



React vs. Next.js: Web-sovellusten suoritusky- vyn vertailu

Leevi Laukka

2023 Laurea



Laurea-ammattikorkeakoulu

React vs. Next.js: Web-sovellusten suorituskyvyn vertailu

Leevi Laukka

Tradenomi (AMK), tietojenkäsittely

Opinnäytetyö

Joulukuu, 2023

Leevi Laukka

React vs. Next.js: Web-sovellusten suorituskyvyn vertailu

Vuosi	2023	Sivumäärä	30
-------	------	-----------	----

Tämän opinnäytetyön tavoitteena on verrata React.js ja Next.js JavaScript-kirjastoja. Kirjastoja verrataan toiminnallisuuden, kehittäjäkokemuksen (developer experience), sivuston latausnopeuden / koon ja helppokäyttöisyyden kannoilta.

Vertailu toteutetaan kehittämällä molemmilla kirjastoilla sama yksinkertainen verkkosivusto, joka hyödyntää kirjastojen ominaisuuksia ja parhaita käytänteitä. Sivustot toteutetaan TypeScript-kielellä. Sivustojen suorituskykyä vertaillaan Edge-selaimen Lighthouse-työkalulla.

Kehitys- ja tutkimustyön tuloksena ehdotettiin kirjastoille omat käyttötarkoituksensa ja -kohteensa. Työn lopuksi pohdittiin myös työn laajuutta ja mahdollista jatkokehitystä.

Leevi Laukka

React vs. Next.js: Comparison of web application performance

Year

2023

Pages

30

The aim of this Bachelor's thesis was to compare the React.js and Next.js JavaScript libraries. The libraries were compared in terms of functionality, developer experience (DX), website loading speed and file size, and ease of use.

The comparison was carried out by developing the same simple website with both libraries, utilizing the features and the best practices of the libraries. The sites were implemented in the TypeScript language. The performances of the sites were compared with the Lighthouse tool of the Edge browser.

As a result of the development and research work, the libraries were found to have their own uses and targets. The scope of the work and possible further development were also discussed at the end of the thesis.

Keywords: typescript, react, nextjs, server, client

Sisällys

1	Johdanto.....	6
2	Työn lähtökohdat.....	6
2.1	Kehittämisprojekti ja tutkimuskysymykset	7
2.2	Aihealueen rajausta	7
2.3	Keskeiset käsitteet.....	7
3	Teknologioiden esittely.....	8
3.1	React.....	8
3.1.1	JSX.....	8
3.1.2	Komponentit	9
3.1.3	Virtual Document Object Model	11
3.1.4	Client Side Rendering	11
3.2	Next.js	12
3.2.1	App Router	12
3.2.2	Optimoinnit.....	13
3.2.3	Server Side Rendering.....	13
4	Toteutettavan sovelluksen toiminnot	14
5	Toteutus ja testaus	15
5.1	React.....	15
5.1.1	Projektin luonti	15
5.1.2	Tiedostorakenne	16
5.1.3	Toiminnallisuuden toteutus	17
5.1.4	Testaus.....	19
5.2	Next.js	22
5.2.1	Projektin luonti	22
5.2.2	Tiedostorakenne	23
5.2.3	Toiminnallisuuden toteutus	23
5.2.4	Testaus.....	25
6	React vs. Next.js	27
7	Yhteenveto	28
8	Jatkokehitysehdotukset	28
	Lähteet.....	29
	Kuviot	30

1 Johdanto

Verkkosivustojen kehitys on muuttunut viime vuosikymmenen aikana paljon. Tarve monimutkaisemmille ja dynaamisille verkkosivustoille on kasvanut. Tästä tarpeesta on syntynyt monia erilaisia JavaScript-ohjelmointikehyksiä kuten React, Angular ja Svelte. Nämä ohjelmointikehykset auttavat kehittäjiä kehittämään suuria dynaamisia verkkosivustoja tehokkaasti.

Vuonna 2013 julkaistun Reactin kaltaisissa kehyksissä on kuitenkin huonot puolensa. Sivuston kasvaessa sivustoa ei välttämättä ole muistettu optimoida kunnolla, joka johtaa hitaisiin mm. latausaikoihin ja huonompaan yleiseen suorituskykyyn, sillä kaikki JavaScript-koodi tulee ajaa käyttäjän selaimessa.

Uudemmat SSR (Server Side Rendering) -teknologiat korjaavat näitä ongelmia. Tällä hetkellä ehkä suosituin SSR-kirjasto on Next.js. Next.js toimii Reactin ”päällä” ja mahdollistaa mm. tarvittavan JavaScript-koodin ajamisen palvelimella käyttäjän laitteen sijaan.

Tämän opinnäytetyön tarkoitus on kertoa Reactista, Next.js:stä ja niiden eroista, sekä verrata niiden suorituskykyä ja kehittäjäkokemusta kehittämällä molemmilla yksinkertainen sivusto ja vertaamalla niitä keskenään.

2 Työn lähtökohdat

Kehittämistyö sai alkunsa halusta selvittää, kuinka kehitys ja saavutettava lopputulos eroavat Next.js ja React-kirjastoilla toteutetun toteutuksen välillä.

SSR- teknologioiden laajempi hyödyntäminen johtaisi parempaan verkkoselainkokemukseen sivustojen loppukäyttäjille lyhyempien latausaikojen ja responsiivisempien sivustojen muodossa, varsinkin suurilla ja monimutkaisilla verkkosivustoilla. On tärkeää levittää tietoa näistä teknologioista.

Kehittämistyön tuloksista hyötyy jokainen yritys tai yksityinen web-kehittäjä, joka kokee sivustonsa toimivan hitaasti tai on muuten kiinnostunut sivustonsa suorituskyvyn parantamisesta.

Kiinnostuin aiheesta YouTube-videoiden kautta. Next.js-kirjasto on saanut viime aikoina paljon lisää toiminnallisuutta (Next.js versio 13 julkaistiin elokuussa 2022) ja on tällä hetkellä suuressa nosteessa web-kehittäjien keskuudessa.

2.1 Kehittämiprojekti ja tutkimuskysymykset

Kehittämistyön tarkoituksena on kuvata yksinkertaisen verkkosivuston kehittäminen Reactilla ja Next.js:llä. Kehittämisen aikana kiinnitetään huomiota kirjastojen ominaisuuksiin ja puutteisiin.

Työn tutkimusosuus yrittää vastata seuraaviin kysymyksiin:

- Kumpi kirjasto on suorituskykyisempi (sivujen latausnopeuden, tiedostojen kokojen ja interaktiivisuuden kannalta)?
- Kumpi kirjasto tarjoaa paremman kehittäjäkokemuksen?

2.2 Aihealueen rajaus

Opinnäytetyössä kuvataan verkkosivuston suunnittelu-, kehitys- ja suorituskyvyn testausvaiheet. Työssä myös tutkitaan Next.js-kirjaston hyötyjä ja haittoja, sekä verrataan Next.js-toteutusta React-toteutukseen. Työ ei käsittele itse toiminnallisuuden testausta, vaan keskittyy puhtaasti suorituskyvyn ja kehittäjäkokemuksen vertailuun.

2.3 Keskeiset käsitteet

JavaScript	Web-ohjelmoinnissa käytetty ohjelmointikieli.
TypeScript	Microsoftin kehittämä JavaScriptin ”laajenne” joka lisää tuen mm. muuttujatyypeille.
DOM: Document Object Model	Oliomalli, jolla kuvataan HTML-sivun rakennetta. DOM mahdollistaa HTML-sivun manipuloinnin JavaScriptin avulla.
SSR: Server Side Rendering	Sivu renderöidään palvelimella, ja lopputulos lähetetään käyttäjälle.
Static Rendering	(Staatinen renderöinti) Sivusto renderöidään projektin build-vaiheessa ja valmis sivusto tallennetaan välimuistiin, josta se jaetaan käyttäjälle.
Lighthouse	Googlen työkalu sivustojen suorituskyvyn testausta varten. Sisältyy kaikkien Chromium-pohjaisten selainten kehittäjätyökaluihin.
First Contentful Paint	Sivun ensimmäinen sisältöä sisältävä piirto.

Time To Interactive

Sivun kaikki elementit ja niihin liittyvät tapahtumakuuntelijat on ladattu, sivu on valmis vastaanottamaan käyttäjän syötteitä.

3 Teknologioiden esittely

Tässä kappaleessa esitellään vertailtavat kohteet, React ja Next.js. Kappaleessa kuvataan molempien kohteiden suosituimpia ominaisuuksia. Vertailussa tulee muistaa, että Next.js perii lähes kaiken Reactin toiminnallisuuden, joten kaikki Reactin ominaisuudet löytyvät myös Next.js-kirjastosta, muttei toisinpäin.

3.1 React

React on Metan (entinen Facebook) vuonna 2013 julkaisema, reaktiivisten käyttöliittymien luontiin ja hallintaan tarkoitettu JavaScript-kirjasto.

