Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Ville Naukkarinen

PSD2-rajapintojen käyttö sovellus-kehittäjien näkökulmasta

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

26.1.2021

Tekijä Otsikko Sivumäärä Aika	Ville Naukkarinen PSD2-rajapintojen käyttö sovelluskehittäjien näkökulmasta 41 sivua 26.1.2021
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaaja	Lehtori Simo Silander

Insinöörityössä oli tavoitteena tutkia, mitä tietoa pankit jakavat vuonna 2018 voimaan tulleen uuden maksupalveludirektiivin eli PSD2-direktiivin edellyttämien rajapintojensa kautta, sekä selvittää, mitä rajapintojen käyttö teknisesti vaatii sovelluskehittäjältä. Työssä tutkittiin neljää Suomessa toimivaa pankkia, jotka olivat Handelsbanken, Nordea, Osuuspankki ja S-Pankki.

Työssä selvitettiin, mitä uusi maksupalveludirektiivi vaatii pankeilta, sekä perehdyttiin pankkien rajapintojen dokumentaatioon ja luotiin esimerkkisovellus, jossa rajapintoja käytettiin. Työssä perehdytään tarkasti, mitä erilaisia turvallisuuteen ja tunnistamiseen liittyviä teknologioita rajapintoja käyttäessä täytyy soveltaa, kuten digitaaliset varmenteet, oAuth 2.0 ja JSON Web Token.

Työssä havaittiin, että vaikka rajapintojen tarjoama tieto on hyvin samankaltaista, eroavat niiden käyttöönoton vaatimukset huomattavasti. Kaikki tutkitut pankit tarjoavat perustietoa pankkitileistä sekä mahdollisuuden käsitellä maksuliikennettä pankin kautta. Turvallisuusvaatimuksia on korkealla tasolla säädetty lainsäädännössä. Turvallisuuden kannalta tärkein asia on tunnistaa luotettavasti rajapintoja käyttävät sovellukset, sovelluksia käyttävät asiakkaat, sekä varmistaa tiedonsiirron luotettavuus kaikkien osapuolien osalta. Työssä havaittiin, että tunnistamiseen liittyvät teknologiat vaihtelevat paljon pankkien välillä ja niiden toteuttaminen on haastavaa.

Insinöörityössä saatiin selville, mitä teknologioita pankit käyttävät ulkopuolisten sovellusten tunnistamisessa. Sen tuloksena syntyi esimerkkisovellus, joka havainnollistaa vaadittavien teknologioiden toteutusta. Sovelluksessa hyödynnetään pankkien tarjoamia rajapintoja ja sovelluksen koodiesimerkkien avulla tarvittavia teknologioita käydään läpi.

Avainsanat	psd2, rajapinnat, pankkiala, oAuth, JWT

Author Title Number of Pages Date	Ville Naukkarinen PSD2 Interfaces from Software Developers' Perspective 41 pages 26 January 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Software Engineering
Instructor	Simo Silander, Senior Lecturer

The object of the study was to examine what information banks share through their application interfaces required by the new Payment Services Directive (PSD2) put into effect in 2018 and to find out what technical requirements the interfaces have when used by a software developer. Four different banks operating in Finland, Handelsbanken, Nordea, Osuuspankki and S-Pankki, were included in the study.

The study examined what the new Payment Services Directive requires from banks, how the application interfaces are documented. Furthermore, an example application was created where the interfaces were utilized. The study examined closely what different safety and authentication technologies must be implemented when utilizing the interfaces, e.g. digital certificates, oAuth 2.0 and JSON Web Token.

It was found that although the information provided by the interfaces is very similar, the requirements for starting to use them differ considerably. All the banks studied provide basic information about bank accounts and a possibility to process payment transactions through the bank. Security requirements are laid down in legislation on a high level and the most important factor of security is to reliably identify the third-party applications, users using the applications and to ensure the security of data transferred between all parties. It was found that the identification technologies vary greatly between banks and are challenging to implement.

The study identified what technologies banks use to identify external applications and resulted in an example application that illustrates the implementation of the required technologies. The application utilizes the interfaces offered by the banks and uses the code of the as an example when going through the necessary technologies.

Keywords	psd2, interface, banking, oAuth, JWT

Sisällys

Lyhenteet

AISP	Account Information Service Providers. Tilitietopalvelun tarjoajat.
eIDAS	Electronic identification, authentication and trust services. Sähköinen tunnistaminen, todennus ja luottamuspalvelut, EU-parlamentin antama asetus teknisistä vaatimuksista tunnistamisessa.
MTLS	Mutual Transport Layer Security. Kaksipuoleinen salausprotokolla, jolla voidaan varmistua molempien osapuolten identiteetistä verkossa.
PISP	Payment Initiation Service Providers. Maksutoimeksiantopavelun tarjoajat.
PKI	Public Key Infrastructure. Julkisen avaimen infrastruktuuri.
PSD2	Payment Services Directive 2. Toinen maksupalveludirektiivi.
QWAC	Qualified Website Authentication Certificate. Hyväksytty verkkosivuston todentamisvarmenne vaaditaan pankkirajapintojen käyttöön.
QSEAL	Qualified Certificate for Seals. Hyväksytty varmenne leimoille.
TLS	Transport Layer Security. Salausprotokolla internetin tietoliikenteeseen.
MTLS	Mutual Transport Layer Security. Kaksipuolinen salausprotokolla, jossa molemmat osapuolet tunnistavat toisensa.
TPP	Third Party Payment Service Provider. Kolmannen osapuolen maksupalveluntarjoajat.

1 Johdanto

Vuoden 2018 alussa voimaan tullut uudistettu maksupalveludirektiivi pakotti pankit avaamaan pääsyn kolmansille osapuolille palveluihinsa ohjelmallisten rajapintojen kautta. Tämä on avannut uusia mahdollisuuksia ulkopuolisille sovelluskehittäjille, jotka voivat nyt hyödyntää pankkien tarjoamaa tietoa ja kehittää niiden päälle omaa toiminnallisuutta. Ennen tätä sääntely ei ollut pysynyt teknologisen kehityksen perässä, koska syntyi uudenlaisia palveluita, joita entinen lainsäädäntö ei kattanut. Tällöin kuluttajien asema markkinoilla oli heikompi.

Insinöörityön tavoitteena oli tutkia, mitä tietoa pankit rajapintojensa kautta jakavat, mitä rajapintojen käyttö vaatii teknisellä tasolla sekä toteuttaa esimerkkisovellus, joka sisältää vaadittavat tunnistamisen mekanismit ja hakee pankeilta tarjolla olevaa tietoa. Tutkittaviin rajapintoihin sisältyi vain uuden maksupalveludirektiivin edellyttämät rajapinnat. Lisäksi työssä vertaillaan rajapintojen toteutusten ja dokumentoinnin eroavaisuuksia.

Työssä havaittiin, että vaikka tarjottavat palvelut ovat pintapuolisesti samankaltaisia, on niiden toteutuksissa, varsinkin tunnistautumisen osalta, isojakin eroja. Rajapintojen hyödyntäminen ei myöskään ole suoraviivaista, vaan edellyttää sovelluskehittäjältä erilaisten teknologioiden kattavaa osaamista. Tässä insinöörityössä käydään läpi, mitä tarjottavat palvelut ovat, minkälaisia turvallisuusvaatimuksia ne sisältävät ja miten sovelluskehittäjä pystyy ne toteuttamaan. Insinöörityön tarkoitus oli helpottaa muiden sovelluskehittäjien toimintaa rajapintojen hyödyntämisessä. Aluksi käsitellään itse rajapintoja ja sitä, mitä niillä voi tehdä. Sitten perehdytään rajapinnoissa käytettäviä tunnistautumiseen liittyviä teknologioita yleisellä tasolla. Lopuksi käydään läpi työssä syntynyt esimerkkisovellus, jossa teknologioita ja rajapintoja on sovellettu.

2 Saatavilla olevat rajapinnat

2.1 Tausta

Aikaisemmin Euroopan unionin jäsenvaltioissa maksupalvelumarkkinoita organisoitiin ja säädettiin kansallisesti. Tätä korjatakseen Euroopan komissio esitti vuonna 2005 ensimmäistä maksupalveludirektiiviä, joka yhdentäisi kansallista sääntelyä ja parantaisi kuluttajien ja uusien toimijoiden asemaa alalla. Direktiivi hyväksyttiin vuonna 2007. Nopean teknologisen kehityksen vuoksi alalle syntyi uudenlaisia toimijoita, joita direktiivin sääntely ei koskenut ollenkaan tai koski vain osittain. Näitä olivat esimerkiksi internetin välityksellä maksukortilta tehtävät maksut, joilla pystyi helposti maksamaan ostoksia. [1, s. 16.]

Näitä puutteita paikkaamaan syntyi Euroopan parlamentin direktiivi (EU) 2015/2366, joka tunnetaan paremmin nimillä toinen maksupalveludirektiivi tai PSD2-direktiivi. Se julkaistiin 23.12.2015. Direktiivin kansallinen täytäntöönpanonpäivä oli 13.1.2018. [2.] Sen tavoitteena on saattaa erilaiset maksupalvelut nykyistä laajemmin sääntelyn piiriin ja saattaa sääntely vastaamaan markkinoilla tapahtunutta kehitystä sekä avata kilpailua pankkitoimialalla [3, s.1].

Merkittävin uudistus pankeille on yhteistyövelvoite kolmansien osapuolten kanssa. Kolmannella osapuolilla tarkoitetaan rekisteröityneitä toimijoita rahoitusalalla, kuten maksuliikenteen välittäjiä verkkokaupan yhteydessä. Ensimmäistä kertaa pankkeja vaadittiin avaamaan pääsyä järjestelmiinsä muille toimijoille alalla. Myös palveluiden hinnoittelusta tuli osa sääntelyä. Merkittävää on, että kolmansilta osapuolilta ei ole sallittua veloittaa maksua uusien rajapintojen käytöstä.

Avattuihin palveluihin kuuluvat tilitiedot, varojen vahvistus ja maksutoimeksiannot. Jotta rajapintoihin voi saada pääsyn, pitää toimijan Suomessa hakea Finanssivalvonnalta Maksulaitosten ja sähköisen rahan liikkeeseenlaskijalaitosten toimilupa sekä rekisteröityä tilitietopalvelujen tarjoajaksi.

Lisäksi direktiivi vaatii asiakkaan vahvaa tunnistautumista, kun kyse on sähköisistä maksutapahtumista. Suomessa vahva tunnistautuminen käytännössä tarkoittaa

verkkopankkitunnusten, poliisin myöntämän sähköisen henkilökortin tai mobiilivarmenteen käyttämistä tunnistautumistilanteessa.

PSD2-rajapintojen toteuttaminen on pakollista kaikille Euroopassa toimiville pankeille eli jokaiselta Euroopassa toimivalta pankilta on löydettävissä jonkinlainen rajapinnan toteutus. Tässä insinöörityössä keskitytään vain Suomessa toimiviin pankkeihin ja niistäkin pieneen valikoituun joukkoon. Tutkittavia rajapintoja tarjoavat pankit ovat Handelsbanken, Nordea, Osuuspankki ja S-pankki. Handelsbanken, Nordea ja Osuuspankki tarjoavat rajapinnan ja dokumentaation omilla verkkosivustoillaan. S-Pankki käyttää Ahvenanmaalaisen Crosskey Banking Solutions -yhtiön tuottamaa Open Banking Market -nimistä rajapintapalvelua, jota käyttää myös useampi muu pienempi pankki kuten Resurs Bank ja Ålandsbanken [4]. Kaikki pankit tarjoavat PSD2-vaatimusten mukaiset rajapinnat REST-arkkitehtuurimalliin (Representational State Transfer) perustuen.

2.2 Tilitietopalvelut

Tilitietopalvelut sisältävät reaaliaikaista tietoa asiakkaan tilistä tai tileistä. Tilitietopalvelut sisältävät vain tietokyselyitä eivätkä sisällä muita aktiivisia toimenpiteitä. Tileistä annetaan perustietoja, kuten tilinumero, tilin nimi ja tilin saldo. Lisäksi tietopyyntöön voidaan sisällyttää tilin maksutapahtumat. Tilitietopalveluista on hyötyä esimerkiksi taloushallinto- ja kirjanpitosovelluksia tuottaville yrityksille. Sovelluksia käyttävillä yrityksillä on tarve tietää reaaliaikaisesti ja tarkasti kulloinkin tilillä oleva saldo maksaessaan laskuja tai palkkoja. Aikaisemmin tähän on voinut käyttää korkeintaan maksullisia rajapintoja, tai niitä ei ole ollut tarjolla ollenkaan. Lisäksi näitä voivat hyödyntää muut pankit. Esimerkiksi Osuuspankki on jo osaltaan hyödyntänyt tätä julkista rajapintaa muilta pankeilta ja julkaissut asiakkailleen mahdollisuuden nähdä tilien saldot ja tilitapahtumat muiden pankkien ylläpitämiltä tileiltä [5].

Kaikilla tutkituilla pankeilla tilitietorajapinta noudatti samaa mallia, jossa rajapintakutsut rajoittuvat tunnistautuneen asiakkaan kaikkien tilien listaamiseen, yksittäisen tilin tilitiedot ja tiliin liitetyt tilitapahtumat taulukossa 1 näkyvällä tavalla. Kaikki toimivat GET HTTP -pyynnöillä. Taulukosta on jätetty pois kunkin pankin yksilöivät verkko-osoitteet.

Taulukko 1. Pankkitilit-rajapinnan yleiskuvaus

	Pyyntöme-todi	Osuuspankki Nordea Handelsbanken	S-Pankki
Listaus tileistä	GET	/accounts	/accounts
Tilitiedot	GET	/accounts/{accountId}	/accounts/{AccountId}/balances
Tilitapahtumat	GET	/accounts/{AccountId}/ transactions	/accounts/{AccountId} /transactions

Tilitietorajapinnat antavat hyvin samankaltaista tietoa jokaisella pankilla. Yksittäisestä tilistä kerrotaan tilin omistajan nimi, tilin nimi ja käyttötarkoitus (esimerkiksi käyttö- tai säästötili), tilin saldo, mahdolliset käyttökatevaraukset ja käytetty valuutta. Yksittäisestä tilitapahtumasta kerrotaan tapahtuman summa, valuutta, kirjaamis- ja maksupäivä, maksaja, vastaanottajan nimi ja tilinumero, viesti ja viite.

Kaikki pankit tarjosivat tilitietojen osalta staattisia, muuttumattomia tietoja, joihin ei voinut itse vaikuttaa, eli jokaisella pyynnöllä tuli aina samat vastaukset. Nordea tarjosi lisäksi mahdollisuuden luoda ja poistaa tilejä, sekä luoda tapahtumia dynaamisesti rajapinnan testaamista varten. Oikeille asiakkaille tilin luontia ja poistoa ei tarjota, koska se on osa pankin omaa liiketoimintaa, eikä PSD2-direktiivi edellytä tällaisen mahdollisuuden tarjoamista.

