TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Raul Ruukel 083861IAPB

AS SMART KINDLUSTUSMAAKLER UUE VEEBIRAKENDUSE ARHITEKTUUR JA PROTOTÜÜBI ARENDAMINE

bakalaureusetöö

Juhendaja: Ago Luberg
Magistrikraad

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: Raul Ruukel

23.05.2018

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on pakkuda välja uus arhitektuur ning arendada selle põhjal välja ühe mooduli funktsionaalsus AS Smart Kindlustusmaakleri veebirakenduse jaoks, mis võimaldaks järk-järgult asendada olemasoleva rakenduse. Sellega peab kaasnema ka tarneprotsess muudatuste automaatseks serverisse paigaldamiseks.

Eesmärkide saavutamiseks on töös vaadeldud olemasolevat rakendust, analüüsitud erinevaid võimalusi arhitektuuri valiku juures ning kirjeldatud tehtud otsuseid. Selle tulemusena on loodud uus rakendus koos tarneprotsessiga ning implementeeritud selles klientide mooduli funktsionaalsus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 6 peatükki, 10 joonist.

Abstract

Architecture and prototype development of the new web application for AS Smart Kindlustusmaakler

Web applications are becoming increasingly important for more and more companies to support and enable their business practices. AS Smart Kindlustusmaakler is no different in this regard, but their current application has multiple problems and it's becoming difficult to find people that would develop the application further. For these reasons they have decided to replace their existing application with a new one.

The goal of this thesis is to choose a new architecture for the new application and develop the functionality of one module in this application. The solution should allow to move gradually, module-by-module to the new architecture. An important part of this is to also provide a deployment process that would enable automatic deployment of changes to the server.

To achieve these goals the author looks at the old application, analyses the different possibilities when choosing the new architecture and describes the decisions made. As a result of these decisions a prototype for the new application is developed along with the deployment process. The functionality of the clients module is also implemented in the new application.

The thesis is in Estonian and contains 41 pages of text, 6 chapters, 10 figures.

Lühendite ja mõistete sõnastik

Apache	Avatud lähtekoodiga veebiserver
API	*Application programming interface*, rakendusliides
Bash	*Bourne Again SHell,* käsureaprotsessor UNIX-'ile
HTML	*Hypertext Markup Language*, hüpertekst-märgistuskeel
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
Javascript	Skriptimiskeel
JSON	*JavaScript Object Notation*, andmevahetus formaat
MVC	*Model-view-controller*, Mudel-vaade-kontroller
MySQL	Avatud lähtekoodiga relatsiooniline andmebaasisüsteem
PHP	Avatud lähtekoodiga skriptimiskeel
SPA	*Single page application*, üheleherakendus
SSH	*Secure socket shell,* protokoll, mis võimaldab turvalist sisselogimist kaugarvutisse
Tomcat	Avatud lähtekoodiga veebiserver Java rakenduste jaoks

Sisukord

Jooniste loetelu

1 Sissejuhatus

AS Smart Kindlustusmaakler on ettevõte, mis tegeleb peamiselt kasko- ja liikluskindlustuse vahendamisega klientidele. Nende igapäevatöös on kriitilise tähtsusega veebirakendus, mis neid protsesse toetab ja võimaldab. Olemasolev rakendus koosneb avalikust veebilehest, kust kliendid saavad pakkumisi küsida, ning sisemiseks kasutuseks mõeldud administreerimisliidesest, mis võimaldab kliente, pakkumisi, poliise, arveid jms hallata.

Olemasolev rakendus on välja kujunenud aastate jooksul erinevate programmeerijate poolt arendatuna ning puudub täpsem dokumentatsioon, mis rakenduse funktsionaalsust kirjeldaks. Samuti puuduvad automatiseeritud testid, mis funktsionaalsust kataks, ning täpsem protsess rakenduse serverisse paigaldamiseks ja muudatuste tarnimiseks. Sellest tulenevalt on muutunud keerulisemaks ka rakenduse haldamine, muudatuste tegemine ning selleks sobilike inimeste leidmine. Nende probleemide lahendamiseks ongi otsustatud võtta ette rakenduse puhtamalt ümber kirjutamine populaarsemate tehnoloogiate peale.

Käesoleva bakalaureusetöö eesmärgiks on pakkuda välja sobivad tehnoloogiad ja arhitektuur uue rakenduse jaoks ning arendada välja ühe mooduli funktsionaalsus, et demonstreerida nende valikute sobivust. Kuna kogu rakenduse uuesti kirjutamine on pikk ja aeganõudev protsess, siis peavad uus ja vana rakendus selleks ajaks kõrvuti tööle jääma. Samuti kuulub bakalaureusetöö skoopi tarneprotsessi paika panemine, nii et muudatuste serverisse saatmine ja testide käivitamine oleks automatiseeritud.

Teises peatükis antakse lühike ülevaade olemasolevast rakendusest. Kolmandas peatükis analüüsitakse arhitektuuri valikut ning kirjeldatakse kasutatud tehnoloogiaid. Neljas peatükk peatub teemadel, mis uue rakenduse arenduse ja tarneprotsessiga kaasnesid. Viimaks vaadeldakse viiendas peatükis tehtud töö tulemust ning jagatakse soovitusi edaspidiseks rakenduse arendamiseks.

2 Olemasolev rakendus

Olemasolev rakendus on saanud alguse mitmeid aastaid tagasi ning on arenenud järk-järgult vastavalt vajadustele. Rakendus on kirjutatud PHP-s ning on aja jooksul viidud üle ka ühelt raamistikult teisele. Tänasel päeval põhineb rakendus CodeIgniter[1] raamistikul, kuid kasutab sellest juba üpris vana versiooni, mis lasti välja aastal 2012.

Serveris töötab rakendus Apache veebiserveri peal ning andmebaasisüsteemina on kasutusel MySQL. Lisaks põhiserverile on olemas ka testserver, milles saab muudatusi enne põhiserverisse panemist katsetada. Testserver on oluline ka käesoleva töö jaoks, sest võimaldab seal kohe tarneprotsessi nii üles seada, et rakenduse viimane seis oleks pidevalt näha. See annab ka kliendile parema ülevaate töö käigust ning võimaluse jagada kiiremini tagasisidet, kui midagi vajab muutmist või parandamist.

Jättes välja raamistiku enda ning muud välised teegid, on vana rakenduse programmikood jaotatud 350 PHP faili vahel, mis sisaldavad kokku umbes 70 000 rida. Lisaks avalikule osale, kus kliendid saavad kindlustuspakkumisi küsida, on rakenduses ka oma töötajatele ning koostööpartneritele suunatud pool, mis on turvatud kasutajanime ja parooliga. Peamiste moodulitena võib selles osas eristada klientide, pakkumiste, poliiside ning arvete moodulit. Samuti on administraatori õigustega kasutajal ligipääs administreerimismoodulile, mis võimaldab seadistada paljusid erinevaid aspekte, mis rakenduse tööd mõjutavad.

Kuna rakendus on võrdlemisi mahukas ning kogu funktsionaalsuse uuel tehnoloogial implementeerimine on pikk protsess, siis võetakse käesoleva töö raames käsile vaid üks moodul, mille põhjal uue rakenduse prototüüpi arendada. Selleks mooduliks valiti klientide moodul, mis kasutab küll andmeid ka arvete, poliiside ning pakkumiste juurest, võimaldades nii põgusalt tutvuda ka nende moodulitega, kuid on samas piisavalt eraldiseisev, et seda saaks sobiva tükina uues rakenduses arendada.

