TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Siim Kilki 193749IADB

Kontoandmete sünkroniseerimine välise partneri süsteemiga AS-i LHV Pank näitel

Bakalaureusetöö

Juhendaja: Toomas Lepikult
PhD

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: Siim Kilki

29.12.2022

Annotatsioon

Tänapäeval on panganduse lahutamatuks osaks kiirmaksed. Maksete ootele jäämine on erandlik nähtus ja tihtipeale seotud sanktsioonide või kahtlaste tehingutega. Maksete ootele jäämist aitab ära hoida pankadevaheliste maksete saaja tuvastamine enne makse sooritamist.

Selliseks teenuseks Ühendkuningriigis on *Confirmation of Payee*, mille peab AS-i LHV Pank Ühendkuningriigi filiaal konkurentsis püsimiseks samuti kasutusele võtma. Käesoleva töö eesmärgiks oli luua toimiv ja automaatne kontoandmete sünkroniseerimise lahendus AS-i LHV Pank ja välise partneri vahel, mis koguks efektiivselt ja kiirelt kokku kõik vajalikud andmed ja edastaks need välise partneri süsteemile.

Rakendusliidese loomisel sai kasutatud parimaid võimalikke tehnikaid, mida etteantud nõuded võimaldasid. Kasutati enne faili genereerimist andmete vahetabelisse kogumise meetodit ning faili edastamiseks välise partneri failiserverisse kasutati JCrafti JSch SSH2 Java teostuse SFTP failiedastuse meetodeid. Toodangus genereeritakse vahetabelist fail ja saadetakse välise partneri failiserverisse keskmiselt ligikaudu 13 sekundiga.

Rakendusliides kaeti 100-protsendiliselt ühiktestidega ning testiti ka manuaalselt. Lisaks kinnitas ka välise partneri vastutav isik, et fail koos nõuetekohaste andmetega jõuab nende failiserverisse. Loodud lahendus ei ole ennast peale valmimist vigadega meelde tuletanud.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 6 peatükki, 11 joonist, 1 tabelit.

Abstract

Account Data Synchronisation with External Partner's System Using the Example of AS LHV Pank

Fast payments are essential part of modern banking. Situations, where payments have pending status, are rare occasions and often related to sanctions and fraudulent activity. Avoiding pending payments can be achieved by identifying of the recipient of interbank payments before the payment is accepted by bank.

Service for that kind of identifying in United Kingdom of Great Britain and Northern Ireland is called Confirmation of Payee, which must be implemented by AS LHV Pank for it's UK branch to stay competitive. The aim of this work was to create a functional and automatic account data synchronization solution between AS LHV Pank and an external partner, which would efficiently and quickly collect all the necessary data and transfer them to the external partner's system for the Confirmation of Payee service.

The best possible techniques were used in the creation of the backend application. Techniques were chosen by the given requirements. In the solution data was collected into an intermediate database table. While generating the *.tsv file, all the data was queried from the intermediate database table. When all the data written into the file, the SFTP file transfer methods of JCraft's JSch SSH2 Java implementation were used to transfer the file to the external partner's file server. In production, the file is generated and sent to the external partner's file server in an average of 12.619 seconds. Created backend works automatically and seamlessly.

The backend application was 100 percent covered by unit tests and also tested manually. In addition, the responsible person of the external partner also confirmed that the file with the proper data reaches their file server. The created solution has not reminded itself with errors after completion.

The thesis is in Estonian and contains 28 pages of text, 6 chapters, 11 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application programming interface*, rakendusliides
AS	Aktsiaselts
CDC	*Change-Data-Capture*, andmebaasitabelis andmemuudatuste fikseerimine edasisaatmiseks.
COP, cop	*Confirmation of Payee*, makse saaja nime kontrolli teenus, mille osa on ka välise partneri kontode sünkroniseerimise teenus pankadele
Domeenipõhine arendus	*Domain-Driven-Design*, arenduse põhimõtted, kus kood jaguneb kolmeks – liidese, domeeni ja infrastruktuuri kiht
Eksperimentaalarendus	*Proof-of-Concept*, lihtne arendus lahenduskäikude testimiseks või võrdlemiseks
Hoidla	*Repository,* rakenduste virtuaalne hoiukoht
IBAN	*International Bank Account Number*, rahvusvaheline pangakonto number, mis on rangelt üks ühele seotud pangakontoga
JPA	*Java Persistence API*, teenus objekt-relatsiooniliseks vastendamiseks
LHV UK	AS-i LHV Pank Ühendkuningriigi filiaal
Plaanur	*Scheduler*, programm, mis ajastab, algatab ja lõpetab tööteid
Püüdur	*Subscriber*, rakenduses publitseeritud sündmuste püüdja ja töötleja
SFTP	*Secure File Transfer Protocol*, turvaline failiedastusprotokoll, mis kasutab SSH-d
SQL	*Structured Query Language*, andmebaasi päringukeel
SSH	*Secure Shell*, protokollisari, mis võimaldab turvalist krüpteeritud kaugpöördust
*.tsv	*Tab-Separated Values file*, tekstipõhine failitüüp, kus tekstiobjektid on eraldatud tabeldusmärgiga
Tööde	*Job*, kasutaja määratletud töötlusüksus
Vastendamine	*Mapping*, objektidele või parameetritele vastete leidmine
VIBAN	*Virtual International Bank Account* Number, virtuaalne rahvusvaheline pangakonto number, mis erineb IBAN-ist selle poolest, et ühe pangakontoga võib olla seotud mitu erinevat VIBAN-it. VIBAN-il pole saldot, see kajastub koondkontol,

	mille alla VIBAN-id kuuluvad
Ümberlüliti	*Feature flag*, lüliti funktsionaalsuses sisse ja välja lülitamiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval on panganduse lahutamatuks osaks kiirmaksed. Maksete ootele jäämine on erandlik nähtus ja tihtipeale seotud sanktsioonide või kahtlaste tehingutega. Pangasiseste maksete saaja nime ja kontonumbri kontrolli on kasutatud juba aastaid. Ühendkuningriigis on alates 2020. aastast hakanud asutus nimega Pay.UK [5] pakkuma analoogset teenust pankadevaheliste maksete saaja tuvastamiseks enne makse sooritamist. Teenuse nimeks on *Confirmation of Payee*. Ühendkuningriigi turul tegutsev AS-i LHV Pank Ühendkuningriigi filiaal ehk LHV UK [1] peab konkurentsis püsimiseks samuti võtma kasutusele *Confirmation of Payee* teenuse.

Käesoleva töö teemaks on Kontoandmete sünkroniseerimine välise partneri süsteemiga AS-i LHV Pank näitel. Töö eesmärgiks on toimiv kontoandmete sünkroniseerimise lahendus AS-i LHV Pank ja välise partneri vahel, mis kogub efektiivselt vajalikud andmed erinevatest andmebaasi tabelitest kokku ja edastab need välise partneri süsteemile. Lõputöö käigus realiseeritud lahendus võetakse pangas kasutusele.

Käesoleva töö teises peatükis kirjeldatakse lahendatavat probleemi. Kontoandmete kokku kogumiseks ja andmete sünkroniseerimiseks parima lahenduse leidmiseks analüüsitakse kolme meetodit kolmandas peatükis. Analüüsi tulemusena valitakse välja üks meetod, mida kasutatakse jälgimissüsteemi rakendusliidese ehitamiseks. Lõputöö neljandas peatükis kirjeldatakse rakendusliideses kasutatud tehnoloogiat. Viiendas peatükis analüüsitakse töö tulemust, testimist, rakenduse kasutust ja võimalikke arenduskohti.

2 Probleemi kirjeldus

Teema on aktuaalne, sest enne kontoandmete sünkroniseerimise teenust jäid teistest kontoandmete sünkroniseerimise teenusega ühinenud Ühendkuningriigis tegutsevatest pankadest AS-i LHV Pank Ühendkuningriigi filiaali klientidele tehtavad maksed nende pankade maksesüsteemides ootele, kuna teenuse kaudu ei leitud vastavate kontonumbrite ja nimede vahelist seost. Teenusega ühinemine ja kontoandmete edastamine:

1. annab võimaluse teostada teistest pankadest makseid AS-i LHV Pank Ühendkuningriigi filiaali klientidele koheselt;

2. aitab ära hoida negatiivset ärimõju panga kliendile. Positiivse kasutajakogemusega kliendid on teenustega rohkem rahul ja on nõus teenuste ja toodete eest ka rohkem maksma.

