LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

SISTĒMU INTEGRĀCIJA IZMANTOJOT MULESOFT API VEIDOTAS SAVIENOJAMĪBAS PIEEJU

BAKALAURA DARBS

Autors: **Guntis Rubenis**

Studenta apliecības Nr.: gr18018

Darba vadītājs: Prof. Laila Niedrīte

RĪGA 2022

ANOTĀCIJA

Bakalaura darbā “Sistēmu integrācija izmantojot MuleSoft Api-veidotas savienojamības pieeju” tiek apskatīta Api-veidotas savienojamības pieeja un tās pielietošanas ieguvumi sistēmu integrēšanā.

Darbā tiek pētīta un analizēta literatūra par sistēmu integrāciju, tās izplatītākajiem lietošanas piemēriem un savienošanas veidiem, kā arī sistēmu integrācijas arhitektūras modeļiem un MuleSoft piedāvāto integrācijas platformu.

Arhitektūras modeļi tiek analizēti, un katram no tiem tiek noteiktas tā izmantošanas priekšrocības un trūkumi.

Darbā ir izstrādāta divu sistēmu integrācija izmantojot MuleSoft piedāvāto integrāciju platformu un API-veidotas savienojamības pieeju kā arī apkopoti ieviestās integrācijas ieguvumi.

Atslēgvārdi: MuleSoft, Api-led, sistēmu integrācija, iPaaS, Anypoint platforma

ABSTRACT

SYSTEM INTEGRATION USING MULESOFT API-LED CONNECTIVITY APPROACH

The bachelor's thesis "System integration using MuleSoft API-led connectivity approach" discusses the API-led connectivity approach and the benefits of its application in systems integration.

The literature on systems integration, its most commonly used examples, and connection methods, as well as system integration architecture models and the integration platform offered by MuleSoft are studied and analyzed in the work.

Architectural models are analyzed and the strengths and weaknesses of each of them are identified.

The integration of two systems using the API-led connectivity approach is developed and the benefits of the implemented integration are summarized.

Keywords: MuleSoft, Api-led, System Integration, iPaaS, Anypoint Platform

SATURS

APZĪMĒJUMU SARAKSTS

Apzīmējums	**Skaidrojums**
API (Application programming interface)	Lietojumprogrammas saskarne
Anypoint platform	Integrāciju platforma
AnyPoint Design Center	Integrāciju platformas komponents lietojumprogrammu izveidei
AnyPoint Exchange	Integrāciju platformas komponents lietojumprogrammu glabāšanai, dokumentēšanai
AnyPoint API Manager	Integrāciju platformas komponents lietojumprogrammu pārvaldībai
AnyPoint Runtime Manager	Integrāciju platformas komponents lietojumprogrammu izvietošanai
iPaaS (integration Platform as a Service)	Integrāciju platforma kā pakalpojums
RAML (RESTful API Modeling language)	API specifikāciju Modelēšanas valoda
PaaS (Platform as a Service)	Platforma kā pakalpojums
HTTP (Hypertext Transfer Protocol)	Programmas līmeņa protokols dokumentu pārsūtīšanai
Enterprise Service Bus (ESB)	Uzņēmuma pakalpojumu kopne
REST (Representational State Transfer)	Arhitektūras stils tīkla lietojumprogrammu projektēšanai.
Docker	Lietojumprogrammu virtualizācijas platforma
Kubernetees	Lietojumprogrammu automatizācijas un pārvaldības sistēma

Runtime Fabric	Konteineru serviss Mule lietojumprogrammu izvietošanai
PCE (Private Cloud Edition)	Vide, lai izvietotu Mule lietojumprogrammas uz lokālajiem serveriem
B2B (Business to Business)	Pieeja, kad viens uzņēmums izmanto cita uzņēmuma pakalpojumus.
SaaS (System as Software)	Sistēma kā lietojumprogramma
CSV (Comma separated values)	Faila formāts, kurš izmanto komatus vērtību atdalīšanai
XML (Extensible markup language)	Datu formāts, kurš ļauj definēt pielāgotas atzīmes vērtību saglabāšanai
JSON (Javascript Object Notation)	Viegls datu apmaiņas formāts
SOAP	Ziņojumapmaiņas protokols strukturētas informācijas apmaiņai
VM (Virtual Machine)	Virtuālā mašīna
SDLC (Software Development life Cycle)	Programmatūras izstrādes dzīves cikls
EDI (Electronic Data Interchange)	Elektroniskā datu apmaiņa
SOA (Service Oriented Architecture)	Uz servisu orientēta arhitektūra
ERP (Enterprise resource planning)	Lietojumprogramma, kas automatizē un palīdz plānot un pārvaldīt dažādus biznesa procesus.
CRM (Customer relationship management)	Lietojumprogramma, kas palīdz uzņēmumiem gūt ieskatu par savu klientu uzvedību un atbilstoši pielāgot uzņēmējdarbību to vajadzībām.

IEVADS

Bakalaura darba tēma “Sistēmu integrācija izmantojot MuleSoft API-veidotas savienojamības pieeju” ir ļoti aktuāla, jo mūsdienās arvien vairāk uzņēmumu savu biznesa darbību veic digitālajā vidē.

Saskaņā ar MuleSoft 2022. gada etalona ziņojumu, kurā dati tika iegūti intervējot 1050 dažādu uzņēmumu IT vadītājus visā pasaulē, uzņēmumu digitālās transformācijas temps turpina paātrināties.[1]

Digitālā transformācija ir process, kura laikā digitālās tehnoloģijas tiek ieviestas visās biznesa jomās, lai radītu jaunus vai pārveidotus esošos biznesa procesus un varētu pielāgoties mainīgajām biznesa un tirgus prasībām.[41]

Gandrīz trīs ceturtdaļas (72%) uzņēmuma un klientu mijiedarbības notiek digitālajā vidē, un 93% no visiem aptaujātajiem norāda, ka ātrums, ar kādu uzņēmumā tiek sākti jauni projekti ir daudz lielāks nekā pirms pieciem gadiem.[1] Papildus tam uzņēmumi, kuri nespēs veiksmīgi veikt digitālo transformāciju var saskarties ar būtiskiem finansiālajiem zaudējumiem. Ir aprēķināts, ka uzņēmums varētu vidēji zaudēt gandrīz $7 miljonus ($6, 846,979), ja tam neizdodas sekmīgi īstenot digitālās transformācijas iniciatīvas.[1]

Uzņēmumiem, uzsākot digitālo transformāciju, ir vairāki punkti, kuros var notikt neveiksmes, bet integrācijas jautājums ir lielākais drauds digitālajai transformācijai, kam seko risku pārvaldība un iekļaušanās noteikumos un standartos. Apmēram 88% no aptaujātajiem uzņēmumiem arī norādīja, ka integrācijas problēmas turpina palēnināt digitālo transformāciju, kas ir pat nedaudz vairāk nekā 2021. gada etalona ziņojumā 87%.[1]

Lai uzņēmumi spētu veiksmīgi darboties, tie izmanto daudzas un dažādas sistēmas un lietojumprogrammas, kas nodrošina dažādu uzņēmuma struktūrvienību darbību. Katra no šīm sistēmām vai lietojumprogrammām ir neatkarīga un rada datus, kas var būt nozīmīgi citām uzņēmuma struktūrvienībām, sistēmām vai lietojumprogrammām. Lai nodrošinātu veiksmīgu uzņēmuma darbību uzņēmumiem ir jānodrošina plūstoša datu apmaiņa starp visām tā struktūrvienībām, sistēmām un lietojumprogrammām.

Lai integrētu lietojumprogrammas, sistēmas un datus no visa uzņēmuma, organizācijas koncentrē resursus nepareizajās jomās, pielāgoto integrāciju projektēšanā izstrādē un testēšanā, kā rezultātā tām strauji pieaug integrāciju izmaksas. [1]

API-veidotas savienojamības pieeja var ievērojami samazināt šīs izmaksas, izmantojot automatizāciju, racionalizētus procesus un atkārtoti lietojamas integrācijas.[1]

Bakalaura darba mērķis ir izpētīt API-veidotas savienojamības pieeju, un apkopot, kādus ieguvumus tās izmantošana sniedz sistēmu integrēšanā.

Lai sasniegtu darba mērķi, tiek izvirzīts veikt sekojošus uzdevumus:

1. Izpētīt literatūru par sistēmu integrāciju.
2. Izpētīt literatūru par sistēmu integrāciju arhitektūras modeļiem.
3. Veikt sistēmu integrāciju arhitektūru modeļu analīzi.
4. Izpētīt MuleSoft piedāvātās intgrācijas platformas sniegtās iespējas.
5. Izveidot sistēmu integrāciju izmantojot MuleSoft piedāvāto integrāciju platformu un API-veidotas savienojamības pieeju.

Darbs sastāv no 6 nodaļām:

Darba pirmajā nodaļā “Sistēmu integrācija” tiek paskaidrots, kas ir sistēmu integrācija, kādi ir sistēmu integrācijas izplatītākie lietošanas piemēri un kādi ir sistēmu savienošanas izplatītākie veidi.

Darba otrajā nodaļā “Sistēmu integrācijas arhitektūras modeļi” tiek analizēti sistēmu integrācijas arhitektūras modeļi, tiek noteikts, kādas ir katra arhitektūras modeļa priekšrocības un trūkumi.

Darba trešajā nodaļā “MuleSoft piedāvātā integrāciju platforma” tiek apskatīta MuleSoft integrāciju platformas piedāvātās iespējas un tās loma API-veidotas savienojamības veidošanā.

Darba ceturtajā nodaļā “Api-veidotas savienojamības izstrāde” tiek izstrādāta un aprakstīta divu sistēmu integrācija, izmantojot API-veidotas savienojamības pieeju un MuleSoft piedāvāto integrāciju platformu.

Darba piektajā nodaļā “Rezultāti” tiek apkopoti darbā paveiktie rezultāti.

Darba sestajā nodaļā “Secinājumi” tiek izdarīti secinājumi, balstoties uz darbā iegūtajiem rezultātiem.

1. SISTĒMU INTEGRĀCIJA

Gandrīz eksponenciālais biznesa lietojumprogrammu pieaugums, lai gan sākotnēji tika atzinīgi novērtēts, ir licis uzņēmumiem saprast, ka tie ir zaudējuši kontroli pār saviem datiem un biznesa procesiem. Uzņēmumi neapzināti ir izveidojuši informācijas rezervuārus, un tiem bieži ir grūtības saskaņot atšķirīgos datus no dažādām lietojumprogrammām un apakšsistēmām. Tagad vairāk nekā jebkad agrāk uzņēmumi meklē veidus, kā atgūt kopējo labumu no šīm lietojumprogrammām, un, kā risinājums tam tiek izmantots sistēmu integrācija.[12]

Sistēmu integrācija ir IT vai inženiertehniskais process, kas saistīts ar dažādu apakšsistēmu, lietojumprogrammu vai komponentu savienošanu vienā lielā sistēmā. Tas nodrošina, ka katra integrētā apakšsistēma funkcionē tā, kā tai būtu nepieciešams. Sistēmu integrāciju izmanto arī, lai pievienotu esošajai sistēmai jaunu funkcionalitāti, savienojot to ar citām sistēmām vai lietojumprogrammām.[11]

Sistēmu integrāciju projekti kļūst arvien sarežģītāki, un, lai pārvarētu šos šķēršļus, ir jāizmanto jauns pieeju un rīku kopums. Visbiežāk sastopamās problēmas mūsdienu sistēmu integrācijās ir:[12]

Sadarbspējas trūkums: cīņa par to, lai atsevišķas sistēmas varētu sadarboties tikai pieaugs, jo lietojumprogrammas kļūst arvien daudzveidīgākas. Lietojumprogrammas joprojām tiek veidotas dažādos datu formātos un protokolos, savukārt mantotās sistēmas ir pārāk stingras un ierobežotas.

Nepārbaudītas ieviešanas: tā kā lietojumprogrammu vide turpina attīstīties, parādās jauni projekti ar jauniem izaicinājumiem un problēmām, kuras ar vecajām integrācijas metodēm nebūs iespējams atrisināt. Šiem projektiem būs nepieciešami rīki ar lielāku elastību, lai pielāgotos neparedzētiem izaicinājumiem un šķēršļiem.

Kapitāla ierobežojumi: lai arī uzņēmumi bieži maksā augstas cenas par lietojumprogrammām, tiem var būt ierobežots budžets sistēmas integrācijai. Sistēmas integrētāji vēlēsies koncentrēties uz integrācijas metodēm, kas ļauj pakāpeniski izmantot sistēmu, ļaujot uzņēmumam augt kopā ar sistēmu, nevis atteikties no būtiskām funkcionalitātēm, lai projekts būtu atbilstošs.

Inovāciju trūkums: pieaugot sarežģītībai, sistēmu integrētājiem būs kārdinājums atgriezties pie tradicionālām pieejām, kas ir pārbaudītas un šķiet drošas. Šī ierastā domāšana

ne tikai radīs integrācijas risinājumu ierobežotas priekšrocības, bet arī sistēmu integrētāji, kuri atkāpsies uz šādu rīcību, ātri atpaliks.

Nepietiekama apmācība: apmācība pēc izvietošanas bieži tiek ignorēta, un tā var neļaut uzņēmumam pilnībā izmantot priekšrocības, ko sniedz nesen integrētā sistēma. Ir svarīgi, lai sistēmu integrācijās tiek izmantotas kompānijas, kuriem ir gan viegli lietojamas platformas, gan pārbaudītas atbalsta komandas galalietotājiem.

Ņemot vērā šos pieaugošos šķēršļus, sistēmu integrācija vairāk nekā jebkad agrāk ir atkarīga no izvēlētajiem rīkiem, metodēm un integrācijas platformām.

Pareizi izvēlēti rīki un metodes veicot sistēmu integrāciju var būt noteicošais faktors veiksmīgai uzņēmuma darbībai un var dot dažādus ieguvumus, kas būtiski uzlabo uzņēmumā notiekošos procesus.

Integrējot uzņēmumā esošās lietojumprogrammas un sistēmas, uzņēmums var:[2]

- Nodrošināt kritiskās informācijas apmaiņu starp vairākiem sistēmas komponentiem un lietotājiem, kam šī informācija ir nepieciešama dažādu procesu veikšanai.
- Vienkāršot un automatizēt uzņēmumā notiekošos procesus, kas ļauj lietotājiem ātrāk piekļūt, nepieciešamajiem datiem uzlabojot efektivitāti.
- Izvairīties no datu dublēšanās un kļūdām, kas var rasties manuālas datu ievades rezultātā.
- Iegūt jaunas iespējas, jo visas uzņēmuma sistēmas un lietojumprogrammas ir savienotas un dati ir pieejami lietošanai.
- Optimizēt uzņēmuma resursu izmantošanu un samazināt izmaksas, jo savienojot sistēmas un automatizējot dažādus procesus, var izvairīties no dažādu atkārtotu darbu veikšanas.

1.1 Izplatītākie sistēmu integrācijas lietošanas piemēri

Šajā nodaļā ir apskatīti izplatītākie sistēmu integrācijas veidi, kas atbilst dažādām biznesa vajadzībām.

1.1.1 Mantoto sistēmu integrācija

Tā kā uzņēmumi nepārtraukti saskaras ar izaugsmi un paplašināšanos, tie saskaras arī ar problēmu, kas saistīta ar mantoto sistēmu integrāciju ar jaunām tehnoloģijām (tīmekļa,

mobilajām lietojumprogrammām, mākoņservisiem un citām). Mantotās sistēmas ir vecas, neelastīgas tehnoloģijas, kas ieviestas, lai atrisinātu iepriekšējās uzņēmējdarbības problēmas. Šīs sistēmas to ilgā darbmūža dēļ mēdz būt trauslas, novecojušas un grūti integrējamas ar jauniem mākoņ un tīmekļa pakalpojumiem. Uzņēmumi joprojām izmanto mantotās sistēmas, jo mantoto sistēmu nomaiņa ir apjomīgs un dārgs process.[3]

Viens no Business Technographics pētījumiem liecina, ka 75% no Ziemeļamerikas un Eiropas uzņēmumu IT budžeta tiek tērēti mantoto sistēmu darbībai un uzturēšanai, atstājot tikai 25% no budžeta jaunām investīcijām. Cits pētījums, ko veica Meritalk un Unisys inovāciju pārvaldes centrs, atklāja, ka gandrīz puse no ASV federālā IT budžeta ir izsmelta, atbalstot mantotās sistēmas.[3]

Mantoto sistēmu uzturēšana ir ne tikai dārga, tā apdraud uzņēmējdarbību, jo kavē turpmāko izaugsmi un paplašināšanos. Nespējot nodrošināt lietotājiem piekļuvi jaunākajām tehnoloģijām, ir grūti attīstīt jaunu biznesu. Mantotās sistēmas aizstāšana kopumā ir dārga, riskanta, prasa daudzu izstrādātāju pārkvalificēšanos un neņem vērā uzņēmuma turpmākās vajadzības. Neatkarīgi no tā, kur uzņēmumi griežas, šķiet, ka tie ir iestrēguši un nespēj atrast ideālu risinājumu savām mantotajām sistēmām.[3]

Tā vietā ,lai mantotās sistēmas aizstātu pilnībā, labāks un izdevīgāks variants ir to modernizēšana izveidojot sakaru kanālu ar jaunākām informācijas sistēmām un tehnoloģiju risinājumiem.

1.1.2 Uzņēmuma programmu integrācija

Uzņēmumu IT arhitektūras pēc savas būtības sastāv no daudzām sistēmām un lietojumprogrammām, kas nodrošina dažādus pakalpojumus, kurus uzņēmums izmanto ikdienas darbībai. Lai pārvaldītu piegādes ķēdi, klientu attiecības, darbinieku informāciju un biznesa loģiku, viena organizācija var izmantot atsevišķas sistēmas, kas vai nu ir izstrādātas savā uzņēmumā, vai arī ir licencētas no trešās puses. Šī modularizācija bieži ir vēlama. Teorētiski uzņēmuma darbības sadalīšana daudzās mazākās funkcionalitātēs ļauj viegli īstenot labākos un jaunākos tehnoloģiskos sasniegumus katrā jomā un ātri pielāgoties mainīgajām uzņēmējdarbības vajadzībām.[4]

Ar uzņēmuma programmu integrācijas palīdzību visas funkcijas tiek ievietotas vienā biznesa ķēdē un tiek automatizēta reāllaika datu apmaiņa starp dažādām lietojumprogrammām.[4]

1.1.3 Starpuzņēmumu integrācija

Starpuzņēmumu (B2B) integrācija ir biznesa procesu un komunikācijas automatizācija starp divām vai vairākām organizācijām. Tas ļauj organizācijām efektīvāk strādāt un tirgoties ar saviem klientiem, piegādātājiem un biznesa partneriem, automatizējot galvenos biznesa procesus. B2B integrācijas programmatūra nodrošina arhitektūru, kas nepieciešama uzņēmuma digitalizēšanai un informācijas ātrai maršrutēšanai caur organizācijas tirdzniecības ekosistēmu.[5]

Organizācijas atklāj, ka lēna, neefektīva un kļūdām pakļauta manuāla informācijas apstrāde nav ilgtspējīga un katram uzņēmumam ir savs sistēmu un lietojumprogrammu kopums failu un ziņojumu apmaiņai ar partneriem. Atšķirīgas tehnoloģijas apgrūtina saziņu.[5]

Lai sasniegtu tādus mērķus kā ieņēmumu palielināšana, tirgus paātrināšana un efektivitātes uzlabošana, organizācijām ir nepieciešams veiksmīgs biznesa tīkls — un tam ir nepieciešams mūsdienīgs B2B integrācijas risinājums. Izmantojot B2B rīkus, organizācijas var ātri un uzticami izveidot digitālu savienojumu un sazināties. Tas var samazināt laiku, kas nepieciešams, lai tirgū nonāktu jauni produkti un pakalpojumi, un palīdzēt uzņēmumiem sasniegt nepieciešamo dinamiku un spēju konkurēt.[5]

1.2 Izplatītākie sistēmu savienošanas veidi

Ir dažādi veidi, kā panākt savienojamību starp atsevišķām sistēmām vai lietojumprogrammām. Šajā nodaļā tiks apskatīti izplatītākie sistēmu savienošanas veidi.

1.2.1 Lietojumprogrammu saskarnes

Lietojumprogrammu saskarne (API) nodrošina problēmas abstrakciju un norāda, kā klientiem vajadzētu mijiedarboties ar programmatūras komponentiem, kas ievieš šīs problēmas risinājumu. Būtībā API definē atkārtoti lietojamus veidošanas blokus, kas ļauj modulāras funkcionalitātes daļas iekļaut galalietotāju lietojumprogrammās.[42]

API var uzskatīt par sastāvošu no diviem pamatelementiem:[13]

- Tehniskās specifikācijas, kas nosaka, kā var apmainīties ar informāciju starp programmām (kas pati par sevi sastāv no apstrādes pieprasījuma un datu piegādes protokoliem)

- Programmatūras saskarnes, kas kaut kādā veidā publicē šo specifikāciju.

Gadu gaitā API bieži ir aprakstījusi jebkāda veida vispārīgu savienojamības saskarni ar lietojumprogrammu. Tomēr pēdējā laikā mūsdienu API var raksturot ar sekojošām īpašībām:[7]

- API atbilst standartiem (parasti HTTP un REST), kas ir izstrādātājiem draudzīgi, viegli pieejami un plaši saprotami
- Pret tiem izturas drīzāk kā pret produktiem nekā pret kodu. Tie ir paredzēti patēriņam konkrētām mērķauditorijām (piemēram, mobilo sakaru izstrādātājiem), tie ir dokumentēti, un tie ir izstrādāti tādā veidā, lai lietotāji varētu saprast to uzturēšanu un dzīves ciklu.
- Tā kā API ir standartizēti, tiem ir daudz stingrāka disciplīna attiecībā uz drošību un pārvaldību.
- .Kā jebkurai citai izstrādājamai programmatūrai, arī API ir savs programmatūras izstrādes dzīves cikls (SDLC) – projektēšana, testēšana, veidošana, pārvaldība un versiju veidošana. Turklāt API ir labi dokumentēti, kas uzlabo to izmantošanu patērētājiem un versiju kontroli.

