TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Markus Maila 206479IABB

Maksete töötlemise ja arveldamise tarkvara integratsioon Playerbank OÜ näitel

Bakalaureusetöö

Juhendaja: Karl-Erik Karu
MSc
Kaasjuhendaja: Oliver Orav
BSc

Tallinn 2023

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: Markus Maila

11.05.2023

Annotatsioon

Playerbank OÜ hindab kõrgelt tehnoloogilist võimekust, mistõttu üritatakse manuaalseid toiminguid minimaliseerida. Pidev fookus on uute tehnoloogiliste lahenduste analüüsimisel ja nõuete sobivusel nende implementeerimisel. Bakalaureusetöö eesmärgiks on analüüsida erinevate maksete töötlemise teenuse pakkujaid ja luua maksete töötlemise ning arveldamise tarkvara integratsioon.

Käesoleva töö analüüsi osas võrreldakse omavahel tehnoloogiad, mida on võimalik kasutada eesmärgi saavutamiseks ja valitakse välja Stripe teenusepakkuja poolt loodud tehnoloogia. Bakalaureusetöö praktilises osas tegeleb autor arendusliku lahenduse elluviimisega ja rakendust testides ühik- ning integratsioonitestidega. Töö tulemusena analüüsitakse erinevate maksete töötlemise teenuse pakkujaid ja luuakse maksete töötlemise ning arveldamise tarkvara integratsioon. Saavutatud lahendus võimaldab Playerbank OÜ klientidel pakutavaid teenuseid tarbida ja ettevõttel teenuste eest tulu realiseerida.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 6 peatükki, 11 joonist, 4 tabelit.

Abstract

Payment processing and billing software integration based on the example of Playerbank OÜ

Playerbank OÜ values highly technological capability, therefore attempting to minimize manual operations. There is a constant focus on analysing new technological solutions and implementing them when suitable. The goal of this bachelor's thesis is to create payment processing and billing software integration.

In the analysis section of this thesis, technologies that could be used to achieve the goal were compared, and an approach based on Stripe technology was chosen. In the practical section of the bachelor's thesis, the author was involved in the development of the solution, implementing it, and testing the application with unit and integration tests. As a result, the initial goals were fully achieved; the solution obtained enables Playerbank OÜ clients to consume the offered services and the company to generate revenue for the services.

The thesis is in Estonian and contains 22 pages of text, 6 chapters, 11 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	Rakendusliides, komplekt rakendustarkvara ehitamiseks ehk *Application Program Interface*
Back-end	Rakenduse loogika ja andmete salvestamine
B2B	Äritüüp, mis viitab ettevõtetele või äriühingutele, kes müüvad oma kaupu või teenuseid teistele ettevõtetele ehk *business-to-business*
B2C	Äritüüp, mis viitab ettevõtetele või äriühingutele, kes müüvad oma kaupu või teenuseid otse tarbijale ehk *business-to-customer*
DTO	DTO ehk *Data Transfer Object* on andmeedastus objekt
Flat-rate billing	Kindlasummaline arveldamine (kuu või aasta baas)
Front-end	Graafiline kasutajaliides
Jira	Tarkvara arendusprojektide haldamiseks ja planeerimiseks
JIT	Interpreteerija ehk *Just-In-Time interpreter*, mis kompileerib käivitamisel baitkoodi masinkoodiks ning optimeerib sageli kasutatavaid koodiosi, mis aitab kaasa rakenduse jõudlusele
JSON	JSON ehk *JavaScript Object Notation* on lihtsustatud andmevahetusvorming, mis põhineb *JavaScript*'i programmeerimiskeele alamhulgal
Library	Koodi kogum (moodul), mida programmeerijad saavad kasutada ülesannete optimeerimiseks
MoR	Juriidiline isik ehk *Merchant of Record*, kes müüb kaupu või teenuseid lõpptarbijale. Tehingu lõppedes võlgneb lõppklient oma ostu eest tasu MoR-ile.
Multiple price billing	Asukohapõhine erineva hinnaga arveldamine
Per seat billing	Arvestatakse inimeste arvu põhjal tasu summa
SQL	Andmebaaside päringukeel ehk *Structured Query Language*
Tiered billing	Ühiku maksumus võib varieeruda vastavalt valitud kogusele
Usage-based billing	Kasutuspõhine arveldamine
REST	*Representational State Transfer*
SaaS	Tarkvara teenusena ehk *Software as a service* on pilvandmetöötluse pakkumine teenusepakkuja määratud tarkvara rentimisena

Security manager	Klass, mis võimaldab rakendustel turbepoliitikat rakendada
Subscription	Tellimus ehk *subscription* on ärimudel, mille puhul klient peab toote kasutamise eest maksma regulaarse ajavahemiku järel korduvat hinda
Stand-up	Igapäevane arendustiimi koosolek

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Playerbank OÜ hindab kõrgelt tehnoloogilist võimekust, mistõttu üritatakse manuaalseid toiminguid minimaliseerida. Käesoleva töö probleemiks, mida autor lahendab, on varasemalt puudunud makselahendus, millest tulenevalt ei suutnud ettevõte pakkuda klientidele terviklikku lahendust ega realiseerida oma teenuste eest saadud tulu.

Käesoleva töö eesmärk on luua vajaolev integratsioon Playerbank OÜ ja maksete töötlemise ning arveldamise tarkvara teenust pakkuva süsteemi vahel, mille tulemusel tekib võimalus saada pakutavate teenuste eest tulu. Tulemusena saadud lahendus peab olema kasutaja jaoks piisavalt mugav, lihtsasti hallatav ja muudaks makseprotsessi võimalikult kiireks, stabiilseks ning autonoomseks. Loodav lahendus peab järgima Playerbank OÜ nõudmisi ja vajadusi. Lahendus peab olema lihtsasti jälgitav ning kaetud nii ühik- kui ka integratsioonitestidega.

Bakalaureusetöö analüüsimise osas uuritakse, millised võimalused ja tehnoloogiad on mugavuse, juriidika, kiiruse ja turvalisuse poolest kõige sobilikumad makselahenduse integreerimiseks. Selgitatakse välja iga tehnoloogia eripärad ning analüüsitakse funktsionaalseid ja mittefunktsionaalseid nõudeid. Selle osa tulemusena valitakse võrdleva analüüsi abil tehnoloogia, mida hakatakse praktilises osas kasutusele võtma.

Bakalaureusetöö praktilises osas proovitakse analüüsist lähtudes välja valitud tehnoloogia abil lahendada töö eesmärki. Käesolev osa nõuab nii olemasolevate tööriistade kui uute kasutusele võetavate tehnoloogiate tundmaõppimist.

2 Taust

Käesolevas peatükis tutvustatakse ettevõtte ja projekti tausta, millele antud bakalaureuse töö põhineb. Ühtlasi selgitatakse makseprotsessi üldist olemust ja kasutust ning antakse ülevaade kahe makselahenduse pakkuja tehnoloogiatest – Stripe ja Paddle.

2.1 Ettevõtte taust

Playerbank OÜ on 2023. aastal asutatud ettevõte, mis pakub professionaalsetele võrkpalluritele, agentidele ja klubidele platvormi, mis aitab neid omavahel kokku viia ning leida uusi spordikarjääri võimalusi.

Rahvusvaheline mängijate hankimine on manuaalne protsess, mille jooksul võtavad klubid ise ühendust agentide või mängijatega. Ideed valideerides teostas Playerbank erinevate osapooltega intervjuusid, mille tulemusena kulub keskmiselt ühe mängija analüüsiks ca 20 tundi.

Playerbank pakub võimalust luua mängijatel, klubidel ja agentuuridel profiil asjakohase teabega, mille järgi rakenduse algoritmid viivad mängijad klubide või agentuuridega parima sobivuse järgi kokku. Näiteks sisaldab mängija profiil füüsilisi andmeid, mängude statistikat, videoid, varasemaid lepinguid ja soovitajaid. Agentuuride profiil koosneb mängijatest, keda agentuur parasjagu esindab. Klubide profiil sisaldab üldist infot klubi kohta ning võimalust lisada aktiivseid pakkumisi. Mängijatel ja agentidel on sobiva pakkumise leidmisel võimalik kandideerida ja meeskonnaga läbirääkimisi alustada.

Tootearendusprotsessi käigus rakendas ettevõte MVP lähenemist, kuna oluline oli teenuse turule viimine võimalikult varakult, et saada kasutajatelt kiiret tagasisidet. Toode oli turule viimise hetkel saavutanud vajaliku minimaalse funktsionaalsuse ning klientide tagasiside põhjal hakati rakendust täiendama. MVP strateegia kasutamise tulemusena vähendas ettevõtte riske ja kulutusi, kuna toodet ei arendatud täielikult välja, ning saavutati parem arusaam oma sihtgrupi vajaduste üle.

