Tallinna Ülikool
Digitehnoloogiate Instituut

Autoremonditöökoja CRM võimaluste loomine Viruauto näitel

Bakalaureusetöö

Autor: Kristo Jürgenson
Juhendaja: Jaagup Kippar

Autor:.....”.....”.....2017

Juhendaja:.....”.....”.....2017

Instituudi direktor:.....”.....”.....2017

Tallinn 2017

Autorideklaratsioon

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

…… ……..(kuupäev)(autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Kristo Jürgenson (sünnikuupäev: 30.09.1993)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Autoremonditöökoja CRM võimaluste loomine Viruauto näitel" mille juhendaja on Jaagup Kippar, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas

Sisukord

Tähiste loetelu

AJAX	Asynchronous JavaScript and XML - andmevahetus brauseri ja veebiserveri vahel, nii et kasutaja iga liigutuse peale pole vaja kogu veebilehte uuesti alla laadida (E-teatmik, kuupäev puudub).
AWS	Amazon Web Services - pilvepõhine veebimajutus keskkond (Rouse, 2014).
CRM	Customer Relationship Management - kliendisuhete haldus, mida kasutatakse firma müügieelse ja müügijärgse tegevuse planeerimiseks ja juhtimiseks (E-teatmik, kuupäev puudub).
CRUD	Create, Read, Update and Delete - kasutajaliidese funktsioonid andmebaasiga töötamiseks (Techopedia, kuupäev puudub).
EC2	Elastic Compute Cloud - Amazon Web Services poolt pakutav virtuaalmasin (Amazon Web Services, kuupäev puudub).
S3	Simple Storage Service - Amazon Web Services poolt pakutav andmesalvestamise keskkond.
URL	Uniform Resource Locator - internetiaadress Igale dokumendile või muule ressursile Internetis vastab oma unikaalne internetiaadress (E-teatmik, kuupäev puudub).

Sissejuhatus

Käesoleva bakalaureusetöö eesmärk on arendada autoremondi töökojale esialgne versioon CRM süsteemist, mis kiirendaks töökoja tööprotsessi. Eesmärgi saavutamiseks on vaja koostada skoop ja luua plaan, kuidas antud rakendust arendada. Töö teema valik tuleneb autori huvist erinevate tehnoloogiate vastu ja soovist saada uusi kogemusi, mis tulenevad rakenduse arendamise protsessist. Töö eesmärk on saavutatud kui mingi versioon rakendusest on töökoja poolt kasutusel.

Esimeses peatükis on kirjeldatud rakenduse esimese versiooni vajadusi ning millist funktsionaalsust oodatakse rakenduselt kaugemas tulevikus. Võrreldakse sarnaseid rakendusi ja nende pakutavaid võimalusi ning lisaks on välja toodud erinevad komponendid, mida kasutatakse rakenduse arendamise protsessis.

Teine peatükk läbib rakenduse arendamise protsessi. Esmalt luuakse arendusplaan ja selle sama peatüki vältel läbitakse tehtud plaan. Tuuakse veel välja uued komponendid, mis on juurde võetud.

Kolmandas peatükis kirjeldatakse valmis rakenduse tootmise keskkonda tõstmist ning tehakse analüüs rakenduse käigus esinenud vigadele. Peatüki lõpus on ära märgitud, mida oleks vaja veel teha, et süsteem vastaks kliendi soovidele.

Rakendusega tutvumiseks võib võtta autoriga ühendust e-maili teel rakenduse autoriga aadressil: kristo.j@hotmail.com.

1 Töökoja haldamine ja vajadused

Käesolevas peatükis antakse ülevaade, mida kliendihalduse rakendus endast kujutab, mis on selle eelised ja omadused ning mida klient ootab ja tahab rakenduselt saada. Tehakse põgus ülevaade veebis olevatest sarnastest rakendustest, mida on võimalik mingi perioodi vältel tasuta kasutada. Tuuakse välja, mis funktsionaalsuseid need pakuvad ja kuidas on rakendus visuaalselt üles ehitatud. Lisaks märgitakse ära, mis tööriistu kasutatakse rakenduse arendamise protsessis.

1.1 Kliendihalduse rakendus

Kliendisuhete juhtimine (ingl. keeles *Customer Relationship Management* ehk CRM) on termin, mis viitab tavadele, strateegiatele ja tehnoloogiatele mida ettevõtted kasutavad, et käsitleda ja analüüsida kliendi interaktsioone ja andmeid läbi kliendi teenindamise tsükli. (Margaret R, 2014). CRM süsteemis on kõige tähtsam kliendiga seonduv info. Kliendi infot peab olema võimalik salvestada ja töödelda nii, et sellest oleks võimalik luua mingit väärtust ettevõttele. Mida rohkem on kliendiga seotud infot, seda kergem on mõista ja välja selgitada kliendi vajadusi. Süsteemi eesmärk on kliendiga ärisuhete parandamine ja müügitulu tõstmine, mis peaks kliendis tekitama tunde, et vajadustest saadakse aru ning ettevõttega tegutsedes ei teki ebameeldivaid kogemusi.

Müügiga seotud ettevõtted kasutavad CRM süsteeme, et müügitulu suurendada. Viruauto[1] näol on tegemist väikese autoremonditöökojaga, mis tegeleb autoremondi teenuse osutamisega. Antud ettevõtte puhul ei pea kliendihalduse rakendus müügi aspektile väga rõhku paigutama. Üks põhieesmärke on kliendi kogemuse parandamine. Rakendus peab tekitama positiivse kogemuse selle näol, et kliendil on ettevõttega suhtlemine kiire, lihtne ja informatiivne. Remonditöökoda hoiab klienti kursis, mis seisus on kliendi auto ja milliseid töid oleks vaja läbida, et saada auto töökorda.

[1] http://viruauto.ee/

1.2 Kohtumine kliendiga

Kliendiga esmasel kokkusaamisel sai kirja pandud punktid, mida esialgu rakenduselt oodatakse. Lisaks lepiti kokku tehnoloogiad, mida hakatakse kasutama suhtlemiseks ja rakenduse staatuse kirjeldamiseks. Kuna Viruauto eelnevalt autoremonditöökoja spetsiifilist rakendust ei kasutanud, vaid kasutasid *Google*'i poolt pakutavaid dokumendi rakendusi, siis esialgne funktsionaalsus pidi võimaldama töökoja haldamist.

Esimene versioon pidi:

- Asendama praegust lahendust
- Kiirendama ja lihtsustama töökoja tööprotsessi
- Parandama suhtlust juhataja ja mehaaniku vahel
- Kuvama lihtsalt ja kiirelt andmeid ning broneeritud teenuseid
- Broneerima ja kuvama teenuseid erinevate tõstukite vahel
- Võimaldama kliendi ja auto andmete salvestamist

Antud punktid märgivad ära esimese versiooni, mida esialgu oleks vaja töökoja haldamiseks. Vajadused tulenevad sellest, et vähendada segadust ja paberimajandust. Teenuste otsimine ja filtreerimine *Excel* tabelitest on suhteliselt aeganõudev töö. Mehaanikud peaksid paberitäitmise asemel märkima varuosasid läbi arvuti.

Lisaks esimese versiooni punktidele sai üles märgitud, mida rakendus peaks hakkama tulevikus võimaldama.

Tuleviku visioon rakendusest:

- Erinevad vaated töökoja kasutajatel (juhataja, mehaanik, klient)
- Reaalajas teadete kuvamine
- Teenustele varuosade lisamine/tellimine
- Teenuse staatus
- Staatuse põhjal teavitamised
- Kliendile automaatne kirja saatmine, kui teenus või staatus on muutunud

- Kliendi sektsioon - vaated ja seisud kliendi auto kohta

1.3 Sarnased rakendused

Enne üldplaani koostamist tuleks uurida, millised autoremonditöökoja haldusrakendused on juba olemas. See annab võimaluse tutvuda, kuidas on sarnastel rakendustel äriline pool üles ehitatud ning kuidas on vaadete ja andmete lisamine tehtud. Lisaks saab mõningaid ideid selle kohta, kuidas mõned süsteemi komponendid arendada.

Uuritavad rakendused peavad vastama tingimustele:

- Veebipõhine
- Võimalus tasuta kasutada mingi perioodi vältel

Paljud ettevõtted pakuvad rakenduse tasuta perioodi ainult kokkuleppe alusel. Kuna autor ei otsi võimalikku asendust, siis neid rakendusi võrdlusesse ei tooda. Paljude rakenduste kasutamine on välistatud, kuna litsentsti või kasutaja saamiseks nõutakse igakuist makset või peab kohe täishinda maksma. Litsentsi täishind on tavaliselt kuskil 1500 euro ulatuses ning kasutajate kuuhind on keskmiselt vahemikus 70-120 eurot. Väikesele töökojale, millel on kuni kolm kasutajat, pole see väga suur kulu. Samas kui töökojal tekib vajadus spetsiifiliste funktsionaalsuste järgi, siis neid rakendusi pole võimalik muuta. Rakendused valiti selle järgi, mis on *Google* otsinugmootoris kõige kergemini leitavamad.

1.3.1 Auto Repair cloud

Joonis 1. Auto Repair Cloud

Auto Repair Cloud[2] on pilvepõhine kõik-ühes platvorm, mis on mõeldud autoremonditöökoja haldamise jaoks. Funktsionaalsuste alla kuuluvad näiteks: tööde planeerimine, arvete loomine, arvete saatmine, automaatne juppide tellimine, tööde ajalugu ning kliendi sektsiooni veebileht (Capterra, kuupäev puudub).

