TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAG40LT

Guido Veek 123610

PITSAKULLERI MOBIILIRAKENDUSE INFOSÜSTEEMI ANALÜÜS JA KASUTAJALIIDESE PROTOTÜÜP

Bakalaureusetöö

Juhendaja: Tarmo Robal
PhD
Teadur

Tallinn 2016

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: Guido Veek

25.05.2016

Annotatsioon

Käesolevas bakalaureuse lõputöös kirjeldatakse Pitsakulleri mobiilirakenduse loomeprotsessi algusest lõpuni. Tuuakse välja ning analüüsitakse vajalikke infosüsteemi erinevaid valdkondi, mida on vaja mobiilirakenduse loomiseks ning seda Pitsakulleri mobiilirakenduse projekti näitel.

Töö eesmärk on anda lugejale arusaadav, detailne ning analüüsiv ülevaade Pitsakulleri mobiilirakenduse loomeprotsessist. Kuna Pitsakulleri rakendus erineb tavalistest pitsakulleri teenustest, kus transport toimub punktist A punkti B, siis tänu sellele on kogu süsteem keerukam. Autor analüüsib ja kirjeldab erinevaid süsteeme ning nende rolle antud rakenduse valmistamise ja ka kasutamisel. Samuti annab autor ülevaate kasutatavatest tarkvaralistest programmidest ja lahendustest ning kuidas ühildada Pitsakulleri rakendus Google Maps-i kaardirakendusega ning SMS-teenustega.

Samuti esitatakse lugejale Pitsakulleri mobiilirakenduse esialgne kasutajaliidese prototüüp.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 7 peatükki, 7 joonist, 2 tabelit.

Abstract

System Analysis and GUI Prototype for Pitsakuller Mobile Application

Creating a single mobile application, depending on its size, can be either complicated or less complicated, but it is certainly not going to be an easy process. Developer has to handle many different parts of the information system and all these parts have to work together as a whole. Pitsakuller delivery application can be called a complicated mobile application, because it is going to consist of many different systems and functions that communicate with each other and this has to be problem-free communication.

There is a variety of methods, tools and tutorials on how to make mobile applications, but in this thesis, the author describes everything that he thinks is the best for making Pitsakuller application, after a detailed study and analysis.

The author's work is designed to the Pitsakuller mobile application, describing the creative process through a detailed description and analyses of the information system and to present graphical user interface, that is made according to the project.

In the first 3 chapters, the author is going to describe Pitsakuller's main systems, systems that are going to carry this application and are going to hold all the information. After that, author is going to explain what kind of programming languages and software he is going to use and how the data moves between server and application. In the final chapter, author is going to present graphical user interface, that has been made according to the plans of this project.

The main purpose of this thesis was to cover all the important aspects of making Pitsakuller mobile application. The author was able to analyse and describe the information system as well as creative process of Pitsakuller mobile application and was able to find out what is the best for this application.

The thesis is in Estonian and contains 42 pages of text, 7 chapters, 7 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface,* rakendusliides ehk programmiliides
Child spawning	Alamklasside tekitamine/väljakutsumine
CLI	*Command-line Interface*, käsurealiides
DB	*Database*, andmebaas
Desktop	Töölaud
Framework	Tugiraamistik, tugiprogramm
GB	*Gigabyte*, faili suuruse ühik
GUI	*Graphical User Interface,* graafiline kasutajaliides
HDD	*Hard Disk Drive*, kõvaketas, välismälu andmekandja
Host	Serverisüsteemi majutaja, süsteemi võõrustaja
HTML	*HyperText Markup Language*, keel, millega märgendatakse veebilehti
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
HTTPS	*Hypertext Transfer Protocol Secure*, turvaline hüperteksti edastusprotokoll
ID	Identifikaator
IDE	*Integrated Development Environment*, arenduse tarkvararakendus programmeerijatele
IP	*Internet protocol address*, internetiaadress, arvutite ja muude arvutivõrus olevate seadmete omavaheliseks suhtlemiseks
JavaScript	Ojektorienteeritud programmeerimiskeel veebilehtede skriptimiseks
JSON	*JavaScript Object Notation,* lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal
JSX	XML laandne koodistiiling
MB	*Megabyte*, megabait, faili suuruse ühik
MVC	*Model View Controller,* tarkvara arhitektuurimuster kasutajaliidese rakendamiseks
Native	Native rakendus ehk pärisrakendus, loodud mingile kindlale platvormile

OS	Operatsioonisüsteem
PHP	*Hypertext Preprocessor*, platvormist sõltumatu programmeerimiskeel
PHP-FPM	*FastCGI Process Manager - Common Gateway Interface*, tegeleb PHP käskude vastuvõtuga, võimalikult kiiresti
Port	Andmesideühenduse lõpp-punkt
PSR-4	Üldine failide laadimise spetsifikatsioon PHP jaoks
SCM	*Source Code Management*, lähtekoodi haldus
Shell	Käsukeele tõlk, mis käivitab standardselt sisendseadelt (nt klaviatuur) saadud sisendeid.
SMS	*Short Message Service*, tekstisõnumite saatmise teenus
SQL	*Structured Query Language*, struktuurpäringukeel, andmebaasi päringukeel
SSD	*Solid State Drive*, pooljuhtketas, välismälu andmekandja
TB	*Terabyte*, faili suuruse ühik
UI	*User Interface*, kasutajaliides
UML	*Unified Modeling Language*, ühtne modelleerimiskeel
VPS	*Virtual Private Server*, virtuaalne privaatne server
XML	*Extensible Markup Language*, laiendatav märgistuskeel, eesmärk on infot jagada erinevate infosüsteemide vahel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Ühe mobiilirakenduse loomine, sõltuvalt selle suurusest, võib olla, kas keeruline või vähem keeruline, kuid kindlasti ei ole see lihtne protsess. Arendajal tuleb koostööd tegema panna paljud erinevad infosüsteemi osad. Pitsakulleri rakendust võib nimetada keeruliseks mobiilirakenduseks, kuna see hakkab koosnema väga paljudest erinevatest süsteemidest ja funktsioonidest, mis omavahel probleemivabalt suhtlema peavad.

Pitsakulleri mobiilirakenduse eesmärk on kliendile pakkuda uudset ning võimalikult kiiret pitsade tellimis- ja transporditeenust. Kliendil on rakendust kasutades võimalik pitsad soovitud sihtkohta tellida ning seda vaid mõne minutiga. Kõik see on võimalik, kuna pitsad on juba eelnevalt kullerite autodes olemas.

Suurim erinevus Pitsakulleri rakendusel, võrreldes tavaliste pitsakulleri teenustega, on see, et klient saab reaalajas jälgida kullerite asukohti kaardil. Kliendile kuvatakse kaart aktiivsete kulleritega, peale mida märgib klient soovitud tellimuse sihtkoha. Kulleritele vajutades avaneb informatsioon autodes olevatest pitsadest, mille hulgast klient valib endale sobivad pitsad välja ning esitab tellimuse. Kullerile laekub teade tellimusest, koos kliendi asukohaga ning olenevalt nende omavahelisest kaugusest, võib kogu protsess aega võtta vähem kui 5 minutit.

Erinevaid meetodeid, õpetusi ning vahendeid, kuidas mobiilirakendust teha on palju, kuid antud töös kirjeldab autor kõike seda, mida tema peale detailset uurimist ja analüüsimist otsustas pidada parimaks, et luua Pitsakulleri mobiilirakendus.

Autori töö eesmärk on Pitsakulleri mobiilirakenduse loomeprotsessi kirjeldamine läbi infosüsteemi detailse analüüsi ja kirjeldamise ning esitada vastavalt projekti kirjeldusele loodud kasutajaliidese prototüüp. Loomeprotsessi kirjeldamisel toob autor välja nii Pitsakulleri rakenduse analüüsi ja arhitektuuri, süsteemi platvormi, kui ka tarkvaraliste lahenduste analüüsi, andmebaasisüsteemi kirjelduse, kolmandate osapoolte kaasamise ning samuti detailse rakenduse arenduskäigu kirjelduse.

Antud bakalaureusetöö teises peatükis kirjeldab autor Pitsakulleri rakenduse üldist funktsionaalsust ning esitleb arhitektuuri joonise ja UML diagrammid. Kolmanda peatüki fookus on Pitsakulleri rakenduse süsteemi platvormil, kus kirjeldatakse valitud serverisüsteemi ning operatsioonisüsteemi. Neljandas peatükis keskendub autor andmebaasisüsteemile ning esitab andmebaasi mudeli. Viiendas peatükis kirjeldab autor Pitsakulleri süsteemi tarkvara, toob välja kasutatud programmid ja programmeerimiskeeled ning selgitab, kuidas ühendada rakendusega SMS-teenused ning Google Maps-i positsioneerimise funktsioon. Kuuendas peatükis kirjeldab autor Pitsakulleri rakenduse arenduskäiku ning esitleb vastavalt projekti kirjeldusele koostatud kasutajaliidese prototüübi.

2 Pitsakulleri rakenduse analüüs ja arhitektuur

Käesolevas peatükis selgitab autor Pitsakulleri mobiilirakenduse funktsionaalsust läbi üldise funktsionaalsuse kirjelduse, süsteemi arhitektuurse joonise ning joonise kirjelduse ja UML diagrammide.

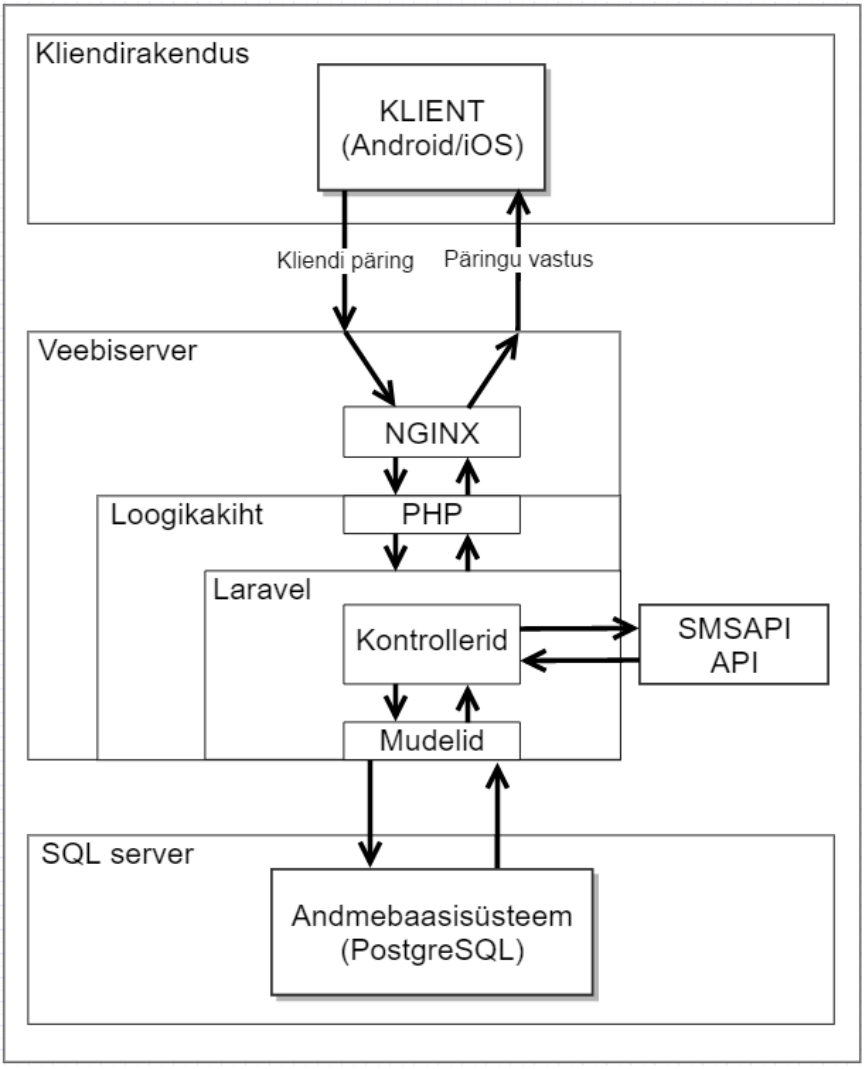
Pitsakulleri mobiilirakenduse põhifunktsioon, mis eristab seda teistest pitsade transpordi teenustest, kus üldjuhul toimetatakse pitsad restoranist kliendi soovitud sihtkohta on see, et kliendil on võimalik reaalajas kõiki aktiivseid Pitsakulleri kullereid kaardilt jälgida. Sarnast kaardivaate lahendust kasutab ka Eestis loodud Taxify mobiilirakendus, mis on loodud taksode tellimiseks. Igal aktiivsel kulleril, mida klient kaardilt näeb on autos juba pitsad soojad olemas. Kliendil on võimalik aktiivse kulleri tooteid vaadata vajutades soovitud auto ikooni peale, mispeale kuvatakse talle info nii kulleri, kui ka autos olevate toodete kohta. Tellimuse koostamiseks tuleb kliendil valida soovitud pitsade kogus ning kinnitada tellimus. Sõltuvalt kulleri ja kliendi omavahelisest kaugusest võib kogu protsess aega võtta vähem kui 5 minutit.

