Tampereen ammattikorkeakoulu

Sähköposti-integraation uudistaminen

Henri Laiho

OPINNÄYTETYÖ
Toukokuu 2019

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

LAIHO HENRI:
Sähköposti-integraation uudistaminen

Opinnäytetyö 42 sivua, joista liitteitä 3 sivua
Toukokuu 2019

Opinnäytetyön tarkoituksena oli kehittää toimeksiantajan sähköposti-integraatiolle uusi versio. Uudistettua versiota tarvittiin, koska vanhan sähköposti-integraation teknologiat ja työkalut olivat vanhentuneita eivätkä olleet ajan tasalla. Tästä johtuivat hankaluudet ylläpidossa ja sähköposti-integraation kehittämisessä.

Vanhempien versioiden ongelmakohtia miettien laadittiin vaatimukset uudelle versiolle. Vaatimusten pohjalta valittiin käytettävät teknologiat ja työkalut. Valinnoissa painotettiin ensisijaisesti soveltuvuutta vaatimuksiin. Toissijaisesti painotettiin toimeksiantajan työntekijöiden aiempaa kokemusta ja arvioitiin kunkin valinnan ylläpidon tasoa. Ylläpidon tasoon vaikuttivat uusien versioiden julkaisutahti, GitHubissa avoinna olevien ongelmien määrä ja niiden yritysten määrä, jotka ilmoittivat julkisesti käyttävänsä kyseistä teknologiaa ja työkalua.

Keskeisenä tuotoksena syntyi uusi versio sähköposti-integraatiosta. Sen toteuttamiseen valittiin node.js, TypeScript ja RabbitMQ. Node.js valittiin, koska asynkroninen ohjelmointi on sillä vaivatonta. Ylläpidettävyyden ja paremman kehittäjäkokemuksen vuoksi valittiin TypeScript JavaScriptin sijaan. TypeScriptin tuomat tyypitykset tekivät sähköposti-integraation kehittämisestä varmempaa. RabbitMQ valittiin, jotta voitiin luopua tietokannan käyttämisestä viestijonona, mikä mahdollisti uuden version monet ominaisuudet, kuten sähköpostiviestien rinnakkaisen käsittelyn ja yksittäisten osa-alueiden uudelleen yrittämisen virhetilanteiden sattuessa.

Uuden version kehittäminen sujui hyvin. Teknologia- ja työkaluvalinnat osoittautuivat onnistuneiksi. Niiden ansiosta kehittäminen oli sujuvaa ja kehittäjäkokemus hyvä. Uuden version jatkokehittäminen on vanhaa huomattavasti helpompaa ja varmempaa. RabbitMQ:n käyttäminen sähköpostiviestien käsittelyssä osoittautui toimivaksi ratkaisuksi.

Asiasanat: javascript, typescript, node.js, rabbitmq

ABSTRACT

Tampere University of Applied Sciences
Business Information Systems
Software Development

LAIHO, HENRI:
Email Integration Reform

Bachelor's thesis 42 pages, appendices 3 pages
May 2019

The goal of the thesis was to reform the client's customer service module's email integration. The client was in need of new improved version since the old version was outdated. The client had inadequate human resources to develop new versions of the email integration. Outdated tools and technologies lead to time consuming development of new features and while developing new features, the old version became ever harder to maintain properly.

Based on the new requirements, several technologies and tools were chosen for the new version. The primary criterion for new technologies and tools were the fit for the use case and the secondary criteria were prior experience amongst the client's developers. Focus was also on the maintainability of the candidates as prior problems were caused by inadequate maintenance.

Node.js was chosen as the main technology to build upon the new version. Email integration mainly consists of asynchronous operations and for such purpose node.js is an excellent choice. TypeScript was chosen over JavaScript due to requirement of easier maintainability and development. The static type checker helps to develop new version more swiftly. Change from database-based message queue to a proper message broker allows many new features. RabbitMQ was the chosen technology. In the new version multiple messages can processed at once. Message processing can be split into multiple segments. Each segment can be individually tried and retried if necessary.

Developing the reformed version was efficient and stress relieving thanks to TypeScript's typings. Chosen technologies and tools were found to be good choices. Node.js performed well and RabbitMQ solution worked. Developing the reformed version is both easier and more efficient for the developers.

Key words: javascript, typescript, node.js, rabbitmq

SISÄLLYS

LYHENTEET JA TERMIT

AMQP	*Advanced messaging queue protocol* on yhteentoimiva avoimen lähdekoodin viestijonoprotokolla
API	*Application programming interface* eli ohjelmointirajapinta on määritelty malli komponentin kanssa keskusteluun
cron	Ohjelmisto, joka ajaa sille ennalta määriteltyjä komentoja määrätyin aikavälein esimerkiksi skriptin kerran tunnissa
Docker	Ohjelma käyttöjärjestelmätason virtualisointiin
JavaScript	Dynaaminen, heikosti tyypitetty ohjelmointikieli
JSON	Ihmisen luettava tekstipohjainen tallennusmuoto, joka koostuu avain-arvo-pareista ja taulukoista
Kubernetes	Ohjelmisto esimerkiksi Docker-konteista koostuvan ympäristön hallintaan, skaalaukseen sekä levitykseen
Node.js	Palvelinpuolen JavaScript-suoritusympäristö, joka soveltuu hyvin asynkroniseen ohjelmointiin
npm	*Node package manager*, oletuspaketinhallintaohjelma JavaScriptille
ORM	Tietokantamallien kartoitus käytettäväksi olioparadigmalla
PHP	Palvelimella suoritettava ohjelmointikieli, tehty alun perin websivujen kehittämiseen
ReDoS	Palvelinestohyökkäys, jossa annetaan säännölliselle lausekkeelle syöte, jonka suorittaminen on erittäin raskasta
REST	*Representational state transfer*, ohjelmistoarkkitehtuurimalli webpalveluiden rajapintojen toteuttamiseen
SQL	*Structured Query Language*, standardi kyselykieli relaatiotietokantoja varten
TypeScript	Tyypitetty ohjelmointikieli, joka voidaan kääntää JavaScriptiksi

1 JOHDANTO

Tämän opinnäytetyön toimeksiantaja on eeedo oy. Eeedo oy on ohjelmistoalan startup-yritys, joka tarjoaa räätälöityjä ohjelmistopalveluita asiakaspalveluun ja kalustonhallintaan. Asiakaspalvelun toimintaan liittyy monia integraatioita, mutta juuri sähköposti-integraatio on olennainen osa asiakaspalvelua.

Opinnäytetyön tarkoituksena on selvittää sähköposti-integraation kehittämistä helpottavia ratkaisuja. Opinnäytetyössä arvioidaan sähköposti-integraation aiempien versioiden kehittämistä hankaloittaneet ongelmakohdat sekä arkkitehtuuriratkaisut, jotka haittaavat sähköposti-integraation jatkokehittämistä. Uutta versioita varten arvioidaan teknologiat ja työkalut, jotka sopivat vaatimusten täyttämiseen.

Vanha versio toimeksiantajan sähköposti-integraatiosta on jäänyt teknisesti jälkeen eikä toimeksiantajan puolesta sen kehittämiseen ole riittävää osaamista. Sähköposti-integraation kehittäminen on hidastunut ja sen ylläpitäminen vaikeutunut. Opinnäytetyön tavoitteena on tuottaa uusi versio sähköposti-integraatiosta toimeksiantajalle, jotta sähköposti-integraatioon haluttuja uusia ominaisuuksia voidaan helpommin. Lisäksi keskitytään sähköposti-integraation ylläpidettävyyteen.

Tässä raportissa tarkastellaan vaatimuksien pohjalta tehtyjä teknologia- ja työkaluvalintoja. Lisäksi tarkastellaan ratkaisuja, joilla pyritään varmistamaan sähköposti-integraation helppo jatkokehittäminen ja ylläpitäminen. Itse sähköposti-integraation toimintaa tarkastellaan teknisestä näkökulmasta ja raportin ymmärtäminen vaatii ohjelmistoalan osaamista.

Opinnäytetyön tuloksena syntyi uusi versio sähköposti-integraatiosta. Uusi versio on edeltäviä versioita paremmin jatkokehitettävissä ja ylläpidettävissä. Se toimii varmemmin ja nopeammin. Teknologia- ja työkaluvalinnat tekevät siitä helpomman ylläpitää.

2 SÄHKÖPOSTI-INTEGRAATIO

Tässä luvussa kuvaillaan, miten aikaisempi versio sähköposti-integraatiosta toimii käytännössä. Toimintaa tarkastellaan ei-teknisestä näkökulmasta, minkä jälkeen keskistytään sähköposti-integraation ongelmiin. Lopuksi tarkastellaan uusia vaatimuksia ja esitetään toimenpiteitä, joilla ongelmia voitaisiin ratkaista.

2.1 Toimintalogiikka

Sähköposti-integraatio käsittelee sähköpostiviestit sähköpostilaatikosta asiakaspalveluun. Sähköposti-integraation tapahtumaketju alkaa, kun asiakas lähettää sähköpostiviestin sähköpostilaatikkoon. Sähköposti-integraatio käynnistyy asetetuin aikavälein ja hakee Outlook-palvelun rajapinnan kautta asetetun määrän viimeisimpiä viestejä. Kun uusimmat viestit on haettu, niiden uniikkeja tunnisteita verrataan tietokantaan tallennettuihin, jo käsiteltyihin, uniikkeihin tunnisteisiin. Tunnisteet, jotka eivät vielä löydy tietokannan jonosta (käsittely kesken) tai arkistosta (käsitelty), lisätään tietokannan jonoon.

Kun viestit on lisätty jonoon, sähköposti-integraatio alkaa käsittelemään viestejä. Sähköpostiviestit käsitellään viesti kerrallaan. Yhdellä sähköpostiviestillä on useita osa-alueita, jotka käsitellään osa-alueet toisensa jälkeen peräkkäin. Sähköpostiviestien käsittelyssä luodaan asiakaspalvelutikettejä, joihin liitetään asiakkaan sähköpostiosoitteen perusteella löytyvät asiakastiedot. Jos sähköpostin mukana on liitetiedostoja, myös ne siirretään sähköpostipalvelimelta asiakaspalvelun omaan tiedostopalveluun ja liitetään tehtävään.

Asiakaspalvelija käsittelee tiketin, ja vastaa asiakkaalle sähköpostilla. Sähköpostiviesti lähetetään asiakkaan sähköpostiin samasta osoitteesta, johon asiakkaan lähettämä sähköpostiviesti on alun perin lähetetty. Sähköpostiviestin otsikkoon lisätään merkkijono, joka muodostuu tiketin tunnisteesta sekä tiketin tiedoista muodostetusta tarkisteesta.

