TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Joosep Noot 164374IAIB

VEEBILEHE LAADIMISKIIRUSE ANALÜÜS KLIENTRAKENDUSE RAAMISTIKU UUENDAMISEL JA LEVINUMATE OPTIMEERIMISMEETODITE RAKENDAMISEL E-KOOLIKOTI NÄITEL

Bakalaureusetöö

Juhendaja: Roger Kerse

Tehnikateaduste magister

Tallinn 2020

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: Joosep Noot

25.05.2020

Annotatsioon

Käesoleva töö eesmärgiks oli parandada veebirakenduse E-koolikott sisulehe laadimiskiirust läbi klientrakenduse raamistiku osalise uuendamise ning optimeerimismeetmete rakendamise.

Töös analüüsiti populaarseimaid klientrakenduse raamistikke, arvestades nii raamistiku jõudlust kui ka olemasolevast rakendusest tingitud nõudmisi. Töö tulemusena valmis hübriidrakendus, mille komponendid on migreeritud valitud raamistikule. Samuti on rakendatud levinumaid optimeerimismeetmeid.

Töö käigus valminud hübriidrakendusel teostatakse veebilehe jõudluse mõõtmisi, mille põhjal analüüsitakse erinevate töös valminud arenduste mõju veebilehe laadimiskiirusele.

Lõputöö tulemus on hübriidrakendus, mille saab võtta aluseks edaspidistele migreerimistöödele. Lisaks on töö tulemusena valminud analüüs, milles on kirjeldatud rakendusele negatiivset ja positiivset mõju avaldanud arendused ning optimeerimisvõimalused tulevikuks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 6 peatükki, 11 joonist.

Abstract

Analysis of web page loading speed when updating front-end framework and applying common optimization methods on the example of E-koolikott

The goal of this thesis is to analyse web page loading speed by creating hybrid web application and applying common optimization methods.

In this thesis, the author gives a brief overview of the existing E-koolikott application and describes why updating the framework is important in the current situation. The author analyses three popular frameworks in terms of their performance and compatibility with the existing application. The characteristic parts of the website are then migrated to the chosen framework and optimization techniques are applied.

Performance measurements are carried out on the hybrid application with an online tool to analyse the impact on different development parts to web page speed.

The result of the thesis is a hybrid application that can be used as a basis of future migration work. In addition, the analysis of web page performance is created, which describes the developments that had a negative or positive impact on web page speed. The analysis also points out optimization opportunities for future development.

The thesis is in Estonian and contains 31 pages of text, 6 chapters, 11 figures.

Lühendite ja mõistete sõnastik

AOT	*Ahead-of-time,* optimeeritud meetod programmeerimiskeele kompileerimiseks
CLI	*Command-line interface*
CSR	*Client Side Rendering*, meetod, mille puhul veebilehe sisu pannakse kokku kasutaja brauseris
CSS	*Cascading Style Sheets*, veebilehe kujundamiseks kasutatav märgistuskeel
DOM	*Document Object Model*, dokumendi objektimudel
HTML	*HyperText Markup Language*, veebilehe struktureerimiseks kasutatav märgistuskeel
HTTPS	*HyperText Transfer Protocol Secure*, turvaline hüperteksti edastusprotokoll
Hübriidrakendus	Rakendus, mis on üles ehitatud toetudes kahele või rohkemale erinevale raamistikule
JSX	*JavaScript XML*, JavaScripti süntaktiline laiendus
LTS	*Long-term Support*, tehnoloogia pikaajaline toetatus
MVC	*Model-View-Controller*
SEF	*Search Engine Friendly*
SEO	*Search Engine Optimization*
SPA	*Single-page Application*, üheleherakendus, veebileht, kus vaate sisu muudetakse lehte uuesti laadimata
SSR	*Server Side Rendering*, meetod, mille puhul veebilehe sisu pannakse kokku serveris
Teek	Kogumik eeldefineeritud funktsioonidest, mallidest, moodulitest või klassidest
URL	*Uniform Resource Locator,* veebiaadress

Sisukord

Jooniste loetelu

1 Sissejuhatus

1.1 Taust

Otsingumootorite tulemuste kuvamine ja seeläbi avalike veebilehtede liiklus sõltub lisaks otsitavate võtmesõnade täpsusele suuresti ka otsingumootorite hinnangust antud veebilehele [4]. Üks hinnangu arvutamise sisendidest on veebilehe kasutusmugavus. Eelistatumad on veebilehed, mis suudavad sisu pakkuda võimalikult paljudele kasutajatele ja seda võimalikult efektiivselt. See tähendab, et veebilehe terviklikkus ja funktsionaalsus peab säilima erinevates brauserites ning seadmetes [12]. Oluline roll veebilehe kasutusmugavuses on lehe laadimise kiirusel, mille korral tuleb arvestada alla laetavate ressursside mahtudega ning veebilehe ligipääsetavusega aeglase internetiühenduse korral. Seda kinnitavad mitmetes globaalsetes ettevõtetes koostatud uuringud, mis näitavad, et juba väikesed parandused veebilehe laadimise kiiruses tõstsid tuntavalt otsingumootorite kaudu sissetulevat liiklust ning seeläbi püsivate kasutajate arvu [33]. Lisaks näitavad Google'i poolt mobiilsetel seadmetel teostatud uuringud, et lehe laadimise aja tõusmisel ühelt sekundilt kuuele sekundile, tõusis tõenäosus kasutaja veebilehelt lahkumiseks 106% [2]. Seetõttu on veebilehe laadimiskiirusel määrav roll veebilehe kasutatavuse tõstmiseks ning säilitamiseks.

1.2 Ülesande püstitus

E-koolikott on õppimiseks loodud veebirakendus, mille eesmärk on koondada digitaalsed õppevarad ühte veebikeskkonda. Veebirakenduse *front-end* on üles ehitatud kasutades raamistikku AngularJS versiooni 1.6.4. AngularJS puhul on tegemist Angulari raamistike esimese versiooniga ning praeguseks on väljas mitmeid uusi Angulari versioone, mille juures on muuhulgas parandatud raamistiku jõudlust. Kuna E-koolikotis on endiselt kasutusel AngularJS raamistik ning selle juurde käivad teegid, millest osad on nüüdseks aegunud, siis on veebilehe laadimiskiirus tuntavalt aeglane. Enim kasutatavad sisulehed muutuvad kasutajale täielikult interaktiivseks keskmiselt 8 sekundiga ning mobiiliseadmete puhul isegi 36 sekundiga.

Töö eesmärgiks on parandada E-koolikoti kasutatavaima lehe laadimiskiirust *front-end* raamistiku uuendamise teel. Selleks analüüsitakse töös Angulari, Vue ja Reacti raamistikke ning valitakse migreerimiseks sobivaim, pidades silmas olemasolevat rakendust ning raamistiku jõudlust. Lisaks jõudluse aspektile lähtutakse raamistiku valikul ka veebilehe kasutusmugavuse ning arendusmugavuse nõuetest. Seejärel migreeritakse osa olemasolevast veebirakendusest valitud raamistikule. Peale selle uuritakse ja rakendatakse töö käigus võimalikke raamistikust sõltumatuid optimeerimismeetmeid veebilehe laadimiskiiruse parandamiseks. Arenduse käigus teostatakse veebilehe jõudlusega seotud mõõtmisi, mille põhjal on võimalik analüüsida arendusetappide kasu ning seejuures leida, milline osa arendusest tõi parima tulemuse.

Oodatav tulemus on hübriidrakendus, kus põhiosa on realiseeritud raamistikul AngularJS ning valitud osa rakendusest on täielikult migreeritud valitud raamistikule. Samuti on veebirakenduses realiseeritud levinumad optimeerimise võimalused. Arenduse tulemusena on valminud veebirakendus, mille täieliku interaktiivsuse saavutamiseks kuluvat aega loetakse Google'i poolt kiireks ehk see on alla 5,2 sekundi [29].

2 Raamistiku valik

Käesoleva töö alustamisel oli E-koolikotis kasutatavaks *front-end* raamistikuks AngularJS v1.6.4. Tegemist on Google'i poolt arendatava raamistikuga, mis võimaldab tavapäraseid staatilisi HTML veebilehti täiendada dünaamilise sisuga, säilitades seejuures koodi loetavuse. AngularJS raamistikus vaikimisi sisalduvad tööriistad muudavad dünaamilise veebirakenduse loomise äärmiselt lihtsaks, vajamata seejuures tavapäraseid ajaliselt mahukaid ettevalmistavaid arendusi [36]. See on ka üks põhjuseid, miks AngularJS on olnud pikalt üks populaarsemaid veebiraamistikke.

Dünaamiliste veebilehtede loomise poolest on AngularJS tuntud oma kahesuunalise andmete sidumise poolest, mis lihtsustab oluliselt SPA-de (*Single-page Application)* loomist ja SPA andmete haldamist. Üldjuhul on veebiraamistike puhul kasutusel ühesuunaline andmete sidumine, mis tähendab, et mudeli andmeid on võimalik saata veebirakenduse vaadetesse, kuid kasutaja tegevuse peale mudel ise automaatselt ei muutu. See tähendab, et muutuste tuvastamine ja seeläbi mudeli uuendamine on arendaja vastutus. Kahesuunalise andmete sidumise puhul liigub info aga kahes suunas, mis tähendab, et muutus veebirakenduse vaates uuendab automaatselt mudeli ning vastupidi [8]. Kuna kahesuunalise andmete sidumiseks on vaja pidevat muutuste tuvastamist, siis rakenduse suurenedes kasvab ka muutuste tuvastamiseks vajalike tsüklite arv, mis on üks suurimaid jõudlusele negatiivset mõju avaldavaid aspekte [1].

Arenduste käigus on tehtud küll mõningaid jõudlust parandavaid täiendusi ja parandusi, kuid otseselt jõudluse parandamisele suunatud arendusi AngularJS puhul pole teostatud, mistõttu pole märgatavat jõudluse muutust aastate jooksul toimunud [3]. Peale selle lõppeb kõige uuema versiooni 1.7.9 LTS (*Long-term Support)* 2021. aasta juulis, mis tähendab, et raamistikku enam ametlikult ei toetata ega tehta sellele uuendusi ja parandusi võimalikele vigadele [11]. Seega pole uute projektide alustamisel enam mõistlik AngularJS'i kasutada ning samuti on olemasolevate rakenduste puhul mõistlik üle minna muudele raamistikele.

Esmapilgul tunduks loogiline jätkata uue Angulariga, mis on samuti Google'i edasiarendus AngularJS'ist. Küll aga on Angulari puhul tegu täielikult uuesti kirjutatud raamistikuga ning seetõttu on AngularJS'i migreerimine Angulari uuele versioonile sama keeruline, kui mistahes muule populaarsele raamistikule [5]. Järelikult on uue raamistiku

valikul mõistlik analüüsida populaarsemate raamistike jõudlust, mahtu ning arenduse mugavuseks ja seejuures koodi kvaliteedi parandamiseks pakutavaid võimalusi, pidades silmas olemasolevat rakendust ja töö eesmärki. Töös analüüsitakse kolme raamistikku: Angular, React.js, Vue.js.

2.1 Nõuded raamistikule

Valitav raamistik peab eelkõige võimaldama jõudluse parandamist võrreldes E-koolikotis kasutatava AngularJS raamistikuga. Samas tuleb raamistiku valikul arvestada ka üldist kasutusmugavust, olemasoleva veebilehe olemust ning arenduse ja õppimise keerukust.

2.1.1 Arenduse lihtsus ja jätkusuutlikus

Raamistiku valikul on oluline silmas pidada selle vaikimisi pakutavaid tööriistu, et arendust saaks alustada võimalikult kiiresti ning vähese lisatööga. Seejuures tuleb arvestada, et kõik E-koolikoti rakenduses realiseeritud lahendused oleksid uuel raamistikul taasloodavad kas samade teekidega või siis alternatiivsete lahendustega. Koos funktsionaalsuse täieliku püsimisega, peab säilima ka rakenduse kasutajaliidese ühtne stiil. E-koolikoti visuaalsed elemendid on suuremas osas arendatud Material Design stiiliraamatut kasutades, mistõttu on vajalik, et valitava raamistiku poolt oleks samuti Material Design toetatud.

Oluline aspekt valiku tegemisel on raamistiku poolt antav vabadus arendajale. Üldjuhul peetakse suuremat vabadust raamistiku puhul positiivseks omaduseks, kuid suuremahuliste projektide puhul võib see pigem kahju tuua. Antud projekti puhul oleks mõistlik kasutada raamistikku, mis nõuab ranget struktuuri koodi ülesehitusel, et ennetada varakult võimalikke vigu, ühtlustada koodi stiili, parendada olemasoleva koodi kvaliteeti ning hoida tulevikus kõrget koodi kvaliteeti.

2.1.2 Kasutusmugavuse säilimine

Sõltuvalt projektist on võimalik raamistikku vahetada kas väiksemate osade kaupa järk-järgult või kirjutada kogu kood korraga uuele raamistikule ümber. Antud projektis toimub migreerimine osade kaupa ning seejuures on plaan migreeritud osa arenduse õnnestumisel kohe kasutusse võtta. Selle võimaldamiseks on kaks teed. Üks võimalus on luua eraldiseisev rakendus, kus on realiseeritud valitud komponendid uuel raamistikul ning ülejäänud rakenduse elemendid jooksevad endiselt olemasolevas eraldisesivas

rakenduses. Selleks, et sellise lahenduse puhul mitte skoobist väljuda, tuleks nende elementide, mis antud töö raames migreerimisele ei kuulu, kasutamiseks suunata kasutaja tagasi vana rakenduse peale. Samuti liikudes vanas rakenduses migreerimisele kuuluvasse detailvaatesse, suunatakse kasutaja jällegi uude rakendusse. Iga sellise suunamise juures laetakse kasutaja jaoks veebilehte uuesti otsast peale, kuna tegemist on eraldiseisvate rakendustega. See on kindlasti kasutusmugavusele negatiivset mõju avaldav efekt. Peale selle tekitab selline lahendus üsna palju lisatööd, sest kõigi kirjeldatud rakenduste vaheliste liikumiste juures tuleb edasi anda kasutaja rolli, sisse logimise staatust ja sessiooni andmeid. Rakenduse täielikul migreerimisel muutub see osa koodist aga tarbetuks ning tuleb uuesti eemaldada.

Vastukaaluks eelnevalt kirjeldatud lahendusele tuleb raamistiku valikul jälgida, et uuel raamistikul arendatud komponente oleks võimalik kasutada otse olemasolevas rakenduses. Seega saab komponenthaaval detailvaadet migreerida uuele raamistikule ja selline lahendus hoiaks ära kahe eraldiseisva rakenduse vahel liikumise, sellest tulenevad keerulised ja lisatööd nõudvad arendused ning kasutusmugavuse langemise.

