Moniasiakkuuden toteutus SaaS-järjestelmissä

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Kesäkuu 2026
Jere Hietikko

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -järjestelmällä.

TURUN YLIOPISTO
Tietotekniikan laitos

Jere Hietikko: Moniasiakkuuden toteutus SaaS-järjestelmissä

LuK-tutkielma, 22 s.
Tietojenkäsittelytiede
Kesäkuu 2026

Moniasiakkuus on keskeinen osa SaaS-järjestelmien toteutusta. Moniasiakkuudella tarkoitetaan useiden asiakkaiden palvelemista yhden sovellusinstanssin kautta. SaaS-järjestelmissä moniasiakkuus mahdollistaa resurssitehokkaita ja skaalautuvia ohjelmistoratkaisuja, mikä on tehnyt kyseisistä järjestelmistä suosittuja.
Tutkielma on luonteeltaan kirjallisuuskatsaus. Tutkielmassa tarkastellaan moniasiakkuuden toteutusta SaaS-järjestelmissä ja tutkitaan moniasiakkuuden eri toteutustapoja ja niiden sovelluksia. SaaS-järjestelmässä tutkitaan keskeisiä huomioitavia asioita ja SaaS-järjestelmän kautta tutkitaan myös niihin liittyviä palvelutasosopimuksia.
Tulokset osoittavat, että moniasiakkuudessa toteutustapa vaihtelee skaalautuvuuden ja tietojen eristettävyyden perusteella. Moniasiakkuudessa keskeistä on tiedon eriyttäminen eri asiakkaiden välillä, joka voi tapahtua tietokanta- ja sovellustasolla. Tietokantatasolla tiedon tallentaminen moniasiakkuutta hyödyntäen voi tapahtua tallentamalla tiedot samaan tietokantaan ja skeemaan, tai eri skeemoihin samassa tietokannassa. Sovellustasolla täytyy yhdistää asiakas omiin tietoihinsa esimerkiksi asiakkaan tunnisteen avulla.
SaaS-järjestelmissä toteutustavassa on myös huomioitava skaalautuvuus ja turvallisuus. Järjestelmien suunnittelussa keskeisessä osassa ovat asiakkaan tarpeet ja toteutusta rajoittaa käytettävissä olevat resurssit. SaaS-järjestelmässä kehittäjän täytyy priorisoida haluttuja ominaisuuksia kehityskustannusten kanssa. Laajennettavat arkkitehtuurit ja konfiguroitavuus ovat usein haluttu ominaisuus, mutta lisäävät kehitys ja ylläpitokustannuksia.
Palvelutasosopimuksissa määritellään ohjelmistossa asiakkaalle tarjottavat ominaisuudet ja rajoitukset. Sopimukset kannattaa rakentaa ominaisuuksien päälle, jota useat asiakkaat hyödyntävät. Asiakaskohtaiset erikoisehdot lisäävät ohjelmiston kustannuksia. Jos sopimuksessa asetetaan vaatimuksia esimerkiksi palvelun saatavuudesta, niin ohjelmistoon täytyy toteuttaa saatavuutta mittaava lisäosa, joka varmistaa sopimuksen täyttämisen.

Asiasanat: SaaS, Moniasiakkuus, pilvipalvelut, palvelutasosopimus

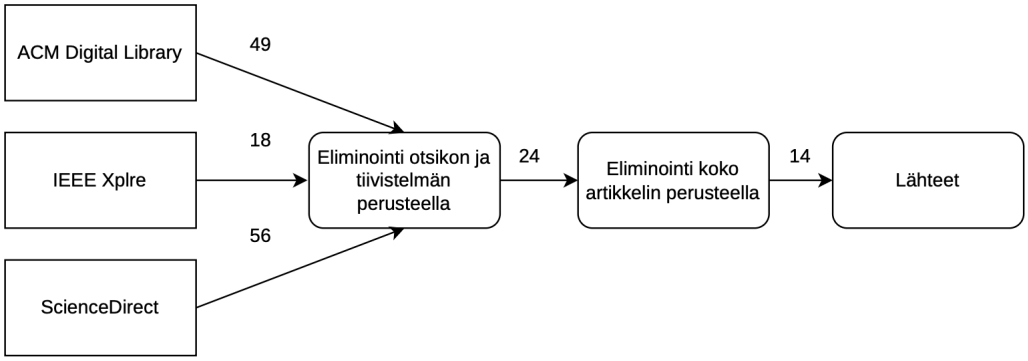
Sisällys

1 Johdanto

Pilviohjelmistot (engl. Software-as-a-Service, SaaS) ovat suosittu tapa toimittaa ohjelmistoja verkon välityksellä. SaaS-järjestelmät ovat pilvi-infrastruktuurin varassa toimivia sovelluksia, joihin kuluttajalla on käyttöoikeus [1]. Usealle asiakkaalle saman ohjelmiston toimittaminen luo omanlaisen toimintaympäristönsä. Moniasiakkuus-käsitteellä tarkoitetaan yleensä kykyä palvella useita asiakasorganisaatioita yhden ohjelmistotuotteen instanssin kautta. [2] Sosiaalisten medioiden, sekä muiden keskeisten internetin kautta käytettävien ohjelmistojen yhteisenä piirteenä on, että useat käyttäjät operoivat samassa ympäristössä, samanlaisesta käyttöliittymästä. Kyseisten tuotteiden suosio on tehnyt moniasiakasympäristöistä keskeisen teknologian yhteiskuntamme päivittäisessä toiminnassa. Luonnollisesti tämä ruokkii kysyntää myös uusille ratkaisuille, mutta tässä tutkielmassa keskitymme kyseisten teknologisten ratkaisujen toteuttamiseen, sekä pohdimme miten eri ratkaisuja tulisi soveltaa tilanteesta riippuen.

Tutkielman lähteiden etsiminen toteutettiin systemaattisesti käyttäen tiettyä hakulausetta ennalta määritellyissä yleisesti hyväksi todetuissa kustantajien tietokannoissa (ACM, IEEE, ScienceDirect). Tämän jälkeen hakutulokset suodatettiin otsikon, sekä tiivistelmän perusteella. Jäljelle jääneistä artikkeleista valittiin sopivimmat koko artikkelin perusteella. Lähteisiin viittaamisessa on viitattu alkuperäisiin artikkeleihin. Tiedonhaun vaiheet on esitetty kuvassa 1.1.

Hakulauseena käytettiin "multi-tenant architecture" AND "SaaS".



Kuva 1.1: Kirjallisuuskatsauksen tiedonhaun vaiheet

Tutkielman tarkoituksena on kartoittaa miten moniasiakasympäristöt on mahdollista toteuttaa SaaS-palveluissa. Näiden aiheiden perusteella muodostettiin ensimmäiset kaksi tutkimuskysymystä. SaaS-järjestelmissä palvelutasosopimuksia käytetään sopimaan asiakkaalle tarjottavat ominaisuudet, joten siitä muodostui kolmas tutkimuskysymys.

TK1: Miten moniasiakasympäristö toteutetaan?

TK2: Miten SaaS-järjestelmä toteutetaan?

TK3: Mitä huomioida palvelutasosopimuksessa?

