TALLINNA ÜLIKOOL
DIGITEHNOLOOGIATE INSTITUUT

JAVASCRIPT SERVERIPOOLNE REST LIIDESTUS

Seminaritöö

Autor: Martin Kask

Juhendaja: Jaagup Kippar

Autor: ………………………………………………………….. „ …... „…………….... 2015

Juhendaja: ……………………………………………………… „ …... „…………….. 2015

Instituudi direktor ………………………………………………… „ …... „…………….. 2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Sisukord

SISSEJUHATUS **4**

1 KLIENDI JA SERVERI VAHELISE SUHTLUS **6**

1.1 Serveri päringute ülesehitus ja lühiajalugu 6

HTTP päringu näide *7*

1.2 REST 9

1.3 REST-i kasutavad levinumad rakendused 10

2 SEMINARITÖÖKS KASUTATAVATE VAHENDITE VALIK **11**

2.1 Kliendi veebirakendus 11

2.1.1 Flatlander's Gem Store *11*

2.1.2 Angular.js raamistik *14*

2.2 Serveripoolne rakendus 15

2.1.1 Node.js tutvustus *16*

2.1.2 Sails.js tutvustus *16*

3 ARENDUSPROTSESS **18**

3.1 Keskkondade ettevalmistamine 18

3.2 Projekti genereerimine 19

3.3 API genereerimine ja vajaduspõhiselt modifitseerimine 20

3.4 API testimine 24

3.4.1 Andmete vaatamine *24*

3.4.2 Andmete lisamine *25*

3.4.3 Andmete kustutamine *27*

3.4.4 Andmete muutmine *27*

3.5 Kliendi rakenduses API kasutusele võtmine 29

3.5.1 Kontrollerite kohandamine *30*

KOKKUVÕTE **31**

KASUTATUD KIRJANDUS **32**

Sissejuhatus

Käesoleva seminaritöö teemaks on JavaScript serveripoolse REST liidestus. Enamus veebilehitsemist võimaldavates seadmetes on olemas JavaScript'i interpretaator, mistõttu on see saanud üheks kõige laiemalt levinud programmeerimiskeeleks. Seega on autor võtnud eesmärgiks kasutada sama keelt nii kliendi kui ka serveripoolses rakenduses ning ehitada „Sharping up with Angular.js“ tasuta kursuse käigus loodud tootekataloogi rakendusele „Flatlander's Gem Store“, andmekiht, mis suudaks tooted ja sellega seonduvat andmebaasis hoiustada.

Teema valimise põhjuseks on autori huvi uuema põlvkonna tehnoloogia vastu, ta on tegelenud alates 2010 aastast programmeerimisega ning asus 2014 aastal leiba teenima veebiprogrammeerijana esialgu PHP meeskonnas, hiljem arendajana Java meeskonnas. Töö projektide arendamisel on pidevalt arendustöid JavaScript keeles, mis võimaldab veebilehe dünaamiliseks muuta. Selleks et veebiarendus oleks võimalikult ajaefektiivne kasutab autor Sails.js veebiraamistiku, mis töötab Node.js platvormil. Eelpool nimetatud tehnoloogia võimaldab luua REST liidestuse ajaefektiivselt, tänu API genereerimise võimalusele ning eelkonfigureeritud rakenduse põhikomponentidele ja turvapoliitikale.

Valmiv rakendus koosneb kahest osast, esimene on kliendipoolne rakendus mis avaneb veebibrauseris, rakendus on programmeeritud HTML5 ja JavaScript keeles ning Angular.js raamistikul. Kliendipoolne rakendus tagab andmete visuaalse kuvamise. Teine osa on serveripoolne rakendus, mis toimib andmekihina, mille pihta kliendi rakendus päringuid teeb ning seejärel tootekataloogis kuvab.

Töö eesmärgiks on lugejale tutvustada REST arhitektuuri stiili ja anda talle vahendid rakenduse arendamiseks. Töö lugemisel omandatud informatsiooni saab kasutada näiteks idufirma loomisel, ning see võib kiirendada arendusprotsessi tohutult. Kuna tegemist on täis komplekt raamistikuga, siis vajaminev konfigureerimine on minimaalne.

Teiseks eesmärgiks on kasutada kogu rakenduse komplektis sama keelt, funktsionaalsuse programmeerimisel, seda püüab autor saavutada Node.js mootoriga Sails.js raamistikul.

Võõrkeelsete lühendite loetelu

API – *Application Programming Interface*, rakendusliides ehk programmiliides

HTML – *HyperText Markup Language,* hüpertekst-märgistukeel ehk kodeerimissüsteem veebidokumentide loomiseks

JavaScript - Netscape'i poolt välja töötatud skriptikeel, mis võimaldab veebiautoritel luua interaktiivseid veebisaite

URL – *Uniform Resource Locator*, unikaalne internetiaadress, mis viitab serveris asuvale dokumendil/ressursile

URI – *Uniform Resource Identifier*, ühtne ressursi-indikaator, mis näitab ära kust ja kuidas kindlat veebiressursi kätte saada

Cachemine – Enimkasutatud ressursside hoiustamine sellisel viisil, et server ei peaks vastust uuesti kompileerima, või kui kompileerib siis ainult osaliselt ehk muutuvad parameetrid

TCP – *Transmission Control Protocol*, levinuim võrgu transpordikihi protokoll

JSON – *Javascript Object Notation*, kerge kaaluga andmevahetus formaat

XML – *Extensible Markup Language*, laiendatav märgistuskeel mida kasutatakse andmete struktureerimiseks

REST – Vaata peatükk 1.2

CLI – *Command Line Interface*, käsurealt(command promt/linux terminal) programmi kasutamisliides

SERVICE – API teenuse kiht, kus tehakse päringud serverisse, selleks et sealt andmeid saada. Serveripoolses arhitektuuris võib tähenda mingit teenust mis võtab andmed andmebaasist ning serveerib neid läbi REST teenus kliendi rakendusele

MVC – *Model View Controller*, programmi disaini muster, milles eraldatakse andmemudel, vaade ja selle kontroller

npm – *Node Package Manager*, Node.js pakihaldus tarkvara

I/O – andmete transpordikiht

Enamus sõna seletustest pärineb www.vallaste.ee e-Teadmikust.

1 Kliendi ja serveri vaheline suhtlus

Veeb oli algselt mõeldud staatilise linkide süsteemina, milles olid hüpertekst (HTML) lehed. Neid tuli alati käsitsi muuta, ning iga väiksemgi muudatus võttis tohutult aega. Selline mudel võimaldab juurdepääsu staatilisele lehele, kuid puudub dünaamilisus, mis laseks kiiresti areneval infoajastul ajakohast infot edastada. Tänapäeval kus e-kaubandusel on väga suur osakaal turust, pole sellise mudeliga midagi peale hakata. Sellisel viisil lehe ajakohasena hoidmine nõuaks suurt inimressurssi ning tellimusi peaks keegi käsitsi vormistama. Kuna kogu see tsükkel võtab tunduvalt rohkem aega, kui serveri selliste andmete muutmisele kulutaks, oli vajadus parema lahenduse järgi.

Dünaamilise mudeli juures hoiustatakse serveris koodi mis suudab genereerida vastavalt etteantud kriteeriumitele staatilise lehe. Seda realiseeritakse serveris töötava programmiga mis peale kliendiprogrammi päringut paneb kasutaja jaoks kokku staatilise HTML lehe ning edastab selle veebilehitsejasse. Kuna väljund on HTML leht, siis tegelikult veebilehitseja ei tea sellest midagi mis serveripoolel toimub. Dünaamilise mudeliga tekkis võimalus serveri poole saadetud päringuga andmeid transportida, ning seda talletada. Järgnevas ülevaates tutvustatakse ajaloolisi meetmeid sellise probleemi lahendamisest ning põhjendatakse, miks autor kasutab just REST lahendust. (Sibola, 2015)

1.1 Serveri päringute ülesehitus ja lühiajalugu

Andmete transportimiseks on ajalooliselt katsetanud erinevaid meetodeid, üheks kõige lihtsamaks variandiks, mida ka ülikoolis kohe alguses tutvustatakse, on URL-i kaudu ehk läbi HTTP kihi andmete edastus. Sellisel juhul peab olema veebiserver vastavalt programmeeritud, et kui nüüd näiliselt kasutaja sisestab mingi kindla veebilehe viida veebibrauseri navigeerimis ribale siis suudab server sellele adekvaatselt vastata.

HTTP/1.0 spetsifikatsiooni järgi jagunesid päringu tüübid kolmeks GET, POST ja HEAD (Burners-Lee, Fielding, & Nielsen, 1996).

Alates HTTP/1.1 versioonist, lisandusid veel 5 uut meetodit, OPTIONS, PUT, DELETE, TRACE ja CONNECT.

URI kaudu tehtavad päringud on GET ja POST päringud, keerulistema nagu HEAD, PUT, OPTIONS, DELETE, TRACE ja CONNECT vajavad juba spetsiaalset tarkvara või programmaatilist lähenemist. Seda seetõttu, et need sisaldavad lisaks päringu päisele ka veel päringu keha elementi, mida URL ribalt mõjutada ei saa.

