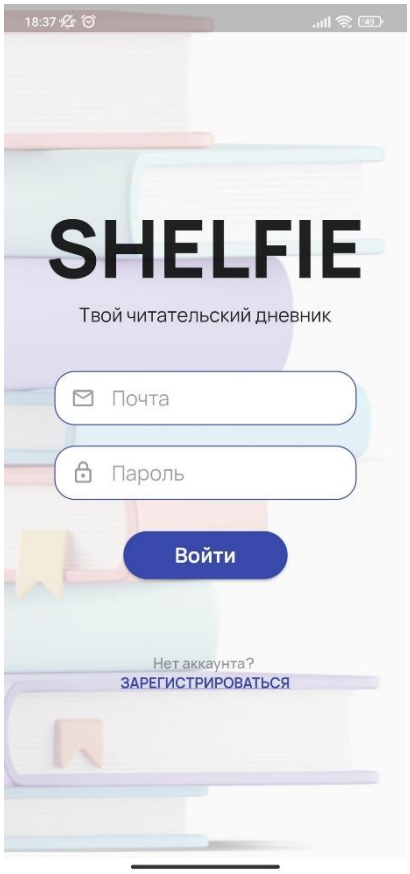

Рисунок 5. Экран авторизации.

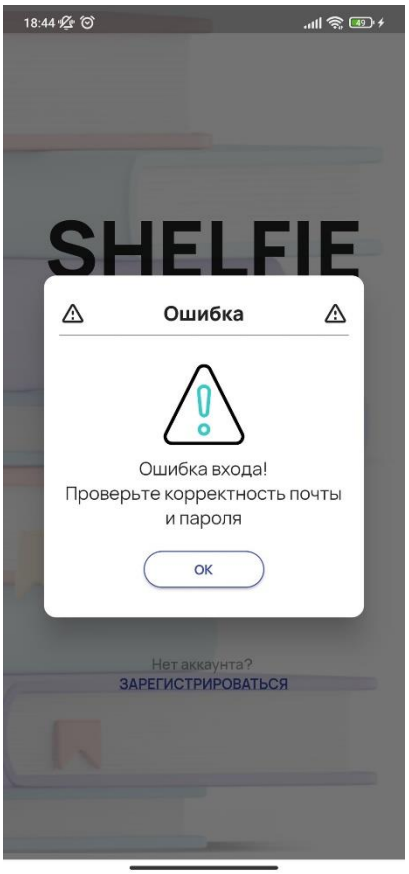

Рисунок 6. Ошибка авторизации.

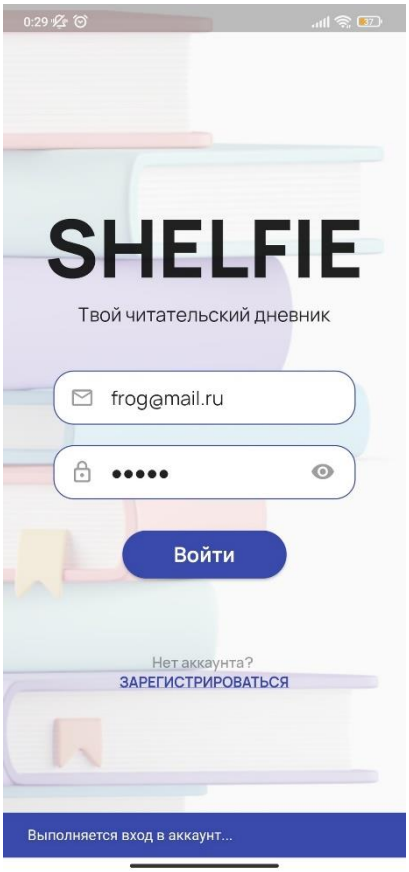

Рисунок 7. Сообщение при авторизации.

Если же ответ сервера – 200, значит введенные данные были найдены в базе данных и введены корректно. В этом случае пользователь успешно входит в приложение и его перенаправляют на экран с рекомендованными коллекциями (подробнее смотрите пункт «3. Коллекции»)

2. Регистрация

Для регистрации нового пользователя необходимо нажать на соответствующий элемент дизайна на экране авторизации (смотрите пункт 1.4 в разделе «1. Авторизация»).

Экран Регистрации содержит следующие элементы:

2.1. Поле ввода имени (логина).

2.2. Поле ввода почты.

2.3. Поле для ввода пароля, которое автоматически скрывает введенный пароль.

2.3.1. На поле для ввода пароля есть иконка глаза, при нажатии на которую введенный пароль показывается (рисунок 9).

2.3.2. При нажатии иконка сменяется на другую в виде «зачеркнутого глаза», как показано на рисунке 9.

2.3.3. При повторном нажатии пароль снова скрывается, и иконка возвращается в свое первоначальное состояние рисунок 8.

2.4. Кнопка для подтверждения введенных данных и подтверждения регистрации нового пользователя («Зарегистрироваться»).

2.5. Кнопка для перехода на экран авторизации (подробнее в разделе «1. Авторизация»), обозначенная соответствующим текстом «ВОЙТИ».

После заполнения всех полей и нажатия на кнопку, описанную в пункте 2.4 выше, происходит первичная проверка на валидность введенных данных. Если поля были заполнены некорректно, то пользователю показывается соответствующее сообщение в формате snackbar’a (Рисунок 13, Рисунок 14, Рисунок 15). Для удобства полноценный экран показан только на Рисунке 12, на остальных же показано только сообщение, которое появляется аналогично. Так, например

- Имя считается корректным, если в нем содержится от 2 до 30 символов.
- Почта считается корректной, если в ней содержится от 7 до 100 символов.
- Пароль считается корректным, если содержит от 8 до 30 символов.

Если все данные были введены корректно, то программа осуществляет POST: /users/user/add запрос к серверу, передавая ему введенные пользователем данные.

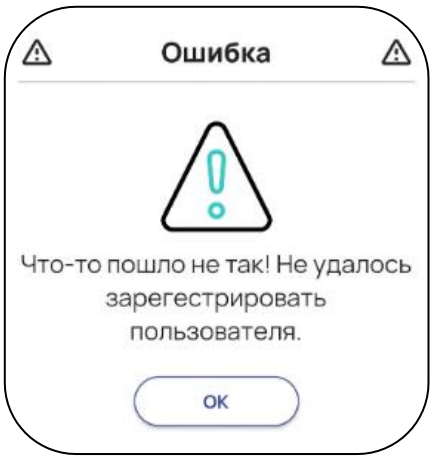

Рисунок 10. Ошибка при Регистрации.

Пока программа ожидает результата сервера, пользователь видит сообщение, изображенное на рисунке 11. Когда программа получает ответ сервера, то начинается проверка результатов.

Если код, вернувшийся с сервера – 400, то регистрация не могла быть осуществлена, так как пользователь с введенной почтой уже существует в базе данных. В таком случае пользователь видит сообщение, показанное на рисунке 10 (сообщение появляется на экране аналогично тому, как показано на рисунке 6.

Если же сервер вернул код 200, то регистрация успешно завершается и пользователя направляют на экран с рекомендованными коллекциями (смотрите раздел «3. Коллекции»), а также программа получает сгенерированный id для пользователя, для дальнейшей работы.

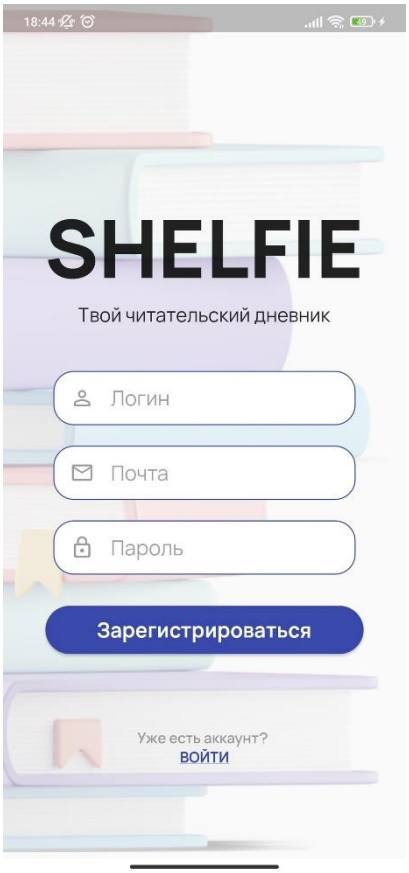

Рисунок 11. Экран Регистрации

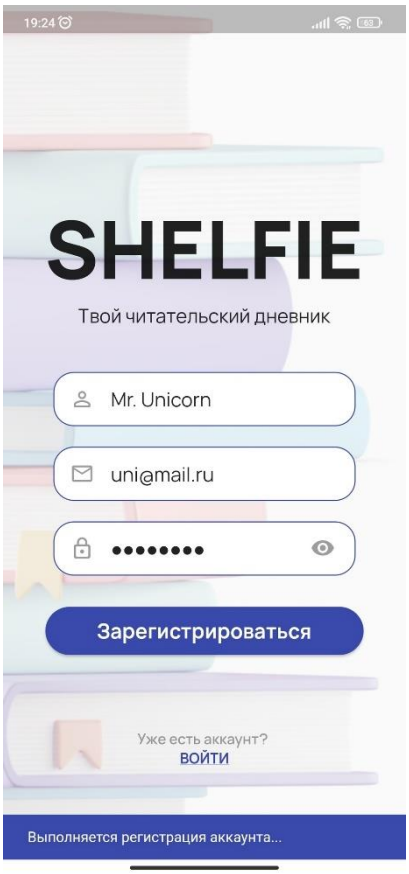

Рисунок 12. Сообщение при регистрации нового пользователя.

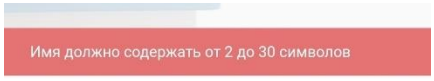

Рисунок 13. Сообщение об ошибке при некорректном заполнении поля "Логин".

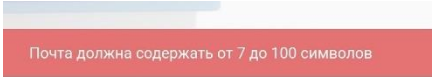

Рисунок 14. Сообщение об ошибке при некорректном заполнении поля "Почта".

Рисунок 15. Сообщение об ошибке при некорректном заполнении поля "Пароль".

3. Коллекции

После успешной авторизации/регистрации программа инициализирует домашнюю страницу с маршрутизатором в лице нижней панели навигации (подробнее смотрите в разделе 10), благодаря чему, пользователь попадает на экран с различными коллекциями, которые являются одинаковыми для всех пользователей и служат некой рекомендацией книг (рисунок 16). Программа получает эти данные при помощи GET: /shelves/collections/common запроса к серверу.

Экран коллекций состоит из следующих элементов:

3.1. Лейбл, показывающий, что пользователь сейчас находится на экране с коллекциями.

3.2. Лейбл, советующий пользователям, посмотреть список сгенерированных подборок (с текстом «Зацените наш топ подборок»).

3.3. Список рекомендованных коллекций, где каждый элемент представляется в виде карточки, на которой можно увидеть:

3.2.1. Название коллекции.

Рисунок 16. Экран рекомендованных коллекций.

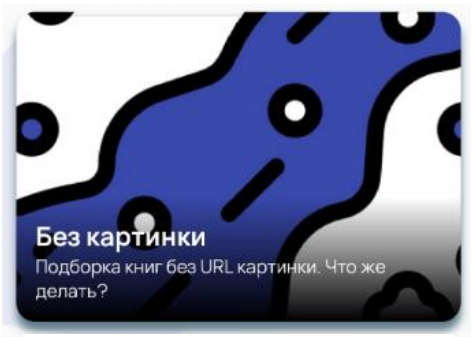

Рисунок 17. Коллекция со стандартной обложкой.

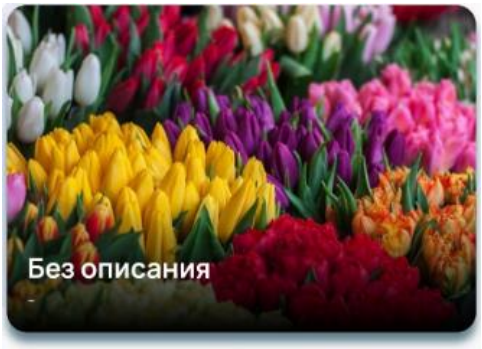

Рисунок 18. Коллекция без описания.

3.2.2. Краткое описание коллекции (если его нет, то вместо текста отображается прочерк, как на рисунке 18.

3.2.3. Изображение, соответствующее коллекции (если у коллекции нет обложки, то вместо нее отображается стандартное изображение коллекции (рисунок 17).

В случае, если у коллекции слишком длинное название или описание, текст обрезается, чтобы поместиться на карточке, а вместо оставшихся символов выводится многоточие, как показано на рисунках 19 и 20.

Рисунок 19. Отображение коллекции с длинным описанием.

Рисунок 20. Отображение коллекции с длинным названием.

Пользователь может просматривать содержимое любой коллекции, которая отображена в списке рекомендаций, просто нажав на нее.

При выполнении этого действия программа отправляет на сервер GET: /shelves/collections/{collectionId} запрос, передавая идентификаторы:

- Выбранной коллекции, как параметр path.
- Текущего пользователя в header.

Получая содержимое коллекции, то есть все книги, которые в ней содержатся, где о каждом произведении известно: ID, название, список авторов, обложка книги, список жанров, средний рейтинг, который поставили пользователи приложения, оценка, которую поставил пользователь, а также текущий статус, в котором она находится. Подробнее о том, как отображается эта информация и как устроена логика данного экрана описана в пункте 4. Поиск книг.

4. Поиск книг

На данный экран можно попасть двумя способами:

- При нажатии на выбранную из списка коллекцию/сборник.
- При нажатии на элемент нижней панели навигации (подробнее смотрите пункт 10) с иконкой лупы.

В первом случае пользователь видит список со всеми книгами, которые включены в эту коллекцию (рисунок 27). Интерфейс данной страницы содержит следующие элементы:

4.1. Лейбл с названием коллекции (если название не помещается на страницу целиком, то оно обрезается, и вместо оставшихся букв показывается многоточие - …, рисунок 28)

4.2. Список всех книг в коллекции, для каждой из которых отображается:

4.2.3. Обложка произведения.

4.2.4. Название (если название не помещается на страницу целиком, то оно обрезается, и вместо оставшихся букв показывается многоточие - …, рисунок 21)

4.2.5. Автор произведения (если автор не один, то на карточке показываются только первые два автора, перечисленные в столбце, как показано на рисунке 21)

Рисунок 21. Экран книг в коллекции.

4.2.6. Первые три жанра произведнния, если их меньше, то отображаются все, что есть.

4.2.7. Средний рейтинг книиги, расчитанный по отзывам всех пользователей, которые ее прочитали.

Рисунок 22. Иконка статуса «Не читаю».

Рисунок 23. Иконка статуса «Буду читать».

Рисунок 24. Иконка статуса «Читаю».

Рисунок 25. Иконка статуса «Прочитал».

