Tallinna Ülikool

Digitehnoloogiate Instituut

Mobile-first seadmetundlik arendusmeetod

Bakalaureusetöö

Autor: Hainer Savimaa

Juhendaja: Jaagup Kippar

Autor:..... ,, ... ,, 2016

Juhendaja:..... ,, ... ,, 2016

Instituudi direktor:..... ,, ... ,, 2016

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina _____ (sünnikuupäev: _____)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

_____

_____

_____

mille juhendaja on _____,

säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas/Haapsalus/Rakveres/Helsingis, _____

Sisukord

Lühendite ja mõistete sõnastik

CSS *Cascading Stylesheets* laadilehed, mis kirjeldavad HTML dokumentide kuvamist.

desktop-first veebilehe ülesehitusviis, kus kõigepealt antakse kujunduslikud atribuudid CSS failis veebilehe vaatele arvutis ning alles seejärel väiksematele seadmetele.

DOM *Document Object Model* eeskiri, mis määrab, millised atribuudid kuuluvad millise objekti juurde ning määrab kuidas objekte ja atribuute käsitleda.

HTML *HyperText Markup Language* kodeerimissüsteem veebidokumentide loomiseks.

HTTP *HyperText Transfer Protocol* andmevahetusprotokoll, mida kasutatakse internetis dokumentide vahetamiseks.

image sprite mitme erineva pildifaili monteerimine ühte pildifaili, eesmärgiga vähendada HTTP päringute arvu.

inline kood failisisene kood. Näiteks Javascripti kood HTML failis *script* märgendite vahel.

keyframe CSS3 võimalus animatsioonide tegemiseks, kus määratakse animatsiooni sätted erinevatel hetkedel.

max-width CSS3 atribuut, mis lubab elemendil kasvada kuni ette määratud suuruseni. Ülesehitusmeetodi seisukohalt kasutatakse seda *media-query*'des märkides selle ulatuse määratud väärtuseni. Kasutatakse enamasti *desktop-first* arendusmeetodi puhul.

media-query CSS3 võimalus, mis laseb määrata erinevatele seadmetele erinevaid kujundusi. Selle abil on võimalik määrata kujundused vastavalt järgnevatele omadustele: vaateakna laius/kõrgus, ekraani orientatsioon (vertikaalpaigutus/horisontaalpaigutus), resolutsioon.

min-width CSS3 atribuut, mis lubab elemendil kahaneda kuni ette määratud suuruseni. Ülesehitusmeetodi seisukohalt kasutatakse seda *media-query*'des, märkides selle ulatuse alates määratud väätusest. Kasutatakse enamasti *mobile-first* arendusmeetodi puhul.

mobile-first veebilehe ülesehitusviis, kus kõigepealt antakse vaabilehele kujundustlikud atribuudid CSS failis kõige väiksemale seadmele ning seejärel liikudes järjest suurematele seadmetele.

placeholder ehk kohatäide.

Sissejuhatus

Veebilehe laadimiskiirus ning kasutusmugavus on ühed olulisemad aspektid, mida kasutajad veebilehtede külastamisel hindavad. Viimastel aastatel on nutitelefoni kasutajate arv kasvanud ligi 1,8 miljardi kasutajani (Bosomworth, 2015). Samas ei ole selle kasvuga suutnud sammu pidada seadmetundlike veebilehtede koguarv. See on omakorda tekitanud olukorra, kus mobiilsetes seadmetes veebilehte külastavatele kasutajatele esitatakse arvutile mõeldud veebileht, mis tekitab pika laadimisaja ning ebamugava kasutatavuse.

Autorit ajendas käesoleval teemal kirjutama huvi seadmetundlike veebilehtede eesliideste arenduse vastu ning soov uurida *mobile-first* arendusmeetodi kasutamist seadmetundlike veebilehtede arenduses.

Valitud teema aktuaalsus ning ühtlasi ka vajadus seisnebki mobiilsetel seadmetel interneti kasutajate arvu jätkuvas suurenemises ning nende seadmete jaoks kohandatud veebilehtede arvu väikeses osakaalus kogu veebilehtede arvust.

Käesoleva bakalaureusetöö eesmärgiks on uurida, kuidas parendada mobiilsetes seadmetes seadmetundlike veebilehtede kuvamis- ning laadimisaegasid *mobile-first* ülesehitusmeetodi abil, anda ülevaade *mobile-first* meetodi olemusest ning põhimõtetest ning testida selle meetodi mõju veebilehe jõudlusele. Autor koostab eesmärgi saavutamiseks ülevaate veebiarenduse ajaloost ja hetkeseisust; tutvustab *mobile-first* ülesehitust ja põhimõtteid ning testib välja toodud põhimõtteid kolme näitekomponendi peal.

Töö jaguneb viieks osaks, kus esimeses tutvustab autor veebiarenduse ajalugu ning hetkeseisu, samuti annab põhjaliku ülevaate mobiilsete seadmete kasutamisstatistika kohta. Teises osas tutvustab autor *mobile-first* ülesehitusmoodust ning võrdleb seda *desktop-first* moodusega. Käesoleva töö kolmandas osas toob autor välja *mobile-first* meetodi praktilised põhimõtted ning ettepanekud jõudluse parendamiseks, mida autor testib töö neljandas osas, kolme välja mõeldud komponendi peal. Töö viiendas osas koostab autor iga komponendi jõudluse kohta mõõtmised ning analüüsib kolmandas osas toodud ettepanekute mõju veebilehe kuvamis- ning allalaadimisaegadele mobiilses seadmes.

Käesoleva bakalaureusetöö käigus arendatud näitekomponendid asuvad aadressil: http://www.tlu.ee/~hainer/loputoo/

1 Veebiarenduse ajalugu ning hetkeseis

Veebilehed on pidevas muutuses ning arengus. Esimesed veebisaidid ilmusid aastal 1995. Need olid algelised veebilehed, mille ehitamiseks kasutati HTML tabeleid, erinevaid fonte ja pilte. Samal aastal lisandus JavaScript, mis lisas HTML'ile võimekust, andes veebilehtedele interaktiivsuse: näiteks erinevad hüpikaknad ning võimaluse sisu sorteerimiseks. Aasta 1996 tõi endaga uue interaktiivse ning graafilise võimaluse - Flash. Uhkete võimaluste kõrval oli Flashi suurimateks probleemideks suur ressursivajadus ning nõudlus uusima Flash Player'i järele. 1998. aastal sündis uus tehnoloogia, mis võimaldas sisu eraldada kujundusest - CSS (*Cascading Style Sheets*). Kuigi esimestel aastatel ei toetanud kõik veebilehitsejad täielikult CSS'i ning algselt ei olnud see kuigi paindlik, ei kujutata tänapäeval veebilehtede eesliideste arendust ilma selleta ette. Alates 2007. aastast, mil nutitelefonid tulid turule, hakkas tekkima nõudlus veebilehtede korrektsemaks kuvamiseks eri suurustega ekraanidel. Ilmavalgust nägid erinevad veebiraamistikud ning võrestik-süsteemid, millest tänasel päevalgi on relevantsed Bootstrap ning Foundation. Kuigi esimene seadmetundlik (inglise keeles *responsive)* veebileht (www.audi.com) ilmus juba 2001. aastal, defineeriti *Responsive Web Design* (edaspidi RWD) alles 2010. aastal Ethan Macrotte'i poolt ning alates sellest ajast on seda lähenemist järjest enam kasutama hakatud. Enne RWD'd kuvati veebilehed mobiilsetel seadmetel täismahus (nagu suurtel ekraanidel) ning lehe kasutamiseks tuli seda suurendada või vähendada (Ruluks, 2014). RWD'd täiendab 2009. aastal Mike Wroblewski *mobile-first* idee. *Mobile-first* käigus arendatakse kõigepealt mobiilsetele seadmetele mõeldud väikese mahuga ning rohkem sisule orienteeritud veebileht, millele lisatakse ekraani suurenedes ressursinõudlikemaid komponente ja sisu (tekst, suuremad joonised ja pildifailid jne). RWD populaarsuse tõusu taga on ka HTML5 ja CSS3, mis sisaldavad mitmeid uusi funktsioone seadmetundliku veebilehtede tarbeks.

Nutitelefonid ja tahvelarvutid tõid kaasa suure hulga erinevaid ekraanisuuruseid ja resolutsioone. Uute nutitelefonide mudelite ekraanid kasvavad aasta-aastalt: viie aastaga (2007-2012) kasvas keskmine ekraanisuurus kolmelt tollilt neljani. Samas võttis kasv neljalt tollilt viieni aega vaid ligikaudu kaks aastat (2012-2014) (Barredo, 2014). Kuigi uute nutitelefonide mudelite ekraanide keskmine suurus jääb 5 tolli lähedusse, on 2015. aasta 3. kvartalis kõige rohkem ostetud 4,7 tollise ekraaniga nutitelefone (Diaconescu,

2015). 4,7 tollise ekraaniga nutitelefonid on kõige populaarsemad Prantsusmaal, Saksamaal, Itaalias, Suurbritannias, Austraalias, Jaapanis ja USA-s (DeviceAtlas, 2015).

Viimasel viiel aastal on mobiilse seadme kasutamine suure kiirusega kasvanud. Kui aastal 2010 kasutas ligi 800 miljonit inimest mobiilset seadet interneti külastamiseks, siis aastal 2015 juba ligikaudu 1,8 miljardit. 2014. aastal ületas mobiilse seadme kasutajate hulk arvuti kasutajate oma -- ennustatavalt kasvab lähiajal mobiili kasutajate arv veelgi ning vahe arvutikasutajatega jätkab suurenemist (Bosomworth, 2015). Mobiilse seadme kasutamise kasvu näitab hästi ka asjaolu, et 2015. aastal edastas "ainult-mobiilse" kasutajate arv "ainult-arvuti" kasutajaid (Lella, 2015).

Mobiilsete seadmete populaarsuse kasv muudab ka internetikasutust: 2015. aastal ületas USA-s esimest korda mobiilsest seadmest tehtud Google otsingupäringute arv arvutitest tehtud otsingute oma (Sterling, 2015). Lisaks on Google asunud jõulisemalt toetama mobiilset veebikasutust -- alates 2015. aasta aprillist tõstab Google mobiilisõbralike veebilehtede hinnanguid, et neid otsingutulemustes eespool kuvada (Makino&Phan, 2015). Mobiilisõbralikeks veebilehtedeks loetakse neid lehti, mis on tundlikud ekraanisuuruste suhtes ning leht kuvatakse vastavale seadmele mõeldud sätetele.

Mobiilsete seadmete hulga kasvuga kaasneb ka seadme kasutusaja suurenemine: 2015 aastal kasutas keskmine täiskasvanud ameeriklane päevas 5,6 tundi digitaalset meediat, millest 51% toimus mobiilsel seadmel ning 42% arvutis. 2008. aastal oli digitaalsele meediale kulunud aeg päevas 2,7 tundi, millest 2,3 tundi moodustas arvuti kasutamine ning kõigest 0,3 tundi mobiilse seadme kasutamine. Seitsme aasta jooksul on jäänud arvuti kasutamise aeg põhimõtteliselt samale tasemele (tõusnud 0,1 tunni võrra päevas), kogu kasv on tulnud mobiilse seadme kasutamise aja tõusust (Bosomworth, 2015).

Kahjuks pole nutitelefonide ja tahvelarvutite kasvuga samas tempos kasvanud mobiilisõbralike veebilehtede osakaal. Kuigi peale Google teadet oma otsingualgoritmi muutmisest, tõusis kahe kuu jooksul ligi 5% mobiilisõbralike veebilehtede osakaal (Kwok, 2015). Forrester Research uuris 135 suurettevõtte (kokku 238 saiti) veebilehti -- kõigest 38% nendest olid mobiilisõbralikud. Kuna maailmas on üle 177 miljoni aktiivse veebilehe, on peaaegu võimatu öelda täpselt, mitu protsenti neist on mobiilisõbralikud. Kui üldistada Forrester Researcheri uuringutulemusi, samas võttes arvesse, et uuriti suuri ettevõtteid, kellel on finantsilised võimalused oma veebilehti uuendada, siis on üldine mobiilisõbralike veebilehtede osakaal tõenäoliselt oluliselt väiksem kui 38%. (Schadler, 2015).

