Antti Paavola

Tunnistautuminen ja oikeuksien hallinta web-sovelluksessa

Opinnäytetyö

Tekniikan ammattikorkeakoulututkinto

Peliohjelmoinnin koulutus

2022

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä	Antti Paavola
Työn nimi	Tunnistautuminen ja oikeuksien hallinta web-sovelluksessa
Toimeksiantaja	Cadmatic Oy
Vuosi	2022
Sivut	46 sivua
Työn ohjaajat	Niina Mässeli, Pekka Vilpponen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli toteuttaa käyttäjien tunnistautuminen ja käyttöoikeuksien hallinta osaksi sähkösuunnitteluprojektien versionhallintatyökalua. Kehitystyön vaatimuksena oli eri tasoisten käyttöoikeuksien jakaminen. Eri tasoiset käyttöoikeudet tarkoittavat tässä tapauksessa käyttäjäkohtaisia, käyttäjäryhmäkohtaisia sekä projektikohtaisia käyttöoikeuksia. Lisäksi käyttäjän tunnistautumisessa lähtökohtana oli, että käyttäjä voi tunnistautua, joko sovellukseen erikseen luoduilla tunnuksilla tai Windows-tunnistautumisen avulla.

Työn toimeksiantaja on Cadmatic Oy, jonka käyttöön lopullisen tuotteen on tarkoitus päätyä. Tämän kehittämistutkimuksen tavoitteena on löytää sopivat teknologiat ja tavat käyttäjien tunnistamiseen ja eri tasojen oikeuksien antamiseen ja niiden tarkistamiseen. Lopullinen päämäärä on integroida tunnistautuminen ja oikeuksien hallinta osaksi olemassa olevaa versionhallintatyökalua, jotta se voidaan ottaa käyttöön.

Opinnäytetyössä käydään ensin läpi teoriaa tunnistautumisen ja käyttöoikeuksien takana. Lisäksi tarkastellaan web-sovelluksen toimintaa yleisellä tasolla, sillä tämäkin web-sovellus nojaa vahvasti muun muassa HTTP-protokollan avulla tehtäviin kutsuihin ja niistä saatuihin vastauksiin. Teoriaosuudessa käydään läpi myös REST-arkkitehtuurimallia, johon sovelluksen taustajärjestelmän toteutus nojaa.

Tuloksia käsittelevä osuus käy läpi tarkemmin edellä mainittuun teoriaan pohjautuvat käytännön toteutukset ja miten juuri tähän sovellukseen valitut teknologiat hyödyntävät teoriaosuudessa käsiteltyjä periaatteita.

Opinnäytetyön tuloksena kehitettiin sovellukseen osia, jotka ratkaisevat tutkimusongelman käyttäjän tunnistautumisesta ja tämän eri tasojen oikeuksien hallinnasta. Käyttäjä tunnistautuu sovellukseen joko kaksivaiheisella tunnistautumisella vahvennetulla käyttäjänimi-salasana-yhdistelmällä tai vaihtoehtoisesti Windows-tunnuksilla. Oikeuksien tarkastamista tehdään kahdella tasolla. Asiakassovelluksessa pyritään estämään sellaisten toimintojen suoritus, johon käyttäjällä ei ole oikeuksia, mutta lopullinen varmistus suoritetaan palvelimella kutsua tehdessä.

Asiasanat: tunnistautuminen, käyttäjänhallinta, web-sovellus, .net, HTTP, REST

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author	Antti Paavola
Thesis title	Authentication and authorization in web-application
Commissioned by	Cadmatic Oy
Time	2022
Pages	46 pages
Supervisors	Niina Mässeli, Pekka Vilpponen

ABSTRACT

The purpose of this thesis was to create components necessary for authenticating users and authorizing users and user-groups in an electrical design project management and version control web-application. The objective for the authorization part of this work was to enable the assignment different access levels to users and groups. In the scope of this thesis, the different levels translate as user-specific, group-specific, and project-specific access rights. As for authentication, the aim was to allow the user to authenticate with Windows-credentials when working on the intranet or with username and password created specifically to the software if the software is used via the internet. Multi-factor authentication was also to be implemented so it would be an option when using traditional authentication with username and password. The aim was also to find optimal or at least viable solutions for the authentication and authorization processes. And integrate them in the existing version control tool.

In the theory section, this thesis reviews the measures that are necessary to develop an authentication and authorization system. General characteristics and functions of web-applications are also examined. Primarily, the focus is on REST-architecture, HTTP-protocol and application programming interfaces, databases, and cryptography. Particular attention is paid to the HTTP-protocol as the messaging between client and backend is done via HTTP-messages.

The outcome of this thesis is a set of components for a web-application backend and client-software that meet the requirements stated by the commissioner. The user authenticates to the application either with their Windows-credentials or with a combination of username and password which is hardened with time-based one-time password two-factor authentication. Authorization is ensured on two levels. First, the client-software tries to prevent the user from using functions they to which they do not have access to, and the final confirmation is made on the server-side within the handler-function.

Keywords: authentication, authorization, web-application, .net, HTTP, REST,

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä tarkastellaan web-sovellukseen tulevien käyttäjän tunnistautumisen ja tämän oikeuksien hallinnan mahdollistavien osien kehitystyötä sekä kyseisten osien kehittämiseen tarvittavaa teoriaa.

Teoriaosuudessa käsitellään tunnistautumista ja oikeuksienhallintaa web-sovelluskehityksen ja tietoturvan näkökulmasta. Lisäksi tullaan käymään läpi web-sovelluksen toiminnan kannalta olennaisten HTTP-protokollan ja REST-arkkitehtuurimallin teoriaa, sillä kehitettävän sovelluksen toteutus nojaa vahvasti näihin kahteen.

Aihe valikoitui toimeksiantajan, Cadmatic Oy:n EAC-tuotteiden, tarpeesta kehittää etätyötä helpottava työkalu, joka mahdollistaa usean sähkösuunnittelijan työskentelyn samassa projektissa ilman jatkuvaa yhteyttä palvelimella olevaan suunnitteluprojektiin.

Cadmatic on vuonna 1985 Turussa perustettu eri teollisuuden alojen suunnitteluohjelmistoja kehittävä ohjelmistotalo. Turun pääkonttorin lisäksi Cadmatic toimii Suomessa Tampereella ja Kotkassa. Suomen ohella Cadmatic Oy:llä on toimipisteitä jokaisella mantereella. Yrityksen toiminta on jaettu kolmeen haaraan teollisuuden alojen mukaan: laiva- ja meriteollisuuteen, prosessiteollisuuteen ja rakennusteollisuuteen. Cadmatic tarjoaa näille teollisuuden aloille suunnattuja tietokoneavusteisia suunnitteluohjelmistoja eli CAD-ohjelmistoja. Lisäksi tuoteperheeseen kuuluu erilaisia informaationhallinnan työkaluja. Tämän opinnäytetyön kehitystyö sijoittuu organisaatiossa rakennusteollisuuden ja tarkemmin sähköteollisuuden alle.

Käytännössä työkalun avulla suunnittelija replikoi projektin kokonaisuudessaan paikalliselle työasemalle. Tämän jälkeen muutoksia voidaan tehdä paikallisesti ilman yhteyttä palvelimelle ja viedä pelkät muutokset palvelimen projektiin, josta ne ovat muiden projektin suunnittelijoiden saatavilla. Työkalun kehitys oli tullut siihen vaiheeseen, että heräsi tarve hallita eri suunnittelijoiden käyttöoikeuksia, jotta suunnittelijat eivät tee muutoksia sellaisiin projekteihin,

joihin heillä ei ole oikeuksia. Pahimmassa tapauksessa saatetaan poistaa tietoa vahingossa palvelimelta, kun tarkoitus oli poistaa projekti paikalliselta työasemalta. Opinnäytetyön lopputuloksen onkin tarkoitus helpottaa sekä ylläpitäjien että suunnittelijoiden työskentelyä siten, ettei henkilö yksinkertaisesti kykene tekemään vahingossa toimintoja, johon hänellä ei ole valtuuksia. Lisäksi on tarkoitus varmistaa, ettei ulkopuolisilla ole pääsyä palvelimen projekteihin ilman asianmukaista tunnistautumista.

Suurin haaste näiden osien kehityksessä on tietoturvan varmistamisen lisäksi eri tunnistautumistapojen liittäminen yhdeksi kokonaisuudeksi.

2 TUTKIMUKSEN LÄHTÖKOHDAT

Tässä luvussa kuvataan tämän opinnäytetyön tavoitteita ja tarkoitusta. Lisäksi esitellään tutkimusmenetelmä, tutkimusongelma sekä tutkimusongelmasta johdetut tutkimuskysymykset.

2.1 Tavoitteet

Tämän opinnäytetyön tavoitteena on toteuttaa käyttäjän tunnistautuminen ja tunnistautuneen käyttäjän eri tasoisten oikeuksien hallinta osaksi sähkösuunnitteluprojektien versionhallintatyökalua.

Lopullisten osien tulee mahdollistaa käyttäjien ja ryhmien lisääminen osaksi replikointityökalua, Windows Active Directory -ryhmien ja käyttäjien tuominen työkalun ryhmiksi ja käyttäjiksi, sisäänkirjautuminen, palvelinkutsujen tekemisen estäminen ilman tunnistautumista. Näiden toteutus vaatii erillisen tietokannan luomisen, johon voidaan luoda suoraan käyttäjiä ja ryhmiä ja johon Active Directory -tietoa voidaan tuoda. Itse projektien tiedot sijaitsevat muualla ja niistä tuodaan vain tunniste, joiden avulla oikeuksia jaetaan.

2.2 Tutkimuskysymykset

Opinnäytetyö vastaa tutkimusongelmasta eli käyttäjän tunnistautumisesta ja oikeuksien hallinnasta web-sovelluksessa johdettuihin tutkimuskysymyksiin:

- Miten toteutetaan käyttäjän tunnistautuminen käyttöliittymän ja palvelimen puolella?

- Miten toteutetaan kaksivaiheinen tunnistautuminen tietoturvan lisäämiseksi?
- Kuinka varmistetaan, ettei väärillä käyttöoikeuksilla voi käyttää sovelluksen toimintoja?
- Miten toteutetaan ryhmä-, käyttäjä- ja projektikohtaiset käyttöoikeudet?

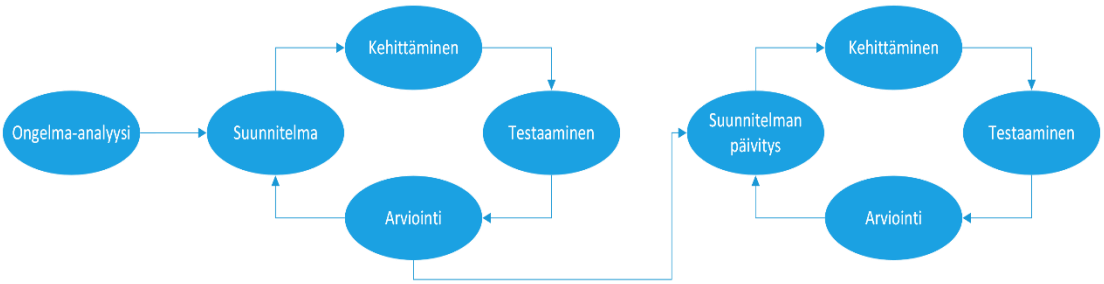
Näihin kysymyksiin vastaamalla ja soveltamalla löydettyjä vastauksia pyritään kehittämään sovelluksen osat, jotka mahdollistavat tunnistautuneen käyttäjän toiminnan sovelluksessa tämän oikeuksien rajoissa.

2.3 Tutkimusote

Tämä opinnäytetyö toteutetaan kehittämistutkimuksena, joka on yksi interventionistisen tutkimuksen muoto. Interventiotutkimukset tähtäävät johonkin muutokseen, kun perinteinen tutkimus pyrkii usein ainoastaan selittämään tutkittavaa ilmiötä. Interventiotutkimuksen lähtökohtana onkin jokin käytännön ongelma, joka pyritään ratkaisemaan tutkimuksen avulla. (Kananen 2017, 10.) Kehitystyö ja muutokseen pyrkiminen ei automaattisesti ole kehittämistutkimusta. Muutoksen aikaansaaminen eli itse kehittäminen on vain osa tutkimustyötä ja sen on perustuttava aiemmin tutkittuun teoriaan. Pelkän kehitystyön ja kehittämistutkimuksen ero onkin tutkimusosio (Kananen 2017, 18).

Kehittämistutkimus on aloitettava ongelma-analyysilla, jonka aikana selvitetään kehitystarpeet ja mahdolliset vastaantulevat haasteet. Ongelma-analyysia kutsutaan myös tarveanalyysiksi. Analyysia voidaan tehdä kahdella eri tavalla, empiirisesti tai teoreettisesti. Analyysi voi myös koostua molemmista tavoista. Empiirinen analyysi voi sisältää esimerkiksi kehitystutkimuksen tuloksena syntyvän tuotteen loppukäyttäjien tarpeiden kartoittamista. Teoreettisella analyysilla tarkoitetaan tutkitun tiedon kirjallisuusanalyysia (Pernaa s.a., 4.) Tämän opinnäytetyön tarveanalyysi toteutettiin empiirisenä ja sen aikana käytiin läpi sähkösuunnittelusovelluksen käyttäjien tarpeita, joista muodostui tutkimusongelma. Opinnäytetyön tarkoituksena onkin ratkaista käytännön ongelma, tunnistautuminen ja oikeuksien hallinta, nojaten kerättävään tutkimusaineistoon. Tärkeimpänä tutkimusaineistona tullaan käyttämään eri protokollien, algoritmien ja suunnittelumallien vakiintuneita määritelmiä, sekä käytettyjen teknologioiden dokumentaatiota. Lisäksi tietoa hankitaan teknologia-alan

verkkojulkaisuista, sillä näistä saadaan nopeasti muuttuvan alan ajankohtaista tietoa.



Kuva 1. Kehittämistutkimuksen sykli (Pernaa s.a., 7)

Kehittämistutkimuksen käytännön toteutus on malliltaan syklinen. Tämä tarkoittaa käytännössä sitä, että tutkimustyössä edetään kuvan 1 kaltaisessa syklissä: tarveanalyysin jälkeen muodostetaan suunnitelma, tehdään kehitystyötä suunnitelman perusteella, testataan, arvioidaan ja päivitetään suunnitelmaa tarpeen mukaan tämän pohjalta. (Pernaa s.a, 7.) Tämä malli soveltuu erittäin hyvin tämän opinnäytetyön toteutuksen malliksi, sillä kehitystyötä tehdään osana ketterää tuotekehitystä, jossa kehitystyötä testataan ja tulosten perusteella kehitystyön suuntaa voidaan muuttaa lyhyelläkin aikavälillä tarpeen niin vaatiessa ja yhden syklin kesto on yhden kuukauden mittaisen kehitysikkunan eli sprintin kesto. Kehitysikkunan sisällä suunnitelmaa voidaan päivittää tarpeen vaatiessa, vaikka päivittäin.