Рисунок 26. Иконка статуса «Перестал».

4.2.8. Иконка с текущим статусом произведения, которая имеет пять вариантов отображения:

4.2.8.1. Не читаю (рисунок 22)

4.2.8.2. Буду читать (рисунок 23)

4.2.8.3. Читаю (рисунок 24)

4.2.8.4. Перестал (рисунок 25)

4.2.8.5. Прочитал (рисунок 26)

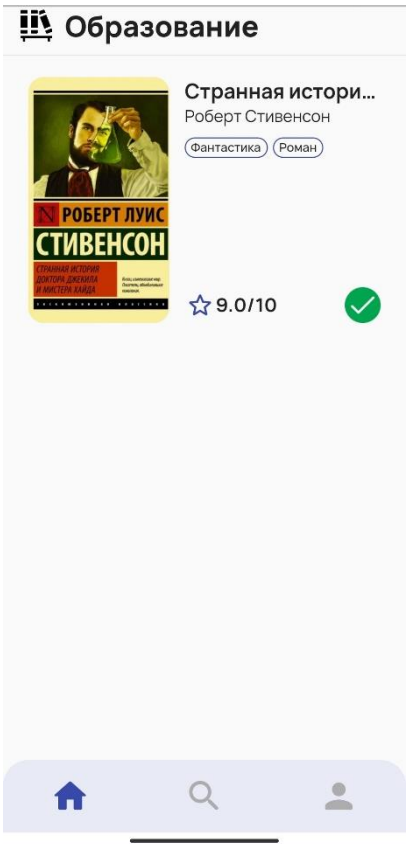

Рисунок 27. Экран книг в коллекции.

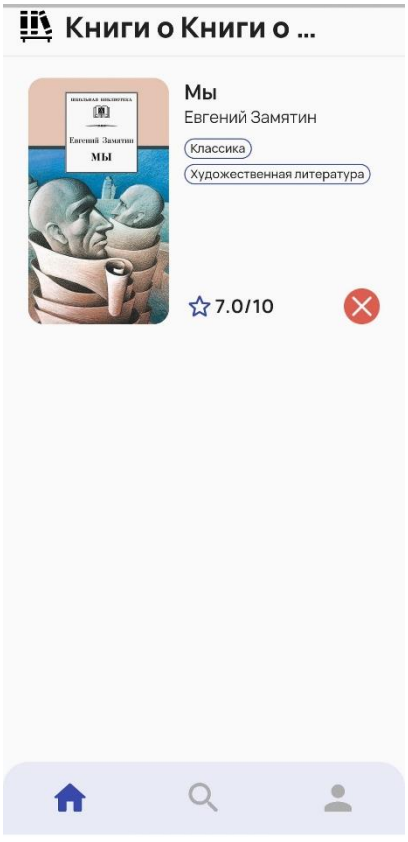

Рисунок 28. Экран книг в коллекции, с длинным названием.

Пользователь может поменять статус произведения различными способами:

- При однократном нажатии в случае, если произведения не было в библиотеке пользователя, оно туда добавляться, а иконка плюса сменяется на изображение двух стрелок (т. е. статус книги сменяется на «Буду читать»).
- При однократном нажатии в случае, если произведение уже есть в библиотеке пользователя, оно из нее удаляется, а иконка статуса книги сменяется на изображение по умолчанию (т. е. статус книги сменяется «Не читаю»).
- При двукратном нажатии произведение отмечается прочитанным, а иконка статуса книги сменяется на изображение галочки (т. е. статус книги сменяется на «Прочитано»).

При выполнении этих действия программа отправляет на сервер POST: /interactions/books/update-status запрос, передавая идентификаторы: текущего пользователя в header, а также идентификатор выбранной книги и ее новый статус в теле запроса.

В случае, если сервер вернет ответ 200, то программа поймёт, что статус был успешно изменен и отобразит иконку заново.

Если же возвращенный код отличается от двухсот, то программа понимает, что статус не был изменен. Иконка остается прежней, а пользователь получает сообщение с текстом: «*Что-то пошло не так! Статус не был изменен*». Сообщение отображается аналогично тому, что было показано выше при описании ошибки авторизации (рисунок 10).

Иначе, если пользователь попал на данный экран при помощи нижней панели навигации, то интерфейс страницы будет отличаться (рисунок 29), а именно, кроме списка книг на экране можно будет увидеть:

4.1 Кнопку, для фильтрации книг (рисунок 30). При нажатии на данную кнопку открывается диалоговое окно, а котором модно настроить различные параметры, по которым будет происходить поиск книг (подробнее о данном функционале можно прочитать в пункте 7. Фильтрация)

4.2 Поисковую строку (рисунок 31).

4.3 Кнопку, для поиска книги по ее ISBN (рисунок 32).

Рисунок 29. Экран поиска книг.

Рисунок 30. Кнопка фильтрации.

Рисунок 31. Поисковая строка.

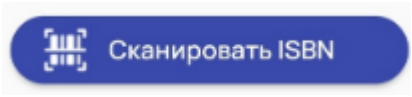

Рисунок 32. Кнопка поиска книги по ISBN.

При нажатии на поисковую строку открывается клавиатура. Пользователь может ввести интересующее его название или автора, чтобы найти произведение. После того, как юзер введет текст для поиска, он должен нажать кнопку «ввод» на клавиатуре. При нажатии на эту кнопку программа отправляет на сервер POST: /books/search запрос, передавая идентификатор текущего пользователя в header и введенную пользователем строку в теле.

В ответ, если возвращенный код был 200, программа получает список всех книг, удовлетворяющих введенному запросу (как показано на рисунке 33). Если же в базе данных не было найдено ни одной книги, которая бы соответствовала предоставленному тексту, то пользователь видит сообщение, оповещающее его об этом (рисунок 34).

Пользователь может просматривать подробную информацию о любой книге из списка на экране поиска. Для этого ему нужно нажать на нее. При выполнении этого действия программа отправляет на сервер GET: /books/books/{bookId} запрос, передавая идентификатор текущего пользователя в header и идентификатор выбранной книги в path.

Получая всю необходимую информацию, а именно: id, название, обложка, список авторов, пользовательский и средний рейтинги, список жанров, язык оригинала, возрастные ограничения, статистику по пользователям, содержащую среднее время чтения и количество пользователей, уже прочитавших данное произведение, описание и текущий статус (подробнее о том, как работает данная страница можно найти в пункте 5. Книги).

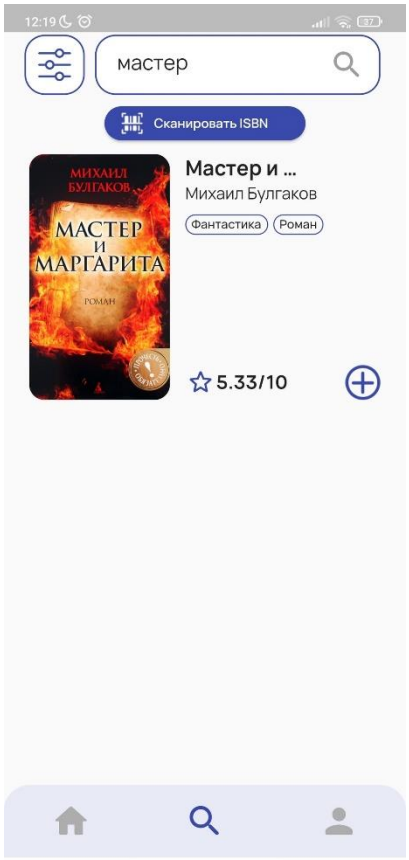

Рисунок 33. Поиск книги при помощи поисковой строки.

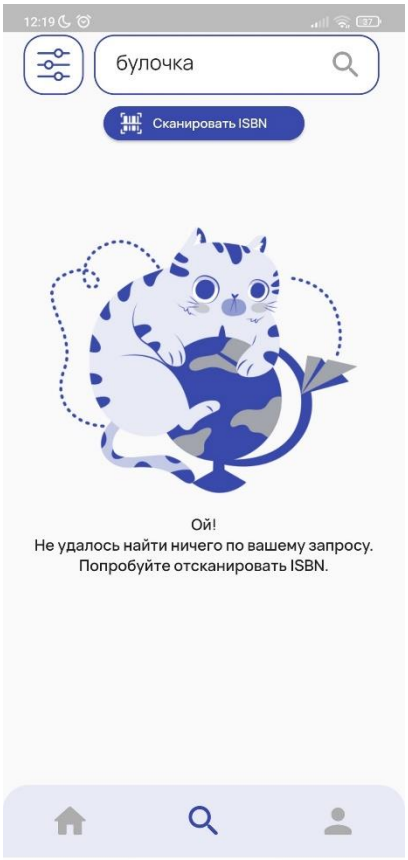

Рисунок 34. Экран, оповещающий, что по запросу ничего не было найдено.

При нажатии на кнопку для поиска книги по ISBN, описанную в пункте 4.3. выше, приложение попросит у пользователя доступ к камере. Если он его не даст, то программа сообщит о том, что не сможет выполнить сканирование ISBN без данного разрешения и вернется на экран поиска. Если же пользователь даст приложению доступ к камере, то он сможет отсканировать ISBN своей книги, чтобы быстрее найти ее в базе с помощью bar_code_scanner [10].

Так как ISBN представляет из себя штрих код, то если пользователь отсканирует что-то другое, например QR код (программа это понимает, проверяя длину отсканированных

данных, т. к. ISBN состоит из 13 символов), то он увидит сообщение об ошибке, как на рисунке 35.

Если же был отсканирован штрих-код, то программа отправляет на сервер GET: /books/search/{isbn} запрос, передавая идентификатор текущего пользователя в header и отсканированный ISBN книги в path.

В случае, если ответом сервера была ошибка с кодом 404, то программа понимает, что такой книги не было найдено в базе данных и выводит пользователю соответствующее сообщение (рисунок 36).

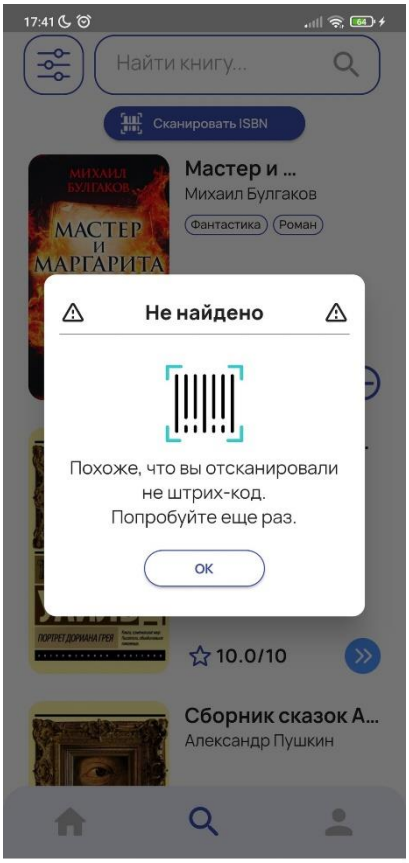

Рисунок 35. Экран ошибки сканирования штрих-кода.

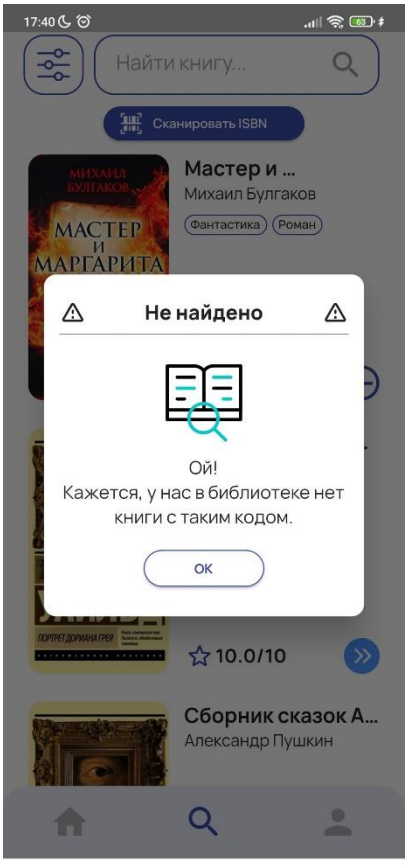

Рисунок 36. Экран ошибки, если книги не было найдено в БД.

Если же сервер вернул код 200, то вместе с ним программа получает подробную информацию о книге и перенаправляет пользователя на страницу, где отображает полученные данные (подробнее смотрите в пункте 5. Книги).

5. Книги

При нажатии на выбранный элемент списка с книгами пользователь попадает на экран с подробной информацией о выбранном элементе со следующими компонентами:

5.1. Обложка произведения

5.2. Название произведения (если не помещается на страницу целиком в две строки, то оно обрезается, и вместо оставшихся букв показывается многоточие - …)

5.3. Список авторов произведения, перечисленных в столбце

5.4. Виджет с десятью звездами – рейтинг, который пользователь ставит произведению при написании отзыва. В зависимости от того, сколько баллов из 10 юзер поставил книге, столько звезд будут желтыми, а остальные серыми)

5.5. Язык оригинала, если он не известен, то отображается прочерк.

5.6. Возрастные ограничения. Если они отсутствуют, то пишется «нет», иначе он выводятся в формате <минимальный_допустимый_возраст>+ (рисунок 38).

5.7. Список всех жанров произведения

5.8. Строка со статистикой по книге, а именно:

5.8.1. Среднее время чтения. Если ни один пользователь еще не прочитал произведение, то вместо времени написано «Неизвестно», как показано на рисунке 40.

5.8.2. Количество пользователей, которые прочитали произведение.

5.8.3. Количество рецензий на книгу.

5.8.4. Количество цитат из книги.

5.9. Панель вкладок с текущим статусом произведения.

5.10. Описание произведения (изначально отображается только первые семь строк).

5.11. Кнопка для просмотра описания целиком с текстом «Развернуть» (рисунок 39).

5.12. Панель вкладок с рецензиями и цитатами (подробнее в разделе 6. Панель рецензий и цитат).

5.13. Кнопка для добавления в сборник (рисунок 41)

Основной интерфейс страницы представлен на рисунке 37.

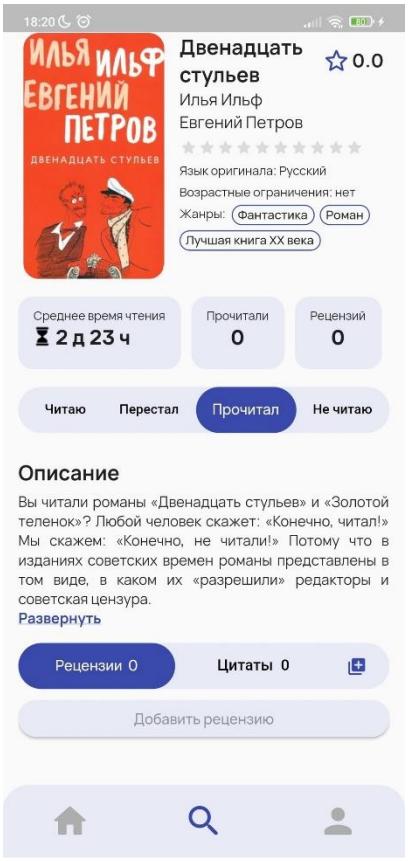

Рисунок 37. Экран с подробной информацией о книге.

