TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kelli Sepp 154902

SERVERIPOOLSE RENDERDAMISE VÕIMALUSED VUE.JS PÕHISES RAKENDUSES

Bakalaureusetöö

Juhendaja: Roger Kerse
Msc

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kelli Sepp

21.05.2018

Annotatsioon

Bakalaureusetöö eesmärgiks on uurida ning võrrelda serveripoolse renderdamise lahendusi Vue.js põhises rakenduses veebilehe laadimiskiiruse tõstmise eesmärgil.

Autor võtab vaatluse alla kolm erinevat serveripoolse renderdamise varianti Vue raamistikku kasutavas rakenduses ning seadistab need eraldi prototüüp rakendustes. Selle käigus annab autor iga variandi konfigureerimisest põhjaliku ülevaate ja hindab iga variandi lihtsust, kogukonna suurust, sobivust eksisteerivasse rakendusse ning jõudlust. Lisaks antakse ülevaade kliendi- ning serveripoolsest renderdamisest üldisemalt.

Töö tulemusena moodustus põhjalik arusaam ning hinnang serveripoolse renderdamise implementeerimise viisidest Vue.js põhises rakenduses.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 4 peatükki, 15 joonist, 0 tabelit.

Abstract
The Possibilities of Server Side Rendering in Vue.js Application

The aim of this thesis is to examine and compare server side rendering solutions for Vue.js application with the goal of improving page speed.

The author takes under scrutiny three different server side rendering options suitable for Vue based applications and sets up each one in a seperate application. While doing the set up, the author gives an overview of the configuring process and evaluates the simplicity, the size of the community, the suitability for excisting application and the performance of each solution. Additionally the author gives a general overview of client and server side rendering.

The result of this thesis is a deep understanding and evaluation of server side rendering options in Vue.js applications.

The thesis is in Estonian and contains 29 pages of text, 4 chapters, 15 figures, 0 tables.

Lühendite ja mõistete sõnastik

CSS	*Cascading Style Sheets*, tehnoloogia HTML elementide kuvamise kirjeldamiseks
DOM	*Document Object Model*, rakendusliides HTML'i ja XML'i kirjutamiseks, mis defineerib dokumendi struktuuri
HTML	*Hypertext Markup Language*, tehnoloogia veebilehe struktuuri kirjeldamiseks
Middleware	*Middleware*'d on Node.js'is kasutatavad funktsioonid, mis saavad mõjutada ning töödelda sissetulevate ning väljaminevate päringute objekte
MVC	*Model-View-Controller*, tarkvaraarenduses tuntud muster, mille kohaselt on rakenduse arendusel olemas mudel, mis tähistab informatsiooni, vaade ehk kasutajale nähtav osa ning kontroller, mis sündmustele vastab
NPM	*Node package manager*, paketihaldur
SEO	*Search Engine Optimization*, praktika, mille käigus otsingumootoris lehed indekseeritakse ja prioritiseeritakse.

Sisukord

Jooniste loetelu

Sissejuhatus

Tänapäeval on veebilehe laadimiskiirus oluline osa lehe külastaja kasutajakogemuse moodustumisel. Üha enam veebirakendusi toetub suures osas enda funktsionaalsuses Javascriptile, mis muudab veebilehe interaktiivseks. Suur Javascripti koodihulk võib aga veebilehe esmast laadimiskiirust aeglasemaks muuta, sest kõik failid tuleb brauseris alla laadida ning aeglase interneti ühenduse puhul võib see kriitiliselt kaua aega võtta. Nii tekib probleem, kus ettevõtted soovivad enda veebirakendusi teha võimalikult interaktiivseteks, kuid samas kaasneb veebilehe esmase laadimise aeglasemaks muutumine. Veebilehe kiiruse tõstmise üheks variandiks on serveripoolse renderdamise implementeerimine, mille tulemusena paraneb veebilehe esmane laadimiskiirus, kuid säilib ka interaktiivsus.

Bakalaureusetöö eesmärgiks on esmalt anda ülevaade erinevatest viisidest veebilehe renderdamiseks ning nende mõjust lehe laadimiskiirusele. Töö praktlises osas annab autor põhjaliku ülevaate kolmest serveripoolse renderdamise võimalusest Vue.js põhinevas rakenduses. Kuna suurtes ettevõtetes on uute tarkvaralahenduste implementeerimine pikk ning kulukas protsess, siis on oluline, et erinevate lahenduste vahel valides tehakse põhjalik uurimus selgitamaks välja iga lahenduse võimalused ja piirangud.

Töö käigus teeb autor tutvust erinevate Vue.js serveripoolset renderdamist võimaldavate moodulite ning raamistikega, annab neist ülevaate, tutvustab konfigureerimise protsessi ning hindab ja võrdleb neid erinevate omaduste põhjal.

1. Veebilehe renderdamine

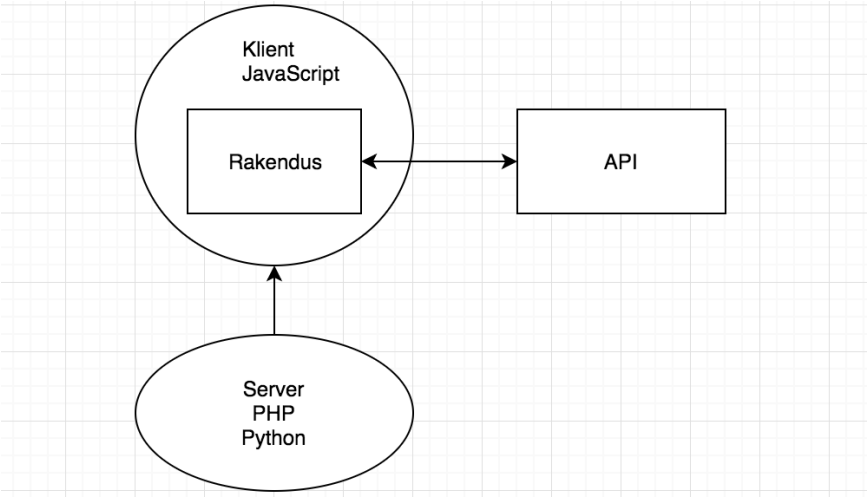
Veebilehe renderdamine on protsess, mille tulemusena kuvatakse veebibrauseris veebilehe sisu. Selle protsessi raames toimub nii informatsiooni kogumine, mallide laadimine, HTML koodi sõelumine ja muu sarnane. Brauseri *rendering engine* sisendiks on HTML kood, mis töödeldakse brauseriaknas veebileheks. Selleks sõelub *rendering engine* serverilt saadud HTML koodi elemente ning loob nendest DOM-puu, kus iga element saab alguse juurelemendist. Sarnaselt HTML koodi sõelumisele töödeldakse ka CSS atribuute sisaldavad failid ning lisatakse juba loodud DOM'ile. Seejärel saadakse render-puu, mis koosneb nii sõelutud HTML kui ka CSS elementidest ning kuvab neid hiearhilises järjekorras. Peale render-puu struktrueerimist jookseb *render engine* rekursiivselt läbi puu HTML elementide ning kaardistab need koordinaatide abil ekraanile. Kaardistatud elementide kuvamiseks suhtleb iga puu osa operatsioonisüsteemi kasutajaliidesega, mis sisaldab disaini ja stiile kõikidele kasutajaliidese elementidele. [1]

Veebilehe sisu käsitsemiseks peale renderdamise protsessi kasutatakse JavaScripti, mis võimaldab veebilehe muuta interaktiivseks. Javascript on dünaamiline programmeerimiskeel, mida kasutatakse veebibrauserites veebilehtede interaktviiseteks tegemiseks. [2]

1.1 Kliendipoolne

Kliendipoolne renderdamine on veebilehe sisu renderdamine brauseris kasutades JavaScripti. Brauserite võimekuse kasvades on veebikeskkond arenenud mitmekülgseks platvormiks, kus Javascript'i *runtime* ning HTML'i standardid teevad võimalikuks mitmekülgsete veebirakenduste arenduse ilma põliste platvormideta. Seetõttu on Javascripti kasutades terviklike veebirakenduste arendamine brauseris üha populaarsem. Taoline rakendus, kus loogika täitmine ei nõua terve lehe täielikku renderdamist, on tuntud kui *Single Page Application*. [3]

Single Page Application suudab reageerida kasutaja interaktsioonile kiiresti, sest uue lehe kuvamiseks või lehel info muutmiseks ei ole vaja kogu leht uuesti serverist laadida. Muudatuse sisseviimiseks vajalik rakenduse kood on juba brauseris alla laetud ning piisab vaid selle jooksutamisest, et uuendatud olekut kuvada. MVC tarkvaramustrit jälgiva rakenduse loogika nagu mudelid, vaated, kontrollerid, on kliendi poolel ning informatsioonile saadakse ligi kasutades rakendusliidest (vt Joonis 1). Serveri, mis võib olla kirjutatud mistahes programmeerimiskeelt kasutades, funktsiooniks on lehe esmasel pärimisel kuvada tühi HTML dokument, mis sisaldab linke Javascripti failidele. Need skriptid tuleb brauseril alla laadida ning jooksutada selleks, et leht täielikult renderdada. Kui leht on täielikult laetud on edasine lehtede vahel navigeerimine ja interaktsioon kiire ning kasutaja saab edais toimetada ilma lehte uuesti laadimata. *Single Page Application'*it täielikult implementeerides on osadel juhtudel võimalik rakendus hoida töös isegi võrgust väljas, sest brauser ei pea üle võrgu serveriga suhtlema ning rakendus suudab infot efektiivselt salvestada brauseri *local storage*'sse. [3]



Joonis 1. Kliendipoolset renderdamist kasutav rakendus.