Kõikide päringute tüüpide vastuste tagastamine on programmeeritav serveripoolses tarkvaras. Kasutatav päringu meetod valitakse vastavalt vajadusele, igal ühel neist on oma eesmärk. Järgnevas lõigus tutvustatakse lühidalt kõikide päringu tüüpide eesmärke ja ülesehitust.

GET – Päring küsib kindla ressursi olemasolu. Päring peaks ainult tagastama ressursi andmed selle olemasolul ning ei tohiks omada kõrval mõjusid. Lihtsaim meetod sellise päringu edastamiseks on veebiserveri poole päringu saatmine veebilehitseja URL riba kaudu.

HEAD – Sarnane GET päringule aga tagastab ainult päise osa, ning päringu keha jäetakse serveripoolsest vastusest välja.

POST – Päring kus andmeid transporditakse kliendi rakendusest serverisse. Näiteks veebilehel tellimuse vormi täitmisel saadetakse kliendi andmed serverisse selleks, et saaks kliendile arve koostada.

PUT – Talletab olemi kaasa antud URI alla, kui seal on juba midagi olemas, siis seda muudetakse, vastasel juhul lisatakse uus ressurss selle URI-ga.

DELETE – Kustutab kaasa antud ressurssi.

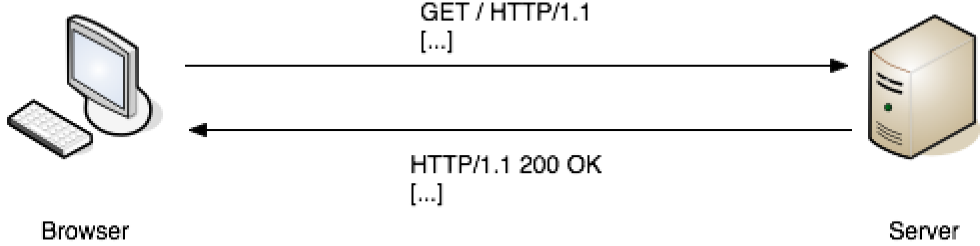
TRACE – Kuvab saadud päringut sellisel viisil, et klient näeb kui tema päringule on tehtud muudatusi seda töötlevate serverite poolt.

OPTIONS – Tagastab serveri pakutavad võimalused.

CONNECT – Loob HTTP tunneli, kasutatakse enamasti turvatud ühenduste jaoks.

PATCH – Muudab osaliselt ressurssi.

HTTP päringu näide



Joonis 1.1 Päring serverisse ja vastuse saamine

Kliendi rakendusest saadetakse serveri poole järgnev päring

```
GET HTTP/1.1

Host: gemstore.dev
```

