Adnan Avni

Palvelinpuoleisen ja selainpuoleisen renderöinnin vertailu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

27.4.2025

Tiivistelmä

Tekijä:	Adnan Avni
Otsikko:	Palvelinpuoleisen ja selainpuoleisen renderöinnin vertailu
Sivumäärä:	33 sivua
Aika:	27.4.2025

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Ilpo Kuivanen

Insinöörityössä verrattiin palvelinpuoleisen (SSR) ja selainpuoleisen (CSR) renderöinnin menetelmiä web-sovelluksissa. Työn tavoitteena oli selvittää, millaisissa käyttötapauksissa kumpikin menetelmä on suositeltava ja mitkä ovat niiden keskeiset erot suorituskyvyn, turvallisuuden, resurssien käytön ja hakukoneoptimoinnin näkökulmasta.

Teoriaosuuden lisäksi toteutettiin kaksi sisällöltään ja rakenteeltaan samanlaista sovellusta, joista toinen rakennettiin Reactilla (CSR) ja toinen Next.js:llä (SSR). Sovelluksia vertailtiin Google Lighthousen mittaustulosten avulla suorituskyvyn ja SEO:n osalta.

Tuloksissa havaittiin, että SSR-menetelmällä saavutettiin parempi ensilatausnopeus ja hakukonenäkyvyys, kun taas CSR-menetelmä tarjosi paremman joustavuuden ja pienemmän palvelimen kuormituksen. Suorituskykyerot korostuivat erityisesti hitailla internetyhteyksillä ja monimutkaisilla sivustoilla.

Työn perusteella palvelinpuoleinen renderöinti soveltuu paremmin raskaampiin ja hakukoneoptimoituihin sivustoihin, kun taas selainpuoleinen renderöinti on edullisempi ja skaalautuvampi ratkaisu kevyempiin ja interaktiivisiin sovelluksiin.

Avainsanat: palvelinpuolen renderöinti, selainpuolen renderöinti, SSR, CSR, suorituskyky, SEO, React, Next.js

---

Tämän opinnäytetyön alkuperä on tarkastettu Turnitin Originality Check -ohjelmalla. Työssä on käytetty OpenAI:n ChatGPT:n GPT-4.5 mallia ideointiin sekä kieliasun viimeistelyyn. Insinöörityön tekijä on vastuussa kaikesta opinnäytetyön sisällöstä ja muotoilusta.

Abstract

Author:	Adnan Avni
Title:	Comparison of Server-side and Client-side Rendering
Number of Pages:	33 pages
Date:	27 April 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Ilpo Kuivanen, Senior Lecturer

The aim of this thesis was to compare server-side rendering (SSR) and client-side rendering (CSR) methods in web applications. The objective was to determine in which cases each method is preferable and to examine the key differences in terms of performance, security, resource usage, and search engine optimization (SEO).

In addition to the theoretical analysis, two similar applications were developed: one with React (CSR) and one with Next.js (SSR). The applications were compared using Google Lighthouse metrics focusing on performance and SEO aspects.

The results showed that SSR provided better initial loading speed and improved search engine visibility, while CSR offered better flexibility and lower server load. Performance differences were more evident with slower internet connections and complex web pages.

Based on the findings, SSR is more suitable for complex, SEO-critical websites, whereas CSR is a cost-effective and scalable choice for lighter, more interactive applications.

Keywords: server-side rendering, client-side rendering, SSR, CSR, performance, SEO, React, Next.js

Sisällys

Lyhenteet

API: *Application Programming Interface*. Sovellusten ohjelmointirajapinta.

CSS: *Cascading Style Sheets*. Kuvauskieli, jolla määritellään internetsivujen ulkoasu ja tyylit, kuten fontit, värit ja asettelu.

CSSOM: *CSS Object Model*. CSS-tyylien selainkohtainen esitysmuoto.

CSR: *Client-side Rendering*. Selainpuolen renderöinti, jossa sivu rakennetaan selaimessa JavaScriptillä.

DNS: *Domain Name System.* Nimipalvelujärjestelmä, joka muuntaa verkkotunnuksia IP-osoitteiksi

DOM: *Document Object Model*. Internetsivun rakenteen selainkohtainen esitysmuoto.

HMR: *Hot Module Replacement.* Kehitystyökalu, joka mahdollistaa JavaScript- ja CSS-muutosten päivittämisen selaimessa ilman sivun uudelleenlatausta.

HTML: *HyperText Markup Language*. Internetsivujen rakenteen kuvauskieli, jolla määritetään sivun sisältö ja rakenne.

HTTP: *HyperText Transfer Protocol*. Tiedonsiirtoprotokolla selaimen ja palvelimen välillä

ISR: *Incremental Static Regeneration*. Inkrementaalinen staattinen generointi, Next.js-kehyksen ominaisuus.

JSON: *JavaScript Object Notation*. Tekstipohjainen tiedonvaihtoformaatti tiedon välittämiseen rajapintojen kautta.

JSX: *JavaScript XML*. Reactin käyttämä syntaksi HTML-tyyppisten rakenteiden kirjoittamiseksi JavaScript-koodiin.

REST: *Representational State Transfer*. Rajapintojen suunnitteluperiaate tiedon välittämiseen selaimen ja palvelimen välillä.

SEO: *Search Engine Optimization*. Hakukoneoptimointi.

SPA: *Single Page Application.* Verkkosovellus, jossa kaikki sivun sisältö ladataan ja hallitaan yhdellä HTML-dokumentilla.

SSR: *Server-side Rendering*. Palvelinpuolen renderöinti, jossa sivu rakennetaan palvelimella ennen selaimelle lähettämistä.

SSG: *Static Site Generation*. Staattinen sivustogenerointi, jossa sivut generoidaan valmiiksi rakennusvaiheessa.

TSX: *TypeScript XML.* JSX-syntaksin laajennus, jota käytetään TypeScriptillä kirjoitetuissa React-komponenteissa.

1 Johdanto

Insinöörityössä vertaillaan palvelinpuolen ja selainpuolen renderöintimenetelmiä web-sovelluksissa. Työssä selvitetään, miten nämä kaksi renderöintimenetelmää eroavat toisistaan niin teoreettisesti kuin käytännön toteutuksissakin ja miten ne vaikuttavat sovellusten suorituskykyyn, turvallisuuteen ja hakukoneoptimointiin.

Web-sovelluksen kehittämisessä renderöintimenetelmän valinnalla on merkittävä vaikutus loppukäyttäjän kokemukseen sekä järjestelmän toimivuuteen. Näiden menetelmien erot ja niiden käytännön vaikutukset ovatkin ajankohtaisia kysymyksiä nykyaikaisessa web-kehityksessä.

Insinöörityön tavoitteena on tuottaa kattava analyysi renderöintimenetelmien eroista. Teoreettisen taustan lisäksi rakennetaan kaksi samanlaista nettisivua, joista toinen hyödyntää palvelinpuolen ja toinen selainpuolen renderöintiä. Sovelluksista mitataan suorituskyvyssä esiintyviä eroavaisuuksia. Tarkoituksena on selvittää, millaisissa käyttötapauksissa toinen menetelmä on osuvampi kuin toinen.

Aihe valittiin, koska web-sovelluskehittäminen on yleistynyt, mutta teknologian valinta voi olla haastavaa, ja se voi vaikuttaa merkittävästi sovelluksen toimivuuteen ja asiakaskokemukseen. Tutkimuksen tuloksilla pyritään tarjoamaan selkeät suositukset siitä, milloin kumpaakin renderöintimenetelmää kannattaa hyödyntää.

2 Teoria

Internetin saatavuuden yleistyessä maailmalla sovellusten kehittäjien on tärkeä perehtyä siihen, miten web-sovellusten kehitys kannattaisi toteuttaa. Erityisesti on syytä pohtia, renderöidäänkö sovellus selaimen vai palvelimen puolella. Oikean toteutustavan valitseminen ja hallitseminen voi säästää yrityksiltä merkittävästi resursseja, ja samalla internetsivut voidaan suunnata oikealle kohderyhmälle siten, että ne ovat helposti saavutettavissa.

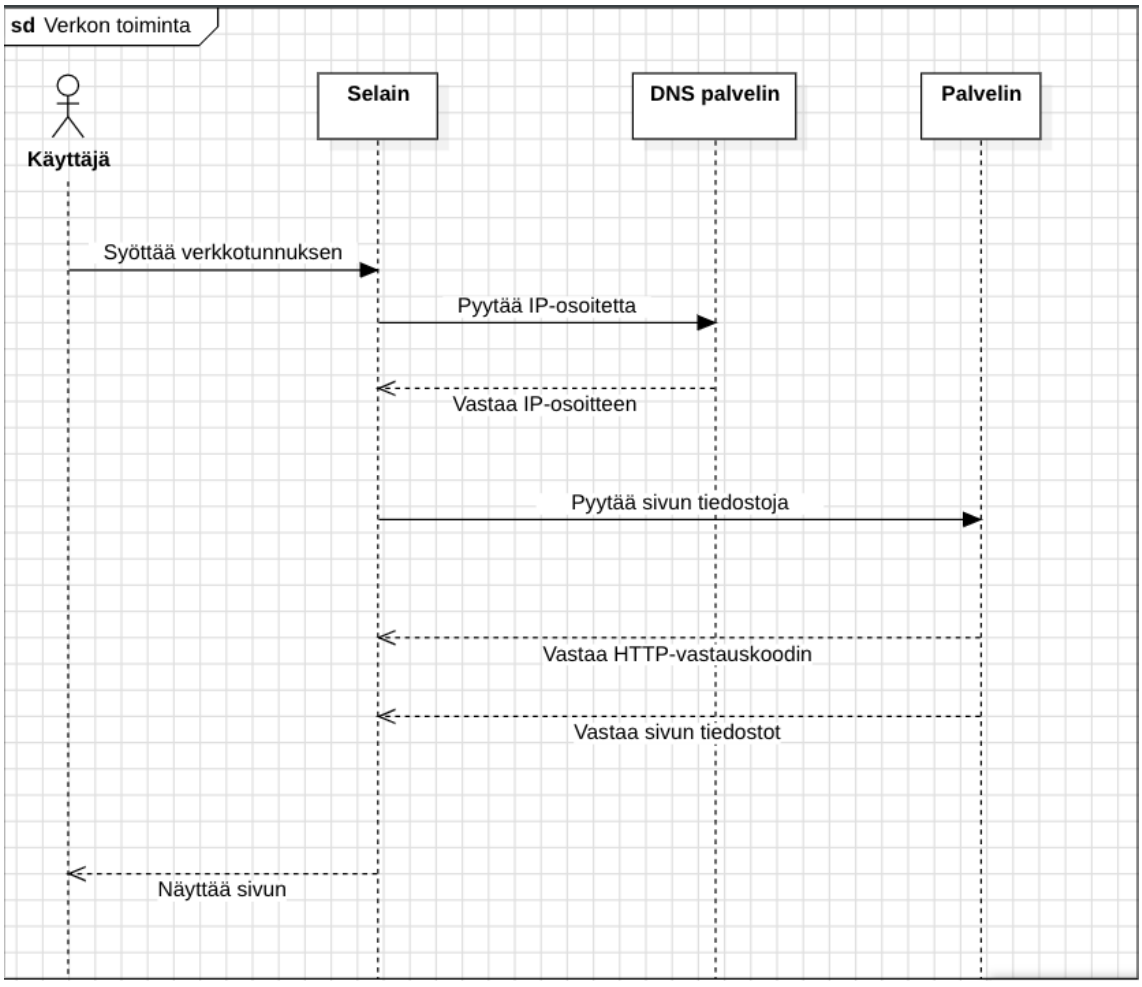
Palvelinpuolen renderöinti on perinteinen tapa toteuttaa web-sovelluksia. Siinä selain pyytää palvelimelta sivua ja saa vastauksena HTML-koodin. Nykyään on kuitenkin yhä yleisempää hyödyntää selainpuolen renderöintiä, jolloin palvelin palauttaa lähes tyhjän HTML-koodin sekä linkitetyn JavaScript-koodin, jonka selain suorittaa. Näin selaimelle muodostuvat tarvittavat elementit selainpuolella.

