Tampereen ammattikorkeakoulu

Microsoft Entra ID -pohjaisen kertakirjautumisen toteuttaminen ParkkiV-järjestelmään

Kettunen Jaakko

OPINNÄYTETYÖ
Maaliskuu 2026

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

Kettunen, Jaakko:
Microsoft Entra ID -pohjaisen kertakirjautumisen toteuttaminen ParkkiV-järjestelmään

Opinnäytetyö 60 sivua, joista liitteitä 0 sivua
Maaliskuu 2026

Työn tavoitteena oli uudistaa ParkkiV-pysäköinninvalvontajärjestelmän tunnistautuminen käyttämään Microsoft Entra ID -ratkaisua. Työ toteutettiin toimeksiantona Finn-ID Oy:lle osana ParkkiV-järjestelmän kehitystä. Tavoitteiksi asetettiin keskitetty käyttäjähallinta, tuki asiakasorganisaatioiden yhtenäisille kirjautumiskäytännöille sekä moniasiakasympäristön edellyttämä käyttöoikeuksien hallinta.

Työssä suunniteltiin ja toteutettiin Entra ID -pohjainen kirjautumisratkaisu, jossa käyttäjän tunnistautuminen ulkoistettiin identiteetintarjoajalle ja kirjautuminen keskitettiin erillisen valintasivun kautta ohjautuvaan prosessiin. Ratkaisussa hyödynnettiin token-pohjaista käyttöoikeusmallia, jonka avulla tunnistettiin käyttäjän oikeudet eri ParkkiV-ympäristöihin. Roolipohjaisen mallin myötä käyttäjä ohjattiin joko suoraan omaan ParkkiV -ympäristöönsä tai käyttäjälle näytettiin lista sallituista kaupungeista.

Työn aikana rakennettiin erillinen valintasivu ja sitä palveleva taustapalvelu. ParkkiV:n taustajärjestelmään rakennettiin kirjautumisrajapinta, joka vastaanotti ja validoi valintasivulta välitetyn tokenin, ennen sovellusistunnon muodostamista.

Työn tuloksena ParkkiV-järjestelmään toteutettiin toimiva ja tietoturvallinen Entra ID -kirjautumisratkaisu. Ratkaisu paransi moniasiakasympäristön käytettävyyttä ja mahdollisti keskitetyn käyttöoikeuksien hallinnan asiakasorganisaation omassa Entra-ympäristössä. Ratkaisu on laajennettavissa kaikille ParkkiV -asiakkaille ja se voidaan jatkokehittää ParkkiV -mobiilisovellukseen.

Asiasanat: microsoft entra id, tunnistautuminen, openid connect, oauth 2.0, moniasiakasympäristö

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software engineering

Kettunen, Jaakko:
Implementation of Microsoft Entra ID-based Single Sign-On for the ParkkiV System

Bachelor's thesis 60 pages, appendices 0 pages
March 2026

The objective of this thesis was to modernize the authentication of the ParkkiV parking enforcement system by implementing a Microsoft Entra ID-based solution. The work was carried out as a commission for Finn-ID Oy as part of the ongoing development of the ParkkiV system. The main goals were to enable centralized user management, support the unified sign-in policies of client organizations, and provide access control suitable for a multi-tenant environment.

During the thesis project, an Entra ID-based sign-in solution was designed and implemented. In this solution, user authentication was delegated to an identity provider, and the sign-in process was centralized through a separate selection page. The solution utilized a token-based access control model to identify the user’s permissions for different ParkkiV environments. Based on this role-based model, the user was either directed straight to the correct ParkkiV environment or shown a list of permitted cities.

As part of the implementation, a separate selection page and a supporting backend service were developed. In addition, a login interface was added to the ParkkiV backend system. This interface received and validated the token forwarded from the selection page before creating the application session.

As a result of the thesis, a functional and secure Entra ID sign-in solution was implemented for the ParkkiV system. The solution improved usability in a multi-tenant environment and enabled centralized access management within the client organization’s own Entra environment. The solution can be extended to all ParkkiV customers and further developed for use in the ParkkiV mobile application.

Key words: microsoft entra id, authentication, openid connect, oauth 2.0, multi-tenant environment

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

☐ Ei

☒ Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni opinnäytetyöprosessin aikana seuraavia tekoälysovelluksia: Open AI ChatGPT

Tekoälysovellusten nimet ja versiot: Open AI ChatGPT 5.2: GPT -5.2

Käyttötarkoitus: Tekoälyä käytettiin ohjelmoinnin tukena valintasivun frontend- ja backend-osioissa sekä ParkkiV-taustajärjestelmän Python-koodissa. Lisäksi tekoälyä hyödynnettiin sisällysluettelon laatimisessa, lähteiden etsimisessä, lauserakenteiden jäsentelyssä ja oikeinkirjoituksen tarkastamisessa.

Osiot, joissa tekoälyä on käytetty: Tekoälyä käytettiin opinnäytetyön suunnitteluvaiheessa ja aiheen rajauksessa, ohjelmointivaiheessa valintasivun frontend- ja backend-toteutuksessa sekä ParkkiV-taustajärjestelmän Python-koodin tukena. Lisäksi tekoälyä hyödynnettiin teoriaosuuden lähteiden etsimisessä, lähdeviitteiden tarkistamisessa, sisällysluettelon laatimisessa sekä tekstin kielenhuollossa.

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

SISÄLLYS

1 JOHDANTO 7
2 PROJEKTIN ESITTELY 8
2.1 Tarkoitus 9
2.2 Tavoitteet 10
3 TUNNISTAUTUMINEN 12
3.1 Autentikaatio 13
3.2 Autorisaatio 14
3.3 Identiteetin- ja pääsynhallinta 15
3.3.1 Identiteetintarjoaja 16
4 TEKNOLOGIAT 17
4.1 Identiteetti ja tietoturva 17
4.2 Entra ID 18
4.2.1 Vaihtoehtoiset ratkaisut 19
4.2.2 Federointi toisen IdP:n kautta 19
4.2.3 Paikalliset tilit 20
4.2.4 Hakemisto- ja AD-integraatiot (LDAP/AD FS) 21
4.3 OAuth 2.0 ja OpenID Connect 22
4.4 Tokenit (ID token vs Access token) 24
4.5 JWT ja allekirjoituksen validointi 25
4.6 Roolit ja rajapintaoikeudet 26
4.7 Väitteet 27
4.8 MSAL, token-välimuisti ja istunnonhallinta 28
4.9 Toteutusteknologiat ja kehitysympäristö 29
5 TOIMINNALLINEN OSUUS 30
5.1 Lähtötilanne 30
5.2 Suunnitelma 31
5.3 Toteutus 33
5.3.1 Entra ID -konfigurointi 34
5.3.2 Valintasivu 39
5.3.3 Node-taustapalvelu 41
5.3.4 ParkkiV-taustapalvelu 45
5.3.5 Ajonaikainen kirjautumisvirta 49
5.4 Lopputulos 50
6 POHDINTA 52
LÄHTEET 56

LYHENTEET JA TERMIT

ABAC	Attribute-Based Access Control
ACL	Access Control List
AD	Active Directory
AD FS	Active Directory Federation Services
API	Application Programming Interface
JSON	JavaScript Object Notation
IAM	Identity and Access Management
IdP	Identity Provider
JWE	JSON Web Encryption
JWK	JSON Web Key
JWKS	JSON Web Key Set
JWS	JSON Web Signature
JWT	JSON Web Token
LDAP	Lightweight Directory Access Protocol
MFA	Multi-Factor Authentication
MSAL	Microsoft Authentication Library
OIDC	OpenID Connect
RBAC	Role-Based Access Control
SAML	Security Assertion Markup Language
SSO	Single Sign-On

1 JOHDANTO

Työn tavoitteena on uudistaa olemassa olevan pysäköinninvalvontajärjestelmän tunnistautuminen käyttämään Microsoft Entra ID -menetelmää. ParkkiV on monikerroksinen pysäköinninvalvontajärjestelmä, jota käyttävät kaupunkien ja kuntien pysäköinninvalvontayksiköt ja kentällä liikkuvat tarkastajat. ParkkiV koostuu toimistonäkymästä ja tarkastajien mobiilisovelluksesta. Entra ID -kirjautuminen tehdään toimistonäkymälle.

Nykyisessä ParkkiV-järjestelmässä on kansallisen tietoturvavaatimuksen puitteissa toteutettu kaksivaiheinen tunnistus, mutta asiakkaat toivovat lisäksi tukea kaupungin omille yhtenäisille kirjautumiskäytännöille. Tavoitteena on toteuttaa ratkaisu, joka mahdollistaa asiakasorganisaatioiden keskitetyn käyttäjähallinnan ja yhdenmukaisen kirjautumiskokemuksen eri asiakkaille ja eri ParkkiV-asennuksille.

Työ päätettiin toteuttaa osana ParkkiV-järjestelmän kehitystä, koska Entra ID -kirjautuminen nousi esiin hankintakontekstissa erillisenä laatutekijänä. ParkkiV osallistui tammikuussa 2026 Lappeenrannan kaupungin pysäköinninvalvonnan kilpailutukseen, jossa Entra ID -kirjautumisen asiakasreferenssi toi sovellukselle erillisiä laatupisteitä tarjousvertailussa. Tämän vuoksi toteutuksella oli sekä tekninen että aikataulullinen tavoite ja se tuli saada valmiiksi määräaikaan mennessä.

Ratkaisun lähtökohtana on moniasiakasympäristö, jossa sama ParkkiV-sovellus on käytössä useilla eri kaupungeilla erillisinä asiakasympäristöinä. Järjestelmän käyttäjällä voi olla käyttöoikeus yhteen tai useampaan kaupunkiympäristöön. Tämän vuoksi kirjautumisratkaisun tulee tunnistaa käyttäjän oikeudet ja ohjata hänet oikeaan asiakasympäristöön tai tarjota valinta useiden vaihtoehtojen välillä.

2 PROJEKTIN ESITTELY

Työn toimeksiantajana toimii Finn-ID Oy. Finn-ID on 1986 perustettu logistiikan digitalisaatioon keskittyvä kokonaisratkaisukeskeinen yritys, jonka työntekijämäärä on noin 40-50 henkilöä. Finn-ID:n tarjontaan kuuluu ohjelmistoja, laitteita ja asiantuntijapalveluita useille toimialoille, kuten sisälogistiikkaan, terveydenhuoltoon ja teollisuuteen (Finn-ID 2026).

Projektissa toteutetaan web-pohjainen valintasivu, sitä palveleva taustapalvelu ja siihen liittyvä palvelinpuolen komponentti. Valintasivulta loppukäyttäjä (pysäköinninvalvoja) ohjataan automaattisesti Entra ID -tunnistautumiseen. Onnistuneen tunnistautumisen jälkeen käyttäjä ohjataan omaan kaupunki-/asiakasympäristöönsä (ParkkiV-asennukseen). Ratkaisu tukee myös tilannetta, jossa käyttäjällä on Entra ID:ssä määritettyjen roolien perusteella pääsyoikeus useampaan eri ParkkiV-asennukseen. Tällöin valintasivu muodostaa käyttäjälle kaupunkilistan hänen käyttöoikeuksiensa mukaisesti ja mahdollistaa siirtymisen haluttuun ParkkiV-instanssiin yhden kirjautumisen avulla (SSO).

ParkkiV-taustapalveluun luodaan toimintamalli, jonka avulla ParkkiV-instanssit voivat luottaa Entra ID:n tuottamaan tunnistetietoon. Valintasivun kautta välitetään token ParkkiV-kirjautumisrajapintaan. ParkkiV:n taustapalvelu validoi tokenin sekä tarkistaa oikeudet ennen istunnon muodostamista. Näin käyttäjän tunnistautuminen ja käyttöoikeuksien hallinta keskitetään Entra ID:hen, mutta varsinainen pääsynvalvonta toteutetaan kuitenkin ParkkiV-ympäristössä.

Opinnäytetyön aikana kartoitetaan myös autentikaation ja autorisaation nykytilaa ja historiaa. Lisäksi työssä vertaillaan Entra ID:lle mahdollisia vaihtoehtoisia ratkaisuja. Vertailun tavoitteena on tunnistaa, millaisia muita identiteetin- ja käyttöoikeuksienhallinnan menetelmiä vastaavissa järjestelmissä käytetään, ja millaisia hyötyjä ja rajoitteita niihin liittyy ParkkiV:n käyttötapauksen näkökulmasta.

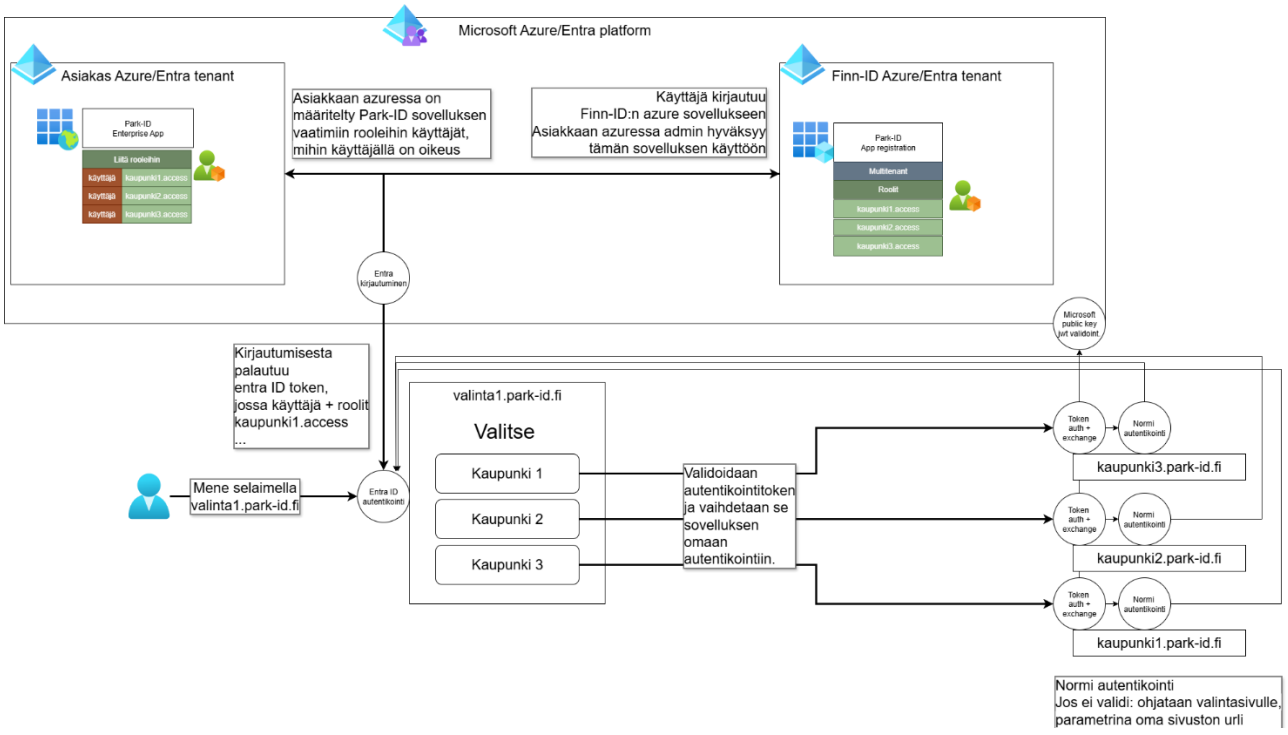
2.1 Tarkoitus

Projektin tarkoituksena on kartoittaa ja toteuttaa yleisesti käytössä oleviin vaatimuksiin perustuva Entra ID -kirjautumisratkaisu. Käytännössä tämä tarkoittaa, että käyttäjien tunnistautuminen ja käyttöoikeuksien hallinta perustuvat asiakkaan oman Entra ID -ympäristön käyttäjiin, ryhmiin ja rooleihin. Kirjautumista ei tarvitse hallita enää erillisellä sovelluskohtaisella käyttäjä-salasanaratkaisulla.

Ratkaisu toteutetaan ennalta määritettyjen vaatimusten pohjalta:

- käyttäjät kirjautuvat kaupungin omilla tunnuksilla organisaation identiteetinhallintaan,
- pääsy voidaan rajata kaupunki-/asiakasympäristöittäin eri rooleilla,
- osa käyttäjistä hallinnoi useampaa kaupunkia ja tarvitsee pääsyn useisiin asennuksiin yhdellä kirjautumisella.

Kuvassa esitetään Entra ID -ratkaisun ensimmäinen arkkitehtuurikuvaus.



KUVIO 1. Entra ID -ratkaisun ensimmäinen arkkitehtuurikuvaus

2.2 Tavoitteet

Projektin päätavoitteena on toteuttaa Entra ID -autentikointi ParkkiV-järjestelmään niin, että kirjautuminen toimii luotettavasti sekä yksittäisen kaupungin käyttäjille että usean kaupungin käyttäjille rooliensa mukaisesti. Tavoitteet voidaan kuvata seuraavasti:

Toiminnalliset tavoitteet

- Toteutetaan moniasiakasympäristöä tukeva Entra ID -sovellusrekisteröinti (App Registration), jota asiakkaiden hakemistot (tenant) voivat käyttää kirjautumiseen.
- Toteutetaan valintasivu, joka:
    - ohjaa käyttäjän Entra ID -kirjautumiseen,
    - lukee kirjautumisen jälkeen käyttäjän roolit (esim. kaupunki1.Access, kaupunki2.Access),
    - listaa käyttäjälle vain ne kaupungit/asennukset, joihin hänellä on oikeus,
    - ohjaa käyttäjän valittuun ParkkiV-asennukseen.
- Toteutetaan ParkkiV taustajärjestelmään autentikaatiorajapinta, joka vastaanottaa valintasivun välittämän tokenin, validoi sen ja sallii kirjautumisen vain, jos käyttäjän rooli ja hakemistotunniste oikeuttaa kyseiseen asennukseen.

Käytettävyys- ja käyttökokemustavoitteet

- Jos käyttäjällä on oikeus vain yhteen asennukseen, kirjautuminen etenee automaattisesti ilman erillistä valintaa.
- Jos käyttäjällä ei ole yhtään vaadittua roolia, käyttäjälle näytetään ilmoitus siitä, ettei käyttöoikeutta ole.

Tietoturva- ja laatuvaatimukset

- Tokenin validointi tehdään NIST:n digitaalisen identiteetin ohjeistuksen sekä JWT/OIDC-standardien mukaisesti. (Temoshok ym. 2025; Jones ym. 2015; Sakimura ym. 2014.)
- Ratkaisu tukee keskitettyä käyttöoikeuksien hallintaa. Asiakas voi liittää käyttäjiä/ryhmiä rooleihin omassa Entra ID -ympäristössään.