[1] https://www.codeigniter.com/

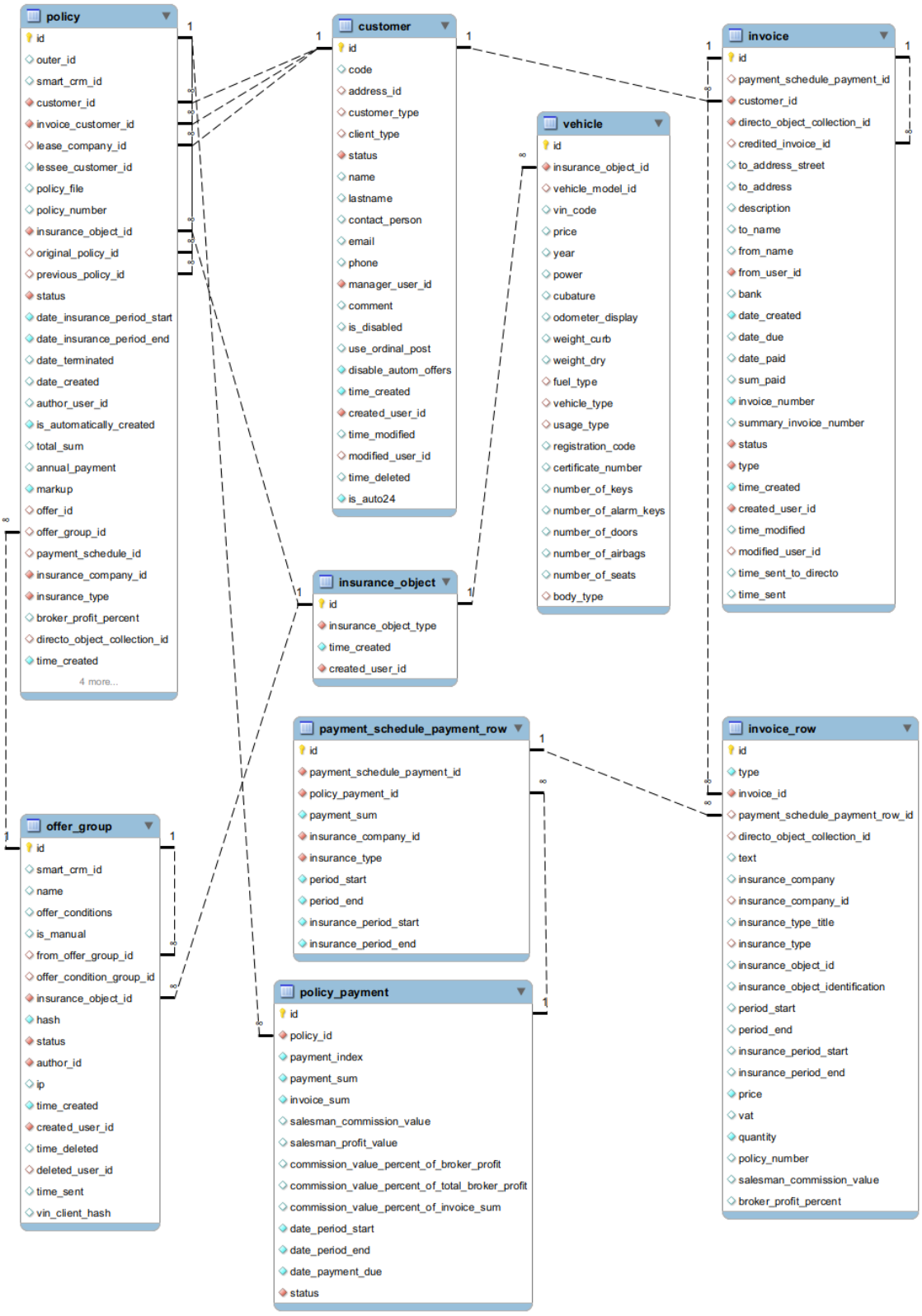
2.1 Andmemudel

Andmebaasis on andmed jagatud 146 tabeli vahel. Põhiliste olemitena saab eristada kliente, pakkumisi, poliise, arveid ning kindlustusobjekte ehk sõidukeid. Olulisemate klientide moodulis kasutatud olemite andmebaasiskeemi kirjeldab Joonis 1.

Klientide andmed on dubleeritud kahes tabelis. Esmase pakkumise koostamisel sisestatakse andmed `client` tabelisse ning pakkumisest poliisi ja arve vormistamisel kopeeritakse andmed `customer` tabelisse. Klientide moodul kasutab andmeid just `customer` tabelist.

Kuigi hetkel saab rakenduses hallata vaid sõidukitega seotud kindlustusi, siis andmebaasis on struktuur paindlikum ning võimaldab salvestada ka muud liiki kindlustusobjektide kohta käivaid kindlustusi. Selleks on andmebaasis kasutusel `insurance_object` tabel, milles saab määrata `insurance_object_type` väljal kindlustusobjekti tüübi. Sellele tabelile viidatakse nii `policy` tabelist kui ka `vehicle` tabelist.

Kuna uus rakendus kasutab sama andmebaasi ning vanas rakenduses ei ole enam suuremaid muudatusi teha plaanitud, siis antud bakalaureusetöö raames andmebaasi struktuuris suuremaid muudatusi ei tehta, vaid modelleeritakse uus rakendus vastavalt olemasolevale andmebaasi struktuurile.



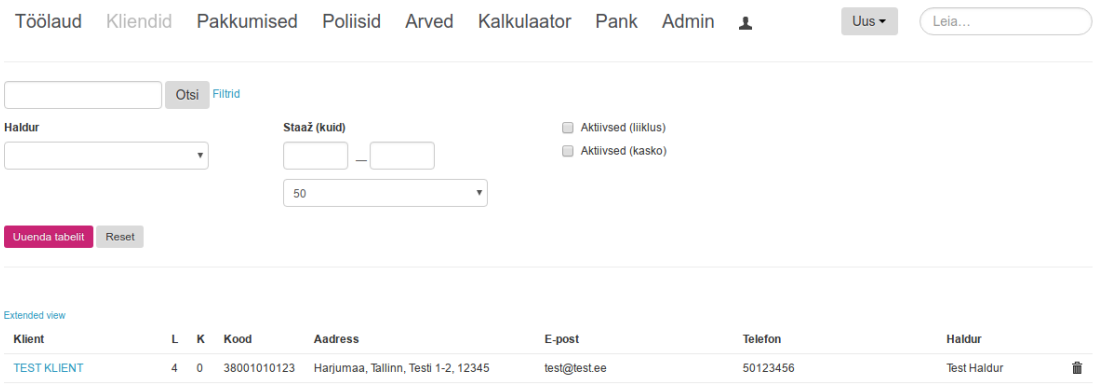
Joonis 1: Peamised olemid klientide moodulis

2.2 Klientide moodul

Klientide moodulis saab eristada 3 erinevat vaadet – nimekiri, detailne vaade ja muutmisvaade.

2.2.1 Klientide nimekiri

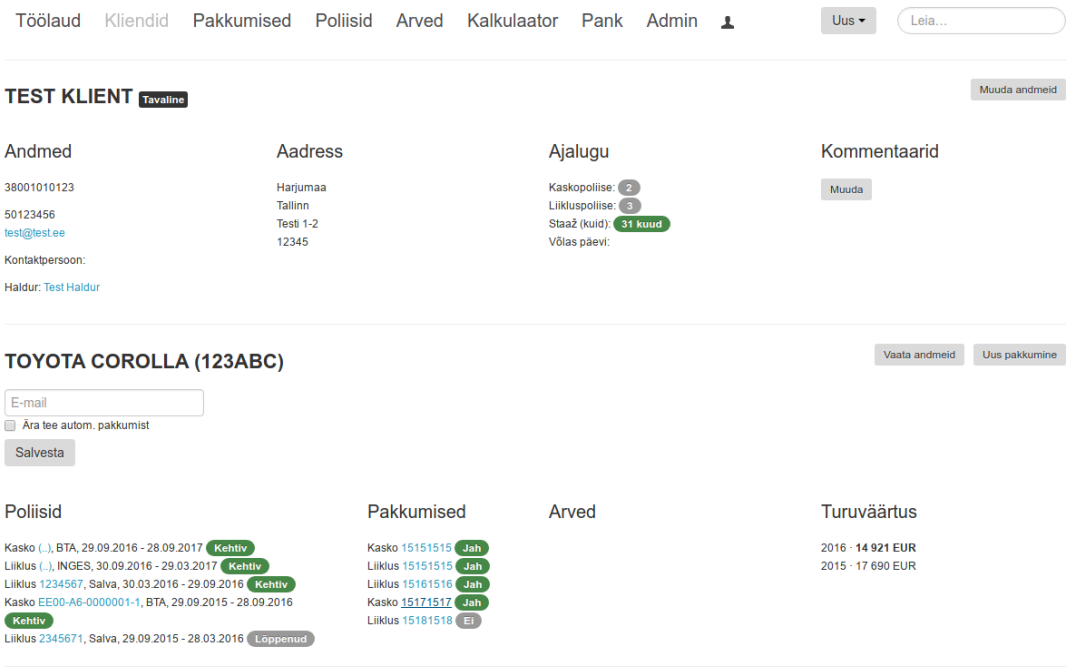
Klientide nimekirjas (Joonis 2) on kuvatud kõik kliendid, kelle staatus pole kustutatud. Samuti on võimalik nimekirja filtreerida halduri, staaži, aktiivsete kasko- või liikluskindlustuspoliiside järgi ning nime või koodi järgi otsides. Nimekiri on jagatud lehekülgedeks 50 kliendi kaupa.



