Tallinna Ülikool

Informaatika Instituut

Eesrakenduste loomise töövoo parendamine

Bakalaureusetöö

Autor: Kristjan Tammekivi

Juhendaja: Jaagup Kippar

Autor:..... ,,2014

Juhendaja..... ,,2014

Instituudi direktor..... ,,2014

Tallinn 2014

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)

Sisukord

Sissejuhatus

Töövoog – lõpptulemuse saavutamiseks vajalike tööülesannete kindlaksmääratud jada

Eesrakenduste (*Front-End Application*) loomine on tänapäeval üsnagi tähtsal kohal tarkvaraarenduses, arvestades interneti järsku kasvu ja seda, et pea iga uuema operatsioonisüsteemiga kaasneb veebilehitseja. Kui algselt arvati, et internetist väga suurt kasu pole, siis tänapäeval on muutunud brauser rakenduste tarne platvormiks. JavaScript on pea kahekümne aastaga muutunud *rollover* efekti tegevast skriptimiskeelest moodsaks programmeerimiskeeleks, mis võimaldab luua väga paindlikke rakendusi. Seda enam ongi vaja vaadata, et tehtavat tööd ei segaks sellised korduma kippuvad asjad, nagu failide varundamine ja üles laadimine, CSS-i reeglitele *vendor prefix*'ide lisamine jne.

Töö eesmärgiks ongi tuua välja ning vajadusel võrrelda erinevaid meetmeid, millega saab parendada eesrakenduste loomisel kasutatavat töövoogu.

Töö valiku põhjenduseks on autori enda huvi teema vastu ning tahtmine parendada iseenda töövoogu. Suure tõenäosusega on veebiarendus töö autori tulevase elukutse keskmes ning töövoo parendamine aitab kaasa tulevastest tähtaegadest kinnipidamist säästes aega tööde pealt, mida saab palju lihtsamini teha.

Et kõige tähtsam osa rakenduste loomises on koodi kirjutamine, toob autor esimeses peatükis välja erinevad koodiredaktorid. Vaatlemisele tulevad 5 koodiredaktorit mis töö autori hinnangul on kõige populaarsemad ja vajavad rohkem tähelepanu: Brackets, Coda 2, Notepad++, Sublime Text 2/3 ning Vim. Lisaks vaatleb töö autor pistikprogramme, mis aitavad veelgi töövoo hõlbustamist.

Teises peatükis kirjutab töö autor eelprotsessoritest, nimelt CSS-i ja JavaScript'i eelprotsessoritest. Kuigi on olemas ka mõned HTML-i eelprotsessorid siis esimeses peatükis vaadeldud Emmet pistikprogrammi tõttu pole see siiski vajalik, kuna see lihtsustab ja kiirendab HTML-i kirjutamist tohutult. CSS-i eelprotsessoritest kuulub vaatlemisele kolm populaarsemat: Sass, LESS ja Stylus. Neist kolmest vajavad Sass ja Stylus kindlasti kompileerimist, kuid LESS-i võib kaasata koos JavaScript'i failiga lehele. JavaScript'i eelprotsessoritest on välja toodud CoffeeScript, Dart ning TypeScript. Kõik neist erinevad süntaksi poolest täielikult: CoffeeScript laseb kasutada süntaksit mis on sarnane Pythonile

ning Rubyle, Dart on täiesti teistsugune programmeerimiskeel, mida võib kompileerida JavaScript'iks ning TypeScript on mõeldud selleks, et olla võimalikult lähedal JavaScript'ile ning lihtsalt pakkuda juba võimalusi, mis suure tõenäosusega tulevad JavaScript'i paari aasta pärast.

Kolmandas peatükis vaatleb töö autor erinevaid võimalusi oma korduma kippuvate tööde automatiseerimiseks, olgu kas siis kasutada vahetamine või projekti varundamine. Peatüki alguses vaatab töö autor mõnda käsureatööriista, millega kiirendada oma töövoogu, näiteks funktsioonide tegemine selleks, et luua uus projekt olemasoleva malli järgi. Peatüki lõpu poole vaatleb töö autor versioonihaldurit Git'i. Põhjus, miks vaatlemisele ei kuulu mingi teine versioonihaldur, on see, et Git laseb kasutajal töötada ilma ühenduseta serveriga. See tuleb kasuks, kui pole parajasti võrguühendust. Lisaks on Git tohutult kiire võrreldes tsentraliseeritud versioonihaldussüsteemidega, sest igal kasutajal on täielik koopia tervest repositooriumist.

Neljandas peatükis vaatleb töö autor erinevaid brausereid ning nende poolt pakutavaid silumise tööriistu. Töö autor vaatleb kõige lähemalt Chrome'i, sest ülejäänute veebilehitsejate tööriistad ei ole nii paljude võimalustega. Lisaks muule toob autor välja võimaluse, kuidas paremini siluda mobiilseadmetel töötavat eesrakendust.

Viiendas peatükis paneb töö autor kirja enda poolt valitud töövood ning põhjendab nende valikut. Valitud töövooge on kaks: üks töövoog väiksemate projektide jaoks, mis ei võta väga kaua aega ning teine, mis on mõeldud projektide jaoks, mis hõlmab potensiaalselt mitut raamistikku ja teeki.

Käesoleval tööl on kolm lisa. Esimeses lisas on tabel, kus autor on võrrelnud CSS-i eelprotsessorite poolt pakutavaid värvifunktsioone, teises peatükis nende aritmeetikafunktsioone ning kolmandas peatükis toob autor välja tsüklite tegemise eelprotsessorites, põhinedes autoril esinenud probleemile.

1. Ülevaade koodiredaktoritest

Kõige elementaarsem osa eesrakenduste loomisel on koodi kirjutamine. Kuigi seda on võimalik teha kõige lihtsamas tekstiredaktoris, ei mõju see produktiivsusele hästi. Heal koodiredaktoril on olemas võimekus kiirendada koodi kirjutamist ning lihtsustada vigade vältimist.

Koodiredaktorite omadused, mis tagavad kiire ja mugava töö, võivad olla näiteks süntaksi esiletõstmine, sulgude paaritamine, koodi lõpetamine, ridade nummerdamine, võime näidata kahte faili kõrvuti jne.

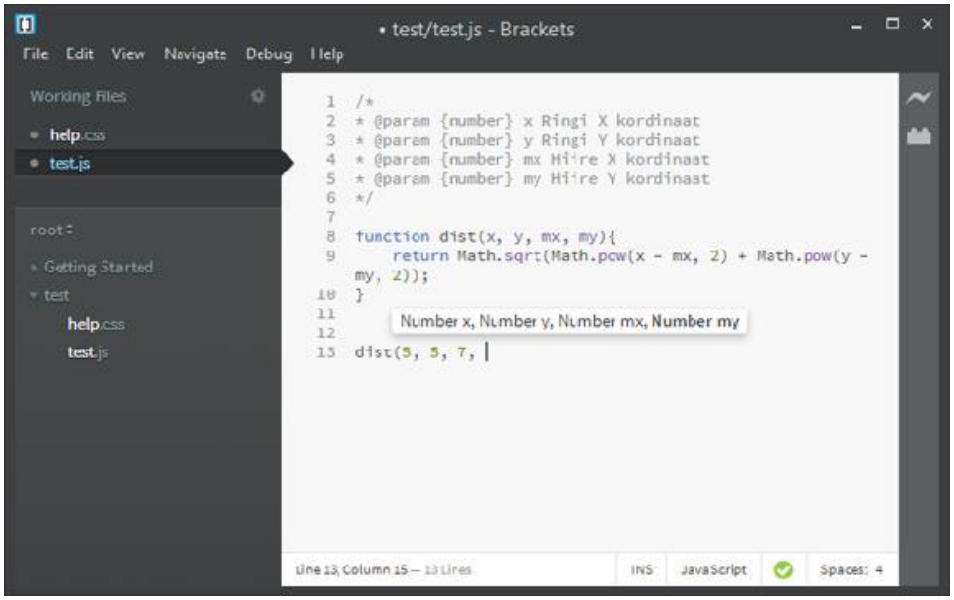
Siin peatükis toob autor välja erinevad koodiredaktorid, nende peamised omadused ning võrdleb neid omavahel. Lisaks toob autor välja ka nende lisad, mis parendavad töövoogu. Välja on toodud 5 koodiredaktorit: Brackets, Coda 2, Notepad++, Sublime Text 2/3 ja Vim. Valiku eelduseks on nende populaarsus ja võime aidata eesrakenduste loomisel. Kolm (Brackets, Sublime Text 2/3 ja VIM) on saadaval mitmel platvormil ning ainult kaks neist on tasulised.

1.1 Brackets

Brackets (ühtlasi tuntud kui Adobe Brackets) on käesolevas töös välja toodud koodiredaktoritest uuemate hulgas. Adobe avalikustas Brackets'i 2012 aastal avatud lähtekoodiga projektina, mille eesmärgiks on luua mitmeplatvormiline koodiredaktor veebilehtede loomiseks.

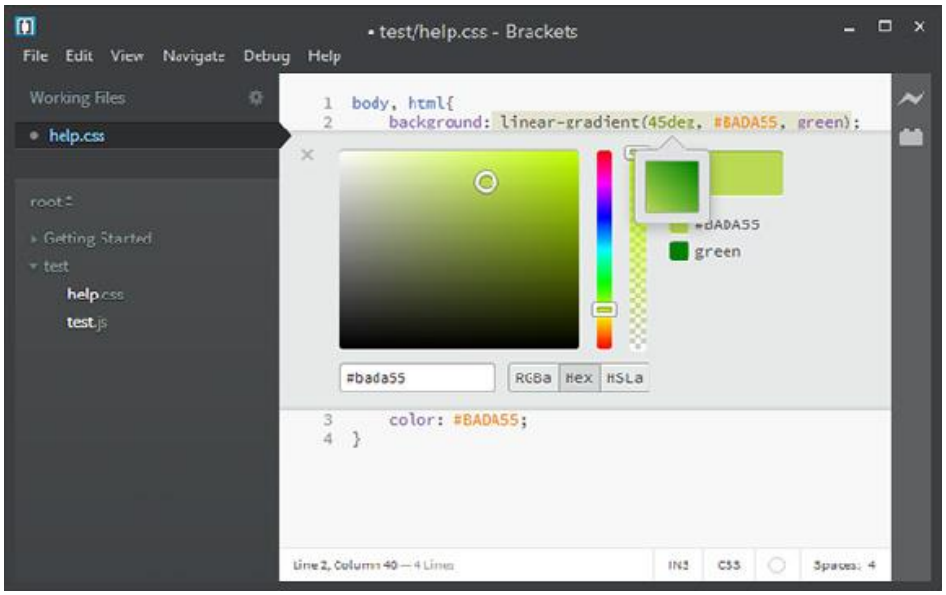
Brackets'i teeb omapäraseks see, et sellega saab koodi trükkides näha, kuidas veebileht muutub. Peale on sellel lihtsasti kasutatav lisade haldur. Lisade hulgas on ka JavaScript'i silur Theseus, mis võimaldab näha reaalajas mis funktsioone ja mitu korda on kutsutud. Ühtlasi laseb see näidata logi sellest, mis parameetrite väärtuseks on funktsioonide kutsumisel ning mis väärtused tagastatakse.

Brackets'il on ka argumentide vihjamine. Sisseehitatud funktsioonidele tulevad automaatselt argumentide tüübid, isetehtud funktsioonidele tuleb juurde panna JSDoc stiilis kommentaare selleks, et lisaks parameetri nimele näidataks ka parameetri tüüpi. (vt Joonis 1)



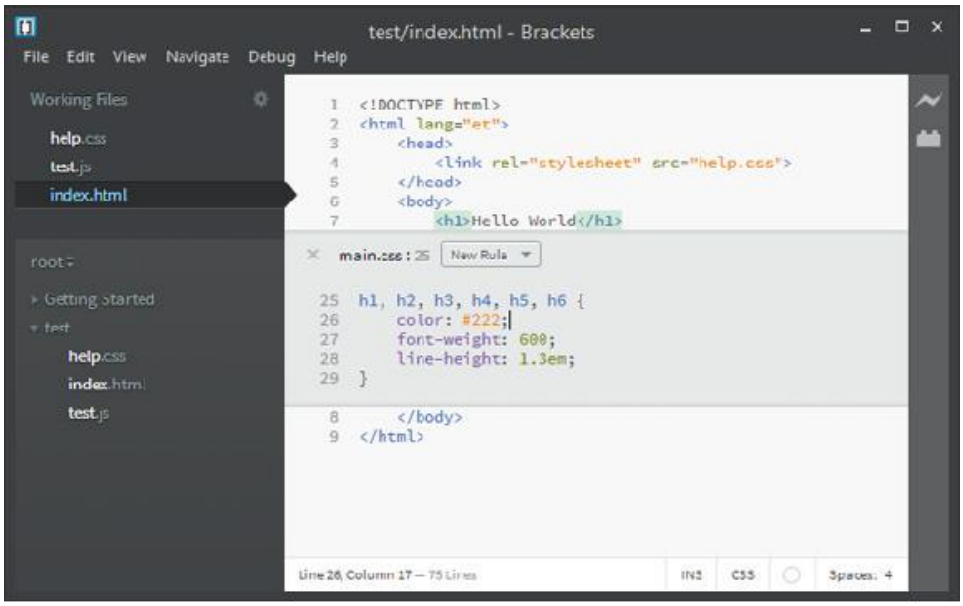
Joonis 1. Brackets'i parameetrite pakkumine

Ühtlasi kiirendab töövoogu see, et on võimalik värve valida läbi graafilise kasutajaliidese. See säästab aega, eemaldades vajaduse otsida sobivat värvi mingi muu rakenduse abil. Gradientide loomisel näeb hiirega selle kohal hõljudes, milline see gradient välja näeb. (vt Joonis 2)



Joonis 2. Brackets'i värviabistaja

Brackets'il on ka sisseehitatud võimalus kiiresti leida dokumentatsiooni juhul kui mingi CSS-i omaduse kasutamine on meelest ära läinud. Peale selle saab lisada CSS-faili reegleid ilma seda faili avamata. (vt Joonis 3)

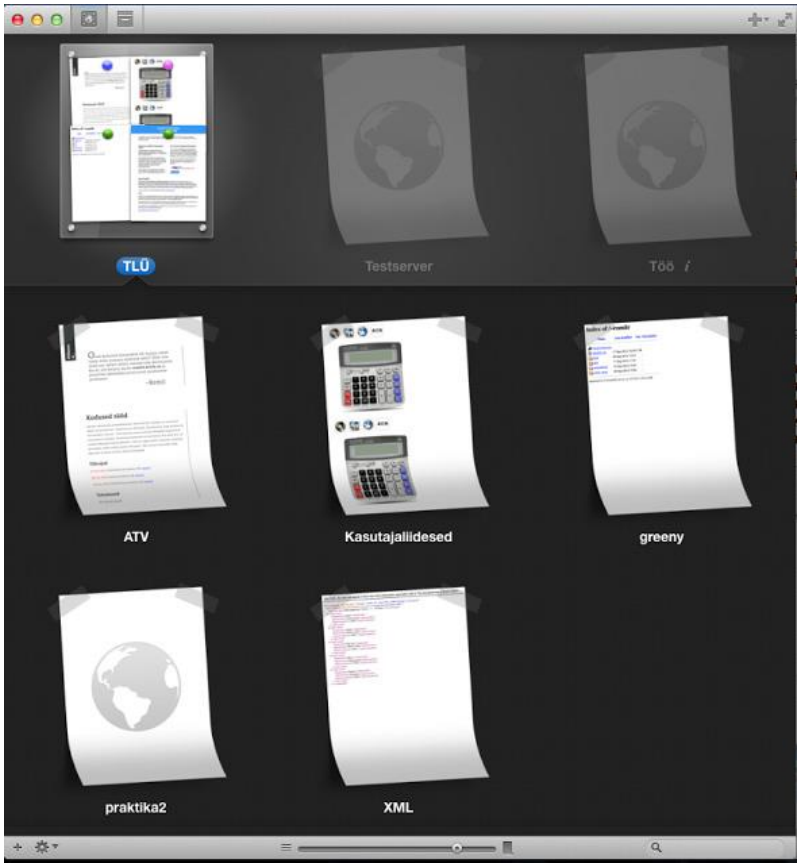


Joonis 3. CSS-i reeglite lisamine Brackets'is

1.2 Coda 2

Coda 2 on loodud firma Panic poolt edasiarendusena nende algsele Codale. Tegu on veebiarenduskeskkonnaga, mis on loodud ainult Mac OS X-i jaoks. Originaalne Coda loodi aastal 2007, Coda 2 tuli välja 2012 aasta esimesel poolel.

