TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Frank Lüübek 120926IAPB

KÕNELIITUMISE VOO ARENDAMINE MOBIILIOPERAATORILE

Bakalaureusetöö

Juhendaja: Ago Luberg
Magister
Lektor

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: Frank Lüübek

[07.05.2018]

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua Elisa iseteenindusbüroosse võimalus liituda kõnepakettidega ja seejuures neid vastavalt vajadusele personaalselt seadistada, ning valida soovi korral paketile sobivaid lisateenuseid ja seadmeid.

Töö käigus ehitatakse olemasoleva raamistiku peale uus, kõneliitumise spetsiifiline funktsionaalsus koos *front-end*, *back-end* ja andmebaasi komponentidega.

Töös kirjeldatakse loodud funktsionaalsuse arendamise protsessi, selle arhitektuuri ja kasutatud tehnoloogiaid. Töö tulemiks on töötav ja esialgu ainult klienditeenindajatele kasutamiseks mõeldud funktsionaalsus, mille abil on võimalik liituda kõnepakettidega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 4 peatükki, 12 joonist, 1 tabelit.

Abstract

Development of a Voice Package Subscription Flow for a Mobile Operator

The aim of this bachelor's thesis is to develop an application for the self service bureau of Elisa Eesti AS for subscribing to voice packages, configuring them according to one's personal needs and choosing additional services or devices suitable for the package.

During the course of the work a new functionality along with the front-end, back-end and database components will be built on top on an existing frame.

The thesis describes the process of the development, its architecture and the technologies that have been used during development. The outcome of the thesis will be a working application, initially meant to be used by customer service representatives only, that would provide the opportunity to subscribe to voice packages.

The thesis is in Estonian and contains 28 pages of text, 4 chapters, 12 figures, 1 tables.

Lühendite ja mõistete sõnastik

ITB	Elisa Eesti veebipõhine iseteenindusbüroo
CI	*Continuous integration* ehk pidevintegreerimine on arenduspraktika, mille kohaselt peavad arendajad koodi ühisesse repositooriumisse laadima mitu korda päevas [24].
front-end	Veebilehe kasutajaliides, mis võimaldab suhtlemist *back-end* rakenduse ja andmebaasiga.
back-end	Veebirakenduse serveripoolne osa, mis tegeleb *front-end*'ist tulnud päringute töötlemise ja andmete väljastamisega.
REST	*Representational State Transfer,* üks populaarsemaid süsteemide liidestamise rakenduste tüüpe [19]
UX	*User experience* ehk kasutajakogemus
liides	ITBs valitud pakett koos kõigi lisateenuste ja seadmetega
DAO	*Data Access Object* ehk objekt, mis võimaldab rakendustel suhelda andmebaasiga
IDE	*Integrated Development Environment* ehk integreeritud programmeerimiskeskkond

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Elisa Eesti AS on Elisa kontserni osa. Kontserni emamaal Soomes välja arendatud rakendused on ka Elisa Eesti koodibaasi põhjaks. Kuigi suures osas on Eestis võetud tehnoloogiliselt uus suund, ei saa olemasolevaid rakendusi veel niipea kasutuselt maha võtta, kuna suur osa vajalikku funktsionaalsust eksisteerib ainult nendes. Täpselt sama lugu on ka käesolevas töös käsitletud projektiga – kõneliitumisega. Elisa eesmärgiks ongi tuua uude ITB'sse (iseteenindusbüroosse) kõneliitumise võimalus, mis kujutab endast võimekust liituda kõnepakettidega ja tellida neile mugavalt ka lisateenuseid ja seadmeid. Projekt tervikuna on äärmiselt mahukas ja praeguse seisuga on plaanis see kasutajateni tuua 9 eraldiseisva inkrementaalse tükina ehk *release'*ina. Käesolev lõputöö keskendub ühele osale *release'*ist järjekorranumbriga 2, milleks on liitumise voog, mille käigus saab kasutaja valida endale sobiva kõnepaketi, valida ja seadistada endale sobivad mobiilinumbrid, ning valida neile soovi korral ka lisateenuseid ja seadmeid. Töö tulemusena valmivad või võetakse juba olemasolu korral taaskasutusele voo kõik erinevad sammud kuni toote ostukorvi suunamiseni.

1.1 Eesmärgid

Töö põhiliseks eesmärgiks on luua Elisa ITBsse kõneliitumise voog, kus on võimalik endale valida soovikohased mobiilinumbrid, neid vastavalt vajadusele seadistada, valida interneti lisateenuseid ja seadmeid, ning valida ka üldiseid lisateenuseid.

Töö teiseks eesmärgiks on kirjutada koodi, mis on vastavuses üldiste parimate praktikatega, ning järgida Elisa-sisest nõuet, et kirjutatud kood peab vastama koodianalüsaatorprogrammi Sonari [22] nõuetele, millest tähtsamad on, et koodis ei tohi olla kriitilisi vigu ja et uue koodi testiga katvus peab olema vähemalt 70%.

1.2 Metoodika

Töö praktilise osa käik vastab Elisa arendusprotsessile. Esmalt selgitab töö tellija üldist töö olemust ja pannakse paika tellimuse nõuded. Seejärel jagab analüütik kogu tellimuse väiksemateks eraldiseisvateks arendusvalmis tükkideks. Seejärel vaadatakse neile väiksematele töödele peale kogu tiimi koosseisus, räägitakse täpsemalt lahti ülesande tehniline taust ja antakse mõningaid suunitlusi lahenduskäigule, ning lõpuks antakse tööle keerukushinnang. Järgmise sammuna võetakse töö arendusse. Kui töö valmis, siis laetakse see versioonihaldustarkvara Git'i kasutades üles koodihoidlasse, kust CI- ehk pidevintegratsioonirakenduse Jenkinsi abil see testkeskkonda üles laetakse.

Lõputöö käigus uusi rakendusi ei looda, vaid arendatakse olemasolevaid rakendusi – seetõttu on lihtsam järgida olemasolevate rakenduste arhitektuurilisi ja stiililisi nõudeid. ITB *front-end* rakendus on kirjutatud kasutades AngularJS'i, HTML'i ja CSS'i. ITB põhilised *back-end* rakendused, mida käesolevad töös täiendatakse, on REST-põhised (üks levinumaid süsteemide liidestamise rakenduste tüüpe), kirjutatud Javas ja kihilist arhitektuuri kasutades. Andmebaasisüsteemiks on MySQL.

Töö käigus luuakse nii vajalikud AngularJS komponendid, HTTP päringud ja nende *back-end* loogika, kui ka andmebaasipäringud. Ettevõttesisese protsessi kohaselt luuakse rakenduse kujunduslik pool spetsiaalse UX (kasutajakogemuse) tiimi poolt, kuid tulenevalt AngularJS'i omapäradest, mida kasutatakse rakenduse *front-end* poole arendamiseks, on HTML dokumenti igal juhul vaja täiendada, ning ka CSS'i täiendamise vajadus pole välistatud.

1.3 Töö ülesehitus

Töö esimeses, analüüsi osas pannakse kirja individuaalsete tööde olulisimad funktsionaalsed ja mittefunktsionaalsed nõuded. Lisaks tuuakse välja tükkideks jagatud tööde nimekiri.

Teises osas vaadeldakse töö arenduse käiku. Esiteks antakse ülevaade töö käigus kasutatud tehnoloogiatest ja seejärel tutvustatakse rakenduste arhitektuuri. Veel kirjutatakse ükshaaval lahti eelmises peatükis kirja pandud tööd, ning kirjeldatakse põgusalt nende funktsionaalsusi ja eesmärke. Viimasena antakse ülevaade töö käigus tehtud programmikoodi testimisest.

2 Analüüs

Selles peatükis kirjeldatakse analüütiku ja arendajate kokku lepitud olulisimaid funktsionaalseid ja mittefunktsionaalseid nõudeid, ning jagatakse töö loogilisteks tükkideks, mida on võimalik üksteisest sõltumatult arendada.

2.1 Funktsionaalsed nõuded

- Voos peab olema võimalik valida endale soovitav mobiilinumber.
- Mobiilinumbreid peab olema võimalik endale broneerida.
- Mobiilinumbreid peab olema võimalik sisestatud mustri järgi otsida.
- Broneeritud ja vabad numbreid peavad olema selgelt eristatavad.
- Numbri aktiveerimise aega peab olema võimalik ise määrata.
- Korraga peab olema võimalik valida kuni kakskümmend mobiilinumbrit.
- Igat valitud numbrit peab olema võimalik eraldi seadistada.
- Igale mobiilinumbrile peab olema võimalik valida interneti lisateenust.
- Igale mobiilinumbrile peab olema võimalik valida seadet (näiteks mobiiltelefoni).
- Peab saama valida soovitud seadme värvi.
- Igale mobiilinumbrile peab olema võimalik valida lisateenuseid.
- Lisateenuste parameetreid peab olema võimalik voos seadistada.
- Voogu peab olema võimalik läbida ilma ühtegi lisateenust ega seadet valimata.
- Voo siseselt peab olema võimalik ka juba läbitud sammudesse tagasi minna.
- Voos edasi liikudes peavad säilima tehtud valikud ilma neid andmebaasi salvestamata.
- Voos tehtud valikud peavad olema selgelt nähtaval senikaua, kuni voos ollakse.
- Voo lõpust peab olema võimalik minna ostukorvi lehele.
- Ostukorvi lehelt peab olema võimalik minna voogu tagasi, kuid ainult ühe liidesega (iseteenindusbüroos valitud pakett koos kõigi lisateenuste ja seadmetega) korraga.
- Ostukorvist voogu tagasi minnes peavad eelnevalt tehtud valikud olema säilinud.