Ühelehelise rakenduse arenduse töövoog on sujuv ja ennetab suure koodimahuga loogika jagamist serveri ning kliendi vahel, mis on sageli kirjutatud erinevates programeerimiskeeltes. Lisaks sobib kliendipoolne renderdamine rakendustele, mis soovivad tulevikus arendada mobiilirakendust, sest serveripoolne kood võib olla sama nii veebi – kui ka mobiilirakenduses. [4]

Kliendipoolne renderdamine on lahendus veebirakendustele, millele on oluline kiire reaktsioon kasutaja interaktiivstele toimingutele, sest brauser suudab päringut töödelda

ise, ilma serverisse päringut tegemata. Selline lähenemine on kogunud populaarsust, sest veebileht reageerib dünaamiliselt ning näiliselt hetkega kasutaja toimingutele. *Single Page Application*'i eesmärk on pakkuda sujuvat kasutajakogemust ilma lehte uuesti laadimata ning ootamiseta. [4]

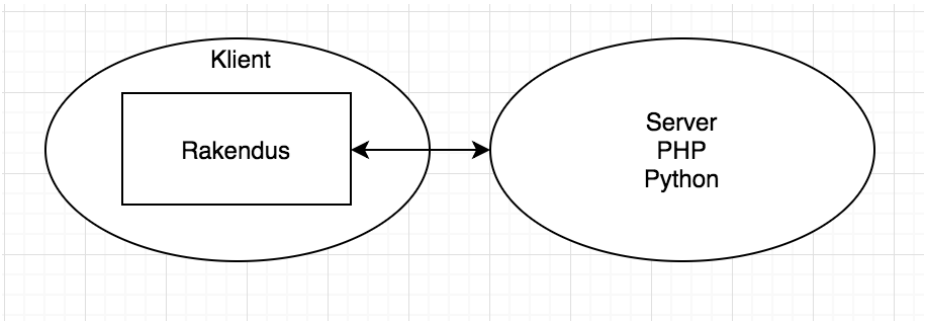
1.1.1 Jõudlus

Veebirakenduse laadimise kiirus on oluline faktor kasutaja kogemuse kujunemises. Mida vähem peab kasutaja ootama veebilehe laadimise järel, seda positiivsem arvamus kujuneb kasutajal rakenduse kasutamisest ning seda tõenäolisemalt kasutab ta rakendust uuesti. Amazoni näitel lehe laadimise kiiruse vähendamine 100 millisekundi võrra suurendab nende kasumit 1% võrra [3]. Kliendipoolse renderdamise puhul teeb brauser serverisse lehe esmasel laadimisel mitu päringut, mis võib sõltuvalt võrgu kiirusest ning serveri asukohast aega võtta. Veebirakenduse komplekssuse kasvades suureneb koodi hulk, mida brauser alla peab laadima ning aeglase internetiühenduse puhul võib esialgne laadimine võtta kriitiliselt kaua. Suureks veebilehe laadimiskiiruse mõjutaks kliendipoolsel renderdamisel on kasutaja internetikiirus ning riistvara uudsus. Statistika andmetel moodustab globaalsest veebiliiklusest 51,12% mobiilseadmete kasutajad [5]. Mobiilseadmete andmeside kiirus on väiksem ning ainult kliendipoolsele Javascriptil põhineva *Single Page Application*'i laadimine on kasutajate jaoks aeglane. Kuna esmalt saadakse serverist tühi HTML dokument siis võib kasutaja enne Javascripti alla laadimist näha tühja valget lehte või laadimist indikeerivat spinnerit. [6]

1.2 Serveripoolne

Serveripoolne renderdamine on olnud läbi ajaloo kõige standardsem viis veebilehe sisu kuvamiseks. Veebilehtede renderdamine toimus serveris, sest brauserite võimekus oli piiratud ning kuvatavad lehed olid pigem staatilised dokumendid, mitte dünaamilised lehed, mis genereerivad API't kasutades omale sisu. Nii tehti veebilehte külastades päring serverisse, mille tulemusena server saatis brauserile täielikult renderdatud HTML koodi, mis sisaldas endas kõike, mis oli veebilehe sisu kuvamiseks vajalik. Selline lähenemine leiab kasutust veel tänapäevalgi näiteks staatiliste lehtede puhul, kus pole Javascripti kasutuseks põhjust. Ilma Javascriptita terviklike veebirakenduste ehitamine on aga pigem harv juhus, sest erinevad interaktiivsed komponendid, nagu rippmenüü või *pop-up'*id, kasutavad Javascripti. Serveripoolse renderdamise kasutamisel tehakse veebirakenduses

iga uue lehe külastamiseks serverisse uus päring, et saada vastava lehe sisu olenemata sellest, milline info muutus ning kui palju erineb uus leht äsja külastatud lehest (vt Joonis 2). [7]



Joonis 2. Serveripoolset renderdamist kasutav rakendus.

Serveripoolse renderdamise puhul on lehe sisu serverist tulles täielik ning brauseris ei ole vaja infot asünkroonselt koguda. Lehe sisu kohesel olemasolul on mitmeid positiivseid omadusi. Näiteks on sisu olemas kohe ka otsingumootori *crawler*'i jaoks, mis alustab tööd koheselt, kui serverist on tulnud vastus. Otsingumootori *crawler* on programm või automatiseeritud skript, mis brausib kogu veebi metoodilisel viisil selleks, et pakkuda ajakohast infot spetsiifilise otsingumootori jaoks [8]. *Crawler*'i ülesandeks on veebilehe indekseerimine ning prioritiseerimine vastavalt selle sisule. Serveri poolt renderdatud veebilehe sisu on *crawl'*imise hetkeks valmis ning kui sisu on relevantne otsingu märksõnadega, siis prioritiseeritakse antud leht, mille tulemusena asub see otsingutulemustes eespool [6].

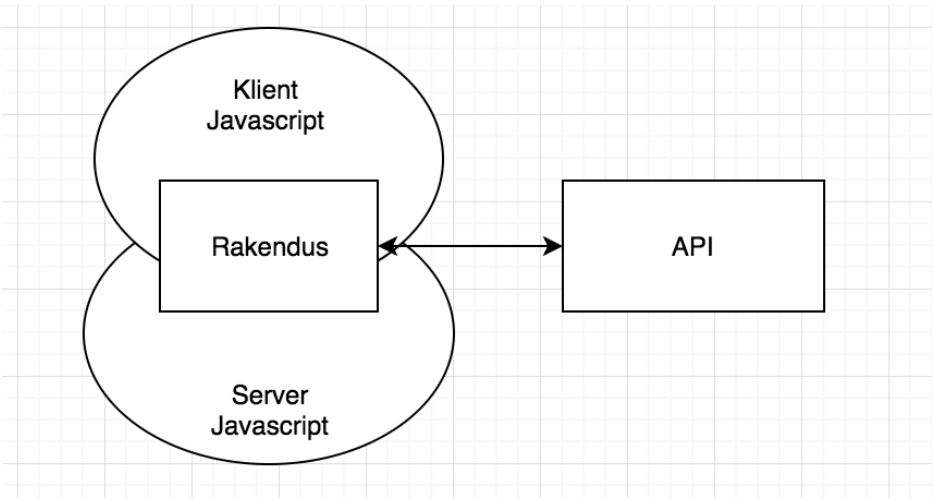
Siiani on populaarsematest otsingumootoritest vaid Google ning Bing uuendanud enda *crawler*'ite metoodikat ning toetavad ka kliendipoolset sünkroonset JavaScripti, suutes DOM'ist lugeda dünaamiliselt lisatud sisu sarnaselt modernsele brauserile [9]. Kuid asünkroonsete rakenduste puhul ei jää *crawler* ootama kuni sisu alla laadimine on lõpetatud ning liigub edasi jättes lehe indekseerimata. Lisaks sellele, et serveri poolt renderatud lehe sisu on nähtaval kõikidele otsingumootorite *crawler*'itele, on lehe sisu nähtav ka sotsiaalmeedia, nagu Facebooki ja Twitteri *crawler*'itele, mis koostavad ning salvestavad vahemällu pildid, pealkirja ja muu info, mida kasutatakse enda platvormidel linkide kuvamiseks [10].

1.2.1 Jõudlus

Serveripoolse renderdamise puhul teeb brauser iga interaktiivse toimingu puhul uue päringu serverisse. Seetõttu tuleb silmas pidada serveri töömahu suurenemist, mis võib omakorda serverile tähendada kõrgemate tehniliste nõuete täitmist ning suurendab serveri ülalpidamise kulusid. Päringu saatmise ning töötlemise kiirus oleneb mitmest erinevast faktorist, nagu kliendi ning serveri vahelise distantsi suurusest ja antud hetkel veebirakendust külastavate kasutajate arvust. Populaarsetel aegadel võib server olla koormatud ning selle tõttu veebirakendus aeglasem. Kuna päritud veebileht renderdatakse serveris, siis peab brauser tegema vähem tööd. Seetõttu sõltub veebilehe esmane laadimiskiirus vähem kasutaja seadmest ja interneti kiirusest. [7]

1.3 Kliendi- ja serveripoolne renderdamine

Tüüpiliselt ei kombineerita serveri- ning kliendipoolset renderdamist põhjusel, et renderdamine toimub erinevates programmeerimise keskkondades ja keeltes. Harilikult on serveripoolne sisu renderdatud keeles nagu PHP, Ruby või Python, kuid kliendipoolse interaktiivsuse lisamine toimub Javascriptiga. Sel juhul on kahe renderdamise viisi kombineerimine äärmiselt kompleksne ning üleliia kulukas. Kasutades aga serveripool Javascripti Node.js keskkonnas, on võimalik neid kahte kombineerida (vt Joonis 3). [3]



Joonis 3. Kliendi- ja serveripoolset renderdamist kasutav rakendus.

