TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Raimo Johanson IADB192855

E-kaubanduse makselahenduste portaali arendamine Ridango AS-i näitel

Bakalaureusetöö

Juhendaja: Meelis Antoi
Magistrikraad

Tallinn 2022

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: Raimo Johanson

02.03.2022

Annotatsioon

Käesoleva lõputöö ülesanne on luua tarkvaralahendus, mis on töö tellija – Ridango AS – e-kaubanduse maksete vahendamise portaaliks ning platvormiks makseteenuseosutajate haldamiseks.

Autor seadis töö eesmärgiks jõuda olukorda, kus klientrakendustel oleks võimalik liidestumise arendamisega alustada, süsteem suudaks vahendada ühe makseteenuseosutaja makseviise ning loodud lahendus oleks sobivaks lähtepunktiks tulevastele arendustele.

Lõputöö tulemusena valminud tarkvaralahendus võimaldab pakkuda teenindavatele rakendustele makseviisi täpsusega erinevate makseteenuseosutajate liidestusi ja kuvada kasutajale tellimuse ja makseviisi valikuks kasutajaliidest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 5 peatükki, 10 joonist, 1 tabelit.

Abstract

Development of an E-commerce Payment Solutions Portal on the Example of Ridango AS

The goal of this thesis is the development of software solution which will be the basis of e-commerce payments mediation and the management of payment service providers for Ridango AS.

The author set the goal for the thesis to be able to provide an interface for client applications – enabling to start the development, to provide payment methods from one payment service provider and that the software solution would be appropriate starting point for future developments.

The software solution created as the result of this thesis enables the configuration of payment service provider on payment method basis and offers a user interface for payment service users enabling them to view order details and select a payment method.

The thesis is in Estonian and contains 35 pages of text, 5 chapters, 10 figures, 1 table.

Lühendite ja mõistete sõnastik

AOP	*Aspect-Oriented Programming,* programmeerimise paradigma
API	*Application Programming Interface*, rakenduse programmeerimisliides
CORS	Cross-Origin Resource Sharing, päritoluülene ressursside jagamine
CSS	*Cascading Style Sheets,* stiililehe kirjelduskeel
DOM	*Document Object Model*, dokumendi objekti mudel
DTO	*Data Transfer Object,* Andmete transpordi objekt
ERD	*Entity Relationship Diagram,* olemi-suhte diagramm
HTML	*HyperText Markup Language,* veebidokumendi struktureerimise keel
HTTP	*HyperText Transfer Protocol,* võrgupäringute protokoll
ID	*Identifier,* identifikaator
JDBC	*Java Database Connectivity,* Java API andmebaasiga ühendamiseks
JPA	*Java Persistence API,* Java püsivuse API
JSON	*JavaScript Object Notation,* Javascripti objekti notatsioon
MVC	*Model-View-Controller,* tarkvaraarhitektuuri suunis
OB	*Open Banking* - standard finantsasutuste süsteemide vahelisele liidestusele
ORM	*Object–Relational Mapping,* objektidevaheliste relatsioonide kaardistamine
POS	*Point of Sale,* müügikoht – käesolevas töös koondnimetus klientsüsteemidele
PSD2	*Payment Services Directive 2,* makseteenuste direktiivi teine parandus
PSP	*Payment Service Provider,* makseteenuseosutaja
PSU	*Payment Service User,* makseteenuse kasutaja
QA	*Quality Assurance,* kvaliteedi tagamine
REST	*Representational State Transfer,* tarkvararakenduse API suunis
SEO	*Search Engine Optimization,* otsingumootorile optimeerimine
SPA	*Single Page Application* – üheleheline rakendus
SQL	*Structured Query Language,* relatsioonilise andmebaasi päringukeel
URL	*Uniform Resource Locator,* veebiaadress / viide ressursile arvutivõrgus
WCAG	Web Content Accessibility Guidelines, *veebisisu ligipääsetavuse suunised*
XHR	*XMLHttpRequest,* võrgupäringute objekt

Sisukord

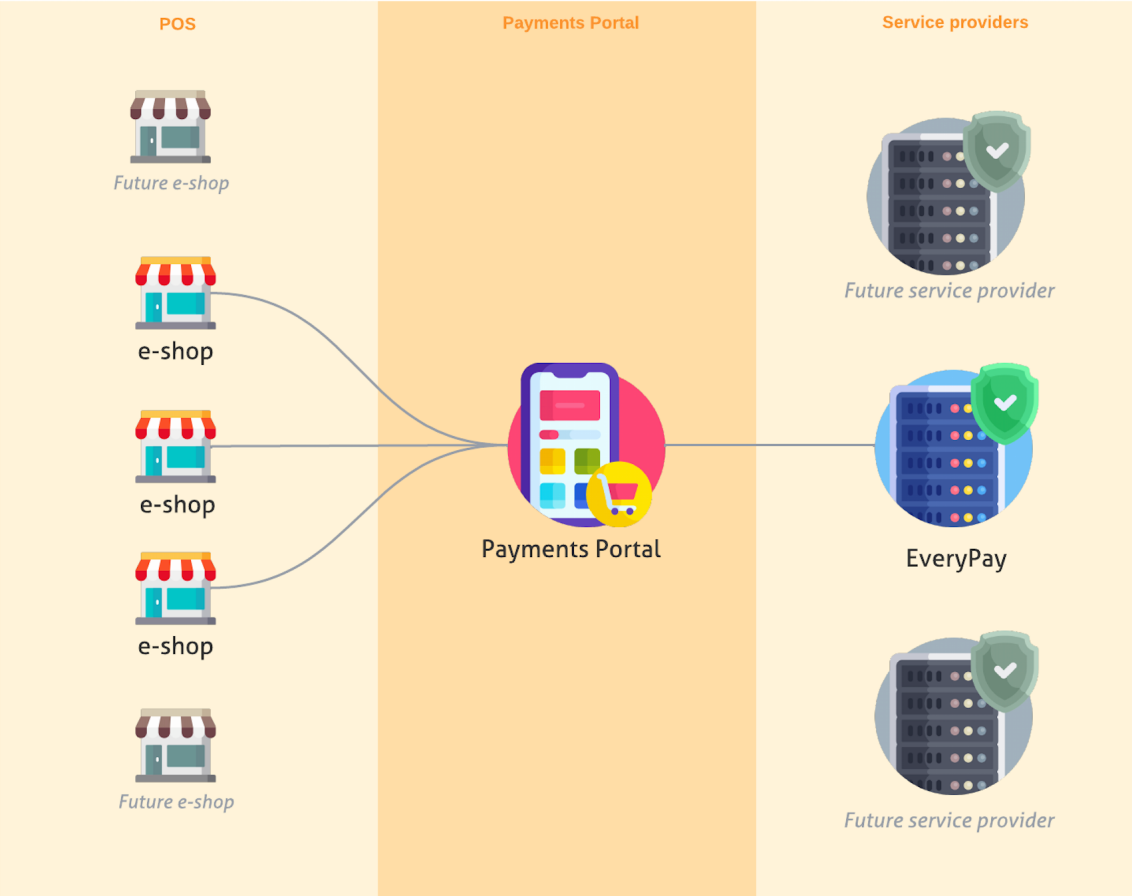
Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Lõputöö eesmärk on luua keskne lahendus e-kaubanduse makselahenduste juurutamiseks ja haldamiseks ühistranspordi piletimüügisüsteemidega tegelevale ettevõttele.

Lõputöö raames arendatava süsteemi tellija on Ridango AS. Erinevates Ridango hallatavates ühistranspordi piletimüügi e-poodides on makselahendused lahendatud e-poe põhiselt. See teeb maksmisega seotud koodibaasi hooldamise kulukamaks, makseviiside seadistamise keerulisemaks (sealhulgas makseteenuse pakkuja vahetamise) ning eksisteeriva lahendusega ei saa ettevõte pakkuda eraldiseisvat makselahendust oma klientidele. Samuti on töö eesmärgiks e-kaubanduse maksetega seonduv tuua Ridango maksemeeskonna vastutusalasse vähendades sellega administratiivset keerukust organisatsioonis.



Joonis 1 Arendatava süsteemi kontseptuaalne eskiis.

2 Analüüs

Käesolevas peatükis kirjeldab autor töö skoopi ja projekti analüüsiga seotud tegevusi. Analüüsi läbiviimiseks kasutati Confluence grupitöötarkvara keskkonda, sest see on varasemalt juba kasutusele võetud Ridango AS poolt sisemiste dokumentide, juhendite ja artiklite loomiseks ning hoiustamiseks [1]. Confluence'i oluline omadus on selle ühilduvus Jira tööde organiseerimise tarkvaraga, mis asub samas Atlassian'i tarkvarade ökosüsteemis [2].

2.1 Töö skoop

Töö skoobis loodav süsteem hakkab pakkuma Ridango AS e-poe ja klientrakendustele e-kommerts maksete võimalust EveryPay ühekordse makse näitel. Eesmärk on võimalikult kiiresti jõuda tarkvaraarendusega e-poodide ja makseteenuseosutaja liidestumisteni ning seeläbi vahetult valideerida lahenduse toimimist ning vajadusel ümberkujundamist. Töös on kirjeldatud arendusmetoodika, analüüs ja töö teostus. Loodava tarkvaralahenduse tarnimine, automaattestide loomine ja administreerimine ei olnud antud töö skoobis. Järjestusskeem süsteemi toimimisest on lisades (vt Lisa 4)

2.2 Äridomeeni taust

Autori põhjendusel on loodava e-kaubanduse makselahenduste portaali loomisel kõige esimeseks liidestatavaks makseteenuseks EveryPay, sest selle teenuse liidestusega on võimalik pakkuda klientidele deebet- ja krediitkaardimakse võimalusi Visa ja Mastercard'i võrkudes ning samuti *Open Banking* liidestusstandardil põhinevat „avatud panganduse" makset [3], millele pani aluse Euroopa Parlamendis 2015. aastal jõustunud makseteenuseosutajate direktiiv (PSD2) [4]. Kuna EveryPay pakub teenust vaid läbi kolmandate ettevõtete, siis Ridango ärilistel kaalutlustel kasutatakse töö käigus EveryPay platvormil põhinevat SEB panga e-äri teenust [5].

2.3 Nõuete kogumine

Nõuete kaardistamine on üks olulisemaid töid igas tarkvaraarendusprojektis – see on vundamendiks kogu projektile ning sellest suuresti sõltub projekti edukus või vastupidiselt selle ebaõnnestumine [6]. Autori arvates on projekti nõuete kogumise etapis hilisema segaduse vältimiseks tähtis kokku leppida kasutatavates andmevorminduse standardites keele, ajatemplite, valuutade jms osas. Nõuete kogumiseks uuris autor töö tellija arendussuuniseid ning viis läbi intervjuud maksemeeskonna tooteomanikuga ja veebimeeskonna juhtivarendajaga (vt Lisa 2).

2.3.1 Funktsionaalsed nõuded

Järgnevas loetelus on väljad toodud nimekiri funktsionaalsetest nõuetest, mis määravad ära, mida loodav süsteem peab olema võimeline tegema:

1. Makseteenuse kasutaja peab saama kasutada avatud panganduse ja kaardi makseviise.
2. Makseteenuse kasutajale peab olema näha tellimuse sisu ja maksumus makseprotsessi alustamise hetkel.
3. Makseviise peab olema võimalik seadistada liidestunud klientrakenduse põhiselt.
4. Kasutajaliides peab olema saadaval eesti ja inglise keeles.
5. Süsteem peab teavitama klientrakendust makseandmete muutuse korral.

2.3.2 Mittefunktsionaalsed nõuded

Järgnevas loetelus on välja toodud nimekiri mittefunktsionaalsetest nõuetest, mis kirjeldavad süsteemi teostusele esitatud täpsustusi standardite, konventsioonide jm näol.

