KULUNVALVONTAJÄRJESTELMÄ PLC:LLÄ TOTEUTETTUNA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Sähkö- ja automaatiotekniikka, Valkeakoski

Syyslukukausi 2021

Sami Alakotila

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Sähkö- ja automaatiotekniikan insinööri Tiivistelmä

Valkeakoski

Tekijä Sami Alakotila Vuosi 2021

Työn nimi Kulunvalvontajärjestelmä PLC:llä toteutettuna

Ohjaajat Mika Oinonen, Janne Laari (Etteplan Finland Oy)

TIIVISTELMÄ

Projektin tavoitteena oli toimittaa Etteplan Finland Oy:lle kulunvalvontajärjestelmä käyttäen RFID tekniikkaa ja ohjelmoitavaa logiikkaa. Järjestelmän tulee olla helposti laajennettavissa ja muokattavissa. Kulunvalvontajärjestelmän avulla pystytään entistä paremmin kontrolloimaan kohteen runsasta henkilöliikennettä ja tehostamaan turvallisuutta olemalla tietoisia alueella olevasta henkilöstöstä.

Projektin toteutukseen kuului fyysisten laitteiden asennus ja PLC:n ohjelmointi. Ohjelmointiin sisältyi kommunikointi RFID lukijoiden kanssa, sekä niiltä saatavan tiedon kommunikointi SQL-tietokannan kanssa. Projektissa käytettiin TCP/IP-kommunikointia tietokannan ja RFID-lukijoiden välillä, vaikka tämä kommunikointimuoto on hyvin laaja-alainen ja haastava, liittyivät suurimmat haasteet kuitenkin datan käsittelyyn. Yksityiskohtaisimmillaan dataa jouduttiin käsittelemään bitti kerrallaan, joten ohjelman sulavan toiminnan varmistamiseksi riitti runsaasti haastetta.

Selkeän toimintatavan löydyttyä datan käsittelyyn pystyttiin luomaan selkeä toimintamalli, jota pystyttiin käyttämään hyväksi koodin eri osa-alueilla. Järjestelmän valmistuttua pystyttiin normaalikäytössä havaitsemaan ja korjaamaan ongelmia, joita ei simuloinnin ja testien aikana ilmennyt. Virhehistorian tallentaminen ja etäyhteys mahdollistivat välittömän reagoinnin ongelman tai muutostarpeen tullen.

Avainsanat PLC, RFID-tunnistus, SQL

Sivut 34 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Electrical and automation engineer **Abstract**

Valkeakoski

Author Sami Alakotila Year 2021

Subject Access control system on PLC

Supervisors Mika Oinonen, Janne Laari (Etteplan Finland Oy)

ABSTRACT

The goal of this project was to deliver an access control system for Etteplan Finland Oy by using RFID technology and a PLC. The system needed to be easily expanded and modified. With an access control system it is possible to have even better control over passenger traffic to the company and to improve common security by acquiring information on the personnel in the target area.

The physical equipment and programming of the PLC were included into the implementation of the project. Communication with RFID readers and the SQL database were included to programming. The TCP/IP protocol was used in the project between the database and the RFID readers. Despite of the wide scope of this type of a communication protocol the biggest challenges were related to data processing, it was needed to process the data one bit at a time. To ensure continuous functionality of the program there were a lots of challenges.

Finding a clear method for processing the data made it possible to develop a functional model that could be used in a different parts of the code. After commissioning the system, it was possible to detect and fix any problems that did not occur during simulation and testing. Using remote control and saving error history enabled reacting fast to any issues if necessary.

Keywords PCL, Radio frequency identification, SQL

Pages 34 pages

Sanasto

EPC	Electronic Product Code
HMI	Human-Machine Interface
LF	Low frequency
PLC	Programmable logic controller
RFID	Radio frequency identification
SQL	Structured query language
ST	Structured text
UHF	Ultra high frequency
UUID	Universally unique identifier
TCP	Transmission Control Protocol
TSPL-2	Tulostimen kommunikointikieli

Sisällys

1 Johdanto

Projektin lähtötilanne oli aloittelevan suunnittelijan näkökulmasta haastava, mutta hyvin mielenkiintoinen. Valmista kommunikointijärjestelmää ei eri laitteiden väliseen kommunikointiin ollut saatavilla, joten sellainen täytyi suunnitella. Tavoitteena oli siis suunnitella toimiva ja helposti muunneltavissa oleva järjestelmä, joka mahdollistaa kommunikoinnin eri ohjelmointikieltä puhuvien laitteiden välillä.

Kulunvalvontajärjestelmiä on mahdollista toteuttaa lukemattomilla eri tavoilla, sen laajuus ja toteutustapa tarkastellaan aina tapauskohtaisesti. Järjestelmää suunnitellessa voidaan siihen integroida käytännössä lähes mitä tahansa muita järjestelmiä, jotka saadaan helposti kommunikoimaan keskenään vakiintuneiden standardien ansiosta. PLC:llä toteutetussa järjestelmässä sen vahvuutena nousee esiin sen laajennettavuus, joka on tehty erittäin helpoksi verrattuna joihinkin eri järjestelmiin.

Kulunvalvontajärjestelmä on järkevää toteuttaa PLC:n avulla, kun tarvitaan hetkellisesti suuria määriä kulkulupia. Ohjelmaa ja järjestelmää voidaan muokata hyvin pitkälle ilman fyysisten komponenttien hankkimista. Tämä on järjestelmän käyttäjälle mielekästä, sillä järjestelmää pystytään helposti päivittämään kuluvan ajan tekniikan tasalle.

2 Kulunvalvontajärjestelmä

Kulunvalvontajärjestelmällä voidaan nimensä mukaan hallinnoida henkilö- tai ajoneuvoliikennettä. Kulunvalvontajärjestelmän tarkoitus on pohjimmiltaan estää ihmisten pääsy alueelle tai rakennukseen, joilla ei ole lupaa siellä kulkea, sekä olla tietoinen alueella tapahtuvasta liikenteestä. Tämä on tärkeää esimerkiksi tulipalon sattuessa, järjestelmän avulla tiedetään, kuinka monta henkilöä on evakuoitavana. Esimerkiksi pelastushenkilökunnalle voidaan heti osoittamaa etsintäalue, jossa mahdollisesti kadonneen henkilön oletetaan olevan.

Kulunvalvontajärjestelmä voi yksinkertaisimmillaan olla esimerkiksi sähkölukko, joka aukeaa oikealla PIN-koodilla. Tällainen järjestelmä on yksinkertainen, eikä siinä ole älyä. Järjestelmää voidaan tehostaa liittämällä siihen esimerkiksi liiketunnistimia tai kameroita. Liiketunnistimet liittyvät yleisesti rikosilmoitusjärjestelmään, jolla havaitaan tai ehkäistään mahdollinen kiinteistöön tai omaisuuteen liittyvä ilkivalta.

Kulunvalvonnan päätelaitteet, joissa ei ole älyä joutuvat aina lähettämään tunnistetiedon esimerkiksi kulkukortista prosessoitavaksi keskukselle. Keskus päättää ohjelmoinnin perusteella voidaanko lupa kulkea myöntää, ja lähettää sen perusteella oven lukituksen avauskäskyn kohdeovelle. Oven lukko ja lukija ovat kaapeloituna tähteen, jolloin kaapelointia vaaditaan huomattava määrä. Älykkäämmissä järjestelmissä voidaan käyttää kenttäväylää, jolloin kaapeloinnin määrä pienenee. Kenttäväylällä toteutetuissa järjestelmissä lukija osaa itse päättää sallitaanko oven avaus. Väylään asennettuna laitteilla tulee olla oma tunnisteensa, joka voi olla IP osoite verkon yli toimivissa järjestelmissä, tai esimerkiksi dippikytkimillä asetettava staattinen osoite. (ISONAS, n.d.)

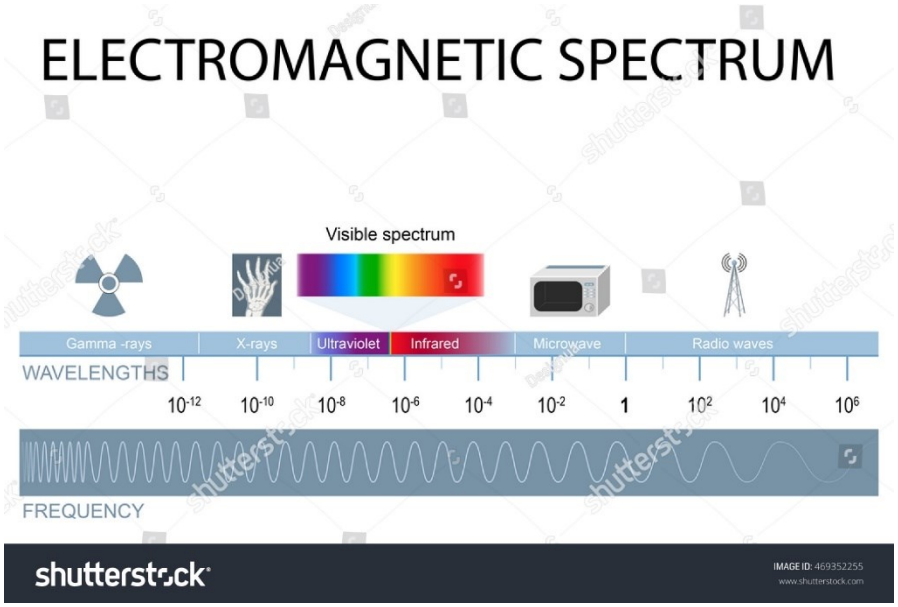
Kulunvalvontaa voidaan suorittaa monella eri tapaa, mekaanisilla, sähköisillä tai biometrisillä järjestelmillä, tai niiden yhdistelmillä. Kamerat tuovat erinomaisen lisän kulunvalvontaan, sillä tallennettu kuva auttaa ratkaisemaan epäselviä tilanteita. Myös auton rekisterikilven tunnistusominaisuus on erittäin käyttäjäystävällinen, se ei vaadi henkilöltä erillisiä toimia, vaan toimii täysin automaattisesti. Tässä projektissa toteutettu RFID-kulkulupajärjestelmä toimi rekisterikilven tunnistuksen rinnalla.

3 RFID-teknologia

Radioaallot ovat sähkömagneettisia aaltoja, jotka kuuluvat sähkömagneettiseen spektriin. Sähkömagneettinen spektri on jaettu useampaan osaan aallonpituuden perusteella Kuva 1 mukaisesti. Lyhyimmät aallonpituudet ovat gammasäteilyä ja pisimmät radioaaltoja, myös näkyvä valo kuuluu sähkömagneettiseen spektriin. Radioaaltojen värähtelynopeutta

mitataan hertseinä (Hz), joka tarkoittaa aaltojen lukumäärää per sekunti. (National Radio Astronomy Observatory, 2014)

Kuva 1. Sähkömagneettinen spektri jaoteltuna eri aallonpituuksiin. (Shutterstock, n.d.)



Myös radiotaajuudet on jaettu moneen ryhmään, sillä eri taajuuksia käytetään eri tarkoituksiin. Kuva 2 on esitetty taajuusalueet jaoteltuna eri luokkiin, aallonpituuksista voidaan päätellä, että matalat taajuudet kulkevat pidemmälle kuin korkeat. Korkeissa taajuuksissa on suurempi amplitudi, mutta ne myös vaimenevat nopeammin. Tästä esimerkkinä on 5G-taajuus, joka sijoittuu UHF-alueelle. 5G-taajuus vaatii lähetysasemia tiheämmin kuin esimerkiksi radiokanavien lähetinasemat, nämä taajuudet sijoittuvat LF alueelle. (National Radio Astronomy Observatory, 2014)

Kuva 2. Radioaallot jaettuna eri taajuuksille. (ITM Components, 2018)

Name	Symbol	Frequency Range	Wavelength
Extremely Low Frequency	ELF	3 Hz - 30 Hz	10,000 km – 100,000 km
Super Low Frequency	SLF	30 Hz - 300 Hz	1,000 km – 10,000 km
Ultra Low Frequency	ULF	300 Hz - 3 kHz	100 km – 1,000 km
Very Low Frequency	VLF	3 kHz - 30 kHz	10 km – 100 km
Low Frequency	LF or LW	30 kHz - 300 kHz	1 km – 10 km
Medium Frequency	MF or MW	300 kHz – 3,000 kHz	100 m – 1 km
High Frequency	HF or SW	3 MHz – 30 MHz	10 m – 100 m
Very High Frequency	VHF	30 MHz – 300 MHz	1 m – 10 m
Ultra High Frequency	UHF	300 MHz – 3,000 MHz	10 cm – 100 cm
Super High Frequency	SHF	3 GHz – 30 GHz	1 cm – 10 cm
Extremely High Frequency	EHF	30 GHz – 300 GHz	1 mm – 10 mm

