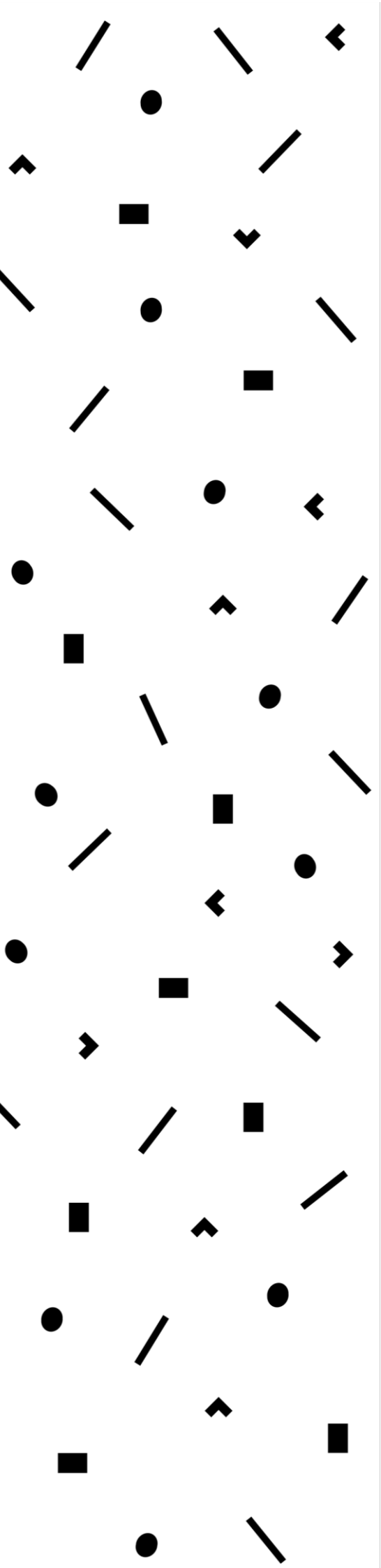


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Modulaarinen oikeuksienhallintasovellus

Case: CGI Security Manager

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2017
Sami Mustalahti

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

MUSTALAHTI, SAMI:

Modulaarinen oikeuksienhallintasovellus
Case: CGI Security Manager

Ohjelmistotekniikan opinnäytetyö, 46 sivua, 1 liitesivu

Kevät 2017

TIIVISTELMÄ

Opinnäytetyössä toteutettiin CGI:n oikeuksienhallintasovelluksesta uusi versio. Sovellusta käyttävät asiakasorganisaatiot jakaessaan työntekijöilleen oikeuksia nähdä yritykselle ja muille työntekijöille kuuluvia tietoja. Suurin osa tästä tapahtuu automatisoidusti, mutta välillä vastaan tulee tilanteita, joissa automatisointi ei riitä esimerkiksi normaalista poikkeavan oikeuksienjakologiikan vuoksi. Security Manager on manuaalinen ratkaisu tähän.

Projekti toteutettiin CGI Suomi Oy:n Digital Insight -osastolle, Lahden-toimipisteelle. CGI (Consultants to Government and Industry) on maailmanlaajuinen IT-palvelualan yritys. Se työllistää yli 68 000 asiantuntijaa ympäri maailmaa. Näistä noin 3 000 henkilöä työskentelee Suomessa.

Opinnäytetyössä käsitellään sovelluksen suunnittelua ja toimintalogiikan toteutusta. Suunnittelussa huomioitavaa oli asiakasorganisaatioiden kasvanut tietojenmäärä, käyttömukavuus sekä helppokäyttöisyys. Lisäksi sovelluksen tuli soveltua mahdollisimman pienin muutoksin Software as a Service- eli SaaS-malliseen markkinointiratkaisuun. Nämä lähtökohdat viitoittivat projektia suunnittelusta aina testaukseen asti.

Toteutuksessa käytettiin Microsoftin työkaluja ja kehitysalustana toimi virtuaalikone Azurella. Ohjelmaa ylläpidetään Internet Information Service -palvelimella, ja se toteutettiin MVC-arkkitehtuuria käyttäen. Tietokantana toimii Microsoft SQL Server.

Asiasanat: oikeuksienhallintasovellus, modulaarinen, Software as a Service, ASP.NET MVC, Internet Information Services, Dapper ORM, Windows-autentikaatio, Forms-autentikaatio, Single Sign On

Lahti University of Applied Sciences
Degree Programme in Information Technology

MUSTALAHTI, SAMI:

Modular Permission Management Software
Case: CGI Security Manager

Bachelor's Thesis in software engineering, 46 pages, 1 page of appendices

Spring 2017

ABSTRACT

The objective of this thesis was to implement a new version of CGI's permissions management software. The software is used by client organizations to distribute access rights to company data for employees. Most of this process is automated but from time to time a need arises to manually manage permissions. These situations may include complex logic or minor changes not worth the time it takes to adjust automation.

The thesis was made for CGI Finland Ltd's Lahti office. CGI (Consultants to Government and Industry) is a global information technology service company. It employs over 68 000 experts worldwide, of whom 3 000 work for CGI Finland.

The thesis deals with planning and implementing of business logic for permission management software. During planning, special attention was to be paid to the increasing amount of data in client database, as well as to convenience and easy operation. Additionally, the software was to work in a Software as a Service delivery method with minimal changes. These principles marked the process if design and implementation from beginning to end.

The implementation was done using Microsoft tools. All development took place in an Azure virtual machine. The finished product is hosted on the Internet Information Services platform and it implements MVC architecture. The database used in the project is Microsoft SQL Server.

Key words: permission management software, modular, Software as a Service, ASP.NET MVC, Internet Information Services, Dapper ORM, Windows-authentication, Forms-authentication, Single Sign On

SISÄLLYS

1 JOHDANTO

Modulaarinen rakenne on kasvava suuntaus sovelluskehityksessä. Modulaarisuus selkeyttää ohjelman rakennetta ja mahdollistaa usean henkilön samanaikaisen kehitystyön projektin eri osa-alueilla ilman, että se vaikuttaa muiden työhön. Suuretkin muutokset joihinkin osiin ovat mahdollisia muuttamatta muita yhtään. Software as a Service on suosittu ohjelmistojen jakamismalli, jossa ohjelmaa ylläpidetään pilvipalvelimella. Tällöin ohjelmaa ei tarvitse ylläpitää asiakkaan palvelimella ja asiakas maksaa ohjelman käytöstä kausiperusteisesti. SaaS-mallin etuja ovat joustavuus, kustannustehokkuus sekä yhtenäiset päivitykset kaikille käyttäjille.

CGI Group Inc (Consultants to Government and Industry) on kanadalainen maailmanlaajuiseksi kasvanut IT-palvelu- ja liiketoimintayritys. CGI:n palveluksessa on 68 000 asiantuntijaa sadoissa toimipisteissä ympäri maailmaa. Suomessa työntekijöitä on noin 3 000. CGI tarjoaa palveluita ja tuotteita laajalla skaalalla aina yrityskonsultoinnista prosessinohjausjärjestelmiin (CGI 2017). Tämä projekti toteutettiin CGI Suomi Oy:n Digital Insight -yksikölle sisäisenä kehitystyönä. Yksikkö myy ja kehittää tietovarastointi- ja raportointiratkaisuja sekä vastaa niiden ylläpidosta asiakkaiden palvelimilla.

Tässä työssä päivitettiin CGI:n tietovarastointiratkaisuissa käyttämä oikeuksienhallintasovellus. Ohjelman vastuulla on jakaa ja ylläpitää asiakasyrityksen henkilöstölle oikeuksia yhtiön dataan. Vanhan Security Managerin käyttöliittymä oli toteutettu Microsoft Silverlightilla, jonka tuki loppui Microsoftin julkistaman Internet Edgen mukana. Projektissa päädyttiin päivittämään samalla myös ohjelman toimintalogiikka vastaamaan kasvaneiden tietomäärien asettamia vaatimuksia ja mahdollisesti tulevaisuudessa julkaistavaa Service as a Software (SaaS) -mallia varten. Kehitysmäärittelyn mukaan tarkoituksena oli, että sama ohjelmakoodi, jota on-premises eli asiakkaan omalle palvelimelle asennettavassa versiossa käytetään, toimisi mahdollisimman vähillä

muutoksilla myös SaaS-mallissa. Ohjelman rakenteelta vaadittiin modulaarisuutta, jotta tarvittaessa voidaan päivittää tai uusia ohjelman osioita muuttamatta toisia.

Ohjelman keskeinen idea on toimia käyttöliittymänä, jossa asiakas-organisaatioiden pääkäyttäjät voivat tarkastella ja muuttaa henkilöiden oikeuksia nähdä organisaation, henkilöstön ja projektien tietoja muiden CGI:n ohjelmien kautta. Suurin osa oikeuksien jakamisesta on kuitenkin tarkoitus toteuttaa automaattisesti niin, että esimerkiksi esimiehellä on oikeus nähdä alaistensa tiedot. Oikeuksienhallintasovellus on suunniteltu käytettäväksi tilanteissa, joissa manuaalinen oikeuksien hallinta on järkevämpi tapa toimia. Näitä ovat esimerkiksi tilanteet, joissa oikeuksienjakologiikka on yleisestä käytännöstä poikkeava. Sovelluksen käyttö ei siis ole säännöllistä ja sillä tehtävät muutokset ovat monimuotoisia.

Tässä opinnäytetyössä keskitytään käsittelemään uuden Security Managerin palvelinpuolen (engl. *backend*) toimintalogiikkaa. Tähän sisältyy sekä on-premises että tuleva SaaS-mallinen versio sovelluksesta. Toimintalogiikka käsittää muun muassa tiedonkäsittelyyn käytettäviä algoritmeja, tietokantakutsuja, käyttöliittymään kulkevan kommunikaation sekä autentikaation ja autorisaation. Lisäksi käydään läpi projektissa käytettyjä ohjelmia ja tekniikoita ja kohdattuja haasteita, jotka keskittyivät erityisesti on-premises- ja SaaS-mallin yhdistämiseen samaan ohjelmakoodiin.

2 KÄYTETYT TEKNIIKAT

Security Manager on CGI:n kehittämä asiakasorganisaatioiden oikeuksienhallintatyökalu. Se kehitettiin alun perin Microsoft Silverlightilla. Yksinkertaisimmillaan pelkkä käyttöliittymän uudistaminen olisi ollut riittävä toimenpide, mutta vanhassa versiossa toimintalogiikka oli sekä sekoittunut ulkoasuun että vanhentunut tämänhetkisiin suunnitelmiin nähden. Siksi päädyttiin koko ohjelman uudistamiseen.

Huomioitavia seikkoja Security Managerin uudistamisessa olivat nopeus, käytättävyys sekä tulevaisuudessa mahdollisuus siirtyä käyttämään ohjelmaa Software as a Service -mallisesti. Koska asiakkaiden tietojen määrä voi olla hyvin suuri, täytyi kehityksessä erityisesti huomioida tietokantaan kulkevien kutsujen sujuvuus ja tiedon manipulointiin käytettävien algoritmien nopeus.

Käytettävissä tekniikoissa tuli olla pitkä tuki, sillä Security Manageria ei ole tarkoitus kehittää aktiivisesti. Tietokannan tauluihin oli sallittua tehdä muutoksia, mutta kehityksessä pyrittiin pitämään nämä muutokset minimissään. Näin helpotetaan vanhojen Security Manager -käyttäjien siirtoa uuden version piiriin. Ohjelman käyttöliittymän piti olla erillinen toimintalogiikasta eli ohjelman piti noudattaa malli-näkymä-ohjain-kehitysmallia, jotta tarvittaessa ohjelmaosioita voidaan sujuvasti vaihtaa.

Kehitysympäristönä toimii Azuren pilvipalvelimella sijaitseva virtuaalikone, ja kehitystyö suoritettiin Microsoft Visual Studio 2015 -ohjelmalla. Versiokontrollointiin käytettin Visual Studio Team Foundationia. Tietokanta, johon ohjelma ottaa yhteyden, on Microsoft SQL Server. Ohjelmaa on tarkoitus ylläpitää Microsoftin luomalla Internet Information Services (IIS) -verkkopalvelimella.

2.1 Software as a Service

Software as a Service (SaaS) tarkoittaa mitä tahansa pilvipalvelua, jossa käyttäjät voivat käyttää ohjelmistosovellusta Internetin yli. Esimerkiksi

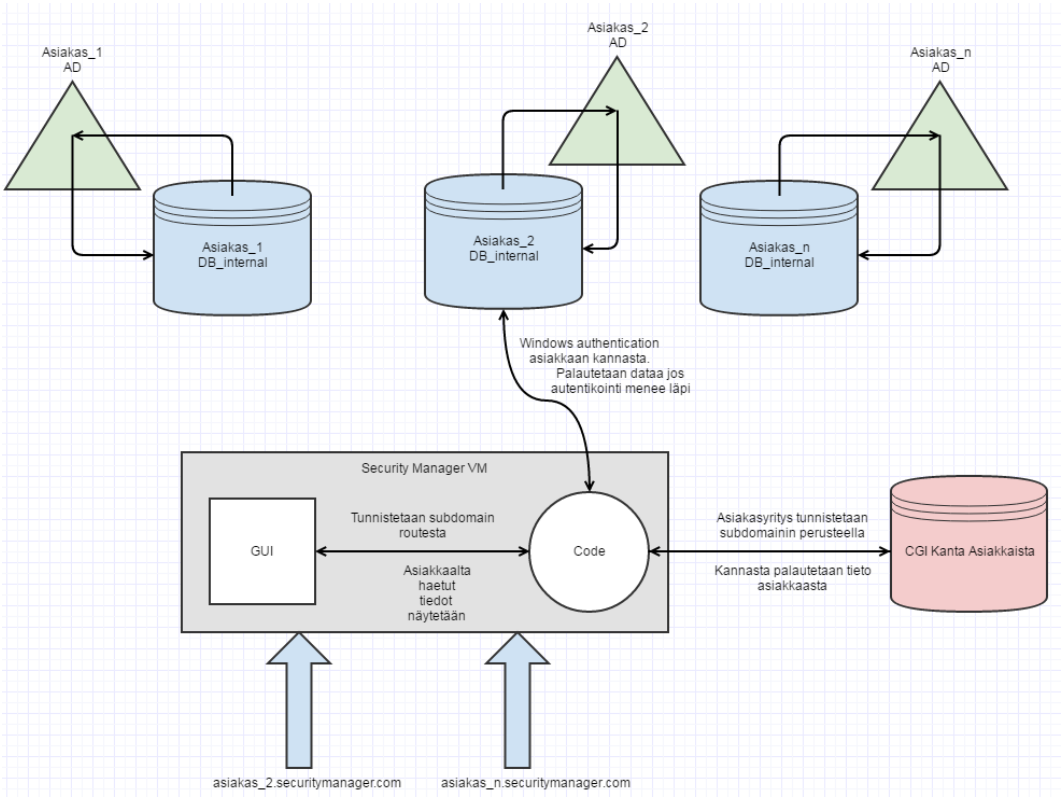
Google, Twitter ja Facebook ovat kaikki SaaS-ohjelmia. Kun normaalisti käyttäjä ostaa ohjelman halutessaan hyödyntää sitä, tilaa SaaS-käyttäjä oikeuden ohjelman käyttöön. Laskutusjakson pituus vaihtelee ohjelman mukaan, mutta varsinkin kuluttajille suunnatuissa ohjelmissa lasku tulee useimmiten kuukauden välein. SaaS on joustava ratkaisu nykypäivän muuttuvaan maailmaan ja sen käyttäjelle tuomia etuja ovat

- pienemmät kustannukset
- automaattiset päivitykset
- joustava pääsy ohjelmaan.