Codal on sisseehitatud failide edastamise võimekus, mis pakub lihtsat kaugfailide haldust kasutades FTP, SFTP, FTP+SSL või WebDAV protokolle (vt Joonis 4). Koodi kirjutades võib koheselt külgribale välja tuua dokumentatsiooni mingi HTML-i, CSS-i, JavaScript'i või PHP süntaksi osa kohta.

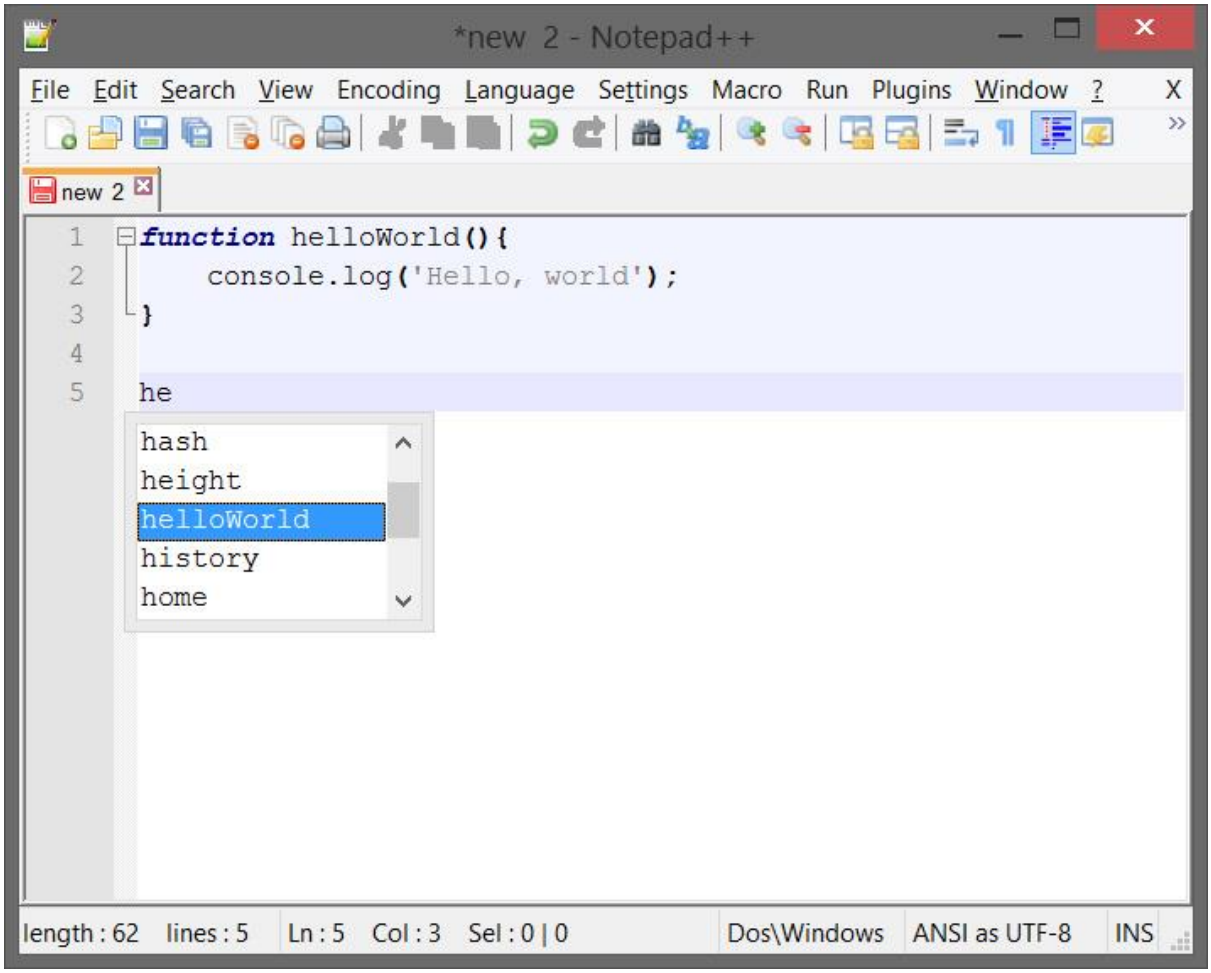


Joonis 4. Coda serverite halduse vaade

1.3 Notepad++

Notepad++ on tasuta ning avatud lähtekoodiga koodiredaktor mõeldud kasutamiseks Microsoft Windows operatsioonisüsteemidel. Notepad++ on äärmiselt populaarne arendusvahend (Fitzpatrick, 2010), see on võitnud kahel korral parima tööriista arendajatele auhinna sourceforge.net projektide seast (sourceforge.net, 2009).

Notepad++ pole kõige võimalusterohkem koodiredaktor siin nimekirjas, kuid siiski võimaldab selliseid asju, nagu sulgude paaritamine (ühe sulu avades paigutatakse kohe ka lõpetav sulg) ning sõnade lõpetamine. Need võimalused küll on algselt keelatud, kuid sätetest saab neid lubada. (vt Joonis 5)



Joonis 5. Notepad++'i süntaksi pakkumine

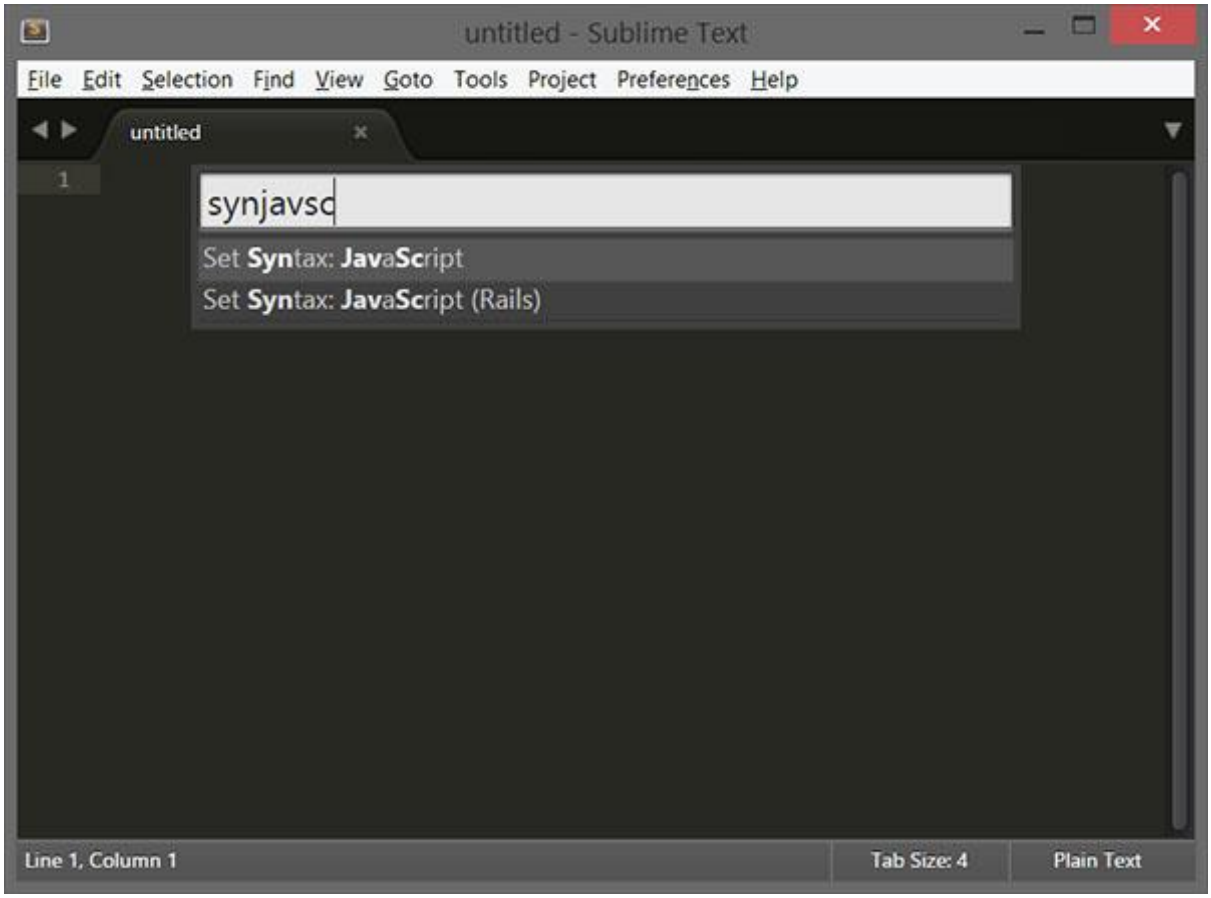
1.4 Sublime Text 2/3

Sublime Text on üks kahest käesolevas töös nimetatud tasulistest koodiredaktoritest, kuid õnneks laseb on olemas tasuta prooviversioon, millel on kõik täisversiooni võimalused, kuid iga paarikümne salvestuse järel tuleb hüpikaken, mis soovitab osta täisversiooni. Aprill 2014 seisuga maksab üks litsents 70 USD-d, kuid see-eest on tegu kasutajale kehtiva litsentsiga, mitte arvutile kehtiva. See tähendab, et kasutaja võib selle paigaldada kõigile oma valduses olevatele arvutitele. Käesoleva töö kirjutamise ajal on Sublime Text 3 veel beetastaadiumis, kuid enamus selle võimalusi on samad.

Sublime Texti versioon 1.0 avaldati aastal 2008 ning on sellest ajast saadik saanud käesoleva töö autori hinnangul üheks populaarseimaks koodiredaktoriks. Sublime Texti võimaluste hulka kuulub Command Palette, Goto Anything ja mitmikuvalikud (Sublime Text, 2013).

1.4.1. Command Palette

Command Palette (avatav klahvidega Shift + Control/Command + p) on Sublime Texti tekstiaken, kuhu saab trükkida erinevaid käske. Käskude hulgas võib näiteks kasutada paigaldatud pistikuid või muuta kõik tühikud tabulaatoriteks ja vastupidi. Lisaks on võimalik muuta projekti süntaksit (Sublime Text, 2013). (vt Joonis 6)



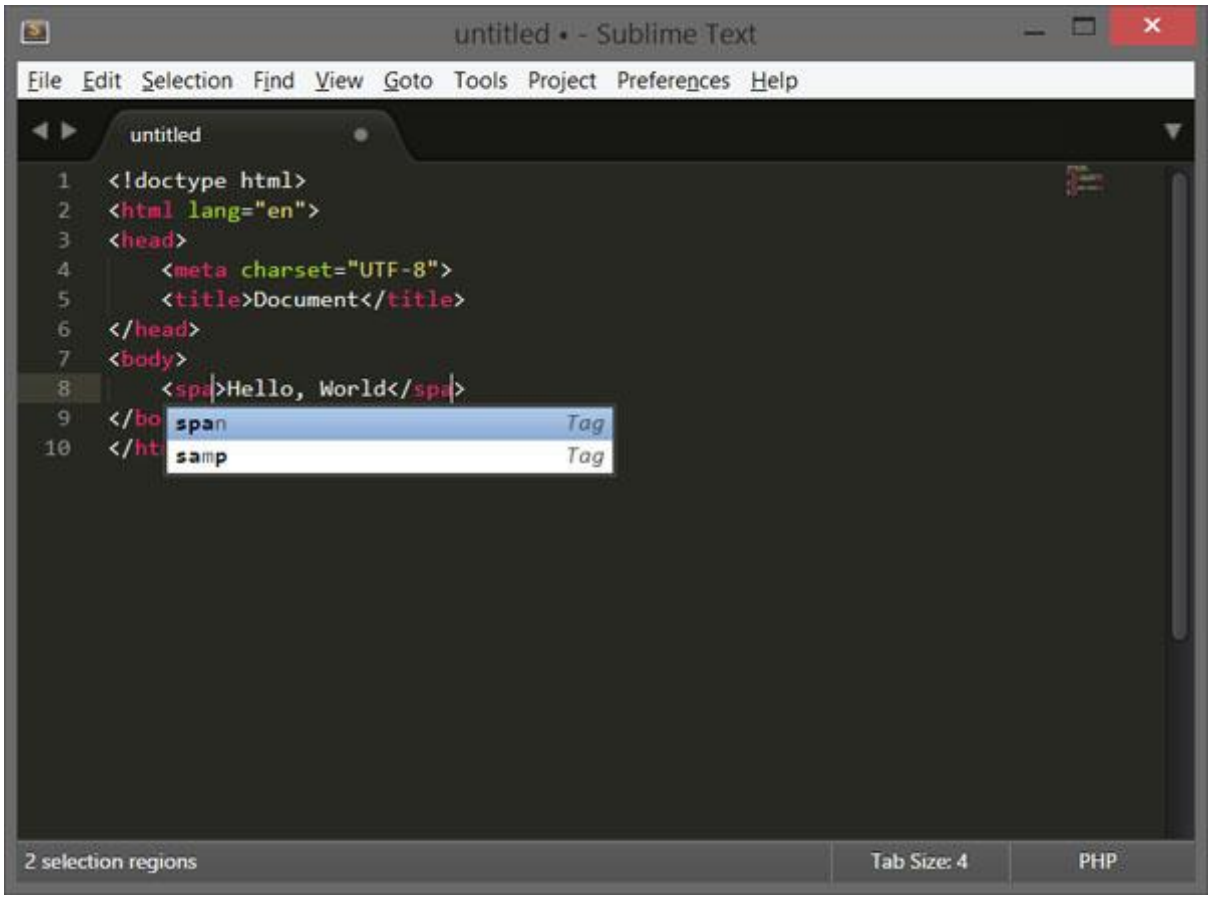
Joonis 6. Sublime Texti Command Palette

1.4.2. Goto Anything

Goto Anything (avatav klahvidega Control/Command + p) on sarnane Command Palette'iga, kuid võimaldab liikuda failide ja failiridade vahel. Trükkides sinna osa failinimest saab väga kiiresti vahetada, mis faili kallal töö käib. Kirjutades sinna koolon ning reanumber, minnakse kursoriga sellele reale, kirjutades @-märgi millele järgneb osa mingist sümbolist, minnakse selle sümboli juurde (näiteks JavaScript'i funktsiooni definitsiooni juurde). JavaScript'i silumisel annab konsool tihti välja täpselt mis realt viga tuli, kasutades Goto Anythingi on võimalik kiiresti sinna reale navigeerida ja viga ära parandada (Sublime Text, 2013).

