Niko Luostarinen

NEXT.JS JA TYPESCRIPT VERKKOSIVUSTON KEHITYKSESSÄ

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2022

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Tradenomi (AMK)
Tekijä	Niko Luostarinen
Työn nimi	Next.js ja Typescript verkkosivuston kehityksessä
Toimeksiantaja	Hurja Solutions Oy
Vuosi	2022
Sivut	27 sivua
Työn ohjaaja	Janne Turunen

TIIVISTELMÄ

Tämän opinnäytetyön tavoite oli perehtyä Javascriptin ja Typescriptin, sekä Next.js:n ja Reactin toimintaan ja eroavaisuuksiin. Tavoitteena oli myös selvittää, kuinka paljon Reactiin on asennettava lisää paketteja, jotta siihen saadaan samat toiminnallisuudet kuin Next.js sisältää.

Toimeksiantaja opinnäytetyölle toimi Hurja Solutions Oy, joka tuottaa verkkopalveluita, räätälöityjä ohjelmistoja, mobiiliapplikaatiota sekä lisätyn todellisuuden (AR) -sovelluksia. Yritys kokeilee aktiivisesti uusia teknologioita, ja tämä työ on osa tällaista kokeilua.

Opinnäytetyön alussa käydään läpi ohjelmointikielet Typescript sekä Javascript ja näihin liittyviä käsitteitä. Tämän jälkeen käydään läpi, mikä on ohjelmistokehys, ja esitellään näistä Next.js sekä React niiden ominaisuuksineen. Työssä tehdään blogisivusto näillä ohjelmistokehyksillä sekä ohjelmistokielillä ja tehdään vertailua näiden välillä.

Työssä sivusto tehdään ensin Next.js:n ja Typescriptin avulla käyttäen Static Site Generation, Incremental Static Regeneration ja Server Side Rendering-menetelmiä. Tämän jälkeen sivusto toteutetaan käyttäen Reactia ja Javascriptiä. Tämän jälkeen samaa sivustoa lähdetään toteuttamaan Reactilla ja tarkoituksena on käyttää samoja renderöinti menetelmiä kuin Next.js esimerkissä.

Opinnäytetyön tuloksena saadun vertailun pohjalta todetaan, että Reactissa jo perustoiminnallisuudet vaatisivat lisäpakettien asentamista. Jos käyttöön halutaan ottaa Static Site Generation-, Incremental Static Regeneration- ja Server Side Rendering -menetelmiä, on vaihtoehtoina ottaa käyttöön jokin nämä taitava ohjelmistokehys tai oman toteutuksen koodaaminen palvelin toteutuksineen.

Asiasanat: ohjelmointi, ohjelmistokehitys, Next.js, React, Javascript, Typescript

Degree title	Bachelor of Business Administration
Author	Niko Luostarinen
Thesis title	Next.js and Typescript in web development
Commissioned by	Hurja Solutions Oy
Time	2022
Pages	27 pages
Supervisor	Janne Turunen

ABSTRACT

The purpose of this thesis was to get familiar with Javascript and Typescript and with the operation and differences of Next.js and React. The goal was also to find out how many extra packages would have to be installed to gain same the features that Next.js already included.

The commissioner of the thesis was Hurja Solutions Oy, a producer of web services, customized software, mobile application, and augmented reality (AR) applications. The company is actively experimenting with new technologies and this work is part of such experimenting

The thesis started with an introduction to the programming languages Typescript and Javascript and related concepts. After that the thesis explained software framework and introduced Next.js and React with their main features. This thesis included a blog page made with these frameworks and programming languages and a comparison between them.

At first the blog page was made with Next.js and Typescript using Static Site Generation, Incremental Static Regeneration ja Server Side Rendering methods. After that the same page was made with React and Javascript. After this the same page was created with React and purpose was to use the same rendering methods as in the Nexj.js example.

The comparison of the thesis results showed that even the main functionalities needed installation of external packages. For the implementation of Static Site Generation, Incremental Static Regeneration and Server Side Rendering methods the options would be to install a framework including these or coding your own implementation with server implementations.

Keywords: programming, software development, Next.js, React, Javascript, Typescript

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on tutkia toimeksiantajan käytössä olevia Javascript ja Typescript-ohjelmointikieliä sekä React- ja Next.js ohjelmistoviitekehyksiä toisiinsa. Käytännön toteutuksessa tehdään sovellus ensin Next.js ja Typescript- yhdistelmällä, ja tämän jälkeen sama React sekä Javascript toteutuksella. Tavoitteena on siis tutkia eroja näiden kahden kokonaisuuden välillä sekä selvittää, kuinka paljon ulkoisia paketteja on asennettava Reactiin, jotta saavutetaan samat toiminnallisuudet kuin Next.js:ssä.

Työn toimeksiantajana toimii kuopiolainen ohjelmistoyritys Hurja Solutions Oy. Yritys tuottaa verkkopalveluita, räätälöityjä ohjelmistoja, mobiiliapplikaatiota sekä lisätyn todellisuuden (AR) sovelluksia ratkaisuhakuisesti ja vahvalla ammattitaidolla. Yritys kokeilee aktiivisesti uusia teknologioita, ja tämä työ on osa tällaista kokeilua.

Työ jakautuu johdannon lisäksi neljään osaan, joista ensimmäisessä (luku 2) käsitellään työssä käytettyjä ohjelmointikieliä sekä niiden ominaisuuksia. Kolmannessa luvussa käsitellään ohjelmistokehyksiä. Tämän jälkeen tutustutaan tarkemmin työssä käytettyihin Next.js- ja React-ohjelmistokehyksiin sekä niiden toiminnallisuuksiin. Neljännessä luvussa käydään läpi käytännön osuus, jossa tehdään yksinkertainen blogisivusto aiemmin mainituilla ohjelmistokehyksillä, sekä ohjelmointikielillä.

Viidennessä luvussa käydään läpi tuloksia käytännön toteutuksesta ja esitellään saatuja havaintoja. Viimeinen luku on päätäntö, jossa käydään läpi työtä ja raportin tekemistä. Tässä osassa käydään läpi kirjoittajan omia ajatuksia aiheesta sekä kirjoittamisprosessista sekä arvioidaan tulosten luotettavuutta sekä niiden hyödyntämistä työn toimeksiantajan näkökulmasta.

2 OHJELMOINTIKIELET

Tässä luvussa esitellään ohjelmointikielet Javascript sekä Typescript, sekä käydään läpi niiden ominaisuuksia ja erovaisuuksia toisiinsa nähden.

2.1 Javascript

Javascript Netscapen vuonna 1995 kehittämä skriptikieli. Alun perin Javascript tunnettiin nimellä Mocha, mutta nopeasti tuli tunnetuksi nimellä LiveScript ja lopulta myöhemmin nimi muuttui Javascriptiksi. (Dickson 2022). Nimestään huolimatta Javascript ei perustu Java-ohjelmointikieleen. Sen syntaksi on kuitenkin tehty samanoloiseksi kuin Java ja C++. Tällä on haluttu vähentää uusien ohjelmointi konseptien omaksumista. (MDN Web Docs. 2022.) Javan lisääminen Javascriptin nimeen on todettu olleen markkinointitempaus, jotta kielelle saataisiin hyväksyntää. (Dickson 2022.)

Javascript on kevyt, ajonaikana käännettävä ohjelmointikieli, jonka tunnetuin käyttötapa on web-kehitys. Alkuperäisesti Javascriptillä oli tarkoitus tehdä lyhyitä koodeja web-sivuille ja niiden pituus oli harvoin muutamia kymmeniä rivejä pidempiä. Ajan myötä Javascriptin suosio kasvoi ja sitä alettiin käyttää monimutkaisempiin toteutuksiin. (Typescript 2022.)

Internetselaimien kehittäjät vastasivat kasvaneeseen Javascriptin suosioon parantamalla sen toimivuutta ja tehokkuutta internetselaimissa. Tästä seurasi, että kehittäjät ryhtyivät käyttämään Javascriptiä entistäkin enemmän. Javascript suosion kasvaessa levisi sen käyttökohteet myös internetselaimien ulkopuolelle kuten palvelimiin. (Typescript 2022.)

Javascript on prototyyppipohjainen ohjelmointikieli, joka tarkoittaa, että luokkia ei ole tarkasti määritelty. Yksinkertaistetusti objekteja voidaan luoda määrittämättä niiden luokkaa. Se on yksi käytetyimmistä ohjelmistokielistä web-kehityksessä. Sen avulla voidaan näyttää muutakin kuin staattista sisältöä web-sovelluksessa. Javascriptiä käytetään myös websovellusten ulkopuolisessa käytössä. (MDN Web Docs. 2022.)