Nii peetakse serveripoolsest renderdamisest Javascripti raamistike puhul rääkides silmas lähenemist, kus lehe esmasel laadimisel kuvatakse serveri poolt renderdatud sisu, kuid edasise kasutaja interaktsiooni puhul renderdatakse muudatused kliendi pool. Sellise

lähenemise puhul kirjutatakse universaalset koodi, mis töötab nii serveris kui brauseris. Javascripti, mis suudab joosta nii Javascripti serveripoolse keele Node.js põhjal serveris kui ka brauseris, nimetatakse isomorfseks. Isomorfset Javascripti kasutades on võimalik veebirakenduses kasutusele võtta nii serveripoolse renderdamise jõudlus ja SEO kui ka kliendipoolse renderdamise paindlikkus ning reageerimiskiirus kasutaja toimingutele. [3]

1.3.1 Jõudlus

Kombineerides serveri- ning kliendipoolset renderdamist viisil, kus esmane lehe HTML renderdamine tehakse serveris, kuid interaktiivsuse lisamine toimub kliendi poolel, näeb kasutaja veebilehe sisu kiiresti. Kui esialgne renderdamine toimub serveris, siis näeb kasutaja laetud lehte kiiremini, sest üle võrgu serverisse tehtud päringute arv on minimaalne ning ei teki pudelikaela efekti, kus brauser jääb ootama JavaScripti failide järgi. Selleks, et veebileht interaktiivseks muutuks, peab brauser endiselt alla laadima ning jooksutama Javascript failid, kuid see toimub taustal. *First Meaningful Paint* on veebijõudluse parameeter, mis tähistab olukorda, kus kasutajale tundub, et veebilehe primaarne sisu on nähtav. Mida väikesem on aeg selle parameetrini jõudmiseks, siis seda kiirem leht kasutajale tundub. [11]

1.4 Vue

Vue.js on 2014. aastal avaldatud Javascripti raamistik kasutajaliidese ehitamiseks, mis rõhub oma kergusele ning lihtsusele [12]. Vue tuum keskendub MVC tarkvaramustri põhiliselt vaadete manipuleerimisele. Raamistik on paindlik ning seda on lihtne ühildada teiste teekide ning moodulitega, tehes võimalikuks raamistiku abil keerukate üheleheliste veebirakenduste serveerimise. [13]

Suurte veebirakenduste haldamiseks on tihtipeale vaja kood jaotada väikesteks isoleeritud osadeks. Vue üheks tunnusjooneks on komponentidepõhine arhitektuur. Raamistikus defineeritakse iga komponent, mis sisaldab endas nii HTML, Javascripti kui ka CSS koodi, eraldi failina. See tagab rakenduse koodi modulaarsuse ning hooldatavuse, sest kõik vajalik ühe komponendi renderdamiseks on ühes failis kompaktselt koos. Üks komponent on ehitusblokk, mida on võimalik taaskasutada ning iga kasutusjuhu järgi modifitseerida. Iga komponent on Vue instants, millel on olemas oma nimi. Nii saab komponenti kasutada killukesena suuremas pildis, andes komponendile kaasa info, mis

on vajalik komponendi kuvamiseks just antud kohas. Üks olulisemaid osasid taaskasutatava komponendi juures on registreeritud atribuudid, millele on võimalik määrata vaikimisi väärtused. Samuti on võimalik atribuudid komponendi instantsi välja kutsudes kaasa anda (vt Joonis 3). [13]

```
<modal-button
    :title="'Subscribe'">
</modal-button>
```

Joonis 4. Vue komponendile atribuutide kaasa andmine.

Vue üheks tunnuseks on DOM'i muutmine vähese pingutusega. Vue kasutab HTML'il põhinevat mallide süntaksit, mis võimaldab deklaratiivselt siduda renderdatud DOM'i ning Vue instantsi informatsiooni. Kõnealune raamistik manipuleerib mälus asuvat virtuaalset DOM'i ning veebilehe muutudes selgitatakse välja erinevused vana ning uuenenud virtuaalse DOM'i versiooni vahel. Erinevuste põhjal selgub, millised komponendid muutusid ja kuidas on neid vaja DOM'is uuesti renderdada. [14] Seetõttu ei renderdata iga muudatuse juures kogu veebilehte uuesti, vaid tehakse kindlaks minimaalne vajalike muudatuste arv. Virtuaalne DOM optimeerib brauseris tehtavat tööd, sest mallid kompileeritakse juba arenduse käigus ning brauseril ei kulu selle peale ressursse [15].

Alates 2016. aastast toetab Vue ka serveripoolset renderdamist. Selle läbi pakub raamistik võimalust kliendipoolse ning serveripoolse rakenduse tugevused kombineerida ning ehitada isomorfne rakendus, kus brauseri tehtud esimese päringu töötleb server ning järgnevad kasutaja tehtud päringud töötleb klient [16]. Vue serveripoolse renderdamise puhul renderdatakse esialgse päringu tulemusena Vue komponendid serveri pool HTML stringideks, mis seejärel saadetakse brauserisse, kus JavaScript need hüdreerib. Hüdreerimine on protsess, mille käigus Javascript muudab serverist tulnud staatilise HTML'i interaktiivseks [13].

Võttes kasutusele serveripoolse renderdamise, tuleb teha kompromisse arendamise lihtsuses, rakenduse ehituse arhitektuuris ning ka serverikoormuses, sest serveri ülesandeks ei ole enam vaid staatiliste failide serveerimine, vaid tööülesannete täitmine, mida varem tegi suures osas brauser. Arvestada tuleb ka asjaoluga, et osa rakenduse

koodist võib olla brauserile spetsiifiline ja seda ei ole võimalik serveris jooksutada. Serveripoolse renderdamise arenduse oluliseks nõudeks on Node.js keskkonna olemasolu. [13]

2. Kasutatud tehnoloogiad

Vue serveripoolsete lahenduste rakendamisel ning nende tulemuste hindamisel kasutas autor erinevaid tehnoloogiad. Järgnevates alampeatükkides annab autor ülevaate kasutatud tehnoloogiatest koos kommentaaridega.

2.1 Webpack

Webpack on tööriist, mida kasutatakse JavaScriptil põhinevas rakenduses staatiliste moodulite nagu skriptide, CSS, fontide, piltide ja paljude teiste failitüüpide kokku pakkimiseks. Webpackil on konfiguratsioon, kus on võimalik määrata, kuidas spetsiifilisi faile laadida. Tööriist töötleb rakendust ning ehitab sõltuvuste graafiku, mis kirjeldab rakenduste moodulite sõltuvust ning seoseid [17]. Seejärel koostab webpack kokku pakitud failidest *bundle* failid, mis lisatakse rakendusele skriptidena [18].

Autor leiab, et serveripoolsete lahenduste rakendades oli webpacki kasutamine ajakokkuhoidlik, sest tööriist automatiseerib failide kompileerimist. Samuti kasutas autor webpacki pluginaid nagu *UglifyJS*, et vähendada ning suruda kokku suured Javascipt failid, nagu Vue teek, vähendades sellega ka skriptide alla laadimise aega brauseris. Lisaks toetus suur osa Vue ametliku serveripoolseks renderdamist mõeldud mooduli vue-server-renderer seadistamine webpacki konfiguratsioonile, kus nii kliendi- kui serveripoolseks renderdamiseks oli vaja üles seada erinev konfiguratsioon.

Autor leiab, et rakenduses webpacki kasutamine oli oluline, sest paljud webpack spetsiifilised funktsioonid, nagu failide importimine kasutades *file-loader*'it ning CSS importimine kasutades *css-loader*'it ei tööta otse Node'ga. Lisaks on webpacki kasutades lihtne transpileerida kliendipoolset koodi ka vanematele brauseritele. Autor kasutas antud lõputöö raames arendatud prototüüp rakendustes webpacki Vue komponentide, piltide ning skriptide töötlemiseks üheks *bundle* failiks.

2.2 Express

Express on paindlik raamistik Node'l põhinevatel serveripoolsete veebi- ning mobiilirakenduse ehitamiseks. Expressi eesmärk on arendusprotsessi lihtsustada pakkudes arendajale funktsioone näiteks HTTP päringute töötlemiseks, marsruutimiseks, mallide ning vaadete käsitlemiseks. Raamistik aitab kirjutada turvalist, modulaarset ning kiiret rakendust. Express annab arendajale palju vabadust, kuna ei suru peale mingi kindla tarkvaramustri või struktuuri jälgimist. [18]

Express toetab ka erinevaid mallide süsteeme, mis võimaldavad staatiliste mallide kasutamist. Need mallid sisaldavad serveripoolseid muutujaid, mis asendatakse renderdamise käigus muutujate tegelike väärtustega. [18] Selline lähenemine muutujate kasutamiseks HTML koodis tekitab vähem müra ning soodustab koodi järjepidevust ja organiseeritust.