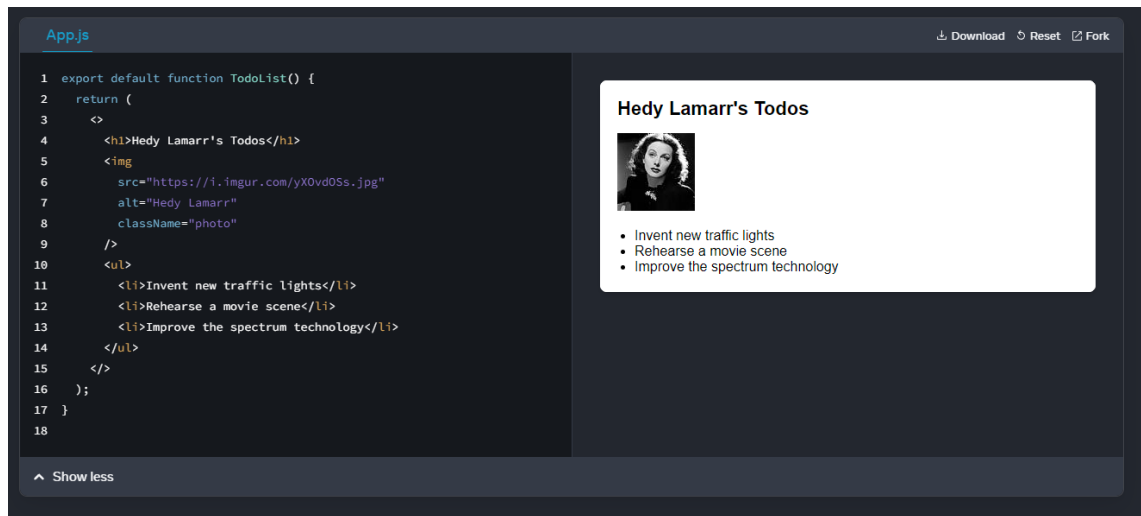
React luotiin helpottamaan suurien web-sovellusten kehittämistä sen komponenttipohjaisen arkkitehtuurin avulla. Suurempia kokonaisuuksia voidaan pilkkoa pienemmiksi, yksinkertaisemmiksi paloiksi.

Tämä kappale kuvaa Reactin tärkeimpiä ominaisuuksia.

3.1.1 JSX

JSX on laajennettu syntaksi HTML-koodin kirjoittamiseen JavaScript-koodin sisälle. JSX:n tärkein ominaisuus on mahdollisuus sisällyttää JavaScriptiä HTML-koodin sekaan. Näin käyttöliittymään saadaan helposti lisättyä logiikkaa. Reactissa komponentit määritellään JSX:llä.

Kuviossa 1 esimerkki komponentin määrittelystä JSX:llä. Tämä komponentti sisältää vain HTML-koodia, eikä ole täten dynaaminen.



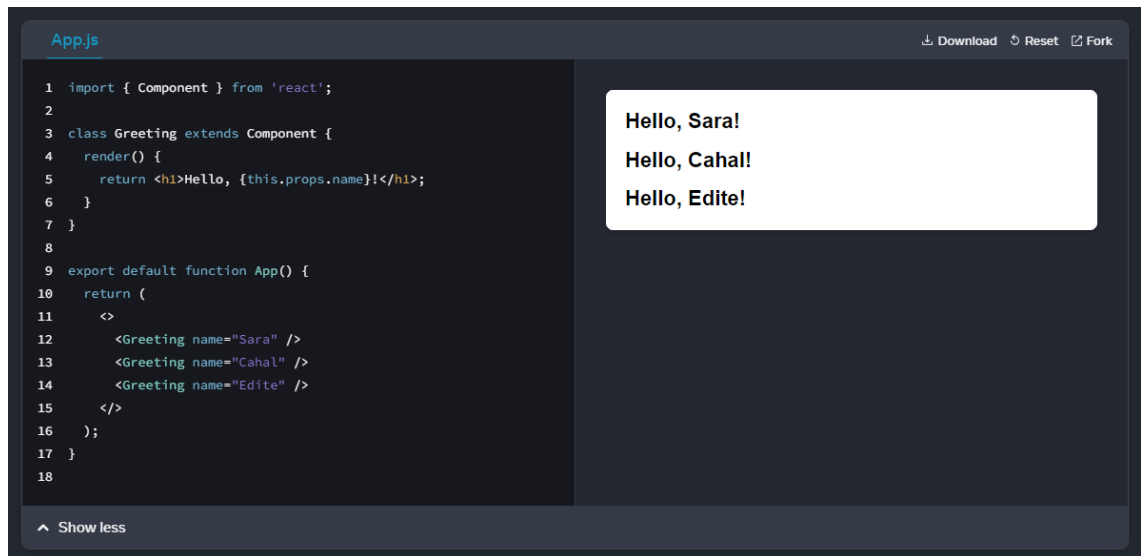
Kuvio 1: Esimerkki JSX:llä luodusta komponentista (Meta 2023)

3.1.2 Komponentit

Reactissa käyttöliittymä voidaan jakaa erillisiin komponentteihin. Komponentit sisältävät käyttöliittymän (HTML / JSX-koodi), komponentin logiikan ja tilan. Komponenteille voidaan antaa parametreja (props), joilla komponentin sisältöön voi vaikuttaa muista komponenteista käsin. Komponentit renderöitiin uudestaan, jos niiden tila tai propsit muuttuivat.

Reactissa komponentteja voidaan luoda kahdella eri tavalla, luokkakomponentteina tai funktiokomponentteina. Komponenttiluokka sisältää useita eri "lifecycle"-metodeja, joita ajetaan, kun komponentti päivittyy tai on päivittymässä. Ainut pakollinen komponenttiin liitettävä metodi on render. Render-metodi ajetaan joka kerta, kun komponentti tulee renderöidä uudestaan. Render-metodin palautusarvon tulee olla kelpo JSX-koodia, joka sitten muunnetaan HTML-koodiksi ja näytetään selaimessa. Luokkakomponenteilla on myös `componentWillUnmount`-metodi, joka ajetaan, kun komponentti on poistumassa DOMista. Tässä metodissa voidaan esimerkiksi sulkea avoimet tietokantayhteydet tai poistaa tietoja, joita sovellus ei enää tarvinnut muistista. Komponenttiluokissa oletus-state ja propsit ovat luokainstanssille luonnin aikana määritettyjä parametreja. (Meta 2023a)

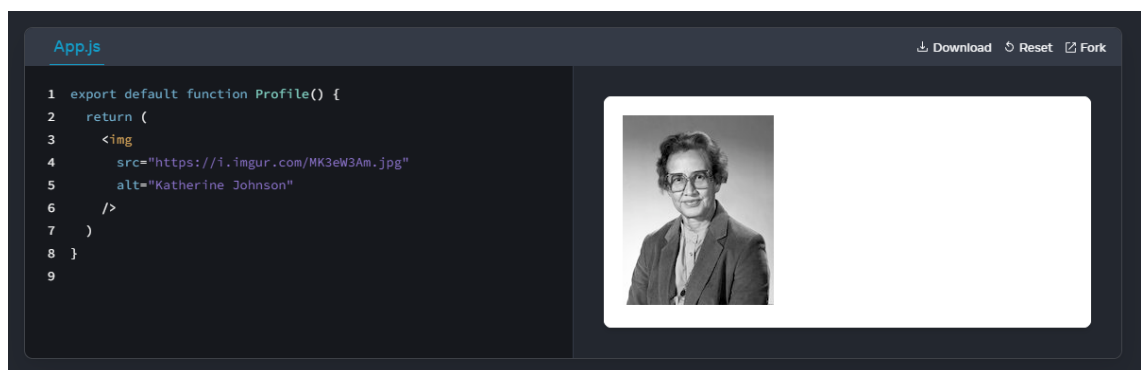
Kuviossa 2 esimerkki luokkakomponentista `Greeting`, joka vastaanottaa tekstiä proppina, ja piirtää tekstin dynaamisesti komponentin otsikkoelementtiin. App-pääkomponentti käyttää `Greeting`-komponenttia kolmesti eri arvoilla.



Kuvio 2: Esimerkki luokkakomponentista (Meta 2023a)

Luokkakomponentit koettiin kömpelöksi tavaksi määrittää komponentteja niiden vaatiman va-
kiotekstin takia, ja Meta, silloinen Facebook kehitti vaihtoehtoiseksi tavaksi funktiokom-
ponentit. Funktiokomponentit ovat normaaleja funktioita, jotka palauttavat JSX-koodia.
Funktiokomponenttien palautusarvo päivittyy automaattisesti, kun komponentti päivittyy, sa-
amalla tavalla kuin render-metodi ajettiin luokkakomponenteissa. Funktiokomponenteissa
propsit saadaan funktion parametreista ja tilaa hallitaan funktiokomponentteja varten luotu-
jen hookkien avulla. (Meta 2023b)

Kuviossa 3 esimerkki funktiokomponentista. Tämä komponentti ei vastaanota proppeja, ja pa-
lauttaa vain staattisesti määritetyn img-elementin



Kuvio 3: Esimerkki funktiokomponentista (Meta 2023b)

Hookit ovat funktioita, jotka on suunniteltu korvaamaan osa luokkakomponenttien tarjoa-
mista metodeista. Tilaa hallitaan state-hookin (useState) kautta. useState palauttaa listan,
joka sisältää kyseisen tilan nykyisen arvon ja funktion, jolla arvoa voidaan päivittää. Muut tär-
keimmät lifecycle-metodien tarpeet voidaan suorittaa Effect-hookilla (useEffect).

