SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

MAESTRONG-PILVIPALVE-LUN KÄYTTÄJÄHALLINNAN UUDISTAMINEN

Mediamaestro Oy

TEKIJÄ/T: Samu Malinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Samu Malinen	
Työn nimi MaestroNG-pilvipalvelun käyttäjähallinnan uudistaminen	
Päiväys 8.4.2020	Sivumäärä/Liitteet 58/1
Toimeksiantaja/Yhteistyökumppani(t) Mediamaestro Oy	

Tiivistelmä

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa uusi käyttäjäystävällisempi, helpommin ylläpidettävä ja ominaisuuksiltaan aiempaa toteutusta laajempi käyttöliittymä yrityskohtaisten käyttäjäroolien ja käyttäjien sekä niiden käyttöoikeuksien hallintaan MaestroNG-pilvipalvelussa.

MaestroNG on Mediamaestro Oy:n kehittämä modulaarinen SaaS-periaatteella tarjottava pilvipalvelu, jonka sovellukset kattavat kaupan alan taloushallinnon, toiminnanohjauksen ja tilitoimistoalan tarpeet. Palvelukokonaisuus koostuu lukuisista selainpohjaisista sovelluksista, joita voidaan ottaa käyttöön asiakkaan käyttötarpeen mukaan.

Paremman käytettävyyden ohella yksi tärkeimmistä tavoitteista, johon uuden käyttöliittymän avulla pyrittiin, oli tehdä järjestelmän yritystason käyttöoikeuksien ylläpidosta, kopioinnista ja raportoinnista huomattavasti entistä helpompaa sekä ohjata asiakasinstanssien ylläpitäjiä antamaan käyttöoikeudet omiin ympäristöihinsä ensisijaisesti käyttäjäroolien kautta yksittäisten henkilökohtaisten oikeuksien sijasta ja viedä näin koko järjestelmän käyttäjähallintaa entistä roolipohjaisempaan suuntaan.

Opinnäytetyössä perehdyttiin MaestroNG-järjestelmän käyttäjähallinnan ja vanhan hallintakäyttöliittymän ongelmiin ja niiden ratkaisumahdollisuuksiin sekä yleisesti pääsynhallinnan käsitteeseen ja sen erilaisiin toteutusmalleihin ja käytettävyyteen.

Tämän jälkeen suunniteltiin ja toteutettiin näiden sekä asiakaspalautteen pohjalta uusi käyttöliittymä yrityskohtaisten käyttäjien ja käyttäjäroolien käyttöoikeuksien hallintaan käyttäen moderneita web-teknologioita kuten AngularJS- ja .NET -ohjelmistokehyksiä sekä hyödyntäen ketterän ohjelmistokehityksen menetelmiä.

Työssä päästiin sille asetettuihin tavoitteisiin ja lopputuloksena syntyi uusi, entistä helppokäyttöisempi ja halutut ominaisuudet sisältävä käyttöliittymä käyttäjien ja roolien käyttöoikeuksien hallintaan ja raportointiin yksittäisen asiakasinstanssiin kuuluvan yrityksen tasolla.

Avainsanat
Käytettävyys, käyttäjähallinta, pääsynhallinta, käyttöoikeus, RBAC, pääsylista, REST, AngularJS, .NET, laadunvarmistaminen, käyttäjäryhmä, rooli, SaaS

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Samu Malinen			
Title of Thesis Renewal of MaestroNG User Management			
Date	8 April 2020	Pages/Appendices	58/1
Client Organisation /Partners Mediamaestro Oy			

Abstract

The purpose of this thesis was to design and implement a new user interface for managing company-specific users and user roles and their permissions on the MaestroNG cloud service that is more user-friendly, easier to manage and has more features than the old implementation.

MaestroNG is a cloud service developed by Mediamaestro Oy that is delivered to users via the SaaS delivery model and houses applications for many uses including financial management in the field of trade, ERP and accounting. The modular service package consists of many browser-based applications for different uses that can be activated according to the needs of the customer.

In addition to better usability, one of the main goals that the new management interface had to accomplish was to make managing, copying, and reporting of company level permissions significantly easier than before and to direct the administrators of customer instances to give permissions to their user environments primarily through user roles rather than isolated personal permissions. This would take the user management of the whole system towards a more role-based approach.

In the thesis the various problems present in the user management of the MaestroNG system and the old management interface and their possible solutions in addition to the concept of access control, different access control models and usability were explored.

Based on this and customer feedback a new user interface for managing company-specific users, user roles and their permissions was designed and implemented using modern web technologies such as the AngularJS and .NET frameworks and utilizing methods of agile software development.

The goals set for the thesis were achieved, and as a result a new, easier to use user interface with the planned features for managing and reporting user roles and permissions of individual companies belonging to a customer instance was implemented.

Keywords
Usability, user management, access right, access control, RBAC, access control list, REST, AngularJS, .NET, quality assurance, user group, role, SaaS

SISÄLTÖ

TERMIT/LYHENTEET JA MÄÄRITELMÄT

ACL (Access Control List, pääsylista)	Lista rajoitetun järjestelmän osan pääsy/käyttöoikeuksista
API (Application Programming interface)	Ohjelmointirajapinta, jonka kautta eri ohjelmistot tai niiden osat voivat kommunikoida toistensa kanssa
CSS (Cascading Style Sheets)	Kieli, jota käytetään merkintäkielellä kirjoitettujen dokumenttien (yleensä HTML-verkkosivujen) tyylien kuvaamiseen
C#	Microsoftin .NET-alustalle kehittämä moderni, oliopohjainen ohjelmointikieli
HTML (HyperText Markup Language)	Merkintäkieli, jolla kuvataan web-sivujen rakenne
HTML5	HTML-kielen viimeisin versio, käytetään usein myös yleisnimityksenä moderneille web-tekniikoille ja web-sovellusten toteuttamiselle niitä käyttäen
HTTP (Hypertext Transfer Protocol)	Web-selainten ja WWW-palvelimien tiedonsiirtoon käyttämä protokolla
JavaScript	Ohjelmointi/skriptauskieli, jonka pääasiallinen käyttötarkoitus on dynaamisten toimintojen lisääminen verkkosivuille
JSON (JavaScript Object Notation)	Tiedonvälitykseen käytettävä avoimen standardin tiedostomuoto
REST (Representational State Transfer)	Roy Fieldingin vuonna 2000 määrittelemä arkkitehtuurimalli HTTP-protokollaan perustuvien ohjelmointirajapintojen toteuttamiseen
RBAC (Role-Based Access Control)	Roolipohjainen pääsynhallinta, käyttäjien käyttöoikeudet määrittyvät heille määriteltyjen roolien kautta
Silverlight	Microsoftin kehittämä Flashin kaltainen web-ohjelmointiympäristö, jonka kehitys lopetettiin tietoturvapäivityksiä lukuun ottamatta vuonna 2013
SQL (Structured Query Language)	Relaatiotietokantojen kyselykieli, jonka avulla niihin voidaan tehdä hakuja, lisäyksiä sekä muutoksia

1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa uusi, entistä helppokäyttöisempi ja ominaisuuksiltaan olemassa olevaa toteutusta laajempi versio MaestroNG-pilvipalvelun yritystason käyttäjien ja käyttäjäroolien hallintakäyttöliittymästä jonka avulla käyttäjien ja roolien käyttöoikeuksien raportointi ja ylläpito olisi huomattavasti entistä helpompaa. Työ toteutettiin osana palvelun meneillään olevaa teknologia- ja käytettävyysuudistusta, jossa palvelun vanhemmista, vielä vanhentuneella Silverlight-tekniikalla toimivista käyttöliittymistä toteutetaan vaiheittain uudet, kaikilla yleisimmillä laitteilla ja selaimilla toimivat versiot.

MaestroNG on Mediamaestro Oy:n kehittämä modulaarinen pilvipalveluna toteutettu toiminnanohjausjärjestelmä, jonka eri sovellukset kattavat laajasti sekä kaupan alan taloushallinnon ja toiminnanohjauksen että tilitoimistoalankin ohjelmistotarpeet. Palvelulla on tuhansia loppukäyttäjiä sekä tilitoimistoasiakkaiden, heidän asiakkaidensa, että kaupan alan asiakkaiden puolelta. Palvelukokonaisuus koostuu lukuisista verkkosovelluksista, joita voidaan ottaa käyttöön asiakkaan tarpeiden mukaan.

Järjestelmää tarjotaan käyttäjille SaaS (Software as a Service) -mallin mukaisesti, jossa palveluntarjoaja tarjoaa omilla palvelimillaan pyörivää ohjelmistoa ja sen palveluita asiakkailleen verkon yli maksua vastaan, jonka suuruus riippuu käytettyjen palveluiden tyypistä ja määrästä. SaaS-toimitusmallin sovelluksien käyttöön siirtyminen tarjoaa organisaatioille merkittäviä säästöjä, kun heidän ei tarvitse itse hankkia ja ylläpitää palvelimia, joilla ohjelmistot toimivat, vaan vastuu palvelun ylläpidosta on palveluntarjoajalla ja asiakkaat pääsevät käsiksi tarvitsemiinsa ohjelmiin mistä tahansa verkkoselaimen kautta. (Atos SE, 2020)

Mediamaestro Oy, lyhyemmin Maestro, on vuonna 1986 perustettu suomalainen ohjelmistotalo, jossa työskentelee noin 65 asiantuntijaa kolmella eri paikkakunnalla: Savonlinnassa, Lappeenrannassa sekä Helsingissä. Maestro tarjoaa kaupan ja palvelun aloille sekä tilitoimistoille ja niiden asiakkaille modulaarisia ohjelmistoratkaisuja.

Opinnäytetyöraportin alussa käydään läpi työn lähtökohtia sekä perehdytään MaestroNG-palvelun nykyiseen käyttäjähallintamalliin ja sen ongelmiin, joita uudella käyttäjien ja roolien hallintanäkymällä pyritään ratkaisemaan. Tämän jälkeen tarkastellaan yleisiä käyttöoikeushallinnan malleja ja esitellään työn toteutuksessa käytettyjä teknologioita. Seuraavaksi perehdytään uuden hallintakäyttöliittymän suunnittelu- ja kehitysprosessiin sekä uusien ominaisuuksien että käytettävyyden osalta. Lopuksi esitellään työn tuloksia ja jatkokehitysmahdollisuuksia.

2 TYÖN LÄHTÖKOHDAT JA ONGELMAN KUVAUS

Tässä luvussa kuvataan opinnäytetyön taustat ja aiheen valintaan johtaneet asiat sekä käydään läpi MaestroNG-palvelun käyttäjien ja käyttöoikeuksien hallintamallia ja sen ongelmia, joita uudella käyttäjien ja roolien hallintakäyttöliittymällä lähdettiin ratkaisemaan.

2.1 Työn tausta

Tekijän ollessa Maestrolla työharjoittelussa kesällä 2019 hänen tekemänsä ohjelmointi-, testaus- ja suunnittelutyöt keskittyivät pääasiassa MaestroNG-järjestelmän ylläpitokäyttöliittymien uudistamiseen osana meneillään olevaa järjestelmän teknologia- ja käytettävyysuudistusta: Vanhojen ylläpitokäyttöliittymien toteuttamiseen käytetyn Microsoft Silverlight-tekniikan tuki on loppumassa lokakuussa 2021 ja tästä vanhentuneesta teknologiasta johtuen sillä toteutetut käyttöliittymät ovat käytettävissä ainoastaan laitteilla, joissa on mahdollista käyttää Internet Explorer 11 -selainta, eli käytännössä pelkästään Windows-työasemilla. (Microsoft, 2019). Teknologiauudistuksessa näistä käyttöliittymistä toteutetaan uudet HTML5-standardien mukaiset kaikilla yleisimmillä laitteilla ja selaimilla toimivat versiot käyttäen Silverlightia modernimpia web-teknologioita ja samalla parannetaan käyttöliittymien käytettävyyttä.

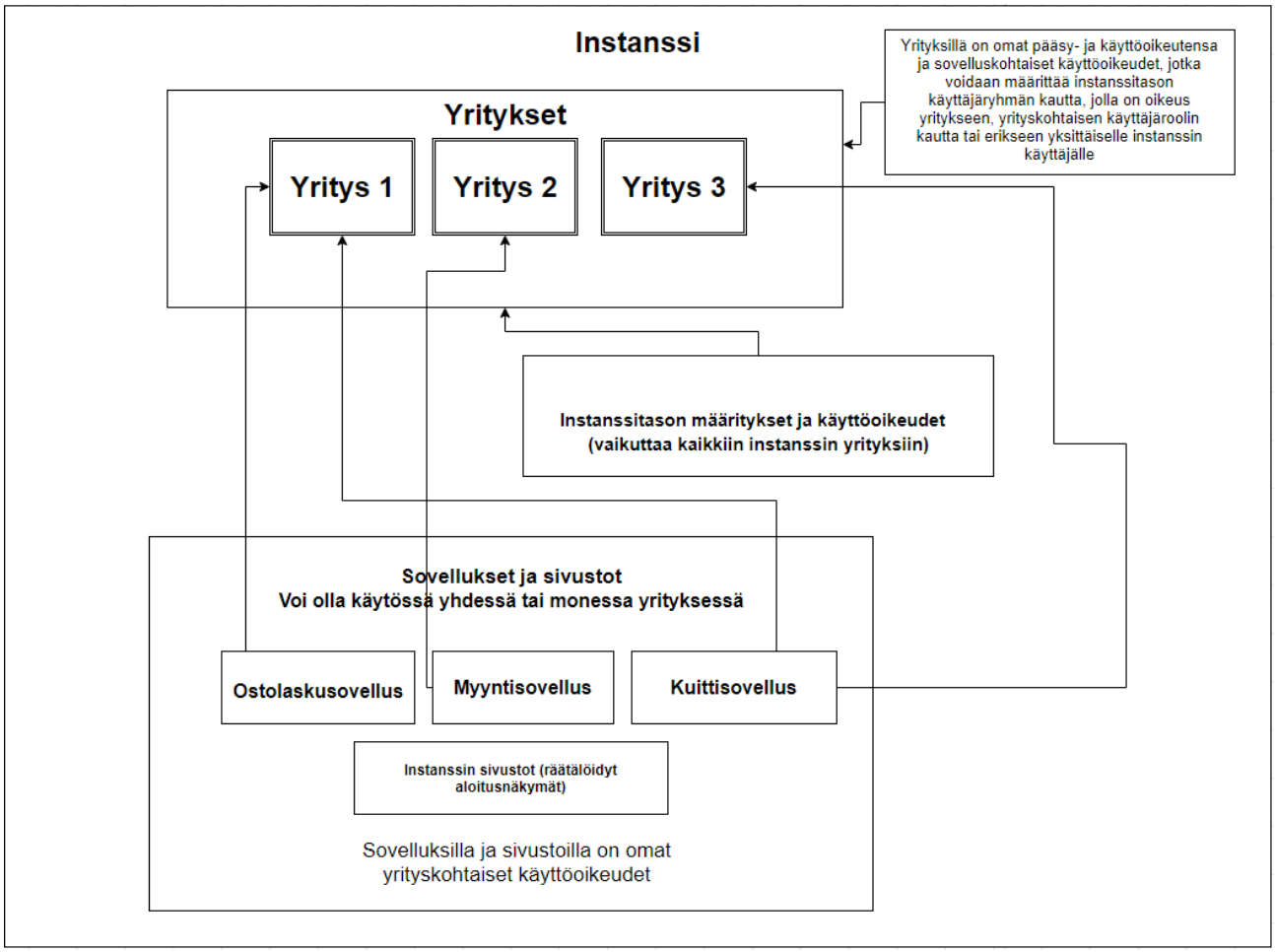
Toimeksiantajan kanssa käytiin jo loppukesästä lähtien keskustelua opinnäytetyön toteuttamisesta Maestrolle, ja että opinnäytetyönkin aihe voisi hyvin liittyä johonkin vielä uudistamatta olevaan hallintakäyttöliittymään. Syksyn edetessä aiheeksi tarkentui järjestelmän yritystason käyttäjien ja käyttäjäroolien ylläpitokäyttöliittymän uudistaminen, josta oli jo olemassa HTML5-versio, joka oli kuitenkin asiakaspalautteen mukaan monimutkainen ja hankala käyttää sekä muutenkin ominaisuuksiltaan puutteellinen useita yleisiä käyttäjähallinnan käyttötapauksia ajatellen, jolloin se ei nykyisellään täyttänyt kyseiselle käyttöliittymälle asetettuja vaatimuksia. Tavoitteeksi työlle asetettiin parannellun käyttöliittymän suunnittelu ja toteutus, jonka avulla järjestelmän käyttöoikeuksien ylläpito, raportointi ja kopiointi käyttäjältä toiselle yritystasolla tulisi huomattavasti entistä helpommaksi, ja käyttöliittymä olisi kuitenkin samalla huomattavasti entistä selkeämpi ja helppokäyttöisempi.

2.2 MaestroNG:n käyttöoikeuksien hallintamalli

Tässä osassa käydään läpi tyypillisen MaestroNG-asiakasinstanssin käyttäjärakenne ja erilaiset tavat, joilla käyttäjän käyttöoikeudet järjestelmässä määrittyvät sekä kuinka niitä voidaan hallita.

2.2.1 Asiakasinstanssi ja käyttäjärakenne

Perinteinen MaestroNG-järjestelmän asiakasinstanssi on esimerkiksi tietyn tilitoimiston järjestelmä, joka sisältää järjestelmässä käytössä olevat sovellukset ja sivustot (eli eri tarkoituksiin räätälöidyt aloitusnäkymät) sekä tilitoimiston asiakasyritykset ja niiden ympäristöt, joilla on omat käytössä olevat sovelluksensa ja pääsy- ja käyttöoikeutensa sekä käyttäjät ja käyttäjäryhmät, joilla on oikeuksia yhteen tai moneen yritystason ympäristöön ja sen sovelluksiin. Instanssitason käyttöoikeudet ja määritykset vaikuttavat kaikkiin instanssin yrityksiin ja instanssitason käyttäjäryhmille voidaan antaa oikeuksia mihin vain instanssin yritykseen. Sovelluksilla ja sivustoilla on omat yrityskohtaiset käyttöoikeutensa, ja käyttöoikeudet voidaan antaa erikseen yksittäisille käyttäjille, instanssitason käyttäjäryhmille tai yritystason käyttäjärooleille. Asiakasinstanssin rakennetta ja käyttöoikeuksien määrittymistä on havainnollistettu kuviossa 1.

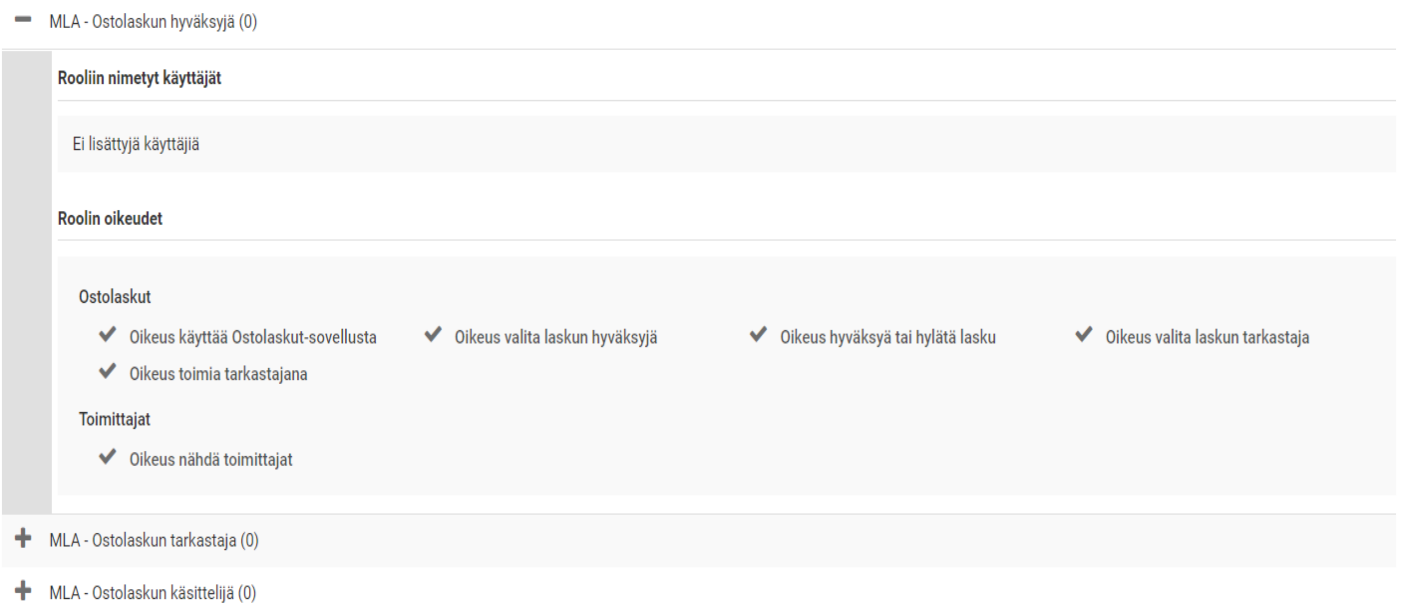


KUVIO 1. Asiakasinstanssin rakenne

2.2.2 Käyttäjäryhmä ja käyttäjärooli

Käyttäjäryhmä on järjestelmän instanssitason ryhmä, jonka avulla voidaan organisoida käyttäjiä. Käyttäjäryhmät voivat olla käytössä monessa eri instanssin yrityksessä ja myös niille voi antaa käyttöoikeuksia yksittäisten käyttäjien lisäksi. Instanssin ylläpitäjä voi muokata olemassa olevia ja luoda uusia käyttäjäryhmiä.

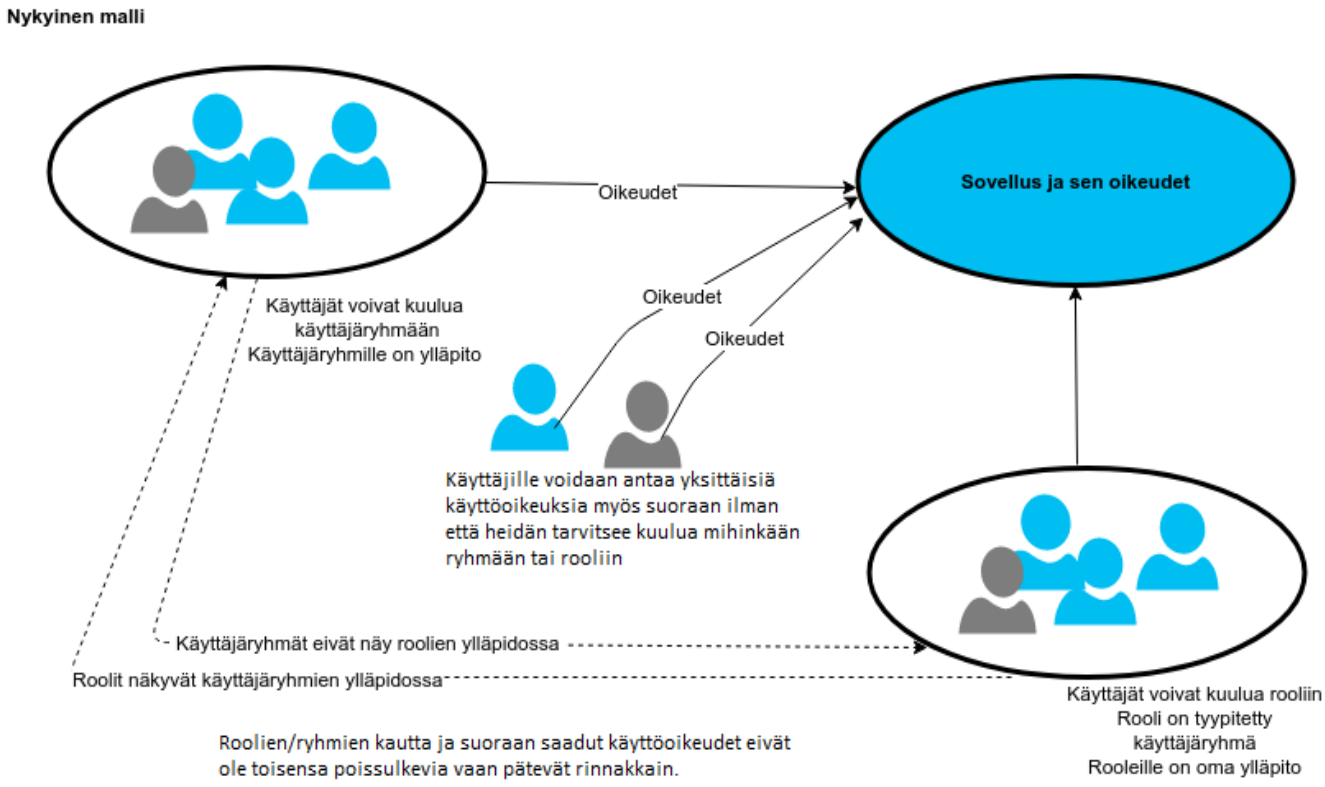
Käyttäjärooli taas on yritystason ryhmä, jolla organisoidaan käyttöoikeuksia. Kun instanssiin perustetaan uusi yritys tai yritykselle asennetaan uusi sovelluslisenssi, luodaan yritykseen automaattisesti käyttäjärooleja, joilla on yritykseen tai asennettuun sovellukseen tiettyjä roolinmukaisia oikeuksia. Tällöin tiettyjen käyttöoikeuksien antaminen roolin mukaiset käyttötarpeet omaaville käyttäjille tulee ylläpitäjälle huomattavasti helpommaksi. Kuvassa 1 on esimerkkinä kuvattu roolit, jotka asentuvat yritykseen, kun yritykselle otetaan käyttöön järjestelmän ostolaskusovellus.



