МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Ведущий разработчик
ООО «ВОРТЕКСКОД»

_____ П.А. Михайлов

“___”_____ 2018 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

“___”_____ 2018 г.

РАЗРАБОТКА ПЛАТФОРМЫ ДЛЯ СОЗДАНИЯ КНИГ С ИНТЕГРИРОВАННЫМ МУЗЫКАЛЬНЫМ СОПРОВОЖДЕНИЕМ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2018.308-034.ВКР

Научный руководитель
кандидат физ.-мат. наук,
доцент кафедры СП
_____С.А. Иванов

Автор работы,
студент группы КЭ-401
_____А.А. Беседин

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова

“___”_____ 2018 г.

Челябинск-2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

В последние несколько десятилетий в нашей жизни происходит бурное развитие контента и его форматов. Сюда входит кинопроизводство, производство сериалов, а с развитием интернета еще дополнились видео площадки, стриминговые площадки и социальные сети. С развитием технологий его потребление стало более доступным. На этом фоне, чтение книг стало не популярным и отошло на второй план. Но все равно остается огромное количество людей, которые предпочитают этот формат.

Звуковое оформление играет очень важную роль при потреблении любого контента: фильмов, сериалов, YouTube-роликов, игр и т.п. Это так же доказывается исследованием, проведенном в кубанском государственном технологическом университете [11]. В этом исследовании говорится о важности музыки в жизни человека, в частности, что она влияет на настроение человека и имеет эмоциональное воздействие. Это самое воздействие и используется при создании контента.

Для подтверждения догадки, был проведен эксперимент, в котором принимали участие обучающиеся группы КЭ-401. В этом эксперименте предлагалось прочитать отрывок главы из книги Анджея Сапковского «Меч предназначения», при этом во время чтения книги включалась музыка. Книга была в жанре роман, поэтому была выбрана мелодичная музыка. Громкость звучания была низкой, чтобы не отвлекать от чтения, но при этом слышать ее мотив на фоне. По результатам тестирования большинству испытуемых понравился такой формат чтения. Так же большинство испытуемые отметили более глубокое вовлечение и степень погружения в события. Из этого можно сделать вывод, что со звуковым сопровождением книгу читать увлекательнее и интереснее, особенно когда мотив соответствует событиям, происходящим в книге.

Таим образом приложение призвано расширить спектр эмоций при чтении книг, что не только понравится любителям чтения, но и так же мо-

жет привлечь людей, которые давно перестали читать или же вовсе не читали книг, потому что считают просмотр видео контента более интересным занятием.

Но создание одного лишь приложения, реализующего задумку, недостаточно. Это должен быть полноценный продукт, который предоставляет пользователю полноценный опыт от использования. Поэтому необходимо разработать полноценную платформу, в которой будет не только приложение, но и место, где пользователь сможет получить такие книги, а также инструментарий для издателей. Такой инструментарий позволит издателям, заинтересовавшимся в предоставлении такого вида контента, с легкостью создать его и сделать доступным для всех пользователей.

Цель и задачи

Целью работы является создание платформы, которая позволит пользователям платформы получать и потреблять специфичный контент, а также создавать его и распространять через данную платформу.

Для достижение поставленной цели необходимо:

1) провести анализ предметной области;
2) провести анализ требований к платформе;
3) разработать архитектуру компонентов платформы;
4) разработать компоненты платформы;
5) протестировать компоненты платформы.

Объем и структура работы

Общий объем работы составляет 45 страниц, основная часть работы содержит 5 глав. Объем библиографии составляет 17 источников.

Краткое содержание работы

В главе «Анализ предметной области» приведены аналогичные проекты и проведен анализ существующих форматов электронных книг.

В главе «Анализ требований» приведены функциональные и нефункциональные требования к компонентам платформы, а также представлена диаграмма прецедентов платформы.

В главе «Проектирование» представлен проект собственного формата книги и диаграмма компонентов платформы с описанием каждого компонента и элемента в нем.

В главе «Реализация» представлен краткий обзор реализации компонентов платформы с обзором используемых инструментов и технологий, а также скриншотами приложений.

В главе «Тестирование» проведено тестирование компонентов платформы и их взаимодействие со стороны пользователя.

В заключении подводятся итоги работы и обозначаются направления для дальнейшего развития проекта.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Целью данной работы является разработка платформы для создания и чтения книг с интегрированным звуковым сопровождением. Разрабатываемая платформа позволит, с одной стороны, создавать и распространять издателям книги с интегрированным музыкальным сопровождением на основе существующего формата (далее по тексту книги, имея в виду такого рода книги). С другой стороны, пользователь сможет получать и потреблять их. Книга позволит читать текст, который будет дополнен музыкальным сопровождением относительного читаемого контекста. Данное сопровождение схоже со звуковым дизайном в играх, где музыка и звуки играют в определенные игровые моменты на основе контекста происходящих событий.

1.2. Анализ существующих реализации проекта

На данный момент существует единственное решение на зарубежном рынке – Booktrack [2]. Оно предоставляет аналогичное решение, которое предоставляет пользователю звуковое сопровождение читаемому тексту. Определение положения читаемого текста происходит на основе выставленного пользователем значения скорости чтения, которое может быть подстроено автоматически. По возможности воспроизведение можно поставить на паузу. Если же указатель на читаемый текст не совпадает с текущим читаемым текстом, то пользователь может менять место указателя и звуковое сопровождение сразу подстраивается под новое положение в тексте. Также стоит отметить, что из обычных функций подобных приложений для чтения электронных книг здесь присутствует только изменение размера шрифта, что является очень скудным.

Минусами описанного решения является определение текущего читаемого текста. Пользователь вынужден взаимодействовать с приложением для того, чтобы поправить положение воспроизведения в следствии

следующих причин:

1) пользователь мог отвлечься и забыть поставить на паузу;

2) пользователь может захотеть перечитать только что прочитанный отрывок, но указатель на паузу он не может поставить, потому что это приводит к помутнению текста, что затрудняет его дальнейшее чтение;

3) пользователь мог отстать или опередить положение воспроизведения, при чем автоматическая подстройка параметра скорости может не помочь, потому что разные отрывки текста могут читаться с разной скоростью или даже перечитываться, для лучшего понимания.

Данные причины могут привести к частым ненужным взаимодействиям с приложением.

Другой проблемой является указатель воспроизведения, который несет в себе два неудобства:

1) это движущийся объект на экране, который отвлекает пользователя от чтения;

2) это элемент, который собою задает темп чтения, из-за чего пользователь может не успевать понимать прочитанный текст стремясь соответствовать скорости указателя или же заставлять пользователя увеличивать скорость, если он опережает указатель.

Рассмотренное выше решение не имеет удобной реализации, и оно не представлено на российском рынке, что полностью исключает возможность его использования на территории России. Поэтому данный проект призван реализовать идею в более удобной и локализованной для пользователя форме.

1.3. Сравнение форматов электронных книг

Проведем анализ существующих форматов электронных книг. От искомого формата требуется распространенность, возможность разметки текста, встраивание медиа файлов, а также отсутствие статической верстки. Сравнение проводилось среди наиболее популярных форматов.

TXT – обычный текстовый файл. Не содержит какой-либо разметки и возможности встроить медиа файлы. Слабо распространен как формат для электронных книг.

FB2 – открытый формат электронных книг разработанный в России на основе XML. Имеет разметку текста, но не имеет возможности хранить медиа файлы. Имеет широкую поддержку.

PDF – открытый формат электронных документов изначально разработанный Adobe Systems. Имеет разметку текста, но при этом статическую верстку страницы. В первую очередь предназначается для представления полиграфической информации в электронном виде. Файлы могут иметь большой вес в связи с наличием встроенных шрифтов и большого объема графики.

DOC – формат электронных документов для текстовых редакторов разработанный Microsoft. В 2008г. формат стал открытым. Имеется разметка текста, но со статической версткой.

EPUB – открытый формат электронных книг разработанный IDPF (International Digital Publishing Forum). Формат основан на web технологиях. Имеется разметка текста при динамической верстке. Имеется возможность встраивать медиа файлы. В силу своей гибкости, является наиболее популярным форматом для электронных книг.

Таким образом, проанализировав наиболее популярные форматы электронных книг, можно прийти к выводу, что для реализации идеи подходит формат EPUB – наиболее популярный и гибкий формат. Данный формат позволит с легкостью разметить текст книги музыкальными фрагментами, при этом храня внутри книги сами музыкальные композиции.

1.4. Обзор EPUB

EPUB – открытый формат электронных книг с расширением .epub. Определяет средства представления, упаковки и кодирования структурированного и семантически улучшенного веб-контента, включая XHTML,

CSS, SVG, изображения и другие ресурсы, для распространения в одно-файловом формате. Книга в этом формате представляет собой ZIP-архив, называемый Container (далее по тексту контейнер), который содержит в себе набор взаимосвязанных ресурсов, в соответствии со стандартом [5].

Контейнер обязательно в себе содержит:

1) корневой не заархивированный файл mimetype, в котором содержится строка «application/epub+zip» – mime тип контейнера;

2) файл container.xml в папке META-INF;

3) как минимум один Publication, состоящий из минимального набора документов:

a) Package Document;

b) Content Document;

c) Navigation Document.

Container.xml является частью OCF (Open Container Format). В данном файле прописаны пути к одному или нескольким Package Document, каждый из которых относится к своему Publication.

Publication – представляет собой единицу интеллектуальной или художественной работы (далее по тексту публикация). Таким образом контейнер может содержать в себе несколько публикаций.