Trilibis'e poolt läbi viidud uuringu kohaselt 155-st uuritud seadmetundlikust veebilehest 69% laadis nutitelefonis liiga kaua aega. Uuringu kohaselt kõigest 21% veebilehtedest laadis lehe alla 4 sekundi. Kuigi need veebisaidid olid seadmetundlikud, tingis pika laadimisaja enamasti suured pildifailid -- suured failid kuvati olenemata ekraani suurusest. (Gesenhues, 2014)

Tänapäeval on enamus mobiilse seadme kasutajatest ostnud midagi internetist. Ainuüksi ärilistel eesmärkidel on tähtis, et veebisait laeks kiiresti, sest iga sekund on arvel -- iga kulunud sekundiga kasvab kasutajate arv, kes loobuvad saidi külastamisest. Jõudlus mobiilses seadmes on oluline, sest:

- 47% kasutajatest eeldavad, et veebileht laeks vähemalt 2 sekundi jooksul
- 40% loobuvad, kui veebilehe laadimine võtab üle 3 sekundi aega
- 79% ostjatest, kes ei ole rahul veebisaidi jõudlusega, ei naase saidile tagasi
- 44% kasutajatest räägivad oma halvast kogemusest tuttavatele
- 52% veebis ostlejatest märgivad, et saidi lojaalsuse jaoks on kiire laadimisaeg tähtis. (Kissmetrics, kuupäev teadmata)

Lehekülgede laadimiskiirus mõjutab suuresti ettevõtte edukust internetis, sest mobiilseid seadmete osakaal internetitoimingutel on järjest kasvav. Seetõttu peaksid mobiilsed seadmed olema arenduses tähtsaim prioriteet, sest 79% kasutajaid ei naase saidile tagasi, kui nad pole selle jõudlusega rahul (Kissmetrics, kuupäev teadmata). Lisaks: 67% mobiilse seadme kasutajatest ostavad suurema tõenäosusega pakutavat teenust või kaupa, kui veebisait on mobiilisõbralik (Fisch, 2012).

Kuigi RWD abil saab kuvada mobiilis korrektset lehte, siis tänu mobiilse seadme kasutajate arvu tõusule, ei saa enam arendada veebi, mis pole maksimaalselt optimeeritud iga seadme jaoks. Nutitelefonide kasutajad on äärmiselt kärsitud -- kõigest mõned sekundid otsustavad lehe külastuse toimumise. Veebisaidi jõudlust mobiilses seadmes peaks parendama oluliselt, kui kasutada koos RWD'ga *mobile-first* arendusmeetodit.

2 Mobile-first disain ja arendus

Alates 2010. aastast, kui Ethan Marcoette defineeris *Responsive Web Design*'i, on tekkinud kaks peamist seadmetundliku disaini-ja arendussuunda: *graceful degradation* ning *progressive enhancement*. Eesti keeles oleks tegu sujuva vähendamise ning progressiivse suurendamisega. Esimese puhul disainitakse ja arendatakse kõigepealt veebisait arvutile, misjärel eemaldatakse elemente vastavalt ekraani suuruse vähenemisele. Teine on vastupidine: kõigepealt disainitakse ja arendatakse mobiilsele seadmele (väikesele ekraanile), kuhu lisatakse elemente vastavalt ekraani suurenemisele. (Parent, 2014)

Mobile-first ehk progressiivne suurendamine on meetod kindlustamaks, et veebilehed näeksid välja ja funktsioneeriksid suurepäraselt ennekõike mobiilsetel seadmetel. Raamatu "Mobile-first" (2009) autor Mike Wroblewski sõnul paneb mobiilsele seadmele disainimine keskenduma kõige tähtsamatele funktsioonidele, mis omakorda viib parema ja vähem kuhjatud kogemuseni, isegi arvutis (Archer, 2015). Kasvav nutitelefonide arvukus lõi olukorra, kus lühikese aja jooksul tekkis suur hulk mobiilse veebi kasutajaid, kes olid sunnitud sirvima veebi, mis oli algselt mõeldud arvutitele. See tähendas pidevat ekraanil suurendamist ja vähendamist, ebavajalikult suurte pildifailide laadimist jms. Mobiilse interneti madal kiirus tekitas veebilehtede pika laadimisaja. Kuigi mobiilse interneti kiirus on aasta-aastalt kasvanud ning Eestiski on kasutusele võetud juba peamiselt 4G ühendus, on probleemid varasemaga jätkuvalt sarnased -- veebilehtede laadimine võtab kaua aega ning mobiilse interneti kiirus on ebaühtlane ja kohati lausa väga aeglane. Samuti tekitab suuremahulise veebisisu laadimine kasutajatele probleeme andmemahuga ning selle piirangutega.

Lisaks internetikiirusele tuleb mobiilsete seadmete puhul arvesse võtta arvutiga võrreldes aeglasemaid protsessoreid ja väiksemaid mälusid. Seetõttu ei ole mõistlik nutiseadmes veebilehtedel kuvada keerukaid animatsioone ja kõrgekvaliteedilisi videoid vm. ressursinõudlikut graafikat.

Võrreldes arvutiekraaniga - tüüpiliselt 1024x768 pikslit (tänasel päeval juba 1366x768 pikslit), on nutitelefoni ekraan (keskmiselt 320x480) palju väiksem. Tihtipeale öeldakse: "*mobile-first = content-first.*" Selle all mõeldakse, et nutitelefonide ekraanide väiksuse tõttu tuleb veebilehe sisu hoolikalt läbi plaanida ning esitada ainult vajalik sisu. Sisu on

kõik, mis veebilehel asub: pildid, tekst, menüüd jms. Kasutajad tulevad veebilehele ennekõike informatsiooni järele ning eriti nutiseadme kasutajad on kärsitud ja soovivad vajalikku infot kiiresti ja lihtsasti saada. Veebilehtedel navigeerimine mobiilsetel seadmetel peab olema võimalikult lihtne ja sirgjooneline -- suurte ja mitmetasandiliste menüüde asemel vähendatud ja lihtsam menüü. Kõigepealt nutitelefonile arendades, luuakse veebisait, mis oleks kasutatav funktsionaalsel tasemel igal seadmel. Lisades sellele "põhjale" erinevaid tingimusskripte (käsujadasid), *media-queries* jms, saab luua parema kogemuse vastavalt seadme kontekstile (Archer, 2015).

Samas leidub ka *mobile-first* kritiseerijaid. Leitakse, et seda meetodit kasutades keskendutakse liialt mobiilsele kogemusele, jõudlusele ning disainile, mis toob omakorda kaasa kehva arvutile mõeldud saidi. See tundub olevat tõenäoline, kuid hästi läbimõeldud arendusprotsess ning disain peaksid eemaldama selle võimaliku puuduse. Heidetakse ette ka, et tegu on lihtsalt traditsioonilise (*desktop-first*) meetodi ümberpööratud variandiga. Nimelt ei lahenda *mobile-first* probleemi, vaid pööras selle ümber. Samuti heidetakse ette, et *mobile-first* disain ja arendus on igav, kuna põhirõhk langeb algselt mobiilsele lehele, mis pakub vähem võimalusi kui arvuti oma. (Archer, 2015)

Mobile-first võtab arvesse eelnevalt nimetatud mobiilse seadme limiteerivaid aspekte ning annab lahenduse (mobiilse) veebi kiirendamiseks ning kasutajatele parema kogemuse pakkumiseks. Veebilehe kiiremaks laadimiseks on *mobile-first* puhul tähtsaks, et laetav andmemaht oleks vastavuses seadme võimekuse ning kontekstiga.

Kuigi mobiilsel seadmel on palju kasutusmugavust vähendavaid aspekte, tasub meeles pidada, et leidub hulgaliselt eeliseid, mis kasutatavust parendavad. Näiteks: positsioneerimine, ekraani puutetundlikkus, helistamisvõimalus.

2.1 Mobile-first, desktop-first arendusmeetodite võrdlus

Arendamine *desktop-first* meetodiga käib tegelikult vastu CSS-i naturaalsele töövoolule -- CSS dokumente loetakse lineaarselt ülevalt alla. See tähendab, et iga dokumendis hiljem esinev deklaratsioon kirjutab eelneva üle. Samuti tekib *desktop-first* arenduse käigus palju üleliigset koodi -- kui kõigepealt arendatakse arvutile (CSS dokumendis üleval pool), siis

seejärel tuleb mobiilsele seadmele arendades üle kirjutada kõik mittesobivad atribuudid. Näiteks (vt Koodinäide 1):

Mobile-first ülesehitusmoodus

```
ul {
  li {
    list-style: none;
    margin: 0;
    padding: 5px 10px;
  }
}
@media only screen and (min-
width: 780px) {
  ul {
    li {
      display: inline-block;
      margin: 0 10px;
      padding: 8px 15px;
      border: 1px solid #333;
      border-radius: 25%;
    }
  }
}
```

Desktop-first ülesehitusmoodus

```
ul {
  li {
    display: inline-block;
    margin: 0 10px;
    padding: 8px 15px;
    border: 1px solid #333;
    border-radius: 25%;
  }
}
@media only screen and (max-width
780px) {
  ul {
    li {
      display: block;
      margin: 0;
      padding: 5px 10px;
      border: none;
      border-radius: 0;
    }
  }
}
```

Koodinäide 1 Mobile-first ja desktop-first arendusmeetodite ülesehituse erinevused

Tegemist on menüüga, mis mobiilsel seadmel on vertikaalne ja arvuti ekraanil (suuremal kui 780 pikslit) horisontaalne. Kasutatud on tüüpilisi menüüdes esinevaid stiile. Esmasel vaatlusel selgub juba, et *mobile-first* kood on tunduvalt kompaktsem ja loetavam kui *desktop-first* kood. Selle põhjuseks on asjaolu, et *mobile-first* puhul ei kirjutata nii palju stiilideklaratsioone üle (Wals, 2015).

Koodipikkuse kokkuhoidu näitab hästi ka järgmine, eelnevast lihtsam ning lühem, näitekood (vt Koodinäide 2):

Mobile-first ülesehitusmoodus	Desktop-first ülesehitusmoodus

`@media screen and (min-width: 480px){`
`  .omadus {`
`  width: 50%`
`  float: left;`
`  }`
`}`
	`.omadus {`
	`  width: 50%;`
	`  float: left;`
	`}`
	
	`@media screen and (max-width: 480px) {`
	`  .omadus {`
	`  width: auto; //või 100%`
	`  float: none;`
	`  }`
	`}`

Koodinäide 2 Mobile-first ja desktop-first koodipikkuste erinevuse näide

Antud näite puhul on samuti selgesti näha, et koodijupp *mobile-first* ülesehitusmooduse puhul on oluliselt kompaktsem. Kui *desktop-first* arenduses määratakse kõigepealt HTML elemendi klassile vaikeväärtusest erinevad omadused ning hiljem, väiksema ekraani puhul, kirjutatakse need uuesti algväärtusteks, siis *mobile-first* arendusmeetodi puhul kasutatakse ära nende elementide vaikeväärtusi. See omakorda hoiab kokku suure osa koodist ning ei aja seda ebavajalikult pikaks ning keeruliseks. Praegusel juhul tuleneb kokkuhoid sellest, et blokk-element on vaikimisi 100% laiusega ning ei "hõlju" (*float*).

Veebilehel ebavajaliku elemendi peitmiseks kasutatakse tunnus-väärtuskomplekti *display: none*. Kuigi see eemaldab elemendi lehel kuvatavate elementide hulgast, laetakse see teatud juhtudel ikkagi alla. See aga suurendab veebilehe mahtu ning pikendab laadimisaega.

Järgnevas näites (vt Koodinäide 3) on element *test* eemaldatud veebilehelt (kasutades tunnus-väärtuskomplekti *display: none*), kui ekraanisuurus on alla 640 piksli. Kuigi need kaks koodinäidet teevad täpselt sama asja, on neil üks suur erinevus. *Mobile-first*

arendusmeetodi korral ei laeta pildifaili alla, kui ekraanisuurus ei ületa 640 pikslit. Pildifail laetakse alla *desktop-first* arenduse puhul. Need kaks väidet kehtivad peaaegu kõigi brauserite korral, v.a. Mozilla Firefox 3.6+ ning Opera Mini ja Opera Mobile, mis ei lae pildifaili alla, kui brauser seda lehel kuvamiseks ei kasuta (Kadlec, 2012).

Mobile-first arendusmeetod	Desktop first arendusmeetod

`.test{`
`  display: none;`
`  }`
`@media only screen and (min-width:640px){`
`  .test{`
`  width: 400px;`
`  height: 300px;`
`  background-image: url(bg-image.jpg);`
`  display: block;`
`  }`
`}`
	`.test{`
	`  width: 400px;`
	`  height: 300px;`
	`  background-image: url(bg-image.jpg);`
	`}`
	`@media only screen and (max-width: 640px){`
	`  .test{`
	`  display: none;`
	`  }`
	`}`

Koodinäide 3 Mobile-first ja desktop-first arendusmeetodite tagatausta laadimise erinevused

Mobile-first meetod mitte ainult ei kasuta lehe laadimiseks vähem samme, vaid ka eemaldab vajaduse laadida üleliigseid pilte. (Baumann, 2015)

Pildifailide tingimuslik laadimine (kasutades *media-query*'t) on viimastel aastatel toimunud suur edasiminek. Viis aastat tagasi leidus mitmeid populaarseid brausereid (näiteks: Android 2.0-3.0 ning Safari 4.0), mis laadisid alla pildifaile, mis ei vastanud ainult konkreetselt täidetud *media-query* tingimustele. Kuna CSS faili loetakse lineaarselt ülevalt alla, laetakse mõnede vanemate brauseri poolt alla kõik elemendile määratud tunnused, kuni leitakse tingimus, mis täidetakse. See kehtib nii *mobile-first* kui ka *desktop-first* ülesehitusmooduse puhul. Erinevus esineb veebilehe laadimiskiiruses mobiilsel seadmes, sest *desktop-first* arendusmeetodi puhul kirjutatakse väiksematele ekraanidele mõeldud tunnused peale suurte ekraanide tunnuseid; samas kui *mobile-first* meetodi puhul on need kõige esimesed.