SaaS-ohjelman käyttö laskee asiakkaan kustannuksia monin eri tavoin. Ensinnäkin ohjelmasta maksetaan käytön mukaan ja yleensä tilauksen voi lopettaa milloin vain. Näin asiakas voi käyttää ohjelmaa vain sitä tarvitessaan. Asiakkaalle ei tule myöskään erillisiä asennus- tai laitekustannuksia, kun ohjelma ja tiedot ovat pilvipalvelimella. (Interoute 2016.)

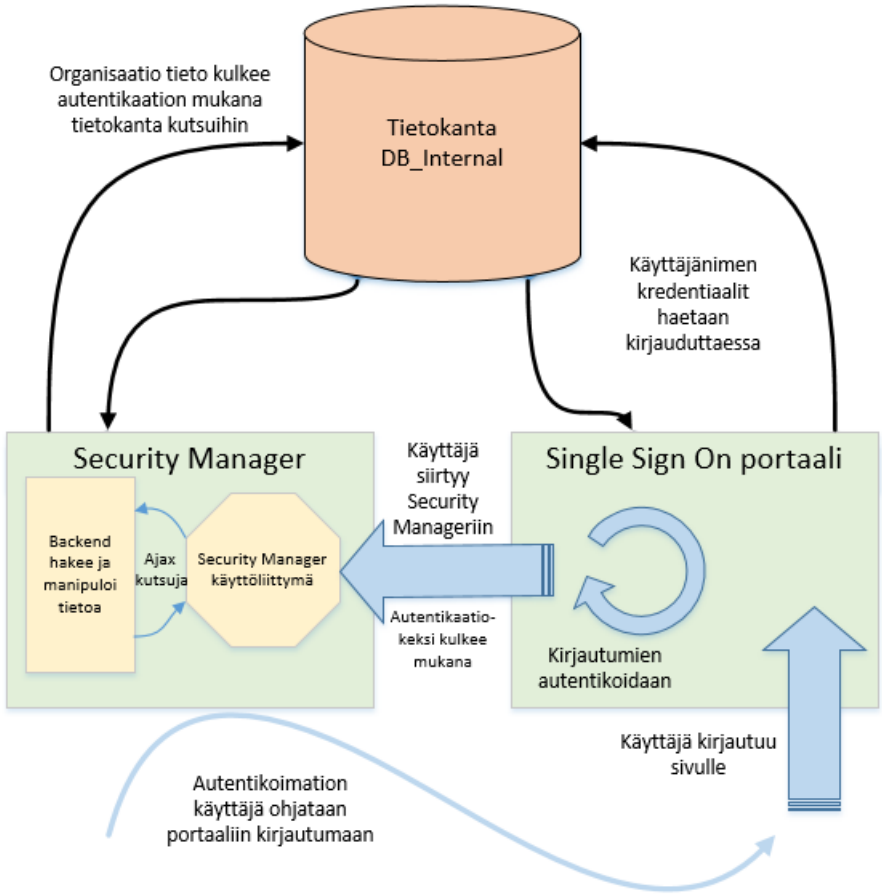
Koska SaaS-ohjelmaa isännöidään keskitetysti, ei asiakkaan tarvitse vastata ohjelman päivityksestä. Jos ohjelmassa havaitaan esimerkiksi tietoturvaongelma, riittää, että pilvipalvelimella isännöitävä ohjelma päivitetään. Kaikki käyttäjät saavat tällöin heti käyttöönsä ohjelman uusimman version. (Interoute 2016.)

Pilvipalvelimella isännöitävälle ohjelmalle yhdistäminen on laiteriippumatonta. Yleensä riittää, että laitteella on toimiva verkkoyhteys. Joskus kuitenkin SaaS-ohjelma voi vaatia toimiakseen jotain toista ohjelmaa estäen näin ohjelman käytön jollakin laitteella. Tällainen tilanne voi ilmetä esimerkiksi pyrittäessä käyttämään puhelimella SaaS-ohjelmaa, joka vaatii toimiakseen jotain tietokoneelle kehitettyä ohjelmistoa. Yleensä tällöin on kuitenkin kyseessä tarkalle kohderyhmälle rajattu ohjelma ja yhteensopimattomuus on tietoinen valinta. (Interoute 2016.)



KUVIO 1. Varhainen konseptisuunnitelma SM SaaS-mallista

Suunnitteluprosessin alkuvaiheessa kehitettiin ideaa puolikkaasta SaaS-mallista, jossa asiakkaiden tiedot sijaitsisivat asiakkaiden omilla järjestelmillä. Tätä suunniteltiin helpottamaan vanhojen on-premises asennuksien siirtämistä SaaS-mallin alle (KUVIO 1). Tässä mallissa tunnistautuminen olisi suoritettu asiakkaan omaa Active Directoryä vasten Windows-autentikaatiota käyttämällä. Jokaiselle asiakkaalle olisi varattu oma aladomaini, josta ohjelma määrittelee, minkä asiakkaan kantaan yhteys luodaan sivustolle tullessa. CGI:n tietokanta sisältäisi taulun asiakasorganisaatioista sekä niiden tietokantoihin yhdistämiseen tarvittavat tiedot. Tämä lähestymistapa osoittautui kuitenkin työlääksi ja haasteelliseksi turvallisuuden suhteen, kun kaikkien asiakkaiden tietokannat olisi pitänyt konfiguroida sallimaan sovelluksen verkon yli lähettämät kutsut.



KUVIO 2. Security Managerin SaaS-malli

Security Managerin SaaS-mallissa päädyttiin lopulta kuvion 2 näyttämään kokonaisratkaisuun, jossa asiakkaiden tiedot sijaitsevat CGI:n hallinnoimassa tietokannassa samalla palvelimella oikeuksienhallinta-sovelluksen kanssa. Sovelluksen käyttämiseen oikeutetut käyttäjä-tunnukset säilytetään tietokannassa ja ne yksilöidään organisaatiotun-nuksella.

Tunnistautuminen suoritetaan Forms-autentikaatiolla Single Sign On -ohjelmassa, joka luo autentikaatiokeksin onnistuneen kirjautumisen jälkeen. Keksi sisältää tiedon käyttäjästä ja organisaatiosta. Security Managerissa organisaatiotieto avataan keksistä ja lähetetään parametrinä tietokantakutsujen mukana. Näin varmistetaan, että haetaan vain kyseiselle organisaatiolle kuuluvat tiedot.

2.2 Kehitysympäristönä Azure

Kaikki projektissa tehty kehitystyö tapahtui Azurella sijaitsevassa virtuaalikoneessa. Azure on Microsoftin luoma pilvipalvelinpalvelu, joka on tarkoitettu ohjelmistojen ja palvelujen kehittämiseen sekä käyttöönottoon. Azuren kehitys julkistettiin lokakuussa vuonna 2008 ja julkaistiin helmikuussa 2010 nimellä Windows Azure. Nimi muutettiin Microsoft Azureksi maaliskuussa 2014. (Microsoft 2014.)

Azure tarjoaa laajan valikoiman erilaisia palveluja aina digitaalisesta markkinoinnista esineiden internetiin. Lisäksi se tarjoaa käyttäjilleen myös rajapinnan, jolla sen tarjoamia palveluita voidaan käyttää. Esimerkiksi virtuaalikoneet voidaan käynnistää kyseisen rajapinnan kautta. Projektissa käytettiin Azuren tarjoamista palveluista vain virtuaalikoineita.

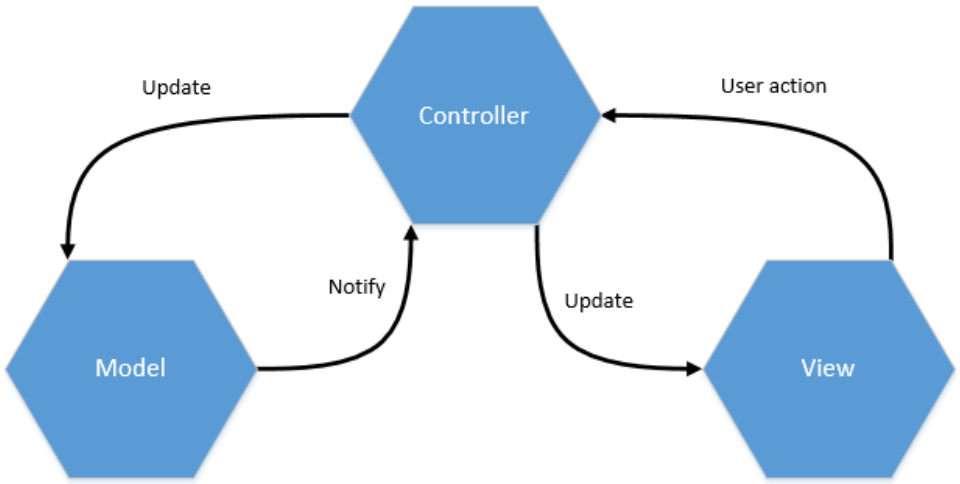
Palveluiden hinnasto ei ole staattinen, vaan se vaihtelee ostettujen palveluiden, alueen sekä käytön mukaan. Esimerkiksi perustason virtuaalikoneen tuntihinta Pohjois-Euroopassa oli 0,118 euroa eli noin 88 euroa kuukaussa ollessaan päällä koko ajan. Tuntihinnat vaihtelivat 0,017 eurosta aina 8,138 euroon riippuen virtuaalikoneen suorituskyvystä. Projektissa käytetty virtuaalikone kuului ylempään keskitasoon, ja sen tuntihinta Azuren laskurin mukaan oli 0,911 euroa. Kyseisessä koneessa oli 28 GB RAM muistia, 1.2 TB keskusmuistia ja 8x2.4 GHz:n proses-soreita. (Microsoft 2017.)

2.3 ASP.NET MVC

ASP.NET MVC on Microsoftin kehittämä verkkosovelluskehys (engl. *web application framework*), jolla voidaan tuottaa dynaamisia verkkosivuja, -sovelluksia ja -palveluita. MVC eli malli-näkymä-ohjain on ohjelmistojen kehitysmalli, jossa ohjelma jaetaan kolmeen lohkoon: malleihin, ohjaimiin ja näkymiin (KUVIO 3). Se on yksi lähestymistapa toteuttaa eräs ohjelmistokehityksen peruskäsitteistä: vastuiden jakaminen eli SoC (engl. *separation of concerns*). MVC-mallin tuomia etuja ovat ohjelman

modulaarinen rakenne, moduulien uudelleen käytettävyys ja ohjelman helpompi ylläpito ja päivittäminen. (Artima 2009; Microsoft 2016a.)

Modulaarisen rakenteen ansiosta voidaan siis MVC-ohjelmissa tarvittaessa tehdä muutoksia käyttöliittymään tai toimintalogiikkaan ilman, että toista tarvitsee muuttaa. Jopa koko moduulin vaihtaminen on mahdollista.



KUVIO 3. MVC (Chrome 2016)

2.3.1 Mallit, näkymät ja ohjaimet

Mallit (engl. *models*) hallitsevat ohjelman logiikkaa, sääntöjä ja tietoa. Yhteydenotot tietokantaan lähtevät malleista ja vastaanotetun tiedon manipulointi tapahtuu malleissa. Mallit eivät riipu näkymistä tai ohjaimista, ja ne voidaan ohjelmoida ja testata erikseen. (Artima 2009.)

Näkymät (engl. *views*) sisältävät käyttöliittymän toimintalogiikan ja ulkoasun. Näkymät ovat kevyesti riippuvaisia ohjaimista. Käyttöliittymän tulee tietää vain ohjainten ja niiden sisältämien toimintojen nimet, joita se kutsuu, sekä se, mitä tietoa kutsun mukana tulee lähettää. Näiden rajojen sisällä näkymiä ja ohjaimia voidaan muokata ilman, että toista tarvitsee muuttaa. (Artima 2009.)

Ohjaimet (engl. *controllers*) itsessään eivät sisällä toimintalogiikkaa, vaan ne toimivat välittäjinä käyttäjältä tulevien käskyjen ja mallien välillä. Ohjaimet sisältävät tiedon siitä, missä muodossa tieto palautetaan: tuodaanko vain tietoa, päivitetäänkö osa sivua vai koko sivu. (Tutorial Points 2016b.)

