TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SAMU LAAKSONEN

PALVELUNA TARJOTTAVAN SOVELLUSALUSTAN ASIAKKUUDENHALLINTA

Diplomityö

Tarkastaja: Outi Sievi-Korte
Tarkastaja ja aihe hyväksytty
2. toukokuuta 2018

TIIVISTELMÄ

Samu Laaksonen: Palveluna tarjottavan sovellusalustan asiakkuudenhallinta
Tampereen teknillinen yliopisto
Diplomityö, 47 sivua
Toukokuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Pervasive Systems
Tarkastaja: Outi Sievi-Korte

Avainsanat: asiakkuudenhallinta, sovellusalusta

Jatkuvasti suurempi osa sovelluksista toteutetaan verkkopalveluina perinteisten työpöytäkoneille asennettavien sovellusten sijaan. Teknologian edistyessä nousevat myös käyttäjien verkkopalveluiden tarjoamille ominaisuuksille asettamat vaatimukset. Monet verkkopalveluiden toteuttajat haluavat kuunnella asiakkaitaan ja tarjota heille näitä ominaisuuksia. Käytännössä kuitenkin monet näistä ominaisuuksista ovat hyvin monimutkaisia, eikä yrityksillä ole mahdollisuuksia toteuttaa näitä itse. Näissä tilanteissa helpoin ratkaisu on ottaa käyttöön kolmannen osapuolen tarjoama sovellusalusta, joka erikoistuu toteuttamaan halutun ominaisuuden.

Tässä työssä tutkittiin palveluna tarjottavan sovellusalustan asiakkuudenhallintajärjestelmän suunnitteluun ja toteutukseen liittyviä haasteita. Ratkaistavia ongelmia olivat muun muassa tiedon eheyden säilyttäminen, ohjelmallisen käytön mahdollistaminen ja järjestelmän tulevaisuudessa tapahtuvan laajennettavuuden mahdollistaminen.

Tutkimus aloitettiin tutustumalla asiakkuudenhallinnan periaatteisiin ja asiakkuudenhallintaprosessien kehittämiseen. Näitä tietoja hyödynnettiin varsinkin laajennettavuuteen vaikuttavia ominaisuuksia suunniteltaessa.

Suunnitteluvaihe alustettiin tutustumalla työssä käytettäviin teknologioihin ja arkkitehtuuriratkaisuihin. Joidenkin teknologioiden käyttö tuli vaatimuksena työn tilaajalta, osa valittiin aiemman kokemuksen perusteella ja osa valittiin niiden sopivuuden vuoksi.

Työn tuloksena toteutettiin asiakkuudenhallintajärjestelmä, jonka työn tilaaja otti käyttöön. Ratkaisut työtä pohjustaviin ongelmiin esiteltiin tietomallin kuvauksella, asiakkuudenhallintasovelluksen tarjoamien rajapintojen kuvauksilla ja UML-kaavioilla.

ABSTRACT

Samu Laaksonen: Customer relationship management software for application-platform as a service product
Tampere University of Technology
Master of Science Thesis, 47 pages
May 2018
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Out Sievi-Korte

Keywords: customer relationship management, application-platform as a service

Continually greater amount of applications is implemented as webservices. These web-services replace the traditional applications that require installation to be usable on the computer. Just like the progression of technology, so do the requirements users have for the webservices they wish to use. Many webservice providers desire to listen to their customers, and thus want to offer the required features. In practice many of these features are highly complicated and companies lack the resources to do the implementations themselves. Easiest solution for this problem is to introduce a third-party managed application-platform as part of webservice providers own implementation. These application-platforms are highly specialized components that usually focus on offering just one feature

This master's thesis focused on design and implementation of customer relationship management software for application-platform as a service product. Among the problems to be solved were issues regarding data integrity in the system, enabling programmatic usage of the implementation and taking in to consideration the future extensibility of the solution.

Work was started by performing research on the fundamentals of customer relationship management and improving customer relationship management process. This information was utilized especially on the design of features that affect future extensibility of the system.

Design phase was initiated by investigating the chosen technologies. Some technology choices were made by the requirements set by the project organizer. Other technologies were chosen based on previous experience using them.

Result of the master's thesis was a customer relationship management software that is currently used by the project organizer. Solutions for the presented problems are introduced using description of used data model, descriptions of the implemented interfaces and with UML-diagrams depicting the architecture of the customer relationship management software.

ALKUSANAT

Tämä diplomityö on tehty Vektorio Oy:n toimeksiannosta. Työn tavoitteena oli suunnitella ja toteuttaa palveluna tarjottavalle sovellusalustalle asiakkuudenhallintajärjestelmä. Haluan osoittaa kiitokseni työn mahdollistamisesta Juha Vesaselle.

Lisäksi haluan kiittää työn ohjaajaa Outi Sievi-Kortea palautteesta ja neuvoista työn aikana.

Vanhempiani Artoa ja Tittaa kiitän itseluottamuksestani ja työskentelymoraalin opettamisesta. Ilman heitä moni vaikea asia olisi jäänyt tekemättä.

Veljeäni Juusoa kiitän asenteesta jatkuvan itsensä kehittämisen suhteen. Hänen esimerkkinsä avulla muistan katsoa tulevaisuuteen ja jaksan tarttua uusiin haasteisiin.

Tyttöystävääni Outia kiitän kannustamisesta ja jaksamisesta välillä vaikeankin diplomityöprosessin aikana.

Tampereella, 20.5.2018

Samu Laaksonen

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

aPaaS	engl. application Platform as a Service, palveluna tarjottava sovellusalusta
API	engl. Application Programming Interface, ohjelmointirajapinta
B2C	engl. Business-to-customer, yrityksiltä kuluttajille myynti
BIM	engl. Building Information Model, rakennustietomalli
CA	engl. Certificate Authority, digitaalisen sertifikaatin myöntäjätaho
CRM	engl. Customer Relationship Management, asiakkuudenhallinta
FIFO	engl. First-in first-out, tiedon organisointi ja hallintatapa
http	engl. Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
IaaS	engl. Infrastructure as a Service, palveluna tarjottava infrastruktuuri
IP	engl. Internet Protocol, protokolla tietoliikennepakettien toimittamiseksi
I/O	engl. Input/Output, tiedon siirtäminen laitteistokomponenttien välillä
JSON	engl. JavaScript Object Notation, tiedostomuoto tiedonvälitykseen
npm	engl. Node Package Manager, pakettienhallintasovellus
PaaS	engl. Platform as a Service, palveluna tarjottava alusta
REST	engl. Representational State Transfer, arkkitehtuurityyli
SaaS	engl. Software as a Service, palveluna tarjottava sovellus
SSL	engl. Secure Sockets Layer, Internet-sovellusten tietoliikenteen salausprotokolla
SOAP	engl. Simple Object Access Protocol, tietoliikenneprotokolla
TDD	engl. Test Driven Development, testivetoinen ohjelmistokehitys
UML	engl. Unified Modeling Language, graafinen mallinnuskieli
URL	engl. Uniform Resource Locator, verkkosivun osoite
V3	Vektor3, 3D -mallien visualisointiin tarkoitettu sovellusalusta
WSDL	engl. Web Service Description Language, kieli tietoverkossa sijaitsevien palveluiden kuvaamiseen
XP	engl. Extreme Programming, ketterän ohjelmistokehityksen metodologia

1. JOHDANTO

Asiakkuudenhallinta on tärkeä osa modernien yritysten toimintaa. Edellisen vuosituhannen lopun jälkeen tapahtuneet harppaukset tietotekniikassa ovat mahdollistaneet yrityksille paremmat mahdollisuudet asiakastietojensa kirjaamiseen ja analysointiin. Tämä yhdistettynä maailman laajuisten markkinoiden kehittymiseen niin, että asiakkailla on entistä enemmän valtaa ja mahdollisuuksia päättää keneltä tuotteensa ostavat, on pakottanut yrityksiä panostamaan asiakassuhteidensa ylläpitoon. Asiakassuhteiden ylläpito vaatii yhä enemmän ja parempaa tietoa asiakkaista ja heidän käyttäytymisestään. Ratkaisuksi tähän ongelmaan on kehitetty erilaisia asiakkuudenhallintajärjestelmiä. Nämä järjestelmät mahdollistavat kaikkien organisaation osien vuorovaikutuksen asiakkaiden kanssa. Tämän johtaa tilanteeseen, jossa asiakas on kaikkien yrityksen toimintojen keskiössä.

Tässä työssä tutkitaan miten palveluna tarjottavan sovellusalustan asiakkuudenhallintajärjestelmä kannattaa toteuttaa. Työssä haetaan erityisesti vastauksia seuraaviin kysymyksiin. Miten varmistetaan järjestelmässä olevan tiedon eheys? Miten tietomalli kannattaa suunnitella laajennettavuuden kannalta? Minkälaisia rajapintoja tarvitaan? Miten hallitaan asiakkuudenhallintajärjestelmän käyttö sovellusalustan muista komponenteista?

Asiakkuudenhallintajärjestelmän tulee tarjota mahdollisuudet asiakkuustiedon käsittelyyn. Lisäksi järjestelmän täytyy olla integroitavissa osaksi sovellusalustan muita järjestelmiä. Kolmantena vaatimuksena järjestelmälle on mahdollisuus tallentaa tietoa asiakkaiden sovellusalustan resurssien käytöstä. Tätä tietoa tullaan käyttämään asiakkaiden laskuttamisessa. Työn tilaajana toimi tamperelainen Vektorio Oy.

Luvussa 2 käydään läpi asiakkuudenhallinnan perusteita ja teoriapohjaa. Luku käy läpi asiakkuudenhallinnan historiaa ja trendejä, joiden ansiosta asiakkuudenhallinta on monissa yrityksissä yhä suuremmassa roolissa osana strategiaa. Lisäksi luvussa käydään läpi asiakkuudenhallintaprosessin kehittämisen teoriaa ja toimintamalleja, joita yritys voi käyttää asiakkuudenhallintaprosessinsa kehittämisessä. Aiheena asiakkuudenhallintaprosessin kehittäminen on hyvin laaja, ja siitä pyritäänkin käymään läpi vain tämän työn kannalta tärkeitä osa-alueita.

Luvussa 3 paneudutaan työn tilaajan asettamiin vaatimuksiin ja työn toteutuksessa käytettyihin teknologioihin. Osa tilaajan asettamista vaatimuksista ja rajoitteista vaikuttaa suoraan työn tuloksena suunniteltavan asiakkuudenhallintajärjestelmän arkkitehtuuriin. Työssä käytettävät teknologiat käydään läpi sillä tarkkuudella, että niiden tarjoamat hyödyt ja haitat erottuvat. Lisäksi luku sisältää esittelyn työn tilaajan järjestelmästä, jonka

osaksi projektin tuloksena valmistuva asiakkuudenhallintajärjestelmä tullaan integroimaan.

Luvussa 4 käydään läpi työn tuloksena suunnitellun ja toteutetun asiakkuudenhallintajärjestelmän arkkitehtuuri. Luvussa esitellään arkkitehtuuri UML -kaavioiden ja järjestelmän tarjoamien rajapintakuvausten avulla. Lisäksi luku esittelee asiakkuudenhallintajärjestelmän relaatio-perustaisen tietomallin.

Luvussa 5 käydään läpi työn toteutusta ja arvioidaan sen onnistumista. Arviointia tehdään tilaajan vaatimuksien toteutumisen ja aikataulun toteutumisen suhteen. Luvussa 6 tehdään työstä yhteenveto.

2. ASIAKKUUDENHALLINNAN PERUSTEET

Tässä luvussa käydään läpi asiakkuudenhallinnan perusteita, jotka kattavat asiakkuudenhallinnan historiaa, asiakkuudenhallinnan kehittymiseen johtaneita muutoksia maailman laajuisessa markkinassa ja asiakkuudenhallintajärjestelmien perusteita. Lisäksi luvussa käydään läpi asiakkuudenhallintaprosessin kehittämistä yrityksessä ja siihen liittyviä vaatimuksia.

2.1 Asiakkuudenhallinta

Asiakkuudenhallinta, engl. Customer Relationship Management (CRM), on määriteltävissä monella eri tavalla. Eräs kuvaavimmista määritelmistä on Gartner Groupin tekemä: ”Asiakkuudenhallinta voidaan määritellä liiketoimintastrategiana, jonka avulla optimoidaan voittomarginaalia, liikevaihtoa ja asiakkaiden tyytyväisyyttä. Nämä mahdollistetaan organisoitumalla käyttäjäsegmenttien mukaisesti, edistämällä asiakkaiden tyytyväisyyttä parantavaa käyttäytymistä ja toteuttamalla käyttäjäkeskeisiä prosesseja.” [13].

Toinen määritelmä CRM:lle saadaan AMT:ltä. Sen mukaan ”Asiakkuudenhallinta on toimintastrategia, joka on keskittynyt asiakkaan ympärille. Strategian tavoitteena on pitää kiinni nykyisistä asiakkaista, houkutella uusia asiakkaita ja nostaa yrityksen asiakkaista saamaa tulosta hyödyntämällä teknologiaa työkaluna. Teknologia mahdollistaa yrityksen liiketoimintojen ja prosessien uudelleenjärjestelyn aiemmin mainittujen tavoitteiden saavuttamiseksi.” [13].

Asiakkuudenhallinta käsitteenä tuli tunnetuksi vasta 1990 -luvun aikana. Sen ansiosta yritysten oli mahdollista olla kanssakäymisessä asiakkaidensa kanssa paljon paremmin kuin aikaisemmin. Yksi perimmäisistä syistä asiakkuudenhallinnan nousuun oli asiakkaiden ja heidän ostostensa seuraamisen vaikeus [28].

Myöhemmin asiakkuudenhallintaan johtanutta toimintaa kutsuttiin alkujaan ”tietokanta markkinoinniksi”. Yrityksillä oli hallussaan tarpeeksi tietoa asiakkaistaan, jotta ne pystyivät luomaan kyselytutkimuksia eri asiakasryhmille tärkeän tiedon keräämistä varten. Ongelmalliseksi tällaisen tiedon hankkimisen teki se, että yrityksillä ei ollut tehokkaita keinoja sen analysoimiseksi. Vähitellen yritykset tajusivat, että ne tarvitsivat lähinnä vain perustiedot, eli mitä asiakkaat ostivat, paljonko he käyttivät rahaa ja miten he käyttivät ostamiaan tuotteita [28].

Mullistava muutos asiakkuudenhallintajärjestelmien käyttöönotossa oli niiden kaksisuuntainen luonne. Aiemmat järjestelmät oli suunniteltu vain sitä varten, että yritykset pystyisivät keräämään tietoa asiakkaistaan ja hyötymään tästä. Uudemmat järjestelmät mahdol-

listivat yritysten luoda suhteita asiakkaisiinsa antamalla heille jotain takaisin. Nämä asiakkaille annetut ns. lahjat saattoivat olla alennuksia, etuja tai jopa suoraan rahaa. Näiden etujen uskottiin parantavan asiakkaiden uskollisuutta yritystä kohtaan [28].

Asiakkuudenhallintaa on mahdollista tarkastella teoreettiselta tai käytännönläheiseltä kannalta. Teoreettisessa tarkastelussa keskitytään yritysten toiminnalliseen strategiaan ja hallinnallisiin käsitteisiin. Tällöin asiakkuudenhallinta on vahvasti sidoksissa yrityksen kulttuuriin, jossa asiakasta korostetaan eräänä yrityksen tärkeimmistä resursseista. Tämän takia yrityksen täytyy tarjota asiakkaan tarpeet täyttävää asiakaspalvelua. Käytännöllisessä tarkastelussa asiakkuudenhallinta voidaan esittää tavoitteena luoda reaaliaikainen alusta yrityksen ja asiakkaan väliseen kommunikaatioon. Tämä on mahdollista hyödyntämällä teknologiaa, jonka avulla saadaan katettua yrityksen markkinointi, myynti ja asiakaspalvelu optimoitujen prosessien alaisuuteen [13].

Asiakkuudenhallintaa ja sen omaksumista on tutkittu paljon. Ohessa listattuja syitä voi pitää hyvänä pohjustuksena asiakkuudenhallinnan omaksumisen tärkeydestä [26].

- Kilpailu asiakkaista on hyvin hankalaa. Ekonomiselta kannalta on halvempaa pitää kiinni olemassa olevista asiakkaista kuin hankkia uusia. Uusien asiakkaiden hankkiminen on jopa 5-10 kertaa kalliimpaa.
- Pareto:n periaatteen mukaisesti voidaan olettaa, että 20% asiakkaista tuo yritykselle 80% sen voitoista. Näiden asiakkaiden säilyttäminen on yrityksille hyvin tärkeää.
- Uusien asiakkaiden hankkiminen vie rahan lisäksi myös aikaa.
- Pareto:n periaatetta soveltaen voidaan päätellä, että lisäämällä uskollisten asiakkaiden määrää 5%:lla, voidaan saavuttaa jopa 25% tuottavuuden kasvu.

2.2 Asiakkuudenhallintaprosessin kehittäminen

Kehitettäessä asiakkuudenhallintajärjestelmää on hyvä ottaa huomioon myös yrityksen asiakkuudenhallintaprosessi ja sen kehittäminen. Prosessin avulla on mahdollista saada selville asiakkuudenhallintajärjestelmän vaatimuksia.

Payne väittää, että asiakkuudenhallinta (CRM) on 2000 -luvulla laajentunut hyvin tärkeäksi aiheeksi [25]. Vaikkakin termi CRM on otettu laajemmin käyttöön vasta 1990 -luvun loppupuolella, ovat periaatteet joihin se perustuu olleet olemassa paljon tätä pidempään [25]. Asiakkuudenhallinta rakentuu varsinkin suhdemarkkinoinnin varaan, joka on ollut olemassa jo kaupankäynnin alkupäivistä asti [25]. Viime vuosikymmenten aikana on esiintynyt useita merkittäviä trendejä, jotka yhdessä ovat mahdollistaneet asiakkaiden paremman palvelemisen informaatioon perustuvan suhdemarkkinoinnin avulla [25]. Mainittavia trendejä ovat tietoteknisten järjestelmien parantuminen, tietoteknisten järjestel-

mien halpeneminen, analytiikkatyökalujen parempi saatavuus, internet-pohjaisen kaupankäynnin kasvaminen ja asiakkaiden eliniän aikana tuottaman arvon tunnistaminen [25].

Kuten Payne esittää asiakkuudenhallintaan on olemassa monia katsantokantoja ja määritelmiä. Yksinkertaisimmillaan CRM:ää voidaan lähestyä kolmelta tasolta [25]:

- tietyn teknologisen ratkaisun tarjoavan projektin toteutus
- toisiinsa liittyvien asiakaslähtöisten teknologisten ratkaisujen toteuttaminen
- kokonaisvaltainen strateginen lähestymistapa asiakassuhteiden hallintaan, tavoitteena luoda asianomaisille lisäarvoa

Tutkimuksiensa perusteella Payne on tullut siihen tulokseen, että on olemassa viisi yhteen toimivaa prosessia, jotka täytyy ottaa huomioon suuressa osassa asiakkuudenhallintaa harjoittavissa yrityksissä. Nämä prosessit ovat [25]:

- **strategian kehittämisen prosessi**
- arvon luonnin prosessi
- monikanavaisen integraation prosessi
- **informaation hallinnan prosessi**
- suorituskyvyn arvioinnin prosessi

Näistä käydään tämän työn puitteissa läpi vain strategian kehittämisen prosessi ja informaation hallinnan prosessi. Tarkastelu on tarkoituksella rajattu osiin, josta ajatellaan olevan suoraan hyötyä työn pohjalta tehtävän asiakkuudenhallintajärjestelmän kehittämisessä.

Seuraavassa kappaleessa tarkastellaan strategisia puitteita, joissa asiakkuudenhallinta toimii. Lisäksi käydään läpi asiakkuudenhallinnan yleistä luonnetta yrityksissä.

Strategiset puitteet

Asiakkuudenhallinta löytyy enenemissä määrin yritysten agendan kärkipäästä [25]. Sekä pienet, että suuret yritykset omaksuvat CRM:n osaksi strategiaansa kahdesta tärkeästä syystä [25]. Ensinnäkin, uusi teknologia mahdollistaa yritysten valitsevan markkinasegmenttejä, mikro-segmenttejä tai yksittäisiä asiakkaita paljon aiempaa tarkemmin [25]. Toisekseen, uudenlainen markkinointiajattelu on tunnistanut perinteisen markkinoinnin rajoitteet ja asiakaskeskeisemmän prosessi-pohjaisen strategian potentiaalin [25].