3 TEOREETTINEN VIITEKEHYS

Kehitystyö perustuu vahvasti olemassa olevaan tietoon ja sen ymmärtämiseen. Jotta voidaan toteuttaa tunnistautumisesta ja oikeuksienhallinnasta sovelluksessa vastaavat osat, on ensin ymmärrettävä web-sovelluksen toimintaperiaatteet ja tässä tapauksessa REST-arkkitehtuurimalliin perustuvat palvelinpuolen ohjelmointirajapinnan toiminta. REST-arkkitehtuurimallin pohjalla taas toimii HTTP-protokolla. Tämän kehitystyön toteuttaminen ei vaadi HTTP-protokollan syvällistä ymmärtämistä, mutta on olennaista tietää miten web-sovelluksen eri osat kommunikoivat sen avulla, jotta voidaan kehittää verkon yli toimivia sovelluksia.

Tiedonvälitykseen tarvittavan teorian lisäksi kehitystyön perustana toimii ymmärrys oikeuksien hallinnasta yleisellä tasolla sekä web-sovelluksen näkökulmasta tarkasteltuna. Tässä opinnäytetyössä käyttöoikeuksia tarkastellaan teoriaosuudessa yleisellä tasolla ja tulokset luvussa tullaan keskittymään tarkemmin samaan asiaan web-sovelluksen näkökulmasta.

Jotta käyttäjien tietoja ja oikeuksia voidaan hallita, on ne tallennettava johonkin. Tässä astuu kuvaan tietokannat, tämän työn tapauksessa relaatiotietokannat. Tässä luvussa tullaan käsittelemään teoriaa relaatiotietokantojen taustalla, sekä vertailemaan relaatiotietokantojen ja dokumentti- eli niin kutsuttujen NoSQL-tietokantojen eroja. Tuloksia käsittelevässä luvussa tutustutaan lähemmin, miten relaatiotietokantoja hyödynnetään web-sovelluskehityksessä.

Kehitystyön yksi merkittävin kohde on monivaiheisen tunnistautumisen kehittäminen käyttäjätunnus-salasana-kirjautumisen vahventamiseksi. Toimivan tunnistautumisen kehittämisen onnistuminen edellyttää erilaisten tunnistautumistapojen ja näiden vahvuuksien ja heikkouksien ymmärtäminen ja tämän tiedon pohjalta oikean tavan valinta kehityskohteeseen.

3.1 HTTP-protokolla

Hypertext Transfer Protocol (*HTTP*) on protokolla, jonka avulla välitetään erilaisia resursseja asiakasohjelmien ja palvelimien välillä. Yksinkertaistettuna tämä tarkoittaa, että asiakasohjelma, monesti web-selain, pyytää resurssin, jonka palvelin palauttaa, jos resurssi löytyy. (An overview of HTTP 2021.) Keskustelu verkossa asiakasohjelmien ja palvelimien, sekä eri palvelimien välillä tapahtuu yksittäisillä viesteillä, jotka ovat protokollan määritelmän mukaan joko pyyntöjä tai vastauksia näihin pyyntöihin (Fielding & Reschke 2014, 1–6). Sekä pyyntö, että vastaus koostuvat neljästä pääosasta: aloitusrivistä, HTTP-otsikoista, tyhjästä rivistä, joka toimii erottimena edellä mainittujen metatietojen ja viestin viimeisen osan eli sisältörivien välillä (An overview of HTTP 2021).

```
GET /api/devices HTTP/1.1
Host: localhost:3003
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:94.0) Gecko/20100101 Firefox/94.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Kuva 2. HTTP-pyynnön otsikkorivit

Kuvassa 2 nähdään, kuinka HTTP-pyynnön ensimmäisellä rivillä määritellään käytetty metodi, halutun resurssin tunniste sekä käytetty protokolla. Kuvan seuraavilla riveillä on määritelty kohdepalvelin ja portti, joka on tässä tapauksessa omalla päätteellä oleva palvelin, pyynnön tehnyt asiakasohjelma sekä määritelmä hyväksytylle tiedolle.

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 786
ETag: W/"312-XesmdtQD7PmKRzfaIcCw/8VEDtg"
Date: Fri, 03 Dec 2021 08:05:29 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

Kuva 3. HTTP-vastauksen otsikkorivit

Kuvan 3 HTTP-vastauksen ensimmäiseltä riviltä selviää käytetty protokolla sekä HTTP-tilakoodin numeroarvo sekä sanallinen viesti. Ensimmäistä otsikkoriviä seuraa päiväys, palvelin, vastauksen pituus bitteinä sekä minkä tyyppinen resurssi palautetaan. Kuvan 1 tapauksessa pyyntö tehtiin /api/devices-tunnisteeseen osoitteessa localhost porttiin 3003 ja vastauksena saatiin JSON-muodossa olevaa dataa. Itse JSON-data löytyy vastauksen sisältöosasta.

HTTP-protokolla määrittelee standardisoituja metodeja eri käyttötarkoituksiin. Määrittelyssä ei oteta kantaa missä muodossa palautettava resurssi on. Määritelmissä on pyritty siihen, että metodit määrittelevät ainoastaan sen mitä tehdään. HTTP-protokolla ei myöskään pakota käyttämään tiettyä metodia tietyn toiminnon suorittamiseen ja valinta jätetäänkin protokollaa käyttävän palvelun tehtäväksi. (Fielding & Reschke 2014, 21.)

GET JA HEAD

GET- ja HEAD-metodien tarkoitus on hakea tietyn resurssin sen hetkinen esitys palvelimelta. GET-pyynnöstä palautetaan kaikki yllä esitellyt vastauskentät. HEAD eroaa tästä siten että sen vastauksena ei tulisi saada sisältökenttää. Pääasiallinen metodi tiedonhakuun on GET. HEAD-metodia käytetään esimerkiksi linkkien testaamiseen ja pelkän metadatan eli otsikkorivien hakemiseen resurssin muutosten havaitsemiseksi. (Fielding & Reschke 2014, 24–25.)

POST

POST-metodia tulisi käyttää, kun halutaan lähettää dataa palvelimelle, joka vaikuttaa saatuun resurssiin. POST-metodia käytetään esimerkiksi web-sivulla olevan lomakkeen tietojen lähettämiseen palvelimelle. Vastaus pyyntöön muodostetaan lähetettyjen tietojen perusteella ja se palautetaan samaan tapaan kuin GET-metodia käytettäessä. (Fielding & Reschke 2014, 25–26.)

PUT JA DELETE

Perinteisesti harvemmin käytetty, mutta tietokannan kanssa keskustelussa hyödyllinen pyyntö. Tätä metodia tulisi käyttää, kun halutaan lähettää täysin uutta tietoa palvelimelle. Myös DELETE on harvemmin perinteisessä web-liikenteessä nähty mutta tietokannan kanssa keskustelussa käytetty metodi. Pyynnön tarkoitus on poistaa resurssi palvelimelta. (Fielding & Reschke 2014, 26–30.)

3.2 REST-arkkitehtuuri

REST eli Representational State Transfer on arkkitehtuurimalli, jonka Fielding (2000, 76) määrittelee rajoitteiden joukkona. Hän kutsuu sitä väitöskirjassaan hybridiarkkitehtuuriksi, joka hyödyntää edeltäjiään ja määrittelee uusia ominaisuuksia. REST-mallin ja muiden arkkitehtuurimallien suurimpana erottava ominaisuutena on sen ajatus yhtenäisestä rajapinnasta komponenttien välisessä keskustelussa. Ajatuksena on yksinkertaistaa ja pienentää sidosta palvelimen ja asiakassovelluksen välillä. (Fielding 2000, 81–82.) Tämän ajatuksen tueksi

on luotu sääntöjä, jotka määräävät minkälainen rajapinnan tulisi olla. Resurssi haetaan siihen viittavan Uniform Resource Identifier (*URI*) -tunnisteen avulla. Resurssista palautetaan sen hetkinen ilmentymä itse resurssin asemesta. Esimerkkinä pyydettäessä käyttäjää tietokannasta palautetaan käyttäjän tiedot esimerkiksi JSON- tai XML-muodossa. Asiakasohjelmalla tulee olla tarpeeksi tietoa, jotta se pystyy tekemään kutsuja palvelimelle. Jokaisen viestin tulisi myös sisältää tieto, kuinka tietoa käsitellään. (Massé 2012, 3–4).

REST-mallille olennaista on myös asiakkaan ja palvelimen logiikan erottaminen toisistaan. Tällä pyritään jälleen pienentämään eri komponenttien välistä yhteyttä ja helpottamaan sovelluksen eri osien kehitystä. (Fielding 2000, 78.) Kun palvelinpuolen toteutus ei ole sidoksissa käyttöliittymän kanssa, pystytään samaa taustajärjestelmää hyödyntämään esimerkiksi eri laitteille tehdyissä sovelluksissa. Yllä mainittujen ominaisuuksien lisäksi tässä tutkielmassa tarkasteltavalle kehitystyölle on olennaista huomioida yksi REST-mallin pääominaisuus: tilattomuus. Tällä tarkoitetaan, että mitkään lähetetyt viestit eivät tiedä lainkaan sovelluksen sen hetkisestä tilasta. Käytännössä tämä tarkoittaa sitä, että tunnisteeseen tehty pyyntö palauttaa saman resurssin huolimatta siitä mistä sitä kutsutaan tai mitä muuta dataa sovelluksella on sillä hetkellä käytössä. Viestien on siis sisällettävä kaikki tarvittava tieto sen käsittelyyn. (Fielding 2000, 78–79.)

3.3 Ohjelmointirajapinta

Ohjelmointirajapinnalla tarkoitetaan ohjelmaa, joka mahdollistaa ohjelmien välisen kommunikaation (Proffit 2013.) Tällä kommunikaatiolla voidaan tarkoittaa esimerkiksi selaimen tekemää pyyntöä hakea käyttäjän tiedot tietokanasta, jonka ohjelmointirajapinta käsittelee tavalla, jonka rajapinnan kehittäjä on sille määritellyt. Web-kehityksen näkökulmasta tämä tarkoittaa käytännössä ohjelmaa, joka kuuntelee sille määritettyyn porttiin tulevia kutsuja.

Tässä kehitystutkimuksessa käytettävä ohjelmointirajapinta noudattaa REST-arkkitehtuuria, joka perustuu HTTP-protokollassa määriteltyihin kutsuihin ja vastauksiin. Käytännössä rajapinnalle tehdään asiakassovelluksesta HTTP-protokollassa määriteltyjä GET-, POST-, DELETE-, PUT- tai UPDATE-kutsuja erilaisiin rajapinnassa määriteltyihin URI-tunnisteisiin (RESTful Web Services -

Messages 2016). On olennaista, että kutsut tehdään käyttämällä juuri HTTP-protokollassa määritellyillä avainsanoilla, jotta REST-mallissa määritelty yhtenäisen rajapinnan vaatimus täyttyy.

Ohjelmointirajapinnan kehityksen hyötyjen ymmärtämiseksi olennainen asia on, että rajapinta vastaa kaikkiin kutsuihin, jotka sille tulee. Rajapinnan näkökulmasta ei ole siis väliä tehdäänkö ne selaimella, mobiililaitteelle kehitetyllä sovelluksella tai komentorivillä, sillä se mitä palautetulla resurssilla tehdään, jää asiakassovelluksen vastuulle. (RESTful Web Services - Statelessness 2016.) Tästä huomataan miksi tilattomuus ja asiakkaan ja palvelimen erottaminen toisistaan ovat niin olennaisia REST-arkkitehtuurille. Sovelluksen eri osia ei voitaisi toteuttaa irrallisina toisistaan ilman näitä ominaisuuksia.

Vaikka rajapinta vastaa kaikkiin sille tehtyihin kutsuihin, voidaan vastauksena saatua tietoa rajoittaa. Opinnäytetyön tuloksia käsittelevässä osassa tullaan tarkastelemaan tilannetta, jossa halutaan rajoittaa rajapinnan vastauksia, että vain asianmukaisen tunnistautumisen tehnyt asiakasohjelma saa halutut tiedot rajapinnan päätepisteeltä. Tällaisessa tapauksessa ilman tunnistautumista tehdylle pyynnölle palautetaan vastaus HTTP-tilakoodi 401, joka tarkoittaa evättyä pääsyä resurssiin.

3.4 Tietokannat

Tietokannalla tarkoitetaan tämän opinnäytetyön yhteydessä elektronista tietokantaa eli jollakin tavalla järjestettyä tiedon pysyvää säilytyspaikkaa sähköisessä muodossa (Oracle 2022). Hyvin yksinkertainen tietokanta saavutetaan jo esimerkiksi tallentamalla tekstitiedostoon tietoa tiettyyn järjestykseen. Esimerkki tällaisesta tietokannasta voisi olla, vaikka henkilöiden tietojen tallennus: Yhden henkilön tiedot rivillä puolipistein erotettuna.