2.1.3 SEO võimalused

Erinevad SEO (*Search Engine Optimization)* lahendused nõuavad üldjuhul üsna palju dünaamilist sisu – vaja on kasutada leheküljele vastavaid metaandmeid, dünaamilisi SEF (*Search Engine Friendly)* URL'e (*Uniform Resource Locator)* ja muid sarnaseid elemente. Seejuures tuleb arvestada seda, et otsingumootorid üldjuhul JavaScripti ei käivita, mistõttu CSR (*Client Side Rendering)* rakenduse puhul ei ole võimalik dünaamilist sisu kasutada. Siinkohal on vajalik uue raamistiku valikul arvestada, et see pakuks ise või võimaldaks kasutada teeke, mis serveerivad dünaamilist sisu otsingumootoritele korrektselt. Veelgi parem variant, mis aitab märgatavalt kaasa ka veebilehe jõudlusele, on raamistiku poolt pakutav SSR (*Server Side Rendering)* võimalus.

2.1.4 Õppimise keerukus

Raamistiku valiku puhul tuleb arvestada, et tuleviku arendusi võivad teostada arendajad, kellel puudub põhjalik kogemus mistahes *front-end* raamistikuga. Seetõttu peab valitav raamistik olema lihtsasti õpitav. Õppimise keerukuse määravad antud juhul valitava raamistiku ametliku dokumentatsiooni põhjalikkus, kogukonna tugi ning raamistikuga töötamiseks vajaminevate tööriistade või tehnoloogiate õppimise vajadus ja keerukus.

2.2 Angular

Angular on Google'i arendatud *front-end* raamistik, mis on kirjutatud TypeScriptis. Üldine Angulari ehitus põhineb moodulitel, HTML (*HyperText Markup Language)* mallidel, teenustel ning komponentidel. Iga rakendus koosneb peamoodulist, mis ühendab kogu ülejäänud rakendust. Moodulid võivad olla näiteks Angulari enda või kolmandate osapoolte teegid või arendaja enda koostatud moodulid, mis täidavad rakenduse mingi alamosa ülesannet. HTML mallid moodustavad rakenduse visuaalse osa, kuhu on Angulari abil võimalik arendada dünaamilist sisu. HTML mallide sisu ja loogika, kuidas dünaamilised elemendid on mallidega seotud, määratakse rakenduse komponendis, mille juurde mall kuulub. Teenuste klassides on kirjeldatud loogika ja andmed, mis on kasutatavad mitmes erinevas vaates või ei ole seotud kindla vaatega [14]. Selline komponendipõhine arhitektuur tähendab, et veebirakenduse vaate saab jaotada väiksemateks komponendipõhisteks osadeks. Seejuures vastutab iga komponent ainult oma osa eest ning komponendid on kasutatavad erinevates vaadetes ja omakorda teiste komponentide sees [15]. Selline koodi ülesehitus tagab hea koodi loetavuse, kindla struktuuri, lihtsama hoolduse ning koodi taaskasutuse võimaluse. Töös analüüsitakse Angular 9 raamistikku.

2.2.1 TypeScript

Üks suurimaid erinevusi võrreldes AngularJS'iga on TypeScripti kasutusele võtmine Angularis. Tegemist on tüübitud JavaScriptiga, mis rakenduse käivitusel kompileeritakse tagasi JavaScriptile võrdväärsele kujule. TypeScripti peamine eesmärk on muuta kood rangemalt struktureerituks ning seeläbi parandada koodi loetavust ja haldamist ning võimalikult varakult tuvastada koodis tekkivaid võimalikke vigu [30]. TypeScripti kasutamisel on ka mõned miinused. Vaatamata sellele, et TypeScript põhineb suures osas JavaScriptile ja on süntaktiliselt JavaScriptiga äärmiselt sarnane, vajab TypeScript ikkagi mingil määral aega õppimiseks. Samuti suureneb vajaliku koodimahu hulk, mistõttu võib mõnevõrra tõusta esialgseks arenduseks kuluv aeg [28].

2.2.2 Jõudlus

Angulari ümberkirjutamisel AngularJS'ilt on arendusmeeskonna poolt silmas peetud AngularJS'i nõrku külgi veebirakenduse jõudluse osas. Angular 2 raamistik oli juba 5 korda kiirem, kui AngularJS versioonid ning alates Angulari 2. versioonist on jõudlust

parandavaid arendusi teostatud veelgi. Uuemates versioonides on oluliselt parandatud kompilaatorite ning HTML'i ja TypeScripti töötlemiseks kasutatavaid tööriiste. Viimases Angulari versioonis on TypeScriptis kirjutatud komponentide brauseri jaoks sobivaks HTML'iks ja Javascriptiks teisendamiseks vaikimisi kasutusel *Ivy renderer*. Tegemist on AOT (*Ahead-of*-time) kompilaatoriga, mis tagab oluliselt kiirema kompileerimise ning optimeerib kompileerimise tulemusena saadud failide ja seeläbi rakenduse suurust [27].

Lisaks on arendatud *Angular Universal* teenus, mis võimaldab kasutada rakenduse serveripoolset renderdamist tavapärase brauseripoolse renderdamise asemel. Eelkõige on see lahendus kasulik veebilehe kiiruse tõstmisel, kuid seejuures võimaldab SSR parandada ka SEO nõudeid dünaamilistes rakendustes, kus üldjuhul SEO jaoks vajaliku dünaamilise sisu kasutamiseks on vaja abistavaid tööriistu kasutada [27].

2.2.3 Kogukonna tugi ja õppimise keerukus

Angulari dokumentatsioon on üldiselt väga põhjalik ning sisaldab muuhulgas juhendit, kuidas migreerida AngularJS rakendus uuele Angulari raamistikule, mis on antud projekti mõistes väga kasulik.

StackOverflow andmetel, mis on arendajate üks enimkasutatavaid keskkondi programmeerimisalastele probleemidele lahenduse saamiseks, oli töö tegemise ajal Angular enim otsitud raamistik võrdluses olevate raamistike seas. Raamistik on olnud kasutamiseks võimalik alates 2016. aasta septembrist ning praeguseks leidub Angulari märksõnaga kokku 216 080 küsimust [24].

Angulari õppimine võib olla mõnevõrra keerulisem, kui puudub täielik kokkupuude TypeScriptiga, mis on Angularis kasutatav programmeerimiskeel. Küll aga on Angularis sisalduv tööriistade hulk täielikum, kui teistel vaatluse all olevatel raamistikel, mistõttu on projektiga alustamine kiirem ning mugavam [22].

2.2.4 Migreerimine

Angulari eeliseks on ametliku dokumentatsiooni hulka kuuluv juhend AngularJS'ilt migreerimiseks ning raamistikus sisalduvad vahendid selle võimaldamiseks. Seejuures on võimaldatud antud töö jaoks oluline osa ehk hübriidrakenduse loomine, mis tähendab, et Angulari komponente saab kasutada juba olemasolevas AngularJS rakenduses ja seega migreerimist teostada järk-järguliselt.

2.3 React

Facebooki poolt arendatud Reacti peetakse üldjuhul pigem JavaScripti teegiks, mitte raamistikuks. Seda võib põhjendada asjaoluga, et raamistikke peetakse töövahenditeks, mis muuhulgas määravad mingil moel veebirakenduse ülesehituse ja struktuuri. React seejuures ei nõua rakenduses mingit kindlat struktuuri, mis ühest küljest tagab kiire ja mugava arenduse, kuid samas võib hoolikalt läbi mõtlemata arenduse puhul rakenduse kood muutuda kiiresti väga segaseks [16]. Kui Angulari ja AngularJS raamistike puhul on juba raamistiku tasandil määratud, kuidas kood suures pildis üles ehitada, siis Reacti puhul on koodi struktureerimine ja kontrolli all hoidmine täielikult arendajate vastutus.

React on tugevalt suunatud MVC (*Model-View-Controller)* mudeli vaate osale, mistõttu on Reactis vaikimisi pakutud ainult vahendid DOM (*Document Object Model*) manipuleerimiseks [22]. Mitmed veebirakenduse elementaarsed elemendid tuleb seega eraldi rakendusse lisada, mis annab jällegi arendajatele rohkem vabadust, kuid seejuures nõuab otsuse tegemine analüüsimist ning mingil määral lisatööd.

2.3.1 JSX

Reacti komponentides kasutatakse JavaScripti täiendust JSX (*JavaScript XML*), mille põhiline tunnusjoon on HTML mallide kasutamise võimalus JavaScripti koodis. Sarnaselt TypeScriptiga kompileeritakse ka JSX kood rakenduse ehitamisel brauserile vastuvõetavaks HTML ja Javascripti koodiks. JSX kasutamine Reacti rakenduses ei ole ilmtingimata kohustuslik, kuid üldjuhul on see kasulik täiendus kasvõi visuaalse koodi puhtuse või vigade tuvastamise osas [13].

2.3.2 Jõudlus

Dünaamiliste veebirakenduste üks olulistest jõudlust mõjutavatest aspektidest on DOM manipuleerimine. Üldjuhul toimib see nii, et vaates olevate komponentide kohta hoitakse puu kujul komponentide kohta käivat infot. Kui mingis DOM elemendis toimub muutus, siis vaate uuendamiseks tuleb see uuesti ehitada, mis on jõudluse mõistes võrdlemisi kulukas tegevus. React kasutab selle ärahoidmiseks virtuaalse DOM tehnoloogiat, mis tähendab, et täiendavalt tavapärasele mälus hoitavale DOM struktuurile, leidub sellest veel koopia, kuhu kantakse esialgu vaates tuvastatud muudatused. Teatud aja tagant toimub nende struktuuride võrdlemine, mille põhjal tehakse muudatused ainult vajalikes veebilehe osades, mitte ei ehitata tervet veebilehte uuesti [9].

Ka Reacti rakenduste puhul on võimaldatud SSR kasutamine, kuid seejuures esineb väike erinevus tavapärase SSR'iga võrreldes. Nimelt Reacti puhul saab SSR'i kasutada ainult esialgse lehe laadimise puhul ning ülejäänud lehed renderdatakse endiselt brauseris [17]. Kui aga arvestada, et SSR kasulikkus ilmnebki eelkõige lehe esialgsel laadimisel nii jõudluse kui ka SEO tasemel, siis on selline lahendus igati sobiv.

2.3.3 Kogukonna tugi ja õppimise keerukus

Reacti dokumentatsioon on samuti väga põhjalik, kuid võrreldes Angulari dokumentatsiooniga on antud töö puhul puuduseks ametliku migreerimise juhendi puudumine.

StackOverflow's olevate Reacti-teemaliste küsimuste arvu põhjal on kogukonna tugi põhimõtteliselt sama suur kui Angulari puhul. Töö tegemise ajal oli Reacti märksõnaga seotud 213 404 küsimust [25]. Küll aga tuleb seejuures arvestada, et React on eksisteerinud juba aastast 2013 ehk mõnevõrra kauem, kui Angular. Reacti kodulehel on täiendavalt mitmeid viiteid ka teistele Reactiga seotud foorumitele. Kuna React on Facebooki enda arendustes kasutusel, siis ei ole põhjust karta arendusmeeskonna toe kadumist.

Õppimisprotsessi peetakse üldiselt sama keeruliseks kui Angulari raamistiku puhul [22]. Õppimist võivad mõnevõrra raskendada JSX kasutamine, vajalike lisatööriistade rakendamine ning kindla struktuuri puudumine.

2.3.4 Migreerimine

Sarnaselt Angularile on võimalik Reacti komponente kasutada olemasolevas AngularJS projektis ehk seeläbi migreerida rakendus Reactile järk-järgult. Teoreetiliselt paraneb selle käigus ka iga komponendi Reactile migreerimisel veebirakenduse jõudlus, kuna vähenevad kahesuunalised andmete sidumised ning migreeritud komponendi puhul võetakse juba kasutusse Reacti poolt pakutav virtuaalse DOM'i tehnoloogia. Sarnaselt Angularis pakutavatele teekidele, on Reacti puhul loodud tööriist, mis lihtsustab komponenthaaval migreerimist AngularJS raamistikult [18].

2.4 Vue

Vue puhul oli algselt tegu ühe arendaja poolt arendatud raamistikuga, kuid selle populaarsuse kasvamisel tekkis Vue arenduse taha suurem arendusmeeskond. Ehituselt on Vue raamistik väga sarnane Reactiga, mille fookus on samuti peamiselt MVC vaate kihil. Vaatamata raamistiku väikesele mahule on Vue poolt pakutavaid veebirakenduse elemente mõnevõrra rohkem kui Reacti puhul. Vued tuntakse üldiselt kui raamistikku, mis on ühendanud Angulari ja Reacti head omadused ning seejuures proovinud vältida nendega seotud kitsaskohti [22].

2.4.1 Jõudlus

Nagu ka Reactis on Vue puhul DOM manipuleerimiseks kasutusel virtuaalse DOM'i tehnoloogia koos mõningate jõudlust veelgi parandavate täiendustega [6]. Lisaks on Vue rakendused mahult mõnevõrra väiksemad, kui Reacti ja Angulari rakendused, mis on samuti jõudlust mõjutav aspekt ning seda eriti mobiiliseadmete osas [6]. Nagu Angularis ja Reactis, on Vue rakenduste puhul võimalik seadistada SSR toetamaks kiiret veebilehe laadimist ning SEO nõuete täitmist [23].

2.4.2 Kogukonna tugi ja õppimise keerukas

Vue ametlik dokumentatsioon on üks põhjalikumaid, kuid kogukonna poolest on Vue praegu veel kõige väiksema toega. Viimasel ajal on Vue olnud võrreldes teiste vaatluse all olevate raamistikega populaarsuse osas oluliselt kiiremini kasvav [35]. Vaatamata sellele, on Vuega seotud teemasid ja materjale internetis märgatavalt vähem kui Reacti ja Angulari puhul. Töö tegemise ajal oli Vue märksõnaga teemasid StackOverflow's ainult 56 444 [26]. Seetõttu võib esineda mitmeid probleeme, millele pole Vue kasutamisel lahendust veel pakutud. See on üks suurimaid ohumärke, miks peab tõsiselt kaaluma kas suuremahulise projekti puhul on Vue kasutusele võtmine otstarbekas.

Küll aga peetakse Vued Angularist ja Reactist oluliselt lihtsamini õpitavaks. Eelkõige tagab lihtsa õppimisprotsessi rakenduse koodi lihtsus – kood põhineb tavalisel HTML'il ning JavaScriptil [22]. Seetõttu saab hakata raamistikku kasutama, ilma et peaks juurde õppima TypeScripti või JSX'i.

2.4.3 Migreerimine

Vue järk-järguliseks migreerimiseks on loodud tööriist ngVue, mis on inspireeritud samasugusest tööriistast Reacti migreerimiseks – ngReact. Nimetatud tööriist võimaldab uusi Vue komponente jooksutada olemasolevas AngularJS rakenduses [19]. Seega saab olemasoleva rakenduse komponente sammhaaval välja vahetada ning nagu Reactigi puhul, peaks teoreetiliselt olema jõudluse paranemist näha juba komponentide välja vahetamise käigus.

