DOI 10.37882/2223–2966.2022.04–2.20

РАЗРАБОТКА ВЕБ АУДИО СЕРВИСА С ИСПОЛЬЗОВАНИЕМ PROGRESSIVE WEB APPS ТЕХНОЛОГИЙ И КЭШИРОВАНИЯ ДАННЫХ

DEVELOPMENT OF A WEB AUDIO SERVICE USING PROGRESSIVE WEB APPS TECHNOLOGIES AND DATA CACHING

L. Karavashkin
S. Molodyakov

Summary. We propose a new approach to developing web audio services based on the use of Progressive Web Apps (PWA) technologies and data caching. PWA allows a web application to be installed on a smartphone directly from the browser. A feature of our service is the use of caching. Thanks to caching, you can achieve efficient use of network resources, as well as work offline. The paper considers caching strategies, as well as their application in the developed web application.

Keywords: progressive web app, service worker, web audio service, cache, caching strategies, single page application.

Каравашкин Лев Александрович
Санкт-Петербургский политехнический университет Петра Великого
karavashkin.la@edu.spbstu.ru
Молодяков Сергей Александрович
Д.т.н., профессор, Санкт-Петербургский политехнический университет Петра Великого
molodyakov_sa@spbstu.ru

Аннотация. Представлен новый подход разработки веб аудио сервисов, основанный на применении технологий Progressive Web Apps (PWA) и кэширования данных. PWA предоставляет возможность веб приложению быть установленным на смартфон прямо из браузера. Особенностью нашего сервиса является использование кэширования. Благодаря кэшированию можно добиться эффективного использования ресурсов сети, а также работать в режиме offline. Рассмотрены стратегии кэширования, а также их применение в разработанном веб приложении.

Ключевые слова: progressive web app, service worker, веб аудио сервис, кэш, стратегии кэширования, одностраничное веб приложение.

Введение

При разработке аудио сервиса решается вопрос, какого типа клиентское приложение разрабатывать. Можно разработать кроссплатформенное приложение или нативное приложение, зависящее от платформы. Выбор между нативным, кроссплатформенным и веб-приложениями может быть сложным [1]. Кроссплатформенный подход как правило требует меньше затрат, но обладает меньшей производительностью в сравнении с нативными. Объединить достоинства кроссплатформенного и нативного приложений позволяет применение технологий Progressive Web Apps (PWA). Разработка PWA — это новый подход, позволяющий объединить простоту разработки веб приложений и типичный для нативных приложений опыт пользовательского взаимодействия. Кроссплатформенность PWA обеспечивается за счет поддержки данной технологии популярными браузерами, например Chrome, Safari и Firefox [2]. Веб аудио сервис предполагает наличие клиентской и серверной частей. Технологии PWA применяются на клиентской части и не требуют дополнительных настроек на сервере.

Перерывы в связи и изменение скорости передачи данных влияют на качество аудио сервиса. Поэтому при создании аудио сервиса необходимо использовать кэширование данных, которое позволяет уменьшить негативное влияние изменения параметров сети и обеспечить offline доступ.

Целью работы является разработка подхода создания веб аудио сервиса на базе PWA технологий и стратегий кэширования данных на стороне клиента.

Обзор технологий и инструментов

Известно множество веб аудио сервисов: Spotify, SoundCloud, Yandex Music, VK Music, Youtube Music [3–5]. Они позволяют прослушивать мелодии, составлять плейлисты и др. Во всех сервисах имеется возможность кэширования аудио. Кэширование позволяет прослушивать мелодии даже после отключения интернета, а также обеспечивает беспрерывное прослушивание в условиях плохого качества сети. Однако возможность кэширования в представленных аудио сервисах не доступна в рамках веб аудио сервисов, а присутствует только в мобильных версиях. Отметим, что в Spotify и Youtube music применяется технология PWA [6, 7]. Но не использованы все возможности этой технологии. Их клиентские приложения можно добавить на рабочий стол из браузера, но нельзя использовать без доступа к интернету.