Joonis 2: Klientide nimekirja vaade

2.2.2 Kliendi detailne vaade

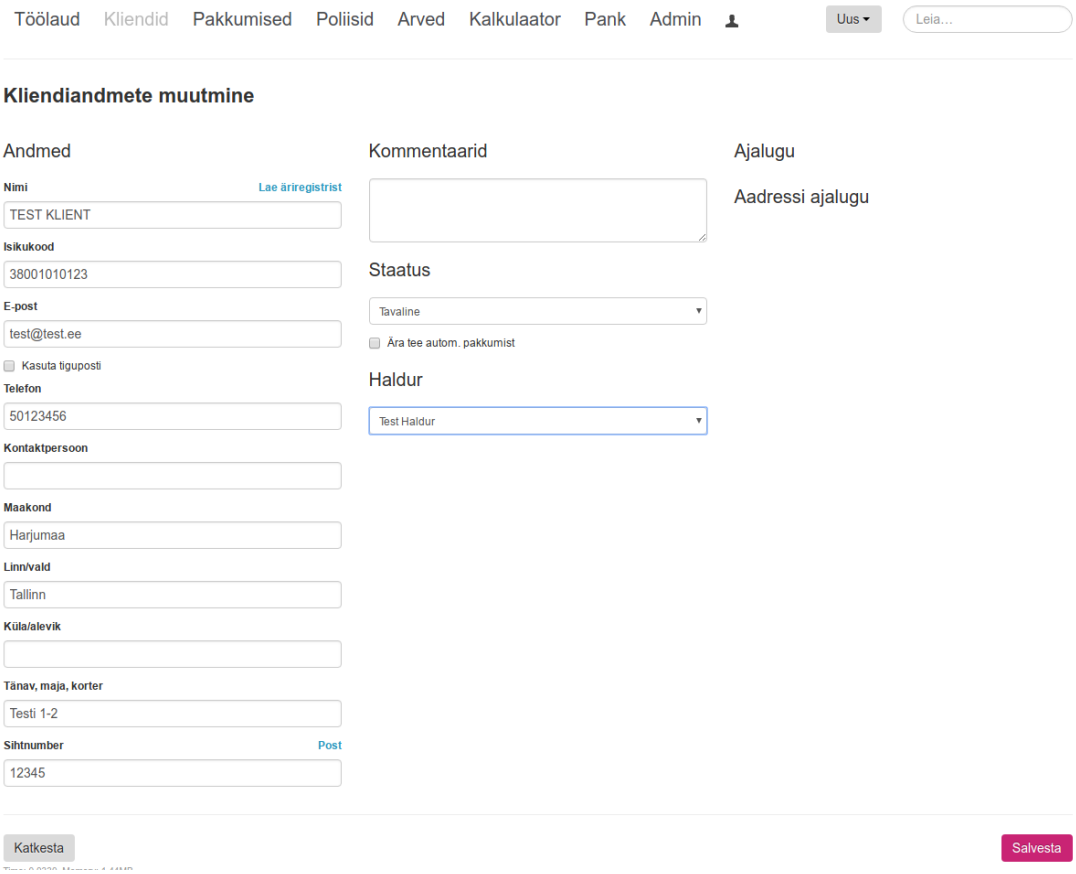
Kliendi detailses vaates (Joonis 3) on kuvatud kõik kliendi andmed. Lisaks on seal välja toodud kõik kliendiga seotud sõidukid ning nende all sõidukiga seotud poliisid, pakkumised, arved ja turuväärtuse muutus aastate lõikes. Kuna iga poliisi kohta on andmebaasis erinev sõiduk (uued read `vehicle` ning `insurance_object` tabelites), siis leitakse sõidukiga seotud poliisid registrinumbri järgi. Pakkumised ja arved leitakse sõiduki alla omakorda poliisi kaudu. Samuti saab kliendi detailses vaates muuta kliendi kommentaari ning seotud sõidukite e-posti aadressi ja automaatsete pakkumiste tegemise tingimust.



Joonis 3: Kliendi detailne vaade

2.2.3 Kliendi muutmisvaade

Kliendi muutmisvaates (Joonis 4) saab määrata kõiki kliendi põhiandmeid ning muuta kliendi staatust ja haldurit. Samuti on siin kuvatud kliendi andmete ja aadresside ajalugu. Ajalugu salvestatakse automaatselt andmebaasis trigeriga, kui muutub nimi, kood, telefon, e-post või mõni aadressi väljadest.



Joonis 4: Kliendi muutmisvaade

3 Uus arhitektuur

Veebirakenduste arendamiseks on palju erinevaid võimalusi ning valikuid, kuidas rakenduse arhitektuuri üles ehitada. Järgnevalt on ära toodud peamised võimalused, mida kaaluti, ning tehtud valikud koos põhjendustega.

3.1 Mikroteenused või monoliitne arhitektuur

Arhitektuuri valikul oli laiem küsimus, kas arendada rakendus mikroteenustena või monoliitse rakendusena.

3.1.1 Mikroteenused

Mikroteenuste arhitektuur on variant teenusorienteeritud arhitektuurist (*service-oriented architecture*), kus kogu süsteem koosneb mitmetest väikestest teenustest, mis töötavad eraldi rakendustena ning suhtlevad omavahel kasutades lihtsaid protokolle (enamasti HTTP protokolli). Iga teenus implementeerib kindlat selgelt piiritletud osa kogu äriloogikast [1] .

3.1.2 Monoliitne arhitektuur

Monoliitse arhitektuuri all peetakse silmas mikroteenustele vastanduvat arhitektuuri, kus kogu süsteem on üles ehitatud ühe rakendusena, ühes koodibaasis, mis töötab serveris ühe protsessina. Sisemiselt võib programmikood olla jagatud selgeteks komponentideks ja kihtideks, kuid lõpptulemusena kompileeritakse see üheks tervikuks [2] .

3.1.3 Võrdlus

Mikroteenustel on monoliitse arhitektuuriga võrreldes mitmeid eeliseid [3] :

- Iga teenus on teistest eraldatud – see võimaldab rakenduse erinevaid komponente sõltumatult arendada, testida ning muudatusi serveris rakendada. Tänu sellele saab rakendust arendavad inimesed jagada paremini väikesteks meeskondadeks, kes tegelevad vaid neile määratud teenustega. Samuti muudab see lihtsamaks teenuse skaleerimise vastavalt koormuse kasvule.

- Iga teenus on suhteliselt väike – tänu sellele on muudatuste tegemine lihtsam ja kiirem, sest teenus käivitub kiiremini ning kogu funktsionaalsus on arendajatele paremini mõistetav.

- Iga teenus võib kasutada täiesti erinevaid tehnoloogiaid vastavalt oma spetsiifilistele vajadustele.

Samas on mikroteenustel ka mitmeid puuduseid [3] :

- Suurem keerukus – kuna kogu rakenduse andmed on jagatud ära erinevate teenuste vahel, siis on oluliselt keerulisem implementeerida kasutusjuhte, mis pärivad või muudavaid andmeid mitmes teenuses. Samuti ei saa selle tõttu tagada andmete konsistentsust andmebaasi tasandil (näiteks seose väljade väliste võtmete abil), vaid seda tuleb lahendada rakenduse tasemel.

- Keerulisem käivitamine – kuna ühe rakenduse asemel tuleb serveris käivitada iga teenus eraldi, siis nõuab see keerulisemat lahendust.

- Suurem ressursinõudlikkus – kuna iga teenus töötab täiesti eraldi protsessina, siis kaasneb sellega ka vastavalt teenuste arvule mitmekordne lisa ressursikulu võrreldes monoliitse arhitektuuriga, kus kogu rakendus töötab ühe protsessina.