Tässä kandidaatintutkielmassa tutkitaan, miten moniasiakasympäristöjä toteutetaan SaaS-järjestelmissä. Toisessa luvussa esitellään lukijalle mitä erilaisia moniasiakkuustoteutuksia on tietokantojen tallennustavoista resurssitehokkaisiin palvelimiin. Lisäksi käsitellään SaaS-järjestelmiä, sekä niihin liittyviä palvelutasosopimuksia. Kolmannessa luvussa tutkitaan käytännönläheisiä ratkaisuja moniasiakkuuden toteuttamiseen ohjelmistosta tietokantoihin. Tutkitaan myös SaaS-järjestelmien toteutustapoja, sekä palvelutasosopimusten noudattamiseksi toteutettavia ohjelmistoratkaisuja. Viimeisessä luvussa esitetään yhteenveto ja johtopäätökset.

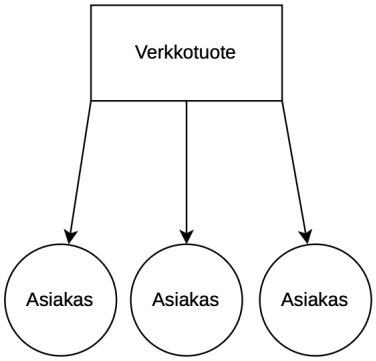
2 Tausta

2.1 Moniasiakkuus

Tässä luvussa tarkastellaan moniasiakasympäristöjä. Ohjelmistoarkkitehtuureista puhuttaessa moniasiakkuus-käsitteellä tarkoitetaan yleensä sitä, että yhden ohjelmistotuotteen instanssin kautta voidaan palvella useita asiakasorganisaatioita. [2] Moniasiakkuudessa keskeistä on resurssitehokkuuden saavuttaminen palvelinten yhteiskäytön mahdollistaman korkeamman käyttöasteen kautta. Kuvassa 2.1 visualisoitu verkkotuote, joka palvelee useita asiakkaita.

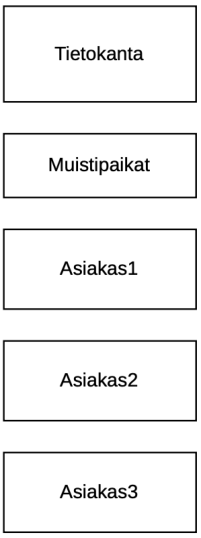
Jaap Kabbedijkin mukaan "Useimmat tutkijat ja käytännön toimijat ovat yhtä mieltä siitä, että moniasiakkuuden käyttö mahdollistaa ohjelmistotoimittajille useiden asiakkaiden palvelemisen yhdestä verkkotuotteesta, mutta tietyt toteutukset vaihtelevat merkittävästi"[2]. Kyseinen määrittely ei tarkenna, miten asiakkaan data tallennetaan, joten moniasiakasympäristön määritelmä jättää tallennustavalle tulkinnanvaraa. Ei voida päätellä käytetäänkö resurssitehokasta tallennustapaa, vai tallennetaanko data erillisiin tietokantoihin.

Vaikka moniasiakkuuden määritelmä ei ole täysin yksiselitteinen, niin järjestelmällä tavoitellaan jonkinlaista tehokkuutta. Guo Changin mukaan moniasiakkuus on yksi keskeisistä tavoista saavuttaa korkeampi voittomarginaali hyödyntämällä mittakaavaetuja [3]. Aiemmassa kappaleessa viitataan useiden asiakkaiden palvelemiseen yhdellä verkkotuotteella. Palvelun kehittäjän näkökulmasta tehokkuus saa-



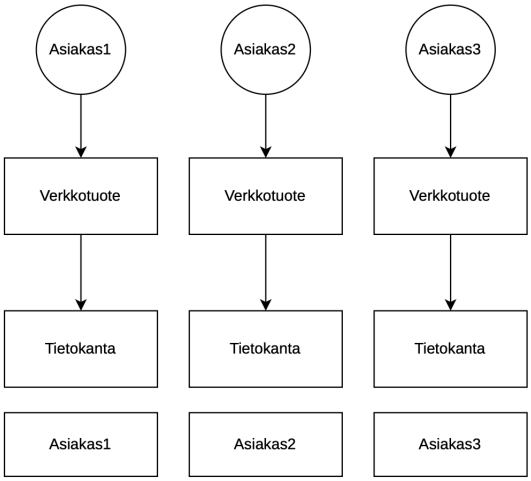
Kuva 2.1: Moniasiakasverkkotuote

vutetaan, kun jokaiselle yritykselle ei tarvitse toteuttaa erillistä tuotetta. Yritys voi myös hyödyntää resurssitehokasta tallennustapaa, jossa eri asiakkaiden tietoja tallennetaan samaan tietokantaan, jolloin myös fyysisten resurssien tarve vähenee. Kuvassa 2.2 on moniasiakastietokanta, johon on tallennettu kolmen asiakkaan tiedot. Yhden verkkotuotteen kautta asiakkaiden palveleminen siirtää resursseja yhteen



Kuva 2.2: Moniasiakastietokanta

paikkaan. Useiden verkkotuotteiden sijasta, kehittäjän täytyy ylläpitää vain yhtä, joten ylläpitotaakka vähenee. Kuitenkin useiden tuotteiden yhdistäminen ja useiden asiakkaiden palveleminen lisää tuotteen monimutkaisuutta. Ohjelmistojen ylläpitä-

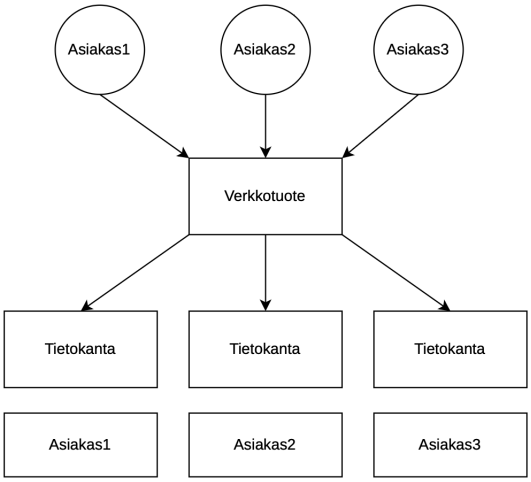


Kuva 2.3: Yksittäisasiakasperiaatteella toteutettu kokonaisuus

jät ovatkin huolissaan, että moniasiakkuuden käyttö lisäisi ylläpito-ongelmia järjestelmien konfiguroinnin seurauksena ja siten poistaisi moniasiakkuudella saavutetut hyödyt [4]. Sama pätee resurssitehokkaisiin tallennustapoihin. Moniasiakkuudessa täytyy siis arvioida tarkasti, milloin kyseinen toteutustapa on järkevä. Konfigurointi lisää ylläpidon taakkaa, mutta samalla laajentaa asiakaskuntaa.

Perinteisempi lähestymistapa etenkin yritysohjelmistoissa on ollut rakentaa ohjelmisto yrityksen tarpeisiin ja ostaa tai vuokrata laskentatehoa sekä tallennustilaa. Laskentatehoa on täytynyt varata enemmän käyttöpiikkejä varten eli tilannetta, jossa ohjelmistoa käytetäänkin yhtäaikaisesti huomattavasti enemmän. Esimerkiksi moni käyttäjä suorittaa samaan aikaan ohjelmistolla paljon laskentatehoa vaativia operaatioita. Tämä johtaa tilanteeseen, jossa osa palvelimista on käyttämättömänä suuren osan ajasta. Tietojärjestelmien ulkoistaminen on ratkaissut tämän ongelman. Kuvassa 2.3 on järjestelmä, jossa yritysten sovellukset toimivat erillään toisistaan.

