KARELIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma

Jani Gröhn

KÄYTETTÄVYYDEN HUOMIOIMINEN JA SUORITUSKYVYN OPTIMOINTI VERKKOSIVUSTOA RAKENNETTAESSA

Opinnäytetyö
Toukokuu 2018

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Toukokuu 2018
Tieto- ja viestintätekniikan koulutusohjelma

Karjalankatu 3
80200 JOENSUU
(013) 260 600

Tekijä
Jani Gröhn

Nimeke
Käytettävyyden huomioiminen ja suorituskyvyn optimointi verkkosivustoa rakennettaessa

Toimeksiantaja
Koodiviidakko Oy

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli tehdä verkkosivustopohja Koodiviidakko Oy:lle ja selvittää, kuinka verkkosivuston rakentaja voi parantaa sivuston käytettävyyttä ja suorituskykyä. Verkkosivuston käytettävyyteen huomioitiin myös sen saavutettavuus, ja opinnäytetyössä huomioitava suorituskykyoptimointi keskittyy asiakaspuolella tehtäviin optimointeihin.

Tehdyssä sivustopohjassa hyödynnettiin opinnäytetyön teoriatiedossa olevia hyvän käytettävyyden ohjeita sekä suorituskykyoptimointitapoja. Sivustopohja on valmis ja käytettävä sivusto, jossa on oikean sisällön sijaan esimerkkitekstiä ja -kuvia. Sivustopohjasta saadaan asiakkaalle verkkosivusto kopioimalla se, vaihtamalla sinne oikeaa sisältöä ja tekemällä siihen asiakkaan haluamat tyylimuutokset.

Sivustopohjaa testattiin käytettävyyden ja suorituskyvyn osalta. Käytettävyystestaus tehtiin käyttämällä sivustoa ja sen toimintoja ainoastaan näppäimistöä käyttäen sekä NVDA-ruudunlukijaohjelman kanssa. Suorituskykytestauksissa käytettiin Google Chromen kehittäjätyökaluja, Googlen Lighthouse-palvelua sekä WebPagetest-sivulataustestiä.

Kieli
suomi

Sivuja 73
Liitteet 4
Liitesivumäärä 7

Asiasanat
verkkosivusto, käytettävyys, saavutettavuus, suorituskykyoptimointi

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2018
Degree Programme in Information and Communications Technology

Karjalankatu 3
80200 JOENSUU
FINLAND
(013) 260 600

Author
Jani Gröhn

Title
Usability Consideration and Performance Optimization in Building a Website

Commissioned by
Liana Technologies

Abstract

The purpose of this thesis was to create a website template for Liana Technologies and explain the ways of how a website developer can improve the usability and performance of a website. The website's accessibility was considered as a part of usability, and performance optimization methods in this thesis concentrate on the client-side optimizations.

Guidelines of good usability and methods of performance optimizations mentioned in the theory section of this thesis were utilised on the established website template. The website template is a complete and usable website which contains example text and images instead of real content. The website template can be used to create a client's website by copying the website installation, switching website contents into real ones and by making the styles changes requested by the client.

The website template was tested in terms of usability and performance. Usability was tested by using the website and its functionalities with only a keyboard and by using a NVDA screen reader. Performance testing was made with Google Chrome's developer tools, Google's Lighthouse service and WebPagetest page speed test.

Language

Finnish

Pages 73
Appendices 4
Pages of Appendices 7

Keywords

website, usability, accessibility, performance optimization

Sisältö

Liitteet

Liite 1 Sivuston muuttujat -tiedosto
Liite 2 JavaScript-koodi taustakuvan srcset-toiminnallisuuden saavuttamiseksi
Liite 3 Verkon nopeuden vaihtelun vaikutus sivulatauksen eri vaiheiden valmistumiseen, kaikki testitulokset eriteltynä.
Liite 4 Verkon viiveen vaihtelun vaikutus sivulatauksen eri vaiheiden valmistumiseen, kaikki testitulokset eriteltynä.

1 Johdanto

Internet on nykymaailman tietosanakirja, viihdekeskus sekä mainostoimisto. Sitä käyttää yli 3,8 miljardia ihmistä maailman koko väestöstä (Internet World Stats 2017), ja verkkosivut ovat tavalliselle käyttäjälle ylivoimaisesti suosituin tapa saada tietoa internetistä. Tämän vuoksi verkkosivustojen tulisi olla sujuvasti toimivia kokonaisuuksia, joita jokainen pääsee selaamaan esteettömästi. Kuitenkin nykypäivänä verkkosivustojen rakentamisen haasteena ovat erilaisien verkkoselaamisen mahdollistavien laitteiden iso määrä, internetyhteyksien nopeuden ja toimintavarmuuden suuri vaihtelu sekä käyttäjien mahdolliset esteellisyydet. Jättämällä nämä asiat huomiotta tuloksena voi olla hidas sivusto, jonka käyttö on vaikeaa tai jopa kokonaan estynyt osalle käyttäjistä.

Tässä opinnäytetyössä on esitelty verkkosivustojen yleisimmät käytettävyysongelmat ja käyttäjäpuolen suorituskykyongelmat sekä autetaan niiden selvittämisessä. Ongelmia ja ratkaisuja on selvitetty teoriatiedon lisäksi rakentamalla esimerkkisivusto ja testaamalla sen toimivuutta kyseisten aihealueiden osalta. Esimerkkisivustoksi rakensin pienyrityksen sivustopohjan, joka sisältää yleisiä pienyrityksien vaatimia ominaisuuksia. Sivustopohja tulee Koodiviidakko Oy:n käyttöön, ja siitä saadaan toimiva verkkosivusto vaihtamalla sisältö, värit ja logot asiakkaan yrityksen mukaiseksi.

2 Verkkosivuston käytettävyys

Verkkosivuston käytettävyys on laaja käsite, jolla tarkoitetaan yleisesti sivuston käyttäjäkokemuksen helppoutta ja intuitiivisuutta. Käyttäjäkokemukseen voidaan laskea kuuluvan sivuston hyödyllisyys, helppokäyttöisyys, navigoitavuus, saavutettavuus, uskottavuus sekä toivottava sisältö (U.S. Department of Health and Human Services, 2017). Käytettävyydelle ei ole olemassa erillistä mitta-asteikkoa, vaan arvioiminen tapahtuu pohtimalla, kuinka hyvin yllä mainitut käytettävyyden kriteerit toteutuvat sivustolla.

Käytettävyyden huomioiminen on tärkeää, sillä yleensä verkkosivuista halutaan tehdä sellaiset, että ne hyödyttävät mahdollisimman montaa käyttäjää ja että käyttäjät palaisivat sivustolle uudelleen myöhemmin. Syynä on usein taloudelliset intressit, joihin sivuston käytettävyys vaikuttaa suuresti. Tutkimuksen (Temkin Group 2017) mukaan jopa 39 % erittäin huonon käyttäjäkokemuksen kokeneista käyttäjistä vähensi tai lopetti kokonaan kyseiseen yritykseen kohdistuvan rahankäytön. Jokainen käyttäjä on tärkeä, ja vaikka sivuston tarkoituksena ei olisikaan kaupallinen hyöty, tulisi hyvän käyttökokemuksen antaminen olla jokaisen verkkosivuston perustavoite.

Verkkosivujen käytettävyys on otettu huomioon myös lainsäädännöllisesti. 26. elokuuta 2016 säädettiin Euroopan unionin direktiivi 2016/2102, joka asettaa saavutettavuuden vähimmäisvaatimukset julkisen sektorin verkkosivustoille ja mobiilisovelluksille. Säädännön tarkoituksena on parantaa julkisten palveluiden saavutettavuutta, ja se vaikuttaa kaikkien julkisen sektorin hallitsemien yhteisöjen verkkosivuihin ja mobiilisovelluksiin.

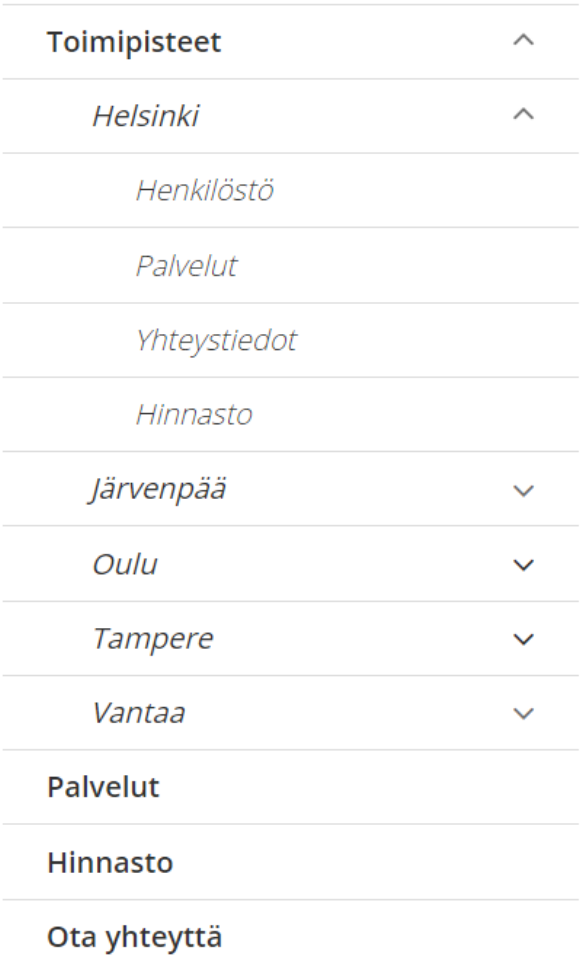
2.1 Käytettävyys sivuston rakenteessa ja ulkoasussa

Käytettävyys tulee ottaa huomioon jo verkkosivuston suunnitteluvaiheessa, sillä sekavaa sivustorakennetta on hankalaa ja kallista muuttaa jälkeenpäin. Rakennetta ja ulkoasua tehtäessä on hyvä muistaa, että sivustoa käyttävät erilaiset ihmiset, eikä heillä kaikilla ole samanlaista tottumusta verkkoselaamiseen kuin sivuston rakentajalla itsellään.

2.1.1 Loogisuus ja yksinkertaisuus

Verkkosivuston täytyy olla koosta riippumatta looginen kokonaisuus, jota käyttäjä voi selata helposti. Sivujen tulee olla jaoteltu järkevästi aihealueen mukaan ylä- ja alasivuihin, eikä sivuhierarkiassa saisi olla liikaa tasoja. Sopiva alasivutasojen

määrä riippuu koko sivuston laajuudesta, sillä liian vähän alasivuja laajalla sivustolla tarkoittaa suurta määrää päätason sivuja, joka voi aiheuttaa laajan ja sekavan navigointivalikon. Kuitenkin jo yli kolmen sisäkkäisen tason sivuhierarkiaa kannattaa miettiä tarkoin. Tällöin voi miettiä olisiko järkevämpää esimerkiksi tehdä kokonaan uusi yläsivu, tai lajitella sivut uudelleen eri ominaisuuden perusteella. Rakennetta suunnitellessa tulee ottaa huomioon myös tulevaisuuden tarpeet. Hyväkin sivurakenne voi paisua laajaksi ja epäloogiseksi, kun myöhemmin lisätään sivuja, jotka eivät sovi minkään valmiiksi tehdyn jaottelun alle.



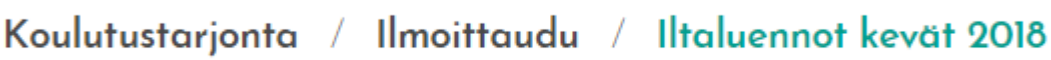
Kuva 1. Looginen sivujaottelu hieman laajemmalla sivustolla.

Yksittäistä sivua tehdessä kannattaa muistaa, että käyttäjän kannalta yksinkertaisempi on parempi. Liiallinen toisiinsa liittymättömien tekstien tai kuvien käyttäminen samassa näkymässä voi aiheuttaa käyttäjälle informaatiotulvan, ja eri asiayhteydet kannattaakin erottaa esimerkiksi tarpeeksi suurilla väleillä. Kuiten-

kin sivun jakamista sivuttaissuunnassa kannattaa käyttää harkiten, sillä usein lukija yhdistää sivuttain olevat asiat helposti toisiinsa esimerkiksi vertailtavina kohteina.

Sivulla käytettävien elementtien, kuten kuvien, tekstien ja valikoiden, täytyy olla selkeitä. Käyttäjän pitää pystyä erottamaan esimerkiksi sivustolla oleva informatiivinen teksti ja mainosteksti toisistaan. Hillityillä tyyleillä ja hyvällä asiayhteyksien erottelulla saadaan aikaan selkeä kokonaisuus. Tämän lisäksi ylimääräisiä ja liian monimutkaisia animaatioita tulee välttää. Animoinnilla tulee aina olla jokin tarkoitus, kuten esimerkiksi viestiä jostain muutoksesta. Hyvä esimerkki animaation käytöstä on antaa käyttäjälle edistymistietoa ladatessa uutta sisältöä.

On myös tärkeää antaa käyttäjälle aina tieto siitä, millä sivulla hän on, sekä mitkä ovat nykyisen sivun yläsivuja. Näin käyttäjä tietää heti mistä sivulla kerrotaan sekä kuinka se on kategorisoitu sivustolla, vaikka hän olisikin päätynyt sivulle ulkoisen linkin kautta. Tämä on tärkeää varsinkin laajemmilla sivustoilla, joissa voi olla useampia samannimisiä sivuja eri yläsivujen alla. Sivupolun esittämiseen on olemassa suosittu ja käytännöllinen ratkaisu nimeltään murupolku (engl. Breadcrumb). Murupolku näyttää käyttäjälle sivupolun suhteessa nykyiseen sivuun, ja päästäkseen edellisille sivuille käyttäjä voi klikata linkkejä murupolusta. Murupolun lisäksi on hyvä avata nykyinen sivupolku mobiilivalikossa, sillä siten valikon aukaisemalla käyttäjä näkee suoraan millä sivulla hän on, ja pääsee vierailemaan sen sisarsivuilla helposti.



Kuva 2. Murupolku.

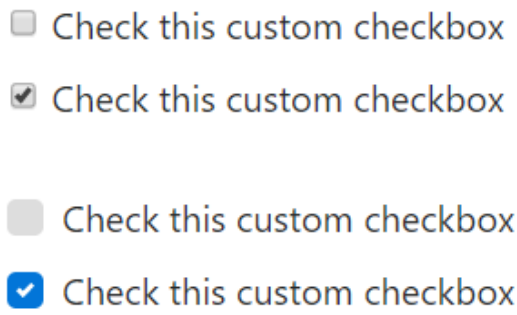
2.1.2 Tuttuus ja yhtenäisyys

Joskus näkee verkkosivustoja, joihin omistaja on halunnut vaihtaa uuden teeman, mutta on ajatellut olla muuttamatta vanhojen sivujen ulkoasua. Sen sijaan vain kaikki uudet sivut rakennettaisiin uuden ilmeen mukaisesti. Taloudellisesti

ajatellen tämä on hetkellisesti ihan hyvä ratkaisu, mutta samalla sivuston käyttäjäkokemus huononee. Käyttäjän selatessa tuttua sivustoa ja siirtyessä uudelle sivulle koko sivuston ilme vaihtuu. Pahimmassa tapauksessa myös toiminnallisten elementtien, kuten valikoiden ja linkkilistojen, paikat vaihtuvat täysin. Sivun teeman vaihtuessa käyttäjän työnkulku pysähtyy ja iskee epäilys siitä, että painoiko hän väärää linkkiä ja päätyi jollekin toiselle sivustolle. Varmistettuaan että sivusto on sama, käyttäjä joutuu totuttelemaan uuden sivurakenteen käyttöön ja silti käyttämään vielä vanhaa rakennetta joillain sivuilla. Erityisen pahasti tällainen sivuston sisäinen teeman vaihtelu vaikuttaa ruudunlukuohjelman käyttäjiin. Se, että joillain sivuilla valikko on kohdistusjärjestyksessä ensimmäisenä ja toisilla vasta neljäntenä aiheuttaa sen, että käyttäjän täytyy kuunnella kaikki mitä ruudunlukija lukee, eikä hän esimerkiksi voi helposti ohittaa valikkoa jonka hän tietää olevan aina ensimmäisenä kyseisellä sivustolla.

Yhtenäisen sisäisen teeman lisäksi verkkosivuston tulee käyttää toiminnallisia HTML-elementtejä oikein, eikä niiden tarkoitusta saa muuttaa. Toiminnallisilla HTML-elementeillä tarkoitetaan käyttöliittymässä olevia painikkeita kuten painike, linkki, valintaruutu tai pudotusvalikko. Suurin osa internetin käyttäjistä ovat tottuneet näihin verkkosivustojen peruspalikoihin ja siksi niiden oikeaoppinen käyttö varmistaa käyttäjälle omalta osaltaan tutun ja sujuvan käyttökokemuksen.

Vääränlainen toiminnallisten elementtien käyttö on niiden alkuperäisen toimintaidean tai ulkoasun muuttamista siten, että käyttäjä ei enää tunnista niiden oikeaa tarkoitusta. Esimerkiksi pudotusvalikon nuolen piilottaminen voi aiheuttaa sen, että käyttäjä luulee sen olevan tavallinen tekstikenttä. Kuitenkin elementtien ulkoasujen hillitty muuntaminen on sallittua, kunhan uudesta ulkoasusta saa vielä helposti selville mikä elementti on ja kuinka sen tulisi toimia. Esimerkiksi ”Lue lisää” -linkki voidaan ulkoasullisesti muotoilla painikkeen näköiseksi, sillä kyseisen ”painikkeen” teksti kertoo selkeästi käyttäjälle mitä tapahtuu, kun sitä klikataan. Myös syöttökenttien ulkoasujen muuttaminen on yleistä, esimerkiksi useissa valmiissa teemoissa tai komponenttikirjastoissa tyylien muutos on tehty hillityllä ja käyttäjäystävällisellä tavalla.



Kuva 3. Valintaruudut tyylittelemättöminä sekä komponenttikirjasto Bootstrap 4:n valmiilla teemalla.

Toiminnallisten HTML-elementtien toimintatarkoituksen muuttaminen ei ole hyväksyttyä. Se voi aiheuttaa käyttäjässä epävarmuutta, koska silloin hän ei enää tiedä miten ennestään tuttu elementti toimii. Esimerkiksi valintaruutujen asettaminen siten, että vain yhden ruudun voi valita kerrallaan rikkoo täysin valintaruudun toiminnallisuuden yleisiä normeja ja muuttaa sen alkuperäistä käyttötarkoitusta. Alkuperäisillä käyttöliittymäelementeillä saadaan helposti tehtyä kaikki käyttäjän ja tietokoneen välinen perustoiminnallisuus verkkosivustolle, ja esimerkin tapauksessakin valintanapin käyttö valintaruudun sijaan ajaa saman asian, mutta käyttäjäystävällisesti.

Valmiiden käyttöliittymäelementtien lisäksi on muita tapoja pitää käyttöliittymä mahdollisimman käyttäjäystävällisenä. Vakiintuneet ikonit ovat yksi hyvä tapa vähentää epäselvyyksiä ja liiallisia selitetekstejä sivustolla. Laajalti tunnettujen ikonien käyttö on suositeltavaa, koska siten käyttäjä tietää ilman näkyvää selitetekstiä mitä se tarkoittaa. Esimerkiksi valikon ikoniksi on vakiintunut niin kutsuttu hampurilaisvalikko, ja päivitä näkymä -ikoniksi kehää kiertävä nuoli.

Kuva 4. Hampurilaisvalikko ja päivitä näkymä -ikonit.

Ikoneiden avulla on tarkoitus antaa käyttäjälle nopeasti tieto mihin asia liittyy tai mitä painike tekee. Sen vuoksi vähemmän tunnettuja tai itse suunniteltuja ikoneja käyttäessä täytyy muistaa pitää ikonit ulkoasultaan mahdollisimman yksinkertaisina. Ulkoasullisesti liian monimutkainen tai värikäs ikoni voi aiheuttaa sen, että

käyttäjä ei heti ymmärrä mitä se tarkoittaa, varsinkin jos käyttäjällä on heikko näkö tai värisokeutta. Tämän lisäksi vakiintumattomissa ikoneissa tulee käyttää myös selitetekstiä. Jos ikoni kuvastaa selvää toiminnallisuutta, esimerkiksi lähetä viesti -painiketta, selitetekstin tulee olla saatavilla vähintään näppäimistöllä kohdistaessa (engl. focus) sekä ruudunlukijaa käytettäessä[1]. Vakiintuneille ikoneille seliteteksti tulee olla vähintään ruudunlukijalla luettavissa.

Muut kuin toiminnalliset ikonit ovat usein otsikon tai tekstin yhteydessä auttamassa aihealueiden jäsentelyä. Käyttäjä näkee ikonit nopealla silmäyksellä ja näin löytää helposti haluamansa kohdan. Näissä tapauksissa ikonin selitetekstinä toimii kyseinen otsikko tai teksti, johon ikoni liittyy. Vakiintumattomien ikoneiden tapauksessa pelkällä ikonilla ilman selitetekstiä ei saa korvata tekstimuotoista otsikkoa tai selitettä.

2.1.3 Erilaisten laitteiden huomioiminen

Verkkosivustoa rakentaessa on hyvä muistaa, että käyttäjät selaavat sivustoa erilaisilla laitteilla, jolloin käyttäjäkokemus ei ole kaikilla samanlainen. Esimerkiksi sivun helppokäyttöisyyteen ja sisällön visuaaliseen jaotteluun kannattaa kiinnittää huomiota rakennettaessa sivustoa myös mobiililaitteille.

Selaimet osaavat automaattisesti suurentaa verkkosivujen elementtien kokoa näytölle sopivaksi, toisin sanoen kuvaa renderöidessä näytön pikselin ja CSS-pikselin koko ei ole aina sama. Esimerkiksi 18 pikselin fonttikoko ei näy viisituumaisella 1920 x 1080 resoluutioisella puhelimen näytöllä oikeasti 18 pikselin korkeudella, vaan se suhteutetaan sopivaksi näytön pikselitiheyden perusteella. Sivuston rakentajan ei siis tarvitse huomioida laitteiden eri pikselitiheyksiä visuaalisten elementtien kokoa miettiessä. Tästä huolimatta täytyy muistaa, että täysikokoisella tietokoneella navigointiin käytetään pientä kursoria ja mobiililaitteilla yleensä sormea. Tämän vuoksi mobiililaitteella pieniin nappeihin on huomattavasti vaikeampi osua kuin hiirtä käyttäessä, ja painikeanimaatiokin jää

[1] Elementtien saavutettavuuskäytännöistä lisää osiossa 2.2.3

usein sormen alle piiloon. Pienillä näyttökoilla pitää varmistaa, että toiminnallisten elementtien koko on tarpeeksi suuri, jotta niihin on helppo osua kosketusnäytölläkin.

Kosketusnäytölliset laitteet eivät käytä CSS:n tai JavaScriptin hover ja focus -tapahtumia samalla tavalla. Hiirtä käytettäessä kursorin sijaintia voidaan seurata koko ajan, mutta se ei ole mahdollista kosketusnäytöllä sormea käytettäessä. Focus-tapahtuma alkaa vain kosketuksella ja usein tapahtuma jää tahtomatta päälle, kun käyttäjä jatkaa sivun vierittämistä alaspäin. Sivuston rakentajan täytyy siis muistaa asettaa esimerkiksi hover-tapahtumalla näytettävä teksti saataville myös klikkaamalla.

On myös hyvä muistaa, että pienen näytön yhdelle sivunäkymälle mahtuu vähemmän asiaa kuin suurelle näytölle. Tämä ei ole yleensä ongelma, sillä teksti ja kuvat laitetaan mobiililaitteilla usein allekkain, mutta asia tulee ottaa huomioon tilanteissa jolloin käyttäjän tulisi nähdä paljon tietoa kerralla. Esimerkiksi taulukoiden tai tuotevertailujen näkymät saatetaan joutua suunnittelemaan aivan eri tavalla mobiililaitteille kuin täysikokoisten tietokoneiden näytöille.

2.2 Saavutettavuuden huomioiminen

World Health Organizationin mukaan noin 15 prosentilla kaikista maailman ihmisistä on jonkinasteinen esteellisyys, joka haittaa heidän normaalia toimintaa. Saavutettavuuden ideana on, että myös jokainen näistä käyttäjistä pääsee selaamaan verkkosivustoa ongelmitta. Käyttäjäkohtaisia esteellisyyksiä voivat olla esimerkiksi värisokeus tai yleisen näkökyvyn heikkous. Myös esimerkiksi toisen käden vammautuminen voi aiheuttaa sen, että käyttäjä voi selata sivustoa vain pelkällä hiirellä tai näppäimistöllä. Verkkosivustoon liittyvään saavutettavuuden määritelmään liittyy olennaisesti se, että jokainen käyttäjä on tärkeä ja jokaisella tulisi olla mahdollisuus käyttää sivustoa niin luontevasti kuin hänen rajoituksiensa puitteissa on mahdollista.