Komponentin `useEffect`-hook ajetaan aina, kun komponentti päivittyy. Sille voidaan määrittää myös palautusarvoksi funktio, joka ajetaan, kun komponenttia ollaan poistamassa, `componentWillUnmount`-metodin tapaan. (Meta 2023c)

3.1.3 Virtual Document Object Model

Virtual DOM (Document Object Model) on ohjelmointikonsepti, jossa haluttu versio käyttöliittymästä pidetään muistissa ja synkronoidaan oikean DOMin kanssa. Reactissa tämä käytäntö mahdollistaa sen deklarativisen luonteen. Reactille kerrotaan, missä tilassa käyttöliittymän tulee olla ja se huolehtii, että DOM vastaa tätä tilaa. Näin ohjelmoijan ei tarvitse huolehtia esimerkiksi elementtien attribuuttien muuttamisesta tai tapahtumien hallinnasta. React hoitaa kaiken tämän automaattisesti Virtual DOMin avulla. (Meta 2023d)

Virtual DOM:in käyttö on perinteisen DOM:in muokkaamista tehokkaampaa, sillä perinteistä DOM:ia muuttaessa koko sivu on piirrettävä uudestaan. Virtual DOM vertaa komponenteissa ja HTML-elementeissä tapahtuvia muutoksia ja piirtää uudestaan vain tarvittavat osat sivusta. Virtual DOM:in avulla komponentit myös renderöityvät samalla tavalla selaimesta tai käyttöjärjestelmästä riippumatta. (Mallick 2023)

3.1.4 Client Side Rendering

Reactissa koko sivusto lähetetään oletuksena käyttäjälle yhtenä html-tiedostona ja koottuna JavaScript-tiedostona. Suurissa sivustoissa, joita ei ole optimoitu hyvin, voi tämä JavaScript-tiedoston lataus ja suoritus kestää jonkin aikaa, eikä sivusto ole toiminnallinen tai pahimmassa tapauksessa edes näy käyttäjälle, ennen kuin tiedosto on suoritettu. Myös jo välimuitissa oleva, huonosti optimoitu React-sovellus voi toimia hitaasti, jos käyttäjällä on hyvin alitehoinen päätelaite, kuten esimerkiksi halvempi älypuhelin.

Sivuston suorituskyvyn mittaamista varten on luotu sivuston eri osien latausnopeuteen liittyviä mittareita. Esimerkiksi Googlen Lighthouse-työkalu käyttää mm. `First Contentful Paint`, `Time to interactive`, `First Meaningful Paint` ja `Largest Contentful Paint`.

`First Contentful Paint` mittaa aikaa siitä, kun sivua alettiin ladata siihen, kun ensimmäinen sisältö (teksti, kuvat, svg- tai canvas-elementit) näkyy sivulla (Walton 2019). `Largest Contentful Paint` mittaa, kuinka kauan sivun suurimmassa sisällössä kestää näkyä (Walton & Pollard 2023).

`Time to Interactive`-aika mittaa aikaa sivun latauksen aloituksesta siihen, kun sivun kaikki tapahtumakuuntelijat ovat latautuneet ja sivu on valmis vastaanottamaan käyttäjäsyötteitä. (Walton 2023)

Pitkät ”First Contentful Paint”- ja ”Time to Interactive”-ajat johtavat käyttäjien turhautumiseen ja lopulta sivuston käytöstä luopumiseen. Vuosina 2016-2017 Pinterest optimoi React-pohjaisen verkkosivustonsa perin pohjin. Sivuston First Meaningful Paint-aika (aika, jossa sivun pääsisältö latautui, ei virallisesti käytössä Lighthouse versio 6 lähtien) laski 4.2 sekunnista 1.8 sekuntiin ja Time to Interactive- aika 23 sekunnista noin viiteen ja puoleen. Sivustolla huomattiin optimointien jälkeen mobiililaitteilla käyttäjäsitoutumisen kasvavan noin 60 %, käyttäjien sivustolla kuluttaman ajan yli 40 % ja käyttäjien tuottamien mainostulojen noin 44 %. (Osmani 2017)

3.2 Next.js

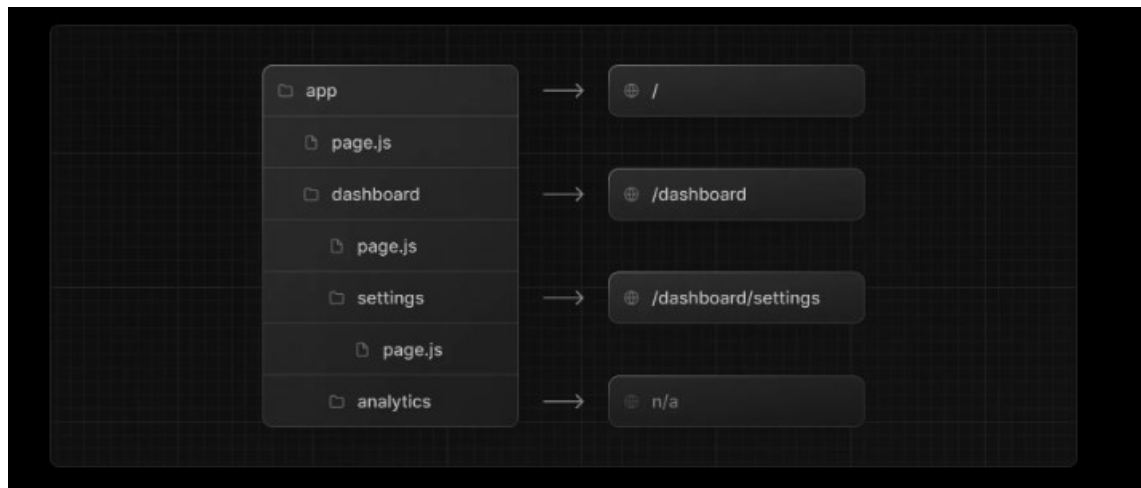
Next.js on Vercelin luoma, avoimen lähdekoodin JavaScript-sovelluskehys (eng. framework) Reactille. Next.js mahdollistaa React-aplikaatioiden renderöinnin palvelimella vaivatta. Next.js:n ensimmäinen versio julkaistiin vuonna 2016. Tämä luku kuvaa Next.js:n oleellisia ominaisuuksia.

3.2.1 App Router

Next.js:ssä jokainen sivuston sivu on erillinen JavaScript-tiedosto, joka sisältää React-komponentin. Esimerkiksi tiedosto /pages/about.js vastaisi selaimessa osoitetta /about. Vuonna 2022 Pages Router korvattiin App Routerilla, joka toimii pitkälti samalla tavalla kuin ennen, mutta mahdollistaa mm. layout- ja ccs-moduulitiedostojen käytön.

App Routeria käyttäessä sivuston tiedostot sijoitetaan app-hakemistoon. Reittejä määritetään hakemistoilla, jotka voivat sisältää hakemistoja ja JavaScript-tiedostoja. Jokainen hakemisto on oma URL-segmenttinsä, josta voidaan tehdä tavoitettavia sivuja luomalla hakemistoon page.js-tiedosto, joka tuo React-komponentin, joka renderöidään käyttäjälle. (Vercel 2023a)

Kuviossa 4 esimerkki app-hakemiston rakenteesta App Routeria käyttäessä. Hakemisto sisältää pääsivun, aliosoitteen dashboard, joka sisältää dashboard-sivun ja kaksi aliosoitetta settings ja analytics, joista settings sisältää oman sivunsa. Esimerkki kuvaa, miten hakemistot määrittyvät URL-osoitteiksi.



Kuvio 4: Esimerkki app-hakemistosta (Vercel 2023a)

3.2.2 Optimoinnit

Next.js tarjoaa valmiiksi optimoituja komponentteja korvaamaan joitakin perinteisiä HTML-elementtejä. Näitä komponentteja ovat mm. Image-, Link- ja Head-komponentit. (Vercel 2023b)

Image-komponentti on rakennettu natiivin `img`-elementin päälle. Se on optimoitu lataamaan kuvat vasta niiden näkyessä sivulla, ja se lataa kuvasta vain laitteelle sopivat versiot, säästäten kaistaa ja muistia. (Vercel 2023c)

Link-komponentti on rakennettu natiivin `a`- eli anchor-elementin päälle. Link-komponentissa viitattu sivu ladataan valmiiksi, ennen kuin käyttäjä edes klikkaa linkkiä. Näin siirtyminen linkatulle sivulle on sulavampaa.

