TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT

Kristina Nassonenko 134566IAPB

# JAVASCRIPT TEEKIDE JÕUDLUSE VÕRDLUS

Bakalaureusetöö

Juhendaja: Teodor Luczkowski
PhD
Lektor

Tallinn 2017

# 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: Kristina Nassonenko

01.12.2016

# Annotatsioon

Käesoleva bakalaureuse töö eesmärgiks on uurida ja võrrelda JavaScript teekide jõudlust. Uurimiseks oli valitud 4 teeki: jQuery, MooTools, Underscore.js ja Lodash.js ning lisaks oli uuritud puhta JavaScripti töökiirus. Teekide jõudluse uurimiseks olid valitud erinevad tegevused ehk funktsioonid, mis töötavad massiivide, objektide, ridade (String'ide) ja DOM-elementidega. Samuti oli kirjutatud rakendus programmeerimiskeeltes PHP ja JavaScript, mis mõõdab koodi täitmise kiirust Benchmark.js teegi abil. Selleks, et teha eksperimendid objektiivsem, testid olid käivitatud kolmes brauseris: Google Chrome, Mozilla Firefox ja Microsoft Edge.

Töö oluliseks tulemuseks on erinevate tegevuste täitmise kiiruse ajamõõdud erinevates teekides, mis annab võimaluse sellise teekide jõudluse võrrelda. Teekide vahel parima jõudluse on näidanud Lodash.js. Underscore.js sai teise kohta, MooTools – kolmanda ning vaatamata oma populaarsusele jQuery on näidanud kõike halvema tulemuse. Tähtis järeldus, mis oli tehtud antud töö tulemuste põhjal, on see, et puhas JavaScript töötab kiiremini ilma teekide kasutamist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 6 peatükki, 12 joonist, 4 tabelit.

# Abstract

## Perfomance comparison of JavaScript libraries

JavaScript is a constantly developing programming language. Nowadays it is used in over than 90% [1] of all websites. Since JavaScript invention in 1995 it's popularity has been constantly growing and a lot of developers started to think, how to make this programming language easier and more comfortable for use, and soon enough, more and more libraries were created by other developers.

The purpose of this thesis is to explore and compare performance of JavaScript libraries. For performance comparison, 4 libraries had been chosen: jQuery, MooTools, Underscore.js and Lodash.js. In addition, performance of native JavaScript was also compared with library results. Performance comparison of libraries was made by measuring execution speed of different functions in different categories, such as arrays, objects, Strings and DOM-elements. For this purpose, application was developed in PHP and Javascript. For benchmarking purposes, Benchmark.js library was used for measuring execution speed of code implementation for every function. To make experiments more objective, tests were launched in 3 different browsers: Google Chrome, Mozilla Firefox and Microsoft Edge, to make experiments more objective.

As the result of this thesis, we have gathered information about functions execution speed in different libraries, this data gives us ability to compare overall perfomance of these libraries.

Summaring results, Lodash has shown best performance among libraries. Underscore.js is on the second place and MooTools is on the third. In spite of jQuery's popularity, it has shown worst results. An important outcome of this thesis is that native JavaScript performs much faster than any JavaScript library.

The thesis is in Estonian language and contains 27 pages of text, 6 chapters, 12 figures, 4 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| Open Source | Avatud lähtekoodiga tarkvara. Tarkvara, mille lähtekoodi ja dokumentatsiooni saab muuta iga soovija. |
| Utility program | Utiliit. Programm, mis on mõeldud kitsale kasutajaskonnale. |
| Plugin | Tarkvaramoodul, mis laiendab teiste tarkvarasüsteemide võimalusi. |
| DOM | *Document Object Model*. Dokumendi objektmudel. Liides, mis suhtleb XML ja HTML dokumentidega. |
| Bug | Viga programmis või süsteemis |
| AMD | *Asynchronous Module Definition*. Asünkroonse mooduli määramine. Lähenemine arenduses, mis annab võimaluse luua moodulid, mis töötavad asünkroonselt. |
| Ajax | *Asynchronous Javascript and XML*. Asünkroonne Javascript ja XML. Rakenduse kliendi poolel kasutatav veebirakenduse tehnika. |
| API | *Application Programming Interface.* Rakendusliides. Reeglistik, mis võimaldab arendajatel suhelda valmisprogrammiga. |
| JavaScript engine | Spetsiaalne programm, mis töödeldab JavaScript koodi brauserites. |
| JIT | *Just-in-time.* Täpselt samal ajal. |
| AST | *Abstract syntax tree.* Abstraaktne süntakse puu. |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

JavaSctipt on tänapäevane ja pidevalt edasiarendatav programmeerimiskeel. Raske on endale ette kujutada kaasaegseid veebilehti, kus seda ei kasutataks. Vastavalt statistikale kasutavad [1] rohkem kui 90% kõikidest veebilehedest JavaScritpt'i või selle teeke. Seda kasutavad nii miljonite kasutajatega hiigelettevõtted, nagu Google, Facebook, Youtube, kuid ka väiksed startupid, millest võib järeldada, et JavaScripti kasutamisel ei ole mingeid piiranguid, seda võib kasutada vaatamata süsteemi suurusele ning kasutajate kogusele. Kuna JavaScript võeti kasutusele 1995. aastal ja selle populaarsus kasvas kiiresti, siis paljud arendajad hakkasid mõtlema, kuidas saaks muuta selle programmeerimiskeele kasutamist mugavamaks ja lihtsamaks. Seega loodi väga palju erinevaid JavaScripti teeke. Teegid aitavad ühendada programmeerimiskeele klasse ning funktsioone, mille abil saab lihtsustada rakenduste arendamist ja lühendada programmkoode. Mõnikord teekide kasutamine aitab isegi kiirendada programmkoodi täitmise kiirust.

## 1.1 Ülesandepüstitus

Kuna JavaSctipt on olnud aktuaalne programmeerimiskeel pika aja jooksul ning selle teekide kogus aina suureneb, tekivad järgnevad küsimused: Kas on mingi erinevus teekides? Kui teekide põhimõtteline ülesanne on koodi optimeerimine ja koodi täitmisekiiruse vähendamine ja siis tekkib ka küsimus: Kas tõesti erinevad teegid täidavad sama funktsionaalsust erineva kiirusega? Need küsimused said aluseks selle lõputöö teema kujundamisel. Käesolevas lõputöös uuritakse teekide jõudlust funktsioonide täitmisekiiruse näitel. Selle lõputöö raames sai koodi täitmisekiiruse mõõtmiseks kirjutatud ka rakendus, kasutades PHP ja JavaScript´i. Kõik kiirusemõõtmised on tehtud teegi Benchmark.js abil, mis püüab käima panna testid nii kiiresti, kui see on võimalik ning samal ajal teha seda maksimaalselt kõrge täpsusega. Selles töös nimetatakse testideks ühe antud funktsiooni täitmist ühes antud teegis. Mõõtmisi teostatakse tsüklitega ja tulemuseks saadakse täidetud operatsioonide arv sekundis. Sõltuvalt tulemustest võib võrrelda ja teha järeldusi, kui kaua töötab üks ja sama funktsionaalsus erinevates JavaScript teekides. Selleks, et testida teekide jõudlust objektiivselt ja operatsioonide

hulka, said valitud funktsioonid, mis töötavad massiivide, ridade (*String*'ide) ja DOM elementidega.

# 2 Teekide ülevaatus

Selle töö raames uurimiseks oli valitud 4 JavaScripti teeke mida võrreldakse natiivse JavaScript'iga. See on:

- jQuery (versioon 3.1.1)
- Underscore.js (versioon 1.8.3)
- Lo-dash.js (versioon 4.17.2)
- MooTools.js. (versioon 1.6.0)

Teegide valimise kriteeriumiseks oli:

1. Teegi dokumentatsiooni olemasolu.
2. Teegid on siiamani uuendatakse. Ilmuvad uued versioonid ning on olemas tehniline tugi.
3. Teegi suurus ei ületa 300 KB
4. Teegid on esitatud avatud lähtekoodiga tarkvarana
5. Kõikidel antud teegidel on olemas oma GitHub[1] 'selts'. See tähendab, et teegi ei arenda kindel firma või arendajate meeskond. Iga soovija võib oma panust teha.

## 2.1 jQuery

jQuery – see on kiire, mugav ja funktsionaalselt jõukas JavaScript'i teek, mis oli pakutud maailmale „open source" projektina 2006. aastal [2]. Möödunud juba 10 aastat ja selle aja jooksul ta sai kõige populaarsemaks ning laialt levinumaks JavaScripti teegiks.

Selle fookustamine baseerub HTML ja JavaScript'i vastastikul mõjul. Funktsionaalsest küljest jQuery annab võimalust kergelt saada ligipääs igale DOM elemendile, teha pöördumist DOM atribuutidele ja manipuleerida nendega. Lisaks jQuery võimaldab töödeldada erinavaid sündmusi ning luua animatsioone ja visuaalseid effekte. Suureks eeliseks on see, et kood, mis on kirjutatud jQuery teegi abil ei konflikteeri funktsioonidega, mis on kirjutatud näiteks natiivse JavaScripti abil. Seepärast selle teegi on mugav kasutada ning võib kombineerida teiste programmeerimiskeeltega. Lisaks

[1] https://github.com/

jQuery teek omab mitu funktsioone, mis laiendavad natiivse JavaScript'i võimalused. Samal ajal igaüks nendest on kättesaadav jQuery objektina, mis aitab välistada konflikte. Teiste sõnadega öeldes funktsioonidel võib esineda üks ja sama süntaksis, kuid jQuery teek täitab funktsiooni ainult siis, kui ta on ümbritsetud jQuery-objektideks. Näiteks niisugused funktsioonid, nagu: `jQuery.trim, jQuery.merge, jQuery.extend, jQuery.map, jQuery.each` [3]. Veel üks teegi eelis on see, et jQuery annab võimalust kasutada mugavat API, mis töötab enamik brauserites ja võimaldab programmeerijatel seda laiendada: näiteks lisada funktsionaalsust, mis oleks vajalik konkreetse rakenduse jaoks. Kuna teegi eriomaduseks on selle süntaks, esimesest pilgust ta võib tunduda keeruliseks ja ebaselgeks. Vaatamata sellele, jQuery teek sobib nii edasijõudnutele kui ka algajatele programmistidele. Suur osa jQuery tööst toimub sümboli $ abil, mille põhiülesandeks on vajaliku elementi identifitseerimine, globaalsete meetodite alustamine ning välja kutsumine.

Iga jQuery operaator (tegevus) algab DOM elementide valimisest, mis kujutab endast CSS kombinatsiooni. Objekt, mis tuleb välja pärast elementide valimisest (tulemus) kujutab endast DOM-elementide kogumit ja sellega võib töötada nagu massiiviga. Tulemusel on olemas pikkus (`.length()`), ning tema elemendideks võib saada liigipääs indeksite abil(Näiteks: `element[0]`).