1.4.3. Mitmikuvalik

Mitmikuvalik lubab korraga valida mitu eraldi paiknevat sõna ning neid muuta. See tuleb kasuks näiteks ümberfaktoreerimisel, kui valida funktsiooni definitsioon ning selle kutsumine, või siis mingi HTML-i märgendi muutmiseks, saab valida algava märgendi ja lõppeva märgendi ning mõlemat korraga muuta. Selle kasutamiseks tuleb valida mingi sõna ning vajutada Control/Command + d, mille järel näidatakse järgmist kohta, kus see sõna esineb. Seejärel saab trükkida harilikult, aga muudetakse sõna mitmes kohas korraga (Sublime Text, 2013). (vt Joonis 7)



Joonis 7. Mitmikuvalik Sublime Textis

1.4.4. Pistikud

Sublime Textil on väga suur hulk võimalusi juurdeandvaid pistikprogramme, pistikuhalduri statistika järgi üle 2200 erineva paki (wbond, 2014).

1.4.4.1. Package Control

Package Control pistikprogramm on mõeldud pistikute haldamiseks. Sellel on oma repositooriumid ning lihtsustab ja kiirendab teiste pistikute paigaldamist. Kuigi pistikute paigaldamine on võimalik ka ilma selleta, võib pistiku otsimine, allalaadimine ja õigesse kausta paigutamine võtta rohkem aega, kui asi väärt on (wbond, 2014).

1.4.4.2. Nettuts+ Fetch

Nettuts+ Fetch (edaspidi lihtsalt Fetch) on pistik, mis lihtsustab teekide ja raamistike allalaadimist. Sisseehitatud on kaks pakki: jQuery ja HTML5 Boilerplate, kuid neid saab lisada minnes *command palette'*ist *Fetch: Manage*'i alla. Fetch'i kasutamiseks tuleb kõigest siseneda *command palette'i* ning vastavalt valida, kas on soov alla laadida ainult ühte faili (*Fetch: File*), mille puhul laetakse soovitud fail hetkel aktiivsesse faili, või siis tervet pakki (*Fetch: Package*), mille järel küsitakse, kuhu kasutaja soovib allalaetud faile lahti pakkida (Way, 2012).

1.5 Võrdlus

Kõigil eelnimetatud koodiredaktoritel on omad eelised ning miinused. Brackets sobib ideaalselt just kujunduse paikasättimiseks, sest võimaldab trükkides juba näha kõige mõju veebilehele. Lisaks Theseuse abil on JavaScript'i silumine väga mugav. Samas on projekt veel oma nooruses ning paljud võimalused on veel puudu (näiteks mitu kursorit, ei võimalda kahte faili kõrvuti lahti hoida ja paljud muud).

Coda 2 võimaldab väga lihtsasti hallata serverites asuvaid faile, pakub väga meeldiva kasutajaliidese ja koodi automaatne täiendamine töötab väga hästi. Samas on tegu tarkvaraga, mis on ainult saadaval Mac OS X platvormil ning litsents maksab 75 USD-d. Saadaval on ka demoversioon, kuid see kestab ainult 7 päeva.

Notepad++ on olnud aastaid populaarsuse tipus, kuid napib võimaluste poolest. Sellel on küll põhilised koodiredaktoritööriistad, nagu süntaksi esiletõstmine, kahe faili kõrvuti redigeerimine ja sulgude paaritamine, kuid koodi automaatne täiendamine JavaScript'I kirjutades on äärmiselt kehv. Lisaks on Notepad++ saadaval ainult Windowsil.

Sublime Text on väga tugev iga külje alt vaadates: koodi automaatne täiendamine töötab väga hästi, see on saadaval Windowsil, Linuxil ja Mac OS X-il (seejuures on Windowsile

võimalik tõmmata *portable* versiooni, mis lubab Sublime Texti käivitada näiteks mälupulgalt olles võõra arvuti taga), pistikprogramme on väga palju saadaval. Käesoleva töö autori hinnangul on ainus negatiivne külg see, et tegu on tasulise tarkvaraga. Kuigi on saadaval ka tasuta versioon, võivad aeg-ajalt ilmuvad sõnumid segada töövoogu. Tegu on autori eelistatud koodiredaktoriga, mille litsentsi on ta ka ostnud.

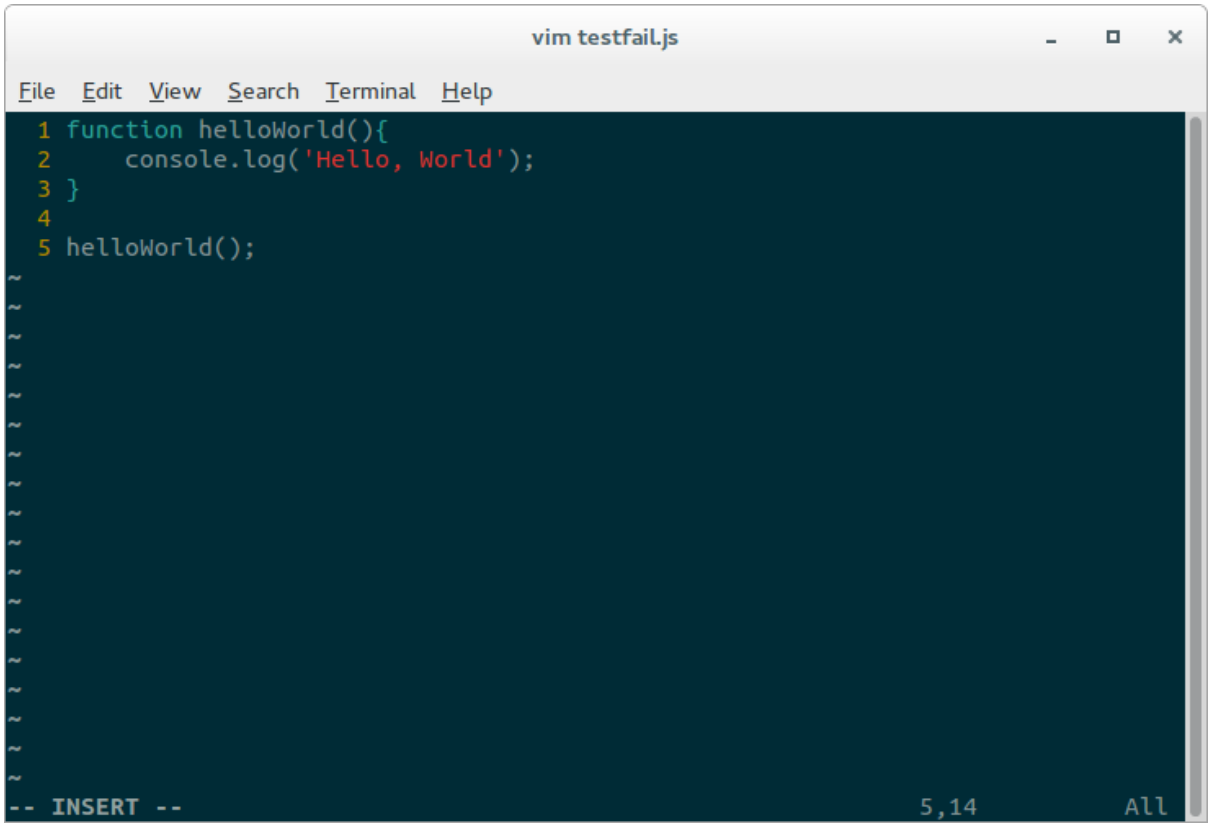
Vim on vanim nimetatud koodiredaktoritest ning osales rivaalsusest Emacs-iga, mida tunti kui *Editor Wars*'i. Vim on vägagi võimalusterohke, kuid selle kasutamine on vägagi keerukas. Käesoleva töö autor küll kasutab Vim'i, kuid ainult serverite konfiguratsioonide muutmiseks, mitte eesrakenduste arendamiseks.

1.6 Vim

Vim on käesolevas töös nimetatud redaktoritest vanim. See on edasiarendus algsest vi redaktorist (VIM tähendab VI iMproved) ning lisab asju, mida koodiredaktorist ootaks (sealhulgas süntaksi värvimist). Kui Vim algselt loodi, siis selle funktsioonid olid nii edasiarenenud, et protsessorid jõudsid seda vaevu jooksutada. Ajapikku siiski on protsessorid jõudnud järele selle nõudmistele ning tänapäeval suudavad arvutid jooksutada ka kõige ressursinõudlikumaid Vim'i käske kiiresti (Robbins, Hannah, & Lamb, 2008).

Vim on tuntud selle poolest, et seda on raske õppida. Sellel on erinevad režiimid ja käskude andmine käib eksklusiivselt klahvivajutustega (sealhulgas kopeerimine ja kleepimine). Selle tõttu vajab see palju pühendumist ja aega, et kasutada efektiivselt. Tõenäoliselt kasutatakse seda kõige rohkem käsurealt, kuid on ka loodud graafilised implementatsioonid.

Vim'il on erinevad režiimid, neist tähtsamad on normaalrežiim, käsurežiim ning sisestusrežiim. Vim'i algselt käivitades satub kasutaja normaalrežiimi, kust edasi saab ta sisestada käske, et vahetada, millises režiimis olla. Sisestusrežiimi lülitamiseks peab kasutaja vajutama kas klahvi Insert või i-d. Seejärel saab kasutaja vabalt trükkida. Tagasi normaalrežiimi minekuks peab vajutama *Escape* klahvi. Seejärel faili salvestamiseks ja redaktorist lahkumiseks peab trükkima käsu *:wq* (*write-quit*) (WikiBooks, 2013). (vt Joonis 8)



Joonis 8. Vim'i koodiredaktor sisestusrežiimis

1.7 Pistikud

Kõik käesolevas töös mainitud koodiredaktorid võimaldavad kasutada pistikuid (*plugin*'e). Pistikprogrammid on vähemalt mingis osas loodud selleks, et lihtsustada programmeerija tööd. Selles peatükis vaatleb autor erinevaid pistikprogramme, mis toimivad mitmel koodiredaktoril.

1.7.1. Emmet

Emmet, algse nimega Zen Coding, on pistikprogrammide hulk, mis võimaldavad kiiresti kirjutada HTML-i ning CSS-i. Projekti autor on Vadim Makeev ning edasi on arendanud alates aastast 2009 Sergey Chikuyonok (Chikuyonok, 2009). Emmet on saadaval kõikidele käesolevas töös mainitud koodiredaktoritele. Küllaga VIM-i pistik on loodud kolmanda osapoole poolt (Emmet, 2013).

Emmet'i kasutamine on lihtne. Kasutaja trükib sisse mingi lühendi, näiteks *ul>li*5*, ning vajutades koodiredaktorile vastavat klahvikombinatsiooni laiendatakse see õigeteks HTML elementideks (antud juhul tuleks *unordered list* viie *list* elemendiga). Ühtlasi on võimalik

kasutada ka CSS-i lühendeid. Spikker erinevatest lühenditest asub aadressil http://docs.emmet.io/cheat-sheet/.

1.7.2. Autoprefixer

Autoprefixer on pistikprogramm, mis võimaldab kiiresti lisada kõik vajalikud *vendor prefix*'id CSS faili. See töötab tehes päringu veebilehele *caniuse.com*, kus on kirjas erinevad HTML-i, CSS-i ning JavaScript'i osad ning mis versioonid neid toetavad mis eesliidetega. Autoprefixer'i pistik on loodud Sublime Texti ning Brackets'i koodiredaktoritele, kuid projekt ise on mõeldud kasutamiseks eelprotsessorina tööriistadega nagu Grunt (Sitnik, 2014).

1.7.3. JSLint & JSHint

JSLint on koodi analüüsi tööriist mõeldud JavaScript'i koodi kvaliteedi kontrolliks. JSHint on JSLint'i harutatud versioon, mis keskendub vähem koodi stiilile võrreldes JSLint'iga. JSLint otsib lisaks programmi tööd segavatele vigadele ka sellistele asjadele nagu muutujate deklareerimine funktsioonide alguses ning tabulaatoreid tühikute asemel kasutamine (Kovalyov, 2014).

Mõlemad tööriistad on põhiliselt mõeldud veebiteenusena, kuid on olemas ka pistikprogramme neist, mis eemaldavad vajaduse pidevalt kopeerida koodi veebilehitsejasse. Brackets'isse on JSLint sisseehitatud, ülejäänud käesolevas töös nimetatud koodiredaktorite jaoks on olemas allalaetavad pistikprogrammid. JSHint'i pistikprogrammid on olemas kõigile eelnimetatud koodiredaktoritele.

2. Eelprotsessorid

Siin peatükis vaatleb käesoleva töö autor erinevaid eelprotsessoreid, mis lihtsustavad ja kiirendavad veebilehtede loomist. Vaadeldavad eelprotsessorid saab jagada kaheks: CSS-i eelprotsessorid ning JavaScript'i eelprotsessorid. CSS-i eelprotsessoreid on mitmeid, kuid töö autor valis välja kolm populaarsemat: Sass, LESS ning Stylus. Ka JavaScript'il on vaadeldud kolme populaarsemat eelprotsessorit: CoffeeScript, Dart ning TypeScript.

2.1 CSS

Siin alampeatükis vaatleb töö autor kolme populaarsemat eelprotsessorit, Sass'i, LESS-i ning Stylust. Nendest kolmest on kõige populaarsem, teiseks tuleb Sass ning viimasel kohal on Stylus (Coyier, 2012).

Kolm nimetatud eelprotsessorit on algselt mõeldud kasutamiseks erinevalt: Sass on loodud kasutamaks Ruby on Rails'i raamistikuga, Stylus sobib väga hästi Node.js'iga jooksutamiseks ning on eelkõige mõeldud kasutamiseks HTML faili kaasatud JavaScript'i failiga.

