DOI 10.37882/2223-2966.2024.11.25

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ АВТОМАТИЗАЦИИ ПЕРЕНОСА МУЗЫКАЛЬНОЙ БИБЛИОТЕКИ МЕЖДУ СЕРВИСАМИ

DEVELOPMENT OF AN APPLICATION FOR AUTOMATING THE TRANSFER OF MUSIC LIBRARIES BETWEEN SERVICES

A. Pantykhin
V. Gladun
I. Malinin
S. Molodyakov

Summary. The article describes the development of an application for automating the transfer of music libraries between different streaming services. The focus is on selecting a system architecture, which is a client-server application using the Swift language and the Spring framework. A comparison algorithm for audio tracks was developed and tested, considering the specific formatting of titles and artists in various services. Performance and functionality tests were conducted.

Keywords: data deduplication, hash function, data storage system, storage optimization, MongoDB, Python, MongoEngine.

Пантюхин Андрей Максимович
Санкт-Петербургский политехнический университет Петра Великого
panandafog@gmail.com
Гладун Владимир Вадимович
Санкт-Петербургский политехнический университет Петра Великого
vladimir.gldn@gmail.com
Малинин Илья Игоревич
Санкт-Петербургский политехнический университет Петра Великого
malinin.ilja@gmail.com
Молодяков Сергей Александрович
Доктор технических наук, профессор, Санкт-Петербургский политехнический университет Петра Великого
molodyakov_sa@spbstu.ru

Аннотация. В статье описывается разработка приложения для автоматизации переноса музыкальных библиотек между различными стриминговыми сервисами. Основное внимание уделено выбору архитектуры системы, которая представляет собой клиент-серверное приложение с использованием языка Swift и фреймворка Spring. Разработан и протестирован алгоритм сравнения аудиозаписей, который учитывает специфику оформления названий и исполнителей в разных сервисах. Проведены тесты производительности и функциональности.

Ключевые слова: автоматизация переноса музыки, стриминговый сервис, алгоритм сравнения, клиент-серверное приложение, Swift, Spring, JWT.

Введение

В последние годы наблюдается значительный рост популярности стриминговых сервисов для прослушивания музыки, которые вытесняют локальное хранение музыки на устройствах. Об этом свидетельствует статистика доходности сфер распространения музыки и рост количества пользователей стриминговых платформ, представленная на рис. 1 [1].

Однако переход на новый сервис требует переноса музыкальной библиотеки, что создает для пользователей неудобства, особенно если библиотека большая. Так как основной платформой для использования музыкальных стриминговых сервисов являются мобильные устройства [2], было первоначально разрабатывать сервис именно для них. Было принято решение начать с разработки приложения для iOS ввиду более высокой платежеспособности пользователей, чем у Android [2].

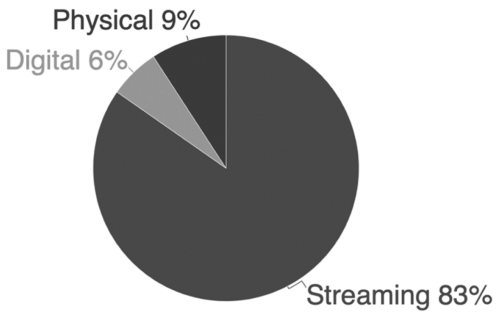

Рис. 1. Доходность сфер распространения музыки

Анализ существующих решений

Во время разработки проекта значительно увеличилась конкуренция среди приложений, аналогичных нашему, особенно на платформе iOS. Ранние конкуренты, такие как «SpotiApp» [3], имели существенные недостатки, например, использовали распознавание названий

песен на скриншотах вместо специализированного API, что делало перенос музыкальной библиотеки медленным и неудобным для пользователей. Во время разработки собственного приложения появились более продвинутые приложения-конкуренты, такие как «Стащи свою музыку» и «FreeYourMusic» [2], которые поддерживают большее количество сервисов, включая те, которые не имеют открытого API, и позволяют автоматически переносить библиотеки без использования скриншотов.