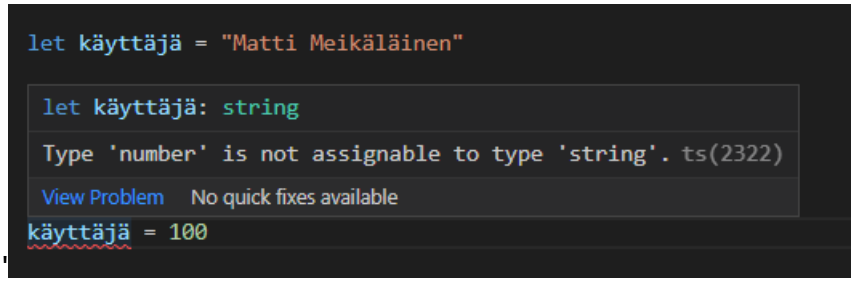
2.2 Typescript

Typescript on Microsoftin kehittämä ohjelmointikieli, joka on Javascriptin yli-joukko, eli se tarjoaa kaikki Javascript-ohjelmointikielen ominaisuudet. Siinä missä Javascript on heikosti tyypitetty ohjelmointikieli, tarjoaa Typescript mahdollisuuden muuttujien tarkempaa tyypittämiseen. (Cleverism 2022.) Tarkemman tyypityksen tarkoituksen on ehkäistä ajonaikaisia virheitä korostamalla niitä jo koodia kirjoittaessa. Typescript on oliopohjainen ohjelmistokieli siinä missä Javascript on prototyyppipohjainen. Typescript tulee myös kääntää Javascriptiksi ennekuin sitä voidaan suorittaa esimerkiksi internet-selaimessa. (GeeksForGeeks 2022.)

Koska Typescript perustuu Javascriptiin on Javascriptin syntaksi suoraan soveltuva Typescriptiin, voi aiemman Javascript koodin voi siirtää suoraan Typescriptiin ja se ei aiheuta virhettä. Kuitenkin tyypityksestä johtuen virheitä voi esiintyä, jos muuttujatyyppiä käsitellään väärin, esimerkiksi numeroa, yritetään asettaa muuttujaan, joka on kirjaimia. Typescript ilmaisee virheet jo kehityksen aikana sekä osoittaa ne myös käännöksen aikana esimerkiksi aiemmin mainittu muuttujan tyypin asettamisen väärään tyyppiin. (Typescript 2022.)

Tyypittäminen

Koska Typescript sisältää Javascriptin, luo Typescript automaattisesti tyypin muuttujalle monissa tapauksissa. Kun esimerkiksi luodaan uusi muuttuja, jolle annetaan arvo mutta ei tyyppiä, tekee Typescript kyseiselle muuttujalle arvon mukaisen tyypin. Javascript itsessään sisältää muutamia tyyppejä kuten esimerkiksi string ja number, mutta ei tarkista annetaanko näille muuttujille tarkoituksenmukainen arvo. Kuvassa 1 on nähtävissä, kuinka Typescriptin tyypin tarkistus huomaa virheellisen arvon.



Kuva 1. Typescript ilmoittaa, kun muuttujalle yritetään antaa väärän tyyppistä uutta arvoa

Yksi tapa tyypittää on luoda interface eli eräänlainen malli. Sen avulla luodaan malli, joka kertoo, millaista tyyppiä muuttujan pitäisi olla tai millainen objektin sisältö on. Kuvassa 2 on esimerkki, jossa on luotu interface ja tämän pohjalta luotu uusi tyypitetty objekti:

```
interface Kayttaja {
  id:number;
  nimi:string;
  tunnus:string
}

const Home: NextPage = () => {

  const uusiKayttaja: Kayttaja = {
    id:10,
    nimi:"Matti Meikäläinen",
    tunnus:"02A"
  }
```

Kuva 2. Interface ja sen avulla tyypitetty objekti

Kun koodi lopulta käännetään, tarkistaa Typescript tässä yhteydessä tyypityksen ja kääntää lopuksi koodin Javascriptiksi, jossa ei tyypityksiä enää ole. Typescript itsessään ei koskaan muuta kirjoitetun koodin tai ohjelman toimivuutta: Typescript tarkistaa vaan koodin tyypityksen, mutta ei ota kantaa koodin toimivuuteen. (Typescript 2022.)

3 OHJELMISTOKEHYKSET

Ohjelmistokehys (engl. framework) on koodikirjasto, joka tarjoaa ohjelmoijalle ennalta kirjoitettua koodi rutiininomaisiin tehtäviin. Modernin web-sovelluksen kehittäminen on mahdollista myös ilman mitään ohjelmistokehystä, mutta se olisi työläämpää: jokainen ominaisuus tulisi kirjoittaa aivan alusta, kun taas ohjelmistokehys sisältää näihin tarpeisiin valmiita pohjia. Ohjelmistokehyksillä on omia sääntöjä sekä suosituksia, joiden tarkoituksena on helpottaa ja tehostaa ohjelmistokehitystä. Se myös luo rajat, millaiseen käyttöön ohjelmistokehys soveltuu ja millaiseen ei. (Khan 2021.)

Ohjelmistokehityksessä puhutaan myös kirjastoista ja niiden eroista ohjelmistokehyksiin. Näiden välinen raja on häilyvä, mutta eroja on: ohjelmistokehys antaa muodon koko sovellukselle, kun taas kirjasto on kokoelma ennalta kirjoitettua koodia, jota otetaan käyttöön tarpeen vaatiessa.(Code Institute 2022). Yksinkertaistetusti voidaan ajatella, että ohjelmistokehys on kokoelma ohjelmistokirjastoja. Ohjelmistokirjaston ja ohjelmistokehyksen välillä on kuitenkin ero, kuinka itse kirjoitettua koodia kutsutaan. (InterviewBit 2021.) Tätä havainnollistetaan kuvassa 3.

Kuva 3. Ohjelmistokirjaston ja ohjelmistokehyksen kutsumisen ero

Kun käytetään ohjelmistokehystä, kutsuu ohjelmistokehys käyttäjän itse kirjoittamaa koodia. Ohjelmistokirjaston tapauksessa käyttäjän kirjoittama koodi kutsuu ohjelmistokirjastoa. (InterviewBit 2021.)

Jotta ohjelmistokehyksiä ja ohjelmistokirjastoja voidaan käyttää, pitää ne tuoda osaksi projektia. Esimerkiksi ohjelmistokirjasto voidaan ottaa käyttöön Content Delivery Networkin (CDN) avulla. CDN on ryhmä palvelimia, jotka on hajautettu useisiin kohteisiin ympäri maapallon. (MDN Web Docs 2022b.)Tuolloin kirjaston toiminnallisuudet saadaan käyttöön lisäämällä tähän sopivalla merkinnällä. (MDN Web Docs 2022c.)

Toinen tapa tuoda ja ottaa käyttöön ohjelmistokirjastoja sekä ohjelmistokehyksiä on käyttää pakettien hallintaa (package manager). Javascriptin yhteydessä on käytettävissä tähän tarkoitukseen oleva ohjelma esimerkiksi npm. Tällöin pakettien hallinta hakee tarvittavat paketit ja asentaa ne käyttäjän tietokoneelle, jonka jälkeen ne ovat käytettävissä kyseisessä projektissa. (MDN Web Docs 2022c.)

3.1 React.js

React.js tai yleisesti React on Facebookin kehittämä ilmainen ja vapaaseen lähdekoodiin perustuva javascript-kirjasto. Käyttöliittymää tehdessä koodi pilkotaan omiin komponentteihinsa, jotka yhdessä muodostavat kokonaisuuden. React tehty JSX:llä, joka on yhdistelmä Javascriptiä ja XML:ää (React 2022c). Reactissa elementtejä luodaan JSX:llä, jotka käännetään Javascriptillä websovelluksessa soveltuvaan muotoon (Code Institute 2022). React sisältää tuen myös Typescriptille (Create React App 2022).

Reactin vahvuuksia on nopeus ja tarkemmin määriteltynä siten, että yksittäinen kehittäjä voi kirjoittaa/muokata omaa komponenttiansa ja ei aiheuta muutoksia kokonaisuuteen. Myöskin komponenttien vuoksi React on joustava sekä mahdollistaa komponenttien uusiokäytön: jokaisen ei tarvitse keksiä samaa pyörää uudelleen ja uudelleen. (Pisuwala 2022.)