3.1.4 Valik

Mikroteenuste eelised tulevad välja eelkõige suuremate süsteemide juures, kus üks monoliitne rakendus muutuks nii suureks, et seda on raske muuta ning skaleerida [4] . Analüüsides olemasolevat rakendust ja tulevikuplaane, ei ole ette näha, et uus rakendus kasvaks nii suureks, et need eelised oleksid antud juhul olulised. Küll aga kaasneks selle valikuga kõik eelnevas peatükis loetletud puudused. Sellest tulenevalt otsustati uus rakendus ehitada monoliitse arhitektuuriga.

3.2 MVC või SPA

Kui monoliitne arhitektuur annab vastuse laiemale küsimusele, kui eraldatud peaksid erinevaid rakenduse komponendid olema, siis ei määra see täpsemalt ära, kuidas peaks neid komponente arendama ning omakorda kihtideks jagama. Kuigi võimalusi selleks on palju, siis on aegade jooksul välja kujunenud mõned kindlad ja populaarsed mustrid, kuidas veebirakendusi üles ehitada.

Antud rakenduse puhul oli kaalumisel kaks varianti – MVC või SPA.

3.2.1 MVC

MVC (*model-view-controller*) on arhitektuuri muster, mis jagab rakenduse kolmeks põhiliseks kihiks – mudel, vaade ja kontroller [5] . Mudelid tegelevad rakenduse andmete pärimise, salvestamise ning seotud äriloogikaga. Vaated tegelevad kasutajaliidese kuvamisega, kasutades selleks enamasti mudeli andmeid. Kontrollerid võtavad vastu kasutaja sisendit, haldavad selle põhjal mudeleid ning tagastavad kasutajale väljundit, milleks on enamasti vaate põhjal genereeritud kasutajaliides.

MVC arhitektuur pakuti välja juba aastal 1979 [6] , aastaid enne tänase veebi kasutusele võttu. Sellegipoolest hakkas selline lähenemine kiirelt populaarsust koguma ka veebirakenduste arendamisel ning on selleks laialdaselt kasutusel tänapäevani. Veebirakenduste puhul tähendab tavapärase MVC kasutamine seda, et nii kogu äriloogika kui ka kasutajaliidese kokku panemine toimub serveris. Kuigi kasutusel võib olla ka Javascripti, mis lehe sisu dünaamiliselt muudab, siis põhiliselt laetakse kasutaja tegevuste tulemusel ikkagi kogu lehe sisu serverist.

3.2.2 SPA

Viimastel aastatel on üha enam populaarsust kogunud SPA (*single page application*) muster veebirakenduste arendamiseks. Sellise lähenemise puhul laetakse esmase pöördumisega serverist kõik vajalik rakenduse kasutajaliidese kuvamiseks (HTML, Javascript, CSS) ning edaspidi vahetatakse kasutaja tegevuste peale serveriga vaid andmeid, mille põhjal muudetakse lehe sisu dünaamiliselt, mitte ei laeta kogu lehte serverist uuesti.

Üldjuhul on selleks kasutatud kasutajaliidese arendamisel mõnda Javascripti raamistikku ning serveris asub teenus, mis pakub kindlat API-t (*application programming interface*) rakenduse andmete pärimiseks ning muutmiseks.

3.2.3 Võrdlus

SPA mustri eelistena võib välja tuua:

- Lehel navigeerimine on kiirem ja serveri koormus väiksem – kuna peale esmast laadimist laetakse serverist vaid andmeid, siis ei pea lehel navigeerides serveris kogu lehte uuesti genereerima ning selle taga ootama.
- Kasutajaliides ning serveripoolne äriloogika on selgelt eraldatud – see võimaldab nende osade arendamist paremini erinevate meeskondade vahel ära jagada. Tuleb vaid kokku leppida API-s, mida need komponendid omavahel suhtlemiseks kasutavad.
- Rakendust on lihtsam teiste süsteemide või kasutajaliidestega (näiteks mobiilirakendus) ühendada – andmete vahetamiseks saab serveris kasutada sama API-t.

Küll aga võib leida ka mõningaid puuduseid:

- Esmane lehe laadimine on aeglasem – kuna esmase laadimisega laetakse kõik vajalik kasutajaliidese jaoks, siis võtab see kauem aega.
- Rakenduse käitumine ei pruugi olla ootuspärane või vajab tavapäraste olukordade lahendamiseks eraldi lahendusi – brauserid on loodud ennekõike toetama tavapärast navigeerimist, kus lehtede vahel liikudes laetakse kogu leht serverist. Seega vajavad SPA puhul eraldi tähelepanu asjad, mis muidu toimivad automaatselt brauseri poolt – edasi-tagasi navigeerimine, lehe laadimise indikaator, lehe laadimise katkestamine, lehelt lahkumise kohta kinnituse küsimine [7] .

- Kogu rakenduse arendamiseks on vaja rohkem teadmisi ja oskusi – kuna kasutajaliides ja serveri pool võivad kasutada täiesti erinevaid tehnoloogiaid, siis peab kogu rakenduse arendamiseks olema kursis nende kõigiga.

3.2.4 Valik

Antud rakenduse puhul osutus valituks tavaline MVC põhine arhitektuur. Peamise põhjusena saab välja tuua just seda, et see võimaldab kogu rakenduse arendada ühe raamistiku peale, mis muudab arendamise lihtsamaks. Kuna puudub suur arendusmeeskond, kes sellega tegeleks, siis on oluline, et üks programmeerija suudaks tegeleda kõikide aspektidega arendusprotsessis. Mida rohkem erinevaid tehnoloogiaid oleks kasutusel, seda keerulisemaks muutuks ka tulevikus rakenduse arendamiseks sobilike inimeste leidmine.

Samuti pole tulevikus näha nii suurt koormuse kasvu, et väike jõudluse vahe oleks antud juhul määrava tähtsusega. Ka sellise API järele, mis täpselt samal kujul andmete pärimist või muutmist võimaldaks, ei paista lähiajal vajadust olevat, kuid selle tekkimisel saab ka MVC arhitektuuri puhul vähese vaevaga selle tekitada. Küll aga muudab serveripoolne lehtede genereerimine lihtsamaks ka sessiooni jagamise perioodil, kui vana ning uus rakendus kõrvuti töötavad.

3.3 Kasutatud tehnoloogiad

Uue rakenduse juures kasutatud tehnoloogiate valikul lähtuti ennekõike sellest, et tegemist oleks võimalikult populaarsete, ent samas ka pikema ajalooga tehnoloogiatega, mis oleks jõudnud juba stabiliseeruda ning ennast tõestada. See muudab probleemidele ja küsimustele vastuste leidmise lihtsamaks. Samuti on tulevikus nii kergem leida uusi inimesi, kes rakenduse arendamisega tegeleda saaks.

3.3.1 Java

Programmeerimiskeelena valiti uue rakenduse arendamiseks Java. Java on objektorienteeritud programmeerimiskeel, mille esimene versioon avaldati 1995. aastal [8] . Java on väga laialt levinud ning on leidnud kasutust väga paljudes erinevates

valdkondades, muuhulgas ka veebirakenduste arenduses. Vaadates statistikat [9] , mis on kokku pandud GitHub[1] keskkonnas projektidele esitatud koodimuudatuste (*pull request*) ja Stack Overflow[2] keskkonnas esitatud küsimuste põhjal, saavutas Java populaarsuse osas teise koha. Seega on Java endiselt väga populaarne ja laialdaselt kasutusel olev programmeerimiskeel. Sama kinnitab ka Indeed[3] keskkonna tööpakkumiste põhjal kogutud statistika [10] , kus Java oli 62 000 tööpakkumisega ülekaalukalt esikohal edastades populaarsuselt järgmist programmeerimiskeelt 16 000 pakkumisega.

3.3.2 Spring

Uus rakendus on arendatud Spring raamistiku peale. Täpsemalt on kasutusel Spring Boot, mis pakub Spring raamistiku vaikimisi seadistusi populaarsemate kasutusjuhtude jaoks, sisseehitatud veebiserverit jms, et muuta veelgi lihtsamaks eraldiseisvate rakenduste arendamine [11] . Spring raamistikust on rakenduse arendamiseks kasutatud peamiselt Spring MVC ja Spring Security komponente, andmete relatsioonilise andmebaasi ja Java objektide vahel kaardistamiseks on kasutusel Hibernate[4] raamistik ning vaadete genereerimiseks kasutatakse Thymeleaf[5] mallimootorit (*template engine*).