- Toteutus dokumentoidaan siten, että ratkaisu on ylläpidettävä ja skaalautuva.

3 TUNNISTAUTUMINEN

Tunnistautumisella tarkoitetaan kokonaisuutta, jossa käyttäjän, palvelun tai sovelluksen identiteetti ensin todennetaan (autentikaatio) ja sen jälkeen sille myönnetään tai evätään pääsy kohdejärjestelmän resursseihin ennalta määritettyjen sääntöjen perusteella (autorisaatio). Tunnistautumiseen kytkeytyy lähes aina istunnonhallinta, jolloin järjestelmän on kyettävä säilyttämään tieto siitä, että käyttäjä on jo todennettu, eikä sen tarvitse todentaa itseään jokaisella yksittäisellä pyynnöllä. (Temoshok ym. 2025; s. 5-6, 16.)

Modernissa web-ympäristössä tunnistautuminen toteutetaan usein federoituna, jolloin sovellus ei hallitse käyttäjän salasanoja, vaan luottaa identiteetintarjoajaan (IdP). Kirjautumisen tuloksena sovellus saa tyypillisesti käyttäjän tunnistetiedot sekä tarvittavat tunnisteet (esim. tokenit). Federoidun tunnistautumisen avulla voidaan toteuttaa kertakirjautuminen useisiin palveluihin. Samalla kirjautumispolitiikat ja esimerkiksi monivaiheinen tunnistautuminen voidaan keskittää identiteetintarjoajaan. (Descope 2023.)

Tunnistautumisen tietoturvan kannalta olennaista on, että istuntoa hallitaan turvallisesti ja että kirjautumisen tuottamia tunnistetietoja (esim. tokenin väitteitä) käsitellään luotettavasti. OWASP:n ohjeistuksen mukaan istunnoissa tulee huomioida esimerkiksi istuntojen aikarajat ja istuntotunnisteiden tulee olla vaikeasti arvattavia. (OWASP n.d.)

Tässä työssä tunnistautuminen ulkoistetaan identiteetintarjoajalle. Sovellus tekee käyttöoikeuspäätökset käyttäjän organisaation ja roolien perusteella ja ohjaa käyttäjän oikeaan ParkkiV-instanssiin. Teknologiat ja protokollat esitellään tarkemmin luvussa 4, ja luvussa 5 kuvataan käytännön toteutus valintasivun ja ParkkiV-instanssien käyttöoikeuslogiikan näkökulmasta.

3.1 Autentikaatio

Autentikaatiolla tarkoitetaan prosessia, jossa käyttäjä tai järjestelmä todentaa identiteettinsä. Autentikaatio perustuu todentamistekijöihin, joita ovat esimerkiksi se mitä käyttäjä tietää (salasana), mitä käyttäjällä on (laite tai kertakoodi) sekä jokin käyttäjään liittyvä ominaisuus (sormenjälki tai kasvojentunnistus). Useamman todentamistekijän käyttö kirjautumisessa (MFA) nostaa todentamisen luotettavuutta ja turvallisuutta. (Temoshok ym. 2025, s. 113, 6-7.)

Tietotekniikkaan liitetyn autentikaation varhainen kehitys kytkeytyy 1960-luvun time-sharing-järjestelmiin, joissa useat eri käyttäjät jakoivat saman tietokoneen. MIT:ssä kehitetty CTSS-järjestelmä otti 1960-luvun alussa käyttöön salasanapohjaisen kirjautumisen. Tuohon aikaan kehitetty ratkaisu on nykystandardeilla heikko. Salasanat säilytettiin järjestelmässä tekstitiedostoissa ja kuka tahansa pystyi tulostamaan ne näkyviin. (Walden & Van Vleck 2011, s. 7, 19.)

Seuraava keskeinen kehitysaskel oli luopuminen selväkielisten salasanojen tallentamisesta. Morris ja Thompson kuvaavat, kuinka varhaisessa UNIX-järjestelmässä salasanat olivat aluksi tallennettuina sellaisinaan, mutta ratkaisua pidettiin tietoturvan kannalta kestämättömänä. Tämän seurauksena siirryttiin malliin, jossa järjestelmään tallennettiin vain salasanasta muodostettu kryptografinen arvo, jota verrattiin kirjautumisen yhteydessä käyttäjän syötteeseen. (Morris & Thompson 1979, s. 594-595.)

1970-luvulla autentikointia vahvistivat julkisen avaimen kryptografiaan perustuvat menetelmät. Diffie ja Hellman esittivät vuonna 1976 julkisen avaimen kryptosysteemin periaatteen sekä ajatuksen siitä, että kryptografialla voidaan toteuttaa kirjallista allekirjoitusta vastaava digitaalinen todennus. Tämä loi perustan myöhemmille digitaalisille allekirjoituksille ja vahvemmille autentikointiratkaisuille. (Diffie & Hellman 1976, s. 644-645, 647.)

Myöhemmin autentikointi kehittyi edelleen kohti keskitetympiä ja laajemmin integroitavia ratkaisuja. Kerberos-järjestelmä kehitettiin tavoitteena tarjota pääsy useisiin eri tietojenkäsittelypalveluihin yhdellä kirjautumisella. Nykyaikaisissa

ympäristöissä sama kehityssuunta näkyy kertakirjautumisessa ja monivaiheisessa tunnistautumisessa. Enterprise Single Sign-On Playbookin mukaan kertakirjautuminen helpottaa sovellusten käyttöä, koska käyttäjä voi siirtyä useisiin palveluihin saman tunnistautumisen avulla. Samalla organisaatio voi yhdenmukaistaa kirjautumiskäytäntöjä ja ottaa monivaiheisen tunnistautumisen käyttöön myös sellaisissa sovelluksissa, joissa sitä ei ole toteutettu valmiiksi. (Saltzer 2021, s. 89; General Services Administration 2021, s. 4.)

3.2 Autorisaatio

Autorisaatio määrittää, mihin kohdejärjestelmän resursseihin autentikoidulla identiteetillä (käyttäjällä, palvelulla tai sovelluksella) on pääsy. Auth0 kuvaa autorisaation nimenomaan todennetun identiteetin toimintaoikeuksina ja resurssipääsynä. (Auth0 n.d.)

Identiteetin autorisaatio tehdään ennalta määritettyjen sääntöjen ja politiikan perusteella. Päätös tehdään esimerkiksi identiteetin roolin, oikeuksien (permissions) tai jonkin muun attribuutin perusteella (kuten identiteetin sijainti tai aika).

Nykypäivän tietoturvamalleissa korostuvat erityisesti Zero Trust ja vähimmän oikeuden periaate. Zero Trust -arkkitehtuuri perustuu ajatukseen, jossa jokainen sisään tuleva pyyntö käsitellään kuin se tulisi ulkoverkosta. Tämän vuoksi jokainen identiteetti autorisoidaan ennen kuin se voi käyttää resursseja, riippumatta siitä ovatko he organisaation sisä- tai ulkoverkossa. (Rose ym. 2020, s. 1-2.)

Vähimmän oikeuden periaate tarkoittaa, että käyttäjälle tai prosessille annetaan vain ne oikeudet, jotka ovat välttämättömiä työtehtävien suorittamiseksi. Tavoitteena on pienentää vahingon ja väärinkäytön vaikutusta, jos tunnus tai palvelu vaarantuu. Käytännössä tämä näkyy esimerkiksi siten, että käyttöoikeudet sidotaan rooleihin tai tehtäviin ja myönnetään tarvittaessa määräajaksi. Ylimääräiset oikeudet poistetaan, kun tarve päättyy. (NIST n.d.b.)

Autorisaatio voidaan toteuttaa useilla yleisillä malleilla. Tavallisimpia niistä ovat:

- RBAC (Role-Based Access Control): oikeudet myönnetään roolien kautta (esim. *Valvoja*, *Pääkäyttäjä*). (Ferraiolo & Kuhn 1992, s. 554-555.)

- ABAC (Attribute-Based Access Control): päätös tehdään attribuuttien ja sääntöjen perusteella (subjekti-objekti-toiminto-ympäristö). (Hu ym. 2019, s. 1-3.)

- ACL/Permission-mallit: käyttöoikeuksia hallitaan sääntölistoilla, joilla määritetään, hyväksytäänkö vai estetäänkö pääsy tiettyyn resurssiin. (Stouffer ym. 2023, s. 98)

3.3 Identiteetin- ja pääsynhallinta

Identiteetin ja pääsynhallinta (IAM) ei tarkoita yhtä yksittäistä järjestelmää, vaan kokonaisuutta prosesseja, käytäntöjä ja teknologioita. Sen avulla hallitaan digitaalisten resurssien käyttäjiä ja heidän käyttöoikeuksiaan. IAM mahdollistaa käyttäjien, palveluiden tai sovellusten identiteetin todentamisen (autentikaatio) ja pääsyn tarvittaviin resursseihin ennalta määritettyjen sääntöjen mukaisesti (autorisaatio). IAM:n tavoitteena on varmistaa, että oikealla toimijalla on oikeat oikeudet oikeaan resurssiin oikeaan aikaan. Samalla se tukee käyttöoikeuksien hallittavuutta sekä tapahtumien jäljitettävyyttä lokituksella ja auditoinnilla. (Microsoft 2026.)

IAM:n olennaisiin osiin kuuluu identiteetin elinkaaren hallinta. Tällä tarkoitetaan usein joiner-mover-leaver-tilannetta, jossa käyttäjätili ja oikeudet perustetaan, työtehtävien muuttuessa oikeudet muutetaan ja tarpeen loppuessa oikeudet poistetaan. Elinkaaren hallinta vähentää riskiä siitä, että järjestelmään jää ”unohtuneita” ja ylimitoitettuja oikeuksia. (Microsoft 2026.)

IAM:n ytimessä ovat autentikaatio ja autorisaatio. Ensin todennetaan identiteetti, jonka jälkeen päätetään mihin resursseihin identiteetillä on pääsy ja mitä toimintoja se saa tehdä. Identiteetille annetut oikeudet voidaan mallintaa esi-

merkiksi rooleilla, ryhmillä ja politiikalla, jolloin hallinta on keskitettyä ja ylläpidettävää. Tässä opinnäytetyössä IAM-kokonaisuus näkyy Entra ID -kirjautumisen kautta. (Microsoft 2026.)

3.3.1 Identiteetintarjoaja

Identiteetintarjoaja (IdP) on palvelu, joka luo ja ylläpitää identiteettitietoja. Se tarjoaa sovelluksille autentikaatiopalvelun ja todentaa käyttäjän tai palvelun identiteetin. IdP välittää kirjautumiseen liittyvän luotettavan tiedon sovellukselle esimerkiksi ID tokenin, access tokenin tai muun varmenteen muodossa. (Microsoft 2025a.)

IdP voi välittää sovellukselle myös käyttäjää kuvaavia attribuutteja ja väitteitä, kuten nimen, ryhmät ja roolit. Sovellus voi hyödyntää näitä tietoja autorisaatiopäätöksen tekemisessä, mutta varsinainen sisäänpääsypäätös tehdään kuitenkin kohdejärjestelmän omien sääntöjen perusteella. (Sakimura ym. 2014.) IdP voi olla organisaation oma (esimerkiksi Microsoft Entra ID) tai ulkoinen tunnistetarjoaja, jolloin käyttäjä voi kirjautua sovellukseen jo olemassa olevilla tunnuksillaan (social login). (Microsoft 2025a.)

4 TEKNOLOGIAT

Tässä luvussa esitellään projektissa käytetyt teknologiat ja niiden rooli kokonaisuudessa. Tavoitteena on antaa lukijalle käsitteellinen pohja luvun 5 toteutuksen ymmärtämiseksi. Luvussa kuvataan Entra ID:hen perustuva federoitu kirjautuminen, OAuth 2.0 / OpenID Connect -protokollat, JWT-tokenit sekä sovelluksen istunnonhallinta. Lisäksi käsitellään käyttöoikeuksien mallinnusta roolien ja rajapintaoikeuksien (scope) avulla sekä vaihtoehtoisia kirjautumisratkaisuja ParkkiV:n näkökulmasta.

4.1 Identiteetti ja tietoturva

Nykypäivän web- ja pilviympäristössä tunnistautuminen ja käyttöoikeuksien hallinta toteutetaan yhä useammin keskitetysti identiteetintarjoajan (IdP) avulla. Tämä vähentää yksittäisen ohjelman tietoturvakuormaa ja vastuuta, koska sen ei tarvitse tallentaa ja käsitellä käyttäjien salasanoja. Keskitetty malli helpottaa myös organisaation yhteisten kirjautumiskäytäntöjen toteuttamista. Tällöin esimerkiksi vahvemmat tunnistautumismenetelmät voidaan keskittää yhden palvelun, kuten Entra ID:n, hallintaan. (Grassi ym. 2020.)

Tietoturvan näkökulmasta sovelluksen vastuulle jää kirjautumisen jälkeinen turvallisuus. Käytännössä tämä tarkoittaa erityisesti istunnonhallintaa ja suojattujen toimintojen käyttöoikeuksien valvontaa. Sovelluksen tulee säilyttää tieto siitä, että käyttäjä on jo todennettu ja samalla suojata istuntotunnisteet selaimessa. Hyviin käytänteisiin kuuluu esimerkiksi istuntojen aikarajojen asettaminen sekä evästeiden suojausasetukset (esim. HttpOnly/Secure). Lisäksi kirjautumisen yhteydessä käsiteltyjä tunnisteita ei tule säilyttää tarpeettomasti eikä tallentaa pysyvästi selaimen puolelle. (OWASP n.d.)

4.2 Entra ID

Microsoft Entra ID (aiemmin Azure Active Directory) on pilvipohjainen identiteetin- ja pääsynhallinnan palvelu (IAM). Se toimii organisaation Entra-hakemistona, jossa ylläpidetään käyttäjien ja ryhmien tietoja ja sovelluksiin liittyviä käyttöoikeuksia. (Microsoft 2025b.)

Entra ID toimii myös identiteetintarjoajana. Se toteuttaa käyttäjien tunnistautumisen (autentikaatio), soveltaa organisaation asettamaa kirjautumispolitiikkaa ja myöntää kirjautumisen tuloksena kohdeohjelmille tietoturvatokeneita (esim. access token, ID token). Varsinaiset käyttöoikeuspäätökset tehdään sovelluksessa tai resurssipalvelussa tokenin sisältämien väitteiden perusteella. (Microsoft 2025b.)

Jotta sovelluksen tunnistautuminen ja käyttöoikeuksien hallinta voidaan delegoida Entra ID:lle, on se ensin rekisteröitävä Entra ID:hen. Tätä kutsutaan sovellusrekisteröinniksi (App registration). Se luo sovellukselle sovellusobjektin (application object), joka sisältää keskeiset asetukset, tunnistetiedot ja mahdolliset roolit. (Microsoft 2024b.)

Kun organisaatio ottaa rekisteröidyn sovelluksen käyttöön omassa tenantissaan, siihen syntyy sovelluksesta yrityssovellus (Enterprise application). Teknisesti tämä vastaa service principal -objektia. Yrityssovellus on saman sovelluksen tenant-kohtainen instanssi, jonka kautta hallitaan sovelluksen käyttöä kyseisessä organisaatiossa. Yrityssovelluksessa määritellään esimerkiksi, ketkä käyttäjät tai ryhmät saavat käyttää sovellusta. Sovellukseen määritetyt roolit jaetaan käyttäjille yrityssovelluksen kautta. (Microsoft 2024a.)

Tässä työssä Entra ID:tä käytetään federoidun kirjautumisen toteuttamiseen OAuth 2.0 / OpenID Connect -mallilla. Lisäksi Entra ID:n sovellusrooleja (app roles) hyödynnetään kuvaamaan kaupunkikohtaisia käyttöoikeuksia ParkkiV-asennuksiin, jolloin käyttäjälle voidaan näyttää ja avata vain ne ympäristöt, joihin hänen roolinsa oikeuttavat.

4.2.1 Vaihtoehtoiset ratkaisut

ParkkiV:n näkökulmasta vaihtoehtoisia tunnistautumis- ja käyttöoikeusratkaisuja voidaan tarkastella kolmen kokonaisuuden kautta. Näitä ovat federoitu kirjautuminen jonkin toisen identiteetintarjoajan avulla, sovelluksen oma käyttäjähallinta, eli paikalliset tilit sekä hakemisto- ja AD-pohjaiset integraatiot. Vertailun tavoitteena ei ole kuvata kaikkia teknisiä yksityiskohtia, vaan arvioida vaihtoehtoja ennalta määritettyjen kriteerien perusteella. Arviointikriteereihin nostettiin seuraavat vaatimukset: moniasiakkuus (eri asiakkaat ja tenant-erottelu), roolien ja ryhmien hallinta, vahvan tunnistautumisen ja kirjautumispolitiikan tuki, ylläpidon vastuunjako, integraation tekninen sopivuus (web + API, tokenit, istunnot) sekä käyttäjäkokemus ja käyttöönotto.

4.2.2 Federointi toisen IdP:n kautta

Federoidussa kirjautumisessa sovellus ei tunnista käyttäjää omilla salasanoillaan, vaan käyttäjä kirjautuu organisaation valitsemaan identiteetintarjoajaan. Käytännössä tämä toteutetaan useimmiten OpenID Connect -mallilla tai SAML 2.0 -mallilla. OIDC pohjautuu OAuth 2.0 -valtuutuskehykseen ja välittää kirjautumisen tulokset tokeneina, kun taas SAML välittää tulokset allekirjoitettuna assertion-sanomana. (Hardt 2012, s. 4-10; Sakimura ym. 2014, luku 1 ja 3; OASIS 2005, s. 14-15.)

ParkkiV:n näkökulmasta federoitu malli tarkoittaa sovelluksen integroimista valittuun IdP:hen sekä käyttöoikeuksien mallintamista hallittavalla tavalla. OIDC:llä tämä tarkoittaa access tokenin ja ID tokenin validointia ja niiden sisältämien väitteiden hyödyntämistä. SAML-mallissa sovelluksen on pystyttävä validoimaan assertion ja poimimaan siitä tarvittavat attribuutit. Molemmat mallit voidaan sovittaa moniasiakkuuteen, mutta se edellyttää organisaatioiden erottelun ja rooli/attribuuttikartoitusten määrittelyä. (Sakimura ym. 2014, luku 3; OASIS 2005, s. 14-15.; Microsoft 2024c.)

ParkkiV:n web+API-arkkitehtuurissa OIDC:n token-malli vähentää integraatiopolkujen määrää verrattuna SAML-painotteiseen ratkaisuun. OIDC kattaa selainkirjautumisen ja API-valtuutuksen, jonka takia federointi on suoraviivaisempi toteuttaa. (Sakimura ym. 2014, luku 1 ja 3; Hardt 2012, s. 4-10.)

