TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Krista Vaabel 164424IAPB

PARKIMISE REGISTREERIMISE RAKENDUS HOTELL L'ERMITAGE ADMINISTRAATORILTELE

Bakalaureusetöö

Juhendaja: Roost, Mart
MSc

Tallinn 2020

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: Krista Vaabel

27.07.2020

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks oli luua veebirakendus hotelli L'Ermitage administraatoritele külastajate parkimise registreerimiseks hotelli territooriumil oleval parkimisalal.

Rakenduse põhieesmärkideks on lihtsustada vastuvõtu administraatori tööd, grupeerides erinevad parkimise registreerimisega seotud tegevused ning võimaldades käsitsi kirjutatud kassa sissetuleku orderi asemel kliendile printida korrektsel kujul arve.

Lõputöö käigus valmis kolmekihilisel klient-server arhitektuuril põhinev rakendus, mille klientrakendus on kirjutatud Vue.js JavaScript raamistikul ning serverrakendus Javal põhineval Spring Boot raamistikul. Andmete salvestamiseks on kasutusel MySQL andmebaas.

Rakenduse kaudu saab hotelli vastuvõtu administraator luua, vaadata, muuta ja kustutada parkimisi, lisada püsiklientidega seotud andmeid ning salvestada ja printida arve inglise ja eesti keeles.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 9 peatükki, 20 joonist, 1 tabelit.

Abstract

A Parking registration application for hotel L'Ermitage receptionists

The purpose of this thesis is to create an application for the receptionists of Hotel L’Ermitage to simplify the parking registration process by grouping different steps into a single application and providing the receptionist with the option to print out a properly formatted invoice, which abides by the style rules of the hotel.

The current process includes writing a check by hand and inserting the parking details into an Excel table. The latter is frequently forgotten during a busy day, where many clients pass through the reception.

The outcome of the thesis is a web application that uses three-tier client-server architecture. The client application is single page application written in Vue.js that can make HTTP requests to the server application which works as a RESTful API. The server application is written in Java using Spring framework.

The functionality of the application includes creating, reading, updating and deleting parking details, adding company details to parking information, inserting, changing and deleting details of frequent guests and auto filling a new parking form based on frequent guest details. After inserting a new parking, it is possible to generate and save an invoice in both Estonian and English language.

The thesis is in Estonian and contains 29 pages of text, 9 chapters, 20 figures, 1 tables.

Lühendite ja mõistete sõnastik

API	*Application programming interface*, rakendusliides
AWS	*Amazon Web Services,* Amazoni Veebiteenused
CORS	*Cross-origin resource sharing*, erinevate domeenide vaheline ressursside jagamine
DI	*Dependency Injection*, sõltuvussüst
DOM	*Document Object Model*, dokumendi objektimudel
DTO	*Data Transfer Object*, andmesaateobjekt
HTML	*Hypertext Markup Language*, keel veebilehtede loomiseks
HTTP	*Hypertext Transfer Protocol*, protokoll andmete edastamiseks arvutivõrkudes
IoC	*Inversion of Control*
JPA	*Java Persistance API*, relatsioonilises andmebaasis olevate Java objektide haldamise spetsifikatsioon või standard
JSON	*JavaScript Object Notation*, tekstiformaat andmete edastamiseks
JWT	*JSON Web Token*, räsitud pääsumandaat
REST	*Representational State Transfer*, veebiteenuste arhitektuuri stiil
Sass	*Syntactically awesome style sheets*, stiililehe keel veebilehtede kujundamiseks
SPA	*Single Page Application*, üheleheline rakendus

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva bakalaureusetöö eesmärgiks on luua rakendus hotelli L'Ermitage vastuvõtu administraatoritele parkimiste üles märkimiseks. Idee rakenduse arendamiseks tuli autori arutelust teiste administraatoritega, töötades antud ametikohal 2018 aasta augustist kuni 2020 aasta maini.

Turismihooajal käib hotellist läbi palju kliente, kellest mitmed kasutavad võimalust jätta oma sõiduk hotelli kõrval asuvasse väliparklasse või hotelli all olevasse garaaži. Tasuliselt parkivaid kliente leidub nii tavaklientide seas, kes tulevad perega puhkama või romantilisele väljasõidule, kui ka äriklientide seas. Hotelli restorani ning sauna külastajad pargivad üldjuhul tasuta ja vaid mõne tunni.

Vastuvõtu administraatori ülesanne on sisestada parkimise andmed Exceli tabelisse ning väljastada teenuse eest tasunud kliendile arve. Üks kõige olulisemaid probleeme, mida antud töö lahendada üritab on arve genereerimise võimalus. Praeguse lahendusega väljastatakse arve kassa sissetuleku orderina, mis on kirjutatud administraatori poolt käsitsi. Kõikide administraatorite käekiri on erinev ning halvema käekirja puhul näeb kassa sissetuleku order välja ebaprofessionaalne. Tavaklientidel ei ole üldjuhul arvega probleeme, kuid äriklientidele ei ole antud lahendus sobiv kuna nad peavad majutuse ning parkimise arve edastama oma firma raamatupidamisele.

Kiirel päeval on parkimiste sisestamine Exceli tabelisse administraatori jaoks pigem teisejärguline ja võib kergelt ununeda. Arendatav rakendus koondab parkimise sisestamise ning arve printimise. Kui arve genereerimiseks peavad olema andmed juba sisestatud, siis ei ole võimalust, et parkimise registreerimine ununeb.

Antud töö teine peatükk tutvustab hotelli ning seletab lahti hetkel kasutusel oleva parkimise registreerimise protsessi vastuvõtu administraatori vaatenurgast. Kolmas ja neljas peatükk kirjeldavad rakenduse arendamisel kasutatavaid tarkvara arendamise arhitektuuri- ja disainimustreid ning tehnoloogiaid. Viies peatükk toob välja rakenduse arendamise üldise protsessi ja kasutajalood ning kuues peatükk kirjeldab arendatava

rakenduse koodi struktuuri nii klient- kui serverrakendusel ja annab ülevaate rakenduse põhifunktsionaalsusest.

Rakenduse arendamise protsess jaotub kolme etappi. Esimese osana pannakse paika rakenduse nõuded, kirjutatakse kasutajalood ja tehakse prototüüp, et kinnitada rakenduse põhifunktsionaalsus ja töövoog hotelli vastuvõtujuhiga enne arendusprotsessi algust.

Teine etapp on rakenduse arendamine. Klientrakendus arendatakse kasutades Vue.js raamistikku ning populaarseid teeke vormide valideerimiseks, välimuse loomiseks, navigeerimiseks ja oleku manageerimiseks. Serverrakenduse loomisel on kasutusel Javal põhinev Spring Boot raamistik. Andmed salvestatakse MySQL andmebaasi.

Rakendusprotsessi viimaseks etapiks on rakenduse ülesseadmine Amazoni veebiteenustesse, et hotelli vastuvõtu administraatorid saaksid rakendust testida ning anda tagasisidet funktsionaalsuse, töövoo, kasutajamugavuse ja välimuse kohta.

2 Parkimise registreerimine hotellis L’Ermitage

Hotell L’Ermitage on neljatärnihotell Tallinnas aadressil Toompuiestee 19. Hotell avati 4. mail 2004. a 91 toaga [1] ning aastal 2014 laiendati hoonet uute tubade ning konverentsiruumiga. Hetkel on hotellis 122 tuba ning kaheksa erinevat toatüüpi – Ühekohaline, kahekohaline, standard premium, äriklassi, superior, superior romantika, superior saunaga ning superior peretuba. Lisaks majutusele ja konverentsiruumile pakub hotell toitlustust restoranis L’Ermitage ning restoranis Katze. Klientidel on võimalik parkida tasulises väliparklas ja garaažis. Parkimise hind sõltub teenuse vahendajaga kokku lepitud hindadest ning hotelli kodulehel olevatest eripakkumistest. Samuti on hotellil püsikliendiprogramm, millega vähemalt 20 ööbimisega klientidele rakendub lisaboonusena tasuta parkimine [2].

2.1 Parkimise registreerimise protsess

Hotelli väliparkla on varustatud tõkkepuuga, et takistada tundmatute autode sisenemine parkla territooriumile. Parkimise õiguse tõendamiseks on vajalik parkimispileti (Vt. Lisa 2) olemasolu autos nähtaval kohal. Parkimispiletil on käsitsi kirjutatud parkimisõiguse algus- ja lõppkuupäev ning tembeldatud parkimispileti number. Administraatori kohustus on üles märkida kõik päeva jooksul toimunud parkimised. Senini on märkimiseks kasutatud Exceli tabelit, mis on varustatud järgmiste tulpadega: parkimispileti number, kliendi nimi, parkimise periood, auto number, parkimise hind ja makseinfo.

Üldjuhul soovivad majutuvad külastajad parkida auto enne tuppa sisseregistreerimist. Administraator kontrollib dokumendi alusel külastaja identiteeti ja broneeringu olemasolu ning väljastab parkimispileti. Kui parkida soovivaid kliente on mitu, siis võib administraatoril ununeda märkida, millised külastajad parkima saadeti.

See tähendab, et ununenud parkimise informatsioon ei jõua Exceli tabelisse ning teenuse eest jääb tasumata.

Antud töös arendatav rakendus tagab selle, et nii parkimise registreerimine, arve printimine kui ka ülevaade kõikidest parkimistest on ühes kohas, mis vähendab võimalust, et parkimise eest tasu küsimine või parkimise märkimine ununeb.