2.5 Valitud raamistik

Võttes arvesse peatükis 2.1 kirjeldatud E-koolikoti rakendusest tulenevaid nõudeid valitavale raamistikule ning valikus olevate raamistike tugevaid ja nõrkasid omadusi, on autor otsustanud veebirakenduse migreerida raamistikule Angular 9.

Alates Angular 2 versioonist on hakatud Angulari arendusel oluliselt rohkem tähelepanu pöörama raamistiku jõudluse tõstmisele. Praeguseks ei ole Angulari, Vue ja Reacti jõudluste erinevused enam nii märgatavad ning Angulari meeskonna fookus on jätkuvalt raamistikul loodava rakenduse suuruse vähendamisel, mis praegu on raamistiku üks negatiivsemaid omadusi [22]. Kuna kõigi valikus olnud raamistike poolt on toetatud SSR, siis jõudluse osas märgatavate erinevuste puudumise tõttu, ei osutunud jõudlus antud juhul määravaks küljeks.

Suurim erinevus, mis osutus raamistiku valikul kaaluvaks ja eristab Angulari selgelt teistest vaatluse all olevatest raamistikest, on Angulariga kaasnev mugavus arendusel. Raamistiku pakis on vaikimisi olemas kõik veebirakenduseks vajalikud elemendid ja tööriistad. Seejuures suunavad Angulari ülesehitus ja programmeerimiskeelena kasutusel olev TypeScript kasutama häid koodi kirjutamise tavasid ning hoidma koodi kvaliteeti. TypeScript on küll toetatud kõigi kolme raamistiku poolt, kuid Angulari puhul on see juba raamistiku pakis sisaldatud. Teiste raamistike puhul nõuaks see mõningaid lisaseadistusi. Ühtlasi on olemas Angulari tiimi poolt arendatud tööriistad migreerimise hõlbustamiseks ning Angular/AngularJS hübriidrakenduse loomiseks, mis annab võimaluse veebirakendust järk-järgult migreerida. Samuti leidub juhend E-koolikotis laialdaselt kasutusel oleva Material Design teegi migreerimiseks AngularJS'iga ühilduvalt versioonilt Angulariga ühilduvale versioonile.

Stackoverflow küsimuste arvu osas on Angular ja React üsna võrdsed ning edastavad Vued umbes neljakordselt. Angulari ja Reacti võrdluses tuleb aga arvestada, et React on eksisteerinud kolm aastat kauem, kui Angular ehk Reactil on pikema aja jooksul kogunenud Angulariga võrdselt Stackoverflow's olevaid küsimusi. Seega on Angulari puhul kõige tõenäolisem leida vastuseid arenduse käigus tekkivatele probleemidele. Lisaks on Google'i, Angulari arendaja poolt tagatud LTS ning Angulari dokumentatsioon on väga detailne ning ametlikke juhendeid leidub isegi hübriidrakenduse loomiseks koos täiendavate soovitustega selle käigus jõudluse tõstmiseks.

3 Arenduskäik

Töö teostamiseks vajalik arendus on jaotatud kolmeks osaks. Esimene osa on Angulari komponentide kasutuselevõtmine, mille käigus kirjutatakse valitud rakenduse osad ümber Angulari koodiks ning tehakse vajalikud seadistused uue koodi käivitamiseks olemasolevas rakenduses.

Teine osa arendusest on Angulariga valminud rakenduse jõudluse analüüsimine ning võimalike optimeerimismeetodite leidmine ja nende rakendamine.

Rakenduse jõudlust saab mõningate vahenditega analüüsida juba lokaalses rakenduses, kuid selleks, et tulemused kajastaksid reaalse rakenduse näitajaid, on vaja seadistada veebiserver ning rakendus sellel käivitada. Arenduse kolmandas osas paigaldatakse server ning seadistatakse automaatne rakenduse ehitusprotsess nimetatud serverile.

Raamistike migreerimise projekti aluseks on võetud E-koolikoti projekti LIVE haru värskeim seis antud töö alustamise hetkel. Migreerimisele lähevad komponendid, mis on töö fookuses olevale veebilehele omased elemendid. Lisas 1 on visuaalselt kirjeldatud rakenduse struktuur, kus migreerimisele minevad komponendid on esile toodud paksu äärisega ning Angulari märksõnaga.

3.1 Angularile migreerimine

Arenduse planeerimisel tuli silmas pidada, et arendatud rakendus oleks võimalik arenduse valmides kohe reaalselt kasutusse võtta. Seetõttu on migreerimisel kasutatud hübriidrakenduse lähenemist, et rakenduse kasutusele võtmisel säiliks kasutusmugavus.

AngularJS raamistiku migreerimisel Angular 9-le tuli esmalt teha mõned ettevalmistavad seadistused ning muudatused koodis. Ettevalmistuste ja üldise migreerimise osas on enamjaolt võetud aluseks vastav Angulari meeskonna koostatud juhend.

3.1.1 TypeScript ja Webpack

Üldjuhul kasutatakse Angulari projektides programmeerimiskeelena TypeScripti. TypeScripti kasutusele võtmine Angulari projektis pole ilmtingimata vajalik, kuid kuna käesolevas projektis jälgiti raamistiku poolt pakutavaid võimalusi koodi kvaliteedi parandamiseks, siis antud juhul on mõistlik seda võimalust kasutada. TypeScripti

JavaScriptiks kompileerimiseks otsustas töö autor kasutada Webpacki eelkõige selle seadistuse kiiruse ja lihtsuse tõttu ning samuti on Webpacki kaudu võimalik mugavalt seadistada erinevaid optimeerimist võimaldavaid tööriistu.

Kuna TypeScript kompileeritakse hiljem Webpacki poolt ümber JavaScriptiks, siis TypeScripti kasutusele võtmiseks piisaks lihtsalt migreerimisele minevate JavaScripti failide laiendite muutmisest [30]. Näiteks olemasoleva faili *material.js* TypeScripti failiks muutmiseks tuleb see fail ümber nimetada *material.ts*. Lihtsalt failide ümbernimetamine ei anna aga mingisugust väärtust ja seetõttu on mõistlik TypeScripti kasutusele võtmise etapis lihtsustada eesolevat migreerimise tööd ja viia kõik migreerimisele minevad failid ühesugusele ehk komponendi kujule.

TypeScriptile migreeritud komponente ei laeta rakendusse edaspidi enam üksikute importidena *index.html* failis, vaid läbi Webpacki, kus need kompileeritakse JavaScriptiks. Kompileerimise tulemuseks on üks JavaScripti fail, mis sisaldab migreeritud komponente kompileeritud kujul ning see fail laetakse rakendusse *index.html* kaudu.

3.1.2 Hübriidrakendus

AngularJS ja Angulari hübriidrakenduse loomiseks on ennekõike vaja paigaldada Angulari teegid. Antud töös on need paigaldatud, kasutades *npm'*i. Pärast Angulari teekide paigaldamist luuakse Angulari põhimoodul, mille kaudu seotakse Angulari komponendid ja teenused AngularJS rakendusega. Samuti tuleb rakenduses kasutatavad Angulari moodulid registreerida peamoodulis *imports* punkti all [32].

Angulari põhimooduli ühendamiseks AngularJS rakendusega on võimalik kasutada Angulari poolt pakutavat moodulit *UpgradeModule* või *downgradeModule*. Üldiselt seisneb nende kahe mooduli erinevus selles, et kasutades *UpgradeModule*'it jookseb rakendus Angulari raamistikul ning migreerimata AngularJS komponendid muudetakse *UpgradeModule*'i abiga Angulari rakendusele vastuvõetavaks. Kasutades *downgradeModule*'it on loogika teistpidine – rakendus jookseb AngularJS raamistikul ning Angularile migreeritud komponendid muudetakse AngularJS rakendusele kasutuskõlblikuks. Valiku tegemine kahe mooduli osas mõjutab ka rakenduse jõudlust. *UpgradeModule*'i kasutamise puhul on AngularJS ja Angulari rakenduse osad tihedamini seotud ning info muutuste kohta AngularJS komponentides saadetakse niisamuti

Angulari rakendusse, mis üldjuhul ei ole tegelikult vajalik. *DowngradeModule*'i puhul hoitakse aga muutuste tuvastamine kahe raamistiku vahel eraldiseisvatena ning see lahendus koormab rakendust märgatavalt vähem ja tagab parema jõudluse. Sellest tulenevalt on antud töös otsustatud kasutada *downgradeModule*'it [31].

3.1.3 Angulari komponendid

Angulari komponendid on suures osas sarnase ülesehitusega nagu ettevalmistuse tulemusena saadud AngularJS komponendid. Komponent tuleb märkida vastava Angulari annotatsiooniga, mille sisse lisatakse komponendi HTML mall või viide sellele ja komponendi *selector*, millega määratakse, mis nimetusega komponenti kasutada saab. Sellele järgneb komponendi klass, kus on realiseeritud komponendi juurde kuuluv funktsionaalsus. AngularJS komponendi *bindings* osa tuleb Angulari komponendi puhul märkida annotatsiooniga *@Input* ning komponendis kasutatavad teenused sisestatakse komponendi klassi konstruktoris, märkides need annotatsiooniga *@Inject* [32].

Komponendi registreerimiseks rakendusse tuleb see kanda Angulari peamoodulisse *declarations* ja *entryComponents* alla. Samuti tuleb komponent lisada AngularJS moodulisse, kasutades Angulari *downgradeComponent* funktsiooni, mis muudab komponendi AngularJS rakendusele vastuvõetavaks [32].

Võimalusel migreeritakse ka Angulari komponentides kasutatavad AngularJS teenused Angulari raamistikule. Migreerimise protsess toimib samal moel, kuid on selle võrra lihtsam, et migreeritud teenuseid ei tule eraldi AngularJS moodulisse registreerida. Registreerida tuleb nimetatud teenused ainult Angulari peamooduli *providers* punkti alla [32].

3.2 Rakenduse serveril käivitamine

Eelkõige mõõtmiste teostamiseks oli vajadus rakendus reaalselt käivitada veebiserveril ning mõõtmisteks kasutatavate tööriistade rakendusele ligipääsu võimaldamiseks seadistada HTTPS (*HyperText Transfer Protocol Secure*) sertifikaadid. Peale selle on serveripoolsete seadistuste abil võimalik rakendada mõningaid optimeerimise meetmeid.

Serverina on kasutusel Apache 2, mis on olemasolev server, kus eelnevalt eksisteeris juba olemasoleva veebirakenduse testkeskkond. Antud töös valminud rakenduse jaoks tuli

luua rakenduse alamdomeenile kuuluv *VirtualHost* vastava seadistusega. Olemasoleva ja loodud *VirtualHosti* seadistuste erinevus seisneb suures osas rakenduseks vajalike failide asukohas. Ühtlasi tuli uuele rakendusele tellida HTTPS sertifikaadid ja seadistada need serveris.

Serveris olevate failide automaatseks uuendamiseks rakenduse koodi muutuste korral seadistas töö autor ka Jenkinsi töö. Töö seisneb selles, et seadistatud arenduse haru muutumise korral versioonihaldusprogrammis, käivitab Jenkins vajalikud käsklused haru põhjal rakenduse failide komplekteerimiseks ning serverisse saatmiseks, seal failide lahti pakkimiseks ja seejärel rakenduse käivitamiseks.

3.3 Raamistikust sõltumatud optimeerimise meetodid

Mitmed optimeerimise meetodid on juba vaikimisi Webpacki poolt aktiveeritud, kui Webpacki kaudu rakenduse ehitamine käivitatakse *production* meetodil. Enamus nendest on algoritmilised meetodid, mis otsivad koodi osade vahel seoseid ja ehitavad selle põhjal koodi failid üles sellisel kujul, et tulemus oleks mahu poolest võimalikult väike ja samal ajal jõudluse osas võimalikult efektiivne. Täiendavalt tehakse Webpacki poolt vaikimisi ära koodi minimeerimine ehk lõpptulemuse koodis on muutujate ja funktsioonide nimed lühendatud ning kõik ebaoluline – tühikud, tühjad read, kommentaarid, reavahetused - on koodist eemaldatud [37].

Lisaks JavaScripti minimeerimisele, on antud töös Webpackis seadistatud CSS (*Cascading Style Sheets*) failide lugemine ja sealjuures minimeerimine. Selle toimimiseks oli vajalik ka CSS failides kasutatavate pildifailide ja fontide lisamine Webpacki seadistusse.

3.3.1 Ainult vajalike funktsioonide importimine

Lõpliku koodipaki suurust aitab märgatavalt vähendada teekidest ainult kasutatavate funktsioonide importimine kogu teegi importimise asemel. Näiteks varasemalt oli koodis kasutatud Lodashi teegi funktsiooni *isEqual()* järgnevalt:

```
import * as _ from 'lodash';
...
if _.isEqual(a, b)
...
```

Joonis 1. Näide Lodashi teegi kasutamisest enne optimeerimist.

Sellise lahendusega tuli koodipaki koostamisel kaasa panna kogu teek ehkki kasutatud oli vaid ühte funktsiooni. Koodipaki mahu vähendamiseks tuleks nimetatud funktsiooni kasutada nii, et pakki läheks ainult kasutatav funktsioon kaasa:

```
import isEqual from 'lodash';
...
If isEqual(a, b)
...
```

Joonis 2. Näide Lodashi teegi kasutamisest optimaalsel kujul.

Mõndade teekide puhul ei ole sellisel kujul mahu vähendamine arendaja enda poolt võimalik. Antud töös oli kasutusel kuupäevade töötlemise teek Moment.js, kus oli vaikimisi funktsioonidega kaasas suures mahus ebaolulist koodi. Selle koodi eemaldamiseks tuli seadistada Webpacki tööriist, mis oskas kirjeldatud osa koodist kõrvale jätta. Antud näite korral oli Moment.js teegi alla kuuluv maht 195 kB ning pärast tööriista kasutamist oli maht vähenenud 47 kB-le.

3.3.2 Preload

Kuna lehe laadimisel ei laeta kõiki faile korraga, siis võivad tekkida olukorrad, kus veebilehe jaoks kriitilised failid ootavad teiste laetavate failide taga, mis pole nii kriitilise tähtsusega. See võib olla üheks põhjuseks veebilehe aeglasel laadimisel. Üldjuhul on kirjeldatud failid omakorda kasutusel teistes failides, mille laadimise lõppemiseks on vajalik kirjeldatud failide laadimine. Kui on teada, millised need failid on, siis saab neile lisada *preload* atribuudi, mis annab veebibrauseri jaoks neile laadimisel kõrgema

prioriteedi. See tagab olukorra, kus need ressursid on selleks hetkeks juba laetud, kui neid muudes failides vajatakse [21].

