TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT

Mihkel Väli 134679

JAVASCRIPTI TESTIMISRAAMISTIKE JASMINE, MOCHA JA QUNITI KASUTAMISE VÕRDLUS VEEBIRAAMISTIKU ANGULARJS RAKENDUSES

bakalaureusetöö

Juhendaja: Deniss Kumlander

Vanemteadur

PhD (tehnikateaduste doktor)

Tallinn 2016

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: Mihkel Väli

15.05.2016

Annotatsioon

Käesoleva bakalaureusetöö „*Javascripti* testimisraamistike *Jasmine*, *Mocha* ja *QUniti* kasutamise võrdlus veebiraamistiku *AngularJS* rakenduses" ülesandeks on välja selgitada, kas *AngularJS* veebiraamistiku abil kirjutatud projektide testimiseks on tõesti parim testimisraamisik *Jasmine* nagu seda *AngularJS* dokumentatsioon soovitab. Selle selgitamiseks võrreldakse *Jasminet* kahe teise testimisraamisikuga – *Mocha* ja *QUnit*. Objektiivse hinnangu saamiseks võrreldakse testimisraamistikke erinevate kriteeriumite alusel, mille ülesandeks on välja tuua testimisraamistike tugevad ja nõrgad küljed. Kriteeriumite olulisuse ja erinevuse rõhutamiseks määratakse igale kriteeriumile olulisuse hinnang. Olulisuse hinnangu ülesanne on tagada tähtsamate kriteeriumite parem kajastatus töö lõplikes tulemustes. Sellist metoodikat rakendades selgitatakse välja parim testimisraamistik, mida soovitatakse kasutada *AngularJS* veebiraamistiku abil kirjutatud projektide testimiseks. Antud ülesannet selliselt lahendades leiti, et parim testimisraamistik *AngularJS* rakenduse testimiseks on *Mocha*, ehkki *AngularJS* dokumentatsioon soovitab oma rakenduste testimiseks *Jasminet*.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 40 leheküljel, 6 peatükki, 16 joonist, 5 tabelit.

Abstract

Comparison of JavaScript unit testing frameworks Jasmine, Mocha and QUnit in an AngularJS project

The purpose of this bachelor's thesis is to identify whether the documentation of AngularJS recommends the best unit testing framework, when it advises to use Jasmine for unit testing AngularJS projects. For this, Jasmine is compared with two other unit testing frameworks – Mocha and QUnit. For an objective evaluation the author is comparing the unit testing frameworks using four different criteria with the purpose to point out the strong and weak sides of each unit testing framework. To emphasize the importances and differences of the criteria, an importance rating is assigned to each criterion. The purpose of the importance rating is to highlight more important criterion better in the final results. Implementing this methodology, the author will identify the best unit testing framework, which is suggested to use for unit testing an AngularJS project.

To solve the established task, the thesis is divided into six chapters. The first chapter introduces the technologies used in the project, brings out the project that is going to be tested, and describes the criteria, on which the unit testing frameworks are compared with. In the next three chapters author analyses each unit testing framework individually and brings out their strengths and weaknesses. In the end of each of the three chapters each unit testing framework is rated using the criteria set earlier. Finally, author compares all of the unit testing frameworks and analyses the results using Web-Hipre.

Solving the assignment in the given way, author discovered that the best unit testing framework to use with an AngularJS project is Mocha, even though the documentation of AngularJS recommends to use Jasmine for unit testing an AngularJS project.

The thesis is in estonian and contains 40 pages of text, 6 chapters, 16 figures, 5 tables.

Lühendite ja mõistete sõnastik

TTD	*Test-driven development*, arendamine testide põhjal
MIT	*Massachusetts Institute of Technology*, Massachusettsi Tehnoloogiainstituut
MVC	*Model-View-Controller*, mudel-kasutajavaade-kontroller, arhitektuuri muster
npm	*Node.js Package Manager*, Node.js pakettide haldusvahend
DOM	*Document Object Model*, dokumendiobjektide mudel
JavaScript	Programmeerimiskeel, mis võimaldab veebiautoritel luua interaktiivseid veebisaite
Veebiraamistik	Veebiarendust toetav struktuur
Testimisraamistik	Testimist toetav struktuur
AngularJS	*JavaScripti* veebiraamistik
Jasmine	*JavaScripti* testimisraamistik
Mocha	*JavaScripti* testimisraamistik
QUnit	*JavaScripti* testimisraamistik
Süntaks	Moodustatavat struktuuri määravad reeglid
Skaleerimine	Mastaabi muutmine
Struktureerimine	Süsteemi korrastamine, liigendamine
Teek, teegid	Failide, programmide, tavakäskluste ja -funktsioonide kogum
Kriteerium	Tunnus millegi eristamiseks
Kood	Arvutiprogramm või selle osa
Ühiktestimine	Tarkvara testimise meetod ühe osa koodi testimiseks
Integratsioon	Erinevate koostisosade lõimumine
Implementeerimine	Testimisraamistiku rakendamine projektis
Blogi	Veebipäevik
Kernel	Ressursijaotust ja muid põhifunktsioone hõlmav operatsioonisüsteemi keskne osa
Linuxi distributsioon	Linuxi kernelil põhinev operatsioonisüsteem
Node.js	Keskkond veebirakenduste serveripoolse osa arenduseks

Moodul	Programmiüksus või osa
jQuery	*JavaScripti* teek veebilehe kliendipoolse osa lihtsamaks haldamiseks
jQuery lite	*jQuery* täispaketi alamhulkadest koosnev teek
Brauser	Veebilehitseja
Karma	*JavaScripti* testide käivitaja
Null	Väärtuse puudumine
BDD	*Behavior-driven development*, arendamine käitumise põhjal
Fail	Tervikuna talletatav või töödeldav, nimega varustatud kirje kogum
Kaust	Failihaldussüsteemi hierarhia arvutis
Juurkaust	Süsteemi kõrgeimal hierarhiatasemel paiknev kaust
Test	Kontroll millegi nõuetele vastavusele
Mock	Matkimine, jäljendamine, kopeerimine
Moodul	Loogiliselt eristatav programmi osa
Kontroller	Programmi tööd juhtiv moodul
Genereerima	Tekitama, esile kutsuma
Expect	Üks sisestamissüntaksi märksõnu, oota
Skript	Käsujada, mida täidetakse ilma kasutajapoolse vahelesegamiseta
Asünkroonne	Mitteüheaegne
Päring	Info otsimise korraldus automatiseeritud infootsisüsteemides
Konfigureerimine	Süsteemi tarkvara korraldus- ja ühendusviis
Dokumentatsioon	Süstematiseeritud dokumentide kogum
Should	Üks sisestamissüntaksi märksõnu, peaks
Assert	Üks sisestamissüntaksi märksõnu, sisesta
Joondama	Teksti või graafika paigutamine veerise suhtes teatud viisil
Atribuut	Juurde kuuluv tunnus
Java	Programmeerimiskeel
Goal	Eesmärk
Criteria	Kriteerium
Alternatives	Alternatiivid

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Automaattestimine on tänapäeva tarkvaraarenduses olulisel kohal. Seda tänu mitmele automaattestimise omadusele. Automaattestid võivad ära hoida *TDD* arendusmetoodikat kasutades kulukaid vigade parandusi, vähendada testimisele kuluvat aega ning parandada koodi kvaliteeti. Sellest tingituna luuakse ja kasutatakse üha enam automaattestimist lihtustavaid ja kiirendavaid raamistikke.

Autori praktilisest kogemustest lähtudes kasutatakse *JavaScripti* rakendustes üllatavalt vähe automatiseeritud ühiktestimist. Analüüsides antud olukorda, leiti mitmeid seda situatsiooni põhjustavaid tegureid. Siin kohal tuuakse välja kaks peamist põhjust.

1. Rakendused leitakse olevat liiga väikesed ja lihtsad ning nendele ei vaevuta projektides automaatteste kirjutama.

2. *TDD* integreerimine arendusprotsessi suurendab tavaliselt projekti eelarvet, mistõttu ei suudeta sellist lähenemist kliendi jaoks piisavalt atraktiivseks teha.

JavaScripti ühiktestimise vähesest populaarsusest hoolimata on selleks loodud mitu väga head testimisraamistikku.

JavaScripti üks populaarsemaid arendamisraamistikke, *AngularJS*, soovitab oma dokumentatsioonis ühiktestimiseks kasutada *Jasmine* testimisraamistikku[1] . Käesoleva lõputöö teemaks on uurida, kas *Jasmine* on parim raamistik *AngularJS* rakenduse testimiseks. Selleks võrreldakse *Jasmine* raamistikku kahe teise *JavaScripti* ühiktestimiseks mõeldud raamistikuga. Nendeks on autori poolt valitud testimisraamistikud *Mocha* ja *QUnit*. Testitakse lihtsat *AngularJS* rakendust, mis võimaldab kirjutada testid suhteliselt sarnased ning suunab töö põhirõhu testimiskeskkondade sisulisele võrdlemisele. Objektiivse hinnangu tagamiseks hinnatakse testimisraamistikke erinevate kriteeriumite alusel, mis toovad välja raamistike positiivsed ja negatiivsed küljed. Lisaks on kriteeriumitele määratud

olulisuse hinnangud, mis tagavad olulisemate kriteeriumite rõhutatud kajastumise lõplikes tulemustes.

AngularJS testimisraamistikega seonduvat on erialases ja akadeemilises kirjanduses suhteliselt vähe kajastatud. Lähtuvalt sellest on töös toetutud käsitletavate raamistike ametlikele dokumentatsioonidele, blogide ja foorumide postitustele, samuti autorit konsulteerinud kolleegide ning tema isiklikule kogemusele.

