TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikita Popov 179351IAIB

TELLIMUSTE PLATVORMI SERVERI OSA LOOMINE TOITLUSTUSKOHTADELE

Bakalaureusetöö

Juhendaja: Aleksei Talisainen
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: Nikita Popov

18.05.2021

Annotatsioon

Selle töö käigus oli arendatud toitlustuskohtade tellimuste platvormi serveriosa. Platvorm võimaldab restoranidel tellimusi vastu võtta ja neid töödelda. Skaneerides nutiseadmega laual asuvat QR-koodi, läheb klient oma seadmes restorani veebilehele, kus tal on võimalus tellida toitu vaid mõne klikiga. Iga QR-kood on ainulaadne ja on seotud kindla lauaga, tänu sellele teavad teenindajad kuhu tellimust viia.

Platvormi serveri osa arendamiseks kasutati programmeerimiskeelt Go ja PostgreSQL andmebaasi haldussüsteemi. Töötav rakendus on leitav aadressil https://app.snakk.ee

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 6 peatükki, 19 joonist.

Abstract

Creating of Server Side of Ordering Platform for Catering Places

In this graduation thesis a server side of ordering platform for catering places was created. The platform allows restaurants to receive and process orders. By scanning the QR code on the table, the customer goes to the restaurant page, where he can order food with just a few clicks. Each QR code is unique and is associated with a specific table, it helps waiters know where to bring the order.

The programming language Go and the PostgreSQL database management system weer used to create the server side of the platform. The running application is located at: https://app.snakk.ee

The thesis is in Estonian and contains 30 pages of text, 6 chapters, 19 figures.

Lühendite ja mõistete sõnastik

DPI	*Dots per inch*, punkti tolli kohta
IA	Arvutisüsteemide instituut
XML	*Extensible Markup Language*, laiendatav märgistuskeel
JWT	*JSON Web Token*, JSON kujuga veeb token
OS	*Operating System, o*peratsiooni süsteem
JSON	*JavaScript Object Notation*, teksti andmevahetuse formaat
API	*Application Programming Interface*, rakendusliides
BackOffice	Restorani juhtimispaneel
HTTPS	*HyperText Transfer Protocol Secure,* turvaline hüperteksti edastusprotokoll
HTTP	*HyperText Transfer Protocol,* hüperteksti edastusprotokoll
Front-end	Kasutajaliides
Back-end	Tagarakendus ehk serveripoolne rakendus
PID	*Process IDentifier,* operatsioonisüsteemi protsessi identifikaator
UUID	*A universally unique identifier,* unikaalse identifikaatori formaat

Sisukord

Jooniste loetelu

1 Sissejuhatus

1.1 Taust ja probleem

Kui söögikoht on rahvast täis võib tekkida probleem, kus klient võib kaua oodata kuni klienditeenindaja tuleb tema juurde tellimust vastu võtma. Samuti läheb tihti kaua aega, et teenindaja tooks lauda lõpparve. Paljudel söögikohtadel ei ole võimalust endale oma mobiilrakendust omada ja see pole alati ka mõttekas, sest kliendi jaoks on tellimuse esitamiseks ebamugav iga konkreetse restorani rakendust installida.

1.2 Eesmärk ja oodatud tulemus

Eesmärk on pakkuda söögikohtadele võimalust tellimusi vastu võtma kohapeal veebirakenduse kaudu. Idee on teha platvorm, mis seisneb kahest osast:

- Veebirakendus söögikohtade klienditele - klient skaneerib laua peal asuva QR koodi, mis suunab teda restorani veebileheküljele, kus tema saab näha menüü, valida endale söök ja maksta selle eest kohapeal kasutades ainult tema nutiseadet.

- BackOffice söögikohtade omanikele ehk meie teenuse klientidele, kus nemad saavad lisada ja eemaldada toidu kategooriat, lisada ja eemaldada toitu, muuta hinda, vaadata tellimuste ajalugu.

Söögikohad saavad mugavalt jälgida tellimusi ja pakkuda mugavat tellimuste keskkonda.

1.3 Ülesehitus

Käesoleva töö teises peatükis on väljatoodud rakenduse valitud tehnoloogiate analüüs. Kolmandas peatükis on väljatoodud rakenduse arendamise protsess: rakenduse nõuded, rakenduse arhitektuur, makse värava teenusepakkujatega liidestamine ja rakenduse majutamine. Neljandas peatükis on väljatoodud rakenduse testimise viisid. Viiendas peatükis on esitatud rakenduse turvalisuse analüüs. Kuuendas peatükis on kirjeldatud valmis rakendus.

2 Valitud tehnoloogiate analüüs

2.1 Serveri tehnoloogiate valik

Võrdluseks oli valitud järgmised serveri tehnoloogiad:

- NodeJS
- PHP
- Go

2.1.1 Nõuded

Tehnoloogiaid võrreldi järgmiste nõuete pealt:

1. Kiirus.
2. Koodi kirjutamise lihtsus.
3. Võimalus serveri ressursse tõhusalt kasutada.
4. Lihtne hooldada koodi.

2.1.2 NodeJS

Asünkroonse sündmustepõhise JavaScripti keskkonnana on Node.js loodud skaleeritavate võrgurakenduste loomiseks. [1]

Tehnoloogial on järgmised eelised:

- Mitteblokeeriv I/O ja asünkroonne päringute töötlemine võimaldavad NodeJS-il taotlusi viivitusteta töödelda. [2]
- NodeJS-is kasutatud V8-mootor töötati algselt välja Chrome'i brauseri jaoks ja seda kasutatakse JavaScripti funktsioonide masinakoodiks kompileerimiseks ning see toimib suurel kiirusel. [2]

- Npm-i registris on saadaval üle miljoni koodi pakette, mis teeb sellest kõigi aegade suurima tarkvararegistri. [3]

Tal on ka oma puudused:

- Asünkroonse olemuse tõttu tugineb NodeJS suuresti *callback'ile* ehk funktsioonidele, mida käivitatakse peale iga järjekorras oleva ülesande täitmist. Mitme ülesande salvestamine taustal järjekorda, millest igaühel on oma *callback* võib põhjustada niinimetatud *„callback hell",* mis mõjutab otseselt koodi kvaliteeti. [2]

- Madala kvaliteediga npm paketid. Kui registris on saadaval palju mooduleid ja pakette, mille vahel on kindlasti palju valida, võivad mõned paketid olla kas ebakvaliteetsed või halvasti dokumenteeritud.

2.1.3 PHP

PHP - populaarne üldotstarbeline skriptikeel, mis sobib veebiarenduseks. [4]

Eelised:

- PHP - üks kiiremaid skriptikeeli, mis võimaldab arendajatel muudatuste või nende tulemusi koheselt näha. [5]

- Võimaldab andmete tõhusat salvestamist ja otsimist tänu oma adaptiivsele olemusele ja mitme andmebaasi toele. [5]

Puudused:

- PHP ei ole mõeldud töötlemisel tekkivate vigade käsitlemiseks ja on ebastabiilne. [5]

- PHP-programmeerimist on suurte rakenduste jaoks keeruline hooldada ja kasutada. See on raskemate rakenduste toetamiseks ebaefektiivne. [5]

2.1.4 Go

Go on väljendusrikas, kokkuvõtlik, puhas ja tõhus. Selle samaaegsusmehhanismide abil on lihtne kirjutada programme, mis kasutavad maksimaalselt ära mitme tuumaga ja võrku

ühendatud masinate võimsust, ning uue tüüpi süsteem võimaldab programme luua paindlikult ja modulaarselt. Go kompileerib kiiresti masinakoodi, kuid sellel on ka mälukoristus ja käitamisaja kajastamise mugavus. See on kiire, staatiliselt sisestatud, kompileeritud keel, mis tundub dünaamiliselt tüüpiseerimisena ja interpriteerimis keelena. [6]

Eelised:

- Sellel on kompilaator ja puudub interpritaator, mis võimaldab Go'l pakkuda suurt jõudlust. [7]
- Eraldatud tõhus mälu haldamine muudab rakenduste kirjutamise lihtsaks. [7]
- Keelel on puhas süntaks, nii et arendajatel on seda alati lihtne kasutada. [7]
- Go abil loodud rakendused on väga skaleeritavad. Arendajad saavad kiiresti tuvastada kitsaskohad, mis võivad mõjutada rakenduse mastaapsust, andes neile piisavalt aega rakenduse optimeerimiseks. [7]
- Selle asemel, et siduda iga täitmise lõim OS-i tasandil ühe lõimega, kasutab Go gorutiinide mõistet. Sõltuvalt gorutiini sooritatud ülesandest võib keele käitamine määrata gorutini operatsioonisüsteemi lõimele ja sundida seda täitma - või panna selle ooterežiimi ja mitte seostada operatsioonisüsteemi lõimiga. Iga Go HTTP-serveri päringut töödeldakse eraldi gorutiinis. Go levitab gorutiinid dispetšeri loogika järgi automaatselt nii paljudele OS-i lõimudele, kui ta seda õigeks peab. [8]

Puudused:

- Võrreldes teiste programmeerimiskeeltega puudub Go-l keerukas abstraktsioon, mis nõuab keerukate funktsioonide rakendamiseks rohkem koodi. [7]
- Võrreldes teiste juhtivate programmeerimiskeeledega on Go-l vähem arenenud arendajate kogukond. [7]

2.1.5 Serveripoolne tehnoloogiate toimivuse võrdlus.

Iga tehnoloogia jaoks on kirjutatud kood, see loeb 64-kilobaidist faili, mis on täidetud juhuslike baitidega, seejärel rakendab N korda sellele SHA-256 räsi ja kuvab saadud räsi

kuueteistkümnendsüsteemis. See on väga lihtne viis sama jõudluse võrdlusaluse käitamiseks koos järjepideva sisend- ja väljundühenduse ning kontrollitud viisiga protsessori kasutamise suurendamiseks. [8]