2.2.1 Värien käyttö

Täytyy muistaa, että verkkosivuja käyttävät myös ihmiset, joilla on heikentynyt värinäkö. Tämän takia värejä ei tulisi koskaan käyttää ainoana keinona ilmoittaa muutoksesta, kertoa toiminnallisuudesta, erotella asiayhteyksiä tai muuten välittää informaatiota käyttäjälle (Web Accessibility Initiative 2017). Web Content Accessibility Guidelines 2.1 (lyh. WCAG 2.1) määrittelee tämän perustason (A) vaatimukseksi, ja sen tulisi toteutua jokaisella verkkosivustolla. Tyypillinen esimerkki värien käyttämisestä väärin on se, kun lomakkeen täytön onnistumisesta tai epäonnistumisesta kertovat viestit on eroteltu ainoastaan väreillä, esimerkiksi värjäämällä tekstikentän reunus vihreäksi tai punaiseksi. Tällaisessa tilanteessa punavihersokean on täysin mahdotonta erottaa virheellistä ja virheetöntä syötettä toisistaan. Oikea ratkaisu olisi liittää kentän viereen värin lisäksi myös teksti tai ikoni, joka kertoo lopputuloksesta.

Tekstin ja taustan värien kontrastisuhteen tulee olla tarpeeksi suuri heikkonäköisiä varten. WCAG:n AA-tason luokitus vaatii tekstin ja taustavärin minimikontrastisuhteeksi 4.5:1, paitsi suurelle tekstille (koko vähintään 14 pt ja lihavoitu, tai koko vähintään 18 pt) vaatimus on 3:1. Suuremman AAA-tason luokitus vaatii kontrastisuhteen 7:1, tai suuremmalle tekstille 4.5:1. Käytännössä näiden suurempien vaatimustasojen täydellinen saavuttaminen rajoitetuilla teeman väreillä voi olla erittäin vaikeaa, minkä vuoksi mielestäni on riittävää, kun pyritään mahdollisimman hyvään tulokseen teeman värejä mukaillen. Apuna taustavärien valintaan voi käyttää verkosta löytyviä esteettömien väriyhdistelmien luontiin tarkoitettuja työkaluja tai Google Chrome -versio 65:n mukanaan tuomaa kehittäjätyökalujen värivalitsinta, joka näyttää saavutettavat värivaihtoehdot suhteessa taustan väriin.

WCAG:ssä olevat kontrastivaatimukset eivät kuitenkaan koske ei toiminnassa olevia käyttöliittymäelementtejä, koristeeluun käytettävää tekstiä, piilotettua tekstiä, logossa sijaitsevaa tekstiä tai tekstiä, joka on osana paljon muuta visuaalista sisältöä sisältävää kuvaa. Näin esimerkiksi lomakkeen väliaikaisesti pois käytöstä oleville painikkeille ei ole vähimmäisvaatimuksia, koska ne kuuluvat ei toiminnassa olevien käyttöliittymäelementtien kategoriaan. Tämä on mielestäni

väärin, sillä myös päältä pois kytkettyjen toiminnallisten elementtien tulisi olla kaikille helposti luettavissa. Nykyisessä tilassaan ohjeistus sallii siis melko yleisen ongelman päältä pois kytkettyjen (engl. disabled) kenttien tai painikkeiden tapauksessa. Esimerkiksi joillain sivustoilla lähetä-painike on kytketty pois päältä, kunnes kaikki kentät on täytetty oikein, ja usein päältä pois kytketyn elementin värit heikennetään vaihtamalla värit vaaleammiksi. Tämä aiheuttaa sen, että kenttien tai painikkeiden teksti ei erotu tarpeeksi hyvin taustaväristä, ei aina edes näköongelmattomalle käyttäjälle.

Kuva 5. Heikkokontrastinen ja pois päältä kytketty lähetä-painike.

2.2.2 Hiirettömyys

Verkkosivustot täytyy rakentaa niin, että niitä pystytään käyttämään myös ainoastaan hiirellä tai näppäimistöllä. Pelkän hiiren käytön mahdollistaminen ei normaalisti aiheuta mitään toimenpiteitä sivuston rakentajalta, sillä hiiren ohella käyttäjä voi usein käyttää virtuaalista näppäimistöä. Vaikka virtuaalista näppäimistöä ei olisikaan saatavissa, käyttäjä voi silti käyttää kirjoittamista lukuun ottamatta kaikkia HTML:n perustoimintoja hiirellä.

Verkkosivun visuaalinen selaaminen pelkällä näppäimistöllä onnistuu helposti, sillä nuolinäppäimillä voi selata sivua ylös ja alas samalla tavalla kuin hiiren rullalla. Ongelmallisempaa on toiminnallisten elementtien käyttäminen, mutta onneksi selaimet antavat käyttäjälle tähän hyvät mahdollisuudet. Käyttäjä voi selata toiminnallisia elementtejä läpi käyttämällä sarkainnäppäintä. Sarkainnäppäimellä pääsee seuraavaan elementtiin, ja painamalla samaan aikaan vaihtonäppäintä pääsee takaisin edelliseen elementtiin. Painikkeita tai linkkejä voi painaa enter-näppäimellä ja esimerkiksi valintaruudun voi valita käyttämällä välilyöntinäppäintä. Kaiken sivustolla olevan toiminnallisuuden, kuten lomakkeiden täytön tai pudotusvalikon selaamisen, tulisi olla käytettävissä myös pelkästään näppäimistöä käyttäen.

Käyttäjälle selain näyttää senhetkisen kohdistettuna olevan elementin asettamalla sille ääriviivat (engl. outline), usein joko yhtenäiset siniset tai katkonaiset harmaat viivat. Näin käyttäjä näkee missä elementissä hän on menossa käyttäessään sarkainnäppäintä niiden selaamiseen. Tarkemmin sanottuna sarkainnäppäin selaa toiminnallisia elementtejä asettamalla nykyiselle elementille kohdistuksen (engl. focus) päälle. Ääriviivat tulevat siis näkyviin myös hiirellä klikkaamalla, ja sen vuoksi tietämättömät sivuston rakentajat saattavat piilottaa ne CSS-määrityksillä, koska ne näyttävät heidän mielestään rumilta. Näin ei siis tulisi tehdä, sillä se estää sivuston käytön kaikilta vain näppäimistöä käyttäviltä käyttäjiltä.

Jos ääriviivat halutaan välttämättä pois niin oikeaoppisin tapa siihen olisi laittaa ne pois päältä niin kauan, kunnes käyttäjä klikkaa sarkainnäppäintä. Näin tunnistetaan näppäimistöä käyttävä henkilö, ja ääriviivat näytetään vain tarpeen tullen. Sarkainnäppäimen käytön tunnistaminen tapahtuu käyttämällä JavaScriptiä, joten myös noscript-varmistus tulee muistaa asettaa niitä käyttäjiä varten, jotka ovat estäneet JavaScriptin käytön[2]. Kuitenkin varmin ratkaisu on olla piilottamatta ääriviivoja ollenkaan, sillä ne ovat osa selaimen natiivitoimintoja ja käyttäjät ovat varmasti jo tottuneet niihin.

Halutessaan ääriviivojen ulkoasua voidaan kuitenkin muokata helposti. Useimmat selaimet antavat vaihtaa esimerkiksi ääriviivojen väriä käyttämällä outline-color -määritystä CSS:ssä. On myös mahdollista korvata ne kokonaan käyttämällä sen sijaan reunuksia (engl. border). Näin saadaan ääriviivoille paljon enemmän kustomointimahdollisuuksia. Tekijän tulee kuitenkin olla erittäin tarkkana, että toiminnallisuus säilyy samana eli ääriviivat asetetaan kaikkien samojen elementtien kohdistustapahtumille. Ääriviivoja muokatessa tulee myös muistaa, että niiden värityksen tulee olla sellainen, joka erottuu taustasta.

Näppäimistökäyttäjä voi kohdistaa kaikkiin toiminnallisiin elementteihin sarkainnäppäintä painamalla. Tämä tarkoittaa käytännössä

[2] JavaScript-estosta lisää luvussa 2.3.

- linkkejä, joille on määritelty href-ominaisuus
- tekstikenttiä, tekstialueita sekä pudotusvalikoita, joilla ei ole disabled-ominaisuutta
- nappeja, joilla ei ole disabled-ominaisuutta
- elementtejä, joilla tabindex-ominaisuus on asetettuna arvoon 0 tai positiiviseksi kokonaisluvuksi
- elementtejä, joilla on contenteditable-ominaisuus asetettuna arvoon ”true”.

Tarvittaessa elementtejä saadaan siis näppäimistöllä kohdistettaviksi asettamalla tabindex-ominaisuus nollaksi. Tällöin selain tunnistaa, että elementti on kohdistettavissa, ja että kohdistusjärjestys on DOM:in (Document Object Model) elementtien järjestyksen mukainen. Tabindex-ominaisuutta ei kannata asettaa yli nollaksi, sillä on helpompaa antaa selaimen hoitaa elementtien kohdistamisjärjestys. Muuten kaikkien elementtien tabindex-arvot jouduttaisiin vaihtamaan uudelleen, jos uusi elementti tulisi vanhojen väliin. Elementti saadaan näppäimistöllä kohdistamattomaksi asettamalla tabindex-ominaisuudeksi arvo ”-1”.

Normaalisti elementtien kohdistamisjärjestys on siis DOM:in elementtien järjestyksen mukainen. On suositeltavaa käyttää normiksi tullutta käytäntöä asettaa sivulle ensin ylätunniste ja navigointilinkit, sitten sivun pääsisältö ja viimeiseksi alatunniste. Tämä on erityisen tärkeää näppäimistökäyttäjien kannalta, sillä kun sivun rakenne on tuttu, sen selaaminenkin on sujuvampaa. Näin käyttäjällä on yleiskäsitys sivun rakenteesta, ja selatessaan hän voi päätellä mikä elementti on seuraavana. Erityisen hyvää käytettävyyttä olisi laittaa sivulle ensimmäiseksi ankkurilinkki, jota painamalla näppäimistökäyttäjä hyppää navigaation yli ja pääsee suoraan sivun sisältöön. Tämä vähentäisi käyttäjältä sarkainnäppäimellä selaamista, kun hän selaa useampia sivuja sivustolla.

2.2.3 Ruudunlukuohjelma

Ruudunlukuohjelma antaa myös heikkonäköisille sekä täysin näkökyvyttömille käyttäjille mahdollisuuden käyttää tietokonetta ja selata verkkosivustoja. Selainta käytettäessä yleisimmät ruudunlukuohjelmat toimivat siten, että käyttäjä navigoi

elementtejä läpi, ja ohjelma lukee hänelle mitä kohdistettu elementti sisältää. Normaali leipäteksti luetaan sellaisenaan, mutta esimerkiksi kuvissa tai linkeissä useat ruudunlukuohjelmat ilmoittavat mikä elementti on kyseessä ennen kuin lukee sen sisällön. Esimerkiksi ”Hinnasto” -linkki voidaan lukea käyttäjälle ”hinnasto, linkki”.

Ruudunlukuohjelmat pystyvät toimimaan eri kielillä käyttämällä erilaista ääntämistä luettavasta kielestä riippuen. Verkkosivuilla ruudunlukuohjelma saa tiedon sivun kielestä katsomalla html-tagin lang-ominaisuutta. Lang-ominaisuudessa tulee olla arvona kielen ISO 639-1 -lyhenne, esimerkiksi englannin kielelle se on ”en” ja suomenkielelle ”fi”. Joskus kuitenkin koko sivun kieli ei ole sama, sillä tekstin seassa voi olla erikielinen nimi tai lainaus. Ruudunlukuohjelma lukee tämän lang-ominaisuuteen määritellyllä kielellä, jolloin voi ääntäminen voi olla täysin väärä ja siksi käyttäjä ei saa tekstistä mitään selvää. Korjauksena tähän on laittaa kyseinen teksti erilliseen span-elementtiin sisällöksi, ja asettaa span-elementtiin lang-ominaisuudeksi tekstin kieli. Tällöin ruudunlukuohjelma vaihtaa ääntämisen kyseisen tekstin ajaksi erikseen määritellylle kielelle.

Erilaisia ruudunlukuohjelmia on monia, mutta niissä kaikissa on samanlainen perustoiminnallisuus. Kaikilla on tarkoituksena lukea näytöllä olevaa tekstiä heikkonäköisille, ja eri ruudunlukuohjelmien erot keskittyvätkin yleensä erilaisiin pikatoimintoihin tai erimuotoisiin lausuntoihin ilmoittaessa elementin tyyppiä. WebAIM:in teettämän kyselyn mukaan vuonna 2017 selvästi suosituimmat ruudunlukuohjelmat ovat JAWS ja NVDA, joista ainakin toista käyttivät melkein 80 prosenttia vastaajista. Kuitenkin samanlaisten perustoiminnallisuuksien vuoksi verkkosivuston rakentajan ei tarvitse keskittyä eri ruudunlukuohjelmien eroihin, vaan yrittää optimoida sivusto niillä käytettäväksi mahdollisimman hyvin.

Ruudunlukuohjelmat käyttävät hyödyksi HTML:n elementtejä yrittäessään arvioida sivun rakennetta, mikä vuoksi sivusto tulee rakentaa mahdollisimman semanttisesti. Tämä tarkoittaa sitä, että sivustolla käytetään HTML:n elementtejä, jotka ovat tarkoitettu siihen mitä niillä halutaan kertoa. Esimerkiksi joissain ruudunlukuohjelmissa on mahdollisuus selata sivun otsikkoja läpi, joten käyttämällä otsikkoelementtejä (h1, h2, h3…) ohjelma tietää mitkä elementit ovat otsikoita.

Käyttämällä otsikon sijaan vaikkapa span-elementtiä suurella fonttikoolla, vain näkökyvylliset käyttäjät tietävät kyseessä olevan otsikko. Samalla tapaa ruudunlukuohjelmat tunnistavat erimerkiksi linkit, painikkeet, taulukot ja lomakkeet. Jos kuitenkin on pakottava tarve tehdä ei-semanttisesta elementistä, kuten div-elementistä, esimerkiksi painike, siitä voidaan erikseen tehdä käytettävä asettamalla sille ominaisuuksiksi tabindex=”0” sekä role=”button”. Tällöin painikkeesta tulee näppäimistöllä saavutettava, sekä ruudunlukuohjelma tietää, että kyseessä on painike.

Edellisen esimerkin painikkeen role-ominaisuus kuuluu World Wide Web Consortiumin kehittämään Web Accessibility Initiative – Accessible Rich Internet Applications -määrittelyyn (lyh. WAI-ARIA), joka lisää käyttöön HTML-ominaisuuksia, joiden tarkoituksena on auttaa parantamaan verkkosivujen käytettävyyttä erityisesti ruudunlukuohjelmien käyttäjille. WAI-ARIA:n määrittelemiä HTML-elementtien ominaisuuksia käyttämällä elementeille voidaan asettaa rooleja, ominaisuuksia sekä tiloja, jotka kertovat ruudunlukuohjelmalle ja sen käyttäjälle tietoa elementistä ja sen käytöstä. Rooleilla voidaan asettaa elementille toiminnallinen tai staattinen rooli, jonka avulla ruudunlukuohjelman käyttäjän ei tarvitse arvailla mikä on kyseisen elementin tarkoitus. Esimerkiksi tavalliselle epäsemanttiselle div-elementille voidaan määritellä role=”navigation”, jolloin ruudunlukuohjelma tietää kyseessä olevan navigaatioelementti. WAI-ARIA:n määrittelemiä HTML-elementtien ominaisuuksia ovat puolestaan esimerkiksi elementin nimike (aria-label) ja vain luku -tila tekstikentälle (aria-readonly). Vaikka se onkin mahdollista, ominaisuuksien arvo ei yleensä vaihdu, vaan sitä varten on eritelty tila-ominaisuudet. Tila-ominaisuuksia ovat esimerkiksi tekstinkentän käytöstä poistaminen (aria-disabled) sekä kahden tilan painikkeiden tilan vaihtaminen (aria-pressed).

Uusin HTML-merkintäkielen pääversio HTML5 toi mukanaan semanttisia elementtejä, joilla on tarkoitus korvata osa rooleista, joita käytettään sivun kiintopisteinä eli sivun eri osioiden jaotteluun (engl. landmark roles). Näitä rooleja ovat esimerkiksi banner, main, footer, complementary sekä navigation, jotka voidaan korvata nykyään header-, main-, footer-, aside- ja nav-elementeillä, vastaavassa järjestyksessä. Uusien HTML5-elementtien tarkoituksena on siis osittain korvata

sivuston epäsemanttisia elementtejä, sekä vähentää tarvetta role-ominaisuuden käytölle, ja siten helpottaa ruudunlukuohjelmalle selkeän sivun rakentamista. Nykyään ruudunlukuohjelmien tuki näille HTML5-elementeille on jo niin hyvä, että niitä voidaan käyttää tuotantoon menevillä verkkosivuilla ilman role-ominaisuuttakin.

Roolien lisäksi muita erikseen mainittavia WAI-ARIA:n tuomia ominaisuuksia ovat nimike ja kuvaus. Nimikkeen tai kuvauksen lisääminen elementteihin on sivuston rakentajan kannalta helppo asia, ja ne auttavat ruudunlukuohjelman käyttäjiä huomattavasti. Niillä voidaan määritellä mille tahansa elementille ihmistenkielinen nimike tai kuvaus, jonka ruudunlukuohjelma lukee käyttäjälle suoraan. Näin käyttäjän ei tarvitse arvailla esimerkiksi ”navigation” -roolin sisältävästä elementistä onko kyseessä päävalikko, murupolku, sivuvalikko vai linkkilista. Nimikkeen saa lisättyä elementille asettamalla sille ominaisuudeksi aria-label=”Nimike”. Jos nimikkeeksi halutaan joku toinen elementti sivulla, esimerkiksi seuraava otsikko, sen sisältö voidaan asettaa luettavaksi nimikkeeksi asettamalla nimettävälle elementille aria-labelledby-ominaisuuden arvoksi kohde-elementin id. Samaan tapaan elementille voidaan asettaa kuvausviittaus toiseen elementtiin asettamalla aria-describedby-ominaisuuden arvoksi halutun kohde-elementin id. Kuvaus lisätään yleensä nimikkeen rinnalle, kun käyttäjälle halutaan antaa enemmän tietoa elementin tarkoituksesta.

Ruudunlukuohjelman käyttäjiä varten on tärkeää muistaa laittaa kaikkiin kuviin vaihtoehtoinen teksti. Ruudunlukuohjelmat eivät voi selittää käyttäjälle kuvan sisältöä, joten sen sijaan käyttäjälle luetaan vaihtoehtoinen teksti. Vaihtoehtoisen tekstin voi laittaa ainoastaan img-elementille asettamalla sille alt-ominaisuuden arvoksi haluamansa tekstin. Tekstin tulisi olla lyhyt, mutta informatiivinen selitys kuvan sisällöstä, jotta käyttäjä ymmärtää lyhyessä ajassa kuvan tarkoituksen. Hyvä tapa on myös laittaa alt-tekstin perään piste, jotta ruudunlukuohjelma pitää pienen tauon tekstin lukemisen jälkeen. Kuitenkaan vain koristeelliseen käyttöön tarkoitetuille kuville ei aseteta alt-tekstiä, vaan tällöin sen arvo tulee jättää tyhjäksi seuraavasti: alt=””. Näin ruudunlukuohjelma tietää, että vaihtoehtoista tekstiä ei ole jätetty vahingossa pois, eikä se yritä paikata asiaa esimerkiksi lukemalla kuvan tiedostonimeä. Elementtien taustakuviksi asetetuille kuville ei voi laittaa alt-

tekstiä. Tässä tapauksessa koristeellisille kuville ei tarvitse tehdä mitään, mutta käyttäjän kannalta merkittäville kuville tulee asettaa vaihtoehtoinen teksti nimikkeellä (aria-label tai aria-labelledby) sekä rooliksi ”img”. Näin ruudunlukuohjelma tietää kyseessä olevan kuva ja se lukee kuvan nimikkeen käyttäjälle.

Lomakkeita tehtäessä tulee muistaa asettaa kaikille kentille näkyvä nimike, jotta käyttäjät tietävät mitä kenttään tulisi täyttää. Lomake saadaan ruudunlukuohjelmalle käytettäväksi käyttämällä kentille suoraa tai epäsuoraa nimikettä. Nimike tulee laittaa HTML:n label-elementtiin tekstisisällöksi. Suora nimike saadaan aikaan asettamalla nimike-elementtiin for-ominaisuudeksi viitattavan kentän id. Epäsuora nimike puolestaan tehdään asettamalla molemmat, nimike sekä kenttä, label-elementin sisään. Molemmilla tavoilla ruudunlukuohjelma saa selville kentän nimikkeen ja lukee sen käyttäjälle. Näin saadaan aikaan lomake, jota voi käyttää sekä näkökyvylliset että ruudunlukuohjelmaan turvautuvat käyttäjät.

Lomakkeiden kentissä käytetään paljon eri ominaisuuksia ja tiloja. Kenttä voi olla erimerkiksi käytöstä poistettu, vain luku -tilassa tai vaadittu kenttä. Yleensä nämä kerrotaan käyttäjälle visuaalisella palautteella, esimerkiksi vaaditun kentän nimikkeessä voi olla *-merkki, ja käytöstä poistettu kenttä on usein kokonaan harmaa. Ruudunlukuohjelman käyttäjä ei todennäköisesti näitä näe, joten kentille tulee asettaa myös vastaavat ominaisuudet tai tilat. WAI-ARIA -määrittelystä esimerkiksi aria-disabled, aria-readonly sekä aria-required -ominaisuudet antavat ruudunlukuohjelmalle tiedon kentän tilasta. Näissä kentän ominaisuuksissa on usein arvona true tai false, mutta joillekin ominaisuuksille on olemassa muitakin arvoja. Input-elementin ominaisuuksia tulisi siis muistaa käyttää aina, kun näytetään käyttäjälle visuaalista palautetta kentän tilasta.

Nykyään ruudunlukuohjelmien tuki HTML:n omille ominaisuuksille on parantunut, joten kentille voidaan käyttää esimerkiksi disabled, readonly ja required -ominaisuuksia ilman aria-etuliitettä. Kuitenkin WAI-ARIA:ssa on joitain ominaisuuksia, joita ei vielä ole HTML:n natiiviominaisuuksina. Tällainen on esimerkiksi aria-invalid-tila, joka kertoo käyttäjälle viallisesta kentän täytöstä. Aria-invalid-ominaisuuteen asetettu arvo kertoo käyttäjälle väärän täytön syystä, ja se voi olla false, true, spelling tai grammar.

2.3 JavaScript-esto

Verkkoselaimissa on mahdollista estää JavaScriptin ajaminen kokonaan, minkä seurauksena osa sivuston ominaisuuksista kuten mobiilivalikot, Ajax-lähetykset, ja JavaScriptillä ladatut kuvat lakkaa toimimasta. JavaScriptin estävät käyttäjät tekevät sen usein joko nopeuden tai turvallisuuden takia. Maailmanlaajuisesti vuonna 2016 arviolta noin 0,2 % kaikista sivulatauksista tapahtui JavaScript estettynä, mutta Suomessa luku nousi yhteen prosenttiin (Blockmetry 2017).

Kuitenkin huomioon tulee ottaa se, että suurimmassa osassa tapauksia käyttäjä on estänyt JavaScriptin omasta halustaan, ja tällöin hänellä olisi mahdollisuus purkaa esto. Siksi sivustoa ei mielestäni tarvitse saada toimimaan täydellisesti ilman JavaScriptiä, vaan estoa käyttäville käyttäjille pitäisi antaa mahdollisuus vähintään sivuston perusselaamiseen. Esimerkiksi JavaScriptillä ladatuille kuville pitäisi asettaa varmistus, jotta ne toimivat JavaScript estettynäkin, mutta monimutkaisille Ajax-lähetyksille ei tarvitsisi asettaa varmistusta. Poikkeuksena ovat kuitenkin sivustot, jotka rakennetaan pyydetysti toimimaan ilman JavaScriptiä. Näitä voivat olla esimerkiksi valtion sisäiset sivustot, jotka vaativat mahdollisimman hyvän turvallisuuden tiedonkulkuun.

Käytännössä JavaScript-estolle tehdään varmistus asettamalla noscript-elementin sisään sisältö, joka halutaan näkyviin vain näille käyttäjille. Noscript-elementti tunnistaa, jos JavaScriptiä ei voida ajaa, eikä sitä silloin näytetä muille käyttäjille. Esimerkiksi JavaScriptillä viiveladattu (engl. lazyloaded) kuva voidaan varmistaa asettamalla kuvan alle noscript-elementtiin kuvaelementti normaalilla src-ominaisuudella. Näin tavallisesti ladattu kuva näkyy vain JavaScriptin estäneille käyttäjille.

```
<img class="lazyload" data-src="/img/logo.png" alt="Yritys Oy" />
<noscript>
    <img src="/img/logo.png" alt="Yritys Oy" />
</noscript>
```

Kuva 6. JavaScript-eston varmistaminen kuvaelementille.

