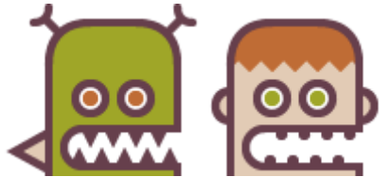
CSS-в-JS, HTML-в-JS, ВСЁ-в-JS

Все гораздо проще, когда всё вокруг JavaScript

Алексей Иванов, *Злые Марсиане*



Чем занимаются Марсиане

Gett

GROUPON

ebay™ SOCIAL

О чем этот доклад

Часть 1.

Серверные приложения

Сокращаем CSS-классы

Было:

```
.block__element_modificator {...}
```

Стало:

```
.aBc {...}
```

Сокращаем CSS-классы

Замена в CSS

1. Собрать все CSS в один файл.
2. Заменить имена классов.
3. Сохранить список замен:

```
{ 'block__element_modificator': 'aBc' }
```

Сокращаем CSS-классы

Замена в HTML

Заменить все вхождения:

```
01. <div class="block__element block__element_modificator">
```

Включая составные имена:

```
01. "block__element" + "_modificator"
```

Сокращаем CSS-классы

Замена в JavaScript

Найти и заменить имена всех классов:

```
01. var className = "block__element_modificator";
02. $elem.addClass(className);
```

Не заменить ничего лишнего:

```
01. var block = ...;
```

Убираем лишний CSS

Легаси, Bootstrap, Font Awesome

Убираем лишний CSS

Что нужно сделать?

1. **Основная задача:**

   Удалить лишние правила для одиночных классов, id и тегов.

2. **Задача со звездочкой:**

   Удалить несуществующие комбинации: `.class1 .class2 {}`.

Убираем лишний CSS

Парсим HTML

1. Понять структуру HTML для каждой страницы с учетом состояний (авторизован, неавторизован, акции, попапы).
2. Получить список классов, id и тегов. Лучше в виде дерева.

Убираем лишний CSS

Парсим JavaScript

1. Понимаем по JS как он может менять наш HTML:

   Добавление классов, добавление элементов, и т. д.

2. Дополняем наше дерево возможными состояниями.

Убираем лишний JavaScript

Убираем лишний JavaScript

1. Удалить то, что не касается DOM: переменные, функции, и т. д.
2. Удалить то, что касается DOM:
    1. Найти какие функции и методы влияют на DOM.
    2. Найти может ли этот DOM быть на странице.
    3. Удалить ненужное.

Отдельная сборка для лендинга

Или не лендинга

Отдельная сборка для лендинга

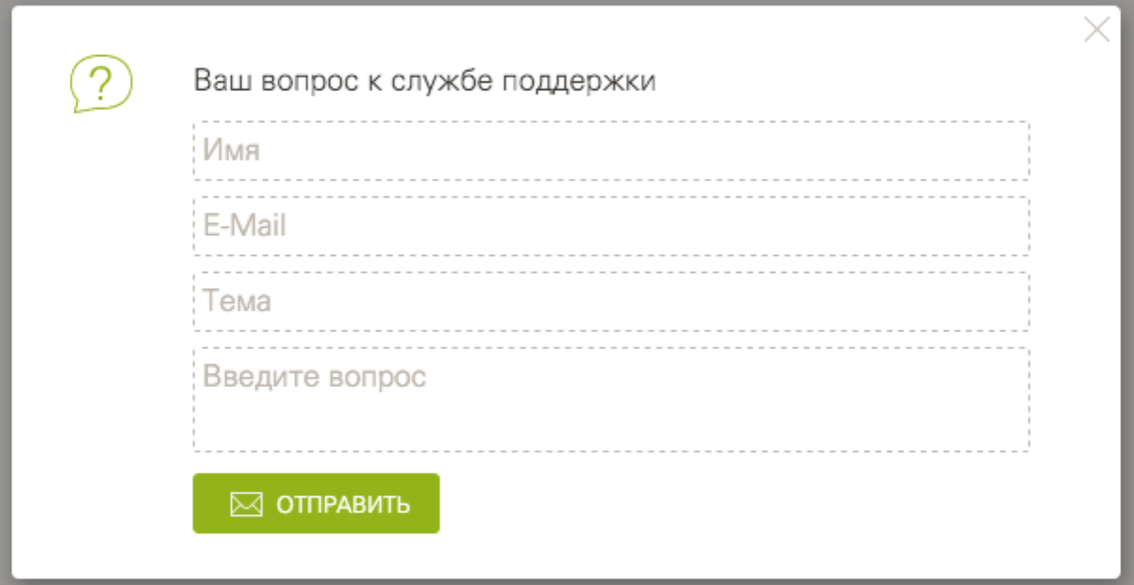
1. Научить наши инструменты понимать что такое страница.
2. Построить HTML-дерево страницы во всех состояниях.
3. Найти какой JavaScript может её изменять.
4. Найти CSS и JavaScript который нужен для этой страницы.

