Karelia-ammattikorkeakoulu
Tietojenkäsittely, tradenomi

Kuvapainoitteisten verkkosivujen optimointi Node.js-ympäristössä

Arttu Huusko

Opinnäytetyö, toukokuu 2022

www.karelia.fi

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Toukokuu 2022
Tietojenkäsittelyn koulutus

Tikkarinne 9
80200 JOENSUU
+358 13 260 600

Tekijä(t)
Arttu Huusko

Nimeke
Kuvapainoitteisten verkkosivujen optimointi Node.js-ympäristössä

Toimeksiantaja
6190 Pohjois-Karjalan ja Japanin Naganon metsäbiotalousyhteistyön kehittäminen 2020-22

Opinnäytetyön tavoitteena oli kuvata eri tapoja parantaa kuvapainoitteisten verkkosivujen latausaikaa Node.js-ympäristössä. Lisäksi työssä verrataan kolmen Node.js-paketin tehokkuutta valokuvien serveriltä viemien resurssien minimoimisessa verkkosivuilla. Opinnäytetyössä myös verrattiin kolmen eri tiedostoformaatin tiedostokokoja. Tulosten perusteella opinnäytetyössä tarjotaan ehdotuksia Koli virtual showroom -verkkosivun jatkokehitykselle.

Datana näiden kolmen Node.js-paketin vertailuun käytettiin kymmentä valokuvaa eri tiedostoformaateissa ja tiedostokoissa. Kaikissa testitapauksissa alkuperäiset kuvat kutistettiin ja pakattiin niin paljon kuin mahdollista menettämättä kuvan visuaalista laatua. Kuvat myös muunnettiin yhtenäiseksi tiedostoformaatiksi. Kaikki testitapaukset tehtiin kymmenen kertaa, jotta minimoitaisiin tietokoneella pyörivien taustaohjelmien vaikutusta kuvan käsittelyihin kuluneeseen aikaan.

Opinnäytetyön tulokset osoittavat, että Sharp-Node.js-paketti on muita testattuja paketteja tehokkaampi kuvanmuokkauksessa kaikilla testatuilla tiedostoformaateilla. Tiedostoformaateista huomattiin, että WEBP-tiedostot ovat keskimäärin pienempiä kuin JPEG- tai PNG-tiedostot.

Kieli
suomi

Sivuja 35
Liitteet 1
Liitesivumäärä 2

Asiasanat
WWW-sivut, ohjelmistokirjastot, tiedostomuodot

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2022
Degree Programme in Information Technology

Tikkarinne 9
FI 80200 JOENSUU
FINLAND
Tel. +350 13 260 600

Author(s)
Arttu Huusko

Title
Optimization of Picture Heavy Websites in a Node.js-environment

Commissioned by Forest Bioeconomy Development Co-operation (FBDC) 2020-2022 between North Karelia Region in Finland and Nagano Prefecture in Japan

This thesis aimed to outline some of the different ways to optimize a picture heavy website in a Node.js environment and compare the performance of three Node.js packages meant for editing and optimizing photos and pictures. A further goal was to compare the file sizes of three different image file formats. Finally, using the results of these test cases the thesis offers suggestions for the continued development of the Koli virtual showroom website.

The data for comparing the performance of the three selected packages was ten different pictures in varying file formats and file sizes. For each tested package all the original image files were made smaller and compressed as much as possible without losing visual image quality and converted to one uniform file format. These tests were performed ten times for each file format to reduce the effects on the processing time spent on the image handling from background programs running on the computer.

The results of the tests found that the Sharp Node.js package is more efficient at editing the photos in all the three file formats tested. It was also found that WEBP files are on average smaller than JPEG- or PNG files.

Language
Finnish

Pages 35
Appendices 1
Pages of Appendices 2

Keywords
file formats, libraries (computing), web pages

Sisältö

Sanasto

Time to live	Aika joka kuvaa verkon yli lähetettävän datan elinaikaa joko ajalla mitattuna, tai reititinhyppyjen lukumäärällä. Kun aika loppuu, datapakettia ei enää säilytetä
MIME-tyyppi	Multipurpose Internet Mail Extensions, tai Internet media type. Standardisoitu tapa luokitella tiedostotyyppejä.
REST	Representational state transfer. Arkkitehtuurityyli ohjelmointirajapintojen toteuttamiseen
API	Application Programming Interface, ohjelmointirajapinta
base64-data	Tiedostomuoto joka esittää binääri dataa 24 bitin sekvensseissä. Käyttötarkoituksena välittää binääri dataa tekstimuodossa formaateille, jotka tukevat vain tekstidataa. Esimerkiksi sisällyttämään kuvatiedostoja HTML-tiedostoihin.
blob-muoto	binary large object, Binääri dataa tallennettuna yhdeksi kokonaisuudeksi

1 Johdanto

Verkkosivut nykyään ovat monimutkaisia kokonaisuuksia jotka koostuvat monista eri osista ja osa käyttäjistä odottaa verkkosivujen latautuvan nopeasti ja olemaan käytettäviä sekuntien sisällä siitä, kun he avaavat verkkosivun. Verkkosivuilla joilla käytetään paljon kuvia tai joissa kuvia täytyy pitää tarpeeksi suurella resoluutiolla, jotta kuvan laatu säilyy hyvänä, täytyy verkkosivujen kehittäjien tasapainoitella sivun latausaikaa ja kuvien laatua eri tavoin, jotta käyttäjät eivät poistu sivulta ennen kuin edes näkevät sen toiminnassa.

Node.js-ympäristössä npm-paketteja käyttäen voi verkkosivun kehittäjä säästää omaa työtaakkaansa käyttämällä muiden kehittäjien tekemiä toiminnallisuuksia, mutta joissakin tilanteissa samalle asialle on monia lähestymistapoja. Uusien kehittäjien voi olla vaikea tietää, että mitä pakettia käyttäen saavutetaan parhaimpia tuloksia esimerkiksi kuvankäsittelyyn liittyvissä asioissa, sillä mahdollisia muuttujia kuvankäsittelyn paremmuuteen on monia.

Tässä opinnäytetyössä kuvataan kuinka Node.js- ja React-ympäristöä käyttävä valokuvapainotteisen verkkosivun valokuvia voidaan käsitellä ja sitä, mitä toimenpiteitä kuvalle voidaan tehdä sen elinkaaren aikana. Lähtien siitä miten käyttäjä lataa sivustolle kuvan, mitä toimenpiteitä kuvalle tehdään ennen sen lähettämistä palvelimelle, jossa kuva säilytetään. Mitä toimenpiteitä kuvalle tehdään ennen sen tallentamista palvelimelle sekä mitä toimenpiteitä kuvalle tehdään ennen kuin se lähetetään palvelimelta käyttäjälle tämän vieraillessa verkkosivustolla.

Näitä asioita tarkastellaan Karelian Japani hankkeen Koli virtual showroom -verkkosivun pohjalta, jonka kehitysvaiheessa kehittäjälle nousi kysymykseksi minkälaisia kaikkia asioita tulisi verkkosivun kehittäjän ottaa huomioon kuvapainotteisessa verkkosivustossa, jotta sivuston latausaika pysyy käyttäjälle tarpeeksi nopeana, jotta tämä haluaisi vierailla sivustolla useamminkin.

Opinnäytetyön tavoitteena teknisten optimointitekniikoiden kuvaamisen lisäksi on myös testata kolmen eri Node.js-paketin välillä ja kolmen eri tiedostotyypin välillä, mitä Node.js-pakettia käyttäen kuvia kannattaisi optimoida verkkosivua

varten ja missä tiedostotyypissä kuvat kannattaisi palvelimella säilyttää. Sekä tarjota kehitysehdotuksia Koli virtual showroom -verkkosivulle edellä mainittujen asioiden pohjalta

2 Yleisiä verkkosivuilla käytettyjä kuvien optimoinnin tekniikoita

2.1 Miksi kuvien optimointi tärkeää verkkosivuilla

Kuvat sekä videot ovat nykypäivän internetissä hyvin suuressa käytössä. Vuonna 2019 Web Almanac raportoi, että verkkosivuilla keskimäärin jopa kaksi kolmesta lähetetystä bitistä olevan visuaaliseen mediaan liittyviä bittejä (Bendell & Sillars 2019). Myös Google raportoi vuonna 2016, että 53 % vierailuista verkkosivuilla hylättiin, jos verkkosivun lataaminen mobiililaitteella kestää pidempään kuin kolme sekuntia. (Think With Google 2016). Tämän takia verkkosivujen kehittäjien on tarpeellista kiinnittää huomiota visuaalisen median käsittelyllisiin seikkoihin säästääkseen palvelimen sekä verkkosivun prosessointiaikaa ja säilyttääkseen visuaalisen median tarpeeksi hyvälaatuisena, jotta se toisi verkkosivun käyttäjälle toivotun kokemuksen. Visuaalisen median optimointi leikkaa myös verkkosivujen kustannuksia. Palvelimella käytetty tila visuaalisen median säilyttämiseksi pienenee, kuin myös verkkosivun käyttämä internetkaistan leveys kutistuu.

2.2 Tekniset keinot verkkosivuston käytön optimoimiseen

2.2.1 Eri tiedostotyyppien vaikutus kuvien viemiin resursseihin verkkosivuilla

Verkkosivuilla esitetyt kuvat koostuvat pääasiassa viidestä eri tiedostoformaatista, joista kaikilla on omat hyvät ja huonot puolensa eri tarkoituksissa. Nämä formaatit ovat JPEG, PNG, GIF, WEBP ja SVG. Web

Almanac kuvaa näiden yleisesti käytettyjen tiedostotyyppien hyviä ja huonoja puolia taulukko 1:n mukaisesti. (Bendell & Sillars 2019)

Taulukko 1. Kuvaus eri tiedostotyypeistä (Bendell & Sillars 2019)

Kuvaformaatti	Positiiviset	Negatiiviset
JPEG	Tuettu kaikkialla. Ideaalinen valokuville.	Kuvan laatu huononee aina tallennettaessa, eli on häviöllinen. Suurin osa dekoodereista ei tue suuren bittisyvyyden valokuvia moderneista kameroista (>8 bittiä per. kanava). Ei tue läpinäkyvyyttä.
PNG	Kuten JPEG ja GIF, on tuettu laajasti. Häviötön. Tukee läpinäkyvyyttä, animaatiota sekä suurta bittisvyvyyttä.	Paljon suurempi tiedostokoko kuin JPEG Ei ideaali valokuville.
GIF	PNG:n edeltäjä, eniten tunnettu animaatioista. Häviötön	Tukee vain 256 väriä, joten visuaalinen ilme kärsii aina tallennettaesssa. Erittäin suuret tiedostokoot animaatioissa.
SVG	Vektoreihin perustuva formaatti jonka kokoa voidaan muuttaa ilman tiedostokoon muuttamista. Perustuu matematiikkaan eikä pikseleihin ja luo puhtaita viivoja.	Ei hyödyllinen valokuviin tai muuhun rasteroituun sisältöön

WEBP	Voi tuottaa sekä häviöttömiä kuvia samoin kuin PNG, sekä häviöisiä kuvia samoin kuin JPEG. Pienempi tiedostokoko kuin JPEG kuvissa.	

