Next.js-verkkosivujen kehittämiseen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus

kevät 2022

Petri Paukkunen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus | Tiivistelmä

Tekijä Petri Paukkunen
Vuosi 2022
Työn nimi Next.js-verkkosivujen kehittämiseen
Ohjaaja Lasse Seppänen

Opinnäytetyön tarkoituksena oli selvittää, kuinka Next.js-sovelluskehys sopii verkkokehitykseen. Tässä tutkittiin, miltä sen kehittäjäkokemus näyttää, mitä se tarjoaa Reactin lisäksi ja miten sillä tuotettujen verkkosivujen tiedonhakutavat eroavat toisistaan.

Opinnäytetyön tietopohja perustuu verkkosivujen perustekniikoihin, sekä niiden renderöintitapoihin. Tämän lisäksi käytiin läpi, mitä React tarjoaa verkkokehitykseen, sekä mitä Next.js tarjoaa sen päälle. Projektissa kehitettiin samanlainen sovellus käyttäen kolmea Next.js:n tarjoamaa renderöintitekniikkaa (SSR, CSR ja SSG). Sovelluksena toimi blogisivusto, ja testaustyökaluna käytettiin Googlen kehittämää Lighthouse-työkalua, jossa keskityttiin sen antamiin suorituskykyarvoihin.

Tutkimuksessa havaittiin, kuinka nämä eri tekniikat eroavat toisistaan. Vaikka SSR ja SSG eivät eronneet suuresti tuloksissa, on SSG-tekniikkaa käyttäen mahdollista nopeuttaa verkkosivua muilla tavoilla. Isompi ero oli kuitenkin CSR-tekniikassa, koska sen tiedonhaku tapahtuu sivun lataamisen jälkeen. Tämä vaikuttaa varsinkin jos tämä tieto on sivun pääaineisto, sekä jos käytössä on mobiililaite. Sen lisäksi todettiin, kuinka helppoa samassa projektissa on käyttää kaikkia näitä eri tekniikoita käyttäen Next.js:sää, valiten parhaiten sopiva tekniikka oikeaan tarkoitukseen.

Avainsanat React, Next.js, SSR, CSR, SSG

Sivut 30 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Author Petri Paukkunen
Subject Using Next.js for website development
Supervisors Lasse Seppänen

Abstract
Year 2022

The purpose of the thesis is to get a better understanding of the website development framework Next.js. The thesis focuses on the developer experience, additions Next.js provides over React.js and the different data fetching methods provided by Next.js.

The knowledge base for the thesis consisted of the basics for web development, and the different rendering methods used by modern frameworks. In addition, it will go over what React brings to web development and what Next.js provides on top of that. For the practical part, a similar blog site was developed using all three data fetching techniques Next.js provides (SSR, CSR and SSG). For testing the website, tool developed by Google called Lighthouse was used, where the focus is on comparing the performance metrics.

Based on the analysis, it can be seen how these different techniques differ. Although SSR and SSG did have similar results, it is possible to speed up SSG websites using other means. The bigger difference was on the CSR-technique, because it needs a full page load before the main page content gets downloaded and displayed, especially when using a slower device such as a phone. In addition, it was noted how easy Next.js makes using all of the different data fetching techniques, choosing and mixing where it makes sense.

Keywords React, Next.js, SSR, CSR, SSG
Pages 30 pages and appendices 1 page

Sanasto

HTML	HyperText Markup Language, verkkosivujen määrittelykieli.
CSS	Cascading Style Sheets, verkkosivujen tyylittelykieli.
JSX	JavaScript XML, ReactJS:n käyttämä JavaScript lisäosa, joka mahdollistaa HTML:n kirjoittamisen JavaScriptissä.
CSR	Client-Side Rendering, käyttäjälle lähetetään tyhjä sivu ja koodipaketti, joka rakentaa nettisivun ulkoasun selaimenpuolella.
SSR	Server-Side Rendering, verkkosivun prosessointi tapahtuu palvelimen puolella ja käyttäjälle lähetetään lähes valmis verkkosivu.
SSG	Static-Site Generation, verkkosivun valmiiksi prosessointi HTML-sivuiksi koontivaiheessa.
SEO	Search Engine Optimisation, hakukoneoptimointi, jolla tarkoitetaan hakukoneiden hakutuloksia parantavia toimenpiteitä.
NPM	Node Package Manager, JavaScript-kirjastojen kanssa käytetty palvelu jakamiseen ja hallitsemiseen.
API	Application Programming Interface, ohjelmointirajapinta, joka mahdollistaa eri ohjelmat pystyvät keskustelemaan keskenään.
REST	Representational State Transfer, arkkitehtuurimalli, joka helpottaa HTTP-rajapintojen luomista.
WCAG	Web Content Accessibility Guidelines, tarkoittaa Verkkosivujen esteettömyysohjeistusta.

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

Verkkosivut alkoivat yksinkertaisina sivuina, joilla pystyi näyttämään tekstiä, mutta ajan kuluessa niiden interaktiivisuus on kasvanut nopeasti. Tämän myötä nykyään pystytään luomaan monimutkaisia sovelluksia, joita kuka tahansa pystyy käyttämään suoraan selaimella. Tämän takia on myös syntynyt monia sovelluskirjastoja auttamaan verkkosovelluksen luomista, aikaisemmin jQuery ja tällä hetkellä esimerkiksi ReactJS ja VueJS.

Opinnäytetyössä tutkitaan NextJS-sovelluskehystä. Se on rakennettu ReactJS-kirjaston päälle, lisäten sekä parantaen sen ominaisuuksia. Tässä verrataan, miten NextJS eroaa ReactJS:stä, mitä se tarjoaa sen päälle ja mitä valintoja siinä on tehty, jotka saattavat vaikuttaa yrityksen päätökseen. Sen lisäksi selvitetään, miten sekä kehitys- että käyttäjäkokemus muuttuu, kun valitaan NextJS.

Projektissa luodaan toimiva blogisivu, joka mahdollistaa blogien listauksen sekä lukemisen. Blogisivut tullaan tekemään kolmella Next.js:n tarjoamalla tavalla, jotta pystytään testaamaan niiden eroja. Näiden testien perusteella saadaan parempi kuva niiden eroista sekä hyödyistä.

Opinnäytetyön tutkimuskysymykset ovat

- Kuinka Next.js parantaa kehittäjäkokemusta?
- Miten Next.js:n tarjoamat tiedonhakutekniikat eroavat?
- Miten Next.js:n sivujen suorituskyvyt eroavat?

2 Verkkosivujen tekniikat

Nykyisien verkkotekniikoiden kehitys alkoi vuonna 1989, kun Tim Berners-Lee aloitti kehittämään tietojärjestelyyn tarkoitettua järjestelmää hänen työnantajille. Parin vuoden kehityksen jälkeen verkon tärkeimpien rakennusosien, kuten HTTP-protokollan ja HTML-spesifikaation luonnokset julkaistiin julkisesti. Näitä teknologioita on jatkokehitetty ja päivitetty tuosta lähtien, ja niistä rakentuu modernin internetin infrastruktuuri. (Hoffmann, ei pvm.). Tässä osassa tutkitaan eri verkkokehityksen palasia ja käydään läpi eri tekniikoita, millä voidaan parantaa käyttäjäkokemusta.

2.1 HTML ja CSS

HTML, eli Hypertext Markup Language, on verkkosivujen pohjapiirustukseen käytetty merkkauskieli. Se mahdollistaa esimerkiksi otsikoiden, taulukoiden, listojen sekä kuvien lisäämisen verkkosivulle, sekä vuorovaikutuksia käyttäen kyselyitä. Ennen puhuttiin eri spesifikaatioista kuten HTML4 tai XHTML1, mutta koska verkkoteknologiat kehittyvät ja käyttäjien tarpeet muuttuvat, standardia kehitetään nopeaa tahtia, kun tulee palautetta nettikehittäjiltä, selainkehittäjiltä tai muilta tahoilta. Vaikka puhutaan HTML5:stä, se ei ole itsessään standardi vaan sillä viitataan HTML Living Standardiin, jota päivitetään aktiivisesti. (w3.org, ei pvm.; whatwg.org, 2022)

CSS, eli Cascading StyleSheets, mahdollistaa HTML:n ulkoasua lisäämällä sääntöjä eri osille HTML:ää, kuinka se piirretään. Sillä pystytään säätämään esimerkiksi värejä, muotoilua sekä tekstin fontteja. CSS on yksi ydinosa verkkokielistä, ja sitäkin päivitetään aktiivisesti. Ennen päivitykset julkaistiin uusina versioina, kuten CSS1, CSS2.1 tai uusin versio CSS3, mutta koska sen tarkoitus on laajentunut niin paljon, on uusien lisäyksien kehitys siirretty uusin moduuleihin. (w3.org, ei pvm.)

2.2 JavaScript ja TypeScript

JavaScript alkoi sivukielenä Javalle Netscape-selaimen aikaan vuonna 1995, mutta on noussut sieltä yhdeksi tärkeimmäksi ohjelmointikieleksi selaimille ja web-standardiksi, jota kehitetään nimellä ECMAScript. (Wirfs-Brock & Eich, 2020). Se on kevyt, tulkittu tai