Tapahtumaketjussa seuraavaksi asiakas huomaa saaneensa sähköpostitse vastauksen sähköpostiviestiinsä asiakaspalvelulta. Asiakas vastaa sähköpostiviestiin ja sähköpostin otsikkoon lisätyllä tunnisteella tunnistetaan sähköpostiviestille oikea tiketti. Asiakkaan lähettämä sähköpostivastaus lisätään kommenttina tiketille ja tiketin tila muutetaan siten, että se päätyy takaisin asiakaspalvelijoiden käsiteltäväksi. Mikäli oikeaa tikettiä ei löydetä, luodaan uusi tiketti viestin perusteella. Kunhan asiakas on liitetty tiketteihin oikein, niin asiakaspalvelija pystyy yhdistämään viestit toisiinsa käsin.

2.2 Aiempien versioiden ongelmat

Aiempien versioiden näkyvin ongelma kehittäjälle on niiden ohjelmointikieli. PHP on alun perin websivujen tekemiseen tarkoitettu palvelimella suoritettava ohjelmointikieli. PHP on saanut ikävän maineen vuosien aikana. Maine johtuu sekavasta syntaksista sekä sen standardikirjaston siivottomuudesta. Lisäksi PHP:ssä on todella paljon epäjohdonmukaisuuksia (Woods, E. 2012). PHP:llä ilman ohjelmointikehyksiä on helppo kirjoittaa sekavaa koodia. Ohjelmointikielenä PHP ei kuitenkaan ole niin iso ongelma, että sillä ei pystyisi ohjelmoimaan. Toinen ongelma on se, että suurin osa yrityksen muista projekteista on siirtynyt käyttämään JavaScriptiä, myös palvelinpäässä. Koska nykyisillä kehittäjillä on huomattavasti enemmän JavaScript-osaamista kuin PHP-osaamista, on ohjelmavirheiden korjaamiseen ja ominaisuuksien kehittämiseen niukasti tekijöitä yrityksen sisällä.

Mitä tulee koodin luettavuuteen, aiempien versioiden koodin rakenne on sekava. Sähköposti-integraation logiikka on hyvin suoraviivainen eikä sisällä yhtäaikaisesti suoritettavaa logiikkaa. Tästä huolimatta logiikkaa ei ole yksinkertaista seurata koodia lukemalla. Sähköposti-integraation logiikka on jaettu useisiin tiedostoihin. Tiedostojen nimet ja niiden sisällön jaottelu eivät ole intuitiivisia. Logiikkaa on myös jaettu useisiin luokkiin, joiden olemassaololle ei ole kaikissa tapauksissa mitään perusteita. Iso osa funktioista rikkoo periaatetta, jossa funktioilla on vain yksi vastuu. Funktiot ketjuttavat kutsuja sen sijaan, että ne tekisivät yhden asian ja palauttaisivat arvoja. Tästä johtuen koodin kehittäminen ja testaaminen on hankalaa, koska funktioihin ei ole eristetty yksittäisiä toimintoja. Tämän asian kor-

jaaminen vaatisi integraation testaamista paremmin. Ongelma on, että sähköposti-integraation logiikka on sisällytetty useisiin funktioihin eikä ole siten helposti tarkasteltavissa.

Sähköposti-integraatio on hidas ja virhetilanteissa erittäin hidas. Sähköposti-integraation aiemmissa versioissa toimintalogiikka on ollut synkronista. Ensiksi kirjaudutaan Outlook-rajapintaan, minkä jälkeen kirjaudutaan asiakaspalvelun rajapintaan. Kirjautumisten jälkeen Outlookin rajapinnasta haetaan uusimmat sähköpostiviestit ja ne lisätään asianmukaisesti tietokannan jonoon käsiteltäviksi. Viestien hakemisen jälkeen viestejä käsitellään yksitellen pois jonosta. Mikäli viestien käsittely epäonnistuu esimerkiksi pyynnön ulkopuoliseen palveluun epäonnistuessa, niin viestit käsitellään uudestaan seuraavalla kerralla. Ruuhkatilanteessa voidaan puhua useista kymmenistä minuuteista ennen kuin viestiä käsitellään uudemman kerran. Tapauksessa, jossa sähköposti-integraatioprosessi kaatuu odottamattomasta virheestä, käsiteltävä viesti jää tilaan ”käsittelyssä”. Seuraava prosessi merkitsee edellisestä integraatiosta tilaan ”käsittelyssä” jääneet viestit epäonnistuneiksi. Vasta sitä seuraava prosessi aloittaa viestin käsittelyn uudelleen. Tällainen toimintatapa viivästyttää selvästi sähköpostiviestien integrointia asiakaspalvelujärjestelmään. Ongelmana on tietokannan käyttäminen viestijonona.

Tietokanta toimii yksinkertaisena viestijonona, mutta siitä puuttuvat monet perusominaisuudet (Stack Overflow 2012). Aiemmista versioista puuttuvat kokonaan tuet useammalle käsittelijälle ja useammalle viestijonolle. Tietokannan rooli viestijonona on toiminut aiemmissa versioissa, koska viestit käsitellään yksitellen ja käsittelijöitä on vain yksi. Tietokannan skaalautuminen ja viestijonojen toimintojen toteuttaminen manuaalisesti on monimutkaista (Johansson, L. 2015). Lisäksi aiemmissa versioissa koko sähköpostiviestin käsittely on vain yksi tehtävä. Käsittely voitaisiin jakaa useampiin osa-alueisiin, jotta sähköposti-integraatiosta saataisiin nopeampi, paremmin virheitä kestävä sekä helpommin testattava.

Sähköposti-integraation aiempien versioiden kehittäminen on hankalaa. Sähköposti-integraatiosta ei ole tarjolla kunnollista kehitysympäristöä ja koko ketjun testaaminen on aina työläs projekti. Ensiksi pitää lähettää sähköpostiviesti laatikkoon ja sitten suorittaa sähköposti-integraatio, joka kerta kun testaa muutoksia

koodissa. Tämä on käytännössä end-to-end -testaamista ja työlästä yksittäisten muutosten testaamiseen. Tietokannasta näkee tapahtuneet virheet ja käsittelyn tilan. Asiakaspalveluympäristöstä tarkistetaan, onko viesti mennyt perille oikeassa muodossa. Koodin sekavuuden ja näiden syiden takia aiempien versioiden ongelmina ovat olleet huonot testit ja testien kattavuus on ollut huono. Koska aiempia arkkitehtuureja ei ole suunniteltu testattavaksi, niin sähköposti-integraation toimintaa on testattava käsin sitä kehittäessä.

2.3 Uudet vaatimukset

Uuden sähköposti-integraation kehityksen pitäisi olla ketterämpää ja vaivattomampaa yrityksen työntekijöille. Ensimmäisenä vaatimuksena olisi ohjelmointikielen vaihtaminen vanhentuneesta ja hankalasti luettavasta PHP:stä uudempaan ohjelmointikieleen, jolle on tarjolla kirjastoja sekä työkaluja parempaa ylläpitoa varten. Ohjelmointikielen tulisi olla sellainen, jota yrityksen työntekijät osaavat laajemmin.

Koodin uudessa sähköposti-integraatiossa tulisi olla paremmin organisoitu. Nykyisen sähköposti-integraation loogista kulkua pitää paremmin jakaa loogisiin kokonaisuuksiin. Koodin tulisi keskittää kokonaisuuksien logiikkaa laajempiin funktioihin ja suorittaa useampia funktioita, jotka palauttavat arvoja, joita laajempi funktio sitten käyttää seuraavien loogisten palasten käsittelyssä. Tämä tarkoittaa, että funktioiden pitää tehdä yksi asia nimensä mukaisesti.

Viestien käsittelyaikojen tulisi olla lyhyempiä. Viestin käsittelyaika tässä tapauksessa tarkoittaa aikaa siitä, kun viesti saapuu sähköpostilaatikkoon, siihen aikaan, kun viesti on käsitelty ja tarvittavat toimenpiteet järjestelmään on tehty. Uuden sähköposti-integraation ei tulisi odotella toimettomana, että toiset palvelut vastaavat sen pyyntöihin. Viestejä ei tulisi käsitellä yksi kerrallaan osa-alueittain synkronisesti, jotta viestejä ei kasaantuisi järjestelmään luettavaksi. Lisäksi viestien käsittelyn pitäisi olla reaktiivisempaa ja alkaa aiempaa pienemmällä viiveellä.

Sähköposti-integraation tulisi käsitellä virhetilanteita aiempia versioita paremmin. Sähköpostiviestien käsittelyn osa-alueet tulisi voida yrittää käsitellä uudelleen itsenäisesti muista osa-alueista virhetilanteissa. Esimerkiksi liitetiedoston lisääminen epäonnistuessa verkkoyhteysongelmien vuoksi, tulisi liitetiedostoa yrittää lisätä uudelleen. Aiemmassa sähköposti-integraatiossa tämänkaltaiset tilanteet johtivat joko siihen, että koko sähköpostiviesti käsiteltiin uudelleen tai siihen, että liitetiedosto jäi lisäämättä. Uuden sähköposti-integraation pitäisi myös voida tunnistautua rajapintoihin uudestaan käyttöoikeuksien vanhentuessa.

Sähköposti-integraatiossa tapahtuvien virhetilanteiden tulisi olla näkyvämpiä. Sähköposti-integraation tulisi ilmaista selkeästi, kun se menettää oikeuden hakea sähköpostilaatikosta sähköpostiviestejä tai asiakaspalvelun rajapinnasta dataa. Myös tietokantayhteyden ja viestijonoyhteyden katkeaminen tulisi hoitaa hallitusti. Virhetilanteista syntyvät lokitietojen tulee olla selkeässä muodossa ja kehittäjille helposti saatavilla. Koska sähköposti-integraatio pyörii Kubernetes-ympäristössä, palvelua tulisi suorittaa Docker-kontissa cron-tehtävän sijaan. Tällöin palvelun lokitietoja saadaan helposti järjestettyä samalla tavalla kuin muissakin palveluissa.

2.4 Ratkaisuja ongelmiin ja vaatimuksiin

Moneen vaatimukseen on helpottavana tekijänä ohjelmointikielen vaihdos. Aiemmissa versioissa ohjelmointikieli sekä siitä juontuvat ongelmat ovat suuressa osassa, kun tarkastellaan kaikkia sähköposti-integraatioon liittyviä ongelmia. Ohjelmointikielen vaihdoksen tulisi helpottaa ylläpidettävän integraation kehittämistä. Lisäksi ohjelmointikielen tulisi paremmin sopia yrityksen nykyiseen osaamiseen.