2.2 Mittefunktsionaalsed nõuded

- Disain peab olema kohandatav igale ekraanitüübile ja platvormile, st olema kohanduv.
- Programmikood peab olema kirjutatud järgides ettevõttesiseseid kokkuleppeid.
- Programmikood peab olema minimaalselt 70% ulatuses kaetud automaattestidega.
- Arendatud programm peab kasutajat arusaadavalt informeerima tekkinud vigadest.
- Kasutajaliides peab olema tõlgitav nii eesti kui vene keelde.

2.3 Töö jaotus eraldiseisvates osadeks

Tuginedes eelnevalt kirja pandud nõuetele jaotati tööd eraldiseisvateks ja üksteisest sõltumatuteks osadeks. Järgnevalt on esitatud suuremate tööde loetelu.

- Numbrivaliku komponent.
- Numbrite seadistamise komponent.
- Valitud liideste eristamise komponent.
- Interneti lisateenuse komponent.
- Seadmevaliku komponent.
- Lisateenuste komponent.

3 Arendus

Selles peatükis kirjeldatakse lühidalt arendusprotsessi, ning antakse ülevaade kasutatud tehnoloogiatest, arendatud komponentidest, arhitektuurist ja testimisest.

Kuna arendus käib ettevõttesisese protsessi järgi, siis ei tegele projektiga mitte ainult autor, vaid ka kogu tema tiim. Kuna kõnealune arendusprojekt on niivõrd suur ja autori tiimi kuulub lõputöö tegemise seisuga lisaks temale tervelt 5 täiskohaga arendajat, siis ei olnud autoril võimalik kõiki lõputöö osasid ise arendada. Järgnevalt on välja toodud täielikult autori poolt arendatud voo osad.

- Numbrite seadistamise komponent
- Valitud liideste eristamise komponent
- Lisateenuste komponent

Lisaks on autor ka teiste sammude raames ellu viinud väiksemaid juurdearendusi ja parandusi, kuid neid siinkohal ette loetlema ei hakata.

3.1 Kasutatud tehnoloogiad

Java on ühtlasi nii programmeerimiskeel kui arvutusplatvorm [1]. Java programmeerimiskeeles on kirjutatud kõik antud lõputöös käsitletud *back-end* rakendused.

Maven on Apache Software Foundation'i poolt arendatud tööriist, mille abil on võimalik lihtsa vaevaga ehitada ja hallata iga Java-põhist projekti. Maveni põhilised eesmärgid on rakenduste ehitusprotsessi ühtlustamine ja lihtsustamine, projekti kohta kasuliku info kättesaadavaks tegemine, arenduse parimate praktikate järgimise ja uuenduste paigaldamise lihtsamaks muutmine [2]. Mavenit kasutatakse töö vältel *back-end* rakenduste ehitamiseks ja nende sõltuvuste haldamiseks.

Guice on sõltuvuste süstimise raamistik Java jaoks. Guice'i kasutamine muudab Java koodi ja testide kirjutamise oluliselt lihtsamini hallatavamaks ja teeb lihtsamaks programmikoodi taaskasutamise [3].

Mockito on ühiktestide kirjutamist hõlbustav raamistik. Mockito võimaldab ühiktestides päris komponentide käitumist juhtida ja kontrollida [4]. Ühiktestide kirjutamisel ei ole alati võimalik ilma abivahenditeta testimiseks isoleerida ühte kindlat ühikut – suuremates rakendustes on tavaline, et igal komponendil on oma välised sõltuvused ilma milleta ta toimida ei saa. Mockito abil on võimalik need sõltuvused lihtsa vaevaga ära kaotada (Mockito termineid kasutades ära *mockida*), mis teeb väga lihtsaks Java meetodite eraldi testimise.

JUnit on Java programmeerimiskeele ühiktestimise raamistik [5]. JUniti abil on võimalik testida Java programmikoodi käitumist läbi integratsiooni- kui ühiktestimise. Kuigi integratsioonitestide kirjutamine on üsnagi tavaline ja isegi soovitatud praktika, ei kirjutata selle lõputöö raames ühtegi integratsioonitesti – rõhuasetus jääb praeguse skoobi kohaselt ühiktestidele.

Jetty on Java HTTP veebiserver, mille abil saab serveerida nii staatilist kui dünaamilist sisu. Jettyt on võimalik kasutada nii rakendustest eraldiseisvana, kui rakendusse integreerituna [6]. Lõputöö kirjutamise hetkel on asjassepuutuvates *back-end* rakendustes kasutatud mõlemat versiooni, kuid üldine suunitlus on üle minna rakendusse integreeritud Jetty kasutamisele.

MySQL on praeguse seisuga maailma populaarseim avatud lähtekoodiga relatsiooniline andmebaasisüsteem [7].

AngularJS on avatud lähtekoodiga Javascriptil põhinev *front-end* veebirakenduste raamistik, mis on loodud üheleherakenduste arendamiseks [8]. Lõputöö kirjutamise seisuga on ilmunud juba stabiilne Angular 5, kuid lõputöös on kasutatud Angulari esimest suuremat väljalaset, milleks on AngularJS. Põhjus, miks süsteemides pole Angulari uuemale versioonile uuendatud on lihtne – AngularJS-i järglane, Angular 2, on kirjutatud puhtalt lehelt ega tal pole oma eelkäijaga muud ühist kui arhitektuurilised põhimõtted [8]. Suure süsteemi kolimine Angular 2+ peale on väga töömahukas, seepärast jätkatakse lõputöös AngularJS'i kasutamist.

HTML on enimlevinud märgistuskeel veebilehtede ja –rakenduste loomiseks [9].

CSS on kujunduskeel, mille abil saab kirjeldada ja juhtida HTML elementide kujundust ja paigutust veebilehel [10].

NPM kujutab endast kahte asja. Esiteks on ta veebipõhine avatud lähtekoodiga Javascripti projektide hoidla. Teiseks on ta käsureapõhine tööriist selle hoidlaga suhtlemiseks, mis on mõeldud hoidlast projektide alla laadimiseks, versioonihalduseks ja sõltuvuste haldamiseks [11].

Karma on Javascripti testide jooksutamise tööriist, mis loob testide jooksutamiseks oma veebiserveri ja jooksutab teste kõigis konfiguratsiooniga määratud veebibrauserites [12]. Kasutajal on võimalik konfiguratsiooniga määrata, millistes brauseritest ta oma ühikteste jooksutada tahab. Veebibrauserite töös on endiselt palju erinevaid nüansse, seega on mängib Karma väga olulist rolli rakenduse testimise elutsüklis.

Jasmine on avatud lähtekoodiga Javascripti testimise raamistik [13]. Käesoleva lõputöö raames on kõik *front-end* rakenduste testid kirjutatud Jasmine-i abil.

Git on hetkeseisuga enim kasutatav versioonihaldustarkvara maailmas. Erinevalt teistest populaarsetest versioonihaldussüsteemidest, nagu Subversion ja CVS, ei hoia Git kogu repositooriumit ühes keskses hoidlas, vaid igal inimesel, kes omab repositooriumile ligipääsu, on oma kohalik koopia kogu versioneeritavast süsteemist [26]. See muudab kogu hallatava süsteemi palju paindlikumaks ja jätkusuutlikumaks.

Jenkins on tööriist, mille eesmärk on teha rakenduste pidevintegratsiooni protsess lihtsamaks. Kuigi Jenkins ei kaota vajadust iga erineva integratsiooni sammu kohta eraldi skriptide kirjutamiseks, saab tema abil enamike tööde eraldi käivitamise *pipeline* ehk torujuhtme meetodil automatiseerida [27]. Sisuliselt tähendab see, et ühe nupulevajutusega käivituvad üksteise järel õiges järjekorras kõik X erinevat tööd, ning tulemusena saab arendatav rakendus vastavasse keskkonda üles laetud, värskendatud ja käivitatud.

3.2 Arendusprotsess

Elisas on 6 erinevat arendustiimi, ning reeglina töötavad tiimid üksteisest sõltumatute töödega. Seetõttu on sellise arendusprotsessi lahutamatuks osaks kvaliteetne versioonihaldustarkvara, milleks kasutatakse Git'i. Selle üks suurimaid eeliseid on kiire harude süsteem, mis võimaldab kiiresti luua eraldiseisvaid koopiaid lähtekoodist, mille kallal siis ilma teisi mõjutamata saab töötada. Kuna koodibaas on kõigil sama ja erinevad tööd kohati ka suuremal määral kattuvad, siis on harude loomise võimalus täiesti kriitilise tähtsusega.

Tiim tegeleb korraga enamasti ühe suure tööga, mida nimetatakse *epic'*uks. Alles arendusjärgus oleva *epic'*u raames laetakse kirjutatud kood üles selle jaoks loodud *feature*-harusse. Kui *epic* on valmis arendatud, siis ühendatakse (*merge*) see haru peaharuga. Iga nädala kuni kahe nädala tagant tehakse peaharust uus *release*-haru, mille pealt uuendatakse süsteeme.