2.2 Maksete töötlemise ja arveldamise tarkvara

Kolmanda osapoole maksete töötlemise ja arveldamise tarkvara kasutamine on nii alustavate kui ka suurte ettevõtete puhul enamlevinud nähtus. Kolmanda osapoole maksete tehnoloogia kasutuselevõtt võimaldab ettevõttel viia toode võimalikult varakult turule ja lihtsustada märgatavalt tarkvara arendajate tööd. Lisaks makseteenuse töötlemise ja arveldamise tarkvarale pakutakse ettevõttele mugavat kasutajaliidest, mis hõlbustab oluliselt maksetega seotuid toiminguid [1].

Maksete töötlemise ja arveldamise tarkvara pakkuvad ettevõtted on vahendajad kaupmeeste, klientide ja tehinguid töötlevate finantsasutuste vahel. Need ettevõtted kooskõlastavad mitte-sularahatehinguid, valideerides kogu maksega kaasa tulnud teabe ja toimetades rahalisi vahendeid kaupmehele pärast müügi lõpuleviimist. Maksete töötlejatena tagatakse mitte ainult raha edastamist kaupmehele, vaid veendutakse ka tehinguga seotud panga raha kättesaamises.

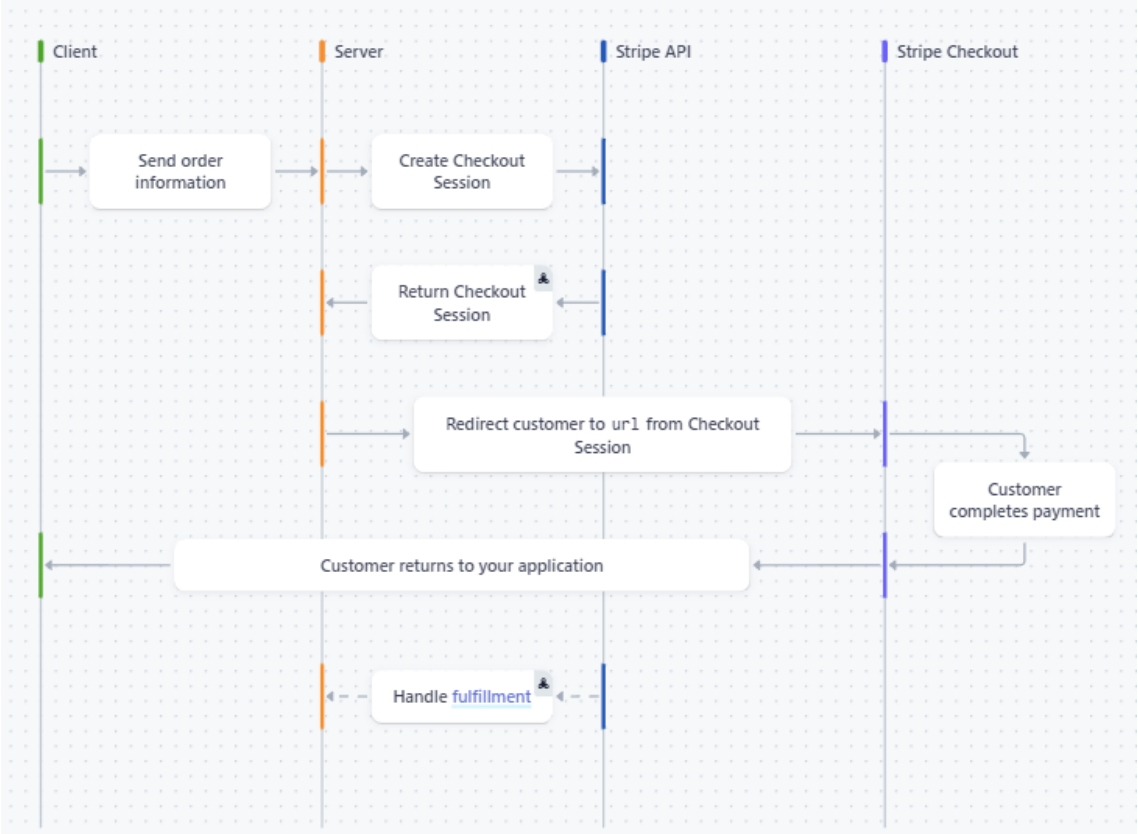
Maksete töötlemise ja arveldamise tarkvara teenused hõlmavad tehingu autoriseerimist, rahastamist ja selle lõpuleviimist. Kuigi tehingu tegemisel kestab kogu protsess sekundeid, siis tegelikult viiakse läbi selle aja jooksul allpool kirjeldatud keerulised toimingud (Joonis 1). Kaupmees saadab oma maksetöötlejale tehingu autoriseerimise taotluse ja seejärel viib maksetöötleja läbi vajalikud sammud:

- Makse tegija esitab tehinguga seotud info pangale
- Pank töötleb tehingut teatud kriteeriumide alusel
- Seejärel saadab pank kas positiivse või negatiivse vastuse kaupmehe pangale ja seejärel kaupmehele endale

Pärast makse autoriseerimisprotsessi järgneb arveldamisprotsess, mille jooksul kantakse tehinguga seotud rahalised vahendid kaupmehe kontole. Pangakaardi väljastanud pank saab tehinguga seotud vajalikud andmed ja teostab järgmised toimingud:

- Kaardi väljastanud pank nõuab tehingu eest kaardiomanikult teatud summa
- Seejärel teeb kaardi väljastanud pank ülekande kaupmehe panka, millest on maha arvatud vahendustasu

- Kaupmehe pank kannab summa kaupmehe kontole [2].



Joonis 1. Makse töötlemise protsess

2.3 Stripe

Stripe on makseteenuste pakkuja, mis võimaldab kaupmeestel käsitleda ja vastu võtta krediit- ja deebetkaardi, kuid ka teisi makseviise [3]. Stripe teenus sobib kõige rohkem nendele ettevõtetele, kes teevad suurema osa oma müügist veebis, kuna selle tehnoloogia on peamiselt suunatud just veebipõhisele müügile [4]. Platvorm tuli turule 2010. aastal kahe Iiri päritolu venna John ja Patrick Collisoni poolt, kes avastasid internetiäri alustamises tühimikku. Nimelt polnud kõige raskem osa idee välja mõtlemine, tehniline implementeerimine ega klientide leidmine, vaid nimelt raha liigutamine. Firma pakkus alguses seitset koodirida ja lubas, et mingeid muid muudatusi pole vaja tarkvara kasutajatel rohkem teha. Arendajad, kes lõid integratsiooni Stripe API kaudu, ei pidanud aastate jooksul makseprotsessiga rohkem tegelema. [5]

Stripe kasutab *webhook*'e, et integreeritavale rakendusele teavitada, kui mingisugune sündmus on maksmisel toimunud. *Webhook*'id on väga otstarbekad asünkroonsete sündmuste korral nagu näiteks kui kliendi pank kinnitab makset, korduva makse

õnnestumine jne. Stripe saadab *webhook*'ina JSON formaadis andmeid, mille põhjal saab integreeritav rakendus teha vajalikke toiminguid enda serveripoolses osas. [6]

2.4 Paddle

Paddle pakub SaaS ettevõtetele makseteenust, mis hõlmab endas terviklikku lahendust maksete läbiviimiseks. Paddle on asutatud 2012. aastal Christian Owens ja Harrison Rose'i poolt ning selle peakontor asub Londonis. [7]

Paddle on MoR teenusepakkuja, millest tulenevalt jääb maksete käsitlemine ja tehingu järgsete toimingute vastutus terviklikult Paddle'i kanda. See hõlmab endas maksude, PCI (ingl *Payment Card Industry Data Security Standard*), kohaliku tarbijaseaduse ja tagasimaksete järgimist. MoR mudel aitab ettevõtetel, kes soovivad laieneda rahvusvaheliselt, lihtsustada uute makseviiside ja valuutade rakendamist, sest protsessi korrektses toimimises jääb Paddle vastutama, mis võimaldab ettevõttel keskenduda täielikult oma prioriteetidele. [8]

Sarnaselt Stripe teenusele kasutab ka Paddle *webhook*'ide loogikat. Viimaseid saadetakse sündmuste käivitamisel Paddle'i platvormil. Sündmusteks on näiteks uus makse, uus tellimus või summa ülekanne ettevõtte pangakontole. Enne sündmuste käsitlemist tuleb määrata lõpp-punkt, kuhu pihta hakatakse HTTP POST-päringuid tegema. POST-päringuid saadetakse application/x-www-form-urlencoded formaadis. [9]

3 Analüüs

Käesolevas peatükis analüüsitakse loodava lahenduse funktsionaalseid ja mittefunktsionaalseid nõudeid. Uuritakse ja võrreldakse kahte erinevat maksete töötlemise ja arveldamise tehnoloogiat ning kolme erinevat programmeerimiskeelt, võttes arvesse nende funktsionaalsuse, jõudluse, turvalisuse jms mõõdikuid. Viiakse läbi andmebaasi analüüs, mille käigus tutvustatakse PostgreSQL süsteemi, ettevõtte parimaid tavasid ja käesoleva töö käigus looduid andmebaasi tabeleid. Analüüsi tulemuste põhjal valitakse tehnoloogiad maksete töötlemise ja arveldamise tarkvara integratsiooni rakendamiseks.