Töö on üles ehitanud selliselt, et algul tuuakse välja testimiskeskkond ja testitav rakendus. Pärast seda tutvustatakse töös käsitletavaid testimisraamistikke ja põhjendatakse nende valikut. Samuti tutvustatakse kriteeriumeid, mille alusel testimisraamistikke võrreldakse. Järgmiseks käsitletakse eraldi peatükkides kõiki kolme testimisraamistikku ja tutvustatakse nende kasutamist *AngularJS* projektis, süntaksi eripärasusi, ühilduvaid teeke ning muid antud testimisraamistikuga kokku puutuvaid huvitavaid teemasid. Pärast iga testimisraamistiku tutvustust ja kasutamist *AngularJS* projektis analüüsitakse antud peatüki viimases alapunktis tema tugevusi ja nõrkusi. Saadud tulem võetakse kokku töö viiendas peatükis, kus võrreldakse ja analüüsitakse testimisraamistike saavutatud tulemusi. Selle alusel pakutakse välja parim testimisraamistik *AngularJS* projekti ühiktestimiseks.

Antud tööd saavad arendajad ja testijad kasutada praktikas *AngularJS* projekti ühiktestimiseks enda jaoks mugavaima ning parima testimisraamistiku valimisel. Samuti saab käesolevas töös minu poolt kasutatud metoodikat teatud mööndustega järgides valida sobivaim ühiktestimisraamistik teiste *JavaScripti* veebiraamistike abil kirjutatud projektide testimiseks.

Ülesande püstitus

AngularJS'i dokumentatsioon soovitab oma projektide ühiktestimiseks kasutada *Jasmine* testimisraamistikku[1] . Sellest tulenevalt on antud bakalaureusetöö eesmärgiks uurida, kas *Jasmine* on parim raamistik *AngularJS* projektide testimiseks nagu dokumentatsioonis soovitatud. Teisiti öeldes uuritakse, kas *Jasmine* on teistega võrreldes märgatavalt parem testimisraamistik, et teda dokumentatsioonis nii eriliselt rõhutada. Eesmärgi saavutamiseks võrreldakse *Jasmine* testimisraamistikku kahe teise *JavaScripti* ühiktestimiseks mõeldud raamistikuga, milleks valiti testimisraamistikud *Mocha* ja *QUnit*. Testimiseks kasutatakse *AngularJS*'i abil autori poolt kirjutatud projekti. Selle projekti põhiülesanne on jätta võrreldavate testimisraamistike abil kirjutatud testid sisult võimalikult sarnaseks. Samas saab autori pakutava *AngularJS*'i abil kirjutatud projektiga välja tuua testimisraamistike suurimad erinevused. Objektiivse hinnangu tagamiseks hinnatakse testimisraamistikke erinevate kriteeriumite alusel. Kriteeriumite ülesandeks on välja tuua testimisraamistike tugevad ja nõrgad küljed. Samas ei ole kõik kriteeriumid testimisraamistike hindamise suhtes ühtmoodi olulised. Kriteeriumite olulisuse ja erinevuse rõhutamiseks on määratud igale kriteeriumile olulisuse hinnang. Kriteeriumite olulisuse hinnangu ülesanne on tagada olulisemate kriteeriumite parem kajastatus lõplikes tulemustes. Ülesande sellisel püstitamisel ja pakutud lahenduskäiku järgides selgub parim testimisraamistik *AngularJS* projekti testimiseks.

1 Testitava projekti ülevaade

1.1 Kasutatavad tehnoloogiad

Töös käsitletud rakendus ning temale kirjutatud testid on üles seatud kasutades *Linuxi* distributsiooni *Ubuntu* versiooniga *14.04LTS*. Raamistike ning muude vajalike töö vahendite paigaldamiseks on kasutatud *Node.js* pakettide haldusvahendit *npm*.

JavaScripti rakendus, millega testimisraamistike ühildan, on minu poolt *AngularJS* veebiraamistikus kirjutatud vastav projekt. *AngularJS* on *MIT* litsentsil põhinev vabavaraline veebirakenduste arendamise lihtsustamiseks loodud raamistik[1] . Tegemist on *MVC* arhitektuuri mudelites kliendi pool andmete mugavaks kuvamiseks loodud raamistikuga. *AngularJS* eesmärgiks on eraldada *DOM* tema rakenduse loogikast, eraldada kliendi poolne rakenduse osa serveri poolsest ning võimaldada struktureeritud rakenduse loomist[1] .

AngularJS'iga koos kasutatakse *Jasmine* ja *Mocha* testimiskeskkondade üles seadmisel moodulit nimega *angular-mocks*[1] , [8] . *Angular-mocksi* abil on võimalik testimiseks luua *AngularJS* projekti objektide abil identselt *mockitud* objekte, mis oma olemuselt on koopia antud objektist ning jäljendab täielikult tema funktsiooni. Testid kirjutatakse töötama *mockitud* objekte kasutades. See garanteerib testi korrektsuse.

Testimisraamistikena võrdlen on *Jasminet*, *Mochat* ja *QUnitit*. *Jasmine* on raamistik, mida *AngularJS* dokumentatsioon soovitab ühiktestimiseks kasutada[1] . Kasutan *Jasmine* etalonina, mille abil saan välja tuua kõigi testimisraamistike tugevad ja nõrgad küljed. Teine populaarne testimisraamistik *JavaScripti* projektide testimiseks on *Mocha*[4] . Koos *Mochaga* kasutan sisestusteeki *expect.js*, mis teeb *Mocha* testide süntaksi *Jasmine* süntaksile väga sarnaseks. Tänu sellele on lihtne välja tuua põhjused, miks *AngularJS* dokumentatsioon ei soovita *Mochat* kasutada. *QUniti* loomise algseks eesmärgiks oli testida *JavaScripti* teegi *jQuery* rakendusi. Kuna ka *AngularJS* kasutab sisemiselt *jQuery* täispaketi alamhulkadest koosnevat teeki *jQuery lite*, siis on huvitav

testida projekti ka *QUnitiga*, et teada saada tema käitumine *AngularJS* veebiraamistikus kirjutatud projektis[1] . Lisaks minu professionaalsele huvile näha kuidas *QUnit AngularJS* projektis töötab, on *QUnit* väga võimekas *JavaScripti* testimisraamistik. See teeb *QUniti* ideaalseks kolmandaks valikuks.

Testitavas projektis testin ainult *AngularJS* kontrollereid, sest töö põhirõhk on suunatud testimiskeskkondade paigaldamisele, süntaksi arusaadavusele ja struktureerimise võimalustele. *AngularJS* kontrollerite testide põhjal on võimalik kirjutada testid ka tema teenustele.

Testimiskeskkondi ning teste käivitatakse võimalikult puhtal kujul brauseris. See tähendab, et testide tööle panemiseks ei kasutata testide käivitamise keskkonda *Karma*, *Node.js* käsurealiidest ega ka muid sarnase põhimõttega rakendusi.

1.2 Testitav rakendus

Lisaks keskkondade paigaldamisele ning testitava rakendusega ühildamisele, võrdlen ka testrakenduse enda testimiseks mõeldud teste. Testitav *AngularJS* rakendus sisaldab kahte funktsionaalsust ning on nähtav Lisas 1. Kontrolleri sisu on nähtav joonisel 1.

```
$scope.variable = 'variable';
$scope.getSum = function() {
   $scope.sum = $scope.x + $scope.y;
};
```

Joonis 1. Testitava rakenduse kontroller

Esimese testi ülesanne seisneb muutuja „$scope.variable” kontrollimises. Testi eesmärk on kontrollida, et see ei oleks *null*. Teise testi ülesanne on kontrollida funktsiooni „$scope.getSum()” korrektse tulemuse saavutamist.

1.3 Teised teemaga seotud tööd

Identse teemakäsitlusega tööd ei ole ma erialasest kirjandusest leidnud. Samuti ei leidnud ma Tallinna Tehnikaülikoolis kaitstud bakalaureuse-ja magistritööde hulgast

analoogset teemakäsitlust. Olen antud teemat uurinud, kasutades erinevaid otsingumootoreid, andmebaase ning erialast kirjandust.

Nii näiteks raamatus „*AngularJS Testing Cookbook*"[12] on käsitletud ühiktestimist kasutades *Jasminet AngularJS* projektis. Antud raamatus puudub täielikult põhjendus testimisraamistiku valiku osas. Teistest testimisraamistikest mainitakse ühe lausega ainult *Mochat*, öeldes, et tegemist on hea testimisraamistikuga.

Raamatus „*JavaScript Unit Testing*"[13] on käsitletud *JavaScriptis* kirjutatud rakenduse testimist *Jasmine* ja *QUniti* abil. Antud raamatus on täielikult käsitlemata jäänud *AngularJS*'i rakenduse testimine. Samuti ei ole käsitletud ühiktestimist, kasutades testimisraamistikku Mocha. Käsitlemata on raamatus mainitud testimisraamistike *Jasmine* ja *QUniti* vahelised võrdlused ning analüüs.

Lõputöö „*Automated testing of a web-based user interface*"[14] on hea ülevaatlik töö veebipõhiste kasutajaliideste automaattestimisest. Kahjuks puudutatakse teemat pinnapealselt. Samuti ei puudutata antud lõputöös *Mochat* ja *Qunitit*. Lõputöös ei ole kasutatud ühesugust kindlat lähenemist probleemile.

Blogipostituste seerias, mis algab postitusega „*Getting started with Unit Testing for AngularJS*"[15] on käsitletud hästi testimisraamistiku *Jasmine* kasutamist *AngularJS* rakenduses. Tuuakse ilmekaid sellekohaseid näiteid. Samas jäetakse täielikult põhjendamata testimisraamistiku valik. Mainimata jäetakse ka võimalikud alternatiivsed testimisraamistikud, mille abil *AngularJS* projekti testida.

Vaatamata eelpool loetletud allikates sisalduvale suhteliselt napile informatsioonile mind huvitaval teemal, kogusin sealt väärtuslikke näpunäiteid oma töös püstitatud ülesande lahendamiseks.