2.1 Verkon perustoiminta

Maailmanlaajuinen verkko kehittyy jatkuvasti, mutta sen perustoiminta nojaa yhä alkuperäisiin ydinteknologioihin. Internetsivut muodostuvat HTML-dokumenteista, joiden määrittelyjen avulla sivun rakenne ja sisältö voidaan esittää selaimessa. Tiedonvälitys toteutetaan HTTP-protokollan mukaisesti, mutta paketeissa kulkeva tieto salataan nykyaikaisessa käytössä lähes aina. Sivujen ulkoasu määritetään CSS:n avulla, ja JavaScriptillä tuodaan toiminnallisuutta. [1.]

Ensin käyttäjän antama verkkotunnus välitetään DNS-palvelimelle, joka palauttaa internetsivun IP-osoitteen eli palvelimen sijainnin verkossa. Tämän jälkeen selain lähettää IP-osoitteeseen pyynnön kyseisestä sivusta. Palvelin vastaa tilanteen mukaan erilaisilla HTTP-vastauskoodeilla, joista yleisimpiä ovat esimerkiksi ”200 OK” (onnistunut vastaus), ”404 Not Found” (pyydettyä resurssia ei löytynyt), ”500 Internal Server Error” (palvelinpuolen virhe). Mikäli vastauskoodi on ”200 OK”, selain vastaanottaa sivun tiedostot pienissä paketeissa. Mikäli palvelin lähettää kaikki tarvittavat JavaScript-, HTML- ja CSS-tiedostot, selain vastaanottaa sivun jo rakenteeltaan valmiina ja näyttää sen lähes sellaisenaan. Jos

puolestaan lähetetään lähes tyhjä HTML-tiedosto, joka sisältää JavaScript-koodin sivun rakentamiseen, selain luo sivun rakenteen itsenäisesti kyseisen koodin perusteella. Kuvassa 1 on havainnollistettu verkon toimintaa sekvenssidiagrammilla. Siitä kuitenkin puuttuu kokonaan sivun rakentaminen, joka tapahtuu joko selaimessa tai palvelimella, mikä riippuu käytetystä teknologiasta ja sivun käyttötarkoituksesta. Kuvassa 1 on myös huomioitava, että käyttäjälle näytetään sivu, jonka jälkeen sivun JavaScript-koodi haetaan vasta palvelimelta, jotta siihen saadaan toiminnallisuus. [2; 3.]



Kuva 1. Sekvenssikaavio verkon toiminnasta.

Kaikki tiedostot ja viestit kulkevat tiedonvälitysprotokollan mukaisesti, ja ne jaetaan pieniin paketteihin korruption välttämiseksi. Mikäli osa tiedostoista vioittuu, pienempiä tiedostopaketteja on helpompi korjata tai lähettää uudelleen. [2.]

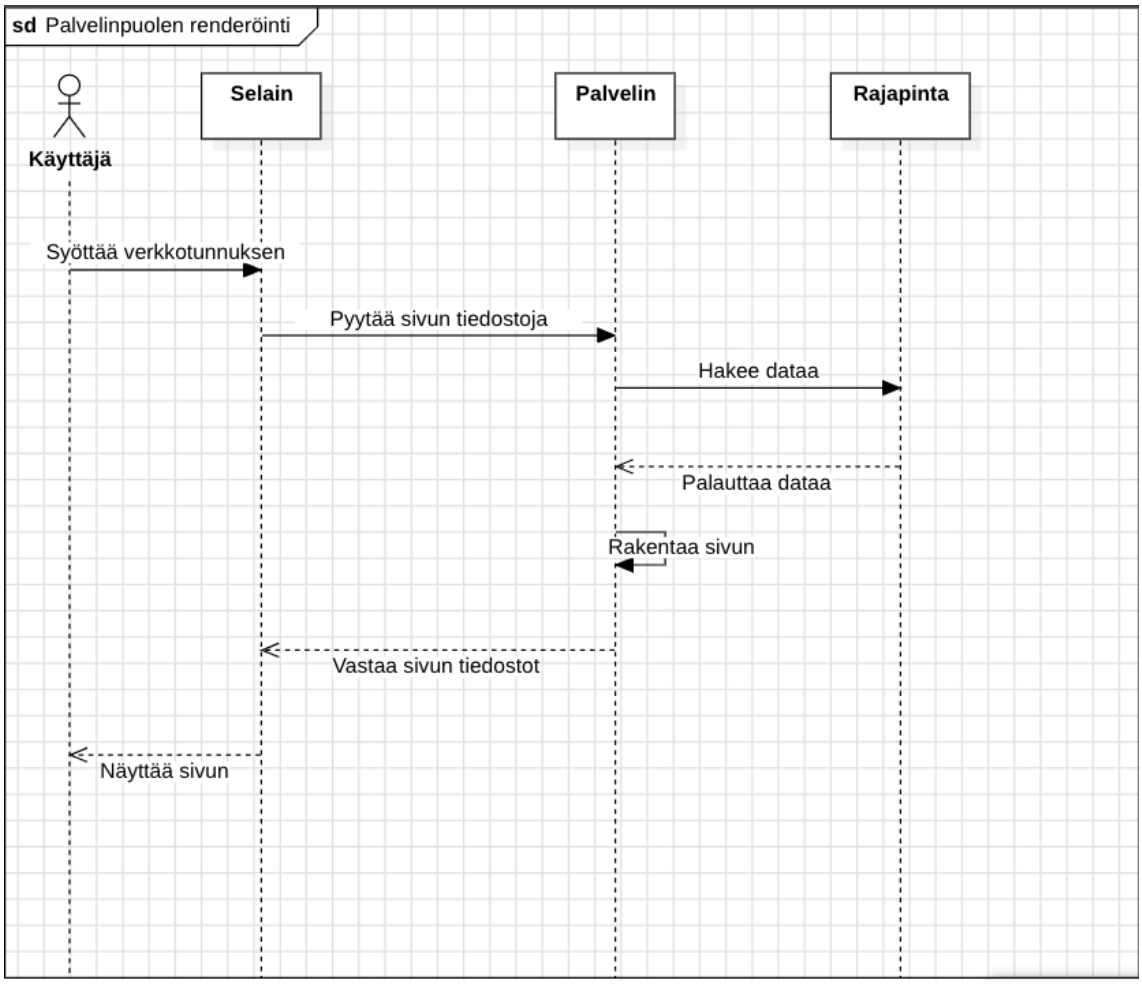
2.2 Palvelinpuolen renderöinti

Palvelimena voi toimia mikä tahansa laite, jossa on verkkoyhteys ja soveltuva ohjelmisto HTTP-pyyntöjen käsittelyyn. Kun pyyntö vastaanotetaan, palvelinohjelma toimittaa sitä vastaavan tiedoston tai muun resurssin. Tiedostot voivat olla monenlaisia, esimerkiksi kuvia, PDF-tiedostoja tai HTML-dokumentteja. [4.]

Mikäli tarvittavat tiedostot ja resurssit on talletettu suoraan palvelimelle, eikä niitä tarvitse hakea muualta, kyseessä on staattinen web-palvelin. Staattiset sivut ovat yleensä nopeita ja turvallisia, mutta niiden päivittäminen edellyttää tiedostojen muokkaamista suoraan palvelimella. Tällainen sivusto voi toimia esimerkiksi yrityksen esittelysivuna, jonka sisältö pysyy pitkään muuttumattomana.

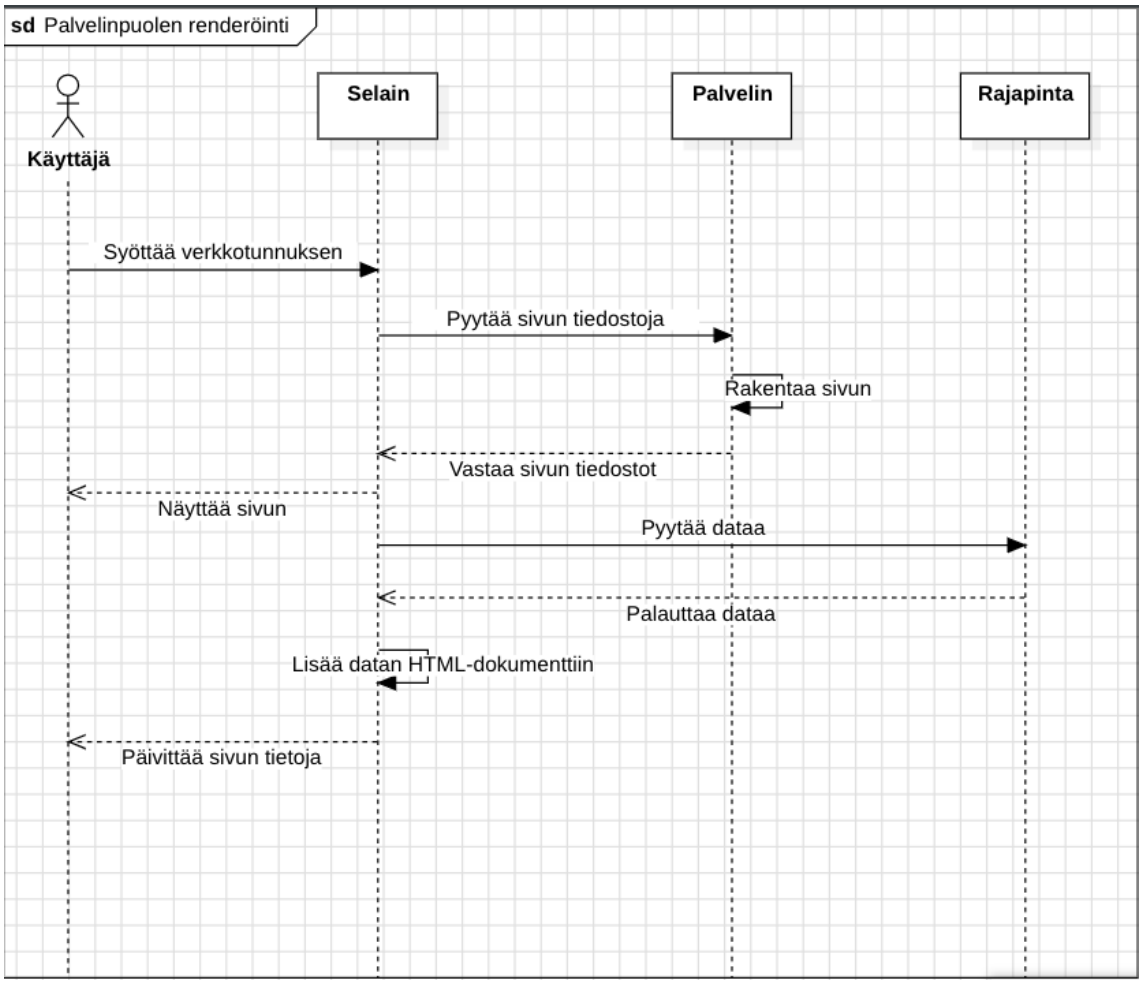
Jos sivuston sisältöä halutaan päivittää joustavammin, käytetään dynaamista toteutustapaa. Tällöin sivu hyödyntää erilaisia tietokantoja ja rajapintoja sisällön hakemiseen, minkä ansiosta sivua voidaan muokata ilman, että tiedostoja tarvitsee muokata jatkuvasti käsin [4].