```
Esimerkki Henkilö1;050-3214xxx;Ryhmä 1;QA street 5
Esimerkki Henkilö2;050-123xxxx;Ryhmä 2;Testaajankatu 3
Esimerkki Henkilö3;050-567xxxx;Ryhmä 3;Esimerkkikatu 7
Esimerkki Henkilö4;050-9012xxx;Ryhmä 3;Kadunkulma 15
Esimerkki Henkilö5;050-3214xxx;Ryhmä 3;Tie 5
Esimerkki Henkilö6;050-1231xxx;Ryhmä 1;Alikulkutunneli 3
```

Kuva 4. Yksinkertainen henkilötietokanta tekstitiedostossa

Kuvan 4 tekstitiedostoon luodussa tietokannassa on monia sen käyttämiseen liittyviä ongelmia. Tiedon määrän kasvaessa tietokannan yksi perustoiminnoista, tiedonhaku, hidastuu valtavasti, sillä pahimmassa tapauksessa tietoa hakeva ohjelma joutuu käymään kaikki rivit läpi etsiessään haluttua tietoa. (Tietokantojen perusteet 2022.) Tämä ei ole välttämättä ongelma muutaman sadan alkion tietokannassa, mutta tietokannan kasvaessa tuhansien alkioiden kokoiseksi, alkaa olla hidasta käydä läpi tuhansia rivejä tekstejä jokaisen pyynnön yhteydessä. Toinen suuri ongelma liittyy tiedon tallentamiseen. Samanaikainen tekstitiedostoon tallentaminen voi aiheuttaa odottamattomia asioita, tietoa voi hävitä, sitä saatetaan kirjoittaa päällekkäin. Lisäksi tallentamisessa ja tiedon muokkauksessa törmätään jälleen tämänkaltaisen tiedon tallentamisen suurimpaan ongelmaan, hitauteen. (Tietokantojen perusteet 2022.)

Yllä kuvattujen ja muiden tiedon pysyvään tallentamiseen liittyvien ongelmien ratkaisemiseksi on kehitetty erilaisia tietokantajärjestelmiä. Tässä Opinnäytetyössä käydään läpi tarkemmin kahta tämän hetken suosituinta järjestelmää, relaatiotietokantaa, sekä dokumenttitietokantaa. Dokumenttitietokanta on osa tietokantojen joukkoa, jota kutsutaan NoSQL-tietokannoiksi. NoSQL viittaa relaatiotietokannoissa yleisesti kyselykielenä käytettyyn Structured Query Language (*SQL*). NoSQL on kuvaava nimi tälle tietokantojen joukolle, sillä tälle tietokantajoukolle on tyypillistä, että tietojen tallennukseen käytetään muuta muotoa, kuin relaatioihin perustuvia tauluja. (What is NoSQL? s.a.)

3.4.1 Relaatiotietokannat

Relaatiotietokantojen toiminta perustuu tauluihin, jotka muodostuvat niille määritellyistä sarakkeista. Tieto tallennetaan rivinä, joka sisältää sarakkeissa määritellyn tyyppistä tietoa. (Peterson 2022.)

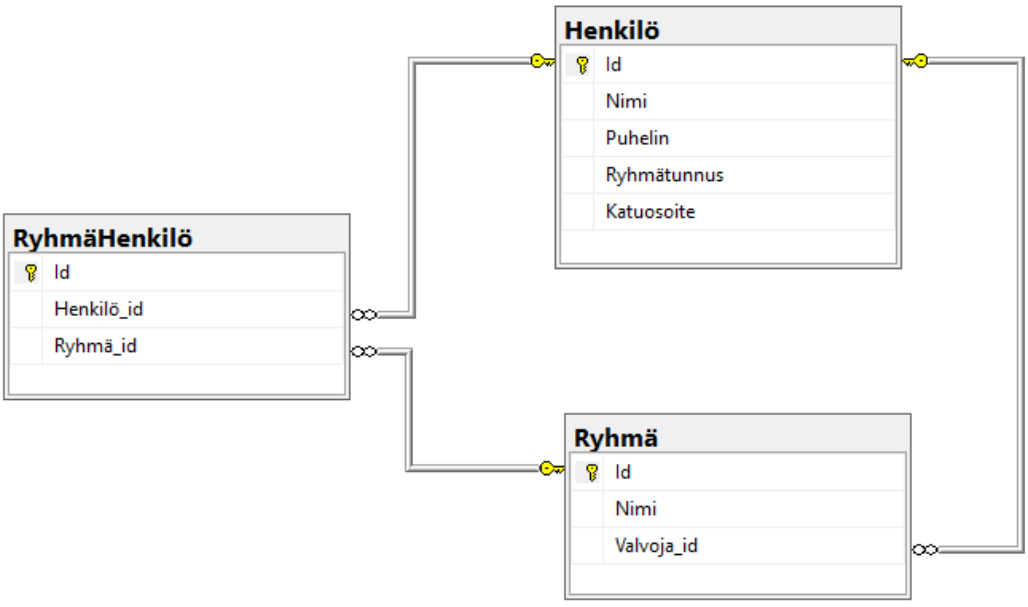
Henkilö

Column Name	Data Type	Allow Nulls
Id	int	☐
Nimi	nvarchar(50)	☐
Puhelin	nvarchar(50)	☐
Ryhmätunnus	nvarchar(50)	☑
Katuosoite	nvarchar(50)	☑
		☐

Kuva 5. Relaatiotietokannan taulu visualisoituna

Kuvassa 5 näkyy visualisoitu relaatiotietokannan taulu, johon tallennetaan samat tiedot sisältäviä rivejä, kuin aiemmin esitetyssä esimerkissä tekstitiedostona tallennetussa tietokannassa. Relaatiotietokannoille tyypillisesti taulussa on ensimmäisenä tieto tietorivin yksilöivä tunniste, ”Id”. Tämä tunniste toimii yleensä perusavaimena, jonka avulla kyseinen rivi voidaan tunnistaa, vaikka sen muut sarakkeet sisältäisivätkin samaa tietoa. (Peterson 2022.)

Itse relaatiomalli tulee esiin, kun siirrytään yhdestä taulun kannasta useita, toisiin tauluihin liittyvää tietoa sisältäviä tauluja käyttävään, relaatioita sisältävään tietokantaan. Relaatiot eli taulujen väliset suhteet ovat yleensä joko yksi moneen, tai monta moneen -suhteita. Yksi moneen -suhde tarkoittaa, yksinkertaisesti sitä, että taulun A rivi viittaa enintään yhteen taulun B riviin. Taulun B rivi sen sijaan voi tässä suhdemallissa liittyä moneen taulun A riviin. (Fernigrini 2021.) Monta moneen -suhteessa sekä taulun A että taulun B rivit voivat liittyä useisiin toistensa riveihin (Babic 2020). Teknisesti yksi moneen -suhde toteutetaan siten, että tauluun A lisätään sarake, joka sisältää viittauksen tauluun B (Fernigrini 2021). Monta moneen -suhdetta ei voida tietokannan näkökulmasta toteuttaa järkevästi ilman liitostaulua, sillä tämä tarkoittaisi datan tarpeetonta kopioimista moneen kertaan. Liitostaulu tarkoittaa ylimääräistä taulua, joka sisältää vain viitteet molempiin tauluihin, joille monta moneen -suhde halutaan muodostaa. (Babic 2020.)



Kuva 6. Yksi moneen ja monta moneen suhde

Kuvassa 6 esitetään taulujen Henkilö ja ryhmä monta moneen -suhde, sekä samojen taulujen yksi -moneen suhde. RyhmäHenkilö-taulu on liitostaulu, joka sisältää viitteet sekä Henkilö-taulun ja Ryhmä-taulun yksilöiviin tunnisteisiin. Näin saavutetaan tilanne, jossa tietokannan henkilö voidaan liittää useaan ryhmään ja ryhmään voidaan liittää useita henkilöitä. Lisäksi ryhmällä on valvoja. Tämä on toteutettu yllä olevassa kuvassa yksi moneen -suhteen avulla. Ryhmä-taulu sisältää sarakkeen Valvoja_id, joka on viite Henkilö-taulun yksilöivään tunnisteeseen. Näin voimme toteuttaa tilanteen, jossa Henkilö voidaan liittää useiden ryhmien valvojaksi, mutta ryhmällä voi olla vain yksi valvoja.

3.4.2 Dokumenttitietokannat

Aiemmin esitellyn NoSQL-termin alle mahtuu monenlaisia tietokantatyyppejä. NoSQL-perheen laajuuden vuoksi tarkastelemme tässä opinnäytetyössä vain dokumenttitietokantoja, sillä se on varteenotettava vaihtoehto relaatiotietokannalle vastaamaan tämän kehitystutkimuksen tarpeisiin.

Dokumenttitietokannan keskeisenä ajatuksena on tallentaa tieto dokumentteihin taulujen asemesta. Nämä dokumentit muistuttavat rakenteeltaan HTTP-protokollaa, REST-arkkitehtuurimallia ja ohjelmointirajapintaa käsittelevissä luvuissa mainittua JSON-olioita. Niihin voidaan siis tallentaa tietoa vapaammin kuin relaatiomallin tauluihin, sillä tietokannassa voidaan käyttää monista ohjelmointikielistä tuttuja muuttujatyyppejä, kuten totuusarvoja, taulukoita ja olioita eli olio-ohjelmointiparadigmassa käytettyjen luokkien ilmentymiä. Tämä voidaankin nähdä yhtenä suurimmista dokumenttitietokannan etuna verrattuna relaatiomalliin, tietokannan käyttö ohjelmallisesti on luontevampaa, sillä käytetty data voidaan tallentaa tietokantaan samanlaisessa muodossa, jossa sitä käsitellään ohjelmakoodissa. (What is NoSQL? s.a.)