Payne:n mukaan, vaikkakin CRM on terminä suhteellisen uusi, periaatteet sen taustalla puolestaan eivät ole [25]. Organisaatiot ovat jo pitkään harjoittaneet asiakassuhteiden ylläpitoa tavalla tai toisella [25]. Nykypäivän CRM:n erottaa tästä se, että organisaatioiden on nyt mahdollista ylläpitää yksilöllisiä suhteita asiakkaidensa kanssa, vaikka heitä olisi

tuhansia tai miljoonia [25]. Teknologiset innovaatiot ja markkinoiden muuttuminen mahdollistavat tällaisen uudistuneen perspektiivin asiakkuudenhallintaan [25].

Tämän vuosituhannen markkinatilanne eroaa paljon menneistä ajoista [25]. Nykyään asiakkaat voidaan mieltää liikkuviksi kohteiksi, ja tämän takia markkinoiden johtavat toimijat voivat pudota jalustaltaan hyvinkin nopeasti [25]. Liiketoimintaympäristön muutokset, kuten kilpailun kasvu ja monipuolistuminen, uusien teknologioiden kehitys ja saatavuus, kasvavat odotukset ja yksilöiden voimaantuminen toimivat ajureina löytää uusia tapoja saavuttaa etulyöntiasema markkinoilla [25]. Tämän johdosta yritykset ovat ymmärtäneet, että pelkästään loistavalla tuotteella kilpaileminen ei enää riitä [25]. Yritysten täytyy pystyä tarjoamaan tasalaatuista ja edukseen erottuvaa palvelua [25]. Kilpailuetu on saavutettavissa hyödyntämällä tietoa asiakkaista, kuten heidän odotuksistaan, mieltymyksistään ja käyttäytymisestään [25].

Jotta asiakkaista on mahdollista saada ajantasaista tietoa, on heidät asiakashankinta vaiheen jälkeen onnistuttava pitämään yrityksen asiakkaina [25]. Kirjassaan Payne mainitsee, että asiakkaan elinikäisen arvon maksimointi on suhdemarkkinoinnin perustavoite [25]. Tässä tapauksessa elinikäiseksi arvoksi lasketaan asiakkaan yritykseen tulevaisuudessa tuottama nettotuotto [25]. Yrityksen hyväksyessä tämän periaatteen asiakkaan elinikäisestä nettotuotosta tulee heidän hyväksyä myös, että kaikki asiakkaat eivät ole yhtä tuottavia [25]. Täten yrityksen tulee kehittää strategioita, joilla pyritään parantamaan haluttujen asiakkaiden kannattavuutta [25]. Parhaassa tapauksessa tämä johtaa positiiviseen kierteeseen, jonka ansiosta yritys saa haluamiaan asiakkaita, joihin liittyvällä tiedolla se pystyy hankkimaan jatkuvasti lisää asiakkaita ja samalla pitämään aiemmin hankkimansa asiakkaat [25]. Lojaalit asiakkaat auttavat tulevaisuuden asiakashankinnassa, puhumalla yrityksessä positiiviseen sävyyn ja täten vähentämällä asiakkaiden hankkimisesta koituvia kuluja [25].

Myös Payne listaa kirjassaan, että asiakkuudenhallinnan nousun taustalla on monia tärkeitä trendejä [25]. Siirtyminen liiketoimintapohjaisesta markkinoinnista suhdepohjaiseen markkinointiin tarkoittaa, että yrityksen ei välttämättä tarvitse saada voitettua asiakasta puolelleen jokaisen ostotapahtuman yhteydessä yhä uudestaan [25]. Täten yrityksen panostaessa asiakassuhteen rakentamiseen, se voi saavuttaa korkeamman liikevaihdon ja paremman tuloksen matalammilla kustannuksilla [25]. Toinen tärkeä trendi on asiakkaiden näkeminen liiketoiminnan edun kannalta tärkeinä toimijoina, sen sijaan että heitä ajateltaisiin vain yleisönä, jolle pyritään toimittamaan mainoksia yrityksen tuotteista [25]. Tämä näkökannan muutos sallii yksittäisten asiakassuhteiden kehittämisen niiden asiakkaiden kanssa, jotka yritys kokee kaikkein tuottavimmiksi [25]. Kolmas trendi on yrityksen organisoituminen prosessien mukaisesti toiminnallisuuden sijaan. Tämä perustuu siihen, että nykypäivänä asiakkaat harvoin haluavat ostaa pelkkää tuotetta [25]. Sen sijaan he ovat kiinnostuneita koko tuotteen elinkaaren ketjusta, toimittamisesta sen mahdolliseen hävittämiseen saakka [25]. Tällaisen ketjun asiakaskokemuksen varmistaminen vaa-

tii yrityksen organisaation vastaavanlaista mukautumista [25]. Neljäs trendi on informaation käyttäminen proaktiivisesti reaktiivisuuden sijaan [25]. Näin toimivien yritysten täytyy tuntea asiakkaansa ja kilpailijansa entistä paremmin. Ennakoivasti toimiva yritys on perinteisesti toimivia kilpailijoitaan vahvempi esimerkiksi tilanteessa jossa asiakas ei ole tyytyväinen hankintaansa [25]. Reaktiivinen yritys ei välttämättä saa asiakkaan tyytymättömyyttä tietoonsa ennen kuin asiakas on jo vaihtanut yrityksen kilpailijan tuotteeseen [25]. Reaktiivinen yritys sen sijaan pyrkii aktiivisesti etsimään tyytymättömiä asiakkaita ja korjaamaan ongelman [25]. Tietojärjestelmien käyttöönotto tiedon arvon maksimoiseksi on viides tärkeä trendi. Asiakastietoon integroituvat myynti-, markkinointi-, asiakaspalvelu- ja tukijärjestelmät parantavat yrityksen mahdollisuuksia kuunnella asiakkaitaan paremmin ja oppia näiltä [25]. Kuudes tärkeä trendi on arvon vaihtokaupan tasapainottaminen [25]. Yksinkertaisimmillaan tämä on suhde asiakkaalta saadun ja asiakkaalle toimitetun arvon välillä [25]. Yritys pyrkii tasapainottamaan tämän suhteen niin, että se pystyy maksimoimaan asiakkaalta eliniän aikana saadun arvon [25]. Tarjoamalla asiakkaalle liian laadukkaita tuotteita keskimääräiseen markkinahintaan voi yritys ajautua tilanteeseen, jossa haluttuja tulomarginaaleja ei saada ylläpidettyä [25]. Toisaalta laskemalla tuotteiden tai palvelujen tasoa liikaa, voi yritys ajautua tilanteeseen jossa sen asiakkaat vaihtavat kilpailijaan [25]. Seitsemäs ja viimeinen tärkeä trendi on yksilöllisen markkinoinnin kehittyminen [25]. Tavallisessa asiakassegmenteille suunnatussa markkinoinnissa ei ole tarvinnut välittää samoista asioista kuin yksittäisiä asiakkaita käsitellessä [25]. Yksittäiset asiakkaat muistavat aiemmat tapahtumat, ovat mukana vuorovaikutuksessa, tekevät valituksia ja kommunikoivat toisten asiakkaiden kanssa [25]. CRM järjestelmät sallivat yritysten toimivan paremmin asiakkaiden kanssa, koska heillä on mahdollisuus päästä käsiksi kaikkeen kyseiseen asiakkaaseen aiemmin ylös kirjattuun tietoon [25].

Asiakkuudenhallintaa kehitettäessä yrityksen tulee käydä läpi viisi toimirajat ylittävää prosessia, jotka on mainittu tässä luvussa ennen tätä kappaletta. Seuraava kappale käsittelee yrityksen strategian kehittämisen prosessia.

Strategian kehitys

Asiakkuudenhallintastrategian kehittäminen aloitetaan hyödyntämällä strategian kehittämisen prosessia [25]. Tämä prosessi muovaa muita neljää avainasemassa olevaa CRM prosessia, mutta ennen kaikkea tämä määrittelee organisaation CRM toimintojen tavoitteet ja parametrit [25]. Strategian kehitysprosessi koostuu liiketoimintastrategian ja asiakasstrategian määrittelystä. Lisäksi sen avulla pidetään huoli, että nämä kaksi strategiaa ovat linjassa toistensa kanssa [25]. Strategian kehitysprosessi keskittyy hankkimaan vastaukset seuraaviin kysymyksiin [25].

- Missä me olemme nyt ja mitä haluamme saavuttaa yrityksenä?
- Keitä ovat asiakkaat joita haluamme ja kuinka meidän pitäisi segmentoida heidät?

Vastauksen hankkiminen näihin kysymyksiin strategian kehityksen prosessin avulla auttaa yritystä matkalla kohti arvon luomista heidän asiakkailleen [25].

Asiakkuudenhallintaa itsessään ei pidä käyttää yrityksen liiketoimintastrategian kehittämisessä [25]. Ennemmin yrityksen liiketoimintastrategiasta täytyy olla täysi ymmärrys, jotta asiakkuudenhallintastrategian kehittäminen on mahdollista [25]. Tämän myötä asiakkuudenhallintatoimet on mahdollista linjata yrityksen liiketoimintastrategian kanssa parhaan tuloksen saavuttamiseksi [25].

Liiketoimintastrategian ohella yrityksen täytyy vahvistaa heidän visionsa [25]. Yrityksen visio heijastaa organisaation uskomuksia, arvoja ja toiveita [25]. Vision on tarkoitus olla jotain, joka kuvastaa yrityksen tavoitteita ja olemassaolon tarkoitusta. Tämän lisäksi sitä voidaan hyödyntää kilpailijoista erottumiseen [25]. Visiota ja yrityksen arvoja kehitetään usein väärin [25]. Monet yritykset ottavat käyttöön yksinkertaisen sloganin, jota kohti he pyrkivät kehittymään. Tämän sijaan yritysten pitäisi tunnistaa markkinoilta ne ominaisuudet, joiden avulla tulevaisuudessa tullaan saavuttamaan kilpailuetua, ja luomaan visionsa näiden ympärille [25].

Vision kehittämisen jälkeen yrityksen pitää tarkastella toimikenttäänsä ja kilpailijoitaan, jotta on mahdollista määritellä oma sijoittumisensa tällä kentällä [25]. Alaa tulee tarkastella sekä nykyisen olemassa olevan, että tulevaisuuden tilanteen suhteen. Tällaisen tarkastelun tärkeydestä toimii hyvänä esimerkkinä viime vuosikymmenten teknologiset keksinnöt [25]. Monet alat ovat muovautuneet näiden keksintöjen ansiosta hyvinkin nopeasti, joten yritys joka oli vahvoilla vielä kaksikymmentä vuotta sitten, mutta ei pystynyt reagoimaan muutoksiin on hyvin mahdollisesti ajettu pois alalta [25]. Nämä alalla kuin alalla tapahtuvat rakenteelliset muutokset voidaan jakaa kahteen luokkaan. Ensimmäisen luokan muutokset johtavat vähennyksiin yrityksen käyttämien välikäsien suhteen [25]. Toisen luokan muutokset puolestaan lisäävät yrityksen hyödyntämien välikäsien määrää [25].

Asiakasstrategiassaan yrityksen täytyy tehdä päätöksiä, minkälaisia asiakkaita se haluaa ja minkälaisia asiakkaita yritys pyrkii välttelemään [25]. Nämä päätökset pohjautuvat yrityksen omiin vahvuuksiin ja heikkouksiin [25]. Jatkuvasti kilpaillussa liiketoimintaympäristössä harvojen yritysten on mahdollista tarjota ”kaikkea kaikille” [25]. Kaikki asiakkaat ovat erilaisia, ja täten myös kaikkia asiakassuhteita täytyy hoitaa eri tavoin [25]. Tätä voidaan pitää yhtenä asiakkuudenhallinnan avain periaatteena [25].

Määritellessä asiakasta, on heidät mahdollista jakaa kolmeen ryhmään: ostajat, välittäjät ja tuotteen loppukäyttäjät [25]. Yrityksen täytyy pyrkiä tasapainottelemaan näihin ryhmiinsä käyttämän ajan, rahan ja vaivannäön kesken saavuttaakseen parhaan mahdollisen tuloksen [25]. Kaikkia näitä kolmea asiakasryhmää on mahdollista tarkastella tarkemmin segmentoimalla ne erilaisten ominaisuuksien mukaan [25]. Segmentoinnin tuloksena saadaan asiakasryhmiä, joita yritys haluaa kasvattaa, ja toisaalta myös asiakasryhmiä joiden

kutistuminen voi olla yritykselle hyväksi [25]. Halutuille segmenteille löytyy yleispäteviä kriteereitä, joita on mahdollista käyttää melkein missä vain yritystoiminnassa [25]. Näitä ominaisuuksia ovat: segmentin koon pitää olla mitattavissa, segmentin ominaisuuksien pitää olla mitattavissa, segmentin täytyy pystyä tuottamaan yritykselle voittoa pitkällä aikavälillä (jotta panostaminen siihen on hyväksyttävissä), segmenttiin kuuluvien asiakkaiden pitää olla tavoitettavissa kohtalaisella rahallisella panostuksella ja segmentin pitää olla kestävä pitkällä aikavälillä [25].

Asiakkuudenhallintastrategian kehitys vaatii yrityksen liiketoimintastrategian ja asiakasstrategian yksityiskohtaista läpikäymistä [25]. Strategioiden ollessa kunnossa, täytyy lisäksi varmistaa, että molemmat on kohdistettu samoihin tavoitteisiin eikä niiden välillä ole ristiriitoja [25].

Tiedonhallinta

Tiedonhallintaprosessi kattaa kaksi avain aktiviteettia: asiakastietojen kerääminen ja kokoaminen, ja kerätyn tiedon hyödyntäminen asiakasprofiilien rakentamisessa paremman asiakaskokemuksen saavuttamiseksi [25]. Nämä aktiviteetit kiteytyvät kahteen kysymykseen [25]:

- Kuinka asiakastieto pitää organisoida?
- Kuinka tätä kerättyä tietoa pystyy käyttämään asiakkaiden käyttäytymisen tulkitsemiseen?

Tiedonhallintaprosessin lopputuloksena on yrityksen käyttöä varten luotu integroitu asiakkuudenhallintaratkaisu [25]. Kaikki asiakkuudenhallinnassa mukana olevat prosessit nojaavat integroituun asiakkuudenhallintajärjestelmään tiedonlähteenä [25]. Nämä prosessit tuottavat lisäksi järjestelmään omaa tietoaan [25]. Täten pohjalta löytyvä asiakastieto nousee tärkeään rooliin tiedon analysoinnin kannalta. Asiakkuudenhallintasovelluksen hyödyt korostuvat päivittäisessä käytössä, kun haetaan hyväksyttävää tasapainoa järkevän ja idealistisen toiminnan välillä [25]. Täten asiakkuudenhallintasovelluksessa täytyy olla saatavilla oikea tieto oikeaan aikaan. Liian vähäinen tai liiallinen tiedon tallentaminen haittaa tähän tavoitteeseen pääsemistä [25].

Kehitettäessä teknologista toteutusta asiakkuudenhallintasovellukseksi, tulisi kehitysprosessissa ottaa huomioon seuraavat asiat: asiakkuudenhallinnan tekniset esteet, tiedon tallentaminen, analytiikkatyökalut, tietojärjestelmät, asiakasrajapinnassa ja taustatoimissa käytössä olevat sovellukset ja kehittyvän teknologian mukanaan tuomat haasteet [25].

Teknisiä esteitä korostaa hyvin vertailu asiakkuudenhallinnan odotusten ja tulosten välillä [25]. Kasvavat odotukset luovat yhä enemmän painetta vanhoille teknisille ratkaisuille, joita ei välttämättä ole aikanaan suunniteltu vastaamaan näihin uusiin vaatimuksiin [25]. Ongelmaa voidaan lähteä ratkaisemaan miettimällä vastauksia seuraaviin kysymyksiin [25]. Keräämmekö tarvitsemaamme tietoa asiakkaistamme? Pystymmekö valjastamaan

kerätyn tiedon yrityksen työntekijöiden käyttöön? Pystymmekö analysoimaan asiakastietoa parhaalla mahdollisella tavalla?

Asiakkuudenhallintajärjestelmän tiedon tallentamisjärjestelmän valinta on tärkeä osta tiedonhallintaa [25]. Yksinkertaisimmillaan ratkaisu voi olla tietokanta, jota yritys käyttää tuotteidensa markkinoinnissa ja myynnissä [25]. Tämän kaltaisesta tietokannasta voi löytyä mahdollisuus postituslistojen luomiselle, joten se sisältää jo yksinkertaisen asiakkaan käsitteen [25]. Tietokantaa kehittyneempi ratkaisu on tietovarasto [25]. Tietovarastojen heikkoutena on usein se, että organisaation eri osastoilla on omat tietovarastonsa, ja täten yrityksen sisältä löytyy monta erilaista käsitystä samasta asiakkaasta [25]. Tietovarastoa monipuolisempi ratkaisu on tietovaranto, johon kootaan asiakkaiden tiedot monista yksittäisistä tietovarastoista [25]. Täten yrityksellä on saatavillaan yhdistetty näkymä jokaisen heidän asiakkaansa tiedoista. Laajin mahdollinen ratkaisu asiakastiedon hallintaan on integroitu asiakkuudenhallintajärjestelmä [25]. Tällaiset monimutkaiset järjestelmät vaativat yritykseltä erityistä panostamista niiden kehittämiseen [25]. Etuna integroiduilla asiakkuudenhallintajärjestelmillä on, että asiakkaiden tiedot ovat jatkuvasti näkyvissä kaikille organisaation osille [25]. Tämä toimii kulmakivenä asiakaslähtöisen liiketoiminnan kehittämiselle [25].

Asiakastiedon tallentamisesta kannattaa pyrkiä ottamaan kaikki hyödyt irti [25]. Tätä varten yrityksen kannattaa hankkia analytiikkatyökaluja, joilla on mahdollista pureutua tallennettuun asiakastietoon [25]. Analysoimalla tietoa on yrityksellä mahdollisuus etsiä siitä kaavoja, joita olisi hyvin vaikeaa huomata ilman tietokoneen laskentakapasiteettia [25]. Tallennettua tietoa voi analysoida saadakseen selville lisää yrityksen markkinasegmentoinnista, asiakkaiden taipumuksista, asiakkaiden profiloinnista tai yrityksen tuottavuudesta [25]. Analyysin tuloksia on mahdollista tarkastella yksinkertaisin visualisoinnein tai monimutkaisempien päätöspuiden avulla. [25]

Asiakkuudenhallintajärjestelmän hankinta on hankala prosessi [25]. Usein valmiit asiakkuudenhallintaratkaisut keskittyvät ominaisuuksiltaan tiettyyn osaan liiketoimintaa [25]. Tämän vuoksi kaiken kattavan ratkaisun hankinta voi olla vaikeaa [25]. Yrityksen täytyy tehdä sisäistä tutkimusta kartoittaakseen tarpeensa, jotta optimaalisen ratkaisun löytäminen on mahdollista [25]. Tästäkin huolimatta yritykset joutuvat usein ottamaan käyttöön monia asiakkuudenhallintajärjestelmiä, jotta ne saavat käyttöönsä kaikki haluamansa ominaisuudet [25].

Kerätessä tietoa asiakkaista tulee yrityksen kiinnittää erityistä huomiota tietoturvaan. Asiakkaan ja yrityksen välisen suhteen syventyessä tulee asiakas jakaneeksi jatkuvasti enemmän ja enemmän henkilökohtaisia tietojaan yrityksen kanssa [25]. Monilla mailla on toisistaan eriävät proseduurit koskien henkilötietojen tallentamista [25]. Tämä monimutkaistaa tiedon tallentamista entisestään.

3. TOTEUTUKSEN PERUSTEET

Tässä luvussa esitellään asiakkuudenhallintasovelluksen toimintaympäristö ja täten sen vaadittuihin ominaisuuksiin vaikuttavat tekijät. Lisäksi luvussa esitellään tekniikat ja teknologiat, joita asiakkuudenhallintasovelluksen toteutuksessa käytetään. Osa teknologioista tuli vaatimuksena työn tilaajan toimesta, osa valittiin aiemman osaamisen perusteella työmäärän vähentämiseksi ja osa teknologioista valittiin koska ne vaikuttivat olevan parhaita ratkaisuja kyseiseen ongelmaan.

3.1 Ympäristö

Työn tilaaja, Vektorio Oy, on erikoistunut rakennusmallien, engl. building information model (BIM) visualisointiin. Heidän internet selain pohjainen tuotteensa Vektor3 (V3) on palveluna toimitettava sovellusalusta. Vektor3 alustaa on mahdollista hyödyntää missä vain rakennusprojektin vaiheessa, oli kyse sitten suunnittelusta, rakennuttamisesta, myynnistä tai ylläpidosta. Alustan tarjoaman rajapinnan ansiosta käyttäjien on mahdollista rakentaa omat sovellusratkaisunsa teknologian päälle, täten rikastaen pohjalla toimivaa malliaineistoa uusin ja mielenkiintoisin tavoin [31].