Kõnealune raamistik võimaldab ka arendajal kasutada rakenduses *middleware*'e, ehk manipuleerida *request* ning *response* objekte [18]. Antud töö käigus kasutas autor Express serverit HTTP päringute töötlemiseks, *middleware*'de kasutamiseks ning staatiliste failide serveerimiseks.

2.3 Lighthouse

Veebilehe laadimiskiiruse mõõtmiseks kasutas töö autor Google poolt arendatud vabavara Lighthouse. Lighthouse on automatiseeritud tööriist, mille eesmärk on veebilehtede kvaliteedi paremaks muutmine. [19] Tööriista on võimalik kasutada nii lokaalsetel, avalikel või autentimist vajavatel veebilehtedel. Lighthouse teeb auditeid veebilehe jõudluse, kättesaadavuse, SEO, progressiivsuse ning heade tavade, nagu näiteks HTTPS protokolli kasutamine kohta. Autor otsustas veebilehe laadimiskiiruse mõõtmise tööriistade valikul Lighthouse kasuks, sest tegemist on uudse ning mugava Google poolt arendatava tööriistaga, mille läbi viidavad auditid, nagu veebilehe laadimiskiirus, ühtisid lõputöö eesmärkidega.

Lighthouse on võimalik jooksutada Google Chrome arendaja tööriistades, käsurealt või Node moodulina. Töö autor otsustas kasutada Lighthouse Google Chrome arendaja

tööriistades, sest see keskkond oli talle varasemalt tuttav ning selline kasutus nõudis kõige minimaalsemat konfigureerimist. Seejärel teeb tööriist mitmeid auditeid mõõdetava veebilehe kohta, mille tulemuste põhjal koostatakse raport. Raportist on võimalik näha millised auditid õnnestusid ja millised põrusid. Iga auditi kohta on eraldi dokumentatsioon, mis selgitab testitava nõude olulisust ning annab soovitusi selle nõude täitmiseks. Lighthouse kasutab auditide läbi viimiseks emulaatorina seadet Nexus 5S. [19]

3. Vue serveripoolse renderdamise võimalused

Käesoleva peatüki järgmistes punktides uuris bakalaureuse töö autor kolme erinevat viisi, kuidas implementeerida serveripoolset renderdamist Vue'l põhinevas rakenduses. Uurimuse inspiratsiooniks on eluline probleem ettevõttes Pipedrive, kus soovitakse lähitulevikus implementeerida serveripoolne renderdamine.

Nimetatud ettevõte eesmärgiks on oma veebilehe laadimisaja vähendamine. Hetkel on Pipedrive veeb Node+Express keskkonnas jooksev veebirakendus, mis kasutab osaliselt Vue komponente ning osaliselt Expressi poolt pakutud EJS malle. Ettevõtte sooviks on lähitulevikus Vue komponentide osakaalu suurendada, eesmärgiga minna üle täielikult Vue kasutusele. Komponentide arvu kasvades suureneb ka skriptide maht, mida brauser peab kliendi poolel alla laadima ning jooksutama selleks, et kasutajale lehte kuvada. Seetõttu on uute Vue komponentide kasutuselevõtt veebilehte aeglasemaks muutnud. Veebilehe kiiruse tõstmiseks ning optimeerimiseks ka tuleviku perspektiivis, kus jätkatakse uute Vue komponentide loomist, on ettevõte enda vajadusi ning võimalusi silmas pidades otsustanud rakendada oma veebilehel Vue serveripoolset renderdamist.

Bakalaureusetöö eesmärgiks on tutvuda erinevate Vue serveripoolse renderdamise variantide omadustega, mida tuleks põhjalikumalt hinnata enne lahenduse kasutusele võtmist. Kuna Vue serveripoolse renderdamise funktsionaalsus on töö kirjutamisel hetkel alles uudne ning kohati tundmatu, siis on oluline teha korralik eeltöö võimalike implementatsiooni meetodite kohta. Suurtes ettevõtetes ning kompleksetes veebirakendustes on tarkvaralahenduse rakendamine kulukas ja aeganõudev. Põhjalikku uurimist tehes, kus töödeldakse läbi implementeerimise protsess, on võimalik kindlaks teha iga variandi võimalused ning piirangud, ennetades ootamatuste tekkimist lõpprakenduses. Järgnevalt on väljatoodud serveripoolsete lahenduste uuritavateks ja hinnatavateks omadused:

1. **Lihtsus õppimisel ja arendamisel.** Uue tarkvaralahendusega tutvumine, selle õppimine ning arendamine võtab aega. On oluline, et nendele tegevustele kulutatud aeg oleks minimaalne.

2. **Kogukonna suurus ning aktiivsus.** Kommuuni suurus näitab tarkvaralahenduse populaarsust ning töökindlust. Lisaks on suurema kogukonna puhul lihtsam probleemide tekkides abi leida.

3. **Sobivus olemasolevale rakendusele.** Antud töö raames uuritakse variantide sobivust olemasolevale rakendusele. On tähtis, et varianti oleks võimalik integreerida juba kasutusel oleva tarkvaraga.

4. **Jõudlus.** Serveripoolse renderdamise lahenduse valimisel on tähtis mooduli või raamistiku jõudlus selleks, et veebilehe laadimiskiirust tõsta.

Serveripoolse renderdamise variantide välja valimisel tugines autor nende populaarsusele ning sobivusele töötamaks Express serveriga. Antud kriteerium põhines nii asjaolust, et Pipedrive veebirakendus jookseb Node + Express keskkonnas, kui ka Express serveri kasutuse populaarsusest. Nendeks variantideks on:

1. Vue-server-renderer moodul
2. Express-vue moodul
3. Nuxt.js raamistik

Ülalmainitud omaduste hindamise eesmärgil rakendas autor iga variandi eraldi viies Vue kliendipoolse prototüüp rakenduse üle serveripoolsele renderdamisele. Järgnevates alampeatükkides annab autor koos oma kommentaaridega ülevaate iga variandi arendusprotsessist.

3.1 Vue-server-renderer

Vue-server-renderer on Vue poolt arenendatud moodul rakenduse serveripoolseks renderdamiseks. Kuna Vue rõhub kasutajaliidese ehitamisele ning oma kergusele, siis ei ole vue-server-renderer osa Vue tuumast ning see tuleb eraldi sõltuvusena rakendusele lisada. Selleks kasutas autor *Node Package Manageri* (NPM), mis on tööriist Node.js

moodulitega seotud tegevuste, nagu alla laadimine ning uuendamine, automatiseerimiseks.

Selleks, et kõnealune moodul olemaolevas rakenduses implementeerida, arendas töö autor esmalt lihtsa Vue juurkomponendi *App.vue*, mille ülesanneteks oli määrata * * konteiner, kuhu renderdatakse lehel vajaminevad komponendid ning element, mille *id=app*, mida kasutatakse kliendipoolselt renderdatud osa *mount*'imiseks DOM'i. Selline protsess on serveripoolse renderdamise puhul tuntud ka kui kliendipoolne hüdreerimine. Hüdratsiooni käigus võtab Vue serveri poolt saadetud staatilise HTML'i ning muudab selle dünaamiliseks ning interaktiivseks DOM'iks.

