Чеклист производительности Front-End на 2017

Ниже вы можете наблюдать список всех тем касательно оптимизации, с которыми Вам, возможно придется столкнуться в 2017 году.

ГОТОВИМСЯ, СТАВИМ ЦЕЛИ

☐ Будь на 20% быстрее ближайшего соперника.

Измерить время начала рендера (с помощью WebPagetest) и время отображения первой значимой картинки (с помощью Lighthouse на нескольких устройства с разной производительностью) — на стандартных скоростях 3G, 4G и при подключении к точке доступа Wi-Fi. Соберите информацию, составьте таблицу, ужмите ее на 20% и таким образом определитесь с бюджетом производительности

☐ Поделитесь чеклистом с коллегами

Удостоверьтесь что каждый член Вашей команды ознакомлен с данным чеклистом что бы избежать возможного недопонимания в будущем процессе. Каждое принятое решение имеет влияние на производительность, и проект будет только в выигрыше, если не только фронтендеры, но и UX и графические дизайнеры решаться следовать этому плану. Обозначьте решения дизайнера в бюджете производительности и расставьте приоритеты о которых упоминалось выше.

☐ Время отклика 100-миллисекунд, 60 кадров в секунду.

Каждый кадр анимации должен сменятся менее чем за 16 миллисекунд, тем самым Вы достигнете заветных 60 fps (1 секунда ÷ 60 = 16.6 миллисекунд). Будьте оптимистом и используйте время простоя с умом. Очевидно, что эта цель относится в большей степени к производительности во время выполнения, а не во время загрузки.

☐ Первая значимая картинка через 1.25 секунды, индекс скорости меньше 1000.

Вашей абсолютной целью должно быть начало отрисовки страницы уже через 1 секунду и значение индекса скорости менее 1000. Максимально-допустимая длительность отрисовки первого значимого изображения – 1250 миллисекунд. Для мобильных устройств неплохим показателем будет начало отрисовки страницы меньше чем через 3 секунды для 3G подключения.

ОПРЕДЕЛИМСЯ С ОКРУЖЕНИЕМ

☐ Выберите для себя и сконфигурируйте инструменты сборки.

Не стоит слишком зацикливаться на том, что считается трендовым инструментом в данный момент. До тех пор, пока Вам удается получать достичь результатов быстро, и Вы не ощущаете проблем в процессе использования инструментов сборки – у Вас должно быть все хорошо.

☐ Прогрессивное улучшение.

Сначала спроектируйте и реализуйте базовые возможности, и только потом, дополняйте их сложными фичами для подходящий браузеров, создавая отказоустойчивый интерфейс. В том случае если ваш вебсайт работает быстро на медленном устройстве, со слабым браузером и посредственным интернетом, вы точно можете быть уверены, что он будет отлично работать на устройстве побыстрее

☐ Angular, React, Ember и компания.

Присмотритесь к Фреймворку, который позволяет отрисовать картинку на стороне сервера. Не забудьте замерять время загрузки как в режиме отрисовки на сервере, так и на клиенте для мобильных устройств, прежде чем приступать к работе с Фреймворком. Каждый из фреймворков по-своему будет влиять на конечную производительность и будет требовать индивидуальной стратегии оптимизации, поэтому Вам нужно отчетливо понимать всю подноготную выбранного фреймворка.

☐ AMP от Google или Instant Articles от Facebook?

Конечно, вы можете добиться отличной производительности и без них, но AMP, к примеру, предлагает фреймворк с превосходной производительностью и бесплатным доступом к CDN, а Instant Articles взвинтят показатели производительности на Facebook. Вы, кстати, можете и сами заняться построением прогрессивных веб приложений.

☐ Подойдите мудро к вопросу о выборе вашей CDN

В зависимости от того насколько статична информация на вашем вебсайте, Вы можете попробовать использовать вынести часть контента во внешний ресурс, выгрузив ее на CDN и запрашивая оттуда статическую версию контента при необходимости, таким образом вы спасётесь от запросов к базе данных. Вы даже можете выбрать для себя статические хостинг-платформы, базирующиеся на CDN, обогащая страницы вашего вебсайта интерактивностью, в качестве прогрессивного улучшения (JAMStack).