```
{
    "Id":1,
    "Nimi": "Ryhmä1",
    "Valvoja": {
        "Id": 1,
        "Nimi": "Vilma Valvoja",
        "Puhelin": "050-123134",
        "Osoite": "Testaajankatu 3",
    },
    "Jäsenet": [
        {
            "Id": 2,
            "Nimi": "Testi Teppo",
            "Puhelin": "050-123134",
            "Osoite": "Testaajankatu 4",
        },
        {
            "Id": 3,
            "Nimi": "Esko Esimerkki",
            "Puhelin": "050-123134",
            "Osoite": "Esimerkkitie 3",
        },
        {
            "Id": 4,
            "Nimi": "Erin Esimerkki",
            "Puhelin": "050-123134",
            "Osoite": "Esimerkkikuja 5",
        }
    ]
}
```


Kuva 7. Esimerkki dokumenttitietokantaan tallennetusta dokumentista

Vertaillaan relaatiotietokannan ja dokumenttitietokannan eroja kuvassa 7 esitellyn dokumentin avulla. Relaatiotietokantoja käsittelevässä aliluvussa käytiin läpi, kuinka tietoa tallennetaan ja jäsennellään tauluihin, jotka sisältävät tauluun liittyvää dataa. Dokumenttitietokantaa käytettäessä voidaan monimutkaisiakin tietorakenteita tallentaa yksittäisinä esiintyminä. Kuten kuvasta kuusi huomaamme, kaikki yhteen ryhmään liittyvät tiedot on tallennettu yksittäisen objektin sisälle. Yksilöivä tunniste ”Id”, sekä ryhmän nimi ovat samankaltaisesti avain-arvo-parina kuten relaatiomallissakin. Erot huomataan, kun tarkastellaan objektin avaimia ”Valvoja” sekä ”Jäsenet”. Relaatiomallissa Ryhmän valvoja tallennettiin viitteenä toiseen tauluun. Dokumenttitietokannassa tieto voidaan tallentaa suoraan toisena objektina ”isäntäobjektin” sisälle. Ryhmän jäsenten tallentaminen tuo eron vielä paremmin esille. Relaatiomallissa yksi moneen -suhteen saavuttamiseksi luotaisiin yhtä monta käyttäjätaulun riviä, joille annettaisiin viite ryhmätaulun riviin, kun JSON-dokumentteja tallennettaessa voidaan henkilöjen tiedot tallentaa suoraan isäntäobjektin sarakkeeseen olioita sisältävänä taulukkona. (What is NoSQL? s.a.)

3.5 Datansiirtokerros

Datasiirtokerros viittaa tässä opinnäytetyössä ohjelman osaan, joka toimii välittäjänä tietokannan ja muun sovelluslogiikan välillä. Datansiirtokerroksen tarkoitus on hoitaa kaikki kommunikaatio sovelluksen käyttämien tietokantojen välillä ja yksinkertaistaa näin muun sovelluksen toimintaa. Näin sovelluslogiikan ei tarvitse ottaa kantaa siihen, mitä tietokantaa käytetään tai kuinka itse tietokannalle tehtävät pyynnöt on toteutettu. Sovellus kutsuu datansiirtokerroksen funktioita tarpeen mukaan, ja siirtokerroksesta palautetaan tarvittava tieto. Erillisen datansiirtokerroksen toteuttaminen helpottaa ohjelman ylläpitoa. Kun tietokanta ja sen käsittely on eriytetty omaksi osaksi, voidaan tietokantaa vaihtaa tai päivittää vaivattomammin, sillä tarvittavat muutokset voidaan tehdä vain yhteen paikkaan. (Data-Access Layer 2022.)

3.6 Monivaiheinen tunnistautuminen

Monivaiheisella tunnistautumisella tarkoitetaan nimensä mukaisesti useassa eri vaiheessa tapahtuvaa tunnistautumista. Perinteinen tunnistautuminen tapahtuu usein yhdessä vaiheessa: käyttäjä syöttää tunnistautumistiedot ja niiden ollessa oikein, tunnistautuminen onnistuu. Monivaiheisessa tunnistautumisessa tämän lisäksi tai asemesta käytetään muita tapoja, jotta minimoidaan mahdollisuus, että tunnistautumassa olisi väärä toimija.

Vaiheet jakautuvat kolmeen kategoriaan: Johonkin mitä käyttäjä tietää, esimerkiksi salasana, salauslause tai pin-koodi. Jotain mitä käyttäjällä on hallussaan, esimerkiksi usb-muistilaitteelle tallennettu salausavain tai puhelimella käytettävä tunnistautumissovellus. Kolmas kategoria on käyttäjä itse, eli sormenjälki- ja kasvojentunnistus tai jokin muu biometrinen tunnistautuminen. Monivaiheisen tunnistautumisen antama lisäturva perustuu siihen, että vaikka jokin taho olisi saanut kirjautumistietosi haltuun, tällä on oltava kaikki tunnistautumisen vaiheet hallussaan, jotta kirjautuminen onnistuu. (Monivaiheinen tunnistautuminen suojaa käyttäjätilejäsi 2022.)

Seuraavassa, kehittämistutkimuksen toteutusta käsittelevässä luvussa käydään läpi laajemmin todennussovellusta hyödyntävän tunnistautumismenetelmän toimintaa ja sen eri osien toteutusta.

3.7 Salausmenetelmät ja hajautus

Vaikka kehitystyössä ei ole tarvetta toteuttaa salaus, tullaan käyttäjätietojen tietoturvallisen tietokantaan tallennuksen ja kaksivaiheisen tunnistautumisen toteutuksen yhteydessä vääjäämättä tilanteeseen, jossa on tarpeen käyttää jotakin salausmenetelmää ja hajautusalgoritmia. Tästä syystä tässä aliluvussa käsitellään yleisellä tasolla hajautusalgoritmeja, sekä kahta eri salauksen muotoa.

Hajautusalgoritmit

Hajautusalgoritmilla tai hajautusfunktiolla tarkoitetaan prosessia, joka pienentää sille syötetyn datan lyhyempään, käytetystä algoritmista riippuvaan, pituuteen. Hajautusalgoritmit voidaan kahteen tyyppiin, salaista avainta käyttäviin ja avaimettomiin algoritmeihin. Avainta käyttäviä hajautusalgoritmien antamia arvoja kutsutaan yleisesti nimellä message authentication code. Avaimettomia algoritmeja taas kutsutaan yhden suunnan algoritmeiksi (Ganesan & Sobti 2012, 2–3.) Molempien muotojen yksi ominaisuus onkin, että algoritmilta saatua arvoa on lähes mahdoton purkaa enää alkuperäiseksi syötteeksi. Lähes ainut tapa saada selville alkuperäinen viesti onkin syöttää algoritmille arvoja ja verrata saatua hajautusarvoa olemassa olevaan arvoon. (Cryptography 2022.)

Hajautusalgoritmeilla muodostettuja arvoja kutsutaan hajautusarvoiksi. Hajautusarvon avulla voidaan esittää data huomattavasti lyhyemmässä muodossa. Hajautusarvoja voidaan käyttää esimerkiksi tiedon oikeellisuuden tarkistamiseen. Alkuperäiselle tiedolle lasketaan hajautusarvo ja esimerkiksi tiedon kopioinnin yhteydessä kopion oikeellisuus voidaan tarkistaa laskemalla kopiosta hajautusarvo ja vertaamalla sitä alkuperäisestä tiedosta laskettuun arvoon. (Ensuring Data Integrity with Hash Codes 2021.)

Epäsymmetrinen salaus

Epäsymmetrisessä salauksessa käytetään kahta avainta, julkista ja salaista. Kohdetieto salataan julkisen avaimen avulla ja salaus puretaan sitä vastaavalla salaisella avaimella. Tyypillinen käyttö tälle salaustavalle on salausta vaativien tiedostojen lähetys. Käyttäjä lähettää julkisen avaimensa toiselle

käyttäjälle, jonka avulla toinen käyttäjä salaa tiedon ja lähettää salatun tiedon alkuperäiselle käyttäjälle, joka pystyy purkamaan julkisella avaimella salatun tiedon salaisella avaimellaan. (Public key cryptography 2021.)

Symmetrinen salaus

Symmetrinen salaus on salauksen muoto, jossa salaaminen ja salauksen purku tapahtuu samalla avaimella. Kuka tahansa, jolla on tiedossa käytetty salausalgoritmi ja salausavain voi purkaa symmetrisesti salatun viestin. (Cryptography 2022.) Symmetristä salausta kutsutaan myös salaisen avaimen salaukseksi, sillä toisin kuin aiemmin esitellyssä epäsymmetrisessä salauksessa, symmetrisissä salausmuodoissa ei käytetä mitään julkisia avaimia.

4 TULOKSET

Tässä luvussa käsitellään teoriaosuuteen pohjautuvan kehitystyön tuloksia. Osuudessa tarkastellaan ensin tunnistautumista ja käyttöoikeuksien hallintaa kokonaisuutena, sen toimintaperiaatetta ja kuinka se liittyy osaksi olemassa olevaa sovellusta. Tämän jälkeen esitellään toteutettujen osien valittuja teknologioita ja näiden suhdetta teoreettiseen viitekehykseen. Teknologioiden läpikäyminen aloitetaan tietokannasta, jotta saadaan käsitys siitä, minkälaista tietoa rajapinnalta pyydetään. Tämän jälkeen sovelluksen toiminta käydään käyttäjän näkökulmasta loogisessa järjestyksessä: ensin asiakasohjelman osat, sovellukseen kirjautuminen ja lopuksi oikeuksien hallinta.

Sovelluksen toiminta perustuu eri kokonaisuuksiin ja niiden vuorovaikutukseen. Keskiöissä ovat projektit, käyttäjäryhmät, käyttäjät, näille asetettavat roolit ja niiden käsittely eri sovelluksen osissa.

Käyttäjän näkökulmasta sovellus on yksinkertainen: ensin kirjaudutaan sisään, jonka jälkeen käyttäjälle listataan projektit, joihin hänellä on jonkintasoinen oikeus. Käyttäjä valitsee listasta projektin, jonka parissa haluaa työskennellä. Projekteja voi katsoa, kopioida tai niihin voi tehdä muutoksia ja palauttaa muutokset palvelimelle, riippuen käyttäjän oikeuksista.

Yllä kuvailtuun toiminnallisuuteen päästään siten, että projekteihin liitetään käyttäjäryhmiä, joille asetetaan projektikohtainen rooli, joka määrää tietyt käyttöoikeudet kyseiseen projektiin. Ryhmiin taas lisätään halutut henkilöt, jotka pystyvät tehdä asiakassovelluksella ryhmän roolin mukaisia toimintoja. Ryhmäkohtaisten roolien lisäksi projekteille voidaan asettaa käyttäjäkohtaisia rooleja samalla logiikalla. Käyttäjällä voi olla projektissa sekä ryhmäkohtaiset että käyttäjäkohtaiset roolit. Tällaisessa tapauksessa käyttäjän henkilökohtainen rooli ajaa muiden roolien edelle. Tämän toiminnallisuuden käytännön toteutukseen paneudutaan käyttäjätietokannan toteutusta käsittelevässä aliluvussa.

Käyttäjällä on kirjautumiseen kaksi vaihtoehtoa, joko käytetään erikseen sovellukseen luotuja tunnuksia tai kirjaudutaan Windows-käyttäjätunnuksilla. Jälkimmäinen tapa vaatii sen, että replikointipalvelin ja käyttäjä toimivat samalla Windows-käyttäjäalueella. Jos käytetään Windows-tunnuksia, haetaan käyttäjälle ensimmäisen kirjautumisen yhteydessä Active Directory -ryhmät, joiden jäsen käyttäjä on, jos niille löytyy vastaavat replikointisovelluksesta.

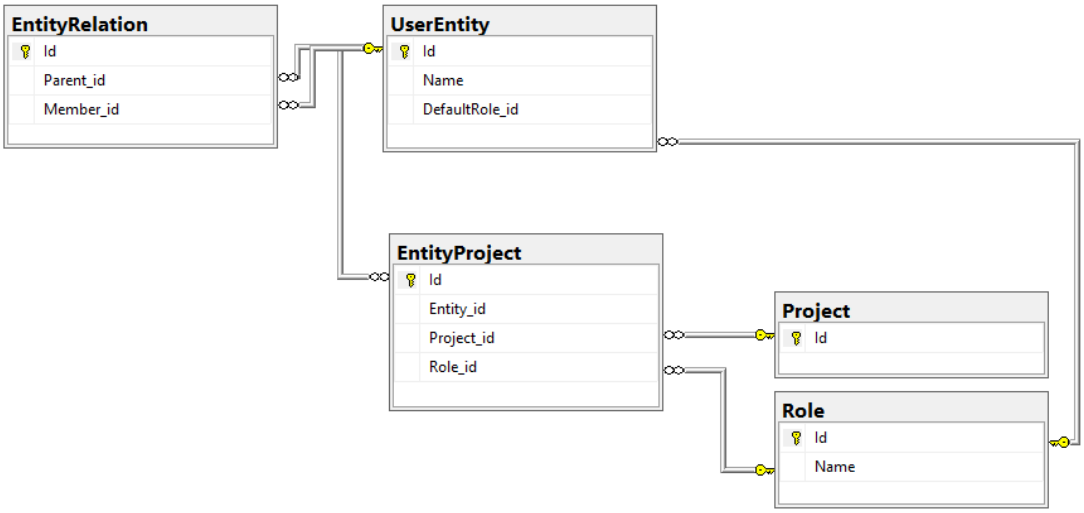
4.1 Käyttäjätietokanta

Jotta tunnistautuminen ja tunnistautuneen käyttäjän oikeuksien hallinta olisi mahdollista, täytyi luoda tietokanta. Käyttäjätietokannaksi valikoitui teoriaosuudessa esitellyistä tietokantamalleista relaatiomalli. Toiminnallisuus olisi ollut varsin mahdollista toteuttaa myös dokumenttitietokannan avulla, mutta seuraavassa kappaleessa esiteltävät syyt puolsivat relaatiomallia käyttävän Microsoft SQL Server -tietokannan käyttämistä tässä kehitystyössä.

Sovelluskehitys lähtee harvoin liikkeelle täysin tyhjästä. Usein rakennetaan toiminnallisuutta jonkin olemassa olevan sovelluksen tai sen osan päälle ja niin on myös tämä kehitystutkimuksen tapauksessa. Monissa tapauksissa sovelluksessa jo käytössä olevat teknologiat rajaavat huomattavasti valikoimaa, mitä on järkevä käyttää myös tulevissa kehityskohteissa. Replikaatiotyökalussa oli tämän opinnäytetyön alkaessa käytössä Microsoftin SQL Server -tietokanta, johon tallennetaan erilaista tietoa sähkösuunnitteluprojekteista. Tämä ei tietenkään olisi estänyt jonkin muun tietokannan tai kantamallin käyttöönottoa käyttäjätietokannaksi. Se olisi kuitenkin vaatinut toisen datansiirtokerrok-

sen luomisen jo olemassa olevan SQL Server -tietokannan kanssa yhteensopivan kerroksen lisäksi. Tätä ei lähdetty toteuttamaan, sillä dokumenttitietokannan käyttöönotosta olisi tässä kehitystyössä saatu hyötyä lähinnä siinä tapauksessa, jos tietokantarakenteeseen olisi tarvetta tehdä muutoksia siinä vaiheessa, kun tärkeää dataa on jo viety tietokantaan ja olisi tarve taulujen suhteiden välisiin muutoksiin. Tämä hyöty saavutetaan NoSQL-tietokannoissa siitä syystä, että dataa voi tallentaa huomattavasti vapaammassa muodossa kuin relaatiomalliin perustuvissa kannoissa. (What is NoSQL? s.a.) Käyttäjätietokannan tapauksessa kyse on kuitenkin niin selkeästä käyttökohteesta, että tarvetta relaatioihin vaikuttaviin muutoksiin ei herkästi ilmaannu.

Toinen suuri etu, on se, että sähkösuunnitteluprojekteja sisältävällä palvelimella on jo oltava Microsoft SQL Server -tietokanta, joten tämä ei luo tarvetta asentaa enempää kolmannen osapuolen järjestelmiä palvelimelle.



Kuva 8. Käyttäjätietokannan taulut ja niiden väliset suhteet

Kuvassa 8 esitetään lopullisen käyttöön tulevan käyttäjätietokannan rakenne. Taulujen nimet eivät ole tuotantoversiossa samat, eikä kuvasta löydy tietoa sisältäviä sarakkeita, tarkoitus on esittää taulujen väliset suhteet. Käyttöoikeuksien hallinta rakentuu UserEntity-, Project- ja Role-taulujen ja niiden relaatioiden ympärille. UserEntity-tauluun tallennetaan tietoa sekä käyttäjistä, että ryhmistä: sovelluksen toimijoita. Ryhmää ja käyttäjää ei tietokannassa erotella toisistaan, sillä ne sisältävät hyvin samankaltaista tietoa. Project-taulun rivit

kuvaavat yksittäistä sähkösuunnitteluprojektia käyttäjätietokannassa. Role-taulun rivit taas sisältävät tiedon käyttäjän roolista.

Projektikohtaisten roolien asettaminen toimijalle tapahtuu aiemmin tietokantojen teoriaa käsittelevässä luvussa esiteltyjen liitostaulujen, yksi moneen ja monta moneen -suhteiden avulla. Kuten kuvasta seitsemän näemme, on UserEntity- ja Project-taulujen välillä monta moneen -suhde, joka on toteutettu EntityProject-liitostaulun avulla. Lisäksi EntityProject-taulussa määritellään toimijan rooli projektissa Role_id-viiteavaimen avulla. Näiden avulla saavutetaan haluttu tilanne, jossa projektiin voi liittyä monta toimijaa ja toimijaan monta projektia.

Toimijoiden lisääminen käyttäjäryhmiin on toteutettu vastaavalla tavalla: EntityRelation-liitostaulun avulla. Liitostaulu sisältää viiteavaimet Parent_id ja Member_id. Molemmat viittaavat UserEntity-taulun Id-sarakkeeseen Member_id viittaa toimijaan, joka on alisteinen Parent_id-viittauksen mukaiseen toimijaan. Näin kykenemme luomaan Toimijoita, joilla voi olla alitoimijoita.

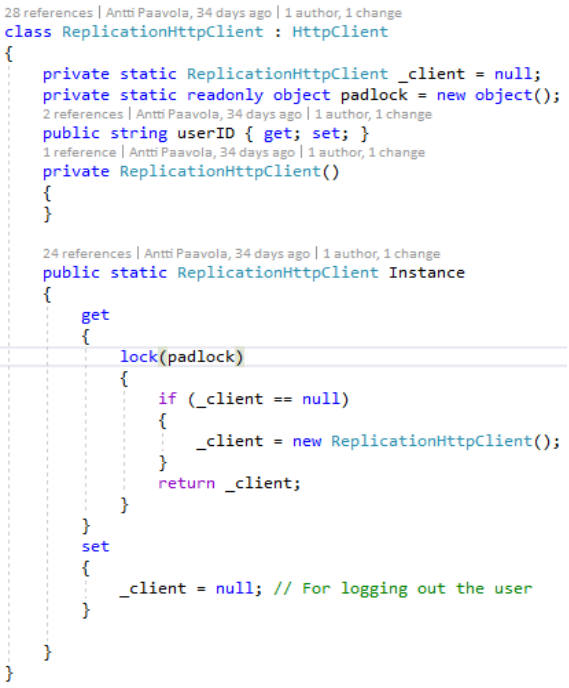
4.2 Asiakasohjelma

Vaikka kirjautuminen ja oikeuksien hallinta toteutetaan pääasiallisesti rajapinnan puolella, on hyvä tutustua myös asiakasohjelman toteutukseen, jotta ymmärretään paremmin kirjautumisen, oikeuksien tarkistamisen ja samalla HTTP-kutsun lähettämisen ja vastauksen saamisen koko prosessi. Asiakasohjelma on tämän kokonaisuuden ensimmäinen vaihe, eli HTTP-kutsun tekijä.

HttpClient

HttpClient on Microsoftin tarjoama valmis luokka HTTP-kutsujen tekemiseen ja HTTP-vastausten vastaanottamiseen. HttpClient, toimii opinnäytetyön teoriaosuudessa esiteltyjen HTTP-protokollan periaatteiden mukaan. Sille annetaan osoite eli URI, johon kutsu tehdään, sekä mahdolliset kutsun mukana menevät otsikko- ja sisältörivit. HttpClient-luokasta on tarkoitus luoda yksi esiintymä ohjelman suorituksen ajaksi sen sijaan, että jokaiselle kutsulle luotaisiin oma esiintymä. Jos jokaiselle kutsulle luodaan uusi esiintymä, pahimmassa tapauksessa voidaan törmätä SocketException-virheeseen, joka tarkoittaa, että käytössä olevat pistokkeet ovat loppuneet. (HttpClient Class s.a.)

Pistokkeella tarkoitetaan tässä yhteydessä yhtä päätepistettä palvelimen ja asiakasohjelman välisessä kommunikaatiossa, joka on sidottu URI-tunnisteessa käytettyyn porttiin (What is a Socket? s.a). Lisäksi saavutamme sovelluksen käyttöä ajatellen suotuisan tilanteen, jossa asiakassovelluksen täytyy kirjautua sisään vain kerran, kun tunniste pysyy HttpClient-esiintymän otsikkorivillä, koko sen elinajan.



Kuva 9. HttpClient-luokan Singleton-toteutus

Kuvassa 9 esitellään kehitetyn asiakassovelluksen tapa toteuttaa yhden HttpClient-esiintymän periaate. Asiakassovellukselle luotiin uusi, Singleton-mallia noudattava, ReplicationHttpClient-luokka, joka perii HttpClient-luokan. Kuvan 14 toteutus on säieturvallinen, säieturvallisuus tarkoittaa käytännössä, ettei kahdesta eri säikeestä voida luoda esiintymää luokasta, jos sellainen on jo olemassa. Säieturvallisuus on saavutettu asettamalla lukko _client-muuttujan arvioinnille. Lukko varmistaa, että sen sisällä olevaa ohjelmakoodia voi suorittaa vain yksi säie kerrallaan. (Implementing the Singleton Pattern in C# s.a.) Singleton on suunnittelumalli, jolla voidaan varmistaa, että luokalla on sovelluksessa vain yksi esiintymä, jota voidaan käyttää mistä tahansa sovelluksen osasta. Singleton toteutetaan yksinkertaisesti: luokalle asetetaan staattinen muuttuja, kuvassa 16 _client, sekä staattinen metodi esiintymän luomiseen ja palauttamiseen (Singleton s.a.) Kuvan 9 Instance-metodissa luodaan

uusi ReplicationHttpClient-esiintymä ja asetetaan se _client-muuttujan arvoksi, jos muuttuja on tyhjä, muuten palautetaan jo olemassa oleva esiintymä.

4.3 Rajapinta ja tunnistautuminen

Alaluvun tarkoituksena on avata REST-arkkitehtuurimalliin perustuvan ohjelmointirajapinnan käytännön toteutusta ja siihen käytettyjä teknologioita

ASP.NET Core

ASP.NET Core on Microsoftin kehittämä ja ylläpitämä avoimeen lähdekoodiin perustuva ohjelmistokehitysrunko. ASP.NET sisältää valmiita työkaluja web-sovelluskehitykseen sekä käyttöliittymäpuolella että taustajärjestelmäpuolella. (Overview to ASP.NET Core 2022.) Tässä kehitystyössä keskitytään REST-arkkitehtuurimallia noudattavan rajapinnan kehitykseen ASP.NET Core -kehyksen työkaluilla.

C#

Kehitystä tehdään C#-ohjelmointikielellä. C# on Microsoftin kehittämä olio-ohjelmointiparadigmaa tukeva käännettävä ohjelmointikieli ja sitä suoritetaan .NET suoritusympäristössä (A tour of the C# language 2022).

Toteutetun rajapinnan toiminta perustuu REST-arkkitehtuurimallin mukaisesti määriteltyihin päätepisteisiin ja datansiirtokerrokseen, joiden yhteistoiminnalla palautetaan resursseja sen perusteella, mihin päätepisteeseen pyyntö tehtiin ja mitä sen mukana lähetettiin. Lisäksi palautukseen vaikuttaa, se onko pyynnön tekijällä oikeutta saada pyydettyä resurssia.

Tarkastellaan käytännön toteutusta esimerkin kautta, jossa asiakasohjelmalla tehdään tunnistautumispyyntö käyttäen käyttäjänimi-salasana-yhdistelmää.

