Opinnäytetyö (AMK)

Tietojenkäsittely

Tietojärjestelmät

2013

Janne Rytkönen

KULUNVALVONNAN TOTEUTUS TEHDEN-OHJELMISTOON

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Tietojärjestelmät

Marraskuu 2013 | 44 sivua

Ohjaaja Anne Jumppanen

Janne Rytkönen

KULUNVALVONNAN TOTEUTUS TEHDEN-OHJELMISTOON

Opinnäytetyön tavoitteena on toteuttaa kulunvalvontakokonaisuus, Tehden-kulunvalvonta, joka toimii osana Tehden-toiminnanohjausjärjestelmää. Tehden-ohjelmisto koostuu useasta eri moduulista. Toteutettava kokonaisuus on osa Tehden-ohjelmiston kulunvalvontamoduulia. Opinnäytetyön toimeksiantajana toimii ohjelmistoalalla toimiva Tehden Oy.

Tehden-kulunvalvonta koostuu pääosin Python-ohjelmointikielellä toteutettavasta kulunvalvontasovelluksesta sekä osasta PHP:llä toteutettavasta Tehden-ohjelmistosta ja sen rajapinnasta, Tehden APIsta. Kokonaisuuteen kuuluu myös korttien lukemisesta huolehtiva NFC-lukija. Kulunvalvonnan tapahtumien tallentamiseen käytetään SQLite-relaatiotietokantaa. Lisäksi kokonaisuuteen kuuluu vielä Python-ohjelmointikielellä toteutettava tietoikkuna, jonka tehtävänä on ilmoittaa yrityksen henkilökunnalle kulunvalvontatapahtumien tietoja.

Tehden-kulunvalvonnan tehtävänä on huolehtia yritysten asiakkaiden kulunvalvonnasta. Asiakkaiden ja kulunvalvonnan välinen kommunikointi tapahtuu NFC-lukijan avulla. Kulunvalvonnan aikana sovellus ottaa yhteyttä Tehden API -rajapintaan, jossa suoritetaan tietojen varsinaiset tarkistukset. Rajapinta lähettää vastauksen sovellukselle, joka välittää tiedot lukijalle ja tietoikkunalle.

Opinnäytetyön tuloksena syntynyt kulunvalvontakokonaisuus on opinnäytetyön kirjoitushetkellä jo käytössä muutamalla yrityksellä. Se onkin hyvä pohja mahdollisille tulevaisuuden toteutuksille, joista yksi potentiaalinen kehitysajatus on ollut kosketusnäytöllä toimiva kulunvalvonta. Tällöin esimerkiksi asiakkaat voisivat tunnistautua omalla sormenjäljellään, eikä erillistä kulkukorttia enää tarvittaisi.

ASIASANAT:

Etätunnistus, Kulunvalvonta, PHP, Python, Rajapinta, SQLite, Tehden, Todentaminen

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Information Systems

November 2013 | 44 pages

Instructor Anne Jumppanen

Janne Rytkönen

ACCESS CONTROL IMPLEMENTATION TO TEHDEN SOFTWARE

The aim of the present bachelor's thesis is to implement an access control system named as Tehden access control. This access control system operates as a part of Tehden ERP system. Tehden software consists of several modules. The implemented Tehden access control will be a part of the Tehden software access control module. The thesis was commissioned by Tehden Oy which operates in the software industry.

Tehden access control consists mainly of the access control application and of a part of Tehden software and its interface Tehden API. The access control application is programmed with Python and Tehden and Tehden API parts are programmed with PHP. In addition, a part of the access control system is NFC card reader. The access control system uses its own database which is implemented with SQLite relation database. The access control system also includes a so called info window that shows the company's personnel information about access control events. The info window is also programmed with Python.

Tehden access control is responsible for the access control of the companies' customers. The communication between customers and the access control system is done by the NFC card reader. During the access control monitoring the application is connected to Tehden API which performs the actual data checks. Tehden API interface will then send a response to the access control application which forwards the response to the NFC card reader and the info window.

The result of the study, Tehden access control, is at the time of writing already used by a few companies. It is a good basis for the possible future implementations. One of the potential development ideas is to create a touch screen based access control system. In this case, for example, customers could authenticate by their own fingerprint and a separate access card would no longer be needed.

KEYWORDS:

Access control, Authentication, Interface, PHP, Python, Remote identification, SQLite, Tehden

SISÄLTÖ

KÄYTETYT LYHENTEET JA SANASTO **7**

1 JOHDANTO **8**

1.1 Tavoite 8

1.2 Kirjallisen työn eteneminen 9

2 TEHDEN **11**

2.1 Tehden Oy 11

2.2 Tehden-ohjelmisto 11

3 KULUNVALVONTA **13**

3.1 Määrittely 13

3.2 Komponentit 13

3.3 Prosessi 16

4 NEAR FIELD COMMUNICATION **18**

4.1 Teknologia 18

4.2 APDU 19

5 OAUTH 2.0 **20**

5.1 Roolit 20

5.2 Todentamisprosessi 20

5.3 Käyttöoikeustietueen päivittäminen 21

6 TEHDEN-KULUNVALVONNAN TOTEUTUSTEKNIIKAT **22**

6.1 Python 22

6.2 Pyscard 22

6.3 PHP: Hypertext Preprocessor 23

6.4 SQLite 23

7 TEHDEN-KULUNVALVONNAN RAKENNE **24**

7.1 Tehden-ohjelmisto 24

7.2 Kulunvalvontasovellus 24

7.3 Tietoikkuna 25

7.4 Tietokanta 26

7.5 Tehden API 27
7.6 NFC-lukija 27

8 TEHDEN-KULUNVALVONNAN TOIMINTA 29
8.1 Kulunvalvonnan asennus 29
8.2 Kulunvalvonnan käynnistys 31
8.3 Kulunvalvonnan liittäminen Tehden-ohjelmistoon 33
8.4 Korttien lukeminen 37
8.5 Kortin liittäminen asiakkaalle 40

9 YHTEENVETO 42

LÄHTEET 44

LIITTEET

Liite 1. Accesscontrolcore-moduulin kortinluku-funktio.
Liite 2. Accesscontrolservice-moduuli.
Liite 3. Api-moduuli.
Liite 4. Cardreader-moduulin alustusfunktiot.
Liite 5. Database-moduulin alustusfunktiot.
Liite 6. Httpserver-moduulin GET-pyynnön käsittelyfunktio.
Liite 7. Oaclient-moduuli.
Liite 8. Infowindow-moduulin yhteys- ja viestifunktiot.
Liite 9. Tehden Accesscontrol API Minimodelin kortinluku-funktio.

KUVAT

Kuva 1. Tehden-kulunvalvonnan tietoikkuna. 26
Kuva 2. ACR1222L USB NFC-lukija (Advanced Card Systems Ltd, 2013). 28
Kuva 3. NFC-lukijan ilmoitus aikaperusteisen kortin käytöstä. 28
Kuva 4. Kulunvalvontasovelluksen asennuskomento. 30
Kuva 5. Windowsin Palvelut -ikkuna. 30
Kuva 6. NFC-lukijan yhteysvirheilmoitus. 35
Kuva 7. NFC-lukijan liittämisen hyväksyminen. 35
Kuva 8. Ilmoitus, kun asiakkaan kortin tietoja ei löytynyt. 39
Kuva 9. Kappaleperusteisen kortin ilmoitus. 40
Kuva 10. Aikaperusteisen kortin ilmoitus. 40
Kuva 11. Asiakkaan tietokortti. 40

KUVIOT

Kuvio 1. Kulunvalvonnan käynnistys. 32
Kuvio 2. Kulunvalvonnan liittäminen. 34
Kuvio 3. Kortin lukeminen. 38
Kuvio 4. Kortin liittäminen asiakkaalle. 41

TAULUKOT

Taulukko 1. Logevent-tietokantataulu. 26
Taulukko 2. Tokensettings-tietokantataulu. 27

KÄYTETYT LYHENTEET JA SANASTO

APDU	Viestintäyksikkö NFC-lukijan ja kortin välillä.
NFC	Langattoman viestinnän tekniikka, joka mahdollistaa tietojen vaihdon kahden laitteen välillä, jotka ovat lähietäisyydellä toisistaan.
Oauth 2.0	Käyttäjän todentamiseen käytettävä ohjelmistokehys.
PHP	PHP: Hypertext Preprocessor, ohjelmointikieli.
Pyscard	Python-moduuli, jonka avulla voidaan lähettää APDU-komentoja.
Python	Ohjelmointikieli.
SQLite	Sulautettu relaatiotietokantajärjestelmä.
Tehden API	Rajapinta, jonka avulla voidaan käyttää ja siirtää tietoja Tehden-ohjelmistosta.
Tehden-ohjelmisto	Web-toiminnanohjausjärjestelmä.

1 JOHDANTO

Kulunvalvonta on tärkeä asia sekä pienessä että suuressa yrityksessä. Kulunvalvonnasta onkin tullut jo arkipäivää monessa yrityksessä. Kulunvalvonta helpottaa niin yrityksen työntekijöiden kuin asiakkaiden arkea. Lisäksi kulunvalvonta vähentää yrityksen kuluja ja kulunvalvonnasta huolehtivien ajankäyttöä. Kulunvalvonnan avulla voidaan myös helpottaa asiakasseurantaa ja täten tehostaa yrityksen toimintaa.

Tehden-ohjelmisto on hyvinvointialoille suunnattu ja pilvipalveluna tarjottava web-toiminnanohjausjärjestelmä. Ohjelmisto koostuu moduuleista, joista yhtenä moduulina toimii kulunvalvontamoduuli. Kulunvalvonnan käyttö on suunnattu erityisesti kuntosaleille ja hyvinvointikeskuksille. Kulunvalvonnassa jokaisella asiakkaalla on oma henkilökohtainen kulkukorttinsa. Asiakkaan kulkukortti on liitetty asiakastiliin, joka sisältää asiakkaalle myytyjä tilituotteita. Tilituotteita voivat olla esimerkiksi yhden kuukauden jäsenyys.

1.1 Tavoite

Tämän opinnäytetyön tavoitteena on toteuttaa kulunvalvontakokonaisuus nimeltään Tehden-kulunvalvonta, joka toimii osana Tehden-ohjelmiston kulunvalvontamoduulia. Kulunvalvontakokonaisuuden tehtävänä on huolehtia yritysten asiakkaiden kulunvalvonnasta.

Kulkukorttien lukemiseen käytetään USB:llä liitettävää kortinlukijaa, joka toimii NFC-teknologialla. Kortinlukija on kontaktiton lukija, joten se lukee kulkukortteja lyhyen etäisyyden päästä. Asiakkaiden kulkukortteina käytetään kortteja, joihin on asetettu ns. NFC-tagit. NFC-tagit voivat olla kiinteästi kortissa kiinni tai ne voivat olla erillisiä tarroja, jotka kiinnitetään korttiin. NFC-tagien avulla NFC-lukija pystyy lukemaan kulkukortteja. Asiakkaat voivat itsepalveluna leimata itsensä saapuneeksi näyttämällä korttinsa kortinlukijalle. Leimauksen aikana asiakkaan kortin asiakastililtä veloitetaan tarvittava määrä tilin saldosta.

Kulunvalvontasovellus toteutetaan Python-ohjelmointikielellä. Sovellus on tietyin väliajoin ja tietyissä eri tilanteissa yhteydessä NFC-lukijan, Tehden-ohjelmiston, Tehden API -rajapinnan ja tietoikkunan kanssa. Keskustelu kulunvalvontasovelluksen ja NFC-lukijan kanssa tapahtuu Pyscard-nimisen Python-moduulin kautta. Sovellus käyttää omaa SQLite-tietokantaa, joka sisältää tiedot kulunvalvontatapahtumista sekä Tehden API -rajapintaan tarvittavista käyttöoikeustietueista. Korttien lukemisen aikana tapahtuva keskustelu kulunvalvontasovelluksen ja Tehden APIn välillä tapahtuu Internet-yhteyden välityksellä.

Kulunvalvontasovelluksen kanssa rinnakkain toimiva tietoikkuna on tarkoitettu yrityksen henkilökunnalle. Tietoikkunan tarkoituksena on ilmoittaa kassahenkilölle asiakkaiden kulkukorttien tietoja kulunvalvontatapahtuman yhteydessä. Joka kerta kun NFC-lukija lukee kortin, kulunvalvontasovellus lähettää Tehden APIn lähettämät vastaukset eteenpäin tietoikkunalle, joka vuorostaan näyttää vastauksen sisältämät tiedot kassahenkilölle. Lisäksi tietoikkunan avulla voidaan tarkastella viimeisimpiä kulunvalvontatapahtumien tietoja.

1.2 Kirjallisen työn eteneminen

Kirjallinen työ voidaan jakaa kahteen osaan: teoriaan ja empiiriseen. Työn empiirinen osa käsittelee kulunvalvontakokonaisuuden toteutustekniikoita, rakennetta ja sen toiminnallisuutta.

Toisessa luvussa esitellään opinnäytetyön toimeksiantajana toiminut Tehden Oy. Tämän lisäksi luvussa esitellään Tehden-ohjelmisto, jonka kulunvalvontamoduuli toimii osana opinnäytteen tuloksena syntyvää kulunvalvonnan kokonaisuutta.

Kolmas luku käsittelee kulunvalvonnan teoriaa. Luvussa kerrotaan mitä kulunvalvonta oikeastaan on, keitä osapuolia kulunvalvonnan prosessiin osallistuu ja kuinka kulunvalvonnan prosessi etenee.

Neljännen luvun aiheena on Near Field Communication eli NFC. Luku käsittelee NFC:tä yleisellä tasolla sekä tarkemmin kuinka NFC-teknologia toimii. Lisäksi

luku käsittelee myös NFC-lukijan ja kortin välillä tapahtuvaa kommunikointia eli APDU-komentoja.