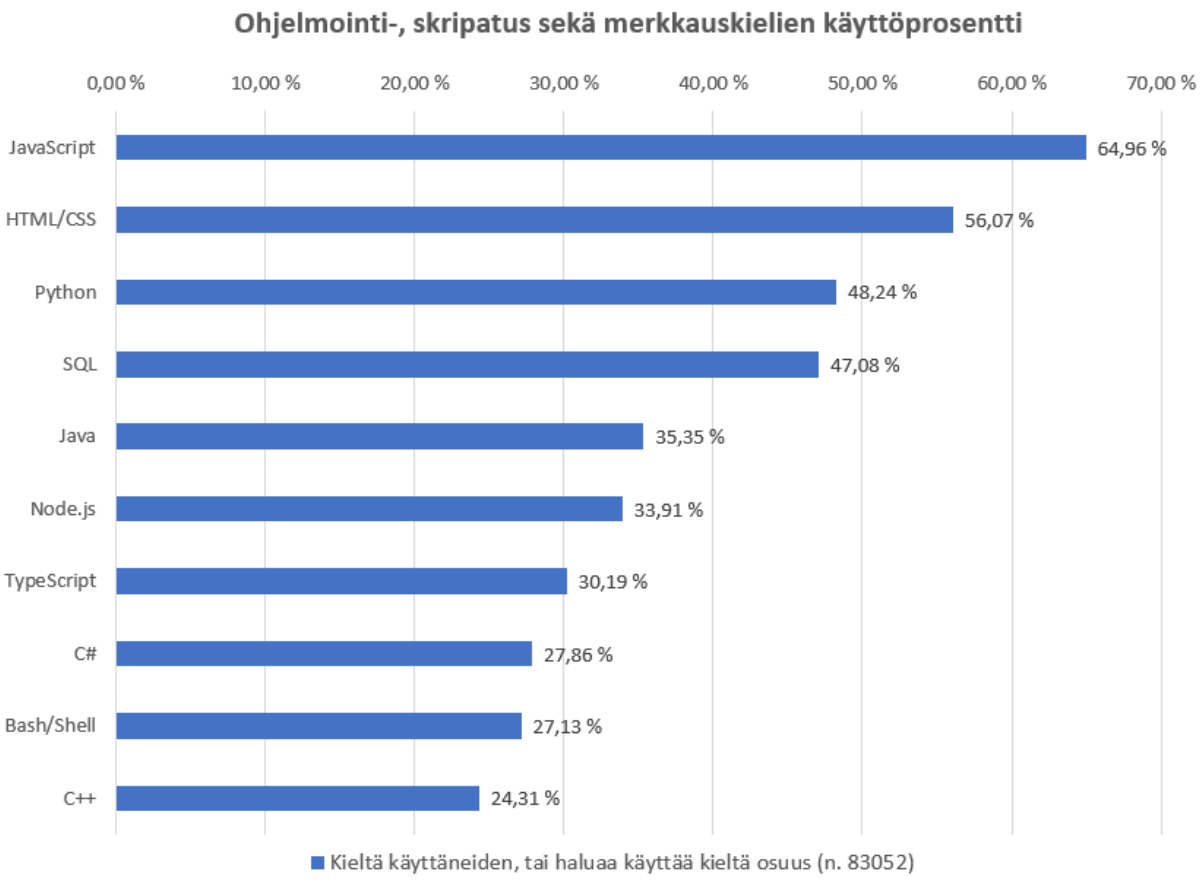
ajonaikana käännetty ohjelmointikieli. Se on prototyyppipohjainen, dynaaminen kieli, joka tukee oliokeskeistä sekä funktionaalista ohjelmointityyliä. Perussyntaksi kuten jos-lausunto ja silmukat ovat tarkoituksella samantapaisia kuten C++ ja Java, jotta JavaScript kielenä tuntuu tutulta jo aiempaa kehityskokemusta omaaville. (mozilla.org, ei pvm.-a)

Mutta JavaScript on heikosti tyypitetty kieli. Tällä tarkoitetaan, että muuttujien tyyppejä ei tarvitse erikseen määritellä, vaan ohjelman ajonaikana varmistetaan muuttujien tyypit toimiviksi. Tämä mahdollistaa helpommat tyyppimuutokset, kun vahvasti tyypitetyissä kielissä vaaditaan selkeä tyyppi muuttujille sekä muunnoksenselittäjä tyyppimuunnoksille esimerkiksi cast-operaattorilla. (Vaggalis, 2010). Vaikka tämä helpottaa kehityskokemusta, koska ei tarvitse niin keskittyä muuttujien tyyppeihin, tämä saattaa aiheuttaa vaikeasti löydettäviä ongelmia sekä vaikeuttaa ohjelmistoympäristön virheilmoituksia.

TypeScript on Microsoftin aloittama projekti lisätä JavaScriptiin vahva tyyppituki. Se on kehitetty JavaScriptin päälle, joten JavaScript koodi toimii normaalisti TypeScriptissä, ja on helppo alkaa lisäämään tyyppejä aikaisempaan koodiin vähitellen. Tämä tapahtuu koodin koontivaiheessa, jolloin tyyppilisäykset ja muut TypeScriptin lisäykset joko poistetaan tai käännetään JavaScriptiin. Tässä tulee etuna parempi integraatio ohjelmistoympäristölle, koska se pystyy tyyppien avulla tarjoamaan ehdotuksia sekä kertomaan virheistä koko projektissa. (typescriptlang.org, ei pvm.)

Nykyään JavaScript on yksi suosituimmista ohjelmointikielistä, kuten Kuva 1:stä näkee, perustuen StackOverflow:in viimeisimpään suosituimmat teknologiat -kyselyyn vuodelta 2021. Tämän lisäksi TypeScript on noussut nopeasti suosioon ohjelmoijien keskuudessa, ja on sijalla 6 samassa kyselyssä. (stackoverflow.com, 2021)

Kuva 1. StackOverflowin suosituimmat teknologiat kyselyn vastausten top 10 ohjelmointikielet. (stackoverflow.com, 2021)



2.3 Node.js ja NPM

JavaScriptin käyttö palvelinpuolella alkoi alun perin Netscapen tarjoamasta palvelinohjelmasta nimeltä Netscape LiveWire, joka mahdollisti dynaamisien sivujen luomisen JavaScriptin avulla. Tämä ei kuitenkaan saanut paljoa käyttäjiä. Kun JavaScript nousi vakavasti otettavaksi kieleksi niin sanotun Web 2.0 -aikaan ja sovellukset kuten Flickr ja Gmail syntyivät, selainten kehittäjät rupesivat parantamaan niiden omia JavaScript-moottoreita. Silloin alkoi myös Node.js -projekti, joka mahdollistaa JavaScriptin suorittamisen selaimen ulkopuolella. (nodejs.dev, ei pvm.).

Node.js on avoimen lähdekoodin alustariippumaton JavaScript-ajoympäristö, joka pyörii Google Chromen V8 JavaScript-moottorilla. Vaikka Node.js toimii yhdellä säikeellä, voidaan sen ominaisuuksia käyttää asynkronisesti,mikä mahdollistaa paremman työtahdin. Kun

ohjelman täytyy odottaa vastausta esimerkiksi tietokannasta tai verkkokyselystä, pystyy se tekemään muita töitä samaan aikaan. (nodejs.org, ei pvm.)

Node.js:n lisäksi samaan aikaan aloitettiin NPM (Node Package Manager), joka mahdollistaa JavaScript-pakettien jakamisen ja hallitsemisen. JavaScript-projekteissa käytetään tiedostoa ”package.json”, joka pitää sisällään tietoa projektista, projektin tarvittavat riippuvuudet sekä projektin rakennuskomennot. NPM-komentotyökalulla on mahdollista ladata JavaScript-paketteja projekteihin helposti, tai asentaa niitä yleisesti koneelle mahdollistaen paketin käytön mistä tahansa. (freecodecamp.org, 2020)

2.4 Saavutettavuus

Saavutettavuus verkkokehityksen yhteydessä tarkoittaa, että mahdollisimman moni ihminen pystyy käyttämään verkkosivua mahdollisimman helposti. Tässä huomioidaan kolmea eri osa-aluetta, eli tekninen toteutus, helppokäyttöisyys sekä sisällön selkeys ja ymmärrettävyys. Tekninen saavutettavuus tarkoittaa HTML-standardin sekä WCAG-ohjeistuksen, eli Web Content Accessibility Guidelines, noudattamista. Verkkosivu kehitetään helppokäyttöiseksi, eli se on helppo hahmottaa sekä navigoida. Sivun pääsisältö erottuu selkeästi sekä palvelun käyttö on vaivatonta. Viimeisenä on sivun ymmärrettävyys, eli teksti on selkeää ja helppolukuista sekä linkkitekstit ovat kuvaavia. Tähän kuuluu myös sivun moninaisuus, eli sisältö on tarjolla tekstin lisäksi esimerkiksi äänenä tai videoina. (saavutettavuusvaatimukset.fi/, ei pvm.-b)

Digipalvelulaki, joka perustuu Euroopan unionin saavutettavuusdirektiiviin, tuli suomessa voimaan 1.4.2019. Tämä ohjeistaa noudattamaan WCAG-ohjeistuksen A- ja AA-tason kriteerejä, mutta nämä eivät siltikään takaa verkkosivun saavutettavuutta tai helppokäyttöisyyttä kaikille käyttäjille. WCAG-ohjeistuksen AAA-tason kriteerit eivät ole lain vaatimia, mutta nämäkin ohjeistukset parantaisivat osan käyttäjien kokemusta. (saavutettavuusvaatimukset.fi/, ei pvm.-a)

Verkkosisällön saavutettavuusohjeet, eli WCAG, pitää sisällään yhteensä 78 kriteeriä, mutta näistä vain 49 ovat lain velvoittamissa A- ja AA-tasoissa. Nämä kriteerit ovat jaoteltu eri kategorioihin ja ne sisältävät tarkemmat ohjeet eri kriteereihin:

- Havaittava

  Sivun käyttöliittymä sekä informaatio tarvitsee esittää niin, että käyttäjällä ei olisi ongelmia havaita niitä. Tähän sisältyy ei-teksimuotoisen tiedon tärkeimmän sisällön olevan myös saatavilla ainakin tekstimuodossa, ja sivulla oleva tieto on selvää rakenteen ja piirteiden kuten värityksen mukaan.

- Hallittava

  Verkkosivun käyttöliittymän käyttöä ja navigointia on mahdollista hallita eri välineitä käyttäen kuten näppäimistöllä eikä sivun hallinnassa ei ole erilaisia isoja esteitä kuten aikarajoja.

- Ymmärrettävä

  Sivun tekstisisällön tulisi olla luettavaa ja sivun komponenttien tavanmukaisia.

- Yhteensopiva

  Sivun tulisi olla yhteensopiva eri nykyisien sekä tulevien asiakasohjelmien kanssa, sisältäen avustavat teknologiat.