ОПТИМИЗАЦИЯ СБОРКИ

- ☐ **Установите для себя четкие приоритеты.**

Проведите «инвентаризацию» всех своих ресурсов и контента (JavaScript, изображения, шрифты, внешние скрипты, и “увесистые” модули на странице, такие как карусели, сложная инфографика, и мультимедийный контент), и разделите их на группы.
Составьте таблицу. Определите основные моменты для устаревших браузеров (т. е. полностью доступный основной контент), затем, улучшенный интерфейс для подходящих браузеров (т.е. улучшенный функционал, полный интерфейс взаимодействия)
и дополнения (активы, который не являются абсолютно-необходимыми и могут быть подгружены позже (lazy-load), такие как веб шрифты, излишние стили, скрипты каруселей, видеоплейеры, кнопки социальных сетей, крупные изображения).

- ☐ **Используйте технику «Сбора вершков»**

Распределите процесс на стадии четко: Выгрузите главную часть моментально, улучшения – по событию DomContentLoaded и прочие дополнения по событию load.

- ☐ **Обратите внимание на микро-оптимизацию и прогрессивную загрузку.**

может понадобится какое-то время прежде чем вы сможет отрендерить страницу. Гораздо лучшим решением будет показывать скелет приложения вместо индикаторов загрузки. Присмотритесь поближе к модулям и техникам по уменьшению скорости изначальной отрисовки страницы (Например, tree-shaking и разделение кода), потому что большинство времени тратится именно на этом этапе на парсинг информации для приложения. Также используйте преждевременный компилятор чтобы выполнить как можно весомую часть рендера на сервере и следовательно выведите используемые данные быстро. В конце концов возьмите на вооружение Optimize.js

- ☐ **Правильно ли настроены HTTP заголовки?**

Обязательно перепроверьте что `expires, cache-control, max-age` и другие заголовки кэша HTTP правильно выставлены. В общем, ресурсы должны подлегать кэшированию или на очень короткий срок (если они могут изменяться) или навсегда (если они неизменны) — вы можете просто изменять их версию в URL при необходимости. По возможности,

используйте `Cache-control: immutable`, предназначенный для защищенных отпечатком пальца статических ресурсов, во избежание ре-валидации

☐ Ограничьте сторонние библиотеки и загружайте JavaScript асинхронно.

Нам, разработчикам, приходится настойчиво указывать браузеру что не стоит ждать окончания выполнения скриптов для начала рендера страницы. Простейшим способом добиться этого являются HTML атрибуты defer и async. На практике, хорошо бы нам отдавать предпочтение defer перед async (как поблажку для пользователей IE9 и старше (cost to users of Internet Explorer), потому что скорее всего иначе Вы сломаете им все скрипты). Также постарайтесь ограничить влияние сторонних скриптов и библиотек

☐ Правильно ли оптимизированы изображения?

Как можно больше используйте адаптивные изображения с атрибутами srcset, sizes и элементом <picture>. Раз мы уже об этом заговорили, удостоверьтесь что вы используете WebP формат для работы с изображениями тэга < picture > и резервным изображением в JPEG.
Вам в помощь Генератор брейкпоинтов для адаптивных изображений а также другие сервисы автоматизированной оптимизации изображений, например Cloudinary.

☐ Как обстоят дела с оптимизацией веб шрифтов?

Велик шанс, что в веб-шрифтах которые вы используете в своем проекте присутствуют символы, которые вы не используете. Поддержка WOFF2 великолепна, а для подстраховки Вы можете использоваться WOFF и OTF специально для браузеров с отстающей поддержкой. Также выберите для себя одну из стратегий Зака Лэзермана – “Полное руководство по стратегиям загрузки шрифтов,” а также используйте кэш сервис воркеров для постоянного кэширования шрифтов

☐ Молниеносно выгрузите критичные CSS стили.

Что бы быть уверенным на все 100% что Ваш браузер начнет рендер страницы настолько быстро, на сколько это возможно, общепринятой практикой становится собирать весь CSS необходимый для начала рендера первой видимой части страницы (известный как «критично-важный» CSS) и добавлять его инлайново в < head> вашего документа таким образом уменьшая запросы зависимости.

☐ Используйте tree-shaking и code-splitting для уменьшения нагрузки.

Tree-shaking это способ расчистить ваш процесс сборки вовлекая только ту часть кода, которая действительно используется на продакшене. Вы можете пользоваться Webpack 2 что бы свести к минимум ненужные экспорты кода.
Code-splitting это еще одна фича Webpack'а, которая разбивает основание кода на куски и подгружает по запросу. Как только Вы определите точки разделения вашего кода, Webpack сможет позаботиться о зависимостях, и выходных файлах.