Iga arendaja teeb esmase töö oma kohalikus masinas. Kui töö valmis, siis laetakse see Git'i abil üles oma *feature* harusse. Seejärel ühendatakse Jenkinsi abil kõik *feature* harud peaharuga ja tulemusena saadud kood laetakse rakenduste kaupa üles ühisesse testkeskkonda. Kuna *feature*-harusid eksisteerib korraga päris palju, siis nende kõigi kokku sulandamine ei pruugi alati õnnestuda. Sellepärast ongi iga arendaja kohus pärast tehtud tööd oma kood testkeskkonda panna ja vajadusel kohe tekkinud konfliktid lahendada. Testkeskkond annab esimese aimduse kas ja kuidas erinevate tiimide arendused koos töötavad. Rakenduste töös välja tulnud vead parandatakse enamasti *feature*-harus.

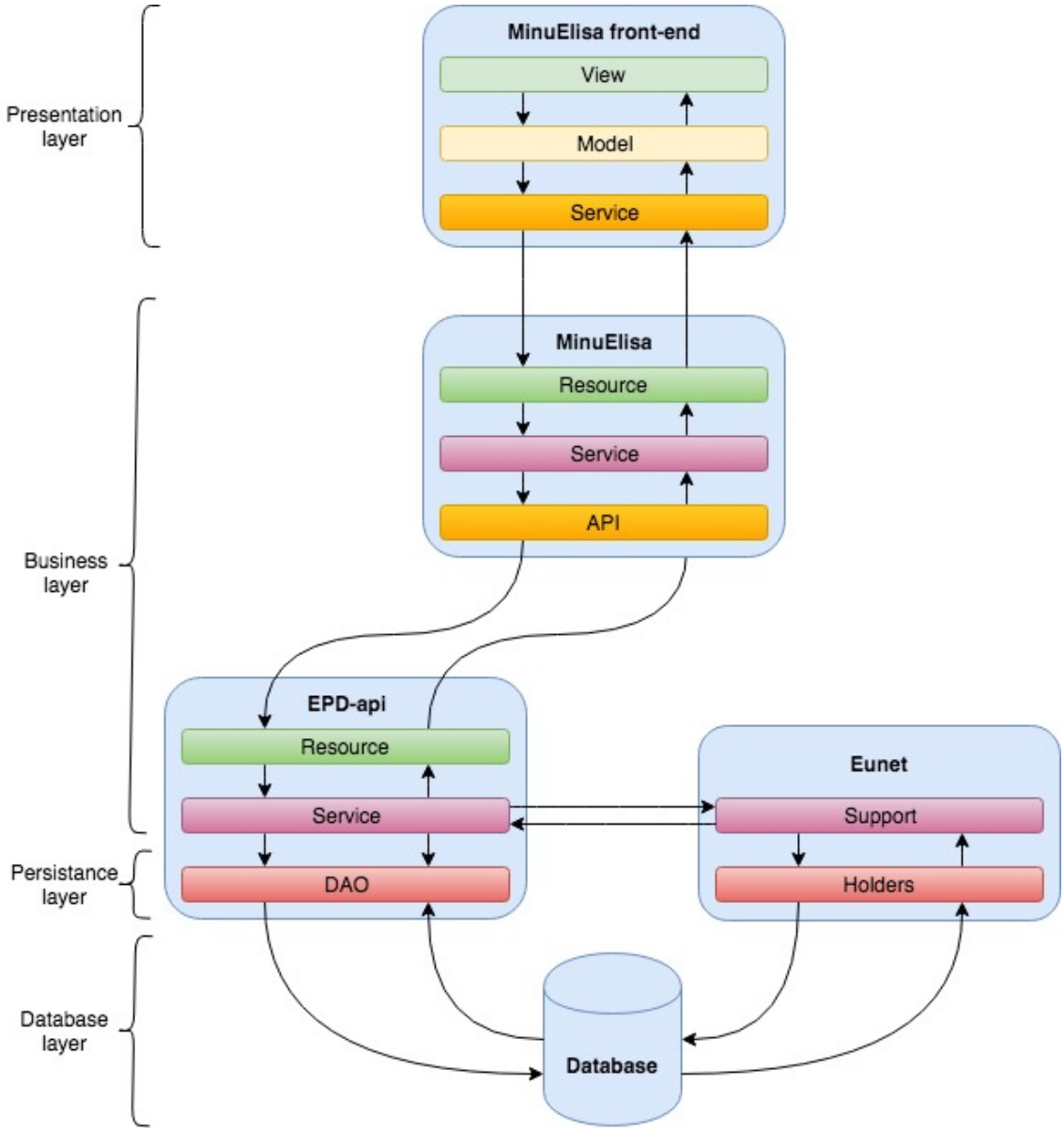
Release-haru tehakse vähemalt nädal enne süsteemide uuendamist ehk *live*'i. Selleks peab peaharus olev kood vastama koodianalüsaatorprogrammi Sonari nõuetele. *Release*-haru eraldi testimiseks on loodud veel üks keskkond – *prelive*. Selles keskkonnas on rakendused ehitatud üksnes järgmisena *live'*i mineva haru koodi pealt eesmärgiga kõik tehtud arendused enne *live'*i veel põhjalikult üle testida. Seda teevad enamasti täiskohaga testijad, ning seda nii läbi manuaal- kui automaattestimise. *Live'*i kuupäevad on tavaliselt eelnevalt paika pandud, kuid kui selleks kuupäevaks pole kõik veel testitud, või on rakenduste tööd endiselt vigu, siis lükatakse *live* edasi.

3.3 Arhitektuur

Lõputöö funktsionaalsus arendatakse iga rakenduse raames juba olemasolevale raamile ja seega on töö käigus järgitud juba kehtivad arhitektuurinõudeid ja –tavasid.

Üldiselt jaotub kogu projekt kaheks osaks: *front-end* ja *back-end*. *Front-end* hõlmab endas sisuliselt kõike, mida kasutaja näeb kui ta veebilehte külastab. Tänapäeva levinuimateks *front-end* tehnoloogiateks on HTML, CSS ja Javascript (või selle raamistikud/teegid), mida omakorda kontrollib veebibrauser [14]. Rakenduse *back-endi* peamine ülesanne on kliendipoolse rakenduse ühendamine andmebaasiga, olulise äriloogika kapseldamine, ning andmete hoidmine ja väljastamine. *Back-end* rakendused töötavad enamasti eraldiseisvates veebiserverites ja tänapäeval on serveripoolsete rakenduste kirjutamiseks loodud palju programmeerimiskeeli [15]. Selles lõputöös on *back-end* rakendused kirjutatud Javas. Andmebaasidega suhtlemiseks on kasutusel relatsiooniliste andmebaaside haldamissüsteem MySQL.

Süsteem kui tervik on ehitatud kihilise arhitektuuri mustri järgi, mis on ka enamike Java rakenduste *de facto* standard [18]. Kihilise arhitektuurimustri järgi ehitatud rakenduste komponente võib vaadelda vertikaalsete kihtidena, kus iga erinev kiht täidab oma spetsiifilist ülesannet. Erinevate kihtide eesmärk on muuta abstraktseks ja kapseldada nendes sisalduv loogika, et ülejäänud kihid saaksid olla teistest sõltumatud [18]. Lõputöös kasutatud süsteem koosneb neljast üldisest kihist ja üheteistkümnest alamkihist (joonis nr 1).



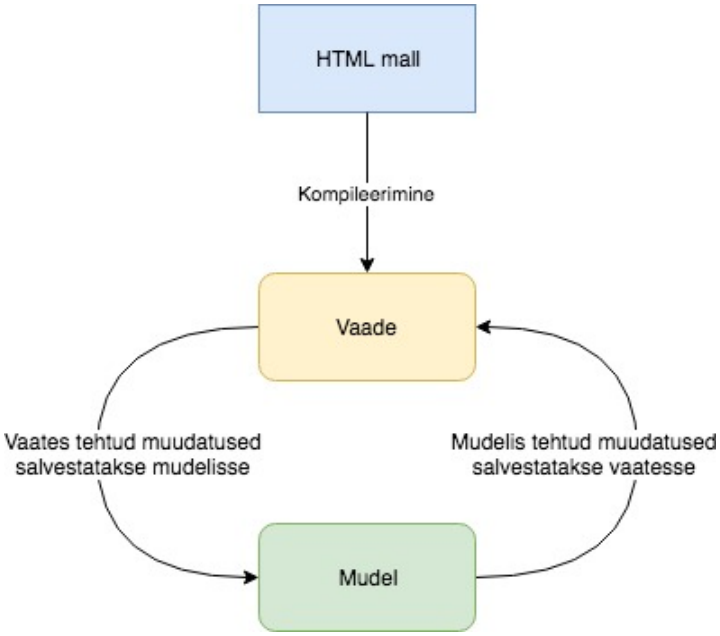
Joonis 1. Süsteemi kihiline arhitektuur

3.3.1 Front-end

Front-end arhitektuuri iseärasused tulenevad kasutatud JavaScripti raamistikust, AngularJS'ist, mis on dünaamiliste üheleherakenduste ehitamiseks mõeldud strukturaalne raamistik [16]. HTML ise on mõeldud staatiliste dokumentide deklareerimiseks, kuid tänapäeva veebirakenduste loomiseks puhtalt temast ei piisa. AngularJS on see, mis HTML oleks olnud, kui ta oleks loodud veebirakendusi silmas pidades [16].