Viidennessä luvussa käsitellään todentamiseen käytettävää ohjelmistokehystä OAuth 2.0:aa. Luvussa esitellään kuinka ohjelmistokehys toimii ja kuinka todennusprosessi käytännössä tapahtuu. OAuth 2.0 -ohjelmistokehystä käytetään pohjana Tehden APIn tunnistautumisvaiheessa.

Luvut 6, 7 ja 8 käsittelevät opinnäytetyön empiiristä osuutta. Kuudes luku kertoo mitä tekniikoita Tehden-kulunvalvonnan toteuttamisessa käytettiin. Seitsemäs luku käsittelee Tehden-kulunvalvonnan rakennetta. Kahdeksannessa luvussa kerrotaan Tehden-kulunvalvonnan toiminnallisuudesta.

Viimeisessä luvussa tehdään yhteenveto, jossa tarkastellaan opinnäytetyön onnistumista. Lisäksi luvussa kerrotaan Tehden-kulunvalvonnan mahdollisesta tulevaisuuden kehityksestä.

2 TEHDEN

2.1 Tehden Oy

Tehden Oy on vuonna 2007 perustettu ohjelmistoalan yritys, joka suunnittelee, valmistaa ja ylläpitää toiminnanohjausjärjestelmiä. Yrityksen toiminta aloitettiin nimellä e-ngine Oy, mutta 1.2.2013 lähtien yrityksen virallinen nimi on ollut Tehden Oy. Yrityksen nimi vaihdettiin, koska sen tunnetuin tuote on ohjelmisto nimeltä Tehden. (Tehden Oy 2013.)

Vuosien 2007–2011 aikana Tehden Oy valmisti toiminnanohjausjärjestelmiä eri yrityksille mittatilaustyönä. Vuonna 2011 yritys aloitti Tehden-pilvipalvelutuotteen kehityksen. (Tehden Oy 2013.)

2.2 Tehden-ohjelmisto

Tehden Oy:n uusin tuote, pilvipalveluna tarjottava Tehden-ohjelmisto, tuli myyntiin joulukuussa 2012. Tehden-ohjelmisto on web-toiminnanohjausjärjestelmä, johon kuuluu tällä hetkellä kassaohjelma-, Internet-ajanvaraus-, tuotehallinta-, asiakashallinta-, raportointi- sekä kulunvalvontamoduuli, johon liitetään tämän opinnäytetyön tuloksena syntynyt kulunvalvontakokonaisuus. Ohjelmistoa on tarkoitus laajentaa vuoden 2013 aikana, jolloin mm. varasto- sekä tilausohjelmat lisätään osaksi ohjelmistoa. (Tehden Oy 2013.)

Tehden-ohjelmisto on tarkoitettu erityisesti hyvinvointialojen yrityksille, mutta sitä voivat käyttää myös muut yritykset, jotka tarvitsevat ohjelman tarjoamia ominaisuuksia. Yritys voi itse päättää, mitä osia ohjelmistosta se haluaa ottaa käyttöön. (Tehden Oy 2013.)

Tehden-ohjelmisto on pilvipalvelu eli se toimii Internet-yhteyden kautta. Ohjelmistoa käytetään selaimella. Tällaista ohjelmistokokonaisuutta ei ole aiemmin ollut markkinoilla. Pilvipalveluna toimiva ohjelmisto tuo monia etuja asennettavaan ohjelmaan verrattuna. Koska Tehden-ohjelmistoa käytetään selaimella,

ohjelmisto ei vaadi erillistä asennusta ja sitä voidaan käyttää milloin tahansa ja missä tahansa. Lisäksi käyttäjän ei tarvitse itse huolehtia ohjelman päivittämisestä. Ohjelmiston päivittäminen tarvitsee tehdä vain Tehden Oy:n palvelimille, jolloin uusin versio ohjelmistosta on kaikkien yritysten käytettävissä. (Tehden Oy 2013.)

3 KULUNVALVONTA

3.1 Määrittely

Nykypäivänä kulunvalvontaa on kaikkialla ympärillämme. Kohtaamme kulunvalvontajärjestelmiä lähes päivittäin esimerkiksi autojen lukoissa, pankkiautomaateilla sekä työpaikan sähköisissä kellokorteissa. Kulunvalvonta on prosessi, joka määrittelee kenellä on oikeus päästä tietoihin käsiksi. Kulunvalvonnan tavoitteena on suojata tietoa kaikilta luvattomilta henkilöiltä.

Kulunvalvonta määrittelee ja kontrolloi kenellä on oikeus olla vuorovaikutuksessa minkäkin kanssa, ja mitä kyseinen henkilö voi tehdä tämän vuorovaikutuksen aikana. Vuorovaikutuksen taso riippuu myönnettävistä oikeuksista suhteessa vuorovaikutuksen kohteena olevaan asiaan. Kulunvalvonnassa oikeudella tarkoitetaan lupaa suorittaa tiettyä toimintaa. Esimerkiksi yrityksen toimitusjohtajalla on oikeus olla suoraan yhteydessä yrityksen pääjohtajaan. Kuitenkin yrityksen työntekijällä, jonka tehtävänä on toimittaa pääjohtajan posti, tätä oikeutta ei ole. (Ballad ym. 2010, 4; Miller & Gregory, 2012, 48.)

3.2 Komponentit

Kulunvalvontatapahtuman osapuolet ovat

- Kulunvalvontajärjestelmä: Järjestelmä, joka toimii sille asetetuilla toimintaperiaatteilla.
- Subjekti: Voi olla esimerkiksi käyttäjä tai sovellus, pyytää oikeutta päästä resurssiin.
- Objekti: Resurssi, johon subjekti haluaa oikeuden päästä käsiksi.

(Ballad ym. 2010, 4.)

Kulunvalvontajärjestelmä

Kulunvalvontajärjestelmän rakenne koostuu seuraavista osista:

- Toimintaperiaatteet: Säännöt, jotka määräävät kenellä on oikeus käyttää suojattuja resursseja.
- Menettelytavat: Toiminnalliset menetelmät, joilla valvotaan toimintaperiaatteiden noudattamista.
- Työkalut: Tekniset menetelmät, joilla valvotaan toimintaperiaatteiden noudattamista.

Menettelytavat ja työkalut hoitavat siis toimintaperiaatteiden noudattamisen. Esimerkiksi monilla yrityksillä on toimintaperiaatteita, jotka määrittelevät, kenellä on pääsy asiakasrekisteriin. Asiakasrekisterit sisältävät luottamuksellisia tietoja, joita voitaisiin käyttää väärin, jos joku ulkopuolinen henkilö pääsisi tietoihin käsiksi. (Ballad ym. 2010, 5)

Toimintaperiaatteiden noudattamiseksi yrityksillä on menettelytapoja, joiden mukaan toimitaan. Esimerkiksi yrityksen asiakasrekisterin tietoihin pääsy myönnetään vain työntekijälle, jolla on riittävät valtuudet nähdä tietoja. Tämän lisäksi työntekijä esittää syyn, miksi hän pyytää kyseisiä tietoja. Mikäli työntekijän pyyntö hyväksytään, hän kirjautuu järjestelmään päästäkseen asiakasrekisteriin. Kirjautumisprosessi on tässä tapauksessa kulunvalvontajärjestelmän työkalu eli tekninen menetelmä. (Ballad ym. 2010, 5.)

Subjekti

Subjekti on henkilö tai sovellus, joka pyytää kulunvalvontajärjestelmältä lupaa päästä käsiksi objektiin. Koska objekti on subjektin kohde, subjekti on aina kulunvalvontaprosessin aktiivinen osapuoli. Subjektin tyyppejä on kolme:

- Valtuutettu: Subjekti, joka on esittänyt oman käyttäjätunnuksensa kulunvalvontajärjestelmälle ja järjestelmä on hyväksynyt subjektin pääsyn objektiin.

- Luvaton: Subjekti, jolla ei ole asianmukaisia käyttäjätunnuksia tai oikeustasoa päästä objektiin käsiksi.
- Tuntematon: Subjekti, joka ei ole esittänyt käyttäjätunnustaan. Täten ei tiedetä, voidaanko pääsy sallia vai ei.

Luvattoman ja tuntemattoman subjekti ero on ajoitus. Tuntemattoman subjektin henkilöllisyyttä ei tiedetä, koska subjekti ei ole vielä yrittänyt pyytää lupaa kulunvalvontajärjestelmältä päästäkseen objektiin käsiksi. Kun tuntematon subjekti tunnistautuu järjestelmälle, subjektin tyyppi muuttuu tällöin joko valtuutetuksi tai luvattomaksi. (Ballad ym. 2010, 5; Miller & Gregory, 2012, 48.)

Objekti

Objektit ovat kulunvalvontajärjestelmän suojaamia resursseja. Objektit ovat aina passiivisia, koska ne ovat subjektien toiminnan kohteita. Objektit voidaan jakaa kolmeen eri tyyppiin:

- Informaatio: Mitä tahansa tietoa, jolla on arvoa.
- Teknologia: Järjestelmät, sovellukset ja tietoverkot.
- Fyysinen sijainti: Rakennukset ja huoneet.

Informaatio on yleisin suojattava resurssi tietoteknisessä kulunvalvonnassa. Esimerkiksi tietokannat ja sovellukset suojataan salasanoin. Suojauksella taataan, että ainoastaan oikeat henkilöt saavat käyttää niiden sisältämiä tietoja. Teknologisten resurssien suojaaminen on myös tärkeää. Epärehellinen käyttäjä voi vaarantaa tietojen koskemattomuuden hyökkäämällä teknologiaa kohtaan, joka tallentaa ja käyttää sitä. Jos luvaton käyttäjä saa pääsyn esimerkiksi tiedostopalvelimelle, käyttäjä pystyy vahingoittamaan tietoja, jotka on tallennettu palvelimelle. Fyysisen kulunvalvonnan tehtävänä on varmistaa, että vain asianmukaisia tunnistetietoja esittävät henkilöt saavat käyttää fyysisiä resursseja. Rajoittamalla esimerkiksi palvelinhuoneelle pääsyä varmistetaan, että ulkopuoliset henkilöt eivät pysty varastamaan tai tuhoamaan palvelimia. (Ballad ym. 2010, 6; Miller & Gregory, 2012, 48.)

3.3 Prosessi

Kulunvalvontaprosessi koostuu kolmesta vaiheesta:

- Tunnistautuminen: Subjekti tunnistautuu kulunvalvontajärjestelmälle.
- Todentaminen: Kulunvalvontajärjestelmä tarkistaa subjektin henkilöllisyyden.
- Valtuuttaminen: Kulunvalvontajärjestelmä päättää salliiko vai estääkö se pääsyn subjektilta objektiin.

Todentaminen tapahtuu tavallisesti subjektilta piilossa, joten subjekti on ainoastaan tietoinen tunnistautumis- ja valtuutusvaiheesta. (Ballad ym. 2010, 7.)

Tunnistautuminen

Kulunvalvontaprosessin ensimmäinen vaihe on tunnistautuminen. Jotta kulunvalvontajärjestelmä tietäisi, miten sen kuuluisi toimia eri tilanteissa, subjektilla ja objektilla on oltava omat yksilölliset tunnisteensa. Subjektin tunniste voi olla vaikkapa käyttäjätunnus tai sähköpostiosoite. Jokaisella objektilla on oma tunniste, joka erottaa sen muista objekteista. Esimerkiksi tietoverkolla voi olla monta tulostinta, jolloin jokaisella tulostimella on oma yksilöllinen tunnisteensa. (Ballad ym. 2010, 7.)

Todentaminen

Tunnistautumisen jälkeen todentaminen on kulunvalvontaprosessin seuraava vaihe. Todentamisvaiheessa kulunvalvontajärjestelmä vaatii todistetta subjektin henkilöllisyydestä. Subjektin henkilöllisyys voidaan todentaa esimerkiksi salasanalla, poletilla tai jaetulla salaisella tiedolla. Poletilla tarkoitetaan esinettä, jonka subjekti omistaa, esimerkiksi älykortti. Jaettu salainen tieto on tieto, jonka tietävät vain kulunvalvontajärjestelmä ja subjekti. Useimmiten salaisena tietona

toimii kysymys, johon ainoastaan subjekti tietää vastauksen. Esimerkiksi kysymys: mikä on vanhimman sisaruksesi toinen nimi? (Ballad ym. 2010, 8.)

Valtuuttaminen

Kun tunnistautumis- ja todentamisvaiheet on tehty, kulunvalvontajärjestelmän tehtävänä on päättää, onko subjekti valtuutettu pääsemään pyydettyihin resursseihin. Subjektin valtuutus määritellään oikeuksilla, jotka riippuvat subjektin henkilöllisyydestä. Esimerkiksi yrityksen henkilöhallinnon johtaja voi olla valtuutettu katsomaan henkilökunnan tietoja, mutta hänellä ei ole valtuuksia muokata yrityksen tilinpäätöstä. Yksinkertaisimmillaan subjekti voidaan valtuuttaa pelkästään käyttäjätunnuksen ja salasanan avulla. Kuitenkin riippuen resurssin arvosta sekä subjektien määrästä valtuuttaminen voi olla paljon monimutkaisempaa. Tällöin subjektit voidaan jakaa eri tasoihin riippuen subjektien oikeuksista. Ylimmän tason subjektilla on kaikki resurssiin liittyvät oikeudet, mutta siitä seuraavat alemmat tasot sisältävät vähemmän oikeuksia. (Ballad ym. 2010, 9.)

4 NEAR FIELD COMMUNICATION

NFC (Near Field Communication) on langattoman viestinnän tekniikka, joka mahdollistaa tietojen vaihdon kahden laitteen välillä, jotka ovat lähietäisyydellä toisistaan. NFC:n avulla esimerkiksi älypuhelin voidaan asettaa NFC:tä tukevan laitteen läheisyyteen, jolloin tietojen lähetys tapahtuu nopeasti ja kätevästi. Tietojen lähettäminen tapahtuu ilman, että puhelimen tarvitsee koskettaa laitetta tai muodostaa erillistä yhteyttä laitteiden välille. (NearFieldCommunication.org, 2013a.)