☐ Улучшайте производительность рендера.

Изолируйте увесистые элементы с помощью политики сдерживания CSS — например, чтобы ограничить стили браузера для компоновки и окраса элементов, для навигации вне канваса, или для виджетов третьей стороны. Удостоверьтесь что нет промедлений при прокрутке страницы и при анимации элементов, и что пользователь сможет получать стабильные 60 кадров в секунду.

☐ "Разогрейте" соединение что бы ускорить загрузку.

Используйте скелетный экраны в интерфейсе и ленивую загрузку всех увесистых компонентов, как шрифтов fonts, JavaScript, карусели, видео и ifram'ы. Используйте ресурсные подсказки в целях сохранения времени на dns-prefetch, preconnect, prefetch, prerender и preload

HTTP/2

☐ Будьте готовы к HTTP/2.

Вместе с Google, двигающимся в сторону более безопасного веба и окончательного восприятия HTTP страниц Chrome'ом как «небезопасные» вам придется решиться сделать ставку, или на конфигурацию окружения HTTP/2 или остаться на HTTP/1.1. HTTP/2 отлично поддерживается; Он никуда не денется и в большинстве случаев лучше бы Вам принять его сторону.

☐ Правильно внедряйте HTTP/2.

Обработка контента через HTTP/2 serving assets over HTTP/2 требует больших доработок и сильно отличается от того, как вы обрабатывали ваши активы раньше. Вам придется найти наилучший баланс между упаковки всех модулей в один и загрузки множества мелких модулей параллельно.

☐ Удостоверьтесь в пуленепробиваемой защите вашего сервера.

Перепроверьте что заголовки безопасности правильно настроены, избавьтесь от заранее известных уязвимостей, и проверьте Ваш сертификат. Все еще не перешли на HTTPS? Присмотритесь к HTTPS-Only Стандарту в качестве инструкции. Также, удостоверьтесь что внешние плагины и отслеживающие скрипты загружены через HTTPS, что межсайтовый скриптинг невозможен.

☐ Поддерживают ли Ваш сервер и CDNы HTTP/2?

Разные сервера и CDNы скорее всего будет поддерживать HTTP/2 по-разному. Попробуйте Is TLS Fast Yet? Что-бы уточнить информацию касательно вашего варианта или в сравнении с другими узнать насколько производителен Ваш сервер и поддержку каких фич стоит ожидать в скором времени.

☐ Используется ли сжатие Brotli или Zopfli?

Brotli, новый формат данных без потерь, который уже широко поддерживается в Chrome, Firefox и Opera. На практике, Brotli представляется более эффективным чем Gzip и Deflate. Сам процесс сжатия может быть затяжным, в зависимости от настроек и чем дольше длится сжатие, тем выше будет уровень самого сжатия. Кстати говоря, декомпрессия происходит быстро. Не будет сюрпризом что браузеры будут поддерживать этот алгоритм только если пользователь посещает вебсайт через HTTPS, ведь алгоритм разработан в Google, но на самом деле, на, то есть еще и технические причины.

☐ Активировано ли OCSP сшивание?

Включая сшивание OCSP на своем сервере, Вы сможете значительно ускорить «рукопожатия» протокола TLS. OCSP не требует чтобы браузер тратил время, загружая список сертификатов и детальную информацию про них для поиска, таким образом уменьшая время затраченное на «рукопожатие».

☐ Успели ли Вы взять IPv6 на вооружение?

Поскольку запасы адресов IPv4 подходят к концу, большинство мобильных сетей получают поддержку IPv6 просто молниеносно (США достигли порога в 50% поддержки IPv6), будет замечательной идеей обновить поддержку Вашего DNS на IPv6 что бы в будущем обеспечить себе спокойный сон.

☐ Пользуетесь ли Вы HPACK сжатием?

В случае если вы используете HTTP/2, стоит убедиться, что Ваши сервера используют сжатие HPACK в заголовках HTTP ответов, чтобы уменьшить использование ресурсов. Из-за того, что сервера HTTP/2 относительно молоды, может иметь место неполноценная поддержка спецификаций, с HPACK, например. H2spec это отличный (и очень детализированный) инструмент что бы проверить работоспособность HPACK.

