Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ
Зав. каф. ЭВМ
_____ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему
ПРИЛОЖЕНИЕ ДЛЯ ВОСПРОИЗВЕДЕНИЯ МУЗЫКИ СО СТРИМИНГОВЫМ СЕРВИСОМ

БГУИР ДП 1–40 02 01 01 036 ПЗ

Студент	Д.М. Карнаух
Руководитель	М.М. Татур
Консультанты:	
от кафедры ЭВМ	М.М. Татур
по экономической части	Л.М. Михинова
Нормоконтролер	А.С. Сидорович
Рецензент	

МИНСК 2021

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС. Кафедра: ЭВМ.
Специальность: 40 02 01 «Вычислительные машины, системы и сети».
Специализация: 40 02 01-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Б.В.Никульшин
«____» _____ 2021 г.

ЗАДАНИЕ
по дипломному проекту студента
Карнаух Дарьи Михайловны

1 Тема проекта: «Приложение для воспроизведения музыки со стриминговым сервисом» – утверждена приказом по университету от 24 марта 2021 г. № 654-с.

2 Срок сдачи студентом законченного проекта: 1 июня 2021 г.

3 Исходные данные к проекту:
3.1 Операционная система: Windows 10.
3.2 Среда разработки: Visual Studio 2019.
3.3 Языки программирования и фреймворки: C#, .NET.
3.4 База данных: Microsoft SQL Server.

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):
Введение 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. 7. Технико-экономическое обоснование разработки и реализации на рынке приложения для воспроизведения музыки со стриминговым сервисом. Заключение. Список использованных источников. Приложения.

5 Перечень графического материала (с точным указанием обязательных чертежей):

5.1 Вводный плакат. Плакат.
5.2 Приложение для воспроизведения музыки со стриминговым сервисом. Схема структурная.
5.3 Приложение для воспроизведения музыки со стриминговым сервисом. Схема функциональная.
5.4 Приложение для воспроизведения музыки со стриминговым сервисом. Диаграмма классов.
5.5 Приложение для воспроизведения музыки со стриминговым сервисом. Диаграмма последовательности.
5.6 Приложение для воспроизведения музыки со стриминговым сервисом. Клиентская часть. Модель данных.
5.7 Приложение для воспроизведения музыки со стриминговым сервисом. Серверная часть. Модель данных.
5.8 Заключительный плакат. Плакат.

6 Содержание задания по экономической части: «Технико-экономическое обоснование разработки и реализации на рынке приложения для воспроизведения музыки со стриминговым сервисом».

ЗАДАНИЕ ВЫДАЛ Л. М. Михинова

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы. Сравнение аналогов. Уточнение задания на ДП	10	23.03 – 30.03	
Структурное проектирование	15	30.03 – 08.04	
Функциональное проектирование	25	08.04 – 24.04	
Разработка программных модулей	20	24.04 – 08.05	
Программа и методика испытаний	10	08.05 – 15.05	
Расчет экономической эффективности	5	15.04 – 20.05	
Оформление пояснительной записки	15	20.05 – 30.05	

Дата выдачи задания: 24.03.21

Руководитель М. М. Татур

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ _____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Музыка окружает меломанов везде, где бы они ни были и что бы ни делали. Теперь она составляет важную часть жизни человека. Помимо прогулок, занятий спортом, где требуется использование мобильного телефона, актуальны приложения на ПК. В современном мире как никогда быстро растет количество музыкального контента. По статистике на популярный мировой стриминговый сервис Spotify ежедневно загружается более 40 000 новых композиций, а их общее количество давно превысило отметку в 50 миллионов. Однако у таких стриминговых сервисов есть один большой недостаток: в основном туда загружают свежие композиции, а старые и малоизвестные остаются вне общего доступа из-за проблем с нарушением авторских прав.

Уже существует огромное количество плееров на разный вкус и цвет, но нет расширенных решений, которые воспроизводят радиостанции и работают со стриминговым сервисом.

Целью данного дипломного проекта является разработка аудиоплеера для ПК с элементами стримингового сервиса. В первую очередь приложение разрабатывается как локальный плеер для воспроизведения композиций на устройстве. Второй частью является элемент стримингового сервиса, позволяющий слушать композиции онлайн.

В соответствии с поставленной целью были определены следующие задачи:

- выбор платформы для создания системы;
- разработка пользовательской и серверной частей системы;
- реализация функциональности плеера;
- разработка взаимодействия со стриминговым сервисом.

Система будет представлять собой десктопное приложение и сервер, которые будут предоставлять следующие функции:

- создание пользователей в системе;
- прослушивание радио;
- прослушивание стриминговых и локальных композиций;
- работа с плейлистами;
- работа с выделенной памятью на сервере для пользователя.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Постановка задачи

В рамках дипломного проекта поставлена задача разработать программный продукт, ориентированный на операционную систему Windows, предоставляющий пользователям возможность прослушивать радио, слушать локальные и онлайн стриминговые композиции, а также хранить свои композиции в облаке. Для достижения поставленной цели необходимо решить следующие задачи:

– изучить существующие аналоги;

– на основании произведенного анализа выбрать и изучить необходимые технологии для реализации дипломного проекта;

– спроектировать и разработать приложение.

1.2 Обзор существующих аналогов

Как бы не развивался мир в сторону онлайн решений, остается большое количество людей, которые не приемлют постоянную зависимость от необходимости подключения к интернету. Такие люди уже составили комфортный плейлист для себя, предпочитая проверенные временем песни. Поэтому единственное решение в таком случае – скачивание и хранение на локальном носителе, а воспроизведение через любой любимый плеер, видов которых множество на любой вкус и потребности.

Стоит учитывать, что необходима возможность синхронизировать различные композиции на различных устройствах одного пользователя. Это решается довольно легко путем выделения места на сервере.

Так же не стоит забывать о тех, кто предпочитает или любит иногда слушать радио онлайн. Обычно человек выбирает несколько радиостанций, которые ему по душе, и делает закладки в браузере на страницах этих радиостанций. Возникает закономерный вопрос: почему бы не соединить любимые радиостанции в одном приложении? Тогда не придется отвлекаться на несколько страниц в браузере, чтобы узнать, какая композиция сейчас играет и переключиться на нее.

Чтобы оценить актуальность разработки нового продукта, необходимо рассмотреть уже существующие решения на рынке. Для этого необходимо рассмотреть как стриминговые сервисы и плееры, так и социальные сети.

1.2.1 ВКонтакте

Ни для кого не секрет, что на территории СНГ [1] самой распространенной социальной сетью является «ВКонтакте». Эта сеть позволяет не только обмениваться новостями и сообщениями, но и предлагает очень удобный сервис для работы с музыкой (рисунок 1.1).

Возможности этой платформы:

– наличие непопулярных композиций;
– обмен композициями без соблюдения авторского права;
– возможность закачивать свои композиции;
– автоматическая рекомендация музыки;
– работа с плейлистами.

Недостатки:

– нет радио;
– нельзя прослушивать локальные аудио;
– несоблюдение авторских прав.

В целом, это очень удобный сервис с широким набором различных композиций, которые популярны на просторах Украины, Беларуси, России и других русскоязычных стран. Но это только онлайн-сервис, необходимо полностью закачать свои композиции, чтобы составить удобные плейлисты.

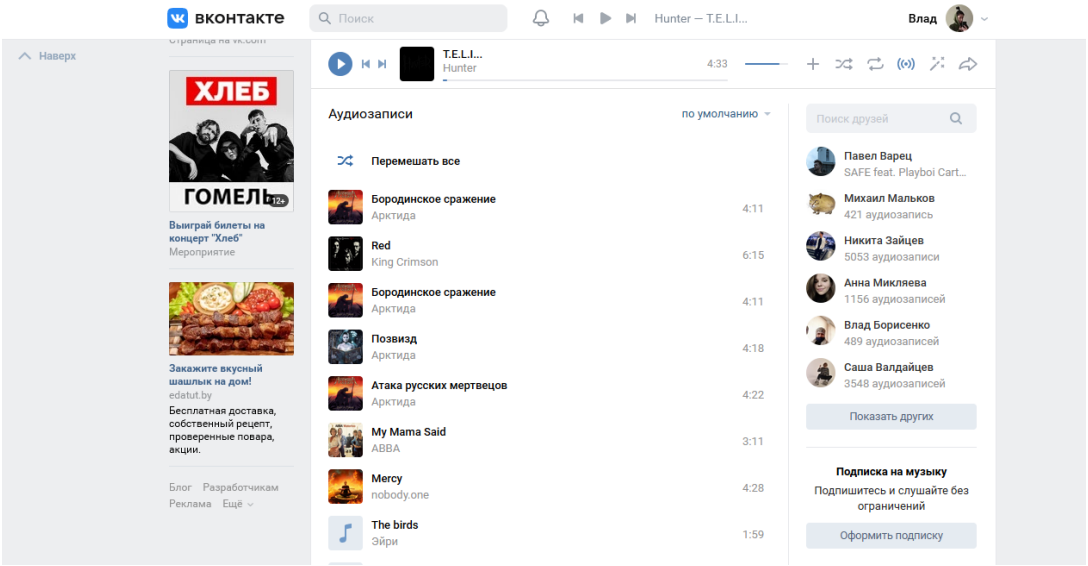

Рисунок 1.1 - Аудиозаписи «ВКонтакте»

1.2.2 Стриминговые сервисы

Рассмотрим возможности стриминговых сервисов на примере двух популярных стриминговых сервисов – Яндекс.Музыка (рисунок 1.2) и Spotify (рисунок 1.3).

Безусловным преимуществом является удобство использования. Помимо большого выбора композиций, отлично работает система рекомендаций, которая предлагает наилучшие похожие композиции в этом жанре, ориентируясь на аудио пользователей с похожим вкусом. Однако, как описывалось выше, строгое соблюдение авторских прав [2] делает невозможным загрузку любимой, но непопулярной или старой музыки, которую правообладатель уже не выложит. И, к сожалению, возможности

подключить радио тоже нет. Хотя радио в этих платформах есть, оно представляет собой автоматический подбор композиций с платформы.

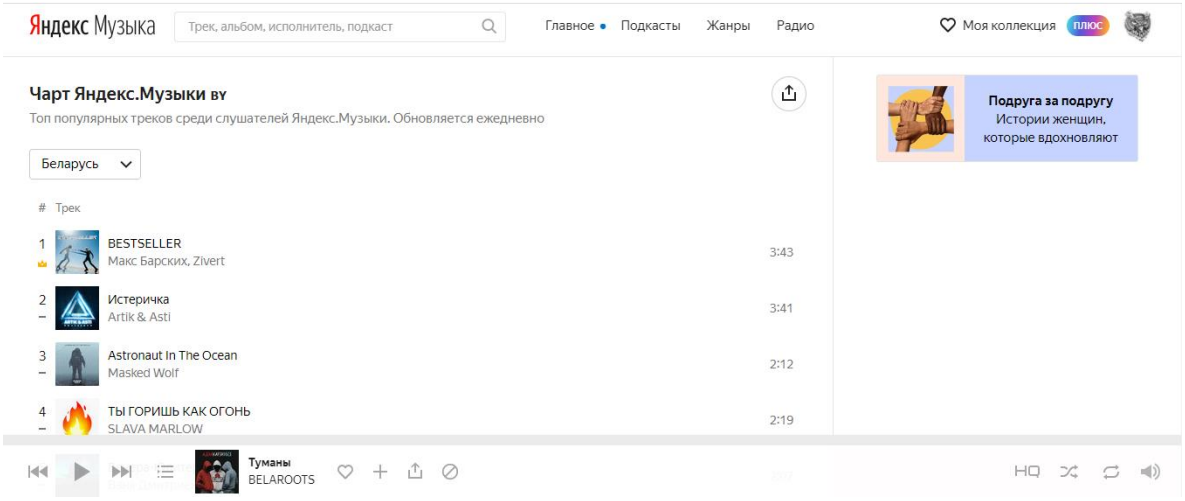

Рисунок 1.2 – Стриминговый сервис «Яндекс.Музыка»

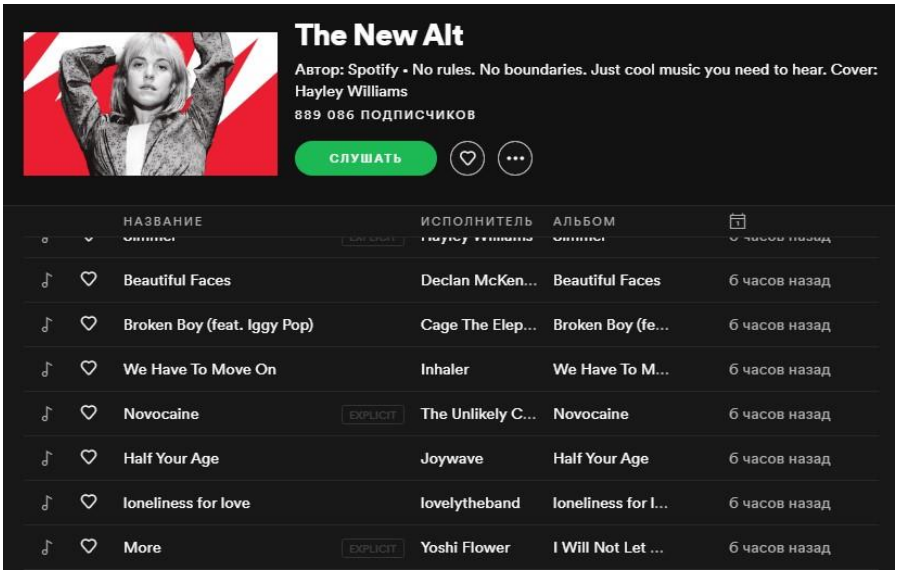

Рисунок 1.3 – Стриминговый сервис «Spotify»

1.2.3 Плееры

Так как наше приложение в первую очередь плеер, необходимо провести краткий обзор аналогов для ОС Windows:

– KMPlayer – является одним из самых мощных мультимедийных плееров для Windows на сегодняшний день (рисунок 1.4). Отличается поддержкой фактически всех существующих форматов. Включает в себя множество визуальных и звуковых эффектов, позволяющих получить удовольствие от красивых картинок. Обладает продвинутым эквалайзером и секвенсором. В добавок есть возможность просмотра видео [3].

Рисунок 1.4 – KMPlayer [3]

– Winamp – проверенная временем утилита для проигрывания разнообразных мультимедийных файлов (рисунок 1.5). Содержит множество обложек для быстрого изменения интерфейса на любой вкус. Доступен вывод подробной информации о композициях, исполнителях и альбомах. Однако, несмотря на богатый функционал, интуитивно понятный продуманный интерфейс, многофункциональный проигрыватель для платформы Windows уже давно не обновляется [4].

Рисунок 1.5 – Winamp [4]

– ComboPlayer – универсальная программа для скачивания torrent-файлов, воспроизведения потокового видео и онлайн-радио, просмотра TV и проигрывания музыкальных композиций (рисунок 1.6). Обеспечивает свободный доступ к огромной базе каналов и станций. Позволяет быстро создавать плейлисты и сортировать контент по категориям. Присутствует поддержка большого количества аудио и видео форматов. Возможна

синхронизация треков с другими компьютерами на базе операционной системы Windows 7, 8 и 10, для этого достаточно войти в учетную запись [5].

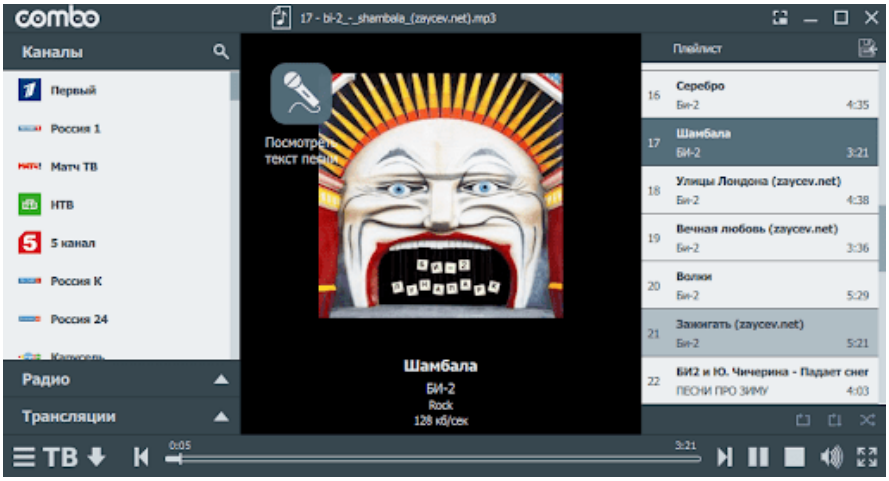

Рисунок 1.6 – ComboPlayer [5]

– VLC Media Player – медиаплеер от разработчика VideoLAN, созданный для воспроизведения музыки и видео файлов (рисунок 1.7). Его особенностью, является возможность открытия «битых» данных. Предоставляет тонкие настройки эквалайзера. Позволяет просматривать потоковые видеотрансляции и прослушивать онлайн-радио [6].

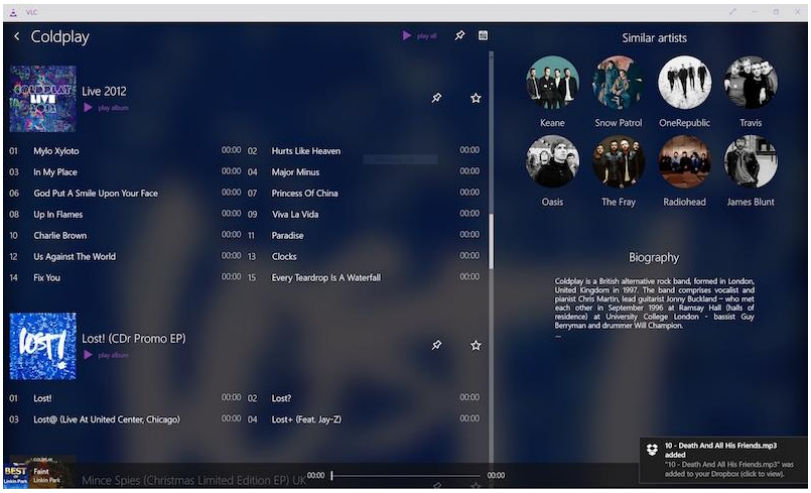

Рисунок 1.7 – VLC Media Player [6]

– AIMP – медиаплеер, созданный под операционную систему Windows, позволяющий комфортно прослушивать музыку практически любого формата (WAV, FLAC, MP3, OGG, CDA и так далее) (рисунок 1.8). Присутствует функция записи звука, редактирования тегов и пакетного переименования треков, чего не найдешь у других популярных проигрывателей. Присутствует возможность создания списков

воспроизведения с любимыми композициями. Доступно расширение дополнительных возможностей при помощи подключения плагинов [7].

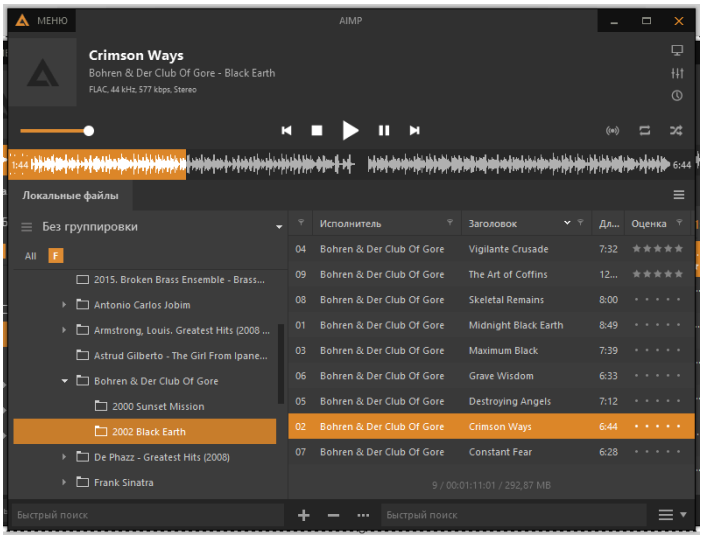

Рисунок 1.8 – AIMP [7]

Для наглядного отображения возможностей каждого плеера составлена таблица сравнения, в которой перечислены основные важные характеристики плееров:

Таблица 1.1 – Возможности плееров

	KMPlayer	Winamp	Combo Player	VLC	AIMP
Онлайн доступ музыки	–	–	–	–	–
Высокое качество воспроизведения	+	+	+	+	+
Воспроизведение различных форматов аудиокодеков	+	+	+	+	+
Аудиореклама	–	–	–	–	–
Синхронизация настроек и треков	–	–	+	–	–
Радио	+	–	+	+	+
Лицензия	free	free	free	free	free

Главным недостатком всех плееров является отсутствие возможности для воспроизведения музыки онлайн и, соответственно, синхронизации между устройствами и со стриминговым сервисом. Напротив, в стриминговых сервисах нельзя найти некоторые композиции. Исходя из разобщенности на

почве авторских прав, нельзя составить комфортный плейлист, включающий композиции из разных источников, а также синхронизировать и проигрывать композиции с разных устройств. Именно эти проблемы необходимо решить в новом плеере.

1.3 Архитектура клиент-сервер

Т.к. тема дипломного проекта – плеер с элементами стримингового сервиса, то будет отсутствовать большинство функций классического сервера, которые направлены на его независимую работу и являются отдельными программами. Поэтому решением выступает использование архитектуры Backend-as-a-Service (BaaS) [8].

Backend-as-a-Service (BaaS) — это модель облачного сервиса, в которой разработчики передают на аутсорсинг все закулисные аспекты веб-или мобильного приложения, так что им остается только написать и поддерживать интерфейс. Поставщики BaaS предоставляют предварительно написанное программное обеспечение для действий, выполняемых на серверах, таких как аутентификация пользователей, управление базами данных, удаленное обновление и push-уведомления (для мобильных приложений), а также облачное хранилище и хостинг.

Провайдеры BaaS предлагают ряд возможностей на стороне сервера:

- управление базами данных;
- облачное хранилище;
- аутентификация пользователя;
- всплывающее уведомление;
- удаленное обновление;
- хостинг.

1.3.1 Примеры BaaS

Практически все Backend-as-a-Service работают приблизительно одинаково хорошо, необходимо выбирать по личным предпочтениям. Далее приведены наиболее популярные сервисы.

Преимущества использования Back4App [9] следующие:

- открытый исходный код;
- реляционные запросы;
- простота использования и гибкость;
- кроссплатформенность;
- хостинг с полностью бессерверной структурой;
- CDN — сеть доставки контента;
- GraphQL - язык запросов для API;.
- круглосуточная поддержка.

Преимущества использования фреймворка Parse [10] следующие:

- управление данными;

– простота и гибкость;
– открытый исходный код;
– пользовательский код, использующий Javascript;
– GraphQL.

Преимущества использования Firebase [11] следующие:

– аутентификация пользователей;
– простота;
– полный набор сервисов аналитики;
– интеграция со службой AdMob;
– A/B-тестирование;
– база данных в реальном времени с синхронизацией;
– автоматическая синхронизация;
– машинное обучение.

1.3.2 Выбор BaaS

Целью данной работы является разработка в первую очередь клиентского приложения. Самыми главными критериями для выбора платформы являются простота, расширенная система хранилища, цена и база данных в реальном времени. Все перечисленные платформы являются хорошими вариантами, но, по моему мнению, Firebase – наиболее подходящий вариант.

В качестве среды для эмуляции стримингового сервиса была выбрана Firebase от Google. Firebase предоставляет бесплатное хранилище на 1 Гб, что поможет в разработке диплома не платить за хранилище. Также простой API ускорит разработку системы. К тому же при дальнейшем расширении базы данных цены и производительность, предоставляемые хостингом, являются вполне хорошими. Но если хостинг нужно будет расширять на многомиллионное количество композиций, то необходимо будет разрабатывать собственные сервера.

1.4 Платформа .NET

Платформа .NET Framework – это программная платформа для построения приложений на базе семейства операционных систем Windows, а также многочисленных операционных систем, таких как Unix и Linux [12]. Ниже приведен краткий перечень некоторых ключевых средств, поддерживаемых платформой .NET:

– возможность взаимодействия с существующим кодом;
– поддержка многочисленных языков программирования;
– общий исполняющий механизм, разделяемый всеми поддерживаемыми платформой языками;
– языковая интеграция;

– обширная библиотека базовых классов;
– упрощенная модель развертывания.

Возможность взаимодействия с существующим кодом является очень полезной, поскольку позволяет комбинировать существующие двоичные компоненты модели компонентных объектов COM (Component Object Model), то есть обеспечивать взаимодействие с ними, с более новыми программными компонентами .NET и наоборот. Приложения на платформе .NET можно создавать с использованием любого числа языков программирования. Одним из аспектов механизма, разделяемого всеми поддерживаемыми .NET языками программирования, является наличие хорошо определенного набора типов, которые способен понимать каждый поддерживающий .NET платформу язык. В .NET поддерживается языковая интеграция, то есть межъязыковое наследование, межъязыковая обработка исключений и межъязыковая отладка кода.

Обширная библиотека базовых классов позволяет избегать сложностей, связанных с выполнением низкоуровневых обращений к интерфейсам прикладного программирования, и предлагает согласованную объектную модель, используемую всеми поддерживающими .NET языками. Согласно упрощенной модели развертывания, библиотеки .NET не регистрируются в системном реестре. Более того, платформа .NET позволяет сосуществовать на одном и том же компьютере нескольким версиям одной и той же сборки.

Основными строительными блоками платформы .NET являются следующие:

– общеязыковая исполняющая среда;
– общая система типов;
– общеязыковая спецификация.

1.5 Интерфейс

Так как приложение будет написано на платформе .NET, для написания интерфейса была выбрана технология WPF (Windows Presentation Foundation). Она является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов [13].

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML:

возможность создавать насыщенный графический интерфейс, используя декларативное объявление интерфейса или код на языке C#.

Преимущества WPF:

– использование традиционных языков .NET-платформы - C# и VB.NET для создания логики приложения;

– возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml;

– независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением;

– новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и другое;

– хорошее взаимодействие с WinForms, благодаря чему в приложениях WPF можно использовать традиционные элементы управления из WinForms;

– богатые возможности по созданию различных приложений: мультимедиа, двухмерная и трехмерная графика, богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и другое;

– аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной;

– создание приложений под множество ОС семейства Windows - от Windows XP до Windows 10.

XAML (eXtensible Application Markup Language) - язык разметки, используемый для инициализации объектов в технологиях на платформе .NET [14]. Применительно к WPF данный язык используется прежде всего для создания пользовательского интерфейса декларативным путем.

XAML не является обязательной частью приложения, можно вообще обходиться без него, создавая все элементы в файле связанного с ним кода на языке C#. Однако использование XAML для данного проекта необходимо и несет некоторые преимущества:

– возможность отделить графический интерфейс от логики приложения, благодаря чему над разными частями приложения могут относительно автономно работать разные специалисты: над интерфейсом - дизайнеры, над кодом логики – программисты;

– компактность;

– понятность;

– код на XAML относительно легко поддерживать.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Для программного средства были определены цели и задачи, которые реализуемое приложение должно решать. В соответствии с целями и задачами были сформулированы требования, владение которыми необходимо для разработки системы. На основе требований систему можно разделить на функциональные модули, каждый из которых является ответственным за свое предназначение.

Данный подход, характеризующийся разбиением системы на модули, дает приложению возможность расширения, добавления новых модулей, а также возможность замены одних функциональных модулей новыми, в которых реализована улучшенная функциональность программного средства. В таком случае система обладает высокой гибкостью.

2.1 Основные блоки разрабатываемого приложения

В разрабатываемом приложении были выделены следующие блоки:

- блок аутентификации;
- блок приложения;
- блок локальной базы данных;
- блок воспроизведения аудио;
- блок работы с плейлистами;
- блок радио;
- блок совершения подписок;
- блок стримингового сервиса;
- блок работы с дополнительной памятью;
- блок взаимодействия с сервером;
- блок сервера.

Каждый блок выполняет характерную для него функциональность, а также взаимодействует с другими блоками системы.

Структурная схема, которая отображает блоки разрабатываемого приложения, перечисленные выше, и связи между ними, приведена на чертеже ГУИР.400201.036 С1.

2.2 Блок аутентификации

Этот блок является вторым исполняемым при открытии приложения. Он является необходимым блоком, поскольку основной его функцией является обеспечение безопасности, устранение несанкционированного доступа к данным.

В данном блоке выполняется вход в учетную запись пользователя. Если таковой нет, она создается. Если выбирается опция создания новой учетной записи, то создается логин, который должен быть почтой, и пароль, у которого

должны быть свои правила составления: длина и наличие специальных символов. После успешной проверки введенных полей выполняется их внесение в базы данных. Сначала идет работа с локальной базой данных: выполняется проверка на наличие уже такого пользователя, если его нет, то заносится почта пользователя, пароль и дата создания пользователя. Затем проверяется подключение к интернету, чтобы внести эти данные в таблицу пользователей в Firebase, используя блок взаимодействия с сервером. Если выполняется вход, то сначала идет проверка пользователя в локальной базе данных, если его там нет, то при подключении к интернету идет обращение к таблице пользователей и проверяется наличие пользователя там. Если его не получается найти, то выводится сообщение об ошибке.