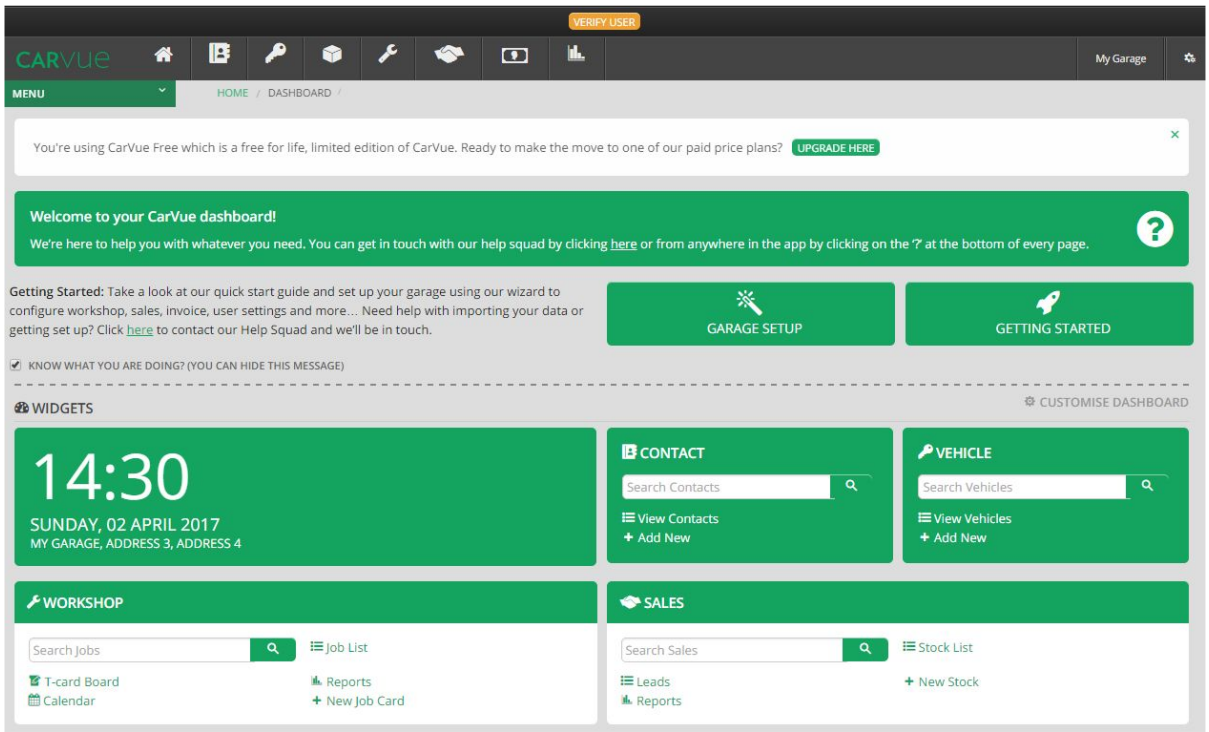
Rakenduse kasutamiseks tuleb esmalt Auto Repair Cloudi veebilehel registreerida kasutaja, kus peab olema täidetud töökoja telefoninumber, töökoja aadress ning nimi. Auto Repair Cloudi tasuta versioon ei paku eriti palju võimalusi töökoja haldamiseks. Veebilehe ülesehitus on suhteliselt keeruline ning väga tihti võib tekkida olukordi, kus võib rakenduse moodulite vahel navigeermisel ära kaduda. Töökoja juhataja rolli täites tekkis palju segadust esialgsel ülesseadistamisel ning uute teenuste registreerimisega. Ülesseadistamise saavutamiseks tuli esmalt ära märkida töökoja lahtioleku ajad, mis pidid vastama mingitele kindlatele reeglitele. Võimalik ei olnud näiteks töökoda pühapäeviti üks tund lahti hoida.

[2] https://autorepaircloud.com/

Kontrollpaneelis valele kohale vajutades viskab alati tagasi Auto Repair Cloudi esilehele ning et tagasi paneeli vaatesse saada, tuli alati uuesti sisse logida.

Tööaja registreerimine ja mehaanikute aja planeerimine võtab palju aega, kuna peab pidevalt navigeerima mitme mooduli vahel, et ühe teenuse andmeid muuta. Kliendile aja määramine ja probleemide kirja panemine käib ka läbi mitme vormi, mis võtab palju aega ning mille tulemusena võivad teised potentsiaalsed kliendid lahkuda. Suurtemate töökodade puhul ei oleks see tõenäoliselt probleemiks, aga väiksemad ärid peavad teenused ja andmed kiiresti kirja panema. Arvete genereerimine ja automaatne saatmine on väga lihtsalt ülesehitatud. Teenuse valmimisel on võimalik sellega seotud arve kas saata kohe kliendi e-mailile või välja printida.

1.3.2 Carvue



Joonis 2. Carvue

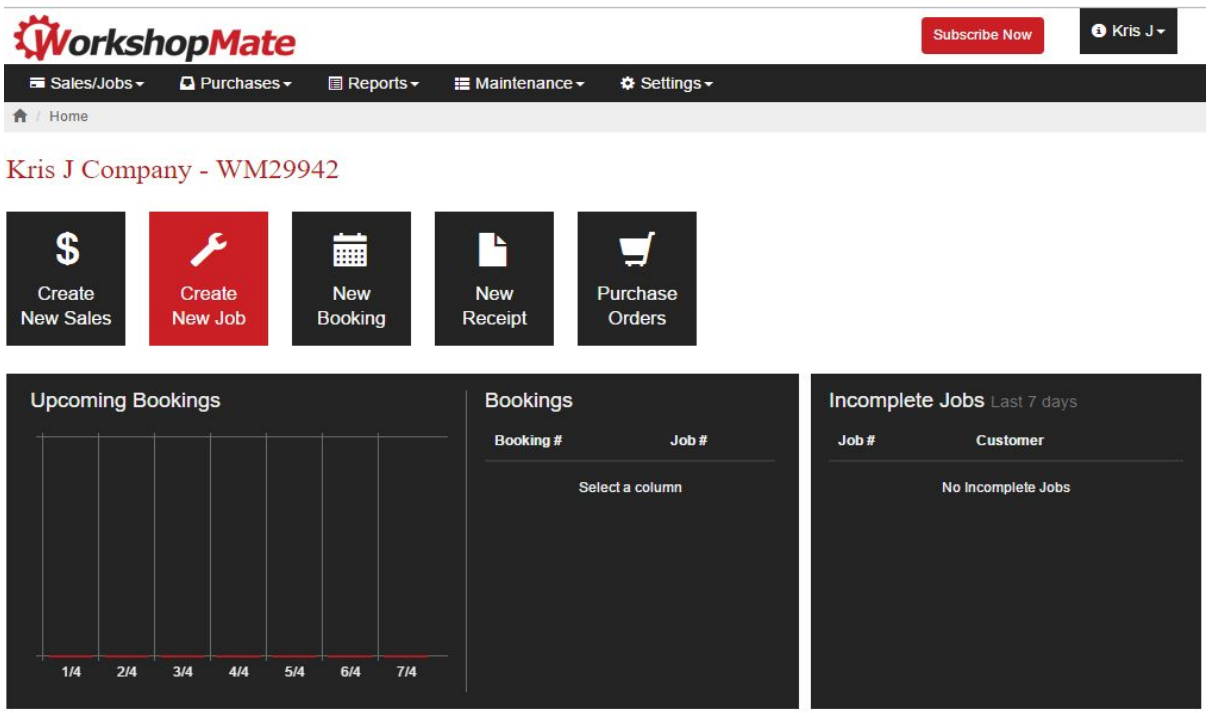
CarVue[3] on veebipõhine töökojahalduse rakendus, millel on moodulid väga lihtsa ülesehitusega. Rõhk on asetatud suure info koguse kuvamisele ning igasugune andmete lisamine ja kuvamine käib läbi tabelite ja nendega seotud vormide. Tähtsamad funktsioonid

[3] https://www.carvue.com/

on olemas ning kergelt kasutatavad. Näiteks teenuste registreerimine, klientide ja autode lisamine ning teenuste aja registreermine. Andmete lisamise vormidel on klientide ja autode seosed hästi lahendatud. Uute teenuste loomisel ei pea moodulite vahel navigeerima, sest teenuse loomise vormi peal on võimalik kuvada kliendi ja auto salvestamise vormi. Tööde staatused on väga huvitavalt lahendatud kombineerides värve ja veerge. Veerud on staatused ning töid saab nende vahel lohistada sarnaselt nagu on Trellos lahendatud.

Kalendri element on samuti väga huvitavalt lahendatud. Klassikalise kalender-vaate asemel on päevade elemendid, kus on näha, mitu tööd on antud päevale registreeritud. Elemendi avamisel on võimalik kuvada kõik tööd ning need mehaanikute vahel ära jagada. Mehaanik näeb päeva peal ainult neid ülesandeid, mis on talle määratud. Samuti on võimalik teha arveid peale töö lõpetamist. Arve luuakse malli alusel, kus kliendi ja auto andmed on juba täidetud. Võimalik on lisada erinevaid töid, mida autoga tehti ning arve automaatselt arvutab hinna kokku. Kliendiga mingit automatiseeritud suhtlemist rakendus ei võimalda ning juhataja peab kõik tegema läbi emaili või telefoni.

1.3.3 Workshop Mate



Joonis 3. Workshop Mate

Workshop Mate[4] on veebibrauseri põhine autotöökodade haldusrakendus. Workshop Mate rakendus sisaldab palju funktsioone, mille eesmärk on säästa kasutaja aega ja vaeva remonditöökoja tööprotsessis. See on arendatud eesmärgiga, et lõpp-kasutajal oleks võimalikult kiire ja lihtne töökoda hallata (Workshop Mate, kuupäev puudub). Uute teenuste, klientide ja autode lisamine on kiire. Rakendus sisaldab andmebaasi varujuppide kohta, mida saab automaatselt juurde tellida ning teenusele on neid lihtne lisada. See on esimene rakendus, mis võimaldab mitme autoremonditöökoja haldamist.

Workshop Mate teenuse registreerimise vorm on omapärane, sest see luuakse kohe arve malli peale. Selle täitmisel peab kindlasti olema lisatud tööd, mida hakatakse tegema ning sellega lüüakse arve hind ka kohe kokku. Jääb mulje, et kui klient hakkab tööd registreerima, siis ta peaks teadma, mis töid tema auto vajab. Õli- ja rehvivahetuse puhul oleks see ideaalne, aga tööde puhul võib alati tekkida ootamatuid probleeme ja lisaväljaminekuid. Visuaalselt on keskkond väga hästi üles ehitatud - kõik komponendid on kergesti leitavad ja eristatavad. Teenusele kliendi ja auto määramise osa on hästi üles ehitatud. Uue kliendi ja auto lisamine on väga lihtne ja kiire, analoogselt üles ehitatud nagu CarVuel, erinevused seisnevad vaid visuaalsel poolel. Rakenduses on veel mitmed kliendiga suhtlemise funktsioonid, mille kasutamine maksab iga kirje saatmise pealt.