AngularJS'i üheks tähtsaimaks kontseptsiooniks on kahesuunaline andmete sidumine (joonis nr. 2). Esiteks kompileeritakse veebibrauseris HTML ja selle tulemuseks on kompileeritud vaade. Vaates tehtud muudatused kajastatakse automaatselt mudelisse ja ka vastupidi. Seetõttu on rakenduse seisu ainsaks tõe allikaks mudel, kuna vaade on lihtsalt mudeli projektsioon ekraanil. Tänu sellele saab kontrolleri loogika hoida lahus vaatest, mis omakorda lihtsustab oluliselt koodi testimist [17].

Süsteemi kihilise arhitektuuri joonisel (joonis nr. 1) moodustab *front-end* projekt süsteemi presentatsioonikihi. Alamkihtide mõistes jaguneb see kolmeks: vaate-, mudeli- ja teenuse kiht. Vaate kihi moodustavad HTML failid koos CSS failidega, mis annavad veebilehele struktuuri ja välimuse. Mudeli kihis on vaate kihti juhtivad AngularJS'i moodulid koos oma kontrolleritega, mis hoiavad endas vaate kihis näidatavaid objekte ja teevad nendega vajalikke toiminguid. Mudeli kiht küsib vajalikku infot teenuse kihist, mis koosneb AngularJS moodulite *service*'itest. Teenuse kihi komponendid saavad oma info omakorda läbi REST päringute, mis tehakse *back-end* rakenduste pihta. Äriloogika *front-end* projekti viimine ei ole Elisas hea tava. Kuigi seda tänapäeval üha enam tehakse, ei ole see hea nimelt turvalisuse seisukohalt [22]. Alati pole võimalik kogu äriloogikat *front-endist* välistada, kuid sellisel juhul peab kindlasti andmete valiidsust kontrollima *back-endis* [22].



Joonis 2. AngularJS kahesuunaline andmete sidumine

Voo piires andmete säilitamine ilma neid andmebaasi salvestamata on lahendatud kasutades HTML5 sessiooni salvestamist. Sessiooni salvestamine (*session storage*) võimaldab salvestada infot otse brauserisse, kuid seda ainult ühe kehtiva sessiooni piires [25]. See tähendab, et brauseris vahelehe sulgemisel, uue vahelehe avamisel või vahelehe sisu värskendamisel luuakse uus sessioon, ning seni salvestatud info läheb kaduma. Selleks kasutatavad komponendid eksisteerivad juba olemasolevas raamistikus, ning iga komponendi piires tuleb lihtsalt andmete salvestamine ja sessioonist lugemine eraldi implementeerida. Sessioonis hoitakse andmeid andmebaasile võimalikult lähedase struktuuriga, et hilisem sessioonist andmebaasi salvestamine, mis toimub voost ostukorvi liikudes, oleks lihtsam.

3.3.2 Back-end

Projekti *back-end* poole moodustavad süsteemi kihilise arhitektuuri joonisel (joonis nr. 1) MinuElisa, EPD-api ja Eunet rakendused koos andmebaasiga. MinuElisa ja EPD-api rakendused on ehitatud REST põhimõtteid [19] järgides. Kuigi REST-põhised rakendused võivad kasutada andmete edastuseks pea kõiki erinevaid protokolle, siis enamasti, nagu ka käesoleva lõputöö raames, kasutavad nad HTTP protokolli [19]. REST-põhiste rakenduste peamine eelis on nende paindlikkus. Andmed ei ole seotud

kindlate ressursside või meetoditega – seega suudavad sellised rakendused hakkama saada erinevat tüüpi päringutega ja tagastada erinevates formaatides andmeid. Lisaks saab REST rakendus päringuid vastu võtta üksteisest sõltumatult ja neid ka üksteisest sõltumatult töödelda [19]. Seega peavad REST rakendused arvestama võimaliku suure koormusega.

MinuElisa rakendus koosneb kolmest kihist: ressursi-, teenuse- ja *api*-kiht. Ressursikiht võtab vastu HTTP päringuid, ning annab need teenusekihile edasi. Teenusekihi üheks tähtsaimaks ülesandeks on ITB kasutajate sessioonihaldus. Ülejäänud REST päringud annab teenusekiht edasi *api*-kihile. Viimane saadab kõik päringud edasi HTTP protokolli kasutades järgmistele rakendustele.

EPD-api rakendus koosneb samuti kolmest kihist: ressursi-, teenuse- ja DAO-kiht (DAO on objekt, mis võimaldab rakendustel suhelda andmebaasiga). Ressursikiht töötab samal põhimõttel, mis MinuElisa oma, võttes vastu HTTP päringuid ja edastades need teenusekihile. Teenusekiht kapseldab endas suurel hulgal äriloogikat, ning olenevalt meetodist kutsub välja DAO-kihi klasse või siis Eunet rakenduse Support klasse. DAO ehk andmetele ligipääsu kiht vastutabki Java päringute ja andmebaasi ühendamise ja andmebaasist tulevate andmete Java objektideks teisendamise eest.

Eunet rakendus koosneb kahest, *support* ja *holder* kihist. Support kiht on nii funktsionaalsuselt kui eesmärgilt sarnane EPD-api teenuse kihiga – ta peidab endas olulist äriloogikat ja kasutab andmebaasi tabelite põhjal loodud *holder* kihi moodustavaid objekte, mida nimetataksegi *holderiteks*, andmete salvestamiseks, muutmiseks ja kustutamiseks.

Eraldi kihiks on ka loetud andmebaas.

3.3.3 REST päringud

Järgnevalt on kirjeldatud arendatud komponentide raames tehtavad REST päringud andmete küsimiseks andmebaasist. Päringud on kõik tehtud HTTP protokolli kasutades, ning alljärgnevas tabelis (tabel nr. 1) on kirjeldatud päringu eesmärk, tüüp ja selle aadress.

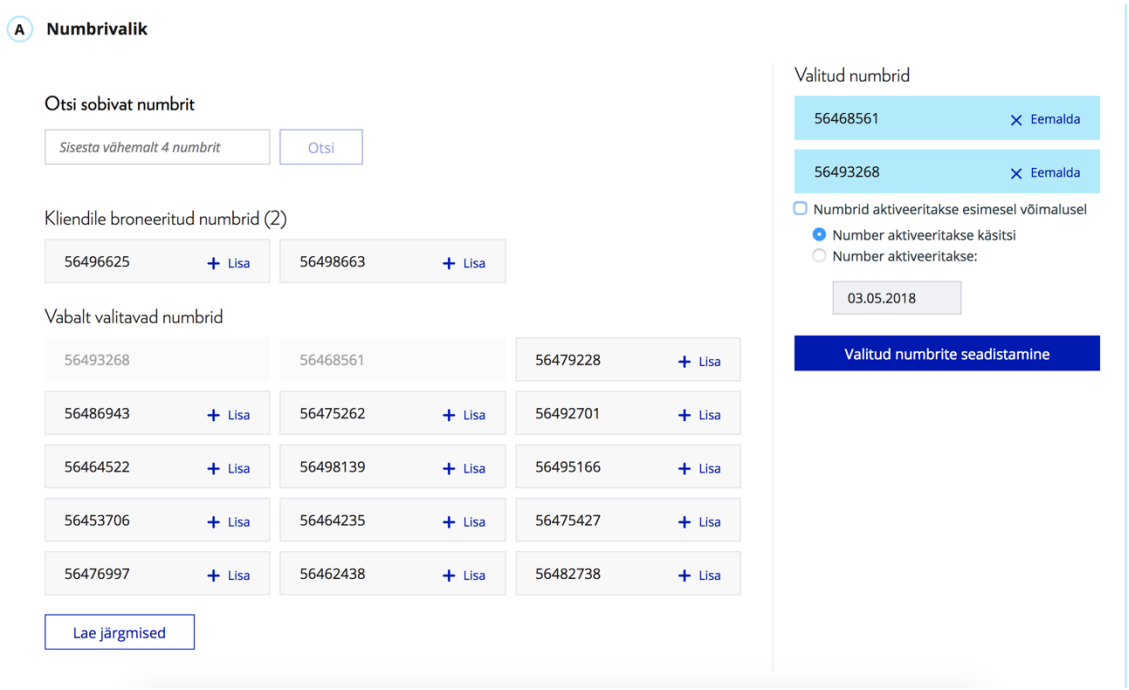
Tabel 1. REST päringute ülevaade.

Eesmärk	**Päringu tüüp**	**Aadress**
Mobiilinumbrite reserveerimine	POST	rest/phone-number/reserve
Vabade mobiilinumbrite arvu pärimine	GET	rest/phone-number/limited/
Toodete avamishindade pärimine	GET	rest/priceLists/opening-product-prices
Toote vaikesätete pärimine	GET	rest/products/default-product-configuration
Numbriomaniku kontaktinfo pärimine	GET	rest/customer/contactInfo/owner/basic
Interneti lisateenuse toode filtreerimine	GET	rest/additional-services/filterbymainproduct/multiple
Interneti lisateenuse toode pärimine	GET	rest/products/byproductids
Seadmete filtreerimine ja pärimine	GET	rest/handsetcampaigns/productId
Lisateenuste filtreerimine ja pärimine	GET	rest/additional-services/filterbymainproduct/multiple

3.4 Arendatud komponendid

Järgnevas alapeatükis tutvustatakse arendustöö raames valminud voo samme ja nende raames arendatud AngularJS komponente.

