

Simo Nurmi

## **VERKKOSIVUN SELAINPUOLEN SUORITUSKYKY**

## **VERKKOSIVUN SELAINPUOLEN SUORITUSKYKY**

Simo Nurmi  
Opinnäytetyö  
Syksy 2017  
Tietojenkäsittely  
Oulun ammattikorkeakoulu

## TIIVISTELMÄ

Oulun ammattikorkeakoulu  
Tietojenkäsittelyn tutkinto-ohjelma, Web-sovelluskehitys

---

Tekijä: Simo Nurmi

Opinnäytetyön nimi: Verkkosivun selainpuolen suorituskyky

Työn ohjaaja: Ritva Virkkala

Työn valmistumislukukausi ja -vuosi: Syksy 2017

Sivumäärä: 25

---

Opinnäytetyössä pohditaan web-sivun selainpuolen suorituskyvyn tärkeyttä ja esitellään keinoja parantaa sitä. Opinnäytetyössä ei ollut toimeksiantajaa.

Verkkosivuston selainpuolen suorituskyky on erittäin tärkeää yritysten liiketoiminnan kannalta. Pienikin hidastuminen sivun latausnopeudessa voi laskea konversiota ja myyntiä. Sivuston täytyy olla nopea myös mobiililaitteilla, sillä valtaosa Internet-käyttäjistä käyttää älypuhelin tai taulutietokonetta.

Selainpuolen suorituskykyä voi optimoida monilla tavoin, mutta suurin osa niistä liittyy jollain tavoin tiedostojen lataukseen tai HTML-dokumentin jäsentämisen ja suorittimen pääsäikeen vapauttamiseen. Dokumentin nopea jäsentäminen on tärkein asia latausajan parantamisessa. Kuormittamaton suoritin takaa sivun interaktiivisuuden.

Yksittäisiä optimointikohteita ovat esimerkiksi CSS-tyylit, JavaScript-ohjelmat, kuvat ja web-fontit. Kuhunkin liittyy omat haasteensa.

CSS-tyylitiedostot pysäyttävät HTML-dokumentin jäsentämisen, mutta sen voi ohittaa jakamalla tyylit kahteen osaan, ja lataamalla vähemmän tärkeitä tyylit myöhemmin.

JavaScript-ohjelmissa on tärkeää olla lataamatta ylimääräistä koodia. Modernit tekniikat helpottavat ohjelmien suorituskyvyn optimointia.

Kuvien optimoinnissa tärkeintä on kuvatiedoston koon pienentäminen sekä oikean kuvatiedostoformaatin valinta. Kuvien lataamisessa tulisi käyttää lazyload-tekniikkaa.

Web-fontit ovat tiedostokooltaan suuria, eivätkä ne ole verkkosivun sisällön kannalta tärkeitä. Fontit voidaan ladata viivytetysti käyttämällä font-display-ominaisuutta, selaimen font loading-ohjelmointirajapintaa tai FontFaceObserver-lisäkettä.

---

Asiasanat: www-sivut, CSS, JavaScript

## ABSTRACT

Oulu University of Applied Sciences  
Degree Programme in Business Information Systems, Web Application Development

---

Author: Simo Nurmi

Title of thesis: Website client-side performance

Supervisor: Ritva Virkkala

Term and year when the thesis was submitted: Fall 2017      Number of pages: 25

---

The thesis explores the importance of performance in the client-side of a website, and presents ways to improve it. The thesis was not commissioned by a client.

Client-side performance of a website is vital to success of a company. Even a small increase in page load times can decrease conversion and sales. Websites also must load fast on mobile devices, since most of Internet users browse using smartphones or tablet computers.

Client-side performance can be optimized in many ways, but most of them are in some way related to resource downloading, HTML document parse or freeing the main thread of the processor. Parsing the HTML document is the most important factor when it comes to improving page load times. An unburdened processor guarantees that the page is interactive.

CSS styles, JavaScript programs, images and web fonts are among the things to optimize. Each of them has its own set of challenges.

CSS stylesheets block HTML document parsing, but it can be circumvented by splitting styles in to two parts, and loading less important styles later.

In JavaScript programs, it's important not to send any unnecessary code to the user. Modern techniques make optimizing JavaScript easier.

When optimizing images, it's important to decrease the image file size and choose the correct file format. Images should be loaded using the lazyload technique.

Web fonts have large file sizes, and they are not needed for the user to start consuming content. Fonts loading can be deferred using the font-display property, font loading application programming interface or the FontFaceObserver plugin.

---

Keywords: websites, CSS, JavaScript

# SISÄLLYS

1	JOHDANTO .....	6
2	SUORITUSKYKY.....	7
2.1	Vaikutukset.....	7
2.2	Mittaaminen.....	8
2.2.1	Google Chromen sovelluskehittäjätyökalut .....	8
2.2.2	Web Page Test .....	9
2.2.3	Google PageSpeed Insights .....	9
3	HTML-DOKUMENTTI .....	11
3.1	Dokumenttioliomalli .....	11
3.2	Resurssipyynnöt.....	11
4	CSS-TYYLITIEDOSTO.....	12
4.1	Renderöinti.....	12
4.2	Jäsennyksen keskeytyminen.....	12
5	JAVASCRIPT .....	14
5.1	Lataus ja suoritus .....	14
5.2	Moduulit.....	14
5.3	Niputustyökalut.....	15
6	KUVAT.....	17
6.1	Tiedostomuodot.....	17
6.2	Lazyload-tekniikka.....	18
7	WEB-FONTIT .....	20
7.1	Viivytetty lataus .....	20
7.2	Varioidut fontit .....	22
8	LÄHTEET .....	24