Üldise rakenduse funktsionaalsuse kohaselt, mille baasil on loodud ka kasutuslood, saab kasutaja enda konto aktiveerida/luua kasutades reaalset telefoninumbrit. Aktiveerimise tõrke/aegumise korral on kliendil võimalik uus kood küsida. Kliendil on võimalik tellimuse sihtkohta kaardile märkida. Süsteem võtab vastu kliendi märgitud sihtkohti ning saadab need kullerile ja ka vastupidi. Kliendil on võimalik vaadata informatsiooni kulleri kohta, kuvatakse nii toodete info, kui kulleri info. Kliendil on võimalik koostada ja tühistada tellimusi ning tellimus võib koosneda erinevatest toodetest ja kogustest. Tellimust saab koostada juhul, kui sihtkoht on märgitud. Peale kliendipoolset tellimuse kinnitamist on kulleril võimalik seda näha ning kuller peab tellimuse aktsepteerima või keelduma sellest. Kulleril on võimalik tarneaega kliendi märgitud sihtkohta muuta. Süsteem lubab kulleril lisada, uuendada ja kustutada tooteid ning nendega seonduvaid andmeid (nt kogus). Kliendil on võimalik toodete/teenuse eest tasuda kaardimaksega ja sularahas. Kulleril on võimalik tellimus lõpetada.

Rakendus on kättesaadav kõikidele iOS ja Android-i operatsioonisüsteemiga seadmetele, mille versioon ei ole vanem, kui 5 aastat. Rakenduse teenuste kasutamiseks peab kliendil olema internetiühendus ning konto edukalt aktiveeritud. Teenuste kasutamise ajalugu salvestatakse nii kulleri, kui ka kliendi kohta. Kuller suudab samaaegselt teenindada ühte klienti korraga. Kuller ei vastuta ootamatustest tuleneva tarneaja muutuse pärast. Kulleril on võimalik tellimus lõpetada ainult peale kliendi toodete tasumist ning ainult siis antakse tooted kliendile üle.

2.1 Pitsakulleri rakenduse arhitektuur

Pitsakulleri rakendus on arhitektuuriliselt jagatud nelja kihti - kliendirakendus, veebiserver, loogikakiht ning SQL server. Joonisel 1 on esitatud Pitsakulleri mobiilirakenduse arhitektuurne joonis.



Joonis 1. Pitsakulleri rakenduse arhitektuuri joonis.

Kliendirakendus hakkab päringuid tegema rakenduse domeenile, näiteks www.pitsakuller.ee, mis suunatakse edasi vastavale IP-le *port*-iga 80 (HTTPS ühenduse puhul 443). See IP kuulub Pitsakulleri serverile, kus asub rakenduse API. Peale igat kliendi päringut, jääb kliendirakendus serverilt vastust ootama.

Serveril on omakorda veebiserveri rakendus, mis vaikimisi kuulab *port*-e 80 ja 443. Päringu saabumisel NGINX (veebiserver) teab, milliselt domeeninimelt (www.pitsakuller.ee) päring tuleb ning vastavalt seadistusele saadab päringu koos andmetega edasi PHP skriptile, kus on Laravel (PHP *framework* ehk tugiraamistik) ja jääb PHP vastust ootama.

Laravel kontrollib millisele marsruudile päring tuli, nt www.pitsakuller.ee/login puhul oleks Laravelis ära registreeritud "/login" marsruut ja vastavalt sellele ära märgitud, milline kontrolleri loogika jooksutatakse. Kontroller valideerib kasutaja saadetud sisendid, näiteks kontrollib, kas telefoninumber näeb välja reaalne ja kontrollib, kas kasutaja on saatnud korrektse SMS-koodi aktiveerimiseks.

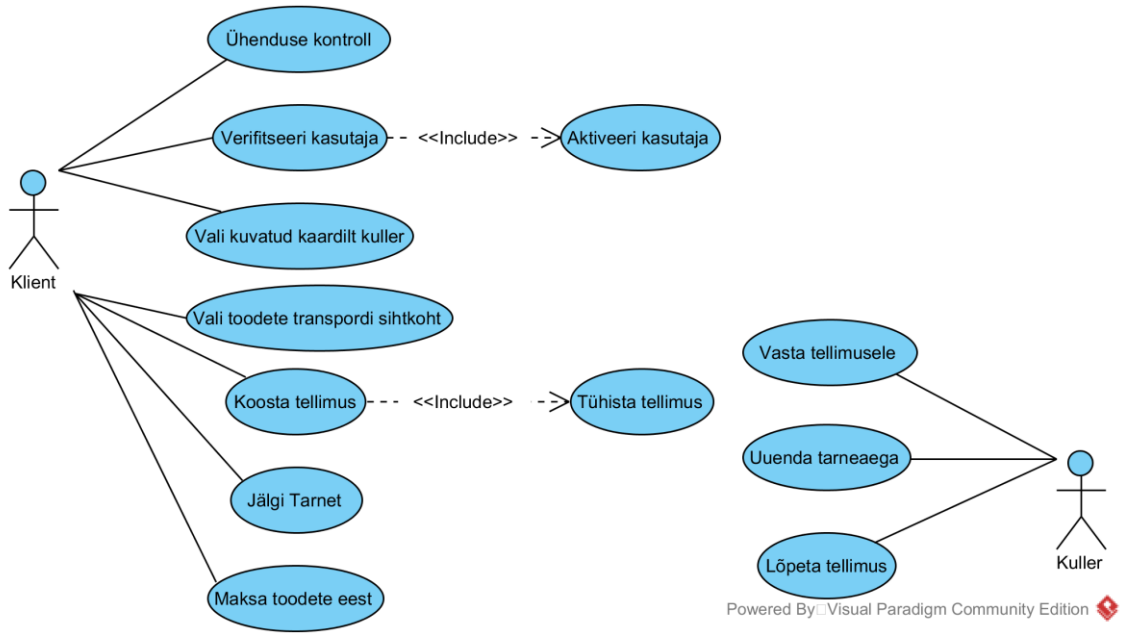
SMS-teenuste eest vastutab SMSAPI poolt koostatud Laravel-ile mõeldud teek, millega on võimalik suhelda SMSAPI API-ga.

Pärast valideerimist saab teostada juba järgmisi tegevusi, näiteks, kui kontroller tahab saada telefoninumbriga seotud tellimusi, siis selleks suhtleb kontroller mudeliga, andes kaasa telefoninumbri, et kätte vastu saada sellega seotud informatsioon. Kui loogika on kontrolleri poolt tehtud tagastatakse vastus.

Mudelid on otseses seoses andmebaasiga, sest üldjuhul mudelid representeerivad andmebaasi tabeleid. Mudelid loevad, uuendavad ja salvestavad andmeid andmebaasidesse.

2.2 Pitsakulleri rakenduse UML diagrammid

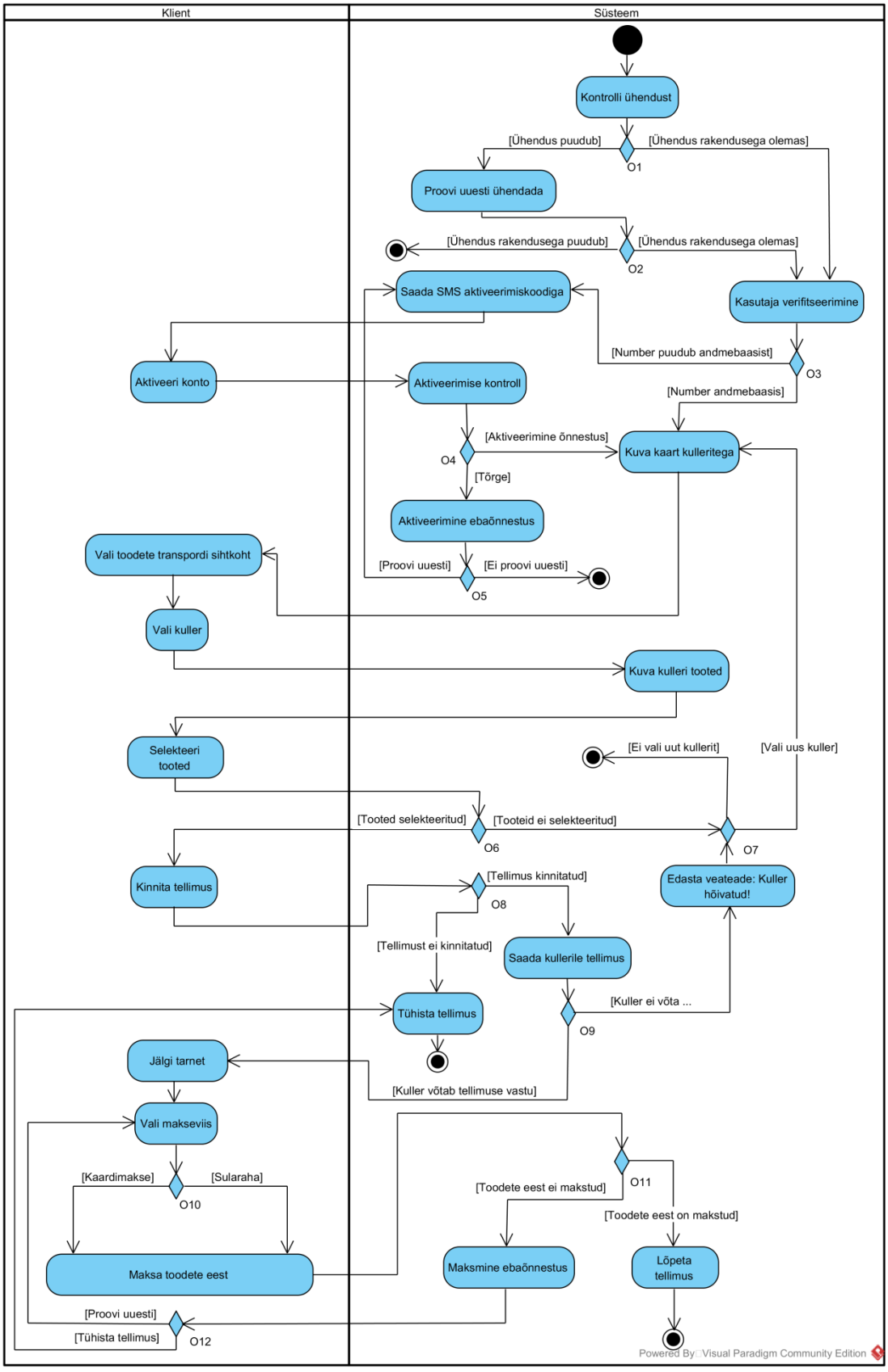
Käesolevas peatükis esitleb autor kasutusjuhtude diagrammi (Joonis 2) ning tegevusdiagrammi (Joonis 3). Diagrammid on koostatud vastavalt loodava rakenduse plaanidele. Kasutusjuhtude diagrammi dokumentatsioon on esitatud Lisas 1 Tabel 1 ning tegevusdiagrammi dokumentatsioon asub Lisas 2 Tabel 2.



Joonis 2. Pitsakulleri rakenduse kasutusjuhtude diagramm

Joonisel 2 esitatud Pitsakulleri rakenduse kasutusjuhtude diagramm kirjeldab süsteemi tööd kliendi ning kulleri vahel. Kliendil on kullerist rohkem funktsionaalsut, kulleri süsteemi pakutav funktsionaalsus on piiritletud ainult tellimustega seonduvale.

Joonisel 3 esitatud Pitsakulleri rakenduse tegevusdiagramm kirjeldab tellimisprotsessi algusest lõpuni kliendi seisukohast. Süsteemi ja kliendi tegevused on eraldi välja toodud.



Joonis 3. Pitsakulleri rakenduse tegevusdiagramm

3 Pitsakulleri rakenduse süsteemi platvorm

Käesolevas peatükis kirjeldab autor Pitsakulleri mobiilirakenduse kandvaid süsteeme. Autor kirjeldab ning analüüsib kasutusele võetavat serverisüsteemi, operatsioonisüsteemi nii arenduse, kui ka serveri jaoks.