2.3 Varojen vahvistus

Varojen vahvistus tarkoittaa, että asiakkaan tilille tehdään asiakkaan luvalla kysely, onko tilillä riittävästi varoja kyselyssä määritellylle summalle. Rajapinta osaltaan edistää kolmansien osapuolten maksukorttien tarjoamista, joissa maksu otetaan asiakkaan pankin ylläpitämältä tililtä [6, s. 16]. Rajapinnan avulla palveluntarjoaja pystyy tarkistamaan etukäteen, voidaanko maksu suorittaa.

Nordeaa lukuun ottamatta kaikki pankit ovat toteuttaneet samankaltaisen rajapinnan tätä varten taulukossa 2 näkyvällä tavalla. Kun asiakkaalta on varmistettu suostumus normaalin tunnistautumismenetelmän avulla, palauttaa rajapinta yksinkertaisen totuusarvon varojen riittävyydestä.

Nordealla tätä varten ei ole erikseen rajapintaa, mutta vastaava toiminnallisuus olisi mahdollista suorittaa erillisen tilitietorajapinnan kautta. Asiakkaan kannalta huonompi puoli tässä ratkaisussa on se, että asiakkaan pitää antaa lupa kaikkiin tilin perustietoihin, jotka sisältävät muutakin kuin yksinkertaisen tiedon siitä, onko varoja vai ei. Kortin tietoihin pääsyn antamalla palveluntarjoajaa saa tietää tarkalleen, kuinka paljon varoja tilillä ja katevarausten määrän. Myös varojen riittävyyden laskeminen jää palveluntarjoajan tehtäväksi.

Taulukko 2. Varojen vahvistus -rajapinnan yleiskuvaus

	Pyyntömetodi	Nordea	Osuuspankki Nordea Handelsbanken
Saatavilla olevien varojen tarkistus	POST	-	/funds-confirmations

2.4 Maksutoimeksiantopalvelut

Maksutoimeksiantopalveluilla tarkoitetaan tililtä toiselle tilille tehtäviä maksuja. Rajapinnat antavat mahdollisuuden tehdä rahansiirtoja Euroopan yhtenäisellä euromaksualueella (englanniksi Single Euro Payment Area, SEPA) SEPA-maksuina sekä ulkomaan maksuina, Euron lisäksi myös muita valuuttoja voi käyttää. Rajapinta mahdollistaa maksun suorittamisen välittömän vahvistamisen, esimerkiksi verkkokaupassa, mikä tarjoaa vaihtoehdon perinteisimmille korttimaksuille [7]. Rajapinta mahdollistaa esimerkiksi maksukorttipalveluiden tarjoamisen toisen pankin ylläpitämältä tililtä. Yritysten taloudenhallintosovelluksissa tämä tarjoaisi ilmaisen rajapinnan maksujen tekemiseen, mistä olisi hyötyä laskujen rahaliikenteen toteuttamisessa.

Pankkien omat rajapinnat olivat hyvin samankaltaisia tässäkin tapauksessa, kuten taulukosta 3 näkee. S-Pankin käyttämä Crosskeyn rajapinta kuitenkin eroaa hieman sanastoltaan ja käyttää termiä international payments.

Taulukko 3. Maksukortit-rajapinnan yleiskuvaukset

	Pyyntö-metodi	Nordea	Osuuspankki	Handelsbanken	S-Pankki
Listaus luoduista mak-suista	GET	`/payments/sepa`	/sepa-payments/{paymentId}/submissions	-	-
Maksun tiedot	GET	`/payments/sepa/{paymentId}`	/sepa-payments/{paymentId}	/payments/{paymentProduct}/{paymentId}/status	/international-payments/{InternationalPaymentId}
Uuden maksun luomi-nen	POST	`/payments/sepa`	/sepa-payments	payments/{paymentproduct}	/international-payments
Maksun poista-minen (ennen toimeen panoa)	DELETE	/payments/sepa/{paymentId}	/sepa-payments/{paymentId}	/payments/{paymentProduct}/{paymentId}/cancellation-authorizations/{cancellationId}	-

2.5 Rajapintojen dokumentaatio

Vertailussa mukana olleiden pankkien dokumentaatio oli rajapintakuvausten osalta hel-posti ymmärrettävä ja sisältää riittävästi tietoa kyseisestä rajapinnasta. Mikäli kehittäjällä on kokemusta rajapintojen käyttämisestä, ei pankkien rajapintojen dokumentaatiossa ole mitään erityisen yllättävää, vaan ne sellaisenaan toimivat kuin mikä tahansa muukin REST-rajapinta. Erityismaininnan ansaitsee Handelsbankenin dokumentaatio, joka tar-peellisten osoitteiden ja parametrien lisäksi tarjoaa valmiin koodiesimerkin HTTP-pyyn-nöstä lukuisille eri ohjelmointikielille. Tämä on ainakin aloittelevan kehittäjän näkökul-masta erinomainen asia, sillä erilaisten otsaketietojen ja parametrien HTTP-pyyntöön asettaminen tapahtuu eri ohjelmointikielillä eri tavalla.

Ongelmallisempi osuus pankkien dokumentaatiossa on rajapintojen suoraa hyödyntä-mistä edeltävä vaihe, eli rajapintaa vasten kehitettävien sovellusten tunnistautuminen ja miten se tapahtuu testiympäristössä. Tunnistautumisvaiheen toteutustapa myös

vaihtelee suuresti, mikä tarkoittaa, että ennen rajapintojen hyödyntämistä jokaiselle pankille joutuu tehdä erilaisen ja työlään vaiheen, eikä yhdestä toteutuksesta ole välttämättä mitään hyötyä toisen pankin kanssa. Yleisesti voi todeta, että Nordean, Osuuspankin ja Handelsbankenin ohjeet tähän vaikuttivat tyydyttäviltä ja niiden avulla pitäisi pystyä vaihe toteuttamaan. Handelsbankenin rekisteröitymisvaiheen dokumentaatio vaikutti hyvin selkeältä ja tunnistautumisvaihe yksinkertaisimmalta, mutta sitä ei ollut mahdollista kokeilla testiympäristössä. S-pankin, eli käytännössä Crosskeyn, dokumentaatiossa tässä oli kummallinen puute, ja dokumentaatio viittasi OpenId Connectin määrityksiin, joka on OAuth 2.0 -standardin päälle rakennettu määrittelykerros, jonka on tarkoitus helpottaa OAuth-toteutuksia sovelluksissa. On mahdollista, että tarkempia ohjeita saa, kun on siirtymässä oikean tiedon käyttäjäksi.

Tunnistautumisvaiheeseen testiympäristössä interaktiivisimman toteutuksen on tehnyt Osuuspankki. Se tarjoaa kehittäjille Github-koodivarastossa Java-pohjaisen sovelluksen, jolla sovelluskehittäjät voivat rekisteröityä testiympäristöön kolmannen osapuolen maksupalvelun tuottajaksi. Ohjelman avulla kehittäjälle generoidaan ja rekisteröidään palveluun kaksi tarvittavaa varmennetta, yksi tiedon digitaalista allekirjoittamista varten ja toinen tiedonsiirron salaamiseksi. Tuotantovarmenteiden (oikean palvelun varmenteiden) rekisteröimiseksi tämäkin vaihe pitää jokaisen suorittaa itse, mutta tarjottu ohjelma toimii samalla esimerkkinä omaan toteutukseen tätä varten.

Lähes kaikki pankit tarjosivat helpotettua polkua itse rajapintojen testaamiseen, jossa tunnistautumisvaiheen pystyi ohittamaan ja siten keskittymään varsinaisten rajapintojen testaamiseen. S-pankin dokumentaatiossa ohituspolkua ei mainittu.

Kaikki pankit tarjosivat testiympäristöjä, joissa oli valmiiksi testidataa. Vain osa käytettävistä rajapinnoista oli sellaisia, että testikäyttäjä pystyi muuttamaan testiympäristön tietoja. Kaikki pankit tarjosivat samoina pysyviä testiasiakkaita, joilla oli määrätty määrä tilejä. Kuitenkin maksupalvelurajapinta oli kaikilla toteutettu siten, että uuden maksun tekemistä ja sen hyväksyntää pystyi testaamaan dynaamisesti. Syntyneet tiedot pankit joko tyhjensivät määräajoin itse tai antoivat käyttäjälle mahdollisuuden palvelun hallintasovelluksen kautta palauttaa alkutilaan.

3 Rajapintojen turvallisuusmääritykset

3.1 Rajapintojen käytön esivaatimukset

Kun kyseessä on rajapinta pankkipalveluihin, täytyy kyetä luotettavasti tunnistamaan sen käyttäjät sen kaikissa merkityksissä. Käyttäjiä ovat kolmannen osapuolen sovellukset, niiden kehittäjät sekä loppukäyttäjät eli pankin asiakkaat. Ymmärrettävästi tämä johtaa siihen, että turvallisuusmääritykset ovat varsin laajat. Asiakkaiden käyttäessä pankkipalveluita verkossa pankit vastaavat itse käyttäjien tunnistamisesta. Usein tämä on toteutettu käyttäjätunnusten, salasanan ja vaihtelevan tunnuskoodin avulla. Kun pankkien asiakkaat käyttävät kolmannen osapuolen sovelluksia, käyttävät he samoja tunnistamistekniikoita kuin pankkien kanssa suoraan asioidessaan. Tässä insinöörityössä keskitytään ensisijaisesti siihen, miten pankit voivat luotettavasti tunnistaa nämä kolmansien osapuolten sovellukset.

Ennen kuin rajapintoja pystyy hyödyntämään oikeiden kuluttajien, eli pankkeja käyttävien asiakkaiden kanssa, on kehittäjän huolehdittava turvallisuudesta usealla eri tasolla. PSD2-rajapintojen käyttö edellyttää rekisteröintiä ja hyväksyntää eurooppalaiselta pankkiviranomaiselta. Suomessa PSD2-rajapintojen käyttäjien täytyy rekisteröityä maksupalvelujen tarjoajaksi Finanssivalvonnalle, rahoitus- ja vakuutusvalvontaviranomaiselle. Toiminta saattaa edellyttää myös maksulaitoksen toimilupaa, kun ehdot esimerkiksi siirrettävän rahan määrästä täyttyvät. Tässä insinöörityössä tämä jätetään huomioimatta, koska pankkien tarjoamia testiympäristöjä pystyy käyttämään kuka tahansa pankin palveluun kehittäjäksi rekisteröitynyt.

Kaikki palvelut vaativat kehittäjäksi rekisteröitymistä ja kehitettävän sovelluksen rekisteröinnin, ennen kuin rajapintoja voi käyttää. Rekisteröinnissä palveluntarjoaja (pankki) antaa sovellukselle rajapinta-avaimen, jolla rajapinta yksilöi sitä käyttävät sovellukset ja osassa palveluita annetaan myös niin sanottu salaisuus, ”secret”, jota pankki käyttää osana sovelluksen tunnistamista. Rajapintoja hyödyntävälle sovellukselle ja palvelua tarjoavalle pankille keskeisintä on vaatimukset siitä, miten pankki pystyy luotettavasti tunnistamaan kolmansien osapuolten sovellukset. Tämä sovelluksen tunnistautuminen perustuu julkisen avaimen infrastruktuuriin (PKI, Public Key Infrastructure).

PKI on viitekehys turvalliseen tiedonsiirtoon eri osapuolten ohjelmistojen välillä, mikä perustuu julkisen avaimen salaukseen ja X.509-standardin digitaalisiin varmenteisiin. [8, luku 13.] Korkealla tasolla kyse on luottamusverkon rakentamisesta, jossa luotettu taho luovuttaa toiselle osapuolelle varmenteen, jonka avulla varmenteen luovuttaja voi myöhemmin varmistua toisen osapuolen identiteetistä.

3.2 Maksupalveluiden turvallisuussääntely

Maksupalvelumarkkinoiden sääntelyssä haluttiin varmistaa myös maksuliikenteen turvallisuus. Euroopan parlamentti antoi 23.7.2014 niin kutsutun eIDAS-asetuksen [9], joka määrittelee tekniset vaatimukset sähköiselle tunnistamiselle ja sähköisiin transaktioihin liittyviin luottamuspalveluihin. Vaatimukset kattavat koko tunnistusjärjestelmän ketjun palveluntarjoajalta käyttäjän asiointirajapintaan asti, mikä koskee muun muassa tietoliikenneprotokollan turvallisuutta ja käytettäviä salausalgoritmeja. Tähän kuuluvat määrittelyt kahdesta varmenteesta eri tarkoituksiin, yksi toisen osapuolen tunnistamiseen ja toinen tiedonsiirtoa varten. Varmenteita kutsutaan määrittelyssä nimillä hyväksytty tunnistautumisvarmenne verkkosivustoille eli Qualified Website Authentication Certificate (QWAC) ja hyväksytty varmenne leimoille eli Qualified Electronic Seal Certificate (QSEAL). Nämä varmenteet ovat aiemmin mainitun X.509-standardin mukaisia digitaalisia varmenteita ja ne sisältävät julkisen sekä yksityisen avaimen. Varmenteita myöntävät hyväksytyt luottamuspalvelun tarjoajat. Suomessa tällainen on Digi- ja väestötietovirasto [10].

QWAC mahdollistaa tiedonvaihdon käyttäen Transport Layer Security -salausprotokollaa (TLS), jolla voidaan varmistaa luottamuksellisuus ja siirrettävän tiedon autenttisuus kahden osapuolen välillä. TLS varmistaa sen, että tiedonsiirron aikana kukaan ulkopuolinen ei ole voinut nähdä sisältöä tiedonsiirron aikana, eikä tietoa ole muutettu. TLS ei kuitenkaan vaikuta tiedonsuojaamiseen sen jälkeen, kun tiedosto on siirretty osapuolten välillä. [11, s. 5.]

QSEAL-varmennetta käytetään sovelluskerroksessa siirrettävän tiedon allekirjoittamiseen, mikä tarkoittaa, että sillä voidaan varmistaa siirrettävän tiedon luotettavuus. Vastaanottava taho voi olla varma, että tiedon allekirjoittanut on varmenteen omistaja ja että tieto ei ole muuttunut allekirjoituksen jälkeen. Tieto myös pysyy suojattuna, kunnes

vastaanottava taho purkaa suojauksen. Nämä määritetyt varmenteet luovat pohjan, jonka päälle on kehitetty erilaisia teknisiä ratkaisuja. Seuraavaksi käydään läpi tutkittavilla rajapinnoilla käytössä olevia tekniikoita.

4 Käytettävät tunnistamisratkaisut

4.1 Tausta

Siitä lähtien, kun internetissä on ollut useampia suosittuja palveluita, on ollut myös tarve yhdistää näitä palveluita siten, että käyttäjä voi helposti hyödyntää palveluita yhdessä. Perinteinen ongelma on ollut, että käyttäjä käyttää palvelua A ja sitä täydentävää kolmannen osapuolen palvelua B, ja käyttäjä pitäisi onnistua luotettavasti tunnistamaan molemmissa palveluissa. Ennen vuotta 2006 tähän ei ole ollut olemassa mitään yhteistä standardia, joten yritykset kehittivät tähän omia ratkaisujaan. Yksi näistä on OAuth.