2.1.1. Sass

Sass (Syntactically Awesome Stylesheets) on laadistiku keel algselt arendatud aastal 2006 Hampton Catlin'i ja Nathan Weizenbaum'i poolt (Painter, 2013). Sass lisab harilikule CSS-ile muutujad, võimaluse ise luua funktsioone või *mixin*'e (funktsioonid, mis võivad tagastada CSS-i reegleid) ning laseb muuhulgas kasutada reeglite pesitsemist.

Sass kasutab faililaienditena nii .sass-i kui ka .scss-i. Erinevus nende vahel on süntaksis, .sass failides võib jätta ära loogelisi sulge ja semikooloneid, .scss failides märgitakse *mixin*'e sümboliga @mixin, .scss-is võrdusmärgiga jne (Sass, 2014).

2.1.1.1. Compass

Compass on Sass'i raamistik, mis lisab mixin'e kiirendamaks CSS-i kirjutamist. Näiteks on võimalik kasutada mixin'e, mis paigutavad õiged *vendor prefix*'id reeglitele (Eppstein, 2013). (vt Koodinäide 1)

```
SCSS:
@import "compass";
div{
  @include border-radius(4px);
```

```
}
Kompileeritav CSS:
div {
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
  border-radius: 4px;
}
```

Koodinäide 1. Compassi kasutamine

2.1.2. LESS

LESS avaldati aastal 2008 ning on vanem, kui Sass'i scss süntaks. Kui Sass oli algselt mõeldud kohandamaks CSS-i täielikult, siis LESS oli CSS-ile väga sarnane ja lihtsalt täiendas seda erinevate funktsionaalsustega (The Less Core Team, 2014).

LESS erineb käesolevas töös teistest CSS-i eelprotsessoritest selle poolest, et on võimalik ka lisaks serveripoolsele kompileerimisele kaasata lehele JavaScript'i fail, mis teisendab kliendipoolel kogu LESS koodi valiidseks CSS-iks (The Less Core Team, 2014).

2.1.2.1. LESS Hat

LESS Hat on LESS-i raamistik, mis sarnaselt Compassile Sass'i jaoks lisab LESS-ile *mixin*'id *vendor prefix*'ide paigaldamiseks (Brzek, 2014).(vt Koodinäide 2)

```
LESS:
@import "lesshat.less";
div{
  .border-radius(4px);
}
Kompileeritav CSS:
div{
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
  border-radius: 4px;
}
```

Koodinäide 2. LESS Hat'i kasutamine

2.1.3. Stylus

Stylus loodi 2011 aasta alguses Roman Komarovi poolt. (Komarov, 2014) ning jätkab sealt, kust Sass'i algne süntaks pooleli jäi: eesmärgiga teha CSS-i süntaks võimalikult lühikeseks. Stylus on eelkõige loodud Node.js'i moodulina kasutamaks on-the-fly kompileerimisega, see tähendab, et kui kasutaja nõuab läbi *node'i* mingit Styluse faili, siis see kompileeritakse enne kasutajale saatmist (learnboost, 2014).

2.1.3.1. Nib

Ka Stylusel on oma raamistikud, neist populaarseim on Nib (Node.js Modules, 2014). Sarnaselt teistele eelnimetatud eelprotsessorite raamistikele, lisab ka see *mixin*'id vastavate *vendor prefix*'ide lisamiseks

```
Stylus:
@import "nib"
div
  border-radius 4px
Kompileeritav CSS:
div{
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
  border-radius: 4px;
}
```

Koodinäide 3. Nib'i kasutamine

2.1.4. Võrdlus

Enamik süntaksist on nimetatud eelprotsessoritel sarnane (mõningate hinnangute järgi on 80% sarnane) ja ainult keerukamad võimalused erinevad (Arlt, 2013).

2.1.4.1. Üldine süntaks

Sass'i on võimalik kirjutada kahte süntaksit kasutades. Vanem süntaks on failides, mille faililaiend on .sass. See võimaldab jätta ära loogelisi sulge ning semikooloneid. Lisaks on see lühem selle poolest, et sõnu *function* ning *mixin* ei kirjutata välja, vaid asendatakse sümbolitega. Käesoleva töö autori hinnangul on see selle tõttu vähemloetavam ning soovitab selle asemel kasutada uuemat .scss süntaksit. Sellel põhjusel on kõik eesolevad näited välja jätnud .sass süntaksi.

Sass'i ning LESS-i süntaks on üsnagi loomulik inimestele, kes on varem CSS-iga kokku puutunud. Ka Styluse süntaks võib olla sarnane, kuid koolonite, semikoolonite, komade ning loogeliste sulgude kasutamine on vabatahtlik. Taaskord, käesoleva töö autori hinnangul siiski abistavad need koodi lugemisel ning on soovitatav neid siiski kasutada. Igal juhul on iga valiidne CSS kood valiidne Sass'i, LESS-i või Styluse kood, mis lihtsustab preprotsessorile üleminekut eelnevalt valmisolevatelt CSS failidelt.

2.1.4.2. Muutujad

Kõigis kolmes eelnimetatud CSS-i eelprotsessoris on olemas võimalus kasutada muutujaid. Muutujate eesmärk on see, et saab hoida mitmes kohas kasutatavat väärtust (näiteks värvi) ühes kohas, nii et kui on vaja seda muuta, siis pole tarvis tervest dokumendist otsida.

Sass'is omistatakse muutujaid pannes muutuja nimele ette $-märgi. Pärast muutujanime on koolon ning siis selle väärtus ning kõige lõpuks semikoolon. LESS-is käib muutujatele väärtuste andmine sama moodi, ainult et $-märgi asemel kasutatakse @-i. Stylusega aga tuleb panna kõigepealt muutujanimi, siis võrdusmärk ning siis mingi väärtus. Kõigi kolme eelprotsessoriga käib muutuja kasutamine sama moodi, tuleb vaid panna muutuja nimi sinna, kus me selle väärtust soovitakse (Sass, 2014) (The Less Core Team, 2014) (learnboost, 2014). (vt Koodinäide 4)

```
Sass:
$taust: #BBB;
body{
  background: $taust;
}

LESS:
@taust: #BBB;
body{
  background: @taust;
}

Stylus:
taust = #BBB
body
  background $taust


Kõigil kolmel juhul kompileerub järgnev CSS (värv võib muutuda #BBB-
st #bbbbbb-ks):
body{
  background: #BBB;
}
```

Koodinäide 4. Muutujad erinevates eelprotsessorites

Töö autori hinnangul on muutujate kasutamine Sass'is ja LESS-is parem, sest muutujate määramine toimub juba tuttava kooloni märgiga, võrdusmärgi kasutamine ei tundu loomulik CSS-i kirjutamisel. Lisaks Sass ja LESS sunnivad kasutama mingid eesliidet, sel juhul ei aja segamini, et mis on muutuja ja mis on CSS-is kasutatav sõna (näiteks sõnadega tähistatud värvid). Styluses on võimalik kasutada muutujatel dollarimärgiga algavaid muutujaid, kuid arendaja peab ise teadlikult valima, et seda teha.

Ühtlasi tasub ära mainida, et Sass'is puudub muutujatel skoop: Muutuja väärtuse muutmisel mingi bloki sees muutub see täielikult. Kui LESS-is ja Styluses seda teha, siis muutuja väärtus muutub ainult selle bloki sees.

2.1.4.3. Pesitsemine

Kõigil kolmel eelnimetatud CSS-i preprotsessoril on sama süntaks üksteise sees paiknevate reeglite kirjeldamiseks. Pesitsemine võimaldab ühe selektori sisse paigutada lisaks selle reeglite ka teisi selektoreid. See annab parema ülevaate koodist ning võimaldab lihtsamini koodist aru saada (Sass, 2014) (The Less Core Team, 2014) (learnboost, 2014). (vt Koodinäide 5)

```
Sass, LESS ning Stylus:
.ymbritsev{
  background: #BBB;
  a{
    color: blue;
    &:hover{
      background: red;
    }
  }
}

Kompileeritav CSS:
.ymbritsev {
  background: #BBB;
}
.ymbritsev a {
  color: blue;
}
.ymbritsev a:hover {
  background: red;
}
```

Koodinäide 5. Reeglite pesitsemine

2.1.4.4. Aritmeetika

Tihti on vaja CSS-is teha arvutusi eri väärtustega. Neid arvutusi võib peast teha, aga kui mingi sisendväärtus peaks muutuma peab arvutuse uuesti tegema (lisaks on alati võimalus, et inimene arvutab valesti). CSS-is on juba olemas *calc* funktsioon, aga neljandik maailma internetikasutajatest ei saa veel seda kasutada oma veebilehitseja tõttu (caniuse.com, 2014). Lihtsam on kasutada eelprotsessorite aritmeetikat. Kõigis kolmes CSS-i eelprotsessoris on olemas lisaks liitmisele, lahutamisele, korrutamisele ning jagamisele ka keerulisemad matemaatilised funktsioonid. Matemaatiliste funktsioonide nimekirjad erinevad eelprotsessorite vahel (learnboost, 2014) (The Less Core Team, 2014) (sass-lang, 2014). (vt LISA 1)

2.1.4.5. Funktsioonid ja mixin'id

Funktsioonid CSS-i eelprotsessorites võimaldavad manipuleerida ühte väärtust. Lisaks sisseehitatud funktsioonidele on Sass'is ja Styluses võimalik ka luua funktsioone ise. Sisseehitatud funktsioonidest võib esile tuua *lighten* ja *darken* funktsioonid, mis vastavalt helendavad või tumendavad mingit värvi etteantud määral.

mixin'id on funktsioonidele sarnased, kuid tagastavad mitte ainult väärtusi, vaid lausa ühe või mitu CSS-reeglit. Üks hea kasutus *mixin*'i jaoks on *vendor prefix*'ide lisamine. Näiteks on võimalik teha *mixin* border-radius'e lisamiseks (Sass, 2014) (The Less Core Team, 2014) (Stylus, 2014). (vt Koodinäide 6)

```
Sass:
@mixin border-radius($radius){
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
  border-radius: $radius;
}

.ymar{
  @include border-radius(10px);
}

LESS:
.border-radius(@radius){
  -webkit-border-radius: @radius;
  -moz-border-radius: @radius;
  border-radius: @radius;
}

.ymar{
  .border-radius(10px);
}


STYLUS:
border-radius(n)
  -webkit-border-radius n
  -moz-border-radius n
  border-radius n

.ymar
  border-radius(10px)

Kompileeritav CSS:
.ymar {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;
}
```

Koodinäide 6. Mixin'id CSS-i eelprotsessorites

Siinkohal on töö autori arvates kõigil kolmel eelprotsessoril üsnagi sarnane süntaks. Ainus asi, mis Sass'i ja Styluse kasuks loeb on see, et saab ise funktsioone teha, mis töötlevad ainult ühte väärtust.

2.1.4.6. Värvide transformeerimine

CSS-i kirjutades esineb tihti vajadus kasutada sama värvi erinevaid variatsioone. Eelprotsessorite abil saab paigutada alusvärvi muutujasse ning seejärel manipuleerida sellega. Kõikidel eelnimetatud eelprotsessoritel on olemas vastavad sisseehitatud funktsioonid, kuid Stylusega saab lisaks ka liita ja lahutada värve omavahel kasutades liitmis- ja lahutamismärke. (vt Koodinäide 7)

```
Sass:
$alus: #666;
h1{
  color: lighten($alus, 10%);
}

LESS:
@alus: #666;
h1{
  color: lighten(@alus, 10%);
}

Stylus:
alus = #666
h1
  color lighten(alus, 10%)
  /*
  Styluses võib teha ka järgnevalt:
  color alus + 10%
  */

Kompileeritav CSS:
h1{
  color: #757575;
}
```

Koodinäide 7. Värvifunktsioonide kasutamine CSS-i eelprotsessorites

Värvide transformeerimine käib kõigil kolmel eelprotsessoril sarnaselt ning tihti jagavad funktsioonid nime. Kõige rohkem värvimuutmisfunktsioone on LESS-il, teiseks tuleb Sass ning viimaseks Stylus. Kuigi enamasti on funktsioonid sarnased, leidub siiski väikseid erinevusi: Styluses värvitooni muutmine lihtsa liitmisega (näiteks #FF00BB + 180deg). (vt LISA 2)

2.1.4.7. Tsüklid

Vahel tuleb ette, et on olemas mitu klassi ning igale ühele neist tuleb teha sarnase kujunduse, mis erinevad üksteisest ainult kergelt. Käesoleva töö autoril on selline probleem ette tulnud: oli vaja teha 10 klassi, millel kõigil on erinevat hallitooni taust ning hiirega selle kohal olles muutub taustavärv erinevat värvi. Seda saaks lahendada mitut moodi, kõige algelisem neist oleks lihtsalt eraldi välja kirjutada kõik need klassid ning nende stiilid, kuid see oleks aeganõudev töö ning kui peaks tekkima vajadus millegi muutmiseks, peab kõike uuesti tegema. Veidi lihtsam oleks kirjutada kõik klassid välja, kuid kasutada muutujaid ning värvide transformeerimise funktsioone, kuid lisaks on veel olemas kolmas võimalus, kasutada tsükleid. Tsüklitega saab genereerida ükskõik kui palju klasse lihtsalt ühte numbrit muutes. Sass'i ja Styluse eelprotsessoritel on see lahendatud üsnagi sarnaselt, kuid kasutab rekursiooni, mixin'i seest kutsutakse iseennast. Käesoleva töö autori hinnangul on selle tõttu LESS-i versioon tsüklitest keerukam lugeda (The Sass Way, n.d.) (Stylus, 2014) (The Less Core Team, 2014). (vt LISA 3)

2.2 JavaScript

JavaScript on peagi juba 20 aastat vana, kuid sellest hoolimata pole see lähedal veel täiuslikkusele. JavaScript'i keelel on mitmeid omapärasid, mis võivad produtseerida vigu, kui nendega mitte arvestada. Lisaks on jätkuvalt puudu mõned võimalused, mida keelest võiks oodata. Mõned nendest [võimalustest] tulevad ECMAScript 6-ga, kuid selle ilmumiseni on veel aega. Igal juhul võib vigadesse takerdumine ning nende jahtimine rikkuda töövoogu ning koodi kirjutamise loomingulist protsessi. Seniks aga on võimalik kasutada eelprotsessoreid.

Siin peatükis vaatab töö autor kolme populaarseimat eelprotsessorit: CoffeeScript, Dart ja TypeScript. Neist kõige populaarsem on CoffeeScript, GitHub'i repositooriumite otsingul tagastatakse 2136 projekti (GitHub, 2014), teiseks tuleb Dart 1932 projektiga (GitHub, 2014) ning lõpuks, kõige uuem nendest kolmest, on TypeScript 383-ga (GitHub, 2014).

Kõik need kolm erinevad üksteisest suurel määral. CoffeeScript on inspireeritud Ruby ja Pythoni keeltest ning pakub lühemat ning selgemini loetavat süntaksit. Dart on hoopiski uus programmeerimiskeel mis transkompileerub JavaScriptiks, loodud Google'i poolt selleks, et tulevikus võib-olla asendada JavaScript. TypeScript on arendatud MicroSofti poolt ning on mõeldud abistamiseks suuremõõtmeliste rakenduste loomisel (Maharry, 2013).

