Walter Ruoppa

JavaScript-kirjastojen (React, Mithril ja Preact) vertailua ohjelmoijan näkökulmasta

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

10.05.2023

Tiivistelmä

Tekijä:	Walter Ruoppa
Otsikko:	JavaScript-kirjastojen (React, Mithril ja Preact) vertailua ohjelmoijan näkökulmasta
Sivumäärä:	23 sivua
Aika:	10.05.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander

Tämän opinnäytetyön tarkoituksena on vertailla JavaScript-kirjastoja React, Mithril ja Preact, kuvata niiden ominaisuuksia ja käyttöönottoa sekä arvioida niiden suorituskykyä. Kirjaston valinta on keskeinen päätös sovelluskehityksessä, ja siksi on tärkeää ymmärtää, mitä kukin kirjasto tarjoaa.

Työn tuloksena on katsaus React-, Mithril- ja Preact-kirjastojen ominaisuuksiin, käyttöönottoon ja suorituskykyyn. Näiden tietojen avulla ohjelmoija voi arvioida, ovatko nämä JavaScript-kirjastot sopivia käytettäväksi omassa sovelluskehityksessä.

Avainsanat: React, Mithril, Preact, kirjastot

Abstract

Author:	Walter Ruoppa
Title:	Comparing JavaScript Libraries React, Mithril and Preact from Programmer's Perspective
Number of Pages:	23 pages
Date:	10 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer

The purpose of this thesis is to compare the JavaScript libraries React, Mithril and Preact, describe their features and deployment, and evaluate their performance. Choosing a library is a key decision in application development, and it is therefore important to understand what each library offers.

The result is an overview of the features, deployment, and performance of React, Mithril and Preact libraries. With this information, the programmer can evaluate whether these JavaScript libraries are suitable for use in their application development.

Keywords: React, Mithril, Preact, Libraries

Sisällys

Lyhenteet

ORM: *Object-relational mapping*. Ohjelmiston oliomallin mukaisen esityksen kuvaaminen relaatiotietokantamallin mukaiseksi esitykseksi ja kääntäen.

TKHJ: Tietokannan hallintajärjestelmä. Ohjelmisto tiedon tehokkaan hakemisen, säilyttämisen ja päivittämisen toteuttamiseksi.

CLI: Command line interface. Tarkoittaa komentorivikäyttöliittymää. Se on käyttöliittymätyyppi, joka mahdollistaa käyttäjien vuorovaikutuksen tietokoneohjelmien tai -sovellusten kanssa tekstipohjaisia komentoja syöttämällä.

CSS: Cascading Style Sheets. Tyylikirjasto. Tämän avulla voidaan muokata elementtien tyyliä ja järjestystä.

DIV: Yleinen HTML-elementti, jonka avulla kehittäjät voivat ryhmitellä verkkosivujen sisältöä ja soveltaa tyylejä elementteihin.

DOM: Document Object Model. JavaScriptin tapa olla vuorovaikutuksessa verkkosivun elementtien kanssa ja käsitellä niitä, mikä mahdollistaa dynaamisen ja interaktiivisen sisällön.

FCP: First Contentful Paint, viittaa ajankohtaan, jolloin selaimen ensimmäinen sisältö latautuu. Osoitus siitä, milloin käyttäjä havaitsee ensimmäistä kertaa jotain sisältöä sivulla.

HTML: HTML-dokumentin tai verkkosivun perusrakenne. Se edustaa erityyppistä sisältöä, kuten tekstiä, kuvia, linkkejä, lomakkeita ja multimediaa.

Hyperscript: JavaScript-kirjasto, joka tarjoaa koodin yksinkertaisen kirjoittamistavan HTML-elementtien luomiseen funktiopohjaisella lähestymistavalla.

LCP: Largest Contentful Paint (LCP) mittaa ajan, jossa pisimmän latausajan omaava elementti on ladattu ja näytetty käyttäjälle. Tämä elementti on yleensä sivun tärkein tai näkyvin sisältö. Latausajat vaikuttavat merkittävästi käyttäjäkokemukseen sivun nopeudesta ja sujuvuudesta.

Lighthouse: Googlen kehittämä sovellus, jolla voi tutkia monia eri sivuston ominaisuuksia. Näitä ovat esimerkiksi suorituskyky ja saavutettavuus.

NPM: Node Package Manager. Komentorivityökalu, jonka avulla kehittäjät voivat helposti asentaa ja hallita JavaScript-projektiensa kirjastoja ja riippuvuuksia.

Solmu: Yksittäinen osa dokumenttioliomallista, josta dokumenttioliomalli muodostuu. Se koostuu puolestaan elementeistä ja yksi solmu kuvaa yhtä elementtiä siinä.

Speed index: Aika, kuinka nopeasti verkkosivun sisältö ladataan ja milloin sen sisältö tulee näkyviin. Liittyy käyttäjäkokemukseen mittaamalla, kuinka nopeasti sivuston näkymä latautuu käyttäjälle.

TBT: Total Blocking Time. Ilmaisee kokonaisaikaa, jonka ajan sivusto ei vastaa käyttäjän syötteisiin.

VCS: Version Control System eli versionhallintajärjestelmä hallitsee tiedoston tai tiedostojoukon tehtyjä muutoksia. Nämä muutokset tunnistetaan usein numeroiksi tai kirjainkoodeiksi, joita kutsutaan versionumeroiksi. Jokaiseen versioon liittyy metatietoja, kuten aikaleima ja tekijä. Versioita voidaan verrata, palauttaa tai yhdistää tarpeen mukaan.

Virtuaalinen dokumenttioliomalli:

Kopio dokumenttioliomallista. Kun verkkosovelluksen tilassa tapahtuu muutos, Virtual dokumenttioliomalli huomaa erot kopion ja dokumenttiolion välillä. Tämän jälkeen sen tarvitsee päivittää vain ne dokumenttioliomallin osat, joiden tila on muuttunut. Tällöin muutetaan.

JSX: Mahdollistaa HTML:n kaltaisen koodin kirjoittamisen JavaScriptin-koodin sisällä.

1 Johdanto

JavaScript-kirjastot ovat valmiita koodikokoelmia, jotka sisältävät usein erilaisia lähestymistapoja, toimintoja tai ratkaisuja ohjelmoinnin haasteisiin. JavaScript-kirjastojen käyttäminen voi säästää kehittäjien aikaa ja vaivaa, sillä kirjastot ovat valmiiksi saatavilla eikä niitä ei tarvitse kirjoittaa alusta alkaen. Kirjastoja on valmiina moniin erilaisiin sovellustarkoituksiin kuten työpöytä-, mobiili- ja verkkosovelluksiin.