3.1 Serverisüsteem

Tänasel päeval on võimalik valida kahe serverisüsteemi vahel, milleks on pilveserver ja füüsiline server. Käesolevas peatükis analüüsib autor mõlemat serverisüsteemi, toob välja plussid ja miinused ning põhjendab ära valiku pilveserverisüsteemi kasuks. Samuti analüüsib ja võrdleb erinevaid teenusepakkujaid ning selgitab, miks osutus valituks Zone.ee pakutav pilveserver.

Pilveserver ehk virtuaalserver on serveri tüüp, kus ettevõttel on võimalik serveriteenust kasutada läbi interneti. Puudub vajadus serverisüsteemi üles seadmiseks, vaid on võimalik teenust sisse osta ning teenusepakkuja vastutab kõige eest (tehnika, serveriruum, hooldus, andmete dubleerimine).

Füüsiline server ehk lokaalne server on serveri tüüp, mis üldjuhul asub ettevõtte pinnal (kuid mitte alati), koos kõige riistvaraga. Olenevalt serveri suurusest on vaja selleks eraldi ruumi ja soovitatavalt peab serveriruum olema puhas, tolmuvaba, konditsioneeritud, püsiva temperatuuriga jne. Füüsiline server on pilveserverist kallim aga samas ka võimekam, sest olemasolevad serveri ressursid on alati garanteeritud.

Üldiselt pakuvad enamus teenusepakkujad tööajaks 99,99%. See tähendab, et ühe aasta jooksul tohib erinevate tehniliste rikete tõttu, nagu näiteks voolu- või internetikatkestus, server maas olla 52 minutit (v.a serveri hooldustöödest tingitud katkestused). 99,99%

tööaega on lihtne lubada aga keeruline kontrollida[1]. Selleks tasub alati uurida ja/või otsida klientide ülevaateid ning arvustusi erinevate teenusepakkujate kohta.

Üks suurimaid eeliseid pilveserveril füüsilise serveri üle on ruumi kokkuhoid. Tihtipeale alustaval ettevõttel ei ole palju reaalset tööpinda, seega tuleb olemasolev pind maksimaalselt ära kasutada. Järgnevalt toob autor välja pilve- ning füüsilise serveri eelised ja puudused.

Pilveserveri eelised ja puudused [1], [2]:

- Virtuaalsed serverid (VPS) ei pea asuma füüsiliselt ettevõtte pinnal ega vaja selleks eraldi ruumi.
- Kliendi andmed on dubleeritud. Ootamatu rikke korral võtab töö üle varuserver ning serveri reaalne maasoleku aeg on minimaalne.
- Serveri paketi (maht, mälu, protsessorid) uuendamine on lihtne. Sõltuvalt teenusepakkujast tuleb valida kas uus pakett, või on võimalik suurendada olemasoleva paketi jõudlust ning mahtu ka eraldi.
- Kliendil on võimalik maksta ainult selle eest, mida tema kasutab.

Pilveserveri puudused [1], [2]:

- Küsitav usaldusväärsus, puudub täielik kontroll. Kuna klient ei tea täpselt, kus tema andmed asuvad, siis tekib küsimus, kas andmed ikka püsivad turvaliselt?
- Võrreldes füüsilise serveriga saab sama hinna eest madalama jõudluse.

Füüsilise serveri eelised ja puudused [1], [2]:

- Võimalik saada täielik kontroll serveri üle.
- Füüsilisele serverile on võimalik valida ükskõik millist OS-i (pilveserveril on OS-ide nimekiri ette antud).
- Võimalik üles seada suure ressursi nõudlusega rakendusi.

Füüsilise serveri puudused:

- Vajab füüsilist ruumi serverite hoiustamiseks.

[1] http://www.proit.ee/pilveserver-kuidas-valida-endale-sobiv

- Serveri riistvara uuendamine ja hooldamine on keerulisem kui pilveserverit kasutades. Kasutaja peab ise kõige eest hoolt kandma.
- Kasutaja maksab alati maksimaalse kasutamise eest

3.1.1 Pilveserveri teenusepakkujate võrdlus ja Pitsakulleri rakenduse serverisüsteem

Võttes arvesse kõiki peatükis 3.1 esitatud eeliseid ja puudusi pilve- ning füüsilise serverisüsteemi kohta, langes Pitsakulleri mobiilirakenduse puhul lõplik valik pilveserverile. Serverisüsteemi *host*-iks ehk majutajaks valis autor Zone Media (edaspidi Zone). Käesolevas peatükis analüüsib autor erinevaid teenusepakkujaid ning selgitab, miks osutus valituks Zone.

Järgnevalt esitab autor kolme erineva teenusepakkuja võrdluse, kus võrdleb kolme erinevat paketti. Võrdluses (Tabel 1) on Zone, Veebimajutus.ee ning välismaise alternatiivina Digital Ocean (DO). Võrdluse huvides valis autor Zone-i paketid võimalikult sarnaseks DO ja Veebimajutus.ee-ga.

Tabel 1. Kolme teenusepakkuja sarnaste pakettide võrdlus

Teenusepakkuja	**SSD kettamaht**	**Tuumad**	**Mälu**	**Andmeedastusmaht**	**Hind**
Digital Ocean	40GB	2	2GB	3TB	18€
	80GB	4	8GB	5TB	71.30€
	160GB	8	16GB	6TB	142.60€
Veebimajutus.ee	40GB	2	2GB	2TB	36€
	80GB	4	4GB	3TB	66€
	160GB	8	8GB	5TB	120€
Zone	40GB	2	2GB	*	32.50€
	75GB	4	8GB	*	76.65€
	150GB	6	12GB	*	122€

*Zone-il ei ole fikseeritud andmeedastusmahtu, vaid kliendile on 3GB väljuvat andmeedastusmahtu tunnis tasuta ning iga väljuv 1GB, mis ületab 3GB tunnis, maksab 0.05€[1].

Projekti huve ning esialgseid eesmärke arvestades, kus teenuse fookus on suunatud Eestile, Soomele ning teistele Balti riikidele, osutus valituks Zone-i pilveserver, mis asub Eestis. Erinevalt DO-ist ning Veebimajutus.ee-st, on Zone paindlikuma paketivalikuga. Zone-is on kliendil juba enne tellimuse kinnitamist võimalik valida paketi maht ning jõudlus tuumade ja mälu näol eraldi[1]. DO ning Veebimajutus.ee kasutavad aga fikseeritud pakette, mida Tabelis 1 näha võib.

Samuti üheks väga suureks plussiks on Zone-i tasuta andmeedastusmaht. Esialgsete (kaudsete) arvutuste ning koodi järgi mahub Pitsakulleri projekti andmeedastusmaht selle 3GB sisse ning mingeid lisatasusid ei tule. Esialgne kliendirakenduse poolt päritav koodinäide koos arvutuste ning selgitustega on esitatud Lisas 3.

Tänu Zone-i paindlikule paketivalikule on kliendil võimalik valida ning hakata maksma just selle eest, mida ta ära kasutab. Teiste esitatud teenusepakkujate pakettidega võib klient tihtipeale hakata üle maksma ega kasuta olemasolevalt ressurssi ära.

DO üks eelistest Zone-i ees on võimalus valida väga mahukaid pakette, näiteks ülivõimas pakett milles on 640GB kettamahtu, 64GB mälu, 20 tuuma ning 9TB andmeedastusmahtu[2]. Zone oma hinnakirjas sellises mastaabis pakette ei paku. Samuti ülemaailmne laienemine oleks Digital Ocean-iga kergendatud kuna serverid asuvad nii Euroopas (Frankfurt, Amsterdam, London), USA-s (New York City, Toronto, San Fransisco), kui ka Aasias (Singapur)[3].

Serverisüsteemi valides tuli silmas pidada ka seda, et valitud majutaja toetaks projektis kasutatavat operatsioonisüsteemi. Zone toetab Pitsakulleri mobiilirakenduse jaoks kasutavat Ubuntu operatsioonisüsteemi.

---

[1] https://www.zone.ee/et/teenus/pilveserver-pro-ssd/hinnad/

[2] https://www.digitalocean.com/pricing/

[3] https://www.digitalocean.com/features/scaling/

3.2 Operatsioonisüsteemid

Kui sobiv serverisüsteem, teenusepakkuja ning pakett valitud, siis järgmise sammuna tuleb otsustada milline operatsioonisüsteem (OS) peale installeeritakse. Antud projekti jaoks valis autor Ubuntu OS-i.

Ubuntu toetab Bash *shell*-i, mis on populaarne käsurealiides. Projektis kasutatavad tööriistad sõltuvad Bash-i funktsionaalsusest ning ei hakkaks Bash-ita tööle. Nendeks tööriistadeks on näiteks Node Package Manager (NPM) ning sellega seonduvalt React Native. Samuti on eelnimetatud tööriistad esmalt arendatud Linux-i peale, enne, kui neid laiendati teistele OS-idele. Ubuntu on populaarne operatsioonisüsteem ning tänu sellele on kasutajate hulk on suur. Tekkinud probleemide korral on suure tõenäosusega keegi sellega juba tegelenud ning lahenduse leidnud, mis aitab omakorda kaasa projekti sujuvale tööle.

Arenduses kasutab autor Ubuntut versiooniga 14.04. Kuna Pitsakulleri rakendus eeldab pikemaajalist serveri kasutust, vähemalt 2 aastat, siis oleks vajalik, et ka tugi oleks pikemaajaline. Versiooni 14.04 toetatakse aastani 2019[1].

[1] http://www.ubuntu.com/info/release-end-of-life

4 Pitsakulleri andmebaasisüsteem ja mudel

Öeldakse, et andmebaas on infosüsteemi tuum [3] ja kuna antud töös on organiseerimist vajavat informatsiooni palju, siis andmebaasidest ei saa üle ega ümber. Selles peatükis keskendub autor andmebaasidele, tutvustades nende olemust ning andmebaaside ülesannet antud rakenduse puhul. Joonisel 4 on esitatud projekti põhjal koostatud andmebaasi mudel ning samuti selgitab autor, miks osutus valituks PostgreSQL andmebaasisüsteemi, mis olid selle eelised teiste andmebaaside ees.

Andmebaasiks nimetatakse hästi organiseeritud, omavahel seotud ja süstematiseeritud andmete kogumit. Andmebaas on objektsüsteemi seisundit esitavate suhestatud, relatsioonis olevate andmete kogum ehk relatsioonis olevate ja säilitatavate andmetabelite hulk [4].

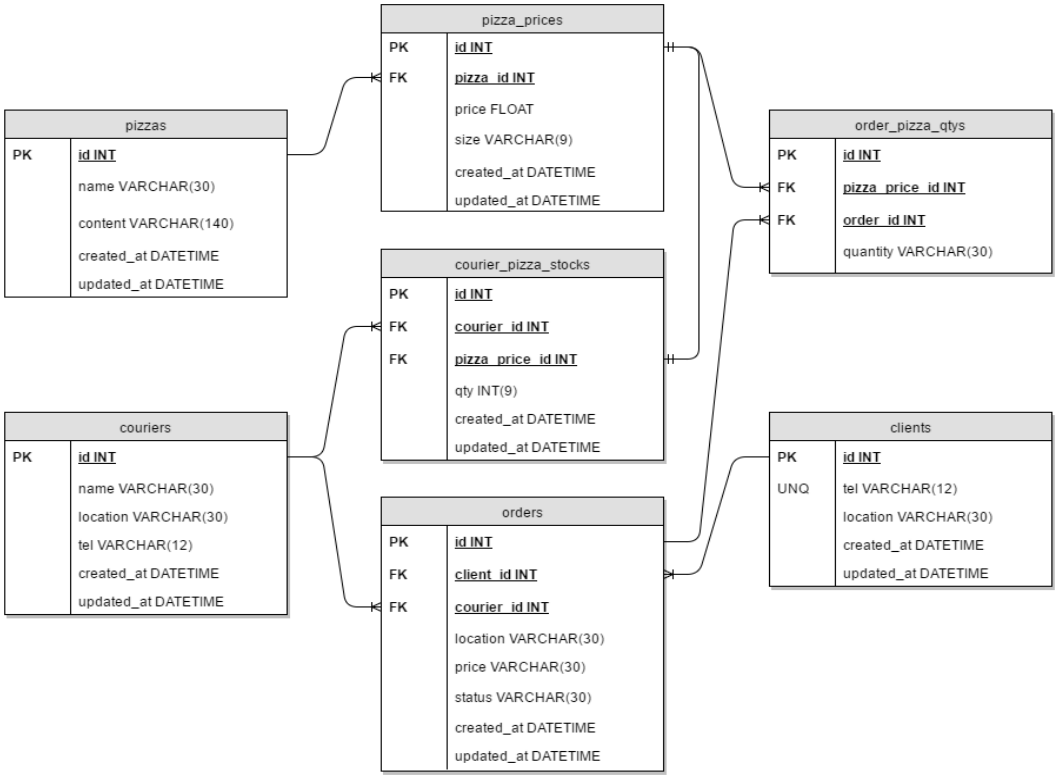
Andmebaasid on korraldatud viisil, mis lubab andmebaasis olevat infot peale korrektselt formuleeritud päringut kasutada, muuta või vajadusel uusi kirjeid lisada. Andmebaasi põhiülesandeks ning -funktsiooniks on luua, koostöös kogu infosüsteemiga, kvaliteetne infokeskkond objektsüsteemi seisundi ja selle muutmise kohta infosüsteemi kasutajatele [3].