3.1 Nõuded kavandatavale lahendusele

Makselahenduse kasutuselevõtuks tuleb valida sobilik tehnoloogia, mida on võimalik integreerida olemasoleva süsteemiga. Enne projekti algust, esitati autorile Playerbank'i poolt tehnoloogilised ja mittefunktsionaalsed nõuded, millele peab lõpplahendus vastama. Järgmisena ongi välja toodud peamised tehnoloogilised ja mittefunktsionaalsed nõuded, millele peavad kavandatava lahenduse loomisel kasutusele võetavad tehnoloogiad vastama.

Tehnoloogilised nõuded:

1. Turvalisus: lahendus peab olema turvaline, et maksete töötlemise ja arveldamise tarkvara kasutamine oleks kasutaja jaoks usaldusväärne. Makse läbiviimine sisaldab delikaatseid isikuandmeid, mistõttu peab teostama põhjaliku testimise integratsiooni korrektses toimimises.

2. Keerukus: lahendus on suunatud kliendile, sellest tulenevalt peab lõpplahendus olema lihtsasti kasutatav ja intuitiivne. Tehniliselt peab saama lahendust lihtsa vaevaga edasi arendada juhul, kui ettevõtte otsustab tulevikus pakkuda klientidele uusi lahendusi ning muid makselahendusi.

3. Kiirus: lahendus ei tohi olla kasutaja jaoks aeglane, kuna see mõjub halvasti kliendi kasutajakogemusele.

4. Aktuaalsus: kasutusele võetav tehnoloogia peab olema usaldusväärne ja tulevikus edasi arendatav. Valitud *back-end* programmeerimiskeel peab olema hallatav ning stabiilselt tehnoloogiaga kaasas käiv, et vältida tulevikus moderniseerumisega seotud probleeme.

5. Teegid: teekide kasutamine peab olema minimaalne ja õigustatud, et vähendada ründepindalat ja parendada koodi hallatavust. Võimalusel valida programmeerimiskeel, millel on olemas valitava maksete töötleja ja arveldamise tarkvara teenuse pakkuja teek.

6. Andmebaasi kasutamine: peab olema kasutaja jaoks mugav ehk sisaldama andmebaasi API-kasutajaliidest.

Funktsionaalsed nõuded:

1. Maksete töötlemise ja arveldamise tarkvara teenuse pakkuja peab võimaldama maksuraporti koostamist.

2. Ettevõtte keskendub peamiselt USA ja Euroopa turule, mistõttu peab tarkvara suutma arveldada USD ja EUR valuutasid, kuid võimalusel toetama ka teisi tugevamaid valuutasid.

3. Ostukorvi kasutajaliidest peab saama ühtlustada Playerbank OÜ brändi stiiliga.

4. Tarkvara peab sisaldama nii tellimuse ja ühekordse makse funktsionaalsust ning võimaldama deebet- ja krediitkaardi makseid.

5. Vahendustasu tehingu pealt on lubatud maksimaalselt 5% + 1€.

6. Integratsioon tarkvara teenusepakkuja vastu peab olema võimalikult lihtne, et tulevikus uusi ärilisi eesmärke saaks hõlpsasti täide viia.

7. Tarkvara peab aktsepteerima B2B ja B2C äritüüpe.

Kogu arendustöö vältel pidi autor arvesse võtma ja järgima tehnoloogilisi nõudeid. Funktsionaalsed nõuded on sätestatud vastavalt ettevõtte nõuetest.

3.2 Tehnoloogia analüüs

3.2.1 Stripe võrdlus Paddle'iga

Stripe ja Paddle lahendavad sarnast probleemi, muutes makseprotsessi ettevõtete jaoks autonoomseks ja kergesti kasutatavaks, erinevad nad nii funktsionaalsete kui ka mittefunktsionaalsete nõuete poolest, mis tuuakse välja järgnevas tabelis (Tabel 1):

Tabel 1. Stripe võrdlus Paddle'iga [10], [11].

	Stripe	**Paddle**
Maksude eest vastutamine ja nende haldamine	Stripe lihtsustab maksuraportite koostamist, kuid kasutavad ettevõtted vastutavad maksude tasumise eest ise.	Paddle vastutab ise täielikult maksude tasumise eest.
Lubatud valuutad	135+ valuutat	20+ valuutat
Ostukorv	Stripe Elements'i kasutades on võimalik ostukorvi osade kaupa üles ehitada ja on tõlgitud 35+ keelde	Kaks erinevat ostukorvi: veebilehele kinnitatud või hüpikakna vormis. Ostukorv tõlgitud 16+ keelde
Tellimuse võimalused	*Flat-rate billing*, *multiple price billing*, *per-seat billing*, *usage-based billing*, *tiered billing*	*Flat-rate billing*, *tiered-billing*, *per-seat billing*
Tehingu vahendustasu	Tehingu pealt 1,4% + 0,25€ Euroopa klientidele ja 2,9% + 0,25€ mitte Euroopa klientidele	Tehingu pealt 5% + 0,5$
Tehniline erinevus	Stripe'i integreerimine ja testimine on lihtne, kuna on olemas kasutajasõbralik API ja Stripe *library*.	Java programmeerimiskeeles puudub eraldi *library*. Test keskkond on kergesti kasutatav.
Lubatud äritüübid	Peaaegu kõik äritüübid on lubatud kasutama Stripe teenust.	SaaS äritüübid. Paddle'iga on olnud juhtumeid, mille käigus suleti ettevõtete kontod ette teatama väidetavalt sobimatu toote või teenuse müümise tõttu.

Läbiviidud analüüsi tulemuste põhjal osutus valituks Stripe tehnoloogia. Hoolimata maksude vastutamise eest, mängis otsustamisel suurt rolli Stripe'i Java teegi olemasolu ja väiksemad tehingutasud. Funktsionaalsetest nõuetest tingituna peab ostukorv vastama Playerbank OÜ brändi stiilile ning Stripe Elements'i kasutamine võimaldab ostukorvi täpselt vastavalt ettevõtte soovidele kohandada. Ettevõtte meeskond oli konsensusel, et kavatsus on lähiajal tungida ka teiste riikide turgudele, mistõttu muutub aktuaalsemaks lubatud valuutade olulisus ja tehingute arvu kasvades vahendustasu suurus.

3.2.2 *Back-end* programmeerimiskeelte võrdlus

Maksete töötlemise ja arveldamise tarkvara integreerimisel on valikus mitmeid erinevaid võimalusi. Järgnevalt tuuakse välja programmeerimiskeelte võrdlus ettevõtte poolt kehtestatud kriteeriumide järgi (Tabel 2):

Tabel 2. *Back-end* programmeerimiskeelte võrdlus [12], [13], [14], [15], [16], [17], [18], [19].