Tiedonhakua on mahdollista toteuttaa joko synkronisesti tai asynkronisesti. Synkronisessa mallissa palvelin ottaa yhteyden rajapintaan ja kokoaa valmiin HTML-sivun ennen sen lähettämistä. Esimerkiksi synkronista mallia käytetään yleisesti PHP:tä hyödyntävissä WordPress-verkkosivustoissa sekä Java Spring -sovelluksissa, jotka luovat HTML-sivut palvelimella ennen niiden toimittamista selaimelle. Asynkroninen malli taas perustuu siihen, että selain muodostaa yhteyden rajapintaan suoraan. Näin sivun sisältöä voidaan päivittää ilman koko sivun uudelleenlatausta. Tämä säästää palvelimen resursseja. koska palvelin ei joudu rakentamaan HTML-sivua kokonaan uudestaan jokaisen pienen sisällön muutoksen yhteydessä. Esimerkiksi modernit web-sovellukset käyttävät yleisesti REST-rajapintoja dynaamisen tiedon noutamiseen. Kuvassa 2 on havainnollistettu synkronista mallia dynaamisissa sivuissa sekvenssikaaviolla.



Kuva 2. Sekvenssikaavio synkronisesta dynaamisen sivun vaiheista.

Synkronista tai asynkronista toteutustapaa on tärkeää pohtia sopivaa teknologiaa valittaessa. Kumpikin malli sisältää omat etunsa ja haittansa. Synkronisessa mallissa kaikki datan haku tapahtuu palvelimella, minkä ansiosta palvelin voi kontrolloida tarkasti rajapintaan kohistuvia pyyntöjä. Tämä voi kuitenkin pidentää sivun latautumisaikaa ja heikentää käyttökokemusta. Kuvassa 3 on havainnollistettu asynkronista mallia dynaamisissa sivuissa sekvenssikaaviolla.



Kuva 3. Sekvenssikaavio asynkronisesta dynaamisen sivun vaiheista.

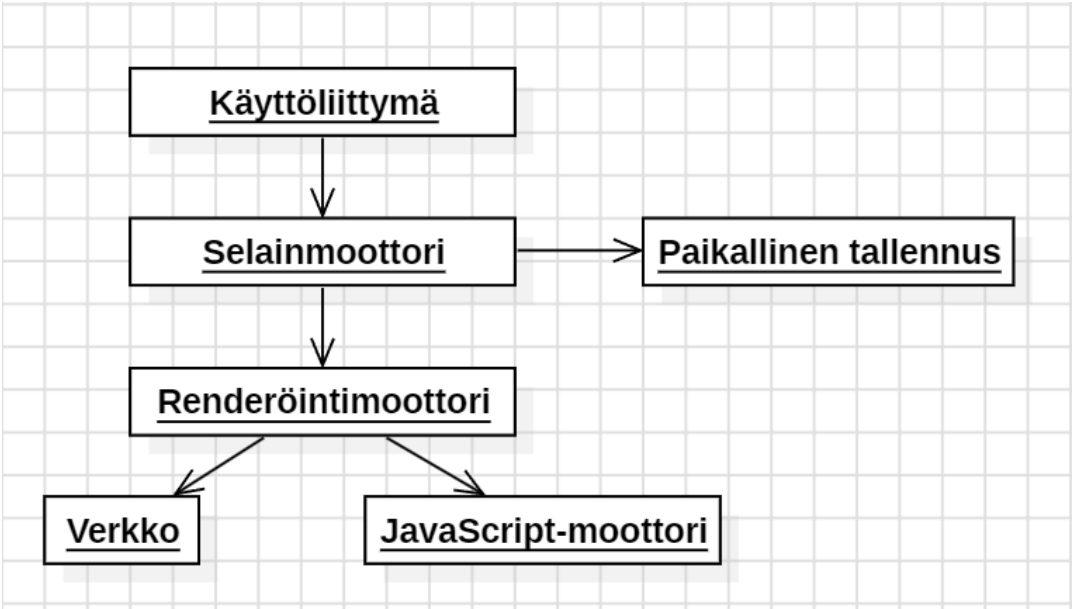
Asynkronisessa mallissa selain hakee ja päivittää tarvittavat tiedot lennossa, mikä parantaa käyttökokemusta ja keventää palvelimen kuormitusta. Haittapuolensa hakukoneoptimointi saattaa monimutkaistua, koska HTML-dokumenttiin lisätään dataa vasta selaimella. Lisäksi on mahdollista, että rajapintakutsut epäonnistuvat, mikä voi johtaa virhetilaan ja keskeneräisen sivun latautumiseen.

Palvelinpuolisessa renderöinnissä (SSR, server-side rendering) internetsivun varsinainen rakentaminen toteutetaan palvelimella. Tarvittavat tiedostot ja materiaalit säilytetään palvelimella ja niitä päivitetään rajapinnoista tai tietokannasta tarpeen mukaan. Selain vastaanottaa pyynnön perusteella valmiiksi luodun HTML-dokumentin, joka sisältää sivun rakenteen ja sisällön. Tämän ansoista selaimen ei tarvitse tuottaa sivua JavaScriptin avulla, vaan käyttäjälle esitetään

suoraan palvelimella laadittu sivu ja JavaScriptiä käytetään vain erilaisiin toimintoihin. [1.]

2.3 Selainpuolen renderöinti

Selainten arkkitehtuuri noudattaa pääpiirteissään samaa peruskaaviota (kuva 3). Käyttöliittymä sisältää hakupalkin, sivunpäivityspainikkeen sekä muita toiminnallisia painikkeita. Näiden toimintojen ohjaus tapahtuu selainmoottorin avulla, joka puolestaan hallitsee renderöintimoottoria. Renderöintimoottori on vastuussa web-sovelluksen näyttämisestä käyttäjälle, ja siihen sisältyy muun muassa JavaScript-moottori ja verkkokerros. JavaScript-moottori suorittaa sivujen JavaScript-koodin, ja verkkokerros vastaa yhteyksistä palvelimiin HTTP- ja HTTPS- pyyntöjen avulla. Lisäksi selaimessa on paikallinen tallennus, jota hyödynnetään esimerkiksi evästeiden säilyttämiseen. [5; 6.]



Kuva 4. Kaavio selaimen arkkitehtuurista

Selainpuoleisessa renderöinnissä (CSR, client-side rendering) internetsivun varsinainen rakentaminen toteutetaan selaimenpuolella. Palvelimelta lähetetään selaimelle lähes tyhjä HTML-tiedosto (esimerkkikoodi 1), joka sisältää viittauksia erilaisiin skripteihin. Aluksi näkyvissä voi olla tyhjä sivu, sillä HTML-tiedostoa

ei ole vielä ladattu kokonaan, eikä tyylitietoja ei ole ehditty soveltaa. Tiedostoon sisältyvät skriptit saavat selaimen keskeyttämään HTML:n jäsentämisen koodin suorittamiseksi, ellei skripteille ole määritetty asynkronista tai viivästettyä latausta.

```
<!DOCTYPE html>
<html lang="fi">
     <head>
          <meta charset="UTF-8" />
          <title>Selainpuolen renderöinti</title>
          <link rel="stylesheet" href="styles.css" />
     </head>
     <body>
          <div id="root"></div>
          <script src="main.js"></script>
     </body>
</html>
```

Esimerkkikoodi 1. React-sovelluksen lähettämä HTML-tiedosto.

JavaScriptin avulla päivitetään sivun DOM-rakennetta ja hallitaan sisällön näkyvyyttä. Sivun renderöinti voi viivästyä, mikäli selain odottaa JavaScript-koodin lataamista, jäsentämistä ja suorittamista ennen kuin kaikki sisältö on näkyvissä [6]. Nykyaikaisissa selaimissa tätä ongelmaa on kuitenkin optimoitu siten, ettei käyttäjä yleensä havaitse merkittävää viivettä. Selain ei välttämättä enää odota koko JavaScript-tiedoston suorittamista, vaan näyttää sivun osia sitä mukaan, kun ne ovat valmiita [7]. Tämä menettely nopeuttaa huomattavasti selainpuolisen renderöinnin piirtonopeutta etenkin monimutkaisissa web-sovelluksissa.

Kun CSS-tiedostot on ladattu, selain muodostaa niistä CSSOM-rakenteen. HTML-tiedostosta puolestaan luodaan DOM-puu. DOM ja CSSOM yhdistetään renderöintipuuksi, jota käytetään elementtien koon, sijainnin ja asettelun laskemiseen. Näiden rakenteiden tehokas käsittely on tärkeä sivun suorituskyvylle, koska monimutkainen tai huonosti optimoitu DOM- ja CSSOM-rakenne voi hidastaa sivun latautumista, interaktiivisuutta ja responsiivisuutta. Lopuksi selain piirtää näkymän käyttäjälle. Renderöintimoottori vastaa kaikista näistä vaiheista, mukaan lukien JavaScript-koodin suorittamisesta, tyylien soveltamisesta ja lopullisesta sivun näyttämisestä. [8.]

JavaScriptin avulla voidaan esimerkiksi ladata dynaamisesti sisältöä, kuten tuoteluottelo tai uutisvirta palvelimelta. Selain tekee taustalla REST-rajapintaan pyynnön, saa vastauksena datan JSON-muodossa ja päivittää sivun sisällön ilman koko sivun uudelleenlatausta. Tämä toimintatapa parantaa käyttökokemusta, koska sivusto reagoi käyttäjän toimintaan nopeasti ja sisältö voidaan päivittää saumattomasti.

2.4 Keskeiset erot

Palvelinpuolen ja selainpuolen renderöinnin keskeiset erot liittyvät suorituskykyyn, resurssien käyttöön ja hakukoneoptimointiin. Palvelinpuolisessa renderöinnissä sivu luodaan valmiiksi palvelimella, minkä ansiosta selain saa heti rakenteeltaan ja sisällöltään valmiin HTML-sivun, mikä parantaa ensilatauksen nopeutta ja tukee tehokkaammin hakukoneoptimointia. Selainpuolisessa renderöinnissä taas suuri osa sisällöstä muodostetaan selaimessa JavaScriptin avulla, jolloin sivusta on dynaamisempi ja palvelimen resurssien käyttö pienenee, mutta käyttäjän laite joutuu tekemään enemmän laskentaa.