(saavutettavuusvaatimukset.fi/, 2022)

2.5 Hakukoneoptimointi

Hakukoneoptimointi tarkoittaa prosessia, jolla saadaan verkkosivu näkymään mahdollisimman korkealla hakukoneiden tuloksissa. Hakukoneet käyttävät ohjelmia, jotka automaattisesti kulkevat internetissä keräämässä tietoja kaikilta mahdollisilta sivuilta, jotta niiden on mahdollista ehdottaa eri sivuja, kun käyttäjä etsii tietoa sitä käyttäen. Hakukoneoptimointi pitää sisällään monia eri aihealueita, kuten oikean HTML-syntaksin kirjoittamisen jolloin niistä on helppo hakea tärkeimmät tiedot, tiedon oikeanlaisuus, eli se on selkeästi kirjoitettu ja helppolukuista ja sivujen yleisyys, eli jos sivuilla on jo paljon käyttäjiä voi hakukone ajatella sen olevan parempi. (mozilla.org, ei pvm.-b)

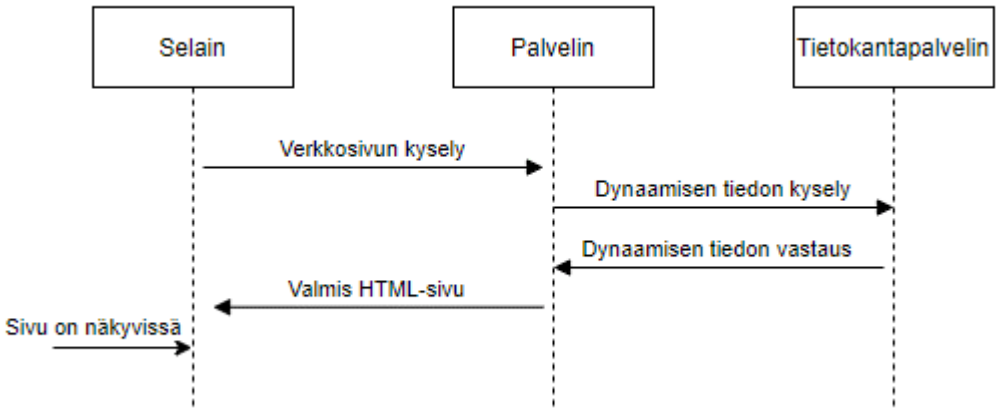
2.6 Julkaisutavat

Verkkosivut alkoivat staattisina tiedostoina yli 30 vuotta sitten, mutta ajan kanssa ne ovat kehittyneet dynaamisemmiksi esimerkiksi PHP:n avulla. PHP, eli *PHP: Hypertext Preprosessor* mahdollisti dynaamisen koodin kirjoittamisen suoraan HTML-näköisen tiedoston sisään, joka esikäsiteltiin jokaisen kyselyn yhteydessä näyttämään käyttäjälle tarvittavat asiat. Nykyään eri kehityskehykset ovat mahdollistaneet erilaisia sivujen renderöinti- sekä julkaisutapoja, jotka saattavat helpottaa kehitystä. Näissä eri tavoissa saattaa kuitenkin olla eri asioita, mitä tarvitsee huomioida, kuten hakukoneoptimointi, sivun suorituskyky sekä missä ja miten sivu tullaan jakamaan. (Luong, 2022)

2.6.1 SSR - Palvelinpuoleinen renderöinti

Palvelinpuolen renderöinnissä palvelin tekee isomman osan työstä. Kuten Kuva 2:ssa näkyy, selaimen kyselyn jälkeen palvelin hoitaa sivulle tarvittavan tiedon hakemisen sekä sen hahmottamisen. Tämä tarkoittaa käyttäjän näkevän sivun nopeammin, mutta sivunvaihdot tarkoittavat sivun uudelleen lataamista. Tämän etuina ovat nopeampi sivun ensilataus sekä parempi hakukoneoptimointi, mutta haittoina ovat palvelinkyselyt, sivunvaihdot ja interaktiiviset verkkosivut. (Danilec, 2020)

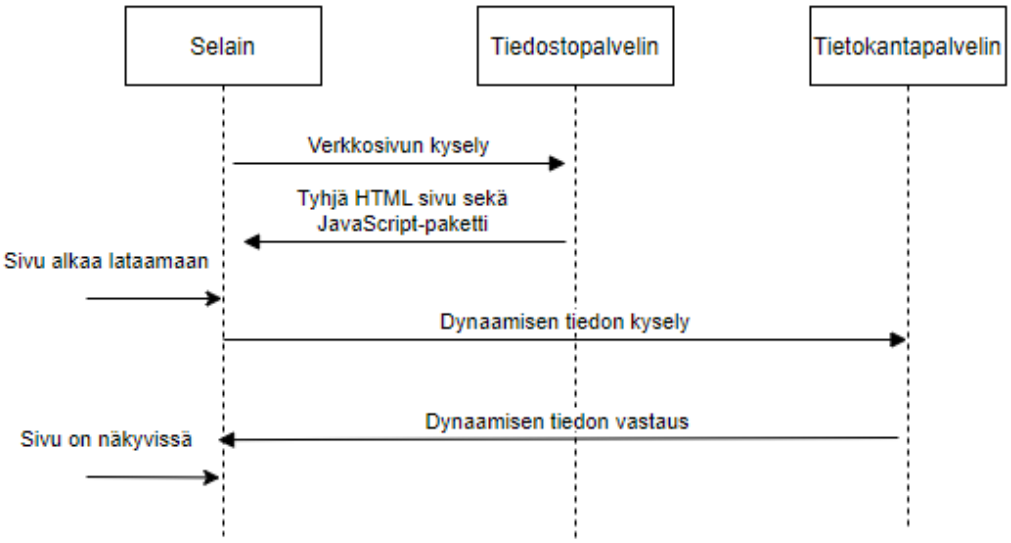
Kuva 2. Palvelinpuolen renderöinnin vaiheet. (Danilec, 2020)



2.6.2 CSR - Käyttäjäpuoleinen renderöinti

Käyttäjäpuolen renderöinti siirtää verkkosivun hahmottamisen käyttäjän selaimelle. Tämä on yleistynyt modernien kehityskehyksien avulla, kuten ReactJS tai Angular. Tässä tapauksessa, kun selain menee verkkosivulle, tulee palvelimelta tyhjä HTML-sivu sekä JavaScript-pakkaus, jonka ajon yhteydessä verkkosivu rakennetaan tyhjän HTML-sivun päälle. Kuva 3:sta näkee, että käyttäjä ei näe mitään ennen kuin selain ehtii ladata ja ajaa JavaScript-pakettia, ja tämänkin jälkeen saattaa lataus hidastua, jos sivu sisältää paljon dynaamisia tietoja, jotka täytyy hakea selaimen puolella. Vaikka sivun ensilataaminen on hieman hitaampaa, sen jälkeinen sivun käyttö on hyvin nopeaa, koska sivua ei tarvitse uudelleen ladata, vaan selain hallitsee sivunvaihdot ja sen tarvitsee uudelleen renderöidä yksittäisiä elementtejä sivulla. (Danilec, 2020)

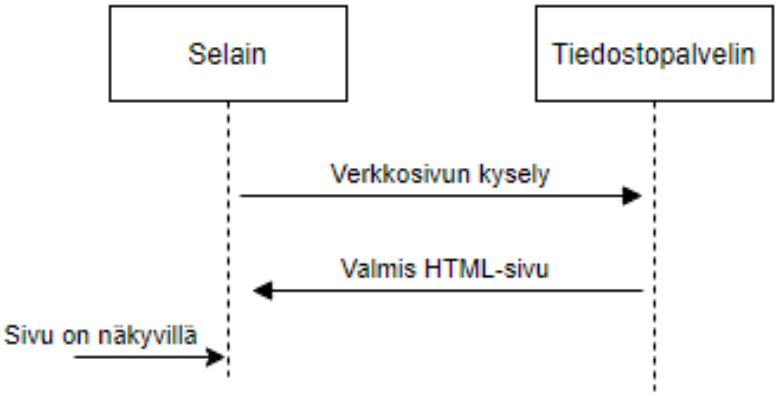
Kuva 3. Selainpuolen renderöinnin vaiheet. (Danilec, 2020)



2.6.3 SSG – Staattisen sivun tuottaminen

Jos tulevalle sivulle ei tule paljoa dynaamista tietoa tai sen nopeus on todella tärkeää, on sivut mahdollista esikäsitellä. Tässä tapauksessa selain saa valmiiksi renderöidyn HTML-sivun, joka näkyy selaimessa heti sen latauksen jälkeen. Tässä etuina ovat nopeampi sivunlataus, hakukoneoptimointi sekä vähemmän palvelinkyselyitä. (Danilec, 2020)

Kuva 4. Staattisen sivun vaiheet. (Danilec, 2020)



2.7 Julkaisutapojen vertailu

Nämä kaikki julkaisutavat ovat omalla tavallaan hyviä ja sopivat eri tarkoituksiin. Kun katsotaan Taulukko 1:stä sekä aikaisempia tietoja, voidaan ajatella selainpuoleisen renderöinnin sopivan esimerkiksi yhtiön sisäisiin sivuihin, missä ei ole hakukoneoptimoinnilla niin tarvetta, mutta sivujen tarvitsee olla nopeita. Toisaalta staattiset sivut ovat parhaita, vain jos sivun tiedot muuttuvat harvoin sen tiedonhakumahdollisuuksien puuttumisen takia, esimerkiksi yhtiön yleisille sivuille. Lopuksi palvelinpuolen renderöinti sopii hyvin dynaamisille sivuille, joilla on tarvetta olla hyvä hakukoneoptimointi, eikä hieman hitaampi sivu haittaa.

Taulukko 1. Julkaisutapojen vertailu (Luong, 2022)

	SSR	SSG	CSR
Suorituskyky	Hitaampi	Nopea	Nopea
Hakukoneoptimointi	Hyvä	Hyvä	Huonompi

Esimerkki kehityskehys	Django, Rails	Hugo	ReactJS, Vue
Jakelu	Palvelin	CDN	CDN

3 ReactJS

ReactJS:n kehitys alkoi Facebookilla sivuprojektina. XHP, joka on Facebookin versio PHP-ohjelmointikielestä, tarkoitus oli minimoida XSS-hyökkäykset. XSS-hyökkäykset tarkoittaa haitallisen koodin piilottamista joko jaettaviin linkkeihin verkkosivulle, tai sivulle tallennettaviin tietoihin kuten kommentteihin, joka suoriutuu sivun latauksen yhteydessä ja mahdollistaa hyökkäjää muokkaamaan verkkosivua tai varastamaan sen sisältäviä tietoja. Tämä ei kuitenkaan auttanut skaalautuvuusongelmassa, koska sovellukset tarvitsivat monia kyselyjä palvelimella. Joten yhtenä ratkaisuna oli siirtää tämä prosessi selaimen puolelle käyttäen JavaScriptiä, ja siitä alkoi ReactJ:n kehitys. (Dawson, 2014)