2.3 Блок приложения

Данных блок представляет целостность десктопного приложения. Из данного блока можно обращаться к функциям приложения, таким как радио, работа с плейлистами, совершение подписок. Главной его особенностью является продумывание интерфейса, который должен быть интуитивно понятен и соответствовать последним трендам в оформлении приложений.

Также этот блок собирает информацию о пользователе, чтобы потом предоставить его статистику использования этим приложением: сколько плей листов у него и сколько времени он прослушивал музыку с начала учетной записи.

2.4 Блок локальной базы данных

База данных в системе выполняет функции долговременного хранения, обеспечения доступа к данным, обновления уже существующей информации или добавления новой. До начала работы приложения данные в базе данных отсутствуют.

В разрабатываемом приложении использована система управления базами данных Microsoft SQL Server. База данных является реляционной. Это значит, что данные организованы в виде набора таблиц, состоящих из набора строк и столбцов. Таблицы характеризуются различными видами связей друг с другом.

Для создания, модификации и управления данными был использован декларативный язык программирования SQL, представляющий собой интерфейс работы с реляционной базой данных.

Ключевой особенностью разрабатываемого приложения касательно базы данных является высокая значимость использования транзакций. Транзакции представляют собой некоторое количество операторов SQL, выполненных в виде последовательности действий над данными, представляющих собой единую неделимую логическую задачу, которая может быть либо выполнена целиком, либо не выполнена совсем. Транзакции

необходимы разрабатываемому приложению для безопасной работы с данными, предотвращения возможной потери данных.

Блок базы данных взаимодействует с блоками авторизации и работы с плейлистами с целью передачи и приема данных, а также их обновления в процессе работы приложения. Он хранит пользователей, составленные плейлисты и путь доступа к локальным композициям.

2.5 Блок воспроизведения аудио

Этот блок является единственным, который проигрывает музыку. Он получает данные от блоков радио и работы с плейлистами, те есть стриминговый поток аудио или путь к локальной композиции.

Этот блок имеет эквалайзер, настройка которого напрямую влияет на качество проигрываемого потока. Краткий обзор настроек эквалайзера:

– громкость звука;

– частота дискретизации, при изменении частоты дискретизации выше 44100 происходит эффект быстрой перемотки, за счет увеличения частоты воспроизведения в секунду;

– более тонкая настройка звука, которую обеспечивают десять полос, от 60 Гц до 16 КГц;

– эффект множественного отражения звуков, более знакомого как эхо;

– эффект, имитирующий многоголосое исполнения с помощью копирования и небольшого изменения исходного звука.

Локальные композиции имеют свой формат кодирования. Аудиоформат – формат представления звуковых данных, используемый при цифровой звукозаписи, а также для дальнейшего хранения записанного материала. Аудио в основном представлены форматом MP3, но могут встретиться другие форматы, к тому же при прослушивании формат аудио-потока может быть другой. Для устранения проблем с расшифровкой разных форматов и для совместимости с разными аудио были выбраны наиболее часто встречающиеся форматы, такие как: AAC, AC3, APE, MPC, TTA, MP3, ALAC, FLAC, OPUS, WEBM/MATROSKA, WMA, WAV.

2.6 Блок работы с плейлистами

Работа с плейлистами является довольно простой, но громоздкой в реализации. Для упрощения этого необходим блок работы с плейлистами. Его функция не только составить плейлисты, но и работать с локальным хранилищем.

Плейлисты представляют собой список из композиций, который можно воспроизводить последовательно или в случайном порядке. Он составляется пользователем из композиций на его устройстве, то есть ПК, и стриминговых

композиций, для которых необходим интернет. Локальные композиции имеют абсолютные ссылки на файл, стриминговые – ссылки в серверном хранилище. Затем синхронизируется с локальной и серверной базами данных.

Кроме составленных плейлистов вручную, необходимо помнить, что пользователь может проиграть какую-то папку с файлами. Тогда плейлист создается временный, он не синхронизируется с базами данных.

Уже составленные плейлисты передаются в блок воспроизведения музыки, где осуществляется доступ к аудиофайлам, их расшифровка и проигрывание.

Стоит так же добавить, что на странице плейлистов будет возможность синхронизировать свои некоторые аудиозаписи с сервером. Это значит, что будет отдельный плейлист, который будет получен с сервера с выделенной памяти пользователю. При нажатии на кнопку синхронизации на композиции, она будет добавлена на сервер, а при переходе в серверный плейлист будут представлены все загруженные композиции с возможностью их удаления.

2.7 Блок радио

Данный блок является самым простым блоком в приложении. Его функции – это представлять плейлисты проигранных композиций и передавать стриминговый поток в блок воспроизведения аудио.

Так как доступ к радио идет через интернет, необходимо удостовериться, что есть онлайн-подключение. К сайтам, на которых это радио размещено, обращается приложение, скачивает гипертекст, затем обрабатывает его регулярным выражением для получения списка проигранных композиций. Регулярное выражение — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. Для поиска используется строка-образец, состоящая из символов и метасимволов и задающая правило поиска.

Список радиостанций можно было бы хранить в локальной базе данных, но из-за того, что пользователь может случайно удалить файл, было принято решение хранить этот список в удаленной базе на сервере.

2.8 Блок совершения подписок

Этот блок не будет полностью реализован, так как необходимо подключать службы, которые могут провести денежную операцию.

К этому блоку идет обращение с блока приложения. Но из-за того, что нельзя подключить реальную денежную оплату, этот блок будет выполнять тестовую функцию. Пользователь будет вручную выбирать предложенные варианты подписки на стриминговый сервис на несколько месяцев.

Платеж представляет собой покупку дополнительного пространства на сервере для синхронизации своих аудиозаписей, потому что плейлисты синхронизируются бесплатно в любом количестве. Цена за дополнительное

место будет установлена позже, но для теста функциональности будут единые фиксированные тарифы, предоставляющие разные объемы памяти на сервере. Пользователь также будет вручную выбирать наиболее подходящий ему тариф.

2.9 Блок стримингового сервиса

Этот блок устанавливает связь со стриминговой частью сервера. Он становится доступным только после совершения подписки на стриминговый сервис. Главными особенностями этого блока является расширенный функционал страниц: вместо обычной страницы с плейлистами он должен предоставлять доступ к описанию исполнителей, альбомов, популярных плейлистов, а также отображать наиболее популярные альбомы и композиции, совмещая на одной странице.

Блок стримингового сервиса в первую очередь взаимодействует с блоком приложения, чтобы правильно отображать полученную информацию с сервера. Также главным блоком связи является взаимодействие с блоком взаимодействия с сервером. Именно он выполняет запросы на сервер, затем на экране приложения отображаются разные страницы: исполнители, альбомы, плейлисты, которые не хранятся в локальной базе данных.

2.10 Блок работы с дополнительной памятью

Дополнительная память – выделенная память на сервере по подписке для пользователя, где будут хранится его личные аудиозаписи для синхронизации. Он становится доступным только после совершения подписки на дополнительную память, которая осуществляется на год, далее ее нужно обновлять.

Блок доступен из приложения, далее осуществляет запрос на сервер и отображаются синхронизированные плейлисты и композиции. Композиции можно удалить с сервера, чтобы освободить место, а из блока работы с плейлистами можно синхронизировать каждую композицию отдельно, то есть поместить на сервер.

2.11 Блок взаимодействия с сервером

Этот блок предоставляет доступ к бэкэнд базе Firebase, создает клиентское подключение к Firestore. Для этого будет использоваться специальная библиотека. Функции этого блока:

- работа с аутентификацией пользователя;
- совершение подписок;
- поиск по композициям;
- загрузка или удаление композиций из выделенной памяти;

– синхронизация плейлистов;
– проигрывание композиций.

Любой блок, которому необходимо будет обратиться к серверу, должен будет обратиться к классу и передать параметры, потом этот класс обратиться к серверу, совершит нужное действие и обработает ошибки в случае их возникновения.

Таким образом, этот класс – своеобразный интерфейс доступа к серверу, который выполняет безопасное подключение и обмен данными.

2.12 Блок сервера

Этот блок не является частью программы, так как он является самим сервером, предоставляющим услугу Backend-as-a-Service. Основными его двумя компонентами служат хранилища Firestore database и Storage. Первое представляет собой нереляционную базу данных, хранящую данные в формате JSON. Storage хранит данные, то есть аудиофайлы, доступ к которым осуществляется по ссылкам в Firestore database.

Существует возможность аутентификации через веб-сервисы, такие как Google, Facebook, номер телефона. Но в нашем случае пока будет использовать аутентификацию почтой с паролем, в будущем планируется добавить аутентификацию через дополнительные сервисы.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В текущей главе описано функционирование приложения и его структура с точки зрения описания данных и обрабатывающих их подпрограмм.

Описание функционального проектирования основывается на следующих уровнях логики приложения:

– уровень базы данных;
– уровень классов;
– уровень представления.

Для характеристики сущностей базы данных разработана модель данных. Для описания классов и их взаимодействий создана диаграмма классов, отображенная на чертеже ГУИР.400201.036 РР.1.

3.1 Локальная база данных

В приложении была использована реляционная база данных, которая представляет собой множество таблиц, имеющих связи между собой. Каждая таблица характеризуется набором строк, несущих в себе информацию о целостной и неделимой сущности, представляющей собой единицу данных, над которой в последствии функционирования приложения будут произведены операции добавления, изменения или удаления. Таким образом, в данном подразделе будет описана структура каждой таблицы базы данных и типы связей между таблицами. Схема базы данных отображена на чертеже ГУИР.400201.036 РР.3.

3.1.1 Таблица `Users`

Одна из таблиц базы данных, имеющая название `Users`, обеспечивает хранение данных о пользователях приложения. Так как пользователи не всегда имеют доступ в Интернет, необходимо обеспечить хранение всех необходимых данных локально. Следующие поля представляют данные о пользователях:

– `Id_user` – уникальный идентификатор пользователя, являющийся первичным ключом. Такой идентификатор позволит создавать расширяемые сервисы для разрабатываемой системы и избежать возможных конфликтов моделей данных, связанных с совпадением идентификаторов. `Id`, как и любой первичный ключ, обязателен, в базе данных имеет тип `uniqueidentifier`;

– `Name` – имя пользователя, которое будет отображаться на экране. Это имя может быть потом изменено пользователем. Тип данных `nvarchar(50)`;

– Surname – фамилия пользователя, которая будет отображаться на экране. Она может быть потом изменена пользователем. Тип данных nvarchar(50);

– Login – почта, к которой привязан аккаунт пользователя. Имя почты должно быть уникальным, как и идентификатор. Его нельзя использовать вместо идентификатора, потому что пользователь может в будущем поменять привязанную к аккаунту почту, в результате чего все данные будут потеряны. Тип данных nvarchar(50);

– Password – пароль пользователя. Пароль имеет свои ограничения, которые будут описаны далее в записке. Тип данных nvarchar(50);

– Create_date – данные о дате регистрации пользователя. Это поле будет использоваться в статистике пользователя, а также с целью предоставления бонусов за приверженность сервису;

– User_image_source – путь к файлу аватара пользователя. Если фотографии нет, то путь – пустая строка, иначе это путь локальный относительный или полный серверный, если фотография хранится на ПК или сервере соответственно. Тип данных nvarchar(MAX);

– Subscription_date – данные о дате окончания подписки пользователя на стриминговый сервис. Это поле изначально равно NULL, так как нет подписки, затем после совершения платежа изменяется на дату последнего дня доступа к стриминговому сервису;

– N_GB – количество купленных гигабайт на сервере для синхронизации треков пользователя. Тип данных int;

– GB_date – данные о дате окончания подписки пользователя на дополнительную память на сервере. Это поле изначально равно NULL, так как нет подписки, затем после совершения платежа изменяется на дату последнего дня доступа к памяти на сервере.

3.1.2 Таблица Songs

Данная таблица позволяет хранить данные о песнях, которые расположены на локальном устройстве или удаленно. Для этого необходимо знать только основную информацию о песнях и место их расположения. Периодически таблица будет обновляться, чтобы исключить случаи недействительных ссылок на композиции путем или удаления этой записи, или изменения пути. Используемые поля данных:

– Id_song – уникальный идентификатор композиции, являющийся первичным ключом. Такой идентификатор позволит отличать разные записи в таблице, которых будет много. Id, как и любой первичный ключ, обязателен, в базе данных имеет тип uniqueidentifier;

– `Type` – тип композиции в зависимости от ее расположения: локальная, стриминговая или синхронизированная. Тип данных `int`;

– `Full_name` – не уникальное имя композиции, взятое с названия файла. В идеальном случае должно быть представлено именем исполнителя и названием композиции, разделенных через тире, чтобы в случае проблем с метаданными быть использовано в названии. Если композиция из стримингового сервиса, то имя складывается автоматически. Тип данных `nvarchar(MAX);`

– `Name` – имя самой композиции, которое берется из метаданных файла или стримингового сервиса. Тип данных `nvarchar(MAX);`

– `Artist` – имя исполнителя песни, которое берется из метаданных файла или стримингового сервиса. Тип данных `nvarchar(MAX);`

– `Album` – название альбома, из которого взята композиция, которое берется из метаданных файла или стримингового сервиса. Тип данных `nvarchar(MAX);`

– `Length` – временная длина композиции, которое берется из метаданных файла или стримингового сервиса. Тип данных `nvarchar(MAX);`

– `Path` – путь к этой композиции, который является полным и абсолютным, то есть независимым от расположения использующего его приложения. Он может быть устаревшим из-за удаления или перемещения пользователем, из-за чего композиция может быть удалена или изменен путь на новый. Тип данных `nvarchar(MAX)`.

3.1.3 Таблица `Playlists`

Данная таблица представляет собой перечень плейлистов, которые помечаются уникальными номерами пользователей. Она не агрегирует композиции в отличие от следующей описанной таблицы. Поля таблицы:

– `Id_user` – уникальный идентификатор пользователя, является внешним ключом. В базе данных имеет тип `uniqueidentifier`;

– `Id_playlist` – уникальный первичный ключ плейлиста, по которому к нему осуществляется доступ. Является первичным ключом. В базе данных имеет тип `uniqueidentifier`;

– `Name` – имя плейлиста. Не должно быть уникальным в таблице, но будет проверятся на уникальность у пользователя. Тип данных `nvarchar(50);`

– `Last_sync` – дата и время последнего синхронизирования плейлиста с сервером. Если приложение найдет различия во временах синхронизации на сервере и в приложении, то плейлист обновится со старой на более новую версию;

– `Last_update` – дата и время последнего изменения плейлиста. Это поле обновляется при изменении плейлиста, чтобы затем автоматически синхронизировать с сервером изменения;

– `Playlist_image_source` – путь к файлу аватара плейлиста. Если фотографии нет, то путь – пустая строка, иначе это путь локальный относительный или полный серверный, если фотография хранится на ПК или сервере соответственно. Тип данных `nvarchar(MAX)`.

3.1.4 Таблица `Playlist_Songs`

Данная таблица агрегирует композиции из таблицы `Songs`, составляя из них последовательности, которые и являются плейлистом. Алгоритм плеера выполняет сборку отдельных композиций в плейлисты, чтобы подать готовый список на проигрывание. Поля таблицы следующие:

– `Id_playlist` – внешний ключ плейлиста, по которому к нему осуществляется доступ. В базе данных имеет тип `uniqueidentifier`;

– `Id_song` – внешний ключ и идентификатор композиции. В базе данных имеет тип `uniqueidentifier`;

– `N_sequence` – порядок композиции в альбоме. В базе данных имеет тип `int`. Выставляется алгоритмом, формирующим плейлист.

3.1.5 Таблица `Radio`

Данная таблица позволяет хранить ссылки на сайты радиостанций. Эта таблица самая простая. Она полностью скачивается при инициализации приложения и со временем обновляется. Используемые поля данных:

– `Id_radio` – уникальный номер радиостанции, который генерируется при добавлении радиостанции разработчиком. В базе данных имеет тип `uniqueidentifier`;

– `Radio` – уникальное имя радиостанции, которое тоже задается разработчиком и обязательно содержит достоверное название. Тип данных `nvarchar(30);`

– `Description` – строка-описание радиостанции, которую создают и редактируют разработчики, добавляя радиостанцию. Тип данных `nvarchar(MAX);`

– `Stream` – URL потока стрима радио для прослушивания. Тип данных `nvarchar(MAX);`

– `Page` – URL радио, откуда можно получать информацию о проигрывающихся композициях: название, исполнитель и длина трека. Тип данных `nvarchar(MAX);`

– `All_regex` – регулярное выражение, которое позволяет выделить в HTML документе общую строку композиции. Тип данных `nvarchar(20);`

– Name_regex – регулярное выражение, которое позволяет выделить в общей строке название композиции. Тип данных nvarchar(20);

– Artist_regex – регулярное выражение, которое позволяет выделить в общей строке имя исполнителя. Тип данных nvarchar(20);

– Radio_image_source – путь картинке-логотипу радио. Тип данных nvarchar(MAX).

3.2 Серверная пользовательская база данных

В качестве сервера была выбрана платформа Firebase. К сожалению, база данных Firestore является нереляционной. Это значит, что записи хранятся не таблицами в упорядоченном виде, а записями формата JSON, между которыми нет прямых связей. В Firebase Firestore таблицы в классическом понимании представляют собой коллекции. Схема базы данных Firestore отображена на чертеже ГУИР.400201.036 РР.4. Далее будет приведено описание коллекций и содержащихся в них документов.

3.2.1 Коллекция Users

Одна из коллекция базы данных, имеющая название Users, как и в локальной базе данных, обеспечивает хранение данных о пользователях приложения. Содержит те же поля, что и локальная таблица Users, но с дополнительными полями. Составлена из документов, уникальные имена которых представлены уникальным Id_user. Поля документа пользователя:

– Id_user;
– Name;
– Surname;
– Login;
– Password;
– Create_date;
– User_image_source;
– Subscription_date;
– N_GB;
– GB_date.

3.2.2 Коллекция UserPlaylists

Эта коллекция является собирающей для коллекций Playlists и Songs. Из-за формата хранения данных очень удобно создать общую коллекцию, которая будет хранить документы, названные уникальным номером пользователя, в которых будут созданы еще коллекции. Данная модель упростит доступ к данным пользователей и не будет неудобна, потому что отсутствует глобальный поиск, как на стриминговой базе данных.

3.2.3 Коллекция `Playlists`

Данная коллекция представляет собой перечень плейлистов, которые помечаются уникальными номерами пользователей. Она агрегирует композиции в отличие от таблицы локальной базы данных. Она не находится в корневом каталоге базы данных. Доступ к документу осуществляется через создание новой коллекции: сначала создается коллекция `UserPlaylists`, в которой создается документ, названный идентификатором пользователя, в котором содержится коллекция `Playlists`. И уже в этой коллекции находятся плейлисты пользователя, которые представляют собой документы, названные идентификатором плейлиста. Данный документ повторяет одноименную таблицу в локальной базе данных и агрегирует таблицу `PlaylistSongs`. Поля документа:

- `Id_playlist`;
- `Name`;
- `Last_sync`;
- `Last_update`;
- `Playlist_image_source`;
- `Songs` – это поле представляет собой фактически содержание таблицы `PlaylistSongs`, где композициям в плейлисте присвоен порядковый номер. Это строка, в которую записаны разделенные знаками препинания идентификаторы и порядковые номера композиций, по типу «`Id_song1 = N1 , Id_song2 = N2 , Id_song3 = N3`».

3.2.4 Коллекция `Songs`

Данная коллекция позволяет хранить данные о песнях, которые расположены в локальной или выделенной на сервере памяти для синхронизации. Эта таблица практически не отличается от одноименной в локальной БД и так же содержит основную информацию о всех композициях. Но как и таблица плейлистов не является корневой коллекцией: сначала создается коллекция `UserPlaylists`, в которой создается документ, названный идентификатором пользователя, в котором содержится коллекция `Songs`. И уже в этой коллекции находятся композиции пользователя, представленные документами с названием ID композиции. Используемые поля данных:

- `Id_song`;
- `Type`;
- `Full_name`;
- `Name`;
- `Artist`;
- `Album`;

– Length;

– Path_local – путь на локальном устройстве, если при синхронизации композиция не скачивалась на сервер для хранения;

– Path_server – путь на сервере, если композиция стриминговая или скачана на дополнительное пространство пользователя во время синхронизации.

3.2.5 Коллекция Payments

Данная коллекция содержит все платежи, которые когда-либо производил пользователь. Она находится в документе пользователя Users. Составлена из документов, уникальные имена которых представлены уникальным Id_payment. Поля документа:

– Id_payment – уникальный идентификатор платежа, потому что все платежи должны сохраняться;

– Money – сумма денег, которую заплатил пользователь, выражается только в долларах. Несмотря на фиксированные цены, необходимо хранить сумму, потому что цены со временем будут изменятся, нужно будет предоставить пользователю и другим запрашивающим лицам отчет;

– Code – код услуги: подписка или покупка дополнительного места, задается константами;

– GBs – количество купленных гигабайт места на сервере. Только целое значение. В случае безлимитной покупки их количество равно константе, кодирующей безлимитное место;

– Subscription – длина подписки на стриминговый сервис или место в месяцах;

– Date_time – дата и время совершения платежа.

3.3 Серверная стриминговая база данных

Эта база данных отличается от пользовательской тем, что теперь здесь в описаниях композиций, альбомов и плейлистов будут содержаться только внешние ключи-идентификаторы, которые в обычной базе данных являются типом uniqueidentifier. Также будет изменен принцип построения БД: будут добавлены дополнительные поля, чтобы быстрее осуществлять доступ к разным типам записей. Идея о вложенности коллекций композиции в альбомы, альбомы в артистов, не является оптимальной, хотя и распространена для такого типа организации баз данных, потому что сразу затрудняется поиск композиций по названию. Она представлена на чертеже вместе с пользовательской базой данных.

3.3.1 Коллекция `Artists`

Данная коллекция является стриминговой и позволяет хранить данные об исполнителях в документах, названных уникальным номером исполнителя. Кроме названия группы или имени исполнителя есть возможность прикрепить фотографию и описание исполнителя. Используемые поля данных в документе:

- `Id_artist` – уникальный `uniqueidentifier` исполнителя, который формируется при загрузке на сервер;
- `Name` – не уникальное имя исполнителя. Нет ограничений для названия;
- `Description` – строка-описание этого исполнителя, которую создают и редактируют владельцы прав на песни этого исполнителя;
- `Path_to_image` – путь к фотографии этого исполнителя, если она имеется.
- `Id_albums` – идентификаторы альбомов этого исполнителя, если они имеются, чтобы не искать в базе данных по ID.
- `Id_songs` – идентификаторы композиций этого исполнителя, которые не вошли в альбомы, если они имеются.

3.3.2 Коллекция `Albums`

Данная коллекция позволяет хранить данные об альбомах в документах, названных уникальным номером альбома. Кроме названия альбома есть возможность прикрепить фотографию и описание исполнителя. Используемые поля данных в документе:

- `Id_album` – уникальный `uniqueidentifier` альбома, который формируется при загрузке на сервер этого альбома;
- `Id_artist` – уникальный `uniqueidentifier` исполнителя, который указывается при загрузке на сервер;
- `Name` – не уникальное имя альбома. Нет ограничений для названия;
- `Ids_songs` – так как база данных не реляционная, то мы в праве указывать несколько ссылок на другие композиции. Они указаны как `Id_song`, разделенные запятыми и написанными уже в правильном порядке, как задумано в альбоме;
- `Description` – строка-описание этого альбома, которую создают и редактируют владельцы прав на этот альбом;
- `Path_to_image` – путь к обложке этого альбома, если она имеется.

3.3.3 Коллекция `Songs`

Данная коллекция позволяет хранить данные об альбомах в документах, названных уникальным номером альбома. Кроме названия альбома есть

возможность прикрепить фотографию и описание исполнителя. Используемые поля данных в документе:

– Id_song – уникальный uniqueidentifier композиции, который формируется при загрузке на сервер;

– Id_album – уникальный uniqueidentifier альбома, который формируется при загрузке на сервер этого альбома;

– Id_artist – уникальный uniqueidentifier исполнителя, который указывается при загрузке на сервер;

– Name – не уникальное имя композиции. Нет ограничений для названия;

– Length – длина композиции в секундах;

– Path – путь к композиции в хранилище.

3.3.4 Коллекция Radio

Данная коллекция позволяет хранить ссылки на сайты радиостанций и их описание. Эта таблица самая простая и полностью повторяет аналогичную таблицу в локальной базе данных. Используемые поля данных:

– Id_radio;

– Radio;

– Description;

– Stream;

– Page;

– All_regex;

– Name_regex;

– Artist_regex;

– Radio_image_source.

3.3.5 Таблица Playlists

– Данная коллекция представляет собой перечень плейлистов, которые помечаются уникальными номерами. Эти плейлисты составлены пользователями или модераторами и включают только стриминговые композиции. В отличие от пользовательских плейлистов она включает описание плейлиста и метаданные. Правильное название скорее будет подборка, но для ясности оставим плейлист. Поля документа:

– Id_playlist – уникальный идентификатор плейлиста;

– Id_user – имя создателя подборки;

– Name – имя плейлиста. Не должно быть уникальным в таблице, но общие данные имя и пользователь должны быть уникальны;

– Create_date – дата и время создания;

– Last_update – дата и время последнего изменения плейлиста;

– `Songs` – строка, в которую записаны разделенные знаками препинания идентификаторы и порядковые номера композиций, по типу «`Id_song1 = N1 , Id_song2 = N2 , Id_song3 = N3`»;

– `Meta` – метаданные плейлиста. Представляются строкой-словарем и задаются разработчиками сервиса;

– `Path_to_image` – путь к файлу аватара плейлиста. Если фотографии нет, то путь – пустая строка, иначе это путь полный серверный.

3.4 Хранилище данных

Хранилище данных в Firebase называется Storage. Данные на платформе Firebase хранятся и распределяются так же, как и в обычных устройствах, то есть в директориях (папках). Необходимо определить, как будет устроена файловая система в хранилище.

Все данные в системе представлены только двумя типами файлов: композиции и изображения. Доступ ко всем файлам осуществляется по прямому адресу, поэтому есть необходимость группировать композиции и изображения в папки.

Пользовательская часть каталога будет представлена следующим образом: создается общая папка `Users`, в которой создаются папки конкретных пользователей, имя которых – идентификатор пользователя. Далее в папке файлом хранится изображение пользователя и находятся папки с изображениями плейлистов и синхронизированными композициями соответственно. Такое разграничение необходимо, чтобы четко задать допустимое место на сервере пользователю, потому что изображения синхронизируются бесплатно, а за место для композиций приходится платить. Именно создание отдельной папки композиций поможет контролировать место, чтобы одна и та же композиция, находящаяся в разных плейлистах, не закачивалась на сервер дважды.

Стриминговая часть каталога – вложенные папки в следующей очередности: артисты, альбомы, композиции. В общей папке создаются папки артистов, содержащие их изображение и папки с альбомами. В свою очередь папки с альбомами содержат изображения этих альбомов и файлы композиций. Все папки называются соответствующими идентификаторами, поэтому при необходимости получить доступ к песне путь к ней будет единственным и универсальным.

3.5 Классы приложения

Приложение построено на WinForms, которое априори включает использование классов. В этом подразделе будут описаны использующиеся классы, большинство из которые по совместительству являются страницами

приложения, некоторые статические. Диаграмма классов представлена на чертеже ГУИР.400201.036 РР.1.

3.5.1 Класс `MainWindow`

Этот класс является классом Windows Forms и отвечает за взаимодействие с пользователем. Основная его функция – это представлять целостность приложения через главную страницу. Это первая страница, которую видит пользователь после того, как успешно вошел в систему. Этот класс агрегирует все страницы приложения:

- `SubPage` – страница магазина, где можно увидеть цену на дополнительные возможности приложения и совершить покупку;
- `SettingsPage` – страница настроек, где можно настроить проигрывание композиций с эффектами;
- `radioPage` – страница радио, где можно выбрать радиостанции и просмотреть их последние треки;
- `MemoryPage` – страница дополнительной памяти, где можно изменить загруженные треки;
- `StreamPage` – страница стримингового сервиса, где можно увидеть онлайн-композиции;
- `MPPage` – страница плейлистов, где можно выбрать плейлист для проигрывания, создать, изменить и удалить.

Также на этой странице строка с проигрываемой дорожкой.