4.1 Pitsakulleri andmebaasisüsteem ja andmebaasi mudel

Andmebaasisüsteemiks on tarkvara, millega saab andmebaase luua ja kasutada. Andmebaasisüsteemil on olemas ka vajalikud funktsioonid (SELECT, INSERT, UPDATE, DELETE jne.) andmete uuendamiseks ja hooldamiseks [4], näiteks:

- INSERT INTO pizza (name) VALUES ('Cananas');
- UPDATE pizza SET name = 'Peetri pitsa' WHERE name='Cananas';

Joonisel 4 on esitatud Pitsakulleri rakenduse põhjal koostatud andmebaasi mudel.



Joonis 4. Pitsakulleri rakenduse andmebaasi mudel

Pitsakulleri rakenduse andmebaasi mudeli tabelite selgitused on esitatud Tabelis 2.

Tabel 2. Pitsakulleri rakenduse andmebaasi mudeli tabelite selgitus

Tabeli nimi	**Tabeli selgitus**
pizzas	Tabelis asub pitsa nimi ning selgitus, mida see pitsa sisaldab.
pizza_prices	Sisaldab pitsa ID-d, millega antud pitsa hind on seotud ning samuti pitsa suurust, et sama pitsa erinevatel suurustel saaks olla erinev hind. Samuti hoiab endas pitsa suuruse nimetust - suur (L), keskmine (M), väike (S).
clients	Kliendi tabel, hoiab endas kliendi andmeid. Hetkel on vajalik kliendi telefoninumber, mis peab olema unikaalne ja kliendi valitud sihtkoht.
couriers	Pitsakulleri andmete tabel: kulleri nimi, tema telefon ja asukoht.
courier_pizza_stocks	Sisaldab courier_id-d - ühel kulleril võib olla mitme erineva pitsa kogused. Samuti hoiab see pizza_price_id-d, millega me saame teada, millise pitsa ja pitsa suurusega on tegu.
order_pizza_qtys	Hoiab pizza_price_id-d, et teada saada, mis pitsa, pitsa suuruse ja hinnaga tegu on. Orders tabeli ID on vajalik, et ühel tellimusel saaks olla küljes mitu erinevat pitsat.
orders	Sisaldab kliendi ID-d, nii saab ühel kliendil olla mitu tellimust (tellimuste ajalugu kliendi kohta). Samuti kulleri ID, millega saab kulleril olla mitu tellimust (tellimuste ajalugu kulleri kohta) ning hoiab antud tellimusega seonduvaid andmeid.

Kuna erinevaid andmebaase on sadu, siis otsustas autor valida kolme tuntud andmebaasi vahel:

- MySQL - Stabiilne andmebaasi süsteem, mis on populaarne valik, sest süsteemi on kaua praktikas kasutatud.
- MariaDB - MySQL loojate poolt tehtud andmebaasisüsteem. Kasutajatugi on nõrgem kuna seda on hetkel vähem kasutatud kui MySQL-i või PostgreSQL-i.
- PostgreSQL - Üks populaarseimaid andmebaasisüsteeme, mis on suure funktsionaalsusega. Valitakse tihtipeale suuremate ja keerulisemate andmebaasidega projektide jaoks.

Kolm eelnevalt väljatoodud andmebaasisüsteemi on kõik väga hinnatud ja tasemel ning nende vahel valides ei ole neist ühegagi võimalik suurt viga teha. Pitsakulleri rakenduse jaoks mingeid otseseid puudusi esitatud andmebaasidel ei ole, seega sobiksid kõik kolm, kuid peale erinevate ülevaadete, artiklite ning võrdluste läbitöötamist langes autori lõplik valik PostgreSQL-i kasuks.

Quora.com-is korraldatud küsitluses, nimega "*If you were to start a new project now (March 2015), would you use MySQL, MariaDB or PostgreSQL and why?*" selgus, et seitsmest vastanud arendajast 5 eelistas PostgreSQL-i. Arendajad jagasid enda kogemusi erinevate andmebaasidega ning eelistavad PostgreSQL-i tema töökindluse, jõudluse ja programmeerija-sõbraliku keskkonna pärast [5].

Samuti Slant.co poolt koostatud võrdlusest selgub, et taaskord PostgreSQL on kolmest võrreldavast eelistatuim. Ainsa miinusena on välja toodud, et väiksemate rakenduste puhul võib olla liigne ressursi raiskamine [6].

Nagu eelnevalt mainitud, siis suurt viga eelnimetatud andmebaaside valimisel teha ei ole võimalik. Autori valik langes PostgreSQL-i peale just teiste arendajate kogemuste, soovituste, positiivsete ülevaadete ning tagasiside tõttu.

5 Pitsakulleri süsteemi tarkvara ja andmete liikumine

Käesolevas peatükis toob autor välja Pitsakulleri rakenduse loomisel kasutatavad tarkvara erinevad osad. Saame teada milliseid programmeerimiskeeli ning programme kasutatakse, tutvustatakse abistavaid tööriistu ning teenuseid ja selgitatakse, kuidas toimub andmete liikumine.

Rakenduse arengu dokumenteerimise on väga mugavaks teinud tööriist nimega Git. Github-i poolt loodud avatud lähtekoodiga tasuta jagatav versioonikontroll Git on kavandatud käsitlema nii väiksemaid, kui ka väga mahukaid projekte kiiresti ning efektiivselt. Git on kergesti õpitav ning jätab projektile väga väikese jälje tänu heale jõudlusele, st kõik toimib kiiresti, ei loo takistusi ega kuluta ülemäära aega. Git ületab sellised lähtekoodi halduse ehk *Source Code Management* (SCM) tööriistad nagu Subversion, Concurrent Versions System, Perforce ja ClearCase tänu oma lisavõimalustele. Nendeks on odav kohalik hargnevus (ettevõttes saavad kõik vajalikud liikmed kergesti projektihaldusele ligi), mugav vaheetappide haldamine (enne koodi lõpliku väljastamist saab seda üle vaadata või muuta) ning saab rakendada mitme erineva arendaja tööprotsesse korraga [7], [8].

Git on Pitsakulleri projekti jaoks vajalik tööriist, kuna sellega võib säästa väga palju aega ja sellest tulenevalt raha, kuna töötunnid maksavad. Kui kood on kogu aeg ühest kindlast kohast kättesaadav, koos muutuste ajalooga, on tekkinud probleemidele lahenduse leidmine lihtsam.

Serveri ning kliendi- ja kullerirakenduse vaheliseks suhtlemiseks kasutab autor **NGINX** veebiserverit. NGINX on tasuta, avatud lähtekoodiga ning suure jõudlusega veebiserver. NGINX-i teenust kasutavad palju suured, kõrge külastatavusega saidid nagu näiteks Netflix, Hulu, Pinterest, WordPress.com ja GitHub [9].

Antud projektis kasutatakse NGINX-i nii testimisel, kui ka arendusel. Kogu suhtlus kliendi- ja kullerirakenduse ning muu süsteemi vahel käib läbi NGINX-i. Tegu on veebiserveri tarkvaraga, mis serveerib nii staatilisi, kui ka dünaamilisi faile üle HTTP ja HTTPS-i, ning suudab jooksutada PHP skripte läbi PHP-FPM teenuse.

PHP-FPM (*FastCGI Process Manager, Common Gateway Interface*) on teenus, mis võtab vastu PHP käsklused ning annab need edasi PHP mootorisse ja ootab vastust, eesmärgiga teha seda võimalikult kiiresti. FPM lisadeks on näiteks kiirendatud üleslaadimise tugi, dünaamiline/staatiline *child spawning*, täpsem protsesside juhtimine[1].

Pitsakulleri rakenduses toimub kogu andmeedastus **JSON** andmeformaadis, sest see on toetatud nii PHP, kui ka JavaScript-i poolt.

JSON (*JavaScript Object Notation*) on kerge, vähenõudlik andmete vahetuse formaat. Arendajate jaoks lihne kirjutada ning lugeda, masinate jaoks lihtne sõeluda ja genereerida. See on tekstiformaat, mis on keelest täiesti sõltuv, kuid samade tavadega, mis on tuttavad C-keelkonna programmeerijatele. Need omadused teevad JSON-ist ideaalse andmete vahetuse keele[2].

JSON on üles ehitatud kahte struktuuri[2]:

- Nime ja väärtuste paaride kollektsioon. Erinevates keeltes on see realiseeritud kui objekt, kirje, struktuur, sõnastik, võtmega nimekiri või assotsiatiivne massiiv.
- Järjestatud väärtuste nimekiri. Enamikes keeltes realiseeritakse seda kui massiivi, vektorit, nimekirja või jada.

5.1 PHP ning PHP tööriistad

PHP ehk *Hypertext Preprocessor*, on laialdaselt kasutatav vabavaraline objektorienteeritud programmeerimiskeel, mis on eriti sobiv veebiarenduseks[3]. See võimaldab jooksutada loogikat/arvutusi serveri poolt ilma, et kasutaja saaks teada lähtekoodi. Kood käivitatakse serveris ning selle tulemusena genereeritakse HTML dokument, mis saadetakse kasutajale üle HTTP või HTTPS-i.

[1] http://php.net/manual/en/install.fpm.php

[2] http://www.json.org/

[3] http://php.net/manual/en/intro-whatis.php

Antud projektis saab kogu põhiloogika olema PHP-s, milleks on näiteks tellimuste salvestamine ja lugemine andmebaasist, õigete tehingute ja tellimuste andmete kuvamine jne.

5.1.1 Laravel

Laravel on PHP *framework* ehk tugiraamistik (tugiprogramm). Laravel implementeerib *Model View Controller* (MVC) arhitektuuri, mis jagab koodi andmebaasiosa, loogikaosa ja visuaalse osa eraldi sektoritesse, mis teeb koodi lugemise ja kirjutamise selgemaks. Samuti kasutab Laravel loogilist kaustade ja failide struktuuri. Kui kausta nimi on "*Controllers*", siis käivad seal Laraveli kontrolleri klassid jne. *Model View Controller* koosneb kolmest erinevast osast:

- *Model* ehk mudel, mis tegeleb andmebaasist andmete lugemisega ja nende töötlemisega.
- *View* ehk vaade on komponent, mis hoiab rakenduse visuaalset osa. Selle projekti raames me *View* komponenti ei kasuta, sest Pitsakuller põhineb Android-il ja iOS-il, kus pole HTML-iga midagi peale hakata.
- *Controller* ehk kontroller võtab vastu kasutaja vormi sisendeid, mida ta töötleb ja valideerib ning suhtleb mudelitega. Joonisel X on koodinäide Laraveli kontrollerist, mis laeb mudelist andmeid ja tagastab need JSON andmetüübina.

Laravel on veebirakenduse *framework*, mille eesmärk on võtta ära programmeerimise valu, lihtsustades laialt levinud käske veebi projektides, nagu näiteks autentimine, marsuutimine, sessioonid ning andmete vahemällu hoiustamine[1].

Jooniselt 5 on näha, kui lihtsaks on Laravel andmete laadimise andmebaasist teinud. Arendaja peab ise vaid ühe rea koodi lisama (Joonis 5, rida 15), ülejäänud on juba Laravel-i poolt, et ostukorvi funktsioon töötaks.