Мы предлагаем использовать следующий набор инструментов для разработки: PWA, инструмент для настройки service worker — Workbox, библиотека для разработки интерфейса — React, менеджер состояний приложения — Redux, база данных — IndexedDB.

Progressive Web App (PWA) — Прогрессивные веб-приложения — это тип программного обеспечения, поставляемого через интернет, созданный с использованием распространенных веб технологий, включая HTML, CSS, JavaScript [2, 8]. Функциональные возможности включают работу в автономном режиме, push уведомления и доступ к оборудованию устройства. Минимальные требования для PWA [9, 10]: использование протокола HTTPS, наличие service worker и манифеста веб-приложения [11].

Применение технологии PWA дает следующие возможности [12]:

1. Отзывчивость. Интерфейс приложения может быть настроен для использования на смартфонах, планшетах, ноутбуках и настольных компьютерах. Отзывчивость может достигаться за счет использования медиа-запросов.
2. Возможность быть независимым от подключения к интернету за счет работы service worker, который входит в состав PWA. Service worker позволяет приложению работать в автономном режиме.
3. Возможность реализовать обновление приложения, благодаря процессу обновления Service Worker [13].
4. Безопасность. Service worker требует использовать https соединение.
5. Возможность установки приложения как из браузера, так и из магазинов приложений, таких как App Store и Play Market.
6. Native-llike. Приложение ведет себя как нативное с точки зрения взаимодействия пользователя с интерфейсом и навигации.

Service Worker — ядро технологии PWA, выполняющее роль проксирующего слоя между веб приложением и браузером. Все запросы браузера идут через него. Он позволяет настроить поведение веб-приложения в режиме offline, перехватывая и обрабатывая сетевые запросы. Ему также доступны push-уведомления и API для фоновой синхронизации [14]. Service Worker представляет из себя JavaScript файл, подключаемый к html странице. Он не имеет доступа к Document Object Model (DOM) веб страницы, работает в отдельном потоке JavaScript, и, следовательно, не блокирует основной поток приложения.

Необходимо использовать кэширование ресурсов, т.е. хранение ресурсов на клиентском устройстве, чтобы уменьшить время отклика на запросы на загрузку аудио, плейлистов и изображений. Однако большинство ресурсов отличаются друг от друга и для каждого ресурса нужно настраивать кэш отдельно. Самый простой способ настройки предоставляется библиотекой Workbox. Workbox Strategies — инструмент, позволяющий настроить поведение при взаимодействии с кэшем для различных сетевых ресурсов [15].

Для определения к какому ресурсу будет применена стратегия кэширования используется инструмент workbox cacheable response. Этим инструментом можно настроить сетевые запросы для кэширования.

Чтобы определить ограничения для кэша используется CacheExpiration. Можно настроить следующие параметры:

1. maxEntries — максимальное количество записей в кэше. Если предел по записям достигнут, то при следующем добавлении в кэш будет заменена запись, к которой реже всего обращались.
2. maxAgeSeconds — максимальное время жизни записи в кэше в секундах.

Для разработки использовалась популярная комбинация для одностраничных веб приложений — React [16] и Redux [17].

React — библиотека для разработки интерфейса веб приложений. React позволяет уменьшить время рендеринга страниц приложения за счет использования Virtual DOM и оптимизаций рендеринга.

Redux — это библиотека для мониторинга состояния веб-приложения, реализующая архитектуру Flux [18]. Flux — это архитектурный шаблон, предложенный Facebook для создания SPA. Предлагается разделить приложение на следующие части: хранилище, диспетчер, представление, действия. Данные в Flux-приложении текут в одном направлении: представления запускают действия, которые передаются через диспетчера, отвечающего за обновление хранилищ, данные которых привязаны к представлениям. Однонаправленный поток данных помогает избежать рассинхронизации между состоянием приложения и интерфейсом. Redux позволяет нам хранить только один объект состояния приложения в локальной базе данных и сокращает время разработки программного обеспечения. Также рассматривалась библиотека MobX, но при ее использовании состояние приложения может быть разделено на части, что может усложнить дальнейшее кэширование состояния. Мы предлагаем использовать IndexedDB в качестве локальной базы данных [19–20]. IndexedDB — API для работы с клиентским хранилищем больших объемов структурированных данных. IndexedDB подходит для хранения redux store.