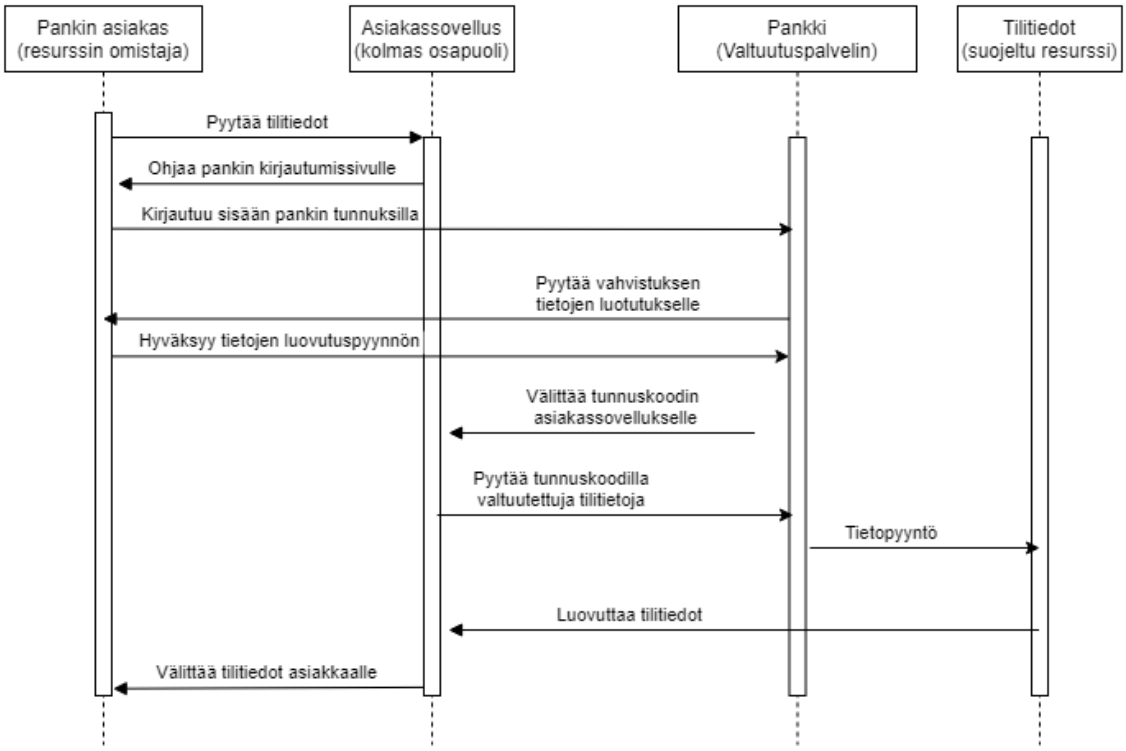
OAuth-konsepti syntyi 2006 yritysten, kuten Twitter, tarpeesta käyttää käyttäjien tietoja ohjelmistorajapinnoissa. Tästä kehitettiin 2010 määritelty standardi nimellä OAuth 1.0a. [12.] Määrittelyssä oli kuitenkin heikkouksia, kuten käytetyt salausalgoritmit, jotka sittemmin ovat osoittautuneet haavoittuviksi. Tämän pohjalta on lähdetty määrittelemään korvaavaa standardia, joka on nimeltään OAuth 2.0.

OAuth 2.0 on Internet Engineering Task Forcen (IETF) määrittelemä valtuuttamisen viitekehys, joka on tarkoitettu antamaan kolmannelle osapuolelle, yleensä toiselle sovellukselle, rajattu oikeus käyttää käyttäjän tietoja toisessa palvelussa HTTP-protokollan välityksellä [13]. OAuth 2.0 paikkasi OAuth 1.0a:n heikkouksia. Suunnittelun lähtökohtana oli modulaarisuus, jossa toteuttajan vastuulle jää monia yksityiskohtia, kuten käytettävät salausalgoritmit. Uusi versio ei myöskään ota kantaa siihen, miten tunnistautumiseen vaadittava tunnus luodaan, tai siihen, miten se siirretään kolmannelle osapuolelle. Tämä tekee standardista joustavamman, ja siten standardi säilyy pidempään käyttökelpoisena tulevaisuudessakin.

4.2 Tekninen määritelmä

OAuth 2.0 (myöhemmin vain OAuth) perustuu seuraaviin osapuoliin, joita ovat resurssin omistaja, asiakassovellus, valtuutuspalvelin ja suojeltu resurssi. Esimerkiksi pankkipalveluissa resurssin omistaja on pankin käyttäjä (asiakas), jolla on pankissa oma tili. Asiakassovellus on PSD2-rajapintoja hyödyntävä kolmannen osapuolen sovellus, joka tarvitsee pääsyn resurssin omistajan suojeltuun resurssiin, eli pankkitilin tietoihin. Valtuutuspalvelin voi OAuth-standardin mukaan olla kuka tahansa, mutta käytännössä se on pankin tarjoama palvelu, joka tunnistaa käyttäjänsä ja jakaa käyttäjän luvalla tietoa kolmannelle osapuolelle.

OAuthin toiminnan ydin on tunnuskoodin (access token) luominen ja sen käyttäminen. Esimerkiksi pankkipalveluissa, kun pankin asiakas (resurssin omistaja) haluaa pääsyn omiin tilitietoihinsa (suojeltu resurssi) ulkopuolisessa sovelluksessa, kuten toisen yrityksen tarjoamassa kirjanpito-ohjelmassa (asiakassovellus), pyytää kirjanpitosovellus valtuutusta pankin valtuutuspalvelimelta pankin asiakkaan puolesta. Asiakas kirjautuu pankkiin sisään, ja valtuutuspalvelin pyytää käyttäjää hyväksymään kirjanpitosovelluksen pyytämät tilitiedot, eli suojellut resurssit. Hyväksynnän jälkeen valtuutuspalvelin palauttaa kirjanpitosovellukselle tunnuskoodin, jota sovellus voi käyttää asiakkaan tietojen pyytämiseen. Tämä on havainnollisettu kuvassa 1.



Kuva 1. Oauthin toimintaprosessi tilitietoja pyydettäessä

Tunnuskoodin määritelmään kuuluu, että lähtökohtaisesti kuka tahansa voimassa olevan koodin haltija on oikeutettu suojeltuun tietoon. Tunnuskoodi on voimassa yleensä rajatun aikaa, jonka päättymisen jälkeen asiakassovelluksen on aloitettava valtuutusprosessi uudestaan. Jos tunnuskoodin voimassaoloaika on lyhyt (tunteja tai muutamia päiviä), olisi käyttäjältä kysyttävä valtuutta yhä uudelleen ja uudelleen. Tämä johti ensimmäisessä OAuthin versiossa usein siihen, että kehittäjät määrittelivät tunnuksille ikuisen tai huomattavan pitkän voimassaoloajan (vuosia), mikä on ongelmallista, jos tunnus päätyy ulkopuolisen haltuun. Tämän vuoksi OAuthin määrittelyssä on mukana myös virkistyskoodi (refresh token), joka annetaan samalla kun asiakassovellus vastaanottaa tunnuskoodin ja sillä voi olla pidempi voimassaoloaika.

Virkistyskoodilla sovellus pystyy pyytämään uutta tunnuskoodia samoilla valtuutuksilla kuin millä alkuperäinen pyyntö on tehty. Virkistyskoodilla itsellään ei siis pääse suoraan käsiksi suojeltuun tietoon. Se antaa vain oikeuden pyytää uutta tunnuskoodia. Virkistyskoodia käyttävä sovellus tunnistautuu valtuutuspalvelimelle kuten alkuperäistä tunnuskoodia hakiessa, mutta ilman käyttäjältä erikseen pyydettävää valtuutusta. Tällöin käyttäjä pystyy jatkamaan sovelluksen käyttöä ilman uudelleen tunnistautumista, eikä

käyttäjän myöskään tarvitse tietää mitään taustalla tapahtuvasta tunnuskoodin vaihdosta. Prosessi voidaan jälleen toistaa, kun uuden tunnuskoodin voimassaoloaika päättyy. [14, luku 1.] Pankkien kanssa sekä tunnuskoodin että virkistyskoodin enimmäisvoimassaoloaika on 90 päivää, joten virkistyskoodin merkitys on vähäinen, ellei asiakassovellus määrittele tunnuskoodille jostain syystä lyhyempää voimassaoloaikaa. Vaikka tunnuskoodi määritelmän mukaan antaa oikeuden suojeltuun tietoon, niin käytännössä suojausta vahvistetaan myös muilla keinoilla, kuten pankkipalveluissa käytettävillä digitaalisilla varmenteilla ja rajapinta-avaimilla, jolloin pelkän tunnuskoodin väärin käsiin joutuminen ei vaaranna vielä tietoa.

4.3 Käytettävä tunnuskoodi

OAuthin kanssa käytettävää tunnuskoodin sisältöä ei ole määritelty muuten kuin, että sen täytyy olla sekä valtuutuspalvelimen ja suojeltavan resurssin sisältävän palvelimen ymmärtämä. Asiakassovelluksen ei tarvitse tietää tunnuskoodista mitään, sillä tunnuskoodi vain välitetään valtuutuspalvelimelta resurssia jakavalle palvelimelle. Määrittelemättä jättäminen tarkoittaa sitä, että tunnuskoodin muoto voi vaihdella suurestikin erilaisten palveluiden välillä. Tunnuskoodit saattavat olla vanhentuvia, peruutettavia tai ikuisia, vaihdellen käyttötarkoituksen mukaan. Tunnuskoodi itsessään ei välttämättä sisällä mitään tietoa sen käyttäjästä. Tällöin sekä tunnuksen luojalla ja tunnuksen käyttäjällä täytyy olla jokin tapa jakaa tieto, kenelle tunnus on luotu, kuten yhteinen tietokanta, josta tieto tarkistetaan. Tämä ei kuitenkaan ole käytännöllistä, tai edes mahdollista, jos kyse on kokonaan erillisistä toimijoista. Ongelman ratkaisemiseksi on kehitetty JSON Web Token -standardi (JWT) [15] tunnuskoodin välittämistä varten.

JWT tarjoaa yksinkertaisen tavan kuljettaa tietoa eri palveluiden välillä. Tämän tiedon joukossa voidaan lähettää valtuutuspalvelimen vaatimaa tietoa, kuten kuka tunnuksen on valtuuttanut, ja voimassaoloaika. JWT-tunnuskoodi perustuu JSON-objektiin (JavaScript Object Notation), joka koodataan Base64URL-muotoon. Base64URL tekee objektista ihmiselle lukukelvottoman, mutta se ei kuitenkaan ole salausalgoritmi. Base64URL-koodauksen tarkoitus on yksinkertaisesti helpottaa tiedon välittämistä eri palveluiden välillä, sillä JWT-tunnukset välitetään usein HTTP-pyyntöjen otsikkotunnisteissa (headers), joissa on rajattu tila käytössä. Base64URL-koodattu tieto käyttää vähemmän merkkejä saman tiedon esittämiseen. Myös erikoismerkit koodataan, jolloin

palveluiden toteuttajien ei tarvitse itse huolehtia erikoismerkkien oikeasta muodosta, ollakseen kelvollisia ennen tiedon siirtoa toiseen palveluun. Se tekee siitä ihanteellisen muodon käytettäväksi HTTP-liikenteeseen. Asiakassovelluksen tunnistamiseen JWT:tä käyttävät Osuuspankki ja S-Pankki.

4.4 JWT:n rakenne

Yksinkertaisimmillaan JWT-tunnus koostuu kahdesta Base64URL-koodatusta osasta, jotka on eroteltu toisistaan pisteillä. Ensimmäinen osa koostuu otsikkotietueista, jotka sisältävät tietoa lähetettävästä tiedosta. Toinen osa on varsinainen sisältö, joka on tarkoitus toiselle osapuolelle välittää. Kun lähetettävä tunnus myös allekirjoitetaan, lisätään perään kolmas, pisteellä erotettu osio. Esimerkkikoodissa 1 eri osiot ovat eri värillä erottelun helpottamiseksi. Allekirjoituksen tarkoitus on taata, että välitettävä tieto on aito, koskematon ja että lähettäjä on se, jonka vastaanottava osapuoli odottaa sen olevan.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImFkeGsxIn0.eyJzdWIiO-
iIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.HDiER-
Iagkulc5UNlTKIJzpWTwH8n1vIa-NM87m89I4Q
```

Esimerkkikoodi 1. Esimerkki JWT-tunnuksesta siirtokelpoisessa muodossaan

Ensimmäinen osa, eli ensimmäinen merkkijono pisteeseen asti sisältää JWT:n otsikkotietueet. Kun ensimmäisen osan Base64URL-koodauksen purkaa, näyttää sisältö kuten esimerkkikoodissa 2.