# 1 JOHDANTO

Olen erityisen kiinnostunut verkkosivujen selainpuolen suorituskyvystä, koska kukaan muu ei ole. Useat sivustot lähettävät valtavia määriä turhaa dataa käyttäjilleen, ja niiden latausajat ja suorituskyky ovat varsinkin mobiililaitteilla surkeita. Ongelmaan ei ole helppoa ratkaisua, sillä siihen ovat syyllisiä niin työntajat, asiakkaat kuin web-kehittäjätkin. Ainoa ratkaisu on puhua siitä.

Opinnäytetyössä kerrotaan mitä verkkosivun selainpuolen suorituskyky on, tarkastellaan sen tärkeyttä yritysten ja käyttäjän näkökulmasta, ja esitellään keinoja sen parantamiseen. Kehityskohteista esitellään yksityiskohtaisesti HTML-dokumentti, CSS-tyylitiedostot, JavaScript-ohjelmat, kuvat sekä web-fontit.

Verkkosivujen tekniikka kehittyi hyvin nopeasti. Voidaan sanoa, että tekniikat, jotka olivat vuosien sitten uusia, ovat nyt jo vanhoja. Opinnäytetyössä tarkastellaan, miten uusimpia tekniikoita voi hyödyntää suorituskyvyn optimoinnissa.

Opinnäytetyötä kirjoittaessani testasin jokaisen suorituskykyä parantavan tekniikan itse. Osa koodistani on liitetty opinnäytetyöhön koodiesimerkkeinä.

Opinnäytetyössä ei kerrota palvelinpuolen suorituskyvyn optimoinnista, vaikka se vaikuttaakin suoraan selainpuolen suorituskykyyn.

## 2 SUORITUSKYKY

Verkkosivun suorituskyvyllä voidaan tarkoittaa kahta asiaa. Useimmiten sillä viitataan verkkosivun latausnopeuteen; nopeasti latautuvalla verkkosivulla on hyvä suorituskyky. Sillä saatetaan myös tarkoittaa verkkosivun käyttöliittymän vastauskykyä.

Suorituskykyyn liittyy myös käyttäjän kokemus suorituskyvystä. Esimerkiksi jos sivu latautuu hyvää vauhtia, mutta käyttäjälle ei näytetä mitään välivaiheita sivun piirtymisestä, käyttäjä voi luulla, että sivu latautuu hitaasti tai ei ollenkaan.

### 2.1 Vaikutukset

Verkkosivuston latausajoilla on suuri merkitys yrityksen liiketoiminnan kannalta. Yhdysvaltalainen verkkokauppa Amazon havaitsi, että 100 millisekunnin viive sivun latautumisessa laski myyntiä yhden prosenttiyksikön verran (Linden 2006, viitattu 17.12.2017).

Googlen tekemässä tutkimuksessa selvisi, että verkkosivun hidastaminen 100–400 millisekunnilla vähensi hakukoneella tehtävien hakujen määrää 0,2–0,6 %. Sen lisäksi pidemmästä latausajasta pidempään kärsineet käyttäjät alkoivat vähentää hakukoneen käyttöä entisestään. (Jake Brutlag 2009, viitattu 17.12.2017.) Suorituskyky vaikuttaakin suoraan asiakasuskollisuuteen. Dynatracen tutkimukseen osallistuneista taulutietokonekäyttäjistä 33 % ei ostaisi verkkokaupasta, jolla on huono suorituskyky, 46 % siirtyisi kilpailijan verkkokauppaan, ja 35 % ei käyttäisi verkkokauppaa millään muullakaan laitteella (Jakober 2012, viitattu 17.12.2017).

Google käyttää suorituskykyä yhtenä mittarina hakutulosten järjestyksessä. Suorituskykyinen sivu näkyy ylempänä hakutulostuloksissa. Googlen PageSpeed Insights palvelussa voi tarkastaa sivuston saamat suorituskykypisteet. Kirjoitushetkellä OAMK:n verkkosivuston etusivun mobiiliversio sai 33 pistettä ja työpöytäversio 24 pistettä sadasta.

Suomi on yksi harvoista maailman maista, jossa käytetään kiinteähintaisia mobiililaajakaistaliittymiä. Esimerkiksi Pohjois-Amerikassa ja Aasiassa datan käytön mukaan hinnoitellut liittymät ovat

yleisiä. OAMK:n verkkosivuston etusivun lataaminen Kanadassa käyttöpohjaisesti hinnoitellulla liittymällä maksaisi 0,52 USA:n dollaria. Verkkosivun hinnan eri maissa voi tarkastaa What Does My Site Cost -palvelussa.

## **2.2 Mittaaminen**

Verkkosivuston suorituskykyä on suhteellisen helppo mitata. Verkkoselaimiin on sisäänrakennettu siihen tarkoitettuja työkaluja kuten esimerkiksi kehittäjän työkalut Google Chrome -selaimessa. Lisäksi on olemassa ulkoisia palveluita kuten Web Page Test ja Google PageSpeed Insights.