Heikkouksiksi Reactissa voidaan laskea sen nopea kehitystahti. Asiat muuttuvat nopeasti mikä edellyttää jatkuvasti uuden sisäistämistä. Myös dokumentaatiot vanhenevat samaa tahtia. React tarjoaa myös pelkästään käyttöliittymän ulkoasuun liittyvät asiat ja muut toiminnallisuudet on hankittava muista kirjastoista esimerkiksi reititin. (JavaTpoint 2022.)

Komponentit

Reactin toiminta perustuu keskeisesti komponentteihin, joita voidaan luoda kahdella tapaa: funktionaalipohjaisesti tai luokkapohjaisesti. Itse Reactin kannalta ei ole väliä kummalla tyylillä komponentteja kirjoitetaan, koska sen näkökulmasta ne ovat yhtä lailla toimivia. (React 2022a.) Kuvassa 4 on esimerkki funktionaalisesta komponentista.

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

const FunktionaalinenEsimerkki = () => {

const [paakayttaja, setPaaKayttaja] = useState(false)

  return (
    <div>
     {
      paakayttaja
      ?
      <h1>Terve Pääkäyttäjä</h1>
      :
      <h1>Terve muu käyttäjä</h1>
     }
    </div>
  )
}

export default FunktionaalinenEsimerkki
```

Kuva 4.Funktionaalinen komponentti, jossa ehdollinen sisältö

Reactissa on myös mahdollista näyttää komponentteja ehdollisesti. Tämä voidaan toteuttaa perinteisellä If-lauseella tai ?:-operaattorilla. Kuva 4 sisältää esimerkin tästä. Tällöin komponentin ulkoasu riippuu määritellystä muuttujasta riippuen, esimerkiksi onko käyttäjä kirjautunut sisälle vai onko käyttäjä kirjautumaton (React 2022b). Reactiin on sisäänrakennettu toiminallisuuksia myös tapahtumien käsittelyyn esimerkiksi napin painallukseen, joita voidaan hyödyntää osana komponenttina. Myös arrayn eli taulukoiden läpikäymiseen ja niiden sisällön esittämiseen (React 2022d) sekä lomakkeiden tietojen käsittelyyn tarjoaa React omat toiminallisuutensa (React 2022c).

3.2 Next.js

Next.js on Reactin avoimen lähdekoodin ohjelmistokehys. Next.js:n avulla voidaan luoda nopeita ja hakukoneoptimoitua web-sovelluksia ilman ulkoisia paketteja tai riippuvuuksia. Koska Next.js on Reactin ohjelmistokehys, sisältää se myös Reactin ominaisuudet (GeeksForGeeks 2022).

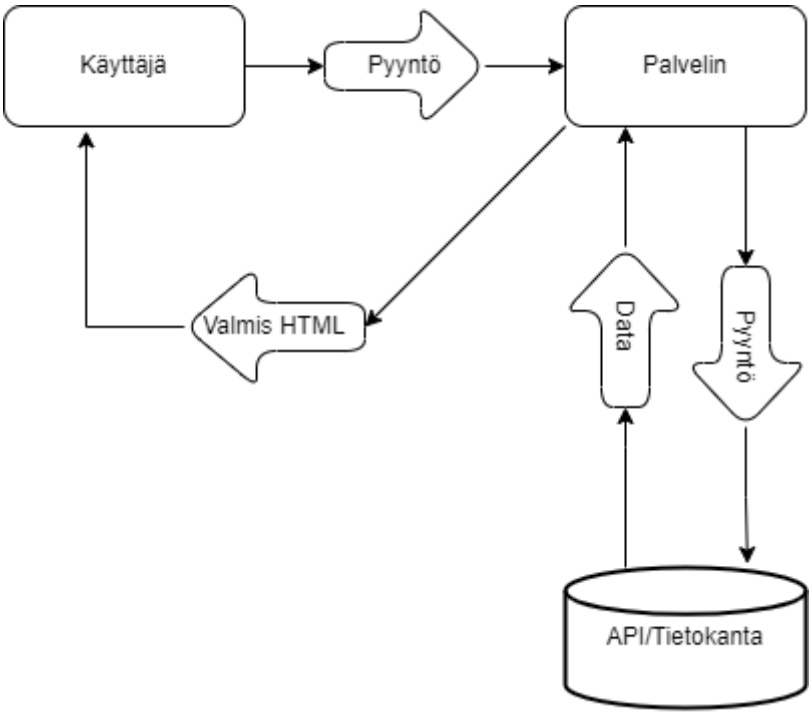
Next.js:n ominaisuuksia ovat sen valmiit komponentit, sisään rakennettu router eli ”reititin” jonka avulla sivuja voidaan luoda laittamalla ne ennalta määritettyyn kansioon, Palvelinpuolen renderöinnin, CSS- ja SASS- tuet ja dynaamiset komponentit. (Navdeep 2022). Yhtenä Next.js:n valttikorttina on pidetty sen palvelinpuolen renderöintiä. Siinä missä React suorittaa renderöinnin käyttäjänpuolella, suorittaa Next.js saman palvelinpuolella joko vaihtoehtoisesti palvelimella jokaiselle pyynnölle erikseen ja tarjoaa valmiin sivun tai käyttää staattista näkymää sivusta, joka renderöidään valmiiksi projektin kääntämisen aikana (Vercel 2022c). Halutessaan Next tukee myös käyttäjäpuolen renderöintiä (Vercel 2022c).

Renderöinti

Välttämätön osa ohjelmistokehitystä on muuttaa kirjoitettu koodi HTML:ksi ja näyttää käyttäjälle sovelluksen käyttöliittymä. Tätä kutsutaan renderöinniksi. Renderöinti voi tapahtua joko käyttäjän päässä eli sivu renderöidään kokonaisuudessaan käyttäjän internetselaimen näkymässä. Toisena vaihtoehtona on renderöidä sivu jo palvelinpuolella. (Vercel 2022a)

Server-Side Rendering

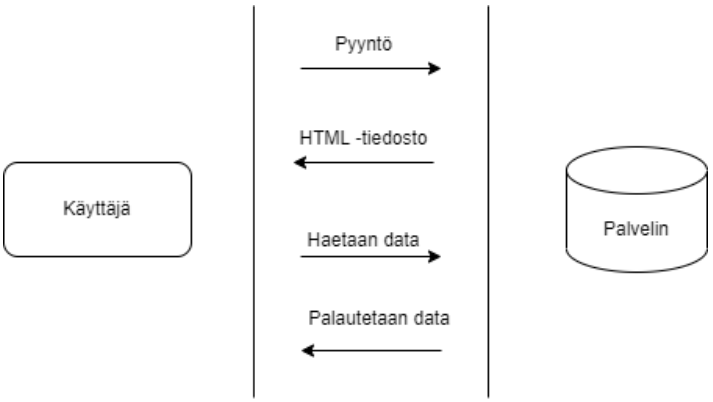
Server-Side Rendering (SSR) on menetelmä, jossa pyydetyn sivun HTML luodaan palvelimella jokaisella pyynnöllä. Palvelin lähettää valmiin sivun näytettäväksi käyttäjälle, koska kaikki tarvittava on jo ladattu palvelimella. Tätä on havainnollistettu kuvassa 5. Next.js:n dokumentaatioissa kuitenkin suositellaan välttämään palvelinpuolen renderöintiä, koska sivu luodaan jokaisen latauspyynnön yhteydessä ja voi näin vaikuttaa suorituskykyyn heikentävästi. Kuitenkin jos sivulla on usein päivittyvää sisältöä, SSR on silloin soveltuva menetelmä (Vercel 2022b).