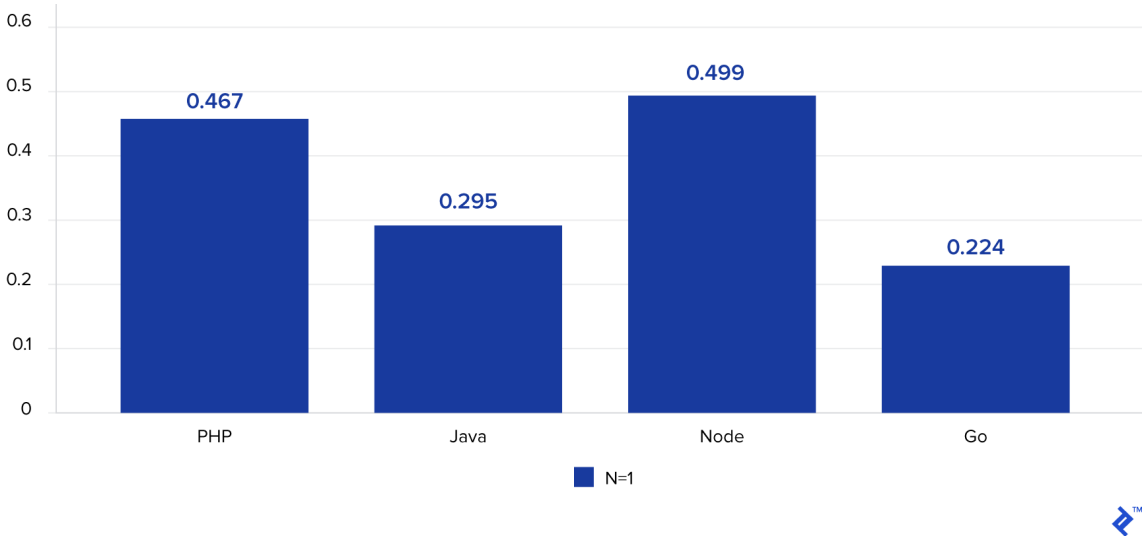
2.1.5.1 Katsekeskkond

Võrdlusuuringud käivitati CentOS 7.3 (Linux 3.10.0-514.el7.x86_64) virtuaalses masinas (Intel Core i7, 4 core @ 2,2 GHz, 1 GB RAM), mis töötab Maci sülearvutis VirtualBoxis. Ajastused võeti ApacheBenchiga (v.2.4.6). [8]

- PHP v5.4.16; Apache v2.4.6
- Java (OpenJDK) 1.8.0_131-b11; Tomcat v7.0.69 (without APR/native)
- Node.js v6.10.3
- Go v1.8.1

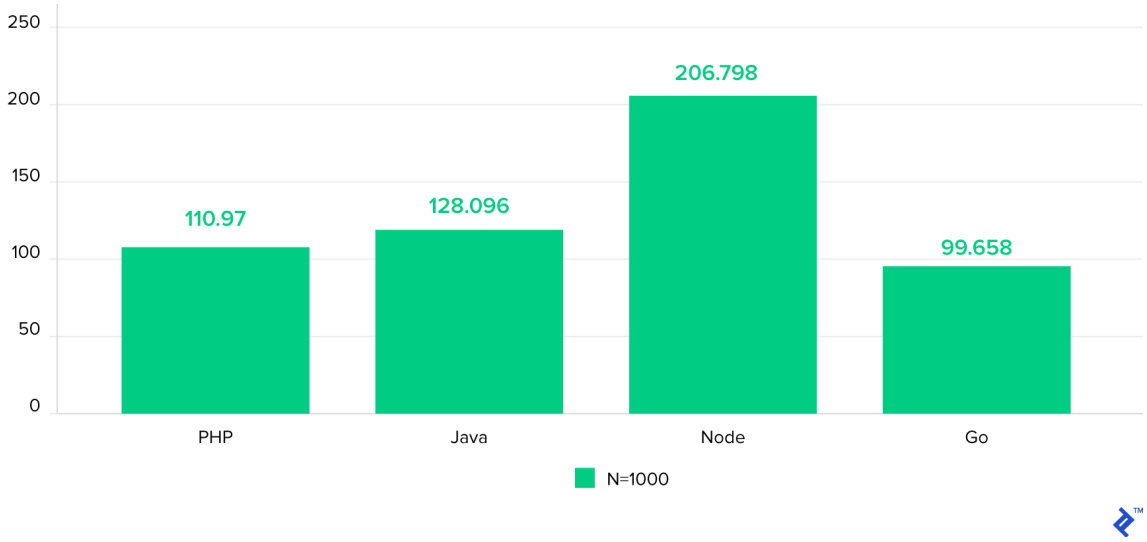
2.1.5.2 Võrdlus

2000 iteratsiooni käivitamine 300 samaaegse taotlusega ja ainult üks räsi ühe päringu kohta (N = 1) annab meile tulemusi, mis on välja toodud joonisel 1.



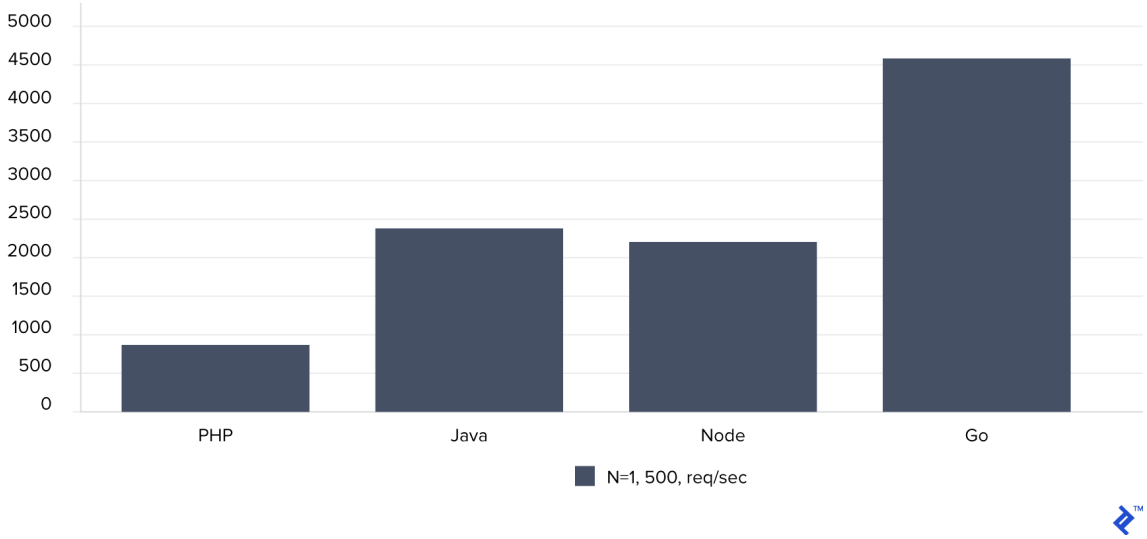
Joonis 1 Ajad on keskmine millisekundite arv taotluse täitmiseks kõigi samaaegsete taotluste korral. Madalam on parem.

Suurendame N väärtuseks 1000, jättes 300 samaaegset päringut - koormus on sama, kuid vaja sooritada sada korda rohkem räsimistoiminguid, tulemused on välja toodud joonisel 2.



Joonis 2 Ajad on keskmine millisekundite arv taotluse täitmiseks kõigi samaaegsete taotluste korral. Madalam on parem.

5000 samaaegse päringu korral oli rikke protsent enamikes keskkondades märkimisväärne. Joonis 3 näitab, et Go sai ülesandega kõige paremini hakkama, järgnesid Java, seejärel Node ja viimane – PHP. [8]



Joonis 3 Päringute koguarv sekundis. Kõrgem on parem.

2.1.6 Back-end tehnoloogia lõplik valik

Eeltoodud eelistest ja puudustest lähtudes valiti serveritehnoloogiaks Go programmeerimiskeel, kuna sellel on suurem kiirus ja lihtne edasine rakenduse mastaapsus.

2.2 Andmebaasi haldussüsteemi valik

Andmebaaside haldussüsteemi valik tulenes järgmistest tehnoloogiates:

- MySQL
- PostgreSQL

2.2.1 MySQL võrdlus PostgreSQLiga

MySQL - maailma populaarseim avatud lähtekoodiga andmebaaside haldussüsteem. [9]

Ajalooliselt on MySQLil olnud maine, et see on lugemismahukate töökoormuste jaoks ülikiire andmebaasi haldussüsteem, mõnikord kirjutamisega segatuna paralleelsuse hinnaga. [10]

PostgreSQL - on võimas avatud lähtekoodiga objektide-relatsioonide andmebaasi haldussüsteem, mida on aktiivselt arendatud üle 30 aasta ja mis on teeninud usaldusväärse, funktsionaalse ja töökindla maine. [11]

MySQL-i ja PostgreSQL-i vahel otsustamisel ei tohiks jõudlus olla enamiku levinud rakenduste tegur - see on niikuinii piisavalt hea, isegi kui arvestada tulevase prognoositava kasvuga. [10]

PostgreSQL on objektidega seotud andmebaaside haldussüsteem, MySQL aga puhtalt relatsiooniline andmebaaside haldussüsteem. See tähendab, et PostgreSQL sisaldab selliseid funktsioone nagu tabelite pärimine ja funktsioonide ülekoormamine, mis võivad teatud rakenduste jaoks olulised olla. PostgreSQL järgib ka rangemalt SQL standardid. [10]

PostgreSQL-il on palju parem dokumentatsioon. MySQL-is võib üksikute valikute mõte olla raskesti mõistetav. Üksikasjalikult on kirjutatud mida nad teevad, kuid selleks, et mõista, kuidas neid õigesti konfigureerida on vaja kasutada mitteametliku dokumentatsiooni ja otsida selle teema artikleid. Sageli on vaja mõista MySQL-i arhitektuuri, kuna ilma mõistmiseta näevad seaded arusaamatud välja. [12]

PostgreSQL-is on laiem andmetüüpide valik nagu UUID, XML, JSON, BOOLEAN ja nii edasi. [13]

PostgreSQL pakub sündmuste *triggereid*, mis võivad kutsuda erinevaid funktsioone. Kui on vajadus PostgreSQL-is tegutseda konkreetsete andmebaasisündmustega nagu INSERT, UPDATE, DELETE või TRUNCATE, võib *trigger* olla kasulik, kuna see kutsub teatud sündmuste jaoks vajaliku funktsiooni. [14]

Trigger seostatakse määratud tabeli, vaade või võõra tabeliga ja ta täidab selles tabelis teatud toimingute tegemisel määratud funktsiooni. Sõltuvalt nõuetest on võimalik luua *trigger* BEFORE, AFTER või INSTEAD. [14]

2.2.2 Andmebaasi haldussüsteemi lõplik valik.

Selles lõputöös valiti ülaltoodud võrdluste põhjal PostgreSQL andmebaasi haldussüsteemi, kuna sellel on MySQLi ees järgmised olulised eelised:

- Sündmuse käivitajad, mis jälgivad muudatusi andmebaasi ridades.
- Dokumentatsioon on kirjutatud lihtsamas keeles.
- Laiem andmetüüpide valik.
- Perspektiivis kasvab andmemaht väga kiiresti, andmebaas peaks suutma keerukaid päringuid kiiresti töödelda.