1.2.2 Tīmekļa āķi

Tīmekļa āķi (Webhooks) var uzskatīt par API tipu, ko vada notikumi, nevis pieprasījumi. Tā vietā, lai viena lietojumprogramma pieprasītu otrai atbildi, tīmekļa āķis ļauj vienai programmai nosūtīt datus citai programmai, tiklīdz notiek konkrēts notikums. Tīmekļa āķi dažkārt tiek dēvēti par "reversajiem API", jo saziņu iniciē lietojumprogramma, kas nosūta datus, nevis tā, kas tos saņem. Tā kā tīmekļa pakalpojumi kļūst arvien vairāk savstarpēji savienoti, tīmekļa āķi vairāk darbojas kā viegls risinājums, lai nodrošinātu reāllaika paziņojumus un datu atjaunināšanu bez nepieciešamības izstrādāt pilna mēroga API.[9]

Tīmekļa āķi parasti tiek izmantoti, lai pārsūtītu mazāka apjoma datus, kā arī tie palīdz nosūtīt un iegūt reāllaika atjauninājumus starp sistēmām. Viens no vispiemērotākajiem tīmekļa āķu izmantošanas scenārijiem ir gadījumi, kad lietojumprogrammai vai sistēmai ir nepieciešama reāllaika funkcija, un uzņēmums nevēlas izmantot API, jo nevēlas ieguldīt laiku un izšķērdēt resursus to uzstādīšanai un izveidei.[14]

1.2.3 Elektroniskā datu apmaiņa

Elektronisku datu apmaiņu (EDI), ir uzņēmējdarbības dokumentu starpuzņēmumu saziņa standarta formātā. EDI vienkāršā definīcija ir standarta elektroniskais formāts, kas aizstāj papīra formāta dokumentus, piemēram, pirkšanas pasūtījumus vai rēķinus. Automatizējot papīra transakcijas, uzņēmumi var ietaupīt laiku un novērst dārgas kļūdas, ko izraisa manuāla apstrāde.[10]

EDI transakcijās informācija tiek pārvietota tieši no datora lietojumprogrammas vienā organizācijā uz datora lietojumprogrammu citā. EDI standarti definē informācijas atrašanās vietu un secību dokumenta formātā. Mūsdienās nozares izmanto EDI integrāciju, lai koplietotu dažādus dokumentu tipus — sākot ar pirkšanas pasūtījumiem un rēķiniem un beidzot ar piedāvājumiem un aizdevumu pieteikumiem un vēl daudz ko citu.[45]

Visas EDI darbības tiek definētas pēc EDI ziņojumu standartiem. Tāpēc ir svarīgi, lai visam būtu piemēroti datu kvalitātes pārvaldības procesi. Ja dokumentā trūkst informācijas vai tā atrodas nepareizā vietā, iespējams, ka EDI dokuments var tik nepareizi apstrādāts.[10]

1.2.4 Starpprogrammatūra

Starpprogrammatūra piedāvā vispārīgus pakalpojumus, kas atbalsta lietojumprogrammu izkliedētu izpildi. Termins starpprogrammatūra norāda, ka tā ir programmatūra, kas atrodas starp operētājsistēmu un lietojumprogrammu un slēpj pamatā esošās tehnoloģijas sarežģītību no tajā darbināmās lietojumprogrammas.[44]

Ir daudz starpprogrammatūru veidu. Daži, piemēram, ziņu starpnieki vai transakciju apstrādes pārraugi, koncentrējas uz viena veida sakariem. Citi, piemēram, tīmekļa lietojumprogrammu serveri, vai mobilo ierīču starpprogrammatūras, nodrošina pilnu saziņas un savienojamības iespēju klāstu, kas nepieciešams, lai izveidotu noteikta veida lietojumprogrammu. Vēl citi, piemēram, integrāciju Platforma kā Pakalpojums (iPaaS) vai Uzņēmuma Pakalpojumu Kopne (ESB), darbojas kā centralizēts integrācijas mezgls visu uzņēmuma komponentu savienošanai.[8]

Starpprogrammatūras galvenā priekšrocība ir tā, ka tā var apstrādāt, manipulēt, bagātināt un organizēt datu plūsmu starp vairākām avota un mērķa sistēmām. Pamata līmenī starpprogrammatūra ļauj izstrādātājiem veidot lietojumprogrammas bez nepieciešamības izveidot pielāgotu integrāciju ikreiz, kad nepieciešams izveidot savienojumu ar lietojumprogrammu komponentiem (pakalpojumiem vai mikroservisiem), datu avotiem,

skaitļošanas resursiem vai ierīcēm. Tas tiek darīts, nodrošinot pakalpojumus, kas ļauj dažādām lietojumprogrammām un pakalpojumiem sazināties, izmantojot kopīgas ziņojumapmaiņas sistēmas, piemēram, JavaScript objektu notāciju (JSON), reprezentatīvo stāvokļu pārsūtīšanu (REST), paplašināto iezīmēšanas valodu (XML), vienkāršo objektu piekļuves protokolu (SOAP) vai tīmekļa pakalpojumus.[8]

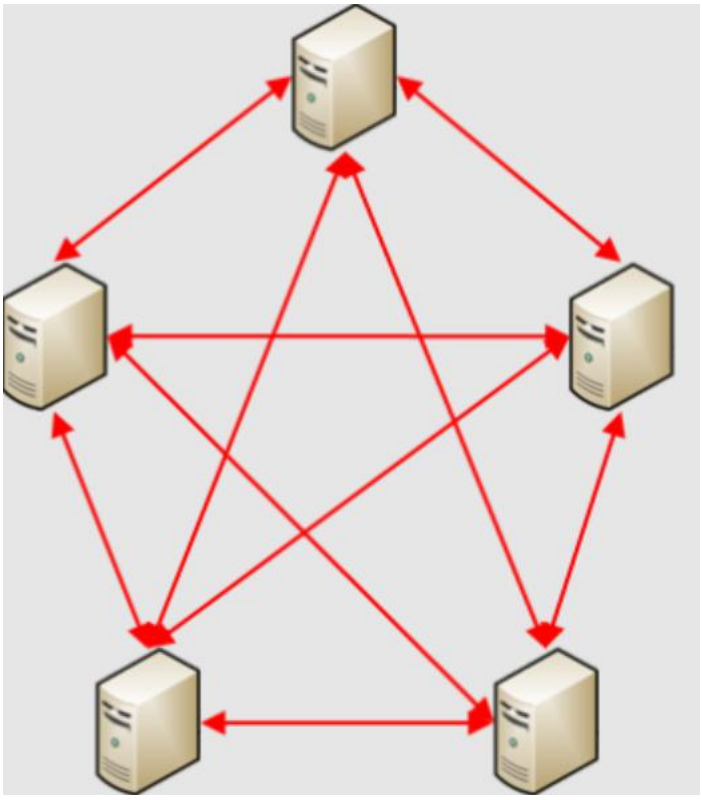
2. SISTĒMU INTEGRĀCIJAS ARHITEKTŪRAS MODEĻI

Sistēmas integrācija ir daudzšķautņaina, un to var pielietot, izmantojot dažādus arhitektūras modeļus, atkarībā no savienojamo komponentu skaita un rakstura.

Šajā nodaļa tiks apskatīti dažādi integrācijas arhitektūru modeļi. Iegūtā informācija palīdzēs veikt katra modeļa analīzi (skatīt nodaļu 2.5), lai noteiktu katra modeļa priekšrocības un trūkumus.

2.1 Point-to-point modelis

Point-to-Point integrācijas arhitektūra integrē katru lietojumprogrammu tieši ar katru no lietojumprogrammām, ar kurām tai ir jāsazinās (skatīt att. 2.1). Šī arhitektūra vēsturiski ir izmantota, lai nodrošinātu uz failiem balstītu integrāciju starp lietojumprogrammām.[15]



2.1 att. **Point-to-Point integrācijas arhitektūras modelis[48]**

Point-to-Point integrācijas ir ātri ieviešams risinājums, kas tikpat ātri var pārvērsties par lielu problēmu. Ja uzņēmuma infrastruktūrai ir tikai daži komponenti, point-to-point integrācija var šķist viegls veids, kā savienot visu kopā. Tomēr daudzi uzņēmumi gadu laikā ir sapratuši, ka point-to-point integrācijas ātri kļūst nepārvaldāmas, trauslas un kaitē gan IT budžetam, gan organizācijas spējai apmierināt pašreizējās un mainīgās biznesa prasības.

Neviena organizācija neplāno saskarties ar integrācijas problēmām un point-to-point integrācijās pārsvarā izveidojas laika gaitā straujo biznesa prasību izmaiņu dēļ.[16]

Ja uzņēmuma infrastruktūrā tiek izmantotas 3 vai mazāk sistēmas, un uzņēmums negaida nekādu izaugsmi nākotnē, point-to-point integrācijas ir piemērots risinājums.

Tomēr šim integrācijas modelim ir daudz trūkumu, kas ir saistīti ar mērogojamību, drošību un dārgām uzturēšanas maksām, ko rada nepārtraukta, nepieciešamība uzraudzīt, uzturēt un atjaunināt vairākas atsevišķas lietojumprogrammas un sistēmas.

Eksponenciāls sarežģītības pieaugums: to var būt grūti pamanīt, ja tiek izmantotas tikai 2 vai 3 komponenti, taču point-to-point savienojumu skaits, kas nepieciešams, lai integrētu noteiktu komponentu skaitu, pieaug eksponenciāli, pievienojot papildu komponentus. Tā ir īpaši nopietna problēma uzņēmumiem, kas sava biznesa modeļa ietvaros paļaujas uz savienojamību ar arvien lielāku komponentu skaitu.

Projektam, kas ietver tikai 3 komponentus, uzņēmumam būs nepieciešami tikai trīs savienojumi. Tomēr, pievienojot vēl tikai divas komponentus, šis skaitlis palielinās uz desmit savienojumiem. Pilna mēroga uzņēmumu tīkliem ar 10 vai vairāk komponentiem būs nepieciešami 45 vai vairāk savienojumu. Katrs no šiem savienojumiem rada papildus darba stundas, kas saistītas ar to dokumentāciju un apkopi.[16]

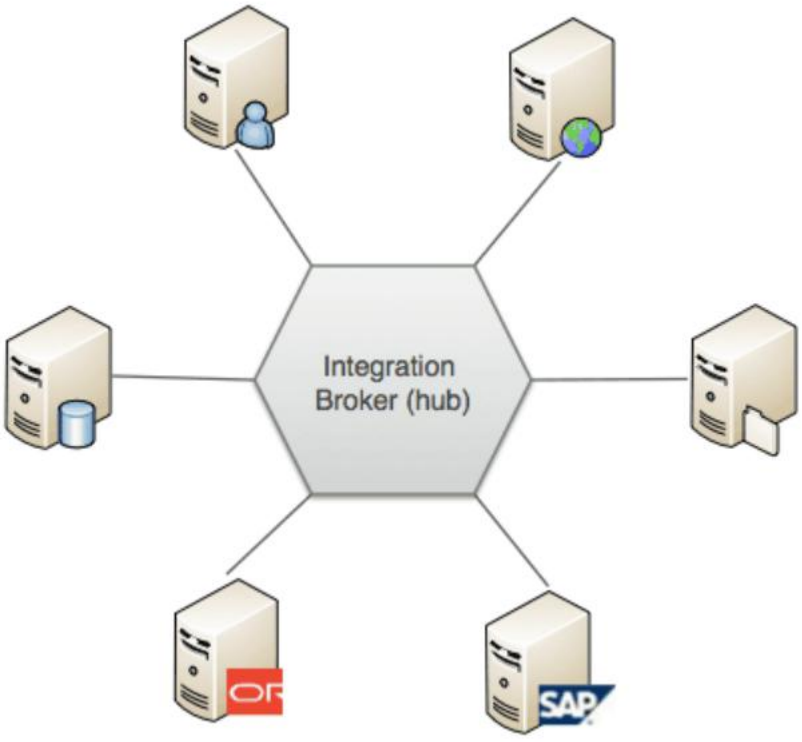
Atsevišķi vājie punkti: Neatkarīgi no tā, kādu integrācijas arhitektūru tiek izmantota, izvairīšanos no atsevišķiem vājajiem punktiem vienmēr vajadzētu būt galvenajai prioritātei. Viens no galvenajiem iemesliem, kāpēc būtu jāizvairās no point-to-point integrācijas, ir tas, ka iegūtā arhitektūra grauj uzticamību. Savienotāju skaits, kas nepieciešams liela mēroga integrācijas nodrošināšanai izmantojot point-to-point pieeju, padara šo arhitektūras veidu neiedomājami sarežģītu. Pieaugot point-to-point arhitektūrai, to kļūst arvien grūtāk dokumentēt, kas palēnina reakcijas laiku neparedzētu problēmu gadījumā. Point-to-point interācija sadala integrāciju vairākos komponentos, taču to dara, neizmantojot komponentus atkārtoti. Tas apgrūtina vājo vietu novēršanu un problemātisko zonu nostiprināšanu.[16]

Nav rīcības veidu ārkārtas gadījumos: ja kritiska drošības atjauninājuma dēļ kāda no sistēmām kļūst nesaderīga ar savienotāja modeli, tad sistēma nedarbosies, tik ilgi, līdz izmaiņas tiks veiktas visos savienotājos, kas savieno šo sistēmas daļu ar citiem komponentiem. Papildus, ja ņem vērā, ka izmaiņu veikšana nav tikai viens soli, bet gan process, kas iekļauj izstrādi, testēšanu un izvietošanu, tad uzreiz ir skaidrs problēmas patiesais lielums.[16]

Dinamikas zudums: Point-to-point integrācijas arhitektūra veido cieši savienotus savienojumus starp komponentiem. Lai nomainītu kādu no infrastruktūras komponentu, katrs savienotājs ir jāpārveido, lai tas darbotos ar jauno sistēmu. Jebkurš jauns komponents ir jāsaista ar katru citu sistēmu infrastruktūrā, izmantojot jaunus cieši savienotus savienojumus. Šāda dizaina rezultātā tiek zaudēta dinamika IT jomā un visā organizācijā kopumā. Mantoto sistēmu sadalīšana atsevišķos servisos kļūs neiespējama iesaistīto savienojumu skaita dēļ. Integrācija ar partneru sistēmām aizņem gadus, nevis mēnešus, un dažos gadījumos to nav iespējams izdarīt vispār. Uzņēmuma departamenti ir spiesti strādāt ar novecojušām tehnoloģijām, jo ir pārāk sarežģīti integrēt jaunas, kā rezultātā sāk zust uzņēmuma produktivitāte.[16]

2.2 Hub-and-spoke modelis

Hub-and-Spoke arhitektūra nodrošina centrālu starpnieku (sauktu par centrmezglu), caur kuru lietojumprogrammas sazinās savā starpā (skatīt att. 2.2). Šajā arhitektūrā katra lietojumprogramma sazinās ar starpnieku (centrmezglu), kas savukārt pārvalda saziņu ar citām lietojumprogrammām. Lietojumprogrammas nesazinās tieši viena ar otru.[43]



2.2 att. **Hub-to-Spoke integrācijas arhitektūras modelis[47]**

Hub-and-Spoke arhitektūras kontekstā centrmezgls nodrošina centrālo punktu dažādiem pakalpojumiem:[15]

Ziņojumu maršrutēšana un komunikācija: centrmezgls nodrošina centrālu iekārtu, kas pārvalda ziņojumu maršrutēšanu un izplatīšanu starp dažādām lietojumprogrammām. Ja ziņojums, pamatojoties uz tā saturu vai tēmu, ir jānovirza uz vairāk nekā vienu lietojumprogrammu, Hub-and-Spoke arhitektūra nodrošina lielāku elastību nekā Point-to-Point arhitektūra.

Ziņojumu pārvaldība, izsekošana un auditēšana: centrmezgls uzglabā, pārbauda, un izseko ziņojumus, kas plūst starp dažādām lietojumprogrammām, tādējādi nodrošinot centrālu iekārtu, kurā tiek pārvaldīta visa mijiedarbība starp lietojumprogrammām.

Ziņojumu un datu transformācija: centrmezgls nodrošina ziņojumu un datu pārveidošanu, lai pārveidotu datus no sūtāmās lietojumprogrammas formāta uz saņemošās lietojumprogrammas formātu. Hub and-spoke arhitektūrā katra lietojumprogramma vienkārši pārveido datus kopējā skatā, no kura datus var pārvērst saņemošās lietojumprogrammas formātā. Kad nosūtošā lietojumprogramma nosūta ziņojumu jaunai lietojumprogrammai, centrmezglam ir tikai jāpiemēro jauna transformācija no kopējā skata uz jaunās lietojumprogrammas formātu. Šī funkcionalitāte novērš vajadzību pēc tiešas datu transformācijas starp katru lietojumprogrammu.

Biznesa procesu pārvaldība: centrmezgls nodrošina biznesa procesu pārvaldības vai darbplūsmas pakalpojumus, lai pārvaldītu daudzpakāpju mijiedarbību starp lietojumprogrammām. Piemēram, viena lietojumprogramma nosūta ziņojumu citai lietojumprogrammai, pēc tam saņem atbildi no šīs lietojumprogrammas un pēc tam nosūta informāciju trešajai lietojumprogrammai. Centralizējot biznesa procesu pārvaldību centrmezglā, Hub-and-Spoke arhitektūra vienkāršo darbplūsmas pārvaldību starp dažādām lietojumprogrammām.

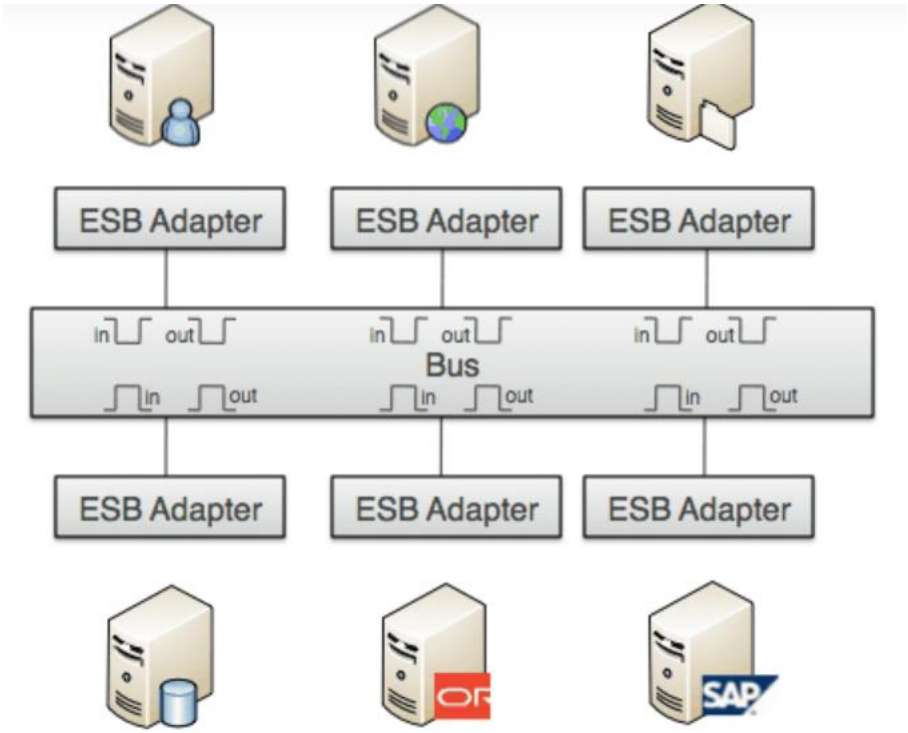
Lai gan Hub-and-Spoke arhitektūra ievieš papildu sarežģītību, lai pārvaldītu centrmezglu, tā piedāvā centrālo punktu, kurā ir vērsta integrācijai specifiska funkcionalitāte, lai nodrošinātu centralizētu pārvaldību, uzturēšanu, dublēšanu un atkopšanu, izsekošanu, maršrutēšanu, pārveidošanu un daudz ko citu. Centralizētā pieeja atvieglo lietojumprogrammu projektēšanu, izstrādi un pārvaldību, jo tām ir jāsazinās tikai ar centrmezglu nevis ar katru lietojumprogrammu atsevišķi.

Kā negatīvās šī modeļa īpašības var minēt augstas sākotnējās izmaksas, kas saistītas ar centrmezgla konfigurēšanu un uzstādīšanu, kā arī to, ka izstrādātājiem stingri jāievēro datu

koplietošanas modeļa noteikumus. Papildus tam, tā kā centrmezgls ir centrālais saziņas punkts, kas pārvalda visus ziņojumus starp lietojumprogrammām, lielas slodzes gadījumā tas var kļūt par vājo punktu vai kompromitēt organizācijas darbību, ja tas kļūst nepieejams.[43]

2.3 Service Orianted pieeja izmantojot Enterprise Service Bus modeli

ESB jeb Enterprise Service Bus ir modelis, kurā centralizēts programmatūras komponents veic sistēmu vai lietojumprogrammu integrāciju (datu modeļu tulkojumus, savienojumu maršrutēšanu un pieprasījumus) un padara šīs integrācijas un tulkojumus pieejamus kā pakalpojumu saskarnes atkārtotai izmantošanai jaunām sistēmām vai lietojumprogrammām (skatīt att. 2.3).[17]



2.3 att. **Enterprise Service Bus integrācijas arhitektūras modelis[47]**

ESB ir būtiska SOA jeb Service Oriented arhitektūras sastāvdaļa, kas definē veidu, kā programmatūras komponentus padarīt atkārtoti lietojamus, izmantojot pakalpojumu saskarnes. Šīs saskarnes izmanto kopīgus saziņas standartus tā, lai tās varētu ātri iekļaut jaunās lietojumprogrammās, katru reizi neveicot dziļu integrāciju. Katrs SOA pakalpojums ietver kodu un datu integrāciju, kas nepieciešama pilnīgas, atsevišķas biznesa funkcijas izpildei. Pakalpojumu saskarnes nodrošina brīvu savienojumu, kas nozīmē, ka tās var izsaukt nezinot, kā tiek īstenota pati integrācija. Pakalpojumi tiek piedāvāti, izmantojot standarta tīkla

protokolus, piemēram, SOAP/HTTP vai JSON/HTTP, lai nosūtītu pieprasījumus lasītu vai mainīt datus.[46]

Šīs pakalpojumu saskarnes var tik būvētas pilnīgi no jauna, bet bieži tās tiek veidotas, atklājot funkcijas no jau esošajām mantotajām sistēmām. Šeit arī rodas nepieciešamība pēc ESB izmantošanas, jo mantotās sistēmas un ierakstu sistēmas parasti izmanto vecus protokolus un novecojušus datu formātus, kas ir jātulko un jāintegrē, lai tie darbotos ar SOA tīkla protokoliem. SOA var tikt ieviests arī bez ESB palīdzības, bet tad sistēmu un lietojumprogrammu īpašniekiem pašiem jāatrod savs unikāls veids, kā piedāvāt pakalpojumu saskarnes un tas prasa lielu darbu, kā arī rada ievērojamas uzturēšanas problēmas nākotnē.[46]

Teorētiski ESB piedāvā iespēju standartizēt un ievērojami vienkāršot komunikāciju un pakalpojumu integrāciju visā uzņēmumā. Izstrādātāji var izmantot vienu protokolu, lai sazinātos ar ESB un izveidot pakalpojumu saskarnes, kas ir atbildīgas par komunikāciju ar aizmugursistēmām un datu pārveidošanu no lietojumprogrammas uz centrālās kopnes formātu, bet ESB ziņā atstāt tādas lietas kā komandu tulkošanu un ziņojumu maršrutēšanu. Tas ļautu izstrādātājiem tērēt ievērojami mazāk laika integrēšanai un daudz vairāk laika lietojumprogrammu konfigurēšanai un uzlabošanai. Un iespēja atkārtoti izmantot šīs integrācijas no viena projekta uz nākamo piedāvātu vēl lielāku produktivitātes pieaugumu un līdzekļu un resursu ietaupijumu.[17]

Lai arī ESB tika veiksmīgi ievietota daudzās organizācijās, tik pat daudzās tas tika uzskatīts par neveiksmes punktu SOA arhitektūras ieviešanā. Izmaiņu vai uzlabojumu veikšana vienā integrācijā bieži radīja problēmas citās un ESB atjauninājumi bieži ietekmēja esošās integrācijas. Tā kā ESB tika pārvaldīta centralizēti, izstrādes komandas drīz vien gaidīja rindā uz integrāciju izstrādi. Pieaugot integrāciju apjomam, ESB serveru augstas pieejamības un avārijas atkopšanas ieviešana kļuva dārgāka un kā starpuzņēmumu projekts, ESB izrādījās grūti finansējams. Galu galā izaicinājumi, kas saistīti ar centralizētas ESB uzturēšanu, atjaunināšanu un mērogošanu, izrādījās tik milzīgi un dārgi, ka ESB bieži vien aizkavēja to produktivitātes pieaugumu, kādu tai un SOA bija paredzēts sasniegt.[17]

2.4 API-veidotas savienojamības modelis

API-veidota savienojamība balstās uz SOA centrālajiem principiem. API-veidota savienojamība ir pieeja, kas nosaka metodes, lai savienotu un padarītu pieejamus uzņēmumā esošos resursus. Šī pieeja maina IT darbības veidu un veicina decentralizētu piekļuvi datiem

un pakalpojumiem vienlaikus, neapdraudot to pārvaldību. API-veidotas savienojamības rezultātā veidojas lietojumprogrammu tīkls. Šis tīkls sastāv no lietojumprogrammām, datiem un ierīcēm, kuras ir vienkārši savienojamas, lai nodrošinātu elastību un pielāgošanās spējas.[18]

API-veidotas savienojamības pieeja izmanto atšķirīgu savienošanas bloku, kas ietver sevī trīs sastāvdaļas:

Interfeisu: datu prezentācija pārvaldītā un drošā veidā.

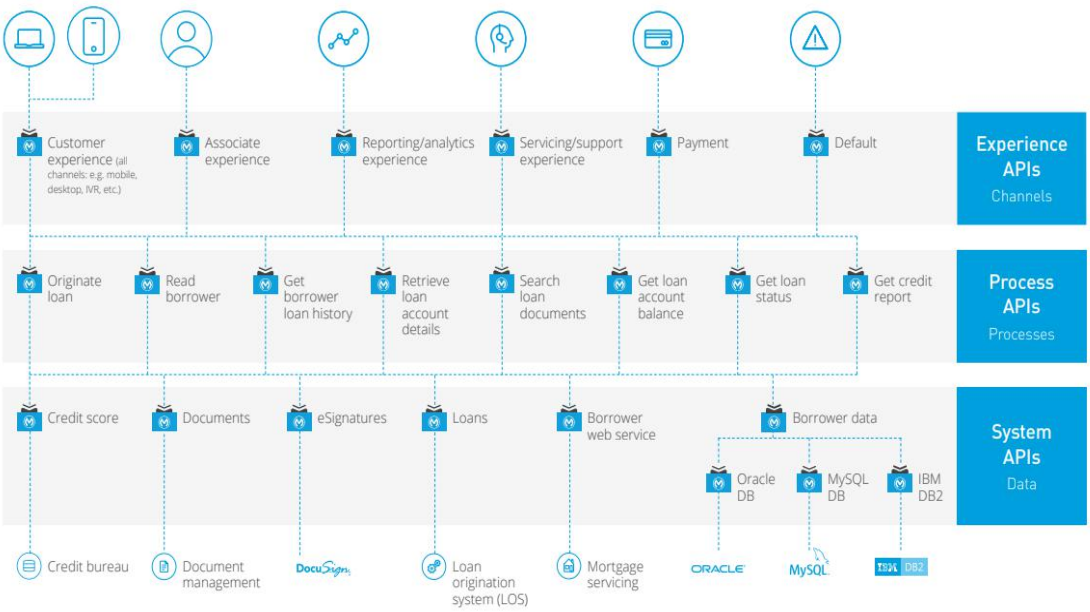
Orķestrāciju: loģikas pielietošana šiem datiem, piemēram, transformācija un bagātināšana.