Web Archive kertoo julkaisussaan myös, että 65 % verkkosivujen ladatuista kuvabiteistä on JPEG-muodossa. Toisella sijalla kuvabittien käytössä on PNG-kuvat, jotka vievät 28 % verkkosivujen kuvabiteistä. Näiden kahden jälkeen SVG, GIF ja WEBP vievät kaikki kukin noin 4 % verkkosivujen kuvabiteistä. (Bendell & Sillars 2019)

2.2.2 Lazy loading -tekniikka verkkosivujen kuvien latautumisessa

Lazy loading on tekniikka jolla tarkoitetaan, että verkkosivuilla olevat elementit ladataan vasta sitten, kun ne näkyisivät käyttäjälle. Web almanac kertoo julkaisussaan, että mediaani verkkosivulla 27 % sivulla olevista kuvista on kerrallaan käyttäjän ruudun ulkopuolella (Bendell & Sillars 2019).

Kehittäjän näkökulmasta lazy loading -tekniikoita voidaan käyttää erittelemällä Javascript-, CSS- ja HTML -tiedostot pienemmiksi palasiksi, jotta käyttäjälle lähetetään vain sillä hetkellä tarvittavia tiedostoja nopeuttaen sivuston latausta. Loput sivuston toiminnoista voidaan ladata sivuston alkunäkemän latautumisen jälkeen, tai kun käyttäjä tekee sivustolla jotain joka vaatii muiden scriptien toimintoja. Kehittäjä voi myös kuvien tapauksessa määritellä tietyille HTML-elementeille loading-attribuutin kertomaan selaimelle, että kuvaa ei tarvitse ladata ennen kuin käyttäjä navigoi verkkosivulla näkymänsä kuvan kohdalle. (Kuva 1) (MDN Web Docs 2021).

```
<img src="image.jpg" alt="..." loading="lazy">
<iframe src="video-player.html" title="..." loading="lazy"></iframe>
```

Kuva 1. Lazy loading -tekniikka kuvan ja videon lataamisessa.

Esimerkkinä tämän opinnäytetyön tarkastelemassa verkkosivussa lazy loading tulee esille siten, että kun käyttäjä ensimmäistä kertaa avaa sivun. Joutuu käyttäjä lataamaan vain päänäkymän. Sivulla esiteltyjen yritysten tarkemmat tiedot sekä niihin liittyvät toiminnot ja kuvat latautuvat vasta, kun käyttäjä klikkaa jonkin yrityksen esittelyikkunan auki.

2.2.3 Browser Caching -tekniikka verkkosivujen käytössä

Kun käyttäjä vierailee verkkosivuilla, käyttäjän verkkoselain tallentaa tiettyjä osia verkkosivusta käyttäjän tietokoneen muistiin. Selain tallentaa välimuistiin verkkosivulta asioita kuten HTML-tiedostoja, Javascript-tiedostoja ja kuvia, jotta seuraavalla kerralla, kun käyttäjä avaa verkkosivun voidaan verkkosivun tiedot ladata käyttäjän tietokoneen välimuistista eikä verkkosivuja palvelevalta palvelimelta. Verkkoselaimet säilyttävät muistissa tiedostoja, kunnes selaimen välimuisti on täynnä, käyttäjä manuaalisesti poistaa selaimen välimuistista tiedot, tai ladatun tiedoston time to live -aika on kulunut. (What is caching? 2022.)

Verkkoselaimen muistia käyttävää caching-tekniikkaa kannattaa käyttää niissä osissa verkkosivua jotka eivät muutu usein. Asiat kuten verkkosivun logo ei todennäköisesti tule verkkosivuilla muuttumaan usein, jos ollenkaan. Kun taas mahdollisesti usein muuttuvat osat kuten verkkosivun toimivuuden takaavat javascript-tiedostot voivat olla jopa haitaksi, jos ne ladataan selaimen välimuistista sen jälkeen, kun niihin on palvelimella julkaistu uudempi versio. (Gash 2018).

Verkkosivun kehittäjän tulee kuitenkin ottaa huomioon, että kun käyttäjä ensimmäistä kertaa vierailee sivustolla ei verkkoselaimen muistin käyttö ole mahdollista, koska selaimella ei tämän verkkosivun tiedostoja ole. Myöskin on mahdollista, että käyttäjä on itse poistanut selaimen välimuistin tai asettanut sen

tulevan poistettavaksi automaattisesti. Kuitenkin verkkosivuilla joilla odotetaan käyttäjien vierailevan useasti on verkkoselaimen välimuistin käyttö hyväksi sivuston latautumisen nopeuttamiseksi palaaville käyttäjille. (Gash 2018.)

Browser-caching tekniikässä on pääasiassa käyttäjän verkkoselaimesta riippuvainen ja verkkosivun kehittäjällä on vähän vaikutusvaltaa siihen, kuinka verkkosivustolla selaimen välimuistia käytetään.

2.2.4 CDN caching -tekniikka verkkosvujen käytössä

CDN eli Content Delivery Network -palvelut voivat tallentaa sisältöä kuten kuvia, videoita, tai verkkosivuja välityspalvelimilla ympäri maailmaa. Välityspalvelinta voidaan käyttää palvelimena joka vastaanottaa käyttäjältä pyynnön ja edelleenlähettää kyseisen pyynnön muille servereille verkostossa. Koska CDN-verkossa olevia palvelimia voi olla ympäri maailmaa pystyy CDN-palveluita käyttävä verkkosivu latautumaan käyttäjälle nopeammin. (What is caching? 2022.) CDN-palveluiden käyttöönotto ja hinnoittelu vaihtelevat eri palveluntarjoajien välillä ja vaatii kehittäjän lisäksi verkkosivun hallinnoijalta käyttöönottoon toimenpiteitä. Esimerkiksi Amazon (2022) ja Cloudfare (2022) tarjoavat ilmaisia versioita tietyillä ehdoilla kuten datakatolla, jonka jälkeen palvelut ovat maksullisia.

2.2.5 Format switching -tekniikka verkkosivujen kuvien käytössä

Verkkosivuilla voidaan HTML Picture -tagin avulla määritellä yhdelle kuva elementille eri vaihtoehtoja siitä, missä formaatissa käyttäjän verkkoselain lataa kyseisen kuvan. Esimerkiksi WEBP-formaatissa olevia kuvia käytettäessä eivät jotkut vanhat selaimet tue kyseistä formaattia (Google 2022.) Tässä tilanteessa picture-tagia käyttämällä voidaan niillä selaimilla jotka eivät tue esimerkiksi WEBP-formaattia ladata käyttäjälle kyseinen kuva jossain toisessa formaatissa. (kuva 2) Myöskin monet CDN-palvelut voivat tarjota format switching -tekniikalle omaa toiminnallisuuttaan. (Piros & Seymour & Portis, 2020).

```
<picture>
    <source srcset="logo.webp" type="image/webp">
    <img src="logo.png" alt="logo">
</picture>
```

Kuva 2. Koodipätkä jolla määritellään, että jos käyttäjän selain ei tue WEBP-formaattia, ohjelmisto lataa käyttöön PNG-tiedostotyypin logon.

3 Opinnäytetyössä käytetyt menetelmät

3.1 Työssä käytetty toimintaympäristö

Työssä on käytetty React-ohjelmointikieltä käyttäjälle näkyviin toiminnallisuuksiin. Tämän lisäksi kuvankäsittelyllisiä toimintoja varten on käytetty Node.js-ohjelmointikieltä sekä siihen olemassa olevia paketteja. Kuvien keräämiseen käyttäjältä käytetään formic-pakettia (Formium 2020). Kuvien lähettämisessä palvelimelta käyttäjälle tämän vieraillessa sivustolla käytetään path-pakettia (Nodejs 2022).

Ennen opinnäytetyössä tehtyjä testausvaiheita verkkosivulla käytettiin react-image-file-resizer-pakettia (Zorluer & al. 2021). Testivaiheessa testataan edellämainitun paketin lisäksi myös sharp-pakettia (Pixelplumbing 2022) sekä imagemagick-pakettia (Imagemagick 2013). Näihin kolmeen valittuun pakettiin päädyttiin, koska koli virtual showroom -verkkosivua kehitettäessä nämä kolme nousivat eniten esille kehittäjän etsiessä internetistä mitä tekniikoita kuvien muokkaamiseen on Node.js-ympäristössä. Näistä kolmesta React-image-file-resizer-paketti tekee kuvanmuokkauksen käyttäjän puolella verkkoselaimen avulla, kun taas Sharp-paketti ja Imagemagick-paketti suorittavat kuvanmuokkauksen Node.js-ympäristössä palvelimen puolella.

Testivaiheissa käytetyllä tietokoneella pyöritetään verkkosivua paikallisesti, jotta internet yhteyden nopeus ei vaikuttaisi testivaiheessa ajassa mitattuihin tietoihin. Tietokone jolla testivaiheet tehdään on Windows 10 -alustalla, jonka prosessorin nopeus 2.4Ghz ja siinä on 16GB RAM-tilaa. Verkkosivua käytetään testausvaiheiden aikana kuvia ladattaessa Chrome-selaimella.

3.2 Työssä käytetty testidata

Työn testivaiheessa testataan aiemmin 3.1-luvussa mainittujen kolmen eri Node-paketin kuvankäsittelyihin viemää aikaa sekä valokuvan tiedostotyypin vaikutusta kuvien lopulliseen tiedostokokoon. Testeissä käytetään kymmentä erilaista kuvitteellisen yrityksen esittelyvalokuvaa. Alkuperäisistä valokuvista kuusi on PNG-muodossa ja loput neljä JPEG-muodossa. Kuvien tiedostokoot vaihtelevat noin 1,6MB ja 12,2MB välillä ja niiden yhteiskoko on noin 39,5MB.

3.3 Työssä käytetty testausprosessi

Koska kuvankäsittelyyn kuluvaan aikaan voi vaikuttaa monet tekijät, kuten esimerkiksi tietokoneella pyörivän käyttöjärjestelmän prosessit. Jokaisessa testivaiheessa kuvien käsittelyn nopeuden testaaminen suoritetaan kymmenen kertaa ja huomioon otetaan näiden kertojen mediaani, jotta minimoitaisiin verkkosivuston ulkopuolisten tekijöiden vaikutusta mitattuihin tuloksiin. Mediaanin käyttö parantaa saatujen tulosten tarkkuutta verrattuna keskiarvon käyttöön, sillä tapauksissa joissa on yksittäisiä erittäin suuria tai erittäin pieniä käsittelyaikoja testien tuloksina, voi keskiarvon käyttö tehdä kokonaistuloksista harhaanjohtavia.

Testit suoritetaan paikallisesti pyörivällä verkkoympäristöllä joten verkon käyttöä ja sen vaikutusta kuvien latausaikoihin ei oteta huomioon kuvien latauksessa käyttäjältä palvelimelle. Lopullisten muokattujen kuvien kokoa tarkastellessa otetaan kuitenkin huomioon verkkosivun kaistan käyttö lähetettäessä kuvia palvelimelta käyttäjälle joka vierailee sivustolla.