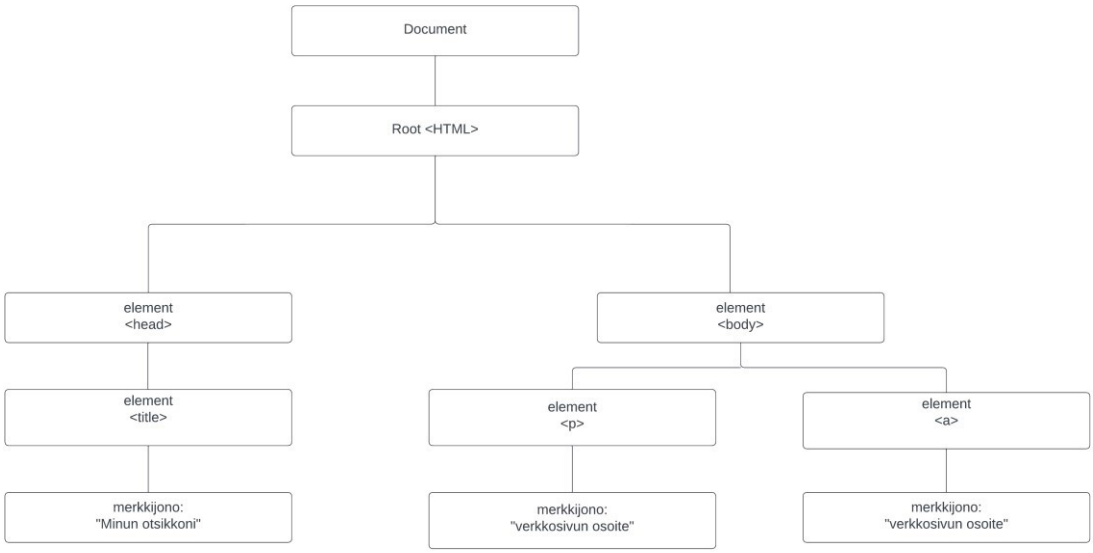
Tämän opinnäytetyön tavoitteena on vertailla JavaScript-kirjastoja React, Mithril ja Preact. Työssä vertaillaan kirjastojen ominaisuuksia, käyttöönottoa ja suorituskykyä.

Ominaisuuksia arvioitiin tarkastelemalla, kuinka ne päivittävät dokumenttioliomallia ja niiden tavoista kirjoittaa JavaScript-koodia kirjoittamisessa. Käyttöönoton helppoutta ja monipuolisuutta arvioitiin tarkastelemalla, kuinka sujuvaa ja yksinkertaista käyttöönotto on ja kuinka monta eri vaihtoehtoa kehykset tarjoavat asennuksessa. Suorituskykyä mitattiin Lighthouse-ohjelmalla (sovellus, jolla voidaan esimerkiksi mitata suorituskykyä ja saavutettavuutta).

Tarkoituksena on vertailla kolmea JavaScript-kirjastoa monesta eri näkökulmasta ja antaa ohjelmoijille ja aiheesta kiinnostuneille arvioita, voisivatko ne sopia omaan käyttöön.

2 Dokumenttioliomalli

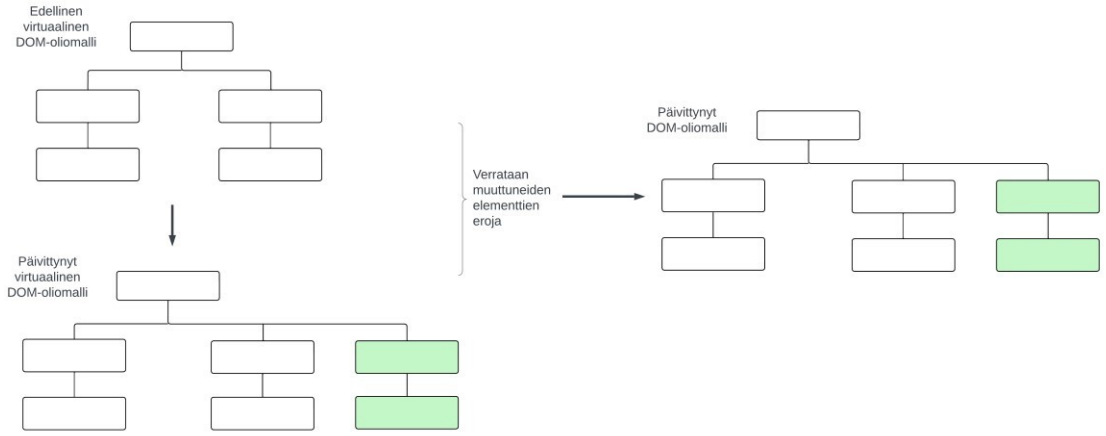
Dokumenttioliomalli (DOM) kuvaa (kuva 1) dokumentin elementtien hierarkkisen puurakenteen. Se mahdollistaa tietojen välittämisen elementtien välillä ja viittaamisen niihin. Dokumenttioliomalli koostuu ennalta määritetyistä määrästä solmuja (engl. nodes), jotka muodostavat hierarkkisen dokumenttipuun. Jokainen solmu dokumenttioliomallissa on oma olio, joka voi myös sisältää useita lapsisolmuja (engl. child nodes). Kaikilla solmuilla, paitsi juurisolmulla (engl. root), on myös oma äitisolmu (engl. parent node). Dokumenttipuut voidaan ajatella muistuttavan rakenteeltaan puuta, jossa on juuri, oksia ja lehtiä. Juuri vastaa dokumentin rakennetta tai juurisolmua, oksat vastaavat lapsielementtejä ja lehdet ovat solmuja, joilla ei ole enää omia lapsielementtejä. [1.]



Kuva 1. Havainnollistava kuva dokumenttioliomallin rakenteesta.

2.1 Virtuaalisen dokumenttioliomallin päivitys

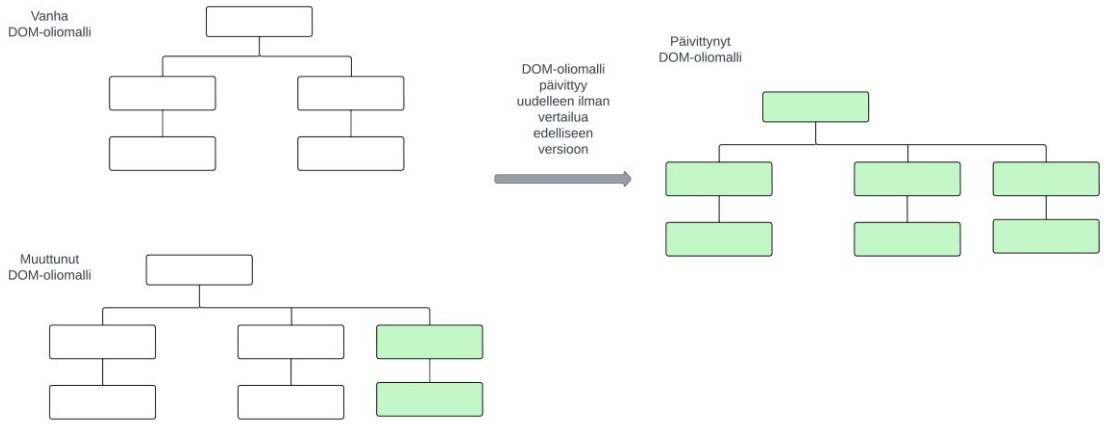
Kirjastojen tavat päivittää dokumenttioliomallia tapahtuu virtuaalisen dokumenttioliomallin [2] avulla (kuva 2). Se on kopio dokumenttioliomallista ja se on kuin piirustus, jolla pidetään kirjaa, mitä muutoksia sovelluksessa tapahtuu. Kun sovelluksessa päivitetään elementtiä, muodostuu uusi versio virtuaalisesta dokumenttioliomallista, jota sitten verrataan vanhaan versioon. Tässä vertailussa löydetään muuttuneet osat, ja tämän avulla tarvitsee väin päivittää muuttuneet elementit dokumenttioliomalliin.