Koodinäide 1 Kliendi HTTP päring

See on standartnäide, mida üks GET päring peaks sisaldama. Host on eraldi märgitud selleks, et toetada erinevaid virtuaalmajutusi ühe sama IP aadressi peal. Alates HTTP/1.1-st on Host atribuut kohustuslik.

```
HTTP/1.1 200 OK
Accept-Ranges → bytes
Connection → Keep-Alive
Content-Encoding → gzip
Content-Length → 138
Content-Type → text/html
Date → Wed, 21 Oct 2015 13:26:30 GMT
ETag → "4fe-5229cbeef001b-gzip"
Keep-Alive → timeout=5, max=100
Last-Modified → Wed, 21 Oct 2015 12:44:57 GMT
Server → Apache/2.4.7 (Ubuntu)

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

Koodinäide 2 Serveri vastus, kliendi päringule

Serveri vastus koosneb päringu n.ö tervisest, kui kõik läks plaanipäraselt tagastab server HTTP vastuse koodiga 200 ja ka vastuse sisu osa.

Tüüpiliselt sisu osa koosneb peast ja kehast. Pea osa näidis serveri päringust (Vt. Koodinäide 1.2) kuhu näidatakse milliseid vastuse osatüüpe see server toetab, konkreetsel juhul bittides sisu. Kas ühendus jäetakse avatuks või sulgetakse kohe peale päringut, seda saab mõjutada nii kliendi kui serveripoolt. Suletakse kui üks eelnevatest soovib selle sulgemist. Milliseid sisu kodeerimise tüüpe server toetab, kui pikk on saadetava sisu pikkus bittides, milline on sisu tüüp, näiteks gzip võimaldab pakituna andmeid transportida ning millisel ajaühikul see vastus on saadetud. „ETag“ on selleks, et saada aru kas serveris puhverdatud versioon ressursist on identne praeguse versiooniga. HTTP/1.1 versiooniga lisandus ka Keep-Alive parameeter, mis võimaldas nüüdsest jätta TCP ühenduse lahti. Selle avamine võtab omajagu aega, ning avatud ühenduse uuesti kasutamisega on serverist vastuse saamine kiirem. Rida „Last-Modified“ indikeerib küsitud objekti viimast muutmise kuupäeva, sellele järgneb rida serveri versiooniga.

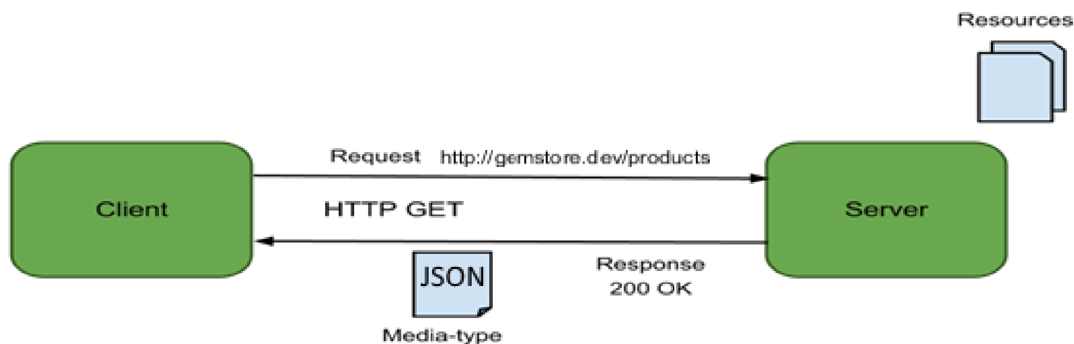
Kõige lõpuks asub päringu keha osa, mis on enamasti kas HTML tüüpi objekt või mõnedel juhtudel ka JSON või siis mõni muu andmetüüp. Näiteks REST liidestus kasutab enamasti XML või JSON andmetüüpi andmete transpordikihis. (Mozilla Developer Network, 2015)

1.2 REST

Representational State Transfer on programmiarhitektuuri stiil ja suhtlusviis kliendi rakenduse ja serveri vahel, mida tihtipeale kasutatakse veebirakenduste arenduses. See kombineerib endas ressursssipõhist rakenduse arhitektuuri, kus URI järgi tuvastatakse millise ressursiga toiminguid teha. Erinevalt üldtuntud programmeerimise stiilist, kus kutsutakse mingit suvalist funktsiooni serveri pool välja, mis teeb mingi operatsiooni, on REST-is väga tähtsal kohal päringu meetod. Sama URI suudab üldiselt teha erinevaid operatsioone, näiteks kui tehakse GET päring teatud ressurssi ja id peale, tagastab seepeale selle ressursi millele vastab päringust saadud id atribuudi. Seevastu POST päring uuendab selle id all olevat kirjet. REST-is iga tegusõna (GET, POST, PUT ja DELETE) omab kindlat tähendust, sellega välditakse kahemõttelisust.

Kui veebiteenus kasutab REST arhitektuuri, siis kutsutakse seda RESTful või REST API-ks. Kombineerides REST teenuse ja JSON andmetüübi, on võimalik luua väga kergekaaluga päringute kihte, seetõttu edastatakse ainult küsitud andmed ning sellepeale ei tehta terve lehe uuesti kompileerimist, mis toimub näiteks halvasti programmeeritud PHP lehega. Selline arhitektuur loob alustalad rakenduse laiendamiseks, näiteks kui teil on olemas REST API, mille peale on ka loodud veebirakendus, siis on sama serveri peale võimalik minimaalsete serveripoolsete muudatustega luua ka mobiili rakendus. (Sargent & Linthicum, 2015)

Ressurssi põhine disain hoiab REST API-s koodi hallatavana, kuna kui sa tead mis ressurssiga sul tegemist on, siis selle järgi on võimalik kiirelt leida ülesse potentsiaalne muudetav koht koodis. Olenevalt serveri faili arhitektuurist, tüüpiliselt asub kood ka serveris ressursi nimele vastavas kausta.



Joonis 2 REST päring serveri ja kliendi vahel

Suhtlus serveri ja kliendi vahel peab olema oma iseloomult olekuta, seega iga päring kliendi poolt serverisse peab sisaldama kogu päringust arusaamiseks vajalikku infot. Kõik sessiooniga seonduv info hoiustatakse kliendi rakenduses. Muidugi siin on erandeid, näiteks autentimise puhul võitakse kanda kliendi autentimise olekut kuskile andmebaasi, selleks et hiljem seda sealt kliendi päringu puhul verifitseerida.

Kihiline süsteem, kliendi rakendus enamasti ei tea mitut kihti ta läbib selleks et suhelda lõppserveriga. See annab kliendi ja lõppserveri vahele võimaluse sättida turvakihte, ning puhverdus servereid mis võtavad lõppserverilt koormust vähemaks.

Valikulisena kuulub veel REST arhitektuuri nõudluse järgi koodi transportimine kliendi rakendusse, mis tagab laiendatavust või rakenduse muutmise võimaluse jooksutatava koodi saatmisega rakendusse.

1.3 REST-i kasutavad levinumad rakendused

Tänapäeval on enamik suuremaid pilvepõhised teenuseid pakkuvad korporatsioonid võtnud kasutusele REST arhitektuuri, parimateks näideteks on ilmselt Amazon, Microsoft ja Google.

Amazon omab suurt arvutusparki, selleks et kasutajad saaksid sinna ise vabalt rakendusi arendada, on vajalik avalik dokumentatsioon ja API mis lubavad siis ise juurde arendusi teha. Amazon laseb selle API kaudu kasutajal kasutada oma elastse otsingu pilve serverit. Lisaks on võimalik sealtkaudu monitoorida, kui palju jõudlust päringute peale kulub, ning kuidas server nendega toime tuleb. Kuna nende serveripark on tohutult suur, võimaldab see seal jooksutada ükskõik kui suure kasutusega rakendust. Siinjuurest tasuks siiski ettevaatlik olla, kuna rakendatakse pay-per-call mudelit, ehk maksad iga API väljakutsumise pealt. Samalaadselt kasutatakse seda ka pilve talletus teenusel. (Amazon, 2015)

Microsoft on üha enam hakanud integreerima RESTful teenuseid oma rakendustesse, alates 2013 Office pakettiga kaasa tulevas OneNote rakenduses ja ka OneDrive on edukalt see implementeeritud. Api kaudu esimeses rakenduses, lugeda ja kuvada märkmeid ning teises, andmetalletus tarkvaras laadida ülesse faile, tirida neid alla, ning küsida nende kohta infot jne. (Microsoft, 2015)

Google alates 2015 aastast nimega Alphabet Inc, kasutab enamus teenustel RESTful API-t arhitektuuri. Üheks kõige kasulikumaks vast on Google Translate, mis võimaldab teksti tõlkida väga mitmetesse keeltesse, sellele lisaks veel tuvastada API kaudu saadetud teksti keel. Kahjuks on see tasuline teenus. (Alphabet Inc, 2015) Lisaks tõlkimisteenusele pakutakse ka Gmaili jaoks API-t, millega saab siis kõiki maile hallata ja ka lugeda. Muidugi sellele eelnevalt tuleb ka kasutaja autentida. Ka Google ei erine teistest suurematest andmehoiustus pilveteenuse pakkujatest, Google Drive omab samamoodi REST API-t nagu konkurendid. Peale eelnimetatud rakenduste kasutab REST arhidektuuri veel Google+, Google Wallet, Google Calendar ja ka Google Analytics.

2 Seminaritööks kasutatavate vahendite valik

Vahendite valikul oli põhiliseks ajendiks autori huvi uuema tehnoloogia vastu. Töö üheks eesmärgiks on luua Flatlanders Gem Store (edaspidi front end) rakendusele server, mis suudaks andmeid serveerida ja neid ka talletada. Seega esialgselt sattus võimalike vahendite hulka palju erinevaid programmerimis keeli kui ka nende raamistikke. Järgmisena oli vaja välja valida keel, milles serveripoolne liidestus programmeerida. Front end funktsionaalsus ise on JavaScript keeles (Angular raamistikul) programmeeritud, seetõttu tundus huvitav väljakutse, samas keeles ka serveri kood luua.

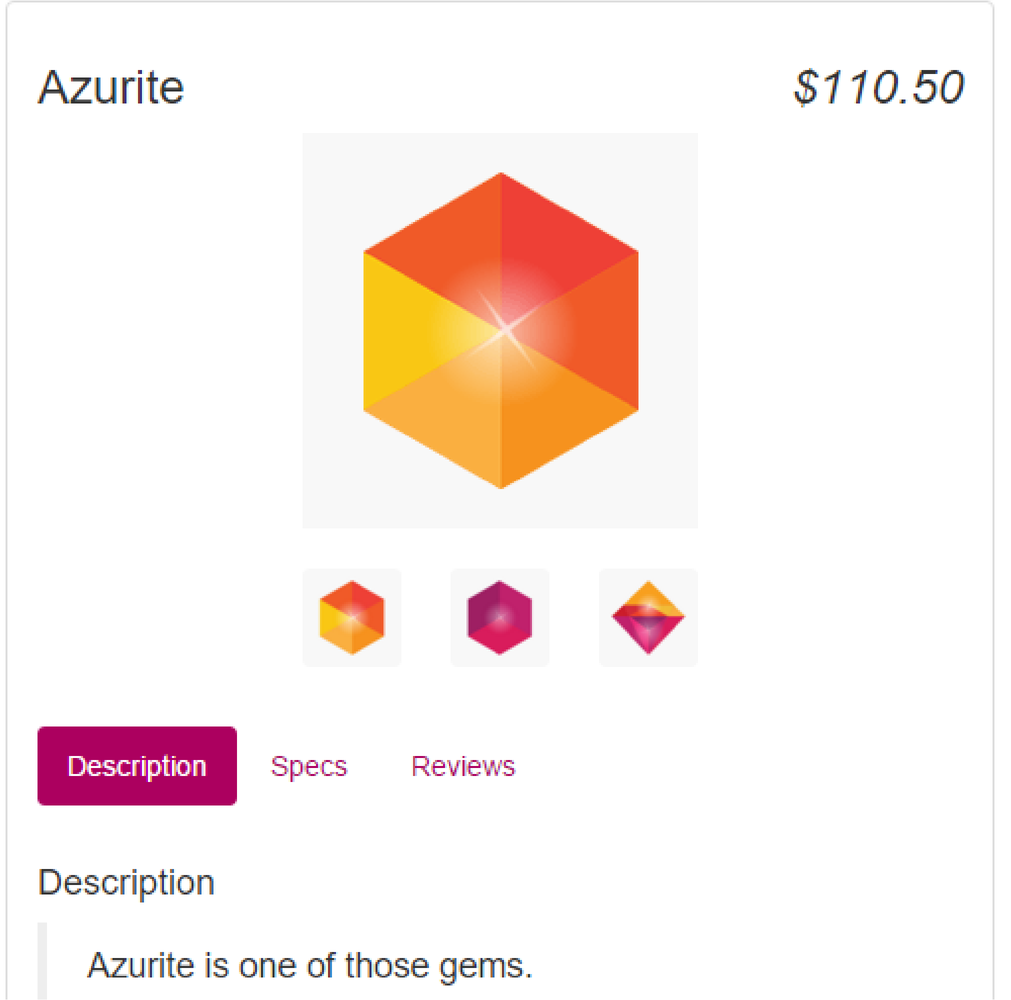
Joonis 3 Tehnoloogia kasutaja rakenduses ning serveris

2.1 Kliendi veebirakendus

Rakendus on programmeeritud Angular.js raamistikule, kuna kursus sellise raamistiku kasutamist ette nägi, siis siin mingeid valikuid autoril endal teha ei jäänud. Angular.js raamistiku programmeerimiskeeleks on JavaScript, ning kasutatakse ka HTML koodi ja koodi malle. Serveriga suhtlemiseks kasutatakse HTTP päringuid, vastavalt REST arhitektuuri nõuetele.

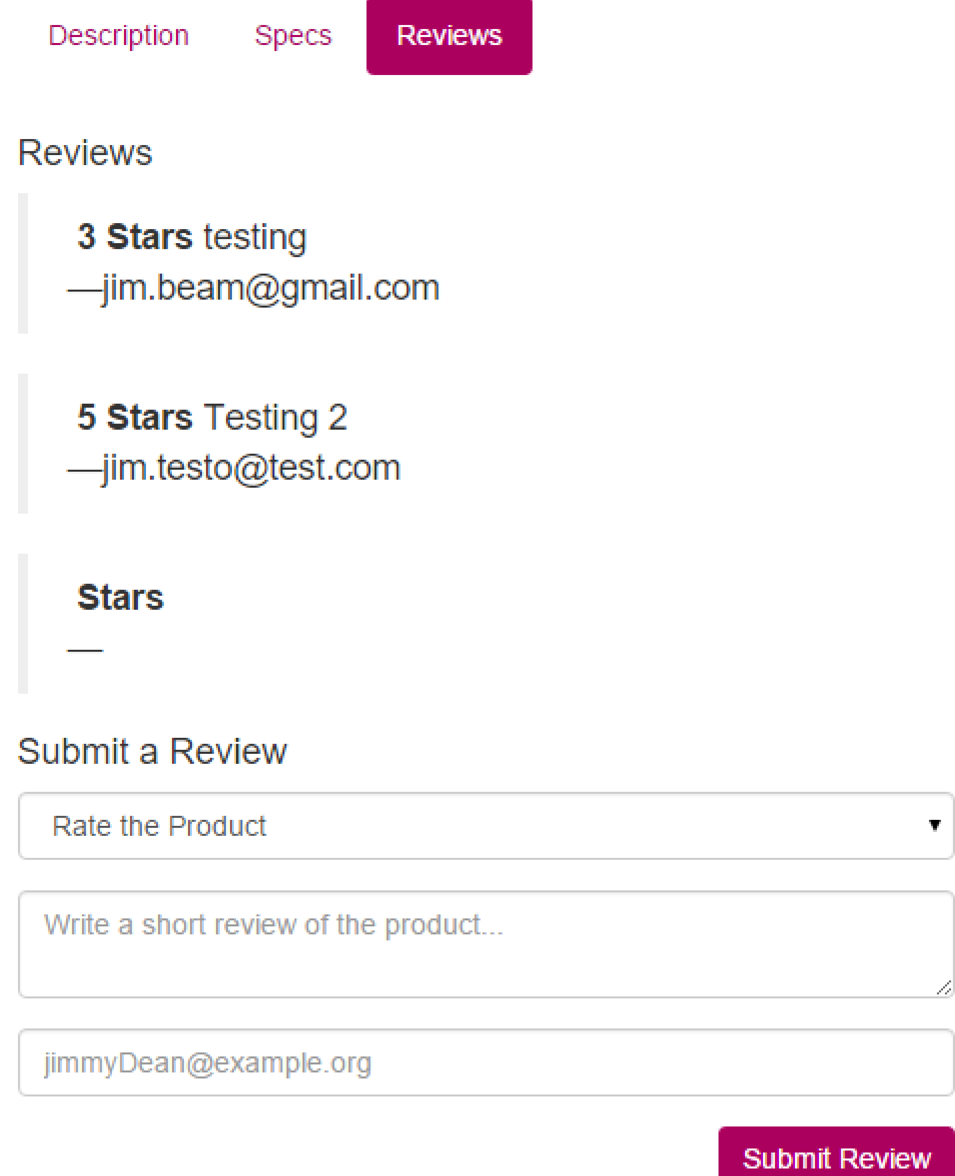
2.1.1 Flatlander's Gem Store

„Flatlander's Gem Store" on Code School (Code School, 2015) poolt hallatava kursuse käigus valmiv veebirakendus, mille programmeerimisel saab kasutaja esmase kogemuse ja algteadmised Angular.js JavaScript raamistikust. Kursus koosneb 12-st videost, 27-st programmeerimisväljakutsest ja 6-st rinnamärgist mida saab teenida, igal taseme edukal sooritamisel, tasemeid on kokku 5. Iga tase ons siis üks komponent veebilehe arenduses. Töö autori poolt loodud koodi hoiustatakse BitBucket GIT versiooni haldus tarkvara repositooriumis avalikult (Kask, 2015).



Pilt 1 Valminud kliendirakenduse kasutajaliides

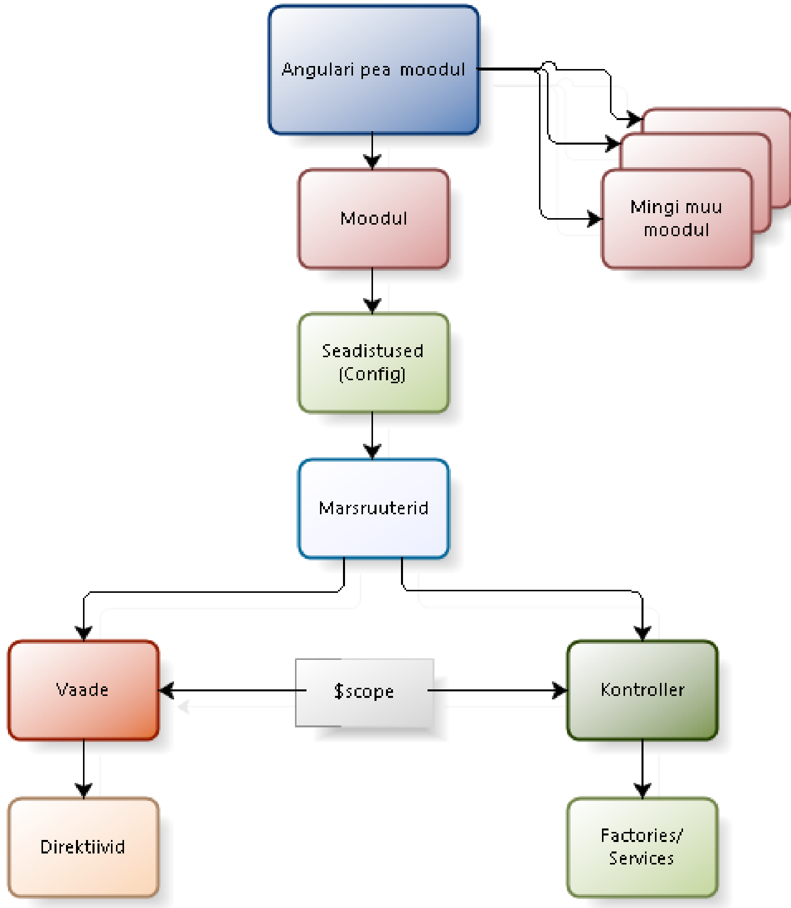