[1] https://laravel.com/docs/4.2/introduction

```php
<?php

class OrderController
{

    function getOrderPizzas()
    {
        // Läbi tellimuse saab kätte mitu erinevat pitsat ja pitsa ühikut iga erineva pitsa jaoks telliti
        $pizzaCounts = Order::where('orderID')->orderPizzaQty();

        // Läbi tellitud pitsa koguste saab teada, mis pitsa hinna ja suurusega on tegu
        $pizzaPrices = $pizzaCounts->pizzaPrice();

        // Mis ostukorv kokku läks maksma - implementeeritakse autori poolt mudeli klassi
        $cartTotal = $pizzaPrices->getTotal();

        // Läbi pitsa hinna saab teada, millise pitsa ja pitsa suurusega on tegu
        $pizzas = $pizzaPrices->pizza();

        // Tagastatakse andmed JSON andmetüübina
        return View()->json([$pizzas, $pizzaPrices, $pizzaCounts]);
    }
}

```

Joonis 5. Laraveli kontrolleri näide

Pitsakulleri rakenduse jaoks pakub Laravel näiteks andmebaaside migratsioone, mis teevad andmebaasi struktuuride kirjutamise väga lihtsaks. Ühtegi rida SQL-i arendaja kirjutama ei pea. Kõik üldlevinud SQL-i funktsiooonid on mudelitesse sisse programmeeritud. Näiteks *Many to Many* (mitu mitmesse) suhte tabeli tegemine nõuab minimaalselt nelja faili loomist, 2 migratsioonifaili, iga migratsioonifail tekitab ühe põhitabeli (suhte tabelid genereeritakse automaatselt) ja 2 mudelifaili (iga eraldi mudel on seotud ühe andmebaasiga).

Failide loomise eest hoolitseb Laravel-i käsurealiides (CLI), milleks on Artisan. Arendaja hooleks on välja mõelda nimed ja kirjutada igasse faili 2 kuni 4 rida koodi. Mudeli sees tuleb ära defineerida, millise teise mudeliga neil suhtlus on ning Laravel teeb ülejäänu[1].

[1] https://laravel.com/docs/5.2/artisan

5.1.2 Composer

See on tööriist, millega saab hallata PHP teeke, mida projektis kasutatakse, hoolitsedes nii installeerimiste, kui ka uuenduste eest[1]. Sisuliselt on Composer paketihaldur, mis on mõeldud PHP pakettide jaoks. Pitsakulleri rakenduse loomisel seatakse kõik vajalikud PHP paketid ülesse just Composer-it kasutades. Kuna autor kasutab enda projektis ka Laravel-i, siis Laravel-i üles seadmine toimub ka just läbi Composer-i.

Kui Composer üles seatud, siis edaspidi tuleb vajalike failide installeerimiseks anda ainult õige käsklus ning süsteem haldab kõik muu ise. Üks kuulsamaid ja kasutatavamaid kohti pakkettide otsimiseks on Packagist.org veebileht, kus on välja toodud PHP paketid/moodulid, mida saab Composer-iga installida.

Uut projekti alustades on Composer-is kõigepealt täiesti tühi kaust. Kui soovitakse kasutada Laravel-i, nagu Pitsakulleri projektis, siis esmalt tuleb käivitada käsklus "composer init", mis küsib mõned küsimused projekti kohta (projekti nimi, selgitus, litsent, jne.). Peale nende väljade täitmist saab hakata soovitud Composer-i mooduleid ja pakette installima, kirjutades "composer *packagename*". Laraveli installimiseks tuleb kirjutada käsklus "composer require laravel/laravel" ja sarnase käskluse annab ka Packagist iga otsitud paketi kohta. Peale käskluse sisestamist tehakse automaatselt "composer.json" faili sissekanne, et kasutatakse Laraveli ja märgitakse ära alla laetud versioon. Kui soovitakse spetsiifilist versiooni, siis seda saab märkida käsureal lisaparameetriga "composer laravel:laravel:4.*". Sellise käsklusega laetakse alla kõige uuem versioon 4 alamversioon, mis hetkel on 4.12.11[2].

5.1.3 PhpStorm

JetBrains-i PhpStorm on kaubanduslik mitmeplatvormne IDE (*Integrated Development Environment*) PHP jaoks. PhpStorm on perfektne töötamaks koos selliste tugiraamistikega nagu näiteks WordPress, Zend Framework, Magento ja antud projektis kasutust leidva Laravel-iga[3].

---

[1] https://getcomposer.org/doc/00-intro.md

[2] https://packagist.org/packages/laravel/laravel#v4.2.11

[3] https://www.jetbrains.com/phpstorm/

Erinevaid IDE-si ja redaktoreid on PHP jaoks palju, näiteks Eclipse PDT, NetBeans, Zend Studio, kuid PhpStorm on PHP maailma võtmas tormina. Öeldakse, et see teeb kõike, mida üks IDE tegema peab ja teeb seda kõige paremini. See on ka põhjuseks, miks terved tiimid PhpStorm-ile üle lähevad [10].

Quora.com-i küsitluse "*What's the best editor/IDE for PHP?"* järgi saavutas PhpStorm 81 häälega kõige rohkem hääli. Järgnesid sellised teksti redaktorid nagu Notepad++ (55 häält), Sublime Text (52 häält) ning neljandal kohal oli IDE nimega Eclipse PDT (33 häält) [11].

Võttes arvesse teiste arendajate arvamusi ning positiivseid ülevaateid ja tagasisidet PhpStorm-i kohta otsustas autor ka Pitsakulleri projektis PhpStorm-i kasuks.

5.1.4 SMS-teenused

Pitsakulleri projekti puhul tuleb kliendil Pitsakulleri mobiilirakenduse esmasel kasutamisel enda konto aktiveerida, konto, mille unikaalseks ID-ks või tunnuseks on kliendi telefoninumber. Esmane aktiveerimine hõlmab endas ka kasutaja loomise protsessi, st kliendi telefoninumbriga ei ole veel registreeritud, vastasel juhul toimuks kasutaja autentimise protsess. See on vajalik nii kliendi kui ka teenusepakkuja usaldusväärsuse tõstmiseks. Samuti saab korduvalt heakorra eeskirju rikkunud või ebaviisakalt käitunud klientide telefoninumbrid ära blokeerida. Näiteks, kui ühe ja sama numbri pealt on 3 korda tellimus esitatud, kuller on kohale sõitnud aga klienti sihtkohas ei ole ega vasta ka telefonile, siis blokeeritakse number ära ning sellelt numbrilt enam tellimusi esitada ei saa.

Pitsakulleri konto mobiiltelefoniga aktiveerimiseks oli autoril võimalik valida kahe erineva lahendusviisi vahel:

- Helistamisega aktiveerimine - rakenduse käivitamisel kuvatakse kliendile aktiveerimiseks telefoninumber, millele helistades ühendatakse konto rakendusega. Sellist süsteemi kasutab Eestis näiteks Pargi.ee mobiilirakendus. Kui telefoninumber on rakendusega ühendatud, siis saab klient kasutada mobiilset parkimist ning parkimistasu lisandub kliendi telefoniarvele. Selline variant on kasutajasõbralik rakendustele, kus summad ei ole suured.

- SMS-iga aktiveerimine - rakenduse käivitamisel kuvatakse kliendile tema telefoni rakendusega ühendamiseks vastav teade, näiteks "Aktiveerimiseks saada SMS". Peale "Saada SMS" nupu vajutamist saadetakse kliendile rakenduse aktiveerimiseks kood. Olenevalt valitud teenusest või valmis ehitatud süsteemi funktsionaalsusest, peab klient aktiveerimiskoodi ise sisestama või, kui rakendusel on piisavalt õigusi, oskab rakendus enda saadetud SMS koodi automaatselt välja lugeda. Peale koodi sisestamist on kliendi telefon rakendusega ühendatud ning kasutamiseks valmis.

Antud projektis otustas autor kasutada SMS-iga aktiveerimise meetodit. Selle süsteemi üles seadmine on tehtud väga mugavaks tänu SMSAPI nimelisele ettevõttele. Arvestades projekti plaane, mis näevad ette laienemist Eestist välja ka teistesse riikidesse, siis aktiveerimise süsteem peab olema rahvusvaheline. SMSAPI just sellist võimalus pakubki, nad toetavad üle 800 mobiilsideoperaatori rohkem kui 200-s riigis[1].

SMSAPI pakub juba eelnevalt valmis tehtud PHP pakette, mis on üles laetud Git-i töökeskkonda. Arendaja peab ainult konfiguratsioonifaili tekitama, et SMSAPI teaks, kas rakendusel on üldse õigus saata SMS-e läbi SMSAPI[2].

Ühe saadetud sõnumi hinnaks Elisa, Telia (vana AS EMT ja Elion) ning Tele 2 võrkudesse on €0.0380[3]. Sellise hinna puhul tuhande kliendi aktiveerimiseks kulub 38€. Kui 10% ehk 100 klienti kasutab Pitsakulleri teenust kasvõi ühe korra, siis on süsteem ennast ära tasunud.

Võrreldes SMSAPI hinda teise sarnase globaalse teenusepakkuja Twilio-ga, siis SMSAPI on odavam. Twilio ühe Eestis saadetud sõnumi hinnaks on $0.05[4]. Lisaks, ei ole Twilio ülesseadmine nii lihtne. Kui SMSAPI-l on juba valmis tehtud PHP pakett, koos juhendiga, siis Twiliol on ainult juhend.

Arvestades Pitsakulleri projekti plaani laieneda Eestist välja ning nii hinda, kui ka kasutusmugavust, otsustas autor valida SMSAPI.

---

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

[2] https://www.smsapi.com/libraries

[3] https://www.smsapi.com/pricing

[4] https://www.twilio.com/sms/pricing/ee

5.2 JavaScript ning JavaScripti tööriistad

JavaScript (sageli lühendatud JS) on kerge, tõlgendatud ja objektorienteeritud esimese klassi funktsioonidega programmeerimiskeel, mida kasutatakse veebilehtede ja rakenduste interaktiivseks muutmiseks. JS on sündmustepõhine skriptimise keel, mis on dünaamiline ja toetab objektorienteeritud, kiireid ja funktsionaalseid programmeerimise stiile [12]. Kogu Pitsakulleri rakenduse funktsionaalsus ehitatakse üles JavaScripti peale.

JavaScript töötab kliendipoolsel rakendusel, mida saab programmeerida ja disainida vastavalt vajadusele, et kuidas rakendus või veebileht mingile sündmusele käituma peab. JS on lihtsasti õpitav ja võimas skriptimise keel, laialdaselt kasutusel näiteks veebilehe käitumise kontrollimiseks [12].

JavaScript-i pakettide haldamiseks tuleb esmalt installeerida **Node Package Manager** (NPM), mis on JS-ile loodud paketihaldur, aitamaks JS-i arendajatel kergesti erinevaid pakette jagada[1]. NPM-iga on väga lihtne JS-il põhinevaid tööriistu alla laadida. Kui kasutatavad tööriistad või kooditeegid peaksid omakorda mingeid teisi teeke vajama, siis tänu NPM-ile laetakse need automaatselt alla. NPM kindlustab ka selle, et kõik paketid, mida kasutatakse, oleks sama versiooniga, kui need, mis serverisse jõuavad. Nii saab alati kindel olla, et kõik töötab korrektselt ja ei tekiks versioonide erinevusest tingitud tõrkeid.

5.2.1 React Native ja Google Maps-i positsioneerimise teek

Pitsakulleri rakenduse jaoks pakub React Native tarkvararaamistikku (tugiprogrammi) JavaScriptile, mis lubab ehitada *native* rakendusi ehk n-ö pärisrakendusi kasutades React-i[2].

Kuna Pitsakulleri rakenduse fookus on Android-i ja iOS operatsioonisüsteemidel, siis nende programmeerimisel tulebki appi React Native. React Native lubab ehitada maailmaklassi rakendusi *native* platvormidel, kasutades väljapeetud arendajate kogemusi JavaScript-i ja React-iga. React Native-i fookus on arendaja efektiivsusel

[1] https://www.npmjs.com/about

[2] https://facebook.github.io/react-native/

kõikide platvormide üle, mida arendajat huvitavad - õpi korra, kirjuta kõikjale[1]. Ehk siis Pitsakulleri rakenduse tervet programmikoodi ei pea täiesti eraldi kirjutama mõlemale platvormile (Android, iOS), vaid ainult see osa, mis on platvormist sõltuv.

Nagu eelnevalt mainitud, kasutab React Native Javascript-i, et luua iPhone-ile ja Android-ile *native* rakendust, mida saab vastavate rakenduste poodidesse üles laadida. *Native* rakenduse kompileerimine on võimalik tänu JavaScript-i süntaksi transformeerijatele, mis suudavad JavaScript-i koodi muuta iPhone-i jaoks Objective-C keelde ja Androidi puhul Javasse[2]. React Native lisab JavaScript-ile lisafunktsionaalsust, mis võimaldaks iPhone-i või Android-i tasemel kompilleerimist.