Ka Spring raamistik on tänaseks juba võrdlemisi pika ajalooga olles esmase versiooniga välja tulnud 2004. aastal [12] . Samuti on see avalike andmete pealt kogutud statistika [13] põhjal olnud pikalt populaarseim raamistik Javas veebirakenduste arendamiseks.

3.3.3 Apache Maven

Programmikoodist töötava rakenduse ehitamiseks on kasutusel Apache Maven. Maven on tööriist Java projektide haldamiseks ja ehitamiseks, mis võimaldab kirjeldada ka sõltuvusi teistest projektidest ning neid sõltuvusi automaatselt projektile juurde laadida. Luues uue Spring Boot projekti genereeritakse ka projekti kirjeldav pom.xml fail Maveni jaoks, mis kasutab projekti ehitamiseks Spring Boot Maven pistikprogrammi. Selle tulemusel on võimalik ühe käsuga käivitada ehitusprotsess, mis kompileerib Java

1 https://github.com/
2 https://stackoverflow.com/
3 https://www.indeed.com/
4 https://hibernate.org/
5 https://www.thymeleaf.org/

koodi, käivitab automaattestid ning pakendab kogu rakenduse üheks käivitatavaks jar failiks.

3.3.4 Git

Versioonihaldustarkvarana on uue rakenduse jaoks kasutatud Giti. Git on 2005. aastal Linuxi kerneli arendamiseks loodud versioonihaldustarkvara, mille eesmärkideks on kiirus, lihtne disain, hea tugi paljudes paralleelsetes harudes arendamiseks, hajutatus ning hea tugi suurtele projektidele [14] . Ilmselt just tänu nendele omadustele on Git saavutanud tohutu populaarsuse ning tänapäeval võib seda pidada standardiks versioonihaldustarkvarade hulgas. Näiteks sel aastal Stack Overflow poolt läbi viidud küsitluses [15] kasutas 74 298 vastaja seast versioonihalduseks 87,2% just Giti.

3.3.5 Jenkins

Tarneprotsessi automatiseerimiseks otsustati kasutada Jenkinsit. Jenkins on avatud lähtekoodiga paindlik automatiseerimisserver (*automation server*), mis aitab automatiseerida kogu protsessi, mis puudutab tarkvaras tehtud muudatuste serverisse saatmist, võimaldades nii tagada pidevat integratsiooni (*continuous integration*).

Jenkins nime all tuli esimene versioon välja 2011. aastal [16] , kui otsustati luua uus haru (*fork*) Hudson nimelisest tarkvarast, mis oli sel hetkel Oracle'i[1] kontrolli all [17] . Hudsonit oli selleks hetkeks arendatud juba 2004. aastast alates [18] .

3.3.6 Docker

Vältimaks otseseid sõltuvusi serveri operatsioonisüsteemist ja paigaldatud tarkvaradest ning standardiseerimaks tarneprotsessi, on uue rakenduse käituskeskkonna tekitamiseks kasutusel Docker. Docker võimaldab rakendusi käivitada isoleeritud konteinerites, mis pakuvad rakendusele virtualiseeritud keskkonda koos kõige käivitamiseks vajalikuga. Erinevalt virtuaalmasinatest ei ole konteinerite jaoks emuleeritud kogu riistvara, vaid jagatakse operatsioonisüsteemi kernelit ja ressursse, mis muudab need oluliselt efektiivsemaks.

[1] https://www.oracle.com/

4 Uus rakendus

Uue rakenduse arendamise käigus ehitati välja vajalik struktuur, millel kogu rakendus põhineb, ning implementeeriti klientide mooduli funktsionaalsus. Selleks kirjutati kokku 6266 rida Java programmikoodi 98 failis.

Kuna olemasoleva rakenduse puhul puudub täpsem dokumentatsioon või nõuded, mis funktsionaalsust kirjeldaks, siis lähtuti uue rakenduse arendamisel eelkõige olemasoleva rakenduse käitumisest ja programmikoodist. Lisaks sellele võeti arvesse ka AS Smart Kindlustusmaakler soove täienduste ja muudatuste osas, mida võiks juba uues rakenduses teha.

Samuti hõlmas arendus ka uue rakenduse jaoks tarneprotsessi paika panemist. Selle alla võib lugeda nii Gitis versioonihalduse üles seadmist, serveris Jenkinsi paigaldamist ja seadistamist kui ka rakenduse Dockeris käivitamise võimaldamist.

Kõige selle juures oli oluline jälgida, et uus ja vana rakendus saaksid üleminekuperioodil töötada segamatult kõrvuti.

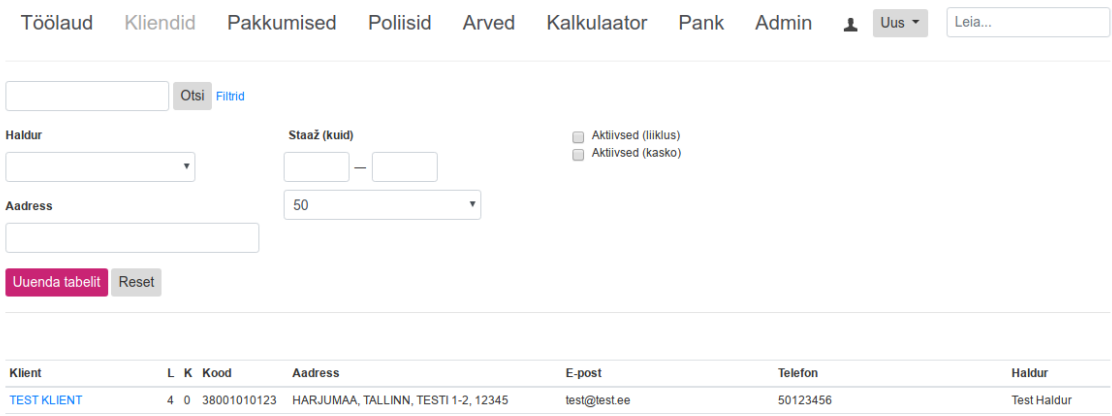
4.1 Olemasoleva rakendusega kõrvuti töötamine

Nõudel, et olemasolev ja uus rakendus kõrvuti töötaksid, on mitmeid põhjuseid. See võimaldab võrrelda uues rakenduses implementeeritud funktsionaalsust olemasolevaga ja anda kiiremat tagasisidet. Samuti võimaldab see ühe mooduli valmimisel juba järk-järgult uut rakendust kasutusele võtta, ilma et peaks ootama kõige valmimist ning siis korraga uuele rakendusele üle minema.

Ülemineku ajal kasutavad nii olemasolev kui ka uus rakendus sama andmebaasi, mis seab piirangud sellele, kui palju saab muuta andmemudelit ja andmebaasi struktuuri. Soovitud on vältida muudatusi, mis võiks tekitada olemasolevas rakenduses vigasid ning vajaks seega ka seal suuremaid muudatusi.

4.1.1 Kasutajaliides

Uue rakenduse kasutajaliidest on püütud hoida võimalikult sarnasena olemasoleva rakendusega, mistõttu on kasutajaliidese loomiseks kasutatud ka uues rakenduses Bootstrap[1] raamistikku. Küll aga on olemasolevas rakenduses kasutusel versioon 2 Bootstrapist, mis on uues rakenduses asendatud 4. versiooniga. Sellest tulenevalt on ka uue rakenduse kasutajaliideses visuaalselt väikseid erinevusi värvide, suuruste ning vahede osas, kuid üldiselt on kasutajaliides piisavalt sarnane vanale, nagu on näha Joonisel 5.



Joonis 5: Klientide nimekirja vaade uues rakenduses

Kasutajaliidese jaoks on oluline, et serveris uue rakenduse käivitamisel oleks seadistusfailis määratud `ee.smartkindlustus.old-app-url` parameetri väärtuseks olemasoleva rakenduse aadress. Selle põhjal koostatakse uues rakenduses hüperlingid nende moodulite jaoks, mida pole veel uues rakenduses implementeeritud.

4.1.2 Sessiooni jagamine

Et uue ja vana rakenduse kõrvuti kasutamine oleks mugavam, on oluline ka see, et rakendused jagaksid sessiooni, st ühes rakenduses autentides oleks kasutaja autenditud ka teises rakenduses. Selle saavutamiseks oli oluline esmalt mõista, kuidas toimub autentimine ja sessioonihaldus olemasolevas rakenduses.