Codeschool kursuses valminud Flatlanders Gem Store rakendus peale seminaritöös kajastatud modifikatsioone. Arendusprotsessi käigus muudetakse ainult andmete transportimist haldavaid skripte, seega visuaal jääb muutumatuks.



Pilt 2 Arvustuste sakk, koos testandmetega

2.1.2 Angular.js raamistik

Eelpool nimetatud kursus õpetab samm sammult Angulari raamistikku kasutama, esmalt alustatakse uue JavaScript rakenduse loomisega. Ning selleks peab kasutaja oskama luua uue Angulari mooduli, oma kõrgema taseme struktuurilt jagunebki see mooduliteks, mis omakorda sisaldavad mooduli seadistusi, kasutaja marsruutimise funktsionaalsust ja milles sees toimuv on kontrollitud kontrollerite poolt.



Joonis 4 Angulari rakenduse arhitektuur

Siin joonisel on välja toodud Angulari rakenduse standardne arhitektuur, millele vastavalt peaks iga suuremat kasutust leidev rakendus programmeeritud olema.

Marsruuterid suunavad kasutaja URI päringuid vastavale leheküljele. Siinjuures Angular.js suudab ilusti ilma brauseris lehe uuesti laadimiseta, ainult muutuvad blokid välja vahetada. Näiteks menüüsid ei pea enamasti lehel liikudes vahetama, need jäetakse laadimata, ning sisublokk tiritakse alla ja kuvatakse kasutajale. Seda tüüpi lähenemine vähendab andmemahtu mida peab kasutaja lehel liikumiseks kulutama, ning ühtlasi jääb ka kasutajale parem mulje kui iga klõpsu peale mis ta lehe linkidel teeb, ei pane tervet lehte algusest laadima ja sellest tulenevalt pilti virvendama.

Kontroller tegeleb lehe funktsionaalse poolega, see reguleerib kasutaja tegevustest tulenevaid protsesse, olgu siis selleks kuskilt andmete pärimine või nende saatmine. Näiteks registreerimisvormi täites ja ära saates, saadab kontroller kasutaja andmed serverisse, millepeale luuakse andmabaasi uus kasutaja ning kui toiming õnnestus kuvab kontroller sellepeale positiivse teate.

Direktiive kasutatakse lehe koodi organiseerimiseks, see võimaldab lehe osad jaotada mallideks, mida saab kasutada korduvalt, ilma et dubleeriks koodi. Nagu enami programmeerijad ilmselt on kokku puutunud, kipub nii mõnigi koodijupp minema üle tuhande rea. Kui jaotada kood ilusti direktiivideks laiali, kaob selline probleem ja ühtlasi on ka lähtekood tunduvalt loetavam.

„$scope" on muutuja, mille sisse saab andmeid paigutada, selleks et neid kontrollerist vaatesse transportida, tavaliselt kasutatakse seda kui on ühe vaate kohta rohkem kui üks kontroller kasutusel. Seda muutujat kahjuks „Sharping Up with Angular.js" kursus ei käsitle, ning autor sai selle olemasolust alles praktilise vajaduse ning uurimust teostades teada.

Joonis 2.2 vastab ainult osaliselt valmivale Gem Store rakenduse arhitektuurile, kuna algajatele ei õpetata seda nii detailselt. Mistõttu jäävad näiteks „Factories/Services" blokk joonisest välja. Põhjusega, et seda ei kasutata, ning päringud tehakse otse kontrolleri sees. Üldiselt heaks tavaks on luua teenus(service), mis serveerib mingit veebi ressurssi mis tuleneb siis REST API päringust. Seepeale võetakse see Ressurss kliendi rakenduse siseselt osadeks, ning kuvatakse kasutajale välja. Ressursi kasutamine hoiab ära üleliigsete päringute tekke, ning lubab arendajal lihtsamini kõrge taseme päringuid koostada. Lisaks eelpool nimetatule pakub võimalust päringu tulemused puhvrisse jätta, mis vähendab serveri koormamist.

2.2 Serveripoolne rakendus

Serveripoolseks mootoriks sai valitud Node.js (edaspidi Node) millel on olemas hea paketihaldur, ning hulgaliselt mooduleid. Lisaks sellele on Node tuntud kui heade laiendamise võimalustega mootor, mis suudab toime tulla väga suure päringute hulgaga, vähese ressurssi juures. (Joyent, 2015)

Puhtalt mootorist ei piisa serveri jaoks, oli vajalik ka sinna peale valida raamistik, mis vähendaks rakenduse püsti seadmiseks kuluvat aega. Valitavate hulka sattus vähemalt 2 raamistikku, Express ja Sails.js. Tänu Sails.js pakutavale eelkonfigureeritud raamistikule, millel oli väga lihtne ülesse seada REST API, sai see valituks. Lisaks eelpool nimetatule võimaldas raamistik genereerida ühe konsooli rea sisestamisega terve uue objekti, mis oli koheselt valmis kasutamiseks.

2.1.1 Node.js tutvustus

Node-i tutvustati esimest korda Euroopa JSConf konverentsil 2009 aastal Ryan Dahl poolt, kes läks esitama projekti mille kallal ta oli juba mõnda aega töötanud. Projektiks oli platvorm mis kombineerib omavahel Google V8 Javascript mootori, sündmusjuhitav arhitektuur ja madalatasemelise I/O[1] API. (Teixeira, 2012).

Dahli projekti edu võtmeks on eelpool nimetatud omadused, mistõttu on paljud suurkorporatsioonid selle tehnoloogia omaks võtnud ja tehnoloogia arengusse panustama hakanud. Seda võimaldab avatud lähtekood. Kiiresti on välja arendatud enimlevinud OP[2] süsteemide tugi, mis annab võimaluse Node.js serverit majutada Windowsi, SunOS, Linuxi ja Mac OS X-i peal. (Joyent, 2015)

Alates versioon 0.6.3 on Node-ga kaasas paketihaldus tarkvara „npm“ (Nodejs, 2015) mis võimaldab Node CLI kaudu käsurealt mooduleid lisada ja eemaldada ning ka püsivate sõltuvuste kirjeldamist(ingl k. dependencies). Selleks on node projekti kaustas „package.json“ fail milles on nimistu projekti üldiste meta andmete kohta (npm, 2015). Näiteks kui teie rakendus vajab andmebaasi, siis saab andmebaasi driveri sõltuvusse ära kirjeldada eelpool nimetatud failis koos teiste atribuutidega. Atribuutideks võivad olla projekti nimetus, versioon, mõni muu projektiga seonduv konstant. Atribuute hoiustatakse seal JSON formaadis. (W3schools, 2015)

2.1.2 Sails.js tutvustus

Sails lihtsustab vajadusepõhist äriklassi Node rakenduse arendamist. See on disainitud et emuleerida sarnast MVC mustrit nagu Ruby on Rails kasutab. Seejuures toetab kaasaegsete rakenduste vajadusi, nagu andmepõhist API-t koos skaleerimisvõimalustega ja teenusele orienteeritud arhitektuuri. See on eriti hea jututubade, reaalajas halduspaneeli või online mängude jaoks, kuid sa saad seda kasutada igal veebi projektil kõikide kihtide realiseerimiseks. (McNeil, 2015)

Raamistiku peale ehitades on su rakendus 100% JavaScript keeles, see kasutab „waterline“ objekti relatsiooni kaardistamist, mis lubab kasutada enamus enimlevinud andmebaase üheaegselt. Sails pakub uut lähenemist relatsiooni mudelile eesmärgiga teha andmete modelleerimist praktilisemaks.

Automaatselt genereeritavad REST API-d, Sailsil on kaasas mallid mis aitavad rakenduse püsti seadmist kiirendada ilma koodi kirjutamata. Selleks peab jooksutama käsklust „`sails generate api toode`“ ja sa saad REST API mis võimaldab tooteid otsida, lehekülgedesse

[1] Ingl. k Input/output mudel - Sisendile peale tagastatakse väljund (nt andmed)

[2] Op – operatsioonisüsteem

jaotada(ingl k. paging), sorteerida, filtreerida, luua, eemaldada, uuendada ning luua sidemeid teiste objektidega. Kuna need on malliga kaasas käivad tegevused, töötavad need ka WebSocketitega ja kõikide toetatud andmebaasidega.

WebSocketid on toetatud vaikimisi, selleks pole vaja midagi juurde programmeerida. Minevikus reaalaja lahendused tähendasid kahe eraldi koodibaasi haldamist. Sails'il seevastu on päringute tõlkimist liides mis ühildab WebSocketi päringud tavaliste HTTP päringute vastuse formaadiga ning ei vaja selleks lisa tööd.

Deklareeritav ja taaskasutatav turvapoliitika, raamistik toetab rolli põhist ligipääsu kontrollimist. Need vahekontrollid toimivad ennem kontrollereid ja tegevusi, kui kasutaja rollil on olemas tegevuseks luba jätkub toiming. Turvapoliitika vahevara on muudetav Express/Connect vahevara vastu, mis tähendab et saab kasutada tuntud npm mooduleid nagu Passport. (Mike, 2015)

3 Arendusprotsess

Valmiva rakenduse arendus algab pihta veebikursuse „Sharping up with Angular.js“ kursuse läbimisega. Selle käigus valmib kliendi rakenduse kood, millele selle seminaritöö raames tehakse andmekiht. Kliendi rakenduse valmis kood on leitav BitBucket repositooriumist[3].

Serveripoolne rakendus valmib arenduse käigus, ning ka selle lähtekoodi hoiustatakse eraldi repositooriumis[4].

3.1 Keskkondade ettevalmistamine

Esmaks valime operatsioonisüsteemi millele rakendust paigaldama hakkame, siin õpetuses kasutan ma Ubuntu 15.04 server versiooni. NB! Eelnevatel katsetustel on esinenud probleeme Windows masinatele Sails.js raamistiku paigaldamisega, seega soovitan tungivalt kasutada õpetuse läbimisel linux masinaid.

Järgnevaks kuna Sails.js jookseb Node.js mootoril on meil vaja ette valmistada node.js. Sellekohased dokumendid leiab raamistiku haldajalt leheküljelt, juhul kui proovid muud keskkonda peale Ubuntu on ka soovitatav sinna linki piiluda.

http://sailsjs.org/get-started

Linuxi terminali kaudu näeb see protsess välja järgnev.

NB! sails.js paigaldamisel administraatori õigused on vajalikud.