☐ Работают ли сервис воркеры для кэширования и сетевых фолбэков?

Если Ваш вебсайт использует HTTPS, стоит присмотреться к «Прагматичному гиду по сервис воркерам» что бы добавлять в кэш сервис воркеров статические материалы и локально хранить офлайн фолбэки (или целые офлайн страницы) и открывать их из памяти устройства пользователя а не отправляться за ними в. Будет полезным также посмотреть «Книгу рецептов для Офлайна» от Джейка и бесплатный Udacity курс “Offline Web Applications.”

ТЕСТИРОВАНИЕ И МОНИТОРИНГ

☐ Отслеживайте предупреждения микс-контента.

Если Вы не так давно перешли с HTTP на HTTPS, не забудьте промониторить предупреждения микс-контента, как активного так и пассивного, для этого подойдёт инструмент Report-URI.io. Вы также можете использовать Mixed Content Scan что бы просканировать Ваш вебсайт с активированным HTTPS-на содержание микс-контента.

☐ Оптимизирован ли Ваш рабочий процесс с Инструментами разработчика (DevTools)?

Выберите для себя инструменты отладки и прокликайте каждую кнопку, ссылку. Убедитесь, что Вы знаете, как проанализировать и улучшать производительность отрисовки страницы, консольный вывод, и, как отладить JavaScript или корректировать CSS стили.

☐ Какие результаты тестирования в устаревших браузерах? В прокси браузерах?

Тестирования в Chrome и Firefox, однозначно недостаточно. Оцените, как Ваш вебсайт выглядит в прокси-браузере и в устаревших браузерах. К примеру, UC Browser и Opera Mini, занимают огромный сегмент рынка в Азии (до 35% в Азии). Исследуйте и замерьте среднюю скорость подключения к сети в странах, на которых ориентирован Ваш продукты, чтобы в будущем не столкнуться с неприятным сюрпризом. Не забудьте провести тестирование с троттлингом и симулируйте дисплей повышенной плотности. BrowserStack - это отличный инструмент, но тестирование на физических девайсах не менее.

☐ Вы установили непрерывный мониторинг?

Установите непрерывный мониторинг за бюджетом производительности с автоматическими извещениями. Установите собственные временные оценки что бы измерять, сравнивать и мониторить метрики в зависимости от сферы деятельности.

МОМЕНТАЛЬНЫЙ РЕЗУЛЬТАТ!

Список достаточно обширный, и выполнение оптимизация может отнять у Вас порядочно времени. Давайте сузим список до десятки самых просто-достижимых целей. Очевидно, Вам придется замерять результаты до начала и после окончания оптимизации, включая время начала рендера и индекс скорости на 3G и проводном подключении.

1. Вашей целью является начало отрисовки страницы не позже чем через 1 секунду на проводном подключении и 3 секунды используя 3G, и индекс скорости величиной менее 1000. Оптимизируйте время начала отрисовки и время интерактивности.
2. Подготовьте критичный CSS для основных шаблонов, и включите их в `<head>` вашей страницы. (Ваш бюджет - 14 KB).
3. Используйте *Defer* и *lazy-load* во всех скриптах, где только возможно, как в своих собственных, так и в сторонних — особенно это касается социальных медиа ссылок, видеоплейеров, и увесистого JavaScript.
4. Добавьте подсказки ресурсов, чтобы ускорить загрузки с быстрым `dns-lookup, preconnect, prefetch, preload и prerender`.
5. Разделите веб шрифты и загружайте их асинхронно (или просто переключайтесь на системные шрифты, как альтернатива).
6. Оптимизируйте изображения, рассмотрите возможность использования WebP для основных страниц (например, лэндингов).
7. Удостоверьтесь что сами заголовки кэша HTTP и их безопасность установлены правильно.
8. Активируйте сжатие *Brotli* или *Zopfli* на сервере. (В случае если это невозможно, не забывайте про сжатие Gzip.)
9. Если HTTP/2 доступен, активируйте сжатие HPACK и начните отслеживать предупреждения смешанного контента. Если вы работаете над LTS, также активируйте сшивание OCSP.
10. Если представляется возможным, кэшируйте содержимое, например, шрифты, стили, скрипты и изображения, а вообще, чем больше – тем лучше! — в кэши сервис воркеров.