3.1.1 Sovellusalusta palveluna

Sovellusalusta palveluna, engl. application-platform as a service (aPaaS), eroaa muista yleisesti tarjottavista palveluista, eli infrastucture-as-a-service (IaaS), platform-as-a-service (PaaS) ja software-as-a-service (SaaS).

Artikkelissaan Zhang kuvailee IaaS:a pilvipalvelumalliksi, jossa pilvipalvelun toimittaja tarjoaa laskenta-, tallennus- ja verkkoresursseja asiakkailleen palveluna silloin, kun asiakkaat niitä tarvitsevat [32]. Markkinoilta löytyy monia pilvipalvelutoimittajia, jotka tarjoavat IaaS ratkaisuja, joista esimerkkeinä mainittakoon Amazon AWS ja Microsoft Azure.

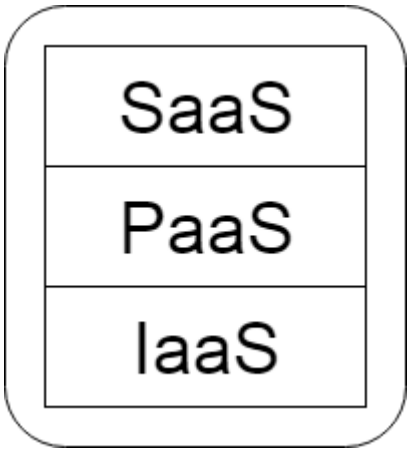
Platform-as-a-service, eli PaaS voidaan kuvata pilvipalvelumalliksi, jossa palveluntoimittaja tarjoaa IaaS:a kehittyneempiä resursseja, kuten käyttöjärjestelmiä, virtuaalipalvelimia, tietoverkkoja, verkkopalvelimia, sovelluspalvelimia, tietokantoja, varmuuskopiointi ominaisuuksia ja katastrofista toipumisessa auttavia palveluita [29].

Kolmas yllä mainituista palveluista eli SaaS on ohjelmiston toimitusmalli, jossa kolmannen osapuolen ohjelmistotoimittaja tarjoaa ohjelmistonsa loppukäyttäjien saataville verkon yli tarjottavana palveluna [1]. Tämä mahdollistaa sovelluksen toimittamisen suurelle määrälle käyttäjiä ilman, että heidän kaikkien täytyy asentaa erillistä asiakassovellusta omalle koneelleen.

Kuten Soni artikkelissaan listaa, kaikkia kolmea yllä kuvattua palvelumallia yhdistävät seuraavat tekijät [29]:

- käyttäjien saatavilla juuri silloin, kun he itse palvelua haluavat käyttää
- laaja saatavuus verkon kautta
- resurssien keskittäminen
- nopea joustavuus
- mitattu käyttö

Yllä listatut palvelut voidaan esittää kuvan 3.1 mukaisessa hierarkiassa sen perusteella, miten ne sijoittuvat toisiinsa nähden.



Kuva 3.1 *Palvelutyyppien keskinäinen hierarkia*

Palveluna tarjottava sovellusalusta sijoittuu kuvassa 3.1 kuvatussa hierarkiassa PaaS ja SaaS kerrosten väliin. Tästä johtuen palveluna tarjottava sovellusalusta sisältää etuja ja heikkouksia sekä PaaS että SaaS arkkityypin palveluista. Jokainen aPaaS tuote on luonnollisesti erilainen, joten yleispätevän ominaisuuslistauksen tekeminen on käytännössä mahdotonta.

Vektor3 tuote on sovellusalusta, joten se on tarkoitettu käytettäväksi osana asiakkaiden omia järjestelmiä. Täten on mahdollista tarkastella osaa sen ominaisuuksista itsenäisesti, mutta käytännössä ominaisuuksia listatessa pitää ottaa huomioon myös sen palvelun ominaisuudet, jonka osana V3 tuote on. Käytännössä kaikki järjestelmät joihin V3 integroidaan ovat itsessään SaaS -tyylisiä. Tämän vuoksi sovellusalusta tarjoaa luonnostaan ominaisuuksia kuten: saatavilla kun käyttäjä haluaa, saatavilla verkon kautta ja automaattinen skaalautuminen. Nämä ominaisuudet koskevat vain sovellusalustan käyttöä, eli jos asiakkaan oma SaaS -järjestelmä ei kykene automaattisesti skaalautumaan käyttäjämäärän mukaan ei sovellusalustan skaalautumisesta ole tässä tapauksessa hyötyä.

3.1.2 Liiketoiminnalliset vaatimukset

Toteutettaessa asiakkuudenhallintasovellusta, on sen suunnittelu aloitettava tilaajan vaatimuksien analysoinnilla. Sovellusalueesta johtuen asiakkuudenhallintasovelluksen vaatimukset ovat erilaiset, kuin yleisempään business-to-customer (B2C) asiakkuudenhallintaan tarkoitetuissa sovelluksissa. Tässä kappaleessa listatut vaatimukset luetaan liiketoiminnallisiksi vaatimuksiksi, koska niiden toimivuudella ja luotettavuudella on suora yhteys tilaajan liiketoimintaan.

Asiakkuudenhallintasovelluksen pitää mahdollistaa asiakkuuksien hallinta. Asiakkaiden tietoja pitää pystyä tallentamaan, muokkaamaan ja lukemaan. Näihin tietoihin sisältyy minimissään asiakasyrityksen nimi, asiakasyrityksen yhteyshenkilön tiedot ja laskutukseen käytettävät tiedot.

Asiakkuudenhallintasovelluksen on mahdollistettava sovellusalustaa käyttävän asiakkaan tunnistaminen. Sovellusalustaa käytetään ohjelmallisesti, eikä loppukäyttäjä ole suoraan tekemisissä taustalla toimivien järjestelmien kanssa. Täten asiakkuudenhallintasovelluksen täytyy tarjota mahdollisuus tunnistaa asiakas ilman, että sovellusalustan päälle rakennetun sovelluksen loppukäyttäjä joutuu tunnistautumaan erikseen sovellusalustaan.

Asiakkuudenhallintasovelluksen on tarjottava valtuutuksien hallinta. Sovellusalustan päälle toteutetun sovelluksen on mahdollista päästä lukemaan ja kirjoittamaan vain kyseisen asiakkaan omia tietoja. Pääsy muiden asiakkaiden tietoihin tai resursseihin on estetty.

Sovellusalustan jatkuvan kehityksen edellytyksenä on asiakkaiden käyttötietojen tallentaminen jatkoanalysointimahdollisuuksia varten. Asiakkuudenhallintasovelluksen tulee pystyä tallentamaan perustiedot ennalta määrätyistä tapahtumista.

Analysointia varten kerättävän tiedon lisäksi asiakkuudenhallintasovelluksen pitää pystyä keräämään tietoa asiakkaiden resurssien käytöstä. Sovellusalustan laskutusmalli on käyttöpohjainen ja asiakkaille lähetettävät laskut luodaan tämän kerätyn tiedon pohjalta.

Asiakkuudenhallintasovelluksen kautta on pystyttävä lähettämään laskuja asiakkaille. Aluksi vaaditaan laskutusmahdollisuus suoriin verkkopankkilaskuihin. Laskutuksen käsittelyyn käytetään kolmannen osapuolen tarjoamia palveluja.

Asiakkuudenhallintasovelluksen on oltava laajennettavissa. Sovellusalusta on jatkuvassa kehityksessä, ja siihen tulee tulevaisuudessa uusia ominaisuuksia. Täten asiakkuudenhallintasovelluksen pitäisi olla mahdollisimman helposti laajennettavissa tukemaan uusia ominaisuuksia, joista halutaan kerätä käyttäjätietoja, ja jotka halutaan sitoa mukaan asiakkaiden laskutukseen. Käytännössä valmistautuminen kaikkiin mahdollisiin laajennuksiin on mahdotonta, joten tämän vaatimuksen osalta pyritään siihen, että tehdyillä ratkaisuilla minimoidaan laajennettavuudelle haitallisten rakenteiden määrä sovelluksessa.

3.2 Teknologiat

Käytettyjen teknologioiden ja tekniikoiden valinta pohjautuu ensisijaisesti työn tilanneen yrityksen käytössä olevaan teknologia-pinoon. Modernit mikropalvelu-arkkitehtuuri -tyylillä toteutetut järjestelmät perustuvat toisistaan erillisenä toteutettujen palveluiden ja näiden välisten rajapintojen hyödyntämiseen. Koska kaikki tiedonvälitys hoidetaan erillisten rajapintojen kautta teknologiakohtaisten ratkaisujen sijasta, voi toteutukseen käytetyn teknologian valita niin, että se on paras kyseisen ongelman ratkaisemiseksi. Käytännössä tällaisissa tilanteissa on kuitenkin usein parempi valita ne teknologiset ratkaisut, jotka ovat jo käytössä järjestelmän muissa osissa. Täten pystytään varmistamaan uusien komponenttien helpompi käyttöönotto ja parempi ylläpidettävyys.

3.2.1 Test-driven development

Test-driven development (TDD), suomennettuna testaamiseen pohjautuva kehittäminen, on niin sanottu opportunistinen ohjelmiston kehitysmenetelmä [2]. Artikkelissaan Bhat mainitsee, että se on ollut ohjelmistoalalla hajanaisessa käytössä jo NASA:n Project Mercuryn ajoista 1960 -luvulta asti, mutta viime aikoina menetelmä on saanut uutta näkyvyyttä yhtenä Extreme Programming (XP) metodeista [2].

Bhat kertoo artikkelissaan, että TDD:n perusperiaate on ohjelmistojen kehittäminen seuraavanlaisten nopeiden iteraatioiden avulla [2]:

- kirjoitetaan pieni määrä automatisoituja yksikkötestejä
- suoritetaan vastakirjoitetut yksikkötestit ja varmistetaan, että ne epäonnistuvat (koska ei ole olemassa oikeaa koodia jota vasten ne suoritetaan)
- toteutetaan ohjelmakoodi, jonka avulla vastakirjoitetut yksikkötestit saadaan ajettua onnistuneesti läpi
- suoritetaan vastakirjoitetut yksikkötestit uudelleen, jotta voidaan varmistaa niiden läpimeno
- refaktoroidaan toteutusta tai mahdollisesti testikoodia tarpeen mukaan
- ajoittain (mieluusti kerran päivässä tai useammin) ajetaan kaikki testit läpi, jotta voidaan varmistua siitä, että uusi sovelluskoodi ei hajota vanhoja testitapauksia

Huomionarvoista on, että seuraamalla yllä esitettyä iteraation rakennetta päästään tilanteeseen, jossa kaikki testit saadaan onnistuneesti ajettua läpi ennen uuden sovelluskoodin lisäämistä. Tämän pitäisi parantaa luotettavuutta uuteen ohjelmistokoodiin, koska on epätodennäköistä, että se luo uusia vikoja, tai piilottaa järjestelmässä jo mahdollisesti olleita ongelmia [2].

Muutamia mahdollisia TDD:n sovellusprojektiin tuomia etuja [2]:

- tehokkuus ja palaute: kirjoitettaessa testit ennen koodia saadaan sovelluksen rakennetta ohjattua suuntaan, jossa toiminnallisuus on hyvin hienojakoisissa komponenteissa
- matalan tason suunnittelu: testit tarjoavat määrittelyn luokille, metodeille ja rajapinnoille
- päivitysten aiheuttamien virheiden ehkäiseminen: ajamalla aiempaa koodia varten suunnitellut testit voidaan vähentää tulevaisuudessa tehtävien pienten korjausten aiheuttamia tahattomia virheitä
- pakottaa ohjelmoijat tekemään testattavaa koodia: testattavalla koodilla on helpompaa hallita mahdollisia regressioita

3.2.2 docker

Docker on Yhdysvaltalaisen Docker Incorporated yrityksen kehittämä teknologia. Dockerin tavoitteena on tarjota konttialusta, joka mahdollistaa kaikenlaisten sovellusten ajamisen hybridi-pilvessä. Alusta mahdollistaa todellisen itsenäisyyden sovellusten ja infrastruktuurin välillä, luoden mallin joka mahdollistaa paremman yhteistoiminnan ja innovaation [7].

Alla on listattu etuja, joita docker alustan käytöllä on saavutettavissa:

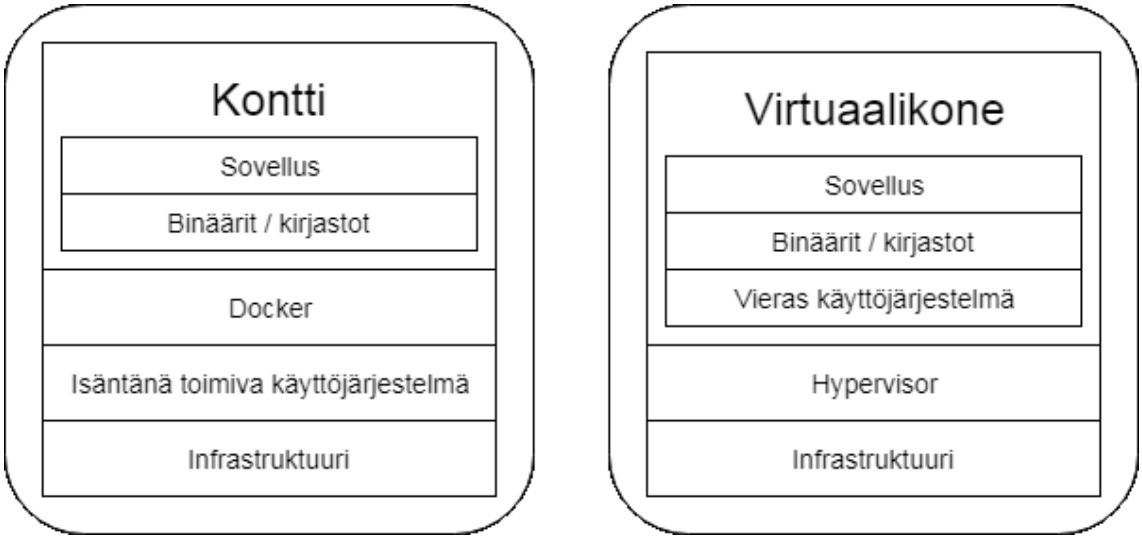
- parempi ketteryys sovellusten kehityksessä ja käyttöönotossa
- portattavuus eliminoi sovelluskehityksessä esiintyvän ”toimii minun koneellani” -efektin
- sisäänrakennetut turvallisuusominaisuudet
- saavutetut kustannussäästöt infrastruktuurin ja toiminnan optimoinnista
- yksinkertaisuus
- avoimuus, joka perustuu avoimen lähdekoodin teknologiaan
- itsenäisyys, saavutettavissa infrastruktuurin ja sovellusten erottamisella

Dockerin toteuttama konttialusta mahdollistaa sovellusten paketoimisen standardoituihin yksiköihin kehitystä, siirtoa ja käyttöönottoa varten. Näitä yksiköitä kutsutaan termillä kontti-kuva. Jotta kontti-kuvien olisi mahdollista toimia standardoituina yksiköinä, täytyy niistä löytyä kaikki tarvittava halutun sovelluksen suorittamista varten. Kontista löytyy siis sovelluksen koodi, ajoympäristö, käyttöjärjestelmän työkalut, käyttöjärjestelmäkirjastot ja tarvittavat asetukset. Näiden vaatimusten ansiosta kontin on mahdollista eristää sovellus sen ympäristöstä täydellisesti, vähentäen ongelmia kehittäjien ja tuotantoinfrastruktuurin välillä [6].

Kontit voidaan luokitella virtuaalikoneiden kaltaisiksi, koska molempien resurssien eristäminen ja käyttöoikeuksien jako toimivat suunnilleen samalla tavalla. Käytännössä kontit ja virtuaalikoneet kuitenkin toimivat täysin eri tavalla. Virtuaalikoneet virtualisoivat

käytössä olevan raudan, mutta kontit puolestaan virtualisoivat käytössä olevan käyttöjärjestelmän. Tämän vuoksi kontit ovat helpommin siirrettävissä eri ympäristöihin ja myös toimivat tehokkaammin kuin virtuaalikoneet [6].

Kuvassa 3.2 esitellään konttiteknologian ja virtuaalikoneen erot abstraktiokerrosten tasolla. Kuva on yksinkertaistettu versio docker:n omissa resursseissa tarjoamasta kuvasta (Docker what-is-container 2018).



Kuva 3.2 *Konttiteknologian ja virtuaalikoneen abstraktiokerrosten vertailu*

Kuten kuvasta 3.2 on nähtävissä, löytyy dockerin ja virtuaalikoneen väliltä paljon yhteneväisyyksiä. Molemmat ratkaisut mahdollista sovellusten ja niiden riippuvuuksien paketoimisen omiin ympäristöihinsä. Lisäksi molemmissa ratkaisuissa on mahdollista ajaa useita eri sovelluksia saman infrastruktuurin päällä. Suurin ero on kuitenkin tavassa, jolla abstraktio on toteutettu. Virtuaalikoneita käyttäessä joudutaan käynnistään kokonainen erillinen kopio käyttöjärjestelmästä, vaikka sovelluksen suorittamiseen riittäisi pieni osa ladattavista resursseista. Docker puolestaan mahdollistaa ns. ”pienimmän yhteisen nimittäjän” mukaisen resurssien käyttämisen. Kontteihin paketoidaan sovellus ja sen tarvitsemat riippuvuudet. Käytön aikana kontti hyödyntää isäntänä toimivan käyttöjärjestelmän ydintä, joten kontit pysyvät kooltaan paljon virtuaalikoneita kevyempinä.

Docker tarjoaa komentorivityökalut konttien hallintaa varten. Kontin sisältö ja toiminnallisuus määritellään erillisellä Dockerfile:llä. Dockerfile syntaksi on yksinkertaista ja itsensä dokumentoivaa [3]. Kuvassa 3.3 on esitetty yksinkertainen Dockerfile, joka on saatavilla avoimena lähdekoodina [11].

```
1  FROM scratch
2  COPY hello /
3  CMD ["/hello"]
```

Kuva 3.3 *Dockerfile esimerkki (https://github.com/docker-library/hello-world)*

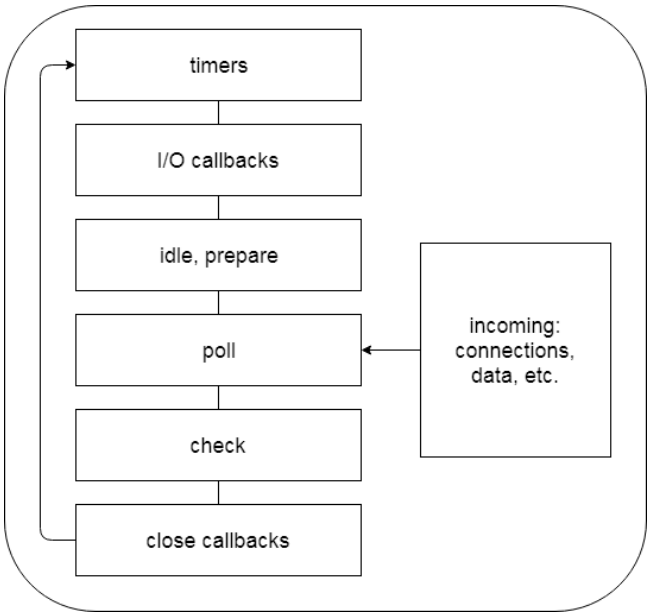
Dockerfile tiedostoilla on oma rakenteensa. Tämä rakenne koostuu avainsanoista ja niihin liittyvistä komennoista. Kuvassa 3.3 esitetyssä esimerkissä on käytössä kolme avainsanaa, FROM, COPY ja CMD. Kaikki Dockerfile tiedost alkavat FROM avainsanalla. Se kertoo mitä docker kontin kuvaa käytetään pohjana nyt määriteltävälle kontille. Esimerkissä on valittuna scratch niminen konttikuva. Seuraava avainsana esimerkissä on COPY, tämä mahdollistaa tiedostojen kopioimisen docker isäntäkoneelta docker kontin sisään. Esimerkkitapauksessa kopioidaan hello niminen tiedosto kontin tiedostohierarkian juureen. Viimeinen esimerkistä löytyvä avainsana on CMD. Tämä avainsana määrittelee sen, että mikä komento suoritetaan, kun kontti käynnistetään [5].