Kuna Vue vaadet renderdades genereeritakse serveris vaid selle vaate *markup*, siis tuli luua HTML mall, mida kasutatakse iga lehe põhjana. Oluline on, et see mall sisaldaks endas kommentaari *<!--vue-ssr-outlet-->*, mis tähistab kohta, kuhu renderatud Vue vaade sisestada (vt Joonis 5). Samuti on võimalik selles mallis ära määrata globaalsed metaandmed. Autori jaoks oli üllatav, et kõnealune moodul ei võimalda igale lehele eraldi metaandmete seadmise võimalust. Sellise funktsionaalsuse lisamiseks tuleb selleks enda rakenduses ise lahendus välja mõelda või kasutada mõnda muud moodulit, nagu näiteks vue-meta, et seda teha.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport"content="user-scalable=no, initial-scale=1 width=device-width">
    <meta name="format-detection" content="telephone=no">
    <title>{{ 'Test vue-server-renderer' }}</title>
</head>
<body>
    <!--vue-ssr-outlet-->
</body>
</html>
```

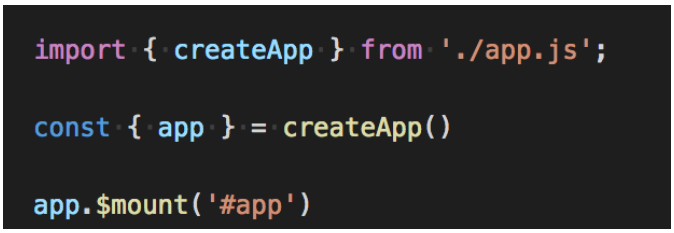
Joonis 5. Vue-server-renderer mooduli rakendusel loodud mall.

Serveripoolse koodi kirjutamisel peab arendaja silmas pidama, et iga serveri päringu jaoks on vaja uut rakenduse instantsi. See probleem puudub kliendi pool, kus brauser hindab rakenduse instantsi iga päringuga uuesti. Eelmainitud probleemi vältimiseks, lõi töö autor eraldi *createApp()* funktsiooni, mis tagastab uue Vue juurkomponendi instantsi (vt Joonis 6). Selle funktsiooni tõstis autor eraldi *app.js* faili, kuhu lisataks tulevikus ka muu põhiline funktsionaalsus, mida on vaja kasutada nii serveri kui kliendi poolel.

Joonis 6. Vue-server-renderer mooduli rakendamisel loodud createApp() funktsioon.

Seejärel lõi autor serveripoolse skripti faili *server-entry.js*, kus eksporditakse funktsioon, mille sees luuakse äsja loodud *createApp()* funktsiooni abil uus Vue instants. Nii saab eksporditud funktsiooni välja kutsuda iga uue render päringuga, tagastades iga kord uue Vue instantsi.

Kliendipoolse skripti faili *client-entry.js* sisuks on eelmainitud funktsiooni *createApp()* kasutades uue instantsi loomine ning selle *mount*'imine juurkomponendi *App.vue* külge (vt Joonis 7). Siinkohal on oluline, et *mount*'imisele ette antud selektor *#app* sisalduks ka *App.vue* komponendis. Sel juhul ei loo klient ise uuesti DOM elemente, vaid hüdreerib juba serveri poolt loodud staatilise HTML'i.



Joonis 7. Vue-server-renderer mooduli rakendamisel kliendipoolne hüdreerimine.

Vue-server-renderer mooduli kasutamine rakenduse üleviimisel serveripoolsele renderdamisele toetus suures jaos webpacki konfigureerimisele. Selleks tuli üles seada nii kliendile kui serverile erinev konfiguratsiooni fail. Kliendi pool kastusele võetava skripti määramiseks, lisas autor webpacki kliendipoolsesse konfiguratsiooni *entry* kohale *client-entry.js.* Samamoodi tuli lisada serveripoolsesse webpacki konfiguratsiooni *entry* kohale fail *server-entry.js.* Määratud failid on skriptid, mille ülesandeks on vastavalt kliendi ja serveri pool lehe renderdamise loogika täitmine.

Viimase tähtsa lülina konfigureeris autor serveripoolse renderdamise failis *server.js*. Selleks tuli iga serveri päringu korral välja kutsuda *createApp()* funktsioon, mille täitmise

lõpetades renderdatakse äsja loodud Vue instants *renderToString()* funktsiooni abil HTML stringiks ning saadetakse brauserile.

3.2 Express-vue

Express-vue moodul on mõeldud vaid Vue serveripoolseks renderdamiseks Express serverit kasutades. Mooduli eesmärk on pakkuda suurtele skalaaruvatele veebirakendustele võimalust mugavalt kombineerida Vue ning Node+Express. Mooduli üheks kasutusjuhuks on MVC tarkvara arhitektuurimustri põhjal ehitatud rakenduse üle viimine Vue serveripoolsele renderdamisele. Sel juhul saab vaadeteks kasutada Vue komponente, kuid rakenduse loogika eest vastutavad Node'l põhinevad kontrollerid ning mudelid. [20]

Expressi serveri kasutamisel on võimalik ühena paljudest Expressi poolt pakutud funktsionaalsustest kasutada mallide süsteeme. Need mallid on renderdatud serveri pool ning nende abil on võimalik HTML dokumendile lisada lihtsalt ja loetavalt serveripoolseid muutujaid, mis *runtime* ajal asendatakse tegelike muutujate väärtustega. Mainitud mallide HTML stringiks renderdamiseks kutsutakse serveris esile *res.render()* funktsioon. [18] Express-vue moodul võimaldab lisaks Expressi mallidele renderdada ka serveris Vue faile. Sel juhul kasutatakse eelnevas peatükis mainitud vue-server-renderer funktsiooni *renderToString()* asemel express-vue mooduli *renderToVue()* funktsiooni, selleks, et Vue fail HTML stringiks renderdada. Kõnealust moodulit kasutades koheldakse <template> *tag*'ide sisu HTML'ina, seejärel teisendatakse <script> *tag*'i sisu Vue süntaksiks ning <style> *tag*'i sisu lisatakse hiljem juba renderdatud HTML stringile. [21]

Mooduli abil serveripoolse renderdamise rakendamiseks laadis autor express-vue alla kasutades Node Package Manager'i, initsialiseeris selle rakenduses *server.js* failis ning võttis instantsi kasutusele *middleware'ina* (vt Joonis 8).

```
const expressVue = require('express-vue');
const app = express();
const expressVueMiddleware = expressVue.init();

app.use(expressVueMiddleware);
```

Joonis 8. . Express-vue mooduli rakendamisel middleware'i kasutusele võtmine.

Kasutades express-vue mooduli funktsiooni *renderVue()*, andis autor serverisse tulnud päringule ette Vue komponendi, mida renderdada. Lisaks andis autor kaasa ka *vueData* ning *vueOption*s objektid, mis sisaldavad endas infot, mida on võimalik Vue komponendiga dünaamiliselt siduda (vt Joonis 9). Nii on võimalik anda komponendile kohe kaasa *data* objekt, mis lisab komponendi enda *data* atribuutidele kaasaantava info. *Data* objekt võib sisaldada näiteks infot päringu teinud seadme kohta, analüütika, tõlgete ja muu komponendi tõrgeteta tööks vajaliku kohta.

```
app.get('/about', (req, res, next) => {
    vueOptions = vueHelper.getVueOptions();
    vueData = vueHelper.getVueData();
    res.renderVue('../../src/components/About.vue', vueData, vueOptions);
})
```

Joonis 9. Express-vue mooduli rakendamisel renderVue() funktsiooni väja kutsumine.

RenderVue() funktsioonile kaasa antud objektis *vueOptions* on võimalik Vue komponendi renderdamisele kaasa anda mitmed veebilehe tähtsad osad. Antud objektis on võimalik kohandada lehe HTML'i ning *tag*'ide sisu. Nii saab lihtsa vaevaga määrata ära näiteks metaandmed, skriptid ja stiilid. Samuti toetab *vueOptions* objekt struktrueeritud info seadmist, mis jälgib Google poolt seatud standardformaati selleks, et pakkuda otsingumootorile vihjeid lehe sisu kuvamise kohta [22].

Lisaks on autori arvates kõnealune moodul lahendus olukorrale, kus ühes rakenduses peab töötama paralleelselt mitu mallide süsteemi. Nii saab vajadusel ühes kontrolleris kasutades *res.renderVue()* funktsiooni renderdada Vue'l põhinevat vaadet ning teises kasutades *res.render()* funktsiooni Express mallil põhinevat vaadet.

3.3 Nuxt

Nuxt on Vue peale ehitatud raamistik, mis on mõeldud universaalsete Vue rakenduste loomiseks. Raamistiku eesmärk on üheleheliste universaalsete rakenduste, kus rakendus renderdatakse nii serveri kui kliendi poolel, arenduseprotsessi lihtsutamine. Nuxt'i kasutades saab arendaja keskenduda vaid rakenduse loogikale ning serveri- ja kliendipoolse koodi jaotuse pärast hoolitseb raamistik. [23]

Nuxt'i kasutades ei pea arendaja ise serveripoolset renderdamist konfigureerima, sest see on raamistikus eelseadistatud. Kõnealune raamistik ei keskendu ainuüksi serveripoolsele renderdamisele, vaid see on üks mitmest funktsionaalsusest, mida Nuxt pakub. Raamistik hõlmab endas nii marsruutimise, serveripoolse konfigureerimise kui ka andmeladude loogikat, pakkudes rakenduse arenduseks kasutajamugavat eelseadistatud keskkonda. Lisaks eelnevalt mainitule on Nuxt'i üks populaarsemaid funktsionaalsusi staatiliste lehtede renderdamine, mille puhul luuakse arendusprotsessi ajal Vue failidest staatilised HTML lehed, mida on seejärel võimalik kuvada ilma serverita. Lisaks on raamistikule omasteks tunnusjoonteks asünkroonse *data* alla laadimine ning *middleware*'de, mille funktsioone saab esile kutsuda enne komponentide renderdamist, lisamine.

Nuxt'i on võimalik lisada Express serveril jooksvale Vue rakendusele *middleware*'ina. Selleks tuleb *server.js* failis importida Nuxt moodulist Nuxt ning *Builder*. Nuxt nõuab konfiguratsiooni faili nimega *nuxt.config.js*, kus autor määras skripti failid, keskkonna muutujaid, metaandmed, pluginad ja muu rakenduse sujuvaks tööks vajaliku info (vt Joonis 10).

```
module.exports = {

  head: {
    title: 'Nuxt serveripoolne renderdamine',
    meta: [
    { charset: 'utf-8' },
    { name: 'viewport', content: 'width=device-width, initial-scale=1' },
    { hid: 'description', name: 'description', content: 'Nuxt.js project' }
    ],
    link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }
    ],
  },
  css: ['~/src/assets/styles/app.scss']
  }
```

Joonis 10. Nuxt'i rakendamisel loodud nuxt.config.js.

Seejärel initsialiseeris autor eelmainitud konfiguratsiooniga uue Nuxt instantsi ning võttis kasutusele *nuxt.render middleware*'i (vt Joonis 11).

