TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Dan Siimson 178783IADB

OFFICE 365 AUTENTIMINE EESTI ELEKTROONILISE IDENTITEEDIGA

Bakalaureusetöö

Juhendaja: Nadežda Furs-Nižnikova MBA,
Raimo Seero BSc

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Dan Siimson

18.05.2020

Annotatsioon

Käesoleva lõputöö eesmärgiks oli luua rakenduse prototüüp, mis võimaldab Office 365 keskkonnale autentimist kasutades Eesti elektroonilist identiteeti.

Arendusprotsessi käigus luuakse rakenduse prototüübi serveri- ja kliendipoolne osa, antakse ülevaade süsteemi struktuurist ning kasutatud tööriistadest.

Ühtlasi tutvustatakse valminud süsteemi ning kuidas seda integreerida Office 365-le.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 8 peatükki, 29 joonist, 3 tabelit.

Abstract

Office 365 authentication with Estonian electronic identity

The goal of current thesis was to create a prototype of a web application, which allows using Estonian electronic identity as identity provider for Office 365.

There will be created both front-end and back-end applications as result of development process. Author will list all the tools, frameworks and technologies that will be used for development

Also gives author an overview of created system and how to make Office 365 work with it.

The thesis is in Estonian language and contains 34 pages of text, 8 chapters, 29 figures, 3 tables.

Lühendite ja mõistete sõnastik

Office 365	Pilvepõhine tarkvara ja teenuste lahendus, mis pakub ligipääsu nii Microsoft Office kontoritarkvarale kui ka muudele seotud teenustele.
Eesti elektrooniline identiteet	Elektrooniline identiteet (lühendatult ka eID) – andmete kogum, mis seob elektroonilises keskkonnas isiku tema füüsilise identiteediga. [1]
DNS	Domeeninimede süsteem. Teenus, mis muudab domeeninime IP-aadressiks või vastupidi.
API	*Application Programming Interface*. Rakendusliides või API-liides. Sisaldab reegleid, mille järgi eesrakendus tagarakendusega suhtleb.
HTML	*HyperText Markup Language.* Keel, milles märgendatakse veebilehti.
JSON	*JavaScript Object Notation.* Andmevahetusvorming, mis põhineb JavaScripti alamhulgal.
ORM	*Object Relationship Mapping.* Objekt-relatsiooniline kaardistamine.
React/ReactJS	JavaScript raamistik kasutajaliidese loomiseks.
Teek	*Library/Package*. Abifailid, mida saab programmis vajadusel kasutada
.NET Core	Raamistik tagarakenduse arendamiseks. Pakub funktsionaalsust luua veebirakendus.
JavaScript	Skriptikeel, mis võimaldab luua interaktiivseid veebilahendusi.
jQuery	JavaScripti teek, mis keskendub JavaScripti ning HTMLi vastastikusele sõltuvusele.
Sildiabiline	*Tag helper*. Pakub eesrakendusele. tuge kasutada otse tagarakenduse koodi läbi HTMLi.
Komponent	React komponent. Kasutatakse vaadete tükkideks jagamiseks.
REST	*Representational state transfer.* Tarkvaraarhitektuuri laad, mis seab paika veebirakenduse piirid.
SSL	*Secure Sockets Layer.* Turvasoklite kiht. Krüptograafiline protokoll, mis turvab võrgusuhtlust.

X509	Rahvusvahelise Telekommunikatsiooni Liidu poolt loodud standard avaliku võtme infrastruktuuris kasutatavatele sertifikaatidele.
WS-Federation	*Web Services Federation*. Protokoll, mille kaudu saab identsustõendi läbirääkimisi teha.
SAML	*Security Assertion Markup Language*. Standard autentimise ja autoriseerimise andmete vahetamiseks osapoolte vahel.
Microsoft Azure	Microsofti platvorm, mis pakub ohtralt erinevaid pilvandmetöötlusteenuseid.
Azure Active Directory	Microsofti pilvepõhine identiteedi- ja ligipääsuteenus, mis pakub autentimist ja ligipääsu ressurssidele.
Azure App Registration	Azure Active Directory teenus, mille kaudu saab rakendusele anda ligipääsu Microsoft Graph APIle.
Azure App Service	Microsofti poolt loodud rakenduste majutamise platvorm.
ID	*Identifier*. Andmebaasi olemi identifikaator.
GUID	*Globally unique identifier*. Globaalselt unikaalne identifikaator.
OCSP	*Online Certificate Status Protocol*. Sertifikaadi kehtivuse protokoll.
JWT	*Json Web Token*. Identsustõendi veebivorming.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänaseks päevaks oleme jõudnud tarkvaralahendustega punkti, kus paljud rakendused on kolinud pilve. Pilvelahendused on küll uudsed ja mugavad, kuid loovad IT-kaugetele kasutajatele peavalu oma kontode haldamise vaatest. Paljud suurte korporatsioonide poolt loodud lahendused kasutavad omapoolseid kasutajate identiteedilahendusi, kus kasutaja ja selle parool on seotud juba mõne muu nende poolt loodud platvormiga.

Inimesed tihtipeale kasutavad samu paroole erinevatel platvormidel ja pole nii käitudes kaitstud andmeleketest tekkitavate turvariskide eest. Kui kasutajaandmed on identsed erinevatel rakendustel, võivad nende kuritarvitajad pääseda ligi nendel olevatele kriitilistele andmetele mitmel pool.

Kasutades Eesti elektroonilist identiteeti tekib võimalus ühendada inimese elektrooniline identiteet tema füüsilise identiteediga ja asendada tavaline parooliga lahendus sertifikaatidega. Selline autentimisviis aitab kaasa nii mõnegi turvariski eemaldamisega.

Lõputöös käsitletava Microsoft Office 365 [2] puhul on tegu kontoritarkvaraga. Platvormi kaudu on võimalik pääseda ligi mitmetele erinevatele tekstitöötlus- ja haldusvahenditele nii pilves kui ka desktop lahenduste kaudu [3]. Eestis on antud teenus laialdaselt levinud paljude inimeste eelnevate Office toodete kogemuse tõttu. Seda kasutatakse nii koolides, firmades kui ka riigiasutustes.

Office 365 kasutab tavaolekus autentimiseks kontode baasil lahendust. Seda siis vaid rakenduse jaoks loodud või mõne kolmanda osapoole kontoga, mis on liidestatud rakendusega. Sellise viisi juures tõuseb päevakorda just eelnevalt kirjeldatud turvarisk.

Lõputöö käigus loodav rakenduse prototüüp tahab muuta tavalise sisselogimise turvalisemaks, viies Office 365 autentimisprotsessi kasutama Eesti elektroonilist identiteeti ja testida, kas sellist lahendust on üldse võimalik luua antud platvormile.

2 Probleemipüstitus

Käesolevas peatükis on juttu sellest, millist probleemi lõputööga lahendatakse. Samuti on ülevaade tingimustest, mida jälgima peab töö tegemisel. Autori roll lõputöös on otsida infot kuidas luua lahendus, lähtuda leitud tingimustest, analüüsida erinevaid tehnoloogiaid ja tööriistu ning luua rakenduse prototüüp koos ülevaatega.

2.1 Kontekst

Planeeritud teenus on veebirakendus Office 365 platvormile [2], mis pakub autentimist kolmanda osapoolena. Rakenduse isikutuvastus toimub läbi Eesti elektroonilise identiteedi [1]. Antud lahendusest on huvitatud mitmed kliendid, sealhulgas ka avalikust sektorist.

2.2 Tehnilised lähtetingimused

Planeeritud rakendus on sõltuv Office 365 teenusest [2] ja Azure platvormist [4]. Azure'i kasutatakse veebirakenduse majutamiseks võrgus ja rakendusele õiguste andmiseks. Samuti kasutab Office 365 Azure Active Directoryt [5] kasutajate hoidmiseks. Office 365 on platvorm, mille jaoks rakendust arendatakse.

2.3 *As-is* lahenduse probleemid

Praeguse lahenduse juures on põhiliseks probleemiks turvarisk. Kasutajaandmete sattumine valedesse kätesse tagab ligipääsu tervele platvormile. Samuti on tekitab antud lahendus lisatööd administraatoritele, kes peavad igapäevaselt tegelema kasutajate haldamisega ning sellest tulenevate probleemidega. Kasutajatel on soov kasutada turvalisemaid ja lihtsamaid teenuseid, kus ei pea kõiki erinevaid paroole meeles pidama.

3 Analüüs

Töö üheks eesmärgiks oli testida lahenduse võimalikkust. Seetõttu autori rollis ei tundunud mõistlik kirja panna otseselt rakedusele nõudeid, vaid mõelda kasutajalugudele, mis sobiksid kontekstiga. Kasutajalugudes on kaetud põhilised soovid, mis kasutajatel tekkida võiksid. Kasutajalood on kirjutatud nii tavakasutaja kui ka administraatori vaatest.

3.1 Kasutajalood

1. Kasutajana soovin kasutada Eesti elektroonilist identiteeti Office 365 autentimisel.
2. Kasutajana soovin, et rakendus oleks lihtsasti kasutatav
3. Administraatorina soovin hallata kasutajaid läbi mugava kasutajaliidese
4. Administraatorina soovin, et rakendus teeks mu igapäevaelu kergemaks

4 Tehnoloogiad

Käesolevas peatükis on juttu erinevatest tehnoloogiatest, mida rakenduse loomisel kasutatud on. Ülevaade antakse nii taga- kui ka eesrakenduse puhul kasutatavatest raamistikest ja teekidest.

4.1 Tagarakenduse tehnoloogiad

Tagarakenduse puhul on vajalik salvestada kontodega seotud andmeid, luua kasutajaid ja käsitleda Office 365st tulevat autentimise päringut. Sellest tulenevalt otsustas autor kasutada .NET Core [6] raamistikku koos Microsoft SQL Serveri [7], IdentityServer 4 [8], ASP.NET Core Identity [9] ja Entity Framework Corega [10]. .NET Core kaudu on lihtne luua kontrollereid, mille kaudu saab eesrakendus suhelda tagarakendusega. Samuti on .NET Core baasil rakendustes lisateekide paigaldamine väga lihtsaks tehtud, kasutades NuGet paketihaldustarkvara [11]. Tagarakenduse tehnoloogiavalik on tehtud põhiliselt autori enda kogemuse pealt ja arvestusega, et teiste raamistikke tundmaõppimine on ajakulukas. Samuti on lahendus valminud suures osas autori töökohas töö käigus, kus kasutatakse Microsofti poolt loodud tehnoloogiaid (sealhulgas valitud .NET Core raamistikku). Alternatiivina oleks saanud kasutada Javat koos Spring [12] raamistikuga, kuid antud raamistikes samale oskustasemele jõudmine oleks liialt ajakulukas ning ei mahuks töö ajaaknasse.

4.1.1 Microsoft SQL Server ja Entity Framework Core

Microsoft SQL Server on relatsiooniline andmebaas ja Entity Framework Core lisab sellele võimaluse kasutada koodist objekt-relatsioonilist kaardistamist [13]. Entity Framework Core kaudu on võimalik domeenimudeli järgi genereerida andmebaasi tabeleid, mis kiirendab arendusprotsessi [14]. Autor valis need teegid, kuna mõlema puhul on tegu Microsofti enda lahendusega (nagu .NET Core puhulgi) ning selle tõttu ei teki integreerimisega probleeme. Alternatiivina oleks võimalik olnud kasutada näiteks MySQLi [15], kuid sellele puudub ametlik ohjur ning kolmanda osapoole lahendus pole nii töökindel, kui olla võiks.

4.1.2 IdentityServer4, IdentityServer4.AspNetIdentity, WsFederation lisateek

IdentityServer4 on .NET Core raamistikule loodud teek, mis lisab võimaluse pakkuda kolmanda osapoolena autentimist teistele platvormidele [16]. IdentityServer4.AspNetIdentity lisab toe, et saaks kasutada süsteemis ASP.NET Identity poolt loodud kasutajaid [17]. IdentityServer4.WsFederation teek lisab IdentityServerile toe WS-Federation protokollile [18]. Office 365 toetab nii WS-Federation kui ka SAML protokolle identsustõendi vastu võtmiseks [15]. Rakenduses osutus valituks WS-Federation protokoll, kuna SAMLi protokolli rakendamiseks vajalik teek SAML2P [19] on tasuline ning ei anna märkimisväärseid eeliseid.

4.1.3 ASP.NET Core Identity

Autori projektis on oluline koht turvalisusel, seetõttu on mõistlik kasutada valmis teeke kriitiliste kohtade peal. Kasutajate jaoks mõeldud süsteem on üks nendest. ASP.NET Core Identity on teek, mis pakub lihtsat, kuid turvalist kasutajate loomist ning haldamist [9]. Samuti on IdentityServeril tugi kasutada ASP.NET Core Identity poolt loodud kasutajaid.

4.2 Eesrakenduse tehnoloogiad

Tagarakendusega suhtlemiseks on vajalik luua klientrakendus, mis muudab kontrollerist tulnud info kasutajale loetavaks. Antud alampeatükis on ülevaade tehnoloogiatest, mida on autor kasutanud rakenduse loomisel.

4.2.1 Kasutatud eesrakenduse raamistikud ja teegid

Eesrakenduse puhul on autor kasutanud mitmesuguseid erinevaid raamistikke ja teeke, mis muudavad arendusprotsessi kiiremaks. Kasutusel on eesrakenduse loomiseks Razor Pages [20], jQuery [21] ja React [22] ning sellele disaini loomiseks Bootstrap [23]. Lihtsamat lähenemist Razor Pagesi ning jQueryga on kasutatud vaadetel, kus pole dünaamilisusel olulist rolli kasutajakogemusele. Reacti baasil lahendus on kasutusel vaadetes, kus on vajalik andmete pidev ümberlaadimine ning väljakuvamine ilma lehte taaslaadimata. Eesrakenduste tehnoloogiavalikust on jQuery ja Razor Pages valitud selle baasil, et jQuery pakub tavalisele JavaScriptile juba valmiskirjutatud funktsioone, mille mis säästab aega ja Razor Pages on .NET Core raamistikku sisse ehitatud [20] ning neid

koos on lihtne kasutada. Samuti on mõlemad kasutusel Microsofti enda poolt .NET Core baasil loodud rakenduste näidistes [20]. Bootstrap on samuti kasutuses .NET Core näidistes [20] ning Bootstrapi dünaamiline pool on ülesehitatud jQueryga [24]. Reacti asemel olid valikus Vue.js [25] ja Angular [26] – kaks teist kõige populaarsemat eesrakenduste tehnoloogiat [27]. Angular ei osutunud valikuks, kuna autori kogemuse pealt pole tegu mugava ja kasutajasõbraliku raamistikuga, millega eesrakendust teha. Vue.js raamistikuga polnud autoril palju kogemusi, mis oleks teinud antud raamistiku kasutamise ajakulukaks ning pikendanud antud töö valmimisaega.