3.2.3 Node.js

Artikkelissaan Liang kuvaa Node.js olevan C++ ohjelmointikielellä kirjoitettu javascript ajoympäristö [14]. Node.js käyttää Googlen Chrome projektin V8 javascript -moottoria sen hyvän suorituskyvyn vuoksi. Tavallisen javascript ohjelmakoodin suorittamisen lisäksi Node.js tarjoaa pääsyn moniin matalan tason käyttöjärjestelmä rajapintoihin, kuten tiedostojärjestelmän hallintaan ja verkkotoimintoihin.

Node.js ajoympäristössä suoritettava javascript koodi voi toimia kahdella eri periaatteella. Nämä periaatteet ovat estävä ja estoton suoritus. Ohjelmistokoodi on estävää, jos sovelluksen suorituksessa päädytään tilanteeseen, jossa Node.js prosessin täytyy odottaa ei-javascript koodilla toteutettua operaatiota ennen kuin javascript koodin suoritusta voidaan jatkaa. Tällaista estävää ohjelmistokoodia voidaan päätyä ajamaan esimerkiksi tiedostojärjestelmän eli I/O:n operaatioiden kohdalla. Estävä koodi suoritetaan siis synkronisesti. Node.js perusperiaatteiden mukaan kaikista sen vakiokirjaston tarjoamista metodeista on tarjolla myös estottomat toteutukset. Nämä estottomat toteutukset suoritetaan asynkronisesti ja niiden käyttäminen on suositeltu tapa toteuttaa Node.js sovelluksia. Estävän ja estottoman ohjelmistokoodin periaatteiden seuraaminen on erityisen tärkeää, koska javascript koodin suoritus Node.js ajoympäristössä on yksisäkeinen. Täten estottomalla koodilla saadaan mahdollistettua suuri suorituskykyhyöty [20].

Estottomasta koodista saatu suorituskykyhyöty perustuu Node.js tapahtumasilmukkaan. Vaikkakin Node.js javascript suoritus on yksisäikeinen, niin tapahtumasilmukka sallii estottomien I/O operaatioiden suorittamisen lykkäämällä ne käyttöjärjestelmän ytimen vastuulle aina kuin se on mahdollista. Käynnistyessään Node.js initialisoi tapahtumasilmukan ja prosessoi sille annetun scriptin. Alla olevassa kaaviossa on kuvattu yksinkertaistettu rakenne tapahtumasilmukan suoritusjärjestyksestä. Kuva pohjautuu Node.js dokumentaation kuvaan [21].



Kuva 3.4 *Node.js tapahtumasilmukan suoritusjärjestys (Node.js event-loop)*

Jokaisella kuvassa 3.4 kuvatulla vaiheella on oma first-in, first-out (FIFO) jononsa, joka sisältää kyseisen vaiheen takaisinkutsut, jotka tapahtumasilmukan tulee suorittaa. Tapahtumasilmukka siirtyy seuraavaan vaiheeseen, kun edellisen vaiheen FIFO jono on tyhjennetty tai takaisinkutsujen maksimimäärä saavutettu. Jokaisen tapahtumasilmukan suorituksen välissä Node.js ajoympäristö tarkistaa onko asynkronisia I/O yhteyksiä tai ajastimia auki, jos kumpiakaan ei havaita olevan, niin ympäristö lopettaa suorituksen siististi [21].

Node.js ympäristö mahdollistaa javascriptin käytön monenlaisten sovellusten kirjoittamisessa. Yleisin käyttötapaus on todennäköisesti http -palvelimen toteutus. Kuvassa 3.5 on esitelty yksinkertaisen http -palvelimen esimerkkitoteutus. Kuvan koodiesimerkki on lainattu suoraan Node.js verkkosivuilta [22].

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Kuva 3.5 *Node.js esimerkkikoodi (https://nodejs.org/en/about/)*

Kuvassa 3.5 ensin luodaan http muuttuja, johon ladataan ympäristön vakiomoduuleihin kuuluva http. Käytettävän IP -osoitteen ja portin määrittelyn jälkeen luodaan http -palve-

lin kutsulla http.createServer. Tämän metodin takaisinkutsu määrittelee suoritettavan sovelluskoodin silloin, kun palvelimelle otetaan yhteys. Tässä tapauksessa koodi palauttaa kutsujalle onnistumista merkkaavan tilakoodin 200 ja merkkijonon ”Hello world\n”. Lopuksi esimerkkikoodissa vielä käynnistetään http -palvelin kuuntelemaan aiemmin määriteltyä IP -osoitetta ja porttia. Koska kyseinen sovellus kuuntelee sisään tulevia yhteyksiä (kuva 3.4 poll), se jää suorittamaan Node.js tapahtumasilmukkaa eikä poistu automaattisesti suorituksesta.

npm

Npm on Node.js varten luotu paketinhallintasovellus. Paketinhallinnan tarkoituksena on tarjota sovelluskehittäjille mahdollisuus jakaa luomaansa ohjelmakoodia valmiiksi paketoituina moduuleina, jotta toiset kehittäjät voisivat hyötyä heidän työstään. Npm tarjoaa julkisen rekisterin, joka toimii avoimen lähdekoodin pakettien jakoalustana. Rekisteriä hyödyntäessään sovelluskehittäjä voi käyttää npm sivustoa pakettien etsimeen. Valmiita paketteja on tarjolla muun muassa front-end kehitykseen, mobiilisovelluksia varten ja robottien ohjelmointiin [23].

Npm tarjoaa komentorivityökalun, jonka avulla kehittäjän on mahdollista hallita Node.js projektissaan käyttämiään paketteja, eli moduuleita. Alustettaessa uuden projektin tulee sitä varten luoda package.json tiedosto. Package.json kuvaa kyseisen projektin moduuliriippuvuudet niiden versionumeroa myöten. Tämän ansiosta projektin kääntäminen helpottuu, koska muiden projektin parissa työskentelevien tarvitsee vain käyttää kertaalleen määriteltyä package.json tiedostoa. Täten he saavat käyttöönsä juuri oikeista moduuleista koostuvat ympäristön projektin ajamista varten [23].

Package.json tiedosto tukee kaksien erilaisten riippuvuuksien listaamista. Perusriippuvuudet ovat paketteihin, joita kehitettävä sovellus käyttää jatkuvasti, ja jotka tulevat löytymään myös sovelluksen tuotantoversiosta. Näiden lisäksi package.json tiedostoon on mahdollista määritellä erikseen kehitysvaiheessa hyödynnettäviä paketteja, joita ei tulla ottamaan käyttöön tuotantoversiossa. Kehitysajan riippuvuudet ovatkin pääosin kehityksen apuna käytettäviä ja testauksessa hyödynnettäviä paketteja. Esimerkki tällaisesta paketista on moduuli joka simuloi tietokannan käyttäytymistä. Käyttämällä simuloitua tietokantaa ei sovelluskehittäjän tarvitse asentaa erillistä tietokantasovellusta sovelluksensa käyttämiseksi ja testaamiseksi.

3.2.4 REST

Representational State Transfer eli (REST) on Roy Fielding:n väitöskirjassaan esittelemä arkkitehtuurityyli, joka on suunniteltu hajautettuja hypermediajärjestelmiä varten [10]. Pohjana REST arkkitehtuurityylille on käytetty monia verkkopohjaisia arkkitehtuurityylejä tehden siitä hybridi ratkaisun, johon on vielä lisätty sen perustana käytettyihin arkki-

tehtuurityyleihin kuulumattomia rajoitteita [10]. Näitä lisättyjä arkkitehtuurillisia rajoitteita on kuusi kappaletta, ja ainoastaan sellaisia palveluita jotka noudattavat näitä kaikkia voidaan kutsua RESTful periaatteet täyttäviksi [10].

Asiakas-palvelin

Asiakas-palvelin arkkitehtuurimalli perustuu huolenaiheiden erottamisen periaatteelle [10]. Erottamalla käyttöliittymän (asiakas) tiedon tallennuksesta (palvelin), pystymme parantamaan käyttöliittymän siirrettävyyttä eri alustoille ja samalla parannamme skaalautuvuutta yksinkertaistamalla palvelimen komponentteja [10]. Tämä erotus sallii molempien komponenttien kehittämisen itsenäisinä kokonaisuuksinaan [10].

Tilattomuus

Tilattomuuden rajoite asiakas-palvelin mallin arkkitehtuurissa tarkoittaa, että kaiken kommunikaation näiden kahden osan välillä on oltava tilatonta [10]. Tämän mukaan jokainen asiakkaalta palvelimelle lähetetty viesti sisältää kaiken kyseisen viestin käsittelemiseen tarvittavan tiedon, eikä viesti saa täten olla riippuvainen palvelimelle tallennetusta kontekstista [10]. Tilattomuuden rajoitteen ansiosta arkkitehtuurin näkyvyys, luotettavuus ja skaalautuvuus paranevat [10].

Cache

Verkon tehokkuuden parantamiseksi arkkitehtuuriin lisätään cache rajoite [10]. Tämä rajoite vaatii, että palvelimelta asiakkaalle palautettu tieto sisältää tiedon siitä, että voiko asiakas cachettaa tämän tiedon myöhempiä pyyntöjä varten [10]. Jos tieto on mahdollista cachettaa, niin asiakas voi käyttää cachetettua versiota pyynnön vastauksesta sen sijaan, että se lähettäisi pyynnön uudestaan palvelimelle [10]. Cache rajoitteen ansiosta on mahdollista osittain tai kokonaan välttyä joidenkin viestin välittämiseltä asiakkaan ja palvelimen välillä [10]. Tämä puolestaan johtaa parantuneeseen tehokkuuteen, skaalautuvuuteen ja loppukäyttäjälle näkyvään latenssien pienenemiseen [10]. Cachen käyttämisen kanssa on hyväksyttävä, että asiakkaalla käytössä oleva tieto ei välttämättä aina ole sen viimeisin versio [10].

Yhtenäinen käyttöliittymä

Keskeisin REST arkkitehtuurityylin muista verkkopohjaisista arkkitehtuurityyleistä erottava ominaisuus on komponenttien välisten rajapintojen yhtenäisyyden painottaminen [10]. Komponenttien rajapintojen yleistäminen yksinkertaistaa koko järjestelmän arkkitehtuuria ja parantaa vuorovaikutusten näkyvyyttä [10]. Tämän periaatteen ansiosta tarjotut palvelut ja niiden toteutukset on eriytetty toisistaan. Negatiivisen puolena tästä rajoitteesta seuraa, että tieto siirretään standardoidussa muodossa, sen sijaan että jokainen sovellus voisi hyödyntää omaan tarpeeseensa optimaalisinta tapaa [10]. Yhtenäisen käyt-

töliittymän saavuttamiseksi on määritelty neljä rajapintaan vaikuttavaa rajoitetta: resurssien tunnistaminen, resurssien manipulointi esitysten kautta, itsekuvaavat viestit ja hypermedian käyttö sovelluksen tilan hallinnassa [10].

Kerroksinen järjestelmä

Kerroksisen järjestelmän arkkitehtuurityyliin kuuluu, että koko järjestelmä on mahdollista rakentaa hierarkkisista kerroksista [10]. Tällaisessa arkkitehtuurissa yksikään komponentti ei ”näe” kauemmaksi, kuin niiden komponenttien tasolle joiden kanssa se toimii [10]. Asettamalla tällaisen rajauksen, pystymme rajoittamaan koko järjestelmän monimutkaisuutta ja lisäämme komponenttien riippumattomuutta [10]. Kerroksisen järjestelmän negatiivisena puolena on järjestelmän kasvava latenssi, joka voi näyttäytyä loppukäyttäjälle tehottomuutena [10].

Koodia tilauksesta

Viimeinen REST arkkitehtuurityylin rajoite on koodia tilauksesta, joka mahdollistaa asiakkaiden lataavan toiminnallisuuttaan laajentavaa koodia palvelimelta [10]. Rajoitteen tarkoituksena on yksinkertaistaa asiakastoteutuksia, koska niiden ei täten täydy etukäteen toteuttaa kaikkea toiminnallisuutta [10]. Vaikkakin toiminnallisuuden lataaminen parantaa järjestelmän laajennettavuutta, niin samalla se vähentää näkyvyyttä ja täten tämä rajoite onkin valinnainen REST arkkitehtuurin toteutuksissa [10].

3.2.5 Relaatiotietokanta

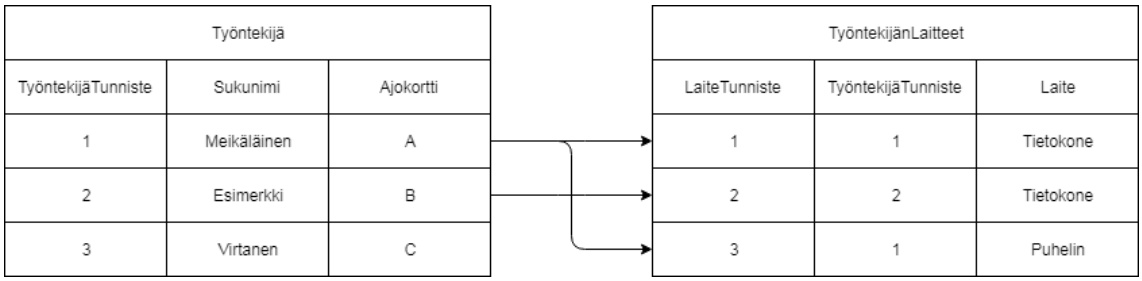
Relaatiotietokanta on relaatiomallia hyödyntävä tietokanta, jonka periaatteet Codd esitteli artikkelissaan [4]. Relaatiotietokanta perustuu ajatukseen, että kaikki sen sisältävä tieto on järjestettynä monikoihin, jotka on mahdollista ryhmitellä relaatioiksi [4].

Monikolla tarkoitetaan tunnistettavaa tietuetta, joka voi koostua monista yksittäisistä arvoista [4]. Tietueen arvot voivat sisältä minkälaista tietoa tahansa. Kuvassa 3.5 on esitetty kuvitteellinen taulu, joka sisältää tietueita työntekijöistä.

Tunniste	Sukunimi	Ajokortti
1	Meikäläinen	A
2	Esimerkki	B
3	Virtanen	C

Kuva 3.6 *Tietueiden käyttöä esittelevä taulu*

Relaatiotietokannan taulut koostuvat kolumneista ja riveistä. Kolumnit koostavat taulun otsikon ja kuvaavat minkälaisia kenttiä yksittäisillä taulun tietueilla voi olla. Rivit puolestaan kuvaavat tauluun sijoitettuja tietueita, joiden arvoja on mahdollista hakea taulun kolumnien avulla. Relaatioiden hyödyt korostuvat, kun käsiteltävää tietoa on enemmän ja yksittäisten tietueiden välille voidaan luoda suoria yhteyksiä. Kuva 3.6 esittelee esimerkin, jossa tietoa sijaitsee kahdessa taulussa, ja näiden taulujen välisellä relaatiolla saadaan luotua lisätietoa.



Kuva 3.7 Kahden taulun välisen relaation hyödyntäminen

Relaatioiden ansiosta yksittäisten taulujen rakenne voidaan pitää mahdollisimman yksinkertaisena. Kuvassa 3.6 esitetyn TyöntekijänLaitteet -taulun ansiosta Työntekijä -taulussa ei tarvitse olla kolumneja, joissa olisi tieto siitä, että minkälaisia laitteita yksittäisillä työntekijöillä on. Jos tieto työntekijöiden laitteista olisi osana Työntekijä -taulua, niin taulun rakennetta jouduttaisiin aina laajentamaan, kun työntekijöille tulee saataville uusia laitteita. Erillisen taulun ansiosta Työntekijä -taulu käsittää puhtaasti vain työntekijöiden tietoja. Tämän periaatteen mukaisesti on suhteellisen helppoa pitää kirjaa työntekijöihin liittyvästä tiedosta tallentamalla se tiedon omien ominaisuuksien mukaiseen tauluun.

4. TOTEUTUS

Tässä luvussa esitellään asiakkuudenhallintajärjestelmä, joka toteutettiin tämän työn perustana toimineen projektin aikana. Toteutettu ratkaisu kuvataan järjestelmän arkkitehtuurin tasolla. Tehdyt arkkitehtuurivalinnat kuvataan UML -kaavioin. Lisäksi luku sisältää toteutuksen tarjoamien verkkorajapintojen kuvaukset.

4.1 Valmiit asiakkuudenhallintasovellukset

Asiakkuudenhallintasovellus ja sen toimivuus on nykypäivänä monelle yritykselle elinehto. Tästä johtuen markkinoilla on monia yrityksiä, jotka myyvät tuotteinaan valmiita asiakkuudenhallintajärjestelmiä. Vaikka työn tarkoituksena on kehittää tilaajalle heidän tarpeisiinsa sopiva asiakkuudenhallintajärjestelmä, niin olisi vastuutonta olla tutustumatta markkinoiden valmiisiin ratkaisuihin. Tässä kappaleessa tehdään lyhyt katsaus kahteen valmiiseen asiakkuudenhallintajärjestelmään ja verrataan niiden ominaisuuksia kappaleessa 3.1.2 listattuihin liiketoimintavaatimuksiin.

Salesforcen Sales Cloud on pilvipalveluna tarjottava asiakkuudenhallintajärjestelmä. Järjestelmän hinnoittelu perustuu käytössä oleviin ominaisuuksiin ja palvelua hyödyntävien käyttäjien lukumäärään. Palvelu tarjoaa tarvittavat ominaisuudet asiakastietojen hallintaan, mahdollisuuden hoitaa laskutus lisämaksua vastaan ja web-palveluna rajapinnan, jotta palveluun on mahdollista integroitua omista järjestelmistä. Valitettavasti palvelu ei tarjoa mahdollisuutta tallentaa omaa tietoa asiakaskohtaisesti myöhempää analytiikkaa varten. Tämän johdosta, jos Salesforce Sales Cloud:a käytettäisiin osana ratkaisua, pitäisi analytiikkaa ja käyttöoikeuksien hallintaa varten tehdä silti oma toteutus [27].

Zoho CRM on toinen vertailuun valittu asiakkuudenhallintajärjestelmä. Myös Zoho tarjotaan verkkoselaimessa käytettävänä pilvipalveluna. Hinnoittelumalli on samankaltainen kuin Salesforcen Sales Cloud tuotteella, eli palvelun käytöstä maksetaan ominaisuuksien ja käyttäjien määrän mukaisesti. Zoho tarjoaa mahdollisuuden asiakastietojen hallintaan, muokattavuutta siihen mitä tietoa palvelussa haluaa hallita ja lisäksi tarjolla web-palveluna tarjottava rajapinta. Valitettavasti suoraa laskutusmahdollisuutta ei tuotteesta löydy, joten laskutusintegraatio pitäisi toteuttaa itse, jos Zoho olisi valittu osaksi asiakkuudenhallinnan ratkaisua [33].

Molemmat tarkastellut asiakkuudenhallintasovellukset on selkeästi suunnattu tukemaan myynnin ja markkinoinnin kautta tapahtuvaa ns. perinteisempää asiakkuudenhallintaa. Tarkemman mietinnän myötä tämä ei osoittautunut mitenkään yllättäväksi. Maailmasta löytyy lukematon määrä yrityksiä, joiden tarpeisiin molemmat mainituista tuotteista riittävät. Tällaiselle suurelle joukolle yrityksiä vaikuttaisi olevan liiketoiminnan kannalta paras ratkaisu tarjota mahdollisimman yleismaailmallisia ratkaisuja. Käytännössä näitä

tuotteita voidaan kuitenkin pitää hyvänä referenssinä, kun toteutetaan asiakkuudenhallintasovellusta uudelle järjestelmälle. Tärkeimpänä huomiona voidaan pitää valmiiden ratkaisujen sopimattomuutta tilanteeseen, jota varten niitä ei ole suunniteltu. Valmiin tuotteen valitseminen, ja sen sovittaminen haluttuun muottiin voi aiheuttaa paljon enemmän töitä kuin oman ratkaisun tekeminen tyhjistä. Tämä vahvistaa ajatusta, että oman asiakkuudenhallintasovelluksen toteutus on tässä tapauksessa oikea ratkaisu.

4.2 Asiakkuudenhallintasovellus

Asiakkuudenhallintasovelluksen toteutus koostuu kahdesta erillisestä osasta. Ensimmäinen osa on Node.js ympäristön päälle toteutettu asiakkuudenhallintasovelluksen logiikka. Toinen osa on tiedon tallentamiseen tarkoitettu MariaDB tietokanta. Molemmat osat toteutetaan docker konttien avulla niiden käyttöönoton ja jatkokehittämisen helpottamiseksi.