Elementi valimise meetodid võib jagada neljaks kategooriaks:

1. meetodid, mis opereerivad kõikide elementidega, mis sobivad antud tingimustele.
2. meetodid, mis tagastavad elemendi väärtust, mis oli leitud esimesena. Näiteks:

   `var width = $('div').width();` Tagastab esimese div elementi laius.
3. meetodid, mis muudavad elementide valimist. Näiteks:

   `$('div#div_id').width(300);` Muudab div'i laiust, kus id on div_id
4. meetodid, mis annavad võimaluse liikuma DOM-puul. Näiteks:

   `$('div').parent();` Tagastab kõik elemendid mis on div'i vanemad.

## 2.2 Underscore.js

Väike javascript'i teek underscore.js või lihtsalt _.js – see on funktsioonide ja utiliitide kogum. Paljud arendajad arvavasid ja arvavad ka praegu,et puhas "javascript" on

ebamugav ja kirjutatud javascriptiga kood on liiga mahukas. Probleemi lahendamiseks 2009. aastal oli loodud teek, mis realiseeris JavaScript'i lisa funktsionaalsust. Underscore.js andis võimaluse lihtsustada tööd massiivide, funktsioonide ja objektidega. Teegi kooseisus on umbes 100 erinevaid funktsioone. Osa nendest aitab lahendada tüüpilised ülesanded nagu töö massiivide või kollektsioonidega. Teine osa aga aitab lahendada palju raskemaid ülesandeid. Näiteks: objektide võrdsuse kontroll, JavaScripti standartiseerimine ning funktsionaalsuse sidumine. Lisaks underscore.js abil saab deligeerida väljakutseid. See tähendab, et kui kood töötab brauseris, kus mingi funktsionaalsus on juba realiseeritud, siis script kutsub välja ja kasutab selle "natiivset" funktsionaalsust (Näiteks funktsioonid mis olid realiseeritud brauseris: indexOf, filter, map, forEach). Lisaks Underscore.js eriomaduseks on see, et teek ei muuda javascript'i sisse ehitatud funktsioonid vaid täinendab neid. See annab võimaluse kasutada underscore.js koos teiste teekidega ja vältida konflikte.

## 2.3 Lodash.js

JavaScripti framework ehk teek Lo-Dash.js on sarnane underscore.js teegiga. Sarnaus seisneb selles, et süntaksis on sama ning teegil on samad funktsioonid töö lihtsustamiseks. Kuid on olemas ka erinevused. Lo-dash on parandatud teek. Arvatakse, et see teek on palju jõukam, töötab kiiremini ja seal on parandatud mõned bugid. Pealegi, Lo-dash annab uued võimalused töö jaoks. Näiteks Lo-dash toetab AMD (asünkroonne mooduli määramine) [4]. Kui rakenduse arendus toimub JavaScripti abil moodulite kaupa, ilmuvad mõned probleemid: muutujad on vaja teha globaalseteks, selleks et neid võiks kasutada rakenduse igas moodulis ning moodulite vahel on vaja seoseid panna. Sellise probleemide lahendamisega aitab AMD. Lodash koosneb mooditest, ja see on veel üks selle teegi eelis. See annab võimaluse koostada teegi ainult vajalikkest funktsioonidest ja järelikult vähendada teegi kaalu.

## 2.4 MooTools

MooTools - see on objekt-orienteeritud JavaScript'i raamistik, mis oli loodud septembris 2006. aastal ja mille autor on Valerio Proietti. Tema tahtis teha teek, mis võiks pärida natiivse JavaScripti võimalused, aga samal ajal teeks lihtsamaks töö DOM elementidega. Tulemuseks on MooTools - teek, mille abil saab arendada jõukad ja paindlikud

veebirakendused. Teek põhineb Moo.fx plugin'il ning lisaks sisaldab plugin nimega More.js, mis võimaldab manipuleerida massiivide, reade ning kuupäevadega. Lisaks MooTools teegil on palju eeliseid: lihtsustatud funktsioonid DOM elementide muutmiseks, lisamiseks ja kustutamiseks; täiendavad komponendid, mis võimaldavad arendada Flash-rakendused; funktsioonid, mis lihtsustavad tööd CSS, Ajax ning standaartsete JavaScript objektidega.

MooTools on jaotatud paljudeks väikseteks osadeks (mooduliteks), mis on väga suur pluss. See annab võimaluse lisada ja kasutada ainult vajaliku teegi osa, teiste sõnadega, puudub vajadus lisada scriptile terve teek. Iga konkreetse rakenduse jaoks kasutatakse enda välja valitud funktsioonide kogum. See on mugav ja teeb rakenduse "kergemaks".

Veel üks teegi eelis - võimalus lokaliseerida MooTools'i inglisekeelne süntaks ehk muuta inglise keel üks kõik millele keelele. See teeb programmeerimist MooTools'i abil veelgi lihtsamaks.

## 2.5 Native JavaScript ehk 'puhas' JavaScript

JavaScript on kõrgetasemeline prototüüpidele orienteeritud programmeerimiskeel. Tavaliselt seda panevad olemasoleva koodi sisse, selleks et tagada juurdepääs rakenduse objektidele ning lisada interaktiivsust veebilehedele. Vaatamata sellele, et JavaScript on objektorienteeritud programmeerimiskeel, ta erineb teistest sama tüüpi keeltest. Erinevusi on mitu:

1. JavaScript kasutab prototüüpidele põhinevat programmeerimist. See tähendab, et keel ei kasuta klassi, ning pärimine toimub objektide kloonimise abil.
2. JavaScriptil on olemas funktsionaalse programmeerimise omadused.

Need omadused teevad JavaScripti paindlikuks ning mugavaks programmeerimiskeeleks nii professionaalsetele kui ka algajatele programmeerijatele. Muidugi on olemas ka puudused:

1. Puudub liides andmebaaside ja veebiserverite jaoks.
2. Puudub süsteem, mis jälgib sõltuvust komponendide vahel ja automaatelt neid paigaldab.

Süntaks:

1. Kõik indentifikaatorid sõltuvad registrist: sõnad mis on kirjutatud väikese tähega erinevad sõnadest mis on kirjutatud suure tähega.

2. Muutujad ei tohi alata numbriga, aga numbrid võib kasutada muutuja sees.

## 2.6 Benchmark.js

Antud töö eksperimentide jooksul kõik koodi täitmise ajamõõdud olid tehtud JavaScript teegi abil Benchmark.js. See teek annab võimaluse panna taimerid ning saada statistilised tulemused, teiste sõnadega benchmark.js kasutatakse koodi täitmisekiiruse testimise jaoks. On olemas mitu meetodeid, kuidas on võimalik luua teste. Igal meetodil on olemas oma eelised ja puudused:

1. Kõige levinum meetod - kood, mis on vajalik kontrollida pannakse tsükli sisse. Pannakse ka mingi number, mis näitab kui palju korda funktsioon peab olema täidetud. Niisugust meetodid kasutavad paljud raamistikud, aga selles on oma miinused. Kuna arvutid ning brauserid pidevalt arenevad, funktsioonide täitmine toimub nii kiiresti, et sellise meetodiga kirjutatud testid lõppude lõpuks annavad tulemuseks 0 [5].

```
var time;
var startDate = new Date; // Saame käesolev kuupäev ja aeg
var iterations = 10;      // Määrame iterratsioonide arvu
while (iterations--) {
 // Kood, mis me soovime kontrollida.
}
time = new Date - startDate; // Tulemus näitab aega millisekundides,
mis oli vajalik koodi täitmiseks.
```

Joonis 1. Esimese meetodi näidiskood.

2. Teine võimalus kontrollida töö kiirust - kindlaks määrada aeg, mille jooksul operatsioonid jäävad täitma. Puuduseks on see, et ühe ja sama operatsiooni täitmisel, tulemus alati erineb, sest operatsioonide täitmine sõltub paljudest faktoritest, näiteks protsessidest mis töötavad taustal [5]. Seepärast on vaja testi käivitada mitu korda ja on võimalik saada ainult keskmist tulemust. Kuid on olemas ka oma eelis: inimesele on palju lihtsam valida fikseeritud aeg, mitte operatsioonide arvu.

```
var operations_per_second, // Operatsioonide arv sekundis
    period,                // Kui palju aega võtab üks operatsioon
    startDate = new Date,  // Käesolev kuupäev ja aeg
    iterations = 0;        // Iteratsioonide arv
do {
  // Kood, mis me soovime kontrollida.
  iterations++;
  time = new Date - startDate;
} while (time < 10000);
time /= 1000;              // teisendame millisekundid sekunditeks
period = time / iterations;
operations_per_second = 1 / period;
```

Joonis 2. Teise meetodi näidiskood.

Sellised probleemid ja puudused võiks lahendada funktsioone kompileerimisega ja tsüklide lahtikeeramisega. Kui panna testi tööle mitu tuhat korda, see loob suurt teksti rida ning kompileerib seda funktsiooni. Selline viis võtab kasutamisele palju mälu ning töö kiirus langeb.

3. Töörist nimega JSLitmus ühendab ülaltoodud meetodeid. Ta kasutab esimene meetod selleks, et korrata testi tsüklis n-korda, aga samal ajal annab võimaluse suurendada tsüklide arvu. [5] Tsüklide arv kasvab seni, kui testi täitmise aeg saab minimaalseks, nagu teises meetodis. Vaatamata sellele, et JSLitmus lahendab probleemid, mis esinevad esimeses meetodis, töörist ei lahenda probleemid teisest meetodist. Kõigest tulemustest töörist valib kolm kõige kiiremaid teste, kuid tulemustel jääb suur ebatäpsus.