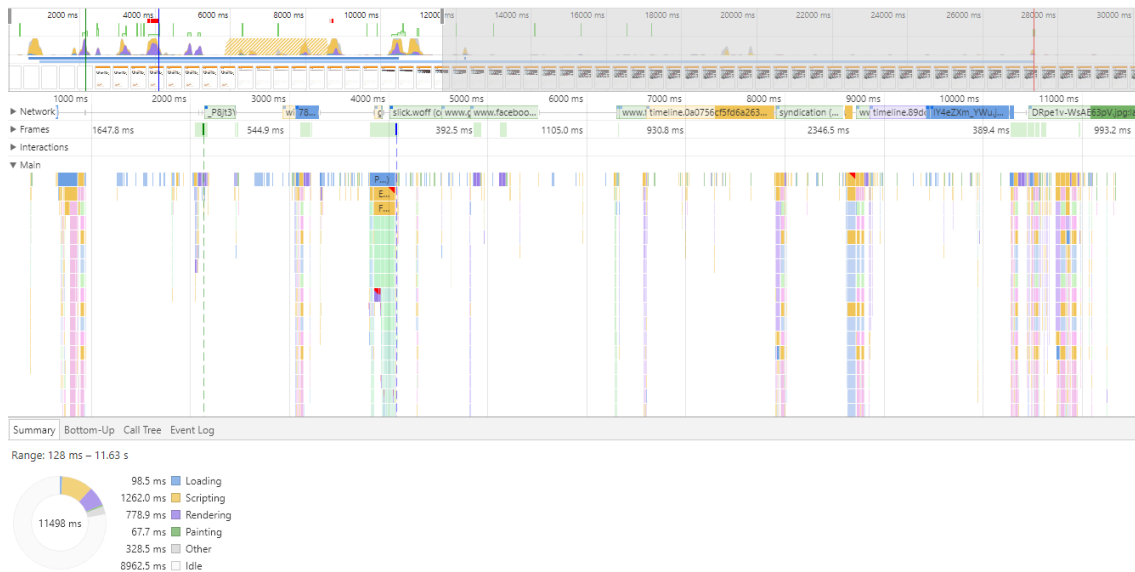
### **2.2.1 Google Chromen sovelluskehittäjätyökalut**

Google Chrome -verkkoselaimessa on monipuoliset, sisäänrakennetut sovelluskehittäjätyökalut. Suorituskyvyn kannalta tärkeimmät niistä ovat verkkoyhteys-, suorituskyky- ja auditointityökalut.

Verkkoyhteistyökaluissa (network) voidaan tarkastaa mm. eri resurssipyyntöjen koot, latausajat ja tunnistetiedot. Resurssien lataus esitetään vesiputousnäkyssä, josta voi helposti nähdä, onko resurssipyyntöjen välillä tarpeettomia riippuvaisuuksia.

Suorituskykytyökalua (performance) käytetään suorituskykyprofiilin luomiseen. Profiilista nähdään millisekunnin kymmenesosan tarkkuudella, mitä selain on tehnyt mitatun ajanjakson aikana. Profiilissa näkyy esimerkiksi dokumentin jäsentäminen, layout- ja paint-tapahtumat, JavaScript-ohjelman suoritus, sekä latausta hidastavat pullonkaulat (kuvio 1.).





KUVIO 1. Suorituskykyökalun luoma profiili

Auditointityökalut (audits) mahdollistavat erilaisten automaattisten tarkistusten suorittamisen. Auditaitavia ominaisuuksia ovat suorituskyky, esteettömyys, parhaat käytännöt sekä yhteensopivuus progressiivisen web-sovelluksen standardeihin.

## 2.2.2 Web Page Test

Web Page Test -palvelussa (<http://www.webpagetest.org>) voi testata, miten verkkosivu latautuu eri puolilla maailmaa. Palvelussa voi valita selaimen sekä haluamansa testipalvelimen sijainnin. Tuloksissa näkyy mm. yksityiskohtainen vesiputousnäkyvä resurssien lataamisesta sekä vinkkejä parannusta vaativista asioista.

## 2.2.3 Google PageSpeed Insights

PageSpeed Insights -palvelu pisteyttää verkkosivut niiden suorituskyvyn mukaan. Google painottaa hakutuloksia näiden pisteiden perusteella.

Verkkosivut arvioidaan erikseen niiden työpöytä- ja mobiilisuorituskyvyn mukaan. Arvioinnissa otetaan huomioon enimmäkseen selainpuolen suorituskyvyn liittyviä asioita, kuten tyylitiedostojen ja JavaScript-ohjelmien latausmetodit, kuvien optimointi sekä näkyvän sisällön priorisointi, mutta myös palvelinpuolen asioita kuten palvelimen vastausaika ja välimuistin hyödyntäminen (kuvio 2.).

## PageSpeed Insights

http://www.oamk.fi/ **ANALYSOI**

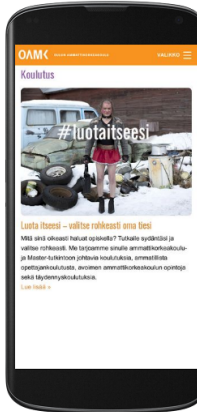
**Mobiili** **Tietokone**

**Poor**  
33 / 100

Sivua ei ole optimoitu, ja sen käyttökokemus on todennäköisesti hidas. Priorisoi seuraavat suositukset ja toimi niiden mukaisesti.