Package Document – файл с расширением .opf, который содержит в себе: метаданные, манифест, spine. В метаданных хранятся общие данные о публикации, такие как: заголовок, автор, год издания и т.п. В манифесте прописаны ссылки на все ресурсы публикации. Spine содержит упорядоченный список ссылок на ресурсы публикации, представляющий собой порядок чтения по умолчанию.

Content Document – один из ресурсов публикации, представляющий из себя HTML или SVG файл. Содержит в себе непосредственно отображаемый пользователю контент.

Navigation Document – специализированный XHTML файл, который содержит в себе человекочитаемую и машиночитаемую глобальную нави-

гационную информацию.

Поскольку стандарт книги не подразумевает встраивание музыкальных фрагментов и хранение музыкальных композиций, то необходимо разработать собственный формат электронной книги на основе EPUB.

2. АНАЛИЗ ТРЕБОВАНИЙ

Платформа представляет собой набор взаимосвязанных компонентов, состоящая из мобильного приложения и веб-приложения. Мобильное приложение позволит пользователю скачивать приобретенные в веб-приложении книги собственного формата и открывать их. Веб-приложение позволит издателям загружать готовые книги базового формата, с помощью инструментария, приводить их к формату книги, использующемуся в платформе, и опубликовывать их. Пользователи в веб-приложении смогут получать и скачивать книги, опубликованные издателями.

2.1. Функциональные требования

В ходе анализа были выделены следующие функциональные требования к разрабатываемой платформе.

1. Платформа должна состоять из двух компонентов: мобильного приложения и веб-приложения.

2. Веб-приложение должно иметь инструментарий для создания книг собственного формата из книг базового формата.

3. Веб-приложение должно обеспечивать регистрацию и авторизацию пользователей.

4. Веб-приложение должно предоставлять издателям возможность:

a) загружать книги базового формата;

b) с помощью инструментария создавать книги собственного формата;

c) опубликовывать такие книги.

5. Веб-приложение должно предоставлять пользователям возможность:

a) находить и приобретать опубликованные издателями книги;

b) скачивать приобретенные книги.

6. Мобильное приложение должно отображать приобретенные в веб-приложение книги авторизированным пользователям.

7. Мобильное приложение должно авторизировать пользователей.

8. Мобильное приложение должно уметь открывать файлы собственного формата из внешних источников.

9. Мобильное приложение должно хранить в себе все открываемые и скачиваемые пользователем книги и выводить их список.

10. Мобильное приложение должно открывать файлы собственного формата.

11. Мобильное приложение должно определять какой фрагмент текста читает пользователь и воспроизводить для этого фрагмента музыку.

2.2. Нефункциональные требования

В ходе анализа были выделены следующие нефункциональные требования к разрабатываемой платформе.

1. Веб-приложение должно быть написано на языках JavaScript и HTML с использованием фреймворка Meteor.

2. Веб-приложение должно иметь разделение пользователей на две роли: пользователь и издатель.

3. Веб-приложение должно предоставлять API ресурсов для авторизированных пользователей.

4. Мобильное приложение должно быть написано на Swift.

5. Мобильное приложение должно быть доступно на устройстве Apple iPad с операционной системой iOS 10 и выше.

6. Мобильное приложение должно отображать приобретенные на сайте книги, по средствам использования API сервера сайта.

7. Мобильное приложение должно в основе своей быть построено на основе FolioReaderKit и расширять его классы чтобы реализовать идею проекта.

8. Создание файлов собственного формата должно производится путем внесения изменение в разметку страниц книги, а также создание отдельного каталога, содержащего в себе медиа файлы для воспроизведения.

2.3. Диаграмма прецедентов веб-приложения

В рамках веб-приложения разрабатываемой платформы предусмотрено три актера (рис. 1).

Рис. 1. Диаграмма прецедентов

1. Пользователь – аутентифицированный пользователь системы имеющий возможность покупать и скачивать книги.

2. Издатель – аутентифицированный пользователь системы с правами издателя имеющий возможность загружать, редактировать и опубликовывать книги. Наследуется от актера Пользователь.

3. Мобильное приложение платформы – мобильное приложение, аутентифицированное как Пользователь системы, с возможностью получать данные о приобретенных книгах, хранящихся в системе.

Перечислим прецеденты *Пользователя.*

1. *Купить книгу* – процесс при котором выбранная книга начинает ассоциироваться с пользователем как купленная.

2. *Скачать книгу* – процесс при котором, приобретенная книга скачивается на устройство Пользователя. Расширяет прецедент *Купить книгу.*

Перечислим прецеденты *Издателя*.

1. *Загрузить книгу* – процесс, при котором книга в базовом или собственном формате загружается на сайт и становиться доступной для редактирования.

2. *Редактировать книгу* – процесс, при котором книга в базовом или собственном формате наполняется музыкальными композициями.

3. *Опубликовать книгу* – процесс, при котором книга становится доступна для покупки *Пользователями*.

Перечислим прецеденты Мобильного приложения платформы.

1. *Получить список книг пользователя* – процесс, при котором актёр получает данные о книгах, приобретённых *Пользователем.*

2. *Получить книгу пользователя* – процесс, при котором актёр получает книгу, приобретённую *Пользователем.*

2.4. Диаграмма прецедентов мобильного приложения

В рамках мобильного приложения разрабатываемой платформы предусмотрено четыре актера (рис. 2).

1. *Неизвестный пользователь* – пользователь, не прошедший аутентификацию в *Веб-приложении платформы* и имеющий возможность читать книги, загруженные из внешних источников.

2. *Аутентифицированный пользователь* – пользователь, прошедший аутентификацию в *Веб-приложения платформы* и способный загружать книги из данного приложения. Унаследован от *Неизвестного пользователя.*

3. *Операционная система* – среда в которой функционирует данная система и которая может запросить открытие книги.

4. *Веб-приложения платформы* – веб-приложение, которое предоставляет данные о приобретенных книгах с возможностью их загрузки для *Аутентифицированного пользователя*.

В данной системе имеются следующие прецеденты.

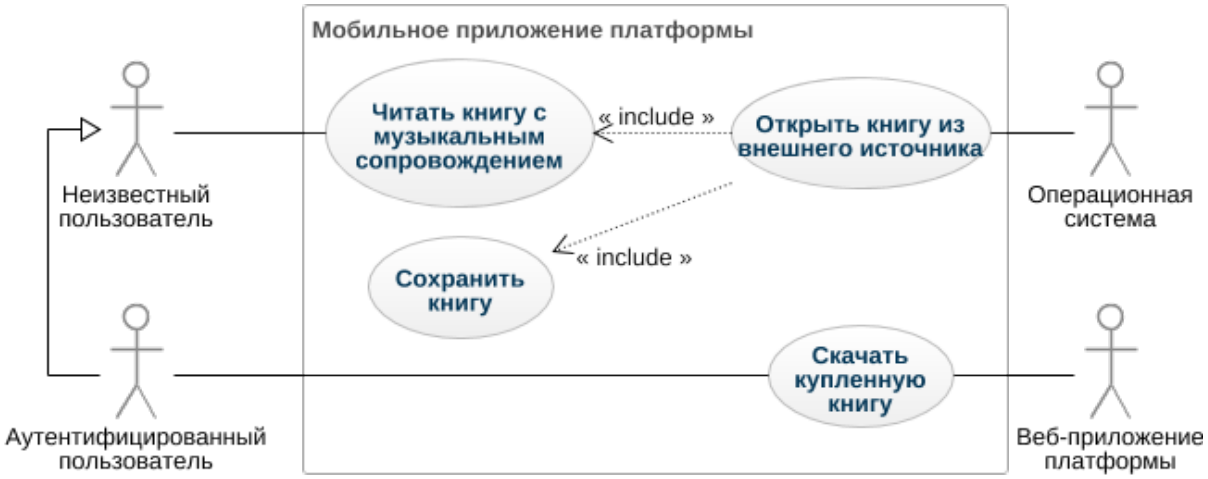


Рис. 2. Диаграмма прецедентов

1. *Читать книгу с музыкальным сопровождением* – описывает процесс выбора *Неизвестным пользователем* книги и её чтения, сопровождающегося музыкальным сопровождением на основе текущего читаемого фрагмента.

2. Скачать купленную книгу – описывает процесс загрузки книги, принадлежащей Аутентифицированному пользователю, с Веб-приложения платформы.

3. *Открыть книгу из внешнего источника* – описывает инициируемый со стороны *Операционной системы* процесс, открывающий хранящуюся вне данной системы книгу пользователю. Включает в себя прецеденты: *Сохранить книгу* и *Читать книгу с музыкальным сопровождением*.

4. *Сохранить книгу* – описывает процесс сохранения книги в памяти системы.

3. ПРОЕКТИРОВАНИЕ

3.1. Формат книги с интегрированным музыкальным сопровождением

Для реализации собственного формата был выбран формат цифровых изданий – Electronic Publication (EPUB). Назовем его MEPUB (Musical Electronic Publication). Такой формат дает обратную совместимость с EPUB – возможно открытие EPUB-файлов в разрабатываемом приложении и открытие MEPUB-файлов в приложениях, которые поддерживают только EPUB.

Для реализации проекта была изучено внутреннее устройство формата EPUB, который в своей основе построен на языке разметки веб-страниц на основе XML – XHTML [14]. Так же можно использовать HTML файлы, но содержащие в себе XHTML код. Самое главное отличие XHTML от HTML заключается в том, что код не может содержать одиночные тэги. Но в рамках данного проекта это ограничение никак не влияет на реализацию.