4.3 Muud vajalikud tehnoloogiad

4.3.1 Eesti elektrooniline identiteet

Eesti elektrooniline identiteet [1] on rakenduses vajalik, et viia Office 365 konto ja rakenduse andmebaasis olevad isikuandmed kokku isiku füüsilise identiteediga. Selle jaoks kasutatakse Mobiil-ID ja ID-Kaardi autentimissertifikaate, millelt loetakse isikuandmeid.

4.3.2 Microsoft Azure

Microsoft Azure on platvorm, mis pakub ohtralt erinevaid pilvandmetöötlusteenuseid [4]. Lõputöö käigus valminud rakenduses on see vajalik nii rakenduse majutamiseks kui ka vajalikele andmetele läbi Microsoft Graph API [28] ligipääsu tekitamiseks. Rakenduse majutamiseks on valitud Azure App Service [29], kuna arendusplatvorm, milles koodi kirjutatakse, pakub sellele tuge laadida rakendus kergelt ülesse.

4.3.3 Microsoft Graph API

Microsoft Graph API on teenus, mille kaudu pääseb ligi erinevate Microsofti teenuste andmetele läbi REST päringute. Lõputöö käigus on seda vajalik kasutada, et laadida Office 365 andmeid kohalikku rakendusse. Selleks, et lihtsustada Graph API päringuid, on kasutatud tagarakenduses teeki Microsoft.Graph [30].

4.3.4 Powershell

PowerShell [31] on Microsofti poolt loodud käsurea programm. Selle kaudu saab Office 365 platvormi seadistada. Vajalike käskude jooksutamiseks on vajalik lisamoodul

AzureAD [32]. PowerShellile puudub alternatiiv, kuna vajalikud teegid on loodud vaid sellele ja millegi muuga seda asendada pole võmalik.

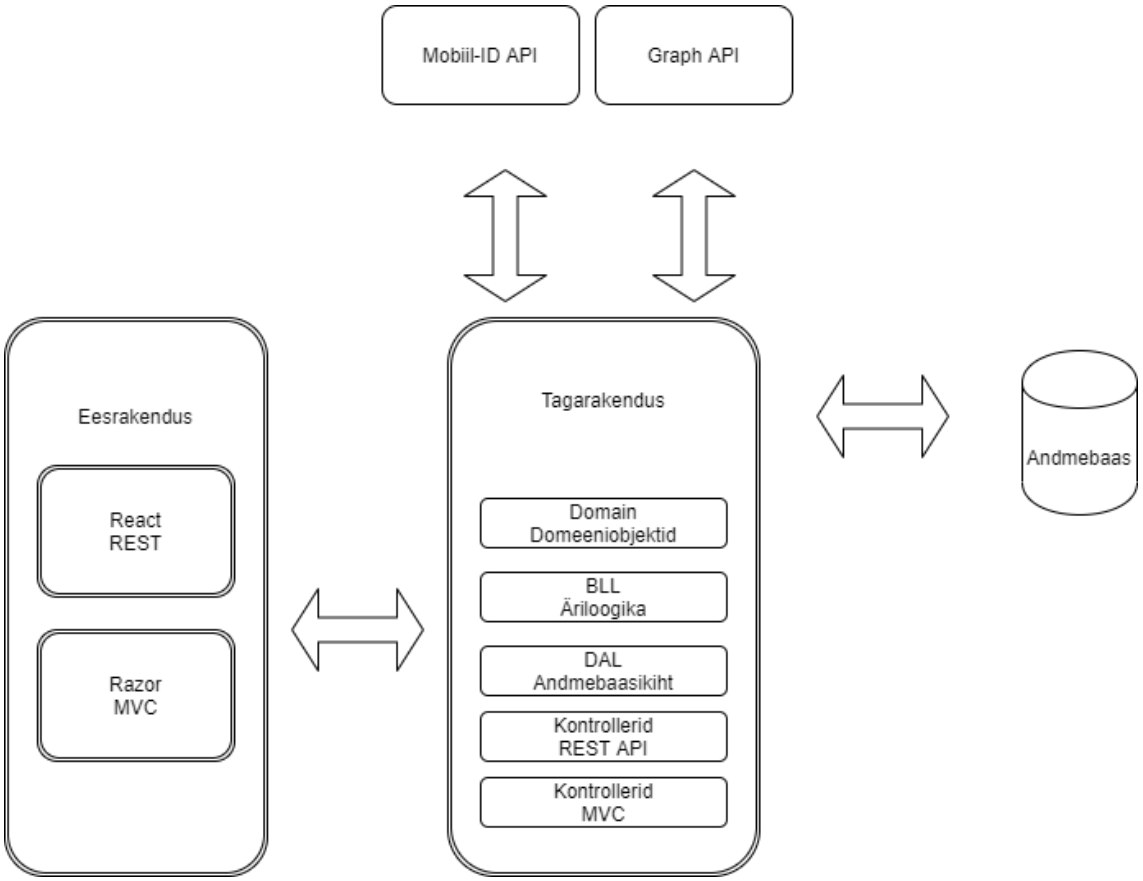
5 Tööriistad

Autori tööriistade valik on järgnev:

1. Jetbrains Rider – Tarkvara erinevate .NET baasil platvormide arendamiseks. Võimaldab kasutada nii C# kui ka erinevate eesrakenduste programmeerimiskeeli [33]. Pakub tuge Azure teenustele läbi Azure Toolkit teegi [34]. Autori rollis tundus antud tarkvara mõistlik valik, kuna mitmes õppeaines saadud kogemus oli positiivne ning palju kiirkäsud on juba meelde jäänud. Alternatiivina oleks saanud kasutada Microsofti Visual Studiot [35], kuid autoril polnud sellega väga palju kokkupuuteid ning uue integraalse programmeerimiskeskkonna õppimatundmine oleks ajakulukas.
2. Postman – Programm tagarakenduse APIde testimiseks [36]. Autor rollis osutus valikuks, kuna koolis on mitmes õppeaines kasutatud ning põhifunktsionaalsus selge. Samuti toetab Postman juba valmis tehtud päringute kollektsioone, mida pakub näiteks töös vajalik Graph API [37]. Alternatiivina oleks võinud autor kasutada Insomniat [38], kuid sellega kokkupuude puudub ning tundmatu asja ülesseadistamine on ajakulukas.
3. Chrome – Veebibrauser lahenduse testimiseks. Toetab igasuguseid lisateeke, mis muudavad arendusprotsessi lihtsamaks [39]. Autor valis antud brauseri, kuna ajapikku on sinna juba paigaldatud suurem kogus erinevaid teeke arendusprotsessi lihtsustamiseks. Alternatiivina oleks võinud autor kasutada Firefoxi [40], kuid sellele on tunduvalt vähem arendusega seotud lisateeke.
4. PowerShell – Käsurida, mis toetab igasuguseid lisafunktsioone nagu skriptimine ning erinevate teekide tugi. PowerShellile alternatiivid puuduvad, kuna arendusprotsessiks vajalik teek AzureAD on loodud vaid sellele.

6 Tehniline lahendus

Eelmised peatükid andsid ülevaate rakenduses kasutusel olevatest teekidest ja tehnoloogiatest, kuid mitte tehnilisest realitsioonist. Selles peatükis kirjeldab autor prototüübi erinevaid osasid nii tagarakenduse kui ka eesrakenduse vaatest ning vajalike teenuste konfiguratsiooni.



Joonis 1. Rakenduse arhitektuur (autori koostatud)

6.1 Rakenduse arhitektuur

Arhitektuur on tähtis pidevalt laienevate rakenduste puhul. Sellega on arvestatud ka lõputöö käigus loodud rakenduse puhul. Tagarakenduse puhul on lähtutud sellest mõttest, et jagada erinevate rollidega tükid võimalikult väikseks. See tagab selle, et nende

väljavahetamine ning uute juurde lisamine on lihtsam protsess. Samuti teeb see rakendusest vajalike asjade ülesleidmise kergemaks. Eesrakenduse puhul on lähtutud samast põhimõttest. Iga vaade on kokku pandud eraldiseisvatest väikestest osadest. Seda toetab ka Reacti ülesehitus, mida on kirjeldatud punktis 6.3.1.

6.2 Tagarakendus

Tagarakendus tegeleb äriloogika rakendamisega ning eesrakenduse toetamisega. Selle kaudu käib terve andmevoog, suhtlemine erinevate APIdega ning sissetulevate päringute käsitlemine.

6.2.1 Kontrollerid

Tagarakenduses on kasutusel kontrollerid, mis tegelevad klientrakenduse poolt tulevate päringutega. Tagarakendus koosneb neljast kontrollerist.

Mobiil-ID kontroller

Antud kontroller tegeleb kasutaja autentimisega läbi Mobiil-ID API [41]. Mobiil-ID APIga töö lihtsustamise jaoks on kirjutatud klass MIdClient (Lisa 1), mis tegeleb päringutega. Rakenduse prototüübis on see seadistatud töötama Mobiil-ID API testteenuse [42] vastu (Lisa 2). Klassil on kaks põhilist meetodit: InitializeAuthentication ja GetAuthenticationStatus. InitializeAuthentication saadab POST päringu Mobiil-ID autentimise URLile. Päringu sisuks on MIdRequest objekt (Lisa 3), mis on ehitatud vastavalt Mobiil-ID autentimispäringu nõuetele [43]. Päringu õnnestumise korral saab rakendus arvutada kinnituskoodi ning tagastada selle koos sessiooni ID-ga kontrollerile (Lisa 1 – InitializeAuthentication). GetAuthenticationStatus meetod teeb GET päringu Mobiil-ID sessiooni staatuse URLile koos eelmisest meetodist saadud sessiooni ID-ga. Kui kasutaja on autentimisprotsessi õnnestunult läbi viinud, tagastab API sertifikaadi koos tulemusega [44]. Muul juhul tagastab API tulemuse koos ebaõnnestumise põhjusega [44] (Lisa 1 – GetAuthenticationStatus).

Mobiil-ID kontrolleri meetodid on jaotatud kolmeks:

1. Index meetod – Index meetod kuvab välja avalehe, mis pakub nii Mobiil-ID kui ka ID-Kaardi baasil autentimist.

2. InitializeMIdAuth – Kasutaja saadab POST päringu läbi vormi, kus on kirjas tema telefoninumber ning isikukood (Joonis 17). Meetod teeb sellest MIdRequest objekti (Lisa 3) ning saadab selle omakorda edasi Mobiil-ID APIsse kasutades MIdClient klassi. Kui saadetud andmed on korrektsed, arvutab tagarakendus kinnituskoodi, tagastab selle klientrakendusele, mille see kasutajale välja kuvab (Joonis 2).

```
public async Task<IActionResult>
InitializeMIdAuth(MobileIdIndexModel model)
{
   if (ModelState.IsValid)
   {
       var req = new MIdRequest
       {
           RelyingPartyUUID =
MIdConstants.RelyingPartyUUID,
           RelyingPartyName =
MIdConstants.RelyingPartyName,
           DisplayText = "Auth to Office365",
           HashType = "SHA256",
           Language = "ENG",
           PhoneNumber = model.PhoneNumber,
           NationalIdentityNumber = model.IdCode,
           DisplayTextFormat = "GSM-7"
       };
       var res = await
_mIdClient.InitializeAuthentication(req);
       return View("AuthStatus",
           new MobileIdAuthStatusModel()
               {ConfirmCode = res.ConfirmCode, ReturnUrl
= model.ReturnUrl, SessionId = res.SessionId});
   }

   return View("Index", new MobileIdIndexModel());
}
```

Joonis 2. Mobiil-ID kontrolleri meetod InitializeMIdAuth (autori koostatud)

3. ValidateLogin – ValidateLogin meetod saadab Mobiil-ID APIsse päringu autentimise staatuse kohta. Kui autentimine õnnestus, saab rakendus Mobiil-ID sertifikaadi base64 sõne kujul ning teeb sellest X509 baasil sertifikaadi, loeb sellelt isikuandmed, otsib vastava kasutaja ülesse kohalikust andmebaasist kasutades AspNet.Identity teegi UserManager klassi instanssi [45] ning logib kasutaja sisse kasutades SignInManager klassi instanssi [46]. Õnnestunud

autentimise järel suunab rakendus kasutaja edasi WsFederation kontrollerisse, mis tegeleb edasisega. Kui autentimine ebaõnnestus, suunatakse kasutaja tagasi avalehele ning kuvatakse välja veateade. Meetodi täpsemat käitumist on näha joonisel 3.

```
public async Task<IActionResult> ValidateLogin(MobileIdValidateModel
model)
{
   var res = await
_mIdClient.GetAuthenticationStatus(model.SessionId);
   if (res.Result == MIdStatusResponse.ResultType.OK && res.Cert !=
null)
   {
       var p = Convert.FromBase64String(res.Cert);
       X509Certificate2 cert = new X509Certificate2(p);

       var subject = cert.Subject;
       var parts = subject.Split(",");

       var idcode = parts[0].Split("=")[1];
       var user = await _userManager.FindByIdCodeAsync(idcode);

       if (user != null)
       {
           await _signInManager.SignInAsync(user, new
AuthenticationProperties());
           return Redirect(model.ReturnUrl);
       }

       return Redirect("https://dansiimson.eu");
   }

   TempData["ErrorMessage"] = res.Result;
   return RedirectToAction("Index");
}
```

Joonis 3. Mobiil-ID kontrolleri meetod ValidateLogin (autori koostatud)

ID-kaardi kontroller

Antud kontroller tegeleb isikutuvastusega läbi ID-kaardi ning selle autentimissertifikaadi. Kontrolleril on ainult üks meetod InitializeAuth. InitializeAuth meetod saab kasutades `Request.Headers[“X-ARR-ClientCert”]` päringu päisest kätte App Service poolt küsitud kliendipoolse sertifikaadi [43]. Antud kontekstis on tegu ID-kaardil oleva autentimissertifikaadiga. Rakendus saab sertifikaadi kätte esialgu baitide kujul, millest rakendus teeb omakorda X509 sertifikaadi (Joonis 4). Valitud sertifikaadilt küsib Azure

App Service vastavalt konfiguratsioonile (konfiguratsiooni käsitlus punktis 6.4.2) isikutuvastuseks vajalikku PIN koodi. Kui isikutuvastusprotsess õnnestub, loeb rakendus sertifikaadilt isikuandmed ning logib kasutaja sisse eeldusel, et vastav kasutaja on süsteemis olemas. Sisselogitud kasutaja suunatakse edasi WsFederation kontrolleri poole, mis saadab identsustõendi Office 365-te. Ebaõnnestumise korral suunatakse kasutaja tagasi avalehele. Meetodi käitumist on näha joonisel 4.

```
public async Task<IActionResult> InitializeAuth(string returnUrl)
{
   var model = new IdCardStatusModel();
   string clientCertFromHeader = Request.Headers["X-ARR-ClientCert"];
   if (clientCertFromHeader != null)
   {
       byte[] bytes = Encoding.ASCII.GetBytes(clientCertFromHeader);
       var cert = new X509Certificate2(bytes);
       var subject = cert.Subject;
       var parts = subject.Split(",");

       var idcode = parts[0].Split("=")[1];
       var user = await _userManager.FindByIdCodeAsync(idcode);

       if (user != null)
       {
           await _signInManager.SignInAsync(user, new
AuthenticationProperties());
           return Redirect(returnUrl);
       }
   }
  return Redirect("https://dansiimson.eu");
}
```

Joonis 4. ID-kaardi kontrolleri meetod InitializeAuth (autori koostatud)

Kasutajatehalduse kontroller

Kasutajatehalduse kontroller on rakenduse väga tähtis lüli. Selle kaudu käib kasutajate üleüldine haldamine, importimine Active Directoryst ja muu seadistus. Kontroller kasutab REST tarkvaraarhitektuuri, mis tagastab andmeid ja tulemusi JSON kujul. Kontrolleri meetodid on välja toodud tabelis 1.