Jotta sähköposti-integraatiosta voidaan toteuttaa helpommin monitoroitava ja ylläpidettävä integraatio, tulee siitä tehdä Kubernetes-ympäristöä varten Docker-kontti. Tällöin integraatiosta on tehtävä tietoinen eri tiloistaan. Tämä vaatii, että luovutaan sähköposti-integraation suorittamisesta cron-tehtävinä, jossa sähköposti-integraatio on aikaväleittäin suoritettava komentosarja. Uuden mallin mu-

kaisesti integraatio pyörii jatkuvasti, jolloin myös sen toimintaa on helpompi seurata. Tällöin esimerkiksi pitää käsitellä oikein tilanteet, joissa kirjautumistiedot vanhenevat tai jotkin riippuvaisuuksista eivät toimi normaalisti.

Tietokannan merkittävyyttä sähköposti-integraation kannalta on vähennettävä. Tietokantaa tulee käyttää vain tallentamaan sähköpostiviestejä yksilöiviä tunnisteita sekä konfiguraatioasetuksia. Toimintoja tulee siirtää tietokannalta viestijonon hallittavaksi. Sähköposti-integraation ajamisesta syntyviä virhetilanteita ei talleteta tietokantaan kehittäjän tarkasteltavaksi. Kaikki lokitiedot tallennetaan siten, että niitä voidaan tarkistella ulkoisilla työkaluilla ja pitää niistä arkistoa.

3 KÄYTETYT TEKNOLOGIAT JA TYÖKALUT

Tässä luvussa käsitellään mitä teknologioita ja työkaluja on valittu sähköpostiintegraation uuden version toteuttamiseen. Alaluvut teknologiat ja työkalut sisältävät kuvauksen ja valintakriteerit valinnoista.

3.1 Teknologiat

Tärkein valintakriteeri teknologioille oli niiden soveltuvuus tähän käyttötapaukseen. Sen lisäksi painotettiin yrityksen työntekijöiden aikaisempia kokemuksia teknologioista. Arvioinnissa huomioitiin myös teknologioiden ajantasaisuutta sekä ylläpidon tasoa, joka vaikuttaa ylläpidettävyyteen. Ylläpidon tasoa arvioitiin uusien versioiden julkaisutahdilla, GitHubissa olevien avoimien ongelmien määrällä sekä kuinka moni yritys ilmoittaa käyttävänsä kyseistä teknologiaa.

3.1.1 RabbitMQ

RabbitMQ on käytetyin avoimen lähdekoodin viestinvälittäjä (RabbitMQ). RabbitMQ tarjoaa AMQP-protokollaa tukevan palvelimen, joka huolehtii viestijonon toiminnasta sekä lisäksi useita työkaluja palvelimen hallintaan. RabbitMQ:lle voidaan lukea viestejä jonoon ja viestejä voidaan reitittää monipuolisesti. RabbitMQ tukee viestien tallentamista myös kovalevylle, jotta viestit säilyvät, mikäli palvelin kaatuu. Lisäksi RabbitMQ tukee vahvistuksia viestien onnistumisista sekä viestien palauttamista jonoihin. Tämä mahdollistaa yksittäisten osa-alueiden useamman käsittelykerran, kunnes viestit ovat käsitelty onnistuneesti.

Vaihtoehtoina RabbitMQ:lle oli Amazon Simple Queue Service, Apachen Kafka ja ActiveMQ. Amazonin tarjoama palvelu hylättiin, koska se toimii pilvessä eikä se siten sovellu konesaleihin toteutettaviin projekteihin (Amazon SQS). Apachen tuotteista Kafka on suositumpi kuin ActiveMQ. ActiveMQ:n JavaScript-kirjastovalikoima ei ollut ajantasainen, mutta Kafkalle löytyi laadukas KafkaJS-kirjasto.

Kafka ja RabbitMQ pitkälti tarjoavat samankaltaisia ominaisuuksia. RabbitMQ:sta on kuitenkin jo kokemusta yrityksessä ja asiakasympäristöissä pyörii jo RabbitMQ-palvelin valmiiksi. Kafkassa on alkeellisemmat reititystoiminnallisuudet ja tarve pitää yllä tietoa viestijonoon liittyvistä viesteistä (Stack Overflow 2012). Nämä tekevät Kafkasta epäsuotuisan teknologiavalinnan tässä käyttötapauksessa.

3.1.2 Node.js

Node.js on avoimen lähdekoodin JavaScript-ajoympäristö. Node.js:n pohjana on Googlen Chrome-selaimen V8 JavaScript-moottori. Alkuperäinen Node.js:n kehittäjä Ryan Dahl julkaisi ensimmäisen version toukokuussa 2009 ja nykyinen versio on versio kymmenen. Perinteisesti JavaScript on ollut selaimissa suoritettava ohjelmointikieli, mutta node.js mahdollistaa JavaScript-sovelluksien ajamisen myös selainten ulkopuolella. Esimerkiksi Node.js:ää käytetään työpöytäsovellusten kehittämiseen Electron-ohjelmistokehyksellä. Node.js on laajassa käytössä ja suurin osa Fortune 500 -yrityksistä käyttää Node.js:ää (Nodejs.org).

Node.js on erittäin hyvä valinta asynkronisen koodin suorittamiseen. Node.js käyttää sisäisesti C++:lla useita säikeitä, mutta itse koodi on yksisäikeinen. Node.js:ssä suoritettavat funktiot lisätään Event Loop -nimiseen silmukkaan, joka mahdollistaa funktioiden suorittamisen palautumisen, kun ulkoiset pyynnöt ovat valmistuneet. Koska sähköposti-integraatiossa koodi on kevyttä ja sisältää yksinkertaisia datamuunnoksia sekä paljon asynkronisia http-pyyntöjä, on node.js hyvä valinta (The Node.js Event Loop…).

Node.js:lle yhtenä vaihtoehtona oli Java. Javalle löytyy kokemusta yritysten työntekijöiltä. Vaihtoehto ei kuitenkaan saanut kannatusta, koska suurin osa yrityksen projekteista on siirtymässä kohti JavaScript-kehitystä. Yrityksessä ei ole aiempia projekteja, jotka olisi toteutettu Javalla.

3.1.3 TypeScript

Node.js:ää käyttäessä JavaScript on oletettu valinta ohjelmointikieleksi. Uusissa vaatimuksissa oli ylläpidettävyys. JavaScript ei ole kuitenkaan tyypitetty ohjelmointikieli. Tyypityksen puute tuo sähköposti-integraation kehittämiseen ylläpidettävyysongelman. Uusia vaatimuksia mukaillen JavaScriptiin tulisi lisätä tyypitys ja sitä varten on muutama ratkaisu.

Ensimmäinen vaihtoehto on Facebookin Flow, joka on staattinen tyypintarkastin JavaScriptille. Flow lisää tyypityksen JavaScriptiin omilla kommenteillaan ja pystyy itse päättelemään osan tyypityksestä suoraan koodista. Flow on suosittu Reactin yhteydessä, joka on myös Facebookin oma kirjasto. Flow:n ongelmiksi on kuitenkin osoittautunut työkalut. Flow myös näyttää jääneen kilpailijansa TypeScriptin varjoon suosiossa (State of JS – Flavors). Kun Facebookin omatkin avoimen lähdekoodin projektin siirtyvät pois Flow:sta TypeScriptiin ([RFC] Migrate Jest to TypeScript).

TypeScript on toinen vaihtoehto tyypitykselle JavaScriptissä. TypeScript on Microsoftin kehittämä JavaScriptin *superset* eli kaikki pätevä JavaScript on myös pätevää TypeScriptiä. TypeScriptin tyypitys tapahtuu lisäämällä kaksoispiste muuttujan perään sekä mitä tyyppiä muuttuja on. TypeScript tämän lisäksi toteuttaa monia JavaScriptiin tulossa olevia ominaisuuksia ja on siten myös erittäin miellyttävä ohjelmointikieli ohjelmoida. TypeScriptin käyttökokemus on Flow:ta mukavampi ja esimerkiksi Microsoftin suositussa Visual Studio Code-koodieditorissa on ensimmäisen luokan tuki TypeScriptille. Lisäksi TypeScriptin kehitys on aktiivisempaa ja uusia ominaisuuksia lisätään TypeScriptiin nopeammin kuin Flow:iin.

Sekä Flow että TypeScript soveltuvat hyvin helpottamaan tyypityksen puutteen aiheuttamaa ylläpidettävyysongelmaa. Flow:n yhdeksi heikkoudeksi osoittautui huono tuki kolmannen osapuolen avoimen lähdekoodin kirjastoissa. Viimeisimmän JavaScript-kyselyn mukaan TypeScript on jo ottanut huomattavan etumatkan Flow:hun verrattuna suosiossaan (State of JS - Flavors). TypeScriptistä on lisäksi jo kokemusta yrityksessä, sillä osa projekteista on siirtynyt käyttämään TypeScriptiä tyypitykseen sekä kaikki uudet projektit aloitetaan suoraan TypeScript-

pohjalta. Tämän projektin työkaluvalintojen arvioinnissa Flow ei tarjonnut riittäviä parannuksia tyypitykseen, jotta olisi nähty syytä kokeilla sitä tässä projektissa.

3.1.4 MariaDB

MariaDB on avoimen lähdekoodin relaatiotietokanta. MariaDB on jatkokehitetty versio MySQL:stä ja sen kehittäjät ovat alkuperäisiä MySQL:n kehittäjiä ajalta ennen kuin Oracle osti MySQL:n. MariaDB pyrkii olemaan mahdollisimman yhteensopiva MySQL:n ominaisuuksien kanssa ja samalla kehittää eteenpäin suorituskykyä ja muita ominaisuuksia. MariaDB näyttää keskittyvän rohkeasti uusien ominaisuuksien kehittämiseen (Dimitrov 2014). Tämän lisäksi MariaDB:n kehitystahti on ripeämpi ja tietoturvapäivitykset hoidetaan asianmukaisesti (Seravo 2015).

MariaDB valittiin, koska se on jo yrityksen käytössä monissa asiakaspalveluprojekteissa. Käyttötarkoitukseen olisi sopinut monikin relaatiotietokanta, koska tietokannan käyttö on hyvin yksinkertaista. MariaDB on kuitenkin parempi valinta MySQL:ään verrattuna sen avoimuuden takia. Oraclen rooli MySQL:n kehityksessä myös on arveluttava tekijä.