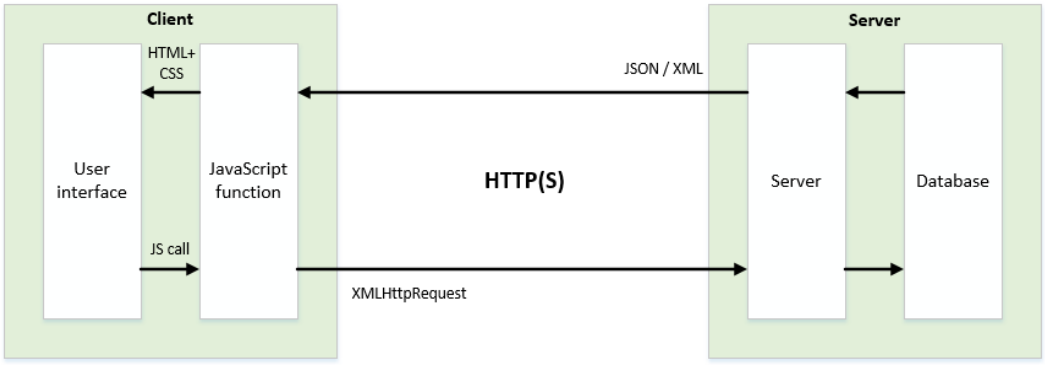
2.3.2 Näkymän ja ohjaimen kommunikointi

Kommunikaatioon näkymän ja ohjaimen välillä päätettiin tässä projektissa käyttää Ajax eli Asynchronous JavaScript and XML -kutsuja niiden tuoman joustavuuden vuoksi. Kun käyttöliittymä kommunikoi Ajaxilla, voidaan näkymät toteuttaa täysin erillisinä malleista ja kokonaan eri tekniikalla. Normaalisti ASP.NET MVC -ohjelmassa näkymistä vastaa Razor-näkymäohjain. Tällöin käytetään hyväksi kielen oliomuotoa lähettämällä näkymiin kulkeva tieto olio-objektissa. Näkymät siis tietävät, missä muodossa tietoa käsitellään malleissa. Ajaxilla toteutettuna kommunikaatio näkymään kulkee JSON-muotoisina objekteina, joita voidaan käsitellä myös muilla tekniikoilla.

Ajax ei ole yksi teknologia, vaan se edustaa laajaa verkkoteknologioiden ryhmää, joilla voidaan toteuttaa kommunikointi palvelimen kanssa ilman, että se häiritsee sivun toimintaa. Ajax-kutsut ovat vuorovaikutteisten sivujen selkäranka. Tällaisten sivujen toteutuksessa yleisesti käytettyjä teknologioita ovat JavaScript, DOM, CSS ja XMLHttpRequest (KUVIO 4). JavaScript-funktiota kutsutaan, kun sivulla tapahtuu ennaltamääritelty tapahtuma, joka vaatii tiedonhakua palvelimelta. Funktiossa luodaan XMLHttpRequest-objekti, joka suorittaa asynkronisen operaation palvelimen kanssa. Tämä on siis Ajax-kutsu palvelimelle. Palvelimelta vastauksena tulevaa tietoa voidaan käsitellä JavaScriptissä. Esimerkiksi tiedon esitysmuodon muuttaminen ohjelmoidusti on helpompaa, kun tyylit eristetään erilliseen Cascading Style Sheet -tiedostoon. Document Object Modelin eli DOM:n avulla tieto ohjataan oikaan kohtaan sivua. (Tutorial Points 2016b.)

Alun perin Ajax-kommunikaatiossa käytettiin tietotyyppinä XML:ää, ja se onkin yhä nimessä mukana. Nykyään XML on monesti korvattu JSON:lla. JSON (JavaScript Object Notation) on kevyt tiedonvaihto formaatti, joka ei ole kieliriippuvainen. Se on johdettu JavaScript olioiden merkintätavasta ja on siten helppo muuttaa natiiviksi JavaScript objektiksi.

Ajax-kutsut toimivat asynkroonisesti eli tahdistamattomasti. Asynkronisuuden ansiosta käyttöliittymä ei pysähdy kutsun suorittamisen ajaksi ja se mahdollistaa tiedonvaihtotason erottamisen presentaatiotasosta. Erottelun avulla pystytään päivittämään sivun sisältöä ilman, että koko sivua tarvitsee ladata uudelleen. (Wrox 2007.)



KUVIO 4. Ajax-kutsu asiakkaalta palvelimelle (Kumar S. 2007)

Ajax-kutsujen asynkronisuuden vuoksi kaikki tietojen lähettäminen ja vastaanottaminen tapahtuu yhteydessä, joka on luotu juuri kyseistä tapahtumaa varten. Kutsu siis lähetetään palvelimella sijaitsevaan ohjaimeen, jossa se jää odottamaan palvelimen suorittamien toimenpiteiden suorittamista. Kun määritellyt toimenpiteet, esimerkiksi tietokantahaku, on suoritettu, lähetetään vastauksen mukaan kuuluva tieto saman kutsun mukana takaisin käyttöliittymään. Koska jokaisen tapahtuman myötä lähetetään uusi kutsu palvelimelle, on tärkeää, että suunnittelussa on otettu huomioon kutsujen aiheuttama viive ja vältetään turhien kutsujen tekemistä. (Wrox 2007.)

2.4 Microsoft SQL Server

Microsoft SQL Server on relaatiotietokantojen hallinnointijärjestelmä eli tietokantapalvelin. Se julkistettiin ensimmäisen kerran vuonna 1989 nimellä SQL Server 1.0. SQL Serverin kehitys on jatkuvaa, ja uusia versioita julkaistaan tasaisin väliajoin. Jokaisesta versiosta on lisäksi useita julkaisuja eri kohderyhmille ja käyttötarkoituksille. Projektissa käytettiin lippulaivaversiota vuodelta 2014.

SQL Server mahdollistaa useiden käyttäjien pääsyn samaan tietokantaan samanaikaisesti. Tämän seurauksena, voidakseen ylläpitää tiedon yhtenäisyyttä, täytyy SQL Serverin pystyä kontrolloimaan resurssien samanaikaista käsittelyä. SQL Serverillä on kaksi tapaa hoitaa tämä: pessimistinen rinnakkaisuus (engl. *pessimistic concurrency*) ja optimistinen rinnakkaisuus (engl. *optimistic concurrency*). (Microsoft 2016b.)

Pessimistisessä rinnakkaisuudessa pääsyä resurssiin kontrolloidaan lukkojen avulla. Lukkoja on kahta tyyppiä: jaettuja ja eksklusiivisia. Jaettuja lukkoja käytetään, kun resurssia luetaan. Tällöin useat käyttäjät voivat lukea samaa resurssia samaan aikaan, mutta sitä ei voida lukita eksklusiivisesti ennen kuin kaikki jaetut lukot ovat auenneet. Eksklusiivistä lukkoa käytetään esimerkiksi resurssia muokattaessa. Tällöin resurssi lukitaan kaikilta muilta käyttäjiltä. (Microsoft 2016b.)

Pääsääntöisesti optimistisen rinnakkaisuuden mallia käytetään ympäristöissä, jossa resurssien käytössä ei juurikaan ole kilpailua. Kun käyttäjä haluaa muokata resurssia, ohjelma joutuu määrittelemään, onko toinen käyttäjä muuttanut resurssia sen lukemisen jälkeen. Koska palvelimen resursseja ei käytetä lukkojen ylläpitoon, saavutetaan tämän mallin avulla keskimäärin parempi suorituskyky ratkaisuissa, joissa todennäköisyys resurssien samanaikaiseen muokkaamiseen on pieni. (Microsoft 2016b.)

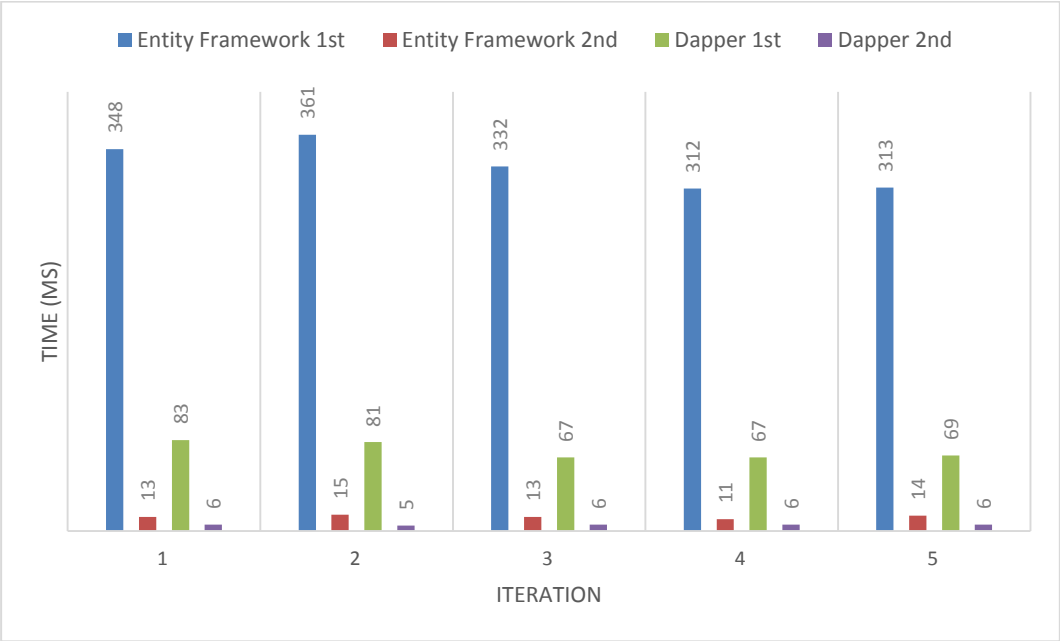
2.4.1 Dapper ORM

Security Managerin tietokantakutsuissa käytetään apuna Dapper ORMia. ORM (engl. *object relational mapper*) on tekniikka, joka mahdollistaa kahden muuten yhteensopimattoman tietotyyppien kommunikoinnin keskenään. Yleisimmin tietokannan ja sovelluksen välissä käytettävän ORM-kehyksen ansiosta ei tietokannan tarvitse tuntea ohjelman käyttämiä objekteja eikä objektien tarvitse tietää mitään tietokannasta. Näin sekä tietokanta että ohjelma voivat molemmat käsitellä tietoa sen natiivissa muodossa. (Learn Now Online 2012; GitHub 2017.)

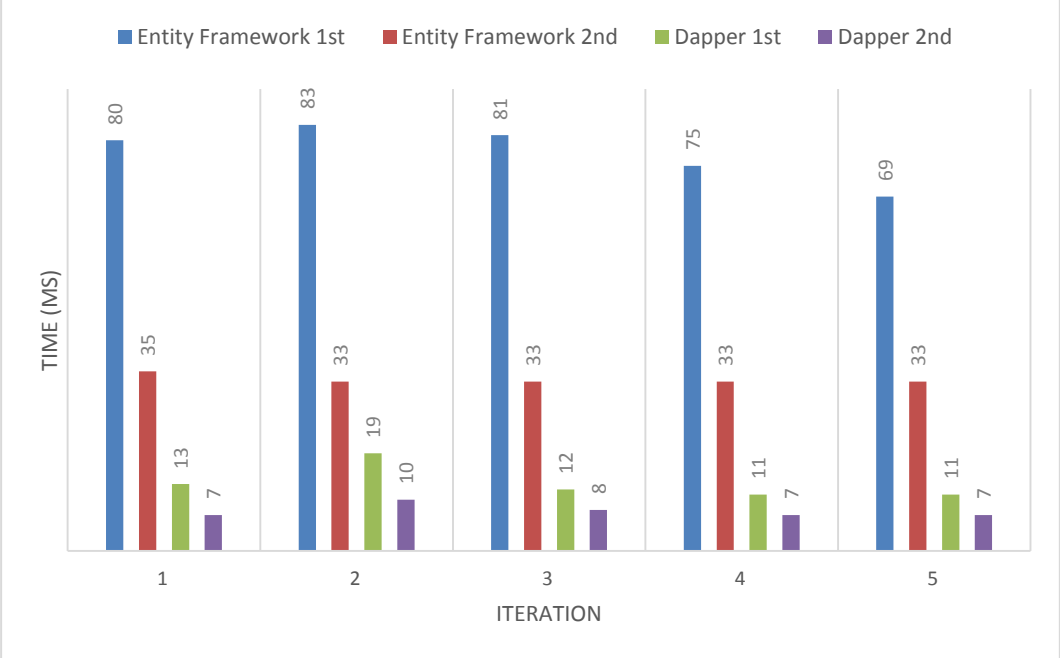
ORMin päätoiminnallisuus on kartoitus, jota se käyttää sitoessaan objektit tietokannan resursseihin. Kartoitus osoittaa, miten objektit ja niiden ominaisuudet liittyvät tietokannan tauluun tai tauluihin. Tätä suhdekarttaa ORM käyttää muuntaessaan tietoa tietokannan ja objektien välillä.

Dapperin kehitti Sam Saffron, joka on yksi stackoverflow.com:n kehittäjistä. Dapper on niin kutsuttu mikro ORM, ja sen suunnittelussa on lähdetty liikkeelle suorituskyvystä ja yksinkertaisuudesta täyden automaation sijaan. Se ei siis muodosta SQL kyselyjä itse kuten esimerkiksi Entity Framework, vaan suorittaa vain objektien kartoituksen ja muunnokset. Tästä syystä Dapper soveltuu hyvin yhdistettäväksi tietokannan tallennettuihin proseduureihin ohjelmissa, jotka eivät käytä isoja ja moniulotteisia tietorakenteita. Myös stackoverflow.com käyttää Dapperia tietokantayhteyksissään. (Orbit One 2015.)

Entity Framework on Microsoftin kehittämä ja .NET ohjelmissa oletuksena oleva täysimittainen ORM. Tämän vuoksi kuvioissa 5 ja 6 esitetyt nopeusvertailut on tehty Dapperin ja Entity Frameworkin välillä. Niistä nähdään, että vaikka molemmissa metodeissa ensimmäinen haku on hidas, korostuu tämä hitaus Entity Frameworkissä. Molemmat metodit kartoittavat tietokannan ensimmäisen kutsun aikana, mutta Entity Framework myös kokoaa kutsun itse.



KUVIO 5. Nopeustesti 1: yksi rivi avaimen perusteella (Don't Panic Labs 2014)



KUVIO 6. Nopeustesti 2: useita rivejä kirjainjonon perusteella (Don't Panic Labs 2014)

Edellä esitetyissä tapauksissa ei käytetty tallennettuja proseduureja vaan Dapperille annettiin ”raaka” SQL lause ja Entity Framework muodosti itse kyselylauseet (GitHub 2014). Erot tuskin ovat yhtä suuret, mikäli kutsut

olisi tehty proseduureihin, joita esimerkiksi Security Manager käyttää. Vaikka erot nopeudessa eivät proseduureja käytettäessä suuria olisikaan, on Dapper silti parempi ORM vaihtoehto Security Managerille sen keveyden vuoksi. Proseduurien vuoksi Security Manager ei tarvitse Entity Frameworkin tuomaa automaatiota SQL kyselyiden muodostuksessa.