ReactJS on deklaratiivinen, komponenttipohjainen, selainpuolen käyttöliittymäkirjasto. Se helpottaa interaktiivisien verkkosivujen käyttöliittymien kirjoittamisen, koska se hoitaa taustalla sivujen päivittämisen, kun sivun tiedot muuttuvat päivittämällä vain tarvittavat osat verkkosivusta. Tätä helpottaa Reactin komponenttipohjaisuus, joka mahdollistaa uudelleenkäytettävien komponenttien kirjoittamisen, joita voidaan käyttää monessa osassa sivun kokonaisuutta. Tämän lisäksi komponentit voivat sisältää omaa tietoa tai jakaa tietoa muille komponenteille. Koska React on täysin selainpuolen käyttöliittymä, voidaan sen lisäksi käyttää mitä tahansa palvelinpuolen teknologiaa, jolla voi jakaa tietoa eri palveluille rajapintojen avulla kuten REST. (reactjs.org, ei pvm.-e) Mikä tekee tästä kirjaston eikä kehityskehyksen on sen suunnittelumalli tai sen puute. ReactJS toteuttaa pelkästään näkymän, jolloin kehittäjä voi itse valita, miten sovelluksen suunnittelee siitä eteenpäin. (Chiarelli, 2018)

3.1 JSX

Koska JavaScript ei sisällä hyvää tapaa esittää HTML-koodia, siihen on kehitetty päälle syntaksilisäosa nimeltä JSX. Kuten Ohjelmakoodi 1:sta näkee, ReactJS:ssa voidaan kirjoittaa sivun logiikka sekä ulkoasumäärittely samassa tiedostossa. (reactjs.org, ei pvm.-c) Taustalla JSX käännetään normaaliksi JavaScriptiksi käyttäen esimerkiksi BabelJS-kääntäjää. JSX pitää sisällään myös sääntöjä, kuten kaikki käyttäjäluodut komponenttien nimet tulee aloittaa isolla kirjaimella. (reactjs.org, ei pvm.-d)

Ohjelmakoodi 1.Esimerkkikoodi JSX, joka mahdollistaa HTML-koodin kirjoittamisen suoraan JavaScriptissä. (reactjs.org, ei pvm.-c)

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

3.2 Hooks

React hooks julkaistiin React 16.8 versiossa stabiiliin linjaan. Nämä mahdollistavat tilan sekä muiden Reactin ominaisuuksien käyttämisen ilman luokkien kirjoittamista, mahdollistaen funktionaalisien komponenttien kirjoittamisen. Lisäys tulee aiemman tavan viereen, joka oli luokkakomponenttien kirjoittaminen, jolloin kaikki aikaisemmin tehty silti toimii eikä sitä ole suunniteltu vanhentumaan. Tämä mahdollistaa kummankin tavan käyttämisen samassa projektissa tai osan komponenttien uudelleenkirjoittamisen tälle tavalle ilman, että mikään rikkoutuisi. Tätä lähdettiin kehittämään, koska aiemmat luokkakomponentit saattoivat olla monimutkaisia sekä käyttäjille oppia, että eri esikäsittelyvaiheille mitä React tekee tai haluaisi tehdä tulevaisuudessa. Tämän lisäksi ne mahdollistavat tilallisen logiikan jakamisen monelle komponenteille. (reactjs.org, ei pvm.-b)

Ohjelmakoodi 2. useState-hookin käyttö komponentissa, joka mahdollistaa tilan tallentamisen sekä käyttämisen.

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

function Example() {
  // Luodaan uusi muuttuja “count”, jota pystyy muokkaamaan “setCount”
  // funktiolla, ja annetaan sille alkuarvoksi 0
  const [count, setCount] = useState(0);

  return (
    <div>
      // Tässä käytetään ylempänä luotua muuttujaa, sekä nappia painamalla
      // nostetaan “count” muuttujan arvoa 1:llä.
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
```

```
        Click me
      </button>
    </div>
  );
}
```

3.3 Create-React-App

Vaikka Reactia pystyy kehittämään ilman mitään erikoisia työkaluja suoraan HTML-koodiin, voi se olla aika vaivalloista. Tämän takia Facebook lähti kehittämään työkalua parantamaan kehityskokemusta. Create-React-App joka mahdollistaa monen tiedoston ja komponentin organisoinnin kansioihin, automaattisen sivunpäivittämisen muutoksien jälkeen sekä parempi lopputuloksen kääntäminen, joka on optimoitu tuotantoon. Tämän avulla pystyy aloittamaan projektin käyttäen eri pohjia, kuten TypeScript-projektin tai projektin, joka sisältää muita kirjastoja jo valmiiksi kuten Redux, joka helpottaa tilan hallinnan koko sovelluksessa. Create-React-App piilottaa monet konfiguraatiot taustalle, mutta mahdollistaa erottautua työkalusta, jolloin kaikki taustalla tapahtuvat konfiguraatiot tulevat näkyville muokattavaksi. Tämä on kuitenkin yksisuuntainen operaatio, joten erottautumisen jälkeen ei pysty palaamaan aiempaan tapaan. (reactjs.org, ei pvm.-a)

4 Next.js

Next.js on kehityskehys, joka mahdollistaa Reactin käyttämisen palvelinpuolen renderöinnissä sekä staattisien sivujen generoinnissa. Se yrittää korjata ongelmia, joihin ReactJS-kehittäjät ovat törmänneet sen käytössä, kuten sivujen esikäsittelyn palvelinpuolella paremman suorituskyvyn tai SEO:n takia. Tämä tapahtuu rakentamalla Reactin päälle isompi kokonaisuus sekä tekemällä siihen tiettyjä valintoja kuten sivujen reititys sekä palvelinpuolen kielen valinnat. Näiden lisäksi sen kehittäjät yrittävät kaiken aikaa parantaa kehityskokemusta sekä lopputulosta kuten viimeisen päivityksen lisäämä Next.js-kääntäjä, joka on rakennettu SWC-alustan päälle, joka helpottaa nopeiden kehitystyökalujen kehittämisen, on monikertaisesti nopeampi kuin aiemmin käytössä ollut työkalu Babel. (nextjs.org, ei pvm.-k)

Next.js automaattisesti tunnistaa, mitkä sivut ovat staattisia eikä tarvitse palvelinpuolen renderöintiä, perustuen tiedonhakutapoihin. Jos sivulla käytetään getServerSideProps-funktiota, kuten Ohjelmakoodi 3:ssa, tarvitsee sivun käyttää palvelinpuolen renderöintiä, mutta muuten sivut voidaan esikäsitellä koontivaiheessa jo staattisiksi HTML-sivuiksi, jotka ovat nopeampia jakaa sekä helpompia tallentaa välimuistiin. (nextjs.org, ei pvm.-a) Jos halutaan staattiset sivut niiden etujen vuoksi, mutta sivuja on liian monta esikäsitellä kerralla tai silti sivuilla tarvitsee olla jotain dynaamista dataa, on mahdollista luoda ja päivittää staattisia sivuja ajovaiheessa lisäämällä sääntö sivujen päivittämiseen, jonka jälkeen sivu tarvitsee uudelleen käsitellä. (nextjs.org, ei pvm.-j) Jos ei palvelinpuolen renderöinnille kuitenkaan ole tarvetta, on mahdollista rakentaa sivuista täysin HTML-pohjainen versio, joka ei tarvitse Node.js-palvelinta ollenkaan. Lisäämällä yksi komento koontivaiheeseen rakentuu HTML-versio verkkosivuista ”/out” kansioon. (nextjs.org, ei pvm.-b)

Ohjelmakoodi 3. Datan hakeminen palvelimen puolella käyttäen getServerSideProps() funktiota. (nextjs.org, ei pvm.-i)

```
function Page({ data }) {
  // Render data...
}

// This gets called on every request
export async function getServerSideProps() {
```

```
  // Fetch data from external API
  const res = await fetch(`https://.../data`)
  const data = await res.json()

  // Pass data to the page via props
  return { props: { data } }
}

export default Page
```

4.1 Reititys

React ei pidä sisällä mitään ylemmän tason helpotusta, vaan kehittäjän tarvitsee joko kehittää oma tapa reititykselle tai käyttää kirjastoa kuten React-Router käsittelemään sivujenvaihdot ja auttamaan komponenttien reititystä. (reactrouter.com, ei pvm.). Next.js kuitenkin on kehittänyt oman reitittimen, joka perustuu tiedostoihin pages-kansion sisällä. Esimerkiksi tekemällä ”about.js” tiedosto, tulee automaattisesti polku ”/about” reititettyä sinne. Sisäkkäiset reitit toimivat kansioiden avulla, eli tiedosto ”/pages/team/about.js” reitittyy ”/team/about” polkuun. Staattisten polkujen lisäksi tiedostoilla voidaan myös luoda dynaamisia polkuja, sisältäen muuttujan nimi hakasulkeisiin tiedostonimessä, kuten ”/pages/blog/[slug].js” mahdollistaa blogipostauksen hakemisen tämän ”[slug]”-muuttujan kautta. (nextjs.org, ei pvm.-l)

Reitityksen avuksi Next.js tarjoaa myös komponentin helpottaakseen selainreititystä. Kuten Ohjelmakoodi 4:sta näkee, Link-komponentin avulla voidaan reitittää selaimen puolelta eri sivulle. Linkkien reitittäminen toimii samalla tavalla kuin HTML a-tagilla menisi, eli tuleva osoite on href-attribuutissa. Jos tämä komponentti ei kuitenkaan riitä, tarvitaan jotain tietoa reitittimeltä Reactin puolella tai jotain tarvitsee muokata käsin, on mahdollista päästä käsiksi käyttäen ”useRouter” React hookia. Tämä hook palauttaa reittiobjektin, josta saa esimerkiksi tämänhetkisen reitin, mahdolliset kyselyn muuttujat sekä muuttujat, joilla on mahdollista muokata selaimen URL-reittiä. (nextjs.org, ei pvm.-l)

Ohjelmakoodi 4. Esimerkkikoodi Next.js-kehyksen lisäämästä Link-komponentti, joka mahdollistaa sivujenvaihdot