2.2.1. CoffeeScript

CoffeeScript on programmeerimiskeel mis kompileerub JavaScript'iks. Selle süntaks on vähem verboosne, kui JavaScriptil ning on inspireeritud Ruby ja Pythoni keeltest. Selle kasutamise eeliseid on mitmeid. Esiteks on selle süntaks lühem, seeläbi on vaja kirjutada vähem koodi (tulenev kood on umbes kolmandiku lühem, kui vastav JavaScript'i kood). Koodi lühendatakse mitut moodi:

- Puuduvad semikoolonid käskude lõpus;
- Muutujaid ei deklareerita, need genereeritakse automaatselt;
- Puuduvad loogelised sulud, selle asemel kasutatakse treppimist;
- Pole vaja kirjutada *return* lauseid, viimane lause tagastatakse automaatselt;
- Puudub märksõna function, selle asemel kasutatakse noolt „->".

Teiseks, nagu käesoleva töö autor tõi välja oma seminaritöös, on JavaScript'is olemas vahendeid, mille kasutamine ei ole soovitatav. CoffeeScript ei lase neid vahendeid kasutada ja läbi selle kasvab toodetava rakenduse kvaliteet. Hoolimata sellest, et CoffeeScript on lühem, kui JavaScript ja toimub transkompileerimine, pole kompileeritav JavaScript märgatavalt aeglasem, kui käsitsi kirjutatud JavaScript (Ali, 2013).

2.2.2. Dart

Dart on Google'i poolt loodud JavaScript'i alternatiiv, kuid siiski on võimalik seda ka transkompileerida JavaScript'iks. Dart'i sihiks on tulevikus asendada JavaScript'i kui veebi *de facto* keelt. Dart'is võib lisada muutujatele tüüpe, kuid pole kohustuslik.

Dart'i hetkel toetab vaid Chromeiumi eriversioon millel on kaasas Dartiumi virtuaalmasin ning kompileeritav JavaScript on ainult 75 protsendilise jõudlusega võrreldes käsitsi kirjutatud JavaScriptiga (Florian Loitsch, 2012). Kuna puuduvad tõendid, et teised brauserid hakkavad toetama Dart'i siis töö autor ei näe piisavat põhjust selle kasutusele võtuks.

2.2.3. TypeScript

Erinevalt CoffeeScriptist on TypeScript JavaScript'i ülemhulk. See tähendab, et iga valiidne JavaScript'i kood on toimib ka TypeScripti koodina. TypeScript on loodud suuremõõtmeliste rakenduste arendamiseks JavaScriptis. See võtab kasutusele osasid EcmaScript 6 osasid nagu staatilist tüüpimist ja klasse. TypeScript on eelkõige loodud kasutamiseks Visual Studioga,

milles toimib nii süntaksi esiletõstmine kui ka koodi automaatne täiendamine, kuid on ka olemas süntaksi esiletõstmise failid Sublime Textile, Vim'ile ning Emacs'ile (Bloch, 2012).

Sarnaselt CoffeeScriptiga aitab TypeScript vähendada JavaScript'i omapärast tingituid vigu, näiteks võrreldes erinevaid andmetüüpe tuleb enne teha mõlemad muutujad sama tüüpi väärtuseks (vt Koodinäide 8)

```
var numberYks = 1;
var stringYks = "1";

if(numberYks == stringYks){
  // JavaScript'is toimib, kuid TypeScriptis ei kompileeru
}
```

Koodinäide 8. TypeScriptis topeltvõrdusmärgi mitte kompileerumine

2.2.4. Hinnang JavaScript'i eelprotsessoritele

Kõik kolm eelprotsessorit erinevad üksteisest vägagi suurel määral, nii süntaksi kui ka eesmärgi poolest. CoffeeScript on neist kõige populaarsem ja pakub kõige lühemat ja loetavamat süntaksit. Dart on loodud JavaScript'i asendama ja kuigi on tulemas ECMA standard Dart'ist (Ecma International, 2014), pole hetkel selle kasutamiseks suurt põhjust, arvestades seda, et kompileeritav JavaScript on märkimisväärselt aeglasem, kui käsitsi kirjutatud JavaScript. TypeScript on arendatud selleks, et pakkuda lisavõimalusi nagu klassid ja staatiline tüüpimine JavaScript'ile. Kompileeritav JavaScript on jõudluse poolest sarnane käsitsi kirjutatud JavaScriptiga.

Kahel JavaScripti eelprotsessoril on olemas nendega seostatud arenduskeskkonnad: Dart'il on selleks Eclipse'il põhinev Dart Editor ning TypeScriptil on Visual Studio vajalike pakkidega. Mõlemad hõlbustavad nende keeltega töötamist. CoffeeScriptil pole oma spetsiifilist arenduskeskkonda.

Igal juhul aitab JavaScript'i eelprotsessorite kasutamine vältida vigu, sest kompilaator annab teada, kui midagi valesti on ning töötavat JavaScript'i faili sel puhul ei asendata. Lisaks näiteks CoffeeScriptiga programmeerides ei ole võimalik teadmatult teha sellist viga, nagu kahe võrdusmärgi kasutamine kolme võrdusmärgi asemel, sest kõik kahekordsed võrdusmärgid kompileeritakse automaatselt kolmekordseteks.

3. Automatiseerimine

Kui on valida üks asi, mis kõige rohkem töövoogu katkestab, siis see on korduma kippuvate tööde manuaalne täitmine, olgu selleks kas uue projekti alustamine, failide kompileerimine või projekti varundamine.

3.1 Aliased ning funktsioonid

Käsureal töötamine võib vajada palju trükkimist, aga kõike seda on võimalik lühendada. Aliasi võib kasutada mingi tihedalt kasutatava käsu lühendamiseks, näiteks ssh ühenduse tegemise (SS64.com, 2014). (vt Koodinäide 9)

```
alias tammekivi="ssh kristjan@tammekivi.ee"
#Edaspidi saab kasutada serveriga ühenduse
#loomiseks ainult käsku "tammekivi"
#(See näide on loodud Bash'i käsurea jaoks)
```

Koodinäide 9. Aliase loomine ssh ühenduse tegemiseks

Funktsioone on võimalik kasutada koos muutujatega. Üks väga mugav funktsioon oleks käsk, mis looks uue kausta ning seejärel siseneks sinna (SS64.com, 2013). (vt Koodinäide 10)

```
function take {
  mkdir $1
  cd $1
}
#Nüüd saab kasutada lihtsalt käsku "take kaustanimi"
#selle asemel, et luua kaust antud nimega ning sellesse vahetada
```

Koodinäide 10. Funktsioonide kasutamine Bash-käsureal

Funktsioone või aliasi võib valmistada lihtsalt käsureale sisse trükkides, kuid sel juhul need püsivad ainult seni, kuni käsurida on lahti. Terminali sulgedes sel juhul kustuksid kõik funktsioonid ning aliased. Selle vältimiseks on võimalik paigutada aliased ja funktsioonid konfiguratsioonifailidesse (gnu.org, 2014).

Veel üks hea kasutus funktsioonidele veebiarendajal oleks uue projekti alustamiseks. Võib teha kausta, kuhu saab panna kõik uues projektis vaja minevad failid (näiteks kohandatud HTML5 Boilerplate). Töö autori väiksematesse projektidesse kuuluvad tüüpiliselt 4 kindlat faili: tühi CSS fail, tühi JavaScript'i fail, normalize.css fail, mis ühtlustab brauserite kujundust ning HTML fail, kus on kõik eelnimetatud failid õigesti kaasatud. Seejärel trükkides funktsiooni koos uue kausta nimega luuakse see kaust ja kopeeritakse vajalikud failid ümber.

3.2 SSHFS

SSHFS võimaldab ühendada eemalasuva serveri kohaliku arvuti failisüsteemi osaks. See eemaldab vajaduse kasutada FTP-d või muud failiedastusprotokolli failide liigutamiseks siis, kui mingi fail muutub. Lisaks on sellel see eelis tavalise SSH ees, et saab kasutada kohalikus arvutis olevaid tööriistu, näiteks mõnda graafilist koodiredaktorit.

Linuxil võib olla olenevalt distributsioonist see kohe alguses kaasas või siis paigaldatav paketihalduriga. Windows'il on olemas sarnane rakendus nimega win-sshfs (mladenovic.ml@gmail.com, n.d.).

SSHFS-i läbi mingi eemalasuva serveri kausta ühendamiseks tuleb kasutada käsku *sshfs*. Käsu järel tuleb kasutada serveri aadressi (ette võib kirjutada vajadusel kasutajanime), seejärel kooloni, mille järel võib olla serveripoolne kaust, mida kasutada ning siis kaust, kuhu see failisüsteem ühendada (Fenski, n.d.). (vt Koodinäide 11)

```
sshfs kristjan@tammekivi.ee: mount
```

Koodinäide 11. sshfs-i abil ühendamine

SSHFS-i lahtiühendamiseks tuleb kasutada käsku *fusermount*, millele järgneb lipp u ning kaust, millelt ühenduse eemaldada (Fenski, n.d.). (vt Koodinäide 12)

```
fusermount –u mount
```

Koodinäide 12. sshfs-i failisüsteemi lahtiühendamine

Kui serverisse ilma tunnelita ei saa, tuleb kõigepealt teha ssh ühendus portide edasisuunamisega ning siis ühendada sshfs läbi selle. (vt Koodinäide 13)

```
ssh -fL 127.0.0.1:5555:greeny.cs.tlu.ee:22 kris88@lin2.tlu.ee -N &&
sshfs kris88@127.0.0.1: mount -p 5555
```

Koodinäide 13. sshfs läbi tunneli

3.3 z

Käsurealt töötades on tihti vaja liikuda erinevate projektide ja kaustade vahel, kuid tihti vajab see päris mitme cd käsu abil. Siin juhul tuleb abiks *directory hopper* ehk siis kausta hüpik nimega z. Pärast selle paigaldamist hakkab see jälgima kasutaja kausta vahetusi ning seejärel reastab neid kausti *frecency* (algoritm mis kombineerib sagedust ning hilisust) järjekorras. Edasi on võimalik kaustade vahel vahetada kasutades käsku z, millele järgneb regulaarne avaldis või osa kausta nimest, millesse on vaja liikuda (rupa, 2013).

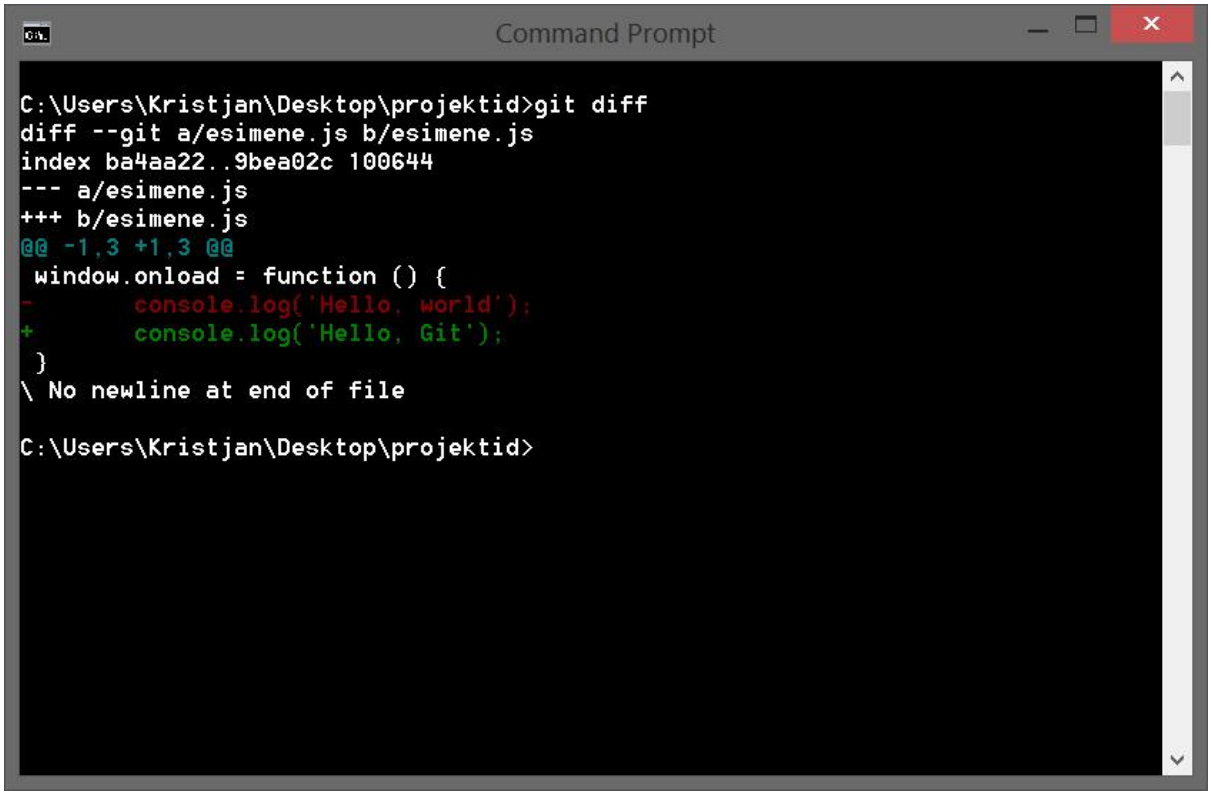
3.4 Git

Versioonihaldus on üsnagi tähtis igal mittetriviaalsel projektil. Kui koodi muutmisel peaks tekkima vigu saab lihtsalt tagasi minna eelneva versiooni juurde tagasi. Lisaks saab koodi lihtsalt liigutada ühest arvutist teise (näiteks lauaarvutist sülearvutisse ja vastupidi). Git sobib selleks väga hästi, sest tegu on hajutatud versioonihaldustarkvaraga, mis tähendab, et kõigil on täielik koopia tervest lähtekoodi ajaloost. See võimaldab arendada isegi siis, kui hetkel puudub ühendus serveriga, näiteks sõites bussiga.

Lisaks sellele, et tegu Git'iga saab töötada ka ilma võrguühenduseta, tuleb kasuks selle kiirus. Võrreldes tsentraliseeritud versionihaldustarkvaraga Subversion, on selle kiirus enamasti kordades suurem, tingitud osalt sellest, et ei hoiustata mitte muudatusi, vaid terveid faile. Ainult kahes kohas on Subversionil eelis: algne repositooriumi kloonimine, sest igal kasutajal on täielik koopia repositooriumist, ning hoiustavate failide maht, sest Git'il hoiustatakse täielikke faile, mitte nende erinevusi (Git, 2014). (vt Joonis 9)

Joonis 9. Git'i võrdlus Subversioniga

Kui Git on arvutisse juba paigaldatud, siis saab uut Git'i repositooriumi alustada väga lihtsalt, kasutades vaid käsku *git init*. Seejärel, kui mingit faili on vaja jälgida, tuleb kasutada käsku *git add failinimi*. Edaspidi *commit*'i tegemiseks saab teha *git commit –m "Commit'i sõnum"*, mille järel luuakse punkt, millesse on alati võimalik tagasi tulla. Pärast faili muutmist on võimalik kasutada käsku *git diff*, mis näitab ära, mis on jälgivates failides muutunud ridade kaupa. (vt Joonis 1)



Joonis 10. Git'i kasutamine Windows'i Command Prompt'ist

Git'i kasutamaõppimine võib olla keeruline, sest erinevalt enamikest teistest versioonihaldussüsteemidest on Git'is kasutusel mitte mõnikümmend käsku, vaid ligemale 150-le (põhinedes nimekirjale, mille saab kasutades käsku *git help –a*). Git'i kodulehel on olemas põhjalik materjal, mis tutvustab kasutajale täpsemalt nii Git'i tööpõhimõtteid kui ka selle käske[1].