Savienojamību: piekļuve avota datiem, neatkarīgi no tā, vai tie nāk no fiziskām sistēmām vai ārējiem pakalpojumiem.

API darbojas kā instrumenti, kas nodrošina datu patēriņu un kontrolē drošu savienojamības pieeju. Tie kalpo kā līgums starp datu patērētāju un datu piegādātāju, kas atdala abas puses un ļauj tām strādāt neatkarīgi vienam no otra. Tomēr integrāciju lietojumprogrammām jābūt vairāk nekā tikai parastam API. API var kalpot kā prezentācijas slānis, ja tas atrodas pāri orķestrācijas un savienojamības plūsmu kopumam. Izveidotajiem API ir jāveic speciālas funkcijas un jānodrošina piekļuve necentralizētiem datiem.[18]

2.4.1 Trīsslāņu API-veidotas savienojamības arhitektūra

Lieliem uzņēmumiem ir sarežģītas savstarpēji saistītas savienojamības vajadzības kam nepieciešami vairāki API-veidotas savienojamības bloki. Lai nodrošinātu šo bloku sakārtošanu un strukturēšanu, ir svarīgi izveidot ietvaru, kas to nodrošina, jo uzņēmuma dinamika un elastība var nāk tikai no vairāku līmeņu arhitektūras, kurā ir trīs atšķirīgi slāņi (skatīt att. 2.4).[18]



2.4 att. **API-veidotas savienojamības integrācijas arhitektūras modelis[18]**

Sistēmas slānis: visas IT arhitektūras pamatā ir kodola uzskaites sistēmas (piemēram, ERP, pamata klientu un norēķinu sistēmas, datu bāzes utt.). Bieži vien šīs sistēmas nav viegli pieejamas savienojamības problēmu dēļ. API nodrošina vienkāršu veidu, kā piekļūt šīm sistēmām, un noslēpj apakšā esošo sarežģītību no lietotāja. Sistēmas API nodrošina veidu, kā piekļūt pamatā esošajām sistēmām un padara to datus pieejamus vienā standartizētā formātā, vienlaikus nodrošinot izolāciju no jebkādām saskarnes vai apakšā esošo sistēmu izmaiņām. Šajos API izmaiņas notiks retāk un tie tiks pārvaldīti rūpīgāk, ņemot vērā to, ka apakšā atrodas nozīmīgas sistēmas.[18]

Sistēmas API galvenās funkcijas:[19]

- Atbild par aizmugursistēmu savienošanu un nepieciešamo neapstrādāto datu iegūšanu no tām.
- Iegūto datu pārveidošanu.
- Iegūto neapstrādāto datu tīrīšanu.
- Padarīt pieejamus iegūtos datus augstāk esošajiem API (Procesa vai Pieredzes API) drošā un uzticamā veidā.
- Aizmugursistēmas kļūdu apstrāde.

Procesa slānis: pamatā esošie biznesa procesi, kas mijiedarbojas un veido datus, ir stingri jānodala no avota sistēmām, kur dati tiek iegūti, kā arī no mērķa sistēmām, uz kurām dati ir jānogādā.[18]

Procesa API ir atbildīgi par vairāku pakārtoto API organizēšanu (t.i., datu apkopošanu no vairākiem pakārtotajiem API).

Procesa API galvenās funkcijas:[19]

- Padarīt pieejamus pārveidotos datus augstāk esošajiem API (Pieredzes API) drošā un uzticamā veidā.
- Biznesa loģikas ieviešana, maršrutēšana un datu apkopošana atbilstoši biznesa vajadzībām.
- Sistēmas API kļūdu apstrāde.
- Atbildīgs par Sistēmas API izsaukumu organizēšanu.

Pieredzes slānis: Dati tiek patērēti no plaša mēroga kanāliem, kuri vēlas iegūt vienus un tos pašus datus, bet dažādās formās. Piemēram, mazumtirdzniecības Point of Sale (POS) sistēma, e-komercijas vietne un mobilā iepirkšanās lietojumprogramma var vēlēties piekļūt vieniem un tiem pašiem klienta informācijas laukiem, bet dažādos formātos. Pieredzes API ir veids, kā pārveidot datus, no kopējā datu avota, tā, lai tie atbilstu katra patērētāja vajadzībām un neveidot, katram kanālam jaunu Point-to-Point integrāciju.[18]

Pieredzes API galvenās funkcijas:[19]

- Kļūdu apstrāde. Kļūdu kartēšana no procesa vai sistēmas API.
- Datu iegūšana no sistēmas vai procesa API.
- Datu transformāciju atbilstoši patērētāju vajadzībām.
- Patērētāja kanālam specifiska API atklāšana.

Šādi veidojot un organizējot API un pēc tam padarot tos atklājamus un pieejamus uzņēmuma pašapkalpošanās vajadzībām, API-veidota savienojamība padara uzņēmumu pielāgojamu, ļaujot uzņēmuma struktūrvienībām izmantot, pārveidot un pielāgot šos API, lai tie atrisinātu mainīgās uzņēmuma vajadzības.

2.4.2 API-veidotas savienojamības priekšrocības un trūkumi

API-veidotas savienojamības priekšrocības:[18]

Veicina uzņēmējdarbību: piedāvājot datus, kā pakalpojumus šī pieeja piedāvā, dažādām uzņēmējdarbības struktūrvienībām, pašapkalpošanās iespēju.

Veicina izstrādātāju produktivitāti: API-veidota savienojamība ir balstīta uz SOA pieeju, kur dažādi arhitektūras elementi ir atdalīti viens no otra un atkārtoti izmantoti, dažādās lietojumprogrammās. Tas novērš dublēšanos un ļauj izstrādātājiem veidot savas integrācijas uz jau esošiem resursiem.

Paredzamākas izmaiņas: modularizējot integrācijas loģiku un, nodrošinot loģisku sadalīšanu starp moduļiem, IT spēj labāk novērtēt un nodrošināt piegādi attiecībā pret koda izmaiņām. Šī arhitektūra ļauj izvairīties no scenārijiem, kad nelielas izmaiņas datubāzes laukos veido būtiskas izmaiņas visā arhitektūrā un palīdz izvairīties no biežām regresa testu pārbaudēm.

Atdalīta un pielāgota pieeja: Šī pieeja saprot, ka nav tādas universālas arhitektūras, kas der pilnīgi visiem. Tas ļauj savienojamībai tikt ieviestai mazos komponentos, ko pēc tam var padarīt pieejamus izmantojot API.

Lielāka pielāgojamība: organizāciju IT arhitektūras parasti tiek pārvaldītas dažādos līmeņos. Pamata sistēmās izmaiņas ir jāveic piesardzīgi, kontrolētā un pārbaudītā veidā, bet gala lietotājiem pieejamajām sistēmām ir jābūt pielāgojamām un izmaiņas jāveic straujāk un biežāk, lai tiktu līdzi inovācijām. Atsevišķi API līmeņi ļauj katrā no tiem ieviest atšķirīgas pārvaldības un kontroles metodes.

API-veidota savienojamība spēlē lielu lomu, arī IT darba slodzes samazināšanā. Bieži vien no IT tiek prasīts ieviest jaunas tehnoloģijas un veikt nepieciešamās izmaiņas esošajās, kā arī uzturēt mantotās sistēmas un to savienojumus ar citām sistēmām. Šīs prasības palielinās nepārtraukti, bet IT resursi paliek nemainīgi, kas beigās noved pie IT piegādes plaisas.[1]

Jaunu projektu skaits, ko IT ir nepieciešams ieviest, salīdzinājumā ar IT spēju to piegādāt tikai pieaug. Jaunu tehnoloģiju ieviešanas pieprasījumi no biznesa puses turpina pieaugt dubultā, bet IT resursi turpina pieaugt lineāri, neatkarīgi, cik daudz resursus biznesa pārstāvji novirza uz konkrēto problēmu. Pamatojoties uz MuleSoft etalona ziņojumu, tad lielākā daļa IT paredz, ka to budžets nemainīsies vai pieaugs tikai nedaudz, kas nozīmē to, ka neierobežoti resursi nav iespējami. Tieši tāpēc API-veidota savienojamība ir piemērotākā integrācijas pieeja.[1]

Izmantojot API-veidotas savienojamības pieeju IT projektu izstrādē uzņēmumi nodrošina projektu piegādi laikā un iekļaujoties budžetā, kā arī izveidotus atkal izmantojamus resursus, kas ietaupīs laiku un naudu katrā nākamajā projektā. Šī pieeja ļauj ātri ieviest jau pirmo projektu un tad paātrināt katru nākamo projektu, pateicoties atkārtojami izmantojamiem resursiem. Papildus tam ar katru projektu tiek veidota infrastruktūra, kas ir spējīga pielāgoties straujajām biznesa izmaiņu prasībām.[18]

API-veidotas savienojamības trūkumi:

Ja lielākā daļa no izveidotajiem API tiek patstāvīgi izmantoti dažādos integrāciju projektos, tad tas noteikti atsver ieguldījumus, kas tika veikti, izveidojot, dokumentējot, testējot un uzturot tos. Savukārt, ja izveidotie API netiek atkārtoti izmantoti, tie vienalga kādam ir jāuztur un prasa izvietošanas resursus, kas rada nepieciešamību pēc papildu cilvēku un finansiālajiem resursiem. Organizācijām izveidojot API, ir jādomā un jāparedz ne tikai konkrētā projekta vajadzības, bet arī turpmāko projektu iespējamās vajadzības un jāveido tie, lai tajos varētu viegli veikt izmaiņas vai pievienot jaunu funkcionalitāti.

2.5 Sistēmu integrāciju arhitektūras modeļu analīze

Šajā nodaļā veiksim iepriekš apskatīto sistēmu integrāciju arhitektūras modeļu analīzi, lai varētu saprast katras pieejas priekšrocības un trūkumi (skatīt tabulu 2.1).

Apkopojot informāciju par izplatītākajiem arhitektūras modeļiem (skatīt nodaļas 2.1-2.4) tika izvēlēts tos analizēt pēc sekojošiem raksturlielumiem:

- Integrācijas ieviešana
- Komponentu atkārtota izmantošana
- Uzturamība
- Mērogojamība
- Izmaksas

2.1 tabula **Integrāciju arhitektūras modeļu analīze**

	Point-to-Point	**Hub-and-Spoke**	**SOA ar ESB**	**API-veidota savienojamība**
Integrācijas ieviešana	Vienkārša ieviešana sākumā, bet sarežģītība pieaug ar	Sarežģīti sākumā, jo jāuzstāda un jākonfigurē	Sarežģīti sākumā, jo jāuzstāda un	Vienkārša ieviešana sākot no pirmās integrācijas, kas kļūst

	katru pievienoto sistēmu.	centrmezgls, bet pēc tam jaunu sistēmu pievienošana ir salīdzinoši vienkārša.	jākonfigurē ESB, bet pēc tam jaunu sistēmu pievienošana ir salīdzinoši vienkārša.	vienkāršāka ar katru nākamo projektu, jo parādās iespēja atkārtoti izmantot esošos komponentus.
Komponentu atkārtota izmantošana	Nav iespējama pielāgoto integrāciju dēļ.	Nav iespējama, visas darbības nodrošina centrmezgls.	Iespējama, katra integrācija piedāvā pakalpojumu saskarni, kuru ir iespējams izmantot atkārtoti.	Iespējama, API tiek veidoti ar mērķi, lai tos varētu izmantot atkārtoti.
Uzturamība	Grūti veikt izmaiņas, sistēmas cieši saistītas viena ar otru.	Vienkārši veikt izmaiņas, sistēmas nav cieši saistītas viena ar otru. Problēmas var sagādāt izmaiņu veikšana centrmezglā. Nepieciešama papildu testēšana.	Vienkārši veikt izmaiņas, sistēmas nav cieši saistītas viena ar otru. Problēmas var sagādāt izmaiņu veikšana ESB vai tā atjauninājumi. Nepieciešama papildu testēšana.	Vienkārši veikt izmaiņas, API nav cieši saistīti viens ar otru.
Mērogojamība	Nepieciešamo savienojumu skaits pieaug eksponenciāli, līdz ar to arī atsevišķas integrācijas un infrastruktūras prasības, kas padara šo pieeju ļoti grūti mērogojamu.	Datu pārveidošana un maršrutēšana notiek centrmezglā, līdz ar to jaunu sistēmu pievienošana palielina slodzi uz to. Lai veiktu mērogojamību,	Datu pārveidošana un maršrutēšana notiek pakalpojumu saskarnēs, kas atvieglo slodzi uz galveno	Šī pieeja nodrošina atkārtotu izmantojamību, un izveidotie API nav cieši saistīti viens ar otru, kas nozīmē to, ka mērogojamība ir iespējama pēc pieprasījuma.

		jāpievieno papildu centrmezgli, kas padara mērogojamību sarežģītu.	mezglu un padara mērogojamību vienkāršāku.	
Izmaksas	Zemas izmaksas sākuma, bet tās krietni pieaug ar katru jauno sistēmu.	Augstas sākuma izmaksas uzstādot un konfigurējot centrmezglu kā arī, pieaugot sistēmu skaitam, centrmezgls var kļūt par vājo vietu un var nākties uzstādīt papildu centrmezglus, kas krietni palielina izmaksas.	Dārgas izmaksas, lai nodrošinātu ESB serveru augstas pieejamības un avārijas atkopšanās iespējas. Iespēja atkārtoti izmanot esošās integrācijas var samazināt izmaksas.	Izmaksas, iespējams, samazinās ar katru nākamo projektu, jo parādās iespēja atkārtoti izmantot esošos komponentus. Ja izveidotie API netiek atkārtoti izmantoti, tas var sadārdzināt izmaksas.

Katrai no šīm pieejām ir savas priekšrocības un trūkumi, un katra no tām var tikt izmantota atsevišķos scenārijos.

Lieliem uzņēmumiem vajadzētu pēc iespējas vairāk izvairīties no Point-to-Point integrācijām, jo to ieviešana, uzturēšana un izmaksas krietni pieaug atkarībā no iesaistīto sistēmu skaita. Savukārt maziem uzņēmumiem, kuri izmanto tikai dažas sistēmas un nākotnē neplāno paplašināties, šis ir labākais risinājums, gan ieviešanas, gan arī izmaksu ziņā.

Arī Hub-and-Spoke pieeja nebūs labākais risinājums, uzņēmumiem, kuri izmanto daudzas sistēmas un nepārtraukti paplašinās, jo tā kā visas darbības tiek veiktas, izmantojot vienu centrālo punktu, tad palielinoties sistēmu skaitam uz to palielinās slodze un tas kļūs par vājo vietu. Šim pieejas veidam ir problēma arī ar mērogojamību, un to ir iespējams izdarīt tikai pievienojot un grupējot papildu centrmezglus, kas ir dārgs un sarežģīts process. Šis pieejas veids, gan ir uzlabojums, salīdzinot ar Point-to-Point integrācijām, jo tas krietni samazina nepieciešamo savienojumu skaitu (integrācija jāsavieno tikai ar centrmezglu), līdz ar to padara integrācijas vieglāk ieviešamas un uzturamas.

SOA pieeja izmantojot ESB ir nākamais solis integrāciju veidošanā, un daudzi uzņēmumi pēdējo gadu laikā ir izmantojuši tieši šo pieeju. Atšķirībā no Hub-and-Spoke šis modelis veic integrāciju pakalpojumu saskarnēs, kas krietni atvieglo slodzi uz centrmezglu un padara mērogojamību vienkāršāku. Papildus tam pakalpojumu saskarnes ir atkārtojami lietojamas, kā arī viegli savienojamas, un atvienojams no ESB, kas ļauj izmantot šīs integrācijas no viena projekta uz nākamo. Tomēr pats ESB var būt ļoti sarežģīts, tam ir nepieciešami ievērojami aparatūras resursi un izmaksas, kas saistītas ar ESB infrastruktūras ieviešanu un uzturēšanu ir ļoti augstas. Papildus tam ESB tiek pārvaldīts centralizēti, līdz ar to sagādājot dažādas problēmas, kas saistītas ar tā uzstādīšanu, uzturēšanu un izmaksām.

API-veidota savienojamība balstās uz SOA principiem, kur dažādi arhitektūras elementi ir atdalīti viens no otra un atkārtoti izmantoti, dažādās lietojumprogrammās. Atšķirībā no SOA ar ESB pieejas, šai pieejai nav centrālās kopnes, kas radīja problēmas tās uzturēšanā un sadārdzināja izmaksas, kā arī varēja kļūt par arhitektūras vājo punktu. Gan API-veidota savienojamība, gan SOA ar ESB piedāvā veidotās integrācijas izmantot atkārtoti citos projektos. API-veidotas savienojamības pieeja sadala integrācijas komponentus trīs dažādos slāņos, kas ļauj tos vieglāk uzturēt, pārvaldīt un aizsargāt. Šī pieeja ir elastīga, un nodrošina nepieciešamo dinamiku biznesa izmaiņu gadījumā, jo komponenti ir veidoti priekš atkārtotas izmantošanas, nav cieši saistīti viens ar otru un ir viegli savienojami. Atkārtojama komponentu izmantojamība ir svarīga arhitektūras sastāvdaļa un, ja lielākā daļa no izveidotajiem komponentiem netiek izmantoti atkārtoti, tad tas var radīt liekas izmaksas un vajadzību pēc papildus resursiem, kas var krietni sadārdzināt šo arhitektūras modeli.

3. MULESOFT PIEDĀVĀTĀ INTEGRĀCIJU PLATFORMA

Ross Meisons un Deivs Rosenbergs 2006. gadā nodibināja kompāniju "MulesSource", kas 2009 tika pārdēvēta par "MuleSoft". Uzņēmums sākotnēji nodrošināja starpprogrammatūras un ziņapmaiņas pakalpojumus, bet vēlāk paplašinājās un sāka nodrošināt iPaaS (intergration Platform as a Service) pakalpojumus izmantojot savu galveno produktu AnyPoint platform, kas ir viena no plašāk izmantotajām platformām lietojumprogrammu, datu un ierīču savienošanai[20]

MuleSoft ir saņēmis dažādas atzinības un Gartner jau septīto reizi ir atzinis uzņēmumu par līderi 2021. gada Magic Quadrant for Enterprise Integration Platform as a Service (iPaaS) programmā, kā arī sesto reizi pēc kārtas ir nosaukts par līderi 2021. gada Gartner Magic Quadrant for Full Life Cycle API Management programmā (skatīt att. 3.1).[21]

3.1 att. **Magic Quadrant for Enterprise Integration Platform as a Service[49]**

Uzņēmumiem digitalizējoties, tie sacenšas, lai ātri ieviestu jaunus produktus un pakalpojumus, radot vairāk funkcionalitātes un datu no vairākām sistēmām nekā jebkad agrāk. Saskaņā ar MuleSoft 2022. gada etalona ziņojumu IT šogad tiek prasīts pabeigt par 30% vairāk projektu, tomēr tikai 37% spēja pabeigt visus projektus, kas viņiem tika prasīti pagājušajā gadā.[1]

API-veidotas savienojamības pieeja var atvieglot un paātrināt projektu piegādi, bet lai varētu ieviest šo pieeju orgnizācijām ir nepieciešams sekojoš iespēju kopums:

- API projektēšana un izstrāde
- API izvietošana
- API pāraudzība un pārvaldība
- API dokumentēšana un pašapkalošanās

Nākamajās nodaļās apskatīsim MuleSoft piedāvāto AnyPoint integrāciju platformu, kas ietver vairākus rīkus, kas nodrošina API dzīvescikla pārvaldības iespējas, sākot no tā izveides līdz pat izbeigšanai. Tas sevī ietver API projektēšanu, publicēšanu, dokumentēšanu, izvietošanu, pārvaldību un analīzi.

3.1 Anypoint platforma

AnyPoint platforma ir hibrīd integrācijas platforma, kas atbalsta plašu integrācijas modeļu klāstu un ļauj organizācijām viegli izveidot un ātri mērogot lietojumprogrammu tīklu.

Platforma ļauj izstrādātājiem izveidot, pārvaldīt API un integrācijas no vienas vienotas platformas. Šis hibrīd integrācijas risinājums apvieno iPaaS, ESB un vienotu API pārvaldības risinājumu. Risinājums piedāvā REST un SOAP API, ko pārsvarā izmanto organizācijas, kurām ir jāveic pilnīga integrācija, apvienojot programmatūru un datu pārvaldību. Galvenās platformas iezīmes ir ātra izstrāde, savienojamība ar visām sistēmām, sadarbība un pašapkalpošanās, kā arī pārredzamība un kontrole.[22]

3.2 Design Center

Design Center ir izstrādes vide, kas ļauj integrētājiem veikt API specifikāciju izstrādāšanu un testēšanu, kā arī nelielu un vienkāršu integrāciju izstrādāšanu tieši platformā. Šis komponents sastāv no diviem izstrādes rīkiem API Designer un Flow designer.[34]

Ar API Designer palīdzību izstrādātāji var veidot API Specifikācijas vai to fragmentus, izmantojot dažādas modelēšanas valodas. Platforma atbalsta specifikāciju izstrādi sekojošās modelēšanas valodās:[33]

- RESTful API Modeling language (RAML)
- OpenAPI Specification (OAS)

- AsyncAPI

API specifikācija definē, kāda būs API funkcionālā darbība un kādi resursi un to tipi būs pieejami. Papildus tam katram resursam tiek noteikta izmantotā HTTP metode, definēti datu tipi un izveidotas pieprasījuma un atbildes struktūras.

Šīs specifikācijas kalpo kā dokumentācija vai līgums kas ir saprotams gan cilvēkiem, gan programmatūrai. Pēc specifikācijas izveidošanas to ir iespējams koplietot ar citiem organizācijas darbiniekiem, padarīt pieejamu plašākai sabiedrībai vai arī prezentēt klientam.[35]

Flow Designer ir grafiskais integrāciju izstrādes rīks, kas pieejams tieši platformā. Ar šī rīka palīdzību izstrādātāji var veikt vienkāršas integrācijas, pārvēršot sistēmas, API un lietojumprogrammas par darbplūsmām.[36] Sarežģītāku integrāciju veikšanai tiek izmantota atsevišķa izstrādes vide AnyPoint Studio, kuru tuvāk apskatīsim 3.6 nodaļā.

3.3 AnyPoint Exchange

AnyPoint Exchange veido globālā krātuvi visiem API vai Mule lietojumprogrammu komponentiem, kas ir izveidoti un izvietoti šajā vietā. Šī ir galvenā vieta, kur izstrādātāji var izvietot, atrast un dalīties ar izveidotajiem API, savienotājiem, veidnēm vai API specifikāciju fragmentiem. Platformas lietotājiem ir iespēja meklēt un izmantot jau iepriekš izveidotus un testētus resursus.[23]

Organizācijas var izveidot privātu Exchange portālu, kas paredzēts tikai konkrētās organizācijas pārstāvjiem. Tas ļauj glabāt organizācijas privātos resursus, kas paredzēti tikai organizācijas iekšējai izmantošanai un nav paredzēti publiskai izmantošanai.