Logiikkakerroksen docker kontin pohjakuvana käytetään virallista node:9 -kuvaa, joka on saatavilla julkisesta docker hub:sta [8]. Tähän kuvaan on valmiiksi asennettuna kaikki Node.js 9 -versiota varten tarvittavat riippuvuudet sekä Node.js versio 9. Kokonainen kontin määrittely on listattuna alla.

```
FROM node:9
ADD package.json package.json
ADD package-lock.json package-lock.json
RUN npm install
ADD . .
EXPOSE 443
CMD ["npm", "start"]
```

Ohjelma 1. *Asiakkuudenhallintasovelluksen logiikan Dockerfile*

Konttiin lisätään package.json ja package-lock.json, jotka listaavat Node.js sovelluksen riippuvuudet. Nämä riippuvuudet asennetaan konttiin kutsumalla RUN npm install. Tämän jälkeen konttiin lisätään asiakkuudenhallintasovelluksen ohjelmakoodit rivillä ”ADD . .”. EXPOSE komennolla sallitaan kontin avata portti 443, jota käytetään http pyyntöjen kuuntelemiseen. Lopuksi käynnistetään sovellus kutsumalla ”CMD [”npm”, ”start”]. Tämä komento viittaa package.json tiedostossa määriteltyyn komentoon, joka käynnistää asiakkuudenhallintasovelluksen.

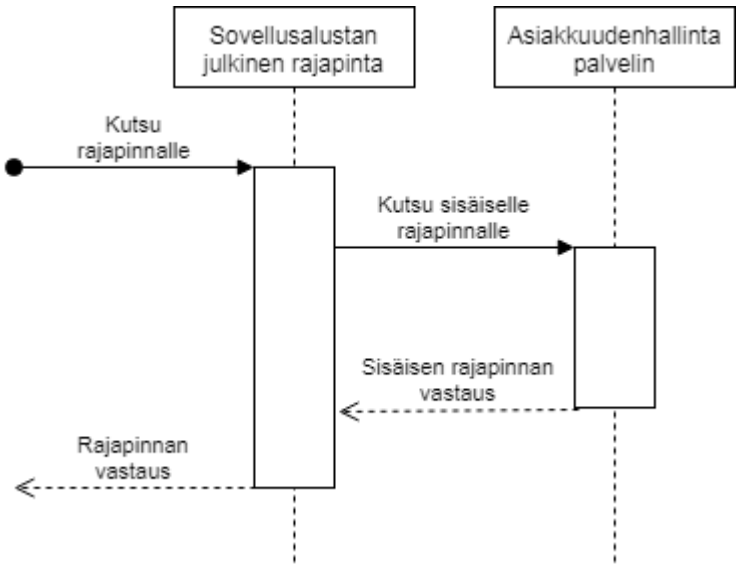
Tiedon tallennuskerroksen docker kontin pohjakuvana käytetään virallista mariadb:10 -kuvaa, joka on saatavilla julkisesta docker hub:sta [8]. Kyseiseen kuvaan on valmiiksi asennettuna kaikki mariadb 10 version riippuvuudet, sekä itse mariadb 10 versio. Koska tallennuskerroksessa käytetään suoraan valmista mariadb docker konttia, niin erilliselle Dockerfile:lle ei ole tarvetta.

Erillisen tallennuskerroksen ansiosta logiikkakerros on mahdollista suunnitella tilattomaksi. Tästä on suora hyöty docker konttiteknologian käytössä. Tilattomalle kontille ei

tarvitse varata mahdollisuutta kirjoittaa levylle pysyvää tietoa. Tämän ansiosta logiikkakerros on suoraan skaalattavissa nostamalla käynnissä olevien kontti-instanssien lukumäärää.

Tilattomuuden vuoksi on luontevaa, että myös asiakkuudenhallintasovellukselta muille järjestelmän osille näkyvä rajapinta on tilaton. Tilattomuus on yksi REST arkkitehtuurin rajoituksista, joten REST on luonnollinen valinta rajapinta-arkkitehtuuria varten. Käytännössä tämän ratkaisun ansiosta asiakkuudenhallintasovelluksen ominaisuuksia hyödyntävät järjestelmän osat eivät tiedä erillisistä logiikka ja tallennuskerroksista. Niille tämä palvelu näyttäytyy vain yhtenä kutsuttavissa olevana rajapintana.

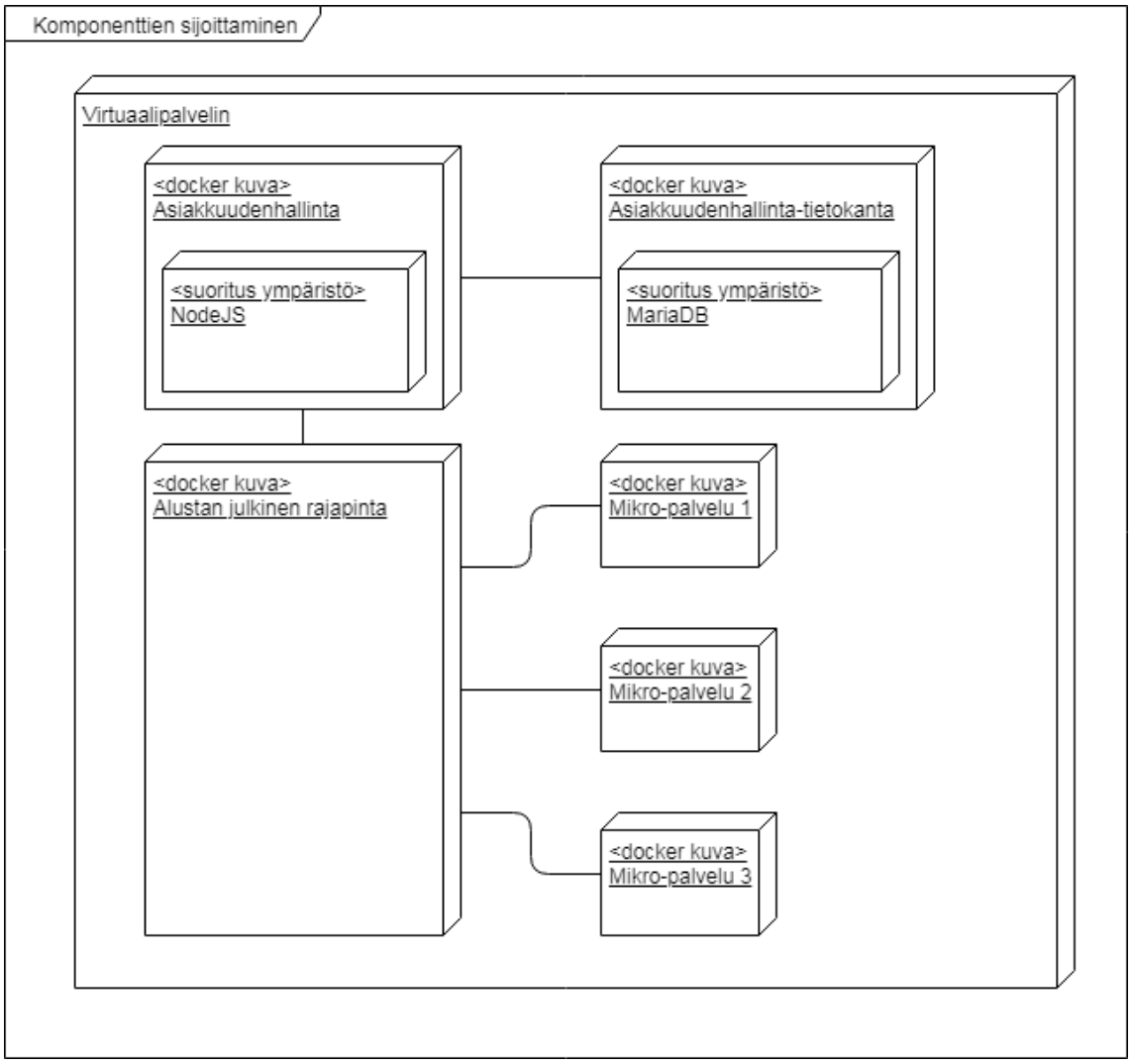
Kuten kappaleessa 3.1 on kuvattu, näyttäytyy Vektor3 -järjestelmä kaikille käyttäjille vain yhtenä sovellusalustan rajapintana. Käytännössä tämä rajapintatoteutus on järjestelmän ainoa osa, joka kommunikoi asiakkuudenhallintasovelluksen kanssa. Tämä kommunikaatio on kuvattu sekvenssikaaviolla kuvassa 4.1.



Kuva 4.1 *Sekvenssikaavio asiakkuudenhallintapalvelimen kutsumisesta*

Kaikki käyttäjien järjestelmään tekemät pyynnöt aiheuttavat kutsun asiakkuudenhallintasovellukselle. Tämän ansiosta kaikkien kutsujen tekijät tunnistetaan ja kutsut voidaan valtuuttaa. Täten muut järjestelmän osat voivat luottaa siihen, että niille sovellusalustan rajapinnalta saapuvat pyynnöt on jo valtuutettu. Tunnistautumisen ja valtuutuksen rajaaminen tälle tasolle helpottaa myös tulevaisuudessa järjestelmän laajennettavuutta. Lisäksi asiakkuudenhallintasovelluksen rajapintakutsun kytkeminen heti järjestelmään tehtävien kutsujen alkuvaiheeseen mahdollistaa statistiikan keräämisen järjestelmän käytöstä.

Asiakkuudenhallintasovelluksen sijoittumisen suhteessa muihin järjestelmän osiin esitellään sijoituskaaviona kuvassa 4.2.



Kuva 4.2 *Sijoituskaavio sovellusalustan arkkitehtuurista*

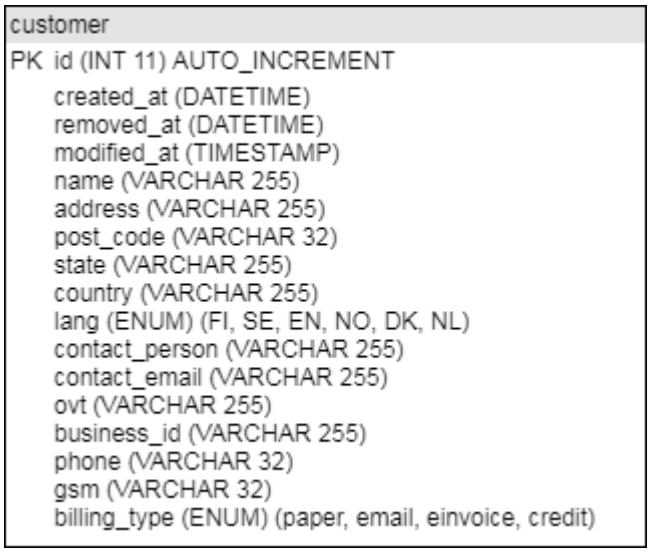
Sovellusalustaa käytetään sen tarjoaman julkisen rajapinnan kautta. Julkinen rajapinta hyödyntää asiakkuudenhallintasovellusta, ennen kuin se lähettää pyyntöjä eteenpäin järjestelmän mikropalveluille.

4.2.1 Tietomalli

Asiakkuudenhallintasovelluksen tietomalli kuvataan käyttäen entiteettejä ja niiden välisiä suhteita. Tässä kappaleessa käydään yksitellen läpi kaikki sovelluksen tiedonhallinnassa käytettävät entiteetit. Lopuksi niistä koostetaan yhteenveto, joka sisältää tiedot niiden keskinäisistä suhteista.

Asiakkuudenhallintasovelluksen tietomalli rakentuu asiakas -konseptin päälle. Asiakastietoa voidaan pitää niin sanottuna juurena kaikelle muulle järjestelmään syötetylle tiedolle. Käytännössä on siis mahdollista yhdistää mikä vain järjestelmästä löytyvä muu entiteetti tiettyyn asiakkaaseen. Ylläpidon ja käytettävyyden helpottamiseksi järjestelmä si-

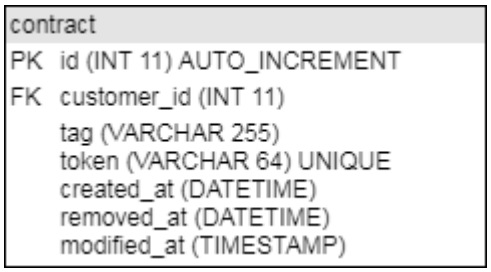
sältää muutaman poikkeuksen tähän sääntöön. Nämä poikkeavat entiteetit voidaan luokitella osaksi järjestelmän perusrakennetta. Yksittäisen asiakkaan tietojen hallintaan käytettävä entiteetti on esitelty tietokantatauluna kuvassa 4.3.



Kuva 4.3 *Asiakastiedon kuvaus tietokantatauluna*

Asiakastieto sisältää tietokenttiä, jotka voidaan luokitella kahteen tyyppiin. Ensimmäinen on perinteinen asiakastieto, jota hyödynnetään asiakasrekisterin ylläpidossa ja laskutuksessa. Tähän luokkaan kuuluvat nimi, osoitetiedot, laskutuksessa käytettävä kieli, yhteyshenkilön tiedot ja ovt. Ovt on laskutettavan asiakkaan verkkolaskutusosoite, jota käyttämällä verkkolaskut saadaan välitettyä suoraan asiakkaalle välityspalveluiden avulla [30]. Toinen osa asiakastiedosta on järjestelmän omaan käyttöön tarkoitettua tietoa, kuten jokaiselle asiakkaalle automaattisesti luotava sisäinen tunnus, luontipäivämäärä ja poistopäivämäärä. Käytännössä palvelusta ei koskaan poisteta tietoa, vaan asiakkaalle kirjoitetaan poistopäivämäärä, ja sen avulla tiedetään, että kyseinen asiakastili on poistettu käytöstä. Lisäksi taulusta löytyy tiedon muokkausajanhetken tallentava kenttä. Tämä kenttä hyödyntää MariaDB tietokannan sisäistä ominaisuutta, joka päivittää automaattisesti tietokantataulun rivin ensimmäisen timestamp tyyppisen kentän arvoa, silloin kun joku rivin arvoista muuttuu [17].

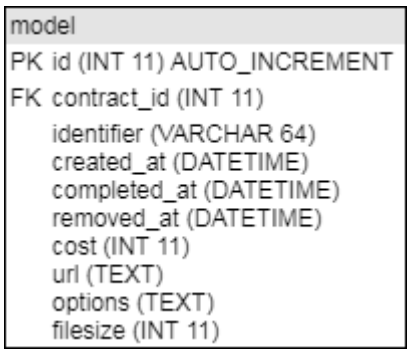
Pelkkä asiakkuuden luominen järjestelmään ei vielä mahdollista sovellusalustan käyttämistä. Lisäksi asiakkaalla pitää olla aktiivisena oleva sopimus. Sopimusentiteettien sisältö esitellään tietokantatauluna kuvassa 4.4.



Kuva 4.4 *Sopimustiedon kuvaus tietokantatauluna*

Sopimustieto sisältää järjestelmän sisäisen tunnisteen jokaiselle sopimukselle. Lisäksi kaikista sopimuksista löytyy vierasavain tieto, joka viittaa siihen asiakkaaseen jonka sopimus on kyseessä. Jokaisella asiakkaalla on mahdollista olla samanaikaisesti useita sopimuksia. Täten yrityksen sisällä eri tahot voivat kehittää sovellusalustan päälle omia ratkaisujaan, ja jokaiseen ratkaisuun liittyvät tiedot on mahdollista pitää erillä toisistaan. Kaikista sopimuksista löytyy tag -kenttä, jonka avulla sopimuksiin on mahdollista liittää metatietoa järjestelmän ylläpitäjän toimesta. Tämä helpottaa sopimusten hallintaa varsinkin siinä tilanteessa, kun samalla asiakkaalla on useita aktiivisia sopimuksia. Sopimuksen token -kenttä sisältää järjestelmänlaajuisesti yksilöllisen merkkijonon, jota sopimuksen tehnyt asiakas voi käyttää kommunikointiin sovellusalustan rajapinnan kanssa. Näiden kenttien lisäksi sopimuksesta löytyy kentät luontiajan ja sopimuksen poistamisajankohdan tallentamista varten. Myös sopimustaulu hyödyntää MariaDB ominaisuutta päivittää automaattisesti timestamp kentän arvo, kun tietokantarivin joku arvo muuttuu. Tälle tiedolle käytetään modified_at saraketta.

Sovellusalustan tärkein ominaisuus on asiakkaiden mallitiedostojen visualisointi. Käsiteltävät mallitiedostot esitetään malli -entiteetteinä tietokannassa. Mallientiteettien sisältö esitellään tietokantatauluna kuvassa 4.5.

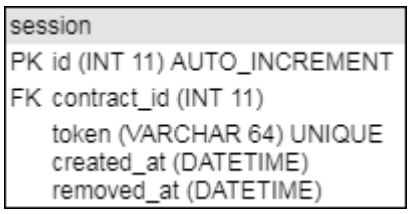


Kuva 4.5 *Mallitiedon kuvaus tietokantatauluna*

Kaikki järjestelmään ladatut mallitiedostot saavat oman järjestelmän sisäisen tunnisteensa. Lisäksi jokainen mallitiedosto on kytkettävissä tietyn asiakkaan sopimukseen contract_id vierasavaimen avulla. Tämän kytkennän ansiosta mallien käyttämisestä sovellusalustalla on mahdollista luoda laskutustieto takaisin tietylle asiakkaalle. Laskutus-

tietoa varten jokainen mallientiteetti sisältää tiedon sen käytöstä perittävästä hinnasta. Sovellusalustalla visualisoitavat 3D -mallit lisätään järjestelmään tarjoamalla ne URL -osoitteen päästä. Tämä URL osoite ja ladatun mallitiedoston koko tallennetaan mallientiteetille myöhempää käyttöä varten. Lisäksi jokaisesta mallitiedostosta tallennetaan sen elinkaareen liittyviä aikaleimoja.

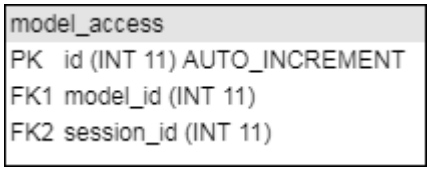
Sovellusalustaa ole tarkoitettu käytettäväksi suoraan asiakkaan sopimuksen myötä haltuunsa saamalla token -avaimella. Kyseisen sopimuskohtaisen avaimen avulla asiakkaiden on mahdollista luoda sovellusalustaan loppukäyttäjiä varten istuntoja. Istuntoentiteettien sisältö esitellään tietokantatauluna kuvassa 4.6.



Kuva 4.6 *Istuntotiedon kuvaus tietokantatauluna*

Jokainen palveluun luotu istunto saa oman palvelunsisäisen yksilöllisen tunnuksensa. Lisäksi jokainen istunto on kytketty contract_id vierasavaimen avulla sen luomisessa käytettyyn sopimukseen. Myös istunnon luontiaika ja poistoaika tallennetaan palveluun. Tärkein osa istuntoa on sen järjestelmän laajuisesti yksilölliseksi määritelty token -merkkijono. Sovellusalustan asiakas syöttää tälle arvon luodessaan uuden istunnon järjestelmään. Tämä on tietoinen suunnitteluratkaisu istuntojen hallinnassa, koska Vektor3 sovellusalusta on tarkoitettu integroitavaksi asiakkaiden omiin järjestelmiin. Lähes kaikki V3 -alustaa hyödyntävät palvelut ovat SaaS -tyylisiä, joten niistä löytyy jo oma istunnon käsite. Täten asiakkuudenhallinta mahdollistaa sovellusalustan asiakkaiden uudelleenkäyttävän istuntotunnuksiaan, jos he niin tahtovat.

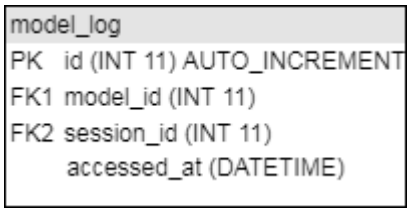
Koska Vektor3 sovellusalusta on tarkoitettu integroitavaksi osaksi asiakkaan omaa järjestelmää niin ei voida olettaa, että asiakkaan omalla järjestelmällä olisi vain yksi käyttäjäryhmä ja kaikilla kyseisen järjestelmän käyttäjillä olisi pääsyoikeudet samoihin resursseihin. Tämän vuoksi asiakkuudenhallintasovellus tarjoaa istuntoihin kytketyn mallitiedostojen pääsyoikeuden hallinnan. Mallien pääsyoikeudenhallinnassa hyödynnettävien entiteettien sisältö esitellään tietokantatauluna kuvassa 4.7.



Kuva 4.7 *Mallitiedostojen pääsyoikeudenhallinta tietokantatauluna*

Pääsyoikeuksien hallintaan käytetyn tietokantataulun rakenne on varsin yksinkertainen. Istunnon luomisen yhteydessä asiakas lähettää palveluun listan URL osoitteista, joihin viittaaviin mallitiedostoihin luotavalla istunnolla on pääsyoikeudet. Rajoitteena pääsyoikeuksien hallinnalle on vaatimus, että mallitiedostot on jo aiemmin lisätty järjestelmään, jotta niiden tunnuksen käyttäminen pääsyoikeustaulun vierasavaimena on mahdollista.