Head-komponentti vastaa html-sivun head-osiota. Sivukomponentin sisällä olevan Head-komponentin sisältö generoidaan sivun html-koodin head-osioon. Näin esimerkiksi otsikot saadaan toimimaan helposti komponenttien kautta. (Vercel 2023d)

3.2.3 Server Side Rendering

Next.js:ssä sivustot voidaan renderöidä usealla eri tavalla. Oletuksena sivusto renderöidään staattisesti, eli kaikki sivut muunnetaan html-tiedostoiksi, jotka voidaan sitten jakaa internettiin. Tällä tavalla renderöidessä sivustot eivät voi olla dynaamisia. Staattinen renderöinti sopii sivustoille, joiden sisältö ei ole dynaamista tai päivitys kovinkaan usein. Esimerkiksi blogi-sivusto tai yksinkertainen tuotesivusto voisi olla sopiva kohde staattiselle renderöinnille. (Vercel 2023e)

Dynaamista renderöintiä käytettäessä sivu renderöidään palvelimella joka kerta kun käyttäjä menee sivulle. Dynaamiset sivut voivat hakea tietoja esimerkiksi tietokannoista tai ulkoisista lähteistä ja sisällyttää näitä sivulle. (Vercel 2023f)

Kolmas mahdollinen tapa renderöidä on striimaaminen (streaming). Next.js:llä sivun voi jakaa osiin ja osia voidaan sitten lähettää käyttäjälle, kun ne ovat valmiita. Näin käyttäjälle voidaan näyttää osa sivusta, vaikkei kaikki sisältö olisikaan vielä täysin valmista. (Vercel 2023g)

Next.js valitsee automaattisesti parhaan käytettävän renderöintitavan sivuston build-vaiheessa käytettävien toiminnallisuuden perusteella. Kaikille esitetyille tavoille yhteistä on se, että sivustot renderöidään palvelimella, eikä käyttäjän laitteella nopeuttaen latausaikoja ja muita viiveitä. Sivustoa jakavan laitteen tulee kuitenkin olla riittävän tehokas generoimaan sivuja lennosta, varsinkin jos sivustossa käytetään paljon dynaamisia sivuja. Staattisesti renderöidyt sivut generoidaan build-vaiheessa kaikki kerralla, joten niiden jakaminen ei vaadi kovinkaan paljoa suorituskykyä generoinnin jälkeen.

4 Toteutettavan sovelluksen toiminnot

Toteutettava sovellus / sivusto tulee olemaan yksinkertainen sääsivusto. Sivustolla tulee olla haku, jolla haetaan haluttu kaupunki, sekä sen tulee hakea kaupungin säätiedot dynaamisen osoitteen kautta, esim "root/weather/vantaa".

Kaikki halutut säätiedot haetaan OpenWeatherin API:n kautta. API:n kautta tuodaan myös kuvakkeet eri säätiloille.

Sivuston molemmat versiot toteutetaan TypeScriptillä kirjaston parhaan mahdollisen kehittäjäkokemuksen saavuttamiseksi. Sivusto toteutetaan ensin Reactilla ja muunnetaan siitä käyttämään Next.js:ää. Sivustoa ei siis luoda täysin alusta Next.js:llä, vaan sivuston rakenne muutetaan käyttämään Next.js:ää ja se optimoidaan sille.

Toteutuksissa käytetään mahdollisimman vähän Reactin ja Next.js:n ulkoisia kirjastoja. Kirjastoja käytetään vain täyttämään kaikki toteutuksessa haluttu toiminnallisuus.

Sivuston ulkonäköön ei kiinnitetä ylimääräistä huomiota. Tutkimuksen tavoitteena on tutkia eri kirjastojen suorituskykyä.

5 Toteutus ja testaus

Tässä kappaleessa kuvataan molempien toteutuksien kehittäminen ja testaus erikseen. Kappaleessa kuvataan projektien luonti, tiedostorakenteen, toiminnallisuuksien toteutus ja testaus. Toteutukset kehitetään VS Code-koodieditorissa.

5.1 React

Tämä kappale kuvaa sovelluksen React-version toteutuksen ja testauksen. Toteutuksessa ja testauksessa on käytetty hyödyksi Reactin omaa dokumentaatiota, ja sieltä löytyviä oppaita.

5.1.1 Projektin luonti

React-toteutus aloitetaan pohjustamalla uusi React-projekti. Tätä varten on tarjolla virallinen create-react-app-skripti, joka generoi pohjan uudelle React-projektille ja lataa kaikki Reactiin kuuluvat kirjastot. Tässä tapauksessa skriptille annetaan template-asetus arvolla typescript. Tämä kertoo skriptille, että käyttäjä haluaa asentaa oletuskirjastojen lisäksi myös typescript-kehitykseen vaadittavat kirjastot. Skripti luo myös valmiiksi muutaman komennon, joilla voi muun muassa käynnistää sovelluksen kehitysversion. Skriptin suorittaminen kuvattu kuviossa 5.

Sovelluspohjan lisäksi skripti luo hakemistoon myös uuden git-tietovaraston.

```

PS C:\Projektit\node> npx create-react-app ont-react --template typescript

Creating a new React app in C:\Projektit\node\ont-react.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template-typescript...

added 1462 packages in 46s

242 packages are looking for funding
  run 'npm fund' for details

Initialized a git repository.

Installing template dependencies using npm...

added 47 packages, removed 1 package, and changed 2 packages in 4s

246 packages are looking for funding
  run 'npm fund' for details

We detected TypeScript in your project (src\App.test.tsx) and created a tsconfig.json file for you.

Your tsconfig.json has been populated with default values.

```

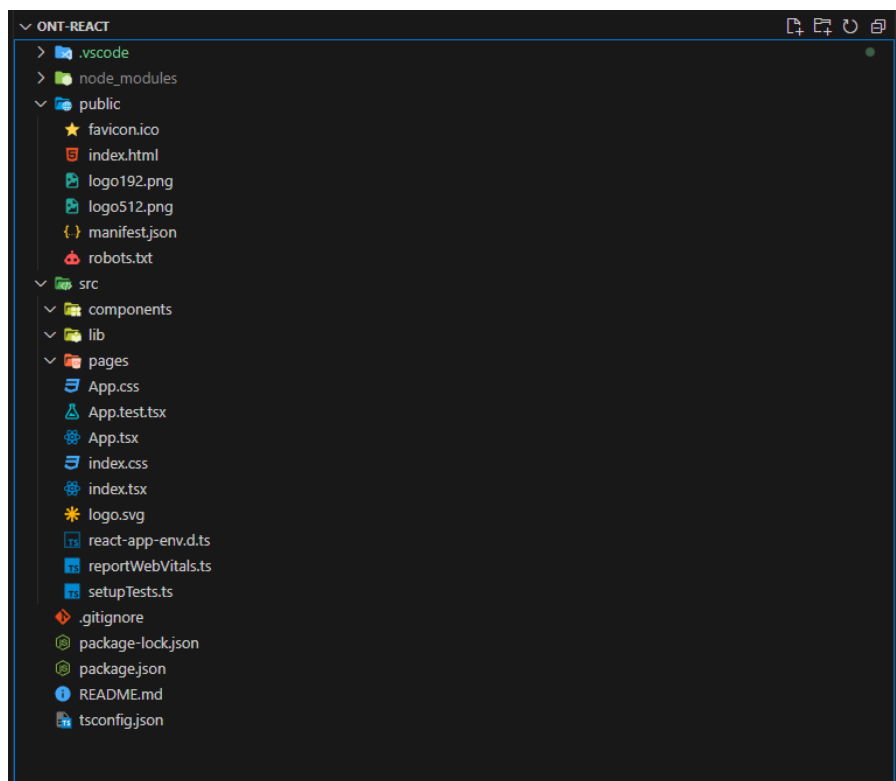
Kuvio 5: Uuden React-projektin luonti create-react-app:in avulla

5.1.2 Tiedostorakenne

Oletuksena create-react-app skripti luo public- ja src-hakemistot. Public-hakemisto sisältää kaikki sivuston staattiset tiedostot, kuten sivuston faviconin ja robots.txt-tiedoston. Src-hakemisto sisältää kaikki projektin tsx-, css- ja muut sivuston build-vaiheessa tarvittavat tiedostot.

src-hakemistoon luotiin components-, lib- ja pages-hakemistot. Components-hakemisto sisältää kaikki sovelluksessa uudelleenkäytettävät komponentit, kuten hakupalkit ja painikkeet. Pages-hakemisto sisältää komponentit, joita käytetään sivuston eri sivuina. Lib-hakemisto sisältää muita mahdollisesti tarvittavia aputiedostoja.

Kuviossa 6 kuvattuna kaikki create-react-app-skriptin luomat tiedostot ja hakemistot, sekä edellä mainitsemani lisäykset.