1.3 Süsteemi arendamise protsessis kasutatavad komponendid

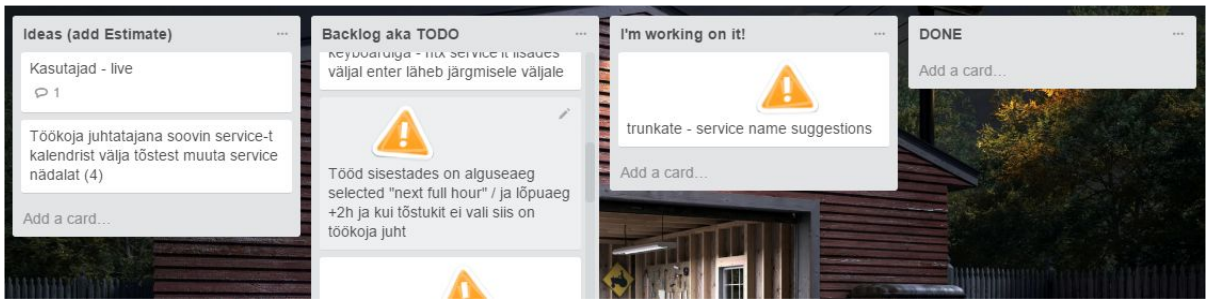
Käesolevas peatükis tuuakse välja rakendused, mida kasutatakse arendamisprotsessi dokumenteerimiseks ning kasutajalugude märkimiseks. Laravel veebiraamistiku tehnoloogiaid eraldi välja ei tooda, kuna autori seminaritöös on need detailsemalt kirjeldatud.

1.3.1 Trello

Rakenduse seisu dokumenteerimiseks hakatakse kasutama Trellot[5]. Trello on koostöö tööriist, mis organiseerib projekti kergesti loetavateks tabeliteks (Trello, 2016). Trellos on võimalik

[4] http://www.workshopmate.com.au/
[5] https://trello.com/

väga lihtsalt näidata, mis seisus on käesolev projekt ning klient saab sinna lisada uusi kasutajalugusid ning ülesandeid. Trellos olevate veergude järgi saab ära märkida, mis seisus on ülesanded. Samuti saab dokumenteerida ja arhiveerida valmis töid ning nendele lisada erinevaid juhendeid, pilte ja kommentaare. Käesoleva töö tarvis kasutatakse Trellot veel vigade märkimiseks, funktsionaalsuste dokumenteerimiseks, kommenteerimiseks (Joonis 4) ning töökoja poolt uute vajaduste märkimiseks.



Joonis 4. Näide Trellos märgitud ülesannete kohta

1.3.2 Versioonihaldus

Versioonide haldamisel ja võimalike duplikaat keskkondade loomisel kasutatakse GitHub[6] tööriista. GitHub on veebipõhine versioonihalduse rakendus, mis võimaldab loodud projekte ja faile repositooriumisse salvestada (How to geek, 2014). Repositooriumeid saavad ka teised muuta ja arendada ning on võimalik luua erinevaid paralleelseid arendusi funktsionaalsuste testimiseks. Lisaks lihtsustab see projekti tõstmist erinevate keskkondade vahel.

1.3.3 Arenduskeskkond

Arenduskeskkond sai ülesseadistatud autori seminaritöö[7] põhjal AWS[8] veebimajutus keskkonda Laravel veebiraamistiku baasil, millele on juurde lisatud LaraAdmin administreerimispaneeli pakk.

Tuleks ära mainida, et arenduskeskkond seati üles veebimajutuse keskkonda selleks, et töökoja kasutajad saaksid arendusprotsessi käigus testida uusi lahendusi ja funktsioone. Lokaalsele masinale keskkonna ülesseadistamisel võivad tekkida olukorrad, kus masin võib

[6] https://github.com/
[7] https://docs.google.com/document/d/1XIhF0gM0rwhol4yiGsSmltaLB7cEfU3L3f6nG3rcNDc
[8] https://aws.amazon.com/

ennast välja lülitada või jõudlusest ja mahust jääb väheseks. Lisaks selgus testimise käigus, et lokaalse masina veebiserveris aegus kasutaja sessioon iga kolme päringu peale ning sellele lahendust esialgu ei leitud. Kui arenduskeskkonnas olev haldusrakendus vastab esimese versiooni nõuetele, siis tehakse sellest koopia ning tõstetakse tootmise keskkonda. Tootmise keskkond tõstetakse omakorda AWS poolt pakutavale EC2 teenuse isendile.

2 Arenduskäik

Arendamise alustamiseks on vaja luua arendusplaan, mille alusel hakatakse looma tabeleid, klasse ja mooduleid. Üldplaani saamiseks on vaja välja selgitada, kuidas remonditöökoja tööprotsess algul töötab.

2.1 Analüüs

Arendusplaani koostamiseks oleks vaja välja tuua, kuidas toimub teenuste registreerimine ja andmete salvestamine enne CRM süsteemi implementeerimist. Selle põhjal on võimalik välja selgitada, mis tabeleid ja mooduleid oleks vaja luua ning kuidas võiks teenuse lisamise vorm välja näha.

Autoremonditöökoja teenuse registreerimise protsess enne CRM süsteemi:

1. Klient helistab või läheb töökotta ning soovib aega broneerida
2. Juhataja annab teada, mis ajal on võimalik hakata autoga tegelema
3. Kui kliendile sobib pakutud aeg, siis juhataja loob teenuse kirje auto jaoks ning paneb kirja kliendi andmed
4. Kliendi andmete lisamisel pannakse kirja ka auto andmed ning märgitakse ära tööd, mida ära tuleb teha
5. Kui auto läheb töösse, siis kontrollitakse üle need punktid, mis kirja on pandud
6. Kui autoga töötamise ajal leitakse mingid vead, millest peaks klienti teavitama, siis seda ka tehakse ning lepitakse kokku, mis edasi saab
7. Kui on vaja varuosasid, siis need märgitakse tõstuki lehele
8. Töö lõppedes märgib mehaanik kui kaua oli auto tõstukil ning kui on vaja varuosasid juurde, siis märgib milliseid

Andmete salvestamine käis algselt *Google* arvutustabelisse ning tööd lisati *Google* kalendrisse. Antud rakenduste kasutamine töökoja protsessi käigus on suhteliselt ajakulukas. Töökoja sisene suhtlemine läbi nende rakenduste on keeruline ja vahepeal tekitavad segadusi. Mehaanikute ja juhataja tööprotsessi kiirendamiseks on Laravelis vaja luua moodulid, kus oleks võimalikult kiire andmeid lisada, muuta ning täiendada. Kasutamine peab olema sujuv ja lihtne ning sisestatud andmed peavad olema kergesti filtreeritavad ning eristatavad. Kõige tähtsam vaade hakkab olema teenuse vaade, kuna selles kuvatakse tõstukite aegu ning kõiki registreerituid teenuseid. Sinna alla kuulub ka autode ja klientide lisamine ning olemasolevate nägemine, ilma et peaks moodulite vahel navigeerima. Lisaks peab olema võimalikult lihtne olemasolevaid töid ajaliselt ringi paigutada ning sisu muuta, kui peaks selleks vajadus tekkima.

Teiste sarnaste rakenduste puhul oli rohkem rõhku pandud erinevate moodulite info kuvamisele ja funktsioonidele. Olemasoleva teenuse info muutmiseks oli moodulite vahel navigeerimine tavapärane. Lisaks oli teistel rakendustel teenus broneeritud mehaanikutele ning mehaanik pidi eraldi broneerima aja tõstukile. Viruauto puhul on vaja leida moodus, et töö lisamine ja uute andmete salvestamine, mis kaasnevad teenuse lisamisega, toimuks võimalikult kiiresti ja sujuvalt. Teenuseid hakatakse planeerima ka tõstukitele, kuna igale tõstukile on määratud tavaliselt üks mehaanik.

Aegade visuaalne paigutamine hakkab käima FullCalendar[9] javascript teegi peale, kuna see oli kohe projektiga kaasas ning seda on lihtne implementeerida vaadetesse.

Kui mehaanik täiendab teenuseid või lisab sinna nõudeid, siis oleks vaja reaalajas saata teavitusi töökoja juhatajale, et töödele on tehtud täiendusi. Selleks ei pea hakkama saatma e-maili või mobiiliga sõnumeid, vaid rakenduse siseselt tuleb teavitus, et teenust on muudetud.

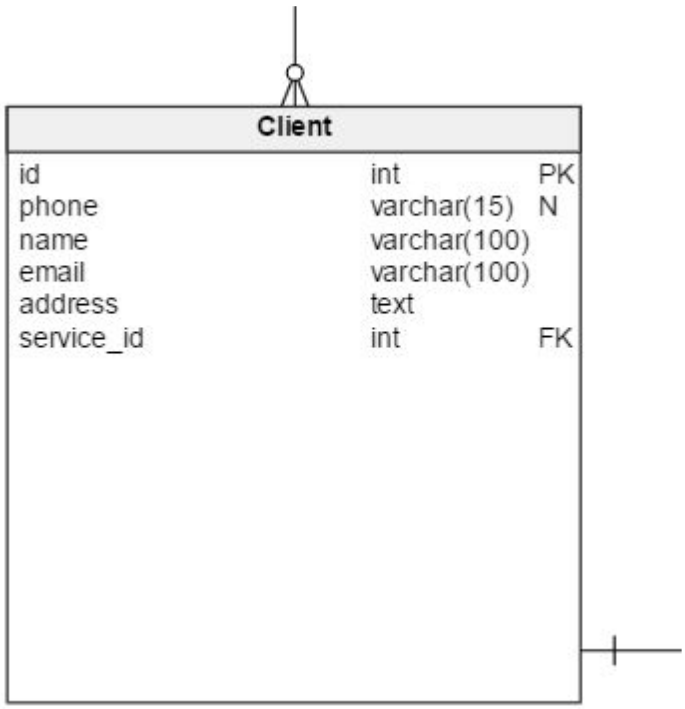
Enne moodulite loomist tuleks välja selgitada, millised hakkavad olema andmebaasi tabeli mudelid ja kuidas on need omavahel seotud. Autor toob välja kolme kõige tähtsama elemendi

[9] https://fullcalendar.io/

ehituse ja nende omavahelise seose. Tasub ära märkida, et LaraAdmin paki ülesseadistamisega on süsteemi põhised moodulid juba automaatselt loodud näiteks kasutajad ja kasutajaterollid.