Lähtetingimused on AS-i LHV Pank süsteemidest lähtuvad. Arendatav kontoandmete edastamise süsteem kasutab Java SpringBoot raamistikku ja Microsoft SQL andmebaasipäringukeelt. Kontoandmete edastamiseks kasutatakse Accounts-Products rakendust, kus asuvad kontode süsteemi abiteenused, mis pole panga põhifunktsioonide toimimiseks vältimatult vajalikud. Vajalikud kliendi- ja kontoandmete muudatused edastatakse Accounts-Products süsteemi muudetud andmete jäädvustamise ehk CDC (ingl k *Change-Data-Capture*) meetodit kasutades. Käesoleva probleemi lahendamiseks vajalikud andmed on seega Accounts-Products rakendusel olemas.

Samuti mängivad probleemi lahenduse leidmisel rolli välise partneri ette antud tingimused, et kontode edastus peab toimuma kasutades SFTP failiedastusprotokolli ja *.tsv failitüüpi. Kontode andmed peab turvaliselt edastama välise partneri SFTP serverisse vähemalt kord päevas, sest edastatud andmed võetakse kasutusele ühe korra ööpäevas öösiti. Partneri rakendus migreerib andmed failist nende andmebaasi.

Probleem lahendatakse kahes osas:

1. kontoandmete kogumine erinevatest konto ja kliendi andmeid sisaldavatest tabelitest ja nende andmete kirjutamine *.tsv faili;

2. kontoandmete edastamine välise partneri serverisse SFTP failiedastusprotokolli kasutades.

3 Kontoandmete kogumise ja faili kirjutamise meetodite analüüs

Antud peatükis analüüsitakse kaasaegseid andmete kogumise meetodeid. Analüüsitud meetodeid võrreldakse omavahel ning nende seast valitakse välja sobivaim meetod kontoandmete sünkroniseerimise rakendusliidese loomise jaoks.

3.1 Meetodi valiku nõuded

LHV Panga IT-süsteem toimib hajussüsteemi põhimõttel, mistõttu kogutakse klientide ja kontode andmeid panga eri rakenduste andmebaasidest.

3.1.1 Funktsionaalsed nõuded

Meetod peab vastama järgmistele funktsionaalsetele nõuetele:

- Meetodiga peab olema võimalik koguda andmeid mitmest andmebaasi tabelist.
- Andmete ühetaolisuse tagamiseks peab andmete muutumisel toimuma automaatne andmete uuendamine.
- Meetod peab toimima automaatselt rakenduse töötamise taustal.
- Meetod peab töötama Microsoft SQL andmebaasi päringukeeles.
- Meetodit peab olema võimalik lõimida Java programmeerimiskeele Spring Boot raamistikus töötava rakendusega.
- Meetod peab olema rakendatav erinevate andmetüüpide korral.

3.1.2 Mittefunktsionaalsed nõuded

Meetod peab vastama järgmistele mittefunktsionaalsetele nõuetele:

- Meetod ei tohi olla liiga aeglane.

- Meetod peab olema lihtsasti rakendatav ja muudetav.

3.2 Kolme meetodi valimine analüüsiks

Accounts-Products rakenduse andmebaasi ACCOUNTS ja VIBAN_ACCOUNTS tabelite andmed on peegeldus valitud andmetest kontode rakenduse vastavates tabelites, kus hoitakse kõige täpsemat seisu vastavalt kontode ja viban-kontode andmetest (viban erineb tavalisest kontost selle poolest, et ühe pangakontoga võib olla seotud mitu erinevat vibanit). Accounts-Products rakenduse andmebaasi CUSTOMERS tabeli andmed on samamoodi peegeldus valitud andmetest kliendiandmete rakenduse vastavast klientide tabelist. Accounts-Products rakenduse tabelitesse peegeldatakse ainult neid andmeid, mida rakendusel vaja läheb. Nii kontode kui kliendiandmete rakendustes töötab Debezium, mis orkestreerib valitud tabelite andmemuudatuste saatmist teistesse rakendustesse.

Accounts-Products rakenduse andmebaasi ACCOUNTS, CUSTOMERS ja VIBAN_ACCOUNTS tabelitest peab kokku koguma kliendi ja tema konto(de) andmed ning kirjutama *.tsv faili, mis saadetakse SFTP failiprotokolli kasutades välise partneri serverisse. Käesolevas töös analüüsitakse kolme meetodit kliendi ja tema konto(de) andmete kokku kogumiseks ja faili kandmiseks:

1. andmete kogumine faili genereerimise ajal;
2. andmete kogumine tabelisse enne faili genereerimist;
3. andmete muutmine alalises failis.

Meetodite võrdlemiseks loodi iga meetodi kohta eksperimentaalarendus (ingl k *proof-of-concept*), mille käigus testiti meetodite toimimise kiirust.

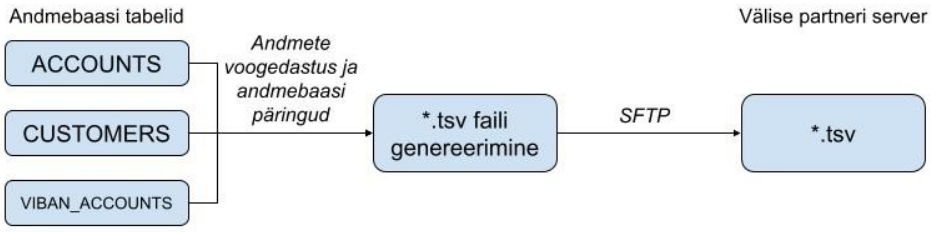
3.3 Meetodite võrdlemise metoodika

Selleks valitud kolme meetodit saaks omavahel võrrelda, on kõige olulisemaks nõudeks protsessi kiirus. Valitud meetodite võrdlemiseks mõõdeti ajalist vahet hetkest, mil algas andmete sisestamine Accounts-Products rakenduse andmebaasi ACCOUNTS, CUSTOMERS ja VIBAN_ACCOUNTS tabelitesse ning lõppes hetkel, mil oli valmis

saanud välisele partnerile edastamiseks mõeldud *.tsv fail. Testandmeteks valiti toodangus olevate andmete suurusjärgus 5000 kliendi kirjet CUSTOMERS tabeli jaoks, 10 000 kontode kirjet ACCOUNTS tabeli jaoks ja 70 000 viban-kontode kirjet VIBAN_ACCOUNTS tabeli jaoks. Testandmed loodi suvalise sisuga nii, et andmebaasi tabelite kohustusliku sisuga veerud saaks kindlasti täidetud.

3.4 Andmete kogumine faili genereerimise ajal

Antud meetodi puhul proovitakse *.tsv faili genereerida plaanuriga (ingl k *scheduler*, programm, mis ajastab, algatab ja lõpetab tööteid [2]) iga 15 minuti tagant. Genereerimiseks kasutatakse töödet (ingl k *job*, kasutaja määratletud töötlusüksus [3]), mille plaanur käivitab. Tööde loob faili. Seejärel pärib tööde LHV UK klientide kontode andmed ACCOUNTS tabelist voona ja kirjutab kontode andmed 1000 kaupa faili. Iga konto andmestiku puhul pärib plaanur kontoga seotud kasutaja andmed CUSTOMERS tabelist ja lisab need andmestikule enne faili lisamist. Peale kontode faili lisamist tehakse sama operatsioon ka VIBAN_ACCOUNTS tabelis olevate viban-kontodega, sealhulgas päritakse iga viban-konto jaoks CUSTOMERS tabelist seotud kliendi andmed. Kui fail on andmetega täidetud, saadab tööde faili välise partneri serverisse kasutades JCrafti JSch SSH2 Java teostuse SFTP failiedastuse meetodeid. Meetodi ülevaatlik skeem on toodud joonisel 1.



Joonis 1: Andmete liikumine faili genereerimise ajal andmete kogumise meetodit kasutades.

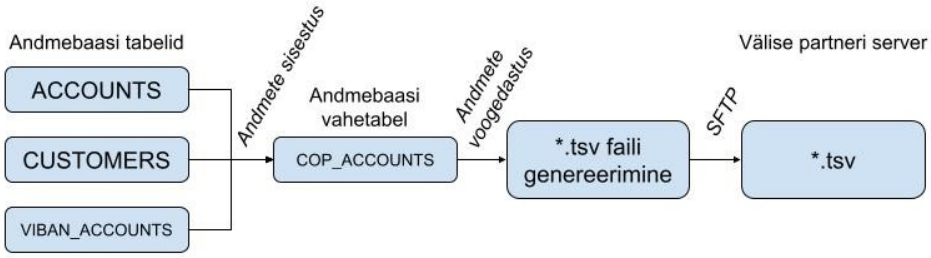
Antud meetodi eeliseks on andmete ajakohasus ja ühtsus, kuna andmed päritakse otse andmebaasi tabelitest faili genereerimise ajal.

Antud meetodi puuduseks on eksperimentaalarenduse käigus mõõdetud andmete sisestamise ja faili genereerimisele kokku kulunud aeg keskmiselt 5841 sekundit.