KUVA 1. Ostolaskulisenssin mukana yritykseen asentuvat roolit.

2.2.3 Käyttöoikeuksien hallinta

Kuten edellisissä kappaleissa jo sivuttiin, MaestroNG-järjestelmässä on monia eri tapoja, joilla käyttöoikeudet voivat määrittyä yksittäiselle instanssin käyttäjälle sekä yritys- että järjestelmätasolla. Käyttäjä voi saada oikeuksia yrityksiin ja niissä käytössä oleviin sovelluksiin suoraan, instanssitason käyttäjäryhmien kautta tai yrityskohtaisten roolien kautta. Näille on nykyisessä mallissa omat erilliset ylläpitonsa, ja sovelluksilla on myös omat erilliset käyttöoikeuksien hallintasivunsa. Määrityksiä, joista ja joiden yhdistelmistä käyttäjäkohtaiset käyttöoikeudet voivat määräytyä on siis järjestelmässä useita, ja monien asiakkaiden kohdalla käytäntö on osoittanut, että esimerkiksi pelkästään yrityskohtaisten oletusroolien käyttö ei riitä käyttöoikeuksien määrittämiseen jokaiseen käyttötapaukseen. Kuviossa 2 on havainnollistettu nykyistä käyttöoikeuksien hallintamallia yksittäisen sovelluksen käyttöoikeuksien näkökulmasta.



KUVIO 2. Yksittäisen sovelluksen käyttöoikeuksien hallinta nykyisessä mallissa.

2.3 Nykyisen hallintamallin ongelmat

Nykyinen oikeuksien hallintamalli on monella tapaa ongelmallinen, koska käyttäjän oikeudet voivat käyttäjäryhmän ja roolin lisäksi määrittyä myös suoraan käyttäjälle itselleen, jokaisessa järjestelmän sovelluksessa on oma käyttöoikeushallintansa eikä järjestelmässä ole nykytilanteessa olemassa yhtä keskitettyä paikkaa josta näkisi kaikki ryhmät ja roolit joihin tietyn yrityksen käyttäjät kuuluvat ja mitä kaikkia käyttöoikeuksia heillä sekä roolien kautta että suoraan on ko. yritykseen ja sen sovelluksiin.

Tämä tekee käyttöoikeuksien ylläpidosta, kopioinnista käyttäjältä toiselle sekä raportoinnista paikoin erittäin haastavaa, sillä käyttäjän ryhmiin lisäämisen lisäksi käyttäjille on usein annettu myös suoria henkilökohtaisia oikeuksia koska yritystason oletusroolien tarjoamat käyttöoikeudet eivät useinkaan riitä kaikenlaisten käyttötapausten vaatimien oikeuksien antamiseen.

Nämä hankaluudet korostuvat esimerkiksi tapauksissa, joissa MaestroNG-järjestelmään migroidaan vanhasta Maestro Expera -järjestelmästä käyttäjäryhmiä, joita voi olla suuri määrä hyvinkin moninaisilla käyttöoikeuksilla, tai ollaan perustamassa uutta käyttäjää, jolle pitäisi saada kopioitua samanlaiset oikeudet kuin olemassa olevalle käyttäjälle, jonka oikeudet eivät määrity pelkästään jonkin tietyn ryhmän kautta, vaan käyttäjälle on annettu yksittäisiä käyttöoikeuksia myös suoraan.

Uutta käyttäjää luodessa on myös asiakaspalautteen perusteella vaikea hahmottaa, mitä kaikkia oikeuksia tietynlaiseen käyttöön tulee antaa ja mistä ne annetaan: Tietyn sovelluksen käyttö saattaa vaatia käyttöoikeuksia useista eri paikoista. Esimerkiksi jos uudella käyttäjällä on tarve käyttää ostolaskusovellusta, tunnuksen perustamisen lisäksi täytyy käyttäjälle antaa oikeuksia ostolaskusovellukseen, erikseen oikeudet tiettyyn yritykseen ja toimittajiin, sekä myös oikeudet tiettyyn sivustoon ja sen työpöytään.

Yhteenvetona suurimmat ongelmat, joihin uudella hallintakäyttöliittymällä lähdetään hakemaan ratkaisua ovat siis seuraavat:

- Käyttöoikeudet ovat hankalia ylläpitää, raportoida ja valvoa kun ei ole olemassa yhtä paikkaa, josta näkisi kaikki yrityksen roolit, käyttäjät ja käyttöoikeudet
- Käyttöoikeuksia on hankala kopioida käyttäjältä toiselle
- Pelkästään käyttäjäryhmien ja roolien käyttö oikeuksien antamiseen on vaikeaa, joten joudutaan turvautumaan myös henkilökohtaisiin käyttöoikeuksiin, joka vaikeuttaa ylläpitoa entisestään

- Nykyinen yritystason käyttäjäroolien hallintakäyttöliittymä on ominaisuuksiltaan puutteellinen ja asiakaspalautteen mukaan monimutkainen käyttää

2.4 Ongelmien ratkaisumahdollisuudet

Keskeisenä tavoitteena uudelle yritystason käyttäjien ja roolien hallintakäyttöliittymälle on helpottaa käyttöoikeuksien ylläpitoa, sekä tehdä huomattavasti helpommaksi nähdä millaisia oikeuksia kullakin käyttäjällä ja roolilla on yritykseen ja sen sovelluksiin, mihin kaikkiin rooleihin käyttäjät kuuluvat ja mitä suoria henkilökohtaisia oikeuksia heillä mahdollisesti on.

Yksi keskeisin tavoite on myös ohjata ylläpitäjiä käyttämään pääsääntöisesti roolipohjaisia käyttöoikeuksia luotaessa yrityksiin uusia käyttäjiä. Tällöin käyttöoikeudet ovat asiakkaan näkökulmasta helpompia ylläpitää, mutta toisaalta myös toimeksiantajan on helpompi tunnistaa tietyntyyppiset käyttäjät esimerkiksi laskutusta ja muuta valvontaa varten. Uusi käyttöliittymä tekisi myös käyttöoikeuksien kopioimisesta käyttäjältä toiselle huomattavasti helpompaa, kun käyttäjien kaikki käyttöoikeudet ja niiden lähteet olisivat siinä näkyvissä.

Nykyisessä käyttöliittymässä ei ole näkyvissä kuin ne yrityksen käyttäjät, jotka kuuluvat joihinkin yrityksen tai sen sovellusten oletusrooleihin sekä nämä roolit ja niiden oikeudet, eikä esimerkiksi yrityksessä käytössä olevia kustomoituja instanssitason käyttäjäryhmiä tai käyttäjille suoraan annettuja käyttöoikeuksia. Nämä pitäisikin saada uuteen käyttöliittymään jotenkin näkyviin.

Ensin lähdettiin pohtimaan, olisiko käyttäjäryhmätkin mahdollista konvertoida rooleiksi, ja siirtyä käyttöoikeuksien hallinnassa kokonaan roolipohjaiseen pääsynhallinnan malliin (engl. RBAC, role-based access control) jossa käyttäjien käyttöoikeudet määrittyvät ainoastaan roolien kautta, joihin he kuuluvat. Nykyinen malli pohjautuu suurelta osin myös pääsylistoihin (ACL, access control list) joiden kautta käyttäjille voidaan antaa myös henkilökohtaisia käyttöoikeuksia.

Melko nopeasti kävi kuitenkin selväksi, ettei tämä ole mahdollista vaan tässä vaiheessa on tyydyttävä välimalliin, jossa hajanaisiakin käyttöoikeuksia on helpompi valvoa ja hallita, ja uusia käyttäjiä luodessa ohjataan käyttämään ensisijaisesti roolipohjaisia oikeuksia. Tämä johtuu siitä, että nykyisissä asiakasympäristöissä on niin paljon olemassa olevia käyttäjiä ja käyttäjäryhmiä, joiden oikeudet määräytyvät myös muilla tavoilla kuin roolien tai käyttäjäryhmien kautta. Erilaisia käyttöoikeushallinnan malleja käydään tarkemmin läpi seuraavassa luvussa.

Uudesta käyttöliittymästä pyritään tekemään myös käytettävyyden kannalta entistä yksinkertaisempi ymmärtää ja käyttää siitä huolimatta, että siihen tuodaan entistä enemmän tietoja ja ominaisuuksia. Pyritään myös varmistamaan, että käyttöliittymä on käytettävissä kaikilla yleisimmillä laitteilla ja selaimilla, ja skaalautuu siten tarvittaessa myös mobiilikäyttöä ajatellen.

Yhteenveto tavoista, joilla käyttäjähallinnan ongelmia ratkaistaan uudessa käyttöliittymässä:

- Uuteen käyttöliittymään tuodaan näkyviin ja muokattavaksi yritystason käyttäjäroolien ja oikeuksien lisäksi myös instanssitason käyttäjäryhmät, joilla on oikeuksia yritykseen ja niiden oikeudet, sekä myös käyttäjien henkilökohtaiset oikeudet ja sellaisetkin käyttäjät, joilla on yritykseen tai sen sovelluksiin oikeuksia vain niiden kautta; helpotetaan oikeuksien raportointia, ylläpitoa ja kopiontia.
- Ohjataan ylläpitäjiä käyttämään ensisijaisesti rooleja annettaessa käyttäjille käyttöoikeuksia ja tehdään rooleista entistä yksinkertaisempia ylläpitää, tavoitteena että käyttäjät kuuluisivat aina johonkin rooliin eikä käyttöoikeuksia tarvitsisi antaa käyttäjille suoraan erikseen.
- Tehdään uudesta käyttöliittymästä entistä yksinkertaisempi ymmärtää ja käyttää vaikka ominaisuudet lisääntyvät.
- Varmistetaan käyttöliittymän käytettävyys myös mobiililaitteilla, vaikka asetusnäkymiä käytetäänkin niillä harvemmin.

3 PÄÄSYNHALLINTA

Tässä luvussa käydään läpi pääsynhallintaa ja roolipohjainen sekä pääsylistapohjainen pääsynhallinnan malli, joihin MaestroNG-järjestelmänkin käyttöoikeuksien hallinta pitkälti perustuu. Käydään läpi myös toimivan pääsynhallinnan ratkaisun merkitystä organisaatioille muun muassa tietoturvan ja tietosuojan (riskienhallinnan) näkökulmasta.

3.1 Pääsynhallinnan käsite

Pääsynhallinnalla tarkoitetaan tietotekniikassa mekanismeja, joilla määritetään tavat ja rajoitteet, joiden perusteella tietyt järjestelmän käyttäjät pääsevät käsiksi tiettyihin järjestelmän resursseihin, lyhyesti voisikin sanoa, että sillä määritellään järjestelmässä "kuka voi tehdä mitäkin". Pääsynhallinta on yksi tärkeimmistä ja perustavanlaatuisimmista turvamekanismeista, joita tämän päivän tietojärjestelmissä käytetään. Se on käytössä oikeastaan kaikissa monen käyttäjän järjestelmissä ja onnistuneen pääsynhallinnan suunnittelu on todella tärkeä mutta samalla myös erittäin haastava osa ohjelmistokehitystä. (Ferraiolo;Chandramouli;Ferraiolo;& Kuhn, 2007, ss. 1-2)

Liiketoiminnan kannalta hyvin toteutetulla pääsynhallinnalla voidaan tarjota yhtä aikaa optimaalinen ja turvallinen tapa resurssien jakoon, mutta taas toisaalta huonosti suunniteltu pääsynhallinta voi turhauttaa loppukäyttäjiä, aiheuttaa suuria ylläpidollisia kuluja tai aiheuttaa arvokkaiden tietojen tuhoutumisen tai joutumisen vääriin käsiin. Pääsynhallinnan määritelmää on tutkittu ja useita erilaisia pääsynhallinnan menetelmiä ja malleja kehitetty jo aina 1970-luvulta lähtien. (Ferraiolo;Chandramouli;Ferraiolo;& Kuhn, 2007, ss. 1-2)

Melkein mitä vain pääsynhallinnan mallia voidaan kuvailla käyttämällä käsitteiden *käyttäjä, subjekti, objekti, operaatio ja käyttöoikeus* välisiä suhteita. Termillä *käyttäjä* tarkoitetaan kohdejärjestelmää käyttäviä ihmisiä, jotka tunnistetaan jonkun autentikointimenetelmän avulla, usein esimerkiksi käyttäjänimen ja salasanan yhdistelmällä. Näin on myös MaestroNG-järjestelmässä. Käyttäjän dialogia järjestelmän kanssa kutsutaan *istunnoksi. Subjekti* on tietokoneohjelman prosessi, joka toimii käyttäjän puolesta eli suorittaa ohjelmassa käyttäjän pyytämän toiminnon eli *operaation.* Käyttäjällä voi olla monia subjekteja yhdessä operaatiossa, vaikka käyttäjällä on vain yksi kirjautuminen ja istunto. Esimerkiksi sähköpostijärjestelmä voi toimia taustalla hakien sähköposteja palvelimelta tietyin väliajoin samalla kun käyttäjä käyttää esimerkiksi verkkoselainta. Kaikki käyttäjän yhtä aikaa järjestelmässä ajamat ohjelmat ovat subjekteja, ja jokaisen ohjelman toimintojen käyttöoikeudet tarkistetaan erikseen, jotta voidaan olla varmoja, että ohjelman käyttäjällä on oikeus ko. toimintoon. *objekti* on mikä vain tietokonejärjestelmän resurssi, esimerkiksi tiedosto tai tietokannassa oleva tieto, joka voi sisältää tai vastaanottaa tietoa. *käyttöoikeuksilla* määritellään käyttäjän oikeus objekteihin ja niille suoritettaviin operaatioihin. Esimerkiksi tiedoston kirjoitus/lukuoikeudet. (Ferraiolo;Chandramouli;Ferraiolo;& Kuhn, 2007, ss. 4-5)

3.2 Pääsynhallinnan malleja

Tässä osiossa tutustutaan lyhyesti rooli-, ja pääsylistapohjaisiin pääsynhallinnan malleihin, joiden yhdistelmästä MaestroNG-järjestelmän pääsynhallintamalli muodostuu.

Kuten aiemmissa luvuissa jo sivuttiin, yksi keskeisimmistä tavoitteista uudelle yritystason käyttäjien ja roolien hallintakäyttöliittymälle on kehittää järjestelmän käyttöoikeuksien hallintaa enemmän roolipohjaiseen suuntaan, ja uusien käyttäjien osalta päästä siihen, että kaikki käyttäjän oikeudet määräytyisivät roolien kautta eikä käyttäjille olisi tarvetta antaa suoria käyttöoikeuksia erikseen.

3.2.1 Roolipohjainen pääsynhallinta

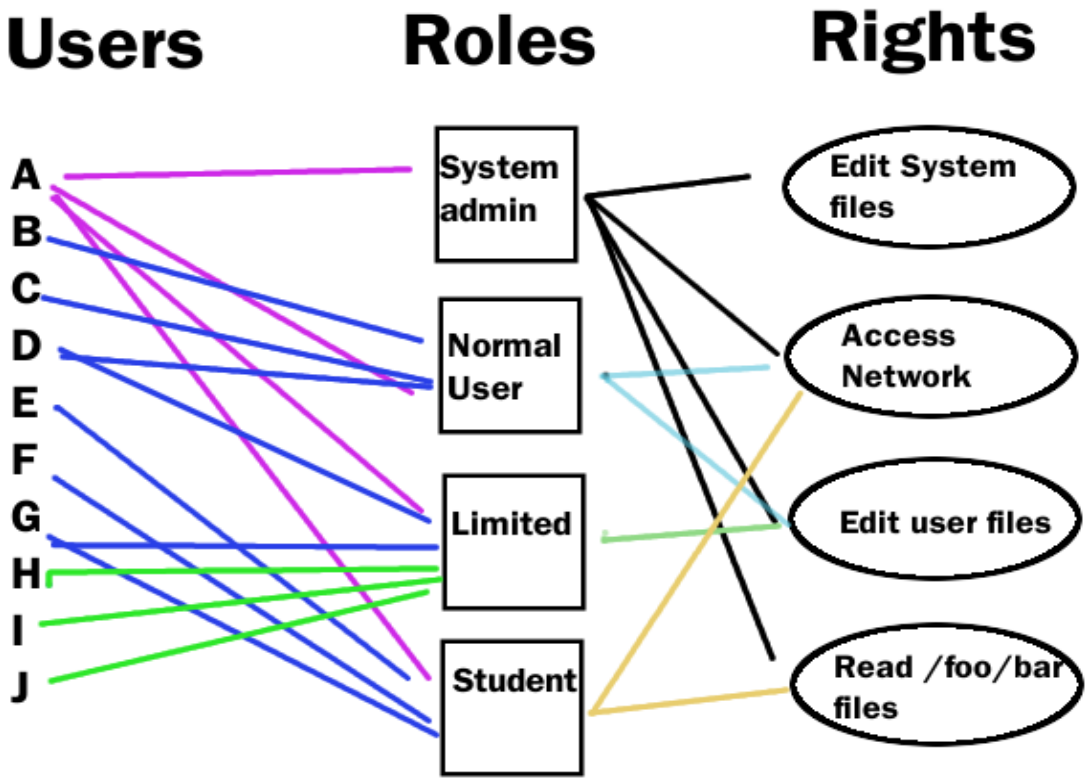
Roolipohjainen pääsynhallinta (engl. role-based access control, RBAC) on pääsynhallinnan malli, jossa käyttäjien käyttöoikeudet määrittyvät sen mukaan, mikä heidän roolinsa on järjestelmässä tai organisaatiossa. RBAC-malli tarjoaa laajat mahdollisuudet käyttöoikeuksien hallintaan pitäen sen kuitenkin helposti ylläpidettävänä, turvallisempana ja huomattavasti vähemmän virhealttiina kuin mallit, joissa käyttöoikeudet annetaan käyttäjille henkilökohtaisesti. (Auth0, Inc, 2013)

Roolipohjaisessa pääsynhallinnassa rooli on käytännössä kokoelma tietyn käyttötarpeen mukaisia käyttöoikeuksia, joka voidaan ottaa käyttöön useille käyttäjille samanaikaisesti. Roolien käyttö tekee käyttöoikeuksien antamisesta, poistamisesta ja muuttamisesta helpompaa kuin käyttöoikeuksien määrityksestä suoraan käyttäjälle. Kun käyttäjäkanta suurenee ja muuttuu monimuotoisemmaksi, rooleista tulee koko ajan hyödyllisempiä.
(Auth0, Inc, 2013)

Kun RBAC-mallia ollaan ottamassa käyttöön, tulee tutkia mitä käyttötarpeita organisaation eri käyttäjillä on, ja jakaa heidät rooleihin heidän yhteisten vastuidensa ja tarpeidensa mukaan. Sen jälkeen jokainen käyttäjä voidaan lisätä yhteen tai useampaan rooliin ja jokaiselle roolille voidaan määrittää yksi tai useampia käyttöoikeuksia. Käyttäjä-rooli ja rooli-käyttöoikeussuhteet tekevät käyttöoikeuksien hallinnasta helpompaa, kun käyttäjiä ja heidän käyttöoikeuksiaan ei tarvitse enää hallita erikseen, vaan heidät voidaan lisätä rooleihin, joiden kautta he saavat työtehtäviensä mukaiset käyttöoikeudet. RBAC-mallissa tärkeää on, että jokaisella roolilla on ainoastaan ne välttämättömät käyttöoikeudet, joiden avulla kyseisen roolin mukaisen työtehtävän suorittaminen onnistuu. (Auth0, Inc, 2013)

Kun RBAC-malli suunnitellaan huolella ja otetaan käyttöön oikealla tavalla, on siitä monia etuja. Sen avulla saavutetaan systemaattinen, nopea ja samantyyppisten käyttäjien kohdalla toistettavissa oleva käyttöoikeuksien määrittely ja se tekee käyttäjien oikeuksien valvonnasta ja sitä kautta liiallisten tai liian vähäisten käyttöoikeuksien tunnistamisesta ja näiden korjaamisesta huomattavista helpompaa, vähentää käyttöoikeuksien määrittelyssä tapahtuvia virheitä ja auttaa näin tietoturvan ylläpidossa. (Auth0, Inc, 2013)

Kuviossa 3 on kuvattu yksinkertainen esimerkki käyttöoikeuksien määrittymisestä roolipohjaisessa käyttöoikeuksien hallinnassa.



KUVIO 3. Käyttöoikeuksien määräytyminen RBAC-mallissa (Massachi, ei pvm)

3.2.2 Pääsylistapohjainen pääsynhallinta

Pääsylista (engl. access control list, ACL) on lista käyttöoikeuksista, joita tietyllä käyttäjällä tai ryhmällä on järjestelmän objektiin ja sille suoritettaviin operaatioihin. Jokainen rajoitettava järjestelmän objekti on yhdistetty pääsylistaan, josta löytyvät kaikki ne käyttäjät, joilla on pääsy kyseiseen objektiin. Tyypillinen objekti voi olla esimerkiksi jokin järjestelmän tiedosto ja sen käyttöoikeuksia esimerkiksi tiedoston luku/kirjoitusoikeudet. (Imperva, ei pvm)

3.2.3 Mallien vertailu SaaS-sovelluksen pääsynhallinnassa

Oikeastaan kaikilla SaaS-toimitusmallin sovelluksilla on oltava jonkunlainen pääsynhallinta osana tuotetta. Jopa kaikista yksinkertaisimmissa monen käyttäjän järjestelmissä on aina jotakin toimintoja, joihin kaikilla käyttäjillä ei ole syytä olla pääsyä. Esimerkiksi käyttäjätunnusten, laskutustietojen tai mahdollisten järjestelmän asetusten hallinta. (Savytska, 2020)

Vertailtaessa RBAC-mallia ja pääsylistoja keskenään voidaan todeta, että pääsylistat tarjoavat mahdollisuuden määritellä eri käyttäjien käyttäjäoikeudet tarkemmin, kun yksittäinen

käyttöoikeus eli pääsy tiettyyn objektiin tai toimintoon voidaan määritellä jokaiselle käyttäjälle erikseen. Tämä tarjoaa roolipohjaista mallia enemmän joustavuutta ja vapautta mutta tekee käyttöoikeuksien ylläpidosta ja valvonnasta huomattavasti vaikeampaa ja altistaa näin myös tietoturvariskeille. (Savytska, 2020)

RBAC-mallilla taas pystytään yleisesti ottaen kuvaamaan esimerkiksi useimpien yritysten organisaatiorakennetta ja sitä kautta sen järjestelmien käyttäjärakennetta pelkkiä pääsylistoja paremmin, ja kun tiettyjen työtehtävien vaatimat käyttöoikeudet tiettyihin resursseihin ryhmitellään rooleihin kategorioittain ei yksittäisiä oikeuksia enää tarvitse antaa käyttäjille erikseen. Tämä tekee käyttöoikeuksien ylläpidosta ja päivittämisestä huomattavasti nopeampaa ja helpompaa eli yleisesti tehokkaampaa sekä myös turvallisempaa tietoturvan kannalta, kun käyttöoikeudet on helpompi rajata ja niitä on helpompi valvoa. (Savytska, 2020)

3.3 Toimivan pääsynhallinnan merkitys organisaatiossa

Yksinkertaistettuna pääsynhallinta on mekanismi, jonka avulla rajoitetaan pääsyä tiettyyn dataan ylläpitäjien harkinnan mukaisesti; tunnistetaan käyttäjän olevan väitetty henkilö ja varmistetaan että heillä on pääsy vain siihen organisaation tietoon, jota he tarvitsevat. *Todennus* on tekniikka, jolla varmistetaan käyttäjän olevan se henkilö, joka hän väittää olevansa. Todennus ei ole itsessään riittävä suojelemaan arkaluontoista dataa. Tarvitaan myös toinen kerros, *valtuutus*, joka määrittää pitäisikö todennetun käyttäjän päästä käsiksi hänen pyytämäänsä resurssiin tai toimintoon. On sanottu, että ilman todennusta ja valtuutusta ei ole olemassa tietoturvaa: Suurimmassa osassa tietomurtoja yksi hyökkäyksen avainkomponenteista on puutteellinen pääsynhallinta. Kun pääsynhallintaa ei olla toteutettu kunnolla, voivat seuraukset organisaatiolle olla tuhoisat. On sanottu myös, että jokainen organisaatio, jonka työntekijät ovat yhteydessä internetiin ja jonka järjestelmissä olevilla tiedoilla voisi olla käyttöä sellaiselle taholle, jolla ei pitäisi kyseisiin tietoihin olla pääsyä tarvitsee vahvan pääsynhallinnan. (Martin, 2019)

Pääsynhallinnasta on tullut yhä keskeisempi osa organisaatioiden riskienhallintaa myös tiukentuneen kansallisen ja kansainvälisen tietoon liittyvän säätelyn myötä. Esimerkiksi SOX-säännöt, EU:n tietosuoja-asetus GDPR, luottokorttien suojaukseen liittyvä säätely, sosiaali- ja terveyshuollon asiakastietoja koskevat lait sekä muut julkisen puolen ohjeet vaikuttavat siihen, miten organisaatioiden on suojattava tietojaan pääsynhallinnan avulla. (Tanskanen, 2016)

4 TYÖSSÄ KÄYTETYT TEKNIIKAT

Tässä luvussa esitellään lyhyesti uuden hallintakäyttöliittymän toteutuksessa käytetyt keskeisimmät teknologiat: Frontend-kehitykseen käytetty JavaScript-ohjelmistokehys AngularJS ja backendin osalta Microsoftin web-kehitykseen tarkoitettu .NET-ohjelmistokehyksen osa ASP.NET, REST-rajapinnat ja tietokantajärjestelmä SQL Server. Näillä tekniikoilla toteutettiin sovelluksen varsinainen toiminnallisuus. Sovelluksen ulkoasu ja responsiivisuus (käyttöliittymän skaalautuminen eri kokoisille päätelaitteiden näytöille) on toteutettu käyttäen HTML-merkintäkieltä ja responsiiviseen web-kehitykseen tarkoitettua Bootstrap-kirjastoa sekä Maestron omia CSS-määrityksiä eri komponenteille.