4.1 Teknologia

NFC:ssä tietojen vaihtaminen tapahtuu kahden laitteen luomalla magneettikentällä, joka toimii 13.56 MHz:n taajuudella. Kommunikaation osapuolina toimivia laitteita kutsutaan aloitteentekijäksi ja kohteeksi. Aloitteentekijä aloittaa kommunikoinnin ja kohteen tehtävänä on odottaa aloitteentekijää. Tästä syystä kommunikoinnissa voi olla enintään kaksi osapuolta. (NFC Development & Consulting, 2013.)

Yleensä jokainen NFC-laite toimii kohteena. Ajoittain laite voidaan muuttaa aloitteentekijäksi, jolloin se etsii ympärillä olevia NFC-kohteita. Etsinnän jälkeen laite muuttuu takaisin kohteeksi. Jos aloitteentekijä löytää kohteen, laitteiden välille muodostetaan yhteys ja tietojen siirtäminen alkaa. (NFC Development & Consulting, 2013.)

Laitteet, jotka käyttävät NFC:tä, voivat olla joko aktiivisia tai passiivisia. Passiivinen laite, esimerkiksi NFC-tarra, sisältää tietoa, jota muut laitteet voivat lukea. Kuitenkaan passiivinen laite ei voi itse lukea muiden laitteiden tietoja. Aktiiviset laitteet voivat sekä lukea että lähettää tietoja. Aktiivinen laite, esimerkiksi älypuhelin, ei ainoastaan kerää tietoja passiivisista laitteista, vaan se voi myös vaihtaa tietoja muiden aktiivisten laitteiden kanssa. Lisäksi aktiivinen laite, passiivisen laitteen salliessa, voi muuttaa passiivisten laitteiden sisältämiä tietoja. (NearFieldCommunication.org, 2013b.)

4.2 APDU

Lukijan ja kortin välillä tapahtuva kommunikointi tapahtuu APDUn avulla. APDU (Application Protocol Data Unit) on viestintäyksikkö lukijan ja kortin välillä. APDUn rakenne on määritelty ISO 7816-standardien mukaisesti. APDU jaetaan kahteen eri osaan: komentoihin ja vastauksiin. (Rebel SimCard 2010.)

Komento-APDU lähetetään lukijalta kortille. Komento sisältää pakollisen neljätavuisen ylätunnisteen (CLA, INS, P1 ja P2). Ylätunnisteella kerrotaan, mikä komento halutaan suorittaa. CLA-tavu on komennon tyyppi. INS-tavu määrittää, mikä komento on kyseessä. P1 ja P2-tavut ovat komennon parametreja. Tämän lisäksi komento sisältää vapaaehtoisen runko-osan, jonka pituus on maksimissaan 255 tavua. Runko-osa koostuu kolmesta osasta (Lc, Le ja Data-kenttä). Lc-tavu määrittelee lähetettävän datan pituuden. Le-tavu vuorostaan määrittelee vastaanotettavan datan pituuden. Data-kenttä sisältää datan, joka suoritetaan ylätunnisteella annetun ohjeen mukaisesti. (Rebel SimCard 2010.)

Vastaus-APDU lähetetään kortilta lukijalle. Vastaus koostuu pakollisesta kaksitavuisesta statusviestistä (SW1 ja SW2) sekä varsinaisesta datasta. Statusviesti kertoo prosessin tilasta komento-APDUn suorituksen jälkeen. Palautettava data vastauksesta on maksimissaan 256 tavua pitkä. (Rebel SimCard 2010.)

Komennot ja vastaukset lähetetään heksadesimaalimuodossa. Esimerkiksi komento, jolla halutaan tietää kulkukortin numero, lähetetään kortille seuraavassa muodossa: [0xFF, 0xCA, 0x00, 0x00, 0x00]. Kulkukortti palauttaa vastauksena seuraavanlaisen taulukon: [[data], 0x90, 0x00]. Data sisältää kortin numeron taulukkona, jonka alkiot ovat heksadesimaalilukuja. Esimerkiksi Pythonin avulla taulukko voidaan käydä läpi ja muuttaa merkkijonoksi, jolloin kortin numero on luettavammassa muodossa. Vastauksen luvut 0x90 ja 0x00 kertovat komennon onnistuneen.

5 OAUTH 2.0

OAuth 2.0 on todentamiseen käytettävä ohjelmistokehys, joka mahdollistaa loppukäyttäjän käyttöoikeudet palvelimen suojattuihin resursseihin resurssin omistajan puolesta. Tämän lisäksi OAuth 2.0 tarjoaa prosessin, jonka avulla loppukäyttäjä voi sallia kolmannen osapuolen sovelluksen käyttää palvelimen resursseja ilman loppukäyttäjän käyttäjätunnusta ja salasanaa. OAuth 2.0 on OAuth-protokollan seuraava kehitysaskel, mutta se ei kuitenkaan ole taaksepäin yhteensopiva sen edeltäjänsä kanssa. (Hardt 2012, 1.)

5.1 Roolit

OAuth 2.0 määrittelee neljä erilaista roolia: resurssin omistaja, resurssipalvelin, asiakas ja todennuspalvelin. Resurssin omistaja päättää luvasta päästä käsiksi suojattuun resurssiin. Jos resurssin omistaja on henkilö, hänestä käytetään nimitystä loppukäyttäjä. Resurssipalvelin ylläpitää suojattuja resursseja, ja hyväksyy ja vastaa resurssia koskeviin pyyntöihin. Asiakkaalla tarkoitetaan sovellusta, joka tekee suojattuihin resursseihin liittyviä pyyntöjä resurssin omistajan puolesta ja luvalla. Todennuspalvelin myöntää käyttöoikeustietueita asiakkaalle onnistuneen todentamisen jälkeen. Resurssi- ja todennuspalvelin voivat olla sama palvelin tai ne voivat toimia myös erikseen. (Hardt 2012, 6.)

5.2 Todentamisprosessi

Todentamisprosessi alkaa asiakkaan pyytäessä lupaa resurssin omistajalta. Pyyntö tehdään joko suoraan resurssin omistajalta tai epäsuorasti käyttämällä todennuspalvelinta välikätenä. Asiakas saa luvan, joka on vastine resurssin omistajan tunnistetiedoista. Luvan saatuaan asiakas pyytää käyttöoikeustietuetta todennuspalvelimelta esittämällä saamansa myönnetyn luvan. Todennuspalvelin todentaa asiakkaan ja vahvistaa voimassa olevan luvan. Onnistuneen vahvistuksen jälkeen todennuspalvelin myöntää asiakkaalle käyttöoikeustietu-

een, jolloin asiakas pyytää suojattua resurssia resurssipalvelimelta esittämällä myönnetyn käyttöoikeustietueen. Resurssipalvelin tarkistaa käyttöoikeustietueen kelvollisuuden ja palvelee pyyntöä, mikäli tietue todetaan voimassaolevaksi. (Hardt 2012, 7.)

5.3 Käyttöoikeustietueen päivittäminen

Mikäli käyttöoikeustietue mitätöityy tai sen käyttöaika ei ole enää voimassa, täytyy tietue päivittää uudempaan tietueeseen. Tietue päivitetään käyttämällä erityisiä päivitystietueita. Kun käyttöoikeustietue ei ole enää voimassa, resurssipalvelin ilmoittaa asiakkaalle virheellisestä tietueesta. Tällöin asiakas pyytää uutta käyttöoikeustietuetta todennuspalvelimelta päivitystietueen avulla. Jos todennuspalvelin toteaa asiakkaan ja päivitystietueen kelvolliseksi, palvelin antaa asiakkaalle uuden käyttöoikeus- ja päivitystietueen. (Hardt 2012, 10.)

6 TEHDEN-KULUNVALVONNAN TOTEUTUSTEKNIIKAT

Kulunvalvontasovelluksen ja tietoikkunan ohjelmointikielenä toimii Python. Pythonin monipuolisuuden ja yhteensopivuuden ansiosta se sopii hyvin rajapinnoilla toimivien sovelluksien ohjelmointikieliksi. Asiakkaiden kulkukorteissa oleville NFC-tageille lähetettävien APDU-komentojen takia kulunvalvontasovellus tarvitsee Pyscard-moduulin, koska Pythonin omista moduuleista tällaista ominaisuutta ei löydy. Jokaisen kulkukortin lukemisen yhteydessä kulunvalvontasovellus ottaa yhteyden Tehden-ohjelmistossa toimivaan Tehden API -rajapintaan. Tehden APIn ja Tehden-ohjelmiston ohjelmointikielenä käytetään PHP:tä. Lisäksi kulunvalvontasovelluksella on oma pienikokoinen tietokantansa, joka toimii SQLite-tietokantajärjestelmää käyttäen.

6.1 Python

Python on tulkittava, oliopohjainen ja korkean tason ohjelmointikieli. Se on yksinkertaisen syntaksinsa ansiosta selkeä ja luettava ohjelmointikieli. Lisäksi Python tukee moduuleita ja paketteja, mikä kannustaa ohjelman modulaarisuutta ja koodin uudelleenkäytettävyyttä. Python-tulkki ja kirjastot on kehitetty avoimen lähdekoodin projektina, ja ne ovat saatavina maksutta kaikille tärkeille alustoille sekä ne ovat vapaasti levitettävissä. (Python Software Foundation 2013)

6.2 Pyscard

NFC-tageille lähetettäviä komentoja ei voida lähettää Pythonin omilla moduuleilla. Tämän ongelman ratkaisemiseksi voidaan käyttää Pyscard Python-moduulia, joka lisää Pythoniin tuen älykortteja varten. Tämän moduulin avulla Pythonilla voidaan ohjelmoida sovellus, joka pystyy APDU-komennoin keskustelemaan älykorttien kanssa. Tämän lisäksi Pyscard sisältää monia hyödyllisiä funktioita, jotka helpottavat ohjelmointia. Pyscard sisältää esimerkiksi funktion,

jonka avulla kulkukortin numero voidaan helposti muuttaa APDU-vastauksen heksadesimaalimuodosta tavalliseksi merkkijonoksi, jolloin se on paremmin ymmärrettävässä muodossa. (Aussel 2009.)

6.3 PHP: Hypertext Preprocessor

PHP on laajasti käytetty avoimen lähdekoodin ohjelmointikieli, jota käytetään erityisesti Web-ympäristössä dynaamisten web-sivujen luonnissa (The PHP Group 2013a). PHP:llä tehdään pääasiassa dynaamisia web-sivuja, mutta tämän lisäksi PHP:llä voidaan tehdä komentorivisovelluksia ja graafisia käyttöliittymiä. PHP:tä voidaan käyttää kaikilla yleisimmillä käyttöjärjestelmillä ja se tukee monia web-palvelimia. (The PHP Group 2013b.)

6.4 SQLite

SQLite on sulautettu relaatiotietokantajärjestelmä. Se on toteutettu pienenä kirjastona, jonka koko voi olla jopa alle 350 KiB. Toisin kuin muut tietokannat, SQLite ei tarvitse erillistä tietokantapalvelinta. Koko tietokanta tauluineen, indeksineen ja näkymineen on tallennettuna yhteen tiedostoon. SQLite käyttää yksinkertaista toteutusta lukitsemalla tietokannan transaktioiden ajaksi. (SQLite 2013.)

7 TEHDEN-KULUNVALVONNAN RAKENNE

Tehden-kulunvalvonta koostuu kuudesta eri osasta, jotka ovat:

- Tehden-ohjelmisto
- kulunvalvontasovellus
- tietoikkuna
- SQLite-tietokanta
- Tehden API
- NFC-lukija.

Tehden-kulunvalvonnan tärkeimmät ja suurimmat osat ovat kulunvalvontasovellus ja Tehden API. Kulunvalvontasovellus on yhteydessä jokaiseen Tehden-kulunvalvonnan eri osa-alueeseen. Sovellus sekä lähettää että vastaanottaa viestejä Tehden-kulunvalvonnan eri osilta ja tarvittaessa välittää niitä eteenpäin. Tehden API on siksi tärkeä osa kokonaisuutta, koska sen tehtävänä on hoitaa kulkukortin tarkistukset ja palauttaa aina oikea vastaus kortin tilasta.

7.1 Tehden-ohjelmisto

Tehden-ohjelmisto koostuu useasta eri moduulista. Näistä moduuleista Tehden-kulunvalvontasovellus toimii yhdessä kulunvalvonta- ja asiakasmoduulien kanssa. Kulunvalvontamoduulia tarvitaan NFC-lukijan liittämiseen Tehden-ohjelmiston kanssa. Asiakasmoduulia puolestaan käytetään asettamaan asiakkaiden jäsenkorttinumero NFC-lukijan kautta.