2.4.2 Tallennetut proseduurit

Tallennetut proseduurit ovat tietokantaan luotuja aliohjelmia, joita voidaan kutsua muista ohjelmista. Ne sisältävät esikäännettyjä SQL lauseita, joilla voidaan esimerkiksi validoida tietoja tai suorittaa hakuja. Tallennettujen proseduurien etuja ovat

- pienempi verkon kuormitus
- turvallisuus
- helpompi ylläpito
- parempi suorituskyky.

Tallennetut proseduurit voivat vähentää verkkoliikennettä asiakkaan ja palvelimen välillä, koska vain proseduurin suorituskomento lähetetään verkon yli. Mikäli proseduuria ei käytettäisi, pitäisi tietokantaan lähettää kutsun mukana koko SQL lause, joka voi olla hyvinkin suuri. (Microsoft 2016d.)

Turvallisuus on keskeinen osa ohjelmistosuunnittelua ja -kehitystä, ja proseduurit tuovat oman lisänsä siihen. Käytettäessä proseduureja ei käyttäjille tai ohjelmille tarvitse myöntää oikeuksia proseduurin käyttämiin resursseihin. Tällöin proseduuri kontrolloi, mitä prosesseja tai toimintoja suoritetaan, ja suojaa siten allaan olevia resursseja. Proseduurit suojaavat myös asiakkaan ja tietokannan välillä tapahtuvaa liikennettä. Proseduuria kutsuttaessa näkyvissä on vain kutsu proseduuriin, jolloin ilkeämieliset käyttäjät eivät näe tietokannan resurssien nimiä tai pysty sisällyttämään omia SQL lauseita kutsuihin. (Microsoft 2016d.)

Käytettäessä proseduureja tietokantahakuihin voidaan tietoon kiinni pääsy keskittää vain yhteen paikkaan, mikä helpottaa kyselyiden optiomointia ja hallinnointia. Tästä on hyötyä myös, jos tietokantaan täytyy tehdä muutoksia, kun vain proseduureja tarvitsee muokata kutsuvan ohjelman sijaan. (Microsoft 2016d.)

Proseduureilla saavututetaan pääsääntöisesti myös parempi suorituskyky. Kun proseduuri suoritetaan ensimmäisen kerran, luodaan siitä suoritussuunnitelma, jota käytetään seuraavien kutsujen prosessointiin. Tilanteissa, joissa taulu tai haettava tieto muuttuu paljon, voi esikäännetty suoritussuunnitelma aiheuttaa kyselyn hidastumisen. Mikäli kyseessä on muutos tauluun tai tietoon, voidaan proseduuri kääntää uudelleen. Jos taas taulusta haettava tieto esimerkiksi vaihtelee huomattavasti määrältään eri hakuparametreillä, voidaan määritellä, että kutsusta ei tehdä suoritussuunnitelmaa (KUVIO 7). Microsoft SQL Server tukee tätä toimintoa 2008 ja uudemmissa versioissa. (Microsoft 2016d.)

```
ALTER PROCEDURE Demo (@id int)
AS
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductKey = @id
OPTION (OPTIMIZE FOR UNKNOWN) -- Kielletään
                              -- suoritussuunnitelman
                              -- luonti
```

KUVIO 7. Tietokantahakua ei optimoida (Nevarez B. 2010)

2.5 Autentikaatio

Autentikaatio on prosessi, jossa asiakkaan identiteetti varmennetaan. Asiakas voi tarkoittaa tässä yhteydessä henkilöä, tietokonetta, ohjelmaa tai palvelua. Käyttäjän identiteettiä kutsutaan turvallisuuspääomaksi (engl. *security principal*). Autentikoituessaan johonkin järjestelmään asiakas välittää palvelimelle pääsytiedot, joilla palvelin suorittaa varmennuksen. Varmennuksen jälkeen asiakkaan pääomalle voidaan antaa oikeudet suorittaa toimintoja ja käsitellä tietoa. (Microsoft 2005.)

Security Manager on oikeuksienhallintaohjelma ja siten sen käyttäjät normaalisti rajoittuvat muutamaan asiakasorganisaatiota kohti. Koska Security Manager on osa laajempaa ohjelmistokokonaisuutta, oli tärkein kriteeri autentikoitumisessa ”yksi sisäänkirjautuminen” (engl. *Single Sign On*) -toiminnon toteuttaminen. Ohjelmistokokonaisuuden käyttömukavuus paranee, kun jokaiseen ohjelmaan ei tarvitse kirjautua erikseen sisään.

Single Sign On voidaan toteuttaa eri tavoilla, mutta helpointa on luottaa koneen Windows-käyttöjärjestelmän tekemään käyttäjän tunnistukseen. Alun perin sekä on-premises- että SaaS-versioiden oli tarkoitus käyttää tätä Windows-autentikaatiota hyväkseen. Tutkittaessa toteutusta SaaS-järjestelmässä kuitenkin selvisi tämän lähestymistavan ongelmat. Koska Windows-autentikaatiota ei pystytä suorittamaan toisella palvelimella olevaa Active Directoryä vasten, olisi asiakkaiden Active Directoryt pitänyt kopioida Security Managerin palvelimelle. Tämän vuoksi päätettiin SaaS-mallissa hyödyntää ASP.NET -kehyksen tarjoamaa Forms-autentikaatiota, jossa käyttäjän tunnistamiseen käytettävät tiedot säilytetään ja haetaan tietokannasta.

2.5.1 Windows-autentikaatio

Active Directory on Microsoftin luoma hakemistopalvelu Windows-domain verkoille. Se koostuu useista palveluista, mutta niistä tunnetuin ja tämän opinnäytetyön kannalta relevantti on Active Directory Domain Services eli AD DS. Domain Servicen tehtävä on säilyttää tietoa domainin jäsenistä (sisältää käyttäjät ja koneet), autentikoida nämä jäsenet sekä määritellä niiden käyttöoikeudet. (Microsoft 2010a.)

Kun ASP.NET -ohjelma, joka on konfiguroitu käyttämään Windows-autentikaatiota, julkistetaan Internet Information Servicessä eli IIS:llä (3.5), luottaa se IISn suorittavan autentikaation. IIS määrittelee autentikointitavan ohjelman konfiguraatiotiedostosta. Onnistuneen autentikoinnin jälkeen IIS lähettää käyttäjää edustavan Windows-poletin ohjelmalle. Polettia säilytetään ”IPrincipal” -objektin sisällä, joka liitetään

verkkopyyntöihin. Windows-autentikaatio sopii parhaiten sisäverkko-ohjelmiin, joissa sekä ohjelma että palvelin sijaitsevat samalla domainilla. (Microsoft 2005; Microsoft 2010a.)

2.5.2 Single Sign On Forms-autentikaatio

Forms-autentikaatiossa käyttäjä tunnistetaan yleensä tietokantaa vastaan. Tällöin tunnistautuminen täytyy suorittaa ohjelman omassa koodissa käyttäjän kirjautumissivulle antamilla tunnuksilla. Onnistuneen autentikoinnin jälkeen käyttäjän identiteetistä luodaan keksi, joka kulkee ohjelmaan tehtyjen kutsujen mukana. Otettaessa Forms-autentikaatio käyttöön määritellään ohjelman konfiguraatiotiedoston autentikaatio elementissä kirjautumissivun osoite sekä keksin nimi, johon käyttäjän identiteeti tallennetaan (KUVIO 8). Lisäksi kuvion esimerkissä on määritelty keksin voimassaoloaika minuutteina sekä tieto siitä pidennetäänkö aikaa aina käyttäjän tehdessä toimenpiteitä eli ollessa aktiivinen. (Microsoft 2012.)

```
<authentication mode="Forms">
    <forms  name="cookie_name"
            loginUrl="url"
            timeout="480"
            slidingExpiration="true">
    </forms>
</authentication>
```

KUVIO 8. Forms-autentikaatioelementti

Single Sign On ominaisuus on mahdollista toteuttaa Forms-autentikaatiolla helposti, kun kaikki ohjelmat on luotu ASP.NET -kehyksellä. Tällöin riittää, että ohjelmien web.config -tiedostoihin on kirjattu identtiset autententikaatio ja koneavain elementit. Koneavain (engl. *machine key*) elementti määrittelee ohjelman salaukseen, purkuun ja autentikoitumisen varmentamiseen käyttämät avaimet ja algoritmit. Autentikaatiokeksin salaaminen ja purkaminen tapahtuu siis koneavain elementin sisältämillä tiedoilla. Normaalisti IIS generoi isännöimilleen ohjelmille uniikin koneavain

elementin, mutta sen voi ja tässä tapauksessa pitääkin määritellä itse. Koska autentikoitumiskeksin nimi ja sen purkamiseen käytettävä avain ja algoritmi ovat näin ollen esillä konfiguraatiotiedostossa, täytyy se muistaa salata, kun ohjelma julkaistaan. (Microsoft 2010c.)

Kun käyttäjien tietoja säilytetään tietokannassa, on tärkeää huolehtia salasanojen kunnollisesta salauksesta. Lisäksi pitää huomioida, että identiteetin sisältävä keksi ei sisällä salasanoja tai muuta arkaluonteista tietoa käyttäjästä. Salasanojen salaukseen on olemassa valmiita ohjelmia. Microsoft tarjoaa käyttöön PBKDF2 algoritmilla toimivan metodin (KUVIO 9).

```
// derive a 256-bit subkey (use HMACSHA1 with 10,000 iterations)
string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
    password: password,
    salt: salt,
    prf: KeyDerivationPrf.HMACSHA1,
    iterationCount: 10000,
    numBytesRequested: 256 / 8));
```

KUVIO 9. Microsoftin salausmetodin kutsu (Microsoft 2016c)

Metodi ottaa parametreinä vastaan salasanan selvänä tekstinä, suolan, käytettävän salausfunktion, toistojen määrän sekä hash-arvon bittien määrän. Suola on sarja sattumanvaraisia merkkejä, jotka lisätään salananan loppuun ennen kuin se iteroidaan salausfunktiossa hashiksi. Tämä vaikeuttaa salasanojen massamurtamista, mikäli jokin ilkeämielinen taho saisi ne haltuunsa. Salausfunktion iterointi vahvistaa salasanaa tietokoneiden kasvavaa laskentatehoa vastaan vaikeuttaen näin salauksen murtamista voimalla (engl. *brute force*).

Tietokantaan tallennetaan hash, suola sekä iteraatioiden määrä. Kun käyttäjä kirjautuu salasanallansa ohjelmaan, haetaan tietokannasta käyttäjätunnukselle kuuluvat salaustiedot. Koska hash on suunniteltu toimimaan vain yhteen suuntaan, täytyy käyttäjän salasana varmentaa lähettämällä tietokannasta haettu suola ja iteraatio lukumäärä syötetyn salasanan kanssa samaan salausmetodiin. Metodi tuottaa uuden hash-

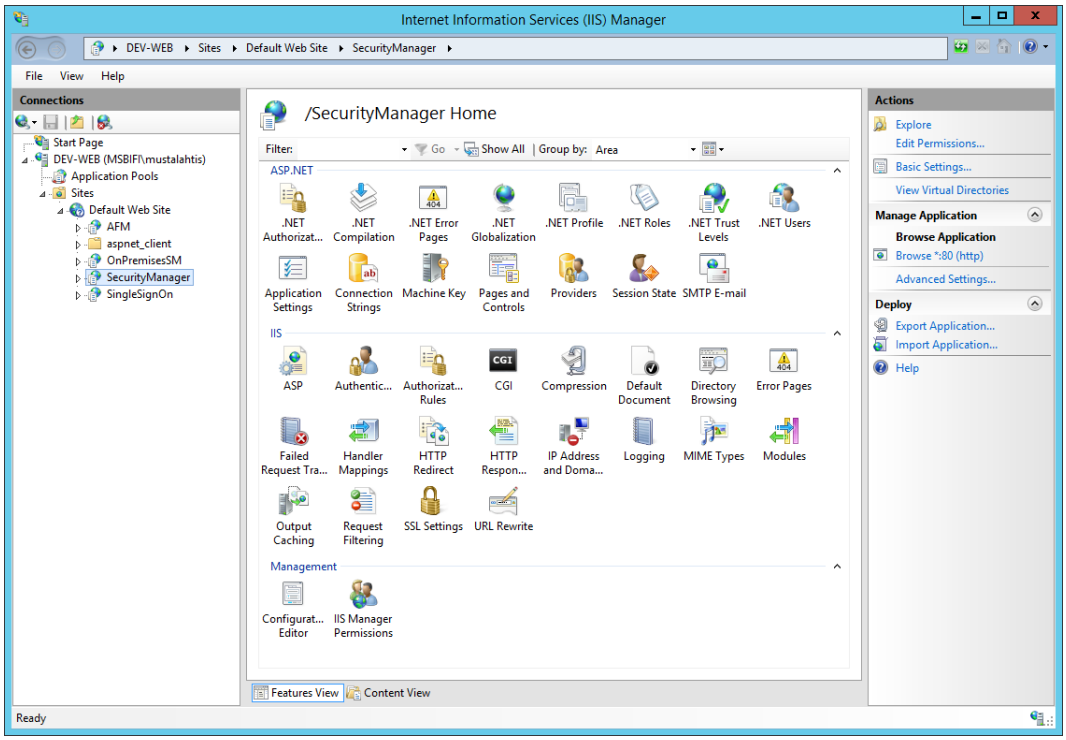
arvon, jota verrataan tietokannasta haettuun arvoon. Mikäli ne ovat identtiset on salasana oikein ja käyttäjä validoidaan. Ainoa hetki, jolloin ohjelma tietää oikean salasanan on silloin, kun käyttäjä kirjautuu sisään oikealla salasanalla. (Microsoft 2016c; Sophos 2016)

2.6 Internet Information Services

IIS on Microsoftin Windows Serverille kehittämä verkkopalvelin-kokonaisuus. Sitä käytetään verkkosivujen ja muun sisällön isännöintiin. Se on rakenteeltaan modulaarinen. Moduuleja hallinnoimalla voidaan palvelimelle lisätä tai poistaa toimintoja. Esimerkiksi Url Rewrite Module mahdollistaa verkko-osoite sääntöjen luomisen palvelimelle. Näin voidaan esimerkiksi ohjata vanhalle verkkosivulle tuleva liikenne uuteen osoitteeseen. (TechTerms 2013.)