2.1.1 Klient

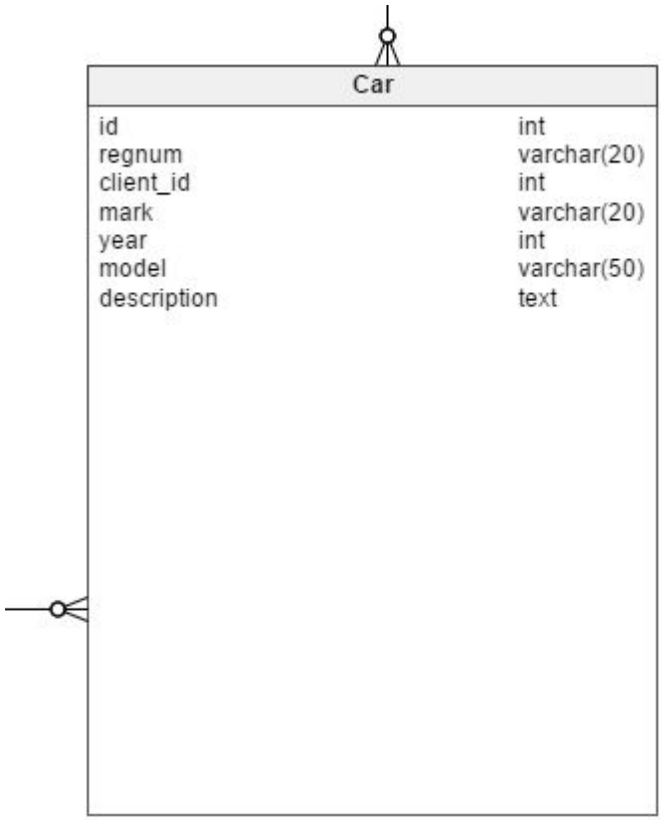
Kliendibaas (Joonis 5) hakkab salvestama infot, tänu millele on võimalik pidada kliendiga ühendust. Kõige tähtsam väli, mis hakkab kliente eristama ja mis peab alati täidetud olema on telefoninumber. Esialgu pole klienditabelis mitte ühtegi viidet teistele tabelitele.



Joonis 5. Kliendi tabeli skeem

2.1.2 Auto

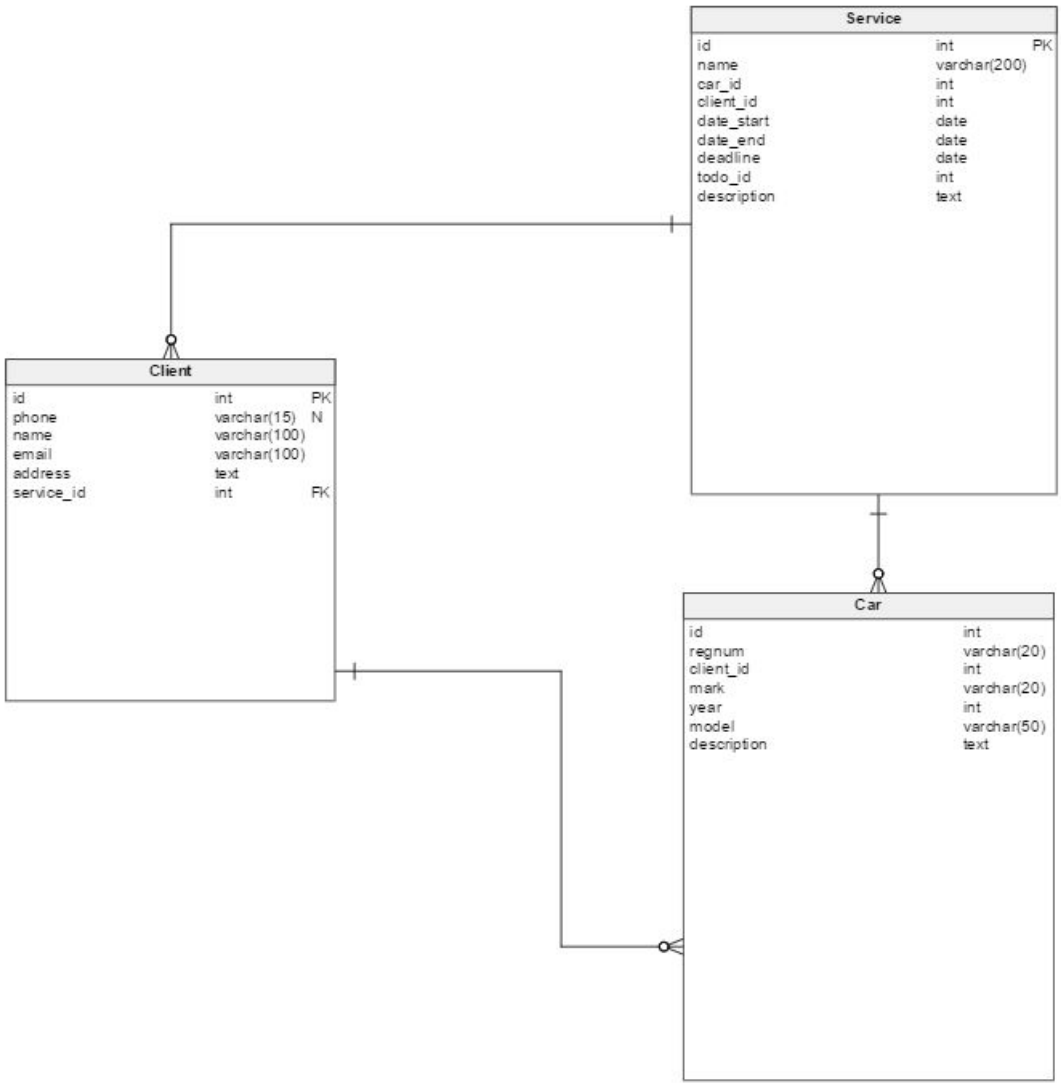
Autobaas (Joonis 6) hakkab salvestama kõike autoga seonduvat. Mudel, aasta, mark ning registreerimisenumber peavad alati täidetud olema. Auto hakkab hoidma ka üht kliendi tabeli viidet, mis seob masina kundega. Kliendi tabelile ei hakka auto veergu looma, kuna üks kunde võib tulla ka mitme masinaga.



Joonis 6. Auto tabeli skeem

2.1.3 Teenus

Teenus (Joonis 7) ehk *service* seob kõik eelnevad komponendid omavahel kokku. Töökoja tähtsaim protsess hakkab pihta uue teenuse registreerimisega. Registreerimisel tuleb ära märkida tööga seotud auto ning klient. Teenusele olemasoleva auto valimisel ei täideta kliendi viide automaatselt, sest auto võib tuua töökotta ka mitte auto omanik. Teenusel peab olema võimalik ära märkida algus- ja lõppaeg, et seda oleks võimalik kalendris töödelda.



Joonis 7. Teenuste tabeli skeem

2.2 arendamine

Arendustööd hakkavad käima arenduskeskkonnas oleva LaraAdmini paki peale. LaraAdmini projekti ülesseadistamisega tuli kaasa erinevaid standardelemente ja -mooduleid, mida käesolevas projektis pole vaja. Enne uute moodulite loomist ja arendamist tuleks ära kustutada ebavajalikud elemendid. Samuti tuleks kustutada kõik viited nendele elementidele, mis automaatselt loodi andmebaasidesse ja klassidesse. Tabelites "modules", "module_fields" ning erinevates kontroller klassides on seosed, mille kustutamata jätmisel

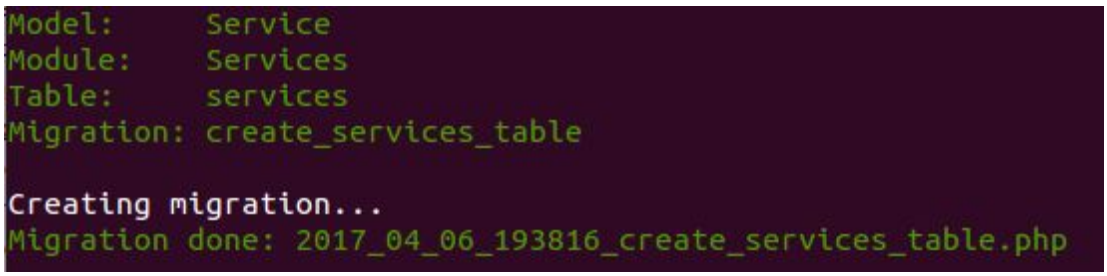
võib rakendus katki minna. Näiteks tuleks ära kustutada organisatsiooni moodule ja kõik viited sellele moodulile. Alles peale elementide kustutamist on võimalik alustada uute moodulite loomisega.

2.2.1 Moodulite loomine

Laravelis on uute moodulite loomine tänu Artisan käsurea tööriistale suhteliselt lihtsaks tehtud. Standardina on Artisanil oma komplekt käske, mis täidavad mingit kindlat funktsiooni, aga LaraAdmin installatsiooniga tuli kaasa uus grupp käske. Moodulid luuakse tabelite põhjal ning tabelid saab luua läbi migratsioonide. Migratsioon on php klassi fail, mis defineerib ära, milline hakkab tabel välja nägema. Uue migratsiooni loomiseks tuleb jooksutada projekti kaustas käsurea tööriistal käsk:

```
php artisan la:migration Services
```

Selle õnnestumise korral kuvab käsurea tööriist õnnestumise (Joonis 8) teate ning luuakse uus fail projekti "/Database/migrations" kausta.



