TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Andrei Zahharov 142786IAPB

ESEMETE JA TEENUSTE VAHETAMISEKS VEEBIRAKENDUSE LOOMINE

Bakalaureusetöö

Juhendaja: Deniss Kumlander
Doktorikraad

Tallinn 2018

Autorideklaratsioon

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

Autor: Andrei Zahharov

20.05.2018

Annotatsioon

Selle bakalaureuse lõputöö käigus on loodud asjade ja teenuste vahetuste jaoks veebirakendus. Antud veebirakendus annab võimaluse kasutajale tasuta ja anonüümselt, teiste kasutajate jaoks, kasutada vahetusplatvormi. Kasutajal puuduvad piirangud kuulutuste lisamisel vahetusplatvormis. Tulemusena on uus võimalus vahetada asju ja teenuseid.

Antud lõputöö on võimalik jagada neljaks osaks. Esimeses osas on väljatoodud olemasolevad eestikeelsed müügi- ja vahetamisplatvormid. Teises osas on esitatud loodud veebirakenduse arhitektuur koos turvalisuse osaga. Kolmandas osas on toodud välja raamistike valiku põhjendus, rakendamise põhiosad. Neljandas osas on väljatoodud valminud veebirakenduse testimine ning terviklik ülevaade rakendusest ja edasiarendamise visioonist.

Veebirakenduse loomisel oli kasutatud Aurelia ja Grails raamistikke ning PostgreSQL andmebaasi. Lõputöö käigus loodud veebirakendus asub järgmisel veebiaadressil: http://kiirvahetus.tk/

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 6 peatükki, 21 joonist.

Abstract

Developing a web application for bartering and swapping.

In this graduation thesis a web application has been created for bartering and swapping goods and services. This web application provides an opportunity for the user to use a free exchange platform, which is anonymous for other users. The users of the platform do not have any restrictions on adding bartering and swapping advertisements. The result is a new way for exchanging things and services.

This thesis can be divided into four parts. In the first one, the existing Estonian sales and exchange platforms are outlined. The second section describes the architecture of the web application developed and the application security. The third part presents root cause of the framework selection and the main parts of implementation. In the fourth part, the completed web application testing, overview and future development vision are outlined.

Aurelia and Grails frameworks, as well as PostgresSQL database were used to create this bartering and swapping web application, which is located at the following web address: http://kiirvahetus.tk/

The thesis is in Estonian and contains 36 pages of text, 6 chapters, 21 figures.

Lühendite ja mõistete sõnastik

ATI	TTÜ Arvutitehnika instituut
DPI	*Dots per inch*, punkti tolli kohta
MTÜ	Mittetulundusühing
OÜ	Osaühing
HTML	*HyperText Markup Language*, hüpertekst-märgistuskeel
XML	*Extensible Markup Language*, laiendatav märgistuskeel
JSX	*JavaScript XML*
JWT	*JSON Web Token*, JSON kujuga veeb token.
JSON	*JavaScript Object Notation*, teksti andmevahetuse formaat
NPM	*Node.js Package Manager*, paketihaldur, on osana Node.js-st.
VPS	*Virtual Private Server*, on virtuaalne masin, seda müüakse Interneti-hostimise teenusena.
HTTP	*HyperTe TCPxt Transfer Protocol*, hüperteksti edastusprotokoll.
TCP	*Transmission Control Protocol*, edastusohje protokoll
UDP	*User Datagram Protocol*, kasutajadatagrammi sideprotokoll
Front-end	Veebilehele ilmuv kasutajaliides
Back-end	Tagarakendus ehk serveripoolne rakendus.
JDBC	*Java DataBase Connectivity*, Java ja andmebaasi ühendus
API	*Application Programming Interface,* rakendusliides
cookie	Veebibrauseris olev andmeblokk, kuhu võib andmeid salvestada.
GDPR	*General Data Protection Regulation*, Üldine andmekaitse eeskiri

Sisukord

Jooniste loetelu

1 Sissejuhatus

1.1 Taust ja probleem

Inimestel on olemas asju, millest nad on valmis loobuda. On kaks varianti kuidas võiks asjadest loobuda suure kaotuseta enda jaoks: üks variant on müüa neid ja teiseks võimaluseks on vahetus. Kui rääkida müügist, siis Eestis on palju müügiplatvorme, kuid platvorme, mis spetsialiseeruks ainult esemete vahetamisele pole ühtegi. Esialgne idee antud asjade ja teenuste vahetamiseks veebirakenduse abil tekkis autoril juhuslikult, kui ta otsis enda esemele midagi huvitavat vahetuseks.

1.2 Eesmärk ja oodatud tulemus

Antud bakalaureusetööl on kolm eesmärki. Põhieesmärgiks on luua veebirakendus asjade ja teenuste vahetuseks. Teiseks eesmärgiks on anda veebirakenduse struktuurist ülevaade ja kolmandaks, anda veebirakenduse edasiarendamise visiooni vaade.

Töö oodatud tulemuseks on testitud ja töötav veebirakendus, milles kasutajad võivad mugavalt vahetada oma esemeid. Veebirakendus peab andma võimaluse kasutajale registreeruda, asju lisada ning kuvada kasutajale ühekaupa võimalike vahetuste variante ja suhelda teiste esemete omanikega.

1.3 Ülesehitus

Käesoleva töö teises peatükis on väljatoodud veebirakendused ja platvormid, kus on võimalik lisada esemete müügi ja vahetuskuulutusi ning tehtud järeldus. Kolmandas peatükis on väljatoodud funktsionaalsed ja mittefunktsionaalsed nõuded, antud on ülevaade arendamisel kasutatud tehnoloogiatest ning osaliselt on väljatoodud põhilised osad rakenduse arendamise protsessist. Neljandas peatükis on esitatud arendatud tulemuse testimine: automaattestid, koormuse testid ja tungimise testid. Viiendas peatükis on kirjeldatud asjade ja teenuste vahetuste veebirakendust ning on esitatud edasiarendamise visioon.

2 Olemasolevat platvormid esemete vahetamiseks ja müümiseks

Kuna Eestis pole ühtegi sarnast vahetuste veebirakendust antud analüüsiks, on väljatoodud veebirakendused, milles on võimalik lisada vahetus- ja müügikuulutusi. Analüüsi käigus juhindutakse koostatud kriteeriumitest.

Kriteeriumid, mille järgi analüüsitakse müügi- ja vahetusplatvormi:

- Võimalus lisada vahetus kuulutusi
- Kuulutuste lisamise hind
- Anonüümsuse tagamine
- Üldine kasutuse viis

Autori soov on näidata antud analüüsiga, et sarnast veebiplatvormi pole Eestis olemas ning teha ülevaade olemasolevatest platvormidest.

Väljatoodud on analüüsiks neli platvormi, milles on võimalus asju ja teenuseid vahetada ning müüa:

- Vaheta[1]
- Okidoki[2]
- Facebookis olevad grupid ja turuplats.
- Osta.ee[3]

[1] http://vaheta.ee

[2] http://okidoki.ee

[3] https://osta-ee.postimees.ee/

2.1 Vaheta.ee

MTÜ Ettevõtlikud Noored poolt 2010. aastal loodud asjade ja teenuste vahetusportaal vaheta.ee [18]. Antud vahetusportaal on ainuke Eestis olev platvorm, milles on võimalik lisada vahetus kuulutusi. 2018. aasta 5. mai seisuga näeb vahetusplatvorm välja mittekasutatav, kuna viimased uudised ja blogi uuendused olid 2010-11 aastatel.

Vaheta.ee on täiesti tasuta vahetusportaal, kus pole reklaami. Antud portaalis on võimalik teha pakkumisi, kas eseme või raha vastu, vahetuse pakkumised on nähtavad kõikidele kasutajatele. Vahetuskuulutused on jagatud kategooriateks ja kasutaja peab ise otsima enda jaoks sobiva vahetusvariandi.

2.2 Okidoki

Okidoki on universaalse temaatikaga kuulutuste veebileht, mis oli loodud 2007. aastal 1. juunil [19]. Oli valitud analüüsiks müümisplatvormide esindajana, kuna enamus neist on sarnased.

Okidoki platvormis on kuulutuste lisamine tasuta, kuid kui kasutaja tahab, et tema kuulutus oleks avalehel, siis peab ta maksma selle eest lisatasu. Okidoki platvormil pole võimalust lisada vahetuskuulutust. Vahetuskuulutuse lisamiseks panevad külastajad hinna määramatuks ning pealkirjas on info vahetuskuulutusest. Platvorm ei taga kasutajate anonüümsust.

2.3 Osta.ee

AllePal OÜ firmaga aastal 1999 loodud ostu-müügi keskkond. Peamiseks tegevuseks on elektroonilised oksjonid. Antud analüüsis on väljatoodud seepärast, et Osta.ee platvorm on kõige suurem e-kaubamaja, mida külastatakse igal tööpäeval üle 800 000 korra [20].

Vaadates Osta.ee peale on kohe arusaadav, et antud portaal on kasutuses oksjoni tüüpi kuulutuste jaoks. Antud platvormil pole võimalik lisada vahetuskuulutusi. Oksjoni ja müügikuulutuste puhul on vajalik maksta komisjoni tasu, mis sõltub eseme kategooriast ja hinnast. Antud ostu-müügi platvormil pole kasutajate vahelist anonüümsust.

2.4 Facebook

Kõige populaarsem sotsiaalne võrgustik [21]. Oli lisatud analüüsi, kuna antud veebirakendusel on olemas enda ostu-müügi keskkond ning esemetega kauplemine on populaarne Facebooki gruppides.

Facebooki ostu-müügi keskkonda ja gruppides postitused on täiesti tasuta. Ostu-müügi keskkonda pole võimalik lisada vahetuskuulutusi, võimalik on lisada ainult müügikuulutusi. Kuid kasutades temaatilisi vahetusgruppe on võimalus vahetuskuulutusi lisada, kuid sellisel juhul kuulutuste otsimine ja esemete vahetuseks pakkumine on ebamugav kasutajate jaoks. Facebook ei taga kuulutuse lisaja anonüümsust.

2.5 Järeldus

Peale väljatoodud platvormide analüüsi võib teha järgmisi järeldusi:

- Ükski väljatoodud platvormidest ei taga kasutajate anonüümsust.
- Kahes platvormis neljast on võimalus lisada vahetuskuulutusi. Ühes veebirakenduses on võimalik lisada määramata hinnaga kuulutuse, kuid otseselt vaadates pole see vahetuskuulutus. Ja viimases veebirakenduses pole ühtegi võimalust lisada vahetuskuulutust.
- Pole ühtegi vahetustele spetsialiseeritud veebirakendust.
- Neljast väljatoodud platvormist on kolmes platvormis kuulutuste postitamine tasuta ja ühel platvormil on tasuline.

3 Asjade ja teenuste vahetuste rakenduse arendus

3.1 Asjade ja teenuste vahetuste veebirakenduse nõuded

Antud peatükis on väljatoodud asjade ja teenuste vahetuste veebirakenduse funktsionaalsed ja mittefunktsionaalsed nõuded.

