Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Научно-образовательный центр «Высшая ИТ школа»

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р. физ.-мат. наук, профессор
_____ О.А.Змеев
подпись
«__11__» ____июня____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА IOS-ПРИЛОЖЕНИЯ ДЛЯ ПРОСЛУШИВАНИЯ АУДИОФАЙЛОВ И РАДИО

по направлению подготовки 09.03.04 Программная инженерия
направленность (профиль) «Программная инженерия»

Перегудова Кристина Сергеевна

Руководитель ВКР
д-р физ.-мат. наук, профессор
_____ А.В. Китаева
подпись
«__10__» ____июня____ 2022 г.

Консультант ВКР
программист Управления цифровых решений ТГУ
_____ Л.С. Иванова
подпись
«__9__» ____июня____ 2022 г.

Автор работы
студент группы № 971810
_____ К.С. Перегудова
подпись
«__08__» ____июня____ 2022 г.

Томск – 2022

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
НОЦ «Высшая ИТ школа»

УТВЕРЖДАЮ
руководитель ООП
д-р.физ.-мат. наук, профессор
_____ О.А.Змеев
«__» февраля 2022 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы бакалавра обучающемуся
Перегудова Кристина Сергеевна
(Ф.И.О. обучающегося)

по направлению подготовки Программная инженерия, направленность «Программная инженерия»

1. Тема выпускной квалификационной работы бакалавра
Разработка iOS-приложения для прослушивания аудиофайлов и радио

2. Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис – «09» июня 2022 г.
б) в ГЭК – «11» июня 2022 г.

3. Исходные данные к работе:
Цель: разработать iOS-приложения для прослушивания аудиофайлов и радио.
цели и задачи ВКР, ожидаемые результаты
Задачи: анализ требований, проектирование архитектуры, разработка приложения.
Ожидаемые результаты: iOS-приложение для прослушивания аудиофайлов и радио, соответствующее требованиям заказчика

Организация, по тематике которой выполняется работа
Общество с ограниченной ответственностью «КОДЭ-Т»

Руководитель выпускной квалификационной работы
д-р физ.-мат. наук, профессор _____ / А.В.Китаева
(должность, место работы) (подпись) (И.О. Фамилия)

Консультант выпускной квалификационной работы
программист управления цифровых решений ТГУ _____ / Л. С. Иванова
(должность, место работы) (подпись) (И.О. Фамилия)

Задание принял к исполнению
«18» 02 2022 г _____ / К. С. Перегудова
(дата) (подпись) (И.О. Фамилия)

АННОТАЦИЯ

Выпускная квалификационная работа: 54 стр., 40 рис., 3 табл., 14 листингов, 37 источников.

ПРОСЛУШИВАНИЕ АУДИОКОНТЕНТА, РАДИО, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, IOS, SWIFT.

Объект разработки: мобильное приложение для прослушивания аудиоконтента и радио

Цель работы: разработать iOS-приложение для прослушивания аудиоконтента и радио.

Спроектировано и разработано мобильное приложение для платформы iOS, позволяющее прослушивать аудиофайлы (музыка, подкасты, аудиокниги) и радиоэфиры.

ОГЛАВЛЕНИЕ

ГЛОССАРИЙ

Шаффл (shuffle) - включение случайного порядка треков.

Репит (repeat) - активация повтора одного трека или всей очереди воспроизведения.

Свайп (swipe) - смахивать, проводить пальцем в сторону по сенсорному экрану смартфона.

Контрол (control) - элемент управления, кнопка.

Скролл (scroll) - форма представления информации, при которой содержимое двигается в вертикальном или горизонтальном направлении.

МП - мобильное приложение.

Фид (feed) - лента, набор блоков.

ВВЕДЕНИЕ

Как известно, мобильное приложение является программным обеспечением, которое специально разрабатывается для конкретной мобильной платформы (iOS, Android и др.) и предназначается для применения на смартфоне, планшете, умных часах и иных мобильных устройствах. Доля пользователей, которые пользуются сетью Интернет с мобильных платформ, за последние годы значительно выросла, поэтому разработка приложений для мобильных устройств является одним из быстро развивающихся направлений в сфере информационных технологий. Мобильные приложения предоставляют пользователям возможность получать доступ к интересующей их информации в любое время и в любом месте. Использование приложений избавляет от долгих ожиданий загрузки графики, изображений, звука и других компонентов, а также экономит время, поскольку программа уже установлена на телефон.

Мобильные приложения в основном разрабатываются для операционных систем Android и iOS. В данной работе рассматривается разработка именно iOS-версии приложения. Операционная система iOS была создана на основе MacOS и предназначена для использования исключительно в продуктах компании Apple - iPhone. Данная ОС характеризуется высокой стабильностью работы, наличием защитных функций от вирусного ПО (программное обеспечение), эффективным распределением ресурсов, обеспечивающим высокую производительность и энергоэффективность как планшетов, так и смартфонов. Согласно статистике, представленной в источнике [1], к концу 2020 года доля пользователей этой платформы в мире составила 24,99%. Поэтому разработка клиентского приложения для устройств семейства iPhone является актуальной задачей.

Заказчиком проекта выступила российская медиакомпания. Таким образом, на основании вышеизложенного, цель данной работы — разработать iOS-приложение для прослушивания аудиоконтента и радио.

Задачи проекта включают в себя анализ требований, проектирование архитектуры и разработку приложения.

1 Анализ требований

Во время работы над проектом, в первую очередь, были проанализированы бизнес-процессы компании Заказчика, функциональные и нефункциональные требования к продукту.

1.1 Варианты использования

Рисунок 1 - Варианты использования системы

В текущей реализации системой поддерживается только авторизованный пользователь. С точки зрения функциональных требований, приложение, в первую очередь, должно давать пользователю возможность просмотра ленты с контентом, откуда можно перейти на детальные страницы контента или сразу начать его воспроизведение. Также есть возможность перейти в другие разделы для поиска контента: поиск и рекомендации «Я слушаю». В добавок, пользователь может управлять плейлистами из раздела коллекции (*Рисунок 1*).

Неотъемлемой частью приложения является плеер. Плеер позволяет пользователю управлять проигрыванием аудиофайлов. В приложении доступно два вида плееров: мини плеер, который доступен с каждого экрана приложения, и большой плеер, который имеет расширенный функционал мини плеера. Оба плеера обладают минимально необходимым функционалом для прослушивания (*Рисунок 2*).

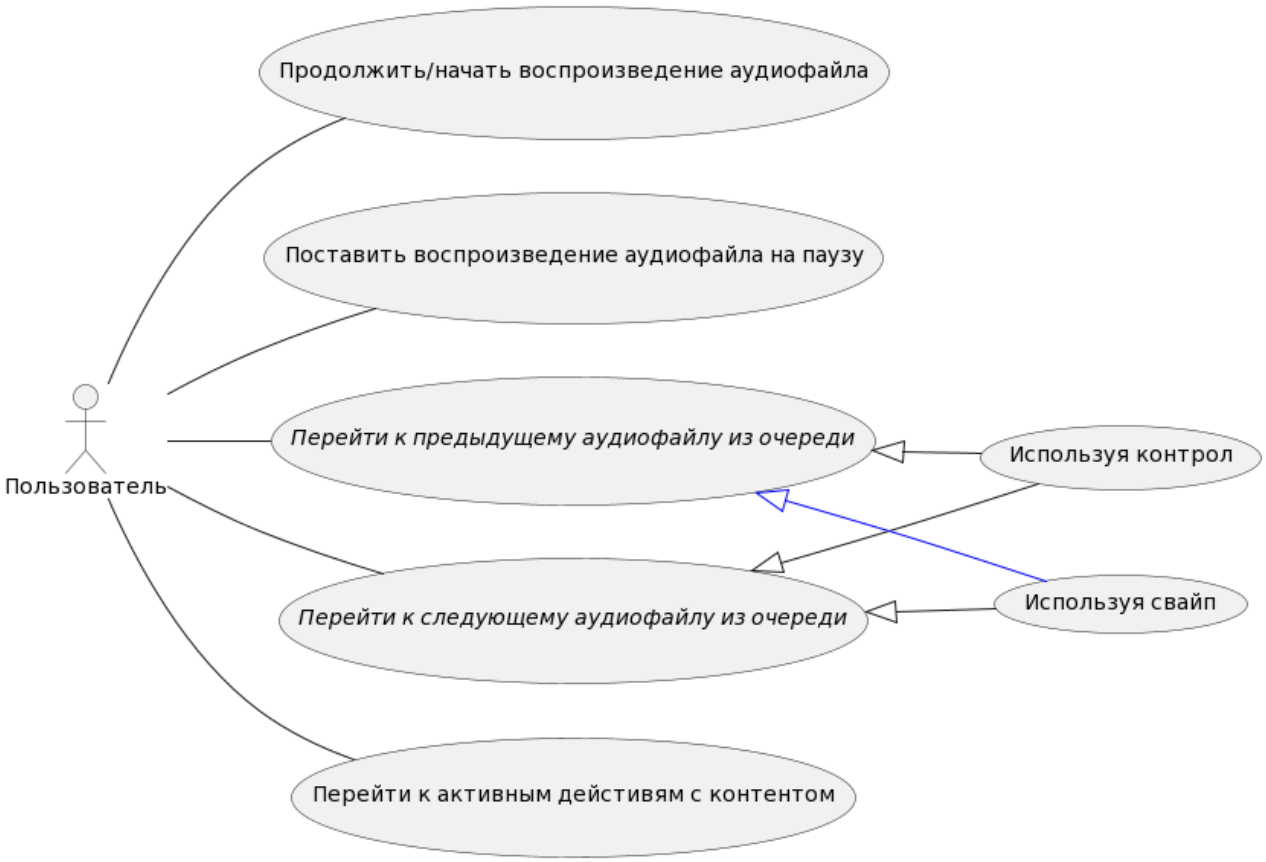


Рисунок 2 - Варианты использования - плеер (основное)

Помимо базовых возможностей работы с плеером, пользователю предоставляются дополнительные возможности через большой плеер (*Рисунок 3*). Так, можно оценивать понравившиеся аудиофайлы и скрывать непонравившиеся. Некоторые действия, такие как перемотка и изменение скорости, шафл и репит, доступны только для определенного типа контента. Также пользователь должен иметь возможность включить таймер сна и перейти к очереди проигрывания (*Рисунок 4*).

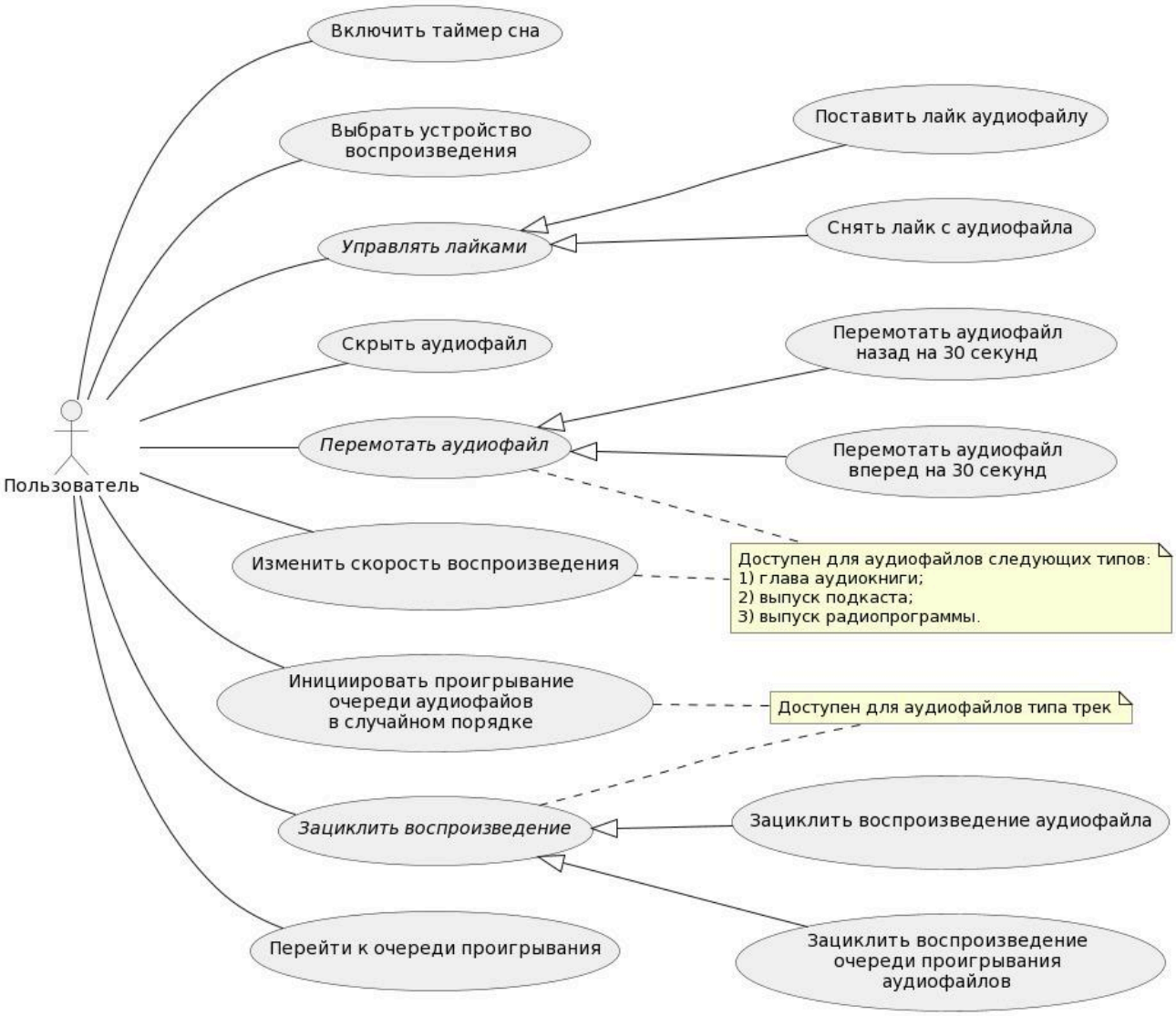


Рисунок 3 - Варианты использования - большой плеер

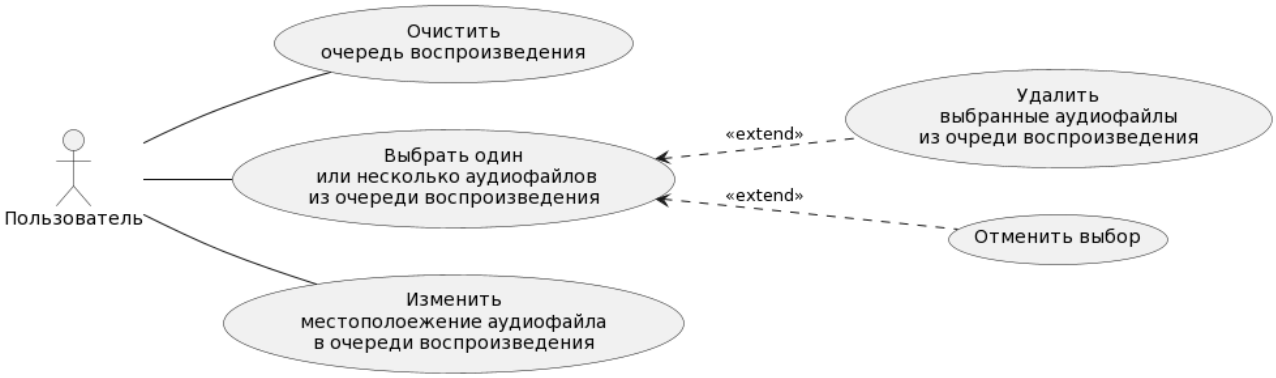


Рисунок 4 - Варианты использования - очередь проигрывания

Наиболее сложным в данной системе является процесс создания плейлиста. В таблице 1 приведен сценарий варианта использования «Создать плейлист».

Таблица 1- Сценарий варианта использования «Создать плейлист»

Пользователь	**Приложение**
Предусловие	
Пользователь находится на экране коллекций	
1. Выбирает раздел «Музыка»	
	2. Устанавливает тип аудиофайла - трек
3. Нажимает кнопку «Плейлисты»	
	4. Открывает список плейлистов
5. Нажимает кнопку «Создать плейлист»	
	6. Отображает диалоговое окно для ввода названия
7. Вводит название плейлиста	
8. Нажимает кнопку «Создать»	
	9. Проверяет название плейлиста, название - не пустое
	10. Создает пустой плейлист
	11. Запрашивает аудиофайлы определенного типа, доступные для добавления в плейлист
	12. Отображает аудиофайлы для добавления в плейлист
13. Нажимает кнопку «Добавить» у трека, который хочет добавить в плейлист	
	14. Добавляет выбранный аудиофайл в плейлист
	15. Меняет название кнопки «Добавить» на «Удалить» у добавленного аудиофайла
Шаги 13-15 могут повторяться до тех пор, пока пользователь не добавит все необходимые аудиофайлы или аудиофайлы для добавления не закончатся	
16. Нажимает кнопку «Удалить» у трека, который не хочет добавлять в плейлист	
	17. Удаляет выбранный аудиофайл из плейлиста
	18. Меняет название кнопки «Удалить» на «Добавить» у выбранного аудиофайла

Окончание таблицы 1

Шаги 16-18 являются необязательными и могут повторяться до тех пор, пока пользователь не удалит все необходимые аудиофайлы или аудиофайлов для удаления не останется	
19. Нажимает кнопку «Готово»	
	20. Отображает созданный плейлист
Расширения	
Расширение шагов 1-2. Пользователь выбирает раздел «Аудиокниги»	
	1.1. Устанавливает тип аудиофайла - глава аудиокниги
Расширение шагов 1-2. Пользователь выбирает раздел «Подкасты»	
	1.1. Устанавливает тип аудиофайла - выпуск подкаста
Расширение шагов 1-2. Пользователь выбирает раздел «Радио»	
	1.1. Устанавливает тип аудиофайла - выпуск радиопрограммы
Расширение шага 9. Приложение проверяет название плейлиста, название - пустое	
	9.1. Изменяет название плейлиста на базовое «Новый плейлист»
Расширение шага 11. Сервер не вернул треки, доступные для добавления	
	11.1. Отображает ошибку отсутствия треков
Расширение шага 13. Пользователь хочет уточнить список треков	
13.1. Пользователь нажимает на строку поиска	
13.2. Пользователь вводит запрос для поиска	
	13.3. Проверяет запрос на пустоту, запрос не пустой
	13.4. Проверяет корректность запроса, запрос корректен
	13.5. Запрашивает аудиофайлы определенного типа, доступные для добавления в плейлист с запросом пользователя
	13.6. Обновляет отображенные аудиофайлы на новые
Расширение шага 13 может повторятся неограниченное количество раз	

Помимо функциональных требований, были зафиксированы и нефункциональные требования:

1) приложение должно работать с уже существующим серверным REST API [2];
2) дизайн приложения должен быть реализован в соответствии с макетами;
3) данные, полученные от сервера, такие как лента, должны кэшироваться, чтобы уменьшить нагрузку на сервер и время ожидания пользователя;
4) целевая операционная система – iOS 13.0 и новее.