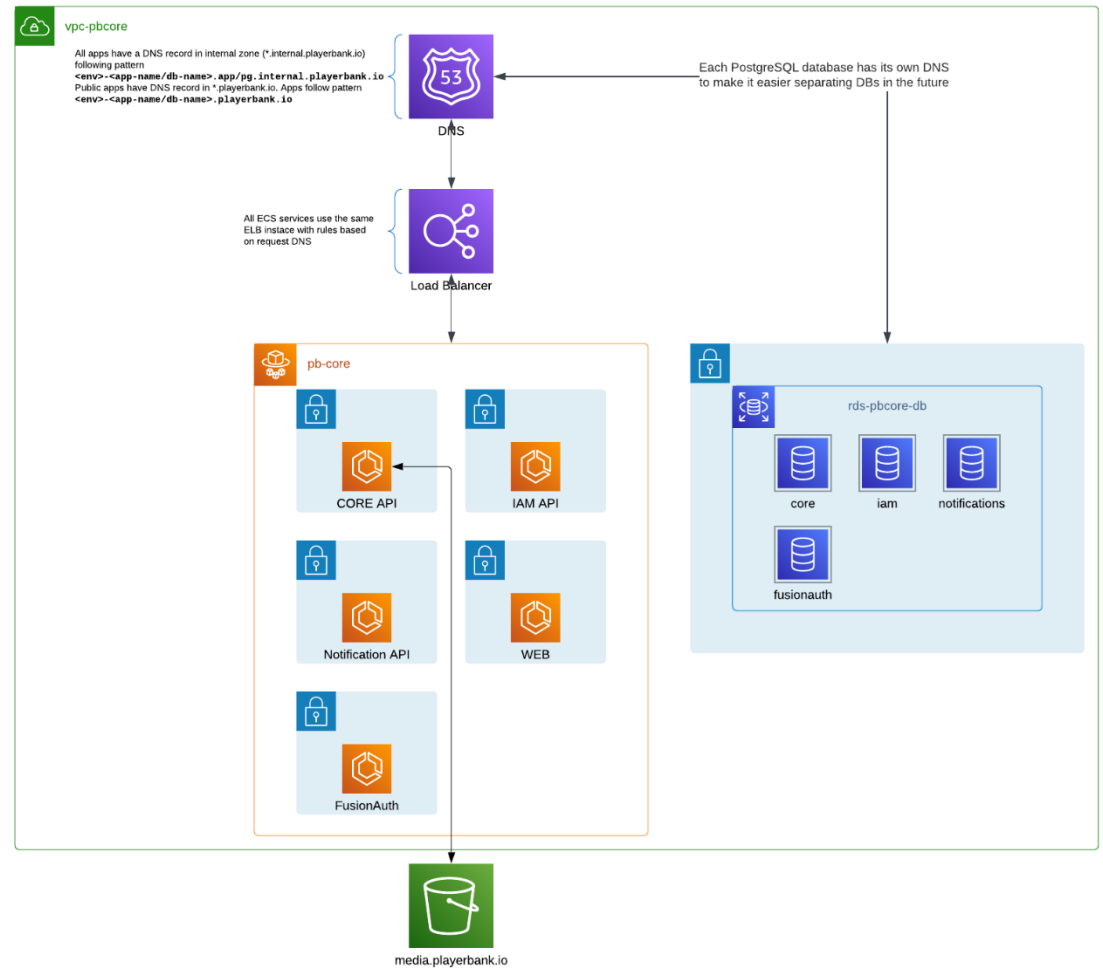
Kriteerium	**Python**	**C#**	**Java**
Turvalisus	Peab täieliku turvalisuse tagamiseks läbi viima detailse kvaliteedi kontrolli	Kompileeritud keel, mis tähendab, et avalikku serverisse salvestatud kood on binaarses vormis. See omakorda raskendab märgatavalt juurdepääsu koodile	Javas on kasutusel turvahaldur, mis võimaldab määratleda klassidele juurdepääsureeglid
Keerukus	Koodi lugemine on sarnane tavalisele inglise keele lugemisele, mis teeb kodeerimise lihtsaks ja kergesti õpitavaks	Lisaks keele omapärale lisab raskust .NET teekide kasutamine ja multiplatvormilisuse juurutamine	Disainiti selliselt, et teda oleks lihtne kasutada, kirjutada, kompileerida ja siluda
Kiirus	Python on interpreteerija keel, kus iga koodirida käivitatakse üksteise järel ning tüübid tehakse kindlaks rea käivitamisel, mis mõjub jõudlusele halvasti	Java ja C# omavad sarnast kompileerimisaega, kuid rakenduse jõudluse võrdlusnäitajad näitavad C#'i jaoks veidi kiiremaid tulemusi	Java käivitamine kipub olema aeglane, aga kompileerimine JIT abil on reegli järgi kiirem kui interpreteerija keeltel

Kriteerium	Python	C#	Java
Aktuaalsus	Tohutult palju informatsiooni ja abimaterjale keele kasutuse kohta ning laialdaselt kasutusel üle kogu maailma	Tarkvaraarenduses üks populaarsemaid keeli, mis tagab C# kogukonna laiapõhjalise toe	Java on tänapäeval üks usaldusväärseimaid programmeerimiskeeli selle pideva arengu tõttu
Teegid	Ulatuslikult teeke	Väga palju väliseid kui ka .net teeke, mis aitavad prototüüpe kiiremini realiseerida	Ulatuslikult teeke
Andmebaasi kasutamine	Puudub andmebaasi kasutajaliides, mis teeb andmebaasidega töötamise keeruliseks	Microsoft ADO.NET olemasolu andmebaasi ühenduse haldamiseks, andmete otsimiseks ja muudeks olulisteks toiminguteks	JDBC API olemasolu andmebaasi haldamiseks oluliselt lihtsustab protsessi

Läbiviidud analüüsi tulemuste põhjal otsustas autor kasutada lahenduseks Java programmeerimiskeelt. Peamisteks valiku argumentideks osutusid andmebaasi kasutus, turvalisus ja keerukus. Lisaks eelmainitud argumentidele puutub autor iga päev kokku Java programmeerimiskeelega, mistõttu jääb vahele programmeerimiskeele selgeksõppimise protsess. Java valiku eeliseks on ka asjaolu, et senised Playerbank'i mikroteenused on arendatud Java keeles, mis muudab maksete töötlemise ja arveldamise tarkvara integratsiooni sidumise olemasoleva süsteemiga mõnevõrra lihtsamaks. Arenduskeskkonnana kasutab autor IntelliJ IDEA't ning koodihalduse eest vastutab Bitbucket.

3.3 Andmebaasi analüüs

Projektis kasutatakse ettevõtte poolt juba seadistatud PostgreSQL andmebaasi andmete hoidmiseks. PostgreSQL on olnud turul enam kui 35 aastat ning on põhjusega pälvinud tugeva usaldusväärsuse oma arhitektuuri, töökindluse ja tarkvara kogukonna pühendumuse tõttu, et pakkuda järjepidevalt tõhusaid ning uuenduslikke lahendusi [20]. Autor puutus projekti käigus *back-end* repositooriumitest kokku ainult IAM projektiga (Joonis 2).



Joonis 2. Playerbank mikroteenuste arhitektuur

3.3.1 Andmebaasi arhitektuur

Andmebaaside struktuurid on nii toodangus kui testimises identsed, kuid jooksevad eraldi serverite peal. Ettevõtte arendajad kasutavad lokaalseks arendamiseks ja katsetamiseks

konteineriseerimistehnoloogiat Docker tööriista abiga. Konteinereid luuakse tõmmiste (ehk *image*) kaudu (Joonis 3) ja käivitatakse *docker-compose up* käsu abil.

```
version: '3.1'

services:
  db:
    image: postgres
    container_name: iam_db
    ports:
      - 54320:5432
    environment:
      POSTGRES_DB: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - ./src/main/resources/db/init.sql:/docker-entrypoint-initdb.d/init.sql
```

Joonis 3. Docker Compose fail

3.3.2 Andmebaasi haldamine

Rakendus kasutab Liquibase andmebaasi haldamise teeki. Liquibase'i abiga on võimalik andmebaasi koodile juurutada külge versioon, mis muudab andmekogu jälgimise lihtsamaks [21]. Liquibase võimaldab andmebaasi muudatusi määrata SQL, JSON, YAML ja XML vormingus (edaspidi *changeset*), millest viimane on kasutusel ka käesolevas ettevõttes [22]. Andmebaasi muudatust tehes on Playerbank'is parimaks tavaks algul identifitseerida, mis tüüpi muudatust tahetakse teha. Tabeli või veergude modifitseerimisel luuakse uus *changeset* baseline.xml failis. Andmete lisamisel arendus- ja testkeskkonda kasutatakse dev.xml faili ning lisatakse andmeid samuti *changeset*'i abil. Uue kirje loomisel määratakse ära *changeset*'i id ja autor. Uue tabeli loomisel kasutatakse lisaks unikaalsele id'le veel *resource_id*'d, mis võimaldab andmeid turvalisemalt identifitseerida. Viimase genereerimiseks tuleb pärast iga tabeli loomist luua tabelile omapärane triger (Lisa 2), mis genereerib vastavalt etteantud eesliitele tabeli kirjetele unikaalse *resource_id*.

3.3.3 Andmebaasi tabelid

3.3.3.1 Tabel *subscription*

Subscription tabel sisaldab endas andmeid, mida saab tellimuse kohta *front-end*'is kasutajale välja kuvada (Lisa 3). Arvestades, et maksete töötlemise ja tarkvara teenuse pakkuja salvestab endale kõik andmed, siis puudus otsene vajadus kõike ettevõtte

andmebaasi maha salvestada (Tabel 3). Tabelis olevate veergude *provider* eesliide määrab ära need veerud, mille andmed saadakse maksete töötlemise teenuse pakkujatelt.

Tabel 3. Tellimuse tabel

subscription	
resource_id	varchar(21)
provider_customer_id	varchar(21)
provider_subscription_id	varchar(255)
provider_price_id	varchar(255)
status	varchar(10)
date_created	timestamp
date_modified	timestamp
provider_payment_method_id	varchar(100)
id	bigint

3.3.3.2 Tabel *plan*

Plan tabel sisaldab sarnaselt *subscription* tabelile samuti andmeid, mida saab tellimuse plaani kohta kasutajale välja kuvada (Lisa 4). *Plan* tabel võimaldab kasutajale kuvada, milliseid tellimuse plaane pakutakse (Tabel 4).

Tabel 4. Tellimuste plaanide tabel

plan	
resource_id	varchar(21)
provider_product_id	varchar(255)
plan_name	varchar(255)
provider_price_id	varchar(255)
currency	varchar(3)
price	numeric(16,2)
date_created	timestamp
date_modified	timestamp
period_type	varchar(50)
id	bigint

4 Realisatsioon

Järgnevas peatükis antakse ülevaade arendusprotsessist valitud tehnoloogia põhjal. Viimasena tutvustatakse, kuidas saavutati soovitud lõppeesmärk, tagatakse koodi kvaliteet ja milliseid teste kasutatakse selle saavutamiseks.

