LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

BIBLIOTĒKA PRIEKŠ TIRGOTĀJU INTEGRĀCIJAS MAKSĀJUMU SISTĒMĀ

KVALIFIKĀCIJAS DARBS

Autors: **Ralfs Eduards Braunfelds**
Studenta apliecības Nr.: rb17055
Darba vadītājs: **Jānis Zuters**

RĪGA 2019

ANOTĀCIJA

Šis dokuments ir Latvijas Universitātes Datorikas fakultātes studiju kursa "Kvalifikācijas darbs II" (DatZ2049) ietvaros izstrādātās "Bibliotēkas priekš tirgotāju integrācijas maksājumu sistēmā" programmatūras tehniskā dokumentācija, kas ietver programmatūras prasību specifikāciju, programmatūras projektējuma aprakstu, programmatūras testēšanas dokumentāciju un projekta organizācijas aprakstu.

"Bibliotēkas priekš tirgotāju integrācijas maksājumu sistēmā" ir *Java* programmēšanas valodas bibliotēka, kuru lietotājiem ir nepieciešams ieviest savā pirmkodā un kuras galvenais mērķis ir atvieglot uz tīkla balstītu tirgotāju sistēmu saziņu ar banku, kura ir ieviesusi uzņēmuma "*Tieto Latvia*" izstrādāto sistēmu. Otrs bibliotēkas mērķis ir atvieglot darbu attiecīgā uzņēmuma izstrādātājiem, ieviešot komandrindas režīma saskarni.

Atslēgvārdi: bibliotēka, tirgotāju sistēma, bankas sistēma, komunikācijas atvieglošana, *Java* programmēšanas valoda, komandrindas režīms.

ANNOTATION

A LIBRARY FOR MERCHANT INTEGRATION INTO A PAYMENT SYSTEM

This document is the technical documentation for “A Library for merchant integration into a payment system”, which has been developed in the scope of the University of Latvia Faculty of Computing course “Qualification paper II” (DatZ2049) and includes the software requirement specification, software design description, testing documentation and the project organization description.

“A Library for merchant integration into a payment system” is a library for the *Java* programming language, which users are expected to include in their source code and whose main goal is to facilitate the communication between a web based merchant system and a bank, that has introduced the system developed by “*Tieto Latvia*”. The other goal for this library is to simplify work for the developers of the mentioned company by introducing a command line interface.

Keywords: library, merchant system, bank system, communication facilitation, *Java* programming language, command line.

SATURA RĀDĪTĀJS

IEVADS

Informāciju tehnoloģiju uzņēmums "*Tieto Latvia*" izstrādā un uztur sistēmu, kuru bankas var ieviest, lai pieņemtu maksājumus un tos realizētu. Tirgotājiem, piemēram, e-veikaliem, savukārt, lai varētu veikt ar maksājumiem saistītas funkcijas savā sistēmā, ir nepieciešams slēgt līgumu ar banku, kurā šāda veida sistēma ir ieviesta. Bankas tirgotājiem pretī sniedz visu nepieciešamo informāciju, lai veiksmīgi veiktu komunikāciju starp abām sistēmām.

Tirgotājiem dotā informācija atklāj to, kā var sasniegt noteiktus serverus, kādos formātos dati ir jāiesūta, kā ir jāapstrādā saņemtie dati un jāveic autorizācija. Saražot pirmkodu, kas atbilst visām šīm prasībām, var būt ļoti laikietilpīgs un sarežģīts process, tāpēc pēc līgumu parakstīšanas bankas tirgotājiem piedāvātu izmantot šo produktu. Šīs bibliotēkas gala produkts ir *.jar* fails, un tās galvenais mērķis ir būt kā starpniekam starp tirgotāju un banku un atvieglot komunikācijas pirmkoda izstrādes procesu, sniedzot jau gatavas klases un metodes.

Bibliotēku izstrādes specifika ir tāda, ka tai ir jābūt saprotamai un vienkāršai, pildot pēc iespējas vairāk uzdevumus, neskarot tirgotāju sistēmu jeb klientu, kas to izmanto. Ir svarīgi noteikt visas tirgotāja prasības un iespējamās situācijas, bet var arī paļauties, ka klients savu pirmkodu modificē bibliotēkas prasībām. Bankas sistēma, savukārt, nedrīkst mainīt savu funkcionalitāti, rīkam ir pilnībā jāpielāgojas bankas sistēmai un pirmkods jāievieš, lai nodrošinātu vieglu modifikāciju, paredzot, ka bankas sistēmas darbība un prasības var mainīties.

Nolūks

Šī dokumenta nolūks ir iepazīstināt sistēmas izstrādātājus ar bibliotēkas prasībām, projektējumu, funkcijām un to pielietojumiem. Šis dokuments ir savstarpēja vienošanās starp pasūtītāju un izpildītāju.

Tā kā šis dokuments ir ticis izstrādāts Latvijas Universitātes Datorikas fakultātes studiju kursa "Kvalifikācijas darbs II" (DatZ2049) ietvaros, tad tā mērķis ir arī aprakstīt izstrādātās programmatūras veikto testēšanu un projekta organizāciju.

Darbības sfēra

"Bibliotēka tirgotāju integrācijai maksājumu sistēmā" ir bibliotēka, jeb programmatūras izstrādes rīks, kurš nodrošina jau gatavu *Java* programmēšanas valodas pirmkodu, jeb klases un metodes *.jar* faila formātā, ko lietotāji var ieviest savā *Java* pirmkodā, lai atvieglotu izstrādes procesu integrācijai bankas maksājumu sistēmā tirgotājiem, kuri ir parakstījuši nepieciešamos līgumus ar banku, kura izmanto IT uzņēmuma "*Tieto Latvia*" bankas sistēmu.

Bibliotēka ļauj lietotājam vieglāk izpildīt ar maksājumiem saistītas funkcijas kā vairāku veidu jaunu maksājumu veikšanu, naudas atgriešanu, maksājuma pabeigšanu/atcelšanu, vienu vai vairāku maksājumu datu ieguvi.

Šis rīks no lietotāja "slēpj" un lietotāja vietā veic visas *HTTP* savienojumu izveides, datu pārsūtīšanu/saņemšanu, nepieciešamo formātu nodrošināšanu un autorizācijai nepieciešamo funkciju veikšanu.

Vairākus konfigurācijas datus, kā savienojuma veikšanas parametrus un autorizācijai nepieciešamos datus, lietotājam ir iespēja uzstādīt, tos padodot caur lietotāja pirmkodu vai konfigurācijas failu.

Uzņēmuma izstrādātājiem izstrādes procesā bieži ir nepieciešams veikt ar maksājumiem saistītas funkcijas testēšanas nolūkos. Šī izstrādājamā bibliotēka paredz izstrādātājiem atvieglot testēšanu, ļaujot lietotājiem funkcionalitātes veikt no operētājsistēmas komandrindas režīma, bibliotēkai nepaļaujoties uz tās ieviešanu tirgotāja sistēmā.

Saistība ar citiem dokumentiem

Dokuments tika izstrādāts saskaņā ar programmatūras prasību specifikācijas standartu LVS 68:1996 "Programmatūras prasību specifikācijas ceļvedis" [1], programmatūras projektējuma apraksta standartu LVS 72:1996 "Ieteicamā prakse programmatūras projektējuma aprakstīšanai" [2] un uzņēmuma "*Tieto Latvia*" bankas sistēmas specifikāciju.

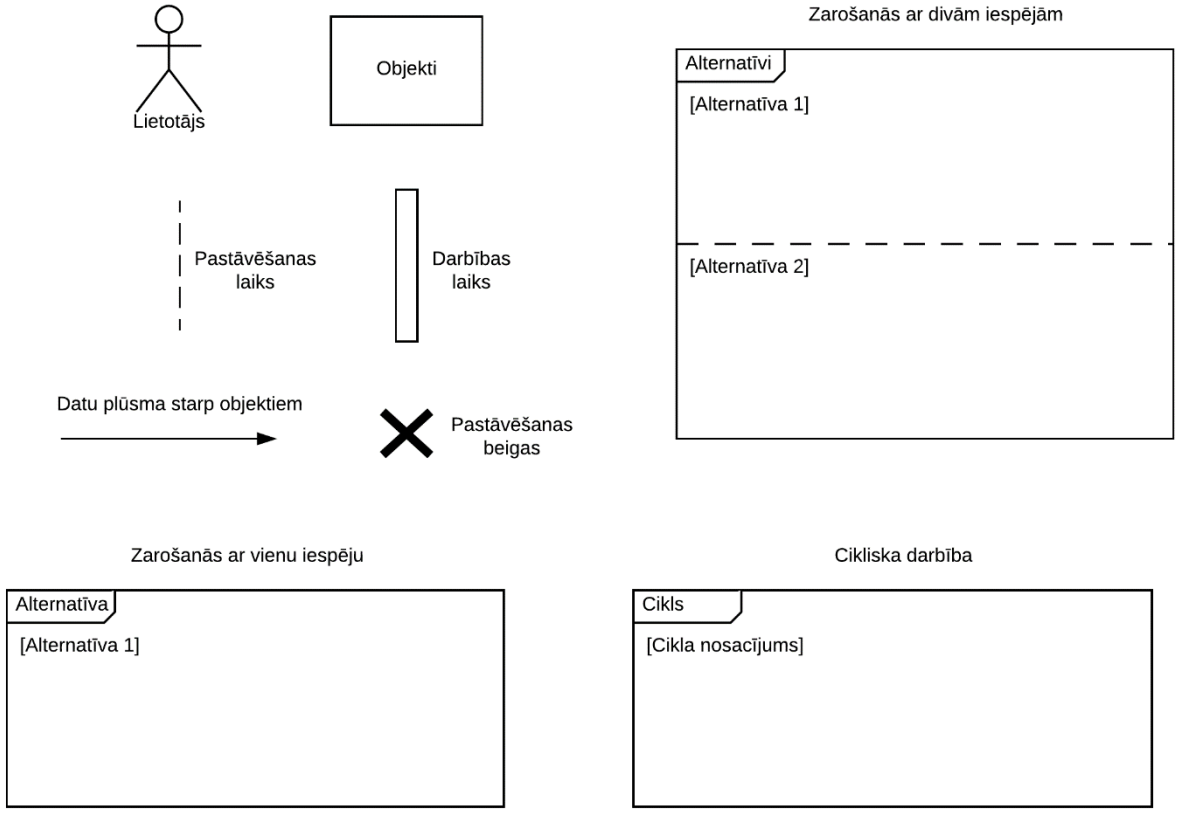
Priekš detalizētākas informācijas par dokumentā minētajām tehnoloģijām un datiem, skatīt *JSON* datu struktūru formātu aprakstu *"Introducing JSON"* [4], *HTTP* savienojumu un to parametru arpakstu *"HTTP and everything you need to know about it"* [5], *TLS* protokolu un to parametru izmantošanu priekš *HTTP* savienojumiem aprakstu *"Everything about HTTPS and SSL (Java)"* [6], *Java* programmēšanas valodas apmācību par tās darbību, datu tipiem un struktūrām, kas tiek minētas šajos dokumentos [7, 8, 9, 10, 11, 12].

Pārskats

Turpmākais kvalifikācijas darba dokuments sastāv no 5 daļām:

- Vispārējais apraksts – īsumā tiek aprakstīti vispārējie faktori, kuri iedarbojas uz produktu un tā prasībām.
- Programmatūras prasību specifikācija – sīkāk tiek aprakstītas programmatūras funkcionālās un nefunkcionālās prasības.
- Programmatūras projektējuma apraksts – ietver daļēju funkciju un saskarņu projektējumu.
- Programmatūras testēšana – tiek aprakstīti testpiemēri un kā testēšana ir tikusi veikta.
- Projekta organizācija – tiek aprakstīts veiktais plānošanas un izstrādes process.

APZĪMĒJUMU SARAKSTS



1. att. – **Secību diagrammas apzīmējumi**

JĒDZIENI

Tirgotājs – cilvēks jeb uzņēmums, kas nodarbojas ar produktu pārdošanu.

E-veikals – tirgotājs, kurš veic savu produktu pārdošanu, izmantojot tīkla lietotni.

Bibliotēka – resursu un funkcionalitāšu kolekcija priekš kādas programmēšanas valodas.

***Java* –** kompilējama programmēšanas valoda, kas ir balstīta uz *OOP* pieeju.

***Java* pirmkods –** programmatūras pirmkods, kurš ticis izstrādās, izmantojot *Java* programmēšanas valodu.

***Java* datu tips –** datu tips, kuru atbalsta *Java* programmēšanas valoda.

Integrācija – sistēmas iekļaušana kādas citas sistēmas darbībā.

Komandrindas režīms – saskarne priekš komandu tiešas ievades datoru operētājsistēmai.

Komandrindas komanda – instrukcija datora operētājsistēmai caur komandrindas režīmu.

Tīkla lietotne – lietotne, kura ir balstīta uz kāda attālināta servera un ir pieejama caur Internetu, izmantojot kādu tīmekļa parlūku.

Klase – pirmkoda šablons priekš objektu izveidošanas.

Objekts – datu struktūra, kura seko kādas klases šablonam.

Metode – kāda noteikta procedūra jeb funkcija, ko spēj veikt kāda klase.

Java Exception – *Java* programmēšanas valodas klase, kura raksturo nevēlamu programmas izpildes laikā radušos kļūdu.

Klases konstruktors – metode, kura tiek izsaukta klases objekta izveides brīdī.

Pavedienu drošība jeb *thread-safety* – programmatūra, kura pareizi veic savu funkcionalitāti, ja to veic vairāki pavedieni vienlaicīgi.

Pavediens – viens process, kurš spēj izpildīt secīgas programmētas instrukcijas.

Sesija – īslaicīgi derīgu datu apmaiņa starp sistēmām, kas parasti apstiprina īslaicīgu piekļuvi kādiem datiem.

***JSON array* -** datu tips, kas satur sakārtotu vērtību sarakstu *JSON* formātā.

JSON object – datu tips, kas satur strukturētas vērtības *JSON* formātā.

JSON identifikators – vērtība, kas identificē katru *JSON* struktūru *JSON* formātā.

***HTTP* pieprasījums –** pieprasījums, kuru klients sūta uz serveri, lai sāktu datu apmaiņu, izmantojot *HTTP* protokolu.

***HTTP* atbilde –** atbilde, ko serveris sūta klientam pēc *HTTP* pieprasījuma saņemšanas, izmantojot *HTTP* protokolu.

***HTTP header* dati –** dati, kas tiek padoti *HTTP* pieprasījumos un atbildēs un nosaka vairākus savienojuma parametrus.

***HTTP body* dati –** dati, kas *HTTP* pieprasījumos un atbildēs seko pēc *header* datiem un parasti satur nepieciešamo, pārsūtāmo informāciju.

***HTTP query* dati –** dati, kuri *HTTP* pieprasījumos tiek definēti kopā ar *URL*.

***application/x-www-form-urlencoded* –** viens no *HTTP* pieprasījumu mediju tipiem, kas nosaka datu formātu.

Atslēgu pāris – publisko un privāto atslēgu pāris, kas tiek lietots *TLS* protokolos priekš datu šifrēšanas un atšifrēšanas.

Atslēgu pāra *alias* – unikāla simbolu virkne, kas identificē atslēgu pāri *Keystore* failos.

Sertifikāts – digitāla informācija, kura apliecina kāda *HTTP* pieprasījuma vai atbildes sūtītāja identitāti.

***Keystore* fails –** faila formāts, kas spēj glabāt atslēgu pārus un sertifikātus.

Boolean – *Java* datu tips, kuram ir iespējamas divas vērtības – *true* un *false*.

***Long* –** *Java* datu tips, kurš spēj glabāt skaitļus 64 bitu izmērā.

Double - *Java* datu tips, kurš spēj glabāt skaitļus ar peldošo punktu 64 bitu izmērā.

***Heap* operatīvā atmiņa –** atmiņas daļa, kur programmas iedala dinamiski veidotu atmiņu.

***Stack* operatīvā atmiņa –** atmiņas daļa, kur programmas iedala statiski veidotu atmiņu.

UNIX Shell – komandrindas komandu interpretētājs priekš *UNIX* tipa operētājsistēmām.

Builder pattern – OOP pieejas objekta izveides un metodes izsaukšanas struktūra.

Agile – moderna programmatūras izstrādes metodoloģija, kas apvieno inkrementālo un iteratīvo izstrādes pieeju un liek uzsvaru uz komandas darbu un kontaktu ar pasūtītāju.

***Scrum* –** pieeja *Agile* metodoloģijas ieviešanai.

Git – atvērtā pirmkoda versiju kontroles sistēma.

SAĪSINĀJUMI

IT – Informāciju tehnoloģijas.

HTTP **-** *Hypertext Transfer Protocol* (protokols, kas nosaka kā caur tīklu tiek veikti pieprasījumi un sniegtas atbildes starp sistēmām).

JDK – *Java Development Kit* (izstrādes rīks, kas ļauj izstrādātājiem veidot *Java* programmas).

JSON **-** *JavaScript Object Notation* (formāts priekš datu apmaiņas caur tīklu).

URL - *Uniform Resource Locator* (tīkla adrese, kas nosaka kāda tīkla resursu atrašanās vietu datoru tīklā).

TLS - *Transport Layer Security* (komunikācijas protokols drošai saziņai Internetā).

OOP - objektorientētā programmēšana (programmēšanas paradigma, kurā programma tiek veidota, izmantojot objektus un klases).

MB - megabaits (informācijas apjoma mērvienība).

1. VISPĀRĒJAIS APRAKSTS

1.1. Esošā stāvokļa apraksts

Bez šī produkta pastāvēšanas, lai tirgotāji varētu veiksmīgi integrēt savu sistēmu bankas maksājumu sistēmā, ir nepieciešams ar to komunikāciju veidot pašrocīgi, sekojot attiecīgās bankas sistēmas dokumentācijām.

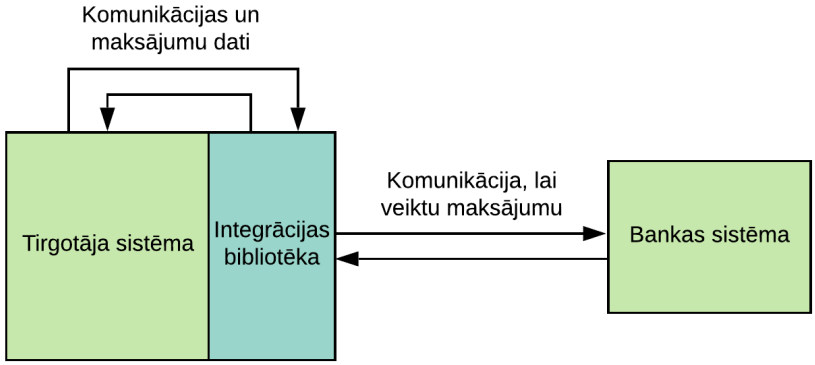
1.2. Pasūtītājs

"Bibliotēka priekš tirgotāju integrācijas maksājumu sistēmā" ir aprakstīta un izstrādāta pēc uzņēmuma SIA "*Tieto Latvia*" pasūtījuma Latvijas Universitātes Datorikas fakultātes studiju kursa "Prakse" (DatZ2033) ietvaros.

1.3. Produkta perspektīva

"Bibliotēka priekš tirgotāju integrācijas maksājumu sistēmā" ir programmatūra, kura ir papildinājums sistēmas pirmkodam, kuru uztur un lieto tirgotāji. Šīs tirgotāju sistēmas arhitektūra un funkcijas ir nenoteiktas un neparedzamas, tiek tikai veikts pieņēmums, ka tās ir tīmekļa lietotnes, kuras ir izstrādātas, izmantojot *Java* programmēšanas valodu, un, ka viena no to ārējām saskarnēm ir attiecīgā banka, ar kuru ir parakstīts līgums priekš ar maksājumiem saistītu funkciju veikšanas.

Kā redzams att. 1.1, tirgotāju sistēma, kura ir ieviesusi izstrādājamo bibliotēku, ir cieši ar to saistīta un, sniedzot tai nepieciešamos datus funkciju izpildīšanai, to izmanto, lai veiktu komunikāciju ar ārējo bankas sistēmu.



1.1 att. – **Produkta perspektīvas ilustratīvs piemērs**

1.4. Darījumprasības

Programmatūrai ir jānodrošina šādas funkcionalitātes:

- Ļauj lietotājam kā bibliotēkas saskarni izmantot pirmkodu vai komandrindas režīmu, abās pielietojot pierādītas un modernas vadlīnijas.
- Lietotājam ir iespēja konfigurēt datus kā serveru adreses, autorizācijas un citus savienojumu veikšanas datus caur pirmkodu vai konfigurācijas teksta failu.
- Veic visus nepieciešamos *HTTP* tīkla savienojums un datu pārsūtīšanu/saņemšanu.
- Nodrošina nepieciešamo datu formātu pārveidošanu.
- Nodrošina nepieciešamo autorizācijas funkciju veikšanu.
- Izvada lietotājam aprakstošus kļūdu paziņojumus, izmantojot *Java Exception* kļūdu paziņojuma klases objektus.
- Ļauj lietotājam izpildīt maksājumu funkcijas, ievadot un pretī saņemot nepieciešamos maksājumu datus.
- Nodrošina drošu un efektīvu darbību, pieņemot, ka lietotāju sistēmas ir tīmekļa lietotnes un ir nepieciešams nodrošināt vairāku funkciju vienlaicīgu darbību.

1.5. Sistēmas lietotāji

Sistēmai ir paredzēts viens lietotāja veids, kurš ir pazīstams ar maksājumu funkciju saistītu biznesa loģiku, tīkla savienojumu un to parametru būtību. Šim lietotājam ir pieejamas visas bibliotēkas funkcijas un ir dota izvēle atbalstīto saskarņu izmantošanai.

1.6. Vispārējie ierobežojumi

Bibliotēkas lietošana ir jāatbalsta *Java* programmēšanas valodas pirmkodiem, kuri izmanto programmatūras izstrādes rīku *JDK 8* vai augstāku versiju, atbilstība ar vecākām versijām nav nepieciešama. Programmatūrai ir jānodrošina veiksmīgu darbību, pieņemot, ka lietotāji ir tīmekļa lietotnes ar vairākiem, vienlaicīgiem funkciju veikšanas pieprasījumiem.

1.7. Pieņēmumi un atkarības

Tiek veikts pieņēmums, ka šī produkta lietotājs to izmanto priekš tīkla lietotnes. Lai izmantotu visas programmatūras funkcionalitātes, lietotājam ir nepieciešams:

- Nodrošināt datoru, ekrānu un datora tastatūru.
- Attiecīgajam datoram nodrošināt Interneta pieslēgumu.
- Attiecīgajam datoram nodrošināt *Java* programmēšanas valodas izstrādes vidi, kura pielieto programmatūras izstrādes rīku *JDK 8* vai augstāku versiju.

2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

2.1. Funkcionālās prasības

2.1.1. Vispārējas nodaļas

Produkta specifikas

Tā kā šī produkta galvenais mērķis ir būt papildinājumam kādai citai sistēmai un rīkam izstrādātāju procesu atvieglošanai, tad ir nepieciešams ievērot īpašas prasības, kuras uz to attiecas. Šajā sadaļā šīs vispārējās prasības tiek uzskaitītas un aprakstītas.

Pieņemot, ka sistēmas uz ko šī bibliotēka tiek bāzēta ir tīmekļa lietotnes, tad ir nepieciešams nodrošināt, ka tā ir droša un pareizi veic funkcijas, izpildoties paralēliem procesiem jeb pavedieniem (*thread-safety*).

Šim produktam ir nepieciešams nodrošināt iespēju lietotājam izpildīt tās darbības no diviem saskarņu režīmiem: lietotāja pirmkoda un komandrindas režīma.

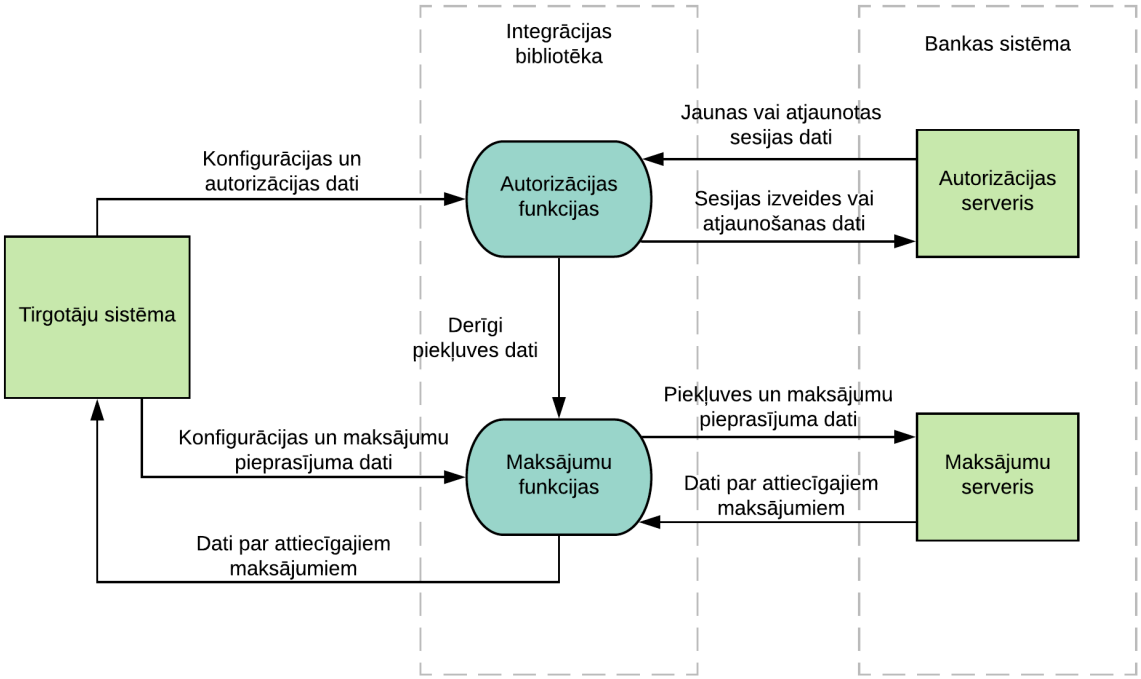
Lietotāja pirmkods ir galvenais saskarnes režīms un to paredz izmantot lietotāji, kuri ir attiecīgās bankas klienti un, kuri to pielieto reālās un jau ieviestās tirgotāju sistēmās. Tā kā lietotāju vēlamie izvaddatu formāti un nepieciešamās datu pielietošanas darbības ir neparedzamas, tad visiem lietotājam atgrieztiem datiem no šīs saskarnes ir jānodrošina iespēja tos iegūt gan *JSON* formātā, gan *Java* atbalstītās datu struktūrās.