3 Verkkosivuston suorituskykyoptimointi

Verkkosivun latausnopeus vaikuttaa suuresti käyttäjäkokemukseen ja sitä kautta myös verkkosivuston käyttömäärään. Latausnopeuden optimointi vaikuttaa kaikkiin päätelaitteisiin, mutta huono suorituskykyoptimointi näkyy erityisesti mobiililaitteilla, sillä niillä verkonyhteys on usein huonompi. Googlen raportin (2016) mukaan noin puolet käyttäjistä odottavat mobiilisivun lataavan näkyväksi alle kahdessa sekunnissa, ja noin 53 prosenttia sivuvierailuista hylätään, jos mobiilisivun lataus kestää yli kolme sekuntia.

Verkkosivun suorituskykyä voidaan parantaa optimoimalla koodia asiakaspuolella tai palvelinpuolella. Tässä raportissa keskitytään asiakaspuolen suorituskykyoptimointiin, sillä usein verkkosivuston rakentaja ei pääse vaikuttamaan palvelinpuolen optimointeihin.

Suorituskykyä optimoitaessa tavoitteena on saada sivusto mahdollisimman nopeasti sellaiseen tilaan, että käyttäjä voi alkaa lukea sen sisältöä. Tähän vaikuttaa eniten ensimmäinen sivulataus, sillä silloin käyttäjän selain lataa ensikerran kaikki CSS- ja JavaScript-tiedostot, jotka selain sitten asettaa välimuistiin. Tiedostojen lataaminen välimuistista on erittäin nopeaa, sillä silloin tiedostoja ei tarvitse ladata internetyhteyden avulla. Sivuston hitaus johtuu siis yleisimmin verkkolatausten hitaudesta, eli käytännössä palvelimilta ladattavat tiedostot hidastavat sivun latautumista. Nykyään eri verkkoselaamiseen kykenevien laitteiden laitteisto on niin hyvä, ettei prosessoinnista johtuvia suorituskykyongelmia yleensä pääse syntymään.

3.1 Tiedostosta verkkosivuksi

Käyttäjän navigoidessa uudelle sivulle selain lähettää pyynnön HTML-dokumentista palvelimelle. Palvelin vastaa pyyntöön oikealla HTML-tiedostolla,

jos sellainen löytyy. Tämän jälkeen alkaa HTML-tiedoston sekä siinä olevien tiedostoviittauksen lataaminen ja parsiminen selaimessa. Suorituskykyä ajatellen erityisen tärkeää on se, missä järjestyksessä selain päättää ladata ja parsia HTML-tiedostossa olevat tiedostokutsut. Näitä ovat kaikki kutsut ulkoisiin tiedostoihin, kuten kutsut CSS- ja JavaScript-tiedostoihin sekä kaikki kuvat. Suorituskyvyn kannalta olisi paras, jos selain lataisi ensimmäisenä ainoastaan tiedostot, joita se tarvitsee renderöidäkseen dokumentin näkyvän sisällön, ja kaikki muut lisäkkeet ladattaisiin jälkeenpäin. Tällöin sivun sisältö saataisiin nopeasti näkyviin, ja käyttäjä pääsisi selaamaan sivua samalla kun muut lisäkkeet ladattaisiin taustalla. Tätä, ennen sivun renderöimistä tapahtuvaa tiedostojen lataus- ja parsimisjärjestystä, kutsutaan englanniksi termillä critical rendering path.

Tavoitteena on siis saada kriittisten lisäkkeiden määrä ja tiedostokoko mahdollisimman pieneksi, jotta käyttäjän kannalta tärkein eli sivun sisällön renderöintiaika saadaan nopeaksi. Jotta ymmärrettäisiin paremmin kriittisten lisäkkeiden latausjärjestyksen priorisoinnin antamaa suorituskykyetua, on ensin hyvä tutkia, miten selain lataa ja renderöi sivun tavallisesti. Oletetaan, että sivulla on yksi CSS-tiedostokutsu ja yksi JavaScript-tiedostokutsu head-osiossa, sekä body-osiossa kuvia ja tekstiä. Ensimmäisenä ladataan HTML-tiedosto ja sen parsiminen aloitetaan. Head-osiosta selain tunnistaa, että CSS- ja JavaScript-tiedostoja tarvitaan. Tässä vaiheessa selain aloittaa CSS-tiedoston, JavaScript-tiedoston sekä kuvien lataamisen palvelimelta. Mitään ei vielä renderöidä näytettäväksi, sillä muuten selain joutuisi tekemään elementtien sijainnin laskemisen ja niiden piirtämisen uudelleen, koska tiedostot todennäköisesti sisältävät joitain tyylimuutoksia HTML-dokumentin sisältöön.

CSS-tiedoston ladattua se käydään läpi ja varmistetaan, että sieltä ei kutsuta mitään muuta ulkoista lisäkettä. CSS-tiedostossa olevista tyyleistä selain rakentaa CSSOM-rakenteen. CSSOM:ia (CSS Object Model) voi ajatella kuin karttana sivulla olevien elementtien CSS-tyyleistä. Renderöidäkseen sivun selain tarvitsee sekä CSSOM:in että DOM:in. DOM on puolestaan HTML-dokumentin elementeistä tehty kartta, joka luodaan parsimalla HTML-tiedoston sisältöä. CSS-tiedoston parsimisen jälkeenkään sivun renderöintiä ei voida vielä aloittaa, sillä JavaScript-tiedostoissa voi olla kutsuja CSSOM:iin. Myös dokumentin parsiminen

keskeytetään, sillä JavaScriptissä voi olla DOM:in muutoksia vaativia kutsuja. Näin ollen JavaScript-tiedoston lataus odotetaan ennen kuin parsimista voidaan jatkaa. Tiedoston ladattua se luetaan sekä tarkistetaan, ettei siinäkään ole kutsuja ulkoisiin lisäkkeisiin. Vasta tämän jälkeen selain voi jatkaa dokumentin parsimista, jonka jälkeen aloitetaan käyttäjälle näkyvän sisällön renderöinti.

Tavallisesti selain siis odottaa kaikki head-osiossa olevat JavaScript- ja CSS-tiedostot ladatuiksi ja parsituiksi ennen kuin mitään näkyvää voidaan renderöidä sivulle. Tämän vuoksi sivun latausnopeutta saadaan parannettua huomattavasti, kun siirretään kaikki ei-kriittinen CSS ja JavaScript ladattavaksi vasta sitten kun sisällön renderöinti on aloitettu. Kuvien osalta selain ei odota kaikkia ladatuiksi ennen kuin se aloittaa sivun renderöinnin, mutta ylimääräiset kuvien lataukset kasvattavat ladattavan sisällön kokoa sekä myöhästyttävät koko sivun latauksen valmistumista.

3.2 CSS-tyylitiedostojen optimoiminen

Sivun latausnopeus saataisiin erittäin nopeaksi siirtämällä aivan kaikki tiedostot ladattavaksi DOM:in rakennuksen ja renderöinnin aloittamisen jälkeen. Kuitenkaan aivan kaikkia CSS-tyylejä ei voida siirtää ladattavaksi jälkeenpäin. Tällöin selain piirtäisi ensin tyylittömän version verkkosivusta, ja heti kun CSS-tiedostot on ladattu ja CSSOM on rakennettu, selain piirtäisi sivun uudestaan CSS-tyylien kanssa, jolloin käyttäjän lukema sivun sisältö hyppäisi eri paikkaan. Tällaista sisällön hyppimistä tulisi välttää hyvän käyttäjäkokemuksen antamisen vuoksi.

Erityisen nopea ja käyttäjäystävällinen sivunlataus saadaan aikaan asettamalla selain lataamaan tyylit ensin vain elementeille, jotka näkyvät käyttäjälle ensimmäisenä ennen kuin sivua rullataan alaspäin (engl. above the fold content). Tällöin selaimella on vähemmän ladattavaa ja se pääsee renderöimään sivun sisältöä nopeammin, ja silti käyttäjälle ensimmäisenä näkyvällä sisällöllä on oikeat tyylit. Tätä varten sivuston rakentajan tulisi eritellä ensimmäisenä näkyvien elementtien tyylit erikseen head-osioon joko suoraan style-elementin sisälle (engl. inline CSS) tai erilliseen CSS-tiedostoon. Loput sivun tyylit asetetaan tavallisesti

CSS-tiedostoon, mutta sitä ei aseteta head-osioon. Sen sijaan juuri ennen body-osion loppumista asetetaan script-elementti, jolla lisätään uusi CSS-linkki head-osioon JavaScriptin avulla. Näin kaikki loput tyylit saadaan latautumaan vasta sivun renderöinnin jälkeen, ja siten varmistetaan nopea ensimmäinen piirto sivulle (engl. first paint), jolloin käyttäjä pääse aloittamaan sivun selaamisen mahdollisen nopeasti. JavaScript-estoa varten head-osioon tulee kuitenkin muistaa asettaa noscript-varmistus, jonka sisällä on tavallinen linkki CSS-tiedostoon.

Seuraavaksi on hyvä varmistaa, että CSS-tyylit ulkoisista tiedostoviittauksista ladataan mahdollisimman nopeasti. Tätä varten kaikki oma ei-kriittinen CSS kannattaa laittaa samaan tiedostoon, sillä tiedoston latauksen ja parsimisen lisäksi kutsu palvelimelle vie selaimelta hieman aikaa. On siis nopeampaa ladata ja parsia yksi 10 kilobitin tiedosto kuin viisi kahden kilobitin tiedostoa. Tämä yksi yhdistetty CSS-tiedosto kannattaa myös minifoida, joka tarkoittaa kaikkien ylimääräisten välilyöntien ja rivivälien poistamista sekä mahdollisten muuttujien nimeämistä vähemmän tilaa vieviksi. Minifoidessa tiedosto muuttuu ihmisille vaikealukuiseksi, joten siksi kannattaa käyttää ulkoisia minifointi-työkaluja. Monille ohjelmointiin tarkoitetuille tekstieditoreille on olemassa ladattavia paketteja, jotka luovat uuden minifoidun tiedoston automaattisesti tallentaessa alkuperäistä tiedostoa. Tällöin alkuperäinen CSS-tiedosto säilyy lukukelpoisena, ja samalla sivustolle saadaan julkikäyttöön erillinen minifoitu versio.

Tiedostojen kokoa voidaan pienentää myös välttämällä turhia selektoriketjuja CSS:ssä. Esimerkiksi monen luokan sisennykset käyttäessä Sass tai Less -esikääntökieliä voivat lisätä lopullisen CSS-tiedoston kokoa huomattavasti. Myös valmiiden teemojen tai komponenttikirjastojen käytön välttäminen vähentää CSS:n kokoa, sillä usein niissä tulee mukana tyylejä, joita ei lopulta käytetä ollenkaan sivustolla.

Kuitenkin joskus on helpompaa ja nopeampaa käyttää valmiita tyylejä, esimerkiksi ikoneille on olemassa paljon ladattavia tyylikirjastoja. Kirjastojen tiedostot voidaan saada verkkosivulle ladattavaksi joko samalta palvelimelta, jossa HTML-tiedostokin sijaitsee, tai sitten ulkoiselta palvelimelta (engl. Content delivery network, lyh. CDN). Yleisesti käytettyjen tyylikirjastojen tiedostot kannattaa laittaa

ladattavaksi ulkoiselta palvelimelta, sillä käyttäjällä voi olla ne jo tallennettuna selaimen välimuistiin. Näin käy, jos aiemmin vieraillulla sivulla on käytetty samalta palvelimelta ladattua tiedostoa, ja siten tiedosto haetaan välimuistista verkkolatauksen sijaan. Samalla tapaa oman palvelimen tiedostot ladataan vain ensimmäisen kerran käyttäjälle, ja sen jälkeen ne tallennetaan välimuistiin. Tämän vuoksi ensimmäisen sivulatauksen jälkeen sivut, joissa ei ole uusia tiedostokutsuja ovat nopeita ladata käyttäjälle.

3.3 JavaScript-tiedostojen optimoiminen

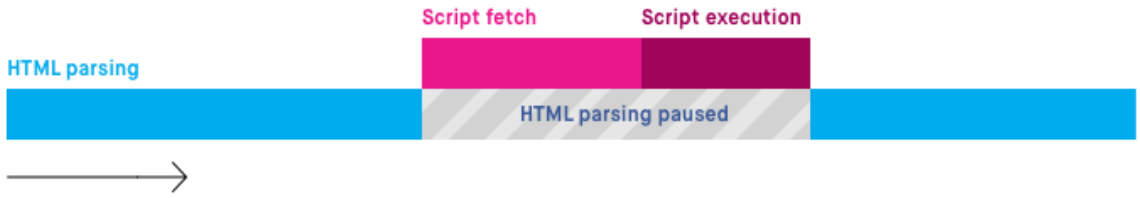
JavaScript-tiedostojen lataamisen optimoiminen on monelta tapaa samankaltainen toimenpide kuin CSS-tiedostoille. JavaScript-tiedostojen määrä tulee pitää mahdollisimman pienenä, jolloin selain tekee vähemmän kutsuja palvelimelle ja sivun lataaminen nopeutuu. Tämän lisäksi yleisille kirjastoille kannattaa suosia ulkoisia palvelimia, sekä kaikki tiedostot tulee minifoida.

Minifoinnilla saavutettu etu voi olla JavaScript-tiedostoissa jopa suurempi kuin CSS-tiedostoissa. JavaScript sisältää usein paljon muuttujia, joiden nimet voidaan minifoitaessa lyhentää jopa yhteen tai kahteen merkkiin. Tätä ei voida tehdä CSS:lle, sillä siinä selektoreiden täytyy olla samat kuin HTML-elementeissä olevat vastineet.

Lisäksi JavaScript-tiedostokutsuille on olemassa ominaisuus nimeltään defer. Asettamalla script-elementille defer-ominaisuuden, selain lataa src-ominaisuudessa olevan tiedoston asynkronisesti, sekä lukee ja suorittaa tiedoston vasta HTML-dokumentin parsimisen jälkeen. Defer-ominaisuudelle ei tarvitse asettaa mitään arvoa sen toimiakseen. Deferin avulla JavaScript-tiedostotkutsut eivät keskeytä sivun parsimista, eli sivun renderöinti voidaan aloittaa nopeammin. Nykyään deferin selaintuki on erittäin hyvä, jopa 95 prosenttia (Can I Use…, 2018), joten sitä kannattaa käyttää kaikkiin JavaScript-tiedostoihin, jotka voidaan suorittaa sivun parsimisen jälkeen. Defer-ominaisuudella varustetut tiedostot suoritetaan samassa järjestyksessä kuin missä ne on sivulle asetettu, joten virheitä ei pääse syntymään väärän suorittamisjärjestyksen takia esimerkiksi jQueryn

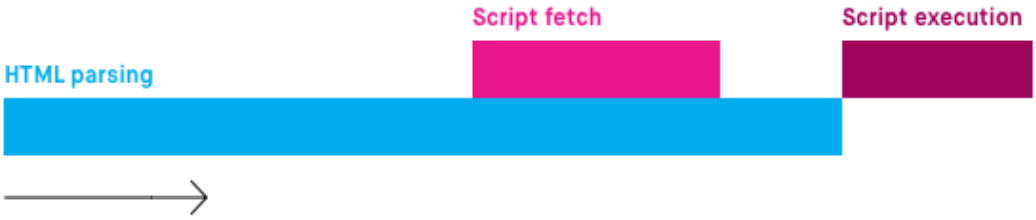
kanssa. Poikkeuksena on vanhat Internet Explorer -versiot (IE9 ja alaspäin), joissa on ongelmia tiedostojen suoritusjärjestyksen kanssa deferiä käyttäessä. Jos näitä selaimia halutaan tukea, deferiä ei tulisi käyttää sivustolla.

Defer-ominaisuuden lisäksi script-elementillä on olemassa async-ominaisuus. Async aiheuttaa sen, että src-ominaisuudessa oleva tiedosto ladataan asynkronisesti, mutta suoritetaan heti latauksen valmistuttua. Tällöin HTML-dokumentin parsiminen ei keskeydy latauksen ajaksi, mutta JavaScriptin suoritus keskeyttää sen. Tämän vuoksi defer-ominaisuutta tulisi käyttää async:in sijaan. Asettamalla script-elementille molemmat ominaisuudet, selain käyttää ensisijaisesti async-ominaisuutta, mutta vanhemmat async-ominaisuutta tukemattomat selaimet turvautuvat deferiin. Deferin tavoin async-ominaisuus ei tarvitse arvoa toimiakseen, eikä kumpaakaan ominaisuutta tulisi käyttää script-elementteihin, joissa ei ole src-ominaisuutta.



Kuva 7. HTML-tiedoston parsiminen JavaScript-tiedoston haun yhteydessä ilman async tai defer -ominaisuuksia.

Kuva 8. HTML-tiedoston parsiminen JavaScript-tiedoston haun yhteydessä käyttäessä async-ominaisuutta.



Kuva 9. HTML-tiedoston parsiminen JavaScript-tiedoston haun yhteydessä käyttäessä defer-ominaisuutta.

3.4 Kuvien optimoiminen

Kuvien lataukset eivät ole HTML-tiedoston parsimista tai sivun renderöintiä estäviä tapahtumia, mutta kuitenkin ne hidastavat sivun täyden lataamisen valmistumista, sekä kasvattavat huomattavasti ladattavan sisällön määrää käyttäjälle. Keskimäärin jopa yli puolet sivun ladattavasta määrästä johtuu sivulla olevista kuvista (HTTP Archive 2018).

Optimoitaessa kuvia niiden tiedostokoko pyritään saamaan mahdollisimman pieneksi, kuitenkin säilyttäen ihmissilmälle erottuva kuvanlaatu hyvänä. Ensimmäinen optimointivaihe kuville on käyttää sivustolla vain oikeankokoisia kuvia. Tämä tarkoittaa sitä, että jos kuvaelementin koko on sivulla 400 pikseliä leveä ja 300 pikseliä korkea, niin siihen ladattu kuva olisi täsmälleen samankokoinen. Tällöin kuva on kevyempi ladata käyttäjälle sekä se näkyy tarkkana, koska selaimen ei tarvitse skaalata kuvaa yhtään. Poikkeuksena tähän ovat erittäin pienet, muut kuin vektorigrafiikkaa käyttävät, ikonit, jotka voidaan ladata tuplakokona kuvatarkkuuden säilyttämiseksi.

Pienentämällä ladattavan kuvan kokoa ja jättämällä kuvaelementiltä leveyden sekä korkeuden määrittelemättä, kuva olisi aina oikeankokoinen ja näkyisi tarkkana sivulla. Tähän kuitenkin liittyy käyttäjän kannalta käytettävyysongelma nimeltään sisällön hyppiminen. Tämä tarkoittaa sitä, että käyttäjän lukema teksti hyppää eri paikkaan sivulla, kun tekstin yläpuolella olevat kuvat renderöidään sivulle. Tämä on ongelma erityisesti mobiililaitteilla, joilla näytönleveys on pieni ja

nettiyhteys hitaampi. Korjatakseen ongelman sivuston rakentajan tulisi asettaa kuvapaikoille korkeus varatuksi aina kun mahdollista, jolloin kuvan tilalla on valkeaa taustaa ennen kuvan renderöintiä. Sisällön hyppiminen ei ole ongelma taustakuvaelementtien kanssa, sillä niiden kokoa määrittelee joko sisällön koko tai elementin korkeus- ja leveysmäärittelyt.

Oikean kuvakoon asettamisen jälkeen kannattaa pienentää kuvan tiedostokokoa kuvan tietoja ja laatua muokkaamalla. Siihen on monta työkalua, joista yleinen on esimerkiksi Adobe Photoshopin ”Vie muodossa” -valinta, jolla voi valita kuvan vientiasetukset, joita muuttamalla näkee suoraan vietävän kuvan tiedostokoon. Vaihdettavia vientiasetuksia on esimerkiksi kuvaformaatin vaihto, kuvanlaatuasetukset, läpinäkyvyys- ja väriasetukset sekä metadatan karsiminen. Tiedostokoon karsimiseksi esimerkiksi taustakuvaksi tulevasta kuvasta kannattaa vaihtaa kuvaformaatti JPG-muotoon, pienentää kuvanlaatua silmämääräisesti sekä poistaa ylimääräinen metadata.

Google on kehitellyt uudehkon vähemmän tilaa vievän kuvaformaatin nimeltään WebP. WebP tarjoaa vaihtoehdoiksi molemmat häviöttömän ja häviöllisen pakkausmetodin. Häviöttömällä pakkaamisella tiedostokoossa säästää keskimäärin 26% PNG-formaattiin verrattuna, ja häviöllisellä pakkaamisella säästetään 25-34% JPEG-formaattiin verrattuna (Google Developers 2016). Kuvan voi vaihtaa WebP-formaattiin lataamalla WebP Converter -ohjelman Google Developers -sivustolta.

WebP on sen verran uusi formaatti, että tällä hetkellä lähinnä vain Google Chrome -pohjaiset selaimet ja Opera tukee sitä (Can I Use…, 2018), minkä vuoksi sille pitää asettaa varmistus vanhempia selaimia varten. Varmistus voidaan tehdä asettamalla HTML5:n mukana tulleeseen picture-elementtiin kaksi source-elementtiä, joilla on srcset-ominaisuudessa kuvan tiedostopolku sekä type-ominaisuudessa tiedoston tyyppi. Ensimmäisessä source-elementissä on WebP-formaatin kuva ja toisessa varmistus, esimerkiksi JPG-muotoinen kuva. Tällöin selain tunnistaa type-ominaisuudesta voiko se käyttää ensimmäistä

source-elementtiä (WebP). Jos ensimmäistä tyyppiä ei tueta, siirrytään seuraavaan. Tämän lisäksi picture-elementtiin asetetaan viimeiseksi tavallinen img-elementti, jolloin selain käyttää sitä, jos se ei tue ollenkaan picture-elementtiä.

```
<picture>
    <source srcset="/kansio/kuva.webp" type="image/webp">
    <source srcset="/kansio/kuva.jpg" type="image/jpeg">
    <img src="/kansio/kuva.jpg" alt="Vaihtoehtoinen teksti">
</picture>
```

Kuva 10. WebP-kuvan käyttö picture-elementissä varmistusten kanssa.

Normaalisti verkkosivuille asetetaan kuvakoot suurien pöytätietokoneiden näyttökokojen mukaan, jolloin mobiililaitteelle ladataan liian isoja kuvia. On täysin turhaa ladata esimerkiksi 1920 pikseliä leveä kuva 800 pikseliä leveän näytön omaavalle laitteelle ja sitten skaalata sen kokoa alaspäin. Tätä varten img- ja source-elementille on kehitetty srcset-ominaisuus, jonka avulla selain laskee automaattisesti oikeankokoisen kuvan ladattavaksi. Srcset-ominaisuuteen voidaan asettaa pilkulla eroteltuna erikokoisten kuvien tiedostopolut sekä niiden oikeat leveydet. Tällöin selain voi laskea näytönleveyteen ja näytöntiheyteen nähden sopivan kuvan ladattavaksi. Esimerkiksi 400 pikseliä leveä kuva ladataan vain alle 400 pikseliä leveillä näytöillä. Suuremmilla näytöillä ladataan seuraavaksi pienin määritelty kuva ja niin edelleen. Näin säästetään paljon kuvien latausaikaa, kun pienempien ruutujen käyttäjille ei tarvitse ladata täysikokoista kuvaa.

```
<picture>
    <source srcset="kuva450px.webp 450w, kuva900px.webp 900w, kuva1920px.webp 1920w" type="image/webp">
    <source srcset="kuva450px.png 450w, kuva900px.png 900w, kuva1920px.png 1920w" type="image/png">
    <img
        srcset="kuva450px.png 450w, kuva900px.png 900w, kuva1920px.png 1920w"
        src="kuva1920px.png"
        alt="Vaihtoehtoinen teksti">
</picture>
```

Kuva 11. Vaihtoehtoisten kuvakokojen käyttö picture-elementin kanssa.

Aina kuvapaikat eivät ole joka näytönleveydellä saman kokoisia, jolloin pelkkä srcset-määrittely ei toimi. Selain olettaa, ettei kuvapaikalla ole maksimileveyttä, jonka vuoksi se laskee sopivan kuvan koon koko näytön leveyden avulla. Tätä

varten source- ja img-elementeille voidaan asettaa sizes-ominaisuus. Sizes-ominaisuuteen listataan kuvapaikan koot eri näytönleveyksillä mediasääntöjen avulla. Esimerkiksi jos kuvapaikka on alle 800 pikseliä leveillä näytöillä kokoleveä, ja muuten 500 pikseliä leveä, se asetettaisiin seuraavasti: sizes=”(max-width: 799px) 100vw, 500px”. Tällöin selain osaa laskea sopivan kuvan ladattavaksi responsiivisille kuvapaikoille. Sizes-ominaisuuden ilman mediasääntöjä määriteltyyn arvoon turvaudutaan silloin, kun mikään aiempi mediasäännöllinen määritelmä ei vastaa nykyistä tilannetta.

```
<picture>
    <source
        srcset="kuva450px.webp 450w, kuva900px.webp 900w"
        sizes="(max-width: 799px) 100vw, 500px"
        type="image/webp">
    <source
        srcset="kuva450px.png 450w, kuva900px.png 900w"
        sizes="(max-width: 799px) 100vw, 500px"
        type="image/png">
    <img
        srcset="kuva450px.png 450w, kuva900px.png 900w"
        sizes="(max-width: 799px) 100vw, 500px"
        src="kuva900px.png"
        alt="Vaihtoehtoinen teksti">
</picture>
```

Kuva 12. Vaihtoehtoisten kuvakokojen käyttö responsiivisen kuvapaikan kanssa picture-elementissä.