Kuva 2. Havainnollistava kuva virtuaalidokumenttiolion päivittymisestä.

2.2 Dokumenttioliomallin päivitys

Dokumenttioliomallin käyttäminen ilman virtuaali dokumenttioliomallin hyödyntämistä vaatii jokaisen elementtien muutoksen yhteydessä päivittämään sen kokonaisuus (kuva 3). Aina kun tilassa tapahtuu muutos, jokainen solmu käydään lävitse ja tarvittavat muutokset tehdään suoraan sen elementteihin. Lisäksi dokumenttioliomallin puurakenne muuttuu isoksi ja monimutkaiseksi helposti ohjelmien kasvaessa, ja tämän tehottomamman lähestymistavan merkitys korostuu. Tästä seuraa ohjelman hidastuminen ja käyttäjäkokemuksen huonontuminen.



Kuva 3. Havainnollistava kuva dokumenttioliomallin päivityksestä hyödyntämättä virtuaalia dokumenttioliomallia.

3 JavaScript-kirjastot

JavaScript-kirjastot ovat olennainen osa ohjelmointia, tarjoten kehittäjille valmiita ratkaisuja monenlaisiin ohjelmoinnin haasteisiin. Tässä kappaleessa tutustutaan JavaScript-kirjastoihin: React, Mithril ja Preact. Käymme läpi näiden kirjastojen ominaisuuksia, käyttöönottoa, suorituskykyä ja suosiota.

3.1 React

React on JavaScript-kirjasto, jonka kehitti Facebook (nykyisin Meta) vuonna 2011. Se on suunniteltu rakentamaan verkkosovelluksien käyttöliittymiä. Se hyödyntää komponenttipohjaista lähestymistapaa, jossa käyttöliittymän eri osat jaetaan erillisiin komponentteihin. Tätä voidaan ajatella pienten osien yhdistämisenä, joista lopuksi valmistuu isompi kokonaisuus. Syy siihen, miksi React on käyttäjien suosiossa, johtuu sen laajasta yhteisöstä ja yhteisön kehittämistä valmiista koodiratkaisuista. React käyttää virtuaalidokumenttioliomallia (kuva 3). [3.]

3.1.1 Reactin ominaisuudet

Reactissa hyödynnetään JSX-kieltä, joka mahdollistaa HTML:n kaltaisen koodin kirjoittamisen JavaScriptin-koodin sisällä. Reactin avulla JSX-koodi muunnetaan virtuaaliseksi dokumenttioliomalliksi, joka tekee dokumenttioliomallin päivittämisen nopeaksi ja optimaaliseksi.

3.1.2 Reactin käyttöönotto

Reactin käyttöönotto on helppoa. Komento "npx create-react-project my-project" kirjoitetaan komentoriville, jossa "my-project" tarkoittaa projektin nimeä. Esiasennettuja riippuvaisuuksia ei ole paljon. Seuraavaksi siirrytään luotuun projektikansioon komennolla "cd <projektin nimi>" ja annetaan komento "npm start". Sovellus käynnistyy ja sitä voi tarkastella verkkoselaimessa. [4.]

3.1.3 Reactin suosio

React on yksi suosituimmista kirjastoista ja viikottaiset lataukset sivulla ovat huikeat 18,5 miljoonaa. [5.]

React on Github-sivustolla saanut noin 207 tuhatta tähteä. Se on yksi suosituimmista avoimen lähdekoodin projekteista. Suuri tähtimäärä osoittaa, että React on saavuttanut laajan suosion kehittäjäyhteisössä. Github-sivuston projektikohtaisten tähtien määrä toimii yleensä indikaattorina projektin suosiosta ja käyttäjämäärästä.

3.2 Mithril

Mithril on JavaScript-kirjasto, jonka Leo Horie kehitti vuonna 2013. Se on suunniteltu erityisesti verkkosovellusten ja käyttöliittymien rakentamiseen. Mithril on kooltaan pieni kirjasto, ja sen pakkauksen koko on 28,14 kb ja pakattuna 10.58 kb. [6.] Tämä nopeuttaa latausaikaa ja parantaa käyttäjäkokemusta.

Mithril.js on JavaScript-kirjasto, joka noudattaa Model-View-Controller (MVC) -mallia. Tämä tarkoittaa, että sovellus on jaettu kolmeen erilliseen moduuliin: malliin (model), näkymään (view) ja käsitteljään (controller). [7.] Tämän avulla sovelluksesta saadaan ylläpidettävämmän, ja sen muokkaaminen on järjestellisempää ja voi helpottaa sovelluksen skaalautuvuutta.

3.2.1 Mithrilin ominaisuudet

Mithril tukee myös JSX:ää, kuten Preactia ja Reactia. Mithril hyödyntää omaa m()funktiota, joka muuntaa JSX:n virtuaalisiksi solmuiksi. Mithril.js:n virtuaalinen dokumenttioliomalli on myös optimoitu päättelemään vähimmäismäärän operaatioita, joiden avulla verkkosovelluksen tilaa saadaan muutettua. Kun käyttäjä on luonut virtuaalisen solmun mithril()-funktiolla, voi käyttäjä hyödyntää m.render()-funktiota luomalla varsinaisen dokumenttioliomallin näistä virtuaalisista solmuista. Render()-funktiolla siis luodaan varsinaiset solmut näiden virtuaalisten solmujen kautta. Lisäksi optimoitu virtuaalinen dokumenttioliomalli nopeuttaa solmujen käsittelyä ja on tehokkaampaa, mikä parantaa verkkosivujen ja sovellusten käytettävyyttä sen nopean latausaikojen ja päivittämisen takia.

3.2.2 Mithrilin käyttöönotto

Mithrilin käyttöönotto alkaa valitsemalla haluttu kohdekansio ja avaamalla se halutussa kehitystyökalussa tai navigoimalla siihen komentorivillä. Tämän jälkeen luodaan uusi Node.js-projekti suorittamalla komento "npm init -yes". Seuraavaksi asennetaan seuraavat riippuvuudet, Mithril.js ja Webpack, komennolla npm: "npm install mithril --save" ja "npm install webpack webpack-cli --save-dev". Lisäksi on luotava index.js-tiedosto ja asennettava skripti package.json-kansioon. Kun komento "npm start" suoritetaan, Webpack suoritetaan käyttämällä src/index.js-tiedostoa sovelluksen aloituspisteenä. Samalla niputetaan kaikki tarvittavat moduulit ja riippuvuudet bin/app.js-tiedostoon. Pohjimmiltaan tämä skripti käynnistää kehityspalvelimen, joka tarkkailee koodin muutoksia ja rakentaa sovelluksen uudelleen aina, kun

muutoksia tehdään. Lopuksi sovellus käynnistetään komennolla "npm start" ja index.html-polku kopioidaan selaimeen sovelluksen suorittamista varten. Lopuksi sovellus käynnistetään npm start -komennolla. Kopioidaan index.html-polku selaimeen, ja sovellus toimii. [8.]

