TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nils-Emil Lille 155082IAPB

KLIENDIPOOLSE TELLIMUSTE ESITAMISE VEEBILAHENDUSE LOOMINE

Bakalaureusetöö

Juhendaja: Inna Švartsman
MSc

Tallinn 2021

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: Nils-Emil Lille

18.05.2021

Annotatsioon

Antud bakalaureusetöö eesmärgiks on luua üks osa veebilahendusest, läbi mille saab rakenduse kasutaja ilma oma nutiseadmesse rakendust installeerimata tellida teenindust ja pakutavaid tooteid.

Töö käigus analüüsitakse teostuseks kõige sobivamaid tehnoloogiaid, juurutamist ja funktsionaalseid nõudeid. Analüüsi tulem teostatakse.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 7 peatükki, 10 joonist, 3 tabelit.

Abstract

Developing a Client-side Web Solution for Order Placement

The purpose of this bachelor's thesis is to create a part of a web solution, through which the user of the application can order service and offered products without installing any applications on his/her smart device.

In the course of the work, the most suitable technologies, deployment and functional requirements are analysed. The result of the analysis is implemented as an application.

The thesis is in Estonian and contains 26 pages of text, 7 chapters, 10 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	*Application programming interfac*e, rakendusliides
CSS	*Cascading Style Sheets*, kaskaadlaadistik
Docker	Konteinertehnoloogia
Docker Registry	Repositoorium Dockeri tõmmiste (ingl. image) hoidmiseks ja levitamiseks
EC2	Amazoni pilveserver
Git	Versioonihaldustarkvara
HTML	*Hyper Text Markup*
HTTP	*Hypettext Transfer Protocol*
IDE	*Integrated Development Environment*, arenduskeskkond
JSON	*JavaScript Object Notation*
LTS	*Long Term Support*, pikaajaline tugi
NoSQL	Mitterelatsiooniline andmebaas
ODM	*Object-Document Mapper*
OS	*Operating System*, operatsioonisüsteem
port	Funktsionaalüksus, mille kaudu andmed saavad siseneda võrku või väljuda sellest
QR	*Quick Response Code*, ruutkood
REST	*Representational state transfer*
SQL	*Structured Query Language*, struktuurpäringukeel
SVG	Skaleeritav vektorgraafika
URL	*Uniform Resource Locator*, universaalne ressursilokaator

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Järgnevas peatükis selgitatakse, miks käesoleva tööga on otsustatud tegeleda. Samuti püstitatakse eesmärk, kuidas luua lahendus, mis võimaldaks antud probleemi lahendada, ja selgitatakse, kuidas teostus on struktureeritud.

1.1 Probleem

Restorani või mõnda muud toitlustusasutust külastades on tihtipeale probleem, et kui külastuse käigus tahame oma arve tasuda või tooteid juurde tellida, peame me ootama teenindajat või minema teda ise kutsuma. Mitmed ettevõtted on sarnast probleemi üritanud juba lahendada, kuid kui me igapäevaselt külastame restorane ja kohvikuid, ei jää meile tihti silma nutilahendused, millega meie külastuskogemus oleks kiirem ja mugavam.

Kui me kasutame sarnaseid lahendusi, on enamasti teadlikule kasutajale koheselt arusaadav, et on kasutatud mõne kolmanda osapoole tarkvara ja mõningal juhul kuvatakse rakenduses ka konkureerivate asutuste infot. Rakendus ja lahendus peaksid erinema teistest sarnastest lahendustest selliselt, et rakendust kasutav kasutaja ei tohiks esmapilgul aru saada, et tegemist on üldise lahendusega, mis on rakendatav mitmele asutusele.

1.2 Eesmärk

Käesoleva bakalaureuse lõputöö eesmärgiks on luua osa kesksest süsteemist, mis võimaldab toitlustusasutuste külalistel lihtsamalt ja mugavamalt oma tellimusi esitada.

Rakenduse dünaamiline disainikonfiguratsioon ja juurutamisprotsess peaksid looma võimaluse, läbi mille oleks võimalik rakenduse disaini ühtlustada asutuse üldise disainiga, jättes kasutajale mulje, nagu oleks toode arendatud spetsiaalselt seda kasutavale ettevõttele.

1.3 Teostus

Töö põhiosa on jagatud kolmeks etapiks: analüüsitakse funktsionaalust, arendatakse tarkvara ja testitakse valminud tarkvara. Analüüsi käigus uuritakse, mis peaks olema kliendipoolse kasutajaliidese võimalikud tegevused. Analüüsitakse erinevaid kasutusvoogusid ja võrreldakse kasutusvoogude aspekte. Rakenduse arendamise etapis luuakse vajalik funktsionaalsus, esialgne juurutamise protsess ja kirjeldatakse valikute põhjused. Rakenduse testimise etapis testitakse rakendust mitmest aspektist. Rakenduse funktsionaalsusele luuakse testid ning seda testitakse erinevate töökoormuste all.

2 Analüüs

Järgnevas peatükis analüüsitakse loodava süsteemi kasutajaliidese pakutavat funktsionaalsust kasutajale, võrreldakse olemasolevaid lahendusi ning seatakse nõudeid, millele rakendus peab vastama. Analüüsi käigus tuuakse välja erinevaid alternatiive, missuguseid tegevusi peaks olema võimalik rakenduses teha ja hinnatakse alternatiivide erinevaid aspekte.

2.1 Sarnased lahendused

Apollo Kinorestoranis on toolidesse integreeritud tahvelarvuti, millest saab klient endale tellida O'Learyse menüüs pakutavaid tooteid, kuid sarnane lahendus on väikeettevõtetele kalli alginvesteeringu tõttu väheatraktiivne. Kesklinnas paiknev Bogi Hookah Place on probleemi lahendanud laudadele asetatud tahvelarvutitega, kus on avatud rakendus, läbi mille saab klient teenindajat kutsuda ja menüüst pakutavaid tooteid tellida. Antud lahenduse eelised on, et klient ei pea sisestama lauakoodi ega sarnaseid tuvastamisviise, vaid saab kohe tellida, lisaks on rakendus eelnevalt konfigureeritud ja määratud kindla laua teenindamiseks; puuduseks on tahvelarvutite vajadus igale lauale.

Samuti on olemas seadmeid mittevajavad lahendused, näiteks Ordly [1]. Ordly võimaldab sisestada lauale esitatud koodi veebilehele ning seejärel kutsuda teenindajat, kuid autori kogemusel oli kohe aru saada, et tegemist pole toodet kasutava asutuse loodud lahendusega, vaid kasutatakse teenust. BaBash on telefonirakendus, läbi mille saab valida asutuse ning tellida ja maksta [2]. Rakenduse installeerimine ei tohi autori hinnangul olla funktsionaalsuse kasutamise nõudeks.

2.2 Mittefunktsionaalsed nõuded

Kuna rakenduse põhifunktsionaalsus on tellimuste esitamine ja teenindaja kutsumine reaalajas, siis peab info liikuma rakenduste eri poolte vahel ilma kasutajaliideseid uuesti laadimata ega uut HTTP-päringut algatamata. Arendaja poolt määratud sagedusel andmebaasist info küsimine tekitaks suuremat koormust ja ei ole parim tava reaalajarakenduste loomiseks, seega on rakenduses kasutatud WebSocketit. WebSocket on tehnoloogia, mis võimaldab kliendi- ja serverrakendusel hoida avatuna sessiooni, kus

klient ja lahendus saavad omavahel saata sõnumeid ilma uut HTTP-päringut tegemata [3]. WebSocket on osa HTML5-standardist ja seda toetavad kõik uuemad brauserid [4].

Veebilehtedel kasutatavad küpsised peavad vastama isikuandmete töötlemisele seatud nõuetele (küpsised töötlevad isikuandmeid, sest arvutit loetakse isiku privaatsfääri osaks). Kui rakendusel on plaanis kasutada küpsiseid, siis peavad olema rakenduses kasutatavate küpsiste kohta info, et kasutaja saaks nende kohta teavet [5].