3.5 Andmete kogumine vahetabelisse enne faili genereerimist

Andmete peegeldamisel salvestatakse andmed vastavalt ACCOUNTS, CUSTOMERS või VIBAN_ACCOUNTS tabelitesse ning antud meetodi puhul avaldatakse iga andmete peegelduse peale Java Spring raamistiku *ApplicationEventPublisher* klassi *publishEvent* meetodit kasutades andmete uuendamise sündmus (ingl k *event*). Avaldatud sündmuse püüab rakenduses kinni sündmuse tüübile vastav *EventSubscriber* nimeline püüdur, mis kontrollib, kas COP_ACCOUNTS tabelis on vastavate andmetega uuendamist vajavaid kirjeid ning uuendab need. Kui kirjeid ei eksisteeri, siis need lisatakse juhul, kui klient, konto või viban-konto on LHV UK klient või tema konto. COP_ACCOUNTS tabel sisaldab kõiki välisele partnerile genereeritava failis vaja minevaid andmeid iga LHV UK kliendi konto ja viban-konto kohta ning lisaks ka konto või viban-konto identifikaatoreid.

*.tsv faili proovitakse genereerida plaanuriga iga 15 minuti tagant. Genereerimiseks kasutatakse töödet. Tööde loob faili, pärib andmed COP_ACCOUNTS tabelist voona ja kirjutab andmed faili 1000 kaupa ning seejärel saadab välise partneri serverisse kasutades JCrafti JSch SSH2 Java teostuse SFTP failiedastuse meetodeid. Meetodi ülevaatlik skeem on toodud joonisel 2.



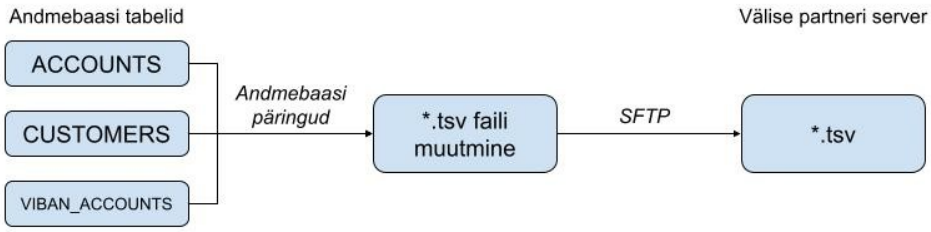
Joonis 2: Andmete liikumine enne faili genereerimist andmete vahetabelisse kogumise meetodit kasutades.

Antud meetodi eelised on eksperimentaalarenduse käigus saadud andmete sisestamisele ja faili genereerimisele kokku kulunud aeg keskmiselt 234 sekundit ja võimalus jämedalt COP_ACCOUNTS tabelist kontrollida, milline seis andmetest välisele partneritele edastatud on. Eeliseks on ka see, et faili genereerimise hetkel on COP_ACCOUNTS tabelis kogu andmestik olemas, mida faili genereerimiseks vaja läheb.

Antud meetodi puuduseks on lisatabeli vajadus andmebaasis, mis osaliselt dubleerib olemasolevates tabelites olevaid andmeid.

3.6 Andmete muutmine alalises failis

Antud meetodi puhul avaldatakse iga andmete peegelduse peale Java Spring raamistiku *ApplicationEventPublisher* klassi *publishEvent* meetodit kasutades andmete uuendamise sündmus. Avaldatud sündmuse püüab rakenduses kinni sündmuse tüübile vastav *EventSubscriber*, mis kontrollib, kas andmed on seotud mõne LHV UK kliendi või tema kontoga. Kui andmed on seotud LHV UK kliendi või tema kontoga, siis luuakse uus *.tsv fail, kuhu hakatakse olemasolevast failist kontode andmeid ümber kirjutama. Kui ümberkirjutamisel jõutakse muutmist vajava konto andmestikuni, siis kirjutatakse uude faili uuendatud kliendi või konto andmed. Kui kirjet ei eksisteeri, siis lisatakse uus konto andmestik faili lõppu. Meetodi ülevaatlik skeem on toodud joonisel 3.



Joonis 3: Andmete liikumine alalises failis andmete muutmise meetodit kasutades.

Antud meetodi eeliseks on faili alatine olemasolu välise partneri serverisse saatmiseks.

Antud meetodi puuduseks on vajadus hoida alati vähemalt ühte *.tsv faili rakenduse failipuus. Faili muutmise ajal on vaja ka teist faili, sest faili ei saa samal ajal lugeda ja kirjutada, kus ühest loetakse failiridu ja teise kirjutatakse. Meetodi puuduseks on ka faili muutmise aeglus suuremahulise andmete muutmise koormuse ajal. Samuti muutub faili muutmine aeglaseks, kui failis olevate ridade arv kasvab väga suureks, sest iga andmemuudatuse korral töödeldakse läbi kõik faili read. Meetodi arendamise keerukus lisab vigade tekkimise ohtu.

Antud meetodi tõsiseks puuduseks on eksperimentaalarenduse käigus mõõdetud andmete sisestamise ja faili genereerimisele kokku kulunud aeg keskmiselt 32 282 sekundit.

3.7 Meetodite võrdlus

Faili genereerimise ajal andmete kogumise meetod ja enne faili genereerimist andmete vahetabelisse kogumise meetod on muus osas nõuete täitmises võrdsed, aga testandmete faili kirjutamise aja poolest on enne faili genereerimist andmete vahetabelisse kogumise meetod selgelt parem teistest meetoditest (tabel 1).

Tabel 1: Kirjeldatud meetodite nõuete täitmise võrdlus.

Nõue	Andmete kogumine faili genereerimise ajal	Andmete kogumine vahetabelisse enne faili genereerimist	Andmete muutmine alalises failis
Meetodiga peab olema võimalik koguda andmeid mitmest andmebaasi tabelist.	Jah	Jah	Jah
Andmete ühetaolisuse tagamiseks peab andmete muutumisel toimuma automaatne andmete uuendamine.	Jah	Jah	Jah
Meetod peab toimima automaatselt rakenduse töötamise taustal.	Jah	Jah	Jah
Meetod peab töötama Microsoft SQL andmebaasi päringukeeles.	Jah	Jah	Jah
Meetodit peab olema võimalik lõimida Java programmeerimiske ele Spring Boot raamistikus töötava rakendusega.	Jah	Jah	Jah
Meetod peab olema rakendatav erinevate andmetüüpide korral.	Jah	Jah	Jah

Testandmete *.tsv faili kirjutamise aeg sekundites.	5841 s	234 s	32 282 s
Meetod peab olema lihtsasti rakendatav ja muudetav.	Jah	Jah	Ei

3.8 Sobivaima meetodi valiku põhjendus

Analüüsi põhjal valiti probleemi lahendamiseks enne faili genereerimist andmete vahetabelisse kogumise meetod ennekõike selle tõttu, et see oli vaatlusalustest meetoditest selgelt kõige efektiivsem (sisestas 10 000 konto ja 70 000 viban-konto kirjet 234 sekundiga kontoandmete faili).

4 Rakendusliidese realiseerimine

Käesolevas peatükis kirjeldatakse nõudeid rakendusliidesele ning kirjeldatakse rakendusliidese ülesehitust.

4.1 Nõuded rakendusliidesele

Rakendusliides peab vastama järgmistele funktsionaalsetele nõuetele:

- Rakendus peab lugema andmed kliendi ja konto andmeid sisaldavatest ACCOUNTS, CUSTOMERS ja VIBAN_ACCOUNTS tabelitest, sest need andmed on rakenduse olemasoleva arhitektuuri kohaselt asuvad andmed vastavates tabelites.
- Rakendus peab looma välise partneri nõuetele vastava ülesehitusega *.tsv faili.
- Genereeritud failis peavad eksisteerima ainult LHV UK aktiivsete klientide aktiivsete kontode andmed, mis tuleneb välise partneri nõuetest failile.
- Rakendus peab saatma genereeritud faili SFTP failiedastusprotokolli kasutades välise partneri serverisse, sest SFTP kasutamine tuleneb välise partneri nõuetest faili edastamisele.
- Rakendus peab välise partneri serverist vananenud failid eemaldama, mis tuleneb välise partneri nõudest pankadel ise hallata failiserveris oma panga kausta sisu ja hoida seal ainult kõige värskemate andmetega fail.

Rakendusliides peab vastama järgmistele mittefunktsionaalsetele nõuetele:

- Faili genereerimine peab olema kiire ja kasutama riistvaralist ressurssi efektiivselt. Efektiivsuse nõue tuleneb panga rakenduste kasutamise eripärast, kus teatud ajahetkedel (näiteks tööpäevade lõppudes, intresside väljamakse kuupäevadel ja enamlevinud palgamakse kuupäevadel) on pangasüsteemide

kasutuse intensiivsus väga suur ja kui nendel hetkedel ka taustaprotsessid võtaks väga palju arvutusmahtu, siis muutuks panga süsteemid väga aeglaseks või kõige äärmuslikumal juhul lakkaks töötamast.