Функции этого класса, которые выполняют функции интерфейса и являются событиями:

- `void MainWindow_SourceInitialized` - событие, направленное на правильную работу окна;
- `void MinimizeWindow_Executed`- событие скрытия окна;
- `void MaximizeWindow_Executed` - событие "на полный экран";
- `void CloseWindow_Executed` - событие закрытия приложения;
- `void CloseMenu_Click` - событие скрытия меню;
- `void OpenMenu_Click` - событие развертывания меню;
- `void Radio_MouseDown` - событие перехода к странице радио;
- `void Playlists_MouseDown` - событие перехода к странице плейлистов;
- `void Shop_MouseDown` - событие перехода к магазину;
- `void Stream_MouseDown` - событие перехода к стриминговому сервису;
- `void Memory_MouseDown` – событие перехода к памяти;
- `void Play_Click` - событие запуска воспроизведения;

– `void Stop_Click` - события останова воспроизведения;
– `void LogOut_MouseDown` - событие выхода из аккаунта;
– `void Ellipse_MouseDown` - событие выхода из аккаунта;
– `void Search_Click` - событие поиска
– `void NextButton` - событие перехода на новую композицию;
– `void EndButton` - событие перехода на предыдущую композицию;
– `void Settings_MouseDown` - событие перехода на страницу настроек;
– `void TimeLine_ValueChanged` - событие перемотки композиции.

3.5.2 Класс `Sign_Up`

Данный класс представляет страницу пользователя и выполняет функцию регистрации. Для регистрации нужны почта и пароль, не обязательно фотография. В классе проверяется правильность написания почты и пароля, чтобы пользовательский аккаунт был надежно защищен. После создания записи она заносится в локальную базу данных, затем при подключении к интернету на сервер.

В страничном классе заданы следующие переменные:
– `string name` – имя, указанное пользователем;
– `string surname` – фамилия;
– `string login` – логин, который является почтой;
– `string password` – пароль, имеющий свои правила;
– `string createDate` – дата регистрации;
– `string imageSrc` – ссылка на фотографию пользователя, которая указывается по желанию;
– `bool BName` – правильность имени;
– `bool BSurname` – правильность фамилии;
– `bool BLogin` – правильность логина;
– `bool BPassword` – правильность пароля;
– `bool load` – флаг разрешения загрузки;
– `Guid Id_user` – сгенерированный идентификатор пользователя;
– `CroppedBitmap cb` – фотография пользователя.

Для проверки правильности ввода имени, почты и пароля заданы следующие регулярные выражения:
– `RName` – проверка имени;
– `RSurname` – проверка фамилии;
– `RLogin` – проверка почты: должна включать знак почты «@» и домен;

– `RPassword` – проверка пароля: должен быть более восьми символов и включать не менее одной цифры, знака препинания, заглавной буквы.

Функции этого класса также представлены событиями, внутри которых происходят все проверки и занесение в базу данных:

– `void Photo_Click` – выбор фотографии пользователя. В этой функции происходит обрезка до 600x600 пикселей;

– `void Sign_Up_Click` – окончательное создание нового аккаунта с проверкой заполненных полей и занесение в базу данных.

3.5.3 Класс `Log_In`

Этот класс позволяет пользователю войти в приложение, и также представляет собой страницу. Чтобы войти в приложение, нужно заполнить логин и пароль, затем класс обратиться к базе данных, чтобы посмотреть, есть ли такой пользователь в системе.

Поля класса:

– `Guid id` – идентификационный номер пользователя;

– `string login` – логин пользователя;

– `string password` – пароль пользователя;

– `bool BLogin` – правильность логина;

– `bool BPassword` – правильность пароля.

Для проверки правильности ввода почты и пароля перед обращением к базе данных заданы следующие регулярные выражения:

– `RLogin` – проверка почты: должна включать знак почты «@» и домен;

– `RPassword` – проверка пароля: должен быть более восьми символов и включать не менее одной цифры, знака препинания, заглавной буквы.

Функция этого класса также представлена событием, внутри которого происходят все проверки и обращение в базу данных: `void Log_In_Click`. Если входа не происходит, все переменные обнуляются.

3.5.4 Класс `Profile`

После регистрации или входа в аккаунт заполняется статический класс профиля пользователя. Он используется во всём приложении, поэтому из любого файла можно напрямую обратиться за данными пользователя. Этот класс простой и содержит в себе основные поля, описывающие пользователя:

– `Guid Id_user` – идентификатор;

– `string name` – имя;

– `string surname` – фамилия;

– `string login` – логин;

– `string passworg` – пароль;
– `string createDate` - дата регистрации;
– `string imageSrc` – аватар;
– `int gbs` – количество купленных гигабайт, если доступен доступ к серверу;
– `int subscription` – наличие подписки, если доступен доступ к серверу;
– `datetime subscription_date` – дата окончания подписки, если доступен доступ к серверу;
– `datetime gbs_date` – дата окончания купленного места, если доступен доступ к серверу.

Также этот класс содержит константу `UNLIM_GB`, которая описывает кодовое количество гигабайт при подписке на безлимитную память. Эта константа равна 100.

3.5.5 Класс `Settings`

Класс настроек отображает страницу настроек аудиодорожки. Про сами настройки было описано в разделе системного проектирования.

Используется объект класса `DispatcherTimer timer` для того, чтобы настраивать перемотку композиций в функции `SSR_ValueChanged`. Также используется объект класса `BASS_DX8_PARAMEQ par` для создания баса в аудио.

Обработчики событий изменения настроек эквалайзера:
– `void Volume_ValueChanged` – изменение звука;
– `void SSR_ValueChanged` – изменение дискретизации аудиокомпозиции;
– `void First_ValueChanged` – изменение первого параметра эквалайзера;
– `void Second_ValueChanged` – изменение второго параметра эквалайзера;
– `void Third_ValueChanged` – изменение третьего параметра эквалайзера;
– `void Fourth_ValueChanged` – изменение четвертого параметра эквалайзера;
– `void Fifth_ValueChanged` – изменение пятого параметра эквалайзера;
– `void Sixth_ValueChanged` – изменение шестого параметра эквалайзера;
– `void Seventh_ValueChanged` – изменение седьмого параметра эквалайзера;

– `void Eighth_ValueChanged` – изменение восьмого параметра эквалайзера;

– `void Nineth_ValueChanged` – изменение девятого параметра эквалайзера;

– `void Chorus_ValueChanged` – изменение возможности проигрывать звук в эффекте, имитирующий многоголосое исполнения с помощью копирования и небольшого изменения исходного звука;

– `void Echo_ValueChanged` – изменение эффекта множественного отражения звуков, более знакомого как эхо.

3.5.6 Класс `Subscribe`

Класс подписок представляет собой страницу с предложениями о покупке подписки на стриминговый сервис и дополнительной покупке гигабайт на серверном пространстве. Этот класс соединяется с сервером для получения актуальной информации о предложениях и совершения подписки. Все цены и предложения заложены в странице класса.

Используемые поля класса:

– `int code_s` – код подписки на стриминговый сервис;

– `int code_gb` – код подписки на память.

Функции этого класса представлены событиями, внутри которых происходят все проверки и работа с сервером:

– `void s3_Click` – кнопка покупки подписки на стриминговый сервис на 3 месяца;

– `void s6_Click` – кнопка покупки подписки на стриминговый сервис на 6 месяцев;

– `void s12_Click` – кнопка покупки подписки на стриминговый сервис на 12 месяцев;

– `void gb5_Click` – кнопка покупки 5 дополнительных гигабайт на сервере на год;

– `void gb10_Click` – кнопка покупки 10 дополнительных гигабайт на сервере на год;

– `void unlim_Click` – кнопка покупки безлимитной памяти на сервере на год;

– `void checkSubscribtion` – функция проверки подписки;

– `void updateSubscribtion` – функция отображения подписки на странице.

3.5.7 Классы радио

Класс `Radio` является страничным и отвечает за обращение к базе данных или серверу при обновлении списка для получения списка радиостанций.

Функция класса `List<Compositions> Composition` – создание листа композиций радиостанций для отображения на странице.

Класс `RadioComposition` является хранилищем актуальной информации о проигрываемых композициях на радиостанции. Его поля:

- `string Time` – время проигрываемой композиции;
- `string Artist` – исполнитель;
- `string Track` – название композиции.

Класс `RadioStation` является хранилищем актуальной информации о проигрываемых композициях на радиостанции. Его поля:

- `int Id` – идентификатор станции;
- `string name` – имя;
- `string descr` – описание;
- `string page` – страница радио с историей;
- `string stream` – ссылка на радиострим;
- `string imageSrc` – путь к картинке радио.

Класс `RadioControl` отвечает за подключение к радиостанциям и отображение их проигрывающихся композиций в данный момент времени, то есть обновление по таймеру. Он составляет часть страницы радио и отображает ячейку радиостанции.

События класса:

- `void StartRadio_Click` – кнопка запуска радиостанции;
- `void StopRadio_Click` – кнопка остановки воспроизведения;
- `void Grid_MouseEnter` – функция подсвечивания при наведении мышкой;
- `void Grid_MouseLeave` – убирание подсвечивание при наведении мышки;
- `Share_Click` – кнопка копирования в буфер обмена играющей композиции.

Класс `RadioPlaylistsControl` отвечает за отображение композиций, которые являются пройденным плейлистом радиостанции. Можно копировать название каждой композиции наведением и кликом мышки по значку. Он является страничным, и открывается по клику на картинку радио на странице радио.

Список функций:

- `void Share_Click` – копирование названия композиции в буфер обмена;
- `void Grid_MouseEnter` – изменение окошка на подсвечивающееся при наведении мышки;
- `void Grid_MouseLeave` – удаление подсветки с окошка при отведении курсора мышки.

3.5.8 Класс UserPlaylist

Этот класс представляет плейлист как структуру. Он используется везде, где нужно обращаться с плейлистами. Его поля:

- `string NULL_NAME` – кодовое имя плейлиста, отправленного на удаление;
- `Guid Id_playlist` – идентификатор плейлиста;
- `Guid Id_user` – идентификатор пользователя;
- `string name` – имя плейлиста;
- `string imageSrc` – путь к обложке плейлиста;
- `DateTime lastSync` – дата и время последней синхронизации с сервером;
- `DateTime lastUpdate` – дата и время обновления;
- `List<Song> songs` – список песен.

3.5.9 Классы работы с плейлистами

Работа с плейлистами начинается после открытия страницы со своими плейлистами. За эту страницу отвечает страничный класс `My_playlists`. Он отображает список плейлистов, с него можно добавить новый плейлист и синхронизировать. За эти функции отвечают следующие методы:

- `void Button_Click` – добавление плейлиста;
- `void Sync_MouseDown` – синхронизация плейлистов.

Для реализации страницы создания плейлиста предназначен класс `CreatePlaylist`. Этот класс позволяет создать новый плейлист, дать ему название, добавить фотографию и указать, на основе какой папки с музыкой он будет создан.

Используемые переменные:

- `CroppedBitmap cb` – создание класса разбивки окна на поля, по которым потом будет ориентироваться компоновка записей в листе;
- `Regex regx_empty` – регулярное выражение для определения пустого имени плейлиста;
- `Regex regx_name` – регулярное выражение для определения правильного имени плейлиста;
- `List<Song> songs` – список песен при добавлении папки;
- `List<Song> songs_all` – список песен на добавление в базу данных.

В классе есть функции не только обработчики событий, но и получения информации. Список добавленных функций:

- `void Image_Click` – добавление картинки-аватара плейлиста;
- `void Create_Click` – создание нового плейлиста. Кнопка подтверждения;
- `void Add_Click` – добавления папки;

– `void CloseWindow_Executed` – закрытие окна.

За отображение плейлистов ответственен класс `PlaylistControl`. Он отображает плейлист на странице плейлистов, который содержит в поле `playlist_`. При клике на него открывается страница плейлиста, а при клике на минус – удаление. За эти функции отвечают следующие методы:

– `void Delete_Click` – добавление плейлиста;

– `void Image_MouseDown` – синхронизация плейлистов.

После открытия плейлиста отображается класс `Playlist`. Он отображает все композиции плейлиста. При клике на «+» добавляются композиции. За эту функцию отвечает метод `void Add_Click`.

3.5.10 Класс `Vars`

Это класс переменных, которые используются по всей программе. Он хранит список плейлистов для различных нужд, является буфером для плейлиста, а также его использует проигрыватель композиций, чтобы ориентироваться в порядке песен. Его поля:

– `UserPlaylist playlist` – используемый на данный момент плейлист;

– `List<Tuple<Guid, string>> allPlaylists` – список идентификаторов и названий всех плейлистов;

– `string AppPath` – строка пути приложения, которая используется для задания полного пути, ориентируясь по относительному;

– `Guid Id_playlist` – буфер идентификатора плейлиста;

– `List<Composition> current` – список композиций, формирующийся при открытии любого списка, например, альбома, плейлиста или поиска;

– `List<Composition> StreamTracklist` – список последовательных композиций, по которому непосредственно работает проигрыватель;

– `int CurrentTrackNumber` – указатель на композицию в списке, ее порядковый номер.

3.5.11 Класс `MusicStream`

Данный класс обрабатывает всю имеющуюся информацию о том, что нужно воспроизводить и пускает дорожку. Он получает информацию о плейлистах, о доступе к композициям, о радио-дорожке и другом.

Поля класса:

– `DispatcherTimer timer` – таймер для дорожки;

– `List<int> BassPluginsHandles` - список плагинов для обработки звука;

– `int HZ = 44100` - дефолтная частота дискретизации;

– `bool InitDefaultDevice` - переменная–инициализатор;

– `bool NextPoint = false` - логическая переменная ручного переключения вперед;

– `bool EndPoint = false` - логическая переменная ручного переключения назад;

– `int Stream` – аудиоканал;

– `int Volume = 50` - громкость звука в процентном отношении;

– `bool HandlerAttached` = false - проверка подписки событий;

– `int SSR = 0` - частота дискретизации в процентном отношении;

– `Echo = 0` – эхо;

– `BASS_DX8_CHORUS _chorus` – класс для воспроизведения многоголосности;

– `BASS_DX8_ECHO _echo` – класс для воспроизведения эхо;

– `int Chorus = 0` – задержка;

– `double CorTime` – время композиции;

– `int[] _fxEQ = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }` – инициализация пустыми значениями эквалайзера;

– `bool isStopped = true` - логическая переменная ручной остановки;

– `bool EndPlaylist` - логическая переменная окончания плейлиста;

– `object senderStart` - ссылка на кнопку старт в ЭУ;

– `object senderStop` - ссылка на кнопку стоп в ЭУ.

Функции класса:

– `bool InitBass` – метод-инициализатор;

– `void PlayPlayer` - метод воспроизведения потока для плеера;

– `void Timer_Tick` – обновление времени дорожки по таймеру;

– `string FormatTimeSpan` – форматирование линии для правильного отображения при проигрывании;

– `void PlayRadio` - метод воспроизведения потока для радио;

– `int GetTimeOfStream` - метод описания полного времени;

– `int GetPosOfStream` - метод описания текущего времени;

– `void StreamLineStart` - метод описания поведения кнопки START для панели воспроизведения;

– `void StreamLineStart` - метод для описания поведения кнопки START в UserControl;

– `void Play` - общий метод воспроизведения потока;

– `void Stop` - общий метод остановки потока;

– `void StreamLineStop` - метод описания поведения кнопки STOP для панели воспроизведения;

– `void StreamLineStop` - метод для описания поведения кнопки STOP в UserControl;

– `bool ToNextTrack` - метод для описания автоматического переключения;

– `bool Next` - метод для описания пренудительного переключения вперед;

– `bool End` - метод для описания принудительного переключения назад;

– `void UpdateEQ` – обновление эквалайзера;

– `void SetFXParameters` – задание параметров для эквалайзера;

– `void SetEcho` – установка параметров для эха;

– `void SetChorus` - установка параметров для многоголосности.

3.5.12 Класс `Song`

Данный класс – класс композиции. Содержит всю информацию о композиции, а также функцию `void addProperties`, которая заполняет поля метаданными файла. Извлечение метаданных из файла происходит с помощью библиотеки ftag. Если метаданные отсутствуют, поля имени и исполнителя заполняются из названия файла, разделенного на две части тире. Если такое не осуществимо, то имя и исполнитель будут подписаны, как сам файл.

Поля класса:

– `Guig Id_song` – идентификатор песни;

– `bool is_local` – пометка о локальности композиции, где `true` – это локальная композиция;

– `string full_name` – полное имя композиции. Если композиция серверная, то составляется из имени и исполнителя, иначе это название файла без расширения;

– `string name` – имя композиции;

– `string artist` – исполнитель композиции;

– `string album` – альбом композиции;

– `string path` – абсолютный путь к файлу;

– `int n_sequence` – дополнительное поле, чтобы указать порядок песни в плейлисте.

3.5.13 Класс `Composition`

В отличие от класса `Song` класс `Composition` не является композицией. Это контрол, «обертка» композиции, которая отображает композицию и позволяет управлять ей. Этот класс используется в проигрывателе, чтобы изменять цвет кнопки проигрывания, из-за этого класс `Song` не может использоваться. Но класс `Composition` содержит в себе

Song, поэтому доступ к информации о композиции осуществляется напрямую.

Поля класса:

– Song song_ – экземпляр композиции, который содержит всю информацию;

– bool search – флаг, находится ли этот контрол на странице поиска;

– int n – порядок в последовательности. Он не повторяет поле n_sequence композиции, потому что использоваться может не только в плейлистах;

– List<Tuple<Guid, string>> id_playlists – список кортежей из имени и идентификатора плейлиста;

– ComboBox playlists – элемент контроля, позволяющий выбрать плейлист, в который можно добавить композицию.

Методы класса:

– void Add_Click – кнопка добавления композиции в плейлист;

– void Drop_Click – кнопка удаления из плейлиста;

– void Start_Click – кнопка запуска композиции;

– void Stop_Click – кнопка остановки воспроизведения;

– void Sync_to_server_Click – кнопка синхронизации с сервером;

– void Grid_MouseEnter – функция подсвечивания при наведении мышкой;

– void Grid_MouseLeave – убирание подсвечивание при отведении мышки;

– Playlists_SelectionChanged – событие выбора плейлиста для добавления композиции.

3.5.14 Класс Database

Класс Database является статическим. Именно через него происходит все обращение к локальной базе данных и обрабатываются ошибки. Класс содержит строку connectionString, через которую происходит подключение к базе данных.

Методы класса:

– SqlDataReader GetUser – получение информации о пользователе;

– void InsertUser – вставка пользователя в базу данных;

– bool UpdateUser – обновление информации о пользователе, точнее его подписок;

– (bool, List<RadioStation>) GetRadio – получение всех радиостанций;

– `void InsertRadio` – вставка радиостанции в базу данных;

– `bool DeleteRadio` – удаление радиостанции;

– `(bool, List<UserPlaylist>) GetPlaylists` – получение всех плейлистов. Можно задавать параметры получения: с песнями или без;

– `bool InsertPlaylist` – вставка одного плейлиста. Обертка для функции, принимающей список;

– `bool InsertPlaylists` – вставка списка плейлистов;

– `bool UpdatePlaylist` – обновление плейлистов. Можно задавать разные параметры: только информацию, без песен;

– `bool DeletePlaylist` – удаление плейлиста. Задание параметров: удаление полностью или зануление имени;

– `(bool, List<Song>) GetSongsFromPlaylist` – получение всех песен из плейлиста. Используется более масштабными функциями;

– `bool InsertSongToPlaylist` - вставка одной песни в плейлист. Обертка для функции, принимающей список;

– `bool InsertSongsToPlaylist` - вставка всех песен в плейлист. Используется более масштабными функциями;

– `bool InsertAllSongsToPlaylist` – полностью заменить все песни в плейлисте;

– `(bool, List<Song>) GetAllSongs` – получение всех композиций из базы данных;

– `(bool, List<Tuple<Guid, string>>) GetSongsPath` – получение всех путей композиций;

– `bool InsertSong` – вставка песни в базу данных. Обертка для функции, принимающей список;

– `bool InsertSongs` – вставка списка песен в базу данных.

3.5.15 Класс `Firebase`

Класс `Firebase` является статическим. Только через него происходит все обращение к серверной базе данных и обрабатываются ошибки. Доступ к базе данных Firestore осуществляется через создание глобальной системной `GOOGLE_APPLICATION_CREDENTIALS` переменной, содержащей путь к файлу настроек, и идентификатора проекта. Доступ к хранилищу Storage осуществляется через общее пространство Google.Cloud, используя ссылку на проект.

Класс имеет следующие атрибуты, которые все являются закрытыми, потому что используются только своими функциями:

– `string project_Id` – идентификатор проекта для доступа к базе данных Firestore. Имеет значение «dco-player-74813»;

– `string bucketName` – ссылка на хранилище Storage. Имеет значение "dco-player-74813.appspot.com";

– `FirestoreDb db` – указатель база данных, который используется для доступа к Firestore;

– `CollectionReference coll_ref` – ссылка на коллекцию;

– `DocumentReference doc_ref` - – ссылка на документ в коллекции.

Класс имеет следующие операции:

– `Init` – инициализация класса: установка глобальной переменной `GOOGLE_APPLICATION_CREDENTIALS` подключение к Firebase;

– `void AddUser` – добавление пользователя;

– `void UpdateUser` – обновление пользовательских полей, описывающих актуальную подписку;

– `bool UserPayment`– совершение платежа. Создает новый документ транзакции;

– `bool CheckUser` – проверка наличия пользователя;

– `bool GetUser` – получение информации о пользователе;

– `List<RadioStation> GetRadio` – получение всех радиостанций;

– `UserPlaylist GetPlaylist` – получение плейлиста;

– `List<UserPlaylist> GetPlaylists` – получение всех плейлистов. Задается параметр получения удаленных плейлистов;

– `bool AddPlaylist` – добавление плейлиста;

– `bool AddPlaylists` – добавление списка плейлистов;

– `bool DeletePlaylist` – удаление плейлиста;

– `List<Song> GetSongs` – получение всех песен из базы данных;

– `bool AddSong` – добавить одну композицию в базу данных;

– `bool AddSongs` - добавить список композиций в базу данных;

– `bool DeleteSong` – удалить одну композицию в базе данных;

– `bool DeleteSongs` - удалить список композиций в базе данных.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В текущей главе описаны внутренние алгоритмы ключевых процедур приложения, среди которых выделены следующие операции:

- аутентификация пользователя: создание и вход;
- создание плейлиста;
- синхронизация плейлистов;
- удаление плейлиста;
- совершение подписки;
- проигрывание плейлистов;
- поиск.

Наглядное общее представление работы системы представлено на диаграмме последовательности ГУИР.400201.036 РР.2.

4.1 Аутентификация

Перед тем, как начать использовать приложение, необходимо выполнить аутентификацию пользователя. После запуска приложения на выбор предлагается два способа: создать новый аккаунт или войти в уже существующий. Стоит упомянуть, что приложение работает в первую очередь с локальной базой данных, потому что его специфика заключается в том, что пользователь работает за стационарным средством. Если в базе данных нет необходимого запрашиваемого пользователя, нужно иметь подключение к интернету.

Описание алгоритма создания профиля в функции `Sign_Up_Click`:

Шаг 1. После нажатия на кнопку регистрации устанавливаем локальную булевую переменную `load` в значение `true`. Это значение изменится на противоположное в любой части алгоритма, когда что-то пойдет не так, чтобы не дать программе загрузиться при неправильном создании аккаунта. Также устанавливаем булевые переменные `BName`, `BSurname`, `BLogin`, `BPassword` в значение `false`, чтобы потом при проверке изменить на `true`.

Шаг 2. Проверяем регулярным выражением `RName` указанное имя пользователя. Оно должно состоять из букв латинского и кириллического алфавита длиной от одного до девятнадцати символов включительно. Если текст подходит заданному выражению, то указываем булевой переменной `BName` значение `true`. Иначе выводим сообщение о правильном написании имени и завершаем функцию. Шаг 19.

Шаг 3. Проверяем регулярным выражением `RSurname` указанную фамилию пользователя. Она должна состоять из букв латинского и кириллического алфавита длиной от одного до девятнадцати символов включительно. Если текст подходит заданному выражению, то указываем булевой переменной `BSurname` значение `true`. Иначе выводим сообщение о

правильном написании фамилии и завершаем функцию. Шаг 19.

Шаг 4. Проверяем регулярным выражением `RLogin` указанный логин пользователя, то есть его почту. Она должна состоять из букв латинского алфавита, включать знак почты, иметь домен, разделенный точкой. Если текст подходит заданному выражению, то указываем булевой переменной `BLogin` значение `true`. Иначе выводим сообщение о правильном написании почты и завершаем функцию. Шаг 19.

Шаг 5. Проверяем регулярным выражением `RPassword` указанный пароль пользователя. Он может состоять из букв латинского алфавита, цифр и специальных символов, а также должен быть длиной от восьми символов. Если пароль подходит заданному выражению, то переходим на шаг 6. Иначе выводим сообщение о правильном написании имени и завершаем функцию. Шаг 19.

Шаг 6. Сравниваем пароль с подтверждением пароля. Если строки совпадают, указываем булевой переменной `BPassword` значение `true`. Иначе выводим сообщение о правильном написании имени и завершаем функцию, переходим на шаг 19.

Шаг 7. Если все переменные `BName`, `BSurname`, `BLogin`, `BPassword` равны true, переходим на шаг 8. Иначе шаг 19.

Шаг 8. Инициализация соединения с Firebase: вызов функции `Init`.

Шаг 9. Если переменная `cb`, содержащая ссылку на изображение, выбранное пользователем в качестве фотографии, не равна пустому значению `null`, то переходим к шагу 10, иначе присваиваем строковой переменной `imageSrc` пустую строку, переходим на шаг 11.

Шаг 10. Кодируем изображение в формат png и сохраняем в папку с фотографиями пользователей под именем и фамилией пользователя. Создаем в строковой переменной `imageSrc` относительный путь к изображению.

Шаг 11. Открываем конструкцию `try`, которая в случае ошибки доступа к базе данных корректно выведет сообщение ошибки и не прервет выполнение программы.

Шаг 12. Проверяем, есть ли такой пользователь в локальной базе данных. Вызов функции `GetUser(login)` в классе `Database`, которая формирует запрос к таблице `Users`, где `Id_user` равен `login`. Возвращает полученные данные из локальной базы данных.

Шаг 13. Проверяем, есть ли такой пользователь в системе сервера. Вызов функции `CheckUser(login)` в классе `Firebase`, которая делает snapshot документа с именем `login` в коллекции `Users`. Возвращает булевое значение, равное наличию документа.

Шаг 14. Если пользователь есть в локальной базе данных или на сервере, то присваиваем `load` значение `false`, выбрасываем исключение. Переходим к шагу 17.

Шаг 15. Присваиваем классу `Profile` данные пользователя:

– Profile.Id_user – созданный новый GUID;
– Profile.name – имя пользователя;
– Profile.surname - фамилия;
– Profile.login – логин (почта);
– Profile.password - пароль;
– Profile.createDate – сегодняшняя дата;
– Profile.imageSrc – путь к фотографии imageSrc;
– Profile.subscriptionDate – для удобства в дальнейшем вместо нулевого значения используется минимальная дата и время DateTime.MinValue;
– Profile.gbs – количество имеющихся Гб места, равно нулю;
– Profile.GBDate – аналогично с датой подписки значение DateTime.MinValue.

Шаг 16. Добавление нового пользователя в локальную базу данных функцией Database.InsertUser(), а также в серверную функцией Firebase.AddUser(). Эти функции используют статический класс Profile для заполнения полей для вставки пользователя в базу данных.

Шаг 17. Если произошла ошибка, оператор catch поймал исключение. Если значение load равно false, то выводится сообщение, что пользователь с таким логином уже существует. Иначе выводится сообщение о том, что нельзя зарегистрировать пользователя, потому что отсутствует подключение к базе данных и интернету. Значению load присвоено false.

Шаг 18. Если load равно true, то выполняется загрузка приложения. Иначе шаг 19.

Шаг 19. Завершение алгоритма.

Алгоритм входа в аккаунт выглядит очень похоже, но гораздо проще, но для понятия работы системы приведем его тоже. Алгоритм по шагам функции Log_In_Click:

Шаг 1. После нажатия на кнопку входа устанавливаем локальную булевую переменную load в значение false. Это значение изменится на противоположное в любой части алгоритма, когда будет сопоставлен пароль пользователя, чтобы не дать программе загрузиться при неправильном создании аккаунта. Также устанавливаем булевые переменные BLogin и BPassword в значение false, чтобы потом при проверке изменить на true.

Шаг 2. Проверяем регулярным выражением RLogin указанный логин пользователя. Если текст подходит заданному выражению, то указываем булевой переменной BLogin значение true. Иначе выводим сообщение о правильном написании логина и завершаем функцию. Шаг 17.

Шаг 3. Проверяем регулярным выражением RPassword пароль пользователя. Если текст подходит заданному выражению, то указываем булевой переменной BPassword значение true. Иначе выводим сообщение

о правильном написании пароля и завершаем функцию. Шаг 17.

