Использование изоморфного подхода для решения проблемы SEO-ориентированности веб-приложений на основе технологий Javascript

А. О. Суворов[I], А. А. Петренко[II], А. В. Аликин[III]

[I] Национальный исследовательский университет «Высшая школа экономики», г. Москва, Россия
[II] Пермский национальный исследовательский политехнический университет, г. Пермь, Россия
[III] Акционерное общество «ЛАНИТ», г. Москва, Россия

Аннотация. Рассматриваются вопросы создания одностраничных веб-приложений на JavaScript-фреймворков, таких как React, Angular, Vue таким образом, чтобы SEO-ориентированность веб-приложений оставалась такой же, как и при использовании статичного контента. Рассмотрен изоморфный подход к разработке одностраничных приложений, основанный на принципе его устройства и стратегиях его использования, приведены принципы построения такого приложения и примеры блока программного кода, а также визуальное сравнение ранжирования стандартного одностраничного приложения и аналогичного приложения, построенного с использованием изоморфной архитектуры.

Ключевые слова: поисковая оптимизация (SEO), PageRank, релевантность, изоморфизм, рендеринг, JavaScript, React, одностраничное приложение (SPA).

DOI 10.14357/20718632210410

Введение

Большую часть современной разработки в области информационных систем составляет веб-разработка. Для решении этих задач наиболее часто используются JavaScript-фреймворки.

Для того, чтобы тот или иной сайт был доступен для маркетингового продвижения за счет повышения рейтинга в результатах поисковых запросов используются специальные SEO методы. SEO («Search Engine Optimization») – англоязычная аббревиатура, которая в переводе на русский означает поисковая оптимизация. SEO представляет собой комплекс разного рода мероприятий по оптимизации (внутренней и внешней) для поднятия позиций сайта в результатах выдачи поисковых систем по определённым запросам пользователей [1]. Например, Google отображает в результатах поиска ссылки на страницы, которые, исходя из своих алгоритмов, считает релевантными (соответствующими запросу) и авторитетными (качественными и уникальными с точки зрения контента и состава гиперссылок).

Поисковая оптимизация применяется, чтобы обеспечить доступность сайта для поисковых систем, и чтобы повысить вероятность того, что сайт будет найден поисковой системой среди множества других сайтов. SEO, обычно, представляет собой набор так называемых «белых методов», которыми пользуются производители веб-контента и веб-мастера для повышения ранжирования сайта [2, 3].

Однако иногда, ввиду широкой развитости некоторых веб-технологий, в частности, JavaScript-фреймворков, SEO-оптимизация

веб-приложений затрудняется или даже полностью не работает, поскольку в погоне за скоростью работы веб-приложений приходится пересматривать подходы в разработке веб-сайтов.

В данной статье будут рассмотрены вопросы использования подходов, позволяющих современным веб-приложениям, использующим JavaScript-фреймворки, не терять свою SEO-ориентированность.

1. Современная поисковая оптимизация

Работа поисковых роботов сложна по своему устройству. Они используют множество разных алгоритмов, которые усложняются с каждым днём.

Поисковые системы используют переменную PageRank для того, чтобы определить релевантность той или иной страницы и, тем самым, поднять её в рейтинге ранжирования.

Общую формулу PageRank для любой страницы можно выразить следующей формулой:

$$PR = \sum_{u \in B_u} \frac{PR(v)}{L(v)}. \quad (1)$$

Таким образом, значение *PageRank* для страницы *u* зависит от значений *PageRank* для каждой страницы *v*, содержащихся в наборе B_u (набор всех страниц, ссылающихся на страницу *u*), деленных на число *L(v)* ссылок на странице *v* [4].

Предполагается, что при расчете PageRank страницы, у которых нет исходящих ссылок, свяжутся со всеми остальными страницами коллекции. Поэтому их значения PageRank разделяются поровну между всеми остальными страницами. То есть, чтобы справедливо распределить рейтинг релевантности со страницами, которые не являются поглотителями, эти случайные переходы добавляются ко всем узлам в Интернете, с остаточной вероятностью, обычно равной 0,85 (d = 0,85), из расчета частоты, которую средний пользователь использует для своего браузера.