Lietotāja operētājsistēmas komandrindas režīms ir mazāk svarīgs saskarnes režīms, un to paredz izmantot uzņēmuma "*Tieto Latvia*" izstrādātāji, lai atvieglotu testēšanas procesus. Šajā saskarnē ir paredzēts bibliotēku izmantot ar tās galaprodukta, *.jar* faila, palaišanas komandrindas komandu "*java -jar* <faila_nosaukums>", un ievaddatus padot kā šīs komandas parametrus.

Tā kā produkta galvenā lietotāju saskarne ir pirmkoda izstrādes vide, tad, sekojot *Java* programmēšanas valodas principiem un vadlīnijām, visiem datu, klašu un metožu nosaukumiem abos lietotāja saskarnes režīmos ir jābūt prezentētiem angļu valodā. Šī iemesla dēļ, visi nepieciešamie ievaddatu un funkciju nosaukumi šajā dokumentā tiek minēti angļu valodā.

Funkciju kopsavilkums

Šajā sadaļā tiek minētas un īsi aprakstītas visas bibliotēkas funkcijas, to iedalījumi un ārējās saskarnes (skat.att. 2.1). Priekš sīkākiem funkciju aprakstiem skat. sadaļu 2.1.2.



2.1 att. – **Funkciju kopsavilkuma ilustratīvs piemērs**

Autorizācijas funkcijas - šo funkciju galvenās saskarnes ir tirgotāju sistēma un bankas sistēmas autorizācijas serveris. Funkciju galvenais uzdevums, veicot sesijas datu pārvaldīšanu, ir sniegt maksājumu funkcijām derīgus piekļuves datus, kuri ir nepieciešami veiksmīgai saziņai ar bankas sistēmas maksājumu serveri.

- Funkcija “create new session” (AF.CNS.01) - pieprasa bankas autorizācijas serverim izveidot jaunu lietotāja sesiju, sniedzot nepieciešamos autorizācijas datus un pretī saņemot jaunus sesijas un to pārvaldīšanas datus.
- Funkcija “refresh session” (AF.RS.02) - pieprasa bankas autorizācijas serverim atjaunot lietotāja sesiju, sniedzot nepieciešamos autorizācijas datus un pretī saņemot jaunus sesijas un to pārvaldīšanas datus.

- Funkcija “get bearer” (AF.GB.03) - veic sesijas datu pārvaldīšanu un atgriež derīgus piekļuves datus. Sesijas dati tiek pārvaldīti, pārbaudot to derīgumu termiņus, veicot funkciju AF.RS.02, lai lieki neveidotu jaunu lietotāja sesiju, ja to ir iespējams atjaunot, vai funkciju AF.CNS.01, ja sesija iepriekš nav izveidota vai to atjaunošana nav iespējama. Šīs funkcijas atgriežamie piekļuves dati ir nepieciešami, lai gūtu piekļuvi citiem bankas sistēmas serveriem un funkcijām.

Maksājumu funkcijas - šo funkciju galvenās saskarnes ir tirgotāja sistēma un bankas sistēmas maksājumu serveris, un tās nodrošina vairāku veida maksājumu darbību izpildes pieprasīšanu attiecīgajam serverim.

- Funkcija “make purchase” (MF.MP.01) - pieprasa maksājumu serverim izveidot jaunu maksājumu, sniedzot nepieciešamos jauna maksājuma datus un pretī saņemot datus par attiecīgo jauno maksājumu.
- Funkcija “finish payment” (MF.FP.02) - pieprasa maksājumu serverim izpildīt maksājumu pabeigšanu uz jau esošu maksājumu, sniedzot nepieciešamos maksājuma pabeigšanas datus un pretī saņemot datus par attiecīgo rediģēto maksājumu.
- Funkcija “reverse payment” (MF.RVP.03) - pieprasa maksājumu serverim izpildīt maksājumu atcelšanu uz jau esošu maksājumu, sniedzot nepieciešamos maksājumu atcelšanas datus un pretī saņemot datus par attiecīgo rediģēto maksājumu.
- Funkcija “refund payment” (MF.RFP.04) - pieprasa maksājumu serverim izpildīt maksājumu naudas atgriešanu, sniedzot nepieciešamos naudas atgriešanas datus un pretī saņemot datus par attiecīgo jauno maksājumu.
- Funkcija “get payment” (MF.GP.05) - pieprasa maksājumu serverim atgriezt datus par jau esošu lietotāja maksājumu, sniedzot attiecīgā maksājuma identificējošus datus un pretī saņemot datus par attiecīgo esošo lietotāja maksājumu.
- Funkcija “get payment list” (MF.GPL.06) - pieprasa maksājumu serverim atgriezt datus par vairākiem jau esošiem lietotāja maksājumiem, sniedzot datus, pēc kā nepieciešams visus lietotāja maksājumus filtrēt, un pretī saņemot attiecīgo esošo lietotāja maksājumu sarakstu.

Citas funkcijas:

- Funkcija “load configuration” (CF.LC.01) – šī funkcija no lietotāja saņem konfigurācijas datus, kuri tiek izmantoti, lai attiecīgi konfigurētu autorizācijas un maksājumu funkciju izpildi. Attiecīgos datus lietotājam ir iespējams ievadīt, izmantojot vienu no divu veida ievaddatu avotiem: lietotāja pirmkodu vai konfigurācijas teksta failu, norādot tā atrašanās vietu sistēmā. Tā kā konfigurācijas dati nosaka visu pārējo funkciju darbību, tad šo funkciju ir nepieciešams veikt pirms jebkuras citas funkcijas veikšanas.

Sesijas dati un to pārvaldīšana

Izpildot bibliotēkas prasītās autorizācijas funkcijas, ir nepieciešams apstrādāt no bankas sistēmas autorizācijas servera saņemtos sesijas datus, kuri sastāv no šādiem laukiem:

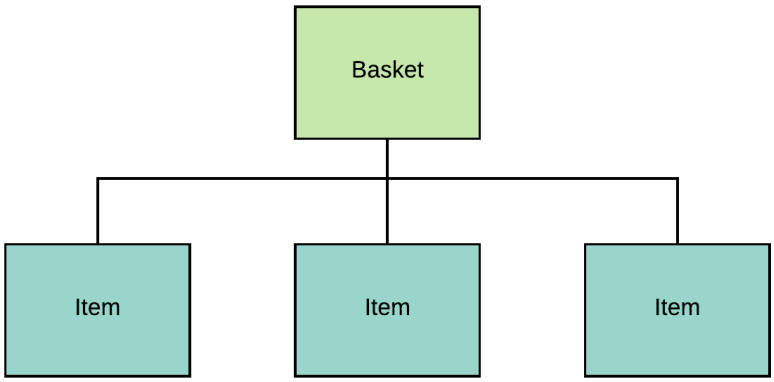
- access_token – unikāla lietotāju sesiju izveidošanas identificējoša simbolu virkne, kas tiek uzrādīta citiem bankas sistēmas serveriem, lai iegūtu piekļuvi to funkciju izpildīšanai.
- refresh_token – unikāla simbolu virkne, kas tiek izmantota, lai iegūtu jaunus sesijas datus, atjaunojot esošu lietotāja sesiju.
- expires_in – skaitlis, kurš apraksta laiku milisekundēs, pēc kura, sākot no sesijas datu izveides brīža, attiecīgais “access_token” lauks vairs nebūs derīgs sesijas izveides identificēšanai.
- refresh_expires_in – skaitlis, kurš apraksta laiku milisekundēs, pēc kura, sākot no datu izveides brīža, attiecīgais “refresh_token” lauks vairs nebūs derīgs, lai atjaunotu esošo lietotāja sesiju.

Lai veicot funkcijas netiktu veikta autorizācija ar jauniem sesijas datiem, kad tas nav nepieciešams, un lieki netiktu veidotas jaunas sesijas, tad izstrādājamai bibliotēkai ir jānodrošina efektīva “access_token” un “refresh_token” lauku datu izmantošana, izvērtējot attiecīgo datu derīguma laikus.

Basket datu struktūra

Viena no datu struktūrām, kas tiek pielietota maksājumu funkcijās ir “basket”, jeb maksājuma iepirkuma groza dati. Šie dati sastāv no vairākiem citiem laukiem un datu struktūrām, tāpēc, lai funkciju apraksti būtu uztveramāki, šie dati un to prasības tiek aprakstītas šajā sadaļā, un citās sadaļās uz šiem datiem tiek veikta atsauce. Visi zemāk minētie kļūdu paziņojumi un to prasības ir apskatāmas šīs nodaļas sadaļā “funkciju kļūdu paziņojumi” (36.lpp).

“Basket” datu tips sastāv no vairākiem laukiem un no “item” datu struktūrām, kā redams att. 2.2.



2.2 att. – **Basket datu struktūras ilustratīvs piemērs**

“Item” datu struktūras ievaddati:

- name (Obligāts) – nenoteikta simbolu virkne.
- price (Obligāts) – *Java Long* datu tipa vērtība, kura ir nenegatīva.
- quantity (Obligāts) – *Java double* datu tipa vērtība, kura ir nenegatīva.
- Nedefinēta lauka nosaukuma un vērtības pāris (Neobligāts) – nenoteiktas simbolu virknes. Šos laukus ir iespējams ievadē pievienot neierobežota skaita reizes.

Prasības “item” datu struktūras ievaddatiem:

- Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 9. kļūdu paziņojums.
- Ja kāds no ievadē padotajiem laukiem ir nepareizajā formātā, tad tiek izvadīts 10. kļūdu paziņojums.
- Kļūdu paziņojumu maksājuma tipa vērtība ir atkarīga no funkcijas, kurā šie dati tiek izmantoti un, kuras aprakstītas sadaļā 2.1.2.

“Item” datu struktūra *JSON* formātā:

```
{
  "name":"<name>",
  "price":<price>,
  "quantity":<quantity>,
  "<nedefinēta_lauka_nosaukums>":
                    "<nedefinēta_lauka_vērtība>"
}
```

- Attiecīgais *JSON object* var saturēt vairāku nedefinētu lauku nosaukuma un vērtības pārus.

“Basket” datu struktūras ievaddati:

- basket_type (Obligāts) – nenoteikta simbolu virkne.
- Nedefinēta lauka nosaukuma un vērtības pāris (Neobligāts) – nenoteiktas simbolu virknes. Šos laukus ir iespējams ievadē pievienot neierobežota skaita reizes.
- items (Obligāts) – “item” datu struktūras vērtības. Šos datus ir iespējams ievadē pievienot neierobežota skaita reizes.

Prasības “basket” datu struktūras ievaddatiem:

- Ja šie dati tiek saņemti no lietotāja operetājsistēmas komandrindas režīma, tad to nepieciešamais ievaddatu formāts ir to *JSON* formāts.
- Laukam “items” ir jāsatur vismaz 1 “item” datu struktūra.
- Ja “items” lauks nav ticis ievadē padots, tad tiek izvadīts 12. kļūdu paziņojums.
- Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 11. kļūdu paziņojums, izņemot priekš lauka “items”.
- Kļūdu paziņojumu maksājuma tipa vērtība ir atkarīga no funkcijas, kurā šie dati tiek izmantoti un, kuras aprakstītas sadaļā 2.1.2.

“Basket” datu struktūra *JSON* formātā:

```
{
  “basket_type”:“<basket_type>”,
  “<nedefinēta_lauka_nosaukums>”:
                    “<nedefinēta_lauka_vērtība>”,
  “items”:[
    <item_JSON_formāts>
  ]
}
```

- Attiecīgais *JSON object* var saturēt vairāku nedefinētu lauku nosaukuma un vērtības pārus.
- *JSON array* ar identifikatoru “items” var saturēt vairākus “item” datu struktūras *JSON* formātus.

Maksājuma saraksts

Funkcija MF.GPL.06 rezultātā atgriež maksājumu sarakstu un, lai vienā *HTTP* atbildē netiktu atgriezts pārlieku liels apjoms ar maksājumu datiem, šis saraksts tiek dalīts sadaļās. Pēc noklusējuma tiek atgriezta 0. sadaļa, kura var saturēt līdz 25 maksājumiem. Lai šos sadaļu parametrus mainītu, ir nepieciešams, veicot attiecīgo funkciju, definēt šādus laukus:

- page[size] – lauks, kurš maina vienas sadaļas maksimālo maksājumu datu skaitu.
- page[number] – lauks, kurš maina atgrieztās sadaļas kārtas numuru.

Veicot attiecīgo funkciju, ir iespējams arī pieprasīt maksājumu serverim šo maksājumu sarakstu atgriezt, saturot maksājumus, kuri atbilst kādiem noteiktiem nosacījumiem jeb filtriem. Lietotājam ir nepieciešama iespēja ievadē šos filtra datus definēt, norādot pieejamo filtru lauka nosaukumu, pieejamo filtrēšanas operāciju un vērtību, pēc kuras datus filtrēt.

Visi iespējamie filtru pieprasījumi tiek veikti šādos formātos, kuru atbalstītie lauku nosaukumi un pieejamās operācijas ir redzamas tabulā 2.1.:

- "filter[<lauka_nosaukums>]=<filtrētā_vērtība>" – šis formāts pieprasa maksājumu serverim atgriezt maksājumu sarakstu, kas satur maksājumus, kuru attiecīgā lauka vērtība sakrīt ar filtrēto vērtību. Šāda formāta pieprasījums pielieto operāciju ar nosaukumu "equals".
- "filter[<lauka_nosaukums>]=less_than(<filtrētā_vērtība>)" – šis formāts pieprasa maksājumu serverim atgriezt maksājumu sarakstu, kas satur maksājumus, kuru attiecīgā lauka vērtība ir mazāka par filtrēto vērtību. Šāda formāta pieprasījums pielieto operāciju ar nosaukumu "less_than".
- "filter[<lauka_nosaukums>]=greater_than(<filtrētā_vērtība>)" – šis formāts pieprasa maksājumu serverim atgriezt maksājumu sarakstu, kas satur maksājumus, kuru attiecīgā lauka vērtība ir lielāka par filtrēto vērtību. Šāda formāta pieprasījums pielieto operāciju ar nosaukumu "greater_than".
- "filter[<lauka_nosaukums>]=between(<filtrētā_vērtība_1>,<filtrētā_vērtība_2>)" – šis formāts pieprasa maksājumu serverim atgriezt maksājumu sarakstu, kas satur maksājumus, kuru attiecīgā lauka vērtība ir lielāka par 1. filtrēto vērtību un mazāka par 2. filtrēto vērtību. Šāda formāta pieprasījums pielieto operāciju ar nosaukumu "between".

2.1. tabula – **filtrējamie lauki un to filtrēšanas operācijas**

Filtrējamā lauka nosaukums	Pieejamās operācijas	Apraksts
acceptor_id	equals	Atgrieztais saraksts satur maksājumus, kuru "acceptor_id" lauks atbilst attiecīgajai vērtībai.
agent_id	equals	Atgrieztais saraksts satur maksājumus, kuru "agent_id" lauks atbilst attiecīgajai vērtībai.
status	equals	Atgrieztais saraksts satur maksājumus, kuru "status" lauks atbilst attiecīgajai vērtībai
created_at	less_than, greater_than, between	Atgrieztā maksājuma saraksta maksājumi tiek filtrēti pēc to "created_at" lauka vērtības, pielietojot vienu no pieejamām operācijām.

HTTP pieprasījumi

Šajā nodaļā tiek aprakstīti visi izstrādājamai bibliotēkai nepieciešamie *HTTP* pieprasījumi, to tipi un nepieciešamie dati. Priekš sīkākas informācijas par šo pieprasījumu izsaucošajām funkcijām skat. sadaļu 2.1.2.

Identifikators	PI.01
Tips	*HTTP/1.1 POST*
URL	
`<authentication_endpoint_URL>`	
***Header* dati**	
• "Content-Type" = "application/x-www-form-urlencoded" • "Accept" = "application/json"	
***Body* dati**	

Attiecīgie dati seko *application/x-www-form-urlencoded* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas.
`grant_type=password&username=<username>&`
`password=<password>&client_id=<client_id>&`
`client_secret=<client_secret>`	

Identifikators	PI.02
Tips	*HTTP/1.1 POST*
URL	
`<authentication_endpoint_URL>`	
***Header* dati**	
• "Content-Type" = "application/x-www-form-urlencoded" • "Accept" = "application/json"	
***Body* dati**	

Attiecīgie dati seko *application/x-www-form-urlencoded* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas.
`grant_type=password&client_id=<client_id>&`
`client_secret=<client_secret>`	

Identifikators	PI.03
Tips	*HTTP/1.1 POST*
URL	
`<authentication_endpoint_URL>`	
***Header* dati**	
• "Content-Type" = "application/x-www-form-urlencoded" • "Accept" = "application/json"	
***Body* dati**	

Attiecīgie dati seko *application/x-www-form-urlencoded* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas.
`grant_type=refresh_token&refresh_token=<refresh_token>&`
`username=<username>&password=<password>&`
`client_id=<client_id>&client_secret=<client_secret>`	

Identifikators	PI.04
Tips	*HTTP/1.1 POST*
URL	
<authentication_endpoint_URL>	
***Header* dati**	
• "Content-Type" = "application/x-www-form-urlencoded" • "Accept" = "application/json"	
***Body* dati**	
Attiecīgie dati seko *application/x-www-form-urlencoded* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas. grant_type=refresh_token&refresh_token=<refresh_token>& client_id=<client_id>&client_secret=<client_secret>	

Identifikators	PI.05
Tips	*HTTP/1.1 POST*
URL	
<transaction_endpoint_URL>	
***Header* dati**	
• "Content-Type" = "application/json" • "Accept" = "application/json" • "Authorization" = "Bearer <access_token>"	
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas. Ja kāda no zemāk minētajām vērtībām nav padota, tad netiek minēts arī tās *JSON* identifikators. *JSON object* ar identifikatoru "attributes" var saturēt vairāku nedefinētu lauku nosaukuma un vērtības pārus.
{
"data":{
"type":"payment",
"attributes":{
"amount":"<amount>",	

```
      "ccy_code":"<ccy_code>",
      "payment_type":"<payment_type>",
      "language":"<language>",
      "description":"<description>",
      "invoice_text":"<invoice_text>",
      "invoice_subject":"<invoice_subject>",
      "ecomm_notify_email":"<ecomm_notify_email>",
      "ecomm_notify_from":"<ecomm_notify_from>",
      "ecomm_notify_cc":"<ecomm_notify_cc>",
      "acceptor_id":"<acceptor_id>",
      "<nedefinēta_lauka_nosaukums>":
                      "<nedefinēta_lauka_vērtība>",
      "pan":"<pan>",
      "cardname":"<cardname>",
      "csc":"<csc>",
      "expiry":"<expiry>",
      "md_status":"<md_status>",
      "xid":"<xid>",
      "sli":"<sli>",
      "cavv":"<cavv>",
      "aav":"<aav>",
      "basket":<basket_JSON_formāts>
      "basket_preauth":<basket_preauth_JSON_formāts>
    }
  },
  "meta":{
    "invoice":<invoice>,
    "confirmed":<confirmed>,
    "finished":<finished>
  }
}
```

Identifikators	PI.06
Tips	*HTTP/1.1 PATCH*
URL	
`<transaction_endpoint_URL>/<payment_reference>/finish`	
***Header* dati**	
• "Content-Type" = "application/json" • "Accept" = "application/json" • "Authorization" = "Bearer <access_token>"	
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas. Ja kāda no zemāk minētajām vērtībām nav padota, tad netiek minēts arī tās *JSON* identifikators.
`{`
`  "data":{`
`  "type":"payment",`
`  "id":"<id>",`
`  "attributes":{`
`  "amount":"<amount>",`
`  "acceptor_id":"<acceptor_ID>",`
`  "basket":<basket_JSON_formāts>`
`  }`
`  }`
`}`	

Identifikators	PI.07
Tips	*HTTP/1.1 PATCH*
URL	
<transaction_endpoint_URL>/<payment_reference>/reversal	
***Header* dati**	
• "Content-Type" = "application/json" • "Accept" = "application/json" • "Authorization" = "Bearer <access_token>"	
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas. Ja kāda no zemāk minētajām vērtībām nav padota, tad netiek minētas arī tās *JSON* identifikators.
{
"data":{
"type":"payment",
"id":"<id>",
"attributes":{
"reversal_amount":"<reversal_amount>",
"suspected_fraud":<suspected_fraud>,
"reversal_basket":<reversal_basket_*JSON*_formāts>
}
}
}	

Identifikators	PI.08
Tips	*HTTP/1.1 POST*
URL	
<transaction_endpoint_URL>/refund	
***Header* dati**	
• "Content-Type" = "application/json" • "Accept" = "application/json" • "Authorization" = "Bearer <access_token>"	
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas. Ja kāda no zemāk minētajām vērtībām nav padota, tad netiek minēts arī tās *JSON* identifikators.
{
"data":{
"type":"payment",
"attributes":{
"original_payment_id":"<payment_reference>",
"amount":"<amount>",
"basket":<basket_JSON_formāts>
}
}
}	

Identifikators	PI.09
Tips	*HTTP/1.1 GET*
URL	
`<transaction_endpoint_URL>/<payment_reference>`	
***Header* dati**	
• "Content-Type" = "application/json" • "Accept" = "application/json" • "Authorization" = "Bearer <access_token>"	
***Body* dati**	
Nav	

Identifikators	PI.10
Tips	*HTTP/1.1 GET*
URL	
`<transaction_endpoint_URL>`	
***Query* dati**	

Attiecīgie dati seko *HTTP query* datu formātam. Attiecīgās vērtības tiek iegūtas no šī pieprasījuma izsaucošās funkcijas. Ja kāda no zemāk minētajām vērtībām nav padota, tad attiecīgais lauks netiek pievienots *HTTP query* datiem.
`page[size]=<page[size]>&page[number]=<page[number]>&`
`filter[acceptor_id]=<filtrējamā_vērtība>&`
`filter[agent_id]=<filtrējamā_vērtība>&`
`filter[status]=<filtrējamā_vērtība>&`
`filter[created_at]=less_than(<filtrējamā_vērtība>)&`
`filter[created_at]=greater_than(<filtrējamā_vērtība>)&`
`filter[created_at]=between(<1._filtrējamā_vērtība>,`
`<2._filtrējamā_vērtība>)`	

***Header* dati**	
• "Content-Type" = "application/json" • "Accept" = "application/json" • "Authorization" = "Bearer <access_token>"	
***Body* dati**	
Nav	

HTTP Atbildes

Šajā nodaļā tiek aprakstītas visas *HTTP* atbildes, to tipi un nepieciešamie dati, kurus ir nepieciešams saņemt un apstrādāt izstrādājamai bibliotēkai. Priekš sīkākas informācijas funkcijām, kuras saņem un apstrādā attiecīgās atbildes skat. sadaļu 2.1.2.