Kad izveidotā API specifikācija tiek publicēta, Exchange portālā tad tai automātiski tiek izveidota dokumentācija un pats API tiek savienots ar visām tā atkarībām, piemēram, fragmentiem, savienojumiem un politikām.[23]

Šajā portālā ir iebūvēts arī testēšanas serviss, kas ļauj lietotājiem pārbaudīt un apskatīt konkrētā resursa funkcionalitāti, pirms tas ir ieviests reālā integrācijā. Tas ļauj novērtēt resursa atbilstību attiecīgās integrācijas vajadzībām un jau laikus saprast, vai tas būs noderīgs, tādā veidā ietaupot laiku un resursus.[23]

Exchange ir iespēja redzēt un sekot līdzi resursu atjauninājumiem, izmantot dažādas resursu versijas un saņemt paziņojumus, ja kādu no versijām tiek pārstāts izmantot.[23]

3.4 API Manager

API Manager ir platformas komponents, kas ļauj lietotājiem veikt centralizētu kontroli un pārraudzību pār izvietotajiem API un lietojumprogrammām. Šajā komponentā var uzstādīt dažādas API politikas, pārraudzīt konkrētās API instances datus kā arī nodrošināt šifrēšanu un autentifikāciju.[27]

Kad izstrādātāji ir publicējuši izveidotās API specifikācijas AnyPoint Exchange, tās ir jāimportē šajā rīkā, lai varētu veikt to pārvaldību. Šeit ir iespējams izveidot, izmainīt un slēgt API instances un to darīt dažādās izstrādes vidēs. Papildus tam izveidotās instances var grupēt ar citām API instancēm, lai izveidotu pakotnes, kas var atrisināt kādu konkrētu lietotāja problēmu.[28][29]

Izvietotajām API instancēm var uzstādīt dažādas politikas, kas palīdz regulēt drošu datu plūsmu un pielāgot API konkrētām vajadzībām. Piemēram, autentifikācijas politika var kontrolēt piekļuvi konkrētajai API instancei. Šis rīks piedāvā dažādas jau iepriekš definētas noklusētās politikas, bet lietotājiem ir iespēja izveidot arī savas pielāgotās politikas, ja tas ir nepieciešams.[30]

API Manager atļauj lietotājiem definēt arī dažādus pakalpojumu piekļuves līmeņus, ar kuru palīdzību ir iespējams uzstādīt papildu ierobežojumus. Piemēram cik daudz izsaukumus, noteiktā laika periodā izsaucošā lietotne var veikt konkrētajai API instancei. Izstrādātāji var nodefinēt vairākus šādus piekļuves līmeņus, katrai konkrētajai instancei un katram līmenim noteikt savu izsaukumu limitu. Tas nodrošina to, ka API instance netiks pārslogota ar nebeidzamiem izsaukumiem, kā arī to, ka izsaucošā lietojumprogramma, saņems paredzēto pieprasījumu skaitu.[31]

Šajā rīkā ir iespējams arī sekot līdzi API brīdinājumiem, kas var ziņot lietotājam, ja API instancei ir veiktspējas problēmas, tiek pārkāpta kāda no uzstādītajām politikām vai arī lietotājs pieprasa piekļuvi tai.[32]

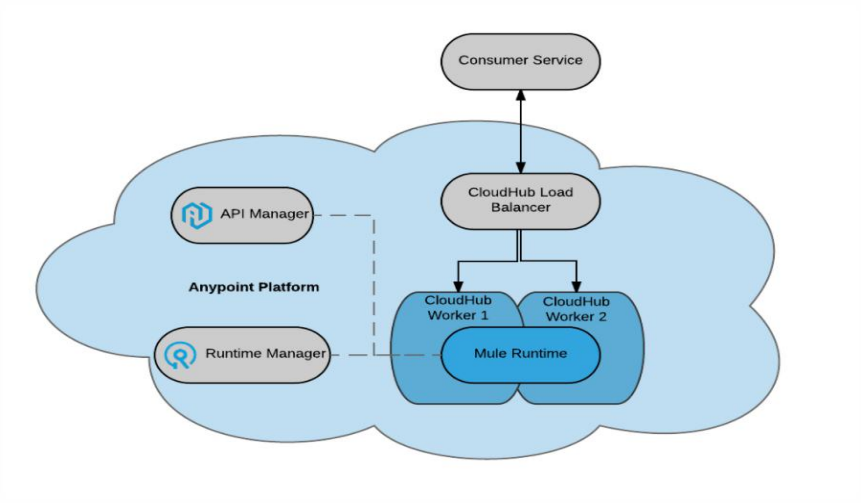
3.5 Runtime Manager

Runtime Manager ir AnyPoint platformas interfeiss, kas nodrošina kopēju skatu uz izvietotajām lietojumprogrammām, serveriem un API.

Izmantojot Runtime Manager lietotāji, var izvietot, pārvaldīt un pārraudzīt Mule lietojumprogrammas dažādās izstrādes vidēs no centrālas atrašanās vietas neatkarīgi no tā, vai lietotnes ir izvietotas mākoņ vai privātajā serverī.[24]

Runtime Manager ir pieejami šādas lietojumprogrammu un API izvietošanas iespējas:

CloudHub izvietojumi: Mule lietojumprogrammas vai API tiek izvietotas CloudHub, un to pārvaldība notiek ar Runtime Manager palīdzību (skatīt att 3.2).

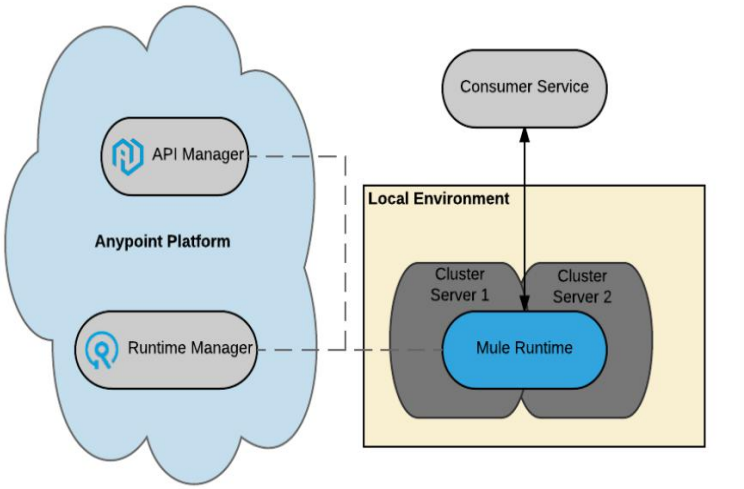


3.2 att. **Runtime Manager CloudHub izvietojums[25]**

CloudHub ir platforma, kā serviss (PaaS) pakalpojums un tas lietotājiem nodrošina visas nepieciešamās servera funkcijas, kas nozīmē to, ka nekādas papildus darbības no lietotāja puses nav nepieciešamas.

CloudHub atbild par izvietoto lietojumprogrammu un API uzraudzību, problēmu noteikšanu un novēršanu, kā arī plānošanu. CloudHub piedāvā dažādus mākoņservisa pakalpojumus, piemēram, vairāku nomnieku iespējamību, augstu pieejamību un mērogošanu pēc pieprasijuma.[24]

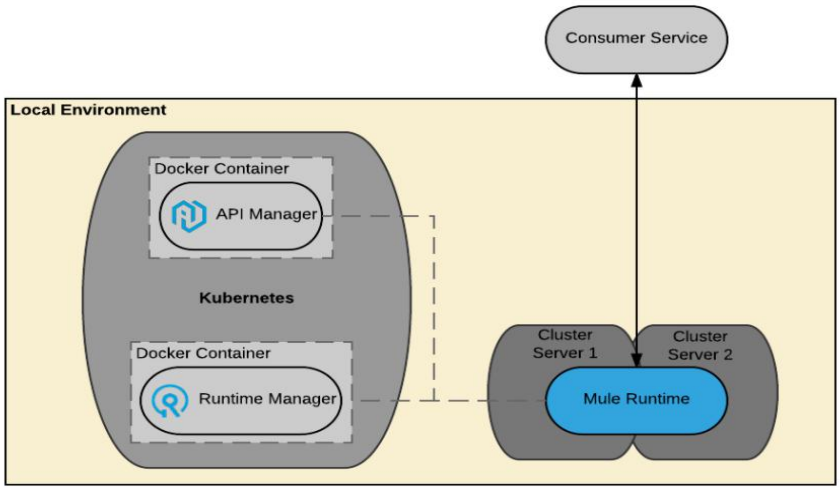
Hybrid izvietojumi: Hybrid izvietojumi piedāvā lietotājiem izvietot Mule lietojumprogrammas vai API uz personīgajiem serveriem un to pārvaldībai izmantot Runtime Manager (skatīt att. 3.3).



3.3 att **Runtime Manager Hibrīda izvietojums[25]**

Izmantojot šo izvietojumu iespēju, lietotāji var nodrošināt elastību un lielāku kontroli par lokālo serveru pielāgošanu un drošību, bet lietotājiem pašiem jānodrošina visa servera funkcijas un izvietošanas infrastruktūra. Lai šo iespēju varētu izmantot, lietotājiem no sākuma jāreģistrē lokālais serveris ar Runtime Manager aģentu. Tālāk lietotājs var pārvaldīt šos serverus un grupēt tos ar cietiem serveriem vai serveru grupām, lai nodrošinātu augstu pieejamību. Kad tas ir paveikts, tad lietotājs var sākt izvietot lietojumprogrammas.[25]

Private Cloud Edition izvietojumi: AnyPoint Platform Private Cloud Edition nodrošina iespēju izvietot un pārvaldīt Mule lietojumprogrammas vai API lokālajos serveros (skatīt att. 3.4).



3.4 att. **Runtime Manager PCE izvietojums[25]**

Izmantojot šo izvietošanas veidu, visa funkcionalitāte lietotājiem ir pieejama lokāli, un tiem nav nepieciešams sadarboties ar ārējām sistēmām vai arī publiski pieejams interneta pieslēgums. Lai nodrošinātu augstu pieejamību, šis izvietojumu veids izmanto Dockers vai Kubernetes.[26]

Šo izvietošanas veidu parasti izmanto uzņēmumi, kuriem ir stingras normatīvās vai atbilstības prasības, kas liedz izmantot mākoņrisinājumu pakalpojumus.[25]

AnyPoint Runtime Fabric izvietojumi: Šis izvietojumu veids darbojas, kā konteineru pakalpojums, kas ļauj darbināt Mule lietojumprogrammas vai API jebkurā vidē un kā centralizēto pārvaldības rīku izmantot AnyPoint platformu. Katra Mule lietojumprogramma darbojas savā izpildlaikā un atsevišķos konteineros. Katram konteineram var izdalīt nepieciešamos resursus brīdī, kad tajā tiek izvietotas Mule lietojumprogrammas vai API. Šī iespēja ļauj katrai Mule lietojumprogrammai vai API būt neatkarīgai un veikt horizontālo mērogojamību, kā arī to, ka vienā un tajā pašā mezglā dažādas lietojumprogrammas un API nekonkurē par vieniem un tiem pašiem resursiem.[25]

Lietotājiem ir divas iespējas, kā pārvaldīt Runtime Fabric izvietojumus:

Runtime Fabric uz pašpārvaldīta Kubernetes: Lietotājs uzstāda Runtime Fabric uz jau eksistējošas Kubernetes vides, un pats veic to pārvaldību. Šī versija atbalsta Azure Kubernetes Service (AKS), Amazon Elastic Kubernetes Service (Amazon EKS) un Google Kubernetes Engine (GKE) pakalpojumus.

Runtime Fabric on VM/Bare Metal: Šajā versijā MuleSoft nodrošina nepieciešamos infrastruktūras komponentus (arī Kubernetes un Dockers), kur tiek uzstādīta Runtime Fabric. Lietotājs var uzstādīt šo versiju virtuālajās mašīnās un tad to pārvaldību veikt pats[25]

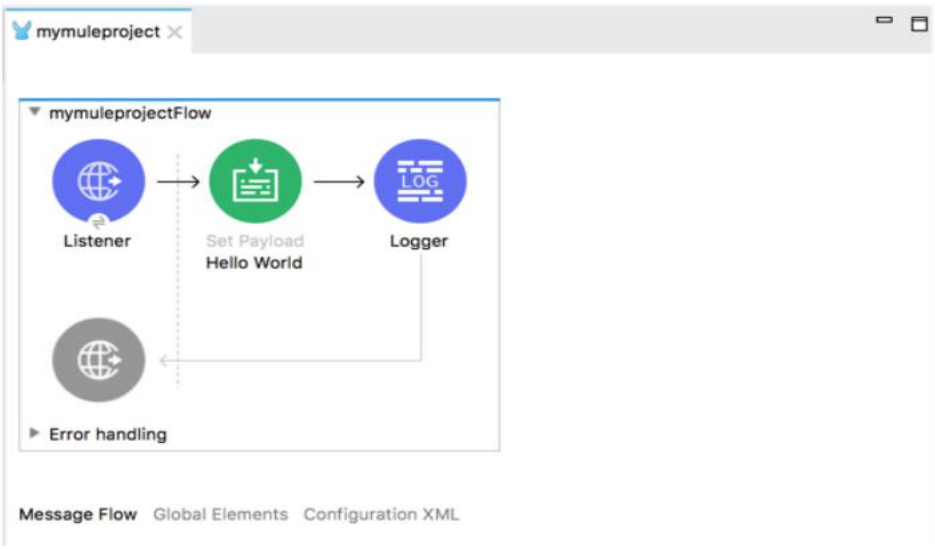
3.6 AnyPoint Studio

MuleSoft ir izstrādājis atsevišķu izstrādes vidi priekš integrāciju veikšanas, kas ir veidota uz Eclipse bāzes. Šī vide nodrošina Mule lietojumprogrammu ērtu un vienkāršu veidošanu un testēšanu. AnyPoint Studio piedāvā sekojošas funkcionalitātes:

- Iespēju darbināt Mule lietojumprogrammas lokālajā mašīnā.
- Vizuālos redaktorus, kas dod iespēju izstrādāt, mainīt, testēt API Specifikācijas un lietojumprogrammas.
- Tiešo savienojumu ar AnyPoint Exchange.

- Iebūvēto vienībtestēšanas rīku Munit.
- Tiešo savienojumu ar CloudHub, kas nodrošina lietojumprogrammu izvietošanu.[37]

Izstrādātāji var veikt izstrādi, gan izmantojot grafisko saskarni, gan arī rakstīt kodu tieši XML failos, kas ir Mule lietojumprogrammu pamatā. Izmantojot grafisko saskarni izstrādātājiem ir iespēja redzēt vizuālo attēlojumu izveidotajām integrāciju plūsmām, kas padara integrāciju veikšanu uzskatāmāku un vieglāk lasāmu (skatīt att. 3.5).[37]



3.5 att. **AnyPoint Studio Mule integrāciju plūsmas grafiskās saskarnes attēlojums[37]**

Grafiskā saskarne ir pieejami iepriekš izveidoti API, savienotāji, transformatori un maršrutētāji, ko izstrādātāji var vienkārši pievienot savai integrācijas plūsmai. Kad plūsmā tiek pievienots jauns elements, izstrādātājam ir jāveic tā konfigurēšana un jāpielieto atbilstošā biznesa loģika. Tā kā AnyPoint studija atbalsta savienojumu ar AnyPoint Exchange portālu, tad izstrādātāji var viegli pievienot jaunus elementus (API, savienotājus, API Specifikācijas, RAML fragmentus utt.) tieši no šī portāla.[37]

Lai izstrādātāji varētu veikt nepieciešamās datu transformācijas, MuleSoft piedāvā savu izveidoto programmēšanas valodu DatWeave. Šī valoda palīdz ātri un vienkārši izveidot risinājumus bieži sastopamajām datu pārveidošanas problēmām. Izmantojot šo valodu, izstrādāji var iegūtos datus pārveidot sev nepieciešamajā formātā, veikt nepieciešamās transformācijas un izmanīt datu struktūru, atbilstoši biznesa vajadzībām.[38]

4. API-VEIDOTAS SAVIENOJAMĪBAS IZSTRĀDE

Šajā nodaļā autors izstrādās divu sistēmu integrāciju izmantojot API-veidotas savienojamības pieeju (skatīt 2.4. nodaļu) un MuleSoft AnyPoint integrācijas platformu (skatīt 3. nodaļu). Par pamatu integrācijai autors izmantos izdomātu uzņēmumu "X", kuram ir nepieciešamība veikt esošo uzņēmuma sistēmu integrāciju ar jaunu sistēmu, kas palīdzētu uzlabot uzņēmuma darbību.

Izstrādes laikā autors plāno ieviest visus trīs API-veidotas savienojamības slāņus (skatīt 2.4.1. nodaļu), lai demonstrētu katra slāņa funkcionalitāti un iespējas. Integrācijā kopā plānots izveidot četrus dažādus API, kur katrs no tiem veiks konkrētu funkcionalitāti. Katram API tiks izveidota API specifikācija un izstrādāts kods izmantojot AnyPoint platform piedāvātās iespējas (skatīt 3. nodaļu).

4.1 Problēmas apraksts

Uzņēmums "X" jau vairākus gadus izmanto SAP, kā galveno uzņēmuma resursu plānošanas sistēmu. Šī sistēma atbild par dažādu uzņēmuma biznesa funkciju pārvaldību, piemēram, iepirkumu, ražošanu, materiālu pārvaldību, pārdošanu, mārketingu, finansēm un cilvēkresursu pārvaldību.[39] Lai pilnībā automatizētu un optimizētu sava uzņēmuma darbību, uzņēmums "X" ir nolēmis papildus izmantot CRM sistēmu Salesforce, kas dotu pilnīgu skatījumu uz klienta datiem un nodrošinātu labāku lietotāju pieredzi. Salesforce nodrošina klientu attiecību pārvaldību, kā arī virkni uzņēmuma lietojumprogrammu, kas ir vērstas uz klientu apkalpošanu, mārketinga automatizāciju analīzi un lietojumprogrammu izstrādi.[40]

Lai panāktu abu sistēmu plūstošu sadarbību, uzņēmums ir nolēmis veikt abu sistēmu integrāciju, veicot nepieciešamo datu sinhronizāciju starp SAP un Salesforce. Kā integrācijas rīku uzņēmums vēlas izmantot kādu no iPaaS platformām, jo tas nevēlas investēt papildu līdzekļus infrastruktūras izveidošanā un uzturēšanā.

Veicot abu sistēmu integrēšanu un izmantojot integrācijas platformu, uzņēmums plāno sasniegt sekojošo:

- Izvairīšanos no divkāršas datu ievades, kas ietaupītu laiku, un naudu.
- Izvairīšanos no datu dublēšanās un kļūdām, ko varētu izraisīt manuāla datu ievade
- Automatizētu datu sinhronizācijas procesu
- Ātrāku un precīzāku informācijas nokļūšanu vajadzīgajās biznesa jomās.

- Vienotu skatu uz visām uzņēmuma integrācijām
- Mērogojamību pēc pieprasījuma

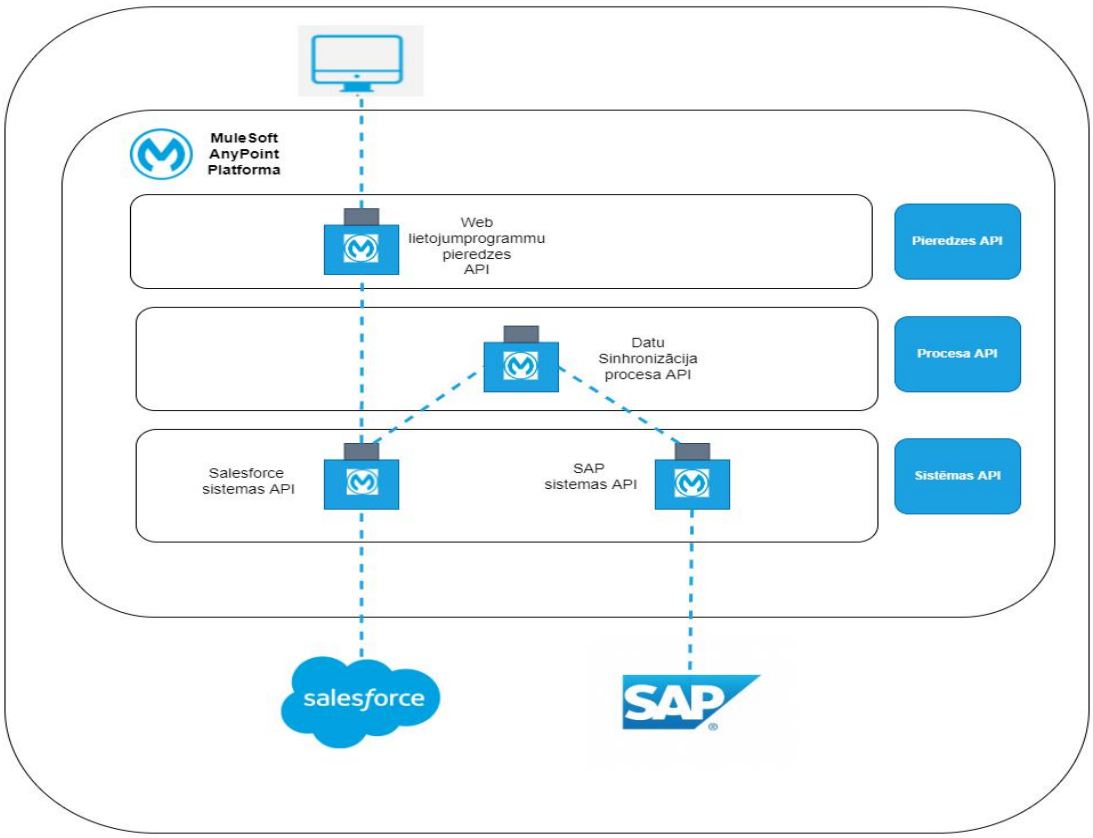
Papildus tam uzņēmumam "X" ir plāni paplašināt savu biznesa darbību nākotnē un uzņēmums jau tagad izskata iespējas kā ar dažādu tehnoloģiju palīdzību, tas varētu uzlabot un sekmēt savu darbību. Tāpēc uzņēmumam ir svarīgi, lai dati no esošajām sistēmām būtu viegli pieejami un pēc nepieciešamības integrējami citās sistēmās.

4.2 Integrācijas arhitektūra

Veidojot integrācijas risinājumu, uzņēmumam "X" tiek piedāvāts izmantot API-veidotas savienojamības pieeju, kas ļautu ātri un vienkārši izveidot nepieciešamās integrācijas. Lai izgūtu datus no esošajām sistēmām un ieviestu nepieciešamo biznesa loģiku, tiks izmantoti API (skatīt nodaļu 1.2.1.). Katrs API tiks veidots ar konkrētu mērķi un domu, lai to pēc tam varētu izmantot atkārtoti. Izveidotie API tiks ievietoti API-veidotas savienojamības ietvarā (skatīt att. 4.1), kas ļaus strukturēt izveidotos uzņēmuma resursus un izveidos elastīgu lietojumprogrammu tīklu, kurā esošos komponentus pēc tam varēs izmantot atkārtoti, papildināt un izmainīt atbilstoši biznesa vajadzībām.

Lai to visu izveidotu, pārvaldītu un uzraudzītu tiek piedāvāts izmantot MuleSoft AnyPoint platformu (skatīt nodaļu 3.), kas nodrošina iepriekš minētās iespējas.

API-veidotas savienojamības arhitektūra sastāvēs no trīs integrācijas slāņiem (skatīt nodaļu 2.4.1) un, lai veiktu uzņēmumam nepieciešamo integrāciju, tiks izveidoti četri dažādi API.



4.1 att. **Uzņēmuma "X" API-veidotas savienojamības arhitektūra**

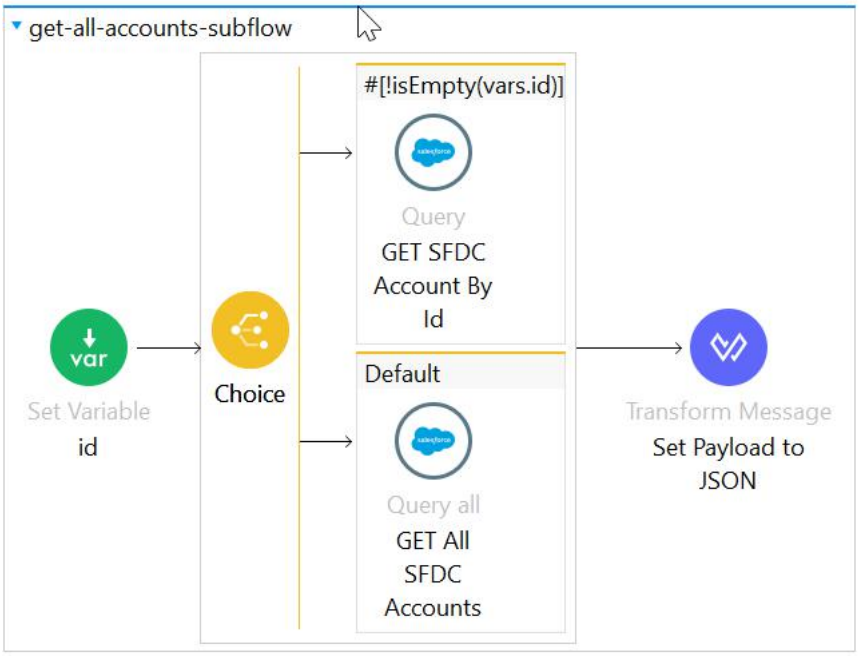
Sistēmas slānī tiks izveidoti divi vispārēji API, kas spēs izgūt, ievietot, mainīt vai dzēst nepieciešamos datus konkrētajās sistēmas. Procesa slānī tiks izveidots viens API, kas veiks konkrēto biznesa vajadzību, kas šajā gadījumā būs datu migrācija un sinhronizācija starp šim divām sistēmām. Pieredzes slānī tiks ieviest viens API, kas ērtā veidā padarīs lietotāju datus pieejamus uzņēmuma "X" tīmekļa lietojumprogrammai. Visi API tiks detalizētāk apskatīti nākamajās nodaļās.