1.2 Модель предметной области

Проанализировав требования к системе, описанные выше, а также бизнес-процессы компании Заказчика, была составлена модель предметной области (*Рисунок 5*).

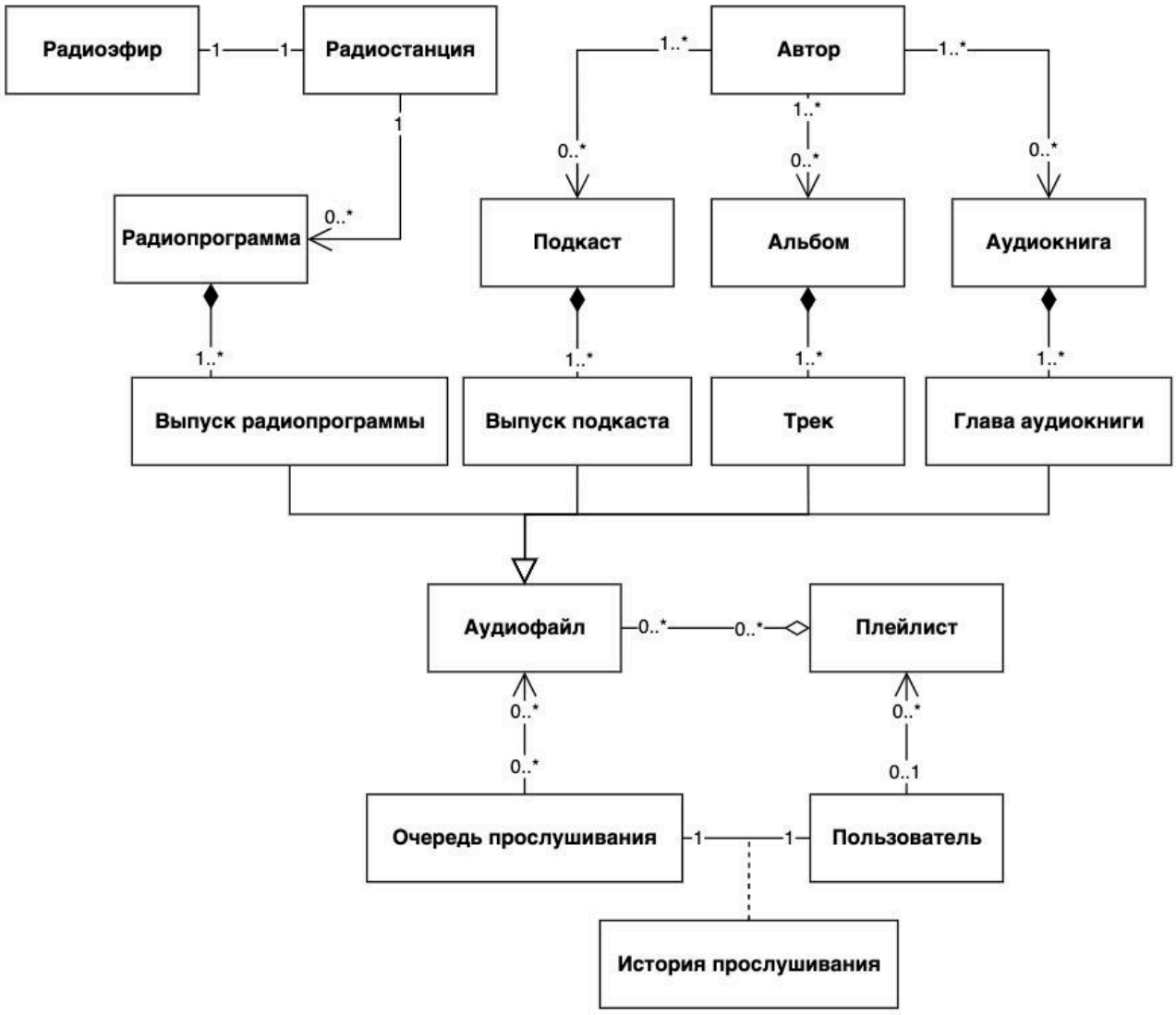


Рисунок 5 - Модель предметной области

Из аудиофайлов разных видов: трек, выпуск подкаста, глава аудиокниги, выпуск радиопрограммы, формируется очередь прослушивания. При добавлении в очередь родительского объекта, в очередь добавляются все ее дочерние элементы. Например, если добавить подкаст в очередь, то добавятся все выпуски данного подкаста. Ассоциацией между пользователем и очередью прослушивания является история прослушивания. Каждой радиостанции соответствует один прямой радиоэфир. Помимо пользовательских плейлистов, системой поддерживаются системные плейлисты, которые создаются автоматически в зависимости от предпочтений пользователя.

2 Проектирование

2.1 Архитектура MVVM

MVVM расшифровывается как Model View ViewModel [3]. Это три разных слоя, которые работают вместе, чтобы упростить управление и тестирование пользовательского интерфейса (*Рисунок 6*).

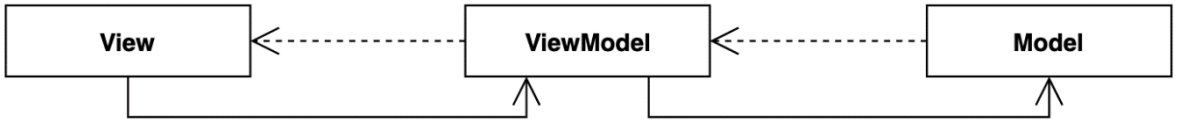


Рисунок 6 - Архитектура MVVM

Эта диаграмма показывает, как связаны компоненты и поток (*flow*) их входов / выходов. Рассмотрим каждый из них и их общие характеристики.

Model (Модель) - слой, отвечающий за работу с данными в приложении. Данный слой представляет собой набор классов или структур, описывающих структуру объектов предметной области. В полях экземпляров этих классов и структур хранятся данные, которые приложение получает от сервера. Также, в слое модели может происходить обмен данных с сервером, кэширование и т. д., однако в данном приложении классы, решающие эти задачи, вынесены в отдельный слой сервисов, описание работы которого приведено далее.

ViewModel (Модель представления) — связующее звено между слоем модели и слоем представления. Она знает Model, но не знает View. ViewModel берет части данных из модели и преобразует их в значения, подходящие для пользовательского интерфейса. Она также предоставляет некоторые функции, которые можно вызывать для выполнения определенных задач, и которые обычно вызываются действиями пользователя. Наконец, ViewModel может вызывать события, которые могут уведомлять другие части приложения о произошедшем изменении.

Их можно использовать для запуска таких вещей, как переход к другим экранам.

View (Представление) — это пользовательский интерфейс. Это слой, который отображает контент и принимает вводимые пользователем данные. В iOS это могут быть пользовательские UIView, ячейки представления таблиц и коллекций, а также UIViewControllers. Представление знает ViewModel, но ничего больше. Оно будет использовать значения ViewModel для заполнения пользовательского интерфейса, и когда пользователь выполняет действие, оно будет перенаправлять его в соответствующую функцию ViewModel для выполнения работы.

Важной особенностью архитектуры MVVM является то, что объекты слоя модели представления могут хранить ссылки на объекты классов модели, однако не хранят ссылки на объекты классов представления. Для передачи данных из модели представления в представление используются различные механизмы — например, замыкания (*Closures*) [4]. Объекты слоя представления имеют ссылки на соответствующие объекты моделей представления для извещения о событиях интерфейса и вызова прочих методов.

2.2 Координаторы

Архитектура MVVM решает проблему разделения ответственности для конкретных экранов приложения или компонентов интерфейса. Однако, необходимо определить, какие классы будут отвечать за создание и настройку классов представления и модели представления для новых экранов, а также за логику навигации и переходов между экранами в целом. Для решения этой проблемы в приложении были использованы Координаторы.

Координатор — это объект, который управляет одним или несколькими ViewController-ами.

Все начинается с координатора приложения. Координатор приложения решает проблему переполненного делегата приложения. Координатор приложения настраивает основной ViewController приложения. Он может создавать и настраивать контроллеры представлений или создавать новые дочерние координаторы для выполнения подзадач.

Делегат приложения (AppDelegate) настраивает окно приложения и контроллер корневого представления, а затем запускает координатор приложения. Инициализация координатора отделена от начала его работы. Это позволяет создавать и запускать его тогда, когда это необходимо.

К преимуществам координаторов можно отнести следующее:

1. Каждый ViewController изолирован

ViewController-ы ничего не знают, кроме того, как представлять свои данные. Когда что-нибудь происходит, ViewController сообщает об этом своему делегату, не зная, кто его делегат.

2. ViewController-ы переиспользуемые

Они ничего не предполагают о том, в каком контексте они будут представлены или для чего будут использоваться их кнопки. Их можно использовать и переиспользовать для улучшения внешнего вида.

3. Координаторы — это полностью контролируемые объекты

Чтобы работать, не нужно ждать вызова *-viewDidLoad* во ViewController. Поведение приложения полностью прозрачно, и UIKit теперь просто библиотека, которую мы используем.

Координаторы помогают сделать приложение и код более управляемыми. Контроллеры представления станут более многоразовыми, а развитие приложения становится проще.

Процесс работы координаторов и настройки экранов в общем виде представлен ниже (*Рисунок 7*).

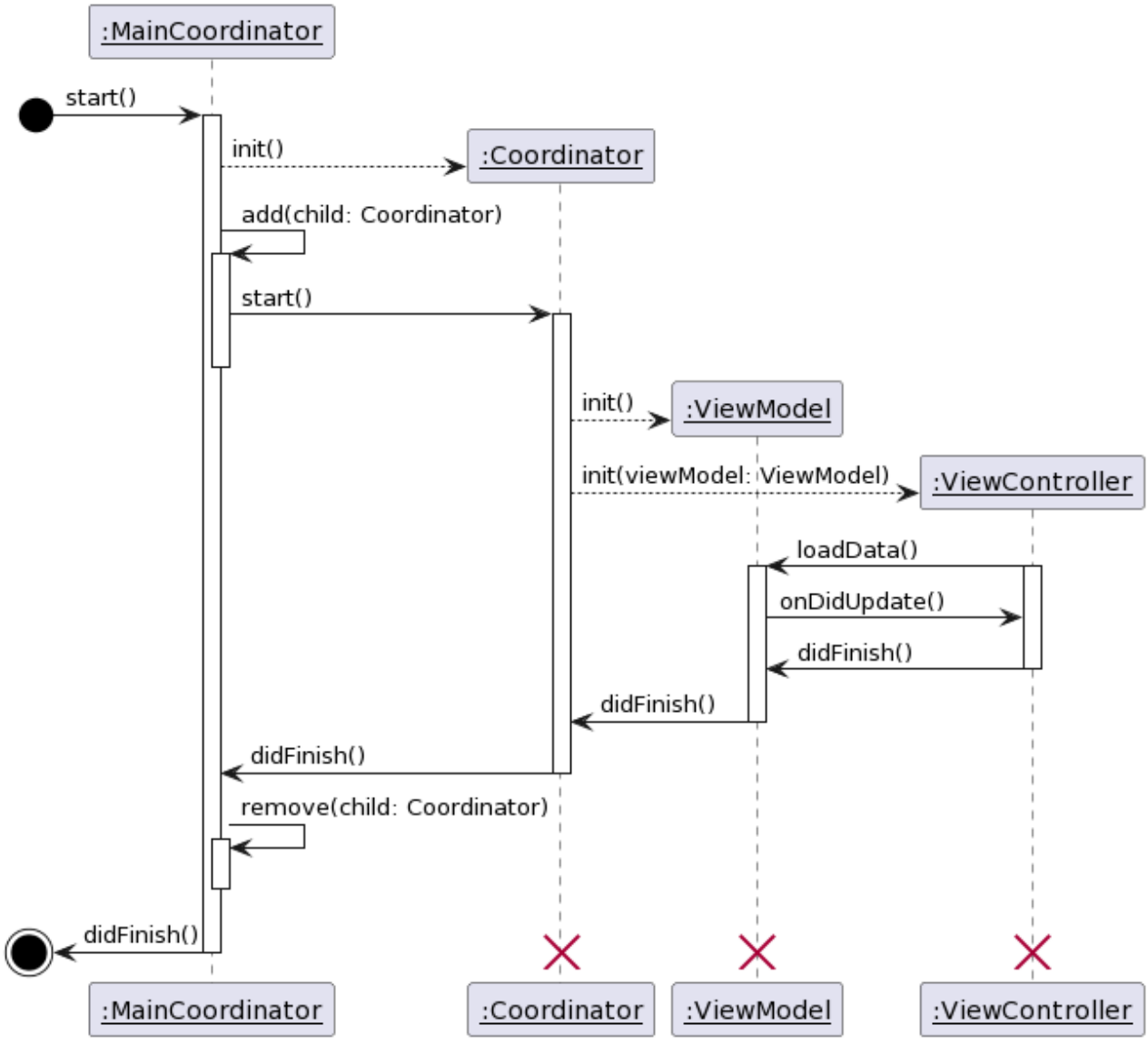


Рисунок 7 - Работа координатора

Каждый координатор должен иметь массив дочерних координаторов – *childCoordinators*, а также метод *start()*, который запускает координатор и отображает следующий экран в приложении. Замыкание *didFinish* необходимо вызывать, когда координатор завершает свою работу, и экраны, отображаемые им, закрываются, чтобы известить родительский координатор о том, что текущий координатор можно удалить из памяти.

Учитывая координаторы, экраны приложения следует реализовать в виде набора классов (*Рисунок 8*).

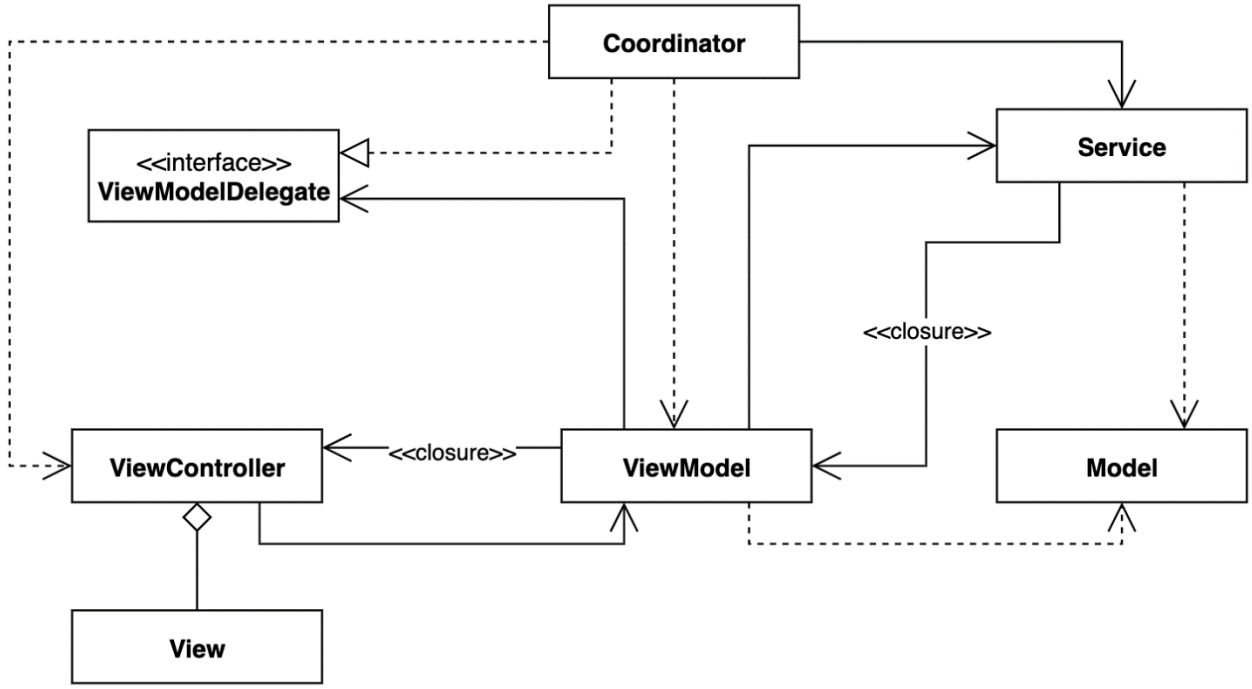


Рисунок 8 - Архитектура MVVM + Coordinator

Данная диаграмма иллюстрирует особенности реализации архитектуры MVVM в данном приложении, с учетом использования паттерна «Координатор» и слоя сервисов. Если возникает необходимость вызывать методы координатора из класса модели представления, используется паттерн «Делегирование» [5]. Класс модели представления содержит ссылку на объект, реализующий интерфейс делегата данного класса. Таким делегатом выступает координатор и при инициализации экземпляра класса модели представления передает ему ссылку на самого себя.