4 Kuvan lataus, käsittely ja tallennus Koli virtual showroom verkkosivun ympäristössä

4.1 Koli virtual showroom -verkkosivu

Sivuston päänäkymässä käyttäjälle näkyy heti eri yrityskategorioihin eroteltuja palluroita. Kun käyttäjä vie hiiren yhden palluran päälle, tarjotaan käyttäjälle yleistietoa minkä yrityksen tiedot kyseisen palluran alle on kirjoitettu (Liite 1). Käyttäjän klikatessa yhtä palluraa avataan tämän yrityksen tarkemmat tiedot pienessä ikkunassa, jossa näkyy yrityksen esittelytekstit ja sivustolle ladatut esittelykuvat yrityksestä. Käyttäjän klikatessa yhtä kuvaa, aukeaa vielä uudessa ikkunassa kyseinen kuva suuremmassa koossa. (Liite 1)

4.2 Käyttäjän kuvan lataaminen sivustolla

Sivustolla on käyttäjälle tarkoitettu React-ohjelmointikielellä tehty lomake, jonka täyttämällä kyseinen käyttäjä voi lisätä hankkeen yhteistyötahojen yritysesittelyitä sivustolle (Liite 1). Yhtenä osana yritysesittelyä ovat yrityksen logo tai logot, jos yrityksellä on eri logot sivuston tukemien kielien välillä. Tämän lisäksi sivustolla on yleisiä esittelykuvia yrityksestä. Lomake hyväksyy logojen ja kuvien lähettämiseen vain tiedostoja joiden MIME-tyyppi on ”image”. (Kuva 3) Kyseisiä tiedostotyyppejä on yli 50 (Internet Assigned Numbers Authority 2022). Joten tässä työn testivaiheessa keskitytään kahteen yleisimmistä kuvatiedostotyypeistä kuten PNG ja JPEG, jotka ovat yhteensä vieneet vuonna 2019 93 % kaikista verkkosivujen kuvabiteistä (Bendell & Sillars 2019). Sekä WEBP formaattia joka on vuonna 2020 saavuttanut tuen kaikilla verkkoselaimilla ja on vuonna 2021 ollut neljäntenä internetin ladattujen kuvabittien käytössä 6.9 % määrällä (Portis & Sillars 2021).

```
<Label>
  {LanguageManager.getTranslation("info-image")}
  <>
    {values.images.length > 0 &&
      values.images.map((image, index) => (
        <Row key={index}>
          <FileInput
            type="file"
            name={`images.${index}`}
            accept="image/*"
            onChange={(event) => {
              {
                addThumbnailToUpload(thumbnailsToUpload.concat( event.currentTarget.files[0]));
                setFieldValue(
                  `images.${index}`,
                  event.currentTarget.files[0]
                );
              }
            }}
          />
        </Row>
      ))}
  </>
</Label>
```

Kuva 3. Verkkosivuston toiminnallisuus käyttäjän lataamien kuvien keräämiseen.

4.3 Kuville tehtävät toimenpiteet ennen palvelimelle lähettämistä

Verkkosivun alkuperäisessä toteutuksessa sekä ensimmäisessä testivaiheessa ennen kuin kuvatiedostot lähetetään palvelimelle on tarkoituksena tehdä jokaisesta kuvatiedostosta pienempi sekä tiedoston koossa, että pikselien määrässä ja täten sopivampi verkkosivuston käyttöliittymään sekä optimoimaan palvelimen tilankäyttöä. Tästä kuvanmuokkaus prosessista kerrotaan tarkemmin node-pakettien toiminnallisuuksien testivaiheessa luvussa 4.5. Testivaiheissa kaksi ja kolme kuvankäsittely kuitenkin suoritetaan vasta palvelimella, joten näissä tapauksissa ainoat toimenpiteet joita kuville tehdään ennen lähettämistä palvelimelle on, että ne muunnetaan sopivampaan muotoon REST API -pyyntöä varten jolla ne lähetetään palvelimelle.

Kun käyttäjä painaa lomakkeen lopussa olevaa lähetä painiketta alkaa varmistus, että lomakkeeseen syötetyt tiedot ovat sopivia sivustolle lisäämistä varten. Lisättyjen kuvien tarkistus alkaa varmistamalla, että kuvatiedostojen yhteinen koko on alle 100MB ja, että ladattavat tiedostot ovat oikeaa tiedostotyyppiä. Tiedostojen tyyppi varmistetaan kuvatiedostoksi tarkastamalla, että tiedoston MIME-tyyppi on image. Tarkastus tehdään Node.js:n Yup-paketin

avulla. (Kuva 4) Jos toinen ehdoista ei toteudu, tulee lomakkeeseen kyseisen kohdan vierelle teksti joka selittää minkä takia tiedoston lähettäminen ei onnistu. Tiedoston koon sekä tyypin tarkastamisen jälkeen ja olettaen, että lomakkeen muut kohdat on täytetty oikealla tavalla, kuvatiedostot lähetetään palvelimelle käsiteltäväksi. (Kuva 5)

```
logoFi: Yup.mixed()
  .required(`${LanguageManager.getTranslation("info-compulsory")}`)
  .test(
    "fileSize",
    "File has to be less than 100MB", //virheilmoitus jos testiä ei läpäistä
    (value) => value && value.size <= 100000000 //testi tiedoston koosta
  )
  .test(
    "fileFormat",
    "Unsupported Format", //virheilmoitus jos testiä ei läpäistä
    (value) => value && value.type.includes("image/") //testi tiedoston MIME-tyypistä
  ),
```

Kuva 4. Käyttäjän lataaman suomenkielisen logon tarkistustoiminnot verkkosivustolla

```
const response = await fetch('http://localhost:8080/api/addCompany', { //REST API -pyyntö
  method: 'POST',
  headers: {
  'Accept': 'application/json',
  'Content-Type': 'application/json'
  },
  body: JSON.stringify({ //lomakkeeseen syötetyt tiedot JSON-muodossa
    form: values,
    logoFi: fiLogoFileData,
    logoEn: enLogoFileData,
    logoJp: jpLogoFileData,
    thumbnails: thumbnailFilesStringData,
    addressName: addressNames,
    addressCoordinate: addressCoordinates,
  })
});
```

Kuva 5. Verkkosivuston lomakkeeseen syötettyjen tietojen lähetys REST API-pyynnöllä.

4.4 Kuvalle tehtävät toimenpiteet palvelimella ja sen tallennus

Kun palvelin saa pyynnön lisätä uuden yrityksen tiedot ja kuvat, kuvien käsittelyprosessi alkaa lukemalla lähetettyjen kuvien base64-datan ja tarkastaa vielä onhan lähetetty tiedosto jokin tuetuista tiedostotyypeistä. (Kuva 6) Tämän

jälkeen ohjelmisto luo kansiorakenteeseensa public-nimiseen kansioon uuden kansion, jonka nimi on CompanyX jossa X on se numero monentena yritys on lisätty ohjelmiston tietokantaan miinus yksi, koska laskeminen aloitetaan nollasta. Ohjelmisto tallentaa logon tai logot kyseiseen kansioon nimellä XLogoFi, XLogoEn, tai XLogoJp jossa X on sama kuin kansion nimessä (Kuva 7) jos englannin- tai japaninkielistä logoa ei ole lomaketta täyttäessä annettu, korvataan sen kielinen logo suomenkielisellä logolla.

```
//erotellaan lähetetystä Base64-datasta kuvan tiedostotyyppi
const ext = req.body.thumbnails[i].substring(req.body.thumbnails[i].indexOf("/")+1, req.body.thumbnails[i].indexOf(";base64"));
if(ext === "jpeg" || ext === "png" || ext === "webp") { //vielä kerran tiedostotyypin tarkastus
  }
  else {
    res.status(415).send(); //lähetetään takaisin virheilmoitus jos tiedostotyypin tarkastus ei onnistu
    return;
  }
}
```

Kuva 6 Base64-datan lukeminen ja tiedostotyypin tarkastus.

```
const fileType = req.body.thumbnails[i].substring("data:".length,req.body.thumbnails[i].indexOf("/"));
const regex = new RegExp(`^data:${fileType}\/${ext};base64,`, 'gi');
const base64Data = req.body.thumbnails[i].replace(regex, "");
//erotellaan base64-data
const filename = `${chosenCompanyID}thumbnail${i}.${ext}`;
//määritellään tiedoston nimi
fs.writeFileSync(__dirname + "/public/" + "Company"+chosenCompanyID + "/" + filename, base64Data, 'Base64');
//Tiedosto tallennetaan sille kuuluvaan kansioon
thisThumbnailFilePath = "http://localhost:8080/api/"+ "Company"+chosenCompanyID + "/" + filename;
//tietokantaan tuleva polku kuvan hakemiselle
```

Kuva 7. Ohjelmiston toiminnot kuvat sisältävän kansion ja kuvatiedoston tekemiseen.

Samat tarkastukset ja prosessit tehdään myös yrityksen esittelykuville, poikkeuksena se, että tiedoston nimeksi laitetaan XthumbnailN, jossa N on monesko esittelykuva sillä hetkellä käsiteltävä kuva on aloittaen laskeminen taas nollasta sekä X sama numero kuin kyseisen yrityksen logossa. Näiden käsittelyjen jälkeen tietokantaan tallennetaan kyseisen yrityksen kohdalle logojen sekä esittelykuvien kohtaan niiden paikallinen tiedostopolku josta kuvat haetaan, kun käyttäjä vierailee sivustolla. (kuva 7)

4.5 Käyttäjälle kuvan lähettäminen

Kun käyttäjä vierailee sivustolla, sivusto hakee kaikki sillä hetkellä sivuston asetuksissa päällä olevan kieliset logot pyytämällä ensin tietokannasta yrityksen kohdalle tallennetun paikallisen tiedostopolun ja käyttämällä sitä REST API -pyyntöön, joka palauttaa halutun yrityksen logon sivustolle ja asettelee sen sivustolla sille kuuluvalle paikalle. Yrityksien esittelykuvat haetaan vasta, kun käyttäjä avaa yhden yrityksen tarkemmat tiedot klikkaamalla tälle yritykselle määritettyä palluraa. Tällä tavoin käyttäjän ei tarvitse kaikkia kuvia ladata heti, kun tämä ensimmäistä kertaa avaa sivuston, nopeuttaen sivuston latausaikaa ja pienentäen palvelimelta työtaakkaa.

4.6 React-image-file-resizer-node paketin toiminta

4.6.1 React-image-file-resizer-paketin laajuus

Ensimmäisenä pakettina opinnäytetyön testivaiheessa oli React-image-file-resizer-paketti. Paketti on noin 17kB kooltaan ja ei sisällä riippuvuuksia muihin node-paketteihin joten se ei kehittäjän näkökulmasta kuluta paljoa resursseja. Paketilla pystyy muokkaamaan kuvista leveyttä, korkeutta, tiedostoformaattia, kiertoa ja laatua. Tiedostoformaateista React-image-file-resizer tukee JPEG, PNG ja WEBP formaatteja. Muokatun lopputuloksen paketti voi tuottaa joko base64 tai blob-muodossa tai tiedostona.