Moniasiakastoteutus on mahdollista toteuttaa asiakkaille erillisillä tietokannoilla. Kuvassa 2.4 esitetään kyseinen toteutus. Kyseisessä toteutuksessa asiakkaiden data on jo suunnittelussa eriytetty ja sovellustasolla täytyy vain yhdistää asiakas



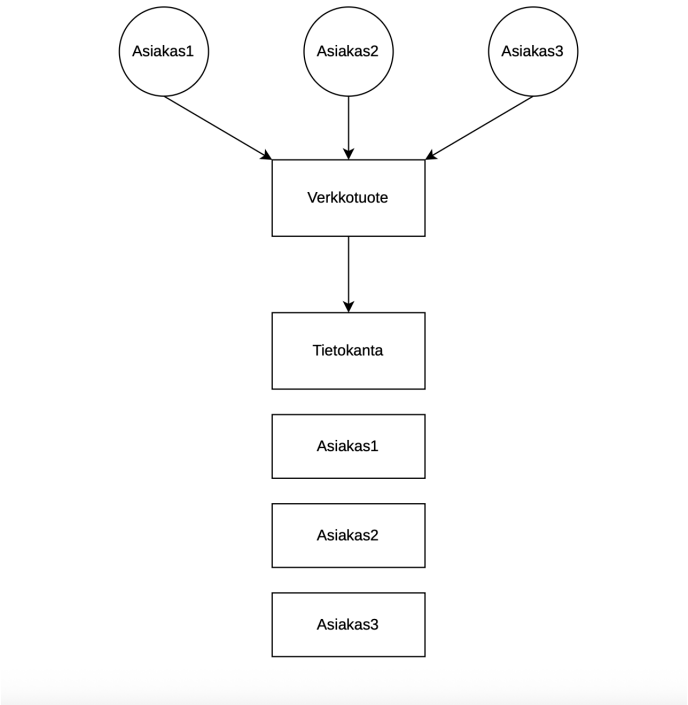
Kuva 2.4: Moniasiakas verkkotuote ja yksittäisasiakas tietokannat

tietokantaansa esimerkiksi asiakkaan tunnisteella. Yksittäisiä tietokantoja resurssitehokkaampi tapa on käyttää kokonaisvaltaista moniasiakastoteutusta. Kuvassa 2.5 näytetään kyseinen toteutus. Kaikkien asiakkaiden tiedot on tallennettu samaan tietokantaan ja tiedot täytyy suojata jollakin tavalla.

2.2 SaaS

SaaS-järjestelmät (engl. Software-as-a-Service) sanantarkka käännös on ohjelmisto palveluna. Yksi SaaS-ohjelmistojen keskeisistä piirteistä on resurssien jakaminen eri asiakkaiden välillä [2]. Yksi tapa saavuttaa resurssien jakaminen on moniasiakkuus, mutta kuten aiemmassa luvussa totesimme, niin moniasiakkuuden määritelmä ei rajaa tiukasti arkkitehtuurin toteutukselle, joten moniasiakkuutta hyödyntävä SaaS-järjestelmä voi käyttää datan tallennukseen single-tenant periaatetta eli tallentaa tiedot omiin tietokantoihin.

SaaS-järjestelmiä edelsi ASP-järjestelmät, kun 1990-luvulla kiinnostus tietojärjestelmien ulkoistamiseen lisääntyi ja tietojärjestelmien osittaisen tai täyden ulkois-



Kuva 2.5: Kokonaisvaltainen moniasiakastoteutus

tamisen todettiin mahdollistavan paremman palvelun tarjoaminen ja kilpailuedun säilyttäminen [5]. 1990-luvun lopulla sovelluspalveluntarjoajat (engl. Application service provider, ASP) alkoivat tarjota ohjelmistotuotteita verkkojen välityksellä. Kalliit palvelimet ja ohjelmistot voitiin ulkoistaa kokonaisuudessaan ASP-tuotteen tarjoajalle, mikä loi kysyntää erilaisille palvelin- ja verkkolaitteistoille. [6] ASP-järjestelmissä mahdollisimman monen yrityksen tietojärjestelmätarpeeseen pyritään luomaan ratkaisu, mutta järjestelmiä voidaan muovata yrityskohtaisesti ja palvelimet toimivat yksittäisasiakasympäristössä (engl. single-tenant architecture).

SaaS ja ASP-järjestelmissä on useita päällekkäisyyksiä ja tiettyissä järjestelmissä näitä voi olla vaikea erottaa toisistaan. Tekninen perusta SaaS ja ASP-järjestelmissä eroaa kuitenkin huomattavasti. ASP-järjestelmät on suunniteltu asiakaskohtaisesti palvelemaan asiakasta tietystä instanssista, kun SaaS-järjestelmät on suunniteltu palvelemaan useita asiakkaita samanaikaisesti samasta sovellusinstanssista [7].

Yhdysvaltain standardointivirasto (engl. National Institute of Standards and

Technology, NIST) määrittelee SaaS-järjestelmät pilvi-infrastruktuurin varassa toimiviksi sovelluksiksi, joihin kuluttajalla on käyttöoikeus. Sovelluksia käytetään ohuen asiakaskäyttöliittymän, kuten verkkoselaimen, tai ohjelmakäyttöliittymän kautta. Kuluttajalla ei ole vaikutusvaltaa ohjelmiston toteutukseen, lukuun ottamatta käyttäjäkohtaisia konfiguraatioasetuksia [1]. SaaS-järjestelmissä asiakas on siis täysin riippuvainen järjestelmän kehittäjästä. Toisaalta IT-järjestelmien ulkoistaminen mahdollistaa yritysten keskittymisen pääliiketoimintaan.

Moniasiakkuuden ylläpitoa käsiteltäessä huomattiin kehittäjien huoli ohjelmistojen konfiguroinnin vaikutuksesta ylläpidettävyyteen. SaaS-järjestelmissä ei ole tätä ongelmaa, koska järjestelmät ovat määritelmän mukaan konfiguroitavissa vain käyttäjäkohtaisilla asetuksilla. Tämän seurauksena ohjelmistoja on suoraviivaisempi ylläpitää, mutta samalla asiakaskunta ja ratkaistavan ongelman laajuus voivat pienentyä.

SaaS-järjestelmät määriteltiin pilvi-infrastruktuurin varassa toimiviksi sovelluksiksi. Määrittely ei rajannut arkkitehtuurin toteutusta moniasiakkuuteen. Microsoftin Frederick Chong ja Gianpaolo Carraro jakavat SaaS-kypsyysmallit neljään osaan: muokattavaan, konfiguroitavaan, konfiguroitavaan moniasiakas-tehokkaaseen ja skaalattavaan konfiguroitavaan moniasiakas-tehokkaaseen [8]. Kypsyysmallien valinta riippuu taloudellisesta logiikasta, sekä teknisen toteutuksen sopivuudesta.

Teknisesti muokattavia SaaS-järjestelmiä on haasteellista, jollei mahdotonta toteuttaa täysin moniasiakasarkkitehtuurilla. Kun taas skaalautuvien järjestelmien toteuttaminen muokattavana yksittäisasiakasjärjestelmänä on teknisesti mahdollista, mutta kustannukset skaalautuvat. Järjestelmiä ei ole siis kustannustehokasta toteuttaa suuremmalla skaalalla.