1.4 Kriteeriumid, nende valik ja määramine

Testimisraamistiku teevad heaks paljude läbimõeldud omaduste kokkulangemine. Kõikide testide korrektne läbimine on testimisraamistikule hinnangu andmise vältimatuks eelduseks. Lisaks peab testimisraamistik olema lihtsalt ja kiiresti kasutatav, õpitav ning ülesseatav. Samas peab testimisraamistik omama palju võimalusi testimaks

erinevaid stsenaariumeid. Oluline on testimisraamistiku juures ka see, et teda saaks kasutada suurte ja väga suurte projektide testimiseks. Vähem oluline on testimisraamistiku süntaksi ja graafilise vaate arusaadavus.

Testimisraamistike hindamiseks ning võrdlemiseks olen loonud kriteeriumid, mille eesmärk on välja tuua iga testimisraamistiku positiivsed ning negatiivsed küljed kasutamisel *AngularJS* projektis. Lisaks on näha, et kõik testimisraamistikku iseloomustavad omadused ei ole võrdväärselt olulised. Selle tõttu olen kogenumate kolleegide konsultatsioonide tagajärjel määranud igale kriteeriumile tema olulisust iseloomustava hinnangu. Kriteeriumite ja neile antud olulisuse hinnangute eesmärk on analüüse ning arvamusi objektiivselt kajastada. Kriteeriumite täitmist hindan 5-palli süsteemis, kus minimaalväärtus „1" tähendab, et kriteeriumi eesmärki ei ole täidetud ja maksimaalväärtus „5" tähendab, et kriteeriumi eesmärk on täielikult täidetud. Lisaks on määratud iga kriteeriumi olulisust kirjeldav hinnang. Olulisuse määramiseks kasutan samuti 5-palli süsteemi, kus minimaalväärtus „1" tähendab vähe olulist ning maksimaalväärtus „5" väga olulist kriteeriumit. Eelnevast tulenevalt hindan *AngularJS* projektis kasutatud testimisraamistikke järgnevalt enda poolt määratud kriteeriumite ja neile määratud olulisuse hinnangute alusel. Samuti põhjendan lühidalt iga kriteeriumi olulisuse hinnangu arvväärtuse põhjust.

Kriteerium 1. Kriteerium käsitleb testide skaleeritavuse ja struktureerimise võimalusi. Kriteeriumi hindamisel arvestan testimisraamistiku võimalusi oma teste struktureerida vastavalt vajadusele. Hea struktureerimise võimalus tagab tihti ka võimaluse teste skaleerida vastavalt vajadusele. Antud kriteeriumi eest saab kõrge hinde testimisraamistik, kus on võimalik oma teste arusaadavalt ja lihtsalt struktureerida. Sellele aitavad kaasa mugavalt kasutatavad funktsioonid, mille abil peab olema võimalik teste vastavalt vajadusele kirjeldada, grupeerida ja struktureerida. Struktureerimise võimalus on igas projektis kõige olulisem kriteerium, kuna ta annab projektile võimaluse kasvada. Kriteeriumi olulisus: 5.

Kriteerium 2. Kriteerium käsitleb süntaksi ja graafilise vaate loetavust ning arusaadavust. Kriteeriumis võetakse arvesse nii testide süntaksi sarnanemist inimkeelele kui ka testide lõpliku graafilise vaate arusaadavust. Antud kriteeriumi eest saab kõrge hinde testimisraamistik, kus on intuitiivselt arusaadav mida mingi osa süsteemist teeb.

Samuti on antud kriteeriumi täitmisel plussiks valikute võimalus sisestamissüntaksis ja graafilises vaates. Süntaksi ning graafilise loetavus ja arusaadavus vähendavad testimisele kuluvat aega ja muudavad testimisprotsessi igat pidi meeldivamaks. Eelneva tõttu leian, et tegemist on olulise kriteeriumiga. Kriteeriumi olulisus: 4.

Kriteerium 3. Kriteerium käsitleb keskkonna ülesseadmise lihtsust ja mugavust. Kriteeriumi hindamisel arvestan *AngularJS* projekti testimiseks mõeldud testimisraamistiku sõltumist erinevatest pakettidest ja välistest teekidest. Vähene pakettide arv muudab testimisraamistiku paigaldamise lihtsamaks, kuna arvestada tuleb vähemate võimalike probleemiallikatega. Sellest tulenevalt saab antud kriteeriumi eest kõrge hinde testimisraamistik, mis sõltub vähesest arvust erinevatest pakettidest. Testimiskeskkonna üles seadmise lihtsus on keskmise olulisusega. Kriteeriumi olulisus: 3.

Kriteerium 4. Kriteerium käsitleb lisavõimalusi ja ühilduvaid teeke antud testimisraamistikus, toetudes dokumentatsioonile. Kriteeriumi hindamisel arvestan kõike huvitavat, mida antud testimisraamisik võimaldab. Antud kriteeriumi eest saab kõrge hinde testimisraamistik, millega on võimalik enda testimiskeskkonnaga ühildada funktsionaalsust parandavaid teeke. Lisafunktsioonide võimalus on alati hea omadus, mis täiendab testimisraamisikku. Samas ei ole see võrreldes sisuliste funktsioonide ja kasutamismugavusega võrreldes kuigi oluline kriteerium. Kriteeriumi olulisus: 2.

Kõikide kriteeriumite olulisuse hinnangud on nähtavad tabelis 1.

Tabel 1. Kriteeriumid ning nende olulisus

Kriteerium	**Hinne**
Skaleeritavuse ja struktureerimise võimalus	5
Süntaks ja graafiline vaade	4
Keskkonna ülesseadmine	3
Lisavõimalused ja ühilduvad teegid	2

2 Jasmine

2.1 Raamistiku tutvustus

Jasmine on vabavaraline *BDD* testimisraamistik *JavaScriptile*[2] . Ta ei sõltu brauserist, DOM´ist ega ka ühestki teisest *JavaScripti* raamistikust[2] . Seega on *Jasmine* sobiv kõigele, kus kasutatakse *JavaScripti*. Tema ühe kasutatavama raamistiku *AngularJS* dokumentatsioon soovitab kasutada *Jasmine'*i, sest viimane pakub funktsioone, mis aitavad teste kergelt struktureerida ja skaleerida[1] .

Jasmine süntaks koosneb kahest struktureerivast funktsioonist „describe()" ja „it()" ning väiksematest võrdlemisele kaasa aitavatest funktsioonidest. Kahe parameetriga funktsiooni „describe()" kasutatakse individuaalsete testide grupeerimiseks plokki. Funktsiooni esimiseks sisendiks on tekst, mis kirjeldab, mida antud testide grupp tegema peab. Teine sisend on funktsioon, kus käsitletakse individuaalseid teste. Individuaalsed testid kirjutatakse „it()" funktsiooni, millel on samuti kaks sisendit. Esimene sisend on tekst, mis kirjeldab konkreetse testi tööd. Teine sisend on funktsioon kuhu kirjutatakse test. Selline lähenemine aitab väga lihtsalt ja mugavalt teste sisaldavat faili struktureerida. See aitab omakorda kaasa koodi üldisele loetavusele ning arusaadavusele[3] .

Testi loomiseks ja hilisemaks võrdlemiseks kasutan ühe sisendiga funktsiooni „expect()". Ülejäänud vajalikud töövahendid on võrdlemist lihtsustavad funktsioonid. Samuti on säilinud võimalus kirjutada enda projekti testimiseks vajalikke lisafunktsioone[3] .

Jasmine pakub võimalusi kirjutada üle raamistiku poolt etteantud funktsioone vastavalt oma vajadustele. Võimalik on oma soovi järgi kohandada vastavuse, võrdlemise ja ka vastuse teksti genereerimise funktsioone. Võimalus on ka oma testidest fokuseerida ainult seda, mida keegi vajalikuks peab[3] .

2.2 Raamistiku implementeerimine projekti

Jasmine testide käivitamiseks on vaja *npm*'i paketti nimega *jasmine-core*[2] . Tuleb panna tähele, et *npm*'is on mitu *jasmine* kasutamiseks mõeldud paketti nagu näiteks *jasmine, jasmine-node* ja *jasmine-expect.* Ehkki need on mõeldud kasutamiseks *Jasmine* projektides, on neil teine otstarve. Pärast *jasmine-core* paigaldamist on vaja üles leida ning lisada projekti *index.html*'is failid *jasmine.css*, *jasmine.js*, *jasmine-html.js* ja *boot.js*[2] . Projekti juurkaustas on kõik failid leitavad kaustast *node_modules/jasmine-core/lib/jasmine-core*. Lisades antud failid *index.html*'i, on tema sisu sarnane joonises 2 toodule.

```
<html ng-app="app" ng-controller="mainCtrl">
    <head>
        <link rel="stylesheet" type="text/css"
      href="node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
    </head>
    <body>
        <script type="text/javascript" src="node_modules/jasmine-
      core/lib/jasmine-core/jasmine.js"></script>
        <script type="text/javascript" src="node_modules/jasmine-
      core/lib/jasmine-core/jasmine-html.js"></script>
        <script type="text/javascript" src="node_modules/jasmine-
        <script type="text/javascript"
      src="node_modules/angular/angular.min.js"></script>
        <script type="text/javascript" src="node_modules/angular-
      mocks/angular-mocks.js"></script>
        <script type="text/javascript" src="script.js"></script>
        <script type="text/javascript" src="tests.js"></script>
    </body>
</html>
```

Joonis 2. *index.html* pärast *Jasmine* testimiskeskkonna ülesseadmiseks vajalike failide lisamist

Juhin tähelepanu sellele, et antud olukorras on *JavaScripti* failide sisestamise järjekord väga oluline. Vastasel juhul võivad tekkida erinevate teekide vahelised konfliktid, kus üks funktsioon ei leia teist funktsiooni teda välja kutsudes üles.