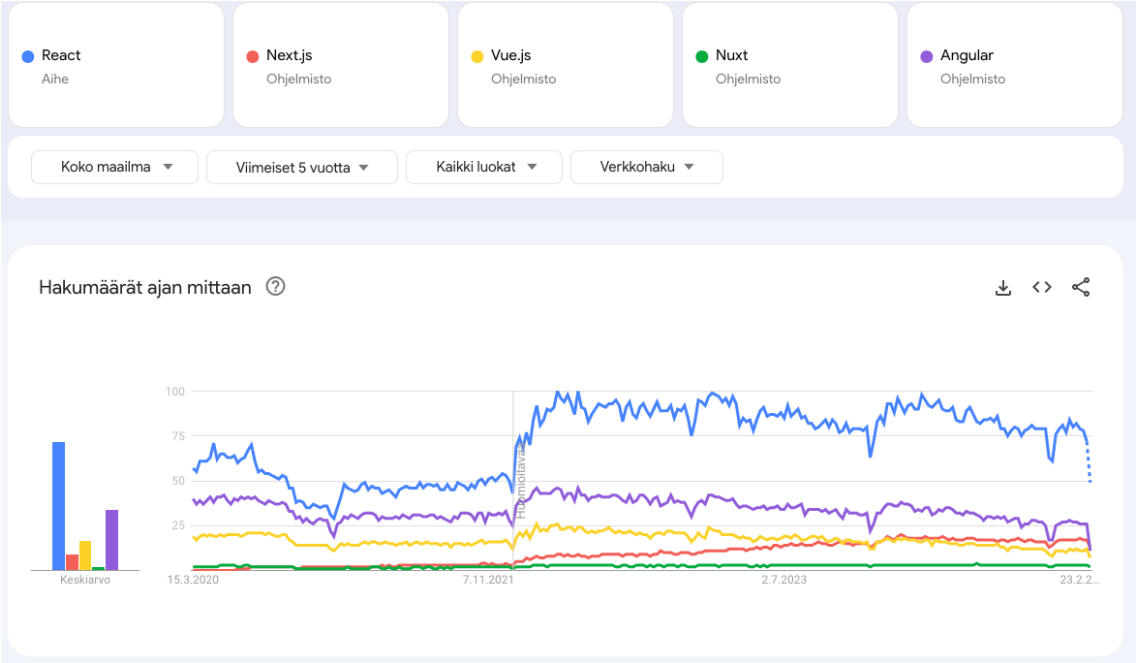
Palvelinpuoleisen renderöinnin etuina ovat yleisesti parempi suorituskyky sekä parempi hakukonenäkyvyys, mikä voi parantaa sivuston löydettävyyttä. Käytännön mittaukset osoittavat, että palvelinpuolella renderöidyt sivut latautuvat tyypillisesti nopeammin ja tulevat interaktiivisiksi lyhyemmässä ajassa verrattuna selainpuoleiseen renderöintiin, mikä parantaa käyttäjäkokemusta etenkin ensimmäisellä sivulatauksella. Selainpuolisen renderöinnin etuna on kuitenkin parempi joustavuus sekä palvelimen resurssien säästö, erityisesti silloin kun sisältä päivitetään usein käyttäjän toiminnan perusteella. Lisäksi selainpuolinen renderöinti mahdollistaa saumattoman käyttökokemuksen, koska sivuston osia voidaan päivittää ilman palvelimelta uuden HTML-sivun hakemista. [9.]

3 Teknologiat

CSR- ja SSR-teknologioita on tarjolla erittäin laaja valikoima. Uusia ratkaisuja kehitetään jatkuvasti lisää, sillä web-kehityksen tarpeet muuttuvat nopeasti ja sovelluksilta vaaditaan yhä parempaa suorituskykyä, joustavuutta ja käyttökokemusta. Monet uudet teknologiat pyrkivätkin olemaan entistä nopeampia ja tehokkaampia.

Vaikka kuka tahansa voi periaatteessa kehittää uusia web-teknologioita, laajimmin käytetyt ratkaisut ovat usein suurten teknologiayhtiöiden luomia ja alun perin niiden omiin käyttötarkoituksiin suunniteltuja. Ajan myötä näitä teknologioita kuitenkin julkaistaan avoimesti, minkä seurauksena ne leviävät laajempaan käyttöön. Toisinaan kehittäjät luovat myös kokeellisia tai vaihtoehtoisia ratkaisuja testatakseen uusia ideoita tai ratkaistakseen ongelman omaperäisillä tavoilla.

Suosituimpiin JavaScript-sovelluskehyksiin kuuluvat React, Angular ja Vue. React on tällä hetkellä laajimmin käytetty (kuva 5), mihin vaikuttavat sen laaja yhteisö- ja kirjastotuki. Vue on verrattain kevyt ja melko helppo omaksua, mutta sillä on toistaiseksi pienempi käyttäjäkunta kuin Reactilla ja Angularilla. Angular on täysimittainen kehys, jonka käyttöönotto edellyttää muita ratkaisuja enemmän perehtymistä ja erityisesti TypeScript-osaamista. Vaikka React on usein valintalistojen kärjessä, jokainen näistä kehyksistä voi olla toimiva ratkaisu, mikäli projektin vaatimukset ja kehittäjätiimin osaaminen vastaavat kyseisen teknologian vahvuuksia ja haasteita. [10; 11.]



Kuva 5. Google Trends -vertailu: React, Next.js, Vue.js, Nuxt ja Angular. [12.]

React, Vue ja Angular tukevat oletusarvoisesti selainpuolista renderöintiä, jolloin käyttöliittymä rakennetaan selaimessa JavaScriptin avulla. Jokaiselle näistä on kuitenkin kehitetty myös palvelinpuolisen renderöinnin ratkaisuja. React hyödyntää Next.js-kehystä ja Vue puolestaan Nuxt-kehystä. Angularille on saatavilla Angular Universal. Näiden työkalujen avulla voidaan parantaa muun muassa hakukoneoptimointia.

3.1 React

Reactia pidetään yhtenä suosituimmista JavaScript-kirjastoista, ja sen kehitys aloitettiin Facebookilla vuonna 2013. Kirjastoa hyödynnetään erityisesti yksisivuisten verkkosovellusten toteuttamisessa. SPA-sovelluksissa käyttöliittymä ja sisältö päivittyvät dynaamisesti ilman, että sivu ladataan kokonaan uudelleen. Reactilla tyypillisesti toteutettuja sovelluksia ovat esimerkiksi sosiaalisen median alustat, verkkokaupat sekä erilaiset interaktiiviset verkkosovellukset. Tällaisissa sovelluksissa käyttäjämärät voivat olla suuria, jolloin Reactin hyödyntäminen vähentää palvelimelle kohdistuvaa kuormitusta, sillä sivuston rakenne muodostetaan vasta käyttäjän selaimessa.

Käyttöliittymä rakennetaan JSX-syntaksilla, joka yhdistää HTML:n ja JavaScriptin. JSX mahdollistaa sen, että HTML-koodi voidaan kirjoittaa suoraan JavaScriptin sisään. JSX muunnetaan tavalliseksi JavaScriptiksi rakennusvaiheessa esimerkiksi Babelin avulla. Vaihtoehtoisesti React-komponentteja voidaan kirjoittaa myös TypeScriptillä, jolloin käytetään TSX-syntaksia. Esimerkkikoodissa 2 havainnollistetaan TSX-syntaksilla toteutettua yksinkertaista laskurikomponenttia. Koodin avulla havainnollistetaan myös TSX- ja JSX-syntaksien toimintaa. [13.]

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

type CounterProps = {
  initialCount: number;
  title: string;
};

function Counter({ initialCount, title }: CounterProps) {
  const [count, setCount] = useState<number>(initialCount);

  return (
    <div>
      <h2>{title}</h2>
      <p>Arvo: {count}</p>
      <button onClick={() => setCount(count + 1)}>Lisää</button>
      <button onClick={() => setCount(count - 1)}>Vähennä</button>
    </div>
  );
}