4.1 Arendusprotsess

Alustuseks loodi Jira keskkonda maksete töötlemise ja arveldamise tarkvara integratsiooni *epic* probleem, mis omakorda sisaldas alamprobleemide kogumit. Autori meeskond viis läbi igal nädalal ühe *stand-up* koosoleku, mille jooksul loodi vajadusel juurde alamprobleeme, prioritiseeriti ülesandeid ning dokumenteeriti protsessi arengut. Enne tehniliste ülesannete tegemist loodi *epic* haru (*branch*), kuhu hakati koodi ühendama (*merge*) alamprobleemide harudest kokku. Haru nime konventsiooni kohaselt algab Jira keskkonnast saadud ülesande identifikaatori ja nimega „feature/DEV-XXX-NAME“, kus XXX on ülesande järjekorranumber ja NAME on ülesande pealkiri, mistõttu on haru nime järgi võimalik selgeks teha, millega on tegemist.

Koodi arendamisel lähtuti agiilsetest meetoditest, mis tähendab tarkvara arendamist järkjärgulisel viisil keskendudes paindlikkusele. Eesmärgiks oli võimalikult kiiresti töötav rakendus kasutusele võtta ja alles pärast seda keskenduda koodi refaktoreerimisele ja parendamisele. Analüütiku puudumise tõttu olid ülesannete kirjeldused minimalistlikud ega mindud sügavuti detailidesse, mistõttu oli arendajatel koodi kirjutades vastutus vajadusel analüüsida ülesannet detailsemalt. Seejärel kirjutas autor olemasolevale koodile juurde ühik- ja võimalusel integratsioonitestid. Järgmisena laadis arendaja oma koodi Bitbucket keskkonda teistele arendajatele ülevaatuseks ja testimiseks. Üles laaditud koodi pidi alati dokumenteerima, et säilitada ülevaade olemasolevast arendusest ja koodi õigsusest ning vajalikkusest. Koodi sobimatusel või vigade ilmnemisel saadeti kood tagasi autorile, kes viis läbi vajalikud parandused ja kordas ülaltoodud protsessi uuesti. Koodi sobivusel ja korrektsel toimimisel lisati see *epic* harru ning alamprobleemide otsa lõppedes lisati *epic* haru omakorda toodangu harusse.

4.2 Lahendus

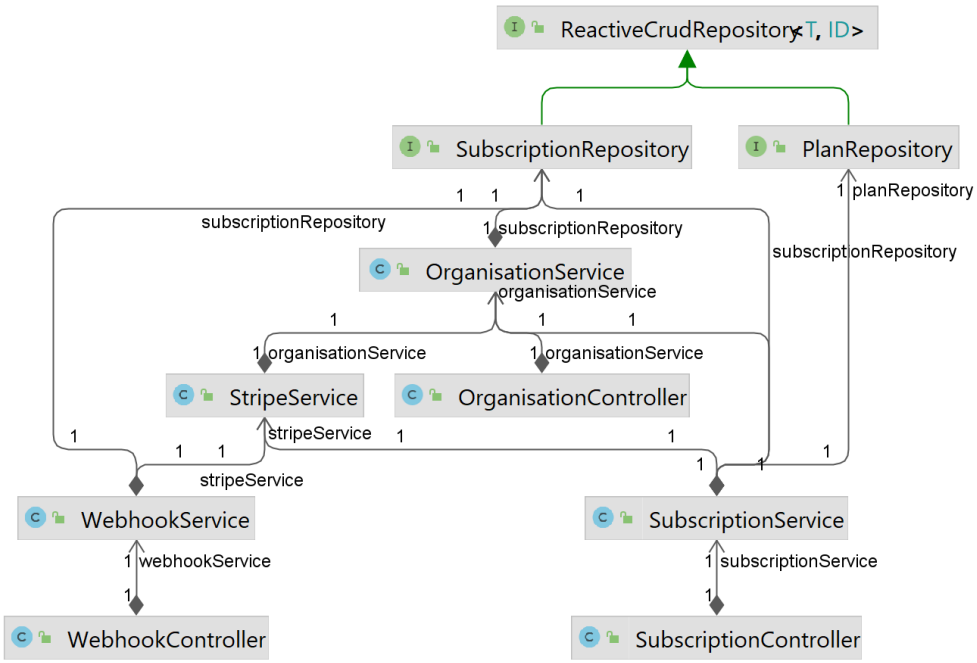
Stripe saadab *webhook*'ina JSON formaadis andmeid, mida tuleb enda rakenduses käsitleda ja andmebaasi õiges formaadis maha salvestada. Lahenduse käigus luuakse kaks kontrolleri, teenuse, domeeni ja andmekihi klassi ning neli DTO klassi. Lisaks luuakse ühik- ja integratsioonitestide klassid ning abistavad klassid testide kirjutamiseks.

4.2.1 Vastuvõetavad ja salvestatavad andmed

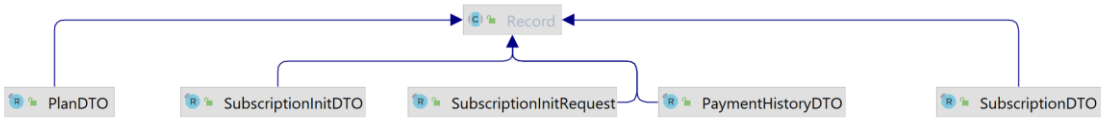
Vastuvõetavatest andmetest (Lisa 5) salvestatakse *customer*, *id*, *priceId*, *status* ja *defaultPaymentMethod* väljade väärtused *subscription* andmebaasi tabelisse. Neid andmeid kuvatakse kasutajale Playerbank rakenduses, kui küsitakse infot tellimusega seotud andmete kohta.

4.2.2 Klassid

Maksete töötlemise ja arveldamise tarkvara integratsiooni implementeerimiseks vajalik kood kirjutati *main* moodulisse. *Controller* kaustas hoitakse kontrolleri klasse, *Domain* kaustas teenuse ja domeeni klasse (Joonis 4) ning DTO kaustas DTO klasse (Joonis 5).



Joonis 4. *Main* mooduli *Controller* ja *Domain* kaustade klassidiagramm



Joonis 5. *Main* mooduli DTO kausta klassidiagramm

4.2.2.1 Kontrolleri klassid

Kontrolleri klassid sisaldavad endas ainult otspunktide meetodeid ja vajalikke annotatsioone üle terve klassi kui ka spetsiifilise meetodi kohta. Meetodi loogika peitub teenuste klassides, mistõttu imporditakse vajalik teenuse klass ja kasutatakse vastavat meetodit (Joonis 6).

```
@RestController
@RequestMapping(value = "/v1/webhooks")
@RequiredArgsConstructor
public class WebhookController {

    private final StripeWebhookService stripeWebhookService;

    @PostMapping(value = "/stripe", consumes =
MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.OK)
    public Mono<Void> handleStripeWebhook(ServerWebExchange webExchange,
@RequestBody String rawJsonBody) {
        return stripeWebhookService.handleStripeWebhook(rawJsonBody,
webExchange);
    }

}
```

Joonis 6. *Webhook* kontrolleri näide

Kõikidel tulevastel *webhook*'ide otspunktidel hakkab olema „/v1/webhooks" eesliide, mis ühtlustab kontrolleri otspunktide struktuuri ja kategoriseerib need vastavalt eesliite järgi ühte kategooriasse.