Для того, чтобы пометить части текста музыкальным фрагментом, нужно их заключить в какой-либо тэг со специальным смыслом. Что бы приложение поняло, что данная часть текста содержит в себе музыкальный фрагмент, можно использовать пользовательские HTML data-* атрибуты. У него простой синтаксис – любой атрибут, чьё имя начинается с «data-», является data-* атрибутом. Атрибут позволяет хранить в себе любую строчную информацию в любом теге HTML. Таким образом он подходит для хранения названия музыкальной композиции. Для атрибута было выбрано название «data-music».

Хранение аудио файлов будет происходит внутри EPUB Container. Для этого будет создаваться отдельная папка внутри контейнера, которая получила название «music». В ней хранятся все аудио файлы.

Придерживаясь стандарта EPUB, список аудио файлов будем хранить в Package Document, там будут указаны все ссылки на содержащиеся в EPUB Package звуковые файлы.

3.2. Компоненты веб-приложения

Веб-приложение состоит из 3 основных компонентов: клиентской части, серверной части и файлового сервера (рис. 3).

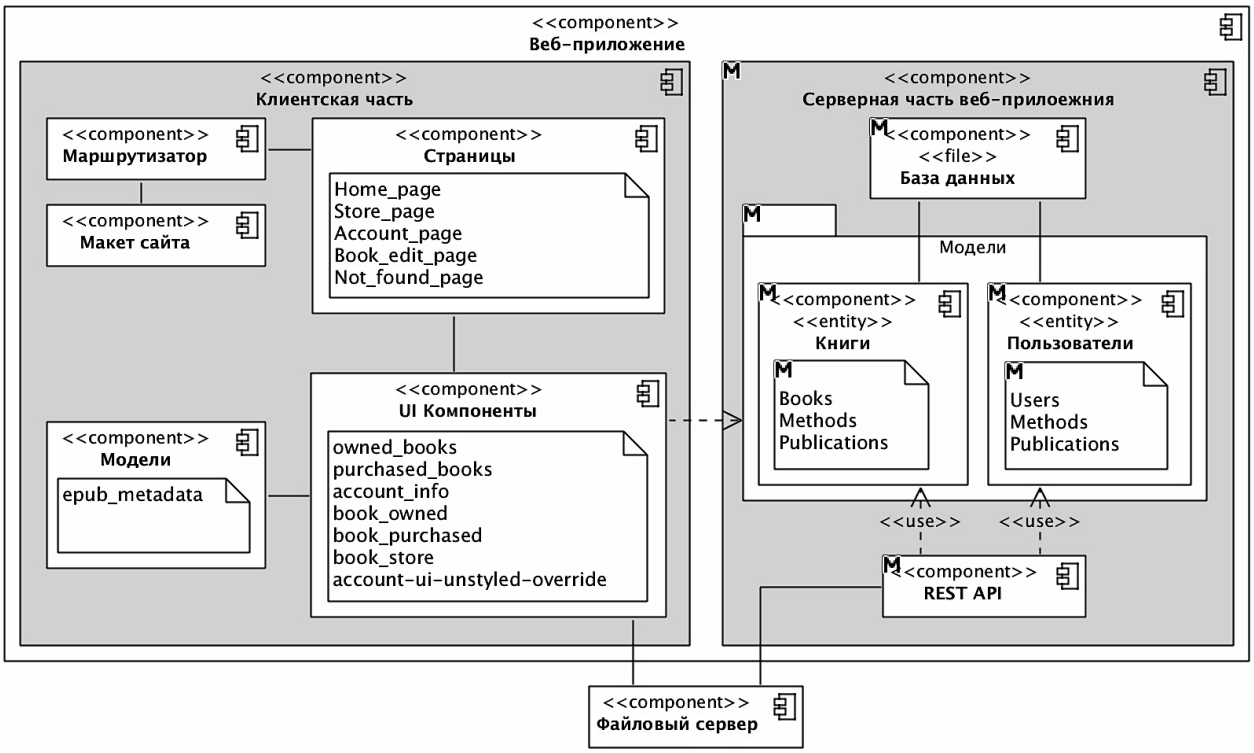


Рис. 3. Диаграмма структурных компонентов веб-приложения

При запросе пользователем веб-приложения ему выдается клиентская часть, которая состоит из следующих компонентов.

1. *Маршрутизатор* определяет навигацию в приложении. Для заданного пути вызывает рендеринг соответствующей страницы.

2. *Макет сайта* является корневым видом для всех страниц. Отображает шапку и подвал страниц.

3. *Страницы* отображаются в зависимости от местоположения пользователя в приложении. Каждая страница представляет из себя набор компонентов и правил их отображения:

a) главная страница (*home_page*) отображает приветствующую пользователя информацию;

b) страница магазина (*store_page*) отображает все опубликованные книги (*store_book*). Доступны для покупки только авторизированным поль-

зователям;

c) страница пользователя (*account_page*) отображает информацию об аккаунте (*account_info*), купленные книги (*purchased_books*) и книги, которыми он владеет, если является издателем (*owned_books*);

d) страница редактирования (*book_edit_page*) отображает инструментарий для редактирования книг для издателей;

e) страница по умолчанию (*Not_found_page*) отображает информацию, что запрашиваемая страница не найдена.

4. *UI Компоненты* занимаются отображением отдельных фрагментов информации на странице. Компоненты могут быть повторно использованы для их отображения на разных страницах.

a. *owned_books* – отображает список принадлежащих издателю книг с возможностью загрузить книгу.

b. *purchased_books* – отображает список купленных книг.

c. *account_info* – отображает информацию об аккаунте. Так же предоставляет возможность пользователю запросить статус издателя.

d. *book_owned, book_purchased, book_store* – отображают одну книгу в списке книг соответствующего типа: книга издателя, купленная книга, книга в магазине. Книгу издателя можно опубликовать, отредактировать или удалить. Купленную книгу можно скачать. Книгу в магазине можно купить авторизированным пользователям.

e. account-ui-unstyled-override – переопределяет отображение окна авторизации/регистрации из Meteor пакета «account-ui-unstyled».

5. *Модели* предназначены для работы с данными пользователя на клиенте.

a. epub_metadata предоставляет метаданные для данной книги.

Компоненты серверной части.

1. *База данных*, хранящая в себе коллекции пользователей и книг. Представляет собой NoSQL документно-ориентированную СУБД MongoDB. Более подробно о структуре базы данных в разделе 3.4.

2. *Книги* – компонент, содержащий модель коллекции книг из базы данных, а также набор действий, связанных с ней.

a. *books* – Meteor модель представления коллекции книг из базы данных.

b. *methods* – методы для работы с моделью вызываемые с клиентской части.

c. *publications* – содержит Meteor публикации, позволяющие клиентской части получать частичную или полную выгрузку модели в зависимости от настроек публикации. Данные передаются on the wire.

3. *Пользователи* – компонент, содержащий модель коллекции пользователей аналогично компоненту *книг*.

4. *REST API* предоставляет авторизированный доступ к ресурсам приложения, в данном случае к книгам.

Последним компонентом веб-приложения является *файловый сервер*. Он необходим из-за отсутствия в Meteor методов для передачи файлов по сети и хранения их в файловой системе. Файловый сервер написан на Node.js, имеет поддержку технологии CORS и CRUD операций. Так же сервер поддерживает работу с EPUB файлами, что позволяет производить CRUD операции так же над файлами, хранящимися в EPUB контейнерах.

3.3. Компоненты мобильного приложения

На рисунке 4 представлена диаграмма структурных компонентов мобильного приложения и его связь с серверной частью веб-приложения, описанной в предыдущем разделе.

Мобильное приложение состоит из следующих компонентов.

1. Контроллеры, являющиеся связующим звеном между модулями и представлением. Обрабатывают события представления, а с помощью модулей получают информацию для отображения в предсталениях.

a. *MBRBaseViewController* – базовый контролер, реализующий вспомогательный функционал для работы с паттерном MVC.