- Rakendus peab jälgima *Domain-Driven-Design* ehk domeenipõhise arenduse põhimõtteid [4] , sest rakendust arendav ja haldav tiim on võtnud eesmärgiks neid põhimõtteid jälgida, et rakenduse kood oleks paremini loetav, arusaadavam ja ühtlase struktuuriga nii tiimi praegustele liikmetele kui ka tulevastele liikmetele, kellel siis on sisseelamine kergem.

4.2 Kasutatavad tehnoloogiad

Loodud rakendusliides on kirjutatud Java programmeerimiskeeles, mis on objekt-orienteeritud programmeerimiskeel [8] . Java programmeerimiskeel oli 2022. aastal populaarsuselt kuues [15] programmeerimiskeel maailmas. Rakendusliides on loodud Java programmeerimiskeeles, sest rakenduse arendamise eest vastutav tiim on võtnud just selle keele enda kasutatavaks programmeerimiskeeleks. Ühe tiimi sees ühe arenduskeele kasutamine muudab arendust lihtsamaks ja kiiremaks ning võimaldab arendajatel kiiremini saavutada selles keeles meistertaseme [4] . Lisaks kasutatakse R. C. Martini [6] soovitatud ühtse vorminduse reeglit, et kood oleks ühtlase tasemega ja hea loetavusega.

Rakendusliidese loomisel kasutati Spring Boot raamistikku, mis lihtsustab Java rakenduste loomist [14] . Andmebaasiga suhtlemiseks kasutati Spring Data JPA [13] , mis võimaldab lihtsalt kasutada JPA-põhiseid hoidlaid (ingl k *repository*). JPA [9] on akronüüm ingliskeelsest terminist *Java Persistence API*, mis on Java teenus objekt-relatsiooniliseks vastendamiseks. Spring Boot ja Spring Data JPA on vastutavas tiimis kasutusele võetud Java programmeerimiskeeles arendamise ja andmebaasiga suhtlemise lihtsustamiseks.

4.3 Protsessi kirjeldus

Käesolevas peatükis on kirjeldatud kontoandmete kogumise ja välisele partnerile saatmise protsessi.

4.3.1 Andmete kandmine vahetabelisse

Andmete uuenemisel Accounts-Products rakendusse ACCOUNTS, CUSTOMERS ja VIBAN_ACCOUNTS tabelites kutsutakse esile Java Spring raamistiku *ApplicationEventPublisher* klassi *publishEvent* meetodit kasutades andmete uuendamise sündmus vastavalt *AccountDataUpdatedEvent*, *CustomerDataUpdatedEvent* või *VibanAccountDataUpdatedEvent* klassi sündmusena. Kõik kolm nimetatud klassi laiendavad *DomainEvent* klassi ja omavad lisaks *DomainEvent* klassist päritud *OffsetDateTime* tüüpi *eventDtime* parameetrile ka vastavalt *Account*, *Customer* ja *VibanAccount* tüüpi objekte.

Kontoandmete muutmisel avaldatud *AccountDataUpdatedEvent*'i püüavad kinni erinevad püüdurid (ingl *subscriber*). Üheks neist on antud töö raames arendatud *CopAccountDataUpdatedEventSubscriber* nimeline püüdur, millel on kaks meetodit – *getEventType*() ja *onDomainEvent*(). *getEventType*() meetod tagastab *AccountDataUpdatedEvent* klassi tüübi. *onDomainEvent*() (joonis 4) saab sisendiks sündmuse ja teostab vajalikud toimingud vastavate cop-kontode andmete uuendamiseks.

```
@Override
@Transactional
public void onDomainEvent(AccountDataUpdatedEvent event) {
   var account = event.getAccount();
   if (!account.isUkAccount() || account.isVirtual()) {
     return;
   }
   var copAccount =
copAccountRepository.findCopAccountByAccountId(account.getAcc
ountId());
   if (account.isActive() && copAccount.isEmpty()) {
     var newCopAccount = new CopAccount(account);
     copAccountRepository.save(newCopAccount);
   }
   if (!account.isActive() && copAccount.isPresent()) {
     var vibanStream =
vibanAccountService.streamAllUnclosedVibanAccountsByMasterAcc
ountId(account.getAccountId());
     if (vibanStream.findAny().isEmpty()) {
       copAccountRepository.delete(copAccount.get());
     }
   }
}
```

Joonis 4. COP_ACCOUNTS tabeli andmete muutmine konto andmete uuendamisel.

Sündmusest küsitakse *Account* tüüpi konto objekt. Järgneb kontroll, kas konto ei ole LHV UK subjekt või konto on virtuaalkonto. Kui kontroll vastab tõele, siis lõpetatakse protsess. Vastasel juhul jätkatakse. Edasi päritakse *CopAccountRepository* klassi *findCopAccountById*() meetodit kasutades COP_ACCOUNTS tabelist kirje konto identifikaatori järgi. Meetod tagastab *Optional<CopAccount>* tüüpi objekti, mis sisaldab *CopAccount* tüüpi objekti, edaspidi cop-konto, või on tühi (*Optional.empty*()).

Järgneb tingimuslause, kas konto on aktiivne ja cop-konto on tühi objekt, millega kontrollitakse, kas kontot pole COP_ACCOUNTS tabelis. Kui see vastab tõele, siis luuakse uus cop-konto vastendades konto andmed cop-kontoks *CopAccount* klassi *adjustDataForAccount*() meetodiga ja salvestatakse see COP_ACCOUNTS tabelisse.

COP_ACCOUNTS tabel (joonis 5) omab veergusid kõigi cop-kontode failis vajaminevate andmete (identification, account_type, name, secondary_identification, scheme_name, is_supported, is_cop_opt_out, is_switched, is_collection_account, acronym, name_alternative), auditeerimiseks vajaminevate andmete (created_dtime, created_by, modified_dtime, modified_by), kirje identifikaatori, kirjega seotud konto identifikaatori ja kirjega seotud viban-konto identifikaatori salvestamiseks. Tabeli

account_id ja viban_id veergudele on lisatud indeksid kirjete kiiremaks pärimiseks nende veergude andmete järgi.

```
CREATE TABLE cop_accounts
(
   copa_id BIGINT IDENTITY NOT NULL,
   account_id BIGINT NOT NULL,
   viban_id BIGINT NULL,
   identification VARCHAR(14) NOT NULL,
   account_type VARCHAR(8) NOT NULL,
   name VARCHAR(140) NOT NULL,
   secondary_identification VARCHAR(140) NOT NULL,
   scheme_name VARCHAR(21) NOT NULL,
   is_supported BIT NOT NULL,
   is_cop_opt_out BIT NOT NULL,
   is_switched BIT NOT NULL,
   is_collection_account BIT NOT NULL,
   acronym VARCHAR(8) NULL,
   name_alternative VARCHAR(140) NULL,
   created_dtime DATETIMEOFFSET NULL,
   created_by VARCHAR(30) NULL,
   modified_dtime DATETIMEOFFSET NULL,
   modified_by VARCHAR(30) NULL,
   CONSTRAINT PK_COPA PRIMARY KEY (copa_id)
)
```

Joonis 5. COP_ACCOUNTS tabeli loomise skript.

Järgmisena kontrollitakse tingimuslausega, kas konto on inaktiivne ja kas cop-konto ei ole tühi objekt. Tingimuse tõele vastavuse korral kustutatakse cop-konto COP_ACCOUNTS tabelist. Et cop-kontot saaks sealt kustutada, peab enne kontrollima, ega ei eksisteeri selliseid viban-kontosid, mille *Account* tüüpi põhikonto (ingl *master account*) suletav konto oleks, sest viban-kontol peab olema alati aktiivne põhikonto. Selleks päritakse konto identifikaatori järgi andmebaasist kõik selle kontoga seotud viban-kontod ja suletakse ka need.

Antud töö raames arendati ka *CopCustomerUpdatedEventSubscriber* nimeline püüdur, mis püüab kinni kliendi andmete muutumisel publitseeritud *CustomerDataUpdatedEvent* tüüpi sündmused. Sellel püüduril on samuti kaks meetodit – *getEventType*() ja *onDomainEvent*(). *getEventType*() meetod tagastab *CustomerDataUpdatedEvent* klassi tüübi. *OnDomainEvent*() (joonis 6) saab sisendiks *CustomerDataUpdatedEvent* tüüpi sündmuse ja pärib kõigepealt sündmusest *Customer* tüüpi objekti ja sealt selle identifikaatori. Identifikaatori alusel päritakse *AccountService*

klassi *getCustomerAccountsUnderBankLegalEntity*() meetodit kasutades kõik kliendiga seotud LHV UK kontod.

```
@Override
@Transactional
public void onDomainEvent(CustomerDataUpdatedEvent event) {
   var customerId = event.getCustomer().getCustomerId();
   accountService
    .getCustomerAccountsUnderBankLegalEntity(customerId,
LHV_BANK_UK_BRANCH)
    .forEach(this::updateCopAccountNamesFor);
}
```

Joonis 6. COP_ACCOUNTS tabeli andmete muutmine kliendi andmete muutmisel.