Kuva 5. Server-Side Rendering

Client-side Rendering

Client-side rendering (CSR) eli käyttäjäpuolen renderöinti on menetelmä, jossa vain HTML renderöidään palvelimella, mutta kaikki muu, esimerkiksi sisällön haku suoritetaan Javascriptillä rajapinnasta vasta, kun sivu on ladattu käyttäjän selaimessa. (Patterns 2022.) Tätä havainnollistetaan kuvassa 6.

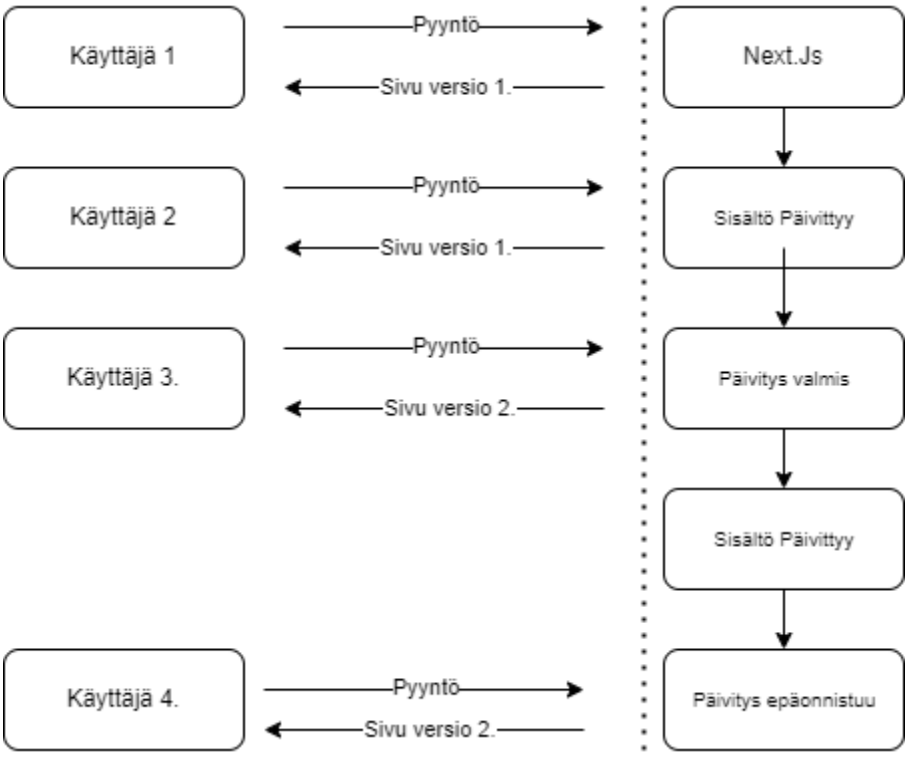


Kuva 6. CSR

Static Site Generation

Static Site Generation -menetelmällä (SSG) sivun HTML luodaan myös palvelimella, mutta poiketen SSR-menetelmästä, itse sivu ei pyöri palvelimella vaan luodaan valmiiksi kääntämisen yhteydessä. (Vercel 2022a). Tiedostoa voidaan käyttää jokaisen latauspyynnön yhteydessä uudestaan ja tallentaa välimuistiin. Menetelmän kääntöpuolena on, että se ei sovellu sivuille, joissa on usein päivittyvää sisältöä. Next sisältää metodit, joilla tietoa voidaan hakea ulkoisista rajapinnoista myös SSG-menetelmällä luoduille sivuille. (Vercel 2022c).

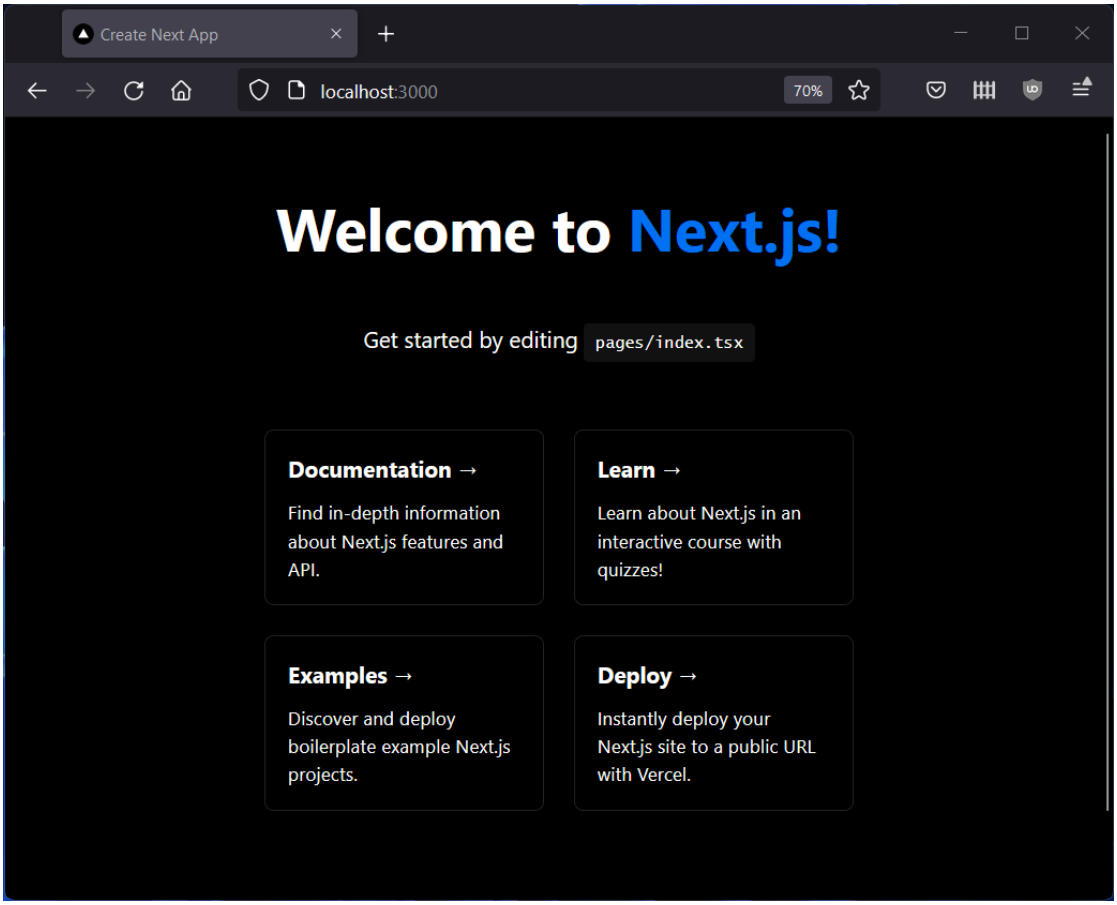
Next mahdollistaa sisällön päivittämisen Incremental Static Regeneration -menetelmän (ISR) avulla. ISR-menetelmällä yksittäistä sivua voidaan päivittää, ilman että koko sivustoa tarvitsee kääntää uudelleen. Tällöin sivusta näytetään välimuistiin tallennettu versio, joka päivittyy halutun syklin välein. Kun sisällön päivitys tapahtuu, näytetään tuona aikana käyttäjällä vanha versio välimuistista. Syklin päätyttyä näytetään seuraavalla pyynnöllä päivitetty sivu tai jos päivitys on epäonnistunut, näytetään vanha versio epäonnistuneen sijaan. (Vercel 2022b). Tätä prosessia on havainnollistettu kuvassa 7.



Kuva 7.Inrcremental Static Regeneration

4 KÄYTÄNNÖN VERTAILU

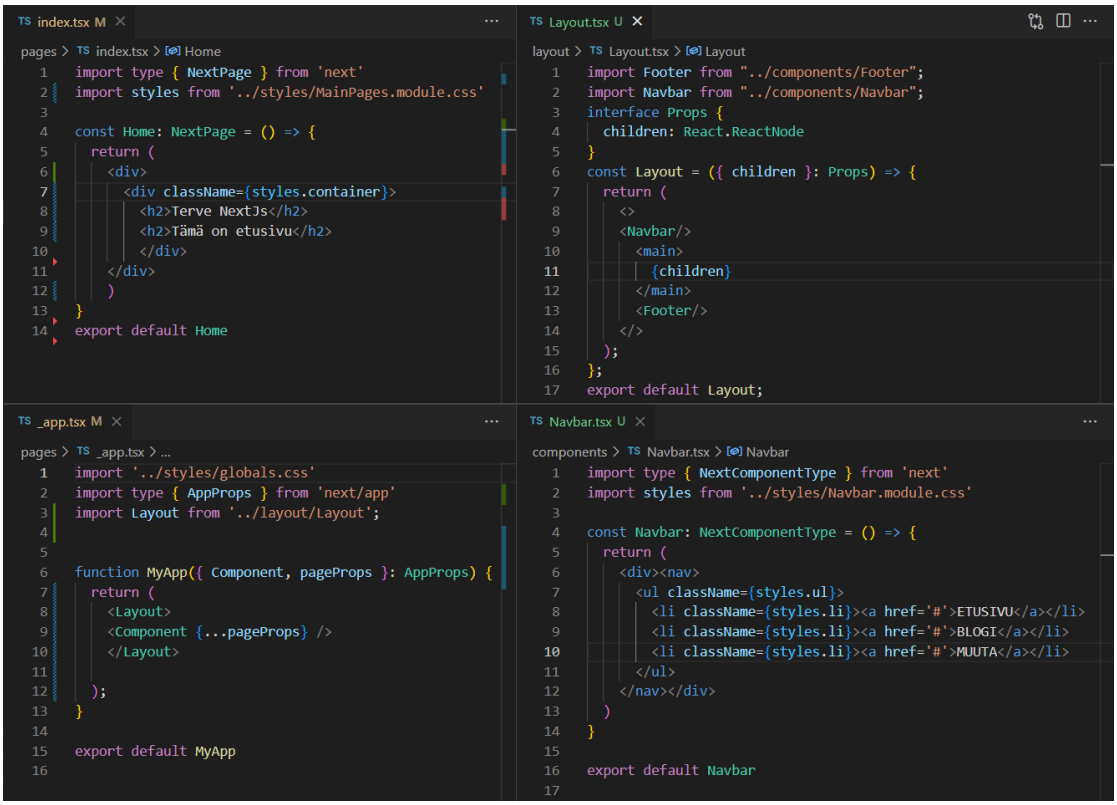
Käytännön vertailun osuudessa käydään läpi Next.js:n ja Reactin eroavaisuuksia käytännön toteutuksessa. Projektissa tehdään pieni blogisivusto, johon haetaan sisältöä ulkoisesta API:sta. Aloitin projektin alustamalla uuden Next.js ympäristön npm:ää käyttäen komennolla : npx create-next-app@latest -typescript. Tämän jälkeen npm hakee projektia varten tarvittavat paketit, jonka jälkeen Next.js sovellus on valmis kehityksen aloittamiseen. Typescript asennetaan osaksi projektia tässä yhteydessä. Tämän jälkeen käynnistin kehitysympäristön komennolla: npm run dev. Tämän jälkeen on nähtävissä Next.js:n luoma oletussivu.



Kuva 8. Next.js-oletussivu

Kuvassa 8 näkyy Next:in oletusetusivu, jonka sisällön seuraavaksi poistin ja ryhdyin muokkaamaan haluamakseni.

Tein hyvin pelkistetyn ulkoasun, jossa on valikko ylhäällä ja footer-elementti alhaalla. Nämä ovat omina komponentteinansa Layout-komponentissa, joka lisätään osaksi _app.tsx tiedostoa. Tätä havainnollistetaan kuvassa 9.

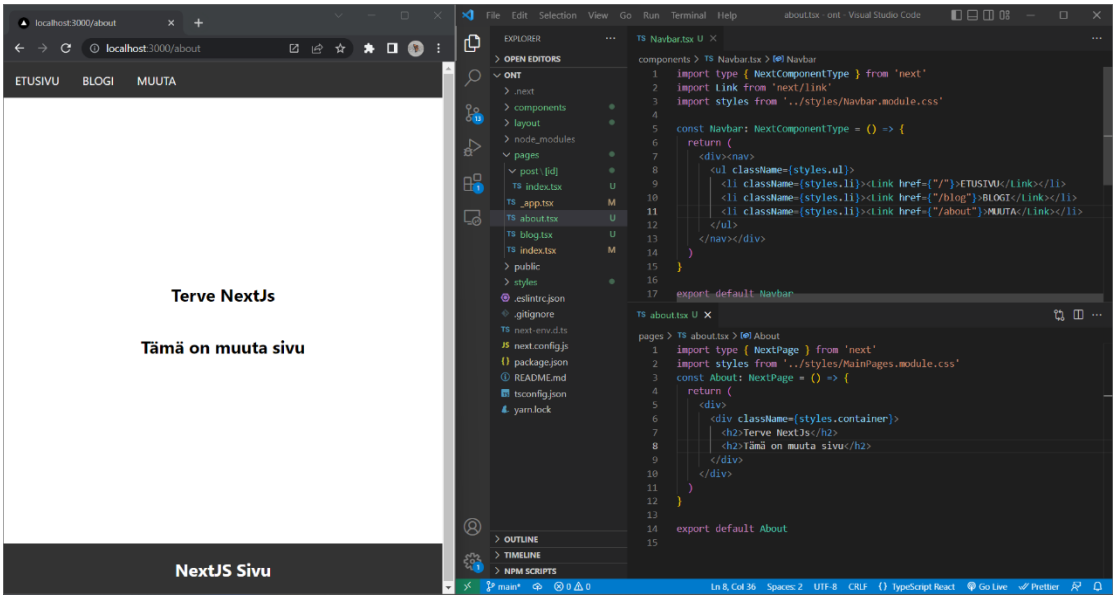


Kuva 9. Layout-komponentin sisältö, sen sijoittaminen osaksi näkymää

Layout-komponentin avulla jokaisella sivulle ei tarvitse erikseen lisätä valikkoa ja footer-elementtiä, vaan nämä sisältyvät valmiiksi jokaiselle sivulle. Kuvassa 10. näkyy sivun perusrunko, jota tässä projektissa käytetään.

Kuva 10. Sivun ulkoasu

Seuraavaksi loin projektiin sivut blog ja about. Kun sivut sijoitetaan Next.js:ssä kansioon Pages, voidaan käyttää sisäänrakennettua reititintä: Next sisältää valmiin komponentin <Link>, jonka luodun linkin avulla voidaan siirtyä sivulta toiselle.



Kuva 11.Pages-kansion sisältö, <Link>-komponentin lisääminen valikkoon

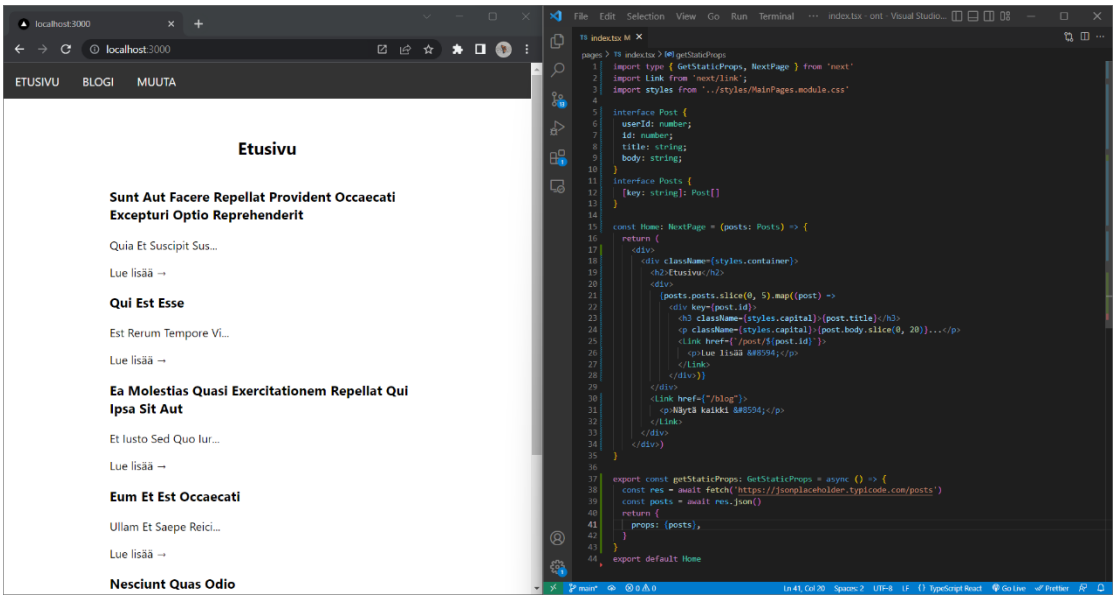
Kun perusrakenne on olemassa ja sivujen välillä liikkuminen onnistuu, on seuraava vaihe hakea sivuille sisältöä. Etusivulla on tarkoituksena näyttää blogikirjoitusten otsikot ja linkki, josta pääsee lukemaan koko kirjoituksen. Etusivu on staattinen sivu, joka renderöidään SSG-menetelmällä. Staattiselle sivulle on mahdollista hakea sisältä ulkoisesta rajapinnasta getStaticProps-funktiolla, josta on esimerkki kuvassa 12.