Identifikators	AT.01
Statusa kods	200
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam.
`{`
`  “access_token”:“<access_token>”,`
`  “refresh_token”:“<refresh_token>”,`
`  “expires_in”:<expires_in>,`
`  “refresh_expires_in”:<refresh_expires_in>`
`}`	

Identifikators	AT.02
Statusa kods	400-500
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam.
`{`
`  “error”:“<error>”,`
`  “error_description”:“<error_description>”`
`}`	

Identifikators	AT.03
Statusa kods	200
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. *JSON object* ar identifikatoru “attributes” ir iespējams saturēt vairāku nedefinētu parametru nosaukuma un vērtības pārus. Ja attiecīgie dati satur vairāk nekā šos minētos datus, tad tos ir nepieciešams izlaist.

```
{
  “data”:{
    “id”:“<id>”,
    “type”:“payment”,
    “attributes”:{
      “amount”:“<amount>”,
      “reversed_amount”:“<reversed_amount>”,
      “refunded_amount”:“<refunded_amount>”,
      “ccy_code”:“<ccy_code>”,
      “payment_type”:“<payment_type>”,
      “cardname”:“<cardname>”,
      “acceptor_id”:“<acceptor_id>”,
      “created_at”:“<created_at>”,
      “pan”:“<pan>”,
      “status”:“<status>”,
      “<nedefinēta_parametra_nosaukums>”:
                    “<nedefinēta_parametra_vērtība>”
      “basket”:<basket_datu_JSON_formāts>,
      “basket_preauth”:<basket_preauth_JSON_formāts>,
      “reversal_basket”:<reversal_basket_JSON_formāts>
    }
  }
}
```

Identifikators	AT.04
Statusa kods	400-500
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. *JSON array* ar identifikatoru “errors” var saturēt vairākus *JSON object* datu tipus ar laukiem “title” un “description”.

```
{
  “errors”:[
    {
      “title”:“<title>”,
      “description”:“<description>”
    }
  ],
  “status”:“<status_code>”
}
```

Identifikators	AT.05
Statusa kods	200
***Body* dati**	

Attiecīgie dati seko *JSON* datu formātam. *JSON array* ar identifikatoru "data" var saturēt vairākus attiecīgos *JSON object* datu tipus, kur katrs sakrīt ar atbildes AT.03 *body* datu *JSON object* ar identifikatoru "data" datu tipiem.

```
{
  "data":[
    {
      "id":"<id>",
      "type":"payment",
      "attributes":{
        "amount":"<amount>",
        "reversed_amount":"<reversed_amount>",
        "refunded_amount":"<refunded_amount>",
        "ccy_code":"<ccy_code>",
        "payment_type":"<payment_type>",
        "cardname":"<cardname>",
        "acceptor_id":"<acceptor_id>",
        "created_at":"<created_at>",
        "pan":"<pan>",
        "status":"<status>",
        "<nedefinēta_parametra_nosaukums>":
                        "<nedefinēta_parametra_vērtība>",
        "basket":<basket_datu_JSON_formāts>,
        "basket_preauth":<basket_preauth_JSON_formāts>,
        "reversal_basket":<reversal_basket_JSON_formāts>
    }
  ]
}
```

Funkciju kļūdu paziņojumi

Bibliotēkas funkciju kļūdainu darbību rezultātā tiek izvadīti šim produktam specifiski, definēti kļūdu paziņojumi, kuriem ir nepieciešams būt *Java Exception* klases objektiem, kuru saturi un prasības tiek aprakstītas šajā sadaļā. Visi citi *Exception* klases objekti, kuri nav daļa no šīs bibliotēkas definētajiem un, kurus ir nepieciešams apstrādāt, arī tiek izvadīti lietotājam.

1. Expected configuration entry “<lauka_nosaukums>” is missing a value.
2. Configuration entry “<lauka_nosaukums>” is incorrectly formatted.
3. Configuration entry “<lauka_nosaukums>” can only be either “password” or “certificate”.
4. <maksājuma_tips> request is missing field “<lauka_nosaukums>”.
5. <maksājuma_tips> request field “<lauka_nosaukums>” is incorrectly formatted.
6. <maksājuma_tips> request is missing either field “<lauka_nosaukums>” or “<lauka_nosaukums>”.
7. <maksājuma_tips> request can not have both “<lauka_nosaukums>” and “<lauka_nosaukums>” fields.
8. Payment type can only be either “SMS” or “DMS”.
9. <maksājuma_tips> request basket item is missing field “<lauka_nosaukums>”.
10. Basket item field “<lauka_nosaukumks>” is incorrectly formatted.
11. <maksājuma_tips> request basket is missing field “<lauka_nosaukums>”.
12. <maksājuma_tips> request basket must have at least 1 item.
13. Tiek izvadīts kļūdu paziņojums, kura paziņojuma saturs sakrīt ar saņemtās *HTTP* atbildes *body* datu lauka “error” vērtību. Tiek dota lietotājam iespēja no kļūdu paziņojuma pirmkodā iegūt visus laukus un atbildes statusa kodu.
14. Kļūdu paziņojumam tiek izmantoti attiecīgās *HTTP* atbildes *body* datu lauki “status” un *JSON array* ar identifikatoru “errors” datu struktūras pirmās *JSON object* datu struktūras lauki “title” un “description”. Tiek izvadīts kļūdu paziņojums, kura paziņojuma saturs sakrīt ar lauka “title” vērtību. Tiek dota lietotājam iespēja no kļūdu paziņojuma pirmkodā iegūt attiecīgos “title”, “description” un atbildes statusa kodu, kurš iegūstams arī no lauka “status”.

15. Tiek izvadīts kļūdu paziņojums, kura paziņojuma saturs sakrīt ar saņemtās *HTTP* atbildes statusa kodu. Tiek dota lietotājam iespēja no kļūdu paziņojuma pirmkodā iegūt attiecīgās atbildes statusa kodu.
16. Could not verify access token, possible auth server malfunction. Tiek dota lietotājam iespēja no kļūdu paziņojuma pirmkodā iegūt attiecīgās *HTTP* atbildes statusa kodu.
17. Purchase request card is expired.
18. Can not send 3dSecure data without card data.

Kļūdu paziņojumu pieļāvumi:

- Attiecīgo bibliotēkas definēto *Exception* klašu iedalījumus un nosaukumus ir atļauts veidot pēc izstrādātāju sprieduma.
- Visus kļūdu paziņojumus, kuri ir specifiski lietotāja komandrindas saskarnes režīmam, ir atļauts veidot pēc izstrādātāju sprieduma.
- Šo kļūdu paziņojumu izsaukšana nav obligāta, ja ir iespējams izstrādāt pirmkodu, lai kāda attiecīgo kļūdu paziņojuma izsaukšana nebūtu iespējama.

2.1.2 Funkciju apraksti

Funkcijas identifikators	CM.LC.01
Funkcijas nosaukums	Load configuration
Ievade	

Ja funkcija tiek veikta, izmantojot lietotāja pirmkodu kā saskarni, tad šie dati tiek iegūti no lietotāja pirmkoda vai no konfigurācijas teksta faila, norādot tā atrašanās vietu sistēmā.
Ja funkcija tiek veikta, izmantojot lietotāja sistēmas komandrindas režīmu kā saskarni, tad šie dati tiek iegūti tikai no konfigurācijas teksta faila, norādot tā atrašanās vietu sistēmā.

- Autorizācijas funkciju konfigurācijas ievaddati:
    - endpoint_url (Obligāts) – nenoteikta simbolu virkne.
    - client_id (Obligāts) – nenoteikta simbolu virkne.
    - client_secret (Obligāts) – nenoteikta simbolu virkne.
    - auth_mode (Obligāts) – simbolu virkne, kura var sakrist tikai ar “password” vai “certificate”.
    - username (Pēc nosacījuma) – nenoteikta simbolu virkne.
    - password (Pēc nosacījuma) – nenoteikta simbolu virkne.
    - keystore_path (Pēc nosacījuma) – nenoteikta simbolu virkne.
    - keystore_type (Neobligāts) – nenoteikta simbolu virkne.
    - keystore_password (Neobligāts) – nenoteikta simbolu virkne.
    - key_alias (Pēc nosacījuma) – nenoteikta simbolu virkne.
    - key_password (Neobligāts) – nenoteikta simbolu virkne.
    - truststore_path (Pēc nosacījuma) – nenoteikta simbolu virkne.
    - truststore_type (Neobligāts) – nenoteikta simbolu virkne.
    - truststore_password (Neobligāts) – nenoteikta simbolu virkne.
    - connection_timeout (Neobligāts) – nenegatīvs skaitlis.
    - response_timeout (Neobligāts) – nenegatīvs skaitlis.
    - maximum_concurrent_connections (Neobligāts) – pozitīvs skaitlis.
- Maksājumu funkciju konfigurācijas ievaddati:
    - endpoint_url (Obligāts) – nenoteikta simbolu virkne.
    - keystore_path (Pēc nosacījuma) – nenoteikta simbolu virkne.
    - keystore_type (Neobligāts) – nenoteikta simbolu virkne.

- keystore_password (Neobligāts) – nenoteikta simbolu virkne.
- key_alias (Pēc nosacījuma) – nenoteikta simbolu virkne.
- key_password (Neobligāts) – nenoteikta simbolu virkne.
- truststore_path (Pēc nosacījuma) – nenoteikta simbolu virkne.
- truststore_type (Neobligāts) – nenoteikta simbolu virkne.
- truststore_password (Neobligāts) – nenoteikta simbolu virkne.
- connection_timeout (Neobligāts) – nenegatīvs skaitlis.
- response_timeout (Neobligāts) – nenegatīvs skaitlis.
- maximum_concurrent_connections (Neobligāts) – pozitīvs skaitlis.

Prasības autorizācijas funkciju konfigurācijas ievaddatiem:

- Ja lauka "auth_mode" saturs ir "password", tad lauki "username" un "password" ir obligāti, bet lauki "keystore_path" un "key_alias" ir neobligāti.
- Ja lauka "auth_mode" saturs ir "certificate", tad lauki "keystore_path" un "key_alias" ir obligāti, bet lauki "username" un "password" ir neobligāti.

Prasības autorizācijas un maksājumu funkciju konfigurācijas ievaddatiem:

- Ja viens no laukiem "keystore_path", "keystore_type", "keystore_password", "key_alias" vai "key_password" ir ticis ievadē padots, tad lauki "keystore_path" un "key_alias" ir obligāti.
- Ja viens no laukiem "truststore_path", "truststore_type" vai "truststore_password" ir ievadē ticis padots, tad lauks "truststore_path" ir obligāts.
- Ja neobligātie lauki "keystore_type" vai "truststore_type" nav tikuši ievadē padoti, tad to vērtība tiek pieņemta kā "PKCS12".
- Ja neobligātie lauki "keystore_password", "key_password" vai "truststore_password" nav tikuši ievadē padoti, tad to vērtība tiek pieņemta kā tukša simbolu virkne.
- Ja neobligātie lauki "connection_timeout" vai "response_timeout" nav tikuši ievadē padoti, tad to vērtības tiek pieņemtas kā 0.
- Ja neobligātais lauks "maximum_concurrent_connections" nav ticis ievadē padots, tad tā vērtība tiek pieņemta kā 20.
- Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 1. kļūdu paziņojums.

- Ja lauks “auth_mode” ir ticis ievadē padots nepareizajā formātā, tad tiek izvadīts 3. kļūdu paziņojums.
- Ja kāds no laukiem ir ticis ievadē padots nepareizajā formātā, tad tiek izvadīts 2. kļūdu paziņojums, izņemot priekš lauka “auth_mode”.

Apstrāde

Visi ievadē padotie lauki tiek saglabāti operatīvajā atmiņā, lai tos ir iespējams izmantot, veicot citas funkcijas.

Apstrāde autorizācijas un maksājumu funkciju konfigurācijas ievaddatiem:

- Tiek uzstādīta *HTTP* pieprasījumu veikšana priekš autorizācijas un maksājumu funkciju veikšanas atsevišķi, izmantojot katra attiecīgos konfigurācijas ievaddatus:
    - Izmantojot lauka “connection_timeout” vērtību, attiecīgajiem *HTTP* pieprasījumiem tiek nodrošināts savienojuma izveidošanas termiņš milisekundēs. Ja uz pieprasījuma adresi neizdodas izveidot *HTTP* savienojumu laikā, ko norāda šī vērtība, tad attiecīgās funkcijas darbība tiek pārtraukta. Ja šī lauka vērtība ir 0, tad attiecīgais savienojuma izveidošanas termiņš tiek uzstādīts neierobežoti ilgs.
    - Izmantojot lauka “response_timeout” vērtību, attiecīgajiem *HTTP* pieprasījumiem tiek nodrošināts *HTTP* atbildes saņemšanas termiņš milisekundēs. Ja no pieprasījuma adreses neizdotas saņemt *HTTP* atbildi laikā, ko norāda šī vērtība, tad attiecīgās funkcijas darbība tiek pārtraukta. Ja šī lauka vērtība ir 0, tad attiecīgais atbildes saņemšanas termiņš tiek uzstādīts neierobežoti ilgs.
    - Izmantojot lauka “maximum_concurrent_connections” vērtību, attiecīgajiem *HTTP* pieprasījumiem tiek nodrošināts ierobežojums tam, cik vienlaicīgu šādu pieprasījumu ir atļauts veikt.
    - Attiecīgie *HTTP* pieprasījumi tiek veikti nodrošinot *TLS*, izmantojot protokolus *TLSv1.1* un *TLSv1.2*.
    - Ja lauki “keystore_path” un “key_alias” ir tikuši ievadē padoti, tad priekš *HTTP* pieprasījumiem tiek pievienota atslēgu pāru izmantošana, kur:
        - Lauks “keystore_path” ir attiecīgā atslēgu pāra *Keystore* faila atrašanās vieta sistēmā.

▪ Lauks “keystore_type” ir attiecīgā *Keystore* faila formāts. ▪ Lauks “keystore_password” ir attiecīgā *Keystore* faila piekļuves parole. ▪ Lauks “key_alias” ir attiecīgā atslēgu pāra *alias* vērtība. ▪ Lauks “key_password” ir attiecīgā atslēgu pāra piekļuves parole. o Ja lauks “truststore_path” ir ticis ievadē padots, tad priekš *HTTP* pieprasījumiem tiek izmantota uzticamo sertifikātu izmantošana, kur: ▪ Lauks “truststore_path” ir attiecīgā uzticamo sertifikātu *Keystore* faila atrašanās vieta sistēmā. ▪ Lauks “truststore_type” ir attiecīgā *Keystore* faila formāts. ▪ Lauks “truststore_password” ir attiecīgā *Keystore* faila piekļuves parole.
Izvade
Nav

Funkcijas identifikators	AF.CNS.01
Funkcijas nosaukums	Create new session
Ievade	
No operatīvajā atmiņā glabātajiem autorizācijas konfigurācijas datiem, kas radušies funkcijas CM.LC.01 rezultātā, tiek iegūti šādi dati: • endpoint_url. • client_id. • client_secret. • auth_mode. • username. • password. • keystore_path. • key_alias.	
Apstrāde	
Ja lauks "auth_mode" ir ar saturu "password", tad tiek iegūts identifikators, kurš sakrīt ar lauka "username" vērtību, un tiek veikts *HTTP* pieprasījums PI.01, izmantojot ievadē iegūtos datus. Ja lauks "auth_mode" ir ar saturu "certificate", tad tiek iegūts identifikators, kurš sakrīt ar lauku "keystore_path" un "key_alias" saturu apvienojumu, un tiek veikts *HTTP* pieprasījums PI.02, izmantojot ievadē iegūtos datus. Ja pretī tiek saņemta *HTTP* atbilde AT.01: • Attiecīgie atbildes *body* sesijas dati tiek saglabāti operatīvajā atmiņā, izmantojot iegūto identifikatoru, lai tos iespējams izmantot, veicot citas funkcijas. Ja operatīvajā atmiņā jau tiek glabāti sesijas dati ar attiecīgo identifikatoru, tad tie tiek aizvietoti. Ja pretī tiek saņemta *HTTP* atbilde AT.02: • Tiek izvadīts 13. kļūdu paziņojums, izmantojot atbildē iegūtos datus. Ja pretī tiek saņemta cita *HTTP* atbilde: • Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.	
Izvade	
Nav	

Funkcijas identifikators	AF.RS.02
Funkcijas nosaukums	Refresh session
Ievade	
No operatīvajā atmiņā glabātajiem konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati: • endpoint_url. • client_id. • client_secret. • auth_mode. • username. • password. • keystore_path. • key_alias. No funkcijas, kura šo funkciju ir izsaukusi: • refresh_token.	
Apstrāde	
Ja lauks “auth_mode” ir ar saturu “password”, tad tiek iegūts identifikators, kurš sakrīt ar lauka “username” vērtību, un tiek veikts *HTTP* pieprasījums PI.03, izmantojot ievadē iegūtos datus. Ja lauks “auth_mode” ir ar saturu “certificate”, tad tiek iegūts identifikators, kurš sakrīt ar lauku “keystore_path” un “key_alias” saturu apvienojumu, un tiek veikts *HTTP* pieprasījums PI.04, izmantojot ievadē iegūtos datus. Ja pretī tiek saņemta *HTTP* atbilde AT.01: • Attiecīgie atbildes *body* sesijas dati tiek saglabāti operatīvajā atmiņā, izmantojot iegūto identifikatoru, lai tos iespējams izmantot, veicot citas funkcijas. Ja operatīvajā atmiņā jau tiek glabāti sesijas dati ar attiecīgo identifikatoru, tad tie tiek aizvietoti. Ja pretī tiek saņemta *HTTP* atbilde AT.02: • Tiek izvadīts 13. kļūdu paziņojums, izmantojot atbildē iegūtos datus. Ja pretī tiek saņemta cita *HTTP* atbilde: • Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.	
Izvade	
Nav	

Funkcijas identifikators	AF.GB.03
Funkcijas nosaukums	Get bearer
Ievade	
No operatīvajā atmiņā glabātajiem autorizācijas konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati: • auth_mode. • username. • keystore_path. • key_alias.	
Apstrāde	
Ja lauks “auth_mode” ir ar saturu “password”, tad tiek iegūts identifikators, kurš sakrīt ar lauka “username” vērtību. Ja lauks “auth_mode” ir ar saturu “certificate”, tad tiek iegūts identifikators, kurš sakrīt ar lauku “keystore_path” un “key_alias” saturu apvienojumu. Tiek veikta pārbaude vai operatīvajā atmiņā pašlaik glabājās sesijas dati, kuru identifikators sakrīt ar iegūto identifikatoru. • Ja operatīvajā atmiņā attiecīgie dati neglabājas, tad tiek veikta funkcija create new session (AF.CNS.01). • Ja operatīvajā atmiņā attiecīgie dati glabājas, tad tiem tiek veikta lauku derīgumu pārbaude: o Ja to “access_token” un “refresh_token” lauki nav derīgi, tad tiek veikta funkcija create new session (AF.CNS.01). o Ja to “access_token” lauks nav derīgs, bet “refresh_token” lauks ir derīgs, tad tiek veikta funkcija refresh session (AF.RS.02), tai ievadē padodot lauku “refresh_token” no attiecīgajiem sesijas datiem.	
Izvade	
Tiek atgriezts lauks “access_token” no tiem operatīvajā atmiņā glabātajiem sesijas datiem, kuru identifikators sakrīt ar apstrādē iegūto identifikatoru.	

Funkcijas identifikators	MF.MP.01
Funkcijas nosaukums	Make purchase
Ievade	

No operatīvajā atmiņā glabātajiem maksājumu konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati:

- endpoint_url.

Tiek veikta funkcija AF.GB.03 un no tās izvades iegūti šādi dati:

- access_token.

No lietotāja tiek iegūti šādi dati:

- Vispārējie maksājuma dati:
  - amount (Obligāts) – *Long* datu tipa vērtība, kura ir nenegatīva.
  - ccy_code (Obligāts) – trīs nenegatīvi cipari. Priekš sīkākas informācijas par lauka formātu skat. *ISO* 4217 valūtas kodu standartu [3].
  - payment_type (Neobligāts) – simbolu virkne, kura var sakrist tikai ar "SMS" vai "DMS".
  - language (Neobligāts) – nenoteikta simbolu virkne.
  - description (Neobligāts) – nenoteikta simbolu virkne.
  - invoice_text (Neobligāts) – nenoteikta simbolu virkne.
  - ecomm_notify_email (Pēc nosacījuma) – nenoteikta simbolu virkne.
  - ecomm_notify_from (Neobligāts) – nenoteikta simbolu virkne.
  - ecomm_notify_cc (Neobligāts) – nenoteikta simbolu virkne.
  - invoice_subject (Neobligāts) – nenoteikta simbolu virkne.
  - acceptor_id (Neobligāts) – nenoteikta simbolu virkne.
  - Nedefinēta parametra nosaukums un vērtība (Neobligāts) – nenoteiktas simbolu virknes. Šos laukus ir iespējams ievadē pievienot neierobežota skaita reizes.
  - basket (Neobligāts) – "basket" datu struktūra.
- Kartes dati:
  - pan (Pēc nosacījuma) – nenoteikta simbolu virkne.
  - cardname (Pēc nosacījuma) – nenoteitka simbolu virkne.
  - csc (Pēc nosacījuma) – 3-4 ciparu skaitlis, kas var sākties ar 0.

- expiry (Pēc nosacījuma) – 4 cipari, kuri identificē datumu un seko formātam "<YY><MM>" (YY – gads divciparu formātā, MM – mēnesis divciparu formātā).

- *3dSecure* dati:
  - cavv (Pēc nosacījuma) – nenoteikta simbolu virkne.
  - aav (Pēc nosacījuma) – nenoteikta simbolu virkne.
  - xid (Pēc nosacījuma) – nenoteikta simbolu virkne.
  - md_status (Pēc nosacījuma) – nenoteikta simbolu virkne.
  - sli (Pēc nosacījuma) – nenoteikta simbolu virkne.
- Maksājuma statusa dati:
  - invoice (Neobligāts) – *Boolean* datu tipa vērtība.
  - confirmed (Neobligāts) – *Boolean* datu tipa vērtība.
  - finished (Neobligāts) – *Boolean* datu tipa vērtība.

Prasības ievaddatiem:

- Visi zemāk minētie kļūdu paziņojumi tiek veikti ar to maksājuma tipa vērtību "purchase".
- Ja maksājuma statusa datu lauks "invoice" ir ar vērtību "true", tad vispārējo maksājumu datu lauks "ecomm_notify_email" ir obligāts.
- Ja viens no kartes datiem ir ievadē ticis padots, tad tie visi ir obligāti.
- *3dSecure* dati ir atļauti tikai, ja visi kartes dati ir tikuši ievadē padoti. Ja šie dati tiek padoti, kad kartes dati nav ievadē padoti, tad tiek izvadīts 18. kļūdu paziņojums.
- Ja viens no *3dSecure* datiem ir ievadē ticis padots, tad *3dSecure* datu lauki "xid", "md_status" un "sli" ir obligāti.
- Ja viens no *3dSecure* datiem ir ievadē ticis padots, tad viens no *3dSecure* datu laukiem "cavv" vai "aav" ir obligāts. Ja neviens no abiem ievadē netiek padots, kad tie ir obligāti, tad tiek izvadīts 6. kļūdu paziņojums.
- *3dSecure* datu lauki "cavv" un "aav" nedrīkst ievadē tikt padoti vienlaikus. Ja šie lauki ir ievadē padoti vienlaikus, tad tiek izvadīts 7. kļūdu paziņojums.
- Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 4. kļūdu paziņojums, izņemot priekš *3dSecure* laukiem "cavv" un "aav".

- Ja lauka “payment_type” vērtība nav tikusi padota, tad tā vērtība tiek pieņemta kā “SMS”.
- Ja lauks “payment_type” ir ievadē ticis padots nepareizajā formātā, tad tiek izvadīts 8. kļūdu paziņojums.
- Ja kāds no ievadē padotajiem datiem ir nepareizajā formātā, tad tiek izvadīts 5. kļūdu paziņojums, izņemot priekš lauka “payment_type”.
- Ja lauka “expiry” ievadītais datums ir senāks par pašreizējo datumu, tad tiek izvadīts 17. kļūdu paziņojums.

Apstrāde

Ja lauka “payment_type” vērtība ir “DMS”, tad ievadē padotais “basket” lauks tiek izmantots pieprasījumu veikšanā ar lauka nosaukumu “basket_preauth”.

Tiek veikts HTTP pieprasījums PI.05, izmantojot ievadē iegūtos datus.

Ja pretī tiek saņemta *HTTP* atbilde AT.03:

- Tiek veikta šīs funkcijas izvade.

Ja pretī tiek saņemta *HTTP* atbilde AT.04

- Tiek izvadīts 14. kļūdu paziņojums, izmantojot atbildē iegūtos datus.

Ja pretī tiek saņemta cita *HTTP* atbilde:

- Ja attiecīgās *HTTP* atbildes statusa kods ir 302., tad tiek izvadīts 16. kļūdu paziņojums.
- Pretēji, tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.

Izvade

Lietotājam tiek atgriezti tie *HTTP* atbildes AT.03 *body* dati, kurus satur *JSON object* datu struktūra ar identifikatoru “data”.

Funkcijas identifikators	MF.FP.02
Funkcijas nosaukums	Finish payment

Ievade

No operatīvajā atmiņā glabātajiem maksājumu konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati:

- endpoint_url.

Tiek veikta funkcija AF.GB.03 un no tās izvades iegūti šādi dati:

- access_token.

No lietotāja tiek iegūti šādi dati:

- id (Obligāts) – nenoteikta simbolu virkne.
- amount (Neobligāts) – *Long* datu tipa vērtība.
- acceptor_id (Neobligāts) – nenoteikta simbolu virkne.
- basket (Neobligāts) – "basket" datu struktūra.

Prasības ievaddatiem:

- Visi zemāk minētie kļūdu paziņojumi tiek veikti ar to maksājuma tipa vērtību "finish".
- Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 4. kļūdu paziņojums.
- Ja kāds no ievadē padotajiem datiem ir nepareizajā formātā, tad tiek izvadīts 5. kļūdu paziņojums.

Apstrāde

Tiek veikts HTTP pieprasījums PI.06, izmantojot ievadē iegūtos datus.

Ja pretī tiek saņemta *HTTP* atbilde AT.03:

- Tiek veikta šīs funkcijas izvade.

Ja pretī tiek saņemta *HTTP* atbilde AT.04

- Tiek izvadīts 14. kļūdu paziņojums, izmantojot atbildē iegūtos datus.

Ja pretī tiek saņemta cita *HTTP* atbilde:

- Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.

Izvade

Lietotājam tiek atgriezti tie *HTTP* atbildes AT.03 *body* dati, kurus satur *JSON object* datu struktūra ar identifikatoru "data".