**Mahdolliset optimoinnit**

- Optimoi kuvat
  - ▶ Näytä korjausohjeet
- Poista hahmonnuksen estävä JavaScript ja CSS sivun yläosan sisällöstä
  - ▶ Näytä korjausohjeet
- Hyödynnä selaimen välimuistia
  - ▶ Näytä korjausohjeet
- Pienennä CSS
  - ▶ Näytä korjausohjeet
- Pienennä JavaScript
  - ▶ Näytä korjausohjeet
- Pienennä HTML
  - ▶ Näytä korjausohjeet



KUVIO 2. PageSpeed Insights

### 3 HTML-DOKUMENTTI

HTML-dokumentti on verkkosivun tärkein osa. Dokumentti koostuu HTML-tunnisteista, joilla kullakin on oma tehtävänsä. Esimerkiksi h1-tunniste kuvaa dokumentin pääotsikkoa, p-tunniste kappaletta ja a-tunniste hyperlinkkiä.

HTML eli HyperText Markup Language kehitettiin alun perin CERNissä tieteellisten dokumenttien tallentamiseen ja toisiinsa linkittämiseen.

#### 3.1 Dokumenttioliomalli

Verkkoselain on yleisin esimerkki HTML-dokumentteja parsivasta ohjelmasta. Selain lataa dokumenttiedoston virtauttamalla, eli se voi alkaa muodostamaan dokumenttia ennen kuin koko tiedosto on latautunut loppuun asti.

Selain jäsentää dokumentissa tekstimuotoisena olevat HTML-tunnisteet HTML-elementeiksi, ja muodostaa niistä dokumenttioliomallin (Document Object Model eli DOM). Dokumenttioliomalli on selaimen muistissa oleva esitys varsinaisesta HTML-dokumentista. (World Wide Web Consortium 2017, viitattu 12.12.2017.)

#### 3.2 Resurssipyynnöt

Jotkin HTML-tunnisteet kuten link-, script-, img-tunnisteet käynnistävät resurssipyynnöjä. Resurssipyynnöinä haettavia tiedostoja voivat olla esimerkiksi tyylitiedostot, JavaScript-ohjelmat, data tai mediatiedostot kuten kuvat ja videot. Resurssipyynnöt saattavat keskeyttää dokumentin parsimisen väliaikaisesti tai varata selaimen pääsäikeen. Nämä pidentävät verkkosivun kokonaislatausaikaa sekä muita latausajan mittareita. Resurssipyynnöjen optimointi onkin yksi tärkeimmistä optimoinnin kohteista.

## 4 CSS-TYYLITIEDOSTO

CSS eli Cascading Style Sheets on merkintätapa, jolla verkkosivun ulkoasu muotoillaan. CSS-määrittelyt koostuvat valitsimista, joille annetaan sääntöjä. Tyylimäärittelyksiä voi tehdä erillisessä CSS-tyylitiedostossa, style-tietueessa tai suoraan tietueisiin style-attribuutin avulla.

### 4.1 Renderöinti

Dokumenttioliomallissa olevat elementit sijoitellaan paikalleen niitä koskevien CSS-sääntöjen mukaan. Sijoittelu tapahtuu layout-tapahtumassa, jossa selain muuttaa suhteelliset arvot absoluuttisiksi pikseliarvoiksi. Lopuksi elementit maalataan sivulle paint-tapahtumassa.

Kaikki elementin sijoitteluun ja laatikkomalliin tehtävät muutokset käynnistävät layout-tapahtuman. Mikäli elementtiin tehtävät muutokset vaikuttavat muihin elementteihin, täytyy myös niiden sijainti laskea uudelleen. Tämän vuoksi voidaan ajatella, että ensimmäisen elementtien sijoittelun ja renderöinnin jälkeen tehtävät layout-tapahtuman käynnistävät muutokset ovat kalliita prosessorille ja haittaavat suorituskykyä ja interaktiivisuutta.

Animaatioissa ei tulisi käyttää ollenkaan layout-tapahtuman käynnistäviä CSS-ominaisuuksia. Pelkän paint-tapahtuman vaativia ominaisuuksia kuten taustaväriä voi käyttää vapaammin, mutta nekin saattavat olla raskaita prosessorille. Ainoat laitteistokiihdytetyt CSS-ominaisuudet ovat opacity ja transform. Ne pystyvät hyödyntämään prosessorin lisäksi myös näytönohjainta, ja sopivat siksi hyvin animointiin.

### 4.2 Jäsennyksen keskeytyminen

Tyylitiedoston lataaminen ja jäsentäminen keskeyttävät HTML-dokumentin jäsentämisen, kunnes molemmat on suoritettu. Tyylitiedoston lataaminen hidastaa koko dokumentin latausaikaa ja varsinkin ensimmäistä maalausta ja sisällön näyttämistä. Sivun näyttö täysin tyhjältä, kunnes kaikki tyylitiedostot on ladattu. Käyttäjä ei voi tietää, latautuuko sivu.

Jäsentämisen keskeytys voidaan ohittaa sisällyttämällä sisällön lukemisen kannalta tärkeimmät tyylit eli ns. kriittiset tyylit style-tunnisteeseen. Tämä tietenkin kasvattaa HTML-dokumentin kokoa, mutta koska dokumentti ladataan virtauttamalla, sen vaikutus on vähäinen.