3.2.3 Mithrilin suosio

Mithrilin viikoittaisten latausten lukumäärä on 58 tuhatta. Sitä kuitenkin käyttävät useat ihmiset. Suosioon vaikuttavat varmaan sen hankalampi käyttöönotto ja nimi, joista moni ei ole kuullut aikaisemmin. Lisäksi se ei esimerkiksi pohjaudu suosittuun olemassa olevaan kirjastoon ja sen oma syntaksi vaikeuttaa sen suosiota.

Mithril on saanut käyttäjiltä Githubin puolella tähtiä noin 14 tuhatta kappaletta [9].Tämä on vähän verrattuna Reactiin, joka sai puolestaan 207 tuhatta tähteä.

3.3 Preact

Preact on pienikokoinen ja nopea JavaScript-kirjasto käyttöliittymien rakentamiseen. Kirjaston on kehittänyt Jason Miller, se julkaistiin vuonna 2016. Preactin pakkauksen koko on 10,35 kb ja koko pakattuna 4,5 kb. Koko poikkeaa hieman Preactin omalla kotisivullaan olevasta 3 kb:n väitteestä [10], joka tekee siitä yhden pienimmistä käyttöliittymien rakentamiseen tarkoitetuista kirjastoista. Se sopii erinomaisesti sovelluksien kehittämiseen nopeutensa ja optimointinsa puolesta.

3.3.1 Preactin ominaisuudet

Preact käyttää myös JSX:ää kuten Reactia. Sillä voi myös käyttää sen omaa h()-funktiota Tämä on nopeampi ja tehokkaampi tapa käsitellä solmuja kuin ilman virtuaalisia solmuja. Preactin virtuaalinen dokumenttioliomalli on optimoitu päättelemään vähimmäismäärän operaatioita, joiden avulla verkkosovelluksen tilaa saadaan muutettua.

Kun käyttäjä on luonut virtuaalisen solmun h()-funktiolla, voi käyttäjä hyödyntää render()-funktiota luomalla varsinaisen dokumenttioliomallin näistä virtuaalisista solmuista. Render()-funktiolla siis luodaan varsinaiset solmut näiden virtuaalisten solmujen kautta.

Tämän ansiosta käyttäjä voi kirjoittaa JSX:ää helpommin ja tehokkaammin dynaamisia käyttöliittymiä verkkosivuille ja sovelluksille. Preactin optimoitu virtuaalinen dokumenttioliomalli tekee solmujen käsittelystä nopeampaa ja optimoidumpaa verrattuna pelkkään dokumenttioliomallin päivittämiseen. Tämä parantaa verkkosivujen ja sovellusten käyttöä, koska sisältö latautuu nopeammin ja sivustosta tulee nopeammin interaktiivinen.

3.3.2 Preactin käyttöönotto

Preactin käyttöönotto on suoraviivaista. Komento "npx preact-cli create default my-project" kirjoitetaan komentoriville, jossa "default" tarkoittaa käytettävää projektimallia. Tässä tapauksessa "default"-malli sisältää esiasennetut reititystoiminnot, ja käytettävissä on vielä viisi muuta mallia. Kun sopiva projektimalli on valittu, siirrytään luotuun projektikansioon komennolla "cd <projektin nimi>" ja annetaan komento "npm run dev". Sovellus käynnistyy ja sitä voi tarkastella verkkoselaimessa. [11.]

3.3.3 Preactin suosio

Preact on suosittu kirjasto. Suosio selittyy varmaan sillä, että Preact perustuu huippusuosittuun React-kirjastoon. Preactin viikottaisten latausten määrä on 2,2 miljoonaa.

Github-sivustolla Preactille annettujen tähtien lukumäärä on noin 34 tuhatta [12].

4 Testattavat sovellukset

Tein pienehkön sovelluksen kaikilla kielellä, joka luo 10 000 div-elementtiä. Molemmat Reactin ja Preactin ohjelmat on kirjoitettu JSX:llä, joten niissä ei ole muuta eroja ulkonäöllisesti kuin Mithrillä hyödynnetty hyperscript ja sen käyttämä MVC-mallin arkkitehtuuri ja sisältämä m()-funktio.

Sovellukseen lisättiin CSS-tyylejä (Cascading Style Sheets, tyylikirjasto. Tämän avulla voidaan muokata elementtien tyyliä ja järjestystä) jotta sovellusta olisi selkeämpää tarkastella (kuva 4).

Kuva 4. Havainnollistava kuva siltä, miltä nettisivu näyttää, kun 10 000 divelementtiä päivittyy selaimelle. Jokaisella Rivillä on 15 div-elementtiä, lukuun ottamatta viimeistä riviä, jossa div-elementtejä on 10 kappaletta. Sarakkeita on yhteensä 667 kappaletta.

4.1 React-koodi

Esimerkkikoodi 1 on yksinkertainen funktio nimeltään ”DivList” luo 10 000 divelementtiä for-silmukan avulla. Tämä tarkoittaa, että silmukka jatkuu 10 000 kertaa, kunnes kaikki div-elementit on siirretty listaan nimeltä ”divs”.

```
import React from 'react'
import './App.css' const
DivList = () => {   const
divs = []
  for (let i = 1; i <= 10000; i++) {
divs.push(
      <div key={i} className="item">
div numero: {i}
      </div>
) }
  return <div className="container">{divs}</div>
}
const App = () => {
   return
(
    <div className="app-container">
      <DivList />
    </div>
  )
}

export default App
```

Esimerkkikoodi 1. Testeissä käytettävän koodin toteutus

4.2 Mithril-koodi

Esimerkkikoodissa 2 näytetään Mithril-kirjaston käyttöä samanlaisen toimintalogiikan toteuttamiseksi kuin Preactissa. Mithril-ohjelmoinnissa käytetään hyperscript-funktiota nimeltään m(), joka mahdollistaa HTML-dokumentin ilmaisemisen JavaScript-syntaksilla. Funktion ensimmäinen parametri on elementtivalitsin (engl. Selector), jolla valitaan käytettävät elementit. Näitä ovat esimerkiksi koodiesimerkeissä esiintynyt div-elementti (koodiesimerkki 1). Funktiolle voidaan antaa vielä valinnaisia ominaisuuksi kuten tyylikirjastosta luokan nimellä ”class : item” ja merkkijonon ”div numero” (esimerkkikoodi 2).

```
import m from 'mithril'
import './style.css' export
const MainView = {   view:
function () {     const
divList = []
    for (let i = 1; i <= 10000; i++) {
divList.push(m("div", { "class": "item" }, "div numero:
" + i))
    }
    return m("div.container", divList)
  }
}
```

Esimerkkikoodi 2. Sovelluksessa toteutettu koodi, millä luotiin 10 000 div-elementtiä.