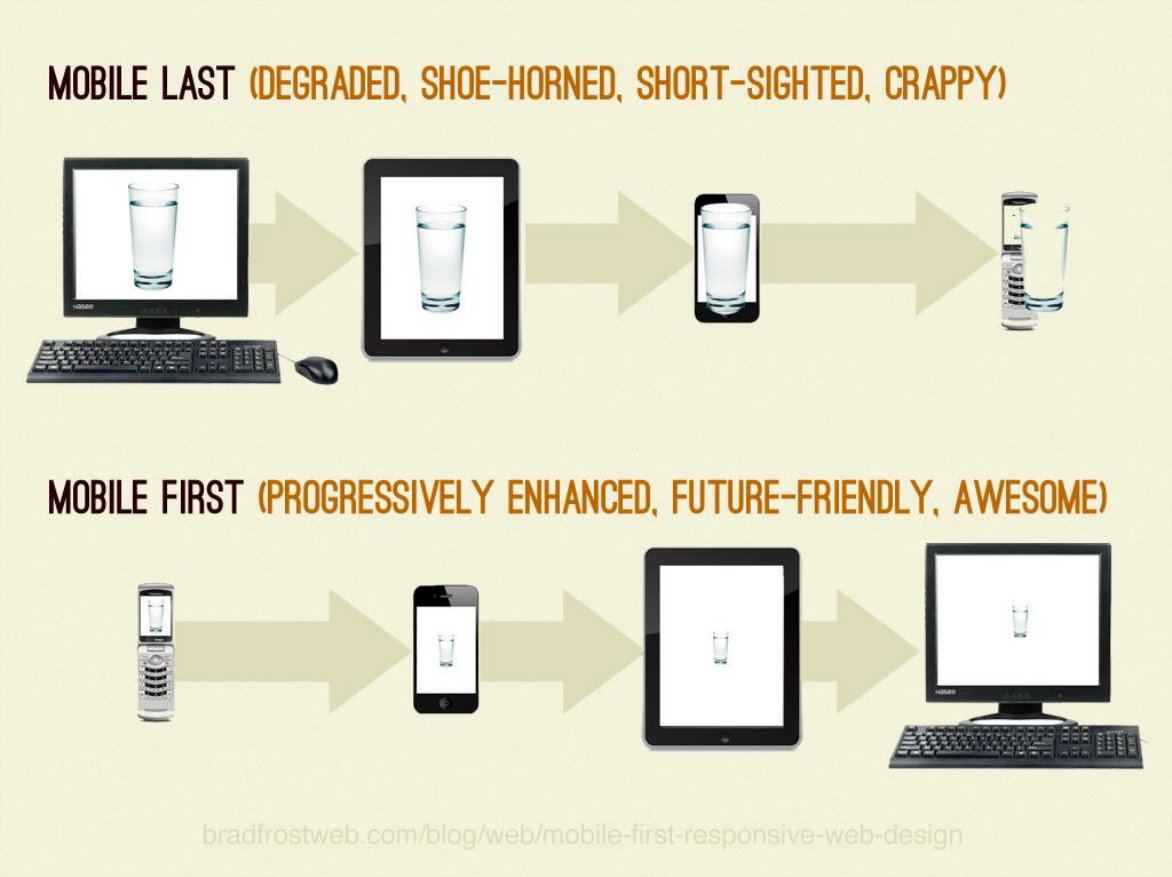
Mobile-first	Desktop-first

`.test {`
`  height: 400px;`
`  background-image: url(pilt-sm.jpg);`
`}`
`@media only screen and (min-width: 640px) {`
`  .test {`
`  background-image: url(pilt-lg.jpg);`
`  }`
`}`
	`.test{`
	`  height: 400px;`
	`  background-image: url(pilt-lg.jpg);`
	`}`
	`@media only screen and (max-width: 640px) {`
	`  .test{`
	`  background-image: url(pilt-sm.jpg);`
	`}`
	`}`

Koodinäide 4 Mobile-first ja desktop-first ülesehitusmooduste tagatausta piltide laadimine

Näiteks eelnevas näitekoodis (vt. Koodinäide 4) laetakse *desktop-first* arendusmeetodi puhul, mobiilses seadmes (ekraanisuurusega alla 640 pikslit), alla ka suurematele ekraanidele mõeldud pildifail (uuemad brauserid suudavad juba ainult õige pildifaili alla laadida). Samas kui *mobile-first* puhul laetakse väikese ekraaniga seadme puhul alla vaid konkreetselt sellele mõeldud tunnused.

Mobile-first ja *desktop-first* arendus-ja disainimeetodite erinevuse võtab kokku siinse teema vast kõige kuulsam illustratsioon veeklaasi näitel:



Joonis 1 Desktop-first ja mobile-first arendusmeetodite võrdlus veeklaasi näitel (Frost, 2012)

Ülal paikneva joonise (vt Joonis 1) peal on näha, kuidas *desktop-first* (joonisel nimetatud *mobile last*) puhul laetakse samas mahus sisu brauseris olenemata seadme suurusest (ja võimsusest) ning kuidas seadme suuruse vähenedes veeklaasina kujutatud veebileht seadmes järjest ebaproportsionaalsemaks muutub. Samas *mobile-first* meetodi puhul jääb laetav sait jõukohaseks igale seadmele. Samas tuleb arvestada, et antud näide toimib ideaalsetes tingimustes ning tõenäoliselt kasvab *mobile-first* meetodi puhul veeklaasi (veebisaidi maht) suurus seadme suurenemisega proportsionaalselt.

3 Veebisaidi jõudluse parendamine mobile-first meetodi abil

Nagu eelnevas peatükis mainitud, tuleks jõudluse parendamiseks kasutada *mobile-first* arendusmetoodikat. CSS failis tuleb kõige pealt arendada veebileht mobiilset seadet silmas pidades ning seejärel alles suurematele seadmetele. Tehniliselt tuleks selleks CSS koodis *media-query*'dena kasutada *max-width* tingimuse asemel *min-width*. CSS faili lugemine toimub lineaarselt ülevalt alla ning *media-query* kujutab endast tegelikult tingimuslauset, mis vaatab, kas ekraan on suurem või väiksem defineeritud tingimusest ning rakendab stiilideklaratsioonid vastavalt tingimuse täitmisele. Seetõttu kasutades *min-width media-query*'t toimub arendamine juba tinglikult *mobile-first* metoodika järgi (arendatakse väiksemalt suuremale).

Mobile-first ülesehituse puhul tuleks kindlasti ära kasutada HTML5 ja CSS3 poolt pakutavaid võimalusi, mis teevad veebilehe arendamise mugavamaks ja kiiremaks ning aitavad parendada veebilehe üldist jõudlust. Samas tuleb meeles pidada, et paljud uued HTML5 ja CSS3 võimalused ei leia toetust kõikidel veebilehitsejatel. Praeguseks on jõutud siiski sinnamaani, kus paljud võimalused on toetatud suurimate brauserite uuematel versioonidel (uued CSS3 võimalused puuduvad Opera Mini ning Internet Explorer kuni versioonini 10) (Deveria, 2014). Tuleviku mõttes lisab kindlust antud võimalusi mitte-toetavate brauserite populaarsuse vähenemine.

Kõige lihtsam viis veebilehe andmemahu vähendamiseks on kasutada eri seadmetel erineva suurusega pildifaile. HTML5 võimaldab DOM'is (Document Model Object) muuta laetavat sisu olenevalt ekraanisuurusele.

```
<img src="pilt_sm.jpg sizes="80vw" srcset="pilt_sm.jpg 480w, pilt_m.jpg 780w, pilt_l.jpg 1200w, pilt_xl 2000w">
```

Koodinäide 5 HTML 5 srcset näide

Ülal asuv koodijupp (vt. Koodinäide 5) kuvab pildi alati 80% ulatuses seadme laiusest ning ekraanile laetakse erinev pildifail, vastavalt sellele, kas seadme laius ületab mõnda seatud tingimust. *Srcset* abil saab määrata lähtefaili ning selle pildifaili mõõtmed, et brauser saaks otsustada, milline lähtefail on kõige parem vastavale vaateaknale.

HTML5 spetsifikatsioon sisaldab uusi semantilisi struktuurelemente, mida kasutades saab luua efektiivsemalt sõelutud (inglise keeles *parse*) veebilehte. Uued struktuurielemendid on näiteks: *header*, *nav*, *article*, *aside* ning *footer*. Semantiline leht on väiksem ja laeb kiiremini. Ühtlasi tähendab lihtsam (üheselt määratletud) DOM kiiremat JavaScripti töötlemist (Everts, 2013). Lisaks leidub HTML5 spetsifikatsioonis ka uusi sisestuselemente. HTML5 sisestusvälja (*input* element) tüübiks saab määrata nüüdsest *e-mail*, *URL (Uniform Resource Locator)*, *number*, *date* ja *time*. Sisestusväljale oodatava sisestustüübi määramine aitab mobiilsetel seadmetel kuvada ekraanile sisule vastav klaviatuur. Näiteks numbriväljale vajutades ilmub ekraanile klaviatuur, kus on ainult numbrid. Lisaks saab nüüdsest määrata otse HTML'is kohatäite teksti (inglise keeles *placeholder*) sisestusväljale. Kohatäide on juhendav tekst, mis kuvatakse sisestusväljal, kui sinna pole midagi sisestatud. See uuendus võimaldab kasutada kohatäidet ilma JavaScript'I abita.

CSS3 ilmumine andis lisaks võimalusi veebilehtede kiiremaks laadimiseks. Kindlasti üks tähtsamaid uuendusi oli asjaolu, et üha enam saab kasutada graafiliste lahenduste realiseerimiseks koodi. *Border-radius* atribuudi abil on võimalik lisada elementidele pehmeid nurki, või need lausa ringikujulisteks teha; *box-shadow* abil saab lisada elementidele varje; *gradient-fill* võimaldab luua värvi-gradatsioonilisi taustasid jne. Enne neid võimalusi pidi näiteks toodud graafilised elemendid realiseerima paljude pildifailidega. Tänaseks ei pea nende lahenduste loomiseks moodsatel brauseritel tegema HTTP (*HyperText Transfer Protocol*) päringuid piltide laadimiseks lehele ega laadima pildifaile, mis omakorda hoiab kokku veebilehe üldist mahtu ning peaks tagama kiirema laadimisaja. Lisaks tuleks mainida, et erinevatel brauseritel on erinevad paralleelselt tehtavate HTTP päringute arvud. Paralleelsete päringute arv näitab, mitu päringut saab samaaegselt teha (mitu faili samaaegselt alla laetakse brauseri poolt) (Tuan, 2014).

Arvestama peab samas sellega, et kuigi CSS3 abil on võimalik vähendada veebisaidil HTTP päringute arvu, lisavad paljud CSS3 graafilised omadused suuremat koormust töötlemisele (inglise keeles *processing*). Kuigi koormuse (laadimisaja) suurenemine on peaaegu minimaalne (0.05s-0.12s, olenevalt omadusest), tuleks sellega kindlasti arvestada, kui lehel on palju elemente, millele on lisatud CSS3 omadusi, sest teatavasti on mobiilsetel seadmetel enamasti aeglasemad protsessorid ja vähem vahemälu. (Alstad, 2015)

Mahu kokku hoidmiseks ning HTTP päringute arvu vähendamiseks kasutatakse tihtipeale pildifailide kokku monteerimist (inglise keeles *image sprite*) üheks suureks pildifailiks. Seejärel positsioneeritakse õige koht CSS abil ning määratakse see (pildifail koos koordinaatidega) valitud elemendi tagataustaks. Positsioneerimiseks saab kasutada CSS atribuuti *background-position,* millele tuleb anda X ja Y telje koordinaadid.