Funkcijas identifikators	MM.RVP.03
Funkcijas nosaukums	Reverse payment
Ievade	
No operatīvajā atmiņā glabātajiem maksājumu konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati: • endpoint_url. Tiek veikta funkcija AF.GB.03 un no tās izvades iegūti šādi dati: • access_token. No lietotāja tiek iegūti šādi dati: • id (Obligāts) – nenoteikta simbolu virkne. • acceptor_id (Neobligāts) – nenoteikta simbolu virkne. • reversal_amount (Neobligāts) – *Long* datu tipa vērtība. • suspected_fraud (Neobligāts) – *Boolean* datu tipa vērtība. • basket (Neobligāts) – "basket" datu struktūra. Prasības ievaddatiem: • Visi zemāk minētie kļūdu paziņojumi tiek veikti ar to maksājuma tipa vērtību "reversal". • Lauks "reversal_amount" nedrīkst tikt ievadē padots un lauka "suspected_fraud" vērtība nedrīkst būt "true" vienlaikus. Ja šī prasība netiek ievērota, tad tiek izvadīts 7. kļūdu paziņojums. • Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 4. kļūdu paziņojums. • Ja kāds no ievadē padotajiem datiem ir nepareizajā formātā, tad tiek izvadīts 5. kļūdu paziņojums.	

Apstrāde
Tiek veikts HTTP pieprasījums PI.07, izmantojot ievadē iegūtos datus. Ja pretī tiek saņemta *HTTP* atbilde AT.03: • Tiek veikta šīs funkcijas izvade. Ja pretī tiek saņemta *HTTP* atbilde AT.04 • Tiek izvadīts 14. kļūdu paziņojums, izmantojot atbildē iegūtos datus. Ja pretī tiek saņemta cita *HTTP* atbilde: • Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.
Izvade
Lietotājam tiek atgriezti tie *HTTP* atbildes AT.03 *body* dati, kurus satur *JSON object* datu struktūra ar identifikatoru “data”.

Funkcijas identifikators	MM.RFP.04
Funkcijas nosaukums	Refund payment

Ievade

No operatīvajā atmiņā glabātajiem maksājumu konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati:

- endpoint_url.

Tiek veikta funkcija AF.GB.03 un no tās izvades iegūti šādi dati:

- access_token.

No lietotāja tiek iegūti šādi dati:

- original_payment_id (Obligāts) – nenoteikta simbolu virkne.
- amount (Neobligāts) – *Long* datu tipa vērtība.
- basket (Neobligāts) – "basket" datu struktūra.

Prasības ievaddatiem:

- Visi zemāk minētie kļūdu paziņojumi tiek veikti ar to maksājuma tipa vērtību "refund".
- Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 4. kļūdu paziņojums.
- Ja kāds no ievadē padotajiem datiem ir nepareizajā formātā, tad tiek izvadīts 5. kļūdu paziņojums.

Apstrāde

Tiek veikts HTTP pieprasījums PI.08, izmantojot ievadē iegūtos datus.

Ja pretī tiek saņemta *HTTP* atbilde AT.03:

- Tiek veikta šīs funkcijas izvade.

Ja pretī tiek saņemta *HTTP* atbilde AT.04

- Tiek izvadīts 14. kļūdu paziņojums, izmantojot atbildē iegūtos datus.

Ja pretī tiek saņemta cita *HTTP* atbilde:

- Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.

Izvade

Lietotājam tiek atgriezti tie *HTTP* atbildes AT.03 *body* dati, kurus satur *JSON object* datu struktūra ar identifikatoru "data".

Funkcijas identifikators	MM.GP.05
Funkcijas nosaukums	Get payment
Ievade	
No operatīvajā atmiņā glabātajiem maksājumu konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati: • endpoint_url. Tiek veikta funkcija AF.GB.03 un no tās izvades iegūti šādi dati: • access_token. No lietotāja tiek iegūti šādi dati: • id (Obligāts) – nenoteikta simbolu virkne. Prasības ievaddatiem: • Visi zemāk minētie kļūdu paziņojumi tiek veikti ar to maksājuma tipa vērtību "payment retrieval". • Ja kāds no obligātajiem laukiem nav ticis ievadē padots, tad tiek izvadīts 4. kļūdu paziņojums.	
Apstrāde	
Tiek veikts HTTP pieprasījums PI.09, izmantojot ievadē iegūtos datus. Ja pretī tiek saņemta *HTTP* atbilde AT.03: • Tiek veikta šīs funkcijas izvade. Ja pretī tiek saņemta *HTTP* atbilde AT.04 • Tiek izvadīts 14. kļūdu paziņojums, izmantojot atbildē iegūtos datus. Ja pretī tiek saņemta cita *HTTP* atbilde: • Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.	
Izvade	
Lietotājam tiek atgriezti tie *HTTP* atbildes AT.03 *body* dati, kurus satur *JSON object* datu struktūra ar identifikatoru "data".	

Funkcijas identifikators	MM.GPL.06
Funkcijas nosaukums	Get payment list
Ievade	

No operatīvajā atmiņā glabātajiem maksājumu konfigurācijas datiem, kas radušies funkcijas CF.LC.01 rezultātā, tiek iegūti šādi dati:

- endpoint_url.

Tiek veikta funkcija AF.GB.03 un no tās izvades iegūti šādi dati:

- access_token.

No lietotāja tiek iegūti šādi dati:

- page[size] (Neobligāts) – skaitlis, kura minimālā vērtība ir 1 un maksimālā 100.
- page[number] (Neobligāts) – nenegatīvs skaitlis.
- filter[acceptor_id]:
    - Filtrējamā vērtība priekš “equals” operācijas (Neobligāts) – nenoteikta simbolu virkne.
- filter[agent_id]:
    - Filtrējamā vērtība priekš “equals” operācijas (Neobligāts) – nenoteikta simbolu virkne.
- filter[status]:
    - Filtrējamā vērtība priekš “equals” operācijas (Neobligāts) – nenoteikta simbolu virkne.
- filter[created_at]:
    - Filtrējamā vērtība priekš “less_than” operācijas (Neobligāts) – nenoteikta simbolu virkne.
    - Filtrējamā vērtība priekš “greater_than” operācijas (Neobligāts) – nenoteikta simbolu virkne.
    - Filtrējamās vērtības priekš “between” operācijas (Neobligāts) – nenoteiktas simbolu virknes.

Prasības ievaddatiem:

- Visi zemāk minētie kļūdu paziņojumi tiek veikti ar to maksājuma tipa vērtību “payment list retrieval”.

• Ja kāds no ievadē padotajiem datiem ir nepareizajā formātā, tad tiek izvadīts 5. kļūdu paziņojums.
Apstrāde
Tiek veikts HTTP pieprasījums PI.10, izmantojot ievadē iegūtos datus. Ja pretī tiek saņemta *HTTP* atbilde AT.05: • Tiek veikta šīs funkcijas izvade. Ja pretī tiek saņemta *HTTP* atbilde AT.04 • Tiek izvadīts 14. kļūdu paziņojums, izmantojot atbildē iegūtos datus. Ja pretī tiek saņemta cita *HTTP* atbilde: • Tiek izvadīts 15. kļūdu paziņojums, izmantojot atbildē iegūtos datus.
Izvade
Lietotājam tiek atgriezti tie *HTTP* atbildes AT.05 *body* dati, kurus satur *JSON array* datu struktūra ar identifikatoru “data”.

2.2. Nefunkcionālās prasības

2.2.1. Veiktspējas prasības

Visas zemāk minētās veiktspējas prasības attiecas tikai uz šī produkta lietošanu, izmantojot pirmkoda saskarnes režīmu.

1. Ja jebkuru funkciju veic 10 pavedieni vienlaicīgi, tad ir nepieciešams nodrošināt, ka bibliotēkas palielinātais sistēmas aizņemtās *heap* operatīvās atmiņas daudzums nepārsniedz 50 MB pēc katras maksājumu funkcijas veikšanas.
2. Ir nepieciešams nodrošināt pēc iespējas mazāku skaitu *HTTP* savienojumu veikšanas, kur iespējams.
3. Kad tirgotāju sistēma ir pilnībā iedarbināta, tad ir nepieciešams nodrošināt jebkuras maksājumu funkcijas izpildes laiku ne ilgāku par 1 sekundi, neskaitot *HTTP* savienojuma izveides un atbildes saņemšanas laiku.
4. Šai programmatūrai ir jānodrošina *thread-safety* jeb drošība un funkciju pareiza izpilde, izpildoties vairākiem paralēliem procesiem jeb pavedieniem, kur tas ir nepieciešams. Vairāku paralēlu procesu izpildi ir nepieciešams nodrošināt ar pēc iespējas mazāk pavedienu darbības īslaicīgu apturēšanu, kur iespējams.

2.2.2. Uzturamība

Bankas sistēmai, ar kuru šim produktam ir nepieciešams sazināties, ir raksturīgi prasībām mainīties, saistībā ar ievaddatu nepieciešamības nosacījumiem un *HTTP* pieprasījumu un atbilžu *body* datu *JSON* formātu lauku identifikatoru un *HTTP query* datu nosaukumu vērtībām. Šī iemelsa dēļ, šī produkta pirmkodu ir nepieciešams izstrādāt tā, lai attiecīgos nosacījumus un vērtības būtu ērti mainīt.

2.2.3. Pārnesamība

Gala produktam ir nepieciešams būt vienam failam *.jar* formātā, kuru ir iespējams bez ierobežojumiem pārlikt un izmantot uz citām sistēmām.

2.2.4. Lietotāja saskarne

Lietotāja saskarnei caur pirmkodu ir jābūt ērtai un saprotamai, lietojot OOP pieejas vadlīnijas priekš funkciju ievaddatu konstruēšanas, izvaddatu pasniegšanas un maksājumu funkciju veikšanas.

Lietotāja saskarnei caur lietotāja sistēmas komandrindas režīma, lietotājam ievaddatus padodot kā šī produkta *.jar* faila palaišanas parametrus, ir nepieciešams sekot *UNIX Shell* komandrindu komandu parametru padošanas vadlīnijām. Izvaddatiem šajā saskarnē ir nepieciešams tikt izvadītiem komandrindas režīma logā pārskatāmā formātā, norādot katru lauku nosaukumus un attiecīgās vērtības.

2.2.5. Produkta izstrādes prasības

Programmatūrai ir jābūt izstrādātai *Java* programmēšanas valodā, izmantojot *Java JDK 8* vai augstāku versiju, atbilstība ar vecākām versijām nav nepieciešama.

Produkta pirmkoda izstrādē ir jāizmanto modernas un pierādītas metodes, kas nodrošina pirmkoda lasāmību un veiktspēju.

3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

3.1. Lietotāja saskarņu projektējums

3.1.1. Pirmkoda saskarne

Sekojot attiecīgā programmatūras prasību specifikācijas dokumenta prasībām, lietotāja saskarne no pirmkoda tiek izstrādāta, lietojot OOP pieeju, kuras vadlīnijas prasa loģiski saistītu pirmkodu kodu vienot klasēs, kurām ir savi attiecīgie lauki un darbības tiek veiktas ar metodēm. Šajā sadaļā tiek aprakstītas tās klases, metodes un to datu formāti, ar kurām lietotājam ir paredzēta tieša saskarne. Priekš informācijas par *builder patter* struktūras, kas tiek pielietota vairāku zemāk minēto klašu metodēs, skatīt avotu [12].

MerchantConnector – šī klase ir galvenais piekļuves punkts visām bibliotēkas funkcijām un vieta, kur nepieciešamos konfigurācijas datus ir nepieciešams padot.

- Piekļuve:
  - `MerchantConnector(String confFilePath)` – klases konstruktors, kurš parametros saņem konfigurācijas teksta faila atrašanās vietu lietotāja sistēmā, kurš satur nepieciešamos konfigurācijas datus.
  - `MerchantConnector(Map<String, String> confMap)` – klases konstruktors, kurš parametros saņem konfigurācijas datus, ievietotus *Java Map* datu struktūras objektā.
- Metodes:
  - `Transactions transactions()` – atgriež lietotājam jaunu Transactions klases objektu.

Transcations – šī klase ir piekļuves punkts maksājumu funkcijām un tā satur metodes maksājumu funkciju veikšanai.

- Piekļuve:
    - Lai šo objektu lietotu, ir nepieciešams to konstruēt, izmantojot attiecīgo klases MerchantConnector metodi.
- Metodes:
    - `Payment makePurchase(PurchaseRequest request)` – ļauj lietotājam veikt funkciju "make purchase" (MF.MP.01), parametros padodot PurchaseRequest klases objektu kā funkcijas ievaddatu avotu un pretī atgriežot Payment klases objektu, kas satur funkcijas izvaddatus.
    - `Payment finishPayment(FinishRequest request)` – ļauj lietotājam veikt funkciju "finish payment" (MF.FP.02), parametros padodot FinishRequest klases objektu kā funkcijas ievaddatu avotu un pretī atgriežot Payment klases objektu, kas satur funkcijas izvaddatus.
    - `Payment reversePayment(ReverseRequest request)` – ļauj lietotājam veikt funkciju "reverse payment" (MF.RVP.03), kā parametru padodot ReverseRequest klases objektu kā funkcijas ievaddatu avotu un pretī atgriežot Payment klases objektu, kas satur funkcijas izvaddatus.
    - `Payment refundPayment(RefundRequest request)` – ļauj lietotājam veikt funkciju "refund payment" (MF.RFP.04), parametros padodot RefundRequest klases objektu kā ievaddatu avotu un pretī atgriežot Payment klases objektu, kas satur funkcijas izvaddatus.
    - `Payment getPaymentById(String paymentId)` – ļauj lietotājam veikt funkciju "get payment", parametros padodot funkcijas ievaddatu lauku "ID" un pretī atgriežot Payment klases objektu, kas satur funkcijas izvaddatus.
    - `PaymentList getPaymentList(PaymentListRequest request)` – ļauj lietotājam veikt funkciju "get payment list", parametros padodot PaymentListRequest klases objektu kā funkcijas ievaddatu avotu un pretī atgriežot PaymentList klases objektu, kas satur funkcijas izvaddatus.

BasketItem – šī klase ir nepieciešama priekš "item" datu struktūras lauku glabāšanas, ievadīšanas un izvadīšanas.

- Piekļuve:
  - `BasketItem()` – konstruktors, kurš atgriež jaunu šīs klases objektu.
  - `static BasketItem create()` – statiska metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš "item" datu struktūras lauku ievadīšanas un izvadīšanas, pielietojot *builder pattern* struktūru.

Basket – šī klase ir nepieciešama priekš "basket" datu struktūras lauku glabāšanas, ievadīšanas un izvadīšanas.

- Piekļuve:
  - `Basket()` – klases konstruktors, kurš atgriež jaunu šīs klases objektu.
  - `static Basket create()` – statiska klases metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš "basket" datu struktūras lauku ievadīšanas un izvadīšanas, pielietojot *builder pattern* struktūru. Šīs datu struktūras "items" lauka datus ir nepieciešams ievadīt vai atgriezt, izmantojot BasketItem klases objektus.

PurchaseRequest – šī klase ir nepieciešama priekš funkcijas "make purchase" (MF.MP.01) ievaddatu ievadīšanas, kas tiek veikta ar attiecīgo Transactions klases metodi.

- Piekļuve:
  - `PurchaseRequest()` – klases konstruktors, kurš atgriež jaunas šīs klases objektu.
  - `static PurchaseRequest create()` – statiska klases metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš attiecīgās funkcijas ievaddatu ievadīšanas, pielietojot *builder pattern* struktūru. Attiecīgās funkcijas ievaddatu laukus "basket" un "basket_preauth" ir nepieciešams ievadīt, izmantojot Basket klases objektu.

FinishRequest – šī klase ir nepieciešama priekš funkcijas "finish payment" (MF.FP.02) ievaddatu ievadīšanas, kas tiek veikta ar attiecīgo Transactions klases metodi.

- Piekļuve:
  - `FinishRequest()` – klases konstruktors, kurš atgriež jaunas šīs klases objektu.
  - `static FinishRequest create()` – statiska klases metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš attiecīgās funkcijas ievaddatu ievadīšanas, pielietojot *builder pattern* struktūru. Attiecīgās funkcijas ievaddatu lauku "basket" ir nepieciešams ievadīt, izmantojot Basket klases objektu.

ReverseRequest – šī klase ir nepieciešama priekš funkcijas "reverse payment" (MF.RVP.03) ievaddatu ievadīšanas, kas tiek veikta ar attiecīgo Transactions klases metodi.

- Piekļuve:
  - `ReverseRequest()` – klases konstruktors, kurš atgriež jaunas šīs klases objektu.
  - `static ReverseRequest create()` – statiska klases metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš attiecīgās funkcijas ievaddatu ievadīšanas, pielietojot *builder pattern* struktūru. Attiecīgās funkcijas ievaddatu lauku "reversal_basket" ir nepieciešams ievadīt, izmantojot Basket klases objektu.

RefundRequest – šī klase ir nepieciešama priekš funkcijas "reverse payment" (MF.RFP.04) ievaddatu ievadīšanas, kas tiek veikta ar attiecīgo Transactions klases metodi.

- Piekļuve:
  - `RefundRequest()` – klases konstruktors, kurš atgriež jaunas šīs klases objektu.
  - `static RefundRequest create()` – statiska klases metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš attiecīgās funkcijas ievaddatu ievadīšanas, pielietojot *builder pattern* struktūru. Attiecīgās funkcijas ievaddatu lauku "basket" ir nepieciešams ievadīt, izmantojot Basket klases objektu.

PaymentListRequest – šī klase ir nepieciešama priekš funkcijas "get payment list" (MF.GPL.06) ievaddatu ievadīšanas, kas tiek veikta ar attiecīgo Transactions klases metodi.

- Piekļuve:
  - `RefundRequest()` – klases konstruktors, kurš atgriež jaunas šīs klases objektu.
  - `static RefundRequest create()` – statiska klases metode, kura atgriež jaunu šīs klases objektu.
- Metodes:
  - Metodes priekš attiecīgās funkcijas ievaddatu ievadīšanas, pielietojot *builder pattern* struktūru.

Payment – šī klase ir nepieciešama priekš maksājumu funkciju MF.MP.01, MF.FP.02, MF.RVP.03, MF.RFP.04 un MF.GP.05 izvaddatu izvadīšanas, kas tiek veiktas ar attiecīgajām Transaction klases metodēm.

- Piekļuve:
  - Lietotājam nav paredzēts veidot jaunu šīs klases objektu, tiek tikai sniegts šīs klases objekts, kurš jau satur visus nepieciešamos izvaddatus, veicot attiecīgās funkcijas.
- Metodes:
  - Metodes priekš attiecīgo funkciju izvaddatu izvadīšanas. Attiecīgo funkcju izvaddatu laukus "basket", "basket_preauth" un "reversal_basket" ir nepieciešams izvadīt, izmatojot Basket klases objektus.

PaymentList – šī klase ir nepieciešama priekš maksājumu funkcijas "get payment list" (MF.GPL.06) izvaddatu izvadīšanas, kas tiek veikta ar attiecīgo Transaction klases metodi.

- Piekļuve:
  - Lietotājam nav paredzēts vaidot jaunu šīs klases objektu, tiek tikai sniegts šīs klases objekts, kurš jau satur visus nepieciešamos izvaddatus, veicot attiecīgo funkciju.
- Metodes:
  - Metodes priekš attiecīgās funkcijas izvaddatu izvadīšanas. Attiecīgā izvadītā maksājuma saraksta katru maksājumu ir nepieciešams izvadīt, izmantojot Payment klases objektus.

3.1.2. Komandrindas režīma saskarne

Sekojot attiecīgajām programmatūras prasību specifikācijas dokumenta prasībām, lietotāja saskarnei no komandrindas režīma ir jānodrošina ievaddatu padošanu kā šīs bibliotēkas galaprodukta *.jar* faila startēšanas parametrus, sekojot *UNIX Shell* komandrindas komandu parametru padošanas vadlīnijām. Šajā sadaļā tiek aprakstīts nepieciešamais, no lietotāja sagaidāmais parametru padošanas formāts, priekš maksājumu funkciju izpildīšanas.

Parametru secība un to formāti ir šādi:

1. Konfigurācijas faila atrašanās vieta lietotāja sistēmā.
2. Identifikators maksājumu funkcijai, kuru lietotājs vēlas izpildīt. Lietotājam pieejamie maksājumu funkciju identifikatori ir šādi:
    - "`--purchase`" – identifikators priekš funkcijas "make purchase" (MF.MP.01) izpildes.
    - "`--finish`" – identifikators priekš funkcijas "finish payment" (MF.FP.02) izpildes.
    - "`--reverse`" – identifikators priekš funkcijas "reverse payment" (MF.RVP.03) izpildes.
    - "`--refund`" – identifikators priekš funkcijas "refund payment" (MF.RFP.04) izpildes.
    - "`--get`" – identifikators priekš funkcijas "get payment by ref" (MF.GP.05) izpildes.
    - "`--get_list`" identifikators priekš funkcijas "get payment list" (MF.GPL.06) izpildes.
3. Visi turpmākie parametri tiek uzskatīti kā attiecīgo funkciju ievaddati, kuri tiek padoti ar šādām prasībām:
    - Katrs ievaddatu lauks tiek padots ievadē kā 2 parametru pāris, kuru 1. parametrs identificē padodamo lauku formātā "`--<lauka_nosaukums>`" un sekojošais parametrs ir attiecīgā lauka vērtība.
    - Ja identificējamais lauks ir lauks, kuru ir nepieciešams padot kā *Java boolean* datu tipa vērtību, tad, nenorādot attiecīgā lauka vērtību, tā vērtība tiek uzskatīta par "true".

Tā kā šī saskarne prasa specifisku formātu padošanu, tad ir nepieciešams izvadīt specifiskus kļūdu paziņojumus to neievērošanas gadījumā. Visi šie kļūdu paziņojumi seko tām pašām prasībām, kas aprakstītas programmatūras prasību specifikācijas dokumenta sadaļā 2.1.1. Iespējamie kļūdu paziņojumi un to izpildīšanās nosacījumi ir šādi:

- Ja ievadē padotais parametru skaits ir mazāks par 2, tad tiek izvadīts kļūdu paziņojums ar saturu "Invalid amount of arguments for any action".
- Ja ievadē padotais 2. parametrs ir padots nepareizajā formātā, tad tiek izvadīts kļūdu paziņojums ar saturu "Unexpected action argument".
- Ja kāds no funkcijas ievaddatu parametru lauka identifikatoriem ir ievadē padots nepareizajā formātā, tad tiek izvadīts kļūdu paziņojums ar saturu "unexpected argument: "<kļūdainais_lauka_identifikators>"."
- Ja kāda no funkcijas ievaddatu parametru lauka vērtībām nav tikusi ievadē padota, tad tiek izvadīts kļūdu paziņojums ar saturu "Expected value for argument "<kļūdainā_lauka_identifikators>"", izņemot priekš laukiem, kuru identificējamais lauks ir lauks, kuru ir nepieciešams padot kā *Java boolean* datu tipu.

Piemērs, kā attiecīgā komanda varētu izskatīties, lai veiktu funkciju "make purchase" ar lauku "amount" vērtību "100", "ccy_code" lauka vērtību "978" un "invoice" lauka vērtību "true":

```
java -jar C:/Users/User/Desktop/Configuration.properties
--purchase --amount 100 --ccy_code 978 --invoice
```

3.2. Funkciju projektējums

3.2.1. Sesijas datu pārvaldīšana

Šajā sadaļā tiek sīkāk aprakstītas programmatūras prasību specifikācijas sadaļas 2.1.1 “Sesijas dati un to pārvaldīšana” sesijas datu derīguma pārbaudes. Šīs pārbaudes tiek pielietotas, veicot autorizācijas funkciu “get bearer” (AF.GB.03).

Lai aprēķinātu sesijas lauku “access_token” un “refresh_token” derīgumu, papildus sesijas laukiem “expires_in” un "refresh_expires_in”, ir nepieciešams izmantot papildus vērtības - pašreizējo laiku, sesijas datu saņemšanas laiku, kurš tiek pieglabāts attiecīgajiem sesijas datiem to saņemšanas brīdī, un drošības laiku.

Drošības laiks, ir paredzēts, lai attiecīgie sesijas lauka dati nezaudētu savu derīgumu pēc to derīgumu pārbaudes un pirms to uzrādīšanas bankas sistēmas serveriem. Šī vērtība tiek iegūta lielākajai no autorizācijas vai maksājumu konfigurācijas datu lauka “connection_timeout” vērtībai pieskaitot 1 sekundi. Ja abu konfigurācijas datu lauks “connection_timeout” ir 0, tad šī vērtība tiek pieņemta kā 15 sekundes.

Lai noskaidrotu vai “access_token” lauks vairs nav derīgs un ir nepieciešams veidot jaunu sesiju vai to atjaunot, tiek pielietota šāda formula:

```
(<pašreizējais_laiks>) > (<datu_saņemšanas_laiks> + <expires_in>
- <drošības_laiks>)
```

Ja pašreizējais laiks pārsniedz datu saņemšanas laika un “expires_in” lauka summu, atņemot drošības laiku, tad lauks “access_token” nav derīgs. Pretēji, “access_token” lauka dati ir derīgi izmantošanai.

Lai noskaidrotu vai “refresh_token” lauka vairs nav derīgs un ir nepieciešams veidot jaunu sesiju, tiek pielietota šāda formula:

```
(<pašreizējais_laiks>) > (<datu_saņemšanas_laiks> +
<refresh_expires_in> - <drošības_laiks>)
```

Ja pašreizējais laiks pārsniedz datu saņemšanas laika un “refresh_expires_in” lauka summu, atņemot drošības laiku, tad lauks “refresh_token” nav derīgs. Pretēji, “refresh_token” lauka dati ir derīgi izmantošanai.

3.2.2. Nefunkcionālo prasību nodrošināšana

Šajā sadaļā tiek aprakstīts, kā no izstrādātāja ir paredzēts nodrošināt programmatūras prasību specifikācijas dokumenta sadaļā 2.2. aprakstītās nefunkcionālās prasības. Visi šajā sadaļā minētie klašu nosaukumi tiek iegūti no sadaļā 3.1.1. aprakstītajām pirmkoda lietotāja saskarnes klasēm.