4.2.1 Sistēmu API

Sistēmas slānī tika izveidoti Sap (lu-sl-sap) un Salesforce (lu-sl-sfdc) sistēmas API, kas ļauj darboties ar SAP un Salesforce esošajiem objektiem. Izmantojot Salesforce sistēmas API ir iespējams darboties ar diviem Salesforce objektiem Account un Contact, bet, izmantojot SAP sistēmas, API ir iespējams darboties ar BusinessPartner un BusinessPartnerAddress objektiem. Galvenā abu sistēmu API funkcionalitāte ir padarīt pamata sistēmu datus vienkārši un droši pieejamus, kā arī standartizēt to datu struktūras un formātu. .

Izveidotais Salesforce sistēmas API nodrošina sekojošus galapunktus:

- **/getAllAcounts -** Šis galapunkts atgriež visus Account objekta ierakstus (skatīt att. 4.2), ir iespējams atgriezt arī konkrētu ierakstu, ja kā vaicājuma parametrs tiek padots meklējamā ieraksta identifikācijas numurs (Id).



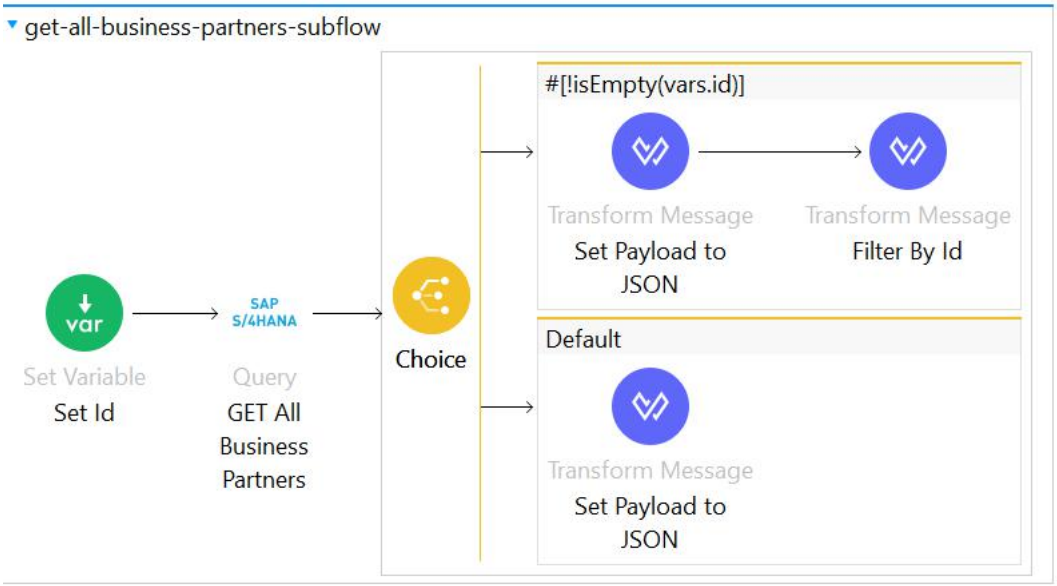
4.2 att. **lu-sl-sfdc sistēmas API get-all-accounts plūsmas koda ekrānskats**

- **/createAccount -** Šis galapunkts ļauj izveidot jaunu Account ierakstu.
- **/updateAccount/{id} -** Šis galapunkts ļauj izmainīt esošu Account ierakstu.
- **/deleteAccount/{id} -** Šis galapunkts ļauj izdzēst esošu Account ierakstu, procesā tiek izdzēsts arī atbilstošais Contact ieraksts.
- **/getAllContacts -** Šis galapunkts atgriež visus Conatact objekta ierakstus, ir iespējams atgriezt arī konkrētu ierakstu, ja kā vaicājuma parametrs tiek padots meklējamā ieraksta identifikācijas numurs (Id).
- **/createContact -** Šis galapunkts ļauj izveidot jaunu Contact ierakstu.
- **/updateContact/{id} -** Šis galapunkts ļauj izmainīt esošu Contact ierakstu.
- **/deleteContact/{id} - -** Šis galapunkts ļauj izdzēst esošu Contact ierakstu.

Pārējos Salesforce sistēmas API galapunktu plūsmu ekrānskatus var aplūkot 1. pielikumā.

Izveidotais SAP sistēmas API nodrošina sekojošus galapunktus:

- **/getAllBusinessPartners** - Šis galapunkts atgriež visus BusinessPartner objekta ierakstus (skatīt att. 4.3), ir iespējams atgriezt arī konkrētu ierakstu, ja kā vaicājuma parametrs tiek padots meklējamā ieraksta identifikācijas numurs (Id).



4.3 att. **lu-sl-sap sistēmas API get-all-business-partners plūsmas koda ekrānskats**

- **/createBusinessPartner** - Šis galapunkts ļauj izveidot jaunu BusinessPartner ierakstu.
- **/updateBusinessPartner/{id}** - Šis galapunkts ļauj izmainīt esošu BusinessPartner ierakstu.
- **/deleteBusinessPartner/{id}** - Šis galapunkts ļauj izdzēst esošu BusinessPartner ierakstu, procesā tiek izdzēsta arī atbilstošā BusinessPartnerAddress.
- **/getAllBusinessPartnerAddreses** - Šis galapunkts atgriež visus BusinessPartnerAddress objekta ierakstus, ir iespējams atgriezt arī konkrētu ierakstu, ja kā vaicājuma parametrs tiek padots meklējamā ieraksta identifikācijas numurs (Id).
- **/createBusinessPartnerAddress** - Šis galapunkts ļauj izveidot jaunu BusinessPartnerAddress ierakstu.
- **/updateBusinessPartnerAddress/{id}** - Šis galapunkts ļauj izmainīt esošu BusinessPartnerAddress ierakstu.
- **/deleteBusinessPartnerAddress/{id}** - Šis galapunkts ļauj izdzēst esošu BusinessPartnerAddress ierakstu.

Pārējos SAP sistēmas API galapunktu plūsmu ekrānskatus var aplūkot 2. pielikumā.

Abi sistēmu API tika veidoti ar mērķi, lai tos būtu iespējams izmantot atkārtoti. Visas plūsmas tika veidotas tā, lai aizmugursistēmas izmaņu gadījumā, būtu jāveic pēc iespējas mazāk izmaiņu kodā. Katra plūsma ir atdalīta viena no otras, un izmaiņas citās plūsmās vai jaunu galapunktu pievienošana neradīs izmaiņas esošajās. Šie API ir paredzēti priekš integrāciju izstrādātājiem, tie iespējo ātru izstrādi nākamajās integrācijās, jo izstrādātājiem, nebūs jātērē laiks, ieviešot šo funkcionalitāti atkārtoti. Tā vietā viņi varēs pievienot jaunu funkcionalitāti esošajos API vai izmantot esošo funkcionalitāti jaunu biznesa procesu veikšanai un jaunas lietotāju pieredzes radīšanai.

4.2.2 Procesa API

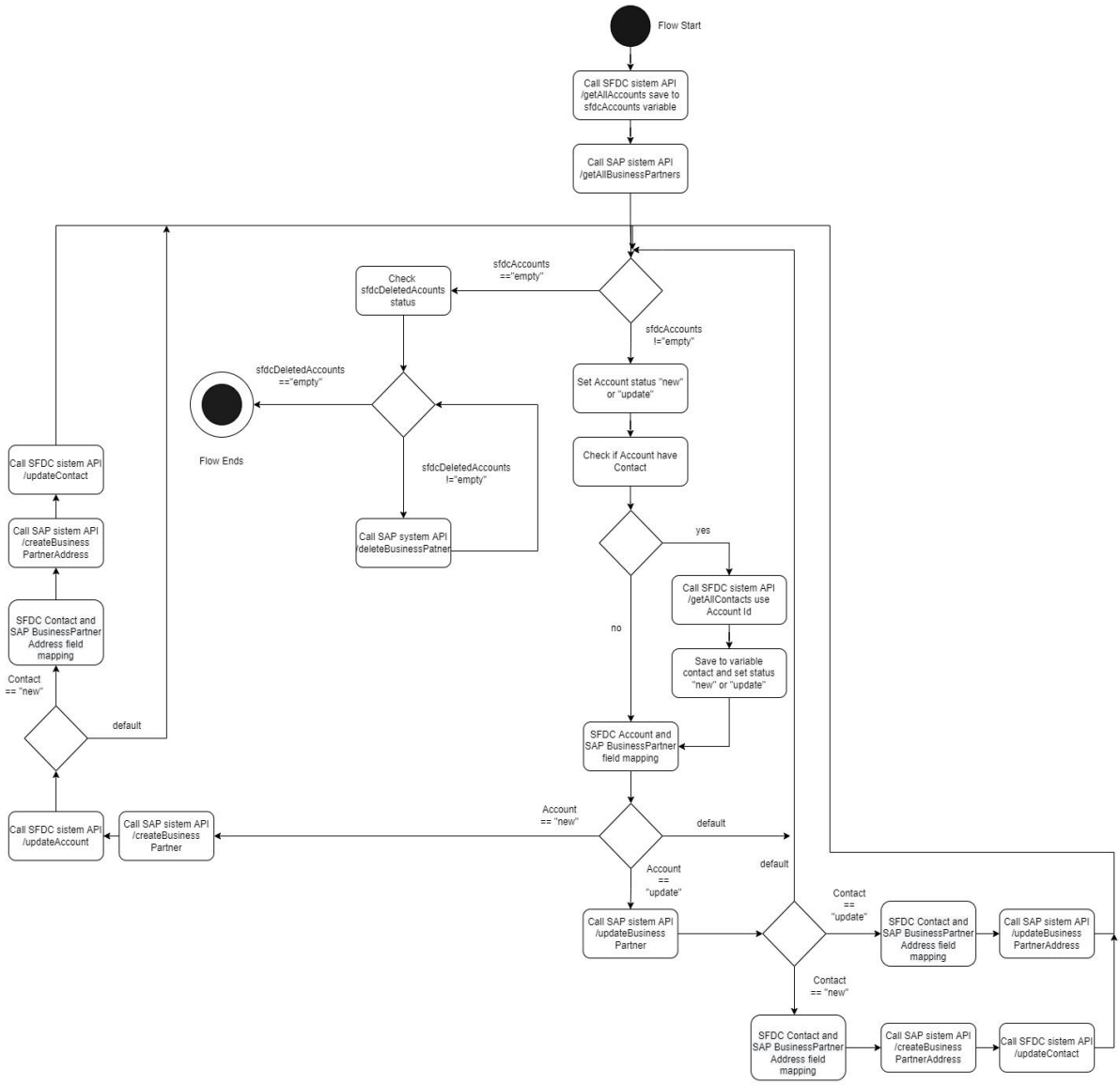
Procesa slānī tika izveidots Datu migrācijas un sinhronizācijas procesa API (lu-pl-sap-sfdc), kurā ir izveidotas divas procesa plūsmas, kuras izpilda datu migrāciju un sinhronizāciju starp SAP un Salesforce sistēmām. Plūsma sfdc-sap-flow izpilda datu sinhronizāciju no Salesforce uz SAP un sap-sfdc-flow izpilda datu migrāciju no SAP uz Salesforce.

Procesa API izmanto abu sistēmu API piedāvāto funkcionalitāti un, veicot dažādas datu transformēšanas un maršrutēšanas, funkcijas izpilda nepieciešamos biznesa procesu.

Lai izpildītu datu migrāciju lietotājiem būs iespēja mauāli izsaukt /startMigrationProcess galapunktu kas izsauks sap-sfdc-flow plūsmu un tā veiks visu datu migrēšanu no SAP uz Salesforce.

Datu sinhronizācijas procesu būs iespējams automatizēt, izmantojot MuleSoft standarta plānotāja procesoru, tas veic datu sinhronizēšanu starp abām sistēmām noteiktos laika periodos, kurus ir iespējams konfigurēt pēc nepieciešamības. Lai izstrādes periodā sinhronizācijas procesu būtu ērtāk testēt, šim API ir izveidots papildus, galapunkts /startSyncProcess, kuru izsaucot var manuāli uzsākt sinhronizācijas procesu. Tālāk var apskatīt plūsmu sfdc-sap-flow (skatīt att. 4.4) un sap-sfdc-flow (skatīt att. 4.6) aktivitāšu diagrammas un datu lauku kartēšanas tabulu (skatīt tabulu 4.1). API specifikācija ir apskatāma 3. pielikumā.

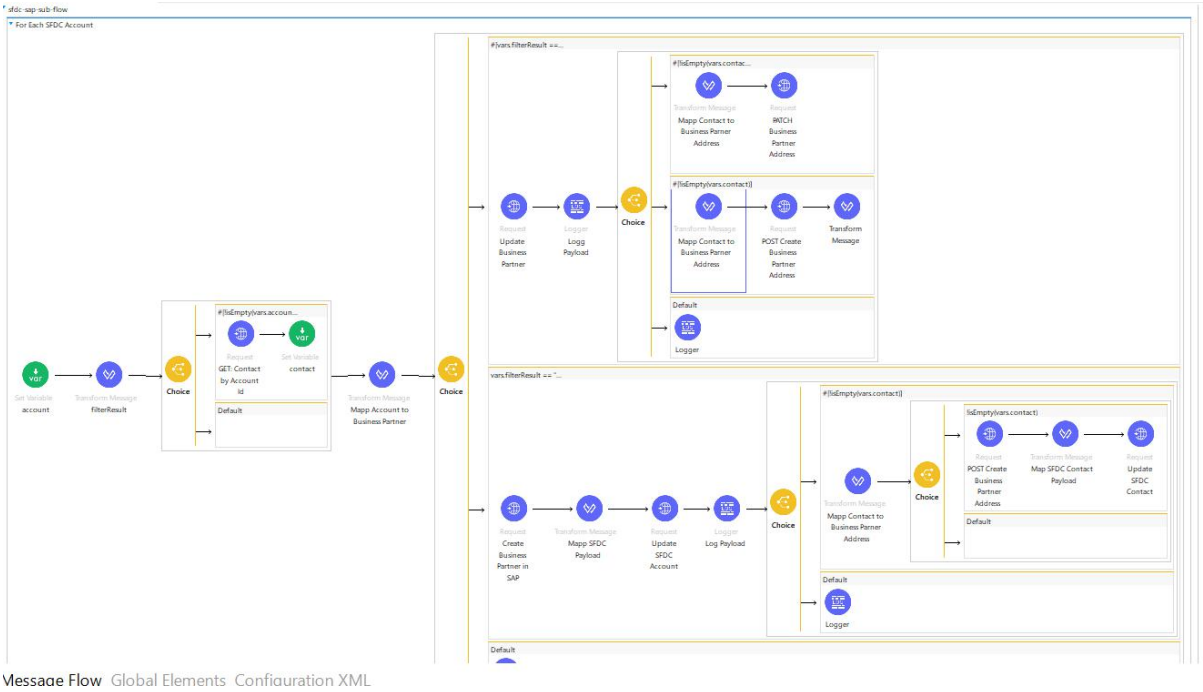
Sfdc-sap-flow aktivitāšu diagramma (skatīt att 4.4) apraksta datu sinhronizācijas procesu no Salesforce uz SAP sistēmu.



4.4 att. **lu-pl-sap-sfdc process API sfdc-sap-flow datu sinhronizācijas plūsmas aktivitāšu diagramma**

Sfdc-sap-flow koda ekrānskats (skatīt att. 4.5 un 4.6) parāda koda vizuālo reprezentāciju MuleSoft AnyPoint studio izstrādes vidē (skatīt 3.6 nodaļu).

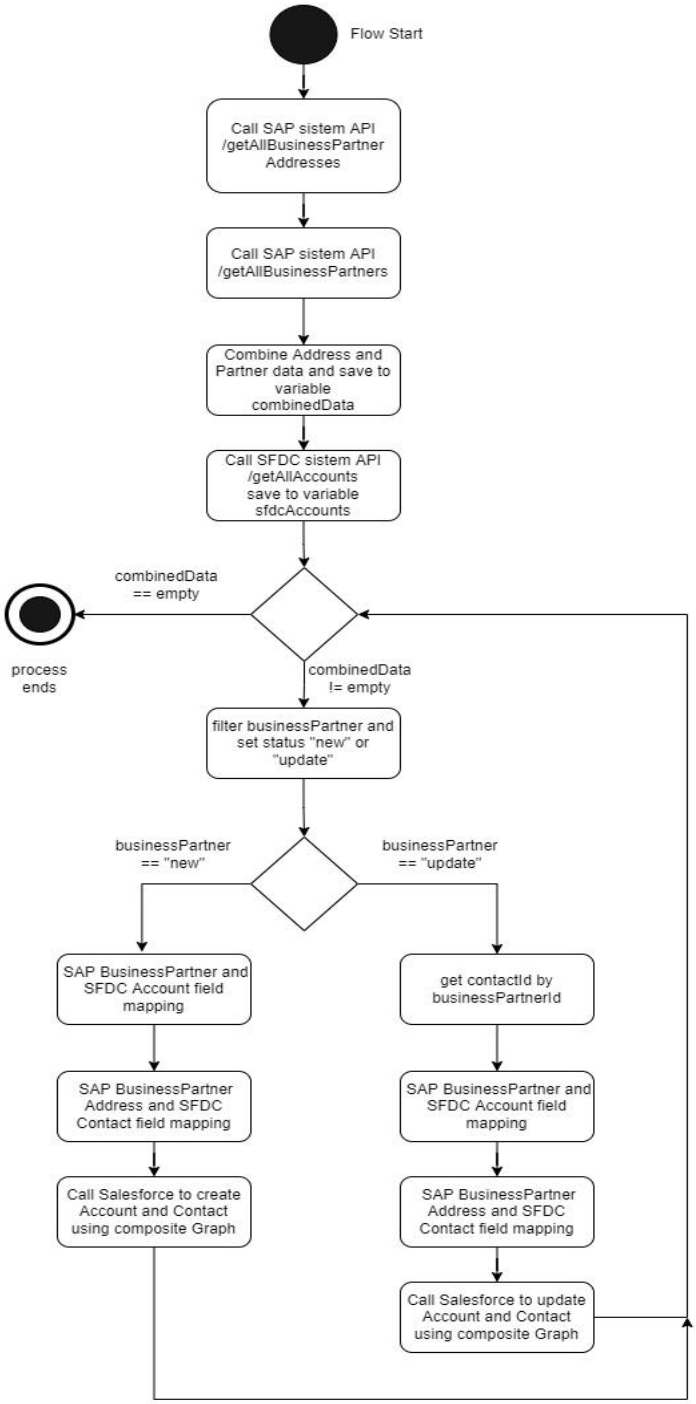
4.5 att. **lu-pl-sap-sfdc process API sfdc-sap-flow datu sinhronizācijas plūsmas un sinchronize-data-delete-subflow apakšplūsmas koda ekrānskats**



4.6 att. **lu-pl-sap-sfdc process API sfdc-sap-subflow datu sinhronizācijas apakšplūsmas koda ekrānskats**

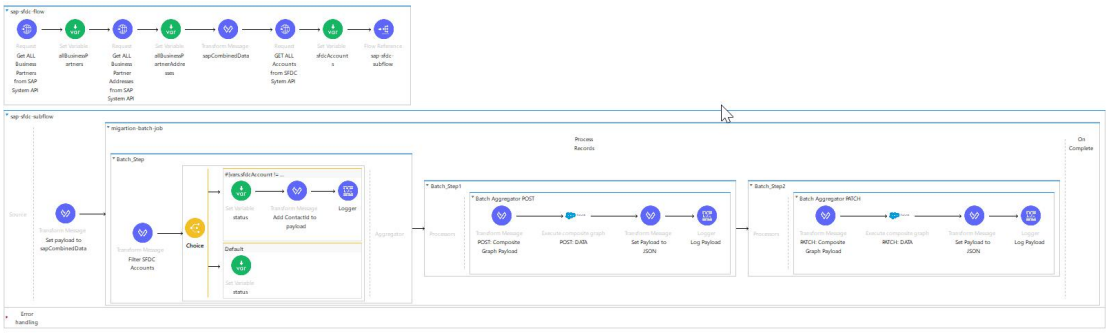
Sfdc-sap-flow plūsma ir sadalīta galvenajā plūsmā un apakšplūsmā, lai izstrādes laikā tas būt vieglāk pārskatāms. Plūsmas laikā vairākas reizes tiek izsaukti, gan SAP, gan Salesforce sistēmu API galapunkti, kas ļauj iegūt un modificēt datus no pamata sistēmām. Procesa laikā dati tiek pārveidoti, saglabāti un maršrutēti atbilstoši sfdc-sap-flow aktivitāšu diagrammas aprakstītajai loģikai (skatīt att. 4.4).

Sap-sfdc-flow aktivitāšu diagramma (skatīt att. 4.7) apraksta datu migrācijas procesu no SAP uz Salesforce sistēmu.



4.7 att. **lu-pl-sap-sfdc process API sap-sfdc-flow datu migrācijas plūsmas aktivitāšu diagramma**

Sap-sfdc-flow koda ekrānskats (skatīt att. 4.8) parāda koda vizuālo reprezentāciju MuleSoft AnyPoint studio izstrādes vidē (skatīt 3.6 nodaļu).



4.8 att. **lu-pl-sap-sfdc process API sap-sfdc-flow datu migrācijas plūsmas koda ekrānskats**

Izstrādājot sap-to-sfdc plūsmu, tika izmantots iepriekš izstrādāts Salesforce Composit Graph savienotājs, ko var atrast pie MuleSoft standartā piedāvātajiem savienotājiem Exchange portālā. Tas padarīja integrācijas loģiku mazāk apjomīgu, jo ļauj ar vienu izsaukumu ievietot vai izmainīt vairākus objektus Salesforce sistēmā.

SAP un Salesforce datu lauku kartēšanas tabulā (skatīt tabulu 4.1) tiek aprakstīti abu sistēmu datu lauki, tiek aprakstīts lauku nosaukums, datu tips, kartēšanas nosacījumi un transformācijas loģika, kas jāņem vērā, veicot datu kartēšanu no vienas sistēmas uz otru.

4.1 tabula **SAP un Salesforce datu lauku kartēšana**

SAP Objekts	**SAP lauka nosaukums**	**Datu tips**	**SFDC Objekts**	**SFDC lauka nosaukums**	**Datu tips**	**Transformācija**
BusinessPartner	BusinessPartner	String	Account	Business_Partner_id__c	Text	
		String		Name	Name	FirstName + Last Name
	Industry	String		Industry	Picklist	Default: Other

	IsSexUnknown	Boolean		Gender__c	Text	if(IsNaturalPerson == “X”){
						Case: (IsMale == true) {
						Male},
						Case: (IsFemale== true) {
						Female},

						Default:Unknown} else{Organization}
	LastName	String		Last_Name__c	Text	
	BusinessPartner IsBlocked	Boolean		Blocked__c	CheckBox	
	Supplier	String		Supplier__c	Text	
	PersonNumber	String		Identity_Number__c	Text	

	IsFemale	Boolean		Gender__c	Text	if(IsNaturalPerson == “X”){
						Case: (IsMale == true) {
						Male},
						Case: (IsFemale== true) {
						Female},
						Default:Unknown}
						else{Organization}

	IsNaturalPerson	String		Gender__c	Text	if(IsNaturalPerson == “X”){
						Case: (IsMale == true) {
						Male},
						Case: (IsFemale== true) {
						Female},
						Default:Unknown}
						else{Organization}

	FirstName	String		First_Name__c	Text	
	IsMale	Boolean		Gender__c	Text	if(IsNaturalPerson == “X”){

						Case: (IsMale == true) {
						Male},
						Case: (IsFemale== true) {
						Female},
						Default:Unknown}
						else{Organization}

BusinessPartner Address	StreetName	String	Contact	Street__c	Text	
	HouseNumber	String		House_Number__c	Text	
	BusinessPartner	String		Business_Partner_id__c	Text	
	PostalCode	String		Postal_Code__c	Text	
	CityName	String		City__c	Text	
	FullName	String		LastName	Text	
	Country	String		Country__c	Text	
	AddressID			Business_Partner_Address_id__c	Text	

Izstrādātajs procesa API tka veidots, lai tajā pēc nepieciešamības varētu pievienot jaunas plūsmas vai izmanīt esošās un tas nekādā veidā neietiekmētu citu plūsmu darbību. Abi procesi tika atdalīti viens no otra un katrs process tika sadalīts apakšplūsmās, kas ļauj vieglāk uzturēt un veikt izmaiņas atbilstošajās pūsmās. Šāda plūsmu sadalīšana apakšplūsmās dod arī iespēju izmantot, kādu daļu no ieviestās integrācijas citā biznes procesā, kas nākotnē varētu tikt pievienots šim API.