3.1.1 Funktsionaalsed nõuded

Asjade ja teenuste vahetuste veebirakenduse loomine põhineb järgmiste funktsionaalsete nõuete peal:

1. Kasutajal on võimalik luua uus konto, sisestades andmeid: e-post, salasõna, eesnimi ja perekonnanimi, telefoninumber on mittekohustuslik.
2. Kasutajal on võimalik veebirakendusse siseneda kasutades e-posti ja salasõna.
3. Kasutajal on võimalik välja logida.
4. Kasutajal on võimalik lisada kuulutusi vahetuseks, sisestades pealkirja, kuulutuse hinnakategooria, kuulutuse kategooria, lisainfo ja lisades pilti.
5. Kasutajal on võimalik vahetuskuulutusi eemaldada.
6. Kasutajal on võimalik vaadata enda lisatud aktiivseid kuulutusi ning veebirakendus kuvab neid ühes vaates.
7. Veebirakendus kuvab võimalikke vahetusvariante ühekaupa.
8. Veebirakendus valib eseme või teenuse vahetuse pakkumise hinna kategooria järgi.
9. Kasutajal on võimalik valida pakutava vahetusvariandi kategooria.
10. Kasutajal on võimalik järjestada vahetuseks pakutavaid variante lisamise kuupäeva järgi.

11. Kahe vahetuskuulutuse omaniku nõusolekul loob veebirakendus vahetuskuulutuste vahel seose.

12. Kasutajal on võimalik näha kõikide aktiivsete kuulutuste seoseid ja valides ühe seose ning minna sõnumi vaate peale.

13. Veebirakendus võimaldab, aktiivsete seosete puhul saata kasutajale sõnumeid.

14. Aktiivse seose korral kuvab veebirakendus teiselt omanikult saadetud sõnumeid.

15. Veebirakendus annab kasutajale võimaluse igale kuvatavale või aktiivse seosega kuulutusele lisada kaebuse.

16. Veebirakendus peab olema võimeline eristama kahte tüüpi kasutajaid: tavakasutaja ja moderaator. Moderaatori puhul kuvada lisamenüü kaebuste ülevaatamiseks.

17. Kaebuse ülevaatamisel on moderaatoril võimalik kuulutuse kasutaja blokeerida, kuulutus blokeerida või märkida kaebus töödelduks.

18. Veebirakendus annab võimaluse kasutajale muuta enda nime, perekonnanime ja salasõna.

3.1.2 Mittefunktsionaalsed nõuded

1. Veebirakendus peab olema *responsive* kasutajaliidesega.

2. Veebirakenduse kasutamine on anonüümne teiste kasutajate jaoks.

3. Veebirakenduse kasutajaliidese keeleks on eesti keel.

4. Veebirakenduse kasutamine on tasuta.

5. Kasutaja vahetuskuulutuste arv ei ole piiratud.

6. Esialgselt peab veebirakendus suutma hoida koormust 100 kasutaja korral.

7. Kasutaja anonüümsuse tagamine teiste kasutajate eest.

3.2 Kasutajate anonüümsus

Lõputöös on realiseeritud anonüümsus teiste kasutajate eest järgmistel põhjustel:

- Kasutajal pole eelarvamusi kuulutuse lisanud inimese kohta, see tähendab, et kasutaja vaatab ainult pakutavat kuulutust mitte ei vali esemeid kuulutuse omanikku eelistades.
- Keegi ei hakka kirjutama või helistama reklaami esitamiseks või kasutajale mitte huvitavaid vahetusi pakkuma.
- Kasutaja otsustab ise kellele millist isiklikku informatsiooni jagada.
- Kasutajal pole vaja karta, et keegi tunneb ta ära.
- Kasutaja anonüümsuse tagamine teiste kasutajate eest.

Anonüümsuse tagamine on realiseeritud ainult teiste kasutajate jaoks. Administratsioonil on võimalus vaadata kõikide kasutajate andmeid ja kuulutusi.

Selleks, et tagada anonüümsust rakenduses, peavad kõik seosed olema realiseeritud ainult kuulutuste vahel. See tähendab, et veebirakenduses pole ühtegi kohta, kus oleks võimalik näha teiste kasutajate andmeid, ka sõnumite vaates on esitatud kuulutuste vaheline dialoog.

3.3 Aurelia

Aurelia on veebi-, mobiili- ja lauarvuti javascripti raamistik. Lõputöös kasutab autor Aurelia raamistiku front-endi arendamiseks.

Aurelia omadused:

- Laiendatav HTML. Aurelia laiendatav HTML-kompilaator võimaldab luua kohandatud HTML-elemente, lisada olemasolevatele elementidele kohandatud atribuute ja juhtimismallide genereerimist, toetab täielikult dünaamilist andmetöötlust ja suure jõudlusega viimistletud renderdamist [1].
- Kahesuunaline andmeedastus. Aurelia raamistik võimaldab võimsat kahesuunalist sidumist mis tahes objektiga. Kohandavate meetodite abil võib

valida oma mudelis iga vara jälgimiseks kõige tõhusama viisi ja automaatselt sünkroonida kasutajaliideselt *javascripti* klassiga [1].

- Võimas kliendipoolne marsrutiseerimine, dünaamiliste marsruutidega, sisene marsrutiseerimine ja asünkroonse ekraani aktiveerimine [1].

- Lai keele tugi. Aurelias on võimalik kasutada: ES5, ES 2015, ES 2016 ja TypeScript skriptikeeli [1].

3.3.1 Aurelia võrdlus Angular ja React raamistikega

Antud osas on toodud Aurelia, Angular 2/4 ja React võrdlus ja Aurelia valiku põhjused. Tugevate külgede võrdlus.

Aurelia on lihtsa ning arusaadava süntaksiga ja struktuuriga raamistik, mis omajagu edendab kiiret ja rahuliku raamistiku õppimist. Aurelias on kasutuses adaptiivne andmete sidumine, on mitu erinevat, kuid kõige kasutavamad on ühesuunaline ja kahesuunaline sidumine. Aurelia raamistik on modulaarne, see vähendab koormust, kuna kasutatakse ainult vajalikke mooduleid. Ning viimane Aurelia tugev külg on sisse ehitatud natiivse mobiili rakendamise toetus.

Angulari 2 kõige tugevam külg on selles, et Angular omab suuremat arendajate arvu. Selle tõttu Angular on palju tihedamini uuendatud. Teiseks, Angularil on parem TypeScripti toetus. Kolmandaks, konfiguratsiooni võimekus on palju suurem kui võrdlevates raamistikes [28].

React on kõige populaarsem võrdluses olevatest, kuid React on teek. Kuna raamistik on populaarne, siis infot Reacti kohta internetis on rohkem, kui teiste raamistike kohta. Teiseks, React kasutab virtuaalset DOM'i, see hoiab ära paljusid levinud vigu. Kolmandaks tugevaks küljeks on nagu Aurelial, natiivse mobiili rakendamise toetus.

React'ist loobumise põhjused:

- Raamistikul on kehv dokumentatsioon [27].

- React pole raamistik, React on teek.

- Liiga kiire raamistiku arendamise tempo, mille tõttu tuleb tihti projekti uuendada.
- Raamistik kasutab JSX ja see põhjustab arendusprotsessi aeglustust.
- Projekti struktuuri järgi on raamistikku keerulisem jälgida võrreldes Aurelia ja Angulariga [27].

Edasi on esitatud võrdlus Aurelia ja Angular vahel.

Konfigureeritavas osas eelistab Aurelia konfiguratsiooni konfidentsiaalsust, nii, et see kasutab vaikimisi konfiguratsioone. Angular eelistab konfiguratsiooni üle konventsiooni, mis tähendab, et peaaegu ilma iga komponendi seadistamiseta, ei saa seda kasutada. Antud juhul Aurelia arendus on palju kiirem [28].

Keerukusega võrreldes on Aurelia palju lihtsamaga semantikaga ja struktuuriga kui Angular. Kuna semantika ja struktuur on lihtsam, see võimaldab arendajatel palju kiiremini keelt õppida ja teha tööd vähem aega raisates.

Andmete sidumine Angularis on ainult kahesuunaline, kuid Aurelias on sidumise tüüpi võimalik valida käsitsi.

Aurelia on modulaarne raamistik. Aurelia koosneb väikestest raamistiku tükkidest, seega on võimalik kasutada ainult arendajale vajalikke osi [28]. Angular on monoliitne raamistik, see tähendab, et kui arendajal on vaja kasutada ühte funktsiooni, siis ta peab lisama oma projekti kogu raamistiku. Selle tõttu on Aurelial skaleeritavus parem kui Angularil.

Populaarsusega võrreldes on Angular palju populaarsem kui Aurelia. Angularil on 1600 toetajat, 100 aga Aurelial. Toetajateks on arendajad ja sponsorid.

Põhjused, miks antud võrdluses tegi autor valiku Aurelia kasutamiseks:

- Kolmest mainitud raamistikust võtab Aurelia õppimine palju vähem aega võrreldes Angulari või Reactiga.
- Võrreldes Angular raamistikuga on Aurelia kiirem enda modulaarsuse tõttu.

- Aurelia arendajad lubavad uuendada testimist enda raamistikus.

3.4 Grails

Grails on võimas raamistik veebirakenduse ehitamiseks. Grails'is on kasutuses Groovy programmeerimiskeel ja Grails on ehitatud Spring Boot'i peale [2].

Kuna Grails on ehitatud Spring boot'i peale ja kogu funktsionaalsus on üle viidud, võib antud kahte raamistiku mitte võrrelda. Võrreldes teiste raamistikega on Grails projekteeritud kiire arengu raamistikuks. Grails'is on võimalik ühe käsuga hallata kõik sõltuvused ja konfiguratsioonid, näiteks uute kontrollerite loomine või pistikprogrammi installeerimine. Grails on hästi skaleeritav, ta on abstraktsioon Spring and Hibernate vahel.

Põhilised Grails'i tugevad küljed:

- Raamistik kasutab Groovy programmeerimiskeelt. Groovy kood on palju selgem, väljendusrikkam ja puhtam kui Java kood, mis tähendab, et rakenduse loomiseks tuleb kirjutada vähem ridu, mis omakorda tõstab arendaja tootlikkust. Groovy formaadiga failides on võimalik kirjutada ka Java koodiga, kuna Groovy on mõeldud Java platvormile keele täiendusena [3].

- Sisse ehitatud REST toetus. Grails sisaldab paindlikke funktsioone, mis võimaldavad luua REST API-si [3].

- GORM (Grails' object relational mapping) pakub väga lihtsalt kasutatavat ja paindlikku kihti Hibernate'i ja mõne muu mitte-relatsioonilise andmebaasi jaoks. GORM toetab kõiki Hibernate toetatavaid relatsioonandmebaase (JDBC). See võimaldab hõlpsasti suhelda GORM'iga toetatud mis tahes andmebaasidega [3].