Iga saadud konto on sisendiks *updateCopAccountNamesFor*() meetodile. *updateCopAccountNamesFor*() meetodis päritakse kõigepealt konto identifikaatori järgi *Optional<CopAccount>* tüüpi objektis cop-konto. Kui päritud objekt ei sisalda cop-kontot ehk on tühi objekt (*Optional.empty*()), siis katkestatakse protsess.

updateCopAccountNamesFor() meetodis kasutatakse konto pärimiseks *findCopAccountByAccountId*() meetodit, kus päritakse *CopAccountJpaEntityRepository* klassi *findCopAccountJpaEntityByAccountIdAndVibanId*() meetodiga andmebaasist *CopAccountJpaEntity* tüüpi cop-konto, mis konverteeritakse *CopAccountJpaEntityMapper* klassi *toDomain*() meetodiga *Optional<CopAccount>* tüüpi cop-kontoks.

Kui cop-konto eksisteerib, siis päritakse konto objektist *Customer* tüüpi kliendi objekt. Kui klient on eraklient, asendatakse cop-konto *name* muutuja väärtus kliendi uue ees- ja perekonnanime kombinatsiooniga. Kui klient on äriklient, asendatakse cop-konto *name* muutuja väärtus kliendi uue ärinimega. Kui ärikliendi puhul on konto mõne viban-konto põhikontoks, siis nende COP_ACCOUNTS tabelis olevate viban-kontode andmetes asendatakse *nameAlternative* väärtus kliendi uue ärinimega (joonis 7). Järgnevalt kontrollitakse nii äri- kui erakliendi puhul, et cop-konto muudetud nimede väärtus oleks kuni 140 tähemärki. 140 tähemärki ületav osa lõigatakse ära. Lõpuks cop-konto salvestatakse COP_ACCOUNTS tabelisse.

```
@Modifying
@Query("UPDATE CopAccountJpaEntity c SET c.nameAlternative
= ?2 WHERE c.accountId = ?1 AND c.vibanId IS NOT NULL")
void updateAllVibanAccountNameAlternatives(Long id, String
name);
```

Joonis 7. Cop-kontode *nameAlternative* väärtuse uuendamine.

Antud töö raames arendati välja ka *CopVibanAccountDataUpdatedEventSubscriber* nimeline püüdur, mis püüab kinni viban-kontode andmete muutumisel publitseeritavad *VibanAccountUpdatedEventEvent* tüüpi sündmused. Sellel püüduril on samuti kaks meetodit, nagu teistel püüduritel – *getEventType*() ja *onDomainEvent*(). *getEventType*() meetod tagastab *VibanAccountDataUpdatedEvent* klassi tüübi. *onDomainEvent*() saab sisendiks *VibanAccountUpdatedEvent* tüüpi sündmuse. *onDomainEvent*() (joonis 8) pärib kõigepealt sündmusest *VibanAccount* tüüpi viban-konto objekti.

```
public void onDomainEvent(VibanAccountDataUpdatedEvent event)
{
   var viban = event.getVibanAccount();
   if (viban.isUkViban() && viban.getType() ==
VibanAccountType.VIBAN) {
     var optionalCopAccount =
copAccountRepository.findCopAccountByVibanIdWithLock(viban.ge
tVibanId());
     if (optionalCopAccount.isEmpty() &&
Arrays.asList(ALLOWED_VIBAN_STATUSES_FOR_COP).contains(viban.
getStatus())) {
       var newCopAccount = new CopAccount();
       newCopAccount.adjustDataForViban(viban);
       copAccountRepository.save(newCopAccount);
       bottomlineService.addAccountIntraday(newCopAccount);
     }
     if (optionalCopAccount.isPresent() &&
Arrays.asList(ALLOWED_VIBAN_STATUSES_FOR_COP).contains(viban.
getStatus())) {
       var copAccount = optionalCopAccount.get();
       copAccount.adjustDataForViban(viban);
       copAccountRepository.save(copAccount);
     }
     if (optionalCopAccount.isPresent() && !
Arrays.asList(ALLOWED_VIBAN_STATUSES_FOR_COP).contains(viban.
getStatus())){
       copAccountRepository.delete(optionalCopAccount.get());
     }
   }
}
```

Joonis 8. COP_ACCOUNTS tabeli andmete muutmine viban-konto andmete muutmisel.

Järgneb tingimuslause, kas viban-konto on LHV UK subjekt ja kas viban-konto tüüp on *VIBAN*. Viban-kontod võivad olla ka *AGENCY* tüüpi, kuid need ei kuulu cop-kontode määratluse alla. Kui tingimuslause vastab tõele, siis päritakse *CopAccountRepository* klassi *findCopAccountByVibanIdWithLock*() meetodit kasutades COP_ACCOUNTS tabelist kirje viban-konto identifikaatori järgi. Kui cop-kontot ei eksisteeri ja viban-konto on lubatud staatusega (aktiivne, blokeeritud või ootel), siis vastendatakse *adjustDataForViban*() meetodit kasutades viban-konto andmed cop-kontoks ning salvestatakse COP_ACCOUNTS tabelisse.

Kui *findCopAccountByVibanIdWithLock*() meetodiga päritud cop-konto eksisteerib ja viban-konto on lubatud staatusega, siis kirjutatakse cop-konto andmed üle vastavate uute viban-konto andmetega ning salvestatakse COP_ACCOUNTS tabelisse. Kui *findCopAccountByVibanIdWithLock*() meetodiga päritud cop-konto eksisteerib ja viban-konto ei ole ühegi lubatud staatusega, siis kustutatakse cop-konto kirje COP_ACCOUNT tabelist.

4.3.2 Vahetabeli esialgne andmemigratsioon

Peale peatükis 4.3.1 kirjeldatud funktsionaalsuste arendamist täideti vahetabel LHV UK subjektide andmetega. Selleks kasutati kahte andmebaasi päringu skripti.

Esimese skriptiga (lisa 2) sisestati COP_ACCOUNTS tabelisse LHV UK klientide kõigi nende **kontode** andmed, mida peatükis 4.3.1 kirjeldatud funktsionaalsused juba COP_ACCOUNTS tabelisse ise lisanud ei olnud.

Teise skriptiga (lisa 3) sisestati COP_ACCOUNTS tabelisse LHV UK klientide kõigi nende **viban-kontode** andmed, mida peatükis 4.3.1 kirjeldatud funktsionaalsused juba COP_ACCOUNTS tabelisse ise lisanud ei olnud.

4.3.3 Faili genereerimine

Faili genereerimiseks kasutatakse neli korda tunnis käivituvat töödet *sendCopAccountsToPartnerJob*() (joonis 9). Tööte esimese sammuna luuakse *WriteCopAccountsToFileCommand* tüüpi käsk, mille muutuja *syncDtime* väärtustatakse loomise hetkel *OffsetDateTime* klassi *now*() funktsiooniga. Järgnevalt kasutatakse *.tsv faili koostamiseks *WriteCopAccountsToFileUseCase* klassi *execute*() meetodit.

```
@QuartzScheduled(cronProperty = "0 0/15 * * * ?", auditUser =
SYSTEM_USER_ID)
public void sendCopAccountsToPartnerJob() {
   try {
     var command = new
WriteCopAccountsToFileCommand(OffsetDateTime.now());
     var state =
writeCopAccountsToFileUseCase.execute(command);
     sendCopAccountsToBottomLineUseCase.execute(state);
   } catch (RuntimeException ex) {
     log.error("Failed to send CopAccounts to Bottomline",
ex);
   }
}
```

Joonis 9. Faili genereerimise ja saatmise tööde.

execute() meetod saab sisendiks eelpool loodud käsu. Meetod algab tingimuslausega, kus *WriteCopAccountsToFileUseCaseSteps* klassi *shouldExecute*() meetodiga valideeritakse käsk ja otsustatakse, kas faili peab genereerima või mitte. Fail genereeritakse ainult siis, kui sama tunni sees pole faili loodud või on see ebaõnnestunud.