Для повышения конкурентоспособности разрабатываемого приложения необходимо расширить набор поддерживаемых сервисов, включая те, которые не имеют открытого API, но популярны среди пользователей. Это можно сделать, добавив возможность экспорта и импорта данных через файлы, а также создав собственный сервер для хранения музыкальных библиотек пользователей. Наличие сервера позволит реализовать дополнительные функции, такие как резервное копирование данных и синхронизация истории переносов между разными устройствами пользователя, что значительно повысит удобство использования приложения.

Таким образом, расширение поддерживаемых сервисов и создание собственного сервера для хранения данных пользователей могут стать ключевыми факторами для повышения конкурентоспособности и удобства использования нашего приложения, особенно в условиях растущей конкуренции на рынке.

Архитектура системы

Было решено создать клиент-серверное приложение, где клиентское приложение будет осуществлять перенос музыки между сервисами через сетевое соединение, а серверная часть — хранить музыкальную библиотеку пользователя и историю её переносов. Взаимодействие между клиентом и сервером будет обеспечиваться через веб-сервис.

Для реализации веб-сервиса рассматривались несколько архитектурных подходов, таких как XML-RPC, JSON-RPC, SOAP и REST [4]. В итоге был выбран REST, так как он не привязан к конкретному протоколу, предлагает гибкость в обмене сообщениями и поддерживает форматы JSON и XML.

Применение архитектуры REST в проекте также обусловлено её популярностью и универсальностью, что делает её оптимальным выбором для создания мобильных клиент-серверных приложений, обеспечивая удобство и эффективность взаимодействия между компонентами системы.

Архитектура системы представлена на рис. 2.

Выбор технических средств

Для разработки клиентского приложения под iOS был выбран язык программирования Swift, поскольку он более современный и удобный по сравнению с Objective-C [5]. В качестве основного фреймворка рассматривались UIKit и SwiftUI. UIKit распространен и стабилен, но SwiftUI, будучи кроссплатформенным, позволяет легко адаптировать приложение для iPad и macOS. Для управления зависимостями предпочтение было отдано CocoaPods из-за его популярности и удобства использования [5].

Серверная часть приложения реализована на Java с использованием фреймворка Spring, который обеспечивает архитектурный каркас и упрощает разработку благодаря автоматической конфигурации через Spring Boot [6]. В качестве базы данных выбрана PostgreSQL, известная своей надежностью и расширяемостью. Для реализации безопасности используется Spring Security, который легко интегрируется с Spring и предоставляет удобные инструменты для контроля доступа к API [6].

Для аутентификации запросов было решено использовать метод OAuth с токенами JSON Web Token (JWT) [7], который не требует передачи паролей и позволяет фильтровать запросы из неавторизованных источников. Это решение обеспечит необходимую защиту данных в проекте, не требующем хранения конфиденциальной информации. Для тестовой версии выбран протокол HTTP, но для финальной версии планируется перейти на более безопасный HTTPS с использованием SSL/TLS.

HTTPS обеспечивает шифрование данных и проверку подлинности через SSL-сертификат, что делает его предпочтительным выбором для защиты данных при передаче. Однако стоит учитывать, что HTTPS снижает скорость работы и требует дополнительных затрат на сертификат [8].

Реализация клиентского приложения

Архитектура MVVM была выбрана в качестве базового паттерна для создания клиентского приложения на SwiftUI. Схема архитектуры MVVM представлена на рис. 3 [9].

В этой архитектуре классы View отвечают за интерфейс, ViewModel управляет данными для отображения, а Model содержит бизнес-логику. View реагируют на изменения в данных через аннотации, что обеспечивает гибкость и динамичность интерфейса.

Для упрощения взаимодействия между компонентами приложения были реализованы классы типа Facade, которые обеспечивают связь между ViewModel и API музыкальных сервисов, а также доступ к базе данных. Это