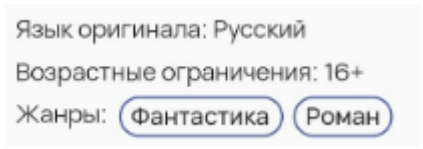

Рисунок 38. Пример отображения возрастных ограничений.

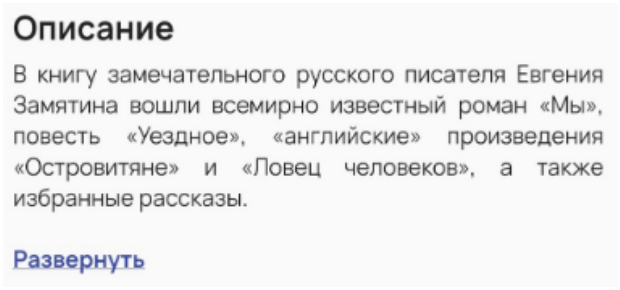

Рисунок 39. Пример отображения описания произведения с кнопкой, чтобы разворачивать его полностью.

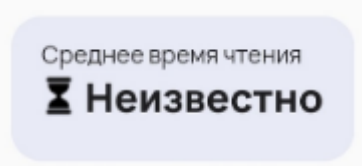

Рисунок 40. Пример отображения среднего времени, если оно неизвестно.

Рисунок 41. Кнопка добавления книги в сборник.

Пользователь может сменить текущий статус произведения при нажатии на соответствующую вкладку на панели, описанной в пункте 5.9 (рисунок 42). При выполнении этого действия программа отправляет на сервер POST: /interactions/books/update-status запрос, передавая идентификаторы: текущего пользователя в header, а также идентификатор выбранной книги и ее новый статус в теле запроса.

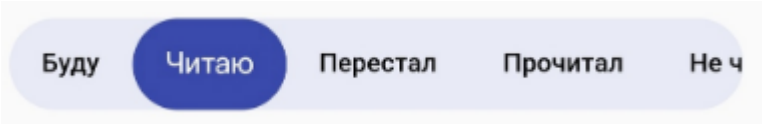

Рисунок 42. Пример отображения панели вкладок с текущим статусом книги.

В случае, если сервер вернет ответ 200, то программа поймёт, что статус был успешно изменен и выделит новый статус (под выделением понимается перенос ползунка, так на рисунке выше «выделен» статус «Читаю»).

Если же возвращенный код отличается от двухсот, то программа понимает, что статус не был изменен. Иконка остается прежней, а пользователь получает сообщение с текстом: «*Что-то пошло не так! Статус не был изменен*». Сообщение отображается аналогично тому, что было показано выше при описании ошибки авторизации (рисунок 10).

Пользователь может читать полное описание произведения. Так как изначально отображается только первые семь строк, юзер может нажать на кнопку, описанную в пункте 5.11. выше с текстом «Развернуть». При нажатии на данную кнопку описание книги показывается полностью, все элементы интерфейса соответственно сдвигаются ниже, а текст кнопки сменяется на «Свернуть». Таким образом, при нажатии на кнопку снова (когда на ней уже написано «Свернуть») описание снова сворачивается, и отображается только первые 7 строк. Для наглядности можно обратиться к рисункам 43 и 44.

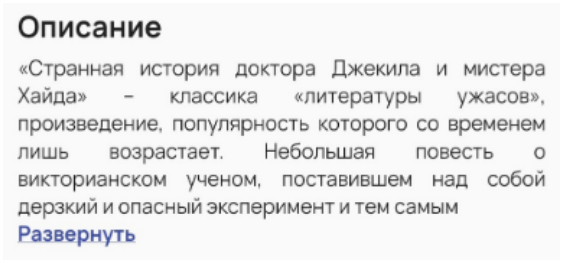

Рисунок 43. Пример отображения описания книги, со свернутым описанием.

Рисунок 44. Пример отображения описания книги, с развернутым описанием.

6. Панель рецензий и цитат

Данный виджет является часть экрана с подробной информацией о книге, описанного выше. Для удобства его описание было выделено в отдельный пункт. Сам виджет представлен на рисунке 45 ниже.

Рисунок 45. Виджет панели вкладок с рецензиями и цитатами.

Данная панель состоит из двух вкладок и кнопки, которая упоминалась ранее в пункте 5.13. По умолчанию, при переходе на страницу открывается вкладка с рецензиями. Сама панель представляет из себя набор следующих элементов:

6.1. Вкладка рецензий, на которой помимо лейбла «Рецензии» так же написано общее количества рецензий на книгу.

6.2. Вкладка цитат, на которой помимо лейбла «Цитаты» так же написано общее количества цитат из книги.

6.3. Кнопка для добавления произведения в сборник.

Перед тем, как отобразить список всех доступных рецензий программа отправляет на сервер GET: /interactions/reviews запрос, передавая идентификатор текущего пользователя в header, а также идентификатор выбранной книги в теле path. В качестве ответа программа получает количество рецензий и список всех отзывов, каждый из которых имеет: ID, название рецензии, основной текст, оценка произведения, а также основная информация о пользователе, который оставил данную рецензию (ID, имя и аватар).

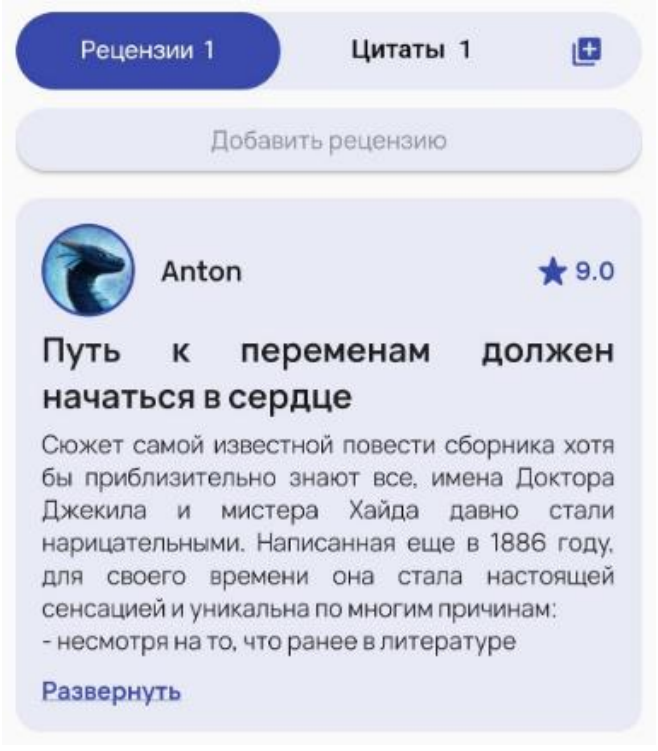

Рисунок 46. Отображение вкладки рецензий.

Аналогично описанию произведения, текст рецензии показывается не полностью, а только первые семь строк. При нажатии на кнопку можно разворачивать/сворачивать текст по желанию.

Рисунок 47. Кнопка добавления новой рецензии.

Помимо списка отзывов о книге, на вкладке с рецензиями так же есть кнопка добавления новой (рисунок 47). При нажатии на нее открывается диалоговое окно, представленное на рисунке 48. На нем пользователь может наблюдать информацию о книге, на которую он собирается оставить рецензию. Также он может выставить оценку произведению нажав на иконку звезды с соответствующим номером. Если пользователь хочет поставить оценку ноль звезд, то ему нужно дважды нажать на звезду с номером один (самую левую на экране). При выставлении оценки количество звезд будет закрашиваться в желтый цвет (рисунок 49). После заполнения всех необходимых полей (название и основной текст) пользователь может либо отменить создание отзыва, нажав на соответствующую кнопку с текстом «Отменить», либо подтвердить добавление, нажав на кнопку «Добавить».

При выполнении второго действия первым делом происходит первичная проверка введенных данных. В данной проверке программа смотрит, что пользователь заполнил оба необходимых поля: название и текст. Если же это не так, то выводится сообщение об ошибке в формате snackbar’a, как показано на рисунках 49 и 50. если же первичная проверка прошла успешно, то на сервер отправляется POST: /interactions/reviews/add запрос, передавая идентификатор текущего пользователя в header. Основная введенная информация (текст названия, рецензии и оценка произведения) а также идентификатор выбранной книги передаются в теле запроса.

Если ответом сервера будет код 200, то программа поймет, что добавление произошло успешно и обновит текущий список рецензий, добавив в него новый. Обновление списка происходит аналогично инициализации, т. е. путем отправки GET: /interactions/reviews запроса.

Если же что-то пошло не так и полученный код будет отличаться, то пользователь увидит сообщение, что его рецензия не была добавлена.

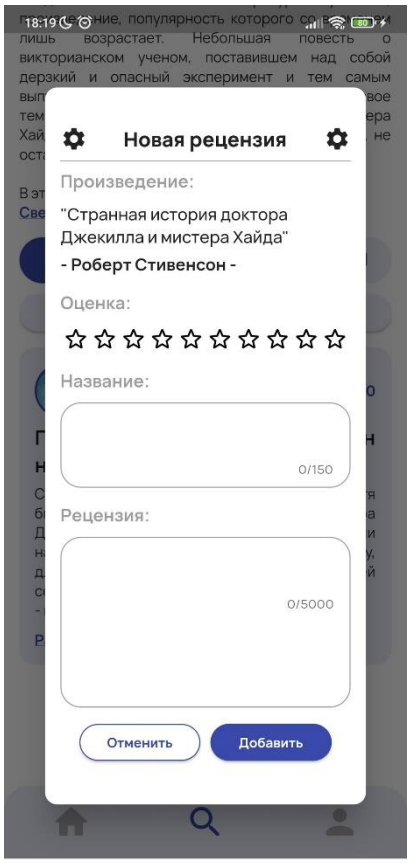

Рисунок 48. Диалоговое окно для создания новой рецензии.

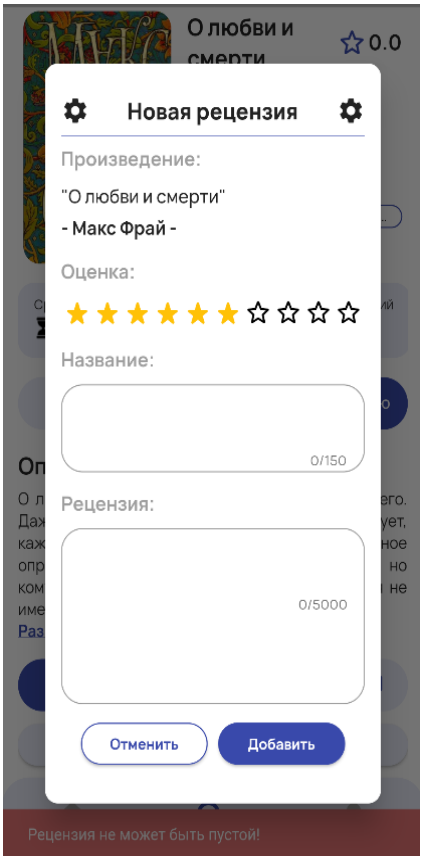

Рисунок 49. Пример отображения ошибки при создании рецензии.

Рисунок 50. Пример отображения ошибки при создании рецензии без названия.

Аналогичным образом работает вкладка с цитатами. Перед тем, как отобразить список всех доступных цитат программа отправляет на сервер GET: /interactions/quotes запрос, передавая идентификатор текущего пользователя в header, а также идентификатор выбранной книги в теле path. В качестве ответа программа получает количество и их где каждый элемент имеет: ID, текст, и параметр Boolean типа (т. е. true или false) сохранена ли данная цитата у пользователя или нет . В зависимости от этого параметра интерфейс карточки может отличаться. Сама вкладка представлена на рисунке 51.

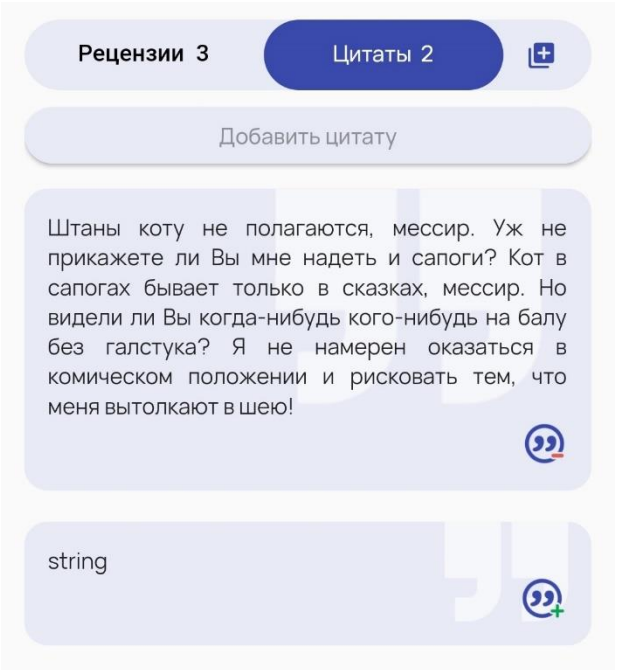

Рисунок 51. Пример отображения вкладки с цитатами.

Рисунок 52. Кнопка добавления цитаты в свою коллекцию.

Рисунок 53. Кнопка удаления цитаты из своей коллекции.

Как можно заметить, в зависимости от того, сохранена ли цитата у пользователя в нижнем правом углу карточки с цитатой, изменяется дизайн кнопки (рисунки 52 и 53).

Так, если пользователь нажмет на кнопку, как на рисунке 52, т. е. захочет добавить чью-то цитату к себе в коллекцию, программа отправит на сервер POST: /interactions/quotes/{quoteId}/save запрос, передавая идентификатор текущего пользователя в header, а также идентификатор выбранной цитаты в path. Если полученный от сервера ответ – 200, то кнопка меняется на противоположную (в данном примере на ту, что изображена на рисунке 53). Если же ответ был получен другой ответ, то пользователь получает сообщение в том же формате, что и на рисунке 10 с текстом: «Что-то пошло не так! Цитата не была добавлена в коллекцию».

Если же пользователь нажмет на кнопку, как на рисунке 53, т. е. захочет удалить цитату из своей коллекции, программа отправит на сервер DELTE: /interactions/quotes/{quoteId}/unsave запрос, передавая те же параметры, что и при добавлении. Аналогично, если полученный от сервера ответ – 200, то кнопка меняется на противоположную (в данном примере на ту, что изображена на рисунке 52), в противном случае пользователь получает сообщение текстом: «Что-то пошло не так! Цитата не была удалена из коллекции».