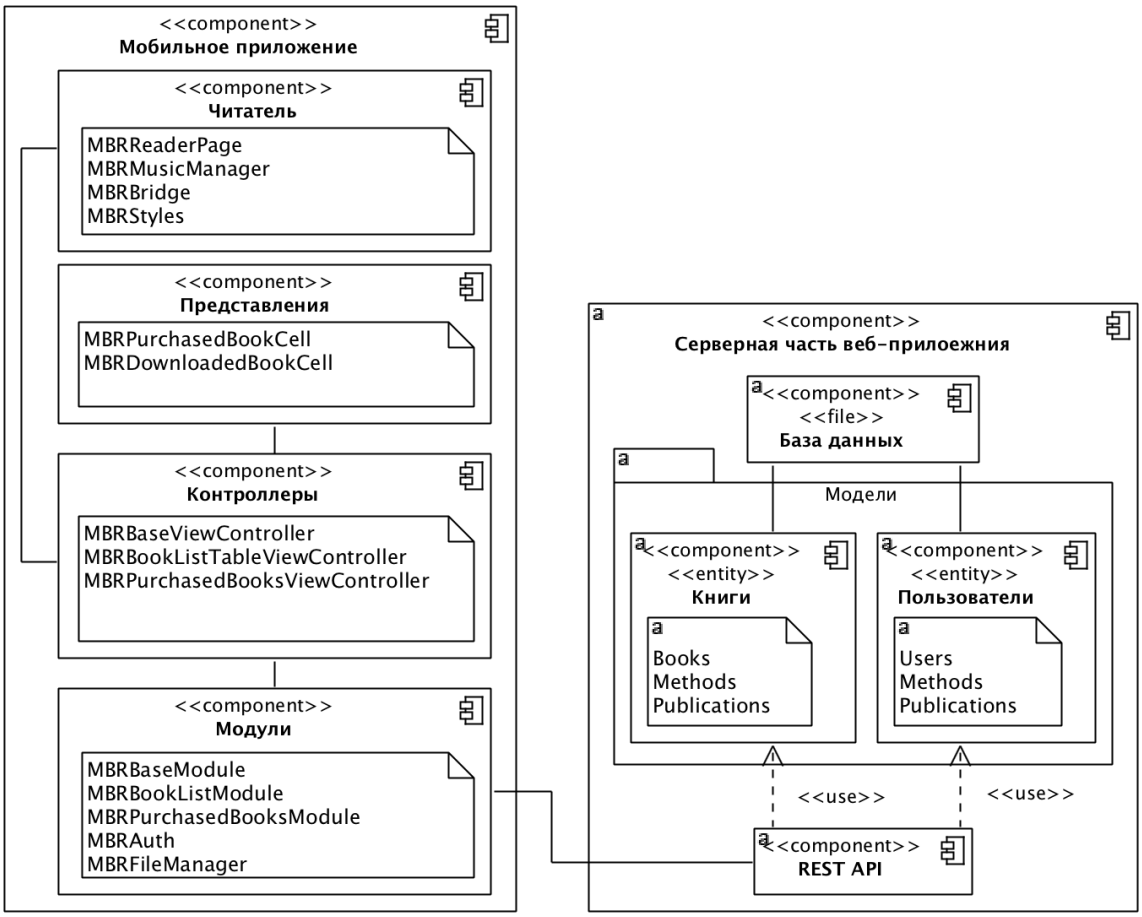


Рис. 4. Диаграмма структурных компонентов системы

b. *MBRBookListViewController* – контроллер, обрабатывающий экран со списком загруженных на устройство книг. На данном экране можно открыть выбранную книгу.

c. *MBRPurchasedBooksViewController* – контроллер, обрабатывающий экран со списком купленных пользователем в веб-приложении книг. Если пользователь не авторизирован, то показывает экран авторизации. На данном экране можно загрузить и открыть книгу.

2. Модули, содержащую логику работы с данными.

a. *MBRBaseModule* является базовым модулем с функционалом для работы с паттерном MVC.

b. *MBRBookListModule* – фасадный модуль для *MBRBookListViewController*. Занимается загрузкой списка книг, хранящихся на устройстве.

c. *MBRPurchasedBooksModule* – фасадный модуль для *MBRPur-*

chasedBooksViewController. Занимается загрузкой списка купленных книг, загрузкой файла книги и авторизацией пользователя при помощи *REST API* серверной части веб-приложения.

d. *MBRAuth* – модель, авторизующая пользователя в веб-приложении при помощи *REST API* этого приложения.

e. *MBRFileManager* – модель, непосредственно общающаяся с файловой системой.

3. *Читатель* – компонент, непосредственно отображающий выбранную пользователем книгу и реализующий функционал воспроизведения музыкальной композиции, соответствующей читаемому фрагменту.

a. *MBRReaderPage* отображает страницу книги пользователю. Отлавливает нажатия пользователя по тексту, передает координаты в скрипт страницы, получая в качестве ответа композицию для воспроизведения и воспроизводит ее.

b. *MBRMusicManager* непосредственно воспроизводит музыкальные композиции и делает переходы между ними.

c. *MBRBridge* – скрипт, который исполняется в среде страницы и может по координатам получать музыкальную композицию.

d. *MBRStyle* – стили рисующие разделители между музыкальными фрагментами.

Процесс обработки нажатия пользователя представлен на диаграмме последовательности на рисунке 5.

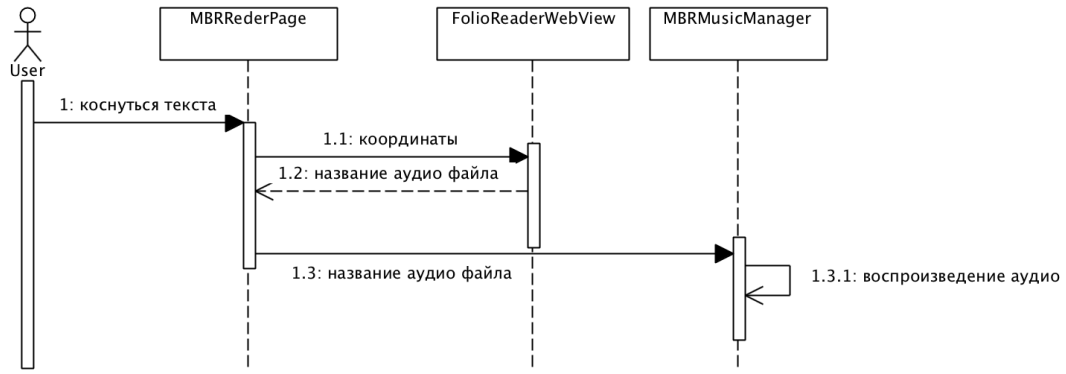


Рис. 5. Диаграмма последовательности процесса обработки касания

3.4. Схема базы данных

База данных имеет следующую структуру (рис. 6).

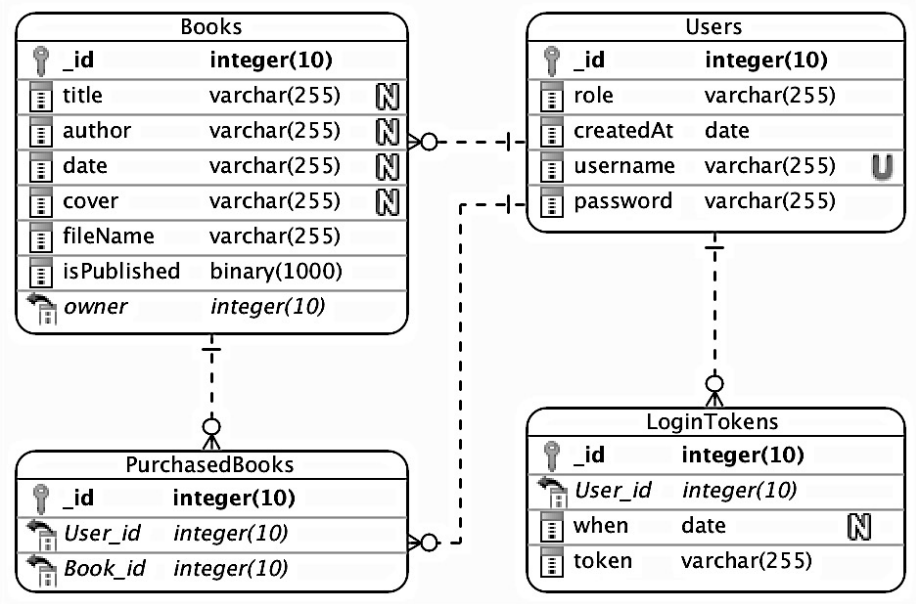


Рис. 6. Диаграмма последовательности процесса обработки касания

Рассмотрим таблицы, представленные выше.

1. *Books* – таблица хранящая все загруженные книги.

a. *_id* – уникальный идентификатор и первичный ключ.

b. *title* – заголовок книги.

c. *author* – автор книги.

d. *date* – дата публикации книги.

e. *cover* – обложка книги.

f. *fileName* – название загруженного файла. Используется для получения файла с файлового сервера.

g. *isPublished* – булева переменная, обозначающая опубликована ли книга. По умолчанию: false.

h. *owner* – идентификатор владельца. Внешний ключ ссылающийся на таблицу *Users*.

2. *Users* – таблица со списком пользователей.

a. *_id* – уникальный идентификатор и первичный ключ.

b. *role* – роль пользователя в системе. Принимает два значения:

«Покупатель» или «Издатель». По умолчанию «Покупатель».

c. *createdAt* – дата создания пользователя.

d. *username* – имя пользователя. Уникальное поле.

e. *password* – хешированный пароль пользователя.

3. *LoginTokens* – таблица, хранящая авторизационные токены пользователя.

a. *_id* – уникальный идентификатор и первичный ключ.

b. *User_id* – идентификатор пользователя, к которому относится токен. Внешний ключ, ссылающийся на таблицу *Users*.

c. *when* – время создания токена.

d. *token* – авторизационный токен пользователя.

4. *PurchasedBooks* – таблица содержащая купленные пользователями книги.

a. *_id* – уникальный идентификатор и первичный ключ.

b. *User_id* – идентификатор пользователя который купил книгу. Внешний ключ, ссылающийся на таблицу *Users*.

c. *Book_id* – идентификатор книги, которую приобрел пользователь. Внешний ключ, ссылающийся на таблицу *Books*.

4. РЕАЛИЗАЦИЯ

4.1. Работа с файлом

Необходимо рассмотреть саму структуру файла и работу с ним. В контейнере храниться вся необходимая информация для отображения контента пользователю, включая мета информацию, такую как имя автора, обложка, заголовок, навигационная информация и т.д., и сами контентные страницы. Нас интересуют сами контентные страницы, в которые внедрены необходимые музыкальные метки.

Контентные страницы представляют из себя XHTML-фалы. Используя атрибут «data-music», общая структура файла будет выглядит следующим образом (рис. 7).

```
...

<div data-music="tarck-1.mp3">
    ...
</div>

<div data-music="track-2.mp3">
    ...
    <div data-music="track-3.mp3">
        ...
    </div>

    <div data-music="tarck-1.mp3">
        ...
    </div>
    ...
</div>

<div data-music="track-3.mp3">
    ...
</div>

...
```

Рис. 7. Схематичная структура контента страницы

В качестве тегов, которые могут выступать в роли контейнеров на странице, содержащих информацию data-music, могут выступать любые теги, при чем не обязательно для этого выделять под это специальные контейнеры.