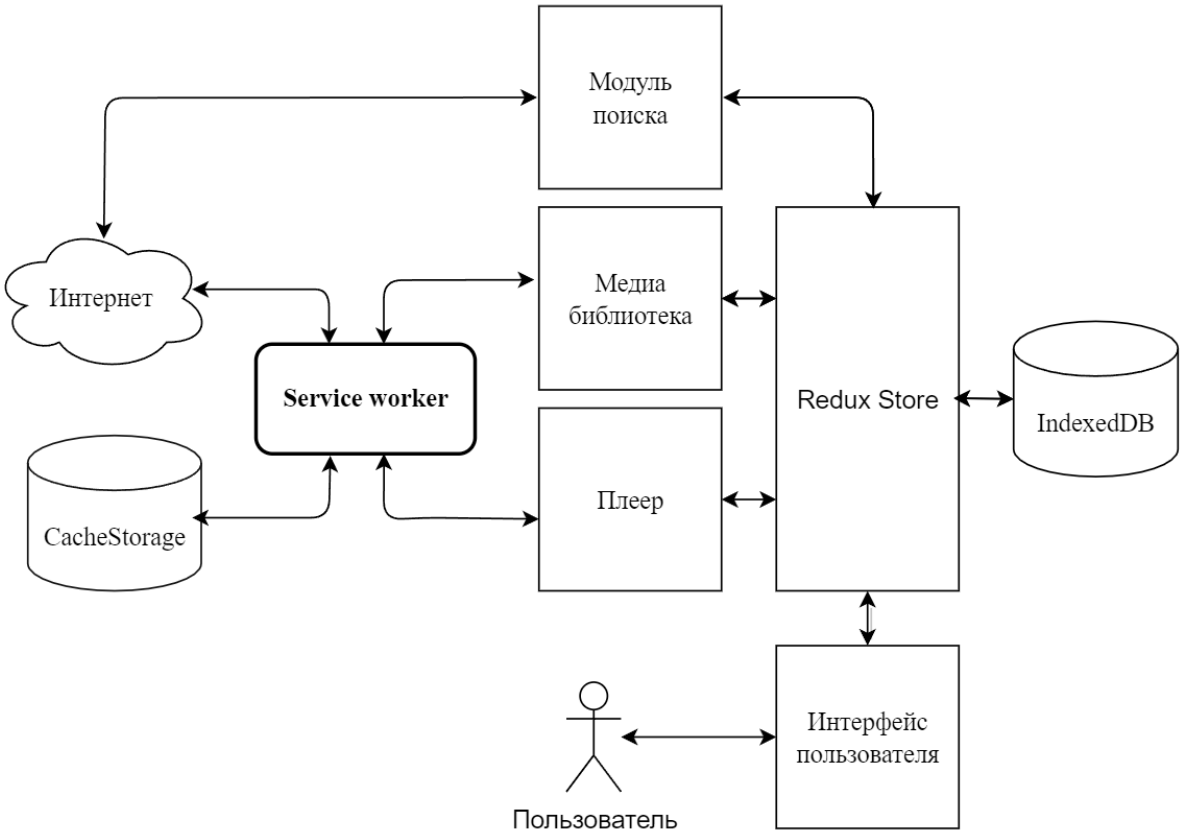

Рис. 1. Архитектура веб аудио сервиса

Аудио сервис с кэшированием данных

Структурная схема разработанного аудио сервиса представлена на рис. 1. Пользователь работает с приложением через пользовательский интерфейс. Состояние интерфейса зависит от состояния Redux Store, который обеспечивает единое место хранения состояния приложения. Состояние приложения включает в себя все плейлисты пользователя и состояние плеера: текущий плейлист, текущая аудиозапись, текущее время проигрывания, режим проигрывания, громкость.

Для сохранения состояния Redux Store между сеансами использования приложения используется модуль локальной базы данных IndexedDB. В процессе использования приложения состояние периодически сохраняется в базе данных, а при запуске приложения состояние восстанавливается из базы данных.