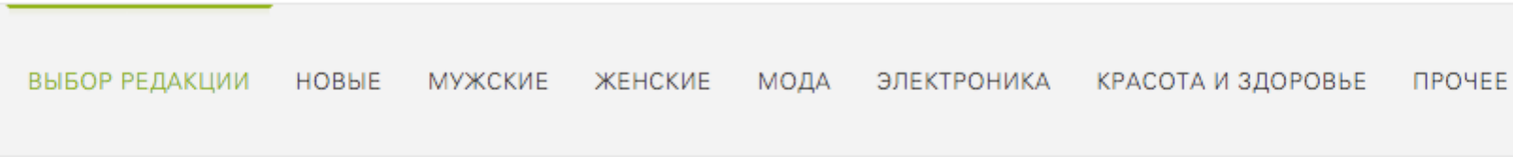
Подсвечиваем неиспользуемый код в IDE

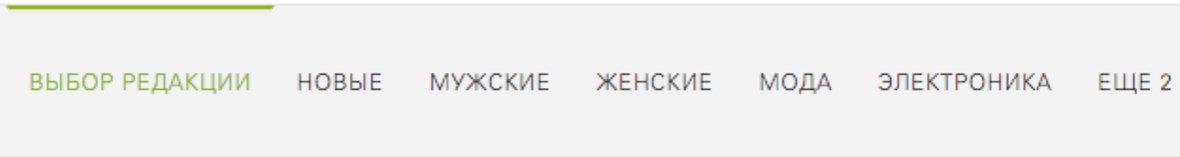
Переносим код между проектами

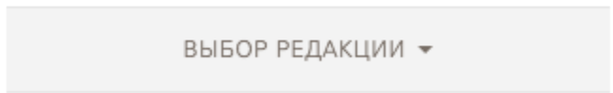
Нытьё

Почему все это так сложно?


Ваш вопрос к службе поддержки
Имя
E-Mail
Тема
Введите вопрос
ОТПРАВИТЬ


ВЫБОР РЕДАКЦИИ
НОВЫЕ
МУЖСКИЕ
ЖЕНСКИЕ
МОДА
ЭЛЕКТРОНИКА
КРАСОТА И ЗДОРОВЬЕ
ПРОЧЕЕ


ВЫБОР РЕДАКЦИИ
НОВЫЕ
МУЖСКИЕ
ЖЕНСКИЕ
МОДА
ЭЛЕКТРОНИКА
ЕЩЕ 2


ВЫБОР РЕДАКЦИИ

Часть 2.

