Краткое описание проекта системы

Цель проекта – создать максимально оптимизированное по скорости загрузки и быстроте отклика Web-приложение, используя современные инструменты балансировки и сжатия. Отдельно будут рассмотрены клиент и серверная части приложения. Основными показателями будут являться:

* LCP (Largest Contentful Paint), FCP (First Contentful Paint) – Для клиента
* Латентность, RPS, CPU Usage, Memory Usage – Для сервера

Почему это важно? Помимо улучшения UX, скорость влияет на продвижение веб-приложения поисковыми системами.

Это не совсем то, что хотелось бы увидеть.

Вы описали цель проекта, я же предлагал уже спуститься на уровень непосредственной проработки проекта. Нужно ответить на простые вопросы. Где будет работать приложение? Что оно будет делать? Какие, может быть, особые функции у него будут? Каковы пределы использования приложения?

Это может выглядеть как-то так.

*ПО будет работать на стороне веб-сервера и бек-энда. Будет осуществляться как статическое сжатие изображений, так и сжатие изображений на лету. Для сжатия текстовой информации будут использоваться стандартные функции веб-сервера. Видео будет или нет? Сжатые изображения будут передаваться в формате (каком или каких?).*

*Уменьшение скорости загрузки будет достигаться за счёт (чего?). Повышение скорости отклика будет достигаться за счет (чего?). Приложение будет масштабироваться в следующих пределах (например, до стольких-то одновременных запросов или ещё как-то)*

Это я написал просто в качестве примера. У вас наверняка свой взгляд уже есть на организацию приложения, вот его и опишите

Приложение будет работать на виртуальном сервере (VPS), возможно на нескольких.

Примерная конфигурация сервера:

* 2-4 ядра
* 2-4 гб оперативной памяти
* Ubuntu

Фронтэнд будет написан с использованием фреймворка React. Далее будут произведены возможно следующие оптимизации:

1. Code Splitting:

- Использование React.lazy() и React.Suspense для динамической загрузки компонентов только при необходимости.

- Инструменты такие, как webpack, позволяют разбить код на более мелкие чанки, загружаемые по мере необходимости.

2. Использование мемоизации:

- Использование React.memo для предотвращения ненужных перерисовок компонентов.

- Использование useMemo и useCallback для мемоизации вычислений и функций.

3. Оптимизация ререндеров:

- Отслеживание за количества ререндеров компонентов и оптимизация их, используя чистые компоненты или методы shouldComponentUpdate (в функциональных компонентах React.memo).

- Часто полезен инструмент React DevTools, который может помочь в анализе производительности компонентов.

4. Управление состоянием:

- Оптимизирование работы со состоянием, минимизируя количество ререндеров. Для больших и сложных приложений могут пригодиться библиотеки, такие как Redux или MobX.

- Использования контекста только для данных, которые действительно необходимы на глобальном уровне.

5. Объем и размер:

- Убедимся, что вы загружаемтолько те библиотеки и модули, которые действительно используем. Используем инструменты для анализа размеров бандлов, такие как webpack-bundle-analyzer.

- Использование инструментов для сжатия и минимизации кода, например, UglifyJS или Terser.

6. Оптимизация рендеринга списка:

- Для рендеринга длинных списков использование virtualized или windowing библиотеки, такие как react-window или react-virtualized, чтобы избежать создания всех элементов одновременно.

7. Использование Progressive Web App (PWA):

- Конвертирование своего приложения в PWA для улучшения производительности и оффлайн-работы. Для этого существует Create React App с встроенной поддержкой PWA.

8. Предзагрузка и предварительно подгружаемые ресурсы:

- Использование атрибутов rel="preload", rel="prefetch", и rel="preconnect" для ускорения загрузки критических ресурсов.

9. Сжатие изображений и мультимедиа:

- Оптимизирование изображений и других мультимедийных ресурсов, используя подходящие форматы и методы компрессии.

10. Использование Service Workers:

- Настройтка Service Workers для кэширования и ускорения загрузки повторных запросов.

Далее мы получим bundle и раздадим его с помощью Nginx

Для него возможно следующие оптимизации

1. Использование кэширования:

- HTTP кэширование: Установить заголовки Cache-Control и Expires, чтобы браузеры могли кэшировать статические файлы локально.

- fastcgi\_cache: Если вы использовать Nginx вместе с FastCGI, попробовать включить fastcgi\_cache.

2. Сжатие:

- gzip: Использование сжатия Gzip для уменьшения размера передаваемых данных. Это можно сделать с помощью директив gzip, gzip\_static и других связанных настройками.

3. Поддержка HTTP/2:

- Использование протокола HTTP/2 для использования возможностей мультиплексирования и улучшенной производительности сети.

4. Использование sendfile:

- Директива sendfile позволяет Nginx отправлять файлы напрямую из файловой системы, что может значительно снизить нагрузку на процессор.

5. Настройка worker\_processes и worker\_connections:

6. Использование статического контента из нескольких доменов (шардинг домена):

- Разделить статический контент между несколькими поддоменами, чтобы обойти ограничения на количество параллельных соединений к одному домену в браузерах.

7. Поддержка заголовков ETag и Last-Modified:

- Настройте ETag и Last-Modified для более эффективного кэширования на стороне клиента.

8. Использование расширений и модулей:

- Подумайте о добавлении модулей и расширений для логирования, кэширования и других улучшений производительности.

Можешь переписать это не как совет, а как указание, например:

- Настройте ETag и Last-Modified для более эффективного кэширования на стороне клиента.

-Настройка ETag и Last-Modified для более эффективного кэширования на стороне клиента

Для тестирование сайта можно использовать следующие устум