Lopulliseen valintaan vaikutti myös hankintakonteksti. Asiakkaiden vaatimuksissa korostui keskitetty Entra ID -kirjautuminen, minkä vuoksi integraation toteuttaminen oli teknisen ratkaisun lisäksi myös sovelluksen kilpailukykyyn vaikuttava tekijä.

ParkkiV:n kriteereillä OIDC/SAML-federointi tukee moniasiakkuutta ja SSO:ta, ja roolien/ryhmien hallinta voidaan keskittää IdP:hen. Tekninen sopivuus on vahva erityisesti OIDC-mallissa web+API-kokonaisuuteen. Käyttöönotto edellyttää kuitenkin asiakkaan hallinnollisia toimia. (Sakimura ym. 2014; luku 3; OASIS 2005, s. 14-15.)

4.2.3 Paikalliset tilit

Paikallisilla tileillä tarkoitetaan mallia, jossa ParkkiV ylläpitää omaa käyttäjärekisteriä ja samalla vastaa käyttäjien tunnistautumisesta omalla kirjautumistoiminnallaan. Tämä edellyttää, että sovellus tallentaa käyttäjätiedot, hoitaa salasanapolitiikan ja toteuttaa mahdolliset lisävaatimukset, kuten monivaiheisen tunnistautumisen, salasanan palautuksen ja kirjautumisen valvonnan. Paikalliset tilit ovat tyypillisempiä pienemmissä järjestelmissä tai kuluttajapalveluissa, joissa asiakkaalla ei ole valmista IAM-ympäristöä, sekä käyttäjämäärät pysyvät suhteellisen alhaisina. (Grassi ym. 2020.)

ParkkiV:n käyttötarkoitukseen paikalliset tilit vaatisivat huomattavasti enemmän tietoturva- ja ylläpitotoimia. Sovelluksen tulisi toteuttaa turvallinen tunnisteiden käsittely, istunnonhallinta, tilien elinkaaren hallinta sekä lokitus ja valvonta. Moniasiakkuus tuo myös lisävaatimuksia. Käyttäjät ja oikeudet on eroteltava asiakasorganisaatioittain, ja kaupunkikohtainen käyttöoikeusmalli on rakennettava kokonaan sovelluksen sisälle. Myös organisaatioiden omat kirjautumissäännöt pitäisi toteuttaa sovelluksessa erikseen. (Grassi ym. 2020.; OWASP n.d.)

Paikallisten tilien etuna on matalan tason kontrolli. Sovellus voi määritellä esimerkiksi käyttäjämallinsa ja käyttöoikeuslogiikkansa riippumatta ulkoisesta IdP:stä. Myös käyttöönotto ulkoisella organisaatiolla on huomattavasti suoraviivaisempaa.

Riskeinä ovat tietoturvavastuun kasvu ja ylläpidon kustannukset. Salasanojen käsittely altistaa sovelluksen laajemmalle uhkakentälle, ja pienetkin toteutusvirheet voivat olla vakavia. Lisäksi käyttäjäkokemuksen kannalta paikalliset tilit eivät tarjoa organisaatioiden tavoittelemaa SSO-kokemusta, eikä käyttäjien poistuminen organisaatiosta tai käyttöoikeuksien muuttuminen heijastu sovellukseen ilman erillisiä prosesseja. (OWASP n.d.; Grassi ym. 2020.)

Yhteenvetona voidaan todeta paikallisten tilien olevan ParkkiV:n käyttötapauksessa heikompi vaihtoehto. Ne lisäävät huomattavasti työmäärää, sovelluksen tietoturva- ja ylläpitovastuuta sekä vaikeuttavat asiakkaiden omien IAM-käytäntöjen hyödyntämistä.

4.2.4 Hakemisto- ja AD-integraatiot (LDAP/AD FS)

Hakemisto- ja AD-integraatioissa lähtökohtana on organisaation oma käyttäjä- ja ryhmähakemisto, kuten Active Directory. LDAP on tyypillisesti hakemiston kyselyrajapinta, jota voidaan käyttää käyttäjätietojen ja ryhmäjäsenyyksien hakemiseen. AD FS (tai vastaava) puolestaan toimii organisaation federointipalveluna. Se voi tarjota SSO:n sekä julkaista kirjautumisen tuloksen sovellukselle protokollien, kuten SAMLin tai OIDC:n, kautta. AD FS/SAML-malli on on-premises-ympäristöissä ollut pitkään yleinen tapa toteuttaa SSO yrityksen omasta hakemistosta. (OASIS 2005, s. 2, 48.)

ParkkiV:n käyttötarkoituksen kannalta LDAP-integraatio ei yksin ratkaise web-kirjautumista. Sen lisäksi tarvitaan erillinen identiteettikerros, joka hoitaa selaimen kirjautumisvirran, MFA:n sekä tokenien tai assertioiden tuottamisen. LDAP-ratkaisu tarkoittaa käytännössä sitä, että organisaation hakemisto pysyy käyttä-

jien lähteenä, mutta kirjautuminen toteutetaan federointipalvelun kautta. Integraatio vaatii silloin sekä yhteensopivan federointiprotokollan että attribuuttien ja roolien/ryhmien kartoitukset, jotta kaupunkikohtaiset käyttöoikeudet saadaan tuotua sovelluksen autorisaatiopäätöksen pohjaksi. (OASIS 2005, s. 29-31, 45-48; Sakimura ym. 2014, luku 3.)

LDAP-ratkaisun hyöty on esimerkiksi se, että organisaatiot voivat hyödyntää olemassa olevaa käyttäjähakemistoaan ja sen hallintaprosesseja. Kirjautumiseen voidaan myös liittää organisaation oma kirjautumispolitiikka ja federointipalvelu. Tämän merkitys korostuu erityisesti organisaatioissa, joissa identiteettiinfrastruktuuri on vahvasti on-premises-painotteinen. (Microsoft 2025f.)

LDAP-ratkaisun rajoitteita ovat arkkitehtuurin monimutkaisuus ja käyttöönoton vaihtelu asiakaskohtaisesti. On-premises -komponentit lisäävät usein operatiivista kuormaa, ja integraatio voi vaatia enemmän asiakaskohtaista konfiguraatiota kuin pilvi-IdP:hen perustuva suora OIDC-integraatio. (Microsoft 2025f; Microsoft 2025.)

Yhteenvetona hakemisto- ja AD-integraatiot ovat ParkkiV:lle varteenotettava vaihtoehto lähinnä silloin, kun asiakasorganisaation identiteettiympäristö on ensisijaisesti on-premises ja federointi hoidetaan valmiiksi esimerkiksi AD FS:n kautta. Pilvi-IdP ja OIDC yleensä kuitenkin tuottavat suoraviivaisemman ja yhtenäisemmän käyttöönoton. (Sakimura ym. 2014, luku 1 ja 3; Microsoft 2025f.)

4.3 OAuth 2.0 ja OpenID Connect

OAuth 2.0 on laajasti käytetty valtuutuskehys web-, työpöytä- ja mobiiliympäristöissä. Siinä käyttäjä tunnistautuu valtuutuspalvelimelle, joka myöntää sovellukselle access tokenin organisaation käytäntöjen mukaisesti. (Hardt 2012, s. 4-7.)

OAuth 2.0 on kehitetty ratkaisemaan perinteiseen käyttäjätunnus-salasana-malliin liittyviä ongelmia tilanteissa, joissa kolmannen osapuolen sovellus tarvitsee pääsyn käyttäjän suojattuihin resursseihin. Ilman OAuth-protokollaa käyttäjän olisi usein jaettava tunnuksensa, käytännössä salasana, sovellukselle, mikä

johtaa tietoturva- ja hallintaongelmiin. Tällöin sovelluksen olisi käsiteltävä käyttäjän salasanaa itse, mikä kasvattaa tietoturvariskiä. Lisäksi sovellukselle voi syntyä liian laajat oikeudet ilman selkeää rajattavuutta tai määräaikaisuutta. (Hardt 2012, s. 5-7.)

OAuth 2.0 ratkaisee ongelman erillisellä valtuutuskerroksella, jossa ohjelma (client) ei saa käyttäjän tunnuksia, vaan se saa käyttöönsä erillisen access tokenin. Token toimii osoituksena siitä, että sovellukselle on myönnetty rajattu käyttöoikeus. Se kertoo, mille resurssille token on tarkoitettu, mitä oikeuksia sillä on ja kuinka kauan käyttöoikeus on voimassa. Tämän ansiosta käyttöoikeus voidaan rajata tarkasti, ja se voidaan tarvittaessa perua. Tässä työssä tokenin sisältämät, ParkkiV:n kannalta olennaiset väitteet (claims) kuvataan tarkemmin kohdassa 4.7. (Hardt 2012, s. 10)

Tokenin myöntää valtuutuspalvelin (authorization server) käyttäjän tai organisaation hyväksynnän perusteella. Resurssipalvelin (resource server) hyväksyy pyynnöt vain, jos mukana on voimassa oleva token. (Hardt 2012, s. 6, 7.)

OpenID Connect (OIDC) on OAuth 2.0:n päälle rakennettu identiteettikerros. Se standardoi käyttäjän tunnistautumisen ja identiteettitiedon välittämisen sovellukselle. OAuth 2.0 kuvaa, miten sovellus saa käyttää suojattuja resursseja access tokenin avulla, kun taas OIDC määrittelee, miten sovellus saa luotettavan tiedon siitä, kuka käyttäjä on. OIDC välittää kirjautumisen tuloksena ID tokenin, joka sisältää käyttäjää kuvaavia väitteitä. Tämän avulla sovellus voi muodostaa käyttäjäistunnon ja toteuttaa esimerkiksi kertakirjautumisen (SSO) useisiin palveluihin saman identiteetintarjoajan kautta. (Sakimura ym. 2014, luku 1 ja 2.)

Yksinkertaistettuna OAuth 2.0 ja OpenID Connect ovat standardoituja toimintamalleja, joiden avulla kirjautuminen ja käyttöoikeuksien hallinta voidaan toteuttaa turvallisesti ilman, että sovellus käsittelee käyttäjän salasanaa suoraan. Molemmissa ideana on, että käyttäjä kirjautuu suoraan luotettuun identiteetintarjoajaan ja sovellus saa kirjautumisen tuloksena tokeneita. (Sakimura ym. 2014, luku 1 ja 2.)

Edellä kuvatun OAuth 2.0 / OpenID Connect -mallin ymmärtämiseksi on olennaista erottaa tunnistautumista kuvaava ID token ja käyttöoikeuksia kuvaava access token toisistaan, sillä niillä on eri käyttötarkoitus ja niitä käsitellään järjestelmässä eri tavoin.

4.4 Tokenit (ID token vs Access token)

OAuth 2.0- ja OIDC-ratkaisuissa kirjautumisen ja resurssien käyttöön liittyvä tieto välitetään sovelluksille tokeneina. ID token kuvaa käyttäjän identiteettiä ja kirjautumista, kun taas access token toimii käyttäjän käyttölupana suojattuun resurssiin (tyypillisesti API:iin). (Hardt 2012, s. 5-10; Sakimura ym. 2014, luku 1 ja 2.)

Access token on merkkijono, jota sovellus käyttää kutsuessaan resurssipalvelua (resource server). Tokenin formaatti voi olla erilainen (esimerkiksi JWT tai opaque), mutta perusidea on sama: se edustaa oikeutta tiettyihin resursseihin ennalta määritetyn ajan. (Hardt 2012, s. 10.)

Tässä opinnäytetyössä ParkkiV:lle välitettävä access token on JWT-muotoinen. Se sisältää väitteitä, kuten aud (mille sovellukselle/resurssille token on tarkoitettu), iss (tokenin myöntäjä) ja exp (voimassaolon päättymisaika). (Jones ym. 2015, s. 9-10.)

ID token on OpenID Connectin määrittelemä token, joka syntyy käyttäjän tunnistautumisen yhteydessä. ID token on turvatunniste, ja toisin kuin access token, sen formaatti on OIDC:ssä aina JWT. ID token sisältää myös väitteitä, jotka kuvaavat kirjautunutta käyttäjää ja autentikaatiotapahtumaa. Kohdesovellus validoi ID tokenin ja käyttää sitä käyttäjäistunnon muodostamiseen. Sen avulla varmistetaan, kuka käyttäjä on kirjautunut sovellukseen. (Sakimura ym. 2014, s. 13-15; Microsoft 2024c.)

Tässä työssä tokenien käyttö etenee seuraavasti. Käyttäjä ohjataan Entra ID -kirjautumiseen, minkä jälkeen valintasivun taustapalvelu vastaanottaa kirjautumisen palautuksen ja vaihtaa autorisaatiokoodin tokeneiksi authorization code

flow'n avulla. (Hardt 2012, s. 24-30; Microsoft 2025d.) Seuraavaksi taustapalvelu muodostaa käyttäjälle valintasivun istunnon tallentamalla MSAL-kirjaston account-tiedon ja tokenivälimuistin istuntoon. Käyttäjän tunnistetiedot ja käyttöoikeudet luetaan tokenin väitteistä ja niiden perusteella muodostetaan käyttäjälle sallittu kaupunkilista.

Kun käyttäjä siirtyy ParkkiV-instanssiin joko automaattisesti tai oman valintansa perusteella, siirrossa välitetään ParkkiV-API:lle myönnetty access token. ParkkiV:n taustapalvelu tekee lopullisen pääsynvalvonnan validoimalla access tokenin (allekirjoitus, issuer, audience ja voimassaolo) sekä tarkastamalla tokenin roolit ja muut instanssikohtaiset hyväksymiskriteerit.

4.5 JWT ja allekirjoituksen validointi

JSON Web Token (JWT) on standardoitu kompakti tapa välittää tunnistautumiseen ja käyttöoikeuksiin liittyvää tietoa verkkoympäristössä eri palveluiden välillä. JWT soveltuu tilanteisiin, joissa kirjautumisen tulos tai valtuutus halutaan siirtää järjestelmien välillä siten, että vastaanottava palvelu voi varmistua tiedon aitoudesta ja muuttumattomuudesta. Tämä perustuu siihen, että token voidaan allekirjoittaa digitaalisesti. Tällöin vastaanottava osapuoli voi varmistaa, että se on peräisin luotettavalta myöntäjältä. (Jones ym. 2015, s. 6-10; Jones ym. 2015a, s. 4-6.)

JWT koostuu tyypillisesti kolmesta osasta: header, payload ja signature. Nämä osat ovat Base64URL-koodattuja ja yhdistetty pisteillä muotoon header.payload.signature. Header kertoo yleensä allekirjoitusalgoritmin (esim. RS256) ja avaimen tunnisteen (kid). Payload sisältää varsinaiset väitteet, kuten tokenin myöntäjän, kohteen ja voimassaoloajan sekä sovelluskohtaisia tietoja, kuten rooleja. Allekirjoitus lasketaan headerin ja payloadin perusteella, ja sen avulla vastaanottaja voi validoida tokenin eheyden ja myöntäjän. (Jones ym. 2015, s. 6-8; Jones ym. 2015a, s. 4-6.)

JWT-token ei ole oletusarvoisesti salattu. Base64URL-koodaus ei ole salausta, vaan se on pelkkä esitysmuoto. Tokenin header ja payload ovat helposti dekoodattavissa. JWT:n allekirjoitus suojaa tokenin eheyttä, eli sisältöä ei voi muuttaa huomaamatta. Se ei kuitenkaan suojaa tokenin luottamuksellisuutta, eli sisältö voi edelleen olla luettavissa. Mikäli tavoitteena on, ettei sisältö ole luettavissa, käytössä on oltava salattu JWT (JWE), tai ”opaque token”. (Jones ym. 2015, s. 6-8; Jones ym. 2015a, s. 4-6; Jones 2015b, s. 1-5.)

Kohdesovellus tekee autorisaatiopäätöksen tokenin sisällön perusteella, jonka takia tokeniin ei voi luottaa sellaisenaan, vaan se pitää ensin validoida. Validoinnissa varmistetaan vähintään neljä asiaa. Ensinnäkin tokenin on oltava myöntäjän allekirjoittama ja muuttumaton. Toiseksi sen on oltava tarkoitettu juuri kyseiselle sovellukselle tai resurssille. Kolmanneksi sen on oltava edelleen voimassa. Neljänneksi myöntäjän on vastattava odotettua identiteetintarjoajaa. (Jones ym. 2015, s. 9-10; Jones ym. 2015a, s. 4-6.)

Entra ID allekirjoittaa myöntämänsä JWT-tokenit omilla yksityisillä avaimillaan. Vastaanottavan palvelun tulee tarkistaa tokenin eheys Entra ID:n julkaisemilla julkisilla avaimilla (JWKS). Julkiset avaimet julkaistaan JSON Web Key Set -muodossa ja ne haetaan Entra ID:n OIDC-discovery-mekanismin kautta. Discovery-dokumentti määrittelee jwks_uri-osoitteen, josta avaimet voidaan noutaa. Tokenin headerissa oleva kid-kenttä (key id) kertoo, millä JWKS-avaimella allekirjoitus tulee tarkistaa. (Jones ym. 2015a, s. 4-6; Jones 2015b, s. 1-5; Sakimura ym. 2023; Microsoft 2024c.)

4.6 Roolit ja rajapintaoikeudet

Käyttöoikeuksien mallinnuksessa on tärkeää erottaa toisistaan sovellustason roolit ja rajapintakutsuihin liittyvät käyttöoikeudet (API scopes). Molemmat tiedot välitetään tyypillisesti tokenien yhteydessä, mutta ne palvelevat eri tarkoitusta. Yleisellä tasolla roolit vastaavat kysymykseen ”kuka käyttäjä on ja mitä hän saa tehdä sovelluksessa”, kun taas rajapintaoikeudet vastaavat kysymykseen ”mihin suojattuun resurssiin sovellus saa kutsuoikeuden”. (Microsoft 2024a; Microsoft 2024b; Hardt 2012, s. 22-24.)

Sovellusroolit ovat Entra ID:n tarjoama tapa toteuttaa roolipohjainen käyttöoikeus (RBAC). Roolit määritellään Entra ID:ssä rekisteröidyn sovelluksen App roles -asetuksissa ja ne voidaan myöntää käyttäjille tai ryhmille sekä myös toisille sovelluksille (service principal). Roolit välitetään tyypillisesti tokenin yhteydessä rooliväitteinä. Tämä sopii tilanteisiin, joissa käyttöoikeus halutaan sitoa sovelluksen sisäiseen oikeusmalliin, kuten ParkkiV:n tapauksessa kaupunkikohtaisiin käyttöoikeuksiin. (Microsoft 2024b; Microsoft 2024a.)