Итого, конечное уравнение выглядит следующим образом:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M_{Pi}} \frac{PR(p_j)}{L(p_j)} \quad (2)$$

где *PR* – PageRank,

p_i – узел (страница), чей *PR* рассчитывается в данный момент,

d – коэффициент затухания, который показывает вероятность того, что пользователь запросит другую случайную страницу во время просмотра сайта. Принято считать коэффициентом затухания значение 0,85 согласно рекомендациям разработчиков данного алгоритма. А оставшаяся вероятность в 0,15 – это вероятность того, что пользователь не перейдёт по ссылке дальше, а закроет вкладку браузера,

N – общее количество активных узлов, участвующих в расчёте,

$p_j \in M_{p_i}$ – множество узлов (страниц), имеющих отношение к текущему узлу p_i,

$PR(p_j)$ – PageRank для каждой страницы p_j, которая ссылается на текущую p_i,

$L(p_j)$ – кол-во ссылок с узла p_j, ссылающегося в том числе и на узел p_i.

То есть PageRank использует весь контент веб-приложения/сайта, и на базе этого даёт ему соответствующую оценку. Поэтому важно, чтобы поисковому роботу Google был доступен весь контент веб-приложения в одну единицу времени. Иначе значение переменной PageRank будет ниже, чем могло бы быть, и соответственно, веб-приложение будет на более низких позициях при поисковой выдаче [5].

Что касается алгоритма ранжирования сайтов от Яндекс, то он был представлен в 2006 году. Одним из вариантов ранжирования является использование текста запроса. Для этого производится вычисление так называемого Score (метки) запроса [6].

Сама встречаемость слова (релевантность) определяется по следующей формуле:

$$W_{single} = \log(p) \cdot \frac{TF}{TF + k_1 + k_2 \cdot DocLength}, \quad (3)$$

где p – условное обозначение веса слова, для которого вычисляется релевантность,

TF – число вхождений леммы в документ,

k_1, k_2 – коэффициенты,

DocLength – длина документа в количестве слов.

Наличие всех слов в запросе определяется следующей формулой:

$$W_{AllWords} = 0,2 \cdot \sum \log(p_i), \quad (4)$$

где p_i – вес конкретного i-го слова в общем наборе слов запроса.

То есть, производится вычисление суммы всех слов в запросе. Но, по сути, главные алгоритмы Яндекс работают таким же образом, как и алгоритмы Google, то есть важно сделать так, чтобы роботам был доступен весь контент сайта в момент загрузки.

Данная особенность поисковых движков не позволяла раньше использовать веб-приложения на JavaScript-фреймворках в тех проектах, где SEO-ориентированность была определяющим фактором.

2. Разработка стандартных веб-приложений на JavaScript

Разработки на чистом JavaScript в сегменте крупных веб-приложений, практически нет. В основном используются фреймворки JavaScript. На сегодняшний день три самых популярных из них - React, Angular и Vue.

Стек технологий – это набор структур и инструментов, используемых для разработки программного продукта. Этот набор фреймворков и инструментов специально выбран для совместной работы при создании хорошо работающего программного обеспечения.

Вот несколько примеров широко используемых сегодня стеков технологий веб-разработки:

– MERN (MongoDB, ExpressJS, ReactJS, NodeJS);

– LAMP (Linux, Apache, MySQL, PHP);

– MEAN (MongoDB, ExpressJS, AngularJS, NodeJS) [7].

Например, стек MERN – это набор инструментов для веб-разработки. Он состоит из рабочих компонентов MongoDB, ExpressJS, ReactJS и NodeJS. Структура данного стека показана на Рис. 1.

Как показано на рисунке, пользователь взаимодействует с компонентами пользовательского интерфейса ReactJS, который работает в браузере. Его интерфейс обслуживается серверной частью приложения, находящейся на сервере, через ExpressJS, работающий поверх NodeJS [8].

Любое взаимодействие, вызывающее запрос на изменение данных, отправляется на сервер Express, который в свою очередь создан на основе NodeJS, который при необходимости получает данные из базы данных MongoDB и возвращает данные во внешний интерфейс приложения, которые затем отображаются конечному пользователю.