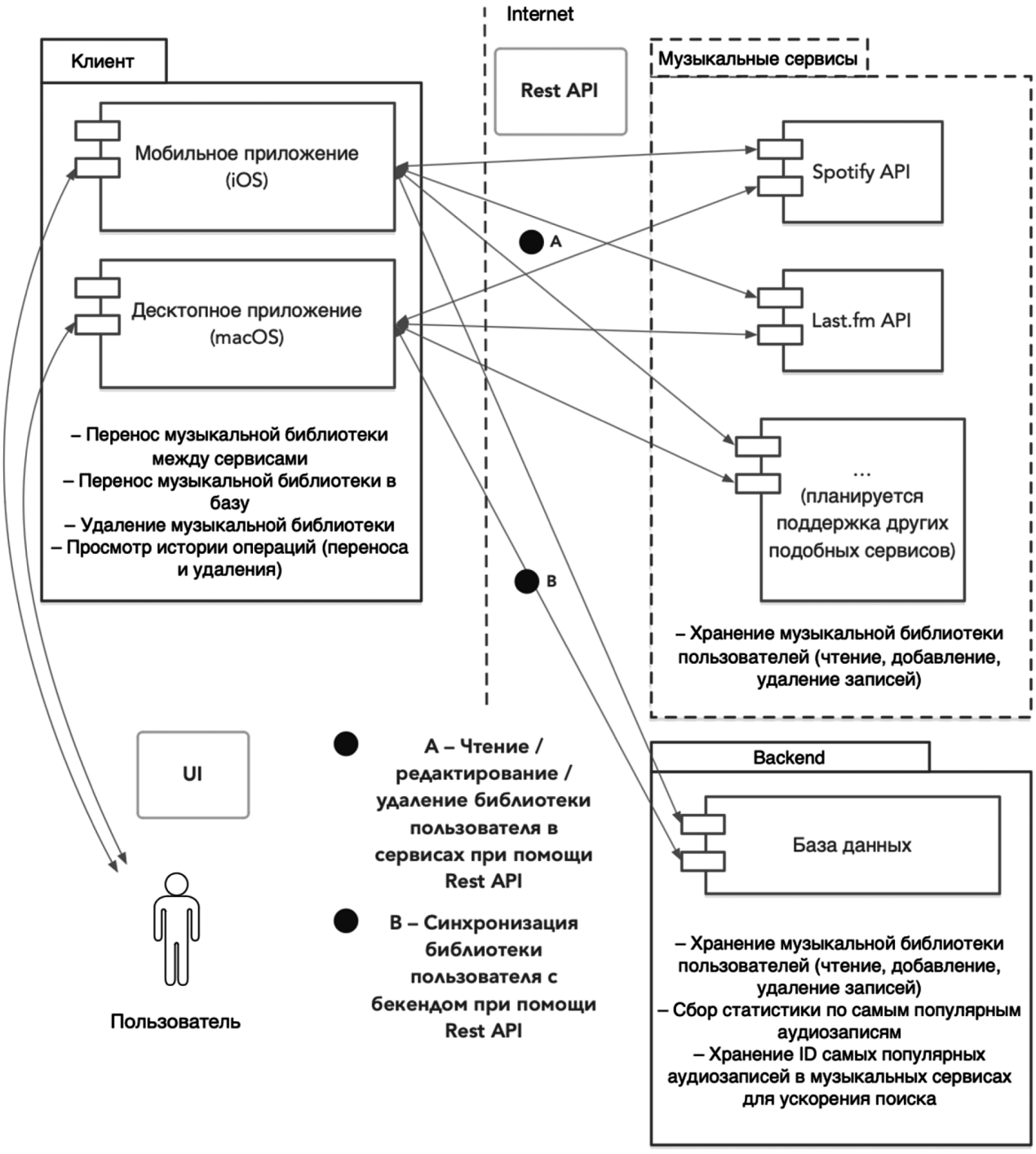

Рис. 2. Архитектура системы

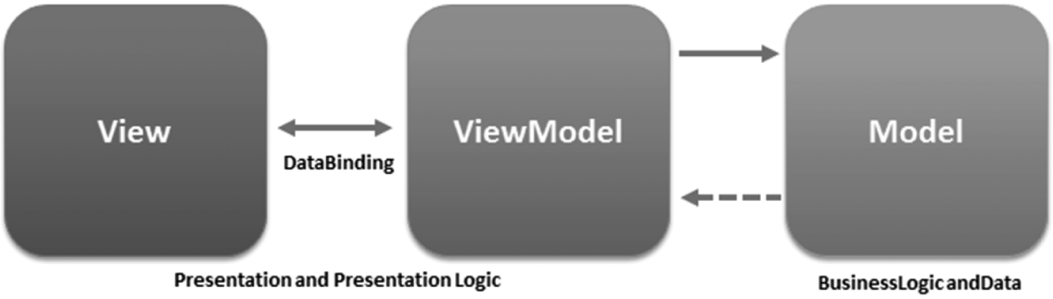

Рис. 3. Архитектура MVVM

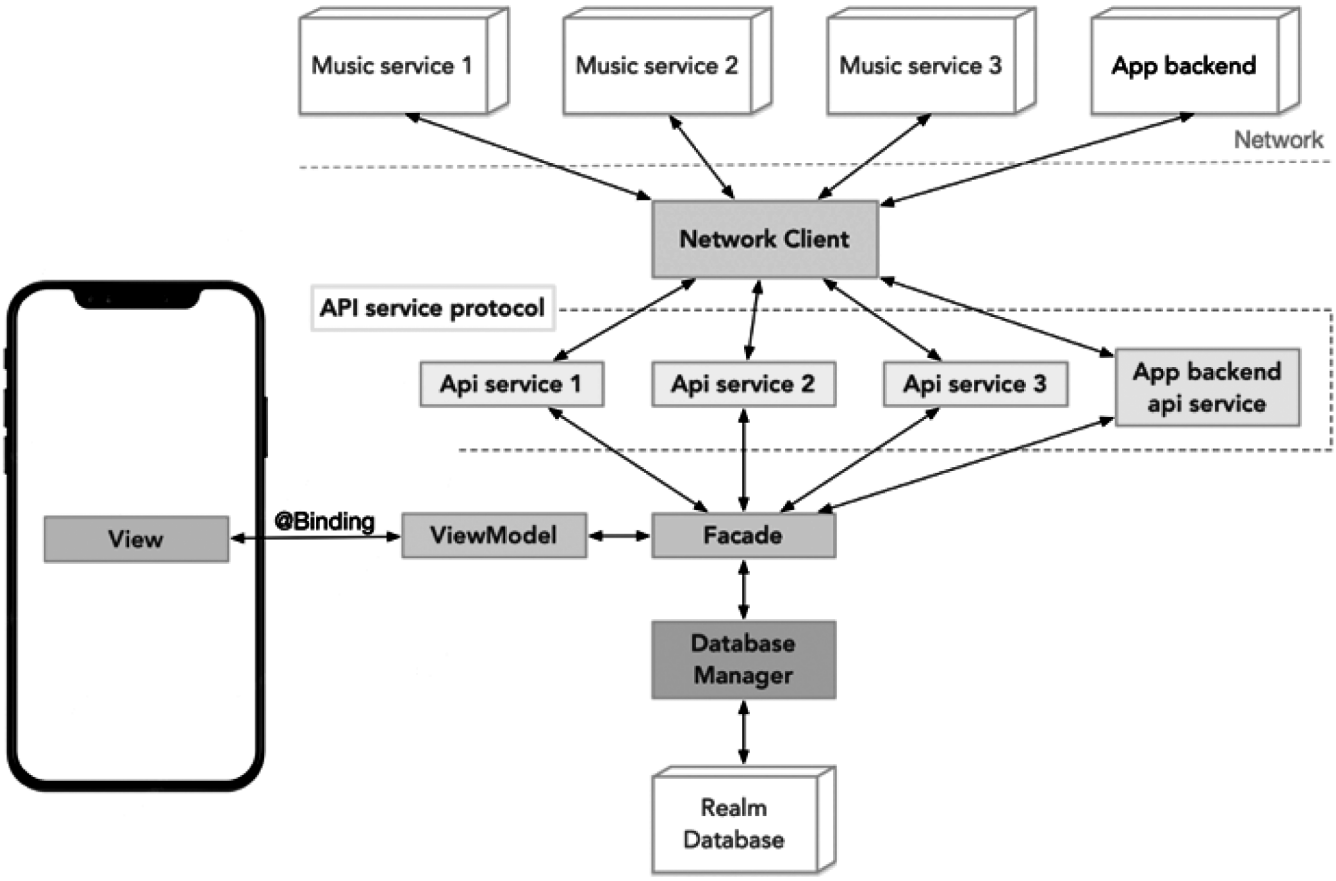

Рис. 4. Архитектура клиентского приложения

решение позволяет легко расширять функциональность приложения, добавляя поддержку новых сервисов и заменяя модули при тестировании, что делает систему гибкой и адаптируемой. Архитектура приложения в упрощенном виде представлена на рис. 4.