2.3 Внедрение зависимостей

В приложении для передачи объектов классов-сервисов в модели представления экранов используется шаблон проектирования, при котором поля или параметры создания объекта конфигурируется извне (Dependency Injection [6]).

Данный механизм предполагает передачу зависимостей через конструктор при инициализации объектов, что позволяет уменьшить зависимость между слоем сервисов и слоем модели представления.

В данном приложении, для реализации механизма внедрения зависимостей, был создан класс AppDependency, который создает объекты необходимых сервисов и хранит ссылки на них. Объект данного класса инициализируется при старте приложения в объекте класса AppDelegate — точке входа в приложение. Далее, созданный экземпляр класса AppDependency через цепочку координаторов передается в конструктор соответствующих моделей представления. Однако, в модели представления нет необходимости иметь ссылки на все существующие в приложении зависимости. В данном проекте для каждой зависимости описан интерфейс, который обязывает конкретную модель представления иметь ссылку на экземпляр класса данного сервиса (Листинг *1*).

```
protocol HasProfileService {
  var profileService: ProfileNetworkProtocol { get }
}

protocol HasAudioPlayerController {
  var audioPlayerController: AudioPlayerController { get }
}
```

Листинг 1 - Интерфейс сервиса

2.4 Пакеты приложения

Согласно диаграмме основных пакетов данного приложения (*Рисунок 9*), главный пакет в приложении - *Modules*. В нём в упорядоченном виде хранятся классы, описывающие экраны приложения. В пакете *Views* хранятся общие компоненты для нескольких экранов. *Utils* - вспомогательные классы, например классы для управления всплывающими баннерами, уведомлениями, валидаторы для текстовых полей и другое. Пакет *Services* содержит сервисы для запросов на сервер, интеграций с другими приложениями, аналитики и т. п.

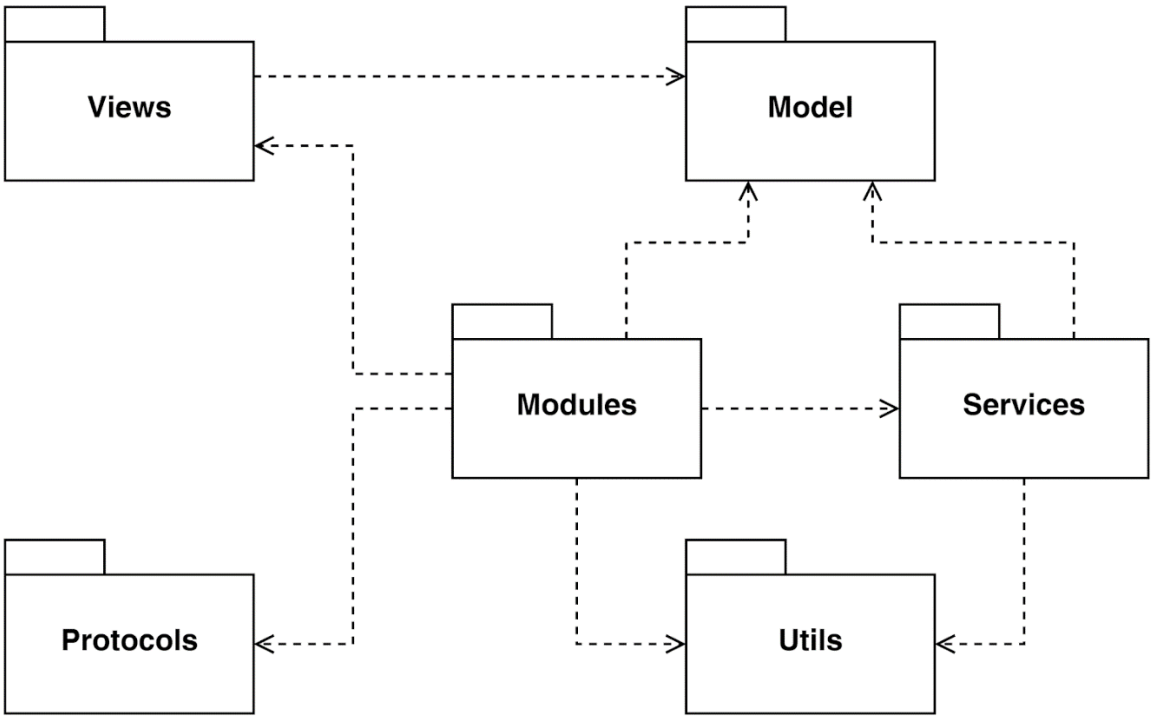


Рисунок 9 - Пакеты приложения

3 РЕАЛИЗАЦИЯ

3.1 Используемые технологии

Для разработки приложения была выбрана среда разработки Xcode 13 от Apple, которая предоставляет доступ ко всем основным функциям, необходимым для создания мобильного приложения для iOS. Этот инструмент предоставляет разработчикам основные необходимые функции для написания кода и тестирования пользовательского интерфейса.

Для проектирования использовались **Draw.io** [7] и **PlantUML** [8] - инструменты для создания диаграмм, блок-схем и другого.

Язык программирования - **Swift 5** [9].

В качестве менеджера зависимостей выбор пал на **CocoaPods** [10], так как его аналог **Swift Package Manage**r [11] не имеет необходимых зависимостей для выбранных библиотек (*Таблица 2*).

Таблица 2 - Сторонние библиотеки

№	Название	Описание
1	Alamofire [12]	Библиотека для выполнения сетевых запросов
2	SnapKit [13]	Библиотека, упрощающая создание элементов интерфейса из кода
3	PromiseKit [14]	Библиотека для упрощенной работы с асинхронными вызовами методов
4	SwiftLint [15]	Утилита для автоматического контроля качества и стиля кода
5	R.swift [16]	Генератор статических ресурсов (локализованные строки, ресурсы из Assets)
6	SwiftyBeaver [17]	Утилита для более удобного форматирования и отображения данных в консоли при разработке
7	Kingfisher [18]	Библиотека для загрузки и кэширования изображений по ссылке
8	NotificationBannerSwift [19]	Библиотека для отображения уведомлений пользователю
9	DominantColor [20]	Библиотека для вычисления среднего цвета изображения
10	KeychainAccess [21]	Библиотека для упрощенной работы с защищенным хранилищем Keychain.

Окончание таблицы 1

11	Firebase [22]	Библиотека для интеграции приложения с платформой Firebase. Firebase — это облачная платформа, предоставляющая такие сервисы, как облачная СУБД, хранилище данных, инструменты для работы с машинным обучением, аналитикой активности пользователей и другие
12	Firebase/Crashlytics [22]	Библиотека для отслеживания экстренных завершений приложения
13	Firebase/Analytics [22]	Библиотека для интеграции продуктовой аналитики в приложение
14	Lottie [23]	Библиотека для работы с анимациями
15	InfiniteScroll [24]	Библиотека для работы с пагинацией
16	Startrek [25]	Библиотека для проигрывания радиоэфиров

Для хранения кодовой базы и совместной работы над проектом использовалась система контроля версий Git в сочетании с системой управления репозиториями GitHub [26].

3.2 Взаимодействие с сервером

Для запросов на сервер существует нативный инструмент — URLSession [37], но работать с ним немного сложнее, чем хотелось бы. Для облегчения этого процесса существует библиотека Alamofire [12]— это обертка над URLSession, которая сильно упрощает взаимодействие с сервером. Alamofire содержит дополнительную логику помимо простого создания сетевого запроса, упрощающие работу с сетевыми запросами. Например, повторный запрос, когда основной запрос завершается сбоем, например, из-за сбоя аутентификации. Возможно легко обновить токен аутентификации и снова вызвать тот же запрос, не затрагивая код реализации. Так же синтаксис для создания запросов намного элегантнее и проще в использовании. Это избавляет от большого количества лишнего кода и значительно упрощает проверку и обработку ошибок.

Асинхронное программирование может привести к огромным делегатам, беспорядочным обработчикам завершения и долгой отладки кода. Но есть лучший способ: обещания. Обещания позволяют писать код в виде последовательности действий, основанных на событиях. Это особенно хорошо работает для действий, которые должны выполняться в

определенном порядке. Для работы с асинхронными вызовами, на проекте использовалась библиотека PromiseKit [14], в которой уже реализовано удобное взаимодействие с обещаниями.

Promise (Обещание) — это объект, представляющий асинхронную задачу. У Promise есть блок *then*, который похож на обработчик завершения *onSuccess* [27]. Он также имеет блок *catch*, который отделяет обработку ошибок от основного пути кода, и который похож на обработчик завершения *onFailure* (Листинг *2*).

```
private func handleResponse<T>(json: Any) -> Promise<T> where T: Decodable {
  var jsonObject: Any? = T.self == EmptyResponse.self &&
    json as? [String: Any] == nil ? [:] : json as? [String: Any]
  if let array = json as? [Any] {
    jsonObject = array
  }
  return Promise<T> { seal in
    if let jsonObject = jsonObject {
      let jsonData = try JSONSerialization.data(withJSONObject: jsonObject,
                                                options: .prettyPrinted)
      firstly {
        self.handleSuccessResponse(jsonData: jsonData)
      }.done { response in
        seal.fulfill(response)
       }.catch { error in
        seal.reject(error)
      }
    } else {
      seal.reject(NetworkErrorService.parseError)
    }
  }
}
```

Листинг 2 - Пример использования promise при серверном запросе

3.3 Кэширование запросов

Время ответа на запросы получения фида (список блоков) с сервера может быть достаточно большим. Помимо этого, при частых запросах на сервер создается высокая нагрузка.

Чтобы уменьшить время ожидания пользователя и снизить нагрузку на сервер, было решено сохранять в память смартфона (кэшировать) фид главной витрины и коллекции пользователя. Обнуление кэша регулируется на стороне сервера.

Для решения этой задачи был создан класс Storage (Листинг *3*). Данный класс занимается сохранением данных в файловую систему мобильного устройства и чтением их из памяти.

Основным в классе является generic-метод setObject(), который сохраняет в файловую систему любые объекты, чей тип реализует интерфейс Codable [28].

```
func setObject(_ object: T, forKey key: String) throws {
    let fileURL = try directoryURL()
                        .appendingPathComponent(key,
                                                isDirectory: false)
    let data = try JSONEncoder().encode(object)
    if fileManager.fileExists(atPath: fileURL.path) {
      try fileManager.removeItem(at: fileURL)
    }
    let result = fileManager.createFile(atPath: fileURL.path,
                                        contents: data)
    guard result else { throw StorageError.couldNotSaveFile }
  }
```

Листинг 3 – Метод кэширования объектов в память смартфона

Метод setObject() принимает два параметра: object (объект, который будет сохранен в файловую систему) и key (уникальный ключ). Данные кодируются в JSON-формат при помощи класса JSONEncoder и записываются в файловую систему с помощью класса FileManager.

Для получения сохраненных в памяти данных на основании ключа реализован generic-метод object() (Листинг *4*). Данный метод возвращает сохраненные в памяти устройства данные, если они актуальны, или ошибку в случае, если данных нет.

```
func object(forKey key: String) throws -> T {
    let fileURL = try directoryURL()
                        .appendingPathComponent(key,
                                                isDirectory: false)
    guard let data = fileManager.contents(atPath: fileURL.path)
    else { throw StorageError.noDataSaved }
    return try JSONDecoder().decode(T.self, from: data)
 }
```

Листинг 4 – Метод получения кэшированных объектов из памяти смартфона

Для кэширования запроса и чтения уже существующего кэша, был создан класс RequestCache (*Рисунок 10*) и соответствующие методы cacheResponse() (Листинг *5*) и fetchCachedResponse() (Листинг *6*). Данный класс занимается сохранением кэша в файловую систему мобильного устройств, чтением его из памяти, а также управлением временем жизни кэша.

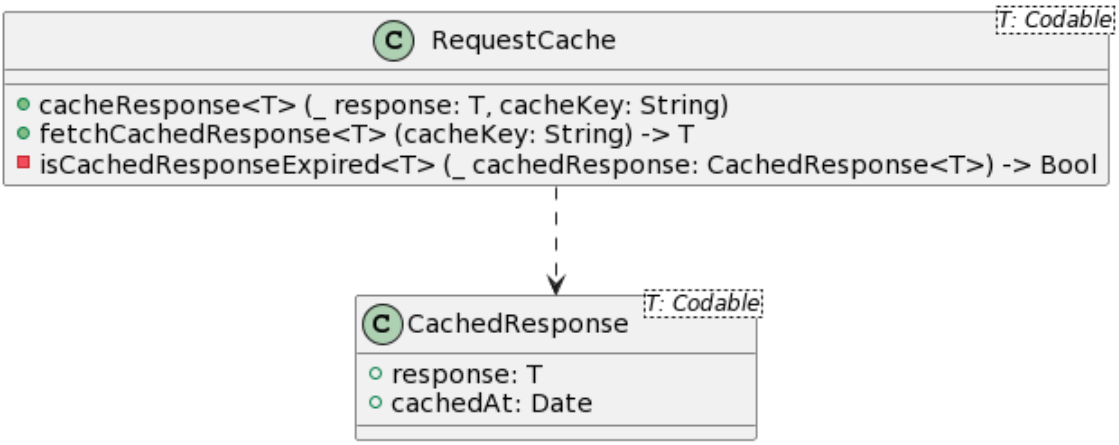


Рисунок 10 – Класс RequestCache

```
func cacheResponse<T: Codable>(_ response: T,
                                    cacheKey: String) throws {
    let cachedResponse = CachedResponse(response: response,
                                        cachedAt: Date())
    let storage = Storage<CachedResponse<T>>()
    try storage.setObject(cachedResponse, forKey: cacheKey)
}
```

Листинг 5 – Метод сохранения кэша в память устройства

```
func fetchCachedResponse<T: Codable>(cacheKey: String) throws -> T? {
    let storage = Storage<CachedResponse<T>>()
    do {
      let cachedResponse = try storage.object(forKey: cacheKey)
      if isCachedResponseExpired(cachedResponse) {
        try storage.remove(forKey: cacheKey)
        return nil
      } else {
        return cachedResponse.response
      }
    } catch {
      guard let storageError = error as? StorageError,
                storageError == StorageError.noDataSaved else {
        throw error
      }
      return nil
    }
}
```

Листинг 6 – Метод получения кэша из памяти

3.4 Формирование главного экрана

После входа в приложение, пользователь попадает на главную витрину, состоящую из контента различных видов: подборки книг, релизы альбомов, рекомендации по жанрам, новые выпуски. Главный экран состоит из набора блоков, которые приходят с сервера.

В силу того, что запрос получения ленты достаточно время затратный, а при возвращении в ленту из других разделов или с детальных страниц необходимо отображать актуальную информацию, было принято решение делать запрос в два этапа.

Перед основным запросом делается HEAD запрос, для уточнения актуальности текущей ленты. Если Timestamp (строка с актуальной датой контента), который пришел в заголовке ответа, актуален и есть кэш раздела, то контент отображается из кэша. Если ранее мобильное приложение не запрашивало фид, т. е. нет кэша в памяти устройства, или Timestamp, который сохранило мобильное устройство - устарел, то делается основной GET запрос фида (*Рисунок 11*).

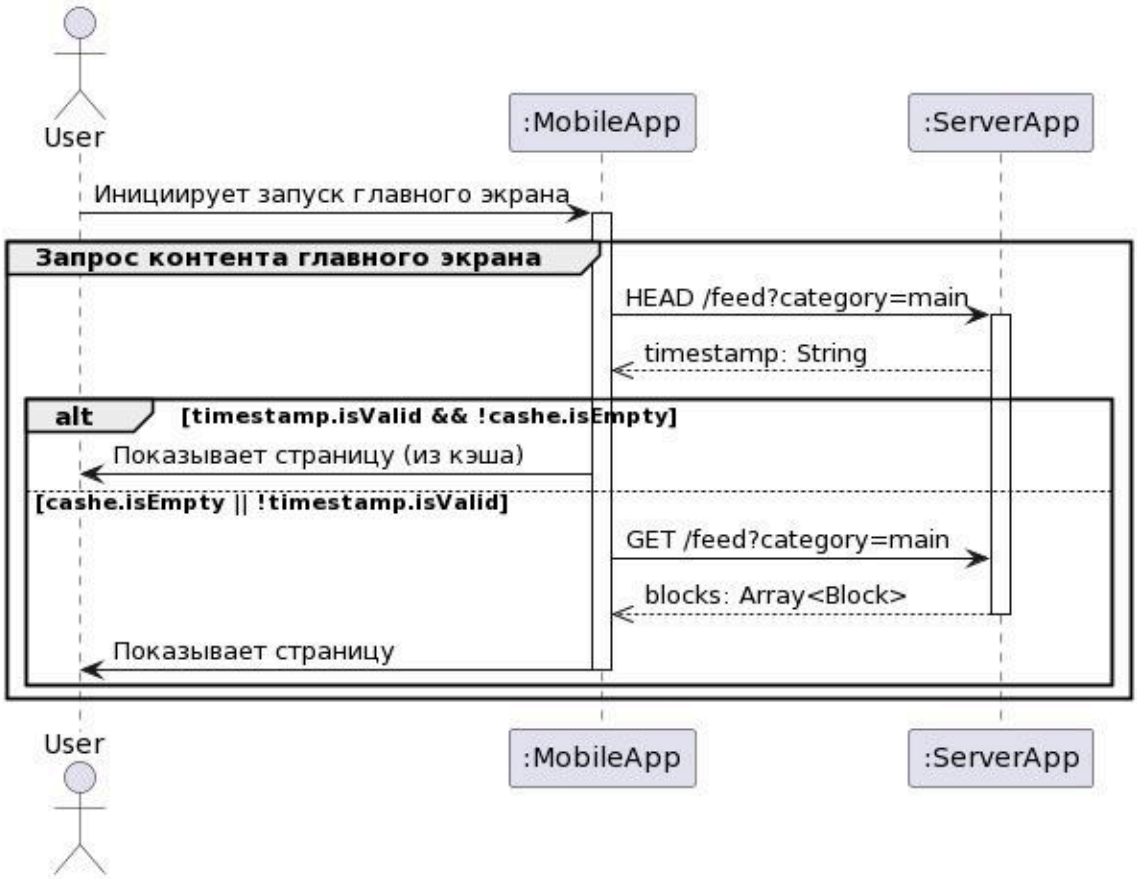


Рисунок 11 – Отображение главного экрана

Таким образом, индикатор загрузки ленты показывается один раз - при перовом входе, а при дальнейших изменениях - лента обновляется незаметно от пользователя и при необходимости.