Image sprite fail näeb visuaalselt välja selline (värvid kujutavad joonisel erinevaid pilte) (vt. Joonis 2):

Joonis 2 Image sprite näide

HTML sümbolite ja eriliste märkide kasutamine lihtsamate ikoonide tarbeks on hea lahendus, kuidas hoida kokku pildifailide kasutamist. HTML sümboleid tuleks lisada otse HTML faili, kasutades selleks spetsiaalset koodi. Näiteks rahu sümboli kasutamiseks, tuleks lisada HTML failis sobivale kohale *☮* (Dixon & Moe, 2015). Kuna tegu on põhimõtteliselt tekstiga, saab seda töödelda nagu tavalist teksti: muuta suurust, värvi jne. Lisaks on HTML sümbolite eeliseks pildifailide ees ka asjaolu, et sümboli suurendes ning kõrgema resolutsiooniga ekraanidel jääb kvaliteet alati samasuguseks. (Frost, 2012)

Piirata võiks Javascripti kasutamist seal, kus see on võimalik. Välise Javascripti laadimine loob uue HTTP päringu. Pigem kasutada välise faili asemel *inline* (HTML faili sisene skript) Javascripti koodi HTML faili sees. Näiteks kulub Starbucksi veebisaidi (ilma Javascript'ita) laadimisele, hea kiirusega internetiga lauaarvutis, 3,53 sekundit, samas laadides koos välise Javascript'iga kulub laadimisele 4,73 sekundit. See on tervelt 34% rohkem. *Inline* Javascripti kasutavad oma veebisaitidel näiteks BBC ning Amazon. (Johansson, 2013) Samas tasub arvesse võtta, et tegu oli stabiilse ühendusega lauaarvutiga, mitte nutiseadmega, millel on tihti ebastabiilne internetiühendus, aeglasem protsessor ning väiksem mälumaht ning vähem vahemälu. Seega tuleks Javascripti

kasutada minimaalselt ning kasutades hoida seda pigem HTML failis (*script* märgendite vahel).

Tänu CSS3 animatsioonidele ning kursorifunktsioonidele, saab nüüdsest palju asju teha ilma JavaScript'ita. Näiteks on võimalik CSS3-s määrata elemendi peal kursoriga "hõljudes" (inglise keeles *hover*) elemendile uusi omadusi. CSS3 võimaldab luua ka *keyframe* animatsioonide jaoks ning määrata erinevaid üleminekute sätteid. Animatsioonidel on ka suhteliselt hea ühilduvus: seda toetavad kõige populaarsemate veebilehitsejate uusimad versioonid (toetatud pole Opera Mini ning Internet Explorer enne versiooni 10). (Deveria, 2014)

Veebilehe laadimist kiirendab ka CSS ja Javascript failide minimeerimine. Minimeerimine tähendab faili kompaktseks tegemist, eemaldades failist kõik tühikud ja reavahetused -- koodi nii-öelda kokfku surumine. Efektiivseima tulemuse saamiseks tuleks kasutada vaid ühte minimeeritud CSS ja ühte minimeeritud Javascript faili. Olenevalt faili suurusest, võib minimeerimine vähendada failimahtu 20-50% (Hornor, 2013). Faili minimeerimist tuleks teha alles kõige lõpus, kuna minimeeritud faili on raske lugeda ning muuta.

4 Mobile-first arendusmeetod komponentide näitel

Autor koostab kaks samasugust näitekomponenti, kuid kasutab nende arendamisel erinevaid ülesehitusmeetodeid. Lisaks on iga valitud komponendi kohta autor kirjutanud põhjaliku kirjelduse selle olemusest, ülesehitusest ning arendusprotsessist. Näitekomponendid valis autor arenduseks vastavalt eelmises peatükis käsitletud teoreetilistele meetoditele. Lisaks võttis autor arvesse veebidisaini hetketrende, soovides saavutada võimalikult autentset objekti uurimiseks.

4.1 Komponent 1 - pakutavate teenuste võrestikriba

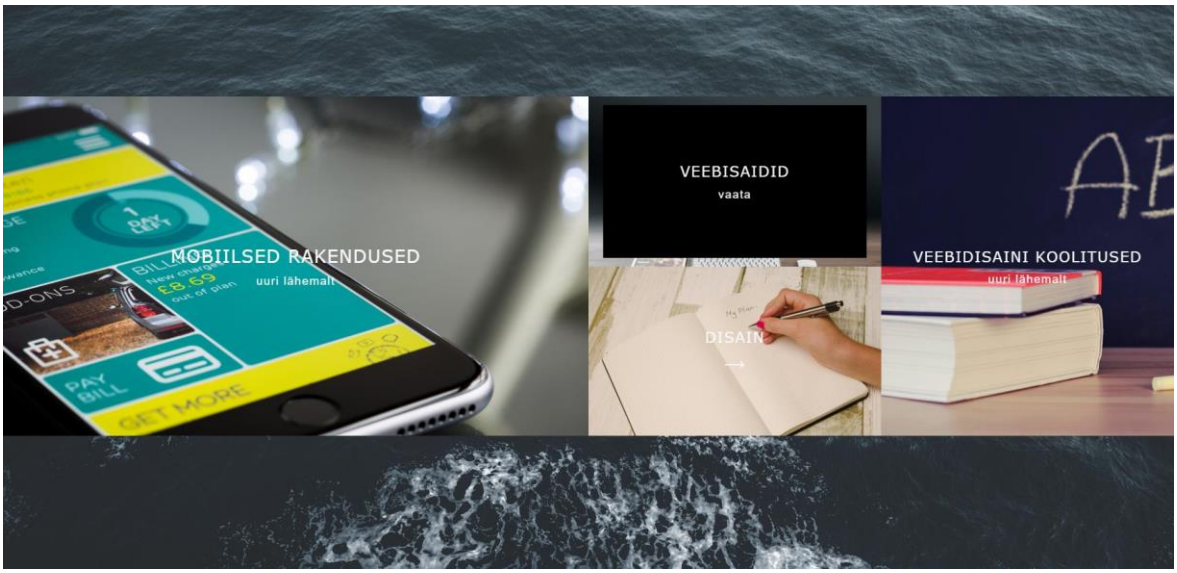
Autor valis esimeseks arendusnäiteks "Pakutavad teenused" komponendi, mida võib leida paljude ettevõtete koduleheküljegedelt. Valiku eesmärgiks on näidata, kuidas muutub andmemaht ning laadimiskiirus, kasutades *mobile-first* ülesehitusmeetodit, seadmetundlike (*responsive)* pilte ning CSS3 poolt pakutavaid interaktsioone.

Ülesehitus koosneb autori poolt arendatav "Pakutavad teenused" komponent neljast pildifailist, mis on reastatud üksteise kõrvale. Mobiilses seadmes on esimene ja viimane (neljas) pildifail 100% laiusega; ning teine ja kolmas pildifail asetsevad kõrvuti reas, kus üks pilt moodustab 50% vaateakna (inglise keeles *viewport*) laiusest. Esimese pildifaili kõrgus moodustab 100% nutitelefoni vaateakna kõrgusest; ülejäänud kolm pildifaili jagavad omavahel ära veel 100% ekraanikõrgust. Alates tahvelarvutitest muutub komponendi kujundus. Kogu komponendi kõrgus on 50% vaateakna kõrgusest. Esimese ja viimase pildi kõrgus on 100%; esimese pildifaili laius on 50% vaateakna laiusest, samas kui ülejäänud piltide laius on 25% vaateaknast. Teise ja kolmanda pildifaili kõrguseks on 50% komponendi kõrgusest. Lisaks on suurema ekraaniga seadmes määratud ka taustapilt.

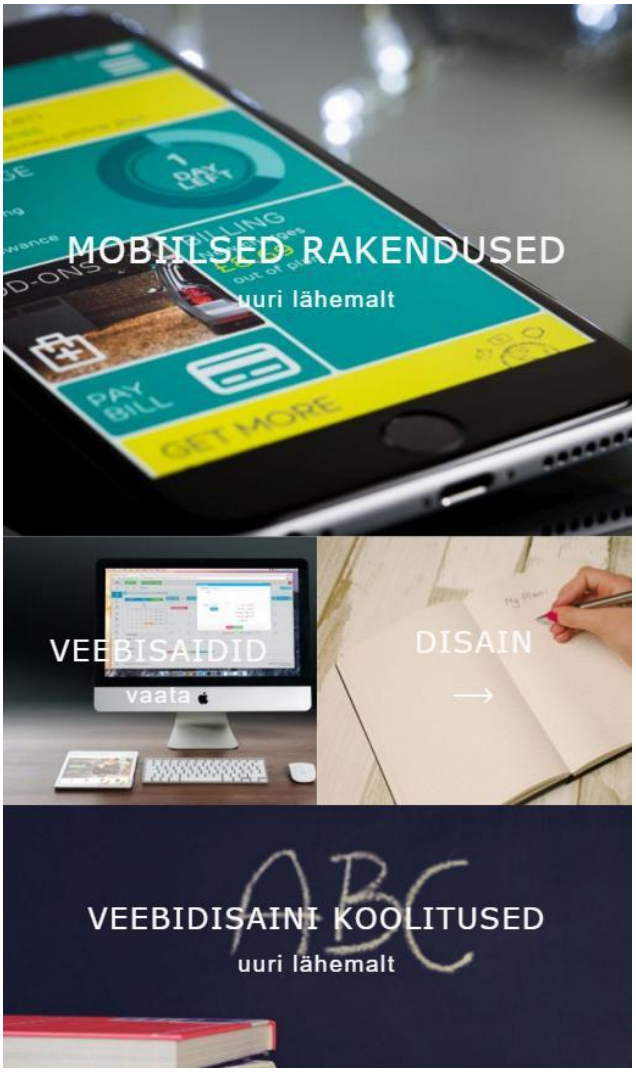
Komponent on üles ehitatud erinevate blokk-elementidega. Kõige kõrgem tase on jagatud kaheks. Esimeses "kastis" on element klassiga *block-1* ning teises on ülejäänud kolm elementi klassidega: *block-2, block-3* ning *block-4*. Blokk-elementide asetust on muudetud vastavalt ette määratud kujundusele, kasutades selleks CSS atribuuti *float*. Antud ülesandes on kõik pildifailid kuvatud CSS abil. Nimelt on CSS failis iga elemendi tagataustaks

määratud vastav fail. CSS'i abil on suurtel ekraanidel (ekraanid laiemad kui 640 pikslit) lisatud lihtne interaktiivsus: kursoriga blokk-elemendile liikudes kuvatakse taustapildi peale värviline kast. Kursori liikumise tuvastamiseks kasutas autor CSS3's leiduvat *hover* funktsiooni. Värvilise kasti tekitamisel kasutas autor CSS'i pseudoelementi *before*. Samuti on suurtel ekraanidel näite eesmärgil määratud ka tagataust, mis on väiksematel ekraanidel eemaldatud. Eemaldamiseks kasutas autor CSS atribuuti *display*, mille väärtuseks väikestel ekraanidel on *none* ning suurtel ekraanidel *block*.

Näitekomponent asub aadressil: http://www.tlu.ee/~hainer/loputoo/esimene/



Joonis 3 Komponent 1 vaade arvutis



Joonis 4 Komponent 1 vaade mobiilses seadmes

4.2 Komponent 2 - veebilehe päis koos seadmetundliku menüüga

Teiseks arendusnäiteks valis autor veebilehe päise. Hetkel on veebidisainis populaarne kasutada ülisuuri päiseid, mis tihtipeale täidavad terve vaateala äärest ääreni ning mille tagataustaks on suur pildifail. Autor valis antud komponendi, kuna soovib uurida, kuidas kasutades *mobile-first* arendusmetoodikat ilma JavaScriptita mõjutab lehekülje laadimiskiirust ning laetavat andmemahtu.

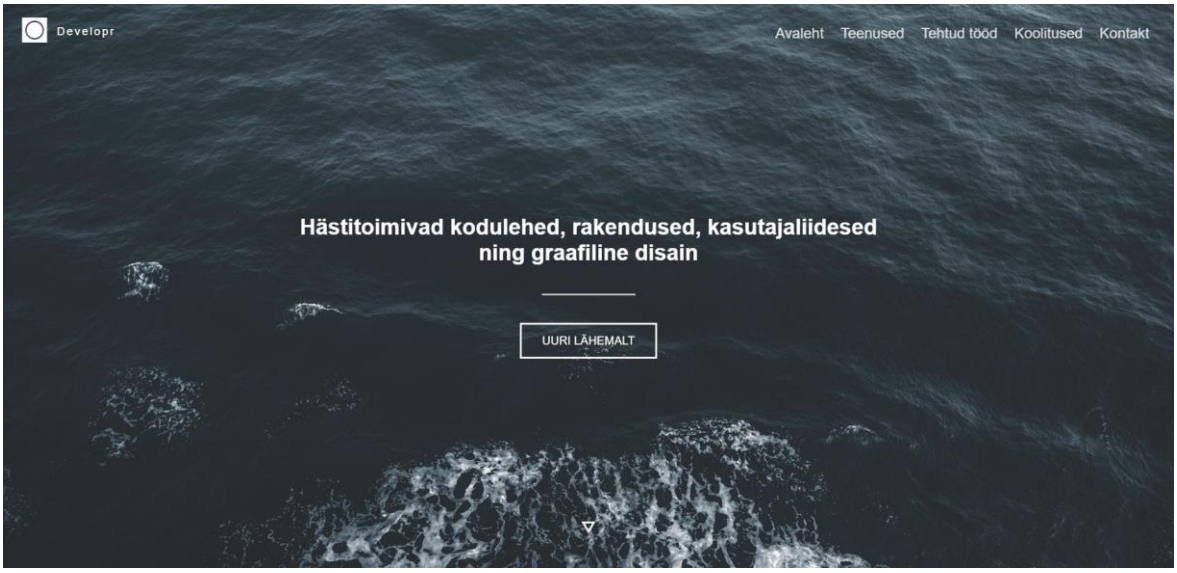
Autor soovib luua päisse seadmetundliku menüü, mis on mobiilses seadmes peidetav ning mida saab kuvada ning peita vastavat lülitil vajutades; samas kui suurematel ekraanidel on menüü kogu aeg nähtav. Lisaks on mobiilses seadmes tegu vertikaalse menüüga ning

suurematel (arvuti) ekraanidel horisontaalselt paikneva menüüga. Antud menüü soovib autor arendada *mobile-first* ülesehitusmooduses ilma JavaScripti jQuery teegita; samas *desktop-first* meetodi puhul plaanib autor antud teeki kasutada. Lisaks plaanib autor *mobile-first* meetodi puhul päise tagataustana kasutada väiksemat pilti ning laadida pildifail kasutades CSS'i; *desktop-first* puhul aga laadida tagatausta pildifail HTML failis. Autor soovib sedasi uurida CSS'i-sisese ning HTML-sisese piltide kasutamise mõju veebilehe laadimiskiirusele.