Vähemmän tärkeät tyylit voidaan ladata JavaScript-ohjelman avulla, kun dokumentin parsiminen on valmistunut. Tämä voidaan toteuttaa esimerkiksi käyttämällä tyylitiedostoa kutsuvan link-tunnisteen rel-attribuutissa arvoa preload, ja muuttamalla se dokumentin jäsentämisen valmistuttua stylesheetiksi (kuvio 3.).

```
<link rel="preload" href="styles.css" as="style" data-stylesheet>

<script>
  window.addEventListener('DOMContentLoaded', () => {
    const stylesheets = document.querySelectorAll('[data-stylesheet]');
    Array.prototype.map.call(stylesheets, (stylesheet) => {
      stylesheet.rel = 'stylesheet';
    });
  });
</script>
```

KUVIO 3. Tyylitiedoston viivytetty lataus

## 5 JAVASCRIPT

JavaScript-ohjelmat ovat pieniä ohjelmia, joilla lisätään toiminnallisuutta verkkosivuille. Ne voivat esimerkiksi manipuloida dokumenttiolion mallia tai kutsua selaimen ohjelmointirajapintoja.

JavaScript-ohjelmat ovat muita resursseja huomattavasti raskaampia. Ohjelma täytyy ladata, kääntää ja suorittaa. Verkkosivustoja täytyy pystyä käyttämään myös halvemmilla älypuhelimilla, joissa on usein heikko suoritin ja vähän muistia. Tämän vuoksi JavaScript-ohjelmien optimointiin täytyy kiinnittää erityistä huomiota. (Osmani 2017, viitattu 28.12.2017.)

JavaScript-ohjelmia ei voi suorittaa monisäikeisesti, mutta niissä voidaan harjoittaa asynkronista moniajoa (asynchronous multitasking) siirtämällä osa suoritusvastuusta selaimelle sen ohjelmointirajapintojen avulla.

### 5.1 Lataus ja suoritus

JavaScript-ohjelman lataaminen vakioasetuksilla keskeyttää HTML-dokumentin jäsentämisen sen latauksen ja suorituksen ajaksi. Tämän vuoksi web-kehittäjillä oli tapana sijoittaa JavaScript-ohjelmien resurssikutsut tekevät script-tunnisteet dokumentin loppuosaan, jolloin ne häiritsivät jäsentämistä mahdollisimman vähän. Tiedoston lataaminen alkoi kuitenkin vasta, kun HTML-dokumentti oli jäsennetty lähes loppuun saakka.

Nykyään parempi käytäntö on sijoittaa ohjelman script-tunniste dokumentin head-osaan, ja antaa sille defer-attribuutti. Head-osaan sijoitetun JavaScript-ohjelman lataus alkaa mahdollisimman nopeasti. Defer-attribuutin kanssa lataus ei keskeytä dokumentin jäsentämistä, eikä ohjelmaa suoriteta, ennen kuin dokumentin jäsentäminen on valmistunut.

### 5.2 Moduulit

JavaScript-moduulit ovat itsenäisiä, uudelleenkäytettäviä JavaScript-ohjelmia. Moduulit voivat kutsua toisiaan ohjelmoijan määrittämällä tavalla. Moduulikutsuja hyödyntämällä voidaan koostaa koko verkkosivun toiminnallisuus. (Kasireddy 2016, viitattu 28.12.2017.)

Useimmat selainvalmistajat lisäsivät selaimiinsa tuen JavaScript-moduuleille vuoden 2017 aikana. Moduulit ladataan yksittäisinä resursseina, eikä ohjelmaa voida suorittaa ennen kuin kaikki vaaditut moduulit on ladattu. Sellaisenaan moduulit eivät paranna suorituskykyä muuten kuin helpottamalla ohjelmien koostamista.

Moduuleja ei ladata selaimissa, jotka eivät tue niitä, sillä ne eivät ymmärrä script-tietueelle annettavaa tyyppiä module. Jotta tavallisia JavaScript-ohjelmia ei ladattaisi moduuleja käyttävissä selaimissa, niille on annettava nomodule-attribuutti (kuvio 4.).

```
<script type="module" src="module.js"></script>  
<script nomodule defer src="nomodule.js"></script>
```

KUVIO 4. Moduulin resurssikutsu HTML-dokumentissa

Vuoden 2017 loppupuolella Google Chrome -selaimen lisättiin tuki dynaamisille moduulikutsuille. Tämä ominaisuus parantaa mahdollisuuksia suorituskyvyn optimointiin moduulien avulla huomattavasti. Dynaamisessa moduulikutsussa moduulit ladataan asynkronisesti. Tämä tarkoittaa, että pääohjelman suoritus voi alkaa heti, kun sille kriittiset moduulit on ladattu, ja muut moduulit voidaan ladata ja suorittaa myöhemmin (kuvio 5.).

```
import criticalModule from './criticalModule.js';  
  
criticalModule();  
  
import('./anotherModule.js')  
  .then((anotherModule) => {  
    anotherModule.default();  
  });
```

KUVIO 5. Dynaaminen moduulikutsu

### 5.3 Niputustyökalut

Niputustyökaluilla tarkoitetaan työkaluja, joilla JavaScript-moduuleja voidaan koota tai jakaa tiedostoihin. Niputustyökalut helpottivat JavaScript-moduulien käyttöä ennen kuin selaimet tukivat niitä,

mutta ne ovat siitä huolimatta hyödyllisiä tai jopa tarpeellisia. Kirjoitushetkellä suosituin niputustyökalu on Webpack. Se on avoimen lähdekoodin projekti, jota mm. Trivago, Adobe ja Google tukevat tuhansien dollarien lahjoituksilla.