Kui on vajadus faili genereerida, siis kutsutakse esile *WriteCopAccountsToFileUseCaseSteps* klassi *createPendingFileState*() meetod, mis saab sisendiks käsu ja mille väljundiks on *CopAccountFileState* tüüpi objekt, mis omab parameetritena korrektset faili nime ja faili staatust *PENDING* ehk ootel. *CopAccountFileState* tüüpi objekt salvestatakse ka andmebaasi COP_ACCOUNTS_FILES tabelisse.

Sellele järgneb *WriteCopAccountsToFileUseCaseSteps* klassi *writeCopAccountsToFile*() meetod, mille sisendiks on eelnevalt loodud ootel staatuses *CopAccountFileState* tüüpi objekt. *writeCopAccountsToFile*() meetod kasutab *BottomlineFileRepository* klassi *writeFileFromAccountStream*() meetodit, mille sisendiks on eelnevalt loodud ootel staatuses *CopAccountFileState* tüüpi objekt ja voogedastusena päritult kõik COP_ACCOUNTS tabeli kirjed, mida päritakse omakorda *CopAccountRepository* klassi *streamCopAccountSyncFileEntries*() meetodiga. *streamCopAccountSyncFileEntries*() meetodis kasutatakse cop-kontode andmebaasist pärimiseks *CopAccountJpaEntityRepository* klassi *streamAll*() meetodit (joonis 10).

```
@Query("SELECT c FROM CopAccountJpaEntity c")
@QueryHints(value = @QueryHint(name = HINT_FETCH_SIZE, value
= "1000"))
Stream<CopAccountJpaEntity> streamAll();
```

Joonis 10. Kõigi cop-kontode pärimine andmebaasist.

writeFileFromAccountStream() meetodis luuakse füüsiline fail virtuaalmasinasse, kus rakendus töötab. Kui samanimeline fail juba eksisteerib, siis protsess katkestatakse. Kui samanimelist faili ei eksisteeri, siis luuakse fail *FileWriter* klassi abil. Kõik COP_ACCOUNTS tabeli read kirjutatakse loodud faili. Kui faili kirjutamine ebaõnnestub, genereerib rakendus *BottomlineServiceException* nimelise erindi, mis sisaldab ebaõnnestumise sõnumit faili nimega – „*Failed to create file* {{faili nimi}}”.

Kui faili loomine õnnestub, siis eelnevalt loodud ootel staatuses *CopAccountFileState* tüüpi objekti staatus muudetakse *WriteCopAccountsToFileUseCaseSteps* klassi *markFileAsReady*() meetodiga staatusesse *READY* ehk valmis saatmiseks ja väärtustatakse muutuja *generatedDTime* hetke ajalise väärtusega *OffsetDateTime.now*() meetodit kasutades. *CopAccountFileState* tüüpi objekt antakse sisendiks *SendCopAccountsToBottomlineUseCase* klassi *execute*() meetodile, mida kirjeldatakse peatükis 4.3.4.

4.3.4 Faili saatmine välise partneri serverisse

Eelnevas peatükis kirjeldati *CopAccountsSyncFileJob* tööte *SendCopAccountsToPartnerJob*() meetodi (joonis 9) esimest poolt, mis kirjutas kõik COP_ACCOUNTS tabeli kirjed *.tsv tüüpi faili ja tagastas *CopAccountFileState* tüüpi objekti faili nime ja staatusega. Järgnevalt kasutab tagastatud objekti *SendCopAccountsToBottomlineUseCase* klassi *execute*() meetod, mille eesmärk on loodud fail saata välise partneri serverisse.

execute() meetodis kasutatakse kõigepealt *SendAccountsToBottomlineUseCaseSteps* klassi *validate*() meetodit, mis saab sisendiks *CopAccountFileState* tüüpi objekti. Selle meetodiga kontrollitakse, et kas objekt on tühi objekt (*null*). Kui jah, siis päritakse andmebaasi COP_ACCOUNTS_FILES tabelist viimati loodud *CopAccountFileState* objekt. *validate*() meetod tagastab valideeritud *CopAccountFileState* tüüpi objekti.

Valideeritud objekt on sisendiks *SendAccountsToBottomlineUseCaseSteps* klassi *shouldExecute*() meetodile, mis kontrollib veelkord üle, et objekt poleks tühi. Lisaks

kontrollib see meetod ka, et objekt oleks staatuses *READY* ehk valmis saatmiseks ning et *SEND_COP_FILE* nimeline ümberlüliti (ingl k *feature flag*) oleks sisse lülitatud. Kui tingimus vastab tõele, siis jätkatakse protsessiga, kui ei vasta tõele, siis katkestatakse protsess. Ümberlüliteid kontrollib toodangukeskkonnas AS-i LHV Pank kontode tootejuht.

Jätkamise korral kutsutakse välja *SendAccountsToBottomlineUseCaseSteps* klassi *uploadFileToBottomlineServer*() meetod, mis saab sisendiks *CopAccountFileState* tüüpi objekti ja leiab selle põhjal rakenduse virtuaalmasinast genereeritud faili ning saadab selle välise partneri serverisse. Peale faili edukat saatmist kustutatakse partneri serverist kõik vanemad failid, kui just üles laetud fail kasutades meetodit *deleteOutdatedFilesFromBottomlineServerExcept*().

Saatmiseks luuakse *BottomlineSFTPService* klassi *uploadFile*() meetodis (joonis 11) sessioon, sessioonis avatakse SFTP kanal, kanali kaudu kopeeritakse fail välise partneri serverisse. Peale faili kopeerimist suletakse kanal ja ühendatakse sessioon lahti.

```
public void uploadFile(String fileName) {
   var session = createSession();
   var sftpChannel = createSftpChannel(session);
   try {
     sftpChannel.put(sftpConfiguration.getSourceDirectory() +
fileName,
     sftpConfiguration.getRemoteDirectory() + fileName);
   } catch (SftpException ex) {
     throw new
BottomlineServiceException(String.format("Failed to upload
file '%s' to Bottomline SFTP.", fileName), ex);
   } finally {
     sftpChannel.exit();
     session.disconnect();
   }
}
```

Joonis 11. Faili laadimine välise partneri serverisse.

Järgmiseks väärtustatakse *SendAccountsToBottomlineUseCaseSteps* klassi *markFileAsSent*() meetodiga *CopAccountFileState* tüüpi objekti *sentDTime* muutuja *OffsetDateTime.now*() meetodit kasutades. Staatuseks määratakse *SENT* ehk saadetud ja salvestatakse COP_ACCOUNTS_FILES tabelisse.

Virtuaalmasinast, kus rakendus töötab, kustutatakse genereeritud fail. Kui mingil põhjusel faili edastamine ebaõnnestub, kirjutab rakendus logisse ERROR tasemel sõnumi toimingu ebaõnnestumisest.

5 Tulemuste analüüs

Antud töö käigus loodud rakendusliides töötab automaatselt ööpäevaringselt. Rakendusliides täidab kõiki püstitatud nõudeid. Välise partneri esindajad on kinnitanud, et LHV UK klientide kontode ja viban-kontode andmed jõuavad ettenähtud vormis partneri serverisse nii testrakenduse kui ka toodangurakenduse puhul. Rakendusliides kasutab meetodite analüüsis selgelt teistest parema töötlemiskiirusega enne faili genereerimist andmete vahetabelisse kogumise meetodit.

Loodud funktsionaalsus käivitub iga 15 minuti tagant. LHV UK klientide kontode ja viban-kontode andmed saadetakse välise partneri serverisse sealjuures ainult kord tunnis. Ülejäänud kolm korda igas tunnis on loodud juhuks, kui eelnev faili genereerimine ja saatmine peaks ebaõnnestuma. Kuna genereeritava faili nimi oleks sama tunni sees alati sama, siis on keeruline saavutada olukorda, kus faili saadetakse mitu korda tunnis ja fail on samas alati kettal olemas. Seetõttu on ka lahendatud faili edastamisel järjekord, et kõigepealt laaditakse välise partneri serverisse uus fail ja alles siis kustutatakse vana fail, et ei tekiks olukorda, kus väline partner soovib hakata LHV UK klientide andmeid töötlema, aga juhuslikult on LHV poolne protsess selles olukorras, kus vana fail on eemaldatud, aga uut faili pole veel partneri serverisse üles laetud.