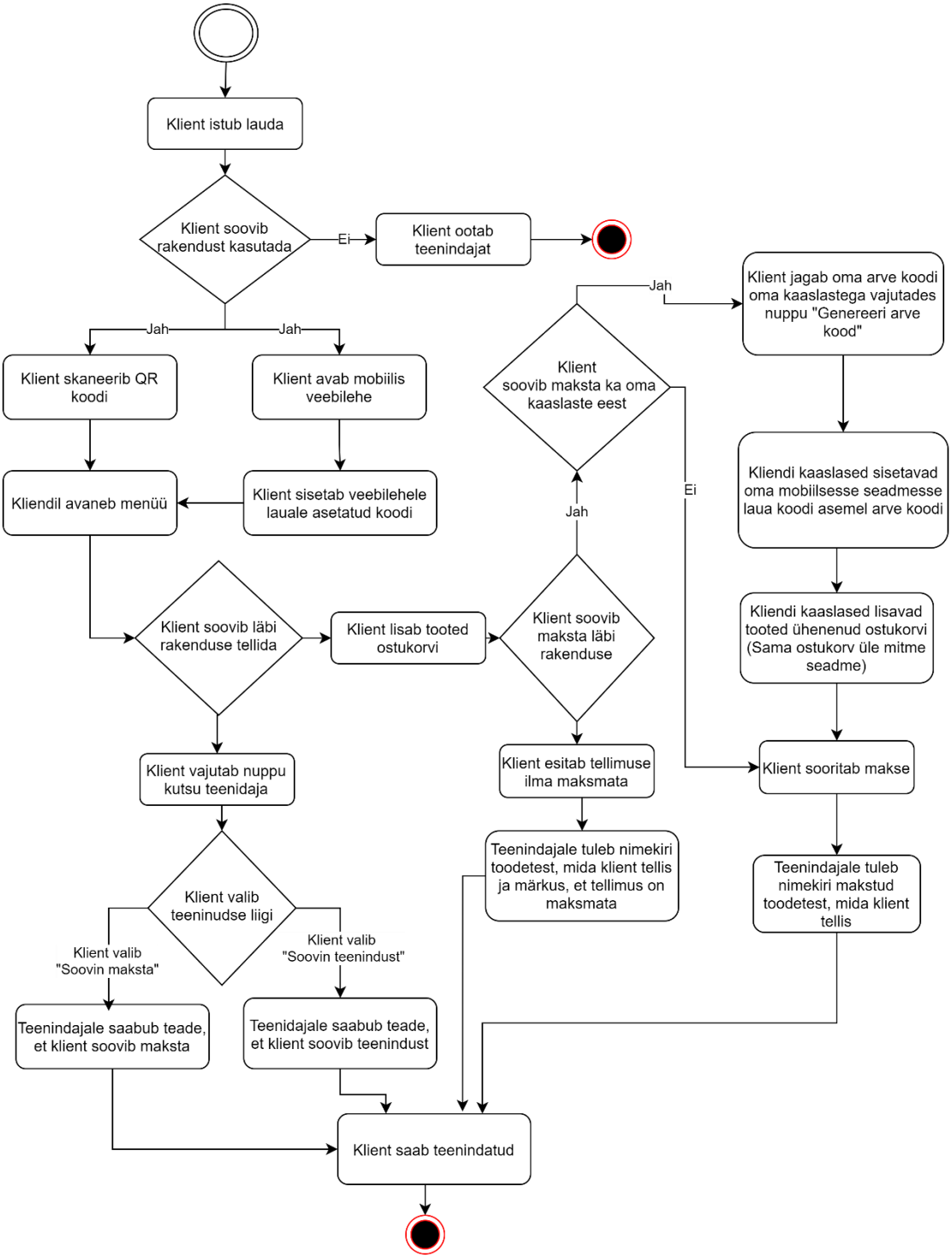
Rakendust hakkab teoreetiliselt kasutama suur hulk inimesi, seega peab rakendus suutma teenindada vajalikul hulgal päringuid. Kasutajaliidese mugavaks kasutamiseks peavad rakendusserveri päringud saama vastuse piisavalt kiiresti, et päringute kiirus ei segaks kasutaja mõttevoolu. Kui päringud on kiiremad kui 0,1 sekundit, siis kasutajale jääb mulje, et rakendus töötab silmapilkselt. Kui päringud on kiiremad kui 1 sekund, siis kasutaja märkab seda, kuid see ei sega tema mõttevoolu [6]. Seega rakenduse päringud võiksid vähesel koormusel olema kiiremad kui 0,1 sekundit ja suuremal koormusel ei tohi päringud olla aeglasemad kui 1 sekund.

2.3 Funktsionaalsuse analüüs

Kui teostataval lahendusel on plaanis teostada rakendusesisest maksmist, siis peab olema selge, kuidas töötab maksmine ja mis riskid võivad esineda rakendusesisesel maksmisel. Kui klient maksab läbi rakenduse, siis peab tehingu tagastamine olema tõestatav, seega ei tohi rakendusesisese makse puhul tagastada tehingut sularahas, vaid läbi rakenduse. Kui klient üritaks tehingut vaidlustada, väites, et summa pole temani jõudnud, siis automaatse tagastuse puhul on seda lihtne ja kiire tõestada [7].

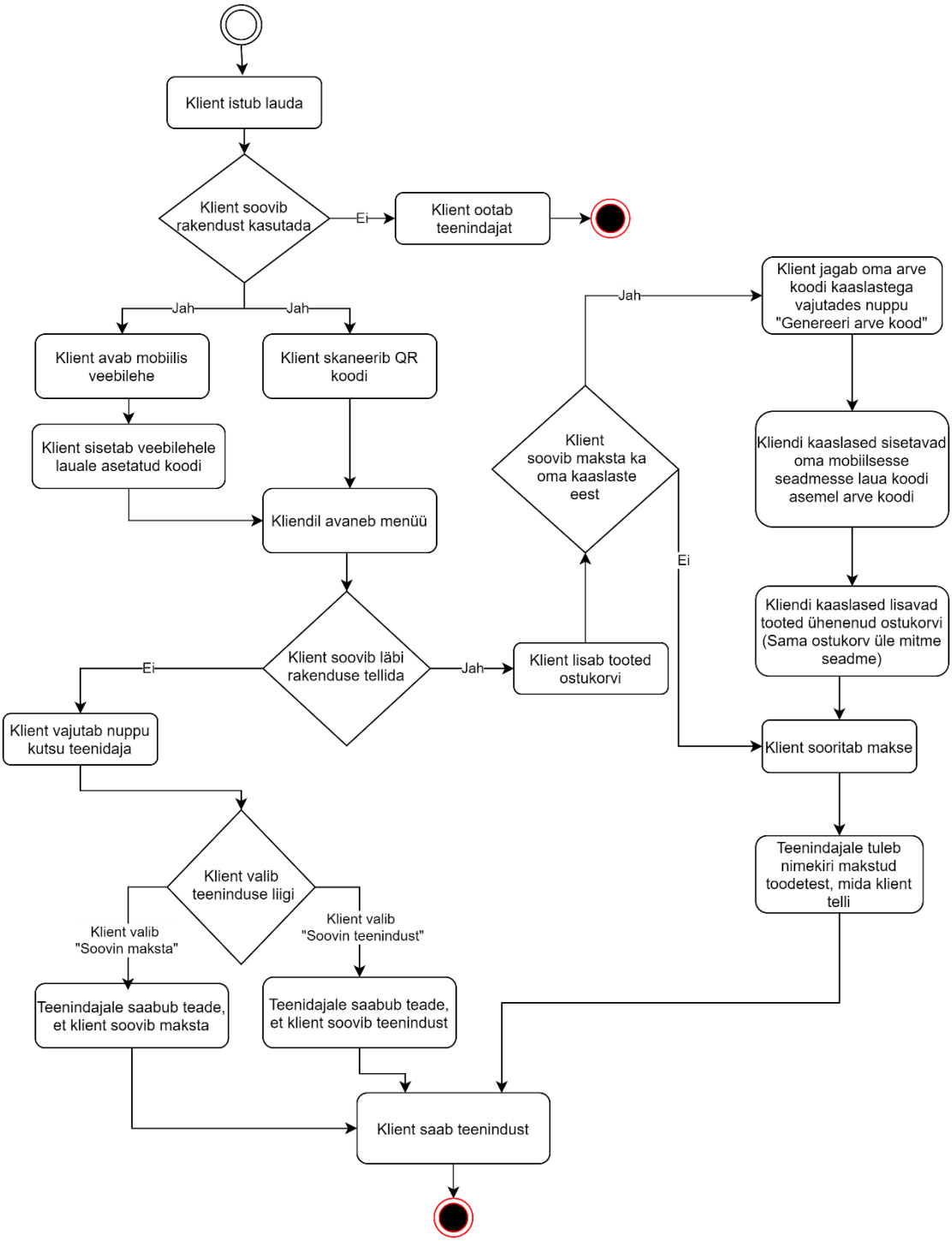
Selleks, et antud rakendus oleks kasutatav, peab see andma ettevõtetele lisaväärtust. Kui klient saab oma tellimuse koheselt esitada, võib see potentsiaalselt suurendada tellitud toodete hulka. Samuti ei pea klient muretsema, kas teenindaja on teadlik tema teenidusvajadusest, vaid saab keskenda külastuskogemusele.

Tellimuse teostamiseks on mitmeid variante. Kõige keerulisem variant oleks, et kliendil on võimalus tellida tooteid nii, et ta saab nii koheselt kui ka hiljem nende eest maksta. Joonisel 1 on välja toodud kasutajavoog maksmisega ja maksmiseta.