```
import Link from 'next/link'

function Home() {
  return (
    <ul>
      <li>
        <Link href="/">
          <a>Home</a>
        </Link>
      </li>
      <li>
        <Link href="/about">
          <a>About Us</a>
        </Link>
      </li>
    </ul>
  )
}

export default Home
```

Koska Next.js sisältää palvelinpuolen renderöinnin on sillä myös mahdollista tehdä API-polkuja, jolla voidaan turvallisesti hakea ja palauttaa tietoa palvelinpuolella. Tätä kannattaa käyttää, jos tarvitsee hakea tietoa tietokannasta tai ulkoisesta lähteestä tai jossa tarvitaan tunnuksia, jota ei voi antaa selaimenpuolelle turvallisesti. Näiden luonti tapahtuu samalla tavalla kuin selaimenpuoleinen reititys. API-funktiot luodaan kansioon ”/pages/api/”, joka automaattisesti reitittyy api-polkuun. Tämän avulla voidaan luoda kokonainen sovellus käyttäen Next.js-työkaluja, tai käyttää sitä vain pariin tarvittavaan lähteeseen ja käyttää jotain muuta palvelinpuolen API-ohjelmointikehystä (nextjs.org, ei pvm.-c). Staattisia tiedostoja, jotka eivät muutu ja niitä tarvitaan paljon, pystyy jakamaan laittamalla ne ”public” nimiseen kansioon. Esimerkkejä staattisista tiedostoista ovat ”robots.txt” ja ”favicon.ico”, joita tarvitaan verkkosivuilla. Nämä tulevat jaetuksi verkkosivun juureen, eli ”/robots.txt”. Tässä pitää huomata, ettei mikään tiedosto ”/public”-kansiossa mene sivureitityksen päälle ”/pages”-kansiossa, muuten tulee virheitä. (nextjs.org, ei pvm.-g)

4.2 Optimoinnit

Next.js sisältää useamman optimoinnin helpottamaan sekä palvelinpuolella että selainpuolella tapahtuvia asioita, jotka parantavat käyttäjäkokemusta monella tavalla. Se sisältää Kuva-komponentin, joka rakentuu html img-tagin päälle ja automaattisesti yrittää optimoida kuvat sopimaan päätelaitetta jo palvelinpuolella. Tämän lisäksi automaattisia selaimenpuolen optimointeja kuten kuvan hakeminen vasta kuin se näkyisi näytöllä ja yrittää estää sivun pohjan uudelleensekoittamista kuvien latautumisen takia. (nextjs.org, ei pvm.-f)

Next.js automaattisesti optimoi sivun fontit, lataamalla ne koontihetkellä ja sijoittamalla sen sisällön suoraan HTML-koodiin. Tämä vähentää selaimen tarvitsemia kyselyitä palvelimelta ja parantaa selaimen latausta ja siellä sivun prosessointia. Tämä tapahtuu automaattisesti kaikille fonteille, jotka lisätään HTML:n head-osioon koontivaiheessa, mutta vain tällä hetkellä tuetuille palveluille eli Google Fonts sekä Adobe Fonts. (nextjs.org, ei pvm.-d)

Näiden lisäksi Next.js voi optimoida verkkosivun lataamat kolmannen osapuolen skriptit kuten analytiikka-, mainos- tai asiakastuki-skriptit. Koska nämä saattavat olla raskaita ladata ja hidastaa sivun lataamista tai kehittäjillä voi olla vaikeuksia sijoittaa skriptit sivuille varmistaen latausjärjestyksen, voidaan käyttää Script-komponenttia ja valita skriptin lataamisstrategia. Kuten Ohjelmakoodi 5 näkyy, skripteille voidaan valita joko ennen sivun interaktiivisuutta, heti interaktiivisuuden jälkeen tai laiskalataus, joka tapahtuu sivun latauksen jälkeen, kun selain on toimeton. (nextjs.org, ei pvm.-e)

Ohjelmakoodi 5. Esimerkkikoodi Script-komponentille, joka on ladattava skripti, joka latautuu heti kun mahdollista, ennen kuin sivu on interaktiivinen.

```
<Script

src="https://cdn.jsdelivr.net/npm/cookieconsent@3/build/cookieconsent.min.j
s"
  strategy="beforeInteractive"
/>
```

4.3 Tapaustutkimus: Hulu

Hulu, suoratoistopalvelu, joka tarjoaa TV-ohjelmia sekä elokuvia katseltavaksi, alkoi käyttämään Next.js:ää heti sen julkaisun jälkeen pienessä roolissa ja myöhemmin alkoi uudelleenkirjoittamaan käytössä olleita palveluita käyttämään Next.js:ää. Isoin hyöty palveluiden muutossa oli sivujen reititys, joka mahdollisti yksittäisien sivujen uudelleenkirjoittamisen, ja silti pitämään vanhan palvelun taustalla pyörimässä. Tämän lisäksi jokainen uusi Next.js-versio, joka toi uusia ominaisuuksia mahdollisti vanhan koodin poiston, koska se on nyt integroitu suoraan ohjelmistokehykseen. Se tiimi, joka aloitti Nex.js:än käytön Hulussa kertoi, että Next.js auttoi heidän ohjelmoijiansa välttämään yleisiä ongelmia, joita esiintyi käyttöliittymäohjelmoinnissa, koska Next.js säätää tiettyjä asioita kuten datan hakemisen. (nextjs.org, ei pvm.-h)

Iso vaatimus Hululle oli palvelinpuolen renderöinti. Koska tietoa tulee monesta lähteestä sekä heillä on iso tarve hakukoneoptimoinnille, palvelinpuoleinen renderöinti auttaa kummassakin ongelmassa. Next.js:än etuna oli helppo siirtymä palvelinpuolen renderöintiin, koska se oli käytännössä vain datanhakemislogiikan siirtäminen. He myös huomasivat virheiden olevan helpompi seurata verrattuna muihin valintoihin mitä oli testattu. (nextjs.org, ei pvm.-h)

Koska Hulu otti Next.js:än käyttöön aikaisin sen kehityksessä, jouduttiin paljon koodia kirjottamaan, jotta asiat saatiin toimimaan. Vuoden jälkeen projektin alusta Zack Tanner, joka oli vanhempi ohjelmistoinsinööri Hululla, kehui Next.js-kehityskehystä vähenneistä bugeista sekä paremmasta produktiivisuudesta. (nextjs.org, ei pvm.-h)

5 Verkkosivun kehittäminen Next.js:n avulla

Jotta saadaan parempi kuva, miten verkkosivujen kehitys toimii käyttäen Next.js-kehityskehystä, tehdään sillä verkkosivuprojekti. Projektina luodaan blogi, jolla saadaan parempi kuva sen hyödyistä ja eri tavoista jakaa, hyödyntäen eri NPM-paketteja helpottamaan työn tekemistä. Tämä olettaa, että isot riippuvuudet on jo asennettu, kuten Node.js sekä koodineditointiohjelma kuten Visual Studio Code.

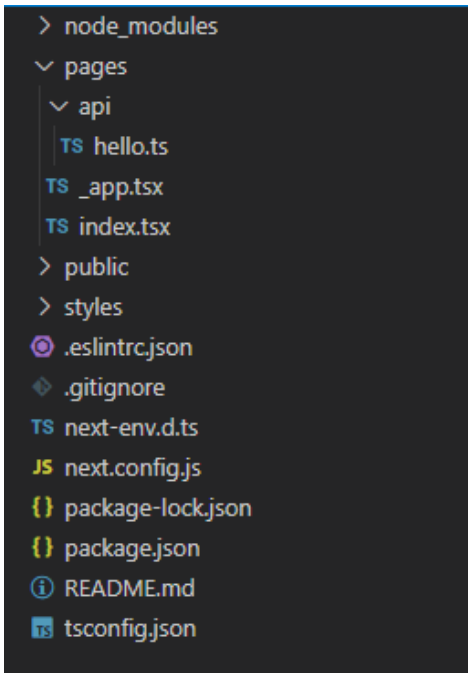
5.1 Projektin suunnittelu ja aloitus

Projektin aloitus Next.js:ää käyttäen on yhtä helppoa kuin React-sivun aloittaminen käyttäen Create-React-Appia. Next.js-sovelluksille on kehitetty oma pakkaus nimeltä create-next-app kehittämisen aloittamiseen, joka rakentaa sovelluspohjan valmiiksi ja pääsee heti vauhtiin. Ajamalla Komento 1:n, npm lataa tarvittavat riippuvuudet ja tekee tarvittavat kansiot sekä tiedostot Kuva 5:n mukaan. Tämän tekohetkellä uusin versio Next.js:stä ja mitä tässä käytettiin, on 12.1, sekä käytetty Node.js versio on v16.13.2.

Komento 1. npx-komento, joka käyttää create-next-app npm-pakettia luodakseen uuden projektin TypeScriptiä käyttäen.

```
npx create-next-app <projektin nimi> --ts
```

Kuva 5. create-next-app:in tuottama sovelluspohja.



Tämän lisäksi kehityksessä tarvitaan muita paketteja. Tässä käytetään Semantic-UI-React komponenttikirjastoa, joka helpottaa Semantic-UI css-kehyksen käyttöä lisäämällä valmiita React-komponentteja valmiiksi muotoiltuna.

5.2 Sivujen kehitys

Koska sivulla tullaan testaamaan eri tapoja, miten Next.js mahdollistaa sivujen jakamisen, tarvitsee sama sivu tehdä eri tavoilla. Yhteistä näillä sivuilla tulee olemaan blogien tiedot, jotka ovat tiedostossa. Tämän lisäksi tarvitsee tehdä API-polku, jolla selainpuolen renderöinnissä tarvittavat tiedot saadaan. Näitä polkuja tulee kaksi, yksi palauttamaan kaikki kirjoitukset sekä toinen, kuten Ohjelmakoodi 6:ssa näkyy, palauttamaan yksittäinen blogipostaus.

Ohjelmakoodi 6 dynaamisen tiedon hakeminen API-polulla

```
export default function handler(
    req: NextApiRequest,
    res: NextApiResponse) {
    const {slug} = req.query
    const blog = blogs.find(blog => blog.slug === slug)
```

```
    res.status(200).json(blog)
}
```

Aloittamalla palvelinpuolen renderöinnistä, sille täytyy tehdä omat polut sekä kaikkien sivujen listaukseen, sekä dynaamiseen polkuun, jolla yksittäisiä sivuja voidaan lukea. Ohjelmakoodi 7:sta näkee, kuinka tieto haetaan ja palautetaan Reactin puolelle. Koska se ajetaan palvelimen puolella, voidaan sillä palauttaa tietoa, joka on tallennettu palvelimen puolelle. Tämä sama koodi toimii myös staattisen sivun generoinnissa, vaihtaen funktion nimeksi ”getStaticProps”.

Ohjelmakoodi 7 Palvelinpuolen renderöinnissä käytetty tiedonhakufunktio

```
import { blogs } from '../../data'

// Palvelinpuolen Renderöinti
export const getServerSideProps = async () => {

    // Palautetaan sivulle haetut tiedot
    return {
        props: {
            blogs
        }
    }
}
```

Koska selainpuolella ei voida hakea suoraan tietoa palvelimelta, täytyy siihen käyttää API-polkuja. Kuten Ohjelmakoodi 8:sta voidaan katsoa, useEffect-hookilla voidaan tehdä asioita, joita tarvitsee tehdä ennen sen käyttöä. Esimerkiksi sillä voidaan hakea tietoa ulkoisesta lähteestä, jonka jälkeen sen palauttamat tiedot tallennetaan komponentin tilaan useState-hookilla. Koska React seuraa komponentin tilaa, voidaan sillä näyttää odotustila ennenkuin useEffect-hookin tiedot saapuvat, ja React automaattisesti uudelleenpiirtää komponentin tulleilla tiedoilla.

Ohjelmakoodi 8 Selainpuolen tiedonhaku React-hookeilla