Meetod – päringu tüüp	**Kirjeldus**
GetUsers() – GET	Tagastab rakenduse andmebaasis olevad kontod
ImportUsers() – GET	Laeb Azure Active Directory'st Office 365 kontod rakenduse andmebaasi
UpdateIdCode(UpdateIdCodeDto dto) – POST	Uuendab kohaliku konto isikukoodi välja, vastavalt dto objektis olevale infole
Search(string q) – GET	Otsib kohalike kontode seast välja need, mis vastavad otsinguterminile q
GetTenantAdmins() – GET	Tagastab meiliaadressid, mida Azure Active Directoryst kohalikku andmebaasi ei laeta
AddAdmin(AddAdminDto dto) – POST	Lisab andmebaasi meiliaadressi, mida Azure Active Directory'st kasutajate importimisel rakenduse andmebaasi ei laeta
RemoveAdmin(RemoveAdminDto dto) – POST	Eemaldab rakenduse andmebaasist emaili aadressi, mida Azure Active Directory'st andmete laadimisel ignoreeritakse
AdminSearch(string q) – GET	Otsib ignoreeritavate emaili aadresside seast välja need, mis vastavad otsinguterminile q

Tabel 1. Kasutajahalduse kontrolleri meetodid (autori koostatud)

Kontrolleri töö abistamiseks on loodud klass AzureService, mis tegeleb Azure ja Graph API-poolsete teenustega. AzureService kaudu toimub Active Directory kasutajate importimine ja nende lisamine rakenduse andmebaasi. Joonisel 5 on näha täpsemalt kasutajate importi rakenduse andmebaasi.

```
public async Task<int> ImportAzureUsers()
        {
            var i = 0;
            var res = await _graphServiceClient.Users.Request().GetAsync();
            var tenantAdmins = await
_context.AzureTenantAdmins.ToListAsync();
            foreach (var user in res)
            {
                if (tenantAdmins.FirstOrDefault(x =>
                    x.Email == user.UserPrincipalName && x.AzureGuid ==
user.OnPremisesImmutableId) == null)
                {
                    var dbUser = await
_userManager.FindByEmailAsync(user.UserPrincipalName);
                    if (dbUser == null)
                    {
                        try
                        {
                            await FixUser(user);
                        }
                        catch (Exception e)
                        {
                            Console.WriteLine(e);
                            return -1;
                        }
                        i++;
                        var appUser = new AppUser()
                        {
                            UserName = user.UserPrincipalName,
                            Email = user.UserPrincipalName,
                            PasswordHash =
                                Guid.NewGuid().ToString(),
                            SubjectId =
Guid.Parse(user.OnPremisesImmutableId)
                        };
                        await _userManager.CreateAsync(appUser);
                    }
                }
            }
            return i;
        }
```

Joonis 5. Kasutajate importimine Azure Active Directoryst AzureService klassis (autori koostatud)

Meetod ImportAzureUsers küsib kõigepealt Graph API kliendilt kõik Azure Active Directorys olevad kontod. Peale seda küsib rakenduse andmebaasilt kõik meiliaadressid, mida importimisel vältima peab. Importimisel käib rakendus läbi tsükliga iga saadud Active Directorys oleva konto, võrdleb seda ignoreeritavate meiliaadressidega ja

kontrollib kas konto on juba rakenduses olemas. Kui kumbki tingimus paika ei pea, alustatakse konto sobilikuks muutmist.

```
private async Task FixUser(User user)
        {
            await ConvertUserToManaged(user);
            await UpdateImmutableId(user);
            await ConvertUserToFederated(user);
        }
```

Joonis 6. AzureService meetod FixUser (autori koostatud)

Joonisel 6 olev FixUser meetod kutsub esimesena välja meetodi ConvertUserToManaged. ConvertUserToManaged muudab Active Directorys oleva konto kõigepealt kasutama Office 365 poolt antud esialgset “onmicrosoft.com” meiliaadressi domeeni (Joonis 7) [47]. Nii saab rakendus muuta kasutaja ImmutableID välja, mis on rakenduse domeeniga seotud meiliaadressiga ainult loetavas staatuses.

```
private async Task ConvertUserToManaged(User user)
        {
            var domain = user.UserPrincipalName.Split("@")[1];
            if (domain != _configuration["AzureDomain"])
            {
                user.UserPrincipalName = user.UserPrincipalName.Split("@")[0]
+ _configuration["AzureDomain"];
                var req = _graphServiceClient.Users[user.Id].Request();
                await req.UpdateAsync(user);
            }
        }
```

Joonis 7. AzureService meetod ConvertUserToManaged (autori koostatud)

Järgmisena kutsub FixUser meetod välja meetodi UpdateImmutableId. Meetodiga UpdateImmutableId muudab rakendus kasutaja ImmutableID välja vastavusse Active Directorys oleva kasutaja ID väljaga (Joonis 8). See on vajalik, kuna uute kasutajate loomisel kasutatab autor suvalist ImmutableID väärtust.

```
private async Task UpdateImmutableId(User user)
        {
            user.OnPremisesImmutableId = user.Id;
            user.Mail = user.Mail.Split("@")[0] +
_configuration["AzureDomain"];
            var req = _graphServiceClient.Users[user.Id].Request();
            await req.UpdateAsync(user);
        }
```

Joonis 8. AzureService meetod UpdateImmutableId (autori koostatud)

Peale ImmutableID muutmist kutsutakse välja meetod ConvertUserToFederated. Sellega muudetakse kasutaja meiliaadress tagasi rakenduse domeeni omaks (Joonis 9). Kui eelmised sammud õnnestusid, luuakse rakenduse andmebaasi konto. Kuna AspNet.Identity teek vajab kasutaja loomiseks parooli, kasutab rakendus selleks genereeritud GUID väärtust (Joonis 5). Kasutajale pannakse kohe külge ka ImmutableID väli. Lõpuks tagastatakse kasutajate arv, mis rakenduse andmebaasi imporditi (Joonis 5).

```
private async Task ConvertUserToFederated(User user)
{
    user.UserPrincipalName = user.UserPrincipalName.Split("@")[0] + "@" +
_configuration["FederatedDomain"];
    var req = _graphServiceClient.Users[user.Id].Request();
    var updatedUser = user;
    await req.UpdateAsync(user);
}
```

Joonis 9. AzureService meetod ConvertUserToFederated (autori koostatud)

WSFederation kontroller

Antud kontrolleri puhul on tegu rakenduse ühe tähtsaima lüliga. Kontroller tuleb valmiskujul teegist IdentityServer4.WsFederation [48]. Kontroller oskab käsitleda sissetulevat autentimispäringut ja luua identsustõendit Office 365 jaoks. Esialgse sissetuleva autentimispäringu puhul suunab kontroller kasutaja logimise vaatele. Kui kasutaja läbib autentimisprotsessi õnnestunult, saadetakse ta tagasi WsFederation kontrolleri poole. Selle peale genereerib kontroller SAMLi baasil identsustõendi ning saadab selle Office 365-te. Antud teegi põhjalikumat seadistust käsitleb autor alampeatükis 6.2.4

6.2.2 Entity Framework seadistus

Entity Frameworki juures tuleb anda kõigepealt rakendusele andmebaasi ühendamiseks ühendussõne (Joonis 10). Selle järgi oskab ohjur (siinkohal Microsoft.EntityFrameworkCore.SqlServer [49]) luua ühenduse andmebaasiga.

```
"ConnectionStrings": {
 "AzureDbConnection": "Server=tcp:xxx.xxx.xxx.xx,1433;Database=authDb;User
ID=SA;Password=<P4SSWORD@>;Encrypt=true; TrustServerCertificate=True\n"
}
```

Joonis 10. Microsoft SQL Serveri ühendussõne Entity Frameworki jaoks (autori koostatud)

Andmebaasi kasutamiseks on vaja luua andmebaasi kontekst, mis peab pärinema Entity Frameworki DbContext klassist [50]. Rakenduses on kasutatud täpsemalt IdentityDbContext klassi, mis lisab toe ka kasutajate loomisele [51] (Joonis 8). IdentityDbContext vajab loomisel ka kasutaja klassi [51], milleks on rakenduse puhul AppUser. AppUser klassi loomist käsitleb autor punktis 6.2.3. Andmebaasi kontekstis tuleb samuti ära märkida tabelid, mida saab pärast kasutada ORMi kaudu. Seda tehakse DbSet klassi objekti [51] kaudu, kus märgitakse ära domeenimudeli klass, mille kohta tabel on (Joonis 11).

```
public class AppDbContext : IdentityDbContext<AppUser>
{
   public DbSet<AzureTenantAdmin> AzureTenantAdmins { get; set; }
   public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options)
   {
   }
}
```

Joonis 11. Entity Framework baasil andmebaasi kontekst (autori koostatud)

Peale andmebaasi konteksti loomist tuleb lisada see Startup.cs klassi, et rakendus seda kasutama hakkaks. (Joonis 12).

```
services.AddDbContext<AppDbContext>(options =>
   options.UseSqlServer(
       Configuration.GetConnectionString("AzureDbConnection")));
```

Joonis 12. Entity Framework seadistus Startup klassis (autori koostatud)

Pärast andmebaasi konteksti loomist ja Entity Frameworki seadistamist on võimalik genereerida andmebaasi tabeleid. Selle jaoks on vaja veel ühte teeki – Microsoft.EntityFrameCore.Design [52]. Selle toel saab kasutada käsurealt käske

```
dotnet ef migrations add InitialDbCreation --project DAL --startup-project
WebApp [52]
dotnet ef database update --project DAL --startup-project WebApp [52]
```

Esimene käsk loob andmebaasi migratsiooni, mille baasil oskab Entity Framework luua andmebaasi tabelid. Teine käsk käivitab protsessi, mis loob migratsiooni baasil tabelid andmebaasi.

Parameetriteks on vaja anda kaasa mõlemale käsule rakenduse andmebaasi konteksti klassiteek ning rakenduse käivituse klassiteek. Loodava rakenduse juures on andmebaasi konteksti klassiteegiks DAL ning käivitamiseks mõeldud klassiteek WebApp.

6.2.3 ASP.NET Core Identity seadistus

ASP.NET Core Identity kasutamiseks on vaja seadistada see eelnevalt ära Startup klassis. Teek kasutab tavaolekus kasutajate jaoks IdentityUser klassi [53], mis sellega kaasa tuleb. Tavaline IdentityUser ei sobi loodava prototüübi korrektseks toimimiseks, kuna sellel puuduvad vajalikud väljad Office 365-le identsustõendi genereerimiseks. Probleemi lahendamiseks on tehtud klass AppUser, mis pärineb IdentityUser klassist. Antud klassi on lisatud väljad kasutaja Azure Active Directory konto ImmutableID ja isikukoodi jaoks, mis on vajalikud identsustõendi genereerimiseks (Joonis 13).

```
public class AppUser : IdentityUser
{
   public Guid SubjectId { get; set; }
   public string IdCode { get; set; }
}
```

Joonis 13. AppUser klass (autori koostatud)

Selleks, et rakendus kasutaks loodud kasutaja klassi, tuleb see registreerida Startup.cs klassis (Joonis 14).

```
services.AddIdentity<AppUser, IdentityRole>(options =>
options.SignIn.RequireConfirmedAccount = false)
   .AddEntityFrameworkStores<AppDbContext>()
   .AddDefaultUI()
   .AddDefaultTokenProviders();
```

Joonis 14. ASP.NET Core Identity lisamine Startup.cs klassis (autori koostatud)

6.2.4 IdentityServer ja selle lisateekide seadistus

Selleks, et kohandada IdentityServer töötama korrektselt, on vaja seda seadistada.

Protsess koosneb mitmest etapist.