Verkkosivun latauksen valmistumisen kannalta yksi tärkeimmistä optimoinneista kuville on laittaa ne lataamaan viiveellä (engl. lazyload). Viivelataus saadaan aikaiseksi JavaScriptillä, ja siihen on olemassa monia julkisia kirjastoja, joilla saadaan viivelataus sekä kuvaelementeille että taustakuville. Viivelataus toimii käytännössä siten, että kuvan tai taustakuvan kuvalähde asetetaan HTML-koodin sijaan JavaScriptillä ajettavassa tiedostossa sivun renderöinnin jälkeen.

Monissa valmiissa viivelatauskirjastoissa kuville tulee asettaa jokin luokka, jolla JavaScript voi tunnistaa viiveellä ladattavat kuvat. Tämän lisäksi kuville tai taustakuvallisille kuvapaikoille asetetaan usein kuvan tiedostopolku src-ominaisuuden sijaan data-src-ominaisuudeksi, jolloin selain ei aloita kuvien lataamista heti.

Monien eri kuvakokojen tapauksessa srcset korvataan data-srcset-ominaisuudella. Koska viivelataus vaatii JavaScriptiä toimiakseen, kuville tulee muistaa asettaa noscript-varmistus asettamalla noscript-elementin sisään kuva tavallisesti src-ominaisuuden kanssa.

```
<img class="lazyload" data-src="kuva.png" alt="Vaihtoehtoinen teksti">
<noscript>
    <img src="kuva.png" alt="Vaihtoehtoinen teksti">
</noscript>
```

Kuva 13. Esimerkki kuvan viivelataamisen HTML-syntaksista.

3.5 Ennenaikainen yhdistäminen palvelimelle

Käyttäessä ulkoisten palvelimien lähteitä lisäkkeiden lataamiseen, selain joutuu muodostamaan yhteyden kyseiseen palvelimeen aina kun sinne viitataan ensikerran. Yhteyden muodostamiseen kuuluu palvelimen DNS-haku, palvelinten välinen TCP-yhdistäminen sekä mahdollinen TLS-neuvottelu. Tavallisesti selain aloittaa palvelimelle yhdistämisen vasta juuri ennen tiedoston hakua, jolloin se hidastaa tiedoston lataamisen aloittamista. Tämä viive voidaan poistaa käyttämällä ennenaikaista yhdistämistä palvelimelle (engl. preconnect).

Ennenaikainen palvelimelle yhdistäminen toimii luomalla head-osioon linkkielementin, jonka rel-ominaisuuteen asetetaan arvoksi preconnect- ja href-ominaisuuteen kohdepalvelimen juuriosoite. Jos kyseessä on ulkoinen palvelin, elementille asetetaan myös crossorigin-ominaisuus, sillä ilman sitä selain tekee palvelimelle ainoastaan DNS-haun. Crossorigin-ominaisuus ei tarvitse arvoa toimiakseen. Preconnect-linkkielementti tulee asettaa head-osioon ennen mitään tiedostoviittausta. Ennenaikainen palvelimelle yhdistäminen kannattaa tehdä jokaiselle ulkoiselle palvelinviittaukselle, jota sivulla tarvitaan, sillä tällöin selain ei joudu odottamaan yhtäkään tiedostolatausta palvelimille yhdistämisien vuoksi. Tällä hetkellä ennenaikainen palvelimelle yhdistäminen toimii selainten käyttöön suhteutettuna noin 66 prosentilla käyttäjistä (Can I Use…, 2018), eikä aiheuta varmistustoimenpiteitä sitä tukemattomille selaimille.

```
<link rel="preconnect" href="https://fonts.googleapis.com" crossorigin>
```

Kuva 14. Ennenaikaisen palvelimelle yhdistämisen käyttö.

4 Sivustopohja

4.1 Alusta ja käyttöönotto

Esimerkkisivustoksi loin sivustopohjan, joka on käytännössä valmis sivusto, mutta sisältäen vain esimerkkitekstiä ja -kuvia. Asiakkaan tehtäväksi sivuston käyttöönotossa jää vain oikean sisällön syöttäminen sivustolle. Sivustopohja toteutettiin Koodiviidakon omalla Sivuviidakko-nimisellä sisällönhallintajärjestelmällä, sillä se mahdollistaa asiakkaalle helpon sisällön vaihtamisen puuttumatta sivuston koodiin. Sivuviidakon ylläpitopuolella asiakas voi muun muassa vaihtaa sivujen kuvat ja tekstit, luoda tai poistaa sivuja sekä järjestellä tai piilottaa sisältöä.

Sivustopohjan käyttöönotto oikeaksi sivustolle vaatii Koodiviidako Oy:lta sivustopohjan asennuksen kopioimisen, verkkotunnuksen liittämisen sivustoasennukseen sekä SSL-sertifikaatin asennuksen. Tämän lisäksi SCSS-tyylitiedostossa oleviin muuttujiin asetetaan asiakkaan haluamat asetukset, esimerkiksi yrityksen teeman värit, jolloin niitä käytetään sivustolla automaattisesti. Näiden toimenpiteiden jälkeen asiakkaalle jää tehtäväksi vain sisällönsyöttö, joka onnistuu helposti Sivuviidakon ylläpitopuolella. Sivustopohja rakennettiin kuitenkin siten, että siihen on helppo vaihtaa tai rakentaa päälle uusia elementtejä asiakkaan niin halutessa.

4.2 Välineet ja käytetyt kirjastot

Sivustoa rakentaessa käytin Sass-esikääntökieltä, jonka avulla saadaan muuttujia CSS:ään, jotka toimivat myös vanhemmilla selaimilla. Sass:n avulla pystyin

tekemään yhden SCSS-tyylitiedoston, johon voidaan asettaa yleiset muuttujat sivulla. Tällaisia ovat esimerkiksi teeman värit, teksti-, taulukko- ja linkkiasetukset, navigaation asetukset sekä ylä- ja alatunnisteen tyylit. Iso osa näistä muuttujista, kuten värit ja fonttityylit, vaikuttavat kerralla erittäin moneen elementtiin sivustolla, jolloin niitä vaihtaessa tarvitsee vaihtaa vain yhtä muuttujaa sen sijaan että kävisi kaikki tyylitiedostot läpi käsin. Sivuston muuttujat -tiedosto on esitelty kokonaan liitteessä 1.

Sivustolla käytetään myös Bootstrap -kirjastoa (versio 4.0.0-beta), joka mahdollistaa yhtenäisen rakenteen ja tekee jatkokehittämisestä helpompaa, sillä Bootstrap on yleisessä käytössä Koodiviidakko Oy:lla. Jatkokehittämistä sivustolle voivat olla esimerkiksi sivustopohjan laajentaminen tai kustomoitujen elementtien tai tyylien tekeminen yksittäisille asiakkaille. Bootstrap 4 käyttää perustyyleissään atomisia CSS-tyyliluokkia, mikä tarkoittaa sitä, että luokat nimetään niiden antamien CSS-tyylimäärittelyjen mukaan. Näin perustyylejä sisältävät luokat ovat monikäyttöisiä sekä vältetään samojen tyylimäärittelyjen uudelleenkirjoittaminen elementtien välisten pienien muutoksien vuoksi. Sivustolle ei kuitenkaan tuoda koko Bootstrap:in CSS-kirjastoa, vaan ainoastaan ne osat, joita sivustolla käytetään. Tämä vähentää lopullisen CSS-tiedoston kokoa, mikä tarkoittaa nopeampaa sivulatausta käyttäjälle.

Sivustopohjan asennus sisältää myös valmiin bootstrap-variables -nimisen SCSS-tiedoston, joka sisältää tarkempia tapoja kustomoida sivuston ja sen komponenttien tyylejä muuttujakohtaisesti. Aiemmin mainitun sivuston muuttujat -tiedoston on tarkoitus olla helposti luettava yleisimmin muutettavien tyylien lähde, ja bootstrap-variables -tiedosto sisältää puolestaan yksityiskohtaisempia kustomointimahdollisuuksia sivuston tyyleihin.

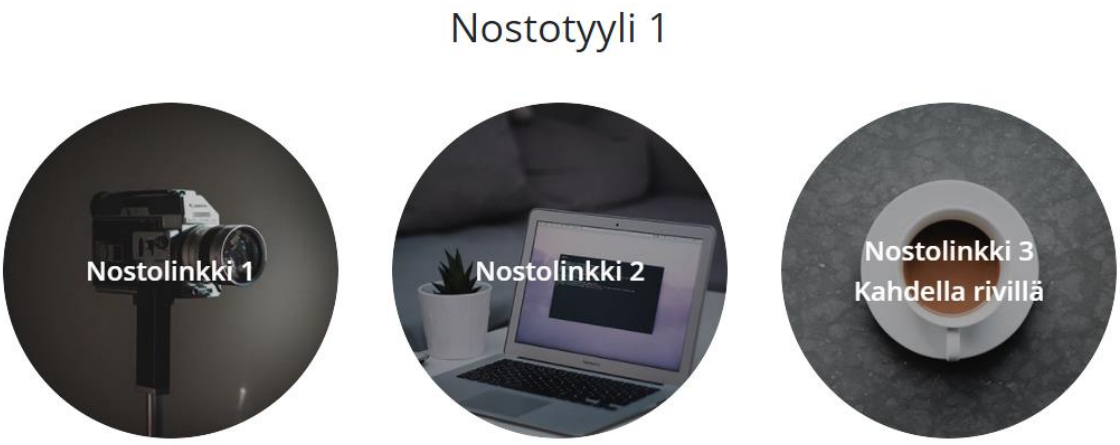
Sivustolla käytetään suosittua JavaScript-kirjastoa nimeltään jQuery (versio 3.2.1), joka helpottaa DOM:in manipulaatiota JavaScript-koodin avulla. Sivustolle tuodaan myös Hyphenator.js -kirjasto tavuttamaan automaattisesti näytön kokoon nähden liian pitkät sanat, sekä jQuery.Lazy -kirjasto kuvien viivelataamista varten.

4.3 Sivuston oletusrakenne ja elementit

Sivuston sivurakenne on tehty ajatellen pienyrityksen tarpeita. Usein pienyritysten sivustot sisältävät melko vähän sisältöä, jolloin valikkorakenteesta saadaan käyttäjälle selkeä. Kuitenkin suuremmankin sisältömäärän sivustot voidaan tehdä sivustopohjan päälle, sillä asiakas voi itse lisätä sivuja ja alasivuja, jolloin ne näkyvät automaattisesti valikoissa. Oletuksena sivustolla on päätason sivuina etusivu, yritys, palvelut sekä yhteydenotto -sivu. Yhteydenotto-sivun alta löytyy yhteystiedot- sekä ota yhteyttä -sivu.

On hyvä muistaa, että Sivuviidakolla asiakas voi luoda, siirrellä ja poistaa sivun kaikkia elementtejä, joten sivujen oletusarvoiset sisällöt ovat täysin korvattavissa. Sivujen yhteisiä elementtejä sivuilla ovat ylätunniste, navigaatio sekä alatunniste. Sivulle saadaan myös halutessa kahden palstan rakenne, jossa on sivuvalikko nykyisen ylätason sivun sisältämistä alasivuista.

Kaikki sivujen elementit ovat tehty mahdollisimman kustomoitavaksi asiakkaalle. Esimerkiksi tekstieditori-, nostolinkit- ja logorivielementteihin sekä kaikkiin lomakkeisiin voi määritellä haluamansa taustavärin ja tekstin värin, sekä useassa elementissä niiden kokoa voidaan vaihtaa. Nostolinkkityylejä on vakiona neljä erilaista, minkä lisäksi asiakas voi tarkemmin määritellä yksittäisiin nostolinkkeihin eri kuvan, kuvan tummennuksen ja kohdistuksen, linkin kohteen ja aukaisumenetelmän sekä tekstinvärin. Tuotelistauskorteille on useita eri tyylejä, kuten pysty- ja vaakasuunnassa olevat tuotekortit sekä kuvattomat ja kuvalliset listaukset. Tekstieditorina on käytössä wysiwyg (What you see is what you get) -tyylinen editori, johon asiakas voi asettaa kuvia, tekstiä eri tyyleillä sekä taulukoita. Henkilöstölistauksessa listataan Sivuviidakkoon tuotuja kontakteja, joille voi asettaa henkilö- ja yhteystietoja näkyville. Henkilöstölistauksen henkilökuville voi valita asetuksista eri kuvasuhteita sekä kuvien pyöristyksen.



Kuva 15. Yksi nostolinkkityyleistä esimerkkikuvilla ja -teksteillä.

Oletuksena sivuston etusivu sisältää navigaation, ylä- ja alatunnisteen, bannerikuvan, paikan yritysesittelytekstille, nostolinkit-elementin, tarjouspyyntölomakkeen ja logorivielementin. Tarkoituksena on antaa heti etusivulla käyttäjälle idea siitä, mitä yritys tekee, sekä näyttää palvelut ja kuvat niistä nostolinkkien muodossa. Tämän lisäksi yrityksen luotettavuutta korostetaan listaamalla yhteistyökumppanien logoja. Käyttäjän kiinnostuksen herättyä tarjouspyyntölomake on helposti saatavilla sivun loppupuolella.

Yritys-sivulla sijaitsee pidempi esittely yrityksestä, ja sinne kannattaa linkata etusivun lyhyestä yritysesittelystä. Tällöin kiinnostunut lukija voi klikata helposti lukemaan enemmän tietoa yrityksestä. Yritys-sivulle voi myös asettaa muutaman yhteistyökumppanin logon korostamaan luotettavuutta.

Palvelut-sivun rakenne riippuu täysin yrityksen palvelutarjonnasta, mutta esimerkiksi parturikampaajalla se voi olla yksi sivu, jossa kerrotaan palveluista ja näytetään hinnasto taulukossa. Auton osien jälleenmyyjällä puolestaan sivurakenne voi olla monimutkaisempi. Tällöin palvelut-sivun alasivut voidaan jakaa esimerkiksi automerkkien tai osien kategorioiden mukaan, jolloin käyttäjä löytää hakemansa varaosan nopeammin.

Yhteydenotto -sivu jakautuu yhteystiedot ja ota yhteyttä -sivuihin. Tarvittaessa sivut voidaan myös yhdistää, jos esimerkiksi yhteystietosivu jäisi muuten vajaan näköiseksi. Vakiona yhteystiedot -sivulla on yrityksen paikka- ja yhteystiedot sekä

henkilöstölistaus, jossa on yksittäisten henkilöiden kuvat ja yhteystiedot. Ota yhteyttä -sivulla on lomake, jonka asiakas voi muokata tarpeidensa mukaiseksi. Lomakkeeseen voi valita pakolliset kentät, kenttätyypit sekä kiitosviestin. Erilaisia kenttätyyppejä ovat esimerkiksi tekstialue, sähköposti, puhelinnumero, valintanapit sekä tiedoston lähetys.

Kuva 16. Yksi henkilöstölistaustyyleistä esimerkkihenkilötiedoilla ja oletuskuvilla.

4.4 Käytettävyys huomioituna

4.4.1 Yleiset elementit ja laitetuki

Sivujen rakenne on tehty käyttäjälle mahdollisimman loogiseksi ja helppokäyttöiseksi. Jokainen sivu sisältää kaikille tutut sivun osiot eli ylätunnisteen, pääsisältöosion sekä alatunnisteen. Ylätunnisteesta löytää sivun logon sekä kaikki sivuston sivut sisältävän navigaation. Alatunnisteessa on usein yrityksen yhteystietoja ja somelinkkejä. Tämän lisäksi sivulle saa asetettua sivuvalikon näkyviin, ja murupolku tehdään automaattisesti kaikille juuritasoa alemmille sivuille. Sivun osioissa käytetään semanttisia HTML-elementtejä, jotka auttavat myös ruudunlukuohjelman käyttäjiä navigoimaan sivulla. Ylätunniste on tehty header-elementillä, sivun pääosio main-elementillä ja alatunniste on footer-elementti. Lisäksi kaksipalstaisilla sivuilla pienempään palstaan käytetään aside-elementtiä.

Sivuston selaintuki kattaa ainakin Firefox, Google Chrome, Microsoft Edge, Internet Explorer ja Android Chrome -selaimista muutamat viimeiset versiot, mutta käytännössä sivusto toimii täysin myös vanhemmillakin selaimilla. Vanhimmillaan täysi tuki annetaan vielä Internet Explorer 11 ja Safari 9.1 selaimille, joka tarkoittaa käyttömääriin suhteutettuna todella laajaa selaintukea. Selaintuen ulkopuolelle jäävillä selaimillakin sivuston toiminnallisuus toimii todennäköisesti oikein, vain ulkoasussa saattaa esiintyä eriäväisyyksiä.

Pieniä näytönkokoja sivusto tukee minimissään 320 pikseliä leveisiin näyttöihin asti, mikä käytännössä kattaa kaikki mobiililaitteet joilla verkkosivuja halutaan selata. Esimerkiksi jopa vuonna 2007 julkaistussa ensimmäisessä iPhonessa on 320 pikseliä leveä näyttö.

Sivustolla toimii kaikki perustoiminnallisuus sekä ulkoasut myös JavaScript-esto päällä. Tällöin JavaScriptiä käyttävät valikot levitetään kokonaan auki CSS:n avulla, ja normaalisti JavaScriptillä lähetettävät lomakkeet lähetetään sen sijaan tavallisena http-pyyntönä. Myös viiveladattavat kuvat asetetaan piiloon, ja sen sijaan näytetään noscript-elementeissä olevat tavalliset kuvat. JavaScriptia käyttämättömät selaimet tunnistetaan asettamalla body-osiolle luokka no-js, joka sitten poistetaan käyttämällä JavaScriptia. Tällöin no-js -luokka jää käyttöön vain silloin kun JavaScript ei ole käytössä, ja kyseistä luokkaa voi käyttää tyylien erittelyyn CSS-tiedostossa.

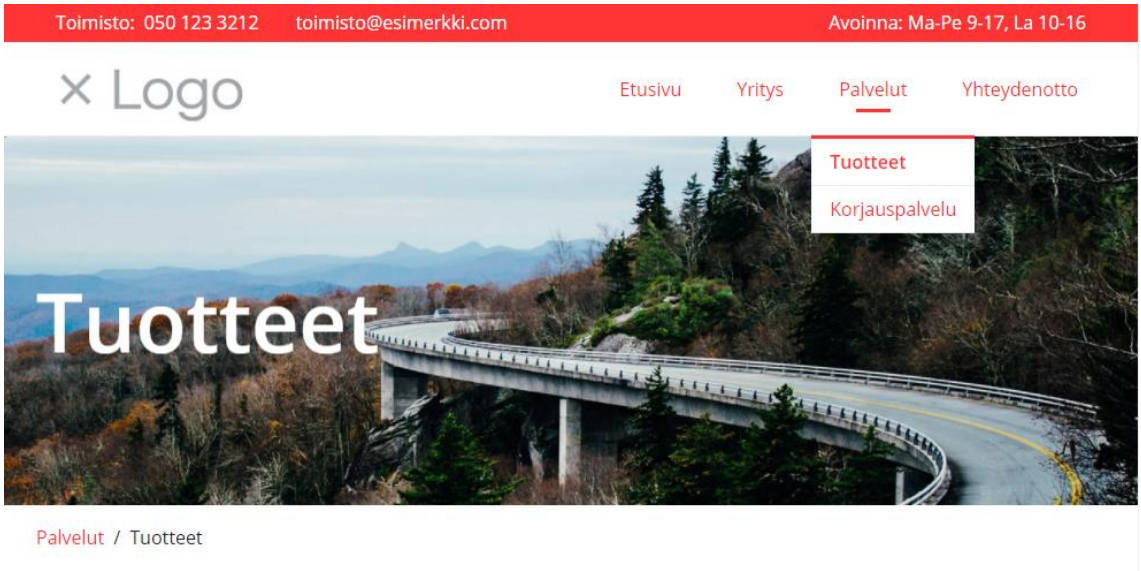
```
<body class="site no-js">
    <script>
        document.querySelector('body').classList.remove('no-js');
    </script>
```

Kuva 17. JavaScriptin toimivuuden tarkistus sivustolla.

4.4.2 Ylätunniste ja navigaatio

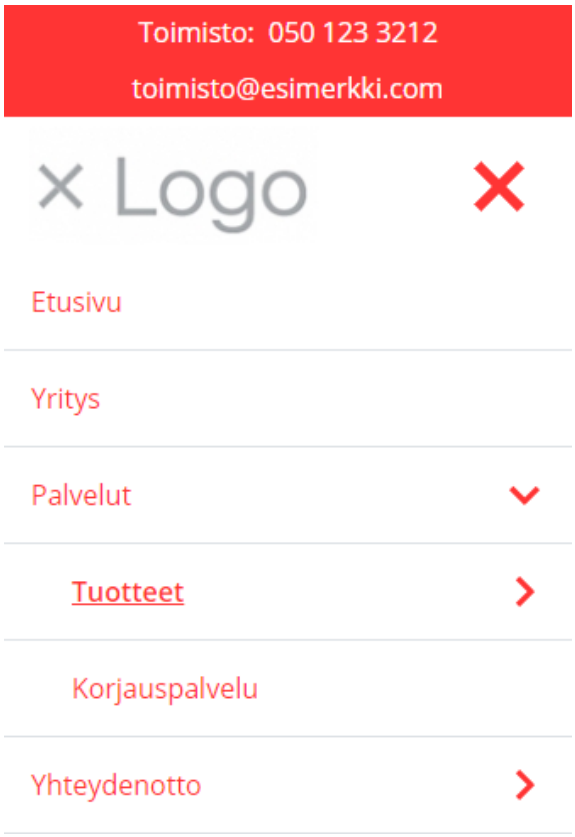
Sivustolla oleva ylätunniste sisältää vakiona yläpalkin, joka sisältää yrityksen tärkeimmät yhteystiedot. Yläpalkki saadaan halutessa piiloon sivuston muuttujat -tiedostosta tai ylläpitopuolen kautta. Yläpalkin alla sijaitsee juurilinkkinä toimiva

yrityksen logo sekä päätason navigaatio, jossa alasivut saadaan auki viemällä hiiri päätason linkin päälle. Tämänhetkinen sivu näkyy alavalikossa tummennettuna ja nykyinen päätason sivu alleviivattuna. Nykyisen sivun otsikon saa automaattisesti näkyviin tekstinä bannerikuvaan, ja koko sivupolku näkyy murupolussa sivun sisältöosion alussa.



Kuva 18. Sivuston ylätunniste, bannerikuva ja murupolku suurella näytönleveydellä.

Pienemmällä näytöllä navigointiin käytetään mobiilivalikkoa. Mobiili- ja sivuvalikossa sivujen linkit näytetään allekkain suurikokoisina, jotta myös kosketusnäyttöä käyttävän on helppo klikata linkkejä. Sivun mahdolliset alasivut saadaan auki painamalla nuoli-ikonin sisältävää painiketta linkin vieressä, mikä on jo vakiintunut tapa avata alavalikko mobiilivalikoissa. Nykyinen sivu näytetään tummennettuna sekä alleviivattuna, ja jos käyttäjä avaa mobiilivalikon alasivulla ollessaan, avataan kaikki nykyisen sivun sivupolun valikot automaattisesti. Itse mobiilivalikko aukaistaan hampurilaisvalikkoikonilla varustettua nappia painamalla, minkä jälkeen ikoni muuttuu raksi-ikoniksi, kun mobiilivalikko on auki.



Kuva 19. Ylätunniste mobiilivalikko aukaistuna.

Kaikissa sivuston navigaatioissa käytetään nav- ja listaelementtejä, jolloin ruudunlukijat tietävät kyseessä olevan navigaatioelementti. Suuren ruudunleveyden normaalisti hover-tapahtumalla toimivaa valikkoa voi selata myös pelkällä näppäimistöllä. Tällöin sivun alavalikko aukeaa aina, kun kyseinen linkki tai sen alasivun linkki on kohdistettuna. Tämän toiminnallisuuden tein kustomoidulla JavaScript-koodilla. Myös mobiili- ja sivuvalikkoa voi selata pelkällä näppäimistöllä.

Mobiilivalikko on elementtien kohdistusjärjestyksessä heti seuraavana valikon aukaisevan hampurilaisvalikkopainikkeen jälkeen, mikä nopeuttaa valikon löytämistä ruudunlukijaa käytettäessä. Valikon linkkejä voi selata käyttämällä sarkainnäppäintä ja alavalikon saa auki painamalla välilyöntiä alavalikon aukaisupainikkeen kohdalla. Ruudunlukukäyttäjille luetaan alavalikon aukaisunapin kohdalla myös alavalikon nimi sekä tieto siitä, onko valikko tällä hetkellä laajennettuna. Tämä toiminnallisuus onnistuu WAI-ARIA:n aria-label- ja aria-expanded-ominaisuuksien sekä JavaScriptin avulla.