4.2.2.2 Teenuse klassid

Teenuse klasse kasutatakse andmebaasiga suhtlemiseks. Teenuse klassid sisaldavad endas kontrolleri loogikat ja tavaliselt ühtlustatakse teenuse meetodi nimi seda kasutatava kontrolleri klassi meetodi nimega (Joonis 7). Nende abil imporditakse vajalikke teiste teenuste kui ka andmekihi klasse, mis võimaldavad suhtlust andmebaasiga. Teenuse klassides kasutatakse tihtipeale abimeetodeid koodi puhtuse hoidmiseks ja dubleerimise vältimiseks.

```
@Service
@Slf4j
@RequiredArgsConstructor
public class SubscriptionService {
    private final OrganisationRepository organisationRepository;
    private final AccountRepository accountRepository;
    private final SubscriptionRepository subscriptionRepository;
    private final StripeService stripeService;
    private final OrganisationService organisationService;
    private final PlanRepository planRepository;
    private final ApiRelationRepository apiRelationRepository;

    public Mono<SubscriptionResponse>
getActiveSubscription(Mono<PlayerbankJWT> token) {
        return token.flatMap(t -> getActiveSubscription(t.getResourceId()));
    }

    public Mono<SubscriptionResponse> getActiveSubscription(String
resourceId) {
        return subscriptionRepository.findActiveByOwnerResourceId(resourceId)
                .map(SubscriptionResponse::create);
    }
```

Joonis 7. *Subscription* teenuse klassi näide

4.2.2.3 Domeeni klassid

Domeeni klasse kasutatakse peamiselt reaalse maailma kontseptsioonide modelleerimiseks. Need klassid sisaldavad tavaliselt nii andmeid kui ka käitumist (meetodeid), mis võimaldavad selle objektiga suhelda [23] (Joonis 8).

```
public record Subscription(
      Long id, String resourceId, Instant dateCreated,
       Instant dateModified, String providerCustomerId,
       String providerSubscriptionId, String providerPriceId,
      String status, String providerPaymentMethodId,
      @With String ownerResourceId) implements ApiResourceBaseRecord
{

    public static Subscription create(
             com.stripe.model.Subscription stripeSubscription,
String ownerResourceId)
    {
        String priceId = getPriceId(stripeSubscription);
        return new Subscription(null, null, null, null,
                stripeSubscription.getCustomer(),
                 stripeSubscription.getId(), priceId,
                stripeSubscription.getStatus(),
                 stripeSubscription.getDefaultPaymentMethod(),
```

```
                ownerResourceId);
    }

    private static String getPriceId(
             com.stripe.model.Subscription stripeSubscription)
    {
        return stripeSubscription.getItems().getData().get(0).
getPrice().getId();
    }
```

Joonis 8. *Subscription* domeeni klassi näide

4.2.2.4 DTO klassid

DTO klasse kasutatakse peamiselt fikseeritud andmete edastamiseks. Nad lihtsustavad andmete edastamist ja töötlemist. Erinevalt domeeni klassidest ei sisalda DTO klassid käitumist (meetodeid) [24] (Joonis 9).

```
public record SubscriptionDTO(
        String resourceId, Instant dateCreated, Instant dateModified,
        String customerId, String subscriptionId,
        String priceId, String status) implements ApiResourceResponse {

    public static SubscriptionDTO fromSubscription(Subscription subscription)
{
        return new SubscriptionDTO(subscription.resourceId(),
             subscription.dateCreated(), subscription.dateModified(),
             subscription.providerCustomerId(),
             subscription.providerSubscriptionId(),
             subscription.providerPriceId(), subscription.status());
    }

    @Override
    public String getType() {
        return "subscription";
    }
}
```

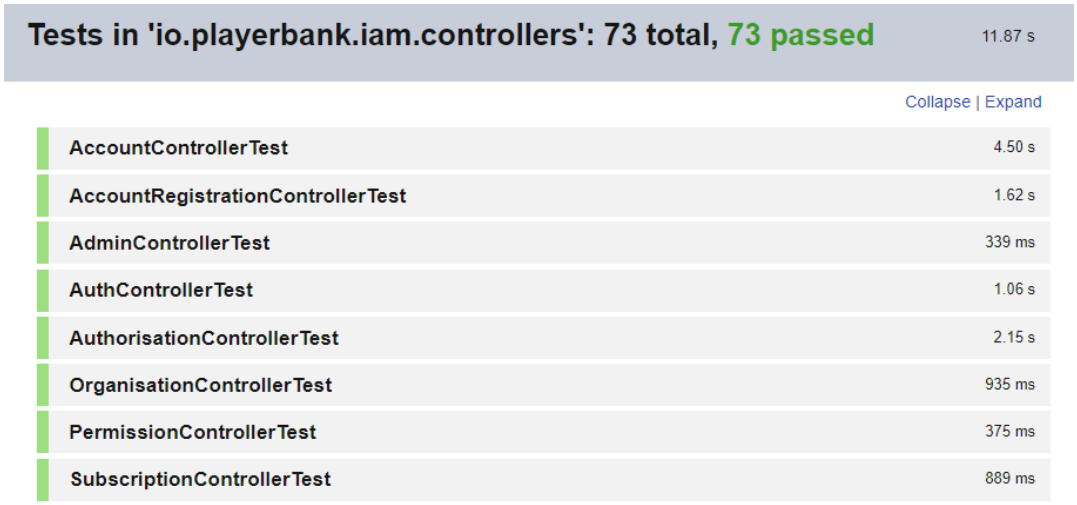
Joonis 9. *Subscription* DTO klassi näide

4.2.3 Testid

4.2.3.1 Integratsioonitestid

Integratsiooniteste kirjutades käiakse üle vähemalt kaks juhtumit. Esimene olukord, kus rakendus saab soovitud vastuse, ja teine olukord, kus tekitatakse tahtlikult viga. Kontrollitakse vastuse sisu, kas vastab õigele DTO klassile, süsteemi korrektsel toimimisel tagastatud andmete õigsust ja JSON formaadi korrektsust.

Kokku on 73 integratsioonitesti, testide kattuvus on 83% ja testid läbitakse 11,87 sekundiga (Joonis 10).

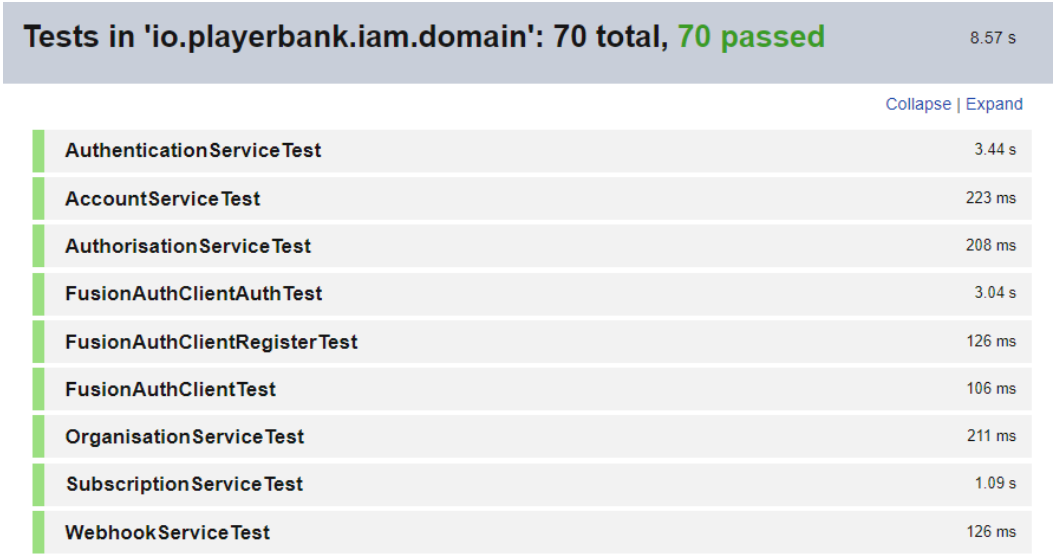


Joonis 10. Integratsioonitestimise tulemused

4.2.3.2 Ühiktestid

Ühikteste kirjutades kehtib sarnaselt integratsioonitestidele sama loogika. Testitakse võimalusel kõik juhud, et veenduda mitte ainult süsteemi korrektses toimimises, vaid ka tulevikus koodi refaktoreerimisel või muutmisel süsteemi järjepidevuses. Testidega kontrollitakse tagastatud andmete õigsust ja DTO tüübi nimetust.

Kokku on 70 ühiktesti, testide kattuvus on 71% ja testid läbitakse 8,57 sekundiga (Joonis 11).



Joonis 11. Ühiktestimise tulemused

5 Tulemused ja edasised tegevused

Käesolevas peatükis tutvustatakse lõputöö ärilist kasumit, tehakse järeldusi ja tuuakse välja võimalusi olemasoleva töö edasi arenduseks.

5.1 Äriline kasum

Lõputöö eesmärk oli luua uus tarkvaralahendus, mis võimaldab ettevõtte klientidel teha makseid ja arveldusi lihtsamalt ning tõhusamalt. Selleks oli vajalik integreerida olemasolev süsteem makseteenuse süsteemiga ühtseks tervikuks ning luua uus funktsionaalsus, mis võimaldaks klientidel teha makseid erinevate makseviisidega ning jälgida oma makse- ja arvelduste ajalugu. Lõputöö tulemusena õnnestus luua edukas tarkvara integratsioon, mis pakub klientidele palju kasu ning parendab nende kasutajakogemust.

Uus tarkvaralahendus pakub ettevõttele suurt ärilist kasu, kuna see aitab suurendada klientide rahulolu ja lojaalsust. Kliendid saavad nüüd teha makseid erinevate makseviisidega ning jälgida oma makse- ja arvelduste ajalugu ühes kohas, mis muudab nende elu palju lihtsamaks. Lisaks sellele võimaldab uus tarkvara ettevõttel säästa aega ja raha, kuna see vähendab vajadust manuaalsete maksete ja arvelduste järele ning vähendab vigade arvu. See omakorda parandab ettevõtte mainet ning suurendab klientide rahulolu ja usaldust.