3.3.3 Ajutine font

Fonti failide suuruse tõttu võib tekkida olukord, kus veebilehel ei kuvata pikka aega teksti, sest fonti laadimine veel käib. Et veebilehel kuvataks tekst võimalikult kiiresti, siis on võimalik laadimise ajaks määrata veebilehe fontiks süsteemi vaikimisi seadistatud font. Selle lahenduse abil näeb kasutaja veebilehel teksti kohe, kui vastav sisu on laetud ning ei pea lisaks sisu laadimisele fonti laadimist ootama [10].

3.3.4 Compression

Veebilehe kiireks laadimiseks on oluline hoida alla laetavate failide maht võimalikult madalana – seda eriti mobiilide puhul, kuna interneti kiirus mobiilis on tavaliselt oluliselt aeglasem kui arvutis ning seetõttu võib veebilehe laadimine mobiilis olla märgatavalt aeglasem. Failide mahtu aitab olulisel määral vähendada *compressioni* ehk failide kokkusurumise aktiveerimine serveri poolel. Kui failide kokkusurumine on seadistatud, siis klientrakenduse poolt küsitavad failid surutakse serveri poolel enne välja saatmist kokku ning veebibrauser oskab need uuesti lahti pakkida. Kirjeldatud meetod aitab failide mahtu vähendada tavaliselt mitmekordselt.

3.3.5 Kasutatava tähestiku määramine

Kasutatava tähestiku määramine on lihtne tegevus, kuid mõju jõudlusele on üsna suur. Tähestiku määramiseks tuleb serveris lisada seadistus, kus iga päringu vastusega antakse kaasa info, mis tähestikku on failides kasutatud. Kui see info puudub, siis jääb veebibrauseri ülesandeks see ise kindlaks teha. Selle kindlaks tegemine võtab aega ning samal ajal peab veebilehtede sisu laadimine ning JavaScripti käivitamine ootama [20].

4 Jõudluse mõõtmine

Arenduse käigus valminud veebirakenduse jõudluse analüüsimiseks teostatakse valitud veebilehtede põhjal mõõtmisi, kus jälgitakse antud töö jaoks olulisi mõõdikuid. Selles peatükis kirjeldatakse mõõtmiste läbiviimist ning analüüsitakse tulemusi.

4.1 Mõõdetavad objektid

Mõõtmised teostatakse E-koolikoti sisulehel, kus toimub komponentide migreerimine. Tegemist on õppematerjali lehega, millel saab olla mitut tüüpi sisu koos õppematerjali kirjeldusega või muu materjali juurde kuuluva lisainfoga teksti kujul. Materjali sisu võimalikud tüübid on:

- Pildifail
- Helifail
- Videofail
- YouTube video
- SoundCloud helifail
- PDF fail
- Slideshare slaidid
- Iframe
- Link välisele materjalile – URL, mis ei vasta eelpool välja toodud tüüpidele, kuvatakse õppematerjalis tavalise URLina.

Kuna mõned tüübid on oma oleku ja kooditasemel realiseerimise poolest üsna sarnased, siis antud töös analüüsitakse mõõtmistulemusi pildifaili, YouTube video, PDF faili, Iframe ja lingiga õppematerjali puhul. Mõõtmised tehakse nii töölaua rakenduse kui ka mobiiliversiooni põhjal.

4.2 Mõõtmiste käik

Mõõtmised teostatakse valitud lehe tüüpidega kuues etapis ja on sõltuvuses töös tehtud arendusetappide järjekorraga, mis tähendab et igas järgnevas etapis sisalduvad ka eelmises arendusetapis teostatud muudatused. Esimesed mõõtmised tehakse töö esialgse seisu põhjal. Töö esialgne rakendus on koostatud töö alustamise ajal E-koolikoti LIVE-

keskkonna haru värskeima seisu põhjal. Teine mõõtmine tehakse pärast rakenduse valitud komponentide TypeScriptile migreerimist ja selle võimaldamiseks Webpacki kasutusele võtmist. Kolmas mõõtmine sooritatakse Angulari kasutusele võtmise ning valitud komponentide Angulari raamistikule migreerimise järel. Mõõtmiste neljas etapp sooritatakse pärast serveripoolsete optimeerimiste seadistamist. Viies mõõtmine tehakse pärast raamistikust sõltumatute *front-endi* poolsete optimeerimisvõimaluste rakendamist. Viimane samm mõõtmistes teostatakse pärast valitud komponentides kasutusel olevate teenuste migreerimist Angulari raamistikule.

Iga etapi mõõtmisi sooritatakse kõikide valitud veebilehe tüüpidega kolm korda nii brauseris kui mobiiliseadmes ning tulemuste analüüsis kasutatakse mõõtmistulemuste keskmist.

4.3 Mõõdetavad parameetrid ja kasutatavad tööriistad

Veebilehe jõudluse mõõtmisel on valdav enamus mõõdikuid seotud kasutaja brauseris tehtava tööga. Üksikud mõõdikud on seotud ka serveri poolel tehtava tööga, kuid antud töös neile ei keskenduta.

Kõige olulisem mõõdik antud töös on Time to Interactive (TTI). TTI tulemus näitab aega millisekundites, mis kulub veebilehe täielikult interaktiivseks ehk kasutajale kogu funktsionaalsusega kasutuskõlblikuks muutumiseni. See on oluline näitaja seetõttu, et lihtsam on veebilehe optimeerimisel keskenduda nähtava sisu kiirele kuvamisele, mistõttu võib kasutajale jääda mulje, et veebileht on juba laetud, kuid tegelikult ei ole see veel kasutatav [29].

Veebileht on täielikult interaktiivne, kui on täidetud tingimused [29]:

- Veebileht on ära laadinud esimesed sisulised visuaalsed elemendid
- Enamikele veebilehe nähtavatele elementidele on registreeritud *event handler*'id
- Veebileht reageerib kasutaja tegevusele 50 millisekundi jooksul

Lisaks TTI'le vaadeldakse antud töös järgnevaid mõõdikuid [7]:

- First Contentful Paint, mis näitab, kui kaua kulub aega millisekundites, pärast kasutaja liikumist veebilehele, esimese DOM elemendi laadimiseks. Seejuures

arvestatakse, et element oleks reaalselt sisuline – näiteks tekst, pilt või muud sarnased elemendid

- Veebilehe kogusuurus, mis näitab, kui suur on veebilehel kasutatavate failide kogusuurus megabaitides
- Veebilehe lõplik laadimiskiirus

Kõiki kirjeldatud mõõdikuid võimaldab analüüsida Dareboost tööriist, mida antud töö raames kasutatakse. Online tööriist võimaldab valida analüüsimiseks kasutatava virtuaalmasina seadme tüübi ja seadme asukoha. Antud töös tehakse analüüsid virtuaalmasinana Google Chrome'i veebilehitsejas ning samuti virtuaalmasinana mobiiltelefonis Samsung Galaxy S6. Seadmete asukohaks on töös valitud Oslo.

4.4 Tulemused

4.4.1 Veebilehe kogusuurus

Kuna veebilehe laadimine algab ressursside pärimise ja nende allalaadimisega, siis veebilehe suurus on näitaja, mis mõjutab ka kõiki järgnevaid mõõdikuid ja on laadimiskiiruse parandamisel kõrge prioriteediga. Veebilehe kogusuuruse mõõtmise tulemused on leitavad Lisades 2-3.

Mõõtmistulemustest selgub, et valitud komponentide migreerimine TypeScriptile ja seejuures Webpacki kasutusele võtmine, ei suurenda ühegi lehe tüübi puhul märgatavalt kogusuurust.

Küll aga on näha hüppelist tõusu komponentide migreerimise järel Angulari raamistikule – erinevus esialgse rakenduse kogumahuga on ligikaudu 2,5 MB. Erinevus tuleneb sellest, et kuna tegemist on hübriidrakendusega, siis eksisteerivad samaaegselt nii AngularJS rakenduse osad kui ka Angulari rakenduse osad. Mõlemad rakendused vajavad seejuures omaette teeke, mis antud juhul moodustavadki valdava osa kogusuuruse erinevusest. Webpacki pakendamise analüüsi tööriista andmetel moodustavad Angulari juurde kuuluvad teegid 1,67 MB kogu paki suurusest, milleks on 2,64 MB. Antud sammu juures on peaaegu kahekordselt vähenenud Iframe'i sisaldava veebilehe kogusuurus. Eelnevalt oli sellise lehe kogusuurus 9,71 MB, kuid antud mõõtmiste tulemuseks on sama lehe suuruseks 5,54 MB, mis on isegi väiksem kui välise materjal linki sisaldav õppematerjal, mille suuruseks antud hetkel oli 5,6 MB. Kuna kogu mõõtmiste vältel oli nimetatud

Iframe'i sisaldav õppematerjal kõige probleemsema jõudlusega ning linki sisaldav õppematerjal vastupidi üks kiiremaid, siis tõenäoliselt ei ole antud juhul mõõtetulemus korrektne. Tulemuse põhjal võib järeldada, et tööriist, millega mõõtmisi teostatakse, ei suutnud antud sammu juures Iframe'i sisu pärida, mistõttu on tulemuses kajastatud veebilehe kogusuurus ilma Iframe sisuta.

Serveripoolsete optimeerimiste teostamine, mis suures osas hõlmas koodi kokkusurumise aktiveerimist, tõi veebilehtede kogusuuruse tagasi lähemale esialgsele tulemusele. Pildifaili sisaldava veebilehe puhul oli tulemus esialgsest isegi parem. Esialgne kogusuurus pildiga õppematerjali puhul oli 4,46 MB ning pärast serveripoolsete optimeerimiste tegemist oli see 4,01 MB. Ülejäänud suurused jäid esialgsest siiski mõnevõrra kõrgemaks. Näiteks välise lingiga õppematerjali puhul oli esialgne suurus 3,3 MB ning serveripoolse optimeerimise järel 3,86 MB. Iframe'i tulemuse osas püsib endiselt kahtlus, mida kirjeldati eelmisel sammul.

Front-endi poolsed optimeerimised hõlmasid enamjaolt ebavajaliku koodiosa eemaldamist rakendusest ning ressursside laadimise järjekorra optimeerimist. *Front-end* optimeerimiste järel oli kõikide tüüpide veebilehe kogusuurus väiksem, kui esialgse rakenduse puhul. Väikseima kogusuurusega oli välise lingiga õppematerjal, mille puhul kogusuurus pärast *front-endi* optimeerimist oli 2,47 MB. Teistest tüüpidest märgatavalt suurema, kuid esialgsest siiski 0,83 MB võrra väiksema kogusuurusega oli veebileht, mis sisaldas Iframe'i. Pärast *front-endi* optimeerimist oli selle lehe kogusuurus 8,67 MB. Antud lehel on teistest oluliselt suurem kogusuurus seetõttu, et sisu kuvatakse eraldiseisvalt veebilehelt, millele enamjaolt ei rakendu töös teostatud optimeerimised.

Kogusuurus pärast Angulari komponentides kasutatavate teenuste migreerimist uuele raamistikule jäi sisuliselt samaks, nagu see oli eelmisel sammul. Kõikide tüüpide puhul suurenes veebilehe kogusuurus vaid 0,01 MB võrra, mis näitab, et Angulari kood ise ei mõjuta veebilehe mahtu kuigi palju.

Mõõtmistulemused ja nende muutumine erinevate arendusetappide vahel olid mobiilivaate puhul sarnased töölaua rakenduse tulemustega. Ainuke erinevus seisnes selles, et mobiilivaate puhul oli Iframe'i sisaldava veebilehe kogusuurus igal sammul märgatavalt kõrgem teistest tüüpidest. See annab kinnitust eelnevalt kirjeldatud kahtlusele, et töölaua rakenduse puhul ei õnnestunud Iframe'i sisu igal sammul laadida.

4.4.2 First Contentful Paint

Esimeste elementide kuvamine veebilehel on kasutaja jaoks indikaator, et tema päringutega tegeletakse. Seetõttu on esimeste elementide kiire kuvamine oluline, et mitte jätta kasutajale muljet nagu oleks veebileht vigane või mitte kättesaadav. *First Contentful Paint* (FCP) näitab aega millisekundites, mis kulub esimese sisulise elemendi kuvamiseks. Sisuliseks elemendiks loetakse näiteks teksti, pilte või muud sarnast [34]. Mõõtmistulemused on leitavad Lisades 4-5.

Töölaua rakenduse mõõtmistulemustes ei esinenud suuri erinevusi arendusetappide vahel. Arenduste lõpptulemuse mõõtmised olid mõnevõrra paremad, kui esialgse veebilehe puhul. Suurim erinevus toimus YouTube videot sisaldava õppematerjali puhul, kus esialgne FCP tulemus oli 1785 ms ning lõpptulemus 970 ms. Väikseim erinevus oli välise lingiga õppematerjali puhul, kus esialgne tulemus oli 1305 ms ja lõpptulemus 942 ms.

Erinevused mobiilivaate mõõtmistulemustes olid aga oluliselt suuremad. Esimesed arendusetapid tulemusi kuigi palju ei mõjutanud, kuid f*ront-endi* optimeerimismeetodite rakendamine parandas märgatavalt kiirust iga veebilehe tüübi puhul. Näiteks välist linki sisaldava veebilehe puhul oli esialgne FCP tulemus mobiilil 3872 ms ning pärast *front-endi* optimeerimist 2335 ms. Iframe'i sisaldava lehe puhul olid need näitajad vastavalt 3450 ms ja 1957 ms, PDF'i sisaldava veebilehe puhul 3598 ms ja 2187 ms, YouTube videot sisaldava lehe puhul 3100 ms ja 2251 ms ning pildifailiga lehe puhul 3259 ms ja 2195 ms.

4.4.3 Time to Interactive

Time to Interactive mõõtetulemused on leitavad Lisades 6-7.

Sarnaselt eelnevatele mõõtmistulemustele, ei ole ka TTI puhul märgatavaid erinevusi rakenduse esialgse ja TypeScriptile migreeritud versioonide tulemustes.

Kuna TTI näitaja sõltub osaliselt laetavate failide suurusest ja nendega töötamise kiirusest, siis on antud mõõtmistulemuste puhul märgata sarnasusi veebilehe kogumahu tulemustega [29]. Antud juhul suurenes TTI aeg hüppeliselt Angulari kasutusele võtmisega ning komponentide Angularile migreerimisega. Linki sisaldava õppematerjali TTI esialgne tulemus oli 5755 ms ning Angularile migreerimise järel oli see 9577 ms.

Kõrgeim TTI aeg, milleks oli 11142 ms, oli PDF faili sisaldava õppematerjali puhul. Esialgse veebilehe puhul oli see 6587 ms.

Heaks TTI tulemuseks peetakse 0-5,2 s, keskmiseks 5,3-7,3 s ning halvaks üle 7,3 s [29]. See näitab, et kui esialgu oli tulemus keskpärane, siis Angulari kasutusele võtmisega langes tulemus kõvasti alla keskmise.