На данный момент существует несколько видов блоков, которые представлены в таблице 3. Рассмотрим подробнее их отличительные особенности.

Таблица 3 - Виды блоков

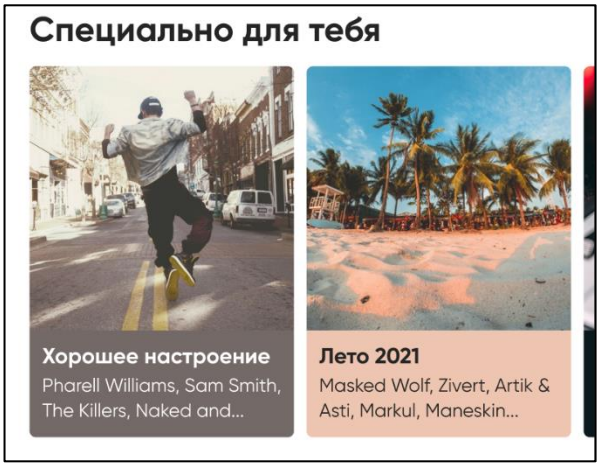
Тип блока	**Описание**
 Рисунок 12 - Блок с типом list_rectangular	Элементы блока данного типа имеют специальное отображение: - изображение квадратной формы; - описание, цвет фона которого является средним цветом изображения; цвет описания - контрастный цвет к цвету фона. Чаще всего в таких блоках отображаются системные плейлисты, на которые стоит обратить внимание

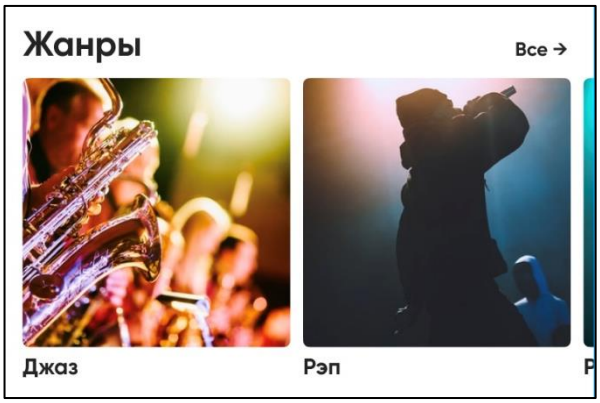
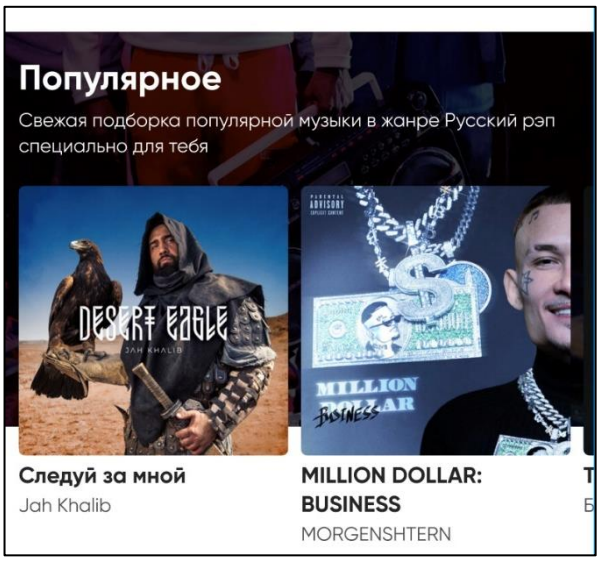
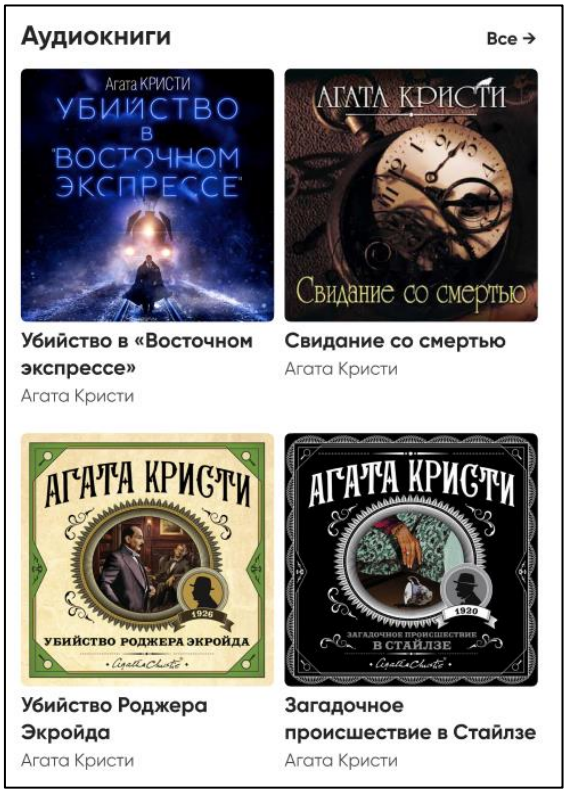
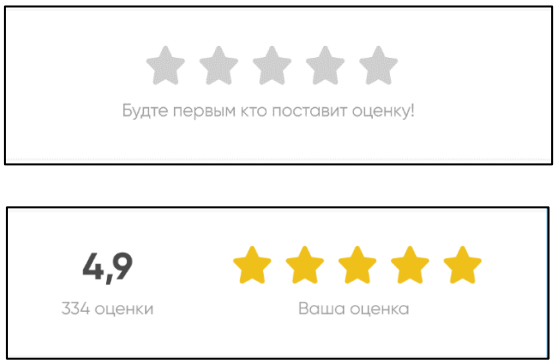
 Рисунок 13 - Блок с типом list_square	Наиболее популярный блок, является стандартным в системе. Размер изображения зависит от ширины экрана. В блоке данного типа могут отображаться как аудиофайлы для прослушивания (трек, выпуск и т. п.), так и их родительские сущности (альбом, подкаст и т. п.)
 Рисунок 14 - Блок с типом genres	Данный блок не имеет визуальных отличий от блока с типом list_square и служит для отображения жанров. Тип выделен отдельно, так как жанр не имеет детальной страницы и не доступен для воспроизведения. При переходе по жанру отображается лента с контентом данного жанра
 Рисунок 15 - Блок с типом list_image_background	Элементы данного блока не имеют отличий от элементов блока с типом list_square. В блоке данного типа добавляется фоновое изображение и описание. Цвет заголовка и описания является контрастным к цвету фонового изображения

 Рисунок 16 - Блок с типом ad	Блок данного типа не имеет заголовка и элементов и служит для отображения рекламных баннеров. Переход по баннеру ведет на страницу мероприятия через нативный браузер
  Рисунок 17 - Блоки с типом list_title_link	Элементы данного блока не имеют отличий от элементов блока с типом list_square. В заголовке блока данного типа появляется дополнительная ссылка для перехода на детальную страницу или в ленту жанра. Если ссылка ведет на автора контента, то дополнительно может отображаться его изображение

 Рисунок 18 - Блок с типом grid_four_squares	Элементы данного блока не имеют отличий от элементов блока с типом list_square. Элементы блока данного типа имеют особое расположение: вместо горизонтальной коллекции элементы отображаются плиткой, друг под другом
 Рисунок 19 - Блоки с типом rating	Блок данного типа отличается от предыдущих и требует дополнительного действия от пользователя. Данный блок отображается только на детальных страницах контента. Для отображения рейтинга делается дополнительный запрос на сервер. После проставления оценки вид блока - изменяется

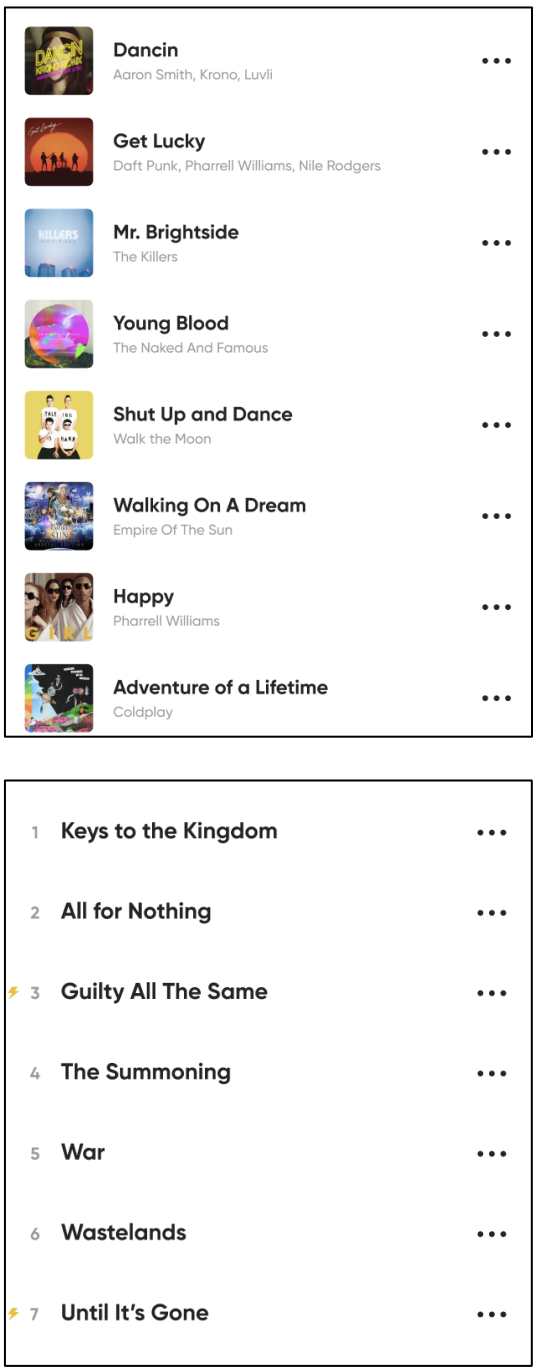


Рисунок 20 - Блоки с типом track

Элементы данного блока имеют особый списочный вид. В данном блоке могут отображаться только аудиофайлы для прослушивания (трек, выпуск подкаста, выпуск радиопрограммы, глава аудиокниги).

Если блок данного типа приходит на странице альбома, то картинка и список авторов не отображаются (т. к. они будут идентичны для всех элементов). Появляется дополнительный элемент - порядковый номер трека в альбоме

Проанализировав таблицу 3, можно заключить, что несмотря на то, что большинство блоков состоят из заголовка и набора элементов, встречаются те, у которых может не быть как заголовка, так и элементов (*Рисунок 21*).

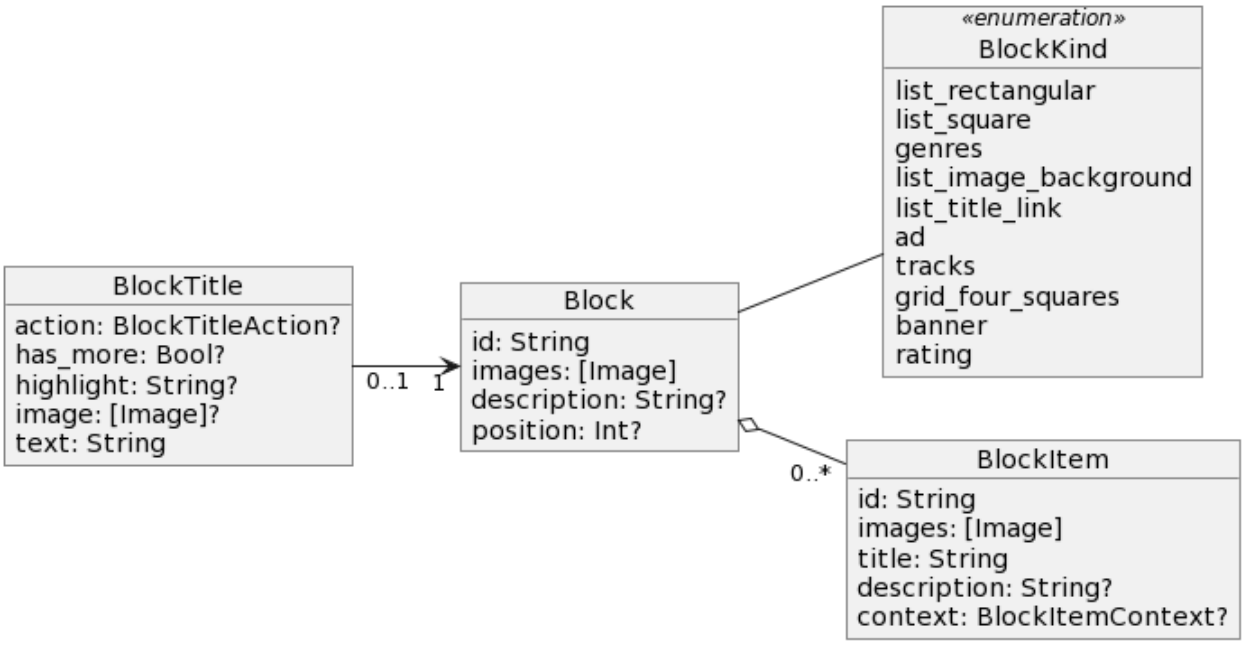


Рисунок 21 – Устройство блока

У класса BlockTitle есть опциональное поле highlight, в зависимости от наличия которого, заголовок отображается иначе. Также у заголовка может быть дополнительное действие для перехода, например, на детальную страницу автора контента. Параметр has_more определяет необходимость показа кнопки «Все» для дальнейшего перехода ко всем элементам блока. (*Рисунок 22*).

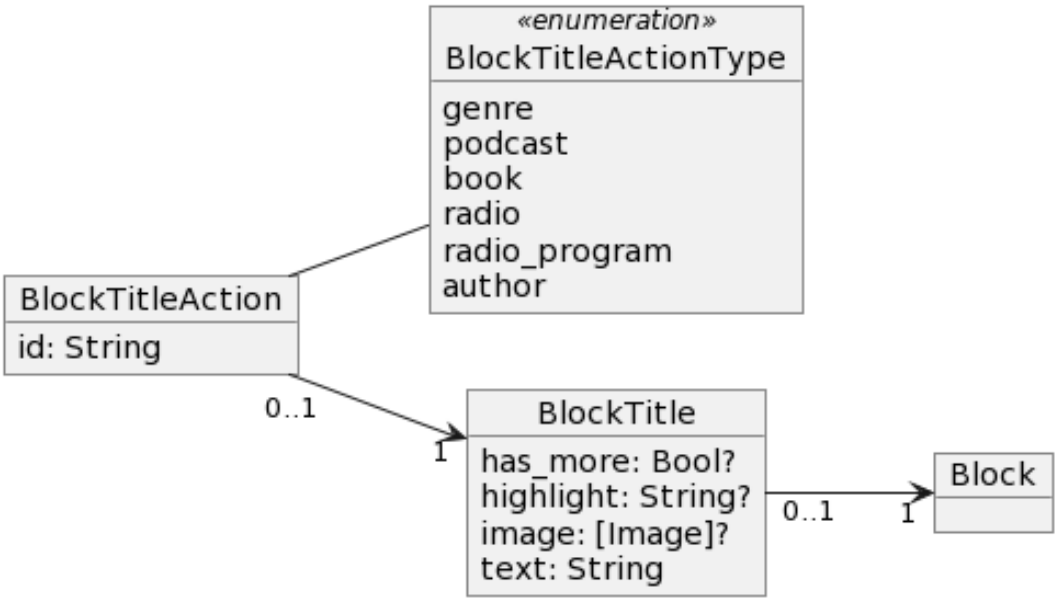


Рисунок 22 – Устройство заголовка блока

В зависимости от типа элемента в коллекции, мы можем как напрямую перейти к прослушиванию контента (трек, выпуск подкаста и др.), так и перейти на детальную страницу (*Рисунок 23*). Для некоторых типов, таких как ad (рекламный баннер) и banner (баннер рекомендательной системы), предусмотрены индивидуальные действия: переход на страницу мероприятия через нативный браузер и переход к системе улучшения рекомендаций, соответственно. Учитывая тип элемента (BlockItemKind) и его контекст (BlockItemContext), детальная информация формируется по-разному. К примеру, у трека отображается автор, а у выпуска подкаста - дата публикации и продолжительность. Существует четыре размера картинок: thumbnail - самая маленькая, для отображения в элементах списка и заголовке; medium - средняя, для отображения в горизонтальных коллекциях; large - большая, для отображения в детальных страницах; original - первоначальный размер картинки. Полное устройство блока показано в Приложении Б.