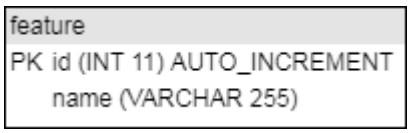
Järjestelmän jatkokehittämisen ja käytön valvomisen mahdollistamiseksi on tärkeää saada tietoa, siitä miten sovellusalustan asiakkaiden käyttäjät sitä hyödyntävät. Asiakkuudenhallintasovelluksen ensimmäisessä versiossa kerätään vain lokitietoa, siitä milloin mitäkin istuntoa on viimeksi käytetty mallitiedostojen hakemiseen palvelusta. Tämä tieto tallennetaan malli-loki entiteetteinä, joiden rakenne on esitelty tietokantatauluna kuvassa 4.8.



Kuva 4.8 *Mallien käyttöloki tietokantatauluna*

Käyttölokiin kirjataan mallitiedostoa pyytäneen käyttäjäistunnon tunnus vierasavaimena, ja pyynnön kohteena olleen mallitiedoston tunnus toisena vierasavaimena. Näiden lisäksi jokaiseen entiteettiin tulee mukaan päivämäärä ja kellonaika, milloin pyyntö on suoritettu.

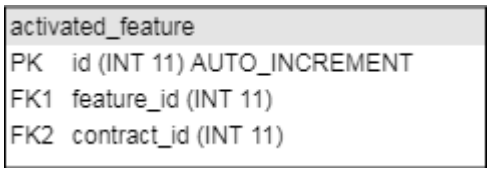
Asiakkuudenhallintasovelluksen ensimmäisessä versiossa tuetaan vain mallitiedostoresurssien käyttämistä. Tästä huolimatta tietomallissa on hyvä varautua järjestelmän laajentamiseen uusilla ominaisuuksilla. Kaikki järjestelmän asiakkaille tarjoamat ominaisuudet kuvataan ominaisuusentiteetteinä tietokantatauluna kuvassa 4.9.



Kuva 4.9 *Järjestelmän ominaisuudet tietokantatauluna*

Ominaisuudet tietokantataulu sisältää yksilöllisen tunnisteen ja nimikentän jokaiselle ominaisuudelle.

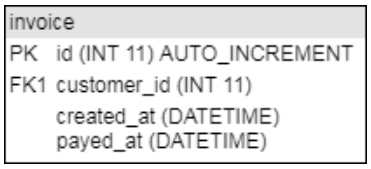
Asiakkaiden käytössä olevat ominaisuudet kuvataan aktivoitu ominaisuus -entiteetteinä tietokantatauluna kuvassa 4.10.



Kuva 4.10 *Aktivoitu ominaisuus tietokantatauluna*

Jokaisella aktivoitu ominaisuus -entiteetillä on järjestelmän sisäinen uniikki tunnisteensa. Lisäksi siltä löytyy kaksi vierasavainta. Ensimmäinen vierasavain osoittaa ominaisuus -taulun entiteetin tunnisteeseen. Toinen vierasavain osoittaa sopimus -taulun entiteetin tunnisteeseen.

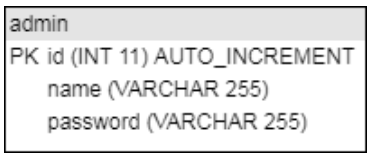
Asiakkaita veloitetaan automaattisesti sovellusalustan käytöstä heidän hyödyntämiensä resurssien mukaisesti. Kaikista laskuista tallennetaan asiakkuudenhallintasovelluksen tietokantaan entiteetti. Laskuentiteetin sisältö on kuvattu tietokantatauluna kuvassa 4.11.



Kuva 4.11 *Lasku tietokantatauluna*

Kaikki laskut saavat yksilöivän tunnisteen niiden luontihetkellä. Lisäksi jokainen lasku sisältää viiteavaimena asiakastunnisteen, jotta tietyn asiakkaiden laskuhistoria on tarkasteltavissa. Näiden tietojen lisäksi kaikille laskuille tallennetaan niiden luomis- ja maksuhetket päivämäärä & kellonaika yhdistelminä.

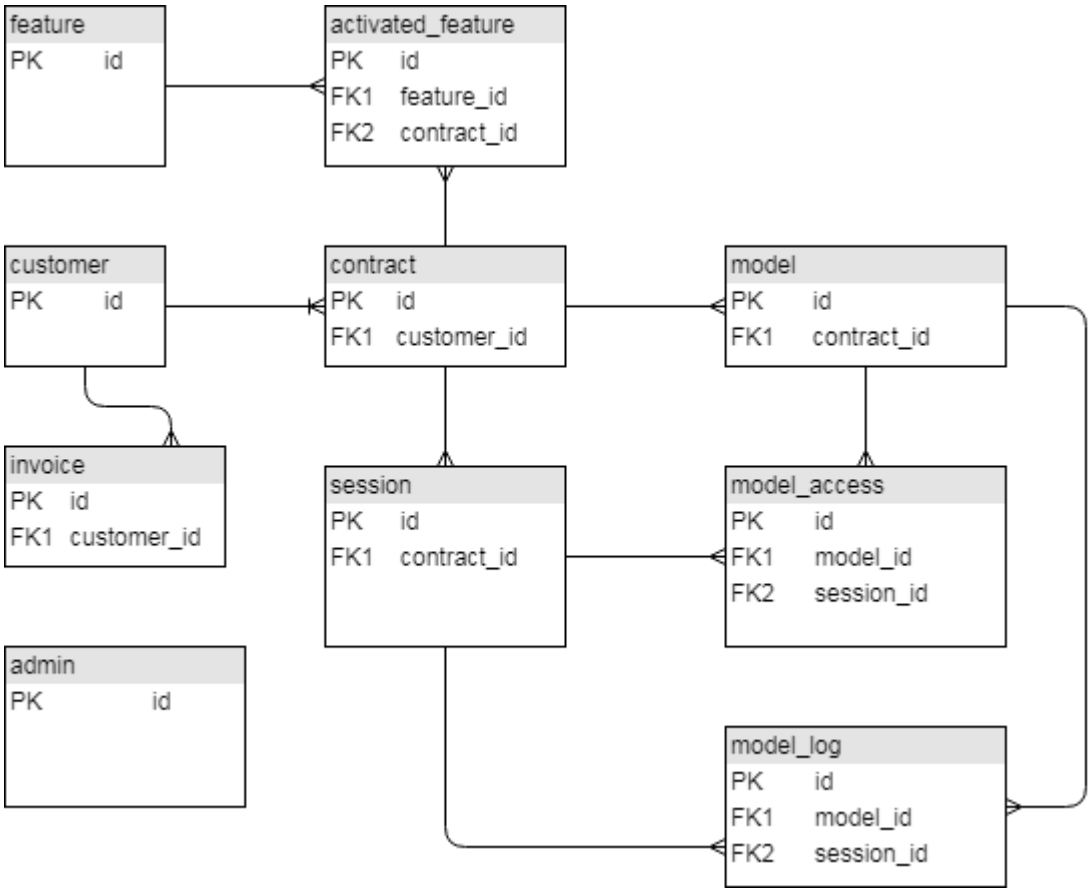
Asiakkuudenhallintasovelluksen käyttöliittymän toteutusta varten tietomallista löytyy järjestelmänvalvoja tunnuksille oma tietokantataulunsa. Tämän taulun rakenne on kuvattu kuvassa 4.12.



Kuva 4.12 *Järjestelmänvalvoja tiedot tietokantatauluna*

Kaikilla järjestelmänvalvoja tunnuksilla on tietokannassa oma yksilöivä tunnuksensa. Tämän lisäksi tunnukset koostuvat nimi ja salasana -kentistä. Salasana suolataan ja siitä lasketaan tiiviste ennen kuin se tallennetaan tietokantatauluun. Salasanan suolaaminen on toimenpide, jossa salasana-merkkijonoon lisätään muita merkkejä. Tämän ansiosta salasanan arvaaminen esimerkiksi sateenkaari -taulujen avulla on paljon vaikeampaa [12].

Tietomallin sisällön kokonaiskuvan ja erilaisten relaatioiden hahmottamiseksi koko asiakkuudenhallintasovelluksen tietomalli on esitelty kuvassa 4.13.



Kuva 4.13 *Asiakkuudenhallintasovelluksen tietomalli*

Tietomallin kuvaus sisältää vain relaatioiden kuvaamiseen tarvittavat tunnisteet ja vierasavaimet. Asiakkuudenhallintasovellus tukee vain 3D -malliresursseja tässä vaiheessa. Muiden resurssien lisääminen järjestelmään on kuitenkin varsin helppoa. Lisäksi muidenkin resurssien suhteen on mahdollista käyttää samanlaisia relaatioita kuin model, model_access, model_log ja session taulujen välillä on.

4.2.2 Arkkitehtuuri

Asiakkuudenhallintasovellus on suunniteltu käyttäen kerrosarkkitehtuuria. Kerrosarkkitehtuurin periaatteiden mukaan tiedon tallentaminen, sovelluslogiikka ja sovelluksen käyttöliittymä erotetaan toisistaan riippumattomiksi kerroksiksi. Kerrosten lukumäärälle ei arkkitehtuurissa aseteta rajoitteita, mutta yksinkertaisissa sovelluksissa käytetään yleensä kolmea aiemmin listattua kerrosta. Kyseiset kerrokset kommunikoivat toistensa kanssa hyvin määriteltyjen rajapintojen kautta. Tämän ansiosta jokaisen kerroksen toteutusta on mahdollista kehittää erillään, riippumatta suoraan toisten kerrosten toteutuksesta. Lisäksi kommunikaation toteuttaminen rajapintojen kautta mahdollistaa yksittäisen kerroksen toteutuksen täydellisen vaihtamisen ilman, että sovelluksen muihin kerroksiin tarvitsee tehdä muutoksia.

Tämän työn laajuuteen tehtiin alusta asti rajaus, että esityskerroksen toteuttaminen jätettäisiin myöhempään ajankohtaan. Osana työn laajuutta oli kuitenkin rajapintojen suunnitteleminen ja toteuttaminen valmiiksi, jotta esityskerroksen toteutus olisi aikanaan mahdollisimman yksinkertaista.

Tallennuskerroksen toteuttamiseen käytetään valmista komponenttia, kuten kappaleessa 4.2 on kuvattu. Tämän komponentin sisäiseen arkkitehtuuriin ei tämän työn puitteissa tutustuta tarkemmin. Valittu tallennusratkaisu, MariaDB, tarjoaa valmiin rajapinnan kommunikointia varten. Tarjotun rajapinnan hyödyntäminen suoraan olisi varsin työlästä, joten sen käyttäminen abstrahoidaan valmiin tietokanta-ajurin taakse.

Asiakkuudenhallintasovelluksen logiikkakerros toteutetaan käyttäen Node.js ympäristöä ja se toteutetaan javascript:llä. Sovelluksessa hyödynnetään npm pakettienhallinnan avulla käyttöönotettuja valmiita moduuleita, jotta kirjoitettavan sovelluskoodin määrää saadaan laskettua. Käytettävät moduulit ovat:

- express
- body-parser
- mysql

Express on minimalistinen ja joustava Node.js palvelinsovellusalusta [9]. Asiakkuudenhallintasovelluksessa expressiä käytetään tarjoamaan sovelluksen verkkopalvelintoteutus, jonka avulla sovelluksen tarjoamat rajapinnat saadaan ulkopuolisten toimijoiden kutsuttaviksi.

Body-parser on Node.js moduuli, joka tarjoaa toiminnallisuuden http pyynnön sisällön parsimiseksi. Kaikki asiakkuudenhallintasovelluksen rajapintojen kutsut käyttävät JavaScript Object Notation (JSON) rakennetta tiedon välittämiseen. Body-parser moduulin avulla palvelimelle sisään tulevan http pyynnön kirjainjono muotoinen sisältö saadaan muutettua sovelluksen helpommin käsiteltäväksi natiiviksi javascript olioksi.

Mysql moduuli tarjoaa tallennuskerroksen ja logiikkakerroksen väliseen kommunikointiin tarvittavan tietokanta-ajurin. Sovelluksen tallennuskerroksen toteutuksessa käytetty MariaDB on yhteensopiva sen edeltäjän MySQL tietokannan kanssa [16]. Tämän johdosta sovelluksessa on mahdollista käyttää näennäisesti eri tietokannalle tarkoitettua tietokanta-ajuria.

Logiikkakerroksen sisäinen toteutus koostuu javascript moduuleista, jotka on toteutettu tämän työn aikana. Moduulit on eroteltu sovelluksen kansiorakenteessa toiminnallisuutensa mukaan. Sovellus käynnistetään ajamalla app.js tiedosto Node.js ympäristössä. Käynnistettäessä asiakkuudenhallintasovellus alustetaan ja käynnistetään verkkopalvelin automaattisesti. Lisäksi tiedosto määrittelee kappaleessa 4.2.3 tarkemmin käsiteltyjen rajapintojen sisääntulopisteet.

Logiikkakerroksen sisäinen rakenne

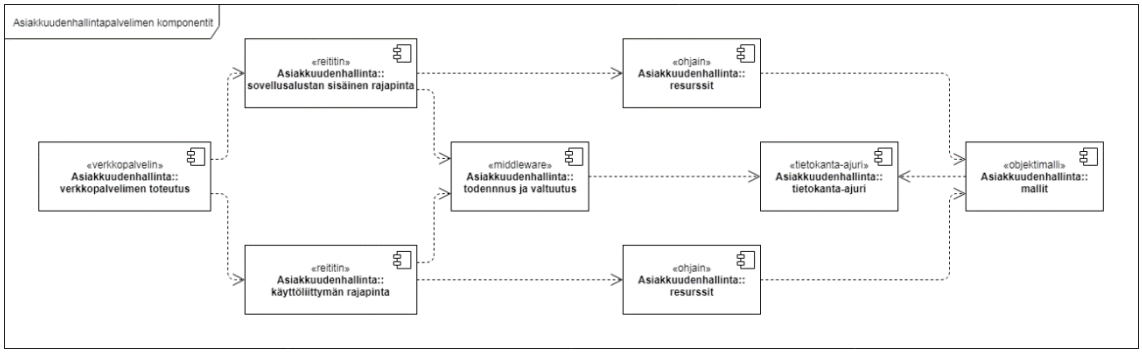
Seuraava taso toteutuksessa on asiakkuudenhallintasovelluksen tarjoamien rajapintojen reitittimet. Reitittimet hyödyntävät kahta eri tyyppistä toteutusta. Ensimmäinen on tunnistautumisen ja valtuutuksen hoitava middleware -moduuli. Toinen on yksittäistä resurssia kuvaava reititin, eli ohjainmoduuli.

Tunnistautumisesta ja valtuutuksesta vastaava moduuli toimii rajapintojen reitittimien ja niiden kutsumien ohjain -moduulien välissä. Asiakkuudenhallintasovelluksen periaatteiden mukaisesti kaikki rajapinnoille tulevat pyynnöt täytyy pystyä tunnistamaan ja kohdistamaan tietyn alustan käyttäjän tekemäksi. Täten kaikki pyynnöt käsitellään tämän moduulin toimesta ennen kuin ne siirretään ohjainmoduuleille. Jos sovellusta on kutsuttu epäkelpoa tietoa sisältävällä pyynnöllä, on tämän moduulin vastuulla pysäyttää kutsuketju, ja palauttaa kutsun lähettäjälle virheviesti.

Ohjainmoduulit ovat vastuussa yksittäisten resurssien funktioiden toteutuksista. Kaikki järjestelmä resurssit eivät tue samoja funktioita. Jos tällaista tukematonta funktiota kutsutaan, on ohjainmoduulin vastuulla palauttaa kutsujalle oikea virheviesti. Ohjainmoduulin saadessa resurssin tukeman funktiokutsun, se kutsuu resurssien malli -moduulien tarvittavia toteutuksia.

Mallimoduulit abstrahoivat niiden esittämän tietomallin tietorakenteen muilta sovelluksen osilta. Tämän abstrahoinnin saavuttaakseen mallimoduulit hyödyntävät tallennuskerrosta varten järjestelmään käyttöönotettua tietokanta-ajuria. Tilanteessa, jossa sovelluksen tallennuskerroksen toteutus vaihdettaisiin toiseen, heijastuisivat kaikki muutokset vain mallimoduulien toteutuksiin. Täten ne suojaavat muita logiikkakerroksen osia muutoksilta, ja niiden voidaan ajatella toimivan rajapintana logiikkakerroksen ja tallennuskerroksen välissä.

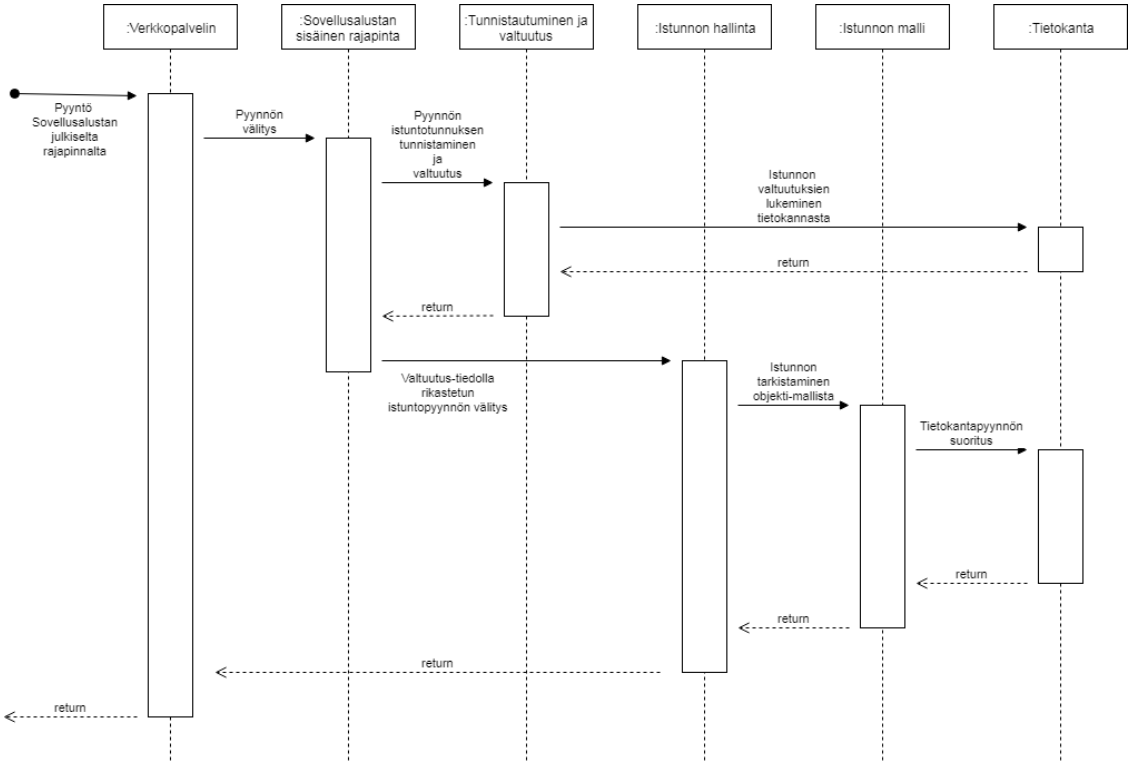
Logiikkakerroksen osien välisistä suhteista on annettu esimerkki kuvassa 4.14.



Kuva 4.14 *Komponenttikaavio asiakkuudenhallintasovelluksen toteutuksesta*

Komponenttikaavio esittää asiakkuudenhallintasovelluksen sisäisen hierarkian mallitiedostojen tietojen suhteen. Mallitiedostoja on mahdollista käsitellä käyttäen kahta eri rajapintaa. Molemmilla rajapinnoilla on omat reitittimensä. Erilliset reitittimet hyödyntävät kuitenkin yhtä ja samaa mallitiedostojen tietomallin tarjoavaa toteutusta.

Kuvassa 4.15 kuvataan sekvenssikaaviona asiakkuudenhallintasovellusten rakenne, ja rakenteen eri osien välisten kutsujen ketju.