```
// Luodaan uusi muuttuja ja sen funktio, ja aloitetaan se tyhjällä listalla
const [blogs, setBlogs] = useState([])
```

```
// Käytetään useEffect-hookkia hakemaan tietoa, ja lopuksi laitetaan
// saadut tiedot yllä luomaan muuttujaan.
useEffect(() => {
    fetch('http://localhost:3000/api/blogs')
        .then(res => { return res.json() })
        .then(data => { setBlogs(data) })
}, [])
```

Sivut, joilla näytetään kokonaiset blogikirjoitukset toimivat melkein samalla tavalla, mutta koska nämä ovat dynaamisia sivuja, tulee polun tiedoston nimeksi ”[slug].tsx”.

Palvelinpuolen renderöinnissä sekä staattisen sivun generoinnissa tieto haetaan ennen käyttäjälle lähettämistä. Tämä mahdollistaa nopean tiedonhaun kaikissa tilanteissa, vaikka käyttäjällä olisi hitaampi yhteys. Sen takia varsinkin selainpuoleisessa tiedonhaussa kannattaa lähettää vain tarvittava tieto takaisin käyttäjälle. Ohjelmakoodi 9:ssa näkyy, kuinka polun dynaaminen muuttuja haetaan useRouter-hookista, jota käytetään samanlaisessa hakufunktiossa. Sen lisäksi useState-hookissa näkyy, että se olettaa Blog-tyyppisen objektin, mutta ”Partial” tarkoittaa, että se saattaa olla vajaa.

Ohjelmakoodi 9 blogin tiedon hakeminen dynaamisessa polussa.

```
const [blog, setBlog] = useState<Partial<Blog>>({})
const {slug} = useRouter().query
useEffect(() => {
    fetch(`http://localhost:3000/api/blogs/${slug}`)
        .then(res => { return res.json() })
        .then(data => { setBlog(data) })
}, [slug])
```

Staattisilla sivuilla, joka sisältää dynaamisia polkuja, tarvitsee käyttää getStaticPaths-funktiota, jotta tiedetään mitkä sivut tarvitsevat esikäsittelyä. Ohjelmakoodi 10:sta näkee, kuinka kaikkien blogien tiedot haetaan, niistä otetaan pelkästään sen uniikki tieto, jonka jälkeen ne palautetaan. Tämä toimii yhteydessä getStaticProps-funktion kanssa, joka käyttää tätä uniikkia tietoa hakemaan sivujen tarvittavat tiedot renderöintiin.

Ohjelmakoodi 10 Staattisien blogisivujen hakeminen

```
import {blogs} from '../../data'
```

```
export const getStaticPaths = async () => {
    // Kartoitetaan blogipostauksien sivuosoitteet
    const paths = blogs.map((blog) => ({
        params: {
            slug: blog.slug
        }
    }))

    // Palautetaan osoitteet
    return {
        paths,
        fallback: false
    }
}
```

Sivujen lisäksi tarvitsee tehdä tiettyjä säätöjä, jotka sisältyvät jokaiselle sivulle. ”pages/_document.tsx” tiedostossa lisätään HTML-tagille kieliattribuutti sekä meta-tagia käyttäen kuvaus verkkosivustosta. Tämän lisäksi ”pages/_app.tsx” tiedostossa lisätään verkkosivuille yhtenäinen ulkoasu. Koska kaikilla sivuilla tulee olemaan samoja komponentteja, kuten navigointipalkki sivun ylhäällä, voidaan ne Ohjelmakoodi 11:n mukaan lisätä tänne, ja itse sivun komponentti, tulee Component-komponentin tilalle.

Ohjelmakoodi 11 Yhteinen pohja kaikille sivuille

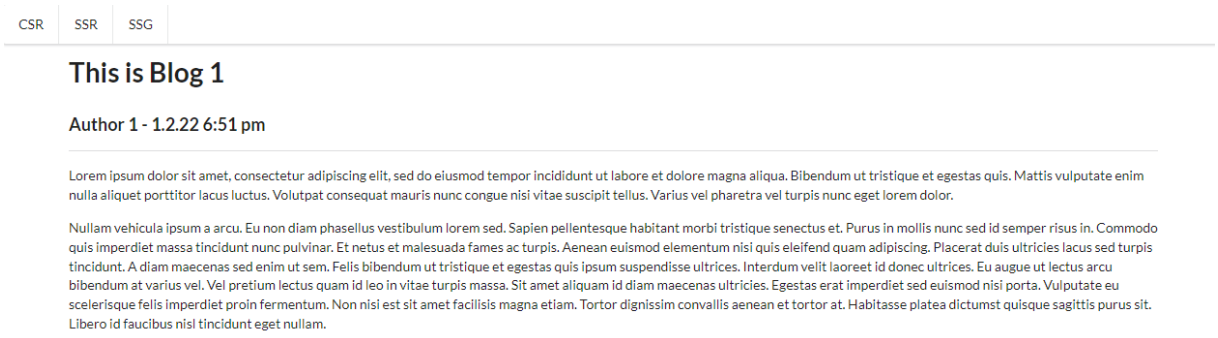
```
function MyApp({ Component, pageProps }: AppProps) {
  return (
    <>
      <Head>
        <title>OpBlogs</title>
        <meta media="viewport" content="" />
      </Head>