[1] https://getbootstrap.com/

Lähemal uurimisel selgus, et olemasolevas rakenduses kasutusel olev CodeIgniter raamistik ei kasuta PHP-sse sisse ehitatud sessioonide tuge, vaid on ise implementeerinud sessioonihalduse[1], mis kirjutab sessiooni andmed andmebaasi. Tänu sellele oli ka lihtsam saavutada sessiooni jagamist.

Sessiooni identifikaatori edastamiseks kasutab CodeIgniter `ci_session` küpsist (Joonis 6). Sinna salvestatakse PHP `serialize` funktsiooniga serialiseeritult assotsiatiivne massiiv väljadega `session_id`, `ip_address`, `user_agent` ja `last_activity`. Lisaks liidetakse küpsise väärtusele juurde MD5 räsi, mis on genereeritud sama serialiseeritud stringi ja CodeIgniteri seadistusfailis määratud krüpteerimisvõtme põhjal. Koos küpsise väärtuse brauserisse saatmisega salvestatakse sessiooni andmed ka andmebaasi `user_session` tabelisse. Ka selles tabelis on eelpool mainitud neli välja ning lisaks väli `user_data`, kuhu salvestatakse samuti serialiseeritud kujul sessiooni andmed. Sisse logimisel lisatakse just sinna väli `user_id`, mille väärtuseks on sisse logitud kasutaja identifikaator.

```
a:4:
{s:10:"session_id";s:32:"2280a26cd1a76949753ce069614c1c72";s:10:"ip_address";
s:9:"127.0.0.1";s:10:"user_agent";s:68:"Mozilla/5.0 (X11; Linux x86_64;
rv:59.0) Gecko/20100101
Firefox/59.0";s:13:"last_activity";i:1524755168;}cbd67ff7b60885bf9f7d53da56d2
1714
```

Joonis 6: Küpsise ci_session näidisväärtus

Seega tuleb uues rakenduses kasutaja tuvastamiseks kontrollida päringuga saadetavaid küpsiseid ning püüda leida selle põhjal andmebaasist vastav sessioon ja seotud kasutaja. Selline küpsiste jagamine on võimalik tänu sellele, et uus ja vana rakendus töötavad samal aadressil, kuid erinevatel portidel. Kasutaja autentimise eest vastutavad uues rakenduses kaks klassi. `PhpSessionPreAuthenticatedProcessingFilter` klass seadistatakse Spring Security filtrina, mis otsib HTTP päringust `ci_session` küpsist, deserialiseerib selle väärtuse ning otsib tulemusest `session_id` väärtust. Lisaks on Spring Security seadistuses määratud kasutaja andmete otsimise teenuseks `PhpSessionPreAuthenticatedUserDetailsService` klass. Selle klassi

[1] https://github.com/bcit-ci/CodeIgniter/blob/e35658b5af498ff3634ca49ae83e006502b9b47d/system/libraries/Session.php

`loadUserDetails` meetodile antakse sisendiks eelnevas filtris leitud `session_id` väärtus, mille põhjal otsitakse andmebaasist `user_session` tabelist sessiooni, sealt omakorda `user_data` väljast `user_id` väärtust ning viimaks selle põhjal kasutajat `user` tabelist. Kui selle tulemusel kasutajat ei leita ning lehekülg, kuhu pöörduti, ei ole avalikult kättesaadav, suunatakse kasutaja vana rakenduse sisse logimise lehele.

4.2 Täiendused võrreldes olemasoleva rakendusega

Võrreldes olemasoleva rakendusega on uues rakenduses klientide moodulis tehtud ka mõningaid muudatusi ja täiendusi, milleks AS Smart Kindlustusmaakler jooksvalt soove avaldas:

- Klientide aadressid salvestatakse uues rakenduses suurtähtedega.
- Klientide nimekirjas on võimalik kliente otsida ka maakonna, linna/valla, küla/aleviku ning tänava järgi.
- Kliente saab määrata aktiivseks või passiivseks.
- Klientidele saab määrata suhtluskeele eesti, vene ja inglise keele hulgast.

Kuigi viimased kaks täiendust nõuavad muudatusi ka andmebaasi struktuuris, et oleks olemas väljad, kuhu see info kliendi kohta salvestada, ei ole see takistuseks, sest nende lisandumine olemasoleva rakenduse tööd ei sega.

4.3 Tarneprotsesss

Tarneprotsessi jaoks on esmane oluline samm, et rakenduse programmikood oleks versioonihaldustarkvaras. Uue rakenduse arendamiseks on kasutusel Git, kus lisaks `master` harule on ka `live` nimeline haru. Põhiline arendustöö tehakse `master` harus, `live` harusse mestitakse muudatused siis, kui need on valmis serveris käivitamiseks. Lisaks on rakenduse jaoks loodud ka privaatne projekt GitLab[1] keskkonnas, mis toimib keskse repositooriumina programmikoodi hoidmiseks.

[1] https://gitlab.com

Kõige tähtsam osa tarneprotsessis on Jenkinsil. Sarnaselt rakendusele otsustati Jenkins paigalda serveris Dockeri konteinerisse, mis aitab standardiseerida paigaldust ning käivitamist. Selleks loodi fail Dockerfile-jenkins (Lisa 1), mis võtab aluseks ametliku Jenkinsi Dockeri kujutise (*image*) ning paigaldab sinna lisaks Maveni ja Dockeri. Lisaks luuakse konteineris uus kasutajagrupp, mille identifikaatoriks määratakse Dockeri kujutise loomise hetkel argumendina kaasa antud `docker_gid` väärtus. Kasutaja jenkins lisatakse loodud gruppi, mis annab kasutajale piisavad õigused, et luua ning käivitada tarnitav rakendus omakorda eraldi Dockeri konteineris.

Jenkinsi paigaldamise ja käivitamise lihtsustamiseks loodi ka bash skript (Lisa 2), mis käivitab selleks kõik vajalikud käsud. Esmalt peatatakse ja eemaldatakse vana Dockeri konteiner, seejärel leitakse docker kasutajagrupi identifikaator ning edastatakse see järgmise käsuga Dockeri kujutise loomisel argumendina Dockerile. Viimaks käivitatakse uus Dockeri konteiner. Ühtlasi määratakse selle käsuga, et Dockeri sokkel (*socket*) konteineris viitab Dockeri sokkelile host operatsioonisüsteemis. Tänu sellele mõjutavad konteineris käivitatud Dockeri käsud host operatsioonisüsteemis töötavat Dockeri mootorit. See ongi viimane vajalik osa selleks, et võimaldada konteineris töötaval Jenkinsil kontrollida teisi temaga paralleelselt töötavaid Dockeri konteinereid.

Paraku vajab siiski Jenkinsi üles seadmine ka mitmeid käsitsi seadistusi, mida automatiseerida ei õnnestu. Samuti on vaja teha seadistusi GitLab keskkonnas, et Jenkinsil oleks ligipääs seal asuvale programmikoodile ning muudatuste puhul saadetaks Jenkinsile teavitusi. Täpsemad juhised nende seadistuste tegemiseks on ära toodud lisas Lisa 3.

Küll aga saab automatiseerida rakenduse ehitamist ja kävitamist, kirjeldades selleks vajalikud sammud ära failis Jenkinsfile (Lisa 4). Esmalt käivitatakse selleks Maveni käsk, mis kompileerib Java programmikoodi ning pakendab tulemuse kokku käivitatavaks jar failiks. Spring Boot raamistik lisab sinna faili ka Tomcat veebiserveri, mistõttu on võimalik kogu rakendust lihtsalt ühe failina käivitada. Serveris peab selleks olema paigaldatud ainult Java käituskeskkond. Sellegipoolest ei käivitata rakendust serveris otse, vaid luuakse faili Dockerfile (Lisa 5) põhjal Dockeri konteiner, milles on vajalik Java käituskeskkond olemas. Loodud jar fail ning serveripõhiste parameetrite seadistusfail kopeeritakse sinna konteinerisse ning konteiner käivitatakse.

4.4 Testimine

Iga rakenduse juures on oluliseks komponendiks ka automatiseeritud testid. Paraku on see üks aspektidest, millele pole olemasoleva rakenduse juures rõhku pandud, mistõttu puuduvad pea täielikult testid, mis rakenduse funktsionaalsust kataks. Seega on see samuti üks osa, mis uue rakenduse puhul peab saama paremini lahendatud. Selleks on uue rakenduse jaoks kirjutatud nii ühikteste kui ka integratsiooniteste, mis katavad 39% meetoditest ja 49% ridadest.