4.2. Инструменты и технологии, используемые при реализации веб-приложения

При реализации веб-приложения использовался мультипарадигменный язык программирования JavaScript [12, 15], фреймворк для этого языка – Meteor [3, 4], язык разметки страниц HTML и язык стилей SCSS.

SCSS – мета язык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей. Транслируется в обычный CSS код.

WebStorm – интегрированная среда разработки на JavaScript, HTML и CSS от JetBrains, предназначенная для разработки веб-приложений. Имеет встроенную поддержку Meteor.

Meteor – изоморфный JavaScript веб-фреймворк с открытым исходных кодом, написанный на Node.js. Позволяет писать приложения реального времени используя поведенческий шаблон издателя-подписчика и собственный протокол общения с базой данных, называемый DDP (Distributed Data Protocol). Имеется собственный менеджер пакетов. В качестве базы данных по умолчанию используется MongoDB, а в качестве шаблонного движка Blaze.

Шаблон издателя-подписчика подразумевает, что есть отправители сообщений, называемые издателями, и получатели сообщений, называемые подписчиками. Издатели не отправляют напрямую сообщения подписчикам, вместо этого сообщения делятся на классы, не содержащие информации об издателях и подписчиках. Таким образом издатели публикуют сообщения, а подписчики подписываются на них.

DDP – клиент-серверный протокол для запросов и обновления серверной базы данных и синхронизированных обновлений этой базы среди клиентов. Основан на шаблоне издателя-подписчика.

MongoDB – NoSQL документоориентированная СУБД. Использует JSON-подобные документы и схему базы данных [10].

Blaze – библиотека для создания пользовательского интерфейса по

средствам написания реагирующих HTML шаблонов.

Node.JS – среда выполнения JavaScript, встроенная в механизм JavaScript V8 для Chrome. Node.js использует управляемую событиями, неблокирующую модель ввода-вывода. Содержит встроенные модули для создания HTTP серверов и работы с файловой системой. Имеет поддержку пакетного менеджера NPM [8].

Epub.js – JavaScript библиотека, предназначенная для ренедринга EPUB документов в браузере [7].

JSZip – JavaScript библиотека предназначенная для работы с zip контейнерами, в том числе и EPUB. Позволяет добавлять, изменять или удалять файлы из zip контейнеров [9].

4.3. Реализация компонентов веб-приложения

Фреймворк Meteor задает определенную файловую иерархию, которая влияет на порядок загрузки файлов при старте приложения и на доступность этих самих файлов на сервере или клиенте. С помощью данной иерархии происходить деление приложения на клиентскую и серверную части.

В качестве реализации шаблона издатель-подписчик в Meteor используются две функции: Meteor.publish и Meteor.subscribe (в скрипте шаблона есть аналог: this.subscribe), соответственно для публикации и подписки на данные. Первая функция принимает название публикации и функцию, которая возвращает Mongo курсор. Mongo курсор это указатель на результат выборки по запросу из базы. Вторая функция принимает название публикации, на которую подписаться, а возвращает заглушку, которая может сказать, готовы ли данные для использования. При этом данные загружаются в локальную minimongo базу данных, в которой доступны те же коллекции, что и на сервере. Данные в коллекциях будут только те, что будут получены при помощи подписок.

Модель сущности представляет из себя объект Meteor класса Collec-

tion, который входит во встроенный пакет Mongo. Данный класс предоставляет методы для работы с коллекцией. При этом серверные модели связаны с базой данных сервера, а клиентские модели с локальной minimongo базой. Поэтому клиент не имеет прямого доступа ни к серверной базе данных, ни к серверным моделям. Клиент может только подписываться, чтобы получать данные, и вызывать Meteor методы, исполняемые на сервере чтобы вносить изменения в серверную базу. Это позволяет ограничить доступ не авторизированных пользователям к базе.

На рис. 8 представлен код создания публикации, которая предоставляет принадлежащие пользователю книги.

```
Meteor.publish('books.user_owned', function() {
 if (!this.userId) return this.ready();

  const user = Meteor.users.findOne({ _id: this.userId }, { owned_books: 1
});
  return Books.find({ _id: { $in: user.owned_books } }, {});
});
```

Рис. 8. Код создания публикации приобретенных книг

На рис. 9 представлен код подписки клиентом на серверную публикацию, описанную выше. Подписка происходит во время создания шаблона. Для подписки нужно передать только название публикации, но в других публикациях могут потребоваться дополнительные данные чтобы сформировать публикуемые данные.

```
Template.owned_books.onCreated(function () {
  this.autorun(() => {
    booksHandel = this.subscribe('books.user_owned');
  });
});
```

Рис. 9. Код подписки на приобретенные книги

На рис. 10 представлен код объявления метода, который добавляет запись о новой книге в коллекцию книг.

Для создания путей в приложении используется встроенный компонент FlowRouter. При запросе той или иной страницы вызывается пользо-

вательское действие, в котором мы вызываем рендер той или иной страницы. На рис. 11 представлен код, создающий маршрут для главной страницы.

```
Meteor.methods({
  'books.insert'(fileName, title, author, date, cover) {
    if (!this.userId)
      throw new Meteor.Error('books.insert.not-logged-in',
        'Must be logged in to create an book.');

    const bookId = Books.insert({
      fileName,
      title,
      author,
      date,
      cover,
      owner: Meteor.userId(),
    });

    Meteor.users.update({
      _id: Meteor.userId()
    }, {
      $push: { owned_books: bookId }
    });
  }
});
```

Рис. 10. Код объявления метода, добавляющего книгу в коллекцию

```
FlowRouter.route('/', {
  name: 'App.home',
  localizedName: 'Главная',
  action() {
    import '/imports/ui/pages/home/home.js';
    BlazeLayout.render('App_body', { main: 'App_home' });
  }
});
```

Рис. 11. Код создания маршрута для главной страницы

Любой UI компонент, в том числе макет страницы и сами страницы, имеет одинаковую структуру, состоящую из трех файлов: шаблон компонента, скрипт, обрабатывающий данный шаблон, и файла со стилями для шаблона. Чтобы этот набор файлов представлял собой единый компонент, в коде скрипта импортируется файл шаблона и файл стилей. После этого данный компонент можно импортировать, импортировав файл скрипта.

В пример можно привести компонент *books_owned* состоящий из: books_owned.html, book_owned.js и book_owned.scss. На рис. 12 представлен код шаблона, отображающий принадлежащие пользователю книги с

возможностью загрузить книгу.

На рис. 13 представлена выдержка из кода, загружающего книгу пользователя. Перед тем, как сохранить в базе данных запись о книге, файл книги предварительно загружается на файловый сервер. И уже после успешной загрузки клиент, отправляет запрос на создание записи в базе данных.

```
<template name="owned_books">
  <div class="container books" id="account_books">
    <h3>Ваши книги</h3>
    <div class="bordered">
      {{#if booksReady}}
        {{#if booksCount}}
          <div class="space-occupier-small">
            <p>
              <span class="add-book">Добавить книгу <i class="fa fa-plus"
aria-hidden="true"></i></span>
            </p>
          </div>
          <div class="book-list clearfix">
            {{#each book in books}}
              {{> book_owned book}}
            {{/each}}
          </div>
        {{else}}
          <div class="space-occupier">
            <p>У вас нет книг.
              <span class="add-book">Добавте хотя бы одну <i class="fa fa-
plus" aria-hidden="true"></i></span>
            </p>
          </div>
        {{/if}}
      {{else}}
        <div class="space-occupier">
          <h4>Загрузка</h4>
        </div>
      {{/if}}
      <input id="upload-book-input" type="file"
accept="application/epub+zip, .mepub"/>
    </div>
  </div>
</template>
```

Рис. 12. Шаблон компонента books_owned

Авторизация и регистрация в приложении происходит с помощью пакета accounts-password для Meteor. Отрисовка интерфейса авторизации/регистрации происходит с помощью пакета accounts-ui-unstyled, в котором некоторые шаблоны переопределены в компоненте *accounts-ui-unstyled-override*. Данные пакеты распространяются через каталог пакетов для Meteor – Atmosphere.

```
Template.owned_books.events({
  'change #upload-book-input'(event) {
    if (event.target.files.length === 0) return;

    const file = event.target.files[0];
    const newFileName = file.name.replace(new RegExp('(\.epub)$', 'gi'),
'.mepub');
    const formData = new FormData();
    formData.append('file', file, newFileName);

    const bookMetadata = new EpubMetadata(file);

    $.ajax({
      url: `${FILE_SERVER_URL + Meteor.userId()}/`,
      type: 'PUT',
      data: formData,
      processData: false,
      contentType: false,
      success(data, statusText, jqXHR) {
        if (jqXHR.status === 201 || jqXHR.status === 200) {
          bookMetadata.ready
            .then(() => {
              Meteor.call('books.insert', newFileName, bookMetadata.title,
bookMetadata.author, bookMetadata.date, bookMetadata.cover);
              location.reload();
            });
        }
      }
    });
  },
});
```

Рис. 13. Код загрузки файла пользователя в компоненте books_owned

4.4. Интерфейс веб-приложения

На рис. 1 приложения представлена главная страница приложения. На этой странице можно увидеть навигацию по сайту в левой части и кнопку для авторизации в правой части.

На рис. 2 приложения представлена страница магазина, где пользователь может приобрести опубликованные издателем книги. Если пользователь не авторизирован, то ему предлагается авторизоваться перед покупкой.