4.3 Preact-koodi

Preactin koodi (esimerkkikoodi 3) on lähinnä identtinen verrattuna React-koodiin (esimerkkikoodi 1). Tämä johtuu siitä, koska Preact pyrkii saavuttamaan 100 %:n yhteensopivuuden Reactin kanssa.

```
import { h } from 'preact' import
style from './style.css'

const DivList = () => {    const divs = Array.from({
length: 10000 }, (_, i) => (
          <div class={style['item-box']}>div numero: {i +
1}</div>
     ))
     return <div class={style['container']}>{divs}</div>
}
const App = () => (
     <div id="app" class={style['app-container']}>
          <DivList />
     </div>
)
 export default
App
```

Esimerkkikoodi 3. Sovelluksessa testeissä käytetyn koodin toteutus.

4.4 Cascading-Style Sheets (CSS)

Kaikissa JavaScript-kirjastoissa hyödynnettiin yhtä ja samaa CSS-tyylipohjaa. Flexbox on CSS-ominaisuus, joka mahdollistaa käyttöliittymäelementtien helpon ja tehokkaan järjestelyn ja sijoittelun joustavasti. Flexboxilla voidaan luoda rajoitettu tila tai säiliö, jossa yksittäisiä elementtejä voidaan muotoilla ja sijoitella joustavasti. [13.]

Flex-wrap-ominaisuutta käyttämällä voidaan määritellä, kuinka monta elementtiä mahtuu yhdelle riville, jonka jälkeen seuraava elementti siirtyy uudelle riville. Tämä prosessi jatkuu, kunnes kaikki elementit ovat latautuneet. Ilman tätä toimintoa divelementit olisivat vain latautuneet allekkain 10 000 kertaa nettiselaimessa

5 Lighthouse-testit

Lighthouse on Googlen kehittämä avoimen lähdekoodin työkalu, joka auttaa kehittäjiä arvioimaan verkkosivustojen laadun ja suorituskyvyn eri osa-alueita. Lighthousen avulla voidaan arvioida monia erilaisia tekijöitä, kuten suorituskykyä ja saavutettavuutta. [14.] Testauksen osalta tässä keskityttiin tarkastelemaan JavaScript-kirjastoja ja suorituskykyä ja nopeutta.

Lighthousen avulla mitattavat suorituskykysuureet ovat 1) ensimmäisen sisällön latausaika (First Contentful Paint), 2) kokonaisestoaika (Total Blocking Time), joka ilmaisee kokonaisaikaa, jonka ajan sivusto ei vastaa käyttäjän syötteisiin, 3) nopeusindeksi (Speed Index), joka ilmaisee, kuinka nopeasti verkkosivun sisältö ladataan ja tulee näkyviin käyttäjälle ja 4) suurimman sisällön latausaika (Largest Contentful Paint). [15.]

Suorituskykytestiä varten luotiin 10 000 div-elementtiä 2 500 div-elementtien erissä käyttäen kaikkia JavaScript-kirjastoja. Aluksi luotiin yksi div-elementti ja sen jälkeen jatkettiin 2 500 div-elementin jaksoissa. Lighthousen avulla luodut suorituskykytestit suoritettiin alusta asti 10 kertaa jokaiselle arvolle ja tulokset kerättiin taulukoihin ja kaavioihin. Suorituskykytestit suoritettiin selaimessa käyttäen Lighthouse-sovellusta. Se päivittää selaimen näkymän ja mittaa aikaa, joka kuluu esimerkiksi, kauanko sivustolla kestää ladata ensimmäinen päivittyvä sisältö ja milloin sivustosta tulee interaktiivinen käyttäjälle.

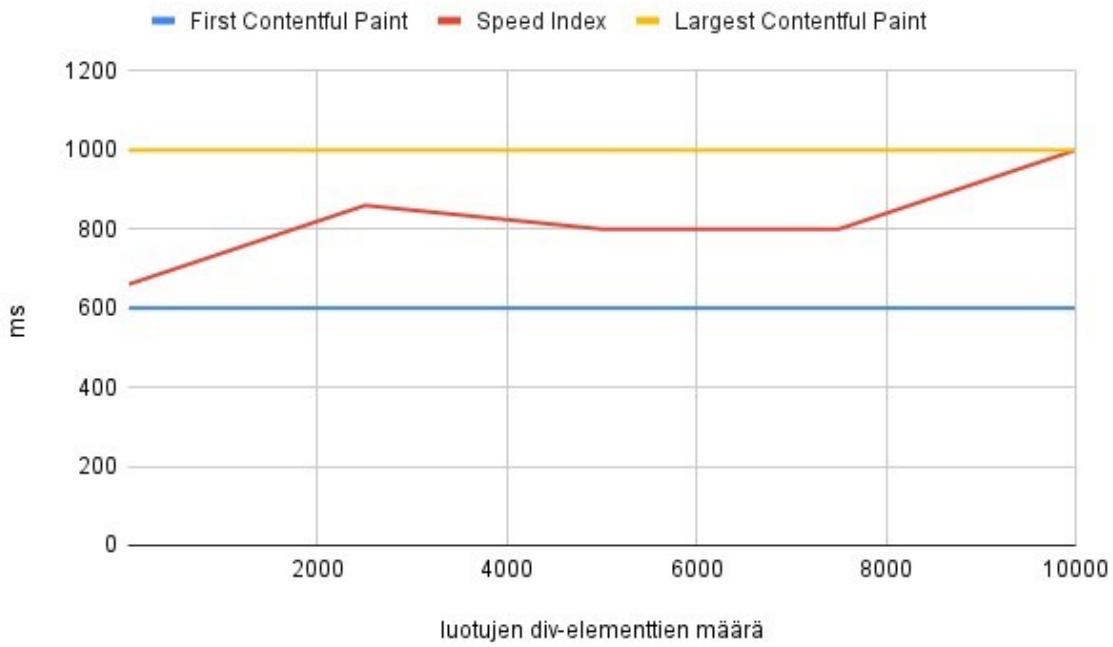
5.1 React-sovelluksen mittaukset

Taulukossa 1 on React-sovelluksen mittattujen aikojen lasketut keskiarvot jokaiselle taulukossa näkyvälle div-elementtien määrille.

Taulukko 1. Reactilla luotujen div-elementtien luonnissa mitatut keskimääräiset ajat.

div-elementtien kappale määrä	First Contentful Paint (ms)	Total Blocking Time (ms)	Speed Index (ms)	Largest Contentful Paint (ms)
10000	600	109	1000	1000
7500	600	76	800	1000
5000	600	46	800	1000
2500	600	17	860	1000
1	600	0	660	1000

Kuvaajassa (kuva 5) Reactin arvot vaihtelivat vain speed indeksin osalta. Arvot First Contentful Paint ja Largest Contentful Paint pysyivät tasaisina testien ajan.



Kuva 5. Div-elementtien määrän vaikutus Reactin suorituskykyyn: FCP:n, Speed Indexin ja LCP:n testitulokset millisekunteina.