IIS tukee seuraavia standardi verkkoprotokollia: HTTP, HTTPS, FTP, FTPS, SMTP ja NNTP. HTTP on hyperteksti protokolla ja toimii tietoliikenteen perustana World Wide Webissä. HTTPS on suojattu versio HTTP:stä. FTP eli File Transfer Protocol on tiedostojen siirtoon käytettävä standardi protokolla ja FTPS on suojattu versio tästä. Simple Mail Transfer Protocollaa (SMTP) käytettään sähköpostiviestinnässä ja NNTP (Network News Transfer Protocol) on SMTP:n kaltainen Usenet -uutisryhmien siirtoon tarkoitettu protokolla. (Microsoft 2010b.)

IIS Manager on graafinen käyttöliittymä verkkosovellusten hallinnointiin (KUVIO 10). Sen avulla ylläpitäjät voivat muokata sovelluskohtaisia konfiguraatioitiedostoja. Muutokset kirjataan sovelluksen web.config tiedostoon, joten samat muutokset voi tehdä muokkaamalla tiedostoa manuaalisesti.



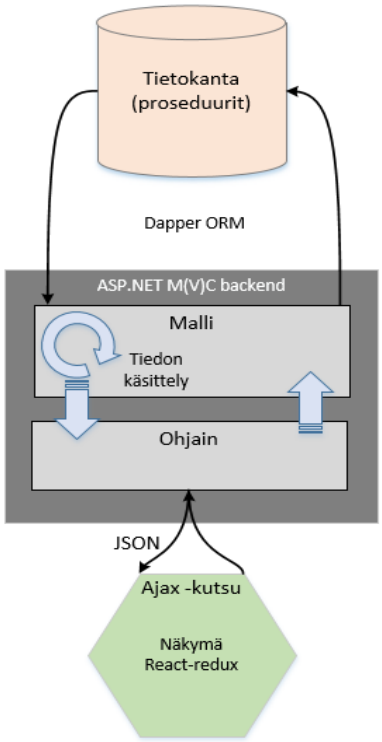
KUVIO 10. IIS Managerin käyttöliittymä

IIS Express on IIS:n kevennetty versio ja saatavilla erillisenä ilmaistuotteena. Se sisältyy myös Visual Studion 2012 versioon ja sitä uudempiin versioihin ohjelmasta. Kehitettäessä ASP.NET ohjelmia toimii IIS Express ohjelman testausalustana. Oletuksena IIS Express sallii vain paikallisen liikenteen ja pyrkii välttelemään konflikteja muiden samalla koneella toimivien verkkopalvelinten kanssa. (Microsoft 2010b.)

3 RAKENNE JA TOIMINTA

Security Managerin toiminta rakentuu normaalin malli-näkymä-ohjain -mallin mukaan, mutta näkymän kohdalla on poikettu ASP.NET MVC:n suosimasta Razor-näkymäohjaimesta. Sen sijaan käyttöliittymä on tehty React-Reduxilla, joka on JavaScript pohjainen vastuiden jakamista toteuttava arkkitehtuurimalli. Siten saavutetaan vielä suurempi irrallisuus mallien ja näkymien välille.

Kuvio 11 näyttää kokonaiskuvan sovelluksen rakenteesta. Kuviosta huomataan kolme isoa kokonaisuutta: tietokanta, backend eli sovelluksen tietojen haku- ja manipulointilogiikka sekä frontend eli käyttöliittymä ja siihen kuuluva logiikka. Rakenteen ansiosta mikä tahansa näistä kolmesta osasta on helposti vaihdettavissa ilman, että muita kahta tarvitsee muuttaa.



KUVIO 11. Security Managerin rakenne

Kommunikointiin ohjaimen ja näkymän välillä käytetään Ajax-kutsuja. Kutsussa määritellään ohjain ja toiminto, jota kutsutaan. Toiminnot eli niin

kutsutut ActionResultit puolestaan kutsuvat mallien metodeja, jotka suorittavat kutsut tietokannassa sijaitseviin tallennettuihin proseduureihin. Tarvittaessa tietokannasta palautettavaa tietoa manipuloidaan mallissa, ennen kuin se palautetaan takaisin ohjaimen ActionResulttiin. Tietokannasta haettu tieto palautetaan Ajax-kutsun mukana takaisin käyttöliittymään JSON-muodossa.

3.1 Ohjaimet

Ohjelmassa on kaksi ohjainta. Niiden tehtävä on vastaanottaa Ajax-kutsuja käyttöliittymästä ja päivittää malleja (KUVIO 3, 2.3). HomeController on ohjelman pääohjain, jonka kautta kaikki tietojen haku ja manipulointi tapahtuu. SettingsController on varattu vain asetuksien muuttamiselle eli esimerikiksi kielen vaihtamiselle. Ohjain ohjaa saamansa Ajax-kutsun oikeaan toimenpiteeseen. Toimenpiteet voivat palauttaa näkymiä, osittaisia näkymiä tai JSON-muotoista tietoa. Tässä sovelluksessa vain päätoimenpide (Index) palauttaa näkymän. Muut toimenpiteet palauttavat aina JSON-muotoista tietoa takaisin käyttöliittymään. Tieto haetaan malleista.

Tiedon hakeminen malleista suoritetaan Retry nimisen apuluokan avulla (KUVIO 12). Retryn tarkoituksena on lisätä koodin virheensietokykyä ja stabiiliutta estämällä ohjelmaa lopettamasta toiminnon suorittamista ensimmäisen virheen jälkeen. Retrystä lisää 4.2.3.

```
[HttpPost]

public ActionResult PersonData(UserKey key)
{
    try
    {
        return Json(new
        {
            success = true,
            data = Retry.WithInterval(() =>
            _repo.PersonData(key.user_key, ExtractOrganization(Request)), 2, 3)
        });
    }
    catch (Exception e)
    {
        e.LogError();
        return Json(e.Handle());
    }
}
```

KUVIO 12. Esimerkki ohjaimen ActionResultista

Kuviossa 12 haetaan malleista käyttäjälle tallennettuja tietoja. Ajax-kutsun mukana vastaanotetaan käyttäjän tunnistava avain UserKey, joka lähetetään PersonData-metodille. Mikäli kyseessä on SaaS-periaatteella toimiva ratkaisu, tulee jokaisen Ajax-kutsun mukana myös käyttäjän autentikoiva keksi (engl. *cookie*). Tämä keksi sisältää tiedon käyttäjän organisaatiosta. Mikäli keksiä ei ole, on kyseessä on-premises asennus, jolloin organisaatiotiedot merkitään tyhjäksi. Tällä tavalla sama koodi toimii molemmissa asennustapauksissa. ExtractOrganization-metodi vastaa keksin poimimisesta kutsusta sekä sen salauksen purkamisesta (KUVIO 13).

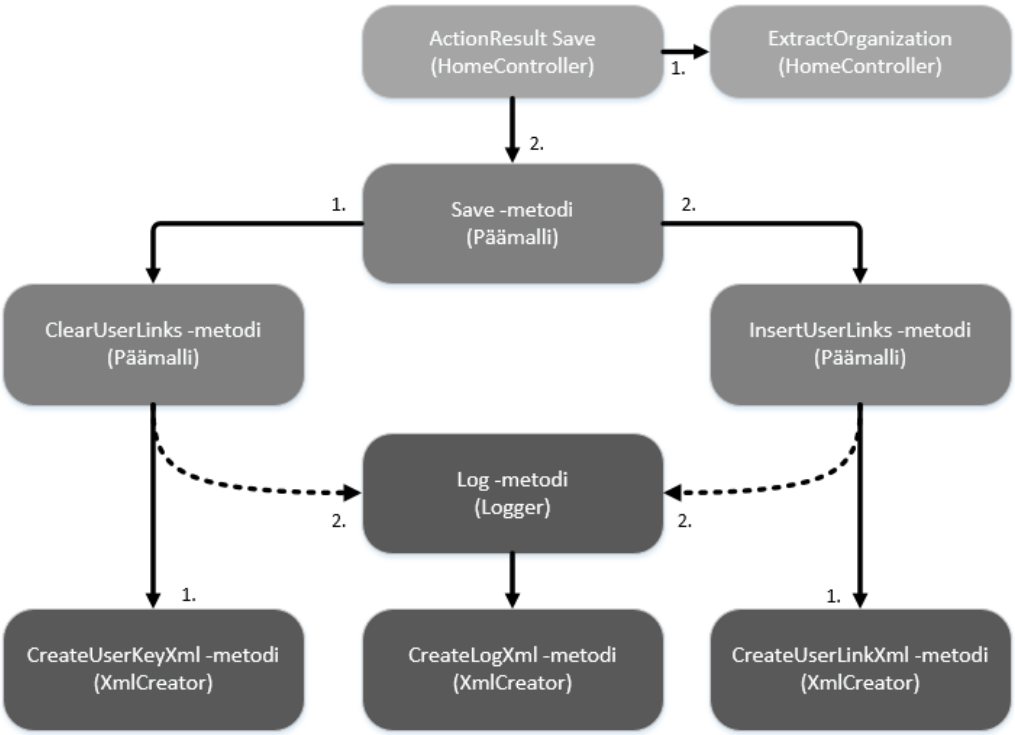
```
protected Organization ExtractOrganization
    (HttpRequestBase request)
{
    Organization org = new Organization();
    if (User.Identity.AuthenticationType == "Forms")
    {
        HttpCookie authCookie =
            request.Cookies[FormsAuthentication.FormsCookieName];
        if (authCookie != null)
        {
            org = new JavaScriptSerializer().
                Deserialize<Organization>
                (FormsAuthentication.Decrypt(authCookie.Value).UserData);
        }
        else
            FormsAuthentication.RedirectToLoginPage();
    }
    else
        org = null;

    return org;
}
```

KUVIO 13. Organisaation erotus kutsusta

3.2 Models

Mallit sisältävät ohjelman raskaan logiikan. Security Managerissa on yksi niin kutsuttu päämalli. Sen tehtävä on toteuttaa tiedonsiirto ohjelman ja tietokannan välilllä sekä tarvittaessa manipuloida vastaanotettua tietoa. HomeController kutsuu ainoastaan päämallia, muiden mallien metodeja kutsutaan tarvittaessa päämallista.



KUVIO 14. Tallennus komennon aloittama kutsuketju

Esimerkiksi kuviossa 14 nähdään tallennuskomennon aloittama kutsuketju. Ohjaimessa kutsusta erotetaan organisaatiotieto, jonka jälkeen toimintakomento kulkee päämalliin Save-metodiin. ClearUserLinks- ja InsertUserLinks-metodit toteuttavat kutsun tietokantayhteyden. XmlCreator luo annetuista tiedoista XML-muotoisen esityksen tietokantaa varten. Kuviossa katkoviiva Log-metodia kutsuttaessa kuvaa erillistä säiettä, jossa lokitus suoritetaan (KUVIO 15). Näin käyttäjälle voidaan Ajax-kutsun mukana palauttaa tieto tallennuksen onnistumisesta ennen kuin Log on vielä kirjannut muutettuja tietoja tietokannan loki tauluun.

```
Thread Log = new Thread(() => Retry.WithInterval(()
    => Logger.Log(true, user_keys, user, org, dimension_keys, group_keys), 5, 6));
```

KUVIO 15. Log-metodin kutsuminen erillisessä säikeessä

Myös Log-metodin kutsuminen tapahtuu Retryn avulla. Koska toiminto tapahtuu erillisessä säikeessä, voidaan tässä tapauksessa pitää pidempi odotusaika yritysten välillä ilman, että ohjelman käyttö häiriintyy.

3.2.1 Päämalli

Kaikki Security Managerin tietokantakutsut toteutetaan Dapperin avulla ja suurin osa niistä sijaitsee päämallissa. Konfiguraatiotiedostossa määritellään ohjelman käyttämä ConnectionString, joka sisältää tietokanta yhteyden luomiseen tarvittavat tiedot: palvelimen ja tietokannan nimen sekä käyttäjänimen ja salasanan, joilla kirjatuminen tietokantaan suoritetaan. Kuviossa 16 viitataan tähän ConnectionStringiin, kun tietokantayhteys luodaan.

```
public List<DimensionDataHierarchy> DimensionData
    (int access_dimension_key, Organization org)
{
    DynamicParameters param =
        new DynamicParameters(new { dimension = access_dimension_key});
    var queryData = QueryParams("get_dimension_data", param, org);

    using (IDbConnection db = new SqlConnection
        (ConfigurationManager.ConnectionStrings["DB_Internal"].ConnectionString))
    {
        List<DimensionDataHierarchy> flat_dims =
            db.Query<DimensionDataHierarchy>((string)queryData[0],
                                             (DynamicParameters)queryData[1],
                                             commandType: CommandType.StoredProcedure)
                                             .ToList();
```

KUVIO 16. Tietokanta kutsujen lähettäminen

Kuviosta 16 nähdään myös, kuinka ohjaimessa kutsusta erotettu organisaatiotieto kulkee tietokantakutsuja lähettäviin metodeihin asti. Ennen tietokantakyselyn lähettämistä QueryParams-metodi määrittelee kyselyn mukana kulkevat parametrit sekä kutsuttavan proseduurin nimen (KUVIO 17). Tietokantakutsua lähetettäessä, kuviossa 16 korostettu rivi, määritellään vastauksena tulevien objektien muoto sekä ilmoitetaan, että kutsu on suunnattu tallennettuun proseduuriin. Parametreiksi annetaan QueryParamsin palauttama proseduurin nimi ja dynaaminen parametripaketti.

```
private object[] QueryParams
    (string procedure_name_base, DynamicParameters param, Organization org)
{
    var result = new object[2];
    if (HttpContext.Current.User.Identity.AuthenticationType == "Forms")
        param.Add("OrgID", org.organization_key);

    result[0] = "security_manager." + procedure_name_base;
    result[1] = param;
    return result;
}
```