2.3 Palvelutasosopimukset

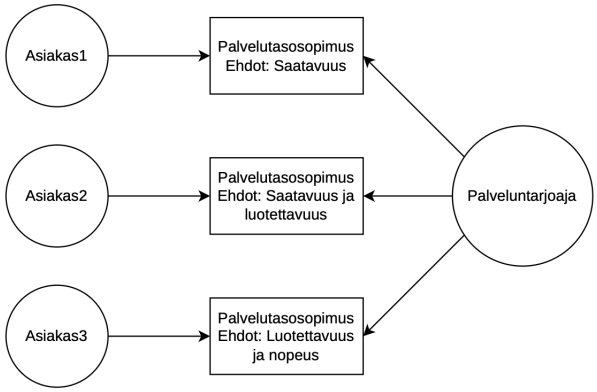
Ohjelmistoista on tullut keskeinen osa monen yrityksen liiketoimintaa. Jos yritys ei itse myy ohjelmistoja, niin ohjelmistot ovat vain yksi kulu ja riski yrityksen liiketoiminnassa. Yritysten tavoitteena on tuottaa voittoa, joten kulujen ja riskien minimointi on luonnollinen osa liiketoimintaa. SaaS-palveluiden keskeisiin ominaisuuksiin kuuluvat alhaiset aloituskustannukset ja kustannussäästöt ylläpidossa [9]. Palvelutasosopimuksella (engl. Service Level Agreement, SLA) solmitaan sovelluksen ehdoista asiakkaan kanssa ja rajoitetaan riskejä.

Perinteiset IT-järjestelmät ovat vaatineet suuria sijoituksia järjestelmän kehittämistä varten, mutta SaaS-järjestelmissä kehittämisen rahoittaminen siirtyy kehittävän yrityksen harteille. Jotta järjestelmät olisivat houkuttelevia etenkin pienille ja keskisuurille yrityksille, järjestelmän tarjoajilla on vahva kysyntä moniasiakkuuden kehityksen mahdollistaville ohjelmistoille ja niihin liittyville palvelinratkaisuille [3]. Kyseisille järjestelmille on siis taloudellisesti looginen perusta, jossa järjestelmä teoriassa hyödyttää kaikkia. Rahoituksen siirtyminen kehittäjälle muodostaa uusia riskejä, jotka tulee huomioida sopimuksissa.

Ostavan ja kehittävän yrityksen täytyy pystyä arvioimaan järjestelmien kustannuksia kululaskelmassaan. SaaS-järjestelmiin liittyy suoraviivaisia kustannuksia, kuten palvelimet ja tietokannat, mutta kehitys ja ylläpitokustannukset ovat haastavampia määritellä ennalta. Järjestelmiin liittyy myös mahdollisia sivukustannuksia, kuten palvelun vikaantumisen takia menetetty asiakas tai tietojen häviäminen. On siis selvää, että vaikka järjestelmä näyttää taloudellisesti järkevältä täytyy kokonaiskuva täytyy arvioida tarkasti. SaaS-palvelussa pelkästään tuote ei vaikuta ostopäätökseen vaan tuotetta tarjoavan yrityksen täytyy olla luotettava, jotta tuotteen voidaan uskoa olevan olemassa myös tulevaisuudessa.

Luotettavuuden takaamiseksi voidaan laatia palvelutasosopimus. SaaS-palveluiden asiakkailla on erilaisia palveluvaatimuksia liittyen toiminnallisuuteen,

vasteaikaan, saatavuuteen, palvelujaksoon ja muihin samankaltaisiin kriteereihin. Palveluntarjoajien ja asiakkaiden väliseen koordinointiin tarvitaan palvelutasosopimus, jonka pohjalta resurssit kohdistetaan [10]. Kuvassa 2.6 näkyy, miten asiakkaiden palvelutasosopimuksien ehdot voivat erota toisistaan.



Kuva 2.6: Palvelutasosopimukset

SaaS-palveluissa eri asiakkailla on eri vaatimukset suorituskyvylle, joten jos kaikkia asiakkaita palvellaan saman palvelutasosopimuksen perusteella muodostuu tilanne, jossa toiset asiakkaat kuluttavat samalla sopimuksella huomattavasti enemmän. Palveluissa voi olla haitallisia toimijoita, joiden käyttäytyminen vaikuttaa haitallisesti muihin asiakkaisiin [3]. Kyseisissä tilanteissa kustannukset jakautuvat eriarvoisesti asiakkaiden välillä. SaaS-palveluihin olisi siis hyödyllistä sisällyttää mittareita hallinnoimaan suorituskyvyn jakamista eri asiakkaiden välillä. Mittarit mahdollistavat myös yksilöllisesti laadittujen palvelutasosopimusten noudattamisen.

Sopimuksiin luonnollisesti liittyy oletus, että sopimus täytetään, mutta sopimuksen rikkoutumisesta pitää määritellä erikseen seuraukset. Moniasiakkuuden perusteella toteutetuissa SaaS-järjestelmissä asiakas voi toimia palvelutasosopimuksen vastaisesti esimerkiksi käyttämällä liikaa resursseja, mikä voi johtaa koko järjestelmän epävakauteen ja sitä kautta estää järjestelmää täyttämästä muiden asiakkaiden palvelutasosopimuksen määrittelemiä ehtoja [10].

SaaS-yritysten perustamiseen liittyy useita vaiheita ja perustajille on tärkeää

saada toimiva tuote toimitettua asiakkaalle mahdollisimman nopeasti. Luotettavuus, saatavuus ja muut palvelutasosopimuksiin liittyvät ei toiminnalliset vaatimukset voivat vaikuttaa toissijaisilta prioriteeteiltä. Uusia SaaS-yrityksiä täytyykin tilaajan näkökulmasta arvioida tarkoin. SaaS-yrityksissä saman koodikannan jakaminen voikin johtaa riskien nopeaan leviämiseen [11].

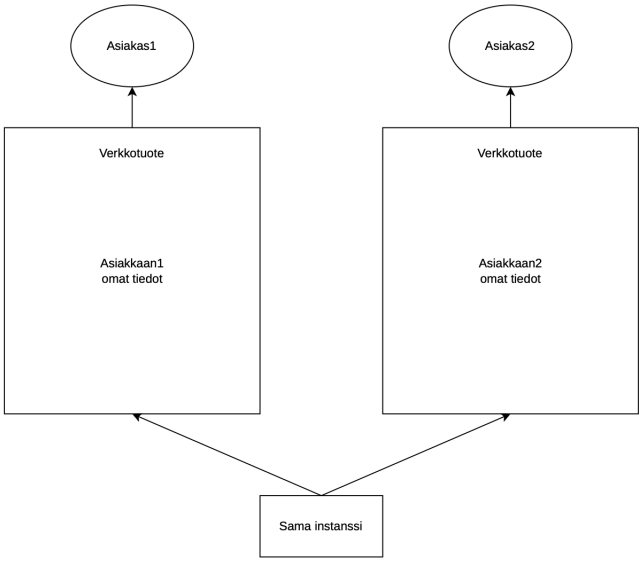
Tekoälyn yleistyessä myös "vibe-koodaus" on yleistynyt. Vibe-koodaus on uusi heikosti määritelty termi ohjelmistokehityksessä, jolla viitataan tekoälytyökalujen ohjeistamiseen luomaan koodia manuaalisen kirjoittamisen sijaan [12]. Erityisesti SaaS-ohjelmistoihin on houkuttelevaa yrittää vibe-koodata ohjelmisto, joko ostajayrityksen näkökulmasta korvaamaan kallis ulkoinen SaaS-ohjelmisto, tai yksityisenä henkilönä kehittää SaaS-ratkaisu sivutuloksi tai korvaamaan oma työpaikka. Vibe-koodaamisessa ongelmaksi muodostuu, jos teknistä taustaa omaamaton henkilö yrittää luoda tuotantovalmiin ohjelmiston. Tekoälyn avulla luotu koodi jätetään usein tietoturvatarkastusten ulkopuolelle, mikä johtaa näkymättömiin haavoittuvuuksiin [12]. Etenkin SaaS-palvelutasosopimusten näkökulmasta tämä luo erittäin huolestuttavia riskejä.