Kuvio 6: React-projektin rakenne VS Code-editorissa

5.1.3 Toiminnallisuuden toteutus

Projektiin lisättiin kaksi ulkoista kirjastoa, react-router-dom ja styled-components. React-router-dom-kirjasto mahdollistaa helpon reitityksen sivuston eri alisivuilla. Kirjasto reitittää annetut reitit annetuille komponenteille ja mahdollistaa muunmuassa tietojen lataamisen tietyn reitin latautuessa. Tämä valmis toiminnallisuus mahdollisti säätietojen lataamisen OpenWeatherMapin API:sta helposti. Styled-components-kirjasto mahdollistaa komponenttien helpon tyylittelyn.

Kaikki erilliset api-kutsut kirjoitettiin funktioiksi lib-hakemistoon api.ts-tiedostoon. Näin kutsut ovat helposti uudelleenkäytettävissä ja eristetty muusta koodista. Esimerkkejä näistä funktioista löytyy kuviosta 8.

Lib-hakemistoon luotiin myös types.ts-tiedosto, jossa määriteltiin api-kutsuista saatavien tietojen muoto mm. automaattista täydennystä varten.

Kuviossa 7 kuvattuna projektin tiedostorakenne komponenttien ja sivujen luonnin jälkeen.



Kuvio 7: React-projektin src-hakemisto toteutusvaiheen loppupuolella

```
const getPath: (path: string) => URL = (path: string): URL => {
  const url = new URL(path, basePath)
  url.searchParams.append(name: "appid", API_KEY)
  url.searchParams.append(name: "units", UNITS)
  url.searchParams.append(name: "lang", LANG)

  return url
}

export const getWeather: (city: string | undefined) ... = async (city: string | undefined): Promise<OWMWeather | undefi... => {
  if (!city) {
    return
  }

  const url = getPath(path: "weather")
  url.searchParams.append(name: "q", city)

  const response = await fetch(url.toString())
  const json: OWMWeather = await response.json()

  return json
}

export const getForecast: (city: string | undefined) ... = async (city: string | undefined): Promise<OWMForecast | undefi... => {
  if (!city) {
    return
  }

  const url = getPath(path: "forecast")
  url.searchParams.append(name: "q", city)

  const response = await fetch(url.toString())
  const json: OWMForecast = await response.json()

  return json
}
```

Kuvio 8: api.ts-tiedoston sisältöä

Components-hakemistoon luotiin komponentit nykyiselle säälle, sääennusteelle ja hakupal-
kille. Kaikkia komponentteja käytetään pages-hakemiston Weather-komponentissa, jota käy-
tetään /weather/:kaupunki- osoitteessa.

Sää tiedot ladataan Weather-komponenttiin react-router-dom-kirjaston useLoaderData-hookin avulla. Molemmille alikomponenteille annetaan sitten omat tietonsa proppeina. Kuviossa 9 kuvattuna tämä komponentti valmiina.

```
const Weather : () => Element = () : Element => {  
  const data: any = useLoaderData()  
  
  return (  
    <div>  
      <Search />  
      <CurrentWeather weatherData={data.weather} />  
      <Forecast forecastData={data.forecast} />  
    </div>  
  )  
}
```

Kuvio 9: Weather-komponentti

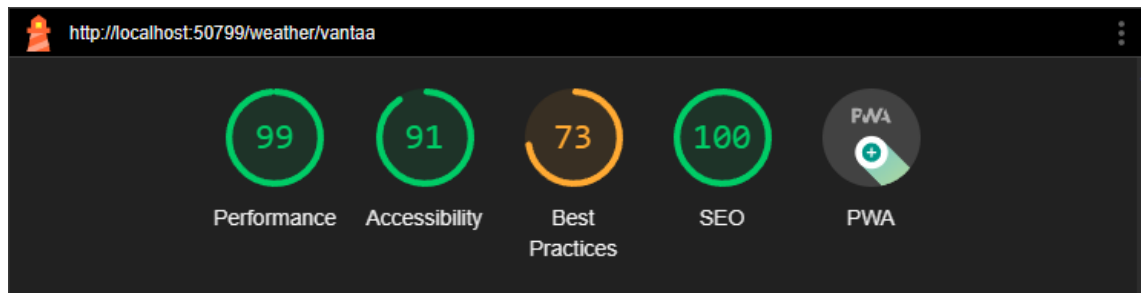
5.1.4 Testaus

Sivustosta luotiin palvelimella jaettava versio npm run build-komennolla. Tämä jaettiin serveriohjelmalla testausta varten.

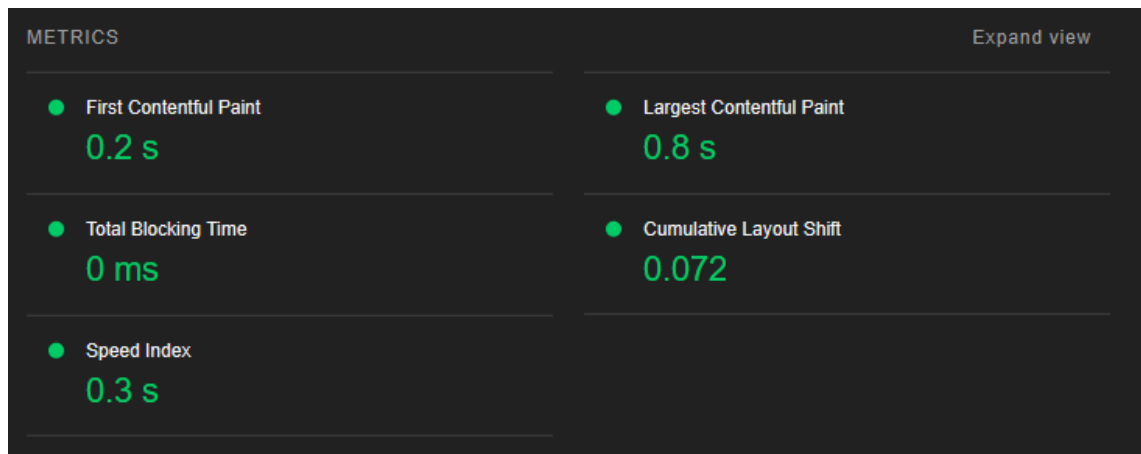
Sivustoa testattiin Edge-selaimen Lighthouse-työkalulla oletusasetuksilla sivulla /weather/vantaa.

React-toteutuksen ensimmäisessä piirroksessa kesti 0.2 sekuntia ja suurimmassa 0.8 sekuntia. Kaikki elementit olivat suoraan valmiita vastaanottamaan syötteitä. Nämä tulokset hieman yllättivät, sillä odotin latauksen olevan hieman hitaampaa. Todennäköisesti sivu ei ole tarpeeksi monimutkainen.

Kuvioissa 10 ja 11 kuvattuna Lighthouse-työkalun tarjoamat yhteenvedot sivun suorituskyvystä.



Kuvio 10: React-toteutuksen Lighthouse-yhteenveto

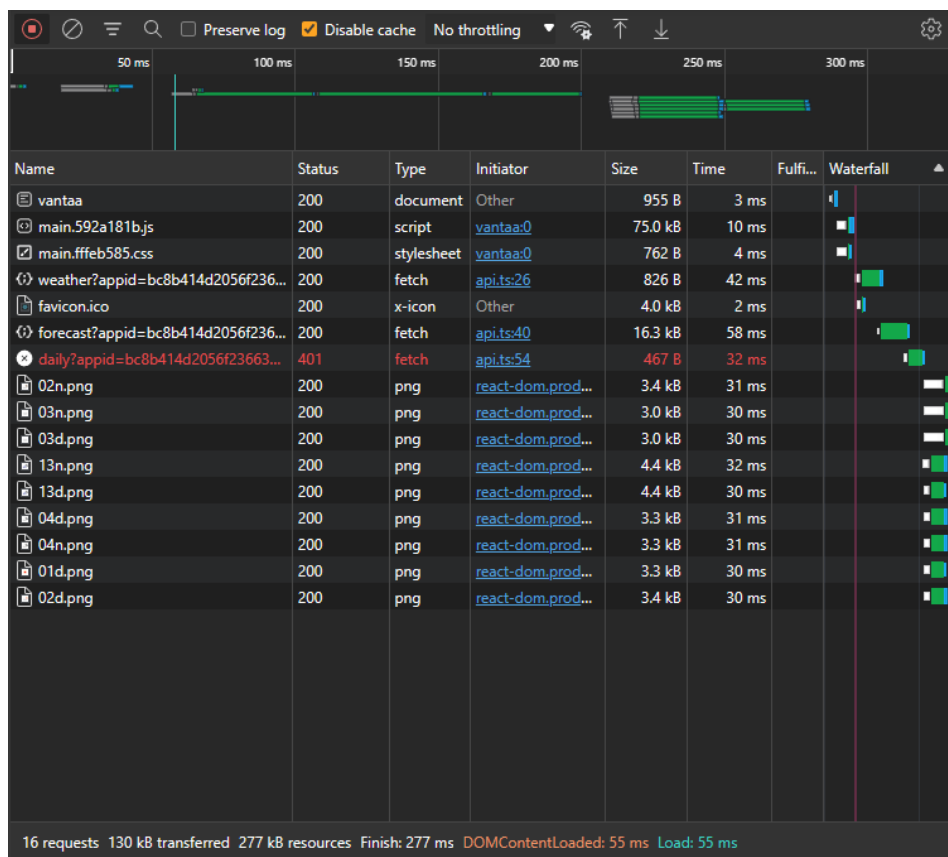


Kuvio 11: React-toteutuksen suorituskyky

Kehittäjätyökalujen Verkko-välilehdeltä voidaan todeta selaimen lataavan, välimuistin tyhjen-nyksen jälkeen ensimmäisellä latauksella 130 kilotavun edestä pakattua dataa (277 kt pakkaamaton). Toisella latauksella ladattujen tietojen määrä putoaa 18 kilotavuun. Selain siis tal-lentaa mm. kuvia välimuistiin, jotta seuraavat latauskerrat olisivat nopeampia. Kun välimuisti poistetaan käytöstä, selain lataa aina täydet 130 kilotavua dataa.