1. Süsteem peab olema liidestatud EveryPay makseteenusepakkujaga.
2. Süsteemi klientrakenduse poolne osa ei tohi olla EveryPay liidese spetsiifiline.
3. Süsteem peab tulevikus võimaldama nii kliendi kui ka kaupmehe algatatud maksetüüpe.

4. Makseteenuse kasutajad peavad olema tuvastatavad seotud klientrakenduste vahel.
5. Süsteem peab tulevikus võimaldama erinevate makseteenusepakkujatega liidestumist.
6. Tellimuse ja maksete andmeid peab säilitama vähemalt 7 aastat.
7. Andmebaasis genereeritud identifikaatorid peavad olema ISO/IEC 9834-8 (128-bit uuid) standardile vastavad.
8. Käsitletavad keelekoodid peavad vastama ISO 639-1 (kahetähelisele) standardile.
9. Käsitletavad ajatemplid peavad olema esitatud ISO 8601 standardile vastavalt.
10. Käsitletavad rahavaluutade koodid peavad olema esitatud ISO 4217 standardile vastavalt.
11. Süsteem peab salvestama süsteemis toimuvaid sündmuseid.
12. Süsteem peab väljastama rakendusliidese dokumentatsiooni liidestavatele klientrakendustele.
13. Andmebaasi tabeli nimed peavad olema inglise keele ainsuse vormis, välja arvatud juhul, kui see on reserveeritud võtmesõna. Mitu-mitmele relatsioonide tabeli nimetamisel tuleb lähtuda äriloogilisest tähendusest, kuid selle puudumisel peavad olema seostavate tabelite nimed tähestikulises järjekorras.
14. Andmebaasi tabelitel peavad veerud olema loomise ja uuendamise ajatempliga.
15. Kasutajaliides peab olema arendatud Angular raamistikul.
16. Kasutajaliides peab olema ekraanilaiusele kohanduv.
17. Kasutajaliides peab olema vastav WCAG nõuetele.
18. Serverrakendus peab olema arendatud Spring Boot raamistikul.
19. Andmebaas peab kasutama PostgreSQL andmebaasisüsteemi.

2.4 Süsteemiarhitektuur

Käesolevat äridomeeni probleemi on võimalik lahendada väga palju erinevate lähenemistega. Järgnevalt toob autor välja alternatiivsed variandid ning põhjendab valitud lähenemisi täpsemalt. Käesoleva töö süsteemi disainimisel lähtus autor Ridango vajadustest, nõuetest ja tänapäevase veebitarkvara arenduse praktikatest ning otsustas hajusa klient-server arhitektuuri kasuks.

Hajussüsteemi tunnuseks on iseseisvalt arendatud komponendid, mis on omavahel ühendatud arvutivõrguga. Seejuures erinevad komponendid on võimelised asuma erinevates arvutisüsteemides – näiteks kasutaja seadmes töötav brauserirakendus ning serveris töötav serverrakendus. Süsteemi osade üksteisest eraldamine hajussüsteemina teeb süsteemi kogu elutsükli vältel paindlikumaks muutuvate tarkvarale seotud nõuete suhtes – sobides hästi agiilse paradigmaga. See teeb soodsamaks vajaduse tekkimisel süsteemi rakendusi eraldiseisvalt välja vahetada ning lisaks arendada välja eraldiseisvaid seadmepõhiseid rakendusi ja vajadusel pakkuda serverrakenduse API liidestust teistele osapooltele [7].

Võimalikuks alternatiivseks lahenduseks on MVC ehk *Model-View-Controller* arhitektuur, mis jagab veebirakenduse kolmeks peamiseks loogiliseks osaks. *Model* osa vastutab andmetega seotud loogika eest, *View* osas on lahendatud kasutajaliidese vaadete loogika ning *Controller* osa ühendab kaht eelnimetatud osa. Võrreldes hajusate rakendustega, on monoliitne MVC oma olemuse tõttu nõuete muutustele vähem paindlik. Mõlemal juhul on aga andmebaas eraldiseisev süsteemi osa [8].

Kasutajaliidese rakendus on juurutatud *Single-Page Application* (SPA) meetodil [9], kus veebirakendus laetakse kasutaja veebibrauserisse üheainsa dokumendina ning sisu muutmiseks tehtavad andmepäringud sooritatakse taustal XMLHttpRequest (XHR) päringutega [10].

Üheks alternatiivseks variandiks on kasutajaliides lahendada *View* osana MVC arhitektuuris. Vaadete vahel navigeerimisest tulenevate võrgupäringute laadimisest kasutajakogemuse langust ei esineks, sest käesolevas töös käsitletav kasutajaliides ei nõua navigeerimist erinevate vaadete vahel. Võrreldes eelmainitud meetodiga on SPA üheks nõrgaks küljeks otsingumootorile optimeerimine (SEO), kuid antud juhul ei ole see

oluline argument, sest käesoleva süsteemi pakutavat teenust ei ole võimalik avalikult kasutada [11].

Serverrakendus põhineb *Representational State Transfer* (REST) arhitektuuristiilil. REST arhitektuuri tunnusteks on universaalne liidestatus, klient-server arhitektuur, olekusõltumatus, vahemällu salvestatus ja kihiline arhitektuur [12]. Alternatiivne võimalus on kasutada Facebook'i poolt 2012. a loodud GraphQL päringukeelt [13], kus võrreldes REST'iga toimuvad päringud kõik ühe otspunkti suunas, andes kaasa vaid spetsiaalse süntaksiga üles ehitatud sõne ning mille peale server tagastab vajaminevad ressursid JSON vormingus. Autori arvates sobib selline lähenemine suuremate rakendustele, nagu selleks on Facebook, rohkem ning antud töös käsitleva süsteemile oma lisakeerukuse ja -sõltuvuse tõttu sobib oluliselt vähem [14].

Relatsiooniline andmebaas on andmebaasi tüüp, mis salvestab andmeid ning nende seoseid teineteise vahel. Relatsioonilised andmebaasid põhinevad relatsioonilisel mudelil – intuitiivsel ning selgel viisil hoides andmeid tabelites [15].

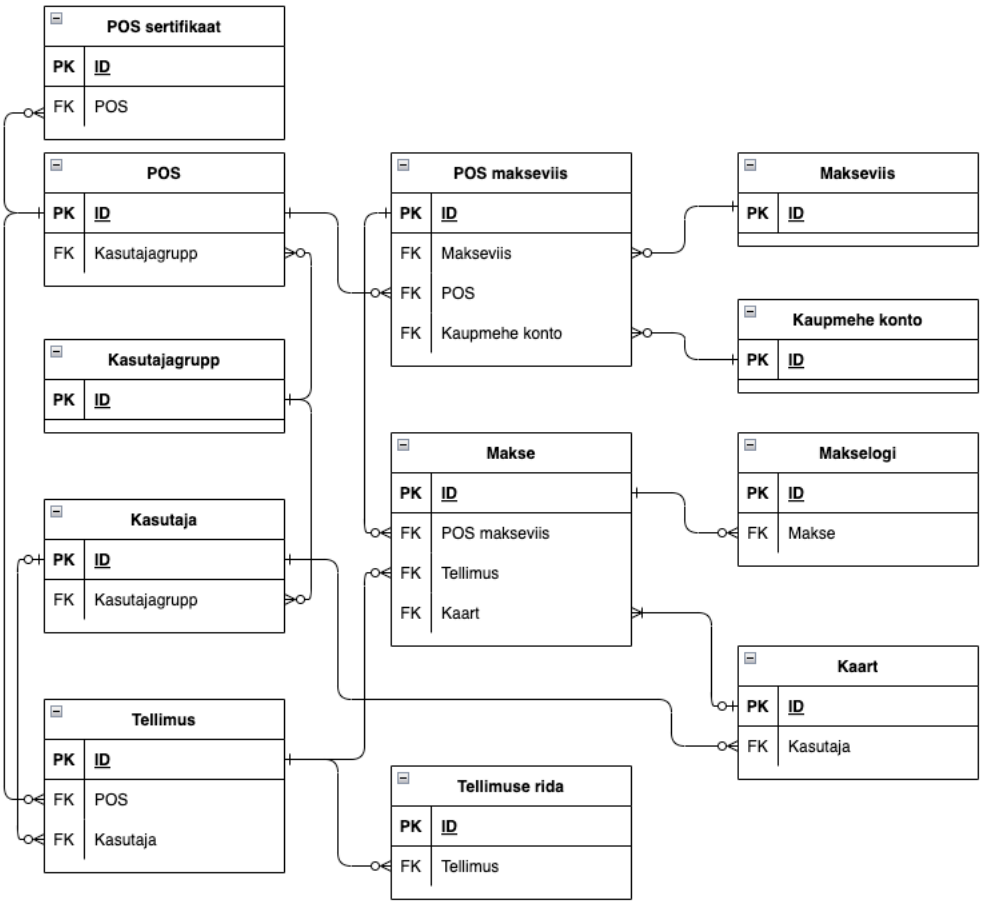
Relatsioonilisele andmebaasile üheks alternatiiviks on mitte-relatsiooniline ehk NoSQL andmebaasisüsteem. NoSQL andmebaasil on dünaamiline andmemudel, mis võrreldes SQL andmebaasiga teeb selle oluliselt paindlikumaks muutustele, sest andmeid ei organiseerita tabelites, vaid neid talletatakse dokumentidena. Antud töös käsitletava süsteemi andmemudel ei muutu kardinaalselt ning süsteemi üheks tähtsaimaks funktsionaalsuseks on makseviiside kombineerimine läbi seoste loomise, seega on SQL andmebaas autori arvates sobivaim valik [16].

Andmebaasisüsteemi valimisel lähtus autor sellest, mis toetab ja sobitub kõige paremini andmemudeliga ning seeläbi süsteemi toimimist, hooldamist ja edasiarendamist pikema perioodi vältel ning suuresti Ridango poolsest arendusvahendite suunistest.

2.5 Andmemudel

Töö autor on andmemudeli loonud toetamaks funktsionaalseid ja mittefunktsionaalseid nõudeid. Andmemudelis välja toodud kaartidega ja logimisega seonduvad olemid on antud töö skoobist väljas, kuid neid oli tarvilik tulevaste arenduste tõttu arvestada ning andmemudelile kanda. Andmemudelit kirjeldav olemi-suhte diagramm ning seda täpsustav teave on lisades (vt Lisa 5). Olemite seoste kiire ülevaate jaoks on autor

koostanud lihtsustatud olemi-suhte diagrammi (vt Joonis 2) ja kirjeldanud nende definitsioonid tabelis (vt Tabel 1).



Joonis 2 Lihtsustatud olemi-suhte diagramm.

Olemi nimi	**Definitsioon**
Kaupmehe konto	Liidestuslepingu sõlminud kaupmehe konto andmed, sh liidestustüüp ja rekvisiidid
Makseviis	Kõik süsteemi toetatud makseviisid
POS makseviis	POS-ile seadistatud makseviisid ning neid pakkuva kaupmehe kontod
POS	POS-i seaded
POS sertifikaat	POS-i väljastatud sertifikaadid allkirja kinnitamiseks
Kasutajagrupp	Kasutajagrupid
Kasutaja	POS kasutajad
Tellimus	Tellimuse andmed

Olemi nimi	Definitsioon
Tellimuse rida	Tellimuse read
Makse	Makseandmed
Kaart	Kaardiandmed
Makselogi	Maksetöötluse võrgupäringute logid

Tabel 1 Lihtsustatud olemite definitsioonid.

2.6 Arendusmetoodika

Agiilsele tarkvaraarendusele on omane struktuurne ja iteratiivne lähenemine projektihaldamisele ning tarkvaraarendamisele. Agiilse lähenemise eelised traditsioonilise arendusmetoodika ees on pidev tagasiside arendatavale tootele, mis maandab projekti käigus määramatusest tulenevaid riske ning pakub iseorganiseeruvatele meeskondadele kindlat metoodikat muutustega toime tulemiseks [17].