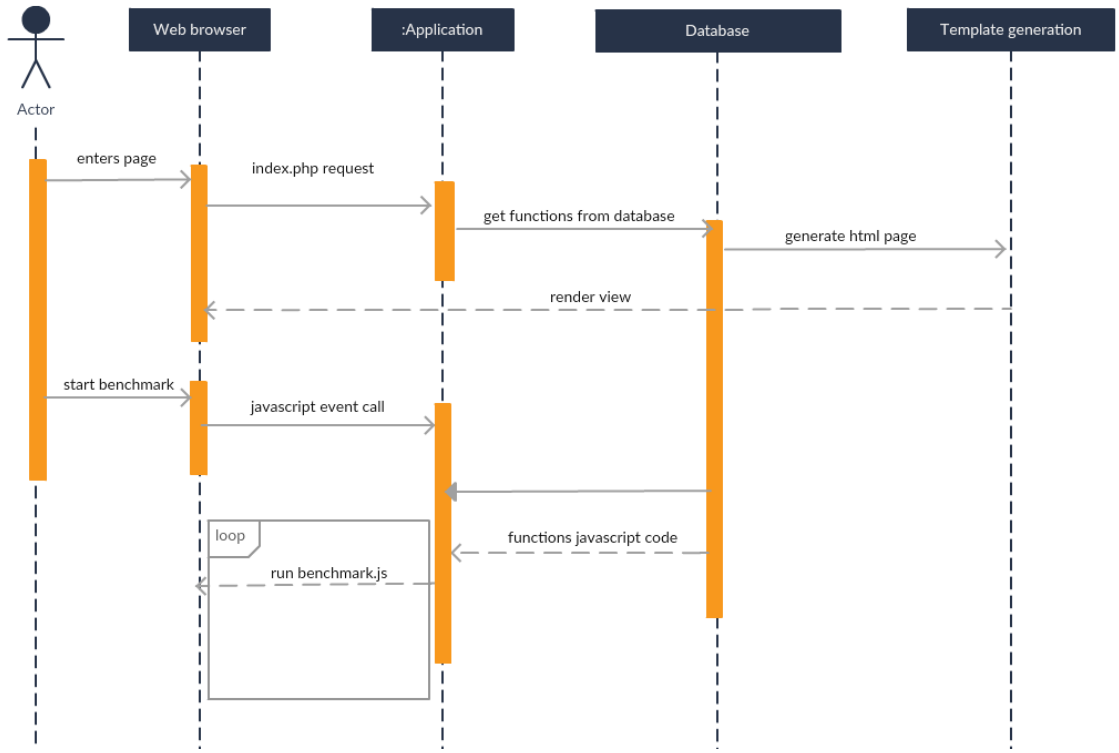
Benchmark.js teegi loojad proovisid ühendada kõike meetodite parimad küljed. Selleks, et saada maksimaalselt täpne tulemus teek töötab lokaalsete meetodite ja muutujatega. Iga testi jaoks teek saab funktsiooni ja paneb seda tsükli sisse (nagu esimesel meetodil) ja tsüklis see töötab antud aja jooksul. Testid korratake mitu korda ja saadakse statistilised andmed.

# 3 Rakendus

## 3.1 Arhitektuur

JavaScripti teekide jõudluse uurimiseks ja võrdlemiseks oli kirjutatud rakendus, mille abil saab mõõta funktsioonide töökiirust konkreetses teegis. Rakendus on jagatud kaheks osaks: backend ja frontend. Backend on kirjutatud programmeerimiskeeles PHP. Backendi funktsionaalsus vastutab töö eest andmebaasiga. Seal toimub ühendus andmebaasiga, funktsioonide nimekirja salvestamine andmebaasi ning andmete väljavõtmine. Andmebaasina kasutatud SQLite, kuhu säilitakse funktsiooni ning koodi mis kasutatakse töö kiiruse testimiseks.

Frontend on kirjutatud HTML ning JavaScripti abil. Lisaks oli kasutatud jQuery ja Lodash teegid. Kõik ajamõõdud on tehtud Bencmark.js teegi abil. Selleks, et teha mõõtmise tulemused maksimaalselt täpseks ning vältida konflikte teekide vahel, iga teegi katsed toimuvad omas keskkonnas ja on „isoleeritud“ teineteist. Isolatsioon on realiseeritud HTML *iframe* märgendi abil, mis töötab konkreetse teegiga, laeb testandmed ja täidab vajalikud skriptid testimiseks.



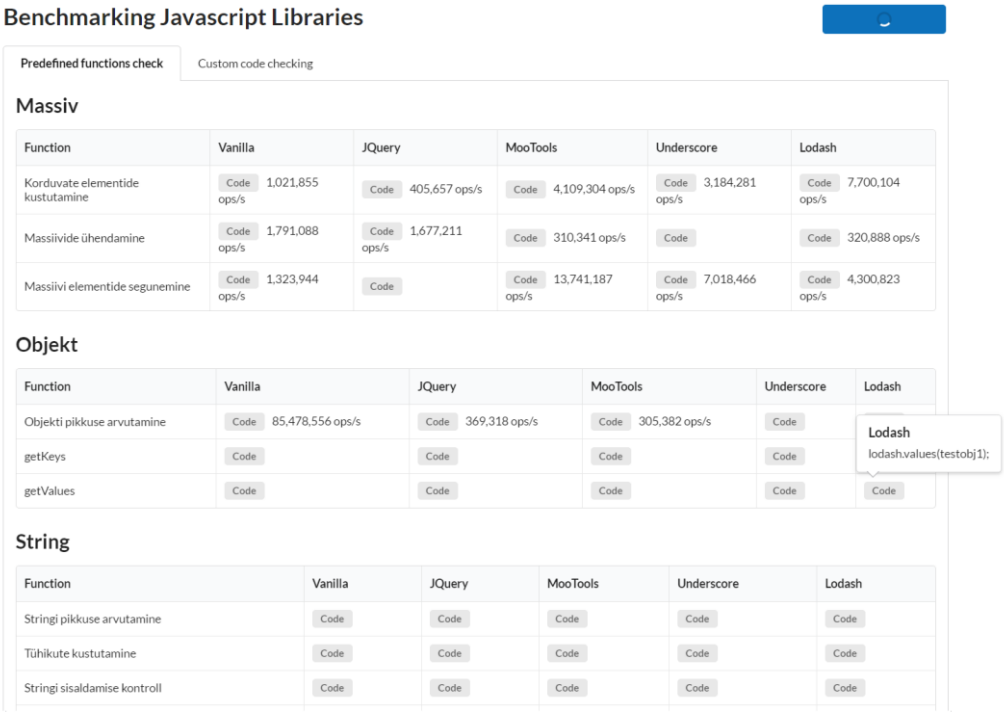

Joonis 3. Rakenduse arhitektuur

Rakendus on loodud keskkonnas PHPstorm. Rakenduse testimiseks oli paigaldatud *Ampps*-server [7]. *Ammps*-server toetab tööd PHP, Python ja Perl programmeerimiskeeltega. Komplekt on cross-brauseriline ehk töötab erinevates brauserites sarnaselt, mis toetab antud töö ühe eesmärki – kontrollida teekide jõudlust erinevates brauserites. Lisaks on seda võimalik kasutada Windows, Linux ja Mac platvormides. Ammps lokaalse serverisse on sissehitatud Apache veebiserver versiooniga 2.4.17 ning PHP versiooniga 7.0.0, mis oli kasutatud rakenduse arendamisel.

Rakenduse kasutajaliidese kujundus on loodud framework'i SemanticUI abil. Lisaks oli kasutatud CSS.

Selleks, et vältida HTML ja PHP koodi segamist šabloonides, oli eraldi loodud *engine.php* fail, mis aitab organiseerida koodi ja lihtsustab koodi lugemist.

Rakenduse esilehel võib näha tabelit, mis koosneb funktsioonidest ja JavaScripti teekidest. Igas tabeli ruudus on esitatud kood, mida kasutakse jõudluse testimiseks. Lisaks on lehel olemas nupp, millele vajutades käivitataks funktsioon testimiseks. Testimise tulemuseks on tabel, kus on näidatud kui palju antud funktsiooni operatsioone sekundis on jõudnud teha teek. Juhul, kui testimises oli viga, ilmub vastavas ruudus 0. Kui antud teegis (välja arvatud puhas JS) puudub vajalik funktsioon, jääb vastav ruut tühjaks.




Joonis 4. Rakenduse liides

Loodud rakenduse kood asub GitHub [1]repositooriumis ning on kättesaadav aadressil: https://github.com/KristinaNa/JavaScript-libraries-benchmark

## 3.2 Funktsioonid testimiseks

Teekide jõudluse võrdlemiseks said valitud erinevad tegevused massiivide, objekti, ridade ning DOM-elementidega. Tegevuste valimise kriteeriumiks oli võimalus neid realiseerida igas valitud teegis. Juhul, kui puhtal JavaScript'l puudus vajalik sisseehitatud funktsioon, oli see asendatud enda poolt kurjutatud funktsiooniga, mis täiendab samat ülesannet. Iga tegevuse jaoks valitud funktsioonid võivad süntaktiliselt erineda, aga saavutavad ühe ja sama eesmärgi. Järgnevalt on esitatud tegevused, mis olid valitud teekide testimiseks ning teekide jõudluse võrdlemiseks.

Tööks massiividega valitud funktsioonid on selle töö autori subjektiivsel arvamusel reaalelus kõige laiemalt levinud ja kasulikud tegevused. Rakenduse kirjutamisel sai võetud arvesse, et iga järgmise testi alguses massiiv võib suurendada lõpmatuseni, mis rikub kiiruse mõõtmise tulemusi. Selle probleemi vältimiseks oli otsustatud enne testi kloonida muutujad, mis sisaldas massiivi.

Tabel 1. Massiivide testimiseks kasutatud funktsioonid.

| **Tegevus** | **Puhas JS** | **jQuery** | **MooTools** | **Underscore** | **LoDash** |
|---|---|---|---|---|---|
| Korduvate elementide kustutamine. | Kirjutatud funktsioon | .unique | .unique | .uniq | .uniq |
| Elementide segunemine | Kirjutatud funktsioon | | .shuffle | .shuffle | .shuffle |
| Elementidele väärtuse omastamine (map) | .map | .map | .map | .map | .map |
| Filtreerimine | .filter | .filter | .filter | .filter | .filter |
| Pikkuse arvutamine | .length | .length | | .size | .size |
| Väiksema elemendi otsimine | .min | | .min | .min | .min |
| Suurema elemendi otsimine | .max | | .max | .max | .max |

[1] https://github.com

| | | | | | |
|---|---|---|---|---|---|
| Elemendi sisaldamise kontroll | indexOf | .inArray | .contains | .contains | .includes |
| Suvalise elemendi saamine (random) | Kirjutatud funktsioon | | .getRandom | .sample | .sample |

Antud töö raames nimetatakse objektideks assotsiatiivsed massiive, millistel on võimalik hoida andmeid kujul võti-väärtus (*key-value*).

```
var objekt = {key: value, key1: value1}
```

Joonis 5. JavaScript objekti näide

Objektide testimiseks valitud tegevused  kontrollivad objekti pikkuse, paaridest võtmete ja väärtuste saamist.