Kui tavalisel JavaScript-il on võimalik kasutada veebikomponente ( ) brauserite UI ehitamiseks, siis React Native-it kasutavas JavaScript-is see võimalik ei ole. Nii Android-ile, kui ka iPhone-ile on juurde lisatud vastavad komponendid. Visuaalsed komponendid tuleb mõlemale platvormile eraldi üles ehitada, kuid see-eest võib põhiloogika jääda samaks. See hoiab kokku nii aega, raha ja teeb koodi testimise lihtsamaks ning stabiilsemaks. Kuid Pitsakulleri rakenduse erinevatele platvormidele ehitamine ei ole ainus asi, kus React Native appi tuleb.

Pitsakulleri mobiilirakenduse üks olulisem ja võrreldes teiste Pitsakulleri teenustega erilisem funktsioon on see, et klient saab reaalajas kulleri asukohta jälgida. Kui klient on enda telefoninumbri ära kinnitanud, siis esimese asjana kuvatakse talle kaart aktiivsete kulleritega. Siit küsimus, kuidas kullerid kaardile liikuma saada?

Antud rakenduses kasutaksime Reactile Native-ile mõeldud teeki, mis muretseb ise Google Maps-i API suhtluse eest. Selle kasulikkus seisneb selles, et teegis on projektile vajalik funktsionaalsus juba olemas, nagu kliendi asukoha kuvamine, kliendi kaardile kullerite kuvamine ning nende positsiooni uuendamine[3].

Ainuke nõue on, et kliendil oleks Android või iOS operatsioonisüsteemiga telefon. Isegi geolokatsioon ei ole kohustluslik, sest kasutaja saab alati valida manuaalselt asukoha, kuhu kuller pitsa peab tooma. Kulleritel aga peab seade geolokatsiooni toetama, et

[1] https://facebook.github.io/react-native/

[2] https://facebook.github.io/react-native/docs/javascript-environment.html

[3] https://github.com/lelandrichardson/react-native-maps

kliendil oleks võimalik kulleri muutuvat asukohta näha. Sujuva liikumise tekitamiseks kaardil, arvestades andmesidemahtu, on optimaalne küsida kulleri asukohta iga kolme sekundi tagant pärast eelmise asukoha uuendust.

Pitsakulleri projekti tulevikuplaanidesse kuulub funktsioon, et kliendile kuvatakse ainult kullerid, kes on 5-10 kilomeetri raadiuses. Selline filtreerimine vähendab koormust serverile ning samuti transpordikulusid, kuid kõik see nõuab lisainvesteeringuid ning palju aktiivseid kullereid.

6 Pitsakulleri rakenduse arenduskäik ja kasutajaliidese prototüüp

Käesolevas peatükis keskendub autor rakenduse arenduskäigu samm-sammulisele kirjeldamisele ning esitamisele tuleb graafiline kasutajaliidese prototüüp.

6.1 Rakenduse arenduskäik

Rakenduse loomisel keskendub autor kõigepealt kliendi- ja kullerirakenduse kasutajaliidesele ning seejärjel liigub servi ja andmebaasi arenduse juurde.

Kliendi- ja kullerirakenduse loomiseks tuleb kõigepealt arenduskeskkond üles seada ning arendamiseks ette valmistada. Arenduskeskkonna OS-iks valis autor Ubuntu töölaua versiooni, millele tuleb omakorda peale installeerida kõik vajalikud tööriistad ja teenused.

Järgmise sammuna, et klientidele mõeldud rakenduse arendamisega saaks pihta hakata, on meil OS-i teenustest esmalt vaja installeerida NPM paketihaldur. See on vajalik JavaScripti jaoks, et rakenduse jaoks vajalikud paketid kiirelt ja lihtsalt alla laadida, milleks on näiteks React Native.

Nüüd, kui NPM on olemas, läheb vaja tööriista, millega koodi kirjutatakse. Selleks kasutab autor PhpStormi, mis toetab ka JavaScripti ning just React Native-i laadseid koodistiilingut, mida nimetatakse JSX-iks. JSX lisab XML laadse koodistiili JavaScriptile. React Native-it saaks kirjutada ka ilma JSX-ita, puhtas JavaScriptis, kuid koodi puhtana hoidmiseks on parem kasutada JSX-i[1]. React Native kood on puhas ja lihtne lugeda, sest see hoiab andmed loogikast eraldi. Siit tulenevalt kasutatakse React

[1] http://buildwithreact.com/tutorial/jsx

Native-is *pure functione* e. puhtaid funktsioone, mis tähendab, et funktsiooni sisendist sõltub tema väljund[1].

React Native rakenduse seisund sõltub ainult andmetest, see tähendab, et komponente uuendatakse ainult sellisel juhul, kui andmetes on toimunud muutus. Samuti tuleb React Native-it kasutades koostada oma rakendus just React-i komponentidest. React-i komponentide reegliks on see, et andmed liiguvad alati ülevalt alla, ehk iga n-ö emakomponent annab edasi vajalikud andmed lapskomponentidele e. alamkomponentidele, nii hoitakse andmete liikumine puhta ja loogilisena[2]. Kui serverist saadetakse vastus uuendatud kulleri koordinaatidega, siis React kontrollib, kas koordinaadid on ikka muutunud. Juhul kui ei ole, ei käivitata rakenduse loogika, aga kui on erinevad, uuendatakse ainult neid komponente, mis sõltuvad koordinaadi andmetest. Klienti tellimuse sihtkoha ning kulleri asukoha kuvamiseks ning uuendamiseks kasutatakse Pitsakulleri rakenduse puhul Reactile Native-ile mõeldud teeki, mis muretseb ise Google Maps-i API suhtluse eest. Selle kasulikkus seisnebki selles, et teegis on projektile vajalik funktsionaalsus juba olemas.

Kui kasutajaliides on valmis, tuleks edasi liikuda serveri ja andmebaaside arenduse juurde. Selleks on meil arenduskeskkonda vaja installeerida PHP teenused, et PHP skripte saaks käivitada. Kuna server toimib läbi domeeni, siis serveri arenduses tuleks samuti kasutada läbi domeeni arendust, selleks installeerime NGINX teenuse, mis käivitab PHP skripte läbi domeeni ning tagastab vajaliku väljundi. Sellega saab luua endale test domeeni, mis töötab ainult lokaalses võrgus. Ilma domeenita ei oleks võimalik ühendada kasutajaliidest serveri API-ga. Domeeni abil luuakse ühendus õige serveriga (NGINX-iga) ja NGINX vaatab domeeni nime, millelt päring tuli ning suunab selle edasi nüüd õigesse PHP skripti.

Peale selle tuleb veel installeerida projektis kasutatav PostgreSQL andmebaasi teenus, mis peale installeerimist on kohe kasutuskõlblik.

Kui eelnimetatu tehtud, ollakse valmis serveripoolset koodi üles seadma. Selleks on vaja Composerit, millega saab kiirelt ja mugavalt PHP paketid alla laadida, paketid nagu

[1] http://www.sitepoint.com/functional-programming-pure-functions/

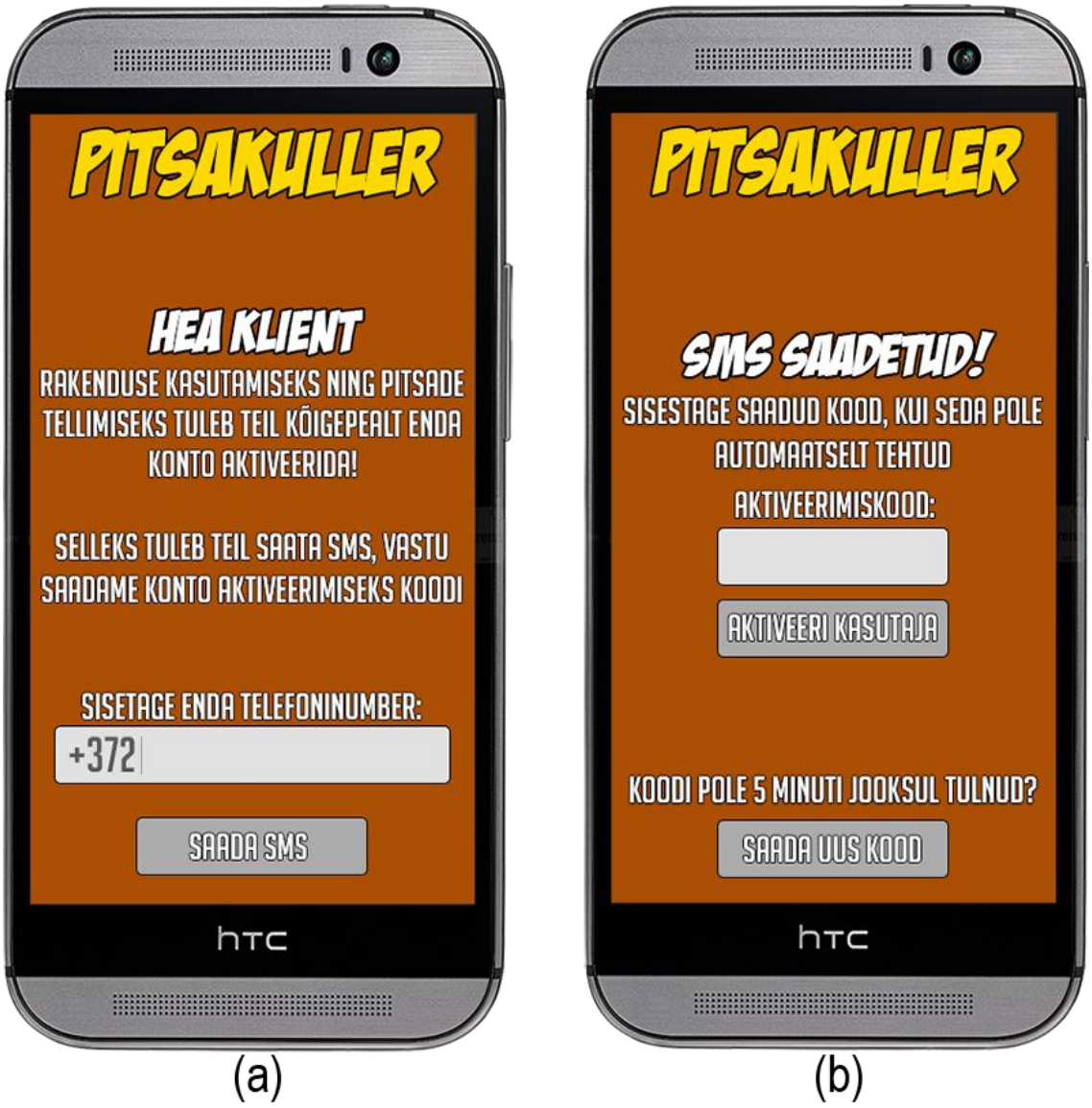
[2] https://facebook.github.io/react/docs/thinking-in-react.html

Laravel. Laravel-iga tuleb kaasa Artisan tehnoloogia, mille eesmärgiks on luua andmebaasitabeleid ja suhteid. Selleks loome vajalikud migratsioonifailid, igale andmebaasitabelile, mis andmebaasimudelil (Joonis 4) märgitud. Luua tuleb vastav migratsioonifail ja ära märkida tabeli väljad, võtmed ja suhted. Kui need on tehtud, käivitatakse terminalis Laravel-i Artisan-i migratsioonikäsklus, millega luuakse terve andmebaas migratsioonide alusel.

Edasi liigutakse API loomisele, mille ülesanne on vastu võtta kliendirakenduse poolt saadetud sisendeid ning uuendada tema rakenduses olevaid andmeid. Pärast API loomist on kliendil võimalik registreerida/aktiveerida kasutaja, mis kantakse andmebaasi, on võimalik saada infot kullerite asukohtadest, müüdavatest toodetest ja esitada tellimusi. Samuti on integreeritud ka SMSAPI teenus, põhimõtteliselt üks API suhtleb teisega. Serverirakendus (serveripoolne kood) teeb päringu SMSAPI-le juhul, kui on registreeritud uus kasutaja, ja sellele numbrile, mis klient sisestas saadetakse kinnituskood, mis peab kliendilt tagasi serverisse tulema kasutaja aktiveerimiseks.