Antud komponendi *mobile-first* ülesehitusmeetodi puhul kasutas autor menüü lüliti (pildil kolm valget joont) tegemisel ainult HTML'i ning CSS'i. Lüliti on HTML failis defineeritud kui märgend (inglise keeles *label*) märkeruudu (inglise keeles *checkbox*) jaoks. Nende kahe ühendamiseks lisas autor *label* elemendile *for* atribuudi, mille väärtuseks oli antud märkeruudu *id* (identifikaator). Seda tehes saab sisestada "linnukese" märkeruudu sisse, kui vajutada menüü lüliti peal. Edasi kasutas autor CSS failis selektorit *checked*, mille abil saab vaadata, kas märkeruutu on pandud linnuke. Kui märkeruut on märgitud, liigub menüü paremalt poolt ekraanile. Selline menüü aktiveerimismeetod toimib analoogselt jQuery *toggleClass* funktsioonile. Suurtel ekraanidel on menüü horisontaalselt ning kogu aeg nähtaval. Autor eemaldas menüü lüliti suurtel ekraanidel, kasutades selleks CSS atribuuti *display: none*. *Mobile-first* meetodi puhul laetakse tagataust läbi CSS faili. Samuti animeeris autor ekraani alaosas oleva kolmnurga, mis liigub lõputult üles-alla. Selle efekti saavutamiseks kasutas autor CSS'i *keyframe*.

Desktop-first arendusmeetodi puhul laetakse ekraanisuurusele vastavad taustapildid läbi HTML faili. Pildifailide ning ekraanisuuruste vastavuse märkimiseks kasutas autor CSS *media-query*'sid, mille käigus eemaldatakse kõik vastaval ekraanisuurusel mittekasutatavad pildifailid DOM'ist, kasutades selleks *display: none* atribuuti. Erinevus mobile-first meetodiga on seegi, et autor kasutas menüü ekraanile toomiseks Javascripti teeki jQuery. Nimelt lõi autor CSS faili uue klassi, mille tunnused lisatakse menüü-elemendile, kui on vajutatud menüü lülitile. Lülitiefekti saavutamiseks kasutas autor jQuery funktsiooni *toggleClass*, mille käigus lisatakse ja eemaldatakse märgitud klass elemendilt vastavalt sellele, kas antud elemendil on juba see klass määratud või mitte. Kui *mobile-first* meetodi puhul olid logo ning menüü lüliti visualiseerimiseks kasutatud HTML sümboleid, siis *desktop-first* puhul kasutas autor nende tarbeks pildifaile.

Näidiskomponent asub aadressil: http://www.tlu.ee/~hainer/loputoo/teine/



Joonis 5 Komponent 2 vaade arvutis

Joonis 6 Komponent 2 vaade mobiilses nutitelefonis

4.3 Komponent 3 - kontaktvorm koos töötajate kaartidega

Kolmandaks näitekomponendiks valis autor kontaktvormi koos vastutavate töötajate kaartidega, mis koosnevad töötaja pildist, nimest ning ametipositisioonist. Autor kasutab selle komponendi *mobile-first* arenduses monteeritud pilte (*image sprite*) ning tagataustana CSS värvi-gradatisooni. Vastukaaluks kasutab autor *desktop-first* ülesehitusmooduse arenduses iga elemendi tarbeks eraldi pildifaile.

Mobiilses seadmes koosneb kontaktvormi komponent kolmest töötajakaardist, kolme sisestusväljaga vormist ning kahest nupust. Lisaks kasutab autor *mobile-first* ülesehitusmeetodi puhul maakaardi kuvamiseks HTML5 seadmetundlike piltide funktsionaalsust.

Autor loodab antud komponendiga võrrelda *image sprite* kasutamise efektiivsust erinevate pildifailide kasutamisega.

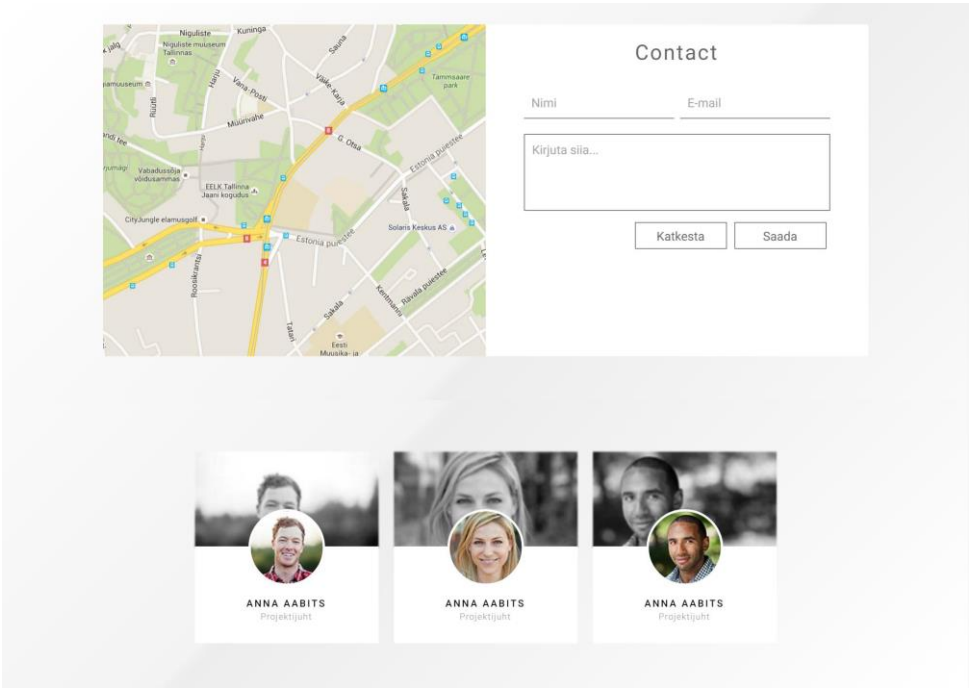
Autor kasutas HTTP päringute vähendamiseks *image sprite* meetodit. Kõik "töötajakaardil" olevad pildid pani autor pilditöötlusprogrammis ühte faili ning määras CSS failis igale töötajakaardile antud pildifailis kindlatel koordinaatidel paiknevad kujutised. Selleks kasutas autor CSS atribuuti *background-position*, mis sai väätusteks X ja Y koordinaadid.

Joonis 7 Komponent 3 arenduses kasutatud image sprite fail

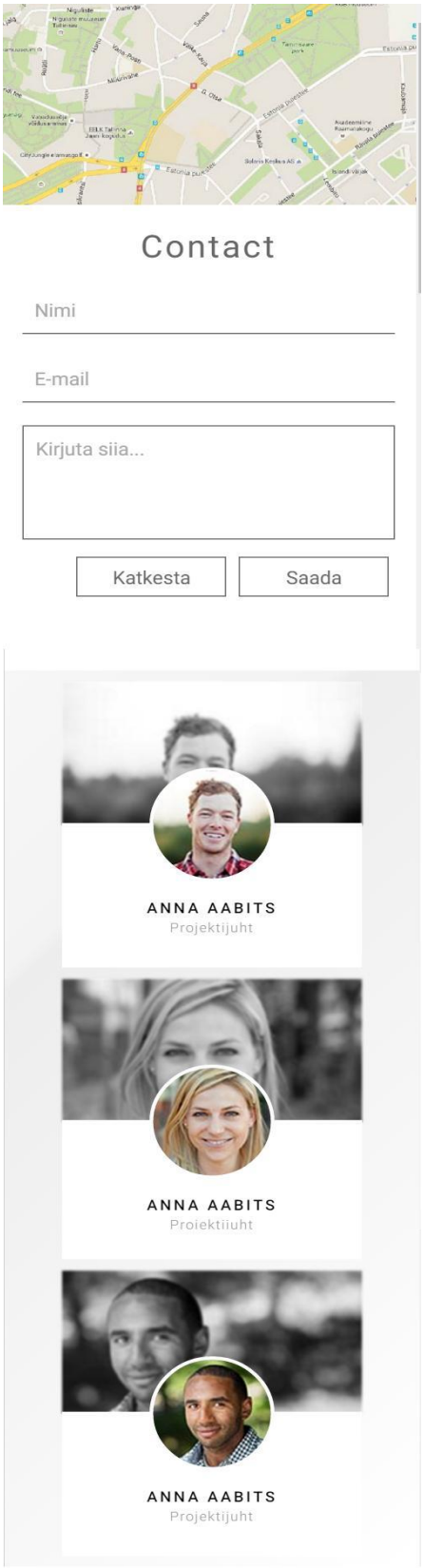
Nii andmemahu kui ka HTTP päringute vähendamiseks määras autor *mobile-first* veebilehel tagataustaks CSS3 sisseehitatud *linear gradient* (lineaarne värvigradatsioon)

funktsionaalsust. Lisaks kasutas autor *mobile-first* meetodi puhul CSS faili minimeerimist. *Desktop-first* veebilehel kasutas autor tagataustal gradatsiooni saavutamiseks pildifaili.

Näitekomponent asub aadressil: http://www.tlu.ee/~hainer/loputoo/kolmas/



Joonis 8 Komponent 3 vaade arvutis



Joonis 9 Komponent 3 vaade nutitelefonis

5 Näitekomponentide tulemused ning tulemuste analüüs

Siinses bakalaureusetöös seatud hüpoteesi kontrollimiseks arendab autor erinevaid kuid paljudel veebilehtedel leiduvaid komponente, kasutades kahte ülesehitusmoodust: *mobile-first* ning *desktop-first*.

Peale komponentide valmimist võrdleb autor kahe ülesehitamismooduse laadimiskiiruseid erinevatel seadmetel (nutitelefon, sülearvuti); samuti võrdleb autor kahe arendusmeetodi puhul tekkinud (laetava) veebilehe andmemahtu. Laetavat andmemahtu kasutab autor võrdluspunktina seetõttu, kuna kasutades eri-resolutsiooniga pildifailide laadimist vastavalt seadmele, tekitatakse veebiprojekti lisaks pildifaile, mis tõstavad kogu projekti üldist mahtu. Autor kasutab *Google Chrome Developer's Tool* tööriistakastis leiduvat *Network* funktsiooni arendatud komponentide laadimiskiiruse, tehtud HTTP päringute ning laetud andmemahu kohta informatsiooni kogumiseks. Programmi poolt tehtud mõõtmiste väljundina kasutab autor *transferred*, *finished* ning *load*. *Transferred* näitab edastatud andmemahtu, *finished* lehel olevate andmete täielikuks allalaadimiseks kulunud aeg ning *load* laadimisaega, millega veebileht kasutajale kuvati. Tööriistas on määratud ka mobiilsele seadmele filter internetikiiruse kohta. Autor valis *Good 3G*, mille allalaadimiskiiruseks on 1Mb/s ning üleslaadimiskiirus on 750Kb/s. Arvutile määras autor filtri, mille üleslaadimiskiiruseks on 6Mb/s ning üleslaadimiskiiruseks 1Mb/s. Brauseriks valis autor Google Chrome versioon 49.0. Mobiilse ekraani suuruseks määras autor iPhone 5 ekraanisuuruse, milleks on 320x568 pikslit. Arvuti ekraani suuruseks määras autor 1366x768 pikslit. Ühesuguste tulemuste saavutamiseks keelas autor brauseril salvestada vahemälusse (inglise keeles *cache*) andmeid. Autor otsustas määrata filtrid andmesidekiirusele, ekraanisuurusele ning veebilehitsejale, et tulemused oleksid salvestatud võrdsetes tingimustes.

Autor teeb andmete õigsuse tagamiseks iga näitekomponendi ülesehitusmooduse kohta kolm mõõtmist. Autor kasutab ülesehitusmooduste omavaheliseks võrdlemiseks mõõtmiste tulemuste keskmist väärtust.

5.1 Komponent 1 mõõtmistulemused

Autor soovis antud komponenti arendades näha kui suur on laadimiskiiruse, HTTP päringute ning andmemahu vahe *mobile-first* ning *desktop-first* arendusmeetodite puhul.