3 Rakenduste arendus

3.1 Rakenduse nõuded

Antud peatükis on väljatoodud serverirakenduse nõuded.

Serverirakendus põhineb järgmistel nõuetel:

1. BackOffice

    1.1. Restorani omanik saab lisada oma restorani.

    1.2. Restorani omanik saab hallata oma restorani andmeid.

1.3. Restorani omanik saab lisada toidu kategooriat.

1.4. Restorani omanik saab hallata toidu kategooriat.

1.5. Restorani omanik saab lisada tooteid.

1.6. Igal tootel saab olema oma kategooria.

1.7. Restorani omanik saab hallata toote andmed.

1.8. Restorani omanik saab tellimuste ajalugu jälgida.

1.9. Restorani omanik näeb tellimuse staatuse reaalajas.

1.10. Restorani omanik saab sisse logida kasutades oma Google kontot.

1.11. Edukate tellimuste vahendid lisatakse restorani sisekontole.

1.12. Klient saab taotleda makset sisekontolt isiklikule pangakontole.

1.13. Administraator näeb kõigi restoranide loendit.

1.14. Administraator saab muuta iga restorani andmed.

1.15. Administraator saab muuta mis tahes kategooria andmed.

1.16. Administraator saab muuta mis tahes toote andmed.

1.17. Administraator näeb kõigi restoranide tellimuste ajalugu.

1.18. Administraator saab restorani sisekontolt maksetaotluse heaks kiita või tagasi lükata.

2. Restorani klientide leht

2.1. Kasutaja näeb toodete ja kategooriate loendit restorani lehel.

2.2. Kasutaja saab restorani lehel tellimuse esitada.

2.3. Kasutaja saab tellimuse eest tasuda.

3. Serveri funktsionaalsus

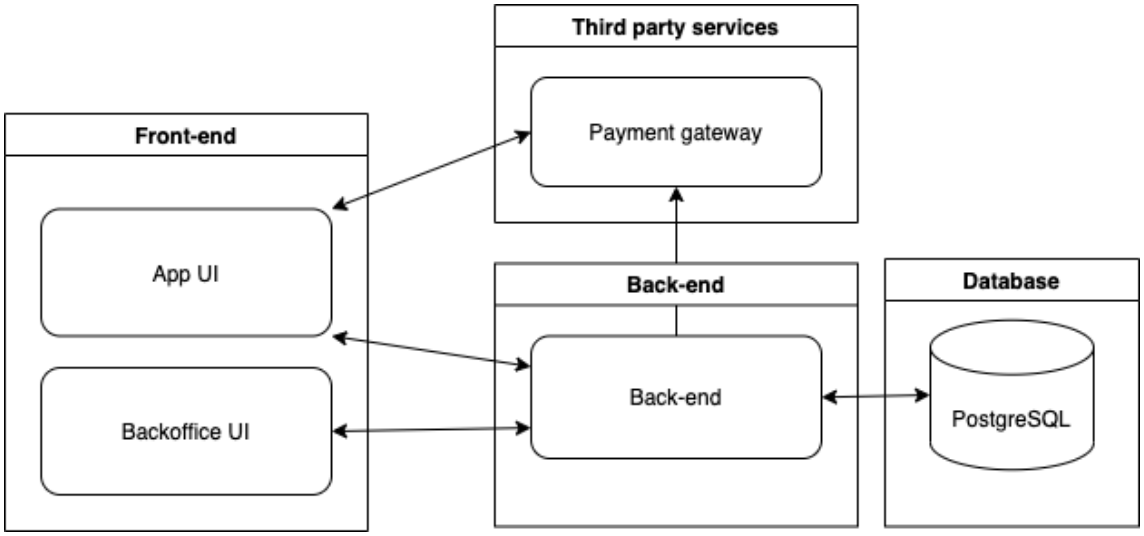
3.1. Restorani jaoks luuakse spetsiaalne unikaalne QR-kood.

3.2. QR-koodi jaoks luuakse spetsiaalsed lühilingid.

3.3. Rakendus peab töötlema maksevärava vastust tellimuse maksmise kohta.

3.2 Veebirakenduse arhitektuur

Antud lõputöö rakenduses on kolm osa: serveri poolne teenus, andmebaas ja makse teenusepakkujatega liidestus. Rakenduse skeem on toodud joonisel 4.

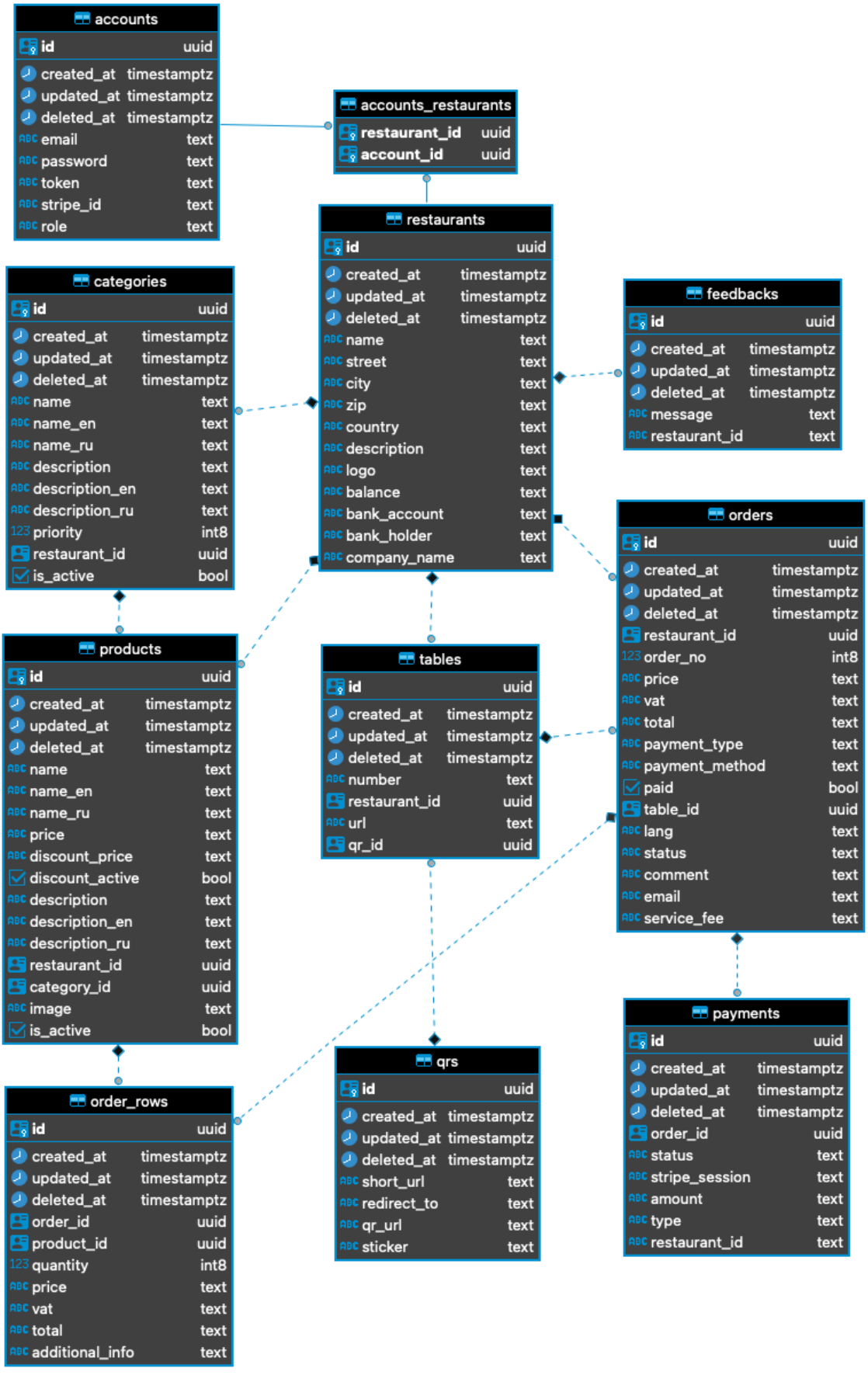


Joonis 4 Rakenduse arhitektuuri skeem.

Front-end plokk on kliendipoolne rakenduse osa, mida ei arvestata selle lõputöö käigus. Serveripoolne rakenduse osa esitatakse monoliitse arhitektuurina. Monoliitse tarkvara kontseptsioon seisneb selles, et rakenduse erinevad komponendid on ühendatud üheks programmiks ühel platvormil. Tavaliselt koosneb monoliitne rakendus andmebaasist, kliendi kasutajaliidesest ja serverirakendusest. Kõik tarkvara osad on ühendatud ja kõiki selle funktsioone hallatakse ühes kohas. [15]
Autori arvamusel aitab monoliitne arhitektuur kiiremini töötavat rakendust luua.

3.3 Andmebaasi arhitektuur

Rakenduse arendamise käigus oli loodud andmebaas milles on 11 tabelit, andmebaasi skeem on toodud joonisel 5. Andmebaas oli loodud GORM'i abil kasutades rakenduses kirjeldatud muutujat ja tüübid.



Joonis 5 Andmebaasi skeem

3.4 Rakenduse arendamine

3.4.1 Autentimine

Autentimine on elektrooniline protsess, mis võimaldab füüsilise või juriidilise isiku e-identimist või elektrooniliste andmete päritolu ja tervikluse kinnitamist. [16]

Autentimine on selle projekti raames vajalik ainult BackOffice pool, kuna autor soovib pakkuda kasutajatele võimalust tellimust teha ilma kasutaja loomist ja autoriseerimist raskendamata. BackOffice'i sisse logimine on vajalik, et klient saaks ennast identifitseerida ning pääseda juurde oma restoranide, kategooriate ja toodete loendisse. Selles projektis registreerimiseks ja autoriseerimiseks kasutatakse JSON Web Token ja Google Sign-In[1].