Kehitykseen käytettiin ohjelmoinnin ja debuggauksen osalta Microsoftin Visual Studio -kehitysympäristöä, ja tietokantakyselyiden kehitykseen SQL Server Management Studiota (SSMS). Versionhallintana projektissa toimi Azure DevOps.

4.1 AngularJS

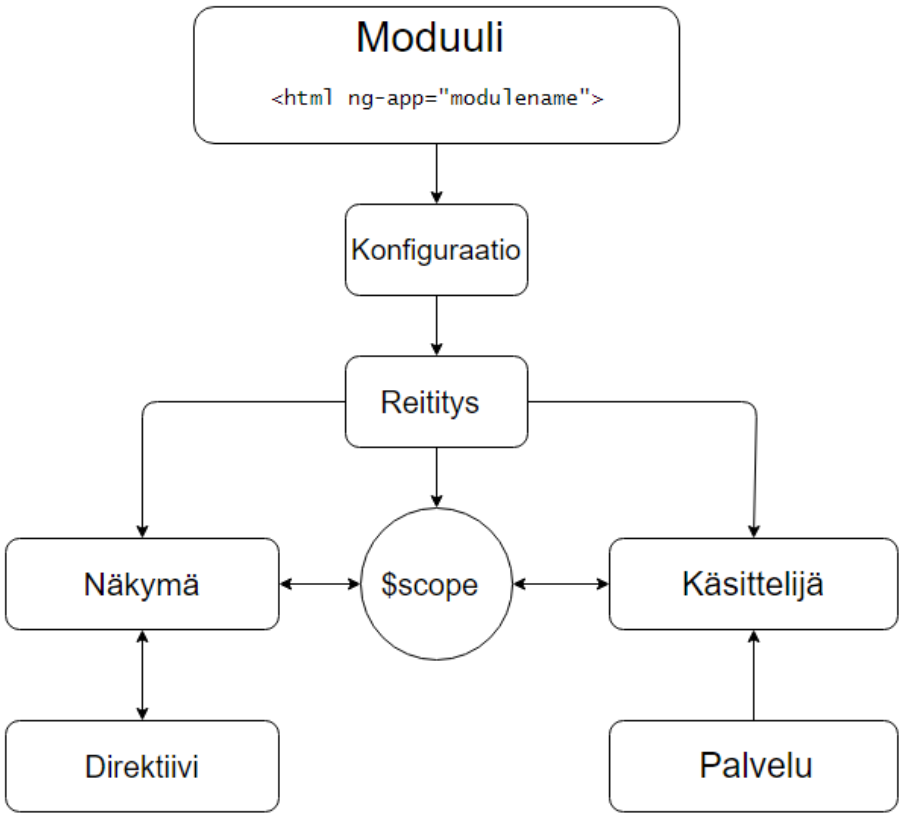
AngularJS on Miško Heveryn ja Adam Abronsin vuonna 2009 kehittämä avoimen lähdekoodin asiakaspuolen (engl. client-side) JavaScript-ohjelmistokehys dynaamisten verkkosovellusten kehittämiseen, jonka Google otti laajasti käyttöön omissa projekteissaan sekä ylläpidettäväkseen vuonna 2010 Heveryn työskennellessä Googlella. (Branas, 2014, s. 26)

AngularJS:n idea on laajentaa perinteisen HTML:n syntaksia *direktiiveillä* siten, että siihen pystytään lisäämään dynaamista toiminnallisuutta kuten esimerkiksi kaksisuuntaista datan sitomista käsittelijän ja näkymän välillä, elementtien piilottamista ja näyttämistä ehdon perusteella tai lomakkeiden validointia lennossa helposti. Normaaleihin HTML-elementteihin voidaan myös lisätä AngularJS:n avulla helposti uutta toiminnallisuutta kuten tapahtumien käsittelyä ja HTML:ää ryhmitellä uudelleenkäytettäviksi kustomoiduiksi komponenteiksi. (Google, ei pvm)

Korkealla tasolla direktiivi on HTML:n DOM-elementissä, yleensä sen attribuutissa tai nimessä oleva merkki, joka kertoo AngularJS:n HTML-kääntäjälle, että kyseiseen elementtiin pitää kiinnittää tietty toiminnallisuus, tai esimerkiksi muuttaa elementtiä tai sen lapsielementtejä tietyllä tavalla. AngularJS sisältää yleisiä käyttötapauksia varten useita valmiita direktiivejä, mutta niitä voidaan kirjoittaa myös itse. (Google, ei pvm)

AngularJS-sovellukset seuraavat hyvin pitkälti malli-näkymä-käsittelijä -arkkitehtuurimallia (engl. Model-View-Controller, MVC), jossa sovelluksen logiikka on jaettu niin, että näkymä on vastuussa tiedon tuomisesta näkyville, tietoa säilytetään mallissa ja käsittelijässä käsitellään tätä tietoa sekä välitetään se näkymästä malliin ja toisinpäin.

AngularJS-sovelluksissa on kuitenkin myös muitakin tärkeitä komponentteja mallin, näkymän ja käsittelijän lisäksi, kuten esimerkiksi palvelut (engl. services), direktiivit (engl. directives) ja suotimet (engl. filters). Näkymä kirjoitetaan puhtaasti HTML-kielellä, jonka toiminnallisuutta laajennetaan AngularJS:n direktiivien avulla. Näkymän takana on käsittelijä, joka sisältää näkymän käyttämän varsinaisen sovelluslogiikan. Kun sovellukset kasvavat isommiksi, on kuitenkin ylläpidollisesti tärkeää hajauttaa osa logiikasta käsittelijästä esimerkiksi palveluihin. Yhteys näkymän ja käsittelijän välillä toteutetaan AngularJS-sovelluksissa *scopen* avulla. Scope on rakenteessa näkymän ja käsittelijän välissä ja sitä käytetään malliin liittyvän tiedon siirtoon. Malli on tavallinen JavaScript-olio (engl. Plain-Old-JavaScript-Object, POJO), tämän hyötynä on se, että tavallisen js-olion käyttö mallina tekee kehityksestä helpompaa, kun mallien esittelyyn ei tarvitse opetella mitään erikoissyntaksia mikäli osaa jo JavaScriptiä. (Branas, 2014, s. 27) Kuviossa 4 ja taulukossa 1 on kuvattu AngularJS-sovelluksen rakenne ja tärkeimmät konseptit.



KUVIO 4. AngularJS-sovelluksen malli isossa kuvassa (Ruebbelke, 2015, s. 7), mukaillen

Konsepti	Tehtävä
Moduuli (module)	Moduuli on sovelluksen 'juuri' joka sisältää kaikki sovelluksen eri osat, kuten esimerkiksi käsittelijät, palvelut ja direktiivit.
Konfiguraatio (config)	Sovelluksen config-blokkeihin voidaan lisätä konfiguraatiota, joka tulee voimaan ennen kuin sovellus käynnistyy, esimerkiksi reititystä ja palveluiden konfiguraatiota.
Reitit (routes)	Reiteillä voidaan määritellä tapoja sovelluksen eri tilojen välillä navigointiin, eri tiloille voidaan konfiguroida myös esimerkiksi tietty käsittelijä ja näkymäpohja.
Näkymä (view)	Näkymä on se, mitä AngularJS renderöi käyttäjän näkyville pohjina käytetystä HTML:stä ja siihen lisätyistä direktiiveistä ja ilmauksista.
$scope	$scope on periaatteessa "liimaa" näkymän ja käsittelijän välillä, ja mallin data siirtyy näkymän ja käsittelijän välillä sen kautta. Malli on tallessa $scopessa, josta siihen pääsevät käsiksi käsittelijät, direktiivit ja ilmaukset.
Malli (model)	Mallissa on se data, mikä näytetään käyttäjälle näkymässä ja jonka kanssa käyttäjä voi vuorovaikuttaa.
Käsittelijä (controller)	Käsittelijä tarjoaa näkymälle varsinaisen sovelluslogiikan. Käsittelijöiden tulisi olla mahdollisimman yksinkertaisia ja koskettaa vain sitä näkymää, jota ne käsittelevät.
Direktiivi (directive)	Direktiivien avulla voidaan laajentaa HTML:n toiminnallisuutta kustomoiduilla attribuuteilla ja elementeillä.
Palvelu (service)	Palveluiden avulla voidaan tarjota uudelleenkäytettävää tietyistä näkymistä riippumatonta sovelluslogiikkaa AngularJS-sovellukseen.
Ilmaus (expression)	Ilmausten avulla voidaan käyttää $scopen muuttujia ja funktioita HTML:stä käsin, ilmaukset kirjoitetaan aaltosulkeiden sisään {{}}.
Suodin (filter)	Suotimien avulla voidaan ilmauksen arvo muokata tiettyyn muotoon ennen käyttäjälle näyttämistä. Suotimet kirjoitetaan ilmausten jälkeen putkimerkillä | erotettuna. Esimerkiksi suodin uppercase muuntaa tekstimuotoisen ilmauksen arvon isoilla kirjaimilla kirjoitetuksi.
Datansidonta (data binding)	Datansidonta on yksi AngularJS:n tärkeimmistä ominaisuuksista. Sen avulla mallin ja näkymän data pidetään reaaliaikaisesti synkronoituina keskenään.

TAULUKKO 1. AngularJS:n tärkeimpiä konsepteja (Google, ei pvm), (Ruebbelke, 2015, s. 6), mukaillen

Kuvissa 2-3 on kuvattu hyvin yksinkertainen esimerkkisovellus, jossa käytetään sisäänrakennettuja `ng-model` ja `ng-if` -direktiivejä. Sovelluksessa on tekstikenttä johon käyttäjä voi kirjoittaa nimensä, ja alla näytetään reaaliaikaisesti nimikentän mukaisesti päivittyvä tervehdys. `ng-model`-direktiivillä tekstikentän teksti saadaan sidottua käsittelijässä `$scope`:n jäsenmuuttujaksi esiteltyyn name-muuttujaan kaksisuuntaisesti. Aaltosulkusyntaksilla `{{name}}` saadaan muuttujan arvo tulostumaan näkymään myös `<span>`-elementin sisään, ja näkymä päivittyy näin reaaliaikaisesti sen mukaan, kun tekstikentän teksti ja sitä kautta myös `name`-muuttujan arvo muuttuu. Tätä kutsutaan nimellä ilmaus (expression). `ng-if`-direktiivillä taas on määritelty, ettei tervehdyksen sisältävää `<span>`-elementtiä renderöidä näkymään ollenkaan, jos `name`-muuttujalla ei ole arvoa. AngularJS-sovellus alustetaan kutsumalla `module`-funktiota, jonka jälkeen alustettuun moduuliin voidaan lisätä riippuvuuksia (dependencies) kuten tässä tapauksessa käsittelijä. Näkymässä moduulin (sovelluksen) ja käsittelijän vaikutusalueet määritellään lisäämällä haluttuihin HTML-tageihin `ng-app` ja `ng-controller` -direktiivit.

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Example</title>
  <script src="//code.angularjs.org/snapshot/angular.min.js"></script>
  <script src="script.js"></script>
</head>
<body ng-app="Example">
  <div ng-controller="Controller">
  Kirjoita nimesi <input ng-model='name'> <hr/>
  <span ng-if="name">Moikka {{name}}!</span>
</div>
</body>
</html>
```

Preview

Kirjoita nimesi Samu

Moikka Samu!

KUVA 2. AngularJS-esimerkkisovellus ja sen index.html-näkymätiedosto

script.js

```
(function(angular) {
  'use strict';
angular.module('Example', [])
  .controller('Controller', ['$scope', function($scope) {
    $scope.name = '';
  }]);
})(window.angular);
```

KUVA 3. AngularJS-esimerkkisovelluksen script.js-tiedosto, jossa alustetaan sovellusmoduuli Example, käsittelijä Controller ja käsittelijän $scope sekä sen name-muuttuja johon nimikentän arvo sidotaan

Angularista on olemassa myös uudempi, TypeScript-ohjelmointikieltä hyödyntävä versio, jota kutsutaan pelkästään nimellä Angular (ilman JavaScriptiin viittaavaa JS-päätettä). Uuden käyttäjähallinnan käyttöliittymän toteutukseen valittiin AngularJS lähinnä siksi että suurin osa MaestroNG-järjestelmän HTML5 osioista mukaan lukien koko yritysten ylläpitosovellus ja nyt uudistettava toteutus käyttäjien ja roolien hallintakäyttöliittymästä on toteutettu AngularJS:llä. Uusikin toteutus on siis helpompi toteuttaa ja istuttaa osaksi yritysten ylläpitosovellusta, kun toteutukseen käytetään AngularJS:ää. Lisäksi käytäntö on osoittanut AngularJS:n olevan toimiva tekniikka web-kehitykseen vielä tänäkin päivänä, ja se on myös yhä laajalti käytössä.

4.2 .NET Framework ja ASP.NET

Uuden käyttäjien ja roolien hallintakäyttöliittymän toteutuksessa käytettiin backendin (palvelinpuolen) osalta .NET-alustaan kuuluvaa ASP.NET -ohjelmistokehystä ja C#-kieltä muun muassa tietokantayhteyksien ja REST-rajapintojen toteutukseen. Tässä osassa esitellään lyhyesti .NET-alusta, C#-ohjelmointikieli sekä ASP.NET-ohjelmistokehys, joka on tarkoitettu verkkosovellusten kehittämiseen Windows-alustalla.

4.2.1 .NET Framework ja C#-ohjelmointikieli

.NET Framework on Microsoftin kehittämä sovellusalusta ja ohjelmistokehys erityyppisten sovellusten kehittämiseen ja ajamiseen Windows-alustalla. Sen tärkeimmät komponentit ovat Common Language Runtime (CLR) ja .NET-luokkakirjasto, joka tarjoaa kehittäjille runsaasti valmista, uudelleenkäytettävää koodia yleisien toiminnallisuuksien helppoa implementointia varten ja matalamman tason ohjelmoinnin tarpeen vähentämiseksi. CLR on moottori, joka vastaa .NET-sovellusten ajamisesta ja tarjoaa niille palveluita kuten automaattisen muistinhallinnan. (Microsoft, 2019)

.NET-sovelluksia voidaan kehittää usealla eri ohjelmointikielellä, kuten C#, F# ja Visual Basic. .NET Framework sisältää myös useita ohjelmistokehyksiä tietyn tyyppisten sovellusten kehittämiseen, kuten ASP.NET verkkosovelluksia varten, ADO.NET datan käsittelyyn, WCF asynkronisten palveluiden ja WPF Windows-työpöytäsovellusten kehitykseen. (Microsoft, 2019)

C# on Microsoftin .NET-alustalle kehittämä moderni, oliopohjainen ja vahvasti tyypitetty ohjelmointikieli, jonka syntaksi perustuu pitkälti muihin C-sukuisiin ohjelmointikieliin, eli C#-kielellä ohjelmointi on helppo oppia aiemmin esimerkiksi C-, C++-, tai Java -kielillä ohjelmoineille. (Microsoft, 2020)

4.2.2 ASP.NET ja REST API

ASP.NET on .NET-alustan dynaamisten verkkosivustojen ja palveluiden luomiseen tarkoitettu avoimen lähdekoodin palvelinpuolen ohjelmistokehys, jonka ohjelmointikielenä toimii C#, F# tai Visual Basic. ASP.NET MVC:n avulla voidaan helposti luoda MVC-mallin mukaisia dynaamisia verkkosovelluksia, joissa palvelinpuolen logiikkaa voidaan kirjoittaa HTML-koodin sekaan C#-kielellä. C#-koodi suoritetaan palvelimella ja tuloksena oleva HTML lähetetään käyttäjälle. Tätä syntaksia kutsutaan nimellä Razor, ja C#-koodiblokit erotetaan HTML:n seasta @-merkillä. Kuvassa 4 on esimerkki Razor-syntaksin käytöstä. ASP.NET:in kirjastot sisältävät myös runsaasti valmista koodia tyypillisiin web-kehityksen perustehtäviin, jolloin kehittäjän ei tarvitse kirjoittaa niitä alusta alkaen itse. Esimerkiksi käyttäjien autentikointiin löytyy ASP.NET:stä valmis, turvallinen implementaatio. (Microsoft, ei pvm)

KUVA 4. Esimerkki Razor-syntaksin käytöstä, jossa palvelinpuolen C#-koodia voidaan kirjoittaa HTML:n sekaan (Microsoft, ei pvm)

ASP.NET:llä voidaan helposti tehdä myös esimerkiksi REST-rajapintoja sen Web API-kehyksellä, joiden avulla sovelluksen frontend ja backend voivat kommunikoida keskenään HTTP-protokollan kautta. (Microsoft, ei pvm) Opinnäytetyön osalta backend-kehitys keskittyikin suurimmilta osin REST-rajapintojen, tietokannan kanssa kommunikoivan palvelinpuolen C#-koodin sekä malliluokkien tekemiseen.

REST (Representational State Transfer) on Roy Fieldingin vuonna 2000 määrittelemä joukko suosituksia web-palveluille, jotka niiden tulee täyttää voidakseen kutsua itseään termillä RESTful: Palveluiden on oltava asiakas-palvelin periaatteella toimivia, eli asiakas tekee HTTP-pyynnön tiettyyn palvelimella olevaan URL-osoitteeseen jonka jälkeen se palauttaa vastauksen, yleensä esimerkiksi JSON- tai XML-muotoisena. REST-rajapinnan on oltava myös tilaton, eli

asiakaspyynnössä pitää olla kaikki tarpeellinen pyyntöön vastaamiseen: Pitäisi pystyä tekemään kaksi tai yli samanlaista HTTP-pyyntöä missä vain järjestyksessä ja saada sama vastaus. Samanlaisten vastauksien pitäisi myös muun muassa olla tallennettavissa välimuistiin. (Sitepoint, 2020)

REST-rajapintaan kohdistuva pyyntö sisältää sen operaation osoitteen (URL) jota halutaan kutsua, rajapinnoissa on usein määritelty useita eri osoitteita eri operaatioille. Esimerkkiosoite voisi olla esimerkiksi https://osoite/user/123 jonka avulla voitaisiin esimerkiksi hakea tai päivittää käyttäjä id:llä 123. Lisäksi pyyntö sisältää HTTP-verbin, jonka avulla pyynnön suoritus voidaan ohjata tiettyyn rajapinnan metodiin, vaikka itse osoite ja/tai pyynnön sisältö voi niissä useassa olla sama. (Sitepoint, 2020) Yleisimmät REST-rajapintojen luonti, luku, päivitys, poisto (CRUD, create, read, update, delete) -operaatioiden määrittelyssä käytetyistä HTTP-verbeistä on kuvattu taulukossa 2.

VERBI	**CRUD-operaatio**	**TOIMINTO**
GET	Luku	Palauttaa pyydetyn datan asiakkaalle
POST	Luonti	Luo uuden alkion
PUT	Päivitys	Päivittää olemassa olevaa alkiota
DELETE	Poisto	Poistaa olemassa olevan alkion

TAULUKKO 2. Yleisimmät REST-rajapinnoissa käytettävät HTTP-verbit (Sitepoint, 2020), mukaillen

Esimerkkinä yllä olevien verbien käytöstä voisivat olla seuraavat tilanteet: GET-pyyntö osoitteeseen /user/ palauttaa listan järjestelmän käyttäjistä, GET-pyyntö osoitteeseen /user/123 palauttaa käyttäjän 123 tiedot, POST-pyyntö osoitteeseen /user/ luo uuden käyttäjän pyynnön bodyssa lähetetyillä tiedoilla, PUT-pyyntö osoitteeseen /user/123 päivittää käyttäjää 123 pyynnön bodyssa lähetetyillä tiedoilla, ja DELETE-pyyntö osoitteeseen user/123 poistaa käyttäjän 123. Verbin ja kohdeosoitteen lisäksi HTTP-pyyntö sisältää myös rungon (body), jossa on usein pyynnön suorittamiseen liittyvää dataa, yleisimmin JSON-muodossa, sekä ylätunnisteen (header) joka voi sisältää esimerkiksi autentikaatioon liittyviä tietoja tai evästeitä.

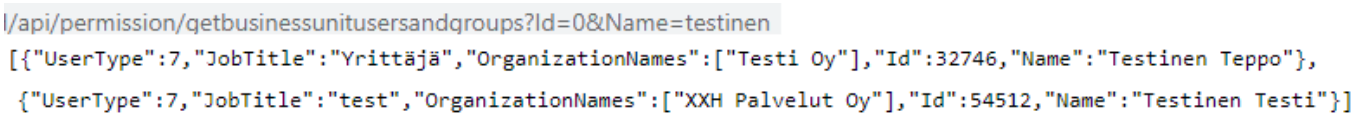
Rajapinnasta takaisin asiakkaalle tulevan vastauksen body voi sisältää mitä vain tilanteesta riippuen, perusdatan lisäksi esimerkiksi HTML:ää, ääntä tai kuvia. Yleisimmin vastauksena tuleva data on kuitenkin JSON-enkoodattua, mutta voi olla myös muussa vastaavassa muodossa. Tämän lisäksi vastauksen ylätunnisteeseen lisätään myös pyynnön onnistumisesta/epäonnistumisesta kertova HTTP-statuskoodi. Yleisimmin onnistuneisiin pyyntöihin vastataan koodilla 200 OK, mutta esimerkiksi uuden alkion luovan POST-pyynnön kohdalla monesti myös 201 Created. Virheistä taas palautetaan yleisimmin esimerkiksi 400 Bad Request, 404 Not Found tai 401 Unauthorized -koodi. (Sitepoint, 2020)

Kuvassa 5 on kuvattu REST-rajapintaimplementaatio MaestroNG:stä, joka hakee asiakasinstanssin käyttäjiä tai käyttäjäryhmiä maksimissaan 10 kerrallaan perustuen GET-pyynnössä tulevaan mahdolliseen hakuparametriin. Uudessa käyttäjien ja roolien hallintakäyttöliittymässä tätä rajapintaa kutsutaan elementistä, josta rooleihin lisättävä käyttäjä haetaan ja valitaan. RoutePrefix-parametrilla valitaan URL:n alkuosa, jolla kaikki käsittelijän rajapintojen osoitteet alkavat, ja Http(Get)- ja Route- parametreilla määritellään allaolevan suoritettavan metodin HTTP-verbi sekä tarkempi polku (URL:n loppuosa).