4.6.2 React-image-file-resizer-paketin käyttöönotto

React-image-file-resizer-paketin käyttöönotto oli mielestäni suoraviivaista. Kuvatiedoston muokkaus vaatii vain yhden funktion kutsumisen, johon syötetään tarvittavat parametrit.(kuva 8) Pakollisina syötettyinä tietoina toimiakseen React-image-file-resizer-paketin funktio vaatii alkuperäisen tiedoston, muokatun kuvan maksimi leveyden sekä muokatun kuvan maksimi korkeuden, lopputuloksen tiedostotyypin, kuvan pakkauksen määrän ja kuvan kierron. Näiden pakollisten parametrien lisäksi tämän projektin toimintojen takaamiseksi vaaditaan myös, että muokattu lopullinen kuva palautetaan

base64-muodossa, jotta se voidaan lähettää yrityksen muiden tietojen mukana REST API -pyyntönä palvelimelle.

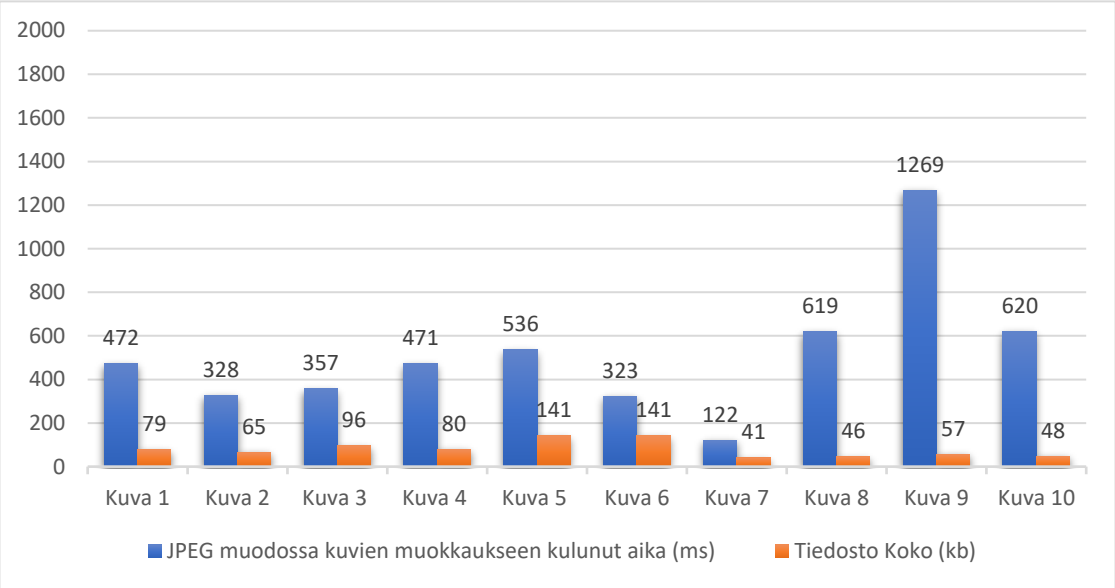
```
const handleFileReSizing = (file) => new Promise((resolve) => {
    Resizer.imageFileResizer(
      file, //alkuperäisen tiedoston polku
      1280, //maksimi leveys pikseleinä
      720, //maksimi korkeus pikseleinä
      "JPEG",//tiedostotyyppi
      65, //pakkaus
      0, //kierto
      (uri) => {
        resolve(uri);
      },
      "base64" //muokatun lopputuloksen palautettava datatyyppi
    );
})
```

Kuva 8. React-image-file-resizer-paketin kuvanmuokkaus funktio, testivaiheessa käytettyjen parametrien kanssa JPEG-tiedostotyyppinä.

4.6.3 React-image-file-resizer-paketin kuvankäsittelyn tulos

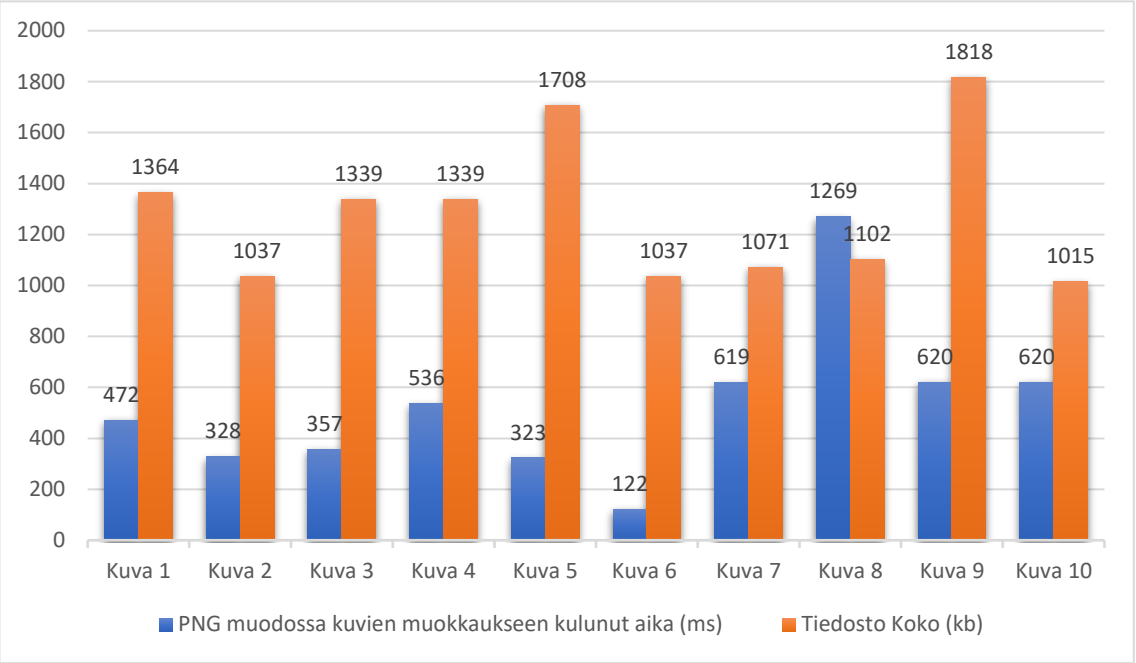
React-image-file-resizer-paketin ensimmäisessä testissä alkuperäiset kuvat pidettiin JPEG-muodossa. Kuvien käsittelyn jälkeen kuvien koko palvelimella oli yhteensä 999KB. Kuvien muokkaukseen kulunut aika oli yhden kuvan kohdalla kymmenen testikerran jälkeen mediaanina pienimmillään 122 millisekuntia ja suurimmillaan 1269 millisekuntia. Yhteensä kaikkien kuvien muokkaukseen kulunut mediaani aika oli 5117 millisekuntia (taulukko 2). Koko prosessiin siitä, kun käyttäjä painaa lähetä nappia, siihen että palvelin vastaa onnistuneesti tallentaneensa uuden yrityksen tiedot ja kuvat kesti mediaanina 5382 millisekuntia.

Taulukko 2. JPEG muodossa kuvien muokkaukseen kuluneiden aikojen mediaani kymmenestä kerrasta.



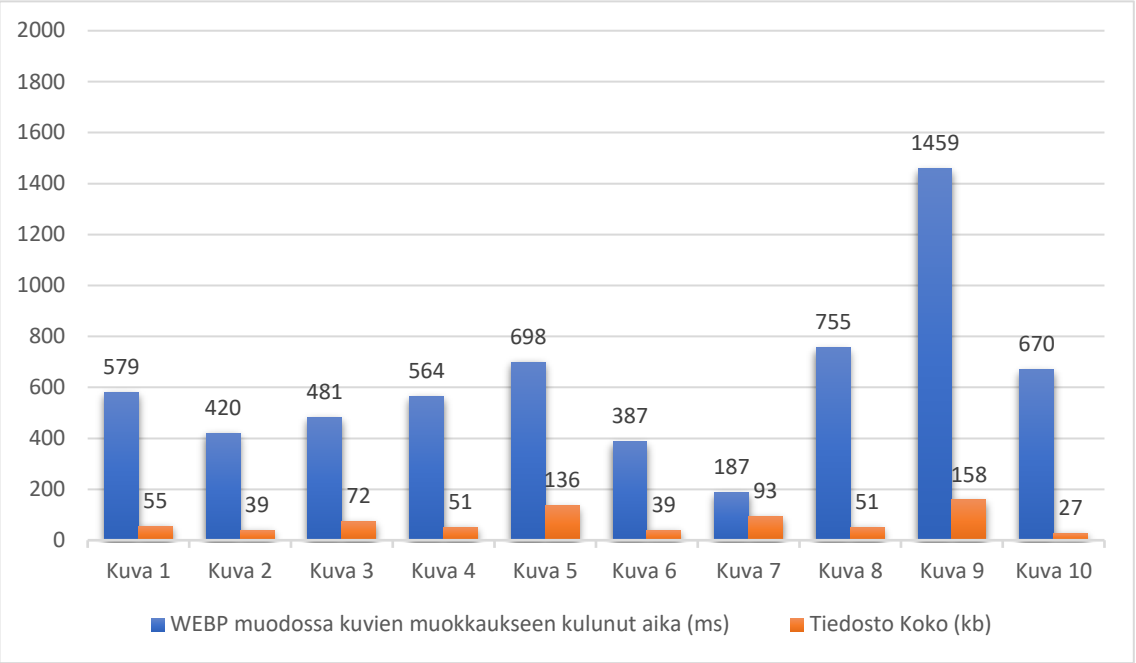
Toisessa testissä muokattiin alkuperäiset kuvat PNG-muotoon. Muokattujen kuvien koko yhteensä oli 12,5MB. Kuvien muokkaukseen kulunut aika oli mediaanina yhden kuvan kohdalla kymmenen testikerran jälkeen pienimmillään 123 millisekuntia ja suurimmillaan 1269 millisekuntia. Yhteensä kaikkien kuvien muokkaukseen kulunut mediaani aika oli 6053 millisekuntia (taulukko 3). Koko prosessiin siitä, kun käyttäjä painaa lähetä nappia, siihen että palvelin vastaa onnistuneesti tallentaneensa uuden yrityksen tiedot ja kuvat kesti mediaanina 6602 millisekuntia.

Taulukko 3. PNG muodossa kuvien muokkaukseen kuluneiden aikojen mediaani kymmenestä kerrasta.



Kolmannessa testissä alkuperäiset kuvat muokattiin WEBP-muotoon. Muokattujen kuvien koko yhteensä oli 716KB. Kuvien muokkaukseen kulunut aika oli mediaanina yhden kuvan kohdalla kymmenen testikerran jälkeen pienimmillään 187 millisekuntia ja suurimmillaan 1459 millisekuntia. Yhteensä kaikkien kuvien muokkaukseen kulunut mediaani aika oli 6223 millisekuntia (taulukko 4). Koko prosessiin siitä, kun käyttäjä painaa lähetä nappia, siihen että palvelin vastaa onnistuneesti tallentaneensa uuden yrityksen tiedot ja kuvat kesti mediaanina 6255 millisekuntia.

Taulukko 4. WEBP muodossa kuvien muokkaukseen kuluneiden aikojen mediaani kymmenestä kerrasta.



React-image-file-resizer-paketin testeissä JPEG-muodossa kuvien käsittelyyn meni vähiten aikaa ja kuvien lopulliset koot olivat yli puolet pienemmät kuin alkuperäiset. PNG-muodossa kuvien koko kasvoi huomattavasti. Kuvien muokkaukseen käytetty aika kasvoi myös JPEG-muodossa olleisiin kuviin verrattuna. WEBP-muodossa muokatut kuvat olivat vielä pienemmät kuin JPEG-muodossa olleet kuvat. Kuitenkin niiden muokkaamiseen meni näistä kolmesta eri kuvankäsittelyn testistä eniten aikaa.