Пользователь может создать новую цитату, нажав на соответствующую кнопку, как на рисунке 54.

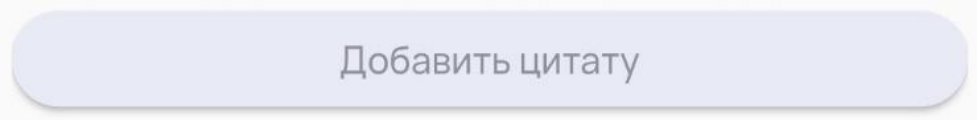

Рисунок 54. Кнопка добавления новой цитаты.

При нажатии на нее открывается диалоговое окно, как на рисунке 55, где отображена информация о произведение, к которому будет привязаны цитата, а также текстовое поле для самой цитаты. После заполнения поля пользователь может либо отменить добавление цитаты, тогда диалоговое окно будет просто закрыто, либо подтвердить добавление.

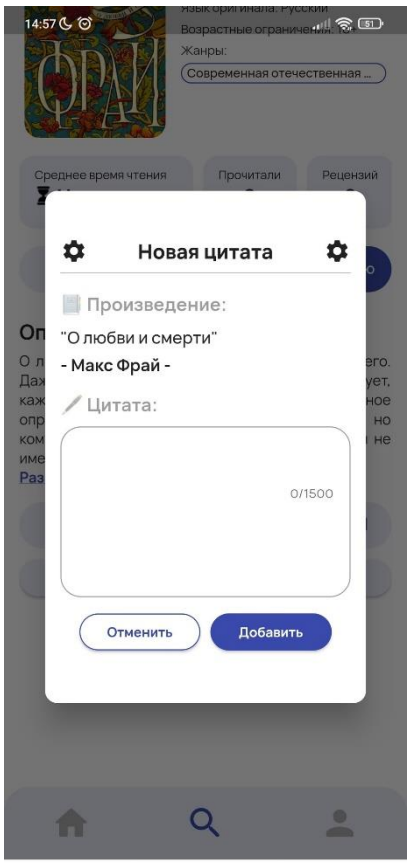

Рисунок 55. Диалоговое окно для создания новой цитаты.

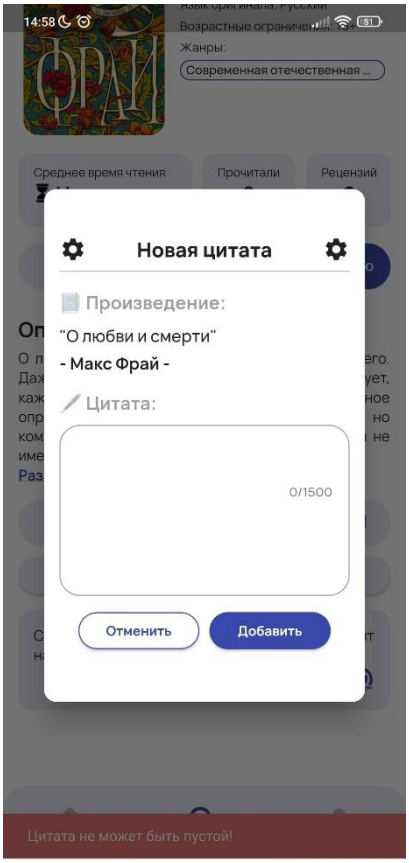

Рисунок 56. Пример ошибки при создании пустой цитаты.

При выполнении второго действия первым делом происходит первичная проверка данных. В данной проверке программа смотрит, что пользователь заполнил необходимое поле для текста цитаты. Если же это не так, то выводится сообщение об ошибке в формате snackbar'a, как показано на рисунках 49 и 50, только с текстом «Цитата не может быть пусто!», как на рисунке 56. если же первичная проверка прошла успешно, то на сервер

отправляется POST: /interactions/quotes/add запрос, передавая идентификатор текущего пользователя в header, а введенная текст и идентификатор выбранной книги в теле запроса.

Если ответом сервера будет код 200, то программа поймет, что добавление произошло успешно и обновит текущий список цитат, добавив в него новый. Обновление списка происходит аналогично инициализации, т. е. путем отправки GET: /interactions/quotes запроса.

Если же что-то пошло не так и полученный код будет отличаться, то пользователь увидит сообщение, что его цитата не была добавлена.

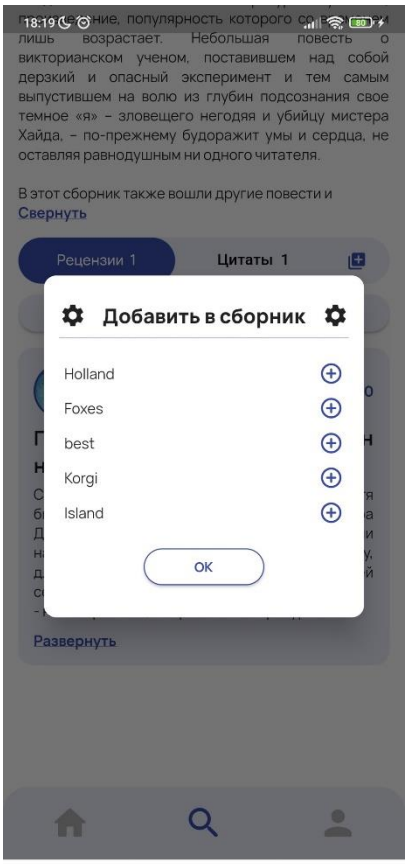

Рисунок 57. Диалоговое окно с коллекциями (книги нет ни в одной)

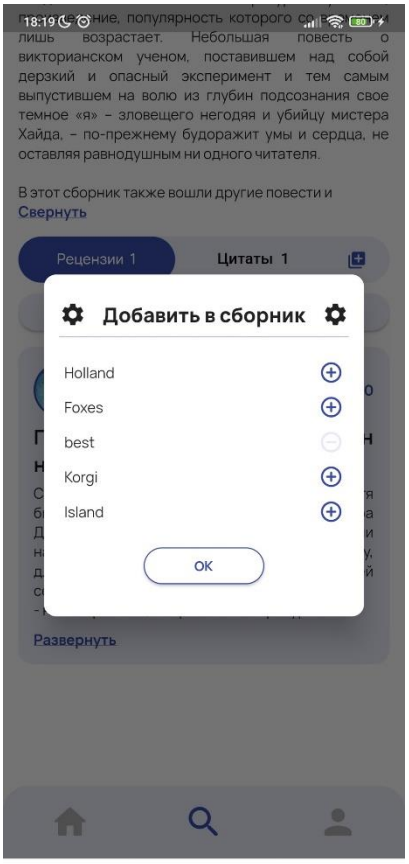

Рисунок 58. Диалоговое окно с коллекциями (книга есть в «best»).

При нажатии на кнопку, описанную 6.3., пользователь может добавлять книгу в существующий сборник. После нажатия выполняется GET: /shelves/collections/check запрос, передавая идентификатор текущего пользователя в header, а идентификатор выбранной книги в теле запроса. В качестве ответа программа получает список коллекций, о каждой из которых известно: ID, название и присутствует ли текущая книга в этом сборнике. Интерфейс данного списка можно увидеть на рисунках 57 и 58.

Если книги нет в коллекции, то иконка на против названия отображается в виде плюса, как на рисунке 59, при нажатии на который программа отправляет POST: /shelves/collections/{collectionId}/add/{bookId} запрос, передавая идентификатор текущего пользователя в header, а в path идентификаторы выбранной коллекции и текущей книги.

Если полученный от сервера ответ – 200, то программа понимает, что книга была успешно добавлена в коллекцию и меняет иконку плюса на минус, как на рисунке 60.

Рисунок 59. Кнопка добавления книги в сборник

Рисунок 60. Кнопка удаления книги из сборника

Если книги уже есть в коллекции, то иконка на против названия отображается в виде светлого минуса, как на картинке 60, при нажатии на который программа отправляет DELTE: /shelves/collections/{collectionId}/remove/{bookId} запрос, передавая идентификатор текущего пользователя в header, а в path идентификаторы выбранной коллекции и текущей книги.

Если полученный от сервера ответ – 200, то программа понимает, что книга была успешно удалена из коллекции и меняет иконку на противоположную.

Так же пользователь может просто закрыть окно сборников, если не хочет добавлять или удалять книги. Тогда он просто должен нажать на кнопку «ОК», которая закроет диалоговое окно, не внося никаких изменений.

7. Фильтрация

Пользователь может настраивать фильтры при поиске книг. Для этого ему нужно нажать на кнопку, описанную в пункте 4.1. и изображенную на рисунке 30. При нажатии на данную кнопку открывается диалоговое окно, изображенное на рисунках 61 и 62. В этом диалоговом окне есть несколько параметров фильтрации:

7.1. Жанр произведения

7.2. Язык, на котором написан оригинал

7.3. Минимальный рейтинг книг

7.4. Возрастные ограничения

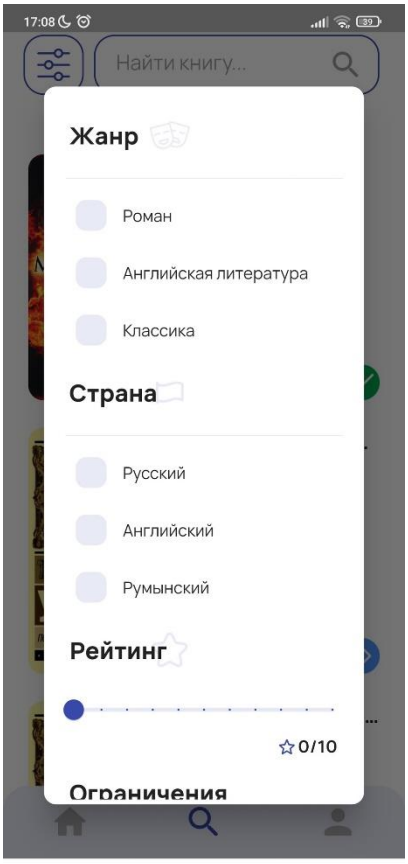

Рисунок 61. Диалоговое окно с фильтрами (первая часть).

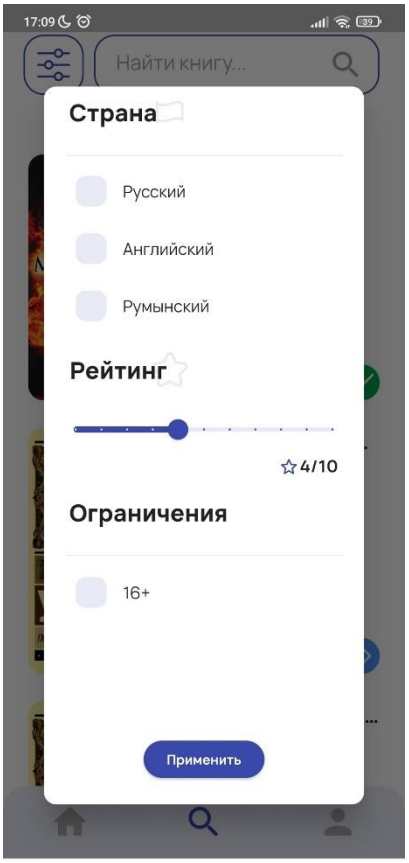

Рисунок 62. Диалоговое окно с фильтрами (вторая часть).

Когда пользователь нажимает на кнопку фильтрации, перед тем как показать диалоговое окно, программа отправляет на сервер GET: /books/search/filters запрос, чтобы получить список возможных вариантов. В качестве ответа на запрос программа получает словарь элементов, где в качестве ключа выступает название категории для фильтрации, а в качестве значения – список из всех возможных вариантов, которые сейчас есть в базе.

Настройка фильтров устроена по-разному, так, например, категории жанр, страна и ограничения представляют из себя список, где выбранные элементы отображаются сверху со значком крестика, чтобы была возможно удалить их из выбранных (рисунок 63).

Выбор минимального рейтинга же представляет из себя слайдер, который нажатием и перетаскиванием настраивает минимальный рейтинг (рисунок 64).

После настроек всех необходимых параметров пользователь может применить фильтры, нажав на соответствующую кнопку, которую можно увидеть на рисунке 62. При

нажатии на эту кнопку программа сохраняет выстроенные пользователем настройки и отправляет на сервер POST: /books/search запрос, передавая идентификатор текущего пользователя в header и параметры фильтрации.

В ответ, если возвращенный код был 200, программа получает список всех книг, удовлетворяющих введенному запросу. Если же в базе данных не было найдено ни одной книги, которая бы соответствовала предоставленному тексту, то пользователь видит такое же сообщение, как при попытке поиска через строку, сообщение, оповещающее его об этом, как на рисунке 34.

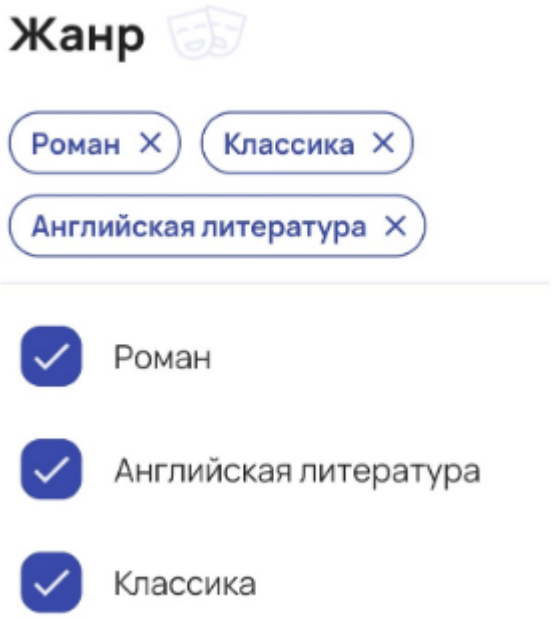

Рисунок 63. Пример отображения списка фильтрации.

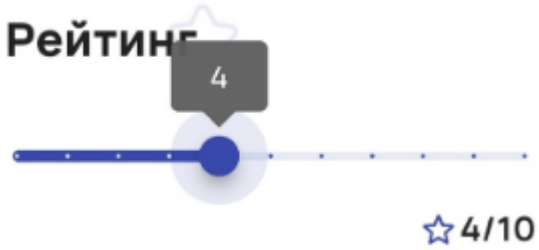

Рисунок 64. Пример отображения слайдера для настройки минимального рейтинга.

8. Профиль

Пользователь может попасть на экран профиля через меню (подробнее смотрите Пункт X. Нижняя панель навигации). При попадании на эту страницу программа делает запрос к серверу GET: /users/profile/{userId} запрос, чтобы получить информацию о профиле, а именно: имя, почта, изображение профиля и баннера, а также статистику пользователя.

После получения ответа сервера программа отображает данные в формате, представленном на рисунке 65. Как можно заметить, страница состоит из следующих элементов:

8.1. Баннер профиля (по умолчанию ставится картинка, как на рисунке 66).