```
sudo apt-get install python-software-properties python g++ make
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Nüüd peaks olema Node.js ja selle jooksutamiseks vajalikud paketid paigaldatud. Jätkame sails.js raamistiku paigaldamisega, sisestage konsooli järgnev käsklus.

```
sudo npm -g install sails
```

Kui kõik plaanipäraselt sujus, on Sails.js raamistik valmis kasutamiseks ning järgnevas peatükkis loome esimese projekti.

Kliendi rakendust saab testimiseks käivitada otse brauserist ilma, et teda peaks eraldi kuskilt serveerima, kuigi oleks mõistlik kui soovitakse rakendus ka teistele kättesaadavaks teha. Seda

[3] https://bitbucket.org/tsopic/gem-store

[4] https://bitbucket.org/tsopic/gem-store-api

sellepärast, et rakendus sisaldab ainult HTML ja JavaScript koodi, mida suudavad enamik brausereid väga edukalt jooksutada.

3.2 Projekti genereerimine

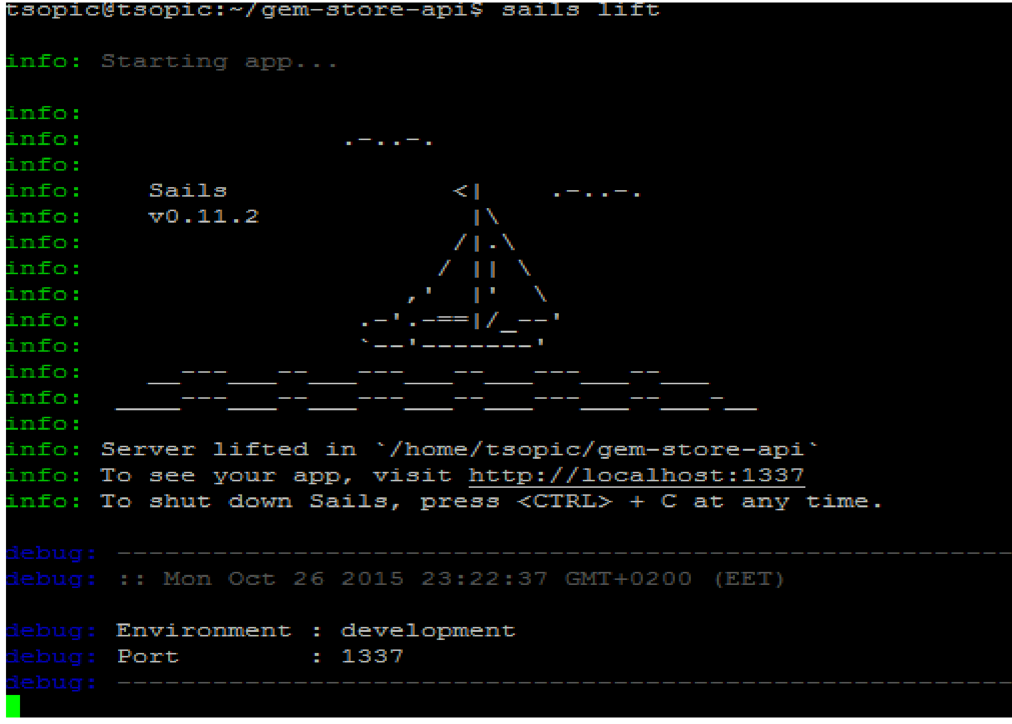
Kõigepealt navigeerige terminalis sobivasse kausta, siin õpetuses teen projekti oma kasutaja kodukausta. Peale sobivasse kausta jõudmist sisestage järgnevad käsud, et genereerida uus Sails-i projekt.

```
cd                                  #viib terminali kodukausta
sails new gem-store-api
```

Nüüd genereeriti meile uus projekt koos näidisrakendusega, selle jooksutamiseks navigeeri projekti kausta, ning sisesta käsk „sudo npm install“, mis tõmbab paketihaldurist kõik vajalikud moodulid, ning sellele järgnevalt „sails lift“ mis paneb sails rakenduse käima. Kogu see protsess peaks välja nägema järgnev.

```
cd gem-store-api
sudo npm install
sails lift
```

Peale käskude sisestamist peaks konsooli tulema merd seilava laevuksese pilt. Ning kuvatakse URL kust on võimalik värskelt loodud rakendusele ligi saada.