Eelnevalt käitudes on *Jasmines* testimiskeskkond üles seatud. Testimiskeskkonna graafilist vaadet saab näha avades brauseris faili *index.html*. Avaneb vaade, kus üleval vasakus nurgas kuvatakse *Jasmine* logo ning selle all testide tulemused. Õnnestumisel kuvatakse testid tekstina. Tekst koosneb lausetest, mis on määratud funktsioonide

„describe()” ja „it()” esimestes sisendistes. Testi „põrumisel” kuvatakse punasel taustal kontrolli mitteläbinud test ning kuvatakse veateade.

2.3 Testid

Jasmine testid on toodud Lisas 2, joonisel 12. Vastavalt töö punktis 1.2 toodule testin *AngularJS* rakenduse kahte funktsionaalsust. Esimene on muutuja „$scope.variable” sisu olemasolu kontrollimine, teine on läbi funktsiooni „$scope.getSum()” väljakutsumise muutuja „$scope.sum” väärtuse muutmine. Võrreldes tavalise *JavaScripti* testimisega, muudab *AngularJS* projekti testimise eriliseks enne igat testi *mockitud* mooduli ja kontrolleri genereerimine. Selle jaoks kasutan teeki *angular-mocks*. Tema abil *mockitud* mooduli ja kontrolleri abil on võimalik saada kätte vajalikud „$scope” muutujad.

Võrdlustega seotud süntaks on loodud funktsiooni „expect()” ümber. „Expect()” on kõige sisemine element testimise süsteemis ning lõpetab antud testi. Lisaks funktsioonile „expect()” on olemas arusaadava süntaksiga meetodid, mis aitavad määrata testi täpse võrdluse. Näiteks analüüsides joonisel 3 rida „expect($scope.variable).toBeDefined()” on koheselt märgatav, et testis oodatakse muutuja „$scope.variable” sisu määratust.

```
describe('declared value at variable', function() {
    it('should not be null', function() {
        var $scope = {};
        $controller('mainCtrl', {$scope: $scope});
        expect($scope.variable).toBeDefined();
    });
});
```

Joonis 3. *Jasmine* testimisraamistiku esimene test

2.4 Hinnang

Jasmine on võrdlemisi kergelt paigaldatav ning kasutatav testimisraamistik *JavaScriptile*. Tegemist on kiiresti õpitava ja kerge vaevaga tööle saadava

testimisraamistikuga. Peale *angular-mocks*'i on *Jasmine* raamistikul kõik testimiseks vajalik sisse programmeeritud, näiteks testide sisestamiskäskluste kasutus *expect* süntaksi näol. Hindan *Jasmine* testimisraamistikku minu poolt punktis 1.3 pakutud kriteeriumite alusel.

Kriteerium 1. Kriteerium käsitleb testide skaleeritavuse ja struktureerimise võimalusi. Funktsioonid „describe()" ja „it()" koos nende sisu kirjeldava sisendiga võimaldavad testitava koodi väga lihtsalt loodavaks ja arusaadavaks teha. Nende abil on võimalik oma projekti teste struktureerida vastavalt vajadusele. Samuti on võimalik „describe()" funktsiooni korduvalt iseenda sees välja kutsuda, mis aitab kaasa testide grupeerimise, struktureerimise ja skaleerimise võimalusele ja arusaadavusele suure testide hulga korral. Kriteeriumi hinne 5.

Kriteerium 2. Kriteerium käsitleb süntaksi ja graafilise vaate loetavust ning arusaadavust. Jasmine on loogiliselt üles ehitatud süsteem, kus teste on lihtne kirjutada ja nende käitumine arusaadavalt jälgitav. Tänu teste hästi kirjeldavatele funktsioonide nimedele on väga lihtne aru saada, isegi ilma testi kirjeldava tekstita, mida antud testiga tahetakse saavutada. Süntaksi poolest on puuduseks võimalus ainult *expect* märksõna sisestussüntaksit kasutada. Testide tulemusi kokku võttev graafiline vaade on samuti kergelt loetav ja arusaadav. Puuduseks tuleb lugeda seda, et erinevalt *Mochast* ei ole *Jasmine* graafilises vaates võimalik lugeda antud testi sisaldavat koodi. Antud kriteerium on minu arvates hästi täidetud. Kriteeriumi hinne 3.

Kriteerium 3. Kriteerium käsitleb keskkonna ülesseadmise lihtsust ja mugavust. Jasmine testimisraamistikku on lihtne siduda *AngularJS* projektiga. Selleks on vajalik lisaks *angular-mocks*'ile alla laadida pakett *jasmine-core*, lisada projekti skriptide hulka punkti 2.2 toodud kolm faili ja enne igat testi genereerida *mockitud* moodulid. *Jasmine* näol on tegemist suhteliselt täieliku testimisraamistikuga, kus kasutajale jäetakse keskkonna ülesseadmise osas minimaalselt kohustusi. Negatiivseks küljeks tuleb lugeda sõltuvus *angular-mocks*'ist. Kriteeriumi hinne 4.

Kriteerium 4. Kriteerium käsitleb lisavõimalusi ja ühilduvaid teeke *Jasmine* testimisraamistikus, toetudes dokumentatsioonile. *Jasmine* on minu arvates võimalikult täielik testimisraamistik. Tema abil on võimalik teha nii asünkroonseid päringuid, vastavalt soovile muuta võrdlemiskriteeriumeid kui ka skriptis esile tõsta soovitud

testid. Samas on *Jasmine* dokumentatsioon ülejäänud abivahendite koha pealt puudulik. Jääb mulje, et *Jasminega* on seotud suhteliselt vähe teeke. Võttes arvesse, et paljud arendajad ootavad raamistikelt palju valikuvõimalusi oma keskkonna konfigureerimiseks vastavalt vajadusele, leian, et kriteerium on täidetud puudulikult. Kriteeriumi hinne 2.

Jasmine on suhteliselt täielik testimisraamistik, kuid puudusena jäetakse kasutajale vähe võimalusi keskkonda oma soovi järgi konfigureerida. Arvestades keskkonna ülesseadmise lihtsust on samas arusaadav, miks *AngularJS* dokumentatsioon soovitab kasutada *Jasminet* ühiktestide kirjutamisel. *Jasmine* kriteeriumite hinded on nähtavad tabelis 4.

Tabel 2. *Jasmine* testimisraamistiku hinded

Kriteerium	**Hinne**
Skaleeritavuse ja struktureerimise võimalus	5
Süntaks ja graafiline vaade	3
Keskkonna ülesseadmine	4
Lisavõimalused ja ühilduvad teegid	2

3 Mocha

3.1 Raamistiku tutvustus

Mocha on vabavaraline *MIT* litsentsil põhinev testimisraamistik *JavaScriptile*. *Mocha* töötab *Node.js* platvormil, kuid võimaldab ka brauseri, asünkroonnsete testide ja paljude teekide ühildamist. *Mocha* dokumentatsioon soovitab kasutada ainuüksi testide sisestamisteegina kokku viite erinevat teeki[4] .

Mocha koodi on lihtne struktureerida, see võimaldab projektide teste skaleerida vastavalt vajadusele. Nagu *Jasmines*, koosneb ka *Mocha* süntaks kahest teste kirjeldavast ja struktureerivast funktsioonist „describe()" ja „it()"[4] . Sarnasus raamistike vahel jätkub samuti kahe funktsiooni sisendites. Esimene sisend on mõlemal tekst, mis kirjeldab plokile vastava sisu tegevust. Teine sisend on funktsioon, mis määrab antud ploki sisu.

Erinevalt *Jasminest*, on *Mochas* võimalik kerge vaevaga kasutada *expect* süntaksi asemel teiste teekide abil süntaksi poolest erinevaid võrdlemise käsklusi. Selleks on vaja *Mochaga* koos kasutada mõnda teist testide sisestamisteeki. Märksõna *expect* kasutatakse teegis *expect.js*. Kasutades teeki *should.js*, jääb testide põhimõte samaks, kuid märksõna *expect* asemel kasutatakse märksõna *should*. Üks populaarsemaid sisestamisteeke, mida *Mochaga* koos kasutatakse, on *Chai*. Tema tugevus teiste teekide ees tuleneb *Chai* võimalustest sisestada teste nii *should*, *expect* kui ka *assert* märksõnasid kasutades. Tänu laiale valikuvõimalusele tööriistu vastavalt vajadusele konfigureerida, saab iga testide kirjutaja valida oma projektile kõige sobivama kasutajaliidese[4] .

Mochas teste kirjutades on palju võimalusi erinevate stsenaariumite testimiseks. *Mocha* ametlikus dokumentatsioonis käsitletakse palju erinevaid näiteid koodi testimiseks.

3.2 Raamistiku implementeerimine projekti

Testide käivitamiseks kasutan testitavas projektis lisaks *Mochale* ka testide sisestamise teeki *expect.js*. Sellega tagan *Mocha* testide sarnasus *Jasminele*, mis teeb nende vaheliste erinevuste võrdlemise lihtsamaks[5] . Nagu varem mainitud, ei teki erinevate sisestusteekide kasutamisel suuri sisulisi erinevusi. Nii *Mocha* kui ka *expect.js*'i pakettide paigaldamiseks on võimalik kasutada *npm*'i[4] , [5] . Siin on vaja pöörata tähelepanu kahele asjaolule. Esiteks ei ole brauserist testide kokkuvõtet vaadates võimalik *Mochat* paigaldada globaalselt, nagu tema dokumentatsioon ütleb. Soovitan sel juhul paigaldada *Mocha npm*'i vaikimisi kausta. Globaalselt paigaldamine on mõeldud testide käivitamiseks käsurealt. Teiseks probleem on *expect.js*'i paigaldamine. Tähele tuleb panna, et installeerides paketti nimega *expect.js*, jääb paketi nimesse punkt sisse. Vale on paigaldada pakett *expectjs* või *expect*.