3.1.5 Docker

Docker on avoimen lähdekoodin ohjelmisto, joka virtualisoi käyttöjärjestelmätason. Docker inc. kehitti Dockerin vuonna 2013. Dockeria käytetään Docker-konttien rakentamiseen ja suorittamiseen. Docker-konttiin määritellään ennalta kaikki riippuvuudet, mitä kontit tarvitsevat sekä rakennusohjeet kontin rakentamiseen. Dockerin hyöty on, että kontti rakennetaan joka kerta samalla tavalla ja konttia voi suorittaa kaikilla yleisimmillä käyttöjärjestelmillä. Kontti toimii riippumatta ympäristön käyttöjärjestelmästä tai siitä, onko kehittäjä asentanut tarvittavia työkaluja. Docker myös eristää kontin toiminnan kaikista muista konttiin kuulumattomista asioista, joka tekee käytöstä turvallisempaa ja ylläpidettävämpää.

Docker valittiin teknologiana, jotta kaikille kehittäjille voidaan taata sama ajoympäristö käyttöjärjestelmästä tai asennetuista työkaluista riippumatta. Dockerin vaihtoehtona olisi ollut laatia tarkat vaatimukset siitä mitä teknologioita ja työkaluja tulee olla asennettuna, jotta sähköposti-integraation pyörisi. Tämänkaltainen toiminta olisi käytännössä johtanut siihen, että kaikki muutokset teknologioiden asetuksiin tai työkaluihin olisivat pysyviä sekä paikallisia kehittäjälle. Ennen pitkään tällaisia muutoksia olisi jäänyt kirjaamatta ohjeisiin. Lisäksi Docker on helpottava teknologia Kubernetes-ympäristöissä, kun koko ohjelmiston voi määritellä ja suorittaa automaattisesti.

3.2 Työkalut

Tärkein työkalujen valintakriteeri oli työkalun soveltuvuus käyttötapaukseen. Lisäksi huomioitiin aikaisempi kokemus työkalusta käytännössä sekä kuinka ylläpidettyjä työkalut olivat. Työkalun ylläpidettävyyteen vaikuttavat työkalun GitHub-sivulta löytyvien ongelmien ja korjausehdotusten määrä. Iso määrä avoimia ongelmia kertoo käyttäjien ongelmista työkalujen kanssa. Avointen korjausehdotusten iso määrää antaa kuvan siitä, että työkalua ei ylläpidetä aktiivisesti. Myös GitHubin päivityshistoria kertoo ylläpidon aktiivisuudesta.

3.2.1 Yarn

Yarn on Facebookin kehittämä pakettienhallintasovellus. Yarn kehitettiin, koska Node.js:n vakio pakettienhallintasovellus npm ei enää vastannut Facebookin tarpeita. Ongelmakohtia oli tietoturvassa, suorituskyvyssä sekä yhteisyydessä (McKenzie, S; Nakawaza, C; Kyle, J). Npm:n kehittäjä npm inc. vastaa myös yleisimmästä JavaScriptin pakettirekisteristä npm registrystä.

Molempia yarnia sekä npm:ää käytetään JavaScriptin pakettienhallintaan. Komennoilla yarn init ja npm init, luodaan uusi package.json-tiedosto, joka kertoo pakettienhallintasovelluksille projektista tietoja. Tärkeimpänä tietona projektin tarvitsemat ulkoiset paketit. Sovelluksilla hallitaan mitä paketteja projektiin lisä-

tään. Sovellusten avulla ladataan riippuvuudet ja niiden sisältämät omat riippuvuudet ja niin edelleen. Lisäksi molemmista sovelluksista löytyy toiminto, jolla koko projektin paketit voidaan tutkia haavoittuvaisuustietokannan avulla. Tällöin projektiin haavoittuvaisuudet selviävät vaivattomasti. Tämä on tärkeää, koska sähköposti-integraatio vastaanottaa syötteensä loppukäyttäjiltä.

Yarnin vaihtoehtona oli npm. Npm:n toiminta on kehittynyt sen jälkeen, kun Yarn vuonna 2016 horjutti sen asemaa. Npm:n kehitystahti sekä ominaisuudet ovat jääneet hieman jälkeen Yarnista. Koska Yarnissa on myös npm;n ominaisuudet, on loogista käyttää Yarnia. Tarpeen tullen Yarnista voi luopua poistamalla yarn.lock-tiedoston ja käyttää npm:ää yarnin sijaan. Suurin osa yrityksen uusista projekteista käyttävät Yarnia pakettienhallintaan.

3.2.2 Amqp.Node

Amqp.Node on JavaScriptille kehitetty RabbitMQ-palvelimen kanssa kommunikointia varten tarkoitettu paketti. Amqp.Node sisältää rajapinnan RabbitMQ-palvelimelle yhdistämiseksi ja sen käyttämiseksi. AMQP ei ole rakenteelta yksinkertainen ja Amqp.Node keskittyykin kanavien ja jonojen käsittelyyn (Amqp.node).

Paketti tarjoaa kaksi rajapintaa. Ensimmäinen nojaa callback-tyyliseen ohjelmointiin, mutta käytämme toista tarjottua vaihtoehtoa: Promiseihin perustuvaa. Promise on objekti, joka edustaa asynkronisen toimenpiteen mahdollista onnistumista tai epäonnistumista (Mozilla Developer Network – Promise). Käyttämällä Promiseihin perustuvaa rajapintaa pääsemme käyttämään *async / await* -syntaksia. Funktio, jolla on async-avainsana palauttaa aina Promisen. Async-funktion sisällä voimme käyttää await-avainsanaa. Tällöin jäämme odottamaan asynkronisen funktion loppuunsuorittamista (Mozilla Developer Network – Async await). Näin voimme kirjoittaa asynkronista koodia, jota luetaan kuin synkronista koodia, ylhäältä alas.

Amqp.Nodea käyttäessä ensin muodostetaan palvelimelle yhteys eli kanava. Tämän jälkeen kanavalle voidaan rekisteröidä eri viestijonoille kuluttajia (*consumers)*. Kuluttajat ottavat vastaan funktion, joka hoitaa viestin käsittelyn. Kanavan

kautta voidaan myös luoda, muokata sekä poistaa kanavia. Lisäksi kanavan kautta eri viestijonoihin(*queue*) tai pörsseihin(*exchange*) voidaan lisätä viestejä. Pörssejä on erityyppisiä, mutta yleisin on *fanout*, joka jakaa viestin kaikkiin jonoihin, jotka ovat kiinni pörssissä.

Amqp.Node valittiin, koska se oli vaihtoehtoisista RabbitMQ:lle sopivista paketeista ylläpidetyin. Amqp.node on myös suositeltu paketti RabbitMQ:n tutoriaaleissa. Sen tarjoama rajapinta on siisti, tosin paketin päivitystahti ei ole vakuuttava ja nykyään uusi versio tulee keskimäärin kerran vuodessa. Bramqp-niminen paketti oli liian matalan tason abstraktio. Sen käyttäminen olisi vaatinut enemmän koodia ja osaamista RabbitMQ:sta (Bramqp). Muita vastaavia aktiivisia paketteja ei ole. Esimerkiksi muuten lupaava korkeamman abstraktion node-amqp ei ole aktiivisessa ylläpidossa (Node-amqp).

3.2.3 Knex

Knex on työkalu SQL-kyselykielen kutsujen muodostamista varten. Sen avulla tietokantakyselyitä ei tarvitse kirjoittaa tekstinä koodiin vaan voimme hyödyntää kirjaston tarjoamia funktioita kyselyn muodostamiseen. Funktioiden käyttäminen auttaa kirjoittamaan selkeämpää koodia, kun ketjutamme SQL-lauseiden eri osat. Tällöin SQL-kyselyssä käytettäviä funktiota ei injektoida SQL-kyselymerkkijonoon mukaan. Knexin käyttäminen oikein ehkäisee SQL-injektioiden riskiä, kun parametrit välitetään tietokanta-ajurille erikseen (Knexjs.org). Knex tarjoaa myös mahdollisuuden kirjoittaa SQL-kyselyn osia raakana. Tarjolla on myös tietokantaan yhdistäminen.

Samanlaista pakettia kuin Knex ei oikein löydy JavaScriptille. Vaihtoehtoina Knexin kaltaisille kyselynrakentajille on käyttää suoraan MariaDB:n JavaScriptille tarkoitettuja tietokanta-ajureita. Tietokanta-ajurit ovat kuitenkin matalan abstraktion rajapinnan kirjastoja, joten monia ominaisuuksia, kuten avointen tietokantayhteyksien säilömistä ja jakamista, joutuu itse asettamaan. Korkeamman abstraktiotason suuntaan Knexistä vaihtoehtona olisi ORMit. Esimerkiksi Bookshelf tai Sequalize. Tässä tapauksessa ORMit eivät tarjoa paljoa hyötyä, kun kyse on parista tietokantataulusta.

3.2.4 Jest

Projektin koodin testaamista varten tarvitaan työkalu. Työkalun tulisi olla mahdollisimman pitkälle määritelty valmiiksi. Työkalun tulisi toimia hyvin TypeScriptin kanssa, jotta testaaminen saadaan toimimaan ilman suurta konfigurointia. Valintaan myös vaikutti aiempi kokemus, joten parhaaksi valinnaksi osoittautui Jest. Työkalun ylläpidettävyyden kannalta asiaan vaikuttivat ongelmien määrä ja uusien versioiden julkaisutahti. Facebookin lisäksi Jestiä käyttävät Twitter, Spotify ja AirBnb, mikä myös vakuutti työkalun ylläpidon tasosta.

Jest on Facebookin kehittämä ohjelmointikehys testaamiseen, joka tarjoaa työkalut testien kirjoittamista ja suorittamista varten. Jest tarjoaa myös itse raportin siitä, kuinka paljon koodista suoritetaan testeissä. Asetukset Jestissä on yleisimpiä käyttötapauksia varten eikä siten sitä tarvitse alusta alkaen määritellä (Jest). Käyttötapauksessamme pitää kuitenkin Jestille luoda asetustiedosto, jossa määrittelemme käyttävämme TypeScriptiä ja Jestin tulee kääntää TypeScript-tiedostot ensin JavaScriptiksi.

Aiempien kokemusten perusteella yksi vaihtoehto oli Jasmine. Jest ja Jasmine ovat hyvin samanlaisia ja molemmat sopivat käyttötapaukseen. Jasmine ei kuitenkaan tarjoa esimerkiksi raportteja testien kattavuudesta. Jasminessa on Jestiin verrattuna vähemmän ominaisuuksia (Zaidman, V. 2019) Toinen vaihtoehtona Jestille oli Mocha. Mocha on erittäin joustava testikehys JavaScriptille. Mocha antaa käyttäjän valita mitä kirjastoja käyttää testikokonaisuuden muodostamiseksi. Kuitenkin tässä projektissa ei ole tarkoituksenmukaista hienosäätää testikirjastoja, joten Jest tarjoaa toimivan ratkaisun vähäisellä määrittelyllä.