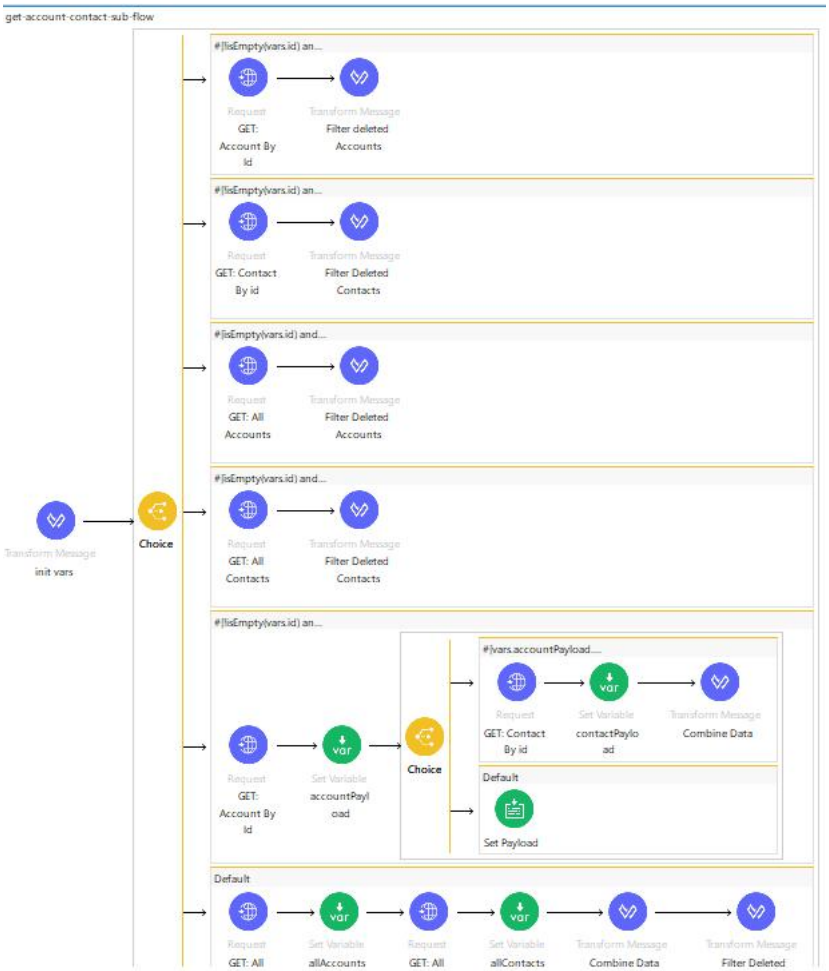
Procesa API izstrādes laikā atkārtoti tika izmantoti dažādi galapunkti no izstrādātajiem sistēmas API, kas krietni paātrināja izstrādes laiku un atviegloja integrācijas sarežģītību, jo sistēmu API piedāvāja datus vienā formātā un standartizētās datu struktūrās.

4.2.3 Pieredzes API

Pieredzes slānī tika izstrādāts Tīmekļa lietojumprogrammas pieredzes API (lu-el-webapp), kas padara iepriekš iegūtos datus pieejamus uzņēmuma "X" tīmekļa lietojumprogrammai. Tā kā uzņēmuma klienta dati tiks pārvaldīti izmantojot Salesforce sistēmu un sistēmas ir sinhronizētas pateicoties procesa API, tad pieredzes API izmanto Salesforce sistēmas API, kā galveno datu ieguves avotu. API specifikācija ir atrodama 4. pielikumā.

Uzņēmuma "X" tīmekļa lietojumprogrammas vajadzībām ir izveidoti sekojoši galapunkti:

/getAccountContact - Izsaucot šo galapunktu ir iespējams iegūt Salesforce lietotāju datus. Izsaucot galapunktu tam var padot sekojošus vaicājuma parametrus account, contact, kuriem iespējamās vērtības ir "yes" vai "no" un id, kur jāpadod attiecīgi Account vai Contact, ieraksta id lauka vērtība (skatīt att. 4.9).

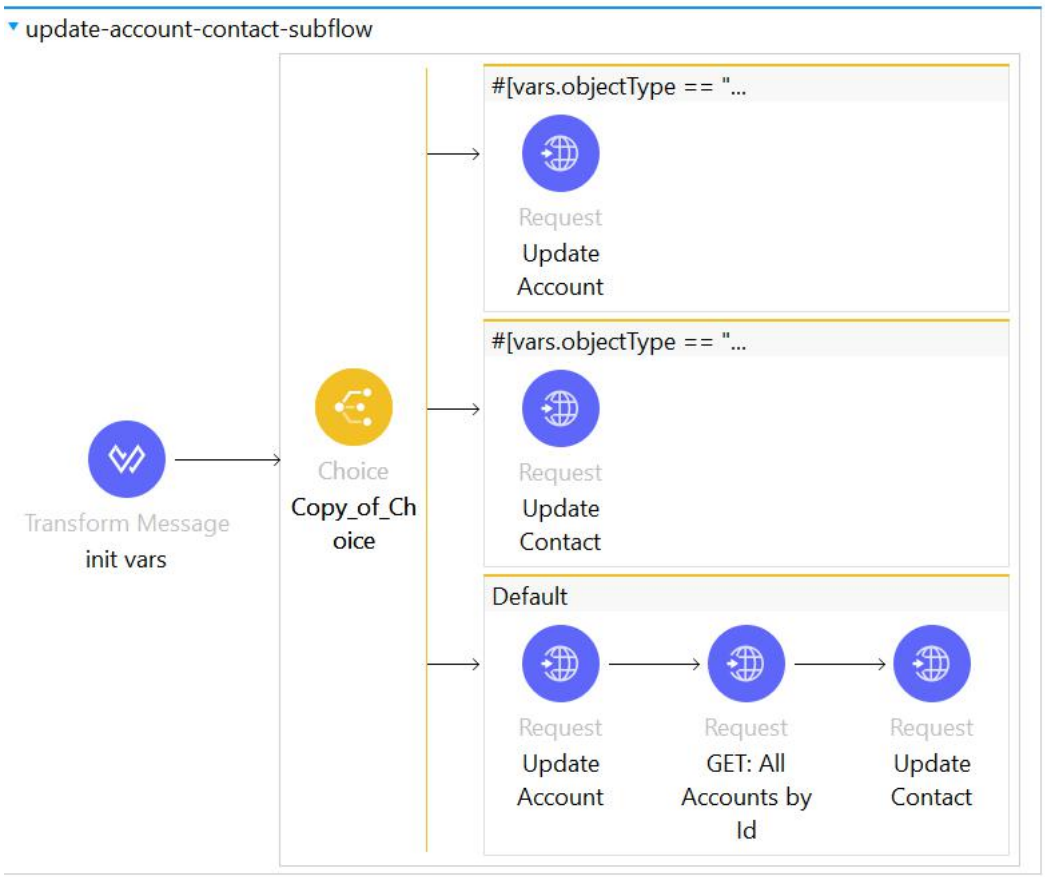


4.9 att. **lu-el-webapp pieredzes API get-account-contact plūsmas koda ekrānskats**

Vaicājuma parametru kombinācijas un to atgriežamie dati ir:

- “account” = “yes” un “contact” = “no” un “id” = AccountId - atgriezīs tikai konkrētā Account ieraksta datus.
- “account” = “no” un “contact” = “yes” un “id” = ContactId - atgriezīs tikai konkrētā Contact ieraksta datus.
- “account” = “yes” un “contact” = “no” - atgriezīs visus Account ierakstus
- “account” = “no” un “contact” = “yes” - atgriezīs visus Contact ierakstus
- “account” = “yes” un “contact” = “yes” - atgriezīs visus Acount un atbilstošā Contact ierakstus.
- “account” = “yes” un “contact” = “yes” un “id” = AccountId - atgriezīs tikai konkrētā Account un Contact datus.

/updateAccountContact - Izsaucot šo galapunktu iespējams izmainīt Salesforce lietotāju datus Account un Contact objektos. Izsaucot galapunktu, tam var padot sekojošus vaicājuma parametrus, objectType, kuram iespējamās vērtības ir “account”, “contact”, “all” kā arī accountId un contactId kuram kā vērtība jāpadod Account vai Contact ieraksta Id vērtība. Atkarībā no to kombinācijām sistēmā tiks izmainīti atšķirīgi dati (skatīt att. 4.10). Papildus vaicājumu parametriem ir jāpadod datu objekts, kurš satur izmaināmos datu laukus un to vērtības.

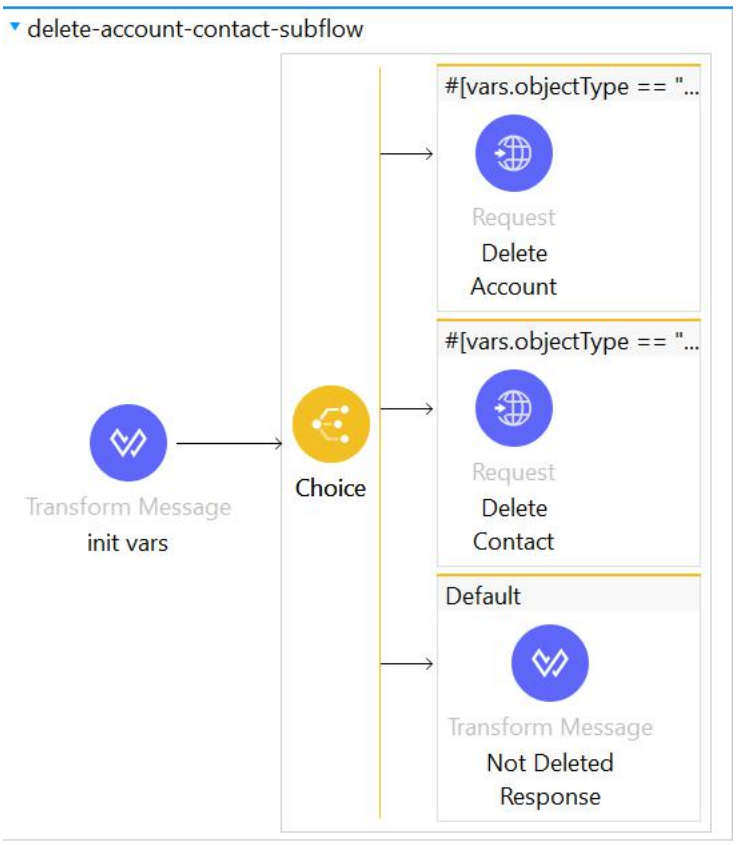


4.10 att. **lu-el-webapp pieredzes API update-account-contact plūsmas koda ekrānšāviņš**

Vaicājuma parametru kombinācijas un to atgriežamie dati ir:

- "objectType" = "account" un "accountId" = AccountId - Izmainīs tikai konkrētā Account ieraksta datus.
- "objectType" = "contact" un "contactId" = ContactId - Izmainīs tikai konkrētā Contact ieraksta datus.
- "objectType" = "all" un "accountId" = AccountId - Izmainīs konkrētā Account un Contact ieraksta datus.

/deleteAccountContact - Izsaucot šo galapunktu iespējams izdzēst Salesforce lietotāju datus Account un Contact objektos. Izsaucot galapunktu, tam var padot sekojošus vaicājuma parametrus, objectType, kuram iespējamās vērtības ir "account", "contact", kā arī accountId un contactId, kuram kā vērtība jāpadod Account vai Contact ieraksta Id vērtība. Atkarībā no to kombinācijām sistēmā tiks izdzēsti atšķirīgi dati (skatīt att. 4.11).

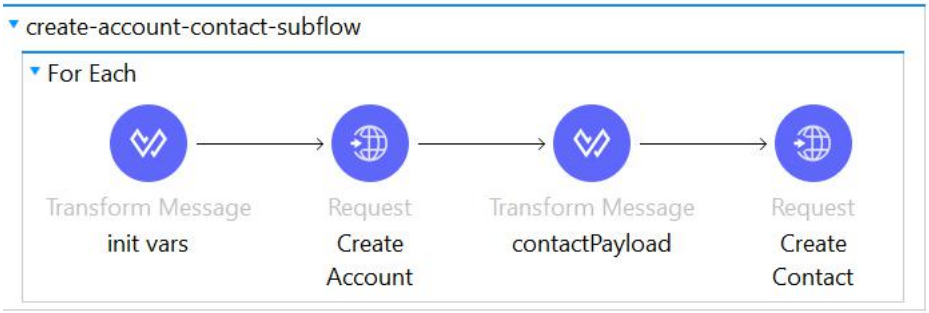


4.11 att. **lu-el-webapp pieredzes API delete-account-contact plūsmas koda ekrānskats**

Vaicājuma parametru kombinācijas un to atgriežamie dati ir:

- "objectType" = "account" un "accountId" = AccountId - Izdzēsīs konkrētā Account un tā Contact ieraksta datus.
- "objectType" = "contact" un "contactId" = ContactId - Izdzēsīs tikai konkrētā Contact ieraksta datus.

/createAccountContact - Izsaucot šo galapunktu iespējams izveidot jaunus Salesforce lietotāju ierakstus Account un Contact objektos. Izsaucot galapunktu, tam jāpadod saraksts ar vienu vai vairākiem datu objektiem, kuri satur Account un Contact datu laukus ar to izveidojamajām vērtībām. (skatīt att. 4.12).



4.12 att. **lu-el-webapp pieredzes API create-account-contact plūsmas koda ekrānskats**

Pieredzes API līdzīgi kā sistēmu un procesa API tika veidots, nodalot dažādas funkcionalitātes atšķirīgās plūsmās, lai izmaiņas vienā plūsmā neietekmētu citas.

Atšķirībā no sistēmu API, kur visas darbības bija vienkāršas un praktiski netika pielietota nekāda datu agregācija, transformācija vai maršrutēšana, pieredzes API ir izveidots tieši pretēji. Pieredzes API notiek datu, kas iegūti no zemāk esošajiem procesa vai sistēmu API, pārveidošana un pasniegšana tādā veidā, lai tos ērti varētu izmantot konkrētā patērētāju grupa, kas šajā gadījumā ir tīmekļa lietojumprogramma.

4.2.5 Izmantotās sistēmas

Salesforce:

Salesforce piedāvā izstrādes vidi, ar ierobežotiem resursiem, kas ļauj izmēģināt un testēt dažādas Salesforce pieejamās funkcijas bezmaksas. Lai varētu piekļūt šai videi, lietotājam ir jāveic reģistrēšanās un šim kontam nav beigu termiņa.

AnyPoint platform:

MuleSoft piedāvā lietotājiem AnyPoint platform pagaidu konta izveidošanu, kas ļauj izmantot platformas funkcionalitāti, ar ierobežotu resursu pieejamību. Pagaidu konts ir derīgs 30 dienas no izveidošanas brīža.

SAP:

Kā SAP sistēmas alternatīva tika izmantots SAP piedāvātais Odata Mock server, kas atveido Business Partner API no SAP S/4HANA mākoņservisa, ar ierobežotu funkcionalitāti. Serveris tika uzstādīts hostēšanas platformā Heroku, kas ir PaaS pakalpojums un piedāvā lietotājiem izvietot, pārvaldīt un mērogot dažādas lietojumprogrammas.

4.2.6 Konfigurāciju pārvaldība

Projekta konfigurācijas pārvaldībai tika izmantota Git versiju kontroles sistēma. Visu API repozitoriji tiek glabāti GitHub sistēmā. Visos API darbs tika veikts uz „master” zara, jo pie tiem strādāja tikai viens cilvēks, un esošā API versija ir arī tā pēdējā versija. Detalizētu infomāciju par katra API repozitorijiem var apskatīt 5. pielikumā.

4.2.7 Testēšana

Izstrādājot API tie tika testēti, izmantojot API testēšanas rīku Postman. Postman darbojas, kā API klients un ļauj tā lietotājiem izveidot dažādus HTTP/S pieprasījumus (GET, POST, PATCH, PUT, DELETE u.c.). Tas atvieglo API testēšanu, jo katram API ir iespējams izveidot savu Postman kolekciju, kas sastāv no galapunktu pieprasījumu lapām, kurās var konfigurēt nepieciešamos datu objektus, vaicājumu parametrus un daudz ko citu.

Katram API tika izveidota sava Postman kolekcija, kas satur HTTPS pieprasījumus un to datus, kā arī vaicājuma parametrus uz konkrētā API galapunktiem. Detalizētu informāciju par kolekcijām un viena testēšanas piemēra aprakstu var skatīt 6. pielikumā.

4.2.8 Kopsavilkums

Visi izveidotie API ir izvietoti AnyPoint platformā, izmantojot CloudHub izvietojumus (skatīt nodaļu 3.5), kas nodrošina to augstu pieejamību, mērogojamību pēc pieprasījuma un avārijas atkopšanās iespējas. Tie ir standartizēti un darbojas, izmantojot HTTPS protokolu, un atgriež datus JSON formātā.

Visi API ir aizsargāti ar Pamata Autentifikācijas politiku, kas ir iespējota, izmantojot API Manager rīku (skatīt nodaļu 3.4), lai izsauktu kādu no API galapunktiem, nepieciešams lietotājvārds un parole, tas nodrošina visu API drošību.

API specifikācijas ir publicētas organizācijas privātajā Exchange portālā (skatīt nodaļu 3.3) un tās ir pieejamas visiem organizācijā esošajiem lietotājiem, kas ļauj visus izveidotos API, to specifikācijas un fragmentus, izmantot atkārtoti. Izstrādātās API specifikācijas organizācijas lietotāji var apskatīt Design Center (skatīt nodaļu 3.2), kā arī lejupielādēt to kodu, tieši no Exchange portāla. Exchange portālā ir pieejama visu izveidoto API dokumentācija un ir iespējams arī notestēt visus izveidotos API un redzēt to funkcionalitāti, pirms to ieviešanas integrācijā.

Izmantojot API-veidotas savienojamības pieeju un MuleSoft piedāvāto integrāciju platformu uzņēmums “X” ir sasniedzis plānotos ieguvumus (skatīt nodaļu 4.1). Papildus tam, izveidotie API ir sākuši veidot uzņēmuma lietojumprogrammu tīklu un jau uzsākot nākamo integrāciju projektu ir potenciāli iespējams izmantot ieviesto API funkcinalitāti tajā. Tas ļautu nākamo projektu ieviest daudz isākā laikā, samazinātu tā izmaksas un atvieglotu tā uzturēšanu pēc ieviešanas.

REZULTĀTI

Bakalaura darba ietvaros autors ir izpētījis un apkopojis informāciju par sistēmu integrāciju, tās izplatītākajiem lietošanas piemēriem un savienošanas veidiem (skatīt nodaļu 1.) kā arī sistēmu integrācijas arhitektūras modeļiem (2. nodaļa). Par sistēmu integrācijas modeļiem iegūtā informācija ir izmantota, lai analizētu, katra sistēmu integrācijas arhitektūras modeļa priekšrocības un trūkumus. (skatīt nodaļu 2.).

Lai varētu ieviest API-veidotas savienojamības pieeju, autors papildus izpēta, analizē un apkopo informāciju par MuleSoft integrāciju platformas piedāvātājām iespējām (skatīt nodaļu 3.). Tiek apskatīti galvenie platformas rīki, kas palīdzēs praktiski ieviest API-veidotas savienojamības pieeju, izmantojot platformas piedāvātas iespējas.

Bakalaura darba ceturtajā nodaļā ir izstrādāta API-veidotas savienojamības arhitektūra, kas sastāv no trīs dažādiem slāņiem un četriem dažādiem API. Visiem API ir izveidotas API specifikācijas, kuras var apskatīt pielikumā, kā arī tiek izstrādāts kods, kurš ir pieejams GitHub sistēmā. Katrs API tiek detalizēti aprakstīts, tiek paskaidrota API funkcionalitāte un loma kopējā API-veidotas savienojamības arhitektūrā. Beigās ir veikts ieviestās integrācijas kopsavilkums (skatīt nodaļu 4.2.8).

SECINĀJUMI

Bakalaura darbā izvirzītais mērķis ir sasniegts. Ir izpētīta API-veidotas savienojamības pieeja un apkopoti kādi ir tās izmantošanas ieguvumi sistēmu integrēšanā.

Darbā plānotie uzdevumi ir izpildīti. Tika izpētīta literatūra par sistēmu integrāciju un izplatītākajiem sistēmu integrācijas arhitektūras modeļiem. Arhitektūras modeļi tika analizēti, un katram no tiem tika noteiktas priekšrocības un trūkumi. Tika izpētītas MuleSoft piedāvātās integrāciju platformas iespējas un rīki, kas palīdz ieviest API-veidotas savienojamības pieeju. Tika izveidota API-veidotas savienojamības arhitektūra un, izmantojot šo pieeju ieviesta integrācija starp SAP un Salesforce sistēmām.

Apkopojot informāciju, kas iegūta no literatūras avotiem un veicot sistēmu integrēšanu izmantojot API-veidotas savienojamības pieeju, tika veikti sekojoši secinājumi:

1. Sistēmu integrācija ir aktuāla jebkura lieluma uzņēmumā, jo arvien vairāk uzņēmumi izmanto dažādas sistēmas un lietojumprogrammas, kas palīdz īstenot dažādus biznesa procesus uzņēmumos.

2. Uzņēmumiem ir svarīgi pēc iespējas mazāk resursus novirzīt integrāciju veidošanā un uzturēšanā, bet realitātē notiek tieši pretējais, un integrāciju veidošanai, uzturēšanai, testēšanai un dokumentācijai tiek tērēti ievērojami uzņēmumu resursi.

3. Integrācijas problēmas galvenokārt rada nepareiza arhitektūras modeļa un integrācijas rīku izvēle, kas nespēj tik līdzi uzņēmuma izaugsmei un straujajām biznesa prasību izmaiņām.

4. Arhitektūras modeļi un integrācijas rīki, kas ir izmantoti līdz šim, nav spējuši apmierināt uzņēmumu vajadzības, jo tie ir pārāk monolīti, neelastīgi, grūti uzturami, sagādā lielas ieviešanas un uzturēšanas izmaksas vai arī kalpo kā vājais punkts visai integrācijas infrastruktūrai.

5. API-veidota savienojamība balstās uz SOA izstrādes principiem, kur integrācijas arhitektūras tiek sadalīta atsevišķi atdalītos un atkārtojami izmantojamos komponentos un pēc tam piedāvāta izmantojot API.

6. Lai veiksmīgi ieviestu API-veidotas savienojamības pieeju ir nepieciešami rīki, kas nodrošina API dzīvescikla pārvaldības iespējas, sākot no tā izveides līdz pat izbeigšanai.

7. Katrs API tiek veidots ar noteiktu funkcionalitāti un pēc tam, atbilstoši funkcionalitātei, tiek ievietots API-veidotas savienojamības ietvarā, vienā no trīs dažādiem līmeņiem.

8. Atšķirīgi līmeņi ļauj katrā no tiem ieviest atšķirīgas pārvaldības un kontroles metodes, kas padara integrācijas elastīgas un vieglāk pielāgojamas.

9. Izveidotie API veido lietojumprogrammu tīklu, kas ļauj atkārtoti izmantot izstrādātos API un to komponentus, iespējo pašapkalpošanās iespējas un decentralizētu piekļuvi datiem un pakalpojumiem, vienlaikus neapdraudot to pārvaldību.

10. Lietojumprogrammu tīkls, atkārtoti izmantojot izveidotos komponentus, dod iespēju, katru nākamo integrāciju projektu pabeigt ātrāk, samazināt tā izmaksas un samazina resursu nepieciešamību tā uzturēšanā pēc ieviešanas.

11. API-veidotas savienojamības pieeja ir atkarīga no izveidoto komponentu atkārtotas izmantošanas, ja tas netiek panākts, tā var kļūt grūti uzturama un radīt papildu izmaksas.