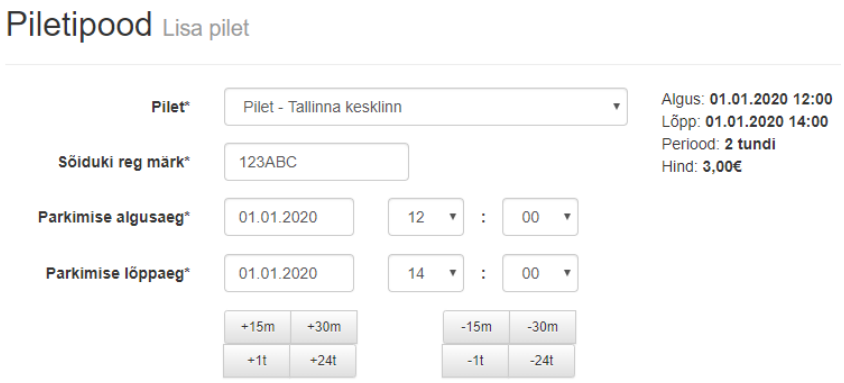
Hotelli kõrval olev väliparkla ala ei kuulu hotellile, seega soovib juhtkond lihtsustada raamatupidaja tööd, eraldades parkimisega seotud tehingud ülejäänust hotelli raamatupidamisest. Hotelliprogrammis parkimise postitamisel kajastub see automaatselt arveldamises. Juhtkonna käsul on hotelliprogrammis lubatud parkimisi postitada vaid siis, kui äriklient soovib firma detailidega arvet (Vt. hotelliprogrammis genereeritud arvet Lisas 1). Muul juhul kirjutatakse arve käsitsi kassa sissetuleku orderina (Vt. Lisa 2).

Arendatav rakendus võimaldab salvestada firma detailid parkimise info juurde ning kuvada seda arvel koos Hotel L'Ermitage OÜ ja Parkovka OÜ andmetega. See kõrvaldab vajaduse postitada parkimisi hotelliprogrammi kaudu toale, kuid klient saab korrektse prinditud arve kõikide vajalike andmetega.

2.2 Alternatiivsed rakendused, mida hotellil oleks võimalik kasutada

Hetkel on saadaval äriklientidele kaks rakendust. EuroPark parkimisprogramm teenindusasutustele ning Parkimine.ee e-Pilet ettevõtjatele.

Parkimine.ee spetsiaalse veebikeskkonna (Vt. Joonis 1) kaudu on võimalik ettevõtjal kliendile müüa parkimispilet linnaparkimisalale või AS-i Ühisteenused väliparklasse. Piletit müüv töötaja peab parkimise registreerimisel märkima soovitud parkla või tsooni, sõiduki numbrimärgi ja parkimise perioodi. [3]



Joonis 1. Parkimine.ee veebikeskkonnas parkimise registreerimise vorm

Parkimine.ee e-Pileti tugev külg on see, et klient peab parkimise soovist vaid teenindajale teada andma ning ei pea ise otseselt protsessi sekkuma. Elektroonilise parkimispileti registreerimine käib kiiresti ja külastaja ei otsima, millised parklad, hinnad ja rakendused üldse eksisteerivad. Samuti saavad kliendid maksta teenuse eest otse hotelli, mis on sobiv kui kliendil ei ole võimalik näiteks mobiilimakseid kasutada [3].

EuroPark pakub samuti spetsiaalset parkimisprogrammi teenindusasutustele. EuroPargi tugevuseks on täislahenduse pakkumine, võimaldades lisaks parkimisprogrammile soetada parkimistehnikat ning saada nõu parklas liikluskorralduse parandamise jms osas [4].

Nii EuroPark kui Parkimine.ee rakenduse kasutamiseks on vaja sõlmida leping vastava teenusepakkujaga. Parkimine.ee rakenduse eesmärk on lubada teenindusasutusel osta elektrooniline parkimispilet kliendi eest, kliendi soovitud parkimisalale [3]. EuroPark rakendus põhineb pigem iseteenindus- ning makseautomaadi kasutamisel [4].

Antud töös arendatava rakenduse positiivseks küljeks on kolmandate osapooltega lepingu sõlmimise ebavajalikkus. Lisaks sellele ei paku ülalnimetatud rakendused võimalust parkimise arvele lisada kliendile vajalikke firma detaile ning erinevaid tasuta- ja ühepäevase parkimise põhjuseid.

Nii EuroPark kui Parkimine.ee rakenduse puhul on oluline täpselt märkida parkimise algus- ja lõppkuupäev kellaajalise täpsusega. Hotelli prioriteet on tagada külastaja mugavus, seetõttu ei ole määratud, et klient peab parkimise lõppkuupäeval lahkuma kindlal ajal vaid külastaja saab parkimiskoha vabastada vastavalt oma reisiplaanidele.

3 Rakenduses kasutatud mustrid

Antud peatükis kirjeldab autor rakenduse arendamisel lähtutud arhitektuuri- ning disainimustreid.

3.1 Kolmekihiline arhitektuur

Kolmekihiline tarkvara arhitektuur on klient-server arhitektuur, mis eraldab esitlus-, äriloogika- ja andmekihi [5].

Klient-server arhitektuuri puhul käitub server ressursside ning teenuste pakkujana ning klient ressursside tarbijana. Klient ja server vahetavad andmeid üle võrgu ning ühe serveriga võib olla ühendatud mitu klienti, kes omakorda võivad olla ühenduses ka mitme serveriga [6].

Esitluskiht on veebibrauseris või veebipõhises rakenduses töötav graafiline kasutajaliides. Klientrakendus suhtleb serverrakendusega rakendusliidese ehk API (*application programming interface*) kaudu. Äriloogikakihiks võib lugeda serverrakendust, mis sisaldab kogu rakenduse äriloogikat ja suhtleb andmebaasiga. Andmekihiks on andmebaasisüsteem [5].

Kolmekihilise arhitektuuri tugevaks küljeks on modulaarsus. Kõik kihid on realiseeritud erinevate rakendustena, mis võimaldab vahetada välja terve kihi ilma teisi kihte mõjutamata. Kasutajaliidest uuendades ei muutu äriloogikaga ega andmebaasiga seotud funktsionaalsus, mis võimaldab kasutajaliidest uuendada serverrakendust muutmata. Lisaks eelnimetatule on lihtsam rakendust skaleerida, kuna skaleerida saab vastavalt vajadusele kas klient- või serverrakendust [5].

Kolmekihiline arhitektuur valiti kuna see sobib muudatuste tegemiseks ilma teisi kihte mõjutamata. Suhtlus läbi rakendusliidese võimaldab hiljem luua esitluskihina töölaua rakenduse ilma, et peaks serverrakendust muutma. Antud töös on esitluskihiks Vue.js rakendus, äriloogikakihiks Spring rakendus ning andmekihiks MySQL andmebaasisüsteem.

3.2 SPA

Üheleheline rakendus ehk *Single Page Application* (SPA) on rakendus, millel on vaid üks HTML (*Hypertext Markup Language*) leht [7].

Mitmeleheküljeline rakendus (*Multi-Page Application*) küsib iga päringuga serverilt uue lehekülje, mis seejärel renderdatakse brauseris. Rakenduse kasvades suureneb päringute arv ning tõuseb koormus serverile. See põhjustab probleeme jõudlusega ning mõjutab kasutajakogemust negatiivselt [7].

SPA puhul ei laeta lehekülge uuesti vaid uuendatakse dünaamiliselt veebilehe osasid, mis muutusid, parandades kasutaja interaktsiooni kiirust ning tagades tegevusele kohese tagasiside. Ühelehelisi rakendusi on lihtsam programmeerida, siluda ning kliendi ja serveri poole eraldatus vähendab saadetava informatsiooni suurust kuna HTML asemel vahetatakse andmeid JSON (*JavaScript Object Notation*) kujul [7].

3.3 REST

REST ehk *Representational State Transfer* on süsteemide vahelist kommunikatsiooni standardiseeriv arhitektuurilaad. REST'il põhinevad süsteemid on tunnusjoontelt olekuta ning eraldavad kliendi ja serveri [8].

Klient- ja serverrakendus on mõlemad üksteisest sõltumatud osad, mille suhtluse tagamiseks tuleb määrata, millisel kujul informatsiooni edastatakse ja vastu võetakse. Kasutades REST liidest määratakse kindlad lõpp-punktid, mille pihta klient saab päringuid teha. Päringute tegemiseks saadab klient serverile HTTP (*HyperText Transfer Protocol*) päringu, mis sisaldab HTTP verbi, päist, ressursi teekonda ning vajadusel ka lasti. Server vastab päringule, märkides lasti sisu tüübi, staatuse koodi ning vajadusel lasti [8].

REST on olekuta, see tähendab, et serverit ei huvita, mis olekus on klient ning vastupidi. Mõlema poole jaoks on oluline vaid hetkel saadud informatsioon [8].

4 Kasutatud tehnoloogiad ning raamistikud

Antud peatükis kirjeldab autor rakenduse arendamisel kasutatud tehnoloogiaid ja raamistikke.

4.1 Gradle

Gradle on avatud lähtekoodiga tööriist, mis automatiseerib tarkvara ehitamist ja teisi protsesse. Gradle kasutab ehitusskripti loomisel konventsioon kui konfiguratsioon põhimõtet ning valdkonnapõhist programmerimiskeelt ehk *domain-specific language*, mis põhineb Groovy'l või Kotlinil [9].

Konventsioon kui konfiguratsioon on põhimõte, mille järgi töökeskkond pakub teatud ulatuses vaike konfiguratsiooni, mida on võimalik vastavalt vajadusele muuta ja ümber kirjutada. Vaike konfiguratsioon vähendab oluliselt arendaja poolt kirjutatava konfiguratsiooni hulka [10].

Gradle varustab kasutajat deklaratiivse programmeerimiskeele elementidega, mida kasutaja saab kasutada build.gradle ehitusskripti kokkupanemiseks vastavalt rakenduse vajadustele. Gradle lihtsustab projektis kasutatavate sõltuvuste haldamist laadides tehised automaatsel alla Maven'i või Ivy repositooriumitest [11], [12].

Antud projektis valiti kooste tööriistaks Gradle, sest Groovy'l põhinevat ehitusskripti on lihtne koostada ja lugeda ning Gradle võimaldab mugavalt hallata sõltuvusi.

4.2 Java

Java on tüübikindel objektorienteeritud programmeerimiskeel, mida saab lugeda nii kompileeritud kui interpreteeritud keeleks. Java lähtekood kompileeritakse baitkoodi ja interpreteeritakse Java virtuaalmasina poolt, mis on saadaval osana Java käivituskeskkonnast. Java käivituskeskkond tagab sõltumatuse platvormist ning sisaldab lisaks virtuaalmasinale ka tuumklasse ning tugiliideseid [13].