8.2. Аватар пользователя (по умолчанию ставится картинка, как на рисунке 66).

8.3. Логин (если он не помещается на старицу целиком, то текст обрезается и вместо него дописывается многоточие, как на рисунке 67).

8.4. Почта (если она не помещается на старицу целиком, то текст обрезается и дописывается многоточие, как на рисунке 67).

8.5. Строка со статистикой.

8.6. Меню (подробнее смотрите пункт 9. Меню профиля).

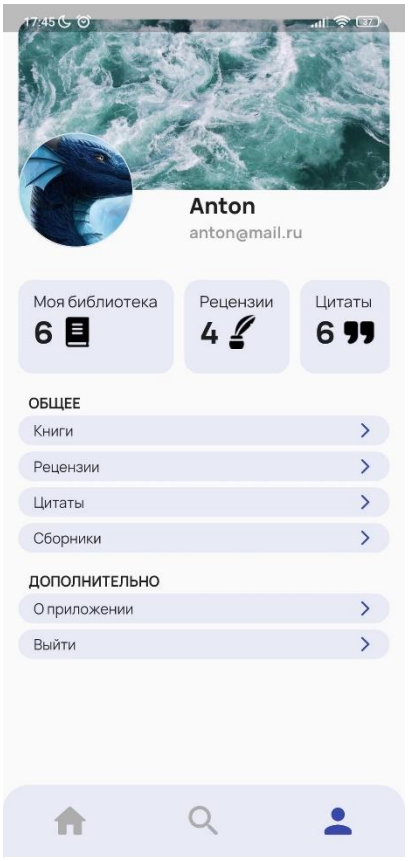

Рисунок 65. Экран профиля пользователя.

Рисунок 66. Пример отображения аватара и баннера по умолчанию.

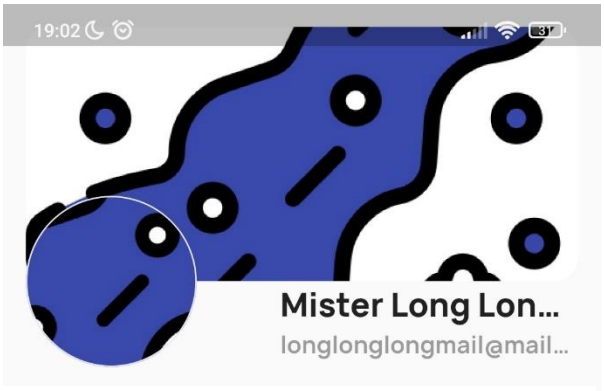

Рисунок 67. Пример отображения длинных логина и почты.

Пользователь может сменить баннер профиля, описанный в пункте 8.1. путем нажатия на него. При выполнении это действия выводится диалоговое окно, как на рисунке 68.

Пользователь может вставить в текстовое поле прямую ссылку на изображение, чтобы изменить картинку. Затем юзер может выбрать: отменить изменения, нажав на кнопку «Отменить» или применить изменения, нажав на кнопку «Изменить».

При выборе второго варианта программа сначала производит первичную проверку на пустоту введенного текста. Если поле пусто, то пользователь видит сообщение об ошибке в формате snackbar'a, как было показано, например, на рисунке 56. В противном случае на сервер отправляется запрос PUT: /users/user/{userId}/set-banner запрос, передавая идентификатор текущего пользователя в header и значение текстового поля в теле.

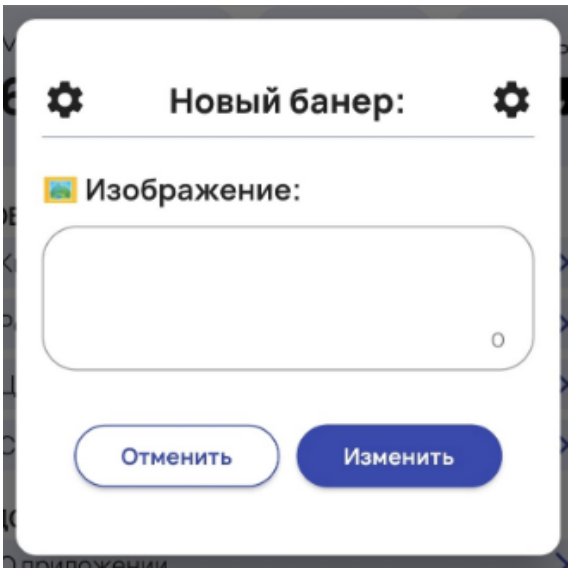

Рисунок 68. Диалоговое окно для смены баннера профиля.

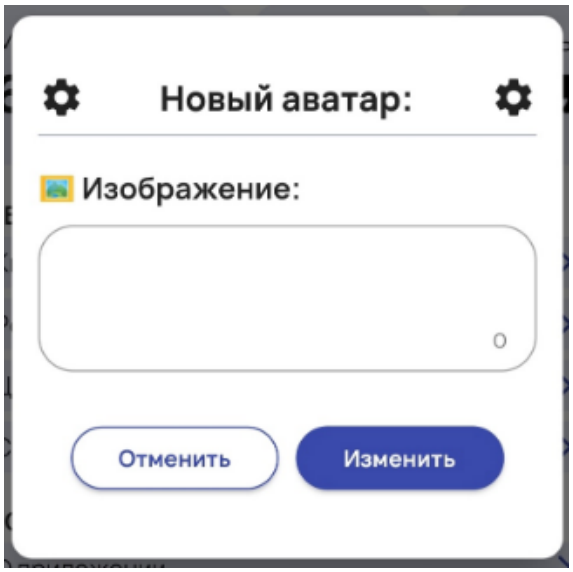

Рисунок 69. Диалоговое окно для смены аватара профиля.

Если ответ сервера был 200, то программа понимает, что баннер был успешно изменён и обновляет страницу профиля, повторно делая запрос GET: /users/profile/{userId}, получая уже обновленные данные.

Если же ответ был иной, то программа понимает, что смене изображение, что-то пошло не так и выводит соответствующее сообщение.

Аналогично смене баннера, пользователь может поменять аватар, путем нажатия на него. В этом случае появляется диалоговое окно для смены картинки профиля, как показано на рисунке 69. Работает он аналогично смене баннера, за исключением того, что выполняется другой запрос: PUT: /users/user/{userId}/set-avatar.

9. Меню профиля

Как можно заметить на рисунке 65, у пользователя есть меню, состоящее из 6 пунктов:

9.1. Книги

9.2. Рецензии

9.3. Цитаты

9.4. Сборники

9.5. О приложении

9.6. Выйти

При нажатии на пункт меню «Книги», описанный в 9.1. или элемент строки статистики, описанный в пункте 8.5. с текстом «Моя библиотека», пользователя перенаправляют на экран с его книгами (рисунок 70).

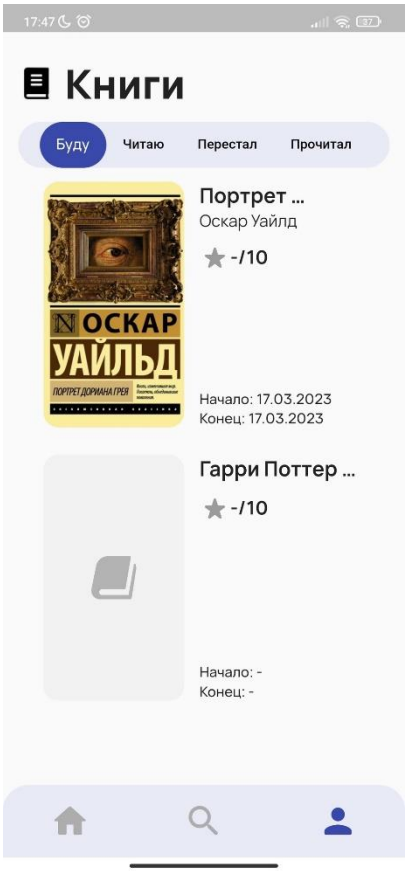

Рисунок 70. Экран с книгами пользователя со статусом «Буду Читать».

Рисунок 71. Экран с книгами пользователя со статусом «Прочитал».

Основным элементом интерфейса на данной странице является панель с вкладками – различными статусами произведений. При переключении между которыми приложение отправляет на сервер GET: /interactions/books/by-status, передавая идентификатор текущего пользователя в header и выбранный статус через query. Получая при этом список книг с запрошенным статусом, где о каждой книге известно: название, обложка, список авторов, пользовательский рейтинг и даты начала и окончания чтения. Если пользователь еще не оценил книгу, то иконка звезды отображается серым, а вместо оценки стоит прочерк. Также, если пользователь еще не начал/закончил читать произведение, то вместо даты ставится прочерк.

Пользователь может просматривать подробную информацию о любой книге из списка на экране его библиотеки. Для этого ему нужно нажать на нее. При выполнении этого действия программа отправляет на сервер GET: /books/books/{bookId} запрос, передавая идентификатор текущего пользователя в header и идентификатор выбранной книги в path. Данное действие так же перенапрявлет пользователя на экран с подробной информацие о книге (подробнее сморите пункт 5. Книги)

При нажатии на пункт меню «Рецензии», описанный в 9.2. или элемент строки статистики, описанный в пункте 8.5. с текстом «Рецензии», пользователя перенаправляют на экран с его отзывами (рисунок 72).

Перед тем, как отобразить весь необходимый интерфейс страницы приложение отправляет GET: /interactions/reviews запрос, передавая идентификатор текущего пользователя в header. В ответ программа получает список отзывов, каждый из которых имеет: название, основной текст, оценку, а также информацию о книге, на которую он был написан (ее обложка, название, авторы).

Если текст рецензии не умещается в 7 строк, то появляется кнопка «Развернуть», которая показывает текст целиком при нажатии. Если повторно нажать на данную кнопку (после первого нажатия текст сменится на «Свернуть»), то текст рецензии снова будет показан не полностью.

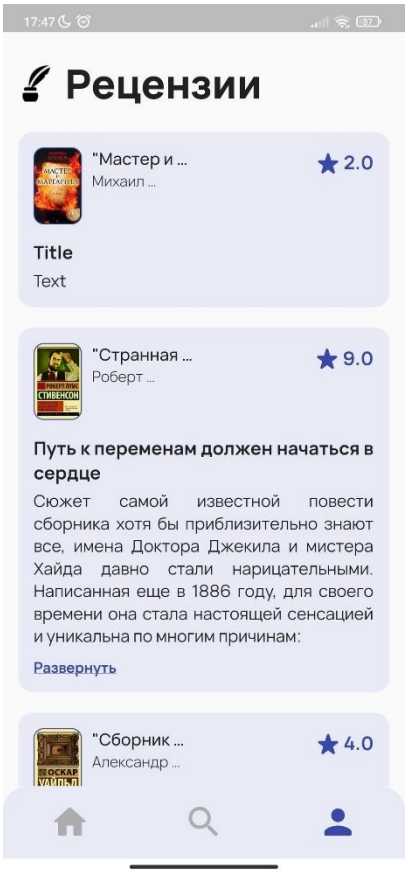

Рисунок 72. Экран с рецензиями пользователя.

Рисунок 73. Экран рецензиями пользователя при удалении.

Пользователь может удалять свои рецензии путем смахивания их влево. При выполнении данного действия задний фон освободившегося пространства будет закрашиваться в красный цвет (рисунок 73). При смахивании, пользователю показывается диалоговое окно с подтверждением удаления (рисунок 74).

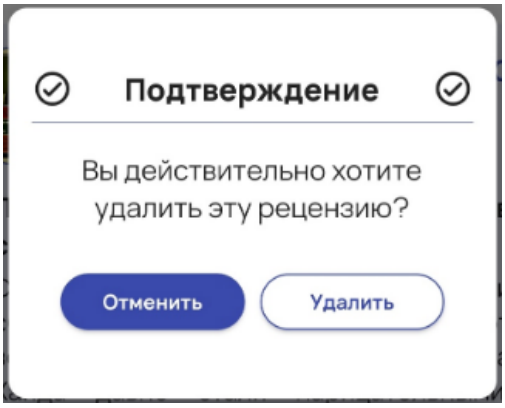

Рисунок 74. Диалоговое окно с подтверждение удаления рецензии.

Если он нажимает на кнопку с текстом «Отменить», то диалоговое окно закрывается, а рецензия остается там же, где и была. Однако если юзер нажимает на кнопку «Удалить», то программа отправляет на сервер DELETE: /interactions/reviews/{reviewId}/remove запрос, передавая идентификатор текущего пользователя в header и идентификатор удаляемой рецензии в path. Как и всегда, если ответом сервера был код 200, то программа удаляет рецензию из списка, в противном случае выводится сообщение об ошибке: «Что-то пошло не так! Рецензия не была удалена.».

При нажатии на пункт меню «Цитаты», описанный в 9.3. или элемент строки статистики, описанный в пункте 8.5. с текстом «Цитаты», пользователя перенаправляют на экран с его цитатами (рисунок 75). Чтобы получить все цитаты пользователя программа отправляет GET: /interactions/quotes запрос, передавая идентификатор текущего пользователя в header.

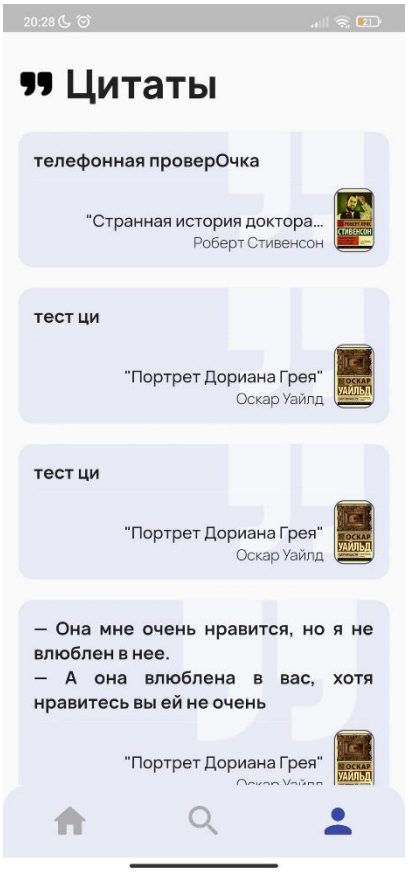

Рисунок 75. Экран с цитатами пользователя.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.02-01 ПЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

При нажатии на пункт меню «Сборники», описанный в 9.4., пользователя перенаправляют на экран с его сборниками (рисунок 76). Чтобы получить все сборники пользователя программа отправляет GET: /shelves/collections/by-user запрос, передавая идентификатор текущего пользователя в header. В качестве ответа программа получает список сборников пользователя, для каждого из которых известны: ID, название, описание и обложка.

Рисунок 76. Экран со сборниками пользователя.