3.2.5 Axios

Sähköposti-integraation monia rajapintakutsuja varten tarvitaan kirjasto, joka yksinkertaistaa kutsujen muodostamisen koodissa. Axios on avoimen lähdekoodin http-pyyntö -asiakaskirjasto. Axios toimii sekä selaimessa että Node.js-ympäristössä. Kirjasto tukee Promiseja, joita käytämme paremman luettavuuden saavut-

tamiseksi. Axios tukee myös JSON-datan automaattista muunnosta JavaScriptissä käytettävään muotoon, mikä helpottaa JSON-muodossa vastauksia palauttelevien rajapintojen kanssa kommunikointia (Axios).

Vaihtoehtoja Axiokselle on monia. Node.js tarjoaa omat standardikirjastonsa http- ja https-moduulit. Node.js:n omat standardikirjastot ovat hyvin lähellä Node.js:n omaa koodia ja ovat matalan tason abstraktioita. Ne soveltuvat ennemminkin kirjastojen tai moduulien kirjoittamiseen, mutta yksinkertaisten http-pyyntöjen toteuttamiseen niissä joutuu kirjoittamaan runsaasti koodia. Request-kirjasto tarjoaa hyvin kattavia toimintoja ja jakaa monen ominaisuuden Axioksen kanssa. Request ei kuitenkaan suoraan tue Promiseja vaan sitä varten pitää erikseen asentaa toinen paketti, request-promise. Axiokselle on myös vaihtoehtona node-fetch. Node-fetch tuo selaimista tutun fetch-rajapinnan Node.js:lle. Node-fetchissä kuitenkin joudumme muuttamaan JSON-datan sekä pyyntöä lähettäessä että vastauksen saapuessa.

Axios valittiin, koska sen tarjoama rajapinta on selkeä ja helppo käyttää. Axios tarjoaa vaihtoehdoista eniten ominaisuuksia. Axios tuki Promiseja suoraan, mikä on hyvä, koska näin uudessa projektissa halutaan käyttää niitä. Axioksesta on myös aiempaa kokemusta selainpuolen JavaScriptistä, joten sen käyttö oli jo testattu.

4 ARKKITEHTUURI

Tässä luvussa esitellään mistä eri komponenteista uuden sähköposti-integraation arkkitehtuuri muodostuu. Arkkitehtuurikaavio sisältää kuvattuna komponenttien väliset vuorovaikutussuhteet. Lopuksi arvioidaan miten arkkitehtuuri mahdollistaa uusien vaatimusten saavuttamisen.

4.1 Arkkitehtuurin komponentit

Sähköposti-integraatio on arkkitehtuurin keskeisin komponentti, keskeinen palvelu, joka yhdistää muut komponentit yhteen. Sähköposti-integraatio pyörii Docker-konttina Kubernetes-ympäristössä. Sähköposti-integraation tehtävänä on vastata kaikista sähköposti-integraatioon liittyvistä palveluista ja sähköposti-integraation sisäisestä logiikasta. Palvelu muodostaa yhteyden muihin arkkitehtuurin komponentteihin ja on siten ainoa komponentti, joka on yhteydessä kaikkiin muihin palveluihin (Liite 1).

Microsoftin Outlook-palvelu toimii sähköpostiviestien talletuksesta vastaavana komponenttina. Palvelu sijaitsee Microsoftin tarjoamana pilvipalveluna Kubernetes-ympäristön ulkopuolella eikä siten ole kehittäjien hallittavissa tai muokattavissa. Outlook Rest API v2.0-rajapintaan tunnistaudutaan Azure AD v2-tunnistaumisella. Tätä tunnistautumista ei voi toteuttaa helposti koneellisesti, joten tämän kirjautumisen suorittaa kehittäjä, kun sähköposti-integraatio otetaan käyttöön. Rajapinnasta haetaan sähköpostilaatikkoon saapuneet sähköpostiviestit sekä niiden liitetiedostot. Rajapinnan kautta voidaan myös lähettää asiakkaalle vahvistusviesti.

Viestien käsittelyyn tarvittavaa viestijonoa varten on Kubernetes-ympäristössä RabbitMQ-palvelin. Tässä käyttötapauksessa riittää yksi RabbitMQ-palvelin eikä tarvita usean palvelimen parvea. Palvelimelle yhdistetään AMQP-protokollalla ja tunnistaumiseen käytetään käyttäjätunnusta ja salasanaa. Palvelimelle yhdistämisen jälkeen voidaan viestien lukemisen lisäksi lisätä viestejä viestijonoihin, luoda tai poistaa jonoja sekä muokata jonojen asetuksia.

Kubernetes-ympäristössä pyörii MariaDB-tietokantapalvelin. Tietokantaa käytetään tallentamaan jo käsitellyt viestit, koska tällaista tietoa ei voi tallentaa viestijonoon pysyvästi. MariaDB-tietokantaan riittää taulu, missä on rivi per käsitelty sähköpostiviesti, joten normaali asennus riittää. Lisäksi tietokantaan halutaan tallentaa päivityssuojausavain, jolla voidaan hakea uusi pääsysuojausavain Outlookin rajapinnan käyttöä varten.

Asiakaspalvelun rajapinta on toimeksiantajan sisäinen rajapinta, joka pyörii Kubernetes-ympäristössä. Rajapinta on REST-rajapinta ja käyttää OAuth 2 -todennusta käyttäjän tunnistamiseen. Sähköposti-integraatiota varten luodaan ympäristöön käyttäjä sähköposti-integraatiolle. Sähköposti-integraatio käyttää tätä käyttäjää rajapintaan tunnistautumiseen ja suorittaakseen sähköposti-integraation mukaiset toimenpiteet.

4.2 Arkkitehtuurikaavio

Liitteen yksi arkkitehtuurikaaviosta löytyvät arkkitehtuurin isoimmat konseptit. Kubernetes-laatikko kuvaa sitä, mitä kaikkea pyörii Kubernetes-ympäristön sisällä. Ulkoisena palveluna tässä tapauksessa on Outlook-rajapinta. Tietokannan virtuaalikone on jaettu eri Kubernetes-ympäristöjen kesken.

Kubernetes-ympäristön sisältä löytyy sähköposti-integraatio, joka pyörii Docker-konttina. Niin ikään RabbitMQ ja asiakaspalvelun rajapinta pyörivät konteissa. Sähköposti-integraatio toimii yhdistävänä tekijänä eri komponenttien välillä ja esimerkiksi asiakaspalvelu ja RabbitMQ eivät keskustele keskenään.

Sähköposti-integraation sisällä koodi on jaettu viiteen osa-alueeseen. Sisäiseen logiikkaan kuuluvat integraation pitäminen tietoisena muiden palveluiden toiminnasta ja käynnistyessään sähköposti-integraatio lähettää palveluiden kirjautumispyynnöt. Tietokannalle sekä molemmille rajapinnoille on omat luokkansa. Viestijonon käsittelyä varten on apufunktioita, jotka helpottavat muun muassa viestien siirtelyä viestijonojen välillä.

4.3 Arkkitehtuurin sopiminen vaatimuksiin

Aiempien versioiden perusteella uudelle sähköposti-integraatiolle oli asetettu useita vaatimuksia. Projektin koodin tulisi olla paremmin organisoitua ja palvelulta tulisi löytyä selkeä arkkitehtuuri. Uuden sähköposti-integraation tulisi lyhentää sähköpostiviestien integraation kestoa ja viivettä. Sähköposti-integraation tulisi toimia jatkuvana palveluna. Virheiden käsittelyn ja hallinnan tulisi olla kattavampaa.

Paremmin organisoitu koodi saavutetaan, kun viestijonon purkaminen ja sinne viestien lisääminen on eriytetty toisistaan. Jokaiselle osa-alueelle on määritelty oma funktio, joka ottaa viestin tiedot parametrina ja palauttaa vastauksen siitä mitä viestille pitäisi tehdä. RabbitMQ:n, Outlook-rajapinnan sekä Asiakaspalvelurajapinnan kutsut on jaettu omiin luokkiinsa. Luokat käyttävät tarvittavia tietoja sisäisesti. Myös tietokannan funktiot on eriytetty muusta koodista omaksi luokakseen. Funktioissa on pyritty single responsibility -periaatteeseen eli funktio tekee vain yhden asian ja palauttaa tuloksen.

RabbitMQ-palvelimen lisääminen ja viestijonojen käyttäminen mahdollistaa uudelleenyritykset pyynnöissä. Uusien yritysten mahdollisuuksien lisäksi viestijonot mahdollistavat myös lyhyemmät käsittelyajat. RabbitMQ voi purkaa viestijonoja monta viestiä kerrallaan ja useampaa osa-aluetta kerrallaan. Tällöin sekä yksittäisen sähköpostiviestin osa-alueiden että useamman sähköpostiviestin purkaminen on nopeampaa. Tämä on mahdollista, koska suurin osa viestien käsittelyn suoritusajasta ohjelma odottaa vastauksia ulkoisista palveluista.

Virheiden käsittely paranee muutamalla muutoksella. Ensinnäkin satunnaiset yhteysvirheet käsitellään oikein, kun viestijono lisää epäonnistuneet viestit takaisin käsiteltäviksi automaattisesti. Viestien käsittelyfunktiot palauttavat tiedon siitä mitä viestille pitää tehdä: palauttaa takaisin jonoon, poistaa onnistuneesti suoritettuna tai hylätä johtuen tietojen virheellisyydestä. Testaamisella saavutetaan monia vaatimuksia. Jotta testaaminen olisi sujuvaa, pitää koodin olla eriytettyä ja hyvin organisoitua. Yksinkertaisia, single responsibility -periaatteella olevia funktioita on helpompi testata. Viestien osa-alueiden käsittelyn jakaminen omiin funktioihin auttaa kirjoittamaan testejä, jotka takaavat paremman virheiden hallinnan.

Entisen virtuaalikoneella olevan PHP-komentosarjana suoritetun cron-tehtävän muuttaminen Docker-kontissa ajettavaksi jatkuvaksi Node.js-prosessiksi poistaa ongelmat, jotka liittyivät edellisen sähköposti-integraation seurantaan ja lokitietojen tallentamiseen. Palvelun lokitiedot voidaan Kubernetes-ympäristössä tallentaa osaksi parempaa lokitiedostojen hallintajärjestelmää.

5 SÄHKÖPOSTI-INTEGRAATION TOIMINTA KÄYTÄNNÖSSÄ

Tässä luvussa keskitytään teknisesti kuvaamaan sähköposti-integraation toimintaa kahdessa eri käyttötapauksessa: uuden sähköpostin saapuessa ja asiakkaan vastatessa sähköpostiin.