Алгоритмы сравнения аудиозаписей

Для сравнения аудиозаписей при переносе музыкальной библиотеки важно использовать подходящий алгоритм. Обычное сравнение строк по названию и исполнителю может быть недостаточным из-за возможных различий в представлении данных о библиотеке в разных сервисах. Поэтому рассмотрены несколько алгоритмов нечеткого сравнения строк, таких как расстояние Левенштейна, алгоритм шинглов и коэффициент Жаккара. Из них наиболее подходящим для данной задачи является алгоритм Левенштейна, который оценивает количество операций, необходимых для преобразования одной строки в другую.

Однако, учитывая разнообразие потребностей пользователей, был разработан собственный алгоритм сравнения, который учитывает специфику оформления названий и исполнителей аудиозаписей в различных сервисах. Этот алгоритм включает в себя несколько шагов: выделение основного названия аудиозаписи, сравнение названий, проверка наличия исполнителей и сравнение длительности треков.

Процесс сравнения начинается с нормализации строк: приведения текста к нижнему регистру и удаления лишних символов. Затем с помощью регулярных выражений выделяется основное название трека, без дополнительных данных в скобках. После этого выполняется сравнение названий и проверка наличия исполнителей в обоих треках. Если исполнители совпадают, проверяется длительность аудиозаписей с учетом допустимого отклонения в 10 %.

Алгоритм включает следующие шаги: 1) выделение названия без дополнительной информации, 2) сравнение названий, 3) проверка наличия исполнителей в обоих аудиозаписях, 4) повторная проверка, поменяв аудиозаписи местами, 5) сравнение длительности аудиозаписей с учетом допустимого отклонения, 6) возврат положительного результата при успешном завершении всех этапов. Такой подход позволяет учесть все возможные нюансы при переносе музыкальных библиотек между различными стриминговыми сервисами.

Этот алгоритм обеспечивает более точное и гибкое сравнение аудиозаписей, позволяя пользователям с минимальными усилиями переносить свои библиотеки