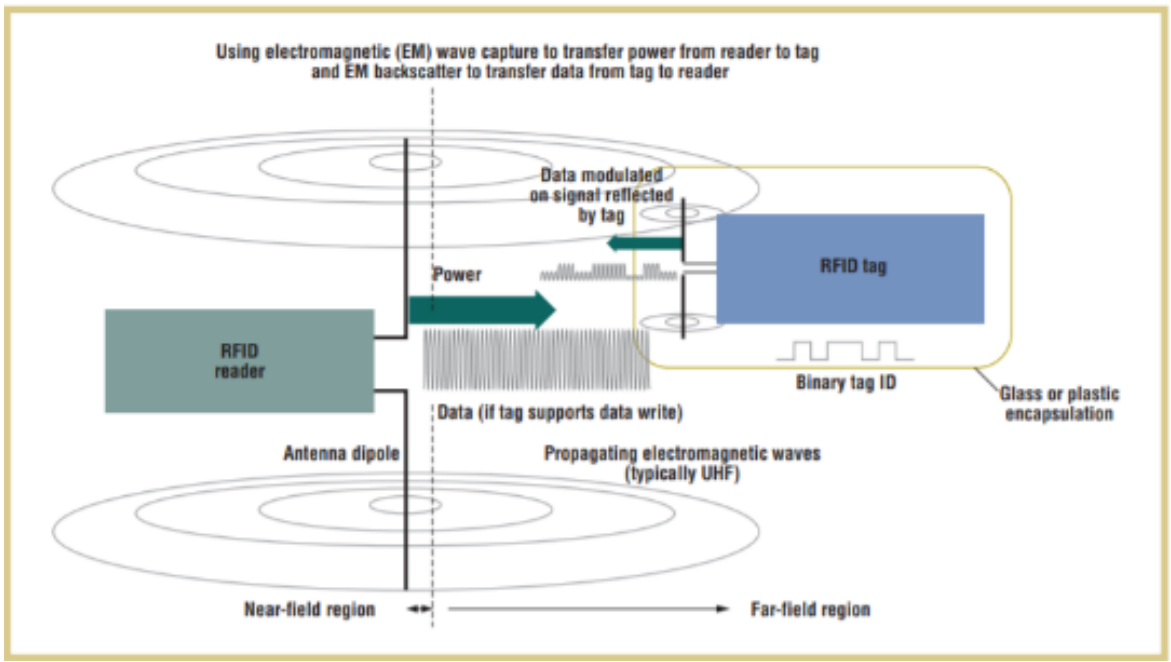
Sähkömagneettisilla aalloilla on positiivinen ja negatiivinen puolijakso, joten yksi aallonpituus käsittää nämä molemmat. Esimerkiksi 50 Hz:n taajuus tarkoittaa, että yhden sekunnin aikana mitataan 50 aallonpituutta. Radioaalto syntyy sähkö- ja magneettikentän värähtelystä, värähtelyn taajuus on suoraan verrannollinen radioaallon pituuteen. Mitä matalampi värähtelytaajuus, sen pidempi aallonpituus ja päinvastoin. Kun sähkökenttä heikkenee, aiheuttaa se samalla magneettikentän vahvistumisen. Kun sähkökenttä ei enää voi heikentyä, on magneettikenttä saavuttanut maksimiarvonsa ja alkaa heiketä kasvattaen jälleen sähkökenttää. Sähkökentän vuorottaisesta voimistumisesta ja heikentymisestä syntyy radioaallon positiivinen ja negatiivinen puoliaika. (National Radio Astronomy Observatory, 2014)

RFID teknologia perustuu radioaaltojen lähettämiseen ja vastaanottamiseen yhdessä RFID-sirun kanssa. Tällä menetelmällä voidaan tehokkaasti ja erittäin nopasti identifioida

henkilöitä, kulkuneuvoja tai materiaalia. RFID-siru voidaan integroida tuotteeseen jo valmistusvaiheessa, se voidaan koteloida, tai kiinnittää esimerkiksi tarralla. Verrattuna esimerkiksi viivakoodin lukemiseen, RFID on ylivoimaisesti monipuolisempi tapa tunnistuksessa. RFID:n lukemiseen ei vaadita suoraa näköyhteyttä sirun ja antennin välillä, sillä radioaallot kulkevat esteiden läpi. Siru voidaan lukea useiden metrien päästä, jolloin tunnistusta varten ei tarvitse erikseen pysähtyä. RFID-siruja voidaan myös lukea kymmeniä, satoja tai jopa tuhansia kerralla, joka tekee siitä erittäin tehokkaan tavan tunnistaa esimerkiksi laatikossa ajoneuvossa tai vaikkapa rahtikontissa olevia siruja ilman lastin purkua. (RFIDLab Finland ry, 2021)

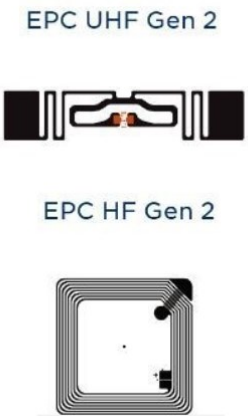
RFID-siru koostuu antennista ja mikrosirusta, toimiakseen se tarvitsee kuitenkin virtaa. Aktiivisessa sirussa on oma paristo tai akku, jonka avulla siru voi lähettää jatkuvasti signaalia, vaikka sitä vastaanottavaa antennia ei olisikaan lähistöllä. Passiivisessa sirussa ei ole paristoa tai akkua, vaan se saa toimintaansa vaadittavan energian itse radioaallosta. Kuva 3 on havainnollistettu tekniikka, jolla tässä projektissa käytettävä passiivisen tunnisteen luku toimii. Lukija lähettää radioaaltoja tietyllä taajuudella, tunnisteen antenni vastaanottaa signaalin ja saa signaalin tehoa hyväksi käyttäen virtaa mikropiirille. Se prosessoi tapahtuman ja heijastaa signaalin takaisin lukijalle, jolle voidaan lukijan mallista riippuen määrittää useita eri toimintoja, joita se suorittaa tapahtuman aikana. (RFID4u, 2021)

Kuva 3. RFID-lukijan ja -tunnisteen välinen kommunikointi. (University of Colorado, 2007)



RFID-tekniikkaa käytetään laajasti eri sovelluksissa ympäri maailmaa, se on ollut olemassa kauan, joten tekniikka on kehittynyttä ja sen valmistajia on paljon. Radiotaajuudet ovat laajasti käytössä, joten tietyt sovellukset on standardoitu eri taajuuksille. RFID-tekniikan käyttö UHF-alueella 860 MHz – 960 MHz määritellään ISO 18000-63-standardissa. Tässä projektissa käytettävät tunnisteet on valmistettu GS1-standardin mukaan. EPC-tunnisteissa on yksilöllinen sarjanumero, eli GS1-tandardiin pohjautuvia tunnisteita ei kahta samanlaista ole olemassa. Projektissa käytettävä tunniste on Kuva 4 esitetty EPC UHF Gen 2-tyyppi, jossa molemmilla sivuilla olevat antennit yhdistyvät keskellä sijaitsevaan mikropiiriin. (ISO, 2015; GS1 Finland, n.d.)

Kuva 4. GS1-standardin mukaiset EPC-tunnistetyypit. (Kuva: GS1 Finland, n.d.)



4 Ohjelmointikielet

4.1 ST ja TSPL-2

Projektissa toteutettu koodi on kirjoitettu ST-kielellä, joka kuuluu IEC 61131-3-standardiin ja on maailmalla laajasti käytetty korkeamman tason ohjelmointikieli, joka perustuu sveitsiläisen professori Niklaus Wirthtin kehittämään pascal-ohjelmointikieleen. ST-kieli on tekstipohjainen ohjelmointikieli, jolla pystytään vaivattomasti toteuttamaan monimutkaisimmatkin järjestelmät. Koodia luetaan vasemmalta oikealle ja ylhäältä alas rivi kerrallaan. Koodia lukeva ohjelma ei kuitenkaan itse tiedä koska luettava rivi päättyy, joten se täytyy kertoa sille erikseen. ST-kielessä rivin päättymistä indikoi puolipiste, mikäli

puolipistettä ei käytä, ohjelma lukee kyseistä riviä loputtomiin, mikä johtaa virheeseen. (PLCopen, n.d.; ISO, 1990)

Vaikka itse koodi on kirjoitettu ST-kielellä, sillä ei kuitenkaan pystytä suoraan hallitsemaan kaikkien laitteiden toimintaa. RFID-tarran tulostin ei ymmärrä ST-kieltä, vaan se käyttää tulostimille kehitettyä TSPL-2-ohjelmointikieltä. Ongelmaan löytyy helppo ratkaisu, tulostimelle lähetetään TCP-yhteyden avulla komento, joka on kirjoitettu TSPL-2-kielellä. Toisin sanoen ST-kielellä toteutetussa koodissa kirjoitetaan tulostimelle tarkoitettu komento STRING-muuttujaan TSPL-2-kielellä. Tekstimuuttuja muunnetaan BYTE-muotoon, jotta se voidaan lähettää toiseen IP-osoitteeseen. Vastaanottava pää eli tässä tapauksessa tulostin ei siis lue tekstimuuttujaa, vaan BYTE-muuttujan. Tulostin purkaa viestin sille ymmärrettävään muotoon, näin voidaan ST-kieltä hyväksi käyttäen käyttää laitteita, joissa on eri ohjelmointikieli.

Nykyaikaiset sivunkuvauskielet kehitettiin 1980 luvulla, joista ensimmäisiä oli vielä tänä päivänäkin käytössä oleva Adoben 1982 kehittämä PostScript. Tätä ennen käytössä olevia tulostustekniikoita oli piirturit, joilla pystyttiin tulostamaan graafisia kuvia, sekä tulostimet, joiden toiminta perustui vanhan kirjoituskoneen toimintaan. Paperille pystyttiin tulostamaan vain tekstiä tällä tekniikalla, jossa musteleima iski merkin kerrallaan paperiin. (Canon, 2013)

Sivunkuvauskielissä, kuten tässä projektissa käytettävä TSPL-2 ei itse koottua tulostuskäskyä prosessoida PLC:llä. Eri tulostimet käyttävät eri kieliä, joiden avulla ne osaavat tulostaa halutun näköisen sivun. Vaikka tulostin itse prosessoi saadun tulostuskäskyn, jokainen kirjain, numero, merkki tai kuva, joka halutaan tulostaa, on sisällytettävä käskyyn. Tulostuskomennon tulee sisältää myös jokaisen erillisen merkin koko ja koordinaatti tulostettavalla alustalla. Poikkeuksena tekstirivi, jolle tarvitsee osoittaa vain sen aloituspiste, josta merkkijono alkaa. (Brady, 2020)

4.2 SQL

SQL kehitettiin 1970-luvulla IBM:n (International Business Machines Corporation) toimesta prototyyppinä tietokannan hallintaa varten. SQL:ää on sen julkaisemisesta lähtien kehitetty lisää, ja siitä on luotu useita eri versioita, kuten MySQL ja SQLite. SQL standardoitiin jo hyvin

aikaisessa vaiheessa, ja tekniikan, tietomäärän ja tarpeen käsitellä sitä kasvaessa standardia on useasti päivitetty nykyaikaan. (Laine, 2002)

SQL on siis ohjelmointikieli, jolla voidaan tehdä kyselyjä ja tallentaa dataa tietokantaan. Kun tietokantaan halutaan tallentaa dataa tai lukea sitä sieltä, on lähetettävässä komennossa määritettävä mitä dataa halutaan käsitellä. Datan käsittelyyn löytyy lukemattomia komentoja riippuen halutusta lopputuloksesta, yksi yleinen komento on SELECT, jonka avulla luetaan dataa. Otetaan esimerkkinä taulukko, jossa on useita sarakkeita, esimerkiksi etunimi, sukunimi, ID ja kuukausittainen bruttopalkka. Käyttäen SELECT-komentoa voidaan hakea taulukosta jokainen rivi, jossa ID-arvoa ei ole tallennettu seuraavalla komennolla:

```
SELECT * FROM dbo.EsimerkkiTaulukko WHERE ID IS NULL
```

Tämä komento hakee dbo.EsimerkkiTaulukko-nimisestä taulukosta kaikki mahdolliset rivit, joiden ID-sarakkeeseen ei ole tallennettu dataa. Komennolla INSERT INTO voitaisiin puolestaan tallentaa dataa taulukoihin, tallennettaessa tulee aina ilmetä mihin taulukkoon tallennetaan, mihin sarakkeisiin tallennetaan ja mitä tallennetaan. Kaikkia taulukon rivejä ja sarakkeita ei ole pakko käsitellä, vaan eri komentoja käyttäen voidaan käsitellä myös yksittäisiä taulukon soluja.

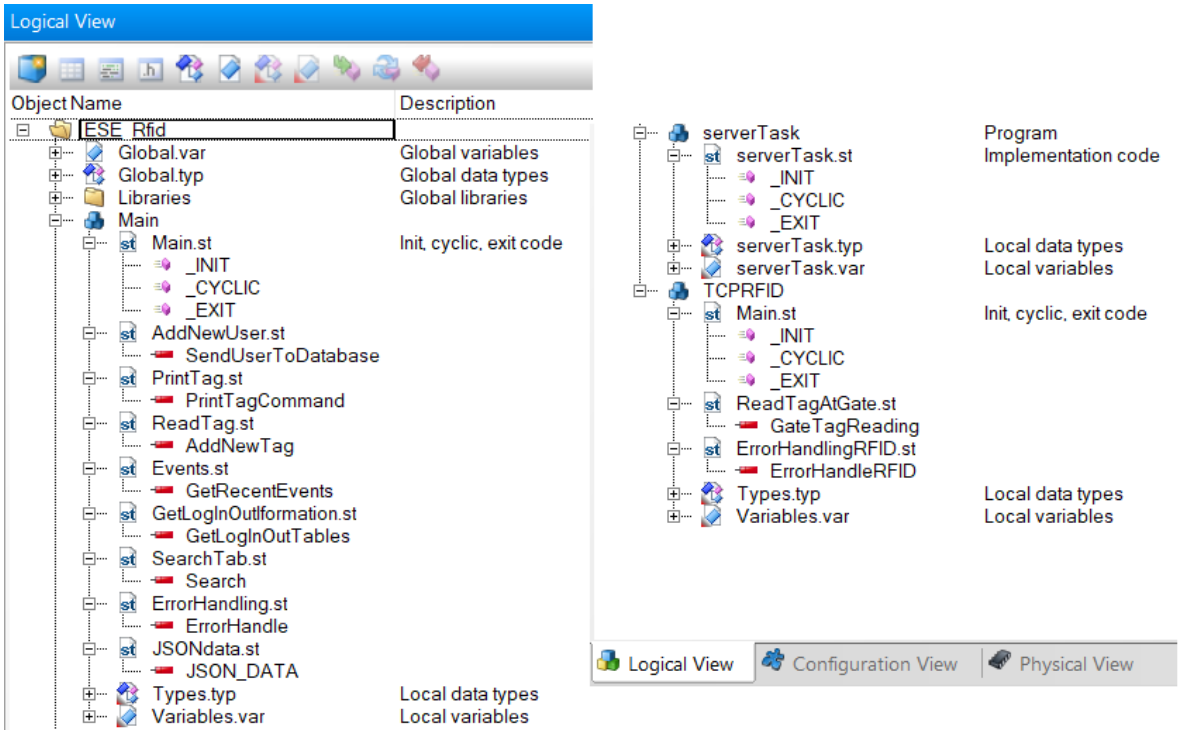
SQL-tietokantaan pystytään taulukoiden lisäksi tallentamaa myös kuvia, excel-taulukoita ja monenlaisia muita datatyyppejä. Kuvia tallennettaessa ei kuitenkaan voida lähettää kuvaa JPG, PNG, tai muissa kuvamuodoissa. Kuvan tallentaminen tapahtuu binäärimuodossa taulukon soluun, solun tyyppi on määritetty tähän tarkoitukseen sopivalla muuttujalla varbinary(max).