Sivuston tiedostojen lataamisessa (ilman välimuistia) kului yhteensä noin 300 millisekuntia per lataus.

Kuviossa 12 kuvattuna Verkko-välilehti React-sivulla. Huomioi ladattujen tietojen listasta myös fetch-kutsut. Nämä ovat OpenWeatherMapin API:sta haettuja tietoja.



Kuvio 12: React-toteutuksen Verkko-välilehti

5.2 Next.js

Tässä kappaleessa kuvataan Next.js-sivuston toteutus ja testaus. Toteutuksessa ja testauksessa on käytetty hyödyksi Next.js:n omaa dokumentaatiota, ja sieltä löytyviä oppaita.

5.2.1 Projektin luonti

Next.js-projekti luodaan hyvin samalla tavalla kuin React-projekti. Projektin luontiin käytetään create-next-app-skriptiä. Next.js:n skripti on helppokäyttöisempi kuin Reactin, sillä se kysyy käyttäjältä, mitä asetuksia projektin luomisessa halutaan käyttää. Skriptin käyttö kuvattu kuviossa 13.

Projektin ohjelmointikieleksi valittiin TypeScript, otettiin käyttöön uusi App Router ja lisäksi luotiin Tailwind CSS-konfiguraatiodostoto Tailwind CSS-kirjaston käyttöä varten.

```
PS C:\Projektit\node> npx create-next-app
✓ What is your project named? ... ont-next
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use 'src/' directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in C:\Projektit\node\ont-next.

Using npm.

Initializing project with template: app-tw

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- autoprefixer
- postcss
- tailwindcss
- eslint
- eslint-config-next

added 330 packages, and audited 331 packages in 8s

116 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
Initialized a git repository.

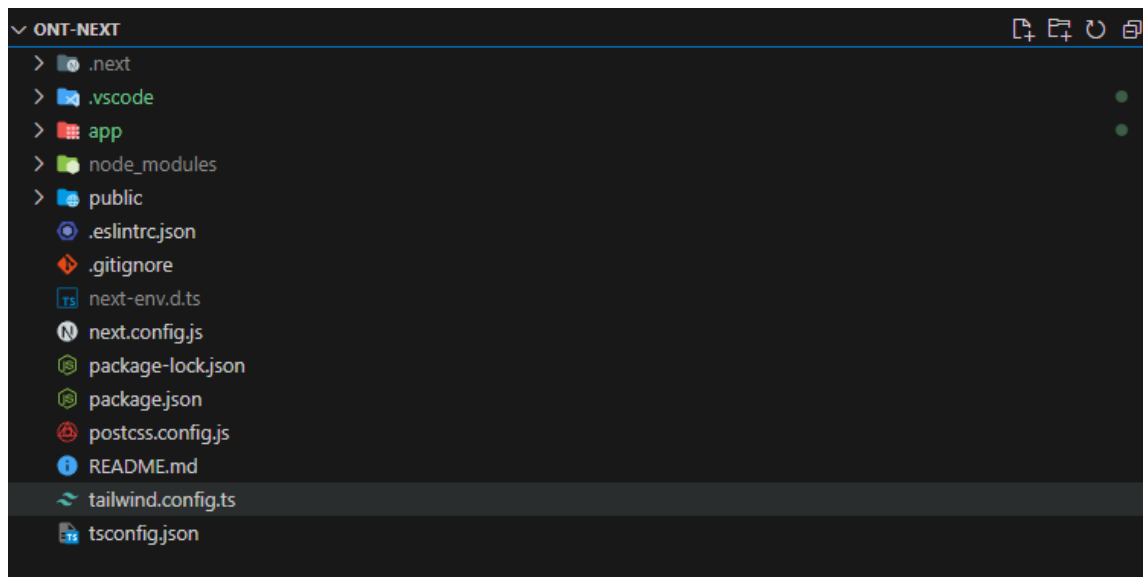
Success! Created ont-next at C:\Projektit\node\ont-next
```

Kuvio 13: create-next-app-skripti

5.2.2 Tiedostorakenne

Skripti loi hyvän pohjan projektille. Projektiin kuuluu App Routerin app-hakemisto ja React-toteutusta vastaava static-hakemisto.

Kuviossa 14 kuvattuna projektin tiedostorakenne skriptin ajamisen jälkeen.



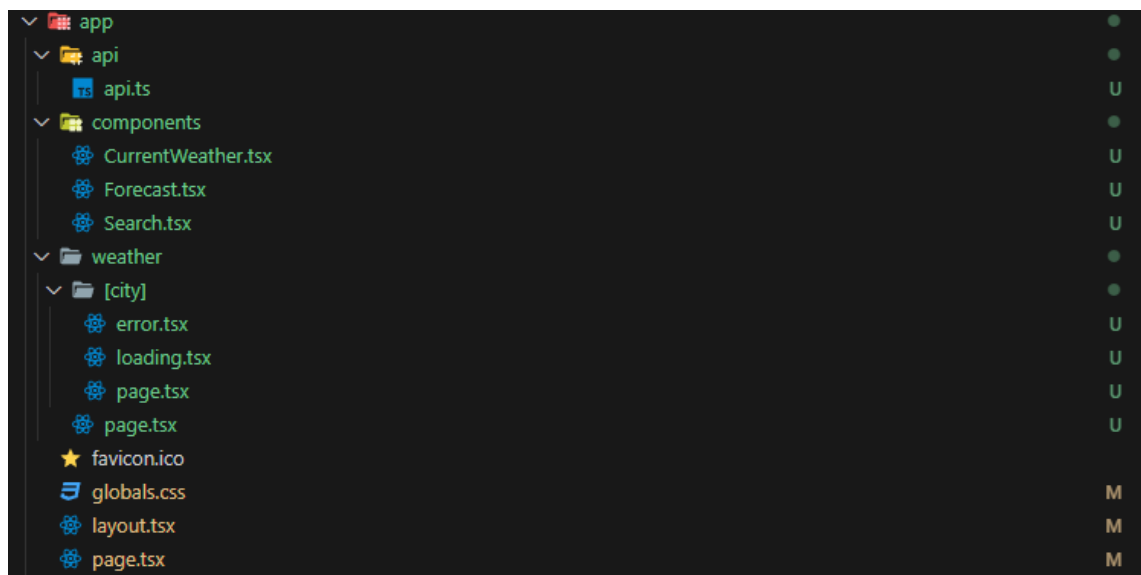
Kuvio 14: Next-toteutuksen tiedostorakenne vertaa Kuvio 6

App-hakemistoon luotiin api, components ja weather- hakemistot. Api-hakemisto sisältää React-toteutuksesta kopioitua api.ts-tiedoston. Components-hakemisto on samoin kopio React-toteutuksen components-hakemistosta, sisältäen sää, ennuste ja hakukomponentit. Rakennetta eroaa weather-hakemiston kohdalla. Weather-hakemisto sisältää page.tsx-tiedoston ja [city]-hakemiston. [city]-hakemisto kertoo Next.js:lle tämän osoitteen olevan dynaaminen. Hakemisto sisältää page.tsx, error.tsx ja loading.tsx -tiedostot kuvaamaan itse sivua, sen virheitä ja lataamistilaa.

5.2.3 Toiminnallisuuden toteutus

Toiminnallisuus saatiin suureksi osaksi kopioitua React-toteutuksen puolelta. Huomattavana erona React-toteutukseen oli se, ettei Next.js-toteutuksessa tarvittu yhtäkään ulkopuolista kirjastoa halutun toiminnallisuuden saavuttamiseksi. React-router-dom-kirjasto korvattiin Next.js:n omalla reitittimellä ja styled-components create-next-app-skriptin ehdottamalla Tailwind CSS:llä.