```
[RoutePrefix("api/permission")]
[Module(Module.Dashboard)]
public class PermissionController : ApiControllerBase
{
  [HttpGet]
  [Route("GetBusinessUnitUsersAndGroups")]
  public IEnumerable<UserAndGroupIdNameModel> GetUsersAndGroupsByCurrentUserBusinessUnitAccess([FromUri]string name = null)
  {
    int showCountLimit = 10;
    List<UserAndGroupIdNameModel> result = UserManager.GetUsersAndGroupsByCurrentUserBusinessUnitAccess(name, showCountLimit + 1);

    if (result == null || result.Count == 0)
    {
      UserAndGroupIdNameModel emptyResult = new UserAndGroupIdNameModel();
      emptyResult.Id = 0;
      emptyResult.Name = TranslationProvider.Translate("RecordNotFound");

      result = new List<UserAndGroupIdNameModel>();
      result.Add(emptyResult);
    }
    else if (result.Count == (showCountLimit + 1))
    {
      var lastItem = result.Last();
      lastItem.Id = 0;
      lastItem.Name = TranslationProvider.Translate("MoreSearchResult");
      lastItem.JobTitle = null;
      lastItem.OrganizationNames = null;
      lastItem.UserType = 0;
    }

    return result;
  }
```

KUVA 5. ASP.NETillä toteutettu REST-rajapinta, joka palauttaa asiakasinstanssin käyttäjiä/ryhmiä maksimissaan 10 kerrallaan hakuparametrin (käyttäjän nimen) perusteella.

Kuvassa 6 taas on kuvattu tuloksena palautuva JSON, kun edellä esiteltyyn rajapintaan lähetetään GET-pyyntö osoitteeseen:
/api/permission/getbusinessunitusersandgroups?Id=0&Name=testinen
Rajapinnasta palautuvat vastauksena kaikki asiakasinstanssin käyttäjät, joiden nimi sisältää merkkijonon ”testinen”.



KUVA 6. Käyttäjähakurajapinnasta palautuva vastaus-JSON.

4.3 Microsoft SQL Server

SQL Server on Microsoftin kehittämä ja markkinoima relaatiotietokantojen hallintaan ja kehittämiseen käytettävä järjestelmä (RDBMS, relational database management system). Kuten muutkin vastaavat RDBMS-järjestelmät, se on rakennettu SQL-kielen päälle, joka on standardisoitu ohjelmointikieli relaatiotietokantojen ja niiden sisältämän tiedon hallintaan ja hakemiseen. SQL Server käyttää Microsoftin laajennusta tavallisesta SQL-kielestä, jota kutsutaan nimellä Transact-SQL tai lyhyemmin T-SQL, joka lisää SQL kieleen joitakin proseduraaliseen ohjelmointiin liittyviä lisäominaisuuksia, jolloin se muistuttaa normaalia SQL:ää enemmän perinteistä ohjelmointikieltä, ja myös tietokannan päähän voidaan sijoittaa ohjelmointilogiikkaa kuten silmukoita tai ehtolauseita. (sqlservertutorial.net, ei pvm)

SQL Server koostuu kahdesta pääkomponentista, joita ovat tietokantamoottori ja SQLOS. SQL Serverin tietokantamoottori koostuu relaatiomoottorista (relational engine) ja varastomoottorista (storage engine). Relaatiomoottori prosessoi tietokantaan tulevat kyselyt ja pyytää niihin perustuen dataa ja sen prosessointia varastomoottorilta, joka vastaa tietokannassa olevan datan ja komponenttien kuten indeksien, näkymien ja proseduurien varastoinnista. SQLOS (SQL Server Operating System) taas on tietokantamoottoria alemmalle tasolle sijoittuva komponentti, joka tarjoaa käyttöjärjestelmälle ominaisia matalamman tason palveluita, kuten muistin ja poikkeuksien hallinnan. (sqlservertutorial.net, ei pvm)

Microsoft tarjoaa SQL Serverin rinnalle useita tiedonhallintaan ja analysointiin liittyviä oheispalveluita ja työkaluja, joista opinnäytetyön kannalta tärkein oli tietokantojen hallintaan ja monitorointiin tarkoitettu SQL Server Management Studio (SSMS), jota käytettiin uuteen hallintakäyttöliittymään liittyvien SQL-kyselyiden kehitykseen.

5 TYÖN TOTEUTUS

Tässä luvussa kuvataan opinnäytetyössä uudistetun MaestroNG:n yritystason käyttäjien ja roolien käyttöliittymän kehitysprosessi suunnitteluvaiheesta varsinaisen toteutusvaiheen kautta aina valmiin käyttöliittymän testaukseen ja tuotantoon vientiin asti.

Ensin käydään läpi suunnitteluvaihe, olemassa ollut toteutus ja sen käytettävyysongelmat sekä toiminnolliset puutteet ja miten niitä päätettiin lähteä uudessa käyttöliittymässä ratkaisemaan. Tämän jälkeen perehdytään näiden pohdintojen pohjalta uudesta käyttöliittymästä piirrettyihin UI-kuviin, joiden pohjalta varsinaista kehitystä lähdettiin tekemään. Seuraavaksi kuvataan toteutusvaihe sisältäen pääpiirteittäin kuvattuna uudistetun käyttöliittymän koko kehitysprosessin, jonka jälkeen esitellään valmis uudistettu käyttöliittymä ja sen toiminnot.

5.1 Suunnitteluvaihe

Tässä osassa käydään läpi uuden toteutuksen suunnitteluvaihe, sisältäen käydyt aloituspalaverit, olemassa ollut käyttöliittymä ongelmineen sekä uuden käyttöliittymän suunnitteluprosessi.

5.1.1 Aloituspalaverit

Kun oli päätetty, että opinnäytetyön aihe tulee käsittelemään nimenomaan MaestroNG:n käyttäjähallintaa ja yritystason käyttäjien ja roolien hallintakäyttöliittymän uudistusta, pidettiin muutamia aloituspalavereita, joissa olivat läsnä lisäkseni tuotepäällikkö ja pääsuunnittelija. Palavereissa käytiin yleisesti läpi MaestroNG:n käyttäjähallintamallia ja sen ongelmia, sekä mihin suuntaan käyttäjien ja käyttöoikeuksien hallintaa haluttaisiin viedä jatkossa. Olemassa ollutta käyttäjähallinnan mallia ja sen ongelmia sekä niiden potentiaalisia ratkaisuja on käyty tarkemmin läpi luvussa 2.

Tiivistettynä havaittiin, että nykyisessä mallissa on suurimpana ongelmana se, että käyttöoikeuksia voi antaa ja joudutaankin antamaan usein yritystason roolien ja instanssitason käyttäjäryhmien lisäksi myös suoraan yksittäisille käyttäjille, joka tekee käyttöoikeuksien valvomisesta, ylläpidosta ja kopioinnista käyttäjältä toiselle hankalaa varsinkin siksi, että järjestelmässä ei ole olemassa keskitettyä paikkaa josta näkisi helposti kaikki käyttäjät, roolit sekä ryhmät ja niiden oikeudet tiettyyn instanssin yritykseen.

Olemassa olevasta yritystason käyttäjien ja roolien hallintakäyttöliittymästä näkyivät vain ne käyttäjät, joilla oli oikeuksia kyseiseen yritykseen yrityksen tai sen sovellusten oletusroolien kautta ja nämä roolit, mutta ei ollenkaan esimerkiksi instanssitason käyttäjäryhmiä ja niiden

jäseniä, joilla on oikeuksia yritykseen, eikä myöskään niitä käyttäjiä, joilla on suoria henkilökohtaisia käyttöoikeuksia yritykseen. Pelkkien oletusroolien käyttö ei riitä läheskään kaikkiin käyttötapauksiin, eivätkä kaikki asiakkaat käytä niitä ollenkaan. Asiakaspalautteen mukaan olemassa oleva käyttöliittymä oli myös käytettävyydeltään hankala. Olemassa ollut käyttöliittymä ja sen ongelmat käydään tarkemmin läpi seuraavassa osiossa.

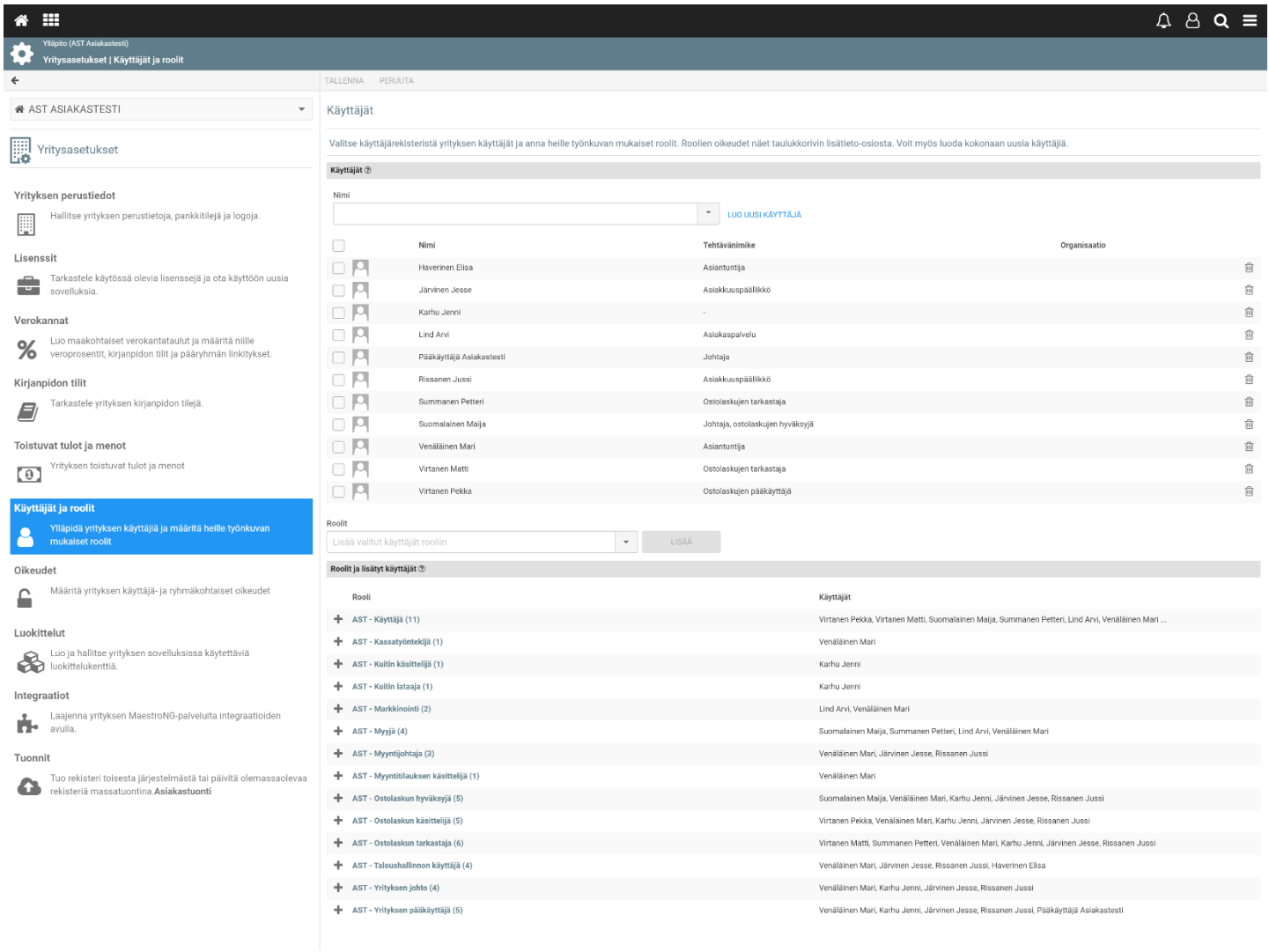
Käyttäjät kuuluvat aina yhteen tai useampaan instanssin yritykseen ja suurin osa käytöstä tapahtuu yritystasolla, joten yritystason hallintakäyttöliittymä oli looginen paikka lähteä ratkaisemaan näitä ongelmia.

Päätettiin, että yritystason käyttöliittymään tuodaan näkyviin ja muokattavaksi yritystason oletusroolien ja niiden käyttäjien lisäksi myös kustomoitavat instanssitason käyttäjäryhmät, joilla on oikeuksia ko. yritykseen, sekä ne käyttäjät, jotka omaavat siihen suoria oikeuksia ja näiden oikeudet helpottaen oikeuksien valvontaa, kopiointia ja ylläpitoa. Tällä ja käyttöliittymäsuunnittelulla yritettäisiin vaikuttaa myös siihen, että järjestelmän käyttöoikeuksien hallinta siirtyisi suurelta osin roolipohjaiseen suuntaan eikä varsinkaan uusille käyttäjille tarvitsisi enää antaa ollenkaan suoria käyttöoikeuksia, vaan kaikki oikeudet tulisivat roolien kautta. Samalla pyrittäisiin myös tekemään käyttöliittymästä myös huomattavasti entistä helpompi ymmärtää ja käyttää vaikka ominaisuudet lisääntyvät.

5.1.2 Olemassa oleva käyttöliittymä ja sen ongelmat

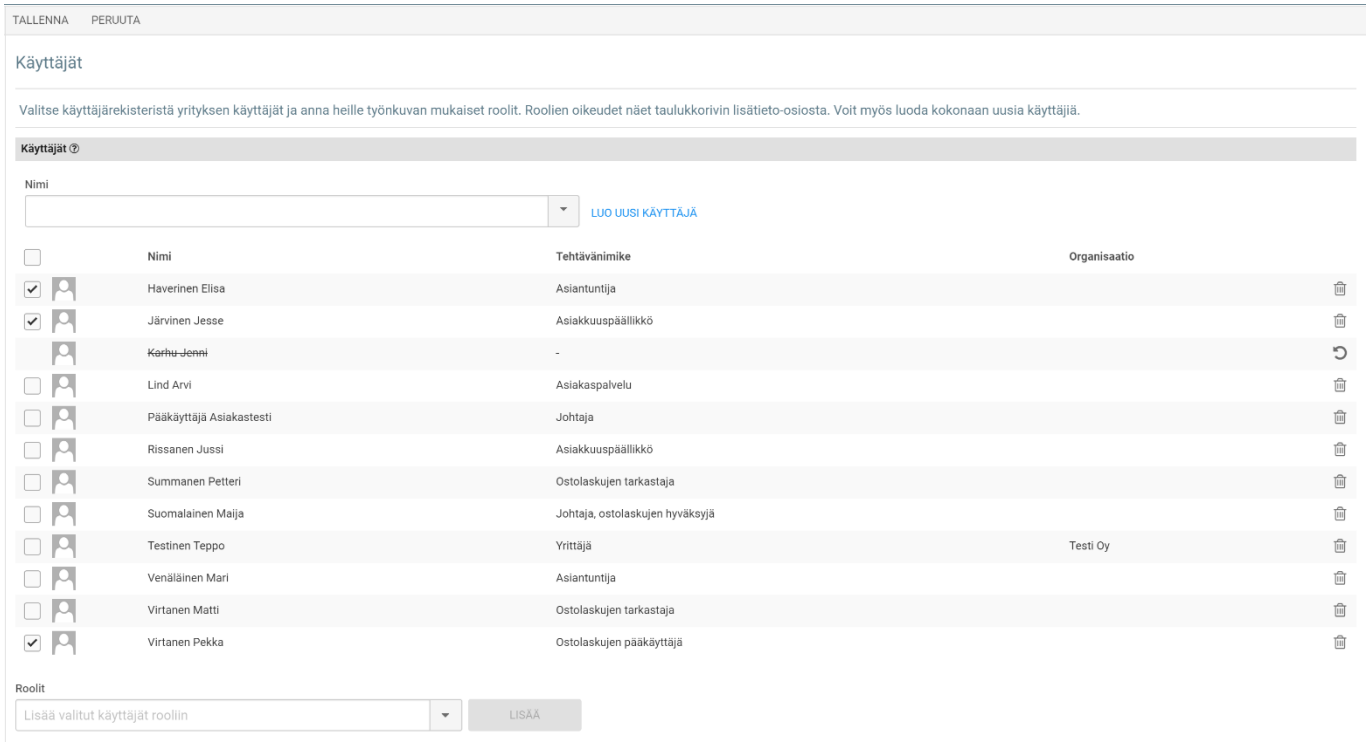
Tässä osassa käydään läpi olemassa ollut käyttäjien ja roolien hallintakäyttöliittymä ja sen ongelmat ja puutteet. Käyttöliittymän toiminta käydään läpi käyttöliittymästä otettujen ruutukaappausten avulla. Kuvat on otettu testiympäristöstä, ja kuvissa näkyvät henkilöiden nimet ovat esimerkkejä, eivät oikeita nimiä.

Kuvassa 7 on esitelty kokonaiskuva vanhasta hallintakäyttöliittymästä. Seuraavissa kuvissa perehdytään tarkemmin sen eri osiin.



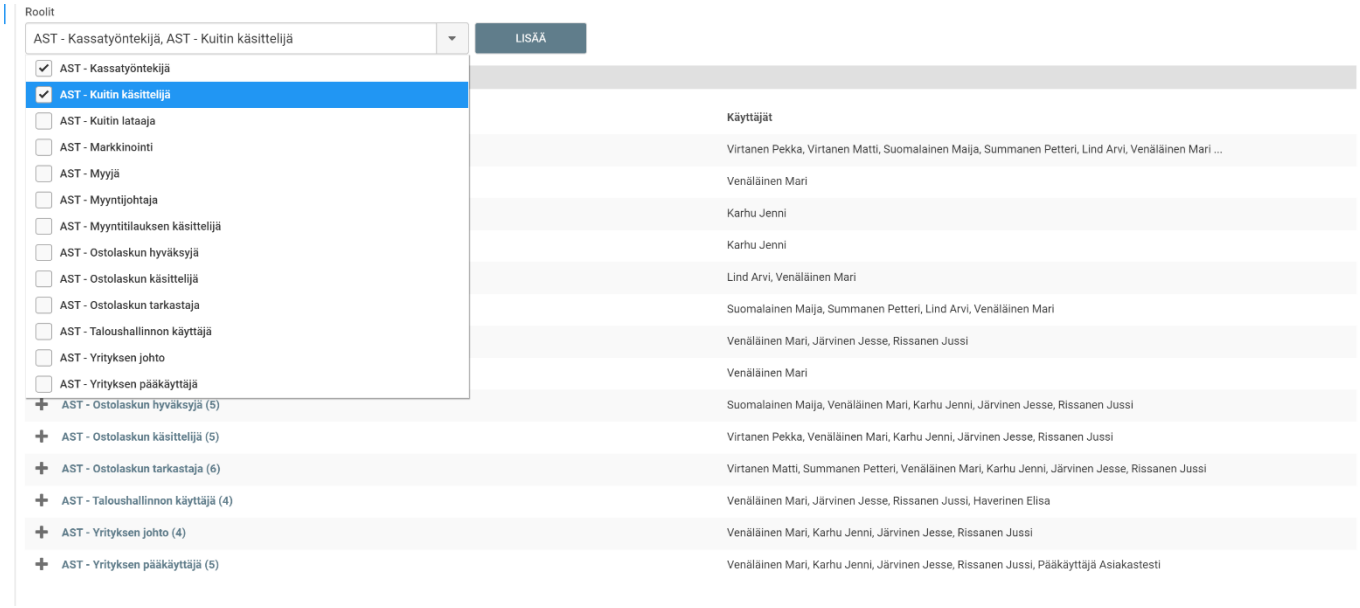
KUVA 7. Kokonaiskuva vanhasta yritystason käyttäjien ja roolien hallintakäyttöliittymästä

Kuvassa 8 on vanhan käyttöliittymän yläosa, jossa näkyvät kaikki johonkin valitun yrityksen oletusrooleihin kuuluvat käyttäjät, joita voidaan checkboxien avulla valita lisättäväksi rooleihin tai poistaa kerralla kaikista yrityksen rooleista roskakoripainikkeella. Tehtyjä muutoksia voidaan myös perua -painikkeen avulla, eikä esimerkiksi poistoa tehdä oikeasti, ennen kuin käyttäjä painaa yläreunan Tallenna-nappia. Listalle voidaan myös lisätä instanssin käyttäjiä valitsemalla listan yläpuolella olevasta valikosta, tai luomalla kokonaan uusia käyttäjiä klikkaamalla vieressä olevaa Luo uusi käyttäjä -nappia. Myöskään näitä lisäyksiä ei kuitenkaan oikeasti tallenneta ennen kuin käyttäjä painaa Tallenna-nappia.

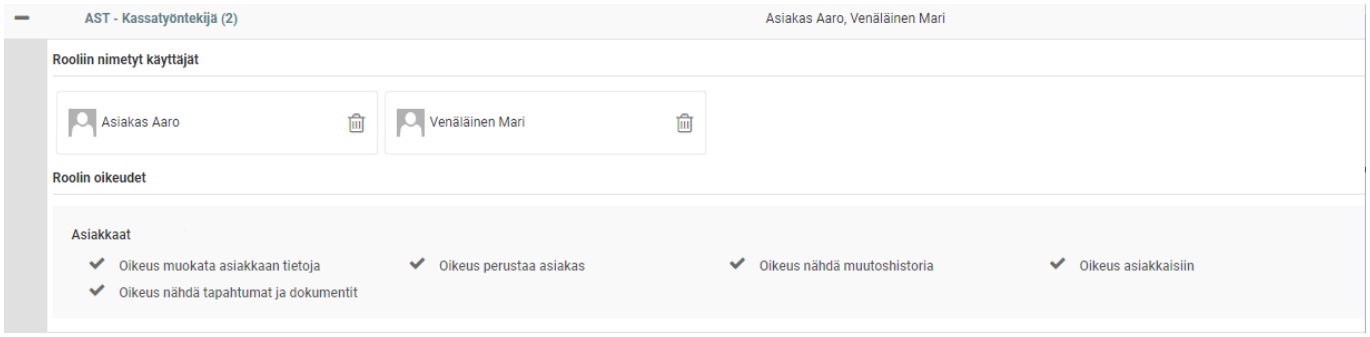


KUVA 8. Vanhan käyttöliittymän yläosa, jossa näkyvät oletusrooleihin kuuluvat käyttäjät, joita voi valita rooleihin lisättäväksi tai poistaa

Kuvissa 9-10 on kuvattu vanhan käyttöliittymän alaosa, jossa näkyvät yrityksen oletusroolit ja niiden käyttäjät sekä laajennetulla rivillä roolikohtaiset oikeudet yritykseen. Roolit-alasvetovalikosta voidaan valita ne roolit, joihin käyttäjälistalta valitut käyttäjät halutaan lisätä. Tämän jälkeen heidän lisäämisekseen rooleihin pitää vielä painaa Lisää-painiketta ja tämän jälkeen tallentaa muutokset erikseen yläpalkin Tallenna-painikkeella ennen kuin ne tulevat voimaan. Käyttäjiä voidaan myös poistaa yksittäisistä rooleista laajennetun rivin roskakoripainikkeella. Tämäkin kuitenkin vaatii erillisen tallennuksen tullakseen voimaan.



KUVA 9. Vanhan käyttöliittymän alaosa, jossa näkyvät kaikki yrityksen oletusroolit sekä niiden jäsenet ja oikeudet. valikosta voidaan valita ne roolit, joihin käyttöliittymän yläosasta valitut käyttäjät lisätään



KUVA 10. Vanhan käyttöliittymän alaosan roolilistan rivi laajennettuna, rooliin kuuluvia käyttäjiä voidaan poistaa (vaatii erillisen tallennuksen) sekä tarkastella roolin oikeuksia

Perehdytään ensin vanhan käyttöliittymän käytettävyysongelmiin. Yhteen näkymään on siinä ahdettu todella suuri määrä tietoa ja toiminnallisuutta, joiden vuorovaikutusta ei kuitenkaan ilmaista käyttöliittymässä kovin selkeästi. Asiakaspalautteen perusteella käyttöliittymä onkin ollut hankala käyttää.

Lisäksi yksinkertaisenkin operaation suorittaminen vaatii tarpeettoman paljon vaiheita, ja monimutkaisempien operaatioiden suorittaminen kerralla menee nopeasti todella sekavaksi, kun ennen tallennusta voidaan yhtä aikaa poistella ja lisäillä käyttäjiä useisiin eri rooleihin.

Esimerkkinä jo yksittäisenkin käyttäjän lisääminen yritykseen ja tiettyihin rooleihin voi vaatia jopa 5 eri vaihetta: Valitaan yritykseen valmiiksi kuulumaton käyttäjä dropdown-valikosta tai luodaan kokonaan uusi käyttäjä, valitaan lisätyn käyttäjän checkbox käyttäjälistalta, valitaan haluttu rooli tai roolit roolivalikosta, klikataan Lisää-painiketta ja tämän jälkeen vielä erikseen yläreunan pientä Tallenna-painiketta, jotta tehdyt muutokset tulevat voimaan. Kuten edellä

mainittiin, on operaatioita mahdollista suorittaa suuriakin määriä kerralla, jonka jälkeen pitää vielä huomata tallentaa, jotta muutokset tulevat voimaan. Erillistä tallennuksen tarvetta ei myöskään indikoida kovin selkeästi, ainoastaan sillä, että yläpalkin pieni Tallenna-painike tulee aktiiviseksi. Nämä kaikki seikat yhdessä tekevät vanhasta käyttöliittymästä käytettävyydeltään heikon.

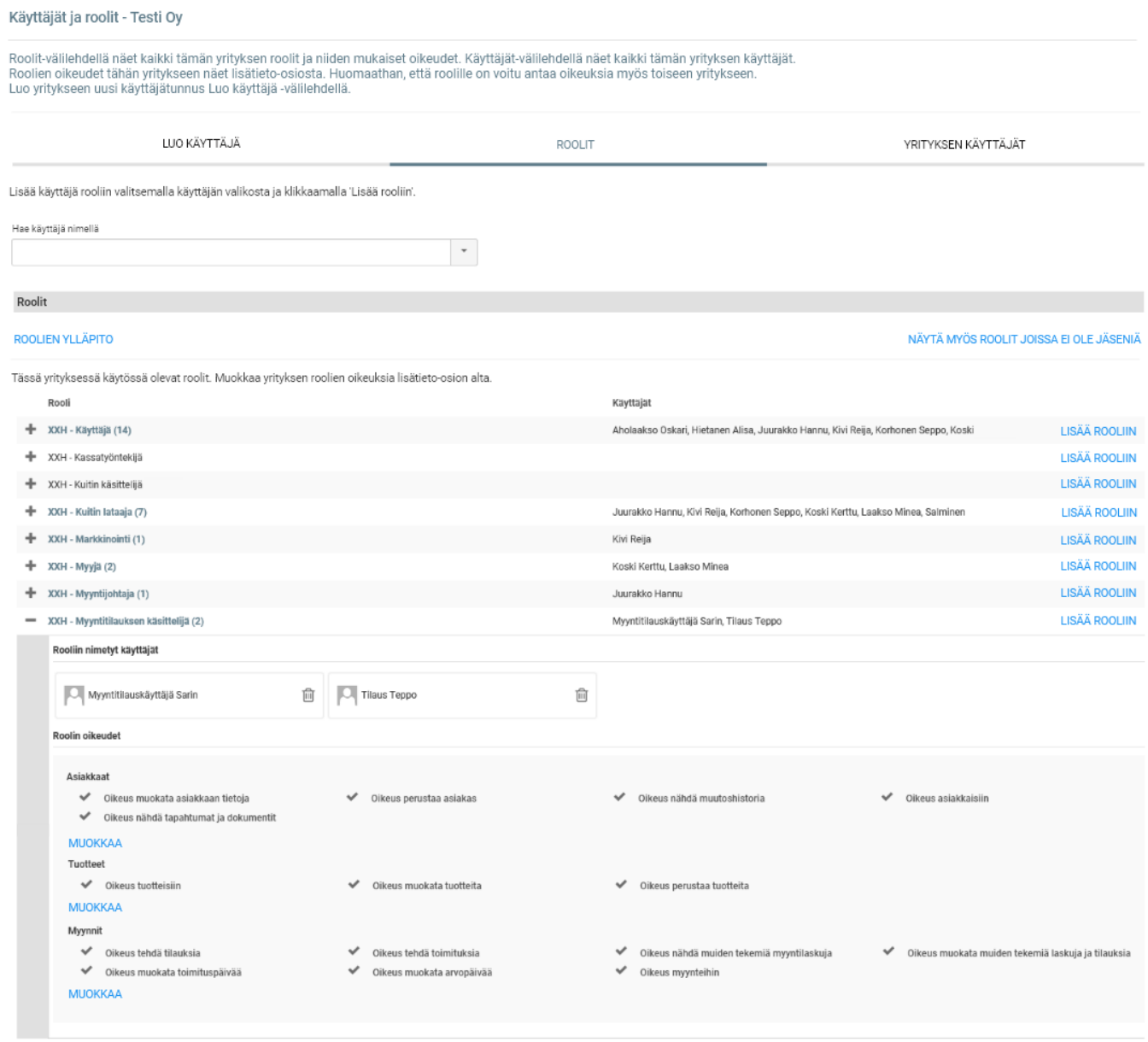