5 PLC:n ohjelmointi

5.1 Rakenne

Koodi on jaettu useampaan eri osioon ohjelmaan helppolukuisuuden ja toiminnallisuuden vuoksi. Koodissa pyörii kolme eri ohjelmaa, pääohjelma, HMI:n kommunikoinnin parametrit, sekä RFID:n luku portilla. Koodiin kuuluvat ohjelmat on esitetty Kuva 5. Pääohjelma eli Main vastaa taulukoiden ja muiden tietojen lähettämisestä HMI:lle, sekä uusien käyttäjien rekisteröimisestä tietokantaan. ServerTask vastaa web-pohjaisen HMI:n parametrien asettamisesta, tätä ohjelmaa kutsutaan ohjelmassa sadan millisekunnin välein TCP-yhteyden ollessa asynkroninen. Liian nopealla syklillä kutsuminen on verrattavissa palvelunestohyökkäyksiin, joissa ruuhkautetaan esimerkiksi pankin sivut yhtäkkisellä ryöpyllä tietoliikennettä. RFID-ohjelmaa eli TCPRFID kutsutaan myös sadan millisekunnin syklissä. Tässä ohjelmassa tapahtuu itse henkilöliikenteen valvominen pääportilla, TCP-kommunikointi tapahtuu yhden RFID-lukijan kanssa, johon on kiinnitetty kaksi antennia toinen sisään- ja toinen uloslukua varten.

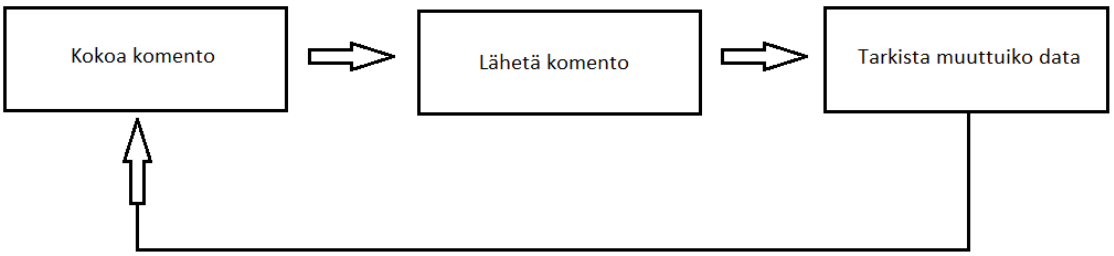
Kuva 5. Koodissa pyörivät ohjelmat ja niiden koodi jaettuna eri osuuksiin.



Koska HMI on WEB-pohjainen, sitä voi käyttää useampi käyttäjä samaan aikaan. Tämän vuoksi myös yhteyksiä SQL-tietokantaan tarvitaan useampi, yksi jokaiselle HMI:n välilehdelle, yksi RFID-lukijalle pääportilla, sekä kaksi koodin virheenkäsittelyyn. HMI:n eri toiminnot, kuten uuden käyttäjän lisääminen, käyttäjän poistaminen ja livetilanteen tarkastelu on kaikki myös koodissa kirjoitettu omiin osioihin, jota kutsutaan koodin Main-osiossa.

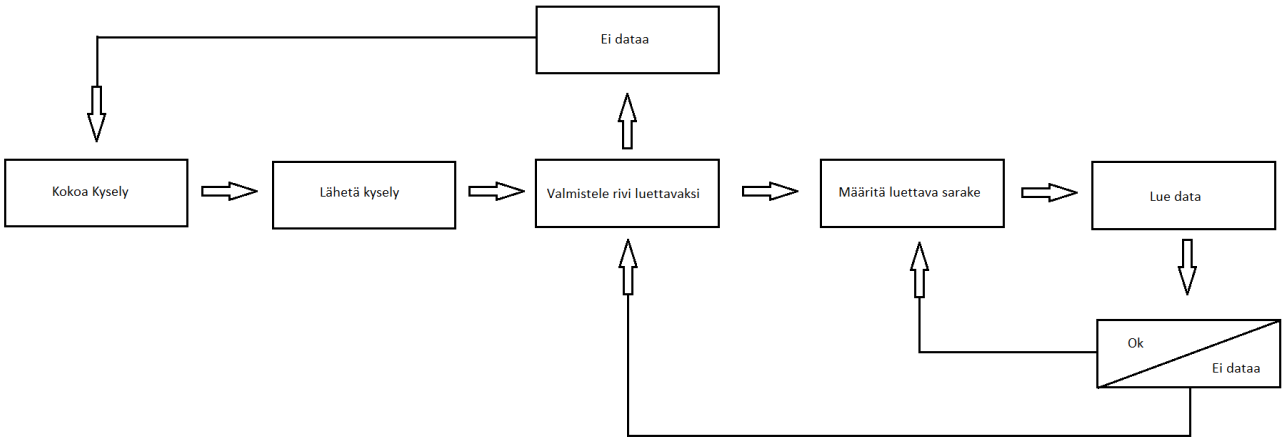
Datan tallentaminen ja sen lukeminen SQL-tietokannan kanssa tapahtuu melkein kaikissa toiminnoissa samalla periaatteella. Tiedon lähettäminen on nopea kolmivaiheinen prosessi, tieto lähetetään STRING-muuttujana, joten komento täytyy ensin muodostaa. Kun lähetettävä komento on muodostettu, se lähetetään tietokantaan. Seuraava vaihe on tarkastaa, mikäli tieto tallentui kohdetaulukkoon, tämä suoritetaan tarkastelemalla muuttuneita rivejä. Tiedon tallentamisen vaiheet on havainnollistettu Kuva 6

Kuva 6. Tiedon tallentaminen tietokantaan.



Datan lukeminen tietokannasta on monimutkaisempaa ja vie enemmän aikaa. Kuten datan lähettämisessä, sen lukemiseenkin muodostetaan ensimmäisenä komento, jossa määritellään mitä dataa halutaan lukea ja mistä. Komento datan kyselystä lähetetään tietokantaan, mutta vastaus ei tule valmiina pakettina vaan se täytyy erikseen lukea. Onnistuneen kyselyn jälkeen täytyy valmistella rivi luettavaksi, jonka jälkeen määritellään riviltä luettava sarake. Vasta näiden valmistelujen voidaan tämä yksittäisen rivin yhden sarakkeen data lukea ja tallentaa sitä varten tehtyyn muuttujaan. Kun data on luettu, siirrytään määrittämään seuraava sarake luettavaksi. Tällä tavalla luetaan kaikki sarakkeet, kunnes luettavaa dataa ei enää ole jäljellä. Kuvan 7 askeleiden mukaisesti siirrytään sarakkeiden luvun jälkeen valmistelemaan seuraava rivi luettavaksi, ja edelleen sarakkeen määrittelyyn. Kun data on luettu kaikilta riveiltä ja sarakkeilta, palataan takaisin alkuun.

Kuva 7. Tiedon tallentaminen tietokantaan ja sen lukeminen sieltä.



5.2 Koodin Main-osio

Main.st koostuu kolmesta osasta, PROGRAM_INIT, PROGRAN_CYCLIC ja PROGRAM_EXIT osista. Init-osassa toteutetaan muuttujien alustus komennot, jotka suoritetaan vain kerran ohjelman alussa, tämän jälkeen ohjelma siirtyy suorittamaan cyclic-osaa. Init-osioon kirjoitetaan kaikki, mikä halutaan toteutuvan ilman eri toimintoja käyttäjän puolelta. Vakiona kaikki muuttujat saavat arvon nolla, eikä esimerkiksi STRING-muuttujassa ole mitään tekstiä kirjoitettuna. Osa SQL-tietokantaan lähetettävistä kyselyistä alustetaan init-osiossa, jossa asetetaan myös TCP-toimilohkojen Enable-muuttuja arvoon TRUE.

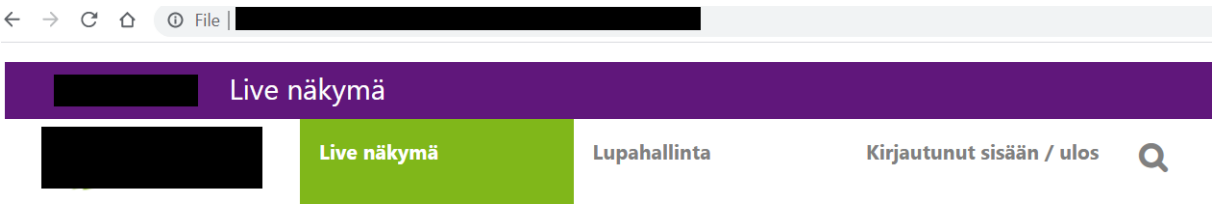
Cyclic-osiossa nimensä mukaisesti kutsutaan koodia ohjelman asetetulla aikavälillä, joka ohjelmassa Main on 10 ms. Main.st:ssä kutsutaan kaikkia koodin muita osioita, jotka ovat Main-ohjelman alla. Kuva 8 esitetään miten Main:stä eriytettyä koodin osaa kutsutaan. Ensimmäisenä näkyy JSON_DATA, tämä osio siirtää dataa HMI:n ja PLC:n välillä aina, kun muuttujien arvot muuttuvat. Tämän jälkeen määritellään muuttujat vastaamaan HMI:ltä tulevia tarvittavia muuttujia PLC:n puolelle, sekä toimiston RFID-lukijan ja tulostimen TCP-yhteyksien parametrit. Tässä kohtaa määritetään vain parametrit, TCP-toimilohkoja ei vielä tässä kohtaa kutsuta. Viimeisenä kutsutaan ErrorHandle-osiota, jossa suoritetaan pääohjelma Main:n virheenkäsittely.

Kuva 8. Koodin erillisen osan kutsuminen ja TCP-toimilohkojen parametrien määrittäminen.

```
100
101     // Call HMI server
102     JSON_DATA;
103
104     // HMI mapping
105     UserInfoVar.AuthorizationDate := JSONVariables.AuthorizationDate;
106     UserInfoVar.CompanyName[0] := JSONVariables.CompanyName[0];
107     UserInfoVar.FirstName[0] := JSONVariables.FirstName[0];
108     UserInfoVar.LastName[0] := JSONVariables.LastName[0];
109     UserInfoVar.JobTitlle[0] := JSONVariables.JobTitle[0];
110     UserInfoVar.SatartDate := JSONVariables.StartDate;
111     UserInfoVar.ValidUntil[0] := JSONVariables.ValidUntil[0];
112
113     // Printer TCP parameters
114     TCPCommsPrinter.IP := '193.208.172.151';
115     TCPCommsPrinter.Port := 9100;
116     TCPCommsPrinter.TimeoutTime := 5;
117     TCPCommsPrinter.EnableConnection := PrinterTCP;
118     TCPCommsPrinter.SendData := SendByte151;
119     TCPCommsPrinter.SendDataString := CommandStringToPrinter;
120     TCPCommsPrinter.TypeString := TRUE;
121
122     // RFID for new user tag reading tcp parameters
123     TCPCommsRFID153.IP := '193.208.172.153';
124     TCPCommsRFID153.Port := 4333;
125     TCPCommsRFID153.TimeoutTime := 5;
126     TCPCommsRFID153.EnableConnection := RFIDEnable153;
127     TCPCommsRFID153.SendData := SendByte153;
128     TCPCommsRFID153.SendDataString := CommandStringToRFID153;
129     TCPCommsRFID153.TypeString := FALSE;
130
131     ErrorHandle;
132
```

Koska HMI:tä voi käyttää moni henkilö samanaikaisesti, voidaan myös tietokantaan liikennöidä samanaikaisesti. Tätä varten on luotava useampi yhteys SQL-tietokantaan, joten jokaiselle koodin osiolle on luotu oma yhteys tietokantaan. Kuva 9 on esitetty neljä eri välilehteä, joita operaattori voi HMI:llä käyttää, livenäkymä, lupahallinta, kirjautunut sisään/ulos ja etsi (suurennuslasi).

Kuva 9. HMI:n välilehdet.