Kuviossa 15 kuvattu Next.js-toteutuksen app-hakemiston tiedostorakenne komponenttien ja sivujen luonnin jälkeen. Tätä rakennetta voi verrata React-toteutuksen tiedostorakenteeseen kuviossa 7.



Kuvio 15: app-hakemisto kehityksen lopussa

[city]-hakemiston page.tsx-tiedosto vastaa React-toteutuksen Weather-komponenttia. React-toteutuksesta poiketen säätiedot haetaan suoraan api.ts-tiedoston kautta, eikä erillisen hoo-kin avulla. Tämän sivun sisältö haetaan ja generoidaan palvelimella aina kun sivu ladataan. Sivun lähdekoodi kuvattuna alla kuviossa 16. Tätä koodia voi verrata kuvion 9 React-toteutuksen Weather-sivun lähdekoodiin.

```
type PageProps = {
  params: {
    city: string
  },
  props: any
}

const Page: ({ params }: PageProps) => ... = async ({params}: PageProps): Promise<Element> => {
  const data: OWMWeather | undefined = await getWeather(params.city)
  const forecastData: OWMForecast | undefined = await getForecast(params.city)

  if (!data || !forecastData) {
    return <div>Säätietojen haku epäonnistui</div>
  }

  return (
    <div>
      <Search />
      <CurrentWeather weatherData={data} />
      <Forecast forecastData={forecastData} />
    </div>
  )
}

export default Page
```

Kuvio 16: [city]/page.tsx-tiedosto

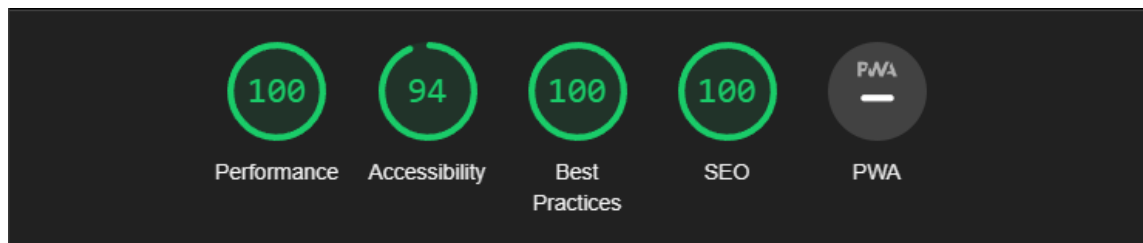
5.2.4 Testaus

Valmis Next.js sivusto paketoitiin `npm run build`-komennolla ja jaettiin `next start`-komennolla, joka käynnistää palvelimen jakamaan valmista, paketoitua sivustoa.

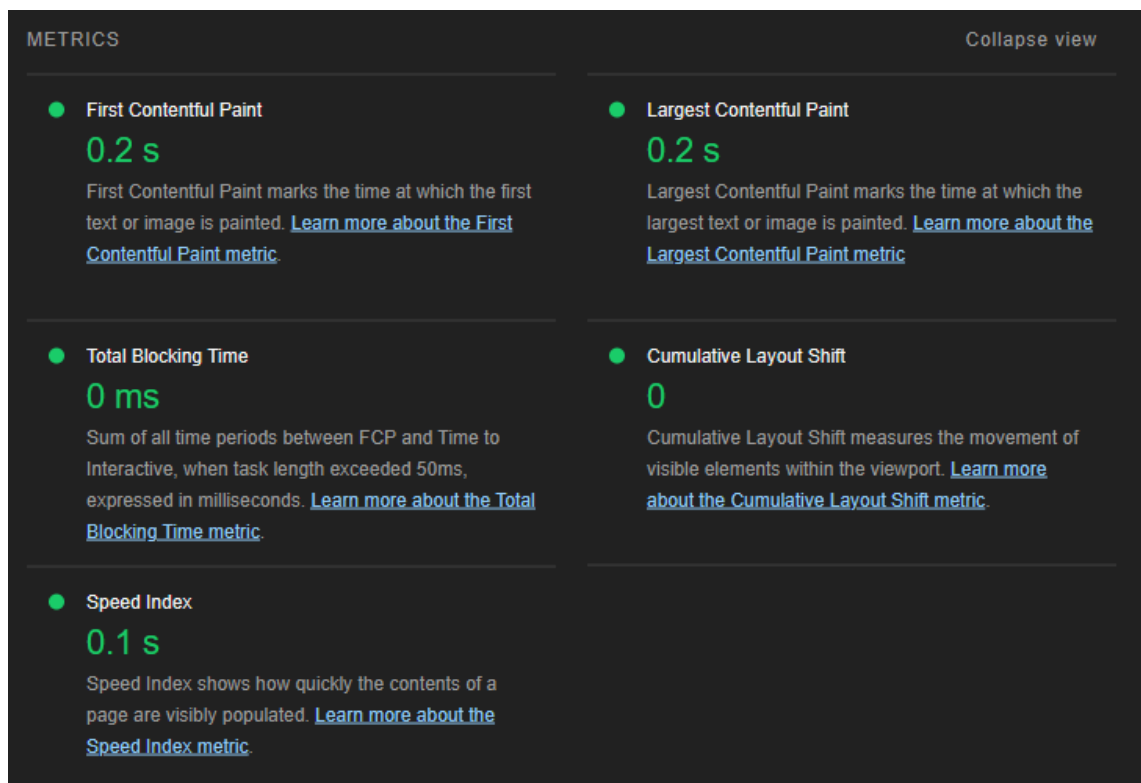
Sivustoa testattiin React-toteutuksen tavoin Edge-selaimen Lighthouse-työkalulla, oletusasetuksilla, sivulla `/weather/vantaa`.

Next.js-toteutuksen ensimmäinen sisältö latautui yhtä nopeasti kuin React-toteutuksessa, mutta suurin sisältö latautui 0.6 sekuntia nopeammin. Tämä säästö tulee Next.js:n Image-komponentin optimoinneista.

Kuvioissa 17 ja 18 kuvattuna Lighthouse-työkalun yhteenveto sivun suorituskyvystä. Näitä kuvia voi verrata React-toteutuksen yhteenvetoihin kuvioissa 10 ja 11.