Joonis 8. Migratsiooni loomise õnnestumine

Migratsioonis tuleb ära määrata LaraAdmin pakile omased andmetüübid (Lisa. tabel 1), et tabeli loomisel tekiksid õiged andmetüübid ning mooduli vaated oskaksid neid õigesti kuvada. Standard Laravel andmetüüpide kasutamisel tekivad elemendid, mis ei ole seotud LaraAdminiga ning selle tulemusena tulevad administreerimise paneeli osas vead. Teenuse tabeli loomiseks pole vaja migratsiooni failis ID andmetüüpi märkida, kuna see luuakse automaatselt migratsioon sisestamisel. Eelnevalt loodud teenuse tabeli skeemi järgi peaks migratsiooni faili sisu välja nägema järgnevalt:

```
Module::generate("Services", 'services', 'name', [
["name", "Service Name", "Name", false, "", 6, 256, true],

["client", "Client", "Dropdown", false, 3, 0, 0, false, "@cars"],
["car", "Car", "Dropdown", false, 3, 0, 0, false, "@cars"],
["date_start", "Töö algus", "Datetime", false, "", 0, 0, true],
["date_end", "Töö lõpp", "Datetime", false, "", 0, 0, true],
["date_deadline", "Töö tähtaeg", "Date", false, "", 0, 0, false],
["description", "Description", "Textarea", false, "", 0, 1000,
false]
["todo", "Todo", "Dropdown", false, 3, 0, 0, false, "@todolist"],
]);
```
Koodinäide 1. Teenuse migratsiooni faili sisu

Kui migratsiooni fail on valmis, tuleb jooksutada käsk:

```
php artisan migrate
```

Migrate käsu õnnestumisel on võimalik genereerida CRUD elemendid, andmete kuvamiseks ning töötlemiseks. Selleks peab käsureatööriistal projekti kaustas jooksutama käsu:

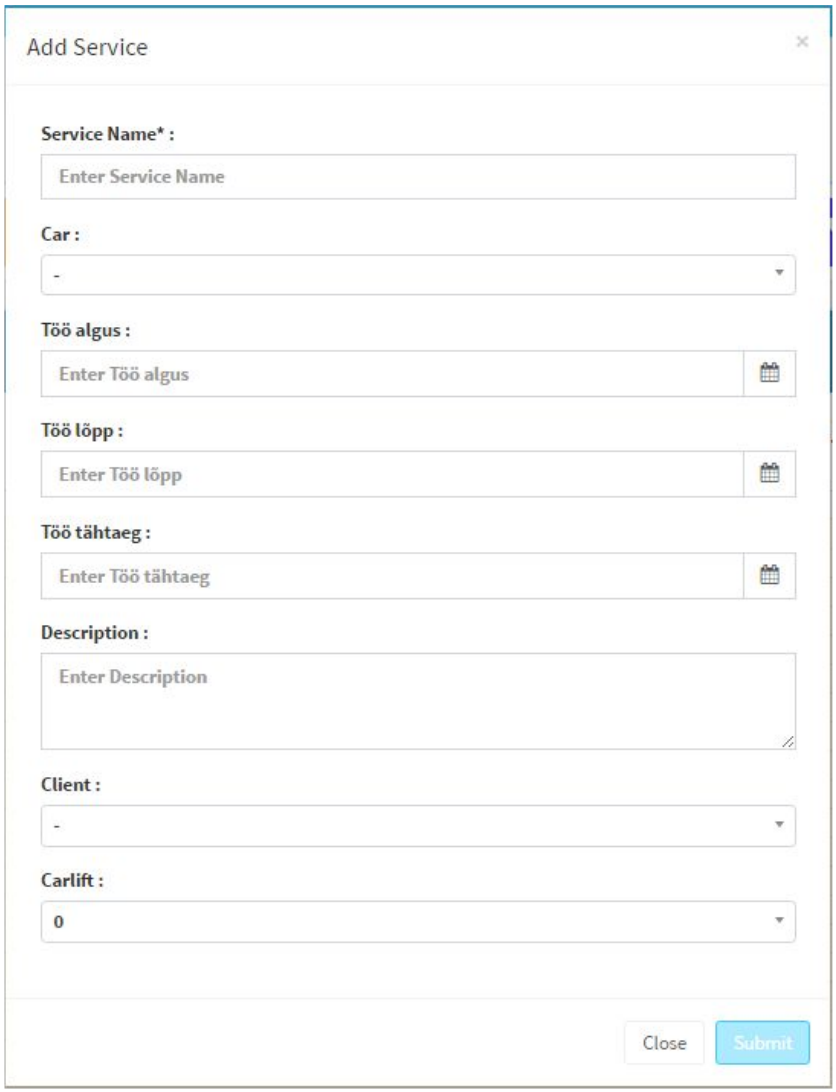
```
php artisan la:crud Services
```

Käsu õnnestumise korral luuakse administreerimise keskkonda uus moodul ning andmebaasi tabelites luuakse vastavat viited ja tabelid selle kohta. Projektis luuakse vaated "../resources/views/la/services" kausta, kus on mitu erinevat "blade.php" faili, mis täidavad erinevaid funktsioone.

2.2.2 Teenus mooduli vaade

Uued mooduli vaated luuakse LaraAdmin pakis olevate standardmallide järgi. Esialgu sisaldab index vaade ainult tabelit, mis kuvab andmeid. Andmete salvestamiseks on

Bootstrap[10] modal, mille väljad on loodud tabeli veergude põhjal (Joonis 9). Lisaks on veel *show* ja *edit* vaated, mis kuvavad ühe valitud teenuse andmeid.



Joonis 9. Automaatselt loodud Bootstrap modal vorm teenusele

Teenuse *index* vaade on juhatajale kõige tähtsam ning siin peaks toimuma enamus ettevõtte töö haldamisest. Olemasolevatele elementidele tuleks juurde lisada kalender, kus on võimalik teenuseid kiiresti päevade vahel ringi tõsta ning kalender hakkab näitama nädala vaates vabasid aegu. Selle implementeerimine on lihtne, kuna LaraAdmin pakiga on juures FullCalendar javascript teek, mis hakkab täitma kalendri rolli. Kalender hakkab standardina olema päeva vaates ning võimalik on kuvada ainult päeva (Joonis 10) ja nädala (Joonis 11)

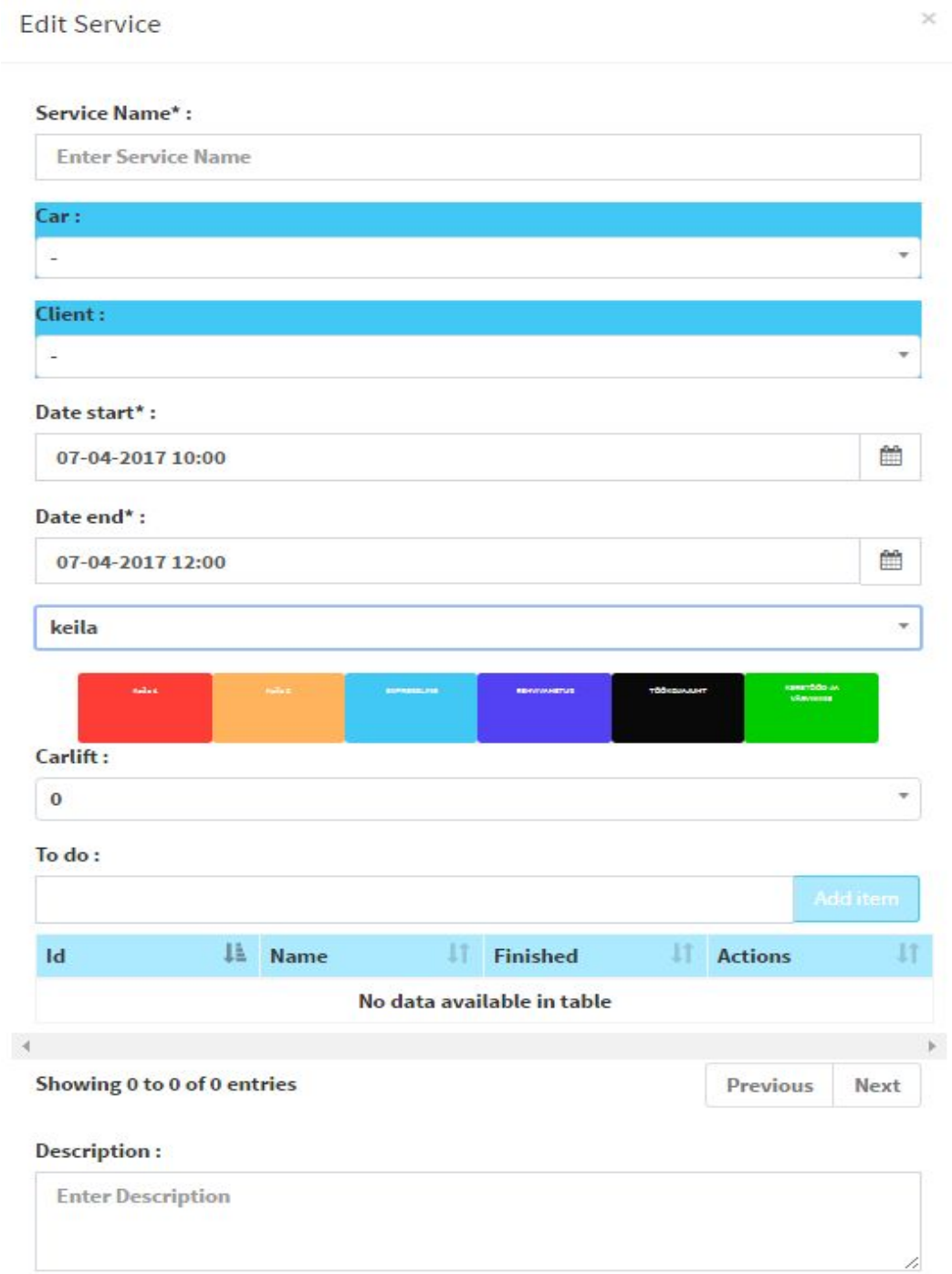
[10] http://getbootstrap.com/

vaadete vahel. Seda sellepärast, et nädala ja päeva kalendri objekte saab lihtsalt ringi tõsta, aga kuu vaates ei panda objektide ringi tõstmisel objektidele aegu külge.