Kokkuvõttes võib öelda, et uus tarkvaralahendus on väärt investeering, mis aitab ettevõttel saavutada edu pikemas perspektiivis. See pakub klientidele rohkem mugavust ning vähendab ettevõtte kulusid, mis omakorda aitab suurendada ettevõtte tulu. Lisaks sellele annab uus tarkvara ettevõttele konkurentsieelise ning võimaldab neil pakkuda klientidele paremat teenust. Uus maksete ja arveldamise tarkvara integratsioon on saanud ettevõtte juhtide täieliku heakskiidu ning klientide tagasiside on olnud äärmiselt positiivne, kuna see on oluliselt lihtsustanud maksete tegemist ja arvelduste jälgimist. Seega võib väita, et uus tarkvaralahendus on oluline ettevõtte kasvu ja arengu jaoks ning aitab kaasa nii klientide kui ka ettevõtte vajaduste rahuldamisele.

5.2 Võimalused edasi arenduseks

Maksmise ja arveldamise tarkvara integratsiooni edasi arendamine on oluline, et optimeerida ettevõtte tööprotsesse. Üks võimalus tarkvaralahenduse edasi arendamiseks on lisada funktsionaalsus tellimuste plaanide (edaspidi tooted) lisamiseks. Praegu lisatakse tooted otse andmebaasi, mis võib olla ebamugav ja aeganõudev. Uue funktsionaalsuse lisamine võimaldaks ettevõtte töötajatel mugavalt lisada tooteid ja hallata tellimusi.

Teine võimalus maksmise ja arveldamise tarkvara edasi arendamiseks on luua eraldi *Admin* vaade, kus ettevõtte haldur saaks ise tellimusi hallata. Praegu hallatakse kõike maksetega seonduvat Stripe keskkonnas, kus iga ettevõtte töötaja omab juurdepääsu kõigile andmetele. Tulevikus ei pruugi ettevõte soovida jagada kõigile töötajatele ligipääsu Stripe'is olevatele andmetele ning eraldi *Admin* vaade võimaldaks halduril hallata tellimusi ja makseid ilma juurdepääsuta Stripe süsteemile.

Lisaks nendele kahele võimalusele võib keskenduda rohkem kliendisuhtluse ja klienditeeninduse parendamisele, et pakkuda klientidele veelgi paremat teenust. Luua funktsionaalsus klientide tagasiside kogumise jaoks, et parendada mitte ainult maksetega seonduvat, vaid kogu rakendust tervikuna. Kõik need võimalused aitavad ettevõttel optimeerida oma tegevust ning parandada klientide kogemust, mis omakorda võib viia suuremate müügitulemusteni ja parema äriedu saavutamiseni.

6 Kokkuvõte

Playerbank OÜ ettevõttes puudus makselahendus, mis takistas ettevõttel pakkuda klientidele terviklikku lahendust ning teenida tulu oma teenuste eest. Selle töö eesmärk oli luua vajaolev integratsioon Playerbank OÜ ja maksete töötlemise ning arveldamise tarkvara teenust pakkuva süsteemi vahel.

Töö analüüsi käigus selgitati välja parim makseteenuste pakkuja tehniliste, äriliste, juriidiliste kui ka ettevõtte poolt seatud mõõdikute poolest. Tehti kindlaks, milline programmeerimiskeel on turvalisuse, keerukuse ja aktuaalsuse poolest kõige sobilikum antud lõputöö raames. Makseteenuste pakkuja ja programmeerimiskeele analüüsi tulemusena osutus valituks vastavalt Stripe teenusepakkuja ja Java programmeerimiskeel. Arenduskeskkonnana kasutati IntelliJ IDEA keskkonda ja koodihalduseks Bitbucket tarkvara.

Lõputöö praktilises osas arendati nõutele ja analüüsi tulemustele põhinev vastav integratsioon. Lahendati ära makseprobleem, mis on varasemalt ettevõtet takistanud tulu realiseerimast. Tulemusena olid lõputöö eesmärgid täies ulatuses saavutatud: maksete töötlemise ja arveldamise tarkvara olemasolu aitab kaasa ettevõtte kasvule ja jätkusuutlikkusele, samuti tehniliselt on olemasolevat arendust lihtsa vaevaga suudetud uute äriliste eesmärkide järgi edasi arendada.

Autor tänab Playerbank OÜ-d selle eest, et ettevõtte võimaldas autoril arendada end erialaselt ning täita lõputöö eesmärk. Autor soovib samuti tänada oma juhendajaid, kes teda suunasid ja toetasid kogu protsessi vältel.

Kasutatud kirjandus

[1] Forbes, „What Is A Third-Party Payment Processor?“ 6. Jaanuar 2023. [Võrgumaterjal]. Loetud aadressil: https://www.forbes.com/advisor/business/third-party-payment-processor/#when_to_use_a_third_party_payment_processor_section. [Kasutatud 20. märts 2023].

[2] BPC, „ What are Payment Processing Services,“ 11. märts 2021. [Võrgumaterjal]. Loetud aadressil: https://www.bpcbt.com/blog/what-is-payment-processing-service/. [Kasutatud 27. veebruar 2023].

[3] Stripe, „Learn about payment methods,“ [Võrgumaterjal]. Loetud aadressil: https://stripe.com/docs/payments/payment-methods/overview. [Kasutatud 17. märts 2023].

[4] Nerdwallet, „What Is Stripe, and How Does It Work to Accept Payments?“ 3. jaanuar 2023. [Võrgumaterjal]. Loetud aadressil: https://www.nerdwallet.com/article/small-business/what-is-stripe. [Kasutatud 25. veebruar 2023].

[5] Wired, „The untold story of Stripe, the secretive $20bn startup driving Apple, Amazon and Facebook,“ 5. oktoober 2010. [Võrgumaterjal]. Loetud aadressil: https://www.wired.co.uk/article/stripe-payments-apple-amazon-facebook. [Kasutatud 25. veebruar 2023].

[6] Stripe, „Use incoming webhooks to get real-time updates,“ [Võrgumaterjal]. Loetud aadressil: https://stripe.com/docs/webhooks. [Kasutatud 25. veebruar 2023].

[7] Golden, „Paddle (company),” [Võrgumaterjal]. Loetud aadressil: https://golden.com/wiki/Paddle_(company)-BWKNGAY. [Kasutatud 30. märts 2023].

[8] Paddle, „Merchant of record: A guide for CFOs,” [Võrgumaterjal]. Loetud aadressil: https://www.paddle.com/resources/merchant-of-record-a-guide-for-cfos. [Kasutatud 30. märts 2023].

[9] Paddle, „Webhook Reference,” [Võrgumaterjal]. Loetud aadressil: https://developer.paddle.com/webhook-reference/bd1986c817a40-webhook-reference. [Kasutatud 30. märts 2023].

[10] Fastspring, „A Detailed Comparison of Stripe vs. Paddle vs. FastSpring (With Reviews),“ [Võrgumaterjal]. Loetud aadressil: https://fastspring.com/blog/stripe-vs-paddle/. [Kasutatud 25. veebruar 2023].

[11] Indiehackers, „ Stripe vs Paddle,“ 21. veebruar 2021. [Võrgumaterjal]. Loetud aadressil: https://www.indiehackers.com/post/stripe-vs-paddle-89161b0d5c. [Kasutatud 26. veebruar 2023].

[12] Data Flair, „Pros and Cons of Java | Advantages and Disadvantages of Java,“ [Võrgumaterjal]. Loetud aadressil: https://data-flair.training/blogs/pros-and-cons-of-java/. [Kasutatud 6. märts 2023].

[13] Tech Fairy, „Java VS Python VS C# detailed comparison, which language to learn first?“ 12. Juuli 2020. [Võrgumaterjal]. Loetud aadressil: https://tech-fairy.com/java-vs-python-vs-c-detailed-comparison-which-language-to-learn-first/. [Kasutatud 6. märts 2023].

[14] Javatpoint, „Java JDBC Tutorial,“ [Võrgumaterjal]. Loetud aadressil: https://www.javatpoint.com/java-jdbc. [Kasutatud 6. märts 2023].

[15] Agilites, „Pros and Cons of Using C# as Your Backend Programming Language,“ 11. jaanuar 2016. [Võrgumaterjal]. Loetud aadressil: https://www.agilites.com/pros-and-cons-of-using-c-as-your-backend-programming-language.html. [Kasutatud 7. märts 2023].

[16] Altexsoft, „The Good and the Bad of C# Programming,“ 29. oktoober 2021. [Võrgumaterjal]. Loetud aadressil: https://www.altexsoft.com/blog/c-sharp-pros-and-cons/. [Kasutatud 7. märts 2023].

[17] O’Reilly, „Chapter 16. Database Connectivity,“ [Võrgumaterjal]. Loetud aadressil: https://www.oreilly.com/library/view/c-for-java/0735617791/ch16.html. [Kasutatud 7. märts 2023].