export default Counter;
```

Esimerkkikoodi 2. Laskurikomponentti toteutettu TSX-syntaksilla.

Esimerkkikoodissa 2 käytetään Counter-komponentille annettavia propseja, joiden avulla komponentti saa tarvitsemansa alkuarvot ulkopuolelta. Komponentin sisäistä tilaa hallitaan reaktiivisesti useState-hookin avulla count-nimisessä muuttujassa, jota päivitetään setCount-funktion kautta. Koodissa käytetään TypeScriptiä, minkä vuoksi komponentille määriteltävät propsit on tyypitettävä. Komponentti renderöi annetun otsikon sekä laskurin arvon, jota voi lisätä tai vähentää painikkeilla.

React-sovellus koostuu uudelleenkäytettävistä komponenteista, jotka voivat olla funktiopohjaisia tai luokkapohjaisia. Nykyisessä React-versiossa suositaan

funktiokomponentteja. Komponenteilla voi olla state, eli sisäinen tila, jonka muutokset aiheuttavat komponentin uudelleen renderöinnin. Lisäksi komponenteille voidaan antaa ulkoisia arvoja eli propseja. [13.]

Sivun renderöinti Reactissa toteutetaan selainpuolella. Prosessi käynnistyy, kun render-funktio sijoitetaan haluttuun HTML-elementtiin, minkä jälkeen React ylläpitää käyttöliittymää virtuaalisen DOMin avulla. Kun komponentin tila muuttuu, uutta ja aiempaa virtuaalista DOMia verrataan keskenään, ja vain muuttuneen osat päivitetään selaimen todelliseen DOMiin [14]. Tämän ansiosta käyttöliittymä toimii nopeasti ja responsiivisesti. [13.]

3.2 Next.js

Next.js on Reactin pohjalle rakennettu JavaScript-sovelluskehys. Sen keskeisenä tehtävä on mahdollistaa React-sovellukselle palvelinpuolen renderöinti. Se käyttää sivujen rakentamiseen Reactista tuttua JSX-syntaksia. Kehyksessä voidaan hyödyntää myös Reactin ominaisuuksia, kuten komponenttien tilanhallintaa, koukkuja ja muita Reactin vakioituja toiminnallisuuksia.

Next.js-kehyksessä on kolme keskeistä tapaa toteuttaa sivujen renderöinti. Ensimmäisenä on perinteinen palvelinpuolen renderöinti (SSR), jossa jokainen sivupyyntö käsitellään palvelimella, ja HTML-dokumentti muodostetaan kokonaan ennen lähettämistä selaimelle. Toinen tapa on staattinen sivustogenerointi (SSG), jossa HTML-dokumentit generoidaan valmiiksi sovelluksen rakennusvaiheessa. Tämä lähestymistapa mahdollistaa nopeimman mahdollisen latauskokemuksen silloin, kun sivujen sisältö ei muutu usein. Kolmantena menetelmänä Next.js tarjoaa inkrementaalisen staattisen generoinnin (ISR), joka yhdistää SSR:n ja SSG:n parhaat puolet. ISR-menetelmässä sivut ovat pääosin staattisia, mutta niiden sisältöä voidaan päivittää määräajoin ilman koko sivuston uudelleenrakentamista. Jokaiselle näistä renderöintitavoista Next.js tarjoaa valmiit funktiot ja selkeät toteutusratkaisut. Taulukossa 1 on havainnollistettu näiden renderöintimenetelmien keskeisiä eroja. [14.]

Taulukko 1. Next.js:n renderöintimenetelmien vertailu

Menetelmä	**HTML:n luonti**	**Ensilataus**	**Dynaamisuus**	**SEO**
CSR	Selain	Hidas	Korkea	Heikko
SSR	Palvelin	Nopea	Korkea	Hyvä
SSG	Rakennusvaihe	Erittäin nopea	Matala	Hyvä
ISR	Rakennusvaihe + palvelin	Erittäin nopea	Keskitaso	Hyvä

Next.js-kehyksestä julkaistiin vuonna 2022 versio 13, joka toi mukanaan merkittäviä muutoksia suorituskykyyn ja kehitystyön tehokkuuteen. Uuden app-hakemistorakenteen myötä sovelluksen reititys määräytyy kansioiden ja niiden sisällä olevien tiedostojen nimien perusteella. Jokaisen reitin käyttöliittymänäkymä määritellään kansioon erikseen luotavalla page.tsx-tiedostolla. Komponentit renderöidään oletusarvoisesti palvelimella, ellei tiedoston alkuun lisätä ”use client” -komentoa, jolloin renderöinti tapahtuu selainpuolella. Suorituskykyä parannettiin muun muassa uudella JavaScript-paketoijalla (Turbopack), Stream Loading -tekniikalla sekä uusilla Next.js-komponenteilla, kuten Image ja Link. [15.]

Datan haku rajapinnasta tai tietokannasta voidaan toteuttaa synkronisesti tai asynkronisesti. Synkronisessa toteutuksessa näkymä rakennetaan palvelinpuolen renderöinnin komponenteilla ja hyödyntäen JavaScriptin fetch-komentoa sekä async ja await -syntaksia [15]. Näin sivu rakennetaan palvelimella kokonaan valmiiksi vasta, kun data on ladattu HTML-dokumenttiin, minkä jälkeen valmis dokumentti lähetetään selaimelle.

Asynkronisessa toteutuksessa data haetaan selaimessa esimerkiksi Reactin useEffect-koukun avulla. Tässä tapauksesa on kuitenkin huomioitava, että

tiedoston alussa tulee käyttää ”use client” -komentoa, jotta selain suorittaa komponentin renderöinnin palvelimen sijaan. Lisäksi Next.js mahdollistaa hybriditoteutuksen, jossa osa datasta haetaan ensin palvelimella ja loppu haetaan selaimessa asynkronisesti.

4 Tekninen vertailu

Renderöintimenetelmän valintaan vaikuttaa sovelluksen käyttötarkoitus. CSR-menetelmän ylläpitokustannukset ovat huomattavasti pienemmät verrattuna SSR-menetelmään, sillä SSR vaatii tehokkaampia palvelimia varsinkin tilanteissa, joissa sivuston käyttäjämäärä on suuri. Renderöintimenetelmien välillä on myös muita merkittäviä eroja, jotka tulee huomioida valintaa tehdessä.

4.1 Suorituskyky

Palvelinpuolen ja selainpuolen renderöintimenetelmien välillä esiintyy suorituskykyeroja. Erot tulevat selkeimmin esiin silloin, kun sivusto on monimutkainen ja sisältää runsaasti sisältöä, kuten kuvia ja API-kutsuja. Myös internetyhteyden nopeus vaikuttaa merkittävästi suorituskykyeroihin. Renderöintimenetelmissä sivujen rakentaminen tapahtuu eri laitteilla, joten sekä palvelimen että käyttäjän laitteen komponentit ja suorituskyky vaikuttavat latausaikoihin.

SSR-menetelmä on yleisesti nopeampi lataamaan sivuston sisällön käyttäjän näkyviin kuin CSR-menetelmä. Nopeampi latausaika johtuu siitä, että HTML-dokumentti rakennetaan valmiiksi jo palvelimella, jolloin sisältö voidaan näyttää selaimessa välittömästi ilman JavaScriptin lataamista ja suorittamista. JavaScriptiä käytetään tällöin ainoastaan sivuston toiminnallisuuden, kuten painikkeiden toimintojen toteuttamiseen. Tämä suorituskykyero korostuu erityisesti monimutkaisilla sivustoilla, joissa on runsaasti sisältöä ja API-kutsuja, sillä palvelimella API-kutsut voidaan käsitellä nopeammin ja tehokkaammin kuin selaimessa. Sen sijaan yksinkertaisilla sivustoilla, joiden sisältöä on vähän, suorituskykyerot ovat erittäin pieniä, sillä selain pystyy käsittelemän ja renderöimään

vähäisen JavaScriptin määrän nopeasti. Tällaisissa tapauksissa latausajat ovat lähes identtiset SSR- ja CSR-menetelmillä. [16.]

Suorituskykyerot korostuvat erityisesti hitailla internetyhteyksillä. CSR-menetelmässä selaimelta lähetetään ylimääräisiä pyyntöjä palvelimelle JavaScript-tiedostojen ja datan hakemiseksi, mikä hidastaa sivuston latautumista merkittävästi [16]. SSR-menetelmässä viivettä vähennetään siten, että palvelimelta lähetetään koko sivuston sisältö yhdessä paketissa, jolloin ylimääräisten pyyntöjen tarve vähenee tai poistuu kokonaan.

Näistä syistä SSR-menetelmä tarjoaa paremman suorituskyvyn useimmissa käytännön tilanteissa, joissa sisältö on monimutkaista tai verkkoyhteydet vaihtelevia. CSR soveltuu parhaiten yksinkertaisiin ja interaktiivisiin sovelluksiin, joissa käyttäjällä on nopeat yhteydet ja tehokkaat laitteet.

4.2 Turvallisuus

Turvallisuutta on vaikea mitata luotettavasti pelkästään lähdekoodin tai yksittäisten sovellusten perusteella [1]. Turvallisuuden arviointi edellyttää yleensä organisaatiotason tarkastelua sekä tietoa todellisista turvallisuusrikkomuksista. Menetelmiä voidaan kuitenkin vertailla sen perusteella, kuinka alttiita ne ovat yleisesti tunnetuille haavoittuvuuksille.

CSR-menetelmässä JavaScript suoritetaan selaimessa, mikä lisää sivuston riskiä joutua Cross-Site Scripting (XSS) -hyökkäyksen kohteeksi. XSS-hyökkäyksessä sivulle injektoidaan haitallista skriptiä esimerkiksi lomakkeiden välityksellä. Kun sivu avataan toisella selaimella, haitallinen skripti tulkitaan luotettavaksi ja se suoritetaan selaimessa. Näin käyttäjän tiedot voivat päätyä hyökkääjän haltuun. SSR-menetelmässä sivu rakennetaan jo palvelimella, mikä vähentää riskiä selaimessa suoritettaviin XSS-hyökkäyksiin. [17.]

SSR-menetelmässä on olemassa riski joutua hyökkäyksen kohteeksi, joissa palvelimen resursseja kuormitetaan liiallisesti esimerkiksi lähettämällä suuri

määrä pyyntöjä samanaikaisesti. Tällainen tilanne tunnetaan palvelunestohyökkäyksenä (DDoS). Siinä järjestelmä pyritään ylikuormittamaan tahallisesti. Seurauksena palvelu saattaa hidastua merkittävästi tai lakata toimimasta kokonaan. Tämänkaltaiset hyökkäykset voivat aiheuttaa merkittäviä taloudellisia vahinkoja.