Töös on kasutatud 2018. aastal avaldatud Java SE 11 versiooni, mis on koos 2014 aastal avaldatud Java SE 8 versiooniga hetkel ainukesed pikaajalise toega versioonid [14].

4.3 Spring Boot

Spring on avatud lähtekoodiga Java raamistik, mille aluseks on *Dependecy Injection* (DI) ning *Inversion of Control (*IoC*)* [15]. IoC on printsiip, mille eesmärk on luua lõdvalt ühendatud klassid, mida on võimalik asendada, laiendada ning testida [16]

Spring Boot on Spring raamistiku laiendus, mis võimaldab kiirelt luua eraldiseisvaid Spring rakendusi [17].

Spring Boot üritab vähendada arendaja poolt kirjutatava konfiguratsiooni hulka pakkudes *starter* sõltuvusi ning võimalusel üritab automaatselt konfigureerida ka kolmandate osapoolte teekidega seotud seadistust [17].

Spring Boot sisaldab HTTP serverit nagu Tomcat või Jetti, mis võimaldab arendajal kohe rakenduse oma masinal tööle saada [17].

Peamiste Sõltuvustena on töös kasutusel Spring Data JPA ning Spring Security. JPA ehk *Java Persistance API* on standard Java objektide salvestamiseks ja nende lugemiseks relatsioonilisest andmebaasist. Annotatsioonid võimaldavad kiirelt konfigureerida erinevaid andmebaasiga seotud aspekte nagu näiteks olemite vahelisi suhteid ning tabelite ja veergude nimesid [18].

4.4 MySQL

MySQL on relatsiooniline avatud lähtekoodiga andmebaasihalduse süsteem [19]. Kuuludes liigituselt serveri andmebaasisüsteemi alla on MySQL sobiv olukorras, kus klient ning server on eraldatud. Serveri andmebaasisüsteemi eelis on suurte andmehulkade töödeldavus ja kasutatavus erinevatel platvormidel [20]. MySQL on relatsiooniline andmebaas ehk andmed salvestatakse eraldi tabelitesse [19].

MySQL sai valitud Springiga integreerimise lihtsuse ning kasutajasõbraliku MySQL Workbench rakenduse olemasolu tõttu.

4.5 Vue.js

Vue on progressiivne Javascript raamistik kasutajaliideste ja üheleheliste rakenduste loomiseks. Olles paindlik ja skaleeruv sobib kasutamiseks suuremate projektidega ja integreerimiseks teiste tehnoloogiatega. Raamistik kasutab HTML-i põhist malli süntaksit, mille abil toimub deklaratiivne DOMi (*Document Object Model*) renderdamine. Vue komponendid on reaktiivsed ehk isendi andmete muutudes uuendatakse vaates olevaid väärtusi koheselt [21] [22].

Peamine põhjus Vue valimiseks oli autori varasem kogemus antud raamistikuga ning autori eelistus kasutada HTML-i põhist malli süntaksi, võrreldes näiteks Reactiga, mis kasutab JSX (*JavaScript XML*) süntaksi. Vuel on väike õpikõver ning mitmeid põhjaliku dokumentatsiooniga ametlikult hallatavaid teeke.

4.5.1 Vue Router

Vue Router on ametlik Vue.js ruuter, mis lihtsusab üheleheliste rakenduste ehitamist. Komponentide vastavusse seadmine marsruutidega võimaldab luua mitmelehekülgelisele veebilehega sarnase navigeerimise ja kuvada pesastatud vaateid router-view elemendi abil. Lisaks sellele on lihtne üles seada navigatsioonitõkiseid, et piirata autentimata ja autoriseerimata kasutajate ligipääs keelatud aadressidele [23].

4.5.2 Vuex

Vuex on Vue.js ametlikult hallatav *state management* teek, mis on keskseks andmehoidlaks kõikide rakenduse komponentide jaoks. Keskse andmehoidla kasutamine on tasuv, kui mitu erinevat komponenti pöörduvad samade andmete poole [24].

4.5.3 Bootstrap-vue

Bootstrap-vue on front-end kaskaadlaadistiku ehk CSS (*Cascading Style Sheets*) ja Vue komponentide teek, mis põhineb Bootstrap 4 raamistikul [25]. Bootstrap-vue võimaldab kasutada üle 85 komponendi ning 45 plugina ning omab Bootstrap raamistikuga sarnaselt, küljenduse adaptiivsuse suurendamiseks, ruudustikupõhist veebilehtede ülesehitamist [26].

Bootrap-vue valiti kuna komponendid on lihtsalt kasutatavad ning slottide ja atribuutide abil kergelt kohandatavad vastavalt vajadusele. Bootstrapi vaiketeemat on võimalik muuta kirjutades üle Sass (*syntactically awesome style sheets*) vaikemuutujad [26].

4.5.4 Vee-validate

Vee-validate on Vue.js mallipõhine valideerimisraamistik vormide valideerimiseks. Vee-validate võimaldab kasutada eeldefineeritud ja ise loodud valideerimisreegleid kohandatud veateadetega. Valideerimist on võimalik rakendada nii tekstiväljale trükkides kui vormi esitamisel [27].

5 Rakenduse arendamine

Enne rakenduse arendamist oli esimeseks sammuks koosolek hotelli vastuvõtujuhiga. Koosoleku peamine eesmärk oli täpselt paika panna rakenduse nõutav funktsionaalsus ja arutusele tulid ka võimalikud hilisemad edasiarendused.

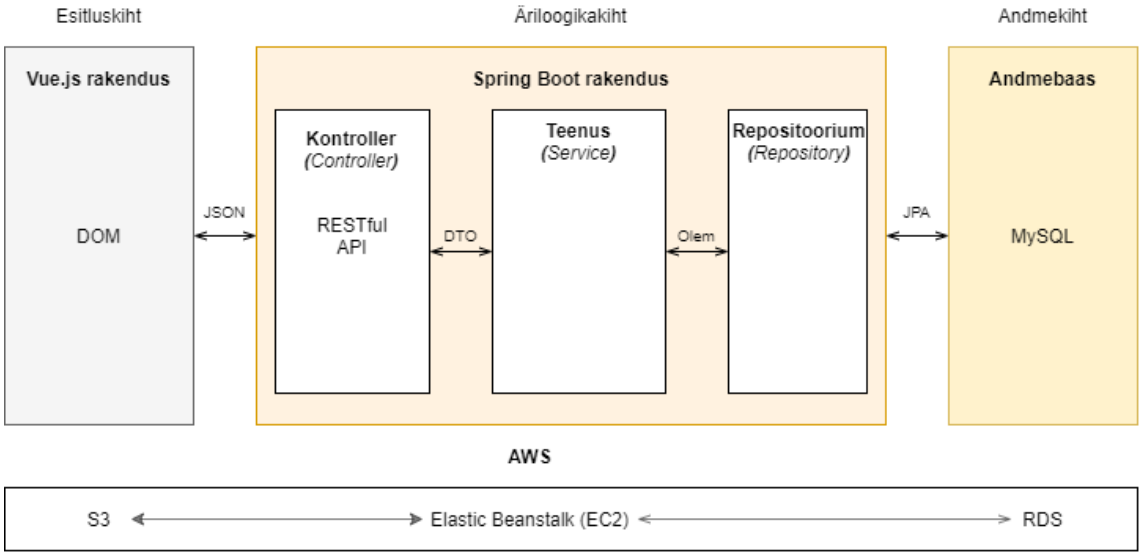
Peale funktsionaalsuse otsustamist koostati rakendusele esialgne prototüüp kasutades AdobeXD tasuta versiooni. Prototüüp loodi sisendiks edasisele arendamisele ja andmaks hotelli juhtkonnale aimu rakenduse põhivoost, et vajadusel kohe alguses muudatusi teha.

Prototüüp on saadaval aadressil: https://xd.adobe.com/view/22be9058-18cf-4466-599b-315e8db95939-548f/.

Koodi, dokumentatsiooni ja tööülesannete haldamiseks kasutati GitLab keskkonda. Koosolekul paika pandud nõuete põhjal koostati kasutajalood, mille baasil loodi GitLabis tööülesanded ehk issued. Issued olid vastavalt vajadusele kas ainult kasutajaloo back-end poolele, front-end poolele või käsitlesid nii front- kui back-endi koos. Issued implementeeriti eraldi harus ning ülesande valmimisel mestiti peaharusse.

Projekti esialgsel ülesseadmisel ja struktuuri paika panemisel kasutati autori tarkvaratehnika aine raames tiimitööna tehtud projekti, mis on avalikult saadaval Github keskkonnas [28].

Rakenduse loomisel lähtuti peatükis 3.1 tutvustatud kolmekihilise arhitektuuri põhimõtetest, eraldades klientrakenduse, serverrakenduse ning andmebaasi (Vt. Joonis 2). Sealjuures on klientrakendus realiseeritud ühelehelise rakendusena.



Joonis 2. Rakenduse üldstruktuur ja andmete liikumine

5.1 Rakenduse funktsionaalsed nõuded

Rakenduse funktsionaalsed nõuded on järgnevalt esitatud kasutajalugudena.

Vastuvõtu administraator peab saama:

- lisada uut parkimist.
- muuta eksisteeriva parkimise andmeid.
- kustutada parkimist.
- vaadata parkimise detaile.
- lisada parkimisele firma detailid.
- eksisteerivalt piletilt eemaldada firma detailid.
- printimiseks alla laadida arve.
- parkimise registreerimisel parkimispiletile kirjutatava pileti numbri
- märkida makseviisi, kuna see on vajalik arvel kuvamiseks
- märkida lõppkuupäevaks sama kuupäeva, mis on alguskuupäev, sest on kliente, kes pargivad vaid samal päeval.
- valida ühepäevase parkimise põhjuse.
- Valida tasuta parkimise põhjuse
- Lisada püsikliendi.
- Vaadata püsikliendi andmeid.
- Lisada püsikliendi profiilile kliendiga seotud firma detailid.
- Muuta püsikliendi andmeid.