6.2 Kasutajaliidese prototüüp

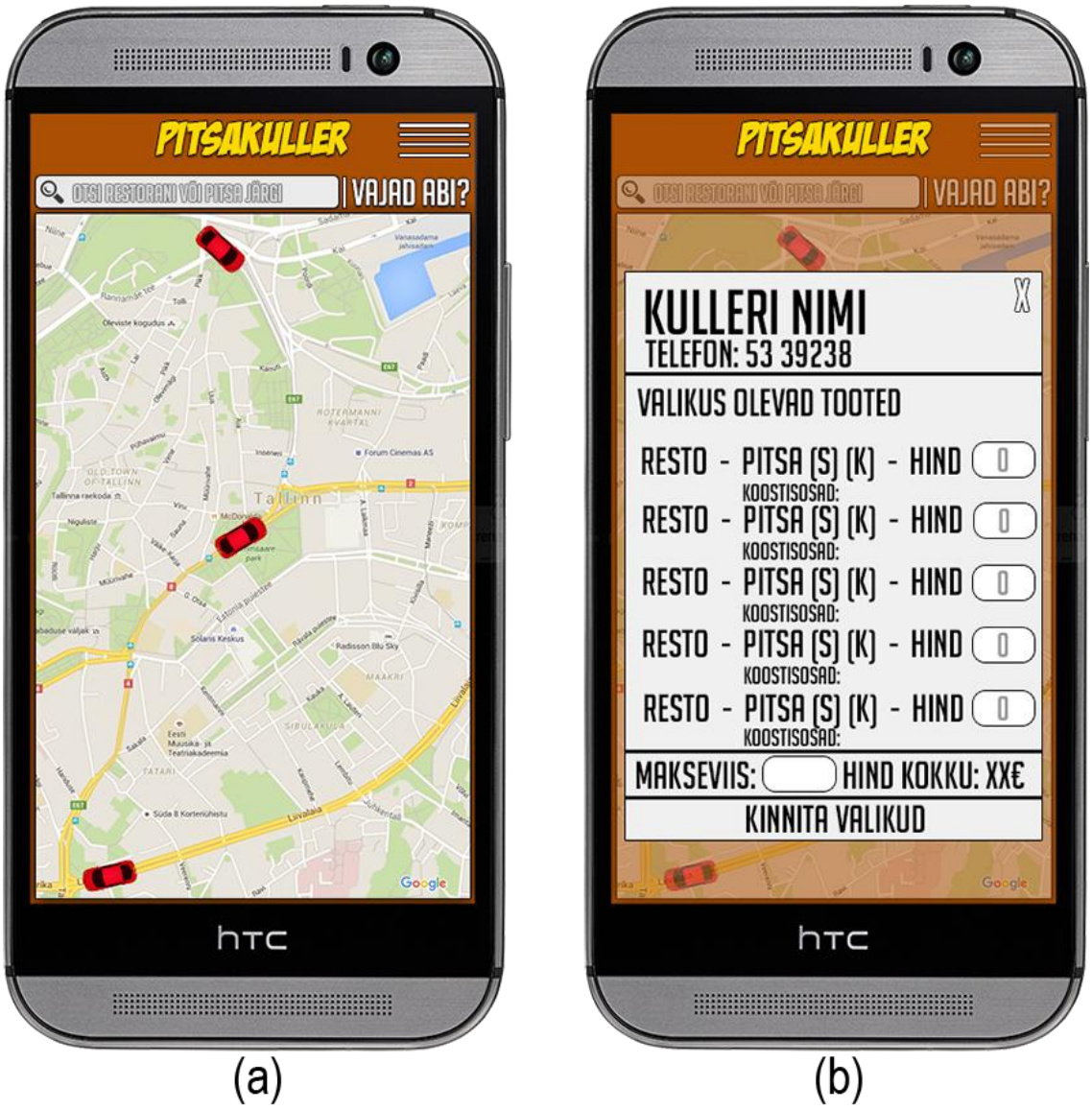
Antud peatükis esitleb autor graafilise kasutajaliidese prototüübi erinevaid vaateid (Joonis 6 ja Joonis 7). Prototüübi vaated on koostatud vastavalt eelmistes peatükkides kirjeldatud funktsioonidele.



Joonis 6. Pitsakulleri rakenduse aktiveerimise kahe etapi vaated

Jooniselt 6 on näha kliendirakenduse aktiveerimise 2 eri vaadet, telefoni pilt (a) on esimene vaade, mida klient rakendust käivitades näeb. Kliendil tuleb Pitsakulleri rakenduse ning -teenuste kasutamiseks enda konto aktiveerida. Selleks peab klient sisestama enda telefoninumbri ning vajutama "Saada SMS" nuppu.

Parempoolne pilt telefonist, pilt (b), ilmub kliendile peale SMS-i saatmist. Kliendile edastatakse teade, et SMS on saadetud ning järgmise sammuna tuleb temal see sisestada aktiveerimiskoodi kasti, juhul, kui rakendus seda mingil põhjusel ise pole automaatselt teinud. Soovituslik sõnumi maksimaalne ooteaeg on 5 minutit, peale seda tuleks kliendil uus sõnum saata.



Joonis 7. Pitsakulleri rakenduse (a) kaardi vaade ning (b) tellimise akna vaade

Joonisel 7 on esitatud 2 erinevat kaardivaadet. Vasakpoolne pilt (a) avaneb Pitsakulleri kliendile peale edukat aktiveerimist, kliendile kuvatakse hetkel aktiivsed olevad kullerid. Kliendil on võimalik autode peale vajutada, peale mida avaneb neile parempoolse pildi (b) vaade. Kliendile kuvatakse kulleri nimi, telefon ning hetkel valikus olevad tooted koos toodete kirjeldusega. (S) tähistab suuruse tähist ning (K) on koguse tähis. Kliendil on võimalik valida rohkem, kui üks toode korraga, kuid mitte rohkem, kui koguses märgitud. Pildil (b) hinna järgi olevasse kasti märgib klient soovitud pitsa koguse, mis on vaikimisi 0. Peale valikuid kuvatakse talle kogu tellimuse hind ning kliendil on võimalik valikud kinnitada. Kuid samuti on kliendil võimalik tellimus tühistada, vajutades pildil (b) üleval paremal nurgas olevat "X" nuppu.

7 Kokkuvõte

Käesoleva bakalaureusetöö autori eesmärgiks oli Pitsakulleri mobiilirakenduse loomeprotsessi kirjeldamine läbi infosüsteemi detailse analüüsi ning kirjelduse. Teiseks eesmärgiks oli esitada vastavalt projekti kirjeldusele loodud kasutajaliidese prototüüp.

Pitsakulleri mobiilirakenduse loomeprotsessi kirjeldamisel tõi autor välja Pitsakulleri rakenduse analüüsi ja arhitektuuri joonise, süsteemi platvormi analüüsi, andmebaasi mudeli ning kirjelduse. Samuti esitles autor vastavalt Pitsakulleri rakenduse funktsioonidele koostatud kasutajaliidese prototüüpi ning samm-sammulist rakenduse arenduskäiku.

Peatükis 2 selgus, milline hakkab olema Pitsakulleri rakenduse üldine funktsionaalsus. Autor esitles rakenduse arhitektuurse joonise ning kasutusjuhtude- ja tegevusdiagrammi, kirjeldamaks süsteemi tööd.

Peatükis 3 sai lugeja aimu Pitsakulleri rakenduse süsteemi platvormist. Autor kirjeldas valikus olevaid serverisüsteeme. Peale analüüse ja erinevate teenusepakkujate võrdlemist langes autori valik Zone Media pilveserveri kasuks. Operatsioonisüsteemiks valis autor Linux-i Ubuntu.

Peatükis 4 esitles autor Pitsakulleri rakenduse andmebaasi mudelit ning andmebaasisüsteemi. Peale võrdlusi ning analüüse osutus autori valikuks PostgreSQL andmebaas. Autori poolt väljatoodud andmebaasisüsteemid on kõik väga tasemel ning suurt viga nende kolme valimisel teha ei ole võimalik. Andmebaasi mudel on koostatud vastavalt rakenduse funktsioonidele ning näitab täpselt, millised tabelid on omavahel seotud.

Kõige mahukamas peatükis 5 tõi autor välja vajalikud süsteemi tarkvara osad ning tööriistad. Valitud veebiserveriks osutus NGINX ning Pitsakulleri rakenduse kood kirjutatakse PHP-s ja JavaScript-is. Lugeja sai teada, kuidas aitavad mobiilirakenduse loomisele kaasa PHP ja JavaScripti tugiraamistikud Laravel ja React Native ning mis ülesanded on tööriistal nimega Git.

Peatükis 6 kirjeldas autor rakenduse samm-sammulist arenduskäiku, millises järjekorras ning mida teha tuleks. Samuti esitas autor projekti plaane ja funktsioone arvestades kasutusjuhtude diagrammi ja tegevusdiagrammi. Viimase teemana tuli esitlusele Pitsakulleri kasutajaliidese prototüüp, mis valmis kõike eelnevat arvestades.

Põhjalikule infosüsteemi erinevate valdkondade analüüsimisele ning uurimusele toetudes suutis autor lugejale esitada Pitsakulleri mobiilirakenduse infosüsteemi analüüsi. Autor kirjeldas kogu süsteemi loomist, alustades nullist, kuni kogu süsteemi üles seadmiseni ning sellega sai käesoleva töö eesmärk täidetud.

Kasutatud kirjandus

[1] When to use cloud platforms vs. dedicated servers - IT World, Matthew Mombrea [WWW] http://www.itworld.com/article/2832631/cloud-computing/when-to-use-cloud-platforms-vs--dedicated-servers.html (29.03.2016)

[2] Pilveserver vs füüsiline server - ProIT, Hannes Mulla [WWW] http://www.proit.ee/pilveserveri-plussid-miinused/ (29.03.2016)

[3] http://euroakadeemia.ee/materjalid/ABp%F5him%F5isted-slaidid.ppt (01.04.2016)

[4] Toomas Mikli. 1998. Sissejuhatus Infosüsteemidesse. ISBN 9985-59-061-9. TTÜ Kirjastus.

[5] If you were to start a new project now (March 2015), would you use MySQL, MariaDB or PostgreSQL and why? - Quora [WWW] https://www.quora.com/If-you-were-to-start-a-new-project-now-March-2015-would-you-use-MySQL-MariaDB-or-PostgreSQL-and-why (20.05.2016)

[6] What are the best open-source relational databases for high performance web applications?[WWW]http://www.slant.co/topics/180/compare/~postgresql_vs_mariadb_vs_mysql (20.05.2016)

[7] Git - Git [WWW] https://git-scm.com (06.04.2016)

[8] About, Staging Area - Git [WWW] https://git-scm.com/about/staging-area (06.04.2016)

[9] Nginx - Nginx [WWW] https://www.nginx.com/resources/wiki/ (19.04.2016)

[10] What are the pros and cons of PHPDesigner and PhpStorm? - Stackoverflow.com [WWW] http://stackoverflow.com/questions/7480660/what-are-the-pros-and-cons-of-phpdesigner-and-phpstorm (20.05.2016)

[11] What's the best editor/IDE for PHP? - Quora [WWW] https://www.quora.com/Whats-the-best-editor-IDE-for-PHP (20.05.2016)

[12] What is JavaScript? - Mozilla Developer Network [WWW] https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (12.05.2016)

Lisa 1 – Kasutusjuhtude diagrammi dokumentatsioon

Tabel 1. Pitsakulleri rakenduse kasutusjuhtude diagrammi dokumentatsioon Joonisele 2

Kasutusjuht: Ühenduse kontroll **Tegutsejad:** Klient **Eeltingimused:** Klient on rakenduse käivitanud. **Kirjeldus:** Rakenduse kasutamine eeldab kliendipoolse rakenduse ühenduse olemasolu serveriga. Ühenduse puudumisel proovib süsteem ühe korra uuesti ühendada, ebaõnnestumisel rakenduse töö lõppeb. Kui rakenduse kasutamise käigus ühendus ära katkeb, proovitakse samuti ühe korra uuesti ühendada, ebaõnnestumisel tühistatakse pooleliolevad toimingud ning rakendus lõpetab töö.
Kasutusjuht: Verifitseeri kasutaja **Tegutsejad:** Klient **Eeltingimused:** Ühendus rakendusega on olemas. **Kirjeldus:** Kontroll, kas kasutaja telefoninumber on andmebaasis olemas. Rakenduse esmasel käivitamisel või mõnel muul põhjusel telefoninumbri andmebaasist puudumise korral on kliendil võimalik endale aktiveerimisekood sõnumi teel küsida.
Kasutusjuht: Aktiveeri kasutaja **Tegutsejad:** Klient **Eeltingimused:** Kliendi telefoninumbrit andmebaasis ei ole, tegu on uue kasutajaga. **Kirjeldus:** Kliendile saadetakse konto aktiveerimiseks SMS aktiveerimiskood. Kliendi konto unikaalseks tunnuseks (ID-ks) on tema telefoninumber.
Kasutusjuht: Vali kuvatud kaardilt kuller **Tegutsejad:** Klient, kuller **Eeltingimused:** Kliendi konto on edukalt aktiveeritud ja/või andmebaasis olemas. **Kirjeldus:** Peale edukat konto aktiveerimist kuvatakse kliendile kaart hetkel aktiivsete kulleritega.
Kasutusjuht: Vali toodete transpordi sihtkoht **Tegutsejad:** Klient, kuller **Eeltingimused:** Kliendile on kuvatud kaart aktiivsete kulleritega. **Kirjeldus:** Peale kaardi kuvamist ning enne tellimuse koostamist peab klient valima toodete transpordi sihtkoha. Süsteem salvestab kliendi asukoha ning edastab selle kullerile peale kliendi tellimuse kinnitamist, koos kliendi valitud toodetega.
Kasutusjuht: Koosta tellimus **Tegutsejad:** Klient, kuller **Eeltingimused:** Klient on valinud nii kulleri, kui ka toodete transpordi sihtkoha.