Mocha ja *expect.js* kaasamiseks projekti on vaja lisaks projekti failidele ja testidele sisestada veel kolm faili. Nendest kaks, *mocha.css* ja *mocha.js*, asuvad juurkausta suhtes kaustas *node_modules/mocha*. *Expect.js* kasutamiseks tuleb *index.html*'i lisada *index.js*, mis asub juurkausta suhtes kaustas *node_modules/expect.js*.

Mocha testide käivitamiseks brauseris peab kirjutama, et *Mocha* kasutaks testide käivitamiseks *BDD* liidest. Selleks tuleb pärast *Mocha* sisestamist käivitada skriptirida „*mocha.setup('bdd')*”. *Mocha* käivitamiseks tuleb pärast testi failide sisestamist käivitada skript „*mocha.run()*”. Need kaks skripti on failide kaasamise järjekorra seisukohast väga olulised. Viimane, mis lisatakse, peab olema *Mocha* failide ja testide suhtes alati „m*ocha.run()*”. Samas peab „*mocha.setup('bdd')*” olema asetatud pärast *mocha.js* lisamist ning enne faili *angular-mocks.js* ja testide lisamist[4] . Graafilises vaates *Mocha* testide tulemuste vaatamiseks on vaja lisada *index.html* faili tekstisektsioon, millele on lisatud globaalne atribuut *id* väärtusega „mocha”. Koodi näide eelneva kohta on toodud joonises 4.

```
<html ng-app="app" ng-controller="mainCtrl">
    <head>
        <link rel="stylesheet" type="text/css"
      href="node_modules/mocha/mocha.css" />
    </head>
    <body>
        <div id="mocha"></div>
        <script type="text/javascript"
      src="node_modules/mocha/mocha.js"></script>
        <script type="text/javascript"
      src="node_modules/expect.js/index.js"></script>
        <script>
            mocha.setup('bdd')
        </script>
        <script type="text/javascript"
      src="node_modules/angular/angular.min.js"></script>
        <script type="text/javascript" src="node_modules/angular-
      mocks/angular-mocks.js"></script>
        <script type="text/javascript" src="script.js"></script>
        <script type="text/javascript" src="tests.js"></script>
        <script>
            mocha.run()
        </script>
    </body>
</html>
```

Joonis 4. *index.html* pärast *Mocha* testimiskeskkonna ülesseadmiseks vajalike failide lisamist

Nii toimides on testimiskeskkond üles seatud. Testimiskeskkonna graafilist vaadet näeb avades brauseris faili *index.html*. Avaneb vaade, kus üleval paremal nurgas on kokkuvõte testide läbimisest ja ajakulust. Vasakule on joondatud testide kirjeldused. Testi ebaõnnestumise korral on ekraanil punasega välja toodud sektsioonid, mis kirjeldavad ebaõnnestumise põhjuseid. Suur pluss ja erinevus *Jasminest* on see, et klikkides „it()” poolt genereeritavale tekstile, on võimalik näha selles oleva testi koodi.

3.3 Testid

Mocha testid on nähtavad Lisas 3, joonisel 14. *Mocha* testid on antud projektis sarnased *Jasmine* omadele. Sarnasus tuleneb peamiselt sellest, et sisestusteegina kasutatakse *expect.js*'i. Kasutades *Mocha* kõrval mõnda muud sisestusteeki, näiteks *should.js*'i,

näeksid testid süntaksi poolest *Jasminest* oluliselt teistsugused välja. Sisestusteekide abil saab muuta testid täpselt selliseks nagu on vaja. Antud projekti raames valisin *expect.js* sisestusteegi. Tegemist on võrdlemisi kergekaalulise teegiga, tema süntaksi sarnasus *Jasminega* aitab nende sisulisi pooli võrrelda. Süntaksi poolest on *Mochale* omane kasutada võrdlemisel funktsioonide vahel punkte.

Kirjutades *AngularJS* projektile teste *Mocha* abil, tuleb jälgida, et *angular-mocks* looks enne igat testi *mockitud* objektid, mida omakorda saab testida. *Mockitud* objektide abil on võimalik, nagu ka *Jasmines*, kätte saada vajalikud muutujad ja funktsioonid.

3.4 Hinnang

Mocha paigaldamine projekti võib nõuda võrreldes teiste testimisraamistikega rohkem aega ja vaeva. See tasub ennast ära kui kasutajal on vaja keskkond enda vajadustele vastavalt üles seada. Hindan antud testimisraamistikku minu poolt punktis 1.3 pakutud kriteeriumite alusel.

Kriteerium 1. Kriteerium käsitleb testide skaleeritavuse ja struktureerimise võimalusi. Kuigi sisestamisteegi saab *Mocha* puhul ise valida, on siiski „describe()" ja „it()" funktsioonid *Mochasse* sisse ehitatud. Nagu *Jasmines*, saab ka *Mochas* „describe()" funktsiooni korduvalt iseenda sees välja kutsuda. See tagab parema grupeerimise, struktureerimise ja skaleeritavuse võimaluse. Kriteeriumi hinne 5[4] .

Kriteerium 2. Kriteerium käsitleb süntaksi ja graafilise vaate loetavust ning arusaadavust. Süntaksi loetavust on *Mochas* raske hinnata, sest igal kasutajal on võimalus valida oma projekti sobiv testide sisestusteek. Samas annab see *Mochale* teiste testimisraamistike ees eelise, sest tekib võimalus valida teek, mis tundub kasutajale kõige lihtsam ja arusaadavam. Brauseris avatav graafiline vaade on *Mochal* väga hästi struktureeritud ja lihtsalt loetav. Vajadusel on võimalik vaadata ainult soovitud testi. Graafilises vaates on võimalik vaadata testi koodi. Kriteeriumi hinne 5.

Kriteerium 3. Kriteerium käsitleb keskkonna ülesseadmise lihtsust ja mugavust. Keskkonna enda soovi järgi ülesseadmisel võivad probleemid lihtsalt tekkida. Esiteks peab otsustama, millist sisestusteeki oma projekti juures kasutada[4] . See võib võtta

palju aega kui ei taheta pealiskaudselt ühele sisestusteegile pühenduda. Teiseks on *Mocha* kasutamine *AngularJS* projektis sõltuv *angular-mocksist*[9] . Sõltudes mitmest teegist on suurem oht igat liiki probleemide tekkeks. Võttes arvesse võimalikke riske, leian, et antud kriteerium on täidetud rahuldavalt. Kriteeriumi hinne 3.

Kriteerium 4. Kriteerium käsitleb lisavõimalusi ja ühilduvaid teeke *Mocha* testimisraamistikus, toetudes dokumentatsioonile. *Mochal* on palju erinevaid võimalusi sisestamissüntaksi ja ka teiste testimiskeskkonna tööriistade seadistamiseks. See teeb temast väga paindliku testmisraamistiku. *Mocha* dokumentatsioon on arusaadav ning jagab vajalikku informatsiooni erinevate võimaluste, funktsioonide ja teekide kohta. Kriteeriumi hinne 5[4] .

Mocha on väga hea tesimisraamisik teadlikule arendajale. Selleks peab arendaja täpselt teadma, mida ta testimiskeskkonnalt ootab. Lisaks pakub *Mocha* dokumentatsioon piisavalt võimalusi vajalik informatsioon kätte saada[4] . Kasutajale, kellele on *JavaScripti* ühiktestimine võõras või kes ei soovi testimisraamistiku paigaldamisega vaeva näha, ei ole *Mocha* tõenäoliselt parim valik. *Mocha* kriteeriumite hinded on nähtavad tabelis 3.

Tabel 3. *Mocha* testimisraamistiku hinded

Kriteerium	**Hinne**
Skaleeritavuse ja struktureerimise võimalus	5
Süntaks ja graafiline vaade	5
Keskkonna ülesseadmine	3
Lisavõimalused ja ühilduvad teegid	5

4 QUnit

4.1 Raamistiku tutvustus

QUnit on *JavaScripti* ühiktestimisraamistik, mis on loodud *JavaScripti* ühe suurima teegi *jQuery* testimiseks. Ühtlasi on sellega võimalik testida kõiki *JavaScripti* sisaldavaid koode. *QUnit* loodigi ajalooliselt olema osaks *jQuery*'st. 2008. aastal nad eraldati, kuid mõndades aspektides oli *QUnit* endiselt *jQuery*'st sõltuv. Nüüdseks on see parandatud ning *QUnit* on täielikult iseseisev *JavaScripti* testimisraamistik[7] .

Oma süntaksi poolest on *QUnit* täiesti erinev *Jasminest* ja *expect.js*'ist. Esimene suur erinevus seisneb selles, et enne igat *QUniti* funktsiooni välja kutsumist tuleb funktsiooni viite ees välja kutsuda globaalne muutja „QUnit". Seejärel tuleb kirjutada punkti taha, millist funktsiooni kasutada soovitakse. Teine erinevus *QUnitit* kasutades on see, et funktsioonide „describe()" ja „it()" süntaksi asemel kasutatakse testi kirjeldamiseks funktsioone „module()", „test()" ja võrdlemisfunktsiooni, näiteks „equal()" või „ok()", viimast sisendit. Kui funktsioon „module()" on testide grupeerimiseks, siis funktsioon „test()" täidab *QUnitis* funktsiooni „describe()" rolli ja võrdlemisfunktsiooni viimane sisend funktsiooni „it()" rolli[7] .

QUnitis kasutatakse testide sisestussüntaksina *assert* märksõna, millele järgneb testi võrdluse osa kirjeldav funktsioon[7] . Märksõna *assert* ei ole omane ainult *QUnitile*. Seda kasutatakse veel näiteks *Java* testimisraamistikus *JUnit*, mille kohta leidub palju viiteid *QUniti* dokumentatsioonis. Samuti kasutatakse *assert* märksõna ka *JavaScripti* testide sisestamisteegis *Chai*[10] .

