Frontend Performance S.Center 🚀

Савичев Игорь

@savichev

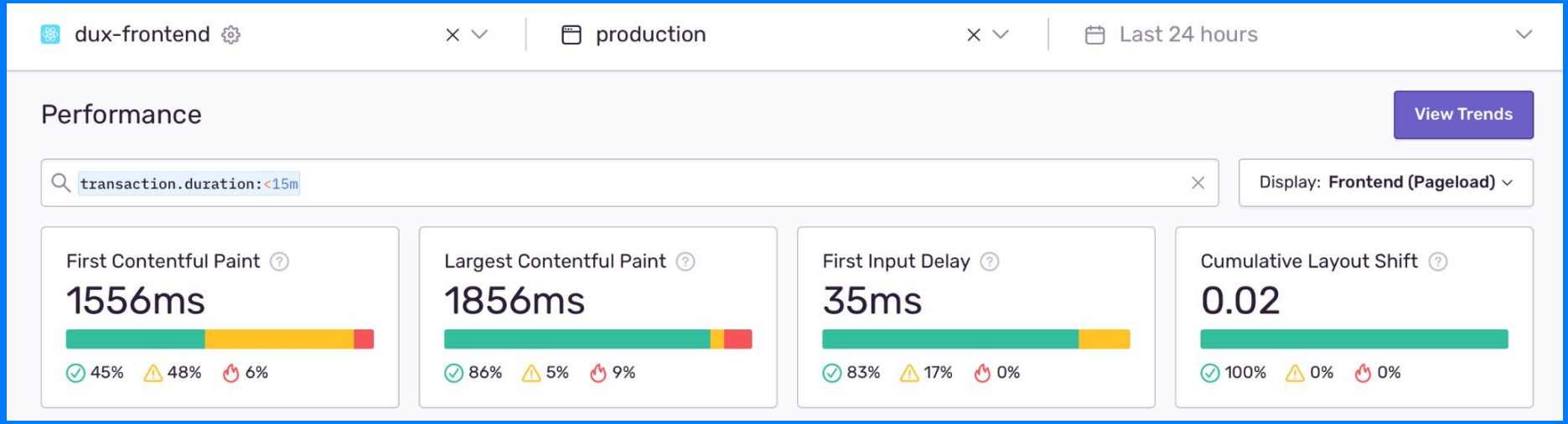
самолет

LightHouse

Проанализировали производительность сайта с помощью **LightHouse**

Раздел	паспорта	объекты учета	документы
Было	44 Performance, 83 Best Practices	36 Performance, 67 Best Practices	46 Performance, 83 Best Practices
Стало	55 Performance, 100 Best Practices	57 Performance, 92 Best Practices	58 Performance, 100 Best Practices