```
{
  "alg": "HS256",
  "typ": "JWT",
  "kid": "adxk1"
}
```

Esimerkkikoodi 2. JWT-otsikkotietueet

Otsikkotietueet paljastavat, että kyseessä on objekti tyyppiä JWT (”typ”) ja se on allekirjoitettu HMAC-SHA-256-algoritmia (lyhennettynä. HS256) käyttäen (”alg”). Viimeisenä tietokenttänä on ”kid”, joka on lyhennys sanoista Key Id, jota käytetään esimerkiksi lähettävän sovelluksen tunnisteena. Tunnuksen voi lähettää myös allekirjoittamattomana,

jolloin “alg”-tietueeseen kirjataan merkintä ”none”. Tällöin kolmatta pisteellä erotettua osaa ei odoteta olevan mukana. Toinen osa purettuna näkyy esimerkkikoodissa 3.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Esimerkkikoodi 3. JWT:ssä välitettävä tietosisältö

JWT:n tietosisältöä kutsutaan vaatimuksiksi (claim). Jotkin otsakkeet sisältyvät standardin määritykseen, kuten esimerkiksi tunnuksen luoja (”iss”), kenelle tunnus on suunnattu (”aud”) ja koska se on luotu (”iat”). Muuten tietosisältöä ei ole rajattu, vaan se voi sisältää mitä tahansa tietoa, mikä on Base64URL-koodattavissa, mikä ei käytä standardissa määriteltyjä otsakkeita eri tarkoitukseen. Esimerkkikoodissa 3 on otsake “name”, joka on tunnuksessa välitettävää tietoa.

Allekirjoituksen luominen ja lukeminen perustuu julkisen avaimen infrastruktuuriin. Käytettävälle allekirjoitustavalle on erilaisia vaihtoehtoja, kuten JSON Web Signature (JWS) ja JSON Web Encryption (JOSE). Yksinkertaistaen kyse on välitettävän tietosisällön ja valitun salausalgoritmin perusteella luotavasta merkkijonosta, jonka vastaanottava osapuoli osaa purkaa. Tästä syystä tunnuksen otsaketiedossa kerrotaan käytettävä salausalgoritmi. Kun tietosisältö ja purettu allekirjoitus täsmäävät, voi vastaanottava osapuoli luottaa välitettyyn tietoon.

PSD2-rajapinnoissa allekirjoitusta varten suositellaan käytettävän QSEAL-varmennetta. Tällöin asiakassovellus käyttää varmenteen yksityistä avainta JWT:n tietosisällön allekirjoittamiseen, ja vastaanottava pankki käyttää varmenteen julkista osaa sen purkamiseen.

4.5 Muut tunnistautumisen mekanismit

Tutkittavista rajapinnoista Nordea ei käytä JWT:tä ja sen allekirjoitusmekanismia asiakassovelluksen tunnistamiseen. Nordealla on käytössä Draft Cavage HTTP Signature -työnimellä tunnettu metodi, joka on tarkoitettu HTTP-viestien allekirjoittamiseen, mutta joka ei ole vielä saavuttanut standardin asemaa. Nordea viittaa dokumentissaan vuonna

2018 kirjoitettuun määrittelyyn [16]. Sen tarkoitus on jotakuinkin sama kuin JWT:n allekirjoittamisessa: tarkoitus on varmistaa, että lähetetty tieto on aito ja muuttamaton sekä lähettäjän identiteetin varmistaminen.

Määrittely ei ole kovin monimutkainen. Joitakin otsaketietoja on pakko käyttää, kuten ”keyId”, joka on toteuttajan valittavissa oleva yksilöintikeino. Nordealla se tarkoittaa rajapinnan käyttäjälle myönnettyä rajapinta-avainta, jota käytetään kaikissa rajapintoihin tehdyissä kutsuissa. Toinen pakollinen kenttä on allekirjoituksessa syntyvälle merkkijonolle. Allekirjoitus muodostetaan HTTP-viestin sisällöstä ja allekirjoituksen otsaketiedoista, jotka Nordean rajapinnassa laitetaan erillisen ”headers”-otsakkeen alle. Koko HTTP-viestiä ei kuitenkaan käytetä sellaisenaan allekirjoituksen luomiseen, vaan tilan säästämiseksi viestin sisällöstä lasketaan ensin hajautusarvo käyttäen RSA-SHA256-algoritmia. Allekirjoitukseen käytetään QSEAL-varmenteen yksityistä avainta. Nordea vaatii allekirjoituksen sisällyttämistä jokaiseen rajanpintaan tehtävään pyyntöön. Testiympäristössä tämän vaiheen voi ohittaa merkitsemällä HTTP-pyynnön otsaketietoihin ”SKIP_SIGNATURE_VALIDATION_FOR_SANDBOX”.

Vertailtaessa tunnistautumisen ratkaisuja pankkien välillä voidaan todeta, että Handelsbanken ei vaadi tiedon allekirjoittamista missään vaiheessa tiedonsiirtoa. Ainut vaatimus on QWAC-varmenteen käyttö, jolla voi taata tiedon lähettäjän identiteetin. Osuuspankki vaatii allekirjoitusvaatimusta vain asiakassovelluksen tunnistamisvaiheessa ja siten myöhempien rajapintakutsujen oletetaan olevan luottamuksellisia, koska lähettäjä on todettu luotettavaksi.

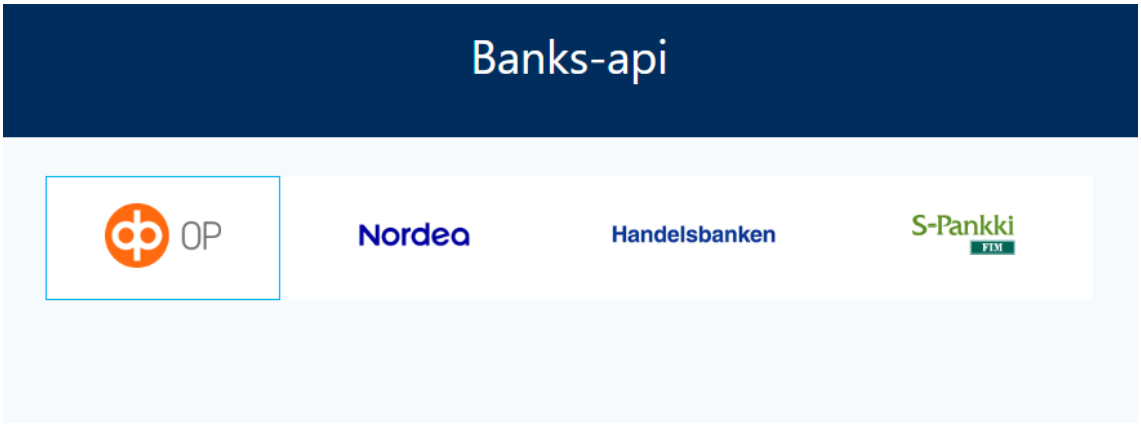
5 PSD2-rajapintojen käyttö Kotlin-sovelluksessa

5.1 Esimerkkisovellus

Insinöörityötä varten kehitettiin esimerkkisovellus, jotta syntyisi konkreettinen malli, jota voisi käyttää esimerkkinä oikean rajapintoja hyödyntävän sovelluksen kehittämiseen. Sovelluksessa pystyy selaamaan eri pankkien tarjoamaa testitietoa tilitiedoista, tilitapahtumista ja maksun suorittamisesta. Tavoitteena oli keskittyä rajapintojen haastavimpaan osuuteen, eli sovelluksen ja rajapinnan keskinäisen tunnistautumisen ja turvallisen

tiedonsiirron toteutukseen. Työssä käydään vaadittavat vaiheet tarkasti läpi koodiesimerkein, jotka ovat esimerkkisovelluksesta.

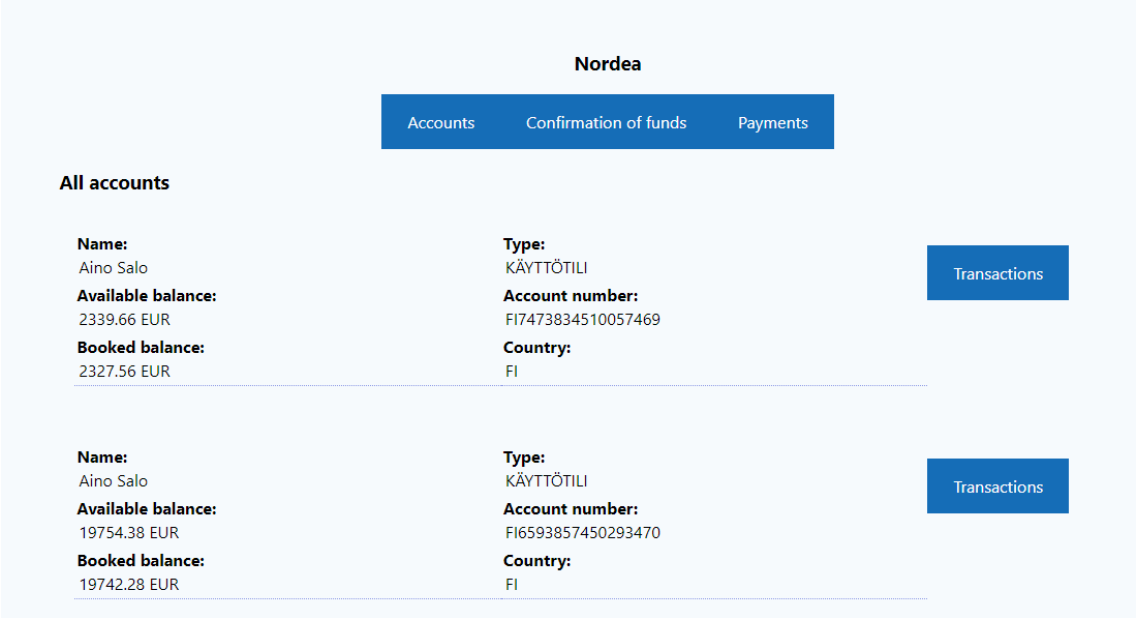
Sovellus toteutettiin siten, että siitä puuttuu pankkien tarjoama ja loppuasiakkaiden käyttämä kirjautumisvaihe. Kuvassa 2 näkyvällä sovelluksen etusivulla valitaan käytettävä pankki logoa painamalla ja asiakkaan tunnistautuminen oletetaan jo suoritetuksi. Pankin valitsemalla sovellus antaa pääsyn kyseisen pankin rajapinnan tarjoamiin tietoihin.



Kuva 2. Sovelluksen etusivu

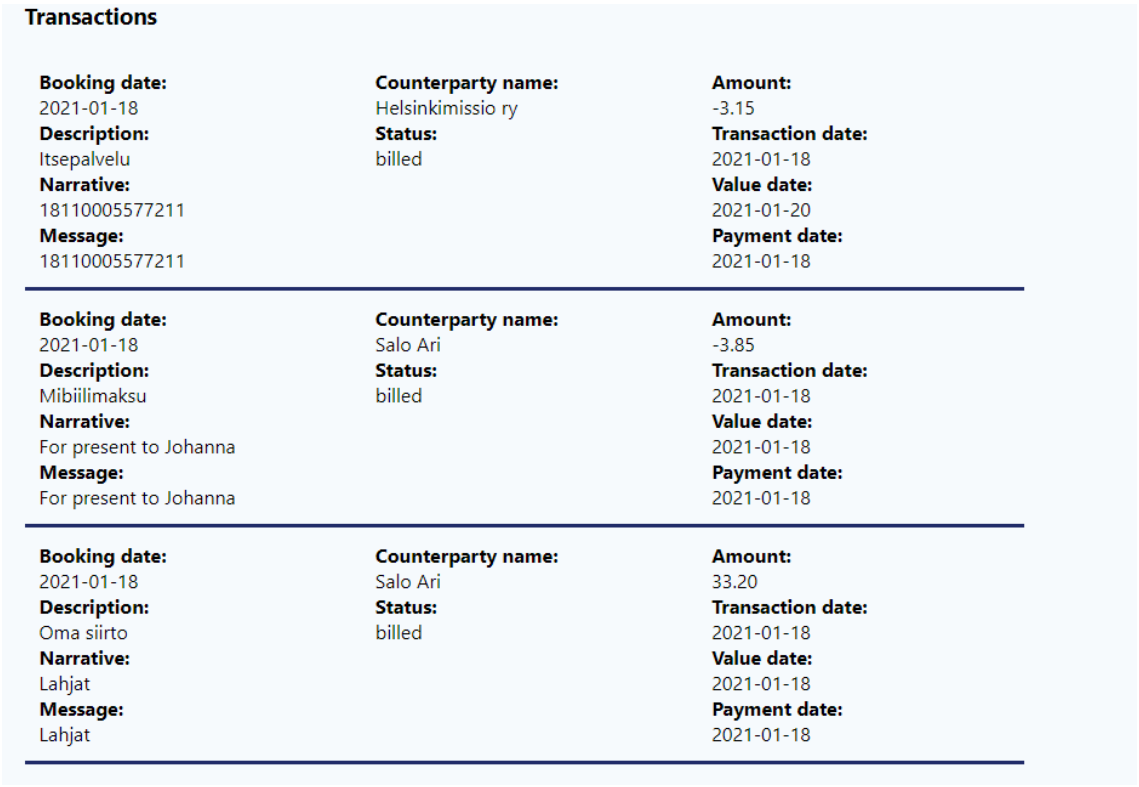
Käyttöliittymä on yksinkertainen, jolla pystyy nopeasti selaamaan läpi sovelluksen tarjonnan. Esimerkkisovellukseen toteutettiin tilitietojen ja maksujen luonti niiltä osin, kuin testirajapintojen toteutus sen salli ilman, että erillistä asiakkaan kirjautumisvaihetta vaadittiin.

Valitsemalla pankin voi saa nähtäville testiympäristön tarjoamat tilit ja niiden tiedot, kuten kuvasta 3 näkee Nordean tarjoamaa testitietoa. Näihin kuuluvat tilin haltijan nimi, tilin tyyppi, tilinumero, tilin saldo ja mahdolliset varaukset sisältävä saldo, joka kuvassa esitetään termillä booked balance.



Kuva 3. Nordean rajapinnasta saadut tilitiedot

Tileiltä saa erikseen niihin kohdistuvat tilitapahtumat, jotka näkyvät kuvassa 4. Tilitapahtumien tietoihin sisältyy odotuksen mukaisia tietoja: vastaanottaja, summa, viesti ja erilaisia päiväyksiä, kuten kirjauspäivä ja maksupäivä. Lisäksi sovelluksessa pystyy luomaan, vahvistamaan ja poistamaan maksuja. Tämä on mukana olevista rajapinnoista ainoa, jossa testidataa voi käsitellä dynaamisesti, kun muut ovat vain tiedonhakemista varten.



Transactions

Booking date: 2021-01-18	**Counterparty name:** Helsinkimissio ry	**Amount:** -3.15
Description: Itsepalvelu	**Status:** billed	**Transaction date:** 2021-01-18
Narrative: 18110005577211		**Value date:** 2021-01-20
Message: 18110005577211		**Payment date:** 2021-01-18
Booking date: 2021-01-18	**Counterparty name:** Salo Ari	**Amount:** -3.85
Description: Mibiilimaksu	**Status:** billed	**Transaction date:** 2021-01-18
Narrative: For present to Johanna		**Value date:** 2021-01-18
Message: For present to Johanna		**Payment date:** 2021-01-18
Booking date: 2021-01-18	**Counterparty name:** Salo Ari	**Amount:** 33.20
Description: Oma siirto	**Status:** billed	**Transaction date:** 2021-01-18
Narrative: Lahjat		**Value date:** 2021-01-18
Message: Lahjat		**Payment date:** 2021-01-18

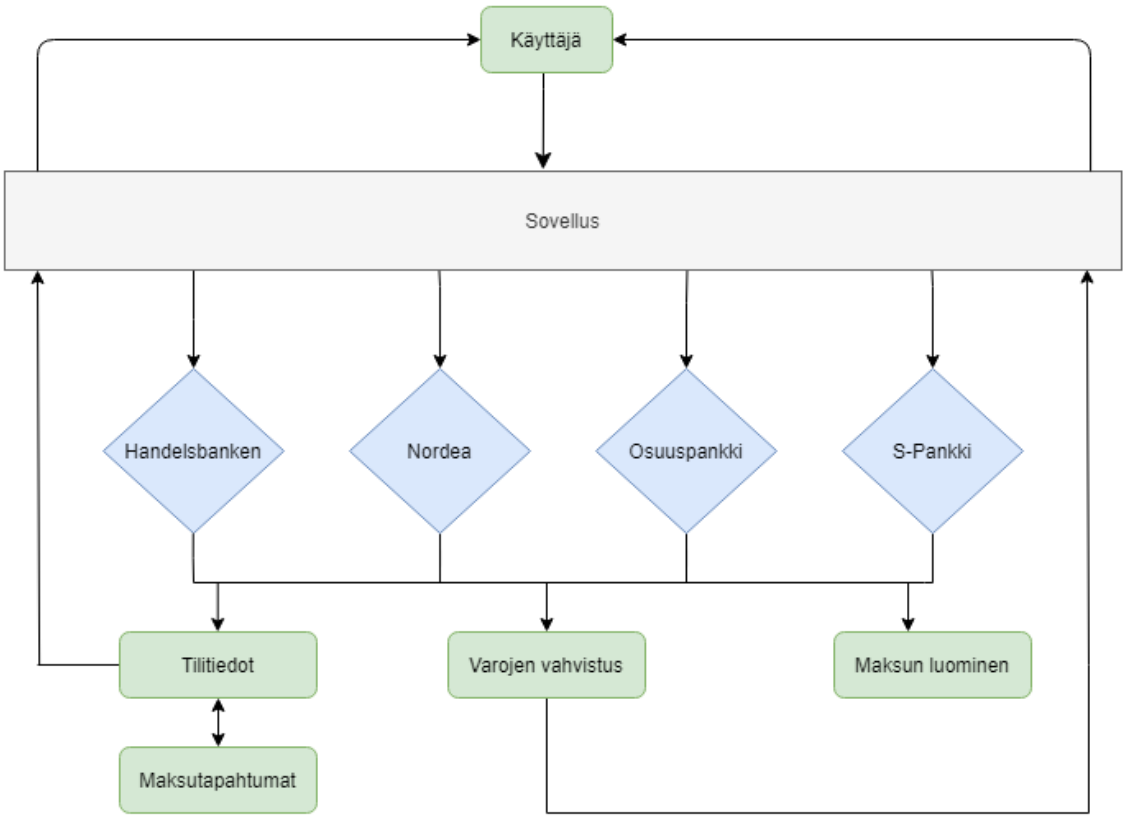
Kuva 4. Tilitapahtumatietoja, jotka kohdistuvat yksittäiseen tiliin.

5.2 Teknologiavalinnat

Esimerkkisovelluksen teknologiavalinnat tehtiin pääasiassa sen perusteella, mikä tuntui itsensä kehittämisen kannalta hyödylliseltä. Palvelinpuolen ohjelmointikieleksi valikoitui Java Virtual Machine -pohjainen Kotlin. Palvelimella käytettiin Java-ympäristöissä suosittua Spring Boot -sovelluskehystä, joka tukee Javan lisäksi Kotlinia. Sovelluksen käyttöliittymä rakennettiin JavaScript-pohjaisella React-kirjastolla, joka on saavuttanut suuren suosion verkkosovelluksissa.

Esimerkkisovelluksessa on toteutettu palvelimelle REST-arkkitehtuurimallin rajapinta, jota käyttöliittymästä kutsutaan. Palvelin toimii lähinnä yksinkertaisena tiedon välittäjänä pankkien omien REST-rajapintojen ja esimerkkisovelluksen käyttöliittymän välillä, mikä on havainnollistettu kuvassa 5. Palvelinpuolen tärkein tehtävä on hoitaa pankkien rajapintojen vaativa tunnistautumisprosessi, jonka jälkeen pankkien tarjoamat rajapinnat ovat käytettävissä. Palvelinsovellus ei sisällä minkäänlaista pysyvää

tallennusjärjestelmää, kuten tietokantaa, joten esimerkkisovellus ei sellaisenaan sovellu oikeaan käyttöön.



Kuva 5. Esimerkkisovelluksen käyttöympäristö

5.2.1 Kotlin

Kotlin on vuonna 2011 julkaistu ohjelmointikieli [17]. StackOverFlow'n kehittäjäkyselyn mukaan kieli on kasvattanut suosiotaan tasaisesti viime vuosina. Vuonna 2018 se oli työelämässä 22. käytetyin kieli, 2019 16. käytetyin ja 2020 13. käytetyin [18; 19; 20]. Yksi merkittävä syy suosiolle on Googlen ilmoitus vuonna 2019, että Android-käyttöjärjestelmän sovelluskehitys tukee jatkossa ensisijaisesti Kotlinia [21].

Kotlinin vahvuuksia ovat ytimekäs syntaksi, staattinen tyypitys ja hyvät kehitystyökalut. Javaan verrattuna monet asiat saadaan esitettyä huomattavasti lyhyemmin ja luettavammin. Kotlin on suunniteltu täysin yhteen toimivaksi Javan kanssa [22], joten sen kanssa pystyy vaivatta käyttämään hyväksi Javan laajaa kirjastokokoelmaa.

Kielen takana on JetBrains-niminen yritys, joka on tuottanut paljon koodieditoreita ja ohjelmistoympäristöjä eri kielille, joten heidän tavoitteenansa on ollut alusta asti hyvät työkalut. Esimerkiksi JetBrainsin IntelliJ IDEA -ohjelmistoympäristössä Javalla tehdyn luokan pystyy kääntämään Kotlin-kieliseksi nappia painamalla. Kokeilujen perusteella tämä toimii varsin hyvin. Tosin kaikissa tilanteissa koodi ei käänny hyvin ihmisen ymmärtämään muotoon ja koodia pitää luettavuuden vuoksi parantaa.

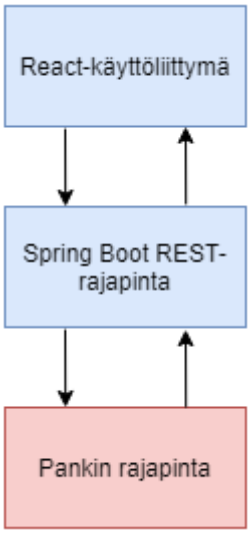
Kielenä Kotlin on melko helppo oppia, jos taustalla on jo Java-osaamista, sillä syntaksi ja toimintatavat ovat kuitenkin lähellä toisiaan. Tarpeen vaatiessa koodin sekaan voi liittää Java-koodia, jolle IntelliJ tarjoaa automaattisesti käännöstä Kotliniksi.

5.2.2 Spring Boot

Spring Boot on Java-pohjainen sovelluskehys, joka rakentuu laajemman, Spring-sovelluskehyksen päälle. Spring Boot tarjoaa nopean aloituksen kehitysvaiheen alkuun, koska se sisältää valmiiksi huomattavan määrän määrityksiä, joita tavallisessa web-sovelluskehityksessä tarvitaan. Esimerkiksi käytettävä HTTP-palvelinsovellus, TomCat, käynnistyy automaattisesti, kun projektin käynnistää ilman, että kehittäjän tarvitsee tehdä mitään. Esimerkkisovellukseen Spring Boot valittiin, koska se on erittäin suosittu, eikä siitä ollut juurikaan aikaisempaa kokemusta.

Spring Initializr -websovelluksella pystyy generoimaan esimerkiksi muutamassa minuutissa käynnistysvalmiin REST-sovelluksen tekemällä kielivalinnan, valitsemalla haluamansa version ja tarpeelliset riippuvuudet sen toteuttamiseksi. Käyttäjä voi valita myös haluamansa projektin rakennustyökalun, Mavenin tai Gradlen. Esimerkkisovellukseen valittiin Maven, koska se oli ennalta tutumpi työkalu.

Spring Boot osoittautui hyväksi valinnaksi: projektin käynnistäminen oli nopeaa ja suosion vuoksi tukea ja ohjeistusta löytyy internetistä paljon. REST-rajapinnan toteuttamisessa monet ominaisuudet perustuvat Javan annotaatioihin, joilla palvelimen asetuksia saa asetettua, mikä teki kehittämisestä suoraviivaista. Kuvassa 6 on havainnollistettu esimerkkisovelluksen arkkitehtuuria.



Kuva 6. Sovelluksen arkkitehtuuri

5.2.3 React

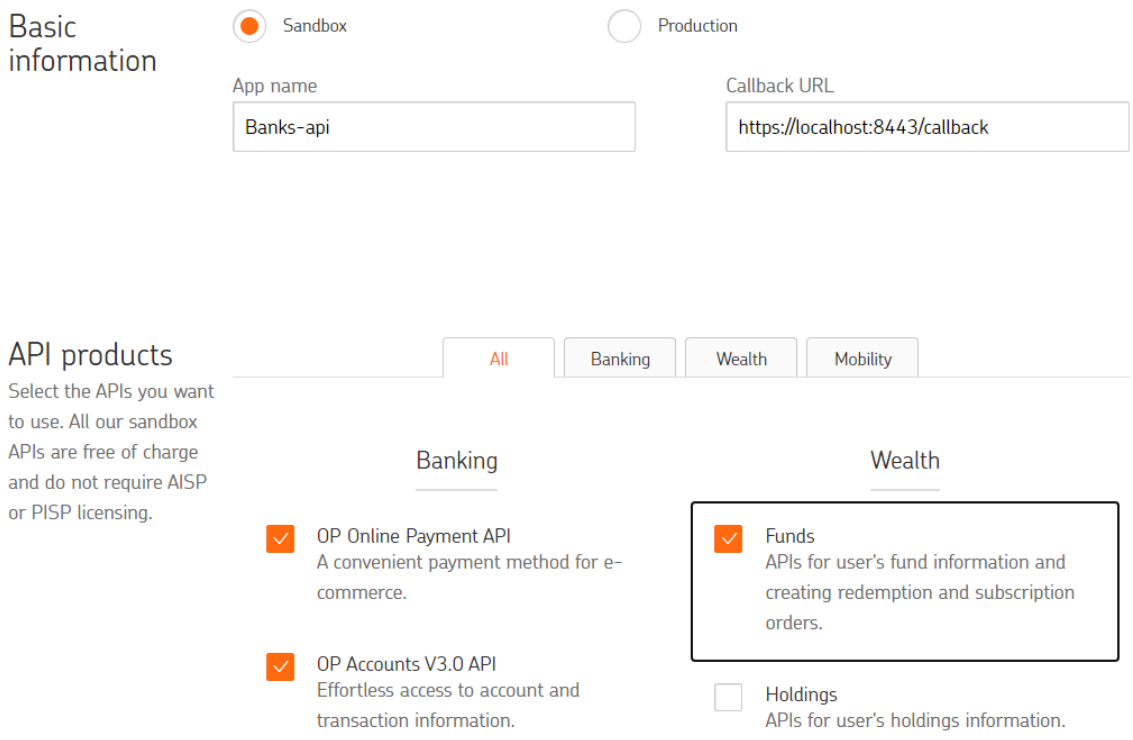
Esimerkkisovellus tarvitsi yksinkertaisen käyttöliittymän, jossa dynaamisesti vaihteleva tieto saadaan vaivattomasti näkyville. React on JavaScript-pohjainen kirjasto, joka lupaa interaktiivisen käyttöliittymän luomisesta kivutonta [23]. React on Facebookin tuottama ja edelleen aktiivisesti kehitettävä avoimen lähdekoodin kirjasto.

Esimerkkisovelluksessa käyttöliittymän painoarvo jäi vähäiseksi huomion keskittyessä ennen kaikkea siihen, miten pankkien rajapintoja pääsee käyttämään. React kuitenkin piti lupauksensa ja yksinkertaisen käyttöliittymän kehittäminen, tiedon hakeminen ja sen näyttäminen oli suhteellisen vaivatonta.

Jatkokehityksenä käyttöliittymän kielen voisi vaihtaa JavaScriptistä TypeScriptiin, joka on Microsoftin kehittämä ohjelmointikieli. TypeScript on JavaScript-pohjainen kieli, joka lisää kieleen staattisen tyypityksen. Tämä helpottaisi sovelluskehitystä, varsinkin mitä isommaksi sovellus kehittyy. React tukee myös TypeScriptiä.

5.3 Toteutuksen aloitus

Ennen kuin rajapintoja voi käyttää, piti kunkin pankin tarjoamalle kehittäjien sivustolle rekisteröityä itse, ja rekisteröidä kehitettävä sovellus. Sovellukselle määritellään nimi, niin sanottu uudelleenohjausosoite (callback URL) ja valitaan käytettävät rajapinnat kuten esimerkiksi Osuuspankin palvelussa kuvassa 7. Pankki käyttää uudelleenohjausosoitetta, kun loppuasiakas on kirjautunut onnistuneesti sisään ja hyväksynyt tietojensa luovutuksen sovellukselle. Uudelleenohjausosoitteen käyttö käydään tarkemmin läpi myöhemmin osiossa 5.5. Kun sovellus on rekisteröity, voi rajapintoja alkaa hyödyntämään. Myös pankkien tarjoamat testivarmenteet täytyy ottaa käyttöön.



Kuva 7. Osuuspankin kehittäjäsivuston sovelluksen määrittely

Ennen kuin rajapintoja voi käyttää, täytyy aiemmin läpikäydyt tunnistautumisen teknologiat toteuttaa itse sovelluksessa. Niitä ennen täytyy huolehtia myös palvelimen sekä loppukäyttäjän tiedonsiirron turvallisuudesta käyttämällä siinä HTTPS-protokollaa (Hypertext Transfer Protocol Secure) HTTP-pyyntöjen kuuntelemiseen. Näin tiedonsiirto pysyy salassa koko sen elinkaaren ajan: loppuasiakkaalta esimerkkisovellukselle,

esimerkkisovelluksesta pankin palvelimelle ja sieltä takaisin loppuasiakkaalle. Esimerkkisovelluksen palvelimella hyödynnettiin tähän Osuuspankin generoimaa varmennetta, vaikka kehityskäytössä olisi mahdollista käyttää itse generoitua ja itse allekirjoitettua varmennetta.

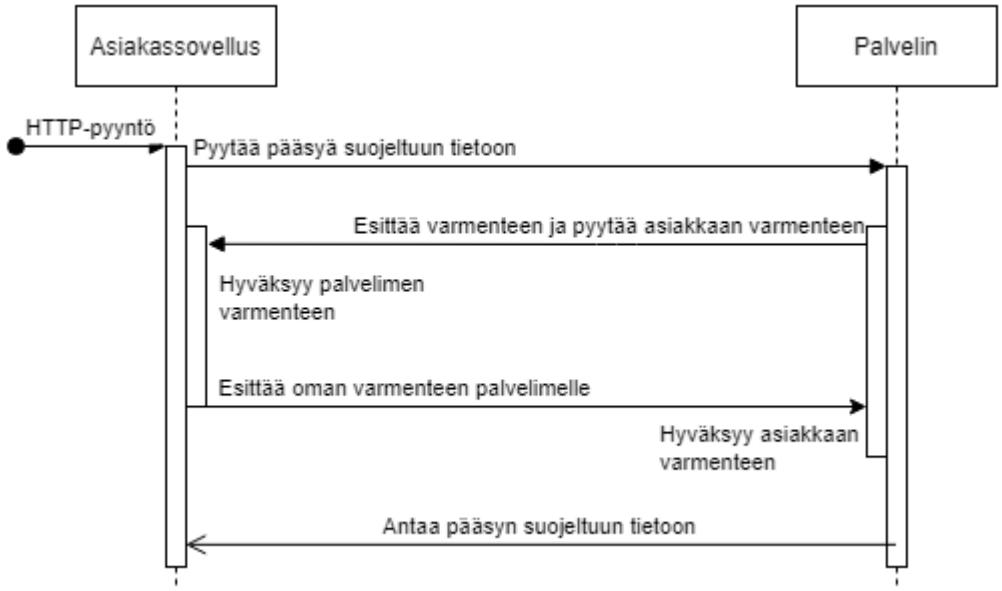
Osuuspankin generoima varmenne on PKCS12-formaatissa, joka sisältää X.509-muotoisen julkisen varmenteen ja siihen yksityisen avaimen. Varmenne tallennettiin sovelluksen resources-hakemistoon. Palvelin konfiguroitiin käyttämään varmennetta kirjoittamalla varmenteen tiedot application.properties-nimiseen tiedostoon esimerkkikoodissa 4 näkyvällä tavalla. Vaadittavia tietoja ovat kuunneltava portti, varmenteen sijainti, varmenteen alias eli nimi ja salasana. Spring Boot osaa etsiä oikean tiedoston, kun tiedostonimen eteen laittaa etuliitteen classpath:.