Turvallisuuden tasoa ei määritä käytetty renderöintimenetelmä vaan se, kuinka huolellisesti valittuja tietoturvakäytäntöjä ja ohjelmistokehityksen parhaita menetelmiä sovelletaan. Turvallisuus saavutetaan noudattamalla hyviä käytäntöjä, ei pelkästään teknologian valinnalla.

4.3 Resurssien käyttö

Molemmat renderöintimenetelmät vaativat palvelimen, johon erilaisilta laitteilta lähetetään pyyntöjä verkon kautta ja nettisivun tiedostot vastaanotetaan. Palvelintarpeet eroavat kuitenkin merkittävästi menetelmän mukaan, sillä palvelimen menetelmien välillä on eroa kapasiteetin ja suorituskyvyn vaatimuksissa. Menetelmien välillä on eroja myös ympäristövaikutuksissa, kuten energiankulutuksessa ja hiilijalanjäljessä.

SSR-menetelmässä palvelinta kuormitetaan merkittävästi enemmän, sillä sivun HTML-dokumentti rakennetaan kokonaan palvelimella. Palvelimelta vaaditaan tällöin suurempaa laskentatehoa ja enemmän muistia, jotta sivujen renderöinti pystytään suorittamaan tehokkaasti ja nopeasti. CSR-menetelmässä sen sijaan kuormitus kohdistuu käyttäjän laitteeseen, jossa sivujen rakentaminen tapahtuu selaimessa JavaScriptin avulla. Näin ollen CSR-menetelmä edellyttää käyttäjän laitteelta riittävää laskentatehoa ja muistia. Palvelimen kevyempi kuormitus mahdollistaa CSR-menetelmälle paremman skaalautuvuuden. Lisäksi CSR:ssä käytetään yleensä selainpuolen reititystä, jolloin sovelluksen sisäinen navigointi ei vaadi jatkuvia uusia pyyntöjä palvelimelle, koska reititystiedot ja tarvittavat resurssit on ladattu selaimeen jo ensimmäisellä pyynnöllä. [1; 16.]

Kuormitus kohdistuu eri renderöintimenetelmissä eri laitteisiin, mutta tärkeää on huomioida myös kaistanleveyden käyttö. CSR-menetelmässä selaimelle

lähetettävät tiedostot ovat huomattavasti pienempiä kuin SSR-menetelmässä. Pienempi vähentää tarvittavat tallennustilan määrää sekä verkkoliikennettä, mikä vaikuttaa suoraan energiakulutukseen erityisesti pitkillä aikavälillä. [18.]

Kun huomioidaan molempien renderöintimenetelmien kokonaisenergiakulutus, mukaan lukien kaikki laitteet ja verkkoliikenteen aiheuttava kuormitus, erot jäävät hyvin vähäisiksi. Ympäristöystävällisyyden arviointi on näin ollen haastavaa, sillä siihen vaikuttaa merkittävästi se, millä tavoin sähkö on tuotettu.

4.4 Hakukoneoptimointi

Hakukoneiden rooli on keskeinen verkkosivustojen löydettävyyden kannalta. Mikäli sivustoa ei löydetä hakukoneiden kautta, jää potentiaalinen käyttäjäliikenne vähäiseksi. Hakukone lähettää verkkoon automaattisia ohjelmia, joita kutsutaan crawlereiksi, spidereiksi tai boteiksi. Näiden ohjelmien tehtävänä on selata verkkosivustoja järjestelmällisesti ja kerätä niiltä tietoa, kuten tekstisisältöä, otsikoita, kuvia, metatietoja ja linkkejä. Mikäli indeksointirobotit eivät pääse käsiksi sivun sisältöön, ei sivua voida sisällyttää hakutuloksiin. Tämän vuoksi sivut tulee rakentaa hakukoneoptimointia (SEO) hyödyntäen, mikäli tavoitteena on saavuttaa parempi näkyvyys ja suurempi käyttäjämäärä. [19.]

SSR-menetelmän etuna on, että hakukoneille tarjotaan valmis HTML-dokumentti, mikä helpottaa sivun sisällön indeksointia. Tämä parantaa merkittävästi sivun löydettävyyttä ja sijoitusta hakutuloksissa. CSR-menetelmässä HTML-dokumentti on aluksi lähes tyhjä, ja varsinainen sisältö rakennetaan selaimessa JavaScriptin avulla. Useimmat hakukoneet eivät kuitenkaan odota JavaScriptin suorittamista, minkä vuoksi sisällön indeksointi voi jäädä puutteelliseksi. Esimerkiksi Googlen hakukone kykenee suorittamaan JavaScriptiä, mutta prosessi on selvästi hitaampi ja epävarmempi kuin valmiin HTML:n lukeminen. Tämän vuoksi CSR-menetelmällä saavutetaan rajallinen hakukoneoptimointi, kun taas SSR-menetelmä tarjoaa siihen selkeän edun. [1;16.]

5 Käytännön toteutukset

Tarkoituksena on rakentaa kaksi sisällöltään ja toiminnoltaan samanlaista web-sovellusta, joista toinen toteutetaan Reactilla ja toinen Next,js:llä. Sivustojen erilaisia ominaisuuksia vertaillaan keskenään useiden osa-alueiden perusteella. Vertailussa kiinnitetään huomiota seuraaviin asioihin:

- kehitystyön vaivannäkö
- suorituskyky
- hakukoneoptimointi.

5.1 Sisältö

Sivuista pyritään tekemään mahdollisimman monimutkaiset ja raskaat, jotta niiden testaaminen vastaisi mahdollisimman hyvin todellisten sovellusten sisältöä ja kuormitusta. Molemmat sivustot toteutetaan perusrakenteeltaan samanlaisiksi ja niiden sisältö sekä ominaisuudet ovat yhteneväiset. Sivut tyylitellään samalla tavalla, jotta lopputuloksena syntyy kaksi rakenteeltaan ja ulkoasultaan identtistä sovellusta.

Sivuihin pyritään lisäämään mahdollisimman paljon erilaisia kuormittavia tekijöitä, kuten API-kutsuja, kuvia, useita sivuja ja suuria määriä DOM-elementtejä. Näiden avulla sivujen rakentamisesta tehdään raskaampaa, eikä lopputuloksena synny pelkistettyä sovellusta, jossa testitulokset olisivat hyvin samankaltaisia molempien menetelmien välillä.

Autentikoinnin lisääminen vastaisi paremmin oikeiden sivustojen rakennetta ja palvelinten kuormitusta, mutta sen vaikutus testituloksiin arvioidaan pieneksi verrattuna sen toteuttamiseen käytettyyn aikaan. Näin ollen autentikointia ei pidetä keskeisenä osana vertailua.

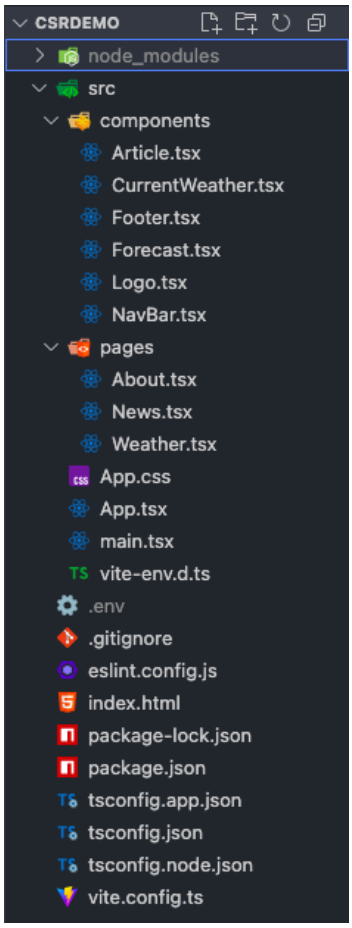
Sovelluksia rakentaessa pyritään noudattamaan parhaita ohjelmointikäytäntöjä. Sivustoihin lisätään asianmukaiset metatiedot, kuten otsikot, kuvaukset ja avainsanat, mikä parantaa sivuston löydettävyyttä hakukoneissa. HTML-

elementtejä käytetään oikeaoppisesti ja semanttisesti, mikä helpottaa hakukoneiden robotteja ymmärtämään sivuston rakennetta ja sisältöä. Sivustojen suorituskykyä optimoidaan, sillä nopeammat latausajat parantavat käyttäjäkokemusta. Lisäksi sivustot suunnitellaan mahdollisimman responsiivisiksi, jotta ne toimivat hyvin myös mobiililaitteilla, sillä hakukoneet huomioivat responsiivisuuden sijoituksia arvioidessa.

Sovellukset sisältävät navigointipalkin, jonka avulla käyttäjien on mahdollista siirtyä sovelluksen eri sivuille. Sovellusten pääasiallisena tarkoituksena on toimia uutissivustona, joista voidaan lisäksi tarkastella säätietoja. Kaikki tämä data haetaan API-kutsujen avulla: uutiset haetaan NewsAPI:sta ja säätiedot Weather API:sta. Sovellukset sisältävät myös staattisen ”Tietoa meistä” -sivun, joka vastaa yleistä käytäntöä verkkosovelluksissa. Näin voidaan testata monipuolisesti raskasta sovellusta, joka rakennetaan molemmilla menetelmillä.

5.2 React-sovellus

Selainpuolen renderöintimenetelmää hyödyntävä sovellus rakennetaan React-kirjastolla. Sovelluksen alustamiseen käytetään perinteisen Create React App-työkalun sijasta Vite-työkalua. Molemmilla tavilla saadaan toimiva React-sovellus, mutta Vite tarjoaa nopeamman kehityspalvelimen, paremman tuen Hot Module Replacementille (HMR) sekä modernimman lähestymistavan ES-moduulien käyttöön. Lisäksi Viten avulla voidaan vaikuttaa helpommin erilaisiin konfiguraatioihin ja mukauttaa kehitysympäristöä tarpeiden mukaan. Kehitysympäristönä käytetään VSCode-editoria, jonka yhteydessä hyödynnetään GitHubin Copilot-lisäosaa koodin kirjoittamisen tukena. Kuvassa 6 on havainnollistettu React-sovelluksen hakemistorakennetta.



Kuva 6. React-sovelluksen hakemistorakenne.

Kuvasta 6 voidaan havaita, ettei sovelluksen hakemistorakenne ole monimutkainen. Kaikki sovelluksen toiminnan kannalta tärkeä sisältö sijaitsee src-kansiossa. Eri näkymät on sijoitettu omaan pages-kansioon, ja niiden välinen reititys toteutetaan react-router-dom-kirjaston avulla. Kirjaston toimintaa havainnollistetaan esimerkkikoodissa 3. Komponentit on erotettu omaan kansioonsa rakenteen selkeyttämiseksi, vaikka niitä ei käytetä uudelleen sovelluksen rajallisen laajuuden vuoksi. Tämä erottelu helpottaa myös mahdollista jatkokehitystä. Sovelluksen kehityksessä on käytetty oikeaoppisesti HTML:n semanttisia elementtejä ja pyritty tekemään siitä mahdollisimman optimoitu sekä hakukoneystävällinen.

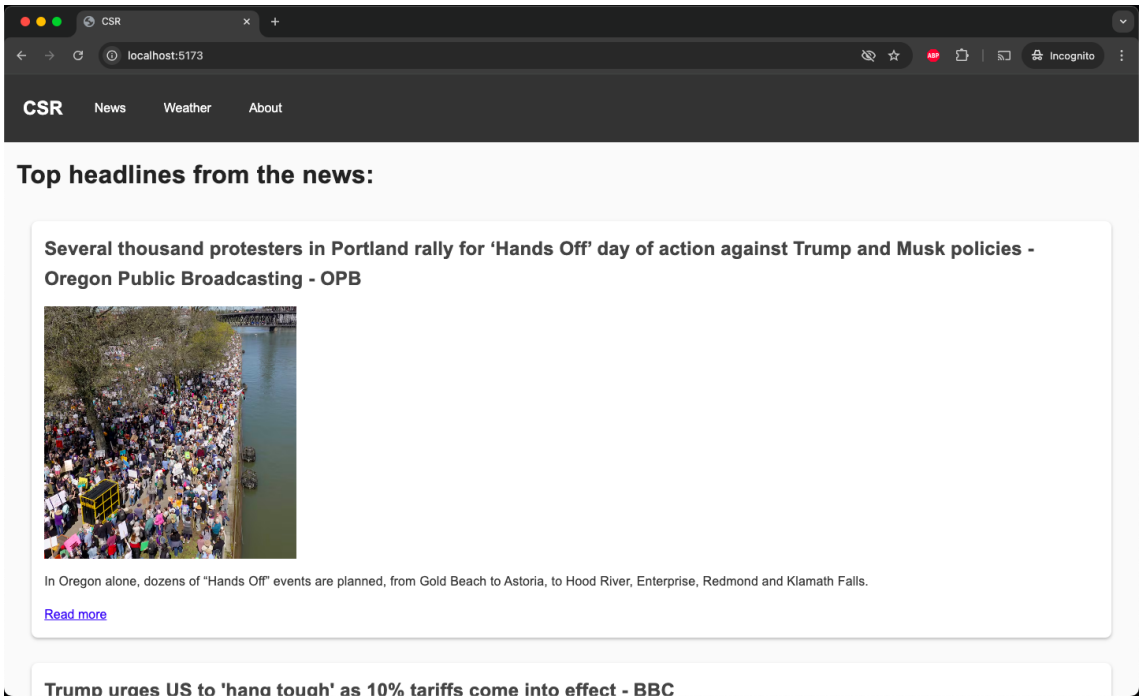
```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import "./App.css";
import News from "./pages/News";
import Weather from "./pages/Weather";
import About from "./pages/About";
import NavBar from "./components/NavBar";
import Footer from "./components/Footer";