KUVIO 17. QueryParams-metodi

QueryParams ottaa parametreina vastaan proseduurin pohjanimen, kutsun vaatimat dynaamiset parametrit sekä organisaatiotiedot. Perustuen käyttäjän autentikaatiotapaan metodi päättelee lisätäänkö organisaatioavain dynaamisten parametrien joukkoon. Lopuksi nimipohjaan lisätään alkupääte. Näin saadaan aikaiseksi tallennetun proseduurin nimi. Toistaiseksi kaikkien proseduurien nimi alkaa ”security_manager.” -liitteellä, mutta tarvittaessa tähän kohtaan voidaan helposti lisätä logiikka alkuliitteen vaihtamiseksi.

```
flat_dims.ForEach(item => { if (item.child_count > 0) { int j = 0;
                                var c = new List<DimensionDataHierarchy>();
                            for (int i = 0; i < flat_dims.Count; i++)
                           { if (flat_dims[i].parent_key == item.access_dimension_data_key)
                                { c.Add(flat_dims[i]); j++; }
                            if(j >= item.child_count)
                                { item.childs = c; break; }
                           } } });
return flat_dims.Where(item => item.parent_key == null).ToList();
```

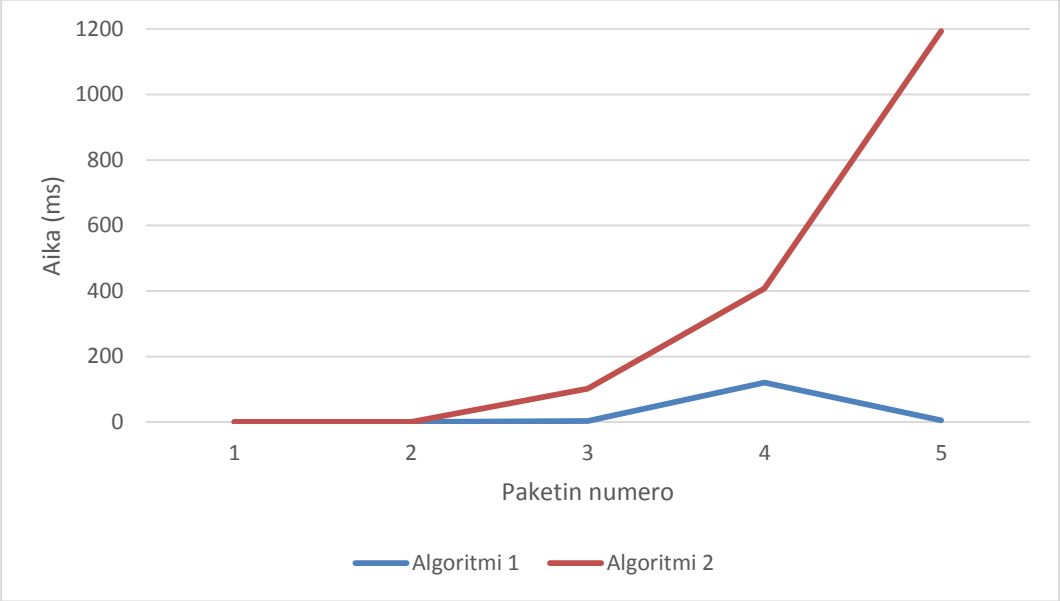
KUVIO 18. Algoritmi, joka muodostaa kuvion 16 datasta puurakenteen

Tässä tapauksessa (KUVIO 16) tietokanta palauttaa lattean listamutoisen esityksen asiakkaan tietokantaobjekteista, jotka lajitellaan algoritmillä hierarkiseen tyyliin. Koska ohjelman pitää sujuvasti käsitellä suuriakin tietomääriä ja puurakenteita, päädyttiin ratkaisussa hyödyntämään myös tietokannan laskentatehoa. Näin ollen kutsuttu proseduuri palauttaa jokaisen objektin kohdalla myös sille kuuluvien lasten lukumäärän. Tällöin

algoritmi voi pitää kirjaa jo löydettyjen lasten lukumäärästä ja lopettaa etsimisen kun määrä on täysi. Tekniikka nopeuttaa puurakenteen luomista huomattavasti.

TAULUKKO 19. Algoritmien nopeus vertailua

	Paketin koko	Rivimäärä	Algoritmi 1 aika	Algoritmi 2 aika	% -nopeampi
1.	1.8 KB	11	228 ms	213 ms	-7,0
2.	256 KB	1827	337 ms	442 ms	23,8
3.	6.1 MB	50304	3.51 s	1.7 min	96,6
4.	12.9 MB	100082	2.0 min	6.8 min	70,6
5.	23.8 MB	164302	5.05 s	19.9 min	99,6



KUVIO 20. Graafi algoritmien suoritusajoista

Taulukossa 19 ja kuviossa 20 on vertailtu kahden eri algoritmin nopeuksia. Algoritmi 1 on sama kuin kuviossa 8 esitetty. Algoritmi 2 toimii muuten täysin kuten algoritmi 1, mutta se ei tiedä objektille kuuluvien lasten lukumäärää. Paketti 4 edustaa vaikeinta mahdollista tilannetta algoritmi 1:lle. Siinä puurakenne on 10 tasoa syvä ja jokaisella tasolla on vain yksi lapsi. Paketti 3 sisältää tietoa, joka rakentuu realistisemmaksi puurakenteeksi. Rakenteen syvyys ja lapsien määrä vaihtelevat siinä huomattavasti. Paketti 5 on erityinen, sillä yhdelläkään sen objektilla ei ole

lapsia. Tällaisen tapauksen kohdalla olisi toki voitu lisätä tietokantaan indikaattori, joka kertoo, että kyseisille tiedoille ei kuulu lapsia. Ratkaisu olisi kuitenkin hyvin jäykkä, jos esimerkiksi tulevaisuudessa yhdelle arvolle haluttaisiinkin antaa lapsi tai lapsia.

Kuviosta 20 voidaan myös huomioida algoritmillä 2 eksponentiaalinen kasvu käsittelyajan suhteessa haettujen rivien määrään. Kun rivien määrä kaksinkertaistuu, aika nelinkertaistuu. 200 000 riviä veisi jo lähes puoli tuntia käsitellä, vaikka yhdelläkään objektilla ei olisi lapsia. Tämä johtuu yksinkertaisesti siitä, että algoritmin on pakko käydä arvolista järjestyksessä läpi ja verrata sitä listan kaikkiin muihin arvoihin.

Päämallin lisäksi ohjelmaan kuuluu apumalleja. Ne eivät yhtä poikkeusta lukuunottamatta ole yhteydessä tietokantaan, vaan vastaavaat tiedon muokkaamisesta ja ohjelman lisätoiminnallisuuksista. Ohjelman apumalleja ovat XmlCreator, Logger sekä Retry. Näiden lisäksi käytetään myös SAASCustomRoles- sekä CustomAttributes-apumalleja, joista tarkemmin kerrotaan luvussa 4.3 Autentikoituminen ja autorisaatio.

3.2.2 XmlCreator

Vanha Security Manager suoritti tiedon lisäämisen tietokantaan lähettämällä yhden kutsun jokaista lisättyä riviä kohti. Tämä lisää verkkoliikennettä ja siten hidastaa ohjelman toimintaa. Koska vanhassa versiossa oli mahdollista tallentaa arvoja vain yhdelle käyttäjälle kerrallaan, ei tämä todennäköisesti muodostunut liian suureksi ongelmaksi. Uudessa Security Managerissa on mahdollista tallentaa arvoja samalla kertaa useille käyttäjille. Tämän vuoksi päädyttiin ratkaisuun, jossa tietokantaan lähetetään vain yksi kutsu. Kutsu sisältää kaikki annetut arvot kaikille valituille käyttäjille XML-muodossa (KUVIO 21).

```
<root>
    <group_link>
        <user_key>16341</user_key>
        <group_key>5</group_key>
        <last_loaded>3/9/2017 5:11:34 PM</last_loaded>
        <modified_by>MSBIFI\mustalahtis</modified_by>
    </group_link>
    <group_link>
        <user_key>16341</user_key>
        <group_key>34</group_key>
        <last_loaded>3/9/2017 5:11:34 PM</last_loaded>
        <modified_by>MSBIFI\mustalahtis</modified_by>
    </group_link>
    <dimension_link>
        <user_key>16341</user_key>
        <dimension_key>361234</dimension_key>
        <last_loaded>3/9/2017 5:11:34 PM</last_loaded>
        <modified_by>MSBIFI\mustalahtis</modified_by>
    </dimension_link>
</root>
```

KUVIO 21. Esimerkki XML-muotoisesta tiedosta

Jokaista lisättävää riviä kohti tiedostoon luodaan yksi solmu (engl. *node*). Solmu sisältää kaiken riville lisättävän tiedon. Tiedot sijoitetaan kahteen eri tauluun. Käytettävä taulu määräytyy solmun nimen mukaan; kuviossa group_link ja dimension_link. Tietokannassa XML:n vastaanottava proseduuri purkaa tiedoston solmukerrallaan ja lisää jokaisen solmun tiedot oikeaan tauluun.

XmlCreator vastaa tämän XML-muotoisen tiedoston luomisesta. Kuten aiemmin kuviosta 14 (3.2) nähtiin, tätä mallia kutsutaan aina tallennuksen yhteydessä päämallista. Tämä apumalli sisältää kolme metodia erilaisten XML-rakenteiden luomiseen. Yksi metodeista luo tiedoston käyttäjäavaimista tietojen poistamista varten. Toinen luo tietorakenteen tietojen tallentamista varten (KUVIO 21). Kolmatta metodia käyttää Logger kirjatessaan lokia ohjelmalla tehdyistä muutoksista tietokantaan.

Taulukossa 22 on vertailtu näiden kahden eri tavan nopeuksia vaihtuvilla tietomäärillä. Siitä nähdään kuinka aika kasvaa lähes lineaarisesti rivimäärän mukana, kun lähetetään yksi rivi kerrallaan. Sinisissä

sarakkeissa olevat arvot osoittavat kauanko XML-muotoisen tiedoston muodostamiseen ja lähettämiseen kului aikaa. Kuten huomataan, XML-tiedoston kanssa toimiminen on huomattavasti tehokkaampi ratkaisu, joka mahdollistaa myös erittäin suurten rivimäärien lisäämisen (100 000 riviä).

TAULUKKO 22. Nopeusvertailutaulukko rivien lisäämisestä tietokantaan

Iteraatio	Ajat (ms)					
	10 Riviä	10 Riviä XML	100 Riviä	100 Riviä XML	1000 Riviä	1000 Riviä XML
1	261	33	2424	59	20167	164
2	224	56	2188	62	20785	129
3	218	42	1958	38	20758	130
4	229	47	2143	45	21509	196
5	239	26	2083	47	20233	202
6	292	43	1948	52	21043	152
7	295	63	2111	36	20895	179
8	223	37	2062	56	20518	140
Keskiarvo	248	43	2115	49	20739	162
100 000 Riviä XML noin 11,5 sekuntia						

3.2.3 Retry

Retry on laajennusluokka, joka sisältää WithInterval-metodin. Sen tehtävä on tuoda ohjelmaan virheensietokykyä sekä selventää koodia vähentämällä turhaa toistoa. Laajennusmetodit ovat staattisia metodeja, joiden ensimmäinen parametri ilmaisee, minkä tyyppisten objektien kanssa metodi on tarkoitettu toimimaan. Esimerkiksi kuviossa 23 esitelty WithInterval-metodi on tarkoitettu käytettäväksi kutsuttaessa muita metodeja. Toistettavan metodin lisäksi parametreina annetaan minimi intervalli, joka odotetaan toistojen välissä sekä maksimi toistojen määrä.

```
public static T WithInterval<T>
    (Func<T> action, int min_interval = 3, int max_attempts = 3)
{
    var exceptions = new List<Exception>();

    for (int i = 0; i < max_attempts; i++)
    {
        try
        {
            if (i > 0)
                Thread.Sleep(min_interval.GenerateInterval());

            return action();
        }
        catch (NullReferenceException e)
        {
            throw new NullReferenceException(e.Message);
        }
        catch (Exception ex)
        {
            exceptions.Add(ex);
        }
    }
    throw new AggregateException(exceptions);
}
```

KUVIO 23. Retry laajennusmetodin toimintalogiikka

Retryn tarkoitus on estää toiminnon suorittamisen lopettaminen heti, mikäli suorituksessa ilmenee virhe. Sen sijaan virheen tapahtuessa odotetaan hetki ennen uutta yritystä suorittaa toiminto. Odotusaika saadaan lisäämällä vähimmäisodotusaikaan arvottu desimaaliluku 0 ja 1 väliltä. Näin toimitaan, kunnes toiminto suoritetaan onnistuneesti tai määritelty määrä uudelleenyrityksiä täyttyy. Retryä käytetään try-catch-lohkon sisällä. Jos toiminnon suorittaminen ei onnistu, nappaa catch-osio tapahtuneen virheen kiinni. Tällöin virhe kirjataan tietokantaan ja käyttäjälle voidaan palauttaa ilmoitus tapahtuneesta virheestä.

3.2.4 Logger

Logger poikkeaa muista apumalleista siinä suhteessa, että myös se lähettää kutsuja tietokantaan. Se kuitenkin mielletään apumalliksi, koska sen käsittelemät tiedot eivät näy suoraan käyttäjille ja sitä ei kutsuta ohjaimista. Loggerin tarkoitus on kirjata tietokantaan tiedot siitä kuka, keneltä ja mitä oikeuksia on päivitetty ja milloin tämä on tapahtunut.