Jos esimerkiksi livenäkymä välilehdellä käyttäjä tekee muutoksen, joka käynnistää kyselyn tietokantaan, kestää aikaa ennen kuin datan lukeminen tietokannasta on saatu suoritettua loppuun. Tämän ajan livenäkymän yhteys on varattu, eivätkä muut käyttäjät saa käyttää

HMI:tä samaan aikaan. Mikäli dataa luetaan tai kirjoitetaan tietokantaan, välilehdelle ilmestyy ”Ladataan” ikkuna, joka estää kaiken muun toiminnan kyseisellä välilehdellä. Jokaisella välilehdellä ollessa oma yhteys tietokantaan, ei yhdellä välilehdellä tapahtuvat muutokset vaikuta muiden toimintaan.

Jokaisen välilehden toiminnot on kirjoitettu koodissa eri osioihin ja näitä kutsutaan Main:ssä. Ennen osion kutsua täytyy kuitenkin luoda yhteys tietokantaan, jotta HMI voi toimia oikein. Jokaisen välilehden yhteys tietokantaan luodaan Kuva 10 mukaisesti, yhteyden luominen koostuu kahdesta osasta. Ensimmäisessä osassa (CALL_CONNECT_FB_INIT) asetetaan tarvittavat parametrit yhteyden muodostamista varten; toimilohkon salliminen, käyttäjätunnus, salasana, IP-osoite ja tietokannan nimi. Muuttujat ovat tekstimuuttujia, mutta niitä ei voi toimilohkolle suoraan tekstimuuttujana määrittää, koska viestintä tapahtuu TCP-yhteyden kautta. Tällaisessa tilanteessa käytetään muistipaikan osoitusta, joka Kuva 10 näkyy funktiona ADR().

Kuva 10. Yhteyden luominen SQL tietokantaan

```
135         // CALLING ACTIONS RELATED TO HMI PAGE 1
136         CASE Actions1 OF
137
138             CALL_CONNECT_FB_INIT:   // Connection parameters for database communication related to HMI page 1
139
140                 DatabaseFBs.ConnectToDatabase1.enable := TRUE;
141                 DatabaseFBs.ConnectToDatabase1.pUserName := ADR(ConnectToDatabaseVar.UserName);
142                 DatabaseFBs.ConnectToDatabase1.pPassword := ADR(ConnectToDatabaseVar.Password);
143                 DatabaseFBs.ConnectToDatabase1.pServerName := ADR(ConnectToDatabaseVar.ServerName);
144                 DatabaseFBs.ConnectToDatabase1.pDatabaseName := ADR(ConnectToDatabaseVar.DatabaseName);
145                 DatabaseFBs.ConnectToDatabase1.databaseSystem := DB_SYSTEM_MS_SQL;
146                 IF BlockConnection1 = 0 THEN
147                     Actions1 := CALL_CONNECT_FB;
148                 END_IF;
149
150             CALL_CONNECT_FB: // Call connect to database FB
151
152                 DatabaseFBs.ConnectToDatabase1();
153                 IF DatabaseFBs.ConnectToDatabase1.status = 0 THEN
154                     ConnectToDatabaseVar.ConnectionIdent1 := DatabaseFBs.ConnectToDatabase1.connectionIdent;
155                     Actions1 := CALL_ACTIONS;
156                 ELSIF DatabaseFBs.ConnectToDatabase1.status <> 65535 THEN
157                     ErrorConnectionIdent := DatabaseFBs.ConnectToDatabase1.connectionIdent;
158                     GetError := 1;
159                 END_IF;
160
161             CALL_ACTIONS:
162
163                 GetRecentEvents;
164
165         END_CASE;
166
```

Muistipaikan osoittamisessa datan tyypillä ei ole väliä, sillä muistista osoitetaan vain tietty alue, jolta data luetaan. Muistipaikan osoitusta käyttämällä ohjelmoijan täytyy itse tietää, minkä tyypin muuttujan osoittaa lähetettäväksi. Toisessa osassa kutsutaan toimilohkoa, joka muodostaa yhteyden. Mikäli yhteyden muodostaminen onnistui, tallennetaan yhteys-ID erilliseen muuttujaan ja siirrytään kutsumaan actionia. Mikäli Kuva 10 näkyvästä

CALL_CONNECT_FB-osiosta ei siirrytä pois, koko ajan kutsuttaessa yhteyden muodostuksen toimilohkoa muodostaisi se jatkuvasti uusia yhteyksiä. Virheen sattuessa olisi mahdollista, että uusi yhteys luotaisiin ennen kuin virheenkäsittely saataisiin suoritettua, joka puolestaan johtaisi virheeseen yhteys ID:n kanssa.

Jokaiselle yhteydelle on määritetty kuvan 11 mukainen laskuri, joka virhetilanteessa estää actionin kutsumisen 30 sekunnin ajan. Tämän jälkeen muodostetaan uusi yhteys ja yritetään uudestaan. 30 sekuntia on määritelty, jotta tietokantaan ei lähetettäisi samaa virheviestiä ohjelmakierron nopeudella.

Kuva 11. Laskuri joka estää hetkellisesti koodin suorittamisen virhetilanteessa.

```
292     // Wait 30 sec after error occured in code related to HMI page 1 before calling actions
293     TP_1(IN := BockTimer1, PT := T#30s);
294     IF TP_1.Q = 1 THEN
295         BlockConnection1 := 1;
296         BockTimer1 := 0;
297     ELSE
298         BlockConnection1 := 0;
299     END_IF;
```

5.3 Livenäkymä ja viimeaikaiset tapahtumat

Tässä osassa koodia käytetään DatabaseFBs.ConnectToDatabase1-yhteyttä, joka hoitaa HMI:n ensimmäisen välilehden, eli livenäkymän kommunikointia tietokannan kanssa. Livenäkymä koostuu kolmesta osasta:

- tällä hetkellä sisään kirjautuneet,
- viimeaikaiset tapahtumat,
- statistiikka.

Tällä välilehdellä on vain muutama toiminto, jotka operaattori pystyy suorittamaan. Molemmat tällä hetkellä sisällä kirjautuneet, sekä viimeaikaiset tapahtumat-osio ovat taulukoita. Näissä taulukoissa sisään kirjautuneissa näytetään 50 nimeä, ja viimeaikaiset tapahtumat-taulukossa 25 tapahtumaa kerrallaan, jotka ovat järjestetty kirjautumisajan mukaan nousevassa järjestyksessä. Molempia taulukoita voi selata enemmän valitsemalla nuolinäppäimistä seuraava, tai edellinen välilehti. Nuolinäppäintä painamalla HMI lähettää PLC:lle USINT-muuttujan arvon, joka määrittää tietokantaan lähetettävän komennon.

Esimerkiksi sisään kirjautuneiden välilehti numero nolla hakee tietokannasta kaikki sisään kirjautuneet välillä 1-50 ja välilehti yksi hakee välillä 51-100. Mikäli haettavaa dataa ei ole, taulukko näkyy tyhjänä.

Events.st lähtee suorittamaan toimintoja koodin WAIT_FOR_EVENT_REFRESH-kohdassa, jos operaattori painaa HMI:n yläreunasta päivityspainiketta, jos jommankumman taulukon välilehteä muutetaan, tai jos pääportilla tapahtuu sisään-, tai uloskirjautuminen. Jonkun näistä toiminnoista tapahduttua, siirtyy koodi etsimään FORM_EVENT_QUERY_TO_DATABASE-kohtaan oikeaa komentoa FOR-lauseella. FOR-lause tarkastelee taulukoiden nykyistä välilehteä, ja valitsee sen perusteella pääohjelman INIT-osiossa ennalta määritellyn komennon. Tässä kohtaa lähetetään valittu komento tietokantaan, mikäli virhettä ei tapahdu siirtyy koodi valmistelemaan ensimmäistä riviä luettavaksi PREPARE_NEXT_EVENT_ROW-osioon. Kun tietokanta antaa komentoa vastaan monta riviä dataa, rivin valmistelu tehdään useaan kertaan. Kun luettavia rivejä ei enää ole, toimilohko antaa siitä virheen. Tästä virheestä ei koskaan mennä koodin virheenkäsittelyyn, sillä se on haluttu virhe. Tämä virhe päättää tietokannasta lukemisen, tai siirtää koodin hakemaan seuraavan taulukon dataa, riippuen suoritetusta toiminnosta HMI:llä tai portilla.

DETERMINE_EVENT_COLUMN_INDEX-osiossa määritellään luettava sarakeindeksi, joka on aina rivin ensimmäisellä lukukerralla yksi. Parametrien määrityksen jälkeen toimilohkoa, joka lukee datan tietokannasta, kutsutaan READ_EVENT_VALUES-osiossa. Tässä osassa koodia tallennetaan tietokannasta luettu data muuttujaan, joka on linkitetty HMI:lle. Kuva 12 on esitetty koodi, jossa dataa luetaan kerrallaan joko sisään kirjautuneiden, tai viimeaikaisten tapahtumien taulukkoon. Tilanne, jossa operaattori on valinnut päivitettäväksi koko välilehden, eikä vain jompaakumpaa taulukkoa, käydään koko datan lukemissykli kahteen kertaan peräkkäin molempien taulukoiden muuttujilla erikseen. Kun saatavilla oleva data on luettu, siirrytään määrittämään seuraava sarake luettavaksi. Uudelleen määritelty sarake luetaan jälleen READ_EVENT_VALUES-kohdassa, joka kattaa koko Kuva 12 näkyvän koodin, tätä toistetaan, kunnes uusia sarakkeita ei enää ole. Sarakkeiden loppuessa on määritettävä uusi rivi luettavaksi, ja sykli alkaa taas alusta.

Kuva 12. Datan lukeminen tietokannasta ja tallentaminen muuttujiin.

```
DatabaseFBs.GetData1();
IF DatabaseFBs.GetData1.status = 0 THEN
    // Only wanted columns are read from database, these columns are determined in Main.st init section
    IF JSONVariables.RefreshLoggedInTabsMain = 1 THEN
        IF GetDataVar.ColumnIndex1 = 1 THEN
            JSONVariables.HMITablesInAtm.ID[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 2 THEN
            JSONVariables.HMITablesInAtm.FirstName[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 3 THEN
            JSONVariables.HMITablesInAtm.LastName[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 4 THEN
            JSONVariables.HMITablesInAtm.Company[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 5 THEN
            JSONVariables.HMITablesInAtm.JobTitle[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 6 THEN
            JSONVariables.HMITablesInAtm.LoginTime[RowNumber1] := GetDataVar.Data1;
        END_IF;
    ELSIF JSONVariables.RefreshEventTabs = 1 THEN
        IF GetDataVar.ColumnIndex1 = 1 THEN                                     // Column 1 in dbo.ESERFIDEvents is ID
            JSONVariables.HMITablesEvents.ID[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 2 THEN                                  // Column 2 in dbo.ESERFIDEvents is first name
            JSONVariables.HMITablesEvents.FirstName[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 3 THEN                                  // Column 3 in dbo.ESERFIDEvents is last name
            JSONVariables.HMITablesEvents.LastName[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 4 THEN                                  // Column 4 in dbo.ESERFIDEvents is company name
            JSONVariables.HMITablesEvents.Company[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 5 THEN                                  // Column 5 in dbo.ESERFIDEvents is job title
            JSONVariables.HMITablesEvents.JobTitle[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 8 THEN                                  // Column 8 in dbo.ESERFIDEvents is action
            JSONVariables.HMITablesEvents.Action[RowNumber1] := GetDataVar.Data1;
        ELSIF GetDataVar.ColumnIndex1 = 9 THEN                                  // Column 9 in dbo.ESERFIDEvents is time of action
            JSONVariables.HMITablesEvents.Time[RowNumber1] := GetDataVar.Data1;
        END_IF;
    END_IF;
    GetDataVar.ColumnIndex1 := GetDataVar.ColumnIndex1 + 1;
    GetEvents := DETERMINE_EVENT_COLUMN_INDEX;
ELSIF DatabaseFBs.GetData1.dbError = -1 THEN                                    // If no data go fetch new row to be rad
    RowNumber1 := RowNumber1 + 1;
    GetEvents := PREPARE_NEXT_EVENT_ROW;
ELSIF DatabaseFBs.GetData1.status <> 65535 THEN
    ErrorConnectionIdent := ConnectToDatabaseVar.ConnectionIdent1;
    ErrorNumber := DatabaseFBs.GetData1.status;
    GetError := 1;
```

Taulukoisen päivittämisen jälkeen jäljelle jää statistiikka, jossa näytetään, kuinka moni työntekijä on kirjautuneena ulos ja sisälle. Tässä näkymässä näytetään myös sisään kirjautuneet yrityskohtaisesti. Statistiikan osalta datan lukeminen tietokannasta tapahtuu samalla tavalla kuin taulukoiden kanssa, eri yrityksiä voi olla enintään 51. Kyselyn muodostamisessa haetaan vain yrityksen nimi, sekä IN/OUT-status, jotka järjestetään yrityksen nimen mukaan. Datan lukemisen ensimmäisellä syklillä luetaan yrityksen nimi, ja toisella syklillä status tieto IN/OUT. Jokaisella datan lukemisen syklillä verrataan yrityksen nimeä edelliseen tallennettuun nimeen, jolloin yrityksen nimi tallennetaan vain, jos se vaihtuu. Yrityksien sisään ja ulos kirjautuneet henkilöt saadaan lisäämällä UINT-muuttujiin aina yksi, kun status tieto on luettu.