export default function App() {
  return (
    <BrowserRouter>
      <NavBar />
      <Routes>
        <Route path="/" element={<News />} />
        <Route path="/weather" element={<Weather />} />
        <Route path="/about" element={<About />} />
      </Routes>
      <Footer />
    </BrowserRouter>
  );
}
```

Esimerkkikoodi 3. React-sovelluksen App.jsx-tiedoston sisältö.

Esimerkkikoodista 3 voidaan havaita, kuinka react-router-dom-kirjasto toimii käytännössä. Kirjasto hallitsee DOM-rakennetta vaihtamalla näkyvillä olevaa sivua reitityksen mukaisesti. Navigointi- ja alatunnistepalkit voidaan sijoittaa App.tsx-tiedostoon, jota voidaan pitää sovelluksen yleisenä näkymänä, jolloin palkit näkyvät kaikissa sovelluksen osissa. Lisäksi kaikki näkymät ladataan selaimeen jo ensimmäisen verkkopyynnön yhteydessä, mikä tekee sivujen välisestä navigoinnista tehokasta ja nopeaa.

Sivuston tyylittely on toteutettu perinteisellä CSS:llä, joka sijaitsee CSS-tiedostossa. Alkuperäisenä ajatuksena oli käyttää tyylittelyyn styled-components-kirjastoa, mutta sen käytössä ilmeni ongelmia Next.js-sovelluksessa. React-sovellusta pyrittiin kuitenkin kehittämään siten, että Next.js-sovelluksen toteuttaminen olisi mahdollisimman nopeaa ja helppoa, koska React-sovelluksessa kirjoitettua koodia voitaisiin käyttää uudelleen myös Next.js-sovelluksessa. Kuvassa 7 on havainnollistettu sivuston sisältöä ja tyylittelyä.

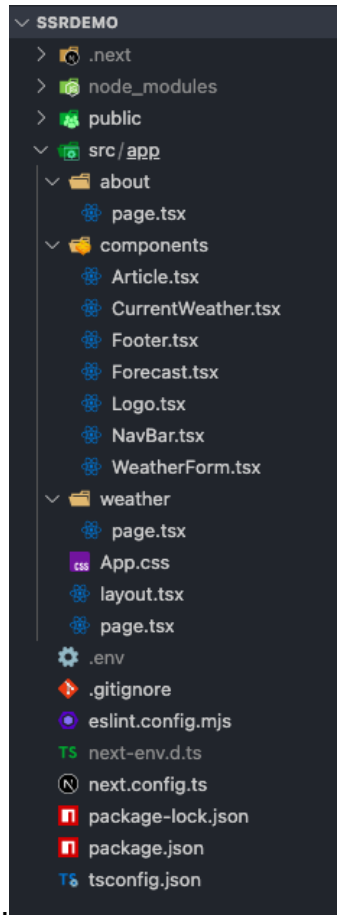


Kuva 7. React-sovelluksen uutisnäkymä.

Sovelluksessa on sisällytetty aiemmin mainitut ominaisuudet. Uutiset ja säätiedot haetaan eri rajapinnoista. Kuvasta 7 voidaan havaita, että uutisartikkelit sisältävät sekä tekstiä että kuvan. Rajapinnasta haetaan kerralla useita artikkeleita, jolloin sovellukseen syntyvä kuorma vastaa mahdollisimman hyvin todellisen verkkosovelluksen kuormaa.

5.3 Next.js-sovellus

Palvelinpuolen renderöintimenetelmää hyödyntävä verkkosovellus toteutetaan Next.js-kehyksellä. Sovellus alustetaan käyttämällä npx create-next-app@latest, joka luo valmiin pohjan sovelluksen kehittämiselle. Hakemistosta poistetaan ylimääräiset tiedostot, jäljelle jääneitä tiedostoja muokataan ja uusia luodaan, jotta sovellukseen saadaan halutut ominaisuudet. Kehitysympäristö on sama kuin React-sovellusta kehittäessä. Kuvassa 8 on havainnollistettu sovelluksen hakemistorakennetta.



Kuva 8. Next.js-sovelluksen hakemistorakenne.

Kuvasta 8 voidaan havaita, että Next.js-sovelluksen hakemistorakenne on hyvin selkeä. Sovelluksen syntaksina käytetään TSX:ää ja komponentit on otettu suoraan React-sovelluksesta. Rajapinnan pyyntöjen optimoimiseksi on luotu erillinen WeatherForm.tsx-tiedosto. Next.js-version 13 myötä sovelluksen sisältö sijaitsee app-kansiossa. Samassa versiossa datan haku voidaan toteuttaa esimerkiksi JavaScriptin perinteisellä fetch-komennolla, jolloin aiemmin käytössä olleita getServerSideProps()-funktion tapaisia ei enää käytetä. Esimerkkikoodissa 4 havainnollistetaan, miten dataa haetaan palvelinpuolelta Next.js:n versiolla 13.

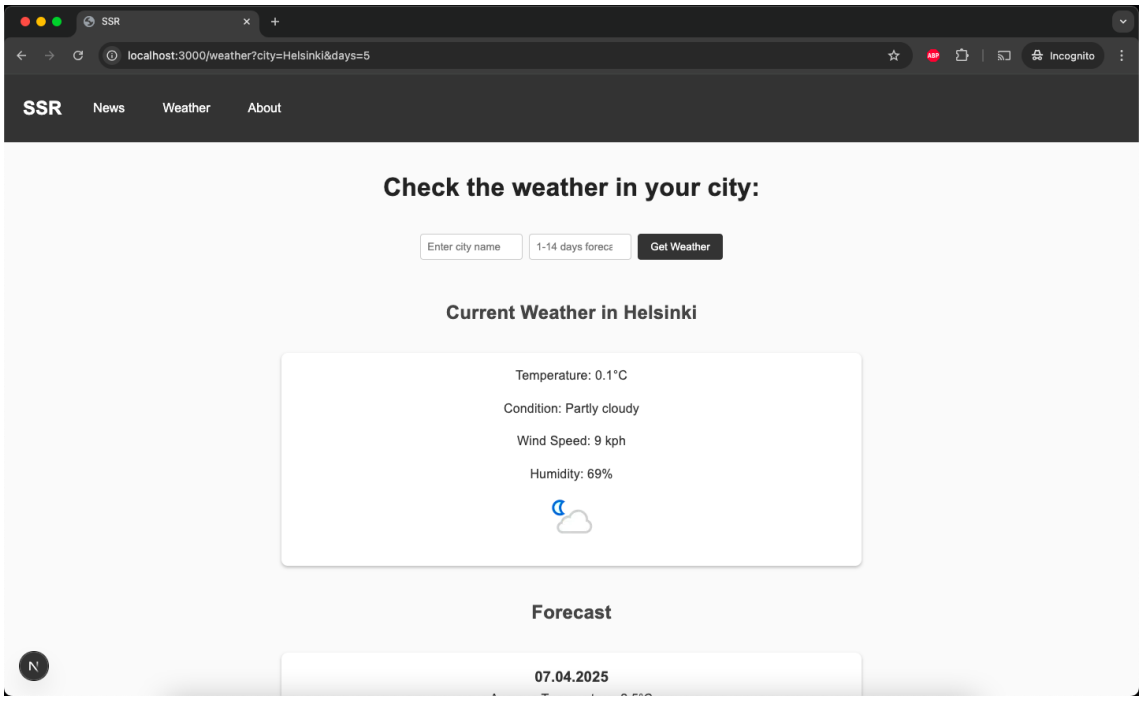
```
export default async function NewsPage() {
  const url = "https://newsapi.org/v2/top-headlines?country=us";
  const key = process.env.VITE_NEWS_APIKEY;
  const query = `${url}&apiKey=${key}`;