Arendusmetoodikaks valis töö autor agiilse Scrum raamistiku, et maandada varakult riske ja samuti planeerida töö funktsionaalseid osasid kindla perioodiga sprintidena ning arendusmeeskonnal on selle metoodikaga kõige rohkem kogemust [18].

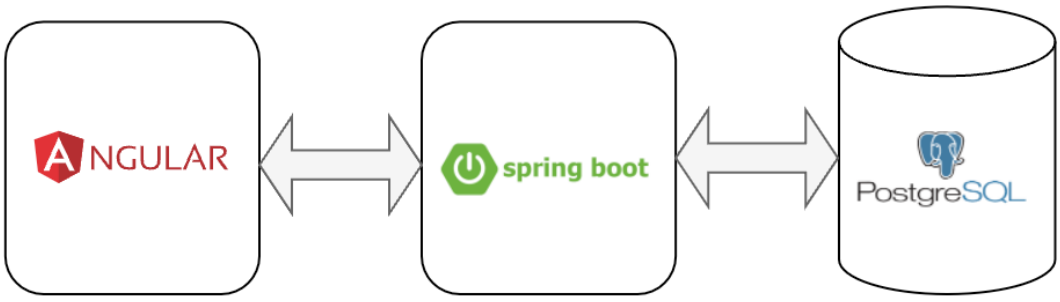
Arendusmeeskonna koosseisus oli tooteomanik*, scrum-master*, kaks tarkvaraarendajat ja testija. Sprindid olid 2-nädalase kestvusega ning koosnesid *daily stand-up, grooming, planning, demo, retrospective* koosolekutest, mida juhtis *scrum-master.* Enne sprindiga alustamist viidi läbi *planning* koosolek, et planeerida töid, millele oli hinnangud seatud ning seati sprindile eesmärk. Iga tööpäevaste *daily stand-up* koosolekutel astusid üles kõik meeskonnaliikmed ja arutati eelmisel tööpäeval toimunust, millised takistused tekkisid ja mida kavatsetakse selle päeva jooksul ette võtta. Sprindi keskel toimus *grooming* koosolek, milles jagati *product backlog'is* kirjeldatud töödele numbrilised keerukuse hinnangud *Fibonacci* jada esimese viie numbri piires. Sprindi lõpus toimus *demo* koosolek, kus esitleti tooteomanikule tehtud töö tulemusi. Järgnes *retrospective* koosolek, kus ühiselt vaadati tagasi toimunud sprindile ja toodi välja positiivseid ja negatiivseid esinenud olukordi ning kas sprindi eesmärk täideti. Negatiivsetele olukordadele loodi tegevuspunktid, et neid kõrvaldada ning selle mõju kontrolliti järgneval sprindil sama koosoleku ajal [19].

Scrum'i juhtimise jaoks sai valitud Atlassian'i Jira keskkond, sest see on nii Ridango poolt kui ka üldiselt tarkvaraarenduse valdkonnas laialdaselt kasutusel ehk arendusmeeskond on sellega tuttav ja see ühildub väga hästi Confluence keskkonnaga. Jira võimaldab projekti töö jaotada eraldi *issue*'deks ehk tööülesanneteks ning paigutada need virtuaalsel tahvlil eraldi tulpadesse: *To do, In progress, Code review, Ready for testing, In testing, Done.*

Töö skoobist oli võimalik tuletada kaks eepost, mille alla kuuluvad erinevad kasutajalood, mis omakorda jagunesid eraldi tehtavateks tööülesanneteks. Eeposed ja kasutajalood ja tööülesanded on lisades (vt Lisa 3)

2.7 Arendusvahendite valik

Arendusvahendite valik on kitsendatud Ridango poolsetest suunistest aktsepteeritud tehnoloogiate osas, mida saab ning mis on sobivad töös käsitletava süsteemi loomiseks. Süsteemi arendamiseks on kasutatud Angular, Spring Boot ja PostgreSQL tehnoloogiaid (vt Joonis 3). Lähtekoodi versioonihalduseks on kasutatud Git versioonihaldustarkvara [20] ja hoidlana on kasutatud Atlassian'i loodud keskkonda Bitbucket. Autori arvates võimaldab Bitbucket teiste Atlassian'i teenustega väga head integratsiooni ning peale lähtekoodi hoidmise samuti automatiseerida integreerimist, testimist, tarneprotsessi jpm [21].



Joonis 3 Arendusvahendite arhitektuurne paiknemine.

2.7.1 Kasutajaliidese rakendus

Angular on avatud lähtekoodiga skaleeritavate veebirakenduste ehitamise komponendipõhine raamistik, mis põhineb Typescript'i programmeerimiskeelel ja mille arendust juhib Google. Angular pakub raamistikusiseselt laia valikut teeke

navigeerimiseks, sisestusvormide halduseks, klient-server ühenduste loomiseks ja palju muud, mida veebirakenduse loomisel võib vaja minna [22].

Angular on ehitatud Typescript programmeerimiskeele baasil. Typescript on loodud populaarse Javascript'i keele süntaksile staatilise programmeerimiskeele elementide lisamise teel, tänu millele on toetatud selles kõik, mis on toetatud Javascript'is. Seejuures on tüüpide lisamine vabatahtlik – andes arendusele kohati lisaefektiivsust arenduskiiruse näol. Typescript teeb programmi kirjutamise veakindlamaks ja mugavamaks [23].

Stiilide efektiivsemaks kirjutamiseks on kasutatud CSS-i eeltöötlemise laiendit Sass. See võimaldab arendajal kirjeldada stiile oluliselt lühema koodiga, tehes koodi haldamise lihtsamaks. Sass toetab ka luua funktsioone, tsükleid ja muid programmeerimise võtteid, kuid autori arvates on selle suurim eelis siiski kirjutatava lähtekoodi mahu optimeerimine. Sass'is kirjutatud kood protsessitakse standardseks CSS koodiks [24].

Ridango on varasemalt välja töötanud monorepositooriumi, mis võimaldab kiiresti uusi rakendusi ning teeke välja töötada. Selleks kasutatakse Nx koodiehitamise tarkvara, mis toetab väga hästi monorepositooriumi lähenemist [25]. Rakenduste loomisel on võimalik ühiseid komponente ja mooduleid jagada teiste rakendustega ja seeläbi tõhustada arendusmeeskondade tööd väärtuse loomisel. Miinuseks monorepositooriumi puhul on koodimuudatuste mõju haldamine ning keerukuse suurenemine, kuid negatiivset mõju on võimalik vähendada automaattestidega ning Nx laadsete tehnoloogiatega, mis optimeerivad suure koodibaasi ehituse, testimise ja tarnimise protsesse [26].

Alternatiivne kasutajaliidese tehnoloogia on Reactjs (lühidalt React). Kui Angular on Typescript'il põhinev raamistik (kogum teekidest), siis React on Javascript'i teek, mis suurimate unikaalsustena kasutab HTML esitamiseks JSX süntaksit ning *Virtual DOM* lähenemist [27]. React on juhitud Facebooki poolt ning see on arenduskommuunides väga populaarne. Võrreldes Angular'iga on sisseehitatud funktsionaalsust vähem, paindlikum erinevatele lahenduskäikudele ja seega on selle tehnoloogia õppimiskurv kohati laugem. Autori arvates on React oma paindlikkuse tõttu heaks alternatiiviks Angular'ile, kuid samas peab mõnevõrra rohkem programmeerima ja kasutama väliseid teeke, et saavutada sama lõpptulemus [28].

2.7.2 Serverrakendus

Spring Boot on Java programmeerimiskeelel põhinev tehnoloogia, mis teeb võimalikult lihtsaks luua eraldiseisvaid toodanguvalmidusega Spring'i raamistikul põhinevaid rakendusi [29]. Spring Boot on laiendus Spring'i raamistikule, seejuures elimineerides vajaduse täpsema konfiguratsiooni järele ning seeläbi suurendades arenduskiirust. Spring pakub põhjalikku infrastruktuuri Java rakenduste arendamiseks nagu näiteks sisse ehitatud *JDBC, MVC, Security* ja *AOP* moodulitega [30].

Alternatiivne serverrakenduse tehnoloogia on Javascript'i programmeerimiskeelel põhinev NestJS raamistik, mis on ehitatud ning toetab täielikult Typescript'i ning töötab NodeJS käituskeskkonnas. NestJS kombineerib elemente objektorienteeritud-, funktsionaalsest ja funktsionaalsest reaktiivsest programmeerimisest. Sisemiselt kasutab see populaarset Express raamistikku ning teeb arendajale vajaduse tekkimisel otseselt saadavaks selle API. Filosoofilisel tasandil püüavad selle tehnoloogia arendajat serveripõhistele Javascript'i rakendustele pakkuda arhitektuurilist struktuuri [31].

Autoril on kogemus mõlema raamistikuga. Tema arvates on need raamistikud kontseptuaalsel tasemel vaadatuna üpris sarnased ning paralleele on võimalik välja tuua mitmeid, kuid peamiselt programmeerimiskeelte erinevuste tõttu lahknevad siiski paljus. Ühisosana on näiteks mõlemas raamistikus laialdaselt kasutatud *Dependecy Injection* ja *Decorator* tarkvara disainimustreid ja palju muid ühiseid kontseptsioone [32][33].

2.7.3 Andmebaas

Andmebaasimootorina kasutatakse PostgreSQL relatsioonilist andmebaasisüsteemi. PostgreSQL on võimas avatud lähtekoodiga relatsiooniline andmebaasisüsteem, mis kasutab ja laiendab SQL päringukeelt, et andmeid turvaliselt salvestada ja on skaleeritav ka kõige keerukamate jõudlust nõudvate lahenduste tarbeks. PostgreSQL juured ulatuvad 1986. aastasse, kui seda projekti alustati *University of California at Berkeley* ülikoolis ning selle põhiplatvormi on arendatud rohkem kui 30 aastat [34].

Vabavaralistest relatsioonilistest andmebaasisüsteemidest on üks alternatiiv MariaDB. See on üks kõige populaarsemaid andmebaasisüsteeme, mille on algselt loonud MySQL arendajad ning on vaikimisi enamustes Linux'i distributsioonides. Seda kasutavad näiteks sellised ettevõtted nagu Alibaba Cloud, Microsoft ja Intel [35]. Autori arvates on PostgreSQL ja MariaDB, mõnda erandit välja arvates, väga sarnased ning on mõlemad

sobilikud antud ülesandele. Peamiselt Ridango arendusvahendite suunistest tingituna otsustas autor andmebaasimootori valikus PostgreSQL tarkvara kasuks.

3 Töökäik

Käesolevas peatükis kirjeldab autor tehnilist teostust kasutajaliidese rakenduse ja serverrakenduse arendamisel ning toob välja tema arvates huvitavamad nüansid.

3.1 Kasutajaliidese rakenduse arendamine

Angular'i rakenduse ülesseadmine monorepositooriumi põhimõttel koodihoidlas oli autori arvates Nx käsureatööriista poolt väga tõhus. Pärast Sass stiililehe eelprotsessori ja navigeerimisemooduli lisavalikute valimist koostas tööriist rakenduse algse kaustastruktuuri. Seejärel oli vajalik monorepositooriumist iseärasustest tulenevalt seadistada rakenduse konfiguratsioon, et lähtekoodis oleks võimalik vahetult viidata ühiselt kasutatavatele teekidele. Seejärel seadis autor üles rakenduse lähtekoodi tarneahela.