```
37  export const getStaticProps: GetStaticProps = async () => {
38    const res = await fetch('https://jsonplaceholder.typicode.com/posts')
39    const posts = await res.json()
40    return {
41      props: {posts},
42    }
43  }
44  export default Home
```

Kuva 12. Funktion getStaticProps-esimerkki

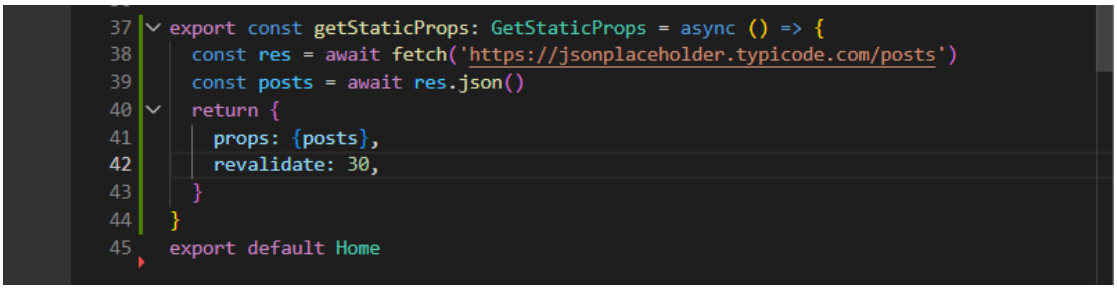
Funktio hakee rajapinnasta posts-tiedot, jotka välitetään eteenpäin sivun käytettäväksi ja käyttäjälle esitettäväksi.

Rajapinnasta saatavia tietoja varten loin myös Typescriptin interfacen, jolla tyypitin rajapinnasta saatavat tiedot. Tämä on havainnollistettu kuvassa 13. joka sisältää myös koko koodin, sekä kuvakaappauksen sivuston ulkoasusta.



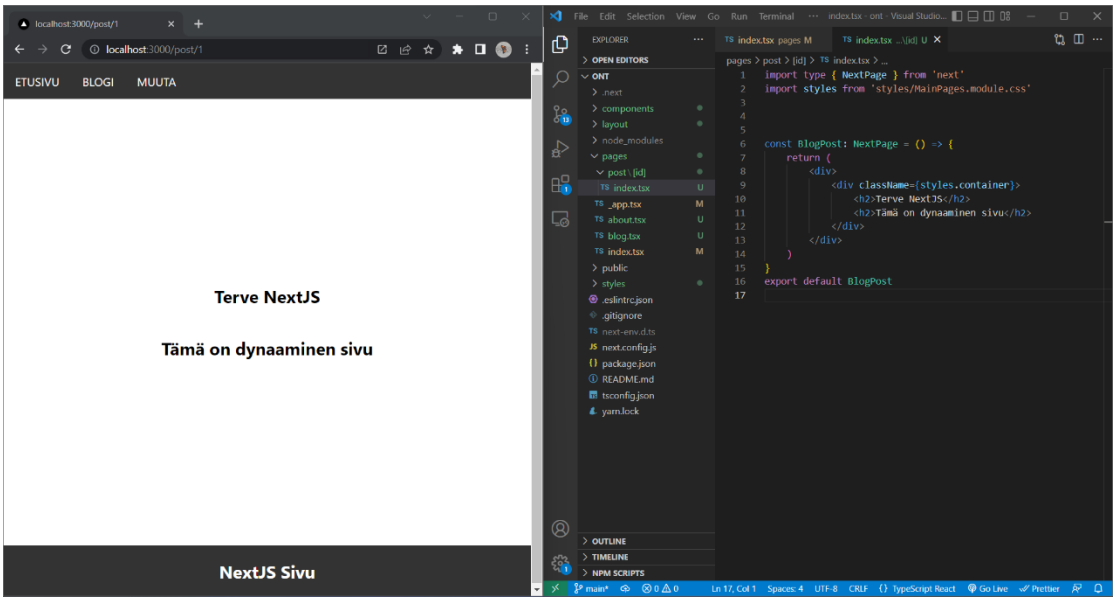
Kuva 13. Etusivun ulkoasu ja koodi

Koska etusivu esittää 5 tuoreinta blogikirjoitusta, pitäisi sisältöä voida päivittää: tämä saadaan ratkaistua ottamalla käyttöön ISR, jolloin sisällöt voidaan päivittää tuoreimpiin ilman, että koko sivustoa täytyy kääntää uudelleen. Tällöin getStaticProps-funktioon lisätään revalidate. Tästä esimerkki kuvassa 14.



Kuva 14. ISR:n lisääminen

Etusivujen linkit eivät johda mihinkään, ja koska sisältö tulee muuttumaan, kun kirjoituksia tulee lisää, tulee linkit muodostaa dynaamisesti. Next sisältää tuen dynaamisille linkityksille. Tekemällä pages-kansioon uuden kansion, tässä tapauksessa post ja sen alle vielä uusi kansio [id]. Tähän viimeiseen kansioon luodaan sivu tavalliseen tapaan. Näin linkkejä voidaan muodostaa /post/[id]-periaatteella. Näitä havainnollistetaan kuvassa 15.



Kuva 15. Esimerkki dynaamisen linkityksen kansio rakenteesta sekä siitä kuinka URL muodostuu selaimen osoitekenttään

Linkityksen lisäksi halutaan sivulla näyttää myös sisältöä. Tässä tapauksessa päädyin käyttämään post/[id]-renderöintiin SSR-menetelmää. Tällöin on mahdollista käyttää Next:in funktiota getServerSideProps, joka mahdollistaa tietojen hakemisen sivulle esimerkiksi ulkoisesta rajapinnasta. Tässä tapauksessa tietojen haku suoritetaan jokaisen pyynnön kohdalla erikseen. Tätä havainnollistetaan kuvassa. Lisäämällä parametrin notFound palautetaan käyttäjälle 404-sivu, mikäli blogi postausta ei löydy.