3.4.1.1 JSON Web Token

JSON Web Token (JWT) on avatud standard (RFC 7519[2]), mis määratleb kompaktse ja iseseisva viisi, kuidas osapoolte vahel JSON-objektina turvaliselt teavet edastada. Seda teavet saab kontrollida ja usaldada, kuna see on digitaalselt allkirjastatud. JWT-sid saab allkirjastada salajase (HMAC-algoritmiga) või avaliku / privaatse võtmepaari abil RSA või ECDSA abil**. [17]**

JSON Web Token koosneb kolmest osast, kõik osad on eraldatud punktiga:

- Päis
- Keha
- Allkiri

Päis koosneb tavaliselt kahest osast: tokeni tüübist, milleks on JWT ja kasutatavast allkirjastamise algoritmist näiteks HMAC SHA256 või RSA. Päis on kodeeritud Base64Url kodeeringuga. [17] Päise näide enne kodeerimist:

[1] https://developers.google.com/identity/sign-in/web/sign-in

[2] https://datatracker.ietf.org/doc/html/rfc7519

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Teine tokeni osa on keha, mis sisaldab nõudeid. Nõuded on avaldused üksuse (tavaliselt kasutaja) kohta ja täiendavad andmed. Nõudeid on kolme tüüpi: registreeritud, avalikud ja eraõiguslikud nõuded. [17]

- Registreeritud nõuded - eelnevalt määratletud nõuete kogum, mis ei ole kohustuslik, kuid soovitatav selleks, et pakkuda kasulike nõuete kogumit. Mõned neist on: *iss* (emitent), *exp* (aegumisaeg), *sub* (subjekt), *aud* (vaatajaskond) [17]
- Avalikud nõuded – vabas vormis määratletud andmed. [17]
- Eraõiguslikud nõuded - kohandatud nõuded, mis on loodud teabe jagamiseks osapoolte vahel, kes nõustuvad nende kasutamises kuid ei ole registreeritud ega avalikud nõuded. [17]

Keha on kodeeritud Base64Url kodeeringuga. [17] Keha näide enne kodeerimist:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Kolmas tokeni osa on allkiri. Allkirja osa loomiseks on vaja võtta kodeeritud päise ja kodeeritud keha ning allkirjastada need osad kasutades päises kirjeldatud algoritmiga. [17]

Allkirja kasutatakse selle kinnitamiseks, et sõnum ei ole muudetud ning privaatvõtmega allkirjastatud tokeni puhul saab see kinnitada ka seda, et tokeni saatja on see, kes on privaatvõtme omanik. Kui midagi tokeni sees on muudetud, siis muutub allkiri mitte kehtivaks.

Näide tokeni allkirja genereerimisega kasutades HMAC SHA256 algoritmi:

```
HMACSHA256(
  base64UrlEncode(päis) + "." +
  base64UrlEncode(keha),
  privaatvõti)
```

[17]

3.4.1.2 Google Sign-In

Google'i sisselogimine haldab OAuth 2.0 voo ja märkide elutsüklit, lihtsustades integreerimist Google'i API-dega. Kasutajal on alati võimalus tühistada juurdepääs rakendusele igal ajal. [18]

Selle lõputöö raames autor integreerib ainult Google Sign-In serveri poolse JWT tokeni verifitseerimist ja vastava kasutaja tokeni genereerimist.

Kehtivuse kontrollimiseks järgmised kriteeriumid peavad olema täidetud:

- Token on Google korralikult allkirjastatud.
- Tokeni "aud" väärtus on võrdne rakenduse kliendi identifitseerimis numbriga.
- Tokeni "iss" on võrdne kas "account.google.com" või https://accounts.google.com
- Tokeni aegumisaeg pole möödas.

Selle kontrollimiseks autor on valinud tokeni genereerimise ja valideerimise pistikprogrammi jwt-go[1], mis loodi just selliste ülesannete lihtsustamiseks.

Pärast tokeni kinnitamist vaja kontrollida, kas kasutaja on juba olemas kasutajate andmebaasis. Kui jah, siis genereerime kasutaja jaoks unikaalse tokeni, kus sees on kirjas tema roll süsteemis. Kui kasutaja pole veel kasutaja andmebaasis, vaja luua tokeni keha andmetest uue kasutajakirje ja genereerida kasutaja jaoks unikaalse tokeni.

3.4.2 URL marsruteerimine

URL-i marsruutimine võimaldab rakenduse aktsepteerima päringu URL-e, mis ei vasta füüsilistele failidele. [19]

[1] https://github.com/dgrijalva/jwt-go

Selle lõputöö raames on kasutuses gorilla/mux[1] marsruteerimise pistikprogramm. Pistikprogramm gorilla/mux rakendab päringute marsruutimist ja dispetšerit sissetulevate päringute vastavusse viimiseks vastava käitlejaga. [20]

```
func main() {
	router := mux.NewRouter()
	router.StrictSlash(true)

	v1 := router.PathPrefix("/api/v1").Subrouter()
	// USER HANDLERS
	v1.HandleFunc("/user/google",
		controllers.GoogleAuth).Methods("POST")
}
```

Igale URL-ile on võimalik seadistada vastutav meetod, mis võtab vastu kõik seadistatud päringu tüüpiga päringud. Üleval on väljatoodud autentimise teenuse marsruteerimise seadistamine.

3.4.3 Andmete hankimine andmebaasist

Autor võtis kasutuselele spetsialiseeritud pistikprogrammi GORM[2].

GORM pakub lihtsalt kasutatavat ja ulatusliku võimalusega kihti rakenduse ja PostgreSQL andmebaasi jaoks. [21]

GORM eelistab konfiguratsiooni asemel konventsiooni, vaikimisi kasutab GORM peamise võtmena ID-d, pluraliseerib struktuuri nime snake_cases tabeli nimeks, snake_case veeru nimeks ja loomise / värskendamise aja jälgimiseks kasutab CreatedAt, UpdatedAt. [22]

Autori rakenduse kasutaja konto struktuur on järgmine:

[1] https://github.com/gorilla/mux

[2] https://gorm.io/

```
type Account struct {
	gorm.Model
	ID          uuid.UUID     `gorm:"type:uuid;primary_key;"`
	Email       string        `json:"email"`
	Role        string        `json:"role";sql:"DEFAULT:'user'"`
	StripeID    string        `json:"-"`
	Restaurants []*Restaurant `gorm:"many2many:accounts_restaurants;"`
	Token       string        `json:"token";sql:"-"`
}
```

GORM määratles gorm.Model struktuur, mis sisaldab välju ID, CreatedAt, UpdatedAt, DeletedAt. [22]

GORM pakub migreerijaliidest, mis sisaldab iga andmebaasi jaoks ühtlustatud API-liideseid, mida saaks kasutada andmebaasist sõltumatute migreerimiste loomiseks. [23]

3.5 Maksmise teenusepakkujatega liidestamine

Et pakkuda tellimuse eest tasumist otse rakendusest, otsustas autor valida maksetöötlejatele Stripe[1] ja Paysera[2].

Stripe pakub mugavat API-d maksete tegemiseks krediitkaardiga, ja lisaks Apple Pay ja Google pay võimalus. On olemas mugav *JavaScript library*, mis annab võimalust lisada vajalikud väljad lehele mõne klikkiga. [24]

Paysera annab võimaluse maksta Eestis populaarsete pankade pangalinkide abil ilma iga pangaga eraldi lepingut sõlmimata. [25]

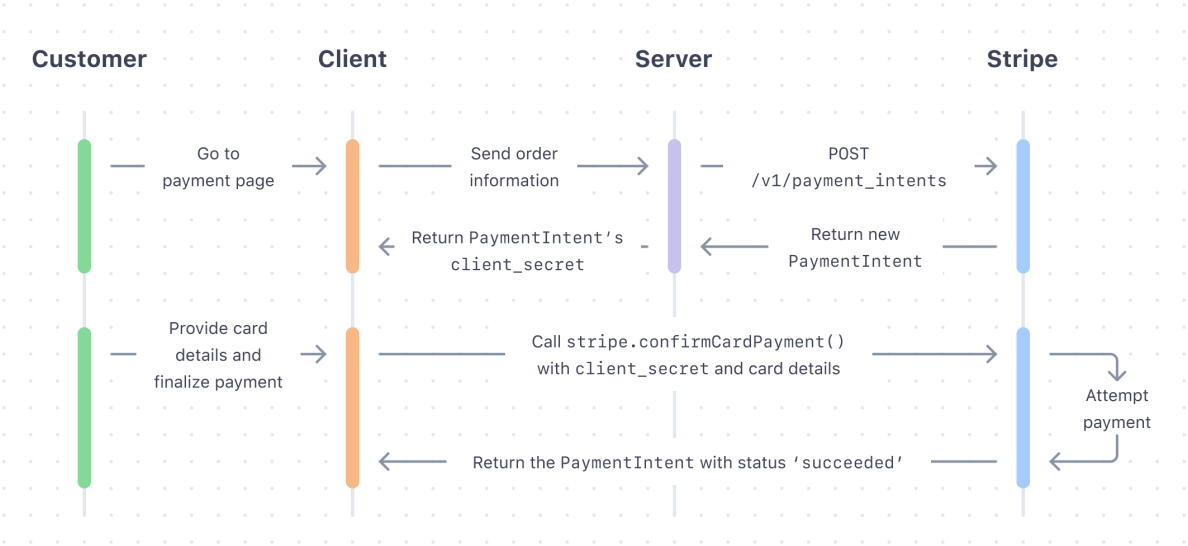
Pangalink on makseviis, kus ostja suunatakse tellimuse kinnitamiseks oma panka. [25]

3.5.1 Stripe

Stripe maksevärava makse elutsükkel on toodud joonisel 6.

[1] https://stripe.com/en-ee

[2] https://www.paysera.ee/v2/et-EE/index



Joonis 6 Stripe maksevärava makse elutsükkel

3.5.1.1 Makse alustamine

Makse alustamiseks süsteem peab saatma Stripe süsteemile *PaymentIntent.* PaymentIntent jälgib kliendi makse elutsüklit, jälgides ebaõnnestunud maksekatseid ja tagades, et makse võetakse ainult üks kord. [26]

Antud lõputöö rakenduses *PaymentIntent* luuakse kui klient kinnitab tellimuse loomist kasutajaliidesest. Suurem osa makse alustamise koodist asub kasutajaliideses, mida see lõputöö ei käsitle.