```
const { Nuxt, Builder } = require('nuxt');
const config = require('./nuxt.config');
const nuxt = new Nuxt(config);

config.dev = !(process.env.NODE_ENV === 'production')

app.use(nuxt.render)
```

Joonis 11. Nuxt'i rakendamisel instantsi loomine ning middleware'i kasutusele võtmine.

Järgmisena lõi autor projekti juurkausta uue kausta *pages*, mille sisuks on Vue vaadete failid. Nimelt otsib lehte pärides Nuxt selle nimelist kausta ning vastavalt URL'ile renderdatakse samanimeline Vue fail. Näiteks brauseris */about* lehele minnes töötleb serveri päringut Nuxt, renderdab *about.vue* komponendi HTML stringiks ning saadab selle kliendile. Nii on Nuxti abil võimalik ka lihtsasti marsruutida, seadistamata ise selleks eraldi loogikat.

4. Analüüs

Järgnevates alampeatükkides hindab autor vue-server-renderer, express-vue ning Nuxt võimaldatud serveripoolsete lahenduse omadusi. Hinnatavateks ning analüüsitavateks omadusteks on õppimise ja arendamise lihtsus, kogukonna suurus ja aktiivsus, sobivus olemasolevale rakendusele ning lahenduse jõudlus.

4.1 Lihtsus õppimisel ja arendamisel

Enne uue lahenduse implementeerimist tuleb arendajal vastava raamistiku või mooduliga tutvuda ning seda õppida. On oluline, et antud raamistiku või mooduli konfigureerimisprotsess oleks kiiresti hoomatav ning rakendatav. Keerulisema protsessi puhul on tähtis ka korraliku dokumentatsiooni olemasolu, millele küsimuste tekkides toetuda.

Autori hinnangul võttis eelnevas peatükis seadistatud serveripoolse renderdamise variantide implementeerimisel enim süvenemist ning lisamaterjalide otsimist vue-server-renderer. Kõnealuse mooduli abil serveripoolse renderdamise konfigureerimine toimus läbi mitmete failide ning koosnes paljudest erinevatest aspektidest. Kuna kõnealuse mooduli rakendamine nõudis teadmisi webpacki tööst, siis võib antud teadmiste puudumise puhul õppimisprotsess veelgi pikeneda.

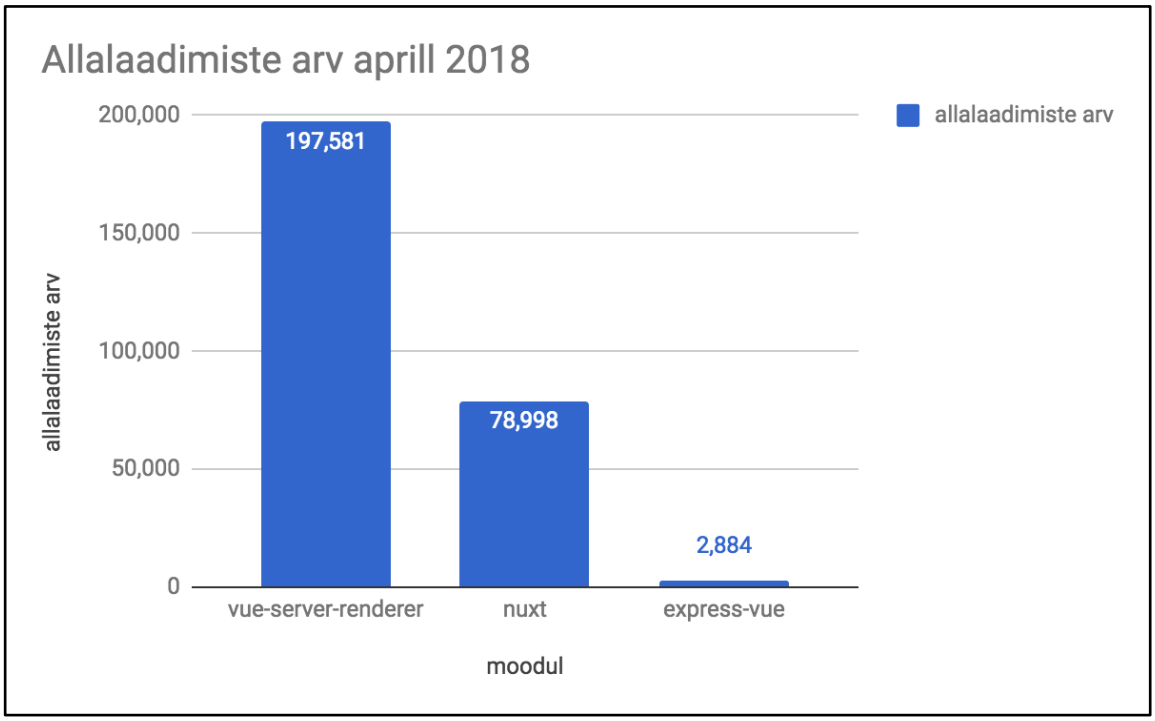
Kõige lühem õppimisprotsess ning kõige lihtsam seadistamine serveripoolseks renderdamiseks oli autori arvates express-vue moodulit kasutades. Mooduli rakendamine serveripoolse renderdamise implementeerimiseks oli lihtne ning arusaadav, sest lisatud koodihulk on väike ning puudub keeruline konfigureerimise protsess. Lisaks arvab autor, et antud mooduli seadistamise hõlpsust võib suurendada olemasolev kogemus Express mallide kasutamisega. Sel juhul ei ole antud mooduli idee võõras ning on lihtsasti hoomatav.

Bakalaureuse töö autori hinnangul oli Nuxt'i konfigureerimine eksisteerivas rakenduses algselt segane ning nõudis laiemat arusaama kogu raamistiku ehitusest. Sel juhul on õppimisprotsess pikem, sest tutvuda tuleb raamistiku ülesehituse, kasutatavate tehnoloogiate ning muude iseärasustega. Pikemalt süvenedes oli serveripoolse renderdamise implementatsioon ise väikese mahuline ning mugav, kuid tuli arvestada, et lisaks Nuxt'i implementeerimisele tuleb viia läbi ka rakenduse struktuuris muudatused. See võib kompleksema rakenduse puhul seadistamise protsessi keerulisemaks muuta. Lisaks leiab töö autor, et Nuxt'i implementeerimine ainult serveripoolse renderdamise eesmärgil on sobilik juhul, kui arendaja soovib kasutajamugavat implementatsiooni ning ei soovi süveneda konfigureerimise detailidesse.

Võrreldes erinevaid serveripoolseid renderdamise viise Vue rakenduses, leidis autor, et teistest oli märkimisväärselt lihtsam express-vue mooduli kasutamine. Kõige keerulisem õppimise ning seadistamise protsess oli rakendades Vue ametlikku vue-server-renderer moodulit. Autori hinnangul oli Nuxt'i kasutades raamistikuga tutvumine aeganõudev, kuid tegelik rakendamine mugav, sest arendaja ei pea süvitsi laskuma konfigureerimise protsessi.

4.2 Kogukonna suurus ning aktiivsus

Kogukonna suurus ning aktiivsus on oluline faktor uue lahenduse kasutusele võtmisel. Probleemide või küsimuste ilmnemisel on tähtis, et leiduks kaasmõtlejaid ning abistajaid. Suurem kasutajate hulk soodustab ka teemakohaste blogi- ning foorumipostituste kirjutamist, millest võib spetsiifiliste küsimuste puhul palju abi olla. Samuti võib olla kogukonna suurus indikaatoriks lahenduse töökindlusele. Autor hindas kogukonna suurust NPM allalaadimiste arvu 2018. aprillikuu statistika põhjal (vt Joonis 12).



Joonis 12. Moodulite alla laadimiste arv 2018 aprillis [24].

Joonisel 12 välja toodud statistika andmetel on populaarseim moodul Vue serveripoolse renderdamise rakendamiseks vue-server-renderer, mida on 2018. aprillikuus alla laetud 197 581 korda. Enam kui poole vähem on alla laetud Nuxt raamistikku ning express-vue osakaal moodustab kolme variandi allalaadimiste arvust kokku vaid 1,5%.

Autori hinnangul oli võrdlemiseks võetud variantide rakendamise käigus otsitud materjalide ning abistavate ressursside arv korrelatsioonis NPM statistikaga. Mooduli populaarsus on kindlasti mõjutav tegur dokumentatsiooni ning näidete koostamisel. Väikesed moodulid, nagu express-vue on arendatud tihti ühe või paari inimese poolt, kes ei rõhu väga põhjalikule dokumentatsioonile, kus on kaetud mitmed erinevad kasutusjuhud. Seetõttu oli kõnealuse mooduli implementeerimise juhiseks ainuke relevantne materjal mooduli Githubi *repository*'s asuv *README* dokumentatsiooni fail. Saadaval olev dokumentatsioon põhines peamiselt koodinäidetele ning oli napisõnaline.

Kirjeldatud variantidest on kõige põhjalikum dokumentatsioon vue-server-renderer mooduli kasutamise kohta. Kõnealune materjal on Vue ametlik serveripoolse renderdamise dokumentatsiooni. Autori hinnangul oli põhjalikust õpetusest hoolimata vue-server-renderer mooduli rakendamise kirjeldust raske jälgida, sest lähtutakse konfigureerimise samm-sammult juhendamisega, kus mõne uue aspekti ilmnedes

eelmised sammud ümber kirjutatakse. Sellist dokumenteerimise viisi on autori arvates keeruline hoomata, kui seadistamine ei toimu sõltuvalt rakenduse iseloomust täpselt samalaadselt ning tekib vajadus dokumentatsiooni ühte- ning teistpidi lugeda.

Põhjalik dokumentatsioon on olemas ka Nuxt raamistikul. Nuxt'i jaoks on palju materjale ning ka videoõpetusi, kuid valdav osa neist keskendub rakenduse nullist ehitamisele. Seetõttu kulus töö autoril Nuxt'i serveripoolse renderdamise rakendamiseks aega, sest eelnevalt uuritud võimalus rakendada Nuxt'i kasutades *middleware'i* ei olnud ametlikus dokumentatsioonis ilmne ning selle kohta puudusid ka täiendavad lisamaterjalid.