Kuten on jo monesti sivuttu, vanha käyttöliittymä ei myöskään täytä tehtäväänsä, koska siinä näkyvät ainoastaan yrityksen oletusroolit, joiden lisäksi todella moni asiakas käyttää myös instanssitason käyttäjäryhmiä ja henkilökohtaisia käyttöoikeuksia yritysten oikeuksien määrittämiseen. Tämä johtaa siihen, että vanha käyttöliittymä ei anna läheskään kaikissa tapauksissa ollenkaan realistista kuvaa siitä, kenellä kaikilla ja mitä käyttöoikeuksia kyseiseen yritykseen on. Kaikki asiakkaat eivät myöskään käytä ollenkaan oletusrooleja, ja tällöin vanhassa käyttöliittymässä ei näy mitään.

5.1.3 Uuden käyttöliittymän suunnittelu

Kun käyttäjähallinnan ja vanhan käyttöliittymän ongelmat ja kehitystavoitteet oli määritelty, oli aika aloittaa uuden käyttöliittymän suunnittelu niiden pohjalta. Uudesta käyttöliittymästä piirrettiin eri puolilta MaestroNG:n ylläpitosovelluksia otettujen ja vanhan käyttöliittymän kuvien pohjalta alustavat UI-kuvat käyttäen Adobe XD-ohjelmistoa.

Päätettiin, että uusi käyttöliittymä tultaisiin käyttökokemuksen suoraviivaistamiseksi ja selkiyttämiseksi jakamaan kolmeen välilehteen.

Välilehti, joka tulisi aukeamaan ensimmäisenä, kun uusi käyttöliittymä avataan, on Roolit-välilehti, jossa näkyisivät kaikki yrityksessä käytössä olevat roolit, vanhasta käyttöliittymästä poiketen myös instanssitason ryhmät, joilla on oikeuksia yritykseen sekä niiden oikeudet ja jäsenet. Roolit-välilehdestä piirretty UI-suunnitelma on esitelty kuvassa 11.



KUVA 11. Uuden käyttöliittymän Roolit-välilehdestä piirretty alustava UI-suunnitelma

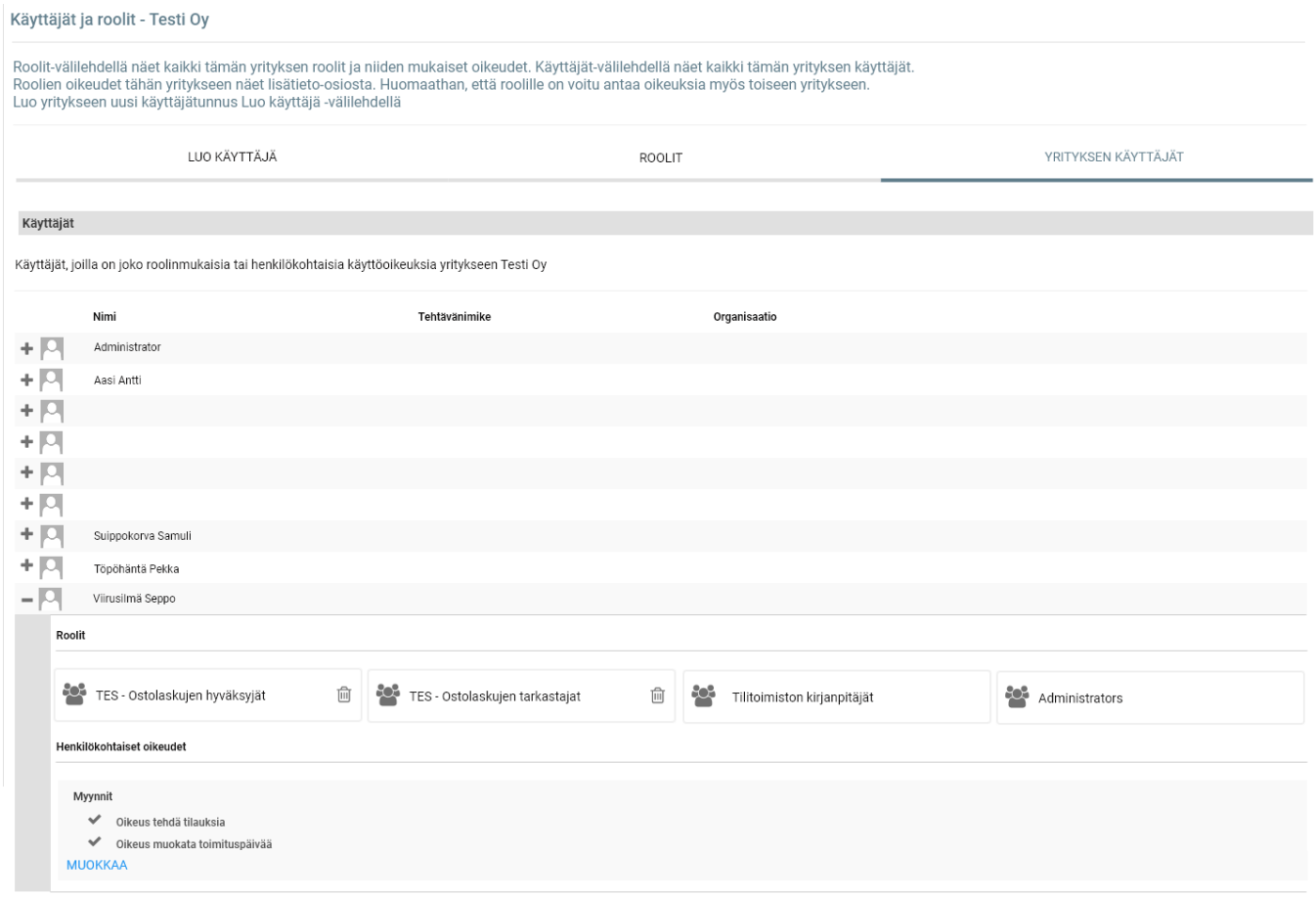
Roolin/ryhmän rivin laajentamalla näkisi vanhan käyttöliittymän tapaan roolin käyttäjät ja roolin oikeudet sovelluksittain. Jokaisen sovelluksen oikeuksien kohdalle generoitaisiin nyt myös Muokkaa-linkki, joka ohjaa kyseisen sovelluksen käyttöoikeushallintaan valittu yritys valmiiksi valittuna. Roolien ylläpito -painikkeella taas pääsisi instanssitason roolien hallintaan, josta voidaan hallita kaikkia rooleja koko instanssin tasolla. Näytä myös roolit, joissa ei ole jäseniä -painikkeella voitaisiin näyttää ja piilottaa kaikki roolit, joissa ei ole jäseniä.

Käyttöliittymän toimintaa tultaisiin yksinkertaistamaan siten, että siinä käsitellään vain yhtä käyttäjää kerrallaan, eikä erillistä tallennusta vaadita. Vertailuna vanhaan käyttöliittymään esimerkiksi käyttäjän lisääminen rooliin tapahtuisi siten, että käsiteltävä käyttäjä haetaan aina Hae käyttäjä nimellä -hakukentästä, kuului käyttäjä jo johonkin yrityksen rooliin tai ei.

Kun käsiteltävä käyttäjä on valittu, ilmestyisi roolilistan riveille Lisää rooliin -painike, jota painamalla käyttäjä lisättäisiin kyseiseen rooliin ja muutokset tallennettaisiin tietokantaan asti heti. Käyttäjä pysyisi tämän jälkeenkin valittuna, mikäli hänet olisi tarve liittää useampaan kuin yhteen rooliin.

Käyttöliittymä muuttuisi näin paitsi selkeämmäksi käyttää ja oppia, mutta myös käyttäjän lisäämisen rooliin vaatimat vaiheet putoaisivat viidestä kahteen. Vaikka käyttäjä olisi tarpeen lisätä useampaan kuin yhteen rooliin tai käsitellä useita käyttäjiä, pysyisi vaadittavien kokonaisklikkausten määrä silti pienempänä tai yhtä suurena kuin vanhassa käyttöliittymässä. Myöskään käyttäjien poistaminen rooleista ei vaatisi erillistä tallennusta vaan tallentuisi tietokantaan heti. Poistamisesta näytettäisiin roskakorikuvakkeen klikkaamisen jälkeen vielä varmistuskysymys.

Yrityksen käyttäjät -välilehdellä taas näytettäisiin kaikki ne käyttäjät, joilla on mitä vain kautta oikeuksia valittuun instanssin yritykseen: Oletusroolien tai instanssitason roolien kautta tai suoraan. Laajennetulla rivillä näkyisivät kaikki ne roolit, joihin käyttäjä kuuluu, sekä kaikki mahdolliset käyttäjän henkilökohtaiset oikeudet yritykseen sovelluksittain. Välilehden UI-suunnitelma on esitelty kuvassa 12.

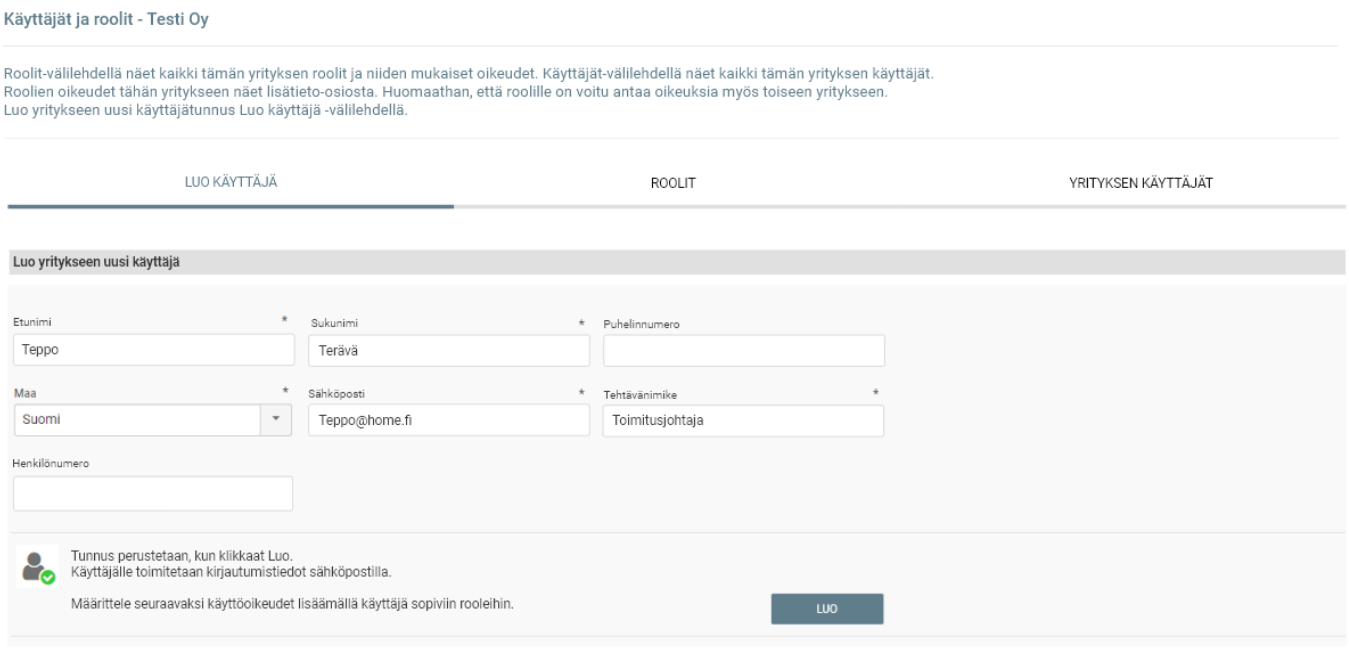


KUVA 12. Uuden käyttöliittymän Yrityksen käyttäjät -välilehdestä piirretty alustava UI-suunnitelma

Käyttäjä voitaisiin myös poistaa yksittäisistä rooleista tätä kautta, eikä tämäkään vaadi erillistä tallennusta. Ennen poistoa kysyttäisiin käyttäjältä varmistuskysymys. Henkilökohtaisten käyttöoikeuksien sovelluskohtaisista Muokkaa-linkeistä pääsisi Roolit-välilehden tapaan navigoimaan suoraan kyseisen sovelluksen käyttöoikeuksien hallintaan.

Tämä välilehti toimisi siis yhdessä roolit-välilehden kanssa kaivattuna lisänä järjestelmään, jonka avulla voitaisiin nyt nähdä yhdestä paikasta kaikki käyttäjät, jolla on valittuun yritykseen mitä tahansa kautta oikeuksia, ja nämä oikeudet sekä mistä ne tulevat.

Uuden käyttäjän luominen yritykseen lisättäisiin osaksi käyttöliittymää Luo käyttäjä -välilehdelle, ja uuden käyttäjän luominen järjestelmään tapahtuisi tulevaisuudessa pääosin tätä kautta, koska käyttäjät kuuluvat käytännössä aina johonkin yritykseen. Välilehden UI-suunnitelma on esitelty kuvassa 13.



KUVA 13. Uuden käyttöliittymän Luo käyttäjä -välilehdestä piirretty alustava UI-suunnitelma

Välilehdellä täytettäisiin uuden käyttäjän perustiedot, joista pakollisia olisivat nimi, maa, sähköposti ja tehtävänimike. Sähköpostista tarkistettaisiin myös oikea muoto, ja että se ei ole käytössä kellään toisella käyttäjällä, sillä sähköpostia käytetään myös käyttäjän kirjautumisnimenä. Kun pakolliset kentät on täytetty, aktivoituisi Luo-nappi, jota painamalla käyttäjä luotaisiin ja vietäisiin automaattisesti valitun yrityksen Käyttäjä-rooliin, jonka kautta uusi käyttäjä saa automaattisesti perusoikeudet yritykseen ja sen oletussivustoon ja hänelle lähetetään kirjautumiseen tarvittavat tiedot sähköpostilla.

Käyttäjä pystyisi siis heti luonnin jälkeen kirjautumaan järjestelmään eikä hänelle tarvitsisi lisätä tätä varten erikseen mitään tiettyjä käyttöoikeuksia. Luonnin jälkeen käyttöliittymä ohjautuisi Roolit-välilehdelle, jossa luotu uusi käyttäjä olisi valmiiksi valittuna ja ylläpitäjä ohjattaisiin näin suoraan lisäämään luotu käyttäjä hänen tehtäviensä mukaisiin rooleihin. Näin päästäisiin siihen tavoitteeseen, että käyttöoikeuksien hallinta saataisiin varsinkin uusien käyttäjien osalta muutettua miltei täysin roolipohjaiseksi.

5.1.4 Suunnitelmien esittely asiakkaille ja kehitysehdotukset

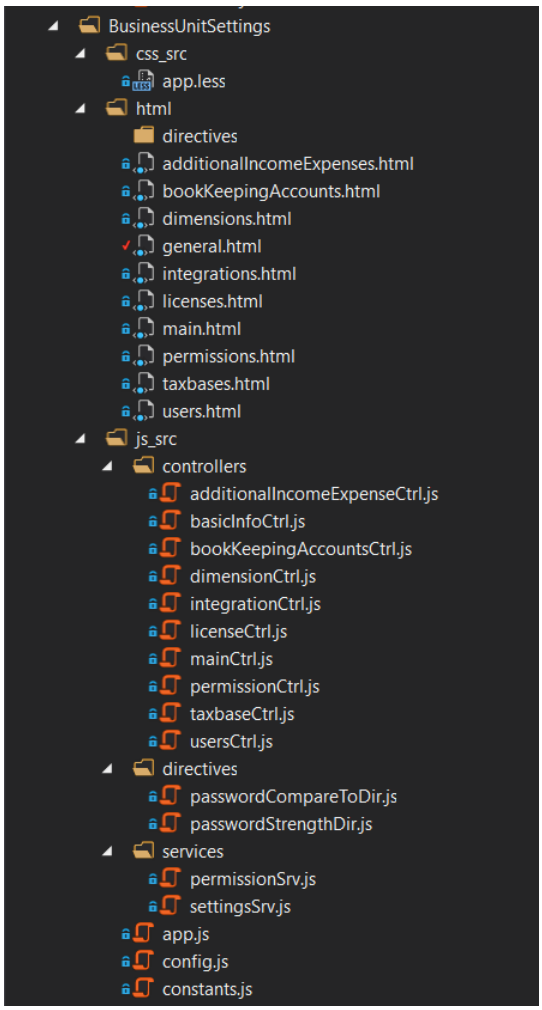
Kun uusi käyttöliittymä oli saatu pääpiirteittäin suunniteltua, valikoitiin muutama vanhaa käyttöliittymää käyttänyt ja siihen muutosta toivonut asiakas, joille uuden käyttöliittymän UI-suunnitelmia esiteltiin. He olivat suunnitelmiin suurelta osin tyytyväisiä, mutta heidän kauttaan saatiin myös muutama hyvä kehitysehdotus, jotka uuteen käyttöliittymään päätettiin toteuttaa. Näitä olivat esimerkiksi enemmän vaihtoehtoisia tapoja käyttäjän luontiin pelkän sähköpostipohjaisuuden lisäksi sekä Excel-vienti käyttäjien roolien jäsenyyksille ja oikeuksille. Asiakkaiden lisäksi joitakin kehitysehdotuksia saatiin myös muilta MaestroNG-kehitystiimin jäseniltä, kun heille esiteltiin käyttöliittymän kehitysversioita.

5.2 Toteutusvaihe

Tässä osassa kuvataan yleisellä tasolla uuden hallintakäyttöliittymän varsinainen kehitysprosessi ja siihen liittyneet vaiheet, joita havainnollistetaan muun muassa kuvilla koodista. Osan lopussa esitellään myös valmis uudistettu käyttöliittymä.

5.2.1 Toteutuksen lähtökohdat

Kun suunnitelmat saatiin tehtyä, päästiin aloittamaan varsinainen toteutus. Koko hakemistorakennetta ja uutta AngularJS-sovellusta ei tarvinnut luoda uudelle käyttöliittymälle erikseen, vaan uutta toteutusta alettiin kehittämään vanhan käyttöliittymän ”päälle”. Frontend-kehitys keskittyi pääosin BusinessUnitSettings-sovelluksen (MaestroNG:n yritystason ylläpitosovellus) osiin, jotka sisältävät käyttäjien ja roolien ylläpitokäyttöliittymään liittyvän asiakaspuolen logiikan sekä näkymät. Näitä olivat esimerkiksi usersCtrl.js -käsittelijä, joka sisältää itse käyttöliittymän toimintalogiikan, users.html -näkymätiedosto, jossa on käyttöliittymän layout, settingsSrv-palvelu, joka hoitaa palvelinpuolen kanssa kommunikoinnin ja config- ja constants -tiedostot, jotka sisältävät sovellukseen liittyvää konfiguraatiota sekä vakioarvoja. Käyttäjän luonnin uusia ominaisuuksia varten projektiin lisättiin muun muassa uuden käyttäjän salasanan asettamiseen liittyvät passwordCompareToDir- ja passwordStrengthDir -direktiivit. Kuvassa 14 on kuvattu yritysasetusten frontend-sovelluksen hakemistorakenne.



KUVA 14. MaestroNG:n yritysasetusten frontend-hakemistorakenne

Myös palvelinpuolen (backend) osalta voitiin osittain hyödyntää olemassa olleita rajapintoja ja malliluokkia sekä tietokannan kanssa keskustelevia luokkia. Uuden käyttöliittymän lisäykset ja logiikkamuutokset kuitenkin vaativat sekä frontendin että backendin osalta monien osien lähes täydellistä uudelleenkirjoitusta.

5.2.2 Backend-toteutus

Kun olemassa olevaan toteutukseen ja sen toimintaan saatiin sekä asiakas- että palvelinpuolen osalta perehdyttyä, voitiin sen uudistaminen tehtyjen suunnitelmien mukaiseksi aloittaa. Työ päätettiin aloittaa käyttäjä- ja roolidatan käyttöliittymään hakevan palvelinpuolen uudelleenkirjoittamisesta, koska frontend-kehitys olisi helpompaa, kun palvelin palauttaisi sitä aloitettaessa jo lopullisen muotoista käyttäjädataa.

Hakulogiikka täytyi uutta toteutusta varten kirjoittaa monelta osin uusiksi, varsinkin siihen liittyvien SQL-tietokantakyselyiden ja niistä saatavaa dataa sopivaan muotoon muokkaavan koodin osalta, jotta tulokseen saatiin mukaan asetettuiden tavoitteiden mukaisesti yrityksen

oletusroolien lisäksi myös kaikki yritykseen oikeuksia omaavat instanssitason käyttäjäryhmät sekä käyttäjien henkilökohtaiset oikeudet.

Käyttäjien, roolien ja oikeuksien hakuun liittyi kaksi SQL-kyselyä, joiden tuloksien perusteella palvelimen asiakaspuolelle palauttama data muodostettiin. Toinen kysely palautti listan kaikista yrityksen oletusroolien oikeuksista, sisältäen myös roolin tunnisteen, tyypin sekä nimen, jonka kautta oikeus tulee. Näiden perusteella palvelinpuolen koodi muodosti listan kaikista yrityksen oletusrooleista ja jokaisen roolin oikeuksista. Kyselyä, sen suorittavaa ja dataa muokkaavaa logiikkaa sekä malliluokkia täytyi muokata niin, että palautuvaan dataan saatiin mukaan myös instanssitason käyttäjäryhmät ja niiden oikeudet yritykseen ja sen sovelluksiin.

Toinen kysely taas palautti listan käyttäjien oikeuksista yritykseen sisältäen oletusroolin tunnisteen, josta oikeus tulee. Näiden perusteella muodostettiin lista yrityksen käyttäjistä, oletusrooleista, joihin he kuuluvat sekä oikeuksista, joita he kunkin roolin kautta saavat. Tähän kyselyyn, malleihin ja dataa muokkaavaan logiikkaan piti saada lisättyä instanssitason käyttäjäryhmien jäsenyyksien ja oikeuksien lisäksi myös lista käyttäjille suoraan annetuista henkilökohtaisista käyttöoikeuksista.

Hakulogiikan muutos aloitettiin uusien SQL-kyselyiden kehityksellä, johon hyödynnettiin SQL Server Management Studio -ohjelmistoa. Instanssitason ryhmien, niiden oikeuksien ja varsinkin käyttäjien suorien henkilökohtaisten oikeuksien saaminen mukaan kyselyiden palauttamaan dataan oikein oli suhteellisen pitkä ja haastava prosessi, joka vaati runsaasti yritystä ja erehdystä.

Tuloksena syntyneistä SQL-kyselyistä tuli kummastakin melko pitkiä ja kompleksisia, kumpikin sadan ja kahdensadan rivin väliltä. Jotta kaikki tiettyyn yritykseen ja sen sovelluksiin vaikuttavat käyttöoikeudet ja ryhmät sekä roolit saatiin selville, tarvittiin molemmissa kyselyissä runsaasti erilaisia ehtoja, dynaamisia osia sekä UNION- ja JOIN-lausekkeita. UNION-lauseke yhdistää SQL:ssä kahden taulusta dataa palauttavan SELECT-lauseen tulokset, ja erilaiset JOIN-lausekkeet taas yhdistävät kahdessa tietokantataulussa olevaa dataa toisiinsa niissä olevan yhteisen kentän arvon perusteella.

Kun SQL-kyselyt saatiin palauttamaan halutunlaista dataa, tehtiin seuraavaksi tarvittavat muutokset datahaun tekevään REST-rajapinnan metodiin, palvelinpuolen malliluokkiin sekä tietokantakyselyt suorittavaan ja niistä saatavan datan malliluokkien mukaiseksi muokkaavaan koodiin. Kuvassa 15 on kuvattu käyttäjä/roolidataa palauttavan REST-rajapinnan metodi.