- Lihtne rakenduse konfigureerimine. Grails'i kasutamisel ei pea te kogu katuste plokkide koodi kirjutama. See võimaldab arendajal kulutada rohkem aega koodide kirjutamiseks, funktsionaalsete võimaluste väljatöötamiseks, mitte raamistiku konfigureerimiseks. Grails ei kasuta XML faile projekti konfigureerimiseks [3].

- Ei takista baasraamistike kasutamist. GORM kaitseb Hibernate'i kasutusest otseselt, kuid samal ajal võimaldab Grails kasutada Hibernate'i. Sama moodi ei takista Grails kasutamist ka Spring'is olevat funktsionaalsust [3].

3.5 PostgreSQL

Antud lõputöös on kasutuses PostgreSQL andmebaas, kuna antud andmebaasil on sarnane litsents nagu BSD[1] ja MIT[2]. Ainuke vahe on selles, et Postgre litsentsiga ei vastuta Postgre seaduslikult selle tarkvara probleemide eest [32]. Teiseks valiku põhjuseks on antud andmebaasi populaarsuse kasv viimaste aastate jooksul [4].

Kui võrrelda MySQL andmebaasiga, siis PostgreSQL'il on järgmised plussid:

- Piirangute kontroll [29].
- Rikas andmetüüpide valik (näiteks: massiivid, json) [29].
- Mitte-blokeeruva indeksi loomine [29].
- Ühised tabeli väljendid [29].

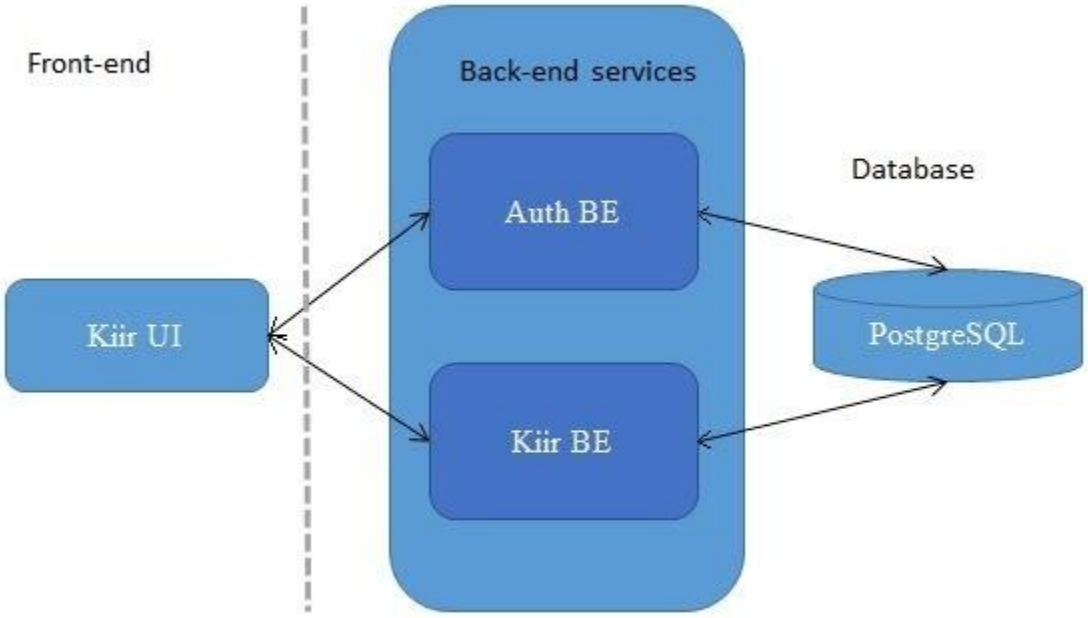
Kuna autor kasutab andmebaasi ainult läbi Grails'is oleva GORM'i, siis otseselt pole erinevust, millist andmebaasi kasutada, kuna GORM ise võib luua tabeleid ja hoiab kõik vajalikud andmed ja funktsioonid endas, näiteks järjestusnumbrid on GORM'i mälus.

3.6 Asjade ja teenuste vahetuste veebirakenduse arhitektuur

Antud töös on kaks serveripoolset teenust: kasutajaliidese rakendus ning andmebaas.

[1] Berkeley Software Distribution - lähtekoodides tarkvara levitamise süsteem

[2] Massachusetts Institute of Technology



Joonis 1 Asjade ja teenuste vahetuste veebirakenduse arhitektuuri skeem.

Kiir UI on kliendipoolne rakenduse osa. Vastutab kogu rakenduse sisu kuvamise eest, marsrutiseerimise eest, samuti väikeste arvutuste eest.

Serveripoolne osa on jagatud kaheks, esimene teenus tegeleb kasutaja autentimisega ja teine tegeleb juba sisse logitud kasutaja tehingutega: kuulutuste lisamise ja tagastamisega, kaebusete lisamise ja tagastamisega ja nii edasi. Selline jagamine vähendab koormust back-end'i peale, kuna antud jagamine annab võimaluse majutada erinevatel serveritel kõik veebirakenduse osad.

Auth BE serveripoolne teenus, mis tegeleb kasutajate registreerimise, kasutajate autentimise ning ka tokenite väljastamise ja nende uuendamisega.

Kiir BE peamine veebirakenduse serveripoolne teenus tegeleb kogu rakenduse serveripoolse tööga. Põhiteenuse ülesanneteks on uute esemete salvestamine koos pildiga, esemete info tagastamine, sõnumite ja kaebuste töötlemine.

Antud arhitektuur vastab kolmele hea arhitektuuri kriteeriumile: süsteemi efektiivsus, süsteemi paindlikkus ja süsteemi laiendatavus. Süsteem on efektiivne, kuna lahendab määratud ülesandeid ja täidab oma funktsionaalsust. Antud süsteem on paindlik, sest süsteemis on võimalik kiirelt ja mugavalt muuta olemasolevat funktsionaalsust või lisada uut. Süsteem on laiendatav seetõttu, et ilma suure mureta on võimalik lisada uusi teenuseid vajaduste järgi ja selleks pole vaja teha süsteemis suuri muudatusi [5].

3.7 Autentimine, autoriseerimine ja rakenduse kaitse

Autentimine on protsess, mille käigus kontrollitakse väidetavat identiteeti. Kõige levivam autentimise viis on kasutajanime ja parooli põhjal [6].

Autoriseerimine on protsess, mille käigus antakse kasutajale mis tahes õigused. Üldjuhul toimub autoriseerimine peale autentimist [7].

Antud lõputöö rakenduses on kasutaja autentimine realiseeritud registreerimisel määratud e-posti ja parooli põhjal.

Autentimine on realiseeritud OAuth 2.0 protokolli põhjal, peale autentimist väljastab süsteem *Access* ja *Refresh* tokenid.

OAuth – Avatud litsensiga autoriseerimis protokoll. Protokolli põhimõte on anda kolmandale ressursile piiratud juurdepääs kasutaja kaitstud ressurssidele ilma kasutajanime ja salasõna sisestamata [8].

Kogu autoriseerimine ja autentimine antud rakenduses käib läbi Auth-BE teenuse. Antud teenus tegeleb Access ja Refresh tokenite väljastamisega logimise või registreerimise põhjal.

3.7.1 Access token

Access token on autoriseerimisserverilt tulnud volitused, mis omakorda kasutatakse kaitstud ressursside juurdepääsuks. Peale autoriseerimist saadetakse access token tekstina kliendile, sagedamini liigub autoriseerimis token lihtsalt krüpteeritud kujul. Tokeni sees on konkreetsed juurdepääsualad ja juurdepääsuperiood antud ressursi omanikult [9].

Access tokeneid võib olla kahte tüüpi: esimene sisaldab andmeid, et pärida autoriseerimisinfot ja andmeid ning teine tokeni tüüp sisaldab endas kõiki autoriseerimisandmeid, teisel juhul peab tokenil olema ka allkirjastatud teksti osa [9].

Access token, asendades erinevaid autoriseerimisvõimalusi ühe tokeniga, loob abstraktsioonikihi. See abstraktsioon võimaldab väljastada tokeneid piiravamalt kui nende saamiseks kasutatav autoriseerimistoetus, selle tõttu teeb ebavajalikuks ressurss serveritel mõista mitmesuguseid autentimismeetodeid [9].

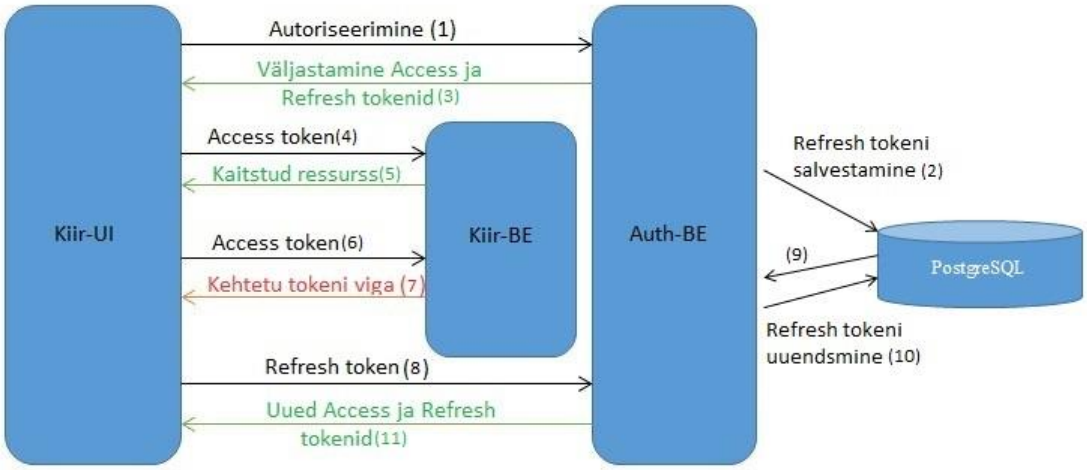
3.7.2 Refresh token

Refresh token on uuendusluba. Kasutajale antakse refresh tokenid koos access tokeniga, et tal oleks võimalik access token ohutult uuendada, juhul kui access token muutub kehtetuks või aegub. Teiseks, refresh tokeni kasutuseks on vaja saada ligipääsukleeve samale või kitsamatega loatega. Refresh tokenite väljastamine on mittekohuslik OAuth protokolli järgi, kuid kui on tehtud otsus kasutada neid, siis tokenite väljaandmine peab toimuma üheaegselt [8].

Refresh token on teksti rida, mis on väljaantud autoriseerimisserverist kliendile. Üldjuhtudel pole refresh token inimloetav. Erinevus access tokenist on selles, et refresh tokenit ei anta mitte kunagi üle ressursi serverile, refresh token liigub ainult kliendi- ja autoriseerimisserveri vahel.

3.7.3 Autoriseerimise ja autentimise näide kasutades OAuth protokolli

Järgmisel joonisel on väljatoodud kogu autentimise ja autoriseerimise protsess lõputöös arendatud rakenduses ning on näidatud, kuidas toimib tokenite uuendamine.



Joonis 2 Autoriseerimise ja autentimise näide, kasutades OAuth protokolli.