Joonis 10. Kalendri päeva vaade

Joonis 11. Kalendri nädala vaade

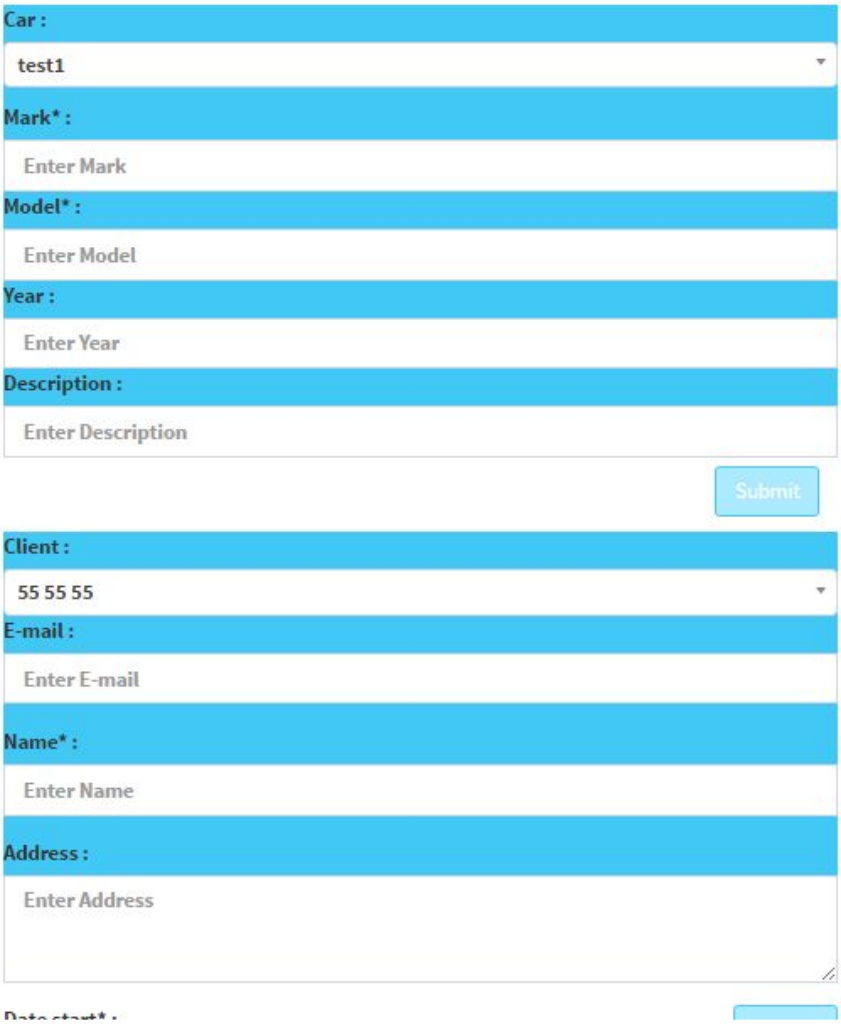
Enne kalendri ja tõstuki lisamist tuleks muuta olemasolevat teenuse lisamise *modalit* ning luua üks teenuse muutmise (Joonis 12) *modal*. Teenuse *modal* peaks töötama võimalikult sujuvalt ja kiiresti. Vähemalt minutiga peaks olema võimalik salvestada teenust nii, et kõik kliendi ja auto andmed oleksid täidetud ja määratud mingile tõstukile.



Joonis 12. Teenuse redigeerimise modal

Uue kliendi ja auto lisamisel hakkab *modal* kontrollima, kas tegemist on uue auto või kliendiga. Uue kirje puhul avaneb vormil lisasektsioon (Joonis 13), mis võimaldab kliendi ja auto andmete täpsustamist. Vigaste andmete vältimiseks tuleb kaasa ka "Lisa" nupp, mida peab vajutama, et luua uued kirjed. See väldib automaatset lisamist, kuna sellisel juhul võib tekkida olukord, kus automaatselt sisestatud andmed võivad olla pisikese trükiveaga.

Tõstukite valimiseks tuleb rida nuppe, millest on kiire ülevaade nii et ei pea ripploenditega tegelema.



Joonis 13. Teenuse auto ja kliendi lisasektsioon

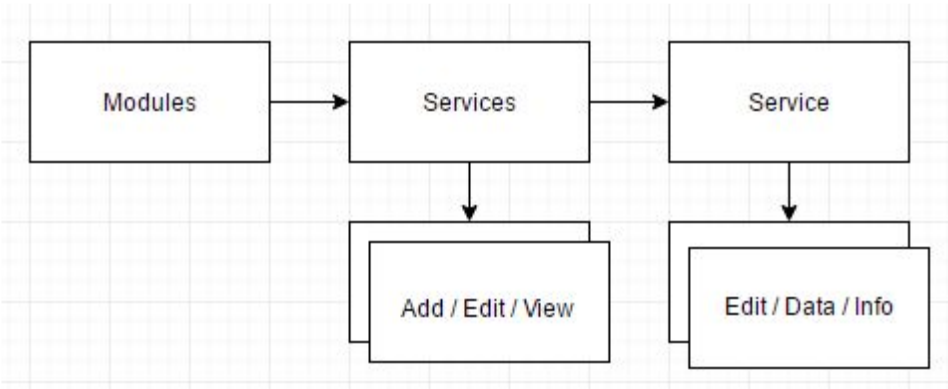
Kalendri kuvamiseks on vaja vaates luua viide projekti "/plugin" kausta, mis sisaldab FullCalendar faile. Selle rea lisamisel on võimalik initsialiseerida uus kalendri objekt ning märkida tuleb sellega kaasnevad meetodid. Kontroller klassi tuleb luua meetod, mis hakkab tagastama teenuse kirjeid ning FullCalendaris tuleb *events* meetodis luua AJAX viide kontrolleri klassile.

2.2.3 Struktuur

Laraveli projekti loomisel on kõik kompnendid loogiliselt ära jaotatud erinevatesse kaustadesse. Mudelid, vaated, kontrollerid ja muud on üksteisest eraldatud, aga projekti struktuur on suhteliselt mahukas (Joonis 14). Lisaks LaraAdmin paki installeerimisega tekkis juurde paljudesse kaustadesse "LA" kaust. "LA" kaustas on kõik LaraAdminiga seotud failid.

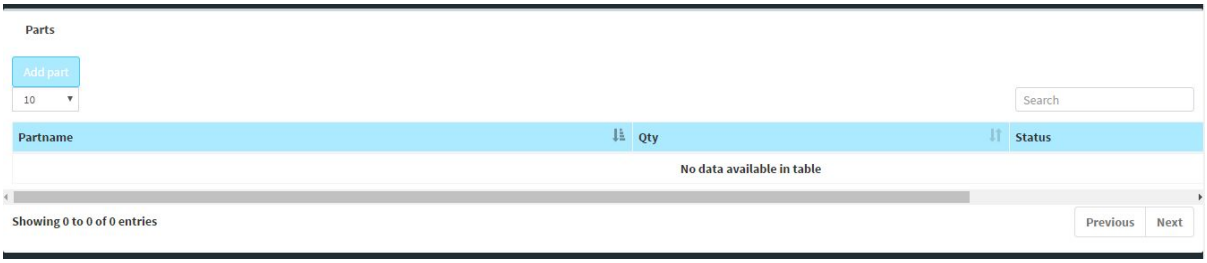
Joonis 14. Projekti struktuur

Moodulite loomisel tekib uus menüüpunkt külgreale. Enamus Kasutajalood saavad alguse külgreal olevatest moodulitest. Struktuur (Joonis 15) on kõigil moodulitel alati sarnase ülesehitusega välja arvatud teenuse moodulil, mis hakkab võimaldama ka kiiret navigeerimist erinevate moodulite *edit* ja *show* vaadetesse.



Joonis 15. Teenuse struktuur

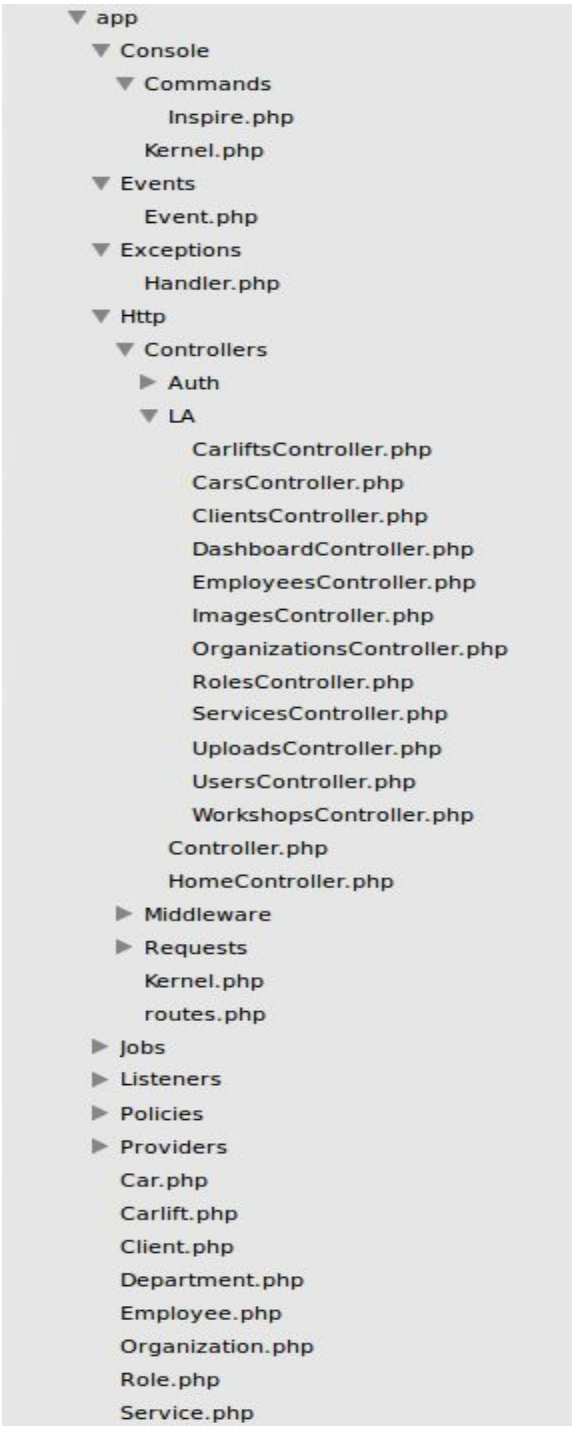
Suur osa kirja pandud kasutajalugudest läbivad teenuse moodulit ning see peab võimaldama väga paljute andmete lisamist ja töötlemist nii mehaaniku vaates kui ka juhataja vaates. Näiteks täiendjuppide lisamine teenusele hakkab olema teenuse *show* vaates. Kasutaja läheb teenuse moodulisse, valib tabelist teenuse ning läheb selle teenuse lehele. Seal on olemas eraldi sektsioon (Joonis 16), kus on võimalik lisada uusi juppe ning kustutada ja muuta olemasolevaid.