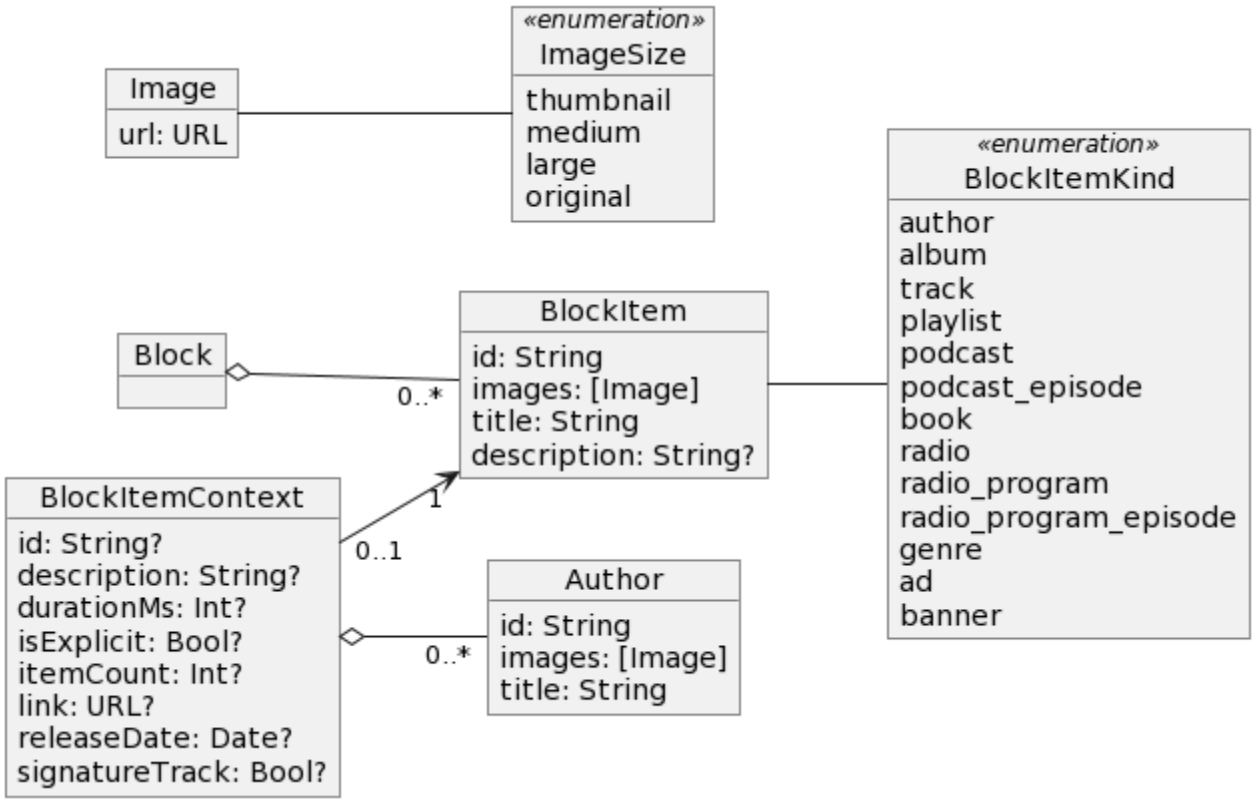


Рисунок 23 – Устройство элемента блока

По ходу разработки, мы столкнулись с проблемой использования стандартной коллекции. С учетом того, что блоки приходят с сервера, мы не можем предугадать конечный размер контента, который придет. Из-за этого высота некоторых ячеек отличалась, в связи с чем появлялся дополнительный пробел между блоками (*Рисунок 24*). Для решения данной проблемы был написан протокол ListViewHaving (Листинг *7*), при помощи которого рассчитывалась высота контента и затем вычитался лишний отступ (Листинг *8*). Высота контента складывается из размера изображения и суммы заголовка (две строки максимум) и описания (две строки максимум).

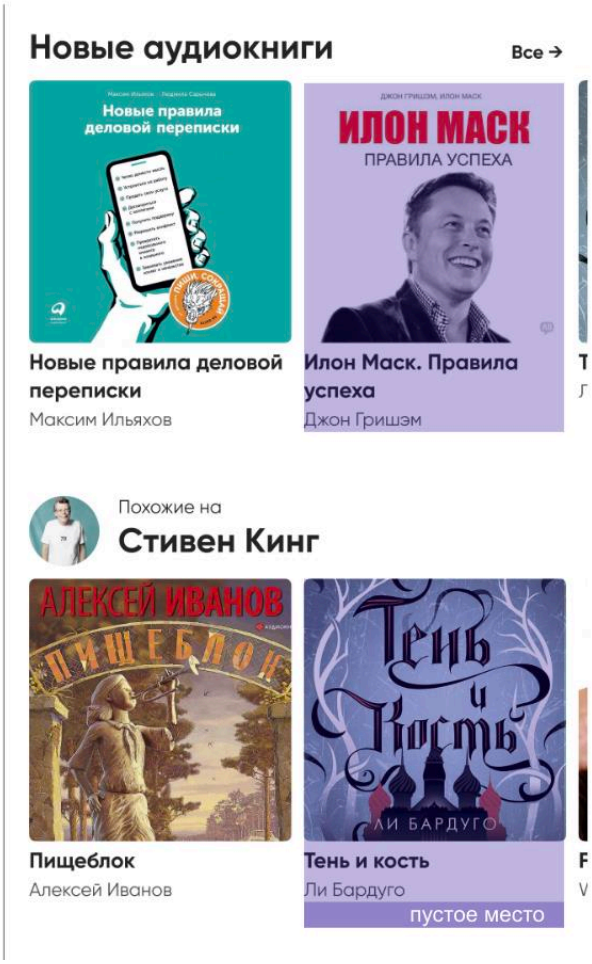


Рисунок 24 - Пример образования пустого пространства при коротких данных

```
protocol ListViewHaving {
var itemWidth: CGFloat { get }
      func getCollectionViewOffset(blockItems: [BlockItem]) -> CGFloat
}
```

Листинг 7 – Параметры и методы протокола ListViewHaving

```
func getCollectionViewOffset(blockItems: [BlockItem]) -> CGFloat {
    var titleHeight: CGFloat = 0
    var descriptionHeight: CGFloat = 0

    blockItems.forEach { item in
      /// получаем текущую высоту
      let newTitleHeight = item.title?.height(for: itemWidth,
     font: .bodyBold
      let newDescriptionHeight = item.description?.height(for: itemWidth,
                                                          font: .footnote)
      /// получаем наибольшую высоту ячеек
      titleHeight = max(titleHeight, newTitleHeight)
      descriptionHeight = max(descriptionHeight, newDescriptionHeight)
    }
    /// ограничиваемся максимально допустимой высотой
    /// допускается 2 строки для заголовка и 2 строки для описания
    titleHeight = min(titleHeight, maxTitleHeight)
    descriptionHeight = min(descriptionHeight, maxDescriptionHeight)

    let totalHeight = titleHeight + descriptionHeight
    /// получаем величину нижнего отступа
    if totalHeight >= maxHeight + maxDescriptionHeight {
      return 0
    } else if totalHeight >= maxTitleHeight + oneLineDescriptionHeight {
      return Constants.fourLinesCellHeight - Constants.threeLinesCellHeight
    } else if totalHeight > oneLineTitleHeight {
      return Constants.fourLinesCellHeight - Constants.twoLinesCellHeight
    } else {
      return Constants.fourLinesCellHeight - Constants.oneLineCellHeight
    }
}
```

Листинг 8 – Реализация метода getCollectionViewOffset()

Для реализации протокола ListViewHaving, был написан класс HomeContainerView, который также реализует протокол для конфигурации ячеек таблицы - CommonTableCellViewModel (*Рисунок 25*). Протокол CommonTableCellViewModel содержит базовые параметры и методы для работы с ячейкой таблицы.

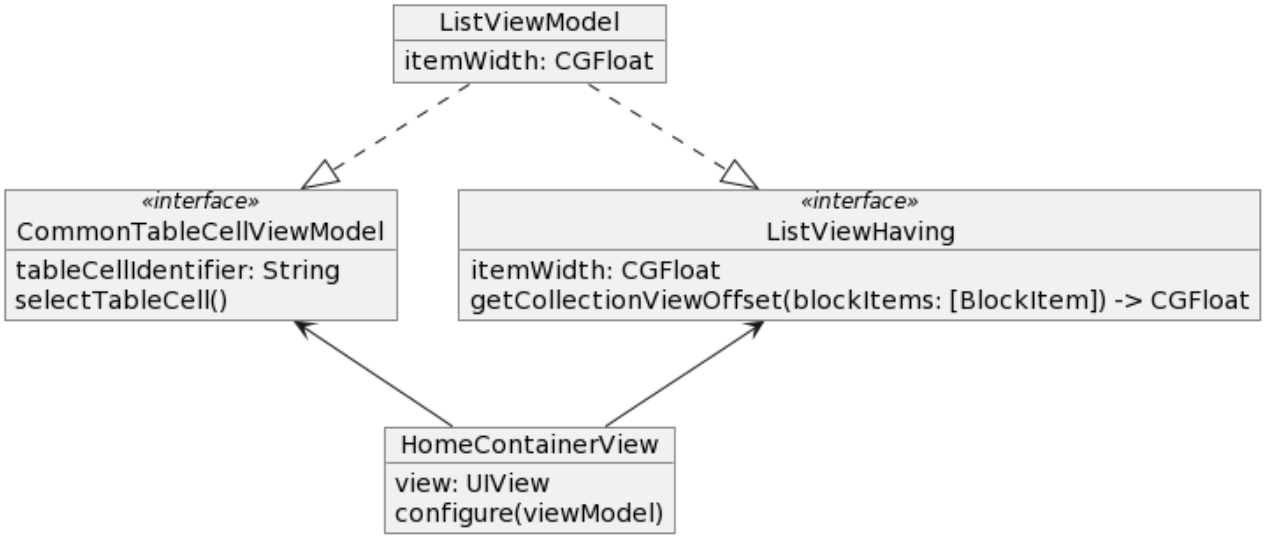


Рисунок 25 – Реализация протокола ListViewHaving

3.5 Формирование детальных страниц

Одной из важных частей приложения являются детальные страницы. Детальные страницы доступны для контента с типом:

- автор;
- плейлист;
- альбом;
- подкаст;
- выпуск подкаста;
- радиопрограмма;
- выпуск радиопрограммы;
- радиостанция.

В зависимости от типа контента, детальные страницы отображаются по-разному. Визуально экран можно разделить на три части: шапка - верхняя половина экрана, основное изображение; контролы воспроизведения - управление проигрыванием, управление лайками, переход к комментариям; описание - основная информация о контенте. Структура классов формирования детальных страниц представлена на рисунке 26. Была написана фабрика, при помощи которой создается необходимый для данного типа контента набор ячеек для отображения.

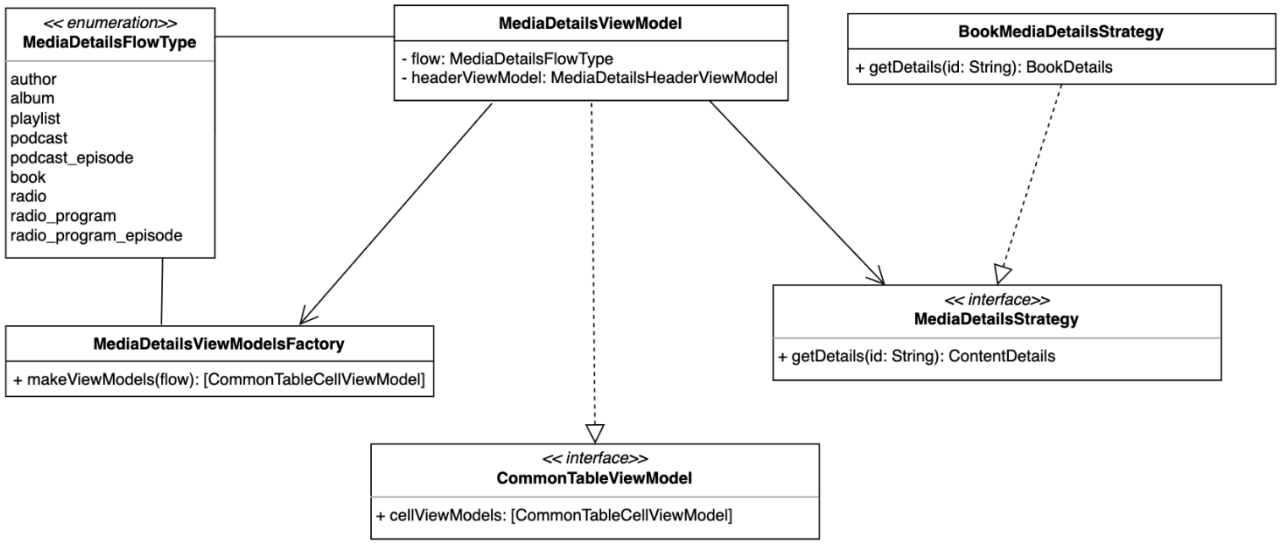


Рисунок 26 – Формирование ячеек детальной страницы

При вертикальном скролле экрана, шапка плавно, с анимацией сворачивается, картинка сменяется ее доминантным цветом, который высчитывается при помощи сторонней библиотеки **DominantColor** [20]. Доскролливание до развернутого/свернутого состояний происходит автоматически, даже если на экране недостаточно для этого контента (*Рисунок 27*). Для этого в таблицу добавляется пустая ячейка, высота которой высчитывается по формуле: высота экрана - высота контента.

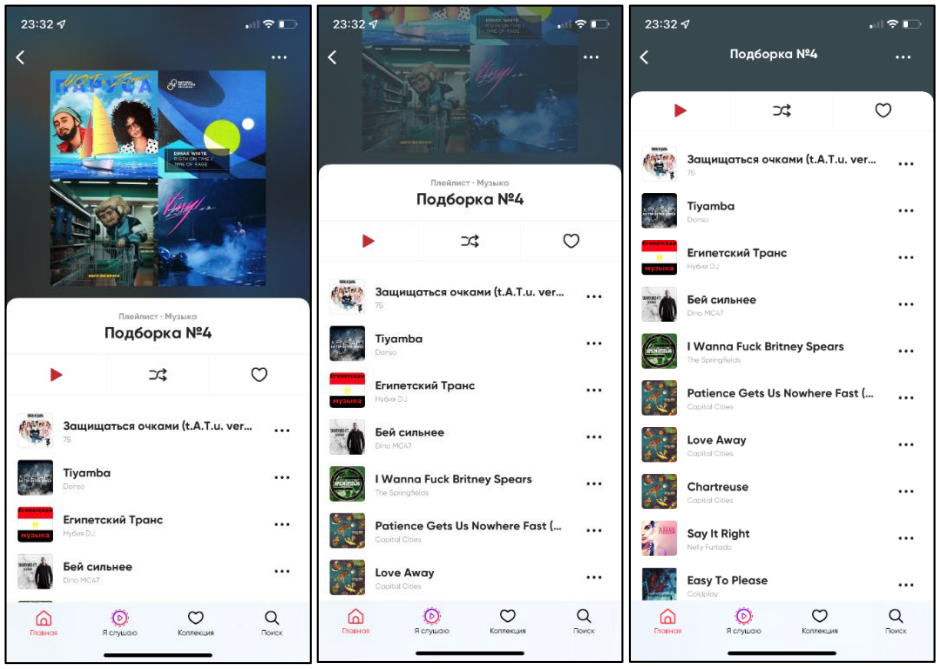


Рисунок 27 – Состояния экрана при скролле

3.6 Плеер

Неотъемлемой частью приложения является плеер. Плеер позволяет пользователю управлять проигрыванием аудиофайлов. В приложении доступно два вида плееров. Мини плеер обладает минимально необходимым функционалом и отображается на всех экранах МП (*Рисунок 28*). По нажатию на мини плеер мы попадаем в большой плеер (*Рисунок 29*), который обладает расширенным функционалом. Внешний вид большого плеера и его функциональность изменяется для разных типов аудиофайлов. Для главы аудиокниги, выпуска подкаста и выпуска радиопрограммы добавляются кнопки для перемотки и изменения скорости проигрывания.

Рисунок 29 - Скриншот большого плеера

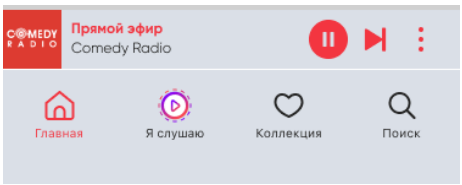


Рисунок 28 - Скриншот мини плеера

Для управления состоянием проигрывания и очередью был реализован класс AudioPlayerController (*Рисунок 30*). Данный класс взаимодействует с различными сервисами, которые подключены через механизм внедрения зависимостей. У класса есть набор делегатов, поэтому любой класс может подписаться на изменения в плеере: сменился текущий аудиофайл, изменилось состояние проигрывания или изменилось время проигрывания.