3.5.1.2 Maksekinnitus

Pärast seda, kui klient on makse sooritanud, saadab Stripe back-endile *webhook* päringu, näidates makse oleku. *Webhook*-de abil Stripe teavitab maksesündmustest, mis toimuvad väljaspool teie maksevoogu. [27]

Autor kasutab selles lõputöös valmis koodi, mida pakub Stripe enda dokumentatsioonis.

Pärast maksekinnitust tellimuse staatus muutub staatusele „makstud“ ja raha lisatakse restorani isiklikule kontole.

3.5.2 Paysera

Paysera ei paku valmis Go koodi, sellepärast autor kirjutas oma integratsiooni koodi kasutades Paysera dokumentatsiooni.[1]

3.5.2.1 Makse alustamine

Makse alustamiseks vaja POST-meetodi abil andmed saatma spetsiaalsele aadressile.

Kõik parameetrid on liidetud URL-encoded ridaks (Joonis 7).

```
['param1' => 'abc', 'param2' => 'Some string with symbols %=&']
    'param1=abc&param2=Some+string+with+symbols+%25%3D%26'
```

Joonis 7 URL-encoded rida

Saadud rida kodeeritakse kasutades base64 kodeeringut (Joonis 8).

```
’param1=abc&param2=Some+string+with+symbols+%25%3D%26'
'cGFyYW0xPWFiYyZwYXJhbTI9U29tZStzdHJpbmcrd2l0aCtzeW1ib2xzKyUyNSUzRCUyNg=='
```

Joonis 8 Base64 kodeeritud rida

Tulemuses asendatakse stringisümbolid "/" tähtedega "_" ja sümbolid "+" tähisega "-". Saame sarnase base64 kodeeringuga, mida on URL-i saatmine ilma täiendava töötlemiseta turvaline. (Joonis 9)

```
'MViDYlV7V0iHR2w2OkJjRFFpY11hizJDhk+EZjl/'
'MViDYlV7V0iHR2w2OkJjRFFpY11hizJDhk-EZjl_'
```

Joonis 9 URL-i saatmise turvaline rida

Lõpptulemuste string kirjutatakse alla - genereeritakse *sign* parameeter. *Sign* parameetri genereerimise algoritm toodud joonisel 10.

```
sign = md5(data + password)
```

Joonis 10 Sign parameetri genereerimise algoritm. Kus *md5* on krüptograafiline räsifunktsioon, *data* - kodeeritud parameetrid, *password* – paysera projekti parool.

3.5.2.2 Maksekinnitus

Pärast seda, kui klient on makse sooritanud, saadab Paysera back-endile *Callback* päringu, kus sees on kirjas kõik vajalikud makse töötlemiseks andmed. [28]

Callbackurl-ile lisatakse 3 täiendavat GET-parameetrit:

[1] https://developers.paysera.com/en/checkout/integrations/integration-specification

- data - kodeeritud parameetrid Paysera süsteemist. Parameetrite sõelumiseks tuleb teha 3 toimingut:
    - Muutke sümbolid "-" sümboliks "+", "_" sümboliks "/"
    - Dekodeerige string, kasutades base64 kodeeringut
    - Too parameetrite massiiv dekodeeritud stringist, mis on *URL-encoded* parameetristring
- ss1 – *data* parameetri allkiri, allkirjastamise algoritm toodud joonisel 10. [28]
- ss2 – *data* parameetri allkiri, allkirjastatud kasutades RSA privaatvõtme koos SHA-1 räsi funktsiooniga. [28]

Enne tellimuse kinnitamist vaja kontrollida vähemalt ühte allkirja. Võimaluse korral kontrollida alati kõrgema turvalisusega ss2 allkirja. [28]

Pärast maksekinnitust tellimuse staatus muutub staatusele „makstud“ ja raha lisatakse restorani isiklikule kontole.

3.6 Loodud veebirakenduse majutamine

Kõik rakenduse osad on majutatud DigitalOceani peal. DigitalOcean on pilveteenus, mis pakkub erinevaid lahendusi veebirakenduste majutamiseks. [29] Selline valik oli tehtud, kuna DigitalOcean tundus lihtsam kasutada ja see pakub majutamiseks kahe kuu jooksul 100 dollarit proovimiseks. [30]

Domeeni nimi on seadistatud kasutamiseks Cloudflare teenusega. Cloudflare on teenus, mis pakub mugavat DNS-i haldussüsteemi, DDoS-kaitse, CDN teenust ja tasuta *https* võimalust ilma veebiserveri seadistamiseta. [31]

3.6.1 DigitalOcean

3.6.1.1 Back-end majutamine

Back-end majutamiseks oli valitud DigitalOcean Droplet Ubuntu 20.04 (LTS) x64 operatsiooni süsteemiga.

DigitalOcean Droplets on Linuxil põhinevad virtuaalsed masinad, mis töötavad virtualiseeritud riistvara peal. Iga loodud Droplet on uus server, mida on võimalik kasutada kas eraldi või suurema pilvepõhise infrastruktuuri osana. [32]

Pärast Dropleti loomist on vaja seadistada veebiserveri, autor valis Apache veebiserveri, kuna testimise masinas oli kasutatud Apache veebiserver ja kõik konfiguratsioonid olid juba valmis tehtud. Veebiserveri konfiguratsioon toodud joonisel 11.

```
<VirtualHost *:80>

ServerName assets.snakk.ee
ServerAlias assets.snakk.ee
DocumentRoot /var/snakk

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>

<VirtualHost *:80>
ProxyPreserveHost On
ProxyRequests Off
ServerName api.snakk.ee
ServerAlias api.snakk.ee

ProxyPass / http://localhost:8000/
ProxyPassReverse / http://localhost:8000/

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Joonis 11 Apache konfiguratsioon

Joonisel 11 on näha, et Apache server kuulab 80 porti ja suunab päringud vastavalt alamdomeeni nimele. Päringud mis tulevad alamdomeenile „api.snakk.ee“ on ümber suunatud sisemisele 8000 pordile, mille peal töötab Go back-end server.

Go back-end serveri käivitamiseks on vaja kompileerida back-end koodi käivitatavasse failidesse [33] (Joonis 12).

```
env GOOS=linux GOARCH=amd64 go build
```

Joonis 12 Konsooli käsk projekti kompileerimiseks käivitatavasse failidesse

Kui back-end on kompileeritud ja käivitatav fail on loodud - vaja faili üleslaadida serverisse. Selle faili käivitamiseks ei ole vaja midagi installida.

Et back-end töötaks püsivalt, vaja luua systemd-teenuse, mis käivitab back-end taaskäivitamisel.

systemd on Linuxi süsteemi põhiliste ehitusplokkide komplekt. See pakub süsteemi- ja teenusehaldurit, mis töötab PID 1-na ja käivitab ülejäänud süsteemi. [34]

Back-end teenuse kood on toodud joonisel 13.

```
[Unit]
Description=simple go application

[Service]
Type=simple
Restart=always
RestartSec=5s
WorkingDirectory=/var/snakk
ExecStart=/var/snakk/snakk-backend

[Install]
WantedBy=multi-user.target
```

Joonis 13 Back-end systemd teenuse kood

Back-end serveri käivitamine toimub ühe käsuga (Joonis 14).

```
service backend start
```

Joonis 14 Back-end serveri käivitamine

3.6.1.2 Andmebaasi majutamine

DigitalOcean on teinud andmebaasi majutamise teenuse, mille pärast ei ole vaja eraldi serveri seadistama.

DigitalOceani hallatavad andmebaasid on täielikult hallatud, suure jõudlusega andmebaasiklastriteenus. Hallatud andmebaaside kasutamine on võimas alternatiiv andmebaaside käsitsi installimisele, konfigureerimisele, hooldamisele ja turvamisele. [35]

4 Rakenduse testimine

Rakenduste testimine kontrollib veebirakendust võimalike vigade suhtes enne selle avaldamist. Veebirakenduse testimine kontrollib veebirakenduse funktsionaalsust, kasutatavust, turvalisust, ühilduvust, toimivust. [36]

4.1 Automaattestimine

Unit testimine on tarkvara testimise tüüp, kus testitakse tarkvara üksuseid või komponente. Selle eesmärk on kontrollida, kas tarkvarakoodi iga üksus toimib ootuspäraselt. Ühikute testimine toimub arendajate poolt rakenduse väljatöötamise ajal. *Unit* testid eraldavad koodiosa ja kontrollivad selle õigsust. Üksus võib olla üksikfunktsioon, meetod, protseduur, moodul või objekt. [37]

```
import "testing"

func TestAbc(t *testing.T) {
	t.Error() // to indicate test failed
}
```

Joonis 15 Go Unit testi näide

Joonisel 15 on välja toodud Go *Unit* testi põhistruktuur. Go-s on olemas sisseehitatud *testing* pakett. *Unit* test on funktsioon, mis aktsepteerib tüübi **testing.T* ja kutsub sellele *Error*. See funktsioon peab algama märksõnaga Test ja viimane nimi peaks algama suurtähega (näiteks TestMultiply ja mitte Testmultiply). [38]

Kasutaja päringu simuleerimiseks autor kasutab Go-s sisseehitatud *http* paketti, millega on võimalik saata *http* päringud. Joonisel 16 on väljatoodud testi kood, mis kontrollib kas tellimus oli salvestatud korrektselt ja hind oli õigesti arvutatud.