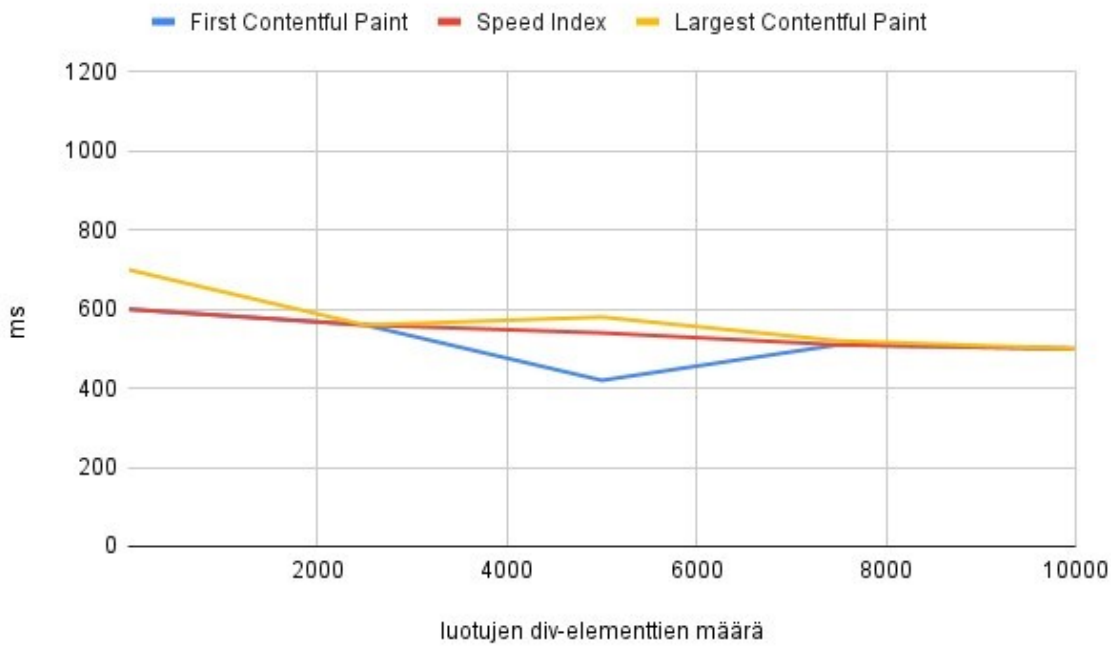
5.2 Mithril-sovelluksen mittaukset

Taulukossa 2 havaitaan, että Mithrilin TBT-arvo (kokonaisestoaika) pysyy merkittävän pienenä, vaikka elementtien määrä kasvaa jopa 10 000:een. LCP-arvo (pisimmän latauksen kestävän elementin latausaika) ei myöskään merkittävästi muutu elementtien määrän kasvaessa. Huomattavaa on kuitenkin, että kaikki mitatut arvot ovat alhaisia. Tämä tarkoittaa, että Mithrilin dokumenttioliomallin päivittäminen on optimoitu hyvin.

Taulukko 2. Mithril div-elementtien luonnissa mitatut keskimääräiset ajat

div-elementtien kappale määrä	First Contentful Paint (ms)	Total Blocking Time (ms)	Speed Index (ms)	Largest Contentful Paint (ms)
10000	500	59	500	500
7500	510	48	510	520
5000	420	35	540	580
2500	560	29	560	560
1	600	10	600	700

Kuvaajassa (kuva 6) Mithrillä saatujen testien arvojen vaihtelu on maltillista. Arvojen vaihtelu ei ole suurta yhden ja 10 000 div-elementin välillä. Tästä voi päätellä, että Mithrilin oma lähestymistapa dokumenttiolion päivittämiseen on optimoitu hyvin, ja siksi se on tehokas ja toimiva.



Kuva 6. Div-elementtien määrän vaikutus Mithrilin suorituskykyyn: FCP:n, Speed Indexin ja LCP:n testitulokset millisekunteina.

5.3 Preact-sovelluksen mittaukset

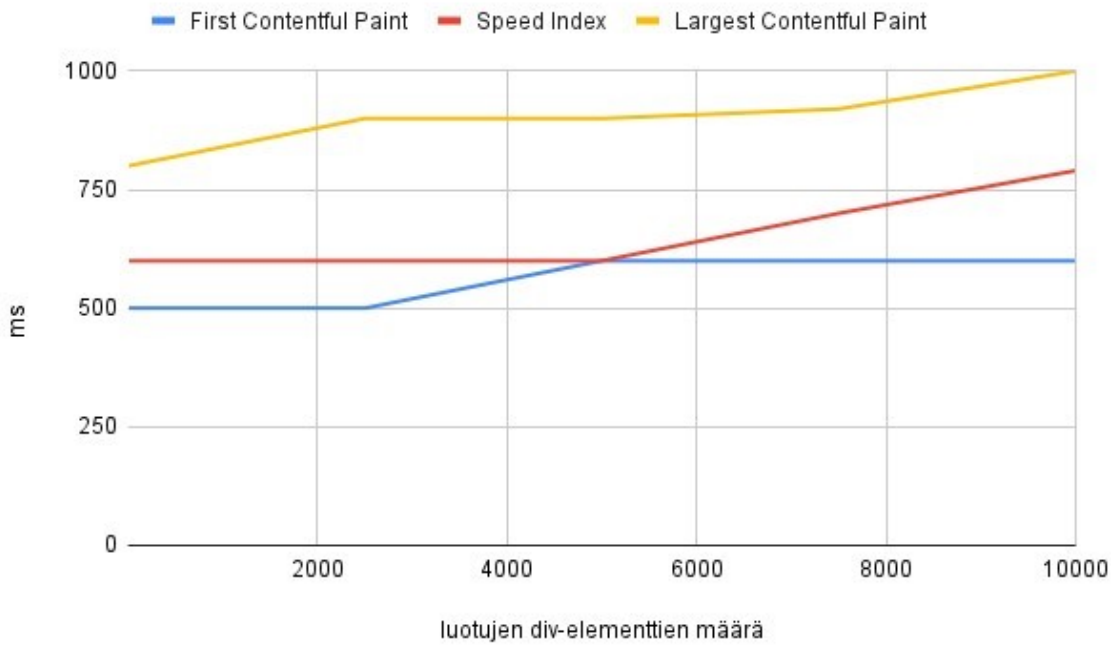
Tarkasteltaessa taulukkoa 3 havaitaan, että TBT- ja LCP-arvot kasvavat sovelluksen div-elementtien määrän kasvaessa. Tämä oli oletettavissa, koska lisääntyvä div-elementtien määrä vaikuttaa siihen, että sivuston lataaminen hidastuu.

Puolestaan kun div-elementtien määrä on pieni, TBT-arvo on lähes olematon. Testien arvot ovat alhaisia, mistä voidaan päätellä, että tässä testitilanteessa kirjasto suoriutui div-elementtien luomisesta tehokkaasti ja nopeasti.

Taulukko 3. Preactilla luotujen div-elementtien luonnissa mitatut keskimääräiset ajat.

div-elementtien kappale määrä	First Contentful Paint (ms)	Total Blocking Time (ms)	Speed Index (ms)	Largest Contentful Paint (ms)
10000	600	96	790	1000
7500	600	69	700	920
5000	600	38	600	900
2500	500	14	600	900
1	500	0	600	800

Kuvaajaa tarkasteltaessa (kuva 7) huomataan taulukossa speed index -arvon nousevan 5000 elementin jälkeen. Tämä tarkoittaa, että div-elementtien kasvaessa sivuston sisällön lataus alkaa hidastumaan.