Ennen natiivia selaintukea niputustyökalut olivat yksi tapa luoda JavaScript-moduuleja. Esimerkiksi Webpack-niputustyökalulla voidaan toteuttaa modulikutsut samalla tavalla kuin natiiveilla JavaScript-moduuleilla. Myös dynaamiset modulikutsut ovat mahdollisia lisäosia käyttämällä.

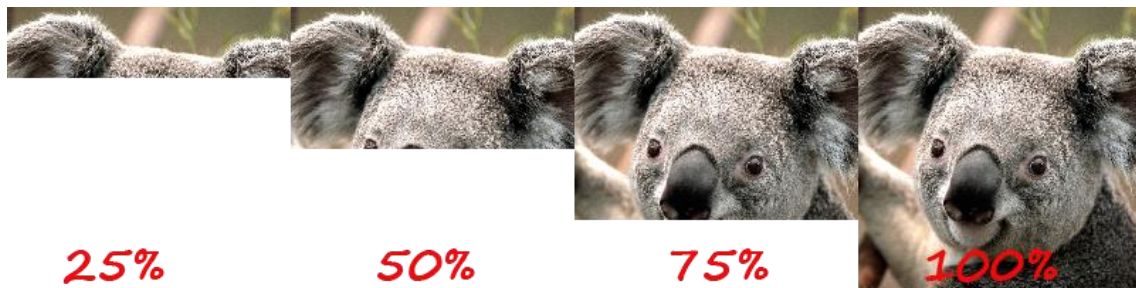


## 6 KUVAT

Suurin osa verkkosivun painosta on tyypillisesti kuvia. Kuva piirretään selaimeen paint-tapahtumassa, mutta kuvien renderöinti on huomattavasti raskaampaa kuin CSS-tyyliin.

### 6.1 Tiedostomuodot

Valokuvat ja vastaavat kuvatiedostot ovat yleensä JPEG-tiedostoja. JPEG on häviöllinen pakkausmenetelmä eli tiedosto pakataan poistamalla siitä informaatiota. Tässä tapauksessa pikseleiden väriarvoja muutetaan yhtenäisemmiksi, jolloin tiedosto pakkautuu paremmin. Tavallinen JPEG-tiedosto ladataan ja piirretään selaimeen ylhäältä alas (kuvio 6.).



KUVIO 6. JPEG-kuvan latautuminen (Darn Office)

Progressiivisessa JPEG-tiedostossa on useita eri tarkkuuksia samasta kuvasta päällekkäin. Kuvasta näytetään käyttäjälle ensin huonolaatuinen versio, jonka laatu vähitellen paranee, kun tiedosto latautuu (kuvio 7.). Kun sisältö pystytään näyttämään nopeammin, vaikkakin huonolaatuisena, käyttäjä kokee sivun olevan suorituskykyinen. Progressiivinen JPEG-tiedosto myös pakkautuu paremmin kuin tavallinen JPEG-tiedosto. (Gediminas 2017, viitattu 29.12.2017.)



KUVIO 7. Progressive JPEG-kuvan latautuminen (Darn Office)

Kun tarvitaan häviötöntä pakkausta esimerkiksi yrityksen logon näyttämiseen, käytetään yleensä PNG-tiedostomuotoa. Tiedosto pakataan häviöttömästi, joten sen laatu on parempi, mutta tiedostokoko on myös suurempi. PNG-kuvia tulisi välttää, ellei ehdottomasti tarvita korkealaatuista, häviötöntä kuvaa.

Googlen kehittämä WebP-tiedostomuoto pakkautuu häviöllisellä pakkaus-menetelmällä 25–35 % pienemmäksi kuin tavallinen JPEG-tiedosto, ja häviöttömänä 26 % pienemmäksi kuin PNG. WebP-kuvia tukevat tällä hetkellä vain Chrome- ja Opera-selaimet, ja muille selaimille täytyy WebP-tiedoston sijasta lähettää JPG- tai PNG-tiedostot. (Google 2016, viitattu 29.12.2017.)

## 6.2 Lazyload-tekniikka

Kaikki verkkosivulla olevat kuvat eivät välttämättä ole heti näkyvissä käyttäjälle. Jos käyttäjä ei vieritä sivua alaspäin nähdäkseen kuvat, vaan navigoi heti toiselle sivulle, kuvat ladattiin ja renderöitiin täysin turhaan. Suorituskyky paranisi huomattavasti, jos vain näkyvissä olevat kuvat ladattaisiin.