```
func TestCreateOrder(t *testing.T) {
	productQuantityString := fake.DigitsN(1)
	productQuantity, _ := strconv.Atoi(productQuantityString)

	testData := prepareData(productQuantity)
	orderComment := fake.Sentence()

	var jsonStr = []byte(`
		{
			"restaurant_id": "` + testData.RestaurantID.String() +
`",
			"table_no": "` + testData.TableID.String() + `",
			"order_rows": [
				{
					"product_id": "` +
testData.ProductID.String() + `",
					"quantity": ` + productQuantityString + `
				}
			],
			"lang": "et",
			"payment_type": "banklink",
			"payment_method": "swedbank",
			"comment": "` + orderComment + `"
		}`)

	req, err := http.NewRequest("POST", "/api/v1/orders",
bytes.NewBuffer(jsonStr))
	if err != nil {
		t.Fatal(err)
	}
	req.Header.Set("Content-Type", "application/json")

	// We create a ResponseRecorder (which satisfies http.ResponseWriter)
to record the response.
	rr := httptest.NewRecorder()
	handler := http.HandlerFunc(controllers.CreateOrder)

	// Our handlers satisfy http.Handler, so we can call their ServeHTTP
method
	// directly and pass in our Request and ResponseRecorder.
	handler.ServeHTTP(rr, req)

	// Check the status code is what we expect.
	if status := rr.Code; status != http.StatusOK {
		t.Errorf("handler returned wrong status code: got %v want %v",
			status, http.StatusOK)
	}

	// Check the response body is what we expect.
	expected := `"redirect":"https://bank.paysera.com/pay/?data=`
	if !strings.Contains(rr.Body.String(), expected) {
```

```
			t.Errorf("handler returned unexpected body: got %v want %v",
				rr.Body.String(), expected)
		}

		var orderReturn OrderReturn

		errDecode := json.NewDecoder(rr.Body).Decode(&orderReturn)
		if errDecode != nil {
			t.Errorf("Error while decoding request body: " +
errDecode.Error())
		}

		assert.NotEmpty(t, orderReturn.Redirect)
		assert.NotEmpty(t, orderReturn.OrderID)

		orderModel := models.GetOrder(orderReturn.OrderID)
		assert.EqualValues(t, orderComment, orderModel.Comment)

		nextOrderNo := orderModel.GenerateOrderNo()
		assert.EqualValues(t, nextOrderNo-1, orderModel.OrderNo)

		expectedPrice := testData.ExpectedPrice
		actualPrice := orderModel.Price
		expectedVat := testData.ProductVat
		actualVat := orderModel.Vat
		expectedTotal := testData.ProductTotal
		actualTotal := orderModel.Total
		expectedCents :=
int(expectedTotal.Mul(decimal.NewFromInt(100)).IntPart())

		assert.EqualValues(t, expectedPrice, actualPrice)
		assert.EqualValues(t, expectedVat, actualVat)
		assert.EqualValues(t, expectedTotal, actualTotal)
		assert.EqualValues(t, expectedCents,
orderModel.GetOrderAmountInCents())

		orderList := make([]*models.Order, 0)
		orderList = append(orderList, orderModel)

		statusFiltered := models.FilterByStatus(orderList, "pending")
		assert.Contains(t, statusFiltered, orderModel)

		statusFiltered = models.FilterByStatus(orderList, "progress")
		assert.Empty(t, statusFiltered)

		assert.EqualValues(t, "pending", orderModel.Status)
		orderModel.ChangeStatus("progress")
		assert.EqualValues(t, "progress", orderModel.Status)
		orderModel.ChangeStatus("done")
		assert.EqualValues(t, "done", orderModel.Status)
		orderModel.ChangeStatus("finished")
```

```
	assert.EqualValues(t, "finished", orderModel.Status)

	statusFiltered = models.FilterByStatus(orderList, "finished")
	assert.Contains(t, statusFiltered, orderModel)

	assert.EqualValues(t, "swedbank", orderModel.PaymentMethod)
	assert.EqualValues(t, "banklink", orderModel.PaymentType)
	assert.EqualValues(t, testData.TableID.String(),
orderModel.Table.ID.String())
	assert.EqualValues(t, testData.RestaurantID.String(),
orderModel.Restaurant.ID.String())
	assert.EqualValues(t, false, orderModel.Paid)