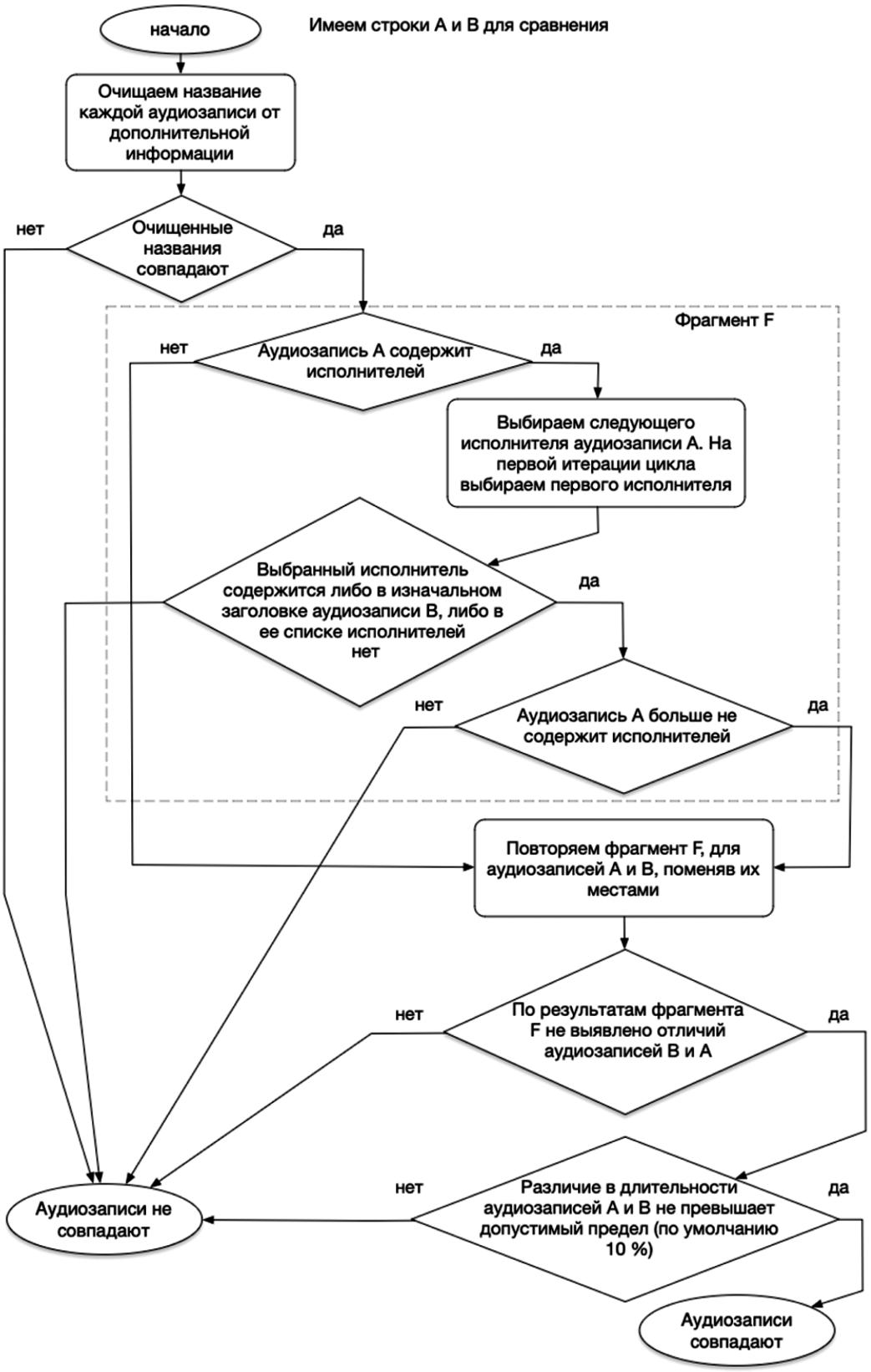

Рис. 5. Алгоритм сравнения аудиозаписей

между сервисами, избегая ошибок и несоответствий. Алгоритм представлен в виде блок-схемы на рис. 5.

Реализация серверной части приложения

Для реализации серверного приложения с использованием Spring были применены стандартные шаблоны проектирования: Controller, Service, Repository и Entity. Эти шаблоны обеспечивают эффективное описание RestAPI, взаимодействие с базой данных, а также позволяют задавать структуру базы данных и связи между таблицами. Классы Controller отвечают за обработку Rest-запросов, Repository обеспечивает доступ к данным без необходимости писать SQL-запросы, а Entity-классы отражают структуру таблиц базы данных и их взаимосвязи.

В качестве базы данных используется PostgreSQL, развернутая в двух экземплярах в Docker-контейнерах — для тестирования и основного сервера. Это разделение позволяет защитить данные от удаления во время тестов и гарантирует независимость тестов друг от друга. Для настройки контейнеров использован Docker Compose, а основная функция базы данных заключается в хранении музыкальных библиотек пользователей и истории их переноса, с центральной таблицей пользователей, связанной с другими таблицами через связи типа Many-to-One.

Процесс регистрации и авторизации пользователей реализован с помощью POST и GET-запросов, создающих и подтверждающих учетные записи через отправку и проверку токенов на электронную почту. После успешной регистрации пользователю выдается JWT-токен, который используется для аутентификации при последующих запросах к серверу. Эта процедура также применяется для смены пароля.

JWT-токены формируются с использованием класса JwtTokenProvider, где каждый токен включает заголовок с информацией об алгоритме шифрования, payload с данными о пользователе и сроком действия токена, а также подпись, созданную с использованием алгоритма HS256. Для проверки токена сервер повторяет процесс его создания и сверяет с полученным в запросе, обеспечивая безопасную авторизацию.

Результаты тестирования

Тестирование серверной части приложения проводилось с использованием Postman для ручного тестирования и Apache JMeter для нагрузочного тестирования. Postman позволил удобно структурировать и протестировать API, а JMeter использовался для оценки производительности сервера при разных нагрузках [10]. Результаты показали, что сервер стабильно справляется с 100 одновременными пользователями, но при 1000 пользователях время ответа значительно увеличивается, что требует дальнейшей оптимизации при необходимости.