Шаг 4. Если все переменные `BLogin` и `BPassword` равны `true`, переходим на шаг 5. Иначе шаг 17.

Шаг 5. Инициализация соединения с Firebase: вызов функции `Init`.

Шаг 6. Открываем конструкцию `try`, которая в случае ошибки доступа к базе данных корректно выведет сообщение ошибки и не прервет выполнение программы.

Шаг 7. Проверяем, есть ли такой пользователь в локальной базе данных. Вызов функции `GetUser(login)` в классе `Database`, которая формирует запрос к таблице `Users`, где `Id_user` равен `login`. Возвращает полученные данные из локальной базы данных.

Шаг 8. Если такого пользователя нет, то шаг 10.

Шаг 9. Проверяется пароль. Если он совпал, то выполняется загрузка информации о пользователе, `load` присваивается `true`, совершается переход на шаг 16, иначе вывод сообщения о неправильном пароле, переход на шаг 17.

Шаг 10. Проверка наличия пользователя на сервере. Вызов функции `CheckUser(login)` в классе `Firebase`. Если пользователя нет, то шаг 17.

Шаг 11. Получение информации пользователя с сервера функцией `Firebase.GetUser(login)`.

Шаг 12. Если шаг 11 успешно выполнен, пользователь заносится в локальную базу данных функцией `Database.InsertUser()`, переход на шаг 14. Иначе шаг 13.

Шаг 13. Выводится сообщение, что такого пользователя нет. Переход на шаг 17.

Шаг 14. Проверяется пароль. Если он совпал, то `load` присваивается `true`, совершается переход на шаг 16, иначе вывод сообщения о неправильном пароле, переход на шаг 17.

Шаг 15. Если в программе произошло исключение, то оно поймано в операторе `catch`. Вывод сообщения о невозможности загрузки пользователя, потому что отсутствует подключение к базе данных и интернету. Переменной `load` присваивается `false`.

Шаг 16. Если `load` равно `true`, выполняется загрузка приложения.

Шаг 17. Завершение алгоритма.

4.2 Создание плейлиста

Создание плейлиста выполняется в классе `CreatePlaylist`. Он представляет собой отдельное окно, где можно выбрать изображение для плейлиста и папку функцией `Add_Click`, на основе которой будет создан плейлист.

Алгоритм полного создания плейлиста по шагам:

Шаг 1. Если папка с аудио не выбрана, то шаг 16.

Шаг 2. В функции `Add_Click` Открытие диалогового окна для выбора папки, из которой будет составлен плейлист. Выбор папки.

Шаг 3. Получение массива строк `allfiles` с путями к файлам аудиозаписей, которые будут внесены в базу данных как отдельные композиции с уникальными путями.

Шаг 4. Получение массива кортежей идентификаторов и путей песен из базы данных функцией `Database.GetSongsPath()`. Если функция выполнена неправильно, то выводится сообщение об отсутствии подключения к БД, переход на шаг 15.

Шаг 5. Задание целочисленной переменной `i` значение 0. Начало цикла по каждому пути композиции `file` из массива путей `allfiles`.

Шаг 6. Создание экземпляра класса Song. Вызов функции `Song.addProperties(file)`.

Шаг 7. В функции `addProperties` с помощью библиотеки `tfile` получаем метаданные о композиции и заполняем соответствующие поля. Если в метаданных нет названия, то названием становится часть названия файла до тире. Если нет исполнителя, то вторая часть после первого тире.

Шаг 8. Если путь композиции уже есть в полученном из базы данных, то идентификатор композиции равен идентификатору существующей с таким же путем. Переход на шаг 10. Иначе шаг 9.

Шаг 9. Создание нового идентификатора композиции. Добавление композиции в последовательность `songs_all`, которая потом будет добавлена в базы данных.

Шаг 10. Присваивание порядковому номеру `n_sequence` значения `i`.

Шаг 11. Добавление песни в последовательность `songs`, которая будет присвоена плейлисту.

Шаг 12. Увеличение счетчика `i` на 1.

Шаг 13. Конец цикла.

Шаг 14. Конец функции `Add_Click`.

Шаг 15. Начало функции `Create_Click`.

Шаг 16. Проверка имени плейлиста регулярным выражением на пустоту. Если имя пустое, то новое имя будет составлено из слова «Playlist» и даты создания плейлиста. Иначе шаг 19.

Шаг 17. Проверка имени плейлиста регулярным выражением на соответствие правильному имени. Если имя соответствует, то новое имя будет указанным пользователем. Переход на шаг 19. Иначе шаг 18.

Шаг 18. Новое имя будет составлено из слова «Playlist» и даты создания плейлиста.

Шаг 19. Целочисленной переменной `i` присвоено значение 1. Начало цикла соответствия имени на уникальность.

Шаг 20. Если имя уже существует, то выводится сообщение о существовании этого имени, к имени добавляется `i`, значение `i` увеличивается

на 1. Повторение цикла, шаг 20. Иначе прерывание, шаг 21.

Шаг 21. Если переменная cb, содержащая ссылку на изображение, выбранное пользователем в качестве фотографии плейлиста, не равна пустому значению null, то кодируем изображение в формат png и сохраняем в папку с фотографиями плейлистов под именем плейлиста и пользователя. Создаем в строковой переменной imageSrc относительный путь к изображению. Иначе присваиваем строковой переменной imageSrc пустую строку.

Шаг 22. Создание экземпляра класса UserPlaylist. Заполнение его полей:

- playlist.Id_playlist – новый идентификатор GUID;
- playlist.Id_user – идентификатор пользователя из класса Profile;
- playlist.name – имя плейлиста;
- playlist.lastUpdate - дата и время на момент создания;
- playlist.lastSync - из-за невозможности использования нулевого значения используется DateTime.MinValue;
- playlist.imageSrc – путь к изображению плейлиста;
- playlist.songs – список песен, полученный на шаге 11.

Шаг 23. Вызов функции Database.InsertSongs(songs_all) для вставления новых композиций в базу данных песен, пути которых уникальны. Если функция выполнена успешно, то шаг 24. Иначе шаг 29.

Шаг 24. Вызов функции Firebase.AddSongs(songs_all) для добавления уникальных песен на сервер.

Шаг 25. Вызов функции Database.InsertPlaylist(playlist) для добавления плейлиста в базу данных. Если функция выполнена успешно, то шаг 26, иначе шаг 29.

Шаг 26. Вызов функции Firebase.AddPlaylist(playlist) для добавления плейлиста на сервер. Если функция выполнена успешно, то шаг 27, иначе шаг 29.

Шаг 27. Изменение поля playlist.lastSync на текущую дату и время.

Шаг 28. Вызов функции Database.UpdatePlaylist(playlist) для обновления плейлиста. Обновляется только поле описания. Переход на шаг 30.

Шаг 28. Вызов функции Firebase.AddPlaylist(playlist) для обновления плейлиста.

Шаг 29. Вывод сообщения о невозможности создать плейлист из-за отсутствия подключения к базе данных.

Шаг 30. Конец алгоритма.

4.3 Синхронизация плейлистов

Синхронизация плейлистов выполняется после нажатия кнопки «Synchronize» на странице «My playlists». Это принудительная синхронизация, которая не выполняется сама по себе, потому что у пользователя могут быть другие устройства, содержащие разные плейлисты. Плейлисты автоматически обновляются при наличии подключения к интернету после создания плейлиста и его изменения. Кроме синхронизации плейлистов выполняется синхронизация композиций, точнее ссылок на них. Потому что в локальной базе данных и удаленной один и тот же пул композиций для удобства оперирования ими.

Алгоритм по шагам синхронизации плейлистов:

Шаг 1. Объявление переменных:

- `playlists_db` – список плейлистов из базы данных;
- `playlists_fb` - список плейлистов с сервера;
- `songs_db` – список песен из базы данных;
- `songs_fb` – список песен с сервера;
- `ids_db`, `ids_fb` – списки идентификаторов плейлистов для удобства поиска.

Шаг 2. Вызов функции `Database.GetAllSongs()` для получения всех песен из базы данных `songs_db`. Если функция вернула `false`, шаг 34.

Шаг 3. Вызов функции `Firebase.GetSongs(null, true)` для получения всех песен из базы данных `songs_fb`. Если функция вернула `false`, шаг 34.

Шаг 4. Начало цикла добавления композиций на сервер. Для каждой композиции `song` из `songs_db`.

Шаг 5. Если `song` не содержится в `songs_fb`, то вызов функции добавления `Firebase.AddSong(song)`.

Шаг 6. Конец цикла добавления композиций на сервер.

Шаг 7. Начало цикла добавления композиций в базу данных. Для каждой композиции `song` из `songs_fb`.

Шаг 8. Если `song` не содержится в `songs_db`, то вызов функции добавления `Database.InsertSong(song)`.

Шаг 9. Конец цикла добавления композиций в базу данных.

Шаг 10. Вызов функции `Database.GetPlaylists(deleted : true)` для получения всех плейлистов из базы данных `playlists_db`. Если функция вернула `false`, шаг 34.

Шаг 11. Вызов функции `Firebase.GetPlaylists()` для получения всех плейлистов из базы данных `playlists_fb`. Если функция вернула `false`, шаг 34.

Шаг 12. Начало цикла добавления плейлистов на сервер. Для каждого плейлиста `playlist` из `playlists_db`.

Шаг 13. Если имя плейлиста равно `UserPlaylist.NULL_NAME`, то шаг 14, иначе шаг 16.

Шаг 14. Вызов функции удаления плейлиста на сервере `Firebase.DeletePlaylist`. Если функция вернула `true` значение, то шаг 15. Иначе шаг 13.

Шаг 15. Вызов функции окончательного удаления плейлиста в базе данных `Database.DeletePlaylist`. Шаг 13.

Шаг 16. Если список плейлистов на сервере `ids_fb` не содержит плейлиста `playlist` или дата обновления плейлиста больше даты последней синхронизации, то шаг 17. Иначе шаг 13.

Шаг 17. Обновление поля плейлиста lastSync на текущее.

Шаг 18. Вызов функции Firebase.AddPlaylist(playlist) для добавления или обновления плейлиста на сервере. Если функция вернула true, шаг 19, иначе шаг 13.

Шаг 19. Вызов функции Database.UpdatePlaylist(playlist) для обновления плейлиста с композициями в базе данных. Шаг 13.

Шаг 20. Конец цикла добавления плейлистов на сервер.

Шаг 21. Начало цикла добавления плейлистов базу данных. Для каждого плейлиста `playlist` из `playlists_fb`.

Шаг 22. Если имя плейлиста равно `UserPlaylist.NULL_NAME`, то шаг 23, иначе шаг 25.

Шаг 23. Вызов функции окончательного удаления плейлиста в базе данных `Database.DeletePlaylist`. Если функция вернула `true` значение, то шаг 24. Иначе шаг 22.

Шаг 24. Вызов функции удаления плейлиста на сервере `Firebase.DeletePlaylist`. Шаг 22.

Шаг 25. Если список плейлистов в базу данных `ids_db` не содержит плейлиста `playlist`, то шаг 26. Иначе шаг 29.

Шаг 26. Обновление поля плейлиста `lastSync` на текущее.

Шаг 27. Вызов функции `Database.InsertPlaylist(playlist)` для добавления плейлиста в базу данных с композициями. Если функция вернула `true`, шаг 28, иначе шаг 22.

Шаг 28. Вызов функции `Firebase.AddPlaylist(playlist)` для обновления плейлиста на сервере. Шаг 22.

Шаг 29. Если последняя синхронизация плейлиста на сервере была позже, чем в базу данных, то шаг 30. Иначе шаг 22.

Шаг 30. Обновление поля плейлиста `lastSync` на текущее.

Шаг 31. Вызов функции `Database. UpdatePlaylist(playlist)` для обновления описания плейлиста в базе данных. Если функция вернула `true`, шаг 32, иначе шаг 22.

Шаг 32. Вызов функции `Firebase.AddPlaylist(playlist)` для обновления плейлиста на сервере. Шаг 22.

Шаг 33. Конец цикла добавления плейлистов на сервер.

Шаг 34. Конец алгоритма.

4.4 Удаление плейлиста

Удаление плейлиста происходит после клика минуса в углу плейлиста на странице всех плейлистов. Так как необходима синхронизация с сервером, плейлист не может быть удален сразу, поэтому было выбрано решение удалить все данные, но оставить описание с «зануленным» именем. Это строковая константа из букв и цифр, которую пользователь не сможет повторить, чтобы случайно не удалить плейлист.

Удаление плейлиста по шагам:

Шаг 1. Вызов функции `Database.DeletePlaylist(playlist_)` с передачей полного плейлиста. Если функция вернула значение `false`, то переход к шагу 8.

Шаг 2. В функции `Database.DeletePlaylist` выполняется запрос к базе данных, в которой в таблице `Playlist_songs` удаляются все записи, где `Id_playlist` соответствует переданному функции значению.

Шаг 3. Снова выполняется запрос к базе данных. Плейлист обновляется в таблице `Playlists`: его именем становится константа `UserPlaylist.NULL_NAME`. Возврат из функции.

Шаг 4. Вызов функции `Firebase.DeletePlaylist` с передачей идентификатора плейлиста. Если функция вернула значение `false`, то переход к шагу 8.

Шаг 5. В функции `Firebase.DeletePlaylist` выполнятся обновление плейлиста: его именем становится константа `UserPlaylist.NULL_NAME`.

Шаг 6. Вызов функции `Database.DeletePlaylist(playlist_, false)` с указанием полностью удалить плейлист из базы данных.

Шаг 7. В функции `Database.DeletePlaylist` выполняется запрос к базе данных. Удаляется из таблицы `Playlists` указанный плейлист.

Шаг 8. Конец алгоритма.

4.5 Совершение подписки

Так как совершение платежа только эмулируется, совершение подписки будет заключаться в изменении статуса пользователя и внесении в его историю платежа.

Алгоритм совершения подписки:

Шаг 1. Вызов соответствующей функции: подписка на сервис на определенное время или подписка на дополнительную память с определенным количеством гигабайт. Если продлевается подписка на место по меньшему

тарифу, то нельзя это сделать, если до конца срока первой более недели.

Шаг 2. Вызов функции добавления платежа `Firebase.UserPayment`. Если функция вернула `false`, значит платеж не состоялся, переход к шагу 9.

Шаг 3. В функции `Firebase.UserPayment` выполняется обновление информации пользователя и создание нового платежа.

Шаг 4. Обновление соответствующего поля класса `Profile`: `subscriptionDate` или `GBDate` и `gbs`.

Шаг 5. Вызов функции `Database.UpdateUser()`. Эта функция занесет обновленную информацию по пользователю в базу данных.

Шаг 6. Вызов функции `UpdateSubscription()` для корректного отображения статуса пользователя на экране.

Шаг 7. В функции `UpdateSubscription` проверяется состояние подписки на стриминговый сервис. Если подписка есть, появляется текст с датой окончания и вкладка со стримингом становится видимой, если нет – надпись «unsubscribed».

Шаг 8. В функции `UpdateSubscription` проверяется состояние подписки на память. Если подписка есть, отображается число гигабайт или надпись «UNLIMITED GB», вкладка управления памятью становится видимой. Если нет – в подписке отображается «0 GB».

Шаг 9. Конец алгоритма.

4.6 Проигрывание плейлиста

Проигрывание плейлиста осуществляется классом `MusicStream`, данные для него хранятся в глобальном классе-переменной `Vars`. Этот алгоритм состоит из вызова множества функций из разных классов, для удобства восприятия некоторые приводиться не будут.

Алгоритм воспроизведения по шагам:

Шаг 1. При переходе в плейлист, альбом или в другие страницы со списками композиций они последовательно заносятся в класс `Vars` в лист current, при необходимости сортируются.

Шаг 2. В экземпляре класса `Composition`, который отображает саму композицию, при нажатии на «Play» в класс `Vars` вносятся:

- идентификатор плейлиста или альбома, если такой есть;
- поле `StreamTracklist` копирует поле current;
- `CurrentTrackNumber` - порядковый номер композиции в плейлисте.

Шаг 3. Вызывается функция проигрывания текущей композиции `MusicStream.PlayPlayer`.

Шаг 4. Вызывается функция включения проигрывателя `MusicStream.Play()`.

Шаг 5. Вызывается функция `MusicStream.StreamLineStart` для

отображения дорожки проигрывателя.

Шаг 6. Меняется иконка проигрывания на композиции на «Stop».

Шаг 7. При срабатывании событий переключения трека на следующий или предыдущий, в соответствующих функциях в предыдущей композиции изменяется иконка проигрывателя на «Play».

Шаг 8. Изменяется `Vars.CurrentTrackNumber`.

Шаг 9. На новой композиции меняется иконка проигрывания на композиции на «Stop».

4.7 Поиск

Поиск можно осуществлять по композициям, исполнителям и альбомам. Если ни одна кнопка не зажата, то поиск идет по всем категориям, иначе по выбранным категориям.

Алгоритм поиска:

Шаг 1. Обращение к базе данных функцией `Database.GetAllSongs()` с целью получить все композиции. Если функция вернула `false`, то шаг 9.

Шаг 2. Если ни одна кнопка выбора поиска не зажата, переопределяем все как зажатые.

Шаг 3. Переводим строку поиска в нижний регистр.

Шаг 4. Начало цикла по каждой композиции `s` из списка `songs`.

Шаг 5. Булевую переменную `match` ставим в `false`. Если совпадет критерий поиска, то она изменится.

Шаг 6. Если критерий выбора – композиция и название композиции в нижнем регистре содержит строку поиска, то `match` становится `true`.

Шаг 7. Если критерий выбора – исполнитель и исполнитель композиции в нижнем регистре содержит строку поиска, то `match` становится `true`.

Шаг 8. Если критерий выбора – альбом и альбом композиции в нижнем регистре содержит строку поиска, то `match` становится `true`.

Шаг 8. Если `match` равно `true`, то создаем композицию с указанием, что она в списке поиска, добавляем в список `Vars.search` и на экран.

Шаг 9. Конец алгоритма.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

При разработке любого приложения важным этапом является выполнение проверки корректности функционирования системы, то есть проведения ее тестирования. При этом существуют различные виды тестирования, которые зависят от области приложения, которая подлежит проверке.

Тестирование программного продукта классифицируется по различным критериям:

– по запуску кода на исполнение;
– по способу доступа к коду;
– по степени автоматизации.

По запуску кода на исполнение выделяют следующие виды тестирования:

– статическое тестирование – проверка кода без его запуска на исполнение;
– динамическое тестирование – проверка функциональности программного продукта с учетом запуска кода на исполнение, то есть тестирование на этапе выполнения программы.

По способу доступа к коду выделяют следующие виды тестирования:

– метод белого ящика – метод, при котором тестировщик может иметь доступ как к внешней, так и ко внутренней структуре программного продукта, включая код. Тестирование на основе дизайна представляет собой пример тестирования методом белого ящика;
– метод черного ящика – метод, при котором у тестировщика нет доступа ко внутренней структуре программного продукта и соответственно к коду, либо отсутствует понимание кода;
– метод серого ящика – метод, который представляет собой комбинацию методов белого и черного ящиков. Данный метод применяется в случае, когда у тестировщика есть понимание только части структуры и кода программного продукта.

По степени автоматизации существуют следующие виды тестирования:

– автоматизированное тестирование – тестирование, при котором наборы, выбранные из тестового покрытия, выполняются без вмешательства человека с использованием средств автоматизации. Данный подход позволяет значительно увеличить скорость выполнения тестов, но сама разработка тестовых наборов возлагается на человека.
– ручное тестирование – тестирование, при котором наборы, выбранные из тестового покрытия, выполняются тестировщиком вручную без использования автоматизации. Следовательно, повторное выполнение одного и того же теста требует усилий человека. Ручное тестирование, в отличие от автоматизированного, не требует детального планирования и оценки рисков, не требует высоких затрат на сопровождение кода.

Исходя из вышеперечисленного, тестирование будет производиться сначала статическое, потом динамическое, по методу белого ящика разработчиком и черного пользователем, ручное. Тестирование было решено разбить на несколько модулей по соответствующим блокам:

- аутентификация;
- радио;
- подписка;
- работа с плейлистами;
- проигрывание музыки
- приложение.

5.1 Аутентификация

Тестирование данного модуля выражается в разном заполнении полей, поведении приложения в случае отсутствия доступа к базе данных или серверу. Данные тестирования приведены в таблице 5.1.

Таблица 5.1 –Тестирование аутентификации

Номер теста	Описание	Данные и условия	Ожидаемый результат	Статус
1	2	3	4	5
Т1	Некорректный ввод имени пользователя при регистрации	1. Пустая строка. 2. Строка с цифрами. 3. Строка с символами. 4. Строка с 20 знаками	Сообщение об ошибке	ОК
Т2	Некорректный ввод фамилии пользователя при регистрации	1. Пустая строка. 2. Строка с цифрами. 3. Строка с символами. 4. Строка с 20 знаками	Сообщение об ошибке	ОК
Т3	Некорректный ввод логина пользователя при регистрации	1. Пустая строка. 2. Строка без «@». 3. Строка без домена. 4. Строка с символами. 5. Строка с пробелом. 6. Строка с 60 знаками	Сообщение об ошибке	ОК
Т4	Некорректный ввод пароля пользователя при регистрации	1. Пустая строка. 2. Строка до 8 знаков. 3. Строка с пробелом. 4. Строка без цифры. 5. Строка без специального символа. 6. Строка со знаками кириллического алфавита. 7. Ввод отличающегося подтверждающего пароля	Сообщение об ошибке	ОК

Продолжение таблицы 5.1

1	2	3	4	5
Т5	Некорректный выбор фотографии при регистрации	1. Выбор файла неразрешенного формата. 2. Выбор изображения менее 600x600 пикселей	Сообщение об ошибке	ОК
Т6	Корректный ввод данных	Корректный ввод данных. Наличие и отсутствие фотографии.	Разрешение регистрации	ОК
Т7	Соединение с базами данных при регистрации	Есть соединение с локальной и серверной	Регистрация	ОК
		Есть соединение только с локальной	Регистрация	
		Есть соединение только с серверной	Регистрация	
		Нет соединения ни с одной базой данных	Сообщение об ошибке	
Т8	Некорректный ввод логина пользователя при входе	1. Пустая строка. 2. Строка без «@». 3. Строка без адреса перед «@». 4. Строка без домена. 5. Строка с символами. 6. Строка с пробелом. 7. Строка с 60 знаками	Сообщение об ошибке	ОК
Т9	Некорректный ввод пароля пользователя при входе	1. Пустая строка. 2. Строка до 8 знаков. 3. Строка с пробелом. 4. Строка со знаками кириллического алфавита. 5. Ввод отличающегося подтверждающего пароля	Сообщение об ошибке	ОК
Т10	Соединение с базами данных при входе	Есть соединение хотя бы с одной базой данных	Вход	ОК
		Нет соединения ни с одной базой данных	Сообщение об ошибке	
Т11	Синхронизация пользователя	Отсутствие пользователя в серверной базе данных	Добавление пользователя в локальную БД	ОК
		Отсутствие пользователя в локальной базе данных	Добавление пользователя в серверную БД	
Т12	Заполнение профиля	Чтение из локальной базы данных	Заполнение профиля	ОК
		Чтение из серверной базы данных	Заполнение профиля	

5.2 Радио

Тестирование модуля радио проводится по наличию подключения к интернету и базам данных, корректному отображению истории и актуальных композиций. Данные тестирования приведены в таблице 5.2.

Таблица 5.2 –Тестирование радио

Номер теста	Описание	Данные и условия	Ожидаемый результат	Статус
Т1	Отображение композиций	–	Отображение композиций	ОК
Т2	Соединение с базами данных при загрузке	Есть соединение с локальной и серверной	Загрузка	ОК
		Есть соединение только с локальной	Загрузка	
		Есть соединение только с серверной	Загрузка	
		Нет соединения ни с одной базой данных	Сообщение об ошибке	
Т3	Работа радио	Отсутствие подключения к интернету	Сообщение об ошибке	ОК

5.3 Подписка

Тестирование модуля совершения подписки проводится по наличию подключения к базам данных, корректному отображению актуальной информации о пользователе и доступе к подпискам, совершению подписки при разных условиях. Данные тестирования приведены в таблице 5.3.

Таблица 5.3 –Тестирование подписки

Номер теста	Описание	Данные и условия	Ожидаемый результат	Статус
1	2	3	4	5
Т1	Соединение с сервером	Отсутствует	Сообщение об ошибке	ОК
		Есть	Создание записи и обновление пользователя	
Т2	Соединение с локальной базой данных	Отсутствует	Нет изменений	ОК
		Есть	Обновление пользователя	
Т3	Отображение доступа к стриминговому сервису	Есть подписка	Отображается	ОК
		Нет подписки	Не отображается	

Продолжение таблицы 5.3

1	2	3	4	5
Т4	Отображение доступа к дополнительной памяти	Есть подписка	Отображается	ОК
		Нет подписки	Не отображается	
Т5	Отображение актуальной информации о подписках	Нет подписки на стриминговый сервис	Текст «unsubscribed»	ОК
		Есть подписка на стриминговый сервис	Отображение даты конца подписки	
		Нет подписки на дополнительную память	Текст «0 Gb»	
		Есть подписка на дополнительную память	Отображение количества места и даты конца подписки	
		Есть подписка на неограниченную дополнительную память	Отображение «unlimited» и даты конца подписки	
Т6	Продление подписки	Продление подписки на стриминговый сервис	Увеличение даты окончания на выбранное время	ОК
		Продление подписки на дополнительную память на не меньшее место	Увеличение даты окончания на год, место – новое количество	
		Продление подписки на дополнительную память на меньшее место позднее недели до конца	Увеличение даты окончания на год, место – новое количество	
		Продление подписки на дополнительную память на меньшее место ранее недели до конца	Сообщение об отказе	

5.4 Работа с плейлистами

Тестирование модуля работы с плейлистами самое объемное. Оно должно включать в себя все возможные случаи неполадок и некорректных действий, например, подключение к базам данных, неправильное заполнение имени, чтение композиций и их метаданных, формирование очередности. Переходы между страницами будут описаны в разделе тестирования приложения.

Все возможные случаи были протестированы вручную методами белого и черного ящика разработчиком и сторонним пользователем. Данные тестирования приведены в таблице 5.4.

Таблица 5.4 – Тестирование работы с плейлистами

Номер теста	Описание	Данные и условия	Ожидаемый результат	Статус
1	2	3	4	5
Т1	Выбор картинки для плейлиста	Выбор файла неразрешенного формата	Сообщение об ошибке	ОК
		Выбор изображения менее 600x600 пикселей	Сообщение об ошибке	
		Выбор правильного формата и разрешения	Сохранение картинки	
Т2	Задание имени плейлисту	Ввод пустой строки	Генерация имени с датой	ОК
		Ввод уникального имени	Сохранение имени	
		Ввод не уникального имени	Генерация имени с цифрой, сообщение о повторяющемся имени	
Т3	Создание плейлиста	С выбором исходной папки	Создание плейлиста, получение очереди композиций	ОК
		Без выбора исходной папки	Создание плейлиста	
Т4	Открытие плейлиста	Нажатие по картинке	Открытие плейлиста	ОК
Т5	Добавление композиций	Выбор одной и нескольких композиций	Обновление плейлиста	ОК
Т6	Удаление композиции	Клик по минусу на композиции	Обновление плейлиста	ОК
Т7	Удаление плейлиста	Клик по минусу на картинке плейлиста	Удаление плейлиста	ОК
Т8	Синхронизация плейлистов	Клин по кнопке синхронизации	Синхронизация	ОК
Т9	Создание новой песни	Передача пути к файлу	Создание новой песни	ОК
Т10	Подключение к базам данных при чтении плейлистов	Есть соединение с локальной и серверной	Загрузка	ОК
		Есть соединение только с локальной	Загрузка	
		Есть соединение только с серверной	Загрузка	
		Нет соединения ни с одной базой данных	Сообщение об ошибке	

Продолжение таблицы 5.4

1	2	3	4	5
Т11	Подключение к базам данных при записи плейлистов	Есть соединение с локальной и серверной	Запись в оба	ОК
		Есть соединение только с локальной	Запись	
		Есть соединение только с серверной	Запись	
		Нет соединения ни с одной базой данных	Сообщение об ошибке	
Т12	Подключение к базам данных при синхронизации плейлистов	Есть соединение с локальной и серверной	Синхронизация	ОК
		Нет соединения с какой-то базой данных	Сообщение об ошибке	

5.5 Проигрывание музыки

Тестирование модуля проигрывания музыки должно включать в себя корректное воспроизведение последовательности, переключения и отображения проигрывания. Данные тестирования приведены в таблице 5.5.