```
export const getServerSideProps: GetServerSideProps = async (context) => {
    const id = context.query.id;
    const res = await fetch(`https://jsonplaceholder.typicode.com/posts?id=${id}`)
    const post = await res.json()
    if (!post || !post.length) {
        return {
            notFound: true,
        }
    }
    return {
        props: { post },
    }
}

export default BlogPost
```

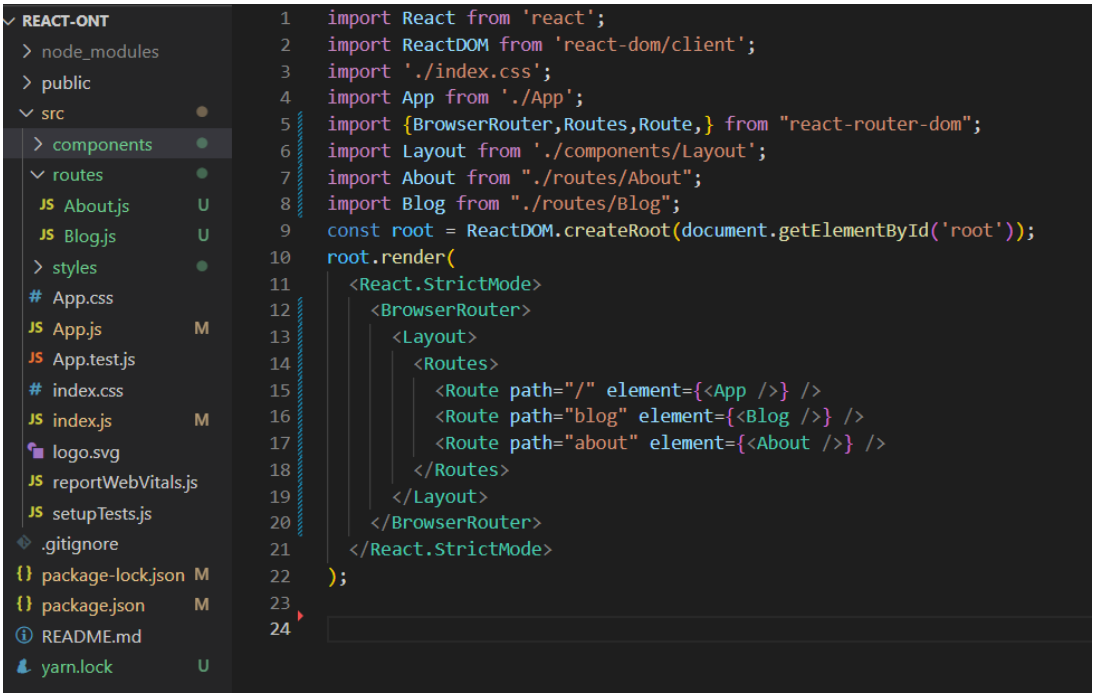
Kuva 16.getServerSideProps ja notFound

Funktio tekee pyynnön rajapintaan ja mikäli pyyntöä vastaava tieto löytyy, palautetaan tieto näytettäväksi sivulle. Tässä tapauksessa, mikäli id:tä vastaava

kirjoitus löytyy, näytetään tämä sisältö käyttäjällä. Tämän jälkeen sivustolla on halutut ominaisuudet käytössä.

React

Alustin uuden React projektin npx create-react-app-komennolla, jolla haetaan projektia varten tarvittavat paketit, jonka jälkeen projekti on valmis kehityksen aloittamiseen. Tämän jälkeen tein ulkoasun vastaamaan Next-projektissa nähtyä ulkoasua sekä loin tarvittavat sivut. Seuraavana aloin valmistella linkkejä sivujen välille. Koska React ei sisällä itsessään reititintä, tulee se asentaa npm-pakettina projektiin. Komennolla npm i react-router-dom haetaan tarvittavat paketit osaksi projektia. Tämän jälkeen vaadittavat komponenti tuodaan osaksi projektia. Nämä komponenti lisätään index.js-tiedostoon, joka on projektin juuressa. Tästä esimerkki on kuvassa 17.



Kuva 17. React router ja sen käyttöönotto

Jokainen reitti ja sitä vastaava komponentti on siis määriteltävä erikseen. Seuraavaksi tuli tarve tehdä dynaamiset linkitykset, jotta kirjoitusten sisällöt saadaan näkymään halutusti. Tätä varten luodaan uusi route eli reitti, jolla määritellään dynaamiset linkitykset. Samassa yhteydessä luodaan myös virheenkäsittely sille, jos käyttäjä menee sellaiseen reittiin, jota ei ole olemassa. Tästä esimerkki on kuvassa 18.