Veiktspējas prasības:

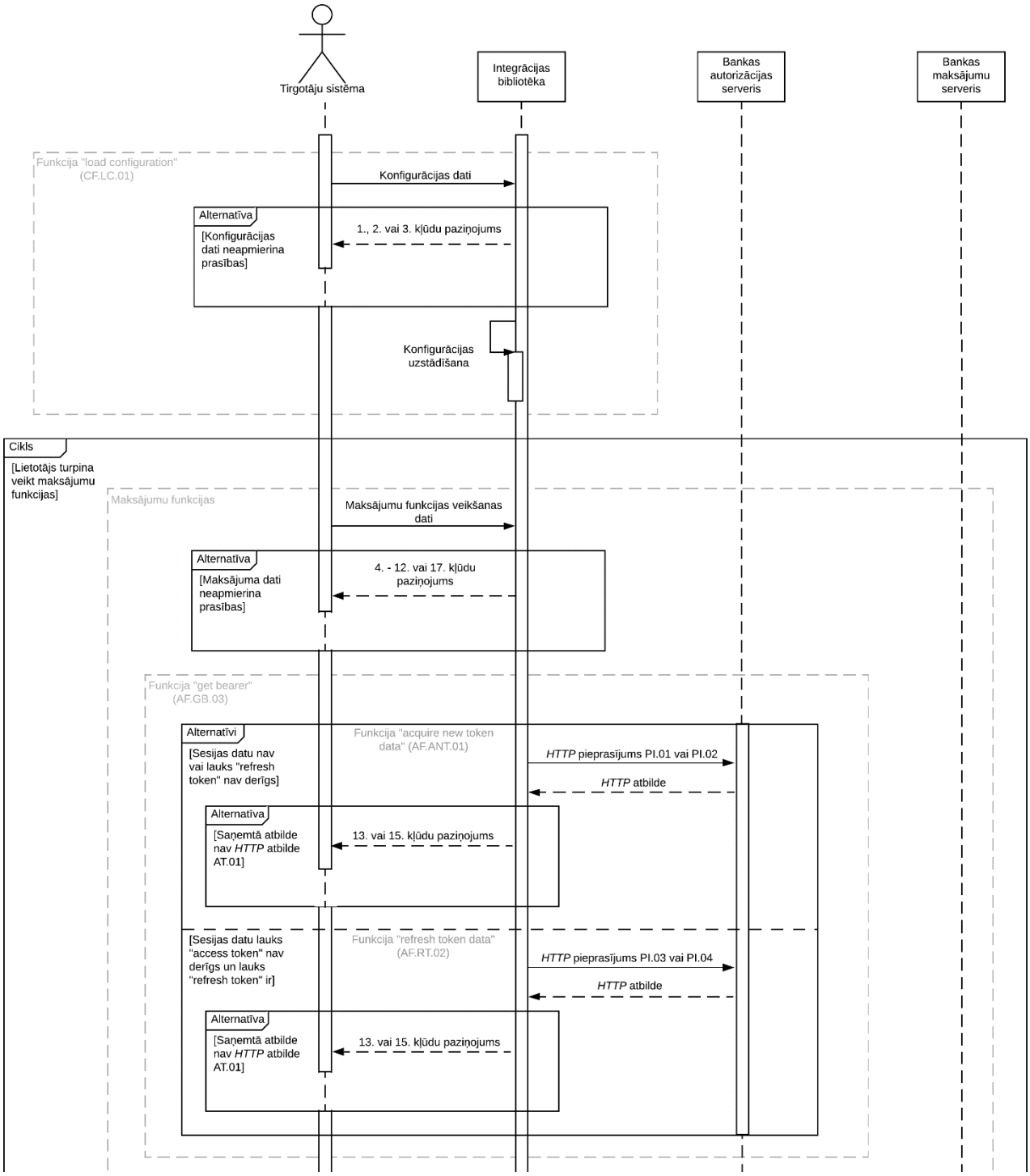
1. Lai nodrošinātu mazāku *heap* operatīvās atmiņas aizņemto daudzumu palielinājumu, vairākiem pavedieniem vienlaicīgi izpildot jebkuru funkciju, tad ir nepieciešams atmiņai dārgākos, pielietotos objektus izmantot atkārtoti priekš katra pavediena, kur iespējams. Šie objekti varētu būt, piemēram, pielietotā *HTTP* savienojumu izveidošanas klases objekts.
2. Mazāku *HTTP* savienojumu veikšanas skaitu ir iespējams sasniegt tikai nodrošinot efektīvu sesijas datu izmantošanu. Efektīva sesijas datu izmantošana tiek nodrošināta, sekojot PPS dokumenta sadaļas 2.1.1. "sesijas dati un to pārvaldīšana" un projektējuma sadaļas 3.2.1. prasībām, kā arī nodrošinot sesijas datu neizdzēšanu no operatīvās atmiņas tirgotāja sistēmas izpildes brīdī, saglabājot tos statiskās *Java* atbalstītās datu struktūrās.
3. Lai nodrošinātu ātrāku maksājumu funkciju izpildi pēc tirgotāju sistēmas palaišanas, tad ir nepieciešams nodrošināt lietotājam iespēju izveidot MerchantConnector klases objektu un ielasīt, uzstādīt konfigurācijas datus sistēmas palaišanas brīdī, attiecīgo objektu un konfigurāciju izmantojot atkārtoti un vienlaicīgi starp katru pavedienu priekš katras maksājumu funkcijas veikšanas.
4. Lai efektīvi nodrošinātu drošību un pareizu funkciju izpildi, izpildoties vairākiem vienlaicīgiem pavedieniem, ir nepieciešams nodrošināt katra pavediena atmiņā ierakstītos datus glabāt *stack* atmiņā. Pavedienu īslaicīga darbības apturēšana var tikt izmantota, darbojoties tikai ar statiskos *Java* atbalstītos datu tipos glabātajiem sesijas datiem.

Citas prasības:

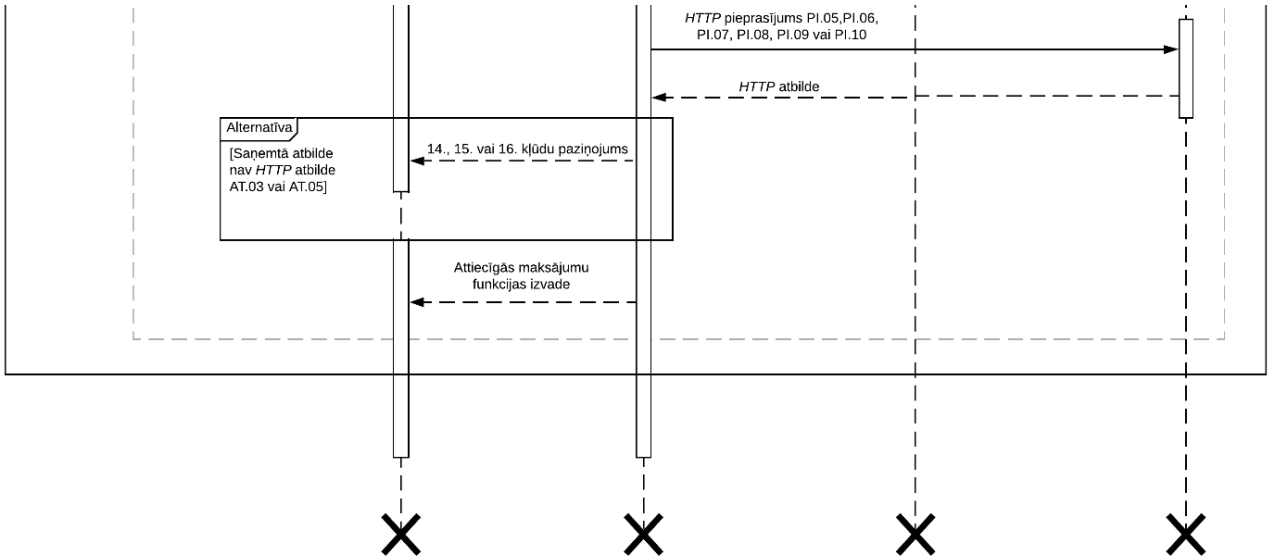
- Pārējās nefunkcionālās prasības tiek nodrošinātas, ievērojot *Java* programmēšanas vadlīnijas un labo praksi.

3.2.3. Kopējās darbības secība

Šajā sadaļā tiek aprakstīta vispārējā darbības secība, tirgotāju sistēmai veicot maksājumu funkciju caur šo produktu un iesaistot visus četrus darbībās iesaistītos aģentus – tirgotāju sistēmu, šo produktu jeb integrācijas bibliotēku, bankas autorizācijas serveri un bankas maksājumu serveri.



3.1 att. – **Kopējās darbības secību diagrammas 1. daļa**



3.2 att. – **Kopējās darbības secību diagrammas 2. daļa**

Kā redzams att. 3.1. un 3.2., darbību sāk tirgotāju sistēma, sniedzot integrācijas bibliotēkai konfigurācijas datus jeb veic bibliotēkas funkciju "load configuration" (CF.LC.01), kuras rezultātā, ja attiecīgie dati neatbilst funkcijas prasībām, tiek izvadīts 1., 2. vai 3. kļūdu paziņojums, un bibliotēkas darbība tiek apturēta, vai pretēji, tiek veikta tās apstrāde, uzstādot attiecīgo konfigurāciju.

Pēc funkcijas CF.LC.01 veikšanas, diagrammā sekojošais cikls identificē sadaļas 3.3.2 veiktspējas prasību nodrošināšanas prasību, ka maksājumu funkcijas ir nepieciešams spēt izpildīt atkārtoti ar vienu un to pašu uzstādīto konfigurāciju.

Veicot jebkuru maksājumu funkciju, tirgotāju sistēmai integrācijas bibliotēkai ir nepieciešams sniegt attiecīgās funkcijas ievaddatus, kuru rezultātā, ja attiecīgie dati neatbilst attiecīgās funkcijas ievaddatu prasībām, tiek izvadīts 4. – 12. vai 17. kļūdu paziņojums, un bibliotēkas darbība tiek apturēta. Maksājumu funkciju ievaddatiem ir nepieciešamas arī autorizācijas funkcijas "get bearer" (AF.GB.03) izvaddati, tāpēc pēc tirgotāju sistēmas saņemto ievaddatu pārbaudes, tiek veikta tās darbība.

Veicot funkciju AF.GB.03 atkarībā no operatīvajā atmiņā glabāto sesijas datu stāvokļa, uz bankas autorizācijas serveri tiek veikts *HTTP* pieprasījums PI.01, PI.02, PI.03 vai PI.04, pretī saņemot *HTTP* atbildi. Ja saņemtā *HTTP* atbilde neatbilst atbildei AT.01, tad atkarībā no saņemtās atbildes tiek izvadīts 13. vai 15. kļūdu paziņojums attiecīgi funkcijas AF.GB.03 prasībām, un bibliotēkas darbība tiek apturēta.

Kad attiecīgās maksājumu funkcijas ievaddati ir tikuši saņemti un pārbaudīti, tad ar tiem uz bankas maksājumu serveri tiek veikts pieprasījums PI.05, PI.06, PI.07, PI.08, PI.09 vai PI.10 atkarībā no veiktās maksājumu funkcijas, pretī saņemot *HTTP* atbildi. Ja saņemtā *HTTP* atbilde neatbilst atbildei AT.03 vai AT.05, tad atkarībā no saņemtās atbildes tiek izvadīts 14. 15. vai 16. kļūdu paziņojums attiecīgi veiktās maksājumu funkcijas prasībām. Pretēji, tiek veikta attiecīgās maksājumu funkcijas izvade atpakaļ tirgotāju sistēmai un maksājumu funkcijas darbība ir veiksmīgi beigusi darbu. Tirgotāju sistēmai šajā solī ir iespēja atkārtot jebkuru maksājumu funkciju, neveicot funkciju “load configuration” (CF.LC.01).

4. PROGRAMMATŪRAS TESTĒŠANA

Šīs programmatūras testēšana tika veikta, rakstot testpiemērus pirmkodā, izmantojot testēšanai specifiskas *Java* bibliotēkas jeb izstrādes rīkus un nodrošinot to darbināšanu pēc katras izmaiņas veikšanas. Visi testpiemēri ir tikuši rakstīti un attīstījušies līdzi ar pirmkoda funkcionalitātes izstrādi.

Visi testpiemēri tika izstrādāti, izmantojot šādas *Java* bibliotēkas:

- *JUnit Jupiter 5.1.0* – šī bibliotēka nodrošina testpiemēru rakstīšanu un palaišanu struktūru un funkcijas. Visi testi šajā struktūrā tiek rakstīti kā testu klašu metodes.
- *Wiremock 2.19.0* – šī bibliotēka nodrošina pagaidu servera uzstādīšanu un konfigurēšanu priekš testpiemēriem, kuri pārbauda pareizu *HTTP* savienojumu izveidi un atbilžu apstrādi.
- *JSONAssert 1.5.0* – šī bibliotēka nodrošina ērtu un efektīvu *JSON object* datu struktūru pārbaudi un salīdzināšanu.

4.1. Testpiemēru projektējums

4.1.1. Funkcionālo prasību testpiemēri

Visi testpiemēri pirmkodā tiek veidoti kā atsevišķas klases metodes, kurām atbilstoši *JUnit* bibliotēkas prasībām, tiek pievienota anotācija "*@Test*". Visas testpiemēru metodes tiek sadalītas klasēs, kur katras testpiemēru klases metodes ir priekš kādas izstrādē izveidotās klases vienas nozīmīgas darbības pārbaudes.

Kopā tika izveidoti 106 testpiemēri, kuri tika sadalīti 7 klasēs. Šie testpiemēri iekļauj arī specifiskus testpiemērus, piemēram, vai tiek izvadīts kļūdu paziņojums, ja konfigurācijas teksta fails nav atrodams vai ielasāms, bet uzskatāmības pēc šajā sadaļā tiks uzskaitīti un apkopoti tikai tie testpiemēri, kuri ir tieši saistīti ar programmatūras prasību specifikācijā un programmatūras projektējuma aprakstā aprakstītajām prasībām.

ConfigurationTest – šīs klases testpiemēri ir priekš izstrādātās Configuration klases funkcionalitāšu testēšanas (skat. tabulu 4.1). Attiecīgā klase nodrošina funkcijas CF.LC.01 ievaddatu saņemšanu, to prasību pārbaudīšanu, saglabāšanu operatīvajā atmiņā.

4.1. tabula – **ConfigurationTest klases testpiemēri**

Identifikators	Ievaddati un uzstādījumi	Sagaidāmais rezultāts
CT.01	Ievadē tiek padota konfigurācijas teksta faila atrašanās vieta, kurš satur visus konfigurācijas datus ar pareizi noformētām vērtībām.	Nekāds kļūdu paziņojums netiek atgriezts. Iespēja no Configuration klases saņemt katru konfigurācijas lauku datus.
CT.02	Ievadē tiek padots *Map* objekts, kas satur visus konfigurācijas datus ar pareizi noformētām vērtībām.	Nekāds kļūdu paziņojums netiek atgriezts. Iespēja no Configuration klases saņemt katra konfigurācijas lauka datus.
CT.03	Ievadē tiek padoti *Map* objekti, kuri satur visus, izņemot viena, obligāta konfigurācijas lauka datus.	*InvalidConfigurationException* klases objekts, kura paziņojums sakrīt ar 1. kļūdu paziņojumu.
CT.04	Ievadē tiek padoti *Map* objekti, kuri satur tikai visus nepieciešamos konfigurācijas datus, ja lauka “auth_mode” vērtība ir “password”.	Nekāds kļūdu paziņojums netiek atgriezts. Ir iespēja no Configuration klases saņemt lauku vērtības, kuriem pastāv noklusējuma vērtības.
CT.05	Ievadē tiek padoti *Map* objekti, kuri satur tikai visus nepieciešamos konfigurācijas datus, ja lauka “auth_mode” vērtība ir “certificate”.	Nekāds kļūdu paziņojums netiek atgriezts. Ir iespēja no Configuration klases saņemt lauku vērtības, kuriem pastāv noklusējuma vērtības.
CT.06	Ievadē tiek padoti *Map* objekti, kuri satur derīgus konfigurācijas datus, izņemot vienu, kurš ir dots vienā no to nepareizajiem formātiem.	*InvalidConfigurationException* klases objekts, kura paziņojums sakrīt ar 2. kļūdu paziņojumu vai 3. kļūdu paziņojumu, ja attiecīgais nepareizi formatētais lauks ir “auth_mode”.

MerchantConnectorTest - šīs klases testpiemēri ir priekš izstrādātās MerchantConnector klases funkcionalitāšu testēšanas (skat. tabulu 4.2). Attiecīgā klase nodrošina funkcijas CF.LC.01 apstrādi priekš autorizācijas un maksājumu funkciju konfigurācijas ievaddatiem.

4.2. tabula – **MerchantConnectorTest klases testpiemēri**

Identifikators	Ievaddati un uzstādījumi	Sagaidāmais rezultāts
MCT.01	Ievadē tiek padoti konfigurācijas dati, kuri satur priekš sistēmā pastāvošu *Keystore* un *Truststore* failiem reālus konfigurācijas datus. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem autorizācijas konfigurācijas datos norādīto *Keystore* faila saturošos sertifikātus un uzrāda *Truststore* faila saturošos sertifikātus. Tiek uzstādīts pagaidu maksājumu serveris, kurš pieņem maksājumu konfigurācijas datos norādīto *Keystore* faila saturošos sertifikātus un uzrāda *Truststore* faila saturošos sertifikātus.	Veicot jebkuru maksājumu funkciju, nekāds kļūdu paziņojums netiek atgriezts.
MCT.02	Ievadē tiek padoti derīgi konfigurācijas dati ar autorizācijas konfigurācijas lauka "response_timeout" vertību "1". Tiek uzstādīts pagaidu autorizācijas serveris, kurš *HTTP* pieprasījumiem atbildi sniedz pēc 20000 milisekundēm.	Veicot jebkuru maksājumu funkciju, tiek atgriezts *SocketTimeoutException* klases objekts.
MCT.03	Ievadē tiek padoti konfigurācijas dati ar maksājumu konfigurācijas lauka "response_timeout" vertību "1". Tiek uzstādīts pagaidu autorizācijas serveris, kurš atbild ar derīgu *HTTP* atbildi AT.01. Tiek uzstādīts pagaidu maksājumu serveris, kurš *HTTP* pieprasījumiem atbildi sniedz pēc 20000 milisekundēm.	Veicot jebkuru maksājumu funkciju, tiek atgriezts *SocketTimeoutException* klases objekts.
MCT.04	Ievadē tiek padoti derīgi konfigurācijas dati ar autorizācijas konfigurācijas lauka "connection_timeout" vertību "1". Tiek uzstādīts pagaidu autorizācijas serveris.	Veicot jebkuru maksājumu funkciju, tiek atgriezts *ConnectTimeoutException* klases objekts.

MCT.05	Ievadē tiek padoti konfigurācijas dati ar maksājumu konfigurācijas lauka "connection_timeout" vertību "1". Tiek uzstādīts pagaidu autorizācijas serveris, kurš atbild ar derīgu *HTTP* atbildi AT.01. Tiek uzstādīts pagaidu maksājumu serveris.	Veicot jebkuru maksājumu funkciju, tiek atgriezts *ConnectTimeoutException* klases objekts.

TokenHandlerTest – šīs klases testpiemēri ir priekš izstrādātās TokenHandler klases funkcionalitāšu testēšanas (skat. tabulu 4.3). Attiecīgā klase nodrošina autorizācijas funkciju "get bearer" (AF.GB.03), "create new session" (AF.CNS.01) un "refresh session" (AF.RS.02) izpildi.

4.3. tabula – **TokenHandlerTest klases testpiemēri**

Identifikators	Ievaddati un uzstādījumi	Sagaidāmais rezultāts
TKHT.01	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="password" • "username"="test1" Operatīvajā atmiņā, izmantojot identifikatoru "test1", tiek saglabāti sesijas dati ar šādām lauku vērtībām: • "access_token"="test1_access" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="0" • "token_acquiring_time"= pašreizējais laiks.	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, lai iegūtu glabātos sesijas datus. • Atpazīts, ka sesijas dati ir derīgi • Atgriezta simbolu virkne "Bearer test1_access".

TKHT.02	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="certificate" • "keystore_path"="test" • "key_alias"="2" Operatīvajā atmiņā, izmantojot identifikatoru "test2", tiek saglabāti sesijas dati ar šādām lauku vērtībām: • "access_token"="test2_access" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="0" • "token_acquiring_time"= pašreizējais laiks.	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, lai iegūtu glabātos sesijas datus. • Atpazīts, ka sesijas dati ir derīgi. • Atgriezta simbolu virkne "Bearer test2_access".
TKHT.03	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="password" • "username"="test3" Operatīvajā atmiņā neglabājās nekādi sesijas dati. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem *HTTP* pieprasījumu PI.01 un pretī atgriež atbildi AT.01 ar šādiem sesijas datiem: • "access_token"="test3_access" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="60000"	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, un atpazīts, ka sesijas dati netiek glabāti un ir jāveic funkcija "create new session" (AF.CNS.01). • Pareizi veic *HTTP* pieprasījumu PI.01. • Pareizi saņem un apstrādā *HTTP* atbildi AT.01. • Atgriež simbolu virkni "Bearer test3_access". • Operatīvajā atmiņā pareizi saglabā attiecīgos datus, izmantojot identifikatoru "test3".

TKHT.04	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="certificate" • "keystore_path"="test" • "key_alias"="4" Operatīvajā atmiņā neglabājās nekādi sesijas dati. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem *HTTP* pieprasījumu PI.02 un pretī atgriež atbildi AT.01 ar šādiem sesijas datiem: • "access_token"="test4_access" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="60000"	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, un atpazīts, ka sesijas dati netiek glabāti un ir jāveic funkcija "create new session" (AF.CNS.01). • Pareizi veic *HTTP* pieprasījumu PI.02. • Pareizi saņem un apstrādā *HTTP* atbildi AT.01. • Atgriež simbolu virkni "Bearer test4_access". • Operatīvajā atmiņā pareizi saglabā attiecīgos datus, izmantojot identifikatoru "test4".
TKHT.05	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="password" • "username"="test5" Operatīvajā atmiņā, izmantojot identifikatoru "test5", tiek saglabāti sesijas dati ar šādām lauku vērtībām: • "access_token"="test" • "refresh_token"="test" • "expires_in"="0" • "refresh_expires_in"="0" • "token_acquiring_time"= pašreizējais laiks. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem *HTTP* pieprasījumu PI.01 un pretī atgriež atbildi AT.01 ar šādiem sesijas datiem: • "access_token"="test" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="60000"	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, lai iegūtu glabātos sesijas datus. • Atpazīts, ka glabāto sesijas datu lauki "access_token" un "refresh_token" vairs nav derīgi un ir jāveic funkcija "create new session" (AF.CNS.01). • Pareizi veic *HTTP* pieprasījumu PI.01. • Pareizi saņem un apstrādā *HTTP* atbildi AT.01. • Atgriež simbolu virkni "Bearer test5_access". • Operatīvajā atmiņā pareizi saglabā attiecīgos datus, izmantojot identifikatoru "test5".

TKHT.06	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="certificate" • "keystore_path"="test" • "key_alias"="6" Operatīvajā atmiņā, izmantojot identifikatoru "test6", tiek saglabāti sesijas dati ar šādām lauku vērtībām: • "access_token"="test" • "refresh_token"="test" • "expires_in"="0" • "refresh_expires_in"="0" • "token_acquiring_time"= pašreizējais laiks. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem *HTTP* pieprasījumu PI.02 un pretī atgriež atbildi AT.01 ar šādiem sesijas datiem: • "access_token"="test" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="60000"	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, lai iegūtu glabātos sesijas datus. • Atpazīts, ka glabāto sesijas datu lauki "access_token" un "refresh_token" vairs nav derīgi un ir jāveic funkcija "create new session" (AF.CNS.01). • Pareizi veic *HTTP* pieprasījumu PI.02. • Pareizi saņem un apstrādā *HTTP* atbildi AT.01. • Atgriež simbolu virkni "Bearer test6_access". • Operatīvajā atmiņā pareizi saglabā attiecīgos datus, izmantojot identifikatoru "test6".
TKHT.07	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "auth_mode"="certificate" • "username"="test7" Operatīvajā atmiņā, izmantojot identifikatoru "test7", tiek saglabāti sesijas dati ar šādām lauku vērtībām: • "access_token"="test" • "refresh_token"="test" • "expires_in"="0" • "refresh_expires_in"="60000" • "token_acquiring_time"= pašreizējais laiks. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem *HTTP* pieprasījumu PI.03 un pretī atgriež atbildi AT.01 ar šādiem sesijas datiem: • "access_token"="test" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="60000"	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, lai iegūtu glabātos sesijas datus. • Atpazīts, ka glabāto sesijas datu lauks "access_token" vairs nav derīgs, bet lauks "refresh_token" ir un ir jāveic funkcija "refresh session" (AF.RS.02). • Pareizi veic *HTTP* pieprasījumu PI.03. • Pareizi saņem un apstrādā *HTTP* atbildi AT.01. • Atgriež simbolu virkni "Bearer test7_access". • Operatīvajā atmiņā pareizi saglabā attiecīgos datus, izmantojot identifikatoru "test7".