3.5 Yeoman

Yeoman avaldati aastal 2012 Google I/O konverentsil ning koosneb kolmest osast:

- yo valmistab ette kõik osad uue rakenduse jaoks, kirjutab Grunt'i konfiguratsiooni faili ning ning haldab Grunt'i ülesandeid ja Bower'i sõltuvusi mida projektiks võib vaja minna;
- Grunt'i kasutatakse projekti ehituseks ja testimiseks;
- Bower'it kasutatakse sõltuvuste haldamiseks.

Uue projekti alustamiseks saab kasutada käsku *yo*, sealt edasi saab paigaldada *Ruby on Rails*'i laadseid generaatoreid ning siis kasutada neid generaatoreid. Näiteks selleks, et

[1] http://git-scm.com/book/en/Getting-Started

kasutada *Ember.js* generaatorit saab kasutada käsku *yo ember*. Seejärel *yo* valmistab ette uue projekti, olenevalt generaatorist küsib, et kas kasutaja tahab kasutada osasid nagu *Modernizr* või *Sass.* Tulemuseks on suur hulk katalooge ja faile. Ettesätitud projekti vaatamiseks saab kasutada *grunt serve* käsku, mis käivitab serveri, avab veebilehitseja sellele lehele ning hakkab jälgima muutusi failis (Yeoman, 2014).

Kõik skriptifailid kontrollitakse automaatselt üle JSHint'iga selleks, et veenduda parimate praktikate jälgimises, pildid optimeeritakse kasutades OptiPNG'd ning JPEGTran'i selleks, et vähendada nende suurust ning CoffeeScript'i ja Compass'i failid kompileeritakse automaatselt (Yeoman, 2014).

4. Brauseri tööriistad

Veebilehitsejad on viimase paarikümne aastaga muutunud tohutult. Enam pole see ainult võimalus vaadata teksti ja pilte üle interneti, vaid on muutunud rakenduste tarne platvormiks. See tähendab, et kirjutatav kood pole enam nii lihtne, nagu oli teksti värvi muutmine hiirega selle kohal olemisega. CSS kood on tihti vägagi keerukas ning vahel tuleb kontrollida, kas ikka õiged elemendid saavad õiged stiilid kätte, vajadusel tuleb neid stiile veidi kohendada ja kontrollida, kas nüüd on saavutatud tulemus.

Uuemates brauserites on saanud juba harilikuks see, et on võimalik inspekteerida DOM-i (*Document Object Model*'i) puud ning elementidele kehtivaid CSS-i kirjeid. Samuti on tüüpiliselt brauserites saadaval JavaScript'i konsool, kuhu on võimalik väljastada muutujate väärtusi silumisel, või siis käivitada funktsioone manuaalselt trükkides lihtsalt funktsiooninime koos käivitavate sulgudega. Siin peatükis vaatan erinevate brauserite pakutavaid võimalusi ning lisaks veel pistikprogramme, mis aitavad kaasa veebiarendusele. Vaadeldavad brauserid on Chrome, Firefox, Internet Explorer ja Safari. Kuna Opera põhineb Chromiumil, siis on selle tööriistad peaaegu identsed ja Operat eraldi ei vaatle.

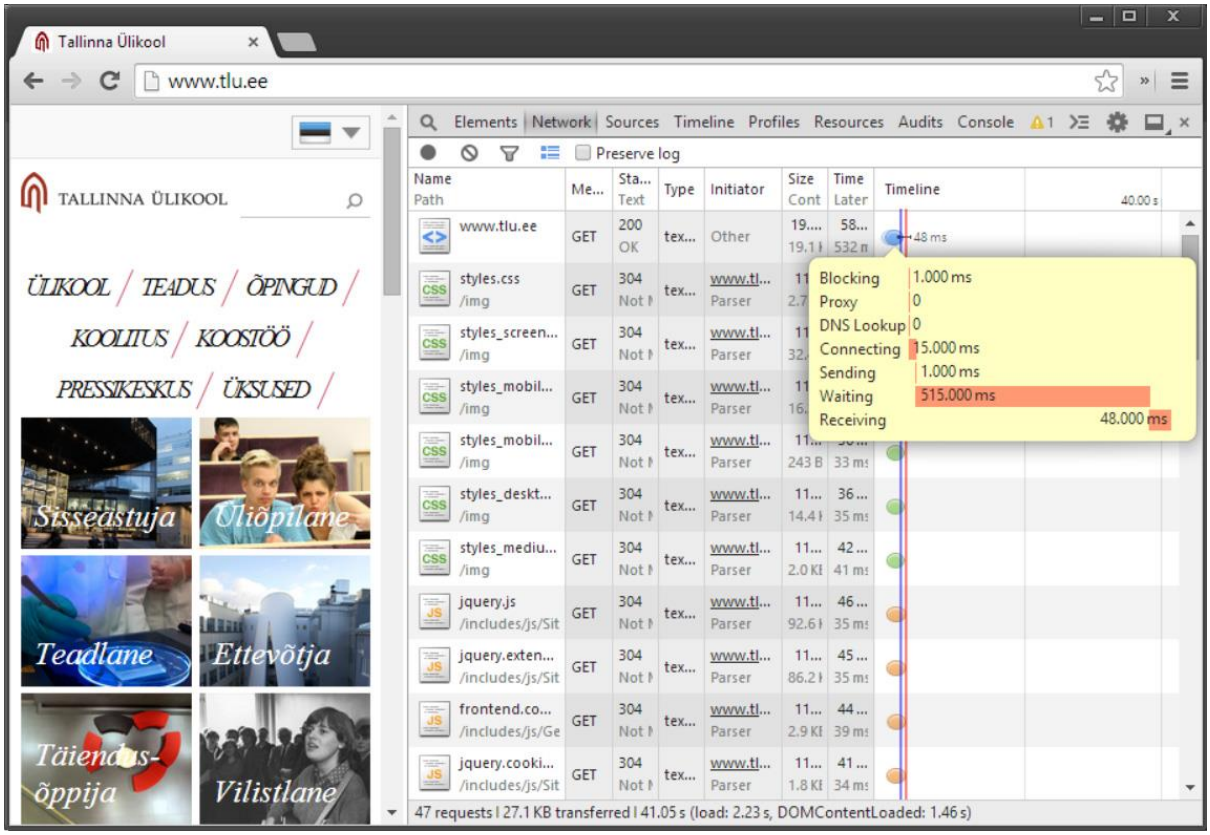
4.1 Brauseri tööriistade kasutamine

Tööriistade kasutamine käib erinevatel brauseritel erinevalt, *Chrome Developer Tools*'i ja *Firefox Developer Tools'i* saab avada vajutades klahvi *F12*, klahvikombinatsiooni *Control/Command + Shift + i* või vajutades lehel parema klahviga ning valides *Inspect Element*. Internet Exploreri tööriistade kasutamiseks saab kasutada kas klahvi F12 või siis sarnaselt Chome'ile ja Firefox'ile vajutades lehel parema klahviga ning valides *Inspect Element*. Safaril on lähtesättena keelatud ja tuleb eelnevalt sätetest lubada, seejärel saab kasutada klahvikombinatsiooni Control/Command + Alt + i (Chrome DevTools, 2014) (Mozilla Developer Network, n.d.) (Microsoft, n.d.) (Apple, 2013).

Kõigil nimetatud brauseritel on võimalik inspekteerida DOM-i (*Document Object Model'i*) elemente, sättida nendel CSS-i ning kasutada JavaScript'i konsooli, millest peaks piisama üksiku brauseri jaoks lehe silumiseks. Keerukamad operatsioonid nagu lehe jõudluse mõõtmine on saadaval ka kõikidel nimetatud brauseritel, kuid seda ei peaks olema tarvis teha iga brauseri jaoks. Sel põhjusel valis töö autor välja, et kirjutab kõige täpsemalt ainult ühest brauserist, Chrome'ist, sest selle tööriistad on kõige rohkemate võimalustega.

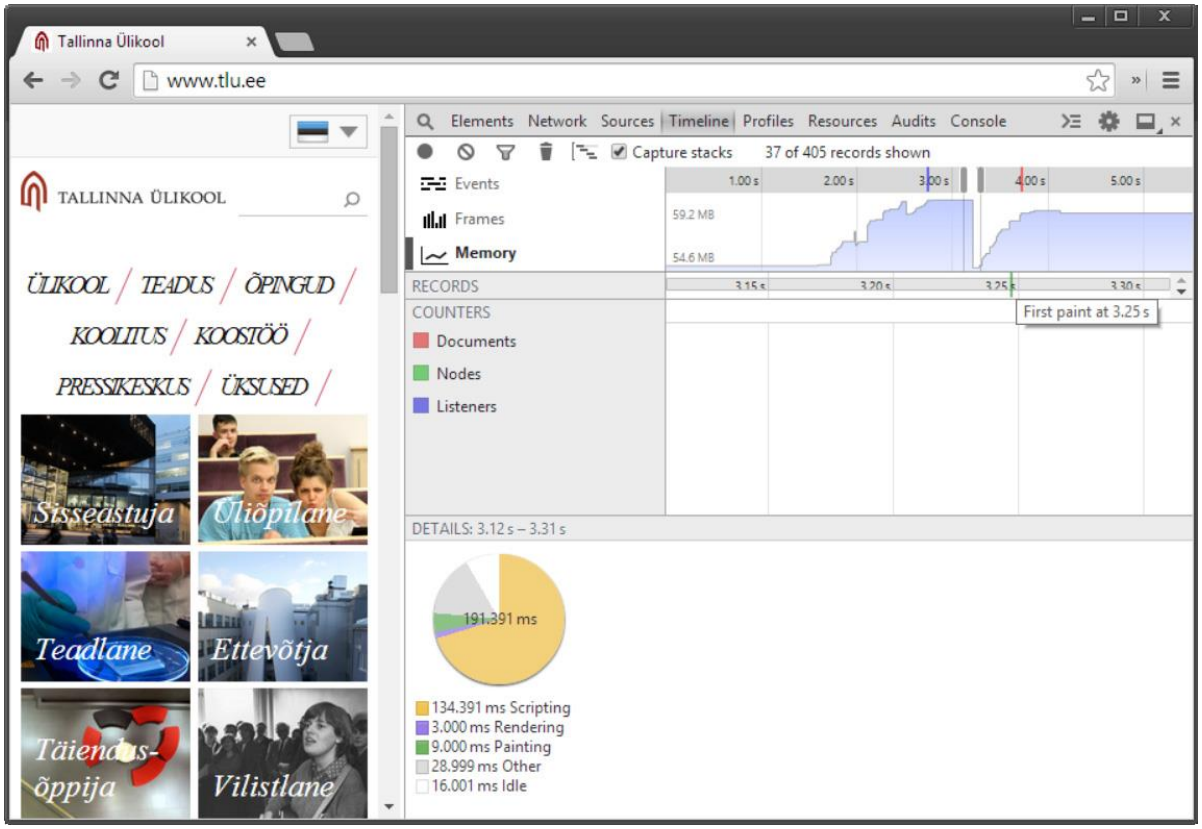
4.2 Jõudluse meetrika

Chrome pakub mitu erinevat võimalust jõudluse mõõtmiseks. Üks moodus on mõõta võrku, seda kui palju läheb erinevate ressursside allalaadimiseks, sealhulgas kui palju aega läks ühenduse loomiseks, andmete saatmiseks, ootamiseks ning andmete vastuvõtmiseks. Lisaks näitab ära, mis mingi ühenduse käivitas, näiteks kui mingi ühenduse lõi JavaScript'i fail, siis sinna ilmubki JavaScript'i faili nimi ning sellele vajutades viiakse kasutaja sellele failile (Google Developers, 2014). (vt Joonis 11)



Joonis 11. Lehe laadimise kiiruste vaade

Teine võimalus on kasutada ajajoont, millel näitab ära kõik mis toimub lehekülje kuvamisel, sealhulgas kui palju aega kuluks enne kui esimest korda asjad ekraanile joonistati, millal *garbage collection* tööle hakkas ja kui palju mälu mingil ajahetkel lehekülg kasutab. See võimaldab tuvastada jõudluse kitsaskohti ning annab arendajale väga hea ülevaate ja võimaluse võrrelda erinevaid muudatusi objektiivselt (Google Developers, 2014). (vt Joonis 12)



Joonis 12. Ajajoon Chrome Developer Tools'is

4.3 Sündmuste kuularid

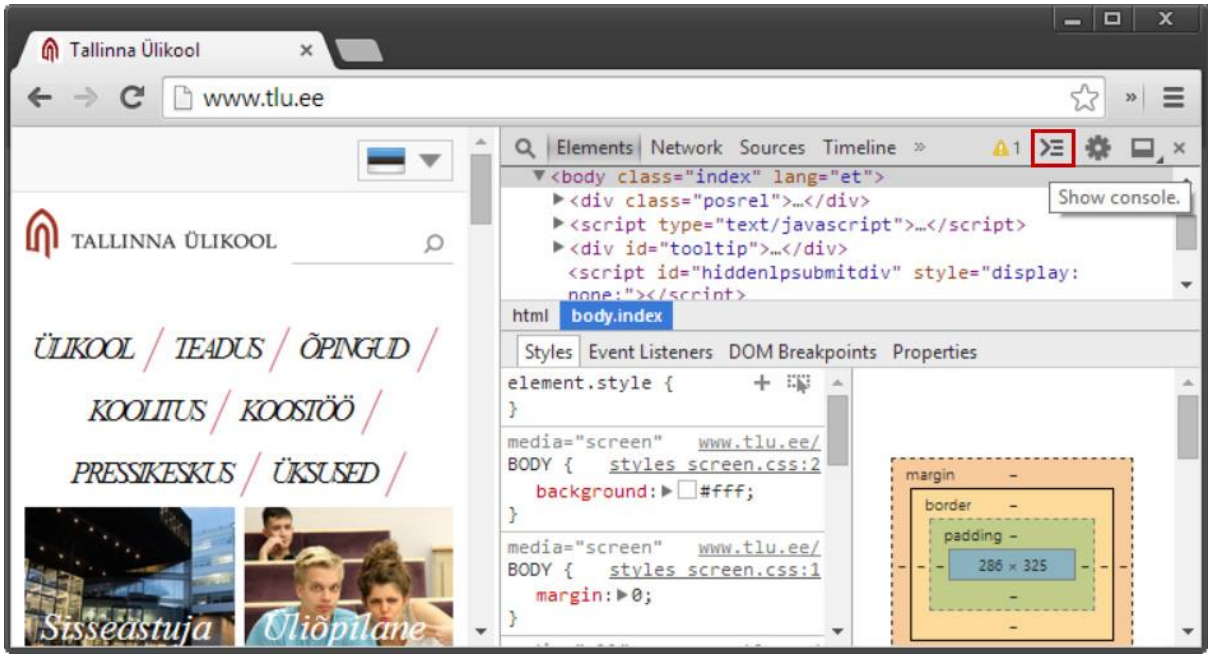
Elements sakil olles võib vaadata lisaks elementide stiilidele ka seda, mis sündmuste kuularid on igal DOM-i elemendil. See tuleb kasuks siis, kui tekib kahtlus, et sündmuste kuularid ei ole õigetele elementidele läinud, või kuulareid võib kogemata mitu tükki tekkinud (Google Developers, 2013).