React – это JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. В основе всех React-приложений лежат компоненты [9]. Компонент – это автономный модуль приложения, который обладает собственным функционалом и представляет собой определенный отдельный элемент интерфейса (кнопка, поле ввода, строка навигации, пагинация и т.д.). Компоненты служат строительными блоками интерфейса программы. Одним из главных преимуществ React является высокая производительность. Стратегия взаимодействия в React с DOM и является основной идеей Virtual DOM от React (Рис. 2).

Virtual DOM представляет собой дерево узлов, в котором перечислены элементы, также их атрибуты и различный контент, такие как объекты и свойства. Метод *render* создает дерево узлов из компонентов React и обновляет это дерево в ответ на изменения в модели данных, вызванные действиями [10].

Каждый раз, когда базовые данные изменяются в приложении React, создается новое представление пользовательского интерфейса, виртуального DOM. Это всё в совокупности

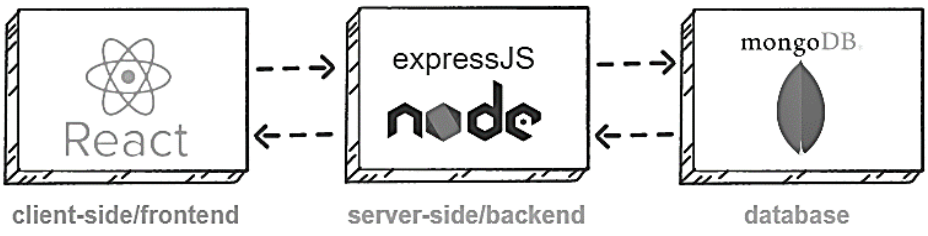

Рис. 1. MERN стек

образует стандартное SPA-приложение. На Рис. 3 показана схема создания стандартного веб-приложения на JavaScript-фреймворках, так называемый клиент-ориентированный подход.

Данный подход полностью отвечает концепции SPA-приложений, однако добавляет большую проблему – нарушение SEO-ориентированности приложения.

3. Существующие решения проблемы нарушения SEO-ориентированности веб-приложений

Существует несколько таких решений. Первое, и самое главное, это максимальное использование на клиенте статических страниц.