TKHT.08	Ievadē tiek padoti nepieciešamie autorizācijas konfigurācijas dati ar šādām lauku vērtībām: • "keystore_path"="test" • "key_alias"="8" Operatīvajā atmiņā, izmantojot identifikatoru "test8", tiek saglabāti sesijas dati ar šādām lauku vērtībām: • "access_token"="test" • "refresh_token"="test" • "expires_in"="0" • "refresh_expires_in"="60000" • "token_acquiring_time"= pašreizējais laiks. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pieņem *HTTP* pieprasījumu PI.04 un pretī atgriež atbildi AT.01 ar šādiem sesijas datiem: • "access_token"="test" • "refresh_token"="test" • "expires_in"="60000" • "refresh_expires_in"="60000"	Veicot funkciju "get bearer" (AF.GB.01), tiek sagaidīts, ka: • Tiek veiksmīgi izmantots pareizais identifikators, lai iegūtu glabātos sesijas datus. • Atpazīts, ka glabāto sesijas datu lauks "access_token" vairs nav derīgs, bet lauks "refresh_token" ir un ir jāveic funkcija "refresh session" (AF.RS.02). • Pareizi veic *HTTP* pieprasījumu PI.04. • Pareizi saņem un apstrādā *HTTP* atbildi AT.01. • Atgriež simbolu virkni "Bearer test8_access". • Operatīvajā atmiņā pareizi saglabā attiecīgos datus, izmantojot identifikatoru "test7".
TKHT.09	Ievadē tiek padoti nepieciešamie konfigurācijas dati. Operētājsistēmā netiek glabāti nekādi sesijas dati. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pretī atgriež atbildi, kuras *body* dati ir tukši un statusa kods ir "400".	Veicot funkciju "create new session" (AF.CNS.01) vai "refresh session" (AF.RS.02), tiek sagaidīts, ka: • Tiek izvadīts *AuthEndpointException* objekts, kurš atbilst 15. kļūdu paziņojumam.
TKHT.10	Ievadē tiek padoti nepieciešamie konfigurācijas dati. Operētājsistēmā netiek glabāti nekādi sesijas dati. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pretī atgriež atbildi, kuras *body* dati satur simbolu virkni "Test error string" un statusa kods ir "400".	Veicot funkciju "create new session" (AF.CNS.01) vai "refresh session" (AF.RS.02), tiek sagaidīts, ka: • Tiek izvadīts *AuthEndpointException* objekts, kurš atbilst 15. kļūdu paziņojumam.

TKHT.11	Ievadē tiek padoti nepieciešamie konfigurācijas dati. Operētājsistēmā netiek glabāti nekādi sesijas dati. Tiek uzstādīts pagaidu autorizācijas serveris, kurš pretī atgriež atbildi AT.02, kuras lauks "error" satur simbolu virkni "test error", lauks "error_description" satur simbolu virkni "test error description" un statusa kods ir "400".	Veicot funkciju "create new session" (AF.CNS.01) vai "refresh session" (AF.RS.02), tiek sagaidīts, ka: • Tiek izvadīts *AuthEndpointException* objekts, kurš atbilst 13. kļūdu paziņojumam.

TransactionModelAndSerializerTest – šīs klases testpiemēri ir priekš izstrādāto maksājumu funkciju ievaddatu un izvaddatu konstruēšanas klasēm (skat. tabulu 4.4). Šīs klases nodrošina lietotājam maksājumu funkciju ievaddatu ievadīšanu, to formātu pārbaudīšanu, izvaddatu saņemšanu un datu formātu maiņu.

4.4. tabula – **TokenHandlerTest klases testpiemēri**

Identifikators	Ievaddati un uzstādījumi	Sagaidāmais rezultāts
TMST.01	Tiek izveidots BasketItem klases objekts, kuram tiek ievadīti visi iespējamie basket item datu struktūras lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās BasketItem klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts sakrīt basket item datu struktūras *JSON* formātu.
TMST.02	Tiek veidoti BasketItem klases objekti, kuriem tiek padotas basket item datu struktūras lauku vērtības, kuras ir vienā no to nepareizajiem formātiem.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 10. kļūdu paziņojumu.
TMST.03	Tiek izveidots Basket klases objekts, kuram tiek ievadīti visi iespējamie basket datu struktūras lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās Basket klases objekta metodi, kas to atgriež kā *JSON object*, rezultāts sakrīt basket datu struktūras *JSON* formātu.

TMST.04	Tiek izveidots PurchaseRequest klases objekts, kuram tiek ievadīti visi iespējamie funkcijas “make purchase” (MF.MP.01) lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās PurchaseRequest klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts sakrīt ar *HTTP* pieprasījuma PI.05 *body* datu *JSON* formātu.
TMST.05	Tiek veidoti PurchaseRequest klases objekti, kuriem tiek padotas funkcijas “make purchase” (MF.MP.01) lauku vērtības nepareizajos formātos.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 5. kļūdu paziņojumu, kuru “maksājuma tipa” vērtība ir “Purchase”, vai 17. kļūdu paziņojumu, ja lauka “expiry” ievadītais datums ir senāks par pašreizējo datumu.
TMST.06	Tiek izveidots FinishRequest klases objekts, kuram tiek ievadīti visi iespējamie funkcijas “finish payment” (MF.FP.02) lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās FinishRequest klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts sakrīt ar *HTTP* pieprasījuma PI.06 *body* datu *JSON* formātu.
TMST.07	Tiek veidoti FinishRequest klases objekti, kuriem tiek padotas funkcijas “finish payment” (MF.FP.02) lauku vērtības nepareizajos formātos.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 5. kļūdu paziņojumu, kuru “maksājuma tipa” vērtība ir “Finish”.
TMST.08	Tiek izveidots ReverseRequest klases objekts, kuram tiek ievadīti visi iespējamie funkcijas “reverse payment” (MF.RVP.03) lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās ReverseRequest klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts sakrīt ar *HTTP* pieprasījuma PI.07 *body* datu *JSON* formātu.
TMST.09	Tiek veidoti ReverseRequest klases objekti, kuriem tiek padotas funkcijas “reverse payment” (MF.RVP.03) lauku vērtības nepareizajos formātos.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 5. kļūdu paziņojumu, kuru “maksājuma tipa” vērtība ir “Reverse”.
TMST.10	Tiek izveidots RefundRequest klases objekts, kuram tiek ievadīti visi iespējamie funkcijas “refund payment” (MF.RFP.04) lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās RefundRequest klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts sakrīt ar *HTTP* pieprasījuma PI.08 *body* datu *JSON* formātu.
TMST.11	Tiek veidoti RefundRequest klases objekti, kuriem tiek padotas funkcijas “refund payment” (MF.RFP.04) lauku vērtības nepareizajos formātos.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 5. kļūdu paziņojumu, kuru “maksājuma tipa” vērtība ir “Refund”.

TMST.12	Tiek izveidots PaymentListRequest klases objekts, kuram tiek ievadīti visi iespējamie funkcijas "get payment list" (MF.GLP.06) lauki pareizos formātos.	Nekāds kļūdu paziņojums netiek atgriezts. Veicot attiecīgās PaymentListRequest klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts satur *HTTP* pieprasījuma PI.10 *query* datus.
TMST.13	Tiek veidoti PaymentListRequest klases objekti, kuriem tiek padotas funkcijas "get payment list" (MF.GPL.06) lauku vērtības nepareizajos formātos.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 5. kļūdu paziņojumu, kuru "maksājuma tipa" vērtība ir "Payment list retrieval".
TMST.14	Tiek izveidots *JSON object*, kurš sakrīt ar *HTTP* atilbes AT.03 *body* datu *JSON object* ar identifikatoru "data" datu struktūru.	Attiecīgo *JSON object* ir iespējams pārveidot par Payment klases objektu. Veicot Payment klases objekta metodi, kas to atgriež *JSON* formātā, rezultāts sakrīt ar sākotnējo *JSON object*.

TransactionRequestValidatorTest – šīs klases testpiemēri ir priekš izstrādātās TokenRequestValidator klases funkcionalitāšu testēšanas (skat. tabulu 4.5). Attiecīgā klase nodrošina maksājumu funkciju ievaddatu nepieciešamības prasības.

4.5. tabula – **TransactionRequestValidatorTest klases testpiemēri**

Identifikators	Ievaddati un uzstādījumi	Sagaidāmais rezultāts
TRVT.01	Ievadē tiek padoti funkcijas "make purchase" (MF.MP.01) ievaddati, kuri satur visus, izņemot viena, obligāta lauka datus.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 4., 6., 7., 9., 11. vai 12. kļūdu paziņojumu, kura "maksājuma tipa" vērtība ir "Purchase",attiecīgi kurš ievaddatu lauks nav ticis padots.
TRVT.02	Ievadē tiek padoti funkcijas "make purchase" (MF.MP.01) ievaddati, kuri satur visus obligātos lauka datus, pie visiem nosacījumiem.	Nekāds kļūdu paziņojums netiek atgriezts.
TRVT.03	Ievadē tiek padoti funkcijas "finish payment" (MF.FP.02) ievaddati, kuri satur visus, izņemot viena, obligāta lauka datus.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 4., 9., 11. vai 12. kļūdu paziņojumu, kura "maksājuma tipa" vērtība ir "Finish", attiecīgi kurš ievaddatu lauks nav ticis padots.

TRVT.04	Ievadē tiek padoti funkcijas “finish payment” (MF.FP.02) ievaddati, kuri satur visus obligātos lauka datus, pie visiem nosacījumiem.	Nekāds kļūdu paziņojums netiek atgriezts.
TRVT.05	Ievadē tiek padoti funkcijas “reverse payment” (MF.RVP.03) ievaddati, kuri satur visus, izņemot viena, obligāta lauka datus.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 4., 7., 9., 11. vai 12. kļūdu paziņojumu, kura “maksājuma tipa” vērtība ir “Reverse”, attiecīgi kurš ievaddatu lauks nav ticis padots
TRVT.06	Ievadē tiek padoti funkcijas “reverse payment” (MF.RVP.03) ievaddati, kuri satur visus obligātos lauka datus, pie visiem nosacījumiem.	Nekāds kļūdu paziņojums netiek atgriezts.
TRVT.07	Ievadē tiek padoti funkcijas “refund payment” (MF.RFP.04) ievaddati, kuri satur visus, izņemot viena, obligāta lauka datus.	*TransactionDataFormatException* klases objekts, kura paziņojums sakrīt ar 4., 9., 11. vai 12. kļūdu paziņojumu, kura “maksājuma tipa” vērtība ir “Refund”.
TRVT.08	Ievadē tiek padoti funkcijas “refund payment” (MF.RFP.04) ievaddati, kuri satur visus obligātos lauka datus, pie visiem nosacījumiem.	Nekāds kļūdu paziņojums netiek atgriezts.

TransactionHandlerTest - šīs klases testpiemēri ir priekš izstrādātās TransactionHandler klases funkcionalitāšu testēšanas (skat. tabulu 4.6). Attiecīgā klase nodrošina maksājumu funkciju apstrādes veikšanu.

4.6. tabula – **TransactionHandlerTest klases testpiemēri**

Identifikators	Ievaddati un uzstādījumi	Sagaidāmais rezultāts
TRHT.01	Ievadē tiek padoti nepieciešamie konfigurācijas dati. Tiek uzstādīts pagaidu maksājumu serveris, kurš saņem *HTTP* pieprasījumu PI.05-PI.10 un atbild at *HTTP* atbildi AT.03.	Veicot maksājumu funkcijas tiek sagaidīts ka: • Tiek veiksmīgi veikts attiecīgais *HTTP* pieprasījums. • Tiek veiksmīgi saņemta un apstrādāta attiecīgā *HTTP* atbilde.
TRHT.02	Ievadē tiek padoti nepieciešamie konfigurācijas dati. Tiek uzstādīts pagaidu maksājumu serveris, kurš saņem *HTTP* pieprasījumu PI.05-PI.10 un atbild at *HTTP* atbildi, kas nesatur *body* datus un, kura statusa kods ir ar vērtību "400".	Veicot maksājumu funkcijas tiek sagaidīts ka: • Tiek izvadīts *TransactionEndpointException* objekts, kurš atbilst 15. kļūdu paziņojumam.
TRHT.03	Tiek uzstādīts pagaidu maksājumu serveris, kurš saņem *HTTP* pieprasījumu PI.05-PI.10 un atbild at *HTTP* atbildi, kura *body* dati dati satur simbolu virkni "Test error string" un, kura statusa kods ir ar vērtību "400".	Veicot maksājumu funkcijas tiek sagaidīts ka: • Tiek izvadīts *TransactionEndpointException* objekts, kurš atbilst 15. kļūdu paziņojumam.
TRHT.04	Tiek uzstādīts pagaidu maksājumu serveris, kurš saņem *HTTP* pieprasījumu PI.05-PI.10 un atbild *HTTP* atbildi AT.04, kuras lauks "title" satur simbolu virkni "test error", lauks "description" satur simbolu virkni "test error description" un statusa kods ir "400".	Veicot maksājumu funkcijas tiek sagaidīts ka: • Tiek izvadīts *TransactionEndpointException* objekts, kurš atbilst 14. kļūdu paziņojumam.

TerminalTest - šīs klases testpiemēri ir priekš izstrādātās Main klases funkcionalitāšu testēšanas (skat. tabulu 4.7). Attiecīgā klase nodrošina funkciju veikšanu no komandrindas režīma. Tiek tikai veiktas dažas papildus ievaddatu formātu pārbaudes, jo tā ir vienīgā funkciju veikšanas daļa, kur abi režīmi neizmanto to pašu pirmkoda daļu.

4.7. tabula – **TerminalTest klases testpiemēri**

Identifikators	**Ievaddati un uzstādījumi**	**Sagaidāmais rezultāts**
TT.01	Tiek veiktas maksājumu funkcijas, ievaddatos izmantojot kļūdainus komandrindas režīma paredzētos formātus.	*MerchantConnectorException* vai *TransactionDataFormatException* klases objekti, kas ir specifiski komandrindas režīma saskarnei.

4.1.2. Nefunkcionālo prasību testēšana

Veiktspējas prasību testēšana

1. Lai pārbaudītu, ka sistēmas palielinātais aizņemtās *heap* operatīvās atmiņas daudzums nepārsniedz 50 MB, izpildoties 10 vienlaicīgiem pavedieniem, izmantojot rīku "*Apache benchmarking tool*" (https://httpd.apache.org/docs/2.4/programs/ab.html) tiek veikti 10 vienlaicīgu pavedienu pieprasījumi uz sistēmu, kas bibliotēku ir ieviesusi, un ar "*NetBeans" Java* programmēšanas vides *"Profiler"* (https://profiler.netbeans.org/) rīku tiek izvērtēta sistēmas maksājumu funkciju izpildes *heap* operatīvās atmiņas palielinātais aizņemtais daudzums. Lai nodrošinātu šo vērtību kā maksimālo iespējamo, tiek veikta katra maksājumu funkcija ar vairākām to datu kombinācijām un tiek izvērtēta lielākā vērtība.
2. Lai pārbaudītu efektīvu sesijas datu izmantošanu un to neizdzēšanu no operatīvās atmiņas, izmantojot rīku "*Apache benchmarking tool*" tiek veikti 10 vienlaicīgi pavedienu pieprasījumi uz serveri, kura šo bibliotēku iekļauj un ar konsoles izdrukām tiek pārliecināts, ka visi pavedieni, izņemot pirmo, izmanto jau iepriekš iegūtus, derīgus sesijas datus.
3. Lai pārbaudītu, ka jebkuras maksājumu funkcijas izpildes laiks ir ne ilgāks par 1 sekundi, neskaitot *HTTP* savienojuma izveides un atbildes saņemšanas laikus, tiek veiktas pašreizējā laika konsoles izdrukas maksājumu funkcijas veikšanas sākumā un beigās un jebkura *HTTP* pieprasījuma un atbildes saņemšanas procesa sākumā un beigās. Atņemot nepieciešamos laikus, tiek izrēķināta attiecīgā vērtība. Lai nodrošinātu šo vērtību kā maksimālo iespējamo, tiek veikta katra maksājumu funkcija ar vairākām to datu kombinācijām un tiek izvērtēta ilgākā vērtība.

4.2 Testēšanas žurnāls

4.2.1 Žurnāls funkcionālo prasību testpiemēriem

Šajā sadaļā visi sadaļā 4.1.1 aprakstītie testpiemēri, kuri attiecas uz funkcionālo prasību testēšanu, to ieviešanas datumi un veiksmīgas vai neveiksmīgas izpildes stāvokļi tiek apkopoti (skat. tabulu 4.8). Priekš informācijas, kā attiecīgo testpiemēru rezultāti pēc to izpildes tiek paziņoti, skat. 1. un 2. pielikumu.

4.8. tabula – **Nefunkcionālo prasību testpiemēru žurnāls**

Datumi / Testpiemērs	20.03.19	26.03.19	12.04.19	15.04.19	07.05.19	08.05.19
CT.01	+	+	+	+	+	+
CT.02			+	+	+	+
CT.03			+	+	+	+
CT.04			+	+	+	+
CT.05			+	+	+	+
CT.06			+	+	+	+
MCT.01			+	+	+	+
MCT.02					-	+
MCT.03					-	+
MCT.04						+
MCT.05						+
TKHT.01	+	+	+	+	+	+
TKHT.02	+	+	+	+	+	+
TKHT.03	+	+	+	+	+	+
TKHT.04	+	+	+	+	+	+
TKHT.05	+	+	+	+	+	+
TKHT.06	+	+	+	+	+	+
TKHT.07	+	+	+	+	+	+
TKHT.08	+	+	+	+	+	+
TKHT.09	-	+	+	+	+	+

TKHT.10	-	+	+	+	+	+
TKHT.11	-	+	+	+	+	+
TMST.01			-	+	+	+
TMST.02			-	+	+	+
TMST.03			-	+	+	+
TMST.04	+	+	-	+	+	+
TMST.05	+	+	-	+	+	+
TMST.06	+	+	-	+	+	+
TMST.07	+	+	-	+	+	+
TMST.08	+	+	-	+	+	+
TMST.09	+	+	-	+	+	+
TMST.10				+	+	+
TMST.11				+	+	+
TMST.12		-	+	+	+	+
TMST.13		-	+	+	+	+
TMST.14	+	+	-	+	+	+
TRVT.01	+	+	-	+	+	+
TRVT.02	+	+	-	+	+	+
TRVT.03	+	+	-	+	+	+
TRVT.04	+	+	-	+	+	+
TRVT.05	+	+	-	+	+	+
TRVT.06	+	+	-	+	+	+
TRVT.07				+	+	+
TRVT.08				+	+	+
TRHT.01 MF.MP.01 MF.FP.02 MF.RVP.03 MF.GP.05	-	+	-	+	+	+
TRHT.02 MF.MP.01 MF.FP.02 MF.RVP.03 MF.GP.05	-	+	+	+	+	+

TRHT.03 MF.MP.01 MF.FP.02 MF.RVP.03 MF.GP.05	-	+	+	+	+	+
TRHT.04 MF.MP.01 MF.FP.02 MF.RVP.03 MF.GP.05	-	+	+	+	+	+
TRHT.01 MF.GPL.06		-	+	+	+	+
TRHT.02 MF.GPL.06		+	+	+	+	+
TRHT.03 MF.GPL.06		+	+	+	+	+
TRHT.04 MF.GPL.06		+	+	+	+	+
TRHT.01 MF.RFP.04				+	+	+
TRHT.02 MF.RFP.04				+	+	+
TRHT.03 MF.RFP.04				+	+	+
TRHT.04 MF.RFP.04				+	+	+
TT.01					+	+

4.2.2 Žurnāls nefunkcionālo prasību testpiemēriem

Šajā sadaļā visi testpiemēri, kuri attiecas uz nefunkcionālo prasību testēšanu (skat. sadaļu 4.1.2) un to rezultāti tiek apkopoti.

Veiktspējas prasību testpiemēru žurnāls

Visi veiktspējas testpiemēri tika veikti datumā 08.05.19, jo tad tika uzskatīts, ka programmatūra ir pietiekami pabeigta, lai veiktspējas parametri vairs nemainītos.

1. Pēc parauga tirgotāju sistēmas palaišanas un 10 vienlaicīgu pieprasījumu veikšanas (kā redzams 3. pielikumā), tika iegūts attiecīgā pieprasījuma veikšanas aizņemtās *heap* atmiņas daudzuma palielinājums. Izmantojot palielinājuma sākumu un beigu datus, kuri redzami 4. un 5. pielikumā, maksimālais iegūtais aizņemtās atmiņas daudzuma pieaugums bija aptuveni 34.44 MB, kas apmierina prasības.
2. Pēc parauga tirgotāju sistēmas palaišanas un 10 vienlaicīgu pieprasījumu veikšanas, tika sistēmas konsolē izdrukāti paziņojumi, kuri norāda uz katra pieprasījuma izmantotajiem sesijas datiem. Kā redzams 6. pielikumā, tikai pirmajam pavedienam bija nepieciešams iegūt jaunus sesijas datus un pārejie 9 tos izmantoja, neveicot atsevišķus jaunu sesiju datu pieprasījumus, apmierinot prasības.
3. Pēc parauga tirgotāju sistēmas palaišanas un maksājumu funkcijas pieprasījuma veikšanas, tika sistēmas konsolē izdrukāti paziņojumi, kuri, kā redzams 7. pielikumā, norāda uz maksājumu funkcijas sākumu un beigu laiku, kā arī uz jebkuru *HTTP* pieprasījumu un atbildes saņemšanas sākumu un beigām. Izmantojot šos datus, maksimālais maksājumu funkcijas veikšanas izpildes laiks bija 618 milisekundes, kas apmierina prasības.

5. PROJEKTA ORGANIZĀCIJA

Uzņēmums, kurā šī programmatūra tika izstrādāta, parasti pielieto *Agile* programmatūras izstrādes dzīves cikla *Scrum* paveidu, bet tā kā visas galvenās prasības bija zināmas un izvērtētas kā ar zemu iespēju mainīties, un šo programmatūru izstrādāja viens cilvēks, tad tika pielietots iteratīvā un inkrementālā dzīves cikla modeļu apvienojums, kur, izmantojot katras iepriekšējās versijas izstrādē iegūtās zināšanas, katrā iterācijā tika uzlabots programmatūras projektējums un pievienotas jaunas funkcionalitātes. Tā kā attiecīgās programmatūras funkcionalitāšu kopums saistībā ar maksājumu funkcijām ir cieši saistīts ar maksājumu servera attīstību, tad, pielietojot šo modeli, tiek nodrošināta viegla maksājumu funkciju pievienošana, kad tas ir nepieciešams.

Tā kā attiecīgais produkts ticis izstrādāts produkta pasūtītāja uzņēmuma telpās, tad prasību neskaidrību gadījumā uzreiz bija pieejamas konsultācijas.

5.1 Darbietilpības novērtējums

Lai novērtētu šīs programmatūras izstrādes darbietilpību, kopā ar vecākajiem izstrādātājiem tā tika sadalīta daļās, tika veikti to svarīgākie pieņēmumi un katras daļas darbietilpība novērtēta dienās, izmantojot trīs punktu novērtēšanu, balstoties uz pieredzi, iepriekš veidotām un līdzīgām komponentēm, profesionālo intuīciju. Trīs punktu novērtēšanā tika izmantota formula S = (A + 4B + C)/6, kur A = optimistiskais, B = reālistiksais un C = pesimistiskais novērtējums priekš dienu skaitu katrai izstrādes daļai. Viena diena tiek pieņemta kā pilna darba diena jeb 8h.

Kā redzams tabulā 5.1, pēc izstrādes procesa sadalījuma, pieņēmumu un darbietilpības novērtēšanas veikšanas, tika aprēķināts, ka šīs programmatūras izstrāde aizņemtu aptuveni 108 dienas.

5.1. tabula – **Darbietilpības novērtējums**

Programmatūras daļa	Pieņēmumi	Novērtējums (dienas)			Rezultāts (dienas)
		Optimistiksias	Reālistiskais	Pesimistiskais	
Apmācība un sākotnējā plānošana	• Sākotnējā pirmkoda projektējuma plānošana. • Nepieciešamo bibliotēku un to izmantošanas pētīšana. • *HTTP* savienojumos nepieciešamo iesaistīto parametru pētīšana. • Konfigurācijas faila formātu pētīšana.	15	18	25	19
Konfigurācija	• Datu ievade/izvade. • Datu pārbaude. • Datu uzstādīšana.	5	8	10	8
Autorizācijas funkcijas	• Datu ievade/izvade. • *HTTP* pieprasījumu veikšana un atbilžu apstrāde. • Sesijas datu pārvaldīšana. • Pavedienu drošības nodrošināšana.	10	13	15	13

Maksājumu funkcijas	• Datu ievade/izvade. • Ievaddatu pārbaude. • *HTTP* pieprasījumu/atbilžu apstrāde.	20	25	30	25
Komandrindas režīma saskarne	• Datu ievade/izvade.	3	5	7	5
Vienībtesti	• Attīstās visas programmatūras izstrādes laikā un ir nepieciešamas priekš katras programmatūras funkcijas.	10	20	25	19
Izmaiņas	• Potenciālā lietoto bibliotēku maiņa. • Potenciālās izmaiņas projektējumā. • Prasību maiņu risks – zems.	10	20	25	19

5.2 Kvalitātes nodrošināšana

Izstrādājamās programmatūras kvalitātes nodrošināšanai tika pielietotas šādas metodes:

- Programmatūras izstrāde, izmantojot *Java* programmēšanas vadlīnijas un OOP pieeju, kas nosaka klašu, metožu un mainīgo nosaukumu prasības, un loģiski vienotu pirmkodu apvienošanu klasēs un metodēs.
- Programmatūras pirmkods tika komentēts, īsi aprakstot klases darbības un tās daļas, kuras izstrādātājiem varētu radīt kādus jautājumus vai, kuras programmatūras uzturētājiem varētu sākumā šķist kā kļūdas, lai novērstu nevēlamas pirmkoda izmaiņas.
- Vienībtestu veidošana un uzturēšana pirmkoda izstrādes laikā.
- Apjomīgāko ārējo izmantoto bibliotēku klases dokumentācijas un citi to lietošanas avoti tika izpētīti, lai nodrošinātu to pareizu un efektīvu izmantošanu.
- Programmatūras izstrādē tika izmantotas izstrādes vide "*NetBeans*" priekš ātras un ērtas pirmkoda rakstīšanas un noformēšanas, kļūdaina pirmkoda un slikta izstrādes stila laicīgas novēršanas.
- Lai ērti pārvaldītu ārējās bibliotēkas no kā programmatūra ir atkarīga, tika izmantots rīks "*Maven*" (https://maven.apache.org/).
- Programmatūras izstrāde tika veikta pasūtītāja uzņēmuma telpās un klātbūtnē, nodrošinot ātru un efektīvu neskaidrību novēršanu.