```
<Route path="post" element={<Post />} >
  <Route path=":id" element={<Post />} />
</Route>
<Route
  path="*"
  element={
    <div className={styles.container}>
      <p>404</p>
      <p>Sivua ei löydy</p>
    </div>
  }
/>
```

Kuva 18. Reitti dynaamiselle linkitykselle

Blogkirjoitukset tulevat ulkoisesta palvelusta, joten ne täytyy hakea näytettäväksi käyttäjällä. Ulkoisista rajapinnoista hakemiseen asensin Axios-paketin komennolla npm i axios. Axioksella haettiin Blogi-sivulle kaikki kirjoitukset nähtäväksi. Tätä havainnollistettu kuvassa 19.

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { Link } from "react-router-dom";
import styles from '../styles/MainPages.module.css'

const Blog = () => {
  const [posts, setPosts] = useState([]);
  useEffect(() => {
    const fetch = async () => {
      try {
        const { data } = await axios.get("https://jsonplaceholder.typicode.com/posts");
        setPosts(data);
      } catch (err) {
        console.error(err);
      }
    };
    fetch();
  }, []);
  return (
    <div>
      <div className={styles.container}>
        <h2>Blogi</h2>
        <div>
          {posts.map((post) =>
            <div key={post.id}>
              <h3 className={styles.capital}>{post.title}</h3>
              <p className={styles.capital}>{post.body.slice(1, 20)}...</p>
              <Link to={`/post/${post.id}`}>
                <p>Lue lisää &#8594;</p>
              </Link>
            </div>)}
        </div>
      </div>
    </div>
  )
}
export default Blog
```

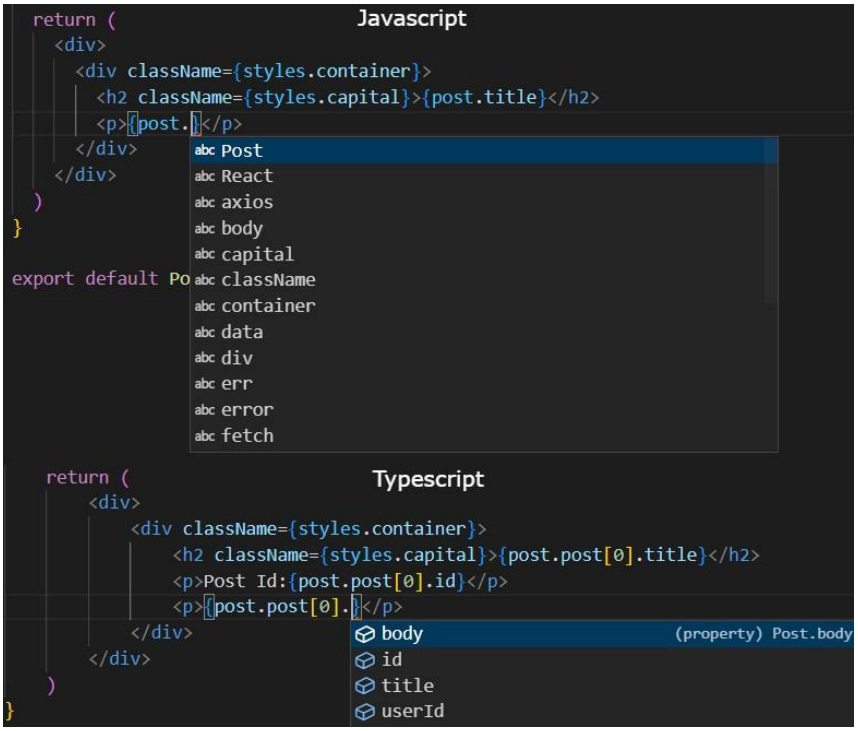
Kuva 19. Sisällön haku Axioksella

Kirjoituksen linkki johtaa /post/${post.id} -sivulle, jossa post.id:tä vastaava sisältö haetaan axioksella. Kirjoituksen id haetaan käyttämällä Reactin useParams()-funktiota. Tästä esimerkki on kuvassa 20.

```
import React, { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import styles from '../styles/MainPages.module.css'
import axios from "axios";

const Post = () => {
  const { id } = useParams();
  const [post, setPost] = useState({});

  useEffect(() => {
    const fetch = async () => {
      try {
        const { data } = await axios.get(`https://jsonplaceholder.typicode.com/posts/${id}`);
        setPost(data);
      } catch (err) {
        const error = {title: "404", body: "Sivua ei löydy"}
        setPost(error)
      }
    };
    fetch();
  }, []);

  return (
    <div>
      <div className={styles.container}>
        <h2 className={styles.capital}>{post.title}</h2>
        <p>{post.body}</p>
      </div>
    </div>
  )
}

export default Post
```

Kuva 20. Post.id:tä vastaava sivu ja id:n hakeminen useParams()-funktiolla

Nyt React-projektilla on samat perustoiminnallisuudet kuin Next-projektilla, mutta React projektiin on pitänyt lisätä paketteja ulkoisista lähteistä alustuksen jälkeen, kun taas Next on sisältänyt kaiken tarvittavan yhdellä alustuksella. Tässä yhteydessä on myös hyvä verrata mitä eroa Typescript tuo projektiin. Kuvassa 21 ovat esimerkit samasta tiedosta tyypitettynä ja ilman tyypitystä.



Kuva 21. Ero Javascriptin ja Typescriptin välillä

Kun Javascriptillä käsitellään post-tietoa, ei koodieditori tiedä, mitä kyseisestä tiedosta haluttaisiin näyttää. Typescriptillä kyseinen tieto on jo tyypitetty, joten koodieditori osaa ehdotta tyypinmukaisia ehdotuksia.

Reactin tapauksessa sivut renderöidään CSR-menetelmällä ja saadakseen React-projektiin käyttöönsä SSG-renderöinnin, tulee tämä usein ottaa huomioon jo projektin alustus vaiheessa. Tarjolla on useita ohjelmistokehyksiä, muun muassa aiemmin käytetty NextJS, joiden päälle projektia lähdetään toteuttamaan. Myöskin SSR-toteutukset sisältyvät eri ohjelmistokehyksiin muun muassa NextJS tai vaativat erillisiä palvelinpuolen ratkaisuja, jotka ovat rajattu tämän opinnäytetyön ulkopuolelle.

5 JOHTOPÄÄTÖKSET

Web-kehitykseen on saatavilla useita erilaisia ohjelmistokehyksiä sekä kirjastoja, jotka lupaavat olla toinen toistaan parempia ja tekevät asiat täysin uudella tavalla. Tässä työssä esiteltiin niistä kaksi ja näitä vertailtiin keskenään. Tehdessäni työtä kävi hyvin nopeasti selväksi, että Next.js tarjoaa suoraan asennuksesta hyvinkin kattavasti työkaluja hyvinkin laajaan frontend toteutukseen. Kun toistin samaa toteutusta Reactilla, piti ulkoisista paketeista tuota paketteja jo pelkästään sivujen välillä liikkumiseen tarvittavan reitittimen muodossa. Suuri seinä tuli eteen, kun olisi haluttu hyödyntää SSG ja SSR- renderöintiä: vaihtoehtoina oli näihin soveltuvien ohjelmistokehyksien asentaminen tai oman toteutuksen koodaaminen palvelin toteutuksineen.