- Muuta püsikliendi profiiliga seotud firma andmeid.
- Automaatselt täita püsikliendi piletiga seotud väljad.
- Lisada lõpetamata parkimise, sest kliendid eelistavad kõigepealt auto ära parkida ning alles, siis tuppa sisseregistreerimisel anda ülejäänud informatsiooni nagu auto registreerimisnumber või arvele vaja minevad firma detailid.
- Märkida parkimine kui „maksmine lahkudes“, sest vastavalt kokkuleppele kliendiga saab külastaja teenuse eest tasuda toast väljaregistreerimisel.
- Parkimise alg- ja lõppkuupäeva kalendrist valida. Alguskuupäev peab olema rakenduse avamisel automaatselt tänane kuupäev.
- Sisse ja välja logida.
- Vaadata kõiki lõpetamata, maksmata ning makstud parkimisi tabelina.
- Vaadata kõiki püsikliente tabelina.
- Filtreerida tabeleid erinevate väljade järgi

5.2 Mittefunktsionaalsed nõuded

Rakendusele seati järgnevad mittefunktsionaalsed nõuded:

- Rakendus peaks kindlasti olema kättesaadav ajavahemikus 07:00 – 23:00, kuid katkestused eelnimetatud ajavahemiku väliselt on lubatud.
- Rakendus peab töötama viimastes Mozilla Firefox ja Google Chrome brauseri versioonides.
- Rakendus peab olema turvatud.

6 Rakenduse struktuur

Rakenduse kood on jagatud kahte suurde alamkausta. Juurkaustas olev src kaust sisaldab Springiga seotud koodi ja web kaust Vue.js raamistikuga seotud koodi.

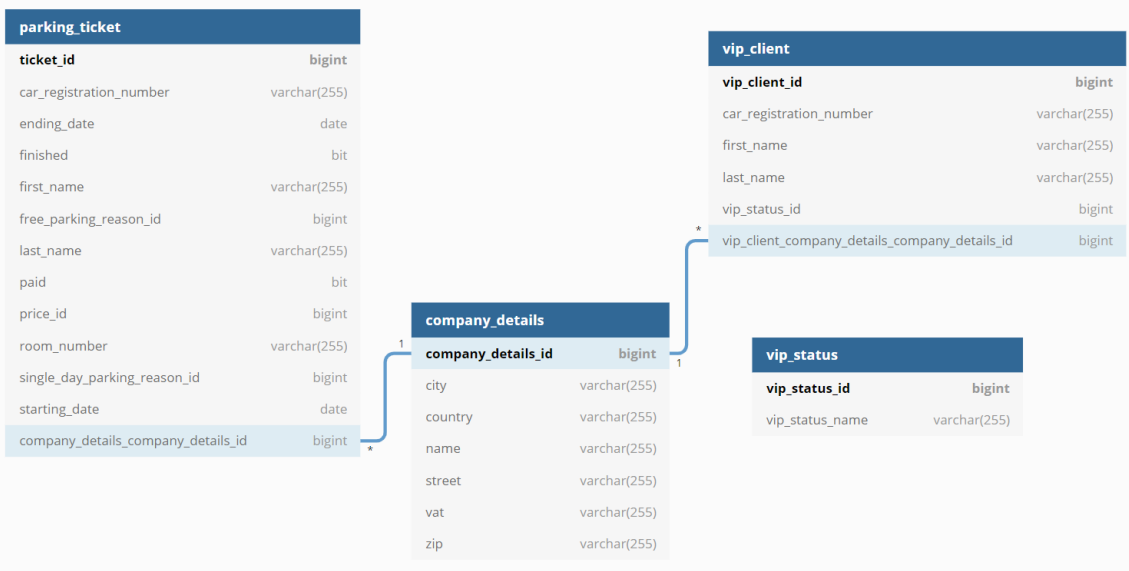
6.1 Andmebaas

Joonis 3 ja Joonis 4 annavad ülevaate Spring Data JPA poolt genereeritud MySQL andmebaasi tabelitest, nendega seonduvatest veergudest ning väljade andmetüüpidest.

`Parking_ticket` tabel sisaldab kõiki parkimisega seotud välju nagu algus- ja lõppkuupäev, kliendi ees- ja perekonnanimi, sõiduki registreerimisnumber ja toa number. Lisaks sellele on `paid` ja `finished` lipud, mis näitavad, kas klient on parkimise eest tasunud ning kas lisatud andmed on lõplikud või vajavad täiendamist.

`Company_details` tabelisse sisestatakse firma andmetega seotud read. Sama struktuuriga firma andmeid läheb vaja nii parkimiste kui püsiklientide puhul, seega salvestatakse kõik firma detailid samasse tabelisse.

Püsikliendi andmed on vajalikud uue pileti loomise vormi automaatseks täitmiseks.



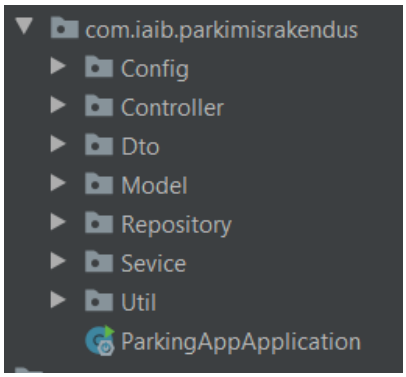
Joonis 3. Andmebaasi diagramm (1).

Püsikliendi staatused, parkimise hinnad, ühepäevaste ja tasuta parkimiste põhjused ning makseviisid on eraldi tabelites, kuna need andmed muutuvad harva., kuid kasutatakse tihti Payment tabelisse salvestatakse teenuse eest tasumisel makseinfo, mis on vajalik arve genereerimisel.

Joonis 4. Andmebaasi diagramm (2).

6.2 Serverrakendus

Rakenduse Java osa on jagatud seitsmesse alamkausta (Vt. Joonis 5).



Joonis 5. Back-end alamkausta struktuur.

Config kaust sisaldab Spring Security ja *cross-origin resource sharing* (CORS) konfiguratsiooni. Alistatud configure meetodid on kasutajadetailide ning HTTP

filtriahela seadmiseks (Vt. Joonis 6), et võimaldada kasutaja autentimine ja autoriseerimine. CORSi konfigureerimine on vajalik, et lubada erinevate domeenide vaheline ressursside saatmine.

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    CorsConfigurationSource corsConfigurationSource(){...}

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception{...}

    @Override
    protected void configure(HttpSecurity httpSecurity) throws
Exception{...}

    ...
}
```

Joonis 6. Spring Security konfiguratsiooni klass.

Model kaust sisaldab @Entity annotatsiooniga varustatud klasse (Vt. Joonis 7), mille abil JPA loob automaatselt andmebaasitabelid ning klassi muutujate põhjal lisab tabelitele veerud. Tabelis kohustuslik olev primaarvõti genereeritakse Spring Data JPAle ette antud strateegia järgi automaatselt (Vt. Joonis 7).

```
@Entity
@Data
public class CompanyDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long companyDetailsId;

    private String name;
    private String vat;
    private String country;
    private String zip;
    private String city;
    private String street;
}
```

Joonis 7.Spring Data JPA olemiklass.

Repository kaust sisaldab `JpaRepository` liidest laiendavaid liideseid (Vt. Joonis 8). Laiendades omakorda `CrudRepository` liidest on vaikimis ära kirjeldatud enamik andmebaasi põhioperatsioonides nagu loomine, lugemine, uuendamine ja kustutamine

ning kõikide ridade lugemine. Samuti on võimalik mugavalt teha spetsiifilisemaid päringuid defineerides meetodi nime korrektse malli alusel ning täpsustades tagastustüübi kas listina või objektina.

Näiteks Joonis 8 olev findByFinishedFalseOrderByTicketIdDesc meetod kontrollib `finished` lipu tõeväärtus ning tagastab id järgi kahanevalt need read, kus vastav tõeväärtus on väär.

```
public interface ParkingTicketRepository extends JpaRepository<ParkingTicket,
Long> {
    List<ParkingTicket> findByFinishedFalseOrderByTicketIdDesc();
    List<ParkingTicket> findByFinishedTrueAndPaidFalseOrderByTicketIdDesc();
    List<ParkingTicket> findByFinishedTrueAndPaidTrueOrderByTicketIdDesc();
}
```

Joonis 8. JpaRepository laiendav liides.

Dto kaust sisaldab andmesaateobjekte ehk DTOsid (*Data Transfer Object*), mida kasutatakse andmete vahetamiseks kliendi ja serveri vahel. Kliendilt tulev JSON objekt vastendatakse taustal andmesaateobjektiks ning alles pöördumisel andmebaasi poole teisendatakse DTO olemiklassiks.

Andmesaateobjektid on kasulikud, kui kliendile ei ole vaja saata kõiki andmebaasitabelis olevaid välju. Joonis 9 kirjeldab andmesaateobjekti parkimise arve andmete edastamiseks serverilt kliendile, pannes teenuse klassis kokku vajalikud väljad erinevatel olemiklassidelt.

```
@Data
public class InvoiceDetailsDto {
    private Long ticketId;
    private LocalDate startingDate;
    private LocalDate endingDate;
    private String firstName;
    private String lastName;
    private BigDecimal price;
    private CompanyDetails companyDetails;
    private String paymentName;
}
```

Joonis 9. Andmesaateobjekt arve genereerimiseks.

Util kaust sisaldab JSON Web Tokeni (JWT) loomiseks ning valideerimiseks vajalike meetoditega utiliitklassi.

Service kaust sisaldab @Service annotatsiooniga teenuse klasse (Vt. Joonis 10). @Service annotatsioon märgib klassi teenuse kihi osaks ning näitab, et antud klass sisaldab äriloogikat.

Teenuse kiht koondab endas kogu rakenduse äriloogika ning on ainukene koht andmebaasi poole pöördumiseks. Varasemas lõigus välja toodud JpaRepository laiendavad liidesed on süstitud sõltuvusena teenuse klassi kasutades @Autowired annotatsiooni ja nende kaudu saab küsida, salvestada, muuta ja kustutada andmebaasis olevaid andmeid.

Teenuse klass suhtleb lisaks andmebaasi poole pöördumise liidesele ka kontrolleri klassiga. Kontrollerist tulevad andmed andmesaateobjektidena, mis on andmebaasiga suhtlemiseks vaja teisendada olemiklassideks. Teisenduseks on kasutusel ModelMapper teek.