5.1 Sähköposti-integraation käynnistyminen

Sähköposti-integraation käynnistyessä muodostetaan asetusten perusteella yhteydet tietokantaan, RabbitMQ-palvelimeen, asiakaspalvelurajapintaan sekä Outlookin rajapintaan. Lisäksi jokaiselle edellä mainitulle riippuvuudelle on määritelty yksinkertainen testi, jonka tarkoitus on varmistaa, että yhteyden lisäksi myös riippuvuus toimii. Esimerkiksi tietokannasta löytyy oikeanlainen taulu ja sieltä saa tiedot haettua (Liite 2). Mikäli tarkastukset epäonnistuvat, niin sähköposti-integraatio päättää itsensä. Kubernetes pitää huolen siitä, että Docker-kontti käynnistetään muutaman kerran uudestaan automaattisesti, jotta voidaan sulkea pois satunnaiset yhteysongelmat. Mikäli muutaman käynnistyksen jälkeen sähköposti-integraatio ei ole saanut muodostettua yhteyttä riippuvuuksiin, niin ongelma on todennäköisesti asetuksissa tai ongelma ei johdu sähköposti-integraatiosta.

Kun tarkastukset on tehty, sähköposti-integraatio käynnistää ajoittaisen sähköpostiviestien hakemisen Outlookin rajapinnasta. Lisäksi Outlookin kanssa muodostetaan Push Notification -tilaus. Uuden viestin saapuessa Outlook ilmoittaa suoraan sähköposti-integraation julkiseen porttiin viestistä, jolloin sähköpostiviestit voidaan hakea heti. Ajoittainen sähköpostiviestien hakeminen täyttää tämän toiminnallisuuden ulkopuolella saapuneiden viestien hakemisen sähköpostilaatikosta. Tällaisia tilanteita syntyy esimerkiksi, jos tilaus vanhenee tai kun sähköposti saapuu sähköposti-integraation ollessa pois päältä. Ajoittain sähköpostiviestien hakeminen toimii ikään kuin varmistuksena sille, että kaikki viestit luetaan, vaikka niistä lähtenyttä ilmoitusta ei olisikaan vastaanotettu.

5.2 Uusi sähköposti

Asiakas lähettää sähköpostiviestin asiakaspalvelun sähköpostilaatikkoon. Mikäli sähköposti-integraatio on Push Notification-tilaussuhteessa Outlook-rajapinnan kanssa, saa sähköposti-integraatio ilmoituksen uudesta viestistä muutaman sekunnin sisällä. Jos näin ei ole, niin sähköposti-integraatio huomaa seuraavan ajoittaisen tarkistuksen aikana, että sähköpostilaatikoista löytyy viestejä tunnisteilla, joita ei ole vielä lisätty tietokantaan.

Sähköposti-integraatio luo viestistä uuden asiakaspalvelutiketin. Mikäli tiketin luonti onnistuu, niin viesti lähetetään uusiin viestijonoihin, jotka käsittelevät viestin osa-alueet. Jos tiketin luonti epäonnistuu, sitä yritetään uudelleen seuraavan ajoittaisen suorituksen aikana. Sähköpostiviestien tunnisteet tallennetaan tietokantaan, jotta sähköpostiviestiä ei käsitellä toista kertaa.

Seuraavat kuvattavat toimenpiteet tapahtuvat mahdollisesti samanaikaisesti. Sähköposti-integraatio lisää sähköpostiviestin sisällön tiketille kommenttina. Tiketille liitetään asiakkaan tiedot asiakaspalvelun rajapintaa hyödyntäen, yhdistävä tekijänä on sähköpostiosoite. Mikäli sähköpostiosoitteella ei löydy asiakastietoja, luodaan uusi asiakaskortti tietokantaan ja liitetään se tiketille. Harvinaisessa tapauksessa, jossa samalla sähköpostilla löytyy useampia asiakkaita, kirjataan tiketille huomio tilanteesta, jotta asiakaspalvelijat voivat perehtyä asiaan tarkemmin ja tehdä ratkaisun siitä mikä asiakas tikettiin liitetään. Sähköpostiviestin mukana tulleet liitetiedostot lähetetään liitetiedostopalvelimelle ja lisätään tikettiin.

5.3 Vastaus sähköpostiin

Automaattinen vastausviesti sisältää tiedon, että palaute tai reklamaatio on nyt asiakaspalveluun kirjattuna ja viestiin vastaamalla voi antaa lisätietoja. Mikäli ympäristössä on käytössä automaattinen vastausviesti asiakkaalle, se lähetetään asiakkaalle heti tiketin integroinnin jälkeen. Otsikossa on tunniste, jolla sähköposti-integraatio osaa liittää vastauksen oikeaan tikettiin. Mikäli automaattista

vastausviestiä ei ole ympäristössä, asiakaspalvelijan lähettää sähköpostina vastauksen, johon asiakas voi vastata. Otsikossa olevan tunnisteen avulla voidaan yhdistää asiakkaan lähettämä sähköpostivastaus oikeaan tikettiin. Tämä kuitenkin vaatii, että sähköpostin otsikon tunniste on ehjä eikä asiakas ole esimerkiksi poistanut sitä. Mikäli sähköpostivastauksesta ei tunnisteta oikeaa tikettiä, voi asiakaspalvelija tarkastella asiakkaan muita tikettejä, mistä näkee helposti oikean tiketin.

Saapuneesta sähköpostiviestistä tallennetaan kommentti keskusteluun. Mikäli sähköpostiviestissä oli uusia liitteitä, ne lisätään tiketille. Jos sähköpostiviesti tuli sen jälkeen, kun asiakaspalvelija on vaihtanut tiketin tilan valmiiksi, niin tiketin tila palautetaan uudeksi, jolloin se palautuu asiakaspalvelijoiden jonoon käsiteltäväksi aikanaan. Tapauksissa, joissa asiakaspalvelutilanne on alkanut esimerkiksi chat-keskustelun tai puhelimen välityksellä, vaihdetaan tiketin kanavaksi sähköposti, kun asiakas vastaa asiakaspalvelijan lähettämään sähköpostiviestiin.

6 VIRHETILANTEIDEN HALLINTA

Tässä luvussa tarkastellaan minkälaisiin virhetilanteisiin sähköposti-integraatio voi törmätä käynnistyessään ja ollessaan toiminnassaan. Virhetilanteet on jaettu riippuvaisuuksien käyttökatkoksiin, epäonnistuneisiin pyyntöihin sekä käyttäjien aiheuttamiin asetusvirheisiin.

6.1 Käyttökatkokset

Käyttökatkokset riippuvaisuuksiin voidaan jakaa kahteen kategoriaan. Jaottelun kriteeri on, että voiko yrityksen kehittäjät käyttökatkokseen vaikuttaa vai ei. Ulkoisissa palveluissa, kuten Outlookin REST-rajapinnassa, eivät kehittäjät voi vaikuttaa sen käyttökatkoksiin. Näissä tapauksissa sähköposti-integraatio on alhaalla niin kauan kuin Outlook on alhaalla. Käyttökatkokset Outlookissa ovat kuitenkin harvassa ja lyhyitä, joten niistä ei ole oleellista haittaa. Käyttökatkosten kestäessä keskimäärin muutamia minuutteja, eivät ne ole kriittisiä tässä käyttötapauksessa. Ulkoisissa palveluissa on yleensä korkea saatavuus.

Käyttökatkokset, joihin kehittäjät voivat vaikuttaa ovat niitä, mitkä eivät korjaannu yleensä itsestään. Näihin käyttökatkoksiin lasketaan RabbitMQ-palvelimen ongelmat. Ongelma RabbitMQ:ssä voi olla muodostaa yhteys, kun sähköposti-integraatiota käynnistetään. Ongelma myös muodostuu, jos RabbitMQ-palvelin lakkaa toimimasta kesken käytön. Käynnistyksessä ongelma ratkeaa sillä, kun sähköposti-integraatio päättää itsensä. Mikäli palvelu ei enää toivu seuraavilla yrityksillä, virhetilanne välittyy kehittäjille Kubernetekseen määriteltyjen valvontojen perusteella. Ajonaikainen virhetilanne keskeyttää sähköposti-integraation toiminnan kokonaan, koska viestejä ei voi käsitellä, lisätä tai poistaa viestijonon ollessa poissa käytöstä. Asiakaspalvelun rajapinnan muuttuessa toimimattomaksi, ei viestijonoa voida purkaa, mutta viestijonoon voidaan lisätä viestejä, mikäli tietokanta, RabbitMQ sekä Outlook toimivat normaalisti. Tietokannan käyttökatkoksen aikana voidaan purkaa viestijonoa, mutta yleensä asiakaspalvelurajapinta ja sähköposti-integraatio jakavat saman tietokannan. Viestijonon purkamisesta ei siis ole hyötyä tietokantaongelmien aikana.

6.2 Epäonnistuneet pyynnöt

Aina http-pyynnöt eivät onnistu palveluiden välillä. Syitä on monia alkaen satunnaisista yhteysongelmista jatkuen aina palveluiden hetkellisiin käyttökatkoksiin. Sähköposti-integraatiossa on erilaisia pyyntöjä, mitkä voivat epäonnistua. Perinteiset http-pyynnöt voivat epäonnistua verkkoyhteyssyistä. Rajapintojen kutsut voivat epäonnistua niin ikään johtuen joko rajapinnasta (5XX-koodit) tai pyynnön muodosta (4XX-koodit). Tietokantoihin liittyvät pyynnöt voivat myös epäonnistua edellisistä syistä. Sähköpostiviestin tunniste voi olla jo kirjattuna tietokantaan.

Tärkeintä on kuitenkin, että virheet tunnistetaan ennalta sekä mitä niille tulisi tehdä. Epäonnistuneet pyynnöt voidaan jakaa kahteen kategoriaan: vika on pyynnössä tai vika on muissa tekijöissä.

Vian ollessa ulkoisessa palvelussa tai konfiguraatiossa, palvelu yleensä palauttaa 500-tilannekoodin vastauksessa pyynnölle. Axios-kirjaston avulla voidaan päätellä helpommin, milloin viesti on epäonnistunut. Esimerkiksi sen takia, että palvelun osoite on konfiguroitu väärin tai palvelun on alhaalla. Tällaisissa tilanteissa järkevin tapaus on odottaa ja yrittää uudelleen tarpeeksi kauan. Mikäli tilanne ei ratkea itsestään jonkin ajan kuluessa, niin viesti on hylättävä ja siitä on raportoitava eteenpäin. Raportointi eteenpäin on tärkeää, jotta voidaan tunnistaa tilanteet, missä vika on ollut pyynnössä eikä ulkoisessa palvelussa.