4.7 Sharp-paketin toiminta

4.7.1 Sharp-paketin laajuus

Toisena pakettina opinnäytetyön testivaiheessa oli Sharp-paketti. Paketti on noin 20MB kooltaan asennettaessa. Tiedostoformaateista sharp tukee JPEG, PNG, WEBP, GIF ja AVIF formaatteja. Sharp-paketti voi myös antaa muokatun tuloksen raakana pikseli datana. Sharp-paketilla on mahdollista suorittaa huomattava määrä eri toimintoja kuvan koon muokkauksen lisäksi, kuten värien manipulaatiota, kuvien yhdistämistä, kuviin filtterien lisäämistä ja muuta.

Sharp ei toimi verkkoselain-ympäristössä joten kuvanmuokkaus voidaan tehdä vasta palvelimella. Tämän takia Sharp on huomattavasti nopeampi kuin esimerkiksi React-image-file-resizer joka tekee kuvanmuokkauksen verkkoselaimessa. Tästä aiheutuu kuitenkin myös se, että käyttäjän lataamat kuvat joudutaan lähettämään palvelimelle siinä muodossa kuin ne käyttäjällä ovat. Tästä johtuen, jos käyttäjän lataamat kuvat ovat suuria, on mahdollista, että käyttäjä ei pysty lataamaan yhtä monta kuvaa verkkosivulle, kuin jos kuvat muokattaisiin ennen lähettämistä, koska sivustolla on rajoitus tiedostojen yhteiskoossa kuinka monta kuvaa sivustolle voi ladata kerralla.

4.7.2 Sharp-paketin käyttöönotto

Sharp-paketin käyttöönotto oli lähestulkoon yhtä helppoa kuin React-image—file-resizer-paketin käyttöönotto. Sharp-paketti toimii kutsumalla yhtä funktiota johon on syötetty tarvittavat parametrit. (kuva 9) Vaikeuksia käyttöönottoon tuotti se, että testiympäristönä käytetty verkkosivu oli rakennettu siten, että kuvat muunnetaan ja lähetetään base64-muodossa palvelimelle. React-image-file-resizer-paketti pystyi tekemään tämän samalla, kun kuva muokattiin, mutta koska Sharp-paketti ei kuitenkaan toimi verkkoselain ympäristössä niin kuvanmuokkaus voidaan tehdä vasta palvelimen puolella ja kuva täytyy muokata base64-muotoon muulla tavalla.

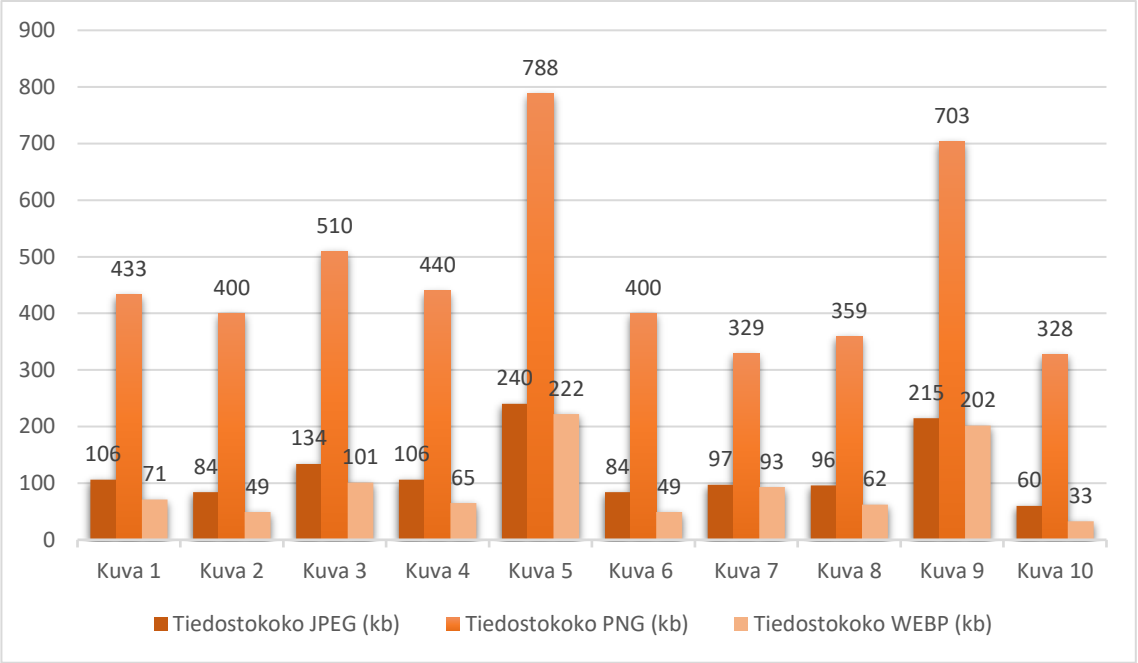
```
var resizePic = function(file)
{
  return new Promise((resolve, reject) => {
    sharp(file) //alkuperäisen tiedoston polku
    .resize ({
      width: 1280, //maksimi leveys pikseleinä
      heigth: 720 , //maksimi korkeus pikseleinä
      withoutEnlargement: true //kuvaa ei suurenneta
    })
    .png({ //tiedostotyyppi
      quality: 65 //pakkaus
    })
    .toFile("image" + imageNr + ".png") //lopputuloksen palautettava tiedosto
    .then(function(outputBuffer) {
      resolve()
    })
  })
}
```

Kuva 9. Sharp-paketin kuvanmuokkaus funktio, testivaiheessa käytettyjen parametrien kanssa PNG-tiedostotyyppinä.

4.7.3 Sharp-paketin kuvankäsittelyn tulos

Koska sharp-paketti ei toimi verkkoselain ympäristössä ja kuvanmuokkaus tehdään palvelimen puolella. Sharp-pakettia käytettäessä säästetään kuvien muokkaukseen käytetyssä ajassa huomattavasti, mutta vaihdossa kulutetaan enemmän verkkokaistaa kuvia ladattaessa käyttäjältä palvelimelle.
Kaikissa testivaihessa Sharp-pakettia käytettäessä yhdenkään kuvan muokkaukseen ei kulunut enempää kuin 1 – 5 millisekuntia. Aika lomakkeen lähetä nappulaa painettaesta vahvistukseen siitä, että yrityksen tiedot oli onnistuneesti tallennettu palvelimelle oli kymmenen kerran mediaanina 7732 millisekuntia. Kuten aikaisemmassakin testivaiheessa, PNG muodossa kuvat olivat suurimpia, JPEG kuvat toisena ja WEBP kuvat pienimpiä tiedostokooltaan. (taulukko 5)

Taulukko 5 sharp-paketilla muokattujen kuvien tiedostokoot JPEG, PNG, GIF ja WEBP muodossa.



4.8 imagemagick-paketin toiminta

4.8.1 imagemagick-paketin laajuus

Kolmantena pakettina opinnäytetyön testivaiheessa oli Imagemagick-paketti. Paketti on noin 21.2kB kooltaan ja ei sisällä riippuvuuksia muihin Node.js-paketteihin. Paketti kuitenkin vaatii Imagemagick-ohjelman komentorivi-työkalut olemaan asennettuina koneella jolla kuvan muokkaus tehdään, joten samoin kuin sharp-pakettia käytettäessä joudutaan kuvan muokkaaminen suorittamaan palvelimella. Tarkoittaen taas, että kun käyttäjä lataa kuviaan sivulle, kuvat tulevat alkuperäisessä koossaan palvelimelle. Tämä voi hidastaa lähetysprosessia, jos alkuperäisien kuvien tiedostokoko on suuri, tai aiheuttaa sen, että käyttäjä ei pysty yhtä montaa kuvaa latamaan palvelimelle, jos ladattavien kuvien yhteinen tiedostokoko on suurempi kuin verkkosivuilla on sallittu.

4.8.2 Imagemagick-paketin käyttöönotto

Imagemagick-paketti vaatii toimiakseen myös Imagemagick-ohjelman komentokehote-työkalut olemaan asennettuina tietokoneella jolla muokkauksen

tekee. Tämän asennus ei kuitenkaan ollut hankalaa tai vaatinut monimutkaisia toimenpiteitä eikä täten tuottanut käyttöönotolle vaikeutta. Kuvanmuokkauksen käyttöönotto itsessään ei paljoa poikennut vaikeuden suhteen aiemmin testatuista paketeista. Toisin kuin React-image-file-resizer-paketin kanssa, Imagemagick pystyy tekemään huomattavasti enemmän erilaisia muokkauksia kuviin. Tämän vuoksi joutuduttiin enemmän tutustumaan imagemagick-paketin dokumentaatioon oikeiden syötettävien funktioiden ja parametrien löytämiseksi. (kuva 10).

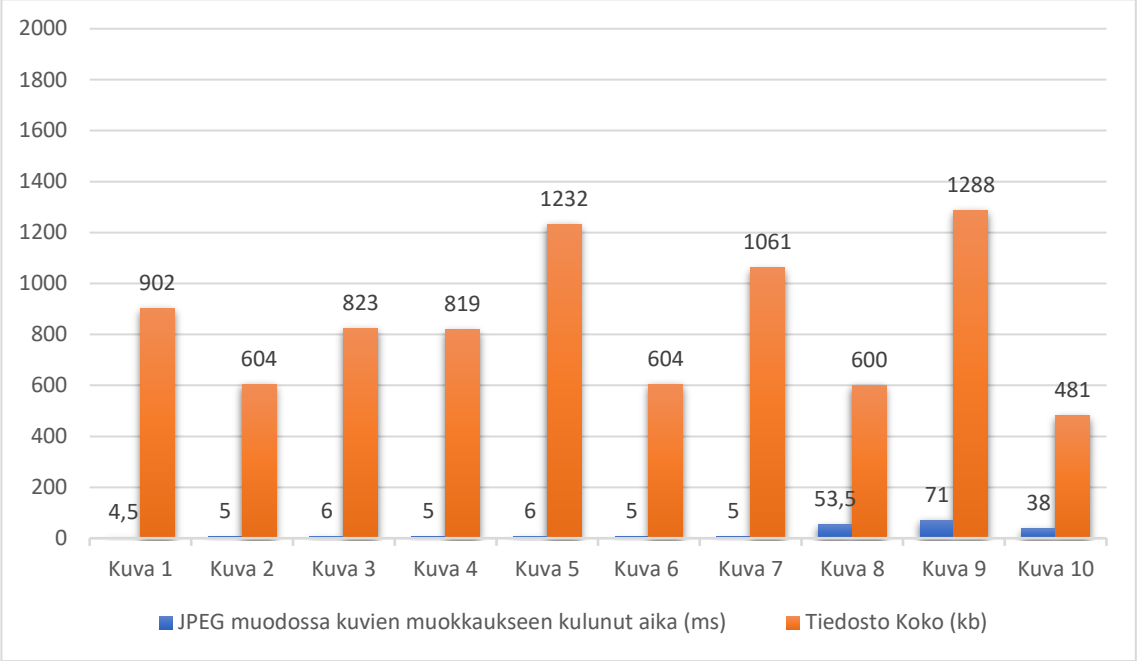
```
var resizePic = function(file)
{
  imageNr++;
    im.resize({
      srcPath: file, //alkuperäinen tiedosto
      dstPath: imageNr + "resized.jpg", //muokatun tiedoston nimi
      width: 1280, //leveys
      height: 720, //korkeus
      quality: 65, //pakkaus
    }), function (err, stdout, stderr){
      if (err) console.log(err);
    }
}
```

Kuva 10 Imagemagick-paketin kuvanmuokkauksen funktio testivaiheessa käytettyjen parametrien kanssa JPG-tiedostotyyppinä.