Lisäksi sivulle on lisätty kohdistusjärjestyksessä heti ensimmäiseksi ohita navigaatio -linkki, jonka avulla näppäimistö- ja ruudunlukuohjelmakäyttäjät voivat halutessaan ohittaa navigaation ja siirtyä suoraan sivun sisältöön. Linkki on ankkurilinkki sivun main-osioon ja se on täysin näkymätön, kunnes linkki on asetettu kohdistukseen näppäimistön avulla. Näin linkki ei vaikuta hiirellä sivua selaavan käyttäjän käyttökokemukseen, mutta helpottaa näppäimistökäyttäjien sivuston käyttöä.

Kuva 20. Kohdistettuna oleva Ohita navigaatio -linkki sivun alussa.

4.4.3 Lomakkeet

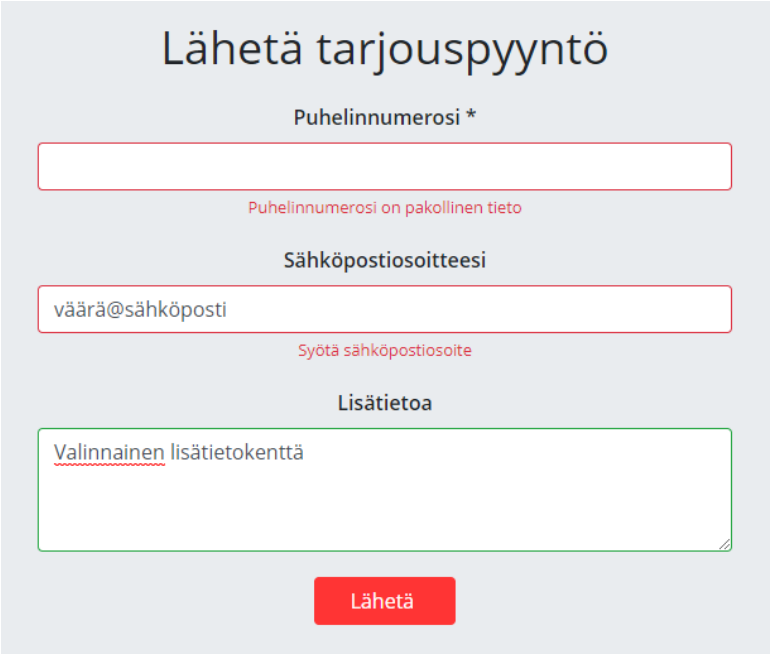
Sivuston lomakkeiden syöttökentissä käytetään kustomoituja tyylejä, kuitenkin säilyttäen käyttäjille tutut kenttien muodot ja toiminnallisuudet. Myös harvinaisemmat kentät, kuten tiedosto-syöttökenttä, on tyylitelty kokonaan teeman mukaisesti. Tyyleihin on käytetty pohjana Bootstrap 4:n tyylejä, ja kenttien tehostevärinä käytetään sivuston muuttujat -tiedostossa asetettua teeman pääväriä. Väritys näkyy esimerkiksi valituissa valintaruuduissa ja -napeissa sekä kohdistetuissa kentissä.

Kuva 21. Kustomoituja lomakkeen tyylejä sivustolla.

Lomakkeissa asetetaan pakollisten kenttien nimikkeiden perään automaattisesti tähtimerkki sekä ruudunlukukäyttäjiäkin varten kentille asetetaan required-ominaisuus. Kenttien nimikkeet ovat asetettu ruudunlukuohjelmille tavoitettaviksi joko suoralla tai epäsuoralla nimikkeellä. Epäsuoraa nimikettä käytetään valintanapeissa ja -ruuduissa, kun taas muissa kentissä käytetään suoraa nimikettä for-ominaisuutta käyttäen. Lomakkeiden selaaminen ja täyttö onnistuu täysin pelkkää näppäimistöä tai näppäimistöä sekä ruudunlukuohjelmaa käyttäen.

Lomakkeiden kentät tarkistetaan aluksi käyttäjäpuolella, jolloin virheellisesti täytettyä lomaketta lähettäessä käyttäjälle näytetään oikein ja väärin täytetyt kentät asettamalla niille vihreä tai punainen reunus. Tämän lisäksi kentän alla näytetään virheteksti virheellisesti täytetyissä kentissä, sillä muuten esimerkiksi punavihersokea käyttäjä ei voisi erottaa virheellisiä ja oikein täytettyjä kenttiä toisistaan. Lisäksi kohdistus asetetaan automaattisesti virheelliseen kenttään. Alustavan tarkistuksen läpäistyään lomake lähetetään käyttäen jQuery:n asynkronista Ajax-pyyntöä. Lähetyksen aikana lähetä-painike asetetaan pois käytöstä ja käyttäjälle näytetään latausanimaatio. Jos palvelinpuolella huomataan virhe, ilmoitetaan vir-

heellisistä kentistä käyttäjälle ilman että mitään käyttäjän antamaa syötettä poistetaan. Palvelinpuolen virheviestissä on myös role-ominaisuutena ”alert”, mikä aiheuttaa sen, että ruudunlukija lukee virheviestin heti sen ilmaannuttua, eikä käyttäjän tarvitse erikseen etsiä sitä.



Kuva 22. Palaute käyttäjälle hänen yritettyä lähettää virheellisesti täytetty lomake.

4.4.4 Kuvat

Sivuston kaikille kuville asetetaan vaihtoehtoinen teksti ruudunlukijatukea varten. Koristeellisille kuville vaihtoehtoinen teksti jätetään tyhjäksi, ja käyttäjille merkityksellisille kuville teksti asetetaan elementin mukaan. Esimerkiksi logorivin elementeillä asiakas syöttää alt-tekstiksikin asetettavan logon nimen itse, kun taas henkilöstölistauksessa kuville tulee vaihtoehtoiseksi tekstiksi automaattisesti kyseisen henkilön koko nimi.

Ruudunlukuohjelmaa tuetaan myös kaikkiin kuvalinkkeihin asetettavilla näkymättömillä selosteilla. Selostetekstit ovat tarpeellisia silloin, kun linkissä ei muutoin olisi kuvan lisäksi minkäänlaista ilmaisutapaa linkin kohteesta. Tämä toiminnallisuus saadaan aikaiseksi joko aria-label-ominaisuudella tai piilotetulla elementillä

linkin sisällä. Kuitenkin piilotettua elementtiä käyttäessä tulee muistaa, että display: none -määrityksellä piilotettua elementtiä ei lueta käyttäjälle ollenkaan.

```css
.sr-only {
    position: absolute;
    width: 1px;
    height: 1px;
    padding: 0;
    overflow: hidden;
    clip: rect(0, 0, 0, 0);
    white-space: nowrap;
    clip-path: inset(50%);
    border: 0;
}
```

Kuva 23. Vain ruudunlukuohjelmalle tarkoitetun luokan CSS-tyylit.

4.5 Suorituskykyoptimoinnit

4.5.1 CSS ja JavaScript -tiedostokutsut

Sivunlatausnopeuden parantamiseksi kaikki sivustolla käytettävät CSS- ja JavaScript-tiedostot, myös CDN:n kautta tulevat, ladataan minifoituina. Tällöin tiedostoista poistetaan kaikki toimivuuteen vaikuttamattomat merkit, kuten rivivälit sekä kommentit, ja muuttujien nimet lyhennetään. Minifointi vaikuttaa paljon tiedostokokoon, esimerkiksi sivuston tyylit sisältävä main.css-tiedosto on ennen minifointia 1.05MB ja minifoinnin jälkeen vain 163KB. Tämän lisäksi tiedosto pakataan gzip-muotoon, jolloin lopulliseksi tiedostokooksi saadaan 20KB. Latausmäärän säästö on siis huomattava, ottaen huomioon, että minifoitu ja pakattu tiedosto toimii täysin samalla tavalla kuin ei-minifoitu versio. Sivuston kehityksessä käytetään tavallisia ei-minifoituja tiedostoja, joista sitten tehdään pakatut ja minifoidut tiedostot julkikäyttöön sivustolle.

Sivustolla käytettävät yleiset JavaScript-kirjastot ladataan CDN:n kautta, joka nopeuttaa sivuston lataamista silloin, kun käyttäjällä on samasta CDN:stä ladattu tiedosto jo selaimen välimuistissa. Tämän vuoksi sivustolla käytetään suosittuja

CDN-palvelimia, joista ladataan jQuery sekä jQuery.Lazy -kirjastojen JavaScript-tiedostot. Käytännössä on melko todennäköistä, että käyttäjä on käynyt aiemmin sivustolla, jossa käytetään ainakin jQuery:a, jolloin hidasta internetyhteyttä käyttäessä voidaan säästää paljon aikaa sivuston ensimmäisellä latauskerralla. Normaalisti Bootstrap 4 vaatisi myös Popper.js -kirjaston sekä Bootstrap:in oman JavaScript-tiedoston. Ne kuitenkin jätetään sivustopohjan tapauksessa pois, sillä sivustolla ei käytetä komponentteja, jotka vaatisivat niitä toimiakseen.

Kaikki selaimella näkyvät kustomoidut CSS-tyylit ovat yhdistetty omaan tiedostoonsa. Näin selaimen tarvitsee tehdä vähemmän pyyntöjä palvelimelle, jolloin sivun latausnopeus paranee. CSS-tyylit sijaitsevat main.css-tiedostossa, mihin tuodaan kaikki elementtien tyylit sekä Bootstrap:n tyylit. Bootstrap:n tyylit ovat karsittuja komponenttikohtaisesti, eli lopulliseen julkipuolen CSS-tiedostoon viedään vain sivustolla käytettyjen Bootstrap:n komponenttien tyylit, mikä pienentää CSS-tiedoston kokoa. Uusien komponenttien tyylejä saa sivustolle poistamalla kehityspuolen SCSS-tiedostoista kommentoinnin importlauseista, mikä helpottaa sivuston jatkokehitystä.

```
@import "bootstrap/scss/card";
// @import "bootstrap/scss/breadcrumb";
// @import "bootstrap/scss/pagination";
// @import "bootstrap/scss/badge";
// @import "bootstrap/scss/jumbotron";
@import "bootstrap/scss/alert";
// @import "bootstrap/scss/progress";
@import "bootstrap/scss/media";
@import "bootstrap/scss/list-group";
// @import "bootstrap/scss/close";
// @import "bootstrap/scss/modal";
// @import "bootstrap/scss/tooltip";
// @import "bootstrap/scss/popover";
// @import "bootstrap/scss/carousel";
```

Kuva 24. Sivustolla käytättämättömät Bootstrap:n komponentit ovat kommentoitu kehityspuolen SCSS-tiedostossa.

Kaikista sivuston tyylitiedostokutsuista main.css-tiedoston ulkopuolelle jää vain tulostustyylit sekä fontit. Tulostustyylien tiedostokutsuun on asetettu ominaisuus

media=”print”, jolloin kyseistä ominaisuutta täysin tukevat selaimet eivät estä sivun renderöintiä tiedostoa ladattaessa. Fontit puolestaan ladataan Googlen omistamasta Google Fonts -palvelusta, jolloin suosittuja fontteja käytettäessä on todennäköistä, että fonttien tyylitiedosto löytyy jo käyttäjän selaimen välimuistista.

CSS-tyylien lisäksi myös JavaScript-koodit on yhdistetty omaan tiedostoonsa nimeltään main.js, joka sisältää kaikki sivustolla käytettävät JavaScript-koodit, joita ei ladata CDN:stä. Myös Hyphenator.js-kirjaston lähdekoodi on main.js-tiedostossa, sillä säädetyistä asetuksista riippuen se on aina erilainen, eikä sillä tämän voi olla vain yhtä CDN-palvelimella sijaitsevaa tiedostoa. Kaikkiin JavaScript-tiedostokutsuihin on asetettu defer-ominaisuus, jonka avulla tiedostojen lataus ei estä dokumentin parsimista tai sivun renderöintiä.

```
<script defer src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.mi
<script defer src="https://cdnjs.cloudflare.com/ajax/libs/jquery.lazy/1.7.6/jqu

<% IF:USER:GROUP_ID IS 1 %>
<script defer src="<% FILE:'/media/layout/dev/janigr/js/main.js' %>"></script>
<% ELSE %>
<script defer src="<% FILE:'/media/layout/build/js/main.js' %>"></script>
<% END IF %>
```

Kuva 25. Kaikissa JavaScript-tiedostokutsuissa on defer-ominaisuus, ja main.js-tiedoston kehitys- ja julkiversio on eritelty kirjautuneen käyttäjän mukaan.

Kaikkien sivustolta kutsuttavien ulkoisten palvelimien kohteisiin tehdään ennenaikainen yhdistäminen käyttämällä preconnect-ominaisuutta. Tällöin selain muodostaa yhteyden kohdepalvelimelle etukäteen eikä vasta silloin, kun tiedoston lataaminen aloitetaan. Preconnect-kutsut ovat sijoitettu HTML-dokumentin head-osioon heti metatagien jälkeen, jolloin selain aloittaa yhdistämisen mahdollisimman aikaisin. Ennenaikainen yhdistäminen auttaa nopeuttamaan sivuston ensimmäisen latauskertaa.

```
<link rel="preconnect" crossorigin href="https://fonts.googleapis.com">
<link rel="preconnect" crossorigin href="https://ajax.googleapis.com">
<link rel="preconnect" crossorigin href="https://cdnjs.cloudflare.com">
<link rel="preconnect" crossorigin href="https://fonts.gstatic.com">
```

Kuva 26. Kaikkia ulkoisia palvelinkutsuja varten on tehty erikseen ennenaikainen palvelimelle yhdistäminen.

4.5.2 Kuvat

Sisällönhallintajärjestelmää käyttäessä asiakas voi asettaa sivustolle haluamiaan kuvia. Tästä huolimatta kuville voidaan suorittaa optimointeja, jotta kuvakokoa saadaan pienennettyä mahdollisimman paljon. Sivuviidakolla voidaan rajoittaa automaattisesti kuvien kokoa, eli jos sivuston rakentaja tietää, että kuvapaikka on maksimissaan 500 pikseliä leveä, asiakkaan asettama kuva voidaan pienentää 500 pikseliä leveäksi. Kuvien koon pienentämisen lisäksi kuville voidaan asettaa päälle progressiivinen lataaminen ja värisävyjen aliotanta, mikä auttaa nopeuttamaan kuvien lataamista käyttäjälle. Kuvien optimointi tapahtuu palvelinpuolella ja käsitellyt kuvat tallennetaan palvelimelle, mistä ne voidaan myöhemmin hakea verkkosivustolle. Esimerkkitestinä verkosta ladattu kuva, joka on resoluutioltaan 6000x3000. Jos sen asettaisi sivustolle sellaisenaan, se veisi tilaa 3,23MB. Pienennettäessä kuvaa 1280 pikseliä leveään kuvapaikkaan sopivaksi, eli samalla kuvasuhteella resoluutioon 1280x853, kuvan tiedostokoko on 880KB. Kun tähän lisätään Sivuviidakon kuvaoptimoinnit, niin 1280x853 resoluutioisen kuvan kooksi saadaan 147KB. Optimoitu testikuva on siis vain noin kahdeskymmeneskahdesosan kokoinen alkuperäisen kuvan tiedostokoosta tai noin kuudesosa ainoastaan pienennetyn kuvan tiedostokoosta.

```
data-src="
    [IF:MODULE.SUUREMMAT-ELEMENTIT]
        [PROPERTY:Kuva@size=600x999;progressive=true;chromasubsampling=true;]
    [ELSE]
        [PROPERTY:Kuva@size=200x999;progressive=true;chromasubsampling=true;]
    [/IF]"
```

Kuva 27. Ehtolause Sivuviidakon moduuliasetusten mukaan viiveladattavan kuvan kuvalähteessä. Molemmat kuvat rajataan leveyden mukaan ja molemmille kuvalähteille asetetaan kuvaoptimointeja päälle.

Käyttäjälle ladattavien tiedostojen määrää vähentää myös se, että sivustolla käytetään yleisissä elementeissä kuvien sijaan CSS-tyylejä ja vektorigrafiikkaa. Esimerkiksi mobiilivalikon hampurilaisikoni ja raksi-ikoni ovat tehty kokonaan CSS-määrityksillä, sekä valikoiden nuoli-ikonit ja alatunnisteen someikonit ovat SVG-kuvia. Nämä vievät huomattavasti vähemmän tilaa kuin kuvamuotoiset ikonit ja ovat tarpeen tullen paremmin skaalautuvia isommiksi ikoneiksi.

Sivustolla käytetään kaikkien elementtien kuvien näyttämiseen viivelatausta, lukuun ottamatta yrityksen logoa ylätunnisteessa. Viivelataamisen avulla kuvien lataaminen voidaan aloittaa vasta sitten kun sivuston muu sisältö on valmiina, mikä nopeuttaa sivun merkittävän sisällön lataamista käyttäjälle. Viivelataamista käytetään sekä tavallisiin kuvapaikkoihin että taustakuviin. Kuitenkin tavallisia kuvapaikkoja käyttäessä kuvapaikan korkeus on normaalisti 0 pikseliä ennen kuvan lataamista, jolloin kuvan alla oleva sisältö hyppää silloin kun kuva renderöidään kuvapaikkaan. Näissä tilanteissa sisällön hyppimistä estetään asettamalla kuvapaikalle korkeutta kuvasuhteen verran padding-top CSS-määrityksellä. Vertikaalisen padding-määrityksen vertailuarvona käytetään elementin vanhemman leveyttä silloin kun sen arvona on prosenttimuotoinen arvo. Näin elementille saadaan siis esimerkiksi 16:9 -kuvasuhteen korkeus asettamalla padding-top-määrittelyyn arvo 56.25%. Kuvan ladattua padding-arvo poistetaan, jolloin kuvapaikan korkeus lasketaan kuvan korkeuden mukaan.

Sivuston kaikkia viiveladattavia kuvia ei ladata kerralla, vaan ensin ladataan ne kuvat, jotka näkyvät käyttäjälle näytöllä. Tämä on viivelataamiseen käytetyn jQuery.Lazy-kirjaston ominaisuus, mikä vähentää yhtäaikaisten latausten määrää sivulla. Muut kuvat ladataan käyttäjän vierittäessä sivua alaspäin ja lähestyessä alempana piilossa olevaa kuvaa. Se, milloin alempana sivulla olevaa kuvaa aloitetaan lataamaan, voidaan määritellä viivelataamisen alustavassa funktiossa. Tällä hetkellä rajaksi on määritelty 1000 pikseliä, minkä pitäisi estää suurimmalta osin kuvien ilmestyminen ruudulle silloin kun ne ovat jo käyttäjän näkyvillä.

Sivustopohjan sivuille on mahdollista asettaa bannerikuva, mikä on kooltaan aina koko sivun levyinen. Bannerikuva käyttää elementin taustakuvaa, jolle ei voi asettaa srcset-ominaisuutta, jonka avulla saataisiin ladattua taustakuva lähdekuvan elementin leveyden mukaan. Kuitenkin koko sivun levyiselle elementille se olisi erityisen tarpeellista, sillä ei kannata ladata esimerkiksi 400 pikseliä leveälle mobiilinäytölle 1920 pikseliä leveää kuvaa. Tämän vuoksi tein JavaScriptillä koodin, jolla kyseinen toimenpide onnistuu.

Taustakuvan srcset-toiminnallisuuden saavuttamiseksi asetetaan taustakuvaelementille ensin kuvalähteeksi pienikokoinen kuva, jotta elementin tausta ei ole

tyhjä kuvan latauksen ajan. Taustakuvana toimivan div-elementin sisällä on tavallinen kuvaelementti, jolle on asetettu tyylit siten, että se on piilossa käyttäjältä, mutta se ei ole piilotettu display: none -määrityksellä. Tällöin selain lataa kuvaelementtiin kuvalähteen normaalisti, mutta kuvaa ei näy käyttäjälle. Tälle kuvaelementille asetetaan viivelatauksen luokka, data-srcset-ominaisuuteen ladattavat kuvakoot, data-src-ominaisuuteen täysikokoisen kuvan lähde, data-sizes-ominaisuuteen kuvapaikan leveydet sekä aria-hidden-ominaisuus arvoon ”true”. Tällöin srcset-ominaisuutta tukevat selaimet laskevat tälle piilotetulle kuvaelementille kuvalähteen tavallisesti data-srcset ja data-sizes -ominaisuuksien mukaan. Aria-hidden-ominaisuus varmistaa sen, ettei ruudunlukijat lue kuvaa käyttäjälle.

```
<div
    class="nojs-hidden"
    style="background-image: url('[PROPERTY:Kuva@size=200x9999;progressive=true;chromasubsampling=

    <img
        class="lazyload set-parent-bg"
        data-srcset="[PROPERTY:Kuva@size=400x9999;progressive=true;chromasubsampling=true;] 400w,
                [PROPERTY:Kuva@size=720x9999;progressive=true;chromasubsampling=true;] 720w,
                [PROPERTY:Kuva@size=991x9999;progressive=true;chromasubsampling=true;] 991w,
                [PROPERTY:Kuva@size=1280x9999;progressive=true;chromasubsampling=true;] 1280w,
                [PROPERTY:Kuva@size=1440x9999;progressive=true;chromasubsampling=true;] 1440w,
                [PROPERTY:Kuva@size=1920x9999;progressive=true;chromasubsampling=true;] 1920w"
        data-src="[PROPERTY:Kuva@size=1920x9999;progressive=true;chromasubsampling=true;]"
        data-sizes="100vw"
        aria-hidden="true"
        alt="" />
</div>
```

Kuva 28. HTML-koodi, jolla saa srcset-toiminnallisuuden elementin taustakuvalle. Kuvasta on karsittu pois ylimääräisiä elementtejä ja luokkia luettavuuden vuoksi.

Sivua ladatessa kuunnellaan JavaScriptilla kaikkien set-parent-bg-luokkaisten kuvalähteiden lataustapahtumaa. Kuvan ladattua asetetaan kuvaelementin vanhemmalle, eli haluamallemme taustakuvaelementille, kuvalähteeksi kuvaelementin currentSrc-ominaisuus. Selain tallentaa currentSrc-ominaisuuteen kuvaelementin nykyisen kuvalähteen, jonka se on laskenut parhaaksi srcset ja sizes -ominaisuuksien avulla. Tällöin kohde-elementin taustakuvalle saa saman toiminnallisuuden kuin mitä tavallinen kuvan srcset-ominaisuus antaa, sillä selainikkunaa leventäessä selain laskee uuden parhaan kuvalähteen kuvaelementille ja käynnistää tarpeen tullen uuden lataustapahtuman. Lisäksi taustakuvan kuvalähdettä asettaessa tarkastetaan samalla, jos selain ei tuekaan currentSrc-ominaisuutta. Tällöin srcset-ominaisuutta ei voida käyttää, mutta kuvalle asetetaan sen sijaan täysikokoinen kuva src-ominaisuudesta.

Selainikkunaa laajentaessa Safari-selaimet eivät kuitenkaan tarkista onko srcset-ominaisuudessa olemassa parempaa kuvalähdettä suhteessa elementin leveyteen. Bannerikuvan osalta tämä on erittäin huono asia, koska kuvapaikka on koko selainikkunan levyinen. Tällöin kapeaa selainikkunaa laajentaessa kuvan resoluutio voi jäädä pieneksi, ja tuloksena on käyttäjälle näkyvä huonolaatuinen kuva. Siksi tein myös varmistuksen, joka pakottaa kuvalähteen uudelleenlaskemisen srcset ja sizes -ominaisuuksien avulla. JavaScript funktio kuuntelee selainikkunan koon muutostapahtumaa ja laskee, onko selainikkunaa laajennettu vai kavennettu. Jos selainikkunaa on laajennettu, jokaiselle set-parent-bg-luokkaiselle kuvapaikalle asetetaan kuvalähteeksi niiden nykyinen kuvalähde. Tämä pakottaa kuvalähteen uudelleenlaskennan myös Safari-selaimilla, eikä asetettavaa kuvaa tarvitse ladata uudestaan, sillä se on jo välimuistissa tallessa. Kuvalähteelle ei tarvitse tehdä uudelleenlaskentaa selainikkunaa kavennettaessa, sillä jo ladattu suurempi kuva ei vaikuta kuvalaatuun huonontavasti pienemmällä kuvapaikalla. Suorituskyvyn parantamisen vuoksi selainikkunan kokomuutoksia lasketaan 250 millisekunnin viiveellä koon muuttamisesta, jotta uutta leveyttä ei tarvitse laskea joka pikselillä erikseen, vaan ainoastaan silloin kun käyttäjä on lopettanut selainikkunan koon muuttamisen. Lisäksi selainikkunan kokomuutoksen kuuntelua ei käytetä, jos selain ei tue kuvaelementin currentSrc-ominaisuutta tai jos set-parent-bg-luokkaisia kuvia ei ole sivulla ollenkaan. JavaScript-koodi taustakuvan srcset-ominaisuuden saavuttamiseksi on kokonaisuudessaan liitteessä 2.