Модуль поиска отвечает за поиск аудио и плейлистов. Модуль “медиа библиотека” отвечает за логику работы с пользовательскими плейлистами. Этот модуль связан с модулем Service Worker, который обеспечивает механизм кэширования сетевых запросов. Модуль “Плеер” отвечает за воспроизведение аудио и параметры воспроизведения. Этот модуль также связан с модулем “Service Worker” для обеспечения кэширования аудио. При использовании приложения с нескольких устройств рекомендуется на стороне сервера реализовать синхронизацию локальных баз данных с серверной.

Особенностью предлагаемого подхода является использование различных стратегий кэширования. Традиционно кэш позволяет сократить время доступа к хранилищам данных за счет их буферного хранения [21, 22]. Использование кэша в Интернете также снижает нагрузку на сервер за счет уменьшения количества запросов к ресурсам, которые уже были добавлены в кэш. При работе с веб-приложениями имеются особенности кэширования. Одной из особенностей является разнообразие используемых данных. Данные могут быть в виде текста, изображений, видео, аудио и т.д. Другой особенностью может быть наличие прокси-кэширования. Кэш прокси может располагаться как на стороне веб-сервера, так и на стороне браузера [23, 24]. Кэширование прокси позволяет серверу или браузеру выступать в роли посредника между пользователем и поставщиком веб-контента. Когда пользователь заходит на веб-сайт, прокси-серверы интерпретируют и отвечают на за-