```
var result = client.PostAsync("api/authenticate", contentRequest, cancellationToken).Result;
```

Kuva 10. Asiakasohjelman tunnistautumispyyntö

Ensimmäinen vaihe on tehdä asiakasohjelmassa pyyntö palvelimelle. Kuvassa 10 näemme, kuinka ohjelmakoodissa tehdään POST-muotoinen HTTP-

pyyntö päätepisteen ”api/authenticate”, ja sen mukana lähetetään contentRequest-muuttuja, joka sisältää käyttäjän tunnukset.

```
[AllowAnonymous]
[HttpPost]
[Route("/api/authenticate")]
0 references | Antti Paavola, 24 days ago | 1 author, 3 changes
public async Task<IActionResult> Login([FromBody] LoginModel model)
{
    var user = await _userService.Authenticate(model.Username, model.Password);
    if (user != null)
    {
        JwtSecurityToken token = await GetSecurityToken(user);

        return Ok(new
        {
            token = new JwtSecurityTokenHandler().WriteToken(token),
            expiration = token.ValidTo,
            user_id = user.Id
        });
    }
    return Unauthorized();
}
```

Kuva 11. Rajapinnan kirjautumisen käsittelijäfunktio

Kuvassa 11 näemme rajapintaan toteutetun funktion, joka vastaa edellisessä vaiheessa tehtyyn HTTP-kutsuun. Kuvan yläreunassa nähtävä merkintä [AllowAnonymous] mahdollistaa päätepisteen kutsumisen ilman tunnistautumista, kun muut rajapinnan päätepisteet sen vaativat. Merkintä [HttpPost] taas määrittää HTTP-pyynnön muodon ja estää muiden tyyppisten kutsujen tekemisen kyseiseen päätepisteeseen.

Yllä kuvailtu funktio vastaanottaa pyynnön mukana lähetetyn datan, tässä tapauksessa LoginModel-tyyppisen käyttäjänimen ja salasanan sisältävän olion vastaavat kentät JSON-muodossa. Funktio käyttää hyödykseen rajapintaan määriteltyä datansiirtokerrosta kutsumalla UserService-luokassa määriteltyä Authenticate-metodia, joka kommunikoi tietokannan kanssa ja tarkistaa löytyykö lähetettyjen tunnusten mukaista käyttäjää. Jos käyttäjä löytyy, luodaan sille allekirjoitettu tunniste, JSON Web Token, jota hyödynnetään muissa rajapinnalle tehtävissä kutsuissa. Tunnisteen sisältöön ja sen käyttöön tutustutaan tarkemmin oikeuksien hallintaa käsittelevässä aliluvussa.

Tunniste palautetaan OK-tilakoodin kanssa. Jos käyttäjää ei löydy tai salasana on väärä, palautetaan Unauthorized-tilakoodi.

```
var resultString = result.Content.ReadAsStringAsync().Result;
var responseData = JsonConvert.DeserializeObject<IDictionary<String, String>>(resultString);
var access_token = responseData["token"];
var user_id = responseData["user_id"];
```

Kuva 12. HTTP-vastauksen käsittely

Onnistunut vastaus luetaan kuvassa 12 merkkijononoksi ja muunnetaan Json-Convert-luokasta löytyvällä metodilla avain-arvo-pariksi. Tämän jälkeen vastauksen eri arvoihin päästään käsiksi syntaksilla muuttuja[”avain”].

```
ReplicationHttpClient.Instance.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", access_token);
```

Kuva 13 Authorization-otsikkorivin asettaminen

Kuvassa 13 asetetaan asiakasohjelmassa käytettävälle HttpClient-oliolle Authorization-otsikkorivin arvoksi rajapinnasta palautettu tunniste. Tämä on tärkeä osa tunnistautumista, sillä asiakasohjelma käyttää kaikissa muissa kutsuissa tätä tunnistetta käyttöoikeuksien varmistamiseen. Myös HttpClient-luokan toimintaa käydään läpi tarkemmin seuraavassa, oikeuksien hallintaa käsittelevässä aliluvussa.

Windows-tunnistautuminen

Intranet-käyttöön suunniteltu Windows-tunnuksilla kirjautuminen eroaa tavallisesta käyttäjänimi-salasana-parilla tunnistautumisesta siltä osin, että päätepisteeseen tehtävän pyynnön mukana ei lähetetä käyttäjän lomakkeeseen kirjaamia tunnuksia, vaan käyttöjärjestelmältä haetut tunnistustiedot.

```
// Create new HttpClientHandler with windows-credentials (username, password, domain)
//
NetworkCredential winCredentials = CredentialCache.DefaultNetworkCredentials;
HttpClientHandler authHandler = new HttpClientHandler() { Credentials = winCredentials, PreAuthenticate = true };
var client = new HttpClient(authHandler);
```

Kuva 14. Windows-tunnistautumiseen käytettävä HttpClient-olio

Kuvassa 14 haetaan käyttöjärjestelmältä sen hetken käyttäjän toimialueen mukaiset tunnistustiedot, jotka ovat tallessa CredentialCache-luokan Default-NetworkCredentials-muuttujassa. Aiemmin esitellystä tunnistautumisesta poiketen HttpClient-oliolle annetaan parametrina HttpClientHandler-olio, jonka

avulla tunnistustiedot saadaan lähetettyä oikeassa muodossa palvelimen rajapintaan. Palvelimen puolella eroina on, ettei käyttäjän tunnistusta tehdä ensisijaisesti käyttäjätietokannan käyttäjistä, vaan toimialueen Windows Active Directory -käyttäjistä.

```
[HttpPost]
[Authorize] // Windows authorization needed to call this endpoint
[Route("/api/authenticate/windows")]
0 references | Antti Paavola, Less than 5 minutes ago | 1 author, 1 change
public async Task<IActionResult> LoginWin()
{
    // Get authenticated windows-user's name from httpcontext
    string userName = HttpContext?.User?.Identity?.Name;
    if (String.IsNullOrWhiteSpace(userName))
        return Unauthorized();

    var userComponents = userName.Split('\\');
    var ctx = new PrincipalContext(ContextType.Domain);
    var userPrinciple = new UserPrincipal(ctx) { SamAccountName = userComponents[1] };
    var search = new PrincipalSearcher(userPrinciple);

    UserPrincipal pUser = (UserPrincipal) search.FindOne();

    if (pUser == null)
        return Unauthorized();



    var user = await _userService.GetUserWithName(userName);
    if (user == null || user.Username == null)
        user = await _userService.PostUser(createUser(pUser.Name, userName, pUser.UserPrincipalName, (int)RoleEnum.Viewer));

    // If user is found: get user's AD-groups and add user to corresponding Replication groups
    //
    if (user != null)
    {
        try
        {
            using (var adGroups = pUser.GetAuthorizationGroups())
            {
                List<string> groupNames = adGroups.Select(g => g.Name).ToList();
                var dbGroups = await _dbUserService.GetGroupsWithName(groupNames);
                List<int> userGroupIDs = dbGroups.Select(g => g.Id).ToList();
                int status = await _dbUserService.AddUserToGroups(user.Id, userGroupIDs);
            }
        }
        catch (Exception e)
        {
            _logger.LogError(e.Message);
        }

        // Create security token for logged-in user and return it
        //
        JwtSecurityToken token = await GetSecurityToken(user);
```

Kuva 15. Rajapinnan Windows-tunnistautumisen käsittelijä

Kuvan 15 käsittelijäfunktion [Authorize]-annotaatio varmistaa, ettei käsittelijää voi kutsua muut kuin oikeat tunnistustiedot lähettänyt käyttäjä Using the [Authorize] Attribute s.a). Jos käyttäjä löytyy Active Directory -tietokannasta, etsitään vastaava käyttäjä replikointisovelluksen käyttäjätietokannasta. Jos tätä ei löydy, luodaan Active Directory -käyttäjän tiedoilla tunnus tietokantaan. Tämän jälkeen palautetaan vastaava JSON Web Token -tunnus, kuten käyttäjänimi-salasana-parilla tehtävässä tunnistautumisessa, jota käytetään sovelluksen muiden kutsujen yhteydessä käyttäjän tunnistamiseen.

4.4 Kaksivaiheinen tunnistautuminen

Tässä kehitystyössä monivaiheinen tunnistautuminen toteutettiin kaksivaiheisena, ensimmäisenä vaiheena on perinteinen käyttäjänimi-salasana-pari ja

toisena vaiheena käytetään edellisessä luvussa esitellyistä kategorioista jotakin mitä omistan -joukkoon kuuluvaa aikaperustaista kertakäyttöistä salasanaa, johon tullaan tästä edespäin viittaamaan lyhenteellä TOTP (= Time-based one time password). TOTP-arvon käyttäjälle toimittamiselle on useita vaihtoehtoja. Suosituimpina näistä ovat tekstiviestin tai sähköpostin mukana lähetettävä salasana tai jotakin todennussovellusta hyödyntävä toteutus. SMS on käytettävyyden ja saavutettavuuden näkökulmasta hyvä vaihtoehto, sillä suurin osa käyttäjistä pystyy vastaanottamaan SMS-viestejä (5 reasons SMS 2FA isn't going away 2021). Sen heikkoutena on kuitenkin alttius monille hyökkäystavoille, kuten kalasteluviesteille ja puhelinnumeroväärennöksille (Elliot 2020). Nämä tietoturvariskit toimivatkin suurimpina syinä sille, että tässä kehitystyössä päädyttiin toteuttamaan todennussovellusta hyödyntävä kaksivaiheinen tunnistautuminen.

Kaksivaiheinen tunnistautuminen TOTP-arvon avulla perustuu siihen, että sama tunnus lasketaan sekä käyttäjän laitteella että palvelimella. Tämä on myös vahvuus tietoturvan näkökulmasta, sillä verkon yli ei vaihdeta mitään muuta tietoa kuin kertakäyttöinen lyhyen ajan voimassa oleva tunnus. Mitään salausavaimia ei siis tunnistautumisen yhteydessä tarvitse jakaa palvelimen ja käyttäjän kesken. (TOTP s.a.) Haavoittuvimmillaan ollaan siinä hetkessä, kun tunnistautuminen otetaan käyttöön ja käyttäjälle lähetetään jaettu salaisuus, jota käytetään myöhemmin kertakäyttöisten tunnusten laskemiseen. Tämän salaisuuden on oltava sama sekä palvelimella että käyttäjän laitteella, jotta voidaan laskea oikea tunnus kirjautumisen yhteydessä. Seuraavaksi käydään läpi sekä TOTP-arvon luominen, sen käyttö ja tarvittavien salausmenetelmien käyttö tietojen tallentamisessa.

TOTP-arvo luodaan käyttäen hyväksi HMAC-tunnukseen (*Hash-based message authentication code*) perustuvaa kertakäyttöistä salasanaa eli HOTP (*HMAC-based one-time password*). HOTP- ja TOTP-arvon ero se, että HOTP-arvojen muodostamisessa käytetään käyttökertojen mukaan nousevaa laskuria, kun TOTP-arvon kanssa laskurina toimii aika (M’Raihi ym. 2011, 3). TOTP-arvon perustana toimivalla HMAC-tunnuksella tarkoitetaan salaista avainta käyttävän hajautusalgoritmin avulla muodostettua merkkijonoa, joka luodaan antamalla hajautusalgoritmille lähtöarvo ja salainen, josta muodoste-

taan lopullinen HMAC-tunnus (Bellare ym. 1997, 3–5). Tällaista tunnusta voidaan käyttää esimerkiksi viestin oikeellisuuden varmistamiseen: antamalla hajautusalgoritmille tismalleen sama viesti, muodostuu tismalleen sama hajautusarvo (Ensuring Data Integrity with Hash Codes. 2021). HOTP- ja TOTP-arvojen muodostamisessa hyödynnetään SHA-1-hajautusalgoritmia, joka tuottaa 20:n tavun kokoisia merkkijonoja (Bellare, Hoornaert ym. 2005, 6). Seuraavaksi käydään läpi tarvittavat vaiheet oikeanlaisen TOTP-arvon luomiseen.

Ensimmäinen vaihe on laskea laskuri, joka on TOTP-arvoa laskettaessa aikaperustainen. Luomishetken laskuri määritetään laskentahetken, Unix-ajan alun ja halutun tunnuksen voimassaoloajan perusteella (M'Raihi, Machani, Pei & Rydell 2011. 4). Kaava 1 esittelee tarkan laskutavan.

$$C = \frac{(T - T_0)}{T_x} \tag{1}$$

jossa T laskuhetken aika Unix ajassa mitattuna
T_0 tämän hetken Unix-ajan alku, 1.1.1970 klo 00:00
T_x aikajänne, käytännössä tunnuksen voimassaoloaika

Toinen vaihe on luoda 20:n tavun kokoinen merkkijono *HS* hyödyntäen algoritmille annettua salaisuutta ja edellisessä vaiheessa laskettua aikaa. Bellare, Hoornaert ym. (2005, 6) määrittävät merkkijonon HS luomisen kaavan 2 mukaisella tavalla.

$$HS = H(K, C) \tag{2}$$

jossa H käytetty hajautusalgoritmi, tässä tapauksessa SHA-1
K jaettu salaisuus, jonka pohjalta hajautus tehdään
C aikaperustainen laskuri

Seuraava vaihe on muuntaa saatu 20:n tavun arvo 4:n tavun kokoiseksi dynaamiseksi binäärikoodiksi ja laskea tämän avulla lopullinen arvo. Muuntoalgoritmin toiminta käydään läpi tarkemmin, kun käsitellään toteutettua ohjelmakoodia TOTP-arvon laskemiseksi. Kaavalla 3 saadaan lopullinen TOTP-arvo.

$$TOTP = mod((DT(HS))\ 10^d) \quad (3)$$

jossa	DT	käytetty muuntoalgoritmi
	HS	edellisessä vaiheessa luotu merkkijono
	d	lopullisen tunnuksen haluttu numeroiden määrä

Lopullinen tunnus saadaan siis ottamalla jakojäännös dynaamiselta muunto-algoritmilta saadun binäärikoodin desimaalimuodosta ja luvun 10^d välisestä jakolaskusta (M'Raihi ym. 2011. 3). Seuraavaksi käydään läpi kaksivaiheisen tunnistautumisen toteutetut osat ja edellisten kaavojen pohjalta luotu TOTP-arvon laskemisen C#-kielinen toteutus, jota käytetään palvelinpuolen TOTP-arvon laskennassa.

Kun käyttäjä pyytää palvelimelta kaksivaiheisen tunnistautumisen käyttöönottoon, luodaan tälle henkilökohtainen salaisuus, jota käytetään TOTP-arvo laskemiseksi käyttäjälle.