```
[RoutePrefix("api/settings/businessunit/userandrole")]
public class ApiUserAndRoleController : ApiControllerBase
{
  [HttpGet]
  [Route("{businessunitid:int}")]
  public UserAndRoleModel GetBusinessUnitUsersAndRoles([FromUri]int businessUnitId)
  {
    using (WorkContext work = WorkContext.Create(WorkContextOptions.ReadOnly))
    {
      BusinessUnitPermissionManager.CheckBusinessUnitPermissions(businessUnitId, SecurityPermissionEnum.BusinessUnit_Access);

      string connectionString = ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

      if (string.IsNullOrEmpty(connectionString))
      {
        throw new ArgumentNullException("Connectionstring");
      }

      using (SqlConnection connection = new SqlConnection(connectionString))
      {
        connection.Open();

        UserAndRoleModel userAndRoleModel;
        using (SqlTransaction transaction = connection.BeginTransaction())
        {
          userAndRoleModel = UserManager.GetBusinessUnitUsersAndRoles(businessUnitId, connection, transaction);
          transaction.Commit();
        }

        // check if current user has permissions to modify instance level roles and add to model
        ISession session = SessionProvider.GetCurrent();
        int securityIdentifierId = SecurityContext.GetSecurityIdentifierId(SecurityObjectEnum.Instance);
        userAndRoleModel.HasInstanceLevelPerms = SecurityContext.HasPermission(session.UserId, securityIdentifierId, SecurityPermissionEnum.Instance_AdministerGroupMembershipsAll);
        return userAndRoleModel;
      }
    }
```

KUVA 15. REST-rajapinnan metodi, joka palauttaa malliluokkien mukaista käyttäjä- ja roolidataa JSON-muodossa, kun siihen lähetetään GET-pyyntö määriteltyyn osoitteeseen

Kun yritystunnisteen sisältävä GET-pyyntö saapuu REST-rajapintaan frontendistä, tarkistetaan ensimmäiseksi pyynnön suorittajan oikeudet kyseiseen yritykseen, jonka jälkeen avataan tietokantayhteys ja esitellään tyhjä UserAndRoleModel -tyyppinen muuttuja. Tämän jälkeen kutsutaan tietokantakyselyiden suorittamisesta ja niistä saatavan datan muokkaamisesta vastaavan UserManager-luokan GetBusinessUnitUsersAndRoles -metodia, joka palauttaa datan UserAndRoleModel-olioksi muotoiltuna esiteltyyn muuttujaan. Uuteen käyttöliittymään tarvitaan uutena ominaisuutena myös tieto, onko käyttäjällä oikeus muokata myös instanssitason ryhmien jäsenyyksiä, nekin kun tuodaan uuteen käyttöliittymään mukaan. Haetaan käyttäjän session avulla vielä tämä tieto, liitetään mallin HasInstanceLevelPerms -muuttujaan ja palautetaan malli frontendiin vastauksena pyyntöön.

Malli sisältää jäsenmuuttujan AllRoles, joka on lista BusinessUnitRoleModel -olioita, jotka mallintavat roolin/ryhmän ja sen oikeudet. Tässä muuttujassa ovat siis tallessa kaikki roolit/käyttäjäryhmät ja niiden oikeudet yritykseen. GroupType-muuttuja sisältää roolin tyypin, joka on uniikki jokaisella yrityksen oletusroolilla ja tietty arvo instanssitason rooleilla. ChangeType-muuttujaa taas hyödynnetään muutosten tallennusvaiheessa. UsersAndRoles -muuttuja taas sisältää listan UserAccountModel -olioita, joissa on tietoja käyttäjistä. UserAccountModel -luokka perii UserAccountOrGroupModel -luokan, jossa on loput käyttäjän tiedot ja roolien jäsenyydet. UserPermissionType -muuttujan avulla eritellään mahdolliset suorat oikeudet roolien kautta tulevista. Käyttäjän roolien jäsenyydet sisältävä Roles-muuttuja sisältää tällöin vain yhden tietyllä tavalla alustetun alkion, jossa suorat oikeudet ovat. Mikäli samalla käyttäjällä on sekä roolien kautta tulevia, että joitakin suoria oikeuksia, on käyttäjästä

UsersAndRoles -listalla kaksi alkiota, joissa UserPermissionType on eri. Nämä yhdistetään toisiinsa frontendin puolella. Backend-malliluokat on kuvattu kuvissa 16-18.

```
namespace Maestro.NG.Dashboard.Models
{
    using System.Collections.Generic;

    public class UserAndRoleModel
    {
        public List<BusinessUnitRoleModel> AllRoles { get; set; }

        public List<UserAccountModel> UsersAndRoles { get; set; }

        public UserAccountModel NewestUser { get; set; }

        // does current user have permissions to modify instance level roles
        public bool HasInstanceLevelPerms { get; set; }

        // differentiate new license deployment from bu settings users and roles, needs a bit different validation logic on save
        public bool IsDeployment { get; set; }
    }
}
```

KUVA 16. Frontendiin palautuvan datan mallintava UserAndRoleModel -luokka

```
namespace Maestro.NG.Dashboard.Models
{
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using Compositions;
    using Maestro.NG.Core.Types;

    public class BusinessUnitRoleModel
    {
        public int? Id { get; set; }

        public string RoleName { get; set; }

        [Required]
        public ChangeType ChangeType { get; set; }

        public List<ModulePermissionModel> ModulePermissions { get; set; }

        [Required]
        public UserGroupType GroupType { get; set; }
    }
}
```

KUVA 17. BusinessUnitRoleModel -luokka, joka mallintaa roolin/ryhmän ja sen oikeudet

```
namespace Maestro.NG.Dashboard.Models
{
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using Maestro.NG.Compositions;

    public class UserAccountOrGroupModel
    {
        public int? Id { get; set; }

        public UserPermissionType UserPermissionType { get; set; }

        public bool Inactive { get; set; }

        [Required]
        public UserType UserType { get; set; }

        [Required]
        public ChangeType ChangeType { get; set; }

        public string DisplayName { get; set; }

        public List<BusinessUnitRoleModel> Roles { get; set; }
    }
}
```

KUVA 18. UserAccountOrGroupModel -luokka, jonka käyttäjätietoja sisältävä UserAccountModel perii ja joka sisältää lisää tietoja käyttäjästä, rooleista, joihin hän kuuluu sekä oikeuksista, joita hänellä on roolien kautta tai suoraan

Rajapinnassa on GET-metodin lisäksi myös samaan osoitteeseen kohdistuva POST-verbin metodi, jossa vastaanotetaan tallennuspyynnön bodyssa GET-metodin palauttamaa mallia vastaava malli frontendissa muuttuneesta roolilistasta, tallennetaan tehdyt muutokset tietokantaan ja palautetaan muuttunut lista takaisin frontendille päivitetyn näkymän näyttämistä varten. Tähänkin tehtiin tarvittavat muutokset, jotta voitiin tukea myös instanssitason roolien muokkausta ja käyttäjän luonnin uusia ominaisuuksia. Tallennettavat muutokset tunnistetaan käyttäjä/roolilistan mallien ChangeType -muuttujan arvon perusteella. ChangeTypella on neljä eri arvoa: Unchanged, New, Changed ja Removed.

5.2.3 Frontend-toteutus ja valmiin käyttöliittymän esittely

Kun palvelinpuoli oli saatu muokattua uuden ylläpitokäyttöliittymän tarpeiden mukaiseksi, voitiin aloittaa käyttäjälle näkyvän puolen, asiakaspuolen eli frontendin kehitys. Tämä aloitettiin siivoamalla vanhasta käyttöliittymästä kaikki, mikä tulisi uuden käyttöliittymän myötä tarpeettomaksi, kuten esimerkiksi yläpalkin Tallenna- ja Peruuta-napit, käyttäjälista checkboxeineen, roolialasvetovalikko ja näihin liittyvä logiikka.

Kun näkymästä ja käsittelijästä oli saatu siivottua ne vanhan käyttöliittymän osat, jotka tulisivat uuden toteutuksen myötä turhiksi, voitiin varsinainen toteutus aloittaa. Kehitys aloitettiin tekemällä käyttöliittymään suunniteltu välilehtitoiminnallisuus, joka toteutettiin ui-bootstrap -kirjaston tab-komponenttien avulla (Kuva 19). Käyttöliittymää alettiin tekemään roolit-välilehdestä alkaen, joka tulisi aukeamaan ensimmäisenä, kun käyttäjä navigoi uuteen näkymään ja sisältäisi suurimman osan toiminnallisuuksista.

```
<div class="grouping col-xs-12" ng-if="!vm.errorMessage">
  <h4>{{ 'UsersAndRoles' | translate }}{{' - ' + vm.businessUnit.Name}}</h4>
  <h5 class="heading">{{ 'UsersAndRolesIngress' | translate }}</h5>
  <div class="rolesnav">
    <uib-tabset active="vm.activeTabIndex">
      <uib-tab index="0" ng-click="vm.activeTabIndex = 0">
        <uib-tab-heading class="uib-tab-heading">{{ 'CreateUser' | translate }}</uib-tab-heading>
        <ng-include src="'createUserTab'"></ng-include>
      </uib-tab>
      <uib-tab index="1" ng-click="vm.activeTabIndex = 1">
        <uib-tab-heading class="uib-tab-heading">{{ 'Roles' | translate }}</uib-tab-heading>
        <ng-include src="'rolesTab'"></ng-include>
      </uib-tab>
      <uib-tab index="2" ng-click="vm.activeTabIndex = 2">
        <uib-tab-heading class="uib-tab-heading">{{ 'CompanyUsers' | translate }}</uib-tab-heading>
        <ng-include src="'usersTab'"></ng-include>
      </uib-tab>
    </uib-tabset>
  </div>
```

KUVA 19. Käyttöliittymä jaettiin kolmeen välilehteen ui-bootstrapin tab-komponenttien avulla. Auki oleva välilehti määrittyy käsittelijän vm.activeTabIndex-muuttujan mukaan, joka alustetaan arvoon 1, jotta roolit-välilehti on auki oletuksena kun käyttöliittymä avataan

Seuraavaksi palvelimelta palautuva data oli saatava muokattua frontendissä sellaiseen muotoon, että se pystyttäisiin näyttämään käyttöliittymässä oikein ja että sitä voitaisiin käsitellä.

Ensimmäiseksi tehtiin tarpeelliset muutokset käyttäjä/roolidatan palauttavaa REST-rajapintaa kutsuvaan settingsSrv-palveluun sekä usersCtrl-käsittelijän alustuskoodiin, joka alustaa käyttöliittymän alkutilanteen muuttujat sekä vakioarvot ja kutsuu initialize-metodia, joka pyytää datan hakemista settingsSrv-palvelulta. Kun data on haettu onnistuneesti, se muokataan sopivaksi uudessa käyttöliittymässä käsittelyä varten createModels-metodissa käymällä dataa läpi for-silmukoissa ja luomalla listat rooleista, käyttäjistä ja roolien sekä käyttäjien oikeuksista sovelluksittain. Tässä metodissa erotellaan myös muun muassa käyttäjät uutta käyttäjät-välilehden listaa varten erilliseen taulukkoon ja etsitään siihen datasta myös käyttäjien henkilökohtaiset oikeudet UserPermissionType:n arvon perusteella. Käsittelijän alustuskoodin kutsumasta initialize-metodista, sen kutsumasta datan hakevasta metodista sekä createModels-metodista on kuvattu osa kuvissa 20-22.

```
var initialize = function () {

  $scope.$parent.vm.updateTitlebar('UsersAndRoles');

  toolbar.resetRight();
  // first time page landing, load roles and initialize Users array
  if (vm.businessUnit.Id) {
    // Check permission
    if (!permissionSrv.hasPermissionToEditUser(vm.businessUnit.Id) || !permissionSrv.hasPermissionToEditBusinessUnit(vm.businessUnit.Id)) {
      vm.errorMessage = $translate.instant('UserDoesNotHavePermissions');
      return;
    }

    loadUsersAndRoles();
  }
  else {
    vm.errorMessage = $translate.instant('SelectCompany');
  }
}
```

KUVA 20. Osa käsittelijän alustuskoodissa kutsuttavasta initialize-metodista, jossa muun muassa asetetaan sovelluksen titlebar:n teksti, sekä tarkistetaan, onko käyttäjällä oikeudet valittuun yritykseen. Mikäli on, kutsutaan loadUsersAndRoles() -metodia, joka kutsuu käyttäjädatan hakevaa settingsSrv-palvelun metodia

```
var loadUsersAndRoles = function () {
  settingsService.loadUsersAndRoles(vm.businessUnit).then(createModels).catch(onLoadError);
}
```

KUVA 21. Käsittelijän loadUsersAndRoles-metodi, joka kutsuu käsittelijään injektoidun settingsService-palvelun metodia, joka hakee käyttäjä/roolidatan rajapinnasta. Jos pyyntö onnistuu, kutsutaan haetun datan käyttöliittymälle sopivaan muotoon muokkaavaa käsittelijän createModels-metodia

```
var createModels = function (combinedData) {

  // take copy of all roles, have to send this back on save; because of reasons
  vm._AllRoles = angular.copy(combinedData.AllRoles);

  // assign models; data might come from cache, therefore copy()

  // remove direct permission user rows from the array used on the roles tab based on UserPermissionType
  vm.users = _.sortBy(combinedData.UsersAndRoles, ['DisplayName']).filter(function (u) { return u.UserPermissionType == 1 });
  vm.usersTabUsers = angular.copy(vm.users);
  vm.directPermUsers = _.sortBy(combinedData.UsersAndRoles, ['DisplayName']).filter(function (u) { return u.UserPermissionType == 2 });
  // find possible users with only direct permissions and if there are any, concat to the users tab array
  var tempArr = _.differenceBy(vm.directPermUsers, vm.users, 'Id');
  if (tempArr) {
    vm.usersTabUsers = _.concat(vm.usersTabUsers, tempArr);
    vm.usersTabUsers = _.sortBy(vm.usersTabUsers, ['DisplayName']);
  }
   // build an array that includes users' direct permissions and also users with only direct permissions for the users tab
  for (var l = 0; l < vm.usersTabUsers.length; l++) {
    for (var m = 0; m < vm.directPermUsers.length; m++) {
      if (vm.usersTabUsers[l].Id === vm.directPermUsers[m].Id) {
        vm.usersTabUsers[l].DirectPermissions = vm.directPermUsers[m].Roles[0].ModulePermissions;
      }
      if (vm.usersTabUsers[l].DirectPermissions) {
        // add module titles to direct permission array
        for (var n = 0; n < vm.usersTabUsers[l].DirectPermissions.length; n++) {
          var module = vm.usersTabUsers[l].DirectPermissions[n];
          module.Title = modulePermissionTitles[module.SecurityObject];
        }
      }
    }
  }

  vm.roleList = [];
```

KUVA 22. Osa backendistä palautuvan datan lopullisen muotoon käyttöliittymässä käsittelyä varten muokkaavasta createModels-metodista, jossa rakennetaan käyttäjät-välilehteä varten erillinen käyttäjätaulukko, jossa ovat mukana myös käyttäjien mahdolliset henkilökohtaiset oikeudet, tässä metodissa muodostetaan myös mm. roolit-välilehdelle listat rooleista ja niiden jäsenistä sekä käyttöoikeuksista sovelluksittain

Kun rajapinnasta palautuva käyttäjä/roolidata oli saatu sopivaan muotoon, voitiin aloittaa itse roolit-välilehden näkymän kehitys. Tämä aloitettiin käyttöliittymän yläosasta muokkaamalla käyttäjähakukentän logiikkaa siten, että käsiteltävä käyttäjä valittaisiin vanhasta käyttöliittymästä poiketen aina siitä, oli käyttäjä jo jossain yrityksen roolissa tai ei. Hakutulos sidotaan searchResult -olioon ja valittaessa käyttäjä kutsutaan onUserSelected -metodia, jossa käyttäjän tiedoista poimitaan tarpeelliset ja laitetaan käyttäjä selectedUser-muuttujaan. Tämän perusteella näytetään muun muassa roolilistan rivien Lisää rooliin -painikkeet ja käyttäjähakukentän viereinen valitusta käyttäjästä kertova infoteksti. Näkymän yläosaan lisättiin myös Roolien ylläpito- ja Näytä roolit, joissa ei ole jäseniä -painikkeet, joista pääsee instanssitason roolien ylläpitonäkymään sekä saa näytettyä/piilotettua tyhjät roolit. Uudessa näkymässä jäsenettömät roolit ovat piilotettuna oletuksena, jolloin esimerkiksi sellaisissa yrityksissä, joissa yrityksen oletusrooleja ei käytetä lainkaan, eivät ne ole turhaan näkyvissä listassa. Kuvassa 23 on esitetty sivun yläosan näkymäkoodi.

```
<script id="rolesTab" type="text/ng-template">
    <p class="usersParagraph">{{'AddUserGuide' | translate}}</p>
    <div class="row user-select">
      <div class="col-sm-5">
        <label class="text-muted">{{ 'FindUserByName' | translate }}</label>
        <div class="userAutoCompleteWrapper">
        <input name="userOrGroup"
              data-maestro-auto-focus
              min-length="1"
              number-search-enabled="1"
              maestro-id-name-autocomplete
              ng-model="vm.searchResult"
              change-fn="vm.onUserSelected(vm.searchResult)"
              url="{{ :vm.usersUrl}}"
              type="text" />
         <span>
          <label class="label label-warning newUserLabel" ng-if="vm.selectedUser.NewUser != null">{{'New' | translate}}</label>
         </span>
        </div>
      </div>
      <div class="col-md-6 title" ng-if="vm.selectedUser && !vm.selectedUser.OrganizationNames">
        <p>{{'SelectedUser' | translate}}{{': ' + vm.selectedUser.Name}}</p>
      </div>
      <div class="col-md-6 title" ng-if="vm.selectedUser && vm.selectedUser.OrganizationNames">
        <p>{{'SelectedUser' | translate}}{{': ' + vm.selectedUser.Name + ' (' + vm.selectedUser.OrganizationNames + ')'}}</p>
      </div>
    </div>
    <div class="newUserInfoMsg" ng-if="vm.selectedUser.NewUser != null">
    <i id="newUserInfoIcon" class="icon fa fa-info-circle newUserInfoMsg"></i><p class="newUserInfoMsg" style="font-size:80%; margin-left:5px;">{{'AddNewUserToRoles' | translate}}</p>
    </div>
    <h5 class="grouping-heading">{{ 'Roles' | translate }}</h5>
    <h4 class="heading usersParagraph">
      <a ng-href="/dashboard/settings/roles/list?bu={{vm.businessUnit.Name}}" target="_blank" rel="noopener noreferrer" class="pull-left">{{'RolesManagement' | translate | uppercase}}</a>
      <a class="pull-right" ng-click="vm.toggleHide()">{{vm.hideEmptyText | translate | uppercase}}</a>
    </h4>
```

KUVA 23. Roolit-välilehden yläosan näkymän määrittävä HTML-koodin osa

Seuraavaksi tehtiin tarvittavat muutokset roolilistan toimintaan, esimerkiksi ng-if direktiiveillä toteutetut ehdot, joiden perusteella Lisää rooliin -napit generoidaan kullekin rooliriville, nappia ei näytetä esimerkiksi silloin, jos valittu käyttäjä on jo kyseisessä roolissa tai jos rooli on instanssin rooli ja näkymää käyttävällä ylläpitäjällä ei ole oikeuksia muokata instanssitason rooleja. Myös rooliriveillä olevien roolin jäsenien poistonapit muokattiin toimimaan käyttöoikeuksien perusteella ja siten, etteivät ne vaadi erillistä tallennusta mutta näyttävät kuitenkin ennen varsinaista poistoa käyttäjälle varmistusdialogin. Tarvittiin myös logiikka, jonka avulla roolirivin jokaisen sovelluksen käyttöoikeustietojen yhteyteen saatiin generoitua linkki, jonka kautta kyseisiä oikeuksia pääsee muokkaamaan valitussa yrityksessä. Roolirivit ja roolien jäsenet sekä oikeudet tulostetaan näkyviin hyödyntäen sisäkkäisiä AngularJS:n ng-repeat -direktiivejä. Kuvissa 24-25 on esitelty osa lisää rooliin -napin klikkauksella kutsuttavasta metodista, joka lisää käyttäjän rooliin sekä sovelluskohtaisten käyttöoikeuksien muokkauslinkit generoivasta metodista.