```
@Service
public class ParkingTicketService {

    @Autowired
    private ParkingTicketRepository parkingTicketRepository;

    @Autowired
    private ModelMapper modelMapper;

    public ParkingTicketDto createParkingTicket(ParkingTicketDto
parkingTicketDto) {
        ParkingTicket savedEntity =
parkingTicketRepository.save(convertToEntity(parkingTicketDto));
        return convertToDTO(savedEntity);
    }
    private ParkingTicket convertToEntity(ParkingTicketDto
parkingTicketDTO){
        return modelMapper.map(parkingTicketDTO, ParkingTicket.class);
    }
    ...
}
```

Joonis 10. Parkimise teenuse klass.

Controller kaust sisaldab @RestController annotatsiooniga varustatud kontrolleri klasse, mille ülesanne on suhelda kliendi poolse rakendusega (Vt. Joonis 11). @RestController annotatsioon tagab selle, et kontrolleri meetodi poolt tagastatav objekt saadetakse HTTP-taotluse vastuses.

```
@RestController
@CrossOrigin
@RequestMapping("/ticket")
public class ParkingTicketController {

    @Autowired
    private ParkingTicketService parkingTicketService;

    @PostMapping("/create")
    public ParkingTicketDto createParkingTicket(@RequestBody
ParkingTicketDto parkingTicketDTO) {
        return
parkingTicketService.createParkingTicket(parkingTicketDTO);
    }
    ...
}
```

Joonis 11. Parkimise kontrolleri klass.

Rakendusel on mitu kontrollerit, mis on igaüks seotud kindla klassiga. Järgnevas tabelis (Vt. Tabel 1) on välja toodud kõik API lõpp-punktid, mille poole klient saab HTTP päringuid teha. Selleks, et pääseda ligi lõpp-punktidele on vajalik eelnev autentimine `/auth` lõpp-punkti kaudu.

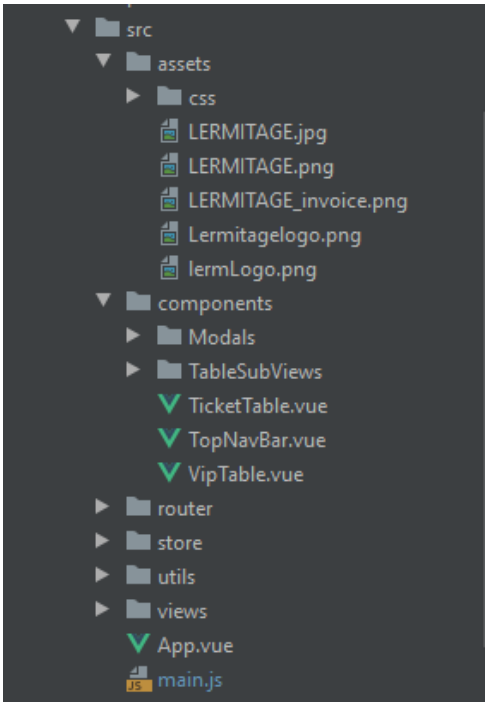
Tabel 1. API lõpp-punktid.

Marsruut	**Meetod**	**Kirjeldus**
/ticket/create	POST	Uue parkimise salvestamine. Tagastab salvestatud pileti
/ticket/update	POST	Parkimise andmete muutmine.
/ticket/getTicket/{id}	GET	Tagastab id järgi pileti
/ticket/unfinished	GET	Tagastab listi lõpetamata parkimistest
/ticket/departure	GET	Tagastab listi lõpetatud, kuid maksmata parkimistest
/ticket/getall	GET	Tagastab listi tasutud parkimistest
/ticket/delete/{id}	DELETE	Kustutab id järgi pileti
/free/save	POST	Salvestab uue tasuta parkimise põhjuse
/free/delete/{id}	DELETE	Kustutab id järgi tasuta parkimise põhjuse

Marsruut	**Meetod**	**Kirjeldus**
/free/getall	GET	Tagastab listina kõik tasuta parkimise põhjused
/invoice/{id}	GET	Tagastab parkimise id järgi arvele vajalikud andmed
/price/save	POST	Uue hinna salvestamine
/price/delete/{id}	DELETE	Olemasoleva hinna kustutamine hinna id järgi
/price/getall	GET	Tagastab listina kõik hinnad
/payment/add	POST	Lisab uue makse
/paymentType/getTypes	GET	Tagastab listing makseviisid
/singleday/getAll	GET	Tagastab listina kõik ühepäevase parkimise põhjused
/singleday/save	POST	Salvestab uue ühepäevase parkimise põhjuse
/singleday/{id}	GET	Tagastab ühepäevase parkimise põhjuse id järgi
/singleday/deletebyid/{id}	DELETE	Kustutab ühepäevase parkimise põhjuse id järgi
/vipClient/addNewVip	POST	Lisab uue püsikliendi
/vipClient/updateVip	POST	Muudab püsikliendi andmeid
/vipClient/delete/{id}	DELETE	Kustutab püsikliendi id järgi
/vipClient/find/{id}	GET	Tagastab id alusel püsikliendi
/vipClient/getAll	GET	Tagastab listi kõikide püsiklientidega
/vipStatus/getAll	GET	Tagastab listi kõikide püsikliendi staatustega
/auth	POST	Õige kasutajanime ja parooli korral lubab ligipääsu teistele API lõpp-punktidele. Tagastab JWT

6.3 Klientrakendus

Vue.js rakendus on samuti jagatud kuude alamkausta (Vt. Joonis 12).



Joonis 12. Vue.js koodi struktuur.

Assets kaust sisaldab rakenduses kasutatavaid pilte ning Sass faili stiili seadmiseks ning Bootsrapi vaiketeema muutmiseks.

Components kaust sisaldab endas väiksemateks osadeks jagatud Vue komponente, milles **Modals** sisaldab modaalseid dialoogaknaid ning **TableSubViews** parkimispiletiga seonduvat detailvaade komponenti ja andmete muutmise komponenti.

Router kaust sisaldab vue ruuterit, kus on ära märgitud kõik rakenduses leiduvad teed. Samuti sisaldab ruuter beforeEach navigatsioonitõkist, mis kontrollib, kas kasutajal on lubatud pääseda ligi autentimist vajavatele lehekülgedele.

Store kaust sisaldab Vuex Store'i, kus erineva eesmärgiga andmed on jagatud nimemoodulite abil eraldi moodulitesse. moodulid sisaldavad olekut (*state*), gettereid *(getters)*, mutatsioone (*mutations)* ja toiminguid (*actions).* Toimingu võivad sisaldada

asünkroonseid operatsioone ja seetõttu kasutatakse toiminguid, et teha päringuid serverile. Päringute tegemiseks kasutakse Axios teeki (Joonis 13). Kui kasutaja on sisse logitud, siis lisatakse Axiose päringutele autoriseerimise päis, millega antakse kaasa JWT. JWT on vajalik serverrakenduselt ligipääsu saamiseks API lõpp-punktidele.