Serveripoolsed optimeerimismeetodid, mis mahu vähendamise osas andsid parima tulemuse, TTI osas olulisi muutusi ei toonud. Parima tulemuse andis see YouTube videot sisaldava õppematerjali puhul, kus pärast Angularile migreerimist oli Time to Interactive aeg 10701 ms ning pärast serveripoolset optimeerimist 9953 ms. Mõnevõrra vähenes ka välise lingiga õppematerjal, mille serveripoolse optimeerimise järgne tulemus oli 9042 ms. Ülejäänud veebilehe tüüpide tulemuste erinevused ei olnud niivõrd märgatavad.

Angularile migreerimisest tingitud kehvasid tulemusi aitas enim parandada *Front-endi* poolsete optimeerimiste tegemine. Parima tulemuse TTI parandamiseks annab JavaScripti optimeerimine ning jälgimine, et kolmandate osapoolte ning väiksema prioriteediga JavaScripti failid ei blokeeriks veebilehe jaoks oluliste JavaScripti failide käivitamist [29]. Just neid punkte jälgitigi *front-endi* optimeerimises. Siinkohal ei andnud optimeerimine positiivset tulemust ainult Iframe'i sisaldava õppematerjali puhul. Põhjuseks on antud mõõtmise puhul niisamuti see, et Iframe'i sisu tuleb väliselt veebilehelt, mille puhul ei rakendu rakenduses teostatavad optimeerimised. Kõikide teiste veebilehe tüüpide puhul paranes TTI aeg võrreldes Angularile migreerimise järgsetele tulemustele tuntavalt: välise lingiga materjali puhul oli Angularile migreerimise järgne tulemus 9577 ms ning pärast *front-endi* optimeerimist 7379 ms, PDF'i sisaldava materjali korral vastavalt 11142 ms ja 9169 ms, YouTube videot sisaldava materjali puhul 10701 ms ja 7974 ms ning pildifailiga õppematerjali puhul 9449 ms ja 7492 ms.

Angularile migreeritud komponentides kasutatavate teenuste migreerimine uuele raamistikule siinkohal tulemusi eriti ei mõjutanud.

Töölaua rakenduse puhul oli TTI lõpptulemus mõnevõrra kehvem, kui see oli esialgsel rakendusel. Mobiilis teostatavate mõõtmiste lõpptulemused olid iga tüüpi veebilehe puhul vastupidi märgatavalt paremad.

Iframe'i sisaldava õppematerjali TTI aeg oli juba esialgses rakenduses märgatavalt aeglane, kuid Angularile migreerimisel ei kuvanud mõõtmisteks kasutatav tööriist TTI osas mingisugust tulemust, millest võib järeldada, et tööriist lõpetas enne töö, kui veebileht saavutas täieliku interaktiivsuse. Nagu ka töölaua rakenduse puhul, kasvasid mobiilis tehtavate mõõtmiste tulemused oluliselt pärast Angularile migreerimist. Suurim muutus oli PDF faili puhul, kus esialgne TTI aeg mobiilis oli 17209 ms ning pärast komponentide Angularile migreerimist 30630 ms. Väikseim muutus toimus YouTube videot sisaldava õppematerjali juures, kus esialgne TTI aeg mobiilis oli 21425 ms ning pärast komponentide migreerimist Angularile 29140 ms.

Tulemuste põhjal selgub, et mobiilivaate puhul aitas märgatavalt TTI aega parandada serveripoolsete optimeerimiste teostamine, mille tulemusena olid TTI ajad erinevat tüüpi veebilehtede puhul üsna sarnased esialgsega rakendusega. Näiteks YouTube videot sisaldava veebilehe TTI aeg pärast serveris teostatud optimeerimist oli 22730 ms, linki sisaldava veebilehe TTI aeg 17991 ms ning pildifailiga veebilehe TTI aeg 18157 ms.

Lisaks serveripoolsetele optimeerimistele parandasid märgatavalt TTI aega *front-endis* teostatavad optimeerimised, mille tulemusena olid mobiilivaadete TTI ajad oluliselt paremad esialgse veebilehe omadest. Välise lingiga veebilehe esialgne TTI aeg oli 16873 ms ja pärast *front-endi* optimeerimisi oli see 12010 ms. Iframe'i sisaldava veebilehe ajad olid vastavalt 33320 ms ja 17904 ms, PDF faili sisaldava lehe ajad olid 17209 ms ja 15758 ms, YouTube videoga lehel 21425 ms ja 16708 ms ning pildifailiga lehel 16873 ms ja 12560 ms. Ka TTI mõõtmiste puhul ei mõjutanud teenuste Angularile migreerimine kuigi palju tulemusi.

4.4.4 Veebilehe laadimise koguaeg

Veebilehe laadimise koguaeg näitab, kui kaua kulus aega millisekundites kõikide ressursside pärimiseks, alla laadimiseks, töötlemiseks ja käivitamiseks [7]. Kuna veebilehe laadimise koguaeg on sõltuvuses kõikide eelmistes punktides kirjeldatud mõõdikutega, siis on tulemuste muster väga sarnane teiste mõõdikute tulemustega – ka laadimise koguaja puhul ei ole TypeScriptile migreerimine tulemustele eriti mõju avaldanud ning Angularile migreerimine on märgatavalt suurendanud veebilehe täielikuks laadimiseks kuluvat aega. Samuti aitas *front-endi* optimeerimismeetodite rakendamine viia tulemused sarnasele tasemele esialgsetega. Veebilehe laadimise koguaja mõõtmistulemused on leitavad Lisades 8-9.

Koguaja mõõtmiste puhul ilmnesid tulemustes jällegi kaheldavad asjaolud seoses Iframe'i sisaldava veebilehe tulemustega pärast Angularile migreerimist. Kuna tulemused Iframe'i puhul olid pärast Angularile migreerimist lähedal või isegi paremad ülejäänud lehe tüüpide tulemustest, siis võib järeldada, et mõõtmiste teostamiseks kasutatav tööriist ei laadinud Iframe'i sisu korrektselt. Pärast *front-endi* optimeerimise arendusetappi oli Iframe'i tulemus sarnaselt esialgsele rakendusele teiste tüüpide tulemustest märgatavalt kõrgem.

Enamike tüüpide korral oli lõplik veebilehe laadimise koguaeg mõnevõrra kehvem, kui esialgsete mõõtmiste puhul: pildifailiga õppematerjali esialgne laadimise koguaeg oli 6394 ms ja lõplik 7886 ms. YouTube videoga materjali puhul olid need ajad vastavalt 7500 ms ja 8513 ms, PDF ajad 7168 ms ja 9412 ms ning Iframe'i sisaldava veebilehe laadimise ajad 17641 ms ja 21140 ms. Välist linki sisaldava veebilehe puhul oli lõplik laadimise koguaeg 7948 ms, mis on veidi parem kui esialgne laadimise koguaeg 8576 ms.

Mobiilil tehtud mõõtmiste tulemused on oluliselt kõrgemad, kuid mustri poolest väga sarnased töölaua rakendusel saadud mõõtmistulemustega. Vastupidiselt töölaua rakendusele on mobiilil tehtud mõõtmiste puhul korrektselt toimunud ka Iframe'i sisaldava veebilehe mõõtmine. Lisaks on mobiili puhul rohkem veebilehe tüüpe, mille korral lõplikud mõõtmistulemused olid paremad, kui esialgsete mõõtmiste puhul. Näiteks välise lingiga õppematerjali puhul oli esialgne laadimise aeg 17477 ms ja lõplik aeg 14816 ms, PDF'i sisaldava lehe puhul oli esialgne aeg 17836 ms ja lõplik aeg 17516, pildiga lehe esialgne laadimise aeg oli 17542 ms ja lõplik aeg 14807 ning YouTube videoga veebilehe esialgne aeg 20251 ms ja lõplik aeg 17956 ms. Iframe'i sisaldava veebilehe lõplik laadimise koguaeg oli mõnevõrra kehvem kui esialgse mõõtmise puhul – esialgne aeg oli 40495 ms ja lõplik aeg 41961 ms.

5 Järeldused

Töö tulemusena valmis hübriidrakendus, mille *front-endi* põhiosa on raamistikul AngularJS ning töö alguses valitud komponendid on migreeritud raamistikule Angular 9. Kõik valitud komponendid ning nende juurde kuuluvad osad õnnestus edukalt migreerida Angulari raamistikule. Lõplikus töös eksisteerivad üksikud stiilivead, mis on tingitud projektis kasutusel oleva Material Design teegi erinevustest AngularJS ja Angulari versioonides. Vead seisnevad suuresti elementide ebakorrektses paigutuses veebilehel ning seega nende parandamine tulevikus ei tohiks mõjutada lõplikus rakenduses saavutatud veebilehe laadimise kiirust. Hübriidrakenduse seadistamisel lähtuti Angulari ametlikust juhendist, kus anti ka seadistuse osas nõuandeid veebilehe kiiruse parandamise seisukohast.

Täiendavalt veebilehe *front-end* raamistiku migreerimisele rakendati töö jooksul mitmeid raamistikust sõltumatuid meetodeid veebilehe laadimiskiiruse tõstmiseks. Nende meetodite realiseerimise võimaldamiseks on rakenduses kasutatud moodulite pakendamiseks Webpacki ning rakendus on käivitatud veebiserveril.

5.1 Jõudlusele negatiivset mõju avaldanud arendustööd

Mõõtmistulemustest selgub, et hübriidrakenduse kujul Angular 9 kasutusele võtmine ei täida veebilehe kiiruse parandamise eesmärki, vaid vastupidi aeglustab seda. Mõõtmistulemuste põhjal võib öelda, et selle põhjuseks on eelkõige rakenduse kogumahu hüppeline tõus, mis on tingitud Angulari jaoks vajalike teekide lisamisest rakendusse. Peale selle on hübriidrakendus endiselt suures osas AngularJS raamistikul, mille jõudlusele negatiivsed osad eksisteerivad ka hübriidrakenduses.

5.2 Jõudlusele positiivset mõju avaldanud arendustööd

Kuigi rakenduse komponentide Angularile migreerimine andis oodatule vastupidise tulemuse, suutsid raamistikust mittesõltuvad optimeerimistööd taastada rakenduse esialgse jõudluse ning vähesel määral seda parandada. Seejuures avaldusid positiivsete arendustööde tulemused tugevamalt mobiiliseadmete puhul, mille jõudluse probleem oli töö alguses oluliselt tähelepanuväärsem, kui töölaua veebilehe puhul.

Iga mõõdetava näitaja korral avaldasid silmnähtavat kasu *front-endis* teostatud optimeerimistööd. Serveri poolel rakendatud optimeerimismeetodid ei näidanud kiiruse osas kuigi suuri muutusi töölaua veebilehe puhul, kuigi rakenduse kogumaht vähenes oluliselt pärast *compressioni* aktiveerimist rakenduses. Mobiiliseadmetel teostatud mõõtmiste puhul oli lisaks märkimisväärsele mahu vähenemisele näha ka serveripoolsetest optimeerimistöödest tingitud kiiruse paranemist.

5.3 Arenduse mõju rakendusele

Töö käigus mõõdeti jooksvalt arenduseks kuluvat aega. Ainuüksi arendusele kulus kokku 220 h ehk 27,5 kaheksatunnist tööpäeva. Selle arvestuse sisse ei kuulu analüüsiks, mõõtmiste tegemiseks ja muude arenduseväliste tööde jaoks kulunud aeg. Ajamahukamad tööd olid antud juhul migreerimiseks ettevalmistuste tegemine, komponentide Angularile migreerimine ning nende toimingute jaoks vajalike seadistuste tegemine.

Kuna veebilehe jõudlus praegusel hübriidrakenduse kujul ei saavutanud eesmärgiks seatud tulemust, siis võib väita, et arenduseks kulunud aeg ei ole kooskõlas praegusel kujul oleva rakenduse saavutatud tulemusega. Seejuures tuleks aga arvestada, et AngularJS LTS lõppeb 2021. aasta juulikuus ning seetõttu on töö autori soovitus migreerimisega jätkata. Siiani arenduseks kulunud ajast on suur osa migreerimiseks esialgsete seadistuste tegemine, mis on ajalises mõttes ühekordne väljaminek ning tulevikuarendustes kulub seega suurem osa arenduseks kuluvast ajast AngularJS koodi Angularile ümberkirjutamiseks. Lisaks on AngularJS lõplikul eemaldamisel rakendusest rohkesti potentsiaali parandada veebilehe laadimise kiirust kasvõi veebirakenduse kogumahu vähendamise poolest. Samuti avanevad mitmed Angulariga seotud optimeerimisvõimalused, mille rakendamine polnud hübriidrakenduse korral teostatavad.

5.4 Optimeerimisvõimalused tulevikus

Tulevaste arendustega seotud optimeerimisvõimalused jagunevad suures osas kaheks. Üks osa on tugevalt seotud Angularile migreerimisega. Rakenduse lõplikul migreerimisel Angularile saab jäädavalt eemaldada AngularJS osad rakendusest, mis praegusel hetkel hoiavad jõudlust madalal tasemel ning seejuures väheneb AngularJS teekide võrra rakenduse kogumaht. Seejärel on võimalik täielikult kasutusse võtta Angulari pakutavad

arendust mugavdavad tööriistad ning jõudlust parandavad tehnoloogiad. Näiteks saab arenduse mugavdamiseks võtta kasutusele Angular CLI (*Command-line interface*) ning Angulari koodi kompileerimiseks *Ivy rendereri*, mis tagab väiksemad failid ning optimaalsema kompileeritud koodi. Tulevikus tasub kasutada ka Angulari Universali pakutavat SSR võimalust.

Teine osa puudutab rohkem koodi kvaliteeti. Arenduse käigus oli korduvalt näha väljakommenteeritud koodi või funktsionaalsusi, mida tegelikult pole enam kasutusel. Vaatamata sellele, et kirjeldatavat koodi tegelikult ei käivitata, võtab see siiski märkimisväärselt ruumi, mis suurendab veebilehe ressursside allalaadimiseks kuluvat aega ning raskendab JavaScripti käivitamist brauseris. Seetõtu tuleks kogu koodi puhul vaadata, mida reaalselt kasutatakse ja kõik ebavajalik koodist eemaldada. Samuti tuleks kriitilise pilguga üle vaadata rakenduses tehtavad päringud, mida praegu tehakse lehe laadimisel keskmiselt üle saja. Niisamuti nagu mittekasutatava koodi puhul, tuleks päringud, mida vaja ei ole eemaldada ning mahukamaid päringuid võimalusel optimeerida. Näiteks praeguses rakenduses on üks mahukamaid päringuid *educationalContext*, mille eesmärk on peamiselt õppevara menüü täitmine. Küll aga päritakse korraga kogu menüü sisu, mille maht on isegi pärast *compressioni* lisamist 134 kB. Soovituslik oleks menüüga seonduv loogika ümber korraldada nii, et alles menüüs sügavamale liikudes tehakse järgnevad päringud, mitte ei laeta kogu sisu korraga. Antud päringu ajutise väljalülitamisega kaasnes üle ühe sekundiline erinevus nii TTI kui veebilehe täieliku laadimise ajas, seega potentsiaalne võit on ainuüksi selle päringu parandamisel umbes üks sekund.