Pilt 3 Ekraani tõmmis konsoolist pärast sails.js rakenduse jooksutamist

Rakendusele pääseb ligi lingilt http://localhost:1337 seda muidugi juhul kui proovite rakendusele ligi pääseda samast masinast kus rakendus püsti on. VirtualBoxi-st sellele rakendusele ligipääsemiseks peab eraldi seadistama võrguadapteri, juhul kui kasutatakse NAT seadistust siis tuleb edastada pea masinale port 1337.

Lisame järgneva käsuga ka sails-disk mooduli mis lubab kasutada ajutist faili andmebaasina.

```
sudo npm install sails-disk
```

3.3 API genereerimine ja vajaduspõhiselt modifitseerimine

Nüüd hakkame siduma omavahel kliendi ja serveri rakendust. Kui teil endiselt pole veel kliendirakenduse koodi, siis tuleks see repositooriumist alla tirida. Koodi leiab peatükki alguses jaluses viidatud lingilt.

Alustame siis Gem Store toodete jaoks API genereerimisega, võtame uuesti lahti terminali ning sisestame järgneva käsu. Pange tähele, et terminalis asuksite projekti peakaustas.

```
sails generate api product
```

Peale käsu sisestamist peaks tulema informatiivne teade, et uus API on loodud. Eelnevalt sisestatud käsk genereeris uue API ressursi nimega product, millele genereeritakse nii kontroller kui mudel. Järgmine kord kui nüüd rakenduse uuesti käima paneme(sails lift), peaks olema see ressurss vastavalt REST arhidektuurile saadaval URI-lt http://localhost:1337/product.

Codeschool kursuses loodud andmefaili „products.json“ üks toode näeb oma struktuurilt välja järgnev[5].

```
   {
     "name": "Azurite",
     "description": "Some gems have hidden qualities beyond their
 luster, beyond their shine... Azurite is one of those gems.",
     "shine": 8,
     "price": 110.50,
     "rarity": 7,
     "color": "#CCC",
```

[5] https://bitbucket.org/tsopic/gem-store/src/ae1ef585ffeab4784fdbe21eacbb3b8b72206194/json/products.json?at=master&fileviewer=file-view-default

```
      "faces": 14,
      "images": [
        "images/gem-02.gif",
        "images/gem-05.gif",
        "images/gem-09.gif"
      ],
      "reviews": []
    }
```

Loome nüüd arvestades seda andmestruktuuri toote mudeli, selleks peame muutma Product.js faili mis asub kasutas „gem-store-api/api/models“. Järgnevalt tuleb avada Products.js fail omale käepärases tekstiredaktoris. Kasutan selleks konsoolirakendust nano.

```
nano Product.js              #avab Product.js faili tekstiredaktoris
```

```
module.exports = {
  schema: true,

  attributes: {
    name: {
      type: 'string',
      required: true
    },
    description: {
      type: 'string'
    },
    shine: {
      type: 'integer'
    },
    price: {
      type: 'float'
    },
    rarity: {
      type: 'integer'
    },
    color: {
      type: 'string'
    },
    faces: {
      type: 'integer'
    },
    images: {
      type: 'array'
    },
    reviews: {
      collection: 'review',
      via: 'product_review'
    }
  }
};
```

Esimene osa kus on ka „`schema: true`“, võimaldab konfigureerida mudelit. Schema rida väljendab seda, et kasutatakse andmebaasi skeemi, mis väldib skeemi väliste atribuutide lisamist. Algseadetega toetab Sails igal API-l suvalise nimega atribuutide salvestamist, seejuures kui skeemipõhine salvestamine on märgitud, talletatakse ainult skeemis defineeritud atribuudid andmebaasi[6].

Skeemi seadetele järgnevad muutujate definitsioonid, mille sees kirjeldatakse ära, mis tüüpi muutujaga on tegu, ning kas ta on nõutud muutuja või mitte. Lisaks sellele on veel palju võimalusi mis lasevad muutujat seadistada vajaduspõhiselt mida kajastab dokumentatsioon[7].

Viimane muutuja „reviews“ on toote arvustuste kogum, millele viitab ka atribuut „collection“, millele järgneb mudeli nimi. Oma olemuselt on see üks mitmele seos, mis on üldiselt tuntud kui andmebaasi objektide seos, kuid antud juhul ei pea olema andmete hoiustamismeetodiks ilmtingimata andmebaas.

Loome review API, et see siduda toodete API-ga mis võimaldaks meil Gem Stores postitada iga toote kohta eraldiseisvalt toote arvustusi.

Selleks avame taaskord terminali ja ning navigeerime projekti juurkausta ning sisestame allpool mainitud käsud.

```
sails generate api review
```

Peale positiivse tagasiside saamist, et API on genereeritud, liigu kausta gem-store-api/api/models. Muudame „review“ mudel vastavaks, et see oleks seotud toote mudeliga. Kasutame jällegi nano redaktorit.

```
cd api/models
nano Review.js
```

```
module.exports = {

  schema: true,
  attributes: {
    stars:{
      type: 'integer',
      max: 5,
      min: 1
    },
    body: {
      type: 'string'
    },
    author: {
```

[6] http://sailsjs.org/documentation/concepts/models-and-orm/model-settings

[7] http://sailsjs.org/documentation/concepts/models-and-orm/attributes

```
      type: 'email'
    },
    product_review: {
      model : 'product'
    }
  }
};
```

Siinkohal näitab product_review muutuja, millise mudeliga see arvustus on seotud. Lisaks kasutame arvustuse hinnangul min ja max väärtusi, mis määrab ära vahemiku kuhu arvustaja hinnagu tähtede arv jääma peab.

3.4 API testimine

Testimiseks kasutan Google Chrome laiendust Postman[8], mida saab vabalt Chrome rakenduste poest alla laadida. Testimise alustamiseks tuleks kõigepealt Sails rakendus käima panna. Seda tehakse käsuga „sails lift“ rakenduse juurkaustas. Kui küsitakse millist andmebaasi migreerimise viisi soovid kasutada, siis testimiseks on täiesti sobiv alter meetod, kui peaks tekkima terminali rakenduse käivitamisel veateateid, tuleks kasutada drop meetodit.

Postman REST Client rakendus võimaldab postitada modifitseeritud HTTP päringuid, kus kasutaja saab defineerida päises ja kehas kaasa antava info. Lisaks võimaldab see testida erinevaid autentimismeetodeid, ning simuleerida erinevate brauserite päringuid ja sellega kontrollida serveri reageerimist päringutele.

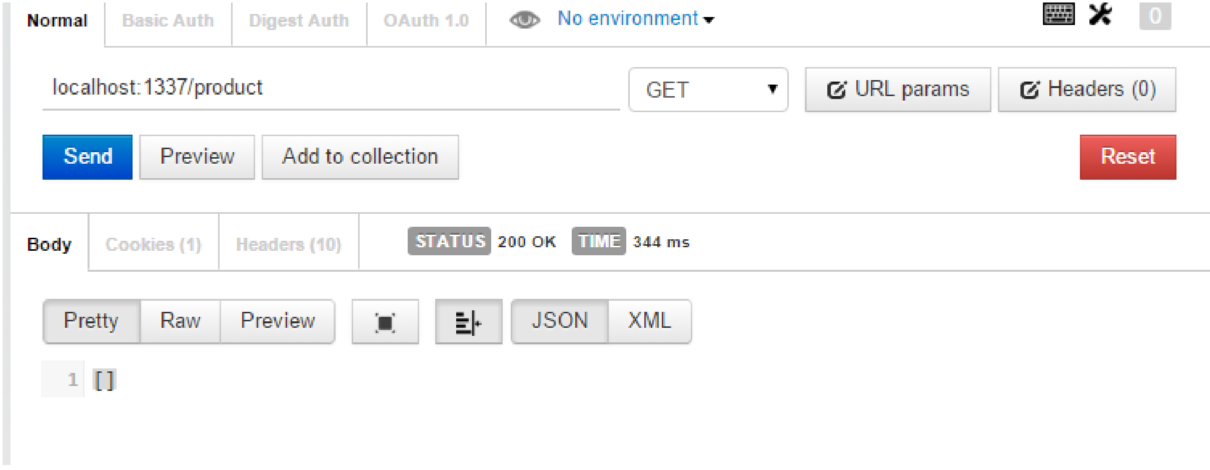
Postmani rakenduse lingi leiab lehekülje lõpust, või sisesta Chrome Appi poodi otsingusõnaks Postman, ning paigalda see „Add to Chrome“ nupule vajutades. Tegemist ei ole kõige viimase versiooniga tarkvaraga, alates Postman 2.0 ist on natuke kujundus muutunud ja muid lisasid. Autor julgustab proovima ka teisi REST kliente, et leida endale kõige meelepärasem.

