Universal Rendering

Julian Beck

Betreuer: Prof. Dr. rer. nat. Christian Zirpins

Zusammenfassung An dieser Stelle sollte später eine Kurzzusammenfassung stehen.

Inhaltsverzeichnis

1 Einleitung

Seit dem Beginn des Webs funktioniert das Surfen wie folgt: Ein Webbrowser fordert eine bestimmte Seite an, ein Server im Internet bearbeitet die Anfrage und generiert ein HTML (Hypertext Markup Language) Dokument als Antwort. Dies bezeichnet man als serverseitiges rendern. In den Anfängen des Webs stellte dies kein Problem dar, da die Browser nicht leistungsstark waren und die Webseiten aus meist statischen Inhalt bestanden. Später mit HTML5 wurden Webseiten dynamischer und interaktiver für den Nutzer, was dazu führte, dass immer mehr Apps, sogenannte Single Page Applikationen, vollständig im Browser auf einer Seite liefen. Um dies zu ermöglichen wird clientseitiges rendern verwendet. Single Page Applications oder kurz SPAs, bieten Vorteile für den Anwender: Sie reagieren schnell auf Benutzerinteraktionen und können zwischen Seiten navigieren, ohne sie komplett neu zu laden. Gleichzeitig sind SPAs komfortabel zu entwickeln, dank moderner Frameworks. Beide Varianten, serverseitiges und clientseitiges rendern, haben Vor- und Nachteile. Universal Rendering kombiniert die beiden Ansätze und erfüllt alle Anforderungen an eine moderne Webanwendung.

1.1 Anforderungen an eine Webanwendung

Eine moderne Webanwendungen sollten folgende Anforderungen erfüllen:

- Damit die Seite von Suchmaschinen gefunden werden kann, sollte sie von Suchmaschinen Crawler indexierbar sein. Dies wird Suchmaschinenoptimierung oder auch SEO (engl. search engine optimization) genannt.
- Eine moderne Webseite muss beim Aufrufen für den Anwender schnell laden. Der Anwender sollte nicht lange warten müssen bis er die Anwendung sieht und mit ihr interagieren kann
- Eine moderne Webanwendung muss dynamisch und interaktiv sein. Die Anwendung sollte auf Benutzereingaben reagieren, ohne lange Ladezeiten oder neuladen der Seite. Die Anwendung sollte einfach zu entwickeln sein und gleichzeitig muss sichergestellt werden, dass der Programmcode einfach zu pflegen ist. Code duplikation sollte minimiert werden.

1.2 Terminologie

Beim Rendern von Webtechnologien unterscheidet man zwischen dem Rendern auf dem Server und dem Client:

- SSR Server-Side Rendering: Rendern auf der Server Seite bezieht sich auf das generieren eines HTML Dokumentes aus einer Single Page Application.
- CSR Client-Side Rendering: Rendern auf der Client Seite bezieht sich auf das generieren und parsen eines HTML Dokumentes zu einem DOM und darstellen für den Anwender.

Der Begriff Universal Rendering beschreibt eine Kombination aus server- und clientseitiges rendern. Dieser Ansatz wird oft auch als Isomorphic oder Server-Side Rendering bezeichnet.

Folgende Begriffe beschreiben unterschiedliche Zeiten beim Laden einer Webseite:

- TTFB: Time to First Byte - Zeit bis zum ersten Byte - Die Zeit zwischen dem Klicken auf ein Link und Erhalten der Daten vom Server.
- FP: First Paint - der Zeitpunkt an dem der erste Inhalt für den User sichtbar wird.
- FCP: First Contentful Paint oder auch First Meaningful Paint - Der Zeitpunkt, an dem der Benutzer den wichtigsten Inhalt einer Website als fertig geladen sieht. Die Zeit bis zum FCP wird auch als kritischer Rendering-Pfad (engl.:Critical Rendering Path) bezeichnet.
- TTI: Time To Interactive - Die Zeit bis die Seite interaktiv wird und der Anwender mit ihr interagieren kann.

Da die Anwendung vollständig auf der Client Seite läuft, ist es schwierig für Suchmaschinen Crawler die seite zu Indexieren was zu einer schlechten SEO führt. Des weiteren, dadurch dass die Webseite nicht auf dem Server, sonder vom Client gerendert wird, muss der User warten bis die seite vollständig gerendert wird. Server Side rendering ist eine Mischung beider Ansätze. Es kombiniert die SEO und Performance von Server seitigen Applikationen und die Interaktivität und flexibilität von Client Side Anwendungen.