Параллельный мир SPA

Зависимости в JavaScript

RequireJS u Browserify

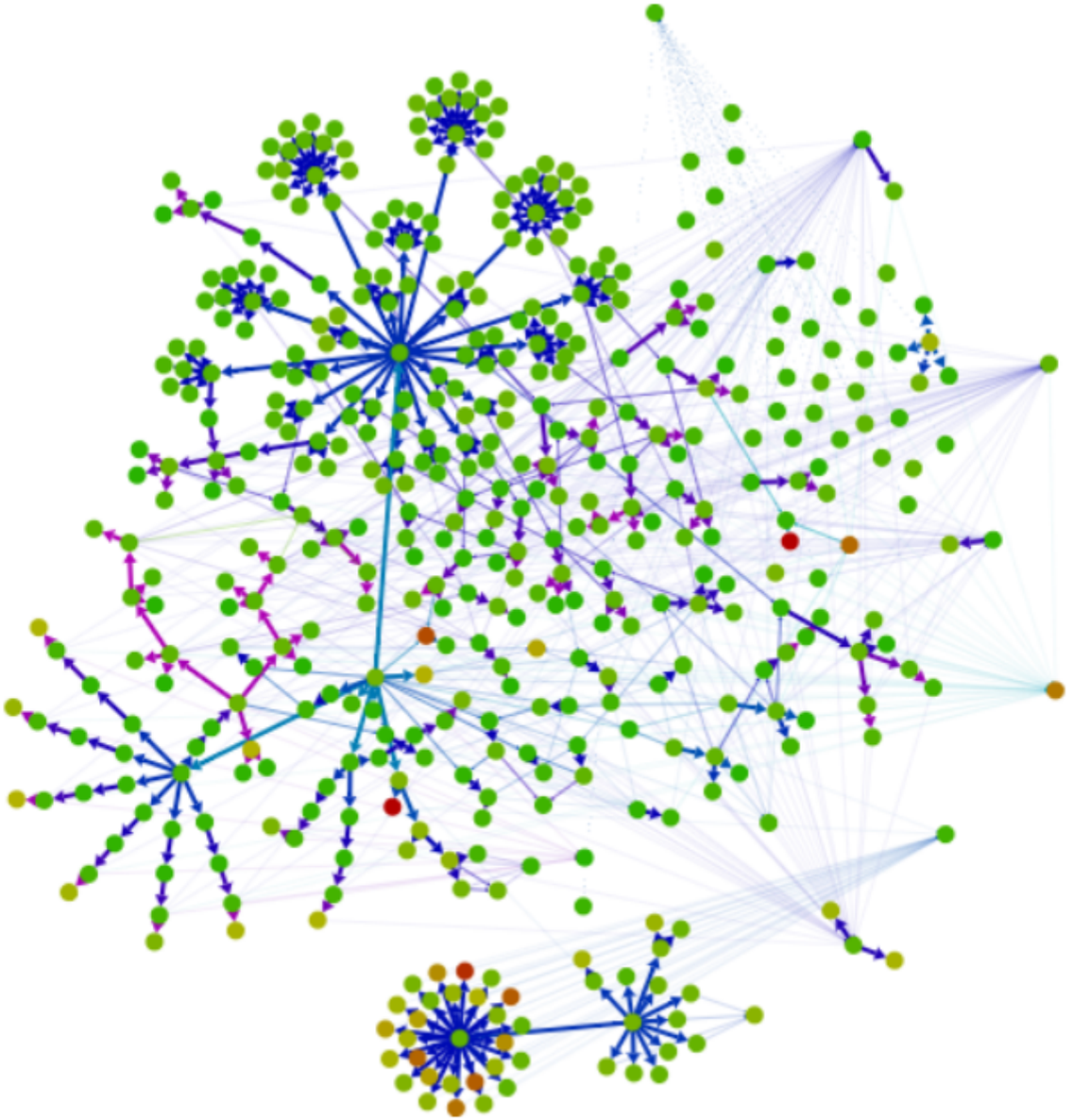
Пример Browserify

file1.js:

```
01. var MyModule = {};
02. MyModule.method = function () {...};
03. module.exports = MyModule;
```

file2.js:

```
01. var something = require('./file1');
02. something.method();
```



Картинка: https://github.com/unindented/webpack-presentation

Dead code elimination

UglifyJS

Пример удаления кода

До UglifyJS:

```
01. function (longName) {
02.     var a = 10;
03.     return longName;
04. }
```

После UglifyJS:

```
01. function (b) { return b; }
```

HTML-B-JS

React, Riot, Hyperscript

React и .jsx

```
01. var Example = React.createClass({
02.     onClick: function () { alert('Hello world!'); },
03.     render: function () {
04.         return <div onClick={this.onClick}>Hello!</div>;
05.     }
06. });
```