Kuva 4.15 *Sekvenssikaavio vastaanotetun pyynnön käsittelystä*

Sekvenssikaavio 4.15 kuvaa asiakkuudenhallintasovelluksen kutsulogiikkaa, kun sen rajapinnalle tulee pyyntö tarkistaa käyttäjän istunnon oikeellisuus. Sovelluksen arkkitehtuurin vuoksi myös istuntojen käsittelyssä tehdään kaksi erillistä tietokantapyyntöä. Ensimmäinen pyyntö tehdään tunnistautumisen ja valtuutuksen yhteydessä. Jos istunto on merkitty poistettavaksi, tai sopimus jonka alaisuudessa istunto on, niin sovellus palauttaa jo tässä vaiheessa virheen. Jos istunto on validi, niin pyyntö jatkuu istunnon hallinnasta vastuussa olevalle ohjaimelle, joka tekee toisen tietokantapyynnön. Tämän pyynnön tuloksena on mahdollista palauttaa tietoa istunnosta. Lisäksi tässä vaiheessa on ohjaimella mahdollisuus kirjata lokitietoa istunnon käytöstä. Lopuksi sovellus palauttaa kutsujalle sen pyytämät tiedot istunnosta.

Testaaminen

Asiakkuudenhallintasovelluksen toteutuksessa noudatetaan TDD:n mukaisia periaatteita. Kaikille yksikkötestattavissa oleville toiminnallisuuksille kirjoitetaan ensin niiden toiminnallisuuden rajat määrittelevät testit. Aluksi kaikki kirjoitetut testit epäonnistuvat. Mutta sovelluksen toiminnallisuuden toteutuksen edetessä yhä suurempi määrä testeistä menee läpi.

Toteutuksessa testit jaetaan kolmeen luokkaan. Aluksi toteutetaan yksikkötestit kaikille sovelluksessa oleville entiteeteille, joka logiikkakerroksessa kuvataan mallein. Nämä testit kattavat entiteettien luonnin, lukemisen, päivittämisen ja poistamisen. Lisäksi kaikki mallit tarjoavat mahdollisuuden lukea kerralla listan entiteettejä.

Toinen testiluokka testaa sovellusta korkeammalta tasolta. Nämä testit kattavat asiakkuudenhallintasovelluksen tarjoaman sisäisen rajapinnan funktiot. Rajapinnan tarjoamat ominaisuudet on esitelty tarkemmin kappaleessa 4.2.3.

Kolmas testiluokka toimii samalla tasolla kuin toinen testiluokka. Näillä testeillä varmistetaan asiakkuudenhallintasovelluksen tarjoaman julkisen rajapinnan toiminnallisuus. Rajapinnan tarjoamat ominaisuudet on esitelty tarkemmin kappaleessa 4.2.3.

4.2.3 Rajapinnat

Asiakkuudenhallintasovellus toteutetaan erillisenä palveluna osana sovellusalustan järjestelmää. Tämän ansiosta kaikki asiakkuudenhallintaan liittyvä sovelluslogiikka on mahdollista sulkea kyseisen palvelun sisään. Kuten kappaleessa 4.2 mainitaan, asiakkuudenhallintaan liittyvä tietokanta on omassa docker kontissaan sijaitseva palvelu. Teoriassa olisi siis mahdollista, että kaikki asiakkuudenhallintaan liittyviä tietoja käyttävät sovellusalustan palvelut voisivat ottaa suoraan yhteyden tähän tietokantaan ja hyödyntäisivät sitä kukin omalla tavallaan. Käytännössä tämä ratkaisu kuitenkin johtaisi tilanteeseen, jossa monet sovellusalustan palvelut käyttäisivät samoja resursseja mahdollisesti eri tavalla. Tämä vaikeuttaisi palvelun ylläpitoa, ja rikkoisi kappaleessa 4.2 mainittua vaatimusta sovellusalustan ominaisuuksien mikropalveluiden luonteesta. Täten on varsin perusteltua, että asiakkuudenhallintasovellus tarjoaa rajatun ja hyvin määritellyn rajapinnan, jonka kautta asiakkuustietoa voidaan hyödyntää sovellusalustan muissa osissa.

Käytännössä asiakkuudenhallintasovelluksen tulee kuitenkin tarjota kaksi erillistä rajapintaa. Näitä rajapintoja tullaan kutsumaan tästä eteenpäin termein ”sisäinen rajapinta” ja ”asiakkuudenhallinnan rajapinta”.

Sisäinen rajapinta

Sisäinen rajapinta on tarkoitettu sovellusalustan muiden palveluiden käyttöön. Käytännössä sisäistä rajapintaa kutsutaan vain sovellusalustan ulkoisen rajapinnan toteutuksesta.

Asiakkuudenhallintasovelluksen sisäisen rajapinnan kutsuminen vaatii erillisen asiakas-sertifikaatin käyttämistä kaikkien suoritettavien pyyntöjen yhteydessä. Asiakas-sertifikaatti on yksi kolmesta SSL sertifikaatin tyypistä. Kaksi muuta ovat palvelimen identiteettisertifikaatti ja todistuksen myöntäjä sertifikaatti, engl. Certificate Authority (CA). Koska käytettävä sertifikaatti on järjestelmän sisäinen, eikä sitä ole tarkoitettu kolmannen osapuolen sovellusten hyödynnettäväksi, pystymme hyödyntämään itse luomaamme CA sertifikaattia asiakas-palvelin sertifikaattien luomiseen [24].

Toinen sisäisen rajapinnan kutsumisen rajoite liittyy sen tarjoamiin resursseihin. Kaikki rajapinnan tarjoamat resurssit linkittyvät sovellusalustan asiakkuuksiin, joten rajapintakutsut käytännössä suoritetaan asiakkaiden puolesta. Tämän vuoksi kaikissa rajapintakutsuissa on oltava mukana joko asiakkuuden tunnistamisessa käytettävä tietue tai asiakkuuteen liittyvän käyttäjäistunnon tietue. Nämä tietueet toimiteen rajapintakutsujen mukana käyttäen niitä varten varattuja http header kenttiä. Asiakkuuden tunnistava tietue tulee lähettää rajapinnalle ”x-customer-key” http header kentän arvona. Käyttäjäistunnon tietue on puolestaan mahdollista lähettää ”x-user-key” http header kentän arvona.

Rajapinnan suunnittelussa on pyritty täyttämään kappaleessa 3.2.4 mainitut REST arkkitehtuurin rajoitteet. Alla sijaitseva taulukko 4.1 esittelee rajapinnan kuvauksen.

Taulukko 4.1 *Asiakkuudenhallintasovelluksen järjestelmän sisäiseen käyttöön tarjoaman rajapinnan kuvaus*

URL polku	http metodi	Kuvaus
/sessions	POST	Luo uuden käyttäjäistunnon
/sessions/:sessionId	HEAD	Palauttaa sessionId muuttujassa kuvatun käyttäjäistunnon tilan
/sessions/:sessionId	DELETE	Poistaa sessionId muuttujassa kuvatun käyttäjäistunnon käytöstä
/models	POST	Luo uuden malliresurssin
/models/:modelId	HEAD	Palauttaa modelId muuttujassa kuvatun malliresurssin tilan
/models/:modelId	DELETE	Poistaa modelId muuttujassa kuvatun malliresurssin käytöstä
/models/:modelId	PATCH	Muokkaa modelId muuttujassa kuvatun malliresurssin sisältöä

Sisäinen rajapinta tarjoaa kaksi resurssia ja metodit niiden hyödyntämiseen. Sessions URL-polut mahdollistavat käyttäjäistuntojen luomisen, käytöstä poistamisen ja niiden tilan tarkistamisen. Käyttäjäistuntojen luominen ja käytöstä poistaminen ovat mahdollisia vain ”x-customer-key” http header:lla varustetuille pyynnöille. Eli käyttäjäistuntoa ei ole mahdollista käyttää uusien käyttäjäistuntojen luomiseen tai olemassa olevien käyttäjäistuntojen poistamiseen käytöstä.

Malliresurssien metodeja on mahdollista kutsua sekä ”x-customer-key”, että ”x-user-key” http header:lla varustetuilla pyynnöillä. Poikkeuksena on malliresurssin poistaminen käytöstä, joka tapahtuu tekemällä DELETE kutsu haluttuun malliresurssiin. Tämä pyyntö vaatii ”x-customer-key” http header:lla tunnistautumisen.

Asiakkuudenhallinnan rajapinta

Asiakkuudenhallinnan rajapinta on paljon sisäistä rajapintaa laajempi. Se on suunniteltu hyödynnettäväksi asiakkuudenhallintasovelluksen käyttöliittymän toteuttamisessa. Täten se tarjoaa perinteiset create, read, update, delete (CRUD), eli luonti, lukeminen, päivittäminen ja poistaminen kaikille asiakkuudenhallintasovelluksen resursseille.

Koska asiakkuudenhallinnan rajapinta on tarkoitettu pohjaksi käyttöliittymän rakentamiselle, on rajapintaa käyttävän käyttäjän tunnistautuminen erilainen kuin sovellusalustan sisäisen rajapinnan tapauksessa. Tätä rajapintaa varten on luotava erillinen käyttäjätunnus tietue, jonka avulla hallitaan rajapinnan oikeuksia. Sisäisessä rajapinnassa tunnistautuminen ja valtuutus tapahtuu automaattisesti oikeaan kontekstiin joko asiakastunnisteen tai käyttäjäistunnon avulla. Asiakkuudenhallinnan rajapinnassa on mahdollista hallita myös asiakastilejä erillisinä resursseina, jolloin käsiteltävä konteksti ei ole selvillä pelkän tunnistautumistiedon perusteella. Myös tämän rajapinnan suunnittelussa on pyritty täyttämään kappaleessa 3.2.4 esitellyt REST arkkitehtuurin rajoitteet. Rajapinnan kuvaus on esitelty resursseittain eriteltynä seuraavissa taulukoissa. Kaikki taulukot on koostettu käyttäen samaa rakennetta, jossa otsikkona on käsiteltävä resurssi, ja jokaisella rivillä on järjestyksessä käytettävä metodi, URL -polku ja kuvaus. Kontekstin hallitsemisen helpottamiseksi rajapinta on suunniteltu hiearkiseksi. Tämän vuoksi rajapinta kuvastaa sen pohjalla olevaa tietomallia.

Taulukko 4.2 *Ulkoisen rajapinnan asiakasresurssin kuvaus*

Asiakas
GET /customers Lukee asiakaskokoelman asiakkaiden yksilöivien tunnisteiden listan
POST /customers Luo uuden asiakkaan asiakaskokoelmaan
GET /customers/:customerId Lukee customerId muuttujassa määritellyn tunnisteen mukaisen asiakkaan tiedot
PATCH /customers/:customerId

Päivittää customerId muuttujassa määritellyn tunnisteen mukaisen asiakkaan tiedot
DELETE /customers/:customerId Poistaa käytöstä customerId muttujassa määritellyn tunnisteen mukaisen asiakkaan

Kuten kappaleesta 4.2.1 selviää, on koko asiakkuudenhallintasovelluksen tietomalli rakennettu asiakasentiteettien varaan. Asiakastietojen hallintaa varten rajapinta tarjoaa hyvin yksinkertaisen joukon resurssin metodeja, joiden avulla on mahdollista rakentaa käyttöliittymä. Tarjottavat metodit sallivat resurssien listaamisen, uusien resurssien luomisen, yksittäisen resurssin tarkastelun, yksittäisen resurssin päivittämisen ja yksittäisen resurssin poistamisen.

Taulukko 4.3 *Ulkoisen rajapinnan sopimusresurssin kuvaus*

Sopimus
GET /customers/:customerId/contracts Lukee customerId muuttujassa määritellyn tunnisteen mukaisen asiakkaan sopimuskokoelman tunnisteiden listan
POST /customers/:customerId/contracts Luo uuden sopimuksen customerId muuttujassa määritellyn tunnisteen mukaiselle asiakkaalle
GET /customers/:customerId/contracts/:contractId Lukee customerId+contractId muuttujayhdistelmällä määritellyn asiakkaan sopimuksen tiedot
PATCH /customers/:customerId/contracts/:contractId Päivittää customerId+contractId muuttujayhdistelmällä määritellyn asiakkaan sopimuksen tiedot
DELETE /customers/:customerId/contracts/:contractId Poistaa customerId+contractId muuttujayhdistelmällä määritellyn asiakkaan sopimuksen käytöstä

Sovellusalustaa ei ole mahdollista käyttää pelkän asiakastilin avulla. Jokaisella asiakkaalla täytyy olla alustan käyttämistä varten aktiivisena oleva sopimus. Sopimusten hallintakäyttöliittymää varten rajapinta tarjoaa metodit asiakkaan sopimusten listausta, uuden sopimuksen luomista, olemassa olevan sopimuksen tietojen tarkastelua, sopimuksen tietojen päivittämistä ja sopimuksen käytöstä poistamista varten. Kaikissa pyynnöissä on mukana tieto asiakkaan kontekstista customerId muuttujassa.

Taulukko 4.4 *Ulkoisen rajapinnan istuntoresurssin kuvaus*

Istunto
GET /customers/:customerId/contracts/:contractId/sessions Lukee customerId+contractId muuttujayhdistelmällä määritellyn asiakkaan sopimuksen istuntojen tunnisteiden listan
POST /customers/:customerId/contracts/:contractId/sessions

Luo uuden istunnon customerId+contractId muuttujayhdistelmällä määritellyn sopimuksen alaisuuteen
DELETE /customers/:customerId/contracts/:contractId/sessions/:sessionId Poistaa käytöstä customerId+contractId+sessionId muuttujayhdistelmällä määritellyn istunnon

Istuntoresurssien hallinnan rajapinta on asiakas- ja sopimusresursseja yksinkertaisempi. Rajapinta tarjoaa metodit asiakkaan sopimuksen alaisten istuntojen listaamiseen, uuden istunnon luomiseen ja olemassa olevan istunnon poistamiseen. Asiakkuuden ja sopimuksen konteksti kulkee pyyntöjen mukana osana pyyntöpolkua.

Taulukko 4.5 *Ulkoisen rajapinnan mallitiedostoresurssin kuvaus*

Mallitiedosto
GET /customers/:customerId/contracts/:contractId/models Lukee customerId+contractId muuttujayhdistelmällä määritellyn asiakkaan sopimuksen mallitiedostojen tunnisteiden listan
GET /customers/:customerId/contracts/:contractId/models/:modelId Lukee customerId+contractId+modelId muuttujayhdistelmällä määritellyn mallitiedoston tiedot
DELETE /customers/:customerId/contracts/:contractId/models/:modelId Poistaa käytöstä customerId+contractId+modelId muuttujayhdistelmällä määritellyn mallitiedoston

Mallitiedostot poikkeavat asiakkuudenhallinnan muista resursseista, sillä niitä ei ole mahdollista luoda suoraan asiakkuudenhallintaan. Tämä on tietoinen päätös, jotta asiakkuudenhallintasovellukselle kommunikointi olisi sovellusalustan järjestelmän kannalta yksisuuntaista. Käytännössä kaikki asiakkuudenhallintasovelluksen malliresurssit lisätään palvelun tarjoaman sisäisen rajapinnan kautta. Täten malliresursseille tarjotaan metodit vain asiakkaan sopimuksen alaisten malliresurssien listaamista, yksittäisen malliresurssin tietojen tarkastelua ja malliresurssin käytöstä poistamista varten.

Taulukko 4.6 *Ulkoisen rajapinnan laskuresurssin kuvaus*

Lasku
GET /customers/:customerId/invoices Lukee customerId muuttujalla määritellyn asiakkaan laskujen tunnisteiden listan
POST /customers/:customerId/invoices Luo customerId muuttujalle määritellylle asiakkaalle uuden laskun
GET /customers/:customerId/invoices/:invoiceId Lukee customerId+invoiceId muuttujayhdistelmällä määritellyn laskun tiedot
PATCH /customers/:customerId/invoices/:invoiceId Päivittää customerId+invoiceId muuttujayhdistelmällä määritellyn laskun tiedot
DELETE /customers/:customerId/invoices/:invoiceId

Poistaa käytöstä customerId+invoiceId muuttujayhdistelmällä määritellyn laskun

Laskutusresurssien hallinnan rajapinta mahdollistaa asiakaskohtaisten laskujen käyttöliittymän toteuttamisen. Muista asiakkuudenhallintasovelluksen resursseista poiketen laskut linkitetään suoraan asiakasresurssiin sopimuksen sijaan. Rajapinta tarjoaa metodit asiakkaan laskujen listaamiseen, uuden laskun luomiseen, laskun tietojen hakemiseen ja laskun päivittämiseen ja laskun poistamiseen.

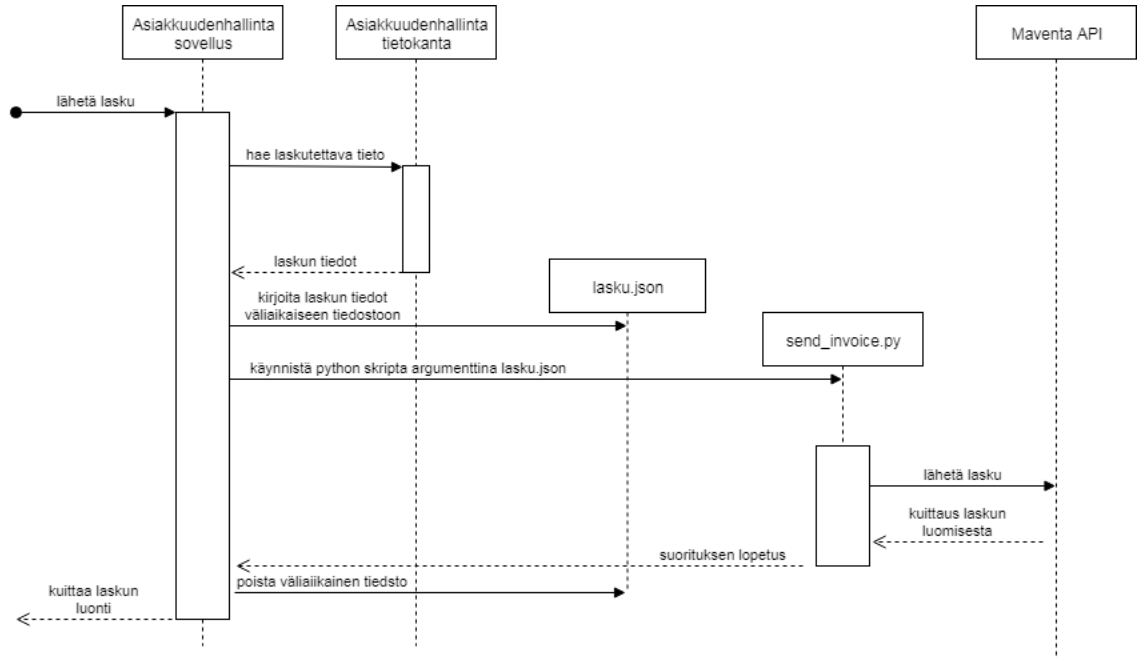
4.2.4 Laskutusintegraatio

Työn tilaajan vaatimuksesta asiakkuudenhallintasovellusta ei haluttu lähteä integroimaan kolmannen osapuolen myyntireskontraohjelmistoon. Ainoaksi vaihtoehdoksi jäi siis käyttää kolmannen osapuolen laskutuspalvelua. Laskutuspalvelun valinta tehtiin kahden vaatimuksen perusteella. Ensimmäinen vaatimus oli, että palvelun täytyy pystyä välittämään verkkolaskuja. Sovellusalustaa käyttävät asiakkaat ovat pääosin yrityksiä joille verkkolaskuttaminen on ainoa realistinen tapa. Toinen vaatimus oli, että palvelun täytyy tarjota rajapinta laskujen lähettämiseen, jotta asiakkuudenhallintasovellus on yhdistettävissä siihen ohjelmallisesti.

Suomessa toimii 29 verkkolaskujen välittäjää [30]. Näistä tarjolla olevista laskutusintegraatiokumppaneista valittiin Maventa [18]. Maventa oli ainoa palveluntarjoaja, joka tarjosi rajapinnan laskujen luomista ja lähettämistä varten [19].

Maventan tarjoama integroitumisrajapinta on toteutettu Simple Object Access Protocol:lla (SOAP). SOAP on protokolla, jonka avulla on mahdollista toteuttaa verkkopalveluita. Näin toteutettujen verkkopalveluiden tarjoamat ominaisuudet kuvataan Web Services Description Language (WSDL) avulla [15]. WSDL ansiosta SOAP palveluiden käyttö on mahdollista valmiiden ohjelmakirjastojen avulla. Laskutusintegraatiossa päädyttiin käyttämään vain yhtä rajapinnan tarjoamista palveluista (invoice_create), joten periaatteessa toteutus olisi ollut mahdollinen jopa ilman erillistä SOAP kirjastoa.