Lisäksi Logger huolehtii virheiden kirjaamisen tietokantaan siinä tapauksessa, että Retry luokka ei ole onnistunut suorittamaan toimintoa yrityksistä huolimatta. Loggerin toteutuksessa on pyritty mahdollisimman universaaliin ratkaisuun. Luokka sisältää kaksi metodia, yhden virheiden ja toisen muutosten kirjaamiseen.

```
public static void Log
    (bool is_insert, string[] user_keys, string current_user,
    Organization org, string[] dimension_keys = null, string[] group_keys = null)
{
    var tmp = new Logger();
    string xml = XmlCreator.CreateLogXML(null, is_insert,
        tmp.CreateLogValueDictionary
        (is_insert, user_keys, dimension_keys, group_keys), current_user);

    var result = EnhanceData(new DynamicParameters(new { Data = xml }), org);

    if (!xml.Equals(false.ToString()))
    {
        using (IDbConnection db = new SqlConnection
            (ConfigurationManager.ConnectionStrings["DB_Internal"].ConnectionString))
        {
            db.Query<string>("security_manager.write_log_entry",
                             result,
                             commandType: CommandType.StoredProcedure);
        }
    }
}
```

KUVIO 24. Loggerin muutostenkirjauksen toimintalogiikka

Kuviossa 24 on esiteltynä Log-metodi, joka vastaa Security Managerissa tehtyjen muutosten kirjaamisesta tietokantaan. ”Is_insert” -arvo määrittelee, onko kyseessä oikeuksien poisto vai lisäys. Oikeudet tallennetaan poistamalla ensin valituilta käyttäjiltä kaikki oikeudet, jonka jälkeen lisätään valitut oikeudet uudelleen tietokantaan. Näin ollen jokaisen tallennustapahtuman yhteydessä kirjataan loki tauluun sekä is_insert = 0, että is_insert = 1. Jotta saadaan lokia varten selville, mitä arvoja keneltäkin käyttäjältä poistetaan, suoritetaan tietokanta haku loki-tauluun jokaisen valitun käyttäjän avaimella. Haku palauttaa viimeisimmän käyttäjää koskevan rivin, jossa is_insert on 1. SaaS-ratkaisussa jälleen lisätään kutsuun lisäksi organisaatiotieto.

TAULUKKO 25. Log -taulu tietokannassa

	log_key	source	is_insert	desc...	user_key	dimension_link_keys	group_lin...	modified_by	entry_date
33	56	Security Manager	0		13907	3275,3276,3277,32...	34,5,3,4,2	34163	2016-12-16 18:51:...
34	60	Security Manager	1		13907	3275,3276,3277,32...	34,4,2	34163	2016-12-16 18:51:...
35	64	Security Manager	0		15417			34163	2016-12-31 13:36:...
36	132	Security Manager	1		15417	5491	3,5,4	34163	2017-02-24 12:13:...

Tietokannassa olevaan lokitauluun kirjataan käyttäjäavain kohtaisesti poistetut tai lisätyt oikeuslinkit, tieto siitä kuka muutokset teki sekä mistä muutokset on tehty (TAULUKKO 25). Tässä tapauksessa henkilöltä 13907 on poistettu kaksi avainta (5 ja 3) ryhmä-linkeistä ja muutokset on tehty Security Managerin kautta. Vaihtoehtoisesti muutokset voisivat olla esimerkiksi automaattisia päivityksiä SSIS-paketeista. Taulussa on myös sarake mahdollisen kuvauksen lisäämiselle tulevaisuutta varten.

3.3 Autentikoituminen ja autorisaatio

Suurimpia haasteita kehitystyössä oli toteuttaa kaksi eri autentikaatio- ja autorisaatiomenetelmää, jotka molemmat ovat luotettavia ja turvallisia. Käytetty menetelmä piti voida vaihtaa vain web.config-tiedostoa muuttamalla, joten koodin täytyi toimia molemmissa tapauksissa. Alun perin tarkoituksena oli toteuttaa molemmille ohjelmaversioille Windowsin Active Directoryyn pohjautuva tunnistautuminen, mutta SaaS-mallin tapauksessa tämä osoittautui liian vaikeaksi toteuttaa ja saatu hyöty kyseenalaiseksi.

3.3.1 On-premises

Kehitysmäärittelyjen mukaan on-premises asennuksessa käyttäjien tunnistaminen pitäisi tapahtua asiakkaan Active Directoryä vasten. Toisin sanoen silloin hyödynnetään asiakkaan kirjautumista koneen Windows-käyttöjärjestelmään. Web.config- eli ohjelman konfiguraatiotiedostoon voidaan tällöin määritellä sekä AD ryhmä- että käyttäjäkohtaisesti, kenellä on oikeus päästä käyttämään Security Manageria (KUVIO 26). Tämä on helpompi tapa, sillä Windowsin kirjautumistietoihin sekä ohjelman kykyyn

lukea Windows-tokenia ilman erityistä konfigurointia voidaan luottaa. Mikäli henkilöllä ei ole oikeutta ohjelmaan, sivua ei ladata ja käyttäjä saa virheilmoituksen.

```
<authentication mode="Windows" />
<authorization>
  <allow roles="MSBIFI\Domain Users"/>
  <deny users="*"/>
</authorization>
<roleManager enabled="true"
             defaultProvider="AspNetWindowsTokenRoleProvider" />
```

KUVIO 26. Web.configissa määritelty Windows-autentikaatio

3.3.2 SaaS

SaaS:n vaatima ratkaisu on huomattavasti hankalampi jo siitä syystä, että toistaiseksi itse SaaS:n toteutuksesta ei ole tarkkoja linjoja. Alkuperäinen suunnitelma oli, että myös SaaS-ratkaisussa hyödynnettäisiin AD:n tietoja. Tutkimuksessa kuitenkin selvisi, että mikäli tällainen ratkaisu otettaisiin käyttöön, jouduttaisiin asiakkaiden Active Directoryt siirtämään samalle palvelimelle, jolla Security Manager sijaitsisi. Koska SaaS oli vasta suunnitelmavaiheessa, ei sen tarkasta toteutustavasta tai ajasta siten ollut varmuutta, kun uutta Security Manageria suunniteltiin. Ratkaisuna päädyttiin kehityksen kannalta helpompaan ja nopeampaan tapaan, jolloin suuria määriä työtä ei mene hukkaan, vaikka lopullinen ratkaisu olisikin jokin muu.

SaaS-tunnistautumiseen käytettiin palvelimella jo olevaa tietokantaa, johon Security Managerikin ottaa yhteyden. Tähän tietokantaan lisättiin uusia tauluja SaaS-organisaatioille, käyttäjille sekä heidän rooleilleen. Lisäksi luotiin toinen ASP.NET MVC -ohjelma, tässä nimettynä Single Sign On, joka vastaa käyttäjien tunnistamisesta SaaS-tauluja vastaan. Single Sign On -ohjelmasta kerrotaan tarkemmin luvussa 3.4.

Security Managerin web.config-tiedostossa määriteltiin ohjelma käyttämään kirjautumisessa Forms-autentikaatiota, joka sijaitsee Single Sign On -ohjelman osoitteessa (KUVIO 27). Kirjautumisen yhteydessä luodaan käyttäjän tunnistava keksi. Käyttäjän tullessa Security Managerin sivulle yrittää ohjelma löytää kyseisen keksin, jonka nimeksi on määritelty kuviossa 27 ”.SSOAUTHCGI”. Mikäli keksiä ei löydy ohjataan käyttäjä Single Sign On -ohjelmaan kirjautumaan sisään. SSO-ominaisuus on toteutettu määrittelemällä molemmille ohjelmille samat koneavain- ja autentikaatioelementit konfiguraatiotiedostossa. Tällöin ratkaisuun on helppo myös myöhemmin lisätä muita ASP.NET:llä tuotettuja ohjelmia.

```
<authentication mode="Forms">
  <forms name=".SSOAUTHCGI"
         loginUrl="SSO/Account/Login"
         timeout="480"
         slidingExpiration="true"></forms>
</authentication>
```

KUVIO 27. Forms-autentikaatio SaaS-ratkaisussa

Käyttäjiä ei rajata web.config-tiedostossa kuten on-premises-ratkaisussa, sillä siitä seuraisi käyttäjän kannalta epäselvä tilanne autorisoimattoman käyttäjän tapauksessa. Käyttäjä nimittäin ohjattaisiin suoraan takaisin Single Sign On -sivulle, jolta hän ohjelmaan oli tullutkin. Tämä kaikki tapahtuu niin nopeasti että käyttäjälle näyttää siltä että Security Managerin linkki ei toimi. Tästä syystä käyttäjien autorisointi suoritetaan itse käyttämällä SAASRoleProvider-luokkaa Security Managerissa, jolloin voidaan luoda myös kustomoitu Authorize-atribuutti suojaamaan ohjaimia ja niiden sisältämiä toimenpiteitä. Kuviossa 28 on Security Managerin konfiguraatiotiedostossa määritelty käytettäväksi SAASRoleProvideria. Huomion arvoista on lisäksi todeta, että roolien tallentamista keksiin ei sallita. Tämä lisää ohjelman suorittamia tietokanta kutsuja, kun käyttäjän roolit tarkastetaan jokaista kutsua varten. Edut tulevat kuitenkin esiin käyttäjän kirjautuessa ulos. Single Sign On -ohjelman ei tarvitse informoida Security Manageria poistamaan keksiä, jossa käyttäjän roolit sijaitsisivat.

```
<roleManager cacheRolesInCookie="false"
             defaultProvider="SAASRoleProvider"
             enabled="true">
  <providers>
    <clear />
    <add name="SAASRoleProvider"
         type="backend.Models.Security.SAASRoleProvider" />
  </providers>
</roleManager>
```

KUVIO 28. RoleProviderin määrittely web.configissa

SAASRoleProviderin toiminta periaate on yksinkertainen, käyttäjänimellä haetaan tietokannasta roolit, jotka käyttäjälle on annettu. Saadusta kokoelmasta tutkitaan löytyykö autorisaation vaatimaa roolia niiden joukosta. Mikäli roolia ei löydy, ohjataan käyttäjä SSO-ohjelmassa sijaitsevalle ”pääsy kielletty” -sivulle (KUVIO 29). Sivun osoite määritellään erikseen web.config-tiedostossa. Näin ratkaistaan aiemmin mainittu ongelma konfiguraatiotiedoston suorittamasta käyttäjien uudelleenohjauksesta.

```
if (filterContext.HttpContext.User.Identity.AuthenticationType == "Forms")
{
    if (filterContext.Result is HttpUnauthorizedResult)
        filterContext.Result =
            new RedirectResult
            (ConfigurationManager.AppSettings["AccessDeniedUrl"]);
}
```

KUVIO 29. Forms-autentikaation uudelleenohjaus

3.4 SaaS Single Sign On -portaali

Single Sign On eli SSO on portaali, jonka tehtävä on SaaS-mallissa kirjata käyttäjä sisään ja ulos. Ohjelmasta löytyy myös linkit niille sivuille, jotka kuuluvat sen luoman autentikaatiokeksin piiriin. Tällä hetkellä ainoastaan Security Manager käyttää SSO:ta, mutta muita ASP.NET -ohjelmia on helppo lisätä myöhemmin. Riittää, että ohjelman web.config-tiedostoon

määritellään autentikaatio ja koneavain elementit identtisiksi SSO:n kanssa.

Myös SSO on luotu ASP.NET MVC:llä, mutta siinä näkymät luodaan Razorilla React-Reduxin sijasta. Tästä johtuen ohjelman näkymät eivät ole niin erillään malleista kuin esimerkiksi Security Managerissa.

3.4.1 Tietokanta

Koska SaaS-mallissa käyttäjiä ei voida tunnistaa Windows-autentikaation avulla, on käyttäjätiedot säilytettävä tietokannassa. Tämän vuoksi Single Sign On -ratkaisu vaatii uusia tauluja asiakasorganisaatioista, ohjelmaan oikeutetuista käyttäjistä organisaatiokohtaisesti sekä näiden käyttäjien käyttöoikeustasoista. Lisäksi olemassa oleviin asiakkaiden tiedot sisältäviin tauluihin täytyy lisätä sarake organisaatioavaimelle. Tämän avaimen perusteella käyttäjät saavat oman organisaationsa tietoa, sillä organisaatioavain sisältyy autentikaatiokeksiin ja se lähetetään tietokantakutsujen mukana tietoa haettaessa.

SaaS-käyttäjät sisältävän taulun tietoihin kuuluu käyttäjänimi, sähköpostiosoite, salasana salatussa muodossa, suola, jota salauksessa on käytetty sekä salausalgoritmillä suoritettujen iteraatioiden lukumäärä. Näiden tunnistamisessa käytettyjen tietojen lisäksi taulussa on tieto siitä, milloin käyttäjä on kirjautunut viimeksi sisään, mihin organisaatioon hän kuuluu ja milloin salasana on asetettu vanhentumaan (TAULUKKO 30).

TAULUKKO 30. Kuva käyttäjä taulusta tietokannasta

	organiz...	username	email	hash	salt	iteration	last_login	password_e...
1	1000	CGI-AD	cgiad@te...	hn6GB+dll...	RXmAu...	22913	2017-03-10 ...	2017-06-08...
2	1001	CGI-CA-AD	cgicaad...	NesfAJS3...	1E+7nfp...	28144	2017-03-10 ...	2017-06-08...

Organisaatio taulu sisältää organisaatiolle määritellyn uniikin tunnistusavaimen, nimen sekä organisaation käyttämän kielen Security Managerin monikielisyystukea varten. Uusien taulujen lisäksi Single Sign On tarvitsee muutamia uusia proseduureja käyttäjien tietojen hakuun ja

päivittämiseen, käyttäjien validoimiseen sisäänkirjautumisen yhteydessä sekä organisaatiotietojen hakemiseen.

3.4.2 Kirjautuminen

Kun autentikoimaton käyttäjä yrittää siirtyä sovellukseen, ohjataan hänet kirjatumaan Single Sign On -portaaliin. Portaalissa käyttäjä syöttää tunnuksensa, jotka ohjelma tarkistaa tietokannasta haettuja tietoja vasten. Mikäli kirjatuminen hyväksytään, ohjataan käyttäjä takaisin uudelleenohjauksen suorittaneeseen sovellukseen autentikoituneena.