Kuvien lataamista tarpeen mukaan kutsutaan lazyload-tekniikaksi. Se voidaan toteuttaa esimerkiksi selaimien IntersectionObserver-ohjelmointirajapinnan avulla. Kuvan lataaminen estetään poistamalla sen img-tunnisteelta src-attribuutti. Rajapinnalle annetaan lista HTML-elementeistä, joiden sijaintia verkkosivulla se seuraa. Kun elementti on selainikkunassa, ohjelma suorittaa sille ennalta määrätyn callback-funktion. Kuva ladataan lisäämällä sille callback-funktiossa src-attribuutti, jossa kuvatiedoston url-osoite on (kuvio 8.).

```
function lazyload(images) {
  const callback = entries => {
    entries.forEach(entry => {
      if (entry.intersectionRatio > 0) {
        entry.target.setAttribute('src', entry.target.dataset.src);
      }
    });
  });

  const intersectionObserver = new IntersectionObserver(callback);

  Array.prototype.map.call(images, image => {
    intersectionObserver.observe(image);
  });
}

export default lazyload;
```

KUVIO 8. Lazyload-tekniikka

## 7 WEB-FONTIT

Ulkoiset web-fontit eivät ole verkkosivun lukemisen kannalta kriittisiä resursseja. Tekstiä pystyy lukemaan käyttöjärjestelmään asennetuilla fonteilla. Web-fontit ovat kuitenkin luettavuuden ja yri-tyksen brändin kannalta tärkeitä. Fonttiedostot ovat tyypillisesti noin 10–20 kilotavun kokoisia. Jokainen eri leikkaus yhdestä fontista täytyy ladata erikseen, minkä vuoksi myös suunnittelijan täy-tyy pitää leikkausten määrä kohtuullisena. Web-fonttien lataaminen ei keskeytä dokumentin jäsen-tämistä.

Fonttien latauksessa on kolme eri vaihetta: esto, vaihto ja epäonnistuminen. Estovaiheessa tekstiä ei näytetä ollenkaan, ennen kuin web-fontti on latautunut, mikä on havaitun suorituskyvyn kannalta tärkeä asia. Vaihtovaiheessa näytetään toissijainen fontti ennen varsinaista fonttia. Fontin latauk-sen epäonnistuessa teksti renderöidään käyttöjärjestelmän oletusfontilla. Kaikki selaimet eivät nou-data näitä sääntöjä. Niissä tekstiä ei näytetä ollenkaan, ennen kuin web-fontti on ladannut, mikä on käyttäjän kannalta huono asia.

Fontit vaativat font-face-määrittelyn, jonka avulla selain löytää oikeat fonttiedostot. Määrittelyn voi tehdä joko CSS-tyylitiedostossa tai JavaScript-ohjelmassa. Molemmissa on hyötynsä ja hait-tansa. Tyylitiedoston lataaminen keskeyttää dokumentin jäsentämisen, ja kirjoitushetkellä Micro-softin Internet Explorer- ja Edge-selaimissa ei ole fonttien lataamiselle vaadittua rajapintaa Ja-vaScript-ohjelmille.

### 7.1 Viivytetty lataus

Koska web-fontit ovat vain visuaalinen lisä, ne kannattaa ladata vasta, kun tärkeämmät resurssit on ladattu. Fonttien viivytettyä lataamista varten on useita eri tekniikoita.

Tehokkain tapa ladata fontti viivytetysti on font-display-ominaisuus font-face-fonttimäärittelyssä (kuvio 9.). Sen avulla kehittäjä voi itse päättää käytetäänkö fontin latauksessa ensisijaisesti esto-vai vaihtojaksoa. Tämä keino estää fontin näkymättömyysjakson, mutta ei layout-tapahtumaa.

```

@font-face {
  font-family: 'Gingham';
  src: url('Gingham.woff2') format('woff2'), url('Gingham.woff') format('woff'),
  url('Gingham.ttf') format('truetype');
  font-weight: normal;
  font-style: normal;
  font-display: swap;
}

```

KUVIO 9. Font-display-ominaisuus

Toinen tapa ladata fontit viivytetysti on font loading -ohjelmointirajapinta. Sen käyttö vaatii, että fonttien face-face-määrittäminen on tehty JavaScript-ohjelmassa. Tämän tekniikan kanssa on suositeltavaa käyttää fontin vaihtamisessa luokkajohjaista ratkaisua näkymättömyysjakson poistamiseksi. Sivulla käytetään aluksi järjestelmän oletusfonttia, ja fontin latauduttua dokumenttivartaloon lisätään luokka, joka vaihtaa fontin web-fontiksi (kuvio 10.). Kirjoitushetkellä Microsoftin Internet Explorer ja Edge eivät tue font loading -rajapintaa.

```

function loadFont() {
  const font = new FontFace('Gingham', 'url(./Gingham.woff2)', {
    style: 'normal',
    weight: 'normal'
  });

  font.load().then(() => {
    document.body.classList.add('fonts-loaded');
  });
}

export default loadFont;

```

KUVIO 10. Font loading -ohjelmointirajapinta

Koska selaimet eivät tue font-display-ominaisuutta ja font loading -ohjelmointirajapintaa kovin hyvin, on suositeltavaa käyttää FontFaceObserver-lisäkettä JavaScript-ohjelmassa (kuvio 11.). Se mahdollistaa viivytetyn fonttien lataamisen kaikissa selaimissa. Ainoa haittapuoli siinä on, että sen käyttö lisää JavaScript-tiedoston painoa noin kymmenellä kilotavulla.

```

import FontFaceObserver from 'fontfaceobserver';

function loadFont() {
  const font = new FontFaceObserver('Gingham');

  font.load().then(() => {
    document.body.classList.add('fonts-loaded');
  });
}

export default loadFont;

```

KUVIO 11. FontFaceObserver-lisäkettä käyttävä JavaScript-moduuli

Verkkokauppa eBayn käyttämä tekniikka on ladata web-fontti taustalla välimuistiin, mutta näyttää se vasta seuraavalla sivunlatauksella. Tämä estää kokonaan fontin vaihdossa selaimessa suoritettavat layout-tapahtumat.

