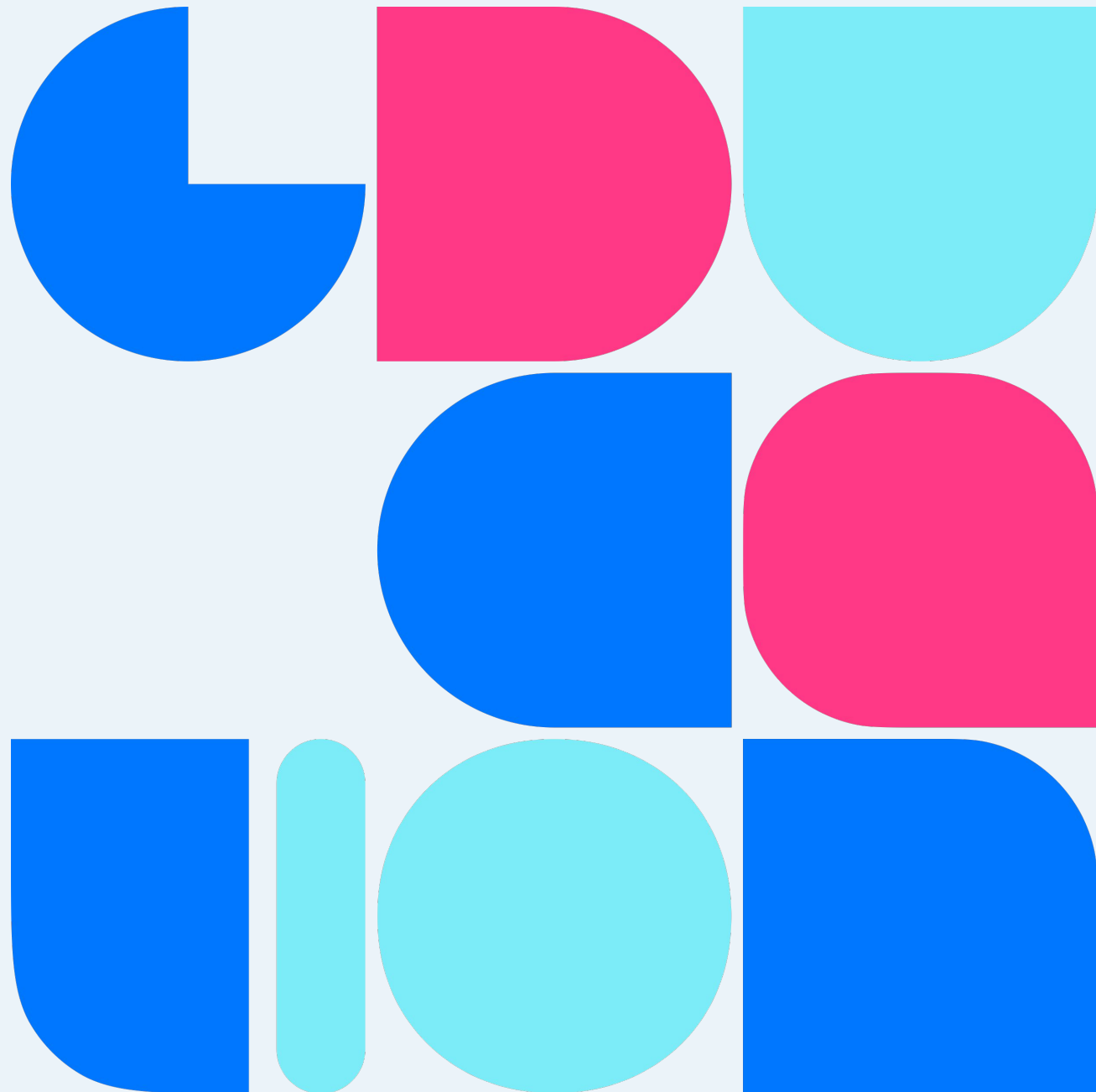




# Лекция 11

## Оптимизация

Дмитрий Зайцев  
Мартин Комитски



# План на сегодня

- Разбор квиза
- Оптимизация
  - Зачем?
  - Что можно оптимизировать?
  - Document
  - Сетевое взаимодействие
  - Вычисления
  - Прочее
  - Практика?

# Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях



# Разбор КВИЗа

# Оптимизация

# Зачем?

# Оптимизация. Для чего нужно оптимизировать?

- Пользовательский опыт
  - Снижение скорости загрузки страницы до 50% сокращает конверсию
  - Снижение быстродействия страницы также увеличивает число отказов
- SEO
  - Скорость загрузки и быстродействие страниц влияет на ранжирование в поиске
- Нагрузка на сервера
  - Количество запросов и объем передаваемых данных прямо влияют на производительность сервера

Что можно  
оптимизировать?



# Оптимизация. Что можно оптимизировать?

- Document
  - Количество элементов
  - Сложность/специфичность селекторов
  - Периодичность пересчета/перерисовки
- Сетевое взаимодействие
  - Блокирующие ресурсы
  - Размер ресурсов
  - Количество запросов
- Вычисления
  - Сложные манипуляции с данными
  - Неоптимальное использование обработчиков событий

# Document

# Оптимизация. Document

## Важно:

- $O(\text{Nodes} * \text{Selectors} * \text{Depth})$
- Сокращать количество элементов до минимально необходимого
- Не использовать недостаточно специфичные комбинаторы

1. `.my-class span {color: red}`
2. `.my-class > span {color: red}`
3. `.my-class_red {color: red}`
- 4.
5. `<html>`
6. `<div>`
7. `<span>text</span>`
8. `</div>`
9. `<div class="my-class">`
10. `<span class="my-class_red">red text</span>`
11. `</div>`
12. `</html>`

# Оптимизация. Document

- Избегать повторного вычисления стилей и пересчета Layout
- Триггеры пересчета Layout (reflow)
  - Изменение «геометрических свойств» элемента
  - Изменение стилей или свойств, влияющий на стили (<https://csstriggers.com/>)
  - Обращение к свойствам элемента, измененным после последнего пересчета

1. `node.classList.add('my-class');`
2. `console.log(node.offsetHeight);`

- И многое другое
- Использовать «слои», но не злоупотреблять ими
  - `will-change: value;`
  - `transform: translateZ(0);`
- Ограничивать размеры областей перерисовки и ее сложность