```
server.ssl.key-store-type=PKCS12
server.ssl.key-store=classpath:OP-TEST-TPP-1234f5-a33d-client.p12
server.ssl.key-alias=client
server.ssl.key-store-password=test
server.port=8443
```

Esimerkkikoodi 4. Varmenteen käyttöön ottaminen Spring Boot -sovelluksessa

5.4 Mutual TLS:n käyttäminen Spring Boot -sovelluksessa

Seuravaksi käsitellään tiedonsiirron turvaaminen esimerkkisovelluksen ja pankin palvelimen välillä. PSD2-rajapinnat vaativat Mutual TLS (MTLS) -protokollan käyttöä, jossa kaksi osapuolta tunnistaa molemmat toisensa siten, että molemmat osapuolet voivat olla varmoja toisen osapuolen identiteetistä. Tällöin molemmilla asianosaisilla on oma varmenne, jonka luottamusverkosto on myöntänyt. Tietoa siirrettäessä MTLS-prosessi on korkealla tasolla kuvan 8 mukainen, jossa molemmat osapuolet esittävät varmenteensa ja molemmat osapuolet varmistavat tahoillaan varmenteen luotettavuuden ennen kuin tietoa siirretään. PSD2-rajapinnoissa tähän käytetään luvussa 3 esiteltyä QWAC-varmennetta.



Kuva 8. MTLS:n tunnistautumisprosessi ennen tiedonsiirtoa

Seuraava vaihe oli konfiguroida pankin rajapintaan tehdyt pyynnöt käyttäen kuhunkin pankkiin pankin antamaa testivarmennetta. HTTPS-pyyntöjen tekemiseen käytettiin Java 11:n myötä tullutta, päivitettyä HTTP-kirjastoa Http Clientia.

Http Client -kirjasto käyttää rakentajarajapintasuunnittelumallia [24, luku Object Creational: Builder], jonka tarkoituksena on erottaa monimutkaisen olion rakenne sen esitystavasta. Oliota luodessa pyydetään ensin rakentajaolio, jonka metodeja kutsumalla monimutkaisen olion luominen on yksinkertaista. Esimerkkikoodissa 5 HttpClient.newBuilder()-metodikutsu palauttaa rakentajaolion. Sen kaikissa metodeissa annettu parametri tallennetaan rakentajaolioon, ja lopuksi metodi palauttaa viitteen kutsuttavaan rakentajaolioon. Tämä mahdollistaa metodikutsujen ketjuttamisen kuten esimerkissä. Lopussa kutsutaan build-metodia, joka luo HttpClient-objektin rakentajalle annetuilla parametreilla.