Vika voi myös olla itse pyynnössä. Pyynnön osoite voi olla väärä tai muuttunut. Myös pyynnön sisältämä tieto voi olla virheellistä. Tämän tilanteen tunnistettuamme, oikea ratkaisu on hylätä pyyntö ja kirjata siitä virheraportti. Tällöin virheellinen pyyntö ei jää pyörimään järjestelmään vaan kehittäjät voivat tutkia ja korjata ongelmatilanteen nopeasti.

Kuitenkin virhetilanteissa nämä kaksi erityyppistä virhettä tulee erottaa toisistaan. RabbitMQ:n tapauksessa voimme yrittää saman viestin käsittelyä sen onnistumiseen asti. Tämä on kuitenkin resurssien tuhlaamista, joten lienee järkevämpää

rajoittaa kuinka monta kertaa viestit voivat epäonnistua. Tämän rajan tultua vastaan, viestit voidaan siirtää omaan jonoonsa. Voimme seurata viestien epäonnistumista luomalla järjestelmän, jossa sama viesti palautetaan käsittelyyn, mutta mukaan liitetään tieto siitä, kuinka monta kertaa viesti on epäonnistunut. Rajan tai viestin elinaikarajoitteen tullessa vastaan, siirretään viesti dead-letter-exchangeen. Dead-letter-exchange on erillinen pörssi, joka voidaan määritellä jonolle paikaksi, minne hylätyt viestit ohjataan (RabbitMQ). Virheraportteja varten voimme tutkia määriteltyä dead-letter-exchangea.

6.3 Asetusvirheet

Asetusvirheet ovat sähköposti-integraation asetusten konfigurointia väärin tai vääränlaisiksi, jolloin sähköposti-integraation toiminta ei ole toivottua. Asetusvirheet voidaan luokitella teknisiin virheisiin sekä prosessivirheisiin. Prosessivirheisiin lasketaan kaikki, jossa asetukset eivät vastaa toivottua lopputulosta.

Tekniset virheet johtavat sähköposti-integraation toiminnan häiriöihin esimerkiksi sähköposti-integraation toimimattomuuteen. Sähköposti-integraation ympäristömuuttujiin tallennetaan tiedot, joilla luodaan yhteys tietokantaan ja RabbitMQ-palvelimelle. Asiakaspalvelu- sekä Outlook-rajapintoja varten tallennetaan tiedot tunnistautumista varten. Näiden tietojen virheellisyys estää luonnollisesti sähköposti-integraation toiminnan, koska tällaiset virheelliset yhteystiedot estävät palvelun käynnistymisen, koska palvelun käynnistämisen yhteydessä myös ulkoisten palveluiden toiminta tarkistetaan.

7 TUOTOKSEN KÄYTTÖ

Tässä luvussa selostetaan, miten lopputuotosta käytetään kehittäjän näkökulmasta. Ensiksi selostus käyttöönotosta, jonka jälkeen kerrotaan lopputuotoksen jatkokehittämisestä ja testien ajamisesta sekä testitulosten tulkitsemisesta.

7.1 Käyttöönotto

Lopputuotoksen käyttöönotto tapahtuu ensiksi rekisteröimällä sovellus Microsoftin Azure Portal -palveluun. Palveluun tulee luoda sovellus. Sovellukselle luodaan salainen avain. Sovelluksen tunniste ja salainen avain lisätään lopputuotoksen konfiguraatiotiedostoon.

Jotta sovellus pääsee käsiksi Microsoft Outlookin sähköposteihin, tulee käyttäjän kirjautua kertaalleen sisään sähköpostilaatikon sähköpostiosoitteella sekä salasanalla lopputuotoksen kirjautumissivulta. Käyttäjän tulee hyväksyä sovelluksen vaatimat käyttöoikeudet sähköpostilaatikon sisältöihin (Liite 3). Tällöin käyttäjä ohjataan sovelluksen asetettuun osoitteeseen mukanaan pääsysuojausavain, jolla rajapintaa voi käyttää sekä päivityssuojausavain, jolla voi hakea uuden pääsysuojausavaimen vanhentuneen tilalle. Sähköposti-integraatio ottaa saadut suojausavaimet vastaan ja tallentaa ne tietokantaan. Lisäksi tulee varmistaa, että tietokantayhteys on määritelty oikein ja tietokanta sisältää tarvittavat tietokantataulut.

7.2 Suorittaminen ja kehittäminen

Sähköposti-integraatiota ajetaan suorittamalla komentorivillä komento *yarn*, joka lataa ja asentaa projektin riippuvuudet. Asennuksen valmistuttua voidaan suorittaa komento *yarn build*, joka tarkistaa ja kääntää TypeScriptin JavaScriptiksi. Käännösvaiheen jälkeen sähköposti-integraatio käynnistetään ajamalla komennon *yarn start*. Mikäli kehittäessä halutaan, että tiedostoja muokattaessa uusin versio käännetään ja suoritetaan, voidaan käyttää komentoa *yarn watch*. Tämä

komento tarkkailee tiedostoja nodemon-paketin avulla sekä kääntää *tsc*-työkalulla TypeScript-tiedostot JavaScript-tiedostoiksi, kun se havaitsee niissä muutoksia.

7.3 Testaaminen

Sähköposti-integraation testaaminen on tärkeää. Ensinnäkin kyseessä on integraatio, jonka toisessa päässä on julkinen sähköpostiosoite. Tämä asettaa vaatimukset sille, että koodia testataan tietoturva huomioiden. Käyttäjät voivat vaikuttaa muun muassa sähköpostiotsikkoihin, viestien sisältöihin sekä liitetiedostoihin. Tätä varten tarvitaan testejä, jotta voidaan varmistua, että nämä kentät siistitään haitallisesta syötteestä oikein.

Toisekseen testaamisella pitää pyrkiä vähentämään tilanteita, missä sähköposti-integraatio käsittelee väärin sähköposteja. Sähköposti-integraation tapahtumat eivät näy asiakaspalvelijalle. Asiakaspalvelija olettaa, että hänen lukemansa tiketti sisältää asiakkaan koko sähköpostin, vaikka todellisuudessa siitä voi olla suodatettu ja siistitty asioita pois. Siksi onkin hyvä testata jo etukäteen, mitä asioita suodattuu viesteistä pois.

Viestijonojen käyttäminen mahdollistaa eri osa-alueiden suorittamisen erikseen ja uudelleenyrittämisen virhetilanteissa. Kuitenkin ongelmana voi olla tilanteet, joissa viestiä ei koskaan voida suorittaa onnistuneesti. Tällaisia tilanteita varten testeistä on hyötyä, koska voidaan testata, että viesti osataan käsitellä oikein myös tilanteessa, jossa vika on viestissä itsessään.

7.3.1 Mitä testataan

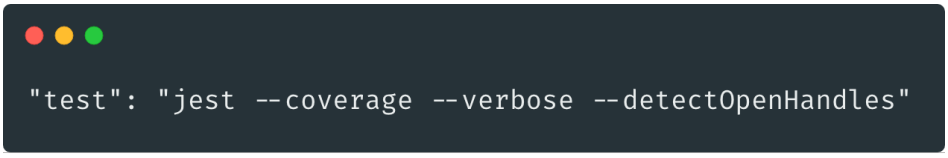
E2E eli End-to-end -testaaminen on hankalasti toteuttavissa tällaisessa projektissa, joka sisältää ulkoisia palveluita ja paljon eri komponentteja arkkitehtuurissaan. E2E-testien sijaan on tuottavampaa keskittyä yksikkötesteihin ja integraatiotesteihin.

Yksikkötesteillä tarkoitetaan *yksikön* eli yksittäisen kokonaisuuden testaamista. Yksikkö on yleensä funktio, mutta yksikkötesteiksi voidaan myös laskea useampi funktio, mikäli niillä ei ole ulkoisia riippuvaisuuksia. Tämän projektin puitteissa yksikkötestattavia asioita esimerkiksi ovat sisällön suodatukset, sähköpostiviestien tunnisteiden tarkistukset sekä yksittäisten api-kutsujen oikeanlainen käsittely.

Integraatiotesteillä pyritään testaamaan, että ohjelman eri komponentit toimivat yhteen. Jäljitelmillä eli mockeilla voidaan testeissä jäljitellä riippuvaisuuksia ja palauttaa hallitusti vastauksia. Tällöin ei tarvitse tallentaa testitikettejä tietokantaan vaan voimme luoda tietokantakomponentista jäljitelmän, joka vain palauttaa saman vastauksen mitä tietokanta palauttaisi viestin onnistuessa. Esimerkiksi voitaisiin testata, että Outlook-jäljitelmän palauttamat tekaistut viestit yritettäisiin lisätä tietokantaan oikein ja ne lisätään oikeisiin viestijonoihin. Tällöin tehtäisiin testejä varten jäljitelmät Outlook-rajapinnasta, Knexistä sekä amqp.nodesta.

7.3.2 Testien ajaminen ja tulosten tulkitseminen

Osana sähköposti-integraation kehitystä testejä ajetaan varmistamaan, että aiemmat toiminnallisuudet eivät ole muuttunut vahingossa. Mikäli toiminnallisuudet ovat muuttuneet tarkoituksenmukaisesti, niin testit päivitetään vastaamaan haluttua lopputulosta. Testit suoritetaan komennolla *yarn test*, joka tulostaa tulokset (Kuva 1).



Kuva 1: Yarn test -komento, jolla suoritetaan testipatteristo, purettuna

Testaamisen alettua Jest listaa kaikki testitiedostot ja testit nimineen. Testien kohdalla on tieto, onko testi kesken, epäonnistunut vai onnistunut. Kuvassa 2 on esimerkki yhdestä testiajosta.

```
PASS __tests__/utilities/utils.unit.test.ts
  Utilities for API
    ✓ should detect identifier in correct format {numbers}{alphanumeric_hash} (4ms)
    ✓ should not detect nonexisting identifier in title
    ✓ should not detect identifier without ticket id and hash
    ✓ should not detect identifier if it has over 2 alphanumerical characters
    ✓ should detect identifier having number (1ms)
    ✓ should detect identifier having only numbers
```

Kuva 2: Testipatteriston tulokset tulostettuna komentoriville

Kuvassa kolme on testien alapuolella näkyvä raportti nykyisen koodin testikattavuudesta. Testikattavuus on eritelty koodiriveihin, funktioihin, haaroihin sekä lausekkeisiin. Raportissa on erikseen määriteltynä edellä mainitut kattavuudet tiedostoa kohden, jonka lisäksi raportti kertoo mitkä rivit ovat testikattavuuden ulkopuolella.

```
---------------|----------|----------|----------|----------|-------------------|
File           |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
---------------|----------|----------|----------|----------|-------------------|
All files      |    22.31 |    14.29 |    13.33 |    23.08 |                   |
 src           |       75 |      100 |        0 |       75 |                   |
  logging.ts   |       75 |      100 |        0 |       75 |                 4 |
 src/api       |    11.54 |    14.29 |        4 |       12 |                   |
  index.ts     |     4.76 |        0 |        0 |     4.94 | ... 53,156,157,158 |
  util.ts      |       40 |    33.33 |       25 |    42.11 | ... 27,28,29,30,31 |
 src/utilities |    92.31 |      100 |       75 |    92.31 |                   |
  utils.ts     |    92.31 |      100 |       75 |    92.31 |                18 |
---------------|----------|----------|----------|----------|-------------------|
```

Kuva 3: Raportti testipatteriston kattavuudesta koodissa

8 POHDINTA

Opinnäytetyön lopputuotos oli onnistunut. Sähköposti-integraation uuden version suunnittelussa ja kehittämisessä ei törmätty kehitystä haittaaviin esteisiin. Teknologia- ja työkaluvalinnat osoittautuivat kehityksen aikana toimiviksi ratkaisuiksi. Uudesta sähköposti-integraatiosta ei kuitenkaan ole tässä vaiheessa vielä tarpeeksi kokemusta tuotantokäytössä, joten on liian varhaista vetää johtopäätöksiä millaisia ongelmia voi esiintyä ajan myötä tai kuinka usein.