Kuvio 17: Next.js-toteutuksen Lighthouse-yhteenveto

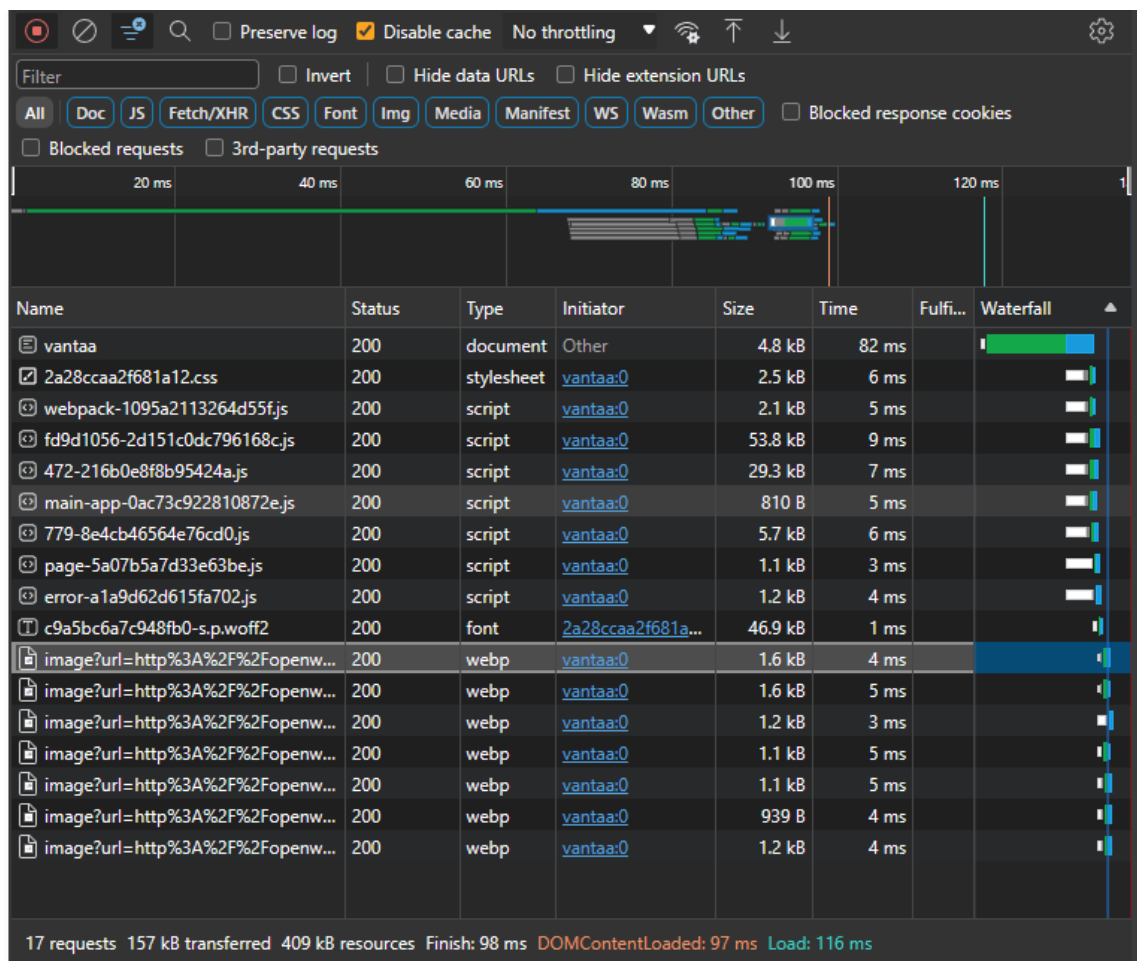


Kuvio 18: Next.js-toteutuksen suorituskyky

Kehittäjätyökalujen Verkko-välilehteä tutkittaessa huomataan, että välimuistin tyhjennyksen jälkeen ensimmäisellä latauskerralla selain lataa tietoja 157 kilotavun edestä (409 kt pakkaamattomana). Tämä ensimmäinen lataus on hieman isompi, kuin React-toteutuksessa. Toisella latauskerralla selaimeen ladataankin vain 4.8 kilotavua päihittäen React-toteutuksen 18 kilotavun lukeman. Ilman välimuistia Next.js-toteutuksessa joudutaan kuitenkin lataamaan enemmän dataa joka sivulatauksella.

Next.js-toteutuksessa tiedostot tulevat kuitenkin paremmin pakattuina, ja niiden lataus kestääkin vain noin 100 millisekuntia, kun React-toteutuksen tietojen latauksessa menee noin 300 millisekuntia.

Kuviossa 19 kuvattuna Verkko-välilehti Next.js-sivulla. Tätä kuviota voi verrata React-toteutuksen Verkko-välilehteen kuviossa 12. Huomaa kuviosta 19 puuttuvat fetch-kutsut. API-kutsut on tehty palvelimella, eikä käyttäjän laitteella, joten niitä ei näy täällä.



Kuvio 19: Next.js-toteutuksen Verkko-välilehti

6 React vs. Next.js

React ja Next.js ovat molemmat hyviä kirjastoja verkkosivustojen kehittämiseen. Next.js parantaa Reactin kehittäjäkokemusta ja suorituskykyä, mutta sen käyttämiseen tarvitaan silti React-osaamista.

Jos kehittäjältä löytyy jo valmiiksi React-osaamista ja halua kokeilla uutta, suosittelen vahvasti Next.js-kirjaston testaamista. Jo olemassa olevan React-projektin saa helposti muutettua Next.js-projektiksi netistä löytyvien oppaiden mukaan ja Nextin tuomat hyödyt näkyvät heti.

Suurille yrityksille, suuriin projekteihin Next.js on selvästi parempi valinta. Suurissa projekteissa pelkästään riippuvuuksien hallinta voi käydä raskaaksi. Next.js:ää käyttäessä ei tarvita niin montaa ulkoista pakettia toiminnallisuuden kehittämiseen, vaan Next sisältää suuren osan niistä jo valmiiksi. Myös asiakkaat kiittävät optimoidun sivuston tuomista hyödyistä.

Tiivistettynä, jos et ole käyttänyt Reactia tai Nextiä, harjoittele React ensin; jos olet käyttänyt Reactia ennen, kokeile Nextiä.

7 Yhteenveto

Tämä opinnäytetyö tarjosi vain pintaraapisun Reactin ja Next.js:n ominaisuuksiin. Kirjastot tarjoavat vielä paljon enemmän, mutta tätä opinnäytetyötä tehdessä niihin ominaisuuksiin ei ollut aikaa paneutua.

Kehitysvaiheessa huomattiin, että Next.js on Reactia ”täydellisempi paketti” ja näin kehittäjäystävällisempikin. Reactin on tosin tarkoitus olla vain riippumaton pohja kehitykselle. Tuotantokäyttöön tarkoitettuja sivuja tekisin siis itse mieluummin Next.js:llä.

Myös testausvaiheessa Next.js:n edut huomattiin. Pienellä vaivalla latausajat laskivat merkittävästi. Suuremmissa projekteissa hyödyt voidaan todennäköisesti nähdä ihan silmin, tässä toteutuksessa puhuttiin kuitenkin vain sekunnin murto-osista.

8 Jatkokehitysehdotukset

Testauksessa käytetyt toteutukset eivät olleet tarpeeksi laajoja kunnollisten tulosten saamiseksi. Huomaan näin jälkikäteen, ettei näin yksinkertainen sivusto välttämättä riitä tämän tyyppiseen suorituskykytestaukseen, eikä minulla ollut aikaa tehdä sivustosta monimutkaisempaa. Jos joku vielä jatkossa aikoo vertailla näitä kahta kirjastoa, tulisi hänen käyttää enemmän aikaa sivuston suunnitteluun ja varmistaa, että sivustosta tulee tarpeeksi laaja. Tällöin ei välttämättä tarvitse kuvata kirjastoja edes niin laajasti kuin tässä työssä.

Lähteet

Meta 2023a. Component - React. Viitattu 8.10.2023. <https://react.dev/reference/react/Component>

Meta 2023b. Your First Component - React. Viitattu 8.10.2023. <https://react.dev/learn/your-first-component>

Meta 2023c. Built-in React Hooks - React. Viitattu 8.10.2023. <https://react.dev/reference/react>

Meta 2023d. Virtual DOM and Internals. Viitattu 5.10.2023. <https://legacy.reactjs.org/docs/faq-internals.html>

Mallick, D. 2023. Virtual DOM vs Browser DOM in React.js. Medium. Viitattu 4.12.2023. https://medium.com/@darshana_18428/virtual-dom-vs-browser-dom-in-react-js-eccf466d6e8b

Osmani, A. 2017. A Pinterest Progressive Web App Performance Case Study. Medium. Viitattu 13.10.2023. <https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154>

Walton, P. 2019. First Contentful Paint | Articles | web.dev. Web.dev. Viitattu 28.11.2023. <https://web.dev/articles/fcp>

Walton, P & Pollard B. 2023. Largest Contentful Paint | Articles | web.dev. Web.dev. Viitattu 28.11.2023. <https://web.dev/articles/lcp>

Walton, P. 2023. Time to Interactive | Articles | web.dev. Web.dev. Viitattu 28.11.2023. <https://web.dev/articles/tti>

Vercel 2023a. Building Your Application: Routing | Next.js. Viitattu 16.10.2023. <https://nextjs.org/docs/app/building-your-application/routing>

Vercel 2023b. Building Your Application: Optimizing | Next.js. Viitattu 18.10.2023. <https://nextjs.org/docs/pages/building-your-application/optimizing>

Vercel 2023c. Optimizing: Images | Next.js. Viitattu 18.10.2023. <https://nextjs.org/docs/pages/building-your-application/optimizing/images>

Vercel 2023d. Components: <Head> | Next.js. Viitattu 18.10.2023. <https://nextjs.org/docs/pages/api-reference/components/head>

Vercel 2023e. Rendering: Static Site Generation (SSG) | Next.js. Viitattu 28.11.2023. <https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation>

Vercel 2023f. Rendering: Server-side Rendering (SSR) | Next.js. Viitattu 28.11.2023. <https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering>

Vercel 2023g. Routing: Loading UI and Streaming | Next.js. Viitattu 28.11.2023. <https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming#what-is-streaming>

Kuviot

Kuvio 1: Esimerkki JSX:llä luodusta komponentista (Meta 2023)	9
Kuvio 2: Esimerkki luokkakomponentista (Meta 2023a)	10
Kuvio 3: Esimerkki funktiokomponentista (Meta 2023b)	10
Kuvio 4: Esimerkki app-hakemistosta (Vercel 2023a).....	13
Kuvio 5: Uuden React-projektin luonti create-react-app:in avulla.....	16
Kuvio 6: React-projektin rakenne VS Code-editorissa	17
Kuvio 7: React-projektin src-hakemisto toteutusvaiheen loppupuolella.....	18
Kuvio 8: api.ts-tiedoston sisältöä	18
Kuvio 9: Weather-komponentti	19
Kuvio 10: React-toteutuksen Lighthouse-yhteenveto.....	20
Kuvio 11: React-toteutuksen suorituskyky	20
Kuvio 12: React-toteutuksen Verkko-välilehti	21
Kuvio 13: create-next-app-skripti	22
Kuvio 14: Next-toteutuksen tiedostorakenne vertaa Kuvio 6.....	23
Kuvio 15: app-hakemisto kehityksen lopussa	24
Kuvio 16: [city]/page.tsx-tiedosto	24
Kuvio 17: Next.js-toteutuksen Lighthouse-yhteenveto.....	25
Kuvio 18: Next.js-toteutuksen suorituskyky	26
Kuvio 19: Next.js-toteutuksen Verkko-välilehti	27