  let articles: ArticleData[] = [];
  try {
    const res = await fetch(query, { next: { revalidate: 60 } });
    if (!res.ok) {
      throw new Error("Failed to fetch news data");
    }
    const data = (await res.json()) as NewsData;
    articles = data.articles;
  } catch (error) {
    console.error("Error fetching news:", error);
  }

  return (
    <>
      <h1>Top headlines from the news:</h1>
      <div className="article-wrapper">
        {articles.length > 0 ? (
          articles.map((article, index) => (
            <Article key={index} article={article} />
          ))
        ) : (
          <p>No articles found.</p>
        )}
      </div>
    </>
  );
}
```

Esimerkkikoodi 4. App-kansiossa oleva page.tsx-tiedosto.

Sivun tyylittely on toteutettu kopioimalla React-sovelluksessa käytetty CSS-tiedosto. Vaihtoehtona kokeiltiin myös styled-komponents-kirjastoa, mutta sen käyttö edellyttää, että komponentit renderöidään selaimella. Next.js-versiossa 13 komponentit renderöidään oletuksena palvelimella, minkä vuoksi kirjastoa ei voitu hyödyntää, ellei renderöintitapaa olisi muutettu selainpuolelle. Tällöin kuitenkin koko vertailun alkuperäinen idea menetettäisiin, koska molemmat sovellukset käyttäisivät samaa renderöintitapaa. Kuvassa 9 on havainnollistettu Next.js-sovelluksen tyylittelyä.



Kuva 9. Next.js-sovelluksen säänäkymä.

Data haetaan palvelimelta ja sivu rakennetaan siellä kokonaan valmiiksi. Rajapinnasta haettu data tallennetaan palvelimen välimuistiin ja päivitetään automaattisesti 60 sekunnin välein. Näin vältetään jatkuvien API-kutsujen lähettäminen, sillä käyttäjän päivittäessä sivua data palautetaan välimuistista, mikäli edellisestä päivityksestä on kulunut alle 60 sekuntia. Tämän mahdollistaa fetch-komennon sisältämä revalidate-ominaisuus.

6 Tulokset

6.1 Mittausmenetelmät ja työkalut

Alkuperäisenä tarkoituksena oli sijoittaa sovellukset julkiselle verkkopalvelimelle. Tämä ei kuitenkaan ollut mahdollista, sillä NewsAPI:n ilmaisen version käyttöehdot rajoittivat rajapinnan käytön ainoastaan kehitysympäristöihin. Tästä syystä mittaukset toteutetaan paikallisesti kahden tietokoneen avulla, joista toinen toimii palvelimena ja toinen asiakaskoneena. Molemmat sovellukset rakennetaan ja käynnistetään samalla tavalla kuin ne toimisivat julkisessa

ympäristössä, mikä takaa realistiset testausolosuhteet ja mahdollistaa tulosten vertailukelpoisuuden. Sovelluksiin ei lisätä ylimääräisiä kuormittavia tekijöitä, kuten tarpeettomia kirjastoja tai raskasta JavaScript-logiikkaa, jotka kasvattaisivat merkittävästi selaimen laskentatehovaatimuksia ja voisivat vaikuttaa mittaustuloksiin. Tarkoituksena on säilyttää sovellukset mahdollisimman realistisina ja optimoituina, sillä myös monimutkaiset sivustot voidaan rakentaa suorituskykyisiksi oikeilla kehitys- ja optimointikäytännöillä. Näin mittaustulokset kuvastavat todellisia kehitystilanteita ilman keinotekoista kuormituksen lisäämistä.

Mittauksissa käytetään Google Chromen kehittäjätyökaluihin sisältyvää Google Lighthousea. Google Lighthouse on työkalu, jolla voidaan auditoida ja optimoida verkkosivustojen laatua useista eri näkökulmista. Työkalu generoi raportin, jossa jokainen kategoria pisteytetään asteikolla 0–100. Arvioitavat kategoriat ovat suorituskyky, saavutettavuus, hyvät käytännöt sekä hakukoneoptimointi. Mittauksissa keskitytään kaikkiin kategorioihin, mutta erityistä huomiota kiinnitetään suorituskykyyn ja hakukoneoptimointiin.

6.2 Kehitystyön vaivannäkö

Reactin ja Next.js:n käyttö edellyttää kehittäjältä jonkin verran aiempaa ohjelmointikokemusta. Erityisesti JavaScriptin perusteiden hallinta on hyödyllistä ennen näiden teknologioiden käyttöönottoa. Molemmat teknologiat ovat kuitenkin suhteellisen helppoja oppia, ja aikaisempi kokemus niiden käytöstä helpotti kehitystyötä merkittävästi. Lisäksi molemmat käyttävät samaa TSX-syntaksia, mikä mahdollisti React-sovelluksessa kirjoitettujen komponenttien ja muun koodin uudelleenkäytön Next.js-sovelluksessa. Tämä nopeutti huomattavasti jälkimmäisen sovelluksen kehittämistä. Next.js:n valmiit rakenteet ja ominaisuudet, kuten automaattinen reititys ja palvelinpuolen renderöinti, vähensivät huomattavasti manuaalisen konfiguroinnin tarvetta, mikä osaltaan paransi kehitystyön tehokkuutta.

6.3 Suorituskyky

Suorituskykymittaukset vahvistavat aiemmin tutkitun teorian paikkansapitäväksi. Google Lighthouse antaa suorituskyvyn pisteytyksessä uutissivulle SSR-menetelmällä arvoksi 93 ja CSR-menetelmällä arvoksi 86. Suurimmat suorituskykyyn vaikuttavat tekijät uutissivulla ovat kuvien lataaminen sekä CSR:n tapauksessa JavaScriptin suuri määrä. Taulukossa 2 on esitetty suorituskyvyn tulokseen vaikuttavia aikoja.

Taulukko 2. Uutisnäkymän Google Lighthouse tulokset

Menetelmä	**FCP**	**LCP**	**TBT**
CSR	1. 1 s	2. 2 s	0 ms
SSR	0. 3 s	0. 6 s	220 ms

Taulukosta 2 voidaan havaita selvä ero suorituskyvyssä. Mittari FCP (First Contentful Paint) kertoo ajan ensimmäisestä elementistä, joka piirtyy sivulle. Mittari LCP (Largest Contentful Paint) ilmaisee ajan, joka kuluu suurimman sivulla olevan kuvan latautumiseen kokonaisuudessaan. Mittari TBT (Total Blocking Time) kuvaa aikaa, joka kuluu ensimmäisestä piirrosta siihen, kun sivun kaikki toiminnot ovat käyttäjän käytettävissä.

Sovellusten säänäkymän mittaustulokset ovat hyvin samankaltaiset kuin uutissivujen tulokset. Merkittävimpänä erona ovat lyhyemmät latausajat, jotka pätevät molempiin renderöintimenetelmiin. Lyhyemmät ajat johtuvat siitä, että säänäkymässä käytetyt kuvat ovat huomattavasti pienempiä ja niitä on määrällisesti vähemmän.

Staattisilla ”Tietoa meistä” -sivuilla suorituskykyerot eivät olleet merkittäviä, sillä sivut sisältävät pääosin tekstiä. Näin ollen sivut latautuvat nopeasti ja kevyesti molemmilla menetelmillä, jolloin havaittavat erot jäävät vähäisiksi.

SSR-menetelmän mittaustulokset osoittavat huomattavasti lyhyempiä latausaikoja, jolloin sisältö tulee käyttäjän näkyville nopeammin. CSR-menetelmän etuna on kuitenkin se, että sivun interaktiiviset ominaisuudet ovat välittömästi käytettävissä, vaikka sisällön latautuminen kokonaisuudessaan kestää hieman pidempään.

Vaikka Google Lighthouse simuloi hitaampaa laitetta ja matalampaa laskentatehoa, mittauksia suoritettiin myös todellisuudessa heikommalla tietokoneella. Tällöin suorituskyky heikkeni molemmilla menetelmillä selvästi. CSR:n latausajat hidastuivat noin puolella sekunnilla, kun taas SSR:n TBT-arvo heikkeni noin kahdella sekunnilla. Tämä johtuu siitä, että vaikka SSR-menetelmässä sivu rakennetaan palvelimella valmiiksi, sivun interaktiiviset toiminnot suoritetaan edelleen selaimessa JavaScriptin avulla, mikä aiheuttaa hitaammalla laitteella pidempää odotusaikaa.

6.4 Hakukoneoptimointi

Hakukoneoptimointia varten on käytettävissä useita eri keinoja. Sovelluksissa hyödynnettiin kaikki mahdolliset menetelmät parhaan mahdollisen lopputuloksen saavuttamiseksi. Molempiin sovelluksiin asetettiin asianmukaiset metatiedot, jotka hakukoneet huomioivat ensimmäisinä indeksointivaiheessa. Lisäksi noudatettiin muita suositeltuja käytäntöjä, joita Google Lighthouse -työkalu ehdotti SEO-kategoriassaan. Linkkeihin lisättiin kuvaavat tekstit ja varmistettiin, että painikkeet ja fontit olivat saavutettavia ja toimivat oikein myös mobiililaitteilla.

Google Lighthouse -työkalun tuloksissa ei havaittu merkittäviä eroja. CSR-sovellus sai suoritusarvosanaksi 92 pistettä ja SSR-sovellus 100 pistettä. Molemmat tulokset ovat hyviä, mutta on tärkeää huomioida, että SSR-menetelmällä hakukoneet pääsevät käsiksi kaikkeen sivustolla olevaan sisältöön, kuten uutisartikkeleihin, koska ne on sisällytetty valmiiksi HTML-dokumenttiin palvelimella. Vaikka työkalun antamat pisteet ovat lähellä toisiaan, hakukoneoptimoinnin todellinen vaikutus voi olla huomattava, sillä CSR-menetelmällä tuotettu

sisältö voi jäädä osittain indeksoimatta, mikä heikentää sivun sijoitusta hakutuloksissa.

Lisäksi SSR-menetelmällä rakennetussa sovelluksessa jokaisella reitillä on oma erillinen HTML-dokumentti, mikä mahdollistaa niiden yksittäisen indeksoinnin hakukoneissa. CSR-menetelmässä reititys toteutetaan selaimessa DOM-elementtien manipuloinnilla saman HTML-dokumentin sisällä, minkä vuoksi hakukoneet eivät välttämättä huomioi kaikkia sivunäkymiä. Tästä syystä esimerkiksi CSR-menetelmän säänäkymä jää indeksoimatta, kun taas SSR-menetelmällä rakennettu säänäkymä saa oman indeksoidun sivunsa, jolloin käyttäjät voivat löytää säätietosivun suoraan hakukoneiden kautta ilman, että heidän tarvitsee kulkea uutissivun kautta.

7 Yhteenveto

Työn tavoitteena oli tutkia eroavaisuuksia palvelinpuolen ja selainpuolen renderöintimenetelmien välillä. Lopullisena tarkoituksena oli muodostaa selkeä käsitys siitä, missä tilanteissa kumpaakin menetelmää kannattaa käyttää. Eroja tutkittiin ensin teoreettisella tasolla, minkä jälkeen suoritettiin käytännön toteutus kehittämällä molemmilla menetelmillä sisällöltään ja rakenteeltaan samanlaiset web-sovellukset. Sovellusten eroja mitattiin, ja tavoitteena oli vahvistaa teoreettiset havainnot käytännön tuloksilla.

Renderöintimenetelmän valinta riippuu vahvasti sovelluksen lopullisesta käyttötarkoituksesta. Molemmilla menetelmillä on omat vahvuutensa ja heikkoutensa, eikä kumpikaan ole automaattisesti paras vaihtoehto kaikissa tilanteissa.

Palvelinpuolen renderöinti on suositeltavaa, kun kyseessä ovat esimerkiksi kaupalliset sivustot, kuten yrityksen verkkosivut tai verkkokaupat, joissa hakukoneoptimointi on kriittisen tärkeää. Lisäksi raskaat ja monimutkaiset sivut voidaan hallita tehokkaammin palvelinpuolella, sillä itse isännöijä voi vaikuttaa palvelimen suorituskykyyn ja näin myös sovelluksen nopeuteen ja luotettavuuteen.

Selainpuolen renderöinti on puolestaan suositeltavaa dynaamisille sovelluksille, kuten reaaliaikaisille chat-sovelluksille. Selainpuolen renderöinti vähentää palvelimen kuormitusta, mikä tarkoittaa matalampia ylläpitokustannuksia ja helpompaa skaalautuvuutta. Suorituskyky on tällöin pitkälti käyttäjän laitteen ja kehittäjän optimointityön varassa, mutta oikein toteutettuna selainpuolen renderöinti mahdollistaa erittäin tehokkaan toiminnan jopa heikoilla internetyhteyksillä.

Yhtä oikeaa renderöintimenetelmää ei ole olemassa. Menetelmä kannattaa valita sovelluksen käyttötarkoituksen ja potentiaalisten käyttäjien tarpeiden perusteella. Mahdollisuuksien mukaan voidaan myös hyödyntää molempien menetelmien vahvuuksia ja rakentaa hybriditoteutuksia.

Lähteet

1 Beke, Mathias. 2018. On the Comparison of Software Quality Attributes for Client-side and Server-side Rendering. Master's Thesis. University of Antwerp.

2 How the web works. 2025. Verkkoaineisto. MDN web docs. <https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works>. Luettu 22.2.2025.

3 The web standards model. 2025. Verkkoaineisto. MDN web docs. <https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/The_web_standards_model#server-side_languages_and_frameworks>. Luettu 24.2.2025.

4 Web Server Concepts and Examples. julkisuustilaisuus 5.10.2020. 2020. Verkkoaineisto. WebConcepts. <https://www.youtube.com/watch?v=9J1nJOivdyw>. Katsottu 24.2.2025

5 How Web Browsers Work. julkisuustilaisuus 27.5.2022. 2022. Verkkoaineisto. Tadas Petra. <https://www.youtube.com/watch?v=EoYkl8rwbiM>. Katsottu 24.2.2025.

6 Client-side Rendering (CSR). Verkkoaineisto. Next.js. <https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering>. Luettu 8.3.2025.

7 How browsers work. 2025. Verkkoaineisto. MDN web docs. <https://developer.mozilla.org/en-US/docs/Web/Performance/Guides/How_browsers_work>. Luettu 10.3.2025.

8 Grosskurth, Alan; Godfrey, Michael W. 2006. Architecture and evolution of the modern web browser. University of Waterloo. <https://plg.uwaterloo.ca/~migod/papers/2006/jss-browserRefArch.pdf>. Luettu 25.2.2025.

9 Taufan Fadhilah, Iskandar; Muharman, Lubis; Tien Fabrianti, Kusumasari; Arif Ridho, Lubis. 2020. Comparison between client-side and server-side rendering in the web development. Verkkoaineisto. IOPscience. <https://iopscience.iop.org/article/10.1088/1757-899X/801/1/012136/pdf>. Luettu 24.2.2025.

10 Levlin, Mattias. 2020. DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte. Master's. Åbo Akademi University. Doria-tietokanta.

11 Cincović, Jelica & Punt, Marija. 2020. Comparison: Angular vs. React vs. Vue. Which framework is the best choice? Information Society of Serbia – ISOS, University of Belgrade, School of Electrical Engineering.

12 Google Trends. 2025. Reactin, Vuen, Angularin, Next.js:n ja Nuxt:in suosiokehitys. Verkkoaineisto. <https://trends.google.com/trends/explore?date=today%205-y&q=%2Fm%2F012l1vxv,%2Fg%2F11h4q9rcf3,%2Fg%2F11c0vmgx5d,%2Fg%2F11g0wgnhgc,%2Fg%2F11c6w0ddw9&hl=fi>. Luettu 16.3.2025.

13 Gackenheimer, Cory. 2015. Introduction to React. Apress, Springer Science+Business Media.

14 Thakkar, Mohit. 2020. Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications. Apress, Springer Science+Business Media.

15 Zhao, Ziang. 2023. Build A Live News Application With Next.js 13. Bachelor's Thesis. Metropolia University of Applied Sciences. Theseus-tietokanta.

16 Nordström, Carl; Dixelius, August. 2023. Comparisons of Server-side Rendering and Client-side Rendering for Web Pages. Master's Thesis. Uppsala University.

17 Bratslavsky, Paul. 2025. Client-Side Rendering vs Server-Side Rendering: Key Differences and Use Cases for Developers. Verkkoaineisto. Strapi. < https://strapi.io/blog/client-side-rendering-vs-server-side-rendering>. Luettu 25.3.2025.

18 Finckelsen, Casper. 2023. Comparing the Environmental Impact of Server-side Rendering and Client-side Rendering. Master's Thesis. Malmö University.

19 Almukhtar, Firas; Mahmoodd, Nawzad; Kareem, Shabab. 2021. Search Engine Optimization: A Review. Applied Computer Science. vol 17, s.70-80.