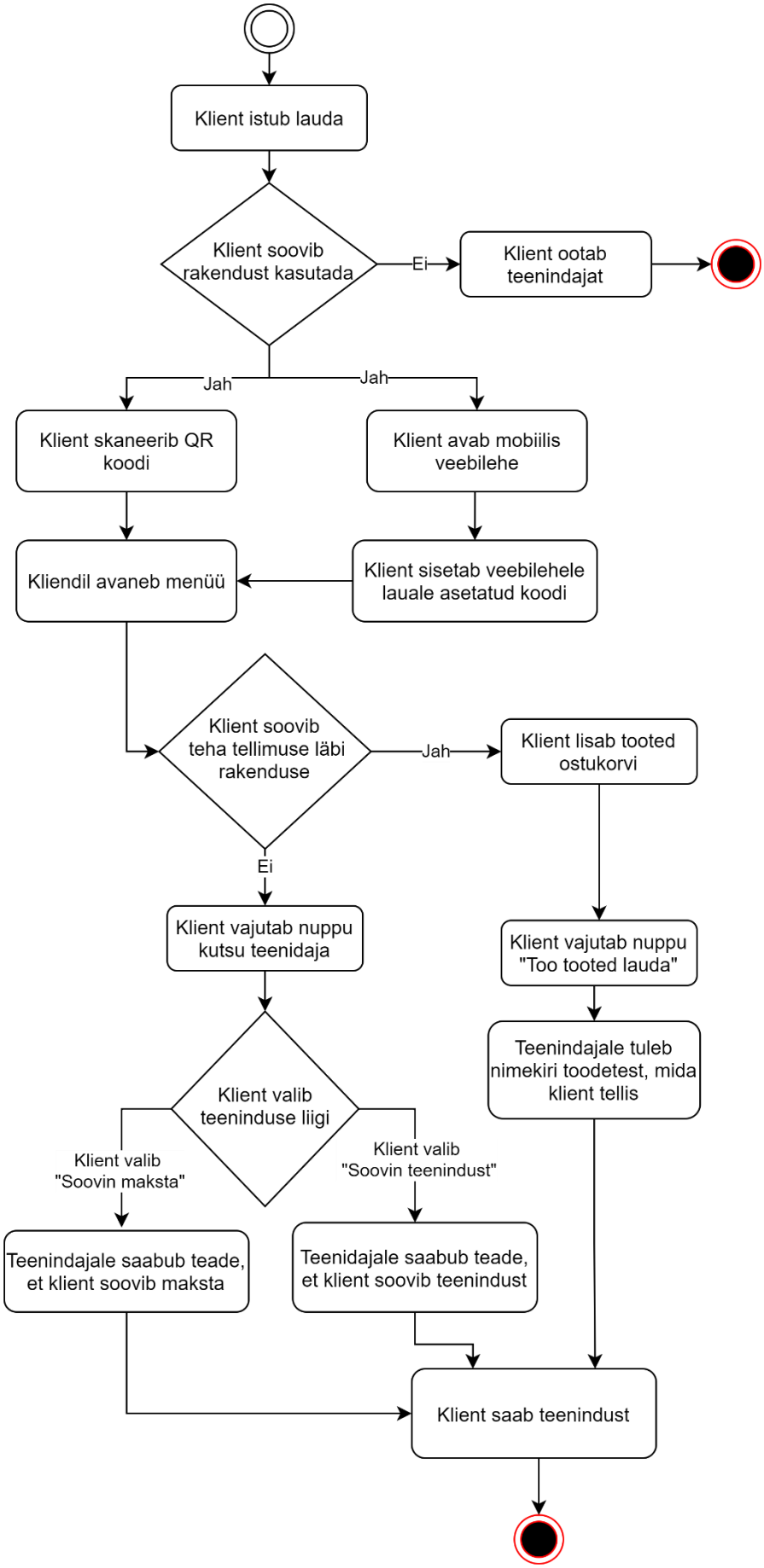
Joonis 1. Kasutajavoog maksmisega ja maksmiseta

Sarnaselt joonisele 1 oleks võimalik tellimine tühistada ilma maksmiseta. Sellisel juhul ei peaks teenindaja jälgima, millised tellimused vajavad maksmist. Ainult maksmisega kasutajavoog on välja toodud joonisel 2.



Joonis 2. Kasutajavoog ainult maksmisega

Kolmas variant oleks teha tellimusi ilma maksmiseta, mis on kujutatud joonisel 3.


Klient istub lauda
Klient soovib rakendust kasutada
Ei
Klient ootab teenindajat
Jah
Jah
Klient skaneerib QR koodi
Klient avab mobiilis veebilehe
Kliendil avaneb menüü
Klient sisetab veebilehele lauale asetatud koodi
Klient soovib teha tellimuse läbi rakenduse
Jah
Klient lisab tooted ostukorvi
Ei
Klient vajutab nuppu kutsu teenidaja
Klient vajutab nuppu "Too tooted lauda"
Teenindajale tuleb nimekiri toodetest, mida klient tellis
Klient valib teeninduse liigi
Klient valib "Soovin maksta"
Klient valib "Soovin teenindust"
Teenindajale saabub teade, et klient soovib maksta
Teenidajale saabub teade, et klient soovib teenindust
Klient saab teenindust

Joonis 3. Kasutajavoog maksmiseta

Nii maksmisega kui maksmiseta tellimuse esitamise eeliseks oleks suurem valikuvõimalus kliendile, kuid see muudaks teenindaja jaoks tellimuste haldamist raskemaks, sest teenindaja ei kanna rakenduse kasutamiseks vajalikku seadet alati kaasas.

Kui klient saaks läbi rakenduse maksta, siis selle eeliseks oleks, et klient ei peaks soovi korral üldse teenindajaga suhtlema ja saaks keskenduda oma külastuskogemusele. Puuduseks oleks, et tehtud tehingute pealt läheb maha maksevahendaja tehingutasu ja tarkvara arendamine oleks keerukam.

Kui klient saaks tellida toitu läbi rakenduse ilma maksmiseta, oleks selle puudus teiste variantide ees see, et see annaks vähem lisandväärtust ettevõttele. Eelisteks oleksid selle lihtne arusaadavus ja selgus. Samuti on antud lahendust lihtsam teostada.

Analüüsides kõiki variante on rakenduses otsustatud esialgselt teostada lahendus, kus klient saab asutuse tooteid tellida ilma maksmiseta. Samuti muudab see variant teenindaja jaoks lihtsamaks tellimuste haldamise, kuna juhul, kui klient on toote eest maksnud, peaks klienditeenindaja saama raha tagastada. Kuna maksmist ei toimu, saab klient tellimuse vajadusel lihtsalt kustutada ja ei pea raha tagastamiseks lisategevusi läbi viima. Tagasimakse võimalus tekiks näiteks juhul, kus makstud toode on tellimuse esitamise hetkel küll süsteemis olemas, kuid füüsiliselt otsas.

Kõik tooted tuleb rakendusele lisada läbi haldusliidese, mille lahendus on loodud Karl Hendrik Mae poolt [8]. Kliendil peaks olema võimalik lisatud esemete seast endale valitud tooted tellida või kutsuda teenindust.

Antud kasutuslugude ja lisaväärtuste valideerimiseks oli läbi viidud intervjuu isikuga, kes tegeleb igapäevaselt antud tegevusvaldkonnaga. Intervjuus tõi isik välja, kuidas kliendi võimalus tellimus kiiremini esitada ei suurenda seda, kui kiiresti jõuab toit kliendini, kuna see on enamasti piiratud sellega, kui kiiresti suudab köök tellimusi täita. Läbilaskevõimet antud rakendus isiku arvamusel ei suurendaks, kuid vähendab koormust teenindajatele ehk asutusel oleks võimalik toime tulla väiksema hulga teenindajatega.

3 Teostus

Järgnevas peatükis kirjeldatakse ja põhjendatakse loodud tarkvaralist lahendust.

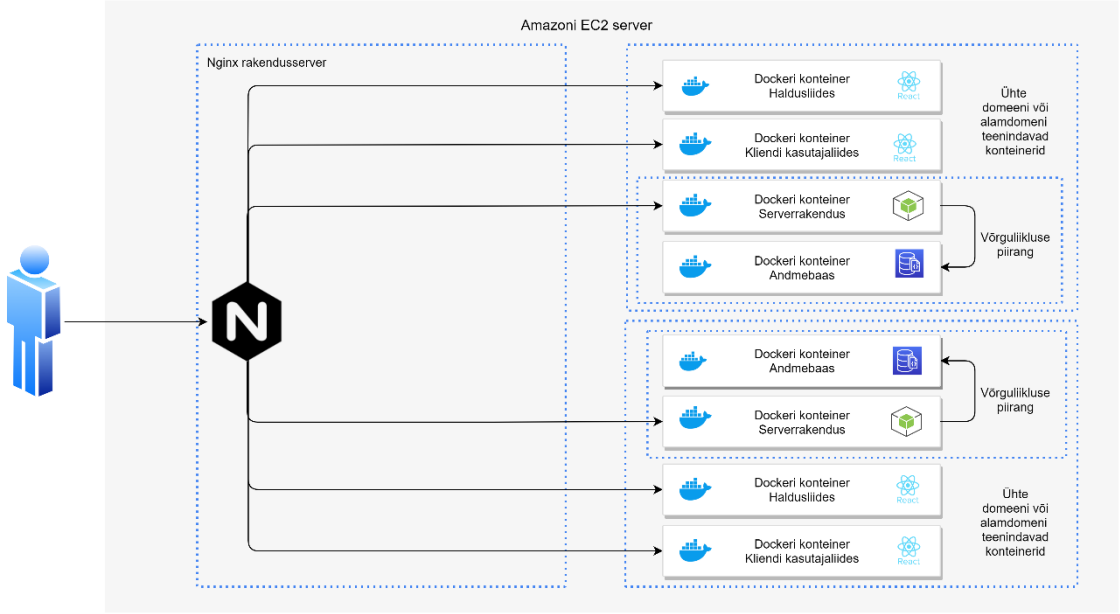
3.1 Taristu

Loodavad rakendused vajavad serverit, kus pakutavad teenused oleksid ligipääsetavad. Projekti arendatakse lokaalses masinas, kuid klientide serveerimiseks kasutatakse Amazoni EC2 pilveserverit. Serveri operatsioonisüsteemiks on valitud Ubuntu 20.04.2 LTS ja töö esitamise hetkel konfigureeritud serveerima sisu https://tellimus.com erinevatelt alamdomeenidelt. Amazon EC2 pilveserverist tuleb pikemalt juttu peatükis 5.1 Amazon EC2.

Kuna toodet kasutavad asutused võivad tahta sisestada samu lauakoode, peab olema võimalik lauakoode erinevate organisatsioonide vahel taaskasutada. Kui kasutaja hakkab lauakoodi sisestama, siis pole infot, millise asutusega see on seotud, seega peab hoidma organisatsiooni laudade info eraldi. Variant oleks eri asutuste andmeid hoida erinevates andmebaasiskeemides. Autor otsustas luua iga kliendi jaoks eraldi andmebaasi, kuna andmebaasiühendust on võimalik võrgutasandil piirata ja sellega väheneb autori hinnangul veaohtlikkus puudulike või vigaste andmete korral. Igale serveris töötavale andmebaasile paikneb seda täiendav serverrakendus. See tekitab võimaluse, kus ei pea rakenduse koodis eristama päringuid asutusepõhiselt.