IdentityServeri seadistus Startup.cs failis. ConfigureServices meetodis tuleb lisada read, mis on välja toodud joonisel 15.

```
var identityserverbuilder = services.AddIdentityServer()
   .AddSigningCredential(cert)
   .AddInMemoryIdentityResources(Config.GetIdentityResources())
   .AddInMemoryClients(Config.GetClients())
   .AddAspNetIdentity<AppUser>();
IdentityServer4.WsFederation.WsFederationBuilderExtensions.AddWsFederation(id
entityserverbuilder);
```

Joonis 15. IdentityServer seadistus Startup.cs (autori koostatud)

Joonisel 15 näidatud igal koodireal on oma kindel ülesanne:

1. AddSigningCredential lisab sertifikaadi, mida kasutatakse identsustõendi allkirjastamiseks. Selleks sobib mistahes X509 sertifikaat [54].
2. AddInMemoryIdentityResources määrab ära, millist protokolli IdentityServer kasutama peab [54].
3. AddInMemoryClients märgistab ära, millised rakendused saavad teha päringuid IdentityServerile [54]. Office 365 rakenduse jaoks vajalikku klienti on näha joonisel 16.
4. AddAspNetIdentity lisab IdentityServerile toe kasutada olemasolevaid ASP.NET Core Identity kasutajaid [55].
5. IdentityServer4.WsFederation.WsFederationBuilderExtensions.AddWsFederation lisab IdentityServerile toe WSFederation protokollile.

```
new Client()
            {
             ClientId = "urn:federation:MicrosoftOnline",
             ProtocolType = ProtocolTypes.WsFederation,
             IdentityTokenLifetime = 360,

                    AllowedScopes = {"openid", "profile"},
                                RedirectUris =
             {"https://login.microsoftonline.com/login.srf"},
            }
```

Joonis 16. AddInMemoryClients juurde märgitud Office 365 klient (autori koostatud)

Configure meetodis tuleb lisada koodirida `app.UseIdentityServer()`. Antud rida lisab rakendusele IdentityServeri, mis sai eelmises meetodis ära seadistatud [54]. Muuta on vaja samuti WsFederation lisateegis olevat identsustõendi genereerimise meetodit, kus esialgu puuduvad Office 365-le vajalikud väljad. Office 365 ootab saadavast SAML identsustõendist välju, mis sisaldavad Azure Active Directorys oleva kasutaja ImmutableID väärtust ja meiliaadressi [56]. Vastuse genereerimisega

tegelevasse meetodisse CreateSubjectAsync on vaja teha vastavad muudatused, mida on näha joonisel 17. Tähtis on selle juures järgida täpselt Office 365 soovidele vastavad identsusväiteid [56].

Atribuut	**Selgitus**
NameID	Azure Active Directory kasutaja ImmutableID väli
IDPEmail	Azure Active Directory kasutaja meiliaadress

Tabel 2. Office 365 poolt identsustõendi oodatavad väljad (autori koostatud)

```
protected async Task<ClaimsIdentity>
CreateSubjectAsync(SignInValidationResult result)
{
 …
   var ImmutableID =
"http://schemas.microsoft.com/LiveID/Federation/2008/05/ImmutableID";
   var iId = new Claim(ImmutableID, result.User.GetAzureGuid());
   var email = new Claim(ClaimTypes.Email, result.User.GetAzureEmail());

nameid.Properties[Microsoft.IdentityModel.Tokens.Saml.ClaimProperties.SamlNam
eIdentifierFormat] = result.RelyingParty.SamlNameIdentifierFormat;

iId.Properties[Microsoft.IdentityModel.Tokens.Saml.ClaimProperties.SamlNameId
entifierFormat] = result.RelyingParty.SamlNameIdentifierFormat;

   var outboundClaims = new List<Claim> { iId, email };
```

Joonis 17. IdentityServer4.WsFederation CreateSubjectAsync klassi muudatusest (autori koostatud)

6.2.5 Reacti toe lisamine

React on tavaliselt ülesse ehitatud nii, et eesrakendus jookseb eraldi programmina. Kasutades lisateeke saab klientrakenduse tööle panna serveri poolel. Selle jaoks on valitud teegid ReactJS.NET [57] ja JavaScriptEngineSwitcher.ChakraCore [58]. ChakraCore puhul on tegu JavaScripti mootoriga, mis tekitab võimaluse rakendada JavaScripti rakendust serveri poolelt. ReactJS.NET lisab rakendusele mitmesugused sildiabilised, mida saab kasutada Reacti koodi väljakutsumiseks. Teekide seadistus on näidatud joonisel 18.

```
services.AddReact();
services.AddJsEngineSwitcher(options => options.DefaultEngineName =
ChakraCoreJsEngine.EngineName)
   .AddChakraCore();

app.UseReact(config =>
{
   config
       .SetReuseJavaScriptEngines(true)
       .SetLoadBabel(false)
       .SetLoadReact(false)
       .SetReactAppBuildPath("~/dist");
});
```

Joonis 18. ReactJS.NET seadistus Startup.cs klassis (autori koostatud)

6.3 Eesrakendus

Antud rakenduses on eesrakendus veebilehekülg, mis pakub kasutajatele võimalusi teha erinevaid toiminguid. Selle jaoks on loodud mitu erinevat kuva, millel on kindel roll. Peamised vaated on:

1. Autentemiseks mõeldud vaade - Selles vaates saab valida kasutaja erinevate Eesti elektroonilise identiteedi poolt pakutavate teenuste vahel ning alustada autentimisprotsessi.
2. Mobiil-ID kinnitusvaade - Selles vaates kuvatakse kasutajale välja Mobiil-ID päringust tulev kinnituskood.
3. Kasutajate haldusvaade - Selles vaates saab administraator importida kasutajaid Azure Active Directoryst ning hallata imporditud kontosid.
4. Ignoreeritavate kontode haldusvaade - Selles vaates saab administraator lisada kontosid, mida ei impordita süsteemi.

6.3.1 Eesrakenduse osad

Reacti ülesehitus

Reacti baasil vaated on üles ehitatud erinevatest komponentidest. Enamasti kasutatakse sisenemispunktina ühte peakomponenti. See omakorda kutsub välja alamkomponendid. Iga alamkomponent omab leheküljel kindlat rolli mõne tüki näitamisel. Komponendi kuvamiseks kutsutakse see välja kujul `<Komponent /> [59].` Reacti puhul saab alamkomponendile anda kaasa anda nii andmeid kui ka funktsioone. Neid saab lisada

lihtsalt `<Komponent funktsioon={this.funktsioon} andmed={this.state.andmed}>` [59]. Reactis on andmete hoidmiseks tehtud `this.state` [60] . Tegu on lihtsalt ühe JavaScripti objektiga, kuhu saab salvestada muutujaid ja palju muud. Kui alamkomponendile antakse midagi kaasa, saab sellele ligi `this.props` [60] objekti kaudu. `Props` on sarnane `state` objektile, kuid sisaldab ainult ülemkomponendilt tulnud välju ning on ainult loetavas olekus [61]. Seda tüüpi lähenemist on kasutatud kasutajatehalduse vaate puhul.

Autentimise vaade

Autentimise vaade on loodud Razor Pagesi baasil koos jQueryga, et kasutada Bootstrapi dünaamilise kuvamise võimalusi. Vaade koosneb kahest erinevast Bootstrapi disainielementide baasil loodud HTMLi vormist, mida saab valida nuppude kaudu. Vormide dünaamilise vahetamisega tegeleb jQuery, kasutades nuppude ID-sid vastavalt kuvamisega ning peitmisega (Joonis 19).

```
$('#mobileid-tab').on('click', function (e) {
 e.preventDefault()
 $(this).tab('show')
 document.getElementById('mId').style.display = "block";
 document.getElementById('idC').style.display = "none";
})
$('#idcard-tab').on('click', function (e) {
 e.preventDefault()
 $(this).tab('show')
 document.getElementById('mId').style.display = "none";
 document.getElementById('idC').style.display = "block";
});
```

Joonis 19. jQuery vormide väljakuvamisloogika (autori koostatud)

Valides mistahes vormi olemasolevatest, kuvatab jQuery vormi välja. Ülevaate andmiseks on kasutatud Mobiil-ID vormi (Joonis 20). Vorm sisaldab kahte välja, milleks on telefoninumber ning isikukood. Vormi valideerimine toimub tagarakenduse poolel. Kui midagi on valesti, kuvatakse jQueryga veateade puudulikult või valesti täidetud väljade kohta. Vormi ärasaatmisel tehakse päring tagarakenduse Mobiil-ID kontrolleri vastu, kutsudes välja meetodi InitializeMIDAuth kasutades `asp-action` [62] sildiabilist. Selline funktsionaalsus tuleb Razor Pages teegist, mis lisab tavalisele HTMLile mitmesugust .NET Core tuge.

```
<form method="post" asp-action="InitializeMIDAuth">
   <input type="hidden" asp-for="ReturnUrl"/>
   <fieldset>
       <div class="form-group">
           @if (ViewData.ModelState.Any(x => x.Value.Errors.Any()))
           {
               @Html.ValidationMessageFor(model => model.PhoneNumber,
"Isikukood on vajalik", new {@class = "text-danger"})
           }
           <input class="form-control border-0" placeholder="Isikukood" asp-
for="IdCode" autofocus>
       </div>
       <div class="form-group">
           @if (ViewData.ModelState.Any(x => x.Value.Errors.Any()))
           {
               @Html.ValidationMessageFor(model => model.PhoneNumber,
"Telefoninumber on vajalik", new {@class = "text-danger"})
           }
           <input class="form-control border-0" placeholder="Telefoninumber"
asp-for="PhoneNumber" autocomplete="off">
       </div>
       <div class="form-group text-center button-margin">
           <button class="btn auth-button" name="button" value="login"
type="submit">Autoriseeri</button>
       </div>
   </fieldset>
</form>
```

Joonis 20. Mobiil-ID vorm (autori koostatud)

Kasutajate haldamise vaade

Kasutajate haldamise vaade on kirjutatud Reactis vastavalt vajadusele andmeid dünaamiliselt välja kuvada ning uuendada.

Tehnilise poole pealt vaadatuna on seal kaks suurt komponenti, mis kuvavad välja kõik alamkomponendid. Komponentide kuvamiseks kasutatakse ReactJS.NET teegis olevaid sildiabilisi Razor Pages vaatel (Joonis 21).

```
@Html.React("Components.RootComponent", new {})
```

Joonis 21. Razor Pages poolt välja kutsutud Reacti komponent (autori koostatud)

Joonisel 21 olev rida kuvab välja põhikomponendi. Tegu on ReactJS.NET teegist tuleneva sildiabilisega, mis võtab argumendina sisse komponendi nime ja Reacti `state` objekti [63]. Siin on kasutatud tühja `state` objekti, kuna klientrakenduse poolel on tehtud funktsionaalsus, mis laeb andmed sisse APIst (Lisa 4).

Reacti vaated koosnevad kokku viiest komponendist: Search, Toaster, Users, TenantAdmins, UserManagement. UserManagement (Lisa 4) ja TenantAdmins (Lisa 5) on kaks suuremat vaadet ehk põhikomponendid, mis kutsuvad alamkomponente välja. Search komponent on lehe päis, mis sisaldab otsingut (Lisa 6). Otsing on seotud vastava tabeliga, mida komponent välja kuvab. Tabeleid on kahte tüüpi kasutajatehalduse juures. Üks tabel kuvab Azure Active Directoryst laetud kontosid (Users komponent) (Lisa 7). Teine tabel on kontode jaoks, mida ignoreerida importimisprotsessil (TenantAdmins komponendi sees olev tabel) (Lisa 5). Mõlema tabeli puhul tulevad andmed Search komponendist otsimisel. Toaster komponent (Lisa 8) on implementatsioon Bootstrapi Toast [64] elemendist. Tegu on teavitusribaga, mis näitab sõnumit mõne sündmuse toimumise kohta. Antud rakenduses kasutatakse seda impordi tulemuse välja kuvamiseks. Toaster komponent vajab väljakutsumisel kaasa parameetritena pealkirja ja sõnumit, mida kuvada. Samuti on vajalik kaasa anda ajaaken selle kohta, kui kaua teavitus nähtav on.

6.4 Seadistus

Antud projekti juures on vajalik seadistada nii Microsoft Azure pilveandmetöötluskeskkonda, Office 365 teenust kui ka domeeni.

6.4.1 Domeeni seadistus

Office 365 vajab, et rakenduste vaheline suhtlus käib üle SSL protokolli. Seetõttu on vajalik luua domeenile SSL sertifikaat. Sertifikaadi loomiseks on mitmesuguseid erinevaid valikuid – nii tasulisi kui ka tasuta. Autor valis tasuta viisi läbi Let’s Encrypt teenuse [65]. Let’s Encrypt pakub arendajatele võimalust luua SSL sertifikaat kolmeks kuuks tasuta [66]. Autor ei kirjeldanud SSLi sertifikaadi saamine protsessi täpsemalt, kuna tegu on suhteliselt mahuka protsessiga ning ei mahu lõputöö sisusse ära. Lisaks sellele on Windowsi baasil masinaga protsess keeruline, kuna tuleb kasutada virtuaalmasinat Linuxiga [67].

6.4.2 Azure seadistamine

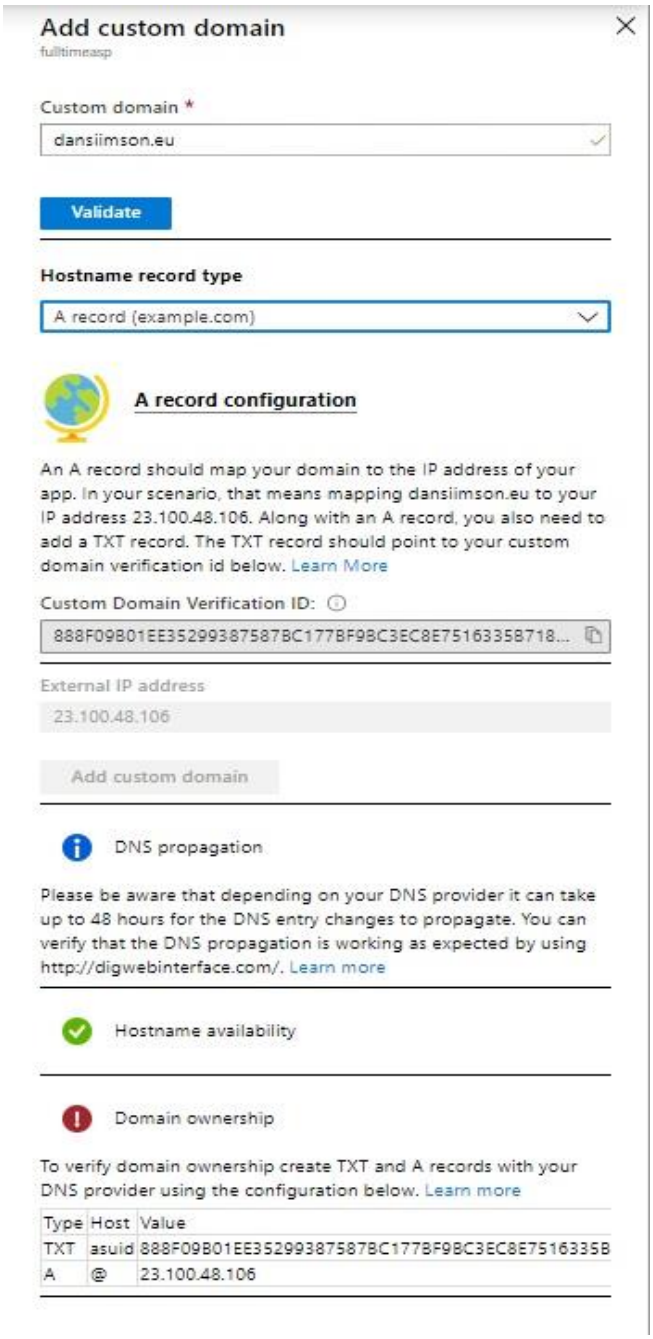
Azurel on rakenduse juures väga tähtis roll. Esiteks majutatakse rakendus Azure App Service teenuses ning teiseks kasutab Office 365 Azure Active Directoryt kõikide kasutajate salvestamiseks [68]. Lõputöö käigus valminud rakenduses kasutatakse Azure

teenuseid, et luua võimalus teha päringuid Microsoft Graph APIle. Selle API kaudu saadakse kätte Active Directorys olevate kasutajate andmed, et luua nende baasil lokaalsed kasutajad, millele saab lisada vajalikud väljad isikutuvastuseks.