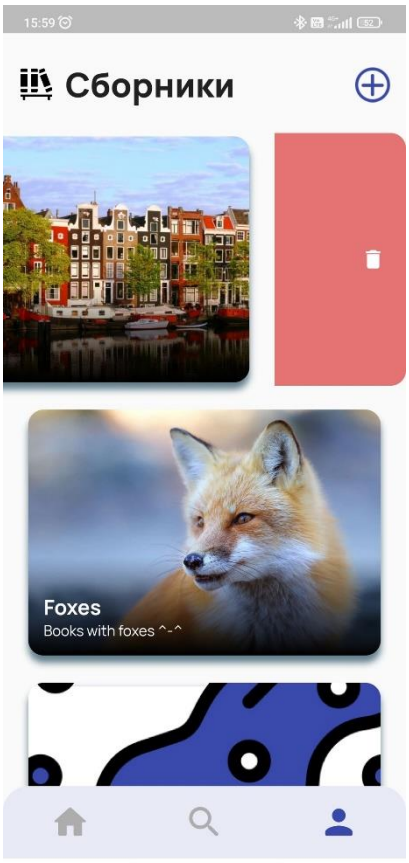

Рисунок 77. Интерфейс при удалении сборника.

Аналогично удалению рецензий, пользователь может удалять сборники, смахивая их влево (рисунок 77). При выполнении данного действия программа показывает пользователю диалоговое окно с подтверждением удаления (рисунок 78). При нажатии «Отмена» окно просто закрывается, не внося изменений с состав сборников, но если пользователь нажмет на кнопку «Удалить», то программа отправит на сервер DELETE: /shelves/collections/{collectionId}/remove запрос, передавая идентификатор текущего пользователя в header и идентификатор удаляемого сборника в path. Если ответом сервера

был код 200, то программа удаляет сборник из списка, в противном случае выводится сообщение об ошибке: «Что-то пошло не так! Сборник не был удален.».

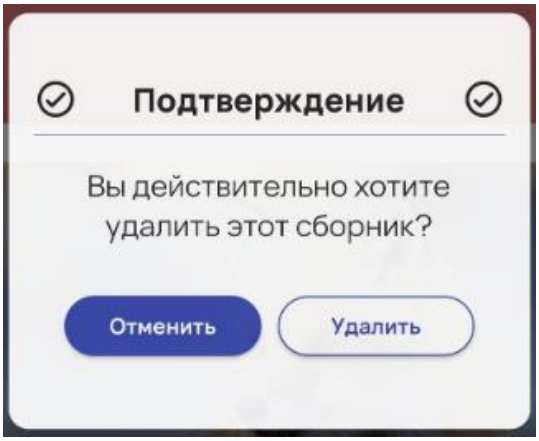

Рисунок 78. Диалоговое окно подтверждения удаления сборника.

Пользователь может просматривать содержимое любого сборника путем нажатия на него. Действия программы при этом аналогичны тем, которые описаны в пункте 3. Коллекции.

Также пользователь может создавать новый сборник, если нажмет на иконку плюса в кружочек напротив лейбла с надписью «Сборники». При совершении данного действия юзеру показывается диалоговое окно с необходимыми для создания сборника полями (рисунок 79). Если пользователь подтверждает создание, нажимая на кнопку «Добавить», программа сначала осуществляет первичную проверку введенных данных – проверяет, что название сборника не пусто. Если же оно отсутствует, то пользователь видит сообщение в формате snackbar’a, что название сборника должно быть не менее двух символов. Если же первичная проверка была успешно пройдена, то программа отправляет на сервер POST: /shelves/collections/create, передавая идентификатор текущего пользователя в header и введенные значение в теле запроса.

Если ответом сервера не будет код 200, то программа выведет сообщение об ошибке: «Что-то пошло не так! Не получилось добавить новый сборник.».

Если же ответом был код двести, но программа обновляет данные, заново вызывая GET: /shelves/collections/by-user и перерисовывая экран. Стоит отметить, что, если пользователь не заполнит поле со ссылкой на изображение, оно присвоится автоматически.

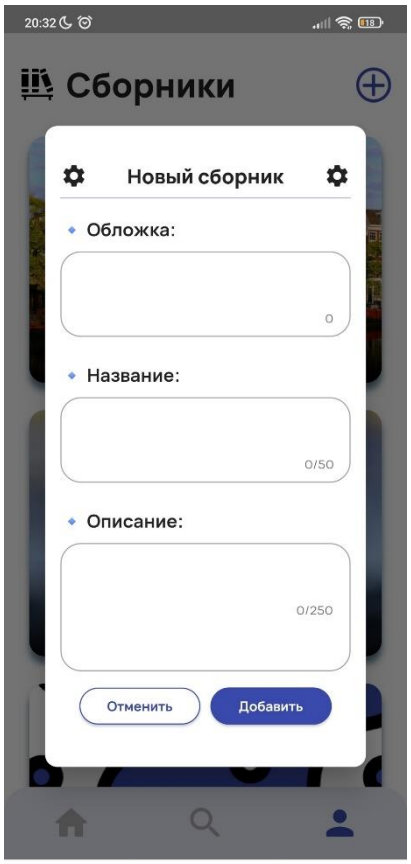

Рисунок 79. Диалоговое окно для создания нового сборника.

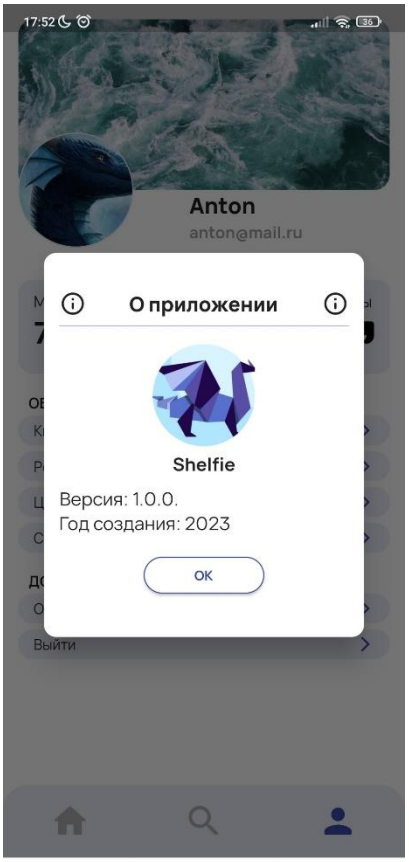

Рисунок 80. Диалоговое окно с информацией о приложении.

При нажатии на пункт меню «О программе», описанный в 9.5 пользователю показывается диалоговое окно с информацией о приложении: его версия, год создание и иконка (рисунок 80).

При нажатии на пункт меню «Выйти», описанный в 9.6 пользователя перенаправляют на экран с авторизации с ранее введенными им параметрами.

10. Нижняя панель навигации

Основная навигация в приложении осуществляется через меню или же нижнюю панель навигации, показанную на рисунке 81. Она представляет из себя панель с тремя элементами:

10.1. Домашняя страница – экран рекомендованных коллекций (смотрите пункт 3. Коллекции)

10.2. Страница поиска – экран поиска книг (смотри пункт 4. Поиск книг)

10.3. Профиль – экран профиля пользователя (смотрите пункт 8. Профиль)

Вся маршрутизация основана на графе возможных переходов и путей в приложении. Для ее написания был использован навигационный пакет «AutoRoute» [9].

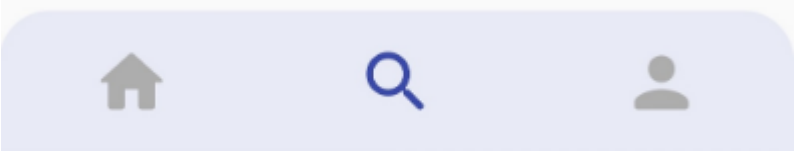

Рисунок 81. Дизайн нижней панели навигации.

Основная идея была в том, что состояние каждого из основных маршрутов, перечисленных в пунктах 10.1. – 10.3. должно было сохраняться. При повторном нажатии на элемент меню, маршрут «обновляется», возвращаясь на корневой экран.

11. Экран ошибок и загрузки

В случае, если у пользователя нет подключения к интернету на любом из этапов пользования приложением, ему показывается экран ошибки, потому что программа не может корректно функционировать, т. к. практически на каждом этапе нам нужна информация из базы данных, а получить ее без запросов к серверу невозможно. Так же данный экран, но с другим сообщением показывается, если полученный от сервера ответа - 500 (Рисунок 82). Под изображением обычно выводится сообщение ошибки.

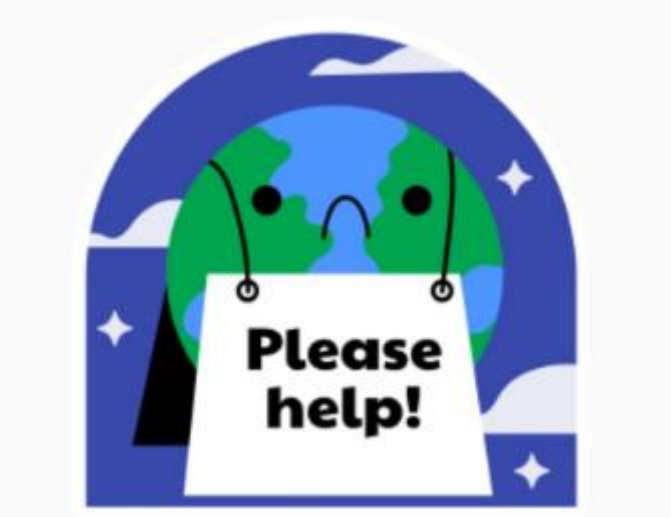

Рисунок 82. Изображение при возникновении ошибки.

Если же программа ожидает ответа от сервера, чтобы отрисовать основные элементы интерфейса на экране, пользователь видит экран загрузке, показанный на рисунке 83.

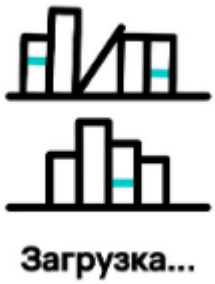

Рисунок 82. Изображение при загрузке данных.

3.5. Описание и обоснование выбора метода организации входных и выходных данных

3.5.1. Описание организации входных и выходных данных

Входные данные приложения – различные действия пользователя при взаимодействии с интерфейсом, вводимая им информация: строки, числа и прочие данные, а также информация, которую мы получаем посредством HTTP-запросов.

Выходные данные приложения – это отображаем интерфейс и HTTP-запросы серверу.

3.5.2. Обоснование выбора метода организации входных и выходных данных

Описанные выше входные данный были выбраны так как они являются оптимальным способом получения информации от пользователя для дальнейшей работы.

Благодаря выходным данным в формате интерфейса пользователю будет легче ориентироваться в приложении, а HTTP-запросы способствуют тому, что в приложении всегда будут отображаться новейшие данные.

3.6. Описание и обоснование выбора состава технических и программных средств

3.6.1. Состав технических и программных средств

3.6.1.1. Состав технических средств

Для функционирования клиентской части приложения требуется мобильное устройство, оснащенное следующими техническими компонентами:

1) Дисплей с минимальным разрешением 720 × 1280 пикселей;
2) Операционная система: Android 10 или более поздняя версия Android, или iOS 13 или более поздняя версия iOS;
3) Минимальный объем оперативной памяти – 2 Гб;
4) Минимальный объем свободной памяти устройства – 150 Мб.
5) Сенсорный экран
6) Подключение к сети Интернет
7) Доступ к камере

3.6.1.2. Состав программных средств

1) Dart версии 2.15.1 или выше
2) Flutter Framework версии 2.14.4 или выше

3.6.2. Обоснование выбора технических и программных средств

Для разработки продукта использовался Flutter Framework так как он позволяет писать кроссплатформенные приложения, для которых на обоих платформах будет одинаковый пользовательский интерфейс. Так как все приложения написанные с Flutter Framework компилируются в машинный код, который использует механизмы визуализации, встроенные в C и C++, приложения получаются очень быстрыми и высокопроизводительными.

Язык программирования Dart был выбран, потому что изначально базировался как инструмент для создания клиентских приложений. Поэтому он оптимизирован под разработку пользовательского интерфейса.

4. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая ценность

Расчет ориентировочной экономической ценности клиентской части мобильного приложения «Shelfie» не предусмотрен.

Расчет экономической ценности мобильного приложения «Shelfie» в целом приведен в настоящем Техническом задании «Клиентская часть мобильного приложения «Shelfie» для трекинга и рекомендации книг».

4.2. Предполагаемая потребность

Данное приложение будет использоваться как любителями книг для удобного отслеживания прочитанных произведений и поиска новых творений, так и людьми, которые только начинают вырабатывать у себя привычку к чтению.

Приложение собирает статистику по пользователям, необходимую для формирования четкого понимания о книге: сколько времени на нее уйдет, насколько популярной она является среди читателей и т. д. Так же в «Shelfie» предусмотрены сгенерированные автоматически подборки, которые отлично подойду для людей, которые совсем не знают, что им почитать. Всё это значительно упростит поиск новой литературы для пользователей.

4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

Помимо перечисленных в настоящем Техническом задании «Клиентская часть мобильного приложения «Shelfie» для трекинга и рекомендации книг» преимуществ, также можно выделить то, что приложение нацелено именно на мониторинг чтения, поэтому оно имеет удобную фильтрацию и поиск, а также статистику, собранную по другим пользователям. Эти преимущества помогут читателям находить именно те книги, которые им действительно будут интересны.

СПИСОК ИСТОЧНИКОВ

1) ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2) ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3) ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4) ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5) ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8) ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9) «AutoRoute» [Электронный ресурс] – навигационный пакет Flutter – Дата обращения: 20.03.2023 – Режим доступа: свободный, ссылка: https://pub.dev/packages/auto_route.
10) «bar_code_scanner» [Электронный ресурс] – плагин для приложений Flutter, который добавляет поддержку сканирования штрих-кодов на Android и iOS. – Дата обращения: 30.03.2023 – Режим доступа: свободный, ссылка: https://pub.dev/packages/flutter_barcode_scanner#flutter_barcode_scanner
11) REST API [Электронный ресурс] – архитектурный стиль взаимодействия компонентов распределённого приложения в сети – Дата обращения: 30.03.2023 – Режим доступа: свободный, ссылка: https://ru.wikipedia.org/wiki/REST
12) http – Flutter библиотека для выполнения HTTP-запросов – Дата обращения: 02.04.2023 – Режим доступа: свободный, ссылка: https://pub.dev/packages/http

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

Таблица 1. Описание и функциональное назначение классов.