4.8.3 Imagemagick-paketin kuvankäsittelyn tulos

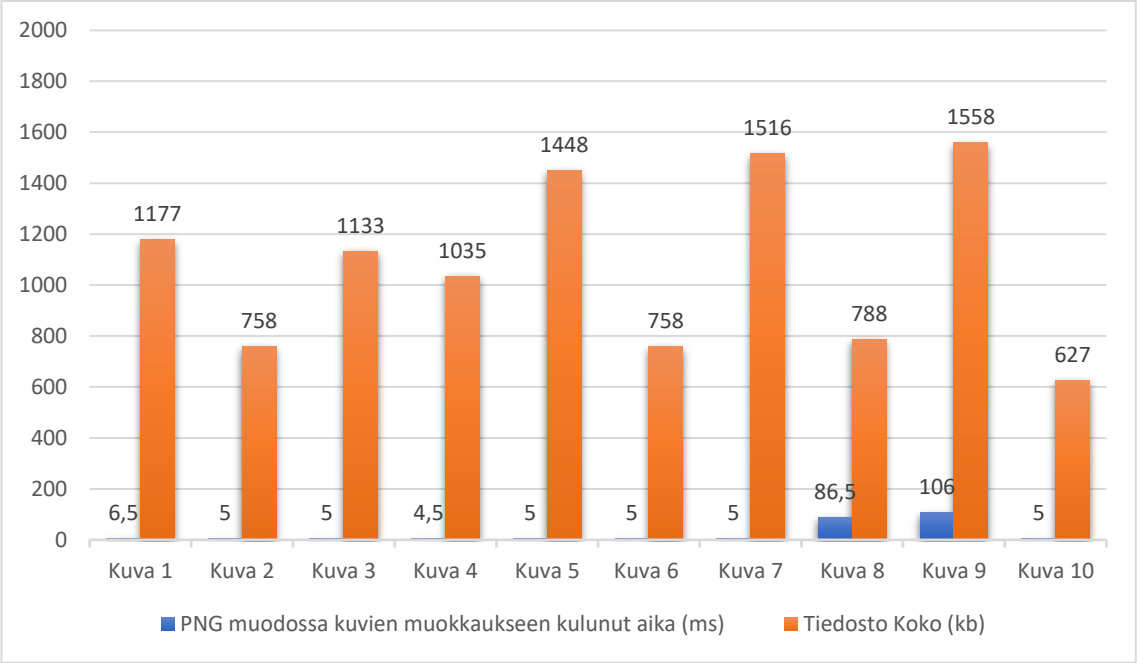
Imagemagick-paketin ensimmäisessä testissä alkuperäiset kuvat muokattiin JPEG-muotoon. Kuvien käsittelyn jälkeen kuvien koko palvelimella oli yhteensä noin 8,20MB. Kuvien muokkaukseen kulunut aika oli yhden kuvan kohdalla kymmenen testikerran jälkeen mediaanina pienimmillään 4,5 millisekuntia ja suurimmillaan 71 millisekuntia. Yhteensä kaikkien kuvien muokkaukseen kulunut mediaani aika oli 199 millisekuntia (taulukko 6). Koko prosessiin siitä, kun käyttäjä painaa lähetä nappia, siihen että palvelin vastaa onnistuneesti tallentaneensa uuden yrityksen tiedot ja kuvat kesti mediaanina 8066 millisekuntia.

Taulukko 6. Imagemagick-paketin JPEG-muodossa kuvien muokkaukseen kuluneiden aikojen mediaani kymmenestä kerrasta.



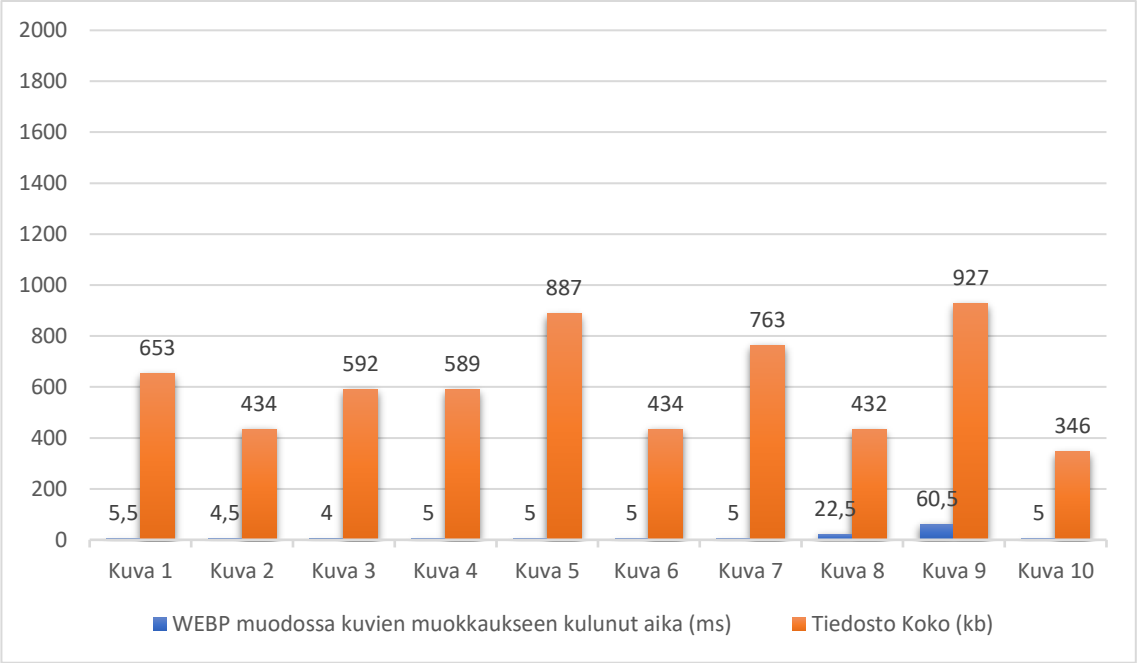
Toisessa testissä alkuperäiset kuvat muokattiin PNG-muotoon. Kuvien käsittelyn jälkeen kuvien koko palvelimella oli yhteensä noin 10,5MB. Kuvien muokkaukseen kulunut aika oli yhden kuvan kohdalla kymmenen testikerran jälkeen mediaanina pienimmillään 4 millisekuntia ja suurimmillaan 106 millisekuntia. Yhteensä kaikkien kuvien muokkaukseen kulunut mediaani aika oli 233 millisekuntia (taulukko 7). Koko prosessiin siitä, kun käyttäjä painaa lähetä nappia, siihen että palvelin vastaa onnistuneesti tallentaneensa uuden yrityksen tiedot ja kuvat kesti mediaanina 8234 millisekuntia.

Taulukko 7. Imagemagick-paketin JPEG-muodossa kuvien muokkaukseen kuluneiden aikojen mediaani kymmenestä kerrasta.



Kolmannessa testissä alkuperäiset kuvat muokattiin WEBP-muotoon. Kuvien käsittelyn jälkeen kuvien koko palvelimella oli yhteensä noin 5,9MB. Kuvien muokkaukseen kulunut aika oli yhden kuvan kohdalla kymmenen testikerran jälkeen mediaanina pienimmillään 4 millisekuntia ja suurimmillaan 60 millisekuntia. Yhteensä kaikkien kuvien muokkaukseen kulunut mediaani aika oli 122 millisekuntia (taulukko 8). Koko prosessiin siitä, kun käyttäjä painaa lähetä nappia, siihen että palvelin vastaa onnistuneesti tallentaneensa uuden yrityksen tiedot ja kuvat kesti mediaanina 7713 millisekuntia.

Taulukko 8. Imagemagick-paketin WEBP-muodossa kuvien muokkaukseen kuluneiden aikojen mediaani kymmenestä kerrasta.