# Оптимизация. Document

- BEM

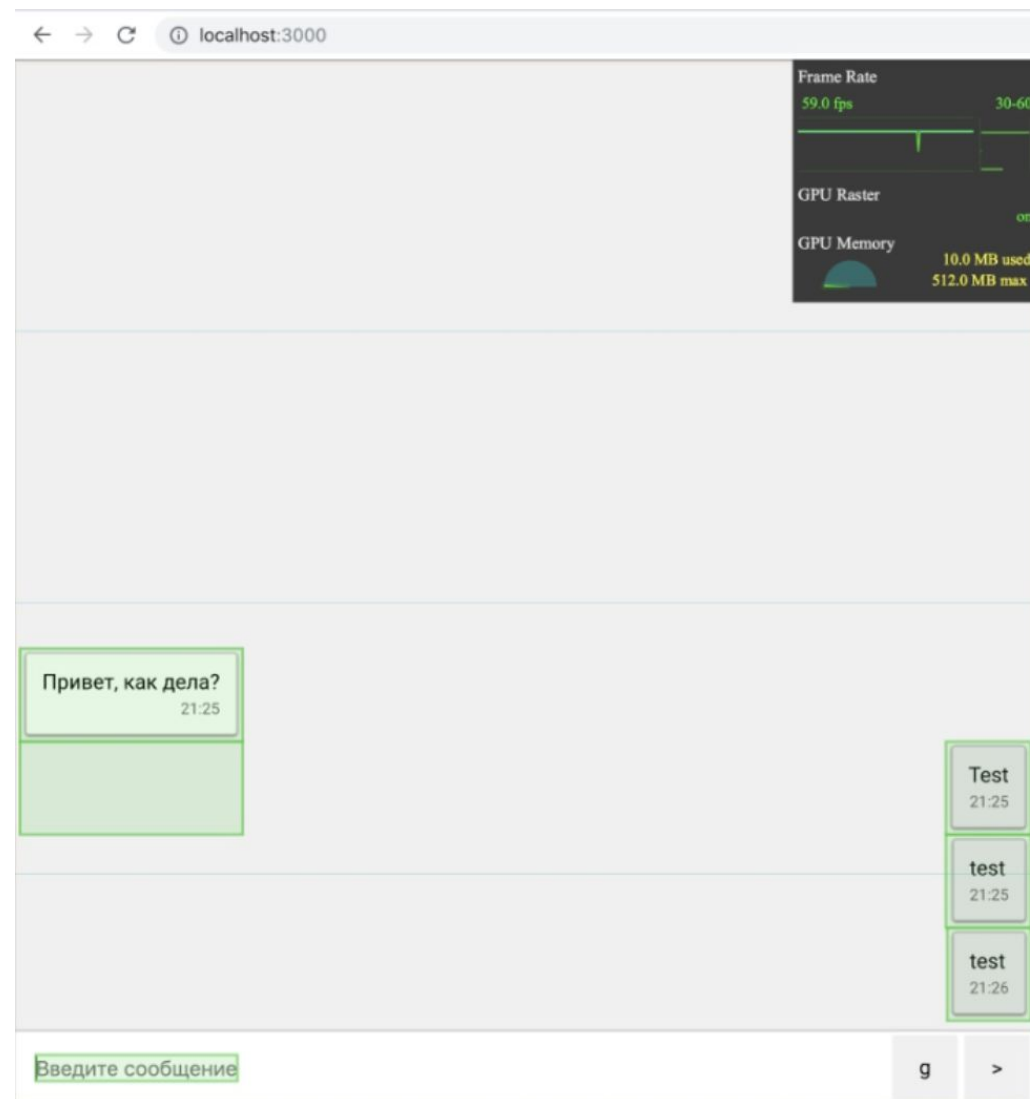
1. `<form class="search-form">`
2.   `<div class="search-form__content">`
3.     `<input class="search-form__input">`
4.     `<button class="search-form__button">Найти</button>`
5.   `</div>`
6. `</form>`

- CSS Modules

1. `// для CRA достаточно добавить .module к имени стилевого файла`
- 2.
3. `import styles from './style.css';`
4. `// import { className } from './style.css';`
5. `element.innerHTML = '<div class="' + styles.className + '">';`

# Оптимизация. Document

- Инструменты анализа производительности
  - [rendering settings](#)
  - [perfomance-tools](#)



# Сетевое взаимодействие

# Оптимизация. Сетевое взаимодействие

- Загрузка скриптов и стилей может откладывать момент отрисовки
- Инлайн критических ресурсов
- Отложенная загрузка
  - defer, async, preload, prefetch
  - js загрузчики



# Сетевое взаимодействие. Inline, async, defer

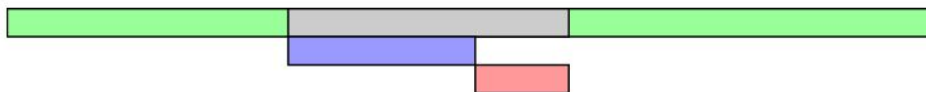
- [async-vs-defer-attributes](#)
- <https://stackoverflow.com/a/39711009>

## Legend

- HTML parsing
- HTML parsing paused
- Script download
- Script execution

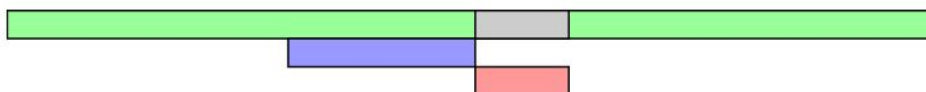
### <script>

Let's start by defining what **<script>** without any attributes does. The HTML file will be parsed until the script file is hit, at that point parsing will stop and a request will be made to fetch the file (if it's external). The script will then be executed before parsing is resumed.



### <script async>

**async** downloads the file during HTML parsing and will pause the HTML parser to execute it when it has finished downloading.



### <script defer>

**defer** downloads the file during HTML parsing and will only execute it after the parser has completed. **defer** scripts are also guaranteed to execute in the order that they appear in the document.