5.3 Konfigurāciju pārvaldība

Konfigurācijas pārvaldībai tika izmantota versiju kontroles tehnoloģija *Git* ar rīku *BitBucket*. Attiecīgā versiju kontroles tehnoloģija tika izmantota, lai saglabātu programmatūras izstrādes iterācijas un spētu sekot līdzi katrai veiktajai izmaiņai un to veikšanas laikiem.

Tā kā šo pirmkodu izstrādāja viens cilvēks, netika veidoti nekādi versijas atzarojumi, un jaunās izmaiņas tika saglabātas tikai pēc izstrādātāja nepieciešamībām.

IZMANTOTĀ LITERATŪRA

[1] Programmatūras prasību specifikācijas ceļvedis, LVS 68:1996, 1996.

[2] Ieteicamā prakse programmatūras projektējuma aprakstīšanai, LVS 72:1996, 1996.

[3] *Codes for the representation of currencies*, ISO 4217:2015, 2015.

[4] Json.org, *Introducing JSON*. [tiešsaiste] Pieejams: https://www.json.org/ [atsauce 05.05.2019].

[5] G. Aviani, *HTTP and everything you need to know about it*, 7. Oct. 2018. [tiešsaiste] Pieejams: https://medium.com/faun/http-and-everything-you-need-to-know-about-it-8273bc224491 [atsauce 28.04.2019]

[6] A. Raj, *Everything about HTTPS and SSL (Java)*, 17 Jan. 2019. [tiešsaiste] Pieejams: https://www.dzone.com/articles/ssl-in-java [atsauce 03.05.2019].

[7] *Oracle, The Java Tutorials: Primitive Data Types.* [tiešsaiste] Pieejams: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html [atsauce 07.05.2019].

[8] *Oracle, The Java Tutorials: Classes and Objects.* [tiešsaiste] Pieejams: https://docs.oracle.com/javase/tutorial/java/javaOO/index.html [atsauce 07.05.2019].

[9] *Oracle, The Java Tutorials: Exceptions* [tiešsaiste] Pieejams: https://docs.oracle.com/javase/tutorial/essential/exceptions/ [atsauce 07.05.2019].

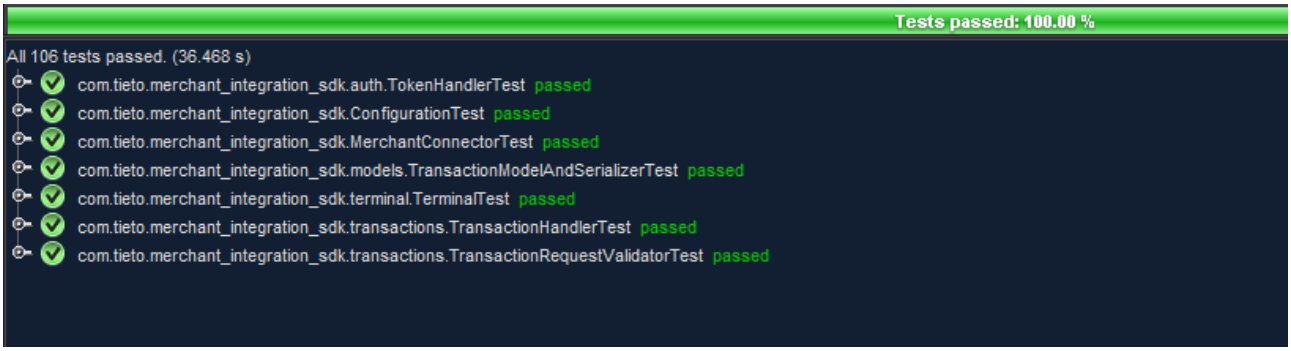
[10] C. Marian, *Java Memory Management*, 7 Jan. 2018. [tiešsaiste] Pieejams: https://dzone.com/articles/java-memory-management [atsauce 07.05.2019].

[11] R. Nel, *Design Patterns: The BuilderPattern*, 13 Dec. 2016. [tiešsaiste] Pieejams: https://dzone.com/articles/design-patterns-the-builder-pattern [atsauce 07.05.2019].

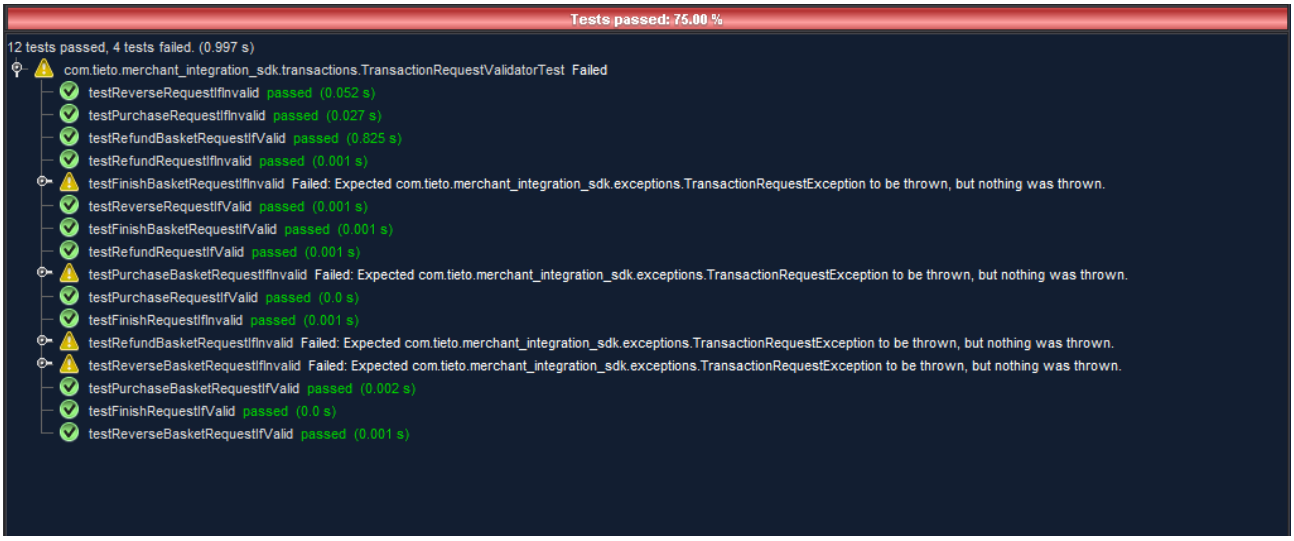
[12] A.Ugarte, *What is Thread-Safety and How to Achieve it?*, 7 Maijs 2019. [tiešsaiste] Pieejams: https://www.baeldung.com/java-thread-safety [atsauce 07.05.2019].

PIELIKUMI

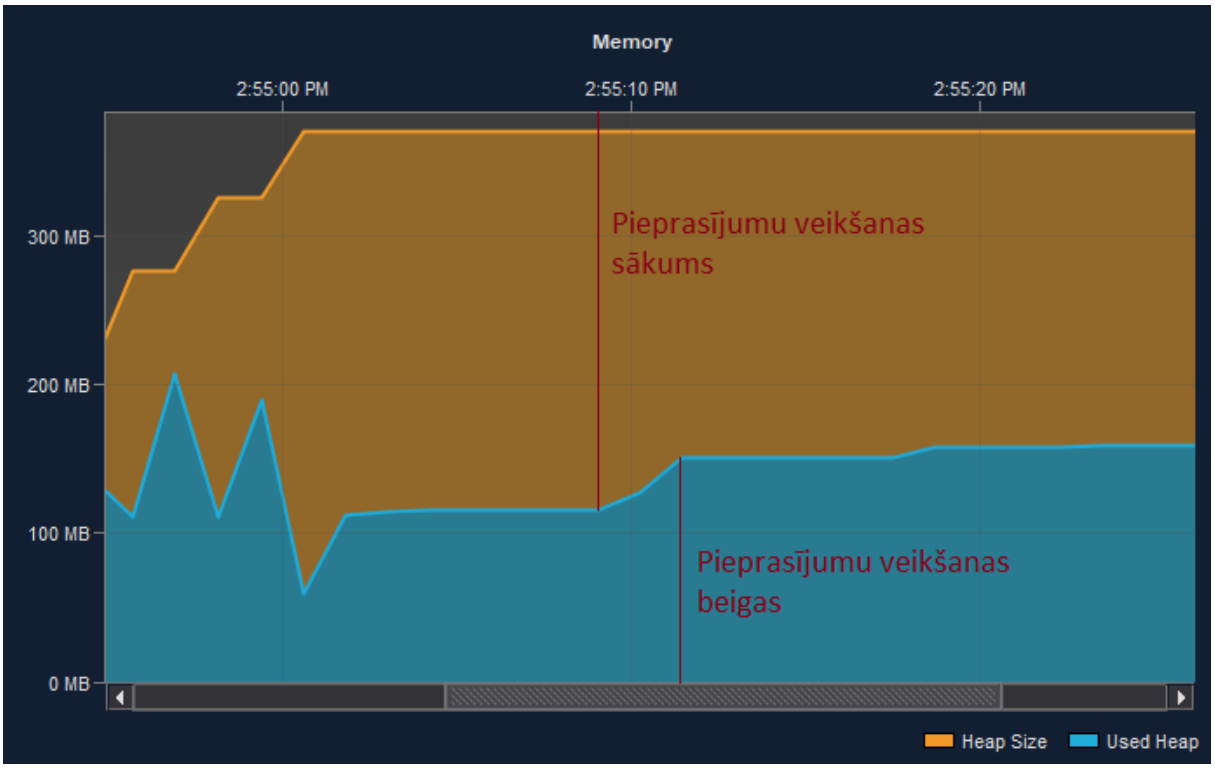
1. Pielikums – veiksmīgu visu 106 testpiemēru izpildes paziņojums.



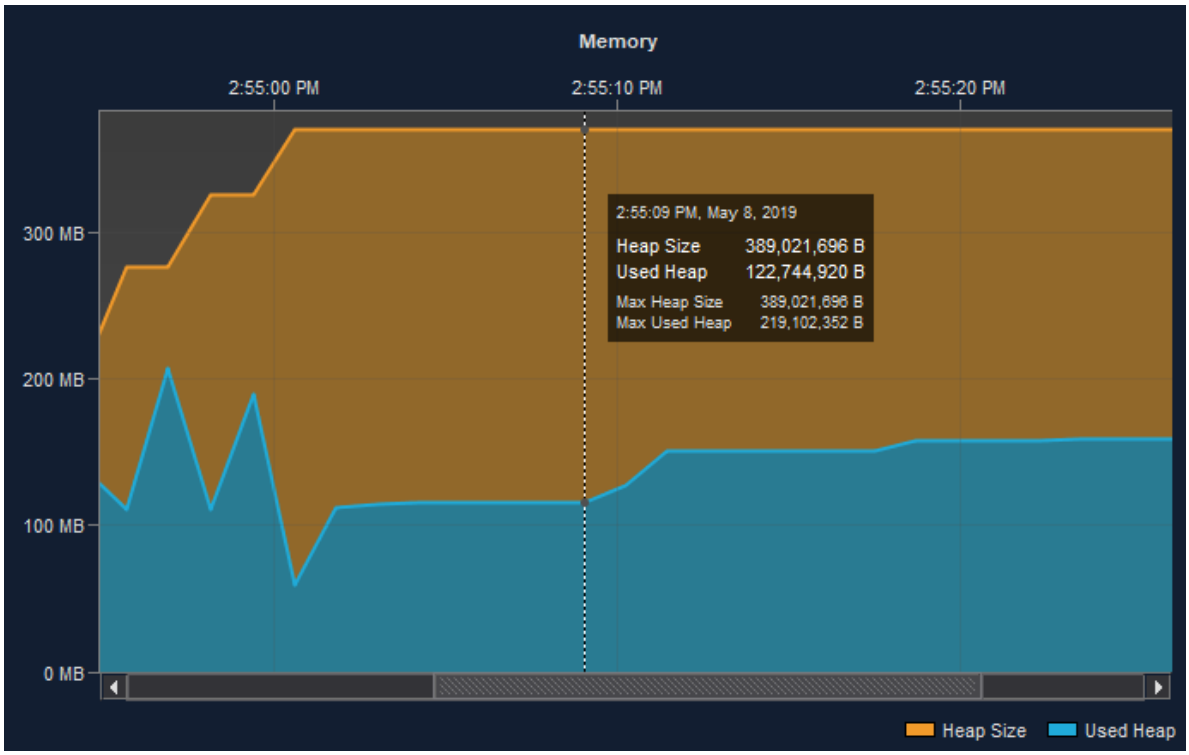
2. Pielikums – neveiksmīgu testpiemēru izpildes paziņojums.



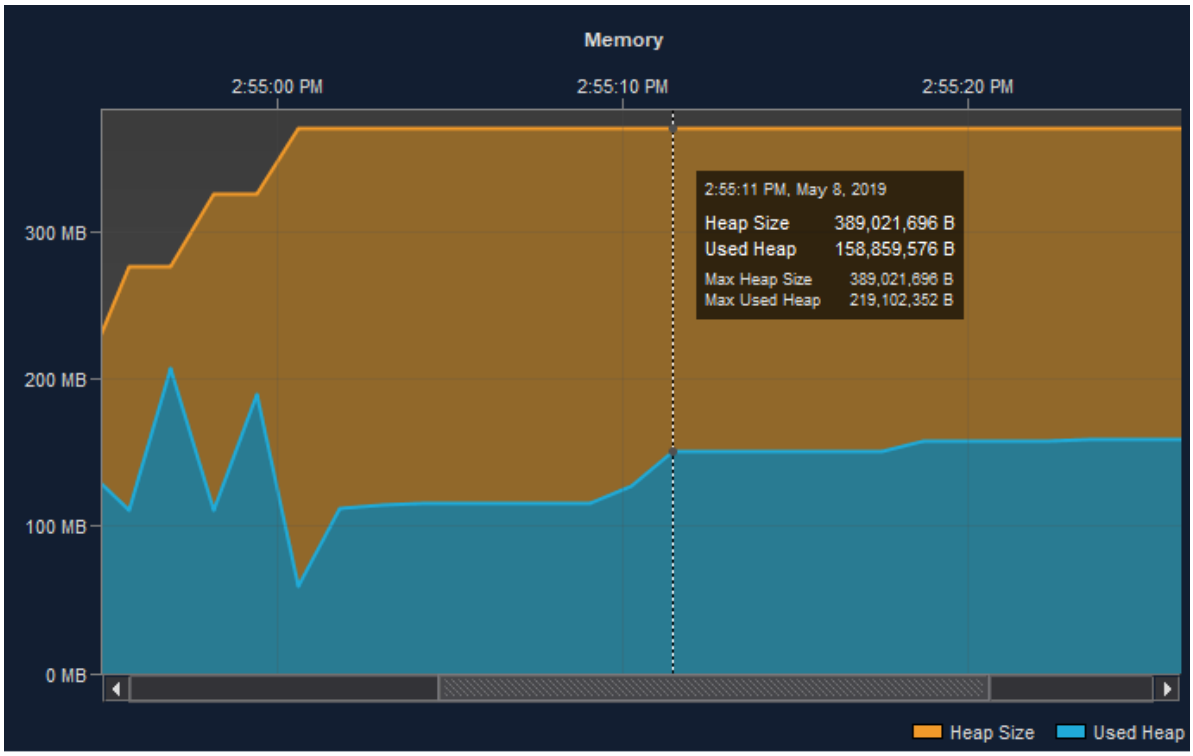
3. Pielikums – parauga tirgotāja sistēmas atmiņas grafiks.



4. Pielikums – parauga tirgotāja atmiņas grafiks ar pirmā pieprasījuma sākuma datiem.



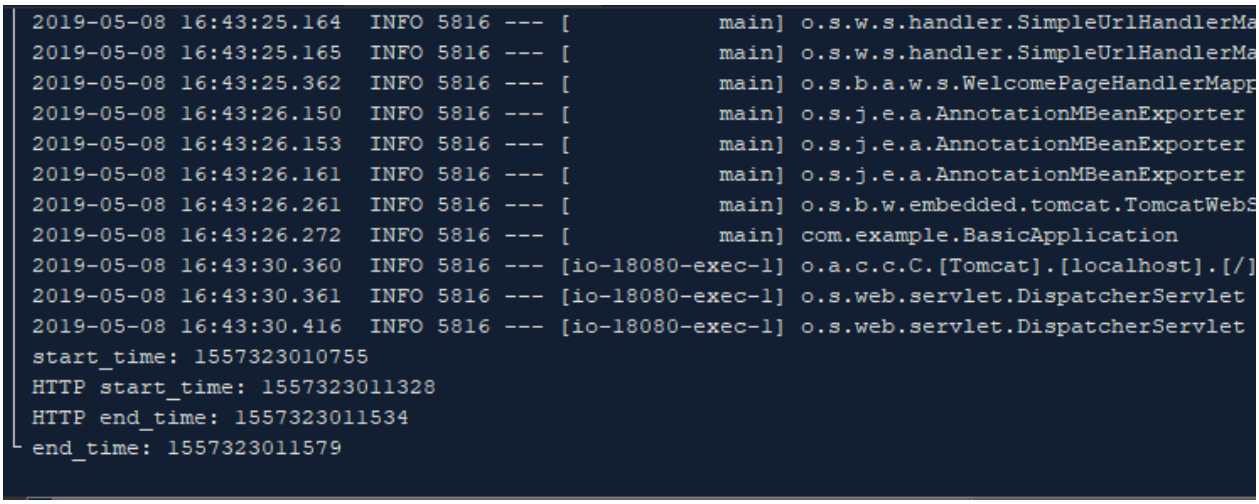
5. Pielikums – parauga tirgotāja atmiņas grafiks ar pirmā pieprasījuma beigu datiem.



6. Pielikums – parauga tirgotāju sesijas datu izmantošana pie vairākiem vienlaicīgiem procesiem.

```
2019-05-08 16:06:14.584  INFO 10700 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/err
2019-05-08 16:06:14.586  INFO 10700 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/err
2019-05-08 16:06:14.662  INFO 10700 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL pat
2019-05-08 16:06:14.663  INFO 10700 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL pat
2019-05-08 16:06:14.905  INFO 10700 --- [           main] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding welcome
2019-05-08 16:06:15.528  INFO 10700 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering be
2019-05-08 16:06:15.531  INFO 10700 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Bean with name
2019-05-08 16:06:15.546  INFO 10700 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Located MBean
2019-05-08 16:06:15.614  INFO 10700 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started
2019-05-08 16:06:15.621  INFO 10700 --- [           main] com.example.BasicApplication             : Started BasicA
2019-05-08 16:06:21.826  INFO 10700 --- [io-18080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing S
2019-05-08 16:06:21.826  INFO 10700 --- [io-18080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServl
2019-05-08 16:06:21.867  INFO 10700 --- [io-18080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServl
Getting new token data
Using existing token
Using existing token
Using existing token
Using existing token
Using existing token
Using existing token
Using existing token
Using existing token
Using existing token
```

7. Pielikums – parauga tirgotāja sistēmas maksājuma veikšanas funkcijas izpildes laiki.



8. Pielikums – TokenHandler klases 1. koda fragments.

```
01. public class TokenHandler {
02. 
03.     private static final String TOKEN_ACQ_TIME_FIELD_NAME = "token_acquiring_time";
04. 
05.     private int tokenExpirySafetyTime;
06.     private Configuration conf;
07.     private Token token;
08.     private String tokenId;
09.     private String endpointURL;
10.     private HttpClient client;
11.     private List<String> requestFields = new ArrayList<String>();
12.     private ObjectReader reader = MerchantConnector.mapper.readerFor(Token.class);
13. 
14.     public TokenHandler(Configuration conf, HttpClient client) throws IOException, AuthEndpointException {
15.         this.conf = conf;
16.         this.client = client;
17. 
18.         requestFields.add(CLIENT_ID_FIELD);
19.         requestFields.add(CLIENT_SECRET_FIELD);
20.         endpointURL = conf.getConfData(AUTH_ENDPOINT_URL_FIELD);
21. 
22.         setSafetyTime();
23. 
24.         if("password".equals(conf.getConfData(AUTH_MODE_FIELD))) {
25.             requestFields.add(USERNAME_FIELD);
26.             requestFields.add(PASSWORD_FIELD);
27.             tokenId = conf.getConfData(USERNAME_FIELD);
28.         }
29.         else {
30.             tokenId = conf.getConfData(AUTH_KEYSTORE_PATH_FIELD) + conf.getConfData(AUTH_KEY_ALIAS_FIELD);
31.         }
32.     }
33. 
34.     private void setSafetyTime() {
35.         int authConnectionTimeout = Integer.parseInt(conf.getConfData(AUTH_CONNECTION_TIMEOUT_FIELD));
36.         int trnConnectionTimeout = Integer.parseInt(conf.getConfData(TRN_CONNECTION_TIMEOUT_FIELD));
37. 
38.         if(authConnectionTimeout == 0 && trnConnectionTimeout == 0) {
39.             tokenExpirySafetyTime = 15;
40.         }
41.         else if(authConnectionTimeout > trnConnectionTimeout) {
42.             tokenExpirySafetyTime = authConnectionTimeout;
43.         }
44.         else {
45.             tokenExpirySafetyTime = trnConnectionTimeout;
46.         }
47.     }
48. 
49.     //The method other remote classes are expected to use to retrieve a valid bearer.
50.     public String getBearer() throws IOException, AuthEndpointException, InterruptedException {
51.         retreiveTokenData();
52.         return "Bearer " + token.getAccessToken();
53.     }
```

9. Pielikums – TokenHandler klases 2. koda fragments.

```
01. private void retreiveTokenData() throws IOException, AuthEndpointException, InterruptedException {
02.     token = TokenRepository.getToken(tokenId);
03.     if(token == null) {
04.         //The first thread to see that token is null locks every other thread until finished creating
05.         //new session.
06.         synchronized(TokenHandler.class) {
07.             token = TokenRepository.getToken(tokenId);
08.             //Checking token for null again so stopped threads see new token data or try again if
09.             //previous thread failed.
10.             if(token == null) {
11.                 createNewSession();
12.             }
13.         }
14.     }
15.
16.     checkAndSetTokenData();
17. }
18.
19. private void checkAndSetTokenData() throws IOException, AuthEndpointException, InterruptedException {
20.     if(!token.accessActive(tokenExpirySafetyTime)) {
21.         //The first thread to see that access token is expired locks every other thread until finishes
22.         //getting new token data.
23.         synchronized(TokenHandler.class) {
24.             token = TokenRepository.getToken(tokenId);
25.             //Checking access token validity again so stopped threads see new token data or try again
26.             //if previous thread failed.
27.             if(!token.accessActive(tokenExpirySafetyTime)) {
28.                 if(token.refreshActive(tokenExpirySafetyTime)) {
29.                     refreshSession();
30.                 }
31.                 else {
32.                     createNewSession();
33.                 }
34.             }
35.         }
36.     }
37. }
38.
39. private void createNewSession() throws IOException, AuthEndpointException {
40.     HttpPost request = new HttpPost(endpointURL);
41.     request.addHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8");
42.     request.addHeader("Accept", "application/json");
43.
44.     String tokenRequestBody = getTokenRequestBody();
45.     StringEntity requestEntity = new StringEntity(tokenRequestBody, ContentType.APPLICATION_FORM_URLENCODED);
46.     request.setEntity(requestEntity);
47.
48.     HttpResponse response = client.execute(request);
49.     int statusCode = response.getStatusLine().getStatusCode();
50.     HttpEntity entity = response.getEntity();
51.
52.     if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
53.         JSONObject inPayloadJson = readResponseJson(entity);
54.         inPayloadJson.put(TOKEN_ACQ_TIME_FIELD_NAME, System.currentTimeMillis());
55.
56.         token = reader.readValue(inPayloadJson.toString());
57.         TokenRepository.setToken(tokenId, token);
58.     }
59.     else {
60.         handleErrorResponse(entity, request, statusCode);
61.     }
62. }
```