1. Kasutaja saadab autentimispäringu koos e-posti ja salasõnaga Auth-BE teenusele.

2. Auth-BE teenus teostab autentimise ning tegeleb kasutaja autoriseerimisega, kui tulnud andmed on õiged luuakse access ja refresh tokenid. Refresh token salvestatakse andmebaasi.

3. Auth-BE väljastab *Access* ja *Refresh* tokeni kasutajale.

4. Kasutaja teeb kaitstud andmepäringu Kiir-BE teenuse poole koos *Access* tokeniga.

5. Kiir-BE teenus kontrollib *Access* tokeni õigsust ja kasutaja õigusi ning tagastab vastuse päringule.

6. Punktid 4 ja 5 võivad korduda kuni *Access* token on kehtiv. Selles punktis teeb kasutaja päringu koos kehtetu *Access* tokeniga.

7. Kiir-BE teenus peale *Access* tokeni valideerimist, avastab et token on kehtetu ning saadab kasutajale tagasi 401 (mitte autoriseeritud) koodi.

8. Kasutaja teeb päringu koos *Refresh* tokeniga *Access* tokeni uuendamiseks.

9. Auth-BE saab päringu, ning võtab andmebaasist aktiivse *Refresh* tokeni ja võrdleb tulnud päringu tokeniga.

10. Juhul kui kaks tokenit on identsed, loob uue paari *Access* ja *Refresh* tokeneid ning salvestab loodud *Refresh* tokeni andmebaasi. Vana *Refresh* token kustutakse.

11. Auth-BE tagastab kasutajale uue paari kehtivaid *Access* ja *Refresh* tokeneid.

3.7.4 JSON Web Token

Antud lõputöö rakenduses peale autoriseerimist väljastatakse iga kasutajale Access token. JWT token on valitud kuna OAuth dokumentatsioonis on seda soovitatud kasutada koos autoriseerimis protokolliga [10].

JSON Web Token on avatud standard (RFC 7519[1]), mis kirjeldab turvalist ja kompaktset andmete liikumist JSON objektis [11]. Kuna JWT kolmas osa on allkirjastus, on kogu tokeni osa kontrollitav. Antud töös allkirjastamine toimub HMAC algoritmi abil, antud algoritm on soovitatud JWT dokumentatsioonis [11].

Tavaliselt JSON Web Token koosneb kolmest osast: päis, tokeni keha ja allkirjastamise osa. Kõik kolm osa tokenis on eraldatud punktiga.

Esimene tokeni osa on päis. Päis sisaldab endas üldist infot tokeni kohta, üldjuhul esimeses osas on kaks muutujat, esimene muutuja sisaldab infot, millise tokeniga on tegu (antud lõputöös on JWT) ja teises muutujas on kirjas, milline kahest allkirjastamise algoritmist on tokenis kasutusele võetud, kas HMAC SHA256 või RSA. Peale andmetega täitmist kodeeritakse esimene osa tokenist Base64Url kodeeringuga [11]. Joonisel 3 on päise näide.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Joonis 3 JWT päis

[1] https://tools.ietf.org/html/rfc7519

Teiseks tokeni osaks on tokeni keha, mis sisaldab endas kogu vajalikku infot, näiteks kasutaja privileegid või kasutaja rollid, standardi järgi peab tokeni kehas olema muutuja koos tokeni kehtivuse infoga. Nagu tokeni päis, peab keha olema kodeeritud kasutades Base64Url kodeeringu. Joonisel 4 on tokeni keha näide [11].

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Joonis 4 JWT keha näide

Kolmandaks JSON Web Tokeni osaks on allkirjastamise osa. Allkirjaosa loomiseks on vaja võtta kodeeritud päise ja keha osa ning allkirjastada kasutades päises olevaga algoritmiga.

Antud lõputöös on kasutuses HMAC SHA256 algoritm ja allkirjastamine näeb välja järgmisena:

HMACSHA256(base64UrlEncode(päis) + "." + base64UrlEncode(keha), salasõna) [12].

Allkirjastamise osa on kasutuses, et kontrollida tokeni originaalkuju säilimist. Kui teha muudatusi ükskõik millises tokeni osas, siis muutub ka allkirjastamise osa mitte kehtivaks, kuna salasõna omavad ainult serveripoolsed teenused ja salasõna sealt kuhugi ei liigu.

3.8 Loodud veebirakenduse back-endi arendus

Nagu 3.1.2 peatükis mainitud, oli loodud kaks serveripoolset rakendust, arendamiseks oli kasutatud Grails. Iga back-end teenuse loomine koosneb järgnevast: uue lingi seadistamisest, kontrolleri ja teenuse loomisest ning tagastava *view* koostamisest.

Edaspidi on esitatud loetletud osad ühekaupa.

3.8.1 URLMappings

URLMappings – on Greilsi pistikprogramm, mis aitab kaardistada kogu rakenduse infrastruktuuri linkide abil [30]. Autori rakenduses pistikprogramm tegeleb päringute jaoks avalike linkide kaardistamisega.

```
class UrlMappings {

    static mappings = {
        "/login" (controller: "auth", action: "login", method: "POST")
        "/refresh" (controller: "auth", action: "refresh", method: "GET")
        "/registration" (controller: "auth", action: "save", method: "POST")
        "/remove" (controller: "auth", action: "remove", method: "PUT")
        "/visitor/$id" (controller: "auth", action: "getVisitor", method:
"GET")
        "/visitor/save" (controller: "auth", action: "saveChanges", method:
"POST")
    }
}
```

Joonis 5 Auth-BE teenus URLide kaardistus.

Igale *mappingul*e on võimalik seadistada vastutav kontroller selle lingi päringute eest koos täpse meetodiga ning täpsustada päringu tüüpi, joonisel 5 on väljatoodud Auth-BE teenuse URLMappingu seadistamine.

Pöördudes konkreetse lingile võib Grails automaatselt luua muutuja päringu lingist. Iga selline muutuja ees on dollari märk. Näiteks kui URLMappingus on seadistatud link järgmiselt - "/visitor/$id", antud juhul tehes päringu $id asemel olev väärtus seotakse koodis olevaga id muutujaga.

3.8.2 Controller

Controller vastutab veebipäringute haldamise eest [15].

Igale endpointile on võimalik seadistada enda kontroller koos klassis oleva funktsiooniga. Järgmisel joonisel on linkide näidiskonfiguratsioon.

```
static mappings = {
        "/item" (controller: "item", action: "save", method: "POST")
    }
```

Joonis 6 Sõnumite endpointi näide

Antud näites on seadistatud päringu vastuvõtu URL ja vastuvõtu meetod ning päringut töötlev kontroller koos klassi funktsiooniga.

3.8.3 Domain

Domain – on mudeli klass, mis kirjeldab tabelit, mis asub andmebaasis. *Domain* klasse võib kirjutada Java või Groovy keeles [13].

All toodud joonisel on esitatud *domain* klassi näide.

```
class Message {
    Item fromItem
    Connection connection
    String text
    Date date
    String status

    static mappings = {
        table name: "message", schema: "public"
        fromItem column: 'from_item_id'
        connectionId column: 'connection_id'
    }
    static constraints = {
        text nullable: false, maxSize: 2000
        date nullable: false
        status nullable: false, maxSize: 1
    }
}
```

Joonis 7 Sõnumite tabeli domain class

Joonisel 7 on esitatud *domain* klassi näide. Joonisel 7 on näha kõik tabelis olevad veerud ning nende veergude piirangud. Klassis olevad muutujad seostatakse automaatselt teise domain klassi tüübiga, näite puhul(joonis 7) kui päritakse ühte sõnumit ja sellega koos süsteem teeb veel 2 päringut, et saada ese ja seos. Peale *domaini* loomist on sellega võimalik teha CRUD[1] operatsioone [14].

[1] Create/Read/Update/Delete Loomine/Lugemine/Uuendamine/Eemaldamine

3.8.4 Grails'i teenused

Grails'is teenuseid kasutatakse, et eraldada äriloogikat teistest kihtidest või komponentidest. Autoril on teenused jaotatud mugavuse jaoks kaheks: üks tüüp tegeleb ainult info pärimisega andmebaasist - sellise teenuse nimetuse lõpus on SearchService, ja teine teenuste tüüp on teenused, mis tegelevad andmete salvestamisega andmebaasi - selliste teenuste nimetused lõppevad ModifyService sõnadega. Antud jaotus on tehtud, et edaspidi oleks teistele arendajatele lihtsam projektis orienteeruda.

```
@Transactional
class ComplaintSearchService {

    def getAll() {
        return Complaint.findAllByStatus('A')
    }
}
```

Joonis 8 Kaebuste otsimise teenus.

Joonisel 8 on väljatoodud kaebuste teenused. Nimetusest on aru saada, et antud teenus tegeleb ainult andmebaasist lugemisega, teenuses on üks meetod, mis otsib kõiki aktiivseid kaebusi ja tagastab neid. Et tervel klassil oleks võimalus suhelda andmebaasiga, on klassi ette vaja lisada *Transactional* annotatsioon nagu näites on väljatoodud, antud *Transactional* lisamise viis on võetud Springi raamistikust [31].

3.8.5 Views

Antud lõputöös *view* fail on kasutatud selleks, et seadistada vastuse malli.

```
model {
    Item item
}

response.status OK

json {
    id item?.id ?: null
    heading item?.heading ?: null
    itemInfo item?.itemInfo ?: null
    creatingDate item?.creatingDate ?: null
    priceCategory item?.priceCategory?.category ?: null
}
```

Joonis 9 Eseme tagastamise mall (JSON view)

Joonisel 9 on väljatoodud JSON view, mis tagastab eseme andmed. Et kasutada antud malli kontrolleris on enne tagastamist vaja viidata antud *view* nimele.

3.9 Loodud veebirakenduse front-endi arendus

Antud lõputöös oli võetud skeleton-typescript-webpack[1] projekti mall. Võetud projektis oli juba seadistatud *typescripti* ja NPMi kasutamine. Projektis kasutatakse NPM paketi haldamiseks ja *Webpack* projekti ehitamiseks [16].

Aurelia on kirjeldatud 3.1.1 peatükis, antud peatükis on väljatoodud kõige olulisemad osad kasutajaliidese loomisel.

3.9.1 Mallid

Iga mall algab teegiga ‘template’ ja malli lõpus sama teek. Malli sisu on tava HTML, võivad olla lisatud komponendid.

Et hallata malle kasutatakse marsruutimist(*routing*) ja konfigureerimist failis. Konfigureeringu näide:

[1] https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-typescript-webpack

```
configureRouter(config: RouterConfiguration, router: Router) {
    config.title = 'Aurelia';
    config.map([
      {
        route: 'feed',
        name: 'feed',
        moduleId: PLATFORM.moduleName('./pages/feed'),
        nav: true
      }
    ]);

    this.router = router;
  }
```

Joonis 10 Aurelia marsruutimise näide

Antud näites route muutuja vastab lingi nimetusele, name – marsruudi nimetusele, mooduleid - malli asukohale, nav - kas on nähtav menüü elementides [17].