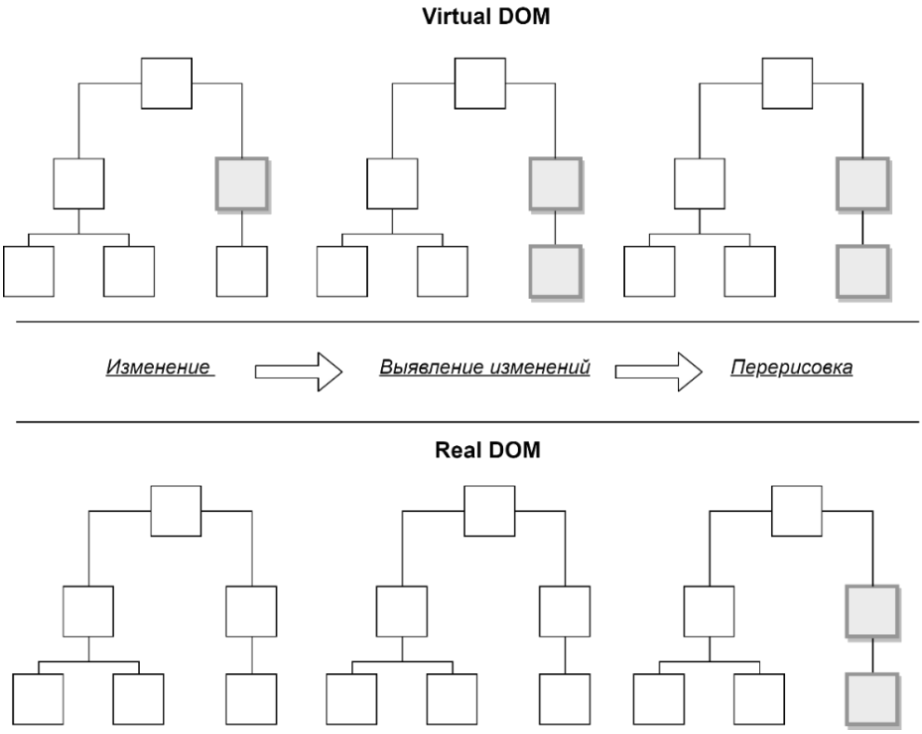

Рис. 2. Работа Virtual DOM в React

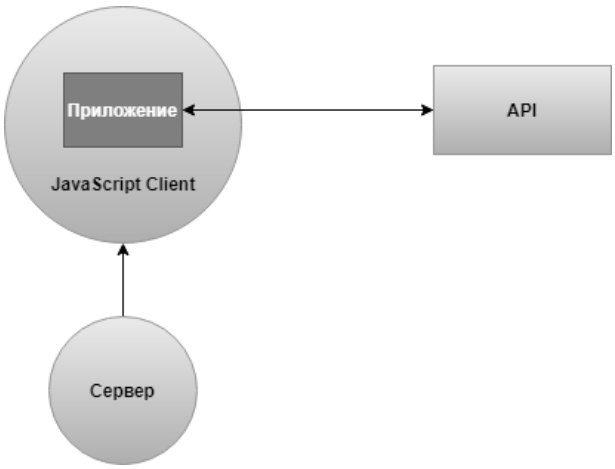

Рис. 3. Клиент-ориентированный подход

То есть весь основной контент пишется на HTML и CSS [7, 11]. Но в таком случае, все плюсы от использования JavaScript- фреймворков сходят на нет, так как скорость работы приложения становится намного меньше, и динамичность веб-приложений исчезает (Рис. 4). По сути, данный метод решения проблемы возвращает веб-разработку на 10-15 лет назад, когда вся сеть Интернет была статической, а JavaScript использовался только для добавления некоторой интерактивности на страницу, например, работа слайдеров и меню.

Другой метод решения данной проблемы – пререндеринг (предварительный рендеринг). Данный метод позволяет использовать некоторые готовые библиотеки, благодаря которым удаётся перехватывать запрос поискового робота. То есть, разработанное веб-приложение понимает, что сейчас его сканирует поисковый робот и выдаёт роботу статический HTML (Рис. 5). Данный метод довольно прост в реализации, и как было сказано выше, имеет некоторые готовые решения [11].

Однако поисковый робот на самом деле состоит из двух специализированных малых роботов. Первый заходит на страницу и сканирует статику, то есть HTML. После него заходит на страницу второй робот и проходит уже по JavaScript сценариям. То есть, разработка данным методом не даст высокую гарантию правильного сканирования, поскольку на момент работы первого поискового робота на странице может не быть никакого контента, а веб-приложение не поймёт, что зашёл робот.

4. Решение SEO проблемы с помощью изоморфного JavaScript

У архитектуры SPA-типа огромный минус – это долгий первый запуск, поскольку нужно ждать, пока всё приложение загрузится с сервера. Исследования показывают, что ожидание загрузки более двух секунд не является приемлемым для большинства пользователей, и они могут просто закрыть страницу не дождавшись

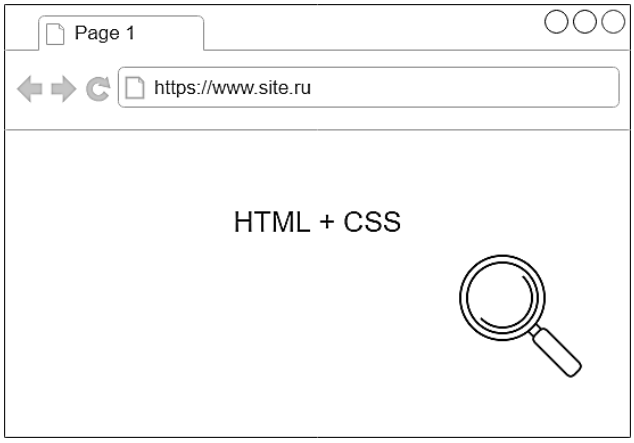

Рис. 4. Разработка на статике

Рис. 5. Пререндеринг

загрузки. Одним из преимуществ изоморфного подхода к созданию приложений является то, что рендеринг происходит на сервере, и становится возможным отображать компоненты после загрузки страницы на клиенте [12].