Rajapintaoikeus (scope) (OAuth 2.0 permission / delegated permission) on rajapintakutsun laajuutta kuvaava käyttöoikeus, joka liittyy resurssiin (resource API) ja sen ”Expose an API” -määrittelyyn. Kun sovellus pyytää access tokenia tietylle resurssille, se ilmoittaa samalla haluamansa rajapintaoikeudet, ja valtuutuspalvelin myöntää tokeniin vastaavan laajuuden. Käytännössä resurssipalvelu tarkastaa, että tokenissa on tarvittava rajapintaoikeus ennen suojatun toiminnon suorittamista. Rajapintaoikeudet ovat erityisen tärkeitä tilanteissa, joissa sovellus kutsuu suojattua web-APIa käyttäjän puolesta, ja samalla halutaan pystyä rajaamaan kutsuoikeutta hienojakoisesti (esim. ”read” vs. ”write”). (Hardt 2012, s. 22-24; Microsoft 2024b.)

4.7 Väitteet

JWT-tokenin payload sisältää väitteitä, jotka ovat avain-arvo-pareja ja kuvaavat tokenin kontekstia. Väitteet kertovat esimerkiksi, kuka tokenin myönsi, kenelle se on tarkoitettu, milloin se vanhenee ja mitä käyttöoikeuksia tokenin haltijalla on. Väitteet ovat keskeisiä, sillä vastaanottava palvelu tekee autorisaatiopäätöksiä tokenin väitteiden perusteella. (Jones ym. 2015, s. 8-10; Jones ym. 2015a, s. 4-6; Hardt 2012, s. 22-24.)

ParkkiV:n valintasivuratkaisussa olennaisia väitteitä ovat seuraavat:

- **iss (issuer)**: kertoo tokenin myöntäjän. Vastaanottava palvelu tarkistaa, että arvo vastaa odotettua identiteetintarjoajaa (tässä työssä Entra ID), eikä token ole peräisin muualta. (Sakimura ym. 2014, s. 15-17; Microsoft 2024c.)

- **aud (audience)**: kertoo mille resurssille token on tarkoitettu. ParkkiV-taustapalvelu tarkistaa tämän, jotta tokenia ei voi käyttää väärää palvelua vastaan (esim. toiseen API:iin tai toiseen sovellukseen). (Hardt 2012, s. 22-24; Jones ym. 2015, s. 9-10.)

- **tid (tenant id)**: kertoo Entra ID -hakemiston tunnisteen, jossa käyttäjä on todennettu. Moniasiakasympäristössä sen avulla voidaan tehdä organisaatiokohtaisia tarkastuksia (esim. mihin kaupunki-instansseihin tietyn organisaation käyttäjät voivat päästä). (Microsoft 2024c.)

- **roles**: sisältää käyttäjälle tai ryhmän kautta myönnetyt sovellusroolit. Tässä työssä ne mallintavat kaupunkikohtaisia käyttöoikeuksia. (Microsoft 2024b; Microsoft 2024a.)

4.8 MSAL, token-välimuisti ja istunnonhallinta

Microsoft Authentication Library (MSAL) on kirjasto, jolla helpotetaan sovelluksen integraatioita Microsoftin identiteettialustan (Entra ID) kanssa. MSAL tarjoaa Node.js-ympäristössä valmiit toiminnot OAuth 2.0- ja OpenID Connect -kirjautumisvirtojen toteuttamiseen, kuten kirjautumispyynnön muodostamisen, authorization code flow -vaiheen käsittelyn sekä tokenien hankinnan ja uusinnan. (Microsoft 2024d; Microsoft 2025c.)

MSAL:n myötä sovelluksen ei tarvitse rakentaa kirjautumisvirtoja ja tokenien käsittelyä käsin, vaan se voi käyttää valmiita ja tuettuja kirjastorajapintoja. ParkkiV-valintasivun taustapalvelu käyttää MSAL:ia toteuttamaan authorization code flow -kirjautumisen ja hankkimaan tarvittavat tokenit Entra ID:ltä. (Microsoft 2025d; Microsoft 2024d; Microsoft 2025c.)

Token-välimuisti on MSAL:n ylläpitämä välimuisti, johon tallennetaan aiemmin hankitut tokenit ja niiden metatiedot. Välimuistin avulla sovellus voi hankkia tokenin hiljaisesti (silent token acquisition). Tällöin käyttäjää ei tarvitse ohjata Entra ID:n kirjautumissivulle, mikä parantaa käyttäjäkokemusta ja vähentää kirjautumisvirran toistumista. MSAL pyrkii ensisijaisesti hyödyntämään välimuistia ennen uuden tokenin pyytämistä. (Microsoft 2025e.)

Istunto (session) kuvaa käyttäjän tilaa kirjautumisen jälkeen, eli sitä, onko käyttäjä sovelluksen näkökulmasta kirjautunut vai ei. Vaikka tunnistautuminen on ulkoistettu Entra ID:lle, sovelluksen on silti hallittava oma istuntonsa, jotta käyttäjää ei tarvitse todentaa jokaisella yksittäisellä pyynnöllä. Web-sovelluksissa istunto toteutetaan usein evästeellä (cookie), joka sisältää istuntotunnisteen ja varsinaiset istuntoon liitetyt tiedot (esim. käyttäjän account-viite sekä tokenien hakemiseen tarvittava tila) säilytetään palvelinpuolella. (OWASP n.d.)

Istunnonhallinnan tietoturvan kannalta olennaista on asettaa istunnolle elinikä, uudistaa istunto kirjautumisen yhteydessä sekä suojata evästeet asianmukaisilla asetuksilla. OWASP korostaa lisäksi istuntojen aikarajoja, evästeattribuutteja ja sitä, että istuntotunnisteiden käsittelyn tulee minimoida väärinkäytön mahdollisuus (OWASP n.d.)

4.9 Toteutusteknologiat ja kehitysympäristö

ParkkiV-valintasivuratkaisu koostuu valintasivun käyttöliittymästä, sitä palvelevasta taustapalvelusta sekä ParkkiV-järjestelmän taustajärjestelmästä. Valintasivun käyttöliittymä on toteutettu React-kirjastolla ja TypeScript-ohjelmointikielellä. Valintasivun taustapalvelu on toteutettu Node.js-ympäristössä Express-kehyksellä. ParkkiV:n varsinainen taustajärjestelmä on toteutettu Python-ohjelmointikielellä. Kehitysympäristönä on käytetty Visual Studio Codea, ja valmis palvelu on asennettu Ubuntu-pohjaiselle virtuaalipalvelimelle, jota ajetaan UpCloud-alustalla.

5 TOIMINNALLINEN OSUUS

Tässä luvussa kuvataan, miten Entra ID-kirjautuminen suunniteltiin ja toteutettiin moniasiakasympäristöön. Luvussa käsitellään lähtötilanne, valitun ratkaisun suunnittelu, toteutuksen keskeiset vaiheet sekä valmis kirjautumisvirta käyttäjän näkökulmasta. Tarkastelun painopiste on niissä ratkaisuissa, joilla keskitetty Entra ID -tunnistautuminen liitettiin osaksi olemassa olevaa ParkkiV -järjestelmää. Lisäksi luvussa arvioidaan, miten toteutus vastasi lähtötilanteen ongelmiin ja millaisia teknisiä rajauksia tai jatkokehitystarpeita toteutukseen jäi.

5.1 Lähtötilanne

Ennen Entra ID -toteutusta ParkkiV käytti sovelluskohtaisia paikallisia tilejä. Käyttäjät tunnistettiin ParkkiV:n omasta käyttäjärekisteristä, johon kirjautumisessa syötetyt tunnukset verrattiin. Sovellukseen oli toteutettu monivaiheinen tunnistautuminen (MFA), mutta tämä oli vielä pilottivaiheessa. Paikallisten tilien mallissa sovellukselle jäi vastuu kirjautumiseen liittyvästä ylläpidosta, kuten käyttäjätilien elinkaaren hallinnasta, salasanapolitiikasta sekä kirjautumisen valvonnasta.

Lähtötilanteessa ilmeni lisäksi käytettävyys- ja hallintaongelmia moniasiakasympäristössä. Jos käyttäjän työtehtävät edellyttivät pääsyä useampaan kaupunkikohtaiseen ParkkiV-instanssiin, käyttäjän oli käytännössä kirjauduttava erikseen eri ympäristöihin ja siirryttävä niiden välillä eri osoitteiden kautta. Samanaikaisesti kaupunkien omia kirjautumiskäytäntöjä ja käyttäjähallintaa ei voitu hyödyntää, koska tunnistautuminen ja käyttöoikeuksien hallinta perustuivat ParkkiV:n sisäiseen käyttäjärekisteriin. Lähtötilan keskeiset ongelmat olivat siten paikallisiin tileihin sidottu kirjautuminen, moniasiakkuuden heikko käytettävyys sekä keskitetyn identiteetinhallinnan puute.

5.2 Suunnitelma

Suunnitelmassa tehtiin selkeä vastuunjako eri järjestelmäosien välillä. Tämän vuoksi Entra ID -konfiguraatio rakennettiin kahdella erillisellä sovellusrekisteröinnillä. Finn-ID:n Entra ID -ympäristöön rekisteröitiin erikseen ParkkiV-kirjautuminen ja ParkkiV-API, koska kirjautumisen käynnistämisen ja resurssipalvelun valtuutuksen erottaminen teki kokonaisuudesta selkeämmän, hallittavamman ja tietoturvan kannalta perustellumman.

ParkkiV-kirjautuminen (valintasivu) suunniteltiin valintasivun taustapalvelun asiakassovellukseksi. Sen tehtäväksi määritettiin kirjautumisvirrasta huolehtiminen sekä valintasivun istunnon muodostaminen. ParkkiV-API puolestaan suunniteltiin resurssisovellukseksi, jolle access token myönnetään. Tähän sovellusrekisteröintiin määritettiin rajapintaoikeus (Park.Access) sekä kaupunkikohtaiset sovellusroolit (kaupunki1.Access, kaupunki2.Access).

ParkkiV-kirjautumiselle luotiin clientId, valintasivun taustapalvelun käyttöön clientSecret sekä kirjautumisen jälkeen käytettävä paluuosoite (redirect URI), joka osoittaa taustapalvelun paluuosoitteeseen (callback). ParkkiV-API:lle puolestaan asetettiin Application ID URI, jota käytetään tokenin audience-tunnisteena.

Suunnitelmassa huomioitiin myös asiakkaan käyttöönotto. Koska sovellukset rekisteröitiin moniasiakkaina, asiakasorganisaation tenanttiin muodostuu käyttöönoton yhteydessä sovellusta vastaava yrityssovellus. Asiakkaan Entra-ylläpitäjä hyväksyy sovelluksen ensimmäisen kirjautumisen yhteydessä (admin consent), liittää käyttäjät ja käyttäjäryhmät sovellukseen sekä jakaa heille tarvittavat kaupunkikohtaiset roolit.

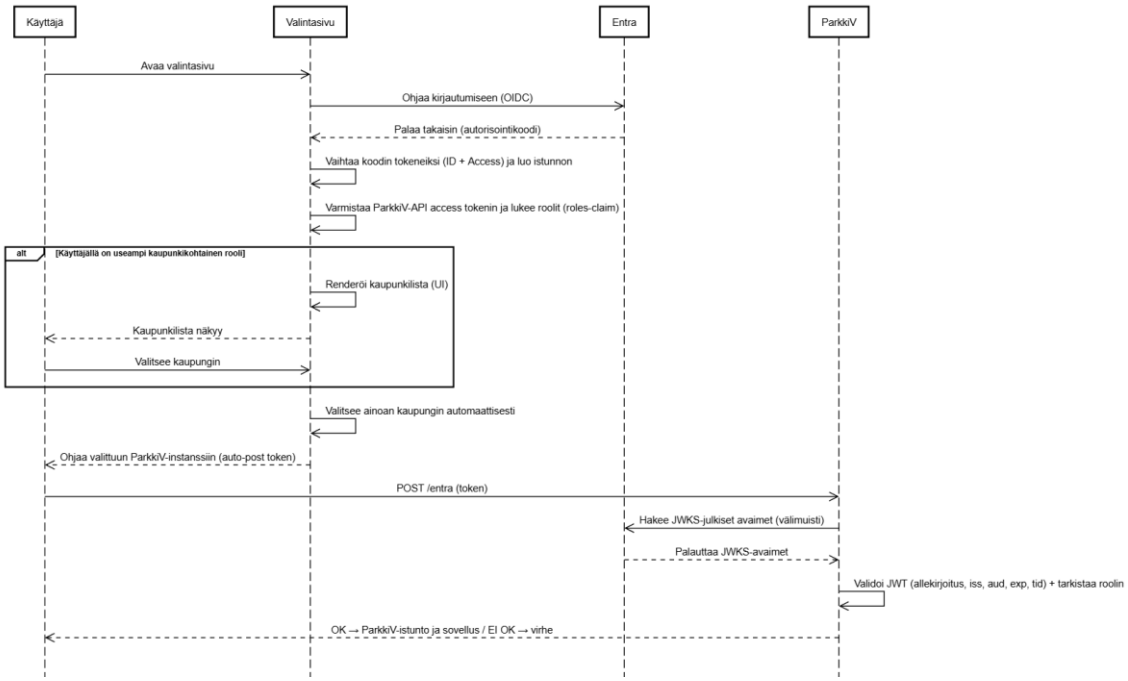
Suunnitelman toisessa vaiheessa päätettiin toteuttaa valintasivu ja sen taustapalvelu. Käyttöliittymä toteutettiin Reactilla ja TypeScriptillä, koska ne vastaavat edelleen web-pohjaisten sovelluksien nykystandardia. Käyttöliittymälle rakennettiin sitä palveleva Node.js/Express-taustapalvelu. Sen vastuulle määriteltiin kirjautumisvirran käsittely, tokenien hankinta sekä portaalin istunnonhallinta.

Taustapalvelu käynnistää kirjautumisvirran ja vastaanottaa Entra ID:n paluuosoitteesta autorisaatiokoodin. Koodi vaihdetaan tokeneiksi, minkä jälkeen käyttäjän kirjautumistila tallennetaan istuntoon. Tämän jälkeen taustapalvelu tarjoaa käyttöliittymälle rajapinnan, jonka avulla voidaan hakea käyttäjän roolien perusteella sallitut kaupungit. Tähän ratkaisuun päädyttiin, koska selainpuolella ei ollut tarkoituksenmukaista käsitellä tokenien hankintaa tai muita luottamuksellisia kirjautumisvaiheita. Node-taustapalvelu muodosti siten selkeän välikerroksen käyttöliittymän ja Entra ID:n välille.

Kolmannessa vaiheessa ParkkiV-taustapalveluun määritettiin kirjautumisrajapinta (/entra), joka vastaanottaa valintasivun välittämän tokenin selaimen lähettämällä POST-pyynnöllä. Tokeniksi valittiin ParkkiV-API:lle myönnetty access token, koska se on tarkoitettu nimenomaan resurssipalvelun (ParkkiV-taustapalvelu) tarkastettavaksi. ID token olisi soveltunut käyttäjän tunnistamiseen valintasivun puolella, mutta ParkkiV-taustapalvelun lopullinen pääsynvalvonta haluttiin perustaa resurssisovellukselle tarkoitettuun tokeniin. ParkkiV-taustapalvelun vastuulle määritettiin lopullinen validointi, koska kohdejärjestelmän tulee tehdä viimeinen päätös siitä, hyväksyykö se kirjautumisen omaan ympäristöönsä.

Valittu arkkitehtuuri oli toimiva ratkaisu ja se selkeytti järjestelmän eri osien vastuunjakoa. Kuitenkin se toi mukanaan omat kompromissinsa. Kahden erillisen sovellusrekisteröinnin malli teki kirjautumisen ja resurssinvalvonnan rooleista selkeämmät, mutta lisäsi Entra ID -konfiguraation määrää. Valintasivun taustapalvelu muodosti selkeän välikerroksen käyttöliittymän ja Entra ID:n välille, mutta lisäsi siten kokonaisuuteen yhden ylläpidettävän komponentin. Lisäksi access tokenin välittäminen HTML-lomakkeella POST-pyynnön kautta oli toimiva ratkaisu, mutta se edellytti erillisen välitysvaiheen rakentamista valintasivun ja ParkkiV-taustapalvelun välille.

Kuvio 2 kokoaa yhteen suunnitelman mukaisen arkkitehtuurin ja kirjautumisvirran päävaiheet. Sen avulla on helpompi seurata, miten valintasivu, Entra ID ja ParkkiV-instanssit osallistuvat kirjautumiseen eri vaiheissa.



KUVIO 2. Entra ID -ratkaisun lopullinen arkkitehtuurikuvaus

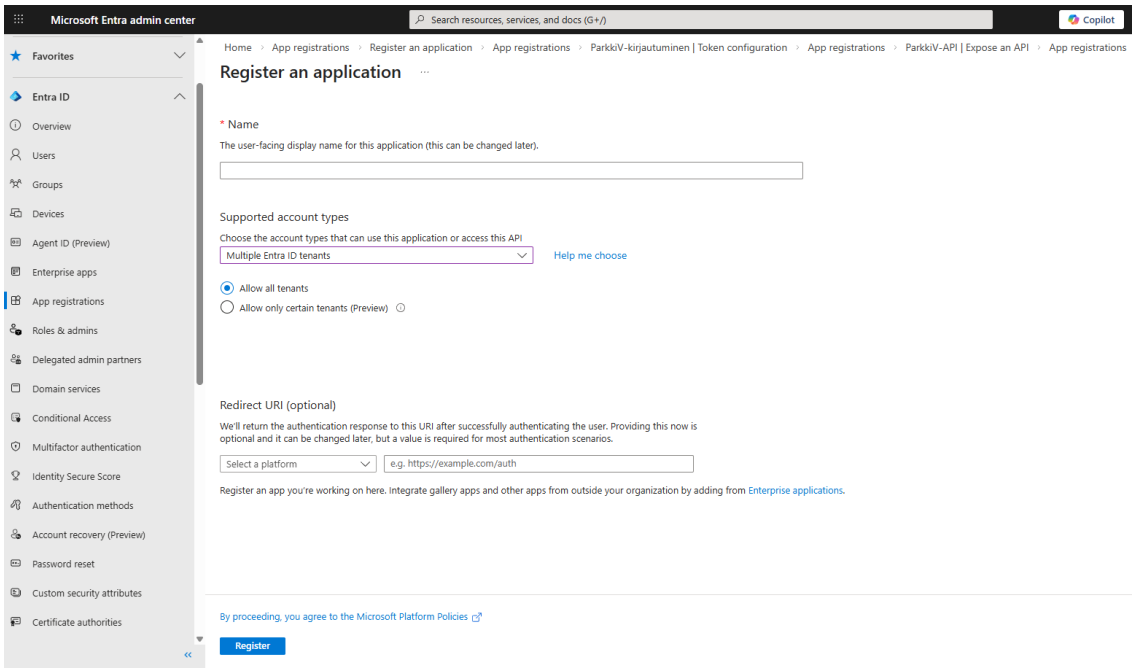
5.3 Toteutus

Tässä alaluvussa kuvataan, miten suunnitelmassa määritetyt ratkaisut toteutettiin käytännössä. Toteutus eteni selkeästi vaiheittain siten, että ensin perustettiin Entra ID:hen tarvittavat sovellukset ja määritettiin niiden konfiguraatiot. Tämän jälkeen toteutettiin valintasivu ja sitä palveleva taustapalvelu, minkä jälkeen ParkkiV:n taustapalveluun tehtiin tarvittavat muutokset tokenin vastaanottoa ja validointia varten. Lopuksi kirjautumisvirta kuvataan ajonaikaisena kokonaisuutena.