7.2 Kulunvalvontasovellus

Kulunvalvontasovellus on kulunvalvonnan kokonaisuuden kannalta sen tärkein osa, koska se on yhteydessä jokaisen muun Tehden-kulunvalvonnan osan kanssa. Kulunvalvontasovellus koostuu Python-moduuleista, joista jokainen

moduuli hoitaa tiettyä sovelluksen toimintaa. Sovellus rakentuu seuraavista moduuleista:

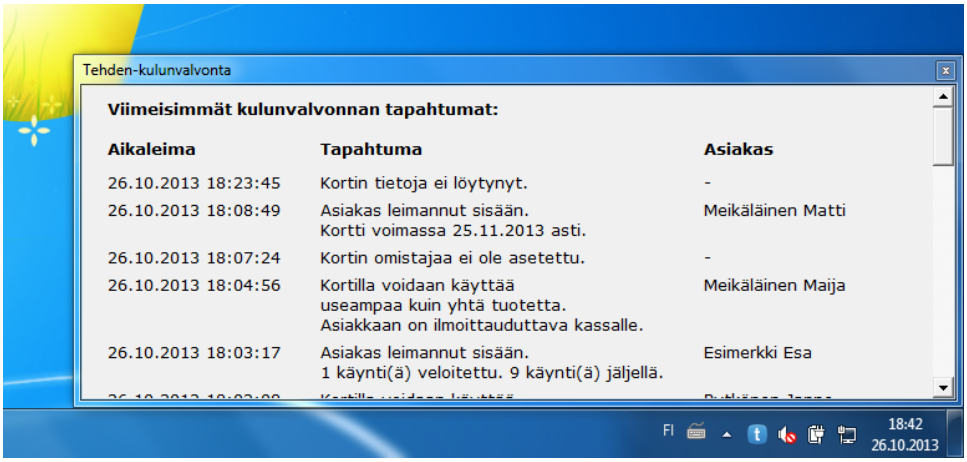
- Accesscontrolcore: Sovelluksen päämoduuli, jonka tehtävänä on hoitaa kulunvalvontatapahtumat ja olla yhteydessä muihin sovelluksen moduuleihin (Liite 1).
- Accesscontrolservice: Huolehtii sovelluksen käynnistyksestä ja sammuttamisesta (Liite 2).
- Api: Hoitaa Tehden API -rajapintaan tehtävät kyselyt (Liite 3).
- Cardreader: Huolehtii NFC-lukijan ja sovelluksen välisestä yhteydestä sekä asettaa NFC-lukijan viestit, summerin äänet ja LED-valot (Liite 4).
- Database: Huolehtii tietokantayhteydestä ja tekee kyselyt tietokantaan (Liite 5).
- Httpserver: Sovelluksen yhdistämisestä Tehden-ohjelmistoon huolehtiva moduuli (Liite 6).
- Oaclient: Hakee Tehden API -rajapinnan käyttöön tarvittavan Oauth 2.0 -käyttöoikeustietueen Tehden-ohjelmiston tietokannasta ja tietueen vanhentuessa päivittää sen (Liite 7).

7.3 Tietoikkuna

Tietoikkuna on itsenäinen ohjelma, joka ottaa vastaan kulunvalvontasovelluksen lähettämiä tietoja. Tietoikkuna koostuu yhdestä Infowindow nimisestä Python-moduulista (Liite 8). Tietoikkunan tehtävä on kertoa yrityksen henkilökunnalle, tavallisesti kassalla toimiville henkilöille, NFC-lukijassa luettavan kulkukortin ja asiakkaan tiedot. Tällöin ongelmatilanteiden sattuessa, henkilökunta pystyy päättelemään ongelman syyn ja auttaa asiakasta tarvittaessa.

Kulunvalvontasovelluksen käynnistyessä se avaa yhteyden tietoikkunaan Web-Socket-protokollaa käyttäen. Jokaisen kulunvalvontatapahtuman yhteydessä kulunvalvontasovellus lähettää yhteyden kautta tiedot tietoikkunaan, jolloin tietoikkuna näyttää kortin ja asiakkaan tiedot. Kun NFC-lukija on odottavassa tilassa, tietoikkuna näyttää listattuna viimeiset 50 kulunvalvontatapahtumaa. Tie-

toikkuna avautuu näytön oikeaan alalaitaan, aivan tehtäväpalkin yläpuolelle ja se on mahdollista avata tehtäväpalkin kautta painamalla sovelluksen kuvaketta (Kuva 1).



Kuva 1. Tehden-kulunvalvonnan tietoikkuna.

7.4 Tietokanta

Kulunvalvontasovellus käyttää omaa tietokantaansa, joka on toteutettu käyttämällä SQLite-tietokantajärjestelmää. Tietokanta sisältää tiedot kulunvalvontatapahtumista ja Tehden APIn tunnistautumistiedoista. Tietokanta koostuu kahdesta taulusta, joista logevent-taulussa (Taulukko 1) on tapahtumien tiedot ja tokensettings-taulu (Taulukko 2) sisältää tunnistautumistiedot.

Taulukko 1. Logevent-tietokantataulu.

Kenttä	Tyyppi	Selite
id	INTEGER PRIMARY KEY	Yksilöllinen rivin tunniste, joka on jokaisessa SQLite-tietokannan taulussa
timestamp	TEXT	Kulunvalvontatapahtuman aikaleima
event	TEXT	Kulunvalvontatapahtuman kuvaus
cardnumber	TEXT	Kortin numero
customer	TEXT	Asiakkaan nimi

Taulukko 2. Tokensettings-tietokantataulu.

Kenttä	Tyyppi	Selite
id	INTEGER PRIMARY KEY	Yksilöllinen rivin tunniste, joka on jokaisessa SQLite-tietokannan taulussa
access_token	TEXT	Tehden APIn käyttöoikeustietue
refresh_token	TEXT	Tehden APIn päivitystietue
base_uri	TEXT	Verkko-osoitteen alkuosa, jonka avulla Tehden APIa voidaan käyttää

7.5 Tehden API

Tehden API on rajapinta, jonka avulla käyttäjät voivat käyttää Tehden-ohjelmistoa ilman, että käyttäjän itse tarvitsee olla kirjautuneena ohjelmistossa. Käyttäjän todentamiseen käytetään Oauth 2.0 -ohjelmistokehystä. Tehden APIn ohjelmointikielenä toimii PHP.

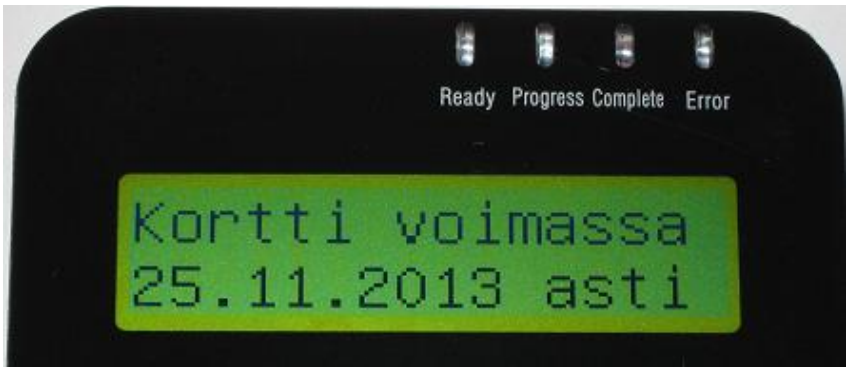
Tehden API on Tehden-ohjelmiston tavoin jaettu omiin moduuleihinsa. Näistä moduuleista kulunvalvontasovellus käyttää rajapinnan kulunvalvontamoduulia. Tehden APIn avulla voidaan päästä käsiksi asiakkaiden kulkukorttien tietoihin. Lisäksi Tehden API sisältää toiminnallisuudet, jotka hoitavat kulunvalvontaan liittyvät tarkistukset (Liite 9). Tarkistusten jälkeen rajapinta lähettää vastauksen kulunvalvontasovellukselle.

7.6 NFC-lukija

ACR1222L on tietokoneeseen liitettävä kontaktiton NFC-lukija (Kuva 2). Lukija kytketään tietokoneeseen USB-portin kautta. Lukija on varustettu LCD-näytöllä, summerilla sekä neljällä LED-valolla. Näiden avulla käyttäjää informoidaan lukijan tilasta. (Advanced Card Systems Ltd 2013)

Kuva 2. ACR1222L USB NFC-lukija (Advanced Card Systems Ltd, 2013).

Tehden-kulunvalvonnan tapauksessa asiakkaan näytettyä kulkukorttinsa NFC-lukijan näyttö kertoo asiakkaalle mahdollisesta veloituksesta tai kuinka kauan kortti on vielä voimassa (Kuva 3). Lisäksi NFC-lukijan summeri ja LED-valot reagoivat eri tavalla onnistuneen ja epäonnistuneen leimauksen jälkeen.



Kuva 3. NFC-lukijan ilmoitus aikaperusteisen kortin käytöstä.

Muiden kuin ACR1222L NFC-lukijan yhteensopivuudesta Tehden-kulunvalvonnan kanssa ei ole täyttä varmuutta. Kuitenkin on hyvin todennäköistä, että myös muut USB:llä toimivat NFC-lukijat voivat toimia täysin ongelmitta Tehden-kulunvalvonnan kanssa.

8 TEHDEN-KULUNVALVONNAN TOIMINTA

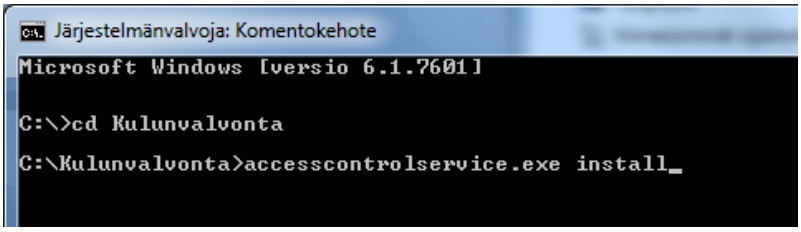
Tehden-kulunvalvonnan tehtävänä on huolehtia yrityksen asiakkaiden kulunvalvonnasta. Joka kerta, kun asiakas asettaa korttinsa NFC-lukijan läheisyyteen, kulunvalvontasovellus ottaa talteen kortin lähettämän APDU-vastauksen. Tämän jälkeen APDU-vastaus muutetaan heksadesimaalimuodosta tavalliseksi merkkijonoksi, minkä jälkeen kulunvalvontasovellus lähettää kortin numeron eteenpäin Tehden API -rajapinnalle.

Tehden API tekee kulkukorttiin liittyvät tarvittavat tarkistukset ja lähettää vastauksen riippuen kortin tilasta. Rajapintayhteyden lisäksi kulunvalvontasovellus voi olla tietyissä tilanteissa yhteydessä suoraan Tehden-ohjelmistoon selaimen kautta. Kulunvalvonnan liittäminen osaksi Tehden-ohjelmistoa tapahtuu edellä mainitulla tavalla.

Tehden-kulunvalvonnan avulla on myös mahdollista liittää kulkukortti asiakkaalle vain näyttämällä korttia NFC-lukijalle. Tätä ominaisuutta käytetään Tehden-ohjelmiston asiakasmoduulissa. Kun asiakkaalle ollaan liittämässä korttia, kulunvalvonnan muu toiminta pysähtyy hetkeksi. Kortin numeron liittämisen jälkeen toiminta palautuu taas normaaliksi.

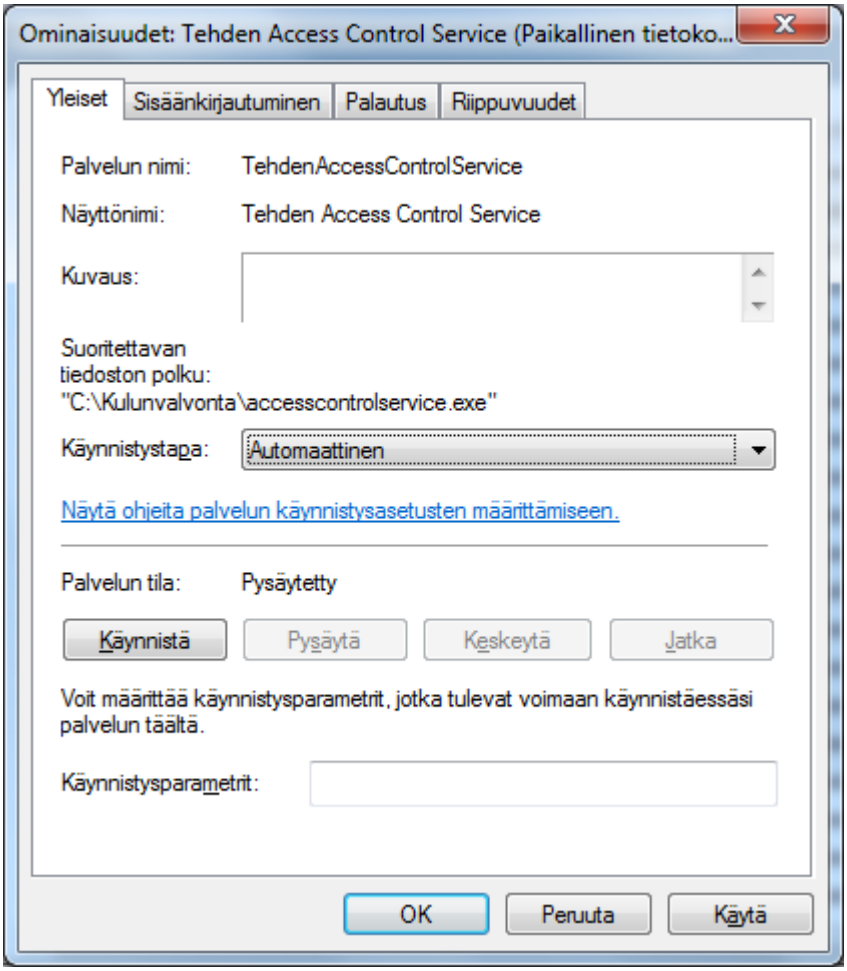
8.1 Kulunvalvonnan asennus

Tehden-kulunvalvonnan kulunvalvontasovelluksen asennus tapahtuu toistaiseksi komentokehotteen kautta. Kun asiakas haluaa kulunvalvonnan asennettavan, asiakkaaseen otetaan yhteys TeamViewer-ohjelmistolla. TeamViewer on ohjelmisto, jonka avulla voidaan ohjata tietokoneita etänä Internetin kautta (TeamViewer, 2013). Tarvittavat tiedostot puretaan haluttuun kansioon. Tämän jälkeen siirrytään komentokehotteella kyseiseen kansioon. Kansiossa ajetaan komento, joka asentaa kulunvalvontasovelluksen (Kuva 4).