IZMANTOTĀ LITERATŪRA

1. MuleSoft. 2022 Connectivity benchmark report [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://resources.mulesoft.com/ty-report-connectivity-benchmark.html

2. ZapUp. What are the Benefits of System Integration [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://zapup.com/what-are-the-benefits-of-system-integration

3. MuleSoft. Legacy System Integration via Mule ESB [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/resources/esb/legacy-system-integration#:~:text=Legacy%20system%20integration%20tackles%20the,in%20order%20to%20drive%20business.

4. MuleSoft. Understanding enterprise application integration - The benefits of ESB for EAI [tiešsaiste]. [atsauce 17.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/resources/esb/enterprise-application-integration-eai-and-esb

5. IBM. What is B2B integration[tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://www.ibm.com/topics/b2b-integration

6. IBM. Aplication Programming Interface [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://www.ibm.com/cloud/learn/api

7. MuleSoft. What is an API [tiešsaiste]. [atsauce 17.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/resources/api/what-is-an-api

8. IBM. Middleware[tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://www.ibm.com/cloud/learn/middleware

9. Mparticle. What is a webhook [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://www.mparticle.com/blog/apis-vs-webhooks

10. IBM. What is electronic data interchange (EDI) [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://www.ibm.com/topics/edi-electronic-data-interchange

11. Techopedia. System Integration [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://www.techopedia.com/definition/9614/system-integration-si

12. MuleSoft. System integration made easy [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/resources/esb/system-integration-esb

13. Techopedia. Application Programming Interface [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://www.techopedia.com/definition/9614/system-integration-si

14. Agilitycms. API vs Webhooks: What's the difference [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://agilitycms.com/resources/posts/-api-vs-webhooks-what-s-the-difference

15. Oracle. Key Integration Concepts [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois04.htm

16. MuleSoft. Eliminating point-to-point integration pain with Mule ESB [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/resources/esb/eliminating-point-point-integration-pain-mule-esb-use-cases

17. IBM. ESB (Enterprise Service Bus) [tiešsaiste]. [atsauce 09.04.2022.]. Pieejams Internetā: https://www.ibm.com/in-en/cloud/learn/esb

18. MuleSoft. Whitepaper API-led connectivity [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/lp/whitepaper/api/api-led-connectivity

19. Dzone. MuleSoft API Led Connectivity Architectural and Design Patterns[tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://dzone.com/articles/mulesoft-api-led-connectivity-architectural-and-de

20. Dzone. What is MuleSoft and Anypoint Platform Capabilities and Strengths [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://dzone.com/articles/what-is-mulesoft-and-anypoint-platform-capabilitie

21. MuleSoft. MuleSoft Positioned as a Leader Again in the Gartner® Magic Quadrant™ for Enterprise Integration Platform as a Service and the Magic Quadrant for Full Life Cycle API Management [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://www.mulesoft.com/press-center/2021-gartner-mq-leader-ipaas-lifecycle-api-management

22. Crozdesk. What Is MuleSoft AnyPoint? [tiešsaiste]. [atsauce 06.04.2022.]. Pieejams Internetā: https://crozdesk.com/software/mulesoft-anypoint

23. MuleSoft. What is API portal [tiešsaiste]. [atsauce 09.05.2022.]. Pieejams Internetā: https://www.mulesoft.com/resources/api/what-is-an-api-portal

24. MuleSoft. AnyPoint Runtime Manager [tiešsaiste]. [atsauce 09.05.2022.]. Pieejams Internetā: https://docs.mulesoft.com/runtime-manager/

25. MuleSoft. Deployment Options [tiešsaiste]. [atsauce 09.05.2022.]. Pieejams Internetā: https://docs.mulesoft.com/runtime-manager/deployment-strategies

26. MuleSoft. About AnyPoint Platform PCE [tiešsaiste]. [atsauce 09.05.2022.]. Pieejams Internetā: https://docs.mulesoft.com/private-cloud/3.0/

27. MuleSoft. AnyPoint API Manager [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/api-manager/2.x/latest-overview-concept

28. MuleSoft. API Groups [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/api-manager/2.x/api-groups-landing-page

29. MuleSoft. API Versions and Instances [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/api-manager/2.x/manage-versions-instances-concept

30. MuleSoft. Policies [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/api-manager/2.x/policies-landing-page

31. MuleSoft. Reviewing SLA Tiers Concepts [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/api-manager/2.x/defining-sla-tiers

32. MuleSoft. API alerts [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/api-manager/2.x/using-api-alerts

33. MuleSoft. API Designer [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/design-center/design-create-publish-api-specs

34. MuleSoft. About Design Center [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/design-center/

35. MuleSoft. Design API Specification [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/general/api-led-design

36. MuleSoft. Flow Designer [tiešsaiste]. [atsauce 09.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/design-center/about-designing-a-mule-application

37. MuleSoft. AnyPoint Studio [tiešsaiste]. [atsauce 17.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/studio/7.11/

38. MuleSoft. DataWeave language [tiešsaiste]. [atsauce 17.01.2022.]. Pieejams Internetā: https://docs.mulesoft.com/dataweave/2.4/

39. SAP. What is SAP [tiešsaiste]. [atsauce 17.01.2022.]. Pieejams Internetā: https://www.sap.com/latvia/about/company/what-is-sap.html

40. Salesforce. What is Salesforce [tiešsaiste]. [atsauce 17.01.2022.]. Pieejams Internetā: https://www.salesforce.com/products/what-is-salesforce/

41. Salesforce. What is Digital Transformation [tiešsaiste]. [atsauce 17.01.2022.]. Pieejams Internetā: https://www.salesforce.com/products/platform/what-is-digital-transformation/

42. Reddy Martin. API Design for C++, Elsevier Science & Technology, 2011. [tiešsaiste]. [atsauce 14.05.2022.]. Pieejams Internetā: http://ebookcentral.proquest.com/lib/lulv/detail.action?docID=667720.

43. Watt Dougal. E-Business Implementation, Taylor & Francis Group, 2003. [tiešsaiste]. [atsauce 14.05.2022.]. Pieejams Internetā: http://ebookcentral.proquest.com/lib/lulv/detail.action?docID=296798.

44. Puder Arno. Distributed Systems Architecture : A Middleware Approach, Elsevier Science & Technology, 2005. [tiešsaiste]. [atsauce 14.05.2022.] http://ebookcentral.proquest.com/lib/lulv/detail.action?docID=269895.

45. IBM, Redbooks. Implementing EDI Solutions, I B M, 2003. [tiešsaiste]. [atsauce 14.05.2022.] http://ebookcentral.proquest.com/lib/lulv/detail.action?docID=3306696.

46. IBM, Redbooks. Patterns : Implementing an SOA using an Enterprise Service Bus, I B M, 2004.[tiešsaiste]. [atsauce 14.05.2022.]

http://ebookcentral.proquest.com/lib/lulv/detail.action?docID=3306519.

47. MuleSoft. ESB or not to ESB revisited Part 2 [tiešsaiste]. [atsauce 17.05.2022.]. Pieejams Internetā: https://blogs.mulesoft.com/dev-guides/how-to-tutorials/esb-or-not-to-esb-revisited-part-2/

48. S-benett.How is SOI Different From Traditional Integration [tiešsaiste]. [atsauce 17.05.2022.]. Pieejams Internetā:https://s-bennett.com/2008/05/20/how-is-soi-different-from-traditional-integration/

49. Informatica.iPaaS Magic Quadrant [tiešsaiste]. [atsauce 17.05.2022.]. Pieejams https://www.informatica.com/ipaas-magic-quadrant.html

PIELIKUMI

1. Pielikums Salesforce Sistēmas API

1.1 API Specifikācija

Lai specifikācija neaizņemtu pārāk daudz vietas, no tās tika izņemtas pieprasijuma un atbildes datu struktūras, pilnīgas specifikācijas var apskatīt AnyPoint platformas rīkos Desing Center vai Exchange. Kā arī pilna specifikācija tika pievienota 3. pielikumā Sinhnronizācijas un Migrācijas procesa API.

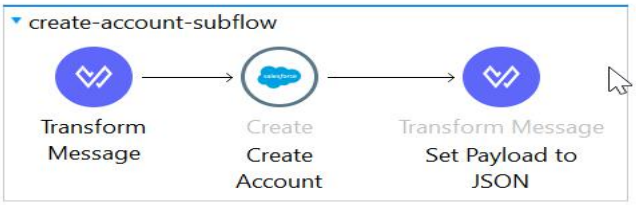
```
#%RAML 1.0
title: lu-sl-sfdc
description: System API for Salesforce
mediaType: [application/json]
protocols: [HTTPS]
version: v1
securitySchemes:
  ClientId_Enforcement_policy: !include securitySchemas/clientSecretEnforcement.raml
securedBy: ClientId  Enforcement  policy
/getAllAccounts:
  get:
    displayName: retrieve Accounts
    queryParameters:
      id:
        description: Account Business Partner Id
        type: string
        required: false
    responses:
      200:
        body:
          application/json:
/getAllContacts:
  get:
    displayName: Retrieve Contacts
    queryParameters:
      id:
        description: Contact Id
        type: string
        required: false
    responses:
      200:
        body:
          application/json:
/createAccount:
  post:
    displayName: Cretae Account
    responses:
      200:
        body:
          application/json:
/updateAccount:
  /{id}:
    patch:
      displayName: Update Account
      responses:
        200:
          body:
            application/json:
/deleteAccount:
  /{id}:
    delete:
      displayName: Delete Account
      responses:
        200:
          body:
            application/json:
/createContact:
  post:
    displayName: Cretate Account Conatact
    responses:
      200:
        body:
          application/json:
/updateContact:
```

```
  /{id}:
    patch:
      displayName: Update Contact
      responses:
        200:
          body:
            application/json:
/deleteContact:
  /{id}:
    delete:
      displayName: Delete Conatact
      responses:
        200:
          body:
            application/json:
```

1.2 Plūsmu ekrānskati

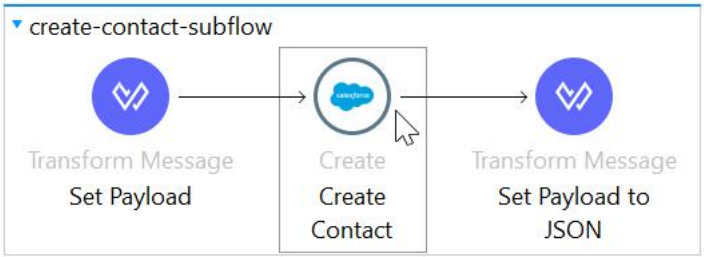
Visu Plūsmu aprakstus skatīt 4.2.1 nodaļā.

/createAccount:



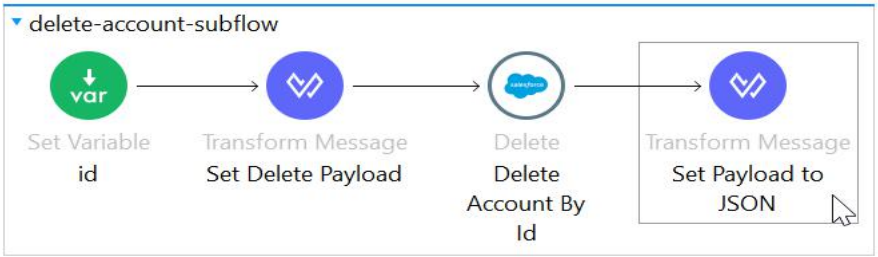
1.1 **lu-sl-sfdc sistēmas create-accounts plūsma**

/createContact:



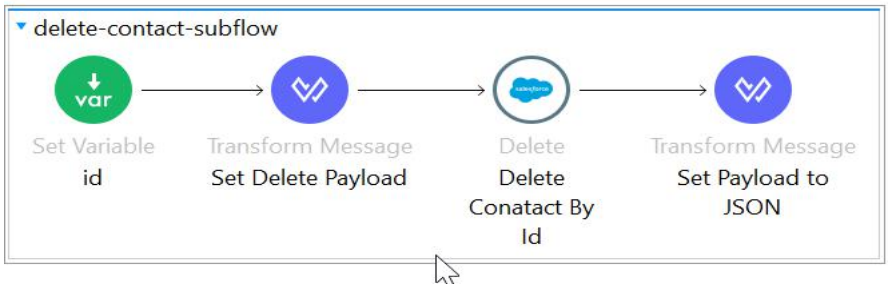
1.2 **lu-sl-sfdc sistēmas API create-contact plūsma**

/deleteAccount{id}:



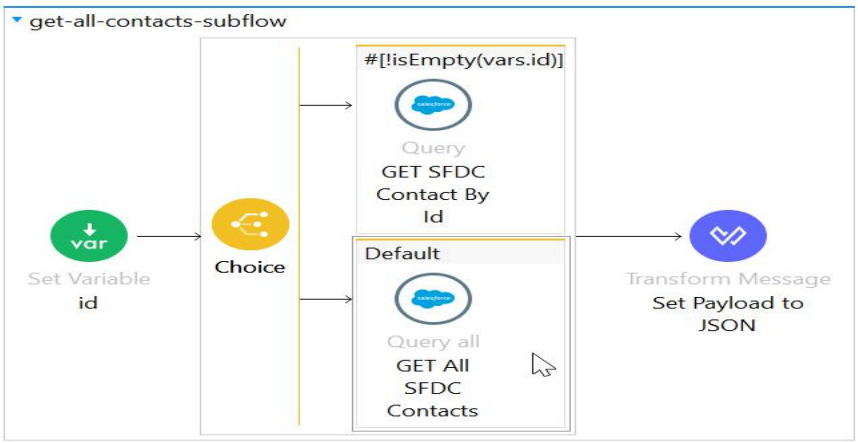
1.3 **lu-sl-sfdc sistēmas API delete-account plūsma**

/deleteContact/{id}:



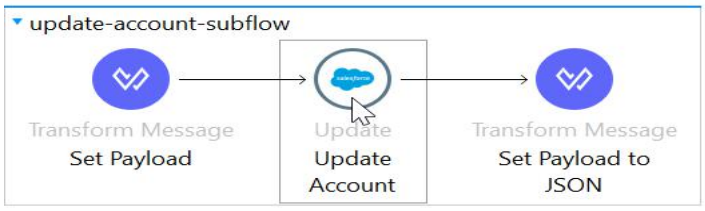
1.4 **lu-sl-sfdc sistēmas API delete-contact plūsma**

/getAllContacts:



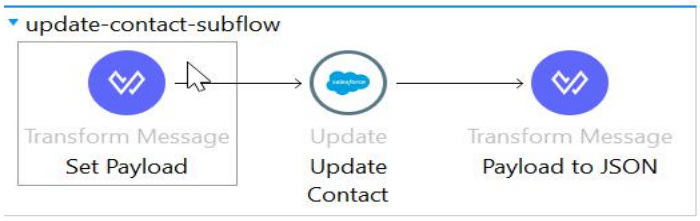
1.5 **lu-sl-sfdc sistēmas API get-all-contacts plūsma**

/updateAccount/{id}:



1.6 **lu-sl-sfdc sistēmas API update-account plūsma**

/updateContact/{id}:



1.7 **lu-sl-sfdc sistēmas API update-contact plūsma**

2. Pielikums SAP Sistēmas API

2.1 API Specifikācija

Pilnu specifikāciju skatīt AnyPoint platformas rīkos Design Center vai Exchange.

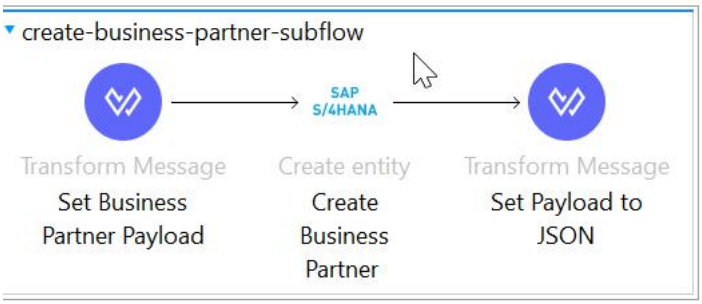
```
#%RAML 1.0
title: lu-sl-sap
description: System API for SAP
mediaType: [application/json]
protocols: [HTTPS]
version: v1
securitySchemes:
  ClientId_Enforcement_policy: !include securitySchemas/clientSecretEnforcement.raml
securedBy: ClientId_Enforcement_policy
/getAllBusinessPartners:
  get:
    displayName: retrieve Business Partners
    queryParameters:
      id:
        description: Business partner Id
        type: string
        required: false
    responses:
      200:
        body:
          application/json:
/getAllBusinessPartnerAddresses:
  get:
    displayName: Retrieve Business Partner Addresses
    queryParameters:
      id:
        description: Business partner Id
        type: string
        required: false
    responses:
      200:
        body:
          application/json:
/createBusinessPartner:
  post:
    displayName: Cretae Business Partner
    responses:
      200:
        body:
          application/json:
/updateBusinessPartner:
  /{id}:
    patch:
      displayName: Update Business Partner
      responses:
        200:
          body:
            application/json:
/deleteBusinessPartner:
  /{id}:
    delete:
      displayName: Delete Business Partner
      responses:
        200:
          body:
            application/json:
/createBusinessPartnerAddress:
  post:
    displayName: Cretate Business Partner Address
    responses:
      200:
        body:
          application/json:
/updateBusinessPartnerAddress:
  /{id}:
    patch:
      displayName: Update Business Partner Address
      responses:
        200:
          body:
            application/json:
/deleteBusinessPartnerAddress:
  /{id}:
    delete:
      displayName: Delete Business Partner Address
      queryParameters:
        addressId:
```

```
    description: Business partner Id
    type: string
    required: true
responses:
  200:
    body:
      application/json:
```

2.2 Plūsmu ekrānskati

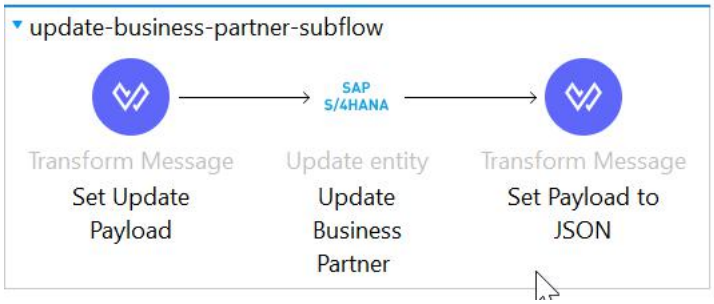
Visu Plūsmu aprakstus skatīt 4.2.1 nodaļā.

/createBusinessPartner:



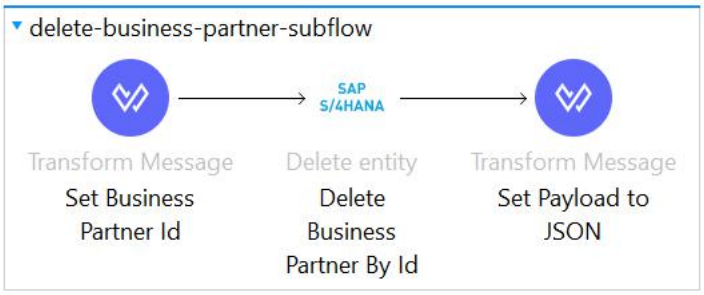
2.1 **lu-sl-sap sistēmas API create-business-partner plūsma**

/updateBusinessPartner/{id}:



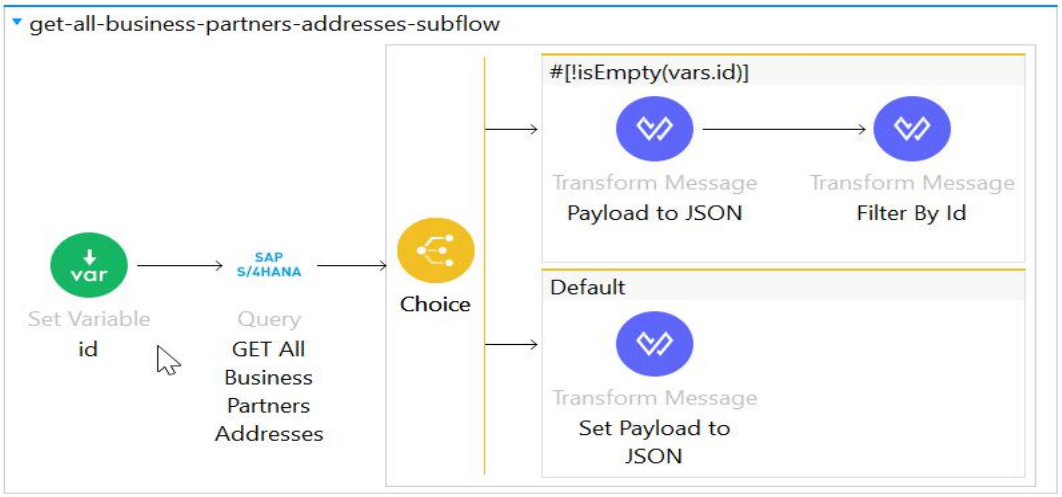
2.2 **lu-sl-sap sistēmas API update-business-partner plūsma**

/deleteBusinessPartner/{id}:



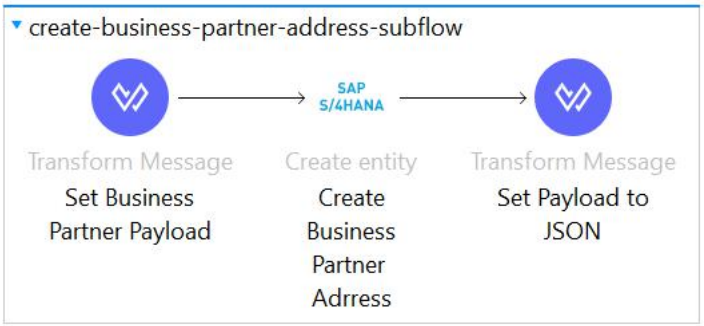
2.3 **lu-sl-sap sistēmas API delete-business-partner plūsma**

/getAllBusinessPartnerAddreses:



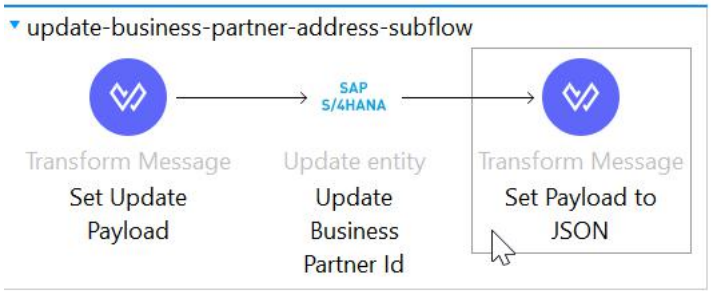
2.4 **lu-sl-sap sistēmas API get-all-business-partner-addresses plūsma**

/createBusinessPartnerAddress:



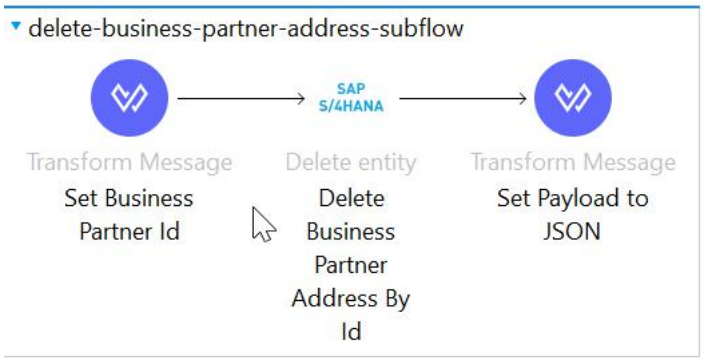
2.5 **lu-sl-sap sistēmas API create-business-partner-addresses plūsma**

/updateBusinessPartnerAddress/{id}:



2.6 **lu-sl-sap sistēmas API update-business-partner-addresses plūsma**

/deleteBusinessPartnerAddress/{id}:



2.6 **lu-sl-sap sistēmas API delete-business-partner-addresses plūsma**

3. Pielikums Datu migrācijas un sinhronizācijas procesa API

3.1 API Specifikācija

```
#%RAML 1.0
title: lu-pl=sap-sfdc
description: System API for SAP
mediaType: [application/json]
protocols: [HTTPS]
version: v1
securitySchemes:
  ClientId_Enforcement_policy: !include securitySchemas/clientSecretEnforcement.raml
securedBy: ClientId_Enforcement_policy
/startSyncProcess:
  get:
    displayName: Synchronize data
    responses:
      200:
        body:
          example:
              {
                  "status": "Success",
                  "recordsProcessed": 8
              }
/startMigrationProcess:
  get:
    displayName: Synchronize data
    responses:
      200:
        body:
          example:
            {
    "onCompletePhaseException": null,
    "loadingPhaseException": null,
    "totalRecords": 5,
    "elapsedTimeInMillis": 2104,
    "failedOnCompletePhase": false,
    "failedRecords": 0,
    "loadedRecords": 5,
    "failedOnInputPhase": false,
    "successfulRecords": 5,
```

```
    "inputPhaseException": null,
    "processedRecords": 5,
    "failedOnLoadingPhase": false,
    "batchJobInstanceId": "7d1ca360-d92c-11ec-98ef-82c5f2120aa5"
}
```

4. Pielikums Tīmekļa lietojumprogrammas pieredzas API

4.1 API Specifikācija

Pilnu specifikāciju skatīt AnyPoint platformas rīkos Design Center vai Exchange.

```
#%RAML 1.0
title: lu-el=webapp
description: Experience API for Web App
mediaType: [application/json]
protocols: [HTTPS]
version: v1
securitySchemes:
  ClientId_Enforcement_policy: !include securitySchemas/clientSecretEnforcement.raml
securedBy: ClientId_Enforcement_policy
/getAccountContact:
  get:
    displayName: Account Contact
    queryParameters:
      account:
        description: Account
        enum:
          ["yes", "no"]
        default: "no"
      contact:
        description: contact
        enum:
          ["yes", "no"]
        default: "no"
      id:
        description: Id
        type: string
        required: false
    responses:
      200:
        body:
/deleteAccountContact:
  delete:
    displayName: Delete Account Contact
    queryParameters:
      objectType:
        description: object type
        enum:
          ["account", "contact"]
      contactId:
        description: Id Contact
        type: string
        required: false
      accounttId:
        description: Id Account
        type: string
        required: false
    responses:
      200:
        body:
/updateAccountContact:
  patch:
    displayName: Update Account Contact
    queryParameters:
      objectType:
        description: object type
        enum:
          ["account", "contact", "all"]
      contactId:
        description: Id Contact
        type: string
        required: false
      accounttId:
        description: Id Account
        type: string
        required: false
    responses:
      200:
        body:
/createAccountContact:
  post:
    displayName: Create Account Contact
    responses:
      200:
        body:
```

5. Pielikums

5.1 Izveidoto API tehniskā informācija

Lai izsauktu API galapunktus jāizmanto:

Lietotājvārds: test.