6 Kokkuvõte

Käesoleva töö eesmärgiks oli E-koolikoti sisulehe laadimiskiiruse parandamine raamistiku vahetamise ning raamistikust sõltumatute optimeerimismeetmete rakendamise abil.

Eesmärgi realiseerimiseks analüüsiti populaarsemaid raamistikke nii jõudluse kui arendusmugavuse vaatenurgast ning valiti antud projekti jaoks autori hinnangul parim lahendus, milleks osutus Angular 9 raamistik. Jõudluse osas on Angular veel mõnevõrra kehvem Reactist ja Vuest eelkõige raamistiku suuruste erinevuse tõttu. Vaatamata sellele, on Angulari jõudlus esialgse versiooniga oluliselt paranenud ning arendusmeeskonna fookus püsib jätkuvalt Angulari suure mahu vähendamisel, mis on üks peamised probleeme Angulari raamistiku puhul [22]. Seetõttu on Angularil potentsiaali olla tulevikus jõudluse poolest võrdväärne Reacti ja Vuega. Seda arvesse võttes osutus kaaluvaks aspektiks Angulari tähelepanuväärne arendusmugavus ja antud projekti jaoks vajalik dokumentatsioon, mis puudus teiste võrreldavate raamistike puhul. Nimelt sisalduvad Angulari raamistikus veebirakenduse arenduseks elementaarsed elemendid, mis Reacti ja Vue puhul suuremas osas puuduvad [22].

Raamistiku valiku järel migreeriti vaatluse all olevale E-koolikoti veebilehele omased komponendid valitud raamistikule ning rakendati raamistikust sõltumatud optimeerimismeetmed, mis hõlmasid endas nii *front-endi* poolseid kui ka serveris teostatavaid optimeerimistöid. Iga suurema arendusetapi juures teostati valitud veebilehtede põhjal jõudluse mõõtmised brauseris ja mobiiliseadmes.

Mõõtmistulemustest selgub, et töö lõpptulemus on vähesel määral parem, kui esialgse veebilehe jõudlus. Sisu poolest lihtsaima ehk ainult välist linki sisaldava E-koolikoti materjali veebilehe TTI oli brauseris tehtava mõõtmise puhul esialgselt 5774 ms ning töö lõpptulemusena 7347 ms. See-eest mobiiliseadmes teostatud mõõtmise esialgne tulemus oli 16873 ms ning töö lõpptulemusena 12560 ms. Sarnane tulemus kajastub veebilehe laadimise koguaja mõõtmistulemustes, kus samasuguse veebilehe puhul oli brauseris esialgne laadimise koguaeg 6394 ms ning lõplik 7886 ms. Mobiiliseadme puhul oli esialgne laadimise koguaeg 17542 ms ja lõplik 14807 ms.

Tulemuste põhjal on näha, et Angulari lisamine rakendusse ei avaldanud ühelegi mõõdikule märgatavat positiivset mõju. Raamistikust sõltumatud optimeerimised, mis hõlmasid endas enamjaolt rakenduse mahu vähendamist ning JavaScripti laadimise optimeerimist, avaldasid see-eest iga mõõdiku puhul positiivset mõju. Arvestades arenduseks kulunud aega ning saavutatud tulemust võib öelda, et praegusel kujul olev Angulari ja AngularJS hübriidrakendus ei saavuta püstitatud eesmärki ega ole otstarbekas meetod veebilehe laadimiskiiruse tõstmiseks.

Autori hinnangul on siiski mõistlik veebilehe raamistiku vahetamisega jätkata, kuna praegusel kujul oleva hübriidrakenduse jõudluse langus on mõõtmisandmete põhjal tingitud rakenduse mahu tõusmisest ja selle läbi JavaScripti rakendamise aeglustumisest. Kogu E-koolikoti migreerimisel uuele raamistikule saab olemasoleva AngularJS koodi koos selleks vajaminevate teekidega eemaldada, mille järel väheneb AngularJS raamistiku võrra koodimaht ning kaob AngularJS negatiivne mõju. Lisaks sellele on olemasolevas rakenduses kasutusel oleva AngularJS raamistiku toetamine ning edaspidine arendus lähiajal lõppemas, mistõttu on soovituslik praegune raamistik igal juhul välja vahetada. Käesoleva töö raames on selle tarvis tehtud juba vajalikud protsessi soodustavat eeltööd.

Kasutatud kirjandus

[1] Aggarwal, S. Angular vs AngularJS – A complete Comparison Guide 2019. [WWW] https://www.techaheadcorp.com/blog/angular-vs-angularjs/ (17.04.2020)

[2] An, D. Find out how you stack up to new industry benchmarks for mobile page speed. [WWW] https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/ (13.05.2020)

[3] AngularJS Changelog. [WWW] https://github.com/angular/angular.js/blob/master/CHANGELOG.md (17.04.2020)

[4] Brooks, N. The Atlas Rank Report: How Search Engine Rank Impacts Traffic. [WWW] http://www.inesting.org/ad2006/adminsc1/app/marketingtecnologico/uploads/Estudos/atlas%20onepoint%20-%20how%20search%20engine%20rank%20impacts%20traffic.pdf (13.05.2020)

[5] Clow, M. Angular 5 Projects: Learn to Build Single Page Web Applications Using 70+ Projects, Apress, 2018 (17.04.2020)

[6] Comparison with Other Frameworks. [WWW] https://vuejs.org/v2/guide/comparison.html (21.04.2020)

[7] Costello, R. How to Measure Site Speed & Performance. [WWW] https://www.deepcrawl.com/knowledge/white-papers/site-speed-performance-guide/how-to-measure-speed-performance/ (01.05.2020)

[8] Data Binding. [WWW] https://docs.angularjs.org/guide/databinding (17.04.2020)

[9] Dhandapani, S. Virtual DOM – the Difference Maker in React JS. [WWW] https://www.pluralsight.com/guides/virtual-dom-difference-maker-react-js (13.05.2020)

[10] Ensure text remains visible during webfont load. [WWW] https://web.dev/font-display/ (15.05.2020)

[11] Hill, D. AngularJS End of Life Announced. [WWW] https://www.convective.com/angularjs-end-of-life/ (17.04.2020)

[12] How Search algorithms work. [WWW] https://www.google.com/search/howsearchworks/algorithms/ (17.04.2020)

[13] Introducing JSX. [WWW] https://reactjs.org/docs/introducing-jsx.html (13.05.2020)

[14] Introduction to Angular concepts. [WWW] https://angular.io/guide/architecture (19.04.2020)

[15] Introduction to components and templates [WWW] https://angular.io/guide/architecture-components (19.04.2020)

[16] Is React library or a framework. [WWW] https://developer.com/is-reactjs-library-or-a-framework-a14786f681a0 (17.04.2020)

[17] Muhsin, M. Why you should render React on the server side. [WWW] https://blog.logrocket.com/why-you-should-render-react-on-the-server-side-a50507163b79/ (21.04.2020)

[18] ngReact GitHub. [WWW] https://github.com/ngReact/ngReact (21.04.2020)

[19] ngVue GitHub. [WWW] https://github.com/ngVue/ngVue (21.04.2020)

[20] PageSpeed: Specify a character set early. [WWW] https://gtmetrix.com/specify-a-character-set-early.html (15.05.2020)

[21] Preload key requests. [WWW] https://web.dev/uses-rel-preload/ (15.05.2020)

[22] Schwarzmüller, M. Angular vs React vs Vue – My Thoughts. [WWW] https://academind.com/learn/angular/angular-vs-react-vs-vue-my-thoughts/ (21.04.2020)

[23] Server-Side Rendering. [WWW] https://vuejs.org/v2/guide/ssr.html (21.04.2020)

[24] Stackoverflow. [WWW] https://stackoverflow.com/questions/tagged/angular (14.05.2020)

[25] Stackoverflow. [WWW] https://stackoverflow.com/questions/tagged/reactjs (14.05.2020)

[26] Stackoverflow. [WWW] https://stackoverflow.com/questions/tagged/vue.js (14.05.2020)

[27] The Good and the Bad of Angular Development. [WWW] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/ (19.04.2020)

[28] The Good and the Bad of TypeScript. [WWW] https://www.altexsoft.com/blog/typescript-pros-and-cons/ (19.04.2020)

[29] Time to Interactive. [WWW] https://web.dev/interactive/ (21.04.2020)

[30] TypeScript – Overview. [WWW] https://www.tutorialspoint.com/typescript/typescript_overview.htm (19.04.2020)

[31] Upgrading for performance. [WWW] https://angular.io/guide/upgrade-performance (17.05.2020)

[32] Upgrading from AngularJS to Angular. [WWW] https://angular.io/guide/upgrade (17.05.2020)

[33] Wagner, J. Why Performance Matters. [WWW] https://developers.google.com/web/fundamentals/performance/why-performance-matters (13.05.2020)

[34] Walton, P. First Contentful Paint. (FCP) [WWW] https://web.dev/fcp/ (15.05.2020)

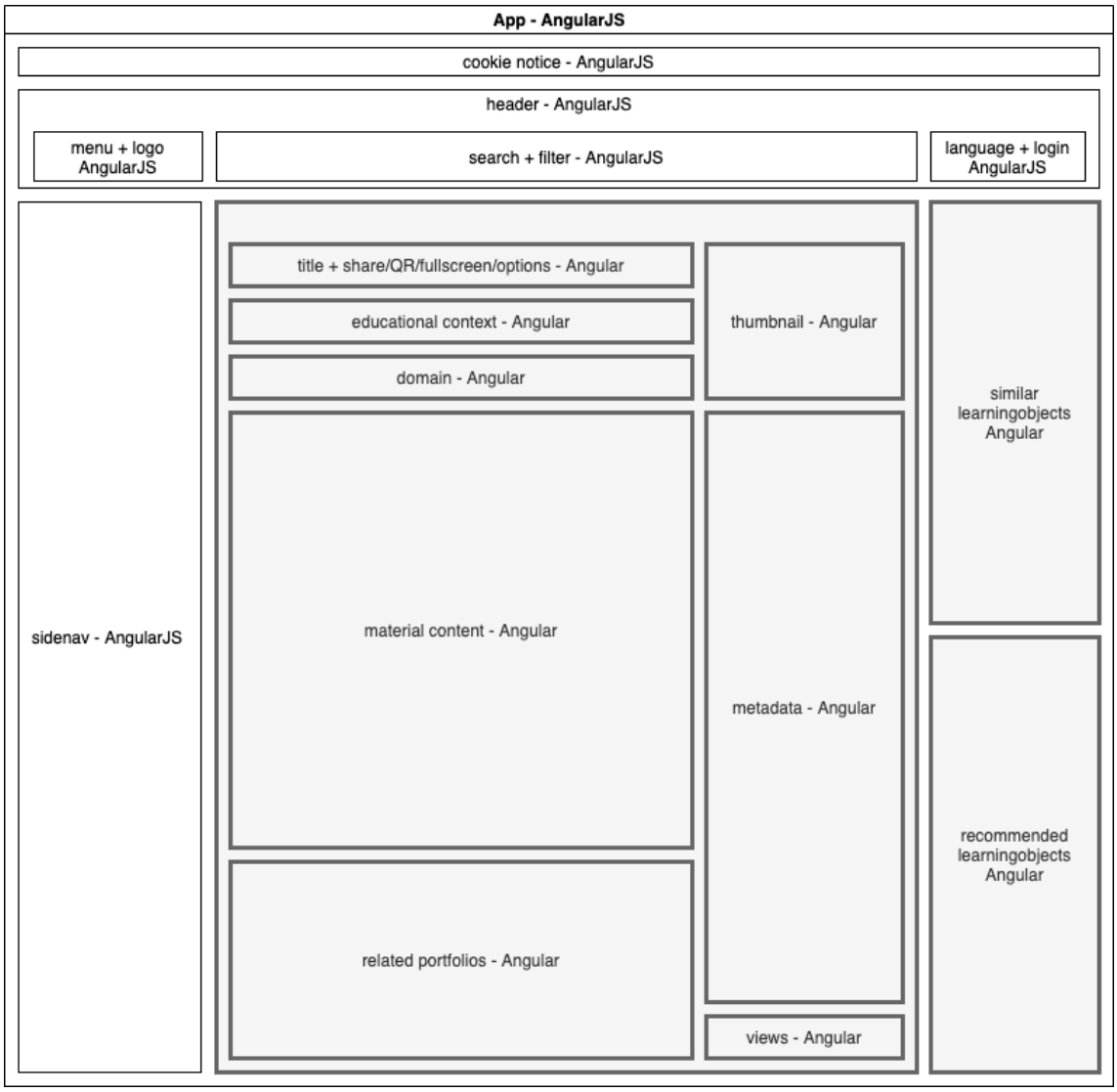
[35] Web framework ratings. [WWW] https://hotframeworks.com/ (21.04.2020)

[36] What is AngularJS. [WWW] https://docs.angularjs.org/guide/introduction#! (17.04.2020)

[37] Why Minify JavaScript Code. [WWW] https://www.cloudflare.com/learning/performance/why-minify-javascript-code/ (15.05.2020)