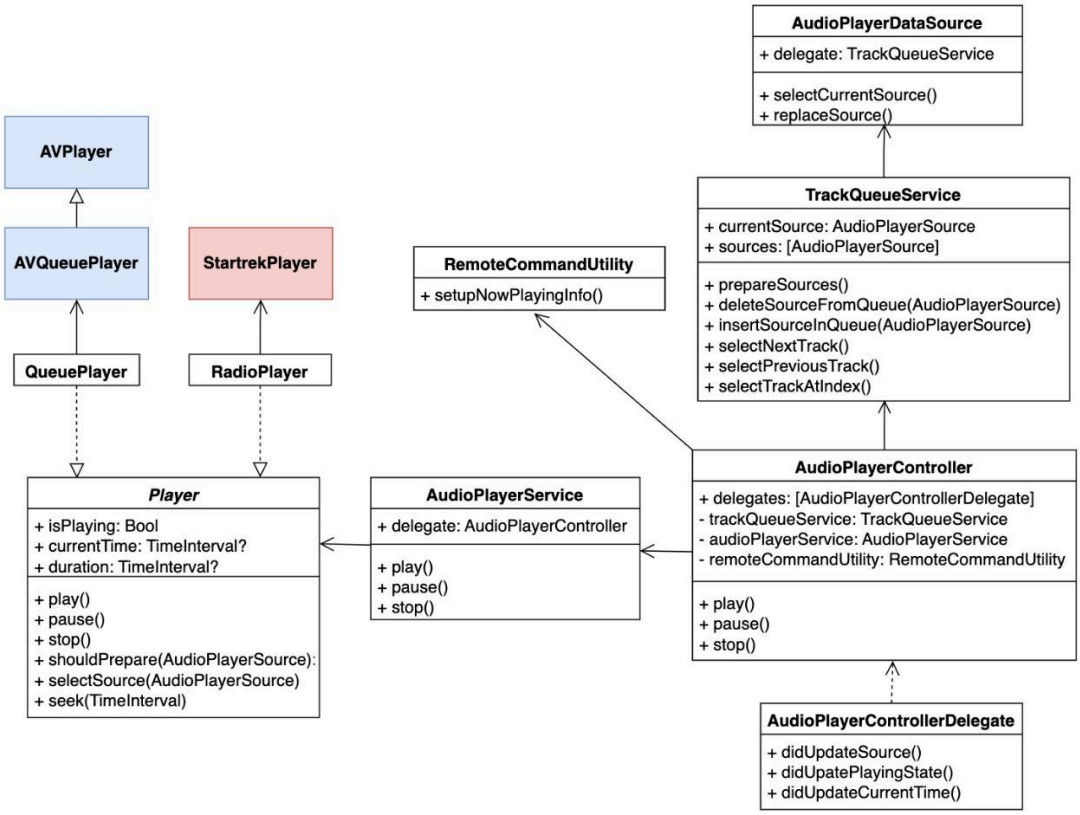


Рисунок 30 - Архитектура плеера

Классы AudioPlayerDataSource и TrackQueueService нужны для работы с очередью и файлами внутри неё. В AudioPlayerDataSource обрабатываются все треки, которые приходят с сервера. Существует три очереди треков - полная, очередь воспроизведения и очередь плеера. Скрытые треки должны пропускаться в плеере, но отображаться особым образом в очереди воспроизведения (*рисунок 31*).

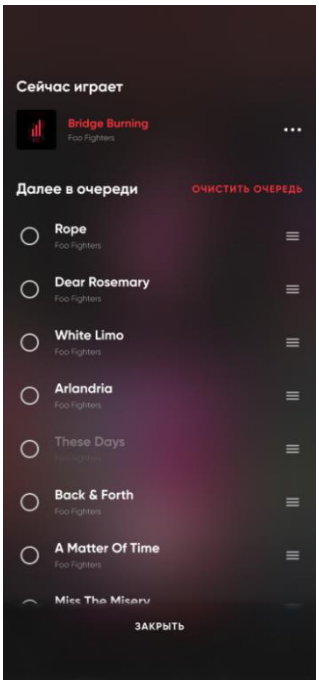


Рисунок 31 - Очередь воспроизведения со скрытыми треками

Класс RemoteCommandUtility служит для установки и обновления информации на экране блокировки устройства и в элементах управления мультимедиа в многозадачном пользовательском интерфейсе (*Рисунок 32*) о мультимедиа, воспроизводимым приложением.

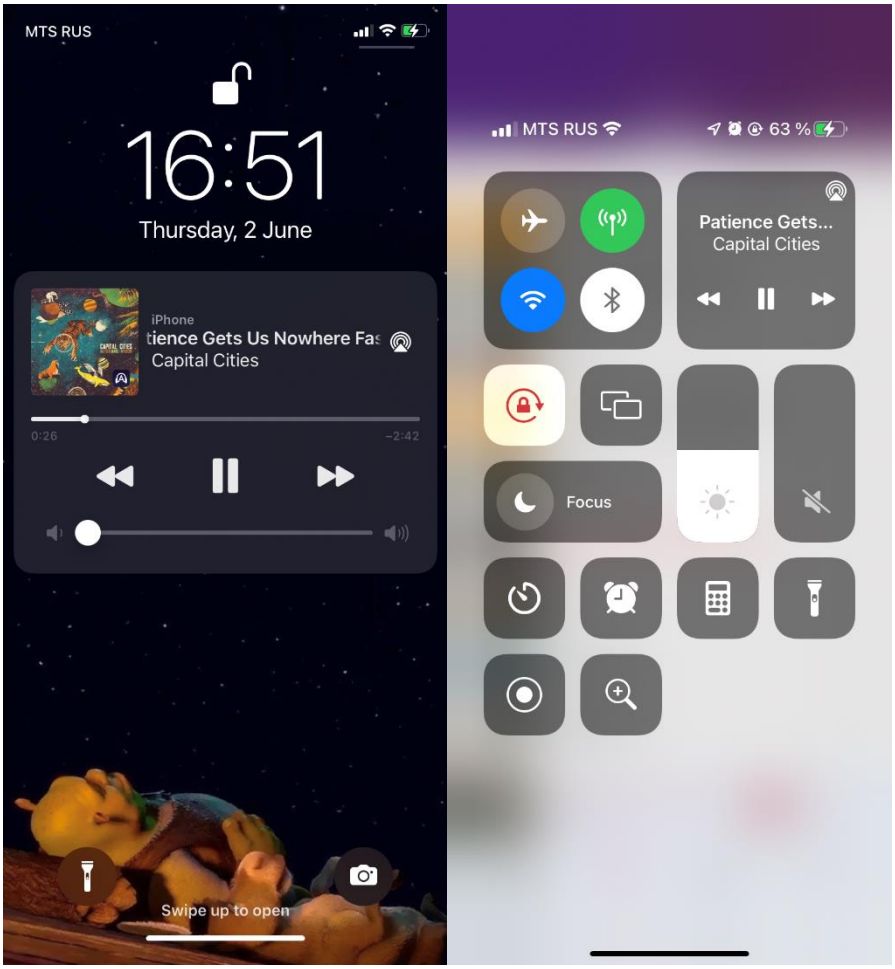


Рисунок 32 - Отображение информации о проигрываемом аудиофайле в системе

Для этого использовался класс MPRemoteCommandCenter [31], а точнее его метод shared() для получения общего объекта командного центра. Это возможно благодаря шаблону проектирования одиночка [34]. Свойства объекта общего командного центра содержат объекты MPRemoteCommand [32], которые реагируют на различные виды событий удаленного управления.

Сервис AudioPlayerService нужен непосредственно для взаимодействия с плеером. Через протокол Player реализованы общие параметры и методы для работы с плеером. Для воспроизведения аудио использовался компонент из iOS SDK – AVQueuePlayer [33], который наследуется от AVPlayer [29] (*Рисунок 33*) и имеет расширенный функционал для работы с очередью. Для работы с радио стримом использовалась сторонняя библиотека Startrack [25]. Сторонние компоненты выделены цветом на диаграмме.

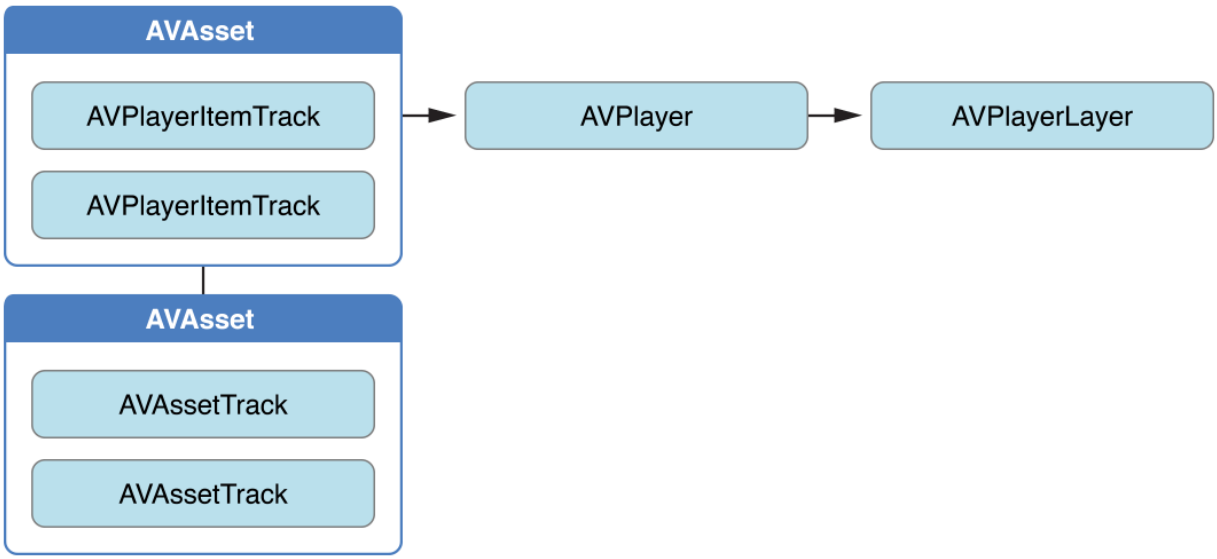


Рисунок 33 - Устройство AVPlayer [36]

Управление проигрыванием, шафлом, репитом и очередью воспроизведения происходит посредством POST запросов на сервер и обновлением состояния после получения успешного ответа (*Рисунок 34*). Это необходимо для сохранения истории прослушивания пользователя. Если действие с плеером требует работу с аудиофайлом, то делается дополнительный запрос на получение ссылки на аудиофайл для проигрывания. Пример взаимодействия с плеером при первом входе в приложение показан на рисунке 35.

Возможные параметры action:

- initialize - инициализация, вызывается на запуске приложения, в ответе приходит состояние плеера, сохраненное в системе ранее;
- play - запрос на воспроизведение, в payload параметре передаются id и опционально context_id, если указан context_id (например, это может быть альбом), то в плеер будут загружены все треки context_id и текущим треком будет установлен трек с id переданным в параметре id, если context_id не указан, то будут загружены все треки объекта id и текущим треком установлен первый трек. В ответе на action play будет сформированная очередь воспроизведения;
- playback_started - событие от плеера приложения о том, что медиа файл текущего трека загружен и воспроизведение стартовало;
- enqueue_next - добавление следующим в очередь, пользователь может добавить в сформированную очередь трек;
- playback_resume - возобновить проигрывание;
- playback_pause - приостановить проигрывание;
- background_pause - поставить на паузу при переходе приложения в фон;

- track_next - переход к следующему треку, событие происходит без участия пользователя;
- track_skip - пропуск текущего трека по команде пользователя;
- track_prev - переход на предыдущий трек по команде пользователя;
- shuffle_enable - включение случайного порядка треков;
- shuffle_disable - отключение случайного порядка треков;
- repeat_one_enable - активация повтора одного трека;
- repeat_all_enable - активация повтора всей очереди воспроизведения;
- repeat_disable - деактивация повтора;
- track_delete - удалить определенные треки из очереди;
- track_reorder - переместить определенный трек внутри очереди;
- track_change - для переключения плеера на конкретный трек в очереди;
- clear_queue - очистить очередь;
- queue_ended - очередь закончилась.

Рисунок 34 - Управление проигрыванием

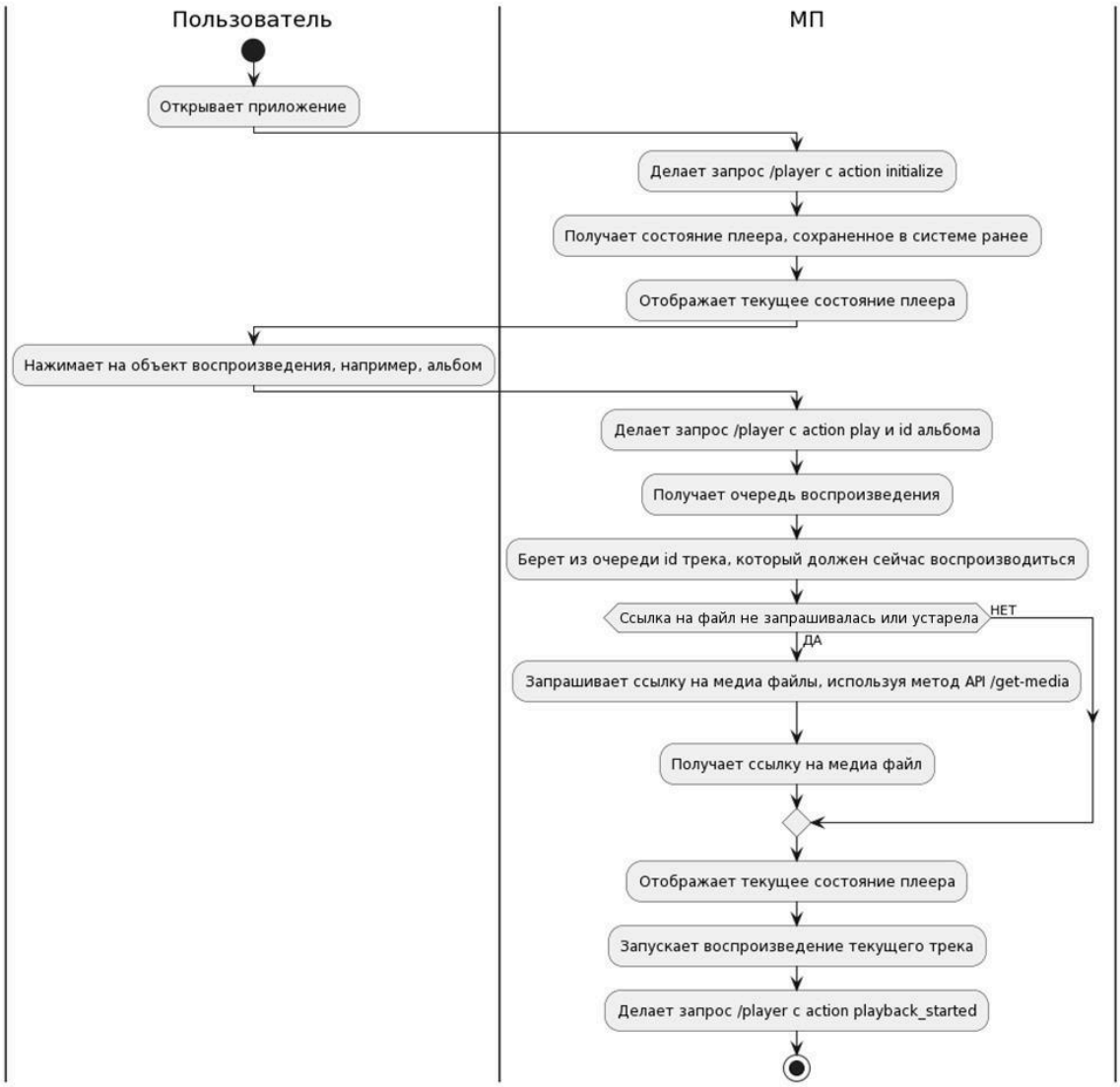


Рисунок 35 - Начало прослушивания

Далее представлен типовой запрос на управление плеером (Листинг *9*).

```
{
  "action": "play",
  "payload": {
    "context_id": "recommended_playlist",
    "ids": [
      "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ],
    "shuffle": "enable",
    "repeat_type": "disabled"
  }
}
```

Листинг 9 - Пример запроса на управление плеером (POST /player)

3.7 Таймер сна

Находясь в большом плеере, можно включить таймер сна (*Рисунок 36*). Пользователь имеет возможность указать время, через которое проигрывание аудиофайла автоматически приостановится. В текущей реализации доступны следующие временные интервалы:

- 5 минут;
- 10 минут;
- 15 минут;
- 30 минут;
- 45 минут;
- 60 минут;
- когда закончится проигрывание текущего аудиофайла:
  - возможность недоступна при проигрывании прямого радиоэфира;
  - если пользователь активировал таймер данного типа и вручную перешёл к следующему/предыдущему треку, то таймер отключается.

Выбор времени реализован через нативный iOS компонент - UIActionSheet [31]. Таймер устанавливается локально и после истечения времени/или после проигрывания аудиофайла система отправляет запрос POST /player с параметром action: playback_pause. Если пользователь закрыл МП, то таймер сна прекращает свою работу.