Tarkoituksena ei ole esittää koko lähdekoodia, vaan nostaa esiin toteutuksen kannalta keskeiset ratkaisut, rajapinnat ja kohdat. Painopiste on niissä muutoksissa, joilla Entra ID -kirjautumisratkaisu liitettiin osaksi ParkkiV-järjestelmää.

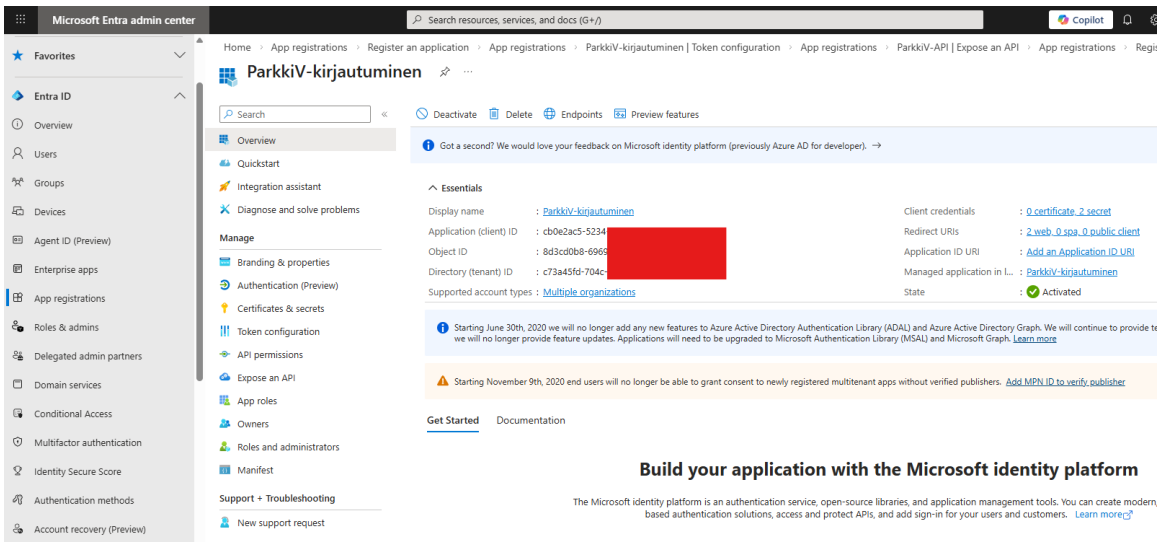
5.3.1 Entra ID -konfigurointi

Kuvassa rekisteröidään uusi sovellus Entra ID:hen ja valitaan Multiple Entra ID tenants -asetus. Moniasiakasvalinta mahdollistaa sen, että eri asiakasorganisaatiot voivat ottaa sovelluksen käyttöön omassa tenantissaan ilman erillistä sovellusrekisteröintiä. Sovellusrekisteröinti tehdään erikseen ParkkiV-kirjautumiselle ja ParkkiV-API:lle.



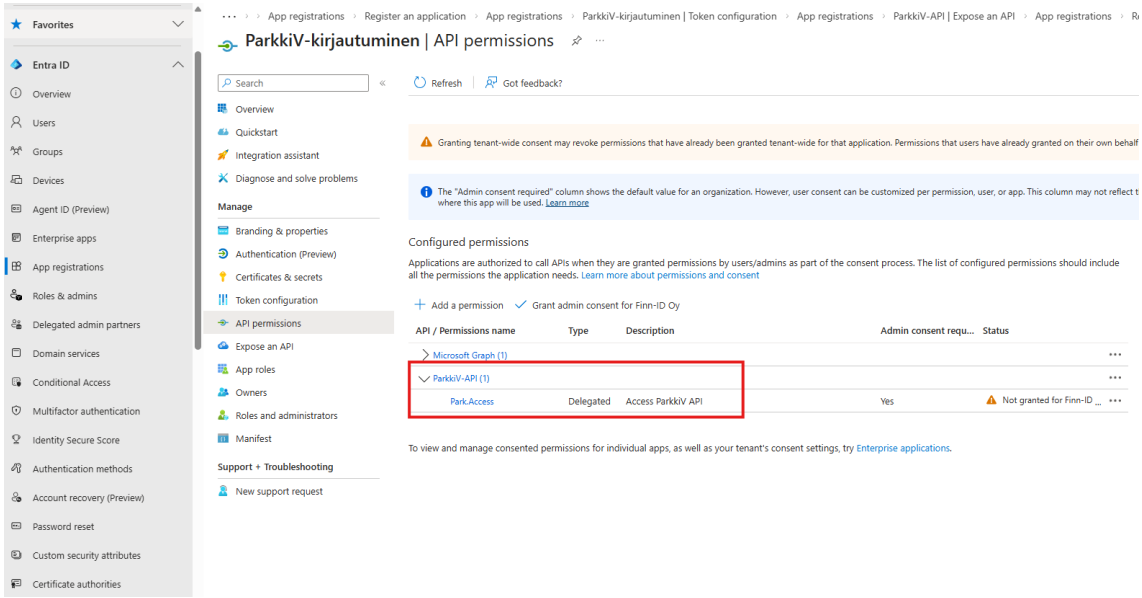
Kuva 1. Entra ID -sovellusrekisteröinti

Yleisnäkymässä näkyvät sovelluksen keskeiset tunnisteet, kuten Application (client) ID ja Directory (tenant) ID, joita valintasivun taustapalvelu käyttää MSAL-konfiguraatiossa. Tunnisteet määrittävät sovelluksen identiteetin Entra-ympäristössä, eivätkä ne yksinään mahdollista kirjautumista ilman erillisiä tunnistetietoja.



Kuva 2. Kirjautumissovelluksen yleisnäkymä

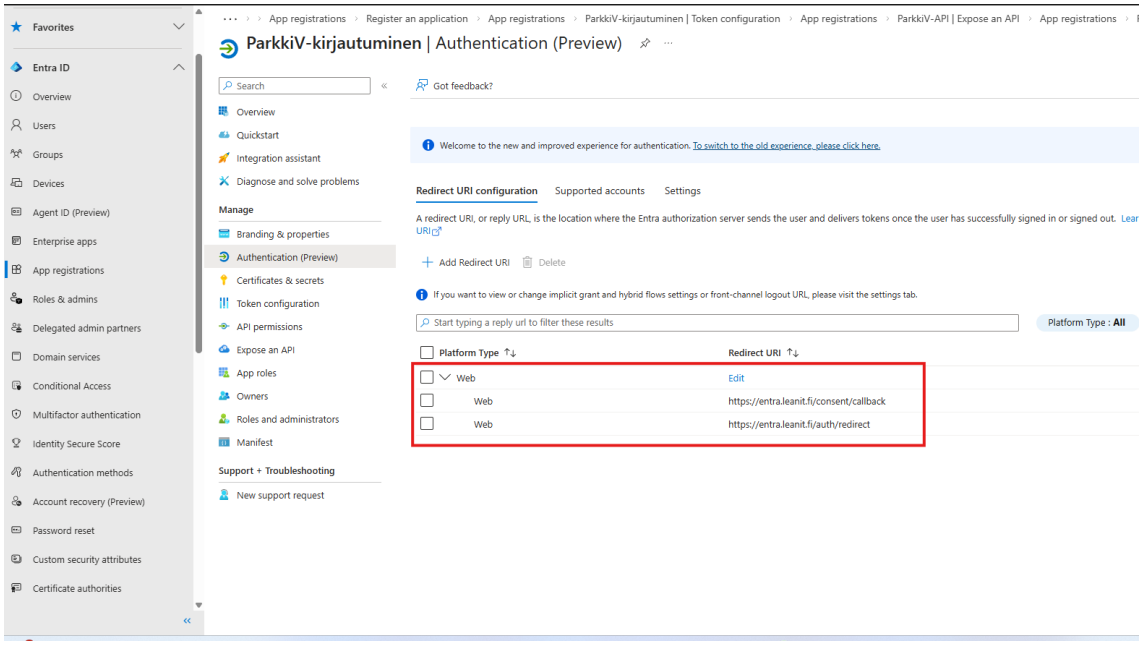
Kirjautumissovellukselle määritetään oikeus pyytää access token ParkkiV-API:n määrittelemälle rajapintaoikeudelle (Park.Access). Tämä on edellytys sille, että valintasivun taustapalvelu voi hankkia ParkkiV-API:lle tarkoitetun tokenin. Nyt tokenin kohdeyleisö (aud) osoittaa ParkkiV-API:in ja käyttöoikeustiedot voidaan välittää tokenin mukana.



Kuva 3. Kirjautumissovelluksen API-oikeudet

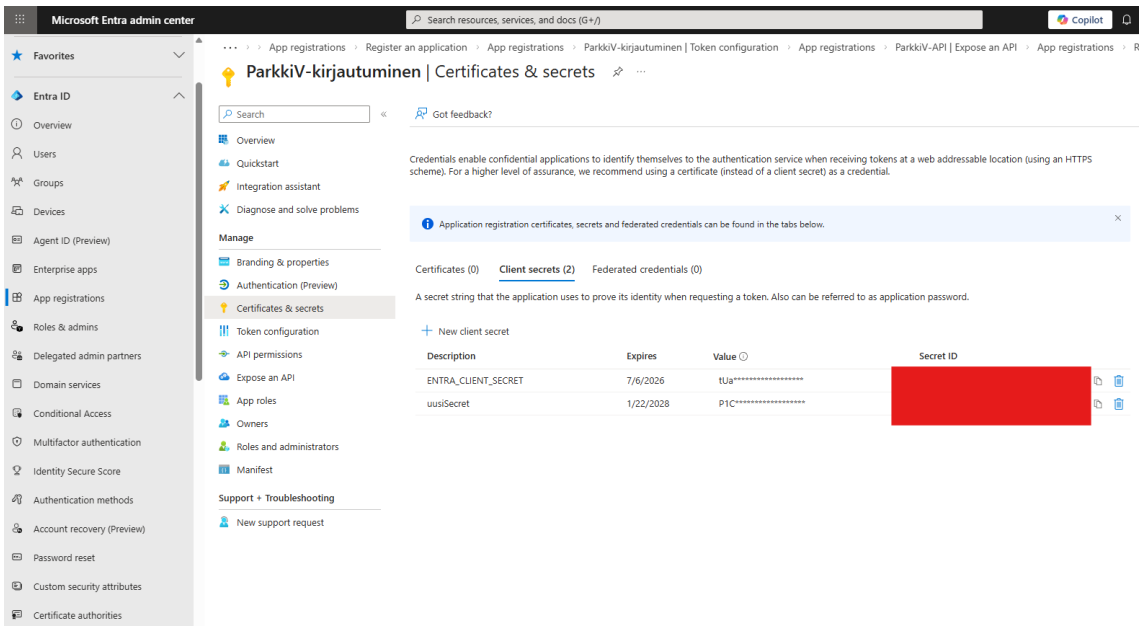
Seuraavaksi määritetään sovelluksen Redirect URI -osoitteet. Redirect URI määrittää sen, mihin Entra ID palauttaa selaimen kirjautumisen jälkeen, autori-

saatiokoodin kanssa. Tässä toteutuksessa /auth/redirect toimii varsinaisen kirjautumisvirran paluuosoitteena, kun taas /consent/callback toimii ylläpitäjän suostumuksen (admin consent) paluuosoitteena.



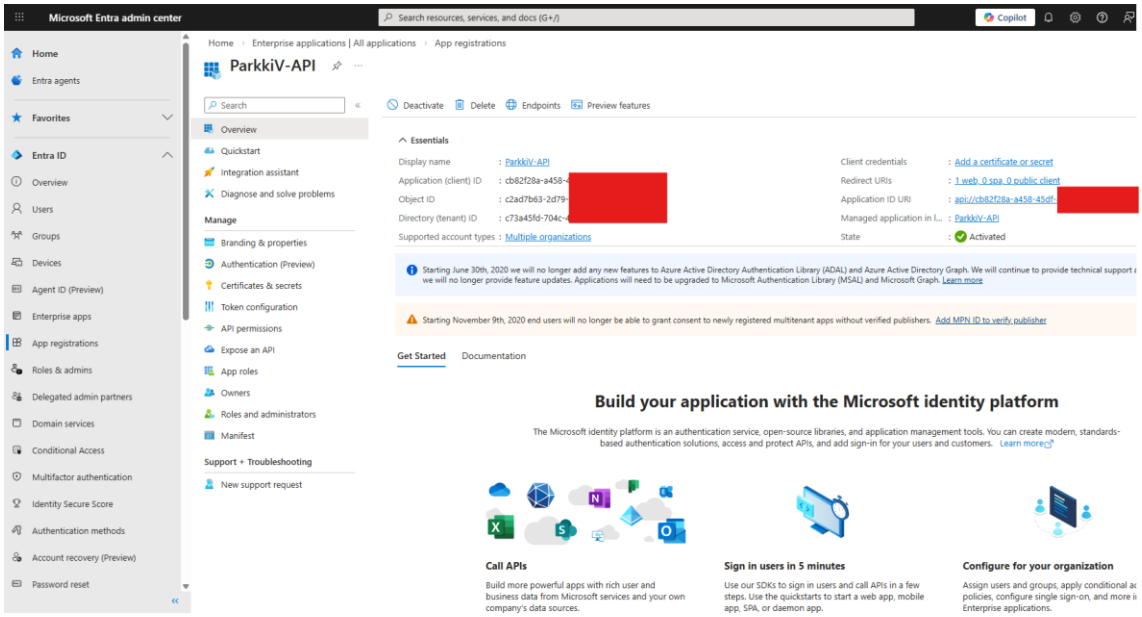
Kuva 4. Kirjautumissovelluksen redirect URI -määritykset

Sertifikaatti välilehdellä luodaan ParkkiV-kirjautuminen-sovellukselle asiakassalaisuus (client secret). Asiakassalaisuus toimii sovelluksen luottamuksellisena tunnisteena, jonka avulla taustapalvelu todentaa Entra ID:lle sovelluksen identiteetin. Sitä käytetään erityisesti silloin, kun taustapalvelu vaihtaa Entra ID:n palauttaman autorisaatiokoodin tokeneiksi ja hakee uusia tokeneita MSAL-kirjaston avulla.



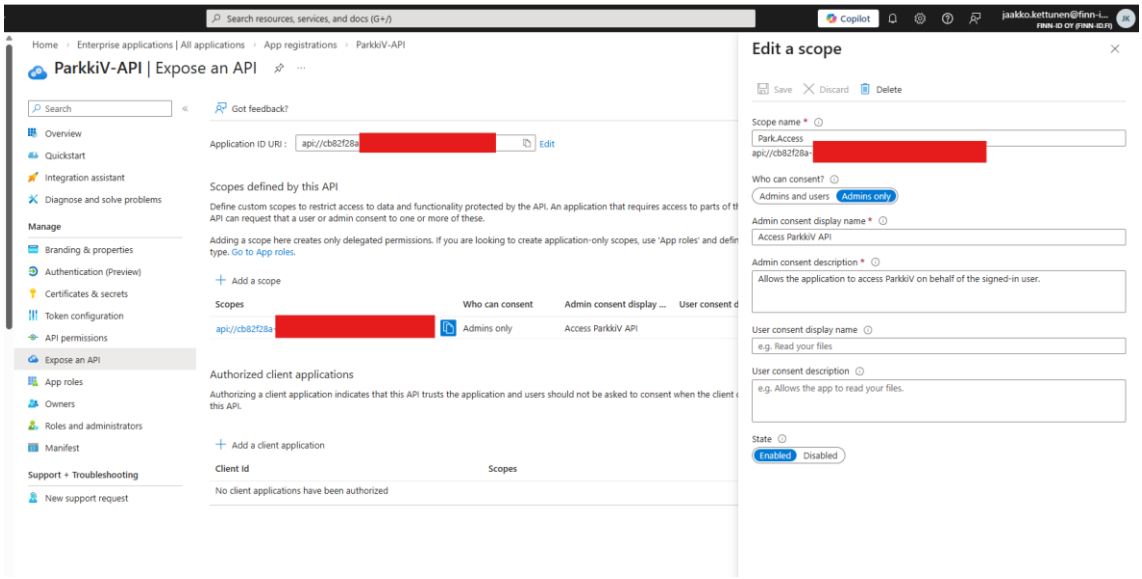
Kuva 5. Kirjautumissovelluksen asiakassalaisuus

Yleisnäkymässä näkyy ParkkiV-API:lle luotu sovellusrekisteröinti Entra ID:ssä. Sivulla esitetään sovelluksen perustiedot, kuten sovelluksen nimi, sovellustunniste ja tenant-tiedot. Tätä sovellusrekisteröintiä käytetään resurssisovelluksena, jolle määritetään rajapintaoikeus ja kaupunkikohtaiset sovellusroolit. Kirjautumissovellus pyytää Entra ID:ltä access tokenin nimenomaan tätä resurssia varten.