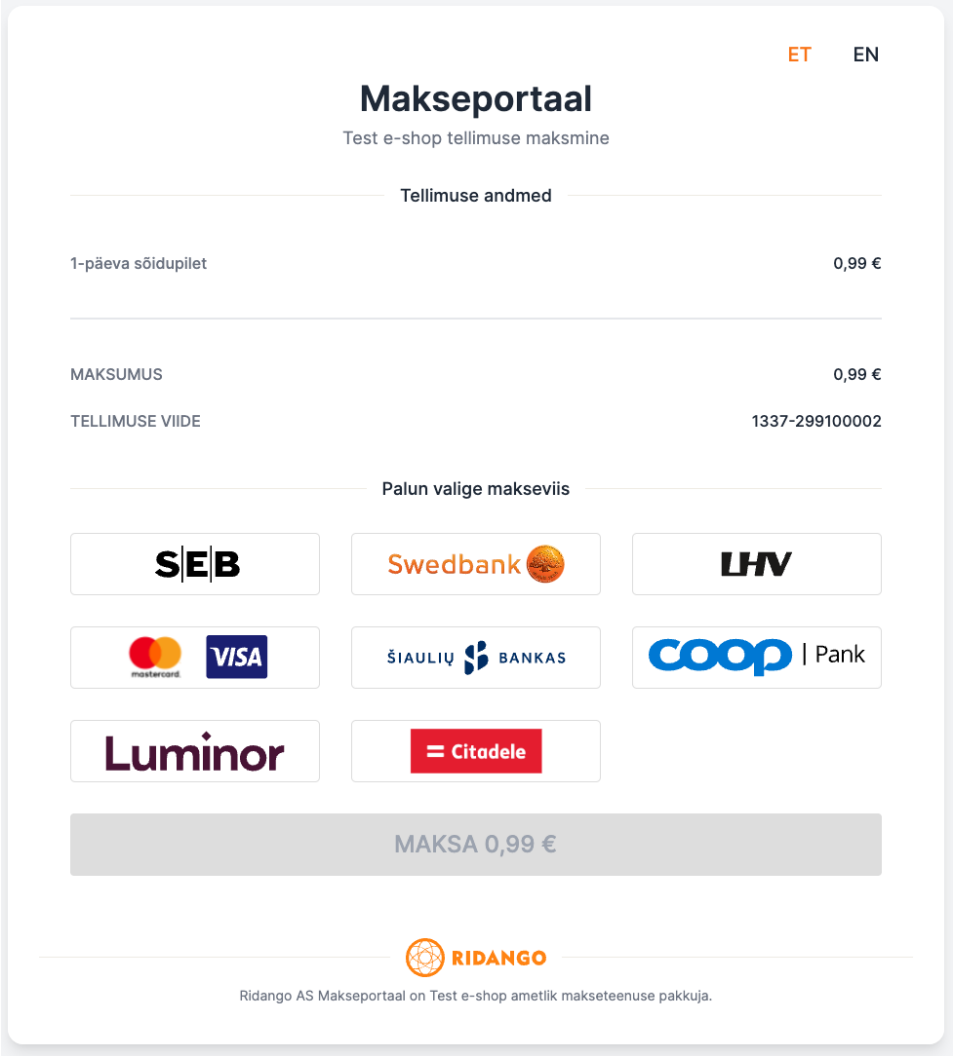
3.1.1 Makse alustamise vaade

Kõige esimene kasutajaliidese vaade, mida kasutaja näeb on makse alustamise ehk *checkout* vaade. Autori loodud tööülesande kirjeldus on antud töö lisades (vt Lisa 6). POS-i süsteemist jõuab kasutaja siia vaatesse spetsiaalsele aadressile suunamise teel, mille põhjal on võimalik tuvastada varasemalt süsteemi edastatud tellimus. Vaate ülesandeks on välja kuvada makse suurus, valuuta, makseviiside valiku ning olemasolul ka tellimuse sisu.

Kujunduse osas on oluline välja tuua, et see peab kuvama mistahes levinud ekraanilaiustel – sealhulgas kitsamate mobiilseadmete ekraanidel. Seda arvestades programmeeris autor vaate HTML elementide paigutuse ekraanilaiusele kohanduvaks (vt Lisa 7). Makseviiside valikud ja „maksa" nupp asuvad standardse HTML vormielemendi sees ning makseviisid käituvad raadionupu põhimõttel, et suurendada WCAG kooskõlastust. Samal põhjusel on nuppudena juurutatud keelevalik ning kõikidele võimalikele elementidele väärtustatud *aria-label* atribuudid.

Vaate laadimisel tuli aadressilt lahendada POS-ile seadistatud esmane keelevalik ning pakkuda kasutajale võimalust seda muuta eesti ja inglise keele vahel. Makseviiside valiku kuvamiseks oli tarvis pärida serverrakenduse API-st POS-ile saadaolevad makseviisid. Tellimuse andmed, sealhulgas POS teenuse nimi ning saadaolevad makseviisid, päriti

samast serverrakenduse API otspunktist, andes päringule kaasa tellimuse ID. Päring teostati Angular'i raamistikku sisseehitatud *HttpClient* võrgupäringute teenuse kaudu [36].



Joonis 4 Makse alustamise vaade.

3.1.2 Maksmiselt tulemise vaade

Kasutaja naasmisel makseteenuse osutaja juurest kuvatakse kasutajale maksmiselt tulemise vaade ning kasutaja suunatakse kohe ümber POS-i poolt tellimuse edastamisel defineeritud aadressile. Vaate eesmärk on töödelda ümber makseteenuseosutaja poolt tulnud URL-i parameetrid standardsele kujule – hoidudes sellega edastamast makseteenuseosutaja iseärasusi.

3.2 Serverrakenduse arendamine

Serverrakenduse ülesseadmiseks kasutas autor Spring Initializr veebirakendust, mis võimaldab kiirelt ja mugavalt kõik sobilikud eelistused ja väliste teekide konfiguratsiooni kokku panna ning alla laadida [37]. Lähtekoodi kirjutamiseks kasutas autor Intellij IDEA integreeritud arenduskeskkonna tarkvara [38]. IDEA's projekti ülesseadmine hõlmas keskkonnamuutujate lisamist, andmebaasiga ühendamiseks vajaminevate rekvisiitide seadistamist, olemihalduri seadistamist Hibernate ORM-iga [39] ning turvaseadete sättimist klient-server lahendusele vastavaks – näiteks, et erinevatelt domeenidelt oleks võimalik serverrakendusele võrgupäringuid sooritada (CORS). Ülesseadmine hõlmas ka andmemudeli juurutust JPA olemite defineerimise näol ning algandmete sisestusefaili *data.sql* koostamist toetamaks edasiseid arendustegevusi (vt Lisa 8).

3.2.1 Tellimuste haldamine

Tellimuste haldamine hõlmab tellimuse salvestamist, ümbersuunamise lingi koostamist ja tellimuse andmete väljastamist.

Tellimuse esitab POS süsteem, tehes HTTP POST päringu serverrakenduse API suunas. Selle peale salvestatakse tellimuse andmed süsteemi andmebaasi. Kui kasutaja on oma ostukorvi muutnud, siis peab olema võimalik POS-il kasutaja ja tellimuse viite ID järgi eelnevalt saadetud tellimus uuendada mistahes palju arv kordi. Seejuures iga kord kontrollitakse, et ühtegi maksekirjet ei ole uuendatava tellimusega seostatud, mis viitaks sellele, et kasutaja on juba alustanud maksmist ning tellimust ei ole lubatud enam uuendada. Kui ühtegi maksekirjet ei ole seostatud, siis olemasolev tellimus kustutatakse ja luuakse uus tellimuse kirje. Selle eelis seisneb selles, et väline süsteem ei pea mälus hoidma, kas tegemist on uue või juba edastatud tellimusega ning tellimuse ID muutumisel tunnistatakse kehtetuks ka varasemalt väljastatud ümbersuunamise link. Tellimuse esitamise päringu sisu valideeritakse annotatsioonidega (vt Joonis 5) ning seejärel töödeldakse seda edasi tellimuse teenuses. Töötlemise käigus luuakse või uuendatakse tellimuse kirje ning kasutaja identifikaatori esitamisel luuakse või leitakse olemasolev kasutajakirje andmebaasist ning seostatakse tellimuse kirjega. Kuna süsteem toetab POS-ide vahelisi ühiseid kasutajaid, siis kasutaja kirjet otsitakse POS-iga seostatud kasutajagrupi ning tellimusel oleva kasutaja identifikaatori järgi. Autori arvates on märkimisväärne nentida, et kasutaja identifikaator on välise süsteemi poolt edastatud ja selle unikaalsust ei ole võimalik kontrollida, seega seda ei ole mõistlik kasutada

relatsioonide loomisel. Päringule vastatakse saadud andmetega, et välisel süsteemil oleks tagasiside andmete õigsuse kohta ning ümbersuunamise lingiga, mis võimaldab kasutajal maksmisele suunduda. Link sisaldab POS-i seadistatud keelevalikut ning tellimuse ID-d, millega on võimalik maksmise alustamise vaatel vajaminevate andmete päring teostada.

```java
@Data
public class CreateOrderRequest {
    @NotBlank
    @NotNull
    private String orderReference;

    @NotBlank
    private String customerId;

    @NotNull
    @PositiveOrZero
    @Max(Integer.MAX_VALUE)
    private int amount;

    @NotBlank
    @NotNull
    private String customerUrl;

    @Valid
    @NotNull
    private List<OrderItem> orderItems;
}
```

Joonis 5 Tellimuse loomise DTO klass validatsiooni annotatsioonidega.

Tellimuse detailandmete päringu teenindamiseks vajab tellimuste teenus vaid tellimuse ID-d. Tagastatakse tellimuse detailandmed ning POS-ile seadistatud saadaolevad makseviisid. Autori arvates on selline lahendus tulevikus mõistes paindlik, sest seda päringut saab hüpoteetiliselt teostada ka POS ise, juhul kui soovitakse vaid API liidestust ning ise kuvada kasutajale saadaolevaid makseviise.

3.2.2 Tellimuse autentimine

Liidestunud POS autenditakse tellimuse päringu allkirja põhjal, kinnitades selle õigsust varasemalt süsteemile esitatud sertifikaadist lahendatud avaliku võtmega. Autori loodud tööülesanne on käesoleva töö lisades (vt Lisa 9). Sertifikaadi esitamisel süsteemile tagastatakse POS-ile vastav ID, mida peab päringule kaasa andma, et süsteem leiaks andmebaasist vastava sertifikaadi ning seeläbi kirjega seostatud POS-i kirje. Autori arvates on selline lahendus mõistlik, sest erinevalt salasõna väljastamisest on allkirjastamiseks vajamineva privaatvõtme konfidentsiaalsena hoidmine liidestunud süsteemide vastutusalas.

Päring koosneb *header* (kodeerimata päis) väljast, milles on esitatud allkirja kinnitamiseks vajaliku sertifikaadi ID. Lisaks sellele on päringus *protected* (kodeeritud päis), *payload* (sisu) ja *signature* (allkiri). Viimased kolm välja on esitatud *Base64* kodeeringus (vt Joonis 6).

Joonis 6 Allkirjastatud tellimuse esitamise päringu sisu

Pärast päringu väljade valideerimist otsitakse võtme ID järgi andmebaasist sellele vastavat sertifikaati ning päringu allkirja kontrollitakse sealt välja lahendatud avaliku võtmega *SignatureVerificationService* klassis (vt Joonis 7). Allkirja kehtivuse korral on päringu esitanud POS tuvastatud ja süsteem töötleb tellimuse esitamise päringut edasi tavapärasel viisil.

```
@Service
@RequiredArgsConstructor
public class SignatureVerificationService {
    private static final String CERTIFICATE_TYPE_X_509 = "X.509";

    public void verifySignature(SignedRequest body, String certPlainText)
            throws SignatureException, MalformedJwtException, ExpiredJwtException, UnsupportedJwtException, CertificateException {
        var jwt : String = String.join( delimiter: ".", body.getProtectedHeader(), body.getPayload(), body.getSignature());

        if (!Jwts.parser().isSigned(jwt)) {
            throw new UnauthorizedException("Request is not signed");
        }
        X509Certificate certificate = parseCertificate(certPlainText);
        Jwts.parser()
                .setSigningKey(certificate.getPublicKey())
                .parse(jwt);
    }

    private X509Certificate parseCertificate(String certificate) throws CertificateException, IllegalArgumentException {
        var certificateFactory : CertificateFactory = CertificateFactory.getInstance(CERTIFICATE_TYPE_X_509);
        var certStream = new ByteArrayInputStream(
                Base64.getMimeDecoder().decode(certificate)
        );
        return (X509Certificate) certificateFactory.generateCertificate(certStream);
    }
}
```

Joonis 7 Allkirja kehtivuse kontrollimine.