```
async getUnfinishedTicketsAction(context) {
            await Axios.get("/ticket/unfinished")
                .then(
                    response => {
                        if (response.status === 200) {
                            const tickets = response.data;
                            context.commit("getUnfinishedTickets",
tickets);
                        } else {
                            console.log("something went wrong");
                        }
                    }
                );
        }
```

Joonis 13. Asünkroonse Axiose päringu tegemine vuexi toimingutes.

Utils kaust sisaldab erinevates komponentides korduvalt vajaminevaid meetodeid, meetodit PDF kujul arve genereerimiseks ning Vee-validate valideerimisreegleid.

Views kaust sisaldab komponente, mida autor on lugenud nii-öelda vaadeteks. Näiteks uue parkimispileti lisamise vaade, mis on omakorda jagatud Components kaustas leitavateks väiksemateks komponentideks.

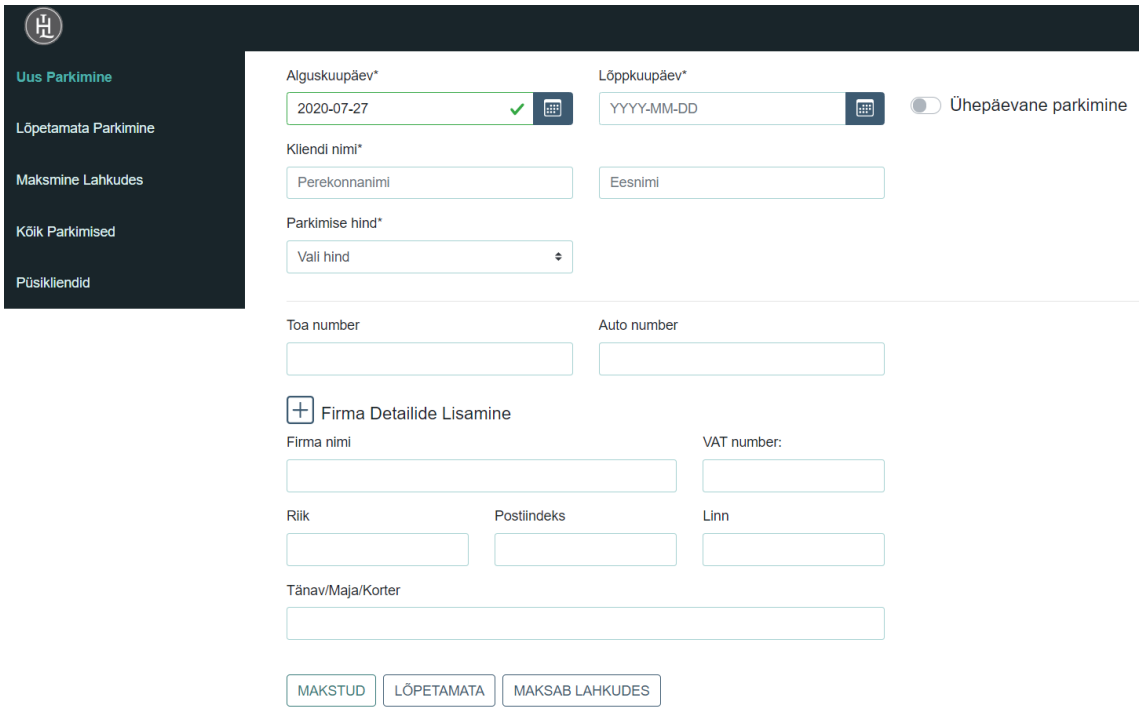
6.3.1 Front-end rakenduse vaated

Rakenduse funktsionaalsuse põhiosaks on parkimise registreerimine. Parkimise registreerimise vorm (Vt. Joonis 14) omab nelja alati kohustuslikku välja: alguskuupäev, lõppkuupäev, kliendi perekonnanimi ja parkimise hind. Need on minimaalsed väljad, mis peavad salvestamiseks täidetud olema.

Kui tegemist on sama päeva parkimisega ehk külastaja pargib vaid mõned tunnid, saab ühepäevase parkimise märkeruudult muuta lõppkuupäeva samaks alguskuupäevaga. Alguskuupäev on vormi laadimisel automaatselt tänane kuupäev. Mõlemat kuupäeva saab kirjutada YYYY-mm-dd formaadis käsitsi ja valida kalendrist.

Ühepäevase ning tasuta parkimise puhul on nõutav põhjuse valimine rippmenüüst (Vt. Joonis 15), et vajadusel administraator, vastuvõtujuht või juhtkonnaliikmed saaksid vaadata, mis põhjusel antud kliendil vastav hind oli.

Eesnimi, toa number, auto number ning kõik firma detailidega seotud väljad on mittekohustuslikud. Ainult sauna või restorani külastavatel klientidel ei ole toa numbrit ning tihtipeale unustavad külastajad registreerimiskaardil oma auto numbri märkida, seega ei saa neid välju märkida kohustuslikeks, kuigi vastuvõtuadministraatoril on kohustus neid võimalusel täita. Firma detailid on tavaliselt vaid äriklientidel ning tavaklientidel antud informatsiooni kajastada ei ole vaja.

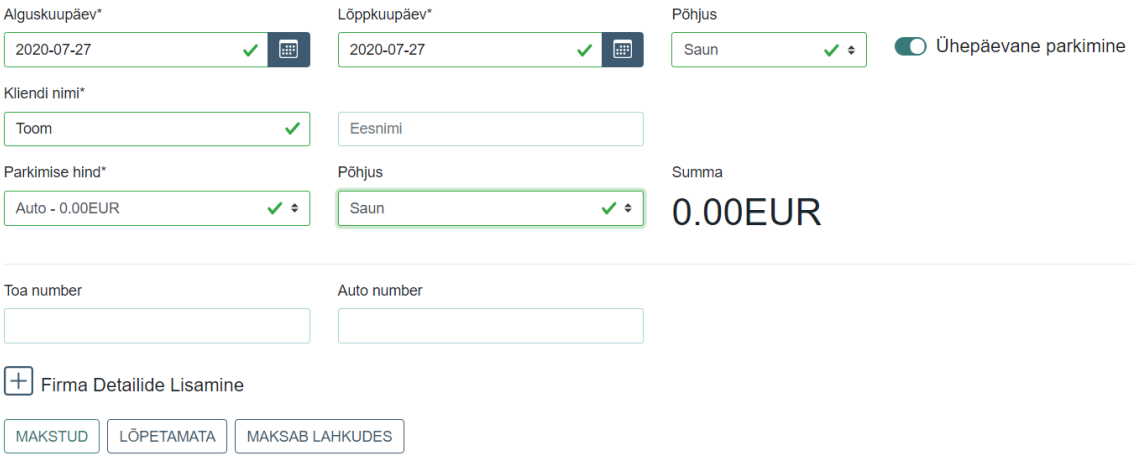


Joonis 14. Uue parkimise lisamise vorm.

Joonis 15. Põhjuste valimine tasuta ja ühepäevaste parkimiste puhul.

Parkimise salvestamiseks on kolm nuppu – makstud, lõpetamata ja maksab lahkudes. Lõpetamata parkimised on andmete täiendamist vajavad parkimised, mille puhul klient pargib enne auto ning alles siis täidab registreerimiskaardi ülejäänud andmetega (sh auto number ning toa number).

Lõpetamata parkimised kuvatakse koondtabelina, klikkides menüüs „Lõpetamata Parkimised“ lingile.



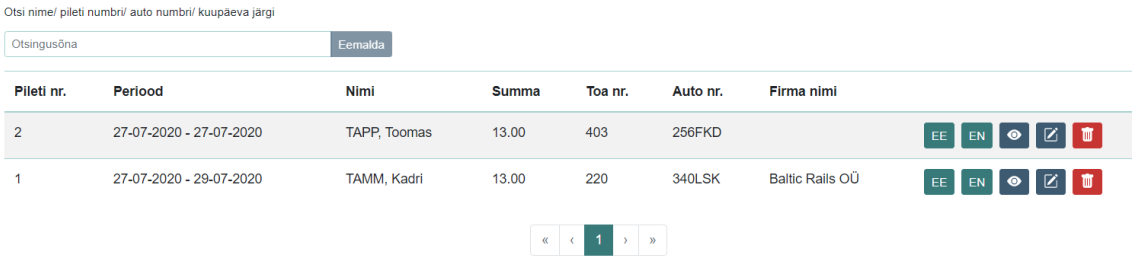
Lahkudes makstavad parkimised omavad juba kõike vajalikku informatsiooni, kuid klient on avaldanud soovi tasuda teenuse eest väljaregistreerimise ajal. Lahkudes makstavad parkimised on menüüs kuvatud „Maksmine Lahkudes“ all.

Makstud parkimised sisaldavad kõiki vajalikke andmeid ning klient on koheselt teenuse eest tasunud. Peale maksa nuppu vajutades kuvatakse kasutajale modaalne dialoogaken (Vt. Joonis 16), millel on näha unikaalne parkimispileti number, makseviisi valimise rippmenüü ning lahter lisakommentaari jaoks. Administraator on kohustatud parkimispileti numbri kirjutama või tembeldama kliendile antavale piletile. Makseviisi valikud on näiteks Visa, Mastercard, sularaha ning hotelliprogrammis kasutatav city ledger. Makstud parkimisi kuvatakse koondtabelina „Kõik Parkimised“ all.

Joonis 16. Makseviisi kinnitamise dialoogaken.

Sarnaselt makseviisi kinnitamisega avaneb dialoogaken ka lõpetamata ja maksab lahkudes nuppudele vajutades, kuvades parkimispileti numbri.

Kõik eelnimetatud parkimise tabelid on struktuurilt (Vt. Joonis 17) sarnased, kuvades pileti numbri, parkimise perioodi, kliendi nime, kogusumma ja olemasolu korral toa numbri, auto numbri ning firma nime. Ridu on võimalik filtreerida pileti numbri, perioodi, nime ja auto numbri alusel. Kõikidel tabelitel on olemas detailvaate, kustutamise ja erineval kujul pileti andmete muutmise nupp. Kõikide parkimiste koondvaates on lisaks eelnimetatutele arve genereerimiseks EE (Vt. Lisa 3) ja EN (Vt. Lisa 4) nupud vastavalt arve genereerimise keelele.



Joonis 17. Kõikide parkimiste koondvaade.

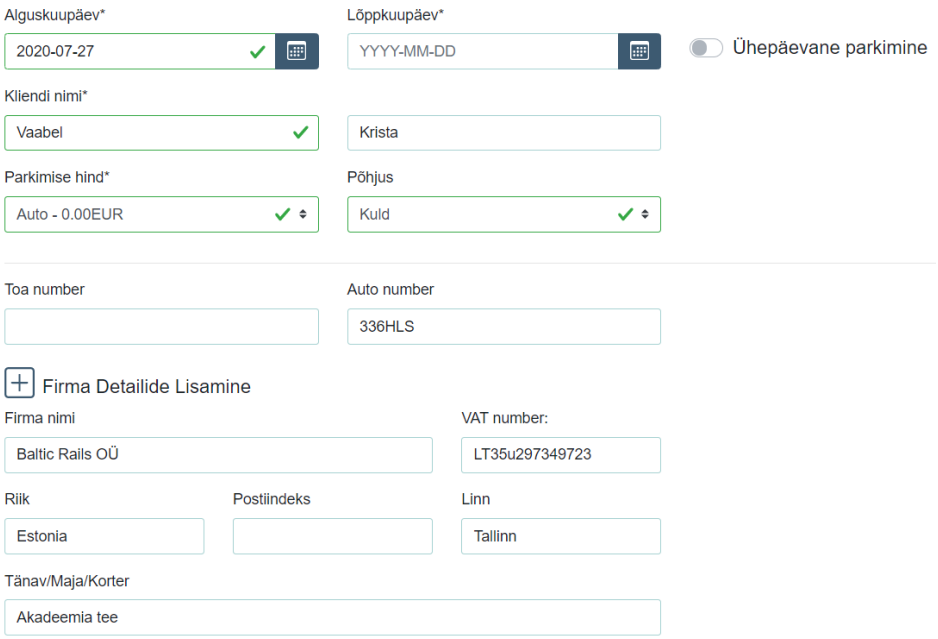
Hotellil on kuueteise lahtioleku aasta jooksul tekkinud palju püsikliente, kelle andmed muutuvad tavaliselt väga harva ning enamasti kasutavad püsikliendid sama numbrimärgiga sõidukit pikema perioodi jooksul. Selleks, et iga kord ei peaks uuesti kliendiga seotud andmeid küsima, otsima ning sisestama on rakenduses loodud süsteem, kus parkimise info on võimalik automaatselt teadaoleva informatsiooniga täita, vajutades klientide koondtabelis vastava püsikliendi real oleval pluss nupul (Vt. Joonis 18).

Joonis 18. Püsiklientide koondtabel.

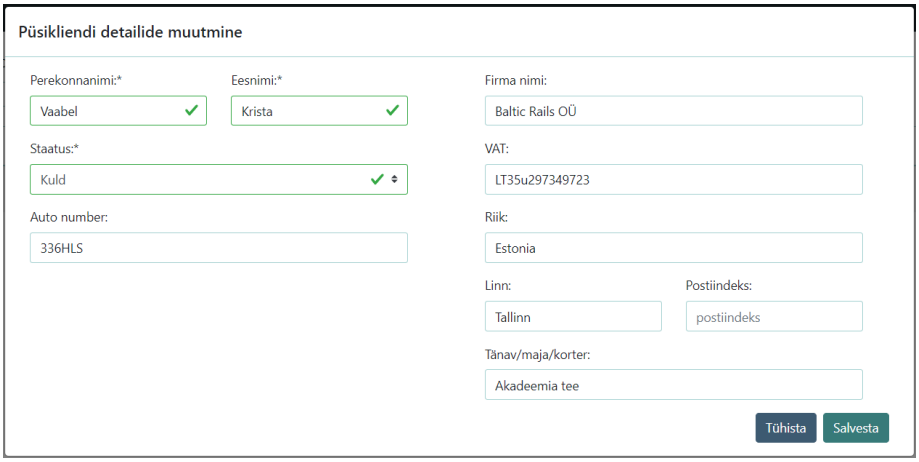
Püsikliendi andmete eeltäitmisel märgitakse kõik teadaolevad väljad (Vt Joonis 19). Administraator peab vaid lisama parkimise lõppkuupäeva ja vajadusel muutma alguskuupäeva.

Kuna püsiklientidele on lojaalsusprogrammi kaudu parkimine olenemata parkimise pikkusest tasuta, siis on parkimise hinna vaikeväärtuseks valitud „Auto - 0 EUR“. Kõiki välju on võimalik vastavalt vajadusele muuta ning need muudatused kajastuvad vaid sellel spetsiifilisel parkimisel, mitte kliendi profiilil.

Kui kliendi nimi, firma detailid, püsikliendi staatus või auto number peaks muutuma, siis saab vastavaid muudatusi teha Joonis 20 näidatud andmete muutmise dialoogaknas. Erinevalt uue parkimise lisamise vormist on püsikliendi puhul kohustuslik nii ees- kui ka perekonnanimi ja kliendi staatus, milleks on hetkel kas kuld või vip klient.