5.4 Uuden käyttäjän lisääminen

5.4.1 Käyttäjän tallennus tietokantaan

Tässä osassa koodia operaattori pystyy lisäämään uusia käyttäjiä. Käyttäjän lisäämistä varten on täytettävä HMI:n tekstikentän henkilötieto-osuus. Vaadittavat henkilötiedot ovat etunimi, sukunimi, yrityksen nimi, työtehtävä, aloituspäivä, lopetuspäivä ja luvan myöntämispäivä. HMI ei anna operaattorin lähettää tietoja, jos yksikin kenttä on tyhjänä. Kun käyttäjätiedot on lisätty, lähetetään tietokantaan ensin kysely käyttäjä ID:stä, kysely hakee tietokannasta vain ID-arvot. Tällä kyselyllä etsitään ensimmäinen vapaa ID-arvo, joka uudelle käyttäjälle on mahdollista lisätä. Alin mahdollinen ID-numero on yksi, joten vertailua varten ennen kyselyä alustetaan IDNo-muuttuja arvoon yksi. Kun rivi on valmisteltu luettavaksi, siirrytään lukemaan dataa. Dataa luettaessa verrataan tietokannasta saatua ID-arvoa ennalta alustettuun IDNo-muuttujaan. Mikäli arvot vastaavat toisiaan, on kyseinen ID-numero varattu, jolloin siirrytään valmistelemaan seuraavaa riviä datan lukemista varten ja lisätään vertailumuuttujaan +1. Tällä menetelmällä ensimmäinen vertailumuuttujasta eriävä arvo joka tietokannasta luetaan, tallennetaan uuden käyttäjän ID-numeroksi.

ID-luvun tallentamisen jälkeen siirrytään kasaamaan komentoa, joka lisää uuden rivin tietokantaan, jossa on juuri HMI:lle syötetyt käyttäjätiedot. Kun komento on kasattu, lähetetään se tietokantaan ja varmistetaan että tiedot tallentuivat. Tämä varmistus tapahtuu lähettämällä kysely, joka tarkastelee muutoksia riveissä. Tässä vaiheessa uudella käyttäjällä ei vielä ole RFID-tarraa, joten UUID (RFID tunnisteen uniikki arvo) -arvoa ei ole vielä määritetty. Käyttäjätietojen tallentamisen jälkeen lähetetään kysely, joka hakee tietokannasta kaikki käyttäjät, joilla ei ole UUID-arvoa. Tyhjä kenttä taulukossa näkyy tekstinä ”NULL”, tätä käytetään ehtona käyttäjätietojen hakemiselle. Tiedot päivitetään taulukkoon, josta operaattori pystyy valita nimen tulostettavaksi RFID-tarraan. Ennen tarran tulostamista on viimeinen mahdollisuus muokata käyttäjätietoja, mikäli tietojen lisäämisessä tapahtui esimerkiksi kirjoitusvirhe. Tässä vaiheessa on myös mahdollista poistaa juuri lisätty käyttäjä.

5.4.2 RFID-tarran tulostaminen

RFID-Tarra tulostetaan Brady:n BBP12 lämmönsiirtotulostimella, joka käyttää TSPL2-kieltä. Tulostettava tarra on tavallinen tarra, johon on integroitu RFID-siru. Tulostimelle kommunikoidaan TCP/IP-yhteyden avulla, sen IP-osoite on staattinen ja se on määritetty koodin Main-osiossa tulostimen TCP-toimilohkon parametreihin. Jotta tarra voidaan tulostaa, täytyy valita taulukosta nimi, jolle tarra tulostetaan. Ohjelma ei anna tulostaa tyhjää tarraa, vaan ilmoittaa ponnahdusikkunalla, että käyttäjän on valittava taulukosta nimi. Tarraan tuleva teksti muodostetaan Kuva 13 osoitetulla tavalla, tarran ensimmäiselle riville tulee koko nimi ja yritys. Ensimmäisen rivin pituus voi vaihdella, jonka vuoksi sen pituutta on tarkasteltava. Liian pitkä yhdistelmä oletusfontilla jättää osan tekstistä pois, sillä tulostin ei itse osaa skaalata tekstiriviä niin, että koko teksti mahtuu yhdelle riville. Koodi laskee tekstimuuttujan pituuden ja valitsee sen mukaan fontin koon niin, että kaikki teksti mahtuu yhdelle riville.

Kuva 13. RFID tarraan tulostettavien tietojen kasaus ja lähetys tulostimelle.

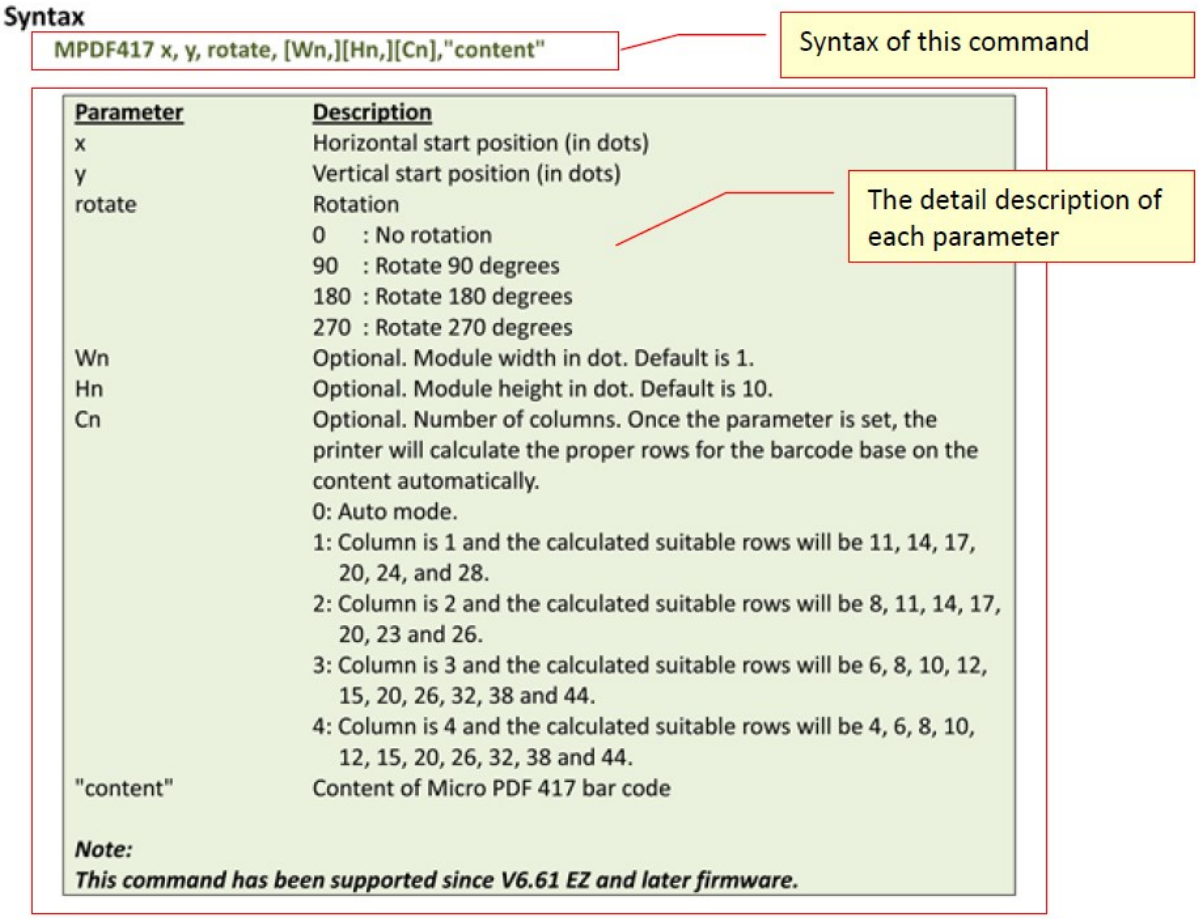
```
   CommandStringToPrinter := '';
   StringLen := 0;
   StringLen := LEN (UserInfoVar.FirstName[0]);
   StringLen := StringLen + LEN (UserInfoVar.LastName[0]);
   StringLen := StringLen + LEN (UserInfoVar.CompanyName[0]);
   IF StringLen < 25 THEN                                                    // Change txt size
       strcat (ADR (CommandStringToPrinter), ADR ('CLS$nTEXT 40,35,"5",0,1,1,"')); // CLS clears image
   ELSIF StringLen >= 25 AND StringLen < 29 THEN
       strcat (ADR (CommandStringToPrinter), ADR ('CLS$nTEXT 40,35,"4",0,1,1,"'));
   ELSE
       strcat (ADR (CommandStringToPrinter), ADR ('CLS$nTEXT 40,35,"3",0,1,1,"'));
   END_IF;
   strcat (ADR (CommandStringToPrinter), ADR (UserInfoVar.FirstName));
   strcat (ADR (CommandStringToPrinter), ADR (SQLCommands.Space));
   strcat (ADR (CommandStringToPrinter), ADR (UserInfoVar.LastName));
   strcat (ADR (CommandStringToPrinter), ADR (SQLCommands.Space));
   strcat (ADR (CommandStringToPrinter), ADR (UserInfoVar.CompanyName));
   strcat (ADR (CommandStringToPrinter), ADR ('"$nTEXT 40,130,"5",0,1,1,"Pursialan Voimalaitos"$n'));
   strcat (ADR (CommandStringToPrinter), ADR ('TEXT 40,215,"5",0,1,1,"'));
   strcat (ADR (CommandStringToPrinter), ADR (UserInfoVar.ValidUntil));
   strcat (ADR (CommandStringToPrinter), ADR ('"$nPRINT 1$n'));
   IF JSONVariables.InteractButtons[1] = 1 THEN                              // Go to call TCP F
       Printer := PRINT_TEXT;
   END_IF

RINT_TEXT: // Send string to printer and return back to wait for new printing

   TCPCommsPrinter();
   IF TCPCommsPrinter.TcpSend_0.status = 0 AND TCPCommsPrinter.TcpSend_0.enable = 1 THEN   // Checks i
       JSONVariables.InteractButtons[1] := 0;
       FOR i := 0 TO 19 DO
           IF ReadTagPermission[i] = '' THEN                                 // Saves us
               ReadTagPermission[i] := UserInfoVar.ID;
               EXIT;
           END_IF;
       END_FOR;
       Printer := FORM_PRINT_TEXT;
   ELSIF TCPCommsPrinter.Status = Error THEN
       ErrorType := 'TCP printer';
       ErrorMsgToDatabaseTCP := TCPCommsPrinter.ErrorDiag;
       GetError := 3;
       Printer := 0;
   END_IF;
```

Tekstin alkuun tulee aina laittaa ”CLS”, tämä tyhjentää tulostimen välimuistin ja varmistaa, että tarraan tulostetaan vain lähetettävän komennon tiedot. Kaikkia tarran rivejä ja komentoja ei voida kirjoittaa muuttujaan yhteen putkeen, vaan yksi komento tai rivi täytyy aina päättää kirjaimiin $n. Tämä tarkoittaa komentorivin päättymistä, jonka avulla tulostin osaa erotella eri toiminnot toisistaan. Jokainen uusi rivi tekstiä vaatii omat koordinaatti- ja fonttitietonsa, nämä tiedot on selostettu Kuva 14. Kun tarran tiedot on koottu lähetettäväksi tekstimuuttujaksi, kirjoitetaan vielä loppuun ”PRINT 1$n”, joka on itse tulostuskomento yhdelle kopiolle. Tulostimen statusta täytyy kysyä erillisellä komennolla, joten se ei anna automaattisesti tietoa, että on vastaanottanut tulostuskomennon. Jotta voidaan varmistaa, että TCP-toimilohko ei lähetä jatkuvasti tulostuskäskyä tulostimelle vaan vain yhden kerran, täytyy tarkastella toimilohkon sisällä tapahtuvaa toimintaa. Toimilohkoa kutsuttaessa tarkastellaan lähettävän osuuden statusta, eli kun toimilohkon datan lähetys ei enää ole ”Busy”, eli kiireinen, voidaan todeta datapaketin olevan lähetetty.

Kuva 14. Syntaksi tulostuskomentoon sijoitettavista parametreista. (Kuva: Brady, 2020)



5.4.3 RFID-tarran lukeminen tietokantaan

Tarran lukemiseen täytyy operaattorin valita nimi samasta listasta, josta valittiin tiedot tarran tulostukseen. RFID-lukija pystyy suorittamaan monia eri toimintoja, tässä tapauksessa käytetään yksinkertaista tarran lukemista, joka lukee tarran UUID-arvon ja lähettää sen PLC:lle välittömästi sen havaittuaan. Lukijan suoritettava toiminto täytyy määrittää lukijalle bitti kerrallaan Kuva 15 mukaisesti, jossa lähetetään lukijalle komento lukea yksi RFID-tunniste. Kaikki suoritettavat toiminnot löytyvät suoraan lukijan käyttöohjeesta. Tarran tulostuksen yhteydessä tallennetaan käyttäjän ID-muuttujallistaan, jota käytetään luvan myöntämiseen tarran lukemisessa. Operaattorin klikatessa ”Lue tarra” painiketta, tarkastellaan FOR-lausekkeen avulla, löytyykö kyseinen käyttäjä listalta. Tällä varmistetaan, että vain käyttäjän RFID-tarra, jonka tiedot on jo tulostettu, voidaan lukea tietokantaan.