3.4.1 Numbrivaliku komponent



Joonis 3. Numbrivaliku komponent

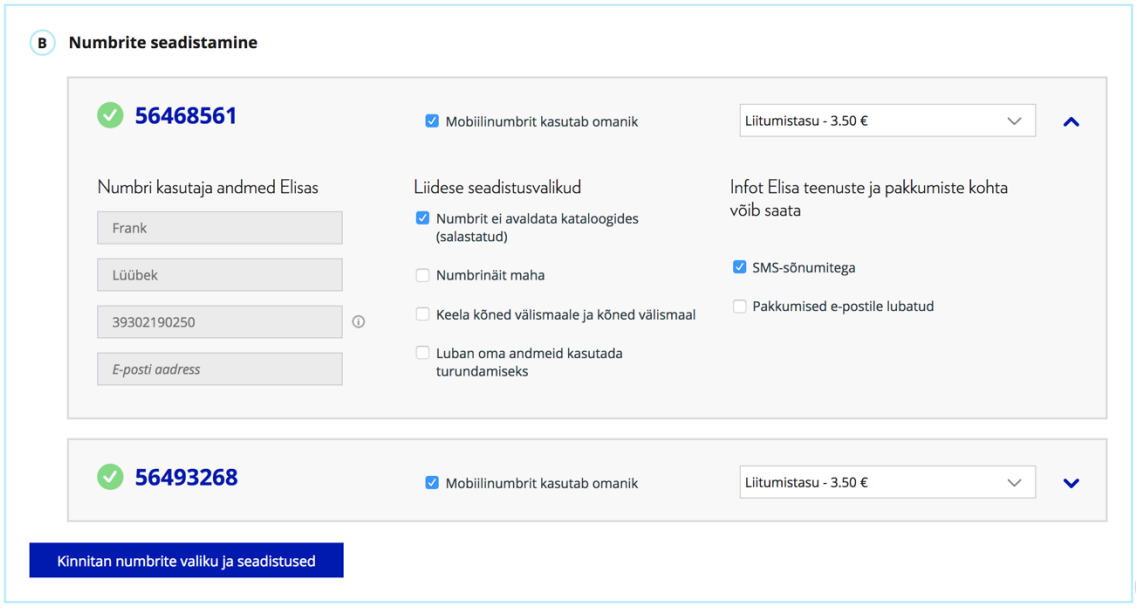
Numbrivaliku komponendi eesmärk on pakkuda kasutajatele võimalust valida endale meelepärased mobiilinumbrid. Vabalt valitavate mobiilinumbrite ploki alla kuvatakse kõik hetkel andmebaasis olevad vabad mobiilinumbrid, kuid kõiki ei näidata korraga välja. Selle asemel saab kasutaja vastavalt tahtele laadida lehele uusi numbreid nupuvajutuse abil. Klikkides mobiilinumbrile liigutatakse see lehekülje paremas osas asuvasse valitud numbrite plokki ja taustal toimub numbri kasutajale broneerimine. Broneeritud numbrid ilmuvad vabade numbrite kohale eraldi plokki. Selleks, et teha sobiva numbri leidmine lihtsamaks, on loodud ka numbrite otsimise komponent, mis filtreerib vabade numbrite hulgast välja kasutaja sisestatud mustrile vastavad numbrid. Lehekülje paremas servas valitud numbrite ploki all saab ka valida numbrite aktiveerimise hetke. Hetkeseisuga on see valik nähtav ainult teenindajale, ning kui klient ise liitumist teeb, siis valitakse vaikimisi esimesel võimalusel aktiveerimine. Kui on valitud vähemalt üks number, siis on võimalik edasi liikuda numbrite seadistamise sammu.

Numbrivaliku samm tervikuna koosneb mitmest AngularJS komponendist. Vastavalt Elisa arhitektuurinõuetele peavad komponendid olema võimalikult taaskasutatavad, ning selleks on kasulik jagada üldine funktsionaalsus mitmeks spetsiaalse otstarbega

komponendiks. Järgnevad on välja toodud numbrivaliku sammu raames arendatud eri komponentide loetelu.

- Üldine numbrivaliku komponent
- Mobiilinumbrite otsingu komponent
- Vabade mobiilinumbrite loetelu komponent
- Valitud mobiilinumbrite komponent
- Broneeritud numbrite komponent
- Numbrite aktiveerimise valiku komponent

3.4.2 Numbrite seadistamise komponent



Joonis 4. Numbrite seadistamise komponent.

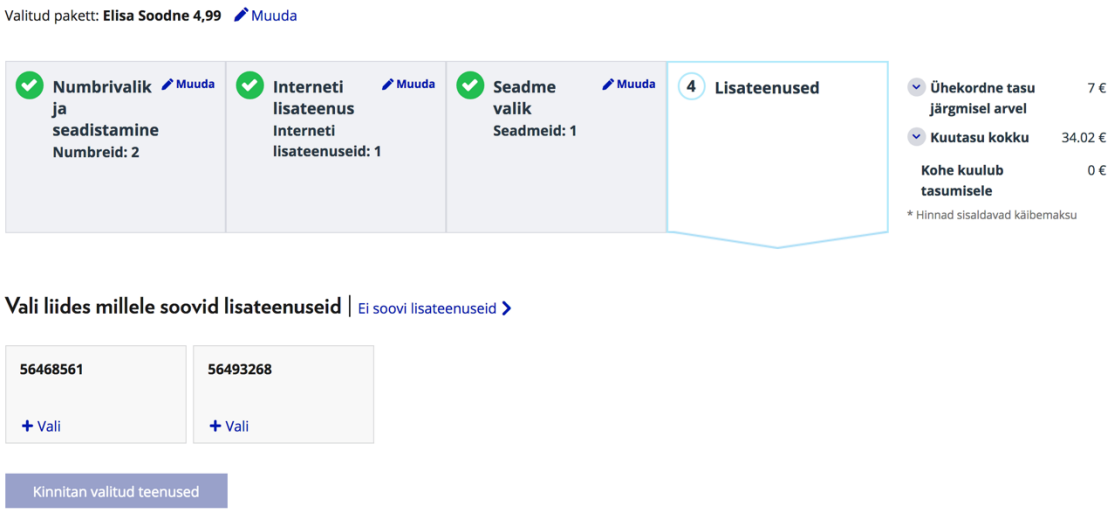
Numbrite seadistamise sammu nimetus kirjeldab väga täpselt ära tema eesmärgi. Kuna Elisas salvestatakse juba midagi tellinud kasutajate kontaktinfo baasi, siis saab tellimise mugavamaks muutmise seisukohast seda mitmel pool ära kasutada. Näiteks siinsamas sammus, kui on valitud seade “mobiilinumbrit kasutab omanik” (mis on ka vaikimisi valitud), siis taustal päritakse baasist kohale omaniku baaskontaktandmed ja täidetakse

nendega vajalikud lahtrid. Lisaks saab seadistada mitmeid mobiilinumbritele omaseid võrguparameetreid ja turundusega seotud sätteid. Liitumistasu valik on nähtav ainult teenindajale, kes valib sobiva tasu vastavalt protseduurile. Kliendi enda liitumise korral liitumistasu ei küsita.

Lisaks tuleb arendamisel mõelda ka programmikoodi optimaalsuse peale. Praegusel juhul on vajalik rakendada kõigile mobiilinumbritele teatud vaikimisi sätteid. Kuna kõik valitud mobiilinumbrid kuuluvad samasse kategooriasse, siis need sätted on kõigil samasugused. Kõiki numbreid on võimalik eraldi seadistada, seega on mõistlik luua selleks eraldi seadistamise komponent. Probleem seisneb selles, et numbreid on võimalik liitumise käigus valida kuni kakskümmend. Optimaalsuse seisukohast oleks täiesti põhjendamatu, kui igaüks neist teeks samasisulise ja sama tulemusega REST-päringu vaikimisi sätete küsimiseks, seega tuleb seda ka komponentide loomisel arvesse võtta. Sama lugu on ka liitumistasude ja omaniku kontaktinfo küsimisega. Järgnevalt on välja toodud numbrite seadistamise sammu raames loodud komponendid.