Joonis 19. Püsikliendi andmetega eeltäidetud uue parkimise lisamise vorm.



Joonis 20. Püsikliendi andmete muutmise dialoogaken

7 Rakenduse testimine hotelli administraatorite poolt

Valmis rakenduse kood laeti AWSi (*Amazon Web Services*), et võimaldada hotelli administraatoritel rakendust testida.

Vue.js rakendus laeti ülesse Amazon S3 *bucket*isse, millel on aktiveeritud staatilise veebilehe majutamine. Spring rakendus laeti AWS Elastic Beanstalk'i ning andmebaas Elastic Beanstalk'i kaudu RDS'i (Amazon Relational Database Service).

Andmete edastamiseks üle HTTPS protokolli osteti aastase kasutusõigusega `.tk` tasuta domeen, mis ühendati Amazon Route 53 abil Elastic Beanstalk'is oleva rakenduse aadressiga. HTTPS protokolli kasutamiseks vajalik sertifikaat anti välja Amazon Certificate Manager poolt.

Testimise eesmärgiks oli saada tagasisidet rakenduse kasutajamugavuse ning töövoo loogilisuse osas.

Administraatorid leidsid, et rakendus on kasutajasõbralik ja lihtsustab kiirel tööpäeval parkimise andmete sisestamist. Kõige rohkem rõhutati arve genereerimisel mitme keele olemasolu positiivsusele.

Rakenduse negatiivsete külgedena toodi välja, et põhjuste ja hindade rippmenüüd võiksid olla paremini kategoriseeritud vähendamaks aega, mis kulub pikemast listist vajaliku valiku üles leidmisele. Samuti toodi välja, et parkimise hinda võiks saada käsitsi muuta.

8 Edasiarenduse võimalused

Käesoleva töö raames loodud rakendus on piisav, et asendada senine paberi kujul toiminud parkimissüsteem, kuid rakendusele on võimalik juurde lisada palju lisafunktsionaalsust.

Hetkel on kasutusel ainult vastuvõtuadministraatori roll, mis võimaldab parkimisi sisestada, muuta, kustutada ja vaadata. Lisades juurde juhtkonna liikme rolli, mis oma olemuselt oleks nagu rakenduse administraatori roll, oleks võimalik kuvada varasemate parkimiste põhjal statistikat, erinevaid graafikuid ja teha ennustusi edasiste parkimiste kohta.

Statistika mugavaks vaatamiseks oleks vaja luua koondpaneel. Näiteks saaks statistika raames kuvada parkimise keskmist hinda, parkimiste arvu teatud perioodil, keskmist parkimise pikkust jms. Statistika oleks vajalik sisend juhtkonna liikmetele, et teha näiteks parkimisega või majutusega seotud eripakkumisi.

Lisaks eelnimetatule peaksid juhtkonna liikmed saama lisada, muuta ja kustutada parkimise hindu, tasuta parkimise põhjuseid ning ühepäevase parkimise põhjuseid, mida hetkel saab teha vaid manuaalselt.

Rakenduse kliendi poolne osa on arendatud kasutades Vue.js raamistikku. Üheks valiku põhjuseks oli võimalus tulevikus minna üle veebirakenduselt töölauarakendusele. Töölauarakendusi on võimalik luua Electron.js raamistikuga, liidestades see eksisteeriva Vue.js rakendusega.

Vastuvõtuadministraatoritele oleks mugavam töölauarakendus, kuna töö toimub alati samas arvutis ning töölauarakendust on tööriistaribalt lihtsam leida, kuid juhtkonna liikmetele, kes võivad tööd teha mitmest seadmest ning ka kontoriväliselt oleks vajalik pigem veebirakendus.

9 Kokkuvõte

Bakalaureusetöö eesmärgiks oli luua rakendus, mis lihtsustab vastuvõtu administraatorite igapäevatööd, grupeerides kõik parkimisega seotud tegevused ühte rakendusse.

Töö raames valminud rakenduses saab administraator lisada parkimisi, muuta parkimise andmeid, lisada parkimise andmetele firma detailid, kustutada parkimisi, lisada püsikliente, luua püsikliendi andmetega eeltäidetud parkimise vormi, muuta püsikliendi andmeid, kustutada püsikliente ning printida arve inglise ja eesti keeles.

Rakendus loodi järgides kolmekihilise klient-server arhitektuuri mustrit. Klientrakendus on üles ehitatud ühelehelise veebilehena kasutades Vue.js raamistikku ja Bootstrap-vue, Vuex, Vue Router, Axios ja Vee-validate teeke. Serverrakendus töötab Spring Boot raamistikul ning toimib RESTful API-na, millele klientrakendus saab HTTP päringuid teha. Andmed salvestatakse MySQL andmebaasi.

Töö käigus valminud rakendus on esimene versioon, mis rahuldab vastuvõtu administraatori jaoks vajalikku funktsionaalsust, kuid vajaks edasiarendust, et rakendus oleks kasulik ka hotelli juhtkonnale.

Kasutatud kirjandus

[1] „Nordic Hotel L'Ermitage tähistas avamist,“ Postimees, 6 10 2004. [Võrgumaterjal]. Available: https://www.postimees.ee/1437859/nordic-hotel-l-ermitage-tahistas-avamist. [Kasutatud 21 07 2020].

[2] „L'Ermitage hotelli koduleht,“ [Võrgumaterjal]. Available: https://lermitagehotel.ee/. [Kasutatud 21 07 2020].

[3] „E-pilet ettevõttele,“ Parkimine.ee, [Võrgumaterjal]. Available: https://www.parkimine.ee/arikliendile/e-pilet-ettevottele/. [Kasutatud 1 August 2020].

[4] „Teenindusasutusele,“ EuroPark, [Võrgumaterjal]. Available: https://europark.ee/parkimistehnika/maaomanikule. [Kasutatud 1 August 2020].

[5] „3-Tier Architecture: A Complete Overview,“ [Võrgumaterjal]. Available: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/. [Kasutatud 31 Juuli 2020].

[6] „Client/Server Architecture,“ 5 Veebruar 2018. [Võrgumaterjal]. Available: https://www.techopedia.com/definition/438/clientserver-architecture. [Kasutatud 31 Juuli 2020].

[7] M. Boyd, „Single Page Applications: A Powerful Design Pattern for Modern Web Apps,“ 9 Märts 2018. [Võrgumaterjal]. Available: https://medium.com/a-lady-dev/single-page-applications-a-powerful-design-pattern-for-modern-web-apps-ec3590bb7e7a#:~:text=What%20Is%20A%20Single%20Page,new%20page%20from%20the%20server.. [Kasutatud 26 Juuli 2020].

[8] „What is REST,“ [Võrgumaterjal]. Available: https://www.codecademy.com/articles/what-is-rest. [Kasutatud 26 Juuli 2020].