На рис. 3 приложения представлена страница пользователя, на которой отображаются его данные с возможностью запросить статус издателя, а также список приобретенных книг, которые он может загрузить. Загрузка происходит путем наведения курсора на обложку книги и нажатия кнопки загрузки, которая появляется при наведении курсора на обложку.

На рис. 4 приложения страница издателя с той же информацией что и у пользователя и дополнительно списком владеемых издателем книг с возможностью добавить таковую. Отредактировать или удалить свою книгу издатель, может нажав на соответствующие кнопки, появляющиеся при наведении на обложку книги.

На рис. 5 приложения представлена страница редактирования книги. В левой части отображается окно со списком музыкальных композиций, находящихся в этой книги с возможностью загрузить таковую. Текст можно выделить, при этом текст выделяется по параграфам. Выделенный текст можно пометить композицией, нажав композицию в списке. Выделение можно удалить, наведя курсов на фрагмент и нажав на крестик для удаления. В правой части окна располагается кнопка сохранения работы. По краям страницы располагаются кнопки следующей и предыдущей страницы.

4.5. Инструменты и технологии, используемы при реализации мобильного приложения

При реализации мобильного приложения используется объектно-ориентированный язык программирования Swift [17], а также JavaScript. Swift используется как нативный язык для разработки под iOS [13]. В свою очередь JavaScript используется для написания методов, обрабатывающих страницы книги.

Xcode – интегрированная среда разработки программного обеспечения для iOS, macOS и tvOS. Она предоставляет разработчику полноценные возможности по отладке продукта на устройстве и симуляторе.

MVC (Model-View-Controller) – шаблон проектирования архитектуры, используется для разделения модели и ее представления.

FolioReaderKit – фреймворк для платформы iOS, позволяющий рендерить EPUB файлы [6]. Может изменять вид и размер шрифтов, создавать заметки, проговаривать текст в слух и отображать оглавление.

AVFoundation – полнофункциональная среда для работы с аудиовидеоматериалами, основанная на времени, в iOS. Позволяет воспроизводить аудио файлы, хранящейся в контейнере книги [1].

4.6. Взаимодействие компонентов мобильного приложения

Поскольку мобильные приложения в своей основе используют сенсорный экран, то в качестве метода, позволяющего приложению понять какой фрагмент текста читает пользователь, наиболее подходящим является нажатие пользователем фрагмента текста, музыкальное сопровождение которого он хочет услышать. Обрабатывая нажатия пользователя, приложение определяет какой музыкальную композицию нужно воспроизвести. Для воспроизведения музыки был написан менеджер музыки, который использует музыкальные плееры из фреймворка Apple – AVFoundation.

Поскольку пользователь не видит внутренней структуры фрагментов и где, собственно, меняется сопровождение, то в местах перехода музыки рисуются разделители при помощи стилей, встраиваемых в страницу.

Запуск приложения происходит в двух вариантах: открытие приложения из списка приложений и открытие приложения при выборе его, как приложения способного открыть файл пользователя. В первом случае пользователю показывается экран со списком хранимых книг.

Также в приложении есть экран со списком приобретенных на сайте книг. При открытии этого экрана проверяется, авторизирован ли пользователь. Если не авторизирован, то показывается форма авторизации. После успешной авторизации пользователю показываются приобретенные книги. Когда пользователь нажимает на книгу из списка, то происходит ее скачивание. При этом авторизация, загрузка списка книг и загрузка книги происходит при помощи API веб-приложения.

Когда пользователь нажимает на скаченную книг с любого из двух экранов, происходит ее открытие посредством объекта MBRReader, который в свою очередь унаследован от оригинального FolioReader (рис. 14).

Далее происходит длинный процесс создания объектов фреймворка, которые в конечном итоге отобразят пользователю книгу. В процессе этой инициализации необходимо использовать объекты с собственной реализацией. Поэтому в цепочке инициализаций используются собственные классы, которые унаследованы от оригинальных. В частности, были созданы классы: MBRReader, MBRReaderCenter и MBRReaderPage, которые были соответственно унаследованы от: FolioReader, FolioReaderCenter и FolioRedaerPage. В процессе этой инициализации к странице подключаются стили и скрипт, который занимается получением музыки по заданным координатам.

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)

        let module = getModule()
        let bookPath = module.bookPathsList![indexPath.row]
        let config = FolioReaderConfig()
        config.scrollDirection = .horizontal;

        MBRReader.presentReader(parentViewController: self, withEpubPath:
bookPath, andConfig: config)
    }
```

Рис. 14. Обработка события нажатия на книгу из списка хранимых

На рис. 15 представлен код, обрабатывающий касания пользователя в книге. Данный обработчик занимается логикой отображения интерфейса и воспроизведения музыки.

```
override func handleTapGesture(_ recognizer: UITapGestureRecognizer) {
        let tapPosition = recognizer.location(in: webView)
        let shouldManageMusic =
Bool(performJavaScript("parser.performTouch(\(tapPosition.x),
\(tapPosition.y))") ?? "false")!

        if !shouldManageMusic {
            super.handleTapGesture(recognizer)
            return
        }

        let currMusic = performJavaScript("parser.currentMusic")