КЛАСС	НАЗНАЧЕНИЕ
Book	Отвечает за сущность книги
BookList	Отвечает за преобразование списка книг и их количества
BookQuote	Отвечает за сущность книжной цитаты
BookQuoteList	Отвечает за преобразование списка книжных цитат и их количества
BookReview	Отвечает за сущность рецензии на книгу
BookReviewList	Отвечает за преобразование списка книжных отзывов и их количества
BookStatistic	Отвечает за статистику по книге
BookStatus	Отвечает за возможные статусы книги
Collection	Отвечает за сущность коллекции/сборника
RecomendedCollections	Отвечает за преобразования списка рекомендованных коллекций
Filters	Отвечает за возможный фильтры для поиска
Genre	Отвечает за сущность жанра книги
GenreList	Отвечает за преобразования списка жанров книги
IdInheritedWidget	Отвечает за безопасную передачу идентификационного номер пользователя во всем приложении
Statistic	Отвечает за сущность статистики пользователя

User	Отвечает за сущность пользователя
UserBook	Отвечает за сущность книг у пользователя в профиле
UserBookList	Отвечает за преобразование списка книг пользователя
UserCollection	Отвечает за сущность пользовательских сборников
UserCollectionList	Отвечает за преобразование пользовательских сборников
UserQuote	Отвечает за сущность пользовательских цитат
UserQuoteList	Отвечает за преобразование списка пользовательских цитат
UserReview	Отвечает за рецензии пользователя
UserReviewList	Отвечает за преобразования списка пользовательских рецензий
BottomBarBubble	Отвечает за анимацию круга в нижней панели навигации
BottomBarItem	Отвечает за сущность элемента меню
DialogButton	Отвечает за сущность кнопки в диалоговых окнах
RoundedButton	Отвечает за сущность кнопки с определенным дизайном
ScanButton	Отвечает за кнопку сканирования ISBN
InputTextField	Отвечает за дизайн текстового поля
PasswordTextField	Отвечает за дизайн текстового поля для пароля
AddCollectionCard	Отвечает за дизайн элемента списка при добавлении книги в коллекцию
QuoteCard	Отвечает за дизайн карточки цитаты на странице с информацией о книге

ReviewCard	Отвечает за дизайн карточки с информацией об отзыве на странице с подробной информацией о книге
UserBookCard	Отвечает за дизайн карточки с книгой в профиле пользователя
UserQuoteCard	Отвечает за дизайн карточки с цитатой в профиле пользователя
UserReviewCard	Отвечает за дизайн карточки с отзывом в профиле пользователя
AboutDialog	Отвечает за создание диалогового окна с информацией о приложении
AddBookDialog	Отвечает за создание диалогового окна для добавления книги в сборник
AddCollectionDialog	Отвечает за создание диалогового окна для добавления коллекции
AddQuoteDialog	Отвечает за создание диалогового окна для добавления цитаты
AddReviewDialog	Отвечает за создание диалогового окна
ChangeAvatarDialog	Отвечает за создание диалогового окна для изменения аватара профиля
ChangeBannerDialog	Отвечает за создание диалогового окна для изменения баннера профиля
ConfiramtionDialog	Отвечает за создание диалогового окна для подтверждения действия
FiltersDialog	Отвечает за создание диалогового окна для выбора фильтров при поиске книг
NothingFoundDialog	Отвечает за создание диалогового окна с оповещением о том, что данные не были найдены
ErrorWidget	Отвечает за создание виджета с сообщением при ошибке
LoadingWidget	Отвечает за создание виджета загрузки
RatingWidget	Отвечает за создание виджета с рейтингом книги
StatusWidget	Отвечает за создание виджета со статусом книги

Constants	Отвечает за хранение основных констант приложения, как, например, цвета и адрес хоста
ImageConstants	Отвечает за хранение основных ссылок на изображения
BookStatisticTabBar	Отвечает за создание виджета с вкладками с цитатами и отзывами по книге
StatusTabBar	Отвечает за создание виджета с вкладками для изменения статуса произведения
BookDesc	Отвечает за создание виджета с описанием книги
BookMainInfo	Отвечает за создание виджета с основной информацией о книге
StatisticRow	Отвечает за создание виджета строки основной статистики по книге
BookInfoPage	Отвечает за создание страницы с подробной информацией о книге
BookPage	Отвечает за создание страницы с книгами для коллекции
CollecionCard	Отвечает за создание интерфейса карточки коллекции
CollectionPage	Отвечает за создание страницы коллекций
FilterList	Отвечает за создание списка для выбора элементов фильтрации
FilterText	Отвечает за дизайн текста заголовков категорий фильтрации
Slider	Отвечает за создание ползунка для выбора минимального рейтинга при фильтрации
HomePage	Отвечает за маршрутизацию в приложении
LogInPage	Отвечает за создания экрана входа в приложение
Menu	Отвечает за создание меню в профиле
MenuListItem	Отвечает за создание элемента меню профиля

StatCard	Отвечает за интерфейс карточки статистики по пользователю
StatisticRow	Отвечает за создание строки статистики в профиле пользователя
ProfileHead	Отвечает за интерфейс шапки профиля
BooksTabBar	Отвечает за интерфейс с вкладками для выбора книг по статусам в профиле пользователя
UserBooksPage	Отвечает за корневую страницу профиля пользователя
UserCollectionPage	Отвечает за создание страницы со сборниками пользователя
UserQuotePage	Отвечает за создание страницы с цитатами пользователя
UserReviewPage	Отвечает за создание страницы с отзывами пользователя
ProfilePage	Отвечает за создание страницы профиля пользователя
ListBookCard	Отвечает за создание интерфейса для отображения книг в профиле пользователя
SearchPage	Отвечает за создание страницы поиска
SignUpPage	Отвечает за создание страницы регистрации
Main	Точка входа в приложение

ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ

Таблица 2. Описание полей класса Book.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_id	**int**	Идентификатор пользователя
_title	**String**	Название книги
_authors	**List<String>**	Список всех авторов книги
_coverImageUrl	**String**	Прямая ссылка на обложку произведения
_genres	**GenresList**	Список жанров книги
_rating	**double?**	Средний рейтинг произведения
_userRating	**int?**	Рейтинг, которые пользователь поставил книге
_status	**BookStatus**	Текущий статус книги в пользовательской библиотеке
_descriprion	**String?**	Описание произведения
_language	**String?**	Язык, на котором был написан оригинал книги
_ageRestriction	**String?**	Возрастные ограничения произведения
_statistics	**BookStatistics**	Статистика по книге

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
getRating	**double?**	**-**	Проверяет, что у книги есть рейтинг и возвращает его, округленным до двух знаков после запятой
allInfoFromJson	**Book**	**dynamic json**	Преобразует все данные о книге из json в нужные типы и возвращает созданный объект
fromJson	**Book**	**dynamic json**	Преобразует данные необходимые для общей информации о книге из json в нужные типы и возвращает созданный объект

Таблица 3. Описание полей класса BookList.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
count	**int**	Общее количество книг
foundBooks	**List<Book>**	Список полученных книг
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**BookList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 4. Описание полей класса BookQuote.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_id	**int**	Идентификатор цитаты
_quoteText	**String**	Текст цитаты
_isSaved	**bool**	True|False параметр, сохранена ли цитата у пользователя
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
reverseQuoteSaved	**void**	**-**	Меняет параметр _isSaved на противоположный
fromJson	**BookQoute**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 5. Описание полей класса BookUoteList.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
count	**int**	Общее количество всех цитат книги
quotes	**List<BookQuote>**	Список текущих цитат
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ

fromJson	**BookQouteList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 6. Описание полей класса BookReview.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
_id	**int**	Идентификатор рецензии
_reviewAuthor	**User**	Автор рецензии
_reviewText	**String**	Текст рецензии
_reviewTitle	**String**	Название (заголовок) рецензии
_rating	**double**	Оценка книги, которую поставил рецензент
МЕТОДЫ		

ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**BookReview**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 7. Описание полей класса BookReviewList.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
count	**int**	Общее количество всех рецензий на книгу

reviews	**List<BookReview>**	Список текущих рецензий
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
fromJson	**BookReviewList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 8. Описание полей класса BookStatistic.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_readingTime	**String**	Среднее время чтение книги
_readCount	**int?**	Количество пользователей, которые прочли книгу
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
fromJson	**BookStatistic**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 9. Описание полей класса Collection.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_id	**int**	Идентификатор коллекции
_imageUrl	**String**	Обложка коллекции

_name	**String**	Название коллекции
_description	**String**	Описание коллекции
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**Collection**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 10. Описание полей класса RecomendedCollections.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
collections	**List<Collection>**	Список рекомендованных всем пользователям коллекций
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**RecommendedCollections**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 11. Описание полей класса Filters.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
languages	**List<String>**	Список всех возможных языков, на которых написаны книги в базе

genres	**List<String>**	Список всех доступных жанров
agerestrinctions	**List<String>**	Список всех возможных возрастных ограничений

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**Filters**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 13. Описание полей класса Genre.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
_id	**int**	Идентификатор жанра
_genreName	**String**	Название жанра

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**Genre**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 13. Описание полей класса GenreList.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
genres	**List<Genre>**	Список всех жанров
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
fromJson	**GenreList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 14. Описание полей класса IdInheritedWidget.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
id	**int**	Наследуемый идентификатор
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
updateShouldNotify	**bool**	**IdInheritedWidget oldWidget**	Определяет нужно ли оповещать

Таблица 15. Описание полей класса Statistic.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_bookCount	**int**	Количество книг в библиотеке пользователя

_reviewCount	**int**	Количество написанных рецензий
_quoteCount	**int**	Количество написанных цитат
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**Statistic**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 16. Описание полей класса User.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
_id	**int**	Идентификатор пользователя
_name	**String**	Имя (логин) пользователя
_email	**String**	Почта пользователя
_profileImageUrl	**String**	Изображение профиля пользователя
_bannerImageUrl	**String**	Баннер (шапка) профиля
_statistics	**Statistic**	Статистика пользователя
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**

setAvatar	**void**	**String newAvatar**	Устанавливает новый аватар профилю
setBanner	**void**	**String newBanner**	Устанавливает новый баннер
setId	**void**	**int id**	Устанавливает пользователю идентификатор
userIdFromJson	**User**	**dynamic json**	Преобразует данные об идентификаторе пользователя из json в нужные типы и возвращает созданный объект
userInfoFromJson	**User**	**dynamic json**	Преобразует данные о пользователе для рецензии из json в нужные типы и возвращает созданный объект
fromJson	**User**	**dynamic json**	Преобразует все данные о пользователе из json в нужные типы и возвращает созданный объект

Таблица 17. Описание полей класса UserBook.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_id	**int**	Идентификатор книги
_title	**String**	Название книги
_authors	**List<String>**	Список авторов книги
_coverImageUrl	**String**	Прямая ссылка на изображение обложки книги

_userRating	**int?**	Рейтинг, который пользователь поставил книге
_startTime	**String?**	Время, когда пользователь начал читать книгу
_endTime	**String?**	Время, когда пользователь закончил читать книгу

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**UserBook**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 18. Описание полей класса UserBookList.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
count	**int**	Общее количество книг в библиотеке пользователя
allBooks	**List<UserBook>**	Список текущих книг пользователя

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**UserBookList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 19. Описание полей класса UserCollectiont.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
_id	**int**	Идентификатор коллекции
_name	**String**	Название коллекции
_isInCollection	**bool**	Содержит ли книга в коллекции
МЕТОДЫ		

ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**UserCollection**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 20. Описание полей класса UserCollectionList.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
allCollections	**List<UserCollection>**	Список всех коллекций пользователя
МЕТОДЫ		

ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**UserCollectionLict**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 21. Описание полей класса UserQuote.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_id	**int**	Идентификатор цитаты
_bookId	**int**	Идентификатор книги, из которой цитата
_bookTitle	**String**	Название книги, из которой цитата
_bookCoverUrl	**String**	Прямая ссылка на обложку книги, из которой цитата
_bookAuthors	**List< String>**	Список авторов книги, из которой цитата
_text	**String**	Текст цитаты

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
getAuthors	**String**	**-**	Преобразует список авторов в строку
fromJson	**UserQuote**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 22. Описание полей класса UserQuoteList.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
count	**int**	Общее количество цитат пользователя
quotes	**List<UserQuote>**	Список текущих цитат
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
fromJson	**UserQuoteList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 23. Описание полей класса UserReview.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
ПОЛЯ		
_id	**int**	Идентификатор рецензии
_reviewTitle	**String**	Название рецензии
_text	**String**	Основной текст рецензии
_rating	**double**	Рейтинг, который пользователь поставил произведению
_bookId	**int**	Идентификатор книги, по которой рецензия

_bookTitle	**String**	Название книги, по которой рецензия
_bookCoverUrl	**String**	Прямая ссылка на обложку книги, по которой рецензия
_bookAuthors	**List< String>**	Список авторов книги, по которой рецензия

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
getAuthors	**String**	-	Преобразует список авторов в строку
fromJson	**UserReview**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 24. Описание полей класса UserReviewList.

ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
ПОЛЯ		
count	**int**	Общее количество рецензий
reviews	**List<UserReview>**	Список текущих рецензий пользователя

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
fromJson	**UserReviewList**	**dynamic json**	Преобразует данные из json в нужные типы и возвращает созданный объект

Таблица 25. Описание полей и методов BottomBarBubble.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
selectedIndex	**int**	Текущий индекс меню
height	**double**	Высота панели меню
bubbleSize	**double**	Размер шарика для анимации
color	**Color**	Цвет панели меню
onSelect	**ValueChanged<int>?**	Функция, которая будет вызываться при выборе нового элемента меню
items	**List<BottomBarItem>**	Список элементов меню
onSelectAgain	**ValueChanged<int>?**	Функция, которая будет вызываться при повторном нажатии на элемент меню

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает анимацию для меню

Таблица 4. Описание полей и методов ButtomBarItem.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
iconData	**IconData?**	Иконка элемента меню

iconSize	**double**	Размер иконки элемента меню
label	**String?**	Текст элемента меню
labelTextStyle	**TextStyle?**	Стиль текста элемента меню
labelMarginTop	**double**	Отступ текста элемента меню
iconBuilder	**BottomBarMatuIconBuilder?**	Аниматор элемента меню

Таблица 27. Описание полей и методов DialogButton.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
text	**String**	Текст на кнопке
reverse	**bool**	Нужно ли менять цвета кнопки на противоположные
press	**VoiceCallback**	Метод, который будет вызываться при нажатии
isAsync	**bool**	Является ли метод асинхронным или нет

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает контейнер для кнопки
newElevatedButton	**Widget**	**BuildContext context**	Создает кнопку по заданным параметрам

Таблица 28. Описание полей и методов RoundedButton.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
text	**String**	Текст на кнопке
press	**VoiceCallback**	Метод, который будет вызываться при нажатии

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает контейнер для кнопки
newElevatedButton	**Widget**	**BuildContext context**	Создает кнопку по заданным параметрам

Таблица 29. Описание полей и методов ScanButton.