Parole: test

SAP sistēmas API (lu-sl-sap):

pamataUrl: https://lu-sl-sap.us-e2.cloudhub.io/api/v1

gitRepository: https://github.com/GuntisRubenis/lu-sl-sap

Salesforce sistēmas API (lu-sl-sfdc):

pamataUrl: https://lu-sl-sfdc.us-e2.cloudhub.io/api/v1

gitRepository: https://github.com/GuntisRubenis/lu-sl-sfdc

Datu migrācijas un sinhronizācijas procesa API (lu-pl-sap-sfdc):

pamataUrl: https://lu-pl-sap-sfdc.us-e2.cloudhub.io/api/v1

gitRepository: https://github.com/GuntisRubenis/lu-pl-sap-sfdc

Tīmekļas lietojumprogrammas API (lu-el-webapp):

pamataUrl: https://lu-el-webapp.us-e2.cloudhub.io/api/v1

gitRepository: https://github.com/GuntisRubenis/lu-el-webapp

6. Pielikums

6.1 Testēšanas piemērs

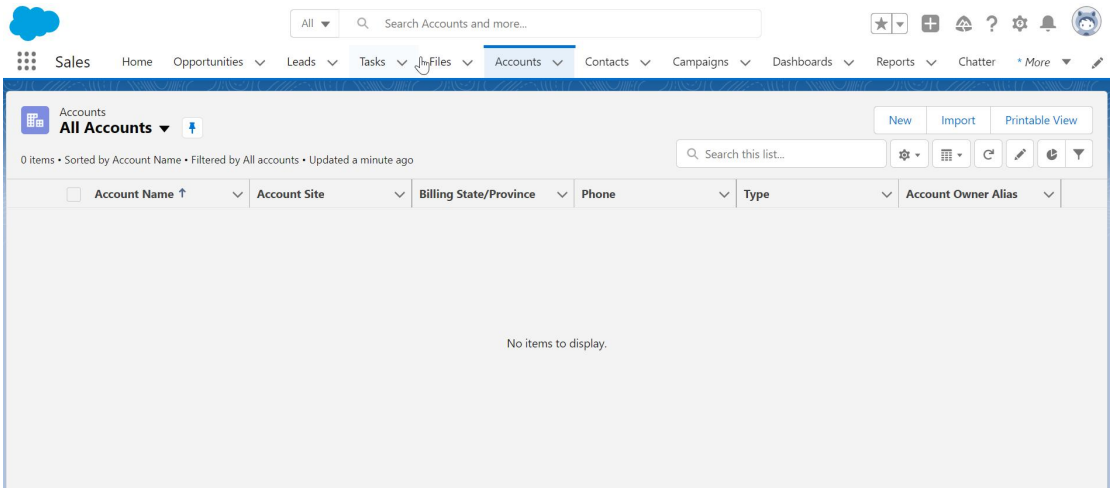
Postman kolekcijas glabājas GitHub sistēmā, un repozitorija ir pieejama:

Url: https://github.com/GuntisRubenis/lu-postman-collections

Tālāk tiek apskatīts datu migrācijas no SAP uz Salesforce testēšanas scenārija piemērs, kura laikā tiek notestēti vairāku API galapunkti un datu migrācijas process. Citi testēšanas scenāriji netika dokumentēti.

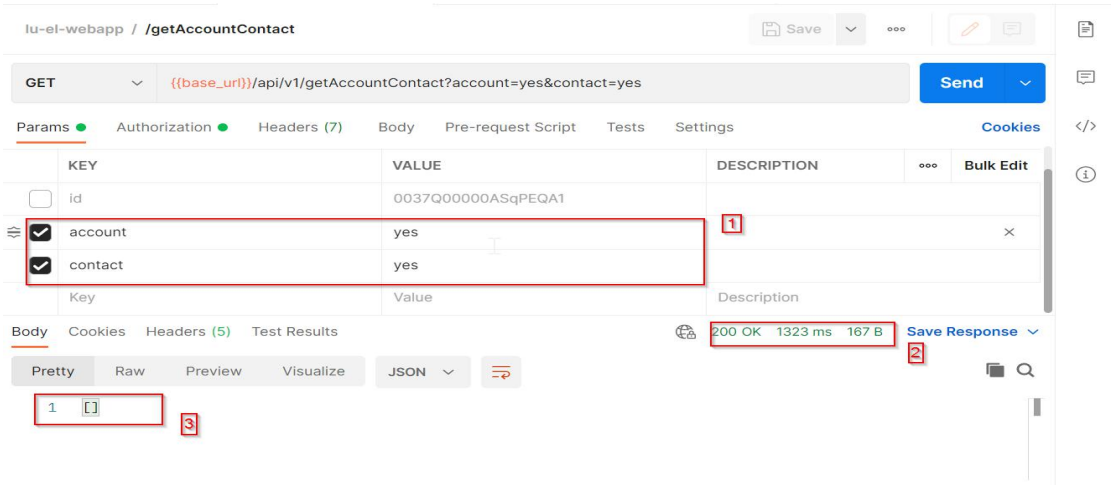
Datu sinhronizācijas testēšanas scenārija piemērs:

1. Ielogojamies Salesforce sistēmā un pārbaudām, vai tā satur Account ierakstus (skatīt 6. pielikuma att. 6.1).



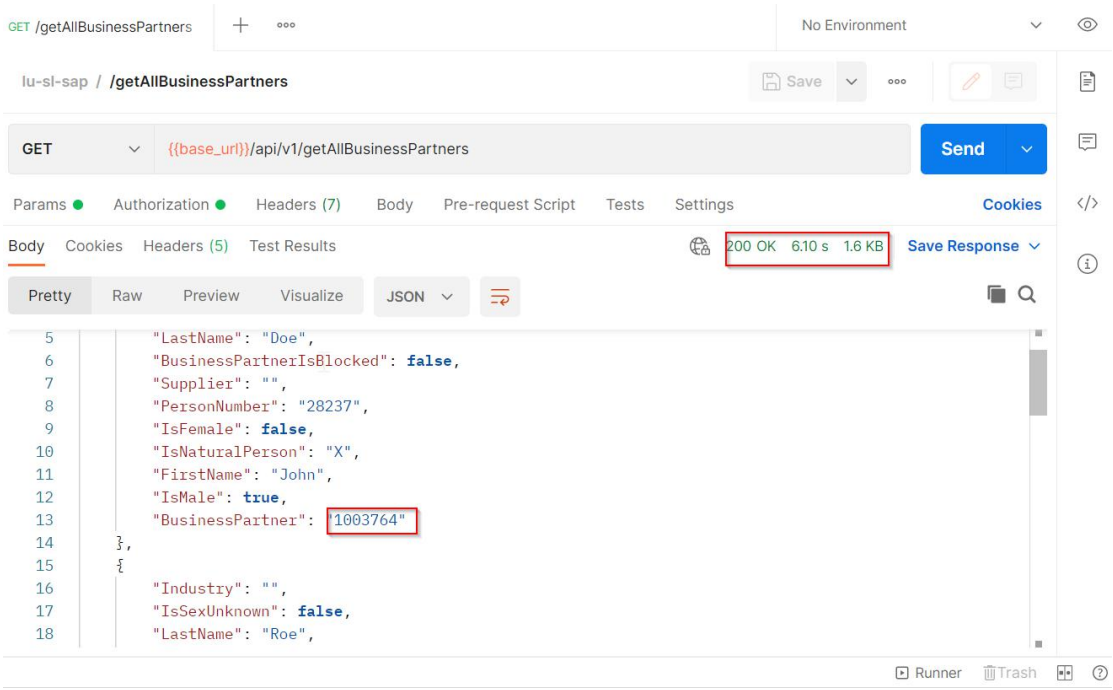
6.1 att. **Salesforce Account objekta skats**

2. Izsaucam Tīmekļa lietojumprogrammas pieredzes API (lu-el-webapp), galapunktu /getAccountContact, konfigurētu, lai tas atgriež visus Account un to Contact ierakstus. Pārbaudām vai arī šis galapunktus neatgriež nevienu Account un Contact ierakstu (skatīt 6. pielikuma att. 6.2).



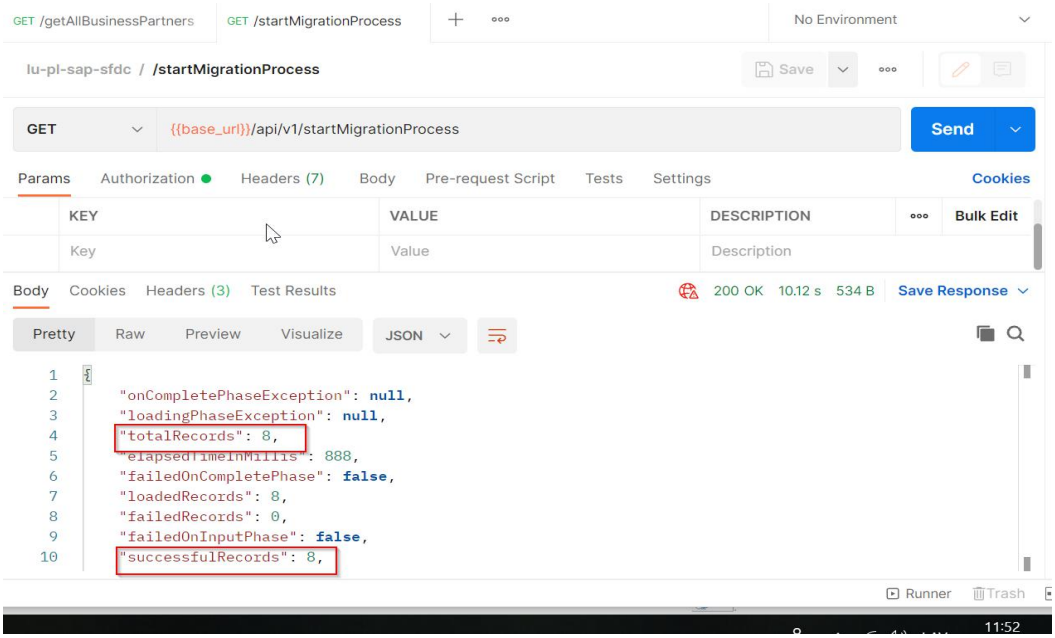
6.2 att. **lu-el-webapp pieredzes API /getAccountContact galapunkta atbilde**

3. Izsaucam SAP sistēmas API (lu-sl-sap), galapunktu /getAllBusinessPartners un pārbaudām, vai BusinessPartner ieraksti ir SAP sistēmā. Redzam, ka tiek atgriezts saraksts ar BusinessPartner ierakstiem (skatīt 6. pielikuma att. 6.3). Ekrānskatā ir grūti parādīt, bet autors secina, ka ir atgriezti 8 ieraksti.



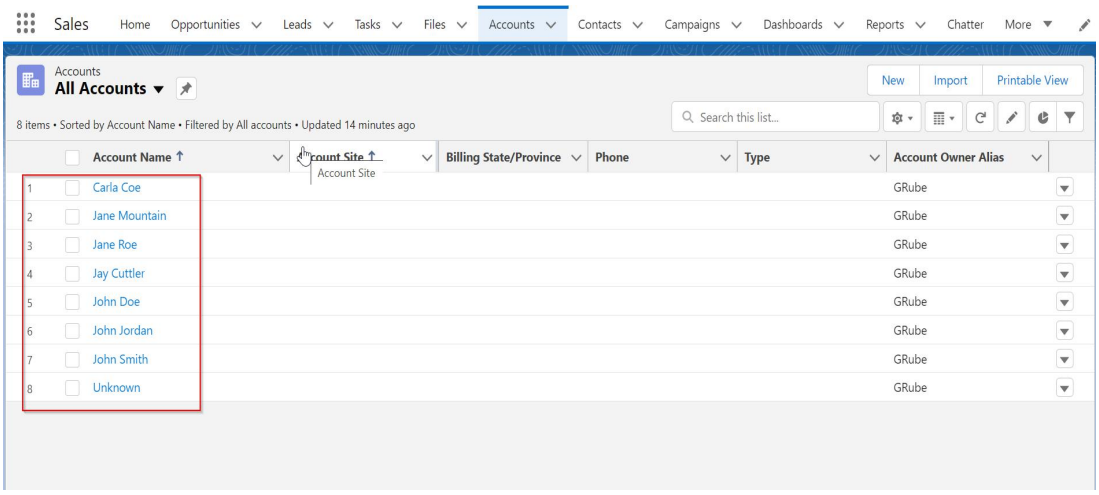
6.3 att. **lu-sl-sap sistēmas API /getAllBusinessPartners galapunkta atbilde**

4. Izsaucam procesa API (lu-pl-sap-sfdc) galapunktu /startMigrationProcess pārbaudām, vai migrācijas procesā nav bijušas kļūdas. (skatīt 6. pielikuma att. 6.4). Redzam, ka visi astoņi ieraksti tika apstrādāti veiksmīgi.



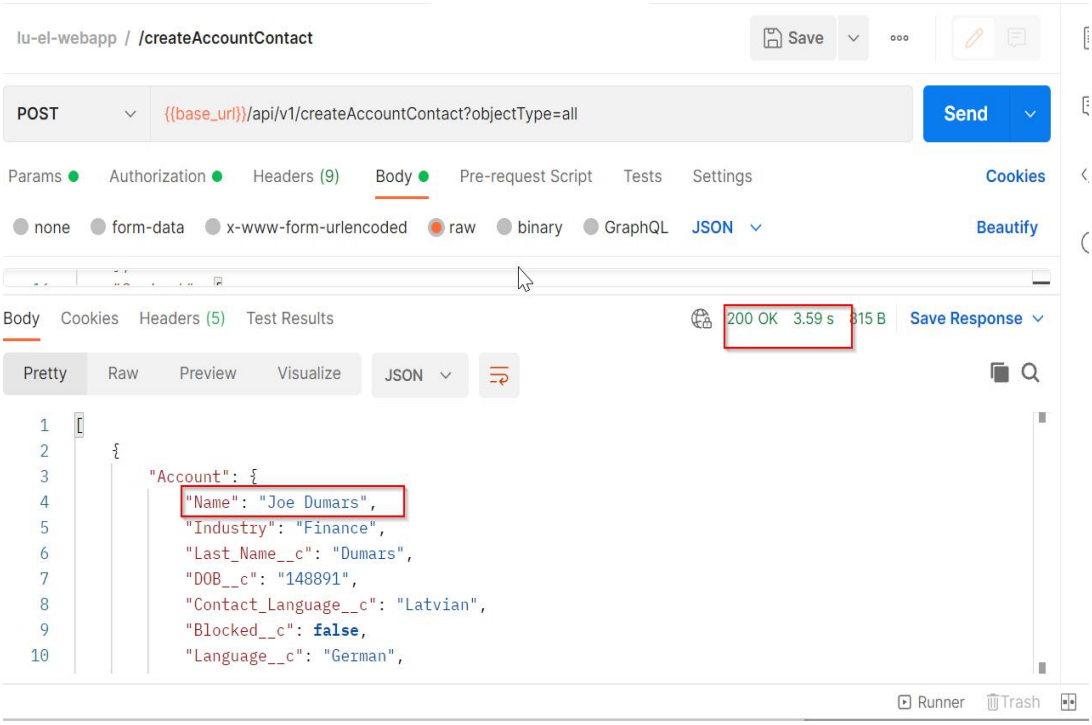
6.4 att. **lu-pl-sap-sfdc procesa API /startSyncProcess galapunkta atbilde**

5. Ielogojamies Salesforce sistēmā un pārbaudām, vai ir izveidoti jauni Account ieraksti (skatīt 6. pielikuma att. 6.5).

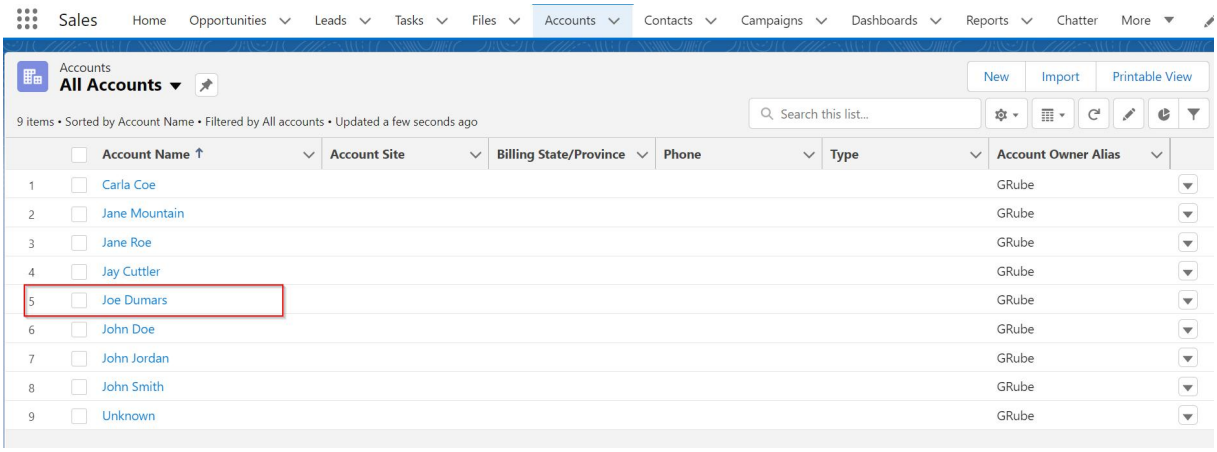


6.5 att. **Salesforce Account objekta skats**

6. Izsaucam Tīmekļa lietojumprogrammas pieredzes API (lu-el-webapp), galapunktu /createAccountContact, konfigurētu, lai tas izveidotu jaunu Account un tā Contact ierakstus (skatīt 6. pielikuma att. 6.6). Pārbaudām, vai Account ir izveidots (skatīt 6. pielikuma att. 6.7).

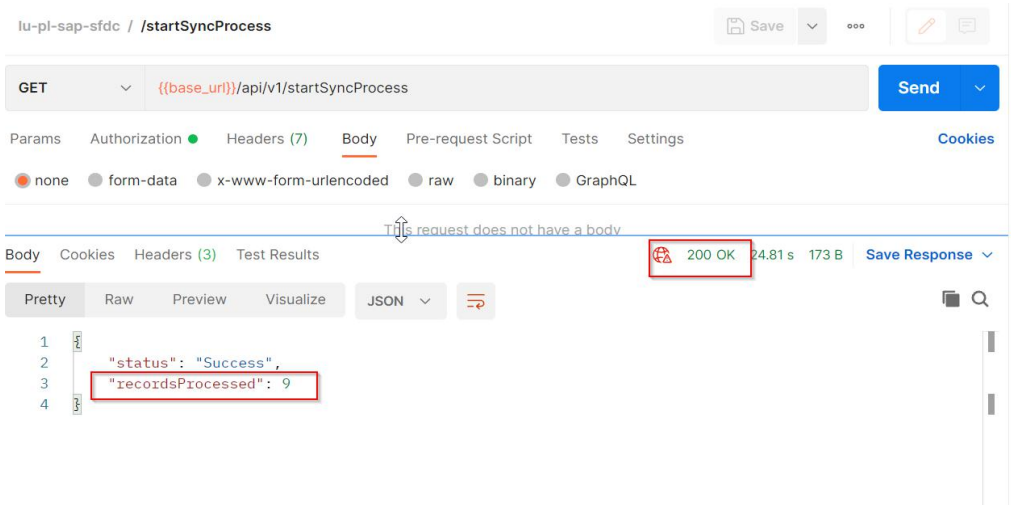


6.6 att. **lu-el-webapp pieredzes API /createAccountContact galapunkta atbilde**



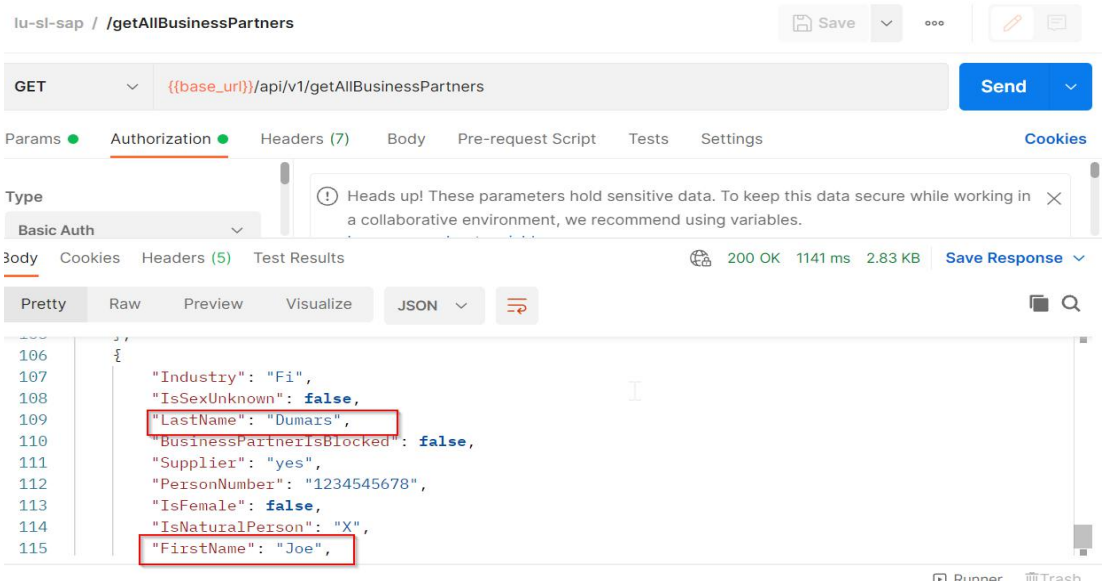
6.7 att. **Salesforce Account objekta skats**

7. Izsaucam procesa API (lu-pl-sap-sfdc), galapunktu /startSyncProcess pārbaudām, vai sinhronizācijas procesā nav bijušas kļūdas. (skatīt 6. pielikuma att. 6.8). Redzam, ka visi deviņi ieraksti ir apstrādāti.



6.8 att. **lu-pl-sap-sfdc procesa API /startSyncProcess galapunkta atbilde**

8. Izsaucam SAP sistēmas API (lu-sl-sap) galapunktu /getAllBusinessPartners un pārbaudām, vai jaunais BusinessPartner ieraksts ir SAP sistēmā. Redzam, ka tiek atgriezts arī jaunizveidotais ieraksts. (skatīt 6. pielikuma att. 6.9).



6.9 att. **lu-sl-sap sistēmas API /getAllBusinessPartners galapunkta atbilde**