# Сетевое взаимодействие

- JS-бандлы

```
1.  /src
2.  └─ main.js
3.  └─ component.js
4.
```



```
1.  /dist
2.  └─ bundle.js
3.
```

# Сетевое взаимодействие

- Caching (expires, cache-control, max-age)
- Gzip (Content-Encoding)
- CDN
- Load Balancers
- Разбиение кода
- Tree Shaking/Dead code elimination
- Memoization
- Service Workers
- Progressive page loading (events, lazy images and iframes)
- Code Minification/Uglification
- Image, font optimization
- AMP/Instant Articles/Telegram's Instant View/Yandex Turbo
- [Resource hints](#)

# Сетевое взаимодействие

- CSS-спрайты

```
1. .foo {  
2.     background: url('../assets/gift.png?sprite');  
3. }
```



```
1. .foo {  
2.     background: url(/sprite.png?[hash]) no-repeat;  
3.     background-position: -100px -0px;  
4. }  
5.
```



# Сетевое взаимодействие

- SVG-спрайты

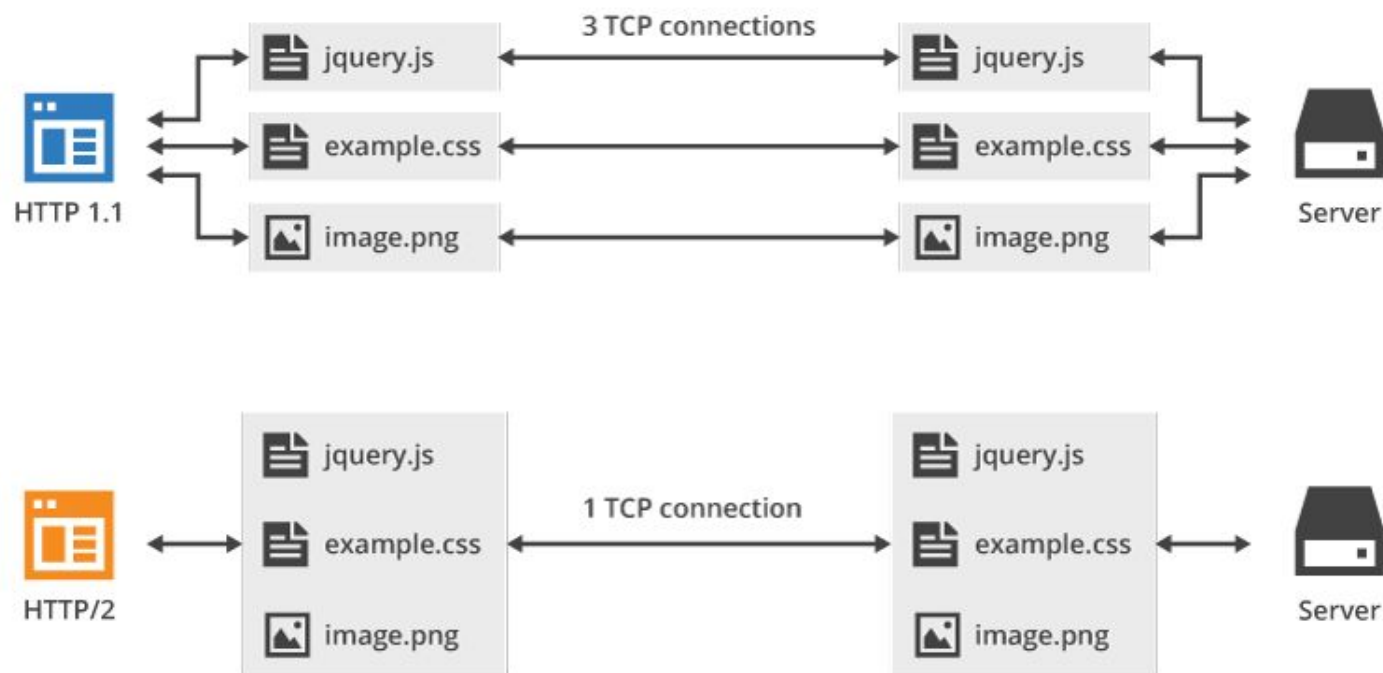
- [Пример](#)

```
1.  <svg version="1.1"
2.    xmlns="http://www.w3.org/2000/svg"
3.    xmlns:xlink="http://www.w3.org/1999/xlink"
4.  >
5.    <symbol id="first" viewBox="0 0 64 64">...</symbol>
6.    <symbol id="second" viewBox="0 0 64 64">...</symbol>
7.  </svg>
8.
9.  <svg>
10.    <use xlink:href="#first"></use>
11.  </svg>
12.
13.  <svg>
14.    <use xlink:href="#second"></use>
15.  </svg>
16.
17.
```