SOAP on protokollana suhteellisen monimutkainen kaikkien sen tarjoamien ominaisuuksien takia. Laskutuspalveluntarjoajan rajapinta ei toiminut Node.js ympäristössä saatavilla olleilla SOAP kirjastoilla, joten integraatio päädyttiin tekemään käyttäen python ohjelmointikieltä ja sille tarjolla olevaa suds -nimistä SOAP kirjastoa. Asiakkaille lähetettävien laskujen luonnin sekvenssi on esitetty kuvassa 4.16.



Kuva 4.16 *Sekvenssikaavio laskun lähettämisestä*

Laskun luominen alkaa asiakkuudenhallintasovellukselle tehdystä pyynnöstä. Tämä pyyntö sisältää tiedon asiakkaasta, jolle halutaan luoda uusi lasku. Lasku luodaan lukemalla asiakkuudenhallintatietokannasta tiedot kyseisen asiakkaan sillä hetkellä hyödyntämistä sovellusalusta resursseista. Näistä tiedoista koostetaan JSON muotoinen tietue, joka tallennetaan väliaikaiseen tiedostoon. Tämä tiedosto sisältää kaiken tarvittavan tiedon laskun luomista varten. Tiedoston rakenne mukailee Maventa API:lle lähetettävän tiedon muotoa. Luotuaan väliaikaisen tiedoston asiakkaanhallintasovellus käynnistää erillisen python prosessin, jossa suoritetaan python koodi laskun lähettämiseksi Maventa API:n kautta. Saadessaan paluuviestin Maventa API:lta, python koodi poistuu suorituksesta. Jos laskun luominen Maventa API:n kautta onnistui, python koodi palauttaa koodin 0. Virheen tapahtuessa palautetaan virhekoodi, jotta asiakkuudenhallintasovellus osaa reagoida virheelliseen laskun luontiin.

5. TYÖN ARVIOINTI

Työssä toteutettu asiakkuudenhallintasovellus oli varsin monipuolinen projekti. Asiakkuudenhallintasovellukselta haluttu toiminnallisuus oli varsin tarkasti tiedossa ennen projektin aloittamista. Lisäksi sovelluksen toimintaympäristö toi projektiin omat rajoitteensa, joista osa voidaan lukea hyödyiksi enemmän kuin haitoiksi.

Asiakkuudenhallintasovelluksen vaatimukset voidaan jakaa kahteen luokkaan. Liiketoimintavaatimukset vaikuttavat suoraan asiakkuudenhallintasovellusta käyttävät yrityksen liiketoimintaan. Näiden vaatimusten voidaan ajatella olevan asiakkuudenhallintasovelluksen pakollisia ominaisuuksia. Tekniset vaatimukset liittyvät tässä projektissa asiakkuudenhallintasovelluksen suoritusympäristöön ja tilaajan asettamiin teknologiavalintoja koskeviin ratkaisuihin.

Liiketoimintavaatimukset listataan tämän työn kappaleessa 3.1.2. Työn tuloksena toteutettu sovellus toteuttaa suurimman osan näistä vaatimuksista. Sovellus mahdollistaa asiakastietojen perustason hallitsemisen. Tähän kuuluu tietojen tallentaminen, muokkaaminen ja lukeminen. Tarjoamalla rajapinnan sovellusalustan rajapintatoteutukselle asiakkuudenhallintasovelluksen avulla pystytään mahdollistamaan asiakkaiden ohjelmallinen tunnistaminen. Sovellusalustan asiakkaiden analysointi on mahdollista asiakkuudenhallinnan malliresurssien käyttöä seuraavan ominaisuuden ansiosta. Sovellusalustan käyttöperustainen laskuttaminen on mahdollista, koska asiakkuudenhallintasovellus pitää kirjaa sovellusalustan asiakkaiden käytössä olevista malliresursseista. Näitä tietoja hyödynnetään laskujen lähettämisessä kolmannen osapuolen palvelun kautta, johon asiakkuudenhallintasovellus tarjoaa integraation. Työn aikana toteutettujen rajapintojen ja varsin yksinkertaisen tietomallin ansiosta uusien resurssien lisääminen sovellusalustan käyttöön on mahdollista. Täten sovellusalustan kehittyessä sen uudet ominaisuudet on helppo kytkeä asiakkuudenhallintasovellukseen.

Liiketoimintavaatimuksista ainoastaan valtuutuksien hallinta jäi toteutukseltaan keskeneräiseksi. Asiakkuudenhallintasovellus mahdollistaa nyt sovellusalustan asiakkaiden istuntokohtaisen kontrollin malliresursseihin. Laajempaa valtuutuksien hallintaa ei järjestelmästä vielä löydy.

Työn tekniset vaatimukset saatiin kaikki täytettyä. Sovellusta on mahdollista ajaa käyttäen docker -säiliötä. Sovelluksen toteutus tehtiin käyttäen Node.js ohjelmointikieltä, joka helpotti vaadittujen REST -rajapintojen toteuttamista. Tiedon tallentaminen erilliseen relaatiotietokantaan helpottaa tiedon oikeellisuuden ylläpitämistä, koska tietokantataulujen määritteisiin on mahdollista lisätä rajoitteita vierasavainten oikeellisuudesta. Täten asiakkuudenhallintasovelluksessa ei periaatteessa voi olla koskaan tallennettuna tietoa, jota ei ole mahdollista kytkeä johonkin asiakkaaseen. Käyttämällä teknologioita docker, Node.js

ja REST mahdollistetaan myös asiakkuudenhallintasovelluksen skaalautuvuus tulevaisuudessa. Koska itse sovelluslogiikka on käytännössä tilaton, on mahdollista käynnistää sovelluksesta uusia instansseja, jos sovelluksen vasteajat kasvavat liian suuriksi. Työn tilaajan vaatimus sovelluksen toteuttamisesta TDD periaatteiden mukaisesti helpotti työn toteutusta huomattavasti. Varsinkin projektin alussa tietomallin iterointi oli varsin tiheää, joten TDD:n ansiosta muutokset pysyivät kurissa. Luonnollisesti myös testejä joutui kirjoittamaan uusiksi uusien ominaisuuksien myötä.

Työssä vältyttiin suuremmilta ongelmilta teknisen toteutuksen suhteen. Kehitettäessä asiakkuudenhallintasovellusta sovellusympäristöön, jossa ei aiempaa toteutusta vielä ole olemassa, antaa se mahdollisuuksia myös asiakkuudenhallintaprosessien kehittämiselle.

Työn tuloksena toteutettu asiakkuudenhallintasovellus saatiin tarpeeksi hyvään kuntoon, jotta se pystyttiin ottamaan käyttöön alkuperäisen aikataulun mukaisesti. Projektilla ei niinkään ollut kiinteää aikataulua, mutta tiettyjä ominaisuuksia haluttiin päästä ottamaan käyttöön jo ennen koko projektin valmistumista.

Asiakkuudenhallintasovelluksen jatkokehityksessä on paljon mahdollisuuksia. Ensimmäinen kehityskohde tulee olemaan asiakkuudenhallintakäyttöliittymän rakentaminen sovelluksen tarjoaman REST-rajapinnan päälle. Tämän jälkeen sovellusta tullaan päivittämään uusilla ominaisuuksilla sovellusalustan laajenemisen mukaan. Kaikki uudet ominaisuudet kytketään asiakkaiden tekemiin sopimuksiin, ja niiden käytöstä tullaan keräämään tietoa analytiikkaa varten. Käyttötietojen kerääminen mahdollistaa monipuolisten analyysien ja statistiikkojen luomisen. Myös nämä ovat osa asiakkuudenhallintasovelluksen jatkokehityskohteita.

6. YHTEENVETO

Työn pohjalla olevan projektin tavoitteena oli toteuttaa asiakkuudenhallintajärjestelmä, joka integroitaisiin osaksi työn tilaajan, Vektorio Oy:n, tuotetta. Vektorio Oy:n tuote on internetselaimessa käytettävä palveluna tarjottava sovellusalusta, joka mahdollistaa 3D -tietomallien visualisoimisen.

Asiakkuudenhallintajärjestelmän vaatimuksilla oli kolme suurta vaikutetta. Ensinnäkin järjestelmää täytyi pystyä käyttämään asiakkaiden tietojen hallitsemiseen. Täten järjestelmään tallennetulle tiedolle piti olla mahdollista suorittaa tavanomaisia operaatioita kuten luominen, lukeminen, päivittäminen ja poistaminen. Toinen periaatteellinen vaatimus oli, että järjestelmän täytyi tarjota mahdollisuus ohjelmalliseen integraatioon. Ohjelmallisen integraation ansiosta asiakkuudenhallinta olisi mahdollista kytkeä muihin sovellusalustan osiin. Kolmas vaatimus oli, että asiakkuudenhallintajärjestelmän piti tarjota mahdollisuus laskujen lähettämiseen asiakkaille palvelun ominaisuuksien käytön perusteella.

Projekti aloitettiin lopputalvesta 2018. Projektin alusta asti tiedostettiin asiakkuudenhallintajärjestelmän potentiaalinen monimutkaisuus ja laajuus. Tämän vuoksi projektissa pyrittiin luomaan ensimmäinen käytettävissä oleva versio, joka olisi mahdollista integroida osaksi tilaajan muuta järjestelmää. Tulevaisuudessa tehtävien laajennoksien vuoksi järjestelmä pyrittiin suunnittelemaan mahdollisimman helposti laajennettavaksi.

Projektia pohjustettiin tutustumalla asiakkuudenhallintaa käsitteleviin artikkeleihin ja kirjallisuuteen. Tämän materiaalin avulla pystyttiin ajattelemaan asiakkuudenhallintaa muutenkin kuin vain teknologisena ratkaisuna jonka työn tilaaja halusi. Opittujen tietojen perusteella asiakkuudenhallintajärjestelmä pyrittiin suunnittelemaan niin, että tilaajayrityksen olisi mahdollista kehittää asiakkuudenhallintaprosessiaan ilman teknologian luomia haittoja. Yrityksen liiketoiminnalle olisi hyvin haitallista, jos asiakkuudenhallintajärjestelmä ei mahdollistaisi eri tyylisten asiakassuhteiden hyödyntämistä. Sovellusalustan tapauksessa käyttötapauksia on varmasti yhtä paljon kuin asiakkaitakin. Täten monet asiakkaat haluavat varmasti kokeilla sovellusalustan ominaisuuksia rajatussa ympäristössä ennen päätöksen tekemistä. Tämän estäminen teknologisilla ratkaisuilla vaikeuttaisi yrityksen asiakkaiden hankintaa huomattavasti.

Työn tilaajalla oli projektissa käytettävien teknologioiden suhteen vaatimuksia, jotka asettivat toteutukselle muutamia reunaehtoja. Asiakkuudenhallintajärjestelmän logiikkakerroksen toteutus piti tehdä käyttäen Node.js puitetta. Lisäksi logiikkakerroksen pitäisi olla ajettavissa docker -säiliössä, jotta sen käyttöönotto ja integrointi osaksi järjestelmää olisi mahdollisimman helppoa. Käytettävän tietokantaratkaisun suhteen työn tilaajalla ei ollut muita vaatimuksia kuin että se toteutettaisiin käyttäen relaatiotietokantaa. Tämä sen

vuoksi, että tallennettavan tiedon eheys olisi pakotettavissa vierasavaimia käyttäen. Lisäksi yrityksestä löytyy relaatiotietokantaosaamista, joten järjestelmän ylläpito ja laajentaminen helpottuvat.

Näiden vaatimusten perusteella suunniteltiin asiakkuudenhallintajärjestelmän arkkitehtuuri, joka tarjosi kaksi erilaista REST -rajapintaa. Ensimmäinen rajapinta tarjoaa operaatiot asiakastiedon luomiseen, lukemiseen, muokkaamiseen ja poistamiseen. Tämän rajapinnan käyttäminen on mahdollista vain erillisten käyttäjätunnusten avulla. Toinen rajapinta mahdollistaa kytkeytymisen asiakkuudenhallintajärjestelmään ohjelmallisesti. Tämän rajapinnan operaatioiden hyödyntäminen vaatii erillisen asiakkuuteen sidotun merkkijono käyttämistä tunnistautumiseen.

Toteutettujen REST -rajapintojen ja erillisen tietokannan ansiosta asiakkuudenhallintajärjestelmän logiikkakerroksesta saatiin täysin tilaton. Tämä mahdollistaa asiakkuudenhallintajärjestelmän logiikkakerroksen skaalautumisen käynnistämällä uusia instansseja docker säiliöistä.

Asiakkuudenhallintajärjestelmän tiedontallennus toteutettiin erillisen docker -säiliönä ajettavan tietokantainstanssin avulla. Tietokannan rakenne suunniteltiin niin, että kaikki asiakkaisiin liittyvät tiedot muodostavat relaatiot, joita seuraamalla minkä vain tietokantaan tallennetun tiedon voi liittää tiettyyn asiakkaaseen.

Työn projektin aikana saatiin luotua tilaajan vaatimusten mukainen asiakkuudenhallintajärjestelmä. Valmiiden rajapintatoteutusten ansiosta yrityksen on mahdollista luoda tiedon esittämistä varten oma käyttöliittymä, kun se tulee tarpeelliseksi. Valitettavasti projektin aikataulun puitteissa ei ollut mahdollista toteuttaa tilaajayrityksen asiakkuudenhallintaprosessia tehtyä teknologista ratkaisua pidemmälle.

Työn tavoitteena oli tutkia miten palveluna tarjottavan sovellusalustan asiakkuudenhallintajärjestelmä kannattaa toteuttaa. Lisäksi oli määritelty kysymyksiä, joiden vastaukset tarjoavat rajoitteita ja suunnitteluratkaisuja asiakkuudenhallintajärjestelmän toteuttamiseksi.

Järjestelmään tallennettavan tiedon eheyden varmistamiseksi kannattaa käyttää relaatiotietokantaa. Täten on mahdollista kytkeä kaikki tallennettavat tietueet toisiinsa vierasavainten avulla. Tämä vierasavainten ketju linkittää minkä vain järjestelmästä löytyvän tiedon tiettyyn asiakastiliin. Vierasavaimista saadaan ratkaisu myös tietomallin laajennettavuuteen. Uusien resurssien hallinnointi on mahdollista, kun ne on kytketty johonkin jo olemassa olevaan tietueeseen. Lisäksi uusille tiedoille on mahdollista käyttää samanlaista valtuutusta ja lokitusta, kuin muillekin tiedoille.

Asiakkuudenhallintajärjestelmältä vaaditaan kaksi erilaista rajapintaa. Ensimmäinen rajapinta tarjotaan käyttöliittymän tarpeisiin. Toinen rajapinta toimii täysin ohjelmallisesti,

ja sillä mahdollistetaan sovellusalustan muiden komponenttien kommunikointi asiakkuudenhallintaan. Käytännössä järjestelmässä ei pitäisi olla kuin yksi komponentti joka on riippuvainen asiakkuudenhallintasovelluksen tarjoamista palveluista. Tämä kytkentä kannattaa tehdä mahdollisimman aikaisessa vaiheessa sovellusalustan asiakkaiden pyyntöjä käsiteltäessä. Täten kaikki muut alustan komponentit voivat luottaa niille saapuviin viesteihin, eikä niiden tarvitse tehdä asiakkaiden tunnistamista tai valtuuttamista.

Esitettyyn tutkimuskysymykseen saatiin vastaus tämän työn kuvaaman asiakkuudenhallintajärjestelmän arkkitehtuurin muodossa. Tutkimusta voidaan pitää onnistuneena.

LÄHTEET

[1] S. Aleem, R. Batool, F. Ahmed, A. Khatak & R.M.U. Ullah, Architecture guidelines for SaaS development process, ICCBDC 2017 Proceedings of the 2017 International Conference on Cloud and Big Data Computing, 2017, pp. 94-95, Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=3141136

[2] T. Bhat & N. Nagappan, Evaluating the Efficacy of Test Driven Development: Industrial Case Studies, ISESE '06 Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, 2006, pp. 356-357., Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=1159787

[3] C. Boettiger, An introduction to Docker for reproducible research, ACM SIGOPS Operating Systems Review - Special Issue on Repeatability and Sharing of Experimental Artifacts, Volume 49 Issue 1, January 2015, 74 p., Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=2723882

[4] E.F. Codd, A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, vol. 13, no. 12, June 1970, Saatavissa: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[5] Docker. Docker reference. Saatavissa: https://docs.docker.com/engine/reference/builder/

[6] Docker. What is container. Saatavissa: https://www.docker.com/what-container

[7] Docker. What is docker. Saatavissa: https://www.docker.com/what-docker

[8] Docker Hub. Docker image repository. Saatavissa: https://hub.docker.com/

[9] Express. Express Node.js module. Saatavissa: https://expressjs.com/

[10] R. Fielding, Architectural Styles and Design of Network-based Software Architectures, 2000, pp. 76-85, Saatavissa: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[11] GitHub. Docker library. Saatavissa: https://github.com/docker-library/hello-world/blob/b0a34596994b120f5456f08992ef9a75ed56f34e/amd64/hello-world/Dockerfile

[12] C. Lee & H. Lee, A Password Stretching Method using User Specific Salts, Proceedings of the 16th international conference on World Wide Web, 2007, 1215 p., Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=1242772

[13] C. Li & L. Ma, Operation Design of Customer Relationship Management System, IEEE 8th International Symposium on Computational Intelligence and Design, 2015, pp. 536-537, Saatavissa: https://ieeexplore-ieee-org.libproxy.tut.fi/stamp/stamp.jsp?tp=&arnumber=7469191

[14] L. Liang, L. Zhu, W. Shang, D. Feng & Z. Xiao, Express supervision system based on NodeJS and MongoDB, 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), 2017, Saatavissa: http://ieeexplore.ieee.org.libproxy.tut.fi/document/7960064/

[15] A.C.C. Machado & C.A.G. Ferraz, Guidelines for Performance Evaluation of Web Services, WebMedia '05 Proceedings of the 11th Brazilian Symposium on Multimedia and the web, 2005, Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=1114234

[16] MariaDB, 2018, Saatavissa: https://mariadb.org/

[17] MariaDB, Timestamp, 2018, Saatavissa: https://mariadb.com/kb/en/library/timestamp/

[18] Maventa, 2018, Saatavissa: https://maventa.com/

[19] Maventa, Verkkolaskutus, 2018, Saatavissa: https://maventa.com/verkkolaskutus/api/api-v1-1-documentation/

[20] Node.js, Overview of blocking vs Non-blocking, 2018, Saatavissa: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

[21] Node.js, The Node.js Event Loop, Timers and process.nextTick(), 2018 Saatavissa: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

[22] Node.js, About Node.js, 2018, Saatavissa: https://nodejs.org/en/about/

[23] npm, Node Package Manager, 2018, Saatavissa: https://www.npmjs.com

[24] H. Park & S. Redford, Client Certificate and IP Address Based Multifactor Authentication for J2EE Web Applications, CASCON '07 Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, 2007, pp. 167-168, Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=1321229

[25] A. Payne, Handbook of CRM: Achieving Excellence Through Customer Management, Routledge, 2005, pp. 1-88, 226-283, Saatavissa: https://ebookcentral.proquest.com/lib/tut/reader.action?docID=255230

[26] A.N. Purbowo, Yulia & A.I. Suryadi, Web Based Customer Relationship Management Application for Helping Sales Analysis on Bike Manufacturer, 2017 International Conference on Soft Computing, Intelligent System and Information Technology, 2017, pp. 347-349, Saatavissa: https://ieeexplore-ieee-org.libproxy.tut.fi/stamp/stamp.jsp?tp=&arnumber=8262594&tag=1

[27] Salesforce, Sales Cloud, 2018, Saatavissa: https://www.salesforce.com/eu/products/sales-cloud/pricing/

[28] M.Q. Shatnawi, M.B. Yassein & H. Al-natour, Customer Relationship Management At Jordan University of Science and Technology:Case Study, Issues and Recommendations, ICET2017, 2017, Saatavissa: https://ieeexplore-ieee-org.libproxy.tut.fi/stamp/stamp.jsp?tp=&arnumber=8308149

[29] M. Soni, Cloud computing basics – platform a service (PaaS), Linux Journal Volume 2014 Issue 238, February 2014 Article No. 4, 2014, Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=2592741

[30] Tieke, Verkkolaskusanasto, 2018, Saatavissa: https://www.tieke.fi/display/verkkolasku/Verkkolaskusanasto

[31] Vektorio Oy, 2018, Saatavissa: https://vektor.io/

[32] Y. Zhang, R. Krishnan & R. Sandhu, Secure Information and Resource Sharing in Cloud Infrastructure as a Service, Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security, 2014, 81 p., Saatavissa: https://dl-acm-org.libproxy.tut.fi/citation.cfm?id=2663884

[33] Zoho, Zoho Features, 2018, Saatavissa: https://www.zoho.eu/crm/features.html