	for _, row := range orderModel.GetOrderRows() {
		assert.EqualValues(t, productQuantity, row.Quantity)
		assert.EqualValues(t, orderModel.Price,
row.Price.Mul(decimal.NewFromInt(int64(productQuantity))))
		assert.EqualValues(t, orderModel.Vat, row.Vat)
		assert.EqualValues(t, orderModel.Total, row.Total)
		assert.EqualValues(t, testData.ProductID.String(),
row.ProductID.String())
		assert.EqualValues(t, orderModel.ID, row.OrderID)
	}
}
```

Joonis 16 Tellimuse loomise Unit test

5 Turvalisuse analüüs

Tungimistestimine, mida nimetatakse ka eetiliseks häkkimiseks, on arvutisüsteemi, võrgu või veebirakenduse testimise tava, et leida turvanõrkusi, mida ründaja saaks ära kasutada. Tungimistesti saab tarkvararakendustega automatiseerida või käsitsi läbi viia. Mõlemal juhul hõlmab see protsess enne testi sihtmärgi kohta teabe kogumist, võimalike sisenemiskohtade väljaselgitamist, kas virtuaalset või tegelikku sissemurdmist ja avastustest teatamist. [39]

Veebirakenduse testimiseks olid valitud järgmised nõuded *hackercombat*[1] portaalist.

Tuvastatud nõrgemat küljed:

[1] https://hackercombat.com/web-application-penetration-testing-checklist/

- Tagasiside vormi test – tagasiside vorm on rämpsposti rünnakute suhtes vastuvõtlik, kuna sellel pole nende eest kaitset. Kõige lihtsam lahendus on CAPTCHA lisamine.

- Kasutaja seansi test – praegu on loodud tokeni eluiga liiga pikk, nii et ründaja, kes tokeni juurde pääseb, saab seda kasutada.

Tugevad küljed:

- HTTPS protokolli kasutamine – Cloudflare abil on võimalik kasutada turvalist HTTPS protokolli ilma täiendatava veebiserveri seadistusega, see kaitseb selliste rünnakute eest nagu *man-in-the-middle*.

- Avatud portide testimine – kõik pordid, mida pole rakenduse toimimiseks vaja, on väljastpoolt juurdepääsemiseks suletud.

- Kasutajatunnus ja parooli testimine - kuna kasutatakse Google'i sisselogimist, on saidile kellegi teise kasutajaga võimalik siseneda ainult siis, kui ründaja saab juurdepääsu omaniku Google'i kontole.

- SQL injection testimine – kasutuses on GORM, mis väldib SQL-koodi sisse kirjutamist. [40]

- Juurdepääsu lubade testimine – kõigil andmete redigeerimisega seotud *endpoint*idel on kasutaja nõuetele vastavuse kontroll olemas.

- DoS testimine - Veebirakendus on DoS (*Denial of Service*) rünnakute eest kaitstud Cloudflare abil.

6 Tulemus

6.1 Tulemus

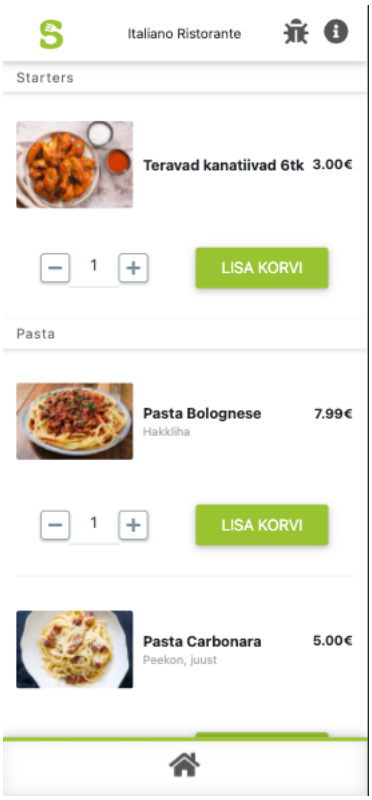
Antud lõputöö käigus oli arendatud minimaalne töötav ja testitav veebirakenduse Back-end ja andmebaas. Antud rakenduses on võimalik:

- Klient saab lisada ja hallata oma restorani andmed.
- Klient saab lisada ja hallata restorani toidu kategooriad.
- Klient saab lisada ja hallata restorani toidu.
- Klient saab sisse logida kasutades oma Google kontot.
- Klient saab vaadata oma restorani tellimuste ajalugu.
- Klient saab printida välja unikaalse QR koodi, mille kaudu kasutaja saab tellimusi esitada.
- Kasutaja saab vaadata restoranis pakutavat toidu.
- Kasutaja saab esitada tellimust restoranile.
- Kasutaja saab maksta tellimuse eest veebirakenduses.

Unikaalne QR kood, mida on genereeritud rakenduse kaudu on toodud joonisel 17.

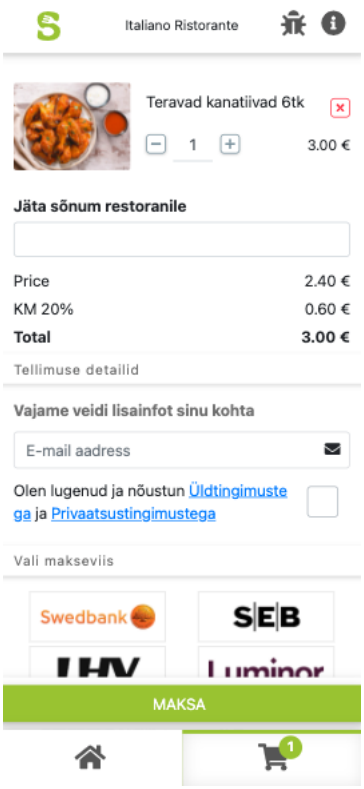
Joonis 17 QR koodi näidis

Kasutaja esimene vaade on toodud joonisel 18.



Joonis 18 Restoraani toidu valiku näide

Kasutaja ostukorv koos maksevõimalusega toodud joonisel 19.



Joonis 19 Kasutaja ostukorv

Rakenduses on tehtud ka tagasiside vorm, mille kaudu kasutajad saavad oma soovitusi ja kaebusi esitada.

Oli loodud ka täiesti töötav minimaalne BackOffice leht, kus restorani omanik saab hallata oma restorani informatsiooni, toodet ja tellimused. BackOffice lehed on välja toodud Lisa 2 peatükis.

7 Kokkuvõte

Käesoleva lõputöö käigus oli loodud tellimuste platvormi serveri osa. Antud rakenduse kasutamine peaks aitama restoraniomanikel pakkuda klientidele uut tellimisviisi. Töö käigus oli tehtud serveri tehnoloogiate ja andmebaaside haldussüsteemide analüüs, antud rakenduse arhitektuuri ülevaade, toodud välja põhiosad arendusest ja testimisest, välja toodud turvalisuse analüüsi tulemused.

Antud rakenduses saab restorani omanik sisse logida kasutades oma Google kontot, lisada ja hallata oma restorani, lisada ja hallata restorani toidu kategooriad, lisada ja hallata restorani toidu, jälgida restorani tellimuste ajalugu. Selleks, et klient saaks restorani lehele juurde pääseda, peab restorani omanik printima spetsiaalse QR-koodi ja asetama selle silmatorkavasse kohta. Restorani klient saab skaneerida QR-koodi kasutades oma nutiseadet, pääseda juurde restorani lehele, valida restoranis pakutavat toidu, maksta tellimuse eest ja esitada tellimust restorani omanikule.

Restorani omanikute sisse logimine oli tehtud kasutades Google Sign-In nuppu koos OAuth2.0 protokolliga ja tokeni tüübiks oli valitud JWT. Autori arvamusest see aitab lihtsustada rakenduse juurdepääsu, kuna kliendile ei ole vaja luua uut kasutajat ja välja mõelda uut parooli. See ka peaks vältima olulise isikuandmete kadumist.

Arenduse tulemuste põhjal võib teha järelduse, et minimaalne töötav tellimuste platvorm on valmis teenindama esimesi kliente ja jätkama arendamist vastavalt kasutajate soovidele.

Kasutatud kirjandus

[1] „NodeJs,“ [Võrgumaterjal]. Available: https://nodejs.org/en/about/. [Kasutatud 08 05 2021].

[2] AltexSoft Inc., „The Good and the Bad of Node.js Web App Development,“ [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/. [Kasutatud 08 05 2021].

[3] npm, Inc., „NpmJS,“ [Võrgumaterjal]. Available: https://www.npmjs.com/. [Kasutatud 08 05 2021].

[4] „PHP,“ [Võrgumaterjal]. Available: https://www.php.net/. [Kasutatud 08 05 2021].

[5] C. Ninjas, „Why Use PHP programming in 2021? Its Pros and Cons,“ 16 04 2021. [Võrgumaterjal]. Available: https://www.codingninjas.com/blog/2021/04/16/why-use-php-programming-in-2021-its-pros-and-cons/. [Kasutatud 08 05 2021].

[6] Google Inc., „Documentation - The Go Programming Language,“ [Võrgumaterjal]. Available: https://golang.org/doc/. [Kasutatud 08 05 2021].

[7] J. K, „Node.js vs Golang: Settling the debate for Good | Acodez,“ 10 02 2021. [Võrgumaterjal]. Available: https://acodez.in/node-js-vs-golang/.

[8] B. Peabody, „Server-side I/O Performance: Node vs. PHP vs. Java vs. Go,“ [Võrgumaterjal]. Available: https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go. [Kasutatud 13 05 2021].