3.2.3 Maksete haldamine

Maksete haldamine hõlmab maksetöötluse alustamist, päringute logimist, POS-ide teavitamist ning makseandmete väljastamist.

Maksetöötluse alustamine on käivitatud päringu peale, mille algatab süsteemi kasutajaliidese rakendus pärast kasutajapoolset makseviisi valimist ning kinnitamist „maksa“ nupule vajutamisega. Päringu sisus on tellimuse- ja makseviisi ID, mille põhjal on võimalik makseprotsessi alustada. Süsteem loob esialgse maksekirje, millega efektiivselt keelatakse ära tellimuse edasine uuendamine, sest tellimusega on seostatud maksekirje. Edasi annab süsteem maksetöötluse üle makseviisiga seotud liidestusteenusele.

Kõik HTTP päringud, mida süsteem teostab makseteenuseosutaja (PSP) poole ning mis tulevad makseteenuse osutaja suunast logitakse maksete logimise teenuse kaudu andmebaasi, et oleks hiljem näha päringutes sisalduvaid andmeid töötlemata kujul ning vajaduse tekkimisel nendega edasist töötlust teha.

Makseandmete, eelkõige staatuse, muutuse peale teeb süsteem automaatselt HTTP GET päringu POS-ile seadistatud *callback* aadressile, teavitades sellega andmete uuenemisest. Selle jaoks on autor loonud POS-i teavituste teenuse, mis on PSP-st liidestusest sõltumatu, kuid mida võivad välja kutsuda erinevate PSP-de teavitusi töötlevad teenused.

3.2.4 Makseteenuse liidestamine

Kui tellimuste ja maksetega seonduv osa süsteemist on makseteenuseosutajast sõltumatu, siis antud töös on makseteenuse liidestamine otseselt seotud EveryPay API-ga. Töö skoobist tulenevalt keskendub autor vaid ühekordse makse tüübile. Autori loodud tööülesande kirjeldus on käesoleva töö lisades (vt Lisa 10).

EveryPay teenuse kutsub välja makseteenus makse alustamise hetkel valides sobiva liidestusteenuse juurutuse *PaymentServiceProviderFactory.find* meetodi kaudu, andes argumendiks makseviisiga seotud kaupmehe konto tüübi *MerchantAccountType*. Liidestusteenuste kandidaate otsib raamistiku *Dependency Injection* lahendus rakenduse-

üleselt konstruktormeetodi parameetrites täpsustatud *PaymentServiceProvider* liidest (*interface*) implementeerivate klasside seast (vt Joonis 8).

```
@Component
public class PaymentServiceProviderFactory {

    private EnumMap<MerchantAccountType, PaymentServiceProvider> services;

    @Autowired
    public PaymentServiceProviderFactory(Set<PaymentServiceProvider> paymentServiceProviders) {
        register(paymentServiceProviders);
    }

    public PaymentServiceProvider find(MerchantAccountType accountType) throws ServiceUnavailableException {
        if (!services.containsKey(accountType)) {
            throw new ServiceUnavailableException("Payment service provider integration not found for " + accountType);
        }
        return services.get(accountType);
    }

    private void register(Set<PaymentServiceProvider> paymentServiceProviders) {
        services = new EnumMap<>(MerchantAccountType.class);
        paymentServiceProviders.forEach(
                service -> services.put(service.getMerchantAccountType(), service));
    }
}
```

Joonis 8 Automaatne integratsiooniteenuste kaardistamine ja valimine.

EveryPay liidestusteenuse leidmisel koostab *EveryPayService.pay* meetod (vt Joonis 9) päringu sisu makse algatamise jaoks. EveryPay poolt genereeritakse igale saadaolevale makseviisile aadressid ning süsteem otsib kasutaja poolt valitud makseviisile vastet päringust saadud makseviiside seast. See omakorda vormindatakse ümber PSP agnostiliseks väljundiks ümbersuunamise URL-i näol.

```
public String pay(PaymentEntity payment) {
    PosEntity pos = payment.getPosPaymentMethod().getPos();
    MerchantAccountEntity merchantAccount = payment.getPosPaymentMethod().getMerchantAccount();
    EveryPayCredentials credentials = new EveryPayCredentials(merchantAccount.getCredentials());

    ZonedDateTime now = ZonedDateTime.now();
    String customerUrl = composeReturnUri(payment);

    EveryPayOneOffRequest payload = EveryPayOneOffRequest.builder()
            .apiUsername(credentials.getApiUsername())
            .accountName(merchantAccount.getAccountName())
            .amount(MoneyUtil.centsToDecimal(payment.getInitialAmount()))
            .customerUrl(customerUrl)
            .orderReference(payment.getOrder().getId().toString())
            .nonce(UUID.randomUUID().toString())
            .timestamp(now.toString())
            .locale(pos.getPreferredLanguage())
            .build();

    if (payment.getOrder().getCustomer() != null) {
        payload.setRequestToken(true);
        payload.setTokenAgreement("unscheduled");
    }

    log.info("Initiating EveryPay one-off payment: paymentId={}", payment.getId());

    var response : EveryPayOneOffResponse = everyPayAPIService.initiateOneOff(payment, payload);

    updatePayment(payment, response);

    return parsePaymentServiceLink(response, payment);
}
```

Joonis 9 EveryPay teenuse *pay* meetod.

EveryPay aktsepteerib makse summana kahekohalise täpsusega ujukoma arvu [40], kuid autori poolt arendatav süsteem käsitleb summasid vaid täisarvudega (sentides). Rahaga ümberkäivate süsteemide puhul on kriitiliselt oluline, et ujukoma ja täisarvude vahelistel teisendustel ei juhtu ümardusviga, sest lihtsalt 100-ga korrutamine ja jagamine ei taga järjepidevaid tulemusi. Autor on loonud selle probleemi vältimiseks eraldi *MoneyUtil* klassi, mis pakub staatilisi meetodeid mõlema variandi vahel konverteerimisel, kasutades selleks ära Java *BigDecimal* klassi *movePointLeft* ja *movePointRight* meetodeid (vt Joonis 10).

```java
public class MoneyUtil {
    public static int decimalToCents(double amount) {
        return (BigDecimal.valueOf(amount)).movePointRight(2).intValue();
    }

    public static double centsToDecimal(int amount) {
        return (BigDecimal.valueOf(amount)).movePointLeft(2).doubleValue();
    }
}
```

Joonis 10 Arvude teisendamine.

Makseteenuseosutaja (PSP) saadab maksetöötluse kohta teavitusi, kuna kasutaja viibimise kestvus makseteenuses on teadmata. PSP poolsete teavituste jaoks võtab süsteem vastu HTTP GET päringuid PSP poolt ning töötleb seda edasi ning vajadusel teostatakse selle sündmuse peale päring PSP suunas makseandmete uuendamiseks. Lisaks sellele saadab süsteem omakorda teavituse välja liidestunud POS-i süsteemile. Teavituse vastuvõtmiseks on tarvilik, et serverrakenduse otspunkt oleks üle interneti leitav ning selle tarbeks võttis autor kasutusele Ngrok puhverserveri teenuse [41]. Pärast Ngroki paigaldamist ja seadistamist võimaldab see *reverese proxy* meetodil [42] kohalikus masina pordile edasi suunata avalikke aadresseid ning seeläbi vastu võtta EveryPay saadetavaid *callback* päringuid.

4 Hinnang tulemusele

Töö teostuse käigus juurutatud lahendused täidavad kõiki funktsionaalseid nõudeid. Tellimuste esitamisega on võimaldatud täpsustada tellimuse ridu, et neid maksetöötluse alustamise hetkel välja kuvada. Maksetöötluse alustamise vaade on saadaval eesti- ja inglise keeles. SEB e-äri teenuse EveryPay platvormi liidestusega on võimaldatud kasutajal valida pangakonto ja -kaardi maksete vahel. Andmemudel on kujundatud sellistelt, et makseviise on võimalik klientrakenduse põhiselt seadistada. Süsteem saadab automaatselt teavitusi liidestunud klientrakendust makseandmete muutusest.

Autori hinnangul vastab loodud lahendus tänapäeva tarkvaraarenduse parimatele praktikatele ja paindlik nii ärilistele kui ka tehnoloogilistele muutustele.

Autori arvates oli töö kulg tänu põhjalikule analüüsifaasile ootuspärane, kuid meeskonnas oli korraga arenduses mitu tarkvaraprojekti ja aja- ning inimressurssi nappuse tõttu oli kõik antud töös käsitletavad ülesanded lahendatud autori poolt isiklikult – välja arvatud meeskonna QA inseneri teostatud ülesannete vastuvõtutestimine.

Ootamatustest toob autor välja algselt konservatiivseks skoobiks hinnanud ülesande kujunemist oodatust ambitsioonikamaks väljakutseks, sest süsteemi disain peab suutma vastu pidada ka tulevikuarendustele ning mille vältimine, keskendudes vaid töö skoobis kirjeldatule, oleks hiljem majanduslikus mõttes negatiivseks osutunud.

Tulevikuarendusteks on autori sõnul planeeritud EveryPay liidestuse täiendused korduv- ja tagasimaksete osas, järgmiste makseteenuseosutajate liidestamine ja süsteemi administreerimine kasutajaliidese abil.

5 Kokkuvõte

Töö eesmärk luua Ridango AS-ile keskne e-kaubanduse makselahenduste portaal sai ülesseatud tingimustel täidetud ja loodud süsteem toetab edasiarendusi ning teenindavatel klientrakendustel on võimalik alustada liidestumise arendamisega. Eelmainitule tuginedes saab autor väita, et eesmärkide täitmine oli edukas.

Töö käigus sai kujundatud nõuetele vastav süsteem ning juurutatud Angular'i, Spring Boot'i rakendused. Analüüsi etapis tutvus autor äridomeeniga, kogus kokku nõuded, planeeris tööde teostamist Scrum projektijuhtimise raamistikus, kujundas andmemudeli ning võrdles süsteemiarhitektuuri lähenemisi ja võimalikke arendusvahendeid.

Äriline kasu hakkab ajapikku kogunema e-kaubandusmaksete haldamisega seonduvate hooldustööde osas ning teenustasudega seotud finantsiliste võitudega, mis saavutatakse tänu erinevate makseteenuseosutajate makseviiside kombineerimisega. Lisandväärtusena ettevõtte portfelli suhtes on võimalik vahendada klientidele makseteenust ilma piletimüügisüsteemi arendamiseta.