```
1 reference | Antti Paavola, 21 hours ago | 1 author, 2 changes
public void SetSecrets()
{

    TOTPGenerator totpGenerator = new TOTPGenerator();
    AesHelper aesHelper = new AesHelper();
    String totpSecret = Base32Encoding.ToString(totpGenerator.GenerateRandomKey());

    using (Aes aes = Aes.Create())
    {
        byte [] encryptedKey = aesHelper.EncryptStringToBytes_Aes(totpSecret, aesHelper.TestKey, aes.IV);
        TotpSecret = Convert.ToBase64String(encryptedKey);
        Iv = Convert.ToBase64String(aes.IV);
    }
}
```

Kuva 16. User-luokan metodi, käyttäjän salaisuuksien luomiselle

Kuva 16 havainnollistaa, kuinka käyttäjälle luodaan salaisuudet käyttäen avuksi myöhemmin esiteltävää kehitystyössä toteutettua TOTPGenerator-luokkaa sekä teoriaosuudessa esitellyn symmetrisen salauksen muotoa: AES-salausta. Symmetrinen salausmuoto valittiin toteutukseen siitä syystä, että salausta ei ole tarvetta purkaa missään muualla kuin palvelimella. Tarvitaan siis vain yksi salainen avain, jolla tieto salataan ja salaus puretaan (Cryptography 2022).

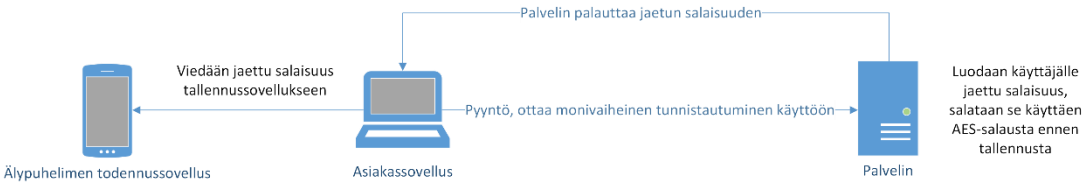
Ensin käyttäjälle luodaan satunnainen 20:n tavun kokoinen tunnus. Tämä tunnus on tallennettava palvelimelle, jotta palvelin ja käyttäjän laite voi laskea saman TOTP-arvon samalla ajan hetkellä. Tietoturvan parantamiseksi tunnus salataan käyttäen AES-salausta, ennen tietokantaan tallennusta muutetaan salatun tunnuksen sisältävä tavutaulukko merkkijonoksi.

```
1 reference | Antti Paavola, 1 day ago | 1 author, 1 change
public byte[] GenerateRandomKey(int length = (int)KeyLengthEnum.SHA1)
{
    byte[] key = new byte[length];
    RandomNumberGenerator rand = RandomNumberGenerator.Create();
    rand.GetBytes(key);
    return key;

}
```

Kuva 17. Satunnaisen avaimen luonti käyttäjälle

Kuvan 17 funktio luo käyttäjälle satunnaisen annetun pituuden kokoisen merkkijonon, joka tallennetaan tavutaulukkona.

On tärkeää, että edellisessä kappaleessa esitelty salaisuus on sama sekä käyttäjällä, että palvelimella. Erityisen tärkeää on, ettei salaisuuksia enää luonnin jälkeen lähetetä verkon yli, eikä tälle myöskään ole tarvetta sillä salaisuus tallennettaan luotaessa salattuna palvelimelle ja käyttäjä tallentaa sen oman laitteen avainohjelmaan, esimerkiksi Microsoft Authenticator -applikaatioon.



Kuva 18. Toteutetun TOTP-tunnistautumisen käyttöönoton havainnollistava kuva

Kuvassa 18 äsken kuvailtu tapahtumaketju on yksinkertaistettu yhteen kuvaan, josta selviää yleisellä tasolla toteutetun monivaiheisen tunnistautumisen käyttöönoton vaiheet.

Tunnistautumista tehdessä käyttäjää pyydetään antamaan sen ajan hetken TOTP-arvo. Kun arvo lähetetään, käynnistetään palvelimella toiminto, joka luo

yhteistä salaisuutta ja aikaa käyttäen vastaavan avaimen, vertaa käyttäjän tunnusta palvelimen tunnukseen ja joko antaa tai evää käyttäjältä pääsyn sovellukseen. Palvelimen TOTP-arvon luonti tapahtuu aiemmin esiteltyjen vaiheiden mukaisesti seuraavalla tavalla.

```
1 reference | Antti Paavola, 1 day ago | 1 author, 1 change
private UInt64 Counter(int subtraction = 0)
{
    var Difference = (DateTime.UtcNow - new DateTime(1970, 1, 1)).TotalSeconds
    return (UInt64)Math.Floor(Difference / Tx);
}
```

Kuva 19. Ensimmäinen vaihe, laskurin luonti

Kuvan 19 funktiossa luodaan laskuri käyttäen .NET-kirjastoista löytyvää DateTime-luokkaa. DateTime-luokasta haetaan tämän hetken aika, josta vähennetään Unix-ajanjakson alkuhetki. Tulos muutetaan sekunneiksi ja se jaetaan arvolla T_x, joka on luotavan TOTP-arvon voimassaoloaika.

```
1 reference | Antti Paavola, 1 day ago | 1 author, 1 change
private String GenerateTOTP(UInt64 Counter, byte[] key, int length = 6)
{
    var hmac = new HMACSHA1(key, true);
    var data = BitConverter.GetBytes(Counter);

    if (BitConverter.IsLittleEndian)
        Array.Reverse(data);

    hmac.ComputeHash(data);
```

Kuva 20. Toinen vaihe, HMAC-arvon luonti

Kuvassa 20 luodaan HMAC-merkkijono antamalla .NET-kirjastosta löytyvälle HMACSHA1-luokalle alkuarvoksi aiemmin luotu yhteinen salaisuus. Lopullinen HMAC-arvo saadaan, kun kutsutaan HMACSHA1-luokan ComputeHash-metodia aiemmin luotu laskurin arvo parametrinaan.

```
1 reference | Antti Paavola, 1 day ago | 1 author, 1 change
private int Truncate(byte[] hmacResult, int length)
{
    int offset = hmacResult[hmacResult.Length - 1] & 0xf;
    int binCode = (hmacResult[offset] & 0x7f) << 24 |
        (hmacResult[offset + 1] & 0xff) << 16 |
        (hmacResult[offset + 2] & 0xff) << 8 |
        (hmacResult[offset + 3] & 0xff);

    return binCode % (int)Math.Pow(10, length);
}
```

Kuva 21. Lyhennysalgoritmin C# toteutus

Kuvassa 21 esitetään C#-kielinen toteutus funktiosta, jonka Bellare, Hoornaert ym. (2005, 7) esittelevät HOTP-arvon luomisen määritelmässään. Tällä funktiolla luodaan lopullinen tunnus muuttamalla HMAC-merkkijono desimaalilukumuotoon ja ottamalla jakojäännös muunnoksen tuloksesta ja luvusta 10^{length}. Kuvan 18 funktion ensimmäisellä rivillä haetaan alkupiste muodostettavalle luvulle. Rivillä näkyvä heksadesimaaliarvo 0xf on binäärimuodossa 1111 ja & tarkoittaa loogista operaatiota nimeltä AND. Alkupiste muodostuu siis seuraavalla tavalla: Oletetaan, että viimeinen tavu on 0110 0100. Tämän tavun ja 1111 tavun välinen AND operaatio muodostaa tavun 0000 0100, sillä muodostettavaan tavuun valitaan ne bitit, joissa on molemmissa arvo eli 1. Operaatiosta saatua tavua vastaavaa desimaalilukuarvoa käytetään lopullisen koottavan luvun alkupisteenä, eli tavuja sisältävän taulukon indeksinä. Neljännestä indeksistä lähtien otetaan seuraavat neljä tavua ja siirretään niitä peräkkäin <<-bittisiirto-operaattorin avulla. Näin saadaan lopullinen tavujono, josta saadaan lopullinen desimaalilukuarvo, josta saadaan jakojäännöksen avulla lopullinen, tässä tapauksessa kuusinumeroinen, desimaaliluku.

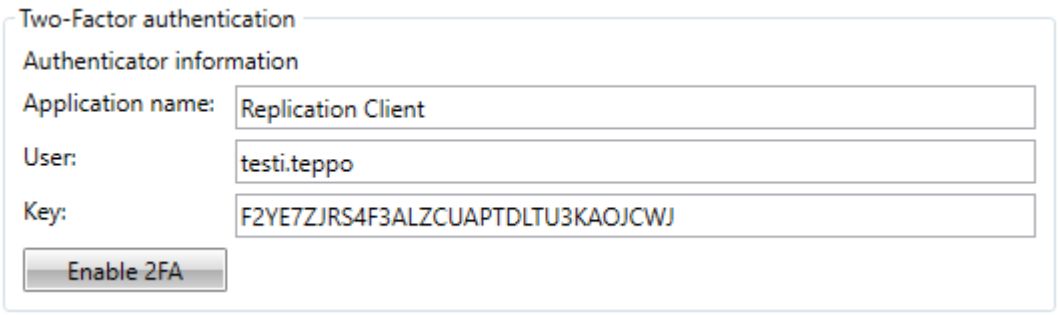
```
1 reference | Antti Paavola, 2 days ago | 1 author, 1 change
public bool ValidateTOTP(string codeToValidate, byte[] key)
{
    if (String.IsNullOrWhiteSpace(codeToValidate))
        return false;