```
return HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .sslContext(buildSslContext(bank))
                .followRedirects(
                    when {
                        shouldRedirect -> HttpClient.Redirect.ALWAYS
                        else -> HttpClient.Redirect.NEVER
                    }
                )
                .build()
```

Esimerkkikoodi 5. HTTP-pyynnöt suorittava HttpClient-objekti luodaan rakentajarajapintasuunnittelumallia hyödyntäen

Tärkein askel on käyttää rakentajasta löytyvää sslContext-metodia varmenteen määrittämiseen, joka esimerkkikoodissa 6 asetetaan SLLContext-tyyppiseen olioon. Metodissa varmenne ladataan tiedostosta ja asetetaan SLLContext-oliolle sekä määritetään käytettävä protokolla, joka PSD2-rajapinnoissa täytyy olla vähintään TLS:n 1.2-versio. Jokainen pankkien rajapintoihin tehtävä HTTP-pyyntö alustettiin käyttäen tätä metodia, mikä mahdollisti turvallisen tiedonsiirron sovelluksen ja pankkien rajapintojen välillä. Koska jokaisen pankin testiympäristö tarjosi oman varmenteen käytettäväksi, piti tämä tehdä kaikille pankeille. Testatessa tosin selvisi, että ainakin Nordean rajapinta ei testiympäristössä välittänyt käytettävästä varmenteesta, mikäli viestit olivat allekirjoittamattomia, vaan se toimi myös Osuuspankin generoimalla varmenteella, mikä tuo joustavuutta kehittämiseen. Seuraavaksi käsitellään, kuinka asiakas saadaan ohjattua pankin kirjautumissivustolle JWT:n avulla.

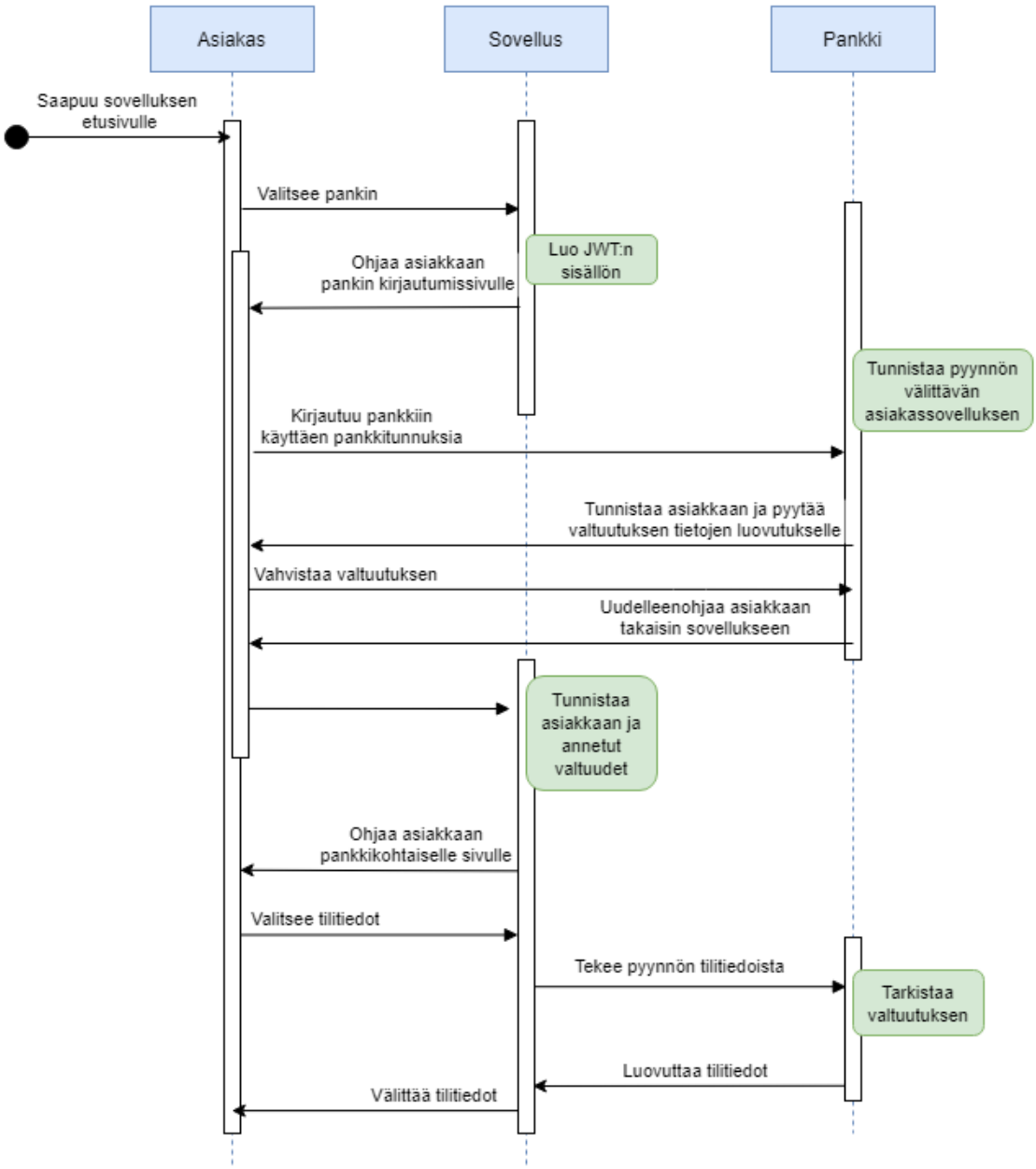
```
private fun buildSslContext(): SSLContext {
        val clientStore = KeyStore.getInstance("PKCS12")
        val resource =
          this.javaClass.classLoader.getResourceAsStream(config.opTppCert)
        clientStore.load(resource, config.opTppCertPassword.toCharArray())
        val sslContextBuilder = SSLContextBuilder()
        sslContextBuilder.setProtocol("TLSv1.2")
        sslContextBuilder.loadKeyMaterial(clientStore,
                config.opTppCertPassword.toCharArray())
        sslContextBuilder.loadTrustMaterial(TrustAllStrategy.INSTANCE)
        return sslContextBuilder.build()
}
```

Esimerkkikoodi 6. Varmenteen asettaminen HTTP-pyynnön tekevälle oliolle.

5.5 Asiakkaan tunnistautuminen ja JWT-tunnuksen luominen

Johtuen eri pankkien vaihtelevasta dokumentaation tasosta ja testiympäristön toteutustavasta esimerkkisovelluksessa käytettiin vain Osuuspankin testiympäristöä autentikointitunnuksen luomiseen. Osuuspankki tarjoaa kehittäjille Github-koodivarastossa Java-pohjaisen sovelluksen, jolla sovelluskehittäjät voivat rekisteröityä testiympäristöön kolmannen osapuolen maksupalvelun tuottajaksi. Sovellus luo käyttäjälle tarvittavat varmenteet PSD2-rajapinnan käyttöä varten sekä yksityisen salausavaimen JWT:n allekirjoittamista varten. Sovellukselle annetaan parametreina oma henkilökohtainen rajapinta-avain ja pankille annettu sovelluksen uudelleenohjausosoite, joka määriteltiin aiemmin kuvassa 7 näkyvällä tavalla.

Termiä uudelleenohjausosoite käytetään silloin, kun käyttäjä ohjataan johonkin muualle, kuin mihin alkuperäinen HTTP-pyyntö on tehty. Tässä tapauksessa käyttäjä ohjataan kahdesti uudelleen: ensimmäisen kerran, kun käyttäjä ohjataan esimerkkisovelluksesta pankin kirjautumissivulle, ja toisen kerran, kun kirjautuminen on onnistunut ja pankki ohjaa käyttäjän takaisin sovellukseen. Asiakassovelluksen ja pankin palvelimet eivät siis kommunikoi tässä kohtaa keskenään, vaan kaikki tieto siirtyy asiakkaan selaimen kautta käyttäen uudelleenohjausosoitetta kuvassa 9 näkyvällä tavalla. Osoitteen mukana välitetään kaikki tarvittava tieto parametreina, jotta osapuolet tunnistavat toisensa, kuten valtuutuskoodi, joka on kohdistettu tiettyyn asiakkaaseen. Seuraavaksi käydään läpi, miten tämä vaihe tehtiin esimerkkisovelluksen ja Osuuspankin välillä.



Kuva 9. Tunnistautumisprosessi asiakkaan, esimerkkisovelluksen ja pankin välillä

Esimerkkisovelluksessa JWT-tunnuksen luomiseen käytettiin Java-pohjaista Jwts-kirjastoa tunnuksen luomisen helpottamiseksi. Kirjasto tarjoaa muun muassa helpottavia metodeja standardoitujen vaatimusten luomiseen, mutta tärkeimpänä nopeuttavana tekijänä on allekirjoituksen toteutus. Kirjaston avulla esimerkkisovelluksen JwtGenerator-luokassa JWT:n luominen korkealla tasolla mahtuu muutamaan riviin koodia, kuten esimerkkikoodissa 7.

```
fun createJwt(authorizationId: String): String {
        val keyFactory = KeyFactory.getInstance("ECDSA")
        val pkcS8EncodedKeySpec =
           PKCS8EncodedKeySpec(loadPEM(config.opSigningKey))
        val privateKey = keyFactory.generatePrivate(pkcS8EncodedKeySpec)
        return Jwts.builder()
                .setHeaderParam("kid", config.opTppKid)
                .setHeaderParam("typ", "JWT")
                .setIssuedAt(Date(LocalDate.now()
                     .toEpochSecond(LocalTime.now(), ZoneOffset.UTC)))
                .setClaims(createClaims(authorizationId))
                .signWith(privateKey)
                .compact()
    }
```

Esimerkkikoodi 7. JWT:n luominen

Metodi saa parametrina String-tyyppisen parametrin authorizationId, joka on valtuutusprosessin alussa Osuuspankilta pyydetty, juuri kyseiselle sovellukselle myönnetty valtuutuskoodi, jota tarvitaan, kun JWT rakennetaan. Allekirjoitusta varten ensimmäisellä rivillä luodaan viittaus elliptisen käyrän digitaalisen allekirjoituksen algoritmia (ECDSA, Elliptic Curve Digital Signature Algorithm) käyttävään KeyFactory-tyyppiseen olioon.

Toisella rivillä luetaan muistiin tiedostosta Osuuspankin antaman yksityinen avain. Tiedoston lukemisen toteutusyksityiskohdat ovat loadPEM-nimisessä metodissa, joka saa parametrina viittauksen ohjelmakonfiguraatioon asetetun tiedostopolun yksityisen avaimen sisältävään tiedostoon. Kolmannella rivillä tästä yksityisestä avaimesta luodaan PrivateKey-tyyppinen olio.

Jwts-kirjasto suosii JWT:n luomisessa rakentajarajapintasuunnittelumallia. Rakentajaolioolle annettiin tarvittavat otsikkotietueet, kuten edellisessä koodiesimerkissä 7. Osuuspankin tapauksessa otsikkotietueet ovat kolmannen osapuolen yksilöivä tunnus “kid” ja sisällön tyyppi ”typ”. Käytettävää algoritmia ei tarvitse kirjata erikseen, vaan se lisätään olioon automaattisesti, kun olio allekirjoitetaan.

Vaatimusten luomiseen Jwts-kirjasto tarjoaa metodeita standardeille tietueille, kuten tunnuksen luomisajankohdan. Välitettävää tieto on suurimmaksi osaksi standardin määrittelemien otsaketietueiden ulkopuolista, ja sitä on niin paljon, että suurin osa vaatimuksista luodaan createClaims-metodissa, esimerkkikoodin 8 osoittamalla tavalla. Vaatimusten sisältö on määritelty Osuuspankin dokumentaatiossa [25]. Sisältö eroaa jonkun verran testi- ja tuotantoympäristöjen välillä. Esimerkki on tehty testiympäristön mukaisesti.

Vaatimukset sisältävät tiedon muun muassa pyydettävän valtuutuksen rajauksista (esimerkiksi tilitiedot), kuinka kauan valtuutus on voimassa ja sovelluksen yksilöivän tunnuskoodin. Metodi antaa paluuarvona hajautustaulun, jossa avaimen tyyppinä on String ja arvon tyyppinä Any. Arvon tyyppinä käytetään luokkaa Any, koska osa tietokentistä on merkkijonotyyppisiä, osa totuusarvoja. Kotlinissa jokaisen luokan yliluokka on Any, mikä mahdollistaa hajautustaulun luomisen, kun kaikki arvot eivät ole saman tyyppisiä. Metodi sisältää edelleen alimetodeja, joissa vaatimuksen luomista jatketaan, mutta ne ovat tekniseltä toteutukseltaan samankaltaisia.