Kasutatud kirjandus

[1] Confluence Overview [WWW] https://www.atlassian.com/software/confluence/guides/get-started/confluence-overview (01.04.2022)

[2] Jira software [WWW] https://www.atlassian.com/software/jira (01.04.2022)

[3] EveryPay Merchant Features [WWW] https://every-pay.com/merchant-features (02.04.2022)

[4] Everything you need to know about PSD2 [WWW] https://www.bbva.com/en/everything-need-know-psd2/ (15.04.2022)

[5] EveryPay Pricing [WWW] https://support.every-pay.com/merchant-support/what-is-the-pricing-for-using-everypays-payment-gateway-platform-for-digital-payments-acceptance/ (15.04.2022)

[6] Why software requirements development matters [WWW] https://www.cio.com/article/241106/why-software-requirements-development-matters.html (01.04.2022)

[7] Client/Server Architecture [WWW] https://www.techopedia.com/definition/438/clientserver-architecture (01.04.2022)

[8] MDN Web Docs - MVC [WWW] https://developer.mozilla.org/en-US/docs/Glossary/MVC (02.04.2022)

[9] MDN Web Docs – SPA [WWW] https://developer.mozilla.org/en-US/docs/Glossary/SPA (02.04.2022)

[10] MDN Web Docs – XMLHttpRequest [WWW] https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest (02.04.2022)

[11] Single Page Application (SPA) vs Multi Page Application (MPA) – Two Development Approaches [WWW] https://asperbrothers.com/blog/spa-vs-mpa/ (03.04.2022)

[12] The REST Architecture [WWW] https://www.baeldung.com/cs/rest-architecture (05.04.2022)

[13] GraphQL: A data query language [WWW] https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/ (03.04.2022)

[14] GraphQL Vs. REST APIs [WWW] https://graphcms.com/blog/graphql-vs-rest-apis (03.04.2022)

[15] What is a Relational Database (RDBMS)? [WWW] https://www.oracle.com/database/what-is-a-relational-database/ (05.04.2022)

[16] SQL vs NoSQL: 5 Critical Differences [WWW] https://www.integrate.io/blog/the-sql-vs-nosql-difference/ (05.04.2022)

[17] What is Scrum? [WWW] https://www.scrum.org/resources/what-is-scrum (06.04.2022)

[18] Kanban vs. Scrum: which agile are you? [WWW] https://www.atlassian.com/agile/kanban/kanban-vs-scrum (06.04.2022)

[19] The 2020 Scrum Guide [WWW] https://scrumguides.org/scrum-guide.html (06.04.2022)

[20] Git About [WWW] https://git-scm.com/about (07.04.2022)

[21] Built for professional teams [WWW] https://bitbucket.org/product (07.04.2022)

[22] What is Angular? [WWW] https://angular.io/guide/what-is-angular (07.04.2022)

[23] TypeScript is JavaScript with syntax for types [WWW] https://www.typescriptlang.org/ (07.04.2022)

[24] CSS with superpowers [WWW] https://sass-lang.com/ (07.04.2022)

[25] Intro to Nx [WWW] https://nx.dev/getting-started/intro (07.04.2022)

[26] Monorepos in Git [WWW] https://www.atlassian.com/git/tutorials/monorepos (07.04.2022)

[27] Virtual DOM and Internals [WWW] https://reactjs.org/docs/faq-internals.html (07.04.2022)

[28] Angular vs React 2022: Which JS Framework your Project Requires [WWW] https://www.simform.com/blog/angular-vs-react/ (07.04.2022)

[29] Spring Boot Overview [WWW] https://spring.io/projects/spring-boot (07.04.2022)

[30] A Comparison Between Spring and Spring Boot [WWW] https://www.baeldung.com/spring-vs-spring-boot (07.04.2022)

[31] NestJS Introduction [WWW] https://docs.nestjs.com/ (01.04.2022)

[32] Dependecy Injection [WWW] https://java-design-patterns.com/patterns/dependency-injection/ (01.04.2022)

[33] Decorator [WWW] https://refactoring.guru/design-patterns/decorator (01.04.2022)

[34] What is PostgreSQL? [WWW] https://www.postgresql.org/about/ (01.04.2022)

[35] New to MariaDB Server? [WWW] https://mariadb.org/ (01.04.2022)

[36] Communicating with backend services using HTTP [WWW] https://angular.io/guide/http (01.04.2022)

[37] Spring initializr [WWW] https://start.spring.io/ (02.04.2022)

[38] Intellij IDEA [WWW] https://www.jetbrains.com/idea/ (02.04.2022)

[39] JPA Provider [WWW] https://hibernate.org/orm/ (10.04.2022)

[40] EveryPay APIv4 Integration Document [WWW] https://support.every-pay.com/downloads/everypay_apiv4_integration_documentation.pdf (18.04.2022)

[41] Ngrok [WWW] https://ngrok.com/ (15.04.2022)

[42] What is a reverse proxy? [WWW] https://www.cloudflare.com/en-gb/learning/cdn/glossary/reverse-proxy/ (15.04.2022)

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

Mina, Raimo Johanson

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „E-kaubanduse makselahenduste portaali arendamine Ridango AS-i näitel“, mille juhendaja on Meelis Antoi.
   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.

15.04.2022

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 – Intervjuu nõuete kaardistamiseks

Intervjuu Ridango transpordimaksete meeskonna tooteomanikuga

1. Millise makseteenuse pakkujaga on vaja liidestada?
   Esimesena on vaja liidestada SEB e-äri teenusega, mis kasutab EveryPay platvormi.

2. Kas tulevikus on plaanis rohkem liidestusi makseteenusepakkujatega?
   Jah, tulevikus on plaanis rohkemate makseteenusepakkujatega liidestuda nagu näiteks Adyen, Sveapay, Nets jne.

3. Milliseid makseviise peab süsteem võimaldama?
   Kõike, mida EveryPay pakub, aga milliseid makseviise teatud e-pood näeb, peab olema võimalik e-poe põhiselt seadistada.

4. Mida on kindlasti vaja maksmise alustamise hetkel kasutajale kuvada?
   Lisaks maksmisele kuuluva summa peab kasutajale olema kuvatud ka tellimuse read.

5. Kas ühel klientrakendusel võib olla mitu lepingut makseteenuse pakkujatega?
   Jah, selline olukord võib tekkida küll, kui näiteks Nets teeb kaardimaksetele parema hinna kui SEB e-äri. Siis sellisel juhul peab olema võimalik makseviise liidestuslepingute põhjal kombineerida äriliselt parima võimalike variantide vahel.

6. Millistes keeltes peab kasutajaliides olema saadaval?
   Kasutajaliides peab olema saadaval eesti ja inglise keeles, kuid uute keelte lisamine peab olema lihtne.

7. Milliseid maksetüüpe on tulevikus plaanis arendada?
   Kindlasti tuleks arvestada kliendi poolt algatatud ja kaupmehe poolt algatatud kaardimaksetega. Selle jaoks on vaja kaardi token hilisemaks kasutamiseks salvestada.

8. Kui kaua peab tellimuste ja maksmistega seotud andmeid säilitama?
   Raamatupidamise seadusest tulenevalt seitse aastat.

Intervjuu Ridango veebitiimi meeskonna juhtivarendajaga

1. Kuidas on korraldatud tänases lahenduses makseandmete vahetus teenusepakkuja ja kaupmehe vahel?
   Üldiselt on andmevahetuse jaoks on kasutusel *callback* päringud.
   Teenusepakkuja süsteem teeb makse referentsiga argumenteeritud päringu meie süsteemi suunas, mille peale meie süsteem teeb detailandmete päringu saamiseks omakorda päringu nende suunas.

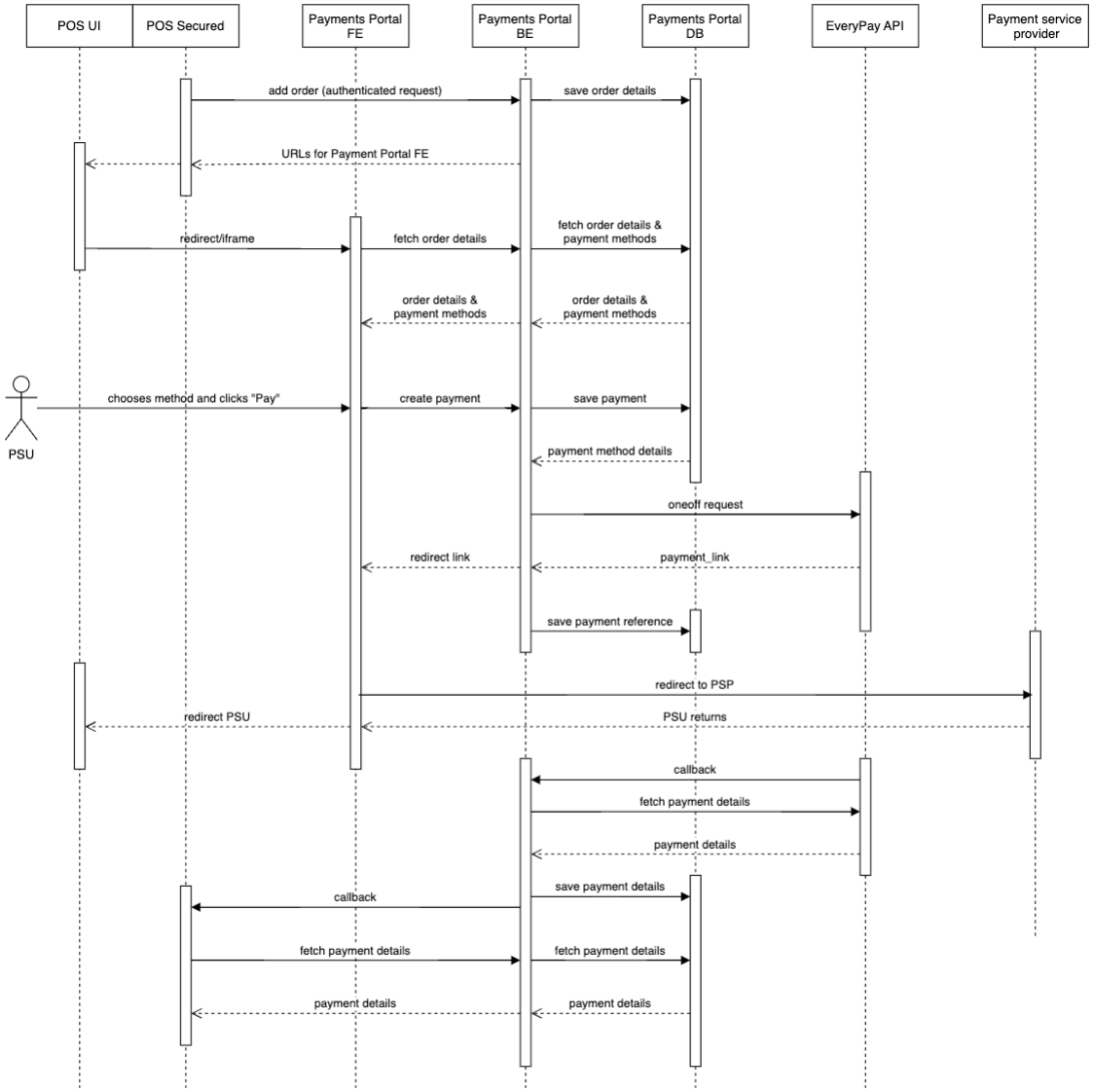
2. Kuidas seotakse kaardi tokeniseerimisel kasutaja tema pangakaardiga?
   Seda on võimalik teha vaid autenditud kasutajakontoga.

3. Kas erinevatel piletimüügi e-poodidel on ühiseid kasutajaid või kasutajagruppe?
   Jah, näiteks ühe Tallinna piletimüügikeskkonna kasutaja saab sama kasutajaga Tartu piletimüügikeskkonnast piletit osta.