Testide kirjutamise juures on suureks abiks ka Spring raamistik, mis võimaldab `MockMvc` klassi abil väga lihtsalt simuleerida rakenduse poole päringute tegemist täpselt nii, nagu see toimub brauseri kaudu rakendust kasutades. Samuti on võimalik lihtsa seadistusega kasutada testide käivitamisel hoopis teist andmebaasisüsteemi, kui muidu rakenduses kasutatakse. Uue rakenduse puhul ongi testides kasutusel hoopis H2 andmebaasisüsteem[1], mis on täielikult mälus paiknev Java andmebaas. Tänu sellele ei vaja see mingit eraldi paigaldust ja on seega väga sobiv just testides andmebaasina kasutamiseks.

Testide käivitamine on ka üheks osaks tarneprotsessist. Testid käivituvad automaatselt, kui Jenkinsis hakatakse Maveniga rakendust ehitama. Juhul, kui mõnes testis tekib viga, jääb tarneprotsess pooleli ning uut rakendust koos muudatustega serveris ei käivitata.

4.5 Monitoorimine

Serveris töötava rakenduse puhul on oluline jooksvalt näha erinevat infot, mis rakenduse tööd puudutab. Eelkõige on seda vaja kahel põhjusel:

1. Vigade tuvastamine – kui programmi töös tekivad vead, on nende parandamiseks äärmiselt oluline saada võimalikult palju infot, kus ja mis põhjusel viga tekkis. Samuti võib rakenduse seadistada nii, et vigade korral saadetaks automaatne teavitus, mille peale saaks vea võimalikult kiirelt parandada.

[1] http://www.h2database.com/

2. Jõudluse jälgimine – meeldiva kasutuskogemuse pakkumiseks peab rakendus toimima võimalikult kiirelt. Monitoorimine aitab tuvastada kohti, kus lehe laadimine on aeglane, näiteks mõne aeglase andmebaasipäringu tõttu. Samuti võiks monitoorimine anda ülevaate serveri jõudlusest, et oleks võimalik reageerida olukorrale, kus koormuse kasvu tõttu hakkab serveri ressurss ammenduma.

Uue rakenduse puhul on monitoorimise võimaluste lisamiseks kasutatud peamiselt Spring Boot raamistikuga kaasa tulevat funtksionaalsust. Selleks on rakendusele lisatud Spring Boot Actuator komponent, mis annab võimaluse pärida suurt hulka erinevat infot, näiteks üldist staatust, erinevaid mõõdikuid, viimaseid HTTP pöördumisi, aktiivseid sessioone jne. Küll aga ananb rakendus seda infot lihtsalt JSON formaadis andmetena, mistõttu paremaks visualiseerimiseks tuleks kasutusele võtta ka mõni muu lahendus, mis selleks otstarbeks loodud on.

Lisaks on rakenduse seadistuses `logging.file` parameetri abil määratud fail, kuhu kirjutatakse jooksvalt logi rakenduse tegevuse ja vigade kohta. Kuna sama logi saadetakse ka standardväljundisse (*stdout*), siis on seda võimalik serveris kuvada ka käsuga `docker logs smartkindlustus`.

Jõudluse jälgimiseks kuvatakse ka iga lehe jaluses aega, kui palju selle lehe genereerimiseks serveril kulus, mis aitab leida kitsaskohti ning neid optimeerida.

5 Tulemus ja soovitused

Käesoleva bakalaureusetöö raames loodu on sobivaks aluseks, mille põhjal jätkata uue rakenduse arendamist. Küll aga on valminud klientide moodul vaid väike osa kogu rakenduse funktsionaalsusest, mistõttu vajab ülejäänud moodulite implementeerimine veel palju tööd. Kuigi loodud funktsionaalsust on osaliselt kaetud ka testidega, vajaks rakendus siiski veel ka rohkemate testide kirjutamist, et kaetavust tõsta.

Edaspidist arendustööd oleks soovituslik jätkata sarnaselt moodulite kaupa, võttes aluseks olemasoleva rakenduse toimimise ning arvestades ka ettevõtte soovidega muudatuste osas. Kui mõni moodul on valmis arendatud ja kõikide huvitatud osapoolte poolt testserveris üle vaadatud, võib nii järk-järgult asendada vana rakenduse mooduleid uue rakendusega. Selleks tuleks vaid vanas rakenduses hüperlingid vastavale moodulile ära muuta, kuid arvestada tuleks kindlasti sellega, et kogu moodul oleks enne terviklikult implementeeritud ja testitud.

Küll aga on ka mitmeid muudatusi, mida saab teha alles siis, kui kogu funktsionaalsus on uues rakenduses implementeeritud ning vana rakendus ei ole enam kasutusel. Nii näiteks tuleb ära kaotada praegune sessiooni jagamiseks kasutusel olev vana rakenduse küpsisega autentimine ning asendada see tavalise vormiga sisse logimisega. Samuti ei saa ette võtta suuremaid muudatusi andmebaasi struktuuris seni, kuni olemasolev rakendus on kasutusel.

Andmebaasi struktuuri muudatuste osas on antud töö skoobis tekkinud ka mitmeid muudatusettepanekuid:

- Klientide andmete dubleerimine `client` ja `customer` tabelites ei tundu vajalik. Kui neid on vaja selliselt eristada, siis võiks selleks kasutada eraldi tulpa `customer` tabelis.

- Iga pakkumisega luuakse hetkel uus kindlustusobjekt ja sõiduk. Kliendi detailses vaates grupeeritakse seetõttu andmeid sõiduki alla registrinumbri järgi. Kuna

registrinumber võib aga ka liikuda ühelt sõidukilt teisele, samuti võivad lihtsalt erinevatel sama sõiduki kohta käivatel ridadel olla vastuolulised andmed, siis ei tundu see korrektne käitumine. Ühe sõiduki jaoks võiks jääda `vehicle` tabelisse üks rida. Kui mõned sõiduki küljest tulevad andmed peavad säilitama ka oma ajaloolise väärtuse, tuleks need andmed salvestada mujale.

- Erinevate tüübi, staatuse jms väljade võimalike väärtuste jaoks on kasutusel seosetabelid väljadega `id` ja `name`. Kuna aga vaid väga vähestel juhtudel on võimalik neid väärtusi rakendusest seadistada, siis ei paista vajadust eraldi tabeliteks, vaid piisaks andmebaasis enum tüüpi väljadest. See aitaks vähendada tabelite arvu andmebaasis ning muudaks ka uues rakenduses nende väärtuste kaardistuse lihtsamaks.

Siiski ei tuleks neid võtta rangete soovitustena, mida tuleb kindlasti järgida, sest käesoleva töö raames ei ole nii põhjalikult süüvitud ülejäänud moodulite toimimisse. Küll aga võiks need olla punktid, mida tulevikus võetakse kaalumisele ning analüüsitakse põhjalikumalt.

6 Kokkuvõte

Käesoleva bakalaureusetöö eesmärkideks oli välja pakkuda arhitektuur ja tehnoloogiad AS Smart Kindlustusmaakler oleamasoleva veebirakenduse asendamiseks uue rakendusega. See hõlmas endas ka prototüübi arendamist klientide mooduli põhjal ning tarneprotsessi paika panemist muudatuste serverisse saatmiseks.

Eesmärkide täitmiseks tutvuti esmalt olemasoleva rakenduse funktsionaalsuse ja ülesehitusega, seejärel analüüsiti erinevaid võimalusi, kuidas saaks uue rakenduse arhitektuuri disainida, ning valiti välja sobivaimad lähtuvalt rakenduse ning ettevõtte vajadustest ning võimalustest. Tehtud otsuste põhjal valiti välja tehnoloogiad, mille abil uus rakendus realiseerida, ning arendati välja klientide moodul uues rakenduses. Seejuures jäid olemasolev ja uus rakendus serveris kõrvuti tööle ning uues rakenduses arendati välja ka võimalus olemasoleva rakendusega sessiooni jagada, et kasutajad ei peaks ennast eraldi mõlemas rakenduses autentima. Viimaks seadistati serveris automaatne tarneprotsess, mis võtaks muudatused versioonihaldustarkvarast ning käivitaks testide edukal läbimisel serveris uue rakenduse koos nende muudatustega.