        if let currMusic = currMusic {
            musicManager!.setMusic(fileName: currMusic)
        }
    }
```

Рис. 15. Обработчик касания

4.7. Интерфейс мобильного-приложения

На рис. 6 приложения представлен экран с авторизацией, где пользователь вводит свой логин и пароль и после нажатия может увидеть список приобретенных книг.

На рис. 7 приложения представлен экран со списком приобретенных книг после авторизации, который идентичен экрану со списком загруженных книг. По нажатию на книгу в списке, она загружается. Процесс и завершение загрузки отображается иконками в правой части списка. После загрузки пользователь может открыть книгу повторным нажатием по книге в списке.

На рис. 8 приложения представлен экран с открытой книгой, где горизонтальными линиями отображаются переходы между музыкальными фрагментами. Когда книга открывается начинается автоматическое воспроизведение корневого композиции. Пользователь, во время чтения, видя, что он начал читать текст за разделительной линией, нажимает на текст за этой линией, после чего музыка плавно сменяется одна на другую. По нажатию на пустое поле или фрагменту текста, где нет музыкального сопровождения, пользователю показывается меню, где он может поменять размер текста, шрифт, способ отображения текста, сменить на светлую/темную темы и многое другое, что включает в себя FolioReader.

5. ТЕСТИРОВАНИЕ

Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование – это одна из техник контроля качества, включающая в себя активности по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов [16].

Тестовый случай – это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или ее части.

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения исходным функциональным требованиям заказчика. Проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям.

5.1. Тестирование веб-приложения

Тест № 1. Цель: проверить доступность основных страниц: главная, магазина и аккаунта.

Ожидаемый результат: при запросе главной страницы, ее отображение. Потом, при помощи навигации на сайте, открытие страницы магазина и аккаунта.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 2. Цель: зарегистрироваться в системе.

Ожидаемый результат: по нажатию на иконку авторизации должна отобразиться форма авторизации. По нажатию на кнопку регистрации в этой форме авторизации, должна появится форма регистрации. После ввода логина, пароля с подтверждением и нажатием кнопки регистрации,

пользователь должен быть авторизирован под ранее введенным логином.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 3. Цель: авторизоваться в системе.

Ожидаемый результат: после ввода логина, пароля и нажатия кнопки авторизации в форме авторизации, пользователь должен быть авторизован в системе.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 4. Цель: загрузить книгу как издатель.

Ожидаемый результат: на странице аккаунта после нажатия на кнопку запроса статуса издателя, получить такой статус и получить доступ к книгам издателя. После нажатия на загрузку книги издателя и выбора книги увидеть загруженную книгу в списке.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 5. Цель: отредактировать книгу как издатель.

Ожидаемый результат: после нажатия на кнопку редактирования книги в списке книг издателя, увидеть страницу редактирования книги. Добавить музыкальную композицию, пометить ею фрагменты текста, удалить некоторые пометки, после чего сохранить книгу. После повторного редактирования книги, изменения должны сохраниться.

Результат: после нажатия на редактирование книги в списке книг издателя открывается страница редактирования, где отображается содержимое книги. При помощи инструментов на странице, загружаются музыкальные композиции и отображаются в списке. После выделение фрагмента текста и нажатия загруженной композиции, фрагмент выделяется цветом композиции выбранном из списка. Пометка фрагментов удаляется при нажатии кнопки удаления в самом фрагменте. Сохранение книги происходит успешно. После повторного открытия изменения не пропадают. Ожи-

даемый и полученный результаты совпадают, цель достигнута.

Тест № 6. Цель: опубликовать книгу.

Ожидаемый результат: после нажатия на кнопку публикации книги в списке книг издателя, зайти на страницу магазина и увидеть опубликованную книгу.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 7. Цель: приобрести книгу.

Ожидаемый результат: после нажатия кнопку покупки книги в магазине, зайти на страницу аккаунта и увидеть приобретенную книгу в списке приобретенных книг.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 8. Цель: скачать приобретенную книгу.

Ожидаемый результат: после нажатия на кнопку загрузки книги списке приобретенных книг, она должна скачать на устройство.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

5.2. Тестирование мобильного приложения

Тест № 1. Цель: открыть файл книги в приложении.

Ожидаемый результат: при открытии списка доступных приложений, которые могут открыть файл, должно отобразится наше приложении. После нажатия на иконку приложения, должно открыться приложения, и книга должна появится в списке.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 2. Цель: отобразить главную страницу приложения со списком книг.

Ожидаемый результат: на экране отображена главная страница при-

ложения со списком книг. У каждой книги должно отобразится ее обложка, название и ее авторы. При отсутствии книг должен отобразиться пустой список.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 3. Цель: скачать приобретенную книгу на сайте.

Ожидаемые результаты: открытие страницы со списком приобретенных книг. При необходимости авторизоваться в системе. Нажать на приобретенную не скаченную книгу, после чего она должна скачаться.

Результат: после открытия экрана с приобретенными книгами на экране отобразилась форма авторизации. После корректного ввода авторотационных данных показался список приобретенных книг. После нажатия на книгу, она скачалась. Ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 4. Цель: отобразить книгу.

Ожидаемые результат: после нажатия на книгу в списке загруженных книг или нажатия на загруженную книгу в списке приобретенных книг, должна открыться книга и, если на открытой странице есть музыкальная композиция, то она должна воспроизвестись.