Joonis 16. Lisajuppide sektsioon teenuse *show* vaates

Kõik muu teenusega seonduv luuakse sarnaste sektsioonidega samasse vaatesse. Klientide ja autode moodulid hakkavad rakendama sarnast struktuuri, ainult et moodulite vaated on pisut erinevad.

Tähtsamad php failid asuvad projekti “app” kaustas (Joonis 17). Mudelid, millega on võimalik andmebaasi tabelitega suhelda. Kontroller klassid on “\http\controllers” kaustas ning enamus kood andmete pärimiseks ning kuvamiseks läheb just nendesse klassidesse. Kontrollerites olevaid meetodeid saab käivitada vaadetes AJAX päringutega, aga et Laravel oskaks meetoditele viidata, peab “routes.php” failis olema url rida, mis viitab kindlale kontrolleri meetodile. ”Routes.php” fail asub “http” kaustas.



Joonis 17. “App” kausta sisu

Vaated, mida kasutajatele kuvatakse asuvad projekti “resources” kaustas. “Resources” kaustas on eraldi alamkaust “views”, kus on kõik rakendusega seotud vaated (Joonis 18). Siia tulevad kõik mooduli vaated ning nende vaadetega seotud komponendid.

Joonis 18. Resources kausta sisu

2.2.4 Teavitused reaalajas

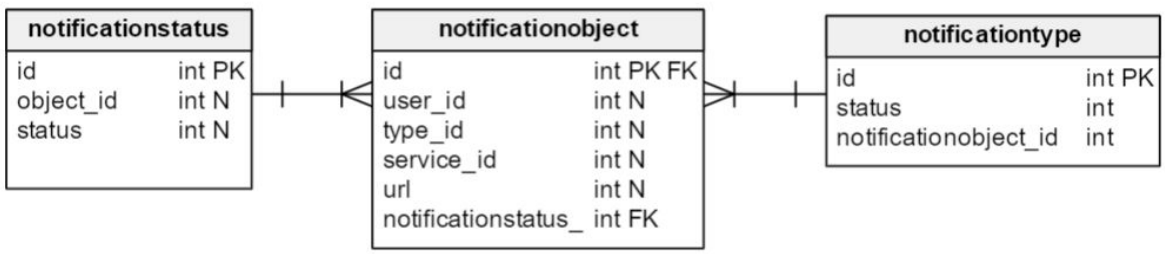
Kuna teenuse ridu hakkab tekkima väga palju, siis juhataja ja mehaanikud ei jõua alati kõiki teenuseid jälgida, et näha kas on toimunud muudatused. Töökoja personal kirjutab küsimusi või märgib muid vajalikke asju teenustele. Selleks, et teised oleksid võimelised muudatusele võimalikult kiiresti reageerima, tuleks kasutajaid teavitada. E-mailide või SMS puhul võivad

teavitused jääda märkamatuks või neid hakkab nii palju tekkima, et hakkavad kasutajaid häirima. Laraveli raamistikul on olemas Pusher bridge[11], millega on võimalik läbi kontroller klasside teavitusi saata. Laravel Pusherit[12] saab väga lihtsalt läbi Composer tööriista installeerida.

Pusher on veebiteenus, mis võimaldab rakendustel andmeid reaalajas kuvada. Pusher töötab vahekihina serveri ja kliendi vahel, mis registreerib tegevusi ja hoiab kliendiga ühendust üle *WebSocketi*(Pusher Community, kuupäev puudub). Kui serveril on uusi andmeid kuvada, siis läbi pusheri saab need kliendile saata ilma, et klient peaks lehte uuendama.

Projekti Pusher teenuse lisamisel tuleks Pusheri veebilehel kasutaja registreerida ning uus kanal luua, mis hakkab rakenduse tegevusi kuulama. Kuna kasutajad ei ole alati CRM rakendusse sisse logitud, siis tuleks teated kuskile kasutajapõhiselt ära salvestada. Teateid hakkavad salvestama 3 tabelit (joonis 19):

1. Notificationobject – hakkab salvestama teateid. Hoiab infot, mis teenuse kohta teade on loodud ning seoses notificationtype tabeliga
2. Notficationtype – hoiab erinevate teadete tüüpe
3. Notificationstatus – seoses notificationobject tabeliga. Loob igale kasutajale uued read notificationobjectist, et kõik kasutajad näeksid, mis muudatused on toimunud. Otsib kõik lugemata read ning tagastab kõik teised väärtused



Joonis 19. Teavitus kirjete tabelid

[11] https://github.com/vinkla/laravel-pusher
[12] https://pusher.com/

Tabeleid ei tohiks läbi migratsioonide luua, pigem tuleks need käsitsi juurde teha. Serveripoolsete teadete saatmiseks tuleks Laravel projekti ".env" konfiguratsiooni failis ära täita Pusheriga seotud muutujad. Kui neid ei ole kohe ".env" failis olemas, siis tuleks need lisada. Selleks on 3 rida:

- PUSHER_APP_ID
- PUSHER_SECRET
- PUSHER_KEY

Kõik väärtused saab Pusher veebikeskkonnas olevate kanalite lehelt.

Et serveri poolt teateid saata, tuleb Pusher objekt luua konfiguratsiooni failis olevate parameetritega (koodinäide 2). Objektil on *trigger* meetod, mis saadab teavitused edasi (koodinäide 3).

```
private function initPusher(){
        $options = array(
            'cluster' => 'eu',
            'encrypted' => true
        );
        $pusher = new \Pusher(
            $app_id,
            $secret,
            $pusher_key,
            $options
        );
```

```
        return $pusher;

    }
$this->initPusher()->trigger('test_channel',        'my_event',
[Auth::user()->id, $service, '2']);
```

Koodinäide 1. Klassis Pusheri initsialiseerimine

Kliendis teadete kätte saamiseks tuleb “app.blade.php” faili Pusher teek ära märkida ning lisama koodi, mis võtaks muudatused vastu ja käituks vastavalt vajadustele.

```
<script
src="https://js.pusher.com/3.2/pusher.min.js"></script>
var pusher = new Pusher('{pusher_id}', {

              cluster: 'eu',

              encrypted: true

            });



            var channel = pusher.subscribe('test_channel');

            channel.bind('my_event', function(data) {

                if(data[0] != '{{ Auth::user()->id }}'){

                //createNotificationFromPusher(data);

                    $('#notification-menu').empty();

                    getNotificationList();

                }

            });
```

Koodinäide 2. Vaates teadete püüdmine

3 Esimene tootmise keskkonna versioon

Esimese versiooni tingimuste täitmisel on võimalik arendus tõsta tootmise keskkonda. Käesolevas peatükis on välja toodud etapid tootmise keskkonna loomisel, testimise tulemused ning vead, mis esinesid rakenduses.

3.1 Tootmiskeskkonda tõstmine

Tootmise keskkond tehakse AWS keskkonna teisele EC2 instanti peale, mis hakkab andmeid salvestama uude S3 andmebaasi. Enne projetki tõmbamist versioonihalduse rakendusest tuleks luua tühi LaraAdmin projekt. Samuti tuleks kustutada ebavajalikud elemendid, et projekti kaust ja andmebaasid oleksid sarnased neile, mis olid arenduskeskkonnas enne arendutööde alustamist. Konfiguratsiooni faile versioonihaldus ei salvesta ning need tuleb ka kõik uuesti ära täita. Ebavajalike elementide mitte eemaldamisel võivad tekkida olukorrad, kus andmebaaside ID'd ja seosed ei klapi.

Pusheri jaoks tuleks luua uus kanal ning kanali parameetrid peaks tootmise keskkonna konfiguratsioonis ära märkima. Pusherit uuesti läbi Composeri installeerima ei pea, kuna versioonihaldus hoiab seda repositooriumis. Lisaks sellele ka kõiki teisi komponente, mis sai juurde lisatud.

3.2 Vead

Kasutajate testimise tulemusel oli kõige suurem viga kasutaja sessiooni periood. Sessioon kestis standardina 30 minutit ning uuenes iga kord, kui kasutaja tegi mingisugust muudatust. Aja täitumisel sessioon lõppes ning kasutaja visati rakenduse keskkonnast välja ning andmeid automaatselt ei salvestatud. Iga kord kui kasutaja visati keskkonnast välja, siis ta pidi alati kasutaja login andmed täitma, et tagasi keskkonda saada. Kõige suurem probleem selle veaga oli see, et vormidel olevaid andmeid ei salvestatud. Kui juhataja täidab näiteks uue teenuse vormi ja ei salvesta seda ära enne sessiooni aja lõppemist, siis kõik vormil olev info läheb

kaduma. Infot ei ole võimalik kuidagi taastada. Selle parandamiseks tuli "app.php" konfiguratsiooni failis määrata sessiooni ajaks 24 tundi ehk 1440 minutit. Sisselogimise sai standardiks "Hoia mind sisselogituna" variant peale pandud, et kasutajad ei peaks pidevalt uuesti sisse logima.

Probleeme oli veel FullCalendar kalendris teenuste muutmisega. Kalendris on sündmused, mis luuakse teenuse andmebaasi põhjal. FullCalendar saadab päringu teenuse kontroller klassile mingis ajavahemikus ning kontroller klass saadab vastu massiivi teenustest. Probleem tekib algus- ja lõpuaja andmetega. Teenuse redigeerimise vormi avamisel täidetakse andmed automaatselt andmebaasist päritud andmetega. Redigeerimise vormil on Bootstrap Datetimepicker[13] read, mis ei vasta andmebaasist saadud formaadile. Aja samaks jätmisel ning vormi salvestamisel ei oska kontroller klass ja andmebaas aega töödelda ning see salvestatakse null väärtusena. Selle vältimiseks on kaks varianti. Üks variant oleks kontroller klassi poolt tagastatud aja väärtus panna kohe õigesse formaati. Enne massiivi tagastamist töötleb kontrolleri meetod läbi kõik massiivi lisatavad algus- ja lõppajad. Teine variant on luua javascript meetod, mis redigeerimise vormi initsialiseerimisel muudab aja õigesse formaati. Hetkel kasutab autor teist varianti, sest teenuse redigeerimisel päritakse ainult ühe teenuse andmeid ning õigesse formaati on vaja panna ainult päritava teenuse andmed.