Kuva 7. Div-elementtien määrän vaikutus Preactin suorituskykyyn: FCP:n, Speed Indexin ja LCP:n testitulokset millisekunteina.

6 Samankaltaisuuksia ja eroavaisuuksia

Kaikki tässä työssä olevat JavaScript-kirjastot nopeuttavat dokumenttioliomallin päivitystä hyödyntämällä virtuaalista dokumenttioliomallia. Vaikka nämä JavaScripit-kirjastot hyödyntävät tätä kyseistä tekniikkaa, niissä on kuitenkin eroja.

6.1 Ominaisuudet

Verrattuna Reactiin, Mithrilin ja Preactin lisäalgoritmien tuoma optimointi virtuaalisen dokumenttioliomallin vertailun kanssa nopeuttavat yhä dokumenttioliomallin vertailusta yhä nopeampaa.

6.2 Käyttöönotto

Käyttöönotto Reactin ja Preactin kannalta on todella suoraviivaista. Mithril vaatii hieman enemmän asetuksien muokkaamista aloitettaessa projektia verrattuna Reactiin ja Preactiin tavasta. Tämä voi olla aluksi haastavaa, mutta se antaa myös enemmän tilaa ja joustavuutta paremman hallinnan sovelluksen alustuksessa.

6.3 Suorituskyky

Lighthouse-testeissä lävitse käytiin suorituskykytestejä. Siinä tutkittiin, miten selaimessa latautuvien div-elementit määrät vaikuttivat JavaScript-kirjastojen React, Mithril ja Preact latausaikoihin. Vaikka testissä käytettiin yksinkertaista dokumenttioliomallirakennetta, niin silti tuloksissa havaittiin eroavaisuuksia, mutta myös samankaltaisuuksia.

Reactin ja Preactin käyrät muistuttivat toisiaan TBT-käyrissä (kuva 11) Preact on pakettikooltaan pienempi vaihtoehto Reactille, joka on suunniteltu toimimaan samalla tavalla kuin React, joka hyödyntää Reactin ohjelmointirajapintaa. Tästä

johtopäätöksenä voidaan vedota, että niiden käyrien samankaltaisuus on mahdollista. Kuitenkin Preactilla on myös omat dokumenttioliomallin optimointinsa, joka voi vaikuttaa sen suorituskykyyn ja käyrien muotoon.

Tutkiessa jokaisen JavaScript-kirjaston omia kuvaajia (kuva 6, kuva 7, kuva 8) olivat ne kaikki erilaisen näköisiä. Näistä jokainen JavaScript-kirjasto on erilainen ja niiden käyrien vaihtelu voi riippua monista tekijöistä. Näitä tekijöitä voivat olla kirjastojen avulla kirjoitetun koodin laatu ja parhaiden käytäntöjen noudattaminen, niiden omat hyödyntävät algoritmit virtuaalidokumenttiolion päivityksen yhteydessä, ja käytettävän selaimen suorituskyky. Lisäksi tuloksiin vaikuttaa, millä laitteella testit on tehty, esimerkiksi puhelin tai pöytäkone.

6.4 Pakkauksien koot

Kaikki nämä kirjastot on julkaistu viiden vuoden sisällä toisistaan. Preact on näistä kirjastoista uusin ja julkaistu vuonna 2016. Preactin pakkauksen koko on vain 10,35 kb ja koko pakattuna 4,15 kb. Puolestaan ensiksi julkaistu React vuonna 2011 on kooltaan isoin. Sen pakkauksen koko on 138.35 kb ja koko 44.48 kb pakattuna. Mithrilin pakkauksen koko on 28.14 kb ja pakattuna 10.58 kb.

6.5 Esiintyviä eroja testauksessa

Kaikkien JavaScript kirjastojen testauksessa huomattiin eroavaisuuksia mitattujen keskiarvojen välillä. Eroavaisuudet tulivat ilmi elementtien määrän kasvaessa. Seuraavaksi tarkastellaan Lighthousen avulla huomattuja eroavaisuuksia kuvaajien avulla.

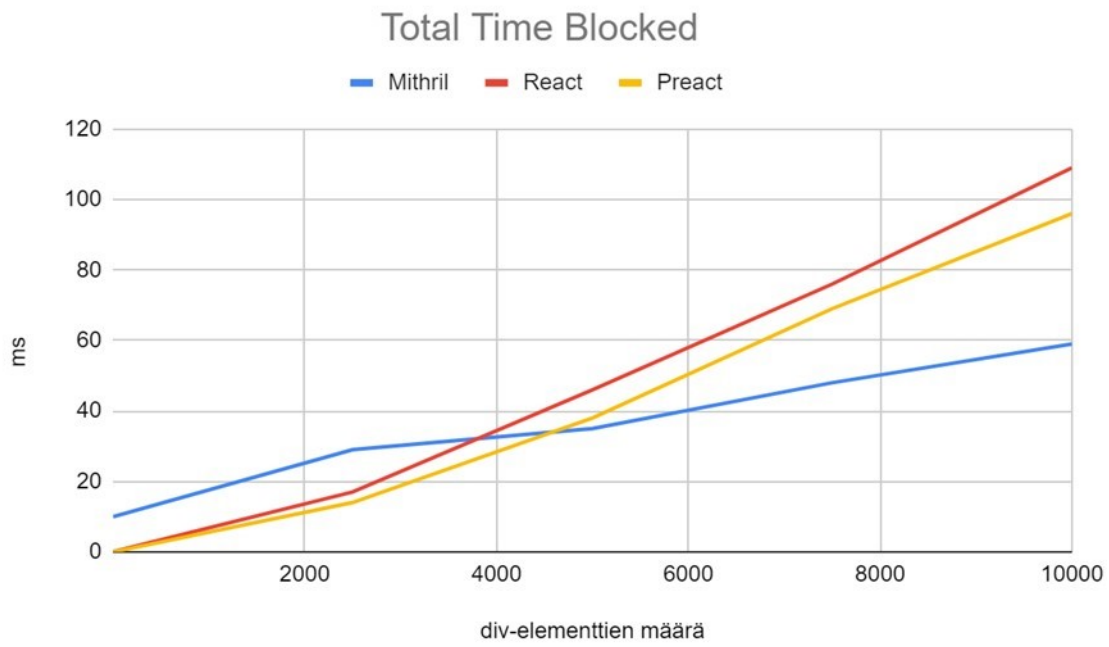
Kuvaajasta (kuva 8) huomataan, että Preactin ja Reactin kuvaajassa olevat käyrät kasvavat lineaarisesti lähempänä 2500 div-elementin jälkeen ja käyrien muodot muistuttavat toisiansa paljon. Kuitenkin lähemmäs 2500 div-elementin jälkeen käyrät nousevat jyrkästi ylöspäin. Mithrilin käyrä puolestaan kasvaa myös, mutta maltillisemmin. Johtopäätöksenä Mihtrilin suorituskyky voi olla parempi kuin

Reactin ja Preactin, koska Mithrillin tapa pävittää dokumenttioliomallia saattaa olla optimoitu paremmin.