Таблица 5.5 – Тестирование проигрывателя

Номер теста	Описание	Данные и условия	Ожидаемый результат	Статус
1	2	3	4	5
Т1	Остановка и проигрывание одной композиции по клику	Клик на проигрывание	Проигрывание, изменение цвета	ОК
		Клик на остановку	Остановка, изменение цвета	
Т2	Переключение и остановка	Клик вперед или назад на панели	Проигрывание следующего или предыдущего трека	ОК
		Вперед после конца трека	Вперед после конца трека	
		Клик на проигрывание и остановку	Проигрывание и остановка трека	
Т3	Ссылка на несуществующий файл	–	Проматывание	ОК
Т4	Конец плейлиста	–	Остановка проигрывания	ОК

5.6 Приложение

Данный раздел посвящен целому приложению как системе. Будут тестироваться переходы между страницами вне зависимости от правильности работы всех внутренних алгоритмов. Данные тестирования приведены в таблице 5.6.

Таблица 5.6 – Тестирование приложения

Номер теста	Описание	Данные и условия	Ожидаемый результат	Статус
T1	Отображение страницы аутентификации	Запуск приложения	Отображение страницы аутентификации	OK
T2	Аутентификация	Правильное заполнение всех полей	Открытие приложения	OK
		Неправильное заполнение одного поля	Сообщение об ошибке	
T3	Стартовая страница	Вход в приложение	Отображение страницы плейлистов	OK
T4	Отображение создания плейлиста	Клик по кнопке создания плейлиста	Отображение окна	
T5	Отображение плейлистов	Клик по кнопке плейлистов	Отображение плейлистов	OK
T6	Отображение плейлиста	Клик по плейлисту	Отображение плейлиста	OK
T7	Отображение подписок	Клик по кнопке подписок	Отображение подписок	OK
T8	Отображение всех радиостанций	Клик по кнопке радио	Отображение всех радиостанций	OK
T9	Отображение истории радиостанций	Клик по картинке радио	Отображение истории радиостанций	OK
T10	Отображение стримингового сервиса	Наличие подписки	Отображение стримингового сервиса	OK
T11	Отображение дополнительной памяти	Наличие подписки	Отображение дополнительной памяти	OK
T12	Отображение поиска	Клик по стрелке поиска	Отображение поиска	OK
T13	Отображение информации и статистики пользователя	Клик по фотографии пользователя	Отображение информации и статистики пользователя	OK
T14	Выход	Клик по кнопке выхода	Отображение страницы аутентификации	OK

5.7 Итог тестирования

Приложение тестировалось вручную методом белого и черного ящика. Метод белого ящика помогал отследить ошибки, незаметные при другом типе тестировании, потому что отслеживалось выполнение каждой функции и поток данных. Метод черного ящика, выполняемый сторонним пользователем, выявил неочевидные и неожиданные ошибки в работе системы.

Все части приложения покрыты тестами. Более мелкие тесты содержатся в больших тестах, как, например, синхронизация плейлистов. Это сделано для упрощения восприятия и тестирования функций в совокупности. Модули приложения протестированы сначала независимо друг от друга, чтобы выявить ошибки было проще.

Параллельно модульному тестированию выполнялось интеграционное тестирование, например, все работы с базой данных или сервером подразумевают тестирование взаимодействия этих модулей. Также модуль проигрывания музыки неразрывно связан с модулем поиска и плейлистами. Все протестированные интеграции не выявили ошибок.

Заключительной частью тестирования стало системное тестирование, описанное в пункте 5.6. Оно показало, что приложение в целом работает корректно и слаженно, переходы между страницами и окнами плавные, взаимодействие с сервером и базой данных корректное.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Процесс инсталляции

Для запуска разработанного в дипломном проекте приложения-плеера со стриминговым сервисом пользователю необходимо выполнить последовательность шагов, описанную в текущем разделе. Приложение состоит из одной части – клиентского приложения, разработанного на платформе .Net Framework. Серверная часть пользователю недоступна, он может с ней взаимодействовать опосредованно только через клиентское приложение.

Для развертывания приложения следует удостовериться, что на персональном компьютере пользователя установлена операционная система Windows 10. Далее перечислены компоненты, которые должны быть установлены в операционной системе:

– платформа .Net Framework версии 4.6.1;

– СУБД Microsoft SQL Server версии 2017 и выше.

Для запуска приложения используется программная платформа .NET Framework, основа которой представлена общеязыковой средой исполнения, подходящей для приложений, написанных на языке C#. Инсталлятор такой платформы версии 4.6.1 может быть найден на официальном сайте компании Microsoft.

Система управления базами данных Microsoft SQL Server необходима для развертывания базы данных, используемой в приложении, и дальнейшей работы с ней. Microsoft SQL Server версии 2016 и выше может быть скачан также с официального сайта компании Microsoft.

Далее необходимо скопировать файлы проекта с диска и запустить файл с расширением .exe, который является приложением.

6.2 Описание интерфейса приложения

Первое, что видит пользователь, это экран аутентификации с открытым окном для входа. Страница для входа изображена на рисунке 6.1. Если пользователь уже зарегистрирован в системе, ему необходимо ввести логин и пароль. Если они введены некорректно, приложение сообщит об этом всплывающим окном. Если возникли проблемы с аутентификацией, приложение напишет, в чем конкретно проблема. Когда пользователь введет правильные логин и пароль, система его распознает и приложение загрузится.

Если пользователь еще не зарегистрирован в системе, то после заполнения полей на странице аутентификации, проиллюстрированной на рисунке 6.2, новая учетная запись создастся. В случае возникновения проблем или неправильном заполнении полей приложение укажет на ошибку.

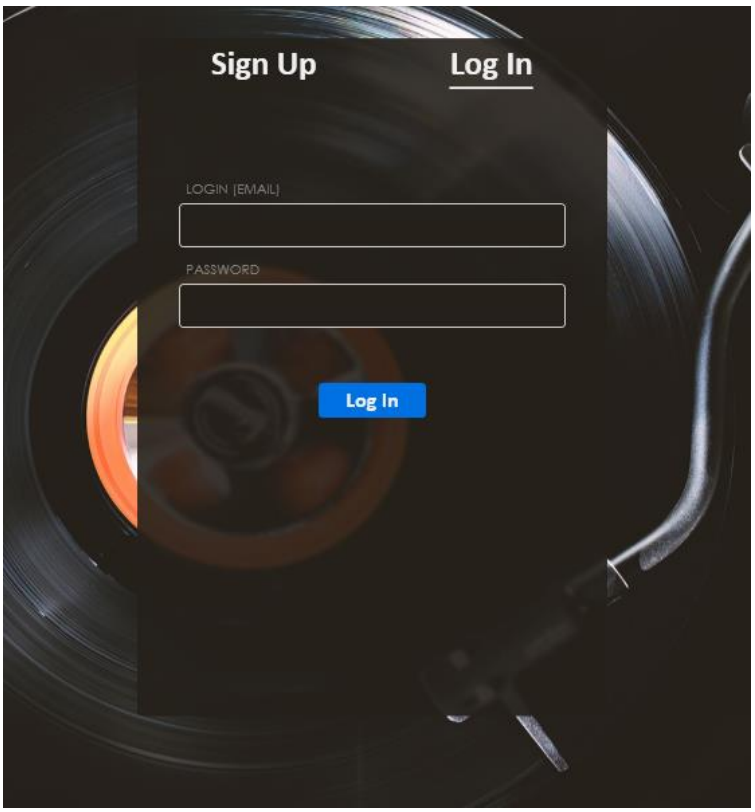

Рисунок 6.1 – Страница входа

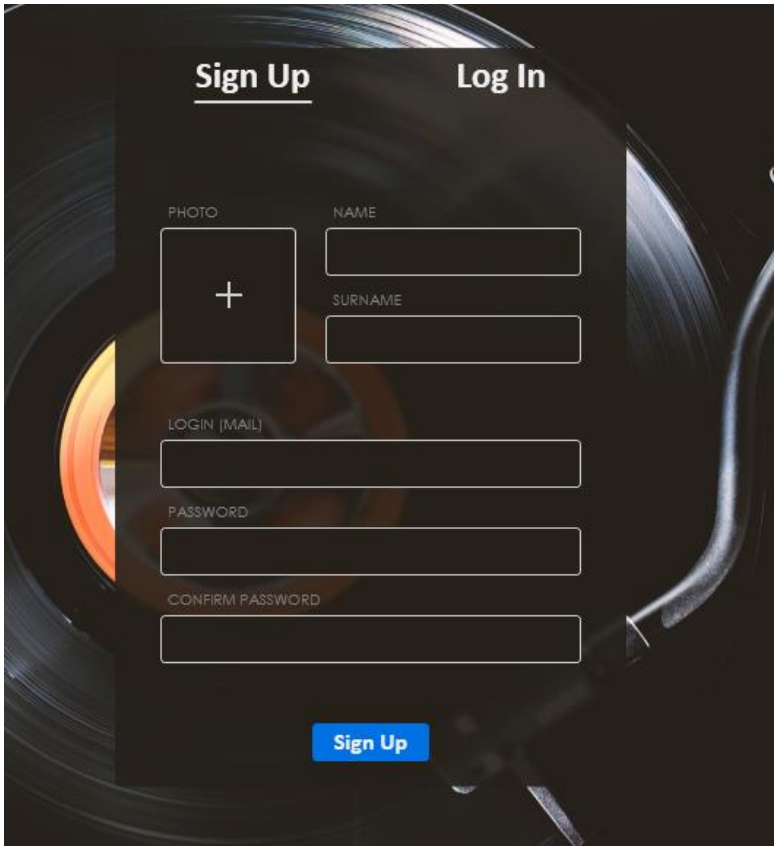

Рисунок 6.2 – Страница регистрации

После успешной регистрации приложение откроется на экране пользовательских плейлистов, как на рисунке 6.3.

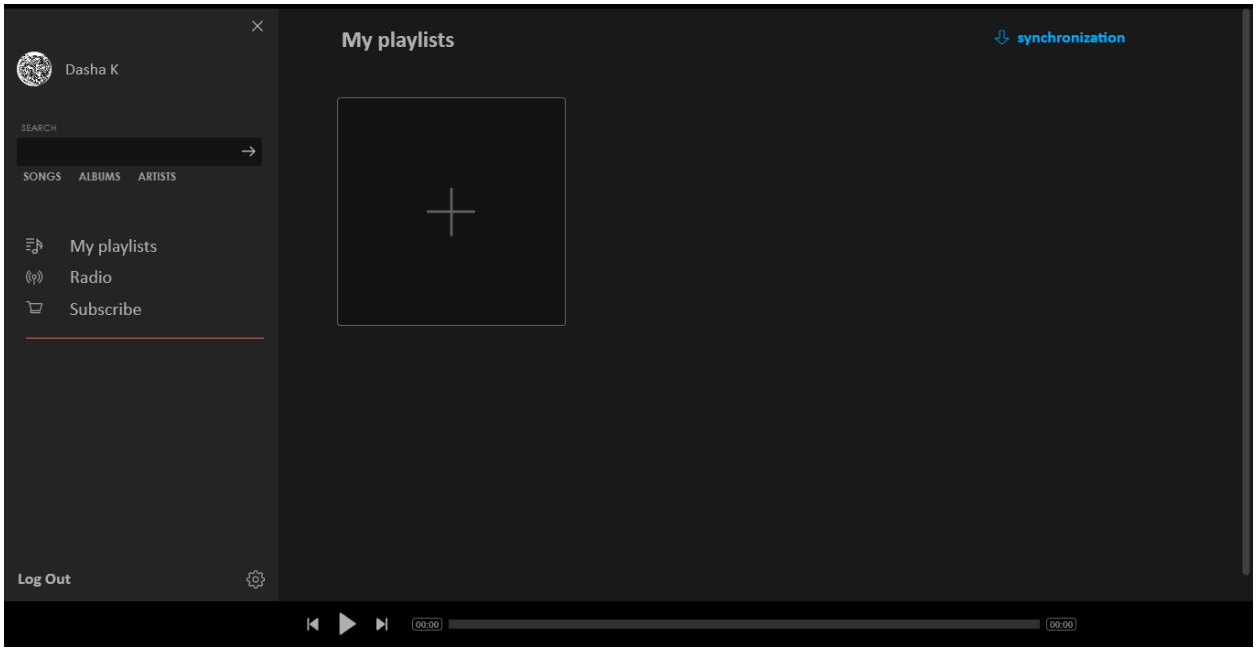

Рисунок 6.3 – Открытие приложения

Создать плейлист можно, кликая мышью на квадрат с плюсиком. Появится окно, как на рисунке 6.4.

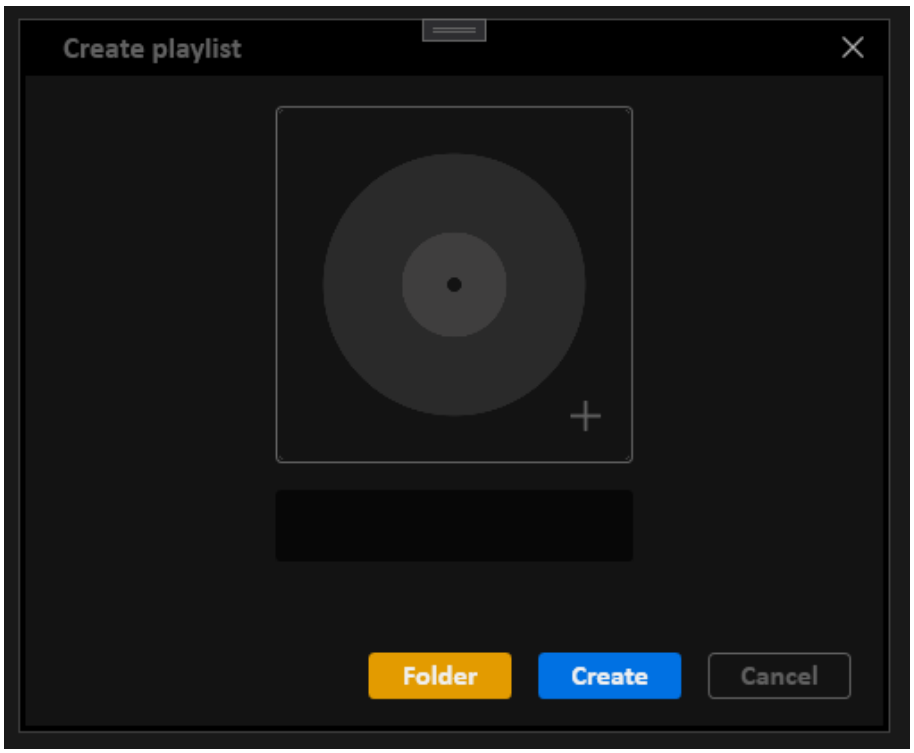

Рисунок 6.4 – Окно создания плейлиста

Приложение предложит выбрать изображение для плейлиста, чтобы быстрее отличать его от других. Можно указать, с какой папки плейлист будет брать аудиозаписи. Для этого можно кликнуть «Folder» и выбрать в диалоговом окне папку.

Необходимо задать имя плейлиста, а в случае отсутствия имени или имени, которое уже встречалось, появится предупреждение. Если имя отсутствует, плейлист будет назван «Playlist» и текущая дата. Пример на рисунке 6.5.

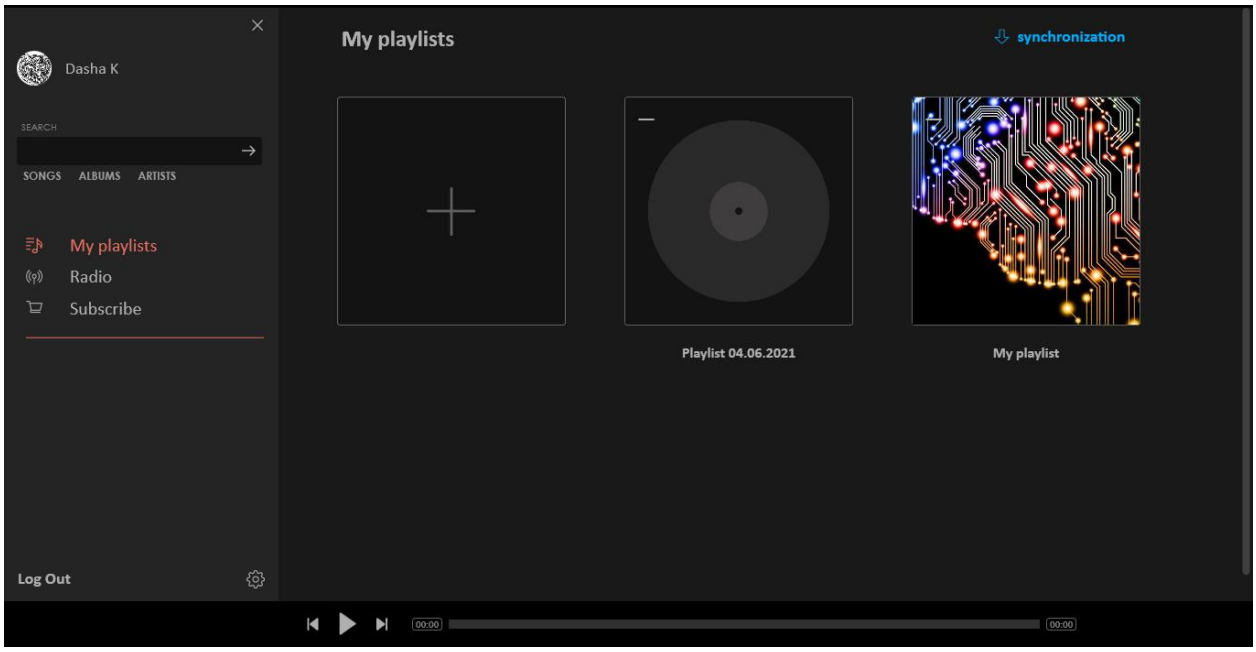

Рисунок 6.5 – Генерация имени плейлиста

Открыть плейлист можно, нажав на картинку плейлиста. Откроется или пустой плейлист, или уже заполненный, если перед этим была указана папка. Открытая страница имеет вид, представленный на рисунках 6.6 и 6.7.

Чтобы добавить в плейлист новые композиции, необходимо нажать на «+» в углу экрана. Далее в диалоговом окне выбрать уже не папку, а композиции и обновить плейлист, то есть зайти на него заново.

Композиции представлены как на рисунке 6.8. Чтобы удалить композицию, необходимо нажать «-» на композиции. Тогда композиция удалится из плейлиста, но не из памяти приложения. Чтобы добавить композицию в плейлист, необходимо нажать «+» на композиции, в выпадающем списке выбрать имя плейлиста и нажать на него. Композиция добавлена.

Чтобы удалить плейлист, необходимо нажать на «-» в левом верхнем углу картинки плейлиста, как на рисунке 6.9.

Значок и надпись синхронизации на главном экране плейлистов позволят синхронизировать плейлисты с сервером. Если сервер недоступен,

приложение выведет сообщение об этом. Обновите плейлисты после синхронизации.

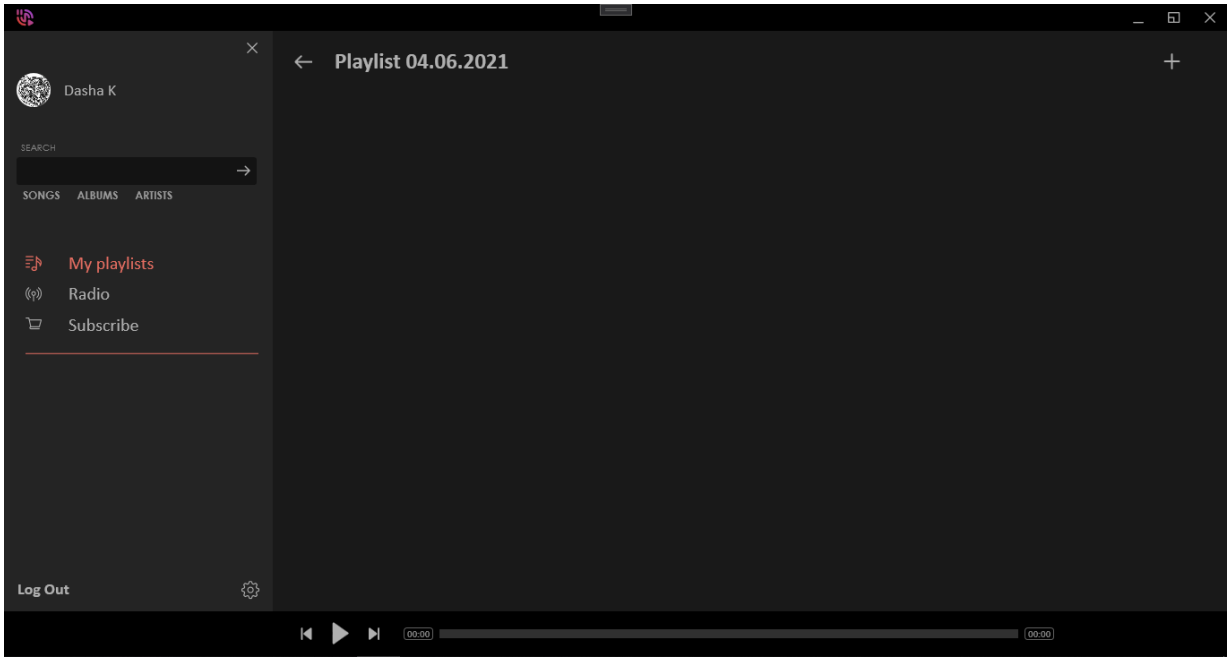

Рисунок 6.6 – Пример пустого плейлиста

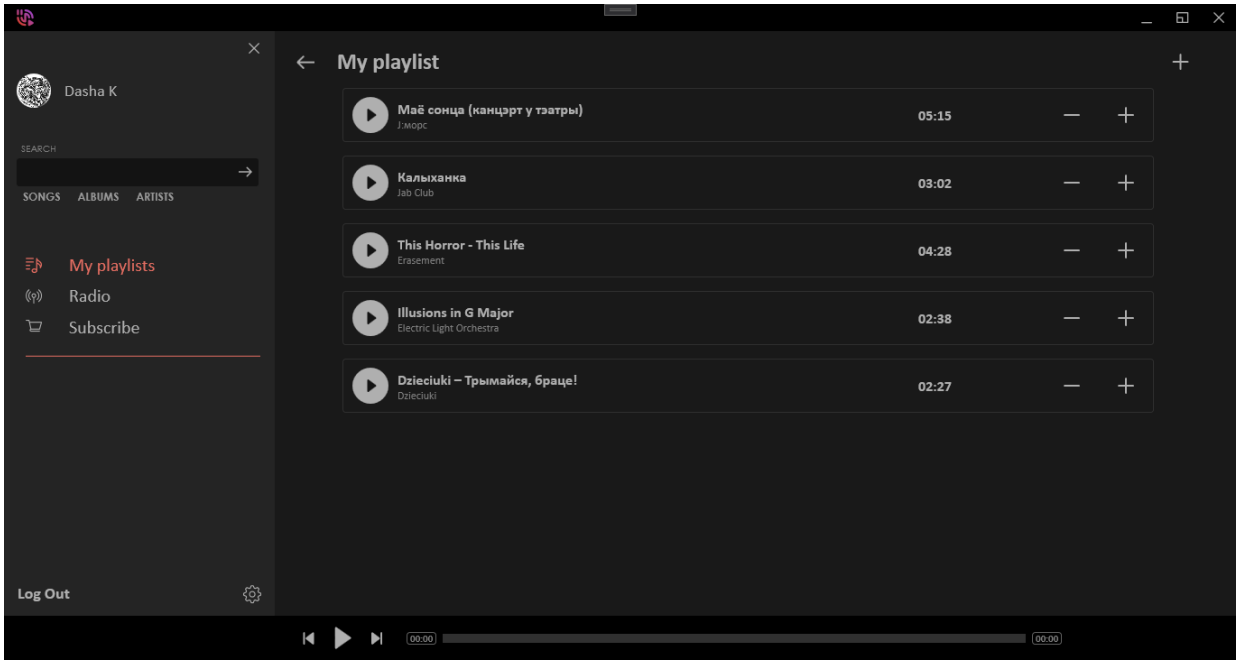

Рисунок 6.7 – Пример плейлиста с указанной папкой

Рисунок 6.8 – Представление композиции

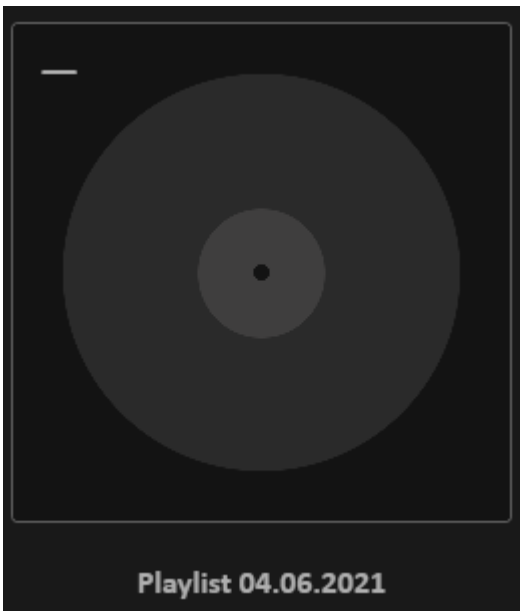

Рисунок 6.9 – Удаление плейлиста

Проигрывание композиции осуществляется нажатием на значок проигрывания. На панели снизу появится бегущая строка и название композиции, как на рисунке 6.10. На панели можно переключать композиции вперед, назад, останавливать и проматывать.

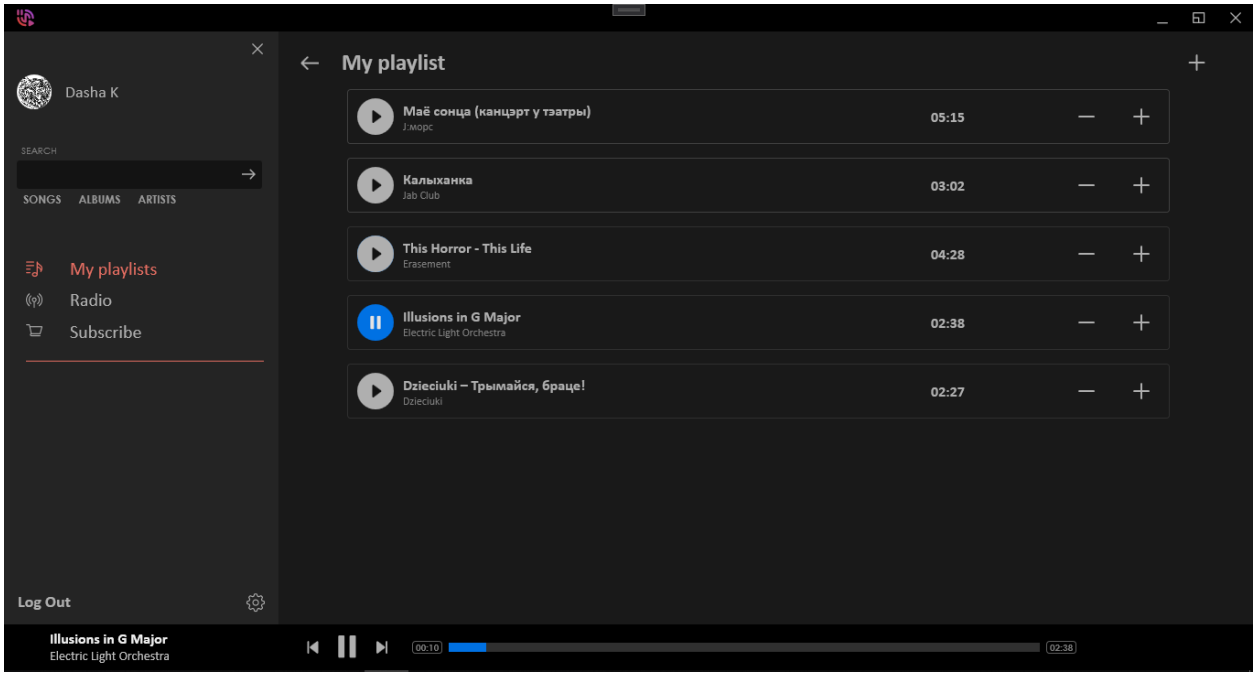

Рисунок 6.10 – Проигрывание композиций

Радио в приложении находится под одноименным названием на боковой панели приложения. Страница радио представлена на рисунке 6.11. Плюсы и

минусы на панели позволяют передвинуть радио вверх или вниз, а средний значок скопирует название проигрываемой композиции в буфер обмена.