Azure App Service seadistus

Azure App Service on veebirakenduse majutusteenus. Lõputöö tulemusena valminud rakenduse prototüüpi majutatakse just seal. Azure pakub mitmesuguseid erinevaid pakette rakenduse majutamiseks, vastavalt kliendi soovile ja vajadusele. Pakettidel on erinevad omadused, mis mõjutavad nii rakenduse jõudlust kui ka konfigureerimisvõimalusi [69]. Kuna lõputöö käigus valminud rakendusele on vajalikud nii domeen kui ka SSL sertifikaat, on vajalik vähemalt App Service B1 pakett [69].

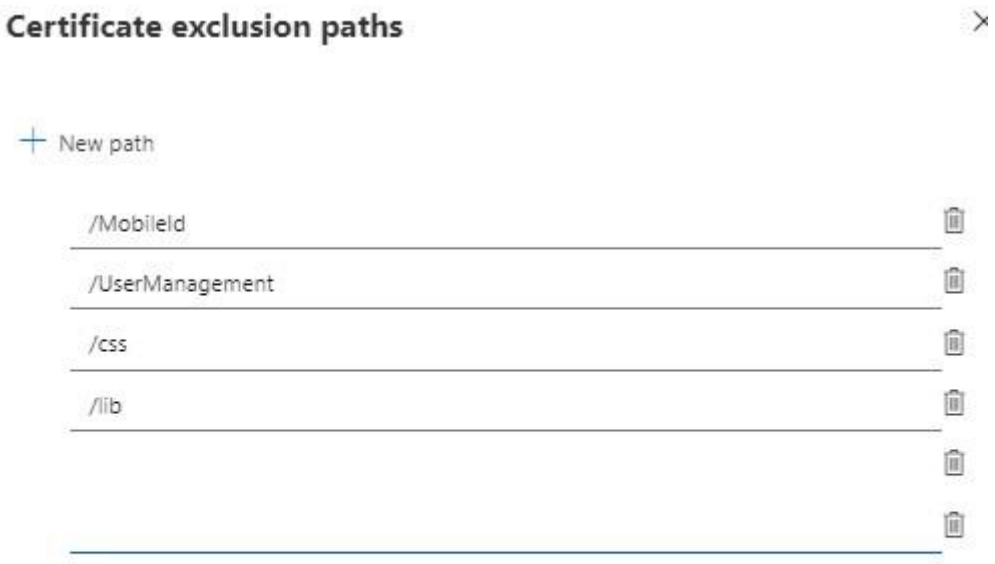
Office 365 nõuete kohaselt tuleb App Services olevale rakendusele lisada domeen. See protsess on tehtud väga lihtsaks. Tuleb valida App Service Settings alt *Custom domains* vaade, lisada domeen, valideerida, et see pole juba mõne teise App Service juures kasutusel, lisada DNSi seadistused domeenile ning lõpuks see ära kinnitada. Protsessi on näha täpsemalt joonisel 22.



Joonis 22. Azure App Service'le domeeni lisamine (autori koostatud)

App Servicele saab SSLi sertifikaati lisada *TLS/SSL settings* vaate kaudu. Kõigepealt tuleb sertifikaat ülesse laadida *Private Key Certificates (.pfx)* vaates. Peale seda saab minna eelmisele kuvale tagasi ning valida *Add TLS/SSL Binding.* Sealt tuleb valida lisatud domeen ning ülesse laetud sertifikaat. Peale seda töötab rakendus üle SSLi. App Service juures on tähtis selle korrektne seadistus vastavalt rakenduse nõuetele. Lõputöö käigus valminud rakendusele oli vaja lisada App Service konfiguratsiooni kaudu vajadus küsida kliendipoolset sertifikaati. Seda saab teha *Configuration* vaates,

kus tuleb valida *General Settings* ja märkida linnuke *Require incoming Certificate*. Azure poolelt on asi esilagu lahendatud nii, et App Service küsib iga tee pealt sertifikaati [70]. Selline lahendus ei sobi antud rakendusega. App Service pakub võimalust ka ära märkida kõik teed, kus pole see vajalik [70]. Seda on tehtud autori poolt loodud rakenduse jaoks. Suletuks on jäetud vaid ID-kaardi kontroller, kus see on vajalik kliendipoolse sertifikaadi kätte saamiseks. Täpsemalt on protsessi näha jooniselt 23, millel on märgitud kõik välistatud teed.



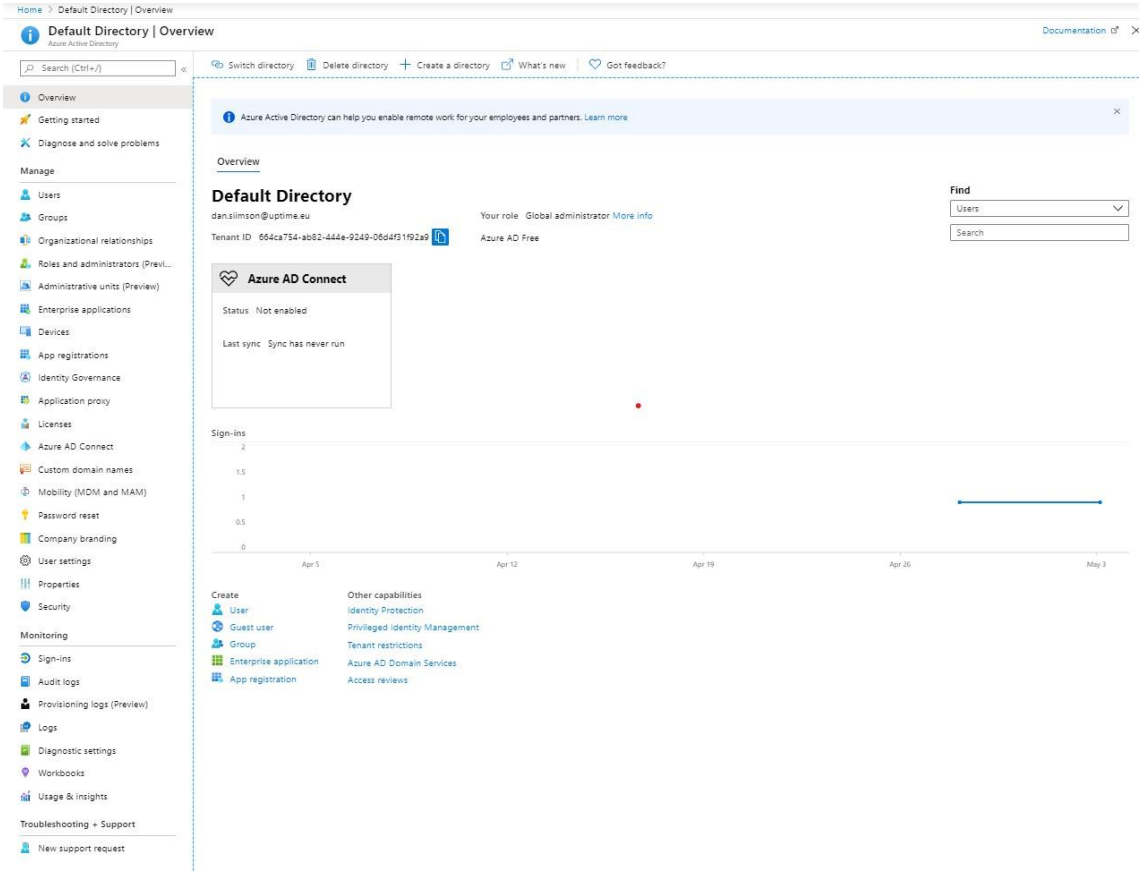
Joonis 23. Azure App Service kliendipoolse sertifikaadi välistamise teed (autori koostatud)

Toodangus pole Azure App Service sobilik rakenduse majutamiseks. Nimelt puudub tugi täispika sertifitseerimisahela loomiseks, mis on vajalik ID-kaardiga autentimise puhul [71]. Samuti ei saa seadistada serverit tegema ID-kaardile vajalikku OCSP sertifikaadi kehtivuskontrolli [72].

Azure App Registration seadistus

Azure App Registrationi kaudu saab anda rakendusele õigusi Microsoft Graph APIle ligipääsuks [73]. Graph API pakub Office 365 teenustele andmetele ligipääsu üle API [74]. Lõputöö käigus loodud rakendus vajab seda, et laadida kohalikku andmebaasi Azure Active Directorys olevad Office 365 kasutajakontod. Nende kontode olemasolu

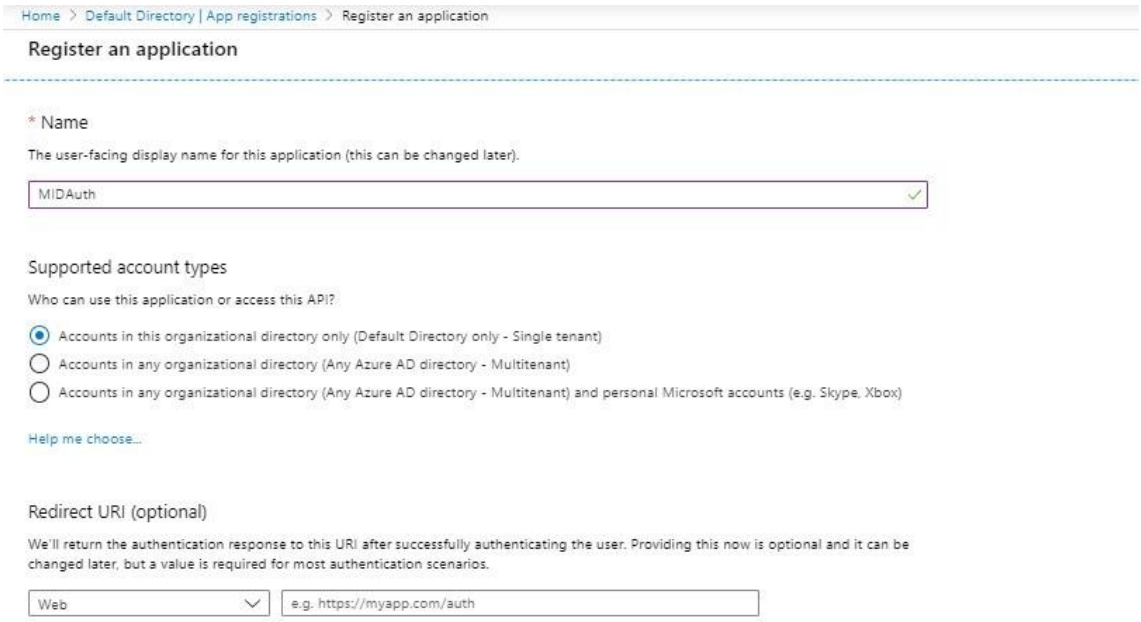
on vajalik identsustõendi genereerimiseks. Joonisel 24 on ülevaade Active Directory avalehest.



Joonis 24. Azure Active Directory ülevaatekuva. (autori koostatud)

App Registration protsessi on võimalik läbi viia kasutades Active Directory teenust (Joonis 25). Protsessi käigus luuakse rakendusele kindel ID, mida kasutatakse hiljem Graph APIle ligipääsuks [75]. Selleks, et loodud kliendi ID-d rakenduses kasutada, on vaja luua veel lisaks kliendile ka salasõna [75]. Seda saab teha juba loodud App Registrationi *Certificates & Secrets* vaates. Kui salasõna on loodud, saab rakenduse poolel anda need väärtused Graph API kliendile. Neid kasutades saab Graph API klient teha autentimispäringu Graph APIsse, kust õnnestumise korral tagastatakse JWT baasil identsustõend [76].

Kasutajate laadimiseks on vaja anda rakendusele App Registrationi kaudu õigus Users.ReadWrite.All [77]. See tagab, et rakendus saab nii lugeda kui ka muuta kasutajaid, mis asuvad Active Directorys.



Joonis 25. Uue App Registrationi lisamine (autori koostatud)

6.4.3 Office 365 seadistamine

Antud peatükk toetub ViewDS „Making Office 365 Work with an External SAML Identity Provider“ [78] ja Microsofti „Use a SAML 2.0 identity provider to implement single sign-on“ [79] artiklitele. Office 365 platvormi on vaja seadistada, et see usaldaks ja oskaks kasutada lõputöö käigus loodavat rakendust kui kolmanda osapoole identiteedipakkujat. Selle seadistamine käib läbi PowerShelli ning sellele paigaldatud AzureAD teegi. Teegi paigaldamiseks tuleb PowerShellis kävitada käsk `Install-Module AzureAD.`

Seadistamiseks on vaja administraatori kontoga logida PowerShelli kaudu sisse Azure Active Directory platvormi, mille kaudu Office 365 seadistus käib. See käib läbi käsu `Connect-MsolService [80]`. Peale käsku avaneb aken, kuhu saab Office 365 administraatori kontoga sisse logida.

Esimesena on lisada uus domeen Office 365 konto külge. Ilma selleta ei saa luua domeenipõhist autentimislahendust [81]. Selle jaoks on käsk `New-MsolDomain` [82]. Käsk vajab käivitamiseks kahte argumenti: Name ja Authentication [78]. Name argumendi külge käib domeen, mida registreerida üritatakse. Argument Authentication määrab selle domeeni autentimisviisi. Loodud rakenduse puhul kasutatakse väärtust Federated. See näitab, antud domeeniga seotud kontode autentimine hakkab toimuma läbi kolmanda osapoole.