Loodud lahenduse korral on faili genereerimise ja välise partneri serverisse saatmise keskmine aeg toodangu keskkonnas 13 sekundit. Keskmise arvutamiseks jälgiti ühe nädala kõiki neid protsesse, mille jooksul fail genereeriti ja välise partneri serverisse saadeti. Selliseid protsesse on iga tunni kohta üks ehk nädalas kokku 168. Aja jooksul, kui LHV UK klientide ja nende kontode ja viban-kontode arv tõuseb, muutub ka protsessile kuluv aeg pikemaks. Arvestades, et toodangus kulub umbes 80 000 cop-kontoandmete välisele partnerile saatmiseks keskmiselt ligikaudu 13 sekundit, siis 15 minuti jooksul on võimalik välisele partnerile saata keskmiselt kuni 5,5 miljoni cop-konto andmed. Vajadusel saaks ka funktsionaalsuse käivitamise sagedust harvemaks muuta, et ka veel suuremaid koguseid andmeid saaks saata. Arvestades, et 2021. aastal

oli Ühendkuningriigi kõige suuremas pangas avatud 2,45 miljonit pangakontot ja kuues kõige suuremas pangas kokku 6,35 miljonit pangakontot [10] , siis loodud lahendusest piisab andmete saatmiseks ka kõige tugevama kontode ja viban-kontode kasvu korral.

Lahenduse jaoks loodud andmebaasi tabelis COP_ACCOUNTS on erinevalt varasematest LHV pangas kasutuses olevatest andmebaasi tabelitest ainult kahe võimaliku väärtuse (jah/ei, tõene/väär, 1/0) väljade andmetüübina kasutatud BIT andmetüüpi [7] . Andmebaasis salvestatakse need väärtused numbrilisel kujul 1 või 0. Lisaks saab kasutada kolmanda variandina tühja väärtust ehk *null*. Varem kasutati andmetüübina TINYINT, mille mahutavus on 1 bait iga ühe tabeli kirje veeru kohta. 1 baidi sisse mahub aga kuni 8 BIT andmetüübiga tabeli kirje veeru väärtused, seega BIT andmetüübi kasutamine TINYINT andmetüübi asemel võimaldab andmebaasis kuni 8 korda vähem salvestusmahtu kasutada.

Välja töötatud lahenduses kasutatakse andmebaasist pärides *Optional<T>* andmetüüpe [12] . Seda lähenemist kasutatakse, kui esineb võimalus, et päritavad andmed võivad eksisteerida ning võivad ka mitte eksisteerida. *Optional<T>* andmetüüpide kasutamine annab lihtsaid võimalusi tingimuslausete moodustamiseks (näiteks *isPresent*() või *isEmpty*() meetodid). *Optional<T>* andmetüübist saab objekti olemasolul objekti kätte *get*() meetodiga. *Optional<T>* andmetüüpide kasutamine parandab koodi loetavust võrreldes tavaliste Java objektidega. *Optional<T>* andmetüübid on kasutusel alates Java 8-st.

Kui cop-konto enam ei täida sünkroniseerimise tingimusi, kustutatakse cop-konto kirje vahetabelist COP_ACCOUNTS. Kustutamise kasuks otsustati, sest COP_ACCOUNTS tabelis hoitakse cop-kontosid, mis vastavad sünkroniseerimise tingimustele. Tingimustele mitte vastavate kirjete säilitamine eeldaks lisanduvat staatuse veergu tabelis ning cop-kontode sorteerimist selle staatuse alusel rakenduses. Selline funktsionaalsus pikendaks protsessile kuluvat aega. Tingimustele mitte vastavate cop-kontode andmeid ei ole vaja säilitada, sest need andmed on tegelikult olemas rakenduse teistes tabelites ning konkreetne ajaline cop-konto seis on nende tabelite pealt taastatav. Selline cop-kontode taastamine nõuaks aga lisanduvat arendust, mida ei võeta ette enne, kui vajadus tekib, sest esmases projekti analüüsis ei tulnud välja, et cop-kontode ajaloolist seisu vaja läheks.

Peale arenduse lõpetamist selgus, et väline partner sünkroniseerib nende serverisse edastatud faili sisu ainult üks kord päevas öösel. See tekitas klientidele olukorra, kus nad küll lõid endale LHV UK konto või viban-konto, kuid ei saanud seda sama ööpäeva jooksul ilma piiranguteta kasutada. Kuigi igal tunnil andmefaili saatmine tõstab kindlust, et välise partneri andmete sünkroniseerimise protsessi ajal on kõige värskemad andmed partnerile edastatud, oleks keskkonnasõbralikum LHV UK klientide kontode ja viban-kontode andmed välisele partnerile edastada ainult üks kord ööpäevas, sest iga protsess kulutab serveripargi arvutite arvutusvõimsust ja seeläbi suurendab elektrienergia kulu.

5.1 Rakendusliidese testimine

Rakendusliidese testimiseks paluti arendaja arvutile teha erand LHV sisevõrgust failide saatmiseks välise partneri SFTP serverisse, sest turvakaalutlustel on arendajate kohalikes masinates töötavatele rakendustele välisühendused keelatud. Testimiseks kasutati ühikteste ja manuaalset testimist. Kõik loodud klassid ja meetodid kaeti 100-protsendiliselt ühiktestidega, sealhulgas kõik võimalikud rakendusliidese lahenduskäigud.

Lokaalseks testimiseks sisestati COP_ACCOUNTS tabelisse toodanguga samas suurusjärgus ehk 80 000 kirjet ning käivitati rakendus. Rakendust hoiti töös kahe tööte tsükli jagu veendumaks, et teise tööte käigus enam uut faili sama tunni sees ei saadeta. Saadetava faili olemasolu välise partneri testserveris kontrolliti PuTTY tarkvaraga [11] , millega saab ligi välistele serveritele. Partneri testserveris olev fail avati ja kontrolliti, et kõik 80 000 test cop-kontot seal ka oleks. Järgmiseks muudeti COP_ACCOUNTS tabelis mõne kirje andmeid ning käivitati rakendus järgmise tunni sees. Peale tööte käigus faili uuendamist veenduti taas PuTTY tarkvara kasutades, et välise partneri testserveris on üksnes uue nime ja muudetud sisuga fail ning eelmise tunni sees loodud fail on eemaldatud.

LHV Panga testserveris testiti tööte tulemuslikkust nii, et lasti töötel kõigepealt test-andmebaasi andmete pealt luua fail, mis saadeti välise partneri testserverisse. PuTTY tarkvara abil kontrolliti faili sisu ja kopeeriti fail hilisemaks võrdlemiseks töö autori arvutisse. Peale seda muudeti testrakenduse kaudu mõne LHV UK subjekti andmeid,

muudetud andmed jõudsid Accounts-Products rakenduse andmebaasi ning järgmise tunni jooksul genereeritud failist kontrolliti, et andmed oleks uues failis muudetud.

Tehtud arendusele ei loodud integratsiooniteste, sest välise partneri SFTP serverit ja faili sinna saatmist on keeruline imiteerida (ingl k *mocking*). Imiteerimine eeldaks teise testserveri püstipanekut, et nende vahel SFTP failiedastusprotokolli kasutades faili saatmist testida. Antud arenduse kontekstis piisas sellest, et testserveris testiti manuaalselt faili saatmist välise partneri testserverisse, saades sealt kinnituse, et failiedastuse tööriistad toimivad.

5.2 Esilekerkinud probleemid

Arenduse käigus kerkis esile ka paar probleemi. Esialgu sai lahendus loodud nii, et COP_ACCOUNTS tabelist võeti *null* väärtusega andmed ja sisestati üks-ühele välisele partnerile edastatavasse faili sama väärtus ehk *null*. Välise partneri nõue oli aga tegelikult, et *null* väärtuse korral peab failis olema väärtuseks tühi sõne ehk „”. Lahenduses muudeti mittekohustuslike andmeväljade puhul automaatne vastendamine tingimuslausega vastendamiseks.

Teine viga esialgse lahenduse juures oli samuti seotud vastendamisega. Spring Data JPA konverteeris COP_ACCOUNTS tabelis olevad BIT andmetüübi väärtused 1 ja 0 väärtusteks true ja false, kuigi välise partneri nõue oli, et need peavad kliendiandmete failis olema samuti väärtused 1 ja 0. Lahenduses muudeti automaatne vastendamine BIT andmetüübi korral tingimuslausega vastendamiseks.

5.3 Edasi arendamist vajavad kohad

Peale arenduse valmimist selgus, et väline partner sünkroniseerib nende serverisse edastatud faili sisu ainult üks kord päevas öösel, mistõttu ei ole vajadust faili genereerida ja partnerile edastada igal tunnil. Faili genereerimise ja partnerile edastamise sagedust võib tulevikus harvendada.

LHV UK klientide andmete sünkroniseerimise teise etapina saab kasutusele võtta veebiteenuse, millega saadetakse iga andmemuudatuse korral välisele partnerile ajakohane andmestik.

Testimist saaks veel parandada automaattestide lisamisega. Automaattestide loomiseks oleks vaja luua veebiteenused, mis kasutavad faili genereerimise ja saatmise töötes kasutatavaid meetodeid. Automaattestide jaoks oleks vaja luua kolm veebiteenust:

- *.tsv faili loomiseks;
- faili edastamiseks välise partneri test serverisse;
- faili pärimiseks välise partneri test serverist.