Työtä tehdessä kävi siis selväksi, että nykyaikaisilla renderöintimenetelmillä varustettua React-toteutusta ei ole järkevää lähteä toteuttamaan tyhjästä, vaan on järkevämpää ottaa käyttöön näitä menetelmiä tukeva ohjelmistokehys. Ilman ohjelmistokehystä edessä olisi useista eri paketeista koottu kokonaisuus, joka ei välttämättä toimi hyvin ja saumattomasti yhteen. Tämä myös aiheuttaa ylimääräistä työtä ylläpidon kannalta, koska tällöin on seurattava pakettien päivityksiä sekä sitä, että ylläpidetäänkö käytössä olevaa pakettia enää.

Next.js:n kaltaisen tai muun vastaavan ohjelmistokehyksen mukana tulee käyttöön kattavat dokumentaatiot, kehittäjäyhteisön tuki sekä sovelluskehitystä helpottavia ominaisuuksia yhdestä paketista. Ylläpidollisesti tällaiset ohjelmistokehykset tuovat helpotusta, koska versio päivitysten yhteydessä tarjolla on aina toimiva tuettu kokonaisuus. Miksi siis tulisi tehdä kaikkea itse, kun valmistakin on tarjolla?

Typescriptin osuus työssä jäi lähinnä maininnan arvoiseksi ja pienten esimerkkien tasolle. Kuitenkin sillä on tärkeä osansa nykyaikaista ohjelmistokehitystä. Omakohtaisesti en haluaisi enää toteuttaa ohjelmistoprojekteja ilman Typescriptiä, mikäli se on vain mahdollista. Typescript vaatii alkuvalmisteluja, mutta tähän käytetty aika saadaan takaisin itse työskentelyvaiheessa.

6 PÄÄTÄNTÖ

Aihe itse työlle tuli projektista, jota olen tehnyt aikanaan työharjoittelussa sekä myöhemmin työntekijänä Hurja Solutions Oy:ssä. Työ on myös osa Hurjan uusien teknologioiden testaamista. Opinnäytetyö sisältää samoja toteutuksia Next.js:llä kuin oikeakin projekti, mutta pienemmässä anonyymisoidussa muodossa. Ohjelmistoratkaisut olivat jo ennakkoon määritetty, ennen kuin aloitin projektissa, joten mikä rajasi opinnäytetyötä.

Työtä tehdessä olin tuttujen asioiden äärellä, joten siitäkin syystä työtä oli mielekästä tehdä. Työ myös pakotti syventymään näihin asioihin ja toi uutta tietoa sekä näkemystä tutuistakin asioista. Opinnäytetyötä tehdessä tulleita havaintoja olen pystynyt siirtämään käytännön toteutuksiin, joten työelämä ja opinnäytetyö ovat tukeneet toinen toisiansa. Jos alkaisin suunnittelemaan vastaavaa työtä uudelleen, vaihtaisin Reactin johonkin toiseen ohjelmistokehykseen ja toisin Typescriptiä enemmän nähtäväksi käytännön tasolla.

Opinnäytetyön käytännön osuudesta saatuja tuloksia voidaan pitää mielestäni luotettavina. Tavoitteena oli selvittää erovaisuudet teknologioiden välillä, ja nämä eroavaisuudet työssä onnistuttiin havainnollistamaan. Työ vastaa myös toimeksiantajan tarpeisiin vertailun ja saatujen tulosten pohjalta. Saatujen tulosten myötä Next.js jää osaksi käytössä olevia teknologioita. Next.js päivittyi myös juuri uuteen versioon, joten senkin myötä sillä on elinkaarta vielä hyvin jäljellä.

Itse opinnäytetyöprosessi eteni alkukankeuksien jälkeen sujuvasti. Vaikeinta oli koko prosessin aloittaminen ja tämän jälkeen aiheen rajaamien sopivan kokoiseksi. Olen aikaisemmin tehnyt yhden opinnäytetyön, mutta siitä prosessista on vain lähinnä huonoja kokemuksia ja nämä kokemukset osaltaan painoivat itseäni prosessin aikana. Kuitenkin kiinnostus aihetta kohtaan nosti yleistä tunnelmaa, ja työ oli lopulta hyvä ja opettavainen prosessi.

LÄHTEET

Barger, R. 2022. Get Started with Next.js – The React Library Your Project Needs. Blogi. Päivitetty 14.1.2022. Saatavissa: https://www.freecode-camp.org/news/nextjs-tutorial/#what-is-next-js. [viitattu 11.9.2022].

Bishal, K. 2022. Difference between TypeScript and JavaScript. Blogi. Päivitettu 19.5.2022. Saatavissa: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/. [viitattu 10.9.2022].

Code Institute. 2022a. What Is JavaScript Framework? WWW-dokumentti. Saatavissa: https://codeinstitute.net/global/blog/javascript-framework/ [viitattu 11.9.2022].

Code Institute. 2022b. What is React.js? WWW-dokumentti. Saatavissa: https://codeinstitute.net/global/blog/what-is-react-js [viitattu 11.9.2022].

Create React App. 2022. Adding Typescript. WWW-dokumentti. Saatavissa: https://create-react-app.dev/docs/adding-typescript/ [viitattu 25.9.2022].

Dickson, B. 2022. A Brief History of JavaScript. Blogi. Päivitetty 21.5.2022. Saatavissa: https://dev.to/dboatengx/history-of-javascript-how-it-all-began-92a [viitattu 11.9.2022].

GeeksForGeeks. 2022. NextJs vs ReactJs. WWW-dokumentti. Saatavissa: https://www.geeksforgeeks.org/nextjs-vs-reactjs-which-one-to-choose/ [viitattu 20.9.2022].

InterviewBit. 2021. Framework vs. Library: Full comparison. Blogi. Päivitetty 23.10.2021. Saatavissa: https://www.interviewbit.com/blog/framework-vs-library/ [viitattu 5.10.2022].

Khan, S. 2021. What Is JavaScript Framework? Blogi. Päivitetty 15.3.2021. Saatavissa: https://generalassemb.ly/blog/what-is-a-javascript-framework/ [viitattu 10.9.2022].

MDN Web Docs. 2022a. CDN. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Glossary/CDN . [viitattu 9.10.2022].

MDN Web Docs. 2022b. JavaScript WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript . [viitattu 11.9.2022].

MDN Web Docs. 2022c. Package management basics. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management . [viitattu 9–10.2022].

MDN Web Docs. 2022d. What is JavaScript? WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [viitattu 11.9.2022].

Navdeep, S. 2022. NextJs Key Features and Its Advantages | The Complete Guide. Blogi. Päivitetty 10.5.2022. Saatavissa: https://www.geestack.com/blog/next.js-features [viitattu: 20.9.2022].

Pisuwala, U. 2022. The benefits of ReactJS and reasons to choose it for your project. Blogi. Päivitettyä. 7.2.2022. Saatavissa: https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html. [viitattu 10.9.2022].

React. 2022a. Components and Props. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/components-and-props.html [viitattu 25.9.2022].

React. 2022b. Conditional Rendering. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/conditional-rendering.html [viitattu 25.9.2022].

React. 2022c. Forms. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/forms.html [viitattu 5.10.2022].

React. 2022d. Introducing JSX. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/introducing-jsx.html. [viitattu 10.9.2022].

React. 2022e. Lists and Keys. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/lists-and-keys.html [viitattu 5.10.2022].

Sirotka, A. 2022. What is React? Blogi. Päivitetty. 18.2.2022. Saatavissa: https://flatlogic.com/blog/what-is-react/. [viitattu 10.9.2022].

Typescript. 2022. TypeScript for the New Programmer WWW-dokumentti. Saatavissa: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html. [viitattu 11.9.2022].

Vercel. 2022a. How Next.js Works. WWW-dokumentti. Saatavissa: https://nextjs.org/learn/foundations/how-nextjs-works/rendering [viitattu 19.9.2022].

Vercel. 2022b. Inrcremental Static Regeneration. WWW-dokumentti. Saatavissa: https://nextjs.org/docs/basic-features/data-fetching/incremental-static-regeneration [viitattu 4.10.2022].

Vercel. 2022c. Pages. WWW-dokumentti. Saatavissa: https://nextjs.org/docs/basic-features/pages [viitattu 11.9.2022].

W3Techs. 2022. Usage statistics of JavaScript libraries for websites. WWW-dokumentti. Saatavissa: https://w3techs.com/technologies/overview/javascript_library [viitattu 11.9.2022].