3 Moniasiakkuuden ja SaaS-järjestelmien arkkitehtuuriratkaisut

3.1 Moniasiakkuuden toteutusmallit

On useita asiakkaita, jotka haluavat hyödyntää samaa palvelua. Asiakkaat toimivat itsenäisesti eivätkä jaa dataa muiden asiakkaiden kanssa. Asiakkailla on erilaisia toiminnallisia ja ei-toiminnallisia vaatimuksia. Vaatimukset esitetään palvelutasosopimuksissa (SLA). [13] Toiminnalliset vaatimukset voivat olla esimerkiksi mitä sovelluksella voi tehdä, kuten tallentaa dokumentteja pdf-muodossa. Ei-toiminnalliset vaatimukset voivat olla esimerkiksi tietoturvavaatimusten täyttäminen ja palvelun luotettavuus. Tavoitteena on toteuttaa sovellusarkkitehtuuri, jossa sovellus on kehitetty siten, että yhden instanssin kautta asiakkaat pystyvät hyödyntämään tuotetta yksilöidyllä datallaan. Kuvassa 3.1 havainnollistettu kyseistä toiminnallisuutta.

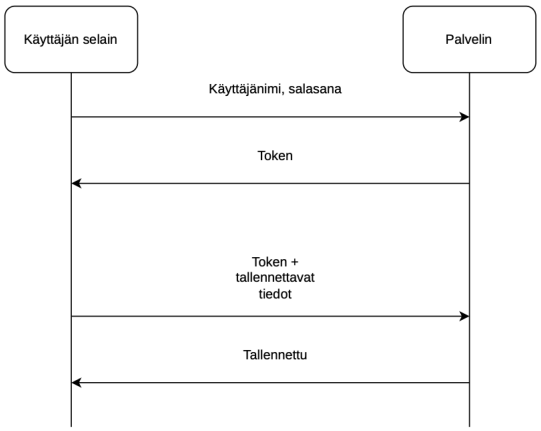
Ensimmäinen ongelma kuvaa 3.1 katsoessa on, miten palvelu tunnistaa asiakkaan. Palvelua, joka on otettu käyttöön yhden konfiguroitavan instanssin tilassa, on kutsuttava asiakkaan kontekstissa, joka sisältää tiedon mikä asiakas kutsui sitä. Asiakkaan konteksti on mekanismi, jolla lähetetään tietoja asiakkaasta palvelun kutsuviestissä. Se voi olla yksinkertaisesti asiakkaan tunniste tai monimutkaiset toden-



Kuva 3.1: Yksittäisinstanssi

nustiedot. Se voi olla esimerkiksi SOAP-otsikko, tai sisältyä viestin hyötykuormaan toimittajatunnuksella. [14] Käytännössä yksi toteutustapa on, että asiakas kirjautuu verkkotuotteeseen oman selaimen käyttöliittymästä lähettämällä salasanan ja käyttäjätunnuksen palvelimelle. Palvelin tarkistaa ovatko tiedot oikein ja onnistuneessa kirjautumisessa palauttaa allekirjoitetun todenteen asiakkaan selaimelle. Kun asiakas tekee uusia kyselyitä palvelimelle, niin kyselyn mukana välitetään todenne, josta palvelin tunnistaa käyttäjän, sekä käyttöoikeudet. Todenne voi sisältää myös toimittajatunnuksen. Kuvassa 3.2 kirjautuminen ja tietojen lähettäminen visualisoitu.

Moniasiakasympäristössä datan tallentamiseen on kolme lähestymistapaa: asiakkailla erilliset tietokannat, asiakkailla jaetut tietokannat, mutta erilliset skeemat, asiakkailla jaettu tietokanta ja jaettu skeema. Jaetun tietokannan kehityskustannukset ovat korkeammat, mutta ylläpitokustannukset matalammat. Erillisissä tietokannoissa ylläpitokustannukset ovat korkeammat, mutta tietojen eristämisen taso on korkein. [15] Tämä tarjoaa hyvän kehyksen tietokannan toteutuksen valitsemiseen. Pienet ja kriittisen datan ohjelmistot on järkevää toteuttaa erillisiä tietokan-



Kuva 3.2: Token-kirjautuminen

toja hyödyntämällä ja skaalautuvat geneeristä tietoa sisältävät sopivat jaettuihin tietokantoihin.

Lähdetään ensin tutkimaan erillisten tietokantojen toteutusta. Toimimme edelleen moniasiakasympäristössä, joten oletamme, että asiakas on kirjautunut verkkotuotteeseen ja haluaa tallentaa dataa. Erillisten tietokantojen toteutuksessa luomme jokaiselle asiakkaalle erikseen tietokannan. Metadata yhdistää jokaisen tietokannan oikeaan asiakkaaseen, ja tietokannan suojaus estää asiakasta käyttämästä vahingossa tai ilkivaltaisesti muiden asiakkaiden tietoja [15]. Eli tarvitsemme metadatalle palveluun erillisen tietokannan, jossa on tiedot asiakkaasta ja asiakkaalle kuuluvasta tietokannasta.

Jaetun tietokannan toteutuksessa on useita asiakkaita, joiden tiedot tallennetaan samaan tietokantaan. Kyseisessä tavassa keskeistä on miten varmistetaan tietojen suojaus, sekä tiedon välittäminen vain tietoon oikeutetulle asiakkaalle. Moniasiakas tietokantojen toteuttamiseen useimmat pilvipalvelut käyttävät kyselymuunnosta (query transformation) yhdistämään useita yhden vuokralaisen loogisia skeemoja sovelluksessa yhteen monivuokralaisen fyysiseen skeemaan tietokannassa [16]. Oikeus nähdä tietoa tarkistetaan siis sovelluskerroksessa, joten haettava tieto täytyy

pystyä yhdistämään asiakkaaseen esimerkiksi asiakkaan tunnisteella. Tietokantatoteutuksessa, jossa asiakkailla on erilliset skeemat, mutta tietokanta on jaettu, täytyy yhdistää asiakas omaan skeemaansa esimerkiksi asiakkaan tunnisteella.

Jaetun tietokannan skeeman toteutus riippuu käyttökohteesta. Yksinkertaisin tapa on lisätä asiakkaan tunniste tietokantatauluun ja jakaa taulut asiakkaiden kesken. Ongelmana kyseisessä toteutuksessa on tietokannan laajennettavuus. SaaS-ohjelmistoissa asiakkailla voi olla tarve erilaisiin tietokantatauluihin ja tällaisissa tapauksissa jaetut tietokantataulut eivät toimi. Ratkaisuna on lisätä asiakkaille yksityisiä tietokantatauluja yhteisten lisäksi [16]. Jaetut tietokantataulut kuitenkin riittävät jos SaaS-ohjelmiston asiakkaiden ei tarvitse tallentaa eri tyyppistä dataa. Jaetut tietokantataulut helpottavat ohjelmiston ylläpitoa, koska taulujen määrä pysyy vakiona asiakkaiden kasvaessa, joka helpottaa sovelluksen skaalaamista.