# Сетевое взаимодействие

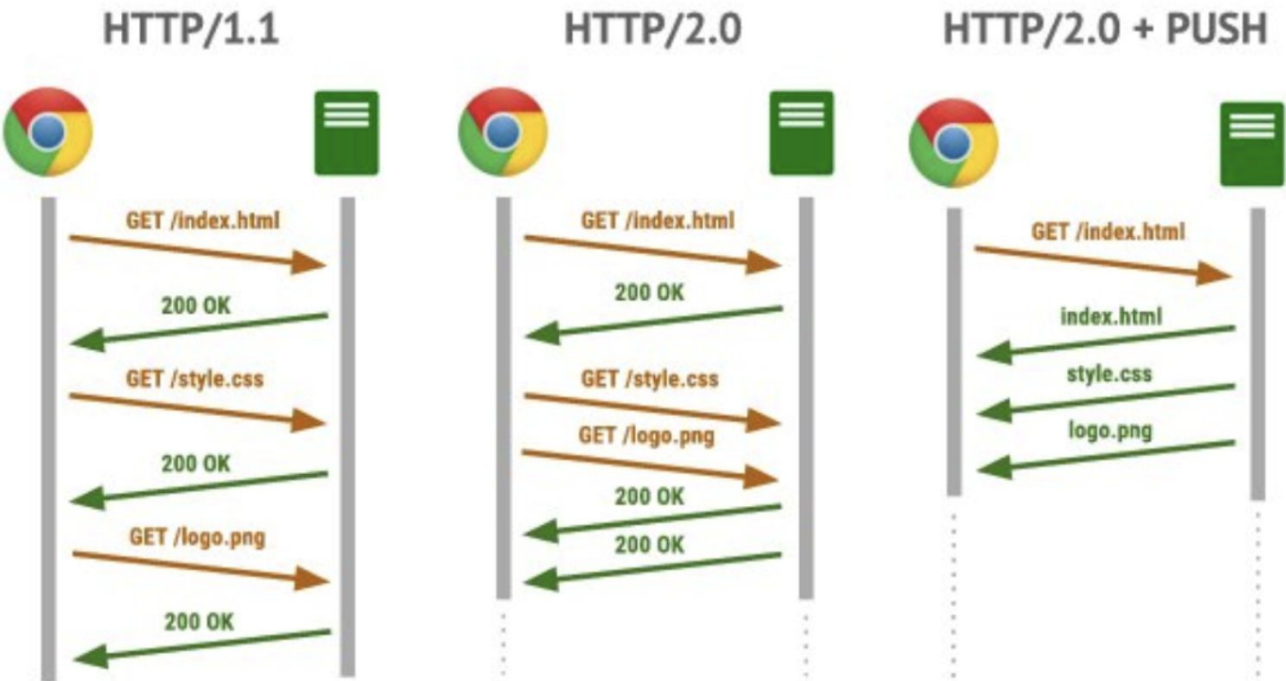
- HTTP/2
- одно соединение для нескольких запросов
- в отличие от Keep-alive ответы могут приходить одновременно

## Multiplexing



# Сетевое взаимодействие

- [HTTP/2 Server Push](#)
  - позволяет превентивно отправлять клиенту необходимые ресурсы
  - немного ломает логику кеширования на клиенте



HTTP/1.1	
Request HTML	/index.html
Request content	style.css
Request content	logic.js
Request content	image.jpg

HTTP/2 Without Server Push	
Request HTML	/index.html
Request content	style.css
Request content	logic.js
Request content	image.jpg