Seega võib püstitatud eesmärgid lugeda täidetuks. Loodud on sobiv alus, mille põhjal uue rakenduse arendusega jätkata. Küll aga vajab kogu ülejäänud funktsionaalsuse uues rakenduses implementeerimine edaspidi veel palju tööd.

Kasutatud kirjandus

[1] Lewis, J., Fowler, M., „Microservices", 25.03. 2014 [WWW] https://martinfowler.com/articles/microservices.html (3. märts 2018).

[2] Annett, R., „What is a Monolith?", 19.11.2014 [WWW] http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html (3. märts 2018).

[3] Richardson, C., „Pattern: Microservice Architecture", 2014 [WWW]. http://microservices.io/patterns/microservices.html (3. märts 2018).

[4] Fowler, M., „MicroservicePremium", [WWW] https://martinfowler.com/bliki/MicroservicePremium.html (3. märts 2018)

[5] „ASP.NET MVC Overview", [WWW] https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx (24. aprill 2018)

[6] Reenskaug, T., „THING-MODEL-VIEW-EDITOR an Example from a planningsystem", [WWW] http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf (24. aprill 2018)

[7] Silver, A., "The disadvantages of single page applications" [WWW] https://adamsilver.io/articles/the-disadvantages-of-single-page-applications/ (24. aprill 2018)

[8] „The History of Java Technology", [WWW] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html (24. aprill 2018)

[9] O'Grady, S. „The RedMonk Programming Language Rankings: January 2018" [WWW] http://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/ (24. aprill 2018)

[10] Misirlakis, S., „The 7 Most In-Demand Programming Languages of 2018" [WWW] https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/ (24. aprill 2018)

[11] „Spring Boot" [WWW] https://projects.spring.io/spring-boot/ (24. aprill 2018)

[12] Risberg, T., „Spring Framework 1.0 Final Released" [WWW] https://spring.io/blog/2004/03/24/spring-framework-1-0-final-released (24. aprill 2018)

[13] „Java Web Frameworks Index" [WWW] https://zeroturnaround.com/webframeworksindex/ (24. aprill 2018)

[14] „1.2 Getting Started - A Short History of Git" [WWW] https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git (26. aprill 2018)

[15] „Developer Survey Results 2018" [WWW] https://insights.stackoverflow.com/survey/2018 (26. aprill 2018)

[16] Kawaguchi, K., „Jenkins 1.396 released” [WWW] http://jenkins-ci.361315.n4.nabble.com/Jenkins-1-396-released-td3257106.html (26. aprill 2018)

[17] Bayer, A., „Hudson's future” [WWW] https://jenkins.io/blog/2011/01/11/hudsons-future/ (26. aprill 2018)

[18] Kawaguchi, K., „Hudson” [WWW] https://web.archive.org/web/20140701020639/https://www.java.net//blog/kohsuke/archive/20070514/Hudson%20J1.pdf (26. aprill 2018)

Lisa 1 – Dockerfile Jenkinsi paigaldamiseks

```
FROM jenkins/jenkins:lts
ARG docker_gid

USER root

# Install maven
RUN apt-get update && apt-get install -y maven

# Install docker
RUN apt-get -y install apt-transport-https ca-certificates curl gnupg2
software-properties-common && \
  curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo
"$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
  add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release;
echo "$ID") \
    $(lsb_release -cs) \
    stable" && \
  apt-get update && \
  apt-get -y install docker-ce

# Add jenkins user to docker group - allows running docker commands
RUN groupadd -g $docker_gid docker-host && usermod -aG docker-host jenkins

USER jenkins
```

Joonis 7: Dockerfile Jenkinsi paigaldamiseks

Lisa 2 – Bash skript Jenkinsi paigaldamiseks ja käivitamiseks

```
#!/bin/bash

# Remove previous container
docker stop jenkins-smartkindlustus || true
docker rm jenkins-smartkindlustus || true

docker_gid=`stat -c "%g" /var/run/docker.sock`

# Build and run new container
# docker.sock is linked to container to enable managing other docker
containers from Jenkins
docker build \
   -t jenkins-smartkindlustus \
   -f Dockerfile-jenkins \
   --build-arg docker_gid=$docker_gid \
  . && \
docker run \
   -d \
   -p 8083:8080 \
   --name jenkins-smartkindlustus \
   -v jenkins_home:/var/jenkins_home \
   -v /var/run/docker.sock:/var/run/docker.sock \
  jenkins-smartkindlustus
```

Joonis 8: Bash skript Jenkinsi paigaldamiseks ja käivitamiseks

Lisa 3 – Jenkinsi seadistamise juhend

1. Kopeeri Dockerfile-jenkins (Lisa 1) ja jenkins.sh (Lisa 2) serverisse. Käivita skript, et paigaldada ja käivitada Jenkins:

```
$ chmod u+x jenkins.sh
```

```
$ ./jenkins.sh
```

2. Ava käsurida käivitatud konteineris:

```
$ docker exec -it jenkins-smartkindlustus bash
```

3. Kopeeri Jenkinsi poolt genereeritud parool failist /var/jenkins_home/secrets/initialAdminPassword:

```
$ cat /var/jenkins_home/secrets/initialAdminPassword
```

4. Loo SSH võtmed jenkins kasutajale (konteineris):

```
$ ssh-keygen
```

5. Kopeeri loodud avalik võti hilisemaks kasutamiseks:

```
$ cat /var/jenkins_home/.ssh/id_rsa.pub
```

5. Ava brauseris Jenkinsi aadress (http://*server*:8083).

6. Sisesta sammus 3 kopeeritud parool.

7. Vajuta "Install suggested plugins"

8. Loo admin kasutaja

9. Seadista loodud SSH võti Jenkinsis:

Credentials > System > Global credentials > Add credentials

Kind: SSH Username with private key

Username: jenkins

Private key: From the Jenkins master ~/.ssh

10. Lisa GitLabis sammus 5 kopeeritud avalik võti kasutaja seadete alla:

Settings > SSH keys

11. Loo Jenkinsis uus projekt:

New item > Pipeline

Vali „Poll SCM”

Definition: Pipeline script from SCM

SCM: Git

Repository URL: GitLabi repositooriumi aadress

Credentials: jenkins

Branches to build: live/master vastavalt serverile

12. Paigalda „Config File Provider” pistikprogramm:

Manage Jenkins > Manage Plugins > Available > Install

13. Lisa Jenkinsisse seadistusfail rakenduse serveripõhiste seadituste jaoks (andmebaasi ligipääsud, port jne):

Projekti vaade > Config files > Add a new Config > Custom file

ID: application.properties

Failis peaks määrama väärtused järgnevatele seadistustele:

spring.datasource.url (andmebaasi aadress),

spring.datasource.username (andmebaasi kasutajanimi),

spring.datasource.password (andmebaasi parool),

muud vajalikud seadistused (server.port, ee.smartkindlustus.old-app-url jne)

14. Lisa GitLabis webhook, mis teavitaks Jenkinsit muudatustest

Settings > Integrations > Add webhook

URL: http://*server*:8083/git/notifyCommit?url=*gitlabi_repositooriumi_aadress*

Trigger: Push events

Lisa 4 – Jenkinsfile rakenduse ehitamiseks ja käivitamiseks

```
pipeline {
    agent any
    triggers {
        pollSCM('')
    }
    stages {
        stage('Build & Test') {
            steps {
                sh 'mvn clean package'
            }
            post {
                success {
                    junit 'target/surefire-reports/**/*.xml'
                }
            }
        }

        stage('Deploy') {
            steps {
                configFileProvider([configFile(fileId:
'application.properties', variable: 'properties')]) {
                    sh 'cp $properties server.properties'
                    sh 'docker stop smartkindlustus || true'
                    sh 'docker rm smartkindlustus || true'
                    sh 'docker rmi smartkindlustus || true'
                    sh 'docker build -t smartkindlustus .'
                    sh 'docker run -d --name smartkindlustus --net=host
smartkindlustus'
                }
            }
        }
    }
}
```

Joonis 9: Jenkinsfile rakenduse ehitamiseks ja käivitamiseks

Lisa 5 – Dockerfile rakenduse paigaldamiseks

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
COPY target/smartkindlustus.jar app.jar
#File for server specific properties (database credentials, port etc) - should be created in Jenkins
COPY server.properties application.properties
ENV JAVA_OPTS=""
ENTRYPOINT exec java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar
```

Joonis 10: Dockerfile rakenduse paigaldamiseks