```
vm.assignUserToRole = function (selectedUser, role) {
  event.stopPropagation();
  var user = _.find(vm.users, { Id: selectedUser.Id });
  // assign if not already assigned
  if (user) {
    //ensure array existence before pushing
    if (!user.Roles) {
      user.Roles = [];
    }
    // add role for user
    user.Roles.push({
      Id: role.Id,
      RoleName: role.RoleName,
      ChangeType: vm.changeTypes.NEW,
      GroupType: role.GroupType
    });
    // add user to roles list
    role.Users.push(user);
    // save changes
    saveUsersAndRoles();
  }
```

KUVA 24. Käsittelijän assignUserToRole metodi, joka lisää valitun käyttäjän rooliin, jonka Lisää rooliin -painiketta painetaan käyttäjälistalta ja tallentaa muutokset

```
vm.createModulePermLink = function(securityObject)
{
  var link = "";
  switch (securityObject) {
    case 1:
      link = "/Products/settings/permissions?businessunitid=" + vm.businessUnit.Id;
      break;
    case 3:
      link = "/dashboard/settings/businessunit/permissions?businessunitid=" + vm.businessUnit.Id;
      break;
    case 5:
      link = "/suppliers/settings/permissions?businessunitid=" + vm.businessUnit.Id;
      break;
    case 7:
      link = "/purchaseinvoice/settings/permissions"
      break;
    case 11:
      link = "/Customers/settings/permissions?businessunitid=" + vm.businessUnit.Id;
      break;
    case 25:
      link = "/Sales/settings";
      break;
    case 26:
      link = "/Products/settings/permissions?businessunitid=" + vm.businessUnit.Id;
      break;
    case 27:
      link = "/Sales/settings";
      break;
    case 39:
      link = "/PurchaseInvoice/Receipt/settings/permissions";
      break;
    case 40:
      link = "/Dashboard/reports/settings/permissions?businessunitid=" + vm.businessUnit.Id;
      break;
    default:
      link = "";
      break;
  }
  return link;
}
```

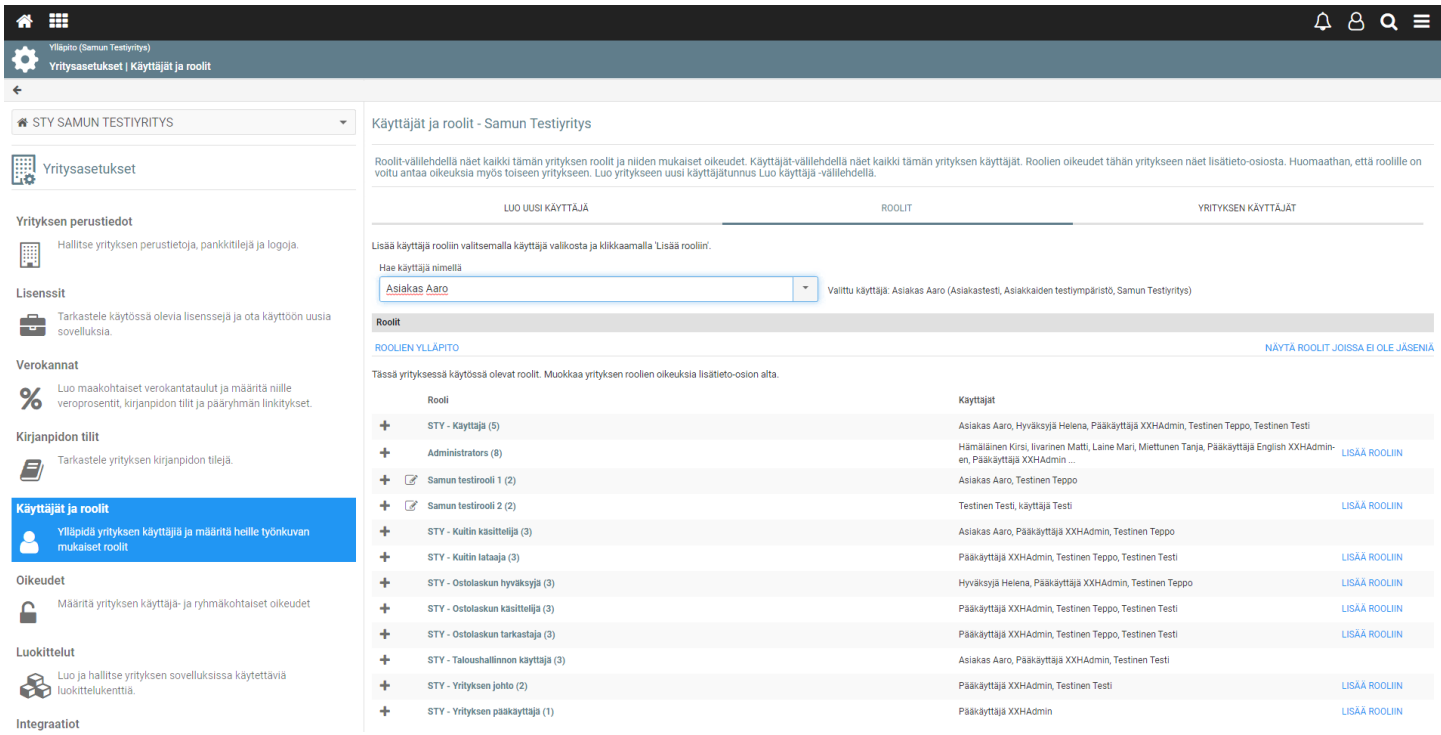
KUVA 25. Käsittelijän createModulePermLink -metodi, jonka avulla saadaan generoitua roolilistan oikeuksiin sovelluskohtaiset muokkauslinkit

Valmista Roolit-välilehteä on esitelty kuvissa 26-29. Siihen saatiin toteutettua kaikki suunnitellut ominaisuudet, jotka tekevät yrityksen käyttöoikeuksien hallinnasta huomattavasti helpompaa. Välilehdellä olivat nyt näkyvissä yrityksen oletusroolien lisäksi myös kaikki yritykseen vaikuttavat instanssitason ryhmät. Näitä ovat kuvassa 26 instanssitason ylläpitäjäryhmä Administrators, ja itse luodut instanssitason roolit Samun testirooli 1 ja 2, joille on annettu yritykseen oikeuksia. Nämä on myös valmiissa käyttöliittymässä merkitty erillisellä ikonilla.

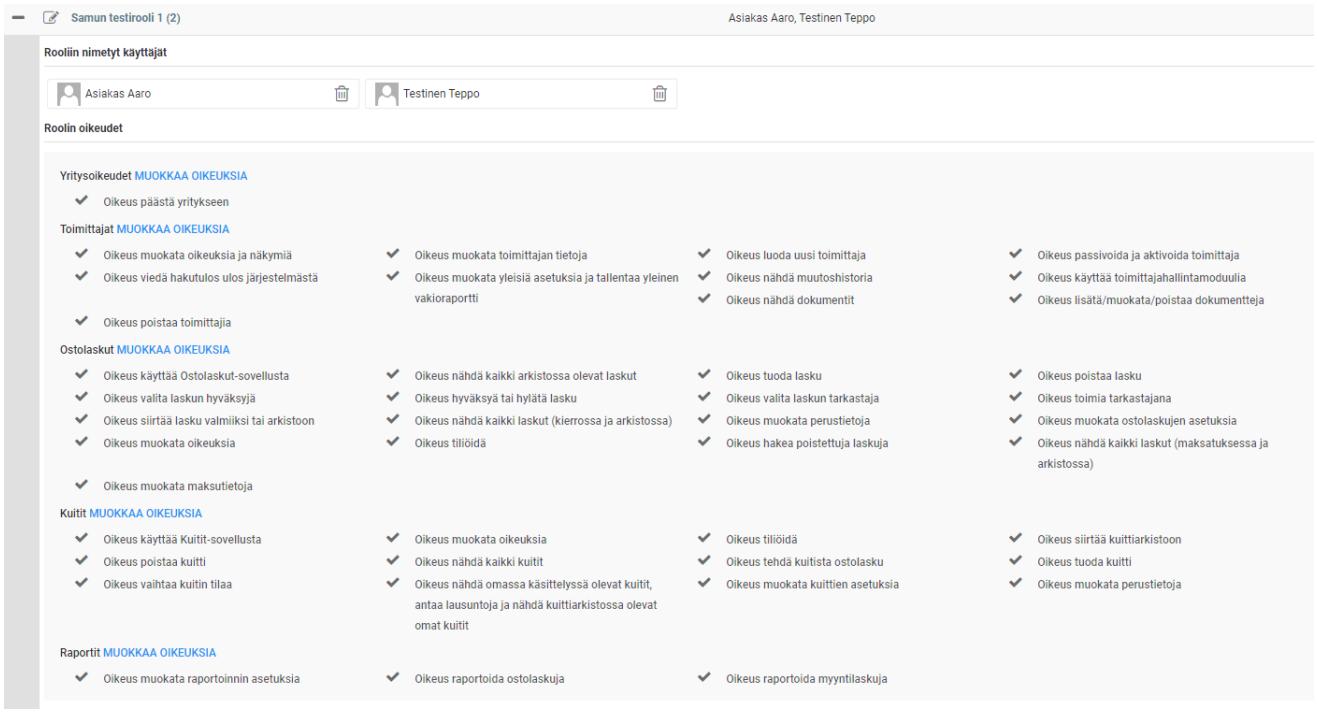
Uudessa käyttöliittymässä käsitellään aina yhtä käyttäjää kerrallaan, joka haetaan ja valitaan sivun yläosan hakukentästä, joka osaa automaattisesti ehdottaa hakuehtoon täsmääviä käyttäjiä. Kun käsiteltävä käyttäjä on valittu, voidaan käyttäjä lisätä haluttuihin rooleihin klikkaamalla ko. rivin Lisää rooliin -painiketta. Tämän muutoksen avulla yksittäisen operaation vaatima klikkausten määrä putosi huomattavasti ja käyttöliittymää saatiin selkeytettyä entiseen verrattuna roimasti.

Kuvassa 27 on esitelty laajennettu roolin rivi uudessa käyttöliittymässä. Roolin jäsenet mahtuvat nyt entistä pienempään tilaan, ja roolien oikeuksia päästään tarvittaessa muokkaamaan kätevästi muokkaa oikeuksia -painikkeiden avulla. Uusi varmistusdialogi, joka näytetään, kun käyttäjä klikkaa roolin jäsenen poistopainiketta, on esitelty kuvassa 28.

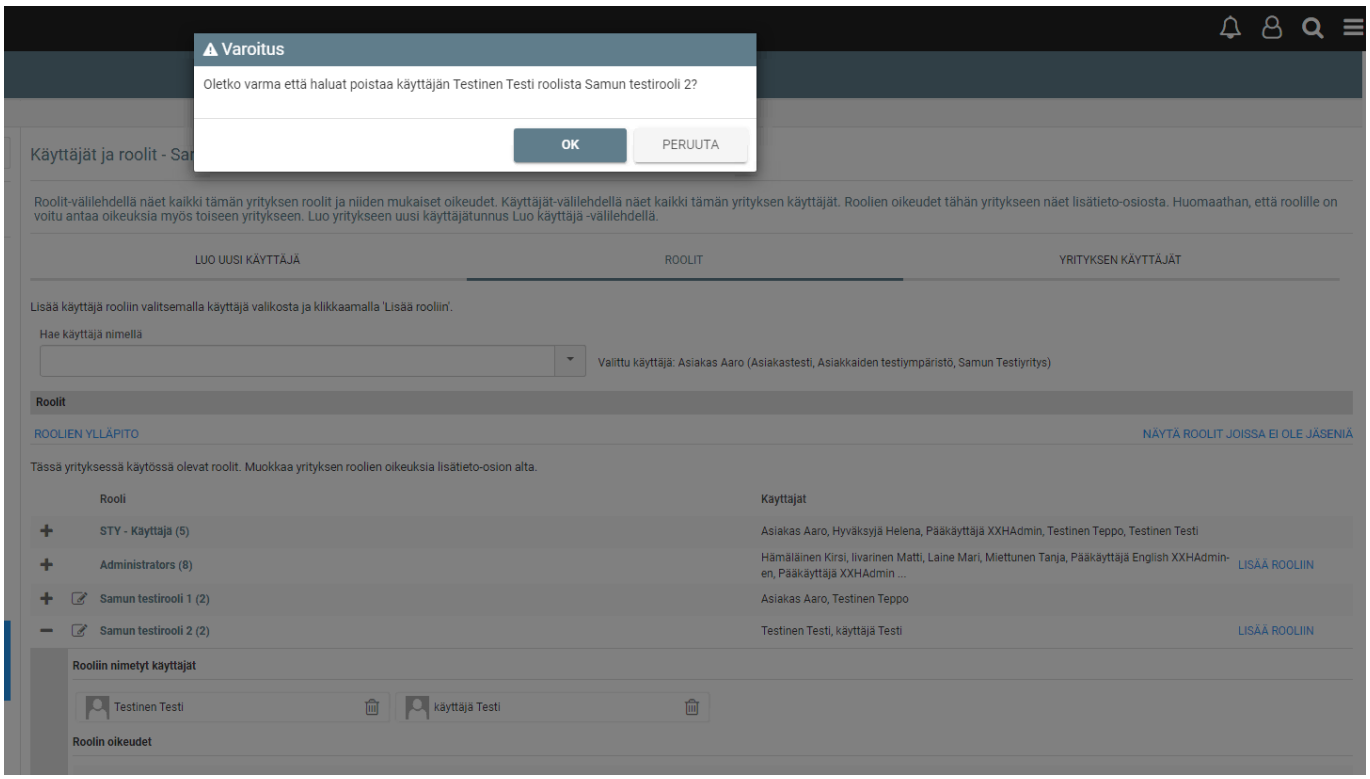
Kuvassa 29 on kuvattu aiemmin uudistamani instanssitason roolien hallintakäyttöliittymä, johon pääsee kätevästi uudesta Roolien ylläpito -painikkeesta, tähän käyttöliittymään tehtiin myös ominaisuus, jolla instanssin roolilista on automaattisesti suodatettuna valittuna olleella yrityksellä, kun käyttöliittymään tullaan uuden yritystason käyttöliittymän kautta.



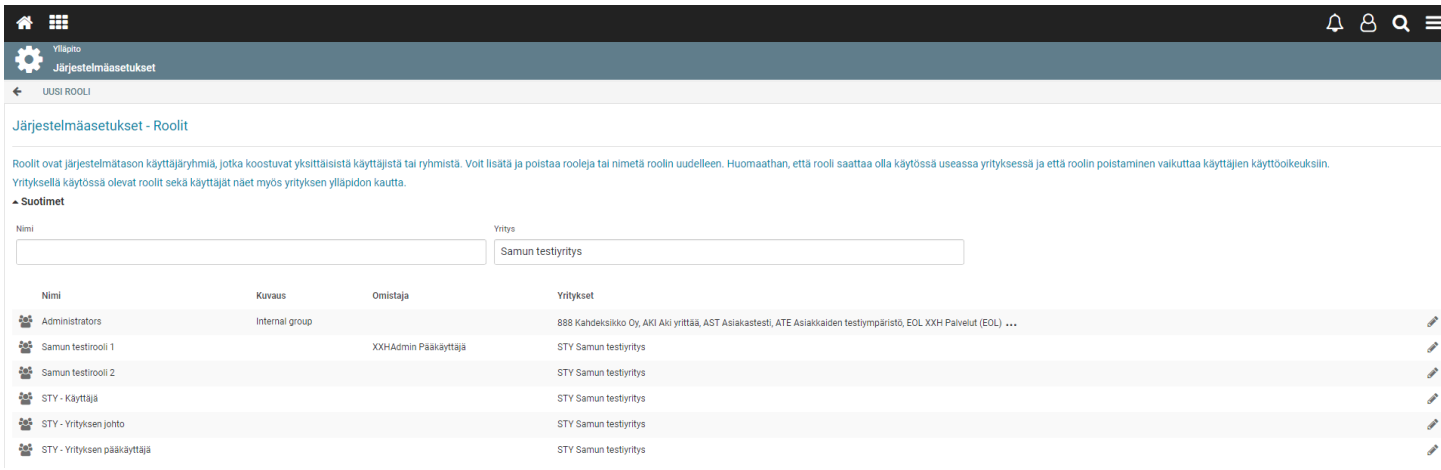
KUVA 26. Uuden käyttöliittymän Roolit-välilehti



KUVA 27. Uuden käyttöliittymän roolilistan rivi laajennettuna



KUVA 28. Roolin jäsenen poiston varmistusdialogi uudessa käyttöliittymässä



KUVA 29. Instanssitason roolien hallintakäyttöliittymä, kun siihen on navigoitu uuden yritystason käyttöliittymän Roolien ylläpito -linkin kautta. Listaa suodatetaan automaattisesti yritysasetuksissa valittuna olleella yrityksellä

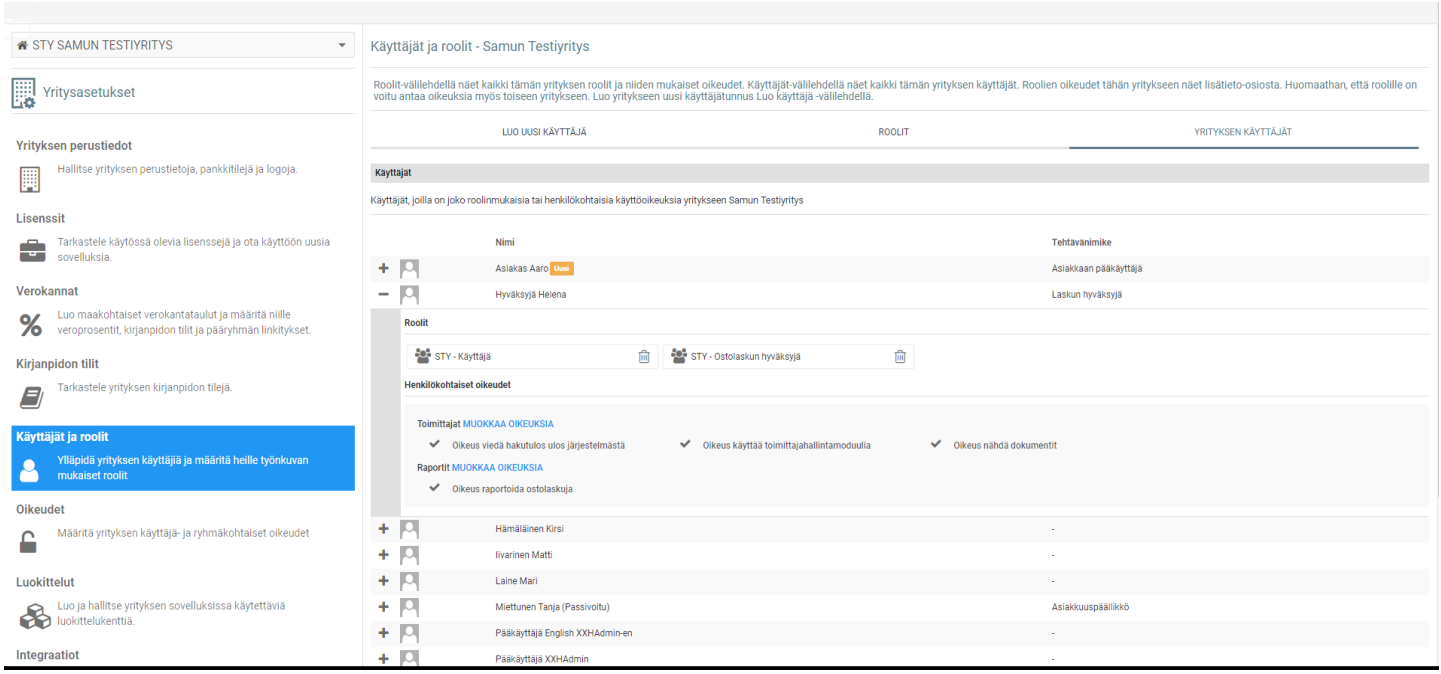
Roolit-välilehden valmistuttua lähdettiin seuraavaksi tekemään yrityksen käyttäjät -välilehteä, jossa näytettäisiin kaikki järjestelmän käyttäjät, joilla on mitä tahansa kautta oikeuksia valittuun yritykseen, heidän jäsenyytensä eri rooleihin, sekä mahdolliset henkilökohtaiset käyttöoikeudet. Data muokattiin sitä käsittelevässä koodissa jo valmiiksi käyttäjät-välilehteäkin varten, joten voitiin saman tien alkaa kehittää sen käyttöliittymää.

Välilehdelle toteutettiin roolit-välilehden roolilistan toteutusta vastaava lista käyttäjistä, ja laajennetun käyttäjärivin alle lista käyttäjän rooleista ja henkilökohtaisista käyttöoikeuksista,

mikäli käyttäjällä oli sellaisia. Tehtiin myös roolilistan poistotoiminnallisuutta vastaava mahdollisuus poistaa käyttäjältä roolien jäsenyyksiä. Käyttäjälistalla näytetään myös Uusi-tagi sellaisten käyttäjien kohdalla, jotka on luotu viimeisen 24 tunnin aikana.

Järjestelmässä voidaan myös passivoida käyttäjiä, tämä tarkoittaa, että kyseisen käyttäjän tunnukset on lukittu eikä hänen ole mahdollista kirjautua sisään. Passiiviset käyttäjät on merkitty käyttäjälistalla (Passivoitu) -tekstillä. Tämä helpottaa ylläpitäjiä siivoamaan pois passivoiduille käyttäjille jääneet käyttöoikeudet, mikäli käyttäjää ei ole tarkoitus aktivoida enää lähitulevaisuudessa.

Uusi Käyttäjät-välilehti tekee erittäin helpoksi nähdä kaikki käyttäjät, joilla on mitä tahansa oikeuksia valittuun yritykseen, ja siinä ovat mukana myös käyttäjien henkilökohtaiset oikeudet, joita ei voinut vanhasta käyttöliittymästä nähdä. Käyttäjät-välilehden tarjoamat tiedot tekevät myös oikeuksien kopioinnin käyttäjiltä toiselle helpoksi, kun ylläpitäjä näkee heti, mihin kaikkiin rooleihin käyttäjä pitää lisätä/mitä oikeuksia pitää antaa, jotta hän saa samat oikeudet kuin olemassa oleva yrityksen käyttäjä. Valmis yrityksen käyttäjät -välilehti on esitelty kuvassa 30.



KUVA 30. Uuden käyttöliittymän Yrityksen käyttäjät -välilehti

Kun yrityksen käyttäjien ja roolien oikeuksien raportointi- ja muokkausmahdollisuudet tarjoavat roolit- ja käyttäjät -välilehdet oli saatu valmiiksi, oli jäljellä vielä uusien käyttäjien luomisen suoraan yritykseen mahdollistavan Luo uusi käyttäjä -välilehden kehitys.

Alustavista suunnitelmista poiketen välilehdelle päätettiin lisätä pelkän sähköpostipohjaisen käyttäjän luonnin lisäksi myös mahdollisuus luoda instanssin tunnukseen ja käyttäjän etunimeen pohjautuvat tunnukset, sekä satunnaisen generoinnin ja salasanan sähköpostilla toimituksen lisäksi myös mahdollisuus määrittää uuden käyttäjän salasana itse. Vanhan käyttöliittymän erilliseen näkymään aukeava käyttäjän luonti tuki pelkästään sähköpostipohjaisia käyttäjiä.

Ensimmäiseksi välilehdelle toteutettiin käyttöliittymäsuunnitelmia mukaileva lomake, johon lisättiin myös valintaruudut ja logiikka erilaisia käyttäjän luontitapoja varten. Pakollisten kenttien täyttämisen ja pituuden tarkasteluun käytettiin AngularJS:n lomakkeiden validointiin tarjoamia direktiivejä, joiden mukaan uuden käyttäjän tallentava Luo-nappi aktivoitiin tai disabloitiin. Sähköpostikentästä validoitiin myös sähköpostin muoto sekä se, ettei samaa sähköpostia ole käytössä kirjautumisnimenä yhdelläkään muulla järjestelmän käyttäjällä. Salasanan itse määrittelemistä varten käyttöliittymään lisättiin myös salasanakentät ja projektiin direktiivit, joiden avulla salasanan vahvuus ja kenttien täsmääminen tarkistettiin.

Kun välilehden käyttöliittymä oli valmis, lisättiin tallennuksen hoitavaan ja muuttuneen käyttäjä/roolilistan palauttavaan palvelinpuolen koodiin vielä logiikka, joka palauttaa viimeksi luodun käyttäjän. Tämän avulla frontendiin voitiin toteuttaa toiminnallisuus, jolla käyttöliittymä saatiin uuden käyttäjän lisäämisen jälkeen siirtymään automaattisesti roolit-välilehdelle, jossa uusi käyttäjä on valmiiksi valittuna hänen lisäämisekseen rooleihin.

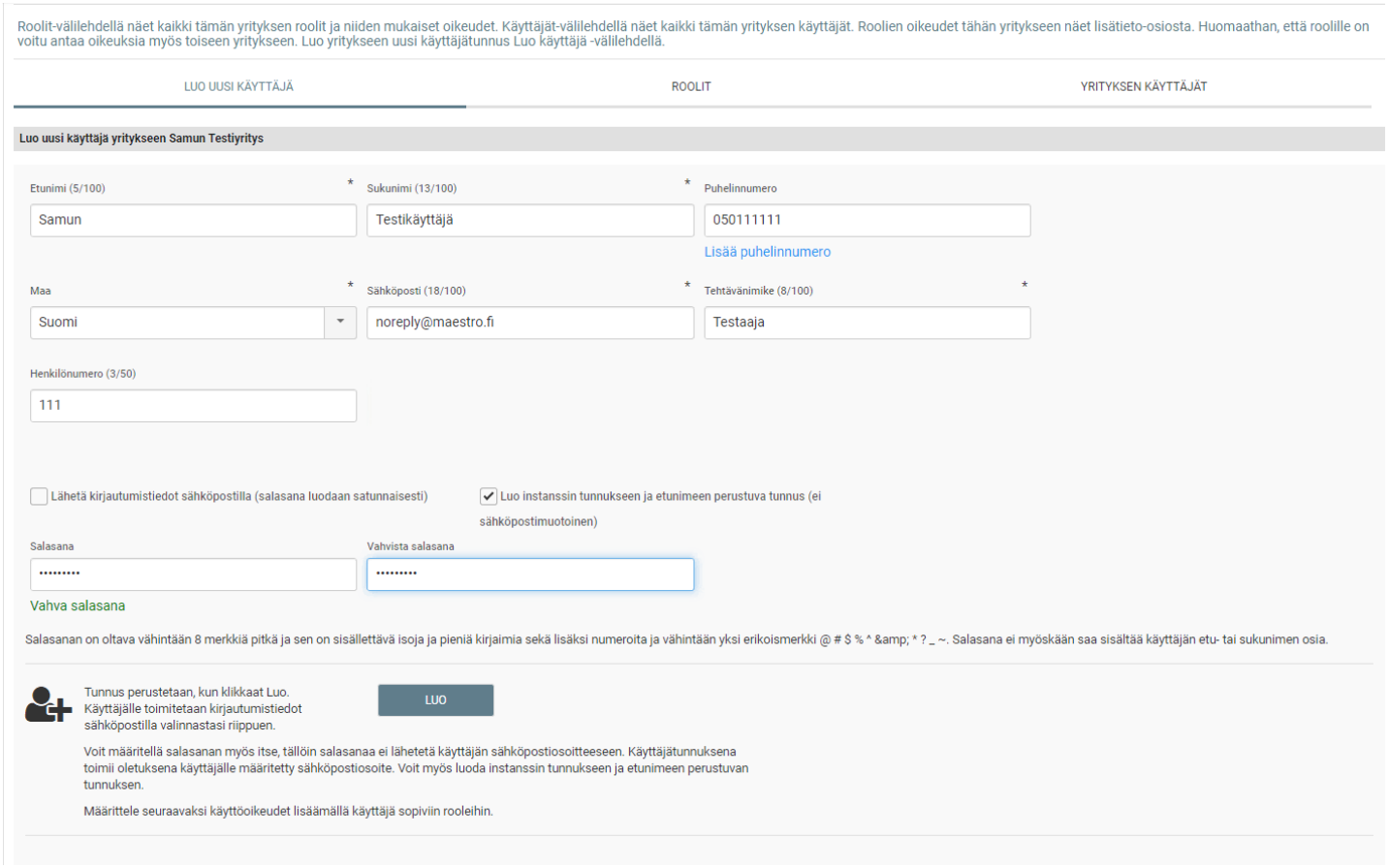
Näin käyttäjän luonnista yritykseen ja tarvittavien käyttöoikeuksien antamisesta uudelle käyttäjälle tehdään huomattavasti selkeämpää, ja saadaan ohjattua ylläpitäjät käyttämään ensisijaisesti roolipohjaisia käyttöoikeuksia uusien käyttäjien kohdalla, jolloin oikeuksien ylläpitokin on helpompaa. Kuvissa 31-34 on esitelty osa uuden käyttäjän luonnin frontend-logiikasta sekä valmiin välilehden käyttöliittymä.