4.4 Emuleerimine

Kahtlemata on kasvav trend veebiarenduses lehtede töötamises veendumine mobiilsetel seadmetel. Mobiilseid seadmeid on aga erinevate resolutsioonidega ning kõiki neid pole võimalik omada. Abiks tuleb Chrome'i emuleerimise funktsioon, mis lubab oma töölauabrauserist emuleerida erinevaid mobiilsed seadmeid, valida saab sisseehitatud mobiilseadmete spetsifikatsioonide hulgast või sättida ise seadme resolutsiooni ja andureid (Google Developers, 2014).

Emuleerimise alustamiseks tuleb kõigepealt avada *Chrome Developer Tools* (selle saab avada vajutades klahvi *F12*). Seejärel saab avada sahtli vajutades kas vastavale nupule (vt Joonis

13) või siis vajutades klahvile *Esc*. Edasi saab vajutada lihtsalt sakile nimega *Emulation* ning ongi olemas kõiksugu valikud. Vajutades nuppu *emulate* muudetakse leht sätete järgi valitud suuruseks. Sensorite alamlehelt saab sättida, kas emuleerida puuteekraani, GPS'i ning kiirendusandurit (Google Developers, 2014).



Joonis 13. Show console nupp

4.5 Mobiilne silumine

Kuigi on võimalik emuleerida mobiilseadmeid Chrome'is, võib siiski vahel tekkida vajadust testida lehti ehtsal seadmel. Mobiilsetel seatmetel viga silumine on aga väga raske, käesoleva töö autor on eelnevalt kasutanud näiteks kahtlustades, et mingi muutuja saab õige väärtuse kuvanud selle muutuja väärtuse lehekülje mingis osas, kuid hiljuti on tekkinud uued võimalused.

Alates versioonist 32 on töölaua Chrome'i võimalik ühendada Androidil jooksva Chrome'iga ja kasutada kõiki Chrome Developer Tools'i tööriistu otse arvutist. Lisaks võimaldab see edastada veebilehitseja vaadet mobiilselt seadmelt arvutisse. Ja arvutist navigeerida veebilehel (sealhulgas emuleerida puuteekraani) (Chrome DevTools, 2014).

Mobiilse silumise jaoks peab olema Androidi silumine (Android debugging) lubatud. Kui see on lubatud, saab avada Androidist Chrome brauseri, ühendada selle USB läbi arvutiga ning siis minna arvutilt aadressile *chrome://inspect* lehele (Chrome DevTools, 2014)

5. Autori poolt valitud töövoog

Töö teema valiku põhjuseks oli leida meetodid, millega autor saaks oma töövoogu parendada. Siin peatükis paneb töö autor kirja eelnevalt enda poolt kasutatud töövoo ning ühtlasi töövood, mida autor nüüdsest kasutab.

5.1 Autori eelnev töövoog

Eelnevalt on töö autori töövoog olnud väga ebaefektiivne. Igal uuel projektil olid kõik teegid ja raamistikud käsitsi alla laetud, koodi kirjutamine toimus Notepad++'iga ja mitte kohalikus arvutis, vaid WinSCP-ga serveris, mille tõttu pidi tihti ootama, et ühendus serveriga taastuks ja saaks kontrollida, kas muudatused failides olid tehtud õigesti.

Mõningal määral aitas HTML-i kirjutamist Emmet'i pistikprogrammi kasutamine. Varundamine toimus kausta kopeerimisega vabalt valitud hetkedel, tihti oli varundamiste vahel mitu päeva.

5.2 Autori uus töövoog

Projekte on erinevaid ja seetõttu pole ainult ühte õiget töövoogu. Lihtsamad projektid, kus pole tähtis, kas kood on minimeeritud või mitte, saab lahendada niisama tehes valmis endale uue projekti malli, mida saab lihtsalt õigete käskudega kopeerida. Suuremad projektid, kus näiteks on vaja kasutada erinevaid teeke ja raamistikke, on lihtsam lahendada Yeoman'i abiga ning kasutada Grunt'i erinevate ülesannete täitmiseks.

Koodiredaktori valik on töö autoril Sublime Text 3. Selle valiku kasuks tuli väga hea süntaksi esiletõstmine, sulgude paaritamine ja Command Palette'i kasutamise kiirus. Lisaks aitab see, et tegu on mitmeplatvormilise tööriistaga. Lisaks kasutab töö autor Vim'i, kui on vaja läbi viia mingi väike muudatus serveris asuvas failis.

Failidega töötamisel järgib töö autor põhimõtet, et tööta kohalikus arvutis ja lükka muudatused serverisse. Seda on teha võimalik mitut moodi, kuid Git tundub neist kõige parem, sest see lahendab ühtlasi lihtsalt ära ka versioonikontrolli probleemi.

Eelprotsessorite valik on üsnagi tasavägine, töö autor on eelnevalt ainult kasutanud Sass'i, kuid nüüd paistab LESS-i kasutamine lihtsam, arvestades seda, et ei pea seda kompileerima,

vaid saab kasutada JavaScript'i faili. JavaScript'i eelprotsessoritest hakkab töö autor suuremate projektide puhul kasutama TypeScript'i. Ehkki CoffeeScript on lühem ja populaarsem, töö autori isiklik eelistus on siiski kasutada loogelisi sulge blokkide määratlemisel, mitte treppimist.

Kokkuvõte

Käesoleva töö eesmärgiks oli tuua välja erinevad meetodid eesrakenduste loomisel kasutatava töövoo parendamiseks. Lisaks sellele oli eesmärgiks neid võrrelda ning töö autori poolt kasutusele võtta ja valitud töövoog välja tuua.

Töö on mõeldud inimestele, kes loovad tihti eesrakendusi, kuid leiavad, et liiga palju aega kulub korduvatele töödele nagu teekide allalaadimine, failide varundamine jne. Lisaks võivad sellest kasu saada inimesed, kes on juba teinud korduvate ülesannete täitmise võimalikult kiireks, kuid ei kasuta eelprotsessoreid. Kuna välja toodud on eelprotsessorite omadused ning osa nende süntaksitest, peaks olema üsnagi lihtne valida välja eelprotsessor, mis neile sobib.

Tööd võib edasi arendada mitmes eri suunas. Üks teema, mida töö autor ei puudutanud, on testimine ja selle automatiseerimine. JavaScript'i testimine on väga mitmekülgne, võib testida nii funktsioone kui ka kasutajaliidest. Lisaks tuleb veenduda, et rakendus töötaks mitmes brauseris ning erinevatel platvormidel. On olemas erinevaid testijooksutajat, üks prominentne on Karma ning on olemas ka teenus nimega BrowserStack, mis võimaldab jooksutada teste virtuaalmasinatel läbi pilveteenuse.

Lisaks märkas töö autor pärast US klaviatuuripaigutisele üleminemist, et koodi kirjutamine on kindlatel põhjustel kiirendatud, näiteks kandiliste ja loogeliste sulgude paigutus on parem, kui Eesti paigutusega klaviatuuril. Võimalik oleks uurida alternatiivsete klaviatuuripaigutuste kasutamise mõju programmeerimise kiirusele.

Summary

The objectives for this Bachelor thesis are to give an overview of methods to improve the workflow for front-end development and also compare them where necessary. Also the author compared a few more popular code editors and the ways they can help improve the workflow.

The first chapter deals with 5 code editors: Brackets, Coda 2, Notepad++, Sublime Text 3 and Vim. All of the aforementioned code editors have their pros and cons, but the author has chosen Sublime Text 3 because it is available on both Windows, Linux and Mac OS X platforms, it has a lot of plugins to further enhance the programming experience and the syntax highlighting in it is spectacular.

The Second chapter is dedicated to different preprocessors for CSS and JavaScript. Of the CSS preprocessors the three most popular were chosen: Sass, LESS and Stylus. Although the three are very similar and the author has used Sass in the past, He will use LESS from now on, because it can be used to compile the style sheets on the client side instead of having to do it every time a change is made.

The third chapter has a few ways to automate repetitive tasks like starting new projects and backing up old ones. In the fourth chapter the author describes the different browser tools that can help debug a web application, including how to debug a website on a mobile platform.

The fifth chapter has the workflows the author will now be using. The author describes two separate workflows, one for smaller projects and one for larger ones. In both cases the author will be using Git, because in His experience a lot of progress has been made undone because there haven’t been proper backups.

Kasutatud kirjandus

Ali, A. (14. August 2013. a.). EVALUATION OF ALTERNATE PROGRAMMING LANGUAGES . Tampere.

Apple. (2013). *Get Oriented*. Allikas: Safari Web Inspector Guide: https://developer.apple.com/library/safari/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/GettingStarted/GettingStarted.html#//apple_ref/doc/uid/TP40007874-CH2-SW1

Arlt, P. (15. Juuni 2013. a.). *CSS preprocessors - Sass, Less and Styluss*. Allikas: GitHub: http://patrickarlt.github.io/sass-less-stylus-slides/

Bloch, O. (01. Oktoober 2012. a.). *Sublime Text, Vi, Emacs: TypeScript enabled!* Allikas: Microsoft Open technologies: http://msopentech.com/blog/2012/10/01/sublime-text-vi-emacs-typescript-enabled/

Brzek, P. (23. Aprill 2014. a.). *LESS Hat 3.0*. Kasutamise kuupäev: 30. Aprill 2014. a., allikas GitHub: github.com/madebysource/lesshat/blob/master/README.md

caniuse.com. (23. Aprill 2014. a.). *calc() as CSS unit value*. Kasutamise kuupäev: 24. Aprill 2014. a., allikas Can I use...: http://caniuse.com/#feat=calc

Chikuyonok, S. (21. November 2009. a.). *Zen Coding: A Speedy Way To Write HTML/CSS Code*. Kasutamise kuupäev: 08. Aprill 2014. a., allikas Smashing Magazine: http://coding.smashingmagazine.com/2009/11/21/zen-coding-a-new-way-to-write-html-code/

Chrome DevTools. (02. Aprill 2014. a.). *Chrome DevTools*. Allikas: Google Developers: https://developers.google.com/chrome-developer-tools/

Chrome DevTools. (17. Veebruar 2014. a.). *Remote Debugging Chrome on Android*. Kasutamise kuupäev: 24. Aprill 2014. a., allikas Chrome DevTools: https://developers.google.com/chrome-developer-tools/docs/remote-debugging

Coyier, C. (11. Juuni 2012. a.). *Poll Results: Popularity of CSS Preprocessors*. Kasutamise kuupäev: 06. Aprill 2014. a., allikas CSS-Tricks: http://css-tricks.com/poll-results-popularity-of-css-preprocessors/

Ecma International. (17. Jaanuar 2014. a.). *TC52 - Dart.* Allikas: Ecma International: http://www.ecma-international.org/memento/TC52.htm

Emmet. (23. Detsember 2013. a.). *Download*. Kasutamise kuupäev: 08. Aprill 2014. a., allikas Emmet.io: http://emmet.io/download/

Eppstein, C. M. (27. November 2013. a.). *Compass Documentation*. Allikas: Compass: http://compass-style.org/

Fenski, B. (kuupäev puudub). *SSHFS - filesystem client based on ssh*. Kasutamise kuupäev: 04. 04 2014. a., allikas die.net: http://linux.die.net/man/1/sshfs

Fitzpatrick, J. (05. Detsember 2010. a.). *Five Best Text Editors*. Kasutamise kuupäev: 08. Aprill 2014. a., allikas lifehacker: http://lifehacker.com/5706475/five-best-text-editors

Florian Loitsch. (22. Oktoober 2012. a.). Allikas: Dart: Structured web apps: www.dartlang.org/slides/2012/10/jsconfeu/javascript-as-compilation-target-florian-loitsch.pdf

Git. (2014). *About*. Allikas: Git: http://git-scm.com/about/small-and-fast

GitHub. (01. Mai 2014. a.). *Search - coffeescript*. Kasutamise kuupäev: 01. Mai 2014. a., allikas GitHub: github.com/search?l=CoffeeScript&q=coffeescript&ref=searchresults&type=Repositories

GitHub. (01. Mai 2014. a.). *Search - dart*. Allikas: GitHub: github.com/search?l=Dart&q=dart&ref=searchresults&type=Repositories

GitHub. (01. Mai 2014. a.). *Search - typescript*. Kasutamise kuupäev: 01. Mai 2014. a., allikas GitHub: github.com/search?l=TypeScript&q=typescript&ref=cmdform&type=Repositories

gnu.org. (06. Aprill 2014. a.). *Bash Startup Files*. Allikas: gnu.org: http://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

Google Developers. (7. Mai 2013. a.). *Editing styles & DOM*. Allikas: Chrome DevTools: developers.google.com/chrome-developer-tools/docs/elements

Google Developers. (12. Jaanuar 2014. a.). *Evaluating network performance*. Allikas: Chrome DevTools: developers.google.com/chrome-developer-tools/docs/network

Google Developers. (10. Veebruar 2014. a.). *Mobile emulation*. Allikas: Chrome DevTools: developers.google.com/chrome-developer-tools/docs/mobile-emulation

Google Developers. (12. Veebruar 2014. a.). *Performance profiling with the Timeline*. Kasutamise kuupäev: 01. Mai 2014. a., allikas Google Developers: developers.google.com/chrome-developer-tools/docs/timeline

Komarov, R. (23. Aprill 2014. a.). *Stylus history.md*. Kasutamise kuupäev: 30. Aprill 2014. a., allikas GitHub: github.com/LearnBoost/stylus/blob/master/History.md

Kovalyov, A. (23. Märts 2014. a.). *Why I forked JSLint to JSHint*. Allikas: Anton Kovalyov: http://anton.kovalyov.net/p/why-jshint/

learnboost. (01. Aprill 2014. a.). *stylus*. Allikas: GitHub: https://github.com/LearnBoost/stylus/blob/master/lib/functions/index.styl#L28

learnboost. (23. Aprill 2014. a.). *Stylus --- expressive, robust, feature-rich CSS preprocessor*. Allikas: Stylus: http://learnboost.github.io/stylus/

Maharry, D. (2013). *TypeScript Revealed.* New York: Apress.