10. Pielikums – TokenHandler klases 3. koda fragments.

```
01.     private void refreshSession() throws IOException, AuthEndpointException {
02.         HttpPost request = new HttpPost(endpointURL);
03.         request.addHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8");
04.         request.addHeader("Accept", "application/json");
05.
06.         String refreshRequestBody = getRefreshRequestBody();
07.         StringEntity requestEntity = new StringEntity(refreshRequestBody, ContentType.APPLICATION_FORM_URLENCODED);
08.         request.setEntity(requestEntity);
09.
10.         HttpResponse response = client.execute(request);
11.         int statusCode = response.getStatusLine().getStatusCode();
12.         HttpEntity entity = response.getEntity();
13.
14.         if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
15.             JSONObject inPayloadJson = readResponseJson(entity);
16.             inPayloadJson.put(TOKEN_ACQ_TIME_FIELD_NAME, System.currentTimeMillis());
17.
18.             token = reader.readValue(inPayloadJson.toString());
19.             TokenRepository.setToken(tokenId, token);
20.         }
21.         else {
22.             handleErrorResponse(entity, request, statusCode);
23.         }
24.     }
25.
26.     private String getTokenRequestBody() {
27.         String reqBody = "grant_type=password";
28.         for(String requestField : requestFields) {
29.             reqBody += "&" + requestField + "=" + conf.getConfData(requestField);
30.         }
31.         return reqBody;
32.     }
33.
34.     private String getRefreshRequestBody() {
35.         String reqBody = "grant_type=refresh_token&refresh_token=" + token.getRefreshToken();
36.         for(String requestField : requestFields) {
37.             reqBody += "&" + requestField + "=" + conf.getConfData(requestField);
38.         }
39.         return reqBody;
40.     }
41.
42.     private void handleErrorResponse(HttpEntity entity, HttpRequestBase request, int statusCode)
43.     throws AuthEndpointException, IOException {
44.         String statusCodeStr = String.valueOf(statusCode);
45.         if(entity == null) {
46.             request.abort();
47.             throw new AuthEndpointException(statusCodeStr);
48.         }
49.
50.         String inPayload = readResponseString(entity);
51.         System.out.println(inPayload);
52.         try {
53.             JSONObject inPayloadJson = new JSONObject(inPayload);
54.             throw new AuthEndpointException(inPayloadJson, statusCodeStr);
55.         }
56.         catch(JSONException e) {
57.             throw new AuthEndpointException(statusCodeStr);
58.         }
59.     }
```

11. Pielikums – TransactionHandler klases 1. koda fragments.

```
01. public class TransactionHandler {
02. 
03.     private HttpClient client;
04.     private String endpointURL;
05.     private ObjectReader reader = MerchantConnector.mapper.reader();
06. 
07.     public TransactionHandler(Configuration conf, HttpClient client) {
08.         this.client = client;
09.         endpointURL = conf.getConfData(Configuration.TRN_ENDPOINT_URL_FIELD);
10.     }
11. 
12.     public Payment makePurchase(String bearer, JSONObject purchaseData)
13.     throws IOException, TransactionEndpointException {
14.         HttpPost request = new HttpPost(endpointURL);
15.         request.setHeader("Content-Type", "application/json");
16.         request.setHeader("Authorization", bearer);
17.         request.setHeader("Accept", "application/json");
18. 
19.         StringEntity requestEntity = new StringEntity(purchaseData.toString());
20.         request.setEntity(requestEntity);
21. 
22.         HttpResponse response = client.execute(request);
23.         int statusCode = response.getStatusLine().getStatusCode();
24.         HttpEntity entity = response.getEntity();
25. 
26.         if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
27.             if(statusCode == 302) {
28.                 request.abort();
29.                 throw new TransactionEndpointException(
30.                             "Could not verify access token,possible auth server malfunction",
31.                             "302");
32.             }
33.             JSONObject inPayloadJson = readResponseJson(entity);
34.             reader = reader.forType(Payment.class);
35.             Payment payment = reader.readValue(inPayloadJson.getJSONObject("data").toString());
36.             return payment;
37.         }
38.         else {
39.             handleErrorResponse(entity, request, statusCode);
40.         }
41. 
42.         return null;
43.     }
44. 
45.     public Payment getPaymentById(String bearer, String paymentId)
46.     throws IOException, TransactionEndpointException {
47.         HttpGet request = new HttpGet(endpointURL + "/" + paymentId);
48. 
49.         request.setHeader("Authorization", bearer);
50.         request.setHeader("Accept", "application/json");
51. 
52.         HttpResponse response = client.execute(request);
53.         int statusCode = response.getStatusLine().getStatusCode();
54.         HttpEntity entity = response.getEntity();
55. 
56.         if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
57.             JSONObject inPayloadJson = readResponseJson(entity);
58.             reader = reader.forType(Payment.class);
59.             Payment payment = reader.readValue(inPayloadJson.getJSONObject("data").toString());
60.             return payment;
61.         }
62.         else {
63.             handleErrorResponse(entity, request, statusCode);
64.         }
65. 
66.         return null;
67.     }
```

12. Pielikums – TransactionHandler klases 2. koda fragments.

```
01. public PaymentList getPaymentList(String bearer, JSONObject filterData)
02. throws IOException, TransactionEndpointException {
03.     String queryParams = getQueryParamsFromJson(filterData);
04.     HttpGet request = new HttpGet(endpointURL + "?" + URIUtil.encodeQuery(queryParams));
05. 
06.     request.setHeader("Content-Type", "application/x-www-form-urlencoded");
07.     request.setHeader("Authorization", bearer);
08.     request.setHeader("Accept", "application/json");
09. 
10.     HttpResponse response = client.execute(request);
11.     int statusCode = response.getStatusLine().getStatusCode();
12.     HttpEntity entity = response.getEntity();
13. 
14.     if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
15.         JSONObject inPayloadJson = readResponseJson(entity);
16.         PaymentList paymentList = new PaymentList();
17.         reader = reader.forType(Payment[].class);
18.         paymentList.setPayments(Arrays.asList(reader.readValue(inPayloadJson.getJSONArray("data").toString())));
19.         return paymentList;
20.     }
21.     else {
22.         handleErrorResponse(entity, request, statusCode);
23.     }
24. 
25.     return null;
26. }
27. 
28. public Payment reversePayment(String bearer, JSONObject reverseData)
29. throws IOException, TransactionEndpointException {
30.     String paymentId = reverseData.getJSONObject("data").getString(PAYMENT_ID_FIELD);
31.     HttpPatch request = new HttpPatch(endpointURL + "/" + paymentId + "/reversal");
32. 
33.     request.setHeader("Content-Type", "application/json");
34.     request.setHeader("Authorization", bearer);
35.     request.addHeader("Accept", "application/json");
36. 
37.     StringEntity requestEntity = new StringEntity(reverseData.toString());
38.     request.setEntity(requestEntity);
39. 
40.     HttpResponse response = client.execute(request);
41.     int statusCode = response.getStatusLine().getStatusCode();
42.     HttpEntity entity = response.getEntity();
43. 
44.     if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
45.         JSONObject inPayloadJson = readResponseJson(entity);
46.         reader = reader.forType(Payment.class);
47.         Payment payment = reader.readValue(inPayloadJson.getJSONObject("data").toString());
48.         return payment;
49.     }
50.     else {
51.         handleErrorResponse(entity, request, statusCode);
52.     }
53. 
54.     return null;
55. }
```

13. Pielikums – TransactionHandler klases 3. koda fragments.

```
01.  public Payment finishPayment(String bearer, JSONObject finishData)
02.  throws IOException, TransactionEndpointException {
03.      String paymentId = finishData.getJSONObject("data").getString(PAYMENT_ID_FIELD);
04.      HttpPatch request = new HttpPatch(endpointURL + "/" + paymentId + "/finish");
05.
06.      request.setHeader("Content-Type", "application/json");
07.      request.setHeader("Authorization", bearer);
08.      request.addHeader("Accept", "application/json");
09.
10.      StringEntity requestEntity = new StringEntity(finishData.toString());
11.      request.setEntity(requestEntity);
12.
13.      HttpResponse response = client.execute(request);
14.      int statusCode = response.getStatusLine().getStatusCode();
15.      HttpEntity entity = response.getEntity();
16.
17.      if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
18.          JSONObject inPayloadJson = readResponseJson(entity);
19.          reader = reader.forType(Payment.class);
20.          Payment payment = reader.readValue(inPayloadJson.getJSONObject("data").toString());
21.          return payment;
22.      }
23.      else {
24.          handleErrorResponse(entity, request, statusCode);
25.      }
26.
27.      return null;
28.  }
29.
30.  public Payment refundPayment(String bearer, JSONObject refundData)
31.  throws IOException, TransactionEndpointException {
32.      HttpPost request = new HttpPost(endpointURL + "/refund");
33.
34.      request.setHeader("Content-Type", "application/json");
35.      request.setHeader("Authorization", bearer);
36.      request.addHeader("Accept", "application/json");
37.
38.      StringEntity requestEntity = new StringEntity(refundData.toString());
39.      request.setEntity(requestEntity);
40.
41.      HttpResponse response = client.execute(request);
42.      int statusCode = response.getStatusLine().getStatusCode();
43.      HttpEntity entity = response.getEntity();
44.
45.      if(statusCode < HttpURLConnection.HTTP_BAD_REQUEST) {
46.          JSONObject inPayloadJson = readResponseJson(entity);
47.          reader = reader.forType(Payment.class);
48.          Payment payment = reader.readValue(inPayloadJson.getJSONObject("data").toString());
49.          return payment;
50.      }
51.      else {
52.          handleErrorResponse(entity, request, statusCode);
53.      }
54.
55.      return null;
56.  }
57.
58.  private void handleErrorResponse(HttpEntity entity, HttpRequestBase request, int statusCode)
59.  throws IOException, TransactionEndpointException {
60.      String statusCodeStr = String.valueOf(statusCode);
61.      if(entity == null) {
62.          request.abort();
63.          throw new TransactionEndpointException(statusCodeStr);
64.      }
```

14. Pielikums – Configuration klases koda fragments.

```
01.     //No need to be synchronized since mutiple threads are only expected to read the data concurrently.
02.     private Map<String, String> confMap;
03.
04.     public Configuration(String confLocation) throws InvalidConfigurationException, IOException {
05.         Properties properties = new Properties();
06.         try(FileInputStream inStream = new FileInputStream(confLocation)) {
07.             properties.load(inStream);
08.         }
09.         confMap = convertPropsToMap(properties);
10.         validateConfMap();
11.     }
12.
13.     public Configuration(Map<String, String> confMap) throws InvalidConfigurationException {
14.         this.confMap = confMap;
15.         validateConfMap();
16.     }
17.
18.     private void validateConfMap() throws InvalidConfigurationException {
19.         for(String reqField : REQ_CONF_GENERAL) {
20.             String value = getConfData(reqField);
21.             if(value == null) {
22.                 throw new InvalidConfigurationException("Expected configuration entry \"" + reqField + "\" is missing a value");
23.             }
24.             if(AUTH_MODE_FIELD.equals(reqField)) {
25.                 if("certificate".equals(value)) {
26.                     allFieldsMustBePresent(AUTH_KEYSTORE_FIELDS);
27.                 }
28.                 else if("password".equals(value)) {
29.                     allFieldsMustBePresent(AUTH_CREDENTIALS_FIELDS);
30.                     ifOneThenAllMustBePresent(AUTH_KEYSTORE_FIELDS);
31.                 }
32.                 else {
33.                     throw new InvalidConfigurationException("Configuration entry \"" + reqField +
34.                                                             "\" can only be either \"password\" or \"certificate\"");
35.                 }
36.             }
37.         }
38.
39.         ifOneThenAllMustBePresent(AUTH_TRUSTSTORE_FIELDS);
40.         ifOneThenAllMustBePresent(TRN_KEYSTORE_FIELDS);
41.         ifOneThenAllMustBePresent(TRN_TRUSTSTORE_FIELDS);
42.
43.         for(String intField : INT_FIELDS) {
44.             String value = getConfData(intField);
45.             //the allFieldsMustBePresent method is not reused because we dont want to check if field is valid
46.             //if assigned default value.
47.             if(value == null) {
48.                 if(TIMEOUT_FIELDS.contains(intField)) {
49.                     confMap.put(intField, DEFAULT_TIMEOUT);
50.                 }
51.                 else if(MAX_POOL_FIELDS.contains(intField)) {
52.                     confMap.put(intField, DEFAULT_MAX_CONCURRENT_CONN);
53.                 }
54.             }
55.             else {
56.                 try {
57.                     int valueInt = Integer.parseInt(value);
58.                     if(valueInt < 0) {
59.                         throw new InvalidConfigurationException("Configuration entry \"" + intField + "\" is incorrectly formatted");
60.                     }
61.                     if(MAX_POOL_FIELDS.contains(intField) && valueInt == 0) {
62.                         throw new InvalidConfigurationException("Configuration entry \"" + intField + "\" is incorrectly formatted");
63.                     }
64.                 }
65.                 catch(NumberFormatException e) {
66.                     throw new InvalidConfigurationException("Configuration entry \"" + intField + "\" is incorrectly formatted");
67.                 }
68.             }
69.         }
70.      }
```

15. Pielikums – TransactionRequestValidator klases koda fragments.

```
01. public class TransactionRequestValidator {
02. 
03.     private String transactionType;
04. 
05.     public void validatePurchaseRequest(JSONObject paymentRequest) throws TransactionRequestException {
06.         transactionType = "Purchase";
07.         JSONObject attributes = paymentRequest.getJSONObject("data").getJSONObject("attributes");
08.         JSONObject meta = paymentRequest.getJSONObject("meta");
09. 
10.         String paymentType = attributes.getString(PAYMENT_TYPE_FIELD);
11. 
12.         fieldMustBePresent(attributes, AMOUNT_FIELD);
13.         fieldMustBePresent(attributes, CCY_CODE_FIELD);
14.         if("SMS".equals(paymentType)) {
15.             basketMustBeValid(attributes, BASKET_FIELD);
16.         }
17.         else {
18.             basketMustBeValid(attributes, PREAUTH_BASKET_FIELD);
19.         }
20.         if(fieldPresentAndTrue(meta, INVOICE_FIELD)) {
21.             fieldMustBePresent(attributes, ECOMM_NOTIFY_EMAIL_FIELD);
22.         }
23. 
24.         List<String> reqCardDataFields = Arrays.asList(PAN_FIELD, CARDNAME_FIELD, CSC_FIELD, EXPIRY_FIELD);
25.         List<String> reqSecureDataFields = new LinkedList(Arrays.asList(MD_STATUS_FIELD, SLI_FIELD, XID_FIELD, AAV_FIELD, CAVV_FIELD));
26.         if(oneFieldPresent(attributes, reqCardDataFields)) {
27.             allFieldsMustBePresent(attributes, reqCardDataFields);
28.             if(oneFieldPresent(attributes, reqSecureDataFields)) {
29.                 reqSecureDataFields.remove(4);
30.                 reqSecureDataFields.remove(3);
31.                 allFieldsMustBePresent(attributes, reqSecureDataFields);
32.                 eitherOfTwoMustBePresent(attributes, AAV_FIELD, CAVV_FIELD);
33.             }
34.         }
35.         else {
36.             neitherFieldMustBePresent(attributes, reqSecureDataFields, "Can not send 3dSecure data without card data");
37.         }
38.     }
39. 
40.     public void validateReverseRequest(JSONObject reverseRequest) throws TransactionRequestException {
41.         transactionType = "Reversal";
42.         JSONObject data = reverseRequest.getJSONObject("data");
43.         JSONObject attributes = data.getJSONObject("attributes");
44. 
45.         fieldMustBePresent(data, PAYMENT_ID_FIELD);
46.         basketMustBeValid(attributes, REVERSAL_BASKET_FIELD);
47.         if(fieldPresent(attributes, REVERSAL_AMOUNT_FIELD)) {
48.             bothMustNotBePresent(attributes, FRAUD_FIELD, REVERSAL_AMOUNT_FIELD);
49.         }
50.     }
51. 
52.     public void validateFinishRequest(JSONObject finishRequest) throws TransactionRequestException {
53.         transactionType = "Finish";
54.         JSONObject data = finishRequest.getJSONObject("data");
55.         JSONObject attributes = data.getJSONObject("attributes");
56. 
57.         fieldMustBePresent(data, PAYMENT_ID_FIELD);
58.         basketMustBeValid(attributes, BASKET_FIELD);
59.     }
```

16. Pielikums – TokenHandlerTest klases 1. koda fragments.

```
@BeforeEach
    public void reset() throws IOException, AuthEndpointException, InvalidConfigurationException {
        wireMockRule = new WireMockRule(8986);
        wireMockRule.start();

        HttpClient client1 = new DefaultHttpClient();
        HttpClient client2 = new DefaultHttpClient();

        id = "test" + String.valueOf(testIncrement);

        Map<String, String> confMapAuthModeCertificate = new HashMap<String, String>();
        confMapAuthModeCertificate.put(AUTH_MODE_FIELD, "certificate");
        confMapAuthModeCertificate.put(AUTH_KEYSTORE_PATH_FIELD, "test");
        confMapAuthModeCertificate.put(AUTH_KEY_ALIAS_FIELD, String.valueOf(testIncrement));
        confMapAuthModeCertificate.put(AUTH_ENDPOINT_URL_FIELD, "http://localhost:8986/token");
        confMapAuthModeCertificate.put(CLIENT_ID_FIELD, "test");
        confMapAuthModeCertificate.put(CLIENT_SECRET_FIELD, "test");
        confMapAuthModeCertificate.put(TRN_ENDPOINT_URL_FIELD, "test");
        Configuration confManagerAuthModeCertificate = new Configuration(confMapAuthModeCertificate);

        Map<String, String> confMapAuthModePassword = new HashMap<String, String>();
        confMapAuthModePassword.put(AUTH_MODE_FIELD, "password");
        confMapAuthModePassword.put(USERNAME_FIELD, id);
        confMapAuthModePassword.put(PASSWORD_FIELD, "test");
        confMapAuthModePassword.put(AUTH_ENDPOINT_URL_FIELD, "http://localhost:8986/token");
        confMapAuthModePassword.put(CLIENT_ID_FIELD, "test");
        confMapAuthModePassword.put(CLIENT_SECRET_FIELD, "test");
        confMapAuthModePassword.put(TRN_ENDPOINT_URL_FIELD, "test");
        Configuration confManagerAuthModePassword = new Configuration(confMapAuthModePassword);

        tokenHandlerAuthModePassword = new TokenHandler(confManagerAuthModePassword, client1);
        tokenHandlerAuthModeCertificate = new TokenHandler(confManagerAuthModeCertificate, client2);
        testIncrement++;
    }

    @AfterEach
    public void stopWireMock() {
        wireMockRule.stop();
    }

    //No stub, thats how we can be sure it works right
    @Test
    public void testGetBearerIfTokenExistsAndIsActiveAuthModePassword()
    throws IOException, AuthEndpointException, InterruptedException {
        Token token = generateToken(id + "_access", "test", 60000, 0);
        TokenRepository.setToken(id, token);

        assertEquals("Bearer " + id + "_access", tokenHandlerAuthModePassword.getBearer());
    }

    @Test
    public void testGetBearerIfTokenExistsAndIsActiveAuthModeCertificate()
    throws IOException, AuthEndpointException, InterruptedException {
        Token token = generateToken(id + "_access", "test", 60000, 0);
        TokenRepository.setToken(id, token);

        assertEquals("Bearer " + id + "_access", tokenHandlerAuthModeCertificate.getBearer());
    }

    //Token repository is empty at the beginning, ensuring that it doesnt exist
    @Test
    public void testCreateNewSessionIfTokenDoesntExistAuthModePassword()
    throws IOException, AuthEndpointException, InterruptedException {
        setUpValidStubGrantTypePasswordAuthModePassword();

        assertEquals("Bearer " + id + "_access", tokenHandlerAuthModePassword.getBearer());

        Token token = TokenRepository.getToken(id);
```

17. Pielikums – TokenHandlerTest klases 2. koda fragments.

```
01.  @Test
02.  public void testCreateNewSessionIfAccessAndRefreshExpiredAuthModePassword()
03.  throws IOException, AuthEndpointException, InterruptedException {
04.      setUpValidStubGrantTypePasswordAuthModePassword();
05.
06.      Token token = generateToken("test", "test", 0, 0);
07.      TokenRepository.setToken(id, token);
08.
09.      assertEquals("Bearer " + id + "_access", tokenHandlerAuthModePassword.getBearer());
10.
11.      token = TokenRepository.getToken(id);
12.
13.      assertEquals(true, token != null);
14.      assertEquals(id + "_access", token.getAccessToken());
15.  }
16.
17.  @Test
18.  public void testCreateNewSessionIfAccessAndRefreshExpiredAuthModeCertificate()
19.  throws IOException, AuthEndpointException, InterruptedException {
20.      setUpValidStubGrantTypePasswordAuthModeCertificate();
21.
22.      Token token = generateToken("test", "test", 0, 0);
23.      TokenRepository.setToken(id, token);
24.
25.      assertEquals("Bearer " + id + "_access", tokenHandlerAuthModeCertificate.getBearer());
26.
27.      token = TokenRepository.getToken(id);
28.
29.      assertEquals(true, token != null);
30.      assertEquals(id + "_access", token.getAccessToken());
31.  }
32.
33.  @Test
34.  public void testRefreshSessionIfAccessExpiredButRefreshActiveAuthModePassword()
35.  throws IOException, AuthEndpointException, InterruptedException {
36.      setUpValidStubGrantTypeRefreshTokenAuthModePassword();
37.
38.      Token token = generateToken("test", id + "_refresh", 0, 60000);
39.      TokenRepository.setToken(id, token);
40.
41.      assertEquals("Bearer " + id + "_access", tokenHandlerAuthModePassword.getBearer());
42.
43.      token = TokenRepository.getToken(id);
44.
45.      assertEquals(true, token != null);
46.      assertEquals(id + "_access", token.getAccessToken());
47.  }
48.
49.  @Test
50.  public void testRefreshSessionIfAccessExpiredButRefreshActiveAuthModeCertificate()
51.  throws IOException, AuthEndpointException, InterruptedException {
52.      setUpValidStubGrantTypeRefreshTokenAuthModeCertificate();
53.
54.      Token token = generateToken("test", id + "_refresh", 0, 60000);
55.      TokenRepository.setToken(id, token);
56.
57.      assertEquals("Bearer " + id + "_access", tokenHandlerAuthModeCertificate.getBearer());
58.
59.      token = TokenRepository.getToken(id);
60.
61.      assertEquals(true, token != null);
62.      assertEquals(id + "_access", token.getAccessToken());
63.  }
```

18. Pielikums – PurchaseRequest klases koda fragments.

```
01. public class PurchaseRequest extends Request {
02. 
03.     private static final String DEFAULT_PAYMENT_TYPE = "SMS";
04.     private static final String CCY_CODE_REGEX = "^[\\d]{3}$";
05.     private static final String CSC_REGEX = "^[\\d]{3,4}$";
06. 
07.     public PurchaseRequest() throws TransactionDataFormatException {
08.         super();
09.         setPaymentType(DEFAULT_PAYMENT_TYPE);
10.     }
11. 
12.     public static PurchaseRequest create() throws TransactionDataFormatException {
13.         return new PurchaseRequest();
14.     }
15. 
16.     public PurchaseRequest setAmount(Long amount) throws TransactionDataFormatException {
17.         validateAmount(amount);
18.         attributes.put(AMOUNT_FIELD, String.valueOf(amount));
19.         return this;
20.     }
21.     public PurchaseRequest setCcyCode(String ccyCode) throws TransactionDataFormatException {
22.         validateCcyCode(ccyCode);
23.         attributes.put(CCY_CODE_FIELD, ccyCode);
24.         return this;
25.     }
26.     public PurchaseRequest setPaymentType(String paymentType) throws TransactionDataFormatException {
27.         validatePaymentType(paymentType);
28.         if(paymentType.equals("SMS")) {
29.             attributes.put(PAYMENT_TYPE_FIELD, paymentType);
30.             if(attributes.has(PREAUTH_BASKET_FIELD)) {
31.                 attributes.put(BASKET_FIELD, attributes.getJSONObject(PREAUTH_BASKET_FIELD));
32.                 attributes.remove(PREAUTH_BASKET_FIELD);
33.             }
34.         }
35.         else if(paymentType.equals("DMS") && attributes.has(BASKET_FIELD)) {
36.             attributes.put(PAYMENT_TYPE_FIELD, paymentType);
37.             if(attributes.has(BASKET_FIELD)) {
38.                 attributes.put(PREAUTH_BASKET_FIELD, attributes.getJSONObject(BASKET_FIELD));
39.                 attributes.remove(BASKET_FIELD);
40.             }
41.         }
42. 
43.         return this;
44.     }
45.     public PurchaseRequest setLanguage(String language) {
46.         attributes.put(LANGUAGE_FIELD, language);
47.         return this;
48.     }
49.     public PurchaseRequest setDescription(String description) {
50.         attributes.put(DESCRIPTION_FIELD, description);
51.         return this;
52.     }
53.     public PurchaseRequest setAcceptorId(String acceptorId) {
54.         attributes.put(ACCEPTOR_FIELD, acceptorId);
55.         return this;
56.     }
57.     public PurchaseRequest setCustomDetail(String detailName, String detailValue) {
58.         attributes.put(detailName, detailValue);
59.         return this;
60.     }
61.     public PurchaseRequest setInvoiceText(String text) {
62.         attributes.put(INVOICE_TEXT_FIELD, text);
63.         return this;
64.     }
```

Kvalifikācijas darbs „***Bibliotēka priekš tirgotāju integrācijas maksājumu sistēmā***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ***Ralfs Eduards Braunfelds*** **_____ ___.05.2019**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: ***prof. Jānis Zuters*** **_____ ___.05.2019.**

Recenzents: ***Jānis Mārtužs***

Darbs iesniegts 27.05.2019.
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Jānis Zuters*_____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
___.06.2019. prot. Nr. _____

Komisijas sekretārs(-e): _____