Teisena tuleb eelmises punktis lisatud domeeni kohta saada domeeni DNSi seadistamiseks vajalikud väljad. Sellega kinnitatakse, et domeeni omanik on tõesti isik, kes proovib seda siduda. Seda tehakse käsuga `Get-MsolDomainVerificationDns` [83]. Käsk vajab kahte argumenti: DomainName ja Mode [78]. DomainName argument külge käib sama domeen, mis sai eelmise käsuga lisatud. Mode argumendi külge käib DNSi kinnitamise meetod.

```
Get-MsolDomainVerificationDns -DomainName dansiimson.eu -Mode DnsTxtRecord
```

Joonis 26. Office 365 domeeni DNSi seadetistamiseks vajalike väljade küsimine (autor ViewDS [78]), muudetud vastavalt autori rakendusele

Kolmandana on vaja kinnitada domeen käsuga `Confirm-MsolDomain [84]`. Käsk kontrollib, kas domeeni DNSil on eelmisest punktist saadud väärtused küljes. Joonisel 27 kasutuses oleva käsu parameetrid on toodud välja eraldi muutujatena tabelis 3 lugemise lihtsustamiseks.

```
Confirm-MsolDomain -DomainName $domainname -IssuerUri $issueruri -
FederationBrandName $domainname -LogOffUri $logoffuri -PassiveLogOnUri
$passivelogonuri -SigningCertificate $cert -PreferredAuthenticationProtocol
$protocol
```

Joonis 27. Office 365 domeeni kinnitamine (autor ViewDS [78])

Muutuja	**Selgitus**
$domainname = "dansiimson.eu"	Domeen, mida seotakse.
$logoffuri = "https://dansiimson.eu/account/logout"	Office 365 rakendusest väljalogimiseks mõeldud URL loodud platvormil.
$passivelogonuri = "https://dansiimson.eu/wsfederation"	Loodud platvormi autentimispäringule vastamise URL.
$cert = "XXXXXXXXXXXXXXXXXXX"	Päringu allkirjastamiseks mõeldud sertifikaat.
$issueruri = "https://dansiimson.eu"	Päringust tuleva identsustõendi allkirjastaja domeen.
$protocol = "WSFED"	Protokoll, mida Office 365 hakkab kasutama päringute loomisel. Rakenduses kasutatakse selleks WS-Federationit.

Tabel 3. Office 365 seadistamiseks vajalikud muutujad (autor ViewDS [78]), muudetud vastavalt autori rakendusele

Kui domeeninimi on kinnitatud, saab lõpuks seadistada vastavale domeenile kindla meetodi, mille kaudu hakkavad autenteerimispäringud käima. Käsk selleks on `Set-MsolDomainAuthentication` [85]. Antud käsk kasutab samu parameetreid, mida kasutati eelmises punktis ehk neid saab taaskasutada.

```
Set-MsolDomainAuthentication -DomainName $domainname -FederationBrandName
$domainname -Authentication Federated -IssuerUri $issueruri -LogOffUri
$logoffuri -PassiveLogOnUri $passivelogonuri -SigningCertificate $cert -
PreferredAuthenticationProtocol $protocol
```

Joonis 28. Office 365 autentemismeetodi määramine (autor ViewDS [78])

Protsessi saab verifitseerida järgnevate sammudega: Luua uus kasutaja käsuga `New-MsolUser` [86]. Käsk vajab parameetritena kaasa uue kasutaja kuvanime, eesnime, perenime, kasutusregiooni, ImmutableID välja ning meiliaadressi, millele konto luuakse. Office 365 dokumentatsiooni kohaselt pole ImmutableID ja UsageLocation

(kasutusregioon) parameetrid kohustuslikud, kuid lisatud domeeniga seotud kontot ei saa luua ilma ImmutableID-ta ja kasutajale Office tasulist litsentsi lisades ilma UsageLocationita, annab PowerShell veateate puuduliku kasutaja kohta. ImmutableID parameeter määrab ära seose Office 365 konto ja rakenduse konto vahel. Uue konto loomisel pole tähtis, mis ImmutableID väärtuseks pannakse, kuna rakenduse äriloogika muudab selle väärtuse samaks Active Directory kasutaja ID-ga.

```
New-MsolUser -UserPrincipalName testuser@dansiimson.eu -
DisplayName "Dan" -FirstName Test -LastName S -UsageLocation EE
-ImmutableID “07F1AD42-0CF8-41AD-82DC-8FD9B46F3755”
```

Joonis 29. Uue kasutaja loomine läbi PowerShelli (autori koostatud)

Peale uue konto loomist saab minna Office 365 portaali ning proovida sisselogimist uue loodud kasutajaga – eelmiste käskude õnnestumise korral suunatakse kasutaja edasi loodud rakendusse.

7 Järeldus

Antud rakenduse loomine oli keeruline. Otsides infot Office 365-le kolmanda osapoole autentimise kohta, leiab väga palju mittesobivaid lahendusi. Tihtipeale olid lahendused ehitatud üles nii, et Officega seotud Active Directorysse lisati juurde kontosid muudest võrgustikest, selle asemel, et siduda need olemasolevaga. Sellist tüüpi lahendus ei sobinud, kuna autor soovis saavutada lahendust, kus olemasolevad kontod oleks kasutatavad ning rakendus toimiks nende sidumisega, mitte uute loomisega.
Välja tuli ka mõni tehnoloogiavalikutes tehtud viga, näiteks Azure App Service valimine rakenduse majutamiseks. Selles juures puudus võimalus luua sertifitseerimisahelat ja kontrollida sertifikaadi kehtivust. Antud probleemi saaks lahendada näiteks kasutades Riigi Infosüsteemi Ameti poolt loodud lahendust TARA [87]. See viiks Eesti elektroonilise identiteediga autentimise üldse rakendusest välja ning autoril poleks vajadust tegeleda faktoritega, mis tekitavad turvariske ja vajavad teistsugust lähenemist. Nii jääks rakenduse ülesandeks ainult käsitleda päringuid, luua identsustõendeid ja pakkuda kasutajahaldust.
Rakenduse tulevase edasiarendamise juures saaks proovida ka veel mitmesuguseid alternatiivseid lahendusi Active Directory poolelt. Active Directory kasutajate juures on väljad, mida saab täita vabalt valitud infoga ning nii saab proovida salvestada kasutaja isikukood otse sinna. See kaotaks vajaduse rakenduse andmebaasi hoida sellisel kujul nagu ta praegu on – hetkel sisaldab kriitilist infot inimeste kohta (saab viia inimese emaili, mis tihtipeale on seotud tema nimega kokku isikukoodiga).

8 Kokkuvõte

Käesoleva töö eesmärgiks oli luua veebirakendus Office 365 autentimiseks Eesti elektroonilise identiteediga. Seega valmis veebirakenduse prototüüp, kus on võimalik tavakasutajal läbi viia isikutuvastus kasutades Eesti elektroonilist identiteeti.
Laiemas mõttes oli töö eesmärk ühendada inimese identiteet otseselt Office 365 platvormi kontoga ja lahendada turvalisuse probleeme.
Probleemipüstituses võeti arvesse ka projekti omapärasid ning kaaluti seejärel erinevaid võimalikke tehnoloogiaid töö sooritamiseks. Järgnevalt valiti sobivad raamistikud ja tehnoloogiad töö sooritamiseks. Autor puutus töö käigus kokku erinevate takistustega, millest mõned jäid ka loodud rakenduse prototüübi lõppversiooni. Töö lõpuks valmisid nii klientrakenduse kui ka veebiteenuse prototüübid. Töö tulemus reaalsuses ei sobi toodangusse, kuna on küljes turvariskid, tulenevalt majutusplatvormi valikust ja puudulikust rollihaldusest. Valides mõne muu teenusepakkuja, arendades rollihalduse lõpuni ning tehes mõned pisimuudatused, on loodud rakendus kasutuskõlblik ka toodangus.
Võib väita, et lõputööd koostades omandas autor juurde teadmisi klientrakenduse ja tagarakenduse ülesehitamise kohta, õppis kasutama erinevaid teeke ning sai palju teada autentimisprotsessi ja veebis kasutatava identiteedi kohta. Kuigi antud lõputöö sisaldab konkreetse eesmärgi jaoks üles ehitatud veebirakendust, on võimalik sarnase põhimõttega luua ka teistsuguseid veebirakendusi.

9 Kasutatud kirjandus

[1] Riigi Infosüsteemi Amet, „Elektrooniline identiteet eID,“ [Võrgumaterjal]. Saadaval: https://www.ria.ee/et/riigi-infosusteem/elektrooniline-identiteet-eid.html. [Kasutatud 02 05 2020].

[2] Microsoft Corporation, „Office 365 Login | Microsoft Office,“ [Võrgumaterjal]. Saadaval: https://www.office.com/. [Kasutatud 17 05 2020].

[3] Microsoft Corporation, „What is Microsoft 365? - Office 365,“ [Võrgumaterjal]. Saadaval: https://support.office.com/en-us/article/what-is-microsoft-365-847caf12-2589-452c-8aca-1c009797678b. [Kasutatud 17 05 2020].

[4] Microsoft Corporation, „Cloud Computing Services | Microsoft Azure,“ [Võrgumaterjal]. Saadaval: https://azure.microsoft.com/en-gb/. [Kasutatud 14 05 2020].

[5] Microsoft Corporation, „Azure Active Directory | Microsoft Azure,“ [Võrgumaterjal]. Saadaval: https://azure.microsoft.com/en-us/services/active-directory/. [Kasutatud 17 05 2020].

[6] Microsoft Corporation, „.NET,“ [Võrgumaterjal]. Saadaval: https://dot.net. [Kasutatud 20 04 2020].

[7] Microsoft Corporation, „SQL Server 2019,“ [Võrgumaterjal]. Saadaval: https://www.microsoft.com/en-us/sql-server/sql-server-2019. [Kasutatud 02 05 2020].

[8] B. Allen ja D. Baiern, „Welcome to IdentityServer4,“ [Võrgumaterjal]. Saadaval: http://docs.identityserver.io/en/release/. [Kasutatud 02 05 2020].

[9] R. Anderson, „Introduction to Identity on ASP.NET Core,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio. [Kasutatud 02 05 2020].

[10] Microsoft Corporation, „Overview of Entity Framework Core,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/ef/core/. [Kasutatud 02 05 2020].

[11] Microsoft Corporation, „NuGet Gallery | Home,“ [Võrgumaterjal]. Saadaval: https://www.nuget.org/. [Kasutatud 02 05 2020].

[12] VMWare Inc., „Spring | Home,“ [Võrgumaterjal]. Saadaval: https://spring.io/. [Kasutatud 02 05 2020].

[13] Learn Entity Framework Core, „Entity Framework Core Documentation And Tutorials | Learn Entity Framework Core,“ [Võrgumaterjal]. Saadaval: https://www.learnentityframeworkcore.com/#what-is-entity-framework-core. [Kasutatud 14 05 2020].

[14] Microsoft Corporation, „Getting Started - EF Core | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/ef/core/get-started/?tabs=netcore-cli#create-the-database. [Kasutatud 14 05 2020].

[15] Oracle Corporation, „MySQL,“ [Võrgumaterjal]. Saadaval: https://www.mysql.com/. [Kasutatud 14 05 2020].

[16] B. Allen ja D. Baiern, „The Big Picture — IdentityServer4 1.0.0 documentation,“ [Võrgumaterjal]. Saadaval:

https://identityserver4.readthedocs.io/en/latest/intro/big_picture.html#how-identityserver4-can-help.

[17] B. Allen ja D. Baiern, „ASP.NET Identity Support,“ 02 05 2020. [Võrgumaterjal]. Saadaval: http://docs.identityserver.io/en/3.1.0/reference/aspnet_identity.html.

[18] B. Allen, D. Baiern ja A. Kowalew, „IdentityServer/IdentityServer4.WsFederation: Sample for implementing WS-Federation IdP support for IdentityServer4,“ [Võrgumaterjal]. Saadaval: https://github.com/IdentityServer/IdentityServer4.WsFederation.

[19] Rock Solid Knowledge, „SAML2P Service Provider Identity Provider OAuth OIDC,“ [Võrgumaterjal]. Saadaval: https://www.identityserver.com/products/saml2p. [Kasutatud 02 05 2020].

[20] Microsoft Corporation, „Introduction to Razor Pages in ASP.NET Core,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio#validation. [Kasutatud 02 05 2020].

[21] The jQuery Foundation, „jQuery,“ 02 05 2020. [Võrgumaterjal]. Saadaval: https://jquery.com/.

[22] Facebook Inc, „React – A JavaScript library for building user interfaces,“ [Võrgumaterjal]. Saadaval: https://reactjs.org/. [Kasutatud 20 04 2020].

[23] Bootstrap Authors, „Bootstrap · The most popular HTML, CSS, and JS library in the world.,“ [Võrgumaterjal]. Saadaval: https://getbootstrap.com/. [Kasutatud 20 04 2020].

[24] Bootstrap Authors, „JavaScript · Bootstrap v4.5,“ [Võrgumaterjal]. Saadaval: https://getbootstrap.com/docs/4.5/getting-started/javascript/#dependencies. [Kasutatud 14 05 2020].

[25] E. You, „Vue.js,“ [Võrgumaterjal]. Saadaval: https://vuejs.org/. [Kasutatud 02 05 2020].

[26] Google, „Angular,“ [Võrgumaterjal]. Saadaval: https://angular.io/. [Kasutatud 02 05 2020].

[27] T. Krotoff, „FrontendFrameworksPopularity.md,“ [Võrgumaterjal]. Saadaval: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190. [Kasutatud 02 05 2020].

[28] Microsoft Corporation, „Overview of Microsoft Graph - Microsoft Graph | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/overview. [Kasutatud 14 05 2020].

[29] Microsoft Corporation, „Azure App Service – app hosting | Microsoft Azure,“ [Võrgumaterjal]. Saadaval: https://azure.microsoft.com/en-us/services/app-service/. [Kasutatud 14 05 2020].

[30] Microsoft Corporation, „NuGet Gallery | Microsoft.Graph 3.5.0,“ [Võrgumaterjal]. Saadaval: https://www.nuget.org/packages/Microsoft.Graph/. [Kasutatud 14 05 2020].

[31] Microsoft Corporation, „PowerShell,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7. [Kasutatud 02 05 2020].

[32] Microsoft Corporation, „AzureAD | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/azuread/?view=azureadps-2.0. [Kasutatud 02 05 2020].