Huomattavaa on myös kuvaajassa (kuva 8), että ennen noin 3800 div-elementtiä React.js renderöimä sivusto on pienemmän TBT-arvon takia käytettävissä ennen Mithrilin sivustoa. Kuitenkin 3800 div-elementin jälkeen Reactin käyrä nousee jyrkästi. Nousu kasvattaa viivettä ja on tällöin myöhemmin käytettävissä kuin Mithrilin renderöimä sivusto.

Puolestaan Preactin ollessa hieman nopeampi kuin Reactin niin kokonaisodotusaika alkaa Preactissa verratessa Mithriliin vasta 4300 divelementin jälkeen.

Tärkeää huomioida, että kyse on millisekunneista ja suurin arvo oli 109. Kaikki arvot ovat huomattavan pieniä ja näissä tapauksissa käyttäjä ei todennäköisesti huomaa eroa eri JavaScript-kirjastojen välillä.



Kuva 8. Havainnollistava kuva TBT (Total Time Blocked) JavaScript-kirjastojen, mitatuista arvoista.

Suurimman elementin lataus sivulla eli LCP (Largest Contentful Paint). Ladatut div-elementit olivat samanlaisia, joten suurempaa käyrän vaihtelua ei testeissä

tullut ilmi. Aikaisemmin kuvaajassa (kuva 8) huomataanPreactin ja Reactin TBT-arvojen olevan hitaampia kuin Mithrilin, kun div-elementtien määrä kasvoi.

Sama johtopäätös voidaan todeta tästäkin tapauksesta. Reactin käyrä poikkesi Mithrilistä ja Preactista, sillä sen FCP ja LCP-arvot pysyivät jatkuvasti samana.

7 Yhteenveto

Tämän insinöörityön tavoitteena oli vertailla JavaScript-kirjastoja React, Mithril ja Preact-ohjelmoijan näkökulmasta. Tarkoituksena on vertailla kolmea JavaScript-kirjastoa monesta eri näkökulmasta ja antaa ohjelmoijille ja aiheesta kiinnostuneille arvioita, sopisivatko ne omaan käyttöön.

JavaScript-kirjastoja vertailtiin ominaisuuksien, käyttöönoton, suorituskyvyn ja suosion kautta. Lopuksi myös käytiin lävitse niiden samankaltaisuuksia ja eroavaisuuksia. Ominaisuuksissa käytiin lävitse julkaisemispäivää, julkaisijaa ja kerrottiin niiden dokumenttioliopäivittämisen tavoista.

Käyttöönotossa tarkasteltiin, millä tavalla JavaScript-kirjasto asennetaan ja vaatiiko se erilaisten asetuksien määrittämistä vai pelkkää komentorivikäyttöliittymän käyttöä.

Suorituskykyä arvioitiin Lighthouse-sovelluksen avulla, jolla mitattiin divelementtien latautumista selaimeen ja kuinka kauan siinä kuluu aikaa riippuen eri JavaScript-kirjastoista. Tuloksista laadittiin taulukot ja kuvaajat. Lopuksi kuvaajia verrattiin keskenään. Vertailussa ilmeni samanlaisuuksia ja myös erilaisuuksia. Vaikka testi ei ollut toteutukseltaan monimutkainen, se tuotti erilaisia arvoja eri JavaScript-kirjastojen välillä, mistä sai alustavaa käsitystä niiden selaimen päivitysajoista.

Suosion vertailussa tarkasteltiin niiden latausmäärää npm-kirjastojen lataussivulta. (engl. Node package manager). Lisäksi suosiota tarkasteltiin myös GitHub-sivuston tähtien (tykkäyksien) kautta.

Lopuksi työssä käytiin lävitse samankaltaisuuksissa ja eroavaisuuksissa niiden yhteisistä piirteittäistä ja eroavaisuuksista. Vaikka niissä oli samankaltaisuuksia kuten kaikki hyödyntävät virtuaalista dokumenttioliomallia, niiden hyödyntämisessä oli kuitenkin eroa optimoinnin kannalta. Eroja tuli pakkauksien kokoeroissa ja ohjelmointiarkkitehtuurissa.

Jatkokehitysideana olisi rakentaa erilaisia sovelluksia ja tutkia näitä JavaScriptkirjastoja niiden avulla ja automatisoida Lighthouse-sovelluksen testaaminen. Lisäksi voisi hyödyntää myös muita testityökaluja tähän samaan tarkoitukseen.

Lähteet

1 A The DOM Explained for Beginners – How the Document Object Model Works. 2021. Verkkoaineisto. Freecodecamp.org. <https://www.freecodecamp.org/news/dom-explained-everything-youneed-to-know-about-the-document-object-model/>. Luettu 9.4.2023.

2 What is virtual DOM. 2017. Verkkoaineisto. Mithril.js.org. <https://mithril.js.org/vnodes.html/>. Luettu 20.4.2023.

3 What is React.js? 2022. Verkkoaineisto. blog.hubspot.com <https://blog.hubspot.com/website/react-js/>.

4 Create a React Base Project. 2021.Verkkoaineisto. <https://medium.com/nerd-for-tech/step-2-create-a-react-base-project-using-npx-create-react-app-84ee840ad130/>. Luettu 2.4.2023.

5 React. 2013. Verkkoaineisto. github.com. <https://github.com/facebook/react/>. Luettu 7.5.2023.

6 Mithril. 2015. Verkkoaineisto. npmjs.com. <https://www.npmjs.com/package/mithril/>. Luettu 1.4.2023.

7 MVC Framework - Introduction. 2015. Verkkoaineisto. tutorialspoint.com <https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm />. Luettu 7.5.2023.

8 Mithril installation. 2015. Verkkoaineisto. mithril.js.org. <https://mithril.js.org/installation.html/>. Luettu 7.5.2023.

9 Mithril. 2017. Verkkoaineisto. github.com. <https://github.com/MithrilJS/mithril.js/>. Luettu 7.5.2023.

10 Preact size. 2019. Verkkoaineisto. npmjs.com. <https://www.npmjs.com/package/preact/>. Luettu 1.4.2023.

11 Getting started Preact. 2020. Verkkoaineisto. preactjs.com. <https://preactjs.com/guide/v10/getting-started/>. Luettu 1.4.2023.

12 Preact. 2019. Verkkoaineisto. github.com. <https://github.com/preactjs/preact/>. Luettu 1.4.2019.

13 A Complete Guide to Flexbox. 2013. Verkkoaineisto. Web.archive.org. https://web.archive.org/web/20230000000000*/https://csstricks.com/snippets/css/a-guide-to-flexbox/>. Luettu 29.4.2023.

14 Lighthouse overview. 2022. Verkkoaineisto. developer.chrome.com <https://developer.chrome.com/docs/lighthouse/overview/>. Luettu 20.4.2023.

15 Lighthouse performance. 2022. Verkkoaineisto. developer.chrome.com. <https://developer.chrome.com/docs/lighthouse/performance/>. Luettu 20.4.2023.