React и скомпилированный .jxs

```
01. var Example = React.createClass({
02.     onClick: function () { alert('Hello world!') },
03.     render: function () {return (
04.         React.createElement(
05.             'div',
06.             { onClick: onClick },
07.             "Hello!"
08.         )
09.     );}
10. });
```

Не только JS

Webpack

Webpack и require

Подгрузит в шапку, добавит в граф пути из `url()` и `@import`:

```
01. var style = require('style.css');
```

Подгрузит как JavaScript объект:

```
01. var json = require('data.json');
```

Положит в папку для готовых ассетов, отдаст путь:

```
01. var img = require('kitten.png');
```

CSS-в-JS

JSS u CSS Modules

Подключение стилей

```
01. var style = require('style.css');
02. var Example = React.createClass({
03.     render: function () {
04.         return (
05.           <div className={style.hello}>Hello!</div>
06.           <div className={style['hel'+'lo']}>Hello!</div>
07.         );
08.     }
09. });
```

Пример JSS

Стили:

```
01. export hello {
02.     fontSize: 12
03. }
```

Скомпилированные стили:

```
01. .hello--jss-0-0 { font-size: 12px; }
```

Пример CSS Modules

Стили:

```
01. .hello { font-size: 12px; }
```

Скомпилированные стили в разработке:

```
01. .style_hello_b8bW2Vg3fwHoz0 { font-size: 12px; }
```

Полная сборка:

```
01. .oz0 { font-size: 12px; }
```

Tree shaking

Rollup u Webpack 2

Tree shaking

require

file1.js

```
01. module.exports = {
02.     header:  "header",
03.     footer:  "footer"
04. }
```

file2.js

```
01. var styles = require('./file1');
```

Tree shaking

import и export в ES6

file1.js

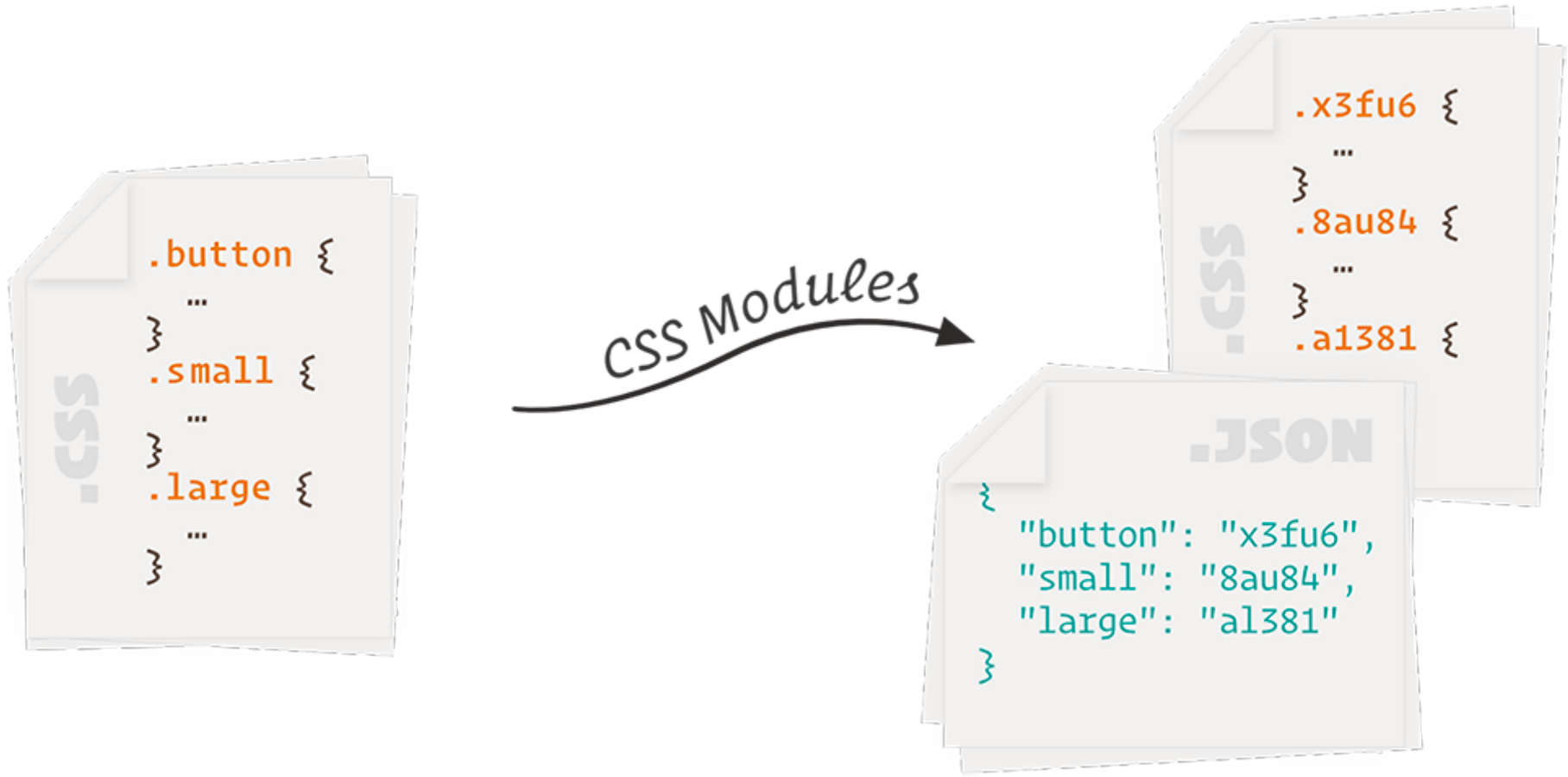
```
01. export const header  = "header";
02. export const content = "content";
03. export const fooner  = "footer";
```