Tabel 2. Objektide testimiseks kasutatud funktsioonid.

| **Tegevus** | **Puhas JS** | **jQuery** | **MooTools** | **Underscore** | **LoDash** |
|---|---|---|---|---|---|
| Võtmete saamine | .keys | .map | .getKeys | .keys | .keys |
| Väärtuste saamine | .values | .map | .getValues | .values | .values |
| Pikkuse arvutamine | .length | .length | .getLength | .size | .size |

Tabel 3. Ridade testimiseks kasutatud funktsioonid.

| **Tegevus** | **Puhas JS** | **jQuery** | **MooTools** | **Underscore** | **LoDash** |
|---|---|---|---|---|---|
| Pikkuse arvutamine | .length | .length | | .size | .size |
| Tühikute kustutamine | .trim | .trim | | | .trim |
| Sisaldamise kontroll | .indexOf | | .contains | | .includes |
| String'i osa vahetamine | .replace | | .substitute | | .replace |
| CamelCase lause | Kirjutatud funktsioon | | .camelCase | | .camelCase |
| Suur algustäht | Kirjutatud funktsioon | | .capitalize | | .capitalize |
| Tegema reast numbri | .parseInt | | .toInt | | .parseInt |

DOM-elementide testimiseks funktsioonide realiseerimiseks Lodash ja Underscore teegide jaoks oli kasutatud täiendressursid: *lodash.dom-traverse* [8] ja *underscore.dom* [9]. Testimiseks olid valitud funktsioonid, mille abil saab saada elemendi, kasutades tema:

Tabel 4. DOM-elementide testimiseks kasutatud funktsioonid.

| **Tegevus** | **Puhas JS** | **jQuery** | **MooTools** | **Underscore** | **LoDash** |
|---|---|---|---|---|---|
| Identifikaator | .getElementById (#id) | $(#id) | $$(#id) | .qsa(#id, document) | .$(#id, document) |
| Klassi nimi | .getElement ByClassName (.class) | $(.class) | $$(.class) | .qsa(.class, document) | .$(class, document) |
| Atribuut | .querySelector() | $(selector[1]) | $$(selector) | .qsa(selector, document) | .$(selector, document) |
| Elemendi lapsed | $().childNodes() | $() .children() | $$() .getChildren() | .qsa(children, document) | .$(children, document) .siblings |

[1] #parent .child[data-search='parent']

# 4 Teekide jõudluse võrreldus

Selles peatükis kirjeldatakse käesolevas töös läbiviidavat eksperimenti.

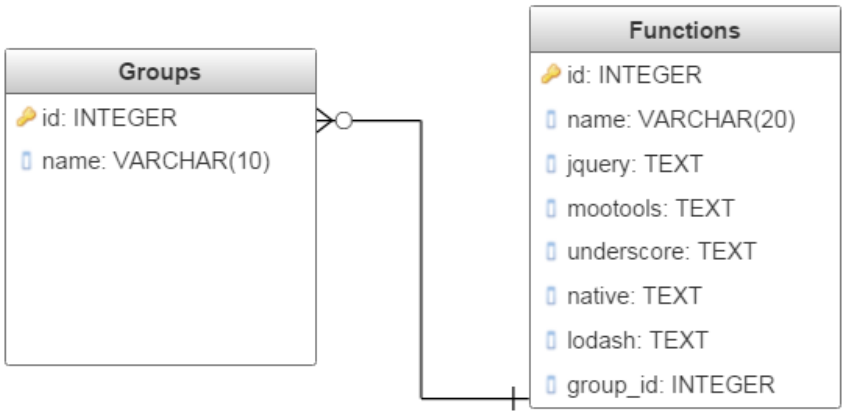
## 4.1 Eksperimendi eesmärgid

Käesoleva eksperimendi eesmärkideks on:

1. Uurida varem valitud teekide jõudlust
2. Võrrelda funktsiooni täitmisekiirust varem valitud teekides.
3. Võrrelda teekide jõudlus erinevates brauserites

## 4.2 Eksperimendi kirjeldus

Kõik katsed olid tehtud loodud rakenduse abil, mis on kirjeldatud punktis 3.1. Selleks, et vältida andmebaasi denormaliseerimist eksperimendi käigus oli andmebaasis loodud 2 tabelit nimidega *groups* ja *functions*. Tabel *groups* sisaldab 2 välja: *id* - identifikaator ja *name*, mis sisaldab endas tegevuste kategooriate nimetusi. Teine tabel sisaldab 7 välja, kuhu kuulub identifikaator, tegevuse nimetus ning 5 välja, kuhu säilitatakse erinevate teekide funktsioone. Tabelid on omavahel seotud identifikaatori ja välja *group_id* kaudu.




Joonis 6. Andmebaasi struktuur.

Enne katsete algust olid kõik tegevuste andmed ja funktsioonid andmebaasi sisse toodud. Katsed said läbiviidud arvutil Lenovo ideapad 510S, mille tehnilised andmed olid järgmised: Intel Core i5-6200U, CPU 2,30 GHz, RAM 8 GB, 62-bitine operatsioonisüsteem, Windows 10 Home. Selleks, et teha eksperimenti objektiivsemaks, sai otsustatud läbi viia katsed erinevates brauserites ning valiti neist kolm:

1. Google Chrome, versiooniga 54.0.2840.99 (64-bit).
2. Microsoft Edge 38.14393.0.0
3. Mozilla Firefox 50.0.1

Brauserid olid valitud populaarsuse järgi ehk siis sai valitud kolm brauserit, milliste kasutamine on maailmas kõige laiemalt leevinud [10]. Otsus teha katsed erinevates brauserites sai võetud vastu seetõttu, et iga brauser sisaldab endas JavaScript-mootorit, mis vastutab JavaScript koodi täitmise eest. Kuna igas brauseris on enda omapärane mootor oma karakteristikutega – siis see võib mõjuda ka koodi täitmisekiirusele ning järelikult avaldada ka mõju eksperemendi tulemustele.

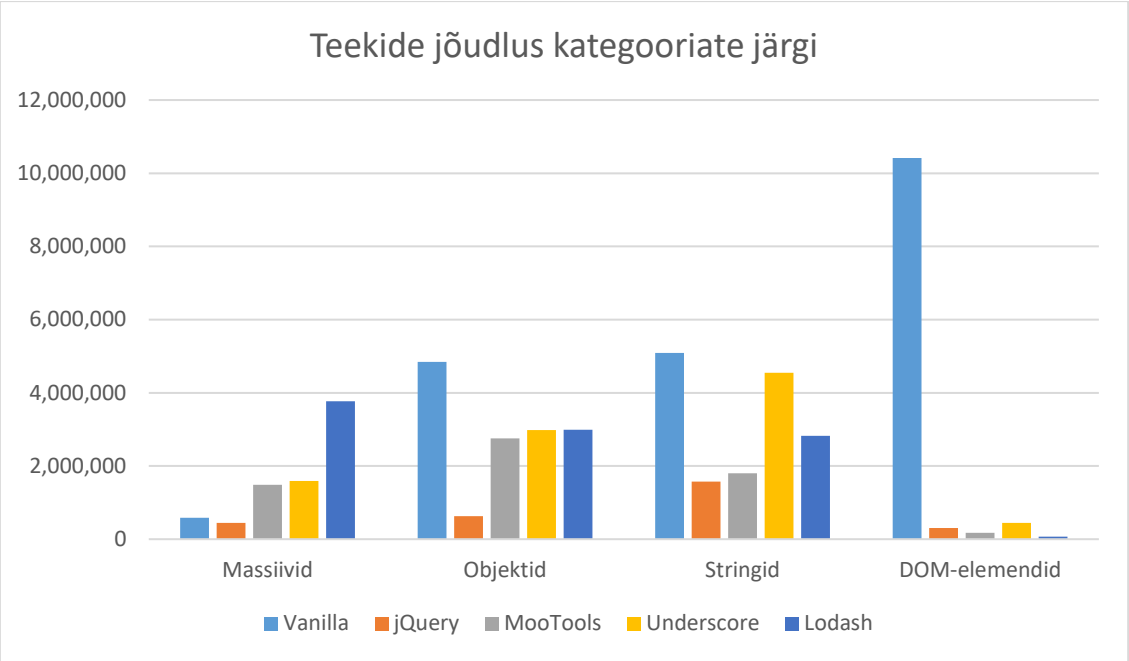
# 5 Eksperemendi tulemused ja analüüs

Eksperemendi käigus sai igas brauseris tehtud 5 katset ning tulemuseks sai väljaarvutatud keskmine väärtus. Igas brauseris said operatsioonid jagatud neljaks grupiks: töö massiividega, objektidega, ridadega (*String*'idega) ja DOM-elementidega. Iga teegi tulemused määratud grupis on esitatud mediaanväärtusena ehk tulemuseks on keskmise liikme väärtus. See on tehtud andmete analüüsiks ja graafiku visuaalse taju lihtsustamiseks. Lisaks on järgmistes paragrahvides analüüsi käigus operatsioonide täitmise resultaadid esitatud protsendilises seoses. Sellisel juhul 100%-ks on võetud konkreetselt antud grupis kogu täidetud operatsioonide summa.

Järgnevalt on esitatud iga brauseri JavaScript-mootori lühikirjeldus, antud brauseris tehtud eksperimendi tulemused ja nende analüüs.

## 5.1 Google Chrome (v. 54.0.2840.99)

Nagu sai varem juba mainitud on igas brauseris olemas oma Javascript-mootor. Google Crome kasutab mootorit nimega V8. Ta on esitatud open-source projektina ning on kirjutatud C++ programmeerimiskeeles. V8 võib kasutada Windows, Mac ja Linux operatsioonsüsteemidega. Mootori abil JavaScript kood kompileeritakse ja täitakse, lisaks aitab mootor õigesti jaotada mälu objektide vahel ning perioodiliselt kustutab mälust objektid, mida brauser enam ei kasuta. Katsete ajal oli kasutatud V8 versioon 5.4.500.41. *Lisa 1* sisaldab tabelid, kus on esitatud katse tulemused. Järgnevalt on esitatud Google Crome katsete tulemused graafiku kujul. Mugavusest lähtuvalt on erinevad tegevused ja funktsioonid massiivi, Stringi, objekti ja DOM – elementidega vaadeldud eraldi. Iga gruppi väärtused on esitatud mediaanväärtusena.




Joonis 7. Teekide jõudlus kategooriate järgi Google Chrome brauseris.