Sentry Performance

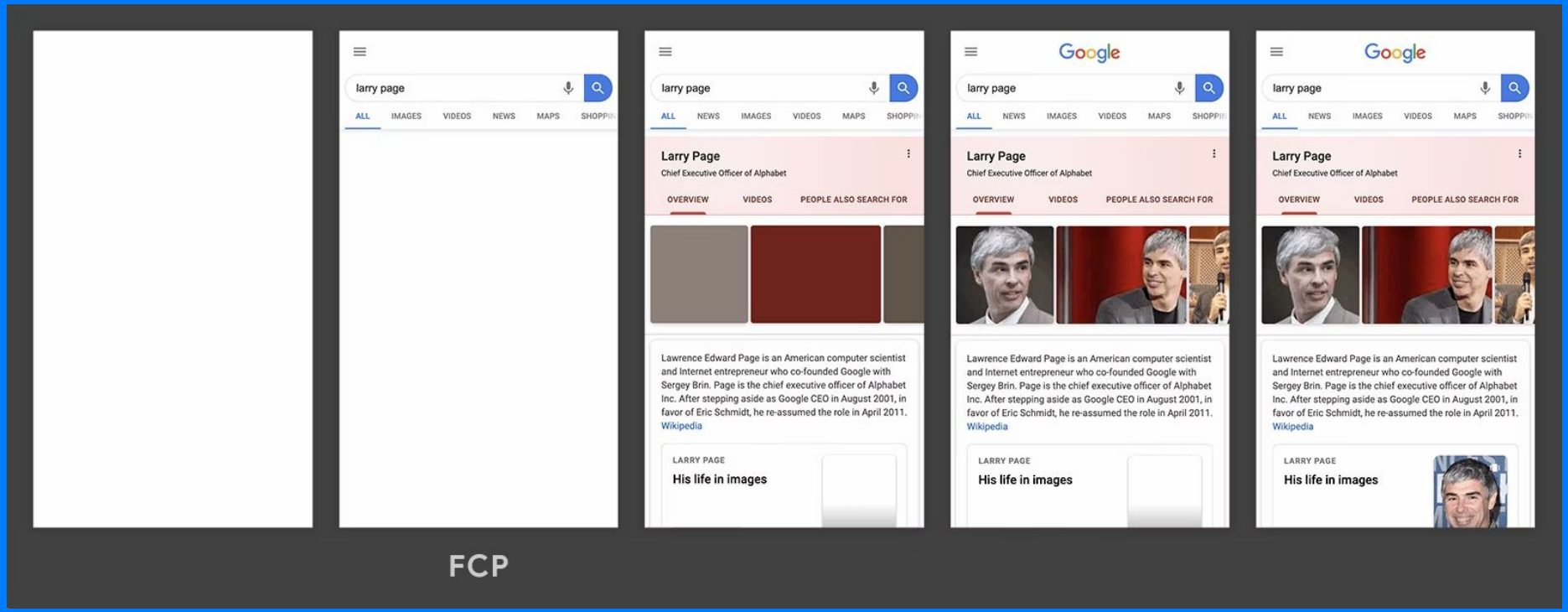


Метрики

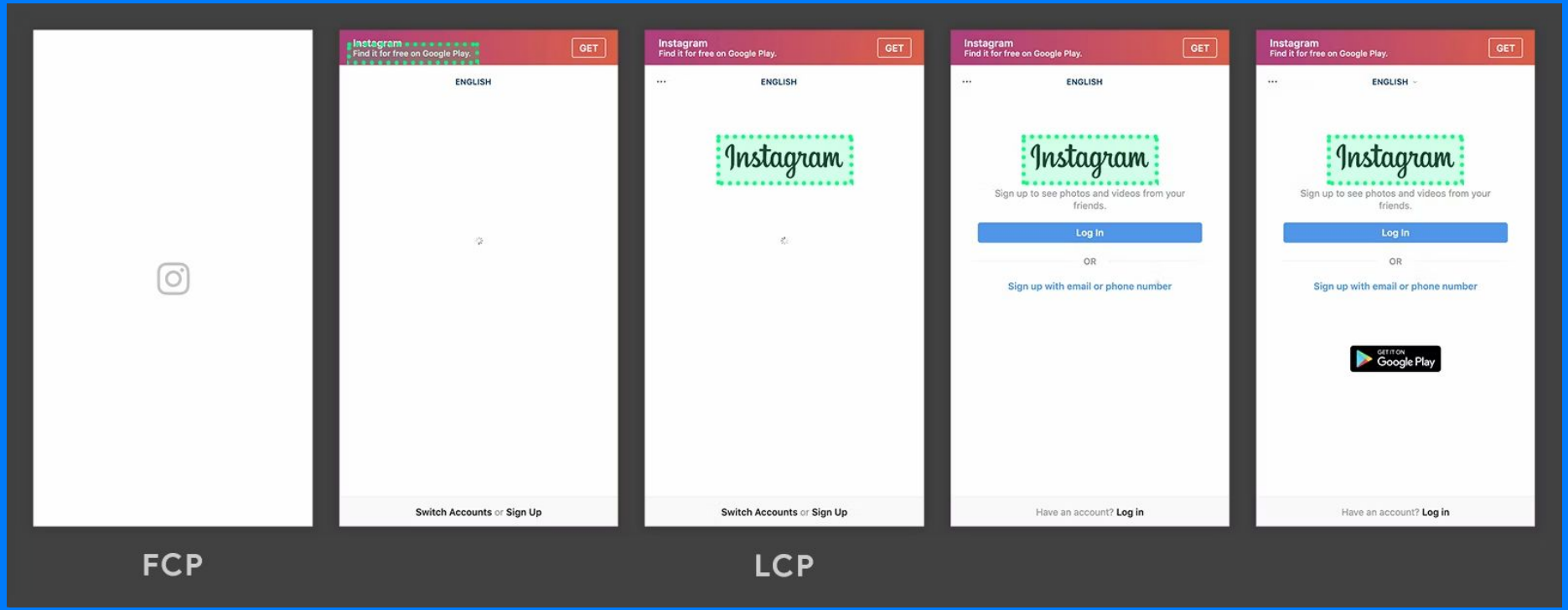
*FCP – измеряет время с момента начала загрузки страницы до момента, когда какая-либо часть содержимого страницы отобразится на экране

*LCP – измеряет время загрузки основного контента

Метрика First Contentful Paint (FCP)



Метрика Large Contentful Paint (LCP)



Что можно проверить у себя на проекте?

