УДК: 004.432 DOI: 10.34824/VKNIIRAN.2023.14.3.010

СЕРВЕРНЫЙ РЕНДЕРИНГ В REACT С ИСПОЛЬЗОВАНИЕМ JAVASCRIPT ПРЕИМУЩЕСТВА И РЕАЛИЗАЦИЯ

© Гишлакаев Сайфулла Умарович (a), Гайрабекова Тамара Израиловна (b)

(a) Чеченский государственный университет им. А.А. Кадырова, Российская Федерация, г. Грозный; студент 4 курса Института математики, физики и информационных технологий, sgishlakaev@mail.ru
(b) Чеченский государственный университет им. А.А. Кадырова, Российская Федерация, г. Грозный; кандидат технических наук, доцент кафедры прикладной математики и компьютерных технологий, sti_ing@mail.ru

Аннотация. Веб-разработка продолжает развиваться, и одним из наиболее актуальных и востребованных направлений является серверный рендеринг. В этой статье мы рассмотрим, что такое серверный рендеринг и как его можно реализовать в React с использованием JavaScript. Мы также рассмотрим преимущества, которые предоставляет серверный рендеринг, и почему он становится популярным в веб-разработке.

Ключевые слова: Реакт, вэб-сайт, вэб-приложение, логика, useState, useEffect.

SERVER RENDERING IN REACT USING JAVASCRIPT ADVANTAGES AND IMPLEMENTATION

© Gishlakaev Sayfulla Umarovich (a), Gayrabekova Tamara Israilovna (b)

(a) Chechen State University named after. A.A. Kadyrov, Russian Federation, Grozny; 4th year student at the Institute of Mathematics, Physics and Information Technologies, sgishlakaev@mail.ru
(b) Chechen State University named after. A.A. Kadyrov, Russian Federation, Grozny; Candidate of Technical Sciences, Associate Professor of the Department of Applied Mathematics and Computer Technologies, sti_ing@mail.ru

Abstract. Web development continues to evolve, and one of the most relevant and popular areas is server rendering. In this article, we will look at what server rendering is and how it can be implemented in React using JavaScript. We will also look at the advantages that server-side rendering provides and why it is becoming popular in web development.

Key words: React, web site, web application, logic, useState, useEffect.

Что такое серверный рендеринг?

Серверный рендеринг (также известный как SSR - Server-Side Rendering) - это процесс генерации HTML-разметки на сервере и ее отправки клиенту для отображения. В традиционном подходе, применяемом React, рендеринг происходит на стороне клиента с использованием JavaScript. При серверном рендеринге React-компоненты выполняются на сервере, генерируется HTML-разметка и передается клиенту. Такой подход позволяет браузеру получить полностью сформированный контент, что улучшает время загрузки страницы и SEO-оптимизацию.

Преимущества серверного рендеринга в React

Улучшенная производительность: Одним из основных преимуществ серверного рендеринга является улучшение производительности при загрузке страницы. Поскольку сервер уже генерирует HTML-разметку, браузеру необходимо только отобразить контент. Это позволяет ускорить время отклика и улучшить воспроизведение страницы.

Улучшенная SEO-оптимизация: Поисковые системы, такие как Google, предпочитают полностью отрендеренные страницы, которые легко проанализировать и индексировать. При использовании серверного рендеринга, страницы имеют полную разметку, что улучшает их видимость в поисковой выдаче. Это особенно полезно для приложений, которые требуют SEO-оптимизации.

Улучшенная доступность и SEO-оптимизация: При использовании серверного рендеринга, страницы будут доступны для поисковых систем и пользователей с отключенным JavaScript. Такие пользователи получат полностью функциональные страницы сразу после загрузки, что повышает доступностьконтента для всех пользователей.

Лучшее взаимодействие с социальными сетями: При использовании серверного рендеринга, страницы приложений будут корректно отображаться при предварительном просмотре ссылок в социальных сетях. Это позволяет предоставить более привлекательные и информативные предварительные просмотры для пользователей, которые переходят на страницу из социальных сетей.

Реализация серверного рендеринга в React с использованием JavaScript

Для реализации серверного рендеринга в React с использованием JavaScript существует несколько подходов и инструментов. Рассмотрим основные из них:

- Использование фреймворка Next.js: Next.js - это популярный фреймворк, основанный на React, который предоставляет инструменты для реализации серверного рендеринга. Он облегчает настройку проекта с серверным рендерингом, предоставляет API для работы с серверными данными и упрощает обработку маршрутов.
- Использование библиотеки React-Router: React-Router - это библиотека, которая позволяет управлять маршрутизацией в React-приложении. Она также поддерживает серверный рендеринг и позволяет генерировать HTML-разметку на сервере для каждого маршрута.
- Использование библиотеки ReactDOMServer: ReactDOMServer - это модуль React, который позволяет рендерить React-компоненты на сервере. Он предоставляет методы, такие как renderToString и renderToStaticMarkup, которые позволяют генерировать HTML-разметку на сервере и передавать ее клиенту.