Kun käyttäjä yrittää kirjautua sisään, hakee ohjelma annetulle käyttäjänimelle kuuluvat kirjautumistiedot. Näitä ovat siis hash-arvo, suola ja iteraatio. Käyttäjän salasana salataan tietokannasta haetulla suolalla ja iteraatiomäärällä. Tätä uutta salattua salasanaa verrataan tietokannasta haettuun hash arvoon. Jos arvot ovat samat, on salasana oikein ja käyttäjälle luodaan autentikaatiokeksi. Keksi sisältää käyttäjän identiteetin sekä organisaationavaimen ja kielen (KUVIO 31). Tämä keksi liitetään verkkokutsuun josta sisäänkirjautuminen alkoi.

```
FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(
    1,                                      // ticket version
    username,                               // authenticated username
    DateTime.Now,                           // issue date
    DateTime.Now.AddHours(8),               // expiry date
    false,                                  // true to persist across browser sessions
    organization_data,                      // can be used to store additional user data
    FormsAuthentication.FormsCookiePath);   // the path for the cookie

string encryptedTicket = FormsAuthentication.Encrypt(ticket);

// Add the cookie to the request to save it
HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookieName, encryptedTicket);
cookie.HttpOnly = true;
Response.Cookies.Add(cookie);
```

KUVIO 31. Autentikaatiokeksin luonti

Keksin luomisen jälkeen tarkistetaan, onko kyseessä ensimmäinen kirjautuminen, salasana vanhentunut tai tietokannan iteraatioluku liian matala. Käyttäjä ohjataan salasanan vaihtosivulle, mikäli jokin näistä täyttyy. Tällä hetkellä vaadittu vähimmäisiteraatio on 20 000 kertaa. Tätä

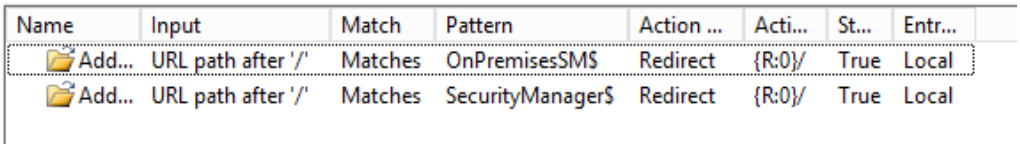
arvoa on hyvä kasvattaa tietokoneiden laskentatehon kasvaessa. Jos salasanaa ei tarvitse vaihtaa, ohjataan käyttäjä joko portaalin pääsivulle, tai mikäli käyttäjä on ohjattu portaaliin toisesta osoitteesta, takaisin lähtösivulle.

3.5 Julkaisu IIS:lle

Molemmat ohjelman versiot on suunniteltu ylläpidettäviksi Internet Information Service palvelimella. Normaalisti ASP.NET -ohjelmat toimivat luotettavasti IIS:llä, ovathan molemmat Microsoftin tuotteita. Security Managerin kohdalla ongelmaksi muodostui kutenkin React-Reduxilla toteutettu käyttöliittymä. Tämän vuoksi ohjelman kansiorakenne poikkeaa normaalista ASP.NET -ohjelmasta. Ongelma ilmeni siirryttäessä Security Manager ohjelmaan ilman, että osoitteen lopussa olisi ollut kauttamerkkiä. Tällöin ohjelma kyllä käynnistyi ja näkymä latautui, mutta kaikki Ajax-kutsut lähtivät ohjelman sijasta IIS:n juureen.

Alkuun ongelmaa pyrittiin ratkaisemaan Security Managerin kautta muokaamalla ohjelman reititystä. ASP.NET MVC -sovelluskehityskehys luo käynnistyksen yhteydessä taulun ohjelman sisältämistä reiteistä ohjainten ja niiden toimintojen (*ActionResult*) pohjalta. Lopulta tämä yritys piti hylätä sen tuomien lukuisten uusien ongelmien vuoksi. Esimerkiksi tietokantaoikeudet eivät enää toimineet tämän jälkeen. Syy tähän jäi epäselväksi.

Ratkaisu löydettiin lopulta Microsoftin omasta ilmaisesta IIS lisäosasta nimeltä URL Rewrite Module. Sen avulla voidaan ohjelmille luoda sääntöjä, joiden mukaan suoritetaan kutsun uudelleenohjaus. Esimerkiksi tässä tapauksessa kutsu ohjataan uuteen osoitteeseen (KUVIO 32). Kun IIS vastaanottaa ehdon (engl. *pattern*) täyttävän kutsun, lopetetaan kutsun suoritus. Toimenpiteen kohdalla merkintä ”{R:0}/” tarkoittaa, että alkuperäisen osoitteen perään lisätään kauttamerkki, jonka jälkeen kutsu ohjataan tähän uuteen osoitteeseen.



Name	Input	Match	Pattern	Action ...	Acti...	St...	Entr...
Add...	URL path after '/'	Matches	OnPremisesSM$	Redirect	{R:0}/	True	Local
Add...	URL path after '/'	Matches	SecurityManager$	Redirect	{R:0}/	True	Local

KUVIO 32. URL Rewrite Module säännöt Security Manager versioille

Ratkaisu on toimiva, mutta ei optimaalinen, sillä URL Rewrite Module ei kuulu normaaliin IIS asennukseen, vaikka onkin Microsoftin tekemä tuote. Moduuli pitää siis ladata ja asentaa erikseen. Tämä aiheuttaa aina yhden ylimääräisen työvaiheen Security Managerin asennukseen.

4 YHTEENVETO

Työn tavoitteena oli toteuttaa korvaava versio Security Manager -oikeuksienhallintasovelluksesta vanhentuvalla teknologialla toteutetun tilalle. Vaikka suurin osa oikeuksienhallinnasta suoritetaan automaattisilla päivityspaketeilla, on manuaalinen käyttöliittymä tärkeä osa kokonaisuudessa. Aina tulee vastaan tilanteita, joissa oikeuksia pitää myöntää erillisen, vaikeasti automatisoitavan logiikan mukaan.

Ohjelman piti pystyä hallitsemaan kasvaneita tietomääriä sujuvasti ja vastaamaan turvallisista ja stabiileista tietokantakutsuista. Erityisenä haasteena kehityksessä ja suunnittelussa oli vaadittu varautuminen tulevaisuuden SaaS-malliin. Varsinkin versioiden erilaiset autentikaatiotekniikat ja tietokantaan tarvittavat rakennemuutokset tuli huomioida toteutuksessa. Projektille asetetut tavoitteet saavutettiin onnistuneesti, vaikka aikataulu hieman venyikin.

Projekti on kuitenkin vielä kaukana valmiista. Seuraava toimenpide on sovellustestaus tuotetulle ohjelmalle. Testauksen on tarkoitus alkaa heti käyttöliittymän valmistuttua keväällä 2017. Tällöin aletaan saada tilastotietoa sovelluksen nopeudesta verrattuna vanhaan versioon sekä käytettävyydestä. Testaus paljastaa myös ohjelmassa piileviä virheitä, joiden korjaaminen onkin seuraava tehtävä. Sovelluksen SaaS-version testauksen ajankohdasta ei vielä tätä opinnäytetyötä kirjoitettaessa ollut varmuutta. Sen ajankohta riippuu SaaS-ratkaisukokonaisuuden implementointiaikataulusta.

Ohjelman jatkokehitys on modulaarisen rakenteen ansiosta helppoa. Tulevaisuudessa pystytään uusien toiminnallisuuksien sisällyttäminen ohjelmaan toteuttamaan huomattavasti nopeammin, kun useampi henkilö voi suorittaa kehitystä eri moduuleissa välittämättä toisten moduulien muutoksista.

Projektin jäljiltä sovelluksella on mahdollista suorittaa samoja toimintoja, kuin vanhalla Security Managerilla. Suunnitteluvaiheessa ohjelmaan

pohditut lisätoiminnallisuudet on pääsääntöisesti myös toteutettu. Näitä olivat esimerkiksi useiden käyttäjien oikeuksien muuttaminen kerralla sekä mahdollisuus tehdä käyttöliittymässä oikeuspohjia. Sovelluksen kehitys jatkunee epäsäännöllisesti. Havaittuja virheitä korjataan ja tarvittaessa toiminnallisuuksia lisätään sovellukseen.

LÄHTEET

Artima 2009. Artikkeli: MVC [viitattu 2.3.2017]. Saatavissa: http://www.artima.com/articles/dci_vision.html

Nevarez B. 2010. Blogi: Optimization for unknown [viitattu 16.3.2017]. Saatavissa: http://www.benjaminnevarez.com/2010/06/how-optimize-for-unknown-works/

CGI 2017. About us: Company overview [viitattu 11.3.2017]. Saatavissa: https://www.cgi.com/en/investors/financial-reports

Chrome 2016. Artikkeli: MVC Acrhitecture [viitattu 2.3.2017]. Saatavissa: https://developer.chrome.com/apps/app_frameworks

Don’t Panic Labs 2014. Blog: Speed Comparison [viitattu 4.3.2017]. Saatavissa: https://dontpaniclabs.com/blog/post/2014/05/01/speed-comparison-dapper-vs-entity-framework/

GitHub 2014. ORM Comparison [viitattu 4.3.2017]. Saatavissa: https://github.com/ceasterday/ORMComparison/blob/master/ORMComparison

GitHub 2017. Dapper [viitattu 4.3.2017]. Saatavissa: https://github.com/StackExchange/Dapper

Interoute 2016. Cloud Articles: What is SaaS? [viitattu 8.3.2017]. Saatavissa: http://www.interoute.com/what-saas

Kumar S. 2007. Artikkeli: AJAX Technologies [viitattu 3.3.2017]. Saatavissa: https://sureshjain.wordpress.com/2007/04/30/ajax-technologies/

Learn Now Online 2012. Blog: Benefits of Object-Relational Mapping (ORM) [viitattu 4.3.2017]. Saatavissa: http://blogs.learnnowonline.com/2012/08/28/4-benefits-of-object-relational-mapping-orm/

Microsoft 2005. Artikkeli: Explained Windows Authentication [viitattu 6.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/ff647076.aspx

Microsoft 2010a. Artikkeli: Active Directory [viitattu 8.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/bb742424.aspx

Microsoft 2010b. Dokumentti: Internet Information Services [viitattu 8.3.2017]. Saatavissa: https://technet.microsoft.com/en-us/library/bb742405.aspx

Microsoft 2010c. Dokumentti: Machine Key Element [viitattu 7.3.2017]. Saatavissa: https://msdn.microsoft.com/library/w8h3skw9(v=vs.100).aspx

Microsoft 2012. Artikkeli: Forms Authentication Overview [viitattu 7.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/7t6b43z4.aspx

Microsoft 2014. Tiedote: Azure [viitattu 2.3.2017] Saatavissa: https://azure.microsoft.com/en-us/blog/upcoming-name-change-for-windows-azure/

Microsoft 2016a. Artikkeli: Key Principles of Software Architecture [viitattu 2.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/ee658124.aspx

Microsoft 2016b. Artikkeli: Optimistic Concurrency [viitattu 3.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/aa0416cz(v=vs.110).aspx

Microsoft 2016c. Artikkeli: Password Hashing [viitattu 7.3.2017]. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing

Microsoft 2016d. Artikkeli: Stored Procedure (Database Engine) [viitattu 4.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/ms190782.aspx

Microsoft 2017. Products: Calculator [viitattu 1.3.2017]. Saatavissa: https://azure.microsoft.com/en-us/pricing/calculator/?service=virtual-machines

Orbit One 2015. Blog: Object Relational Mappers [viitattu 4.3.2017]. Saatavissa: http://blog.orbitone.com/post/Introduction-to-DapperNet3

Sophos 2016. Artikkeli: Password Safety [viitattu 7.3.2017]. Saatavissa: https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/

Tech Terms 2013. Määritelmä: IIS [viitattu 8.3.2017]. Saatavissa: https://techterms.com/definition/iis

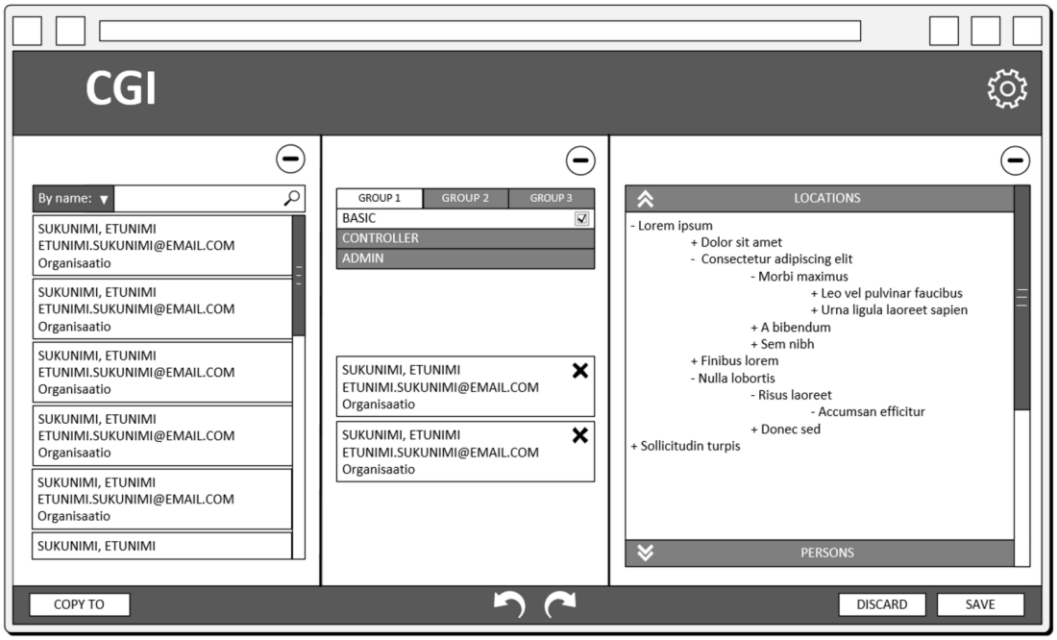
Tutorial Points 2016a. Artikkeli: Ajax Technologies [viitattu 2.3.2017]. Saatavissa: https://www.tutorialspoint.com/Ajax/Ajax_technology.htm

Tutorial Points 2016b. Artikkeli: MVC Framework: Introduction [viitattu 2.3.2017]. Saatavissa: https://www.tutorialspoint.com/mvc_framework/mvc_framework_controllers.htm

Wrox 2007. Artikkeli: What is Ajax [viitattu 3.3.2017]. Saatavissa: http://www.wrox.com/WileyCDA/Section/id-303217.html

LIITTEET

LIITE 1 Security Manager ulkoasu suunnitelma



LIITE 2 Single Sign On -portaalin käyttöliittymä