[9] „Gradle User Manual,“ [Võrgumaterjal]. Available: https://docs.gradle.org/current/userguide/userguide.html. [Kasutatud 22 Juuli 2020].

[10] F. Sáez, „Convention Over Configuration,“ [Võrgumaterjal]. Available: https://facilethings.com/blog/en/convention-over-configuration. [Kasutatud 31 Juuli 2020].

[11] S.-. K. Kompus, „Tarkvara ehitust automatiseeriva tööriista Gradle,“ 2016. [Võrgumaterjal]. Available: http://www.cs.tlu.ee/teemaderegister/get_file.php?id=412. [Kasutatud 22 Juuli 2020].

[12] B. Muschko, „Next-generation builds with Gradle,“ Manning Publications, Veebruar 2014. [Võrgumaterjal]. Available: https://learning.oreilly.com/library/view/gradle-in-action/9781617291302/kindle_split_011.html. [Kasutatud 21 Juuli 2020].

[13] D. Leuck, M. Loy ja P. Niemeyer, „Chapter 1. A modern Language,“ O'Reily

Media, Inc, 3 Aprill 2020. [Võrgumaterjal]. Available: https://learning.oreilly.com/library/view/learning-java-5th/9781492056263/ch01.html. [Kasutatud 23 Juuli 2020].

[14] N. H. Minh, „Java SE versions history,“ 2 Aprill 2020. [Võrgumaterjal]. Available: https://www.codejava.net/java-se/java-se-versions-history. [Kasutatud 23 Juuli 2020].

[15] „Why Spring,“ [Võrgumaterjal]. Available: https://spring.io/why-spring. [Kasutatud 26 Juuli 2020].

[16] „Inversion of Control,“ [Võrgumaterjal]. Available: https://www.tutorialsteacher.com/ioc/inversion-of-control. [Kasutatud 26 Juuli 2020].

[17] „Spring Boot,“ [Võrgumaterjal]. Available: https://spring.io/projects/spring-boot. [Kasutatud 27 Juuli 2020].

[18] M. Tyson, „What is JPA? Introduction to the Java Persistence API,“ 2 Aprill 2019. [Võrgumaterjal]. Available: https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html. [Kasutatud 30 Juuli 2020].

[19] „What is MySQL?,“ [Võrgumaterjal]. Available: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. [Kasutatud 24 Juuli 2020].

[20] E. Eessaar, „Serveri andmebaasisüsteem,“ 24 Juuli 2020. [Võrgumaterjal]. Available: https://maurus.ttu.ee/download.php?aine=346&document=31685&tyyp=do. [Kasutatud 24 Juuli 2020].

[21] „What is Vue.js,“ [Võrgumaterjal]. Available: https://vuejs.org/v2/guide/index.html. [Kasutatud 25 Juuli 2020].

[22] „Intro to Vue.js,“ [Võrgumaterjal]. Available: https://www.vuemastery.com/courses/intro-to-vue-js/vue-instance/. [Kasutatud 25 Juuli 2020].

[23] „Vue Router,“ [Võrgumaterjal]. Available: https://router.vuejs.org/. [Kasutatud 24 Juuli 2020].

[24] „What is Vuex,“ [Võrgumaterjal]. Available: https://vuex.vuejs.org/. [Kasutatud 26 Juuli 2020].

[25] „Bootstrap vs Bootstrap Vue: What are the differences?,“ 16 Juuli 2020. [Võrgumaterjal]. Available: https://stackshare.io/stackups/bootstrap-vs-bootstrap-vue#pros. [Kasutatud 25 Juuli 2020].

[26] „Bootstrap-vue,“ [Võrgumaterjal]. Available: https://bootstrap-vue.org/. [Kasutatud 25 Juuli 2020].

[27] A. Awad, „Vee-validtae,“ [Võrgumaterjal]. Available: https://logaretm.github.io/vee-validate/. [Kasutatud 25 Juuli 2020].

[28] K. Vaabel, P. Päll ja M. Raudvere, „ExpenseTracker,“ [Võrgumaterjal]. Available: https://github.com/notapotplant/ITI0208-life. [Kasutatud 7 Aprill 2020].

Lisa 1 – Parkimise arve hotelliprogrammist

Tel. +372 6996 400, Fax +372 6996 401
info@lermitagehotel.ee www.lermitagehotel.ee

L'ERMITAGE
HOTEL ★★★★

EUROSTANDARD NORDIC
LV40103735492

Katlakalna 9
Riga
LATVIA

Invoice

Invoicenumber	**238107**
Date:	17.07.2020
Room:	513
Page:	1/1
Ref.No.	EXP-1680010311-0

Karina Abele

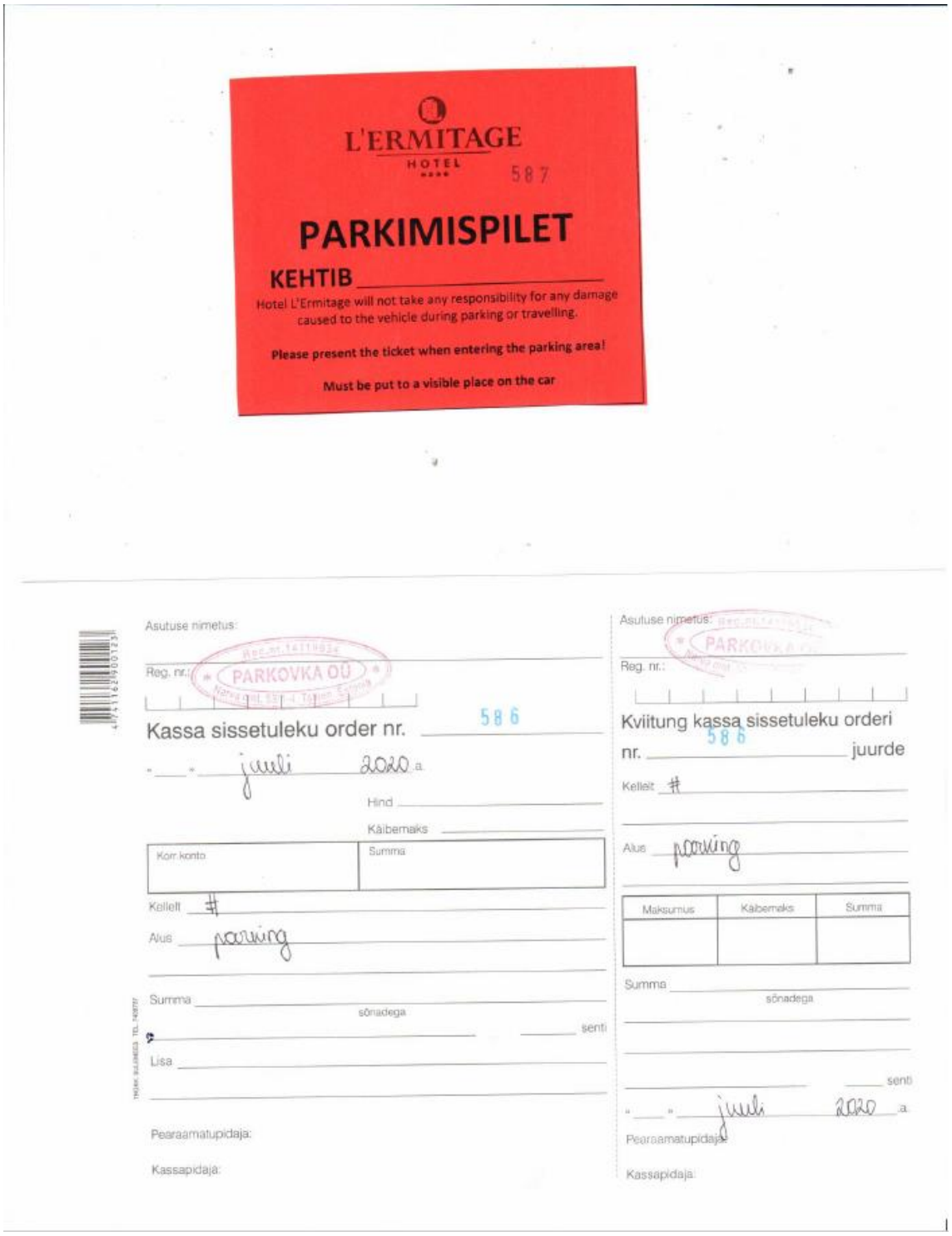
Date	Description	Price EUR	Qty	Amount EUR
17.07.2020	Parking car	13,00	2	26,00
Net amount 20 %				21,67
VAT 20 %				4,33
Total				**26,00 EUR**
Paid:				
17.07.2020	Cash			26,00
Open Balance				**0,00 EUR**

Thank you for choosing us. We are looking forward of being your host again soon.

Invoice printed by : Metsla, Mari-Liis

Hotel L'Ermitage OÜ Toompuiestee 19, 10137 Tallinn, ESTONIA Reg nr: 10574156 KMKR/VAT nr: EE100563793	**Pank / Bank: Swedbank AS** Konto / acc. EE252200221012281159 SWIFT: HABAEE2X **Pank / Bank: AS LHV Pank** Konto / acc. EE697700771003594033 BIC/SWIFT: LHVBEE22

Lisa 2 – Parkimispilet ja kassa sissetulekuorder



L'ERMITAGE
HOTEL ****
587

PARKIMISPILET

KEHTIB _____

Hotel L'Ermitage will not take any responsibility for any damage caused to the vehicle during parking or travelling.

Please present the ticket when entering the parking area!

Must be put to a visible place on the car

Asutuse nimetus:

Reg. nr.: PARKOVKA OÜ

Kassa sissetuleku order nr. 586

"___" juuli 2020 a.

Hind

Käibemaks

Kont konto	Summa

Kellelt #

Alus parving

Summa

sõnadega

senti

Lisa

Pearaamatupidaja:

Kassapidaja:

Asutuse nimetus:

Reg. nr.:

Kviitung kassa sissetuleku orderi nr. 586 juurde

Kellelt #

Alus parving

Maksumus	Käibemaks	Summa

Summa

sõnadega

senti

"___" juuli 2020 a.

Pearaamatupidaja:

Kassapidaja:

Lisa 3 – Eesti keelne arve parkimisrakendusest

Tel. +372 6996 400, Fax +372 6996 401
info@lermitagehotel.ee www.lermitagehotel.ee

Baltic Rails OÜ
LT239473829290

Akadeemia tee 11/2
Tallinn ,12616
Estonia

Arve nr. **1**
Kuupäev: 13.07.2020

Arve

Tamm Toomas

Alguskuupäev	Kirjeldus	Öö hind EUR	Ööde arv	Summa EUR
13.07.2020	Parkimine	6.50	1	6.50

Tasutud: City Ledger 6.50

Hotel L'Ermitage OÜ Toompuiestee 19, 10137 Tallinn, ESTONIA Reg nr: 10574156 KMKR/VAT nr: EE100563793	Parkovka OÜ Narva mnt. 53/1-4, Tallinn, Estonia Reg nr: 14119934

Lisa 4 – Inglise keelne arve parkimisrakendusest

Tel. +372 6996 400, Fax +372 6996 401
info@lermitagehotel.ee www.lermitagehotel.ee

Invoice nr. **2**
Date: 27.07.2020

Invoice

Suurmäe Tiit

Date	Description	Price per night EUR	Nights	Total EUR
27.07.2020	Parking	13.00	1	13.00

Paid: Cash 13.00

Hotel L'Ermitage OÜ Toompuiestee 19, 10137 Tallinn, ESTONIA Reg nr: 10574156 KMKR/VAT nr: EE100563793	Parkovka OÜ Narva mnt. 53/1-4, Tallinn, Estonia Reg nr: 14119934