      <Navbar />
      <Container>
        <Component {...pageProps} />
      </Container>
    </>
  )
}
```

6 Projektin tutkimus

Sivun toiminnallisuus toteutettiin kolmeen kertaan, eri tiedonhakutavoilla, joita testaamalla saadaan parempi kuva niiden vahvuuksista ja heikkouksista. Eri tavat sisältävät sekä yleisen sivun, missä näkyy listalla kaikki blogikirjoitelmat, sekä dynaamiset sivut eri blogikirjoituksille. Blogisivujen tiedot tulevat tiedostosta ja niiden tekstit ovat joko kuvaavia nimi tai satunnaisesti luotuja tekstinpätkiä Lorem Ipsumia käyttäen. Kuten Kuva 6:sta näkyy, erillisillä blogikirjoitelmasivuilla tulee näkyviin yksittäisen kirjoitelman otsikko, kirjoittaja, julkaisuaika sekä itse kirjoitelma.

Kuva 6 Projektin esimerkkisivu



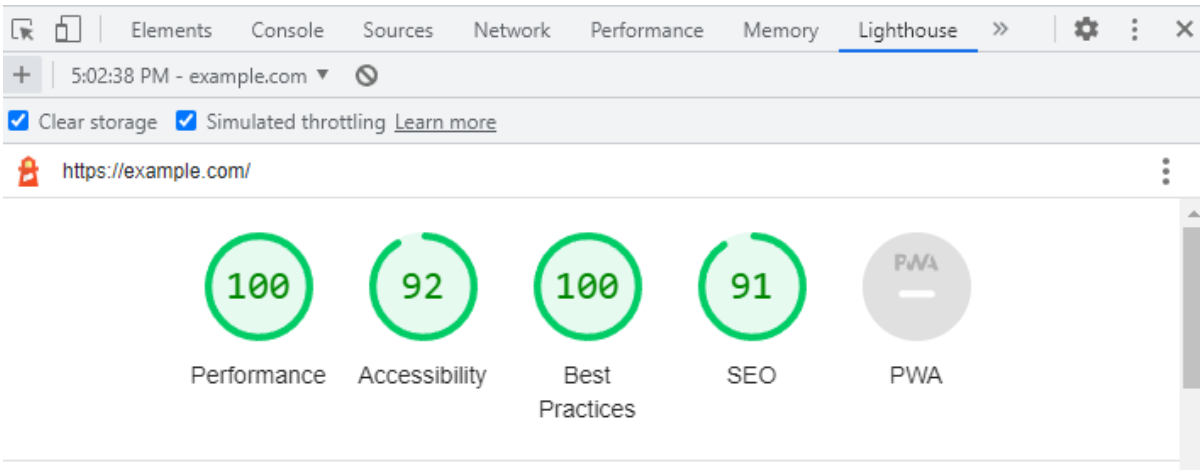
Lopuksi kun sivut tarkastettiin, ettei siellä ollut huomattavia virheitä ja käsipelin testaten, että kaikki toimii kuten pitäisikin, on aika valmistella sivut testattavaksi. Sivut rakennettiin tuotantokäyttöön, jolloin niitä on mahdollista testata paremmin.

6.1 Vertailu

Jotta saadaan parempi kuva näiden eroista, tarvitsee niitä testata ja vertailla niiden tuloksia. Yksi hyvä työkalu siihen on Googlen kehittämä Lighthouse, joka on avoimen lähdekoodin automatisoitu työkalu verkkosivujen laadun tarkasteluun. Sillä on mahdollista saada tietoa esimerkiksi verkkosivun suorituskyvystä, saavutettavuudesta sekä hakukoneoptimoinnista, kuten Kuva 7:sta näkyy. Sitä on mahdollista käyttää esimerkiksi Chrome-selaimen

kehitystyökaluista tai verkkosivulta ”https://pagespeed.web.dev/”, jos sivu on jo julkaistu. (developers.google.com, ei pvm.)

Kuva 7 Lighthouse-tulokset example.com sivustosta.



Näistä suorituskyky, saavutettavuus, parhaat käytännöt sekä SEO ovat samoja eri tekniikoissa mitä projektissa, ja niiden tulokset eivät muutu, joten tässä keskitytään verkkosivujen suorituskykyyn. Suorituskykytestissä seurataan kuutta eri kohtaa, mistä eri painoarvojen mukaan saa aika hyvän kuvan verkkosivun suorituskyvystä. Seuraavassa listassa selitetään nämä eri osa-alueet mitä Lighthouse tutkii, ja josta sivun arvosana tulee. (web.dev, ei pvm.-d)

First Contentful Paint (FCP), eli Ensimmäinen Sisällön Piirros, mittaa kuinka kauan sivulla kestää, ennen kuin ensimmäinen elementti on piirretty verkkosivun DOM:iin. Tämä sisältää kuvat, Canvas-elementit, johon on piirretty jotain sekä SVG:t, mutta ei sisällä iframe-elementtejä. Tämä koskee 10 % kokonaistuloksesta. (web.dev, ei pvm.-b)

- Speed Index (SI), eli Nopeusindeksi mittaa kuinka nopeasti verkkosivun lataaminen ja esittäminen kestää. Tämä tapahtuu ottamalla kuvia verkkosivun latausvaiheessa, ja laskemalla verkkosivun visuaalinen edistyminen. Tämä koskee 10 % kokonaistuloksesta. (web.dev, ei pvm.-e)

- Largest Contentful Paint (LCP), eli Isoimman Sisällön piirros, mittaa kuinka kauan kestää sivun isoimman elementin piirtämiseen. Tämä sisältää isoimman kuvan tai

tekstielementin latautumisen, joka näkyisi sivun latautumisen yhteydessä. Tämä koskee 25 % kokonaistuloksesta. (web.dev, ei pvm.-c)

- Time to Interactive (TTI), eli Aika Interaktiivisuuteen, mittaa kuinka kauan kestää, ennen kuin käyttäjä pystyy käyttämään verkkosivua. Tämä tarkoittaa, että verkkosivu on piirtänyt jo jotain sivulle, mikä mitataan FCP:ssä. Tämän lisäksi verkkosivun näkyvien elementtien tapahtumienkäsittelijät ovat rekisteröity sekä verkkosivu vastaa käyttäjän kosketuksiin 50 millisekunnin sisällä. Tämä koskee 10 % kokonaistuloksesta. (web.dev, ei pvm.-f)

- Total Blocking Time (TBT), eli Kokonainen Lukittu Aika, mittaa kuinka kauan kestää ennen kuin verkkosivu vastaa käyttäjän komentoihin, esimerkiksi hiirellä tai näppäimistöllä. Tämä siis mittaa aikaa tehtävissä, jotka menevät yli 50 millisekunnin, sitä ylimenevää aikaa ensimmäisen sisällön piirron sekä interaktiivisuuden välillä. Tämä koskee 30 % kokonaistuloksesta. (web.dev, ei pvm.-g)

- Cumulative Layout Shift (CLS), eli Kasautunut Sijoittelun Muutos, mittaa isointa muutosta verkkosivulla, jos elementtejä lisätään verkkosivulle ensilatauksen jälkeen, kuten mainokset tai kuvat, joiden kokoa ei tiedetä, jotka latauksen yhteydessä puskee muita elementtejä alemmas verkkosivulla. Tämä koskee 15 % kokonaistuloksesta. (web.dev, ei pvm.-a)

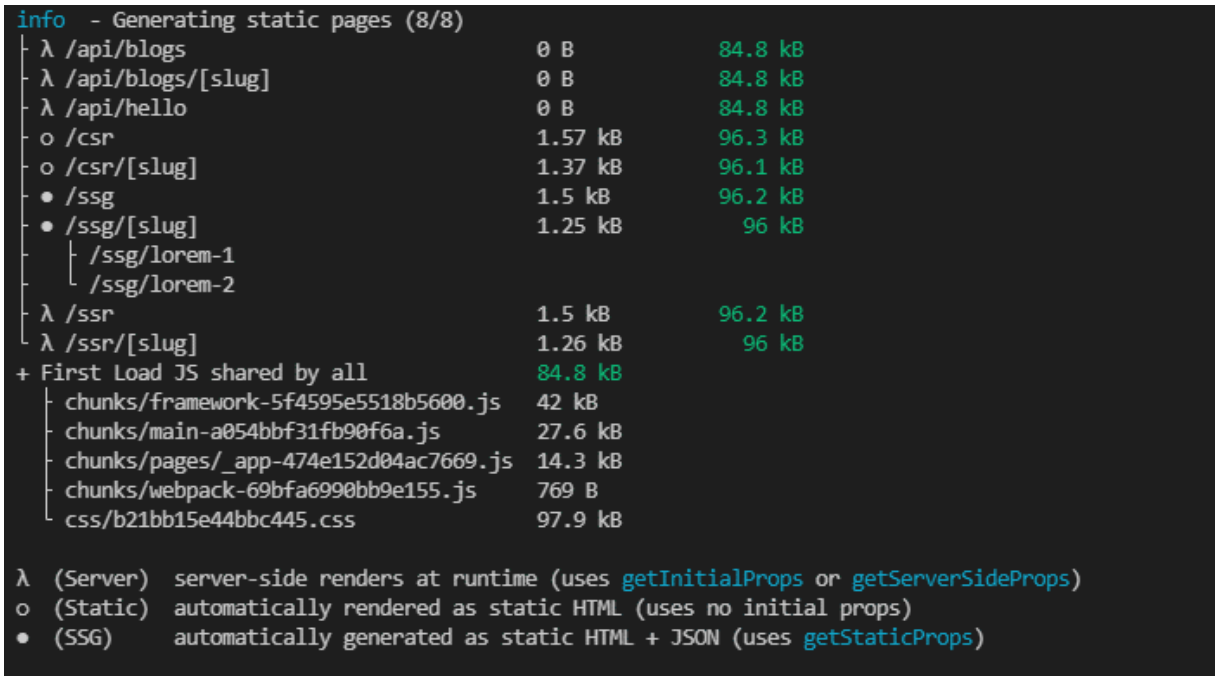
Näiden avulla saadaan hyvä kuva verkkosivun suorituskyvystä, ja pystytään saamaan parempi kuva eri tekniikoista mitä projektissa luotiin.

6.2 Tulokset

Kun verkkosivu on valmis, voidaan sitä alkaa testaamaan käyttäen Lighthousea. Jotta sivut toimivat optimaalisesti, täytyy ne kääntää tuotantoon, koska kehitysvaiheessa sivut eivät toimi täydellä nopeudella helpottamaan kehitystä. Tämä tapahtuu ajamalla rakennuskomento, ja siitä tulee hyvä palaute sivujen tiloista Kuva 8:n mukaan. Siitä näkyy, kuinka tietyt kansiot ja tiedostot on merkattu, esimerkiksi ”/csr” polku on pelkkiä staattisia

tiedostoja, koska siellä tiedonhaku tapahtuu selaimella, ja ”/ssg” kansiossa näkyy valmiiksi generoituja sivuja, koska ne rakennetaan tässä vaiheessa.

Kuva 8 Sivujen kääntämisvaiheessa tuleva palaute verkkosivustosta.



Kun sivut on rakennettu ja aloitettu jakamaan sivuja normaaliin tapaan, voidaan sivuja alkaa testaamaan. Testauksessa on käytetty testaushetkellä uusinta Chrome-selainta versio 98 puhtaalla asennuksella, sekä sen sisältämää Lighthouse 9.1.0. Testausmenetelmänä oli mennä samalle blogisivulle, tehdä sivulle Lighthouse-testi simuloiden mobiililaitetta, ottaa arvot ylös ja mennä seuraavaan menetelmään, toistaen tämän 3 kertaa jotta saadaan luotettavampia tuloksia. Näiden tuloksien keskiarvoilla saatiin parempia lukemia, jotka on listattu Taulukko 2:een. Koko testaus tehtiin vielä toiseen kertaan, valiten pöytätietokoneen simulointi, jonka tulokset löytyvät Taulukko 3:sta.

Taulukko 2. Kolmen ajon tuloksien keskiarvot Lighthouse-testistä simuloiden mobiililaitetta.

Testausmenetelmä	CSR	SSR	SSG
FCP	1.20 s	1.20 s	1.20 s

SI	1.20 s	1.20 s	1.20 s
LCP	2.90 s	2.20 s	2.27 s
TTI	2.33 s	1.97 s	2.10 s
TBT	56.67 ms	23.33 ms	33.33 ms
CLS	0.009	0	0

Taulukko 3. Kolmen ajon tuloksien keskiarvot Lighthouse-testistä simuloiden tietokonetta.

Testausmenetelmä	CSR	SSR	SSG
FCP	0,30 s	0,30 s	0,30 s
SI	0,33 s	0,30 s	0,33 s
LCP	0,70 s	0,57 s	0,53 s
TTI	0,30 s	0,30 s	0,30 s
TBT	0 ms	0 ms	0 ms
CLS	0.5	0	0

6.3 Pohdinta

Tuloksia katsoessa pitää muistaa, että nämä testit tehtiin todella suotuisissa olosuhteissa. Palvelin ajettiin samalla laitteella kuin testausalusta, eikä sovelluksessa tarvinnut mitään ulkoista tietolähdettä kuten tietokantaa, jonka kyselyistä tulisi ylimääräistä viivettä. Tuloksista silti näkee eroja eri tekniikoissa, joista saa paremman kuvan niiden mahdollisuuksista.

Pöytätietokoneen arvot ovat paremmat, kun vertaa Taulukko 2:a ja Taulukko 3:a. Kaikki sivut lataavat ja piirtyvät hyvin nopeasti, mutta pienen eron huomaa isoimman sisällön piirroksessa sekä kasautuneen sijoittelun muutokset- osioissa. Kummatkin näistä selittää selainpuoleinen tiedonhaku, joka täyttää sisällön sivun ensilatauksen jälkeen. Next.js tuottamat staattiset sivut, sekä sen tarjoama palvelimen puolella renderöidyt sivut lataavat nopeasti koska sivun päätietoja voidaan aloittaa piirtämään heti. Tämän takia sivun pääsisältö kannattaisi hakea palvelimen puolella ja sisältö, joka ei ole niin tärkeää nähdä heti ensimmäisenä, voidaan silti hakea selaimen puolella.

Samanlailla kuin pöytäkoneella, mobiililaitteellakin näkyy isoja eroja selaimen puoleisessa tiedonhaussa, eritysesti ison sisällön piirroksessa, mutta huomattava ero sen lisäksi näkyy kokonaisessa lukitussa ajassa sekä pieni ero jos mitataan aikaa interaktiivisuuteen. Kokonaisuudessakin nämä luvut ovat huomattavasti hitaampia kuin tietokoneella, koska usein mobiililaite on vähemmän tehokas eikä se ole yhdistetty johdolla internettiin. Mobiililaitteella tulee huomaamaan, jos sivun tärkeän sisällön lataamiseen kestää yli sekunnin lisää, niin kuin Taulukko 2:stä näkee.

Vaikka projektin aihe, eli blogisivu, on hyvin suotuisa palvelimen puolen renderöimiselle, minkä näistä tuloksista, sekä aikaisemmasta teoriasta kuitenkin voidaan nähdä, on Next.js:stä paljon etua. Koska se mahdollistaa kaikki nämä eri tiedonhakutavat, jolloin kehityksen aikana voidaan valita paras mahdollinen tekniikka eri osille projektia. Tämän avulla voidaan tehdä isompi osa kokonaisuutta samassa projektissa, sekä se auttaa kehittäjiä, koska kaikki osat tehdään samalla tavalla.

7 Yhteenveto

Verkkosivujen kehittäminen on edistynyt nopeasti, ja React on tällä hetkellä suositun kirjasto niiden kehittämiseen. Se kuitenkin sisältää tiettyjä kipukohtia, mitä Next.js yrittää korjata. Next.js tuntuu käytännössä hyvin kehitetyltä verkkosivujen kehityskehykseltä, jossa on selviä parannuksia React-pohjalle. Vaikka se tekee enemmän valintoja kehittäjän puolesta, joka voidaan nähdä huonona asiana, tuntuu se silti hyvin paljon React:ilta ja sen lisäykset tuntuvat selviltä ratkaisuilta näihin ongelmiin.