Kõik juurutatavad rakendused kasutavad paigaldusel konteinertehnoloogiat. Igal asutusel on neli konteinerit, millest igas konteineris on üks järgnev rakendus: haldusliides, kliendi kasutajaliides, andmebaas ning serverrakendus. See võimaldab erinevate serverrakenduste ja andmebaaside suhtlust piirata ning on välistatud koodivea või muude sarnaste vigade tõttu asutuste andmete vahetusseminek. Samuti on seatud võrgupiirang nii, et teise asutuse rakendust ei saaks ühendada temale mittevastava andmebaasi külge.

Igal kliendil on oma valitud domeen või selle domeeni alamdomeen. Nginx puhverserver suunab vastavalt domeeninimele sissetulevad päringud õigetesse konteineritesse. See võimaldab antud lahendust vajadusel hõlpsasti rakendada lahendust telliva asutuse olemasoleva domeeni või alamdomeeni alt. Joonisel 4 on kujutatud näidislahendust kahe asutuse puhul.



Joonis 4. Taristu kahe asustuse puhul

3.2 Juurutamine

Rakenduste juurutamiseks on valitud konteinertehnoloogia, kuna konteinerite uuendamine ja käivitamine töötab samamoodi igas keskkonnas. Autor ehitab konteinertõmmised automaatse skriptiga, tõmmised laaditakse Docker Registrysse üles, seejärel on võimalik tekitatud tõmmised käivitada välisvõrgust ligipääsetavas serveris.

Kuna rakenduse kõik osad vastutavad ainult ühe kliendi eest, peab olema eristatav, kuidas päringud jõuavad ettenähtud rakenduseni. Selleks kasutatakse Nginx pöördpuhverserverit. Joonisel 5 on kujutatud Nginx-konfiguratsiooni eeldefineeritud malli, kus nimekiri kliendi parameetritest käiakse ükshaaval läbi ja malli sees olevad muutujad asendatakse *sed*-käsuga parameetri väärtusega [9].

```
server {
        listen 80;
        listen [::]:80;
        server_name ${domain};
        return 301 https://${domain};
}
server {
        listen 80;
        listen [::]:80;
        server_name ${domainAdmin};
        return 301 https://${domainAdmin};
}
server {
        server_name ${domainAdmin};
        listen 443 ssl;
        ssl_certificate /etc/letsencrypt/live/${domainAdmin}/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/${domainAdmin}/privkey.pem;
        location /api {
                proxy_pass http://localhost:${apiPort}/api;
        }
        location /socket.io {
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_set_header X-NginX-Proxy false;
                proxy_pass http://localhost:${apiPort};
                proxy_redirect off;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
        }
        location / {
                proxy_pass http://localhost:${adminUiPort};
        }
}
server {
        server_name ${domain};
        listen 443 ssl;
        listen [::]:443 ssl;
        location /api {
                proxy_pass http://localhost:${apiPort}/api;
        }
        location / {
                proxy_pass http://localhost:${clientUiPort};
        }
        ssl_certificate /etc/letsencrypt/live/${domain}/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/${domain}/privkey.pem;
}
```

Joonis 5. Nginx-konfiguratsiooni mall

Nginx-konfiguratsiooni loomise skripti sisenditeks on kasutajaliidese konteineri port, haldusliidese konteineri port, serverrakenduse konteineri port, kasutajaliidese domeeni nimi ja haldusliidese domeeni nimi.

3.3 Kasutajaliides

Kuna kasutaja peab oma tellimust mugavalt esitama, siis on vajalik ka kasutajaliides. Kasutajaliides peab olema kiirelt ja mugavalt kättesaadav. Rakendust kasutatakse ainult toitlustusasutust külastades, seega ei tohiks olla rakenduse kasutamise nõudeks see, et rakendus installeeritakse kliendi nutiseadmesse. Sellest tulenevalt on rakendus loodud veebilehena. Kasutajaliidese raamistike valik on suur ja eksisteerib mitmeid raamistikke, millel on tugev kasutajas- ja kogukond. Kõigi kasutajaliidese loomise raamistike üldine eesmärk on aidata arendajal luua taaskasutatavaid komponente, lihtsustades ja kiirendades arendusprotsessi, ja kuna erinevate raamistike võrdlusel ei ole ühelgi kindlal raamistikul selget eelist teiste ees, on valitud Stack Overflow küsitluse tulemuste põhjal kõige populaarsem [10]. Haldusliidese autor valis haldusliidese loomiseks samadele põhjustele tugineves samuti Reacti [8]. Valitud front-end raamistikust tuleb pikemalt juttu peatükis 5.6 „React“. Autor otsustas kasutada kasutajaliideses kolmandate osapoolte poolt loodud eeldefineeritud komponente, komponentide raamistikuks oli valitud Material-UI. Material-UIst tuleb lähemalt juttu peatükis 5.8 Material-UI.

Kasutaja peab saama kasutajaliidese kaudu sisestada oma laua koodi, seega domeenile saabudes kuvatakse talle vaadet, kus ta saab koodi sisestada (vt lisa 2). Kui klient on sisestanud lauale asetatud koodi, muutub aktiivseks nupp, millega valideeritakse sisestatud kood (vt lisa 3), ning kasutaja suunatakse asutuse menüülehele (vt lisa 4). Koodi sisestamine pole ainuke võimalus, kuidas jõuda menüüvaatesse, alternatiivina on võimalik kasutada URL-iga suunamist. Õigele URL-ile on võimalik jõuda skaneerides asutuse loodud QR-koodi.

Peale koodi sisestamist või QR-koodi skaneerimist on võimalik kasutajal tellida teenindust kas maksmiseks või muul eesmärgil (vt lisa 6). Samuti saab kasutaja tellimiseks avatud tooteid sorteerida vastavalt kategooriatele (vt lisa 5). Lisatud tooted saab kasutaja tellida lauda (vt lisa 7). Peale teeninduse tellimist või tellimuse esitamist kuvatakse kasutajal sellekohane teade (vt lisa 8). Kui vajutada pakutava toote peale, avaneb toote detailvaade (vt lisa 9).

Kliendipoolset kasutajaliidest on võimalik konfigureerida. Esialgselt on võimalik konfigureerida lehel paiknevate elementide värve. Värvid määrab ära, millist värvi on rakenduse menüü, nupud ja teavitused.

3.4 Andmebaas

Kuna töö maht on suur, rakenduse nõuded võivad valideerimise käigus palju muutuda ja aja jooksul tekkivate andmemahtude hulk on väike, siis on rakenduse andmebaasiks valitud NoSQL, kuna võimaldab andmebaasi kiiremini muutusi sisse viia.

Rakenduse põhitegevused on tellimuste esitamine ja teeninduse kutsumine. Tellimuste endi puhul on tellimuse vastuvõtmisel info olemas ka toitlustusasutuste kassasüsteemis, seega tellimuste info hoidmine ei ole vajalik pikemas perspektiivis, vaid piisab ainult tööpäeva jooksul tehtud tellimuste salvestamisest.

Andmekirjeid tekitatakse iga tellimuse esitamise kohta ning iga teenindaja kutsumise peale, seega andmekirjete vähene hulk võimaldab kasutada ka vähem efektiivsemaid tehnoloogiaid andmete hoiustamisel. Andmebaasis hoitakse järgnevat infot: menüükategooriad, pakutavad tooted, tellimused, teeninduse kutsumine, haldusliidese kasutajad ja kliendiliidese konfigureeritav disain.

Tabelis 1 on välja toodud kasutajaliidese konfigureerimise sisemine struktuur.

Tabel 1. Kasutajaliidese konfiguratsiooni sisemine andmestruktuur

Välja nimi	**Andmetüüp**	**Selgitus**
key	Sõne	Väärtus, mis viitab sellele, mida konfigureeritakse.
value	Sõne	Väärtus, mis on konfigureeritava elemendi väärtus.
_id	Sõne	Unikaalne identifikaator.

Tellimuste ja teeninduse kutsumise andmestruktuuri lahendamiseks on mitmeid variante. Kõige intuitiivsem lahendus oleks lahendada tellimused ja teeninduse kutsumised eraldi dokumentidena. Kuid selline lahendus poleks autori hinnangul rakenduse jõudluse seisukohalt kõige soodsam. Tellimused kui ka teeninduse kutsumised kuvatakse haldusliideses ühes nimekirjas. Kui tellimuse ja teeninduse kutsumine oleksid eraldi, poleks võimalik tõhusalt teostada automaatset lehekülgjaotust, sellepärast on disainitud tellimuse sisemine andmestruktuur nii tellimuse kui ka teeninduse kutsumise puhul ühe dokumendina.

Tabelis 2 on välja toodud tellimuse sisemine struktuur.

Tabel 2. Tellimuse sisemine andmestruktuur

Välja nimi	**Andmetüüp**	**Selgitus**
tablecode	Sõne	Lauakood, kust tellimus tuli.
orderContent	Nimekiri sõnedest	Viide MenuItemAmount dokumendi kirjetele.
isWaiting	Jah/Ei	Kas tellimus on täitmata ja vajab tähelepanu.
createdTime	Kuupäev koos kellaajaga	Kuupäev ja kellaaeg, millal tellimus esitati.
callType	Sõne	Tüüp, näitab kas tegemist oli tellimuse esitamisega, maksmise sooviga või muul eesmärgil teenindusega.
_id	Sõne	Unikaalne identifikaator.