    return codeToValidate.Equals(GenerateTOTP(Counter(), key), StringComparison.OrdinalIgnoreCase);
}
```

Kuva 22. TOTP-arvon validointi

Kuvassa 22 esitellään metodi, joka validoi käyttäjän palvelimelle lähettämän TOTP-arvon. Metodi saa parametrina, TOTP-arvon ja jaetun salaisuuden tavuina. Näiden avulla muodostetaan sen hetken aikaikkunan TOTP-arvo edellä

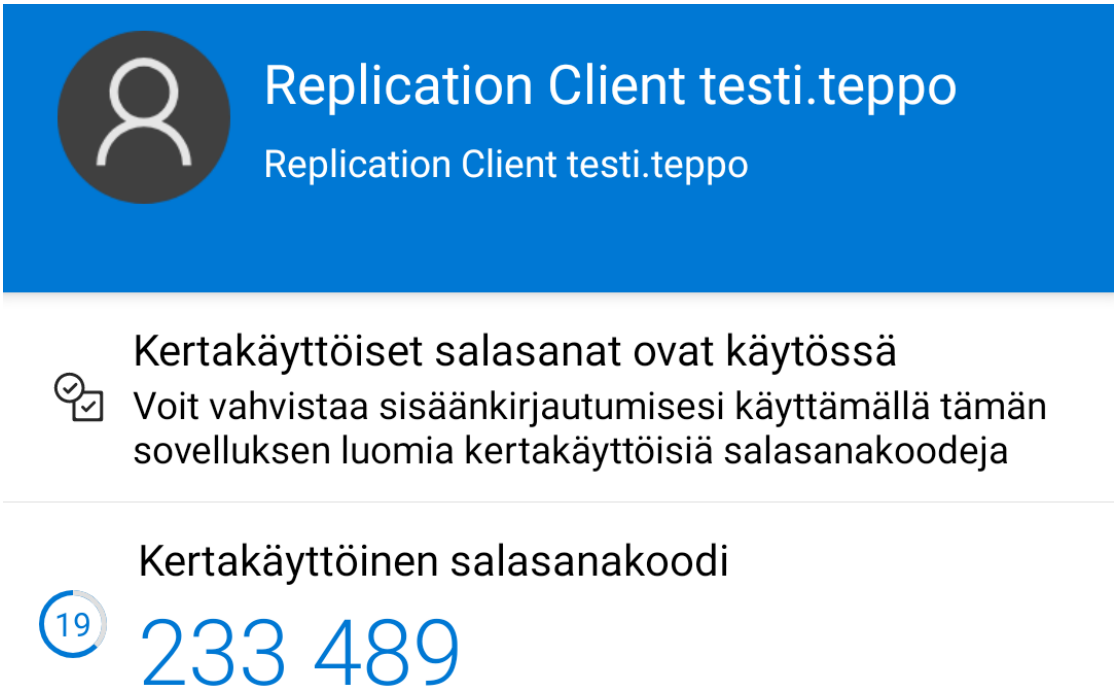
esitettyjen vaiheiden mukaisesti. Jos lähetetty TOTP-arvo on tyhjä tai ei vastaa palvelimella muodostettua arvoa, palautetaan epätosi, muuten palautetaan tosi.

Käyttäjälle edellä kuvailtu prosessi näyttäytyy hyvin yksinkertaisena. Käyttäjä pyytää käyttöliittymästä kaksivaiheisen tunnistautumisen aktivoimista tililleen, jolloin palvelin luo käyttäjälle aiemmin esitellyn jaetun salaisuuden ja lähettää sen asiakassovellukselle. Tämä hetki on tunnistautumisen haavoittuvaisin osa, sillä aktivointi on ainut hetki, jolloin jaettua salaisuutta käsitellään avoimesti ja se on salaamattomana näkyvissä. Palvelimella salaisuus salataan heti sen luonnin jälkeen ennen sen tallennusta. Usein salaisuus luetaan QR-koodina johonkin tunnistautumissovellukseen. Opinnäytetyön kirjoitushetkellä asiakassovelluksen QR-koodin luominen ei ollut vielä valmis, joten jaettu salaisuus annetaan merkkijonona.



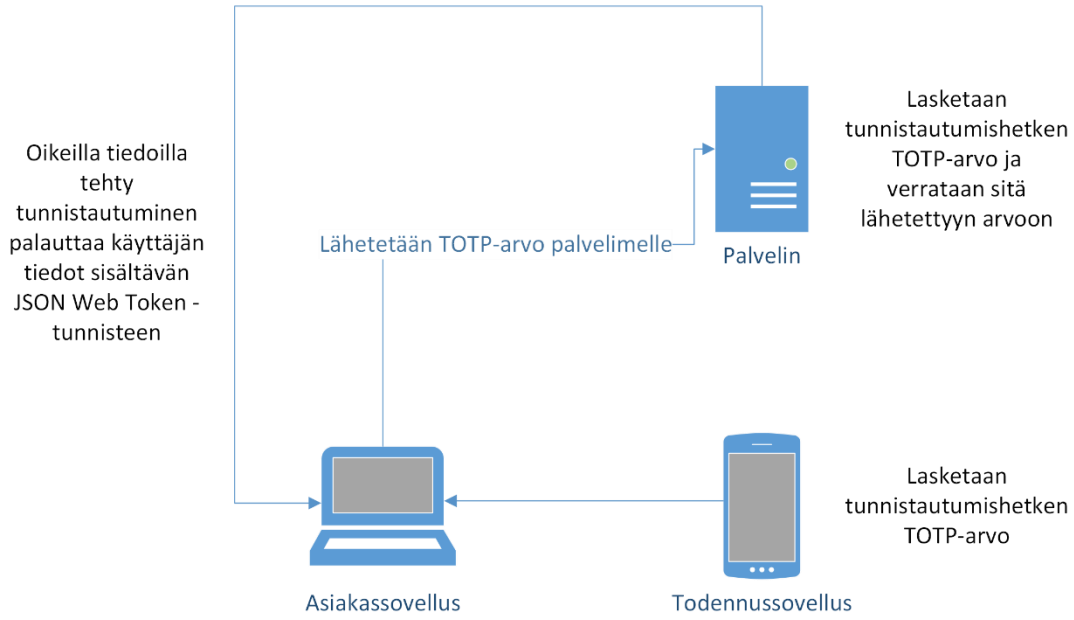
Kuva 23. Kaksivaiheisen tunnistautumisen aktivointi

Kuvassa 23 nähdään sovellukseen testaamiseen luodun tilin tekemä pyyntö kaksivaiheisesta tunnistautumisesta. Jaettu salaisuus voidaan ottaa ylös mihin tahansa tunnistautumissovellukseen, joka laskee ajan ja jaetun salaisuuden perusteella TOTP-arvoja. Tällaisia sovelluksia ovat muun muassa aiemminkin tekstissä mainittu Microsoft Authenticator ja Google Authenticator.



Kuva 24. Tunnistautumissovelluksen luoma koodi

Kuvassa 24 nähdään aiemmin annetuilla tiedoilla luotu Microsoft Authenticator -tili, joka luo 30 sekunnin välein uuden kertakäyttöisen salasanan käytettäväksi. Tässä huomataan käytännössä aiemmin tekstissä mainittu TOTP-tunnistautumisen etu: salaisuuden luomisen jälkeen käyttäjän tai palvelimen ei tarvitse kommunikoida keskenään mitään salaista tietoa, sillä sama TOTP-arvo lasketaan molemmissa paikoissa. Ainoastaan kertakäyttöisen salasanan lähettäminen riittää. Tässä luvussa aiemmin esitellyt kaavat noudattavat TOTP-tunnuksen luomiselle asetettuja standardeja, joten voidaan luottaa, että oikealla ajan hetkellä ja oikealla salaisuudella palvelimen laskema arvo täsmää samoilla arvoilla laskettuun käyttäjän sovelluksen arvoon.



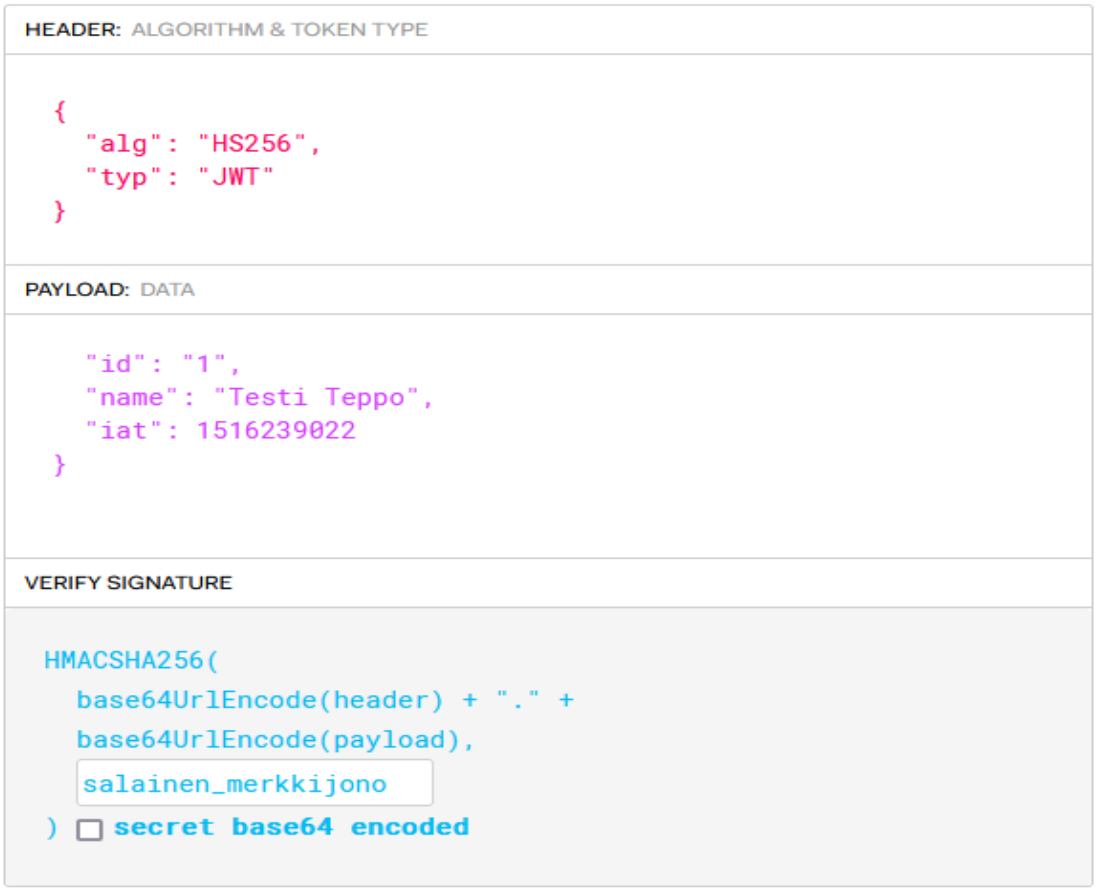
Kuva 25. Toteutetun kaksivaiheisen tunnistautumisen vaiheet

Kuvassa 25 esitetään jälleen yleisellä tasolla yllä kuvailtu tapahtumaketju. Käyttäjä katsoo aiemmin luodun salaisuuden mukaisen TOTP-arvon todennussovelluksesta, lähettää tiedot palvelimelle, joka laskee samanaikaisesti TOTP-arvon, vertaa sitä saamaansa tunnukseen ja palauttaa vastauksen takaisin asiakassovellukselle.

4.5 Oikeuksien hallinta

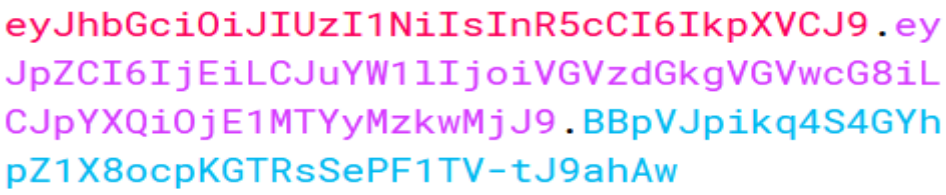
JSON Web Token

JSON Web Token on merkkijono, johon koodataan sisään Claim-nimellä kutsuttuja avain-arvo-pareja hyödyntäen jotakin salausalgoritmia. Avain esitetään aina merkkijonona, kun taas arvo voi sisältää minkä tahansa JSON-muotoisen arvon. Yleisin ja tämänkin kehitystyön JSON Web Tokenin käyttötarkoitus on auktorisointi lähettämällä kerran luotu ja allekirjoitettu tunniste kaikkien HTTP-pyyntöjen mukana. (Introduction to JSON Web Tokens s.a.)



Kuva 26. JSON Web Token osat (Introduction to JSON Web Tokens s.a.)

Tunniste koostuu kolmesta osasta: otsikkorivistä, sisällöstä ja allekirjoituksesta. Kuvassa 26 on eritelty tunnisteen eri osat, otsikkorivi sisältää tiedon tunnisteen tyypistä ja käytetystä hajautusalgoritmista. Sisällössä on itse käytettävä tieto, kuvan 26 esimerkissä tieto sisältää käyttäjän id-tunnisteen, nimen ja tiedon milloin tunniste on luotu. Viimeisenä on allekirjoitus, joka luodaan koodaamalla otsikko- ja tietokentät salausalgoritmilla. Näiden lisäksi allekirjoituksessa käytetään salaista merkkijonoa. (Introduction to JSON Web Tokens s.a.)



Kuva 27. Tunniste koodauksen jälkeen (Introduction to JSON Web Tokens. s.a.)

Lopputuloksena on kuvan 27 mukainen merkkijono. Purkamalla merkkijono saadaan kuvan 27 mukaiset selkokieliset tiedot.

Oikeuksien hallinta ohjelmassa toteutetaan aiemmassa aliluvussa esitellyn tietokannan ja sisäänkirjautumisen yhteydessä käyttäjälle luodun tunnisteen avulla. Edellisessä aliluvussa esiteltiin, kuinka kirjautuessa luodaan JSON Web Token, joka asetetaan ReplicationHttpClient-esiintymän otsikkoriville. Kuten yllä mainittiin, pysyy tämä otsikkorivi muuttumattomana esiintymän koko elinajan. Näin voidaan tehdä tunnistautumista vaativia kutsuja rajapinnalle ilman, että rajapinnalta tarvitsee pyytää uutta tunnistetta jokaisen kutsun yhteydessä. Tarkastellaan oikeuksien hallinnan koko prosessia käymällä läpi yhden rajapinnalle tehtävän kutsun vaiheet.

```
string query = String.Format("API/UpdateProjectData/{0}?ReleaseFileLocks={1}", project.ProjectGUID, releaseFileLocks == null ? false : releaseFileLocks.Value);

var result = await ReplicationHttpClient.Instance.PostAsync($"{project.ServerUrl}{query}", contentRequest, cancellationToken);
result.ThrowIfNotSuccess();
```

Kuva 28. Asiakassovelluksen kutsu rajapintaan.

Kuvassa 28 asiakassovellus tekee kutsun rajapinnan päätepisteeseen. Tässä vaiheessa on jo tehty tunnistautuminen ja ReplicationHttpClient-esiintymän otsikkorivillä on oikeassa muodossa oleva tunniste.

```
var userName = this.User?.Identity?.Name;
var userID = this.User?.Claims.Where(c => c.Type == ClaimTypes.Sid).FirstOrDefault().Value;

if (String.IsNullOrWhiteSpace(userID)) throw new AuthenticationException("UpdateProjectData fail, User ID is empty");
if (String.IsNullOrWhiteSpace(userName)) throw new AuthenticationException("UpdateProjectData fail, User name is empty");
if (String.IsNullOrWhiteSpace(userComputerName)) throw new ArgumentNullException("UpdateProjectData fail, User computer name is empty");

// Check user's and user's group authorization for project here
var projectRoles = await _dbUserService.GetUserRolesInProjectGroupsAsync(userID, projectGUID);
// If user has no overriding user role, get group roles
if (projectRoles.Count() <= 0)
    projectRoles = await _dbUserService.GetUserRoleInUserProject(userID, projectGUID);