Работа изоморфного приложения заключается не в замене традиционного серверного API, а в устранении долгого ожидания загрузки страницы. К тому же SPA-приложения плохо взаимодействуют с SEO-инструментами и могут быть не проиндексированы полностью поисковыми системами. При изоморфных приложениях изначально отправляется обычная HTML страница, которая быстро загружается и показывается клиенту. Когда обрабатываются компоненты React (или другого js-фреймворка, далее именно на примере React) на сервере и отправляется HTML-код клиенту, React на стороне клиента замечает, что HTML уже существует [9]. Данный фреймворк просто прикрепляет обработчики событий к существующим элементам по необходимости [13]. Этот подход увеличивает производительность приложения, также ускоряя процесс разработки (Рис. 6).

Для работы с данными в изоморфном приложении выделяют две стратегии:

1. Загрузка пользовательских данных до клиентского отображения. Преимущество в том, что на клиент приходит полный контент, но, если загрузка данных на сервере будет слишком долгой, пользователь будет видеть пустую страницу и, скорее всего, покинет приложение.

2. Загрузка пользовательских данных после клиентского отображения. С данной стратегией интерфейс получается более отзывчивым, но после первого отображения необходимо производить ререндеринг (повторный рендеринг) представления с дозагруженными данными.

Каждая из стратегий имеет свои особенности, и выбор стратегии производится от конкретной предполагаемой модели использования приложения.

Стандартный сценарий для создания изоморфного приложения может включать в себя разного рода инструменты. Например, AWS Lambda – это бессерверный вычислительный сервис, который реагирует на определенные события приложения, запуская в ответ на них соответствующий программный код и отвечает за автоматическое выделение необходимых вычислительных ресурсов для стабильной работы приложения.

Стек AWS компонентов может быть следующим:

– AWS Amplify – высокоуровневая платформа для управления сервисами AWS, в основном для мобильной и веб-разработки;

– AWS Lambda – запуск кода в облаке без управления серверами;

– AWS Cloudfront (CDN) – сеть доставки контента;

– AWS Simple Storage Service (S3) – хранение статических ресурсов.

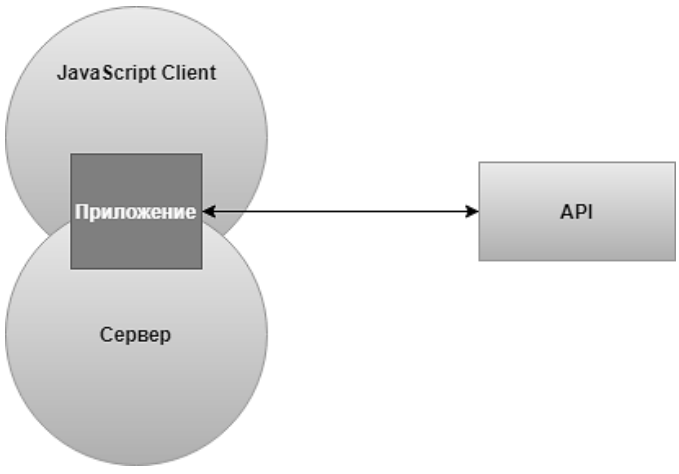

Рис. 6. Клиент-серверный подход

На Рис. 7 показана архитектура изоморфного приложения с помощью вышеперечисленных инструментов.

В листинге кода 1 показан стандартный базовый App React приложения, построенного на изоморфной архитектуре. По сути, оно ничем не отличается от базового стандартного приложения.

Ниже показан листинг кода 2 серверного JavaScript, в котором показана генерация и отправка HTML-файлов на клиентскую сторону, которая и позволит поисковым роботам увидеть контент ещё до загрузки всего приложения.

Вышеприведенный подход позволяет полностью решить проблему недостаточного или нулевого ранжирования SPA-приложений, построенных на JavaScript-фреймворках. Он позволяет отправлять статические файлы с контентом на клиентскую часть независимо от самого приложения (Рис. 8).