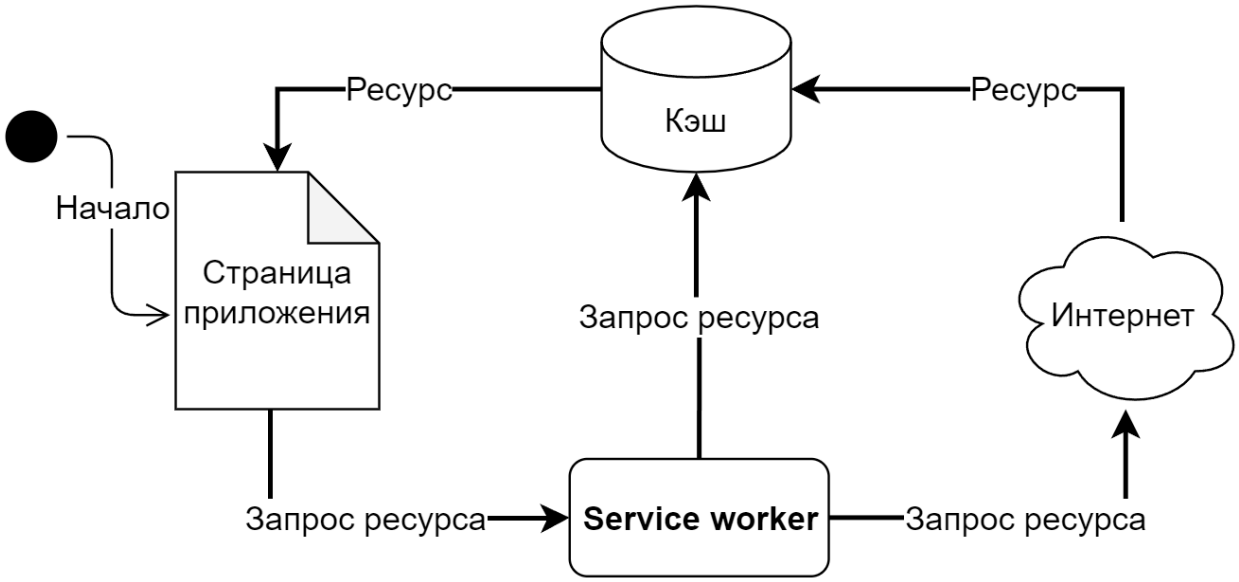

Рис. 2. Стратегия state-while-revalidate

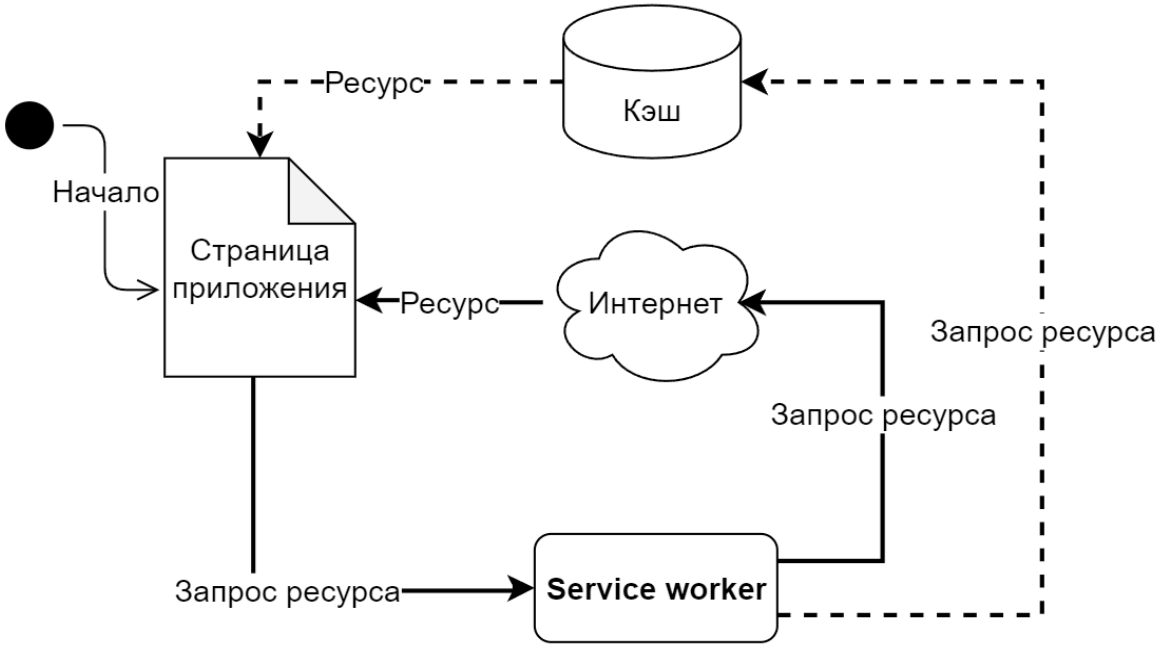

Рис. 3. Стратегия network first

просы от имени исходного сервера. Кэширование через прокси упрощает кэширование веб-ресурсов, так как не требует специальной настройки веб-приложения. Веб-приложение работает с API как обычно.

Service Worker использует CacheStorage [25] для хранения кэша сетевых запросов. Service Worker в фоновом режиме перехватывает запросы и кэширует необходимые ресурсы. Чтобы определить, по какой схеме будут кэшироваться ресурсы, используются workbox strategies.

Разработанная система позволяет использовать следующие стратегии кэширования.

Стратегия Stale-while-revalidate (рис. 2). Запрос на ресурс передается service worker. Service worker ищет