Kirjeldus: Peale kulleri valikut asub klient tellimust koostama. Kliendil on võimalik tellimus tühistada ja rakenduse töö lõpetada või peale tühistamist valida uus kuller. Et koostatud tellimust edastada, peab klient kõik valitud tooted ja makseviisi kinnitama ning seejärel tuleb kinnitada kogu tellimus. See on vajalik kullerile teate saatmiseks.
Kasutusjuht: Tühista tellimus **Tegutsejad:** Klient **Eeltingimused:** Klient on alustanud tellimuse koostamist. **Kirjeldus:** Kliendil on võimalik tellimus tühistada, kullerile sellest vastavalt teadet ei laeku. Peale tühistamist on kliendil võimalik valida uus kuller või rakenduse töö lõpetada.
Kasutusjuht: Vasta tellimusele **Tegutsejad:** Kuller, klient **Eeltingimused:** Klient on tellimuse kinnitanud. **Kirjeldus**: Kui klient on tellimuse kinnitanud, esitatakse see kliendi poolt valitud kullerile. Kuller peab nüüd tellimusele vastama, kas keeldumisega või aktsepteerimisega. Kliendile saadetakse teade, kas tellimus on vastu võetud või tühistatud.
Kasutusjuht: Jälgi tarnet **Tegutsejad:** Klient, kuller **Eeltingimused:** Kuller on tellimuse aktsepteerinud ning toodete transport on alanud **Kirjeldus**: Kliendil on võimalik jälgida kulleri hetkelist asukohta kaardil, asukohta uuendatakse iga kolme sekundi tagant.
Kasutusjuht: Uuenda tarneaega **Tegutsejad:** Kuller, klient **Eeltingimused:** Toodete transpordiga on alustatud. **Kirjeldus**: Kulleril on võimalus eeldatavat tarneaega vastavalt vajadusele muuta. Näiteks, ootamatutest ummikutest tuleneva ajakulu tõttu.
Kasutusjuht: Maksa toodete eest **Tegutsejad:** Klient, kuller **Eeltingimused:** Tooted on kliendi valitud sihtkohta transporditud. **Kirjeldus**: Kliendil on võimalik toodete eest maksta kahel viisil - kaardiga ja sularahas. Maksmine on kliendile toodete kättesaamiseks kohustuslik. Kaardimakse korral toimub maksmine läbi makseterminali. Kui maksmine õnnestus, saab kuller vastava teate ning tellimuse saab lõpetada.
Kasutusjuht: Lõpeta tellimus **Tegutsejad:** Kuller, klient **Eeltingimused:** Toodete eest on makstud. **Kirjeldus**: Kui toodete eest on tasutud, annab kuller kliendile tooted ning lõpetab hetkel aktiivse tellimuse.

Lisa 2 - Tegevusdiagrammi dokumentatsioon

"O" tähistab otsust.

Tabel 2. Pitsakulleri rakenduse tegevusdiagrammi dokumentatsioon Joonisele 3

Tegevus	**Tegevuse kirjeldus**
O1	Ühenduse puudumisel proovitakse ühe korra uuesti serveriga ühendada. Ühenduse saavutamisel alustatakse kasutaja verifitseerimisega.
Proovi uuesti ühendada	Kui esimesel korral rakendusel serveriga ühendust saada ei õnnestunud, proovib ühe korra veel. Ebaõnnestumise korral rakendus lõpetab töö.
O2	Ühenduse puudumisel rakendus lõpetab töö. Ühenduse saavutamisel alustatakse kasutaja verifitseerimisega.
Kasutaja verifitseerimine	Kui ühendus serveriga olemas, toimub esimese tegevusena kasutaja verifitseerimine. Kontrollitakse telefoninumbri olemasolu andmebaasis.
O3	Kui number on andmebaasis olemas, kuvatakse kaart kulleritega. Kui numbrit ei ole, peab klient sisestama enda telefoninumbri ning vastavale numbrile saadetakse SMS aktiveerimiskoodiga.
Aktiveeri konto	Kasutaja peab aktiveerimiskoodi sisestama vastavale väljale, kui seda mingil põhjusel automaatselt tehtud ei ole.
Aktiveerimise kontroll	Kontroll, kas kasutaja on enda telefoninumbri aktiveerinud. Kui aktiveerimist pole 5 minuti jooksul toimunud, siis avaneb kliendil võimalus saata uus SMS.
O4	Tõrke korral veateade ning kasutajal on võimalik uuesti SMS koodi küsida. Eduka aktiveerimise korral kuvatakse kaart kulleritega.
O5	Kliendil võimalik küsida uus aktiveerimiskood vajutades "Saada uus kood" nuppu või lõpetada rakenduse töö.
Kuva kaart kulleritega	Kliendile kuvatakse kaart hetkel aktiivsete kulleritega.
Vali toodete transpordi sihtkoht	Peale kaardi kuvamist valib klient toodete transpordi sihtkoha. Vaikimisi pakutakse kohta, kus klient ise hetkel asub aga tal on võimalik kohamärget kaardil liigutada vastavalt tema soovile.
Vali kuller	Klient valib kaardilt sobiva kulleri vajutades valitud auto peale.
Kuva kulleri tooted	Peale kulleri (auto) ikoonile vajutamist kuvatakse kliendile valitud kulleri tooted.

Tegevus	**Tegevuse kirjeldus**
Selekteeri tooted	Klient selekteerib tooted enda poolt valitud kulleri toodetest.
O6	Kui tooted selekteeritud, tuleb kliendil valida makseviis. Kui tooteid ei selekteerita, on kliendil võimalik valida uus kuller.
O7	Kliendil võimalik valida kaardilt uus kuller või lõpetada rakenduse töö.
Kinnita tellimus	Kui tooted selekteeritud, kuvatakse kliendile toodete hind kokku ning peale makseviisi valikut peab klient kinnitama valikud. Tal on võimalik ka tellimus tühistada vajutades "X" nuppu.
O8	Kui tellimus kinnitatud saadetakse see kullerile, kes peab sellele vastama. Kui tellimust ei kinnitatud, tühistatakse tellimus ning rakendus lõpetab töö.
Saada kullerile tellimus	Kullerile saadetakse kliendi soovitav tellimus koos kliendi asukohaga.
O9	Kulleril on võimalik tellimus vastu võtta või tellimusest keelduda. Kui kuller ei võta tellimust, saab klient veateate. Kui kuller vastab tellimusele, alustatakse tarnega.
Jälgi tarnet	Kliendil on reaalajas võimalik kaardilt jälgida valitud kulleri asukohta ning eeldatavat saabumise aega.
Vali makseviis	Kui tooted kohal, valib klient makseviisi ning tasub toodete eest. Kliendil on võimalik makseviisi muuta vastavalt vajadusele.
O10	Kliendil on võimalik valida kaardimakse või sularahaga maksmise vahel. Kohapeal on kliendil võimalik makseviisi vahetada vastavalt vajadusele.
O11	Kui toodete eest on makstud, lõpetatakse tellimus ning rakendus lõpetab töö. Kui toodete eest ei ole makstud, on kliendil võimalik uuesti proovida või tellimus tühistada.
O12	Klient proovib uuesti tasuta või tühistab tellimuse.
Lõpeta tellimus	Kui toodete eest edukalt tasutud, lõpetatakse tellimus.

Lisa 3 - Andmeedastusmahu esialgsed arvutused

Hetkel saab teha ainult esialgseid arvutusi, arvestades seda, mis rakenduses käigus hakkab olema. Maksimaalseks päise suuruseks valis autor 4,096 baiti, sinna sisse kuuluvad siis brauseri informatsioon nagu kuupäev, küpsised jms. 4,096 baidi sisse mahub kõik vajalik, koos varuga. Ühe karakteri keskmiseks suuruseks tuleks 2 baiti, kuna kasutatakse ka {}, [] ning " märke. ASCII tabeli järgi on tavalised karakterid 1 bait (8 bitti) ning kasutatavad sümbolid 4 baiti. Vastavalt Google Maps-ile on pikkus- ja laiuskraadi jaoks vaja reserveerida 11 numbrit.

Ütleme, et klient näeb kaardil nelja kullerit ning halvimal juhul on tellimuse pikkuseks 20 minutit ja klient otsustab terve tellimuse aja vaadata ekraani. See tähendab, et klient küsib kõigi kullerite asukohti iga kolme sekundi tagant ning kulleri asukohaga käivad alati kaasas ka kulleril küljes oleva informatsioon. Esialgse koodi järgi (esitatud Lisas 3 Joonis A) on kulleril ID, tema asukoht, pitsade nimed, koostisosad, kogused, suurused ning hind. Koodis on välja toodud 3 erinevat pitsat, kogusega 2, millel on 6 koostisosa. Selline kood sisaldab endas 735 tähemärki, ehk 1,470 baiti. Arvestada tuleb ka muutuvate toodetega ja tuleviku koodi uuendustega, seega igaks juhuks anname varuks veel 1,000 baiti ja lisame juurde päise suuruse. Seda kõike arvestades on ühe kliendi päringu suuruseks 1,470 + 1,000 + 4,096 = 6,566 baiti.

Nagu eelnevalt mainitud, siis klient näeb nelja kullerit korraga ning otsustab tellimust täies pikkuses jälgida. Nelja kulleri andmeid uuendatakse kliendile iga 3 sekundi tagant, seega 4 x 6,566 = 26,264 baiti ning ühes minutis kasutab klient 20 x 28,912 = 525,280 baiti. Kui tellimuse pikkuseks on 20 minutit, siis kogu tellimuse peale kasutab klient ära 20 x 578,240 = 10,505,600 baiti ehk 10,5MB. Kui samuti käituksid veel 100 klienti, siis kuluks 20 minuti jooksul 1,050MB ehk 1.05GB, mis on üks kolmandik sellest, mida Zone jagab ühes tunnis tasuta. Sellises mahus andmeedastusmahu kasutamine on halvim juht, kui 100 aktiivset klienti otsustaksid terved 20 minutit ekraani vaadata ning uuendusi pärida. Arvutused näitavad, et server suudab sellega toime tulla. Kuid, kui peakski lisa mahtu vaja olema, siis Zone-i lisa 1GB maksab 0.05€, mis on taskukohane.

```
{
     "id": 999,
"location": {
     "long": -100.000000,
  "lat": -100.000000
},
"pizzas": [
     {
     "id": 99999,
    "name": "Hakkliha pizza kolmejuustuga",
    "contents": [
           "Hakkliha",
           "Paprika",
           "Jalapeno",
           "Mozzarella",
           "Talu juust",
           "Mingi juust veel"
    ]
    "quantity": 2,
    "Prize": 00.00,
    "sizes": [
           "M",
           "XL"
    ]
  },
    "pizzas": [
     {
     "id": 99999,
    "name": "Hakkliha pizza kolmejuustuga",
    "contents": [
           "Hakkliha",
           "Paprika",
           "Jalapeno",
           "Mozzarella",
           "Talu juust",
           "Mingi juust veel"
    ]
    "quantity": 2,
    "Prize": 00.00,
    "sizes": [
           "M",
           "XL"
    ]
  },
    "pizzas": [
     {
     "id": 99999,
    "name": "Hakkliha pizza kolmejuustuga",
    "contents": [
           "Hakkliha",
           "Paprika",
```

```
            "Jalapeno",
            "Mozzarella",
            "Talu juust",
            "Mingi juust veel"
      ]
      "quantity": 2,
      "sizes": [
            "M",
            "XL"
      ]
    }
  ]
}
```

Joonis A. Esialgne kliendi päringu kood