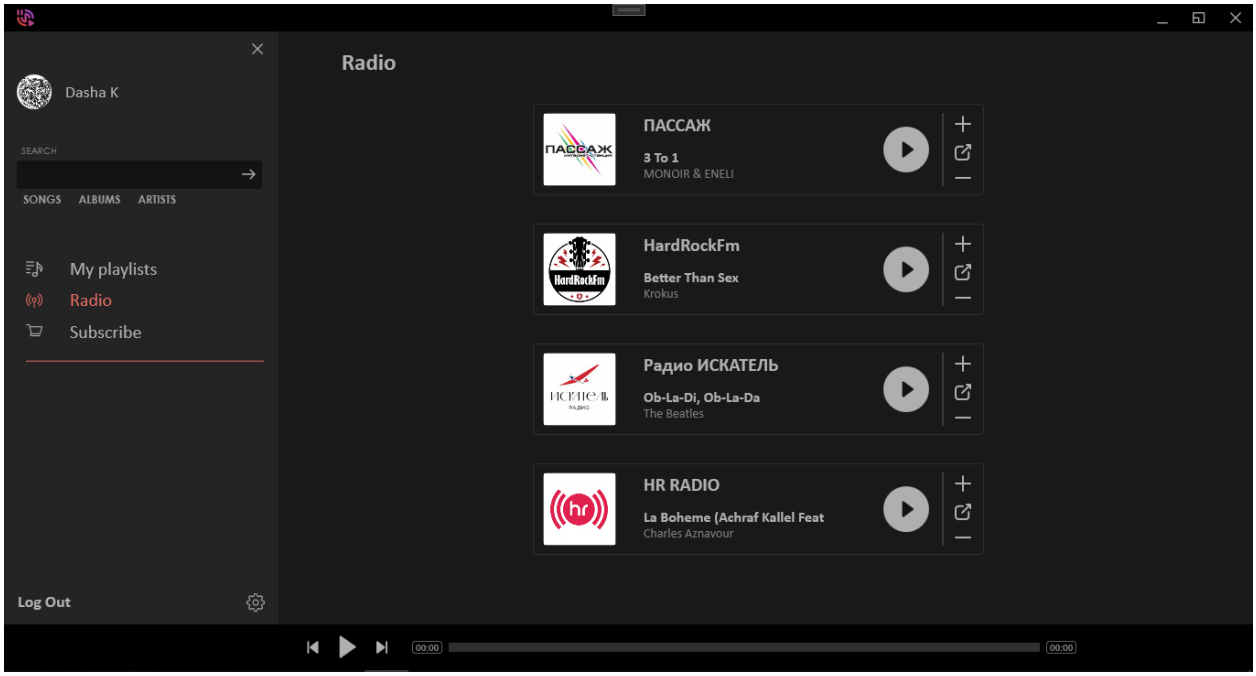

Рисунок 6.11 – Страница радио

Включить и управлять радио можно через кнопку воспроизведения и панель. После клика на картинку радиостанции откроется ее описание и недавний плейлист:

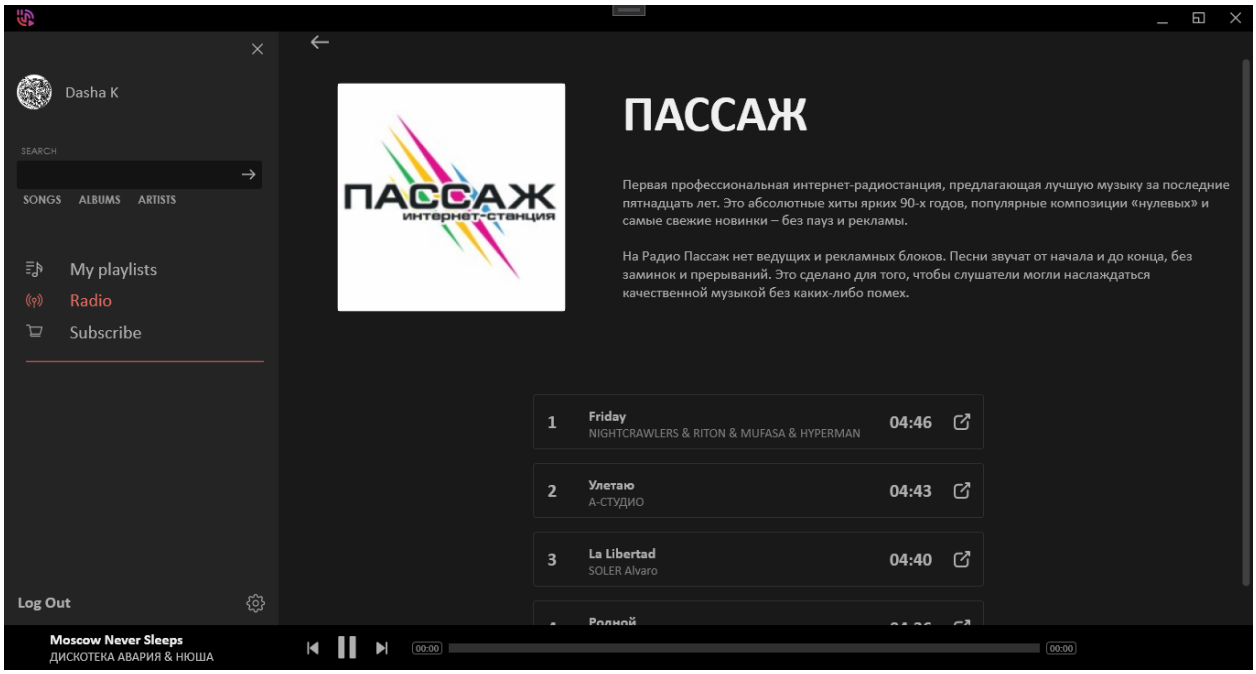

Рисунок 6.12 – Описание радиостанции

Для поиска по композициям необходимо ввести желаемую строку в поиск и нажать стрелочку. Полученный список песен можно проигрывать как отдельный плейлист:

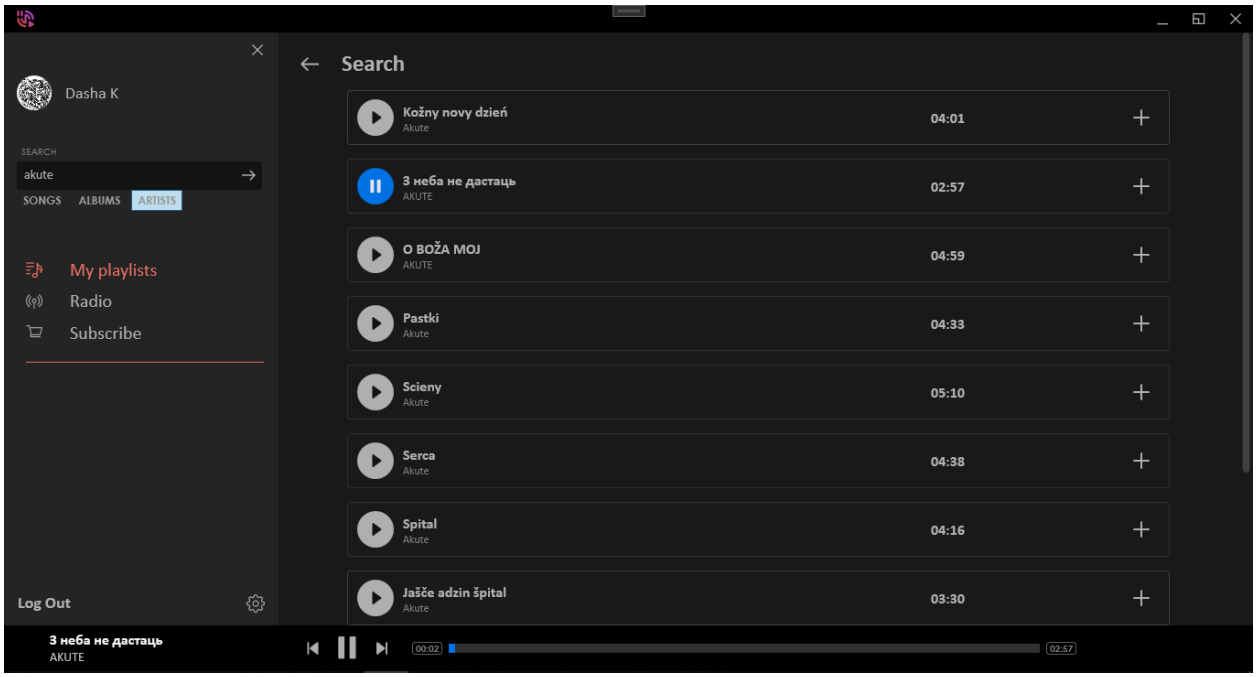

Рисунок 6.13 – Поиск и проигрывание композиций

Кроме этого, можно посмотреть статистику пользователя, нажав на его фотографию:

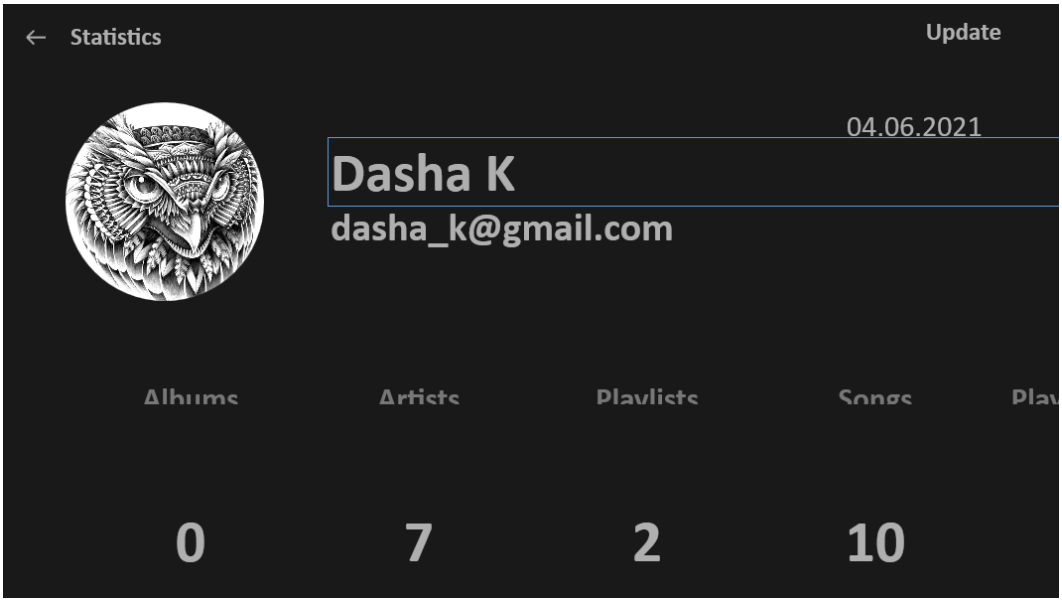

Рисунок 6.14 – Статистика пользователя

Приобрести дополнительный функционал можно на странице подписок. На ней отображаются актуальные данные по действующим подпискам. При отсутствии таковых она выглядит так:

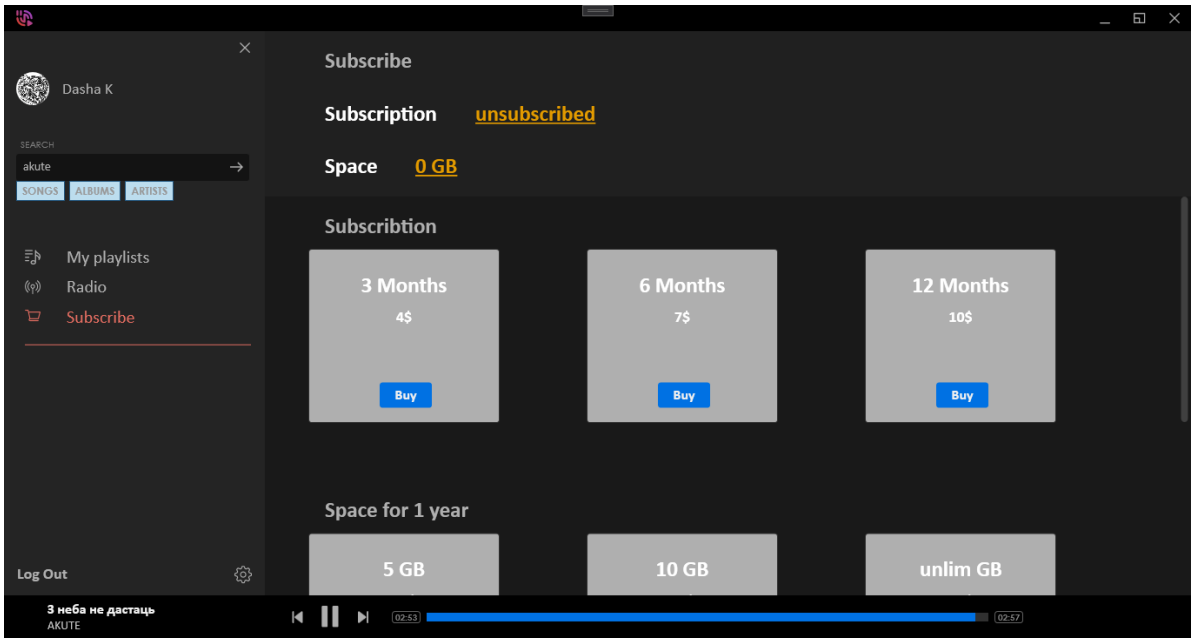

Рисунок 6.15 – Страница подписок

После выбора плана подписки совершить платеж можно, кликнув на «Buy» тарифного плана. Страница изменится и станет отображать дополнительные функции. Пример приложения при подписке на дополнительную память и стриминговый сервис:

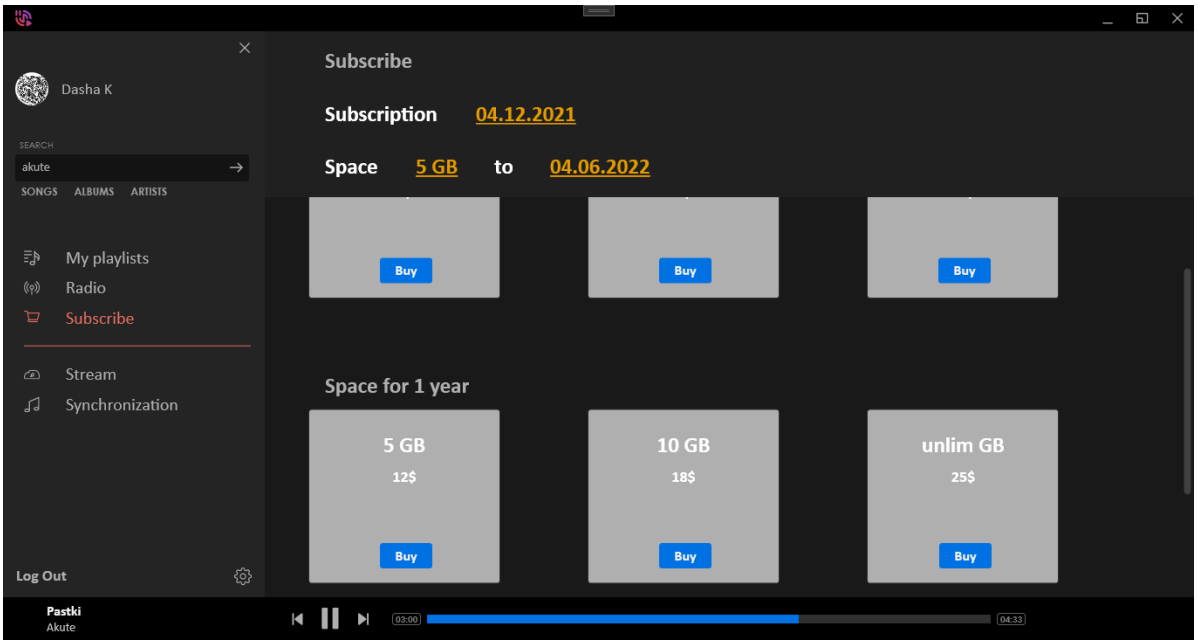

Рисунок 6.16 – Приложение после подписок

После подписки на дополнительную память на композиции появиться дополнительная кнопка – кнопка синхронизации, выглядящая, как стрелка. На рисунке 6.17 она подсвечена белым. Нажав на нее, можно синхронизировать композицию с сервером.

Рисунок 6.17 – Возможность синхронизации композиции

Настроить воспроизведение можно, нажав значок настроек внизу экрана. Появится эквалайзер:

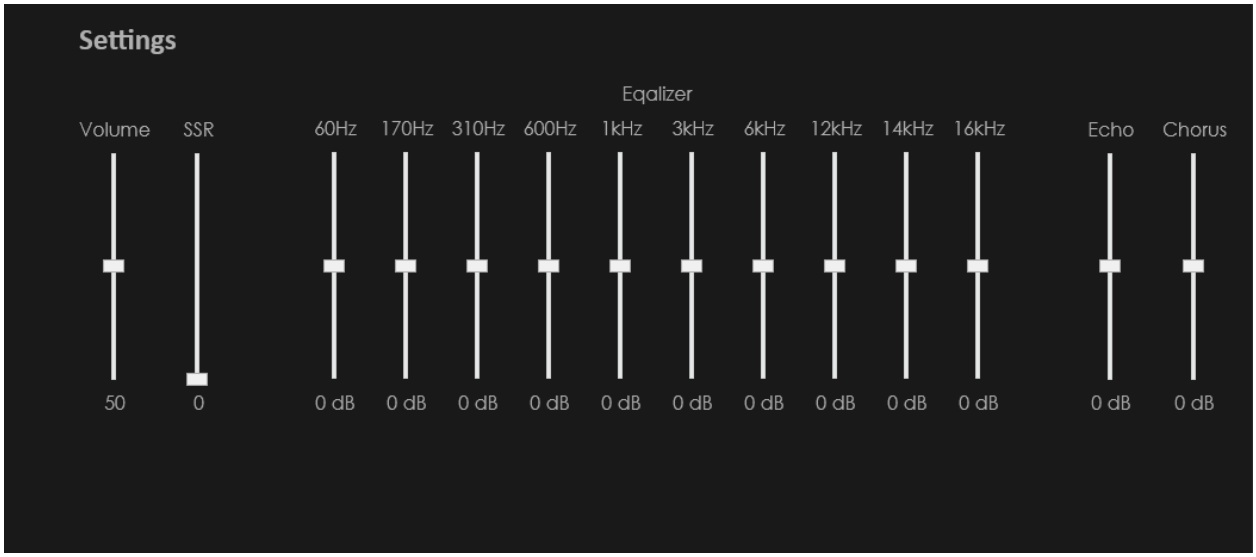

Рисунок 6.18 – Эквалайзер

Выйти из приложения можно, нажав «Log out» внизу экрана.

Стоит упомянуть, что приложение имеет свою базу данных, которая хранится в папке проекта. Если приложение не находит файла, на который ссылалось раньше, то оно его пропускает в проигрывании, затем удаляет из базы данных. Если удалить эту базу данных, то несинхронизированные данные восстановлению не подлежат! Поэтому будьте аккуратны, удаляя или перемещая файлы из папки проекта!

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ПРИЛОЖЕНИЯ ДЛЯ ВОСПРОИЗВЕДЕНИЯ МУЗЫКИ СО СТРИМИНГОВЫМ СЕРВИСОМ

7.1 Характеристика программного средства

В результате работы над дипломным проектом было реализовано приложения для воспроизведения музыки со стриминговым сервисом, которое предполагает прослушивание радио, стриминговых и локальных аудиозаписей. Приложение позволяет объединить в одну систему стримингового сервиса и классического плеера с радио.

Программное средство предназначено для широкого круга покупателей, которые являются физическими лицами. Каждый покупатель имеет свой аккаунт, с которого может дополнительно купить расширение пространства для загрузки своих аудиокомпозиций.

Актуальная потребность в созданном программном средстве связана с проблемой отсутствия готового решения, которое объединяет в себе сразу радио, плеер, стриминговый сервис, возможность синхронизации аудио. Программное средство решает актуальную потребность пользователей в одном приложении, которое заменит им несколько подобных.

Разработка и совершенствование программного средства предполагает его использование широким кругом потребителей на рынке и рассчитана на высокий спрос для свободной его реализации на рынке информационных технологий. Создание продукта возложено на специализированную организацию, основным видом деятельности которой является разработка. В разработке программного средства участвуют различные специалисты, в числе которых бизнес-аналитик, системный архитектор, программист, тестировщик и дизайнер.

Основные затраты на программный продукт определяются выплатой заработной платы участникам команды. А также необходимы средства для продвижения продукта на рынке за счет использования рекламы.

Программный продукт был разработан для получения прибыли путем подписки на сервис и расширении его возможностей. Для увеличения экономического эффекта команда разработчиков занимается усовершенствованием продукта, его доработкой, обработкой неисправностей, а также внедрением новых модулей в систему для удовлетворения актуальных потребностей организаций, новых бизнес-требований. Также в качестве одного из способов увеличения экономического эффекта предпринимаются доработки, усовершенствование продукта с использованием индивидуального подхода к платформам операционных систем, особенностей их структуры.

Разработанный программный продукт обладает следующими особенностями:

– система является универсальной, может быть развернута как на современном, так и на устаревшем программном обеспечении;

– пользовательский интерфейс интуитивно-понятный, позволяет пользователю быстро адаптироваться к приложению;

– в приложении организован безопасный доступ к данным строго ограниченному кругу лиц, определенному администратором.

7.2 Расчет затрат на разработку программного средства для реализации его на рынке

С первых дней разработки программного средства команда анализирует актуальные проблемы пользователей, связанные с музыкой и способах ее проигрывания и использования. Важным этапом является определение организацией-разработчиком ключевых индивидуальных качеств реализуемого приложения по сравнению с характеристиками уже существующих на рынке продуктов, которые являются конкурентами данному. Этот этап является существенным, так как он определит уровень потребительского спроса на продукт в будущем, способность превзойти конкурентов и выбиться в первые ряды на рынке.

Таким образом, команда формирует требования к продукту и минимальные сроки для его реализации. Необходимо также учитывать всевозможные риски при разработке и на основе рисков сделать вывод о сроках, необходимых для разработки программного средства в случае возможных проблем с функционалом, системой безопасности, большого числа выявленных при тестировании ошибок. Также в течение периода разработки продукта команда может вносить небольшие изменения в требования к продукту, что может привести к временным задержкам.

Расчет затрат на основную заработную плату команды разработчиков осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке программного средства отдельными исполнителями по формуле:

$$З_о = К_{пр} \sum_{i=1}^{n} З_{чi} \cdot t_i, \quad (7.1)$$

где $К_{пр}$ – коэффициент премий (по данным предприятия или в диапазоне 1,5–2); n – категории исполнителей, занятых разработкой программного средства; $З_{чi}$ – часовая заработная плата исполнителя i-й категории, р.; t_i – трудоемкость работ, выполняемых исполнителем i-й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, ч.

В разработке программного средства участвуют разнообразные специалисты, в числе которых бизнес-аналитик, системный архитектор,

программист, тестировщик и дизайнер. Бизнес-аналитик ответственен за анализ, определение потребностей пользователей и поиска всевозможных путей их решения. Системный архитектор необходим команде разработки программного средства с целью принятия ключевых решений в проекте касательно внутреннего устройство программной системы, а также подбора технических средств для ее реализации. Программист занимается непосредственно разработкой приложения на основе заданной архитектуры и сформулированных бизнес-требований. Тестировщик контролирует качество разрабатываемого продукта по мере выполнения определенных задач программистом. Дизайнер необходим для команды с целью разработки простого и интуитивно-понятного пользовательского интерфейса.

Таким образом, с учетом характеристик функций специалистов организацией была определена трудоемкость работ для каждого члена команды для разработки требуемого продукта. Размер заработной платы специалиста каждой категории соответствует установленному в организации-разработчике фактическому ее размеру. Дополнительно к основной заработной плате выплачивается премия, составляющая 20% от основного заработка. Часовая заработная плата каждого исполнителя определена путем деления его месячной заработной платы на количество рабочих часов в месяце, принятое равным 168 ч.

Расчет затрат на основную заработную плату команды разработчиков приведен в таблице 7.1.

Таблица 7.1 – Затраты на основную заработную плату команды разработчиков

Категория исполнителя	Месячная заработная плата, р.	Часовая заработная плата, р.	Трудоемкость работ, ч	Итого, р.
Бизнес-аналитик	2184	13	60	780
Системный архитектор	3360	20	90	1800
Программист	2520	15	560	8400
Тестировщик	1680	10	160	1600
Дизайнер	2016	12	70	840
Всего затраты на основную заработную плату разработчиков				13420
Премия 160%				21472
Всего затраты на заработную плату команды разработчиков				34892

Для разработки программного средства общая сумма инвестиций включает в себя основную, дополнительную заработную плату разработчиков, отчисления на социальные нужды и прочие расходы организации.

Расчет дополнительной заработной платы разработчиков производится по следующей формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $Н_д$ – норматив дополнительной заработной платы, для организации-разработчика принят равным 12%.

Отчисления на социальные нужды определяются по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (7.3)$$

где $Н_{соц}$ – ставка отчислений в ФСЗН и Белгосстрах (в соответствии с действующим законодательством по состоянию на 01.01.2020 г. – 34,6 %).

Прочие расходы рассчитываются по формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.4)$$

где $Н_{пр}$ – норматив прочих расходов, для организации-разработчика принят равным 35%.

Значение расходов на реализацию определяется формулой:

$$Р_р = \frac{З_о \cdot Н_р}{100}, \quad (7.5)$$

где $Н_р$ – норматив расходов на реализацию, принят равным 3%.

Общая сумма затрат на разработку определяется следующей формулой:

$$З_р = З_о + З_д + Р_{соц} + Р_{пр} + Р_р. \quad (7.6)$$

Расчет затрат на разработку программного средства приведен в таблице 7.2.

Таблица 7.2 – Основные статьи затрат на разработку программного средства

Наименование статьи затрат	Расчет по формуле (в таблице)	Сумма, р.
1	2	3
Основная заработная плата разработчиков	См. таблицу 7.1	34892
Дополнительная заработная плата разработчиков	$З_д = \frac{34892 \cdot 12}{100} \approx 4187$ р.	4187

Продолжение таблицы 7.2

1	2	3
Отчисления на социальные нужды	$\mathrm{Р}_{\text{соц}} = \frac{(34892 + 4187) \cdot 34{,}6}{100} \approx \approx 14487$ р.	14487
Прочие расходы	$\mathrm{Р}_{\text{пр}} = \frac{34892 \cdot 35}{100} = 12212$ р.	12212
Расходы на реализацию	$\mathrm{Р}_{\text{р}} = \frac{34892 \cdot 3}{100} \approx 1046$ р.	1046
Общая сумма затрат на разработку	$\mathrm{З}_{\text{р}} = 34892 + 4187 + 14487 + 12212 + 1046 = 66824$ р.	37393

Таким образом, была посчитана общая сумма затрат, требуемых организации-разработчику для реализации программного средства. Коммерческая реализация проектного решения требует инвестиций для разработки программного средства, для эксплуатации этого средства, систем безопасности, использования программных средств и платформы для хостинга.

При определении величины затрат на разработку, в результате анализа необходимого минимума специалистов команды, их трудоемкостей работы, а также с учетом возможных рисков при разработке, была определена стоимость реализации продукта. На основе полученных данных организацией-разработчиком определяется стоимость лицензии для продажи продукта потенциальным пользователям.

7.3 Расчет экономического эффекта от реализации программного средства на рынке

Экономический эффект является конечным результатом внедрения программного средства на рынок. Экономический эффект характеризуется дополнительным доходом, который может быть получен за счет продажи подписок и расширения возможностей.

После проведения анализа количества платных подписок Яндекс.Музыки и Spotify (3 миллиона и 155 миллионов соответственно) было предположено, что начальный приток клиентов составит 100 000 в год. Учитывая, что Spotify – мировая известная площадка, а Яндекс.Музыка развивается с 2010 года и монетизируется дополнительно за счет бесплатной подписки с рекламой, цифра в 100 000 является вполне достижимой.

Для анализа цены на продукт были просмотрены цены на два популярных стриминговых сервиса, описанных выше. Подписка на Яндекс.Музыку на март 2021 года стоит 5,49 р. в месяц или 4,57 р. в месяц при подписке на год. Подписка на Spotify на март 2021 года стоит 4,99$ (≈ 13 р.) в месяц. Чтобы привлечь пользователей, необходимо сделать подписку немного

дешевле, чем у конкурентов, поэтому наиболее приемлемой ценой выбрана цена в 3,00 р. К тому же на старте такая цена поможет не отпугнуть потенциальных покупателей.

Прирост чистой прибыли (экономический эффект) организации-разработчика описывается следующей формулой:

$$\Delta П_{ч}^{р} = (Ц_{отп} \cdot N - НДС) \cdot Р_{пр} \cdot \left(1 - \frac{Н_{п}}{100}\right), \quad (7.7)$$

где $Ц_{отп}$ – отпускная цена копии (лицензии) программного средства, р.; N – количество копий (лицензий) программного средства, реализуемое за год, шт.; НДС – сумма налога на добавленную стоимость, р.; $Р_{пр}$ – рентабельность продаж копий (лицензий), (20-40%); $Н_{п}$ – ставка налога на прибыль согласно действующему законодательству, % (по состоянию на 01.01.2020 г. – 18%).