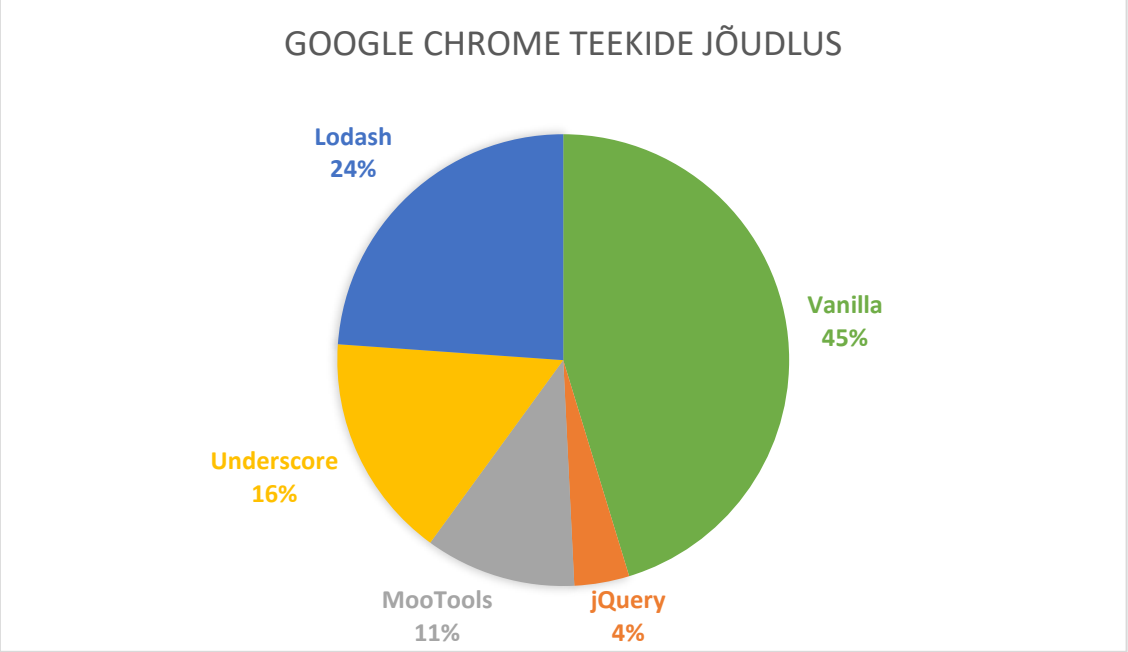
Google Chrome brauseris tehtud katsed massiividega näitasid, et kõige rohkem operatsioone sekundis jõuab teha Lodash teek. Kui esitada selle teegi resultaate protsendilises seoses, siis tuleb välja, et Lodash on täitnud umbes 47.8% kõikidest operatsioonidest, mis oli tehtud töötades massiividega. Selline tulemus näitab, et selles operatsioonide gruppis Lodash on kõige kiirem teek. Halvema tulemuse on näidanud jQuery, mille tulemuseks on 5.6%, mis on vähem kui pool miljoni operatsiooni sekundis võrreldes teiste teekidega (välja arvutatud puhas JavaScript) milliste tulemusteks on poolteist miljoni ja rohkem. Underscore ja MooTools teegid näitasid umbes üht ja sama tulemust, mis olid vastavalt 19.2% ja 19.8% ja seega peaaegu võrdsed. Vaatamata sellele, et kiiruse mõõtmisel need teegid paiknesid teisel kohal, moodustas tulemuste erinevus Lodash teegiga umbes 28%.

Teistes gruppides parima tulemuse näidanud kood sai puhta JavaScripti abil.

Objektide grupis jQuery on kõige aeglasem teek. Selle operatsioonide arv moodustab ainult 4% võrreldes teiste teekidega. MooTools, Underscore ja Lodash teegid tegid tööd objektidega umbes sama kiirusega ning nende tulemuseks on umbes kolm miljonit operatsiooni sekundis, mis vastavalt protsentides võrduvad 19.4%, 20.9% ja 21,1%. Lisaks puhta JavaScripti heale tulemusele sai Underscore teek töötades ridadega (*String*'idega), umbes 28.7% kõikidest operatsioonidest ridadega. jQuery ja MooTools teegid näitasid halvima tulemuse – 9.9% ja 11.3%.

Operatsioonid DOM-elementidega on näidanud kõike ootamatumaid tulemusi. Nagu graafikult võib näha, puhas JavaScript töötab kõige kiiremini ja täidab ühe sekundi jooksul kümme ja pool miljoni operatsiooni. Niisugust tulemust ei olnud mitte üheski grupis. Kui vaadelda tulemusi protsendilises seoses, siis puhas JavaScript koostab 91% ja kõik teised teegid ei ületa 4%. Vaatamata sellele, et võrreldes puha JavaScriptiga teised teegid töötades DOM-elementidega näitasid umbes sarnaseid tulemusi, sai kõige halvema tulemuse  Lodash teek. Ühe sekundi jooksul teek täitis vaid 67 tuhat operatsiooni. See on kõige madalam resultaat ja protsendilises seoses moodustab 0.6%.

Enne sai vaadeldud antud brauseris eksperemendi resultaate igas kategoorias eraldi. Nüüd vaadates Google Chrome brauseris tehtud eksperimente ja nende tulemusi on võimalik analüüsida kõiki 4 kategooriaid koos, üldistada resultaate, teha järeldusi ja vastata küsimustele, millise teegi jõudlus on kõige kõrgem. Järgnevalt on esitatud antud brauseri üldtulemused: täidetud operatsioonide arv, mis on näidatud protsentides.




Joonis 8. Teekide jõudlus Google Chrome brauseris.

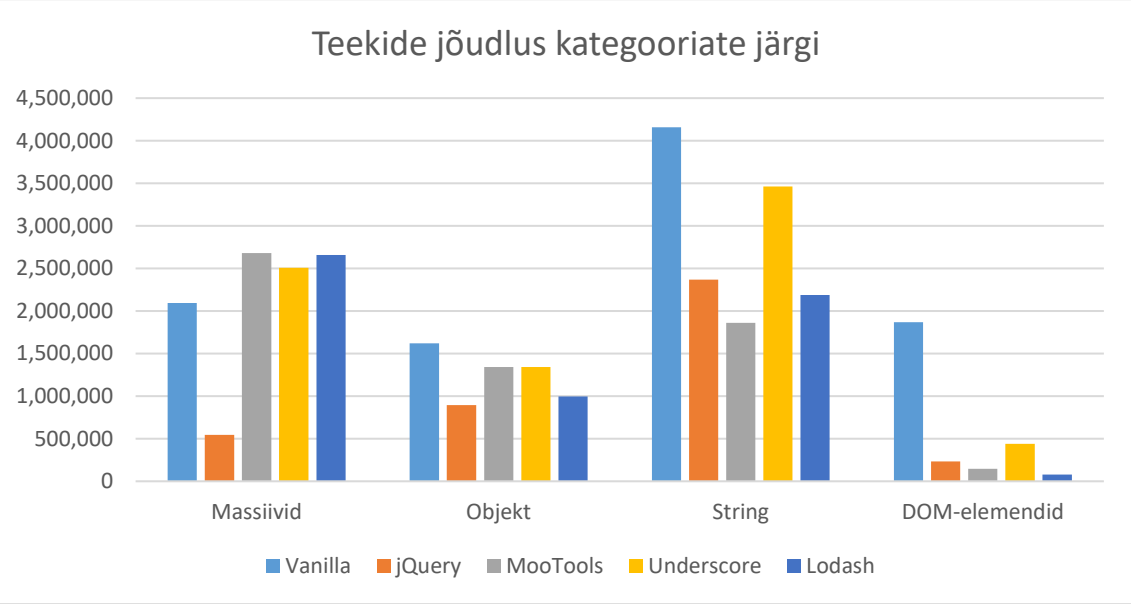
Nagu graafikust näha on suurema operatsioonide arvu  täitnud puhas JavaScript. Sellest on võimalik teha järelduse, et ilma teekideta kirjutatud JavaScript koodi täitmine võtab vähem aega, mis tähendab, et puhta JavaScript'i jõudlus on suurem ja võrreldes teistega on kõige parem. Teisel kohal on Lodsh teek. Kõige halvema resultaadi antud brauseris on

näidanud jQuery. Lodash teek sai teise kohta. Kolmandal kohal on Underscore teek ja neljandal kohal on MooTools teek. Kõikide teekide vahel on hästi suur kiiruse erinevus.

## 5.2 Microsoft Edge (v. 38.14393.0.0)

Brauser Microsoft Edge kasutab JavaScript-mootorit nimega Chakra. See mootor erineb Google Chome V8 mootorist. Chakra arhitektuur lubab kompileerida koodi JIT (*just-in-time* ehk täpselt samal ajal) ja kiirendada koodi täitmist. Mootor loeb JavaSctipt koodi, analüüsib selle süntaksist ja kompileerib seda ilma optimisatsioonita [11].

*Lisa 2* sisaldab tabelid, kus on esitatud katsete tulemus Microsoft Edge brauseris. Järgmiselt on näidatud tulemused graafiku kujul:

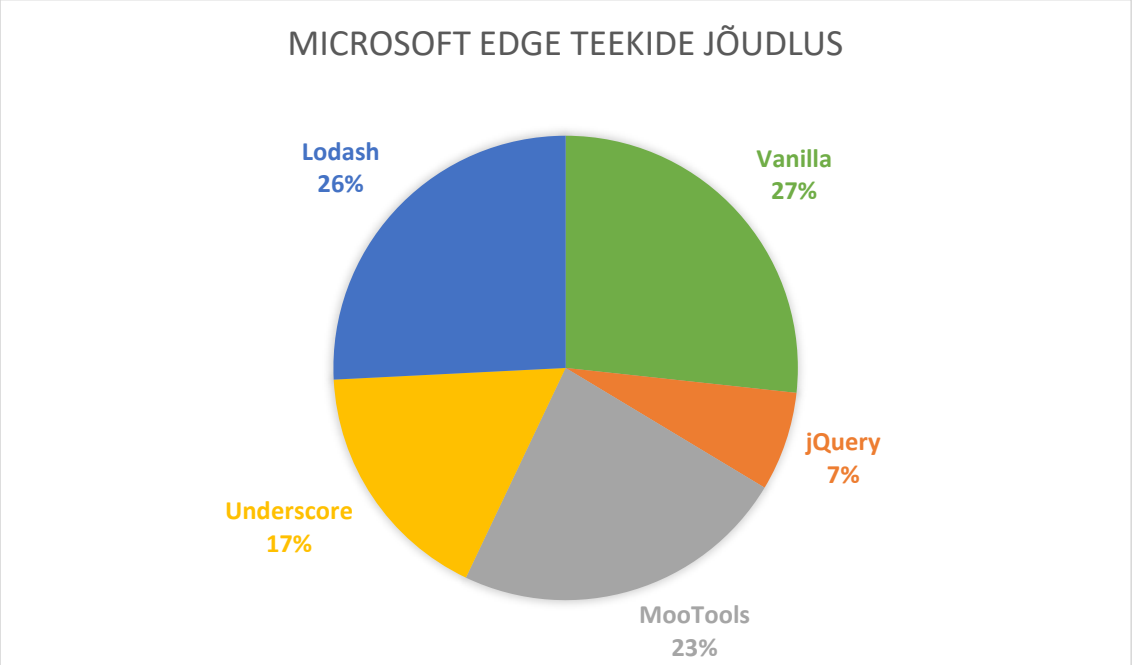



Joonis 9. Teekide jõudlus kategooriate järgi Microsoft Edge brauseris.