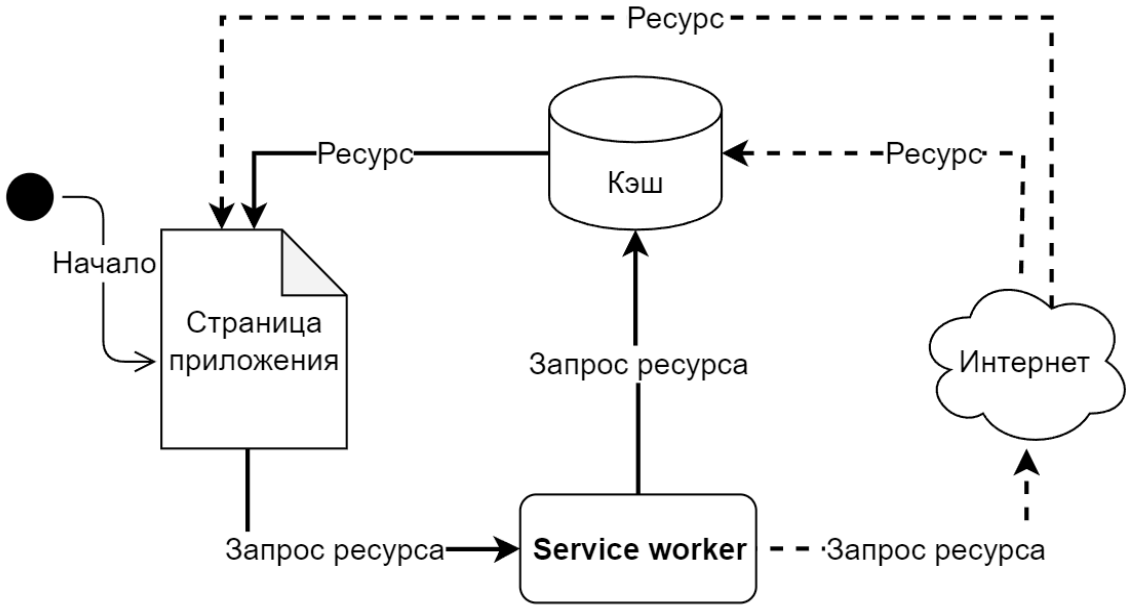

Рис. 4. Стратегия cache first

этот запрос в кэше и отдает приложению ответ из кэша. В тоже время отправляется реальный сетевой запрос, ответ которого заменит текущий кэш.

Стратегия Network first (рис. 3). Сплошными линиями показана передача запросов и ответов при наличии соединения с Интернетом. Пунктирным линиями соответствует отсутствие соединения с Интернетом. По умолчанию будет попытка получить ответ из сети. Если запрос выполнен успешно, полученный ответ обновит кэш. Если сеть не смогла вернуть ответ, то будет использоваться ответ из кэша.

Стратегия Cache first (рис. 4). Ответ на запрос в первую очередь ищется в кэше. Если кэш отсутствует, отправляется реальный запрос в сеть. Полученный ответ передается клиенту, а также будет добавлен в кэш, чтобы последующий запрос обслуживался из кэша.

Cache only strategy — ответ возвращается только на предварительно сохраненный в кэше ресурс. Network only strategy — запрет на использование кэша для ресурса.

При сравнении этих стратегий можно отметить следующие особенности. Наименьшая задержка для запроса ресурсов достигается при использовании стратегий Cache first, Stale-while-revalidate, Cache only. В начале все они пытаются получить сохраненный ресурс из кэша. Разница между этими стратегиями заключается в возрасте запрошенного ресурса, который будет возвращен. В стратегии Cache only отсутствует механизм обновления ресурсов. В первой стратегии кэширования ресурс не будет обновляться до тех пор, пока существует запись в кэше. Стратегия Stale-while-revalidate обновляет ресурс в кэше после каждого запроса, но результат этого обновления можно получить только при следующем запросе.

Получение наиболее новой версии запрашиваемого ресурса достигается при использовании стратегий Network first, Network only. Network only всегда возвращает ресурс из реального запроса сети и запрещает кэширование ресурса. Network first пытается получить ресурс из сети, а если сети нет или запрос завершился с ошибкой, то возвращает ресурс из кэша.

Мы предполагаем, что пользователь будет использовать веб аудио сервис с различных устройств. Поэтому состояние пользовательской библиотеки на разных устройствах может отличаться. Чтобы избежать состояния рассинхронизации, а также учитывая описанные выше особенности стратегий кэширования предлагаются следующие рекомендации.

Интерфейс SPA представляет собой файл javascript, который также является веб-ресурсом, который можно добавить в кэш. При запросе на открытие веб-приложения service-worker может вернуть этот файл из кэша пользователю. Этот механизм обеспечит запуск приложения без интернета. Стратегию устаревания во время повторной проверки следует использовать для сохранения пользовательского интерфейса приложения, если вы планируете обеспечить возможность обновления

вашего приложения. Если приложение не планируется обновлять в будущем, следует применить стратегию только кэширования.