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает дизайн кнопки
getBookByISBN	**Future<Book>**	**int id, String isbn, BuildContext context**	Отправляет на сервер запрос с поиском книги по ее ISBN
barcodeScan	**Future<void>**	**BuildContext context**	Анализирует отсканированный ISBN

Таблица 30. Описание полей и методов InputTextField.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
height	**double**	Высота текстового поля
maxLen	**int**	Максимальное количество строк текстового поля
onChanged	**ValueChanged<String>**	Метод, вызывающийся при изменение значения в текстовом поле

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает текстовое поле по заданным настройкам

Таблица 31. Описание полей и методов PasswordTextField.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_showEye	**bool**	Показывается ли сейчас иконка с глазом или нет
_passwordEncrypted	**bool**	Скрывается ли сейчас пароль или нет
_password	**String**	Текст, введенный в текстовое поле

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает текстовое поле для пароля

Таблица 32. Описание полей и методов AddCollectionCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
collection	**UserCollection**	Информация о сборнике пользователя
userId	**int**	Идентификатор текущего пользователя
bookId	**int**	Идентификатор текущей книги
showFlag	**bool**	Находится ли книга сейчас в коллекции или нет

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран с информацией о текущей теме
addTooCollection	**Future<void>**	**-**	Отправляет на сервер запрос на добавление книги в сборник
removeFromCollection	**Future<void>**	**-**	Отправляет на сервер запрос об удалении книги из сборника

Таблица 33. Описание полей и методов QuoteCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
quote	**BookQuote**	Текущая цитата
showFlag	**bool**	Показывать ли всю цитату целиком или только первые 7 строк

id	**int**	Идентификатор текущего пользователя

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран с информацией о текущей теме
saveQuote	**Future<void>**	**-**	Отправляет на сервер запрос на сохранение цитаты
unsaveQuote	**Future<void>**	**int id**	Отправляет на сервер запрос на удаление цитаты
onQuoteClick	**Future<void>**	**int id**	Определяет какой метод должен быть вызван при нажатии на кнопку

Таблица 34. Описание полей и методов ReviewCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_currentTopic	**Future<TopicInfo>**	Текущая тема

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран с информацией о текущей теме
getTopic	**Future<TopicInfo>**	**-**	Отправляет на сервер запрос для получения информации о текущей теме

Таблица 35. Описание полей и методов UserBookCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
press	**VoiceCallback**	Метод, вызываемый при нажатии на кнопку в диалоговом окне
book	**UserBook**	Текущая книга

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно
ratingWidget	**Widget**	-	Создает виджет с рейтингом книги
timeWidget	**Widget**	**Size size**	Создает виджет с датами начала и конца чтения книги

Таблица 36. Описание полей и методов UserQuoteCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
quote	**UserQuote**	Текущая цитата
showFlag	**bool**	Показывать ли текст целиком или только первые 7 строк

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает интерфейс карточки с цитатой

Таблица 37. Описание полей и методов UserReviewCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
review	**UserReview**	Текущая рецензия
showFlag	**bool**	Показывать ли текст целиком или только первые 7 строк

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает дизайн карточки с рецензией
deleteReview	**Future<void>**	**int id**	Отправляет на сервер запрос об удалении рецензии

Таблица 38. Описание полей и методов BookStatisticsTabBar.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_id	**int**	Идентификатор текущего пользователя
_quoteList	**BookQuoteList**	Список цитат книги
_reviewList	**BookReviewList**	Список рецензий на книгу
_tabController	**TabController**	Контроллер вкладок цитат и рецензий
_book	**Book**	Текущая книга
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
build	**Widget**	**BuildContext context**	Создает дизайн панели вкладок
getReviewList	**Future<BookReviewList>**	**int id, int take, int skip**	Отправляет на сервер запрос на получение всех рецензий книги
getQuoteList	**Future<BookQuoteList >**	**int id, int take, int skip**	Отправляет на сервер запрос на получение всех цитат книги
onQuoteGoBack	**Future<FutureOr>**	**dynamic value**	Обновляет лист с цитатами при возвращении назад
onReviewGoBack	**Future<FutureOr >**	**dynamic value**	Обновляет лист с рецензиями при возвращении назад

Таблица 39. Описание полей и методов AddBoolDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
userId	**int**	Идентификатор пользователя
bookId	**int**	Идентификатор текущей книги

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно для добавления книги в коллекции
getUserCollection	**Future<List<UserCollection>>**	**-**	Отправляет на сервер запрос о получении всех сборников пользователя

Таблица 40. Описание полей и методов AddCollectionDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
userId	**int**	Идентификатор текущего пользователя
title	**String**	Название будущей коллекции
coverUrl	**String**	Обложка для будущей коллекции
desc	**String**	Описание будущей коллекции

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно для создания нового сборника
createCollection	**Future<void>**	**-**	Отправляет на сервер запрос с созданием новой коллекции
checkRestrictions	**bool**	**BuildContext context**	Проверяет, что введенные пользователем данные, соответствуют требованиям

Таблица 41. Описание полей и методов AddQuoteDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_book	**Book**	Текущая книга

_id	**int**	Идентификатор текущего пользователя
_text	**String**	Текст будущей цитаты
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно для создания цитаты
addQuote	**Future<void>**	**BuildContext context**	Отправляет на сервер запрос на создание новой цитаты

Таблица 42. Описание полей и методов AddReviewDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_book	**Book**	Текущая книга
_id	**int**	Идентификатор текущего пользователя
_text	**String**	Текст будущей рецензии
_rating	**int**	Название будущей рецензии
_title	**String**	Пользовательская оценка книги
МЕТОДЫ		
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**

build	**Widget**	**BuildContext context**	Создает диалоговое окно для создания рецензии
addReview	**Future<void>**	**BuildContext context**	Отправляет на сервер запрос о создании новой рецензии
checkRestrictions	**bool**	**BuildContext context**	Проверяет, что введенные пользователем данные, соответствуют требованиям

Таблица 43. Описание полей и методов ChangeAvatarDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
userId	**int**	Идентификатор текущего пользователя
newAvatar	**String**	Прямая ссылка на новое изображение профиля

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно для изменения аватара
setAvatar	**Future<void>**	**String newAvatar**	Отправляет на сервер запрос с изменением аватара

Таблица 44. Описание полей и методов ChangeBannerDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
userId	**int**	Идентификатор текущего пользователя

newBanner	**String**	Прямая ссылка на новое изображение баннера

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	
setBanner	**Future<void>**	**String newBanner**	Отправляет на сервер запрос с изменением баннера

Таблица 45. Описание полей и методов FiltersDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
genres	**FilterList**	Список всех доступных жанров
countries	**FilterList**	Список всех доступных языков
restrictions	**FilterList**	Список всех доступных возрастных ограничений
slider	**SliderWidget**	Виджет для настройки минимального рейтинга

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно для настройки фильтров
getFilters	**Future<Filters>**	**-**	Отправляет на сервер запрос на получение всех доступных параметров фильтрации

Таблица 46. Описание полей и методов NothingFoundDialog.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
text	**String**	Текст диалогового окна
title	**String**	Заголовок диалогового окна
imageUrl	**String**	Изображение (ссылка на gif)

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает диалоговое окно

Таблица 47. Описание полей и методов StatusWidget.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
bookId	**int**	Идентификатор текущей книги
bookState	**BookStatus**	Текущий статус книги

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает виджет
changeStatus	**Future<void>**	**String status**	Отправляет на сервер запрос на изменение статуса

getStateIcon	**String**	**BookStatus status**	Возвращает иконку по статусу
getStringStat	**String**	**BookStatus status**	Возвращает текстовое представление статуса

Таблица 48. Описание полей и методов SignUpPage: Body.

ПОЛЯ			
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**	
_email	**String**	Почта для регистрации пользователя	
_name	**String**	Логин для регистрации пользователя	
passwordField	**PasswordTextField**	Пароль для регистрации пользвоателя	
МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран регистрации
addUser	**Future<void>**	**-**	Отправляет на сервер запрос на создание нового пользователя
loginUser	**Future<int>**	**-**	Отправляет на сервер запрос на авторизацию нового пользователя

Таблица 49. Описание полей и методов SearchPage.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_id	**int**	Идентификатор текущего пользователя
_query	**String**	Строка, введённая в поиск
_languages	**List<String>**	Список выбранных для фильтрации языков
_genres	**List<String>**	Список выбранных для фильтрации жанров
_ageRestrictions	**List<String>**	Список выбранных для фильтрации возрастных ограничений
_minRating	**int**	Выбранный для фильтрации минимальный рейтинг
dialog	**FilterDialog**	Диалоговое окно с фильтрацией

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран поиска
searchBooks	**Future<List<Book>**	-	Отправляет на сервер запрос для поиска книг по заданным параметрам
onGoBack	**FutureOr**	**dynamic value**	Обновляет страницу при возвращении назад
setFilters	**FutureOr**	**dynamic value**	Применяет введенные фильтры

searchField	**Widget**	**Size size**	Создает строку поиска

Таблица 50. Описание полей и методов ProfilePage.

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает страницу профиля
getUser	**Future<User>**	**int id**	Отправляет на сервер запрос на получение информации о пользователе
onGoBack	**FutureOr**	**dynamic value**	Обновляет страницу при возвращении назад
buildBody	**Widget**	**User user**	Создает основной интерфейс страницы профиля

Таблица 51. Описание полей и методов UserReviewPage.

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает интерфейс страницы с пользовательскими рецензиями
getUserReviews	**Future<userReviewList>**	**int id**	Отправляет на сервер запрос на получение всех пользовательских рецензий

Таблица 52. Описание полей и методов UserQuotePage.

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает интерфейс страницы с пользовательскими цитат
getUserQuotes	**Future<userQuoteList>**	**int id**	Отправляет на сервер запрос на получение всех пользовательских цитат

Таблица 53. Описание полей и методов UserCollectionPage: Body.

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает интерфейс страницы с пользовательскими сборников
getUserCollections	**Future<userCollectionsList>**	**int id**	Отправляет на сервер запрос на получение всех пользовательских сборников
onGoBack	**FutureOr**	**dynamic value**	Обновляет страницу при возвращении назад
deleteCollection	**Future<void>**	**int userId, int colId**	Отправляет на сервер запрос на удаление сборника
buildCard	**Widget**	**BuildContext context**	Создает карточку со сборников
deleteBackGroundItem	**Widget**	**-**	Создает интерфейс заднего фона при удалении сборника
drawHead	**Widget**	**BuildContext context**	Создает интерфейс заголовка страницы

Таблица 54. Описание полей и методов BooksTabBar.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_tabController	**TabController**	Контроллер вкладок
tabText	**List<String>**	Заголовки вкладок

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран с книгами пользователя
getAllBooks	**Future<List<UserBook>>**	**int id, int take, int skip, String stat**	Отправляет на сервер запрос на получение всех книг пользователя по заданному статусу
onGoBack	**FutureOr**	**dynamic value**	Обновляет страницу при возвращении назад
getStatusFromTab	**String**	**int id**	Возвращает статус книги по индексу вкладки
getStateIcon	**int**	**BookStatus status**	Возвращает индекс вкладки по статусу книги

Таблица 55. Описание полей и методов LogInPage.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_email	**String**	Почта пользователя

passwordField	**PasswordTextField**	Пароль для регистрации пользователя

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран аторизации
loginUser	**Future<int>**	**-**	Отправляет на сервер запрос на авторизацию нового пользователя

Таблица 56. Описание полей и методов HomePage.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
curContext	**BuildContext**	Текущий контекст приложения
_routes	**List<PageRoueInfo<dynamic>>**	Список доступных маршрутов для меню

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает нижнюю панель навигации
_selectTab	**void**	**int index**	Определяет действия при нажатии на элемент меню

Таблица 57. Описание полей и методов CollectionPage.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
_futureCollection	**Future<List<Collection>>**	Будущий список доступных коллекций

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран с рекомендованными коллекциями
getCollections	**Future<List<Collection>>**	**-**	Отправляет на сервер запрос на получение рекомендованных коллекций

Таблица 58. Описание полей и методов BookPage.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
collectionId	**int**	Идентификатор текущей коллекции
collectionName	**String**	Имея текущей коллекции

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает экран с книгами в выбранной коллекциии
getCollectionBooks	**Future<List<Book>**	**int id**	Отправляет на сервер запрос на получение книг

onGoBack	**FutureOr**	**dynamic value**	Обновляет страницу при возвращении назад

Таблица 59. Описание полей и методов BookInfoPage.

ПОЛЯ			
ИМЯ	**ТИП**		**НАЗНАЧЕНИЕ**
bookId	**int**		Идентификатор выбранной книги
МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает страницу с подробной информацией о выбранной книге
getAllBookInfo	**Future<Book>**	**int id**	Отправляет на сервер запрос на получение подробной информации о выбранной книге

Таблица 60. Описание полей и методов ProfileHead.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
id	**int**	Идентификатор текущего пользователя
banDialog	**ChangeBannerDialog**	Диалоговое окно для смены баннера профиля
avdialog	**ChangeAvatarDialog**	Диалоговое окно для смены аватара профиля
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
build	**Widget**	**BuildContext context**	Создает виджет с аватаром и баннером профиля
changeBanner	**FutureOr**	**dynamic value**	Изменяет баннер профиля и обновляет страницу
changeAvatar	**FutureOr**	**dynamic value**	Изменяет аватар профиля и обновляет страницу
showChangeAvatarDialog	**void**	**-**	Показывает диалоговое окно для изменения аватара
showChangeBannerDialog	**void**	**-**	Показывает диалоговое окно для изменения баннера

Таблица 61. Описание полей и методов StatCard.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
press	**VoiceCallBack**	Метод, который вызывается при нажатии на элемент строки статистики
test	**String**	Текст, отображаемый на строке статистики
countNum	**int**	Счетчик, отображаемый на строке статистики
iconName	**String**	Иконка, отображаемая на строке статистики
МЕТОДЫ		

ИМЯ	ТИП ВОЗВРАЩАЕМОГО	АРГУМЕНТЫ	НАЗНАЧЕНИЕ
build	**Widget**	**BuildContext context**	

Таблица 62. Описание полей и методов BookMainInfo.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
book	**Book**	Текущая книга

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает виджет с основной информацией о книге
infoText	**Text**	**String text-**	Создает текст с описанием
tenStarWidget	**Widget**	**int rating**	Создает виджет рейтинга пользователя
starIcon	**GestureDetector**	**bool filled**	Создает иконку звезды
retingWidget	**Widget**	**double? rating**	Создает виджет среднего рейтинга произведения
genreWidget	**Widget**	**String genre**	Создает виджет жанра произведния

Таблица 63. Описание полей и методов BookDesc.

ПОЛЯ		
ИМЯ	**ТИП**	**НАЗНАЧЕНИЕ**
descText	**String**	Основной текст описания
descTextShowFlag	**bool**	Флаг, определяющий нужно ли показывать описание целиком или только первые 7 строк.

МЕТОДЫ			
ИМЯ	**ТИП ВОЗВРАЩАЕМОГО**	**АРГУМЕНТЫ**	**НАЗНАЧЕНИЕ**
build	**Widget**	**BuildContext context**	Создает виджет с описание произведения

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					