Tellimus on seotud tellitud toodetega, selle jaoks on vahedokument, kus hoitakse infot, kui palju ja milliseid tooteid telliti.

Tabel 3. Tellitud toote koguse sisemine andmestruktuur

Välja nimi	**Andmetüüp**	**Selgitus**
amount	Number	Kogus tellitud tooteid.
menuItemId	MenuItem	Viide MenuItem dokumendi kirjele. Identifikaator.
_id	Sõne	Unikaalne identifikaator.

3.5 Serverrakendus

Serverrakendus oli kirjutatud kasutades Node.js- ja Express-raamistikku. Expressist tuleb lähemalt juttu peatükis 5.5 Express ja Node.js-raamistikust peatükis 5.7 Node.js. Node.js oli valitud, kuna JavaScriptil põhinev tehnoloogia toetab rakenduse sündmuspõhist iseloomu. Populaarne tehnoloogia, mida otseühenduste puhul kasutada, on Socket.io. Socket.io on teek, mis võimaldab reaalajas kahesuunalist ja sündmuspõhist suhtlust brauseri ja serveri vahel [11]. Kui võrrelda JavaScipti teegi ja Java teegi populaarsust, siis Javale sobiva teegi puhul oli töö esitamise hetkel suurusjärgus 4700 GitHubi tähte ja JavaScriptile sobiva teegi puhul suurusjärgus 53000 GitHubi tähte. Suurem populaarsus

julgustab autorit veelgi JavaScriptil põhinevat tehnoloogiat valima. GitHubi tähtede arv võimaldab kasutajatel koodihoidlaid tunnustada ja ka uuendusi oma GitHubi uudistevoogu suunata [12]. Lisaaspekt, mis mõjutas tehnoloogia valikud oli, et autor polnud varem Node.js-rakendusi kirjutanud ja soovis uut tehnoloogiat katsetada.

Autoriseeritud haldusliidese kasutajad loovad serverrakendusega otseühenduse, mis hoiustatakse serverrakenduses mälus. Kui klient esitab tellimuse, tehakse POST-tüüpi päring serverrakendusele, ning sama päringu raames teostatakse mitu tegevust: esmalt salvestatakse tellimus andmebaasi ja seejärel võetakse mälus olev nimekiri otseühendustest ja saadetakse läbi otseühenduse sõnum kõigile ühendunud haldusliidese kasutajatele. Sõnumi sisu on esitatud tellimus, mis kuvatakse koheselt haldusliidese tellimuste nimekirjas. Selline lahendus eemaldab vajaduse kindlaksmääratud sagedusel info uuenduspäringute teostamiseks – info uuendamine teostub siis, kui esineb vastav sündmus.

4 Testimine

Rakenduse valideerimiseks teostatakse kaks tegevust. Kõigepealt testitakse, kas loodud funktsionaalsus töötab, ja seejärel testitakse, kas see rakendus töötab tõrgeteta ka erinevatel töökoormustel.

4.1 Funktsionaalsuse testimine

Kuna rakenduse ärilist loogikat pole palju ja rakenduse funktsionaalsus võib olla muutuv, siis oli otsustatud kasutada tehnoloogiat Selenium. Testid olid loodud läbi Selenium IDE rakenduse. Läbi vastava rakenduse salvestati järgnevad tegevused: avati veebirakenduse leht, sisestati kood, esitati tellimus, kutsuti teenindus maksmiseks, kutsuti teenindus tellimiseks. Selenium IDE võimaldab luua rakenduse funktsionaalsusele automaatseid testi ilma koodi kirjutamata.

4.2 Koormustestimine

Koormustestide eesmärk on rakenduse käitumise kindlakstegemine suure töökoormuse all. Koormustestid annavad teadmise, kas rakendus töötab soovitud töökoormuse all vastavalt eeldustele või mitte. Eelnevalt oli seatud eesmärk, et vähese koormuse all võiksid päringud olema kiiremad kui 0,1 sekundit ja nõue, et vajalikul koormusel peab koormus olema kiirem kui 1 sekund. Koormustestid testivad serverrakenduse päringuid, HTMLi, CSSi ja JavaScripti ressursse koormustestid ei päri. Autori hinnangul on antud veebiressursi koormus ajas konstantne. Koormustestides teeb iga koormustesti voog läbi järgnevad sammud:

1. Valideerib laua info, saades vastuse, kas laud eksisteerib või mitte
2. Küsib asutuse kõik tootekategooriad
3. Küsib kõik asutuse tooted
4. Valib toodete nimekirjast juhuslikult kaks toodet ja esitab tellimuse
5. Kutsub teenindust

Samuti on sammude vahel ooteaeg 100 millisekundit, et imiteerida päris kasutaja käitumist.

4.2.1 Madal koormus

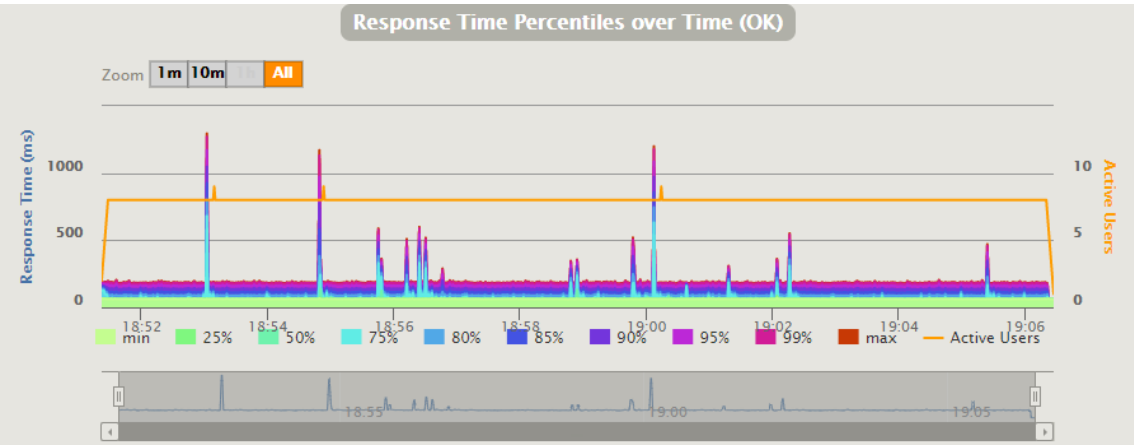
Väikesel koormusel on testitud koormusteste nii, et 15 minuti vältel alustab igas sekundis rakenduse kasutamist üks kasutaja, kes teeb kõik defineeritud kasutusjuhud läbi. Joonisel 6 kujutatud koormustestide väljavõte Gatlingi programmi poolt genereeritud raportist.

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	11250	11250	0	0%	351.562	59	79	113	221	262	1326	107	63
Laua info	2250	2250	0	0%	70.312	179	203	220	260	319	1326	214	64
Tootekategooriad	2250	2250	0	0%	70.312	59	75	88	118	140	390	81	21
Kõik tooted	2250	2250	0	0%	70.312	60	76	88	117	141	511	82	24
Esita tellimus	2250	2250	0	0%	70.312	61	75	90	118	139	380	82	19
Kutsu teenindus	2250	2250	0	0%	70.312	60	72	83	112	130	385	78	18

Joonis 6. Gatlingi raporti väljavõte - koormustestide tulemus madalal koormusel

Kokku tehti 15 minuti jooksul 11250 päringut, lauakoodi valideerimise päringu keskmine kestvus oli 214 millisekundit, mis on rohkem kui autori loodetud kiirus. Teised kasutusvoo poolt tehtud päringud olid keskmiselt alla 100 millisekundi.
Üksikute päringute puhul oli märgata rohkem kui sekundilist vastusaega. Joonisel 7 on kuvatud väljavõte Gatlingi programmi raportist päringute kestvuse kohta.

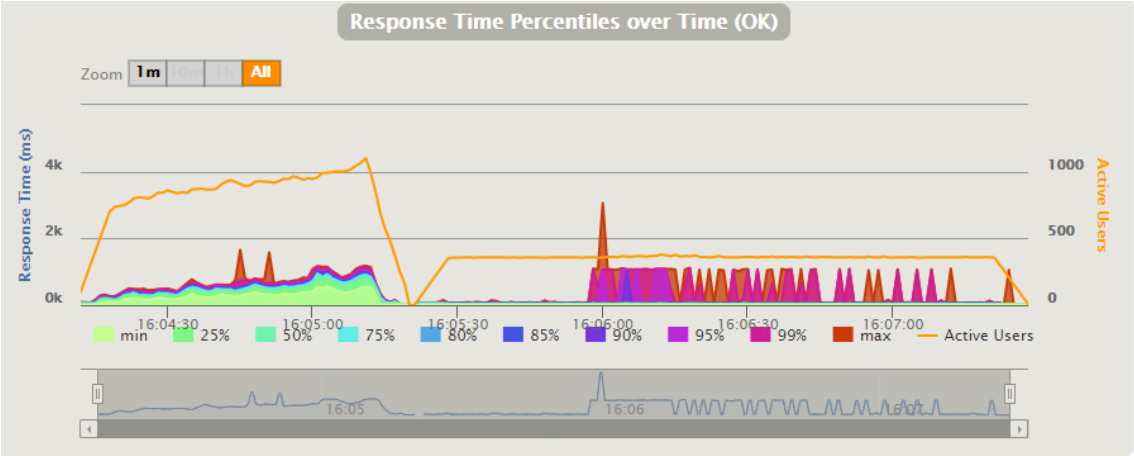


Joonis 7. Gatlingi raporti väljavõte - koormustestide päringute kiirused madalal koormusel

4.2.2 Kõrge koormus

Teiseks testiti rakendust selliselt, et hakkaksid tekkima tõrked rakenduse tavapärases töös. Kuna rakendust testitakse Amazoni pilveserveri vastu, siis simulatsiooni jooksutamise periood ei ole nii pikk, vältimaks võrguliikluse arvet. Simulatsiooni iga

kasutaja läbib eelnevalt defineeritud kasutusvoo. Esimesed 60 sekundit alustas igas sekundis 100 kasutajat kasutusjuhtude täitmist. Seejärel oli 10 sekundit pausi ja algas uuesti koormuse suurendamine lisades igas sekundis 60 kasutajat. Mahus 100 kasutajat sekundis hakkas esinema tõrkeid rakenduse töös ja päringute aeg hakkas pikenema. Lisades mahule juurde 60 kasutajat sekundis olid päringu vastused aeglased, kuid keskmise päringu vastus oli alla 1 sekundi, olles kiirem kui autori seatud eesmärk. Joonisel 8 on kuvatud koormustestide päringute kiirused suurel koormusel.