Для сохранения пользовательской медиа библиотеки следует использовать стратегию Network first для получения одним запросом самой новой версии библиотеки из сети, но сохраняя возможность получения кэшированной версии в случае отсутствия интернет-соединения. Другой вариант — использование стратегии stale-while-revalidate. В этом случае состояние также будет синхронизировано, но после повторного запроса.

Для хранения изображений и аудиозаписей следует использовать стратегию Cache first. Эти ресурсы являются неизменяемыми во времени, но их нельзя добавить в кэш предварительно.

На кэширование содержания аудио стоит уделить особое внимание. Так как обычно запрос на прослушивания аудио представляет из себя Range Request в результаты которого аудио загружается постепенно по мере прослушивания, то для того, чтобы получить кэш на всю аудиозапись, в фоне отправляется запрос на получение полного аудио, и только когда всё аудио загружено, оно попадает в кэш.

Мы использовали статистику [26] при выборе параметров кэширования, таких как количество аудиозаписей в кэше и время жизни кэша. В среднем пользователи тратят от 99 до 140 минут в день на прослушивание музыки в популярном приложении Spotify в зависимости от региона проживания [26]. Средняя продолжительность музыкального произведения составляет 3 минуты 30 секунд. Исходя из этого, в среднем в день может понадобиться от 28 до 40 аудиозаписей в кэше.

Мы рассматриваем временной отрезок в один день. Это покроет все ситуации, когда при обычном использовании может потребоваться офлайн-доступ: аварийное отключение домашнего интернета, потеря связи в метро, в машине и при полете на самолете. Время жизни кэша для аудио должно быть установлено с запасом от 2 до 7 дней. Это позволит пользователям прослушать давно добавленную мелодию, а также вернуться к использованию приложения после перерыва.

Изучение поведения пользователей и алгоритмов адаптации требует дальнейших исследований. Например, то, как часто пользователи ищут новую музыку и используют плейлисты, может повлиять на выбор наиболее эффективной стратегии и настроек кэширования. Еще одной темой для исследований является изучение механизмов защиты авторских прав. Например, если платная подписка пользователя заканчивается, может потребоваться удалить кэширования аудио. Это можно реализовать, ограничив время жизни кэша аудио и периодически продлевая время жизни, если подписка активна.

Заключение

В статье предлагается новый подход к разработке аудио сервиса с использованием стратегий кэширования. Стратегию stale-while-revalidate следует использовать для сохранения пользовательского интерфейса приложения. Это позволяет при необходимости заменить старую версию приложения на новую, сохранив при этом возможность запуска приложения без интернета. Используйте стратегию Network first, чтобы сохранить пользовательскую медиатеку. Используйте стратегию Cache first для хранения изображений и аудиозаписей. Использование технологии Progressive Web Apps позволяет приложению работать онлайн и офлайн и предоставлять пользователю интерфейс, идентичный нативному приложению. Используемые решения позволяют динамически изменять параметры кэша и применяемые стратегии на основе изучения поведения пользователей.

ЛИТЕРАТУРА

1. Majchrzak T.A., Biorn-Hansen A., Gronli T.-M. Progressive Web Apps: the Definite Approach to Cross-Platform Development? // Proceedings of the 51st Hawaii International Conference on System Sciences.— 2018.— P. 5735–5744.
2. Progressive Web Apps browser support & compatibility [Электронный ресурс]. URL: https://www.goodbarber.com/blog/progressive-web-apps-browser-support-compatibility-a883/ (дата обращения: 25.03.2022).
3. Spotify digital music service [Электронный ресурс]. URL: https://www.spotify.com/ (дата обращения: 25.03.2022).
4. SoundCloud online audio distribution platform [Электронный ресурс]. URL: https://soundcloud.com/ (дата обращения: 25.03.2022).
5. Rahimi R.A., Park K.H. A Comparative Study of Internet Architecture and Applications of Online Music Streaming Services: The Impact on the Global Music Industry Growth // 2020 8th International Conference on Information and Communication Technology, ICoICT 2020.— 2020.— P. 1–6, DOI: 10.1109/ICoICT49345.2020.9166225
6. YouTube available to install as Progressive Web App (PWA) — 9to5Google [Электронный ресурс]. URL: https://9to5google.com/2021/01/24/youtube-install-pwa/ (дата обращения: 25.03.2022).
7. Spotify PWA: The Movement is Now — SimiCart Blog [Электронный ресурс]. URL: https://www.simicart.com/blog/spotify-pwa/ (дата обращения: 25.03.2022).