3.9.2 Komponendid

Komponendid (*Components*) on Aurelia raamistiku peamised ehitusplokid. Kasutades malle on võimalik oluliselt vähem kirjutada koodi.

Mallist mallisse on võimalik üle anda muutujaid ja terviklike meetodite koopiaid. Igas mallis on oma HTMLi ja töötlemiseks *typescripti* failide komplekt.

```
<template>
  <require from='../components/item'></require>
       ...
  <item-component item.bind="itemFromList"></item-component>
       ...
</template>
```

Joonis 11 Aurelia komponendi kasutamise näide.

Joonisel 11 on eseme komponent kasutuses tabeli mallis. Antud joonisel on näha ka, et eseme komponendile toimub eseme andmete üleandmine.

3.9.3 Päringud

Nagu dokumentatsioonis oli soovitatud, kasutatakse päringu tegemiseks aurelia-fetch-client.

```
httpClient.fetch(EnvConfig.registrationEnv, {
      method: 'POST',
      body: json(regData)
    });
```

Joonis 12 post meetodi näide

Joonisel 12 on väljatoodud *post* meetodiga näidis päring koos üle antud infoga.

Aurelia-fetch-client pakub veebirakenduses olulist funktsionaalsust: päringu parameetrite vaikimisi konfiguratsioon, püüdurid ja tsentraliseeritud päringute jälgimine.

3.9.4 Marsruutimine

Kuna Aurelia on moodulitest koosnev raamistik, siis igas moodulis võib olla ka oma marsruutimine. Et kasutada Aurelia marsruutimist, on komponendis vaja lisada <router-view></router-view> element. Kogu marsruutimine käib configureRouter() meetodis.

```
configureRouter(config: RouterConfiguration, router: Router) {
    config.map([
      {
        route: ['', 'table'],
        name: 'table',
        moduleId: PLATFORM.moduleName('./table'),
        title: 'Kõik kuulutused'
      }]
}
```

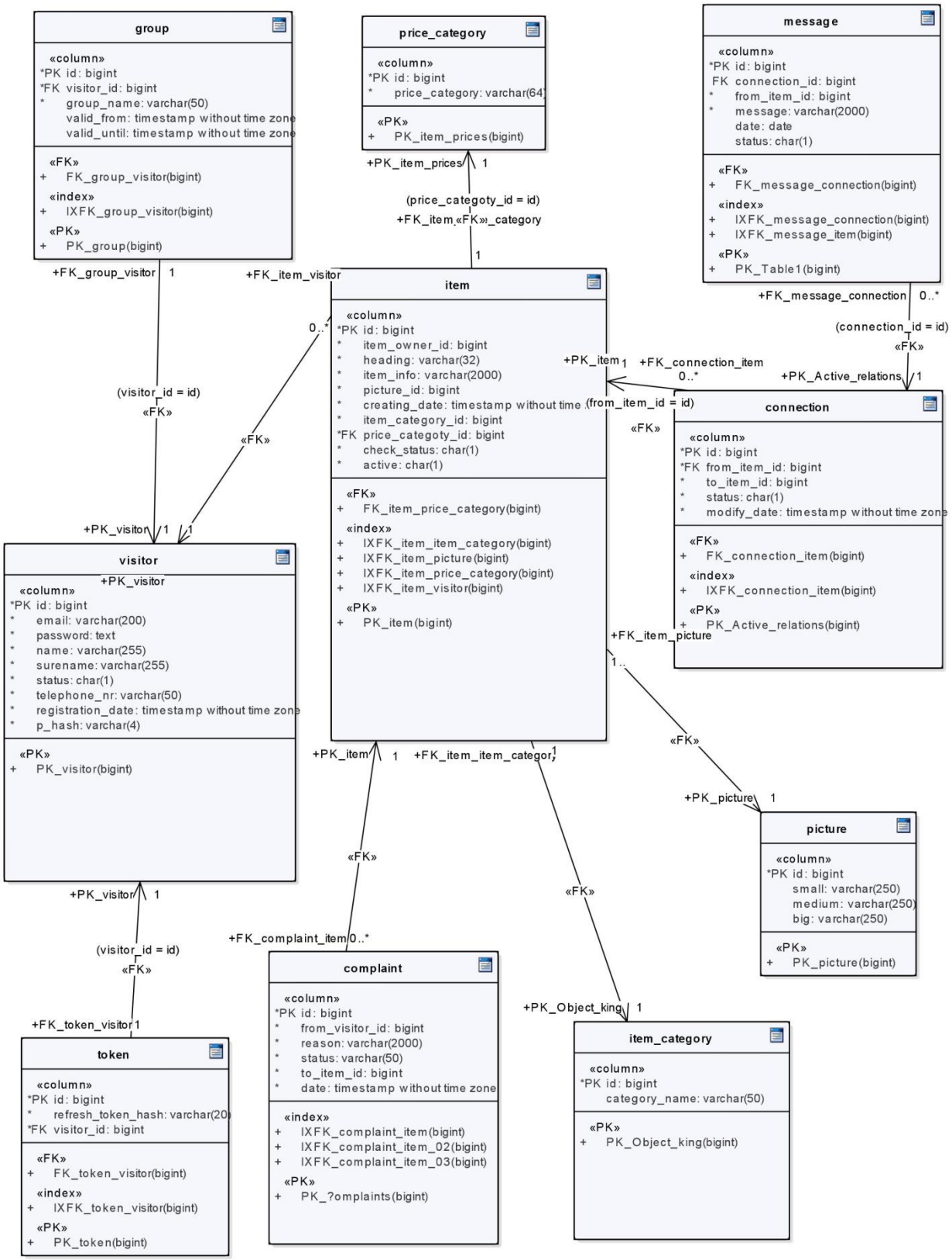
Joonis 13 Marsrutiseerimise näide.

Joonisel 13 on väljatoodud ainult ühe vaate marsrutiseerimise seadistamine, kuna teised on sarnased. Konfiguratsioonis on järgmised väljad:

- route - sisestatud URL-i fragmentidega on kooskõlas olev mustriga. Võib sisaldada ka mitu väärtust.
- name – Marsruudi tähistamiseks nimetus.
- moduleId - on mooduli id, mis ekspordib komponendi, mida tuleks marsruudi sobitamisel kasutada.
- Title – Vaate pealkiri.

3.10 Loodud veebirakenduse andmebaas

Antud lahenduse käigus oli loodus andmebaas milles on kokku 10 tabelit, andmebaasi skeem on joonisel 14. Kogu andmebaasi dokumentatsioon on Lisa 1 peatükis. Andmebaas oli loodud GORM'i abil peale *domainide* loomist ja *back-endi* teenuste käivitamist. Kõik väljade tüübid olid automaatselt valitud GORM'i poolt.



Joonis 14 Kiirte vahetuste veebirakenduse andmebaasi skeem.

3.11 Veebirakenduse majutamine

Kõik veebirakenduse osad on majutatud Google Cloudi peal. Google Cloud on pilvandmetöötluse teenus, samas infrastruktuuris on Google lõppkasutajate toodetes [22]. Valik Google Cloudi kasutada oli tehtud, kuna VPS'ide konfiguratsioon on palju parem kui konkurentidel [26].

Andmebaasi loomisel Google Cloudis polnud midagi vaja seadistada, selle pärast antud lõigus jääb andmebaas mainimata.

3.11.1 Front-end majutamine

Kliendipoolne rakendus on majutatud linuxi VPS masinal. Linux oli valitud, kuna front-endi projekti jaoks polnud vaja võimsat masinat.

Peale VPS'i tellimust oli vaja seadistada linux masin. Kuna sisemisele IP aadressile pole võimalik päringuid väljastpoolt teha, olid päringud ümbersuunatud välise IP sisemise peale kasutades nginx. Nginx on HTTP server ja tagurpidi puhverserver, e-posti puhverserver ja üldine TCP / UDP puhverserver [23].

```
map $http_upgrade $connection_upgrade {
  default upgrade;
  '' close;
}
upstream websocket {
  server localhost:5000;
}
server {
  listen 80;
  location / {
    proxy_pass http://websocket;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }
}
```

Joonis 15 nginx seadistus skript

Joonisel 15 on näha, et nginx server kuulab 80 välis porti ja sunab ümber kõik päringud sisemisele 5000 pordi peale, millel töötab node.js sisene server. Et käivitada node.js server, on vajalik installeerida node.js NPM, minna kausta, kus asub kokkupandud

front-endi versioon ja sisestada vajalik käsk terminalis. Järgmisel joonisel on samm-sammult väljatoodud node.js ja npm installerimine, 1 ja 2 rida ning serveri käivitamine - 3 rida.

```
(1) sudo apt-get install node.js
(2) sudo apt-get install npm
(3) nohup serve &
```

Joonis 16 Node.js serveri käivitamine

3.11.2 Back-end majutus

Kaks back-endi teenust on majutatud Windows Serveri 2016 VPS'i peal. Windows Server oli valitud, kuna Google Cloudil antud masin talub kahe teenuse koormust ja on graafilise ühendamise võimalus.

Et päringud jõuaksid teenuseni, on vaja lisada vajalikke reegleid Google Cloud sätetes ja Windows Serveris tulemüüri sätetes. Google Cloud-is on vaja teha lahti Firewall rules vaade ning lisada uus reegel. Reeglis on vaja täpsustada pordi number, kelle jaoks port on avatud ja millist protokolli kasutatakse päringuteks. Windows Serveris on samuti vaja lisada reegel Windows Firewall with Advanced Security menüüs, kõik nõutavad väärtused on samad nagu Google Cloud sätetes.

4 Veebirakenduse testimine

4.1 Rakenduse automaattestimine

Üheks oluliselt tugevaks küljeks Grails'is on sisseehitatud testimistööriistad. Rakenduse *back-endi* loomise käigus olid kirjutatud automaattestid, antud juhul unit testid ja integratsiooni testid. Testidega on kaetud 74.3% koodist.

4.1.1 Unit testid

Unit testid on kõige kiiremad automaattestidest, kuna nad testivad ilma füüsiliste ressursside olemasoluta. Unit testid testivad rakenduse koodi meetodite või blokkide järgi. Testimiseks kasutab autor sisseehitatud Grails Testing Support teeki. Kuna Unit testid on kiiremad kui integratsiooni testid, siis enamus koodist on kaetud nendega, et säästa aega [24].

```
@Mock([Complaint])
class ComplaintModifyServiceSpec extends Specification {

    void "Complaint modify service test"() {
        when:
            ComplaintModifyService complaintModifyService = new
ComplaintModifyService()

            Complaint response = complaintModifyService.saveComplaint([
                    visitorId: 2,
                    toItemId: 1,
                    kaebusePohus: 'test kaebus'])
        then:
            response.reason == 'test kaebus'

    }
}
```

Joonis 17 Testi näide

Grails'is on vaikimisi kõik testid sama struktuuriga nagu on näidatud joonisel 17. Iga testide grupp on mähitud meetodiga *void* ja edasi on lisatud testimise kommentaar.