Kuva 15. Lukijalle lähetettävä komento yhden tunnisteen lukua varten.

```
// Determine command to read a single tag
SendByte153[0] := 16#A5;
SendByte153[1] := 16#05;
SendByte153[2] := 16#00;
SendByte153[3] := 16#00;
SendByte153[4] := 16#00;
SendByte153[5] := 16#5F;
SendByte153[6] := 16#30;
SendByte153[7] := 16#F4;
SendByte153[8] := 16#01;
SendByte153[9] := 16#1D;
SendByte153[10] := 16#C6;
```

Jos käyttäjälle löytyi lupa lukea UUID-arvo, siirrytään vasta tässä vaiheessa Lukemaan RFID-tarran UUID:tä. Kun lukija lähettää PLC:lle tiedon löydetystä UUID-arvosta, koostuu viesti aina samalla tavalla Kuva 16 mukaisesti. Siinä näkyy muun muassa datapaketin pituuden tieto, antennin ID, UUID-arvo sekä CRC eli rivin lopetus merkki. Koska datapaketti koostuu aina samalla tavalla, voidaan bitistä yksi lukea datapaketin pituus ja vähentää siitä kaiken muun tiedon pituus, joka ei kuulu UUID-arvoon. Tämä laskettu INT-arvo on UUID-arvon pituus. UUID-arvo alkaa aina samasta kohtaa datapakettia, se tallennetaan muuttujaan käyttämällä FOR-lauseketta, joka lukee arvoja alkaen bitistä 11 lasketun INT-arvon verran. Kun UUID on tallennettu, lisätään se olemassa olevan käyttäjän tietoihin. Päivitys käyttäjätietoihin lähetetään tietokantaan, ja tehdään kysely rivien muuttumisesta, jolla varmistetaan tiedon tallentuminen. TCP-toimilohkolta saapuva datapaketti nollataan

kutsumalla sitä ja varmistamalla, että RFID-tarroja ei ole enää havaittavissa. Tällä varmistetaan, että uutta tarraa lukiessa jo luetun tarran tiedot eivät ole muistissa ja päästä koodia eteenpäin sen tiedoilla. Tämän jälkeen kyseinen RFID-tarra on valmiina käytettäväksi pääportin sisään- ja uloslukijoissa.

Kuva 16. Lukijan lähettämä datapaketti luettaessa yksi RFID tunniste. (Kuva: NordicID, 2017)

RESPONSE PACKET CONTENTS

Field	Value	Description	
Header	A51300000049 (6 bytes)	Header consisting of:	
		A5	
		1300 = 0x0013 (19)	Payload + CRC length
		0x0000	Command flags
		0x49	Header check sum
Command	0x30	Command echo	
Status	0x00	0 = OK	
Antenna	0x00	Antenna ID where the tag was scanned from.	
RSSI	0xCC	Signed 8-bit; 0xCC = -52dBm.	
Scaled RSSI	0x64	RSSI-%: 0x64 = 100%.	
Tag's EPC contents	CCDD44303132 333400000000	The EPC contents the tag backscattered. Note that if the tag has e.g. the XPC_W1 present then the first two bytes would contain that 16-bit word value in big-endian format.	
Payload CRC	0x62 0x58	Little endian; value is **0x5862.**	

5.5 Sisään ja ulos kirjautuneet

Kolmannella välilehdellä on kaksi taulukkoa, joissa näkyy kaikki sisään ja ulos kirjautuneet. Taulukoiden tiedot päivittyvät samalla tavalla kuin livenäkymä-välilehdellä sillä erolla, että tässä osassa koodia molempien taulukoiden kyselyn muodostaminen tietokantaan tehdään CASE:n eri osioissa, kun livenäkymässä molempien taulukoiden kysely valitaan samassa CASE:n osassa. Tämä tarkoittaa sitä, että koodi on pidempi. Komennon muodostaminen, rivien valmisteleminen ja datan lukeminen tehdään molemmille erikseen. Erikseen koodauksen hyvä puoli on, että pitämällä kahden eri taulukon toiminnot erillään virheellisen koodauksen todennäköisyys pienenee huomattavasti ja koodi on selkeämmin luettavissa ulkopuolisen silmin. Livenäkymä välilehdellä sama koodi on tehty noin puolta vähemmällä

rivillä koodia helppolukuisuuden kustannuksella, täten koodin kunnollinen kommentointi korostuu entisestään.

Myös sisään- ja uloskirjautuneiden taulukoissa voidaan valita useammasta eri välilehdestä, joita vaihtamalla vain kyseinen taulukko päivittyy. Operaattori voi halutessaan päivittää molemmat taulukot klikkaamalla päivityspainiketta, joka on jokaisella välilehdellä. Sisään kirjautuneiden taulukossa on erillinen lisätoiminto eli painike, jolla voidaan pakottaa käyttäjän uloskirjautuminen. Operaattorin tehdessä uloskirjaaminen HMI:n kautta, täytyy hänen valita nimi sisään kirjautuneiden listasta. Ulos kirjaaminen tapahtuu valitun nimen ID:n perusteella, ja tiedot on päivitettävä kahteen eri SQL-taulukkoon, RecentEvents, sekä Main-taulukoihin. Uloskirjaus päivittää käyttäjätiedot tietokantaan ja myös automaattisesti päivittää sekä livenäkymän, että sisään- ja uloskirjautuneiden välilehtien taulukot.

5.6 Etsi-välilehti

Viimeinen HMI:n välilehdistä on etsi-osio, jossa operaattori voi hakea kaikkia käyttäjiä nimen tai yrityksen avulla. Välilehdelle pääsee klikkaamalla HMI:n suurennuslasin kuvaa, joka päivittää taulukon automaattisesti. Etsi-taulukkoon päivitetään kaikki käyttäjät, jotka on rekisteröity Main-taulukkoon. Kaiken kaikkiaan käyttäjiä voidaan hakea 350 kappaletta jaettuna kuudelle välilehdelle. Tätä lukua voidaan muuttaa ainoastaan muuttamalla komentoa koodin Main-osiossa.

Jokaiselle käyttäjälle annetaan luomisen yhteydessä tagin voimassaoloaika, koska se tulostetaan tarraan, ei sitä voida pidentää. Mikäli käyttäjän kulkuoikeutta halutaan jatkaa tarrassa ilmoitetun päivämäärän jälkeen, on luotava uusi käyttäjä ja tulostettava uusi kulkulupatarra. Etsi-välilehti on ainoa paikka, jossa kerran luotu käyttäjä voidaan poistaa. Vanhentuneet kulkuluvat hidastavat turhaan datan lukemista etsi-välilehden taulukkoon ja vievät tuhraan muistia tietokannassa. Vaikka muistia kuluu hyvin vähän, on tietokantaa silti tarpeellista siivota, sillä myös vanhentuneet käyttäjätiedot näkyvät HMI:n talulukoissa. Vuosittaiset voimalaitoksen huolto- ja uudistushankkeet tuovat aina uudestaan suuren määrän työntekijöitä, eikä kulkulupaa voida myöntää ulkopuolisen yrityksen jäsenille määräaikaa pidemmäksi.

Käyttäjän poistamista varten muiden toimintojen lailla operaattori valitsee taulukosta poistettavan nimen, joka poistetaan käyttämällä käyttäjän ID:tä ehtona poistokomennossa. Komento muodostetaan Kuva 17 esittämällä tavalla, josta huomataan, että käyttäjä tarvitsee poistaa ainoastaan Main-taulukosta. Main-taulukko on ainoa taulukko tietokannassa, josta pääportin RFID-lukijat lukevat käyttäjätietoja. UUID-arvon puuttuessa Main-taulukosta ei enää tietokannasta poistettu käyttäjä voi kulkea portista. Poistettujen käyttäjätietojen mukana poistuu myös ID-arvo, joka vapautuu uusien käyttäjien käyttöön.

Kuva 17. Komennon muodostaminen käyttäjän poistamista varten tietokannasta

```
FORM_DELETE_USER_COMMAND: // Form a command string for deleting selected user

    JSONVariables.DeleteUser[1] := 0;
    StrToDatabase4 := '';
    strcat (ADR (StrToDatabase4), ADR ('DELETE from dbo.ESERFIDMain WHERE ID = $''));
    strcat (ADR (StrToDatabase4), ADR (JSONVariables.ID[2]));
    strcat (ADR (StrToDatabase4), ADR ('$''));
    DatabaseFBs.Excecute4.enable := TRUE;
    DatabaseFBs.Excecute4.pSqlStatement := ADR(StrToDatabase4);
    DatabaseFBs.Excecute4.connectionIdent := ConnectToDatabaseVar.ConnectionIdent4;
    DatabaseFBs.Excecute4();
    IF DatabaseFBs.Excecute4.status = 0 THEN
        SearchData := CHECK_DELETE_USER_ROWS_AFFECTED;
    ELSIF DatabaseFBs.Excecute4.status <> 65535 THEN
        ErrorConnectionIdent := ConnectToDatabaseVar.ConnectionIdent4;
        ErrorNumber := DatabaseFBs.Excecute4.status;
        GetError := 1;
    END_IF;
```

5.7 Pääportin RFID-lukijat

Pääportin RFID-lukijoiden koodi pyörii omassa ohjelmassaan, jonka toistosykli on 100 ms, eli kymmenen kertaa hitaammin kuin Main-ohjelma. Tämän nopeammin ei ohjelman tarvitse olla, sillä ei saavuteta hyötyä yhteyden ollessa asynkroninen ja matkan lukijoille ollessa verraten pitkä. RFID-ohjelman Main-osio on rakenteeltaan täysin samanlainen kuin Main-ohjelman, mutta RFID:n Main-osiossa muodostetaan vain kaksi yhteyttä tietokantaan, yksi lukijoiden tiedonsiirtoa ja yksi virheenkäsittelyä varten.

Pääportista kulkevilla jää aina aikaleima tapahtuneesta, tämä aika muodostetaan käyttämällä B&R:n kirjastoa, josta saadaan aika PLC:n sisäisestä kellosta. Päivämäärä ja aika on muotoa DATE_AND_TIME, joten jotta voidaan muodostaa aikaleima tietokantaan, täytyy

se purkaa osiin Kuva 18 osoittamalla tavalla. Tietokantaan tallennettava päivämäärä ja aika täytyy olla tietyssä muodossa, joten vuosien, kuukausien ja muiden ajan osioiden väliin on asetettava oikeanlaiset merkit, muuten SQL-palauttaa virheen. Ohjelman pyöriessä 100 millisekunnin syklillä myös ajan tarkkuus on sama, ollen kuitenkin riittävän tarkka tämän projektin käyttötarkoitukseen.

Kuva 18. DATE_AND_TIME-muuttujan purkaminen osiin ja kasaaminen tietokantaan sopivaksi.

```
// Form a date time string to be used in database information
DTGetTime_0(enable := TRUE);
gDT1 := DTGetTime_0.DT1;
DT_TO_DTStructure(gDT1, ADR (DTStruck));

DateAndTimeStruckt.YearString := UINT_TO_STRING(DTStruck.year);
DateAndTimeStruckt.MonthString := USINT_TO_STRING(DTStruck.month);
DateAndTimeStruckt.DayString := USINT_TO_STRING(DTStruck.day);
DateAndTimeStruckt.HourString := USINT_TO_STRING(DTStruck.hour);
DateAndTimeStruckt.MinuteString := USINT_TO_STRING(DTStruck.minute);
DateAndTimeStruckt.SecondString := USINT_TO_STRING(DTStruck.second);
DateTimeStringToDatabase := '';
strcat (ADR (DateTimeStringToDatabase), ADR (DateAndTimeStruckt.YearString));
strcat (ADR (DateTimeStringToDatabase), ADR ('-'));
strcat (ADR (DateTimeStringToDatabase), ADR (DateAndTimeStruckt.MonthString));
strcat (ADR (DateTimeStringToDatabase), ADR ('-'));
strcat (ADR (DateTimeStringToDatabase), ADR (DateAndTimeStruckt.DayString));
strcat (ADR (DateTimeStringToDatabase), ADR (' '));
strcat (ADR (DateTimeStringToDatabase), ADR (DateAndTimeStruckt.HourString));
strcat (ADR (DateTimeStringToDatabase), ADR (':'));
strcat (ADR (DateTimeStringToDatabase), ADR (DateAndTimeStruckt.MinuteString));
strcat (ADR (DateTimeStringToDatabase), ADR (':'));
strcat (ADR (DateTimeStringToDatabase), ADR (DateAndTimeStruckt.SecondString));

gDateTimeString := DateTimeStringToDatabase;
```

Toisin kuin RFID-lukija toimistossa, jota käytetään uuden käyttäjän UUID:n lukemiseen tietokantaan, pääportin lukijaa kutsutaan aktiivisesti, jolloin se etsii jatkuvasti RFID-tarroja. Myös pääportin lukijalle täytyy komento määrittää bitti kerrallaan, ja lukijan vastaus on myös aina samassa muodossa. Pääportin lukija ei lue vain yhtä tarraa kerrallaan, vaan sille lähetettävä komento pystyy keräämään monta eri tarraa bufferiin, jolloin useampi käyttäjä voidaan kirjata sisään tai ulos samanaikaisesti.

Kun lukija havaitsee UUID-arvon, se tallentuu bufferiin. Lukijalta saatavan datapaketin rakenteen ollessa aina sama, TCP-toimilohkon ReceiveData-muuttujan kymmenes bitti kertoo kuina monta UUID-arvoa bufferissa on. Koodi tarkastelee tätä kymmenettä bittiä, sen ollessa eriarvoinen kuin nolla, lähetetään lukijalle komento, joka hakee kaikki UUID-arvot bufferista ja tyhjentää sen. TCP-toimilohkolta saatava data on aina muuttujatyyppiä BYTE, joten se on muutettava kahteen eri muuttujaan. Tietokantaan lähetettävää tietoa varten

koko lukijalta saapuva datapaketti muutetaan STRING-muuttujaksi, ja datan käsittelyä varten koodissa muutetaan datapaketti myös INT-muuttujaksi.

Kuva 19 muodostetaan tallennetusta datapaketista löydetyt UUID-arvot yksi kerrallaan. UUID:n kokoaminen alkaa aina samasta kohtaa, koska saapuva datapaketti oli aina rakenteeltaan samanlainen (Kuva 20). RFID-tarrojen UUID on aina saman mittainen tarrasta riippumatta, tämän tiedon avulla voidaan datapaketista noutaa yksi UUID kerrallaan. Kuva 20 nähdään, että mikäli bufferissa oli enemmän kuin yksi UUID, ovat ne datapaketissa tallennettuna peräkkäin.