3.3 Lisa arendused

Järgmised kaks suuremat arendustööd oleks mehaanikuvaate täiendamine ning CRM süsteemile omased funktsioonide lisamine.

Mehaanikuvaade kuvab praegu tõstukitele määratud teenused. Mehaanik saab määrata, kas teenus on tõstuki peal või mitte. Lisaks saab mehaanik lisada küsimusi, juppe ning muuta teenuse staatust. Kui mehaanik lõpetab autoga töötamise ning võtab masina tõstukilt, tuleb tal täita protokoll. Protokollis märgib mehaanik, mida autoga tehti, milliseid uusi juppe on vaja tellida, et auto saaks korda ning mis staatuses on auto. Järgmine etapp süsteemis oleks

[13] https://eonasdan.github.io/bootstrap-datetimepicker/

protsess veebikeskkonda luua ning võimaldada mehaanikul seda teha tahvelarvutil või telefonil.

CRM süsteemi tähtsamad funktsioonid, mida oleks vaja järgmisena arendada, on kliendi automaatne teavitamine, mis staatuses on auto. Lisaks sellele on vaja juhatajal lisajuppide vaadet ning automatiseerida nende tellimist või seda lihtsustada. Kui teenusel on vaja juppe, siis juhatajal oleks vaja saada kiire ülevaade, milliseid juppe on vaja tellida ning võimaldada läbi keskkonna automaatset tellimist. Sellega seoses tuleb luua varuosade tabel, mis hoiab infot lisajuppide kohta.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua autoremonditöökojale esimene versioon töötavast CRM rakendusest. Töö läbib erinevaid rakenduse arendamise protsesse ning antakse põgus ülevaade, mida töö käigus tehakse ning kasutatakse.

Esmalt sai kirja pandud, millist funktsionaalsust pidi esimene versioon võimaldama ning kuidas hakkab rakendus tulevikus töötama. Selle põhjal oli võimalik luua arendusplaan, mille põhjal hakati rakendust looma.

Arendamine hakkas pihta uute moodulite loomisega ning moodulitele andmetöötlusega seotud vormide loomisega. Vormide valmimisel sai hakata vaateid muuta. Reaalajas teavitamine lahendati Pusher komponendi lisamisega. Arenduskeskkonnas oleva rakenduse testimisega eriti vigu esile ei tulnud. Suuremad vead tulid välja tootmise keskkonnas ning need said kiirelt lahendatud.

Töö eesmärki saab lugeda saavutatuks, kuna esialgne versioon süsteemist sai tootmiskeskkonda ülesseadistatud ning töökoda hakkas seda kasutama. Lisaks on olemas juba plaanid, rakenduse edasi arendamiseks.

Töö käigus õppis autor palju uut seoses erinevate tehnoloogiatega ning erinevaid autoremonditöökoja süsteeme tundma. Samuti õppis autor oma aega paremini planeerima ja koostööd tegema. Töö pani autorit mõistma, miks on tähtis kliendi ja teenusepakkuja suhtlemine. Nimelt on võimalik väga kergesti üksteisest valesti aru saada ning sellest tingituna kannatavad kõik osapooled. Samuti sai selgeks see, et ka detailselt ette planeerides, tuleb töö käigus ikka ette üllatusi ja ootamatuid probleeme. Murekohti ei tohiks aga karta, kuna just need arendavadki inimest kõige enam.

Kasutatud kirjandus

Auto Repair Cloud (kuupäev puudub). Auto Repair Cloud features. Loetud aadressil http://www.autorepaircloud.com/#/features

Auto Repair Cloud (kuupäev puudub). Auto Repair Cloud release. Loetud aadressil http://www.autorepaircloud.com/#/release

Amazon (kuupäev puudub). Amazon S3. Loetud aadressil https://aws.amazon.com/s3/

Capterra (kuupäev puudub). Capterra Auto Repair Cloud. Loetud aadressil http://www.capterra.com/auto-repair-software/spotlight/147213/AutoRepair%20Cloud/InterTAD

Github (kuupäev puudub). Github Laravel Pusher. Loetud aadressil https://github.com/vinkla/laravel-pusher

How to geek (2014). How to geek what is Github. Loetud aadressil https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/

LaraAdmin (kuupäev puudub). LaraAdmin Migrations CRUDs. Loetud aadressil http://laraadmin.com/docs/1.0/migrations_cruds

Github (kuupäev puudub). Github What is pusher. Loetud aadressil https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html

Techtarget (kuupäev puudub). Techtarget Amazon AWS. Loetud aadressil http://whatis.techtarget.com/definition/Amazon-Web-Services-AWS

Techtarget (kuupäev puudub). Techtarget CRM. Loetud aadressil http://searchcrm.techtarget.com/definition/CRM

Techopedia (kuupäev puudub). Techopedia CRUD. Loetud aadressil https://www.techopedia.com/definition/25949/create-retrieve-update-and-delete-crud

Trello (kuupäev puudub). Trello What is Trello. Loetud aadressil http://help.trello.com/article/708-what-is-trello

E-teatmiks (kuupäev puudub). E-teatmik. Loetud aadressil http://www.vallaste.ee/

Wokshop Mate (kuupäev puudub). Workshop mate. Loetud aadressil http://www.workshopmate.com.au/

Wokshop Mate (kuupäev puudub). Workshop mate software features. Loetud aadressil http://www.workshopmate.com.au/workshop-software-features/

Wokshop Mate (kuupäev puudub). Workshop mate about us. Loetud aadressil http://www.workshopmate.com.au/about-us/

Summary

CRM Opportunities of an Autovehicle Workshop on the Example of Viruauto

The purpose of this Bachelor thesis was to develop a working CRM application for an autovehicle workshop. This thesis goes through the development cycle and gives a brief summary of the work that is being done and the programs that are used.

The work starts off with writing down the requirments for the application's first version and what it should be capable of in the future. Taking this into account, a development plan was constructed that was the basis for developing the application.

Development started with creating new modules and forms that allowed to work with databases. With the completion of the forms, the next step was to develop the application views. For realtime notifications, Pusher application was integrated into the project. Application testing in the development environment did not yield too many bugs and the bugs that did come up were small. Bigger issues came when the application was imported into the live environment, but they were easily fixed.

The purpose of this thesis can be considered completed because a working version of the CRM application is in the live environment and in use by the workshop. In addition to the application being in the live environment, there are also plans and ideas as to what is going to be done next.

During the writing of this thesis the author got a lot of experience regarding new technologies and different autovechicle administration applications. Also the author learned how to effectively plan his time and work with others. Author learned the importance of communication between the client and the service provider. Namely it is easy to miscommunicate between the parties in which case both parties suffer. It is impossible to plan everything in advance because errors and bugs come up regardless. These bugs should not be feared because they give the most experience.

Lisad

Tabel 1. LaraAdmini andmestruktuurid migratsiooni faili jaoks

UI Type (3rd)	Default Value (5th)	Min Length (6th)	Max Length (7th)	Popup Values (9th)
	In String	Integer	Integer	
Address	["address", "Address", "Address", false, "", 0, 1000, true],			
	true / false	0	0	
Checkbox	["restricted", "Restricted", "Checkbox", false, false, 0, 0, false],			
	In Decimal	Minimum Value	Maximum Value	
Currency	["price", "Price", "Currency", false, 0.0, 0, 0, true],			
	now()2016-06-23	0	0	
Date	["date_release", "Date of Release", "Date", false, "now()", 0, 0, false],			
	now()	0	0	
Datetime	["time_started", "Start Time", "Datetime", false, "now()", 0, 0, false],			
	In Decimal	Minimum Value	Maximum Value	
Decimal	["weight", "Weight", "Decimal", false, 0.0, 0, 20, true],			
	In String / Integer	0	0	Array / @table_name
	["publisher", "Publisher", "Dropdown", false, "Marvel", 0, 0, false, ["Bloomsbury","Marvel","Universal"]],			
Dropdown	["publisher", "Publisher", "Dropdown", false, 3, 0, 0, false, "@publishers"],			
	In String	Integer	Integer	
Email	["email", "Email", "Email", false, "", 0, 0, false],			
	File Path (String)	0	256	
File	["project_file", "Project File", "File", false, "project_file.pdf", 0, 256, false],			
	In Float	Minimum Value	Maximum Value	
Float	["weight", "Weight", "Float", false, 0.0, 0, 20.00, true],			

HTML	In String	0	0	
	["biography", "Biography", "HTML", false, " This is description ", 0, 0, true],			
Image	Image Path (String)	0	256	
	["profile_image", "Profile Image", "Image", false, "img_path.jpg", 0, 256, false],			
Integer	In Integer	Minimum Value	Maximum Value	
	["pages", "Pages", "Integer", false, 0, 0, 5000, false],			
Mobile	In String	0	20	
	["mobile", "Mobile", "Mobile", false, "+91 8888888888", 0, 20, false],			
Multiselect	In String / Integer	0	Max Selections	Array / @table_name
	["media_type", "Media Type", "Multiselect", false, ["Audiobook"], 0, 0, false, ["Print","Audiobook","E-book"]],			
	["media_type", "Media Type", "Multiselect", false, [2,3], 0, 0, false, @media_types],			
Name	In String	5	256	
	["name", "Name", "Name", false, "John Doe", 5, 256, true],			
Password	In String	6	256	
	["password", "Password", "Password", false, "", 6, 256, true],			
Radio	String	0	0	
	["status", "Status", "Radio", false, "Published", 0, 0, false, ["Draft","Published","Unpublished"]],			
String	In String	0 Integer	256 Integer	
	["author", "Author", "String", false, "JRR Tolkien", 0, 256, true],			
Taginput	Array	0	0	
	["genre", "Genre", "Taginput", false, ["Fantacy","Adventure"], 0, 0, false],			
Textarea	In String	0 Integer	1000 Integer	
	["description", "Description", "Textarea", false, "", 0, 1000, false],			

	In String	5 Integer	256 Integer	
TextField	["short_intro", "Short Introduction", "TextField", false, "", 5, 256, true],			
	In String	0	Default 256	
URL	["website", "Website", "URL", false, "http://dwij.in", 0, 0, false],			