Lisaks eelpool mainitud funktsioonidele pakub *QUnit* veel suurt hulka erinevaid meetodeid oma testimiskogemuse võimalikult mugavaks ja arusaadavaks tegemisel.

4.2 Raamistiku implementeerimine projekti

Testimiskeskkonna implementeerimiseks projekti on vaja kahte faili: *qunit.css* ja *qunit.js*. Mõlemad on saadavad *npm*'i paketis *qunitjs*[7] . Pärast paigaldamist on failid leitavad projekti juurkausta alamkaustas *node_modules/qunitjs/qunit*. *QUniti JavaScripti* faili lisamisel projekti tasub meeles pidada, et see oleks projekti lisatud enne testide faili. Vastasel juhul ei leia *QUnit* teste üles.

Teiste testimisraamistikega võrreldes on *QUnitiga* töötamise eeliseks veel see, et *AngularJS* projekti testimiseks ei ole vaja paketti *angular-mocks*[11] . See teeb minimaalseks testimisraamistiku sõltuvuse teistest komponentidest. Võimalikult väike sõltuvus erinevatest komponentidest on tarkvaraarenduses eeliseks, kuna nii vähendatakse protsesside ebaõnnestumise võimalikke põhjuseid.

Sarnaselt *Mochale* kuvab *QUnit* oma testide tulemused *index.html* faili tekstisektsiooni. Sellele peab olema lisatud globaalne atribuut *id* väärtusega „qunit". See aitab graafilises vaates paigutada testimistulemused õigesse kohta[7] . Koodi näide eelneva kohta on toodud joonises 5.

```
<html ng-app="app" ng-controller="mainCtrl">
    <head>
        <link rel="stylesheet" type="text/css"
      href="node_modules/qunitjs/qunit/qunit.css" />
    </head>
    <body>
        <div id="qunit"></div>
        <script type="text/javascript"
      src="node_modules/jquery/dist/jquery.min.js"></script>
        <script type="text/javascript"
      src="node_modules/qunitjs/qunit/qunit.js"></script>
        <script type="text/javascript"
      src="node_modules/angular/angular.min.js"></script>
        <script type="text/javascript" src="script.js"></script>
        <script type="text/javascript" src="tests.js"></script>
    </body>
</html>
```

Joonis 5. *index.html* pärast *QUniti* testimiskeskkonna ülesseadmiseks vajalike failide lisamist

Nii on testimiskeskkond üles seatud. Testimiskeskkonna graafilist vaadet näeb avades brauseris faili *index.html*. Avaneb sinakates toonides vaade, kus on näha testide tulemused. Ebaõnnestunud testid kuvatakse laias punases raamis. Sinna kuvatakse testi sisu, oodatav väärtus ja väärtus mis saadi testi käivitamisel. Samuti tuuakse välja fail ja faili rida, kus tekib testis probleem.

4.3 Testid

QUniti testid on toodud Lisas 4, joonisel 8. Kuna *QUnit* ei sõltu *AngularJS* projekti testimisel *angular-mocks*'ist, on *AngularJS* projekti objektide *mockimise* kood erinev kui *Jasmines* ja *Mochas*. Alustuseks leitakse nime abil moodul. Siis võetakse moodulist „$rootScope" ja küsitakse moodulilt kontrollerit. Seejärel seotakse antud kontroller „$scopega" ära. Eelnevast järeldub, et objektide *mockimise* põhimõte on sama kui *angular-mocksi* kasutades[11] .

QUnit kasutab testide sisestamissüntaksina märksõna *assert*[7] . See tähendab, et testid põhinevad objektide sisestamisel ning seejärel nende võrdlemises. *QUnitile* on omane enne iga funktsiooni välja kutsumist see spetsiaalselt ära märkida. Selleks kasutatakse enne funktsiooni globaalset muutujat „QUnit", mis annab *QUnitile* märku, et kutsutakse välja tema funktsioon[7] . Seejärel eraldatakse globaalne muutuja ja temalt küsitav funktsioon punktiga. On mõistetav, et koondamaks kõik funktsioonid ühte kohta, lõid *QUniti* arendajad globaalse muutuja „QUnit". Suure koguse testide kirjutamisel muutub aga globaalse muutuja trükkimine enne iga testi tüütuks. Samuti väheneb koodi loetavus. Lisaks väheneb *QUnitis* koodi loetavus ka võrdluste ja teksti sisestamisel testi. Kasutades näiteks võrdlemiseks funktsiooni „equal()", tuleb funktsiooni esimeseks sisendiks kaasa anda testitav väärtus, teiseks sisendiks oodatav väärtus ja kolmandaks testi sisu kirjeldav tekst. Võrreldes seda näiteks *Jasmine* ja *expect.js* süntaksiga, kus on selgelt eristatav, mida mingi sisend teeb, on *QUniti assert* süntaksit keerulisem lugeda. Eelnevat illustreerib hästi joonisel 6 olev summa funktsionaalsuse kontrollimiseks kirjutatud test.

```
QUnit.test('sum calculator', function() {
    scope.x = 2;
    scope.y = 4;
    scope.getSum();
    QUnit.assert.equal(scope.sum, 6, '2 + 4 should equal 6');
});
```

Joonis 6. *QUnit* testimisraamistiku esimene test

4.4 Hinnang

QUnit on väga kergesti paigaldatav ja kasutatav raamistik *AngularJS* projekti testimiseks. *QUnit* püüab olla võimalikult täiuslik testimisraamistik. Seda iseloomustab näiteks see, et *AngularJS* projekti testimiseks ei pea kasutama *angular-mocksi*[11] . Projekti moodulite *mockimise* võimalus on *QUnitisse* juba sisse ehitatud. Samas tekitab soov ideaalne olla probleeme testide kokkuvõtte graafilises vaates. *QUnit* kuvab testide kokkuvõttes liiga palju informatsiooni, millest mingi osa on testide analüüsi seisukohalt ebavajalik. Hindan antud testimisraamistikku minu poolt punktis 1.3 pakutud kriteeriumite alusel.

Kriteerium 1. Kriteerium käsitleb testide struktureerimise ja skaleeritavuse võimalusi. Esmamulje on, et *QUniti* teste on võrreldes teiste testimisraamistikega lihtsam struktureerida, sest *QUnitis* on selleks kolm teste kirjeldavat funktsiooni võrreldes *Jasmine* ja *Mocha* kahe teste kirjeldava funktsiooniga[7] , [3] , [5] . Tegelikkuses selgub aga, et *QUnitis* saab teste grupeerides ja stuktureerides kasutada ainult funktsiooni „module()” sees funktsiooni „test()”. Funktsioonide välja kutsumine grupeerimise eesmärgil ise funktsioonide enda sees graafilises vaates ei kajastu. See erineb *Jasmine* ja *Mocha* „describe()” funktsioonist. Seega väheneb *QUniti* testide grupeerimisvõimalus. Minu hinnangul kannatab „tänu” sellele ka skaleeritavus. Kriteeriumi hinne 4.

Kriteerium 2: Süntaksi loogilisus ja graafilise vaate arusaadavus. Punktis 4.3 analüüsisin, et *QUniti* teste on keeruline lugeda tänu testides kasutatavate võrdlemisfunktsioonide ülesehitusele. Samuti on *QUniti* graafilises vaates liiga palju ebavajalikku informatsiooni, vajalik informatsioon jääb raskelt leitavaks ning loetavaks. Antud kriteerium ei ole minu arvates hästi täidetud. Kriteeriumi hinne 2.

Kriteerium 3. Kriteerium käsitleb keskkonna ülesseadmise lihtsust ja mugavust. Võttes arvesse punktis 4.2 analüüsitut, võin väita, et *QUniti* keskkonna ülesseadmine on lihtne. Vaja on kõigest ühte paketti, mis sisaldab kogu vajaliku *AngularJS* projekti testimiseks. Seega väidan, et kriteerium on täidetud väga hästi. Kriteeriumi hinne 5.

Kriteerium 4. Kriteerium käsitleb lisavõimalusi ja ühilduvaid teeke *QUniti* testimisraamistikus, toetudes dokumentatsioonile. Kuigi *QUnit* ei võimalda testide sisestussüntaksit muuta, evib ta palju muid võimalusi keskkonna ülesseadmiseks oma vajaduse järgi. Need võimalused on välja toodud *QUniti* hästi koostatud dokumentatsioonis, kus tuuakse välja kõik funktsioonid, mida *QUnitiga* töötades on võimalik kasutada. Lisana tuuakse näide iga funktsiooni kasutamise kohta. Kriteeriumi hinne 4.

QUnit on üldjoontes hästi organiseeritud *assert* märksõnal põhinev testimisraamistik *JavaScriptile*. Teda on lihtne implementeerida *AngularJS* projekti. Selleks on vaja ainult kahte faili. Kui süntaksi ebamugavused välja arvata, on tegemist väga hea testimisraamistikuga. *QUniti* kriteeriumite hinded on nähtavad tabelis 4.

Tabel 4. *QUnit* testimisraamistiku hinded

Kriteerium	**Hinne**
Skaleeritavuse ja struktureerimise võimalus	4
Süntaks ja graafiline vaade	2
Keskkonna ülesseadmine	5
Lisavõimalused ja ühilduvad teegid	4

5 Järeldused

5.1 Testmisraamistike tugevad ja nõrgad küljed

AngularJS raamistik soovitab oma dokumentatsioonis kasutada ühiktestimiseks testimisraamistikku *Jasmine*. Minu analüüsi põhjal selgub, et kuigi *Jasmine* testimisraamistikku on lihtne paigaldada, on temal teatud puudujääke. Suurimaks puuduseks pean võimaluste vähesust, sest *Jasminega* ühildub vähe teeke ja tema dokumentatsioon on puudulik[1] .

Mochal puuduvad *Jasmine* nõrkused. *Mochal* on hea dokumentatsioon, mis sisaldab lahendusi ja näiteid erinevatele probleemidele. Lisaks on *Mocha* jaoks loodud palju ühilduvaid teeke. *Mocha* tugevaim külg on võimalus valida endale sobiv sisestamissüntaksi teek[4] .