Для определения налога на добавленную стоимость используется формула:

$$НДС = \frac{Ц_{отп} \cdot N \cdot Н_{дс}}{100\% + Н_{дс}}, \quad (7.8)$$

где $Н_{дс}$ – ставка налога на добавленную стоимость в соответствии с действующим законодательством, % (по состоянию на 01.01.2020 г. – 20 %).

Организация-разработчик не является резидентом Парка высоких технологий, поэтому расчеты прироста чистой прибыли производятся без упрощения формулы.

Налог на добавленную стоимость рассчитаем по формуле (7.8):

$$НДС = \frac{3 \cdot 100000 \cdot 20}{100 + 20} = 50000 \text{ р.}$$

Затем определим прирост чистой прибыли организации-разработчика по формуле (7.7):

$$\Delta П_{ч}^{р} = (100000 \cdot 3 - 50000) \cdot 0{,}4 \cdot \left(1 - \frac{18}{100}\right) = 82000 \text{ р.}$$

7.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Оценка экономической эффективности разработки и реализации программного средства на рынке зависит от результата сравнения инвестиций в его разработку (модернизацию, совершенствование) и полученного годового прироста чистой прибыли.

Таким образом, по результатам расчета прироста чистой прибыли организации видно, что значение годовой чистой прибыли превышает общую сумму затрат на разработку программного продукта. Инвестиции окупятся менее чем за год функционирования продукта.

Оценка экономической эффективности инвестиций в разработку программного средства осуществляется с помощью расчета рентабельности затрат по формуле:

$$Р_{и} = \frac{\Delta П_{ч}^{р}}{З_{р}} \cdot 100\ \%, \quad (7.9)$$

где $\Delta П_{ч}^{р}$ – прирост чистой прибыли, полученной от реализации программного средства на рынке, р.; $З_{р}$ – затраты на разработку программного средства, р.

Рассчитаем рентабельность затрат по формуле (7.9):

$$Р_{и} = \frac{82000}{66824} \cdot 100\ \% \approx 123\%.$$

Из экономических законов следует, что затрат на разработку программного средства и его реализация на рынке информационных технологий окажутся экономически эффективными, если рентабельность превысит 100%, учитывая ставку по банковским долгосрочным депозитам. Поэтому после анализа определено, что по установленной цене программное средство целесообразно разрабатывать и реализовывать.

7.5 Выводы об экономической эффективности и целесообразности инвестиций

В результате проделанной работы была доказана целесообразность разработки приложения для воспроизведения музыки со стриминговым сервисом как с технической, так и экономической точки зрения.

По результатам проведения технико-экономического обоснования были получены следующие результаты:

1. Инвестиции на разработку программного средства равны 66824 р.
2. Прирост чистой прибыли организации-разработчика равен 82000 р.
3. Рентабельность затрат равна 123%.

Программное средство является затребованным на рынке, его разработка требует определенных инвестиций. Но по итогу анализа экономической эффективности разработки и реализации программного средства на рынке было доказано, что затраты окупятся в первый же год функционирования продукта и разработка продукта является эффективной.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломного проекта было создано приложение для воспроизведения музыки со стриминговым сервисом. Такое приложение соответствует всем требованиям, которые были определены на начальном этапе разработки на основе выявленных задач и целей. Также в итоге были определены преимущества и недостатки приложения.

Среди преимуществ разрабатываемой системы можно выделить следующие:

– в соответствии с технико-экономическим обоснованием продукт будет окуплен в первый же год его функционирования на рынке;

– интерфейс пользователя является простым и интуитивно понятным;

– архитектура приложения построена таким образом, что существующую функциональность беспрепятственно можно расширить.

Были выявлены следующие недостатки приложения:

– на рынке существует большое число аналогов, которые имеют небольшие отличия и большую функциональность;

– развертывание серверной части привязывается к операционной системе Windows;

– приложение сделано для использования на компьютере, хотя по тенденциям последнего времени пользователям удобнее пользоваться телефонами.

В соответствии с выявленными недостатками в будущем для развития проекта запланировано внедрение гибкости в пользовательский интерфейс с целью улучшенного доступа к сервису со стороны мобильных устройств, расширение функциональности, которая бы выделила проект среди аналогов, существующих на рынке, а также перевод разработки с фреймворка, привязанному к операционной системе Windows, на кроссплатформенный фреймворк.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Рейтинг социальных сетей на 2020 год [Электронный ресурс]. – Электронные данные. – Режим доступа: https://gs.seo-auditor.com.ru/socials/2020/ – Дата доступа: 20.03.2021.

[2] Условия авторских прав у Яндекс [Электронный ресурс]. – Электронные данные. – Режим доступа: https://yandex.by/support/music/performers-and-copyright-holders/terms.html – Дата доступа: 20.03.2021.

[3] Плеер kmplayer [Электронный ресурс]. – Электронные данные. – Режим доступа: https://kmplayer.ru.uptodown.com/windows – Дата доступа: 20.03.2021.

[4] Плеер winamp [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.winamp.com/ – Дата доступа: 20.03.2021.

[5] Плеер comboplayer [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.comboplayer.ru/ – Дата доступа: 20.03.2021.

[6] Плеер videolan [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.videolan.org/index.be.html – Дата доступа: 20.03.2021.

[7] Плеер aimp [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.aimp.ru/ – Дата доступа: 20.03.2021.

[8] Backend-as-a-Service [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.cloudflare.com/learning-serv/serverless/glossary/backend-as-a-service-baas/ – Дата доступа: 21.03.2021.

[9] Регулярные выражения [Электронный ресурс]. – Электронные данные. – Режим доступа: https://parseplatform.org/– Дата доступа: 21.03.2021.

[10] Платформа back4app [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.back4app.com/features – Дата доступа: 21.03.2021.

[11] Платформа firebase [Электронный ресурс]. – Электронные данные. – Режим доступа: https://firebase.google.com/use-cases– Дата доступа: 21.03.2021.

[12] Фреймфорк .NET [Электронный ресурс]. – Электронные данные. – Режим доступа: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet – Дата доступа: 21.03.2021.

[13] Технология WPF [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/sharp/wpf/1.php – Дата доступа: 21.03.2021.

[14] Язык XAML [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/sharp/wpf/2.php – Дата доступа: 21.03.2021.

ПРИЛОЖЕНИЕ А
(обязательное)

Спецификация программного дипломного проекта

ПРИЛОЖЕНИЕ Б
(обязательное)

Серверная часть. Модель данных

ПРИЛОЖЕНИЕ В
(обязательное)

Клиентская часть. Модель данных

ПРИЛОЖЕНИЕ Г
(обязательное)

Листинг кода

Файл *MainWindow.xaml.cs*

```
using System;
using System.Runtime.InteropServices;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Interop;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using Un4seen.Bass;

namespace DCO_Player
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        //Максимизация окна без полей
        private static IntPtr WindowProc(IntPtr hwnd, int msg, IntPtr wParam,
IntPtr lParam, ref bool handled)
        {
            switch (msg)
            {
                case 0x0024:
                    WmGetMinMaxInfo(hwnd, lParam);
                    handled = true;
                    break;
            }
            return (IntPtr)0;
        }

        private static void WmGetMinMaxInfo(IntPtr hwnd, IntPtr lParam)
        {
            MINMAXINFO mmi = (MINMAXINFO)Marshal.PtrToStructure(lParam,
typeof(MINMAXINFO));
            int MONITOR_DEFAULTTONEAREST = 0x00000002;
            IntPtr monitor = MonitorFromWindow(hwnd,
MONITOR_DEFAULTTONEAREST);
            if (monitor != IntPtr.Zero)
            {
                MONITORINFO monitorInfo = new MONITORINFO();
                GetMonitorInfo(monitor, monitorInfo);
                RECT rcWorkArea = monitorInfo.rcWork;
                RECT rcMonitorArea = monitorInfo.rcMonitor;
                mmi.ptMaxPosition.x = Math.Abs(rcWorkArea.left -
rcMonitorArea.left);
                mmi.ptMaxPosition.y = Math.Abs(rcWorkArea.top -
rcMonitorArea.top);
```

```
                mmi.ptMaxSize.x = Math.Abs(rcWorkArea.right -
rcWorkArea.left);
                mmi.ptMaxSize.y = Math.Abs(rcWorkArea.bottom -
rcWorkArea.top);
            }
            Marshal.StructureToPtr(mmi, lParam, true);
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct POINT
        {
            /// <summary>x coordinate of point.</summary>
            public int x;
            /// <summary>y coordinate of point.</summary>
            public int y;
            /// <summary>Construct a point of coordinates (x,y).</summary>
            public POINT(int x, int y)
            {
                this.x = x;
                this.y = y;
            }
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct MINMAXINFO
        {
            public POINT ptReserved;
            public POINT ptMaxSize;
            public POINT ptMaxPosition;
            public POINT ptMinTrackSize;
            public POINT ptMaxTrackSize;
        };

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
        public class MONITORINFO
        {
            public int cbSize = Marshal.SizeOf(typeof(MONITORINFO));
            public RECT rcMonitor = new RECT();
            public RECT rcWork = new RECT();
            public int dwFlags = 0;
        }

        [StructLayout(LayoutKind.Sequential, Pack = 0)]
        public struct RECT
        {
            public int left;
            public int top;
            public int right;
            public int bottom;
            public static readonly RECT Empty = new RECT();
            public int Width { get { return Math.Abs(right - left); } }
            public int Height { get { return bottom - top; } }
            public RECT(int left, int top, int right, int bottom)
            {
                this.left = left;
                this.top = top;
                this.right = right;
                this.bottom = bottom;
            }
            public RECT(RECT rcSrc)
            {
                left = rcSrc.left;
```

```
                top = rcSrc.top;
                right = rcSrc.right;
                bottom = rcSrc.bottom;
            }
            public bool IsEmpty { get { return left >= right || top >=
bottom; } }
            public override string ToString()
            {
                if (this == Empty) { return "RECT {Empty}"; }
                return "RECT { left : " + left + " / top : " + top + " /
right : " + right + " / bottom : " + bottom + " }";
            }
            public override bool Equals(object obj)
            {
                if (!(obj is Rect)) { return false; }
                return (this == (RECT)obj);
            }
            /// <summary>Return the HashCode for this struct (not garanteed
to be unique)</summary>
            public override int GetHashCode() => left.GetHashCode() +
top.GetHashCode() + right.GetHashCode() + bottom.GetHashCode();
            /// <summary> Determine if 2 RECT are equal (deep
compare)</summary>
            public static bool operator ==(RECT rect1, RECT rect2) { return
(rect1.left == rect2.left && rect1.top == rect2.top && rect1.right ==
rect2.right && rect1.bottom == rect2.bottom); }
            /// <summary> Determine if 2 RECT are different(deep
compare)</summary>
            public static bool operator !=(RECT rect1, RECT rect2) { return
!(rect1 == rect2); }
        }

        [DllImport("user32")]
        internal static extern bool GetMonitorInfo(IntPtr hMonitor,
MONITORINFO lpmi);

        [DllImport("User32")]
        internal static extern IntPtr MonitorFromWindow(IntPtr handle, int
flags);

        public static MainWindow Instance { get; private set; }

        public static Radio radioPage;
        public static My_playlists MPPage;
        public static Subscribe SubPage;
       // public Albums AlbumsPage;
        public static Settings SettingsPage;

        public MainWindow()
        {

            InitializeComponent();
            Instance = this;
// Ссылка на окно
            WindowState = WindowState.Maximized;
// При запуске состояние окна - на полный экран
            SourceInitialized += MainWindow_SourceInitialized;

            Frame.NavigationUIVisibility = NavigationUIVisibility.Hidden;
// Прячем панель навигации
```

```
            AccountName.Text = Profile.name + " " + Profile.surname;
// Имя и фамилия в профиле
            if(Profile.imageSrc != "" && Profile.imageSrc != null)
            {
                AccountImage.ImageSource = new BitmapImage(new
Uri(Environment.CurrentDirectory + Profile.imageSrc, UriKind.Absolute)); //
Изображение профиля
            }

            radioPage = new Radio();
// Новые образы страниц, для загрузки их вместе с окном
            MPPage = new My_playlists();
            SubPage = new Subscribe();
            SettingsPage = new Settings();
            if (Profile.GBDate > DateTime.Now)
                MainWindow.Instance.SpaceTextBlock.Visibility =
Visibility.Visible;
            if (Profile.subscriptionDate > DateTime.Now)
                MainWindow.Instance.StreamTextBlock.Visibility =
Visibility.Visible;
            Frame.Navigate(MPPage);

            MusicStream.InitBass(MusicStream.HZ);
        }

        // Событие, направленное на правильную работу окна
        private void MainWindow_SourceInitialized(object sender, EventArgs e)
        {
            IntPtr handle = (new WindowInteropHelper(this)).Handle;
            HwndSource.FromHwnd(handle).AddHook(new
HwndSourceHook(WindowProc));
        }

        // Событие скрытия окна
        private void MinimizeWindow_Executed(object sender, RoutedEventArgs
e)
        {
            WindowState = WindowState.Minimized;
        }

        // Событие "на полный экран"
        private void MaximizeWindow_Executed(object sender, RoutedEventArgs
e)
        {
            WindowState = WindowState == WindowState.Maximized ?
WindowState.Normal : WindowState.Maximized;
            if (WindowState.Normal == 0)
            {
                this.Width = 1350;
                this.Height = SystemParameters.FullPrimaryScreenHeight + 8;
                this.Top = 8;
                this.Left = 8;
            }

        }

        // Событие закрытия приложения
        private void CloseWindow_Executed(object sender, RoutedEventArgs e)
        {
            Close();
        }
```

```
        // Событие скрытия меню
        private void CloseMenu_Click(object sender, RoutedEventArgs e)
        {
            ButtonMenuOpen.Visibility = Visibility.Visible;
            ButtonMenuClose.Visibility = Visibility.Collapsed;
            LogOut.Visibility = Visibility.Collapsed;
            Menu.Margin = new Thickness(4,0,0,0);
            search.Visibility = Visibility.Collapsed;
            My_playlists.Margin = new Thickness(10, 140, 0, 0);
            ButtonMenuOpen.Margin = new Thickness(0, 6, 6, 0);
            LogOutGrid.Margin = new Thickness(5, 0, 13, 11);
            Songs.Visibility = Visibility.Collapsed;
            Albums.Visibility = Visibility.Collapsed;
            Artists.Visibility = Visibility.Collapsed;
        }

        // Событие развертывания меню
        private void OpenMenu_Click(object sender, RoutedEventArgs e)
        {
            ButtonMenuOpen.Visibility = Visibility.Collapsed;
            ButtonMenuClose.Visibility = Visibility.Visible;
            LogOut.Visibility = Visibility.Visible;
            Menu.Margin = new Thickness(14, 0, 0, 0);
            search.Visibility = Visibility.Visible;
            My_playlists.Margin = new Thickness(10, 47, 0, 0);
            ButtonMenuOpen.Margin = new Thickness(0, 6, 6, 0);
            LogOutGrid.Margin = new Thickness(15, 0, 15, 11);
            Songs.Visibility = Visibility.Visible;
            Albums.Visibility = Visibility.Visible;
            Artists.Visibility = Visibility.Visible;
        }

        // Событие перехода к странице радио
        private void Radio_MouseDown(object sender, MouseButtonEventArgs e)
        {
            SubscribeBrush.Brush = SubscribeTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#AFAFAF"));
            RadioBrush.Brush = RadioTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#E46E62"));
            PlaylistsBrush.Brush = PlaylistsTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#AFAFAF"));
            Frame.Navigate(radioPage);
        }

        // Событие перехода к странице плейлистов
        private void Playlists_MouseDown(object sender, MouseButtonEventArgs
e)
        {
            SubscribeBrush.Brush = SubscribeTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#AFAFAF"));
            PlaylistsBrush.Brush = PlaylistsTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#E46E62"));
            RadioBrush.Brush = RadioTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#AFAFAF"));
            Frame.Navigate(new My_playlists());
        }

        // Подписки
        private void Subscribe_MouseDown(object sender, MouseButtonEventArgs
e)
        {
```

```
            SubscribeBrush.Brush = SubscribeTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#E46E62"));
            PlaylistsBrush.Brush = PlaylistsTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#AFAFAF"));
            RadioBrush.Brush = RadioTextBlock.Foreground =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#AFAFAF"));
            Frame.Navigate(SubPage);
        }

        // Событие запуска воспроизведения
        private void Play_Click(object sender, RoutedEventArgs e)
        {
          //  MusicStream.Stop();
            MusicStream.Play();
            MusicStream.StreamLineStart();
        }

        // События останова воспроизведения
        private void Stop_Click(object sender, RoutedEventArgs e)
        {
            MusicStream.Stop();
            MusicStream.StreamLineStop();
        }

        // Событие выхода из аккаунта
        private void LogOut_MouseDown(object sender, MouseButtonEventArgs e)
        {
            Bass.BASS_ChannelStop(MusicStream.Stream);
            Bass.BASS_StreamFree(MusicStream.Stream);

            Start start = new Start();
            start.Show();

            MainWindow.Instance.Close();
        }
        private void Stream_MouseDown(object sender, MouseButtonEventArgs e)
        {

        }

        private void Space_MouseDown(object sender, MouseButtonEventArgs e)
        {

        }
        private void Ellipse_MouseDown(object sender, MouseButtonEventArgs e)
        {
            Frame.Navigate(new Statictics());
        }

        private void Search_Click(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(new Search());
        }

        public void NextButton(object sender, RoutedEventArgs e)
        {
            MusicStream.Next();
            MusicStream.NextPoint = true;
        }

        public void EndButton(object sender, RoutedEventArgs e)
        {
```

```
            MusicStream.Back();
            MusicStream.EndPoint = true;
        }

        private void Settings_MouseDown(object sender, MouseButtonEventArgs
e)
        {
            Frame.Navigate(SettingsPage);
        }

        private void TimeLine_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            if(Math.Abs(TimeLine.Value - MusicStream.CorTime) > 2) {
                Bass.BASS_ChannelSetPosition(MusicStream.Stream,
(double)MusicStream.GetTimeOfStream(MusicStream.Stream) * (TimeLine.Value /
646.0));
            }

        }


    }
}
```

Файл *Firebase.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using Google.Cloud.Firestore;
using Google.Cloud.Storage.V1;
using Google.Apis.Storage.v1.Data;

namespace DCO_Player
{
    static class Firebase
    {
        static string project_Id = "dco-player-74813";
        // static string bucketName = "dco-player-74813.appspot.com";

        static FirestoreDb db;
        static CollectionReference coll_ref;
        static DocumentReference doc_ref;
        public static void Init()
        {
            string credential_path = @"D:\!! New diplom\dco-player-74813-
c9cace3048cd.json";


Environment.SetEnvironmentVariable("GOOGLE_APPLICATION_CREDENTIALS",
credential_path);

            db = FirestoreDb.Create(project_Id);


            //  StorageClient storage = StorageClient.Create();
            //  Bucket bucket = storage.CreateBucket(project_Id, bucketName);
            //  foreach (var b in storage.ListBuckets(project_Id))
            //      Console.WriteLine(b.Name);
        }
```

```
        public static void AddUser()
        {
            doc_ref = db.Collection("Users").Document(Profile.login);
            Dictionary<string, object> user = new Dictionary<string, object>
            {
                { "Id_user", Profile.Id_user.ToString() },
                { "Name", Profile.name },
                { "Surname", Profile.surname },
                { "Login", Profile.login },
                { "Password", Profile.password },
                { "Create_date", Profile.createDate.ToString() },
                { "User_image_source", Profile.imageSrc },
                { "Subscription_Date", Profile.subscriptionDate.ToString() },
                { "N_GB", Profile.gbs },
                { "GB_Date", Profile.GBDate.ToString() }
            };
            doc_ref.SetAsync(user);
        }

        public static void UpdateUser()
        {
            doc_ref = db.Collection("Users").Document(Profile.login);
            Dictionary<string, object> user = new Dictionary<string, object>
            {
                { "Subscription_Date", Profile.subscriptionDate.ToString() },
                { "N_GB", Profile.gbs },
                { "GB_Date", Profile.GBDate.ToString() }
            };
            doc_ref.UpdateAsync(user);
        }

        public static bool UserPayment(float money, int code, int gbs, int
sub)
        {
            Guid id_pay = Guid.NewGuid();
            doc_ref = db.Collection("Users/" + Profile.login +
"/Payments").Document(id_pay.ToString());
            Dictionary<string, object> pay = new Dictionary<string, object>
            {
                { "Id_payment", id_pay.ToString() },
                { "Money", money },
                { "Code", code },
                { "GBs", gbs },
                { "Subscription", sub },
                { "Date_time", DateTime.Now.ToString() }
            };
            doc_ref.CreateAsync(pay);
            DocumentSnapshot snapshot = doc_ref.GetSnapshotAsync().Result;
            if (!snapshot.Exists)
                return false;
            UpdateUser();
            return true;
        }

        public static bool CheckUser(string login)
        {
            doc_ref = db.Collection("Users").Document(login);
            DocumentSnapshot snapshot = doc_ref.GetSnapshotAsync().Result;
            if (snapshot.Exists)
                return true;
            else
                return false;
```

```
        }

        public static bool GetUser(string login)
        {
            doc_ref = db.Collection("Users").Document(login);
            DocumentSnapshot snapshot = doc_ref.GetSnapshotAsync().Result;
            if (!snapshot.Exists)
                return false;

            Profile.Id_user =
Guid.Parse(snapshot.GetValue<string>("Id_user"));
            Profile.name = snapshot.GetValue<string>("Name");
            Profile.surname = snapshot.GetValue<string>("Surname");
            Profile.login = snapshot.GetValue<string>("Login");
            Profile.password = snapshot.GetValue<string>("Password");
            Profile.createDate =
DateTime.Parse(snapshot.GetValue<string>("Create_date"));
            Profile.imageSrc =
snapshot.GetValue<string>("User_image_source");
            Profile.subscriptionDate =
DateTime.Parse(snapshot.GetValue<string>("Subscription_date"));
            Profile.gbs = snapshot.GetValue<int>("N_GB");
            Profile.GBDate =
DateTime.Parse(snapshot.GetValue<string>("GB_date"));
            return true;
        }
        public static void DeleteUser()
        {
            doc_ref = db.Collection("Users").Document(Profile.login);
            doc_ref.DeleteAsync();
        }

        public static void toserver(RadioStation reader)
        {
            doc_ref = db.Collection("Radio").Document(reader.name);
            Dictionary<string, object> radio = new Dictionary<string, object>
            {
                { "Id_radio", reader.Id.ToString() },
                { "Radio", reader.name },
                { "Description", reader.descr },
                { "Stream", reader.stream },
                { "Page", reader.page },
                { "Radio_image_source", reader.imageSrc }
            };
            doc_ref.CreateAsync(radio);
        }

        public static List<RadioStation> GetRadio()
        {
            List<RadioStation> radioStations = new List<RadioStation>();
            coll_ref = db.Collection("Radio");
            QuerySnapshot snapshot = coll_ref.GetSnapshotAsync().Result;

            foreach (var doc in snapshot.Documents)
            {
                RadioStation station = new RadioStation();
                station.Id = doc.GetValue<int>("Id_radio");
                station.name = doc.GetValue<string>("Radio");
                station.descr = doc.GetValue<string>("Description");
                station.stream = doc.GetValue<string>("Stream");
                station.page = doc.GetValue<string>("Page");
```

```
                station.imageSrc =
doc.GetValue<string>("Radio_image_source");

                radioStations.Add(station);
            }
            return radioStations;
        }

        public static UserPlaylist GetPlaylist(string Id_playlist)
        {
            string[] split1 = { " , " };
            string[] split2 = { " = " };
            UserPlaylist playlist = new UserPlaylist();
            doc_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Playlists").Document(Id_playlist);
            DocumentSnapshot doc = doc_ref.GetSnapshotAsync().Result;
            if (!doc.Exists)
                return null;
            playlist.Id_user = Guid.Parse(doc.GetValue<string>("Id_user"));
            playlist.Id_playlist =
Guid.Parse(doc.GetValue<string>("Id_playlist"));
            playlist.name = doc.GetValue<string>("Name");
            playlist.lastUpdate =
DateTime.Parse(doc.GetValue<string>("Last_update"));
            playlist.lastSync =
DateTime.Parse(doc.GetValue<string>("Last_sync"));
            playlist.imageSrc =
doc.GetValue<string>("Playlist_image_source");
            var s = doc.GetValue<string>("Songs");
            List<Tuple<Guid, int>> songs = new List<Tuple<Guid, int>>();
            if (s != "")
                songs = s.Split(split1, StringSplitOptions.None)
                        .Select(p => p.Split(split2,
StringSplitOptions.None))
                        .Select(p => new Tuple<Guid, int>(Guid.Parse(p[0]),
int.Parse(p[1])))
                        .ToList();
            songs = songs.OrderByDescending(p => p.Item2).ToList();
            playlist.songs = GetSongs(songs.Select(p => p.Item1).ToList());

            return playlist;

        }
        public static List<UserPlaylist> GetPlaylists(bool deleted = false)
        {
            List<UserPlaylist> playlists = new List<UserPlaylist>();
            coll_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Playlists");
            QuerySnapshot snapshot = coll_ref.GetSnapshotAsync().Result;

            foreach (var doc in snapshot.Documents)
            {
                UserPlaylist playlist = new UserPlaylist();
                playlist = GetPlaylist(doc.GetValue<string>("Id_playlist"));
                if (playlist.name == UserPlaylist.NULL_NAME && deleted)
                    playlists.Add(playlist);
                else
                    continue;
            }
            return playlists;
        }
```

```
        public static bool AddPlaylist(UserPlaylist playlist)
        {
            return AddPlaylists(new List<UserPlaylist> { playlist });
        }
        public static bool AddPlaylists(List<UserPlaylist> playlists)
        {
            int count = 0;
            string songs;

            foreach (var playlist in playlists)
            {
                var doc_name = playlist.Id_playlist.ToString();
                doc_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Playlists").Document(doc_name);
                if (playlist.songs.Count > 0)
                {
                    songs = string.Join(" , ", playlist.songs.Select(kv =>
kv.Id_song.ToString() + " = " + kv.n_sequence).ToArray());
                }
                else
                    songs = "";

                Dictionary<string, object> pl = new Dictionary<string,
object>
            {
                { "Id_user", playlist.Id_user.ToString() },
                { "Id_playlist", playlist.Id_playlist.ToString() },
                { "Name", playlist.name },
                { "Last_update", playlist.lastUpdate.ToString() },
                { "Last_sync", DateTime.Now.ToString() },
                { "Playlist_image_source", playlist.imageSrc },
                { "Songs", songs }
            };
                doc_ref.SetAsync(pl);

                if (doc_ref.GetSnapshotAsync().Result.Exists)
                    count++;
            }
            if (count == playlists.Count)
                return true;
            else
                return false;
        }

        public static bool DeletePlaylist(Guid id, bool NULL_NAME = true)
        {
            doc_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Playlists").Document(id.ToString());
            if (NULL_NAME)
            {
                Dictionary<string, object> pl = new Dictionary<string,
object>
            {
                { "Name", UserPlaylist.NULL_NAME },
                { "Id_playlist", id.ToString() }
            };
                doc_ref.SetAsync(pl);
            }
            else
                doc_ref.DeleteAsync();
            DocumentSnapshot snapshot = doc_ref.GetSnapshotAsync().Result;
            if (snapshot.Exists == NULL_NAME)
```

```
                return false;
            else
                return true;
        }

        public static List<Song> GetSongs(List<Guid> Id_songs = null, bool
all = false)
        {
            List<Song> songs = new List<Song>();
            List<Song> songs_playlist = new List<Song>();
            Song s;
            coll_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Songs");
            QuerySnapshot snapshot = coll_ref.GetSnapshotAsync().Result;
            int i = 0;
            foreach (var doc in snapshot.Documents)
            {
                Song song = new Song();
                song.Id_song = Guid.Parse(doc.GetValue<string>("Id_song"));
                song.is_local = doc.GetValue<bool>("Is_local");
                song.full_name = doc.GetValue<string>("Full_name");
                song.name = doc.GetValue<string>("Name");
                song.artist = doc.GetValue<string>("Artist");
                song.album = doc.GetValue<string>("Album");
                song.length = doc.GetValue<int>("Length");
                song.path = doc.GetValue<string>("Path");
                songs.Add(song);
            }
            if (all)
                return songs;
            else
            {
                foreach(var id in Id_songs)
                {
                    s = songs.First(o => o.Id_song == id);
                    s.n_sequence = i;
                    songs_playlist.Add(s);
                    i++;
                }
                return songs_playlist;
            }
        }

        public static bool AddSong(Song song)
        {
            return AddSongs(new List<Song> { song });
        }
        public static bool AddSongs(List<Song> songs)
        {
            int count = 0;

            foreach (var song in songs)
            {
                doc_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Songs").Document(song.Id_song.ToString());