Antud testi näites on kaks plokki: *when* ja *then*. *When* ploki sees initsialiseeritakse andmeid ja objekte ning *then* plokis toimub vastuse kontrollimine.

Et unit testidel oleks võimalik testida meetodeid, milles on tegu päringutega andmebaasi, on vaja kasutada *Mock* annotatsiooni klassi nimetuse ees. Selleks, et lisada ajutisi andmeid *domaini* ja simuleerida andmebaasi töös enda andmeid Iga annotatsioon algab kommertsmärgiga (@). Joonisel 17 on väljatoodud näide annotatsiooni lisamisega, peale antud annotatsiooni lisamist, on testi sees võimalik luua esemeid ja seoseid ning salvestada neid, et edaspidi testida vajalikke meetodeid.

4.1.2 Integratsiooni testid

Integratsiooni testides ühendatakse programmi koodi meetodid ja testitakse gruppides. [24] Antud testimise viis on aeglasem kui unit testid, kuna testitavam koodi kogus on suurem ning integratsiooni testid saavad ligi Grails'i keskkondadele, näiteks testil on võimalus kirjutada andmeid andmebaasi. Integratsiooni test näeb välja samasugusena nagu unit test, näide on esitatud joonisel 17, ainuke erinevus on selles, et integratsiooni testi klassi ees on @Integration annotatsioon.

Kuna integratsiooni testid kasutavad andmebaasi andmete salvestamiseks, on peale iga integratsiooni testi vaja andmeid sealt kustutada, selleks et autoriseerida seda, on vaja klassi nime ette lisada @Rollback annotatsioon. Antud annotatsioon tagastab andmed sellisesse olekusse, milles nad olid enne testimist [25].

4.2 Koormuse testid

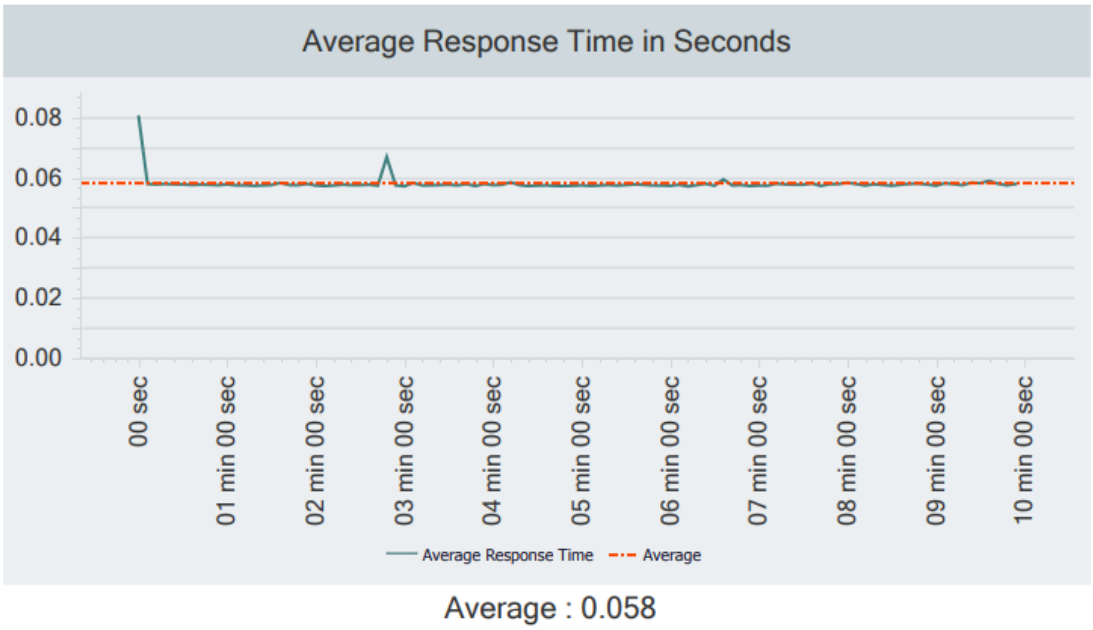
Koormuse test on planeeritud testide kogum, et testida süsteemi funktsionaalsust konkreetsete üheaegsete päringutega. Antud testi nimetatakse ka mahu katseks, kuna koormuse test tagab veebisüsteemi võimekust töödelda eeldatavat liikumismahtu. Koormuse testi peamiseks eesmargiks on tõestada, et süsteem suudab töödelda oodatud päringute mahtu, minimaalse vastuvõtu võimaluse langemisega.

Koormuse testimiseks oli kasutatud dotcom-monitor[1] veeb portaal, antud portaal annab tasuta võimaluse kasutada testide jaoks 100 kasutajat. Nimetatud kasutajate arvust

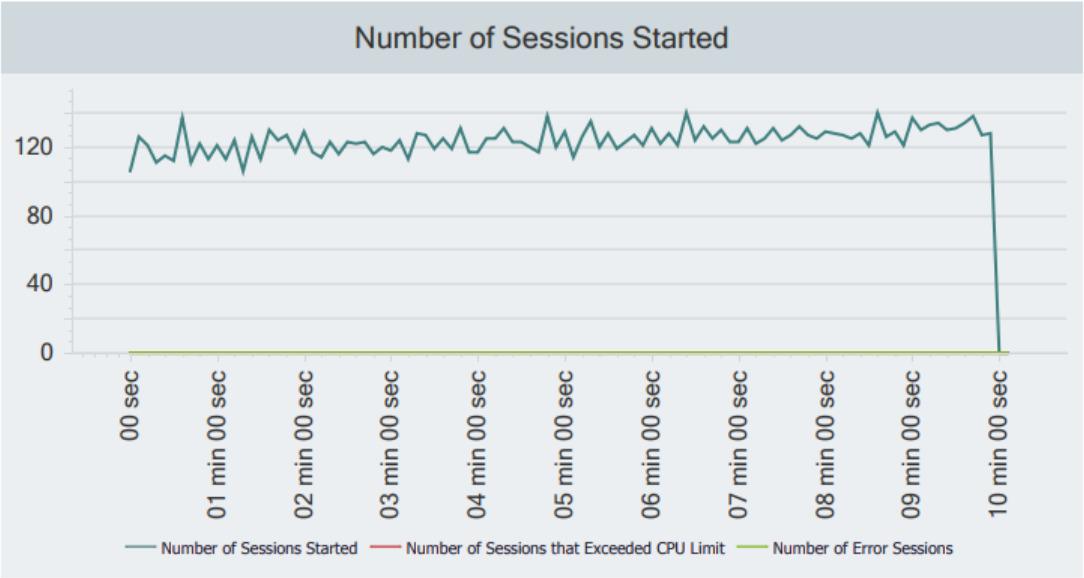
[1] https://www.dotcom-monitor.com

piisab, kuna veebirakendus on veel arendamisel ja samaaegselt vahetusplatvormil olevate kasutajate arv ei ületa 100 inimest.

Järgmisel kahel joonisel on esitatud keskmine reaktsiooniaeg ja alustatud seansside arv sekundis.



Joonis 18 Keskmise reaktsiooniaja testi tulemus.



Joonis 19 Maksimaalselt võimalik seansside alustamiste arv sekundis.

Koormuse testi alguses alustatakse veebirakenduse testimist 80 kasutajaga ja iga minut nende arvule lisatakse 2 kasutajat. Testi kestvus oli 10 minutit, mille lõpus kasutajate arv lähenes 100 juurde. Sessioonide graafikul(joonis 19) on näha, et seansside arv tõuseb iga minutiga, kuid kui vaadata keskmist reaktsiooni graafikul joonisel 18, pole selle aja jooksul reaktsiooni aeg muutunud ja on sama terve testimise jooksul. Sellest võib teha järelduse, et planeeritud kasutajate arvule suudab veebirakendus vastu pidada ja tavalisel kasutusel võib kasutajate arv olla suurem.

4.3 Tungimise testimine

Tungimise testimine on simuleeritud rünnak veebirakendusele, teostatakse turvalisuse hindamiseks ja nõrkuste tuvastamiseks.

Loodud veebirakenduse testimiseks olid võetud baastestimise nõuded cybrary[1] ja hackercombat[2] portaalidest.

Veebirakenduse tungimise testide tuvastatud tugevad küljed:

[1] https://www.cybary.it/0p3n/penetration-testing-checklist/

[2] https://hackercombat.com/web-application-penetration-testing-checklist/

1. SQL injectioni test – põhineb päringu tegemisel suvalises SQL-koodi sisse kirjutamine, antud tungimine mõjub andmebaasile. Rakendatud veebirakenduses pole võimalik tungimist tekitada, kuna kasutuses on GORM'is olev funktsionaalsus ilma SQL'i koodita.

2. Kasutaja seansi test – Testi põhimõtte, on kontrollida, et kasutaja välja logimisel sessioonid katkestatakse ja sisse logimisel luuakse uued sessioonid, kuna häkkerid võivad kasutada avatuid seansse enda kasuks. Kuna veebirakenduses kasutatakse tokeneid, milles on kehtivuse aeg, siis kui keegi suudab varastada tokenit, on kurjategijal maksimaalselt 10 minutit juurdepääsu aega või vähem, kui konto omanik logib sisse teist korda.

3. Avatud portide test – Avatud testide käigus oli kontrollitud, et mittekasutatavad pordid on suletud, kuna häkkerid võivad kasutada avatud veebirakenduse porte oma vajaduste jaoks. Kuna Google Cloudis on vaikimisi kõik pordid kinni, siis üleliigseid porte pole avatud.

4. Tundlikute andmete URLis test - Testi käigus oli kinnitatud, et URLis pole tundlikuid andmeid. Tundlikke andmete puhul teostakse *post* päring.

5. Cookie test – Antud testi põhimõtteks on kontrollida, et *cookides* olev info on mitteloetavas formaadis ja on turvaline. Loodud rakenduse *cookides* hoitakse ainult JWT tokeni kodeeritud kujul.

6. Juurdepääsu test – Antud test on suunatud andmete saamise võimalustele ilma sellekohase õigusteta kontrollimisele. Veebirakenduses on ainult kaks *endpointi*, milles pole õiguste kontrolli. Kõikides *endpont*ides kus töödeltakse tundlikke andmeid on teostatud õiguste kontroll, õigused kontrollitakse valides *token*i järgi

Veebirakenduse nõrgad küljed:

1. Failide skaneerimine – Veebirakenduses puudub serveripoolne failide skaneerimine, mis ideaalis peaks olema enne faili salvestamist.