Пример реализации серверного рендеринга с использованием ReactDOMServer:

```
// Импортируем необходимые модули
import React from 'react';
import ReactDOMServer from 'react-dom/server';

// Определяем React-компонент
const App = () => (
<div>
<h1>Привет, мир!</h1>
 <p>Это пример серверного рендеринга в React.</p>
 </div>
);

// Генерируем HTML-разметку на сервере
const html = ReactDOMServer.renderToString(<App />);

// Отправляем сгенерированную разметку клиенту
res.send(html);
```

Использование Next.js

```
// pages/index.js

import React from 'react';

const Home = () => (
 <div>
<h1>Добро пожаловать на главную страницу!</h1>
<p>Это пример страницы с серверным рендерингом в React с использованием Next.js.</p>
</div>
);

export default Home;
```

В этом примере мы создаем простую страницу "Главная", которая будет отображаться на главном маршруте (путь "/"). Файл находится в директории "pages" и называется "index.js".

- В первой строке мы импортируем React для использования JSX синтаксиса.
- Затем мы определяем компонент Home, который будет представлять нашу главную страницу. Внутри компонента у нас есть div элемент с заголовком h1 и абзацем p, который содержит текстовое содержимое.
- Наконец, мы экспортируем компонент Home по умолчанию, чтобы он мог быть использован в других частях приложения.

Это простой пример страницы, которая будет отображаться с использованием серверного рендеринга в Next.js. Next.js автоматически обрабатывает серверный рендеринг для страниц, определенных в директории "pages".

Использование React-Router

```
// server.js

import React from 'react';
import { renderToString } from 'react-dom/server';
import { StaticRouter, Route } from 'react-router-dom';
import App from './App';

// Определяем серверный роутинг
app.get('*', (req, res) => {
const context = {};

// Генерируем HTML-разметку на сервере с использованием React-Router
const html = renderToString(
<StaticRouter location={req.url} context={context}>
 <Route path="/" component={App} />
</StaticRouter>
);

// Отправляем сгенерированную разметку клиенту
res.send(`
<!DOCTYPE html>
<html>
 <head>
 <title>React Server-Side Rendering</title>
</head>
<body>
<div id="app">${html}</div>
<script src="client.js"></script>
</body>
</html>
);
});
```

В этом примере мы создаем серверный файл "server.js", который отвечает за генерацию серверной разметки с использованием React-Router.

- В первых строках мы импортируем необходимые модули из React и React-Router.
- Затем мы определяем обработчик маршрута для всех запросов (app.get('*')), который будет выполняться при любом URL.

- В этом примере мы создаем серверный файл "server.js", который отвечает за генерацию серверной разметки с использованием React-Router.
- В первых строках мы импортируем необходимые модули из React и React-Router.
- Затем мы определяем обработчик маршрута для всех запросов (app.get('*')), который будет выполняться при любом URL.
- Мысоздаем пустой объект context, который будет использоваться React-Router для передачи информации о рендеринге на сервере.
- Далее, мы используем метод renderToString из react-dom/server, чтобы сгенерировать HTML-разметку на сервере. Мы оборачиваем наше приложение (App) в StaticRouter, который принимает текущий URL и context объект. Затем мы определяем маршрут с помощью Route компонента, указывая путь и компонент, который должен быть отображен для данного пути.
- Затем мы отправляем сгенерированную HTML-разметку клиенту с помощью метода res.send(). Мы также включаем скрипт client.js, который будет отвечать за клиентский рендеринг и взаимодействие с React компонентами.

В целом, этот пример демонстрирует как реализовать серверный рендеринг с использованием React-Router. Мы используем StaticRouter для обработки маршрутов на сервере и renderToString для генерации HTML-разметки, которая будет отправлена клиенту.

Заключение

Серверный рендеринг в React с использованием JavaScript является мощным инструментом, который предоставляет множество преимуществ, таких как улучшенная производительность, SEO-оптимизация, улучшенная доступность и взаимодействие с социальными сетями. Реализация серверного рендеринга в React возможна с помощью фреймворков и библиотек, таких как Next.js, React-Router и ReactDOMServer.

Выбор конкретного подхода зависит от требований и особенностей вашего проекта. Однако, независимо от выбранного инструмента, серверный рендеринг в React является полезным и эффективным способом оптимизации производительности и улучшения пользовательского опыта. Рассмотрите возможность использования серверного рендеринга в ваших проектах и экспериментируйте с различными подходами, чтобы достичь наилучших результатов.

ЛИТЕРАТУРА

1. Николас З. ECMAScript 6 для разработчиков. 2017 г.
2. Прасти Н. Введение в ECMAScript 6. 2016г.
3. Марейн Х. Выразительный JavaScript. Современное веб-программирование. 3-е издание. 2020 г.
4. Рыбаков Е. JavaScript и Node.js для Web-разработчиков. 2022 г.
5. Черный Б. Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений. 2021 г.

REFERENCES

1. Nicholas Z. ECMAScript 6 for developers. 2017
2. Prasti N. Introduction to ECMAScript 6. 2016.

3. Marein H. Expressive JavaScript. Modern web programming. 3rd edition. 2020
4. Rybakov E. JavaScript and Node.js for Web developers. 2022
5. Cherny B. Professional TypeScript. Development of scalable JavaScript applications. 2021