Основные подходы к созданию изоморфного приложения включают в себя несколько важных факторов:

– общая часть приложения не должна зависеть от среды исполнения;

– функциональность, зависящая от среды исполнения, должна быть вынесена в отдельный модуль;

– для клиентской и серверной части приложения должен быть общий роутер, потому что рендер по URL на клиенте и на сервере должен давать одинаковый результат;

– так как на каждый запрос сервер должен создавать новое состояние приложения, то в приложении не должно быть синглтонов и глобальных переменных;

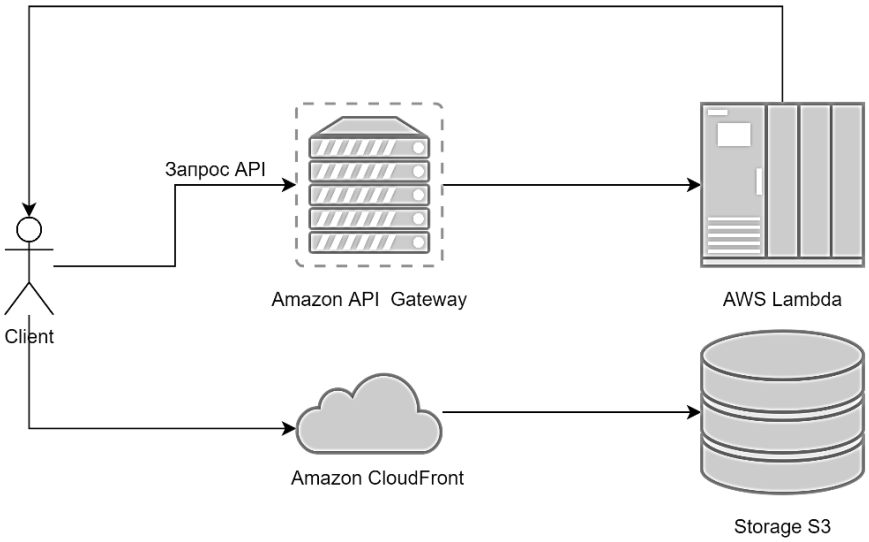

Рис. 7. Архитектура изоморфного приложения с помощью S3

Листинг кода 1. Базовый App.js файл

```
import React from 'react';
import './App.css';
function App() {
  return (
    <div className="My-App">
      <header className="My-app-header">
        Server Rendered React App
      </header>
    </div>
  );
}
export default App;
```

Листинг кода 2. Отправка статического контента

```
const express = require('express');
const awsSEM = require('aws-serverless-express/middleware');
const fs = require('fs');
const bodyParser = require('body-parser');
const render = require('./client/render').default;
const expApp = express();
expApp.use(bodyParser.json())
expApp.use(awsSEM.eventContext())
expApp.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*")
  res.header("Access-Control-Allow-Headers",  "Origin,  X-Requested-With,  Content-
Type, Accept")
  next()});
expApp.get('*', function(req, res) {
// Чтение файла index.html из the create-react-app build
  const html = fs.readFileSync("./client/index.html", "utf-8");
// Рендеринг приложения react на стороне сервера
  const markup = render();
// Отправка статического контента клиенту
  res.send(html.replace(`<divid="root"></div>`,`<div
id="root">${markup}</div>`))});
module.exports = expApp;
```

Рис. 8. Изоморфность

– на стороне сервера не нужно применять систему реактивности, так как в ней нет необходимости, потому что сервер просто создаёт статическую разметку.

На Рис. 9 показан сравнительный график использования изоморфного приложения со стандартным приложением.

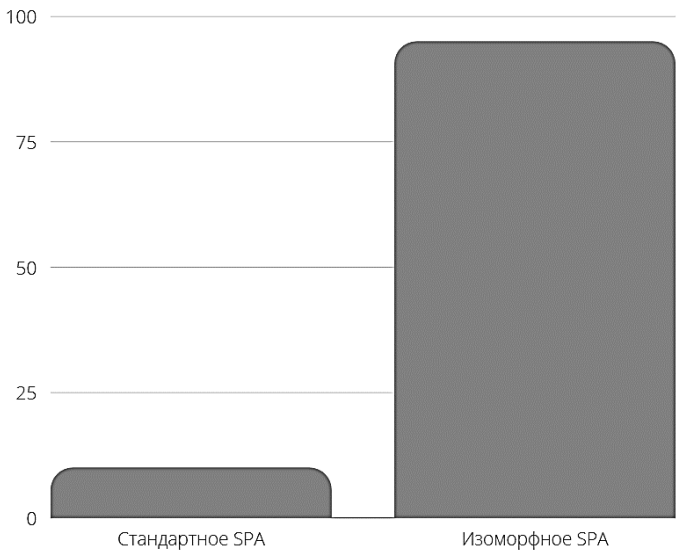

Рис. 9. Сравнение значения PageRank стандартного и изоморфного SPA-приложений

Таким образом, поисковые роботы смогут легко сканировать контент приложения, и поисковый рейтинг у сайта, использующего изоморфный подход, будет таким же, как у обычного статического сайта.