6 Kokkuvõte

Lõputöö eesmärgiks oli luua AS-ile LHV Pank lahendus AS-i LHV Pank Ühendkunignriigi filiaali kliendi-, konto- ja viban-konto andmete edastamiseks välisele partnerile, kes pakub Ühendkuningriigis pankadevaheliste maksete saaja makse-eelse tuvastamise teenust. Selleks oli vaja kokku koguda nõutud kliendi-, konto- ja viban-konto andmed, genereerida neist andmetest *.tsv formaadis fail ja saata see välise partneri failiserverisse SFTP failiedastusprotokolli kasutades.

Eesmärgi täitmiseks võrreldi kõigepealt kolme andmete kogumise ja faili kirjutamise meetodit – (1) andmete kogumine faili genereerimise ajal, (2) andmete kogumine vahetabelisse enne faili genereerimist, (3) andmete muutmine alalises failis. Meetodite võrdlemiseks loodi iga meetodi kohta eksperimentaalarendus. Teine nimetatud meetoditest oli ülejäänutest ühe kuni kahe suurusjärgu võrra kiirem.

Rakendusliidese loomiseks kasutati enne faili genereerimist andmete vahetabelisse kogumise meetodit (2) ning faili edastamiseks välise partneri failiserverisse kasutati JCrafti JSch SSH2 Java teostuse SFTP failiedastuse meetodeid. Toodangus genereeritakse vahetabelist fail ja saadetakse välise partneri failiserverisse keskmiselt ligikaudu 13 sekundiga.

Rakendusliides kaeti 100-protsendiliselt ühiktestidega ning testiti ka manuaalselt. Lisaks kinnitas ka välise partneri vastutav isik, et fail koos nõuetekohaste andmetega jõuab nende failiserverisse. Loodud lahendus ei ole ennast peale valmimist vigadega meelde tuletanud.

Kasutatud kirjandus

[1] AS LHV Group, *Who are we?*, 2021. [Online]. Loetud aadressil: https://www.lhv.com/about/#who Kasutatud: 30.10.2022.

[2] Cybernetica AS, *Plaanur (2)*, 2022. [Online]. Loetud aadressil: https://akit.cyber.ee/term/8141-plaanur-2 Kasutatud 01.12.2022.

[3] Cybernetica AS, *Tööde*, 2022. [Online]. Loetud aadressil: https://akit.cyber.ee/term/5348-toode Kasutatud 01.12.2022.

[4] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Boston: Addison Wesley, 2003.

[5] Pay.UK, *Confirmation of Payee*, 2022. [Online]. Loetud aadressil: https://www.wearepay.uk/what-we-do/overlay-services/Confirmation-of-Payee/ Kasutatud: 30.10.2022.

[6] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship.* Pearson Education, 2009.

[7] Microsoft, *bit (Transact-SQL)*, 2022. [Online]. Loetud aadressil: https://learn.microsoft.com/en-us/sql/t-sql/data-types/bit-transact-sql?view=sql-server-ver16 Kasutatud: 21.11.2022.

[8] Oracle, *Getting Started with Java*, 2021. [Online]. Loetud aadressil: https://dev.java/learn/getting-started-with-java/ Kasutatud: 18.11.2022.

[9] Oracle, *What is a JPA Entity?*, 2009. [Online]. Loetud aadressil: https://docs.oracle.com/cd/E16439_01/doc.1013/e13981/undejbs003.htm Kasutatud: 19.11.2022.

[10] Statista Research Department, *Total number of open basic bank accounts in the United Kingdom (UK) as of June 2021, by bank*, 2022. [Online]. Loetud aadressil: https://www.statista.com/statistics/857662/number-of-basic-bank-accounts-per-bank-uk/ Kasutatud: 20.11.2022.

[11] S. Tatham, *PuTTY*, 2022. [Online]. Loetud aadressil: https://apps.microsoft.com/store/detail/putty/XPFNZKSKLBP7RJ Kasutatud: 18.11.2022.

[12] R.-G. Urma, *Tired of Null Pointer Exceptions? Consider Using Java SE 8's "Optional"!*, 2014. [Online]. Loetud aadressil: https://www.oracle.com/technical-resources/articles/java/java8-optional.html Kasutatud 21.11.2022.

[13] VMware, *Spring Data JPA*, 2022. [Online]. Loetud aadressil: https://spring.io/projects/spring-data-jpa Kasutatud: 19.11.2022.

[14] VMware, *Spring Boot*, 2022. [Online]. Loetud aadressil: https://spring.io/projects/spring-boot Kasutatud: 18.11.2022.

[15] L. S. Vailshery, *Most used programming languages among developers worldwide as of 2022*, 2022 [Online]. Loetud aadressil: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ Kasutatud: 18.11.2022.

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

Mina, Siim Kilki

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kontoandmete sünkroniseerimine välise partneri süsteemiga AS-i LHV Pank näitel", mille juhendaja on Toomas Lepikult

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.

29.12.2022

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 – COP_ACCOUNTS tabeli esmane andmemigratsioon kontode andmetega

```
DECLARE @batch_size INT = 1000
DECLARE @row_count INT = @batch_size
WHILE @row_count > 0
   BEGIN
     BEGIN TRANSACTION
       ;WITH cte AS (
         SELECT TOP (@batch_size) a.account_id, a.account_no_bban,
c.customer_type, c.first_name, c.last_name, c.company_name
         FROM COMMON.accounts a
           INNER JOIN COMMON.customers c
           ON a.customer_id = c.customer_id
         WHERE a.bank_legal_entity_id = 2
           AND a.is_active = 1
           AND a.account_id NOT IN (SELECT account_id FROM
COMMON.confirmation_of_payee_accounts WHERE viban_id IS NULL)
       )
       INSERT INTO COMMON.confirmation_of_payee_accounts
(account_id, identification, account_type, name, scheme_name,
is_supported, is_cop_opt_out, is_switched, is_collection_account)
       SELECT cte.account_id,
         cte.account_no_bban,
         CASE
           WHEN cte.customer_type = 'LEGAL'
           THEN 'Business'
           ELSE 'Personal'
         END,
         CASE
           WHEN cte.customer_type = 'LEGAL'
           THEN (SUBSTRING(',,' + cte.company_name, 0, 140))
           ELSE (SUBSTRING(',' + cte.first_name + ',' +
cte.last_name, 0, 140))
         END,
         'SortCodeAccountNumber',
         1, 0, 0, 0
       FROM cte;
       SET @row_count = @@ROWCOUNT
     COMMIT TRANSACTION
   END;
```

Lisa 3 – COP_ACCOUNTS tabeli esmane andmemigratsioon viban-kontode andmetega

```
DECLARE @batch_size INT = 1000
DECLARE @row_count INT = @batch_size
WHILE @row_count > 0
   BEGIN
     BEGIN TRANSACTION
       ;WITH cte AS (
         SELECT TOP (@batch_size) v.account_id, v.viban_id,
v.account_no_bban, v.owner_type, v.status, v.type, v.person_name,
v.company_name, c.company_name as master_account
         FROM COMMON.viban_accounts v
           INNER JOIN COMMON.accounts a
           ON v.account_id = a.account_id
           INNER JOIN COMMON.customers c
           ON a.customer_id = c.customer_id
         WHERE a.bank_legal_entity_id = 2
           AND a.is_active = 1
           AND v.status NOT IN ('PENDING', 'REJECTED', 'CLOSED')
           AND v.type = 'V'
           AND v.viban_id NOT IN (SELECT viban_id FROM
COMMON.confirmation_of_payee_accounts WHERE viban_id IS NOT NULL)
         )
         INSERT INTO COMMON.confirmation_of_payee_accounts
(account_id, viban_id, identification, account_type, name,
scheme_name, is_supported, is_cop_opt_out, is_switched,
is_collection_account, name_alternative)
         SELECT cte.account_id,
           cte.viban_id,
           cte.account_no_bban,
           CASE
             WHEN cte.owner_type = 'LEGAL' THEN 'Business'
             WHEN cte.owner_type = 'PRIVATE' THEN 'Personal'
           END,
           CASE
           WHEN cte.owner_type = 'LEGAL' AND cte.company_name IS NOT
NULL AND cte.status != 'PENDING'
           THEN (SUBSTRING(',,' + cte.company_name, 0, 140))
           WHEN cte.owner_type = 'PRIVATE' AND cte.person_name IS
NOT NULL AND cte.status != 'PENDING'
           THEN (SUBSTRING(',,' + cte.person_name, 0, 140))
           END,
           'SortCodeAccountNumber',
           1, 0, 0, 0,
           cte.master_account
         FROM cte;
         SET @row_count = @@ROWCOUNT
    COMMIT TRANSACTION
END;
```