Tabel 1 Komponent 1 mobile-first arendusmeetodi mõõtmistulemused nutitelefonis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	1.33s	1,35s	1,32s
Kõikide andmete laadimiseks kulunud aeg	1,4s	1,4s	1,38s

Ülal olev Tabel 1 näitab, et *mobile-first* ülesehitusmoodus kasutas veebilehel 8 HTTP päringut ning laadis alla 184KB andmeid. Kolme mõõtmise keskmine kuvamiseks kulunud aeg oli 1,33s ning veebilehel olevate andmete täielikuks allalaadimiseks kulus keskmiselt 1,39s.

Tabel 2 Komponent 1 mobile-first arendusmeetodi mõõtmistulemused arvutis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	554ms	770ms	717ms
Kõikide andmete laadimiseks kulunud aeg	1,97s	2,32s	2,49s

Ülal olev Tabel 2 näitab, et arvutis laetud veebileht tegi kokku 9 HTTP päringut ning laadis iga kord alla 960kB andmeid. Kolme mõõtmise keskmine kuvamiseks kulunud aeg oli 680ms ning veebilehel olevate andmete täielikuks allalaadimiseks kulus keskmiselt 2,26s

Tabel 3 Komponent 1 desktop-first meetodi tulemused nutitelefonis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	3,94s	3,89s	3,88s
Kõikide andmete laadimiseks kulunud aeg	3,88s	3,95s	3,94s

Ülal olevast Tabelist 3 selgub, et *desktop-first* ülesehitusmoodus mobiilses seadmes kasutas veebilehel 9 HTTP päringut ning laadis alla andmeid 668Kb. Kolme mõõtmise keskmine kuvamiseks kulunud aeg oli 3,9s ning veebilehel olevate andmete täielikuks allalaadimiseks.

Tabel 4 Komponent 1 desktop-first arendusmeetodi mõõtmistulemused arvutis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	677ms	671ms	815ms
Kõikide andmete laadimiseks kulunud aeg	2,22s	2,54s	3,09s

Ülal paiknev Tabel 4 näitab, et arvutis tegi veebileht kokku 9 HTTP päringut ning laadis iga kord alla 958kB andmeid. Kolme mõõtmise keskmine kuvamiseks kulunud aeg oli 721ms ning veebilehel olevate andmete täielikuks allalaadimiseks kulus keskmiselt 2,6s

Kõigepealt hakkab autorile silma loomulikult mobiilses seadmes laadimiskiiruste vahe, kui võrrelda *mobile-first* ning *desktop-first* ülesehitusmooduseid. *Mobile-first* veebileht tegi ühe HTTP päringu vähem ning laadis alla kõigest 184Kb andmeid. Samas kui *desktop-first* veebileht laadis mobiilses seadmes alla 668Kb andmeid.

HTTP päringute ning andmemahu erinevuse põhjustas tagataustaks oleva pildifaili laadimine. Nimelt *mobile-first* meetodi puhul ei laetud mobiilses seadmes tagataustaks olevat pildifaili alla. Allalaadimine toimus küll aega *desktop-first* arendusmeetodi puhul.

Autor ei soovita kasutada CSS *display: none* tunnus-väärtus komplekti Chrome veebilehitsejas *desktop-first* meetodi puhul, sest antud brauser laeb alla ka pildifailid, mis on peidetud. Autor katsetas antud *desktop-first* veebikomponenti ka Mozilla Firefox brauseris, kus lehitseja ignoreeris pildifaili, millele oli CSS failis määratud *display: none* ning allalaadimist ei toimunud ning arvatavasti esineb erinevusi erinevates lehitsejates.

Ajaga on toimunud ka teatud mõttes areng veebilehitsejate töös seoses CSS *media-query*'dega. Nimelt mõned aastat tagasi oli probleem paljudes lehitsejates, et brauser laadis alla kõik CSS failis, eelnevates *media-query*'des defineeritud pildifailid (näiteks elemendi tagataust), kuni jõudis täidetud *media-query*'ni. *Desktop-first* meetodi puhul mobiilses seadmes tähendas see seda, et alla laeti ka arvutis (elementide taustana) kuvamiseks mõeldud pidifailid, mis muutis veebilehe laadimise väga aeglaseks. Autor katsetas veebikomponendi tööd seoses *media-query*'des allalaetavate pildifailidega Google Chrome (ver. 49) ning Mozilla Firefox (ver 40) veebilehitsejates, kus ei toimunud ebavajalike ning mittekasutatavate pildifailide allalaadimist.

5.2 Komponent 2 mõõtmistulemused

Autor soovis käesoleva komponenti arendades uurida, kui suur on laadimiskiiruste, HTTP päringute ning andmemahu erinevus *desktop-first* ning *mobile-first* arendusmeetodite vahel. Konkreetselt katsetas autor Javascripti ning HTML sümbolite kasutamise mõju veebilehele.

Tabel 5 Komponent 2 mobile-first arendusmeetodi mõõtmistulemused nutitelefonis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	599ms	610ms	648ms
Kõikide andmete laadimiseks kulunud aeg	943ms	932ms	967ms

Ülal painev Tabel 5 kirjeldab, et mobiilses seadmes tehti 3 HTTP päringut (CSS fail, tagatausta ning HTML fail). Veebileht laadis alla 49KB andmeid ning veebibrauseril kulus selle kuvamiseks keskmiselt 619ms. Kõikide andmete allalaadimiseks kulus kolme katse peale keskmiselt aega 947ms.

Tabel 6 Komponent 2 mobile-first arendusmeetodi mõõtmistulemused arvutis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	569ms	583ms	561ms
Kõikide andmete laadimiseks kulunud aeg	883ms	910ms	900ms

Ülal olevat Tabel 6 vaadates, selgub et *mobile-first* arendusmeetod tegi arvutis 3 HTTP päringut. Samas oli tarvis veebilehe kuvamiseks allalaadida 197KB andmeid. Brauseril kulus keskmiselt 571ms veebilehe kuvamiseks, sama kui kõik andmed laeti alla keskmiselt 897ms.

Tabel 7 Komponent 2 desktop-first arendusmeetodi mõõtmistulemused nutitelefonis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	2,52s	2,64s	2,61s
Kõikide andmete laadimiseks kulunud aeg	2,51s	2,60s	2,60s

Ülal paiknev Tabel 7 näitab, et d*esktop-first* arendusmeetodiga üles ehitatud veebileht tegi laadimisel 9 HTTP päringut (HTML ja CSS failid, jQuery fail, Javascripti fail, kaks tagatausta pildifaili, ning kolm muud pildifaili), mis kokku andis 369KB andmeid. Mobiilses seadmes kulus brauseril kuvamiseks keskmiselt 2,59s ning andmete täielikuks allalaadimiseks kulus keskmiselt 2,57s.

Tabel 8 Komponent 2 desktop-first arendusmeetodi mõõtmistulemused arvutis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	1,37s	1,40s	1,19s
Kõikide andmete laadimiseks kulunud aeg	1,14s	1,37s	1,17s

Tabel 8 näitab, et *desktop-first* ülesehitusmeetod tegi arvutis samuti 9 HTTP päringut, mille andmemaht oli kokku 369KB. Veebilehe kuvamiseks veebilehitsejas kulus keskmiselt 1,32s ning veebilehel olevate andmete täielikus allalaadimiseks kulus keskmiselt 1,23s.

Autorile hakkab kõige pealt silma *mobile-first* ning *desktop-first* ülesehitusmeetoditega veebilehtede kuvamiseks kulunud aegades esinevad suured erinevused. *Mobile-first* veebilehel kulus mobiilses seadmes kuvamiseks keskmiselt 619ms; samas kui *desktop-first* veebileht samas seadmes kulutas keskmiselt 2,59s kuvamiseks. Erinevuse üheks põhjuseks on kindlasti tehtavate HTTP päringute suur vahe -- *mobile-first* puhul on neid 3 kuid *desktop-first* korral on päringuid 9. Samas on ka allalaetavas andmemahus suur vahe: mobiilses seadmes on nad vastavalt 49KB ning 369KB. Erinevused tulevad pildifailide laadimisel. Nimelt *desktop-first* veebileht laeb logo, menüülüliti ning animeeritud kolmnurga visualiseerimiseks alla kolm pildifaili, mis teevad kokku 3 HTTP päringut lisaks, mahuga 55,8KB. Samuti tehakse üks lisapäring mittekasutatava taustapildi laadimisel, mille andmemaht on mobiilses seadmes 186KB. Lisaks tõstavad HTTP päringute arvu kaks välist Javascript faili -- üks autori poolt loodud fail ning üks välises serveris asuv jQuery fail. Mainitud failid teevad kokku 2 HTTP päringut ning nende kogumahuks on 84KB.

Samuti hakkab autorile silma *desktop-first* ülesehitusmooduse puhul on kuvamiseks kulunud aeg ning andmete täielikuks allalaadimiseks kulunud aeg peaaegu võrdsed; kohati on kuvamise aeg isegi suurem. Autori arvates tingib selle Javascripti kasutamine, mis avaldab mõju veebilehe visualiseerimisele.

Sarnaselt eelmisele komponendile kasutas autor ka siin õige tagatausta kuvamiseks *desktop-first* meetodi puhul CSS tunnus-väärtuskomplekti *display: none*.

Laadimiskiiruse parendamiseks soovitab autor uute veebilehitsejate korral kasutada lihtsamate interaktsioonide ning animatsioonide tarbeks CSS3 poolt pakutavaid võimalusi, sest lisaks HTTP päringutele, hoiab see kokku ka andmemahu pealt.

5.3 Komponent 3 mõõtmistulemused

Kolmanda komponendiga soovis autor uurida *image sprite* tehnika kasutamise mõju veebilehe andmemahule, HTTP päringute arvule ning üldisele laadimiskiirusele.

Tabel 9 Komponent 3 mobile-first arendusmeetodi mõõtmistulemused nutitelefonis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	787ms	688ms	651ms
Kõikide andmete laadimiseks kulunud aeg	1,79s	1,88s	1,77s

Tabel 9 sisaldab endas andmeid, mis näitavad, et *mobile-first* arendusmeetodiga veebileht tegi mobiilses seadmes kokku 8 HTTP päringut, mille allalaetud andmemaht oli kokku 162KB. Veebilehe kuvamiseks kulus kolme katse peale keskmiselt 708ms ning kõikide andmete allalaadimiseks kulus keskmiselt 1,81 sekundit.

Tabel 10 Komponent 3 mobile-first arendusmeetodi mõõtmistulemused arvutis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	781ms	851ms	913ms
Kõikide andmete laadimiseks kulunud aeg	1,16s	1,35s	1,26s

Tabel 10 kohaselt, tegi *mobile-first* ülesehitusmeetodiga veebileht arvutis laadimisel 8 HTTP päringut ning laadis kokku alla 221KB andmeid. Veebilehe kuvamiseks kulus keskmiselt 848ms ning kõik failid laeti brauseri poolt alla keskmiselt 1,26s.

Tabel 11 Komponent 3 desktop-first arendusmeetodi mõõtmistulemused nutitelefonis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	811ms	789ms	765ms
Kõikide andmete laadimiseks kulunud aeg	2,83s	2,91s	2,77s

Ülal olev Tabel 11 kirjeldab, et d*esktop-first* arendusmeetodiga veebileht kulutas laadimiseks 14 HTTP päringut ning laadis alla 337KB andeid. Keskmiselt kulus kuvamiseks aega 788ms ning kõikide andmete laadimiseks kulus keskmiselt 2,84 sekundit.

Tabel 12 Komponent 3 desktop-first arendusmeetodi mõõtmistulemused arvutis

	esimene mõõtmine	teine mõõtmine	kolmas mõõtmine
Kuvamiseks kulunud aeg	436ms	369ms	378ms
Kõikide andmete laadimiseks kulunud aeg	1,77s	1,64s	1,87s

Andmed ülal paiknevas Tabelis 12 näitavad, et arvutis tegi desktop-first ülesehitusega veebileht 14 HTTP päringut ja laadis alla 397KB andmeid. Veebilehe esmaseks kuvamiseks kulus keskmiselt 812ms ja kõikide andmete laadimiseks kulus brauseril 1,76s.

Autorile hakkab kõige pealt silma mobiilses seadmes andmete täielikuks allalaadimiseks kuluvate aegade erinevused. *Mobile-first* ülesehitusmeetodiga veebileht laeb kõik andmed alla keskmiselt 1,81 sekundiga, samas kui *desktop-first* veebilehel kulub selleks keskmiselt 2,84 sekundit. Kuna nende kahe veebilehe allalaetava andmemahu erinevus on 175KB, siis tuleneb see vahe tõenäoliselt enamjaolt HTTP päringute arvust ning allalaetavate failide andmemahust (suuremate failide allalaadimine kestab kauem ning brauser saab korraga

ainult teatud arv päringuid teha). *Mobile-first* veebileht, mis kasutas "töötajakaartide" pildifailide laadimisel *image sprite* meetodit, tegi kokku 8 HTTP päringut; samas, kui *desktop-first* veebileht, kus laeti kõik pildid eraldi failidest, tegi laadimisel 14 päringut. CSS faili minimeerimine mobile-first meetodi puhul andis kokkuhoidu faili andmemahust 57% ehk 8KB (*mobile-first* 6KB ning *desktop-first* 14KB). Mõõtmise tulemusena selgus samuti, et *image sprite* meetodi kasutamine antud juhul ei avaldanud suurt mõju veebilehe kuvamise kiiruses mobiilses seadmes. Suured erinevused puudusid ka *mobile-first* ning *desktop-first* veebilehtede vahel arvutis nii lehe kuvamise kiiruses kui ka andmete allalaadimisaegades.