NPM veebilehelt saadud statistika ning autori kogemuse põhjal saab öelda, et suurim kogukond koos lisamaterjalidega on vue-server-renderer moodulil. Enam kui poole väiksem kogukond on Nuxt raamistikul. Suure kogukonna puudumine on märgatav eriti express-vue mooduli puhul, kus probleemide ilmnedes tuleb mooduli autoriga kontakti võtta või moodulisse ise panustada.

4.3 Sobivus olemasolevale rakendusele

Soovides võtta kasutusele serveripoolne renderdamine veebilehe kiiruse tõstmise eesmärgil, rakendatakse see lahendus ilmselt juba olemasolevasse rakendusse, mitte ei hakata uut rakendust ehitama. Seega on moodulit või raamistikku hinnates oluline, et neid oleks võimalik implementeerida juba eksisteerivas veebirakenduses.

Vue-server-renderer mooduli implementeerimine koosnes paljudest erinevatest aspektidest, mille puhul on kompleksema rakenduse korral eksimise oht suur. Samas leiab autor, et kõnealuse variandi rakendamisel on arendajal palju kontrolli oma rakenduse struktuuri ning loogika säilitamise osas, mis teeb esialgse seadistuse küll keerulisemaks, kuid võimaldab serveripoolset renderdamist kohandada just vastavalt antud rakenduse vajadustele.

Autori hinnangul oli kõigist kolmest variandist kõige mugavam olemasolevas rakenduses implementeerida express-vue moodulit. Seda mõjutab asjaolu, et moodul rõhubki kasutuslihtsusele skaleeruvates veebirakendustes. Lisaks leidis autor implementatsiooni käigus, et mooduli ülesehituse idee võimaldab vastavalt marsruudile määrata

renderdamiseks Vue malli või Express malli. Selline paralleelselt töötav lahendus võib olla kriitilise tähtsusega kompleksetele rakendustele, kus head tarkvara arenduse põhimõtteid jälgides viiakse rakendus serveripoolsele renderdamisele üle samm-sammult.

Vastupidiselt vue-server-renderer ning express-vue moodulitele nõuab Nuxt kindla kaustade struktuuriga projekti seadistust, mille rakendamine võib suuremahuliste ning kompleksete rakenduste puhul olla aeganõudev ning keeruline. Lisaks arvab autor, et raamistikku vaid serveripoolse renderdamise eesmärgil seadistamisel võib tekkida olukord, kus arendaja peab Nuxti pakutud lisafunktsionaalsuste piiramiseks lisatööd tegema selleks, et rakenduse loogika säilitada. Seetõttu näeb autor raamistikus potentsiaali just uue Vue põhise rakenduse ehitamisel, mille puhul hoitaks palju aega kokku funktsionaalsuste, nagu marsruutimine, andmeladude ning serveripoolne renderdamine pealt.

Autor leiab, et antud variantidest on kõige mugavam rakendada olemasolevasse rakendusse express-vue moodulit. Lisaks hindab autor kõrgelt mooduli vue-server-renderer pakutavat vabadust. Kõige vähem sobivamaks variandiks olemasoleva rakenduse jaoks peab autor Nuxt raamistikku, mis võib nõuda rakenduses suuri struktuurimuudatusi.

4.4 Jõudlus

Eesmärgiga rakendada serveripoolset renderdamist just veebilehe laadimiskiiruse tõstmisega, on oluline faktor valiku tegemisel lehe laadimiskiiruse hindamine. Kuna käesoleva bakalaureusetöö eesmärk on serveripoolse renderdamise lahenduste uurimine veebilehe laadimise kiiremaks muutmiseks, siis toob siinkohal autor võrdlusesse ka Vue'l põhineva kliendipoolse rakenduse jõudluse.
Jõudluse mõõtmiseks lõi autor igas prototüüp rakenduses identse veebilehe, mis koosnes autori ehitatud Vue komponentidest. Testitaval lehel kasutati korduvalt kolme erinevat komponenti, millel kõikidel olid oma stiilid ning atribuudid. Joonisel 13 on välja toodud väljavõte autori arendatud Vue komponendist, mida kasutati veebilehe kokku panemisel (vt Joonis 13).

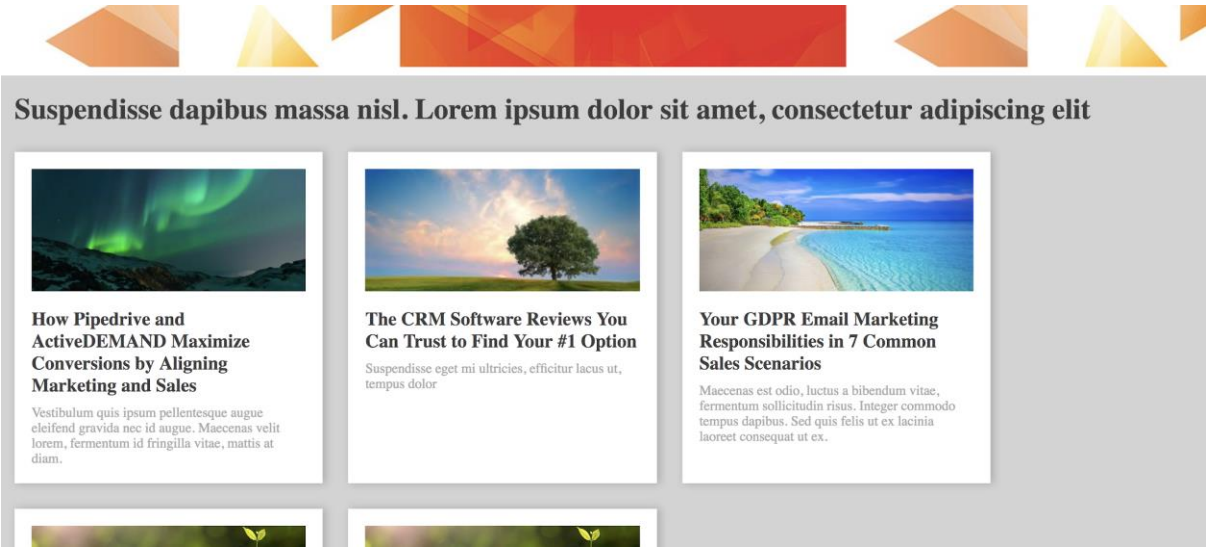
```
<template>
    <div class="content-wrapper">
        <div class="cards-small">
            <div v-if="title"
            class="title">
            {{ title }}
            </div>
            <div class="items">
                <a v-for="item in items"
                class="item"
                :href="item.url">
                    <div v-if="item.featureImage"
                        class="feature-image"
                        :style="{
                        backgroundImage: 'url(' + item.featureImage + ')'
                        }">
                    </div>
                    <div class="item-title">
                        {{ item.title }}
                    </div>
                    <div class="item-subtitle">
                        {{ item.subtitle }}
                    </div>
                </a>
            </div>
        </div>
    </div>
</template>
```

Joonis 13. Väljavõte Vue komponendist, mis kuvab infot kastidena.

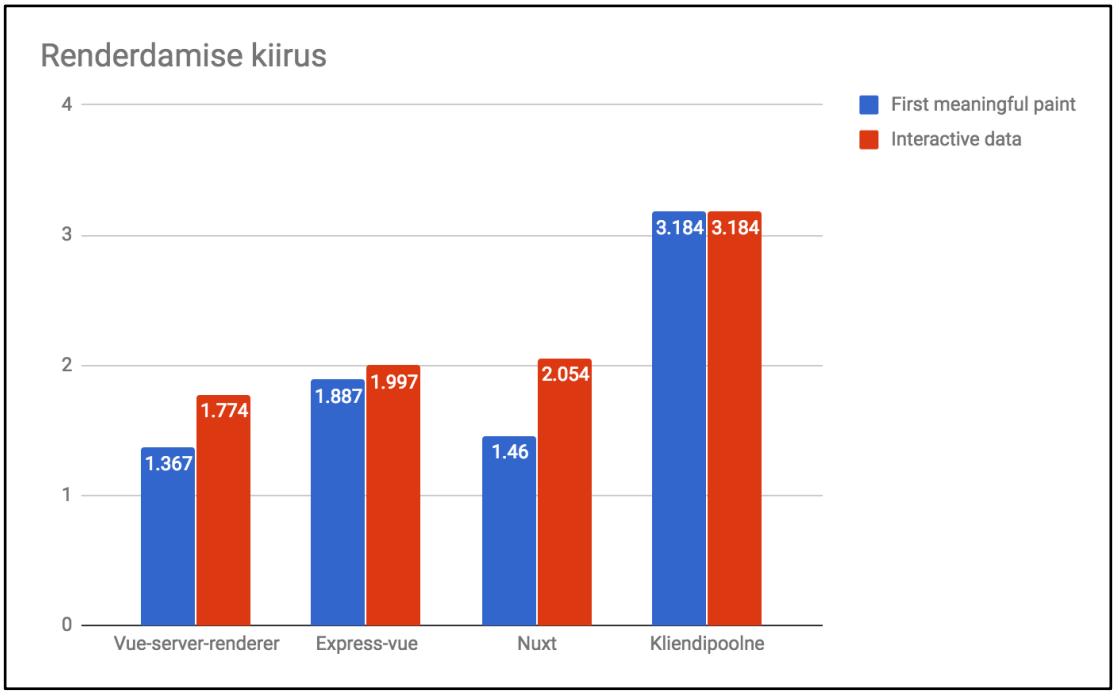
Testimise eesmärgiks oli välja selgitada iga rakenduse jõudlus veebilehe kuvamiseks mõõtes veebilehel kahte faktorit:

- **First meaningful paint.** See parameeter tähistab aega, mis brauseril kulus kuvamaks kasutajale tähenduslikku sisu.
- **First interactive data.** Antud parameeter tähistab aega, mis brauseril kulus veebilehe minimaalselt interaktiivseks muutmiseks. Selle punkti saavutades on suurem osa lehe elementidest interaktiivsed. [25]

Kuna kõik prototüüp rakendused on valminud samadel tingimustel, on ainuke laadimist mõjutav faktor serveripoolse renderdamise moodul või raamistik ise. Võrreldes serveripoolset renderdamist kasutavaid rakendusi kliendipoolset renderdamist kasutava rakendusega on võrdluspunktiks renderdamise laad ise. Iga variandi jõudluse hindamiseks mõõtis töö autor testitava veebilehe (vt Joonis 14) laadimise kiirust brauseris Chrome kasutades selleks automatiseeritud tööriista Lighthouse. Kõikide rakenduste puhul viis autor katse läbi kolm korda ning arvutas välja keskmise väärtuse (vt Joonis 15).



Joonis 14. Kuvatõmmis jõudluse testimiseks loodud veebilehest.



Joonis 15. Veebilehe kiiruse mõõtmise tulemused.

Mõõtmise tulemusena selgus, et eelnevas peatükis implementeeritud lahendustest pakkus kõige kiirema lehe esmase laadimiskiiruse vue-server-renderer moodul. Mooduli abil renderdatud lehe laadimiskiirus oli üle poole, ehk pea 57% kiirem kui sama veebilehe renderdamine kliendipoolselt. Võrreldes vue-server-renderer laadimiskiirust express-vue mooduliga, oli laadimiskiirus 27,6% kiirem. Kõige väiksem erinevus oli vue-server-rendereri ning Nuxt renderdatud lehe laadimiskiiruses, kus vue-server-renderer puhul võttis lehe laadimine 6,4% vähem aega.