```
private fun createClaims(authorizationId: String): Map<String, Any> {
    return mapOf("aud" to "https://mtls.apis.op.fi",
            "scope" to "openid accounts",
            "iss" to config.opTppId,
            "response_type" to "code id_token",
            "state" to generateState(),
            "redirect_uri" to config.opRedirectUrl,
            "max_age" to 86400,
            "client_id" to config.opApiKey,
            "claims" to nestedClaims(authorizationId))
}
```

Esimerkkikoodi 8. Vaatimusten luominen

JWT-olion allekirjoittamiseksi kutsutaan signWith-metodia, joka saa parametrina metodin alussa luodun yksityisen avaimen. Viimeiseksi kutsutaan compact-metodia, joka vastaa useimpien rakentajarajapintaa toteuttavien build-metodia, jossa paluuarvona saadaan luotava olio, tässä tapauksessa String-muotoinen, allekirjoitettu JWT-tunnus.

Luotu JWT liitetään osaksi käyttäjän uudelleenohjausosoitetta, osoiteparametrina. Kun asiakassovellus tarvitsee luvan käyttäjän tietoihin, luodaan kyseinen JWT ja ohjataan asiakas Osuuspankin kirjautumissivulle tätä uudelleenohjausosoitetta käyttäen. Taustalla Osuuspankki varmistaa pyynnön aitouden käyttäen JWT:n sisältämää tietoa. Asiakkaan kirjautumisen ja luovutettavien tietojen hyväksynnän jälkeen Osuuspankki ohjaa asiakkaan jälleen uudelleen, tällä kertaa asiakassovelluksen etukäteen määrittelemää uudelleenohjausosoitetta käyttäen, joka tekee pyynnön asiakassovelluksen palvelimelle. Uudelleenohjausosoite sisältää tiedon käytettävästä tunnuskoodista, jolla asiakkaan tietoihin pääsee käsiksi.

Käytännössä tämän toimivuutta ei kuitenkaan päässyt loppuun asti testaamaan, koska prosessi epäonnistui kirjautumissivulle ohjatessa ilmoitukseen allekirjoituksen

virheellisyydestä. Tämä tapahtuu siitä huolimatta, että itse allekirjoituksen toimivuuden pystyi itse todentamaan käyttämällä varmenteen julkista avainta.

Osa pankeista tarjoaa merkittävästi yksinkertaistetun asiakkaan tunnistamisvaiheen testiympäristössä testaamisen helpottamiseksi. Testiympäristössä kenenkään ei tarvitse kirjautua pankkitunnuksilla palveluun, vaan tämä vaihe toteutetaan keinotekoisesti pankin palvelimella. Asiakassovelluksen näkökulmasta tunnistautuminen toimii kuten oikeassa käyttötapauksessa eli asiakassovelluksen tehdessä valtuutuspyynnön, saa se vastauksena valtuutuskoodin, jonka kuviteltu asiakas on kirjautumisellaan antanut.

5.6 HTTP-viestin allekirjoittaminen

Nordean rajapinnoissa JWT:tä ei käytetä, mutta ne edellyttävät kaikkien rajapintapyyntöjen allekirjoittamista pyynnön tekijän identiteetin vahvistamiseksi. Allekirjoituksen sisältö eroaa hieman käytetyn HTTP-metodin perusteella. GET-pyyntöjä tehtäessä hajautusarvoa ei lasketa, vaan allekirjoitus tehdään otsaketietojen perusteella. Viesteissä, joissa viestin sisältö välitetään pyynnön vartalossa, kuten POST- ja PUT-pyynnöissä, lasketaan myös hajautusarvo, jota käytetään otsaketietojen lisäksi. Allekirjoituksen toteuttamisen apuna voi käyttää Tomitriben http-java-signatures-kirjastoa (https://github.com/tomitribe/http-signatures-java), kuten Nordea ehdottaa. Esimerkkikoodissa 9 määritellään POST-metodin vastaanottava päätepiste, joka edelleen lähettää pyynnön Nordean rajapintaan.

```
@PostMapping("/payments")
fun createPayment(@RequestBody paymentRequest: PaymentRequest): String {
    val sepaPaymentJson = createPaymentJson(paymentRequest)
    val request = NordeaApiHeaders(config)
        .createPOSTRequest(URI.create(paymentsUrl), sepaPaymentJson)

    val r = httpClient.send(request, HttpResponse.BodyHandlers.ofString())
    logger.log(Level.INFO, "payment initiation requested:  ${r.body()}")
    return r.body()
}
```

Esimerkkikoodi 9. Maksunluontipyynnön vastaanottava metodi

Ennen pyynnön edelleenlähettämistä täytyy pyynnön otsaketiedot asettaa määritellyllä tavalla, mikä tapahtuu NordeaApiHeaders-luokan createPOSTRequest-metodissa. Ensimmäiseksi pyynnön vartalosta lasketaan hajautusarvo RSA-SHA256-algoritmia

käyttäen. Apuna tässä käytetään Apachen DigestUtils-kirjastoa, jolla on yksinkertainen sha256-metodi, joka vastaanottaa tavutaulukon. HTTP-pyynnön JSON-merkkijonomuotoinen vartalo muunnetaan ensin tavutaulukoksi ja annetaan se metodille, joka palauttaa hajautusarvon uutena tavutaulukkona. Ennen hajautusarvon muuttamista merkkijonoksi, koodataan se määrittelyn mukaisesti vielä Base64-algoritmilla. Tästä syntynyt merkkijono on lopullinen hajautusarvo, joka asetetaan HTTP-pyynnön ”Digest”-otsakkeeseen.

Allekirjoitukselle täytyy määritellä käytettävät otsakkeet, joihin kuuluvat vastaanottavan palvelun palvelinosoite, luomisajankohta, sisältötyyppi ja vartalosta laskettu hajautusarvo. Allekirjoituksen otsakkeiden nimien täytyy täsmätä varsinaisen HTTP-pyynnön otsakkeita. Tomitriben http-java-signatures-kirjastolla luodaan allekirjoitus käyttäen parametreina verkko-osoitetta, Nordean rajapinta-avainta (keyId), RSA-SHA256-algortmia, käytettävää HTTP-metodia ja HTTP-pyynnön otsaketietoja esimerkkikoodi 10 osoittamalla tavalla. Allekirjoituksessa käytettävien otsakkeiden nimet annetaan listana Signature-olion konstruktorille. Niiden pitää olla määritellyssä järjestyksessä, koska ne liitetään osaksi merkkijonoa, josta allekirjoitus luodaan. Koska allekirjoituksen luomisessa käytetään osana myös vartalon hajautusarvoa, voidaan vastaanottavassa päässä myös viestin vartalon todeta olevan muuttumaton.

```
fun createInsertSignature(requestURI: URI, httpMethod: String, headers:
Map<String, String>): String {
   val path = getPath(requestURI)
   val signature = Signature(keyId, "rsa-sha256", null, INSERT_HEADERS)
   val signer = Signer(key, signature).sign(httpMethod, path,
   headers).toString()
   return stripSignature(signer)
}
```

Esimerkkikoodi 10. HTTP-pyynnön allekirjoitusosan luominen

5.7 Rajapintapyynnöt

Esimerkkisovelluksen palvelin sisältää pankkien rajapintoja vastaavat päätepisteet. Kun käyttäjä valitsee haluamansa pankin, tekee sovellus pyynnön pankin vastaavaan. Sovelluksessa käytetään myös samanlaista sanastoa kuin pankin tarjoamassa rajapinnassa, esimerkiksi ”/accounts”-osoite esimerkkisovelluksessa vastaanottaa osoiteparametrina tilin yksilöivän tilitunnuksen (AccountId) ja välittää sen vastaavanlaiseen osoitteeseen Handelsbankenin palvelimelle kuten esimerkkikoodissa 11.

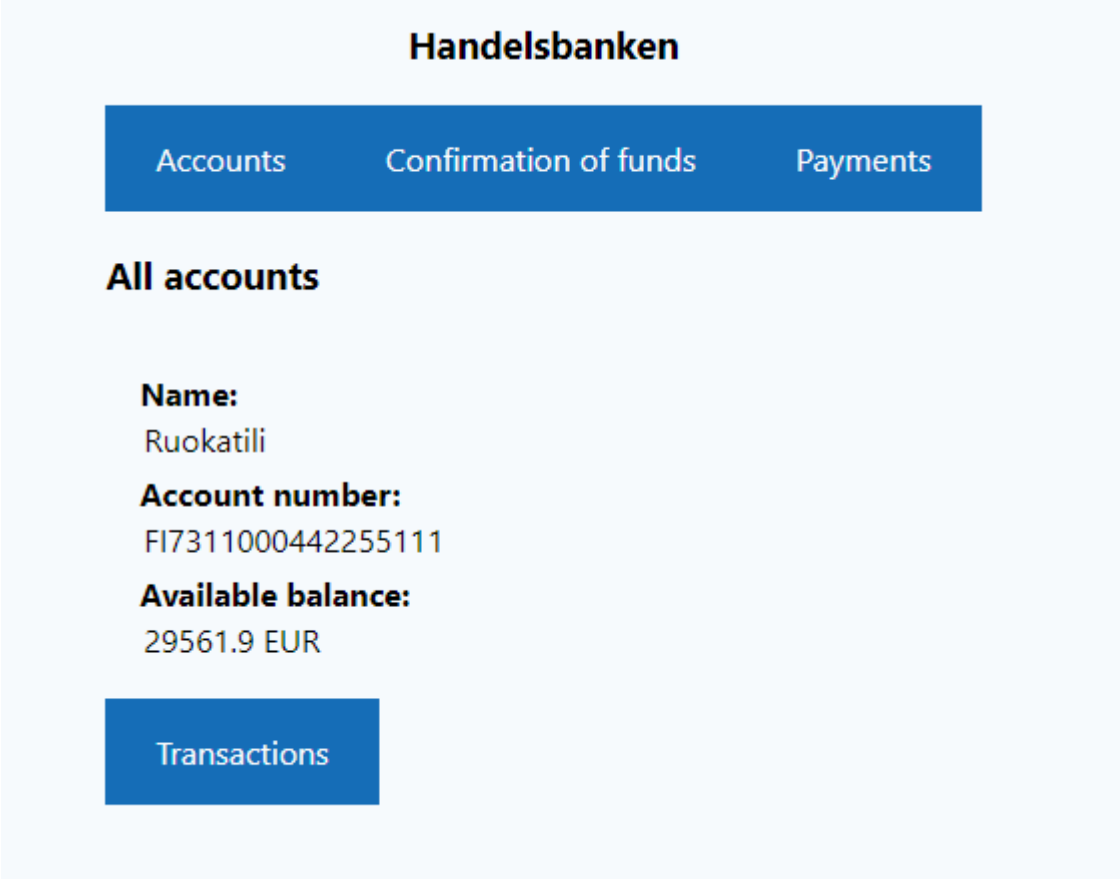
```
@GetMapping("/accounts/{accountId}", produces =
      [MediaType.APPLICATION_JSON_VALUE])
fun account(@PathVariable accountId: String): String {
        val requestBuilder = HttpRequest.newBuilder()
             .GET()
             .uri(URI.create("https://sandbox.handelsbanken.com/openbank
                 ing/psd2/v2/accounts/${accountId}?withBalance=true"))
             .setHeader("X-IBM-Client-Id", config.handelsbankenApiKey)
             .setHeader("Authorization", authorizationKey)
             .setHeader("accept", "application/json")

        val response = httpClient.send(requestBuilder.build(),
                           HttpResponse.BodyHandlers.ofString())
        logger.log(Level.INFO, "Handelsbanken account $accountId requested")
        return response.body()
```

Esimerkkikoodi 11. Handelsbankenin tilitietopyynnön määrittelyesimerkki sovelluksessa

Myös muut päätepisteet vastaavat samaa mallia. Koska turvallisuuteen liittyvät määrittelyt on jo tehty, on rajapintapyynnön tekeminen tässä kohtaa yksinkertaista. Pyyntöön vain liitetään pakolliset otsaketiedot, jotka on määritelty Handelsbankenin dokumentaatiossa. Koodissa HTTP-pyyntöön käytettävä httpClient on jo ennalta määritelty käyttämään varmennetta tietoa pyydettäessä. Tässä tapauksessa käyttäjän yksilöivä valtuutuskoodi (authorizationKey) on staattinen, aina samana pysyvä testivaltuutuskoodi ja varsinaista käyttäjän sisäänkirjautumista ei tarvita. Handelsbanken vastaa pyyntöön JSON-objektilla kuvan 10 mukaisesti. Esimerkkisovellus välittää vastauksen käyttöliittymälle. Kuvassa 11 osa näistä vielä sovelluksessa esitettynä.