Kuva 4. Kulunvalvontasovelluksen asennuskomento.

Kulunvalvontasovelluksen asennus tapahtuu komentorivin kautta, koska se toimii Windows-palveluna. Jotta kulunvalvonta olisi toiminnassa aina tietokoneen ollessa käynnissä, palvelun käynnistystapa täytyy vielä muuttaa automaattiseksi. Tämän asetuksen muuttaminen tapahtuu Windowsin Palvelut-ikkunasta (Kuva 5).



Kuva 5. Windowsin Palvelut -ikkuna.

Kun Windows-palvelun asetukset on saatu kuntoon, täytyy vielä asettaa myös tietoikkuna käynnistymään automaattisesti tietokoneen käynnistymisen yhteydessä. Tämä tapahtuu yksinkertaisesti kopioimalla tietoikkunan tiedoston pikakuvake Windowsin Käynnistys-kansioon. Tällöin Windows käynnistää tietoikkunan heti Windowsin käynnistyttyä.

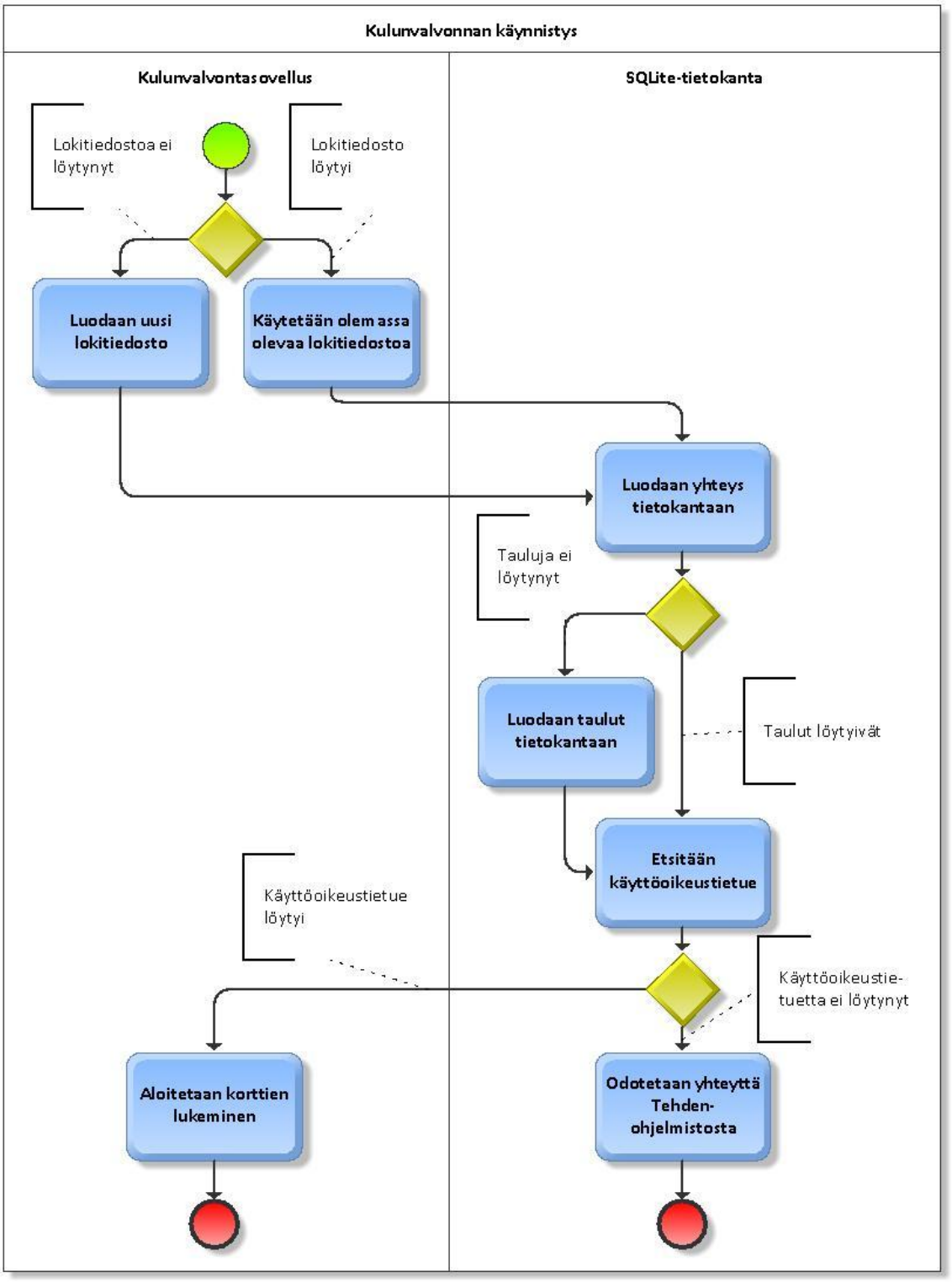
Kulunvalvontasovelluksen ja selaimen välinen keskustelu tapahtuu WebSocket-protokollaa käyttäen. Jotta yhteys sovelluksen ja selaimen välillä onnistuisi, tietoliikenne näiden kahden välillä pitäisi sallia. Asettamalla selaimeen uusi sertifikaatti kulunvalvontasovellusta varten, voidaan mahdollistaa yhteys sovelluksen ja selaimen välillä.

Kulunvalvonnan asennus on varsin monimutkainen prosessi. Asennusta onkin tarkoitus tulevaisuudessa helpottaa tekemällä erillinen asennusohjelma, jolloin kaikkia asetuksia ei tarvitsisi asettaa käsin. Tällöin myös asiakas voisi vain ladata asennusohjelman ja asentaa kulunvalvontasovelluksen itse, eikä erillistä etäyhteyttä asiakkaaseen enää tarvittaisi asennusvaiheessa.

8.2 Kulunvalvonnan käynnistys

Kulunvalvonnan käynnistyksen yhteydessä suoritetaan tarvittavat toiminnot, jotta kulunvalvonta toimii oikein (Kuvio 1). Ensimmäisen käynnistyksen yhteydessä luodaan lokitiedosto kansioon, jossa kulunvalvontasovellus sijaitsee. Ensimmäisen käynnistyskerran jälkeen lokitietojen kirjaamista jatketaan jo olemassa olevaan tiedostoon. Lokitiedostoon merkitään tietoja sovelluksen tilasta, jolloin sen avulla voidaan havaita mahdolliset virhetilanteet.

Alustusvaiheen jälkeen kulunvalvontasovellus ottaa yhteyttä tietokantaan. Jos tietokantaa ei löydy, uusi tietokanta luodaan samaan kansioon, jossa sovellus sijaitsee. Kun tietokantaan on saatu yhteys, tarkistetaan ovatko tietokannan taulut olemassa. Mikäli tauluja ei ole olemassa, uudet taulut luodaan. Tietokannan ja taulujen luominen tapahtuu käytännössä vain ensimmäisen käynnistyksen yhteydessä, ellei tietokantaa ole myöhemmin poistettu.



Kuvio 1. Kulunvalvonnan käynnistys.

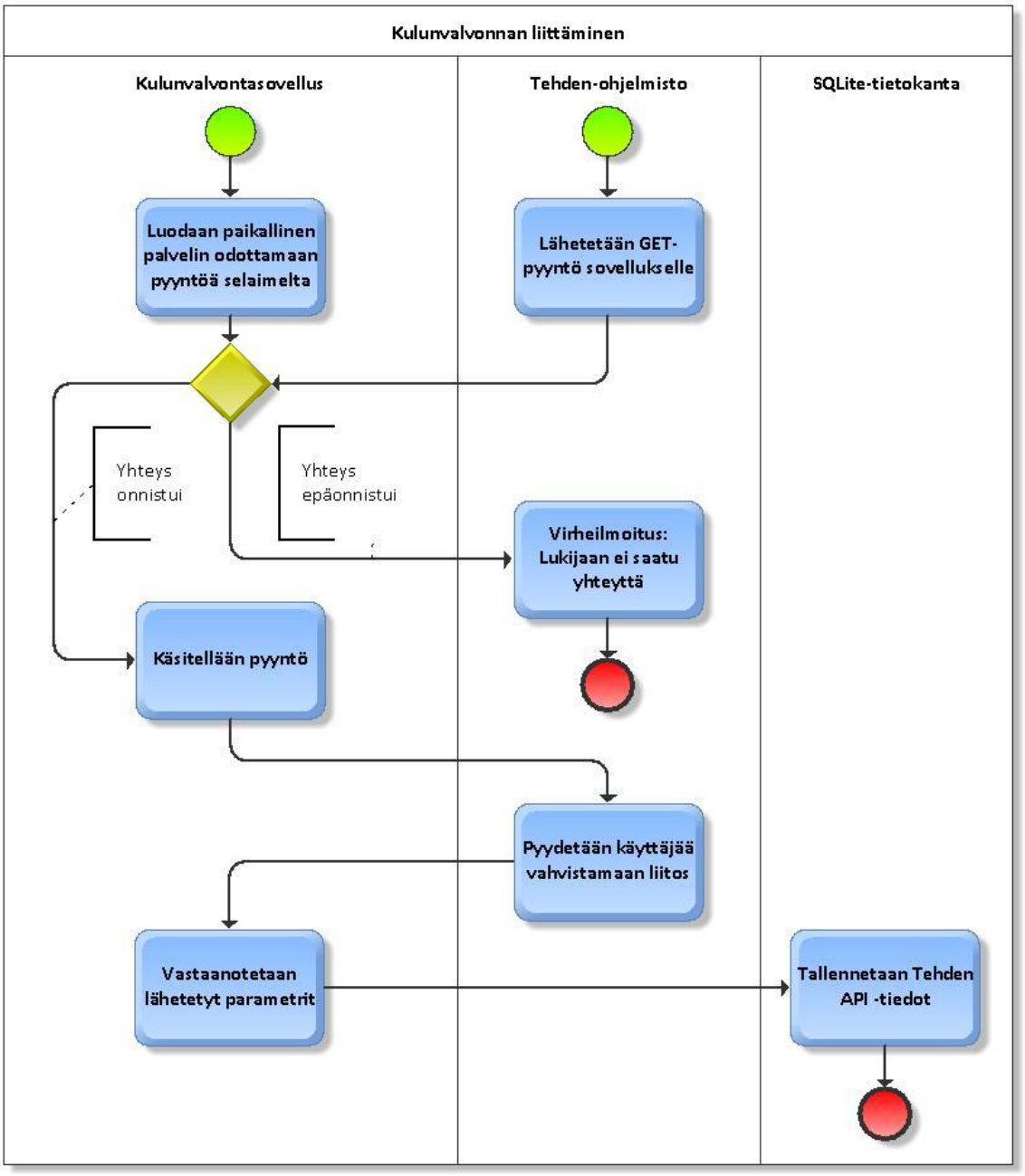
Tämän jälkeen tarkistetaan, onko liitos kulunvalvontasovelluksen ja Tehden-ohjelmiston välillä jo tehty. Kulunvalvontasovellus hakee tietokannasta tiedon

siitä, onko tietokantaan tallennettu käyttöoikeustietueen tiedot. Mikäli tietoja ei löydy, kulunvalvontasovellus jää valmiustilaan odottamaan yhteyttä Tehden-ohjelmistosta.

Käyttöoikeustietueen löytyessä kulunvalvonta käynnistyy normaalisti. Tällöin kulunvalvontasovellus käynnistää kolme säiettä, joista jokaisella on oma tehtävänsä. Yksi säie hoitaa viestien lähettämisen tietoikkunalle, toinen hoitaa kulkukorttien lukemisen ja kolmas pitää yllä selaimen ja kulunvalvontasovelluksen välistä yhteyttä. Kun jokaisen tehtävät toimivat omilla säikeillään, tehtävät voidaan hoitaa samanaikaisesti. Selaimen ja kulunvalvontasovelluksen välistä yhteyttä käytetään asiakkaan kulkukortin asettamisessa, jolloin selain keskustelee kulunvalvontasovelluksen kanssa.

8.3 Kulunvalvonnan liittäminen Tehden-ohjelmistoon

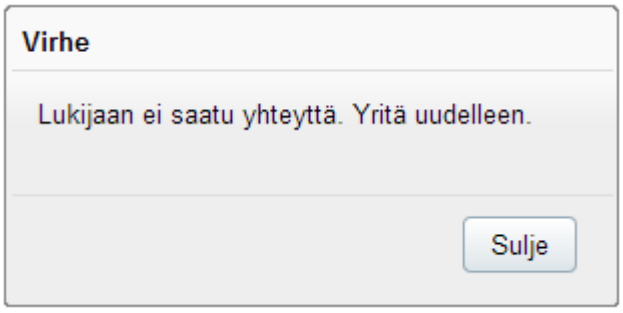
Ennen kulunvalvonnan varsinaista käyttöä on muodostettava yhteys kulunvalvontasovelluksen ja Tehden-ohjelmiston välille (Kuvio 2). Kulunvalvonnan liittäminen tehdään Tehden-ohjelmiston kulunvalvontamoduulin kautta. Tehden-ohjelmisto vaatii käyttäjältä kirjautumista, jotta käyttäjä voi käyttää ohjelmiston eri toimintoja. Tehden-kulunvalvonta ei kuitenkaan voi käyttää muiden käyttäjien tunnistetietoja, vaan yhteys Tehden-ohjelmiston ja kulunvalvonnan välillä on toteutettava OAuth 2.0 -todentamisprotokollaa käyttäen.



Kuvio 2. Kulunvalvonnan liittäminen.