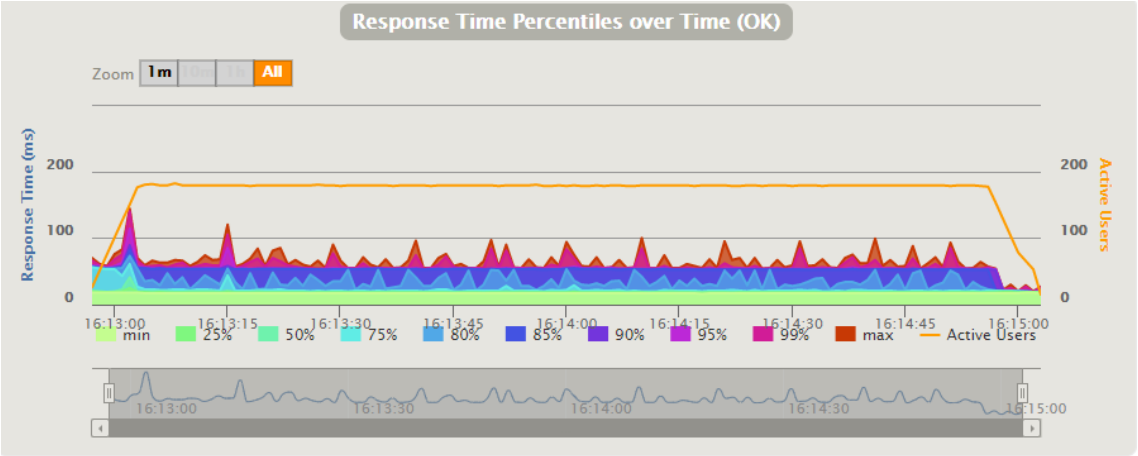
Joonis 8. Gatlingi raporti väljavõte - koormustestide päringute kiirused suurel koormusel

Kogu 200 sekundit kestnud simulatsiooni puhul teostati 68320 päringut. Kasutusvoogusid prooviti teostada 13800 korda, millest ei õnnestunud 1040.

4.2.3 Otseühenduste mõju rakenduse jõudlusele

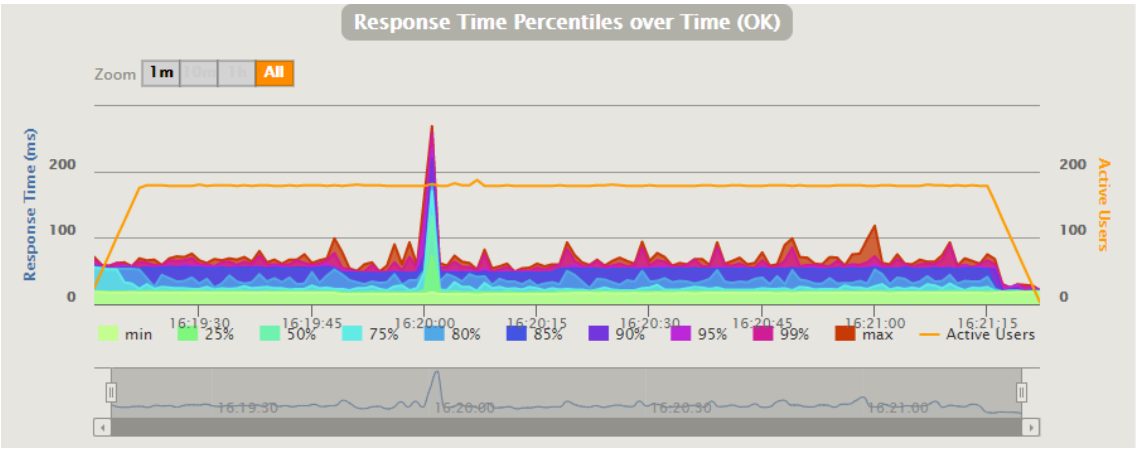
Kuna salvestatud tellimused ja teeninduse kutsumised edastatakse läbi otseühenduse haldusliidesesse, peab testima ka otseühenduse mõju rakenduse koormusele. Selleks teostati võrreldavad koormustestid. Ühel puhul koormustest ilma otseühendusteta ja teisel juhul kaheksa otseühenduse olemasoluga koormustest. Simulatsioon nägi ette, et igas sekundis alustab rakenduse kasutamist 25 kasutajat. Mõlema variandi puhul teostati kokku 15000 päringut. Joonisel 9 on Gatlingi programmi poolt loodud raport, kus on näha

päringu vastuste kiirus simulatsiooni korral, kus puudusid otseühendused.



Joonis 9. Gatlingi raporti väljavõte - päringute kiirus ilma otseühendusteta

Joonisel 10 on Gatlingi programmi poolt loodud raport, kus on näha päringu kiirused, kuid selle simulatsiooni puhul oli loodud viie erineva seadmega kaheksa otseühendust. Kõigist seadmetest oli reaalajas näha uusi lisandunud tellimusi ning teeninduse kutsumisi.



Joonis 10. Gatlingi raporti väljavõte - päringute kiirus otseühendustega

4.3 Testimise tulemus

Koormustestide tulemus on rahuldav, koormustestide ajal oli märgata EC2-serveri protsessori krediidi kasutamise suurenemist üle ettenähtud limiidi. Rakenduse testimisel madalal koormusel protsessori krediidi suurenemist märgata ei olnud. Rakenduse tegelik koormus hakkab eeldatavasti sarnanema madalal koormusel töötamise simulatsioonile, mitte kõrgel koormusel teostatud simulatsioonile. Lisaks otseühenduste mõju

hindamisele ei tuvastanud autor suurt mõju rakenduse tööle, seega võib autori hinnangul tehnoloogia valiku ja lahenduse tulemusega igati rahule jääda.

5 Tehnoloogiad

Järgnevas peatükis antakse ülevaade kasutatud tehnoloogiatest.

5.1 Amazon EC2

Amazon EC2 on pilveteenus, mis pakub pilves turvalist ja muudetava võimsusega serverit [13].

5.2 Nginx

Nginx on vabavaraline tarkvara, mida kasutatakse peamiselt veebiserverina [14]. Kuna kõigi rakenduste jaoks kasutatakse sama serverit, peab lisama rakenduste kasutamise jaoks konfigureerima pöördproksi.

5.3 MongoDB

Andmebaasiks on valitud MongoDB. MongoDB ei vaja andmete hoiustamiseks eeldefineeritud struktuuri, seega muudatuste sisseviimine on väga lihtne. Andmeid hoitakse JSONi-laadsetes dokumentides [15].

5.4 Mongoose

Kuna andmebaasiks on valitud MongoDB, on Node.js rakendusele lisatud Mongoose. Mongoose on ODM-raamistik, mis pakub skeemipõhist lahendust andmete modelleerimiseks, päringute koostamiseks, andmete salvestamiseks ja palju muud. See võimaldab andmebaasiga suhtlust valideerida ja suhelda MongoDB-andmebaasiga ja kiirendab arendusprotsessi [16].

5.5 Express

Express on Node.js-raamistik, mis võimaldab struktureerida veebirakendust erinevate HTTP-päringute teenindamiseks. Express on minimalistlik ja paindlik veebirakenduste raamistik, mis on loodud API-de arendamise hõlpsamaks muutmiseks [17].

5.6 React

React on JavaScripti teek taaskasutatavate kasutajaliidese komponentide loomiseks. Seda haldab Facebook ning üksikute arendajate ja ettevõtete kogukond [18]. Reactil on väga tugev kogukond ja sellest tulenevalt on antud teegi kohta saadaval väga palju infot. Reacti virtuaalne DOM (VDOM) on programmeerimiskontseptsioon, kus kasutajaliidese virtuaalne esitus on mälus ja sünkroonitakse reaalse DOM-iga sellise teegi kaudu nagu ReactDOM [19].

5.7 Nodejs

Node.js on käivituskeskkond JavaScripti programmi jooksutamiseks, mis jookseb JavaScripti V8-käivitusmootoril [20]. JavaScripti V8-mootor muudab JavaScripti koodi masinkoodiks, mida arvuti saab käivitada [21].

5.8 Material-UI

Material-UI on taaskasutatud komponentide raamistik, kus loodud komponendid on loodud kasutades Material Design disainisüsteemi. Material Design on Google'i loodud disainisüsteem, mis aitab meeskondadel luua veebis kvaliteetseid digitaalseid kogemusi [22].