8. Aguirre V. et al. PWA and TWA: Recent Development Trends // Argentine Congress of Computer Science.— 2019.— P. 205–214.
9. Inkane N.S., Kotak S.A., Manekar A.S. Splay: A Lightweight Video Streaming Application // Proceeding — 1st International Conference on Innovative Trends and Advances in Engineering and Technology, ICITAET 2019., 2019.
10. Andrade Cardieri G. de, Zaina L.A.M. Playing the role of Co-designers on Mobile PWAs: An investigation of end-users interaction // ICEIS2020 — Proceedings of the 22nd International Conference on Enterprise Information Systems.— 2020.— P. 476–486.
11. Web App Manifest MDN [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/Manifest (дата обращения: 25.03.2022).
12. Steiner T. What is in a Web View: An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser // The Web Conference 2018 — Companion of the World Wide Web Conference, WWW 2018.— 2018 https://doi.org/10.1145/3184558.3188742
13. Service Worker API — Интерфейсы веб API | MDN [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/API/Service_Worker_API (дата обращения: 25.03.2022).
14. Chinprutthiwong P. et al. Security Study of Service Worker Cross-Site Scripting. // ACM International Conference Proceeding Series.— 2020 DOI: https://doi.org/10.1145/3427228.3427290
15. Workbox Strategies — Chrome Developers [Электронный ресурс]. URL: https://developer.chrome.com/docs/workbox/modules/workbox-strategies/ (дата обращения: 25.03.2022).
16. Chęć D., Nowak Z. The performance analysis of web applications based on virtual DOM and reactive user interfaces // Advances in Intelligent Systems and Computing.— 2019.
17. Redux — A predictable state container for JavaScript apps. | Redux [Электронный ресурс]. URL: https://redux.js.org/ (дата обращения: 25.03.2022).
18. In-Depth Overview | Flux [Электронный ресурс]. URL: https://facebook.github.io/flux/docs/in-depth-overview/ (дата обращения: 25.03.2022).
19. Xu H. et al. JSNVM: Supporting Data Persistence in JavaScript Using Non-Volatile Memory // Proceedings of the International Conference on Parallel and Distributed Systems — ICPADS.— 2019 DOI: 10.1109/PADSW.2018.8644622
20. Kimak S., Ellman J. The role of HTML5 IndexedDB, the past, present and future // 2015 10th International Conference for Internet Technology and Secured Transactions, ICITST 2015.— 2016 DOI: 10.1109/ICITST.2015.7412126
21. Nagarajan V. et al. A Primer on Memory Consistency and Cache Coherence, Second Edition // Synth. Lect. Comput. Archit.— 2020.— 294 p.
22. Ермаков Н.В., Молодяков С.А. Разработка системы хранения с использованием методов быстрого доступа к данным // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки.—2021. -№ 08. -С. 50–56 DOI 10.37882/2223–2966.2021.08.09
23. Kumar C., Marston S. Accelerating the Internet in the presence of Big Data: Reducing user delays by leveraging historical user request patterns for web caching // J. Inf. Syst. Technol. Manag.— 2019.— v. 16 DOI: 10.4301/S1807–1775201916006
24. Malavolta I. at al. Evaluating the impact of caching on the energy consumption and performance of progressive web apps // Proceedings — 2020 IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2020.— 2020.— P. 109–119 https://doi.org/10.1145/3387905.3388593
25. CacheStorage — Интерфейсы веб API | MDN [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/API/CacheStorage (дата обращения: 25.03.2022).
26. Spotify Revenue and Usage Statistics (2022) — Business of Apps [Электронный ресурс]. URL: https://www.businessofapps.com/data/spotify-statistics/ (дата обращения: 25.03.2022).