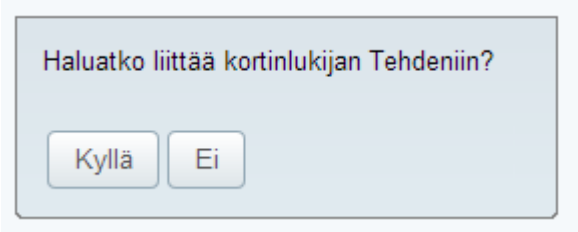
Kun kulunvalvonta käynnistetään eikä tarvittavaa käyttöoikeustietuetta löydy tietokannasta, kulunvalvontasovellus luo paikallisen www-palvelimen ja jää odottamaan yhteyttä Tehden-ohjelmistosta. Tehden-ohjelmistossa luodaan HTTP-protokollan mukainen GET-metodi. Kulunvalvontasovellus käsittelee pyynnön ja vastaa takaisin Tehden-ohjelmistolle. Mikäli yhteys kulunvalvonta-sovellukseen saadaan onnistuneesti, selain avaa uuden välilehden, jossa ku-

lunvalvonnan liittäminen toteutetaan. Jos Tehden-ohjelmisto ei saa yhteyttä kulunvalvontasovellukseen, käyttäjä saa siitä virheilmoituksen (Kuva 6).



Kuva 6. NFC-lukijan yhteysvirheilmoitus.

Kulunvalvonnan liittäminen osaksi Tehden-ohjelmistoa tapahtuu lomakkeen avulla. Lomakkeen tiedot ovat kaikki valmiiksi asetetut ja käyttäjältä piilossa, joten käyttäjän tarvitsee ainoastaan hyväksyä liitos (Kuva 7).



Kuva 7. NFC-lukijan liittämisen hyväksyminen.

Lomake lähettää seuraavat parametrit POST-metodia käyttäen:

- client_id: Asiakkaan todentamisessa käytettävä asiakastunniste.
- redirect_uri: Absoluuttinen URL, jonne todennuspalvelin ohjaa käyttäjän hyväksytyn todennusprosessin jälkeen.
- response_type: Pyynnön vastaus, joka voi olla joko käyttöoikeustietue, valtuutuskoodi tai molemmat edellä mainituista.
- state: Mikä tahansa merkkijono, josta on hyötyä ohjelmalle, joka käyttää sitä. Tehden-kulunvalvonnan tapauksessa se verkko-osoite, jonka avulla voidaan kutsua Tehden APIa.

Lähetetyt parametrit käsitellään Tehden-ohjelmiston omassa oauth-moduulissa. Jos kaikki tiedot ovat kelvollisia, käyttäjä ohjataan uudelle sivulle, jonka osoite on muodostettu oauth-moduulissa. Osoitteen loppuosa sisältää valtuutuskoodin (response_type) ja Tehden API -rajapintaan tarvittavan verkko-osoitteen (state), jonka avulla Tehden APIa voidaan käyttää. Kun käyttäjä ohjataan uudelle sivulle, kulunvalvontasovelluksen luoma www-palvelin ottaa talteen osoitteessa olevat valtuutuskoodin ja verkkosivun osoitteen. Kyseiset tiedot tarvitaan käyttötietueen pyytämiseksi. Sovellus lähettää GET-pyynnön Tehden-ohjelmistoon. Pyyntö sisältää seuraavat parametrit:

- base_uri: Tehden-ohjelmiston palauttama state-parametri, joka sisältää Tehden API -rajapintaan tarvittavan verkko-osoitteen alkuosan.
- grant_type: Valtuutuskoodin tyyppi.
- code: Tehden-ohjelmiston palauttama valtuutuskoodi.
- redirect_uri: Osoite, jonne käyttäjä ohjataan GET-pyynnön jälkeen. Tehden-kulunvalvonnan tapauksessa käyttäjä ohjataan paikalliselle palvelimelle.

Parametrien lisäksi pyynnön ylätunnisteeseen lisätään kulunvalvontamoduuliin liittyvän käyttäjätunnuksen (client_id) ja salasanan muodostama base64-koodattu merkkijono.

Tehden-ohjelmiston oauth-moduuli palauttaa käyttöoikeustietueen, tietueen tyypin ja voimassaoloajan sekä päivitystietueen. Nämä tiedot tallennetaan sovelluksen tietokantaan, josta tiedot haetaan jokaisen kulunvalvontatietojen päivittämisen yhteydessä.

Tietojen tallentamisen jälkeen käyttäjälle ilmoitetaan sovelluksen liittämisen Tehden-ohjelmiston kanssa onnistuneen. Onnistuneen liitoksen jälkeen Tehden-kulunvalvonta on valmiina aloittamaan varsinaisen toiminnan.

8.4 Korttien lukeminen

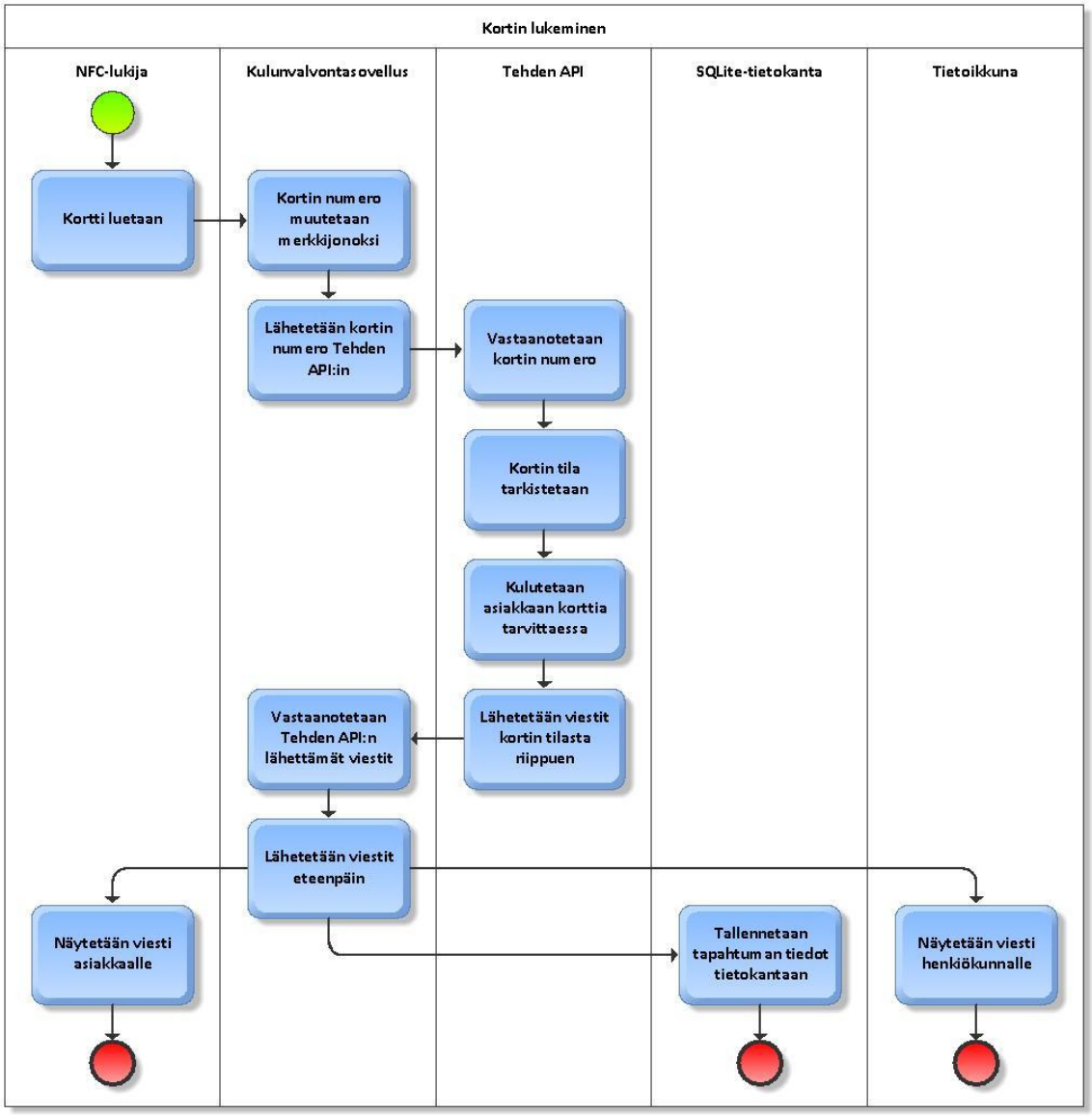
Kun Tehden-kulunvalvonta on valmiina toimimaan, NFC-lukijalla voidaan alkaa lukea asiakkaiden kulkukortteja. Korttien lukemisen aikana kulunvalvontasovellus on Tehden-ohjelmistoa lukuun ottamatta yhteydessä kulunvalvonnan kaikkiin muihin osiin (Kuvio 3). Kulunvalvontasovelluksen käynnistyttyä sovellus etsii tietokoneeseen liitetyn NFC-lukijan ja yrittää ottaa yhteyttä löydettyyn lukijaan. Jos NFC-lukijaan ei saada yhteyttä, sovellus yrittää ottaa uudelleen yhteyttä lukijaan tietyin aikavälein. Yhteysyrityksiä tehdään niin kauan kunnes NFC-lukijaan saadaan yhteys tai sovellus sammutetaan.

Onnistuneen yhteyden muodostamisen jälkeen NFC-lukijan LCD-näyttö ja LED-valot asetetaan valmiustilaan. Kun NFC-lukija on valmiustilassa, lukija jää odottamaan luettavia kulkukortteja. NFC-lukija pysyy valmiustilassa, kunnes lukijan läheisyyteen asetetaan kortti.

Kun NFC-lukijan läheisyyteen asetetaan kulkukortti, kulunvalvontasovellus muodostaa kortin lähettämästä APDU-vastauksen tiedoista merkkijonon, joka koostuu kirjaimista ja numeroista. Tämä merkkijono on jokaisen kortin yksilöllinen tunnistetieto. Tunniste lähetetään Tehden API -rajapintaan, joka tunnisteen tietojen perusteella lähettää vastauksen kulunvalvontasovellukselle. Tehden APIssa tehdään seuraavat tarkistukset, joiden perusteella takaisin lähetettävä vastaus muodostuu:

- Onko korttia tallennettu tietokantaan?
- Onko kortilla omistajaa?
- Onko kortti aktiivisessa tilassa?
- Onko kortti liitettynä kulunvalvontaryhmään?
- Onko kortti voimassa?
- Onko korttia liitetty asiakastiliin?
- Onko kortti aikaperusteinen vai kappaleperusteinen?
- Onko kortin vaihtoaika vielä voimassa?
- Onko kortin saldo loppunut?

- Voidaanko kortilla käyttää useampaa tuotetta?
- Onko kortin saldo riittävä?



Kuvio 3. Kortin lukeminen.

Jos kortti on aikaperusteinen tai kappaleperusteinen ja kortilla on saldoa jäljellä tai vaihtoaika on voimassa, Tehden API lähettää vastauksen, joka kertoo oliko kortinluku onnistunut. Muissa tapauksissa Tehden API lähettää vastauksen, jossa kerrotaan kortinlukemisen epäonnistuneen. Vastaus sisältää aina seuraavat asiat riippumatta siitä oliko kortinluku onnistunut vai ei:

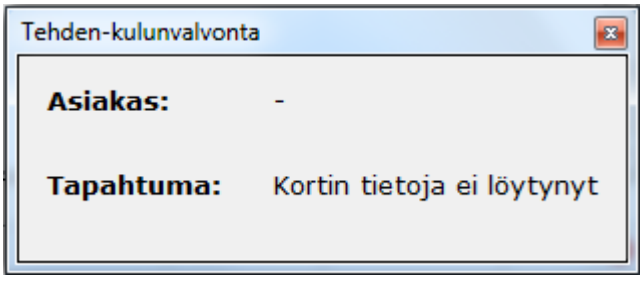
- Asiakkaan nimi, jos asiakas löytyy tietokannasta.

- Tehden-tietoikkunan viesti.
- NFC-lukijan viesti, joka voi olla erilainen kuin tietoikkunan viesti, koska lukijan LCD-näyttöön mahtuu vain rajattu määrä merkkejä.
- Vastauksen tila, joka kertoo oliko kortinluku onnistunut vai epäonnistunut. Tilan mukaan asetetaan NFC-lukijan summerin ääni ja LED-valot.

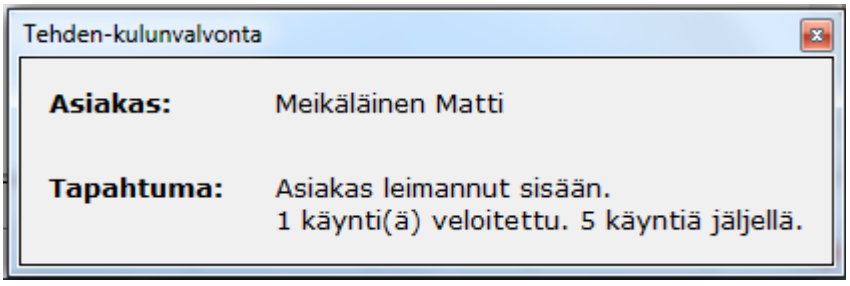
Onnistuneissa kortinluvuissa ennen vastauksen lähetystä kulunvalvontasovellukselle kortinlukutapahtuma tallennetaan Tehden APIn kautta tietokantaan. Kortin ollessa aikaperusteinen tai kappaleperusteinen, jossa on vaihtoaika vielä voimassa, leimauksesta tallennetaan tapahtuman tiedot tietokantaan. Jos kortti on kappaleperusteinen eikä siinä ole vaihtoaikaa enää voimassa, kortin arvosta vähennetään kulutettava määrä krediittejä.

Tietojen tallennus tapahtuu transaktiossa, joten jonkin asian mennessä vikaan tallennusvaiheessa tietokannan tiedot palautetaan takaisin. Tällöin Tehden API lähettää vastauksen kulunvalvontasovellukselle, jossa kerrotaan leimauksen epäonnistuneen.