Kuva 19. UUID arvojen tallentaminen muuttujiin.

```
FIND_NEXT_UUID_VALUE:   // Finds next UUID value from data stream

    UUIDToDatabase152 := '';
    PayloadLen152 := STRING_TO_INT(UUIDStringArray152[1]);
    PayloadLen152 := PayloadLen152 - 6;
    FOR BufferIndex := ArrayReadingIndex TO BufferIndex + PayloadLen152 -2 DO
        IF ArrayReadingIndex = BufferIndex THEN
            AntennaID := UUIDArrayNo[ArrayReadingIndex + 1];
            FOR p := BufferIndex + 2 TO BufferIndex + 13  DO
                strcat (ADR (UUIDToDatabase152), ADR (UUIDStringArray152[p]));
            END_FOR;
            ArrayReadingIndex := p;
            GateRead := CHECK_IF_UUID_BLOCKED;
        END_IF;
        EXIT;
    END_FOR;
```

Kuva 20. Saapuvan datapaketin rakenne luettaessa RFID tunnisteita bufferista. (Kuva: NordicID, 2017)

RESPONSE PACKET CONTENTS

Byte(s)	Value	Description	
0…5	A5200000007A (6 bytes)	Header consisting of:	
		A5	
		2000 = 0x0020 (32)	Payload + CRC length
		0x0000	Response flags
		0x7A	Header check sum
6	06	Command echo.	
7	00	Status: 0 = OK.	
First tag entry			
8	0D	Bytes to follow (13); this tag entry's size – 1.	
9	00	Antenna ID (source antenna).	
10…21	30 38 E5 11 C7 35 D2 C0 D9 C8 2A 25	Tag's EPC.	
Second tag entry			
22	0D	Bytes to follow (13); this tag entry's size – 1.	
23	00	Antenna ID (source antenna).	
24…35	30 00 00 00 07 89 00 40 00 00 00 02	Tag's EPC.	
36…37	65 CD	Unsigned 16-bit, little-endian: packet CRC-16 = **0xCD65**.	

Lukijat portilla ovat lähekkäin, joten ulos kirjautuessa ei haluta vahingossa välitöntä sisään kirjausta. Kun UUID on tallennettu muuttujaan, tarkastellaan, onko kyseisellä UUID-arvolla ollut tapahtumaa viimeiseen 60 sekuntiin. Jos Viimeisimmästä tapahtumasta on kulunut vähemmän aikaa kuin 60 sekuntia, jätetään kyseinen UUID huomiotta ja palataan alkuun etsimään tunnisteita bufferiin. Lupatarkastelun päästäessä UUID:n eteenpäin haetaan kyseisellä UUID-arvolla tietokannasta sille kuuluvan käyttäjän tiedot. Kun käyttäjätiedot on onnistuneesti haettu tietokannasta, täytyy sinne tallennettua RFID-tarran voimassaoloaikaa verrata nykyhetkeen. Jos käyttäjän kulkuluvan voimassaoloaika on mennyt umpeen, keskeytetään prosessi ja palataan lukemaan uusia UUID-arvoja. Kulkuluvan voimassaolon vahvistamisen jälkeen voidaan muodostaa komento tietokantaan tapahtuman kirjausta varten.

Pääportilla on yksi lukija, johon on liitetty kaksi antennia, näiden antennien ID:tä käytetään määrittämään komennon muodostuksessa, oliko kyseessä ulos-, vai sisäänkirjautuminen. Tieto tapahtumasta päivitetään SQL:n Main-taulukkoon ja muodostetaan kokonaan uusi rivi dataa RecentEvents-taulukkoon. Tässä kohtaa koodi kulkua on tarpeen tallentaa juuri lähetetyn käyttäjän UUID-muuttujaan, jota tarkasteltiin estämään välittömät uudelleenkirjaukset. Pääportin avauskäsky annetaan myös tässä kohtaa koodin kulkua, se on sekunnin pulssi, joka lähetetään modbus-väylällä porttia hallinnoivalle Siemens 1200-logiikalle. Jos bufferissa oli enemmän kuin kaksi UUID-arvoa, siirrytään takaisin koodin alkuosaan erottelemaan UUID tallennetusta bufferista. Kun uusia UUID-arvoja ei enää ole tarkasteltavaksi, kutsutaan TCP-toimilohkoa sekunnin ajan, jotta lukijalta saapuva datapaketti nollaantuisi.

5.8 Virheenkäsittely

Virheenkäsittely on tärkeä osa koodia, hyvin toteutettuna se auttaa paikantamaan vian syytä virhetilanteessa, eikä operaattorilta vaadita suuria toimia. Tässä projektissa ainoita virheenkäsittelyä kaipaavia tapauksia ovat yhteys- tai SQL-ongelmat. Kuten muu osa koodista, myös virheenkäsittely koostuu CASE:sta.

Toimilohkoja kutsuttaessa tarkastellaan sen statusta, mikäli status numero on nolla, tarkoittaa se, että toiminto on suoritettu onnistuneesti, tai että toimilohko on neutraalissa tilassa. Normaalitilanteessa statuksen poiketessa nollasta se on yleensä ”kiireinen”, joka esimerkiksi SQL-toimilohkoilla on 65535. Tilassa kiireinen toimilohko suorittaa parhaillaan sille annettua komentoa, eli pääasiallisesti toimilohkojen statukset ovat jompikumpi näistä kahdesta. Statuksen poiketessa näistä kahdesta on se merkki virheen tapahtumisesta toimilohkoa kutsuttaessa.

Kuva 21 tarkastellessa voidaan todeta, että kun toimilohkoa (DatabaseFBs.PrepareRow2) kutsutaan, tarkastellaan heti seuraavana sen statusta. Statusta tarkastelemalla siirrytään koodissa eteenpäin, jonka johdosta IF-lausekkeen ensimmäinen ehto on, että status on nolla ja toiminto on suoritettu onnistuneesti. Virhetilanteen toteamiseksi toisena ehtona tarkastellaan, onko toimilohkon status jokin muu kuin kiireinen. Eli jos toimilohko ei ole saanut onnistuneesti suoritettua komentoa, mutta ei myöskään ole sitä suorittamassa, on tapahtunut virhe. Virhetilanteessa tallennetaan toimilohkon yhteys-ID ja statusnumero, sekä laukaistaan virheenkäsittely ActivateError-muuttujan avulla. Tietokantaan on samaan aikaan auki monta yhteyttä, mikäli virhe ei ole toimilohkon sisäinen, vaan tulee tietokannasta, tarvitaan oikea yhteys-ID SQL-virheviestin hakemiseen. Status numeron tallentamisella määritetään oliko virhe toimilohkon sisäinen, vai SQL-virhe.

Kuva 21. Rivin valmisteluun käytettävän toimilohkon kutsuminen.

```
DatabaseFBs.PrepareRow2();
IF DatabaseFBs.PrepareRow2.status = 0 THEN
    AddUser := READ_ID_VALUE;
ELSIF DatabaseFBs.PrepareRow2.status <> 65535 THEN
    ErrorConnectionIdent := ConnectToDatabaseVar.ConnectionIdent2;
    ErrorNumber := DatabaseFBs.PrepareRow2.status;
    ActivateError := 1;
END_IF;
```

Virheen sattuessa ei joka tilanteessa haluta laukaista virheenkäsittelyä, tällainen tilanne syntyy aina, kun luetaan dataa tietokannasta useamman rivin ja sarakkeen verran. Tällaisessa tilanteessa on Kuva 22 mukaisesti kolme IF-ehtoa. Kuva 22 esitetty koodi on uuden käyttäjän lisäämisestä, jossa valmistellaan uusi rivi datan lukemista varten. Rivin valmistelun jälkeen määritettiin luettava sarake, ja sarakkeiden loputtua siirryttiin valmistelemaan uutta riviä. Kun luettavia rivejä ei enää ole saatavilla, antaa tietokanta siitä virheen. Tämän virheen tunnus numero on aina sama (100), joten sitä käytetään tunnistamaan, että kaikki saatavilla oleva data on luettu. Samalla menetelmällä luetaan myös sarakkeet, jos sarakkeita ei ole saatavilla, saadaan tietokannasta puolestaan virhekoodi -1.

Kuva 22, Rivin valmisteleva toi ilohko uuden käyttäjän lisäämisessä.

```
DatabaseFBs.PrepareRow2();
IF DatabaseFBs.PrepareRow2.status = 0 AND DatabaseFBs.PrepareRow2.dbError <> 100 THEN
    AddUser := DETERMINE_COLUMN_INDEX;
    GetDataVar.ColumnIndex2 := 1;
ELSIF DatabaseFBs.PrepareRow2.status <> 65535 THEN
    IF DatabaseFBs.PrepareRow2.dbError = 100 THEN           // If all rows and columns
        AddUser := WAIT_FOR_INTERACT;
        JSONVariables.LoadingPleaseWait[1] := 0;
    END_IF;
ELSIF DatabaseFBs.PrepareRow2.status <> 65535 THEN
    ErrorConnectionIdent := ConnectToDatabaseVar.ConnectionIdent2;
    ErrorNumber := DatabaseFBs.PrepareRow2.status;
    ActivateError := 1;
END_IF;
```

Kun virheenkäsittely laukeaa, halutaan ensimmäisenä tietää virheen syy, tätä varten - tallennettiin toimilohkon yhteys-ID. Virheviesti täytyy aina hakea erikseen tätä varten tehdyllä toimilohkolla, jolle tarvitsee määrittää parametreja. Toimilohkolle täytyy antaa lupa toimia, määrittää yhteys-ID, muuttuja johon virheviesti tallennetaan ja sen koko. Kun viesti on luettu onnistuneesti, tarkastellaan mikä yhteys virheen aiheutti. Kuva 23 mukaisesti tarkastellaan, mikä yhteys aiheutti virheen ja tallennetaan se lähetettäväksi tietokantaan. HMI on jaettu välilehtien mukaan osiin, joille jokaiselle on muodostettu oma yhteys tietokantaan. Välilehti, jolla virhe sattui, nollataan koodissa ja estetään sen toiminta 30 sekunniksi. 30 sekunnin jälkeen yritetään muodostaa yhteys uudelleen, ja operaattorin täytyy suorittaa haluamansa toiminto uudelleen. Tämä 30 sekuntia antaa aikaa mahdolliselle yhteyshäiriölle korjaantua itsestään, mutta myöskin estää virhetiedon lähettämisen tietokantaan turhan usein, sillä jokainen virhe tallennetaan tietokantaan.

Kuva 23. Virheenkäsittely, jossa tarkastellaan mikä yhteys-ID virheen aiheutti.

```
DatabaseFBs.ErrorMsg();
IF DatabaseFBs.ErrorMsg.status = 0 THEN
    IF ErrorConnectionIdent = ConnectToDatabaseVar.ConnectionIdent1 THEN
        BockTimer1 := 1;
        Actions1 := CALL_CONNECT_FB_INIT;
        GetEvents := WAIT_FOR_EVENT_REFRESH;
        JSONVariables.RefreshEventTabs := 0;
        JSONVariables.RefreshTableValues[0] := 0;
        UpdateEventsTab := 0;
        JSONVariables.RefreshLoggedInTabsMain := 0;
        JSONVariables.LoadingPleaseWait[0] := 0;
        ErrorTab := 'Events';
    ELSIF ErrorConnectionIdent = ConnectToDatabaseVar.ConnectionIdent2 THEN
        BockTimer2 := 1;
        Actions2 := CALL_CONNECT_FB_INIT;
        AddUser := WAIT_FOR_INTERACT;
        JSONVariables.InteractButtons[0] := 0;
        JSONVariables.LoadingPleaseWait[1] := 0;
        JSONVariables.ModifyUser := 0;
        JSONVariables.DeleteUser[0] := 0;
        AddUUIDToId := WAIT_FOR_READ_TAG_COMMAND;
        JSONVariables.InteractButtons[2] := 0;
        ErrorTab := 'Add new user';
```

Kuva 24 määritetään virhetyyppi, joka voi olla joko TCP-toimilohkon, tai SQL-toimilohkon aiheuttama. Jos virhe tapahtuu TCP-toimilohkolla, tallennetaan jo virheenlaukaisun yhteydessä, tuliko virhe pääportin vai toimiston lukijasta. SQL-toimilohkon virhe voi puolestaan syntyä itse toimilohkossa, tai tulla tietokannasta. Tämän määrittämistä varten tallennettiin virheenkäsittelyn laukaisemisen yhteydessä virhe numero. Kun virhetyyppi ja -viesti on tallennettu, täytyy varmistaa, että lähetettävä virheviesti on tietokantaan kelpaavassa muodossa. Kun tietokanta antaa virheen, sen lähettämässä viestissä saattaa olla heittomerkkejä. Koska koodissa STRING-muuttuja määritetään aina kahden heittomerkin sisään, täytyy sen sisällä olevat heittomerkit määrittää eri tavalla, eli lisäämällä $-merkki sen eteen ($’).