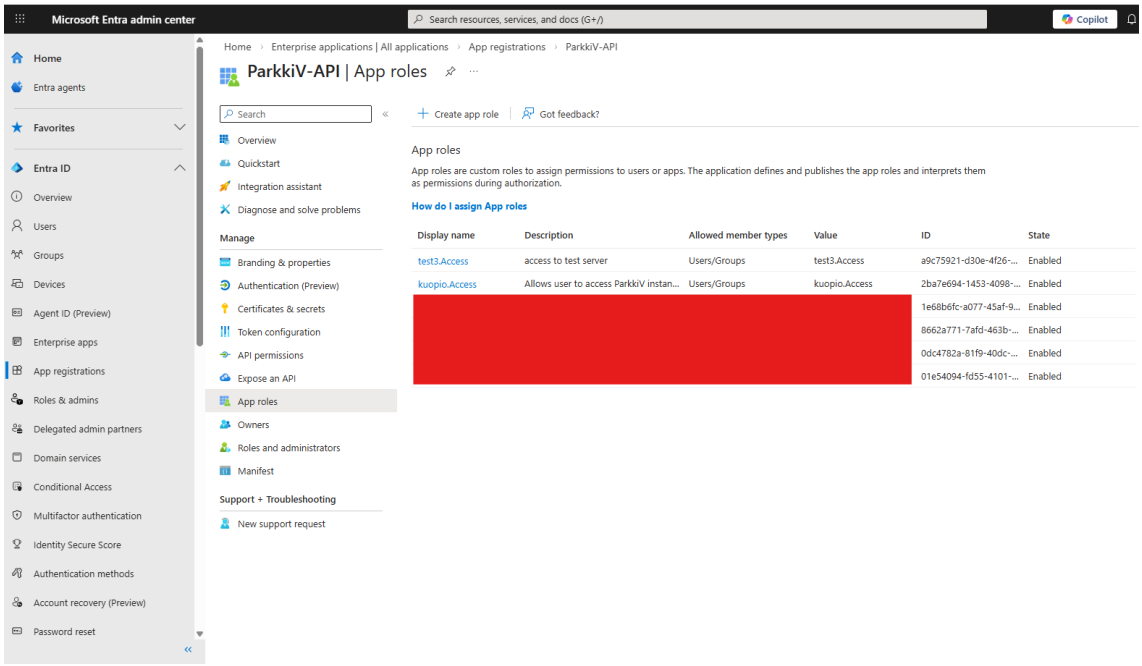
Kuva 6. ParkkiV-API:n yleisnäkymä

Seuraavaksi määritetään ParkkiV-API-sovellukselle Park.Access-rajapintaoikeus kohdassa Expose an API. Tämä määrittää, että kirjautumissovellus voi pyytää Entra ID:ltä access tokenin juuri tätä ParkkiV-API-resurssia varten. Käytännössä rajapintaoikeus toimii käyttöoikeuspyynnön kohteena silloin, kun valintasivun taustapalvelu hakee tokenia ParkkiV:n kirjautumisrajapintaan välitettäväksi.



Kuva 7. ParkkiV-API:n rajapintaoikeuden määrittely

Lopuksi määritetään ParkkiV-API-sovellukselle kaupunkikohtaiset sovellusroolit. Näillä rooleilla mallinnetaan käyttäjän käyttöoikeus eri ParkkiV-instansseihin. Asiakasorganisaation ylläpitäjä myöntää roolit käyttäjälle tai ryhmälle, jolloin ne välittyvät tokenin mukana roles-väitteenä. Valintasivun taustapalvelu muodostaa näiden väitteiden perusteella käyttäjälle näkyvän kaupunkilistan. Tämän jälkeen ParkkiV-taustapalvelu tekee instanssikohtaisen lopullisen tarkistuksen vertaamalla tokenin rooli- ja tenant-tietoja (roles, tid) kyseiselle kaupungille ennalta määritettyihin hyväksymiskriteereihin.



Kuva 8. ParkkiV-API:n sovellusroolit

Entra ID -konfiguroinnin tuloksena järjestelmään muodostui selkeä työnjako kahden sovellusrekisteröinnin välille. ParkkiV-kirjautuminen huolehtii käyttäjän kirjautumisvirrasta ja tokenien hankinnasta, kun taas ParkkiV-API toimii resurssisovelluksena, jolle access token myönnetään ja jonka rooleihin varsinainen käyttöoikeusmalli perustuu.

5.3.2 Valintasivu

Tavoitteena oli pitää valintasivun käyttöliittymä mahdollisimman kevyenä. Varsinainen kirjautumislogiikka ja tokenien käsittely jätettiin taustapalvelun vastuulle. Käyttöliittymän rooliksi jäi käyttäjän ohjaaminen oikeaan etenemispolkuun. Tämä vähensi selainpuolella käsiteltävien luottamuksellisten tietojen määrää.

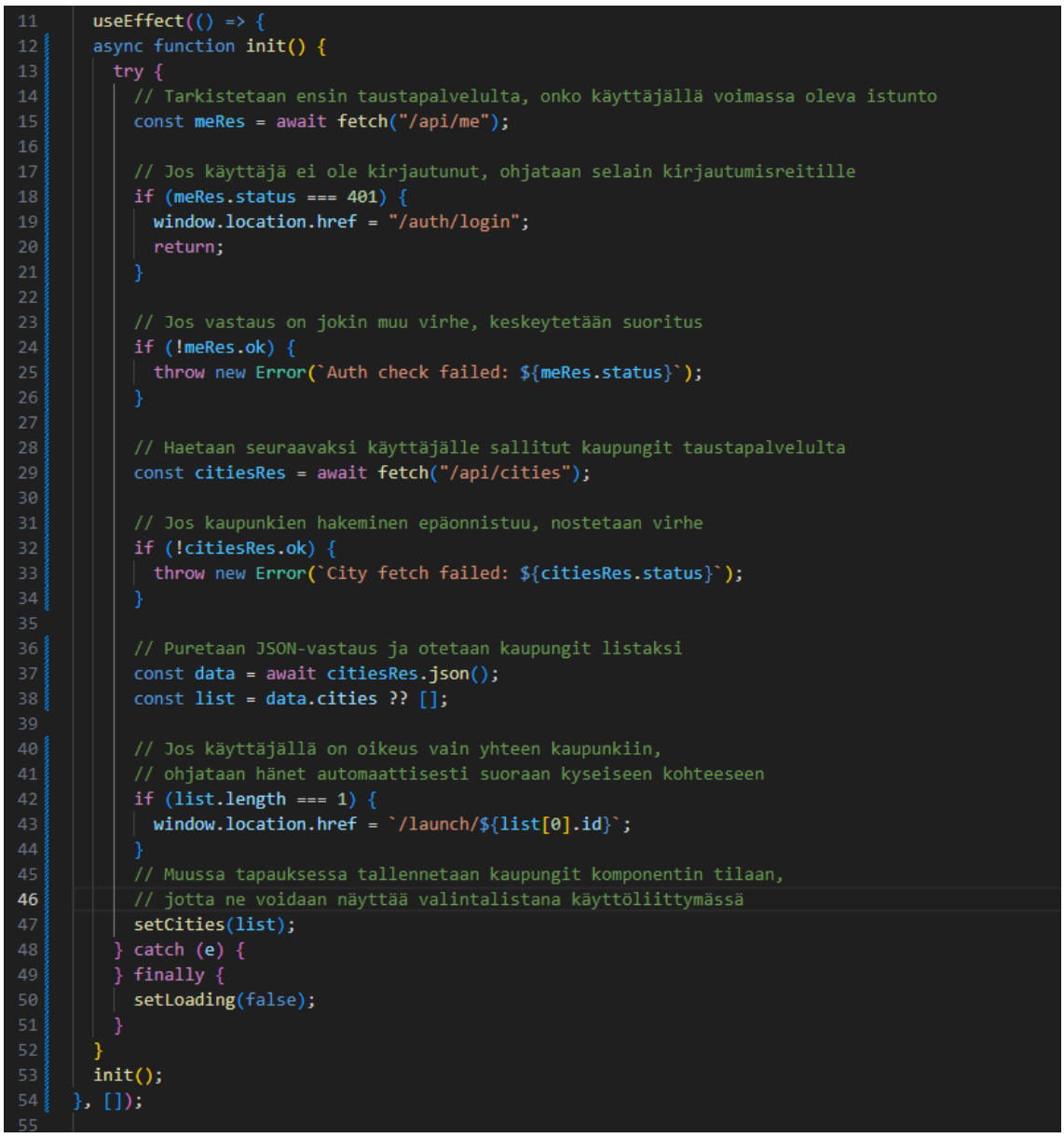
Käyttöliittymä näyttää käyttäjälle ParkkiV-logon, tervetulotekstin sekä listan käyttäjälle sallituista kaupungeista. Kuvan tilanteessa käyttäjällä on oikeus kahteen kaupunkiin, jolloin ne esitetään erillisinä painikkeina. Käyttäjältä edellytetään valintaa vain silloin, kun oikeuksia on useampaan kohteeseen. Muussa tapauksessa käyttöliittymä ohittaa valintavaiheen ja ohjaa käyttäjän automaattisesti eteenpäin.

Kuva 9. Valintasivun käyttöliittymä

Käyttöliittymä rakentuu yhden pääkomponentin ympärille, joka hakee tarvittavan tiedon taustapalvelulta ja reagoi saatuun vastaukseen. Toiminta käynnistyy heti sivun latauduttua, kun React-komponentin useEffect-koukussa suoritetaan alustava tarkistus. Ensin taustapalvelulta kysytään, onko käyttäjällä voimassa oleva kirjautumisistunto. Mikäli istuntoa ei ole, käyttäjä ohjataan automaattisesti kirjautumisreitille.

Kun istunto on olemassa, käyttöliittymä pyytää taustapalvelulta käyttäjän roolien perusteella sallitut kaupungit. Jos sallittuja kaupunkeja on vain yksi, käyttäjä ohjataan suoraan kyseiseen instanssiin. Useamman kaupungin tilanteessa käyttäjälle näytetään lista kaupungeista, joihin hän voi siirtyä.

Kuvassa esitetään käyttöliittymän keskeinen logiikka. Komponentti tekee ensin istunnon tarkistuksen taustapalvelulta ja hakee tämän jälkeen käyttäjän sallittujen kohteiden listan. Käytännössä tämä toteutetaan kutsumalla ensin /api/me-rajapintaa, jolla tarkistetaan istunnon olemassaolo, ja sen jälkeen /api/cities-rajapintaa, josta saadaan valintasivulla näytettävä kaupunkilista. Tilanteen mukaan komponentti ohjaa käyttäjän joko kirjautumisreitille (/auth/login) tai valitun ParkkiV-instanssin käynnistysreitille (/launch/{id}).



```
useEffect(() => {
async function init() {
  try {
    // Tarkistetaan ensin taustapalvelulta, onko käyttäjällä voimassa oleva istunto
    const meRes = await fetch("/api/me");

    // Jos käyttäjä ei ole kirjautunut, ohjataan selain kirjautumisreitille
    if (meRes.status === 401) {
      window.location.href = "/auth/login";
      return;
    }

    // Jos vastaus on jokin muu virhe, keskeytetään suoritus
    if (!meRes.ok) {
      throw new Error(`Auth check failed: ${meRes.status}`);
    }

    // Haetaan seuraavaksi käyttäjälle sallitut kaupungit taustapalvelulta
    const citiesRes = await fetch("/api/cities");

    // Jos kaupunkien hakeminen epäonnistuu, nostetaan virhe
    if (!citiesRes.ok) {
      throw new Error(`City fetch failed: ${citiesRes.status}`);
    }

    // Puretaan JSON-vastaus ja otetaan kaupungit listaksi
    const data = await citiesRes.json();
    const list = data.cities ?? [];

    // Jos käyttäjällä on oikeus vain yhteen kaupunkiin,
    // ohjataan hänet automaattisesti suoraan kyseiseen kohteeseen
    if (list.length === 1) {
      window.location.href = `/launch/${list[0].id}`;
    }
    // Muussa tapauksessa tallennetaan kaupungit komponentin tilaan,
    // jotta ne voidaan näyttää valintalistana käyttöliittymässä
    setCities(list);
  } catch (e) {
  } finally {
    setLoading(false);
  }
}
init();
}, []);
```

Kuva 10. Valintasivun pääkomponentin logiikka

5.3.3 Node-taustapalvelu

Valintasivun taustapalvelun tehtävänä on käsitellä Entra ID-kirjautumisvirtaa, ylläpitää käyttäjän istuntoa ja hakea käyttäjän oikeuksia vastaava access token. Lisäksi se tarjoaa käyttöliittymälle rajapinnan, jonka kautta haetaan sallittujen kaupunkien lista.

Entra ID:hen liittyvät asetukset luetaan ympäristömuuttujista ja käyttäjän istuntotiedot säilytetään palvelinpuolella. Istuntotietoja varten taustapalveluun mää-

ritettiin express-session-istunnonhallinta. Istuntoon tallennetaan käyttäjän tiliviite sekä MSAL-kirjaston token-välimuisti. Seuraavaksi esitellään taustapalvelun toiminnan kannalta keskeiset toteutusratkaisut.

Kuvassa esitetään MSAL-konfiguraatio. msalCachePlugin lukee token-välimuistin käyttäjän istunnosta ennen välimuistin käyttöä ja tallentaa sen takaisin istuntoon, jos välimuistin sisältö on muuttunut. msalClient(req) muodostaa Entra ID:hen yhdistetyn MSAL-”asiakkaan”, jolle annetaan clientId, clientSecret, authority ja istuntokohtainen välimuisti.

```
/**
 * plugin stores that cache into req.session so:
 * - tokens survive across requests
 * - each user gets their own cache
 */
//beforeCacheAccess: load cache from session into MSAL
//afterCacheAccess: save MSAL cache back into session if changed

function msalCachePlugin(req) {
  return {
    beforeCacheAccess: async (cacheContext) => {
      if (req.session.msalCache) {
        cacheContext.tokenCache.deserialize(req.session.msalCache);
      }
    },
    afterCacheAccess: async (cacheContext) => {
      if (cacheContext.cacheHasChanged) {
        req.session.msalCache = cacheContext.tokenCache.serialize();
      }
    },
  };
}
/**
 *MSAL "Confidential Client" for this request/session.
 * - clientId (Portal app registration)
 * - clientSecret (Portal app registration secret)
 * - authority (which Entra endpoint / tenant policy to use)
 * - token cache plugin (so it caches tokens in req.session)
 */
function msalClient(req) {
  return new ConfidentialClientApplication({
    auth: {
      clientId: process.env.ENTRA_CLIENT_ID,
      authority: process.env.ENTRA_AUTHORITY,
      clientSecret: process.env.ENTRA_CLIENT_SECRET,
    },
    cache: { cachePlugin: msalCachePlugin(req) },
  });
}
```

Kuva 11. MSAL-asiakkaan ja token-välimuistin istuntokohtainen määritys valintasivun taustapalvelussa

Kuvassa esitetään kirjautumisvirran kaksi keskeistä reittiä: /auth/login ja /auth/redirect. Ensimmäinen käynnistää kirjautumisen muodostamalla Entra ID:n kirjautumisosoitteen getAuthCodeUrl-kutsulla ja ohjaamalla selaimen kirjautumiseen. Jälkimmäinen vastaanottaa Entra ID:n palauttaman autorisaatiokoodin ja vaihtaa sen tokeneiksi acquireTokenByCode-kutsulla. Lopuksi käyttäjän account-tieto tallennetaan istuntoon ja käyttäjä ohjataan takaisin valintasivulle.

```
111  /**
112   * (1) Start login:
113   * React app sends user here to start Entra sign-in.
114   *
115   * getAuthCodeUrl builds the Entra authorization URL for OAuth/OIDC.
116   * User is redirected to Entra to authenticate.
117   */
118  app.get("/auth/login", async (req, res) => {
119    const cca = msalClient(req);
120
121    const authCodeUrl = await cca.getAuthCodeUrl({
122      redirectUri: process.env.ENTRA_REDIRECT_URI,
123      scopes: [process.env.PARK_API_SCOPE],
124    });
125
126    res.redirect(authCodeUrl);
127  });
128  /**
129   * (2) Callback endpoint:
130   * Entra sends user back here with ?code=...
131   *
132   * acquireTokenByCode exchanges the authorization code for tokens.
133   * store tokenResult.account in session, so later user can acquire tokens silently.
134   */
135  app.get("/auth/redirect", async (req, res) => {
136    const {code, error, error_description} = req.query;
137    if (error){
138      console.error(`Error received from Entra: ${error} - ${error_description}`);
139      return res
140        .status(500)
141        .type("html")
142        .send(`<!doctype html>···
204  </html>`);
205    }
206    if (!code) return res.status(400).send("Missing code");
207
208    const cca = msalClient(req);
209
210    const tokenResult = await cca.acquireTokenByCode({
211      code,
212      redirectUri: process.env.ENTRA_REDIRECT_URI,
213      scopes: [process.env.PARK_API_SCOPE],
214    });
215
216    // Marks the user as logged in (session-based)
217    req.session.account = tokenResult.account;
218    res.redirect("/");
219  });
```

Kuva 12. Kirjautumisvirran käynnistävä login-reitti ja autorisaatiokoodin käsittelevä callback-reitti

Kuvassa toteutetaan reitti /api/cities. Reitti hakee käyttäjän access tokenin ja lukee siitä tarvittavat väitteet. Tämän jälkeen tarkistetaan, että token on tarkoitettu ParkkiV-API:lle. Lopuksi käyttäjän rooleja verrataan ennalta luotuun cities-listaan, ja vastauksena palautetaan vain ne kaupungit, joihin käyttäjällä on oikeus.

```
238  /**
239   * (4) Returns list of cities the user can access.
240   *
241   * Steps:
242   * - get Park API access token from MSAL
243   * - decode token payload (to read roles claim)
244   * - verify token is intended for Park API (audience check)
245   * - filter cities by requiredRole
246   * - return allowed city list
247   */
248  app.get("/api/cities", requireAuth, async (req, res) => {
249    const token = await getAccessToken(req);
250    const payload = decodeJwtPayload(token);
251
252    // Sanity check: token must be for Park API (aud)
253    if (payload?.aud !== process.env.PARK_API_AUDIENCE) {
254      return res.status(500).json({ error: "Unexpected token audience" });
255    }
256
257    const roles = new Set((payload.roles || []).map(String));
258
259    const allowed = cities
260      .filter((c) => roles.has(c.requiredRole))
261      .map((c) => ({ id: c.id, name: c.name }));
262
263    res.json({ cities: allowed });
264  });
265
```

Kuva 13. Käyttäjän sallittujen kaupunkien muodostaminen access tokenin rooliväitteiden perusteella

Kaupunkeihin liittyvät tiedot koottiin erilliseen cities.js-tiedostoon. Tiedostossa määritetään kaupungin tunniste, käyttöliittymässä näytettävä nimi, vaadittu rooli ja kohdeinstanssin /entra-reitti. Keskitetty määritys helpotti sekä kaupunkilistan muodostamista että käyttäjän ohjaamista oikeaan ParkkiV-instanssiin.