Lisa 3 – Scrum ülesehitus

Epic: Setting up Payments Portal
As PO, I want only registered e-shops to be able to submit orders, so that there are no unauthorised records BE: Setup initial database BE: Secure order receiving endpoint for only allowed apps
As POS, I want to submit order details, so that they can be showed to the PSU later BE: Create endpoint for receiving orders from e-shops
As POS, I want to get a checkout link, so that I can hand off the payment process BE: Create endpoint for returning checkout links
As a PSU, I want to select the payment method, so that I can choose my preferred option. FE: Create route for checkout BE: Create endpoint providing order details and payment methods
As a PSU, I want to see order details, so that I know the order is correct. BE: Create endpoint providing payment details and e-shop return URL
Epic: EveryPay one-off payment
As a PO, I want to integrate to EveryPay, so that I can provide payment options BE: Implement EveryPay one-off payment integration
As a POS, I want to know the payment status, so that I know how to process the order. BE: Create endpoint for receiving callbacks from EveryPay BE: Notify e-shops from status change
As a PSU, I want to know the payment status, so that I know whether it was successful. FE: Create route for returning from PSP BE: Create endpoint to provide payment details

Lisa 4 – Süsteemi järjestusskeem



1. POS süsteem teeb turvalise päringu kood tellimuse andmetega süsteemi (Ridango-sisene nimetus *Payments Portal* – edaspidi PP) suunas. POS võib teha seda mitmeid kordi, et uuendada tellimuse andmeid.

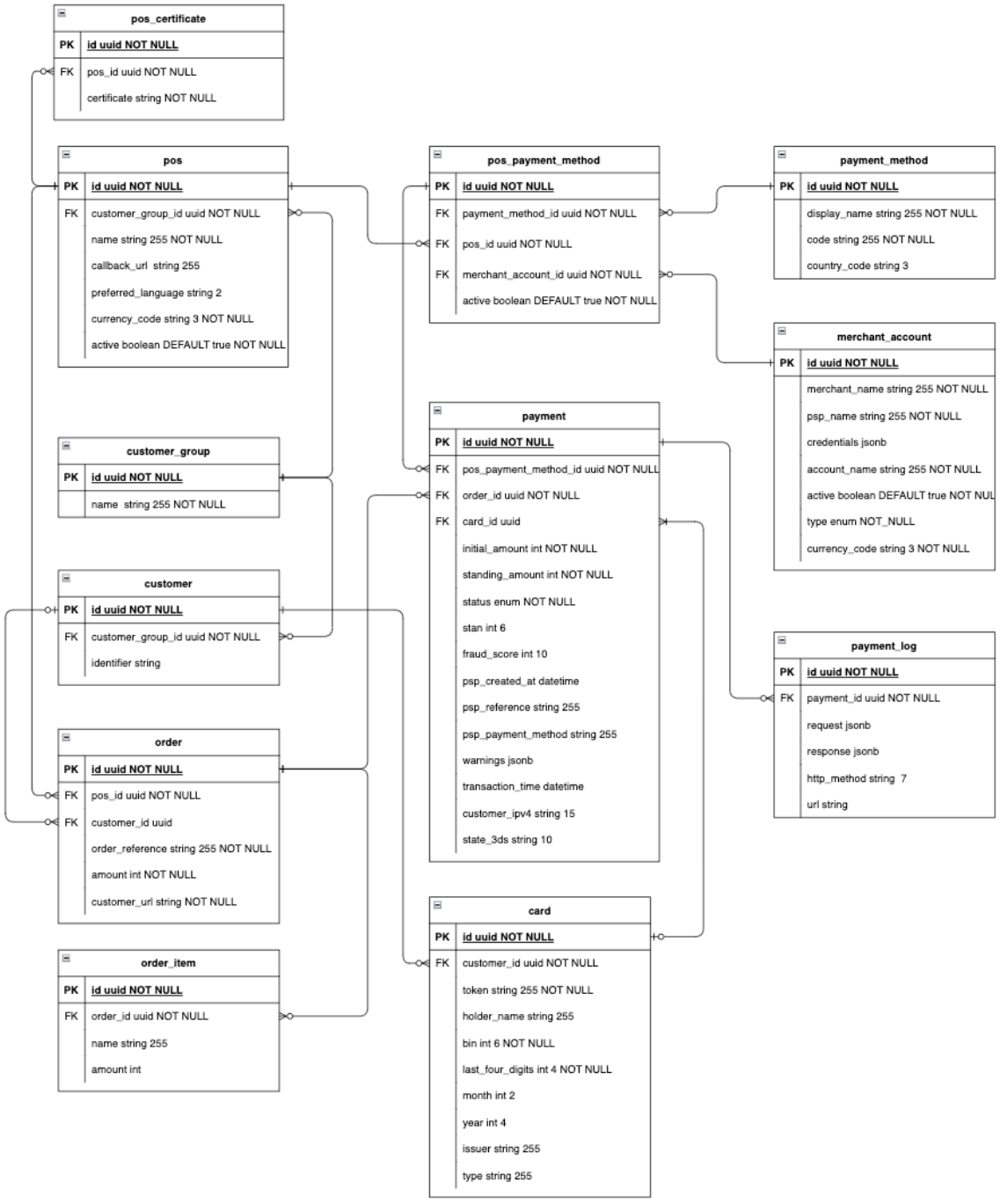
2. PP salvestab tellimuse andmed andmebaasi ja tagastab saadud tellimuse ning ümbersuunamise URL-i.

3. POS suunab makseteenuse kasutaja (PSU) edasi PP *checkout* vaatele.

4. PP pärib tellimuse andmed ja võimalikud makseviisid ning kuvab need välja.

5. PSU valib sobiva makseviisi ning vajutab „Maksa“ nupule.
6. PP loob makse kirje andmebaasi ning sõltuvalt makseviisist valib liidese (antud töös EveryPay).
7. PP teeb päringu EveryPay API-le ning suunab PSU ümber saadud makseteenuse URL-ile.
8. Maksmiselt naasemisel suunatakse kasutaja ümber PP makselt naasmise vaatesse ning PSU suunatakse kohe ümber POS-i peale.
9. Samal ajal eelmise punktiga oodatakse maksestaatuse muutuse *callback* päringut EveryPay poolt.
10. Maksestaatuse uuenemise korral päritakse makse detailid EveryPay API-lt ning salvestatakse need andmebaasi.
11. POS-i *callback* URL-iga teostatakse GET päring
12. POS pärib makse detailandmeid.

Lisa 5 – Olemi-suhte diagramm



merchant_account	Account data holding the contract
merchant_name	Merchant name. eg Ridango, Elron
psp_name	PSP name. eg SEB EveryPay
credentials	Credentials used with PSP API
account_name	Account name. eg EUR3D1
active	Whether or not the merchant account can be used with payment method

type	PSP integration type. eg EveryPay
currency_code	Merchant account currency. Has to match with POS currency.

payment_method	**Payment method data**
id	Unique identifier
display_name	Payment method name
code	Internal code for payment method
country_code	Country code for Open Banking method

pos_payment_method	**Payment method attached to POS**
id	Unique identifier
payment_method_id	Referenced payment method
pos_id	Referenced POS
merchant_account_id	Referenced merchant account
active	Whether or not this is currently available

pos	**POS (e-shop, client application) data**
id	Unique identifier
customer_group_id	Referenced customer group
name	POS display name
callback_url	URL where Payments Portal will perform notification request
preferred_language	Default preferred language that will be used in Payment Portal
currency_code	POS currency code. Is used for setting up payment method with appropriate merchant account
active	Whether or not this is currently available

pos_certificate	**POS certificates for authentication**
id	Unique identifier (is used as "kid")
pos_id	Referenced POS
certificate	Certificate provided by POS

customer_group	**Customer groups in order to allow POS to share users**
id	Unique identifier
name	Name of the group

customer	**Customer data**
id	Unique identifier
customer_group_id	Referenced customer group
identifier	External identifier provided by POS

order	**Order data**

id	Unique identifier. Is visible to PSU
pos_id	Referenced POS
customer_id	Referenced customer
order_reference	Order reference provided by POS
amount	Order amount
customer_url	POS URL where PSU is redirected after payment

order_item	**Order row data**
id	Unique identifier
order_id	Referenced order
name	Display name for order row
amount	Order row amount

payment	**Payment data**
id	Unique identifier. Is visible to PSU
pos_payment_method	Referenced POS payment method
order_id	Referenced order
card_id	Referenced card
initial_amount	Initial amount the payment record was created
standing_amount	Standing amount (after possible refund)
status	Payment status (enum)
stan	System Trace Audit Number
fraud_score	Fraud score provided by PSP
psp_created_at	Payment creation time of PSP
psp_reference	Payment reference of PSP
psp_payment_method	Payment method of PSP
warnings	Warnings from PSP
transaction_time	Payment transaction time from PSP
customer_ipv4	Customer IPv4
state_3ds	3DS state

card	**Payment card data**
id	Unique identifier
customer_id	Referenced customer
token	Token provided by PSP
holder_name	Card holder name
bin	Bank Identification Number
last_four_digits	Card PAN last four digits
month	Card expiration month
year	Card expiration year
issuer	Card issuer
type	Card type. Eg Visa, Master

payment_log	**Payment data log for network requests**
id	Unique identifier
payment_id	Referenced payment
request	Request body. Null for GET
response	Response body
http_method	Request http method
url	Request url

Lisa 6 – Maksmise alustamise vaate tööülesanne

Description

Checkout route for displaying the order details and payment methods.

Route: `/checkout/{order-id}?lang={language-code}`

1. Order ID is used to request order details from backend API.
    a. `GET {backend-api-url}/orders/{order-id}`
    b. Show loading state while requesting the details
    c. If an error occurs just display general error message in snackbar.
2. `lang` is optional. Default language is english (`en`)
3. Language should be changeable from UI - either by dropdown or links
4. Display order:
    a. amount
    b. items (if there are any)
5. Display available payment methods.
    a. Option behaves as radio button
6. Payment method logos, store them to FE assets:
    a. SEB https://igw-seb-demo.every-pay.com/assets/payment_methods/seb-0d4c46990b1c01e424f3dcd921a3fdb1f27686dcc999452f5bc95d0b7e12664a.svg
    b. SWEDBANK https://igw-seb-demo.every-pay.com/assets/payment_methods/swedbank-986d24a501d2ec2ac43a7b90daf2f766c731e5a759de813c8140fee87b804843.svg
    c. LHV https://igw-seb-demo.every-pay.com/assets/payment_methods/lhv-999be249bbe15a2e549a80d44a1df27b298b598a9d242365313fad77deb13a84.svg
    d. CITADELE https://igw-seb-demo.every-pay.com/assets/payment_methods/citadele-1b81cd63732949874f46f0bfd4c55609a36e32f670ca6f1d2518dc4801ea3d7f.svg
    e. COOP https://igw-seb-demo.every-pay.com/assets/payment_methods/coop-f1c55f2bdfc32f2689e98a0877d543a378b4e61a94b16bc26ed9d98495673ca2.svg
    f. SIAULIU https://igw-seb-demo.every-pay.com/assets/payment_methods/siauliu-80c8c2a452f384ae423765697988ab27fce8efd8684a1dd94922df0891aef20d.svg
    g. MASTERCARD/VISA https://igw-seb-demo.every-pay.com/assets/payment_methods/cards-9807ea71c428f0695fae455e78aec234fb6b9ef35bfb4b4f4722c205f0fed7ed.svg
    h. LUMINOR https://igw-seb-demo.every-pay.com/assets/payment_methods/luminor-0289d73a37e430a56aaaea8040c840fd89d2f1b22489fee0a29e7dda25879f4b.svg
7. Submit button labelled "Pay" → `POST {backend-api-url}/payments`
    a. Disabled until payment method is not selected
    b. Display loading state to button when it is clicked and request is pending
    c. Redirect to received redirect link
    d. If an error occurs: display it inside snackbar.