Projektissa tutkittiin Next.js:n kehittäjäkokemusta, sekä kuinka sen tarjoamat verkkosivujen palvelutavat eroavat toisistaan. Tuloksista huomattiin, kuinka palvelinpuolen renderöinti auttaa verkkosivun suorituskyvyssä, mobiililla sekä tietokoneella. Tämän lisäksi todettiin, kuinka helppoa projektissa oli käyttää näitä eri tekniikoita verkkosivujen tiedonhakuun, joka nopeuttaa sivun lataamista ja parantaa käyttäjäkokemusta.

Vaikka verkkokehityksen ekosysteemi muuttuu nopeaa tahtia, on React noussut yhdeksi suosituimmaksi kirjastoksi verkkosovelluksien kehittämiseen. Sen tulevaisuus saattaa alkaa siirtymään myös palvelimen puolelle, alkaen Reactin tulevasta ominaisuudesta nimeltä palvelinkomponentit, mutta kunhan Next.js pystyy päivittymään Reactin rinnalla ja kehittämään sen päälle, saattaa se olla hyvä vaihtoehto seuraavaan projektiin.

Lähteet

Chiarelli, A. (2018). *Beginning React: Simplify Your Frontend Development Workflow and Enhance the User Experience of Your Applications with React*. Packt Publishing, Limited. http://ebookcentral.proquest.com/lib/hamk-ebooks/detail.action?docID=5477665

Danilec, A. (2020, syyskuuta 21). *Client-Side Rendering or Server-Side Rendering—What is the best solution for your next application?* https://www.blog.duomly.com/client-side-rendering-vs-server-side-rendering-vs-prerendering/

Dawson, C. (2014, heinäkuuta 25). JavaScript's History and How it Led To ReactJS. *The New Stack*. https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/

developers.google.com. (ei pvm.). *Lighthouse | Tools for Web Developers*. Google Developers. Noudettu 1. maaliskuuta 2022, osoitteesta https://developers.google.com/web/tools/lighthouse

freecodecamp.org. (2020, heinäkuuta 16). *What is npm? A Node Package Manager Tutorial for Beginners*. FreeCodeCamp.Org. https://www.freecodecamp.org/news/what-is-npm-a-node-package-manager-tutorial-for-beginners/

Hoffmann, J. (ei pvm.). *The Web's Timeline*. The History of the Web. Noudettu 21. toukokuuta 2022, osoitteesta https://thehistoryoftheweb.com/timeline/

Luong, A. (2022, tammikuuta 24). *Practical Guide to Web Rendering*. https://crystallize.com/blog/web-rendering

mozilla.org. (ei pvm.-a). *About JavaScript—JavaScript | MDN*. Noudettu 28. tammikuuta 2022, osoitteesta https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

mozilla.org. (ei pvm.-b). *SEO - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. Noudettu 21. toukokuuta 2022, osoitteesta https://developer.mozilla.org/en-US/docs/Glossary/SEO

nextjs.org. (ei pvm.-a). *Advanced Features: Automatic Static Optimization | Next.js*. Noudettu 7. helmikuuta 2022, osoitteesta https://nextjs.org/docs/advanced-features/automatic-static-optimization

nextjs.org. (ei pvm.-b). *Advanced Features: Static HTML Export | Next.js*. Noudettu 14. helmikuuta 2022, osoitteesta https://nextjs.org/docs/advanced-features/static-html-export

nextjs.org. (ei pvm.-c). *API Routes: Introduction | Next.js*. Noudettu 4. helmikuuta 2022, osoitteesta https://nextjs.org/docs/api-routes/introduction

nextjs.org. (ei pvm.-d). *Basic Features: Font Optimization | Next.js*. Noudettu 6. helmikuuta 2022, osoitteesta https://nextjs.org/docs/basic-features/font-optimization

nextjs.org. (ei pvm.-e). *Basic Features: Handling Scripts | Next.js*. Noudettu 9. helmikuuta 2022, osoitteesta https://nextjs.org/docs/basic-features/script

nextjs.org. (ei pvm.-f). *Basic Features: Image Optimization | Next.js*. Noudettu 4. helmikuuta 2022, osoitteesta https://nextjs.org/docs/basic-features/image-optimization

nextjs.org. (ei pvm.-g). *Basic Features: Static File Serving | Next.js*. Noudettu 7. helmikuuta 2022, osoitteesta https://nextjs.org/docs/basic-features/static-file-serving

nextjs.org. (ei pvm.-h). *Case Study—Hulu | Next.js*. Noudettu 9. helmikuuta 2022, osoitteesta https://nextjs.org/case-studies/hulu

nextjs.org. (ei pvm.-i). *Data Fetching: GetServerSideProps | Next.js*. Noudettu 10. helmikuuta 2022, osoitteesta https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props

nextjs.org. (ei pvm.-j). *Data Fetching: Incremental Static Regeneration | Next.js*. Noudettu 7. helmikuuta 2022, osoitteesta https://nextjs.org/docs/basic-features/data-fetching/incremental-static-regeneration

nextjs.org. (ei pvm.-k). *Learn | Next.js*. Noudettu 31. tammikuuta 2022, osoitteesta https://nextjs.org/learn

nextjs.org. (ei pvm.-l). *Routing: Introduction | Next.js*. Noudettu 31. tammikuuta 2022, osoitteesta https://nextjs.org/docs/routing/introduction

nodejs.dev. (ei pvm.). *A brief history of Node.js*. A Brief History of Node.Js. Noudettu 21. tammikuuta 2022, osoitteesta https://nodejs.dev/learn/a-brief-history-of-nodejs

nodejs.org. (ei pvm.). *About*. Node.Js. Noudettu 28. tammikuuta 2022, osoitteesta https://nodejs.org/en/about/

reactjs.org. (ei pvm.-a). *Create a New React App – React*. Noudettu 30. tammikuuta 2022, osoitteesta https://reactjs.org/docs/create-a-new-react-app.html

reactjs.org. (ei pvm.-b). *Introducing Hooks – React*. Noudettu 30. tammikuuta 2022, osoitteesta https://reactjs.org/docs/hooks-intro.html

reactjs.org. (ei pvm.-c). *Introducing JSX – React*. Noudettu 26. tammikuuta 2022, osoitteesta https://reactjs.org/docs/introducing-jsx.html

reactjs.org. (ei pvm.-d). *JSX In Depth – React*. Noudettu 29. tammikuuta 2022, osoitteesta https://reactjs.org/docs/jsx-in-depth.html

reactjs.org. (ei pvm.-e). *React – A JavaScript library for building user interfaces*. Noudettu 29. tammikuuta 2022, osoitteesta https://reactjs.org/

reactrouter.com. (ei pvm.). *Declarative routing for React apps at any scale | React Router*. Noudettu 31. tammikuuta 2022, osoitteesta https://reactrouter.com/

saavutettavuusvaatimukset.fi/. (ei pvm.-a). Tietoa WCAG-ohjeistuksesta. *Saavutettavuusvaatimukset*. Noudettu 6. elokuuta 2022, osoitteesta

https://www.saavutettavuusvaatimukset.fi/digipalvelulain-vaatimukset/tietoa-wcag-kriteereista/

saavutettavuusvaatimukset.fi/. (ei pvm.-b). Yleistä saavutettavuudesta. *Saavutettavuusvaatimukset*. Noudettu 3. toukokuuta 2022, osoitteesta https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/

saavutettavuusvaatimukset.fi/. (2022, heinäkuuta 16). WCAG 2.1: Lain vaatimukset. *Saavutettavuusvaatimukset*. https://www.saavutettavuusvaatimukset.fi/digipalvelulain-vaatimukset/wcag-2-1/

stackoverflow.com. (2021). *Stack Overflow Developer Survey 2021*. Stack Overflow. https://insights.stackoverflow.com/survey/2021/

typescriptlang.org. (ei pvm.). *Typescriptlang.org*. Noudettu 29. tammikuuta 2022, osoitteesta https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

Vaggalis, N. (2010, marraskuuta 3). *Weakly Typed Languages*. https://www.i-programmer.info/programming/theory/1469-type-systems-demystified-part2-weak-vs-strong.html

w3.org. (ei pvm.). *HTML & CSS - W3C*. Noudettu 6. maaliskuuta 2022, osoitteesta https://www.w3.org/standards/webdesign/htmlcss

web.dev. (ei pvm.-a). *Cumulative Layout Shift (CLS)*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/cls/

web.dev. (ei pvm.-b). *First Contentful Paint*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/first-contentful-paint/

web.dev. (ei pvm.-c). *Largest Contentful Paint (LCP)*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/lcp/

web.dev. (ei pvm.-d). *Lighthouse performance scoring*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/performance-scoring/

web.dev. (ei pvm.-e). *Speed Index*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/speed-index/

web.dev. (ei pvm.-f). *Time to Interactive*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/interactive/

web.dev. (ei pvm.-g). *Total Blocking Time*. Web.Dev. Noudettu 3. maaliskuuta 2022, osoitteesta https://web.dev/lighthouse-total-blocking-time/

whatwg.org. (2022, tammikuuta 21). *FAQ — WHATWG*. https://whatwg.org/faq#living-standard

Wirfs-Brock, A., & Eich, B. (2020). JavaScript: The first 20 years. *Proceedings of the ACM on Programming Languages*, *4*(HOPL), 77:1-77:189. https://doi.org/10.1145/3386327

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyön projektin aikana tulevaa tietoa pidetään tekijän tietokoneella E-asemalla. Siitä pidetään ajanmukainen varmuuskopio koulun tarjoamassa OneDrive-pilvipalvelussa. Projektista tulevat tiedot tullaan säilyttämään ainakin vuoden verran opinnäytetyön valmistumisen jälkeen. Opinnäytetyössä ei ole tarvinnut käsitellä luottamuksellista tietoa tai tietoa, jota tarvitsisi tuhota. Teoriaosuuden lähteet ovat merkitty lähdelistaan