- Есть ли минификация JS/CSS кода
  - **TerserWebpackPlugin**
  - **CssMinimazerPlugin**
- Используются ли современные форматы шрифтов
  - **woff2**
  - **font-display: swap;**
- Есть ли оптимизация изображений
  - **tinypng**, **svgoptimazer**
- Есть ли неиспользуемый код
  - работает ли **tree-shaking**, т.е. правильный **import** библиотек
- Есть ли возможность подключить **CDN** для статики
- Есть ли алгоритмы сжатия (**ZIP**/**Brotli**) в **nginx**
  - пример
- Есть ли возможность использовать **React.LazyLoad**
- Есть ли возможность применить
  - это **<link />**
  - это **<script />**
- Есть ли возможность поднять целевую сборку **ECMA** стандарта
- Есть ли возможность отказаться от избыточных библиотек в бандле
  - **lodash/moment** и прочие
- **NODE_ENV** = “production”

Минификация **JS/CSS** кода

```
optimization: {
  minimize: true,
  minimizer: [
    new TerserWebpackPlugin( options: {
      parallel: true,
      terserOptions: {
        ecma: 2020,
      },
    }),
    new CssMinimizerPlugin(),
  ],
},
plugins: [new MiniCssExtractPlugin()],
```

webpack.config.js

Современные форматы шрифтов **woff2**

	OTF	WOFF2	SAVINGS
Single weight	234kb	111kb	↓ 52%
Variable font	405kb	112kb	↓ 72%

Подключая шрифт используем свойство **font-display: swap;**

Оптимизация изображений

Существует множество сервисов для статической оптимизации изображений

- **tinypng**
- **svgoptimazer**

Подключая картинки используем свойство атрибут **loading=”lazy”** тэга **<img/>**

TreeShaking

С нововведением **import/export** в **ES6** появилась возможность забирать в бандл только то, что реально используется в коде

- включаем **mode: “production”**
- используем синтаксис **import/export**

```
module.exports = {
  // ...,
  mode: 'production',
  // ...,
};
```

webpack.config.js

```
export function add(a, b): number {
    console.log("add");
    return a + b;
}
no usages
export function minus(a, b): number {
    console.log("minus");
    return a - b;
}
```

utils.ts

```
import { add } from "./utils";

add(1, 2);
```

index.ts

Используем **CDN** для статики

Краткая инструкция:

- Заказываем бакет у **DevOps**'ов
- Меняем **publicPath** для статики
- Настраиваем **job**'у заливки статики перед деплоем окружения
- включаем галочку сжатие на источнике **gzip/brotli**

```
31  output: {
32    publicPath: `https://cdncenter.constructo.tech`,
33  },
```

webpack.config.js

```
28  AWS_ACCESS_KEY_ID=$SEL_S3_ACCESS_KEY_ID \
29  AWS_SECRET_ACCESS_KEY=$SEL_S3_SECRET_ACCESS_KEY \
30      aws --endpoint-url=$S3_ENDPOINT_URL s3 sync build $S3_BUCKET_NAME/$S3_PREFIX
```

job.yaml

холодная загрузка **main.js**

5 sec → 100 ms

Если нет **CDN**, только **nginx**

```
38  brotli on;
39  brotli_static on;
40  brotli_comp_level 6;
41  brotli_min_length 1024;
42  brotli_types
43    application/*
44    font/*
45    image/svg+xml
46    text/*
```

nginx.conf

```
87  new CompressionPlugin({
88    test: /\.(js|css|html|svg)$/,
89    algorithm: 'brotliCompress',
90    compressionOptions: {
91      params: {
92        [zlib.constants.BROTLI_PARAM_QUALITY]: 11,
93      },
94    },
95    threshold: 1024,
96  }),
```

webpack.config.js

холодная загрузка **main.js**

5 sec → 1.5 sec

Разбиваем код на чанки

Краткая инструкция:

- Используем плагин **splitChunks**
- По возможности оборачиваем страницы в **React.LazyLoad**

```
    splitChunks: {
      chunks: 'async',
      maxInitialRequests: Infinity,
      minSize: 0,
    },
  },
```

webpack.config.js

```
import { lazy } from 'react';

const MarkdownPreview = lazy(() => import('./MarkdownPreview.js'));
```

lazyMarkdownPreview.js

link “rel=”, **script** async/defer

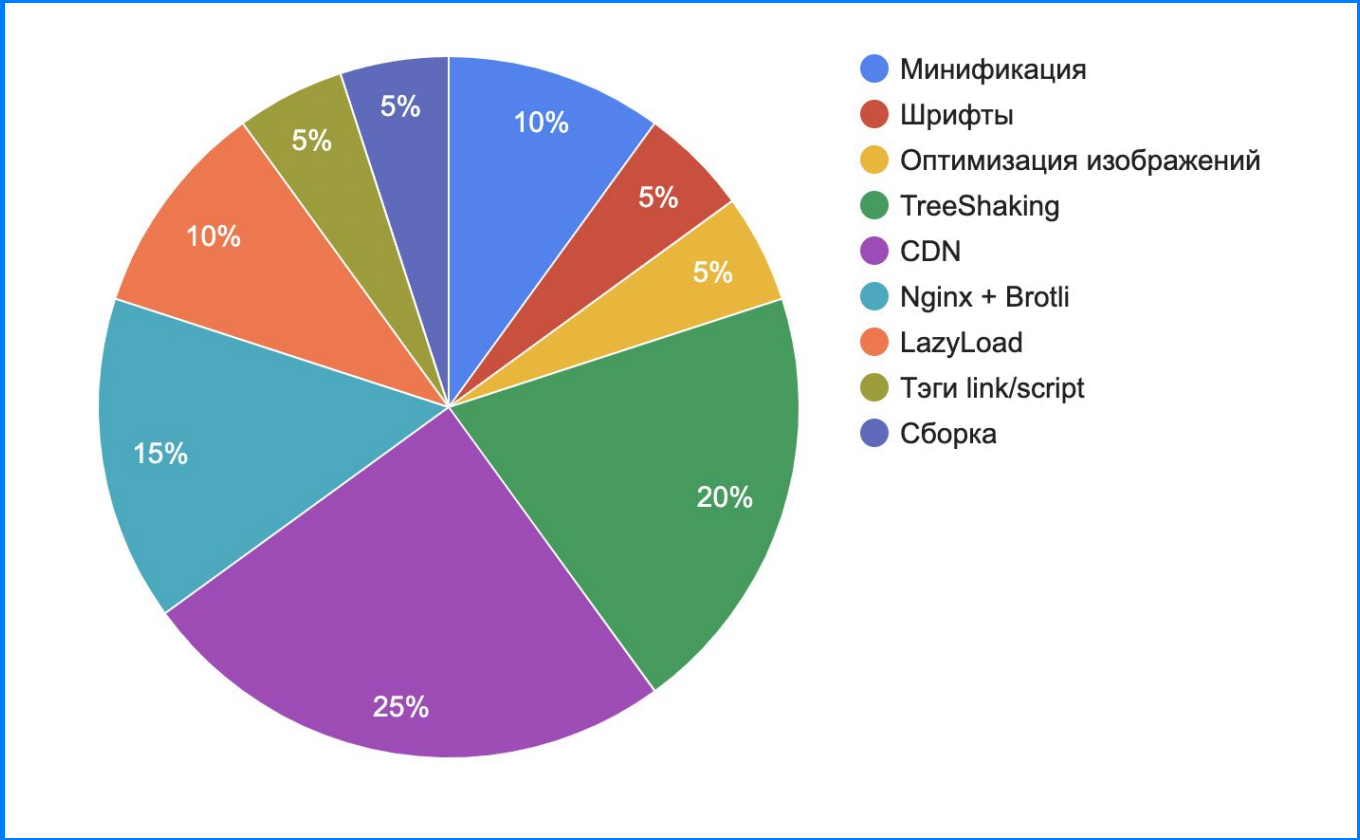
- **<link rel= "preload">** — когда вам понадобится ресурс через несколько секунд
- **<link rel= "prefetch">** — когда понадобится ресурс на следующей странице
- **<link rel= "preconnect">** — когда вы знаете, что вам скоро понадобится ресурс, но вы ещё не знаете его полный URL
- **<link rel= "prerender">** — когда вы уверены, что пользователи перейдут на определённую страницу, и хотите ускорить её отображение

- **<script async>** — Порядок загрузки (кто загрузится первым, тот и сработает).
- **<script defer>** —Порядок документа (как расположены в документе).

Рекомендации к **сборке**

- **NODE_ENV = “production”**
- **moment/lodash** и прочие библиотеки избыточны
- Проверить есть ли возможность поднять целевую сборку **ECMA**

Фича/импакт круговая



После применения оптимизаций

main.js

15.3 MB → 1.6 MB

Сэкономили **956%** сетевого трафика

С чего начать?

webpack.analyze.js