6 Edasised arendused

Kuna valminud tarkvara pole lõplik, vaid täieneb jooksvalt, on järgnevas peatükis välja toodud nimekiri töödest, mille teostamist autor tulevikus kaalub.

6.1 Monitooringu ja seirelahenduse lisamine

Hetkel puudub ülevaade, kui mõnes rakenduses tekivad vead. Kuna konteinereid on palju, ja seeläbi logide regulaarselt manuaalne ülevaatamine ajakulukas, on autori hinnangul tähtis lisada keskne logimine ja seire. See loob võimaluse reageerida tekkinud vigadele kiiresti. Ilma keskse logimise ja seireta võib vigane rakendus tekitada probleeme pikka aega, enne kui probleemi olemasolu tuvastatakse.

6.2 Konteinerite orkestreerimine

Hetkel töötab lahendus ühe serveri peal. Kui kliente lisandub liiga palju ja on vaja rohkem konteinerdatud rakendusi käivitada kui üks server lubab, peab alustama serveri seadistamist algseadistatud serverist. Selle vältimiseks oleks võimalik kasutusele võtta konteineri orkestreerimistehnoloogia, mis lubaks servereid kerge vaevaga seadistatud klastrisse lisada.

6.3 REST API dokumentatsiooni loomine

Kuna võib esineda tehniliselt nõudlikemaid kliente, kes soovivad oma menüüd lisada mitte läbi graafilise liidese, vaid läbi REST-teenuste, võiksid olla kõik loodud teenused dokumenteeritud. Dokumentatsiooni lisamiseks on olemas erinevad lahendused, mis muudavad selle loomise väga lihtsaks. Populaarne dokumentatsiooni loomise tööriist on Swagger.

Swagger on OpenAPI-spetsifikatsiooni ümber ehitatud avatud lähtekoodiga tööriistade komplekt, mis aitab REST API-sid kujundada, ehitada, dokumenteerida ja tarbida. OpenAPI-spetsifikatsioon on REST API-de API-vormingus kirjelduse vorming [23].

6.4 TypeScript toe lisamine

TypeScript on avatud lähtekoodiga keel, mis põhineb JavaScriptil, mis on üks maailma enimkasutatud tööriistu, lisades staatilise tüübi määratlusi. Tüübid võimaldavad kirjeldada objekti kuju, pakkudes paremat dokumentatsiooni ja võimaldades TypeScriptil kontrollida, kas kood töötab õigesti [24].

Kuna parem dokumentatsioon ja veatuvastus parandaks koodikvaliteeti, võiks tulevikus kaaluda TypeScipti kasutuselevõttu.

6.5 Alamdomeeni registreerimise automatiseerimine

Hetkel toimub uue alamdomeeni registreerimine käsitsi. Kui see protsess ära automatiseerida, oleks võimalik kliendil endal registreerida vajalik alamdomeen ilma autori sekkumiseta. Kui alamdomeeni registreerimine oleks automatiseeritud, oleks võimalik seejärel kutsuda välja ka autori loodud skriptid Nginx-pöördproksi uuendamiseks. Sellisel juhul saaks toote kasutaja ise alamdomeeni tekitada ja sellele vastava juurutamise, sisestades vaid alamdomeeni nime ja tulevase administraatori e-posti aadressi.

6.6 Maksmise võimaluse lisamine

Kuigi töö esitamise hetkel oli valitud kasutusvoog ilma maksmiseta, ei tähenda, et tulevikus ei pruugi tekkida vajadust maksmisvõimalust lisada. Hetkel ei ole antud arendus prioriteediks.

6.7 Mitmekeelsuse toe lisamine

Hetkel toetab rakendus ainult ühte keelt. Kuna paljud asutused pakuvad menüüd ka välisturistidele, võiks rakenduses olla ka tugi muudele keeltele. Teised keeled võiksid samuti olla rakenduse konfiguratsiooni alt konfigureeritavad. Antud arendus oleks suure mahuga, kuna ei hõlmaks endas vaid andmebaasi ja serverrakenduse muudatusi, vaid ka haldusliidese disaini ja funktsionaalsust.

6.8 Dockeri tõmmiste ehitamise automatiseerimine

Hetkel ehitatakse ja laaditakse tõmmised Docker Registrysse samast keskkonnast, kus toimub arendus. Tulevikus võiks tõmmiste ehitamine olla lahendatud läbi GitHubi toimingu. GitHubi toimingud aitavad tarkvaraarenduse olelusringi jooksul ülesandeid automatiseerida. GitHubi toimingud on sündmuspõhised, mis tähendab, et saate käske käivitada pärast määratud sündmuse toimumist. Näiteks on võimalik igal korral, kui keegi loob hoidla tõmbenõude, käivitada automaatselt käsu, mis omakorda käivitab tarkvara testimise skripti [12]. Autori näitel oleks võimalik seda toimingut kasutada Dockeri tõmmiste ehitamiseks ja isegi rakenduse automaatseks juurutamiseks.

7 Kokkuvõte

Käesoleva bakalaureuse lõputöö eesmärgiks oli luua osa kesksest süsteemist, mis võimaldab toitlustusasutuste külalistel lihtsamalt ja mugavamalt oma tellimusi esitada. Rakenduse kliendipoolse konfigureeritava disaini ja juurutamise eesmärk on luua lahendus, läbi mille oleks võimalik rakenduse disaini ühtlustada asutuse üldise disainiga, jättes kasutajale mulje, nagu oleks toode arendatud spetsiaalselt seda kasutavale ettevõttele. Mõlemad eesmärgid suudeti autori hinnangul täita.

Töö jaoks vajalik tehnoloogia oli suurel määral autori jaoks uus ja sellest tulenevalt võttis koodiosa palju aega ja vajas pidevalt koodikvaliteedi parandamist. Autor kasutas esmakordselt järgnevaid tehnoloogiaid: Express, Mongoose, React, Node.js, Socket.io, Gatling, Selenium.

Rakenduse demokeskkonna ligipääs on saadetud erinevatele asutustele proovimiseks ja autor näeb lahenduses lisaväärtust seda kasutusele võtvatele asutustele. Valminud rakendus pole lõplik versioon ja täieneb jooksvalt. Rakenduse kõikide osade kood on esitamise hetkel nähtav GitHubi Git-koodihoidlates:

- Serverrakendus - https://github.com/nils-emil/resto-order-api
- Haldusliides - https://github.com/nils-emil/resto-order-admin-ui
- Kliendi kasutajaliides - https://github.com/nils-emil/resto-order-client-ui
- Konfiguratsioon - https://github.com/nils-emil/resto-order-config

Kasutatud kirjandus

[1] „Ordly,“ [Võrgumaterjal]. Saadaval: https://www.ordly.io [Kasutatud 20. veebruar 2020].

[2] „Babash,“ [Võrgumaterjal]. Saadaval: https://babash.eu/home [Kasutatud 12. jaanuar 2021].

[3] „The WebSocket API,“ [Võrgumaterjal]. Saadaval: https://www.w3.org/TR/websockets/ [Kasutatud 15. märts 2020].

[4] „The WebSocket API (WebSockets),“ [Võrgumaterjal]. Saadaval: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API [Kasutatud 15. märts 2020].

[5] „Identifiable individuals and identifiers,“ [Võrgumaterjal]. Saadaval: https://gdpr.eu/eu-gdpr-personal-data/ [Kasutatud 30. märts 2020].

[6] „A study on tolerable waiting time: how long are Web users willing to wait?,“ [Võrgumaterjal]. Saadaval: https://www.tandfonline.com/doi/pdf/10.1080/01449290410001669914 [Kasutatud 3. aprill 2021].

[7] „Kuidas hoiduda kaardimakse pettustest?,“ [Võrgumaterjal]. Saadaval: https://www.kaubandus.ee/sisuturundus/2017/04/13/kuidas-hoiduda-kaardimakse-pettustest. [Kasutatud 28. veebruar 2020].

[8] „Toidukohtade tellimuste esitamise keskkonna haldusliidese veebilahenduse arendus,“ [Võrgumaterjal]. Saadaval: https://digikogu.taltech.ee/et/Item/8423b72f-629f-4b07-935a-40ec8573904f [Kasutatud 9. mai 2021].

[9] „sed,“ [Võrgumaterjal]. Saadaval: https://linux.die.net/man/1/sed [Kasutatud 23. aprill 2020].

[10] „Developer Survey Results 2019,“ [Võrgumaterjal]. Saadaval: https://insights.stackoverflow.com/survey/2019 [Kasutatud 20. aprill 2020].

[11] „Socket.io,“ [Võrgumaterjal]. Saadaval: https://socket.io/docs/v4 [Kasutatud 1. mai 2021] .

[12] „Learn GitHub Actions,“ [Võrgumaterjal]. Saadaval: https://docs.github.com/en/actions/learn-github-actions [Kasutatud 30. märts 2020].

[13] „Amazon EC2,“ [Võrgumaterjal]. Saadaval: https://aws.amazon.com/ec2/ [Kasutatud 9. mai 2021].

[14] „What is NGINX?,“ [Võrgumaterjal]. Saadaval: https://www.nginx.com/resources/glossary/nginx/ [Kasutatud 4. aprill 2020].

[15] „MongoDB vs MySQL Differences,“ [Võrgumaterjal]. Saadaval: https://www.mongodb.com/compare/mongodb-mysql [Kasutatud 1. märts 2020].

[16] „What Is MongoDB?,“ [Võrgumaterjal]. Saadaval: https://www.mongodb.com/what-is-mongodb [Kasutatud 20. aprill 2020].

[17] „Expressjs,“ [Võrgumaterjal]. Saadaval: Available: https://expressjs.com/ [Kasutatud 1. aprill 2020].