Diese Arbeit beginnt mit den der Funktionsweise von server - und clientseitigen Rendern. Es werden die Vor - und Nachteile der jeweiligen Vorgehensweisen untersucht und beschrieben warum eine Notwendigkeit für Universal Rendering besteht. Nach einer Einführung in den Universal Rendering Vorgang, werden die verwendeten Technologien und deren Funktionsweise für das Rendern beschrieben. Anschließend werden die positiven und negativen Aspekte des Universal Rendering beschrieben und Alternativen genannt. Danach beschreibt die Arbeit unterschiedliche Frameworks zum implementieren der Rendering Technologie und zeigt wie Unternehmen Universal Rendering verwenden. Das Fazit erarbeitet wann welche Rendering Technologie verwendet werden soll und zeigt den aktuellen Stand von Suchmaschinen Crawler.

2 Serverseitiges Rendering

2.1 Serverseitiges Rendering mit Ajax

Dies ist ein Zitat [BeKR09]. test[Jaso16] test[uJoh18] test[Joel16]

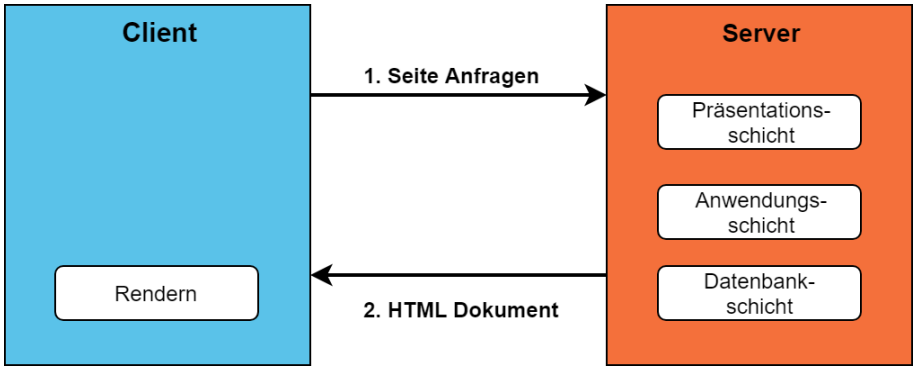

Abbildung 1. HTMl Dokument einer React Seite

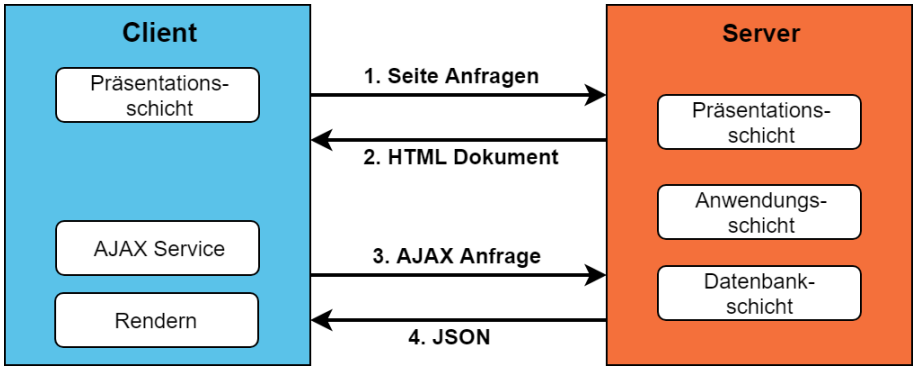

Abbildung 2. HTMl Dokument einer React Seite

3 Clientseitiges Rendering

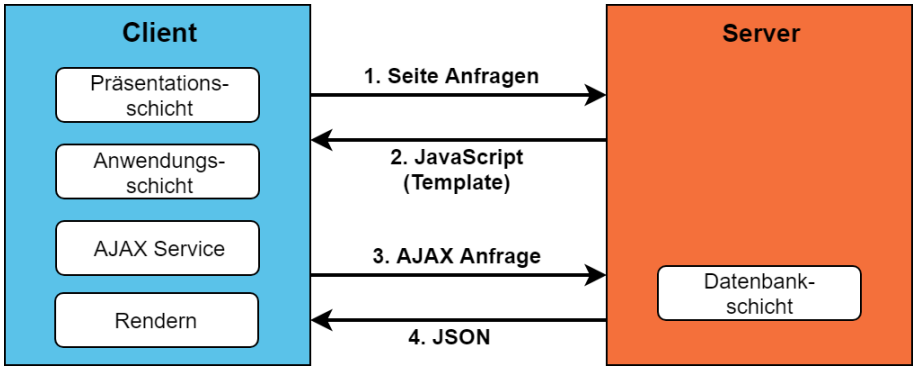

Abbildung 3. HTMl Dokument einer React Seite

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="shortcut icon" href="/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <link rel="manifest" href="/manifest.json" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