Результат: после нажатия на книгу, открывается ее содержимое на последней открытой странице. Поскольку на этой странице была музыка, она начала играть с плавным нарастанием по звуку. Ожидаемый и полученный результаты совпадают, цель достигнута.

Тест № 5. Цель: сменить музыку.

Ожидаемый результат. После нажатия на текст после разделителя происходит плавная смена одной композиции на другую.

Результат: совпадает с ожидаемым.

Тест № 6. Цель: открыть меню.

Ожидаемый результат: по нажатию на пустое поле страницы открывается меню. В этом меню пользователь может изменить тип, размер

шрифтов, тип отображения контента, сделать заметки и воспроизвести в слух текст.

Результат: ожидаемый и полученный результаты совпадают, цель достигнута.

ЗАКЛЮЧЕНИЕ

При выполнении работы были решены следующие задачи:

1) проведен анализ предметной области;
2) проведен анализ требований;
3) разработана архитектура компонентов;
4) разработаны компоненты;
5) протестированы компоненты.

Данный проект в целом имеет следующие перспективы по улучшению пользовательского опыта:

1) более полное информирование пользователя при редактировании книги;
2) добавление навигации при редактировании книги;
3) встраивание прослушивания музыкального сопровождения прямо в редактор;
4) встраивание фильтра в магазин;
5) встраивание магазина книг в мобильное приложение;
6) общий редизайн системы.

СПИСОК ЛИТЕРАТУРЫ

1. Apple Developer Documentation. [Электронный ресурс] URL: https://developer.apple.com/documentation/ (дата обращения: 08.04.2018).

2. Booktrack. Soundtrack for books. [Электронный ресурс] URL: https://booktrack.com/ (дата обращения: 01.03.2018).

3. Build Apps with JavaScript | Meteor. [Электронный ресурс] URL: https://www.meteor.com/ (дата обращения: 02.04.2018).

4. Dobrin G. Build Applications with Meteor. – Birmingham: Packt Publishing, 2017. – 355 p.

5. EPUB 3.1. International Digital Publishing Forum. [Электронный ресурс] URL: http://idpf.org/epub/31/ (дата обращения: 20.03.2018).

6. FolioReaderKit. Flexible and powerful Epub reader. [Электронный ресурс] URL: https://github.com/FolioReader/FolioReaderKit/ (дата обращения: 09.04.2018).

7. GitHub – futurepress/epub.js: Enhanced eBooks in the browser. [Электронный ресурс] URL: https://github.com/futurepress/epub.js/ (дата обращения: 02.04.2018).

8. Index | Node.js v10.1.0 Documentation. [Электронный ресурс] URL: https://nodejs.org/dist/latest-v10.x/docs/api/ (дата обращения: 02.04.2018).

9. JSZip. [Электронный ресурс] URL: https://stuk.github.io/jszip/ (дата обращения: 01.04.2018).

10. The MongoDB 3.6 Manual – MongoDB Manual 3.6. [Электронный ресурс] URL: https://docs.mongodb.com/manual/ (дата обращения: 20.04.2018).

11. Богатырева Ж.В., Шутилова М.Ф. Влияние музыки на человека. // Современные наукоемкие технологии. – 2013. – № 7-2. – С. 181–183.

12. Браун Э. Изучаем JavaScript. Руководство по созданию современных веб сайтов. – Пер. с анг. – М.: Диалектика, 2017. – 368 с.

13. Конвэй Дж., Хиллегасс А. Программирование под iOS. Для профессионалов. – СПб.: Питер, 2013. – 608 с.

14. Муссиано Ч., Кеннеди Б. HTML и XHTML. Подробное руководство. – М.: Символ-Плюс, 2011. – 752 с.

15. Современный учебник Javascript. [Электронный ресурс] URL: https://learn.javascript.ru/ (дата обращения: 21.03.2018).

16. Тампе Л. Introducing Software Testing. – М.: Вильямс, 2003. – 368 с.

17. Харазян А.А. Язык Swift – самоучитель. – СПб.: БХВ-Петербург, 2016. – 176 с.

ПРИЛОЖЕНИЕ

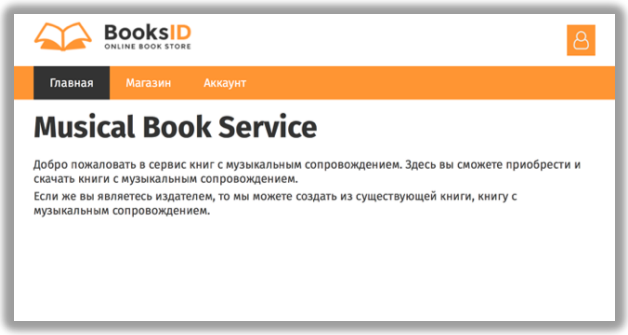


Рис. 1. Главная страница веб-приложения

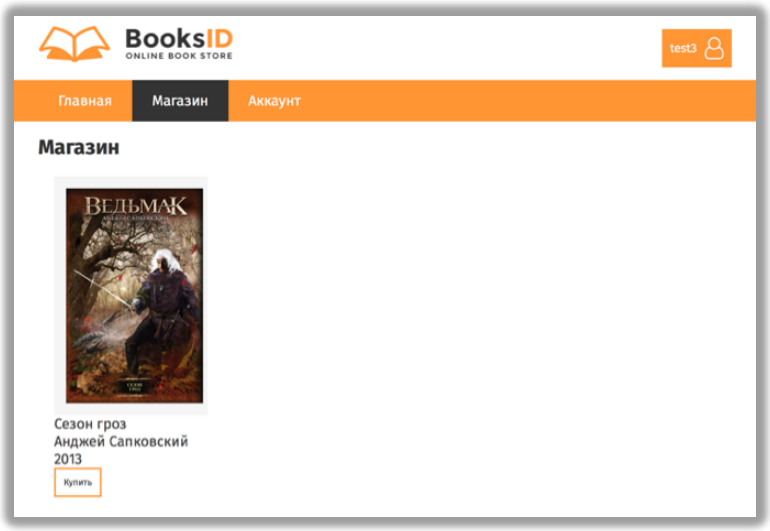


Рис. 2. Страница магазина веб-приложения

Рис.3. Страница аккаунта пользователя веб-приложения

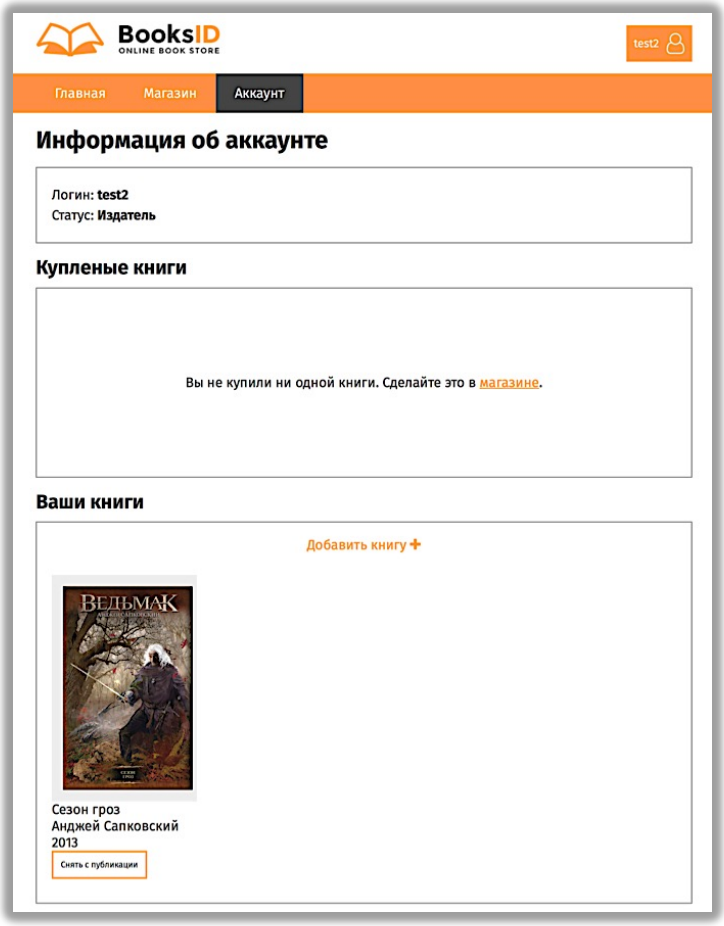


Рис. 4. Страница аккаунта издателя веб-приложения

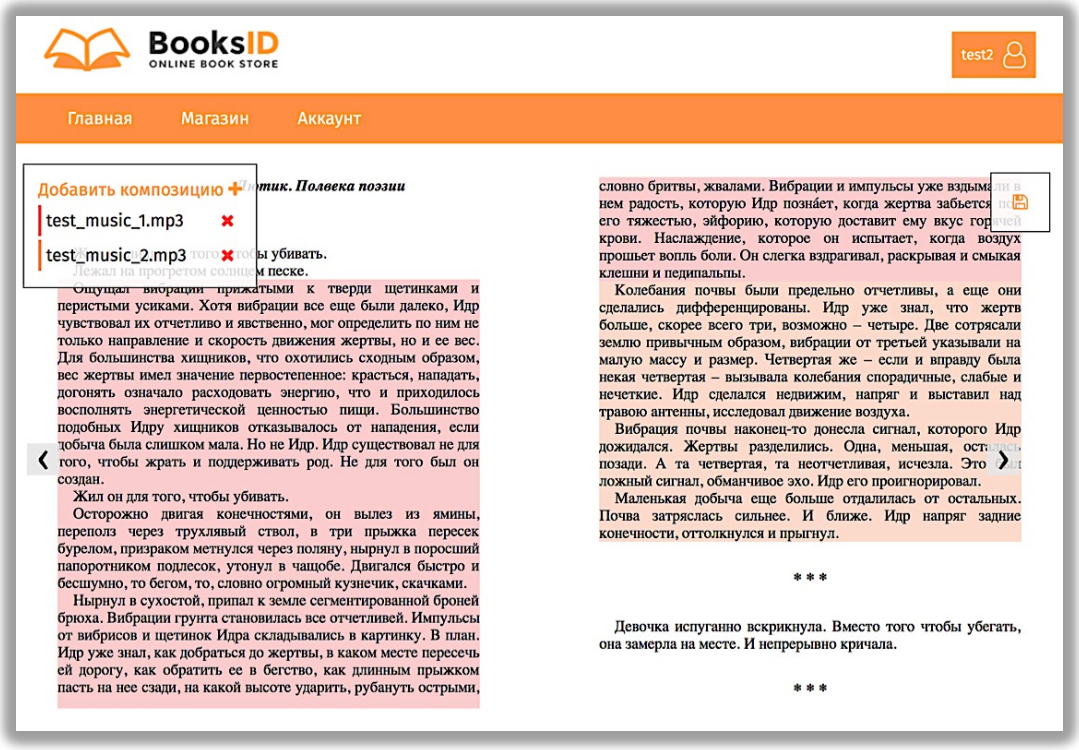


Рис. 5. Страница редактирования книги веб-приложения

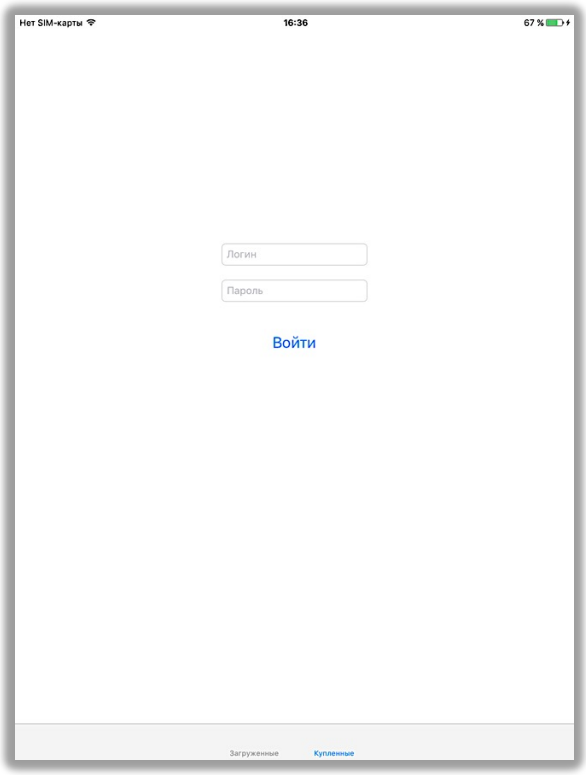


Рис. 6. Экран авторизации в мобильном приложении

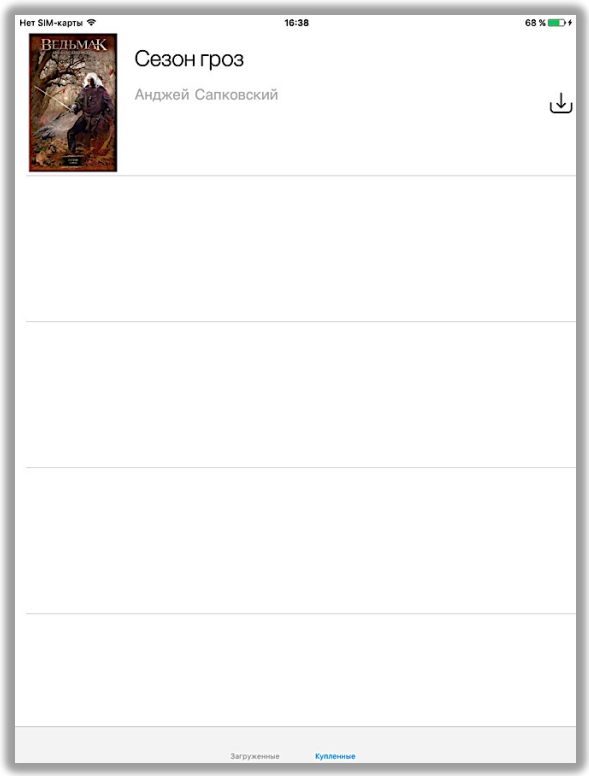
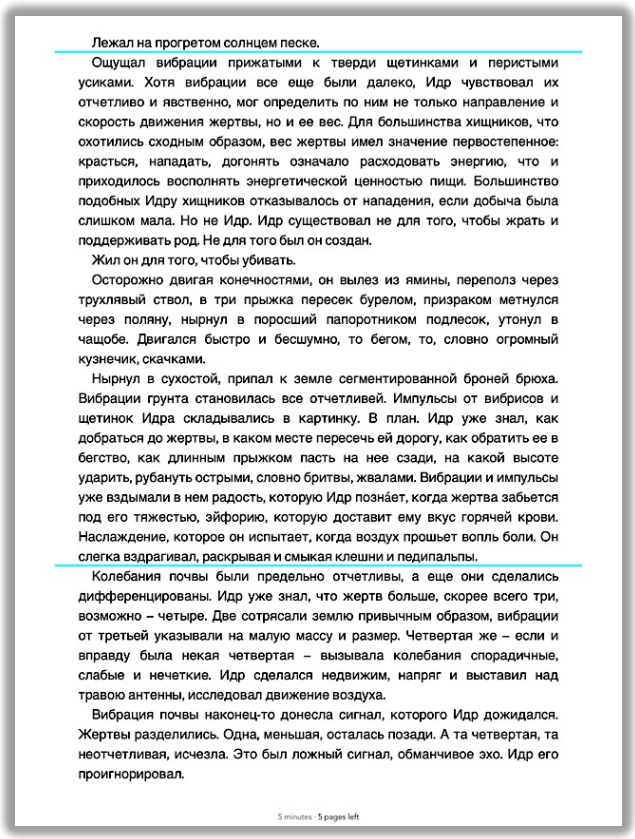


Рис. 7. Экран приобретенных книг



Лежал на прогретом солнцем песке.

Ощущал вибрации прижатыми к тверди щетинками и перистыми усиками. Хотя вибрации все еще были далеко, Идр чувствовал их отчетливо и явственно, мог определить по ним не только направление и скорость движения жертвы, но и ее вес. Для большинства хищников, что охотились сходным образом, вес жертвы имел значение первостепенное: красться, нападать, догонять означало расходовать энергию, что и приходилось восполнять энергетической ценностью пищи. Большинство подобных Идру хищников отказывалось от нападения, если добыча была слишком мала. Но не Идр. Идр существовал не для того, чтобы жрать и поддерживать род. Не для того был он создан.

Жил он для того, чтобы убивать.

Осторожно двигая конечностями, он вылез из ямины, переполз через трухлявый ствол, в три прыжка пересек бурелом, призраком метнулся через поляну, нырнул в поросший папоротником подлесок, утонул в чащобе. Двигался быстро и бесшумно, то бегом, то, словно огромный кузнечик, скачками.

Нырнул в сухостой, припал к земле сегментированной броней брюха. Вибрации грунта становилась все отчетливей. Импульсы от вибрисов и щетинок Идра складывались в картинку. В план. Идр уже знал, как добраться до жертвы, в каком месте пересечь ей дорогу, как обратить ее в бегство, как длинным прыжком пасть на нее сзади, на какой высоте ударить, рубануть острыми, словно бритвы, жвалами. Вибрации и импульсы уже вздымали в нем радость, которую Идр познáет, когда жертва забьется под его тяжестью, эйфорию, которую доставит ему вкус горячей крови. Наслаждение, которое он испытает, когда воздух прошьет вопль боли. Он слегка вздрагивал, раскрывая и смыкая клешни и педипальпы.

Колебания почвы были предельно отчетливы, а еще они сделались дифференцированы. Идр уже знал, что жертв больше, скорее всего три, возможно – четыре. Две сотрясали землю привычным образом, вибрации от третьей указывали на малую массу и размер. Четвертая же – если и вправду была некая четвертая – вызывала колебания спорадичные, слабые и нечеткие. Идр сделался недвижим, напряг и выставил над травою антенны, исследовал движение воздуха.

Вибрация почвы наконец-то донесла сигнал, которого Идр дожидался. Жертвы разделились. Одна, меньшая, осталась позади. А та четвертая, та неотчетливая, исчезла. Это был ложный сигнал, обманчивое эхо. Идр его проигнорировал.

5 minutes · 5 pages left

Рис. 8. Экран с книгой в мобильном приложении