## 7.2 Varioidut fontit

Varioiduista fonteista voidaan luoda tuhansia eri leikkauksia vain yhden fontitiedoston pohjalta. Kirjainten paksuudelle ja leveydelle on molemmille tuhat eri arvoa, eli mahdollisia yhdistelmiä on yhteensä miljoona. Paksuuden ja leveyden lisäksi muita asetuksia ovat kaltevuus ja optinen koko. (Rutter 2017, viitattu 30.12.2017.)

Christoph Koeberlinin luoma kokeellinen Gingham-fontti oli yksi ensimmäisistä varioiduista fonteista. Se tukee vain paksuuden ja leveyden varioimista.

Koska jokaista leikkausta varten ei tarvitse ladata omaa fontitiedostoa, fonttien lataamiseen kuluva aika putoaa huomattavasti (kuvio 12.). Pienemmät tiedostokoot ovat parempia myös käyttöpohjaisesti laskutettuja liittymiä varten.

Selaimet tukevat varioituja fontteja kohtuullisen hyvin. Kirjoitushetkellä niitä tukevat Chrome ja Safari sekä niiden mobiiliversiot, mikä koskee yli 50 % Internet-käyttäjistä. Tuki varioiduille fonteille on tulossa myös Firefox-selaimen.

Lato Light

Lato Regular

**Lato Bold**

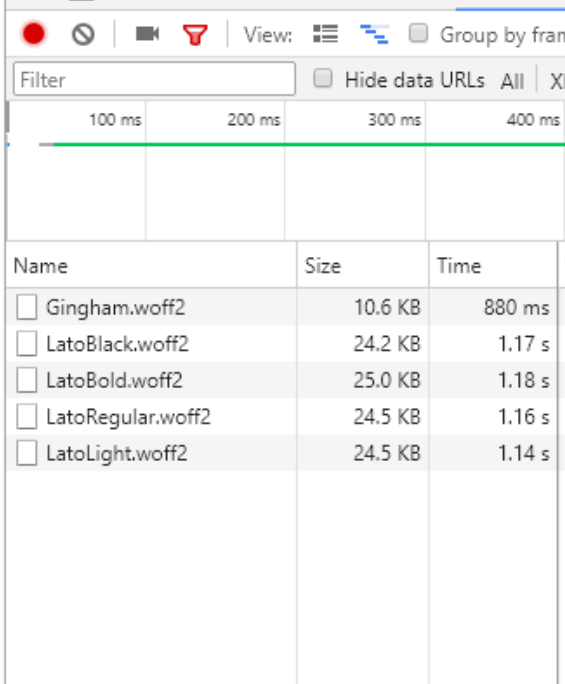
**Lato Black**

Gingham Light

Gingham Regular

**Gingham Bold**

**Gingham Black**



Name	Size	Time
<input type="checkbox"/> Gingham.woff2	10.6 KB	880 ms
<input type="checkbox"/> LatoBlack.woff2	24.2 KB	1.17 s
<input type="checkbox"/> LatoBold.woff2	25.0 KB	1.18 s
<input type="checkbox"/> LatoRegular.woff2	24.5 KB	1.16 s
<input type="checkbox"/> LatoLight.woff2	24.5 KB	1.14 s

KUVIO 12. Lato- ja Gingham-fonttien koot ja latausajat

## 8 LÄHTEET

A new image format for the web. 4.3.2016. Google Developers. Viitattu 29.12.2017, <https://developers.google.com/speed/webp/>

Brutlag, J. 23.6.2009. Speed Matters. Google Research Blog. Viitattu 17.12.2017, <https://research.googleblog.com/2009/06/speed-matters.html>

Darn Office. How to Always Deliver and Never Lose Sight of The Big Picture. Viitattu 31.12.2017 <http://darnoffice.com/always-deliver/>

Gediminas, B. 11.4.2017. Improving Website Performance – Using Progressive JPEG images. Viitattu 29.12.2017, <https://www.hostinger.com/tutorials/website/improving-website-performance-using-progressive-jpeg-images>

Koeberlin, C. Gingham – A Free Variable Font. Viitattu 31.12.2017, <http://koe.berlin/variablefont/>

Jakober, L. 12.3.2012. Are Companies Meeting Tablet User Expectations? Dynatrace. Viitattu 17.12.2017, <https://www.dynatrace.com/blog/are-companies-meeting-tablet-users-expectations/>

Kasireddy, P. 22.1.2016. JavaScript Modules: A Beginner's Guide. Viitattu 28.12.2017, <https://medium.freecodecamp.org/javascript-modules-a-beginner-s-guide-783f7d7a5fcc>

Linden, G. 28.11.2006. Make Data Useful. Google Sites. Viitattu 17.12.2017, <http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt>

Osmani, A. 15.11.2017. The Cost of JavaScript. Medium. Viitattu 28.12.2017, <https://medium.com/dev-channel/the-cost-of-javascript-84009f51e99e>

Rutter, R. 21.2.2017. Get started with variable fonts. Medium. Viitattu 30.12.2017, <https://medium.com/@clagnut/get-started-with-variable-fonts-c055fd73ecd7>



World Wide Web Consortium 3.10.2017. HTML 5.1 2nd Edition. Viitattu 12.12.2017,  
<https://www.w3.org/TR/html51/introduction.html>