Uuden sähköposti-integraation version toteuttaminen uudella arkkitehtuurilla on mahdollistanut monia uusia ominaisuuksia sähköposti-integraatioon. TypeScript on tuonut helppoutta ja varmuutta sen kehittämiseen. Myös pienikin määrä testejä perustoiminnallisuuksiin on auttanut kehittämään sähköposti-integraatiota varmemmaksi toiminnaltaan ja kehityksen aikana näkee testien suoriutuvan. Sähköposti-integraatio integroi viestit aiempaa nopeammin, koska useita sähköpostiviestejä voidaan käsitellä yhtäaikaisesti.

Jatkokehittämisen kannalta testikattavuutta tulisi lisätä, jotta virhetilanteet pystyttäisiin hallita paremmin. Testikattavuudella on suora yhteys sähköposti-integraation ylläpidettävyyteen. Lisäksi TypeScriptin tuomaa tyypitystä on syytä käyttää laajalti jatkossakin. Viestijonot tuovat mahdollisuuden yrittää sähköposti-integraation suorittamia osa-alueita uudelleen, mutta virhetilanteissa voivat aiheuttaa ongelmia. Tällaisia tapauksia voivat olla kolmannen osapuolen asiakasintegraatiot, jotka saattavat olla useitakin päiviä alhaalla.

Aihe, jota voisi tutkia lisää opinnäytetyöhön liittyen on RabbitMQ. Koska sähköposti-integraatio käyttää viestijonoja, RabbitMQ:n ja yleisesti viestijonojen syvempi osaaminen auttaisi monessa asiassa. Ainakin RabbitMQ:n suorituskykyä, asetuksia sekä virhetilanteiden hallintaan tulisi vielä tutkia lisää. Lisäksi voidaan tutkia mahdollisuutta tukea eri datalähteitä ja rajapintoja. Tällaisia ovat esimerkiksi Googlen Gmailin rajapinta tai sähköpostien lukemiseen tarkoitettua IMAP-protokolla.

LÄHTEET

10 reasons to migrate to MariaDB (if still using MySQL). 2015. Luettu 12.04.2019 https://seravo.fi/2015/10-reasons-to-migrate-to-mariadb-if-still-using-mysql

Amazon SQS. Luettu 12.02.2019 https://aws.amazon.com/sqs/

Amqp.node. 2019. README.md. Luettu 13.04.2019 https://github.com/squaremo/amqp.node

Async await – Mozilla Developer Network. Luettu 18.04.2019 https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

Axios. 2019. README.md Luettu 18.04.2019 https://github.com/axios/axios

Bramqp. 2018. README.md. Luettu 13.04.2019 https://github.com/bakkerthehacker/bramqp

Dimitrov, A. Why should you migrate from MySQL to MariaDB? 2014. Luettu 21.03.2019 https://mariadb.com/resources/blog/why-should-you-migrate-from-mysql-to-mariadb/

Greif, S; Benitte, R; Rambeau M. State of JavaScript 2018. Luettu 13.04.2019 https://2018.stateofjs.com/

Jest. 2019. Luettu 21.04.2019. https://jestjs.io/

Johansson, L. Why a database is not always the right tool for a queue based system. 2015. Luettu 21.03.2019 https://www.cloudamqp.com/blog/2015-11-23-why-is-a-database-not-the-right-tool-for-a-queue-based-system.html

Knexjs.org. 2019. Luettu 18.04.2019. https://knexjs.org/#Raw

Knex. 2018. README.md Luettu 18.04.2019 https://github.com/tgriesser/knex/

McKenzie, S; Nakawaza, C; Kyle, J. 2016. Yarn: A new package manager for JavaScript. Luettu 22.04.2019. https://code.fb.com/web/yarn-a-new-package-manager-for-javascript/

Node-amqp. 2018. README.md. Luettu 13.04.2019 https://github.com/postwait/node-amqp

Nodejs.org. Event loop, timers and process.nextTick(). Luettu 21.04.2019 https://nodejs.org/es/docs/guides/event-loop-timers-and-nexttick/

Node.js Foundation Advances Platform with More Than Three Million Users https://nodejs.org/en/blog/announcements/foundation-advances-growth/

Promise – Mozilla Developer Network. Luettu 13.04.2019 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

RabbitMQ.com. Luettu 19.03.2019 https://www.rabbitmq.com/

Stackoverflow.com 2012. Is there any reason to use RabbitMQ over Kafka https://stackoverflow.com/questions/42151544/is-there-any-reason-to-use-rabbitmq-over-kafka

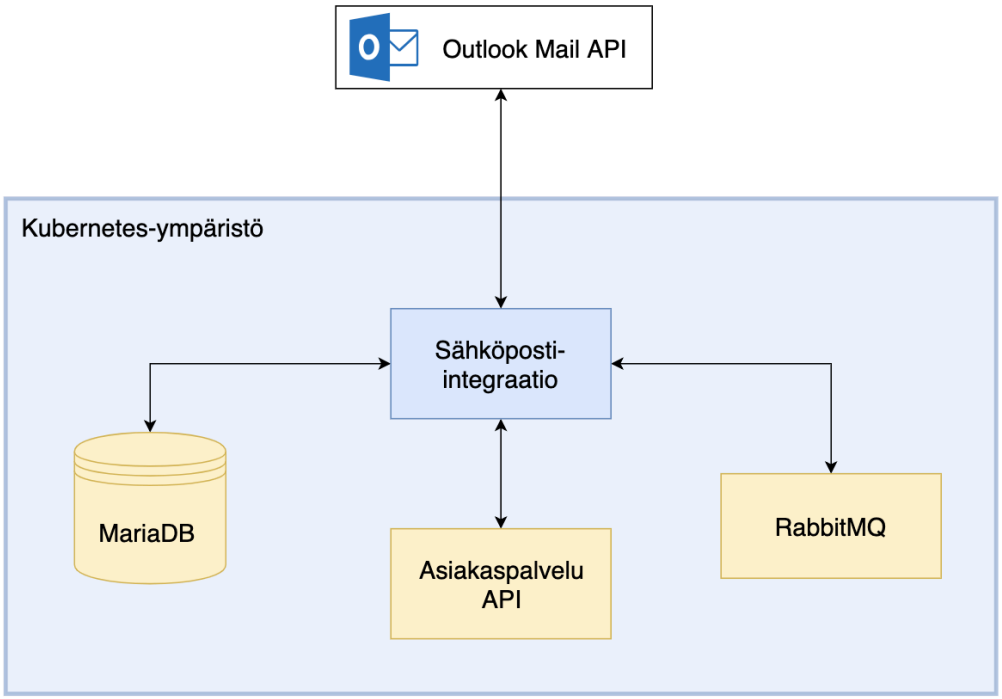
Stackoverflow.com 2012. Why do we need message brokers like RabbitMQ over a database like PostgreSQL? Luettu 14.04.2019 https://stackoverflow.com/questions/13005410/why-do-we-need-message-brokers-like-rabbitmq-over-a-database-like-postgresql

Woods, E. PHP: a fractal of bad design. 2012. Luettu 13.04.2019 https://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/

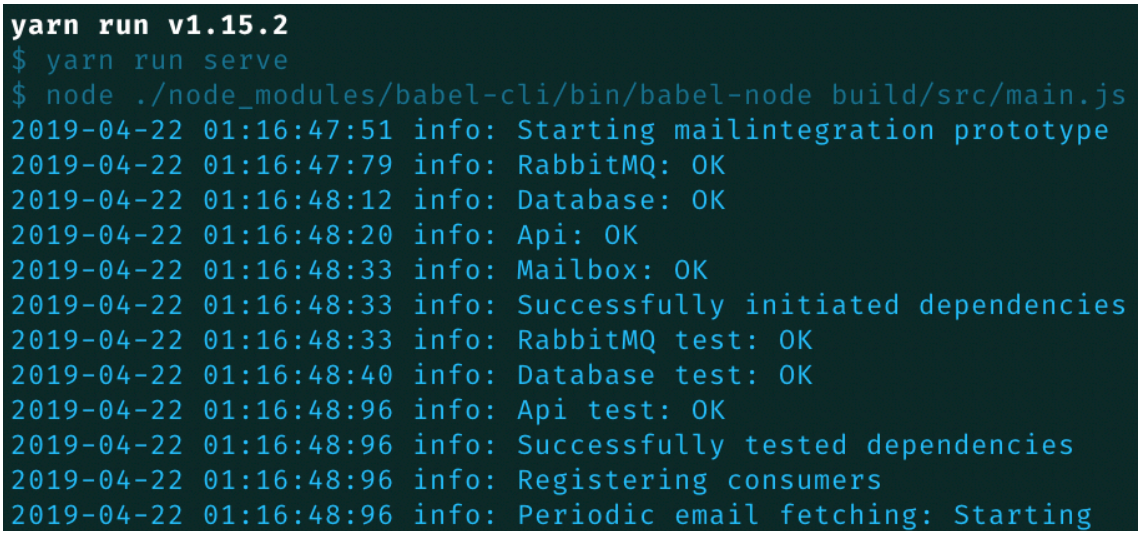
Zaidman, V. An Overview of JavaScript Testing in 2019. 2019. Luettu 01.03.2019 https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2019-264e19514d0a

LIITTEET

Liite 1. Arkkitehtuurikaavio



Liite 2. Sähköposti-integraation käynnistyksen aikaiset testit



Liite 3: Oikeuksien myöntäminen sähköposti-integraatiolle