QUnit on kui omapärane kombinatsioon *Jasminest* ja *Mochast*. *QUnitil* on hea dokumentatsioon ning samas on *QUnitit* lihtne paigaldada. *QUniti* tugevaim külg on aga tema kasutamise võimalused *AngularJS* projektis. Erinevalt *Jasminest* ja *Mochast* ei vaja *QUnit AngularJS* projekti objektide *mockimiseks* teeki *angular-mocks*. See vähendab *QUniti* sõltuvust välistest teguritest. *QUniti* nõrgemaks küljeks on samas tema pikk ja raskelt loetav süntaks[7] .

Kõigi testimisraamistike hindeid kokkuvõttev ülevaade on nähtav tabelis 5.

Tabel 5. Kõiki raamistikke kokkuvõtvad hinded

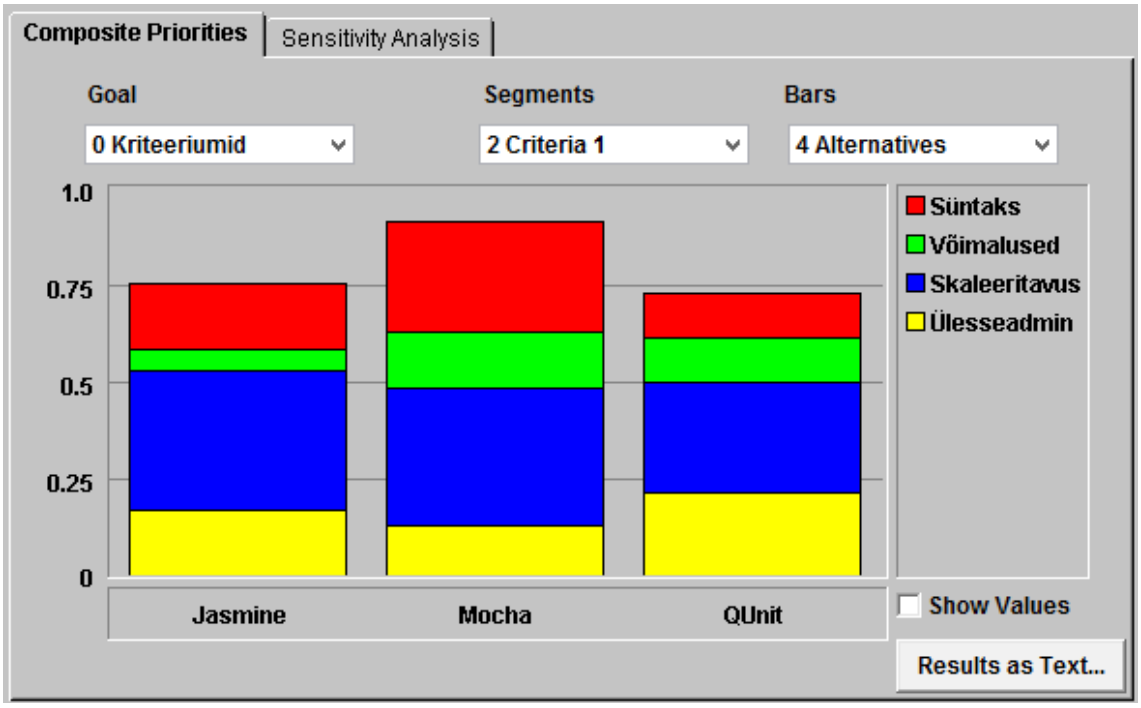
Kriteerium	**Jasmine**	**Mocha**	**QUnit**
Skaleeritavuse ja struktureerimise võimalus	5	5	4
Süntaks ja graafiline vaade	3	5	2
Keskkonna ülesseadmine	4	3	5

Kriteerium	Jasmine	Mocha	QUnit
Lisavõimalused ja ühilduvad teegid	2	5	4

5.2 Testimisraamistike analüüsimine ja tulem

Testimisraamistike analüüsiks antud töös kasutan *Java* põhist programmi *Web-Hipre*, mis toetab mitmemõjuriliste otsustusprobleemide lahendamist. *Web-Hipre* on loodud Helsingi tehnikaülikooli poolt ja on akadeemilistel eesmärkidel kasutamiseks tasuta.

Web-Hipre abil saan analüüsida ja luua seosed töös käsitletud testimisraamistike ja kriteeriumite vahel. *Web-Hipre* arvestab oma analüüsis iga testimisraamistiku kriteeriumi hinnet ja igale kriteeriumile vastavat olulisuse hinnangut. Analüüsiks loon *Goal* gruppi ühe elemendi nimega „Kriteeriumid”. Lisaks loon *Criteria 1* gruppi neli elementi „Süntaks”, „Ülesseadmine”, „Võimalused” ja „Skaleeritavus”, mis tähistavad punktis 1.3 välja toodud kriteeriumeid. *Alternatives* gruppi loon kolm elementi „Jasmine”, „Mocha” ja „QUnit”, mis tähistavad analüüsitavaid testimisraamistikke. Kui elemendid on gruppidesse jaotatud, loon nendevahelised seosed. Seosed loon *Goal* ja *Criteria 1* kõikide elementide vahel, samuti *Criteria 1* ja *Alternatives* kõikide elementide vahel. Olulisuse hinnangute andmiseks määran elemendile „Kriteeriumid” topeltklikkides kriteeriumitele olulisuse hinnangu arvnäitajad tuginedes punktis 1.3 toodule. Järgmiseks sisestan iga testimisraamistiku iseloomustamiseks hinded, mille ma iga kriteeriumi täitmise eest neile andsin. Antud töös on kriteeriumite täitmise hinded nähtavad iga testimisraamistiku peatüki viimases alapunktis. Hinded saab sisestada *Criteria 1* elementidel topeltklikkides. Avades näiteks elemendi „Süntaks” sisu vaate, on näha kõik *Alternatives* grupi elemendid ehk testimisraamistikud. Testmisraamistiku nime taga olevasse sisestusvälja kirjutan kümnendmurruna hinde, mille testimisraamistik antud kriteeriumi eest sai. Antud metoodikat kasutades ja täites kõigi grupi *Criteria 1* elemendite sisud, saab testimisraamistikud hinnatud. Web-Hipre koostatud analüüsi raportit näeb ülemisest menüüst „Analysis” alammenüü elemendi „Composite Priorities” peal klikkides. Graafiliselt on analüüs nähtav joonisel 7.



Joonis 7. Analüüsi graafiline vaade

Joonisel 7 paremal all nurgas oleval nupul nimega „Results as Text" klikkides on analüüsi tulemused nähtavad ka teksti kujul, kus lõplikud tulemused on kuvatud kümnendmurruna. Analüüsi tulemused tekstina on toodud joonisel 8.

```
Value Tree
0 Kriteeriumid
    2 Skaleeritavus 0.357
        4 Jasmine 1.000
        4 Mocha 1.000
        4 QUnit 0.800
    2 Ülesseadmine 0.214
        4 Jasmine 0.800
        4 Mocha 0.600
        4 QUnit 1.000
    2 Süntaks 0.286
        4 Jasmine 0.600
        4 Mocha 1.000
        4 QUnit 0.400
    2 Võimalused 0.143
        4 Jasmine 0.400
        4 Mocha 1.000
        4 QUnit 0.800

Composite Priorities
           Jasmine    Mocha      QUnit
Süntaks    0.171      0.286      0.114
Võimalused 0.057      0.143      0.114
Skaleerita 0.357      0.357      0.286
Ülesseadmi 0.171      0.129      0.214
Overall    0.757      0.914      0.729
```

Joonis 8. Analüüsi tulemused tekstina

Analüüsi tulemusena selgub, et parim raamistik *AngularJS* projekti ühiktestimiseks on *Mocha*. Tema koondhinne on 0,914. Paremuselt teine testimisraamistik on *Jasmine*, saades hindeks 0,757. Testimisraamistik *QUnit* sai kolmanda tulemuse, hindeks 0,729.

Võrdlustabelist selgub, et *Mocha* peaks olema *AngularJS* projekti testimiseks põhieelistus, kuna tema koondhinne on oluliselt kõrgem võrreldes teiste testimisraamistikega. Nagu eelnevalt korduvalt mainitud, soovitab *AngularJS* dokumentatsioon *Jasmine*'i, mille koondhinnete erinevus võrreldes *QUnitiga* on minimaalne. Seega soovitan analüüsi tulemusena *AngularJS* projekti testimiseks kasutada *Mochat* kui kõige efektiivsemat testimisraamistikku.

6 Kokkuvõte

Antud bakalaureusetöö sisuks on *JavaScripti* testimisraamistike *Jasmine*, *Mocha* ja *QUniti* kasutamise võrdlus veebiraamistiku *AngularJS* rakenduses. Töös võrreldi nimetatud testimisraamistikke autori määratud ja põhjendatud kriteeriumite alusel, kasutades kokkuvõtvaks analüüsiks Helsingi tehnikaülikooli poolt loodud programmi *Web-Hipre*. Töö tulemusena selgus, et parim testimisraamistik *AngularJS* projekti testimiseks on testimisraamistik *Mocha*. Saavutatud tulemus erineb *AngularJS* dokumentatsioonis kasutada soovitatud testimisraamistikust, milleks pakutakse testimisraamistikku *Jasmine*. Selline soovitus on ka mõneti mõistetav, kuna tegemist on populaarse testimisraamistikuga ja tema kohta on veebis saadaval palju informatsiooni.

Antud bakalaureusetöö raames tehtud analüüsi tulemusi saab oluliselt laiendada kriteeriumi lisamisega, mis mõõdab testimisraamistiku töö kiirust. Kiiruse erinevuste paremaks selgitamiseks on seega vaja testitavat projekti oluliselt suurendada. Eelnevale lisaks peab testides pöörama tähelepanu samuti vähemolulistele testimisraamistike võimalustele. See nõuaks aga käesoleva töö olulist laiendamist.