5.4 Analüüsi kokkuvõte

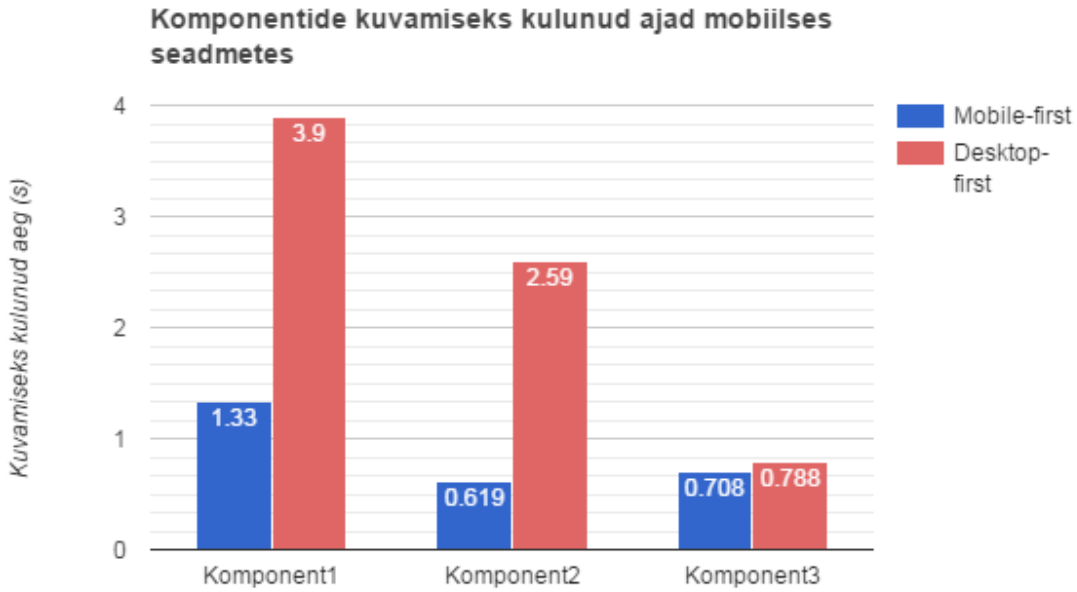
Antud peatükis võrdleb autor ainult mobiilses seadmes saadud mõõtmiste tulemusi. Autor esitab tabelina komponentide keskmised kuvamiskiirused ning keskmised andmete allalaadimiseks kulunud ajad; samuti andmemahu ning HTTP päringute arvu. Seejärel esitab autor mõõtmise tulemused võrdlevate joonistena.

Alljärgnev Tabel 13 sisaldab iga komponendi kohta nutitelefonis tehtud mõõtmistulemuste keskmiseid väärtuseid.

Tabel 13 Kolme komponendi mõõtmistulemuste kokkuvõte nutitelefonides

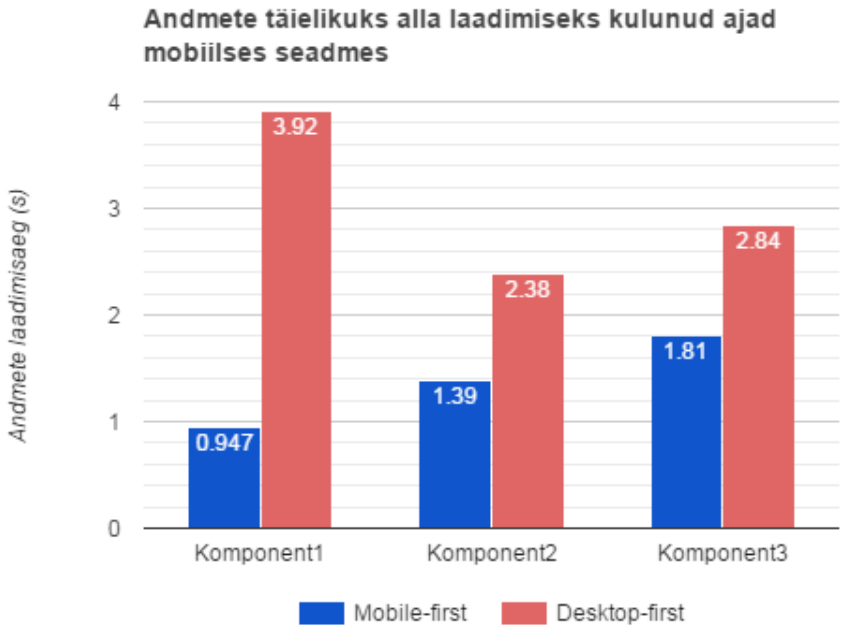
	HTTP päringud	Andmemaht	Kuvamise kiirus	Andmete allalaadimiseks kulunud aeg
Komponent 1				
Mobile-first	8	184KB	1.33s	1.39s
Desktop-first	9	668KB	3.9s	3.92s
Komponent 2				
Mobile-first	3	49KB	0.619s	0.947s

Desktop-first	9	369KB	2.59s	2.57s
Komponent 3				
Mobile-first	8	162KB	0.708s	1.81s
Desktop-first	14	337KB	0.788s	2.84s



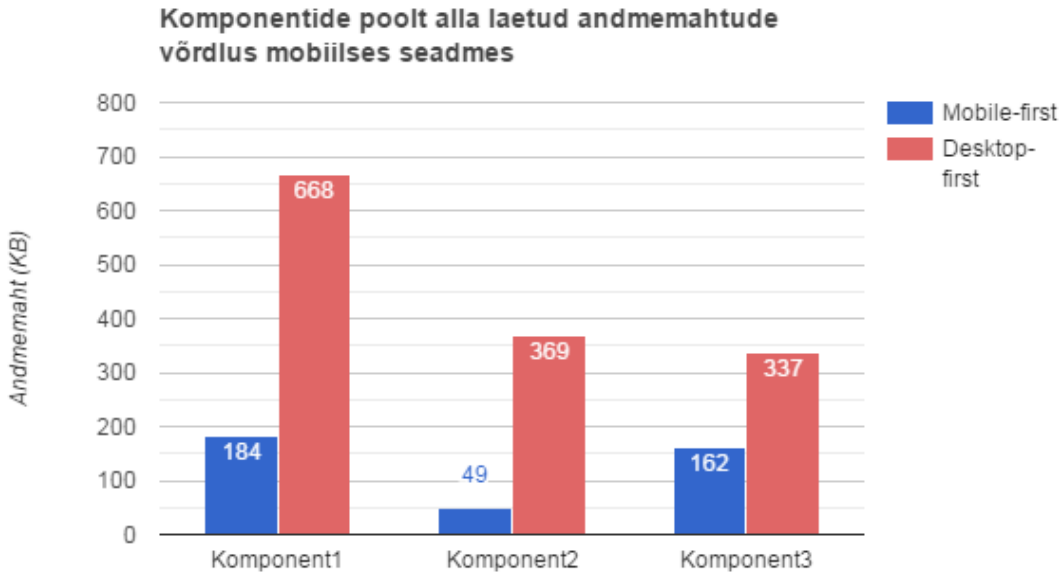
Joonis 10 Komponentide kuvamiseks kulunud ajad mobiilses seadmes (nutitelefon)

Ülalolevalt jooniselt (vt. Joonis 10) selgub, et *mobile-first* arendusmeetodi kasutamine võrreldes *desktop-first* meetodiga, toob kaasa märkimisväärse erinevuse veebilehe kuvamiskiirustes mobiilsetes seadmetes. Eelkõige just esimese ning teise komponendi puhul. Kuvamiskiiruste vahet saab esimese kahe komponendi puhul selgitada andmemahu erinevusega.



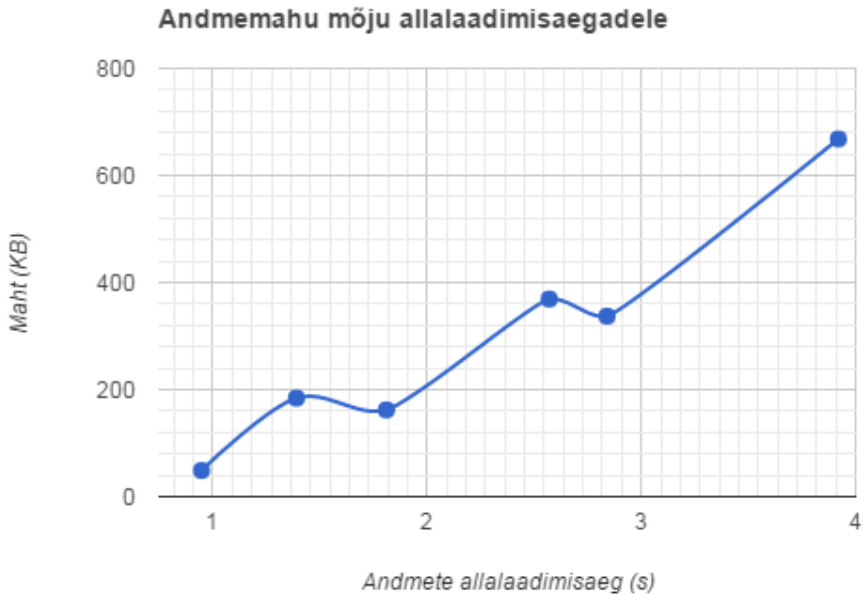
Joonis 11 Andmete täielikuks allalaadimiseks kulunud ajad mobiilses seadmes (nutitelefon)

Nagu kuvamiskiiruste puhul, esineb *mobile-first* ning *desktop-first* arendusmeetodite võrdlusel suur erinevus ka andmete täielikuks allalaadimiseks kulunud aegades mobiilsetes seadmetes (vt. Joonis 11).



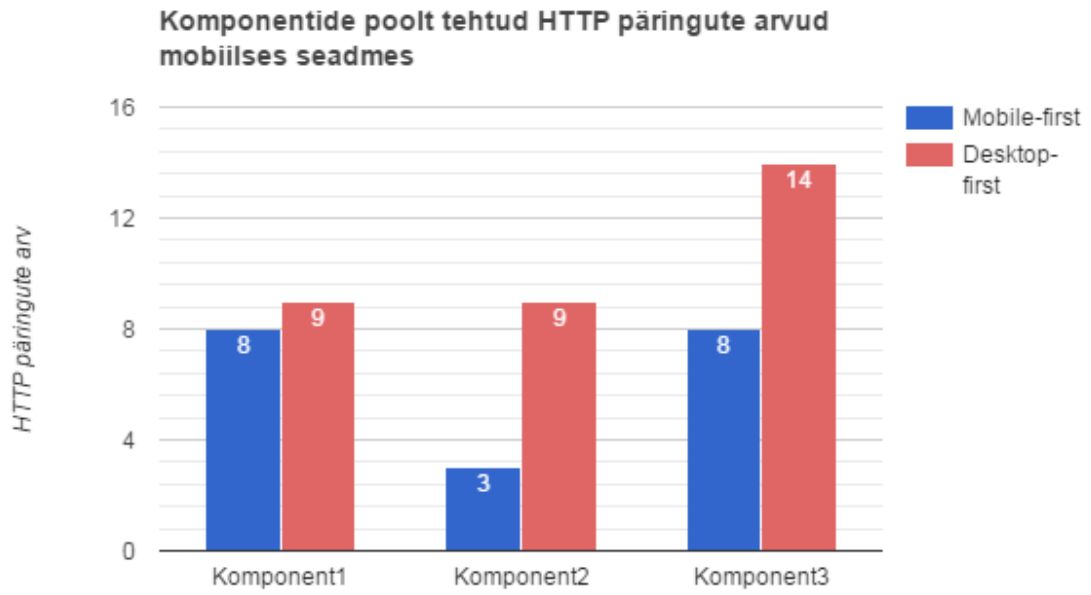
Joonis 12 Komponentide poolt alla laetud andmemahtude võrdlus mobiilses seadmes (nutitelefon)

Mobile-first arendusmeetod tagas kõigi kolme komponendi korral väiksema allalaetava andmemahu hulga (vt. Joonis 12). Kuigi antud komponentide puhul on mõlema ülesehitusmeetodi korral andmemaht suhteliselt väike, esinevad juba väikesel mudelil mitmekordsed vahed laetavates andmemahtudes. See omakorda tõestab *mobile-first* ülesehituse kasulikkuse, just mobiilses seadmes veebilehe laadimiskiiruse parendamisel.