Kui vaadata tulemustele Microsoft Edge brauseris üldiselt, siis võib öelda, et kokku teegid täitsid vähem operatsioone sekundis, kui Google Chrome brauseris. Töötades massiividega said parima tulemuse Lodash ja MooTools teegid, nende tulemused on peaaegu sarnased: 25.4% ja 25.6%. Järgmisena tulemuste järgi tuleb Underscore, mis näitas kõrget operatsioonide arvu (23.9%), kuid erineb esimestest kahest umbes 1.5% võrra. Kõige madalama resultaadi näitas jQuery. Kui võrrelda selle teegi tulemust kogu täidetud operatsioonide arvuga massiivides ja esitada saadud jQuery teegis resultaat protsentides, siis tulemuseks on 5.2%.

Gruppides objekti, ridade ja DOM-elementidega parima tulemuse näitas kood, mis oli realiseeritud puhta JavaScript'iga. Grupis objektidega suurt vahet teekide vahel ei ole. Väikse väärtusega resultaadi näitasid Lodash ja jQuery teegid – 16.1% ja 14.4%. MooTools ja Underscore tulemused olid sarnased – umbes 21.7%. Töös ridadega kõik teegid said erinevad tulemused ja ei olnud sarnast resultaati. Halvima tulemuse sai MooTools teek, mis protsendilises seoses koostab umbes 13%, mis on suhteliselt hea tulemus. Natuke parema resultaadi said Lodash ja jQuery teegid, mis koostavad vastavalt 15.6% ja 17%. Nagu Google Chrome, Microsoft Edge brauser töötades DOM-elementidega näitab, et kõige kiiremini koodi täidab puhas JavaScript. Kõik teised teegid jäävad puhtale JavaScriptile alla. Esitades antud grupis tulemused protsentides, saame, et puhas JavaScript täidab 67.6% kõikidest operatsioonidest. Teisel kohal on Underscore, millele kuulub 15.9%. Järgmisena tuleb jQuery, mis täidab 8.4% ja viimasena on MooTools (5.3%) ja Lodash (2.8%) teegid.

Järgnevalt on esitatud Microsoft Edge brauseris vaadeldud teekide lõpptulemused, mis on saadud 4 kategoorijate üldistusel.




Joonis 10. Teekide jõudlus Microsoft Edge brauseris.

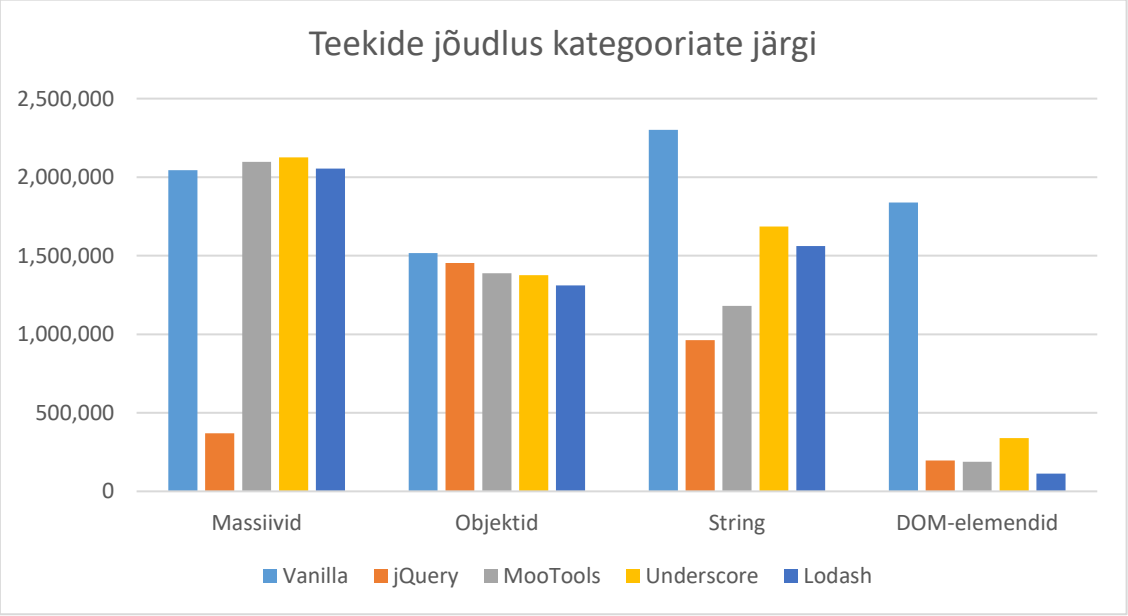
Nagu graafikust on näha, sai parima tulemuse kood, mis oli täidetud puhta JavaScript'i abil. Kuna ühe sekundi jooksul täidetud operatsioonide arv on suurim, mis ka tähendab, et JavaScripti kasutamine ilma teekideta omab suuremat jõudlust. Teisel kohal on Lodash

teek, mis ei jäänud kaugele maha ja mille tulemused on sarnased puhta JavaScriptiga. Kolmandal kohal on MooTools teek, mis näitas ka suurt operatsioonide arvu ja head täitmisekiirust. Järgmisena tuleb Underscore ja viimasel kohal on jQuery teek kõige väiksema operatsioonide arvuga, mis said täidetud ühe sekundi jooksul. See tähendab, et antud brauseris jQuery näitas kõige väiksema jõudlust võrreldes teiste teekidega.

## 5.3 Mozilla Firefox (v. 50.0.1)

Mozilla kasutab JavaScript-mootorit nimega SpiderMonkey ja see on maailma esimene programmeerimise JavaScript-mootor. SpiderMonkey on kirjutatud C++ programmeerimiskeeles ja ühendab endas kompilaatori, interpritaatori, dekompilaatori ja kasutamata objektide kustutamist standartsete klassidega. Mootorit sisse ehitatakse programmi või rakendusesse, seal kus on vajalik JavaScript koodi täitmine. Kõige populaarsem nendest on Mozilla Firefox, aga SpiderMonkey kasutab näiteks ka Yahoo! otsingussüsteem.

*Lisa 3* sisaldab tabelid katsete tulemustega, mis on saadud Mosilla Firefox brauseris. Järgnevalt on näidatud tulemused graafiku kujul:




Joonis 11. Teekide jõudlus kategooriate järgi Mozilla Firefox brauseris.

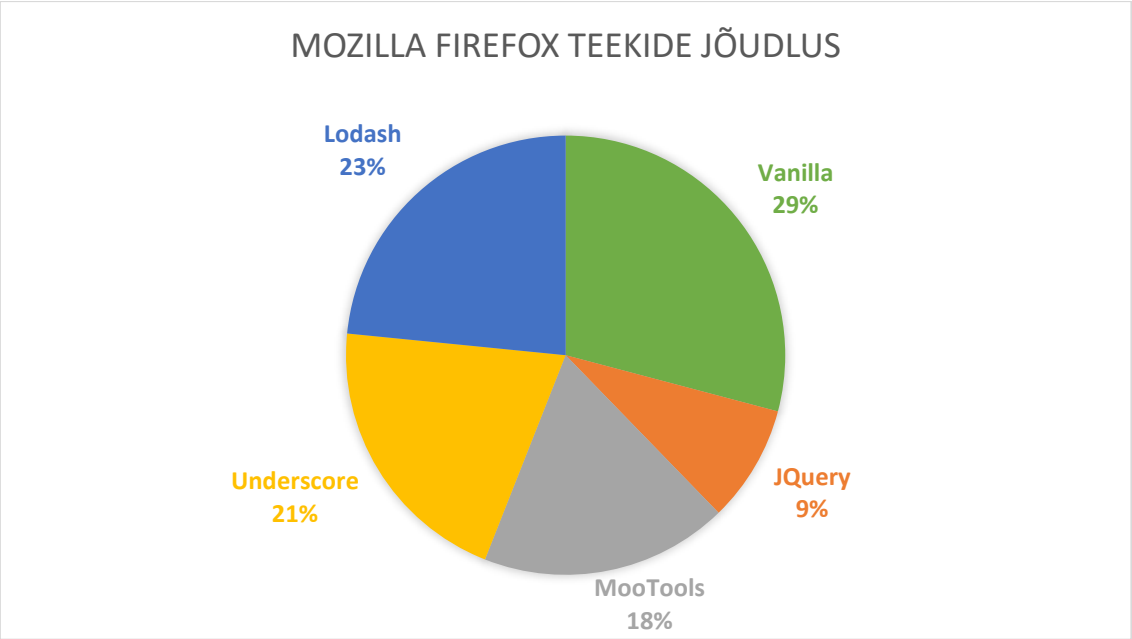
Katsed, mis olid tehtud Mozilla Firefox brauseris näitavad, et grupis, kus sai kontrollitud töö massiividega said kõik teegid peale jQuery enam-vähem sarnased tulemused, kuigi parima tulemuse sai Underscore teek. Kui esitada tulemused antud grupis protsendilises seoses, siis Underscore täitas 24.4% ülesannetest massiividega, järgmisena tuleb MooTools teek (24.2%), kood täidetud puhta JavaScripti abil (24%) ja neljandal kohal on Lodash teek (23.7%). Kuna vahe teekide tulemuste vahel on väga väike, umbes 0.2% võib teha järelduse, et Mozilla Firefox brauseris vaadeldud teegid töötavad massiividega sama kiirusega. Halvema tulemuse näitas jQuery teek, umbes 4%.

Töötades objektidega näitasid kõik teegid suhteliselt häid tulemusi. Kõige suurema operatsioonide arvu täitas kood, mis oli kirjutatud puhta JavaScripti abil, umbes 22% kõikidest operatsioonidest, mis oli tehtud antud grupis. Teisel kohal on jQuery teek, mis täitis 20.6% operatsioonide arvust. Järgnevalt tulevad MooTools (19.7%) ja Underscore teegid (19.5%), mille tulemused erinevad teineteist ainult 0.2% võrra ehk võib öelda, et need 2 teeki töötavad objektidega sama kiirusega. Väiksema operatsioonide arvu täitas Lodash teek, kuid ta koostub 18.6% ja teistest teekidest kaugele maha ei jää.

Grupis, kus sai kontrollitud teekide töö ridadega (*String*'idega) said teegid erinevad resultaadid. Parima tulemuse sai puhas JavaScript (30%). Teisel kohal on Underscoor teek, see koostas protsendises umbes 22% ning Losash teek mis oli andis võrdse 20%. Järgmisena tuleb MooTools (15.3%) ja väiksema operatsioonide arvu täitas jQuery teek, umbes 12.5%.