Microsoft. (kuupäev puudub). *Discovering Windows Internet Explorer Developer Tools*. Allikas: Microsoft Developer Metwork: http://msdn.microsoft.com/en-us/library/dd565628(v=vs.85).aspx

mladenovic.ml@gmail.com. (kuupäev puudub). *win-sshfs*. Allikas: Google code: https://code.google.com/p/win-sshfs/

Mozilla Developer Network. (kuupäev puudub). *Firefox Developer Tools*. Allikas: Mozilla Developer Network: https://developer.mozilla.org/en/docs/Tools

Node.js Modules. (30. Aprill 2014. a.). *Stylus modules*. Allikas: Node.js Modules: https://nodejsmodules.org/tags/stylus

Painter, B. (19. Veebruar 2013. a.). *Introduction to Sass: Part 1 – Installation*. Kasutamise kuupäev: 06. Aprill 2014. a., allikas Adobe Developer Connection: http://www.adobe.com/devnet/html5/articles/introduction-to-sass-part-1-installation.html

Robbins, A., Hannah, E., & Lamb, L. (2008). *Learning the vi and Vim Editors.* Sebastopol: O'Reilly.

rupa. (2013). *z*. Allikas: GitHub: https://github.com/rupa/z

Sass. (26. Märts 2014. a.). *Sass Basics*. Kasutamise kuupäev: 06. Aprill 2014. a., allikas sass-lang.com: http://www.sass-lang.com/guide

sass-lang. (26. Märts 2014. a.). *Functions*. Allikas: sass-lang.com: http://sass-lang.com/documentation/Sass/Script/Functions.html

Sitnik, A. (8. Aprill 2014. a.). *autoprefixer*. Kasutamise kuupäev: 08. Aprill 2014. a., allikas GitHub: https://github.com/ai/autoprefixer

sourceforge.net. (2009). *SourceForge Community Blog | 2009 CCA: Winners*. Kasutamise kuupäev: 8. Aprill 2014. a., allikas SourceForge: http://sourceforge.net/blog/cca09/winners/

SS64.com. (29. September 2013. a.). *function Man Page*. Kasutamise kuupäev: 20. Aprill 2014. a., allikas SS64.com: http://ss64.com/bash/function.html

SS64.com. (17. Jaanuar 2014. a.). *alias Man Page*. Kasutamise kuupäev: 20. Aprill 2014. a., allikas SS64.com: http://ss64.com/bash/alias.html

Stylus. (23. Aprill 2014. a.). *Iteration -- Stylus*. Kasutamise kuupäev: 30. Aprill 2014. a., allikas Stylus: http://learnboost.github.io/stylus/docs/iteration.html

Stylus. (07. Aprill 2014. a.). *Mixins*. Kasutamise kuupäev: 19. Aprill 2014. a., allikas Stylus: http://learnboost.github.io/stylus/docs/mixins.html

Sublime Text. (08. Juuli 2013. a.). *Sublime Text: The text editor you'll fall in love with*. Kasutamise kuupäev: 28. Aprill 2014. a., allikas Sublime Text: http://www.sublimetext.com/

The Less Core Team. (05. Märts 2014. a.). *Functions*. Allikas: Less.js: http://lesscss.org/functions/#math-functions

The Less Core Team. (05. Märts 2014. a.). *Getting Started*. Allikas: Less.js: lesscss.org

The Less Core Team. (05. Märts 2014. a.). *Language features*. Kasutamise kuupäev: 19. Aprill 2014. a., allikas Less.js: http://lesscss.org/features/#mixins-feature

The Sass Way. (kuupäev puudub). *Sass control directives: @if, @for, @each and @while*. Kasutamise kuupäev: 30. Aprill 2014. a., allikas The Sass Way: http://thesassway.com/intermediate/if-for-each-while

Way, J. (02. Jaanuar 2012. a.). *Introducing Nettuts+ Fetch*. Allikas: Tuts+: http://code.tutsplus.com/articles/introducing-nettuts-fetch--net-23490

wbond. (2014). *About*. Allikas: Package Control: https://sublime.wbond.net/about

wbond. (2014). *Stats*. Allikas: Package Control: https://sublime.wbond.net/stats

WikiBooks. (19. Juuni 2013. a.). *Learning the vi Editor/VIM/Modes*. Kasutamise kuupäev: 28. Aprill 2014. a., allikas WikiBooks: http://en.wikibooks.org/wiki/Learning_the_vi_Editor/Vim/Modes

Yeoman. (2014). *Yeoman - Modern workflows for modern webapps*. Allikas: Yeoman: http://yeoman.io/

LISA 1: Eelprotsessorite aritmeetikafunktsioonid

	http://sass-lang.com/documentation/Sass/Script/Functions.html	http://lesscss.org/functions/#math-functions	https://github.com/LearnBoost/stylus/blob/master/lib/functions/index.styl#L28
Funktsioon:	**Sass'i süntaks:**	**LESS-i süntaks:**	**Styluse süntaks**
'+ - * /'	'+ - * /'	'+ - * /'	'+ - * /'
lagi	ceil	ceil	ceil
põrand	floor	floor	floor
reaalarv protsendiks	percentage	percentage	
ümarda	round	round	round
ruutjuur		sqrt	
absoluutväärtus	abs	abs	abs
siinus		sin	sin
arkussiinus		asin	
koosiinus		cos	cos
arkuskoosiinus		acos	
tangens		tan	
arkustangens		atan	
pii		pi	PI
astendamine		pow	
mod		mod	
miinimum	min	min	min

maksimum	max	max	max
summa			sum
Keskmine			avg
protsent kümnendarvuks			percent-to-decimal
radiaanid kraadideks			radians-to-degrees
kraadid radiaanideks			degrees-to-radians
juhuarv	random	`Math.random()`	

LISA 2: CSS-i eelprotsessorite värvifunktsioonid

	Sass	LESS	Stylus	Seletus
rgb	rgb(punane, roheline, sinine)	rgb(punane, roheline, sinine)	rgb(punane, roheline, sinine)	Tagastab punasest, rohelisest ning sinisest väärtusest kombineeritud värvi
rgba	rgba(punane, roheline, sinine, alfa)	rgba(punane, roheline, sinine, alfa)	rgba(punane, roheline, sinine, alfa)	Tagastab punasest, rohelisest, sinisest ning alfas väärtusest kombineeritud värvi
hsl	hsl(toon, küllastus, heledus)	hsl(toon, küllastus, heledus)		Tagastab toonist, küllastusest ning heledusest kombineeritud värvi
hsla	hsla(toon, küllastus, heledus, alfa)	hsla(toon, küllastus, heledus, alfa)	hsla(toon, küllastus, heledus, alfa)	Tagastab toonist, küllastusest, heledusest ning alfast kombineeritud värvi
hsv		hsv(toon, küllastus, väärtus)		Tagastab toonist, küllastusest ja väärtusest kombineeritud värvi
hsva		hsva(toon, küllastus, väärtus, alfa)		Tagastab toonist, küllastusest, väärtusest ja alfast kombineeritud värvi
Punane komponent	red(värv)	red(värv)	red(värv)	Tagastab punase värvikomponendi algvärvist
Roheline komponent	green(värv)	green(värv)	green(värv)	Tagastab rohelise värvikomponendi algvärvist
Sinine komponent	blue(värv)	blue(värv)	blue(värv)	Tagastab sinise värvikomponendi algvärvist
Värvide segamine	mix(värv1, värv2 [, tase])	mix(värv1, värv2 [, tase])	(värv1 / 2) + (värv2 / 2)	Segab kaks värvi kokku omavahel
Toonikomponent	hue(värv)	hue(värv)	hue(värv)	Tagastab HSL värVimudeli järgi Hue komponendi
Küllastuskomponent	saturation(värv)	saturation(värv)	saturation(värv)	Tagastab HSL värVimudeli järgi Saturation komponendi
Heleduskomponent	lightness(värv)	lightness(värv)	lightness(värv)	Tagastab HSL värVimudeli järgi Lightness komponendi
HSV toonikomponent		hsvhue(värv)		Tagastab HSV värVimudeli järgi Hue komponendi
HSV küllastuskomponent		hsvsaturation(värv)		Tagastab HSV värVimudeli järgi Saturation komponendi

HSV väärtuskomponent		hsvvalue(värv)		Tagastab HSV värVimudeli järgi Value komponendi
Tooni muutmine	adjust-hue(värv, kraadid)	spin(värv, kraadid)	värv + kraadid	Muudab algvärvi tooni etteantud kraadi võrra
Lighten	lighten(värv, protsent)	lighten(värv, protsent)	lighten(värv, protsent)	Muudab algvärvi HSL-i värVimudeli Lightness komponenti etteantud hulga võrra heledamaks
Darken	darken(värv, protsent)	darken(värv, protsent)	darken(värv, protsent)	Muudab algvärvi HSL-i värVimudeli Lightness komponenti etteantud hulga võrra tumedamaks
Küllasta	saturate(värv, protsent)	saturate(värv, protsent)	saturate(värv, protsent)	Muudab algvärvi HSL-i värVimudeli Saturation komponenti etteantud hulga võrra suuremaks
Vähenda küllastust	desaturate(värv, hulk)	desaturate(värv, hulk)	desaturate(värv, hulk)	Muudab algvärvi HSL-i värVimudeli Saturation komponenti etteantud hulga võrra väiksemaks
Muuda halliks	greyscale(värv)	greyscale(värv)	greyscale(värv)	Muudab HSL-i värVimudeli Saturation komponendi nulliks
Vastandvärv	complement(värv)		complement(värv)	Tagastab etteantud värvi vastandvärvi
Tagurpidivärv	invert(värv)		invert(värv)	Lahutab iga RGB värvikomponendi 255-st
Läbipaistvus	alpha(värv) / opacity(värv)	alpha(värv)	alpha(värv)	Tagastab RGBA värVimudeli Alpha komponendi
Muuda läbipaistvust	rgba(värv, läbipaistvus)	fade(värv, protsent)		Tagastab RGBA värvi etteantud läbipaistvusega
Muuda läbipaistmatuks	opacify(värv, hulk) / fade-in(värv, hulk)	fadein(värv, protsent)		Muudab värv etteantud hulga võrra vähem läbipaistvamaks
Muuda läbipaistvaks	transparentize(värv, hulk) / fade-out(värv, hulk)	fadeout(värv, protsent)		Muudab värv etteantud hulga võrra rohkem läbipaistvamaks
Reguleeri värvi	adjust-color(värv [, $red: väärtus] [, $green: väärtus] [, $blue: väärtus] [, $hue: väärtus] [$saturation:			Suurendab või vähendab ühte või enamat värvi komponenti

	väärtus] [$lightness: väärtus] [$alpha: väärtus])			
Skaleeri värvi	scale-color(värv [, $red: väärtus] [, $green: väärtus] [, $blue: väärtus] [$saturation: väärtus] [$lightness: väärtus] [$alpha: väärtus])			Skaleerib ühte või enamat värvi omadust
Muuda värvi	adjust-color(värv [, $red: väärtus] [, $green: väärtus] [, $blue: väärtus] [, $hue: väärtus] [$saturation: väärtus] [$lightness: väärtus] [$alpha: väärtus])			Muuda ühte või enamat värvi komponenti
aRGB	ie-hex-str(värv)	argb(värv)		Tagastab Internet Explorer-i filtriomadusele sobiiva värvistringi
Tajutav heledus		luma(värv)		Arvutab värvi luma (tajutava heleduse)
Kontrast		contrast(alusvärv, värv1, värv2 [, tase]		Tagastab, kumb kahest värvist on suurema kontrastiga võrreldes alusvärviga
Korruta		multiply(värv1, värv2)		Korrutab kahe värvi väärtused omavahel ja jagab 255-ga
Sõel		screen(värv1, värv2)		Teeb vastupidise korrutamisele
Aplikatsioon		overlay(värv1, värv2)		Tingimustega teeb heledamad värvid heledamaks ning tumedamad tumedamaks, baseerub esimesel värvil
Pehme valgus		softlight(värv1, värv2)		Sarnane aplikatsioonile, kuid väldib täielikku musta ja täielikku valget
Tugev valgus		hardlight(värv1, värv2)		Sarnane aplikatsioonile, kuid värvide rollid on vahetuund
Erinevus		difference(värv1, värv2)		Teine värv lahutatakse komponentidepõhisel t esimesest, negatiivsed väärtused

				pööratakse ümber
Väljajätmine		exclude(värv1, värv2)		Sarnane erinevusele, kuid väiksema kontrastiga
Keskmine		average(värv1, värv2)		Arvutab kahe värvi keskmise
Eitus		negation(värv1, värv2)		Vastupidine erinevusele
Tumeduskontroll			dark(värv)	Kontrollib, kas värv on tume
Heleduskontroll			light(värv)	Kontrollib, kas värv on hele
Toon			tint(värv, hulk)	Segab värvile etteantud hulga valget värvi juurde
Varjund			shade(värv, hulk)	Segab värvile etteantud hulga musta värvi juurde
Kokku	27	38	23	

LISA 3: Tsüklid CSS eelprotsessorites

Sass'i näide:

```
$grey: #141414;
$purple: #800080;
@for $i from 1 through 10{
  .rnd#{$i}{
    background: darken($grey, $i * 0.5);
    &:hover{
      background: adjust-hue(desaturate($purple, 50%), i * 36);
    }
  }
}
```

LESS-i näide:

```
@grey: #141414;
@purple: #800080;
.make(10);
.make(@n, @i: 1) when (@i =< @n) {
  .rnd@{i} {
    background: darken(@grey, @i * 0.5);
    &:hover{
      background: spin(desaturate(@purple, 50%), @i * 36);
    }
  }
  .make(@n, (@i + 1));
}
```

Stylus näide:

```
grey = #141414
purple = #800080
for i in (1..10)
  .rnd{i}
    background: darken(grey, i * 0.5)
    &:hover
      background: desaturate(purple, 50%) + i * 36deg
```

Kompileeritav CSS:

```
.rnd1 {
  background: #131313;
}
.rnd1:hover {
  background: #60203a;
}

.rnd2 {
  background: #111111;
}
.rnd2:hover {
  background: #602d20;
}

.rnd3 {
```

```
  background: #101010;
}
.rnd3:hover {
  background: #605320;
}

.rnd4 {
  background: #0f0f0f;
}
.rnd4:hover {
  background: #466020;
}

.rnd5 {
  background: #0e0e0e;
}
.rnd5:hover {
  background: #206020;
}
.rnd6 {
  background: #0c0c0c;
}
.rnd6:hover {
  background: #206046;
}
.rnd7 {
  background: #0b0b0b;
}
.rnd7:hover {
  background: #205360;
}

.rnd8 {
  background: #0a0a0a;
}
.rnd8:hover {
  background: #202d60;
}

.rnd9 {
  background: #090909;
}
.rnd9:hover {
  background: #3a2060;
}

.rnd10 {
  background: #070707;
}
.rnd10:hover {
  background: #602060;
}
```