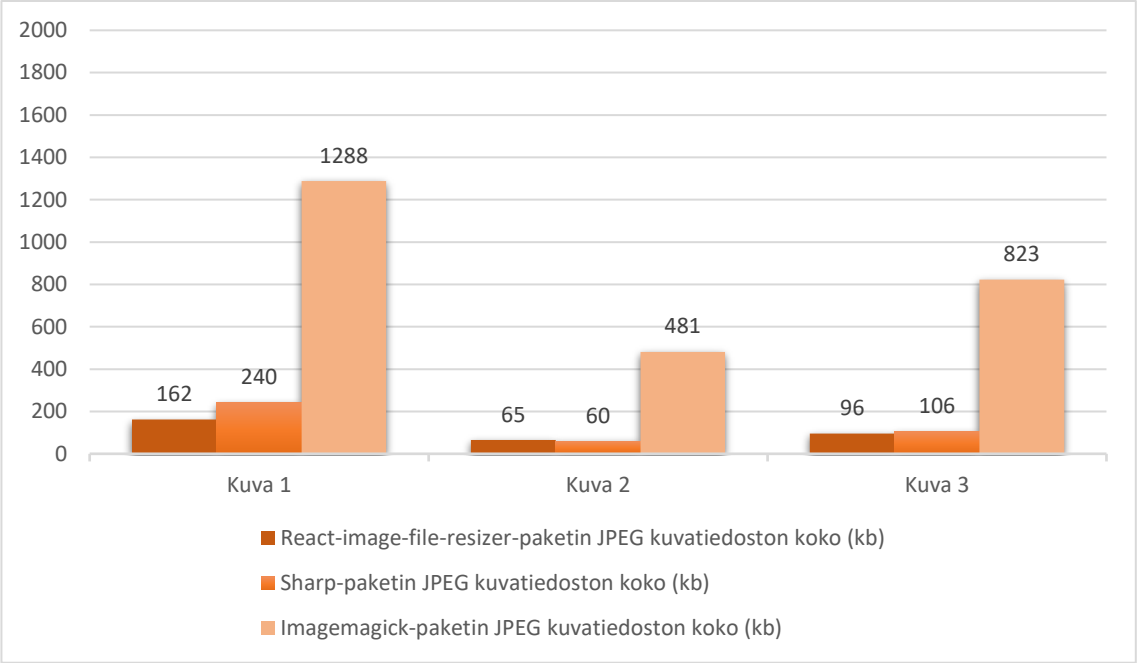


5 Pakettien tulosten vertailut

5.1 Testivaiheiden tuottamien tulosten vertailut

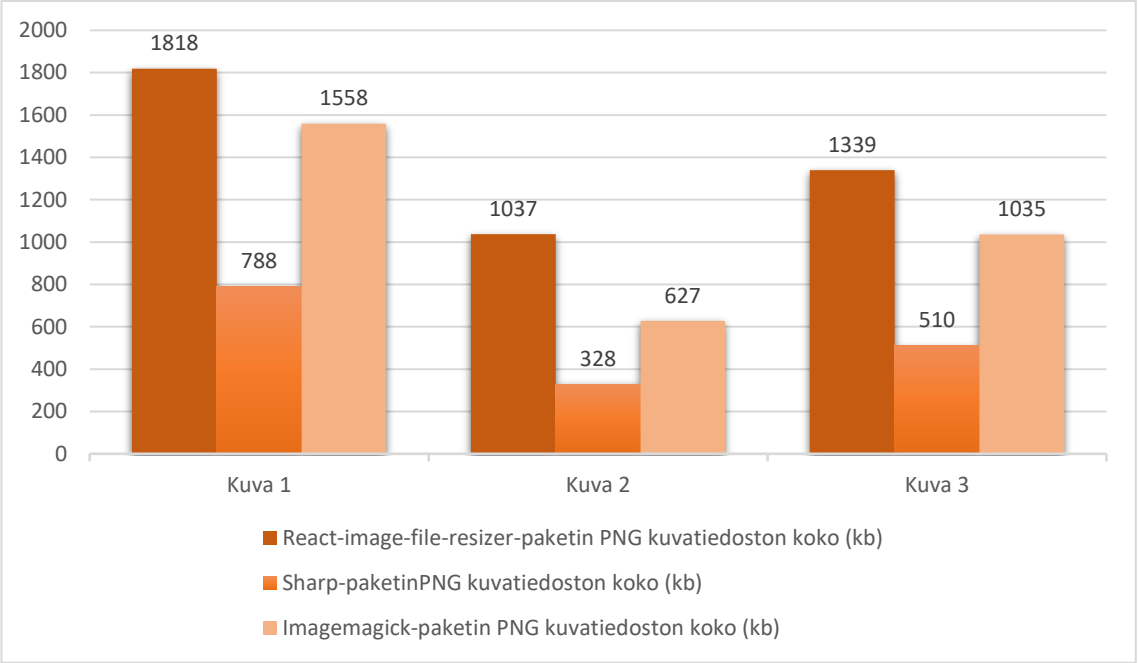
Testattujen Node.js-pakettien välillä JPEG-tiedostoformaatiksi kuvia muunnettaessa React-image-file-resizer- ja Sharp-pakettien tulokset olivat tiedoston koossa lähes samanlaisia toistensa kanssa. Imagemagick-paketti sen sijaan piti tiedostokoon suurena verrattuna kahteen muuhun pakettiin. Kuvissa itsessään ei havaittu näkyvää laadun muutosta alkuperäisten ja muokattujen kuvien välillä tässä testissä. (Taulukko 9.)

Taulukko 9. Suurimman, pienimmän ja satunnaisesti valitun kuvan tiedostokoot kuvanmuokkauksen jälkeen JPEG-muodossa kaikissa kolmessa testivaiheessa.



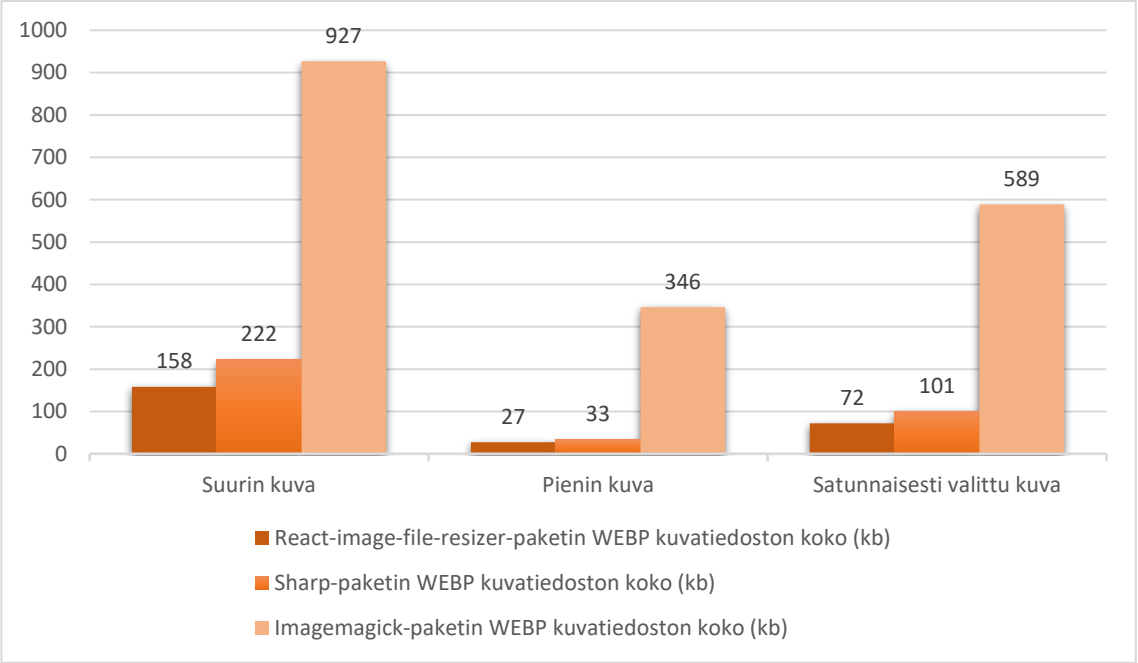
Testattujen Node pakettien välillä PNG-tiedostoformaatiksi kuvia muunnettaessa taas React-image-file-resizer ja Imagemagick-pakettien muokattujen kuvien tiedostokoot olivat suuremmat kuin Sharp-paketin muokattujen kuvien tiedostokoot. Kuvissa itsessään ei havaittu näkyvää laadun muutosta alkuperäisten ja muokattujen kuvien välillä tässäkään testissä. (Taulukko 10)

Taulukko 10. Suurimman, pienimmän ja satunnaisesti valitun kuvan tiedostokoot kuvanmuokkauksen jälkeen PNG-muodossa kaikissa kolmessa testivaiheessa.



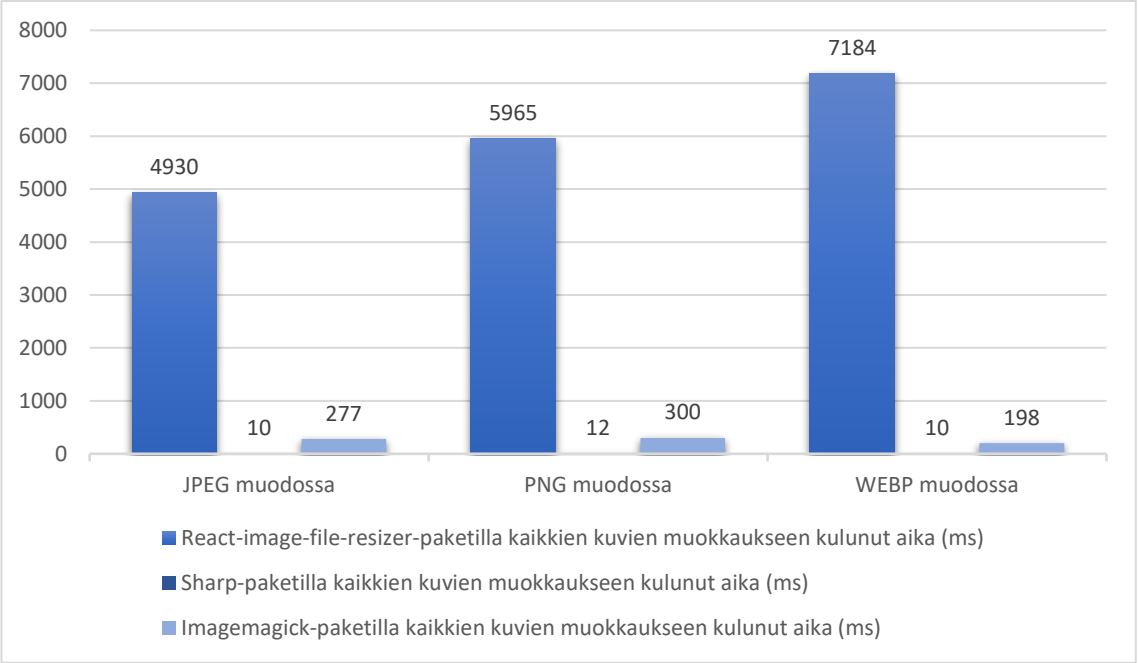
Testattujen Node.js-pakettien välillä WEBP-tiedostoformaatiksi kuvia muunnettaessa taas Imagemagick-paketin muokattujen kuvien tiedostokoot olivat suuremmat kuin Sharp- ja React-image-file-resizer-pakettien muokattujen kuvien tiedostokoot. Kuvissa itsessään ei havaittu näkyvää laadun muutosta alkuperäisten ja muokattujen kuvien välillä tässäkään testissä. (Taulukko 11)

Taulukko 11. Suurimman, pienimmän ja satunnaisesti valitun kuvan tiedostokoot kuvanmuokkauksen jälkeen WEBP-muodossa kaikissa kolmessa testivaiheessa.



Nopeuden kannalta Sharp-paketti ja Imagemagick-paketti olivat React-image-file-resizer-pakettia nopeampia (Taulukko 12). Koli virtual showroom -verkkosivun toimintojen takia on kuitenkin otettava huomioon se, että Sharp- ja Imagemagick-pakettia käytettäessä kuvan muokkaus voidaan suorittaa vasta palvelimella. Tästä johtuen on mahdollista, että käyttäjä joka haluaa ladata kuviaan sivulle saattaa joutua vähentämään kuvia joita haluaa sivustolle ladata, jos tämä ei niitä itse osaa muokata pienemmäksi. Ohjelmistoissa joissa ei ole merkitystä missä vaiheessa kuvan muokkaus suoritetaan, ovat Sharp- ja Imagemagick-paketeilla nopeuden kannalta parempia kuin React-image-file-resizer.

Taulukko 12. Kaikkien kuvien kuvanmuokkaukseen kulunut aika JPG-, PNG- ja WEBP-tiedostoformaateiksi.



5.2 Testivaiheissa käytettyjen pakettien käyttöönoton vertailu

Käyttöönottamisen vaikeudessa kaikki paketit olivat lähestulkoon yhtä vaikeita. Sharp-paketissa ja Imagemagick-paketissa on enemmän toimintoja joten oikean toiminnon löytämiseen saattaa kulua kehittäjältä enemmän aikaa mutta toiminnallisuudeltaan kaikki kolme pakettia toimi tämän opinnäytetyön testivaiheissa yhden funktion avulla. Ajallisesti käyttöönotossa Imagemagick-paketissa kuitenkin kesti pisimpään sillä tämä paketti vaati Imagemagick-komentokehote työkalujen olevan asennettuina koneella jolla kuvat muokataan.