Operatsioonid DOM-elementidega näitasid ootamatu resultaadid. Väga suurelt rebis teistest ette ja andis parima tulemuse puhas JavaScript, mis protsendilises seoses koostas 68.8% mida võib näha ka graafikust. Graafikust on ka näha, et teiste teekide tulemused ei erine teineteiset suurelt. Underscore teek sai täidetus umbes 12.6% kõikidest operatstioonide arvust antud grupis. MooTools ja jQuery teegid said peaaegu sarnased tulemused: 7% ja 7.3%. Kuna nende tulemuste vahe on ainult 0.3%, võib öelda, et nad töötavad sama kiirusega DOM-elementidega. Halvema tulemuse sai Losash teek, mis täitas umbes 4.2% operatsioonidest grupis.

Eelnevalt sai vaadeldud 4 gruppi ehk kategooriat eraldi. Selleks, et aru saada millised resultaadid sai iga teek, analüüsime kategooriad kokku ja protsentides esitatud tulemusi on võimalik näha järgmistel graafikutel.




Joonis 12. Teekide jõudlus Mozilla Firefox brauseris.

Mozilla Firefox brauseris tehtud eksperemendid näitasid, et kõige suurema operatsioonide arvu, mis oli täidetud ühe sekundi jooksul tegi puhas JavaScript. Sellest võib teha ka järelduse, et puhas JavaScript omab kõrgema jõudlust võrreldes koodiga mis oli kirjutatud teekide abil. Järgmisena tuleb Lodash teek, mis jääb JavaScript'ist kaugele 6% võrra. Kolmandal ja neljandal kohtadel on Underscore ja MooTools teegid. Kõige halvema resultaadi näitas jQuery teek, mille operatsioonide arv sekundis on väiksem, järelikult ja jõudlus on halvem võrreldes teiste teekidega.

## 5.4 Eksperementide üldistus

Käesolevas alapeatükis antakse eksperimendi tulemuste põhjal ülevaate ja järeldused. Vaadeldakse kuidas mõjutab JavaScript-mootor brauserite kooditäitmise kiirusele, võrreldakse teegid omavahel. Lisaks vaadeldakse milline teek näitas parima tulemuse tehtud eksperimendides ja mis võis mõjutada tulemusi.

Vaadeldes eksperimentide tulemusi erinevates brauserites ja võrreldes neid brausereid omavahel on esimesest pilgust on näha, et iga brauser täitab koodi erineva kiirusega. See tähendab, et JavaScript-mootor, mis on ehitatud iga brauseri sisse ja mille põhiliseks ülesandeks on JavaScript koodi töötlemine ja täitmine, mõjub omamoodi koodi täitmise kiirusele ja antud töö raames mõjub ka eksperimendide tulemustele. Kõige suurema operatsioonide arvu, mis oli täidetud näitas Google Chrome (peaaegu 10.5 milj.). Eksperemendi tulemusest võib teha järelduse, et ühed ja samad operatsioonid võrreldes teiste brauseritega, täidab Google Chrome kiiremini. Pärast Google Chrome brauserit järgneb koodi täitmise kiiruselt Microsoft Edge, mille suuremaks tulemuseks on peaaegu 6 miljonit operatsiooni sekundis. Kõige aeglasema tulemuse andis Mozilla Firefox, mille täidetud operatsioonide arv oli 2.7 miljoni sekundis. Kõigis kolmes brauseris näitas puhas JavaScript parima tulemuse töötades objekti, ridade (String'ide) ja DOM-elementidega. See tähendab, et kolmes kategoorias neljast, mis sai kontrollitud selle töö raames, sai parima tulemuse kood, mis kirjutatud puhta JavaScripti abil ehk töötas kiiremini võrreldes teiste teekidega.

Testid on näidanud, et töötades massiividega parimaks valikuks antud teekide hulgast oleks Lodash teek. Kuna Lodash teegi loomise üheks sihiks oli massiivide iteratsiooni optimeerimine ja antud töös massiivide testimiseks paljud funktsioonid kasutasid iteratsiooni, testides massiividega Lodash on näidanud parima tulemuse. Vaatamata faktorile, et töötades DOM-elementidega kõik tegevused olid realiseeritud kõikides teekides, kasutasid Undescore ja Lodash teegid selleks täiendressurse: *lodash.dom-traverse* [8] ja *underscore.dom* [9] ning MooTools kasutas eraldi ühendatud moodulit. Need faktorid võiksid ära seletada nii suurt tulemuste erinevust puhta JavaScripti ja kõige teiste teekide vahel, töös DOM-elementidega kõikides brauserites. Lisaks natiivne JavaScript töötades näiteks DOM-elementidega otsib vajalikku elementi madalalt tasemel. Juhul, kui elementi otsib jQuery või mõni teine teek, mis kasutab selektoreid peab esiteks aru saama, mis tähendab antud selektor ja mida ta peab tagastama. Need

operatsioonid võtavad töötlemiseks aega ja vähendavad koodi täitmiskiirust [12]. Veel üks järeldus, mida võib teha on see, et vaatamata sellele, et Underscore ja Lodash teegid on väga sarnased ja Lodash on mingis mõttes parendatud Underscore teek, testid on näidanud, et jõudlusel nendel on suur erinevus. Võrreldes teiste teekidega, Lodash näitas head tulemused, töötades massiivide ja objektidega. Samal ajal Underscore teek kiiremini täidab testid objektidega.

Nagu oli juba mainitud paragrahvis 2.1 on jQuery väga populaarne JavaScript teek, mida vastavalt statistikale [13] kasutatakse 71.9% kõikidel lehekülgedel. Vaatamata sellele, et antud töö raames jQuery näitas kõige väiksemat operatsioonide arvu kõikides testitud kategooriates. Lisaks kõikides brauserites on ta näidanud halvima tulemuse. Antud töös jQuery saadud tulemust võib ka põhjendada sellega, et üheks jQuery sihiks on töötamine paljudes brauserites, mis lisab tema tööle erinevaid operatsioone, mille töötlemine võtab aega ja vähendab koodi täitmisekiirust.

Kui järjestada antud töös testitud teegid nende jõudluse tulemuste järgi, kus esimesena seisab kõrgeima jõudlusega teek, siis saame sellise pingerivi:

1. Puhas JavaScript
2. Lodash
3. Underscore
4. MooTools
5. jQuery

Kindlasti on selge, et iga eduka rakenduse arendamise jaoks on vaja õigesti valida arendamise instrumente. Kõik sõltub sellest, millised ülesanded peab rakendus lahendama. Näiteks rakenduse prototüüpimisel tuleks kasutada jQuery, sest arendamine selle teegi abil ei võta palju aega, sellel on palju dokumentatsiooni ja informatsiooni arendamise kohta ning lisaks jQuey annab võimaluse realiseerida peaaegu ükskõik millise ülesande. Kui arendajale on vajalik kiire ridade (*String'*ide) töötlemine, realiseerimiseks sobiks MooTools teek. Lisaks see teek on kasulik juhul, kui arendamisel on tähtis rakenduse kaal, sest ta annab võimaluse jagada teeki mooduliteks ja kasutada ainult vajalikke osasid. Nagu oli juba mainitud, juhul, kui rakendus põhiliselt töötab massiividega, heaks instrumendiks on Lodash teek, sest ta on kiire ja annab võimaluse

kirjutada koodi „lühidalt“. *DOM-heavy* rakendustel ehk rakendustel, kus töö suurim osa on töö DOM-elementidega tuleks kasutada Underscore teeki, kuid on vaja arvesse võtta, et selleks on vaja ühendada täiendteek (*third-party library*). Kuna testid on näidanud, et puhta JavaScripti jõudlus on suurem kui teekidel, on selge, et selle kood on igal juhul töötab kiiremini. Valida puhta JavaScripti tuleks siis, kui rakenduse töökiirus on palju tähtsam, kui arendamise mugavus.

Vaadeldes  teekide populaarsust ja käesolevas töös saadud tulemust, võib teha järelduse, et paljud arendajad püüavad vältida koodi kirjutamis puhta JavaScripti abil, vaatamata asjaolule, et ilma teekide kasutamiseta JavaScript töötab kiiremini. Selleks võib leida mitu põhjust. Näiteks inimesed eelistavad arendamisel mugavust ja aja säästmist kiirusele. Lisaks annavad teegid võimaluse arendada rakendusi kiiremini ja teha koode lühemaks.

# 6 Kokkuvõte

Antud töö eesmärgiks oli JavaScript teekide jõudluse võrdlus koodi täitmisekiiruse näitel. Lisaks oli eesmärk uurida, kas mõjutab JavaScript-mootor, mis on ehitatud brauserite sisse eksperimendide tulemust või mitte.

Eesmärgi saavutamiseks oli valitud puhas JavaScript ja 4 teeki: jQuery, MooTools, underscore ja Lodash. Samuti oli valitud mitu tegevust, mille abil oli võimalik kontrollida, mis kiirusega erinevad teegid töötavad massiivide, objektide, ridade (String'ide) ja DOM-elementidega. JavaScript teekide töökiiruse kontrollimiseks oli kirjutatud rakendus programmeerimiskeeles PHP ja selleks, et mõõta kui palju aega võtab iga tegevuse koodi täitmine oli kasutatud JavaScripti teek - Benchmark.js. Kõik valitud tegevused olid realiseeritud valitud teekides ja testandmetena enne katsete algust ka andmebaasi sisse toodud. Selleks, et muuta eksperimenti objektiivsemaks, olid katsed tehtud ka erinevates brauserites: Google Chrome versiooniga 54.0.2840.99, Microsoft Edge 38.14393.0.0 ja Mozilla firefox versiooniga 50.0.1.

Töö oluliseks tulemuseks on erinevate tegevuste täitmise kiiruse mõõdistamine erinevates teekides, mis andis võimaluse selliste teekide jõudlust võrrelda. Töö käigus sai kõikidele esitatud küsimustele vastused ka leitud ning püstitatud eesmärgid saavutatud. Käesoleva töö käigus saadud tulemus näitab, et puhas JavaScript töötab kiiremini ilma teekide kasutamiseta. Teekide võrdlemisel selgus, et ideaalset teeki ei ole. Igal teegil on oma plussid ja miinused, tugevad ja nõrgad küljed. Tähtis järeldus, mis sai tehtud antud töö tulemuste põhjal oli, et teegi populaarsus pole mitte alati võrdeline selle mugavuse ja töökiirusega. Jõudluse kontroll erinevates brauserites näitas, et JavaScript-mootor ikkagi mõjutab koodi täitmise kiirust, kuid teekide käitumine erinevates brauserites on enam-vähem sarnane.

# 7 Kasutatud kirjandus


[1] „w3techs,“ [Võrgumaterjal]. Available: https://w3techs.com/technologies/details/cp-javascript/all/all. [Kasutatud 03 09 2016].

[2] „jQuery blog,“ [Võrgumaterjal]. Available: http://blog.jquery.com/2006/01/24/jquery-blog/. [Kasutatud 2016 09 05].