Заключение

В ходе исследования проблемы было выяснено, что существующие методы пререндеринга и работа со статическим HTML не позволяют полностью использовать возможности веб-приложения, либо же приводят к недостаточному сканированию веб-сайтов поисковыми роботами. Соответственно, SEO-ориентированность этих сайтов ухудшается.

Предложенное в статье решение данной проблемы, а именно использование изоморфного подхода позволяет решить вопрос SEO-ориентированности путём предварительной загрузки статических страниц с контентом сайта. Таким образом, можно быть уверенным, что поисковый робот, когда бы он ни просканировал страницу, сможет обнаружить контент и сможет его проиндексировать.

В статье показаны варианты использования и построения изоморфного приложения. Соблюдение этих правил позволяет получить правильное изоморфное приложение, которое практически закрывает проблему SEO-ориентированности SPA-приложений.

Литература

1. Дыкан А. Клиентское SEO. М.: Эдитус, 2016. 280 с.
2. Ковалев С. В. Базовое продвижение сайтов (SEO). Основные 20% информации по работе с сайтами для эффективного продвижения. Екатеринбург: Издательские решения, 2017. 25 с.
3. Петросян А. С. Seo Boom. Эффективная оптимизация сайтов. М.: ЛитРес, Самиздат, 2018. 110 с.
4. Page L. et al. The PageRank citation ranking: Bringing order to the web. Technical Report. Stanford InfoLab, 1998. 17 p.
5. Мелькин Н., Горяев К. Искусство продвижения сайта. Полный курс SEO: от идеи до первых клиентов. Вологда: Инфра-Инженерия, 2018. 280 с.
6. Гулин А., Маслов М., Сегалович И. Алгоритм текстового ранжирования Яндекса на РОМИП-2006. Труды четвертого российского семинара по оценке методов информационного поиска РОМИП'2006. Санкт-Петербург: НУ ЦСИ, 2006. С. 40–51.
7. Duckett J. Web Design with HTML, CSS, JavaScript and jQuery Set. Wiley, 2019. 1152 p.
8. Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. 2-е издание. СПб: Питер, 2021. 336 с.
9. Stefanov S. React: Up & Running: Building Web Applications. O'Reilly Media, 2016. 222 p.
10. Crockford D. How JavaScript Works. Virgule-Solidus LLC, 2018. 280 p.
11. Thomas M.T. React in Action. Manning Publications, 2018. 360 p.
12. Strimpel J., Najim M. Building Isomorphic Javascript Apps: From Concept to Implementation to Real-World Solutions. O'Reilly Media, 2016. 210 p.
13. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020. 310 p.

Суворов Александр Олегович. Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики» (НИУ ВШЭ), г. Москва, Россия. Доцент кафедры информационных технологий в бизнесе, кандидат технических наук. Количество печатных работ: 59 (в т.ч. 1 монография). Область научных интересов: информационные технологии, разработка программного обеспечения, интеллектуальный анализ данных, радиолокационное распознавание. E-mail: aosuvorov@hse.ru (Ответственный за переписку).

Петренко Александр Анатольевич. Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет» (ПНИПУ), г. Пермь, Россия. Доцент кафедры информационных технологий и автоматизированных систем, кандидат технических наук. Количество печатных работ: 34 (в т.ч. 1 монография). Область научных интересов: информационные технологии, автоматизация производственных процессов, искусственный интеллект. E-mail: hawk321@mail.ru

Аликин Артем Вадимович. Акционерное общество «ЛАНИТ», г. Москва, Россия. Разработчик. Область научных интересов: информационные технологии, разработка программного обеспечения, машинное обучение. E-mail: tommyfongenious@yandex.ru

Using Isomorphic Approach for Solving SEO Problems of JavaScript-Based Web Applications

A. O. Suvorov[I], A. A. Petrenko[II], A. V. Alikin[III]

[I] HSE University, Moscow, Russia
[II] Perm National Research Polytechnic University, Perm, Russia
[III] Joint Stock Company "LANIT", Moscow, Russia