    <script src="/static/js/bundle.js"></script>
    <script src="/static/js/0.chunk.js"></script>
    <script src="/static/js/main.chunk.js"></script></body>
</html>
```

Abbildung 4. HTMl Dokument einer React Seite

4 Universal Rendering

4.1 Isomorphic JavaScript

4.2 Virtuelles DOM

4.3 Clientseitige Hydration

4.4 Rendering Ablauf

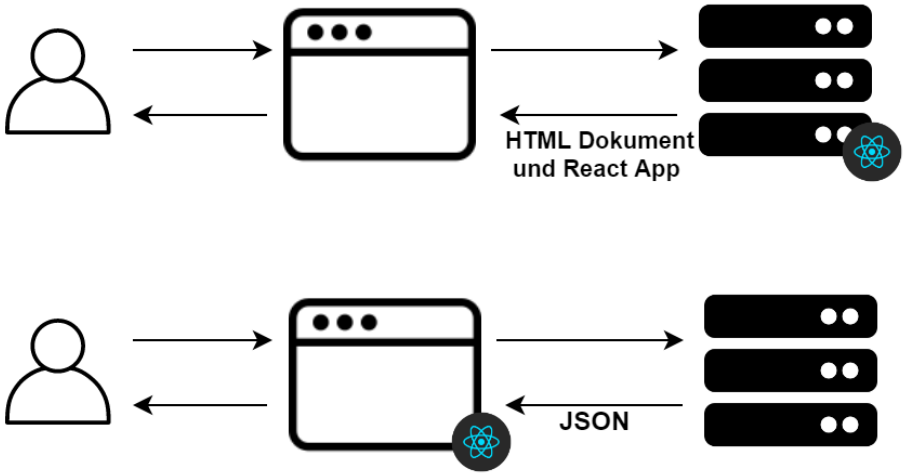

Abbildung 5. HTMl Dokument einer React Seite

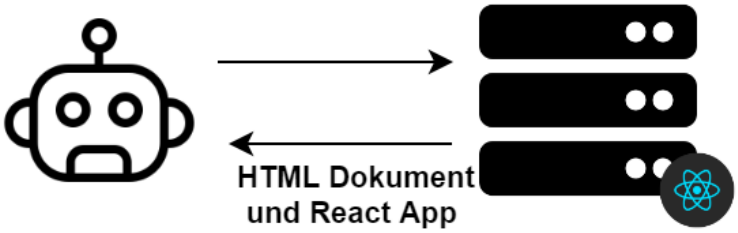

Abbildung 6. HTMl Dokument einer React Seite

4.5 Vorteile

4.6 Nachteile

4.7 Alternativen

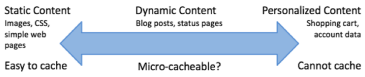

Abbildung 7. HTMl Dokument einer React Seite

```
<template>
  <LazyHydrate when-visible>
      <Article/>
  </LazyHydrate>
</template>
```

Abbildung 8. HTMl Dokument einer React Seite

5 Frameworks

5.1 React und Next.js

5.2 Vue.js und Nuxt.js

5.3 Angular Universal

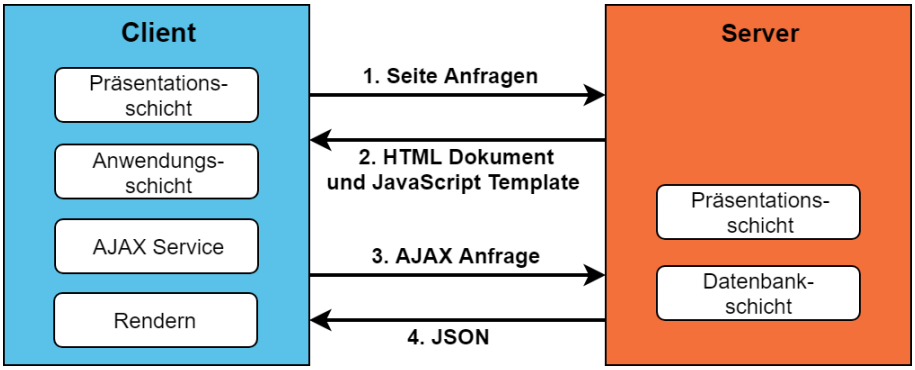

Abbildung 9. HTMl Dokument einer React Seite