5 Sivustopohjan testaukset

5.1 Suorituskykytestaus

Sivuston suorituskykyä mitataan käyttäen Google Chromen kehittäjätyökaluja sekä Googlen kehittämää WebPagetest-palvelua. Google Chromen kehittäjätyökaluilla on mahdollista muun muassa määrittää verkon nopeuden ja viiveen rajoituksia sekä nähdä helposti sivunlatauksen eri tapahtumien valmistumisaikoja.

Kehittäjätyökaluista löytyy myös Audits-paneeli, jolla saa sivun nopeudelle ja rakenteelle arvostelun pistemuodossa, sekä näkee mitä sivustolla voidaan vielä parantaa. Audits-testi käyttää Googlen Lighthouse-palvelua arvioidakseen sivuston ominaisuuksia.

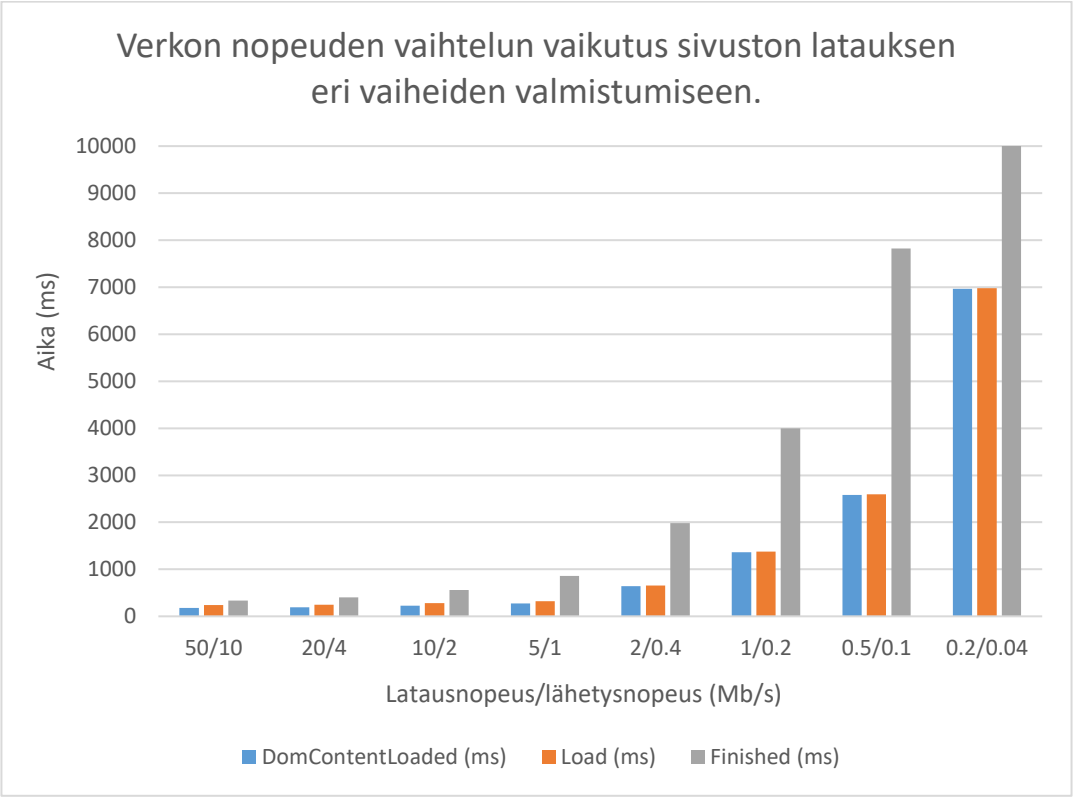
Tarkempia sivunlatausanalyyseja nähdään käyttäen WebPagetest-palvelua, joka testaa sivulatausta käyttäen oikeita selaimia ja yhteyksiä. Testeihin voi itse valita muun muassa maantieteellisen sijainnin mistä testi ajetaan, yhteystyypin, yhteysnopeuden, käytettävän selaimen sekä tehtävien testien määrän. Testien tuloksista nähdään yksityiskohtaisesti tiedostojen lataus ja suoritus -järjestykset, sivunlatauksen eri vaiheiden tarkat ajat, ladattavien tiedostojen koot sekä paljon muuta.

5.1.1 Verkon nopeuden vaihtelun vaikutus sivulataukseen

Ensimmäisten testien tarkoituksena on antaa kuva siitä, kuinka verkon nopeuden vaihtelu vaikuttaa sivulatauksen valmistumisnopeuteen. Testit tehdään käyttäen Google Chromea (versio 65.0) incognito-tilassa, välimuisti kytkettynä pois päältä sekä rajoittaen verkon nopeutta. Testit tehdään täysin peräkkäin, jolloin selaimen yhdistämisaika palvelimille saattaa olla pienempi kuin juuri selaimen avattua. Tässä testissä keskitytään vain verkon nopeuden vaihteluun, joten prosessorin tehoa ei rajoiteta. Testikoneena toimii Dell:n kannettava tietokone, jossa on Intel® Core™ i5-7200U 2.50Ghz kaksiydinprosessori sekä 16GB keskusmuistia. Selainikkunan leveys testien aikana on 1366 pikseliä, sillä se oli verkkoselaamisessa eniten käytetty pöytätietokoneen näytönleveys vuoden 2018 alussa (StatCounter Global Stats, 2018).

Testattavana sivuna käytetään sivustopohjan etusivua, sillä se sisältää eniten erilaista sisältöä, sekä koska usein käyttäjä tekee sivuston ensimmäisen sivulatauksen etusivulle. Verkon nopeuden rajoitus tehdään Google Chromen kehittäjätyökaluista löytyvällä verkon nopeuden rajoittimella, jonka testasin pienellä virhemarginaalilla luotettavaksi käyttäen suosittua Ooklan Speedtest-palvelua.

Sivun latausnopeudet katsotaan tekemällä jokaisella testattavalla verkon nopeudella 5 testiä ja laskemalla niistä keskiarvo. Lähetysnopeudeksi asetetaan noin viidesosa latausnopeudesta. Verkon viive jätetään rajoittamatta, sillä sen vaikutus sivun latausnopeuteen testataan erikseen. Tapahtumat joita mitataan ovat DOM Content Loaded, Load sekä Finish. Tiedot näiden tapahtumien valmistumisesta näkyy Google Chromen kehittäjätyökalujen Network-paneelista. Käyttäjälle merkittävin tapahtuma on DOM Content Loaded, jonka jälkeen aloitetaan tekstien renderöinti. Load-tapahtuma tapahtuu silloin, kun sivun kaikki tiedostokutsut ovat ladattuina ja sisältö renderöitynä. Load-tapahtumaa seuraa Finished-tapahtuma, joka tarkoittaa sitä, että kaikki sivuston kyseisellä hetkellä tarvitsemat resurssit ovat ladattu kokonaan.



Kaavio 1. Verkon nopeuden vaihtelun vaikutus sivuston latauksen eri vaiheiden valmistumiseen. Huom. viimeinen Finished-tapahtuman valmistumisaika ei mahdu kokonaan kuvaajaan.

Testien tuloksesta nähdään kuinka DOM Content Loaded-tapahtuman kannalta yli 5Mb/s latausnopeudella saavutettu etu on suhteessa huomattavasti pienempi kuin hitaammilla yhteyksillä. Tämän vuoksi verkkosivun latausnopeuden suorituskykyoptimointi on erityisen tärkeää mobiililaitteille, sillä niillä verkon nopeudet

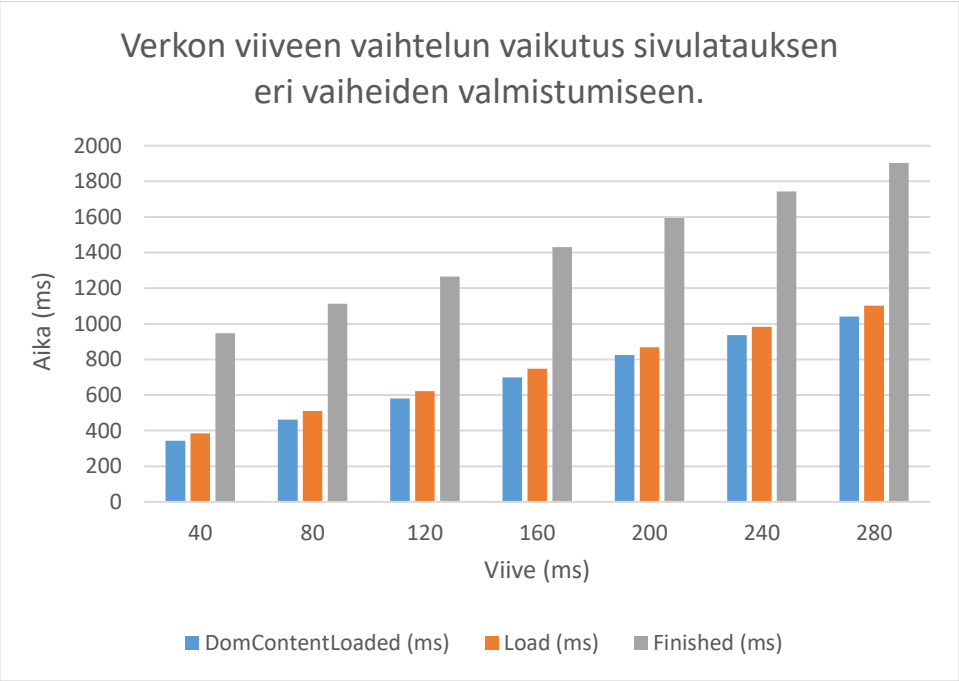
ovat yleisesti hitaampia kuin esimerkiksi pöytätietokoneilla. Kuitenkin latausnopeudeltaan vain 0.5Mb/s verkollakin optimoidun esimerkkisivun sisällön renderöinti aloitettiin alle kolmessa sekunnissa käyttäjän sivulle siirtymisestä. Testituloksiin kuitenkin vaikutti samassa selainikkunassa peräkkäin tehdyt testit, sillä tällöin selaimella ei välttämättä kestä niin kauan aikaa yhdistää uudelle palvelimelle kuin niin sanotulla täysin tyhjiltään lataamisella. Peräkkäiset testit kuitenkin antavat tarkemman tiedon sivun tiedostokutsujen lataamisnopeuksien eroista eri verkon nopeuksilla, sillä tiedostojen lataamisnopeuksien vaihtelu ei ole niin suurta kuin vaihtelu palvelimille yhdistämisien ajoissa.

Testituloksista huomataan myös se, kuinka DOM Content Loaded ja Load -tapahtumien välinen aika on erittäin pieni, eikä verkon nopeus vaikuta siihen käytännössä ollenkaan. Tämä johtuu siitä, ettei esimerkkisivulla ole kuin kaksi pientä viivelataamatonta kuvaa. Jos viivelataamattomia kuvia olisi enemmän, Load-tapahtuma myöhästyisi niin kauan kunnes kaikki kuvat olisivat ladattuina sekä renderöityinä. Nyt pienet kuvat ladataan melkein aina ennen DOM Content Loaded -tapahtumaa sekä pienen resoluution vuoksi ne renderöidään nopeasti. Load-tapahtuma myöhästyisi myös, jos käytössä olisi esimerkiksi huonosuorituskykyinen mobiililaite, mutta yleisesti verkon nopeudella on prosessointikykyä suurempi merkitys sivun latausnopeuden kannalta.

Sivulataustesteissä Finished-tapahtuma valmistuu aina huomattavasti Load-tapahtuman jälkeen. Tämä on hyvä asia, sillä käytännössä esimerkkisivustopohjan tapauksessa Load ja Finished -tapahtumien välissä tapahtuu vain viiveladattavien kuvien lataaminen. Ilman viivelataamista kuvat ladattaisiin samaan aikaan muiden resurssien kanssa, jolloin Load- ja DOM Content Loaded -tapahtumat voisivat viivästyä. Nyt siis kuvien lataus ei häiritse muiden resurssien lataamista ja käyttäjä pääsee aloittamaan sivun tekstisisällön selaamisen nopeammin. Tämän testin kaikkien testikierrosten tulokset ovat esitelty erikseen liitteessä 3.

5.1.2 Verkon viiveen vaihtelun vaikutus sivulataukseen

Seuraavan testin ideana on tutkia kuinka viiveen lisääminen vaikuttaa sivuston eri lataustapahtumien valmistumiseen. Testiolosuhteet ovat samat kuin verkon nopeuden vaikutuksen testissä, eli testattavana sivuna toimii esimerkkisivuston etusivu, prosessoritehoa ei rajoiteta ja selaimena käytetään Google Chromea (versio 65.0) incognito-tilassa. Viivettä rajoitetaan Google Chromen kehittäjätyökaluista löytyvän rajoittimen avulla. Verkon latausnopeus on rajoitettu nopeuteen 5 Mb/s ja lähetysnopeus on 1 Mb/s.



Kaavio 2. Verkon viiveen vaihtelun vaikutus sivuston latauksen eri vaiheiden valmistumiseen.

Testin tuloksista huomataan, että myös verkon viiveen kasvu vaikuttaa verkkosivun lataustapahtumien valmistumisaikaan heikentävästi. Samalla nähdään, kuinka viiveen vaikutus on lineaarinen, toisin kuin verkon nopeuden vaihtelun vaikutus. Suuri viive vaikuttaa siis sivulatauksen nopeuteen, vaikka verkon nopeus olisi muuten hyvä. Kuitenkin viiveen vaikutus koskee eniten mobiililaitteiden omistajia, sillä mobiililaitteissa on usein langaton 2G, 3G tai 4G -verkkoyhteys, jolloin viive on suurempi kuin kiinteissä verkkoyhteyksissä.

Sivulatauksessa viiveen hidastamiskertoimeen vaikuttaa olennaisesti ladattavien tiedostojen määrä. Jos sivulla on monta pientä tiedostoa, jotka selain joutuu erikseen lataamaan, on myös palvelimelle yhdistämispyyntöjä paljon enemmän. Tällöin viive vaikuttaa enemmän kuin yhden ison tiedoston ladatessa. Esimerkkisivustolla on kuitenkin melko vähän tiedostokutsuja sivulatauksien yhteydessä, jolloin viiveen vaikutus jää pienemmäksi. Viivetestin kaikkien testikierrosten tulokset ovat esitelty erikseen liitteessä 4.

5.1.3 Sivulataustestit WebPagetest-palvelulla

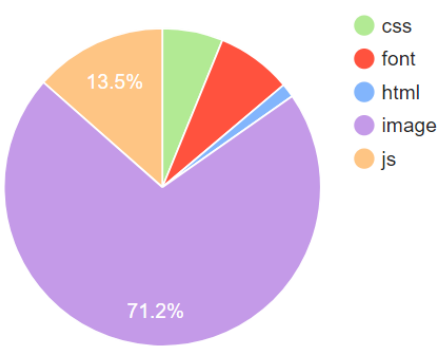
Esimerkkisivuston tarkemmat lataustestit tehdään WebPagetest-palvelua käyttäen, sillä se antaa tarkat tiedot sivulatauksen tapahtumista sekä vesiputousmallin eri vaiheista latauksen aikana. Testit tehdään käyttäen Amsterdamissa sijaitsevaa palvelinta, Chrome selainta ja eri yhteysnopeuksia. Sivulataustestejä ajetaan joka testikierroksella viisi kappaletta ja tutkittavaksi otetaan DOM Content Loaded -tapahtuman mukaan järjestetystä joukosta mediaanituloksen omaava sivulataustesti.

Testeistä tutkitaan esimerkkisivuston latausaikoja eri verkkoyhteyksillä sekä lisäksi katsotaan kuinka kaikista helpoimmat suorituskykyoptimoinnit vaikuttavat sivulatausnopeuteen. Tuloksia tulkitaan sivulatauksen lataustapahtumien aikojen avulla sekä tiedostokutsujen käyttäytymistä esittävän vesiputousmallin avulla. Käyttäjän kannalta kaikista tärkein tutkittavista asioista on kuitenkin DOM Content Loaded -tapahtuma, sillä oikein optimoidulla sivulla sen lähettyvillä aloitetaan ensimmäinen merkityksellinen piirto, jolloin selain aloittaa renderöimään käyttäjän kannalta merkittävää sisältöä sivulle.

5.1.4 Testi tyypillisellä suomalaisella verkkoyhteydellä

Ensimmäinen testi tehtiin verkkoyhteydellä, jossa latausnopeus on 20.5 Mb/s, lähetysnopeus 4 Mb/s ja viive 30ms. Testin latausnopeus simuloi keskimääräistä

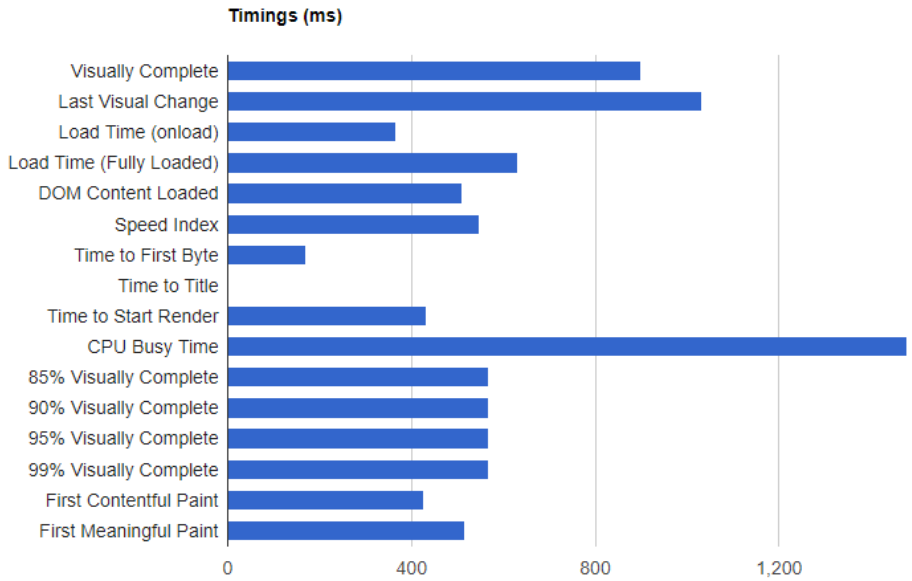
latausnopeutta Suomessa vuoden 2017 ensimmäisellä neljänneksellä (Akamai, 2017). Lähetysnopeus on latausnopeuteen suhteutettuna arvioitu nopeus ja viive kuvaa tyypillistä kiinteän laajakaistayhteyden viivettä.



MIME Type	Bytes	Uncompressed
image	268,932	268,023
js	51,098	133,962
font	29,032	28,592
css	23,371	173,440
html	5,338	0
flash	0	0
other	0	0
video	0	0

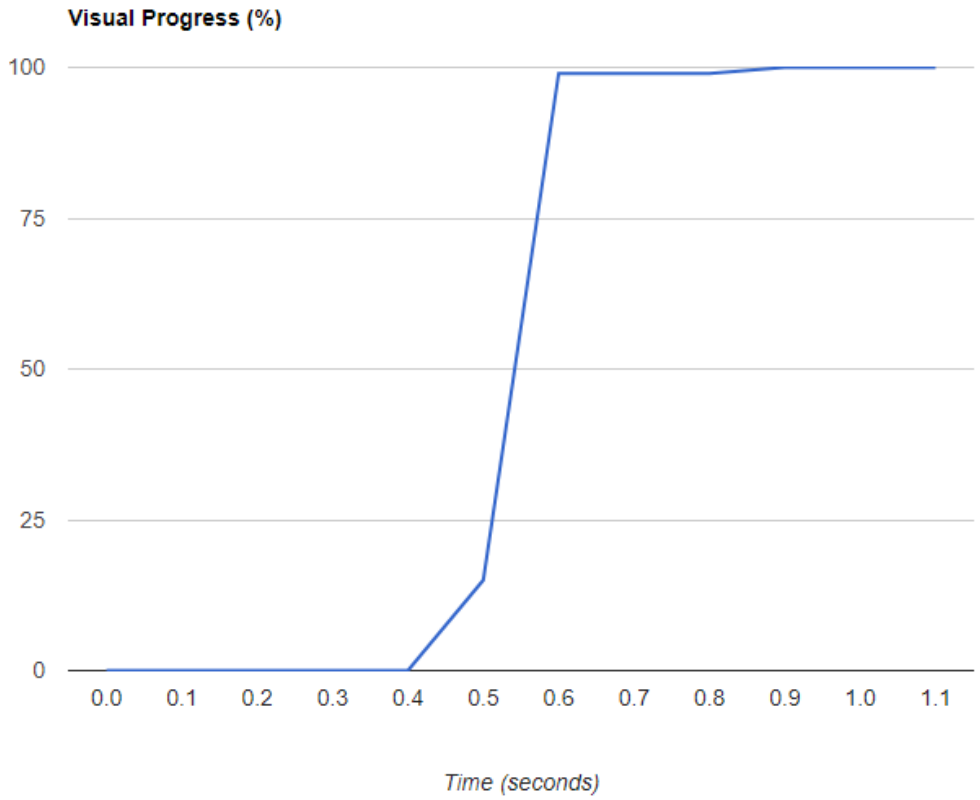
Kuva 29. Esimerkkisivuston ladattujen MIME-tyyppien määrät prosentteina sekä tiedostokoot tavuina.

WebPagetest-palvelun tuloksista nähdään muun muassa erottelu esimerkkisivuston ladatusta sisällöstä MIME-tyypeittäin (Multipurpose Internet Mail Extensions). Erottelusta huomataan, kuinka kuvien yhteenlaskettu tiedostokoko on selvästi muita tyyppejä suurempi. Tämä ei kuitenkaan ole ongelma, sillä sivuston kaikki suuremmat kuvat ladataan viiveellä, jolloin niiden lataaminen ei hidasta merkittävän sisällön renderöintiä. Taulukosta huomataan myös, kuinka tiedostojen pakkaus vaikuttaa lopulliseen tiedostokokoon. Esimerkiksi sivun CSS-tiedostot veisivät yli seitsemän kertaa enemmän tilaa pakkaamattomina.

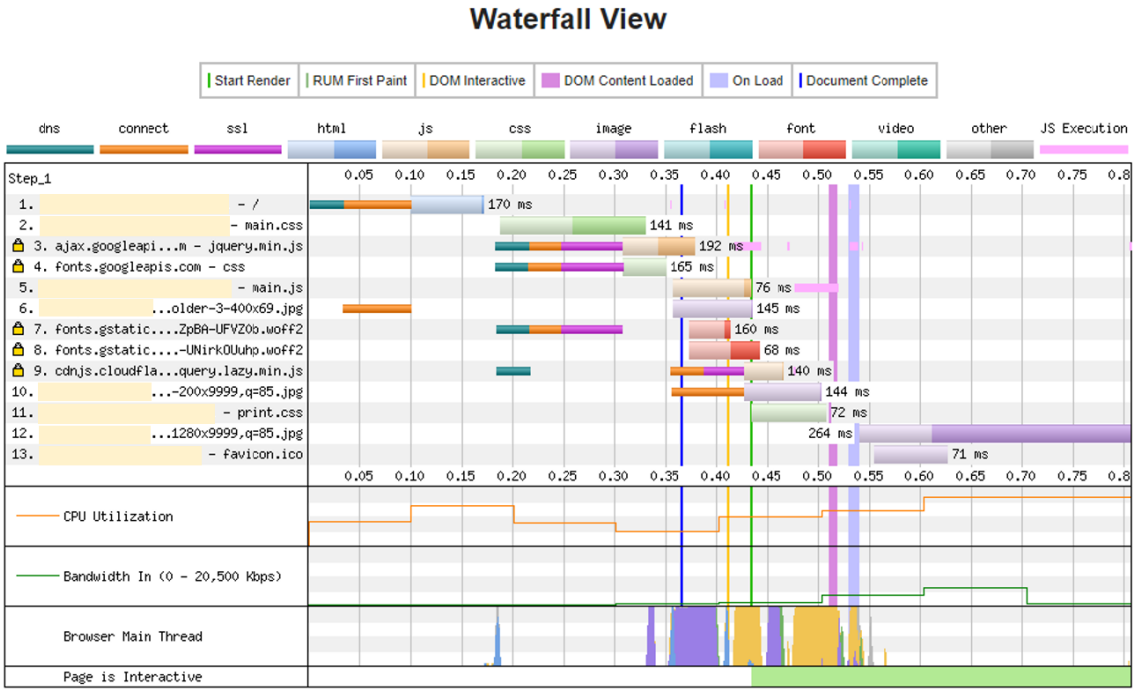


Kuva 30. Esimerkkisivuston eri latausvaihteet ja niiden valmistumisajat.

WebPagetest antaa myös tarkan kaavion sivulatauksen eri latausvaiheista. Kaaviosta nähdään kuinka ensimmäinen sivulle piirto (426ms) tapahtui jo ennen DOM Content Loaded -tapahtumaa (511ms). Ensimmäinen merkityksellinen piirto tapahtui melkein heti DOM Content Loaded:in jälkeen (517ms). Ensimmäinen merkityksellinen piirto voi tässä tilanteessa tarkoittaa esimerkiksi taustavärien renderöinnin aloittamista sivulla. Kuitenkin kaikkien renderöintiin vaadittujen tiedostojen latauduttua itse renderöinti tapahtuu nopeasti, sillä sen aika riippuu täysin laitteiston ja selaimen suorituskyvystä.