Mõõtmiste analüüsist selgus, et kõige kiiremini laadis testitav veebileht vue-server-rendereri ning Nuxt'i kasutavates veebirakendustes. Kahe kõnealuse variandi erinevus lõi aga rohkem välja interaktiivsuse koha pealt, kus Nuxt'i kasutava veebirakenduse leht võttis 13,6% kauem aega, et interaktiivsus saavutada. Vähene erinevus esmase laadimise ning interaktiivsuse koha pealt oli express-vue moodulit kasutavas veebirakenduses, kus vastavate näitajate vahe oli vaid 0,11 millisekundit.

Ootuspäraselt võttis lehe laadimine kõige enam aega kliendipoolse Vue rakenduse serveeritud lehel, kus nii esmane laadimine ja interaktiivsus mõlemad võtsid aega 3,184 millisekundit. Üldistavalt võib öelda, et see oli ligi poole rohkem kui serveri poolt renderdatud lehe puhul. Aeglaseid näitajad võib põhjendada asjaoluga, et lehe kuvamiseks pidi brauser vajalikud skriptid esmalt alla laadima ning siis ka jooksutama.

Jõudluse analüüsimise tulemusena selgus, et kõige kiirem serveripoolse renderdamise lahendus on vue-server-renderer ning kõige aeglasem express-vue moodul. Võrreldes kliendipoolse renderdamisega on võimalik serveripoolset renderdamist kasutades veebilehe esmast laadimiskiirust ligi poole võrra tõsta.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli tutvuda erinevate Vue serveripoolse renderdamise lahendustega. Töö käigus anti esmalt ülevaade erinevatest veebilehe renderdamise laadidest ning seejärel tutvuti lähemalt serveripoolse renderdamise võimalustega Vue raamistikus.

Töös uuris töö autor lähemalt kolme Vue serveripoolse renderdamise lahendust: vue-server-renderer moodul, express-vue moodul, Nuxt raamistik. Selleks, et saada põhjalik arusaam iga variandi iseloomust, piirangutest ja võimalustest implementeeris autor iga lahenduse eraldi rakenduses. Implementatsiooni käigus sai autor hinnata lahenduste omadusi ning selle läbi neid omavahel võrrelda.

Uurimuse käigus selgus, et valitud variantidest oli kõige lühem õppimise- ning arendamise protsess express-vue moodulit kasutades ning kõige pikem vue-server-renderer moodulit kasutades. Töö kirjutamise hetkel on kõige populaarsem lahendus vue-server-renderer, mida toetab nii NPM statistika kui ka dokumentatsiooni põhjalikkus ning lisamaterjalide kättesaadavus. Ühe olulise faktorina hindas autor lahenduse implementeerimise mugavust eksisteerivas veebirakenduses ning leidis, et kõige mugavam on express-vue mooduli rakendamine, kuid piisavalt paindlik on ka vue-server-renderer kasutamine. Kõige kiirem serveripoolse renderdamise lahendus oli vue-server-renderer moodulit kasutades ning kõige aeglasem express-vue moodulit kasutades.

Kasutatud kirjandus

[1] S. Sonnes, "What Does It Mean To "Render" A Webpage?," [WWW]. https://www.pathinteractive.com/blog/design-development/rendering-a-webpage-with-google-webmaster-tools/#.WwL0c1OFPfZ . (18.05.2018)

[2] T. point, "What Is Javascript?," [WWW]. https://www.tutorialspoint.com/javascript/javascript_overview.htm . (18.05.2018)

[3] S. Brehm, "Isomorphic JavaScript: The Future of Web Apps," [WWW]. https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc. (18.05.2018)

[4] P. Skólski, " Single-page application vs. multiple-page application," [WWW]. https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. (10.05.2018)

[5] Statista, "Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2017," [WWW]. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices. (12.05.2018)

[6] A. Zerner, "Client-side rendering vs. server.side rendering," [WWW]. https://medium.com/@adamzerner/client-side-rendering-vs-server-side-rendering-a32d2cf3bfcc. (12.05.2018)

[7] J. Vega, "Client-side vs. server-side rendering: why it’s not all black and white," [WWW]. https://medium.freecodecamp.org/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d . (05.05.2018)

[8] M. Brick, "What is search engine crawler?," [WWW]. http://www.brickmarketing.com/define-search-engine-crawler.htm. (12.05.2018)

[9] P. Hund, "SEO vs. React: Web Crawlers are Smarter Than You Think," [WWW]. https://medium.freecodecamp.org/seo-vs-react-is-it-neccessary-to-render-react-pages-in-the-backend-74ce5015c0c9. (12.05.2018)

[10] T. F. Crawler, "Facebook for Developers," [WWW]. https://developers.facebook.com/docs/sharing/webmasters/crawler . (10.05.2018)

[11] G. Developers, "First Meaningful Paint," [WWW]. https://developers.google.com/web/tools/lighthouse/audits/first-meaningful-paint. (10.05.2018)

[12] A. Ktquez, "An overview of Vue.js and the future of the framework," [WWW]. https://codeburst.io/an-overview-of-vue-js-and-the-future-of-the-framework-7830dd9726f0. (10.05.2018)

[13] Vue, "Vue.js," [WWW]. https://vuejs.org/v2/. (22.04.2018)

[14] T. Freed, "What Is Virtual DOM?," [WWW]. https://tonyfreed.blog/what-is-virtual-dom-c0ec6d6a925c. (10.05.2018)

[15] S. Deyne, "Dealing with templates in Vue.js 2.0," [WWW]. https://sebastiandedeyne.com/dealing-with-templates-in-vue-20. (10.05.2018)

[16] A. Grigoryan, "Dealing with templates in Vue.js 2.0," [WWW]. https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8. (10.05.2018)

[17] Webpack, "Webpack.js," [WWW]. https://webpack.js.org/concepts/. (05.05.2018)

[18] Express, "Express.js," [WWW]. https://expressjs.com/. (05.05.2018)

[19] D. Google, "Lighthouse," [WWW]. https://developers.google.com/web/tools/lighthouse/. (10.05.2018)

[20] GitHub, "Express-vue," [WWW]. https://github.com/express-vue/express-vue. (10.05.2018)

[21] D. Cherubini, "Vue.js and Express can they live together?," [WWW]. https://cherubini.casa/vue-js-and-express-can-they-live-together-c17d48bc41b7. (10.05.2018)

[22] G. Developers, "Introduction to Structured Data," [WWW]. https://developers.google.com/search/docs/guides/intro-structured-data. (10.05.2018)

[23] Nuxt, "Nuxt.js," [WWW]. Kättesaadav: https://nuxtjs.org/. (10.05.2018)

[24] P. Vorbach, "NPM," [WWW]. https://npm-stat.com/charts.html?package=express-vue&package=vue-server-renderer&package=nuxt&from=2018-04-01&to=2018-04-30. (10.05.2018)

[25] G. Developers, "First Interactive," [WWW]. https://developers.google.com/web/tools/lighthouse/audits/first-interactive. (10.05.2018)