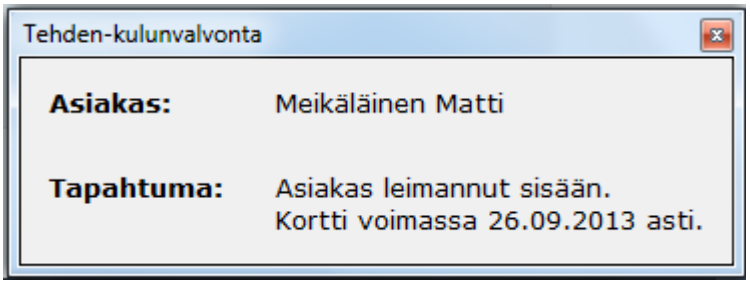
Tehden APIn lähetettyä vastaus ja kulunvalvontasovelluksen vastaanotettua se sovellus asettaa NFC-lukijan viestin ja summerin äänen. Kun lukijan viesti on asetettu, kulunvalvontasovellus lähettää tietoikkunalle oman viestin, jonka tietoikkuna näyttää tietokoneen näytöltä (Kuva 8, Kuva 9, Kuva 10). Lopuksi Tehden APIn lähettämän viestin tiedot tallennetaan sovelluksen omalle tietokannalle logevent-tauluun. Näitä tietoja käytetään silloin, kun tietoikkuna näyttää viimeisimpiä kulunvalvontatapahtumia.



Kuva 8. Ilmoitus, kun asiakkaan kortin tietoja ei löytynyt.



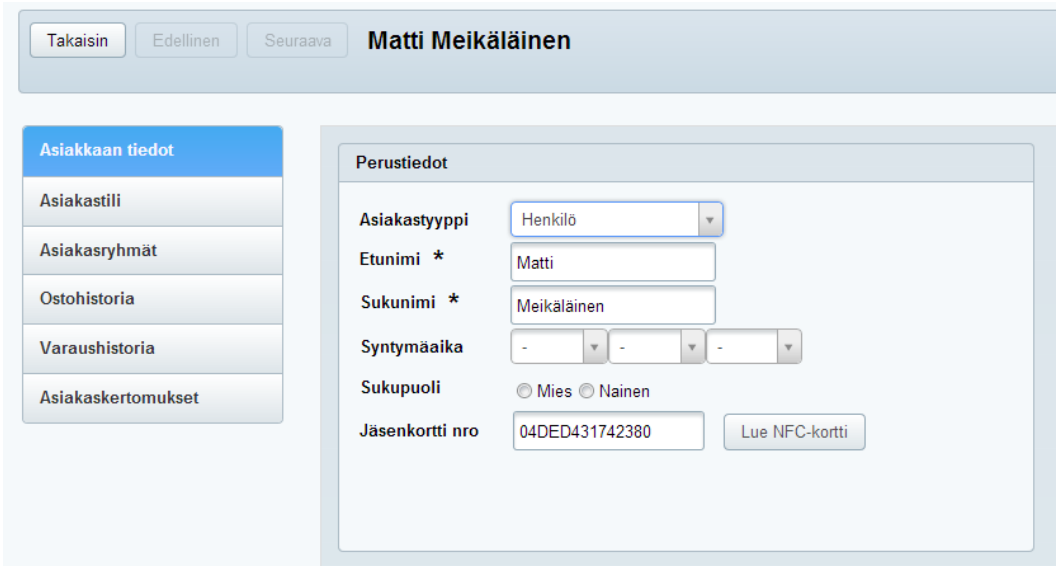
Kuva 9. Kappaleperusteisen kortin ilmoitus.



Kuva 10. Aikaperusteisen kortin ilmoitus.

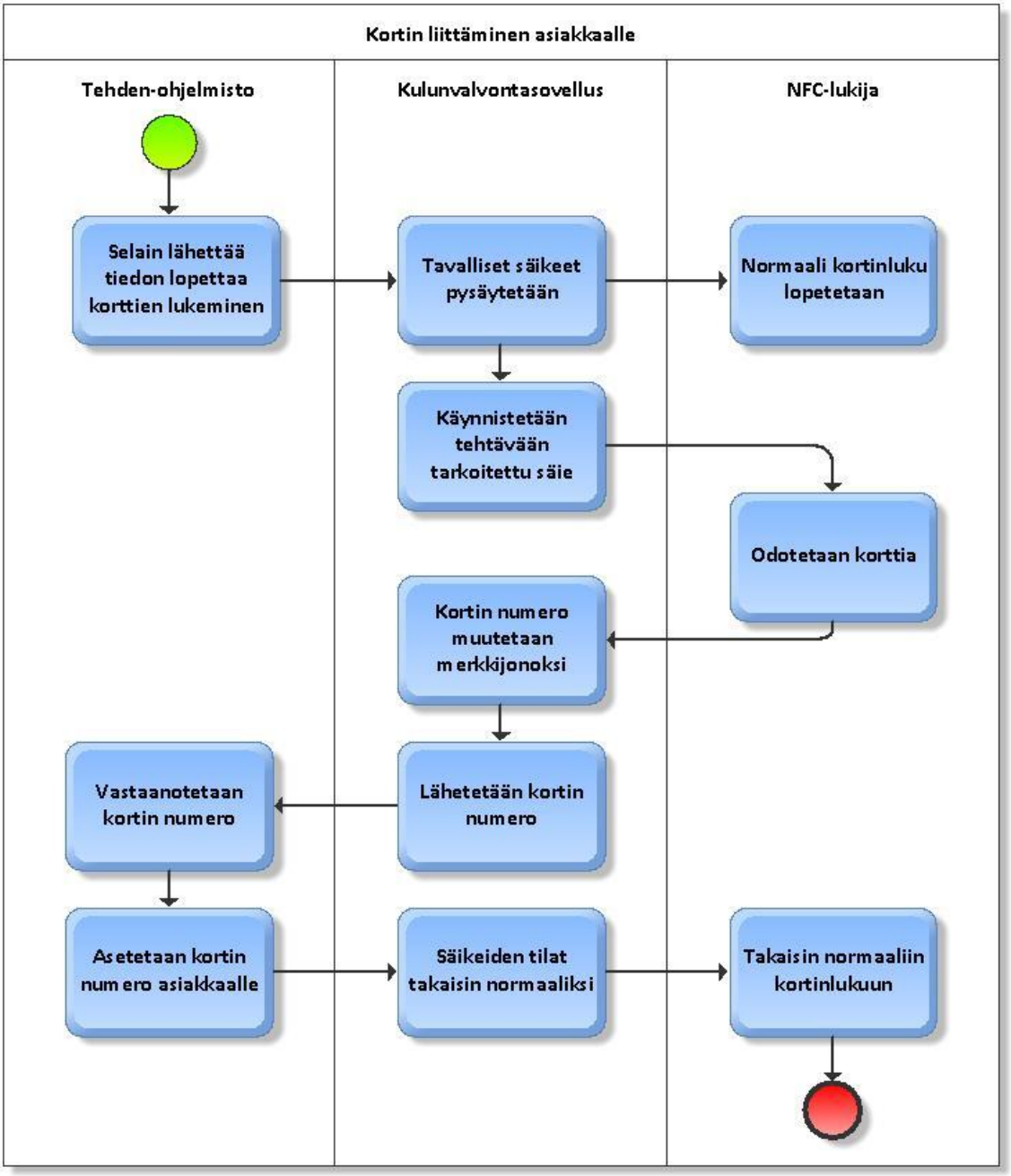
8.5 Kortin liittäminen asiakkaalle

Kulkukortin liittäminen asiakkaalle tapahtuu Tehden-ohjelmiston asiakasmoduulissa. Asiakkaalle on mahdollista asettaa kortin numero NFC-lukijan kautta asiakkaan tietokortilta (Kuva 11).



Kuva 11. Asiakkaan tietokortti.

Kortin liittämisen ajaksi kulunvalvonta pysäytetään hetkeksi, jotta kortti voidaan asettaa asiakkaalle (Kuvio 4). Tehden-ohjelmiston käyttäjä painaa Lue NFC-kortti -painiketta, jolloin kulunvalvonta lopetetaan. NFC-lukija jää odottamaan korttia niin kauan, että käyttäjä asettaa kortin lukijan läheisyyteen tai peruuttaa toiminnan. Kun kortti on luettu, sen numero ilmestyy lomakkeen jäsenkortti nro -kenttään ja kulunvalvonta palaa takaisin sen normaaliin toimintaan.



Kuvio 4. Kortin liittäminen asiakkaalle.

9 YHTEENVETO

Tehden-kulunvalvonnan toteutusprojektin alussa olin hieman epätietoinen, kuinka minun tulisi edetä siinä. Tämä siitä syystä, että kulunvalvonta oli tuttua minulle ainoastaan pääpiirteittäin. Lisäksi tietoni eri rajapintojen toiminnasta sekä ohjelmien ja laitteiden välisestä kommunikoinnista oli melko vähäistä. Myös kokemukseni Python-ohjelmointikielestä oli käytännössä jäänyt yhteen ammattikorkeakoulun opintokurssiin. Olin kuitenkin valmis opiskelemaan uusia asioita projektin toteuttamiseksi.

Projektin alkuvaiheessa minulle annettiin tarvittavat tiedot ja vaatimukset kulunvalvonnan toteuttamiseen, joten alun pienestä tietämättömyydestäni huolimatta sain projektin vähitellen aloitetuksi. Python-ohjelmointikielen opettelukaan ei loppujen lopuksi ollut kovin vaikeaa, vaan pääsin nopeasti ohjelmointikielestä jyvälle. Myös tietoni rajapintojen toiminnasta parantui projektin edetessä. NFC-teknologia ja sen toiminta oli minulle melko uusi asia ennen kulunvalvontaprojektia, mutta tietoni siitä ovat projektin jälkeen kasvaneet. Täysin uusi asia minulle oli OAuth 2.0 -ohjelmistokehys, jonka toiminnasta olen oppinut paljon uutta.

Projektin aikana Tehden-kulunvalvonnan vaatimukset ja tavoitteet muuttuivat hieman, mutta loppujen lopuksi sain toteutettua toimeksiantajan haluaman kulunvalvontakokonaisuuden. Sekä toimeksiantaja että Tehden-kulunvalvontaa käyttävät yritykset ovat olleet tyytyväisiä opinnäytetyön tuloksena syntyneeseen kulunvalvontaan. Opinnäytetyön kirjoitushetkellä Tehden-kulunvalvonta onkin jo käytössä muutamalla yrityksellä. Lisäksi kulunvalvontaa käyttävien yritysten määrä on ollut hiljalleen kasvamaan päin.

Tehden-kulunvalvontaa on tarkoitus kehittää tulevaisuudessa entistä paremmaksi ja toimivammaksi kokonaisuudeksi. Esimerkiksi kulunvalvonnan tämänhetkinen asennusprosessi on melko työläs ja prosessia onkin tarkoitus muuttaa enemmän automaattiseen muotoon erillisellä asennusohjelmalla. Tällöin jopa asiakkaiden olisi itse mahdollista suorittaa kulunvalvonnan asentaminen.

Nykyinen kulunvalvonta on myös hyvä pohja lähteä kehittämään uusia tulevaisuuden kulunvalvontatoteutuksia. Yksi kehitysajatus onkin ollut kosketusnäytöllä toimiva kulunvalvonta. Kosketusnäytön avulla voitaisiin helpottaa ja hoitaa jopa monimutkaisempia kulunvalvontatilanteita. Kosketusnäyttöä käyttäen esimerkiksi asiakkaat eivät tarvitsisi erillistä kulkukorttia, vaan he voisivat tunnistautua vaikkapa omalla sormenjäljellään.

LÄHTEET

Advanced Card Systems Ltd 2013. ACR1222L NFC Reader with LCD Technical Specifications. Viitattu 29.3.2013 http://www.acs.com.hk/drivers/eng/TSP_ACR1222L_v1.00.pdf.

Aussel J-D. 2009. Pyscard user's guide. Viitattu 27.4.2013 http://pyscard.sourceforge.net/pyscard-usersguide.html.

Ballad B.; Ballad T. & Banks E. 2010. Access control, authentication, and public key Infrastructure. Sudbury, MA: Jones & Bartlett Learning.

Hardt, E. 2012. The OAuth 2.0 Authorization Framework. Viitattu 27.3.2013 http://tools.ietf.org/pdf/rfc6749.pdf.

Miller, L. & Gregory, P 2012. CISSP For Dummies. Hoboken, NJ: John Wiley & Sons, Inc.

NearFieldCommunication.org 2013a. About Near Field Communication. Viitattu 27.5.2013 http://www.nearfieldcommunication.org/about-nfc.html.

NearFieldCommunication.org 2013b. How Near Field Communication Works. Viitattu 23.6.2013 http://www.nearfieldcommunication.org/how-it-works.html.

NFC Development & Consulting 2013. NFC. Viitattu 23.6.2013 http://www.nfc.cc/technology/nfc.

The PHP Group 2013a. PHP: What is PHP? Viitattu 19.8.2013 http://php.net/manual/en/intro-whatis.php.

The PHP Group 2013b. PHP: What can PHP do? Viitattu 19.8.2013 http://www.php.net/manual/en/intro-whatcando.php.

Popular Science 2011. Everything You Need to Know About Near Field Communication. Viitattu 16.3.2013 http://www.popsci.com/gadgets/article/2011-02/near-field-communication-helping-your-smartphone-replace-your-wallet-2010.

Python Software Foundation 2013. What is Python? Executive Summary. Viitattu 18.8.2013 http://www.python.org/doc/essays/blurb.

Rebel SimCard 2008. What is an APDU? Viitattu 25.3.2013 http://blog.rebelsimcard.com/what-is-an-apdu.htm.

SQLite 2013. About SQLite. Viitattu 29.3.2013 http://www.sqlite.org/about.html.