Kuva 31. Sivun näkyvän sisällön renderöinnin edistyminen sivulatauksessa.



Kuva 32. Vesiputousmalli esimerkkisivun tiedostolatauksista.

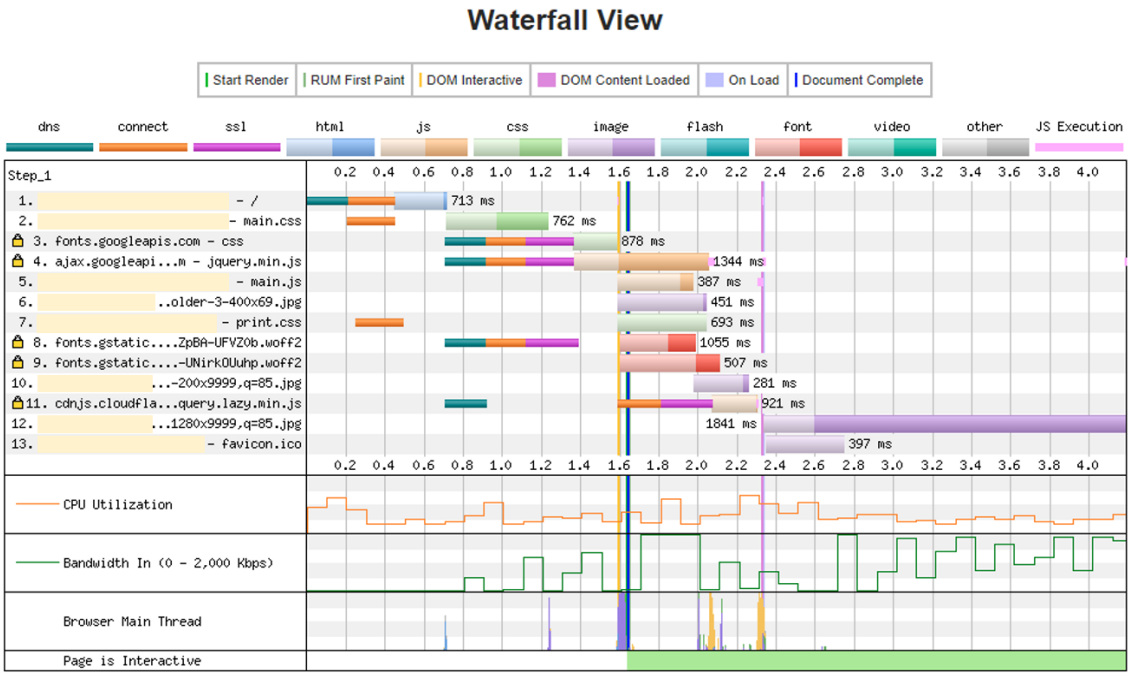
Sivulatauksen vesiputousmallista huomataan, kuinka selain aloitti yhdistämällä palvelimelle, jossa HTML-tiedosto sijaitsee. DNS-haussa sekä palvelimelle yhdistämisessä kesti 99 millisekuntia. Palvelimelta selain sai HTML-tiedoston takaisin 69 millisekunnissa ja tiedosto ladattiin kahdessa millisekunnissa. HTML-tiedostoa parsittiin 20 millisekuntia, jonka jälkeen selain aloitti lataamaan main.css-tiedostoa ja suoritti preconnect-linkkien avulla ennenaikaisen yhdistämisen CDN-palvelimille. Samanaikaisesti main.css-tiedoston latauksen kanssa selain aloitti lataamaan tyylitiedostoa fonteille sekä jQuerya.

Main.css-tiedoston lataamisen jälkeen selain aloitti hakemaan loppuja JavaScript-tiedostoja, fontteja sekä yrityksen logoa ja sivun bannerissa sijaitsevaa paikanpitäjäkuvaa. Yrityksen logoa ei ladata viiveellä, sillä se on ensimmäisenä käyttäjälle näkyvässä näkymässä, ja sen tiedostokoko on suhteellisen pieni. Bannerin paikanpitäjäkuva ladataan ensin pieniresoluutioisena, jotta banneri ei olisi hitailla yhteyksillä taustaltaan pitkään vain yksivärinen, johon vain yhtäkkiä ilmestyy kuva taustalle.

Ensimmäisen ruudulla näkyvän asian selain piirsi 426 millisekunnin kohdalla, samalla kuin toinen fonteista ja bannerikuva olivat vielä latautumassa. Ensimmäinen merkityksellinen piirto tapahtui DOM Content Loaded -tapahtuman jälkeen ajassa 517 millisekuntia, ja dokumentti oli valmis 631 millisekunnin kohdalla. Tiedostolatausten vesiputousmallia tarkastellessa on kuitenkin hyvä muistaa, ettei tiedostojen latausajat ole joka sivunlatauskerta samanlaisia, minkä vuoksi testejä tehdään monta ja tarkasteltavaksi otetaan mediaanitulos

5.1.5 Eri suorituskykyoptimointitapojen vaikutus sivulataukseen

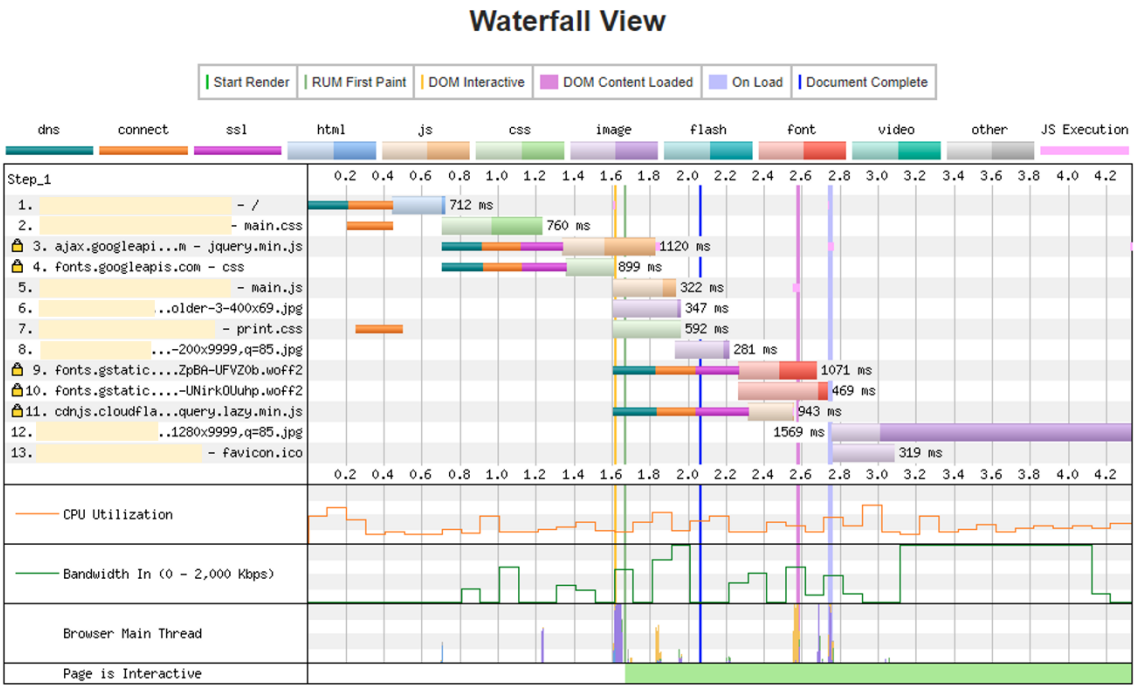
Seuraavaksi tutkitaan kuinka sivuston rakentajan kannalta yksinkertaisesti tehtävät suorituskykyoptimointitavat vaikuttavat sivulatauksen valmistumiseen. Testaaminen tehdään jälleen WebPagetest-palvelulla, jotta testeistä saadaan mahdollisimman tarkat tulokset. Testit tehdään simuloiden hitaampaa verkkoyhteyttä, sillä suorituskykyoptimoinnin tulokset näkyvät parhaiten hitailla yhteyksillä. Verkon latausnopeus on 2 Mb/s, lähetysnopeus 0.5 Mb/s ja viive 200 millisekuntia. Samaan tapaan kuin viime testissä, testikierroksia tehdään viisi kappaletta ja tarkasteltavaksi otetaan DOM Content Loaded -tapahtuman mukaan järjestetystä joukosta mediaanituloksen omaava sivulataustesti.



Kuva 33. Vertailukohteena olevan sivulatauksen vesiputousmalli.

Vertailukohteena olevan sivulataustestin ensimmäinen piirto sivulle tapahtui jo 1646 millisekunnin kohdalla. Kuitenkin DOM Content Loaded -tapahtuma oli valmis vasta 2 324 millisekunnissa ja Load-tapahtuma 2 334 millisekunnissa. Vesiputousmallista nähdään, kuinka selain suoritti ennenaikaiset palvelimille yhdistämiset heti HTML-dokumentin latauduttua ja isot kuvat ladattiin vasta Load-tapahtuman jälkeen, kun dokumentti oli muuten valmis.

Ensimmäiseksi testataan, kuinka ennenaikaisen palvelimelle yhdistämisen ottaminen pois vaikuttaa sivustolataukseen. Muuten testin puitteet ovat samat, eli sama selain samalla yhteysnopeudella ja -viiveellä.

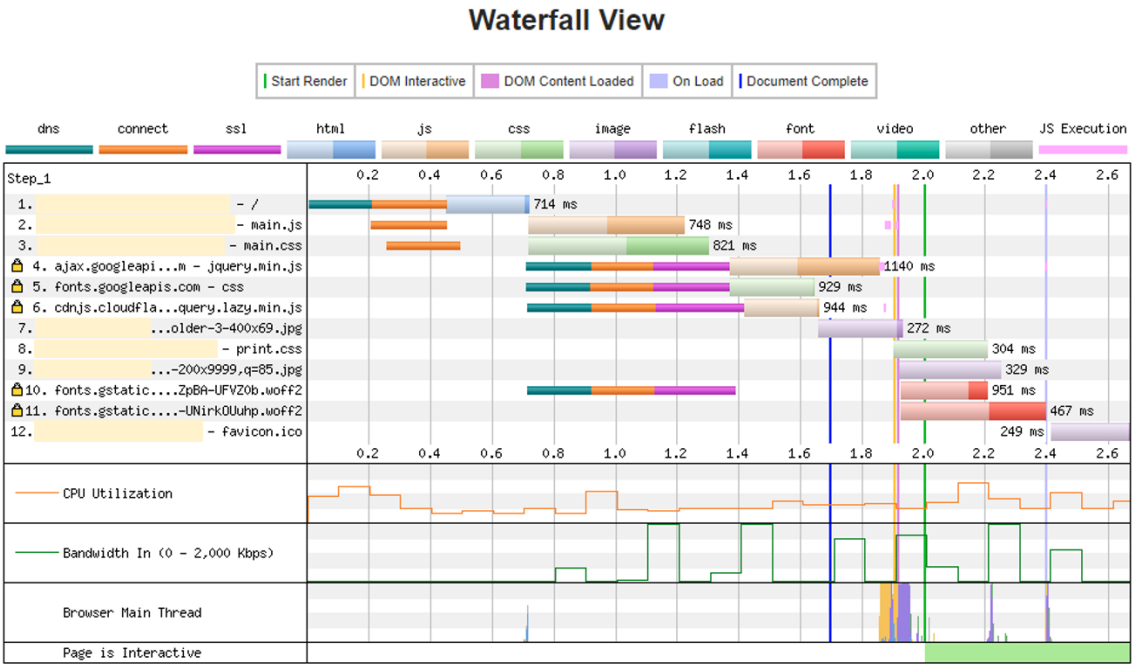


Kuva 34. Vesiputousmalli sivulatauksesta ilman ennenaikaista palvelimelle yhdistämistä.

Ilman ennenaikaista palvelimelle yhdistämistä suoritetun testin vesiputousmallista huomataan, kuinka selain suorittaa yhdistämisen vasta silloin, kun tiedoston lataaminen aloitettaisiin. Ensimmäiseen testiin verrattuna DOM Content Loaded -tapahtumaa se hidasti 227 millisekuntia. Sivulatauksen hidastuminen olisi kuitenkin paljon suurempaa sivustolla, jossa ladataan useampia kriittisiä tiedostoja tai käytetään useampaa CDN-palvelinta. Tämä sen vuoksi, että HTTP/1- tai

HTTP/1.1-protokollaa käytettäessä selain asettaa tietyt rajoitukset samanaikaisten yhteyksien maksimimäärälle. Kuitenkin esimerkkisivustolla ladattavien tiedostojen määrä on niin pieni, minkä vuoksi yhtäaikaisten yhteyksien maksimimäärää ei saavuteta, jolloin ennenaikaisen palvelimelle yhdistämisen vaikutus on pienempi. Ennenaikaisen yhdistämisen tekeminen sivustolle on niin helppoa, etten näe syytä minkä vuoksi sitä ei tehtäisi, varsinkin kun sille ei tarvitse tehdä varmistusta vanhempia selaimia varten.

Seuraava testi suoritetaan olemalla käyttämättä defer-ominaisuutta JavaScript-tiedostokutsuissa. Edelleen kyseessä on sama yhteysnopeus sekä -viive, ja ennenaikainen palvelimelle yhdistäminen on kytketty päälle, jotta tulos on vertailukelpoinen ensimmäisen testin kanssa.



Kuva 35. Esimerkkisivun sivulatauksen vesiputousmalli ilman defer-ominaisuutta JavaScript-tiedostokutsuissa.

Vesiputousmallista nähdään, kuinka ilman defer-ominaisuutta kuvia tai fontteja ei ladata samanaikaisesti JavaScript-tiedostojen kanssa. Oikeastaan tässä testituloksessa deferin pois ottaminen nopeutti DOM Content Loaded -tapahtuman aikaa, mutta samalla ensimmäisen merkityksellisen piirron aika lykkääntyi noin 500 millisekuntia DOM Content Loaded -tapahtuman ajasta. Tällöin käyttäjälle ei piirretty ruudulle mitään tekstisisältöä, koska fonttien lataaminen oli vielä kesken.

Deferin pois ottamisen vaikutus olisi vielä suurempi, jos esimerkkisivustolla olisi enemmän JavaScript-tiedostokutsuja. Nyt kaikki vain yleiset kirjastot ladataan erikseen CDN:stä ja loput tiedostot ovat yhdistetty. Lopullinen aika ensimmäiselle merkittävälle piirrolle oli 2431 millisekuntia sivulle siirtymisestä.

5.1.6 WebPagetest-palvelun sivulataustestien tulokset eri verkkoja käytettäessä

Testasin WebPagetest-palvelulla esimerkkisivuston etusivun latausnopeutta palvelun ehdottamilla verkon nopeuksilla ja viiveillä, jotta tuloksista saadaan mahdollisimman realistiset. Kaikki testit tehtiin Amsterdamin palvelimelta ja käyttäen Google Chrome -selainta. Alla olevassa taulukossa listataan simuloitavan yhteyden nimi, verkon latausnopeus, lähetysnopeus, viive sekä sivulatauksen DOM Content Loaded -tapahtuman sekä ensimmäisen merkityksellisen piirron ajat. On kuitenkin hyvä huomioida, että WebPagetest-palvelun valmiit verkon simulaationopeudet eivät ole Suomen mittakaavassa samanlaisia, esimerkiksi Suomessa 3G Fast -verkon latausnopeus olisi huomattavasti nopeampi kuin 1.6 Mb/s.

Taulukko 1. WebPagetest-palvelun raportoimat testaustulokset esimerkkisivuston etusivun sivulatauksesta eri verkkoyhteyksillä.

Yhteyden nimi	Latausnopeus (Mb/s)	Lähetysnopeus (Mb/s)	Viive (ms)	DOM Content Loaded (ms)	Ensimmäinen merkityksellinen piirto (ms)
FIOS	20	5	4	331	404
LTE	12	12	70	885	831
Cable	5	1	28	598	587
3G Fast	1,6	0,768	150	1940	1829
2G	0,280	0,256	800	9913	9921

Tulokset ovat hyviä ottaen huomioon, että kaikki sivulataukset olivat tehty täysin puhtaalla selaimella ilman välimuistia. Myös palvelimen sijainti vaikuttaa tulok-

seen, sillä Suomesta tehdyllä sivustovierailulla viive olisi pienempi. Erityisesti viiveen vaikutuksen huomaa hyvin vertailtaessa LTE ja Cable -yhteyksien latausaikoja. Huomattavasti hitaammalla mutta pienempiviiveisellä Cable-yhteydellä DOM Content Loaded -tapahtuma oli melkein 300 millisekuntia nopeampi kuin LTE-yhteydellä.

Tuloksista nähdään myös, kuinka ensimmäinen merkityksellinen piirto tapahtui usein hieman ennen DOM Content Loaded -tapahtumaa. Tähän syynä on todennäköisesti se, että selain aloitti jo piirtämään kuvia ja CSS-määrityksissä olevia värejä ennen kuin fontit olivat valmiita. Niinpä yksi parannusehdotus olisi käyttää fonttien viivelataamista, mutta sitä varten asiakkaan haluama fontti pitäisi olla tiedossa, sillä väliaikaisesti näkyvästä fontista tulisi saada mahdollisimman samannäköinen sujuvaa fontin vaihtoa varten. Kuitenkin oikeassa sivunlataustilanteessa on melko todennäköistä, että suosittuja Google Fonts -palvelun fontteja käytettäessä fontit ovat jo ladattuina käyttäjän selaimen välimuistissa.

5.2 Testaus käyttäen Googlen Lighthousea

Google Chromen kehittäjätyökaluista löytyvä Audits-paneeli sisältää verkkosivun arviointiin tarkoitetun testin, joka käyttää testaukseen ja arviointiin Googlen Lighthouse-palvelua. Lighthouse voi arvioida sivun suorituskykyä, saavutettavuutta, hakukoneoptimointia, parhaiden käytäntöjen käyttöä sekä PWA (Progressive Web App) -ominaisuuksia. Testauksen aikana käytetään rajoitettua 3G-yhteyttä sekä prosessoritehoa, mikä simuloi mobiililaitteen laitteistoa. Testauksen jälkeen palvelu luo raportin sivusta, joka sisältää pisteytyksen testatuille osa-alueille sekä parannusehdotuksia. Esimerkkisivuston etusivulle tehtävässä testissä jätetään PWA-ominaisuuksien testaus pois, sillä sivusto ei ole tarkoitus olla Progressive Web App. Sivuston tekstien ja tekstien taustojen värit asetetaan sivuston muuttujat -tiedostosta sekä elementtien asetuksista saavutettaviksi, sillä lopullisessa tuotteessa värien asettaminen on asiakkaan sivuston rakentajan sekä asiakkaan itsensä vastuulla.

Kuva 36. Lighthouse-testin antama esimerkkisivun etusivun pisteytys.

Suorituskyvyn pisteytys on 92, mikä on hyvä tulos. Asioita, joita raportti ehdottaa on kriittisten CSS-määrien vähentäminen. Täysien pisteiden saavuttamiseen vaaditaan melkein täydellistä optimointia, mikä on ulkoisia CSS-kirjastoja käyttäessä vaikeampaa. Esimerkkisivusto käyttää Bootstrapia, joka on karsittu ja asetettu yhteen CSS-tiedostoon. Tällöin niin sanottuja above the fold -tyylejä ei ole eritelty omaan tiedostoon muun muassa siksi, koska sivuston rakentamiseen käytettiin atomisia luokkia. Oikeaoppisesti vain ensimmäisessä näkymässä käytetyt luokat tulisi asettaa head-osiossa olevaan CSS-tiedostoon, mutta tällöin sivustopohjasta luodun asiakkaan sivuston muokkaus vaikeutuisi. Esimerkiksi jos asiakkaan sivustolle haluttaisiin muuttaa ylätunnisteen ulkoasua, pitäisi head-osion CSS-tiedostoa muokata samalla kun ylätunnisteeseen lisätään atomisia luokkia. Jos asiakkaan sivuston rakentaja sattuisi unohtamaan CSS-tiedoston muokkaamisen, tuloksena olisi sivulatauksen yhteydessä pitkään näkyvät väärät tyylit sekä sisällön hyppiminen. Lighthouse-testin raportti ehdottaa myös parannusta kuvien käyttöön käyttämällä webp-kuvia. Sivustolla käytetään kaikkien staattisten kuvien sijaan CSS-määrittelyitä ja SVG-kuvia, mutta lopulta sisällönsyöttöjärjestelmässä asiakas on vastuussa muiden kuvien asettamisesta sivustolle.

Saavutettavuuden pistemäärä oli täydet pisteet eli 100. Kuitenkin Lighthousen kaltaiset testit eivät korvaa täysin manuaalisia testejä. Automatisoiduissa testeissä arvioinnissa ei voida tietää mitkä elementit tarvitsevat esimerkiksi aria-label-ominaisuutta, tai onko navigaation ohittamislinkkiä käytössä. Lighthouse-testin arviointi perustuu muun muassa olemassa olevien aria-ominaisuuksien ja kuvien alt-tekstien tarkistukseen sekä tekstin kontrastin saavutettavuuteen. Vaikkei Lighthouse voi kertoa onko saavutettavuus täydellistä, hyvät pisteet tarkoittavat, että saavutettavuutta on kuitenkin mietitty sivustoa rakentaessa.

Parhaiden käytäntöjen käyttö jää 75 pisteeseen. Asiat, mihin Lighthouse-testi kertoo parannusehdotuksia ovat HTTPS ja HTTP/2 -protokollien käyttämättömyys, manifest.json-tiedoston short_name-ominaisuuden puuttuminen sekä kuvien väärän kuvasuhteen käyttäminen. Sivustopohjaa kopioidessa asiakkaan oikeaan sivustoon asennetaan SSL-sertifikaatti, jolloin HTTPS-ilmoitus korjataan. Samoin korjaantuu myös manifest.json-tiedoston short_name, kun kopioidun sivuston rakentaja käy asettamassa siihen arvon asiakkaan yrityksen nimen mukaan. Väärä kuvasuhdeilmoitus tulee puolestaan tekemästäni taustakuvan scrset-toiminnallisuuden mahdollistavasta komponentista. Siinä lähdekuva on pakko olla piilottamatta display:none -määrityksellä, jotta selain käyttää sen srcset-ominaisuutta. Kuva kooltaan 1x1 pikseliä, ja se on piilossa käyttäjiltä ja ruudunlukijalta muilla tavoin, jolloin Lighthouse ei tunnista sitä piilotetuksi, vaan ehdottaa asiaan korjausta. Lopulliseen asiakkaan sivustoon pistelukema nousee siis yli 90 luokkaan, jolloin silloinkin kuvasuhdeilmoitus on väärästä aiheesta.

Hakukoneoptimoinnin tulos on täydet pisteet eli 100. Hakukoneoptimointiin saa pisteitä esimerkiksi käyttämällä HTML-dokumentin metatageja sekä linkkien ominaisuuksia oikein.

5.3 Saavutettavuustestaus

Esimerkkisivuston saavutettavuutta testataan ensimmäiseksi selaamalla sivustoa ja käyttämällä sen toiminnallisuuksia ainoastaan näppäimistöllä. Erityiset onnistumiset sekä kaikki epäonnistumiset raportoidaan testatessa sivustoa. Näppäimistötestaamisen lisäksi sivustoa testataan käyttäen ruudunlukuohjelmaa. Testien aikana käytettävä selain on Google Chrome (versio 65.0).

5.3.1 Näppäimistökäytön testaus

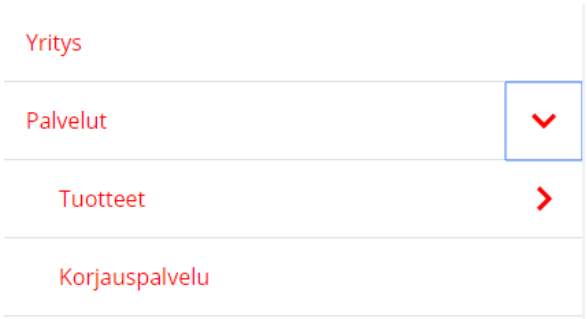
Heti ensimmäisenä kohdistettavana elementtinä sivustolla on ohita navigaatio -linkki, jolla pääsee suoraan sivun main-osioon. Linkkiä enter-näppäimellä paina-

malla näkymä ja kohdistus siirtyy main-osioon, jolloin seuraava elementti kohdistusjärjestyksessä on main-osion ensimmäinen kohdistettava elementti. Sivuston kaikissa kohdistettavissa elementeissä näkyy ääriviivat, eli kohdistuksesta kertova elementtiä ympäröivä kehys.

Kuva 37. Näppäimistöllä kohdistettu sähköpostilinkki sivuston ylätunnisteessa.

Myös sivuston hover-toiminnolla toimivaa valikkoa voi selata näppäimistöllä. Siirtyessä alavalikolliseen linkkiin, käyttäjälle näytetään linkin alasivut. Seuraavana kohdistuksessa on seuraavan juuritason linkin sijaan alavalikon ensimmäinen linkki. Samoin käy alavalikollisen alasivun kohdalla, eli navigaation jokaiselle sivulle pääsee, vaikka sen sivutaso olisi suurempi kuin kaksi. Kuitenkin vieläkin paremman käytettävyyden saisi, jos käyttäjä saisi käyttää valikoissa esimerkiksi nuolinäppäimiä, mutta ainakin tämän kokoisella sivustolla linkkien selaaminen ei ole niin suuri ongelma, varsinkaan kun sivustolla on olemassa ohita navigaatio -linkki.