- Üldine numbrite seadistamise komponent, mis varustab alamkomponente ühiste andmetega
- Individuaalse numbri seadistamise komponent

3.4.3 Liideste eristamise komponent

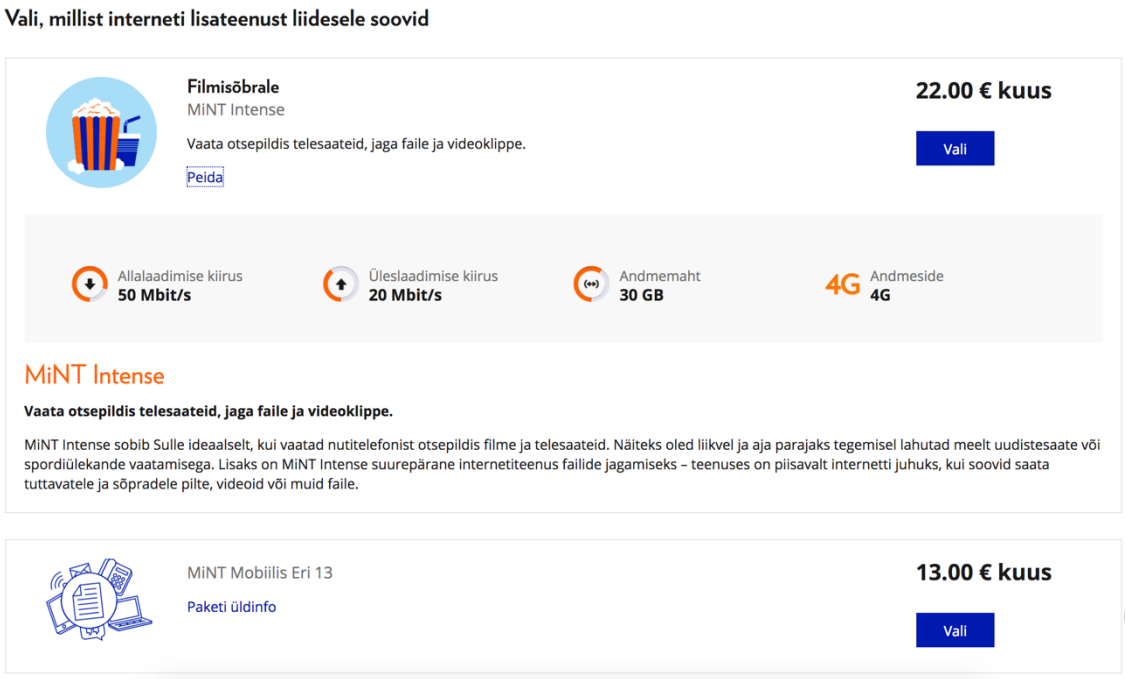


Joonis 5. Liideste eristamise komponent

Liideste eristamise komponent on ainus töö osa, mis ei ole eraldiseisev voo samm – ta on tööriist, mida kasutakse numbrite seadistamise sammu järgsetes sammudes. Liideste eristamise komponent on vajalik, et voo sammudes, mis tulevad pärast numbri seadistamist oleks võimalik kõigi liidestega eraldiseisvalt tegelda. Ainult ühe valitud liidese korral on see komponent peidetud. Eristamise komponendi teiseks ülesandeks on näidata igas möödunud sammus tehtud valikuid voo progressiribal. Oluline nüanss on see, et kuna sama liitumise raamistikku kasutatakse ka paljude teiste toodete puhul, peab vaatama, et eristamise komponent rakenduks ainult kõneliitumise voo puhul ja ainult numbrite seadistamisele järgnevates sammudes. Järgnevalt on välja toodud eristamise komponendi raames loodud komponendid.

- Liideste eristamise komponent.
- Progressiribal lisainfo kuvamise komponent

3.4.4 Interneti lisateenuse komponent

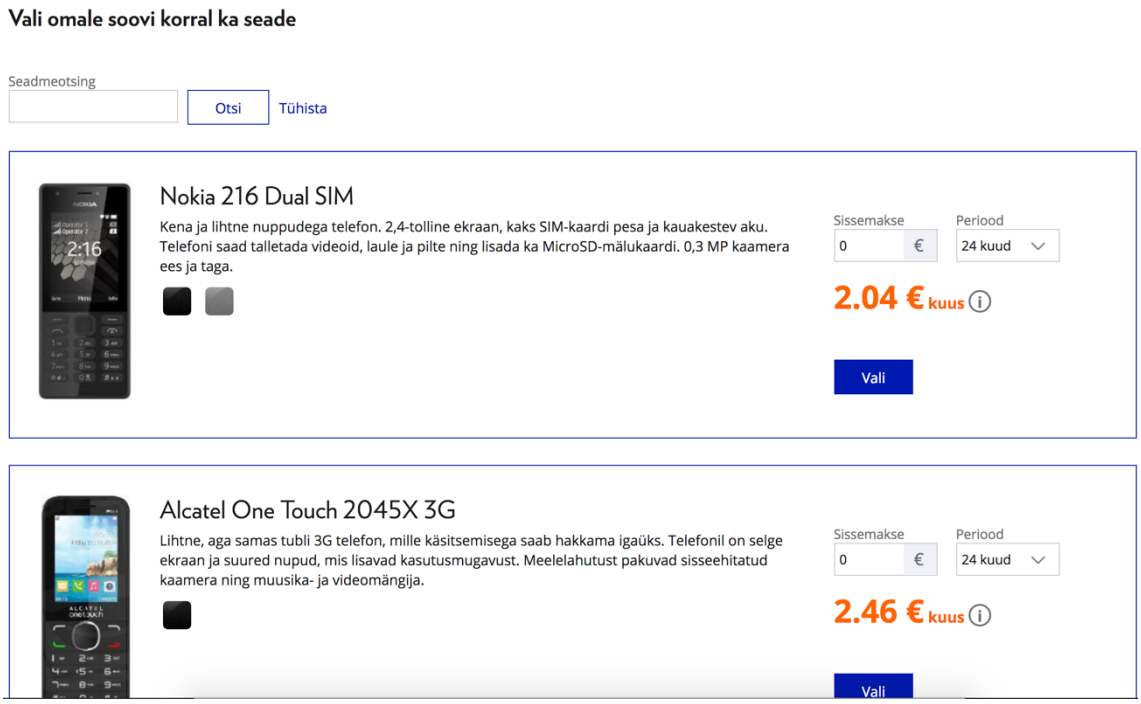


Joonis 6. Interneti lisateenuse komponent.

Interneti lisateenuse sammus saab valida liidesele soovi korral interneti lisapaketi. Kõik internetipaketid küsitakse andmebaasist ja valitud liidesele sobivad filtreeritakse välja *back-endis*. Pakette on võimalik võrrelda nende parameetrite põhjal. Järgnevalt on välja toodud interneti lisateenuse sammu raames loodud komponendid.

- Interneti lisateenuste nimekirja komponent
- Üksiku lisateenuse komponent

3.4.5 Seadmevaliku komponent

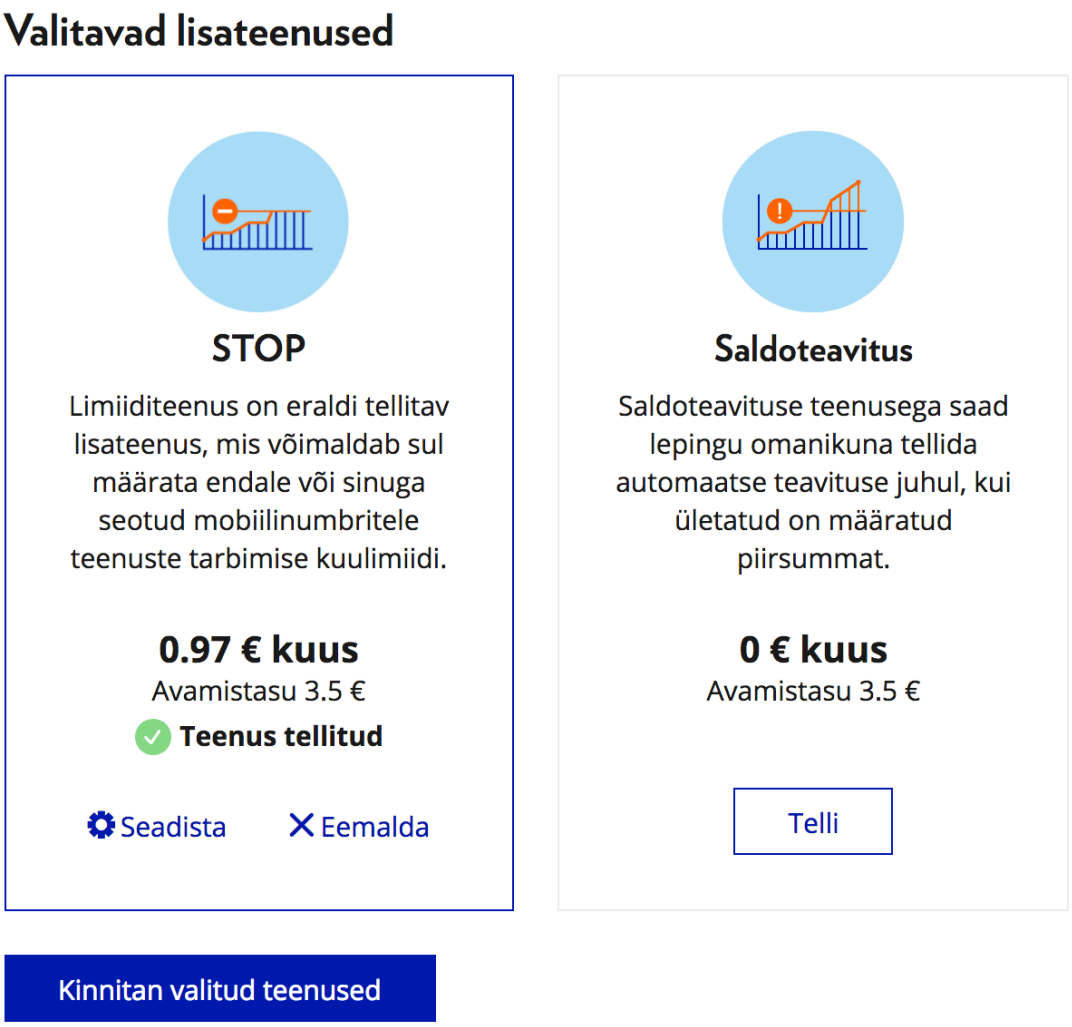


Joonis 7. Seadmevaliku komponent.