Kui rakendus lisatud, avage brauseris rakenduste aken või navigeerige aadressile chrome://apps/ ning käivitage Postmani rakendus.

3.4.1 Andmete vaatamine

Vaatame olemasolevaid tooteid, selleks teeme GET päringu läbi Postmani REST API kliendi, aadressi http://localhost:1337/product pihta ning saadame päringu ära.

[8] https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?utm_source=chrome-app-launcher-info-dialog



Pilt 4 Kuvatõmmis Postman päringust

Hetkel peaks vastuseks tulema tühi JSON jada, kuna me pole veel ühtegi toote objekti andmebaasi lisanud. Juhul kui objektid on olemas, tagastatakse kõik objektid korraga, kuna pole ühtegi filtreerimis klauslit lisatud.

Selleks et pärida objekte ühekaupa, peaks lisama URI lõppu objekti id, ning näidispäringu URI näeks välja selline http://localhost:1337/product/1.

3.4.2 Andmete lisamine

Andmete lisamiseks on vaja saata API URI pihta POST päring, koos JSON kujule formuleeritud andmetega. Näidis andmed saab kliendi rakenduse lähtekoodist, „products.json" failist. Objekte tuleb lisada ühekaupa.

```
{
    "name": "Azurite",
    "description": "Some gems have hidden qualities beyond their luster, beyond their
shine... Azurite is one of those gems.",
    "shine": 8,
    "price": 110.50,
    "rarity": 7,
    "color": "#CCC",
    "faces": 14,
    "images": [
      "images/gem-02.gif",
      "images/gem-05.gif",
      "images/gem-09.gif"
    ],
    "reviews": []
  }
```

Postmanis kasutan raw andmeformaati andmete API-se postitamiseks. Konkreetse päringu peale API-sse postitamist staatuse kood peaks olema 201, mis viitab sellele et objekt on loodud.

Pilt 5 Andmete postitamine API-sse

Siit pildilt on välja jäänud vastuse JSON objekt, milles on näha, et Sails genereerib igale objektile unikaalse id konstanti ning ka loomise ja uuendamise kuupäeva muutujad. Vastus kehaosa koodina näeb välja järgnev.

```
{
    "name": "Azurite",
    "description": "Some gems have hidden qualities beyond their luster, beyond their
shine... Azurite is one of those gems.",
    "shine": 8,
    "price": 110.5,
    "rarity": 7,
    "color": "#CCC",
    "faces": 14,
    "images": [
        "images/gem-02.gif",
        "images/gem-05.gif",
        "images/gem-09.gif"
    ],
    "createdAt": "2015-10-27T15:52:52.324Z",
    "updatedAt": "2015-10-27T15:52:52.325Z",
    "id": 3
}
```

Atribuudid nimedega „createdAt“ ja „updatedAt“ indikeerivad millal on objekt loodud ja kuna viimati muudetud, ning viimane atribuut „id“ on objekti põhine unikaalne konstant, mida kasutatakse identifikaatorina.

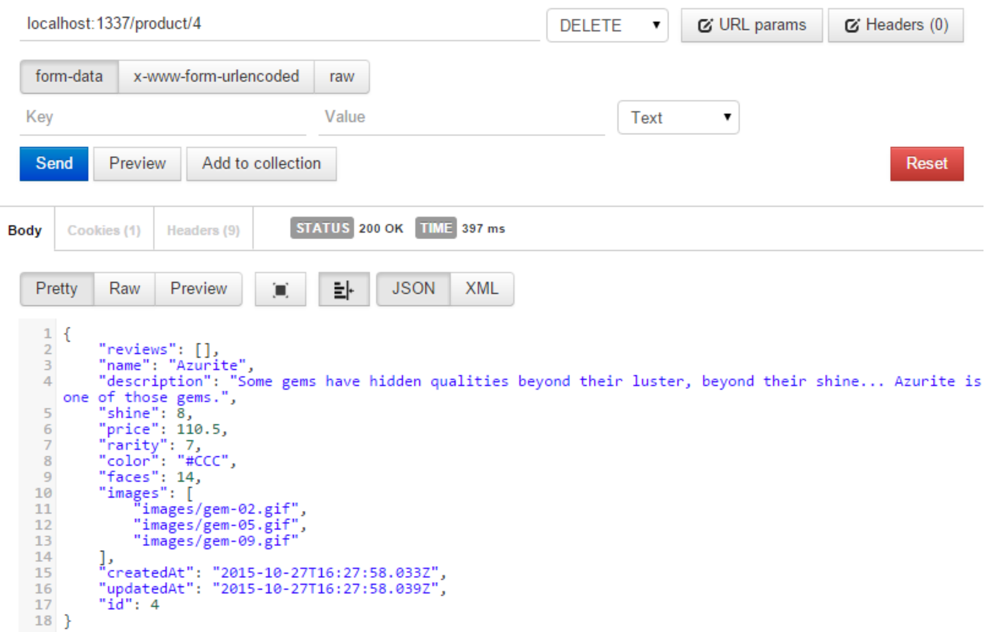
3.4.3 Andmete kustutamine

Andmete eemaldamine eeldab URI kasutamist järgnevas vormis.

```
DELETE /:model/:record
```

Kus tuleks välja vahetada :model mudeli nimega, konkreetsel juhul on see „product“ ja „record“ on andmebaasi objekti id, mida soovitakse kustutada.

Seega URI mille peale peaks tegema DELETE päringu on http://localhost:1337/product/1. Näidis päring näeks välja järgnev.



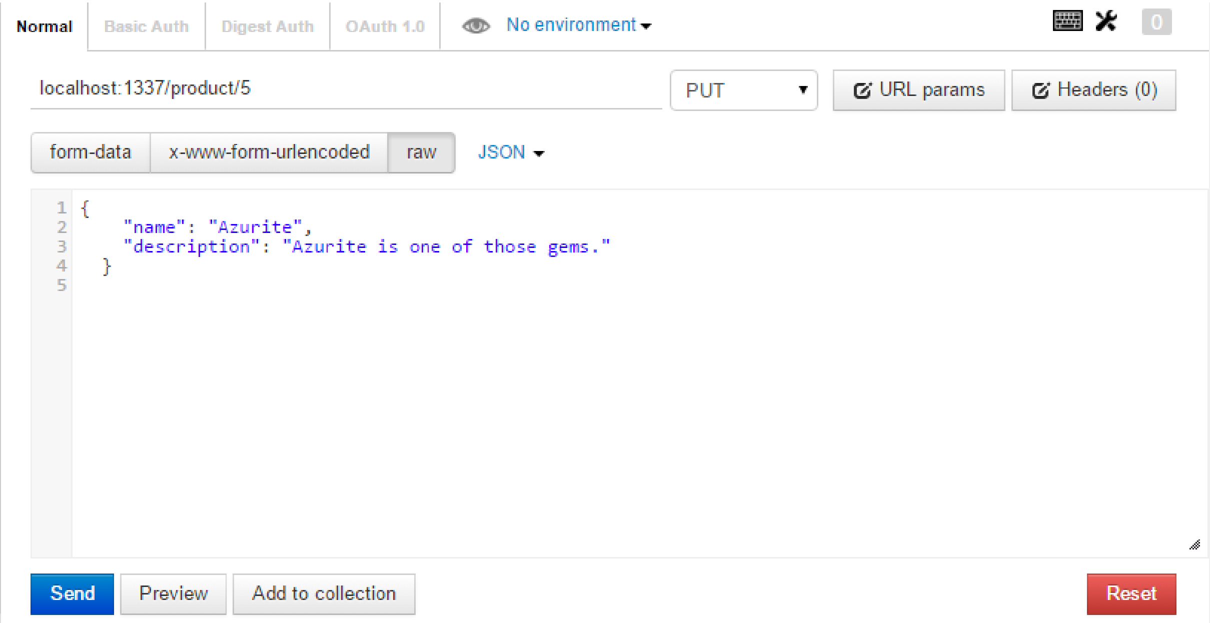
Pilt 6 Objekti eemaldamise päring

3.4.4 Andmete muutmine

Objektide muutmiseks tuleb teha PUT päring kus URI sisaldab, mudeli nime ja id-d. Atribuudid mida muuta tuleks saata HTTP kehana kodeeritud vormiks või JSON väärtustena.

```
PUT /:model/:record
```

Selle päringu näidis päring näeks välja järgnev.



Pilt 7 Andmete muutmine

Päringu saatmisele tuleb vastusena muudetud objekt JSON kujul ning staatuskood 200 OK mis tähendab, et päring läks läbi probleemideta. URI id peab vastama muudetava objekti id-le konkreetsel juhtumil muudetakse objekti id-ga 5.

3.5 Kliendi rakenduses API kasutusele võtmine

Ülesandeks on kliendirakendusse Sails api kasutamise integreerimine. Probleemi lahendamiseks on mitmeid võimalusi, otsustasin kasutada „angular-sails“ bower paketti. Bower on npm laadne paketihaldur, sellega saab väga edukalt hallata kliendi veebirakenduse teeke.