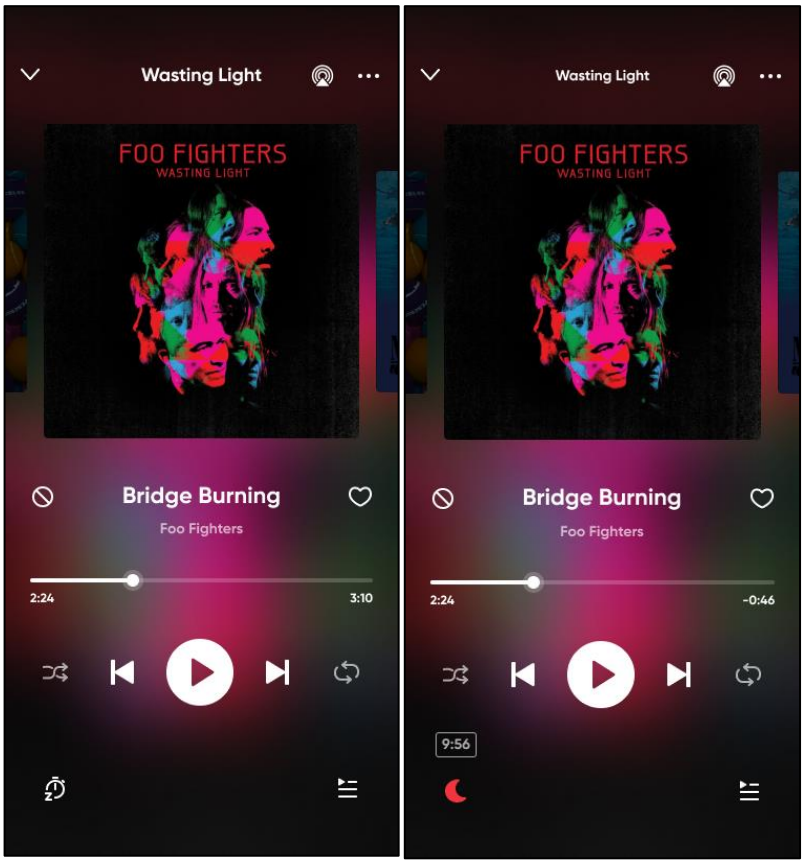


Рисунок 36 - Состояния таймера сна

3.8 Использование Universal Links (универсальных ссылок)

Каждой сущностью в приложении, будь то трек или альбом, можно поделиться (*Рисунок 37*). Соответственно, нужно было предоставить пользователю возможность, перейдя по ссылке с внешнего источника (например, из мессенджера), попасть на соответствующий экран. Также переход в некоторые разделы приложения возможен с веб-версии. Для этого использовались универсальные ссылки (UniversalLinks [35]). Смысл таких ссылок - отправлять пользователя в приложение, если он переходит на страницу сайта, которую приложение поддерживает (*Рисунок 38*).

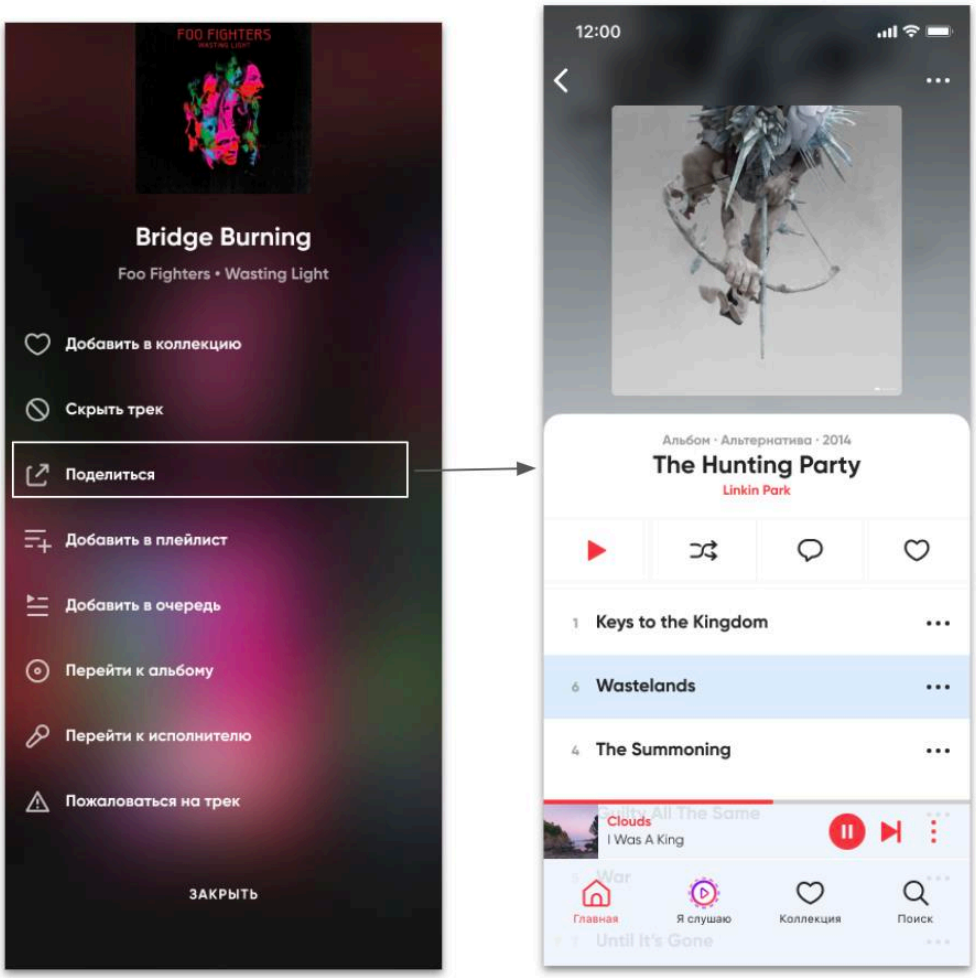


Рисунок 37 - Флоу «Поделиться»

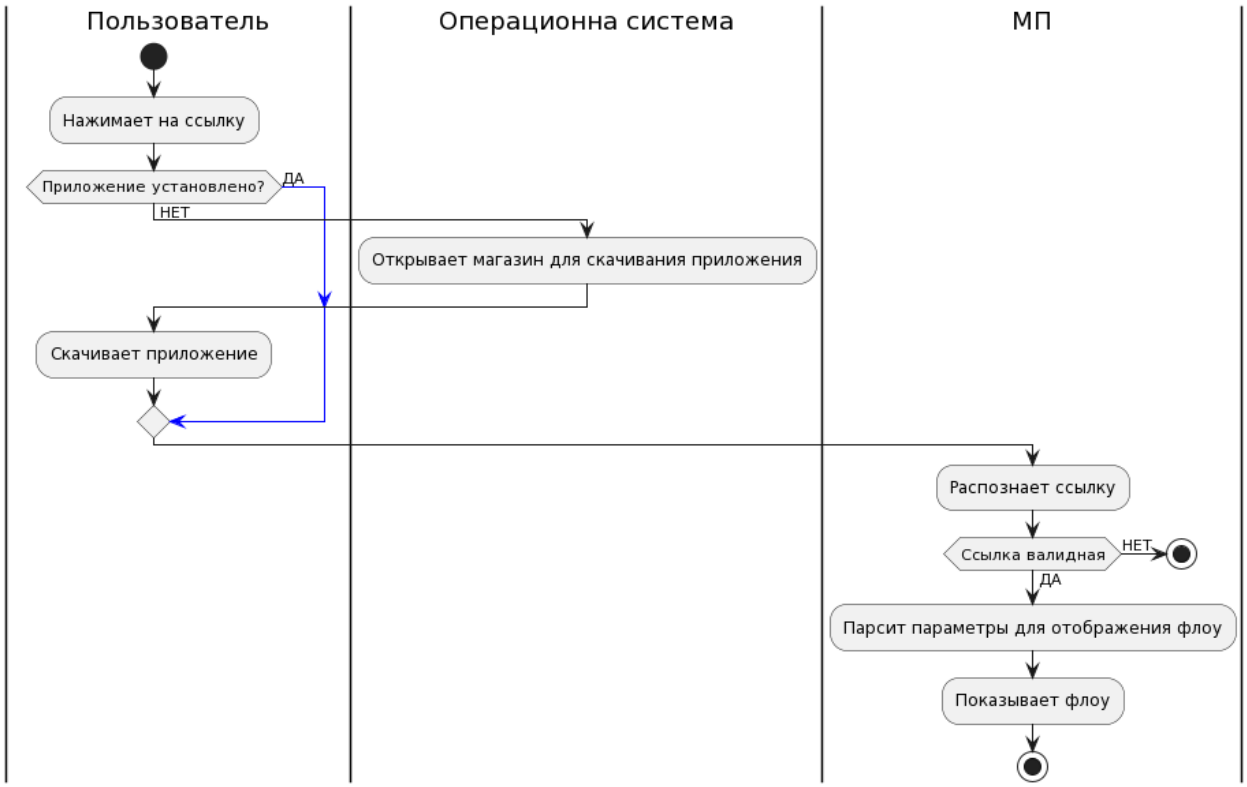


Рисунок 38 - Взаимодействие с универсальными ссылками

Для использования универсальных ссылок необходимо добавить домены, поддерживающиеся приложением в Associated Domains, а также добавить файл apple-app-site-association (Листинг *10*) в корневой каталог каждого домена.

```
{
	"applinks": {
	"apps": [],
	"details": [{
		"appID": "teamId.com.bundleId.of.your.app",
		"paths": ["/albums/*", «/profile»]
	}]
	}
}
```

Листинг 10 - пример файла apple-app-site-association

Для парсинга ссылки и перехода на соответствующий экран, была написана фабрика DeepLinksFactory, которая возвращает определенный тип флоу для перехода в зависимости от параметров (*Рисунок 39*).

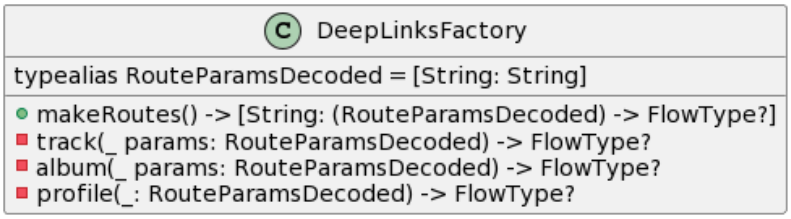


Рисунок 39 - Класс DeepLinksFactory

```
"/albums/:id/:item_param": track
"/albums/:id": album
"/profile": profile
```

Листинг 11 - Пример конечных путей для перехода в приложение

3.9 Анимации

Анимации в мобильном приложении нацелены на выполнение эстетической функции. В приложении используется несколько видов анимации: JSON анимации, реализованные при помощи сторонней библиотеки Lottie [23], и анимация из набора картинок.

В исполняемом коде для Lottie анимации можно задать размер, скорость воспроизведения и число проигрываний — это позволяет точнее настроить анимацию (Листинг *12*).

```
import Lottie // импортирование библиотеки

private let animationView = AnimationView()

// настройка анимации
animationView.animation = Animation.named(viewModel.animationName)
animationView.loopMode = .loop
animationView.backgroundBehavior = .pauseAndRestore
animationView.animationSpeed = 1
animationView.play()  // воспроизведение анимации
```

Листинг 12 - Пример реализации Lottie анимации

Для того, чтобы сделать анимацию из набора картинок, был написан протокол AnimationImagesHaving (Листинг *13*). С помощью встроенного таймера картинки меняются друг за другом.

```
protocol AnimationImagesHaving: AnyObject {
  var format: String { get }
  var imagesPath: String { get }
  var animationStartIndex: Int { get }
  var animationPauseIndex: Int { get }
  var animatedImages: [UIImage] { get set }

  func playAnimation()
  func pauseAnimation()
}
```

Листинг 13 - Интерфейс AnimationImagesHaving

В силу того, что картинки имели черный фон (вместо привычного - прозрачного), нужно было также применять режим наложения - `layer.compositingFilter = "screenBlendMode"`. Таким образом, черный цвет вычитался. Итоговый результат можно увидеть ниже (*Рисунок 40*).

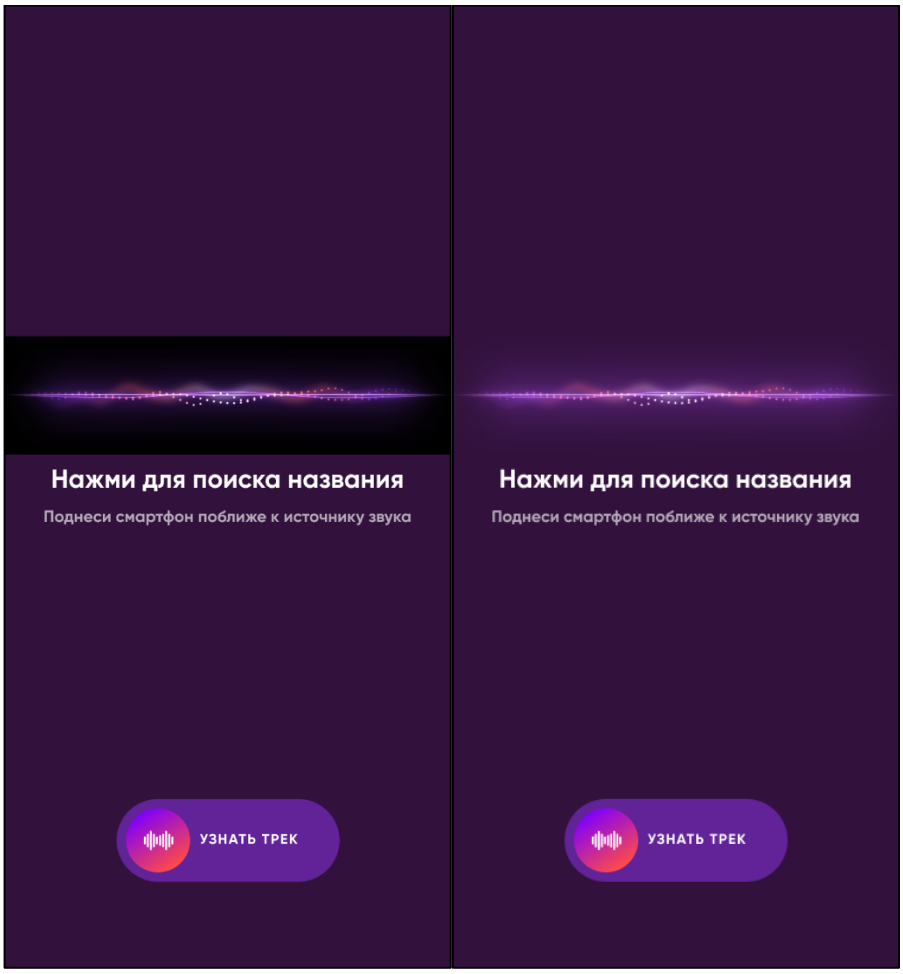


Рисунок 40 - Пример анимации с эффектом наложения

3.10 Локализация строк в приложении

Все строки в приложении выносятся в *.strings файлы и локализуются. Так как проект рассчитан на русскую аудиторию, то и язык разработки по умолчанию - русский.

Для каждого модуля приложения (1-3 экрана, обычно связаны в один координатор) создается отдельный *.strings файл с локализованными строками. Если нужно какие-то строки использовать повсеместно, то они помещаются в Common.strings.

При необходимости склонения слова/фразы, создается словарь (*.stringsdict) для локализации. Если слов немного - можно создать один словарь для всех, иначе лучше также разделить по модулям.

Для удобства чтения сгенерированной переменной разработчиками, а также для того, чтобы переводчики могли понять контекст строки, которую они переводят, нужно придерживаться определенных правил именования идентификаторов локализованных строк:

1. Идентификаторы именуются в нижнем регистре, все слова разделяются точками.
2. Если файл *.strings назван не так, как экран, на котором будет отображена строка - то первое слово в идентификаторе будет именем экрана.
3. Следующее слово - краткое описание и название элемента UI, в котором будет располагаться локализованный текст. Если элементов несколько, и они вложенные - лучше перечислить хотя бы пару уровней вложенности по названиям (например, если текст нужен для ячейки в таблице).
4. Завершить можно словом title / text / option или подобными.

Пример локализации строк представлен на листинге 14.

```
"screen.title" = "Настройки";
"sound.quality.link.title" = "Качество звука";
"infinity.queue.link.title" = "Не заканчивать прослушивание";
"infinity.queue.link.description" = "Если в твоей очереди больше нечего слушать
– мы подберём что-то похожее";
"explicit.content.link.title" = "Пропускать контент для взрослых";
"complain.link.title" = "Пожаловаться на контент";
"feedback.link.title" = "Обратная связь";
"information.link.title" = "Сведения";
"radio.region.title" = "Регион радио";
```

Листинг 14 - Пример локализации строк на экране настроек

ЗАКЛЮЧЕНИЕ

В процессе работы над проектом было спроектировано и реализовано мобильное приложение для прослушивания аудиофайлов и радио, которое позволяет пользователям не только совершать различные действия для комфортного прослушивания (формирование очереди, автоматическая остановка воспроизведения и т. п.), но и искать контент благодаря поиску и системе рекомендаций.

В данной выпускной квалификационной работе были рассмотрены различные типы мобильных приложений и их операционные системы. Был изучен процесс разработки мобильного приложений, принятый в компании, а также написан исходный код мобильного приложения для платформы iOS, позволяющий прослушивать аудиофайлы, в том числе музыку, подкасты, аудиокниги, и радиоэфиры, что является отличительной особенностью данной разработки.

При разработке использовались современные технологии и подходы к проектированию приложений для мобильных операционных систем.

Создание приложения завершено, исходный код передан Заказчику для дальнейшей доработки и публикации. Он полностью соответствует требованиям, которые были поставлены со стороны заказчика. Все задачи выполнены, цель работы достигнута.