TeamViewer 2013. TeamViewer - maksuton etähallinta ja työpöydän etäjakaminen internetissä. Viitattu 31.8.2013 http://www.teamviewer.com/fi.

Tehden Oy 2013. Yritysesittely. Viitattu 8.3.2013 http://www.tehden.com/sites/default/files/yritysesittely_1.pdf.

Accesscontrolcore-moduulin kortinluku-funktio

```
def __read_card(self):
        """
        Connect card reader and set reader message, LEDs
        and buzzer based on response received from Tehden API.
        Save reading event to SQLite database.
        """
        self.automatic_card_read_event.clear()

        while not self.automatic_card_read_event.isSet():
            info('Automatic card reading')
            try:
                reader = CardReader()

                card_number = reader.get_cardnumber()

                messages = self.__handle_card_read(card_number)

                for message in messages:
                    reader.clear_lcd()
                    reader.set_lcd_message(0x00, 0x00, message['reader']['line_1'])
                    reader.set_lcd_message(0x00, 0x40, message['reader']['line_2'])

                    if self.do_buzz:
                        reader.set_buzzer(message['status'])
                    reader.set_led_state(message['status'])

                    try:
                        if self.infowindow:
                            self.infowindow.sendall(dumps(message['window']))
                    except Exception:
                        pass

                    db = Database()
                    db.start()
                    db.set_log_event(
                        card_number,
                        message['window']['event'],
                        message['window']['customer']
                    )
                    db.commit()
                    db.close()

                    self.automatic_card_read_event.wait(0.5)

                self.automatic_card_read_event.wait(3.5)

            except Exception:
                pass
```

Accesscontrolservice-moduuli

```
class AccessControlService(win32serviceutil.ServiceFramework):

    _svc_name_ = 'TehdenAccessControlService'
    _svc_display_name_ = 'Tehden Access Control Service'

    _svc_deps_ = ['LanmanWorkstation']

    def __init__(self, args):
        win32serviceutil.ServiceFramework.__init__(self, args)

        # Set the current working directory to be the same where the service is
        # executed from. Otherwise the log file will go to C:\Windows\System32
        os.chdir(os.path.abspath(os.path.dirname(sys.argv[0])))

        self.core = AccessControlCore()

    def SvcStop(self):
        self.core.stop()

    def SvcDoRun(self):
        self.core.start()


if __name__ == '__main__':
    # Note that this code will not be run in the 'frozen' exe-file!!!
    win32serviceutil.HandleCommandLine(AccessControlService)
```

Api-moduuli

```
class Api:

    def request(self, method_type, service_uri, uri_parameters):
        """
        Build either GET or POST request and it's parameters.
        Set request headers and send it with Oauth 2.0 token
        to Tehden API. If errors occurs during request try to
        fetch new access token by refresh token.
        """
        token = None

        try:
            db = Database()
            token = db.get_token()
            db.close()

            if token:
                url = token['base_uri'] + service_uri

                if method_type == 'GET':
                    request = urllib2.Request(url + '?' + urlencode(uri_parameters))
                else:
                    request = urllib2.Request(url, urlencode(uri_parameters))

                request.add_header('Accept', 'application/vnd.tehden.api-v1+json')
                request.add_header('Accept-language', 'fi')

                request.add_header(
                    'Authorization',
                    'Bearer ' + b64encode(token['access_token'])
                )

                response = urllib2.urlopen(request)
                return loads(response.read())
            else:
                info('API token error: ' + token)

        except urllib2.HTTPError as e:
            error = loads(e.read())

            exception('API error')
            info(error)

            if token and error['error'] == 'invalid_grant':

                parameters = {
                    'grant_type': 'refresh_token',
                    'refresh_token': token['refresh_token'],
                    'base_uri': token['base_uri']
                }

                oac = Oaclient()
                token_process = oac.request_token(parameters)

                if 'access_token' in token_process:
                    info('New access token requested successfully')
                    return self.request(method_type, service_uri, uri_parameters)
                else:
                    info('Error in new access token request')
```

Cardreader-moduulin alustusfunktiot

```
class CardReader():

    def __init__(self, mode='standby'):
        """
        Try connect to card reader. If reader isn't found
        try to reconnect reader indefinitely.
        """
        try:
            if mode == 'standby' or mode == 'manual':
                debug('Standby')
                readerConnected = None

                interval = 1

                while readerConnected is None:
                    try:
                        self.__init_cardreader(mode)
                        debug('Try to set reader to standby')
                        readerConnected = True

                    except Exception:
                        exception('Try to connect reader again')
                        sleep(interval)
                        if interval < 16:
                            interval *= 2
                        pass

                cardtype = AnyCardType()
                debug('Request card')
                cardrequest = CardRequest(timeout=5, cardType=cardtype)
                debug('Wait for card')
                self.cardservice = cardrequest.waitforcard()
                self.cardservice.connection.connect()
                debug('Card connected')

            else:
                debug('Stop')
                self.__init_cardreader()

        except Exception:
            pass

    def __init_cardreader(self, mode=None):
        """ Init card reader by setting LEDs, buzzer and default message. """
        r = readers()
        self.reader = r[0].createConnection()
        self.reader.setProtocol(SCARD_PROTOCOL_UNDEFINED)
        self.reader.connect(SCARD_PROTOCOL_UNDEFINED, SCARD_SHARE_DIRECT)

        self.__init_led()
        self.__init_buzzer()
        self.clear_lcd(True)

        if mode == 'standby':
            self.set_lcd_message(0x00, 0x00, 'Näytä kortti', True)
        elif mode == 'manual':
            self.set_lcd_message(0x00, 0x00, 'Manuaalinen', True)
            self.set_lcd_message(0x00, 0x40, 'kortinlukutila', True)

        self.reader.disconnect()
```

Database-moduulin alustusfunktiot

```
# Database's tables.
TABLES = {
    'logevent':
    'timestamp TIMESTAMP, '
    + 'event TEXT, '
    + 'cardnumber TEXT, '
    + 'customer TEXT',

    'tokensettings':
    'access_token TEXT, '
    + 'refresh_token TEXT, '
    + 'base_uri TEXT',
}


class Database:
    def __init__(self):
        """ Create if needed and connect to database.  """
        self.connection = sqlite3.connect(path.realpath('accesscontrol.db'))
        self.connection.isolation_level = None
        self.connection.row_factory = sqlite3.Row
        self.cursor = self.connection.cursor()

    def init_tables(self):
        """ Loop through tables and create them. """
        self.start()
        for table in TABLES:
            columns = TABLES[table]
            self.__create_table(table, columns)

        self.commit()

    def __create_table(self, table, columns):
        """ Create database tables.
        :param table: Table name.
        :param columns: Table's column names and types.
        """
        sql = 'CREATE table IF NOT EXISTS ' + table + \
              ' (id INTEGER PRIMARY KEY, ' + columns + ')'
        self.cursor.execute(sql)
```

Httpserver-moduulin GET-pyynnön käsittelyfunktio

```
class SecureHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        """
        Handle HTTP request received from the browser
        and request Oauth 2.0 token.
        """
        referer = self.headers.getheader('Referer')

        if referer:
            global keep_alive
            path = self.path

            if '?' in path:
                path, tmp = path.split('?', 1)
                qs = parse_qs(tmp)

                if 'connect' in qs and qs['connect'][0] == 'true':
                    self.__response(True)

                elif 'code' in qs and 'state' in qs:
                    status = False
                    code = qs['code'][0]
                    state = qs['state'][0]

                    if len(code) > 0:
                        parameters = {
                            'grant_type': 'authorization_code',
                            'code': code,
                            'base_uri': state,
                            'redirect_uri': 'https://localhost'
                        }

                        oaclient = Oaclient()
                        result = oaclient.request_token(parameters)

                        if 'access_token' in result:
                            keep_alive = False
                            status = True

                    self.__message(status)
```

Oaclient-moduuli

```
class Oaclient:

    def request_token(self, parameters):
        """
        Handle Oauth 2.0 access token fetching or updating.
        Save token to the database.
        :param parameters:
            array that contains information needed
            in fetching or updating token
        :return:
        """
        try:
            if parameters['grant_type'] == 'authorization_code':
                parameters['redirect_uri'] = 'https://localhost'

            request = urllib2.Request(
                parameters['base_uri']
                + '/oauth/oaclient/token?'
                + urlencode(parameters)
            )

            request.add_header(
                'Authorization',
                'Basic ' + b64encode('TehdenAccessControl:SSSHSUPERSECRET')
            )

            result = urllib2.urlopen(request)

            token = loads(result.read())

            if 'access_token' in token:
                token['base_uri'] = parameters['base_uri']

                db = Database()
                db.start()
                db.set_token(token)
                db.commit()
                db.close()

            return token

        except urllib2.HTTPError as e:
            error = loads(e.read())
            info('Oaclient error: '+error['error'])
            return error['error']
```

Infowindow-moduulin yhteys- ja viestifunktiot

```
def __infowindow_socket(self):
        """
        Init web socket connection between Tehden
        access control program and receive messages.
        """
        HOST = 'localhost'
        PORT = 50009
        ENCODING = 'cp1252'

        self.queue = Queue()

        while 1:
            try:
                s = socket(AF_INET, SOCK_STREAM)
                s.connect((HOST, PORT))

                while 1:
                    data = s.recv(1024)
                    if data and data != 'manual_card_read':
                        message = loads(data, encoding=ENCODING)
                        self.queue.put(message)
                    elif not data:
                        s.close()

            except Exception:
                pass

    def __show_message(self):
        """ Set message and open info window for a moment. """
        try:
            if not self.root.winfo_viewable() or self.table_frame.winfo_viewable():
                message = self.queue.get(0)
                self.__set_window_size(400, 125)
                self.root.after(0, self.__set_window_position)
                self.root.after(0, self.canvas.destroy)
                self.root.after(0, self.scrollbar.forget)

                self.message_frame = Frame(self.root)
                self.message_frame.grid()

                grid = [
                    ['Asiakas:', message['customer']],
                    ['Tapahtuma:', message['event'].replace("\\n", "\n") + "\n"]
                ]

                for row_no, row_data in enumerate(grid):
                    self.__set_grid_row(
                        self.message_frame,
                        row_no,
                        row_data,
                        self.MESSAGE_PADX,
                        self.MESSAGE_PADY,
                        'bold_first_col'
                    )

                self.root.after(0, self.root.deiconify)
                self.root.after(5000, self.root.withdraw)
                self.root.after(5000, self.message_frame.destroy)
                self.root.after(5000, self.__init_events_table)

            except Exception:
                pass

            self.root.after(250, self.__show_message)
```

Tehden Accesscontrol API Minimodelin kortinluku-funktio

```
/*
     * Handle card read checks
     * and return response message
     * to access control program.
     *
     * @param $cardNumber:
     * String that contains alphanumeric characters.
     *
     * @param $acPythonVersion:
     * String that contains Python program's version number.
     *
     * @return array:
     * Array contains:
     * - customer name
     * - info window message
     * - card reader message
     * - current status
     */
    public function handleCardRead($cardNumber, $acPythonVersion = FALSE) {

        $this->accessControlPythonVersion = $acPythonVersion;

        $card = $this->getCard($cardNumber);

        if (!$card) {
            return $this->setMessages('-', $this->dblang->line('card_not_found'));
        }

        $customer = $this->getCustomer($card->id);

        if (!$customer) {
            return $this->setMessages('-', $this->dblang->line('card_owner_not_set'));
        }

        if ($card->active == 'f') {
            return $this->setMessages($customer->name, $this->dblang->line('card_not_active'));
        }

        $groups = $this->getCardGroups($card->id);

        if (!$groups) {
            return $this->setMessages($customer->name, $this->dblang->line('card_access_denied'));
        }

        $isValid = $this->isCardValid($card->id);

        if (!$isValid) {
            $windowMessage = $this->dblang->line('card_expired');
            $readerMessage = $this->dblang->line('card_expired_short');
            return $this->setMessages($customer->name, $windowMessage, $readerMessage);
        }

        $currentAccounts = $this->getCurrentAccounts($card->id);

        if (!$currentAccounts) {
            return $this->setMessages($customer->name, $this->dblang->line('account_not_found'));
        }

        $reservations = $this->getReservations(time(), $customer->meta id);
        $filteredAccounts = $this->filterAccounts($reservations, $currentAccounts);
```

```
        if (!$filteredAccounts) {

            if ($currentAccounts[0]['isperiodic']) {
                $consumeAmount = NULL;
            }
            else {
                $consumeAmount = $this->getCardConsumeAmount(
                    $currentAccounts[0]['accountproduct_id']
                );

                if ($consumeAmount === FALSE) {
                    $windowMessage = $this->dblang->line('card_multiple_products');
                    $readerMessage = $this->dblang->line('sign_up_to_cashier');
                    return $this->setMessages($customer->name, $windowMessage, $readerMessage);
                }
        }

            $currentAccounts = $this->setUsableAccount($currentAccounts[0], $consumeAmount);
        }
        else {
            $currentAccounts = $filteredAccounts;
        }

        if ($this->isLatestVersion()) {
            $messages = array();
        }
        else {
            $messages = '';
        }

        $this->db->trans start();

        try {
            $this->usingAccount = TRUE;
            foreach ($currentAccounts as $currentAccount) {
                if ($this->isLatestVersion()) {
                    $messages[] = $this->useAccount($currentAccount, $customer, $card->id);
                }
                else {
                    $messages = $this->useAccount($currentAccount, $customer, $card->id);
                }
            }
        }
        catch (Exception $e) {
            $this->usingAccount = FALSE;
            $this->db->set_trans_status(FALSE);
            $this->db->trans complete();

            if ($this->isLatestVersion()) {
                return array($this->setErrorMessage($customer->name, $e->getMessage()));
            }
            else {
                return $this->setErrorMessage($customer->name, $e->getMessage());
            }

        }

        $this->usingAccount = FALSE;
        $this->db->trans_complete();

        return $messages;
```