[18] „React,“ [Võrgumaterjal]. Saadaval: https://reactjs.org/docs/getting-started.html [Kasutatud 13. aprill 2020].

[19] „What is the Virtual DOM?,“ [Võrgumaterjal]. Saadaval: https://reactjs.org/docs/faq-internals.html [Kasutatud 8. mai 2021].

[20] „Node.js,“ [Võrgumaterjal]. Saadaval: https://nodejs.org/en/ [Kasutatud 13. aprill 2020].

[21] „What is V8?,“ [Võrgumaterjal]. Saadaval: https://v8.dev/ [Kasutatud 15. aprill 2020] .

[22] „Material Design,“ Saadaval: https://material.io/design/introduction [Kasutatud 9. mai 2021].

[23] „What Is Swagger?,“ [Võrgumaterjal]. Saadaval: https://swagger.io/docs/specification/about/ [Kasutatud 8. mai 2021].

[24] „What is TypeScript,“ [Võrgumaterjal]. Saadaval: https://www.typescriptlang.org/ [Kasutatud 9. mai 2021].

[25] „Saving repositories with stars,“ [Võrgumaterjal]. Saadaval: https://docs.github.com/en/github/getting-started-with-github/saving-repositories-with-stars/ [Kasutatud 12. mai 2021].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

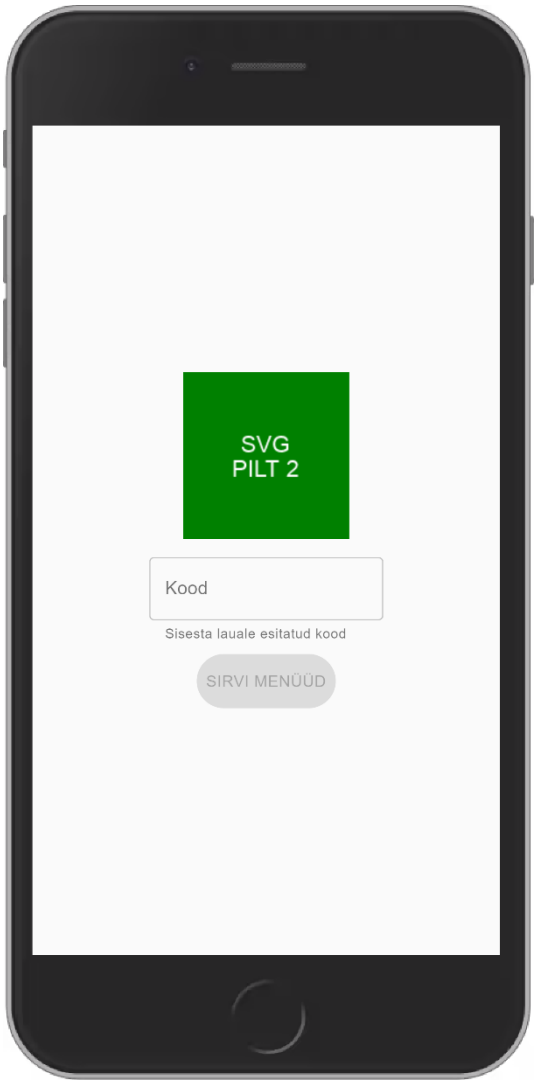
Mina, Nils-Emil Lille

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kliendipoolse tellimuste esitamise veebilahenduse loomine“, mille juhendaja on Inna Švartsman
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2021

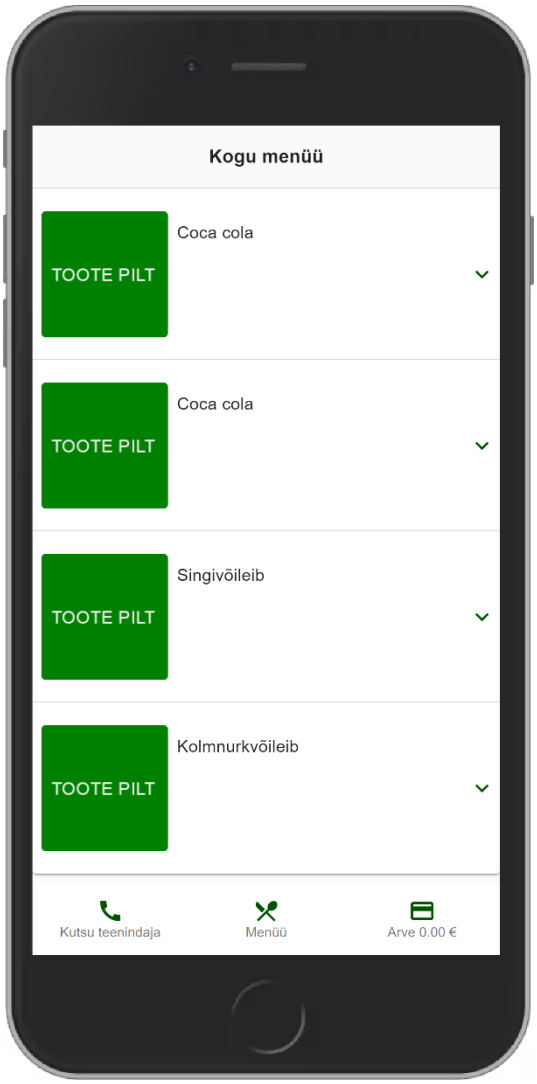
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Lauakoodi sisestamise vaade



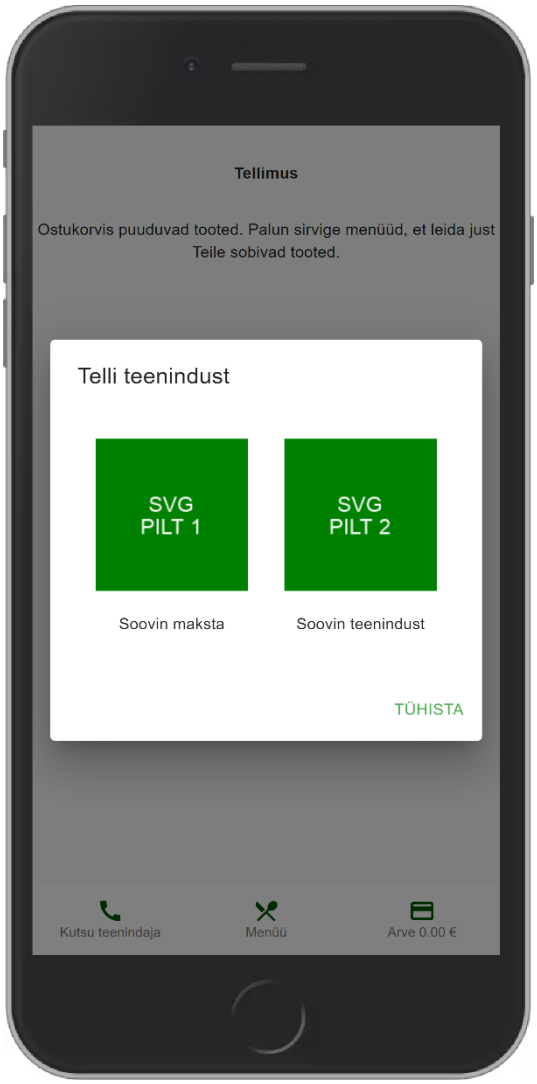
Lisa 3 – Lauakoodi sisestamise vaade peale koodi sisestamist

Lisa 4 – Toodete nimekirja vaade

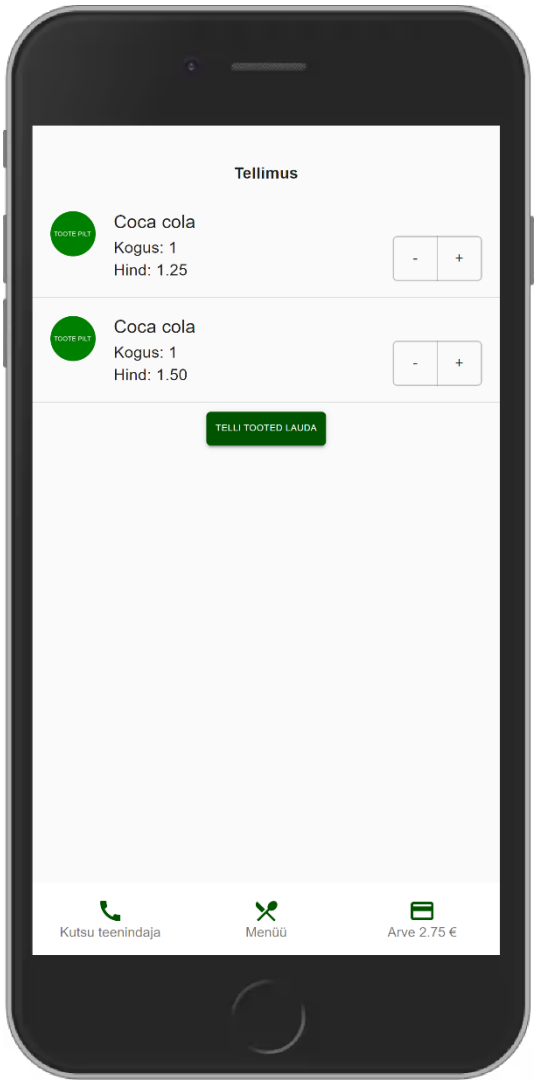


Lisa 5 – Kategooriate nimekirja vaade

Lisa 6 – Teeninduse valiku vaade



Lisa 7 – Ostukorvi nimekirja vaade



Lisa 8 – Kasutaja teavituse kuvamine

Lisa 9 – Toote detailvaade