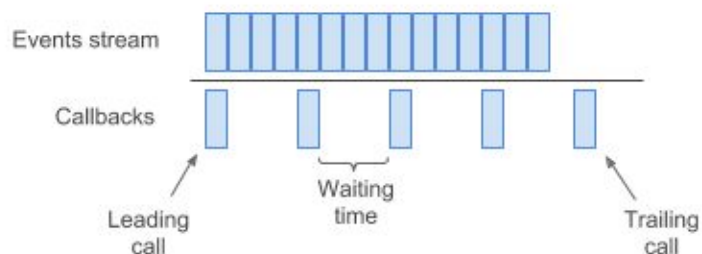
HTTP/2 With Server Push	
Request HTML	/index.html
Request content	style.css
Request content	logic.js
Request content	image.jpg

# Вычисления



# Вычисления

- throttling – вызов функции не чаще чем один раз за определенный интервал времени



- debouncing – вызов функции один раз, для серии вызовов происходящих чаще, чем заданный интервал времени



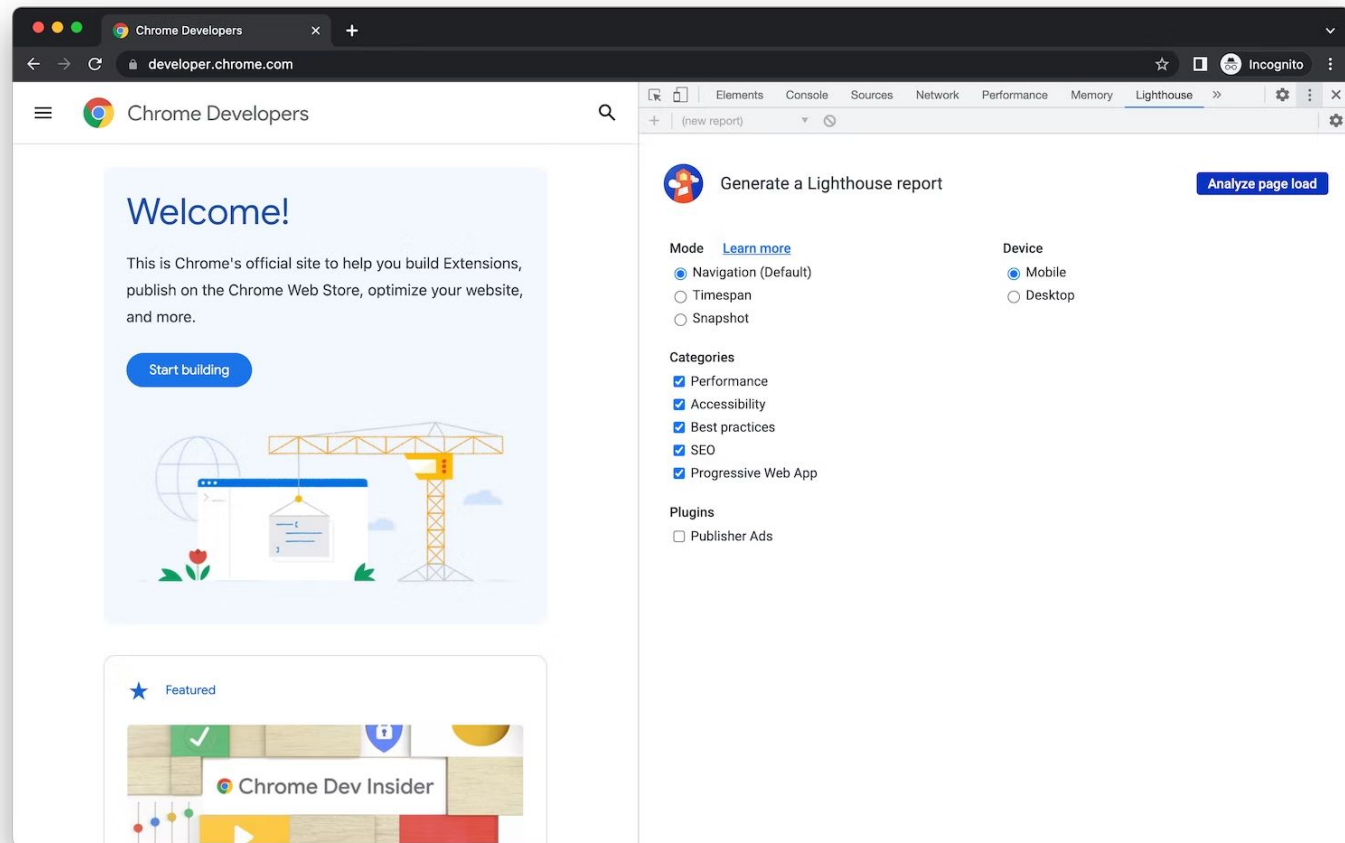
# Прочее

Дополнительные пути для оптимизации:

- Add long-term headers expiration dates
- Make fewer HTTP requests
- Avoid URL redirect
- Avoid empty SRC or HREF
- Remove duplicate JavaScript and CSS
- Make AJAX cacheable
- Avoid HTTP 404 (Not Found) error
- Remove unsupported components
- Use cookie-free domains
- Reduce cookie size
- Remove unnecessary CSS rules
- Don't scale images in HTML
- Reference images in the HTML
- Reduce DNS lookups
- Resize images

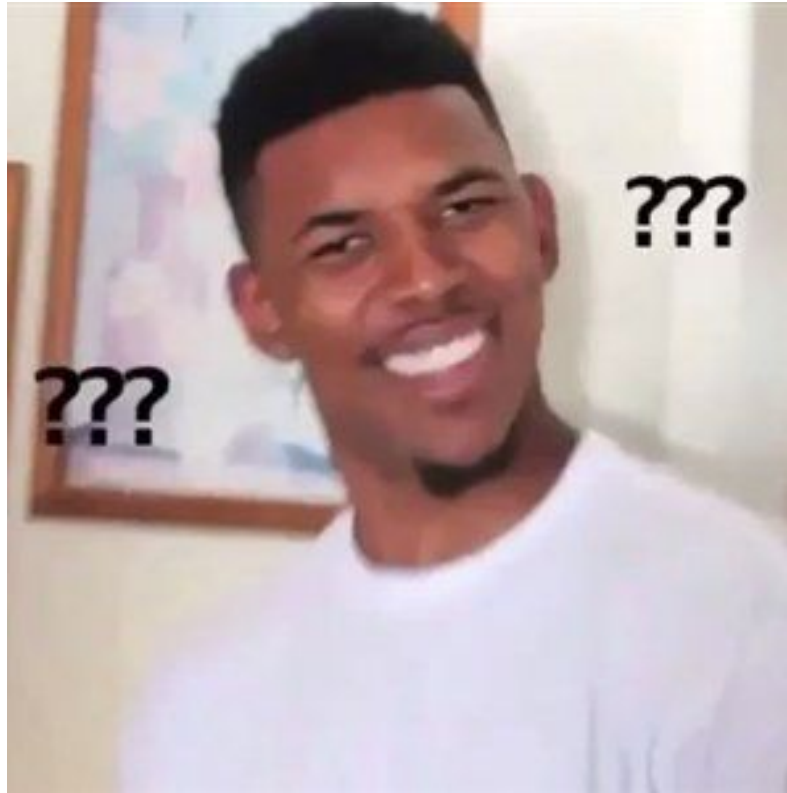
# Прочее. Инструмент

- [Lighthouse](#)



# Оптимизация?

Вопросы?



# Полезные ссылки

- <https://habr.com/ru/company/badoo/blog/320558/>
- <https://www.udacity.com/course/browser-rendering-optimization--ud860>
- <https://habr.com/ru/post/316618/>
- <https://tproger.ru/translations/how-to-boost-frontend/>
- <https://www.smashingmagazine.com/2020/01/front-end-performance-checklist-2020-pdf-pages/>
- <https://www.imperva.com/learn/performance/front-end-optimization-feo/>
- <https://techbeacon.com/app-dev-testing/23-front-end-performance-rules-web-applications>
- <https://mentormate.com/blog/front-end-optimization-habits-effective-developers/>
- <https://habr.com/ru/company/yandex/blog/195198/>

# Домашнее задание №11

1. Оптимизация
2. Производительность
3. Утечки

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в репозитории.

Срок сдачи

*5 декабря*

# Мем дня



# Спасибо за внимание!



# Пока!

Присоединяйтесь к сообществу про образование в VK

- [VK Образование](#)