Стоит отметить, что текущая версия приложения является только первым этапом. Для обеспечения развития и поддержки приложения в процессе работы использовались архитектурные подходы и решения, которые позволяют быстро и просто добавлять новый функционал. В будущем планируется расширить функционал текущего приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

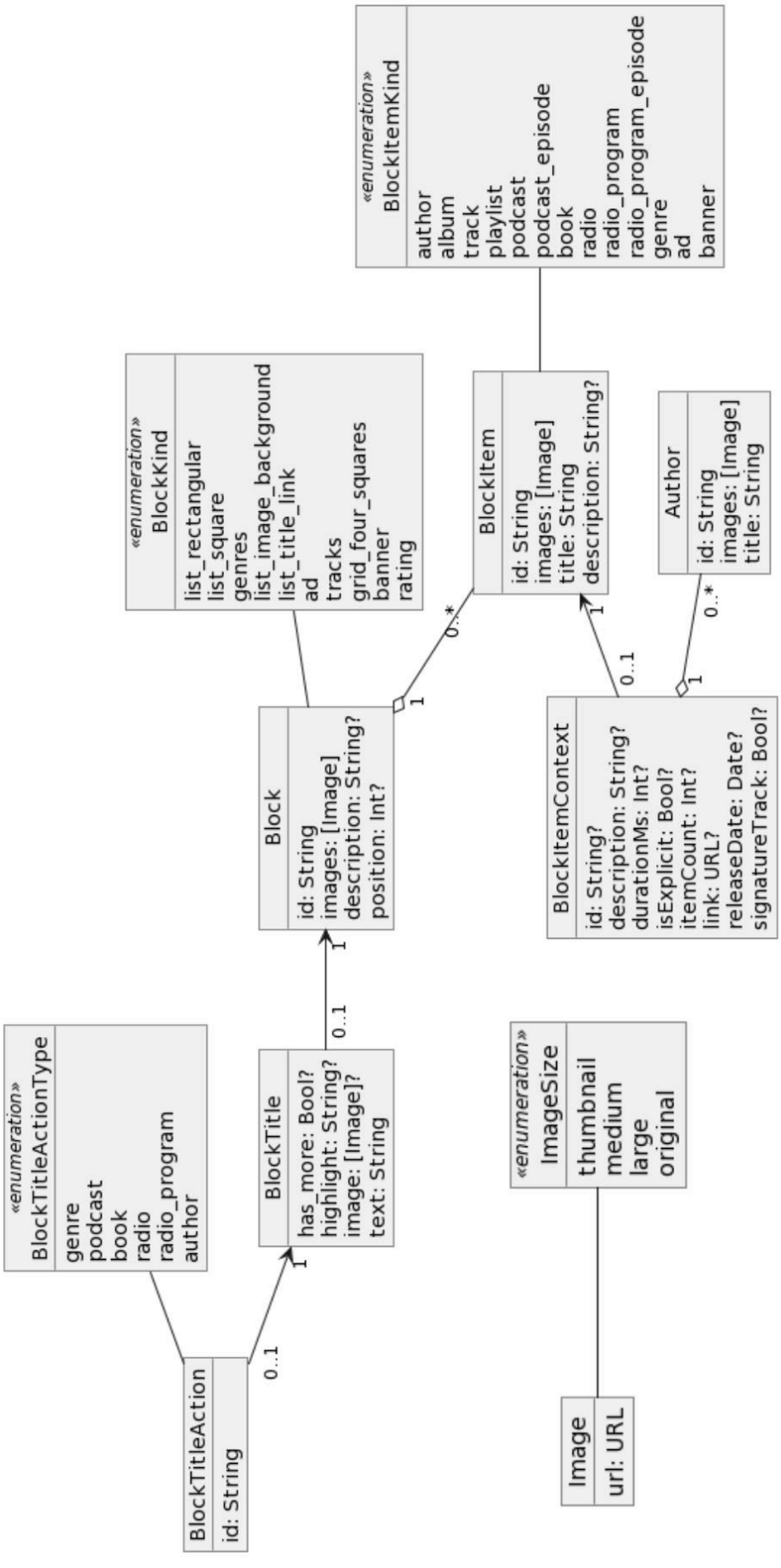
1. Согласно статистике, к концу 2020 года доля пользователей этой платформы в мире составляет 24,99% // Hi-tech – [Б. м.], 2022. – URL: https://hi-tech.ua/dolya-ios-vtroe-nizhe-chem-android-no-deneg-na-prilozheniya-v-nej-tratyat-v-2-raza-bolshe/ (дата обращения: 15.02.2022).
2. API // Википедия. Свободная энциклопедия. – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/API (дата обращения: 08.03.2022).
3. Model-View-ViewModel // Википедия. Свободная энциклопедия. – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/Model-View-ViewModel (дата обращения: 16.02.2022).
4. Замыкание (программирование) // Википедия. Свободная энциклопедия. – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/Замыкание_(программирование) (дата обращения: 16.01.2022).
5. Связывание данных // Википедия. Свободная энциклопедия. – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/Связывание_данных (дата обращения: 16.03.2022).
6. Симан М. Внедрение зависимостей в .NET / М. Симан. – СПб. : Питер, 2014.
7. Draw.io // Draw.io – [Б. м.], 2022. – URL: http://draw.io/ (дата обращения: 18.01.2022).
8. PlantUML // PlantUML – [Б. м.], 2022. – URL: https://plantuml.com/ (дата обращения: 18.01.2022).
9. Swift // Википедия. Свободная энциклопедия. – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/Swift_(язык_программирования) (дата обращения: 15.01.2022).
10. CocoaPods // CocoaPods – [Б. м.], 2022. – URL: https://cocoapods.org (дата обращения: 12.01.2022).
11. Swift Package Manager // Swift – [Б. м.], 2022. – URL: https://swift.org/package-manager/ (дата обращения: 12.02.2022).
12. Alamofire // GitHub – [Б. м.], 2022. – URL: https://github.com/Alamofire/Alamofire (дата обращения: 12.02.2022).
13. SnapKit // GitHub – [Б. м.], 2022. – URL: https://github.com/SnapKit/SnapKit (дата обращения: 12.02.2022).
14. PromiseKit // GitHub – [Б. м.], 2022. – URL: https://github.com/mxcl/PromiseKit (дата обращения: 12.02.2022).
15. SwiftLint // GitHub – [Б. м.], 2022. – URL: https://github.com/realm/SwiftLint (дата обращения: 12.02.2022).
16. R.swift // GitHub – [Б. м.], 2022. – URL: https://github.com/mac-cain13/R.swift (дата обращения: 12.02.2022).

17. SwiftyBeaver // GitHub – [Б. м.], 2022. – URL: https://github.com/SwiftyBeaver/SwiftyBeaver (дата обращения: 12.02.2022).
18. Kingfisher // GitHub – [Б. м.], 2022. – URL: https://github.com/onevcat/Kingfisher (дата обращения: 12.04.2022).
19. NotificationBanner // GitHub – [Б. м.], 2022. – URL: https://github.com/Daltron/NotificationBanner (дата обращения: 12.02.2022).
20. DominantColor // GitHub – [Б. м.], 2022. – URL: https://github.com/indragiek/DominantColor (дата обращения: 12.02.2022).
21. KeychainAccess // GitHub – [Б. м.], 2022. – URL: https://github.com/kishikawakatsumi/KeychainAccess (дата обращения: 12.02.2022).
22. Firebase // GitHub – [Б. м.], 2022. – URL: https://github.com/firebase/firebase-ios-sdk (дата обращения: 12.02.2022).
23. Lottie // GitHub – [Б. м.], 2022. – URL: https://github.com/airbnb/lottie-ios (дата обращения: 12.02.2022).
24. InfiniteScroll // GitHub – [Б. м.], 2022. – URL: https://github.com/pronebird/UIScrollView-InfiniteScroll (дата обращения: 12.02.2022).
25. Startrek
26. GitHub – [Б. м.], 2022. – URL: https://github.com (дата обращения: 12.02.2022).
27. Futures and promises // Khanlou – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/Futures_and_promises/ (дата обращения: 16.02.2022).
28. Codable // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/documentation/swift/codable (дата обращения: 12.02.2022).
29. Mobile Operating System Market Share Worldwide // StatCounter Global Stats – [Б. м.], 2022. – URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата обращения: 15.01.2022).
30. AVPlayer // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/documentation/avfoundation/avplayer (дата обращения: 12.02.2022).
31. MPRemoteCommandCenter // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/documentation/mediaplayer/mpremotecommandcenter (дата обращения: 12.02.2022).
32. MPRemoteCommand // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/documentation/mediaplayer/mpremotecommand (дата обращения: 12.02.2022).

33. AVQueuePlayer // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/documentation/avfoundation/avqueueplayer (дата обращения: 12.02.2022).
34. Одиночка (шаблон проектирования) // Википедия. Свободная энциклопедия. – [Б. м.], 2022. – URL: https://ru.wikipedia.org/wiki/Одиночка_(шаблон_проектирования) (дата обращения: 08.03.2022).
35. UniversalLinks // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/ios/universal-links/ (дата обращения: 12.05.2022).
36. Источник: // Apple Developer Documentation – [Б. м.], 2021. – URL: https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/02_Playback.html
37. URLSession // Apple Developer Documentation – [Б. м.], 2022. – URL: https://developer.apple.com/documentation/foundation/urlsession (дата обращения: 12.02.2022).

ПРИЛОЖЕНИЕ А

Устройство блока



ПРИЛОЖЕНИЕ Б

Дополнительные скриншоты мобильного приложения

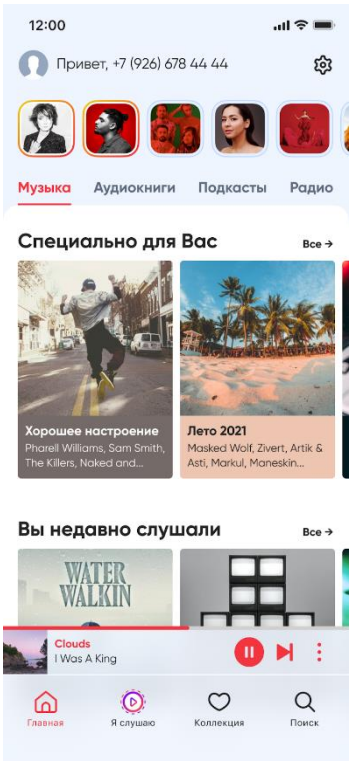


Рисунок Б.1 - Главный экран

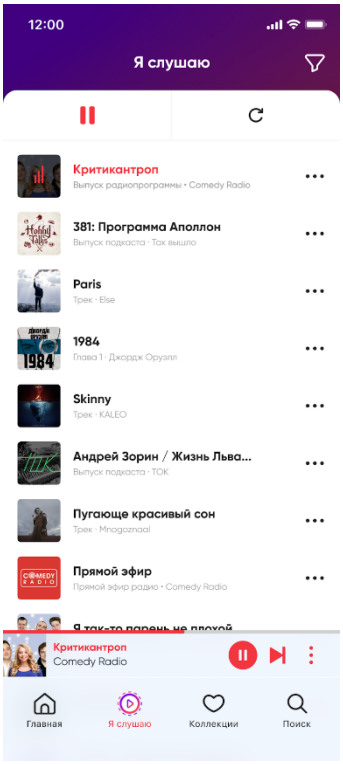


Рисунок Б.2 - Раздел «Я слушаю»

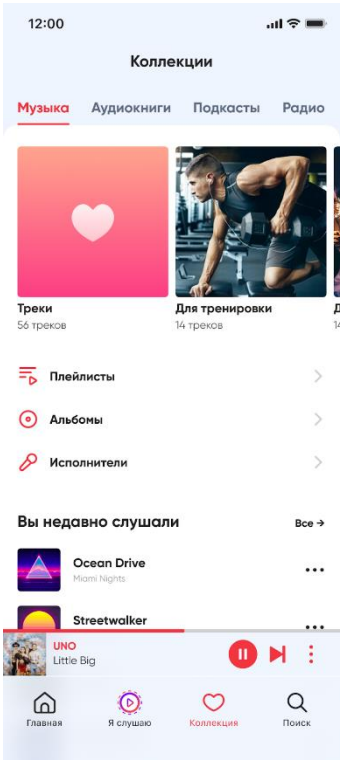


Рисунок Б.3 - Раздел «Коллекции»

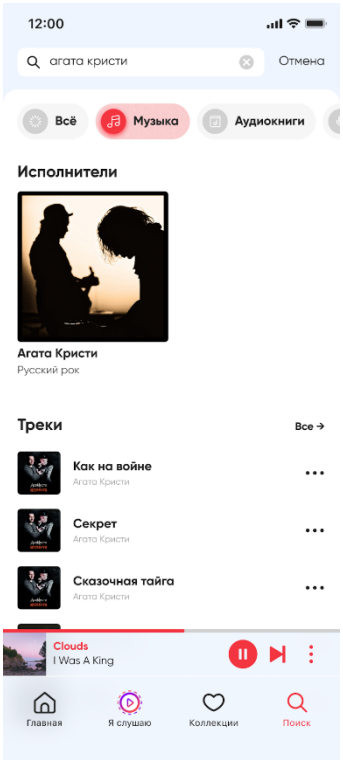


Рисунок Б.4 - Раздел «Поиск»

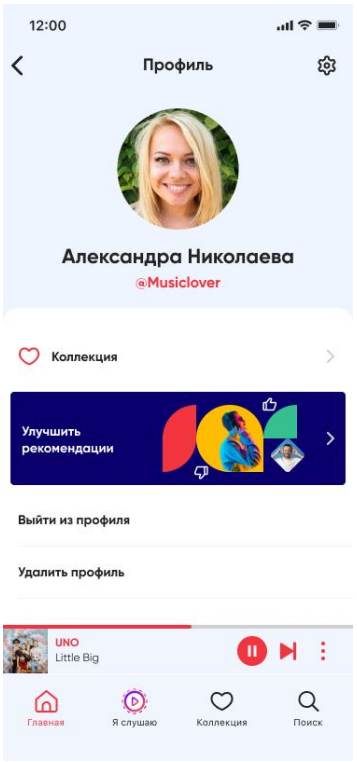


Рисунок Б.5 - Профиль

Рисунок Б.6 - Система рекомендаций

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Кисаева А.В
10.06.2022

Отчет о проверке на заимствования №1

Автор: Перегудова Кристина Сергеевна
Проверяющий: Перегудова Кристина Сергеевна

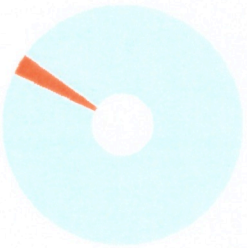
Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 36
Начало загрузки: 08.06.2022 06:08:50
Длительность загрузки: 00:00:07
Имя исходного файла: ВКР_Перегудова К. С.pdf
Название документа: ВКР_Перегудова К. С
Размер текста: 54 кБ
Символов в тексте: 55555
Слов в тексте: 6196
Число предложений: 476

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 08.06.2022 06:08:58
Длительность проверки: 00:00:16
Комментарии: не указано
Модули поиска: Интернет Плюс



ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
2,76%	0%	0%	97,24%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте	Комментарии
[01]	0,98%	1,08%	https://politehvpo.ru/files/docs/nauch_rab/konf21.pdf https://politehvpo.ru	04 Мар 2022	Интернет Плюс	6	7	
[02]	0,12%	0,63%	https://research-journal.org/wp-content/uploads/2020/03/3-1-93.pdf https://research-journal.org	01 Апр 2020	Интернет Плюс	2	9	
[03]	0,07%	0,49%	АНАЛИЗ ОСОБЕННОСТЕЙ ПРИМЕНЕНИЯ ФРЕЙМВОРКА VUE.JS В ВЕБ-ПРИЛОЖЕНИЯХ | sibac.info https://sibac.info	20 Мар 2020	Интернет Плюс	1	7	
[04]	0,43%	0,43%	https://kpfu.ru/student_diplom/10.160.178.20_2627777_Kashapov_M.N._razrabotka_dizajna_prilozheniya.pdf https://kpfu.ru	21 Янв 2022	Интернет Плюс	2	2	
[05]	0%	0,43%	https://kpfu.ru/student_diplom/10.160.178.20_2627777_Kashapov_M.N._razrabotka_dizajna_prilozheniya.pdf https://kpfu.ru	26 Апр 2022	Интернет Плюс	0	2	
[06]	0,4%	0,4%	Операционные системы в смартфонах: виды, плюсы и минусы каждой - Мобайл гуру https://r3d.su	30 Апр 2020	Интернет Плюс	2	2	
[07]	0%	0,4%	Что такое: операционная система телефона — Статьи https://35zip.ru	08 Июн 2022	Интернет Плюс	0	2	

[08]	0%	0,4%	Что такое: операционная система телефона — Статьи https://35zip.ru	08 Июн 2022	Интернет Плюс	0	2
[09]	0,37%	0,37%	https://www.tsu.ru/upload/medialibrary/759/rekomenduemyy_shablon_programmy_gia-26.01.2021.docx https://tsu.ru	24 Янв 2022	Интернет Плюс	2	2
[10]	0,14%	0,28%	https://etu.ru/assets/files/sbornik_sto-2021.pdf https://etu.ru	29 Апр 2022	Интернет Плюс	1	2
[11]	0,25%	0,25%	Работа с сервером с помощью Alamofire на Swift / Хабр https://habr.com	08 Июн 2022	Интернет Плюс	2	2
[12]	0%	0,24%	https://lib.tsu.ru/win/produkzija/metodichka/pril1.pdf https://lib.tsu.ru	25 Мая 2022	Интернет Плюс	0	1
[13]	0%	0,22%	Анализ рынка цифровой дистрибуции мобильных приложений https://yandex.ru	28 Дек 2018	Интернет Плюс	0	2
[14]	0%	0,22%	Анализ рынка цифровой дистрибуции мобильных приложений https://cyberleninka.ru	04 Мая 2020	Интернет Плюс	0	2
[15]	0%	0,22%	Dynamic Influence on Replicator Evolution for the Propagation of Competing Technologies http://arxiv.org	19 Мар 2020	Интернет Плюс	0	2
[16]	0%	0,22%	Анализ рынка цифровой дистрибуции мобильных приложений https://yandex.ru	29 Окт 2019	Интернет Плюс	0	2
[17]	0%	0,22%	http://sciff.ru/wp-content/uploads/2018/07/Sciff_6_22.pdf http://sciff.ru	15 Ноя 2019	Интернет Плюс	0	2