[9] Oracle Corporation, „MySQL :: MySQL Products,“ [Võrgumaterjal]. Available: https://www.mysql.com/products/. [Kasutatud 09 05 2021].

[10] K. Hristozov, "MySQL vs PostgreSQL -- Choose the Right Database for Your Project," 19 07 2019. [Online]. Available: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres. [Accessed 09 05 2021].

[11] The PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database," [Online]. Available: https://www.postgresql.org/. [Accessed 09 05 2021].

[12] O. Tsarev, „PostgreSQL vs MySQL,“ Mail.ru Group, 27 01 2015. [Võrgumaterjal]. Available: https://habr.com/ru/company/mailru/blog/248845/. [Kasutatud 09 05 2021].

[13] Oracle Corporation, „PostgreSQL Type Mapping,“ [Võrgumaterjal]. Available: https://dev.mysql.com/doc/workbench/en/wb-migration-database-postgresql-typemapping.html. [Kasutatud 09 05 2021].

[14] R. Dhumal, „Everything you need to know about PostgreSQL triggers,“ 18 11 2019. [Võrgumaterjal]. Available: https://www.enterprisedb.com/postgres-tutorials/everything-you-need-know-about-postgresql-triggers. [Kasutatud 09 05 2021].

[15] A. D., „Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless?,“ [Võrgumaterjal]. Available: https://rubygarage.org/blog/monolith-soa-microservices-serverless. [Kasutatud 12 05 2021].

[16] E. L. Teataja, „EUROOPA PARLAMENDI JA NÕUKOGU MÄÄRUS (EL) nr 910/2014,“ [Võrgumaterjal]. Available: https://eur-lex.europa.eu/legal-content/ET/TXT/HTML/?uri=CELEX:32014R0910. [Kasutatud 10 05 2021].

[17] Auth0 Inc., „JSON Web Token Introduction,“ [Võrgumaterjal]. Available: https://jwt.io/introduction. [Kasutatud 10 05 2021].

[18] Google Inc., „Integrating Google Sign-In into your web app,“ [Võrgumaterjal]. Available: https://developers.google.com/identity/sign-in/web/sign-in. [Kasutatud 10 05 2021].

[19] E. Reitan, „URL Routing,“ 09 08 2014. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/url-routing. [Kasutatud 10 05 2021].

[20] K. Kisiel ja M. Silverlock, „gorilla/mux,“ 01 08 2016. [Võrgumaterjal]. Available: https://github.com/gorilla/mux/blob/master/README.md. [Kasutatud 10 05 2021].

[21] Jinzhu, „GORM Guides,“ [Võrgumaterjal]. Available: https://gorm.io/docs/index.html#Overview. [Kasutatud 12 05 2021].

[22] Jinzhu, „Declaring Models,“ [Võrgumaterjal]. Available: https://gorm.io/docs/models.html#Conventions. [Kasutatud 12 05 2021].

[23] Jinzhu, „Migration,“ [Võrgumaterjal]. Available: https://gorm.io/docs/migration.html. [Kasutatud 12 05 2021].

[24] Stripe Inc., „Stripe Checkout,“ [Võrgumaterjal]. Available: https://stripe.com/docs/payments/checkout. [Kasutatud 13 05 2021].

[25] Paysera EE OÜ, „Maksevärav sinu e-poele I Paysera Checkout,“ [Võrgumaterjal]. Available: https://www.paysera.ee/v2/et-EE/checkout-maksevarav. [Kasutatud 13 05 2021].

[26] Stripe Inc., „Custom payment flow,“ [Võrgumaterjal]. Available: https://stripe.com/docs/payments/integration-builder. [Kasutatud 13 05 2021].

[27] Stripe Inc., „Triggering actions with webhooks,“ [Võrgumaterjal]. Available: https://stripe.com/docs/payments/handling-payment-events. [Kasutatud 13 05 2021].

[28] Paysera LT, „Callback,“ [Võrgumaterjal]. Available: https://developers.paysera.com/en/checkout/integrations/integration-callback. [Kasutatud 13 05 2021].

[29] DigitalOcean, LLC, „DigitalOcean – The developer cloud,“ [Võrgumaterjal]. Available: https://www.digitalocean.com/. [Kasutatud 13 05 2021].

[30] DigitalOcean, LLC, „Referral Program,“ [Võrgumaterjal]. Available: https://www.digitalocean.com/referral-program/. [Kasutatud 13 05 2021].

[31] Cloudflare, Inc., „Cloudflare - The Web Performance & Security Company | Cloudflare,“ [Võrgumaterjal]. Available: https://www.cloudflare.com/. [Kasutatud 13 05 2021].

[32] DigitalOcean, LLC, „Recommended Initial Droplet Configuration,“ 02 06 2020. [Võrgumaterjal]. Available: https://docs.digitalocean.com/droplets/tutorials/recommended-setup/. [Kasutatud 13 05 2021].

[33] M. Mudrinić, „How To Build Go Executables for Multiple Platforms on Ubuntu 16.04,“ 30 05 2017. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/tutorials/how-to-build-go-executables-for-multiple-platforms-on-ubuntu-16-04. [Kasutatud 13 05 2021].

[34] systemd, „systemd,“ [Võrgumaterjal]. Available: https://systemd.io/. [Kasutatud 13 05 2021].

[35] DigitalOcean, LLC, „Managed Databases,“ 14 02 2019. [Võrgumaterjal]. Available: https://docs.digitalocean.com/products/databases/. [Kasutatud 13 05 2021].

[36] Guru99, „Web Application Testing: 8 Step Guide to Website Testing,“ [Võrgumaterjal]. Available: https://www.guru99.com/web-application-testing.html. [Kasutatud 13 05 2021].

[37] Guru99, „Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE,“ [Võrgumaterjal]. Available: https://www.guru99.com/unit-testing-guide.html. [Kasutatud 13 05 2021].

[38] U. Hiwarale, „Unit Testing made easy in Go,“ 11 05 2019. [Võrgumaterjal]. Available: https://medium.com/rungo/unit-testing-made-easy-in-go-25077669318. [Kasutatud 14 05 2021].

[39] L. Rosencrance ja P. Mehta, „What is pen test (penetration testing)? - Definition from WhatIs.com,“ 10 2018. [Võrgumaterjal]. Available: https://searchsecurity.techtarget.com/definition/penetration-testing. [Kasutatud 14 05 2021].

[40] Jinzhu, „Security | GORM - The fantastic ORM library for Golang, aims to be developer friendly.,“ [Võrgumaterjal]. Available: https://gorm.io/docs/security.html. [Kasutatud 14 05 2021].

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

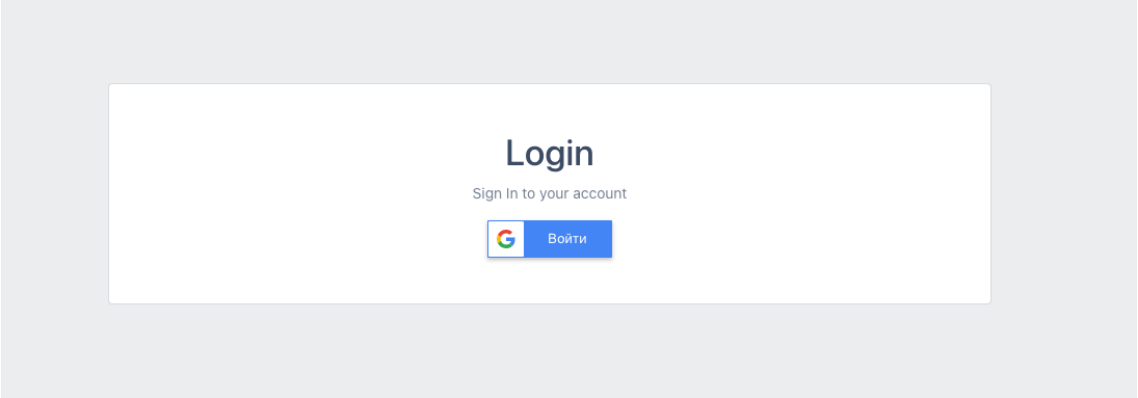
Mina, Nikita Popov

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tellimuste platvormi serveri osa loomine toitlustuskohtadele“, mille juhendaja on Aleksei Talisainen
   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.

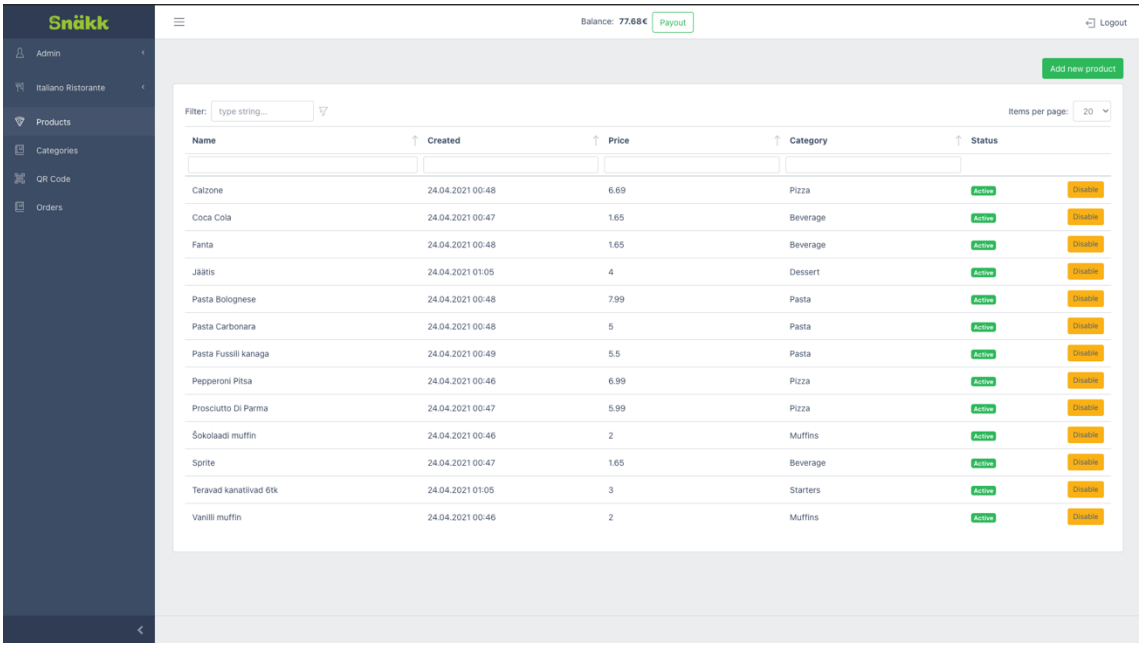
18.05.2021

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.

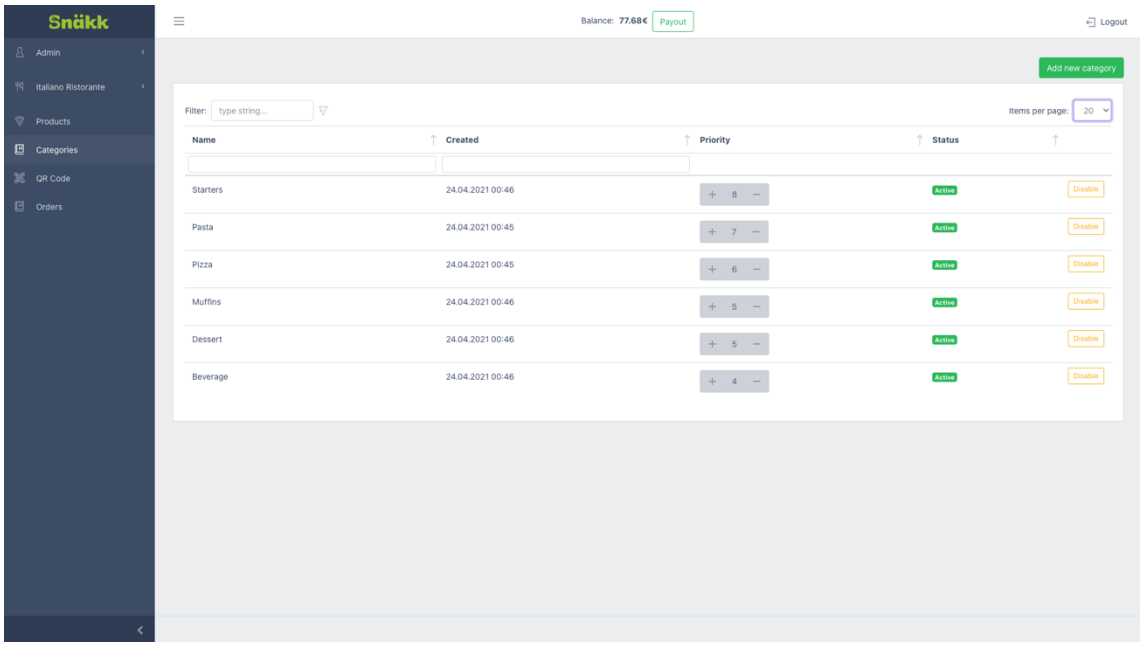
Lisa 2 – BackOffice lehed.



Joonis 1 BackOffice sisse logimine

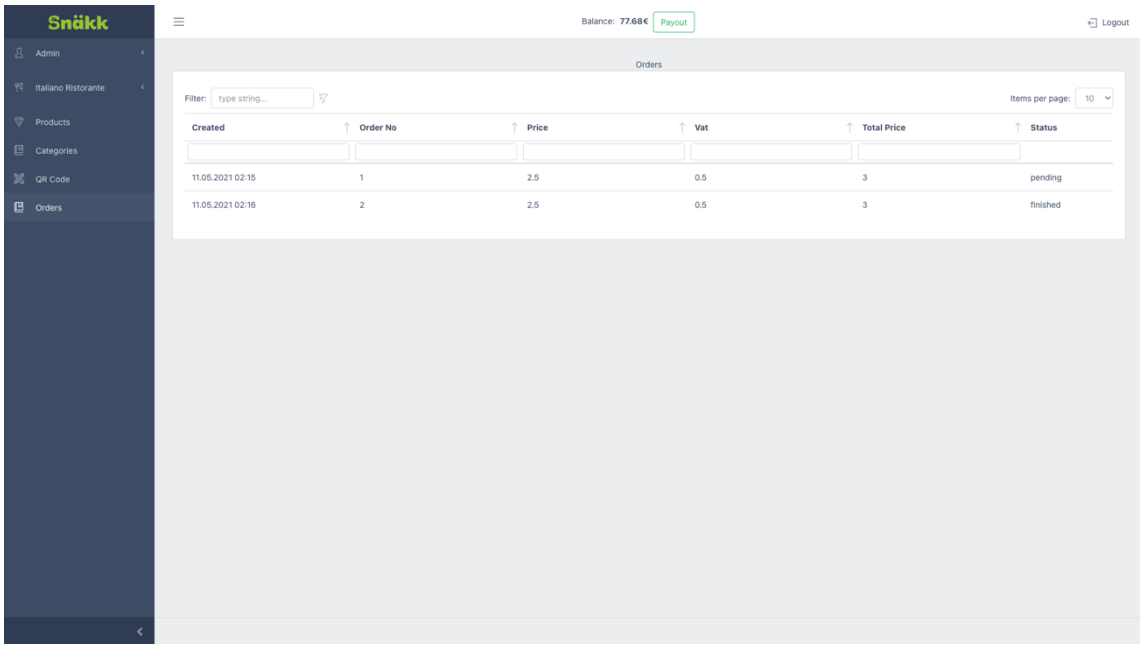


Joonis 2 BackOffice toodete loetelu

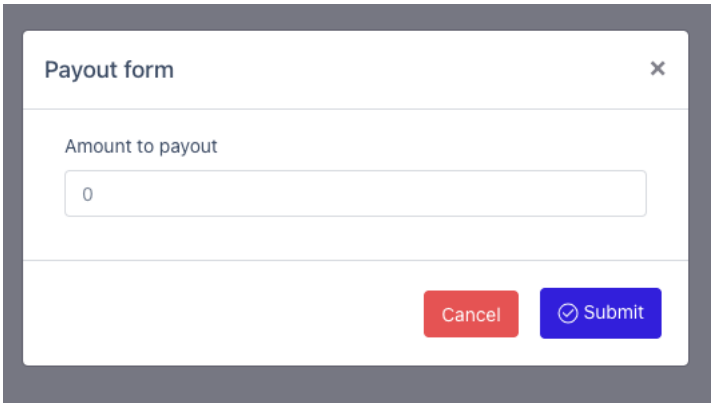


Joonis 3 BackOffice kategooriate loetelu

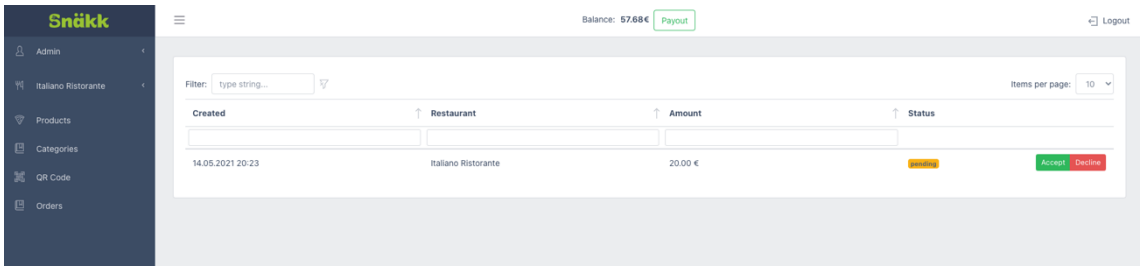
Joonis 4 BackOffice restorani unikaalne QR kood



Joonis 5 BackOffice tellimuste ajalugu



Joonis 6 BackOffice väljamakse aken



Joonis 7 BackOffice administaatori väljamakse päringute loetelu