Joonis 13 Andmemahu mõju allalaadimisaegadele

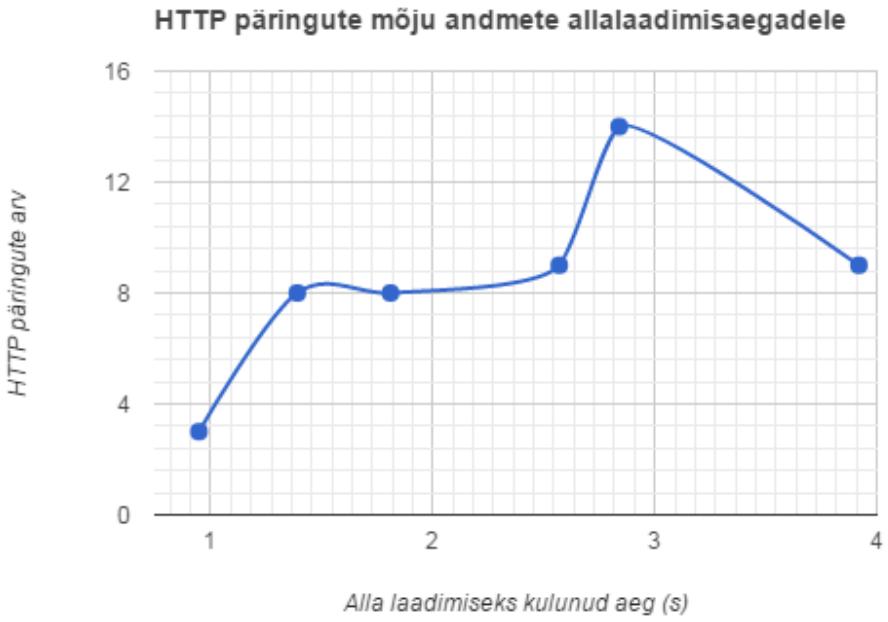
Ülalolev joonis (vt. Joonis 13) näitab, kuidas muutub käesolevas töös tehtud näitekomponentide andmemahtude suurenemisel veebilehe andmete allalaadimiseks kulunud aeg. Joonis näitab üheselt andmemahu suurt mõju andmete allalaadimisaegadele.



Joonis 14 Komponentide poolt tehtud HTTP päringute arvud mobiilses seadmes (nutitelefon)

Kolme komponendi peale kokku tehti *mobile-first* meetodi korral 19 HTTP päringut ning *desktop-first* meetodi korral 32 päringut. Autor kontrollis käesoleva töö teoreetilises osas esitatud ettepanekuid HTTP päringute vähendamiseks töö praktilises osas arendatud näitekomponentide peal. Tulemuseks oli HTTP päringute väiksem arv *mobile-first* meetodi korral võrdluses sama komponendi *desktop-first* variandiga.

Mõõtmistulemuste analüüsi käigus selgub, et HTTP päringute ja allalaadimiseks kulunud aegade vahel ei leidu otsest ja ühest suhet (vt joonis 15). Seda aitab selgitada mõnevõrra asjaolu, et veebilehitsejad suudavad paralleelselt teha mitmeid päringuid korraga ning kiirust mõjutavad pigem failide suurused.



Joonis 15 HTTP päringute mõju andmete allalaadimisaegadele

Kolme näiteks arendatud komponendi korral andis *mobile-first* ülesehitusmeetodi kasutamine kiiremad kuvamisajad ning allalaadimisajad võrreldes *desktop-first* meetodiga. Just mõõtmistulemused olid *mobile-first* meetodi puhul väiksemad nii andmemahu kui ka HTTP päringute arvu poolest. Just andmemaht mõjutas kõige rohkem laadimisaegu, mistõttu tuleks kindlasti kasutada seadmetundlike pilte, CSS3 poolt pakutavaid interaktsiooni- ning animatsioonivõimalusi; samuti võimaluse korral HTML sümboleid pildifailide asemel. HTTP päringute arvu mõju laadimiskiirustele oli liiga ebastabiilne, mistõttu ei saanud autor ühtset kinnitust päringute vähendamise kasulikkusest veebilehe laadimiskiiruse parendamisel.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli uurida, kuidas *mobile-first* ülesehitusmeetodi abil parendada nutitelefonides seadmetundlike veebilehtede kuvamis- ning laadimisaegasid.

Lisaks andis autor käesoleva töö esimeses osas ülevaate veebiarenduse ajaloost ning hetkeseisust; töö teises ning kolmandas osas *mobile-first* meetodi olemusest ning põhimõtetest. Töö neljandas osas arendas autor kolm näitekomponenti, mille autor arendas eesmärgiga testida eelnevalt välja toodud mobile-first meetodi põhimõtete mõju veebilehe laadimiskiirusele. Autor arendas kolm näitekomponenti, kasutades selleks kahte erinevat ülesehitusmoodust: *mobile-first* ning *desktop-first*. Iga arendatud näitekomponendi ülesehitusmeetodi kohta lisas autor põhjaliku ülevaate ülesehitusest ning uuritavatest *mobile-first* arendusmeetodi põhimõtetest.

Töö viiendas osas koostas autor mõõtmised iga komponendi kohta: mõlema ülesehitusmooduse kohta koostas autor mõõtmised nii mobiilses seadmes kui ka arvutis. Näitekomponentide peal läbi viidud mõõtmistulemuste võrdluse käigus leidis autor (küll mõne erandiga), et katsetatud *mobile-first* meetodi põhimõtted tagavad mobiilses seadmes väiksema andmemahu ning kuvamis- ja laadimisaegade kiirenemise. Katsetatud põhimõtetest ei andnud oodatud tulemust HTTP päringute kokkuhoid, sest uuemad veebilehitsejad suudavad paralleelselt teha mitmeid päringuid. Küll aga avaldasid jõudlusele tugavat mõju seadmele kohandatud piltide kasutamine, *mobile-first* ülesehituse kasutamine ning lihtsamate interaktsioonide ja animatsioonide jaoks CSS'i kasutamine jQuery asemel.

Käesolev töö võiks abiks olla mistahes tasemel veebilehe esiliidese arendajatele, kes ei ole veel kursis *mobile-first* arendusmeetodiga ning selle tööpõhimõtetega.

Kasutatud kirjandus

1. Alstad, Kent 2015. *We're Not There Yet: Images, Scripts and Redirects Are Slowing Down the Top Mobile Travel Sites*, allikas: http://www.webperformancetoday.com/2015/12/16/were-not-there-yet-images-scripts-and-redirects-are-slowing-down-the-top-mobile-travel-sites/ (13.02.2016)
2. Archer, James 2015. *Why is it so Easy to Get "Mobile First" Wrong?*, allikas: http://deep.design/mobile-first/ (27.01.2016)
3. Barredo, Alex 2014. *A comprehensive look at smartphone screen size statistics and trends*, allikas: https://medium.com/@somospostpc/a-comprehensive-look-at-smartphone-screen-size-statistics-and-trends-e61d77001ebe#.dk8sqy8qs (14.01.2016)
4. Baumann, Dylan, 2015. *Why you've got to start practicing mobile-first development*, allikas: https://getflywheel.com/layout/start-practicing-mobile-first-development/ (05.02.2016)
5. Bosomworth, Danyl 2015. *Mobile Marketing Statistics Compilation*, allikas: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ (09.01.2016)
6. Deveria, Alexis 2014. *Can I Use*, allikas: http://caniuse.com/ (13.02.2016)
7. Deveria, Alexis 2014. *Can I Use: CSS Animations*, allikas: http://caniuse.com/#search=animation (19.02.2016)
8. DeviceAtlas, 2015. *Mobile Web Traffic Report 2015 Q3*, allikas: https://deviceatlas.com/sites/deviceatlas.com/files/pdf/DeviceAtlas%20-%20Mobile%20Traffic%20Report%20Q3%202015.pdf (14.01.2016)
9. Diaconescu, Adrian 2015. *4.7 inches is the world's most popular mobile screen size*, allikas: http://pocketnow.com/2015/11/20/world-most-popular-mobile-screen-size-resolution (14.01.2016)
10. Dixon & Moe 2015. *HTML Symbols, Entities and Codes - HTML Arrows*, allikas: http://htmlarrows.com/symbols/ (16.03.2016)
11. Everts, Tammy 2013. *Rules for Mobile Performance Optimization*, allikas: https://queue.acm.org/detail.cfm?id=2510122 (15.03.2016)

12. Fisch, Masha 2013. *Mobile-friendly sites turn visitors into customers*, allikas: http://googlemobileads.blogspot.ca/2012/09/mobile-friendly-sites-turn-visitors.html (22.01.2016)
13. Frost, Brad 2012. *Creating a Mobile-First Responsive Web Design*, allikas: http://www.html5rocks.com/en/mobile/responsivedesign/ (16.03.2016)
14. Frost, Brad 2012. *The Many Faces Of 'Mobile-First'*, allikas http://bradfrost.com/blog/mobile/the-many-faces-of-mobile-first/
15. Gesenhues, Amy 2014. *Study: Load Times For 69% Of Responsive Design Mobile Sites Deemed "Unacceptable"*, allikas: http://marketingland.com/study-load-time-69-mobile-sites-deemed-unacceptable-81126 (14.01.2016)
16. Hornor, Justyn, 2013. *10 Ways to Speed Up Your Site*, allikas: http://www.sitepoint.com/10-ways-to-speed-up-your-site/ (25.02.2016)
17. Johansson, Johan 2013. *How To Make Your Websites Faster On Mobile Devices*, allikas: https://www.smashingmagazine.com/2013/04/build-fast-loading-mobile-website/ (19.02.2016)
18. Kissmetrics, kuupäev teadmata. *How Loading Time Affects Your Bottom Line* https://blog.kissmetrics.com/loading-time/?wide=1 (14.01.2016)
19. Kadlec, Tim 2012. *Media Query & Asset Downloading Results*, allikas: https://timkadlec.com/2012/04/media-query-asset-downloading-results/ (18.03.2016)
20. Kwok, Cody 2015. *Ranking change to help you find mobile-friendly sites rolling out today*, allikas: http://insidesearch.blogspot.com.ee/2015/04/ranking-change-to-help-you-find-mobile_21.html (11.01.2016)
21. Lella, Adam 2015. *Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S*, allikas: https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S (11.01.2016)
22. Makino, Takaki & Phan, Doantam, 2015. *Rolling out the mobile-friendly update*, allikas: https://googlewebmastercentral.blogspot.com.ee/2015/04/rolling-out-mobile-friendly-update.html (09.01.2016)
23. Parent., 2014. *Mobile first development:lean & mean*, allikas: http://www.madebyparent.com/mobile-first-development-lean-mean/ (22.01.2016)
24. Ruluks, Sandijs 2014. *A brief history of web design for designers*, allikas: http://blog.froont.com/brief-history-of-web-design-for-designers/ (12.01.2016)

25. Schadler, Ted 2015. *Google's Mobile-friendly Seach Will Bury Your Mobile-unfriendly Sites*, allikas: http://blogs.forrester.com/ted_schadler/15-04-21-googles_mobile_friendly_search_will_bury_your_mobile_unfriendly_sites (11.01.2016)
26. Sterling, Greg 2015. *It's Official: Google Says More Searches Now On Mobile Than On Desktop*, allikas: http://searchengineland.com/its-official-google-says-more-searches-now-on-mobile-than-on-desktop-220369 (09.01.2016)
27. Tuan, Nguyen Anh 2014. *Developers Corner: Maximum concurrent connections to the same domain for browsers* ,allikas: http://sgdev-blog.blogspot.com.ee/2014/01/maximum-concurrent-connection-to-same.html (30.03.2016)
28. Wals, Donny 2015. *Mobile-first is a great workflow*, allikas: http://blog.donnywals.com/mobile-first-is-a-great-workflow/ (05.02.2016)

Summary

Mobile-First Responsive Development Method

The aim of the present Bachelor's thesis was to give an overview of mobile-first responsive web development and how to increase the performance of websites on mobile devices.

In the first chapter of this thesis, the author gave an overview of web development's history and current state. In the second and third chapter, the author described mobile-first development, its basic ideas how to improve the performance of a website and a brief overview how mobile-first development differ from desktop-first development.

In the fourth chapter, the author created two versions (mobile-first and desktop-first) of three different components, typically found on an average website: a header, offered services belt and a contact form. The author also gave thorough descriptions of the development and testing ideas for each component.

In the fifth chapter, the author measured the performance of the components and analysed which of the used mobile-first development performance increasing ideas were effective and which weren't.

The results of the performance test confirmed that using mobile-first development decreased the loading times of these components in mobile devices. The most effective were: responsive images, using CSS instead of jQuery and using mobile-first structure. To the author's surprise, the reduction of HTTP request didn't have a confirmed affect on loading times. This could be explained by the new browsers capabilties to make multiple HTTP requests at the same time.

This thesis could be of help for any front-end web developer who is not familiar with mobile-first development.