[18] Akamai, „The Pros and Cons of Python Programming,“ 6. jaanuar 2023. [Võrgumaterjal]. Loetud aadressil: https://www.linode.com/docs/guides/pros-and-cons-of-python/. [Kasutatud 8. märts 2023].

[19] PixelCrayons, „Pros and Cons of Python Programming Language,“ 6. jaanuar 2023. [Võrgumaterjal]. Loetud aadressil: https://www.pixelcrayons.com/blog/python-pros-and-cons/. [Kasutatud 8. märts 2023].

[20] PostgreSQL, „About,“ [Võrgumaterjal]. Loetud aadressil: https://www.postgresql.org/about/. [Kasutatud 10. märts 2023].

[21] Liquibase, „Accelerate Your Database Deployments,“ [Võrgumaterjal]. Loetud aadressil: https://www.liquibase.com/?_ga=2.12930366.806797454.1678430707-1469911029.1678430707. [Kasutatud 30. märts 2023].

[22] Liquibase, „How Liquibase Works,“ [Võrgumaterjal]. Loetud aadressil: https://www.liquibase.com/how-liquibase-works. [Kasutatud 30. märts 2023].

[23] martinFowler, „Domain Driven Design,“ [Võrgumaterjal]. Loetud aadressil: https://martinfowler.com/tags/domain%20driven%20design.html. [Kasutatud 26. aprill 2023].

[24] Oracle, „Record Classes,“ [Võrgumaterjal]. Loetud aadressil: https://docs.oracle.com/en/java/javase/16/language/records.html. [Kasutatud 26. aprill 2023].

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

Mina, Markus Maila

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Maksete töötlemise ja arveldamise tarkvara integratsioon Playerbank OÜ näitel" , mille juhendaja on Karl-Erik Karu
   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.

11.05.2023

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 – Resource_id triger fail

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

CREATE FUNCTION create_resource_id(prefix CHAR(5))
    RETURNS VARCHAR(21)
AS
$$
BEGIN
    IF prefix IS NULL OR prefix = ''
    THEN
        RAISE 'Prefix cannot be empty or null';
    ELSE
        RETURN prefix || '-' || SUBSTRING(
                ENCODE(sha256(DECODE(
                        REPLACE(gen_random_uuid()::TEXT, '-', ''),
                        'hex')), 'hex')
                FOR 15);
    END IF;
END
$$ LANGUAGE plpgsql;

CREATE FUNCTION trigger_api_resource_fields()
    RETURNS TRIGGER AS
$$
BEGIN
    IF (new.date_created IS DISTINCT FROM old.date_created) AND (tg_op !=
'INSERT') THEN
        RAISE EXCEPTION 'Do not mess up the date_created';
    ELSEIF new.date_modified IS DISTINCT FROM old.date_modified THEN
        RAISE EXCEPTION 'Do not mess up the date_modified';
    ELSEIF (new.resource_id IS DISTINCT FROM old.resource_id) AND (tg_op !=
'INSERT') THEN
        RAISE EXCEPTION 'Do not mess up the resource_id';
    ELSEIF (tg_op = 'INSERT') THEN
        new.date_created = CURRENT_TIMESTAMP;
        new.date_modified = CURRENT_TIMESTAMP;
        new.resource_id = create_resource_id(tg_argv[0]);
    ELSEIF (tg_op = 'UPDATE') THEN
        new.date_modified = CURRENT_TIMESTAMP;
    END IF;
    RETURN new;
```

```
END

$$ LANGUAGE 'plpgsql';
```

Lisa 3 – *Subscription* tabeli lisamise *changeset*

```
<changeSet id="subscription-table" author="markus.maila">
    <sql splitStatements="false">
            CREATE TABLE iam.subscription
            (
                id                          BIGINT       NOT NULL GENERATED BY
DEFAULT AS IDENTITY (START 10000) PRIMARY KEY,
                resource_id                 VARCHAR(21)  NOT NULL UNIQUE,
                provider_customer_id        VARCHAR(21)  NOT NULL,
                provider_subscription_id    VARCHAR      NOT NULL UNIQUE,
                provider_price_id           VARCHAR      NOT NULL,
                status                      VARCHAR(10)  NOT NULL,
                date_created                TIMESTAMP WITH TIME ZONE NULL,
                date_modified               TIMESTAMP WITH TIME ZONE NULL,
                provider_payment_method_id VARCHAR(100) NOT NULL
            );

            CREATE TRIGGER subscription_audit
                BEFORE INSERT OR
            UPDATE on iam.subscription FOR EACH ROW
                EXECUTE PROCEDURE
PUBLIC.trigger_api_resource_fields('subsc');
        </sql>
</changeSet>
```

Lisa 4 – *Plan* tabeli lisamise *changeset*

```
<changeSet id="plan-table" author="markus.maila">
    <sql splitStatements="false">
            CREATE TABLE iam.plan
            (
                id                  BIGINT         NOT NULL GENERATED BY
DEFAULT AS IDENTITY (START 10000) PRIMARY KEY,
                resource_id         VARCHAR(21)    NOT NULL UNIQUE,
                provider_product_id VARCHAR(255)   NOT NULL,
                plan_name           VARCHAR(255)   NOT NULL,
                provider_price_id   VARCHAR(255)   NOT NULL,
                currency            VARCHAR(3)     NOT NULL,
                price               NUMERIC(16, 2) NOT NULL,
                date_created        TIMESTAMP WITH TIME ZONE NULL,
                date_modified       TIMESTAMP WITH TIME ZONE NULL,
                period_type         VARCHAR(50)    NOT NULL CHECK
(period_type IN ('MONTHLY', 'YEARLY'))
            );

            CREATE TRIGGER plan_audit
                BEFORE INSERT OR
            UPDATE on iam.plan FOR EACH ROW
                EXECUTE PROCEDURE PUBLIC.trigger_api_resource_fields('plan');
        </sql>
</changeSet>
```

Lisa 5 – *Subscription* objekti vastuvõetavad andmed


```
{
  "id": "sub_1N145w2eZvKYlo2Cc3C9kjVw",
  "object": "subscription",
  "application": null,
  "application_fee_percent": null,
  "automatic_tax": {
    "enabled": false
  },
  "billing_cycle_anchor": 1682499180,
  "billing_thresholds": null,
  "cancel_at": null,
  "cancel_at_period_end": false,
  "canceled_at": null,
  "cancellation_details": {
    "comment": null,
    "feedback": null,
    "reason": null
  },
  "collection_method": "charge_automatically",
  "created": 1682499180,
  "currency": "usd",
  "current_period_end": 1685091180,
  "current_period_start": 1682499180,
  "customer": "cus_NmdM2kzIsg1HE8",
  "days_until_due": null,
  "default_payment_method": null,
  "default_source": null,
  "default_tax_rates": [],
  "description": null,
  "discount": null,
  "ended_at": null,
  "items": {
    "object": "list",
    "data": [
      {
        "id": "si_NmdM333VQ1TvcK",
        "object": "subscription_item",
        "billing_thresholds": null,
        "created": 1682499181,
        "metadata": {},
        "price": {
          "id": "plan_NhghU1SujcyX99",
          "object": "price",
```

```
        "active": true,
        "billing_scheme": "per_unit",
        "created": 1681358379,
        "currency": "usd",
        "custom_unit_amount": null,
        "livemode": false,
        "lookup_key": null,
        "metadata": {},
        "nickname": null,
        "product": "prod_NhghRAaWdUziIo",
        "recurring": {
          "aggregate_usage": null,
          "interval": "month",
          "interval_count": 1,
          "usage_type": "licensed"
        },
        "tax_behavior": "unspecified",
        "tiers_mode": null,
        "transform_quantity": null,
        "type": "recurring",
        "unit_amount": 1000,
        "unit_amount_decimal": "1000"
      },
      "quantity": 1,
      "subscription": "sub_1N145w2eZvKYlo2Cc3C9kjVw",
      "tax_rates": []
    }
  ],
  "has_more": false,
  "url": "/v1/subscription_items?subscription=sub_1N145w2eZvKYlo2Cc3C9kjVw"
},
"latest_invoice": "in_1N145w2eZvKYlo2CoDVh57FS",
"livemode": false,
"metadata": {
  "site_url": "https://coerver.draftserver.com/players-club",
  "platform": "MemberPress Connect acct_1FIIDhKEEWtO8ZWC",
  "ip_address": "27.34.31.146"
},
"next_pending_invoice_item_invoice": null,
"on_behalf_of": null,
"pause_collection": null,
"payment_settings": {
  "payment_method_options": null,
  "payment_method_types": [
    "card",
    "link"
  ],
  "save_default_payment_method": "on_subscription"
},
"pending_invoice_item_interval": null,
"pending_setup_intent": null,
```

```
  "pending_update": null,
  "schedule": null,
  "start_date": 1682499180,
  "status": "incomplete",
  "test_clock": null,
  "transfer_data": null,
  "trial_end": null,
  "trial_settings": {
    "end_behavior": {
      "missing_payment_method": "create_invoice"
    }
  },
  "trial_start": null
}
```