[33] JetBrains s.r.o, „JetBrains Rider,“ [Võrgumaterjal]. Saadaval: https://jetbrains.com/rider/. [Kasutatud 20 04 2020].

[34] JetBrains s.r.o, „Azure Toolkit for Rider - Plugins | JetBrains,“ [Võrgumaterjal]. Saadaval: https://plugins.jetbrains.com/plugin/11220-azure-toolkit-for-rider. [Kasutatud 14 05 2020].

[35] Microsoft Corporation, „Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio,“ [Võrgumaterjal]. Saadaval: https://visualstudio.microsoft.com/. [Kasutatud 17 05 2020].

[36] Postman, „Postman,“ [Võrgumaterjal]. Saadaval: https://getpostman.com/. [Kasutatud 20 04 2020].

[37] Microsoft Corporation, „Use Postman with the Microsoft Graph API - Microsoft Graph | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/use-postman. [Kasutatud 17 05 2020].

[38] Kong Inc, „Insomnia | API Design Platform and REST Client,“ [Võrgumaterjal]. Saadaval: https://insomnia.rest/. [Kasutatud 17 05 2020].

[39] Google, „Google Chrome'i veebibrauser,“ [Võrgumaterjal]. Saadaval: https://www.google.com/intl/et/chrome/. [Kasutatud 14 05 2020].

[40] Mozilla Corporation, „Firefoxi allalaadimine — Tasuta veebilehitseja — Mozilla,“ [Võrgumaterjal]. Saadaval: https://www.mozilla.org/et/firefox/new/. [Kasutatud 17 05 2020].

[41] SK ID AS, „SK - Teenused - Mobiil-ID - Tehniline lisainfo (MID_REST_API),“ [Võrgumaterjal]. Saadaval: https://www.skidsolutions.eu/teenused/mobiil-id/tehniline-lisainfo-mid-rest-api. [Kasutatud 14 05 2020].

[42] Riigi Infosüsteemi Amet, „Environment technical parameters · SK-EID/MID Wiki - DEMO,“ [Võrgumaterjal]. Saadaval: https://github.com/SK-EID/MID/wiki/Environment-technical-parameters#demo. [Kasutatud 17 05 2020].

[43] Riigi Infosüsteemi Amet, „SK-EID/MID - 3.2.3 Request Parameters,“ [Võrgumaterjal]. Saadaval: https://github.com/SK-EID/MID#323-request-parameters. [Kasutatud 17 05 2020].

[44] Riigi Infosüsteemi Amet, „SK-EID/MID - Example Responses,“ [Võrgumaterjal]. Saadaval: https://github.com/SK-EID/MID#337-example-responses. [Kasutatud 17 05 2020].

[45] Microsoft Corporation, „UserManager<TUser> Class (Microsoft.AspNetCore.Identity) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.usermanager-1?view=aspnetcore-3.1. [Kasutatud 14 05 2020].

[46] Microsoft Corporation, „SignInManager<TUser> Class (Microsoft.AspNetCore.Identity) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.signinmanager-1?view=aspnetcore-3.1. [Kasutatud 14 05 2020].

[47] Microsoft Corporation, „Domains FAQ - Microsoft 365 admin | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/microsoft-365/admin/setup/domains-faq?view=o365-worldwide#why-do-i-have-an-onmicrosoftcom-domain. [Kasutatud 15 05 2020].

[48] B. Allen, D. Baiern ja A. Kowalew, „IdentityServer4.WsFederation/WsFederationController.cs at net461 · IdentityServer/IdentityServer4.WsFederation,“ [Võrgumaterjal]. Saadaval: https://github.com/IdentityServer/IdentityServer4.WsFederation/blob/net461/src/IdentityServer4.WsFederation/WsFederation/WsFederationController.cs. [Kasutatud 14 05 2020].

[49] Microsoft Corporation, „NuGet Gallery | Microsoft.EntityFrameworkCore.SqlServer 3.1.4,“ [Võrgumaterjal]. Saadaval: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer/. [Kasutatud 14 05 2020].

[50] Microsoft Corporation, „DbContext Class (Microsoft.EntityFrameworkCore) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext?view=efcore-3.1. [Kasutatud 14 05 2020].

[51] Microsoft Corporation, „IdentityDbContext Class (Microsoft.AspNetCore.Identity.EntityFrameworkCore) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.entityframeworkcore.identitydbcontext?view=aspnetcore-3.0. [Kasutatud 14 05 2020].

[52] Microsoft Corporation, „Getting Started - EF Core - Create the database | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/ef/core/get-started/?tabs=netcore-cli#create-the-database. [Kasutatud 14 05 2020].

[53] MIcrosoft Corporation, „IdentityUser Class (Microsoft.AspNetCore.Identity.EntityFrameworkCore) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.entityframeworkcore.identityuser?view=aspnetcore-1.1. [Kasutatud 14 05 2020].

[54] B. Allen ja D. Baiern, „Startup — IdentityServer4 1.0.0 documentation,“ [Võrgumaterjal]. Saadaval: http://docs.identityserver.io/en/release/topics/startup.html. [Kasutatud 14 05 2020].

[55] B. Allen ja D. Baiern, „ASP.NET Identity Support — IdentityServer4 1.0.0 documentation,“ [Võrgumaterjal]. Saadaval: https://identityserver4.readthedocs.io/en/latest/reference/aspnet_identity.html. [Kasutatud 14 05 2020].

[56] Microsoft Corporation, „Azure AD Connect: Use a SAML 2.0 Identity Provider for Single Sign On - Azure | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/azure/active-directory/hybrid/how-to-connect-fed-saml-idp#required-attributes. [Kasutatud 16 05 2020].

[57] Facebook Inc., „React integration for ASP.NET MVC | ReactJS.NET,“ [Võrgumaterjal]. Saadaval: https://reactjs.net/. [Kasutatud 16 05 2020].

[58] A. Taritsyn, „NuGet Gallery | JavaScriptEngineSwitcher.ChakraCore 3.5.4,“ [Võrgumaterjal]. Saadaval:

https://www.nuget.org/packages/JavaScriptEngineSwitcher.ChakraCore/. [Kasutatud 16 05 2020].

[59] Facebook Inc., „Components and Props – React - Rendering a component",“ [Võrgumaterjal]. Saadaval: https://reactjs.org/docs/components-and-props.html#rendering-a-component. [Kasutatud 16 05 2020].

[60] Facebook Inc, „Component State – React,“ [Võrgumaterjal]. Saadaval: https://reactjs.org/docs/faq-state.html#what-is-the-difference-between-state-and-props. [Kasutatud 16 05 2020].

[61] Facebook Inc, „Components and Props – React - Props are Read-Only,“ [Võrgumaterjal]. Saadaval: https://reactjs.org/docs/components-and-props.html#props-are-read-only. [Kasutatud 16 05 2020].

[62] Microsoft Corporation, „Tag Helpers in forms in ASP.NET Core | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-3.1#the-form-action-tag-helper. [Kasutatud 16 05 2020].

[63] Facebook Inc, „Server-Side Rendering | ReactJS.NET,“ [Võrgumaterjal]. Saadaval: https://reactjs.net/features/server-side-rendering.html. [Kasutatud 16 05 2020].

[64] Bootstrap Authors, „Toasts · Bootstrap,“ [Võrgumaterjal]. Saadaval: https://getbootstrap.com/docs/4.3/components/toasts/. [Kasutatud 16 05 2020].

[65] Internet Security Research Group, „Let's Encrypt - Free SSL/TLS Certificates,“ [Võrgumaterjal]. Saadaval: https://letsencrypt.org/. [Kasutatud 16 05 2020].

[66] Internet Security Research Group, „FAQ - Let's Encrypt - Free SSL/TLS Certificates,“ [Võrgumaterjal]. Saadaval: https://letsencrypt.org/docs/faq/#what-is-the-lifetime-for-let-s-encrypt-certificates-for-how-long-are-they-valid. [Kasutatud 16 05 2020].

[67] S. Neal, „Let’s Encrypt for Windows 10 - Beyond the Helpdesk - Medium,“ [Võrgumaterjal]. Saadaval: https://medium.com/beyond-the-helpdesk/lets-encrypt-for-windows-10-e07556c811b4. [Kasutatud 18 05 2020].

[68] Microsoft Corporation, „Azure integration with Office 365 | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/office365/enterprise/azure-integration. [Kasutatud 16 05 2020].

[69] Microsoft Corporation, „App Service Plans | Microsoft Azure,“ [Võrgumaterjal]. Saadaval: https://azure.microsoft.com/en-us/pricing/details/app-service/plans/. [Kasutatud 16 05 2020].

[70] Microsoft Corporation, „Configure TLS mutual authentication - Azure App Service | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-configure-tls-mutual-auth#access-client-certificate. [Kasutatud 14 05 2020].

[71] Riigi Infosüsteemi Amet, „EE Certification Centre Root CA hierarhia > Juursertifikaadid, sertifitseerimisahelad > Teadmistebaas > Arendajale > ID.ee,“ [Võrgumaterjal]. Saadaval: https://www.id.ee/index.php?id=30374. [Kasutatud 16 05 2020].

[72] Riigi Infosüsteemi Amet, „Sertifikaadi kehtivuse kontroll (OCSP) > Sertifikaadi kehtivusinfo kontroll > Isikutuvastus > Arendajale > ID.ee,“ [Võrgumaterjal]. Saadaval: https://www.id.ee/index.php?id=30271. [Kasutatud 16 05 2020].

[73] Microsoft Corporation, „Manage app registration and API permission for Microsoft Graph notifications - Microsoft Graph | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/notifications-integration-app-registration. [Kasutatud 16 05 2020].

[74] Microsoft Corporation, „Overview of Microsoft Graph - Microsoft Graph | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/overview#whats-in-microsoft-graph. [Kasutatud 16 05 2020].

[75] MIcrosoft Corporation, „Get access without a user - Microsoft Graph | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/auth-v2-service. [Kasutatud 16 05 2020].

[76] Microsoft Corporation, „Get access without a user - Microsoft Graph | Microsoft Docs - Get an access token,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/auth-v2-service#4-get-an-access-token. [Kasutatud 16 05 2020].

[77] Microsoft Corporation, „Microsoft Graph permissions reference - Microsoft Graph | Microsoft Docs - User permissions,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/graph/permissions-reference#user-permissions. [Kasutatud 16 05 2020].

[78] ViewDS, „Making Office 365 Work with an External SAML Identity Provider,“ [Võrgumaterjal]. Saadaval: https://www.viewds.com/making-office-365-work-with-an-external-saml-identity-provider/. [Kasutatud 20 04 2020].

[79] Microsoft Corporation, „Use a SAML 2.0 identity provider to implement single sign-on,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/previous-versions/azure/azure-services/dn641269(v=azure.100)?redirectedfrom=MSDN. [Kasutatud 17 05 2020].

[80] Microsoft Corporation, „Connect-MsolService (MSOnline) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/msonline/connect-msolservice?view=azureadps-1.0. [Kasutatud 16 05 2020].

[81] Microsoft Corporation, „Use a SAML 2.0 identity provider to implement single sign-on | Microsoft Docs - Set up trust between your SAML Identity provider and Azure AD,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/previous-versions/azure/azure-services/dn641269(v=azure.100)?redirectedfrom=MSDN#set-up-a-trust-between-your-saml-identity-provider-and-azure-ad. [Kasutatud 17 05 2020].

[82] MIcrosoft Corporation, „New-MsolDomain (MSOnline) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/msonline/new-msoldomain?view=azureadps-1.0. [Kasutatud 16 05 2020].

[83] Microsoft Corporation, „Get-MsolDomainVerificationDns (MSOnline) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/msonline/get-msoldomainverificationdns?view=azureadps-1.0. [Kasutatud 16 05 2020].

[84] Microsoft Corporation, „Configure TLS mutual authentication - Azure App Service | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-configure-tls-mutual-auth#exclude-paths-from-requiring-authentication. [Kasutatud 16 05 2020].

[85] Microsoft Corporation, „Set-MsolDomainAuthentication (MSOnline) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/msonline/set-msoldomainauthentication?view=azureadps-1.0. [Kasutatud 16 05 2020].

[86] Microsoft Corporation, „New-MsolUser (MSOnline) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/msonline/new-msoluser?view=azureadps-1.0. [Kasutatud 16 05 2020].

[87] Riigi Infosüsteemi Amet, „Riigi autentimisteenus (TARA),“ [Võrgumaterjal]. Saadaval: https://e-gov.github.io/TARA-Doku/. [Kasutatud 17 05 2020].

[88] Microsoft Corporation, „Code First to a New Database,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database. [Kasutatud 14 05 2020].

[89] R. Anderson ja R. Nowak, „Introduction to Razor Pages in ASP.NET Core | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio#using-layouts-partials-templates-and-tag-helpers-with-razor-pages. [Kasutatud 14 05 2020].

[90] Microsoft Corporation, „DbSet<TEntity> Class (Microsoft.EntityFrameworkCore) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbset-1?view=efcore-3.1. [Kasutatud 14 05 2020].

[91] B. Allen ja D. Baiern, „ASP.NET Identity Support — IdentityServer4 1.0.0 documentation,“ [Võrgumaterjal]. Saadaval: http://docs.identityserver.io/en/release/reference/aspnet_identity.html. [Kasutatud 14 05 2020].

[92] Microsoft Corporation, „Confirm-MsolDomain (MSOnline) | Microsoft Docs,“ [Võrgumaterjal]. Saadaval: https://docs.microsoft.com/en-us/powershell/module/msonline/confirm-msoldomain?view=azureadps-1.0. [Kasutatud 16 05 2020].