                Dictionary<string, object> s = new Dictionary<string, object>
            {
                { "Id_song", song.Id_song.ToString() },
                { "Is_local", song.is_local.ToString() },
                { "Full_name", song.full_name },
                { "Name", song.name },
```

```
                { "Artist", song.artist },
                { "Album", song.album },
                { "Length", song.length },
                { "Path", song.path }
            };
                doc_ref.SetAsync(s);

                if (doc_ref.GetSnapshotAsync().Result.Exists)
                    count++;
            }
            if (count == songs.Count)
                return true;
            else
                return false;
        }
        public static bool DeleteSong(Song song)
        {
            return DeleteSongs(new List<Song> { song });
        }
        public static bool DeleteSongs(List<Song> songs)
        {
            int count = 0;

            foreach (var song in songs)
            {
                doc_ref = db.Collection("UserPlaylists/" +
Profile.Id_user.ToString() + "/Songs").Document(song.Id_song.ToString());
                doc_ref.DeleteAsync();
                if (doc_ref.GetSnapshotAsync().Result.Exists)
                    count++;
            }
            if (count == songs.Count)
                return true;
            else
                return false;
        }
    }
}
```

Файл *Database.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Configuration;
using System.Data.SqlClient;

namespace DCO_Player
{
    static class Database
    {
        static string connectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;

        public static SqlDataReader GetUser(string login)
        {
            string sqlExpression = "SELECT * FROM Users WHERE Login =
@login";
            SqlConnection connection = new SqlConnection(connectionString);
            connection.Open();
```

```
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            command.Parameters.Add(new SqlParameter("@login", login));
            return command.ExecuteReader();
        }

        public static void InsertUser()
        {
            string sqlExpression = "INSERT INTO Users (Id_user, Name,
Surname, Login, Password, Create_date, User_image_source, Subscription_date,
N_GB, GB_date) VALUES" +
                    " (@Id_user, @Name, @Surname, @Login, @Password,
@Create_date, @User_image_source, @Subscription_date, @N_GB, @GB_date)";

            using (SqlConnection connection = new
SqlConnection(connectionString))
            {
                connection.Open();
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                command.Parameters.Add(new SqlParameter("@Id_user",
Profile.Id_user));
                command.Parameters.Add(new SqlParameter("@Name",
Profile.name));
                command.Parameters.Add(new SqlParameter("@Surname",
Profile.surname));
                command.Parameters.Add(new SqlParameter("@Login",
Profile.login));
                command.Parameters.Add(new SqlParameter("@Password",
Profile.password));
                command.Parameters.Add(new SqlParameter("@Create_date",
Profile.createDate.ToString("d")));
                command.Parameters.Add(new SqlParameter("@User_image_source",
Profile.imageSrc));
                command.Parameters.Add(new SqlParameter("@Subscription_date",
DateTime.MinValue.ToString("d")));
                command.Parameters.Add(new SqlParameter("@N_GB", "0"));
                command.Parameters.Add(new SqlParameter("@GB_date",
DateTime.MinValue.ToString("d")));
                command.ExecuteNonQuery();
                connection.Close();
            }
        }

        public static bool UpdateUser()
        {
            bool db = true;
            string sqlExpression = "UPDATE Users SET Subscription_date =
@Subscription_date, N_GB = @N_GB, GB_date = @GB_date " +
                "WHERE Id_user = @Id_user";
            try
            {
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    SqlCommand command = new SqlCommand(sqlExpression,
connection);
                    command.Parameters.Add(new
SqlParameter("@Subscription_date",Profile.subscriptionDate));
                    command.Parameters.Add(new SqlParameter("@N_GB",
Profile.gbs));
```

```
                    command.Parameters.Add(new SqlParameter("@GB_date",
Profile.GBDate));
                    command.Parameters.Add(new SqlParameter("@Id_user",
Profile.Id_user));
                    command.ExecuteNonQuery();
                    connection.Close();
                }
            }
            catch
            {
                db = false;
            }
            return db;
        }

        public static (bool, List<RadioStation>) GetRadio()
        {
            List<RadioStation> radioStations = new List<RadioStation>();
            bool db;
            string sqlExpression = "SELECT * FROM Radio";
            try
            {
                SqlConnection connection = new
SqlConnection(connectionString);
                connection.Open();
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                SqlDataReader reader = command.ExecuteReader();

                if (reader.HasRows)
                {
                    while (reader.Read())
                    {
                        RadioStation station = new RadioStation();
                        station.Id = (int)reader.GetValue(0);
                        station.name = reader.GetValue(1).ToString();
                        station.descr = reader.GetValue(2).ToString();
                        station.stream = reader.GetValue(3).ToString();
                        station.page = reader.GetValue(4).ToString();
                        station.imageSrc = reader.GetValue(5).ToString();

                        radioStations.Add(station);
                    }
                }
                db = true;
                reader.Close();
                connection.Close();
            }
            catch
            {
                db = false;
            }
            return (db, radioStations);
        }

        public static void InsertRadio(List<RadioStation> radioStations)
        {
            string sqlExpression = "INSERT INTO Radio (Id_Radio, Radio,
Description, Stream, Page, Radio_image_source) VALUES" +
                    " (@Id_Radio, @Radio, @Description, @Stream, @Page,
@Radio_image_source)";
```

```
            using (SqlConnection connection = new
SqlConnection(connectionString))
            {
                connection.Open();
                foreach (var station in radioStations)
                {
                    SqlCommand command = new SqlCommand(sqlExpression,
connection);
                    command.Parameters.Add(new SqlParameter("@Id_Radio",
station.Id));
                    command.Parameters.Add(new SqlParameter("@Radio",
station.name));
                    command.Parameters.Add(new SqlParameter("@Description",
station.descr));
                    command.Parameters.Add(new SqlParameter("@Stream",
station.stream));
                    command.Parameters.Add(new SqlParameter("@Page",
station.page));
                    command.Parameters.Add(new
SqlParameter("@Radio_image_source", station.imageSrc));
                    command.ExecuteNonQuery();
                }
                connection.Close();
            }
        }

        public static bool DeleteRadio(string name)
        {
            try
            {
                string sqlExpression = "DELETE FROM Radio WHERE Radio.Radio =
@name";
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    SqlCommand command = new SqlCommand(sqlExpression,
connection);
                    command.Parameters.Add(new SqlParameter("@name", name));
                    command.ExecuteNonQuery();
                    connection.Close();
                }
            }
            catch
            {
                return false;
            }
            return true;
        }

        public static (bool, List<UserPlaylist>) GetPlaylists(bool songs =
true, bool deleted = false)
        {
            List<UserPlaylist> playlists = new List<UserPlaylist>();
            bool db;
            string sqlExpression = "SELECT * FROM Playlists WHERE
Playlists.Id_user = @Id_user"; // Делаем запрос к плейлистам
            try
            {
                SqlConnection connection = new
SqlConnection(connectionString);
                connection.Open();
```

```
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                command.Parameters.Add(new SqlParameter("@Id_user",
Profile.Id_user));
                SqlDataReader reader = command.ExecuteReader();

                if (reader.HasRows) // если есть данные
                {
                    while (reader.Read())
                    {
                        UserPlaylist playlist = new UserPlaylist();

                        playlist.Id_user = (Guid)reader.GetValue(0);
                        playlist.Id_playlist = (Guid)reader.GetValue(1);
                        playlist.name = reader.GetValue(2).ToString();
                        if (playlist.name == UserPlaylist.NULL_NAME &&
!deleted)
                            continue;
                        playlist.lastUpdate = (DateTime)reader.GetValue(3);
                        if (reader.GetValue(4) == DBNull.Value)
                            playlist.lastSync = DateTime.MinValue;
                        else
                            playlist.lastSync = (DateTime)reader.GetValue(4);
                        playlist.imageSrc = reader.GetValue(5).ToString();

                        playlists.Add(playlist);
                    }
                }
                db = true;
                reader.Close();
                connection.Close();
                if (songs)
                {
                    int i = 0;
                    bool pl;
                    while (i < playlists.Count)
                    {
                        (pl, playlists[i].songs) =
GetSongsFromPlaylist(playlists[i].Id_playlist);
                        if (!pl)
                            db = false;
                        i++;
                    }
                }
            }
            catch
            {
                db = false;
            }
            return (db, playlists);
        }

        public static bool InsertPlaylist(UserPlaylist playlist)
        {
            return InsertPlaylists(new List<UserPlaylist> { playlist });
        }

        public static bool InsertPlaylists(List<UserPlaylist> playlists)
        {
            bool db;
            List<UserPlaylist> exist_playlists = new List<UserPlaylist>(); ;
            (db, exist_playlists) = GetPlaylists();
```

```
            if (!db)
                return false;
            var ids = exist_playlists.Select(f => f.Id_playlist).ToList();

            string sqlExpression = "INSERT INTO Playlists (Id_user,
Id_playlist, Name, Last_update, Last_sync, Playlist_image_source) VALUES" +
                    " (@Id_user, @Id_playlist, @Name, @Last_update,
@Last_sync, @Playlist_image_source)";
            try
            {
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    foreach (var playlist in playlists)
                    {
                        if (ids.Contains(playlist.Id_playlist))
                            continue;
                        SqlCommand command = new SqlCommand(sqlExpression,
connection);
                        command.Parameters.Add(new SqlParameter("@Id_user",
playlist.Id_user));
                        command.Parameters.Add(new
SqlParameter("@Id_playlist", playlist.Id_playlist));
                        command.Parameters.Add(new SqlParameter("@Name",
playlist.name));
                        command.Parameters.Add(new
SqlParameter("@Last_update", playlist.lastUpdate));
                        if (playlist.lastSync == DateTime.MinValue)
                            command.Parameters.Add(new
SqlParameter("@Last_sync", DBNull.Value));
                        else
                            command.Parameters.Add(new
SqlParameter("@Last_sync", playlist.lastSync));
                        command.Parameters.Add(new
SqlParameter("@Playlist_image_source", playlist.imageSrc));
                        command.ExecuteNonQuery();
                    }
                    connection.Close();
                }
                foreach (var playlist in playlists)
                {
                    if (!InsertSongsToPlaylist(playlist.songs,
playlist.Id_playlist))
                        db = false;
                }
            }
            catch
            {
                db = false;
            }
            return db;
        }

        public static bool UpdatePlaylist(UserPlaylist playlist, List<Song>
songs = null, bool s = false)
        {
            bool db = true;
            string sqlExpression = "UPDATE Playlists SET Last_sync =
@Last_sync, Last_update = @Last_update " +
                "Where Playlists.Id_playlist = @Id_playlist";
            try
```

```
            {
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    SqlCommand command = new SqlCommand(sqlExpression,
connection);
                    command.Parameters.Add(new SqlParameter("@Id_playlist",
playlist.Id_playlist));
                    if (playlist.lastSync == DateTime.MinValue)
                        command.Parameters.Add(new SqlParameter("@Last_sync",
DBNull.Value));
                    else
                        command.Parameters.Add(new SqlParameter("@Last_sync",
playlist.lastSync));
                    command.Parameters.Add(new SqlParameter("@Last_update",
playlist.lastUpdate));
                    command.ExecuteNonQuery();
                    connection.Close();
                }
                if (s && songs != null)
                    if (!InsertSongsToPlaylist(songs, playlist.Id_playlist))
                        db = false;
                if (s && songs == null)
                    if (!InsertAllSongsToPlaylist(playlist))
                        db = false;
            }
            catch
            {
                db = false;
            }
            return db;
        }

        public static bool DeletePlaylist(UserPlaylist playlist, bool
NULL_name=true)
        {
            string sqlExpression;
            bool deleted = true;
            try
            {
                sqlExpression = "DELETE FROM Playlist_songs WHERE
Playlist_songs.Id_playlist = @Id_playlist";
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    SqlCommand command = new SqlCommand(sqlExpression,
connection);
                    command.Parameters.Add(new SqlParameter("@Id_playlist",
playlist.Id_playlist));
                    command.ExecuteNonQuery();
                    connection.Close();
                }
                if (!NULL_name || playlist.lastSync == DateTime.MinValue)
                {

                    sqlExpression = "DELETE FROM Playlists WHERE
Playlists.Id_playlist = @Id_playlist";
                    using (SqlConnection connection = new
SqlConnection(connectionString))
                    {
```

```
                        connection.Open();
                        SqlCommand command = new SqlCommand(sqlExpression,
connection);
                        command.Parameters.Add(new
SqlParameter("@Id_playlist", playlist.Id_playlist));
                        command.ExecuteNonQuery();
                        connection.Close();
                    }
                }
                else
                {
                    sqlExpression = "UPDATE Playlists SET Name = @NULL_NAME
Where Playlists.Id_playlist = @Id_playlist";
                    using (SqlConnection connection = new
SqlConnection(connectionString))
                    {
                        connection.Open();
                        SqlCommand command = new SqlCommand(sqlExpression,
connection);
                        command.Parameters.Add(new SqlParameter("@NULL_NAME",
UserPlaylist.NULL_NAME));
                        command.Parameters.Add(new
SqlParameter("@Id_playlist", playlist.Id_playlist));
                        command.ExecuteNonQuery();
                        connection.Close();
                    }
                }
            }
            catch
            {
                deleted = false;
            }
            return deleted;
        }

        public static (bool, List<Song>) GetSongsFromPlaylist(Guid
Id_playlist)
        {
            List<Song> songs = new List<Song>();
            bool db;
            string sqlExpression = "SELECT * FROM Songs, Playlist_songs WHERE
Playlist_songs.Id_song = Songs.Id_song  and Playlist_songs.Id_playlist =
@Id_playlist"; // Делаем запрос к исполнителям
            try
            {
                SqlConnection connection = new
SqlConnection(connectionString);
                connection.Open();
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                command.Parameters.Add(new SqlParameter("@Id_playlist",
Id_playlist));
                SqlDataReader reader = command.ExecuteReader();

                if (reader.HasRows) // если есть данные
                {
                    while (reader.Read())
                    {
                        Song song = new Song();

                        song.Id_song = (Guid)reader.GetValue(0);
                        song.is_local = (bool)reader.GetValue(1);
```

```
                        song.full_name = reader.GetValue(2).ToString();
                        song.name = reader.GetValue(3).ToString();
                        song.artist = reader.GetValue(4).ToString();
                        song.album = reader.GetValue(5).ToString();
                        song.length = (int)reader.GetValue(6);
                        song.path = reader.GetValue(7).ToString();
                        song.n_sequence = (int)reader.GetValue(10);

                        songs.Add(song);
                    }
                }
                db = true;
                reader.Close();
                connection.Close();
                songs = songs.OrderByDescending(o => o.n_sequence).ToList();
            }
            catch
            {
                db = false;
            }
            return (db, songs);
        }

        public static bool InsertSongToPlaylist(Song song, Guid Id_playlist)
        {
            return InsertSongsToPlaylist(new List<Song> { song },
Id_playlist);
        }
        public static bool InsertSongsToPlaylist(List<Song> songs, Guid
Id_playlist)
        {
            bool db = true;
            List<Song> exist_songs;
            (db, exist_songs) = GetSongsFromPlaylist(Id_playlist);
            if (!db)
                return false;
            int max = 0;
            foreach (var s in exist_songs)
                if (s.n_sequence > max)
                    max = s.n_sequence;

            string sqlExpression;
            try
            {
                sqlExpression = "INSERT INTO Playlist_songs (Id_playlist,
Id_song, N_sequence) " +
                    "VALUES (@Id_playlist, @Id_song, @N_sequence)";
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    foreach (var song in songs)
                    {
                        SqlCommand command = new SqlCommand(sqlExpression,
connection);
                        command.Parameters.Add(new
SqlParameter("@Id_playlist", Id_playlist));
                        command.Parameters.Add(new SqlParameter("@Id_song",
song.Id_song));
                        command.Parameters.Add(new
SqlParameter("@N_sequence", ++max));
                        command.ExecuteNonQuery();
```

```
                }
                connection.Close();
            }
        }
        catch
        {
            db = false;
        }
        return db;
    }

    public static bool InsertAllSongsToPlaylist(UserPlaylist playlist)
    {
        bool db = true;

        string sqlExpression;
        try
        {
            sqlExpression = "DELETE FROM Playlist_songs WHERE
Playlist_songs.Id_playlist = @Id_playlist";
            using (SqlConnection connection = new
SqlConnection(connectionString))
            {
                connection.Open();
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                command.Parameters.Add(new SqlParameter("@Id_playlist",
playlist.Id_playlist));
                command.ExecuteNonQuery();
                connection.Close();
            }

            sqlExpression = "INSERT INTO Playlist_songs (Id_playlist,
Id_song, N_sequence) " +
                "VALUES (@Id_playlist, @Id_song, @N_sequence)";
            using (SqlConnection connection = new
SqlConnection(connectionString))
            {
                connection.Open();
                foreach (var song in playlist.songs)
                {
                    SqlCommand command = new SqlCommand(sqlExpression,
connection);
                    command.Parameters.Add(new
SqlParameter("@Id_playlist", playlist.Id_playlist));
                    command.Parameters.Add(new SqlParameter("@Id_song",
song.Id_song));
                    command.Parameters.Add(new
SqlParameter("@N_sequence", song.n_sequence));
                    command.ExecuteNonQuery();
                }
                connection.Close();
            }
        }
        catch
        {
            db = false;
        }
        return db;
    }

    public static (bool, List<Song>) GetAllSongs()
```

```
        {
            List<Song> songs = new List<Song>();
            bool db;
            string sqlExpression = "SELECT * FROM Songs";
            try
            {
                SqlConnection connection = new
SqlConnection(connectionString);
                connection.Open();
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                SqlDataReader reader = command.ExecuteReader();

                if (reader.HasRows)
                {
                    while (reader.Read())
                    {
                        Song song = new Song();
                        song.Id_song = (Guid)reader.GetValue(0);
                        song.is_local =
Convert.ToBoolean(reader.GetValue(1));
                        song.full_name = reader.GetValue(2).ToString();
                        song.name = reader.GetValue(3).ToString();
                        song.artist = reader.GetValue(4).ToString();
                        song.album = reader.GetValue(5).ToString();
                        song.length = (int)reader.GetValue(6);
                        song.path = reader.GetValue(7).ToString();

                        songs.Add(song);
                    }
                }
                db = true;
                reader.Close();
                connection.Close();
            }
            catch
            {
                db = false;
            }
            return (db, songs);
        }

        public static (bool, List<Tuple<Guid, string>>) GetSongsPath()
        {
            bool db;
            List<Tuple<Guid, string>> paths = new List<Tuple<Guid,
string>>();
            string sqlExpression = "SELECT Id_song, Path FROM Songs";
            try
            {
                SqlConnection connection = new
SqlConnection(connectionString);
                connection.Open();
                SqlCommand command = new SqlCommand(sqlExpression,
connection);
                SqlDataReader reader = command.ExecuteReader();
                if (reader.HasRows)
                    while (reader.Read())
                        paths.Add(new Tuple<Guid, string>
((Guid)reader.GetValue(0), reader.GetValue(1).ToString()));
                reader.Close();
                connection.Close();
```

```
                db = true;
            }
            catch { db = false; }
            return (db, paths);
        }

        public static bool InsertSong(Song song)
        {
            return InsertSongs(new List<Song> { song });
        }

        public static bool InsertSongs(List<Song> songs)
        {
            bool db = true;
            string sqlExpression = "INSERT INTO Songs (Id_song, Is_local,
Full_name, Name, Artist, Album, Length, Path) VALUES" +
                    " (@Id_song, '1', @Full_name, @Name, @Artist, @Album,
@Length, @Path)";
            try
            {
                using (SqlConnection connection = new
SqlConnection(connectionString))
                {
                    connection.Open();
                    foreach (var song in songs)
                    {
                        SqlCommand command = new SqlCommand(sqlExpression,
connection);
                        command.Parameters.Add(new SqlParameter("@Id_song",
song.Id_song));
                        //  command.Parameters.Add(new
SqlParameter("@Is_local", song.is_local));
                        command.Parameters.Add(new SqlParameter("@Full_name",
song.full_name));
                        command.Parameters.Add(new SqlParameter("@Name",
song.name));
                        command.Parameters.Add(new SqlParameter("@Artist",
song.artist));
                        command.Parameters.Add(new SqlParameter("@Album",
song.album));
                        command.Parameters.Add(new SqlParameter("@Length",
song.length));
                        command.Parameters.Add(new SqlParameter("@Path",
song.path));
                        command.ExecuteNonQuery();
                    }
                    connection.Close();
                }
            }
            catch { db = false; }
            return db;
        }

    }
}
```

Файл *Sing_In.xaml.cs*

```
using System;
using System.Configuration;
using System.Text.RegularExpressions;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Data.SqlClient;


namespace DCO_Player
{
    /// <summary>
    /// Логика взаимодействия для Log_In.xaml
    /// </summary>
    public partial class Log_In : Page
    {
        string login { get; set; }
        string password { get; set; }

        Regex RLogin = new Regex(@"(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})");
        Regex RPassword = new Regex(@"(?=.*[0-9])(?=.*[!@#$%^&*])(?=.*[a-
z])(?=.*[A-Z])[0-9a-zA-Z!@#$%^&*]{9,}");
        Regex RPassword = new Regex(@"[A-Za-z\d@$!%_*\-#?&]{8,}$");

        bool BLogin = false;
        bool BPassword = false;

        public Log_In()
        {
            InitializeComponent();
        }

        private void Log_In_Click(object sender, RoutedEventArgs e)
        {
            bool load = false;
            if (RLogin.IsMatch(Login.Text))
            {
                BLogin = true;
                login = Login.Text;
            }
            else
            {
                MessageBox.Show("Поле должно содержать правильное имя
почты");
            }

            if (RPassword.IsMatch(CPassword.Password))
            {
                BPassword = true;
                password = CPassword.Password;
            }
            else
            {
                MessageBox.Show("Пароль не соответсвует правилам пароля");
            }

            if (BLogin && BPassword)
            {
                Firebase.Init();
                try
                {
                    SqlDataReader reader = Database.GetUser(login);

                    if (reader.HasRows) // если есть данные
                    {
                        reader.Read();
                        if (reader.GetValue(4).ToString() == password)
```

```
                        {
                            Profile.Id_user = (Guid)reader.GetValue(0);
                            Profile.name = reader.GetValue(1).ToString();
                            Profile.surname = reader.GetValue(2).ToString();
                            Profile.createDate =
(DateTime)reader.GetValue(5);
                            Profile.imageSrc = reader.GetValue(6).ToString();
                            Profile.subscriptionDate =
(DateTime)reader.GetValue(7);
                            Profile.gbs = (int)reader.GetValue(8);
                            Profile.GBDate = (DateTime)reader.GetValue(9);
                        }
                        else
                        {
                            MessageBox.Show("Неправильный пароль!");
                            load = false;
                        }

                        reader.Close();
                        load = true;
                    }
                    else
                    {
                        if (Firebase.CheckUser(login))
                        {
                            if (Firebase.GetUser(login))
                                Database.InsertUser();
                            if (password == Profile.password)
                                load = true;
                            else
                            {
                                MessageBox.Show("Неправильный пароль!");
                                load = false;
                            }
                        }
                        else
                        {
                            MessageBox.Show("Такого пользователя нет!");
                            load = false;
                        }
                    }

                }
                catch
                {
                    MessageBox.Show("Не получается загрузить пользователя.
\n" +
                        "Отсутствует подключение к базе данных и интернету");
                    load = false;
                }
            }


            if (load)
            {
                MainWindow mainWindow = new MainWindow();
                mainWindow.Show();
                Start.Instance.Close();
            }
        }
    }
}
```

Файл *Sing_Up.xaml.cs*

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Data.SqlClient;
using System.Configuration;
using System.Text.RegularExpressions;
using Microsoft.Win32;
using System.Windows.Media.Imaging;

namespace DCO_Player
{
    /// <summary>
    /// Логика взаимодействия для Sign_Up.xaml
    /// </summary>
    public partial class Sign_Up : Page
    {
        Guid Id_user = Guid.NewGuid();
        string name { get; set; }
        string surname { get; set; }
        string login { get; set; }
        string password { get; set; }
        string imageSrc { get; set; }

        Regex RName = new Regex("^([A-я ]|[A-z ]){1,19}$");
        Regex RSurname = new Regex("^([A-я ]|[A-z ]){1,19}$");
        Regex RLogin = new Regex(@"(\w+@[a-z_]+?\.[a-z]{2,6})");
        Regex RPassword = new Regex(@"(?=.*[A-Za-
z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!%*#?&]{8,}$");

        bool BName = false;
        bool BSurname = false;
        bool BLogin = false;
        bool BPassword = false;
        bool load = true;

        CroppedBitmap cb;

        public Sign_Up()
        {
            InitializeComponent();
        }

        private void Sign_Up_Click(object sender, RoutedEventArgs e)
        {
            BName = false;
            BSurname = false;
            BLogin = false;
            BPassword = false;
            load = true;

            if (RName.IsMatch(Name.Text))
            {
                BName = true;
                name = Name.Text;
            }
            else
            {
                MessageBox.Show("Поле Name должно содержать буквы
кириллического либо латинского алфавита");
```

```
                return;
            }

            if (RSurname.IsMatch(Surname.Text))
            {
                BSurname = true;
                surname = Surname.Text;
            }
            else
            {
                MessageBox.Show("Поле Surname должно содержать буквы
кириллического либо латинского алфавита");
                return;
            }

            if (RLogin.IsMatch(Login.Text))
            {
                BLogin = true;
                login = Login.Text;
            }
            else
            {
                MessageBox.Show("Поле login должно содержать имя почты");
                return;
            }

            if (RPassword.IsMatch(CPassword.Password))
            {
                if (CPassword.Password == Password.Password)
                {
                    BPassword = true;
                    password = Password.Password;
                }
                else
                {
                    MessageBox.Show("Пароли не совпадают");
                    return;
                }
            }
            else
            {
                MessageBox.Show("Пароль должен содержать не менее 8 символов,
включать спецсимволы, загланые буквы, числа");
                return;
            }

            if (BName && BSurname && BLogin && BPassword)
            {
                Firebase.Init();

                if (cb != null)
                {
                    BitmapEncoder encoder = new PngBitmapEncoder(); //
Создаем новый образ кодировщика
                    encoder.Frames.Add(BitmapFrame.Create(cb)); // кодируем
наше обрезанное изображение в png и далее ниже его сохраняем

                    using (var fileStream = new
System.IO.FileStream(Environment.CurrentDirectory +
"/Images/Profiles/Avatar/" + Name.Text + "_" + Surname.Text + ".png",
System.IO.FileMode.Create))
                    {
```

```
                    encoder.Save(fileStream);
                }

                imageSrc = "/Images/Profiles/Avatar/" + Name.Text + "_" +
Surname.Text + ".png"; // Ссылка на аватар
            }
            else
            {
                imageSrc = "";
            }

            try
            {
                SqlDataReader reader = Database.GetUser(login);
                if (reader.HasRows || Firebase.CheckUser(login)) // если
есть данные
                {
                    load = false;
                    throw new Exception();
                }
                reader.Close();

                Profile.Id_user = Id_user;
                Profile.name = name;
                Profile.login = login;
                Profile.password = password;
                Profile.surname = surname;
                Profile.createDate = DateTime.Today;
                Profile.imageSrc = imageSrc;
                Profile.subscriptionDate = DateTime.MinValue;
                Profile.gbs = 0;
                Profile.GBDate = DateTime.MinValue;

                Database.InsertUser();
                Firebase.AddUser();
            }
            catch
            {
                if (load == false)
                    MessageBox.Show("Пользователь " + login + "уже
существует");
                else
                    MessageBox.Show("Не удается зарегестрировать
пользователя. \n" +
                            "Отсутствует подключение к базе данных");
                load = false;
            }
        }

        if (load)
        {
            MainWindow mainWindow = new MainWindow();
            mainWindow.Show();
            Start.Instance.Close();
        }

    }

    private void Photo_Click(object sender, RoutedEventArgs e)
    {
        try
        {
```

```
                OpenFileDialog openFileDialog = new OpenFileDialog();
                openFileDialog.Filter = "All supported
graphics|*.jpg;*.jpeg;*.png|JPEG (*.jpg;*.jpeg)|*.jpg;*.jpeg|Portable Network
Graphic (*.png)|*.png";
                if (openFileDialog.ShowDialog() == true)
                {
                    Uri uri = new Uri(openFileDialog.FileName); // Получаем
ссылку на файл (картинку)

                    System.Windows.Controls.Image croppedImage = new
System.Windows.Controls.Image();
                    BitmapImage bm = new BitmapImage(uri); // Создаем новый
образ битового изображения
                    cb = new CroppedBitmap(
                       bm,
                       new Int32Rect((int)(((int)bm.PixelWidth - 600) / 2),
(int)(((int)bm.PixelHeight - 600) / 2), 600, 600));       // Выбираем
настройки обрезки

                    Photo.Background = new ImageBrush(cb);
                }
            }
            catch
            {
                MessageBox.Show("Изображение должно быть 600x600 пикселей");
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Д
(обязательное)

Ведомость документов