Myös mobiilivalikon aukaiseminen onnistuu pelkkää näppäimistöä käyttäen. Aukaisun jälkeen kohdistusjärjestyksessä seuraavana on valikon ensimmäinen linkki. Linkkejä voi selata ja alavalikot saa auki aukaisupainikkeesta, jolloin seuraava kohdistettava elementti on alavalikon ensimmäinen linkki.



Kuva 38. Mobiilivalikko, jossa käyttäjä on juuri avannut alavalikon käyttäen näppäimistöä.

Etusivun nostoelementtien selaaminen onnistuu vaivatta ja kohdistettuna olevalla linkillisellä nostolla on ympärillään kohdistuksesta kertovat ääriviivat. Myös lomakkeen täyttäminen ja lähettäminen onnistuu näppäimistöllä. Virhetilanteessa

kohdistus viedään automaattisesti virheelliseen kenttään. Onnistuneen lähetyksen jälkeen sivua ei päivitetä tai vieritetä, ja kohdistus jatkaa lomakkeen jälkeen olevasta elementistä vaikka lähetetty lomake korvattiinkin kiitosviestillä. Linkillisiä logorivin elementtejä voi selata läpi tai klikata näppäimistöä käyttäen. Samoin toimii alatunnisteen yhteystieto- ja somelinkit.

Alasivuilla käyttäjä voi selata tai painaa murupolun linkkejä normaalisti, ja sivuvalikot toimivat samalla tapaa kuin mobiilivalikkokin. Yhteystiedot-sivulla yrityksen ja henkilöstölistauksen sähköpostit ja puhelinnumerot ovat linkkejä, joita voi myös selata sarkainnäppäintä käyttäen. Linkkejä painamalla lähetetään sähköpostia tai soitetaan henkilölle käyttöjärjestelmän määrittämää oletussovellusta käyttäen.

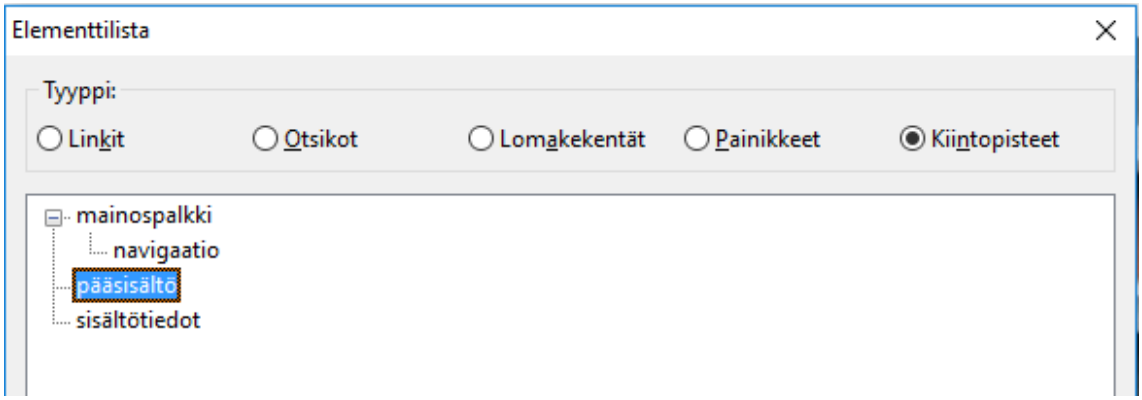
Koko sivuston läpikäytyä testin lopputuloksena on onnistunut käyttäjäkokemus myös näppäimistökäyttäjälle. Ainoana huomiona on työpöytäkoolla näytettävä hover-valikko, jonka näppäimistökäytön voisi tehdä vieläkin sulavammaksi antamalla käyttäjälle mahdollisuus selata alavalikoiden linkkejä esimerkiksi nuolinäppäimiä käyttäen.

5.3.2 Ruudunlukijan käytön testaus

Esimerkkisivuston ruudunlukuohjelmakäytettävyyttä testataan käyttäen NV Access:in kehittämää NVDA-ruudunlukuohjelmaa. NVDA on ilmainen ruudunlukuohjelma, joka toimii Microsoft Windows -käyttöjärjestelmällä. Testauksessa käyttöjärjestelmänä toimii Windows 10 64bit.

Sivustolla ollessaan NVDA:lla saa esille elementtilistan, joka listaa tämän hetkisellä sivulla käytettävät elementit joihin käyttäjä voisi haluta siirtyä. Elementtilistassa on kategorioitu erikseen linkit, otsikot, lomakekentät, painikkeet ja kiintopisteet. Kategorisointi onnistuu siksi, koska sivustolla on käytetty semanttisia elementtejä esimerkiksi otsikoissa ja napeissa. Yleiskuvan sivusta antaa kiintopisteet-kategoria, joka luodaan roolien sekä semanttisten rakenne-elementtien avulla. Kiintopistelista sisältää mainospalkin eli ylätunnisteen, pääsisällön sekä sisältötiedot eli alatunnisteen. Mainospalkin sisällä listataan oikeanlaisesti vain

yksi navigaatio, vaikka rakenteellisesti ylätunnisteessa on sekä mobiilinavigaatio että tavallinen navigaatio, joita näytetään käyttäjälle selainikkunan leveyden mukaan. Kiintopisteen valitsemalla käyttäjä pääsee suoraan sen alkuun ja ruudunlukija alkaa lukemaan siitä ensimmäistä elementtiä.



Kuva 38. NVDA-ruudunlukuohjelman elementtilistanäkymä.

Sivustoa selatessa ruudunlukija ilmoittaa, jos käyttäjä menee semanttisen elementin sisään tai lähtee sieltä. Esimerkiksi nav-elementtiin mentäessä käyttäjälle kerrotaan kyseessä olevan navigaatio, joka on samalla myös kiintopiste. Navigaation linkkejä selatessa ruudunlukija kertoo linkin nimen ja sen olevan linkki. Alavalikkoon mentäessä ruudunlukija kertoo kyseessä olevan luettelo, ilmoittaa lueteltavien linkkien määrän sekä aloittaa lukemaan ensimmäistä linkkiä. Luetteloilmoitusta ei tulisi automaattisesti, jos navigaation rakenteessa ei käytettäisi semanttista ul-listaelementtiä.

Esimerkiksi nostolinkkien otsikon kohdalla ruudunlukija kertoo otsikon tekstin lisäksi myös sen tason. Otsikoiden eri tasojen käyttäminen on hyödyllistä, sillä ruudunlukijalla voi kuunnella seuraavat otsikot tason mukaan, mikä helpottaa sivun sisällön hahmottamista. Nostolinkki-elementeissä olevia koristeellisia kuvia ei huomioida, vaan käyttäjälle luetaan nostolinkin teksti sekä kerrotaan kyseessä olevan linkki.

Ruudunlukija ilmoittaa käyttäjälle lomakkeen syöttökentistä kertomalla sen tyypin ja nimikkeen. Pakollisten kenttien tapauksessa kerrotaan syöttökentän olevan pakollinen ja ilmoitetaan samalla, jos kenttään syötetty arvo ei ole oikeanlainen. Valintaruutu ja -nappi -ryhmistä ruudunlukija ilmoittaa käyttäjälle äänimerkillä

aina kun sellaiseen mennään tai siitä lähdetään. Lähetä-painikkeen kohdalla ruudunlukija lukee painikkeen tekstin ja kertoo kyseessä olevan painike. Virheellistä lomaketta lähettäessä kohdistus siirretään virheelliseen kenttään ja ruudunlukija lukee kyseisen kentän tyypin ja nimen sekä kertoo sen olevan virheellinen. Palvelinpuolella huomatun täyttövirheen tapauksessa käyttäjälle ilmoitetaan virheviesti ja virheellinen kenttä. Onnistuneen lomakkeenlähetyksen jälkeen käyttäjälle luetaan kiitosviesti vaihtamatta kohdistuspaikkaa sivulla.

Esimerkiksi logorivin logot ja alatunnisteen somelinkit ruudunlukuohjelma lukee eri tavalla. Käyttäjälle ilmoitetaan kyseessä olevan graafinen linkki, ja linkin nimi otetaan joko kuvan vaihtoehtoisesta tekstistä tai somelinkkien tapauksessa aria-label-arvosta. Somelinkeillä ei voida käyttää kuvan vaihtoehtoista tekstiä, sillä ne ovat asetettu sivustolle SVG-elementteinä.

Tuotelistauksessa ruudunlukijaohjelma lukee tuotteiden nimet, kuvaukset ja hinnat normaalisti. Eritellyt hinnat luetaan kerralla, esimerkiksi: ”Opiskelijat 40 euroa”. Yhteystiedot-sivun yrityksen yhteystiedot ruudunlukijaohjelma lukee muuten hyvin, mutta tekstieditorissa tehdyt peräkkäiset rivivälit eli br-elementit luetaan myös käyttäjälle sanomalla ”Tyhjä”. Henkilöstölistauksessa kaikkia henkilön tietoja ei lueta kerralla, vaan käyttäjä voi lukea jokaisen tietorivin yksitellen.

Ruudunlukijan käytön testauksen tulokset ovat positiivisia. Kaikki tieto on käyttäjän saavutettavissa ja sivuston semanttiset elementit paransivat käyttäjäkokemusta huomattavasti. Parannettavaa olisi vielä piilottaa tekstieditoreissa asetetut peräkkäiset rivinvaihdot ruudunlukijoilta, jolloin niitä ei luettaisi käyttäjälle ollenkaan. Toinen huomio on Hyphenator.js-kirjaston tekemä pitkien otsikoiden tavutus. Tavutus toimii asettamalla sanalle automaattisesti visuaalisesti piilotetut tavuviivat, joista sana tavutetaan sen muuten mennessä selainikkunan yli leveyssuunnassa. Ainakin NVDA-ruudunlukijaohjelma lukee automaattisesti tavutetut sanat käyttäjälle tavutettuina, mikä ei kuulosta niin sujuvalta kuin tavallinen ruudunlukijan lukema teksti. Ratkaisuna tähän voisi olla esimerkiksi vain ruudunlukijoille näkyvä nappi, jolla tavutuksen saa pois käytöstä, ja jossa selitettäisiin miksi niin haluttaisiin tehdä.

```
<h1 class="hyphenate">Lä&shy;he&shy;tä
tar&shy;jous&shy;pyyn&shy;tö</h1> == $0
```

Kuva 39. Hyphenator.js-kirjaston tekemä sivulla näkymätön tavutus otsikolle.

6 Yhteenveto

Verkkosivuston käytettävyyttä pidetään usein itsestäänselvyytenä, eikä sitä välttämättä huomata, ennen kuin se on puutteellista. Verkkosivuston rakentajan tulee kuitenkin huomioida käytettävyys koko rakentamisprosessin ajan, aina suunnittelun hyväksymisestä sivuston testaukseen asti. Helppokäyttöinen sivusto saadaan aikaiseksi pitämällä sivuston rakenne yksinkertaisena, käyttämällä käyttäjille tuttuja sivustoelementtejä oikein sekä huomioimalla erilaiset laitteet, joilla sivustoa voidaan selata. Tällöin sivusto on käyttäjälle tuttu ja looginen selata, sekä se on käytettävissä useilla eri laitteilla ilman käyttäjäkokemuksen huononemista.

Loogisuuden ja hyvän laitetuen lisäksi verkkosivuston käytettävyyteen liittyy olennaisesti saavutettavuus. Verkkosivuston saavutettavuus tarkoittaa sitä, että kaikki käyttäjät voivat selata verkkosivustoa niin luontevasti kuin heidän mahdollisten esteellisyyksiensä puitteissa on vain mahdollista. Esteellisyyden omaava käyttäjä voi olla esimerkiksi näppäimistökäyttäjä, jolla on käden motorisia käyttövaikeuksia, tai heikkonäköinen ruudunlukijaohjelman käyttäjä. Selain antaa hyvät perusmahdollisuudet saavutettavuuden huomioimiseen sivustolla, jolloin verkkosivuston rakentajan vastuulle jää osata hyödyntää niitä esimerkiksi käyttämällä sivustolla paljon semanttisia elementtejä sekä osata antaa käyttäjälle tietoa muutenkin kuin pelkällä visuaalisella palautteella.

Käyttäjäkokemuksen kannalta myös verkkosivuston suorituskykyoptimointi on oleellista. Verkkosivuston rakentaja voi helposti unohtaa suorituskykyoptimoinnin tärkeyden, sillä usein sivustoa rakennettaessa käytetään hyvää laitteistoa ja nopeaa verkkoyhteyttä. Todellisuudessa verkkoa selataan myös käyttäen hidasta verkkoyhteyttä, jolloin optimoimattomilla sivustoilla käyttäjä ei välttämättä malta

odottaa sivuston latautumista, vaan hakee haluamansa tiedon muualta. Tällöin verkkosivusto on kyseisen käyttäjän osalta täysin epäonnistunut tehtävässään, oli se sitten olla tietolähteenä käyttäjille tai omien palveluiden mainostaminen.

Sivulatausta optimoimalla käyttäjä pääsee lukemaan sivun sisältöä nopeasti sen sijaan, että joutuisi katsomaan pitkään valkoista ruutua ja latausikonia. Käyttäjäkokemuksen kannalta tärkeimpiin optimointeihin kuuluu sivun renderöinnin estävien kriittisten tiedostojen määrän ja tiedostokoon pienentäminen, mikä onnistuu esimerkiksi yhdistelemällä ja minifoimalla käytettävät CSS sekä JavaScript -tiedostot. Ladattavan sisällön määrää saadaan parhaiten pienennettyä optimoimalla sivustolla käytettävät kuvat, sillä usein kuvat vievät reilusti suurimman osan sivun latausmäärästä. Tämän vuoksi oikeankokoisten kuvien käyttö ja ylimääräisten kuvatietojen poistaminen pienentää huomattavasti käyttäjän laitteelle ladattavaa turhaa sisältöä.

Esimerkkisivustoksi tehdystä sivustopohjasta tuli helposti muokattava sekä sivuston rakentajalle että asiakkaalle itselleen. Sivustopohjassa käytettävät värit ja perustyylit ovat muokattavissa yhdestä muuttujia sisältävästä tiedostosta, ja asiakas voi määritellä sivurakenteen ja sisällön itse Koodiviidakko Oy:n Sivuviidakko-sisällönhallintajärjestelmällä. Toimiva asiakkaan verkkosivusto saadaan kopioimalla sivupohjan asennus, muokkaamalla sivuston tyylit asiakkaan haluamiksi sekä asentamalla SSL-sertifikaatti. Sivustopohjassa on valmiina pienyrityksien yleisesti vaatimat sivut ja elementit, joilla saa aikaiseksi toimivan ja sisältörikkaan sivuston. Testauksien mukaan esimerkkisivusto on helposti käytettävissä myös pelkästään näppäimistöä käyttäen tai ruudunlukijan kanssa. Myös erilaisilla verkkoyhteyksillä tehtyjen sivulatauksien latausaikatulokset olivat hyviä, mikä tarkoittaa nopeasti toimivaa sivustoa hitaammillakin verkkoyhteyksillä.

Lähteet

Internet World Stats. 2017. World Internet Users and 2017 Population Stats. http://www.internetworldstats.com/stats.htm. 12.9.2017

U.S. Department of Health and Human Services. 2017. User Experience Basics. https://www.usability.gov/what-and-why/user-experience.html. 1.11.2017

Customer Experience Matters. 2017. The Ultimate CX Infographic, 2017. https://experiencematters.blog/wp-content/uploads/2017/10/CXMatters_2017.pdf. 5.11.2017

EUR-lex. 2016. Directive of the European Parliament and of The Council on the accessibility of the websites and mobile applications of public sector bodies. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CONSIL:PE_38_2016_INIT. 5.11.2017.

Springer, T. 2017. EU Directive 2016/2102/EU: What Is Covered? Level Access. https://www.levelaccess.com/eu-directive-what-is-covered. 5.11.2017.

World Health Organization. 2016. Disability and health fact sheet. http://www.who.int/mediacentre/factsheets/fs352/en. 23.11.2017.

World Wide Web Consortium. 2017. Web Content Accessibility Guidelines (WCAG) 2.1. Web Accessibility Initiative. https://www.w3.org/TR/WCAG21. 27.11.2017

Web Accessibility In Mind. 2017. Screen Reader User Survey #7 Results. https://webaim.org/projects/screenreadersurvey7. 04.01.2018.

Blockmetry. 2017. What percentage of browsers with javascript disabled? 13.2.2017. https://blockmetry.com/blog/javascript-disabled/. 16.1.2018.

DoubleClick by Google. 2016. The need for mobile speed: How mobile latency impacts publisher revenue. https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/. 17.1.2018.

Can I Use… 2018a. Defer attribute for external scripts. https://caniuse.com/#feat=script-defer. 30.1.2018.

HTTP Archive. 2018. Stats January 15. 2018. http://httparchive.org/compare.php?&r2=Jan%2015%202018&s2=All. 5.2.2018.

Google Developers. 2017. Lossless and Transparency Encoding in WebP. https://developers.google.com/speed/webp/docs/webp_lossless_alpha_study. 6.2.2018.

Google Developers. 2016. WebP Compression Study. https://developers.google.com/speed/webp/docs/webp_study. 6.2.2018.

Can I Use… 2018b. WebO image format. https://caniuse.com/#feat=webp. 6.2.2018.

StatCounter GlobalStats. 2018. Desktop Screen Resolution Stats Worldwide. http://gs.statcounter.com/screen-resolution-stats/desktop/worldwide. 19.03.2018.

Akamai. 2017. Akamai's state of the internet Q1 2017 executive summary. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-executive-summary.pdf. 21.3.2018.

Kuvalähteet

Kuva 7. Ire Aderinokun. 2017. Asynchronous vs Deferred JavaScript. https://bitsofco.de/async-vs-defer/. 2.2.2018.

Kuva 8. Ire Aderinokun. 2017. Asynchronous vs Deferred JavaScript. https://bitsofco.de/async-vs-defer. 2.2.2018.

Kuva 9. Ire Aderinokun. 2017. Asynchronous vs Deferred JavaScript. https://bitsofco.de/async-vs-defer. 2.2.2018.

Sivuston muuttujat -tiedosto

```
// Set basic site properties here
// For more specific settings modify _bootstrap-variables.scss


// Theme variables
// ================================================================================
$site-color-primary:                #ff3434;
$site-color-secondary:              #546e7a;
$site-color-tertiary:               #fff;
$site-text-color:                   #212529;
$site-fontfamily:                   'Open Sans', sans-serif;



// Body
// ================================================================================
$site-body-width:                   auto;
$site-html-background:              #eee;
$site-mobile-centered-text:         true;



// Headings
// ================================================================================
$site-headings-color:               $site-text-color;
$site-headings-fontweight:          400;



// Links
// ================================================================================
$site-link-color:                   $site-color-primary;
$site-link-hover-decoration:        none;



// Tables
// ================================================================================
$site-table-heading-background:     $site-color-secondary;
$site-table-heading-color:          #fff;
```

Sivuston muuttujat -tiedosto

```
// Header & Navigation
// ====================================================================================

// Header
$site-header-border-color:          #eee;

// Headerbar
$site-headerbar-visible:            true;
$site-headerbar-background:         $site-color-primary;
$site-headerbar-color:              #fff;
// Desktop nav

$site-nav-navlink-color:            $site-color-primary;
$site-nav-highlight-color:          $site-color-primary;

// Mobile nav
$site-mobilenav-toggler-color:      $site-color-primary;
$site-mobilenav-navlink-color:      $site-nav-navlink-color;

// Side nav
$site-sidenav-navlink-color:        $site-color-secondary;
$site-sidenav-heading-background:   $site-color-secondary;
$site-sidenav-heading-color:        #fff;




// Footer
// ====================================================================================
$site-footer-background:            #1b1b1b;
$site-footer-color:                 #fff;
$site-footer-link-color:            $site-link-color;
$site-footer-link-hover-color:      rgba($site-footer-link-color, 0.6);
$site-footer-link-hover-decoration: $site-link-hover-decoration;
```

JavaScript-koodi taustakuvan srcset-toiminnallisuuden saavuttamiseksi

```
import debounce from '../../js/custom/debounce';

let lastWindowWidth = $(window).width();


/**
 * Sets background-image of parent to match
 * child img currentSrc. This way srcset-style
 * behaviour can be used for background-images
 */
function setParentBackground() {
    const $img = $(this);
    const $target = $img.parent();
    // Use src if srcset is not supported
    const src = $img.get(0).currentSrc || $img.get(0).src;
    // Prevent flash of white background with data URI
    if (src.lastIndexOf('data:', 0) === 0) return;
    $target.css('background-image', `url('${src}')`);
}


/**
 * Force srcset checkup if window is resized wider.
 * Fixes Safari's srcset behaviour, which doesn't
 * check for new image source on window resize
 */
const handleWindowResize = debounce(() => {
    const windowWidth = $(window).width();
    if (windowWidth <= lastWindowWidth) return;
    lastWindowWidth = windowWidth;
    reSrcImages();
}, 250);


/**
 * Trigger broser's src recalculation from
 * srcset by setting image src to be itself
 */
function reSrcImages() {
    const $images = $('.set-parent-bg');
    $images.each(function triggerSrcCheck() {
        const $img = $(this);
        const src = $img.attr('src');
        $img.attr('src', src);
    });
}


$('.set-parent-bg').on('load', setParentBackground);

// Set listener for window resize event if images exist and browser supports srcset
if ($('.set-parent-bg').get(0) && typeof $('.set-parent-bg').get(0).currentSrc !== 'undefined') {
    $(window).on('resize', handleWindowResize);
}
```

Verkon nopeuden vaihtelun vaikutus sivulatauksen eri vaiheiden valmistumiseen, kaikki testitulokset eriteltynä.

Latausnopeus (Mb/s)	Lähetysnopeus (Mb/s)	DomContentLoaded (ms)	Load (ms)	Finished (ms)
50	10	169	215	298
50	10	208	272	377
50	10	160	210	311
50	10	183	243	375
50	10	176	245	326
20	4	181	230	384
20	4	192	253	408
20	4	204	250	400
20	4	195	260	410
20	4	182	248	399
10	2	214	267	547
10	2	235	294	574
10	2	231	293	574
10	2	234	290	569
10	2	205	265	547
5	1	274	305	844
5	1	264	296	834
5	1	268	307	845
5	1	270	324	862
5	1	310	373	911
2	0.4	602	614	1940
2	0.4	703	710	2030
2	0.4	648	663	1990
2	0.4	617	633	1960
2	0.4	646	661	1980

Verkon nopeuden vaihtelun vaikutus sivulatauksen eri vaiheiden valmistumiseen, kaikki testitulokset eriteltynä.

1	0.2	1390	1410	4030
1	0.2	1240	1250	3870
1	0.2	1320	1350	3970
1	0.2	1480	1490	4110
1	0.2	1370	1390	4010
0.5	0.1	2770	2780	8000
0.5	0.1	2650	2660	7890
0.5	0.1	2480	2480	7720
0.5	0.1	2290	2310	7540
0.5	0.1	2720	2730	7970
0.2	0.04	6860	6870	19900
0.2	0.04	7020	7040	20110
0.2	0.04	6920	6930	19960
0.2	0.04	6910	6920	19950
0.2	0.04	7130	7140	20170

Verkon viiveen vaihtelun vaikutus sivulatauksen eri vaiheiden valmistumiseen, kaikki testitulokset eriteltynä.

Latausnopeus (Mb/s)	Lähetysnopeus (Mb/s)	Viive (ms)	DomContent-Loaded (ms)	Load (ms)	Finished (ms)
5	1	40	358	394	956
5	1	40	332	371	933
5	1	40	361	401	964
5	1	40	325	370	932
5	1	40	342	389	951
5	1	80	458	509	1110
5	1	80	476	515	1120
5	1	80	471	524	1130
5	1	80	468	510	1110
5	1	80	441	492	1090
5	1	120	605	635	1280
5	1	120	584	619	1260
5	1	120	565	606	1250
5	1	120	587	635	1280
5	1	120	558	618	1260
5	1	160	698	737	1420
5	1	160	700	749	1430
5	1	160	700	746	1430
5	1	160	689	750	1430
5	1	160	712	758	1440
5	1	200	824	863	1590
5	1	200	829	865	1590
5	1	200	795	857	1580
5	1	200	841	876	1600
5	1	200	837	878	1610

Verkon viiveen vaihtelun vaikutus sivulatauksen eri vaiheiden valmistumiseen, kaikki testitulokset eriteltynä.

5	1	240	948	984	1750
5	1	240	926	992	1750
5	1	240	921	981	1740
5	1	240	955	991	1750
5	1	240	932	968	1730
5	1	280	1020	1110	1920
5	1	280	1010	1080	1880
5	1	280	1060	1100	1900
5	1	280	1060	1120	1920
5	1	280	1050	1100	1900