3.2 SaaS-järjestelmien arkkitehtuurit ja skaalautuvuus

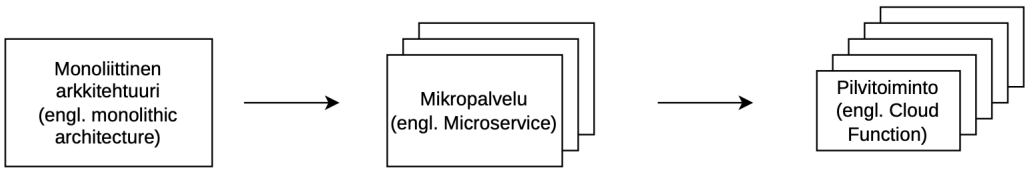
SaaS (engl. Software as a Service) on tapa toimittaa ohjelmistoja etänä internetin kautta [17]. Verkkotuotteesta asiakkaan palveleminen vaatii internetiin yhdistetyn palvelimen, joka mahdollistaa asiakkaalle tuotteen toimittamisen. SaaS-ohjelmiston toimittaminen asiakkaalle riippuu asiakkaan tarvitsemasta konfiguraation määrästä, sekä ohjelmiston toimintalogiikasta. SaaS-ohjelmisto voidaan toimittaa yhden tai useamman instanssin kautta. Yhden instanssin palveluksi kutsutaan palvelua, joka ei ole konfiguroitavissa ja useat asiakkaat käyttävät samaa palvelua. Usean instanssin palveluksi kutsutaan palvelua, joka ei ole konfiguroitavissa, mutta asiakkaat käyttävät erillisiä palvelun instansseja. Konfiguroitavan instanssin palvelussa on yksi palvelun instanssi, mutta palvelu on konfiguroitavissa. Usean konfiguroitavan instanssin palvelussa, palvelu on konfiguroitavissa ja asiakkaat käyttävät erillisiä

instansseja. Mielivaltaisten instanssien palvelumallissa, aiemmin mainittuja malleja käytetään sekaisin [14].

Kuten aiemmasta kappaleesta käy ilmi, niin SaaS-järjestelmistä on useita toteutustapoja käyttötarkoituksesta riippuen. Toteutusten parametrit määritellään palvelutasosopimuksilla (SLA). SLA-sopimukset edustavat toteutussuunnitelmaa pilvi-infrastruktuurin toiminnallisista ominaisuuksista. Pilvipalvelun asiakkaat arvioivat SLAn parametrien perusteella palvelun suorituskykyä. Infrastruktuuri suunnitellaan pilvipalvelun tarjoajien CSP (engl. Cloud Service Provider) tarjonnan perusteella. [18] SaaS-palvelun kehittäjän täytyy ensin tutustua pilvipalvelun tarjoajien tarjoamiin infrastruktuurivaihtoehtoihin ja tämän jälkeen tutkia mikä infrastruktuuri vastaa parhaiten pilvipalvelun asiakkaiden tarpeita.

Eri asiakkailla voi olla erilaisia tarpeita, jotka määritellään palvelutasosopimuksessa. Tästä muodostuu uusi haaste SaaS-palveluille, kun ohjelmisto ei olekaan aivan samanlainen kaikille käyttäjille. Yksi ratkaisu tähän on palvelukeskeiset arkkitehtuurit SOA (engl. Service Based Architecture). SOA-pohjaiset sovellukset ovat joukko palveluita, jotka omaavat löyhän kytkennän, joka mahdollistaa uusien palveluiden liittymisen helposti [19]. Tämäntyyppiset arkkitehtuuriratkaisut soveltuvat etenkin laajalle asiakaskunnalle suunniteltuihin SaaS-ratkaisuihin, joissa useilla asiakkailla on tarvetta mukauttaa sovellusta. Voi olla houkuttelevaa käyttää kyseisiä ratkaisuja, koska se lisää mahdollisten asiakkaiden määrää, mutta kyseiset ratkaisut lisäävät ohjelmiston monimutkaisuutta.

SaaS-ohjelmistojen taustaluvussa käsittelimme, miten SaaS-ohjelmistoissa keskeinen tavoite on yleensä saavuttaa resurssitehokkuus, joka mahdollistaa kannattavan liiketoiminnan. SaaS-ohjelmistoissa arkkitehtuurin suunnittelussa keskeisessä asemassa onkin arkkitehtuurin skaalautuvuus kestävästi. Asiakkaiden resurssien hallinta voidaan jakaa kolmeen palveluarkkitehtuuriin: monoliittiseen arkkitehtuuriin (engl. monolithic architecture), mikropalveluihin (engl. microservice), ja pilvitoi-



Kuva 3.3: Palveluarkkitehtuurit [20]

mintoihin (engl. cloud function) [20]. Kuvassa 3.3 näkyy arkkitehtuurit ja miten ne voivat rakentua toisen arkkitehtuurin sisään.

Arkkitehtuurin skaalautuvuuteen vaikuttaa siis paljon sovelluksen laajennettavuus. Kuvassa 3.3 Monoliittisen arkkitehtuurin sisällä ovat mikropalvelut ja mikropalveluiden sisällä pilvitoiminnot. Arkkitehtuureja voi siis yhdistellä ja se onkin järkevää sovelluksesta riippuen. Etenkin laajennettavuuden näkökulmasta mikropalvelut ja pilvitoiminnot laajenevat monoliittista arkkitehtuuria sujuvammin.

SaaS-ohjelmiston tilaajan näkökulmasta kannattaa kysyä ohjelmiston kehittäjiltä ohjelmiston arkkitehtuurista, jos ohjelmistoa tarvitsee laajentaa tulevaisuudessa. Tilaajan näkökulmasta ohjelmiston tarjoajien tekninen tausta ja yrityksen vakavaraisuus on myös hyvä tarkistaa. SaaS-yrityksen mennessä konkurssiin on epätodennäköistä saada jäljellä olevista lisensseistä korvausta. Etenkin vibe-koodattujen ohjelmistojen yleistyessä, ohjelmistoja voi kehittää pienellä pääomalla, sekä osaamisella. Usean asiakkaan ympäristössä tietoturvaloukkaus voi johtaa tietojen paljastumiseen kilpailijoille [4]. Sovelluksesta riippuen tilaajan riskit ovat laajat.

3.3 Palvelutasosopimusten tekninen toteuttaminen

Palvelutasosopimukset luvussa käsittelimme palvelutasosopimuksiin liittyviä velvoitteita, kuten luotettavuuden takaaminen ja miten eri asiakkaiden resurssienkulutus voi vaikuttaa muiden asiakkaiden palvelunkäyttöön ja palvelutasosopimuksiin.

Palvelutasosopimukset etenkin laskentatehoa vaativissa ohjelmistoissa, voivat määritellä kuinka paljon laskentatehoa asiakkaalla on saatavilla tietyllä ajanhetkellä. Etenkin tällaiset sopimukset lisäävät ohjelmistojen monimutkaisuutta, koska sopimusten varmistamiseksi täytyy kehittää erilaisia ohjelmistoratkaisuja. Yksi suosituimmista resurssien allokointikäytännöistä on ollut max-min-reiluus, jossa jokainen käyttäjä saa painoonsa suhteutetun osuuden resursseista [21]. George-Valentin Iordache kokeili kahden eri algoritmin suoritusaikaa. Ensimmäinen algoritmi oli saapumisjärjestyksessä palveleva FCFS-algoritmi (engl. First Come First Served) ja toinen algoritmi oli asiakkaiden tärkeyden huomioiva algoritmi. Tärkeyden huomioiva algoritmi vähensi tärkeiden asiakkaiden palveluaikaa 10ms, mutta kokonaissuoritusaika oli 470ms, kun saapumisjärjestyksessä palvelevan algoritmin kokonaissuoritusaika oli 430ms. [18] Resurssiallokaatiossa on siis hyvä huomioida, että allokaatio itsessään kuluttaa resursseja.