Тестирование клиентской части приложения включало создание модульных тестов с использованием XCTest для проверки ключевой логики, особенно алгоритмов сравнения аудиозаписей. Были созданы протоколы для упрощения тестирования, позволяющие подменять зависимости и улучшать расширяемость кода. Также рассматривалось автоматизированное UI-тестирование, но оно было сочтено нецелесообразным из-за трудоемкости дальнейшей поддержки тестов, поэтому основной акцент был сделан на ручном тестировании.

Алгоритмы сравнения аудиозаписей, задействованные при переносе библиотек, были протестированы тремя методами: простым сравнением названий, использованием коэффициента Левенштейна и собственным алгоритмом. Простое сравнение позволило перенести 70 % записей, коэффициент Левенштейна — 88 %, а собственный алгоритм — 99 %. Каждый метод имеет свои преимущества: простое сравнение минимизирует ошибки, но переносит меньше записей, коэффициент Левенштейна достигает средней точности, а собственный алгоритм максимально переносит записи, но допускает небольшие отклонения.

Выбор алгоритма зависит от предпочтений пользователя: точность или количество перенесенных аудиозаписей. Все три алгоритма могут быть полезны для разных категорий пользователей, обеспечивая гибкость и адаптируемость процесса переноса музыкальных библиотек между сервисами.

Заключение

По итогам проделанной работы была разработана полноценная версия программного продукта. Выявлено, что архитектура системы позволяет легко добавлять новые компоненты и обеспечивает достаточный уровень защиты данных от несанкционированного доступа.

Модульная структура программы делает ее удобной для командной работы, что важно при расширении проекта. В процессе разработки подтвердился высокий спрос на подобное решение, так как появились новые конкуренты, что подчеркивает актуальность идеи.

Для дальнейшего развития проекта целесообразно расширить поддержку большего числа сервисов, особенно тех, которые не имеют открытого API. Это можно реализовать через интеграцию с веб-страницами этих сервисов или через экспорт музыкальных библиотек в различные форматы файлов, поддерживаемые другими платформами.

ЛИТЕРАТУРА

1. Music Streaming App Revenue and Usage Statistics / Текст: электронный // Business of Apps. — URL: https://www.businessofapps.com/data/music-streaming-market/ (дата обращения: 16.08.2024).
2. iPhone vs Android market share / Текст: электронный // Macworld. — URL: https://www.macworld.com/article/673487/iphone-vs-android-market-share.html (дата обращения: 16.08.2024).
3. AppStore / Текст: электронный // Apple. — URL: https://www.apple.com/app-store (дата обращения: 16.08.2024).
4. Huang Z., Fan X., Li Z., Zhao C., Chen G., and Liu Y. Analysis of Anomaly Detection Techniques Applied to Web API Network Scenario // IEEE 11th Joint International Information Technology and Artificial Intelligence Conference. — 2023. — vol. 11.
5. Rahkema K., Pfahl D., and Ramler R. Analysis of Library Dependency Networks of Package Managers Used in iOS Development. // 2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft). — 2023.
6. Mythily M., Kanakala V.R., and Nambiar R. An Extensive Review of Spring Boot Testing Based on Business Requirements of the Software. // 2023 4th International Conference on Smart Electronics and Communication. — 2023.
7. Nugraha A.F., Kabetta H., Buana I.K.S., and Hadiprakoso R.B. Performance and security comparison of json web tokens (jwt) and platform agnostic security tokens (paseto) on restful apis. // 2023 IEEE International Conference on Cryptography, Informatics, and Cybersecurity. — 2023.
8. Zineddine A., Chakir O., Sadqi Y., Maleh Y., Gaba G.S., Gurtov A., and Dev K. A systematic review of cybersecurity assessment methods for HTTPS // Computers and Electrical Engineering. — 2023.
9. Indrawan D., Kusumo D.S., and Puspitasari S.Y. Analysis of the Implementation of MVVM Architecture Pattern on Performance of iOS Mobile-based Applications. // Jurnal Ilmiah Penelitian dan Pembelajaran Informatika. — 2023.
10. Tiwari V., Upadhyay S., Goswami J.K., and Agrawal S. Analytical Evaluation of Web Performance Testing Tools: Apache JMeter and SoapUI. // 2023 IEEE 12th International Conference on Communication Systems and Network Technologies. — 2023.