Kuva 24. Virhetyypin määrittäminen

```
2:  // Check if error was because of SQL function block error, TCP error or SQL database error

    IF ErrorType = 'TCP printer' OR ErrorType = 'TCP tag reader' THEN
        ErrorMsgToDatabase := ErrorMsgToDatabaseTCP;
    ELSE
        IF ErrorNumber = 65534 THEN
            ErrorMsgToDatabase := DatabaseFBErrorMsg[0];
            ErrorType := 'Function block';
        ELSIF ErrorNumber = 65530 THEN
            ErrorMsgToDatabase := DatabaseFBErrorMsg[1];
            ErrorType := 'Function block';
        ELSIF ErrorNumber = 34932 THEN
            ErrorMsgToDatabase := DatabaseFBErrorMsg[2];
            ErrorType := 'Function block';
        ELSIF ErrorNumber = 34940 THEN
            ErrorMsgToDatabase := DatabaseFBErrorMsg[3];
            ErrorType := 'Function block';
        ELSIF ErrorNumber = 34931 THEN
            ErrorMsgToDatabase := dbGetErrorMsgVar.pErrorMessage;
            ErrorType := 'SQL server';
        END_IF;
    END_IF;
    GetError := 3;

3:  // Delete all $' marks from a string that is to be sent to database, it causes SQL error

    ErrInt := FIND(ErrorMsgToDatabase,'$'');
    IF ErrInt = 0 THEN
        GetError := 4;
    ELSE
        ErrorMsgToDatabase := DELETE(ErrorMsgToDatabase,2,ErrInt);
    END_IF;
```

Jos SQL-virheviesti, joka sisältää $’-merkkejä yritetään tallentaa tietokantaan sellaisenaan, aiheuttaa se uuden virheen, koska tekstimuuttujan sisältämä teksti loppuu ensimmäiseen tällaiseen merkkiin. Vaikka se on oikea tapa määrittää heittomerkit tekstimuuttujan sisään, lukee tietokanta sen eri tavalla, kuin että se olisi koodissa kirjoitettu. Kuva 24 CASE:n kohta kolme etsii tietokannasta saadusta virheviestistä kaikki $’-merkit ja poistaa ne, ennen kuin siirrytään kasaamaan lähetettävää viestiä. Kun virheen tiedot on aikaleiman kanssa lähetetty tietokantaan, täytyy yhteys, jolla virhe sattui katkaista tietokantaan Disconnect-toimilohkolla. Tämä tarvitsee tehdä vain, jos vian syy oli SQL-toimilohkossa, TCP-toimilohko käsittelee virheen itse, ja muodostaa yhteyden automaattisesti uudelleen.

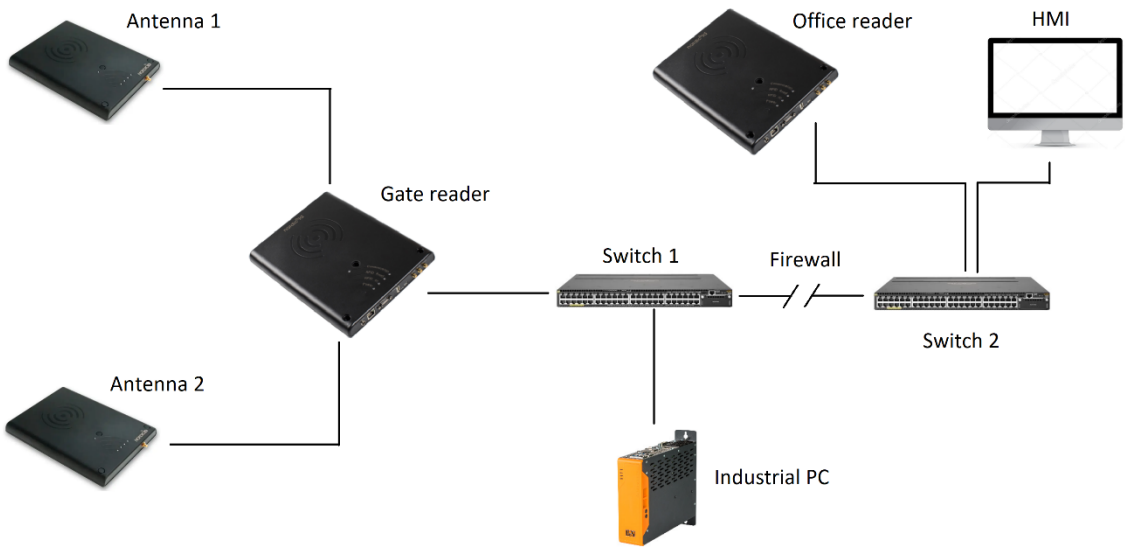
6 Järjestelmän asentaminen

Järjestelmän asennus aloitettiin teollisuus PC:n, lukijoiden ja tulostimen toiminnallisuuden testaamisella kytkimen kautta, kytkimen portit oli konfiguroitu etukäteen järjestelmää varten. Tässä vaiheessa testaamisen ohessa koodiin tehtiin vielä pieniä muutoksia ennen

laitteiden asentamista oikeille paikoille. Teollisuus PC asennettiin laitekaappiin DIN-kiskoon Siemens portinohjauslogiikan viereen, näin vältyttiin ylimääräiseltä kaapeloinnilta jännitteensyötön ja potentiaalintasauksen ollessa samassa tilassa. Kaapelointia pääportin lukijaa varten ei erikseen tarvittu, vaan pystyttiin hyödyntämään olemassa olevia yhteyksiä.

Kuva 25 on esitetty RFID-lukijoiden asettelu. Yhteydet teollisuus PC:ltä kytkimelle ja kytkimeltä pääportin lukijalle on muodostettu CAT 6 U/UTP-liitosjohdoilla ja olemassa olevalla kaapeloinnilla. PC toimii 24 VDC jännitteellä, ja sille asennettiin oma muuntaja samaan laitekaappiin. RFID-lukijoiden mukana toimitettiin niille omat muuntajat, kuitenkin kaapeloinnin ja työmäärän vähentämiseksi käytettiin POE (power over ethernet) -syöttöä laitteiden sitä tukiessa. Pääportin lukija asennettiin portin ohjauskeskukseen sen vieressä, antennien asennuspaikka sijoittui portin rakenteisiin niin, etteivät ne haitanneet portin liikettä. Antennin kytkettiin lukijan AUX1- ja AUX 2-liittimiin pakkauksessa mukana tulleilla SMA-liitosjohdoilla.

Kuva 25. Järjestelmän layout. (BR-automation, n.d.; NordicID, 2020; NordicID, 2021; Depositphotos, 2015; ITPlanet, n.d.)



Koodia tehdessä pyritään etukäteen ennakoimaan kaikki mahdolliset vikatilanteet ja tapahtumat, mitä järjestelmän käytössä saattaa ilmetä. Asennusvaiheen jälkeen aloitettiin laajempi testaaminen, minkä aikana pyrittiin tuottamaan kaikki mahdolliset tilanteet mitä järjestelmän käytössä saattaa ilmetä, jotta voitiin testata koodin toiminnallisuus. Tällaisia tilanteita olivat muun muassa useamman kuin yhden tarran lukeminen kerrallaan ja RFID-

tarran pitäminen antennin lukuetäisyydellä pidempiä aikoja. Koodin virheenkäsittely ei vielä tässä vaiheessa ollut valmis, joten sitä viimeisteltiin testauksen yhteydessä.

7 Yhteenveto

Kokonaisuudessaan projekti oli haastava ja erittäin kehittävä, syvällisempi perehtyminen kommunikointiin auttoi ymmärtämään tiedon käsittelyä bittitasolla. Itse koodin kirjoittaminen alkoi hitaasti, sillä ensin täytyi ymmärtää laitteiden välistä kommunikointia. Tämän opetteluun ja testaukseen käytettiin runsaasti aikaa, mutta kuitenkin ajan käyttö tähän oli erittäin kannattavaa. Ymmärryksen lisääminen kommunikoinnissa mahdollisti myöhemmässä vaiheessa muutosten tekemisen koodiin erittäin sujuvasti ja nopeasti, ongelman tai muutoksen tarpeen ilmetessä oli heti selkeää, miten asiaa lähdettiin ratkaisemaan.

Ohjelmointi pyrittiin tekemään mahdollisimman valmiiksi ennen järjestelmän asentamista asiakkaalle, kuitenkin tiukan aikataulun vuoksi esimerkiksi koodin virheenkäsittelyä jouduttiin jatkamaan asennusvaiheen jälkeen. Tästä ei kuitenkaan aiheutunut projektin kannalta haittaa, sillä testivaiheessa voimalaitoksella pystyttiin samanaikaisesti tätä toteuttamaan. Itse laitteiden asennus sujui ripeästi lähes kaikkien tarvittavien tarvikkeiden ollessa välittömästi saatavilla. Suurimmat haasteet ilmenivät testausvaiheessa, kun päästiin käsittelemään pääportin lukijalta saatua dataa. Toimistoon sijoitettu lukija, jolla rekisteröidään uudet käyttäjät sisään järjestelmään, käytti yksinkertaisempaa komentoa RFID tarran lukuun, joten sen kanssa ei ongelmia ollut.

Järjestelmää on aktiivisesti paranneltu asiakkaalle luovutuksen jälkeen, sillä yleensä mahdolliset virheet tai puutteet koodissa ilmenevät laajemman käytön yhteydessä. Etäyhteys PLC:lle ja tietokantaan mahdollistivat tapahtumien seuraamisen ja koodin parantelemisen reaaliaikaisesti välittömästi tarpeen tullen.

Järjestelmän valmistuttua ja sitä muokatessa havaittiin asioita, joita tulevaisuudessa voitaisiin parantaa. Tästä esimerkkinä taulukoiden päivitys HMI:lle ja yhteyksien luominen tietokantaan. Aktiivisessa järjestelmässä HMI:n käyttö tietyllä välilehdellä estetään, jos jonkun taulukon arvot muuttuvat. Tällaisessa tilanteessa taulukko alustetaan ja tiedot

haetaan tietokannasta uudelleen, näin käyttäjä näkee reaaliaikaisesti taulukon tyhjenevän ja uusien arvojen päivittyvän taulukkoon. Vaihtoehtoinen tapa tähän olisi suorittaa datan kerääminen taustalla, ja päivittää HMI:llä näytettävä tieto yhden ohjelmakierron aikana useamman sijasta.

Yhteyksiä SQL-tietokantaan luodaan aktiivisessa järjestelmässä useita eri HMI:n välilehdille ja lukijoita varten. Eli tietokantaan on muodostettu yhteys, vaikka sillä ei välttämättä tehdä mitään, tämä tekee kyseisetä yhteydestä turhan, vaikka se välttämättä järjestelmän toimintaan ei vaikuttaisikaan. Vaihtoehtoinen ratkaisu olisi muodostaa yhteys aina tapahtuma ilmetessä, eli kun esimerkiksi lisätään uusi henkilö tietokantaan, muodostetaan yhteys sillä hetkellä, kun operaattori painaa lähetä-painiketta. Pienillä muutoksilla voitaisiin myös tehdä bufferi datan lähettämiseen, jolloin HMI:n painikkeita ei tapahtuman käsittelyn ajaksi tarvitsisi poistaa käytöstä.

Lähteet

Brady. (2020). *TSPL and TSPL2 Programming Manual for the BBP12*. https://support.bradyid.com/s/article/TSPL-and-TSPL2-Programming-Manual-for-the-BBP12

BR-automation. (n.d.). *Industrial PCs*. https://www.br-automation.com/en/products/industrial-pcs/automation-pc-3100/system-units/5apc3100kbu1-000/

Canon. (2013) *Introduction to Page Description Languages*. https://web.archive.org/web/20131228134307/http://www.cmrepro.com/documents/CanonUnderstandingPDLs.pdf

Depositphotos. (2015.). *Vector illustration computer transparent screen*. https://depositphotos.com/68218641/stock-illustration-vector-illustration-computer-transparent-screen.html

GS1 Finland. (n.d.) *GS1 RFID/EPC -tunnistetyypit*. https://gs1.fi/fi/ohjeet/yritystunniste/rfid

ITPlanet. (n.d.). *Networking*. https://it-planet.com/en/p/hp-jl074a-229066.html?number=4252665000.1&gclid=EAIaIQobChMIgvK5nNCI8wIVEpOyCh1TigMAEAQYBSABEgJri_D_BwE

ISO. (1990). *ISO 7185:1990*. https://www.iso.org/standard/13802.html

ISO. (2015). ISO/IEC 18000-63:2015. https://www.iso.org/standard/63675.html

ISONAS, (n.d.). *The Evolution of Access Control*.https://www.isonas.com/news-education/the-evolution-of-access-control/

ITM Components. (2018). *Radio Frequency Explained*. https://itm-components.co.uk/blogs/news/radio-frequency-explained

Laine,H. (2002). *History of SQL*.
https://www.cs.helsinki.fi/u/laine/tuelip/sql_material/sql_history.html

National Radio Astronomy Observatory. (2014). *Types of Radio Waves*.
https://web.archive.org/web/20140423162103/https://public.nrao.edu/types-of-radio-waves

NordicID. (2017). *Example scan single response*.
https://github.com/NordicID/nur_sdk/blob/master/embedded/NUR_AN003_scan_single.pdf

NordicID. (2017). *Simple buffer response example.*
https://github.com/NordicID/nur_sdk/blob/master/embedded/NUR_AN004_inventory_and_tag_buffer.pdf

NordicID. (2020.). *NORDIC ID SAMPO S0*. https://www.nordicid.com/device/nordic-id-sampo-s0/

NordicID. (2021.). *NORDIC ID SAMPO S2*.
https://www.nordicid.com/nordic_id_sampo_s2_one_series_datasheet_v14/

PLCopen. (n.d.) *Logic*. https://plcopen.org/technical-activities/logic

RFIDLab Finland ry. (2021). *Mitä on RFID?.* https://www.rfidlab.fi/rfid-teknologia/

RFID4u. (2021). *Dig Deep – Construction of RFID Tags*. https://rfid4u.com/dig-deep-construction-of-rfid-tags/

Shutterstock. (n.d.). *Electromagnetic spectrum*. https://www.shutterstock.com/fi/image-vector/electromagnetic-spectrum-different-types-radiation-by-469352255

University of Colorado. (2007). *An Introduction to*

RFID Technology.
https://home.cs.colorado.edu/~rhan/CSCI_7143_001_Fall_2002/Papers/rfid_intro_01593568.pdf