Walraven et al. [22] vertailivat Javalla toteutetun hotellivaraus SaaS-ohjelmiston toteutusta kahdella eri tavalla. Erillinen virtuaalikone asiakkaille ilman klusterointia, jolloin sovellus ei ole moniasiakkainen. Toisessa toteutuksessa virtuaalikoneet jaetaan jokaiselle asiakkaalle FIFO-kurmantasaajalla (engl. First In First Out). Tämän jälkeen simuloitiin sovelluksen toimintaa lähettämällä pyyntöjä eri toteutuksiin. Erillisten virtuaalikoneiden toteutuksessa muiden asiakkaiden korkea pyyntöjen määrä ei vaikuta toisiinsa. Jaettujen virtuaalikoneiden toteutuksessa yhden asiakkaan aggressiivinen käyttäytyminen vaikuttaa selvästi muiden asiakkaiden suorituskykyyn [22]. Erillisten asiakaskohtaisten virtuaalikoneiden hyödyntäminen siis takaa asiakkaille tietyn määrän resursseja ja toteutus ei ole monimutkainen. Tämä ei kui-

tenkaan ole resurssitehokas lähestymistapa, koska asiakkaille täytyy varata paljon resursseja virtuaalikoneeseen, joka on suuren osan ajasta käyttämättömänä. Yritysohjelmistoissa tämä on kuitenkin yksi vaihtoehto, jolloin palvelutasosopimus täytetään varmasti. Moniasiakkuuden näkökulmasta kuormantasaajan hyödyntämä algoritmi riippuu asikkaiden palvelutasosopimuksista. Aiemmassa kappaleessa verrattiin max-min-reiluutta FCFS-algoritmiin (engl. First Come First Served). FCFS ja FIFO toimivat samalla tavalla. Max-min-reiluus jakoi resurssit hyvin tärkeille asiakkaille, mutta oli kokonaisajassa hitaampi, kuin FCFS. FCFS ja FIFO algoritmit ovat yksinkertaisia toteuttaa, mutta voivat kohdella asiakkaita epätasaisesti. SaaS-ohjelmistoissa, jotka hyödyntävät moniasiakkuutta resurssien jakamiseen ja resurssit muodostavat merkittävän kulun olisi hyödyllistä seurata asiakkaiden resurssienkulutusta.

Asiakaskohtainen resurssien kulutuksen seuranta moniasiakasratkaisuissa on keskeistä palvelutasosopimusten noudattamiseksi. Liitettävä väliohjelmisto suorituskyvyn eristyksen valvomiseksi mahdollistaa ohjelmiston laajentamista tukemaan suorituskyvyn eriytystä ilman suuria muutoksia lähdekoodiin [22]. Asiakkaan kuluttamia resursseja voidaan laskea asiakkaan jalanjäljellä. Seurataan järjestelmää ja lasketaan asiakkaiden jalanjälki erikseen. Tämä voi vaatia järjestelmä- ja sovellustason parametrien mittaamista ja erittelemistä. Kun asiakas lähettää pyynnön, resurssien käyttö mittaa vastaamista pyyntöön, sekä vasteaikaa. Läpimenoaika merkitään asiakkaan identiteetillä ja näistä mittareista lasketaan asiakkaan jalanjälki. [13] Nyt käytettäessä FIFO-kuormantasaajaa resurssien kulutus nähdään asiakaskohtaisesti. Suorituskyvyn valvomisen mahdollistava ohjelmisto avaa myös uusia mahdollisuuksia, kuten resurssien tasapainottamista reaaliajassa. Myös max-min-reiluudessa kuluja voidaan seurata näin, mutta koska max-min-reiluudessa asiakas saa resursseja suhteessa painoonsa, niin myös kustannukset voidaan jakaa suhteessa asiakkaan painoon.

Resurssien jakaminen käytön mukaan mahdollistaa tarkempia hinnoittelumalleja, jotka ovat reilumpia asiakkaille. Walraven et al. [23] tutkivat suorituskyvyn eriyttämistä motivoivien esimerkkien kautta. Dokumenttien prosessointipalvelussa suorituskyvyn eriyttämiseen tarjottiin seuraavia kriteerejä: minimiläpivienti, määräajat tietyille tapahtumille, ja minimi vasteaika. Minimiläpiviennillä tarkoitetaan, että asiakkaiden pitää saada tietty määrä tehtäviä koko ajan toteutettua ja tehtävien määrä voi riippua asiakkaan tärkeydestä. Määräajoilla viitataan tehtäviin, jotka toistuvat tietyin määräajoin, kuten kuittien prosessointi ennen veroilmoitusta. Minimivasteajalla viitataan vähemmän laskentatehoa vaativien pienten tehtävien nopeaan prosessointiin. [23] Esimerkki tarjoaa konkreettisia parametreja, joita voidaan hyödyntää optimoinnin perusteena. Sovellettaessa muihin SaaS-järjestelmiin täytyy etsiä omasta järjestelmästä vastaavat parametrit ja soveltaa optimointia niihin.

SaaS-yrityksen tilaajan näkökulmasta voi olla haastavaa tietää, mitä teknisiä valintoja kehittäjät ovat tehneet ja vastaako tuote omaa tarvetta. Yrityksen kanssa onkin järkevää tehdä ainakin jonkinlainen palvelutasosopimus. Tilaajan näkökulmasta on tärkeää miettiä omia tarpeita ja kustannuksia. Esimerkiksi jos yrityksesi käyttää SaaS-ohjelmistoa tekemään varauksia, jotka eivät sisällä kriittistä tietoa, niin silloin kannattaa priorisoida saatavuutta. Jos yrityksesi käyttää ohjelmistoa salaisien liiketoimintojen toteuttamiseen, niin turvallisuus on keskeistä. Asiakkailla tulisi olla mekanismi, joka varmistaa palvelutasosopimuksen täyttymisen [13]. Mekanismit, jotka varmistavat palvelutasosopimuksen täyttymisen turvaavat asiakkaiden, sekä ohjelmiston tarjoajan oikeuksia. Palvelutasosopimusta neuvoteltaessa ohjelmiston tarjoajalla voikin olla valmiita kriteerejä, johon hinnoittelu perustuu ja joihin yritys on rakentanut ratkaisut.

4 Yhteenveto

Tutkielmassa tarkasteltiin SaaS-ohjelmistojen kehitystä moniasiakasympäristön avulla. Keskityttiin SaaS-ohjelmiston kokonaisvaltaiseen toteuttamisen, sekä ohjelmiston eri osiin liittyviin ongelmiin ja mahdollisiin ratkaisuihin. Ratkaisuja tutkittiin monesta eri näkökulmasta ja tavoitteena oli selittää ratkaisut käytännönläheisesti niiden vahvuuksien perusteella.

TK1: Miten moniasiakasympäristö toteutetaan?

Moniasiakasympäristö voidaan toteuttaa monilla eri tavoilla tietokanta- ja sovellustasolla. Moniasiakkuus toteutetaan tallentamalla asiakkaan yksilöivä tieto ja yhdistämällä asiakas tietokantaan, jossa asiakkaan tiedot sijaitsevat. Yksilöidyssä tietokantatoteutuksessa asiakas täytyy yhdistää asiakkaan omaan tietokantaan metadatan avulla. Jaetussa tietokantatoteutuksessa asiakas yhdistetään dataansa asiakkaan tunnisteen avulla. Sovellustasolla asiakkaan tunnistamisratkaisuna asiakkaan tiedot välitetään palvelimelle palvelun kutsuviestissä kirjautuessa, jolloin monet asiakkaat voivat hyödyntää samaa palvelua. Toteutustapaan vaikuttaa kriteerit, kuten skaalautuvuus ja tietojen eristettävyys.