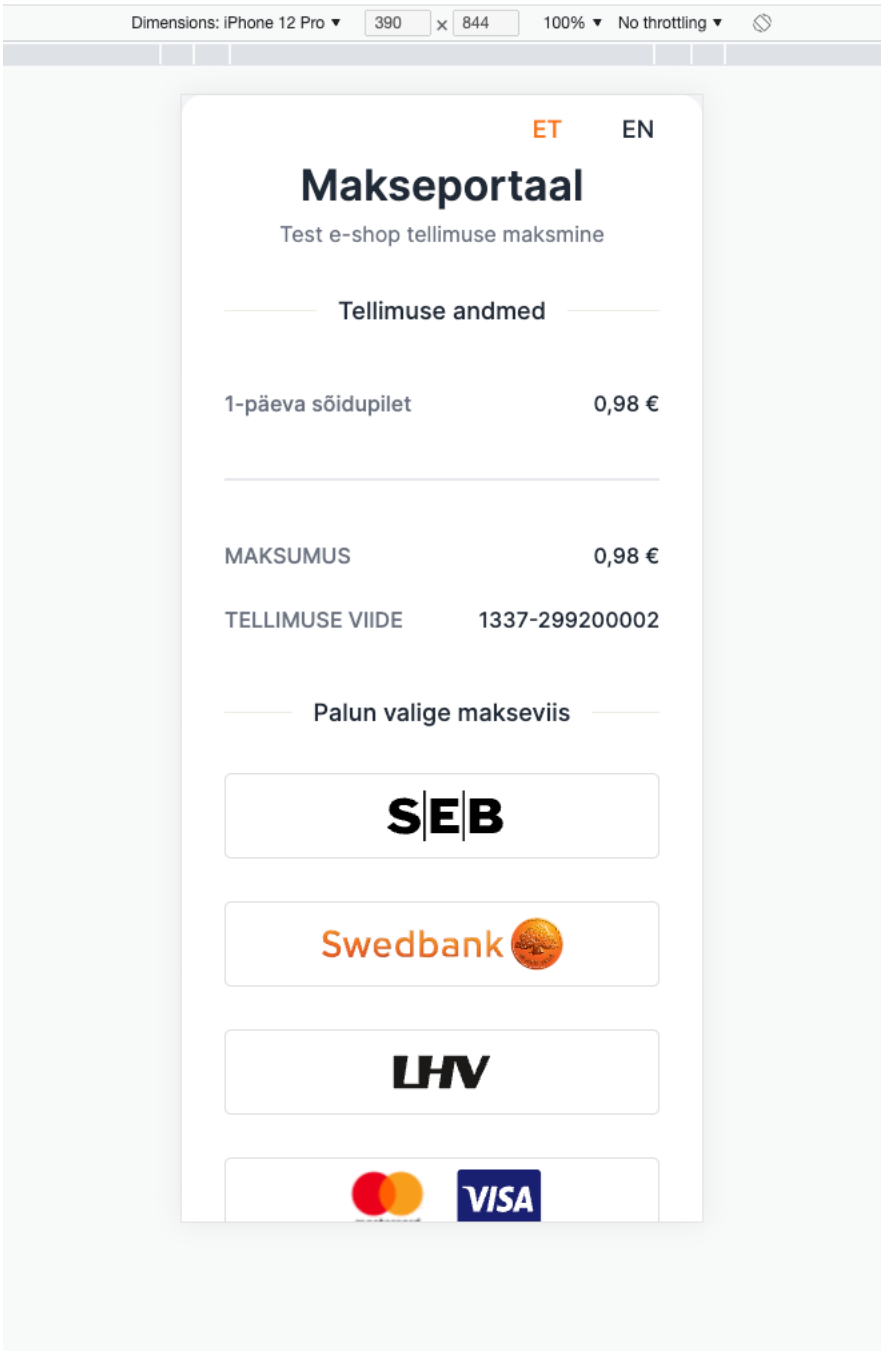
Prerequisites

1. PG-1651: FE: Setup frontend application IN APPROVAL must be done
2. PG-1654: BE: Create endpoint providing order details and payment methods TO DO must be done
3. PG-1653: BE: Create endpoint for payment initiation TO DO must be done

Acceptance criteria

1. Correct order details (including order items if there are any) are displayed
2. User can choose payment method from available methods
3. After successful submit user is redirected to link provided from service
4. User can change UI language
5. UI displays correctly on small (mobile) screens as well

Lisa 7 – Maksmise alustamise vaade kitsal ekraanil



Lisa 8 – Algandmete sisestuskäsud

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp" WITH SCHEMA public;

INSERT INTO merchant_account (id, merchant_name, psp_name,
credentials, account_name, type, currency_code)
VALUES ('1d039b98-90a5-4881-baa1-9d83bd403821',
        'Ridango',
        'SEB EveryPay TEST',
        '{
          "api_username": "secret",
          "api_password": "secret",
          "api_gateway": "https://igw-demo.every-pay.com/api/v4"
        }',
        'EUR3D1',
        'EVERYPAY',
        'EUR');

INSERT INTO payment_method (id, display_name, code, country_code)
VALUES (public.uuid_generate_v4(),'AS SEB BANK', 'seb_ob_ee', 'EE'),
       (public.uuid_generate_v4(),'SEB Banka AS', 'seb_ob_lv', 'LV'),
       (public.uuid_generate_v4(),'AB SEB Bankas', 'seb_ob_lt', 'LT'),
       (public.uuid_generate_v4(),'SWEDBANK AS', 'swed_ob_ee', 'EE'),
       (public.uuid_generate_v4(),'SWEDBANK AS', 'swed_ob_lv', 'LV'),
       (public.uuid_generate_v4(),'SWEDBANK AB', 'swed_ob_lt', 'LT'),
       (public.uuid_generate_v4(), 'LHV AS', 'lhv_ob_ee', 'EE'),
       (public.uuid_generate_v4(), 'Visa/MasterCard', 'visa_mc',
null),
       (public.uuid_generate_v4(), 'Šiaulių Bankas AB', 'sb_ob_lt',
'LT'),
       (public.uuid_generate_v4(), 'Coop Pank AS', 'coop_ob_ee',
'EE'),
       (public.uuid_generate_v4(), 'Luminor Bank AS', 'luminor_ob_ee',
'EE'),
       (public.uuid_generate_v4(), 'Luminor Bank AS', 'luminor_ob_lv',
'LV'),
       (public.uuid_generate_v4(), 'Luminor Bank AS', 'luminor_ob_lt',
'LT'),
       (public.uuid_generate_v4(), 'Citadele Banka', 'citadele_ob_ee',
'EE'),
       (public.uuid_generate_v4(), 'Citadele Banka', 'citadele_ob_lv',
'LV'),
       (public.uuid_generate_v4(), 'Citadele Banka', 'citadele_ob_lt',
'LT');

INSERT INTO customer_group (id, name)
VALUES (public.uuid_generate_v4(), 'Test group 1');

INSERT INTO pos (id, name, customer_group_id, callback_url,
preferred_language, currency_code)
SELECT '7b847fa0-ac80-4c29-a99e-034384f1653a',
       'Test e-shop',
       id,
       'www.example.com',
       'en',
       'EUR'
FROM customer_group
LIMIT 1;
```

```
INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'seb_ob_ee';

INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'swed_ob_ee';

INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'lhv_ob_ee';

INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'visa_mc';

INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'coop_ob_ee';

INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'citadele_ob_ee';

INSERT INTO pos_payment_method (id, pos_id, merchant_account_id,
payment_method_id)
select public.uuid_generate_v4(), a.id, b.id, c.id
from pos a
         cross join merchant_account b
         cross join payment_method c
where c.code = 'luminor_ob_ee';
```

Lisa 9 – Tellimuse autentimise tööülesanne

Description

Secure order receiving endpoint for only registered apps. Implement JWS for `POST orders` payloads.

1. Validate request body
   a. Return Bad Request (400) exception if any attribute is missing. Including `kid` in `header` object
2. Use `header.kid` to look for the public key in table `pos_certificate.id`
   a. Certificate column `pos_certificate.certificate`
   b. If key is not found then return status 404 - key not found
3. Verify signature (use BouncyCastle for example):
   a. *(base64 encoded)* `protected` *contains* `alg` *attribute*
   b. if signature is invalid return status 401
4. Find POS `pos_certificate.pos_id`
   a. If `pos.active` = `false` respond with http status 403
5. Parse request body payload from base64
6. Continue processing the request as usual → providing posId from `pos_certificate.pos_id`

Prerequisites

1. PG-1648: BE: Create endpoint for orders from e-shops IN TESTING must be done

Acceptance criteria

1. Requests with valid signature can successfully post order details referencing the actual e-shop
2. Requests with non-existent `header.kid` receives error code 400
3. Requests with Invalid signature receives error code 401
4. Requests for inactive POS with valid signature receives error code 403

Testing with default key and certificate

1. Use Online JWT tool to create signed request. Paste this token to the encoded token window to get things started.
2.
```
eyJhbGciOiJFUzI1NiJ9.eyJvcmRlclJlZmVyZW5jZSI6Ik8xMjk4NzMwMTIzODEyMyIsImN1c3RvbWVySWQiOiJDLTEyMzQ1Njc4MDIiLCJjdXN0b21lclVybCI6Imh0dHBzOi8
```
3. After pasteing you can change all the values in decoded payload part as needed.
4. Use this private key to sign the token.
5.
```
-----BEGIN PRIVATE KEY-----MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgJH30ePjD4lTheNAB7G/WVT/IVjBCqsbt+2JXtyw79tahRANCAAQk9Q4PyQVevl
```
6. Make sure the algorithm is set as ES256
7. Click on the arrow pointing left. This will sign the JWT.
8. The JWT consists of three parts: header, payload, signature
9. POST /orders request body will look like this:
10.
```
{
    "header": {
      "kid": "e9bc097a-ce51-4036-9562-d2ade882db22"
    },
    "protected": "eyJhbGciOiJFUzI1NiJ9",
    "payload": "eyJvcmRlclJlZmVyZW5jZSI6Ik8xMjk4NzMwMTIzODEyMyIsImN1c3RvbWVySWQiOiJDLTEyMzQ1Njc4MDIiLCJjdXN0b21lclVybCI6Imh0dHBzOi8vd3d3
    "signature": "bub_VwcqAcPHB4Dauo6-trvsZ8p6Sn_lgS41ghIgZVHA0x3vZSJ3XHFHcJtKuciJRGAvrtx4mG2gsPkLUTgD9w"
  }
```
11. The kid "e9bc097a-ce51-4036-9562-d2ade882db22" is pointing to the certificate already stored at Payments Portal database that was generated with the private key (step 5)
12. protected = base64 header, payload = base64 payload, signature = base64 signature (step 8)

Testing with other keys

1. Generate new private key using the algorithm RS256/ES256

Lisa 10 – EveryPay liidestamise tööülesanne

Description

This endpoint is called from Payments Portal FE application when user selects payment method and click "Pay". EveryPay v4 specification: SEB Pank (EE) e-commerce payment solution

Create payment `POST /payments` triggers payment initiation process.

1. Choose integration method depending on `pos_payment_methods.merchant_accounts.type`. In this case it will be EVERYPAY
2. Credentials for connecting to EveryPay are stored at `merchant_accounts.credentials` column
3. Details for oneoff payment request are found in SEB APIv4 integration document
    a. `customer_url` is `{payments-portal-fe-url}/payments/{payment-id}`
        i. `payment-id` being the newly created payment record id.
    b. `order_reference` is `orders.id` not `orders.order_reference` because `orders.order_reference` is external ID (from e-shop).
4. Parse EveryPay `api/v4/payments/oneoff` response
    a. Store payment data to `payment` entity.
5. Match the payment method selected by user and found inside `payment_methods[]` list and return the `payment_link`
6. Log events to server log. Detailed in DEBUG mode and brief in INFO mode
7. Log EveryPay requests and responses to `payment_logs` table

Prerequisites

1. PG-1653: BE: Create endpoint for payment initiation TO DO must be done

Acceptance criteria

1. User is redirected selected to payment method provider
2. User is returned to Payments Portal return route PG-1655: FE: Create route for returning from PSP TO DO
3. Payment process is finished and displayed in SEB EveryPay portal Test Portal - SEB
    a. Raimo Johanson will provide login credentials
4. `payment_logs` table has records of interactions with EveryPay API