Kasutatud kirjandus

[1] Google, “AngularJS – Superheroic JavaScript MVW Framework”, 2010-2016, [Online]. Available: https://angularjs.org/ [Accessed 21 May 2016].

[2] Pivotal Labs, “jasmine-core - npm”, 2008-2015, [Online]. Available: https://www.npmjs.com/package/jasmine-core [Accessed 21 May 2016].

[3] Pivotal Labs, “intoduction.js - Jasmine”, [Online]. Available: http://jasmine.github.io/edge/introduction.html [Accessed 21 May 2016].

[4] “Mocha – the fun, simple, flexible JavaScript test framework”, May 2016, [Online]. Available: https://mochajs.org/ [Accessed 21 May 2016].

[5] Rauch G., “Expect.js”, 2011, [Online]. Available: https://github.com/Automattic/expect.js [Accessed 21 May 2016].

[6] Veskioja T., “Lühike Web-Hipre kasutusjuhend eesti keeles”, [Online]. Available: http://maurus.ttu.ee/ained/IDN5120/doc/20/Web_Hipre_juhend.html [Accessed 21 May 2016].

[7] “QUnit: A JavaScript Unit Testing framework”, [Online]. Available: http://qunitjs.com/ [Accessed 21 May 2016].

[8] Watmore J., “Unit Testing in AngularJS with Mocha, Chai, Sinon & ngMock – So many libraries, what does what?”, April 2015, [Online]. Available: http://jasonwatmore.com/post/2015/04/09/Unit-Testing-in-AngularJS-So-many-libraries-what-does-what.aspx [Accessed 21 May 2016].

[9] Scott B., “Setting up Karma with Mocha, PhantomJS and Chai”, February 2015, [Online]. Available: http://bendetat.com/karma-and-mocha-for-angular-testing.html [Accessed 21 May 2016].

[10] “Chai Assertion Library”, [Online]. Available: http://chaijs.com/ [Accessed 21 May 2016].

[11] O'Mara J., “AngularJS testing with QUnit”, November 2013, [Online]. Available: http://www.pittrap.com/angularjs-testing-qunit/ [Accessed 21 May 2016].

[12] Bailey S., “AngularJS Testing Cookbook”, Birmingham, UK, March 2015, [Online]. Available: https://books.google.ee/books?id=nda6BwAAQBAJ&lpg=PP1&dq=angularjs%20unit%20test&lr&pg=PA42#v=onepage&q&f=false [Accessed 21 May 2016].

[13] Saleh H., “JavaScript Unit Testing”, Birmingham, UK, January 2013, [Online]. Available: https://books.google.ee/books?

hl=en&lr=&id=bjw0IHfUCzMC&oi=fnd&pg=PT8&dq=javascript+unit+testing&ots=dSuNpUAxjU&sig=JBU1n0ip-1ZlmTvoJSuPedNFIxk&redir_esc=y#v=onepage&q&f=false [Accessed 21 May 2016].

[14] Kastegård S., "Automated testing of web-based user interface", Linköpings University, Sweden, June 2015, [Online]. Available: http://www.diva-portal.org/smash/get/diva2:819982/FULLTEXT01.pdf [Accessed 21 May 2016].

[15] Braithwaite B., "Getting started with Unit Testing for AngularJS", May 2015, [Online]. Available: http://www.bradoncode.com/blog/2015/05/12/angularjs-testing-getting-started/ [Accessed 21 May 2016].

Lisa 1 – Testitav rakendus

```
<html ng-app="app" ng-controller="mainCtrl">
      <head>
            <!-- Angular -->
            <script type="text/javascript"
src="node_modules/angular/angular.min.js"></script>
            <!-- Angular controller -->
            <script type="text/javascript" src="script.js"></script>
            <!-- Tests -->
            <script type="text/javascript" src="tests.js"></script>
      </head>
      <body></body>
</html>
```

Joonis 9. Testitava rakenduse *index.html*

```
angular.module('app', []).controller('mainCtrl', function
CalculatorController($scope) {
    $scope.variable = 'variable';
    $scope.getSum = function() {
        $scope.sum = $scope.x + $scope.y;
    };
});
```

Joonis 10. Testitava rakenduse *script.js*

Lisa 2 - *Jasmine*

```
<html ng-app="app" ng-controller="mainCtrl">
    <head>
        <meta charset="utf-8">
        <title>Jasmine tests</title>
        <!-- Jasmine style -->
        <link rel="stylesheet" type="text/css"
href="node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
    </head>
    <body>
        <!-- Jasmine -->
        <script type="text/javascript" src="node_modules/jasmine-
core/lib/jasmine-core/jasmine.js"></script>
        <script type="text/javascript" src="node_modules/jasmine-
core/lib/jasmine-core/jasmine-html.js"></script>
        <script type="text/javascript" src="node_modules/jasmine-
core/lib/jasmine-core/boot.js"></script>
        <!-- Angular and Angular Mock -->
        <script type="text/javascript"
src="node_modules/angular/angular.min.js"></script>
        <script type="text/javascript" src="node_modules/angular-
mocks/angular-mocks.js"></script>
        <!-- Angular controller -->
        <script type="text/javascript" src="script.js"></script>
        <!-- Tests -->
        <script type="text/javascript" src="tests.js"></script>
    </body>
</html>
```

Joonis 11. *index.html* pärast *Jasmine* failide lisamist

```
describe('app', function() {

    beforeEach(module('app'));
    var $controller;
    beforeEach(inject(function(_$controller_){$controller =
_$controller_;}));

    describe('declared value at variable', function() {
        it('should not be null', function() {
            var $scope = {};
            $controller('mainCtrl', {$scope: $scope});
            expect($scope.variable).toBeDefined();
        });
    });

    describe('sum calculator', function() {
        it('2 + 4 should equal 6', function() {
            var $scope = {};
            $controller('mainCtrl', { $scope: $scope });
            $scope.x = 2;
            $scope.y = 4;
            $scope.getSum();
            expect($scope.sum).toBe(6);
        });
    });

});
```

Joonis 12. *tests.js* pärast *Jasmine* testide kirjutamist

Lisa 3 - *Mocha*

```
<html ng-app="app" ng-controller="mainCtrl">
    <head>
        <meta charset="utf-8">
        <title>Mocha tests</title>
        <!-- Mocha style -->
        <link rel="stylesheet" type="text/css"
href="node_modules/mocha/mocha.css" />
    </head>
    <body>
        <div id="mocha"></div>
        <!-- ExpectJS & MochaJS -->
        <script type="text/javascript"
src="node_modules/mocha/mocha.js"></script>
        <script type="text/javascript"
src="node_modules/expect.js/index.js"></script>
        <script>
            mocha.setup('bdd')
        </script>
        <!-- Angular & Angular Mock -->
        <script type="text/javascript"
src="node_modules/angular/angular.min.js"></script>
        <script type="text/javascript" src="node_modules/angular-
mocks/angular-mocks.js"></script>
        <!-- Angular controller -->
        <script type="text/javascript" src="script.js"></script>
        <!-- Tests -->
        <script type="text/javascript" src="tests.js"></script>
        <script>
            mocha.run()
        </script>
    </body>
</html>
```

Joonis 13. *index.html* pärast *Mocha* failide lisamist

```
describe('app', function() {

    beforeEach(module('app'));
    var $controller;
    beforeEach(inject(function(_$controller_){$controller =
_$controller_;}));

    describe('declared value at variable', function() {
        it('should not be null', function() {
            var $scope = {};
            $controller('mainCtrl', {$scope: $scope});
            expect($scope.variable).not.to.be(null);
        });
    });

    describe('sum calculator', function() {
        it('2 + 4 should equal 6', function() {
            var $scope = {};
            $controller('mainCtrl', { $scope: $scope });
            $scope.x = 2;
            $scope.y = 4;
            $scope.getSum();
            expect($scope.sum).to.be(4);
        });
    });

});
```

Joonis 14. *tests.js* pärast *Mocha* testide kirjutamist

Lisa 4 – *QUnit*

```
<html ng-app="app" ng-controller="mainCtrl">
    <head>
        <meta charset="utf-8">
        <title>QUnit tests</title>
        <!-- QUnit style -->
        <link rel="stylesheet" type="text/css"
href="node_modules/qunitjs/qunit/qunit.css" />
    </head>
    <body>
        <div id="qunit"></div>
        <!-- JQuery & QUnit -->
        <script type="text/javascript"
src="node_modules/jquery/dist/jquery.min.js"></script>
        <script type="text/javascript"
src="node_modules/qunitjs/qunit/qunit.js"></script>
        <!-- Angular & Angular Mock -->
        <script type="text/javascript"
src="node_modules/angular/angular.min.js"></script>
        <!--<script type="text/javascript" src="node_modules/angular-
mocks/angular-mocks.js"></script>
        <!-- Angular controller -->
        <script type="text/javascript" src="script.js"></script>
        <!-- Tests -->
        <script type="text/javascript" src="tests.js"></script>
    </body>
</html>
```

Joonis 15. *index.html* pärast *QUniti* failide lisamist

```
var injector, controller, scope;

QUnit.module('app', {
    beforeEach: function() {
        injector = angular.injector(['ng', 'app']);
        console.log(injector);
        scope = injector.get('$rootScope').$new();
        controller = injector.get('$controller')('mainCtrl', {
            $scope: scope
        });
    }
});

QUnit.test('declared value at variable', function() {
    QUnit.assert.ok(scope.variable, 'should not be null');
});

QUnit.test('sum calculator', function() {
    scope.x = 2;
    scope.y = 4;
    scope.getSum();
    QUnit.assert.equal(scope.sum, 6, '2 + 4 should equal 6');
});
```

Joonis 16. *tests.js* pärast *QUniti* testide kirjutamist