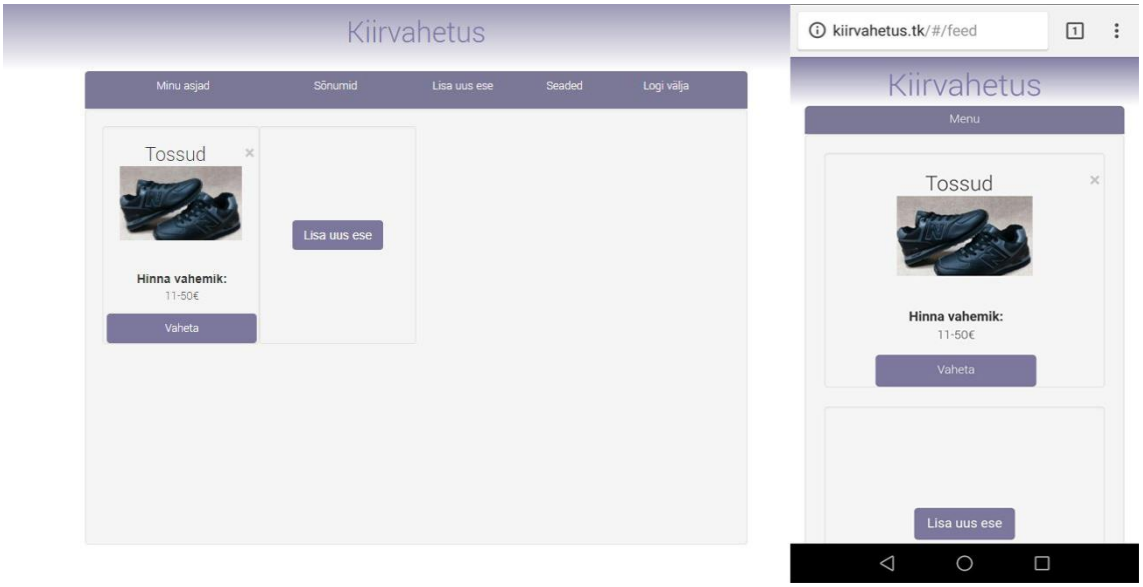
2. HTTP protokolli kasutamine – antud arendusetapis on kasutuses ebaturvaline HTTP protokoll, kaubandusliku majutusele üleviimisel on vaja veebirakenduse üle viia HTTPS protokolli peale.

3. Veebirakenduse sisemise vaate testimine – Antud testiga oli avastatud sissepääsu vormi tõsine puudus, veebirakendused pole piiratud ebaõnnestanute logimiste arv. See halvendab turvalisust, kuna kurjategijad võivad antud viga muuta veebirakenduse mitte saadavaks.

5 Tulemus ja edasiarendamise visioon

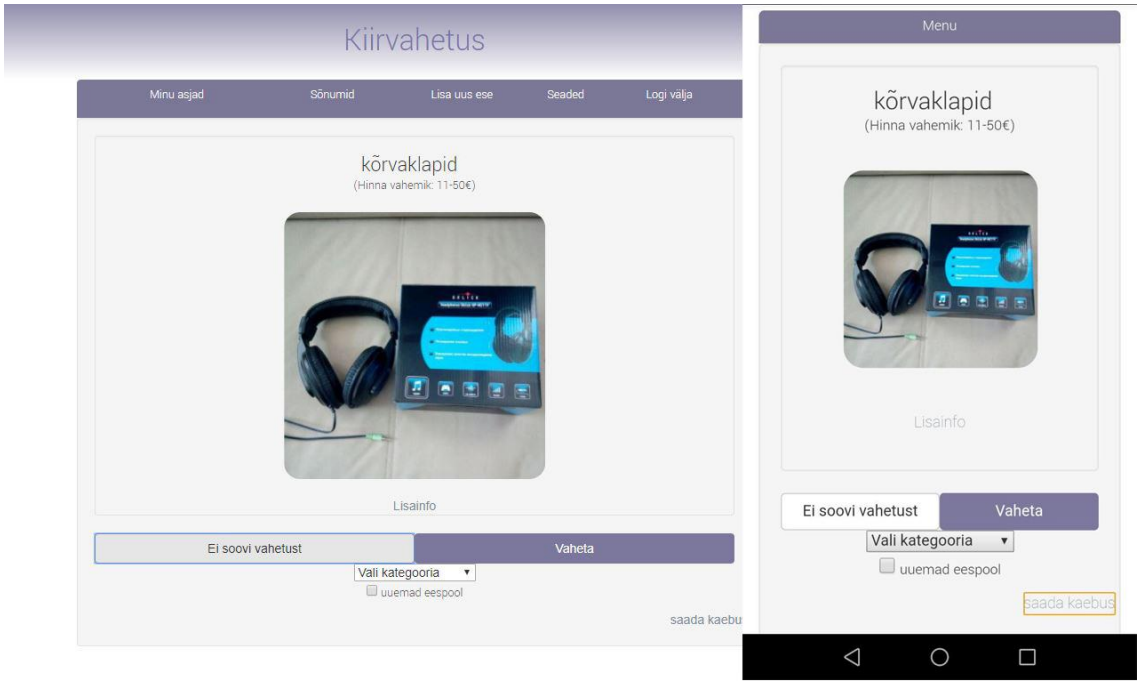
5.1 Tulemus

Tulemusena antud lõputöös on täiesti töötav ja testitav veebirakendus esemete ja teenuste vahetuste jaoks. Antud rakenduses on kasutajal võimalik järgnev: luua uus konto, lisada enda esemeid vahetamiseks, vaadata võimalikke vahetusvariante konkreetse kuulutuse jaoks, suhelda teiste kuulutuste omanikega. Rakenduses on realiseeritud täielik anonüümsus teiste kasutajate suhtes.



Joonis 20 Asjade ja teenuste vahetuse veebirakenduse kasutaja esemete nimekiri.

Kasutajal on lubatud lisada piiramatus koguses vahetukuulutusi. Joonisel 20 on kasutaja esemete vaade, realiseeritud arvuti ja mobiili kasutajaliidese versioonid. Vahetuseks peab kasutaja valima enda kuulutustest ühe ja peale seda talle süsteem valib teise omaniku kuulutuse vahetuseks. Süsteem valib pakutava kuulutuse hinna kategooria järgi, antud kategooria on sisestatud kuulutuse loomisel. Kasutajal on võimalik iseseisvalt valida kuulutuse kategooria ning on võimalus sorteerida pakutavaid kuulutusi loomise järgi, seda on näha joonisel 21.



Joonis 21 Rakenduses pakutud vahetuseks eseme vaade.

Rakenduses on realiseeritud kaebuste esitamise võimalus. Iga kasutaja võib esitada kaebuse igale kuulutusele. Kaebuse võib esitada kahest vaatest: esimene võimalus on pakutavas kuulutuses vahetamiseks vaates ja teine võimalus sõnumi vaates.

Kaebuste töötlemiseks oli loodud teine kasutajate tüüp – ’moderaator’. Antud tüübiga kasutajale on näha kaks menüüriba: esimene tavakasutaja ja teine on moderaatori menüü, kust on võimalik minna üle kaebuste töötlemise vaatesse.

Nagu joonisel 21 on kujutatud, vahetusi pakub rakendus ühekaupa ja kõiki vahetusvariante ühes vaates pole võimalik näha. Et saada võimaluse kirjutada teise eseme omanikule on vaja, et kaks omanikku annaksid enda nõusoleku vahetuseks.

5.2 Edasiarendamise visioon

Edasi on väljatoodud punktidena funktsionaalsus, mida on võimalik lisada tehtud rakendusele.

- Lõputöö käigus on kõik valmistatud, et oleks võimalik realiseerida administraatori vaated. Antud vaates oleks võimalik anda ükskõik millisele kasutajale moderaatori õigused ning andmebaasi tabelis ’group’ olid lisatud kehtivuse väljad.

- Moderaatoritele on võimalus lisada uus vaade esemete ülevaatamiseks, esemete tabelis on lisatud Check_status veerg, mis signaliseerib eseme kontrollimisest.

- Võimalik teha kasutajale teatamist uute kirjade puhul ning sõnumites teha teatamist sõnumite lugemiseks teise kasutaja poolt. Selleks on sõnumite andmebaasi tabelis loodud veerg ’Status’, mille järgi võib antud funktsionaalsust realiseerida.

- Kuulutuse loomisel salvestatakse pildid otse serverarvutile, parem oleks kasutada näiteks amazon s3 teenust kõikide piltide salvestamiseks.

- Teha võimalikakus luua kasutaja ja logida sisse kasutades facebook, google kasutajaid.

- Üle viia kogu majutus kaubanduslikkude serverite peale ja võtta kasutusele HTTPS protokoll.

- GDPR vastavaks veebirakenduse uuendamiseks.

6 Kokkuvõte

Käesoleva töö käigus oli loodud asjade ja teenuste vahetusteks veebirakendus. Antud veebirakenduse kasutamine on tasuta ning tagab anonüümsust teiste kasutajate eest. Töö jooksul oli üle vaadatud olemasolevad müügi- ja vahetamisplatvormid, antud ülevaade veebirakenduse arhitektuurist ja kasutatud raametikest, toodud välja põhiosad arendusest ja testimisest ning koormuse ja tungimise testide tulemus.

Põhinedes funktsionaalsete ja mittefunktsionaalsete nõuete peale oli loodud veebirakendus esemete ja teenuste vahetuseks. Antud veebirakenduse saab kasutaja tasuta kasutada ja lisada piiramatus koguses kuulutusi. Käesolevas rakenduses on loodud kahte tüüpi kasutajad: tava kasutaja ja moderaator. Tavakasutajale on võimalik: lisada uusi kuulutusi, vaadata pakutavaid veebirakendusega vahetusi konkreetsele kuulutusele, suhelda teiste kuulutuste omanikega, lisada kaebusi kuulutuste peale ja konto andmeid muuta.

Veebirakenduse kasutajaliidese jaoks oli kasutatud Aurelia raamistikku, Grails oli kasutuses serveripoolseks arendamiseks. Kogu veebirakendus oli majutatud Google Cloudi VPS peal. Autoriseerimis protokollina oli kasutatud OAuth2.0 ja valitud tokeni tüübiks oli JWT. Saavutatud tulemusega autor sai aru, et raamistike valik oli õige ning kõik nimetatud lõputöös raamistikkude eelised olid kasutatud arendamise käigus.

Tungimise testide tulemuse ja koostatud edasiarendamise visiooni põhjal võib teha järelduse, et asjade ja teenuste vahetuste veebirakenduse arendus pole veel lõpetatud ja on olemas edasiarendamise võimalus. Valitud raamistikega on edasiarendus palju kiirem ja lihtsam.