```
// create new user save action
vm.onUserSaved = function () {
    var user = angular.copy(vm.newDetails);
    user.DisplayName = user.LastName + ' ' + user.FirstName;
    user.Name = user.DisplayName;
    // 'normal user' enum
    user.UserType = DEPLOYMENT.USERTYPES.NORMAL;
    user.ChangeType = vm.changeTypes.NEW;
    user.Roles = [];
    // add to common users role
    if (vm.commonUsersRoleIndex !== null) {
        vm.roleList[vm.commonUsersRoleIndex].Users.push(user);

        var role = vm.roleList[vm.commonUsersRoleIndex];
        user.Roles.push({
            Id: role.Id,
            RoleName: role.RoleName,
            ChangeType: vm.changeTypes.NEW,
            GroupType: role.GroupType
        });
    }

    //add to model and save changes
    vm.users.push(user);
    vm.users = _.sortBy(vm.users, ['DisplayName']);
    console.log(user);
    saveUsersAndRoles();
}
```

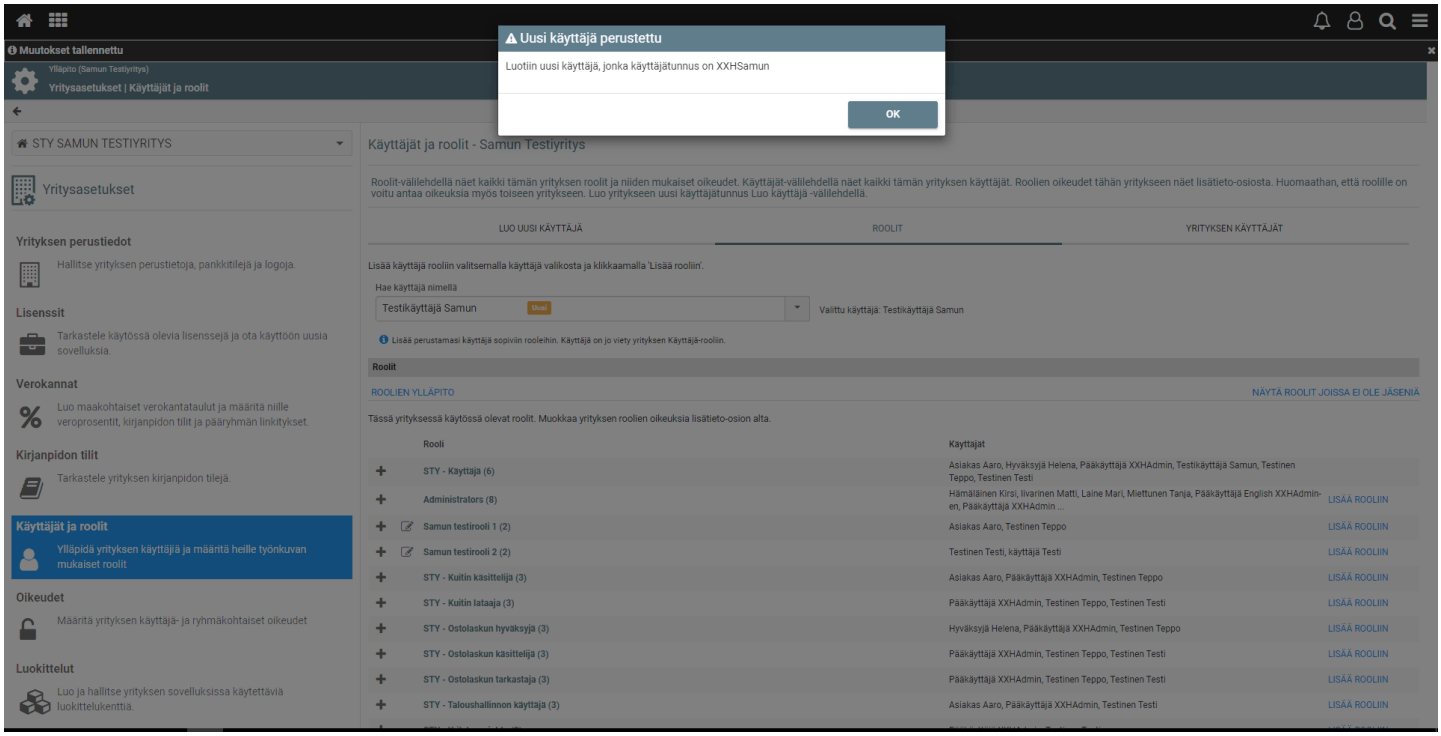
KUVA 31. Käsittelijän onUserSaved -metodi, jota kutsutaan, kun uuden käyttäjän luonnissa klikataan Luo-painiketta. Rakennetaan käyttäjän tietomalli lomakkeeseen annetuista tiedoista, lisätään käyttäjä yrityksen käyttäjärooliin ja käyttäjälistaan, ja tallennetaan muutokset

```
// if a new user was added, autoselect from dropdown, switch to roles tab and reset new user form
if (combinedData.NewestUser) {
    combinedData.NewestUser.Name = combinedData.NewestUser.LastName + ' ' + combinedData.NewestUser.FirstName;
    vm.activeTabIndex = 1;
    vm.onUserSelected(combinedData.NewestUser);
    vm.newDetails = { LanguageId: languageId, PhoneNumbers: [''], SendEmail: true };
    vm.emailValid = false;
    vm.searchResult = combinedData.NewestUser;
    vm.showNewUserInfo(combinedData.NewestUser);
}
```

KUVA 32. Osa backendistä palautuvan datan lopullisen muotoon käyttöliittymässä käsittelyä varten muokkaavasta createModels-metodista, jossa tarkistetaan, onko datassa tieto uudesta käyttäjästä. Jos on, valitaan uusi käyttäjä automaattisesti valikosta, nollataan uuden käyttäjän luontilomake, vaihdetaan roolit-välilehdelle ja näytetään uudesta käyttäjästä kertova modaali



KUVA 33. Uuden käyttöliittymän Luo uusi käyttäjä -välilehti



KUVA 34. Luo-nappia painettu kuvan 34 tilanteessa. Uusi käyttäjä on luotu, siirrytty roolit-välilehdelle ja valittu uusi käyttäjä automaattisesti rooleihin lisäämistä varten.

5.2.4 Valmiin käyttöliittymän testaus

Kun uusi käyttöliittymä oli saatu kokonaisuudessaan valmiiksi ja koodi läpäissyt koodikatselmoinnin, asennettiin se testiympäristöön sen testaamista varten ennen varsinaista tuotantokäyttöön asentamista.

Ohjelmistotestaus on prosessi, jonka tavoitteena on arvioida testattavan ohjelmiston toimintaa ja selvittää täyttääkö ohjelmisto sille määritetyt vaatimukset, ja löytää siitä mahdolliset viat osana laadunvarmistusta. Ohjelmistotestaus on tärkeä osa melkein mitä tahansa ohjelmistoprojektia, koska sen avulla voidaan välttää ohjelmistovirheiden päätyminen tuotantoon asti ja löytää ne jo aikaisessa vaiheessa kehitystä, jolloin niiden korjaaminen tulee usein halvemmaksi ja mahdolliset haittavaikutukset jäävät pienemmiksi. (Rajkumar, 2019)

Ohjelmistotestaukseen on kolme erilaista päälähestymistapaa, joita ovat White Box-, Black Box- ja Grey Box -testaus. White Box -testauksessa testitapaukset suunnitellaan järjestelmän sisäisen koodirakenteen perusteella, ja sisäistä perspektiiviä järjestelmään ja ohjelmointitaitoja hyödynnetään testauksessa. Black Box -testauksessa taas testataan itse käyttöliittymän/ohjelmiston oikeanlaista toimintaa loppukäyttäjän näkökulmasta ilman tietämystä sen varsinaisesta sisäisestä koodirakenteesta. Grey Box-testaus taas on edellisten yhdistelmä. (Rajkumar, 2019)

Uuden hallintakäyttöliittymän testaus hoidettiin Black Box -testauksena siten, että testaajalle laadittiin testausohjeet, joiden kautta käytiin läpi ja varmistettiin uuden käyttöliittymän jokaisen osa-alueen oikea toiminta. Nämä löytyvät välilehdittäin taulukoituna opinnäytetyön liitteistä. Näiden lisäksi määritettiin testattavaksi myös kaikki olemassa olleiden käyttöliittymien alueet, joihin uuden käyttöliittymän myötä tehdyt koodimuutokset voivat vaikuttaa, sekä vertaileva testaus yrityksen oletusroolien osalta vanhan käyttöliittymän kanssa.

Uudelle käyttöliittymälle kirjoitettiin myös joitakin yksikkötestejä (engl. unit tests), joita ajetaan järjestelmän buildin yhteydessä. Yksikkötesteillä testataan kooditasolla tietyn ohjelman lähdekoodin osan oikeanlaista toimintaa.

6 YHTEENVETO

Tässä luvussa käydään yhteenvetona läpi opinnäytetyössä saavutettuja tuloksia sekä toteutetun käyttöliittymän jatkokehitysmahdollisuuksia. Loppupohdinta-osiossa pohditaan tehtyä kehitystyötä tekijän ammatillisen kehityksen sekä toimeksiantajan saaman hyödyn näkökulmasta.

6.1 Työn tulokset ja jatkokehitysmahdollisuudet

Opinnäytetyössä päästiin sille määriteltyihin tavoitteisiin, ja uusi, entistä käyttäjäystävällisempi, mutta samalla MaestroNG-järjestelmän yritysten käyttöoikeuksien ylläpitoa huomattavasti entiseen verrattuna helpottava yritystason käyttäjien ja roolien hallintakäyttöliittymä saatiin suunniteltua, toteutettua ja vietyä tuotantokäyttöön asti asetetussa noin kahden kuukauden määräajassa. Käyttöliittymään saatiin myös tuotua kaikki opinnäytetyön puitteissa toteutettavaksi suunnitellut ominaisuudet. Toimeksiantaja on ollut uuteen toteutukseen tyytyväinen ja asiakkailtakin on jo saatu joitakin positiivisia kommentteja. Uusi käyttöliittymä on ollut tätä kirjoittaessa tuotantokäytössä vajaan kuukauden ajan.

Työ eteni alusta loppuun melko suoraviivaisesti, eikä toteutuksen aikana kohdattu mitään merkittäviä vaikeuksia, koska minulla oli Maestrolla käytössä olevista tekniikoista, itse MaestroNG-järjestelmästä ja vastaavista hallintakäyttöliittymien toteutuksista ennestään kokemusta aiemmin tekemieni projektien kautta, ja kehitystiimistä löytyy ammattilaisia, joista joku osaa kyllä auttaa asiassa kuin asiassa. Apuakin oli siis tarvittaessa hyvin tarjolla.

Joitakin jatkokehityssuunnitelmiakin käyttöliittymälle on olemassa, joista ensimmäiseksi toteutettavia voisivat olla esimerkiksi yrityksen käyttäjien käyttöoikeuksien ja roolien jäsenyyksien vienti raporttina Excel-taulukkolaskentaohjelmaan sekä uuden käyttäjän tietojen tuonti suoraan Active Directorysta. Myöhemmin voitaisiin toteuttaa esimerkiksi suora oikeuksien kopiointi käyttäjältä toiselle.

6.2 Loppupohdinta

MaestroNG:n käyttäjähallinnan uudistaminen uuden hallintakäyttöliittymän toteutuksen kautta oli opinnäytetyön aiheena mieluisa, ja se antoi minulle tekijänä jälleen lisää alakohtaisen kehitykseni kannalta tärkeää kokemusta ohjelmistokehityksestä ja sen osa-alueista kuten käyttäjälähtöisestä suunnittelusta, asiakastyöstä, käyttöoikeushallinnasta, ketterän ohjelmistokehityksen menetelmistä sekä web-sovelluskehityksestä. Opin opinnäytetyön aikana myös taas paljon lisää MaestroNG-järjestelmän rakenteesta ja arkkitehtuurista varsinkin sen käyttäjähallinnan osalta.

Jos toteutuksesta täytyisi nostaa esille haastavin osuus, oli se SQL-kyselyiden kehitys, joiden avulla uuteen käyttöliittymään saatiin näkyviin myös ne roolit ja käyttöoikeudet, joita vanhassa käyttöliittymässä ei ollut näkyvissä. Tämä oli melko pitkä ja paljon yritystä ja erehdystä vaatinut prosessi, joka kuitenkin opetti tekijälle paljon lisää monimutkaisemmasta SQL:stä.

Opinnäytetyön aikana kävi selväksi, ettei toimivan käyttäjähallinnan toteuttaminen MaestroNG:n kokoiselle järjestelmälle ole helppo tehtävä, vaan se vaatii runsaasti suunnittelua ja siinä täytyy ottaa huomioon todella paljon asioita. Myöskään alun perin toteutettua mallia ei ole helppo enää jälkikäteen lähteä kokonaisvaltaisesti muuttamaan, kun se on ollut tuhansien loppukäyttäjien käytössä jo vuosia. Uuden käyttöliittymän avulla käyttäjähallintaa saatiin kuitenkin selkeytettyä loppukäyttäjän näkökulmasta roimasti muuttamatta kuitenkaan liiaksi sen syvempiä kerroksia.

Uuden käyttöliittymän toteutus jo olemassa olleen vanhan käyttöliittymän päälle opetti tekijää myös lukemaan ja ymmärtämään toisten tekemää koodia paremmin, kun ennen uuden toteutuksen tekemistä täytyi tutustua kokonaisvaltaisesti olemassa olleen toteutuksen toimintaan lukemalla ja debuggaamalla koodia. Näin voitiin myös hyödyntää olemassa olleen koodin toimivia osia uuden käyttöliittymän toteutuksessa eikä pyörää tarvinnut joka asian osalta keksiä uudelleen.

Toimeksiantajan näkökulmasta opinnäytetyön tuloksena syntyi uusi, helpommin ylläpidettävä ja loppukäyttäjälle helpompikäyttöinen versio käyttöoikeuksien hallintakäyttöliittymästä, joka vaikuttaa suoraan positiivisesti suuren käyttäjämäärän käyttökokemukseen ja sitä kautta mahdollisesti myös asiakastyytyväisyyteen. Lisäksi kun oikeuksien raportoinnista tulee helpompaa ja käyttöoikeuksien hallinta muuttuu roolipohjaisemmaksi, on käyttäjien rooleja ja oikeuksia helpompi valvoa esimerkiksi laskutuksen näkökulmasta. Samalla valmistui myös taas yksi osa meneillään olevasta järjestelmän teknologia- ja käytettävyysuudistuksesta.

LÄHDELUETTELO

Atos SE. (2020). *Apprenda - SaaS Overview - SaaS Delivery Model.* Haettu 10. Maaliskuu 2020 osoitteesta https://apprenda.com/library/software-on-demand/saas-delivery-model/

Auth0, Inc. (2013). *Role-Based Access Control: Auth0 Docs*. Haettu 17. Maaliskuu 2020 osoitteesta https://auth0.com/docs/authorization/concepts/rbac

Branas, R. (2014). *AngularJS Essentials.* Packt Publishing. Haettu 19. Maaliskuu 2020

Ferraiolo, D.;Chandramouli, R.;Ferraiolo, D. F.;& Kuhn, D. (2007). *Role-based Access Control.* Artech House, Inc. Haettu 17. Maaliskuu 2020

Google. (ei pvm). *AngularJS Developer guide*. Haettu 19. Maaliskuu 2020 osoitteesta https://docs.angularjs.org/guide/

Imperva. (ei pvm). *What is Access Control List: Imperva*. Haettu 18. Maaliskuu 2020 osoitteesta https://www.imperva.com/learn/data-security/access-control-list-acl/

Martin, J. A. (21. Elokuu 2019). *What is access control? A key component of data security: CSO Online*. Haettu 18. Maaliskuu 2020 osoitteesta https://www.csoonline.com/article/3251714/what-is-access-control-a-key-component-of-data-security.html

Massachi, S. (ei pvm). *Role-Based Access Control - Computer Security - A brief look*. Haettu 7. Huhtikuu 2020 osoitteesta https://sites.google.com/site/cacsolin/role-based-access-control

Microsoft. (2019). *Get started with the .NET Framework: Microsoft Docs*. Haettu 23. 3 2020 osoitteesta https://docs.microsoft.com/fi-fi/dotnet/framework/get-started/

Microsoft. (Heinäkuu 2019). *Windows Support: Microsoft Corporation.* Haettu 12. Maaliskuu 2020 osoitteesta Silverlight End of Support: https://support.microsoft.com/en-gb/help/4511036/silverlight-end-of-support

Microsoft. (2020). *A tour of C# - C# guide: Microsoft Docs*. Haettu 23. Maaliskuu 2020 osoitteesta https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Microsoft. (ei pvm). *ASP.NET Web APIs: REST APIs with .NET and C#*. Haettu 23. Maaliskuu 2020 osoitteesta https://dotnet.microsoft.com/apps/aspnet/apis

Microsoft. (ei pvm). *What is ASP.NET: .NET*. Haettu 23. Maaliskuu 2020 osoitteesta https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet

Rajkumar, S. (13. Marraskuu 2019). *What is Software testing*. Haettu 6. Huhtikuu 2020 osoitteesta https://www.softwaretestingmaterial.com/software-testing/

Ruebbelke, L. (2015). *AngularJS In Action.* Manning Publications. Haettu 20. Maaliskuu 2020

Savytska, A. (6. Tammikuu 2020). *Medium: How to design access control system for Saas application*. Haettu 18. Maaliskuu 2020 osoitteesta https://medium.muz.li/how-to-design-access-control-system-for-saas-application-b6455c944186

Sitepoint. (5. Helmikuu 2020). *Do You Know What A REST API IS? - Sitepoint*. Haettu 24. Maaliskuu 2020 osoitteesta https://www.sitepoint.com/developers-rest-api/

sqlservertutorial.net. (ei pvm). *What is SQL Server*. Haettu 25. Maaliskuu 2020 osoitteesta https://www.sqlservertutorial.net/getting-started/what-is-sql-server/

Tanskanen, T. (18. Toukokuu 2016). *Käyttäjähallinta on riskienhallinnan ytimessä: Spellpoint*. Haettu 18. Maaliskuu 2020 osoitteesta https://spellpoint.fi/kayttajahallinta-on-riskienhallinnan-ytimessa/

LIITE 1: UUDEN KÄYTTÖLIITTYMÄN TESTITAPAUKSET

Roolit-välilehden testaus
Kun uusi käyttöliittymä avataan (Yritysasetukset -> Käyttäjät ja roolit, avautuu oletuksena välilehti 2 (Roolit)
Uudessa näkymässä näkyvät yrityksen omien oletusroolien lisäksi myös yrityksessä käytössä olevat instanssitason roolit ja niiden käyttöoikeudet sekä jäsenet
Sovelluskohtaisten käyttöoikeuksien hallintasivuille pääsee käyttöoikeuslistan linkeistä, valittu yritys pysyy valittuna myös niihin tultaessa
Käyttäjää ei voi suoraan lisätä yrityksen oletusrooliin "Käyttäjä" vaan siihen lisääminen tapahtuu automaattisesti, kun käyttäjä lisätään johonkin muuhun rooliin. Käyttäjää ei myöskään pysty lisäämään rooliin, jonka jäsen käyttäjä jo on.
"Roolien ylläpito" -painikkeen kautta pääsee instanssitason roolienhallintasivulle, jossa roolilista on automaattisesti suodatettuna valitulla olleella yrityksellä
"Näytä myös roolit, joissa ei ole jäseniä" -painikkeella voidaan piilottaa/näyttää roolit, joissa ei ole jäseniä
Kun roolilistan rivi laajennetaan, näytetään roolin jäsenet ja roolin käyttöoikeudet sovelluksittain
Roolin jäsenen voi poistaa roskakoripainikkeesta, poistossa on varmistuskysymys
Mahdolliset passivoidut roolin jäsenkäyttäjät on merkitty (Passivoitu) tekstillä, passivoidut käyttäjät eivät löydy "Hae käyttäjä nimellä" -hausta
Viimeisimmän vuorokauden aikana luodut roolin jäsenkäyttäjät on merkitty Uusi-tagilla
Ei vaadita erillistä tallennusta, vaan tallennus suoritetaan jokaisen lisäys/poisto-operaation jälkeen
Viimeisimpänä käsitelty laajennettu listan rivi pysyy laajennettuna tallennuksen jälkeen
Yrityksen pääkäyttäjä voi lisätä ja poistaa jäseniä vain yritystason rooleista
Instanssitason rooleista käyttäjien poistaminen ja käyttäjien lisääminen niihin vaatii järjestelmätason käyttöoikeuden "Oikeus ylläpitää kaikkia käyttäjäryhmiä"
Itse luotujen instanssitason roolien riveillä näytetään ikoni ja tooltip ”Itse luotu rooli”
Roolin käyttäjät ovat aakkosjärjestyksessä

Yrityksen käyttäjät-välilehden testaus
Yrityksen käyttäjät -välilehdellä näkyvät kaikki instanssin käyttäjät, joilla on käyttöoikeuksia valittuun yritykseen joko joidenkin roolien kautta tai yksilöllisesti määriteltynä
Kun rivi laajennetaan, näytetään lista rooleista, joissa käyttäjä on jäsenenä, käyttäjä voidaan myös poistaa ko. rooleista, poistossa on varmistuskysymys eikä se vaadi erillistä tallennusta
Laajennettu rivi säilyy laajennettuna poiston jälkeen
Laajennetun käyttäjärivin "Henkilökohtaiset oikeudet" -listassa näytetään sovelluksittain käyttäjälle henkilökohtaisesti annetut käyttöoikeudet (ei roolien kautta tulevia oikeuksia)
Instanssitason rooleista käyttäjien poistaminen vaatii järjestelmätason käyttöoikeuden "Oikeus ylläpitää kaikkia käyttäjäryhmiä"
Yrityksen pääkäyttäjä voi poistaa jäseniä vain yritystason rooleista
Sovelluskohtaisten käyttöoikeuksien hallintasivuille pääsee käyttöoikeuslistan linkeistä, valittu yritys pysyy valittuna myös niihin tultaessa
Mahdolliset passivoidut käyttäjät on merkitty (Passivoitu) tekstillä
Viimeisimmän vuorokauden aikana luodut roolin jäsenkäyttäjät on merkitty Uusi-tagilla

Luo uusi käyttäjä -välilehden testaus
Välilehdeltä voidaan luoda yritykseen uusi käyttäjä kolmella eri tavalla: o Salasana generoidaan satunnaisesti ja kirjautumistiedot lähetetään käyttäjän sähköpostiosoitteeseen (sähköpostimuotoiset käyttäjätunnukset) o Salasana voidaan asettaa manuaalisesti, jolloin sähköpostia ei lähetetä (sähköpostimuotoiset käyttäjätunnukset) o Voidaan luoda myös instanssin tunnukseen ja uuden käyttäjän etunimeen perustuva tunnus (vaaditaan salasanan asetus manuaalisesti ja ei lähetetä sähköpostia)
Pakollisia kenttiä ovat Etunimi, Sukunimi, Maa, Sähköposti, Tehtävänimike
Lähetä kirjautumistiedot sähköpostilla-, Luo instanssin tunnukseen ja etunimeen perustuva tunnus -valinnoista riippuen myös salasanakentät ovat pakollisia, piilotetaan oletuksena
Salasanan täytyy olla vähintään 8 merkkiä pitkä, sisältää pieniä ja isoja kirjaimia sekä lisäksi numeroita ja erikoismerkkejä, salasana- ja vahvista salasana -kenttien täytyy täsmätä
Sähköpostikenttä tarkistaa onko syötettyä sähköpostia käytössä kirjautumisnimenä jo jollain käyttäjällä, jonka kirjautumistapa on sähköpostiosoite, myös sähköpostin muoto on validoitu
Kun painetaan Luo-nappia, uusi käyttäjä perustetaan ja lisätään automaattisesti yrityksen "Käyttäjä" -rooliin sekä ohjataan automaattisesti roolit-välilehdelle, jossa juuri luotu käyttäjä on valmiiksi valittuna valikosta Uusi-tagin ja "Lisää luomasi käyttäjä rooleihin" -infotekstin kera. Käyttäjä voidaan suoraan lisätä haluttuihin rooleihin. Näytetään myös ilmoitus–popup, jossa kerrotaan, että luotiin uusi käyttäjä ja tämän kirjautumisnimi