[3] S. St.Laurent, jQuery Cookbook, O'Reilly Media Inc, 2010.

[4] Leanpub, „LoDash Cook Book,“ [Võrgumaterjal]. Available: https://leanpub.com/lodashcookbook/read. [Kasutatud 2016 11 5].

[5] J.-D. Dalton. [Võrgumaterjal]. Available: https://mathiasbynens.be/notes/javascript-benchmarking. [Kasutatud 28 12 2016].

[6] „GitHub Inc,“ [Võrgumaterjal]. Available: https://github.com/getify/You-Dont-Know-JS/blob/master/async%20%26%20performance/. [Kasutatud 2016 11 2].

[7] „Hongkiat Blog,“ [Võrgumaterjal]. Available: http://www.hongkiat.com/blog/ampps-server/. [Kasutatud 2016 10 02].

[8] „Lodash dom-traverse,“ [Võrgumaterjal]. Available: http://szarouski.github.io/lodash.dom-traverse/explain.html. [Kasutatud 2016 10 22].

[9] „GitHub Inc,“ [Võrgumaterjal]. Available: http://1000ch.github.io/underscore.dom/. [Kasutatud 2016 10 08].

[10] „Global Statistic Counter,“ [Võrgumaterjal]. Available: http://gs.statcounter.com/#browser-ww-monthly-201511-201611-bar. [Kasutatud 2016 12 01].

[11] „GitHub Inc,“ [Võrgumaterjal]. Available: https://github.com/v8/v8/wiki/Introduction. [Kasutatud 2016 11 20].

[12] „Stack Overflow Internet Services Inc.,“ [Võrgumaterjal]. Available: http://stackoverflow.com/questions/20444882/why-is-jquery-selector-function-so-slow-compared-to-native-dom-methods. [Kasutatud 2016 11 5].

[13] „w3techs statistics,“ [Võrgumaterjal]. Available: https://w3techs.com/technologies/overview/javascript_library/all. [Kasutatud 2016 11 08].

[14] „Lodash official documentation,“ [Võrgumaterjal]. Available: https://lodash.com/custom-builds. [Kasutatud 09 12 2016].

# Lisa 1 – Google Crome eksperimendi tulemused

Tabel 5. Töö massiiviga.

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Korduvate elementide kustutamine | 589,266 | 133,317 | 2,068,053 | 288,922 | 3,766,275 |
| Elementide segunemine | 537,378 | | 4,885,760 | 1,567,048 | 2,007,326 |
| Elementidele väärtuse omastamine (map) | 516,519 | 446,765 | 400,741 | 729,542 | 6,536,818 |
| Filtreerimine | 513,527 | 183,790 | 509,562 | 666,072 | 2,553,481 |
| Pikkuse arvutamine | 30,304,170 | 711,388 | | 2,772,080 | 2,426,039 |
| Väiksema elemendi otsimine | 624,051 | | 907,485 | 2,114,674 | 7,608,001 |
| Suurema elemendi otsimine | 586,114 | | 908,315 | 1,591,980 | 3,964,707 |
| Elemendi sisaldamise kontroll | 15,270,168 | 5,404,730 | 9,343,177 | 1,776,386 | 1,249,718 |
| Suvalise elemendi saamine (random) | 13,689,881 | | 12,633,261 | 2,133,734 | 12,862,584 |

Tabel 6. Töö objektidega.

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Pikkuse arvutamine | 32,271,396 | 265,892 | 157,723 | 3,090,610 | 2,989,331 |
| Võtmete saamine | 4,842,460 | 630,644 | 3,778,881 | 1,922,145 | 4,744,736 |
| Väärtuste saamine | 2,284,890 | 769,077 | 2,753,971 | 2,980,024 | 2,134,325 |

Tabel 7. Töö ridadega (Strings)

| **Tegevus** | **Puhas JS** | **JQuery** | **MooTools** | **Underscore** | **Lodash** |
|---|---|---|---|---|---|
| Pikkuse arvutamine | 62,995,650 | 140,765 | | 4,549,446 | 10,375,880 |
| Tühikute kustutamine | 13,073,002 | 3,002,827 | | | 2,820,169 |
| Sisaldamise kontroll | 5,090,855 | | 2,526,037 | | 5,875,041 |
| Stringi osa vahetamine | 2,331,197 | | 1,798,939 | | 2,385,733 |
| CamelCase lause | 112,910 | | 1,401,927 | | 133,360 |
| Suur algustäht | 323,684 | | 348,946 | | 1,917,232 |
| Teha reast numbri | 7,477,344 | | 3,432,656 | | 4,706,328 |

Tabel 8. Töö DOM-elementidega

| **Function** | **Puhas JS** | **JQuery** | **MooTools** | **Underscore** | **Lodash** |
|---|---|---|---|---|---|
| Identifikaator | 13,212,064 | 1,536,716 | 181,157 | 1,721,672 | 84,697 |
| Klassi nimi | 12,444,670 | 211,045 | 170,676 | 569,315 | 66,228 |
| Atribuut | 3,443,805 | 168,942 | 162,718 | 320,280 | 69,162 |
| Elemendi lapsed | 8,379,583 | 394,276 | 211,565 | 328,681 | 23,567 |

# Lisa 2 – Mozilla Firefox eksperimendi tulemused

Tabel 9. Töö massiiviga

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Korduvate elementide kustutamine | 1,560,567 | 124,164 | 1,448,086 | 1,553,116 | 1,924,681 |
| Elementide segunemine | 782,937 | | 2,142,902 | 1,568,242 | 1,516,384 |
| Elementidele väärtuse omastamine (map) | 2,141,478 | 1,196,080 | 2,063,538 | 554,017 | 1,521,993 |
| Massiivi filtreerimine | 1,892,295 | 336,218 | 2,071,307 | 582,048 | 1,614,063 |
| Pikkuse arvutamine | 2,184,316 | 373,533 | | 2,275,156 | 1,890,520 |
| Väiksema elemendi otsimine | 637,848 | | 1,869,692 | 2,108,506 | 2,273,803 |
| Suurema elemendi otsimine | 606,546 | | 1,808,736 | 1,850,811 | 2,301,846 |
| Elemendi sisaldamise kontroll | 2,215,233 | 2,196,208 | 1,762,253 | 2,165,951 | 2,165,349 |
| Suvalise elemendi saamine (random) | 2,424,471 | | 2,176,461 | 2,175,069 | 2,310,445 |

Tabel 10. Töö objektidega

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Pikkuse arvutamine | 2,287,455 | 746,192 | 599,545 | 1,338,064 | 1,223,679 |
| Võtmete saamine | 1,422,060 | 1,416,028 | 1,444,340 | 556,370 | 1,293,850 |
| Väärtuste saamine | 1,212,676 | 1,368,122 | 1,185,057 | 1,326,741 | 1,139,965 |

Tabel 11. Töö ridadega

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Pikkuse arvutamine | 2,295,255 | 160,833 | | 1,623,222 | 1,367,634 |
| Tühikute kustutamine | 2,173,479 | 1,564,087 | | | 1,605,200 |
| Sisaldamise kontroll | 1,886,184 | | 1,170,111 | | 1,808,509 |
| Stringi osa vahetamine | 2,325,546 | | 1,052,414 | | 2,287,462 |
| Camelcase lause | 322,622 | | 706,853 | | 127,224 |
| Suur algustäht | 265,036 | | 253,093 | | 717,386 |
| Tegema reast numbri | 2,178,999 | | 1,185,921 | | 1,669,478 |

Tabel 12. Töö DOM-elementidega

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Identifikaator | 2,261,981 | 766,944 | 221,330 | 954,550 | 121,457 |
| Klassi nimi | 2,025,358 | 237,780 | 147,074 | 400,061 | 120,306 |
| Atribuut | 1,131,567 | 145,484 | 134,010 | 233,577 | 116,546 |
| Elemendi lapsed | 1,745,838 | 118,835 | 199,798 | 248,584 | 40,701 |

# Lisa 3 – Microsoft Edge eksperimendi tulemused

Tabel 13. Töö massiividega

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Korduvate elementide kustutamine | 1,254,412 | 544,382 | 1,786,778 | 722,972 | 3,206,185 |
| Elementide segunemine | 1,449,457 | | 4,904,172 | 2,115,087 | 2,053,391 |
| Elementidele väärtuse omastamine | 2,199,222 | 830,969 | 2,245,323 | 1,210,932 | 2,648,001 |
| Filtreerimine | 2,094,264 | 164,924 | 2,113,273 | 1,026,965 | 2,658,449 |
| Pikkuse arvutamine | 6,910,901 | 378,800 | | 3,431,152 | 1,876,049 |
| Väiksema elemendi otsimine | 1,322,671 | | 2,664,499 | 2,904,305 | 3,028,135 |
| Suurema elemendi otsimine | 1,372,556 | | 2,698,732 | 2,756,975 | 2,905,691 |
| Elemendi sisaldamise kontroll | 4,990,854 | 4,133,394 | 4,357,626 | 2,506,845 | 1,683,232 |
| Suvalise elemendi saamine (random) | 4,242,339 | | 6,126,639 | 3,382,904 | 4,894,024 |

Tabel 14. Töö objektiga

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Pikkuse arvutamine | 5,007,585 | 538,911 | 286,314 | 1,229,518 | 998,100 |
| Võtmete saamine | 1,618,872 | 894,847 | 1,611,904 | 1,469,850 | 1,226,536 |
| Väärtuste saamine | 983,590 | 920,316 | 1,343,436 | 1,343,436 | 877,805 |

Tabel 15. Töö ridadega

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Pikkuse arvutamine | 6,447,338 | 84,509 | | 3,464,285 | 2,022,595 |
| Tühikute kustutamine | 3,885,639 | 4,650,340 | | | 4,132,348 |
| Sisaldamise kontroll | 4,156,528 | | 2,208,421 | | 2,577,693 |
| Stringi osa vahetamine | 5,438,378 | | 1,814,422 | | 2,188,101 |
| Camelcase lause | 527,505 | | 1,860,823 | | 97,526 |
| Suur algustäht | 353,073 | | 387,744 | | 1,016,930 |
| Tegema reast numbri | 5,921,761 | | 2,680,540 | | 3,641,026 |

Tabel 16. Töö DOM-elementidega

| Tegevus | Puhas JS | JQuery | MooTools | Underscore | Lodash |
|---|---|---|---|---|---|
| Identifikaator | 2,445,203 | 676,950 | 159,743 | 1,301,807 | 99,926 |
| Klassi nimi | 1,997,203 | 317,388 | 144,469 | 612,718 | 91,321 |
| Atribuut | 270,154 | 97,522 | 82,747 | 161,737 | 66,405 |
| Elemendi lapsed | 1,740,651 | 145,243 | 150,444 | 265,538 | 25,665 |