Kasutatud kirjandus

[1] Aureila [WWW] http://aurelia.io/ (12.03.2018)
[2] Grails [WWW] https://grails.org/ (12.03.2018)
[3] 15 Reasons Why You Should Use The Grails Web Application Framework [WWW] http://www.indicthreads.com/10774/15-reasons-grails-web-application-framework/ (13.03.2018)
[4] DB-Engines Ranking [WWW] https://db-engines.com/en/ranking (28.04.2018)
[5] Создание архитектуры программы [WWW] https://habr.com/post/276593/ (28.04.2018)
[6] Autentimine [WWW] https://courses.cs.ut.ee/2013/infoturve/fall/Main/Autentimine (20.03.2018)
[7] Mõisted [WWW] https://sisu.ut.ee/autentimine/m%C3%B5isted (20.03.2018)
[8] The OAuth 2.0 Authorization Framework [WWW] https://tools.ietf.org/html/rfc6749 (23.03.2018)
[9] Accesse token [WWW] https://www.oauth.com/oauth2-servers/access-tokens/ (21.03.2018)
[10] OAuth 2.0 [WWW] https://oauth.net/2/ (28.04.2018)
[11] What is JSON Web Token? [WWW] https://jwt.io/introduction/ (28.03.2018)
[12] Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 [WWW] https://tools.ietf.org/html/rfc4868 (28.03.2018)
[13] Domain Class Usage [WWW] http://docs.grails.org/3.1.1/ref/Domain%20Classes/Usage.html (01.05.2018)
[14] Basic CRUD [WWW] https://docs.grails.org/latest/guide/GORM.html#ref-command-line-create-domain-class (01.05.2018)
[15] Controller Usage [WWW] https://docs.grails.org/latest/ref/Controllers/Usage.html (01.05.2018)
[16] aurelia-skeleton-navigation [WWW] https://github.com/aurelia (20.04.2018)
[17] Router Configuration [WWW] http://aurelia.io/docs/routing/configuration#basic-configuration (21.04.2018)
[18] Avati vahetusportaal vaheta.ee [WWW] http://vaheta.ee/est/uudised/avati-vahetusportaal-vahetaee (06.05.2018)
[19] Mis asi on okidoki? [WWW] https://www.okidoki.ee/info/about/ (06.05.2018)
[20] Firmast [WWW] https://osta-ee.postimees.ee/firmast (06.05.2018)
[21] Most popular social networks worldwide as of April 2018 [WWW] https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/ (06.05.2018)
[22] Engineered to help business thrive [WWW] https://cloud.google.com/why-google-cloud/ (07.05.2018)
[23] Nginx [WWW] https://nginx.org/en/ (07.05.2018)
[24] Grails Testing Support [WWW] https://testing.grails.org/latest/guide/index.html (07.05.2018)
[25] [Groovy] Annotation Type Rollback [WWW] http://docs.grails.org/3.3.5/api/grails/transaction/Rollback.html (07.05.2018)

[26] A Tale of Two Clouds: Amazon vs. Google [WWW] https://medium.com/@robaboukhalil/a-tale-of-two-clouds-amazon-vs-google-4f2520516a38 (08.05.2018)
[27] The Good and the Bad of ReactJS [WWW] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/ (08.05.2018)
[28] WHO IS MORE DEMANDED IN 2018? AURELIA VS ANGULARJS DEVELOPERS [WWW] https://mobilunity.com/blog/aurelia-vs-angularjs-developers/ (07.05.2018)
[29] Are there any reasons to use PostgreSQL over MySQL? [WWW] https://www.quora.com/Are-there-any-reasons-to-use-PostgreSQL-over-MySQL (08.05.2018)
[30] URLMappings [WWW] http://docs.grails.org/3.1.1/ref/Plug-ins/URL%20mappings.html (09.05.2018)
[31] transactional [WWW] http://docs.grails.org/3.1.1/ref/Services/transactional.html (09.05.2018)
[32] What is the license of PostgreSQL? [WWW] https://wiki.postgresql.org/wiki/FAQ#What_is_the_license_of_PostgreSQL.3F (16.05.2018)

Lisa 1 – Andmebaasi dokumentatsioon

Edasi on väljatoodud iga andmebaasi tabelid ühekaupa koos tabeli väljade kirjeldustega.

Group – Antud tabel hoiab endas kasutajate gruppe. Antud grupid on kasutuses, et tagada juurdepääs valitud rakenduse funktsionaalsust kindlatele kasutajatele. Näiteks, tavakasutajal on “standard“ grupp, mis annab kasutajale juurdepääsu esemete lisamisele, vahetamisele ja teiste kasutajatega suhtlemisele. Kui vahetada gruppi „moderator“ peale, lisandub võimalus kaebuste töötlemiseks.

Veeru nimi	**Veeru tüüp**	**Kirjeldus**
id	bigint	Tabeli rea unikaalne identifikaator.
visitor_id	bigint	Grupiga seotud kasutaja identifikaator.
group_name	varchar(250)	Grupi nimi.
valid_from	timestamp without timezone	Grupi kehtivuse alguse kuupäev.
valid_until	timestamp without timezone	Grupi kehtivuse lõpetamise kuupäev, antud veerg on mittekohustuslik.

Token- Antud tabelit on vaja, et säilitada *Refresh* tokeni sisu. *Refresh* tokeni sisu on vajalik tokenite uuendamisel, mille käigus võrreldakse tulnud *Refresh* tokenit süsteemis olevaga tokeniga.

Veeru nimi	**Veeru tüüp**	**Kirjeldus**
id	bigint	Tabeli rea unikaalne identifikaator.
Refresh_token_hash	varchar(20)	Refresh tokeni sisu, on juhuslikult genereeritud.
Visitor_id	bigint	Kasutaja identifikaator, kellele on antud token väljastatud.

Visitor - kasutaja andmete säilitamiseks tabel. Kasutaja parooli salvestab süsteem andmebaasi krüpteeritud kujul.

Veeru nimi	**Veeru tüüp**	**Kirjeldus**
id	bigint	Tabeli rea unikaalne identifikaator.
email	varchar(200)	Kasutaja sisestatud registreerimise käigus e-post
password	text	Kasutaja sisestatud registreerimise käigus salasõna, salvestatakse krüpteeritud kujul.
name	varchar(255)	Kasutaja sisestatud registreerimise käigus nimi.
surename	varchar(255)	Kasutaja sisestatud registreerimise käigus perekonnanimi
status	char(1)	Antud väli hoiab endas kasutaja staatust. Praegu on kaks varianti: ’A’ ja ’B’. ’A’- tähendab, et kasutaja on aktiivne ja saab veebirakendust kasutada. ’B’- staatus on blokeeritud kasutajatel, antud kasutajatel pole võimalust kasutada veebirakendust.
Telephone_nr	varchar(50)	Kasutaja sisestatud registreerimise käigus telefoninumber, antud väli on mittekohustuslik.
Registration_date	timestamp without timezone	Kasutaja registreerimiskuupäev kellaajaga.
P_hash	varchar(4)	Juhuslikult valitud süsteemiga 4 sümbolid. Kasutatakse, et tõsta parooli säilitavust. Enne parooli krüpteerimist lisatakse need 4 sümbolid parooli külge.

Picture – tabel esemete piltide asukohaga.

Veeru nimi	Veeru tüüp	Kirjeldus
Id	bigint	Tabeli rea unikaalne identifikaator.
Small	varchar(250)	Väikse pildi asukoht.
medium	varchar(250)	Keskmise suurusega pildi asukoht.
Big	varchar(250)	Suure suurusega pildi asukoht.

Item – Kasutajate lisatud asjade ja teenuste andmete hoidmiseks tabel.

Veeru nimi	Veeru tüüp	Kirjeldus
Id	bigint	Tabeli rea unikaalne identifikaator.
Item_owner_id	bigint	Eseme omaniku identifikaator.
Heading	varchar(32)	Kasutaja lisatud eseme pealkiri.
Item_info	varchar(2000)	Kasutaja lisatud eseme lisainfo.
Picture_id	bigint	Identifikaator piltide tabeli peale. Viitab kasutaja lisatud eseme pildile.
Creating_date	timestamp without timezone	Eseme lisamise kuupäev koos kellaajaga.
Item_category_id	bigint	Viide eseme kategooria tabeli peale.
Price_category	bigint	Viide hinna kategooria tabeli peale.
Check_status	char(1)	Antud väli tähistab moderaatori kontrolli läbimist. Antud ajal on kolm staatuse varianti. ‘W’- läbivaatamine, ‘A’- moderaator kinnitas eseme lisamise. ‘B’- moderaator lükkas tagasi eseme lisamise.
Active	char(1)	Antud väli tähistab eseme aktiivsust. Väljal on 3 väärtust. ’A’ – aktiivne ese, ’B’- blokeeritud ese ja ’R’- kustutatud ese.

Price_category – Tabel esemete hinnakategooriate hoidmiseks. Antud hinnakategooriat on esitatud eseme loomisel ning süsteemis kasutatakse vahetuspakkumiste otsimiseks.

Veeru nimi	**Veeru tüüp**	**Kirjeldus**
Id	bigint	Tabeli rea unikaalne identifikaator.
Price_category	varcahr(50)	Hinnakategooria.

Message – sõnumite hoidmiseks tabel.

Veeru nimi	**Veeru tüüp**	**Kirjeldus**
Id	bigint	Tabeli rea unikaalne identifikaator.
Connection_id	bigint	Esemete seose identifikaator.
From_item	bigint	Eseme identifikaator. Viide esemele kellelt sõnum on saadetud.
Message	varchar(2000)	Sõnumi sisu.
Date	timestamp without timezone	Sõnumi loomise kuupäev koos ajaga.
Status	char(1)	Sõnumi staatus. Rakenduses on kasutuses, et näidata, kas sõnum on loetud või mitte. Staatusel on kaks väärtust: ’W’- sõnum pole läbi loetud, ’D’ – sõnum läbi loetud.

Complaint – Kaebuste hoidmiseks tabel.

Veeru nimi	Veeru tüüp	Kirjeldus
Id	bigint	Tabeli rea unikaalne identifikaator.
From_visitor_id	bigint	Kasutaja identifikaator. Viide kasutajale, kellelt kaebus on esitatud.
Reason	varchar(2000)	Kaebuse põhjus, tekst lisatud kaebuse loomisel.
Status	char(1)	Kaebuse staatuse väli. On kaks väärtust, 'W' – kaebus järjekorras moderaatori üle vaatamisele, 'A' – kaebus on üle vaadatud moderaatori poolt.
To_item_id	bigint	Ese identifikaator. Viide esemele, mille peale tuli kaebus.
Date	timestamp without timezone	Kaebuse loomise kuupäev.

Connection – Tabel hoiab kasutajate esemete vahel seoseid. Antud seoseid kasutatakse, et aktiveerida võimalust saata teistele omanikele sõnumeid.

Veeru nimi	Veeru tüüp	Kirjeldus
Id	bigint	Tabeli rea unikaalne identifikaator.
From_item_id	bigint	Eseme identifikaator.
To_item_id	bigint	Eseme identifikaator.
Status	char(1)	Seose staatus. Võib olla 4 erinevat väärtust: 'W'- ootab teise eseme omaniku otsust, 'A'- aktiivne seos , 'D'- üks omanik keeldus vahetusest , 'R'- kustutatud seos.
Modify_date	timestamp without timezone	Viimase muudatuse kuupäev kellaajaga.

Item_category – Tabel esemete kategooriate hoidmiseks. Antud hinnakategooriat on esitatud eseme loomisel ning süsteemis kasutatakse lisaparameetrina pakutava eseme otsimiseks.

Veeru nimi	**Veeru tüüp**	**Kirjeldus**
Id	bigint	Tabeli rea unikaalne identifikaator.
Category_name	varchar(50)	Eseme kategooria nimi.