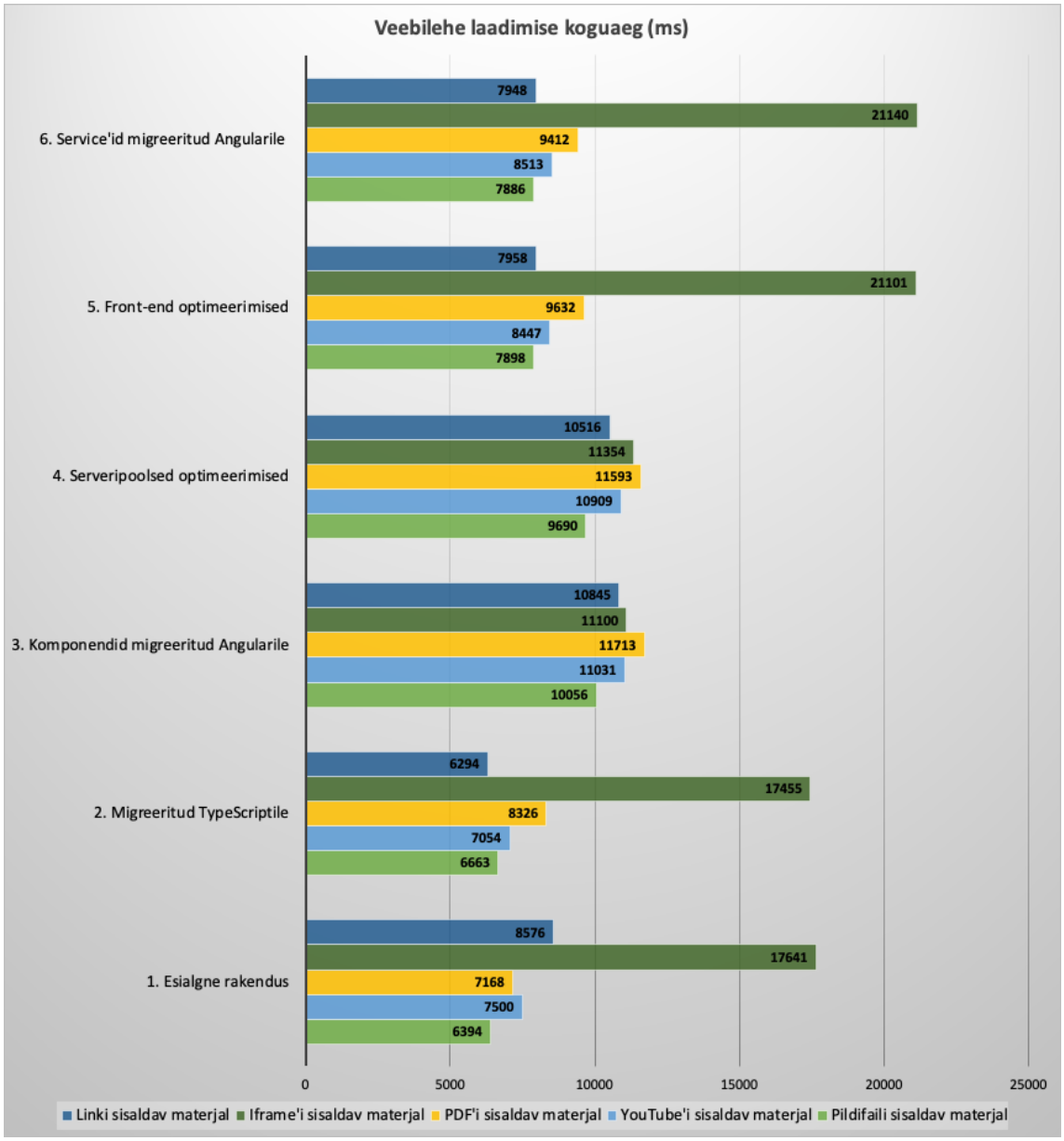
[38] Witalec, S. Apps That Work Natively on the Web and Mobile [WWW] https://blog.angular.io/apps-that-work-natively-on-the-web-and-mobile-9b26852495e7 (19.04.2020)

Lisa 1 – Mõõdetava veebilehe struktuur



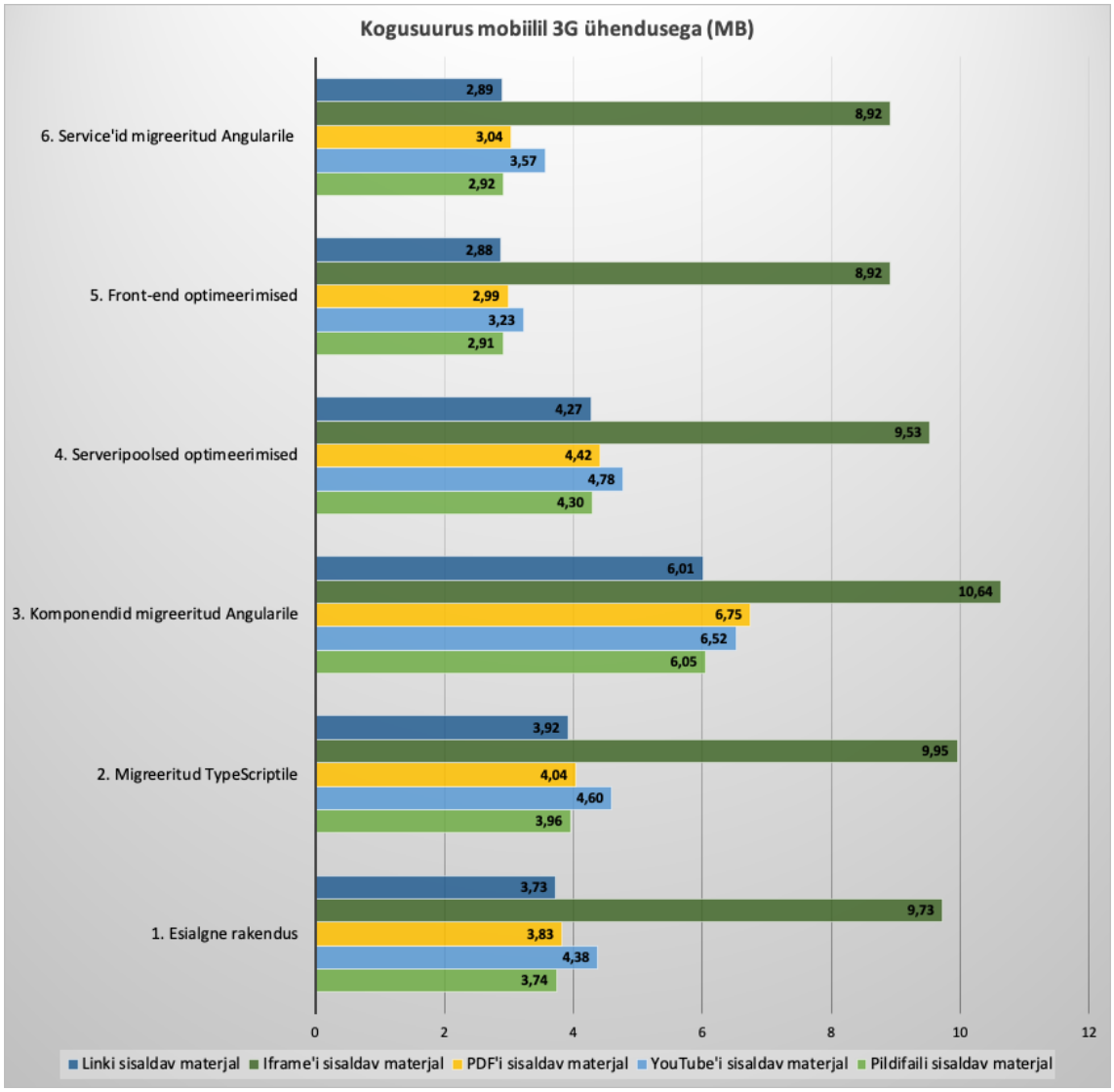
Joonis 3. Mõõdetava veebilehe struktuur.

Lisa 2 – Veebilehe kogusuurus



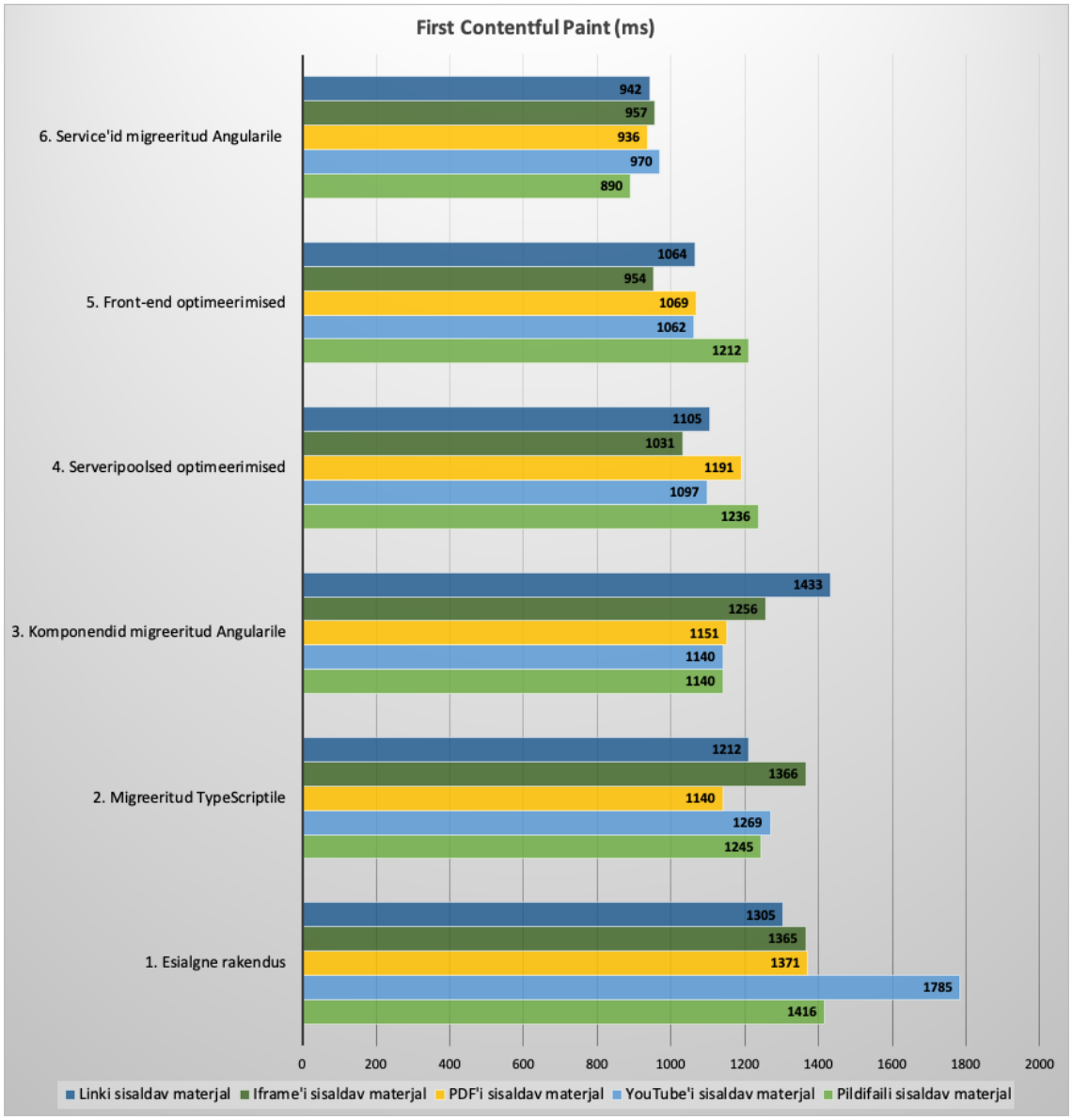
Joonis 4. Veebilehe kogusuurus.

Lisa 3 – Veebilehe kogusuurus mobiilil



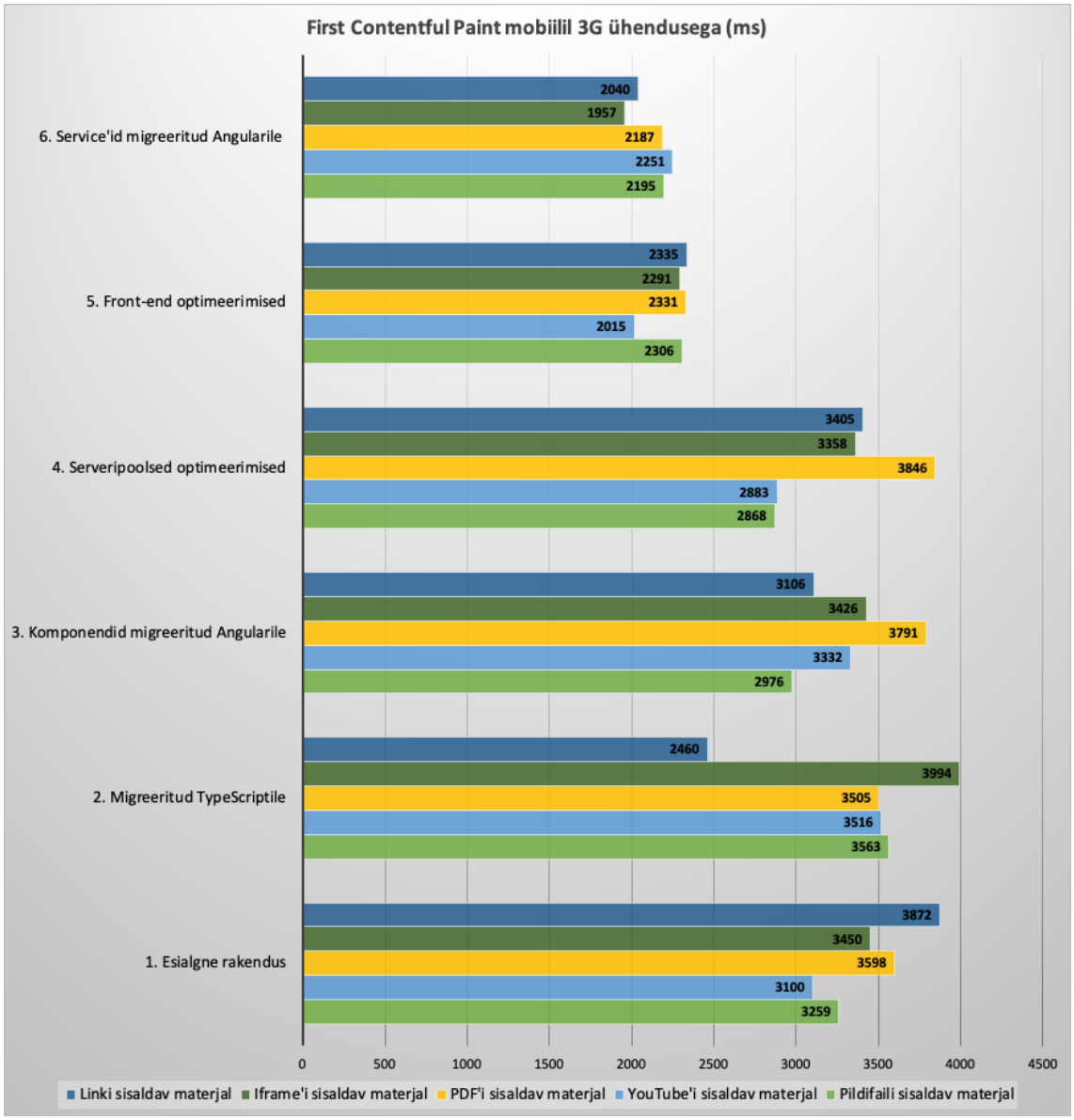
Joonis 5. Veebilehe kogusuurus mobiilil.