Kuvassa toteutetaan reitti /launch/:cityId. Reitti tarkistaa ensin, että pyydetty kaupunki löytyy määrityksestä. Seuraavaksi haetaan käyttäjän access token ja varmistetaan, että tokenissa on kyseisen kaupungin vaatima rooli. Tämän jälkeen palvelu palauttaa HTML-lomakkeen, joka lähettää tokenin automaattisesti

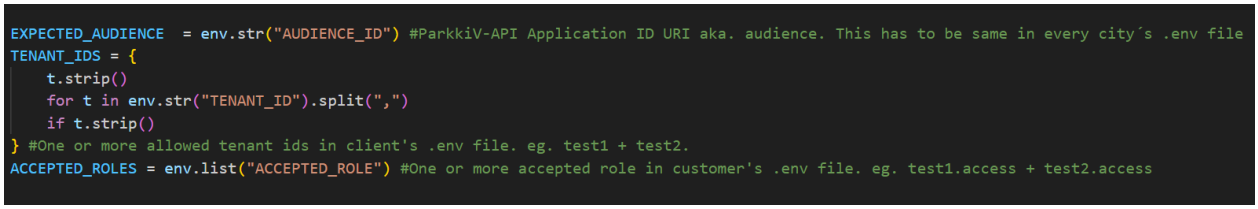
POST-pyynnöllä valitun ParkkiV-instanssin /entra-reitille. HTML-lomaketta käytetään, jotta token voidaan välittää pyynnön rungossa eikä URL-parametrina.

```
/**
 * (5) Launch city:
 * React app navigates here when user chooses a city.
 *
 * Steps:
 * - confirm city exists in whitelist config
 * - acquire Park API access token
 * - ensure user has the required city role
 * - return a tiny HTML page that auto-submits a POST to the city /entra endpoint
 *   including the access token in a hidden form field.
 *
 * Cache-Control: no-store prevents browser/proxies from storing this page because it contains a token.
 */
app.get("/launch/:cityId", requireAuth, async (req, res) => {
  const city = cities.find((c) => c.id === req.params.cityId);
  if (!city) return res.status(404).send("Unknown city");

  const token = await getAccessToken(req);
  const payload = decodeJwtPayload(token);
  const roles = new Set((payload.roles || []).map(String));

  if (!roles.has(city.requiredRole)) return res.status(403).send("Not allowed");

  res.setHeader("Cache-Control", "no-store");
  res.type("html").send(`<!doctype html>
<html>
  <body>
    <form id="f" method="post" action="${city.handoffUrl}">
      <input type="hidden" name="token" value="${escapeHtml(token)}" />
    </form>
    <script>document.getElementById('f').submit();</script>
  </body>
</html>`);
});

/**
 * Logs the user out of portal by destroying their session.
 */
app.post("/auth/logout", (req, res) => {
  req.session.destroy(() => res.status(204).end());
});
```

Kuva 14. Käyttäjän ohjaaminen valittuun ParkkiV-instanssiin ja access tokenin välittäminen POST-pyynnöllä

5.3.4 ParkkiV-taustapalvelu

ParkkiV-taustapalvelun tehtävänä on vastaanottaa valintasivulta välitetty token ja tehdä lopullinen kirjautumispäätös. Toteutuksessa lisättiin oma /entra-reitti tokenin vastaanottoa varten. Samalla rakennettiin toimintamalli, jossa token välitetään selaimen kautta palvelinpuolen tarkastukseen. Lopuksi token validoidaan Entra ID:n julkisilla avaimilla ja verrataan ParkkiV-instanssin vaatimiin käyttöoikeuksiin. Onnistuneen tarkastuksen jälkeen käyttäjälle muodostetaan sovellusistunto. Seuraavissa kuvissa esitetään toteutuksen keskeiset vaiheet.



Kuva 15. Access tokenin validoinnissa käytettävien ympäristömuuttujien määrittely

Kuvassa toteutetaan /entra reitti, joka vastaanottaa valintasivulta lähetetyn access tokenin. Reitti lukee lomakepostin rungosta tokenin ja purkaa siitä ilman lopullista luottamusta tenant-tunnisteen (tid). Tunnisteen avulla määritetään oikea tenant-kohtainen issuer-osoite ja JWKS-avainten hakupolku. Tämän jälkeen tarkistetaan, että tokenin tenant kuuluu kyseisen ParkkiV-instanssin sallittuihin tenantteihin.

Reitti tekee myös ensimmäisen tokenin tarkastuksen ennen tiedon siirtämistä seuraavaan vaiheeseen. Onnistuneessa tilanteessa token tallennetaan lyhytaikaisesti palvelinpuolen muistiin ja selaimeen asetetaan lyhytkestoiset evästeet. Tietoturvan vuoksi selaimen evästeeseen ei tallenneta varsinaista access tokenia, vaan satunnaisesti muodostettu arvo, jota käytetään lyhytkestoisena viitetunnisteena oikean tokenin hakemiseen välimuistista.

Lopuksi käyttäjä ohjataan sovelluksen kirjautumisnäkymään (/office2), jossa varsinainen kirjautumisen jatkokäsittely tehdään. Tähän ratkaisuun päädyttiin, koska ensimmäinen POST-pyyntö tulee suoraan selaimelta Django-taustapalveluun, eikä sitä voida käsitellä käyttöliittymän JavaScriptissä ennen kuin palvelin on ensin vastaanottanut sen.

```python
def entra_login(request):
    if request.content_type == 'application/x-www-form-urlencoded':
        data = parse_qs(request.body.decode('utf-8'))
        access_token_list = data.get('token', [])
        access_token = access_token_list[0] if access_token_list else None
    else:
        access_token = None
    if not access_token:
        return JsonResponse({'detail': 'Missing token'}, status=400)
    try: #this is only for getting the TID user is coming from
        #this tid is later compared to the list of allowed tids in the user's .env file
        unverified_decoded = jwt.decode(
            access_token,
            algorithms=['RS256'],
            options={
                "verify_signature": False,
                "verify_exp": False,
                "verify_aud": False,
                "verify_iss": False,
            },
        )
        token_tid = unverified_decoded.get("tid")
        if not token_tid:
            return JsonResponse({'detail': 'Missing tid claim'}, status=400)

        # check tenant is in the allowed list
        if not check_tenant(token_tid):
            return JsonResponse({'detail': 'Invalid tenant'}, status=400)

        #build tenant spesific issuer and jwks
        issuer = dynamic_issuer(token_tid)
        jwks_url = dynamic_jwks(token_tid)

        #signing key from tenant jwks
        jwk_client = jwt.PyJWKClient(jwks_url)
        signing_key = jwk_client.get_signing_key_from_jwt(access_token)
```

Kuva 16. /entra-reitin toteutus ja tokenin vastaanotto valintasivulta

Seuraavaksi luetaan selaimelta saadut väliaikaiset evästeet ja niiden perusteella haetaan aiemmin välimuistiin tallennettu access token. Palvelu lukee evästeistä viitetunnisteen (entra-ref), käyttäjän sähköpostitunnisteen (entra-email) sekä tenant-tunnisteen (entra-tid). Entra-ref ja cookie-tid arvoilla haetaan evästeavain, jolla saadaan access token käsittelyyn.

Palvelu tarkistaa, että välimuistista löytynyt tieto vastaa selaimen mukana tullutta viitetunnistetta. Jos tietoa ei löydy tai viitetunniste ei täsmää, kirjautuminen keskeytetään. Onnistuneen haun jälkeen välimuistista luetaan access token sekä siihen liittyvä tenant-tunniste.

Tokenista luetaan tenant-tunniste ilman allekirjoituksen tarkistusta. Tätä käytetään varmistaakseen, että tokenin tenant-tunniste vastaa välimuistiin tallennettua tenanttia ja että tenant-tunniste kuuluu kyseisen ParkkiV-instanssille sallittuihin tenantteihin.

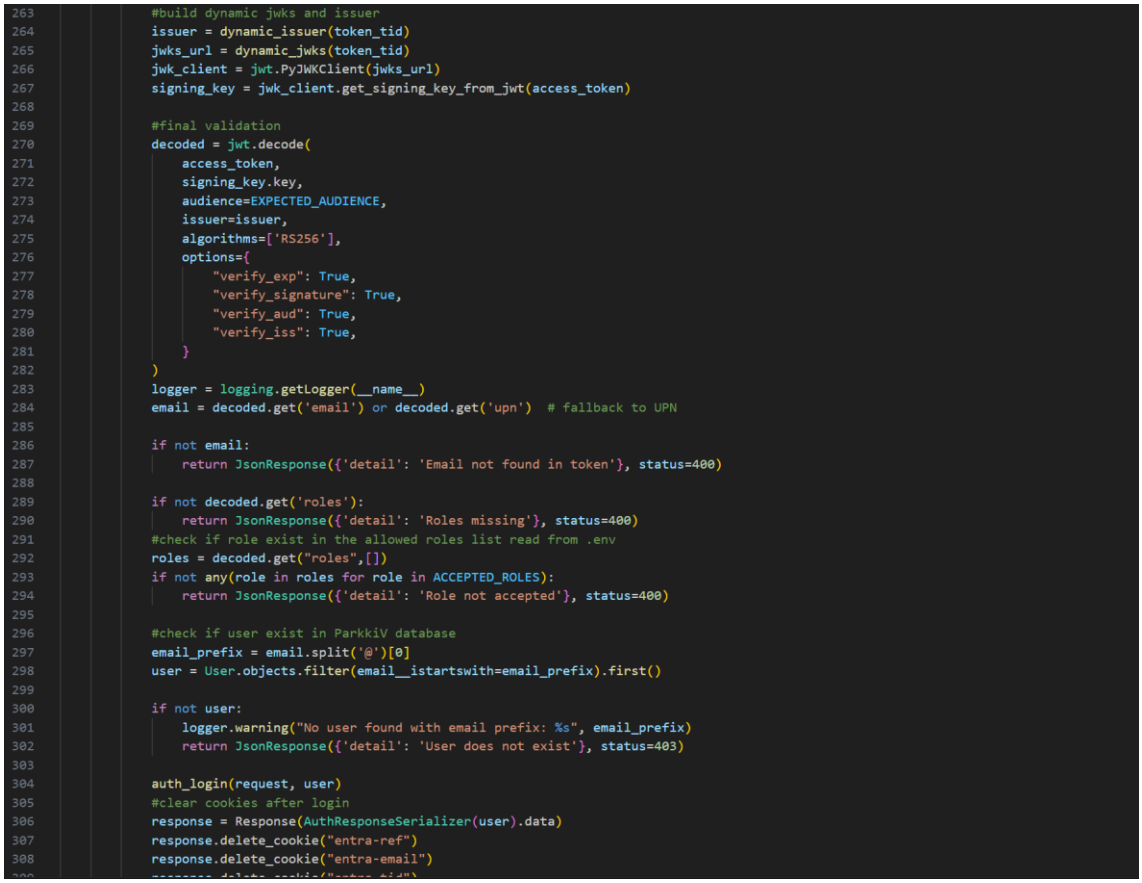
```python
def login(request):
    #get cookies
    entra_ref = request.COOKIES.get('entra-ref')
    entra_email = request.COOKIES.get('entra-email')
    cookie_tid = request.COOKIES.get('entra-tid')

    if not entra_ref and not entra_email and not cookie_tid:···
    else:
        if not entra_ref or not entra_email or not cookie_tid:
            return JsonResponse({'detail': 'Missing parameters'}, status=400)

        #get cache key
        cache_key = f"user_entra_token_{entra_email}_{cookie_tid}"
        #data from cache (tid, access_token)
        data = cache.get(cache_key)

        if not data or data.get("cache_ref") != entra_ref:
            return JsonResponse({'detail': 'Error fetching token from ParkkiV'}, status=400)

        cache_tid = data.get("tid")
        if not cache_tid:
            return JsonResponse({'detail': 'Missing tenant info'}, status=400)

        access_token = data.get('access_token')
        if not access_token:
            return JsonResponse({'detail': 'Missing token'}, status=400)
        try:
            unverified_decoded = jwt.decode(
                access_token,
                algorithms=['RS256'],
                options={
                    "verify_signature": False,
                    "verify_exp": False,
                    "verify_aud": False,
                    "verify_iss": False,
                },
            )
            token_tid = unverified_decoded.get("tid")

            if not token_tid:
                return JsonResponse({'detail': 'Missing tenant info'}, status=400)

            if token_tid != cache_tid:
                return JsonResponse({'detail': 'Tenant mismatch'}, status=400)
                    # check tenant is in the allowed list
            if not check_tenant(token_tid):
                return JsonResponse({'detail': 'Invalid tenant'}, status=400)
```

Kuva 17. Tenant-tunnisteen alustava lukeminen ilman allekirjoituksen tarkistusta

Lopuksi toteutetaan access tokenin lopullinen validointi. Palvelu muodostaa tokenin tenant-tunnisteen perusteella oikean issuer-arvon ja JWKS-osoitteen. JWKS-osoitteen avulla haetaan tokenin allekirjoituksen tarkistamiseen tarvittava julkinen avain. Access-token validoidaan kokonaisuudessaan tarkistamalla allekirjoitus, kohdeyleisö, myöntäjä ja voimassaoloaika.

Validoinnin jälkeen tarkistetaan, että token sisältää vähintään yhden ParkkiV-instanssille sallituista rooleista. Käyttäjä sidotaan ParkkiV:n omaan käyttäjärekisteriin etsimällä paikallinen käyttäjä saadun sähköpostitunnuksen perusteella.

Jos kaikki tarkistukset onnistuvat, kirjataan käyttäjä sisään ja muodostetaan sovellusistunto. Lopuksi kirjautumisprosessissa käytetyt väliaikaiset evästeet poistetaan selaimesta.



```
        #build dynamic jwks and issuer
        issuer = dynamic_issuer(token_tid)
        jwks_url = dynamic_jwks(token_tid)
        jwk_client = jwt.PyJWKClient(jwks_url)
        signing_key = jwk_client.get_signing_key_from_jwt(access_token)

        #final validation
        decoded = jwt.decode(
            access_token,
            signing_key.key,
            audience=EXPECTED_AUDIENCE,
            issuer=issuer,
            algorithms=['RS256'],
            options={
                "verify_exp": True,
                "verify_signature": True,
                "verify_aud": True,
                "verify_iss": True,
            }
        )
        logger = logging.getLogger(__name__)
        email = decoded.get('email') or decoded.get('upn')  # fallback to UPN

        if not email:
            return JsonResponse({'detail': 'Email not found in token'}, status=400)

        if not decoded.get('roles'):
            return JsonResponse({'detail': 'Roles missing'}, status=400)
        #check if role exist in the allowed roles list read from .env
        roles = decoded.get("roles",[])
        if not any(role in roles for role in ACCEPTED_ROLES):
            return JsonResponse({'detail': 'Role not accepted'}, status=400)

        #check if user exist in ParkkiV database
        email_prefix = email.split('@')[0]
        user = User.objects.filter(email__istartswith=email_prefix).first()

        if not user:
            logger.warning("No user found with email prefix: %s", email_prefix)
            return JsonResponse({'detail': 'User does not exist'}, status=403)

        auth_login(request, user)
        #clear cookies after login
        response = Response(AuthResponseSerializer(user).data)
        response.delete_cookie("entra-ref")
        response.delete_cookie("entra-email")
```

Kuva 18. Access tokenin lopullinen validointi ja sovellusistunnon muodostaminen

5.3.5 Ajonaikainen kirjautumisvirta

Kun käyttäjä avaa valintasivun osoitteen, selain lataa käyttöliittymän. Käyttöliittymä tarkistaa Node-taustapalvelulta, onko käyttäjällä voimassa oleva istunto. Jos istuntoa ei ole, käyttäjä ohjataan Entra ID -kirjautumiseen.

Onnistuneen kirjautumisen jälkeen Entra ID palauttaa käyttäjän takaisin valintasivun taustapalvelun paluuosoitteeseen autorisaatiokoodin kanssa. Node-taustapalvelu vaihtaa koodin tokeneiksi ja tallentaa käyttäjän kirjautumistilan istuntoon. Lopuksi taustapalvelu hakee access tokenin ParkkiV-API:lle.

Seuraavaksi taustapalvelu lukee tokenista käyttäjälle myönnetyt roolit ja muodostaa niiden perusteella listan sallituista kaupungeista. Jos sallittuja kaupunkeja on vain yksi, käyttäjä ohjataan automaattisesti suoraan kyseiseen instanssiin. Jos sallittuja kaupunkeja on useita, käyttäjä valitsee kaupungin valintasivulta. Node-taustapalvelu ohjaa selaimen valitun kaupungin launch-reitille. Launch-reitti tarkistaa vielä, että valittu kaupunki ja käyttäjän roolit vastaavat toisiaan.

Lopuksi Node-taustapalvelu palauttaa HTML-lomakkeen, joka lähettää access tokenin POST-pyynnöllä valitun ParkkiV-instanssin /entra-reitille. Tämä muodostaa varsinaisen siirtovaiheen valintasivulta ParkkiV-järjestelmään.

ParkkiV-taustapalvelu vastaanottaa tokenin ja hakee Entra ID:n julkisen avaimen JWKS-osoitteesta. Taustapalvelu validoi tokenin ja tarkastaa sen keskeiset väitteet. Jos token läpäisee validoinnin ja käyttöoikeustarkastuksen, ParkkiV liittää kirjautumisen paikalliseen käyttäjätiliin, muodostaa oman sovellusistunnon ja päästää käyttäjän sisään. Jos validointi epäonnistuu, kirjautuminen estetään ja käyttäjä palautetaan takaisin valintasivulle virheviestin kanssa.

Käyttäjän näkökulmasta kirjautuminen etenee yhtenäisenä virtana, vaikka taustalla siihen osallistuu useampi eri komponentti. Videoilla esitetään Entra ID -kirjautumisvirta käyttäjän näkökulmasta.

Entra ID -kirjautuminen alusta loppuun kahdella kaupungilla
Entra ID -kirjautuminen yhdellä kaupungilla
Entra ID -kirjautuminen yhdellä kaupungilla

5.4 Lopputulos

Lopputuloksena ParkkiV-järjestelmään toteutettiin toimiva Entra ID -kirjautumisratkaisu, joka vastaa määrittelyvaiheessa asetettuja keskeisiä tavoitteita. Ratkaisu mahdollistaa kirjautumisen asiakasorganisaation omilla Entra ID -tunnuksilla. Eri kaupunkien käyttöoikeudet voidaan rajata keskitetysti Entra ID:ssä

määritettyjen roolien perusteella. Usean kaupungin käyttöoikeusmalli mahdollistaa kirjautumisen useaan eri instanssiin saman valintasivun kautta.

Toteutuksen keskeiset osat ovat valintasivu ja sitä palveleva Node-taustapalvelu, sekä ParkkiV-järjestelmään integroitu /entra-reitti. Valintasivun taustapalvelu hoitaa kirjautumisvirran, hankkii tokenit ja pyytää ParkkiV:n taustapalvelua tekemään viimeisen tarkastuskierroksen tokenien väitteiden perusteella.