Lisa 1 – MIdClient klass

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using MIdClient.Dto;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

namespace MIdClient
{
  public class MIdClient : IMIdClient
  {
    private static HttpClient _httpClient;

    public MIdClient(HttpClient client)
    {
      _httpClient = client;
    }

    private async Task<HttpResponseMessage> Post<T>(string endpoint, T req)
    {
      DefaultContractResolver contractResolver = new DefaultContractResolver
      {
        NamingStrategy = new CamelCaseNamingStrategy()
      };
      var url = MIdConstants.BASEURL + endpoint;
      var json = JsonConvert.SerializeObject(req, new JsonSerializerSettings
      {
        ContractResolver = contractResolver
      });
      var strContent = new StringContent(json, Encoding.UTF8, "application/json");
      var res = await _httpClient.PostAsync(url, strContent);
      return res;
    }

    private async Task<MIdStatusResponse> Get(string endpoint, string sessionId)
    {
      var url =
$"{MIdConstants.BASEURL}{endpoint}{sessionId}{MIdConstants.TIMEOUT}10000";
      using (_httpClient)
      {
        _httpClient.Timeout = TimeSpan.FromMilliseconds(12500);
        using (var res = await _httpClient.GetAsync(url,
HttpCompletionOption.ResponseHeadersRead))
        {
          using var json = res.Content.ReadAsStringAsync();
          Console.WriteLine(json.Result);
          return JsonConvert.DeserializeObject<MIdStatusResponse>(json.Result);
        }
      }
    }

    public async Task<MIdInitializeAuthResult> InitializeAuthentication(MIdRequest req)
    {
      var hash = MIdHashHandler.GenerateHash();
```

```
            req.Hash = Convert.ToBase64String(hash);
            var verifcationCode = MIdHashHandler.GetVerifcationCode(hash);
            var res = await Post(MIdConstants.AUTHENTICATE, req);
            if (res.IsSuccessStatusCode)
            {
                var returnObj =
                    JsonConvert.DeserializeObject<MIdInitializeAuthResult>(await
res.Content.ReadAsStringAsync());
                returnObj.ConfirmCode = verifcationCode;
                return returnObj;
            }
            else
            {
                return JsonConvert.DeserializeObject<MIdInitializeAuthResult>(await
res.Content.ReadAsStringAsync());
            }
        }

        public async Task<MIdStatusResponse> GetAuthenticationStatus(string sessionId)
        {
            var mIdStatusResponse = await Get(MIdConstants.AUTHENTICATIONRESULT, sessionId);
            return mIdStatusResponse;
        }
    }
}
```

Lisa 2 – Mobiil-ID API testkeskkonna konstandid

```
public static class MIdConstants
  {
    public const string BASEURL = "https://tsp.demo.sk.ee/mid-api/";
    public const string RelyingPartyUUID = "00000000-0000-0000-0000-000000000000";
    public const string RelyingPartyName = "DEMO";
    public const string AUTHENTICATE = "authentication";
    public const string AUTHENTICATIONRESULT = "authentication/session/";
    public const string TIMEOUT = "?timeoutMs=";
  }
}
```

Lisa 3 – MIdRequest klass

```
public class MIdRequest
  {
    public string RelyingPartyUUID { get; set; }
    public string RelyingPartyName { get; set; }
    public string PhoneNumber { get; set; }
    public string NationalIdentityNumber { get; set; }
    public string Hash { get; set; }
    public string HashType { get; set; }
    public string Language { get; set; }
    public string DisplayText { get; set; }
    public string DisplayTextFormat { get; set; }
  }
}
```

Lisa 4 – UserMangement komponent

```
import {Component, Fragment} from 'react';
import {Search} from './search.jsx';
import {Users} from './users.jsx';
import {Toaster} from "./toaster.jsx";
import axios from 'axios';

export default class UserManagementComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {data: [], toastStatus: "", toastText: "", toastHeader: ""};
    this.searchUser = this.searchUser.bind(this);
    this.actionResult = this.actionResult.bind(this);
    this.fetchUsers = this.fetchUsers.bind(this)
  }

  componentDidMount() {
    this.fetchUsers();
  }
  fetchUsers() {
    axios.get('usermanagement/getusers')
      .then(res => {
        this.setState({data: res.data})
      })
      .catch(err => {
        console.warn(err);
      });
  }
  searchUser = (childData) => {
    this.setState({data: childData});
  }
  actionResult = () => {
    this.fetchUsers();
  }
  toast = (childData) => {
    console.log(childData);
    this.setState({toastText: childData.text, toastHeader: childData.header});
  }

  render() {
    return (
      <div>
        <Search root="/usermanagement/" url="search" importCallBack={this.actionResult}
searchCallBack={this.searchUser} toastCallBack={this.toast}/>
        <div className="user-management-splitter"></div>
        <Toaster text={this.state.toastText} header={this.state.toastHeader} delay={3000}/>
        <Users users={this.state.data}/>
      </div>
    )
  }
}
```

Lisa 5 – TenantAdmins komponent

```
import {Component, Fragment} from 'react';
import {Search} from './search.jsx'
import axios from "axios";
import {Toaster} from "./toaster.jsx";

export default class TenantAdminsComponent extends Component {

  constructor(props) {
    super(props);
    this.state = {
      tenantAdmins: [],
      adminEmail: "",
      toastStatus: "",
      toastText: "",
      toastHeader: "",
      errors: {email: ""}
    };
    this.timeout = 0;
    this.addAdmin = this.addAdmin.bind(this);
    this.handleChange = this.handleChange.bind(this);
    this.clearToaster = this.clearToaster.bind(this);
    this.validateForm = this.validateForm.bind(this);
  }

  componentDidMount() {
   this.loadAdmins();
  }

  validateForm(text) {
    const validEmailRegex =

RegExp(/^(([^<>()\[\]\.,;:\s@\"]+(\.[^<>()\[\]\.,;:\s@\"]+)*)|(\".+\"))@(([^<>()[\]\.,;:\s@\"]+\.)
+[^<>()[\]\.,;:\s@\"]{2,})$/i);
    if (validEmailRegex.test(text)) {
      this.state.errors.email = "";
    } else {
      this.state.errors.email = "Email isn't valid"
    };
  }

  addAdmin(event) {
    let postItem = {
      email: this.state.adminEmail
    };
    axios.post('/usermanagement/addadmin', postItem)
      .then(res => {
        this.setState({toastText: "Admin added", toastHeader: "Success"})
        this.loadAdmins();
      }).catch(err => {
      this.setState({toastText: "Something went wrong while adding admin", toastHeader:
"Unsuccessful"})
    });
    event.preventDefault();
  }
  loadAdmins() {
```

```
    axios.get('/usermanagement/GetTenantAdmins')
      .then(res => {
        console.log(res);
        this.setState({tenantAdmins: res.data})
      })
      .catch(err => {
        console.log(err);
      })
  }
  searchAdmin = (childData) => {
    this.setState({tenantAdmins: childData})
  }

  removeAdmin(idx) {
    let admin = this.state.tenantAdmins[idx];
    let postItem = {
      id: admin.id
    };
    console.log(postItem);
    axios.post('/usermanagement/removeadmin', postItem)
      .then(res => {
        this.setState({toastText: "Admin removed", toastHeader: "Success"})
        this.loadAdmins();
      })
      .catch(err => {
        this.setState({toastText: "Something went wrong while removing admin", toastHeader:
"Unsuccessful"})
      })
  }

  handleChange(event) {
    this.setState({adminEmail: event.target.value});
    this.validateForm(event.target.value);
  }
  clearToaster() {
    this.setState({toastStatus: "", toastText: "", toastHeader: ""})
  }

  render(props) {
    let errors = this.state.errors;
    let adminsList = this.state.tenantAdmins.map((user, idx) => {
      return (
        <tr className="user-table-row" key={idx}>
          <td>{user.email}</td>
          <td>
            <button className="btn btn-danger" type="submit" name="remove" onClick={() =>
{this.removeAdmin(idx)}}>
              Remove
            </button>
          </td>
        </tr>);
    });
    return (
      <div>
        <Search root="" url="adminsearch" searchCallBack={this.searchAdmin}/>
        <div className="user-management-splitter"></div>
        <Toaster callBack={this.clearToaster} text={this.state.toastText}
header={this.state.toastHeader} delay={3000}/>
        <div className="row">
```

```
            <div className="col-2">
              {this.state.errors.email.length > 0 &&
              <div className="form-group">
                <span className='text-danger'>{this.state.errors.email}</span>
              </div>}
            </div>
            <div className="col-4">
              <form className="form-inline" onSubmit={this.addAdmin}>
                <div className="form-group">
                  <input type="text" className="form-control" name="email" placeholder="Admin
email" onChange={this.handleChange}/>
                </div>
                <div className="form-group">
                  <input disabled={errors.email.length > 0} className="btn btn-default"
type="submit" value="Add"/>
                </div>
              </form>
            </div>
          </div>
          <div className="col-6">
          </div>
          <div className="row">
            <div className="col-2"></div>
            <div className="col-8">
              <a className="user-heading">Tenant admins</a>
              <table className="table table-margin">
                <thead className="thead">
                <tr className="table-head-text">
                  <th scope="col">Email</th>
                  <th scope="col">Action</th>
                </tr>
                </thead>
                <tbody>
                {adminsList}
                </tbody>
              </table>
            </div>
          </div>
        </div>
    );
  }
}
```

Lisa 6 – Users komponent

```
import { Component, Fragment } from 'react';
import axios from "axios";

export class Users extends Component {

  constructor(props) {
    super(props);
    this.idRef = React.createRef();
    this.inputRef = React.createRef();
    this.timeout = 0;
    this.state = {error: {idcode: ""}}
    this.validateForm = this.validateForm.bind(this);
    this.submit = this.submit.bind(this);
  }

  submit = idx => event =>  {
    let user = this.props.users[idx];
    let postItem = {
      "userGuid": user.id,
      "idCode": event.target.value
    }
    if (this.timeout) clearTimeout(this.timeout);
    this.timeout = setTimeout(() => {
      if (this.state.error.idcode.length !== 0) {
        return;
      }
      axios.post('usermanagement/UpdateIdCode', postItem)
        .then(res => {
        })
        .catch(err => {
        })
    }, 500);

  }
  validateForm(text) {
    const pattern =
      RegExp(/[1-6][0-9]{2}[1,2][0-9][0-9]{2}[0-9]{4}/)
    if(pattern.test(text)) {
      this.state.error.idcode = ""
    } else {
      this.state.error.idcode = "Incorrect ID code"
    }

  }

  render(props) {
    console.log(this.props.users);
    let usersList = this.props.users.map((user, idx) => {
      return (
        <tr className="user-table-row" key={idx}>
          <td>{user.normalizedEmail}</td>
          <td>{user.subjectId}</td>
          <td>
            <form>
              <input type="hidden" name="userGuid" value={user.id}/>
```

```
                    <input type="text" className="form-control" name="idCode"
defaultValue={user.idCode ? user.idCode : ""} onChange={this.submit(idx)}/>
                  </form>
                </td>
                {this.state.error.idcode.length > 0 &&
                <td className="form-group">
                  <span className='text-danger'>{this.state.error.idcode}</span>
                </td>}
              </tr>);
      });
      return (
        <div className="row">
          <div className="col-2"></div>
          <div className="col-8">
            <a className="user-heading">Local Users</a>
            <table className="table table-margin">
              <thead className="thead">
              <tr className="table-head-text">
                <th scope="col">Email</th>
                <th scope="col">Immutable guid</th>
                <th scope="col">Id code</th>
              </tr>
              </thead>
              <tbody>
              {usersList}
              </tbody>
            </table>
          </div>
        </div>
      );
    }
}
```

Lisa 7 – Search komponent

```
import {Component, Fragment} from 'react';

export class Search extends Component {
  constructor(props) {
    super(props);
    this.state = {data: []};
    this.searchUser = this.searchUser.bind(this);
    this.importUsers = this.importUsers.bind(this);
  }

  searchUser(event) {
    fetch(this.props.root + this.props.url + "?q=" + event.target.value)
      .then(data => {
        return data.json();
      })
      .then((json) => {
        this.props.searchCallBack(json);
      })
      .catch(err => {
        console.warn(err);
      });
  }
  importUsers() {
    fetch('usermanagement/importusers')
      .then(data => {
        return data.json();
      })
      .then((json) => {
        let responseObj = {header: "", text: ""};
        if (json.userAmount === -1) {
          responseObj.header = "Import failed"
          responseObj.text = "Something went wrong."
        } else {
          responseObj.header = "Import succeeded"
          responseObj.text = json.userAmount + " users imported."
          this.props.importCallBack();
        }
        this.props.toastCallBack(responseObj);
      })
      .catch(err => {
        console.log(err);
      })
  }

  render() {
    let importRender = this.props.root !== "" ?
      <li className="nav-item">
      <a className="nav-link" onClick={this.importUsers} id="importusers">Import Users</a>
    </li> : "";
    return (
      <nav className="navbar navbar-expand-sm navbar-light nav-white">
        <div className="col-2">
        </div>
        <div className="col-8">
          <ul className="navbar-nav mr-auto nav-items">
```

```
                <li className="nav-item">
                  <a className="nav-link" href="/usermanagement">Home <span className="sr-
only">(current)</span></a>
                </li>
                <li className="nav-item">
                  <a href={"/usermanagement/addtenantadmin"} className="nav-link">Tenant
admins</a>
                </li>
              </ul>
              <ul className="nav navbar-nav navbar-right nav-items navbar-light nav-white">
                {importRender}
                <form className="form-inline my-2 my-lg-0 float-right">
                  <input name="q" className="form-control mr-sm-2" placeholder="Search"
                      aria-label="Search" onChange={this.searchUser}/>
                </form>
              </ul>
            </div>
            <div className="col-2">
            </div>
          </nav>
      );
    }
}
```

Lisa 8 – Toaster komponent

```
import {Component, Fragment} from 'react';

export class Toaster extends Component {
  constructor(props) {
    super(props);
    this.state = {visible: true};
    this.close = this.close.bind(this);
  }

  componentWillReceiveProps(nextProps) {
    this.setTimer();
    this.setState({visible: true});
  }

  componentDidMount() {
    this.setTimer();
  }

  setTimer() {
    if (this._timer != null) {
      clearTimeout(this._timer)
    }

    this._timer = setTimeout(function () {
      this.setState({visible: false});
      if (this.props.callBack) {
        this.props.callBack();
      }
      this._timer = null;
    }.bind(this), this.props.delay);
  }

  componentWillUnmount() {
    clearTimeout(this._timer);
  }

  close() {
    this.setState({visible: false})
  }

  render() {
    console.log(this.state.visible);
    if (!this.state.visible || this.props.text === "") {
      return null;
    }
    return (
      <div className="position-absolute w-100 p-4 d-flex flex-column align-items-end">
        <div className="w-25">
          <div className="toast ml-auto show" role="alert" aria-live="assertive" aria-
atomic="true">
            <div className="toast-header">
              <strong className="mr-auto">{this.props.header}</strong>
              <button type="button" className="ml-2 mb-1 close" aria-label="Close"
onClick={this.close}>
                <span aria-hidden="true">&times;</span>
```

```
                </button>
              </div>
              <div className="toast-body">
                {this.props.text}
              </div>
            </div>
          </div>
        </div>
      );
    };
}
```