```
{
  "accountId": "ae50bb90-6cf3-11e9-9c41-e957ce7d7d69",
  "iban": "FI7311000442255111",
  "currency": "EUR",
  "bic": "HANDFIHH",
  "ownerName": "Anni Kanerva",
  "balances": [
    {
      "balanceType": "AVAILABLE_AMOUNT",
      "amount": {
        "currency": "EUR",
        "content": 29561.9
      }
    }
  ],
  "_links": {
    "transactions": {
      "href": "https://sandbox.handelsbanken.com/openbanking/psd2/v2/accounts/ae50bb90-6cf3-11e9-9c41-e957ce7d7d69/transactions"
    }
  }
}
```


Kuva 10. Handelsbankin tilitietopyynnön vastaus JSON-muodossa.

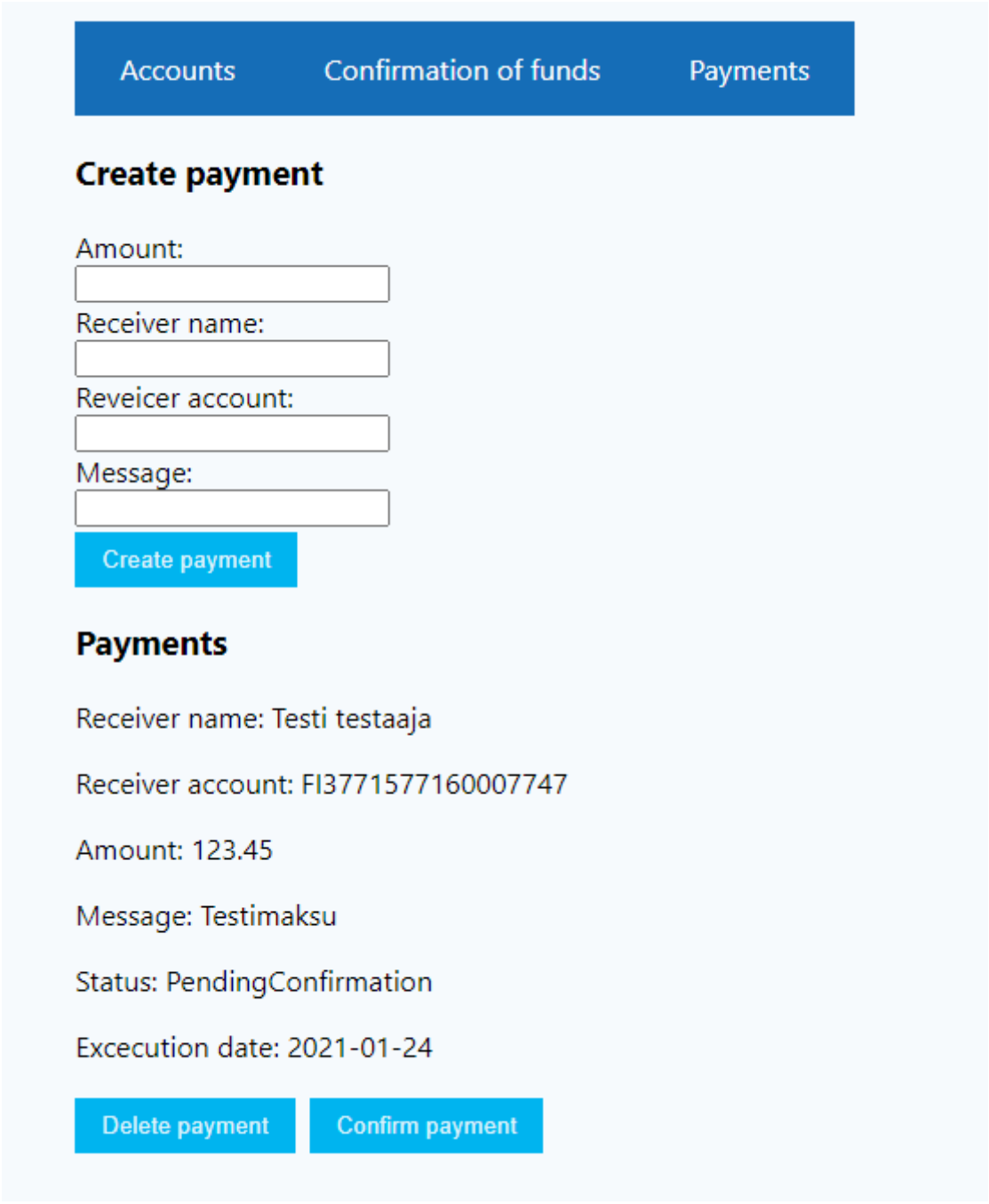


Kuva 11. Handelsbankenin tilitiedot sovelluksessa

Maksupalvelurajapinnat ovat vuorovaikutteisempia ja maksuja pystyy itse luomaan, vahvistamaan ja poistamaan. Aikaisemmin esimerkkikoodissa 9 näkyi maksunluonti Nordean rajapintaan. Esimerkkikoodissa 12 näkyy maksun vahvistaminen, johon käytetään PUT-metodia. Koska maksun tiedot vastaanotetaan osana HTTP-pyynnön vartaloa, on funktion parametriin lisätty @RequestBody-annotaatio, jolloin Spring Boot osaa automaattisesti luoda parametreistä ConfirmRequest-objektin. Tämä edellytyksenä on, että ConfirmRequest-luokkaan on määritelty HTTP-pyynnön vartalon parametrien kanssa saman nimiset kentät. Kuvassa 12 nähdään vielä sovelluksessa luotu maksu, jonka pystyy poistamaan tai vahvistamaan.

```
@PutMapping("/remove/payments/confirm")
    fun confirmPayments(@RequestBody payments_ids: ConfirmRequest): String {
        val objectMapper = ObjectMapper()
        val paymentIdsJson = objectMapper.writeValueAsString(payments_ids)

        val requestBuilder = HttpRequest.newBuilder()
            .PUT(HttpRequest.BodyPublishers.ofString(paymentIdsJson))
            .uri(URI.create("${paymentsUrl}/confirm"))
            .setHeader("Authorization",
     "Bearer ${NordeaAuthController(config).getAccessToken().access_token}")
            .setHeader("Content-Type", "application/json; charset=UTF-8")
            .setHeader("Accept", "application/json")

        val request = NordeaApiHeaders(config)
                .setToUnsigned(requestBuilder).build()
        val r = httpClient.send(request, HttpResponse.BodyHandlers.ofString())
        logger.log(Level.INFO, r.body())
        return r.body()
    }
```

Esimerkkikoodi 12. Maksun vahvistaminen Nordean rajapinnassa



Kuva 12. Nordean rajapintaan luotu testimaksu.

5.8 Jatkokehitys

Esimerkkisovelluksen suurin hyöty on havainnollistaa PSD2-rajapintojen vaatimien tunnistautumismekanismien toteutus ja näyttää, mitä tietoa ja toiminnallisuutta PSD2-rajapinnat tarjoavat. Sen hyödyllisyyttä oikeaan käyttöön kasvattaisi, jos sovelluksessa olisi mukana myös tapa tallentaa pysyvästi tietoa tietokantaan. Tällaisenaan sovellus antaa puutteellisen kuvan koko prosessista, koska käyttäjän myöntämät valtuudet täytyy joka käyttökerralla kysyä uudestaan, mikä ei ole todellisten käyttötapojen mukaista eikä käyttäjäystävällistä. Rajapintoja käyttävän sovelluksen tulisi myös pystyä turvallisesti tunnistamaan omat käyttäjät ja yhdistämään ne annettuihin valtuuksiin. Sovellus ei myöskään sellaisenaan toisi lisäarvoa kuluttajille, vaan sen pitäisi olla osa suurempaa palvelukokonaisuutta, kuten osa taloudenhallintosovellusta, jossa saatavia tietoja voisi hyödyntää.

Tunnistautumisprosessin toteutusta voisi jatkaa useammalle pankille, koska se on merkittävästi työläin osa rajapinnan käyttöönotossa. Samankaltaisten rajapintojen käytön toteutus useammalle eri pankille ei itsessään ole välttämättä kuitenkaan kovin arvokasta, koska teknisesti ne ovat hyvin lähellä toisiaan.

Koska pääsy PSD2-rajapintoihin on viranomaisvalvonnan alla eivätkä tavalliset kehittäjät pääse oikeaan tietoon käsiksi, voisi luoda palvelun, joka toimisi välikätenä muille kehittäjille. Palvelu pystyisi tarjoamaan ohjelmistokehittäjille ohjelmallisen rajapinnan, joka antaisi pääsyn heidän omiin tietoihinsa. Tämän rajapinnan päälle voisivat he toteuttaa uusia sovelluksia henkilökohtaiseen käyttöön. Koska toimiluvan voimassa pitäminen ei ole ilmaista, tarvitsisi käyttäjiltä periä jonkinlaista pientä maksua palvelusta. Todennäköisesti tällainen palvelu voisi tosin perustua vain tiedon hakemiseen, ei sen muuttamiseen eli maksupalveluita ei voisi tarjota. On myös kyseenalaista, menisikö toimilupahakemus tällä idealla läpi.

Kaupallisemmaksi samaa ajatusta voisi myös laajentaa siten, että tarjoaisi yhtenäisemmän rajapinnan useammalle eri pankille. Tunnistamismekanismin yksinkertaistaminen voisi olla asiakasyrityksille arvokasta, sillä juuri siinä eri pankkien toteutukset eroavat suuresti. Samalla käytettävien rajapintojen pienet eroavaisuudet voisi poistaa kokonaan siten, että asiakassovelluksen ei tarvitsisi niitä huomioida. Tätä varten tosin on jo olemassa muita toimijoita, kuten Open Banking Aggregation SDK.

6 Yhteenveto

Vuonna 2018 voimaan astunut PSD2-direktiivi, eli uudistettu maksupalveludirektiivi on merkittävä uudistus tehden ennen saavuttamattomasta tai vähintäänkin maksullisesta tiedosta kohtalaisen helposti saatavaa ja ilmaista. Rajapinnat edesauttavat uuden liiketoiminnan syntyä ja parantavat kuluttajien asemaa pankkimarkkinoilla. Myös isot, jo vakiintuneessa asemassa olevat pankit kuten Osuuspankki ja Danske Bank ovat hyödyntäneet uusia rajapintoja tarjoten mahdollisuutta nähdä tilitietoja muilta pankeilta oman sovelluksen sisällä. Tilitietopalveluista olisi hyötyä taloushallinto- ja kirjanpitosovelluksille. Sovelluksia käyttävillä yrityksillä on tarve tietää reaaliaikaisesti kulloinkin tilillä oleva tarkka saldo maksaessaan laskuja tai palkkoja. Se, että rajapinnat ovat tarjolla ilmaiseksi, helpottaa palvelun tuomista asiakkailleen.

Tässä työssä käytiin läpi, mitä tietoa rajapinnoista on saatavilla ja mitä niiden käyttö teknisesti vaatii. Tekniset vaatimukset eri osapuolten tunnistamiseen olivat vaihtelevia sekä haastavia toteuttaa. Työn tuloksena syntyi esimerkkisovellus, jossa näitä tekniikoita oli toteutettu ja joissa havainnollistettiin tunnistautumisen toteutusta sekä rajapintojen tietosisältöä. Pankkien tarjoamat rajapinnat ovat pakollisten osuuksien osalta hyvin samankaltaiset, mikä osaltaan helpottaa useamman pankin tukemista kolmannen osapuolen sovelluksessa.

Rajapintojen käyttämisen aloittaminen ei ole kuitenkaan yksinkertaista, vaan vaatii monimutkaisen tunnistautumisrakenteen alleen, jotta rajapintoja voi käyttää. Nämä rakenteet täytyy käytännössä rakentaa käsin erikseen jokaiselle pankille. Pankit (tai heidän käyttämät rajapintapalvelut) tarjoavat vaihtelevan tasoista dokumentaatiota, mikä osaltaan vaikuttaa siihen, kuinka helppoa rajapinnan käyttöönotto on. Asiakassovelluksen tunnistamisvaiheessa olivat myös suurimmat erot eri pankkien välillä, eikä toteutuksen tekeminen yhdelle pankille helpota toteutuksen tekemistä toiselle.

Lähteet

1 Euroopan komissio. 2013. Commission staff working document. Verkkoaineisto. Euroopan komissio. <https://eur-lex.europa.eu/resource.html?uri=cellar:906ed6d3-f509-11e2-a22e-01aa75ed71a1.0001.04/DOC_1&format=PDF>. Luettu 17.1.2021.

2 Finanssivalvonta. 2019. PSD2. Verkkoaineisto. Finanssivalvonta. <https://www.finanssivalvonta.fi/saantely/saantelykokonaisuudet/psd2>. Päivitetty 23.3.2019. Luettu 15.12.2020.

3 Finanssivalvonta. 2018. PSD2 – Ajankohtaiskatsaus. Verkkoaineisto. Finanssivalvonta. <https://www.suomenpankki.fi/globalassets/fi/raha-ja-maksaminen/maksujarjestelmat/suomen-pankki-katalystina-maksuneuvosto/mn10_psd2_ajankohtaiskatsaus.pdf>. Luettu 28.12.2020.

4 Open Banking Market. Verkkoaineisto. Crosskey Banking Solutions. <https://crosskey.io>. Luettu 17.1.2021.

5 OP avaa monipankkipalvelun taloudenhallinnan sujuvoittamiseksi. 2020. Lehdistötiedote. Osuuspankki. <https://www.op.fi/op-ryhma/medialle/tiedotteet?id=3685019_PRC>. Luettu 17.1.2021.

6 Arminen, Tuuli. 2018. Toisen maksupalveludirektiivin (PSD2) vaikutukset kuluttajille. Opinnäytetyö. Metropolia Ammattikorkeakoulu. Theseus-tietokanta.

7 Euroopan komissio. 2018. Payment Services Directive: frequently asked questions. Verkkoaineisto. Euroopan komissio. <https://ec.europa.eu/commission/presscorner/detail/fr/MEMO_15_5793>. Luettu 12.12.2021.

8 Ballad, Bill; Ballad, Tricia & Banks, Erin. 2010. Access Control, Authentication, and Public Key Infrastructure. E-Kirja. Jones & Bartlett Learning.

9 Euroopan parlamentin ja neuvoston asetus. 2014. 910/2014. 23.7.2014.

10 Sähköinen allekirjoitus ja muut eIDAS-palvelut. 2020. Verkkoaineisto. Kyberturvallisuuskeskus. <https://www.kyberturvallisuuskeskus.fi/fi/toimintamme/saantely-ja-valvonta/sahkoinen-allekirjoitus-ja-muut-eidas-palvelut>. Päivitetty 9.7.2020. Luettu 17.1.2021.

11 API Evaluation Group. 2018. eIDAS and TPP Identification (PSD2). Verkkoaineisto. API Evaluation Group. <https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2018-11/API%20EG%20058-18%20v1.0%20eIDAS%20and%20TPP%20identification%20%28PSD2%29.pdf>. Luettu 17.1.2021.

12 WP-API. 2015. WP REST API - OAUTH 1.0a Server. Verkkoaineisto. <https://oauth1.wp-api.org/docs/introduction/OAuth-1.html>. Päivitetty 9.3.2017. Luettu 17.1.2021.

13 Internet Engineering Task Force. 2012. The OAuth 2.0 Authorization Framework. Verkkoaineisto. <https://tools.ietf.org/html/rfc6749>. Luettu 11.11.2021.

14 Richer, Justin & Sanso, Antionio. 2017. OAuth 2 in Action. E-kirja. Manning Publications Co.

15 Internet Engineering Task Force. 2015. JSON Web Token (JWT). Verkkoaineisto. <https://tools.ietf.org/html/rfc7519>. Luettu 12.11.2021.

16 Network Working Group. 2018. Signing HTTP Messages. draft/cagage/http/signatures-10. Verkkoaineisto.<https://tools.ietf.org/html/draft-cavage-http-signatures-10>. Luettu 2.1.2021.

17 Krill, Paul. 2011. JetBrains readies JVM/based language. Verkkoaineisto. Infoworld. <https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html>. Luettu 19.12.2020.

18 Developer Survey Results. 2018. Verkkoaineisto. Stack Overflow. <https://insights.stackoverflow.com/survey/2018#technology-_-programming-scripting-and-markup-languages>. Luettu 25.12.2020.

19 Developer Survey Results. 2019. Verkkoaineisto. Stack Overflow. <https://insights.stackoverflow.com/survey/2019#technology-_-programming-scripting-and-markup-languages>. Luettu 25.12.2020.

20 Developer Survey Results. 2020. Verkkoaineisto. Stack Overflow. <https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers>. Luettu 25.12.2020.

21 Android's Kotlin-first approach. 2019. Verkkoaineisto. Google Developers. <https://developer.android.com/kotlin/first>. Päivitetty 22.1.2021. Luettu 25.12.2020.

22 Learn Kotlin. 2020. Verkkoaineisto. Kotlin Foundation. <https://kotlinlang.org/docs/reference/java-interop.html>. Luettu 25.12.2020.

23 React. A JavaScript library for building user interfaces. Verkkoaineisto. Facebook Inc. <https://reactjs.org>. Luettu 25.12.2020.

24 Gamma, Erich; Helm, Richard; Johnson, Ralph & Vlissides, John. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. E-kirja. Addison-Wesley Professional.

25 Workflow for OP PSD2 Account Information Service API. Verkkoaineisto. Osuuspankki. <https://op-developer.fi/p/accountauthorizationflow>. Luettu 17.1.2021.