Kui bower on puudu, siis see tuleks paigaldada läbi npm paketihalduri. Seda saab teha järgneva terminali käsuga.

```
npm install –g bower
```

Nüüd kui bower on paigaldatud lisame angular-sails teegi. Eelnimetatud teek vajab töötamiseks ka sails.io ja ka socket.io-client teeki, ka need on võimalik bowerist alla laadida. Järgmisena paigaldame need paketid terminali käskudega, olles kliendirakenduse juurkaustas.

```
bower install socket.io-client
bower install angular-sails
```

Nüüd tekkis projekti juurkausta uus kaust bower_components, kus on olemas vajalikud teegid API integreerimiseks. Järgnevalt on vaja need teegid lugeda sisse index.html faili päises. Selleks muudame **index.html** päist vastavalt.

```
<!DOCTYPE html>
<html ng-app="gemStore">
<head>
    <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css" />
    <script type="application/javascript" src="bower_components/socket.io-
client/socket.io.js"></script>
    <script type="text/javascript" src="js/angular.js"></script>
    <script type="application/javascript"
src="bower_components/sails.io.js/sails.io.js"></script>
    <script type="application/javascript" src="bower_components/angular-
sails/dist/angular-sails.js"></script>
    <script type="text/javascript" src="js/app.js"></script>
    <script type="text/javascript" src="js/products.js"></script>
</head>
```

Pange tähele, et teekide sisse lugemisel oleks järjekord samasugune nagu ülevalpool olevas koodijupis, vastasel juhul suure tõenäosusega tekib viga ning rakendus ei hakka tööle. Kui te olete läbinud CodeSchool kursuse ning kasutate sealt pärinevat koodi siis failide asukohad võivad erineda kursuses tehtud koodi omast.

Modifitseerime Angulari.js pea mooduli, app.js faili ngSails moodulit kasutama selleks et saaksime päringuid läbi teegi teha. Selleks peame selle lisama app.js faili mooduli deklaratsiooni sisse.

```
var app = angular.module('gemStore', ['store-directives', 'ngSails']);
```

Nüüd konfigureerime API URL-i, et päringud käiks õige aadressi pihta. Selleks peame $sailsProvider konfiguratsioonis ära määrama urli.

```
var app = angular.module('gemStore', ['store-directives', 'ngSails']);

  app.config(['$sailsProvider', function ($sailsProvider) {
      $sailsProvider.url = 'http://localhost:1337';
  }]);
```

Päringute URL konfigureeritud, kohandame angulari päringud sails päringuteks ümber. Hetkel toimub 2 päringut. Esimene päring toimub lehe laadimisel, sellega loetakse sisse poe tooted. Teine päring toimub kui kasutaja saadab arvustuse (review).

3.5.1 Kontrollerite kohandamine

Esmalt muudame „StoreController"-it, asendame HTTP GET päringu sails päringuga. Kui päring õnnestus, asendame poe tooted saadud andmetega.

```
app.controller('StoreController', ['$sails', function($sails){
  var store = this;
  store.products = [];
  $sails.get('/product').success(function(data){
      store.products = data; // kirjutame toodete objekti päringust saadud tooted poe
                             //toodete üle
  });
}]);
```

Järgmisena muudame ReviewControllerit. Postitama kasutaja arvustusi, konkreetse toote külge. Selleks on meil vaja määrata päringus toote id, mille külge arvustus seotakse.

```
app.controller('ReviewController',['$sails', function($sails) {
  this.review = {};

  this.addReview = function(product) {
      if(!product.reviews){ // Kui pole arvustusi siis loome tühja jada
          product.reviews=[];
      }
      this.review.product_review = product.id; //pärime toote id ja lisame selle reviewle
      $sails.post('/review/' , this.review).success(function(data){
          product.reviews.push(data); // lisame lokaalselt arvustuse
      });
    this.review = {}; // tühjendame arvustuse peale saatmist
  };
}]);
```

Kokkuvõte

REST API päringud on sisuliselt spetsifikatsioonile vastavalt implementeeritud HTTP päringud. Seejuures on järgitud HTTP spetsifikatsiooni, REST võimaldab ehitada skaleerimisvõimalustega rakendust kuhu saab rakendada mitu kliendi rakendust minimaalsete muudatustega. Täiskomplektis REST arhitektuuriga rakenduse ehitamiseks on väga hea raamistik Node.js mootoril nimega Sails.js. Eelnimetatud raamistik säästab arendaja aega rakenduse valmistamisel.

Töö käigus said püstitatud eesmärgid täidetud. Nendeks oli tutvustada lugejale REST arhitektuuri ja anda vahendid sellel arendamiseks ning kogu rakenduskomplektis kasutada JavaScript keelt. Lisaks sellele valmis ka õpetus, kuidas võtta Sails.js API kasutusse Flatlanders Gem Store rakenduse andmekihina Angular.js raamistikus. Õpetus tutvustab Node paketihalduriga(npm) teekide ja tarkvara pakettide paigutamist serverisse ja viitab võimalusele kuidas teeke hallata kliendi rakenduses Bower tarkvaraga.

Arendusprotsessi käigus valmistatakse ette keskkond Sails.js raamistiku paigaldamiseks, õpetatakse Sails.js rakendusi käivitama, ning REST API-sid genereerima, neid vajadusepõhiselt konfigureerima ja andmemudeleite muutmist ja skeemipõhiseks piiramist. Lisaks sellele tutvustatakse Postman REST kliendiga API testimise meetodit.

See töö on suunatud alustavatele IT startupidele kes soovivad oma rakendust kasutada Node.js APIt ning tuleks võtta kui õppematerjali. Kui reaalselt rakendus kasutusse võtta, oleks vaja mitmeid ümber konfigureerimisi teha. See õpetus ei kajasta kuidas Flatlander's Gem Store rakendust avalikus veebis majutada.

Töö edasi arendamiseks võiks õpetada, kuidas kliendi rakendust läbi veebimajutuse serveerida, ning see panna Sails.js rakendusega suhtlema. Selleks võiks kasutada nginx serverit, teadaolevalt on sellise lahenduse jaoks vaja nginx-i ja Sails.js vahele konfigureerida I/O transpordikiht.

Kasutatud kirjandus

Alphabet Inc. (25. Oktoober 2015. a.). *Using REST - Translate API — Google Cloud Platform.* Allikas: Google Cloud Platform: https://cloud.google.com/translate/v2/using_rest

Amazon. (24. Oktoober 2015. a.). *Amazon.* Allikas: Amazon APIs - Amazon Apps & Services Developer Portal: https://developer.amazon.com/public/apis

Burners-Lee, T., Fielding, R., & Nielsen, H. F. (Mai 1996. a.). *Hypertext Transfer Protocol -- HTTP/1.0.* Allikas: RFC 1945: https://tools.ietf.org/html/rfc1945#section-8

Code School. (4. Aprill 2015. a.). *AngularJS Tutorial for Beginners.* Allikas: CodeSchool: https://www.codeschool.com/courses/shaping-up-with-angular-js

Code School LLC. (14. Aprill 2015. a.). *Sharping up with angular.js.* Allikas: Code School: https://www.codeschool.com/courses/shaping-up-with-angular-js

Joyent. (1. Aprill 2015. a.). *Node.js.* Allikas: Node.js: https://nodejs.org/download/

Kask, M. (15. Aprill 2015. a.). *Flatlander's Gem Store.* Allikas: Bitbucket: https://bitbucket.org/tsopic/gem-store

Mark, M. (27. Oktoober 2015. a.). *Get Started | Sails.js.* Allikas: Sails.js: http://sailsjs.org/get-started

McNeil, M. (4. Aprill 2015. a.). *Sails.js | Realtime MVC Framework for Node.js.* Allikas: Sails.js: http://sailsjs.org/

Microsoft. (24. Oktoober 2015. a.). *Office developer documentation.* Allikas: Office Dev Center: https://msdn.microsoft.com/en-us/library/office/dn641952.aspx

Mike, M. (26. Oktoober 2015. a.). *Features | Sails.js.* Allikas: Sails.js: http://sailsjs.org/features#?associations

Mozilla Developer Network. (21. Oktoober 2015. a.). *HTTP | MDN.* Allikas: Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/HTTP

Nodejs. (4. Aprill 2015. a.). *Node v0.6.3.* Allikas: Node: http://blog.nodejs.org/2011/11/25/node-v0-6-3/

npm. (04. Aprill 2015. a.). *npm Documents.* Allikas: node package manager: https://docs.npmjs.com/files/package.json

Sargent, M., & Linthicum, D. (22. Oktoober 2015. a.). *TechTarget.* Allikas: REST (representational state transfer) definition: http://searchsoa.techtarget.com/definition/REST

Sibola, A. (17. Oktoober 2015. a.). *Veebiinfosüsteemid Magistritöö.* Allikas: Sibol: http://www.sibola.ee/opingud/aulis-sibola-magistritoo/serveripoolsed-skriptid/

Teixeira, P. (1. Oktoober 2012. a.). *Professional Node.js: Building Javascript Based Scalable Software.* Allikas: Google Books: https://books.google.ee/books?id=ZH6bpbcrlvYC

W3schools. (04. Aprill 2015. a.). *w3schools.com.* Allikas: W3schools: http://www.w3schools.com/json/