```
<body data-n-head="">
    <div data-server-rendered="true" id="__nuxt">
        <div id="__layout">
            <div>
                <div>
                    <h1 class="title">
                        vue-server-side-rendering
                    </h1>
                    <h2 class="subtitle">
                        Server side rendering with vue using nuxt
                    </h2>
                    <div class="links">
                        <a href="https://github.com/nuxt/nuxt.js"
                            target="_blank" class="button--grey">GitHub</a>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
```

Abbildung 10. HTMl Dokument einer React Seite

6 Universal Rendering in der Praxis

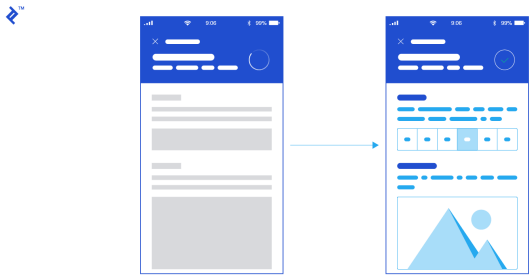

Abbildung 11. HTMl Dokument einer React Seite

```
import cowsay from 'cowsay-browser';

function CowsayHi() {
  return <pre>{cowsay.say({ text: 'hi there!' })}</pre>;
}

export default CowsayHi;
```

Abbildung 12. HTMl Dokument einer React Seite

7 Fazit und Ausblick

```jsx
import Link from 'next/link';

function Header() {
  return (
    <nav>
      <ul>
        <li>
          <Link prefetch href="/">
            <a>Home</a>
          </Link>
        </li>
        <li>
          <Link prefetch href="/about">
            <a>About</a>
          </Link>
        </li>
        <li>
          <Link prefetch href="/contact">
            <a>Contact</a>
          </Link>
        </li>
      </ul>
    </nav>
  );
}

export default Header;
```

Abbildung 13. HTMl Dokument einer React Seite

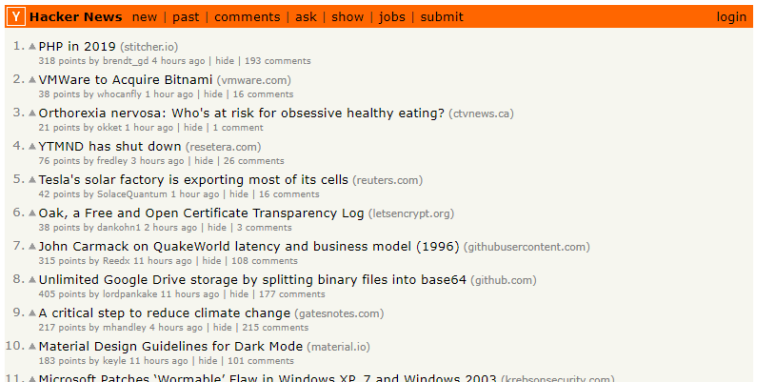

Abbildung 14. HTMl Dokument einer React Seite

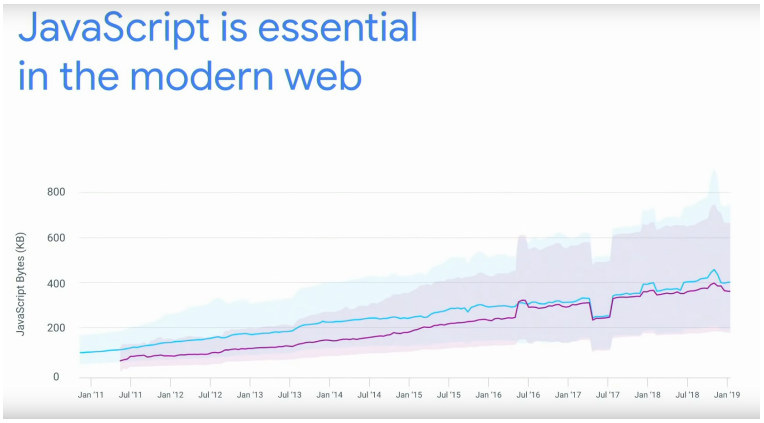

Abbildung 15. HTMl Dokument einer React Seite

Literatur

BeKR09. Steffen Becker, Heiko Koziolek und Ralf Reussner. The Palladio Component Model for Model-driven Performance Prediction. *Journal of Systems and Software*, Band 82, 2009, S. 3–22.

Jaso16. Maxime Najim Jason Strimpel. *Building Isomorphic JavaScript Apps.* O'Reilly. 2. Auflage, 2016.

Joel16. Chen Joel. ReactJS SSR Profiling and Caching, Sep 2016.

uJoh18. Tom Greenaway und John Müller. Deliver search-friendly JavaScript-powered websites. 2018.