Työ parantaa käyttäjäkokemusta sekä ylläpidettävyyttä. Käyttäjä hyödyntää olemassa olevia tunnuksia, ja niiden hallinta on keskitetty asiakkaan omaan Entra ID ympäristöön. Tämä vähentää sovelluskohtaista käyttöoikeuksien hallintaa ja tietoturvakuorman määrää.

Työ dokumentoitiin vaaditulla tavalla ja se muodostaa selkeän perustan jatkokehitykselle. Ratkaisu on mahdollista laajentaa kaikille ParkkiV-asiakasorganisaatioille, ilman että kirjautumismallia tarvitsee suunnitella uudelleen jokaiselle ympäristölle.

Liiketoiminnallisesta näkökulmasta toteutus parantaa ParkkiV-järjestelmän kilpailukykyä. Entra ID -kirjautuminen vastaa asiakkaiden esittämiä vaatimuksia keskitetystä tunnistautumisesta ja tukee tarjouskilpailuissa huomioitavia laatutekijöitä. Ratkaisun merkitys konkretisoitui Lappeenrannan pysäköinninvalvonnan kilpailutuksessa, jossa Entra ID -kirjautuminen toi sovellukselle 10 lisälaatupistettä tarjousvertailussa.

6 POHDINTA

Työn aikana kohdattiin useita teknisiä ja käytännöllisiä haasteita. Yksi keskeisemmistä haasteista liittyi ParkkiV:n sisäiseen tokenin käsittely- ja validointirakenteeseen, joka muuttui projektin aikana muutamaan otteeseen. Lopullisessa toteutuksessa access tokenia käsitellään useassa eri vaiheessa ja siihen kohdistuu alustavia sekä lopullisia tarkastuksia yhteensä 4 kertaa. ParkkiV:n puoleisesta toteutuksesta tulikin alkuperäiseen suunnitelmaan verrattuna hieman toisteisempi ja raskaampi. Sinänsä ratkaisussa ei ole mitään vikaa ja useampi tarkistusvaihe lisääkin varmuutta siitä, että token on oikeasta tenantista, tarkoitettu oikealle resurssille ja se sisältää tarvittavat käyttöoikeudet.

Projektin loppuvaiheessa havaittiin ongelma, jota esiintyi Chrome-selaimella päivän ensimmäisen kirjautumisen yhteydessä. ParkkiV:n toimintamalli on rakennettu niin, että access tokenia siirretään palveluiden välillä väliaikaisesti palvelinpuolen välimuistin ja lyhytkestoisten evästeiden avulla. Alkuperäisessä toteutuksessa evästeiden ja välimuistin aikarajat olivat vain 3 sekuntia, mikä aiheutti tilanteen, jossa kirjautumisprosessi ei ehtinyt aina valmistua ennen väliaikaisen tiedon vanhenemista. Ongelmaa tutkittiin selaimen kehitystyökaluilla ja Network-näkymästä nähtiin, että päivän ensimmäisellä kirjautumiskerralla selain joutui lataamaan suuren määrän käyttöliittymäsovelluksen resursseja. Työkalun mukaan selain latasi noin 140 erillistä tiedostoa 4,3 megatavun sekuntinopeudella. Koko prosessin kesto oli noin 30 sekuntia ennen kuin sovellus oli täysin käyttövalmis.

Koska käyttöliittymä latautui päivän ensimmäisen kirjautumisen yhteydessä kohtuuttoman hitaasti, väliaikaiset evästeet ehtivät vanhentua ennen kuin sovellus ehti hyödyntää niitä kirjautumistapahtumassa. Tämän seurauksena automaattinen kirjautuminen ParkkiV-instanssiin epäonnistui. Jos käyttäjä valitsi saman instanssin uudelleen, kirjautuminen onnistui ilman viivettä, koska selain oli jo ladannut tarvittavat sovellusresurssit.

Ongelma korjattiin kahdella toimenpiteellä. Evästeiden ja välimuistin aikarajaa nostettiin 30 sekuntiin, jolloin selain ehtii käsitellä kirjautumiseen liittyvät vaiheet.

Lisäksi kirjautumisprosessiin lisättiin välikäsittelysivu, jonka tehtävänä on suorittaa palvelinpuolen kirjautuminen ennen varsinaisen käyttöliittymän lataamista. Tällä varmistettiin, että käyttäjän istunto on muodostettu jo ennen kuin selain alkaa ladata laajempaa käyttöliittymäsovellusta.

Ratkaisu paransi kirjautumisen luotettavuutta merkittävästi. Havainto korostaa, että myös tekniset pienet yksityiskohdat, kuten evästeiden aikarajat tai sovelluksen latausjärjestys, voivat vaikuttaa merkittävästi käyttäjäkokemukseen ja järjestelmän toimintavarmuuteen.

Uusi tunnistautumistapa otettiin ensimmäiseksi käyttöön Kuopion kaupungissa. Käyttöönotto toi esiin haasteita Entra ID -sovellusten hyväksymisprosessissa. Ongelmia aiheutti se, että kirjautumisportaali kutsui API-sovellusta, jolle asiakkaan ylläpito ei ollut vielä tenantissaan hyväksynyt tarvittavia käyttöoikeuksia. Käyttöönottojärjestyksen merkitys korostui, koska ensin oli annettava lupa API-sovellukselle ja vasta sen jälkeen valintasivu-sovellukselle. Tekninen toteutus ei yksin riitä, vaan onnistunut käyttöönotto vaatii selkeää viestintää ja ohjeistusta asiakkaan Entra-ylläpidon kanssa.

Entra ID -ratkaisun operatiiviset riskit jäivät mataliksi. Keskeisin riski liittyi siihen, että käyttäjälle ei ole myönnetty oikeaa sovellusroolia asiakkaan Entra ID -ympäristössä. Tällöin käyttäjä ei pääse kirjautumaan järjestelmään, eikä vääränlaista pääsyä synny. Vaikka asiakkaan Entra -ympäristössä näytetään kaikki käytettävissä olevat roolit, pelkkä rooli ei yksinään riitä pääsyyn väärään ParkkiV-instanssiin. Sisään tulevasta tokenista tarkistetaan tenant-tunniste, jota verrataan kyseiselle instanssille sallittuihin tenantteihin. Lisäksi tokenin mukana tullutta sähköpostitietoa verrataan ParkkiV:n omaan käyttäjärekisteriin. Näin ollen käyttöoikeus ei perustu pelkästään käyttäjälle annettuun rooliin vaan se muodostuu usean eri muuttujan yhteistuloksesta. Suunniteltu arkkitehtuurimalli vähentää väärinkäytön mahdollisuutta, vaikka asiakasorganisaation roolien hallinnassa tapahtuisi virheitä.

Työn alkuvaiheessa tehtiin tietoisia rajauksia ja kompromisseja. Vaihtoehtoisena toteutustapana olisi ollut rakentaa Entra ID -kirjautuminen ParkkiV-järjes-

telmän sisälle ilman erillistä valintasivua. Tähän vaihtoehtoon ei kuitenkaan päädytty, koska se olisi kasvattanut työn laajuutta huomattavasti ja tehnyt toteutuksesta monimutkaisemman, erityisesti usean kaupungin ympäristöjen hallinnan osalta. Valintasivu mahdollisti sen, että eri kaupunkien käyttöoikeudet voidaan keskittää yhteen kirjautumispisteeseen, vaikka taustalla olevat ParkkiV-instanssit sijaitsevat eri ympäristöissä. Valittu handoff-malli (tokenin siirto lomake-POST:lla ja väliaikaisen viitteen avulla) lisäsi toteutukseen uuden välivaiheen, mutta se mahdollisti tokenin siirtämisen pyynnön rungossa ilman URL-parametreja ja vähensi selainpuolelle jäävää vastuuta.

Toteutuksen toimivuus varmistettiin testaamalla keskeiset kirjautumisskenaariot sekä rajatapaukset. Kirjautuminen yhteen instanssiin, kirjautuminen useaan instanssiin roolien perusteella ja tilanne, jossa käyttäjällä ei ole rooleja. Testattiin myös tyypilliset virhetilanteet, kuten väärä audience, vanhentunut token tai roolien puuttuminen. Lisäksi varmistettiin, että kohdejärjestelmä estää kirjautumisen, jos käyttäjä ei löydy paikallisesta käyttäjärekisteristä tai jos tokenin tenant ei kuulu instanssille sallittuihin arvoihin. Jatkossa ratkaisua voidaan vahvistaa laajemmalla regressiotestauksella ja erillisillä tietoturvatarkastuksilla.

Tietoturvan näkökulmasta ratkaisu pienentää salasanoihin ja paikalliseen tunnistautumiseen liittyviä riskejä. Käyttäjän tunnistautuminen ulkoistetaan Entra ID:lle ja kohdejärjestelmä nojaa allekirjoitettuun ja validoituun access tokeniin. Tokenin siirtovaihe valintasivun ja ParkkiV:n välillä muodostavat toteutuksen tietoturvan kannalta herkimmän kohdan. Jatkokehitystä ajatellen onkin tärkeää varmistaa, että token ei päädy koskaan lokiin, väliaikaisen viitteen elinikä pidetään mahdollisimman lyhyenä ja evästeiden suojausasetukset ovat tiukimmat mahdolliset.

Tulosten näkökulmasta ratkaisu on laajennettavissa. Käyttöönotto uudessa kaupungissa edellyttää käytännössä tenant-tunnisteen selvittämistä, sovellusten hyväksymistä asiakkaan Entra-ympäristössä sekä tarvittavien sovellusroolien liittämistä käyttäjille tai käyttäjäryhmille. Ratkaisu on siten hyödynnettävissä ParkkiV:n nykyisillä sekä tulevilla asiakkailla. Käyttöönotto edellyttää kuitenkin selkeää ohjeistusta asiakkaan Entra-ylläpidolle, jotta sovellusten hyväksyntä, käyttäjien liittäminen ja roolien määritys tehdään oikeassa järjestyksessä.

Jatkokehityksen näkökulmasta ParkkiV:n sisäistä Entra ID -käsittelyä olisi mahdollista suoraviivaistaa. Nykyinen toteutus toimii, mutta se sisältää toistuvia välivaiheita, mikä tekee kokonaisuudesta raskaamman ja vaikeamman ylläpitää. Ratkaisua voisi yksinkertaistaa esimerkiksi vähentämällä väliaikaisen cookie- ja välimuistikäsittelyn määrää tai siirtymällä selkeämpään palvelinpuoliseen session handoff -malliin.

Kokonaisuutena työ osoitti, että Entra ID -kirjautuminen voidaan liittää ParkkiV-järjestelmään toimivasti moniasiakasympäristössä. Ratkaisu parantaa käyttäjäkokemusta ja ylläpidettävyyttä, koska asiakasorganisaatiot voivat hyödyntää omia tunnuksiaan ja hallita käyttöoikeuksia keskitetysti. Samalla se tukee liiketoiminnallisia tavoitteita, koska Entra ID -kirjautuminen vastaa tarjouskilpailuissa ja asiakastarpeissa nousseita vaatimuksia.

LÄHTEET

Auth0. n.d. *What is authorization? - Examples and definition.* Verkkosivu. Viitattu 3.2.2026. https://auth0.com/intro-to-iam/what-is-authorization

Descope. 2023. *Fed Auth 101: What Is Federated Authentication.* Verkkosivu. Viitattu 3.2.2026. https://www.descope.com/learn/post/federated-authentication

Diffie, W. & Hellman, M. E. 1976. *New Directions in Cryptography. IEEE Transactions on Information Theory* 22(6), 644-654. PDF-dokumentti. Viitattu 16.3.2026. https://ee.stanford.edu/~hellman/publications/24.pdf

Ferraiolo, D. F. & Kuhn, D. R. 1992. *Role-based access controls.* 15th NIST-NCSC National Computer Security Conference, 554-563. PDF-dokumentti. Viitattu 5.2.2026. https://csrc.nist.gov/files/pubs/conference/1992/10/13/role-based-access-controls/final/docs/ferraiolo-kuhn-92.pdf

Finn-ID Oy. 2026. *Finn-ID.* Verkkosivu. Viitattu 16.3.2026. https://www.finn-id.fi/

General Services Administration. 2021. *Enterprise Single Sign-On Playbook, s. 4.* Verkkoaineisto. Viitattu 16.3.2026. https://www.idmanagement.gov/playbooks/sso/

Grassi, P., Garcia, M. & Fenton, J. 2020. *Digital Identity Guidelines (SP 800-63).* Verkkosivu. Viitattu 17.2.2026. https://pages.nist.gov/800-63-3/

Hardt, D. 2012. *The OAuth 2.0 Authorization Framework (RFC 6749), s. 4-10, 22-30.* Verkkosivu. Viitattu 13.2.2026. https://datatracker.ietf.org/doc/html/rfc6749

Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A., Lang, A. & Cogdell, M. 2019. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations (NIST SP 800-162), s. 1-3.* Verkkosivu. Viitattu 5.2.2026. https://csrc.nist.gov/pubs/sp/800/162/upd2/final

Jones, M. 2015b. *JSON Web Key (JWK) (RFC 7517), s. 1-5.* Verkkosivu. Viitattu 13.2.2026. https://datatracker.ietf.org/doc/html/rfc7517

Jones, M., Bradley, J. & Sakimura, N. 2015. *JSON Web Token (JWT) (RFC 7519), s. 6-10.* Verkkosivu. Viitattu 17.2.2026. https://datatracker.ietf.org/doc/html/rfc7519

Jones, M., Bradley, J. & Sakimura, N. 2015a. *JSON Web Signature (JWS) (RFC 7515), s. 4-6.* Verkkosivu. Viitattu 13.2.2026. https://datatracker.ietf.org/doc/html/rfc7515

Microsoft. 2024a. *Application and service principal objects in Microsoft Entra ID.* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/identity-platform/app-objects-and-service-principals

Microsoft. 2024b. *How and why apps are added to Microsoft Entra ID.* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/identity-platform/how-applications-are-added

Microsoft. 2024c. *OpenID Connect.* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/identity-platform/v2-protocols-oidc

Microsoft. 2024d. *Overview of the Microsoft Authentication Library (MSAL).* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/identity-platform/msal-overview

Microsoft. 2025. *Cloud-first Identity Management: Guidance for IT Architects.* Verkkosivu. Viitattu 24.3.2026. https://learn.microsoft.com/en-us/entra/identity/hybrid/guidance-it-architects-source-of-authority

Microsoft. 2025a. *Identity providers for workforce tenants - Microsoft Entra External ID.* Verkkosivu. Viitattu 16.4.2025. https://learn.microsoft.com/en-us/entra/external-id/identity-providers

Microsoft. 2025b. *What is Microsoft Entra?* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/fundamentals/what-is-entra

Microsoft. 2025c. *Authentication flow support in MSAL.* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/identity-platform/msal-authentication-flows

Microsoft. 2025d. *Acquire token requests in MSAL Node.* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/msal/javascript/node/acquire-token-requests

Microsoft. 2025e. *Token caching in MSAL Node.* Verkkosivu. Viitattu 13.2.2026. https://learn.microsoft.com/en-us/entra/msal/javascript/node/caching

Microsoft. 2025f. *Deploying Active Directory Federation Services in Azure.* Verkkosivu. Viitattu 16.3.2026. https://learn.microsoft.com/en-us/windows-server/identity/ad-fs/deployment/how-to-connect-fed-azure-adfs

Microsoft. 2026. *What is identity and access management (IAM)?* Verkkosivu. Viitattu 3.2.2026. https://www.microsoft.com/en-us/security/business/security-101/what-is-identity-access-management-iam

Morris, R. & Thompson, K. 1979. *Password Security: A Case History. Communications of the ACM* 22(11), 594-597. PDF-dokumentti. Viitattu 16.3.2026. https://rist.tech.cornell.edu/6431papers/MorrisThompson1979.pdf

NIST. n.d.b. *Least privilege.* Verkkosivu. Viitattu 3.2.2026. https://csrc.nist.gov/glossary/term/least_privilege

OASIS. 2005. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, s. 2, 14-15, 29-31, 45-48.* Verkkosivu. Viitattu 17.2.2026. https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

OWASP. n.d. *Session Management Cheat Sheet.* Verkkosivu. Viitattu 13.2.2026. https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Rose, S., Borchert, O., Mitchell, S. & Connelly, S. 2020. *Zero Trust Architecture (NIST SP 800-207), s. 1-2.* Verkkosivu. Viitattu 3.2.2026. https://csrc.nist.gov/pubs/sp/800/207/final

Sakimura, N., Bradley, J., Jones, M. & Waite, R. 2023. *OpenID Connect Discovery 1.0 incorporating errata set 2.* Verkkosivu. Viitattu 13.2.2026. https://openid.net/specs/openid-connect-discovery-1_0.html

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. & Mortimore, C. 2014. *OpenID Connect Core 1.0 incorporating errata set 1, s. 3-17.* Verkkosivu. Viitattu 13.2.2026. https://openid.net/specs/openid-connect-core-1_0.html

Saltzer, J. H. 2021. *On the Origin of Kerberos. IEEE Annals of the History of Computing* 43(1), 89-91. PDF-dokumentti. Viitattu 16.3.2026. https://web.mit.edu/Saltzer/www/publications/Kerberosorigin.pdf

Stouffer, K., Pease, M., Tang, C., Zimmerman, T., Pillitteri, V., Lightman, S., Hahn, A., Saravia, S., Sherule, A. & Thompson, M. 2023. *Guide to Operational Technology (OT) Security (NIST SP 800-82r3), s. 98.* Verkkodokumentti. Viitattu 1.3.2026. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf

Temoshok, D., Fenton, J. L., Choong, Y.-Y., Lefkovitz, N., Regenscheid, A., Galluzzo, R. & Richer, J. P. 2025. *Digital Identity Guidelines — Authentication and Lifecycle Management (NIST SP 800-63B), s. 5-7, 16, 113.* Verkkosivu. Viitattu 5.2.2026. https://pages.nist.gov/800-63-4/sp800-63b.html

Temoshok, D., Fenton, J. L., Choong, Y.-Y., Lefkovitz, N., Regenscheid, A., Galluzzo, R. & Richer, J. P. 2025. *Digital Identity Guidelines: Authentication and Authenticator Management (SP 800-63B-4), s. 4.* Verkkosivu. Viitattu 13.2.2026. https://csrc.nist.gov/pubs/sp/800/63/b/4/final

Walden, D. & Van Vleck, T. 2011. *The Compatible Time-Sharing System (1961-1973): Fiftieth Anniversary Commemorative Overview.* PDF-dokumentti. Viitattu 16.3.2026. https://multicians.org/thvv/compatible-time-sharing-system.pdf