Lisa 4 – First Contentful Paint



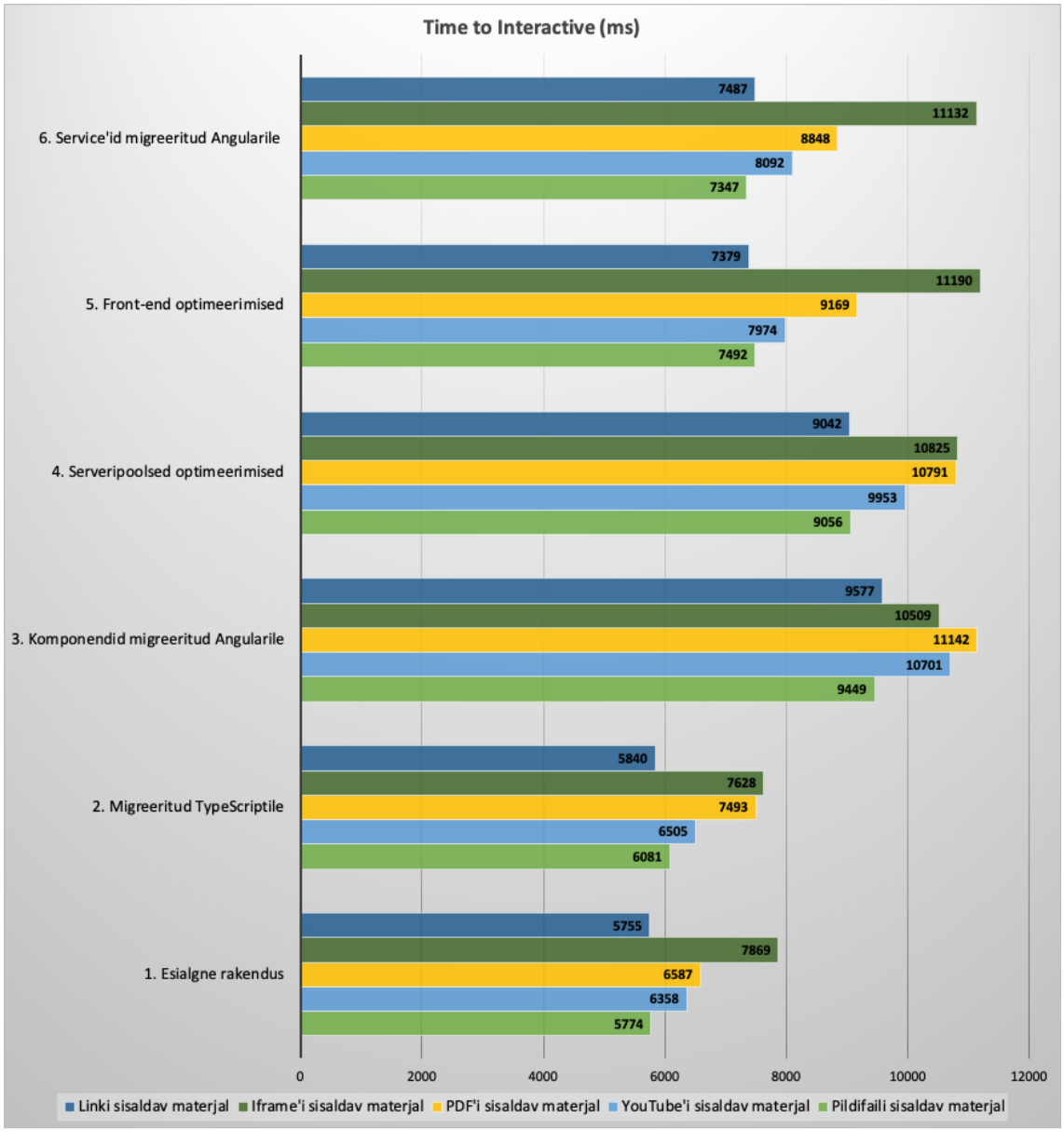
Joonis 6. First Contentful Paint.

Lisa 5 – First Contentful Paint mobiilil



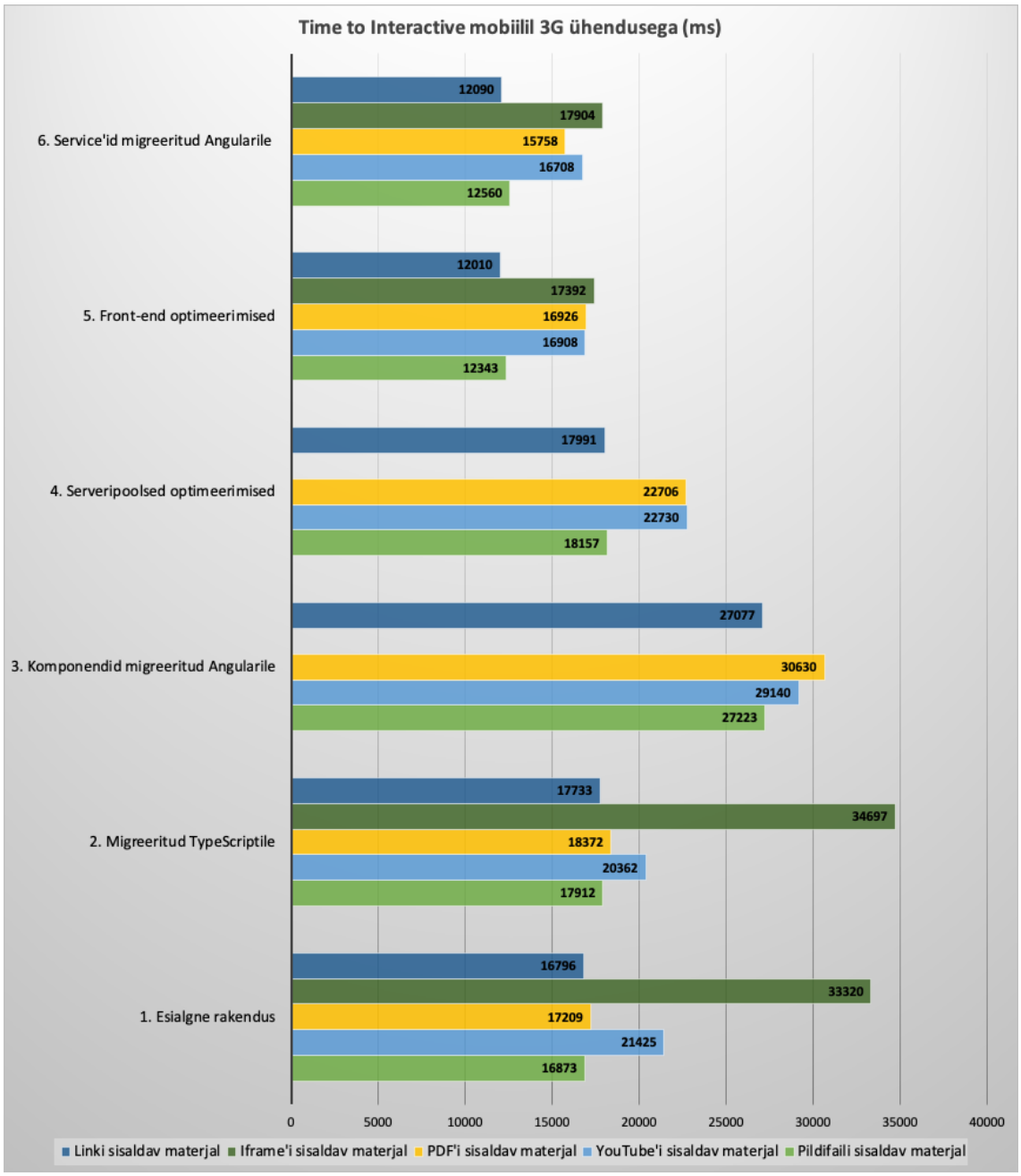
Joonis 7. First Contentful Paint mobiilil.

Lisa 6 – Time to Interactive



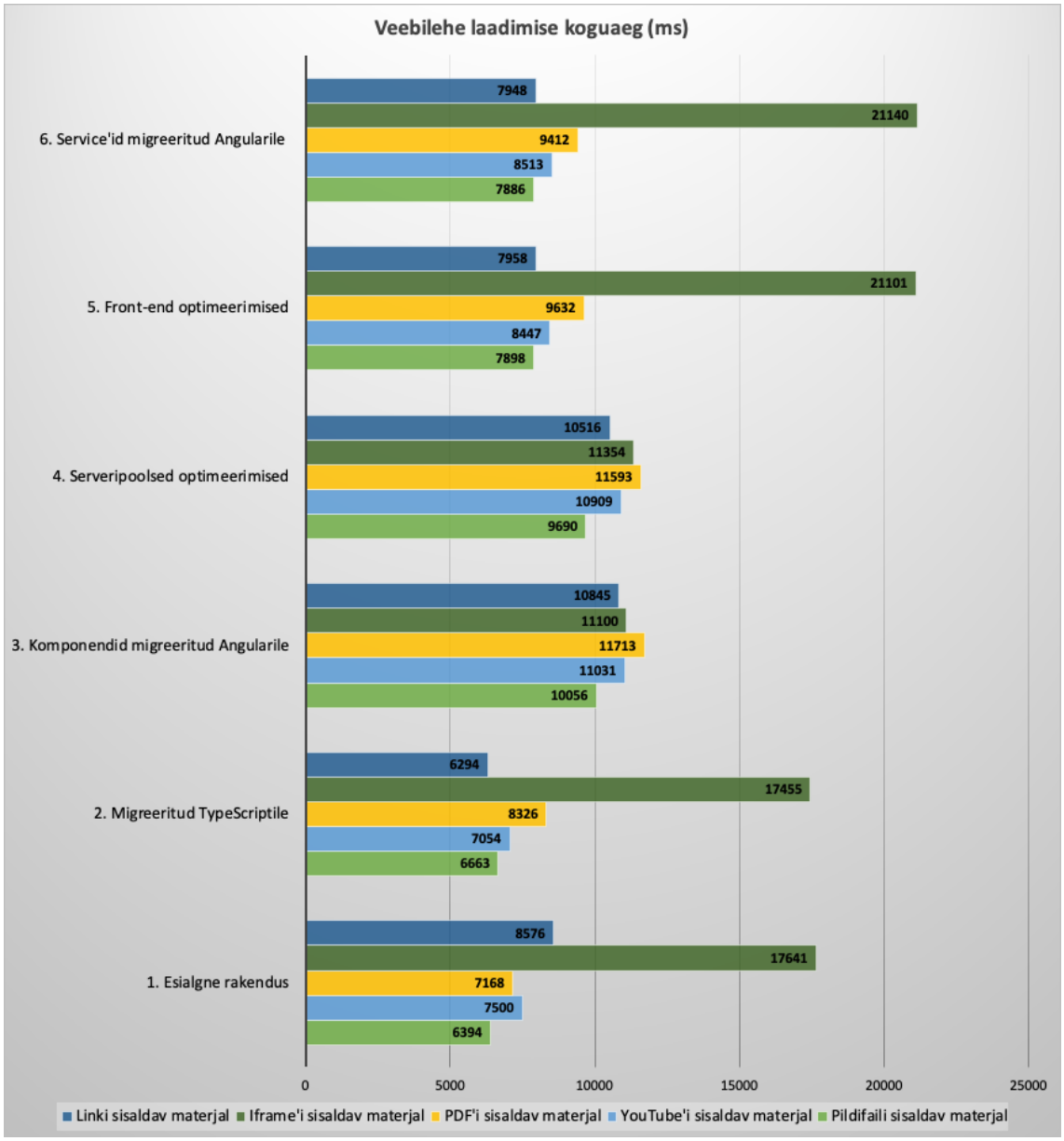
Joonis 8. Time to Interactive.

Lisa 7 – Time to Interactive mobiilil



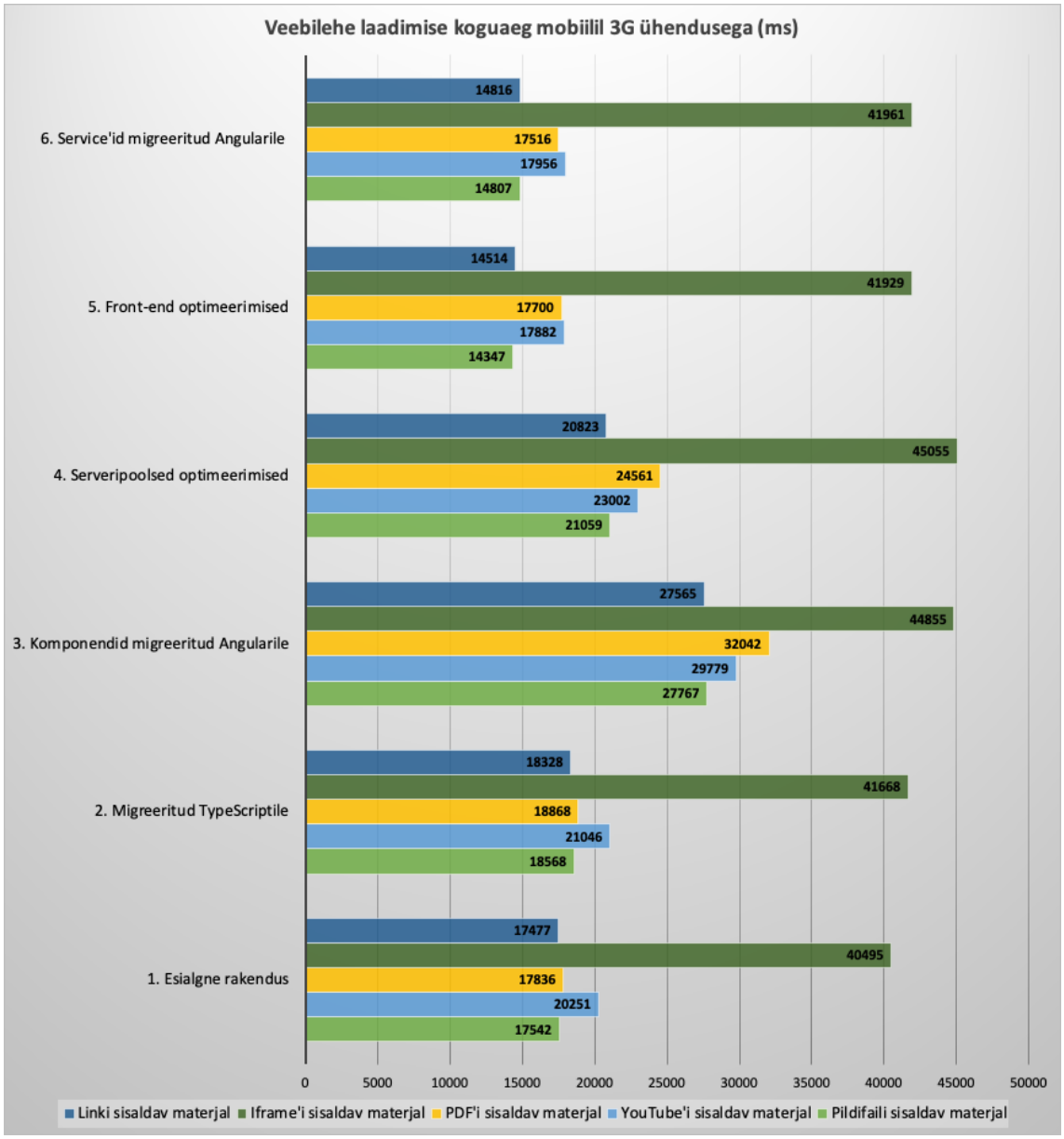
Joonis 9. Time to Interactive mobiilil.

Lisa 8 – Veebilehe laadimise koguaeg



Joonis 10. Veebilehe laadimise koguaeg.

Lisa 9 – Veebilehe laadimise koguaeg mobiilil



Joonis 11. Veebilehe laadimise koguaeg mobiilil.