file2.js

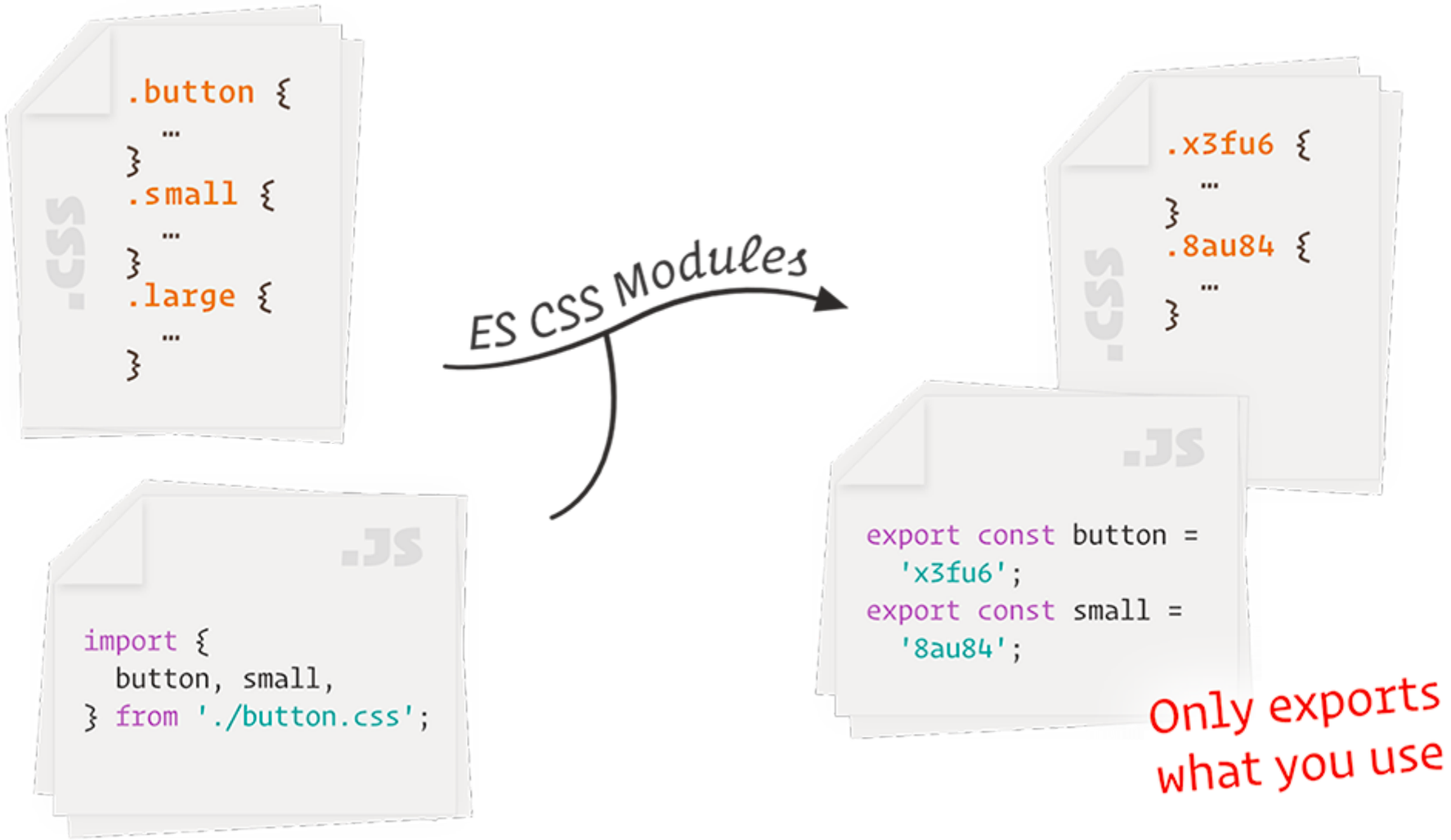
```
01. import {header, footer} from file1;
```