Abstract. The article is devoted to solving the problems of search engine optimization of single-page applications built based on modern technologies of reactive JavaScript, including such JavaScript-frameworks as React, Angular, Vue, which allow developers to quickly create and scale interactive applications. However, such single page application is a complex area with a huge number of nuances in terms of search engine optimization because JavaScript is a programming language that allows to create dynamically updated content. The main issues of modern search engine optimization are considered: the principles of search engine operation, the basic principles, and algorithms for ranking web pages by relevance are described. Using the React framework as an example, the main features of the development of modern single-page applications based on the modern JavaScript technology stack are described and the main disadvantages of such applications from the point of view of search engine optimization are shown, as well as existing approaches to solving this problem are considered, their disadvantages are highlighted, and a more effective and perfect hybrid approach based on the principles of isomorphism is proposed. An isomorphic approach to the development of single-page applications based on the principle of its design and strategies for its use is considered in detail, the principles of building such an application and examples of a block of program code are given, as well as a visual comparison of the ranking of a standard single-page application and a similar application built using an isomorphic architecture.

Keywords: search engine optimization (SEO), PageRank, relevancy, isomorphism, rendering, Single Page Application (SPA), JavaScript, Document Object Model (DOM), API, React.

DOI 10.14357/20718632210410

References

1. Dykan A. Klientskoe SEO [Client SEO]. Moscow, Jeditus, 2016, 280 p.
2. Kovalev S. V. Bazovoe prodvizhenie sajtov (SEO). Osnovnye 20% informacii po rabote s sajtami dlja jeffektivnogo prodvizhenija [Basic website promotion (SEO). The main 20% of information on working with sites for effective promotion]. Ekaterinburg, Izdatel'skie reshenija, 2017, 25 p.
3. Petrosjan A. S. Seo Boom. Jeffektivnaja optimizacija sajtov [Seo Boom. Effective website optimization]. Moscow, LitRes, Samizdat, 2018, 110 p.

4. Page L. et al. The PageRank citation ranking: Bringing order to the web. Technical Report. Stanford InfoLab, 1998. 17 p.
5. Mel'kin N., Gorjaev K. Iskusstvo prodvizhenija sajta. Polnyj kurs SEO: ot idei do pervyh klientov [The art of website promotion. The Complete SEO Course: From Idea to First Customers]. Vologda, Infra-Inzhenerija, 2018, 280 p.
6. Gulin A., Maslov M., Segalovich I. Algoritm tekstovogo ranzhirovanija Jandeksa na ROMIP-2006 [Yandex text ranking algorithm on ROMIP-2006]. Trudy chetvertogo rossijskogo seminara po ocenke metodov informacionnogo poiska ROMIP'2006. Sankt-Peterburg, NU CSI, 2006, pp. 40–51.
7. Duckett J. Web Design with HTML, CSS, JavaScript and jQuery Set. Wiley, 2019. 1152 p.
8. Braun I. Veb-razrabotka s primeneniem Node i Express. Polnocennoe ispol'zovanie steka JavaScript. 2-e izdanie [Web development with Node and Express. Full use of the JavaScript stack. 2nd edition.] Sankt-Peterburg, Piter, 2021, 336 p.
9. Stefanov S. React: Up & Running: Building Web Applications. O'Reilly Media, 2016, 222 p.
10. Crockford D. How JavaScript Works. Virgule-Solidus LLC, 2018, 280 p.
11. Thomas M.T. React in Action. Manning Publications, 2018, 360 p.
12. Strimpel J., Najim M. Building Isomorphic Javascript Apps: From Concept to Implementation to Real-World Solutions. O'Reilly Media, 2016, 210 p.
13. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020, 310 p.

Suvorov A. O. Candidate of Engineering Sciences, Docent, HSE University, 20 Myasnitskaya street, Moscow, 101000, Russia, e-mail: AOSuvorov@hse.ru

Petrenko A. A. Candidate of Engineering Sciences, Perm National Research Polytechnic University, 29 Komsomolsky pr., Perm, 614990, Russia, e-mail: hawk321@mail.ru

Alikin A. V. Software Developer, Joint Stock Company "LANIT", building 1, 14 Murmansk proezd, Moscow, 129075, Russia, e-mail: tommyfongenious@yandex.ru