if (!RoleHelper.CheckAuthentication(RoleEnum.User, projectRoles))
    throw new AuthenticationException("You don't have the permission to check-in project-files. Need at least Edit-rights");
```

Kuva 29. Rajapinnan käsittelijäfunktio

Kuva 29 sisältää varsinaisen oikeuksien tarkistuksen. Rajapinnan käsittelijäfunktio parsii kutsun mukana tulleesta JSON Web Token-tunnisteen Claim-avain-arvo-pareista käyttäjän Sid-arvon, eli käyttäjän tietokannassa olevan yksilöivän tunnisteen. Jos tunniste, käyttäjän nimi tai tietokoneen nimi puuttuu, heitetään poikkeus, joka palauttaa käsittelijäfunktion kutsujalle tiedon, että jokin näistä kolmesta puuttui pyynnöstä. Jos tarvittavat tiedot löytyvät kutsusta, tarkistetaan tietokannasta käyttäjän Sid-tunnisteen avulla, minkä tasoiset oikeudet käyttäjällä tai ryhmällä, johon käyttäjä kuuluu, on projektissa ja verra-

taan sitä toiminnolle määritettyyn minimioikeuteen. Oikeuksien ollessa kunnossa, käyttäjälle palautetaan haluttu resurssi, muuten palautetaan poikkeus ja viesti, ettei käyttöoikeuksia kyseiseen toimintoon ole.

5 JOHTOPÄÄTÖKSET

Kehitystyön tavoitteena oli kehittää tunnistautumisen ja tunnistautuneiden käyttäjien oikeuksien tarkistus ja hallinta osaksi olemassa olevaa web-sovellusta. Vaatimukset käyttäjän tunnistautumiselle kehityksen alussa olivat mahdollisuus kirjautua sovellukseen sekä käyttäjänimi-salasana-parilla, että Windows-työasemalle kirjautuneen käyttäjän Windows-tunnuksilla. Lisäksi näitä vahventamaan tulisi kehittää monivaiheisen tunnistautumisen mahdollisuus. Käyttäjän oikeuksien hallinnan taas tuli mahdollistaa erityyppisten käyttöoikeuksien liittäminen projekteille eri tasoisesti, käyttäjäkohtaisesti sekä käyttäjäryhmäkohtaisesti. Kehitystyölle asetetuista tavoitteista muodostui tutkimusongelma, josta johdettiin teoriaosuudessa esitellyt tutkimuskysymykset. Opinnäytetyö pyrki vastaamaan näihin kysymyksiin kehitystutkimuksen keinoin kerättyä tutkimusaineistoa hyödyntäen.

Kehitystyön tuloksena syntyi REST-arkkitehtuurimallia noudattelevan palvelinpuolen ohjelmointirajapinnan sekä tätä rajapintaa käyttävän asiakassovelluksen osia. Nämä tulokset-luvussa läpikäydyt osat vastaavat työlle tutkimuksen alussa määriteltyihin tutkimuskysymyksiin ja poistavat näin ongelman, josta ratkaistut kysymykset olivat johdettu. Seuraavassa luvussa arvioidaan tarkemmin kehitystyön onnistumista, tuloksien luotettavuutta, työn aikana ilmenneitä haasteita ja eritellään mahdollisia jatkokehityskohteita.

6 POHDINTA

Työtä voi pitää onnistuneena sillä sen tuloksena syntyi käyttäjän tunnistautumisen mahdollistavat osat ohjelmointirajapintaan eli palvelimen puolelle sekä tätä rajapintaa käyttävään asiakassovellukseen. Lisäksi tunnistautumisen pohjaksi suunniteltiin ja luotiin kehitysympäristössä toimiva tietokanta, jonne kyetään tallettamaan tarvittava tieto, jotta tunnistautumista voidaan tehdä. Suurimmat onnistumiset kehitystyössä ovat kaksivaiheisen tunnistautumisen kehitys ja Windows-tunnuksilla kirjautumisen mahdollisuus. Tutkimuksen tarve-

analyysia tehdessä huomattiin, että varsinkin Windows-tunnistautumisen mahdollisuus on käyttäjien näkökulmasta erittäin tärkeä, sillä tuotantoon mennessä sovelluksen ensimmäiset käyttäjät tullevat toimimaan sisäverkossa ja kirjautuminen tehdään tällöin toimialueverkon Windows-tunnuksilla. Kaksivaiheinen tunnistautuminen taas oli tietoturvan lisäämisen kannalta tärkein yksittäinen kehityskohde. Suunnitteluvaiheessa päädyttiin siihen, että helppokäyttöinen kaksivaiheinen tunnistautuminen on yksi edellytys sille, että rajapintaa voidaan joskus käyttää myös muuten kuin sisäverkossa. Molempia toteutuksia voidaan pitää onnistuneina myös käytettävyyden näkökulmasta, sillä Windows-kirjautuminen tapahtuu syöttämättä mitään tunnuksia asiakassovelluksen käyttäessä kirjautuneen käyttäjän tunnistustietoja. Kaksivaiheinen tunnistautuminen taas toteutettiin tunnistautumissovelluksia hyödyntäen ja käyttöönoton jälkeen käyttäjän on vaivatonta katsoa kertakäyttöinen avain puhelimestaan. Näiden lisäksi toteutettiin käyttöoikeuksien tarkistuksia palvelinpuolen ohjelmointirajapintaan. Tämän hetken toteutus vastaa kehitystyön alussa sille asetettuun vaatimukseen. Rajapintaan tehtyjen kutsujen käsittelijäfunktiot tarkistavat kutsun tekijän oikeudet ja riittämättömillä oikeuksilla tehtyyn kutsuun ei vastata pyydetyillä resursseilla. Oikeuksien tarkistamisen toteutuksessa on kuitenkin vielä paljon parannettavaa. Rajapintaan olisi järkevä toteuttaa jokin keskitetty ratkaisu oikeuksien tarkistamiselle sen sijaan, että jokaisen kutsun käsittelijäfunktiossa on erikseen toteutettu käyttöoikeuksien hakeminen sille lähetetystä tunnisteesta ja niiden tarkistaminen. Tämä parantaisi ohjelmakoodin luettavuutta ja helpottaisi mahdollisten muutosten tekemistä.

Kehitystyö perustui tutkimuksen alussa ja sen aikana hankittuun tutkimusmateriaaliin. Aineistona käytettiin laajalti erilaisia yleisesti hyväksyttyjä määritelmiä muun muassa REST-arkkitehtuurimallin, HTTP-protokollan ja käytettyjen salausmenetelmien ja hajautusalgoritmien osalta. Lisäksi tutkimuksen tukena käytettiin eri sovelluskomponenttien osien kehittäjien tuottamia dokumentaatioita. Edellä mainittujen asioiden valossa ei ollut yllättävää huomata, että kehitetyt osat vastasivat hyvin teoriassa esiteltyjä malleja

Kehitystyön aikana sovellukseen toteutetuille osille tehtiin jatkuvaa testaamista itse kehittämisen tukena. Yhden funktion tai funktion osan valmistuessa sitä testattiin oikealla ja väärällä syötteellä, jotta saatiin tieto siitä antaako to-

teutettu funktio tai sen osa toivotunlaisia tuloksia. Tämän testauksen perusteella arvioitiin, onko lähestymistapa oikea ja sitä muutettiin tarpeen vaatiessa. Tällainen testaus ei vielä riitä siihen, että kehitystyön tulosta voidaan pitää luotettavana ja valmiina käyttöönotettavaksi todellisessa ympäristössä. Tätä voidaankin pitää suurimpana jatkokehityskohteena ja puutteena tässä kehitystyössä. Testausta olisi ollut hyvä toteuttaa järjestelmällisemmin jo kehitystyön aikana esimerkiksi funktioiden yksikkötestien muodossa. Tällaisella järjestelmällisellä testaamisella olisi voitu varmistua siitä, että muutoksia tehdessä funktiot täyttävät edelleen niille asetetut vaatimukset. Ilman järjestelmällistä testaamista kaikki virheet eivät tule esiin, kun testataan manuaalisesti vain uusinta toteutettua sovelluksen osaa. Ennen käyttöönottoa palvelimen rajapinnalle tulisi tehdä kattavaa testausta, jotta voidaan varmistaa, että sovellus ei sisällä tunnettuja tietoturvariskejä. Tätä testausta tulisi tehdä koko sovelluksen elinkaaren ajan ja päivittää sen komponentteja, jos niissä ilmenee ongelmia.

Käyttäjien oikeuksien tarkistamisen, testauksen ja tietoturvan varmistamisen lisäksi jatkokehityskohteita löytyy myös asiakassovelluksen puolelta. Käyttökokemuksen parantamiseksi olisi syytä tutkia, mitä tietoa voitaisiin tallentaa välimuistiin, jotta rajapintaan ei tarvitsisi tehdä niin paljon kyselyitä. Kyselyiden määrä ei muodostu ongelmaksi yhtäaikaisten käyttäjämäärien ollessa pieniä, mutta niiden kasvaessa voi ohjelman käyttö hidastua runsaasti. Myös kaksivaiheisen tunnistautumisen käyttöönottoa voisi parantaa. Käyttöönoton yhteydessä käyttäjälle näytettävä salaisuus tulisi toimittaa QR-koodina, jotta käyttäjän ei tarvitse syöttää puhelimeen 20-merkkiä pitkää merkkijonoa. Pitkien merkkijonojen syöttämisessä käy helposti näppäilyvirheitä, jolloin käyttöönotto menee pieleen. Sovelluksen todellista käyttöä ajatellen olisi erittäin hyödyllistä kehittää keskitetty hallintapaneeli, josta suunnittelijat ylläpitäjä kykenisi lisäämään ja poistamaan käyttäjiä ja muokkaamaan näiden oikeuksia. Tällä hetkellä nämä toiminnot tehdään suoraan tietokannassa. Tämä tapa ei ole optimaalinen. Sovelluksen loppukäyttäjät eivät lähtökohtaisesti ole tietojenkäsittelyn ammattilaisia eikä voida olettaa, että heiltä löytyy tarvittava osaaminen tietokantakyselyiden tuottamiseen. Hallintapaneelin toteutusta pohdittiin jo tämän opinnäytetyön tarveanalyysin aikana, mutta se rajattiin ulos sillä sen toteuttaminen olisi laajentanut työtä ja vienyt huomiota ydinkomponenttien kehitykseltä. Tämän opinnäytetyön teoriaosuutta voidaan kuitenkin hyödyntää myös

mahdollisessa hallintapaneelin kehitystyössä, sillä se voidaan toteuttaa hyödyntäen samaa rajapintaa, jonka avulla tunnistautuminen ja tämän hetken oikeuksien tarkistaminen on toteutettu.

LÄHTEET

An overview of HTTP. 2021. MDN Web Docs. WWW-dokumentti. Päivitetty 3.10.2021. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview [Viitattu 10.10.2022]

A tour of the C# language. 2022. Microsoft Documentation. WWW-dokumentti. Päivitetty 29.9.2022. Saatavissa: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ [Viitattu 10.10.2022]

Babic, T. 2020. What Is a Many-to-Many Relationship in a Database? An Explanation with Three Examples. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://vertabelo.com/blog/many-to-many-relationship/ [Viitattu 14.11.2022]

Bellare, D., Canetti, R., Krawczyk, H. 1997. HMAC: Keyed-Hashing for Message Authentication. PDF-dokumentti. Saatavissa: https://www.rfc-editor.org/rfc/pdfrfc/rfc2104.txt.pdf [Viitattu 15.11.2022]

Bellare, D. Hoornaert, F. M'Raihi, D., Naccache, D. & Ranen, O. 2005. HOTP: An HMAC-Based One-Time Password Algorithm. PDF-Dokumentti. Saatavissa: https://www.rfc-editor.org/rfc/pdfrfc/rfc4226.txt.pdf [Viitattu 11.11.2022]

Cryptography. 2022. Cyber Security Base 2022. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://cybersecuritybase.mooc.fi/module-4.3/1-symmetric [Viitattu 11.11.2022]

Data-Access Layer. 2022. Geeks for geeks. WWW-dokumentti. Päivitetty 17.8.2022. Saatavissa: https://www.geeksforgeeks.org/data-access-layer/ [Viitattu 15.10.2022]

Elliot, M. 2020. Do you use SMS for two-factor authentication? Here's why you shouldn't. WWW-dokumentti. Päivitetty 8.4.2020 Saatavissa: https://www.cnet.com/news/privacy/do-you-use-sms-for-two-factor-authentication-heres-why-you-shouldnt/ [Viitattu 18.11.2022]

Ensuring Data Integrity with Hash Codes. 2021. Microsoft Documentation. WWW-dokumentti. Ei päivitystietoja.Saatavissa: https://learn.microsoft.com/en-us/dotnet/standard/security/ensuring-data-integrity-with-hash-codes [Viitattu 15.11.2022]

Fernigrini, L. 2021. What Is a One-to-Many Relationship in a Database? An Explanation with Examples. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://vertabelo.com/blog/one-to-many-relationship/ [Viitattu 14.11.2022]

Fielding, R. T. 2000. Architectural Styles and the Design of Network-based Software Architectures. PDF-dokumentti. Saatavissa: https://www.ics.uci.edu/%7Efielding/pubs/dissertation/fielding_dissertation.pdf [Viitattu 11.10.2022]

Fielding, R. T.& Reschke, J. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. PDF-Dokumentti. Päivitetty 7.2014 Saatavissa: https://www.rfc-editor.org/info/rfc7231 [Viitattu 11.10.2022]

HttpClient Class. Microsoft Documentation. WWW-Dokumentti. Ei päivitystietoja. Saatavissa: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-6.0#definition [Viitattu 11.11.2022]

Ganesan, G., Sobti, R. 2012. Cryptographic Hash Functions: A Review. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://www.researchgate.net/profile/Geetha-Ganesan/publication/267422045_Cryptographic_Hash_Functions_A_Review/links/549cf6d10cf2b8037138c35c/Cryptographic-Hash-Functions-A-Review.pdf [Viitattu 18.11.2022]

Implementing the Singleton pattern in C#, C# in Depth Articles. WWW-dokumentti. Ei päivitystietoja. Saatavilla https://csharpindepth.com/articles/Singleton [Viitattu 11.11.2022]

Introduction to JSON Web Tokens. JWT documentation. WWW-dokumentti. Ei päivitystietoja. Saatavilla: https://jwt.io/introduction [Viitattu 14.11.2022]

Kananen, J. 2017. Kehittämistutkimus interventiotutkimuksen muotona. Tampere: Suomen Yliopistopaino Oy.

Kleppman, M. 2017. Designing Data-Intensive Applications. Sebastopol: O'Reilly Media

Massé, M. 2012. REST API Design Rulebook. Sebastopol: O'Reilly Media

Monivaiheinen tunnistautuminen suojaa käyttäjätilejäsi. Kyberturvallisuuskeskus. WWW-dokumentti. Päivitetty 15.3.2022. Saatavissa: https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/ohjeet-ja-oppaat/monivaiheinen-tunnistautuminen-suojaa-kayttajatilejasi [Viitattu 11.11.2022]

M'Raihi, D., Machani, S., Pei, M. & Rydell, J. 2011. TOTP: Time-Based One-Time Password Algorithm, PDF-dokumentti. Saatavissa: https://www.rfc-editor.org/rfc/pdfrfc/rfc6238.txt.pdf [Viitattu 11.11.2022]

Overview to ASP.NET Core 2022. Microsoft Documentation, WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0 [Viitattu 11.11.2022]

Pernaa, J. Kehittämistutkimus tutkimusmenetelmänä. PDF-dokumentti. Saatavilla: https://tuhat.helsinki.fi/ws/files/127650174/2013_Pernaa_KT_tutkimusmenetelmana_KT_kirj [Viitattu 28.11.2022]

Peterson, R. 2022. Relational Data Model in DBMS | Database Concepts and Example. WWW-dokumentti. Päivitetty 29.10.2022. Saatavissa https://www.guru99.com/relational-data-model-dbms.html [Viitattu 14.11.2022]

Proffit, Brian 2013. What APIs Are and Why They're Important. WWW-dokumentti. Ei päivitystietoja. Saatavissa: http://readwrite.com/2013/09/19/api-defined [Viitattu 20.10.2022]

Public key cryptography. 2021. IBM Documentation. WWW-dokumentti. Päivitetty 1.3.2022. Saatavissa: https://www.ibm.com/docs/en/ztpf/2020?topic=concepts-public-key-cryptography [Viitattu 10.11.2022]

RESTful Web Services - Statelessness. 2016. TutorialsPoint. WWW-dokumentti. Ei päivitystietoja. Saatavissa: http://www.tutorialspoint.com/restful/restful_statelessness.htm [Viitattu 10.10.2022]

RESTful Web Services - Messages. 2016. TutorialsPoint. Ei päivitystietoja. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://www.tutorialspoint.com/restful/restful_messages.htm [Viitattu 10.10.2022]

Robinson, K. 2021. 5 reasons SMS 2FA isn't going away. Ei päivitystietoja. WWW-dokumentti. Saatavissa: https://www.twilio.com/blog/sms-2fa-security [Viitattu 18.11.2022]

Singleton. Refactoring Guru. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://refactoring.guru/design-patterns/singleton [Viitattu 11.11.2022]

TOTP. Twilio Docs. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://www.twilio.com/docs/glossary/totp [Viitattu 18.11.2022]

Using the [Authorize] Attribute. 2022.Microsoft Documentation. WWW-dokumentti. Päivitetty 11.5.2022. Saatavissa: https://learn.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api#using-the-authorize-attribute [Viitattu 18.11.2022]

What is a database? Oracle. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://www.oracle.com/database/what-is-database/ [Viitattu 11.10.2022]

What is NoSQL? MongoDB. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://www.mongodb.com/nosql-explained [Viitattu 11.11.2022]

What is a Socket? The Java Tutorials. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html [Viitattu 11.11.2022]

Tietokantojen perusteet 2022. Helsingin Yliopisto. WWW-dokumentti. Ei päivitystietoja. Saatavissa: https://tikape.mooc.fi/syksy-2022/content/osa-1/index.html#mik%C3%A4-on-tietokanta [Viitattu 11.11.2022]