Tree shaking и CSS Modules

es-css-modules



Картинка: https://github.com/jacobp100/es-css-modules



Картинка: https://github.com/jacobp100/es-css-modules

Оптимизация CSS

CSSO

Настройки CSSO

Фильтрация селекторов:

```
{
    "tags": ["ul", "li", ...],
    "classes": ["module2-baz", ...]
}
```

Скоупы:

```
"scopes": [
    ["module2-baz", "module2-qux"]
]
```

Чтение и правка JavaScript

Babel.js не только для es6

Задачи для Babel.js

1. Сбор статистики по подключению стилей.
2. Оборачивание каких-то фрагментов кода своими функциями.
3. Свой синтаксис в JS.
4. И т. д.

Решение проблем из первой части в SPA

Сокращаем CSS-классы: CSS Modules в dev-режиме (также получаем в процессе автоматическую изоляцию CSS)

Убираем лишний CSS: Webpack + JSS, либо Webpack + CSS Modules ES, либо Webpack + Babel.js + CSSO

Убираем лишний JS: Webpack 2/Rollup + UglifyJS

Сборка под отдельную страницу: Webpack

Перенос кода между проектами: Подключение всех зависимостей через import/require.

Часть 3.

К чему все это?

Границы между HTML, CSS и JS – искусственные

Разные технологии должны друг о друге знать чтобы эффективно друг с другом работать

SPA умеют эти знания получать и потом эффективно с ними работать

1. import/export,
2. Граф зависимостей,
3. Приведение при компиляции к одному типу,
4. Dead code elimination и tree shaking на основе информации из дерева,
5. Babel.js для изменения и переписывания js-бандла.

Эти штуки могут работать не только для SPA

Где это может еще пригодиться

1. Интеграция с IDE.
2. Рендеринг на сервере:
    1. Шаблоны отделены от стилей и логики.
    2. JS-шаблонизаторы и инструменты сборки сейчас сильно заточены под SPA.
3. Огромная куча других инструментов для оптимизации и анализа.

Вопросы?

Я всегда рад поговорить с кем-нибудь про этих штуки.

- Подходите обсуждать на afterparty.
- Пишите в:
  - Twitter: **@iadramelk**
  - Facebook: **@iadramelk**