Kuna seadmevaliku komponent oli juba tehtud teiste toodetega liitumiste voogude raames ja Elisa üks põhimõtteid on võimalikult palju tehtud tööd taaskasutada, siis täiesti uut komponenti kõneliitumise seadmevaliku sammu raames tegema ei hakatud – eesmärk oli kohandada juba olemasolev komponent sobivaks ka kõneliitumise voole. Taaskasutamine on ühest küljest kasulik ja praktiline, kuid teisest küljest tuleb olla muutuste tegemisel äärmiselt ettevaatlik, et tööle jääks ka teised kohad, kus seda komponenti kasutatud on.

Seadmed küsitakse jällegi välja baasist ja sobivad filtreeritakse välja *back-endis*. Võimalik on ka seadmeid loetelust otsida. Hetkeseisuga puudub kõneliitumise voos võimalus seadet välja osta, kuid võimalik on osta seade osamaksetega ja seejuures saab valida vastavalt oma krediidihinnangule sissemakset ja maksete perioodi. Valida saab ka seadme värvi. Seadme valiku samm koosneb ühest komponendist.

3.4.6 Lisateenuste komponent

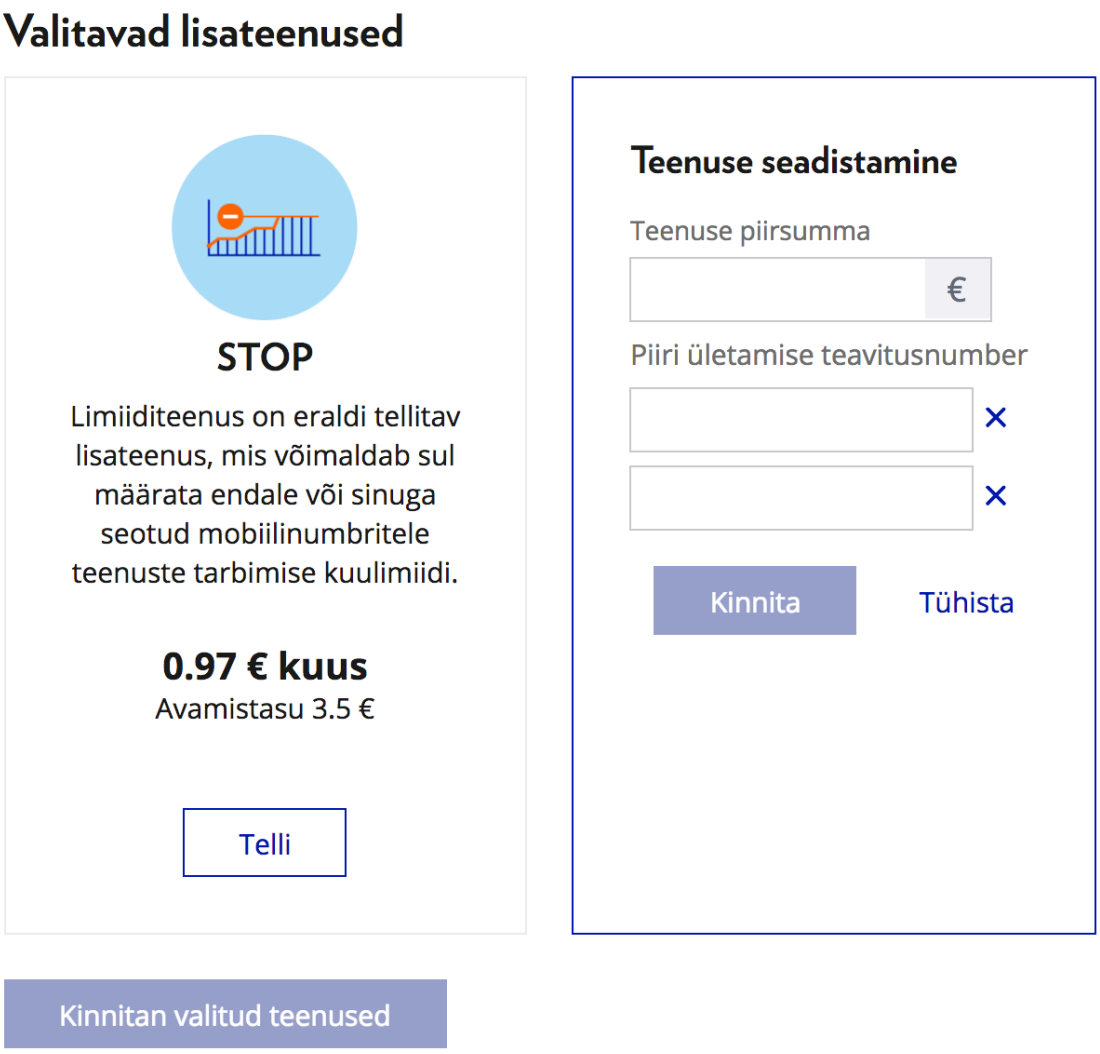


Joonis 8. Lisateenuste komponent.

Lisateenuste komponendis(joonis nr 8) on hetkeseisuga kaks valitavat lisateenust, kuid komponent on ehitatud nii, et uute teenuste lisamine, mida tulevikus ka tõenäoliselt tehakse, oleks lihtne. Lisateenust valides tuleb see ka vastavalt teenusele seadistada (joonis nr 9). Kuna iga lisateenuse seadistamise leht võib olla erinev, siis on selliste jaoks vaja eraldi teenuse- ja seadistamise komponenti. Samas on vaja ka eraldi

komponenti teenustele, millel ole erinevusi. Ka selles sammus küsitakse lisateenuste nimekiri baasist ja sobivad filtreeritakse välja *back-endis*. Järgnevalt on välja toodud lisateenuste sammu raames arendatud komponendid.

- Üldine lisateenuste komponent
- Lisateenuste nimekirja komponent
- Vaikimisi lisateenuse esilehe komponent
- Vaikimisi lisateenuse seadistamise komponent
- STOP lisateenuse esilehe komponent
- STOP lisateenuse seadistamise komponent
- Saldoteavituse lisateenuse esilehe komponent
- Saldoteavituse lisateenuse seadistamise komponent



Joonis 9. Lisateenuste seadistamine.

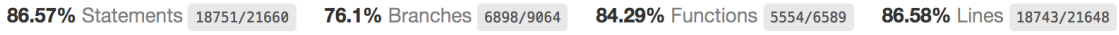
3.5 Testimine

Tarkvara testimise võib väga laias laastus jagada kaheks: manuaaltestimine ja automaattestimine. Käesolevas töös on autor keskendunud vastavalt ettevõtte nõuetele automaattestimise ühele alaliigile – ühiktestimisele. Ühiktestimine on tarkvara testimise osa, mille käigus testitakse tarkvara komponente ehk ühikuid eraldiseisvalt [20]. Ühik on süsteemi väikseim eraldi testitav osa, milleks on enamasti funktsioonid või meetodid. Ühiktestimine on kasulik väga mitmel tasemel – see teeb koodi muutmise lihtsamaks, arendusprotsess muutub kiiremaks, kood on taaskasutatavam, kuna ühiktestimise hõlbustamiseks on vaja kirjutada modulaarset koodi, ning defektide parandamine

ühiktestimise käigus on palju vähem kulukam võrreldes sellega, kui viga oleks tarkvara hilisemas elutsüklis välja tulnud [20].

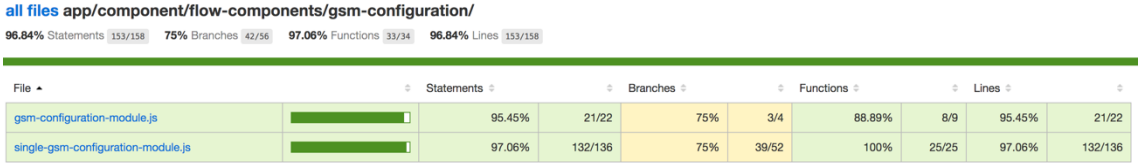
Testimisel peab silmas pidama kehtestatud programmikoodi testidega katvuse nõuet – uue kirjutatud programmikoodi testidega katvuse protsent peab olema vähemalt 70. Nõue kehtib projekti kui terviku kohta.

3.5.1 Front-end ühiktestid



Joonis 10. Front-end projekti üldine testidega katvus.

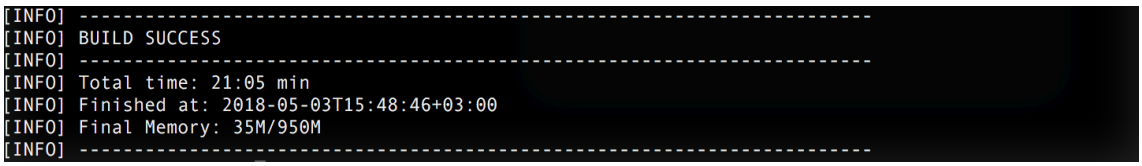
Front-end ühiktestid on kirjutatud kasutades Javascripti testide jooksutamise tööriista Karmat [12] ja Javascripti testimise raamistikku Jasmine'i [13]. Karma muudab testide jooksutamise mugavaks – teste on võimalik käivitada korraga või siis valikuliselt ja seda läbi käsurea või siis otse IDE'st (integreeritud programmeerimiskeskkond), milleks autor kasutab JetBrains'i poolt arendatud IntelliJ IDEA tarkvara. Eelnevalt sai mainitud, et Karma abil on võimalik teste jooksutada kõigis veebibrauserites eraldi, aga erinevate veebibrauserite käitumise testimiseks kirjutatakse ettevõttes Selenium teste, mille kirjutamisega tegelevad täiskohaga testijad. Ühikteste jooksutavad arendajad PhantomJS abil, mis on sisuliselt veebibrauser ilma graafilise kasutajaliideseta [21]. Joonisel nr 10 on näha front-end projekti üldine testidega katvus, ning joonisel nr 11 on ära toodud numbri seadistamise komponendi testidega katvus. Üks Karma ülesannetest on pärast testide jooksutamist koostada tulemustest ülevaade HTML faili kujul(mis kajastub ka joonistel 10 ja 11), mida saab mugavalt verbibrauseriga vaadata.