TK2: Miten SaaS-järjestelmä toteutetaan?

SaaS-järjestelmät voidaan toteuttaa eri moniasiakas- ja arkkitehtuuriratkaisuilla. Keskeisessä asemassa ovat järjestelmien asiakkaiden tarpeet, sekä kustannusten osalta soveltuvat ratkaisut. SaaS-järjestelmiin soveltuu erilaisia arkkitehtuurivalin-

toja, kuten SOA (engl. Service Based Architecture). Eli löyhän kytkennän omaava joukko palveluita, joka on laajennettavissa. Myös monoliittista arkkitehtuuria, mikropalveluja ja pilvitoimintoja voidaan hyödyntää. Todettiin, että arkkitehtuureja voi käyttää rinnakkain ja eri arkkitehtuureissa on omat vahvuutensa, kuten pilvitoiminnoissa laajennettavuus ja monoliitissa järjestelmän kokonaisuus. Asiakkaan näkökulmasta SaaS-järjestelmän toteutuksessa ovat keskeisiä luotettavuus ja mahdollinen laajennettavuus.

TK3: Mitä huomioida palvelutasosopimuksessa?

Palvelutasosopimuksella varmistetaan, että SaaS-järjestelmä vastaa asiakkaan tarpeita. Sopimuksella voidaan asettaa myös rajoitteita asiakkaalle, kuten resurssienkäytön rajoitukset. Palvelutasosopimusten noudattamisen varmistamiseksi täytyy toteuttaa ohjelmistoon lisäosia, joiden avulla määritellään asiakkaan kuluttamien resurssien määrä. On erilaisia resurssienjakamisalgoritmeja, kuten min-max-reiluus ja FCFS-algoritmi (engl. First Come First Served). Resurssienjakoalgoritmit nopeuttivat tärkeiden asiakkaiden palvelemista, mutta hidasti palvelun kokonaisaikaa. Asiakkaiden resurssienkulutus vaikutti muiden asiakkaiden mahdollisuuteen kuluttaa resursseja, joka vaikeuttaa palvelutasosopimusten noudattamista vihamielisten toimijoiden tapauksessa. Asiakkaan näkökulmasta palvelutasosopimuksessa tulee huomioida tärkeät ominaisuudet ja turvata ne sopimuksella.

Jatkotutkimuksissa voisi keskittyä uusien moniasiakasratkaisujen kehittämiseen, etenkin monimutkaisemmissa SaaS-järjestelmissä. Myös tekoälyn roolia voisi tutkia moniasiakasjärjestelmien kehittämisessä.

Lähdeluettelo

[1] P. Mell, T. Grance et al., "The NIST definition of cloud computing", *NIST*, 2011.

[2] J. Kabbedijk, C.-P. Bezemer, S. Jansen ja A. Zaidman, "Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective", *Journal of Systems and Software*, vol. 100, s. 139–148, 2015.

[3] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang ja B. Gao, "A framework for native multi-tenancy application development and management", teoksessa *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, IEEE, 2007, s. 551–558.

[4] C.-P. Bezemer ja A. Zaidman, "Multi-tenant SaaS applications: maintenance dream or nightmare?", teoksessa *Proceedings of the joint ercim workshop on software evolution (evol) and international workshop on principles of software evolution (iwpse)*, 2010, s. 88–92.

[5] V. Grover, M. J. Cheon ja J. T. Teng, "A descriptive study on the outsourcing of information systems functions", *Information & Management*, vol. 27, nro 1, s. 33–44, 1994.

[6] M. A. Smith ja R. L. Kumar, "A theory of application service provider (ASP) use from a client perspective", *Information & management*, vol. 41, nro 8, s. 977–1002, 2004.

[7] H. Liao, ”Design of SaaS-based software architecture”, teoksessa *2009 International Conference on New Trends in Information and Service Science*, IEEE, 2009, s. 277–281.

[8] F. Chong ja G. Carraro, ”Architecture strategies for catching the long tail”, *MSDN Library, Microsoft Corporation*, vol. 910, 2006.

[9] R. Vidhyalakshmi ja V. Kumar, ”Design comparison of traditional application and SaaS”, teoksessa *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2014, s. 541–544.

[10] X. Cheng, Y. Shi ja Q. Li, ”A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA”, teoksessa *2009 Joint Conferences on Pervasive Computing (JCPC)*, IEEE, 2009, s. 599–604.

[11] W. Su, C. Lin, K. Meng ja Q. Liu, ”Modeling and analysis of availability for SaaS multi-tenant architecture”, teoksessa *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, IEEE, 2014, s. 365–369.

[12] S. Harkar. ”What is vibe coding?”, IBM, viitattu 11. toukokuuta 2026. url: `https://www.ibm.com/think/topics/vibe-coding`.

[13] S. Kalra ja T. V. Prabhakar, ”Patterns for managing tenants in a multi-tenant application”, teoksessa *Proceedings of the 22nd European Conference on Pattern Languages of Programs*, 2017, s. 1–10.

[14] R. Mietzner, T. Unger, R. Titze ja F. Leymann, ”Combining different multi-tenancy patterns in service-oriented applications”, teoksessa *2009 IEEE International Enterprise Distributed Object Computing Conference*, IEEE, 2009, s. 131–140.

[15] F. Chong, G. Carraro ja R. Wolter, ”Multi-tenant data architecture”, *MSDN Library, Microsoft Corporation*, s. 14–30, 2006.

[16] S. Aulbach, T. Grust, D. Jacobs, A. Kemper ja J. Rittinger, ”Multi-tenant databases for software as a service: schema-mapping techniques”, teoksessa *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, s. 1195–1206.

[17] Salesforce. ”What is SaaS”, Salesforce, viitattu 11. toukokuuta 2026. url: `https://www.salesforce.com/saas/`.

[18] G.-V. Iordache, ”An analysis of service level agreement parameters and scheduling in multi-tenant cloud systems”, teoksessa *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, IEEE, 2019, s. 140–145.

[19] J. Jing ja J. Zhang, ”Research on open SaaS software architecture based on SOA”, teoksessa *2010 International Symposium on Computational Intelligence and Design*, IEEE, vol. 2, 2010, s. 144–147.

[20] Z. Zhang, C. Yang, J. Wang, G. Hou ja Y. Yang, ”Scalable Resource Provisioning For Multi-tenant SaaS With Cloud Functions”, teoksessa *2022 International Conference on Computing, Communication, Perception and Quantum Technology (CCPQT)*, IEEE, 2022, s. 181–187.

[21] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker ja I. Stoica, ”Dominant resource fairness: Fair allocation of multiple resource types”, teoksessa *8th USENIX symposium on networked systems design and implementation (NSDI 11)*, 2011.

[22] S. Walraven, T. Monheim, E. Truyen ja W. Joosen, ”Towards performance isolation in multi-tenant saas applications”, teoksessa *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, 2012, s. 1–6.

[23] S. Walraven, W. De Borger, B. Vanbrabant, B. Lagaisse, D. Van Landuyt ja W. Joosen, "Adaptive performance isolation middleware for multi-tenant saas", teoksessa *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, IEEE, 2015, s. 112–121.