6 Pohdinta

Testivaiheessa testattujen pakettien testitulosten suosittelen, että Koli virtual showroom -verkkosivun jatkokehityksessä otetaan käyttöön nykyisen React-image-file-resizer-paketin tilalle kuvanmuokkauksen toiminnalísuuksiin Sharp-paketti. Vaikkakin Sharp-paketti pystyy tekemään kuvanmuokkauksen vasta sen jälkeen, kun käyttäjä on kuvat lähettänyt palvelimelle, on kuvanmuokkauksen kestossa säästetty aika huomattava sivuston toimivuuden kannalta.(taulukko 12)

Sharp-pakettia käytettäessä kuvien muokkaamisessa käytetyt parametrit voidaan pitää muuten samana kuin Sharp-paketin testivaiheessa (kuva 9), paitsi tiedostotyyppi kohta suositellaan vaihtamaan JPEG-tiedostoformaattiin, koska JPEG-tiedostotyypin kuvat olivat huomattavasti pienempiä kuin PNG-tiedostotyypin kuvat (taulukko 9, taulukko 10). Vaikka WEBP-tiedostotyypin kuvat ovat lähes yhtä pieniä kooltaan kuin JPEG-kuvat ei tätä tiedostotyyppiä voida suositella olemaan käytettynä ainoana tiedostotyyppinä johtuen WEBP-formaatin huonosta tuesta vanhoilla selaimilla tai laitteilla. Kuitenkin aiemmin opinnäytetyössä kuvattua format switching -tekniikkaa hyödyntäen voidaan jatkokehityksessä harkita WEBP-tiedostotyypin kuvien käyttöä siten, että selaimilla jotka tukevat WEBP-tiedostoformaattia lataavat kuvat tässä formaatissa ja niille selaimille tai laitteille jotka eivät tue WEBP-tiedostoformaattia ladataan kuvat JPEG-tiedostoformaatissa. Tässä tapauksessa joudutaan myös ottamaan huomioon se, että palvelimella joudutaan pitämään kaksi versiota kaikista kuvista joka voi viedä paljon kovalevytilaa.

Opinnäytetyössä kuvattua browser caching -tekniikan käyttöönottoa myös suositellaan Koli virtual showroom -verkkosivun staattisten elementtien kanssa. Tämä nopeuttaa verkkosivun latautumista käyttäjille jotka vierailevat sivustolla ensimmäisen kerran jälkeen. CDN caching -tekniikan hyödyllisyyttä voidaan jatkokehityksessä miettiä, jos sivuston kävijämäärä oletetaan kasvavan suureksi, mutta koska CDN caching -tekniikka riippuu kolmannen osapuolen palveluntarjoajien hinnastoista nostaisi tekniikan käyttöönotto tämänhetkisen toteutuksen rahallista budjettia huomattavasti.

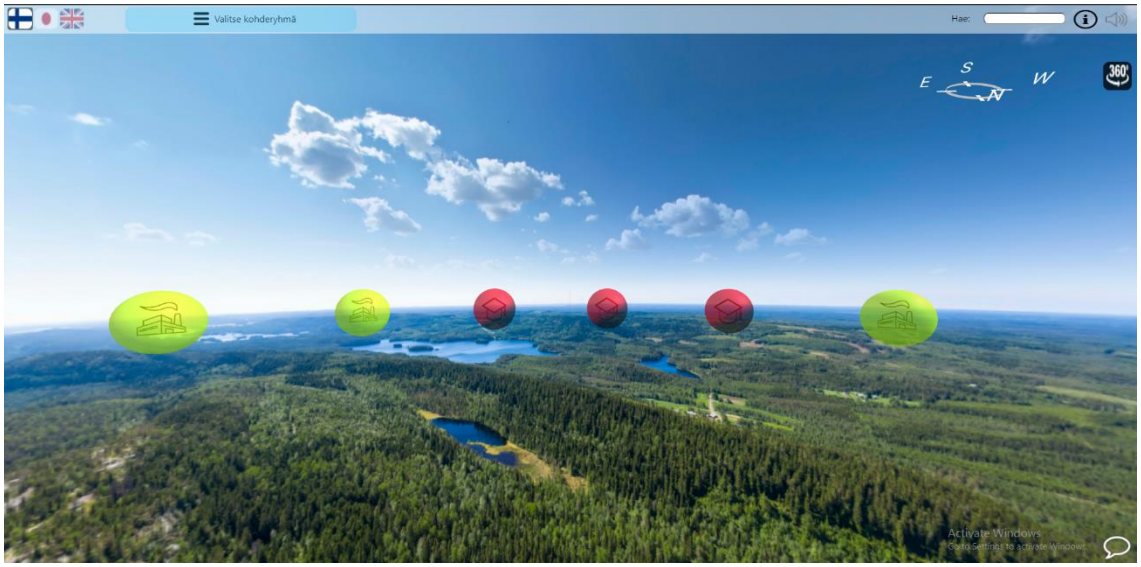
Opinnäytetyössä kuvattua Lazy loading -tekniikkaa voidaan Koli virtual showroom -verkkosivun tämänhetkisestä toteutuksesta parantaa esimerkiksi siten, että yritysten esittelykuvista tallennetaan muokatuista ja pienennetyistä kuvista vielä pikselikoossa pienempi versio jotka ladattaisiin yrityksen esittelyikkunan kuvakaruselliin, kun tämä ikkuna avataan. Suurempi kuva ladattaisiin vasta sitten, kun käyttäjä klikkaa kuvakarusellistä pienempää kuvaa jolloin sivusto näyttää kuvan kokonaan.

Lähteet

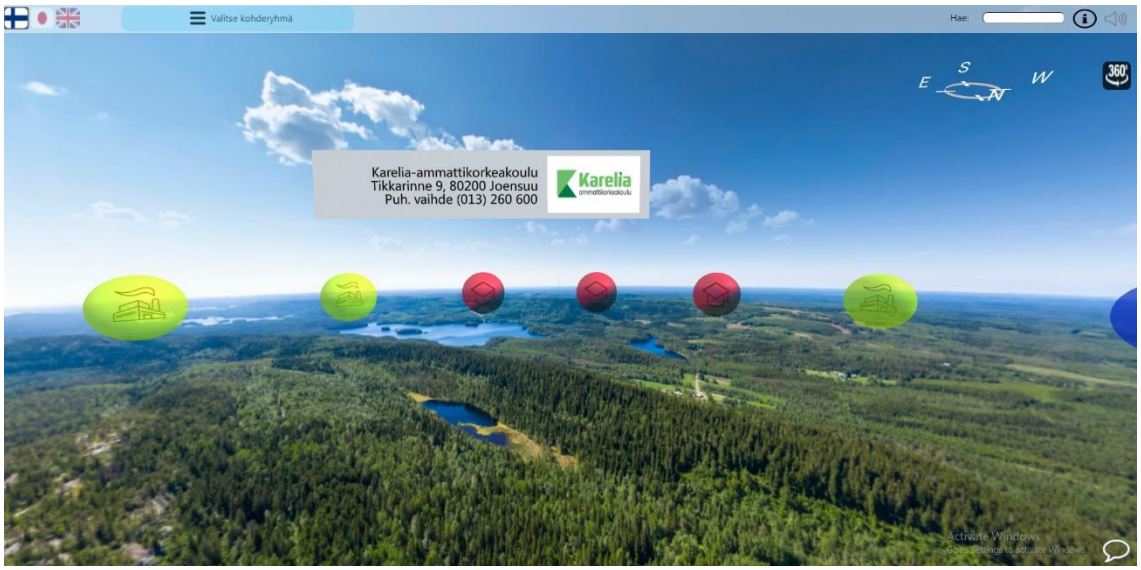
Amazon. Amazon cloudfront cdn https://aws.amazon.com/cloudfront/ 28.04.2022.
Bendell, C. & Sillars, D. 2019. HTTP Archive's annual state of the web report. 4. https://almanac.httparchive.org/en/2019/ 22.1.2022.
Cloudfare. Cloudfare cdn. https://www.cloudflare.com/cdn/ 28.04.2022
Dave Gash. 2018. HTTP Caching https://developers.google.com/web/fundamentals/performance/get-started/httpcaching-6 06.04.2022
Google. 2022. Which web browsers natively support WebP? https://developers.google.com/speed/webp/faq 06.04.2022
Google 2016. Mobile site load time statistics. https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-site-load-time-statistics/ 19.04.2022.
Internet Assigned Numbers Authority. 2022. Media types. https://www.iana.org/assignments/media-types/media-types.xhtml#image 20.1.2022.
MDN Web Docs. 2021. MIME types. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types 20.1.2022
Piros T & Seymour B & Portis E. 2020 HTTP Archive's annual state of the web report 5. https://almanac.httparchive.org/en/2020/media#picture-format-switching 06.04.2022
Portis E & Sillars D. 2021 HTTP Archive's annual. 5 state of the web report. https://almanac.httparchive.org/en/2021/media 19.04.2022.
What is caching? 2022. Cloudfare https://www.cloudflare.com/learning/cdn/what-is-caching 06.04.2022
Pixelplumbing. 2022. https://sharp.pixelplumbing.com 19.04.2022
Rasmus Andersson 2013. NPM https://www.npmjs.com/package/imagemagick 19.04.2022
Onur Zorluer & al. 2021. NPM https://www.npmjs.com/package/react-image-file-resizer 19.04.2022
Nodejs. 2022. https://nodejs.dev/learn/the-nodejs-path-module 19.04.2022
Formium. 2020. https://formik.org 19.04.2022

Kuvia Koli virtual showroom -verkkosivuston käyttäjän näkymistä

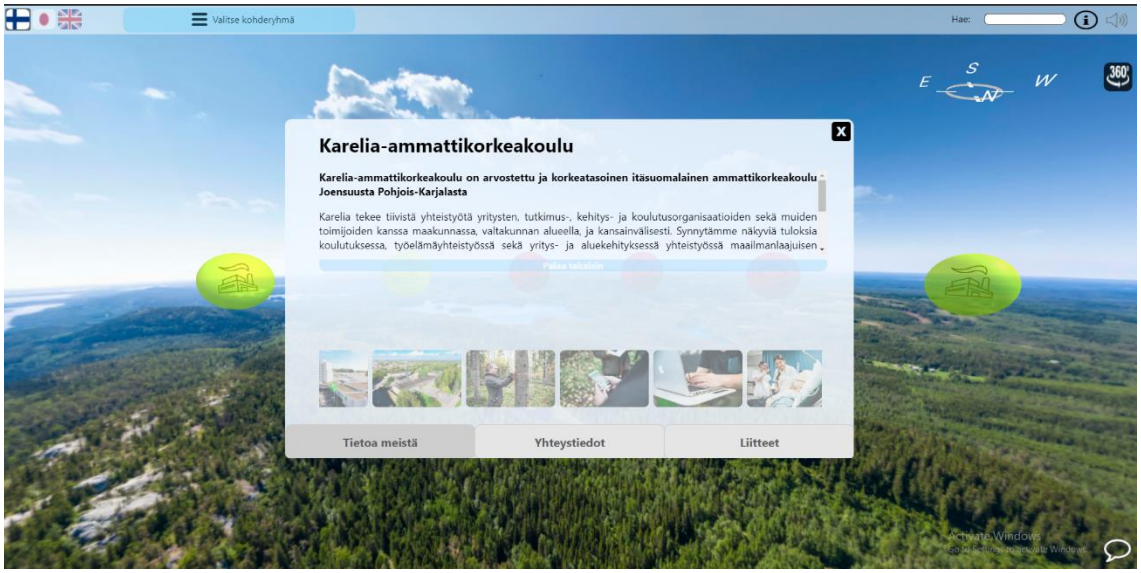
Verkkosivun lomakkeessa kohta johon logot sekä kuvat syötetään



Verkkosivun päänäkymä



Verkkosivun lyhyet yritystiedot näkymä



Verkkosivun tarkemmat yritystiedot näkymä