Joonis 11. Numbrite seadistamise komponentide testidega katvus.

3.5.2 Back-end ühiktestid

Back-end ühiktestid on kirjutatud kasutades Java testimise raamistikke Mockitot [4] ja JUnitit [5]. Mockito kasutamine on vajalik, kuna *back-end* poolel on rakenduste ülesehitus keerukam – kihilise arhitektuuri mustri järgimine tähendab, et klassidel on palju väliseid sõltuvusi, mida on testimisel vaja hallata. Mockito abil saab lihtsa vaevaga testida meetodeid ja juhtida nende väliste sõltuvuste käitumist. Teste jooksutatakse kas integreeritud programmeerimiskeskkonnas või siis käsurealt Maveni [2] abil. Mõlemal juhul saab ülevaate õnnestunud ja ebaõnnestunud testidest, kuid koodi katvuse nõuet saab vaadata Sonarist pärast koodi GIT repositooriumisse laadimist. Joonisel nr 12 on näha käsurea testide jooksutamise väljundit.



Joonis 12. Back-end ühiktestide tulemus käsurealt jooksutamisel.

4 Kokkuvõte

Käesoleva töö eesmärk oli saada valmis töötav ja esmastele nõuetele vastav versioon kõneliitumise voost. See eesmärk sai täidetud kuna see funktsionaalsus on hetkeseisuga valmis teenindajatele kasutamiseks. Samuti on loodud kõik eeldused tulevikus voole täiendavate funktsionaalsuste lisamiseks.

Töö teine eesmärk oli kirjutada koodi, mis vastav Elisa-sisestele parimatele praktikatele ja nõuetele. Arendatud komponendid loodi sellised, et neid oleks võimalik ka mujal programmis taaskasutada. Kirjutatud kood vastab koodianalüsaatorprogrammi Sonari nõuetele, ning kõigi rakenduste testidega katvus on lõputöö kirjutamise hetkel suurem kui 70%.

Samuti tehti töö praktilise osa raames olulisi ja mahukaid täiendusi tellimuse kinnitamise ja teenuste loomise protsessidele, kuid kuna need kaks protsessi on sellised, mida läbivad kõik ITBst tellitavad tooted ja kõik tiimid puutuvad nende arendustega kokku, siis on protsesside raames selgeid autorluse piire veelgi raskem defineerida. Seega otsustas autor selle arenduse osa lõputöö skoobist välja jätta.

Kuna lõputöö kirjutamise seisuga jõudis kõneliitumine kogu oma funktsionaalsusega *live'*i ja eelnevalt said täidetud kõik selleks vajalikud nõuded, siis võib lugeda kõik tööle seatud eesmärgid täidetuks.

Kasutatud kirjandus

[1] “What is Java?”, [Võrgumaterjal]. Available: https://www.java.com/en/download/faq/whatis_java.xml. Kasutatud (10.04.2018)

[2] „What is Maven?“, [Võrgumaterjal]. Available: https://maven.apache.org/what-is-maven.html. Kasutatud (10.04.2018)

[3] “Google Guice Github”, [Võrgumaterjal]. Available: https://github.com/google/guice. Kasutatud (10.04.2018)

[4] “Mockito framework site,” [Võrgumaterjal]. Available: http://site.mockito.org/. Kasutatud (10.04.2018)

[5] “JUnit Github”, [Võrgumaterjal]. Available: https://github.com/junit-team/junit4/wiki. Kasutatud (10.04.2018)

[6] “Eclipse Jetty”, [Võrgumaterjal]. Available: https://www.eclipse.org/jetty/about.html. Kasutatud (10.04.2018)

[7] “AngularJS”, [Võrgumaterjal]. Available: https://angularjs.org/. Kasutatud (10.04.2018)

[8] “Difference between AngularJS and Angular 2”, [Võrgumaterjal]. Available:https://www.codeschool.com/discuss/t/difference-between-angularjs-vs-angular-2/29820. Kasutatud (10.04.2018)

[9] “HTML 5”, [Võrgumaterjal]. Available: https://w3c.github.io/html/. Kasutatud (10.04.2018)

[10] “CSS Introduction”, [Võrgumaterjal]. Available: https://www.w3schools.com/css/css_intro.asp. Kasutatud (10.04.2018)

[11] “What is NPM?”, [Võrgumaterjal]. Available: https://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm/. Kasutatud (10.04.2018)

[12] “Karma – how it works?”, [Võrgumaterjal]. Available: http://karma-runner.github.io/2.0/intro/how-it-works.html. Kasutatud (10.04.2018)

[13] “Jasmine documentation”, [Võrgumaterjal]. Available: https://jasmine.github.io/. Kasutatud (10.04.2018)

[14] “Front-end vs. Back-end”, [Võrgumaterjal]. Available: https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html. Kasutatud (20.04.2018)

[15] “Front-end vs Back-end development”, [Võgumaterjal]. Available: http://www.reusserdesign.com/blog/front-end-vs-back-end-development/. Kasutatud (20.04.2018)

[16] “What is AngularJS”, [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/introduction. Kasutatud (20.04.2018)

[17] “AngularJS databinding”, [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/databinding. Kasutatud (20.04.2018)

[18] “Layered architecture”, [Võrgumaterjal]. Available: https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html. Kasutatud (20.04.2018)

[19] “RESTful API”, [Võrgumaterjal]. Available: https://www.mulesoft.com/resources/api/restful-api. Kasutatud (20.04.2018)

[20] “Unit testing”, [Võrgumaterjal]. Available: http://softwaretestingfundamentals.com/unit-testing/. Kasutatud (03.05.2018)

[21] “What is PhantomJS?”, [Võrgumaterjal]. Available: https://scotch.io/tutorials/what-is-phantomjs-and-how-is-it-used. Kasutatud (03.05.2018)

[22] “What is Sonar”, [Võrgumaterjal]. Available: https://www.sonarqube.org/. Kasutatud (03.05.2018)

[23] “Business rules must be enforced by the api”, [Võrgumaterjal]. Available: https://designingforscale.com/business-rules-must-be-enforced-by-the-api/. Kasutatud (03.05.2018)

[24] “Continuous integration”, [Võrgumaterjal]. Available: https://www.thoughtworks.com/continuous-integration. Kasutatud (07.05.2018)

[25] “Window.sessionStorage”, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage. Kasutatud (15.05.2018)

[26] “What is Git?”, [Võrgumaterjal]. Available: https://www.atlassian.com/git/tutorials/what-is-git. Kasutatud (15.05.2018)

[27] “Jenkins CI server explained”, [Võrgumaterjal]. Available: https://www.infoworld.com/article/3239666/devops/what-is-jenkins-the-ci-server-explained.html. Kasutatud (15.05.2018)

Lisa 1 – Üldise lisateenuste komponendi test

```
it('should not allow same service to be inserted multiple times when the
service is being configured', function() {
  var chosenProduct = {
    companyProduct: false,
    consumerProduct: false,
    id: 60,
    level: "Lisa",
    mutualExclusionGroups: [],
    name: "Lisateenus",
    price: {
      monthlyPrice: 0
    },
    productGroup: "Tootegrupp"
  };

  controller.confirmAdditionalService(chosenProduct, productConf);
  productConf.limit = 999;
  productConf.notificationNumber = '9876';
  controller.confirmAdditionalService(chosenProduct, productConf);

  expect(contextCart.data.rows[0].data.childRows.length).toEqual(1);

});
```

Lisa 2 – Numbrite seadistamise sammu vaikimisi tootekonfiguratsiooni test

```
@Test
public void
getDefaultProductConfiguration_shouldAssignNoFieldsWhenParamsNotPresent() {
   Integer productId = 1523;
   doReturn(new HashMap<String, String>()).when(productParamService)
         .getProductParamsForAccountCreator(productId);
   doReturn(Boolean.FALSE).when(productPropertyService)
         .hasProperty(productId, PPROP_DENY_FOREIGN_CALLS_BY_DEFAULT);

   Map<String, Boolean> result =
productService.getDefaultProductConfiguration(productId);

   assertEquals(Boolean.FALSE,
         result.get(ProductService.CONFIGURATION_FOREIGN_CALLS_DISABLED));
   assertEquals(Boolean.FALSE,
         result.get(ProductService.CONFIGURATION_DENY_FOREIGN_CALLS));
   assertEquals(Boolean.FALSE,
         result.get(ProductService.CONFIGURATION_SECRET_NUMBER));
   assertNull(result
         .get(ProductService.CONFIGURATION_DATA_BARRING_VISIBLE));

}
```