VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Toni Kivipuro

# AUTOMAATIO PAKKAUSSOLUN DATAN KERÄÄMINEN OPC UA -PROTOKOLLAA KÄYTTÄEN

Tekniikka
2024

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

## TIIVISTELMÄ

| | |
|---|---|
| Tekijä | Toni Kivipuro |
| Opinnäytetyön nimi | Automaatio pakkaussolun datan kerääminen OPC UA -protokollaa käyttäen |
| Vuosi | 2024 |
| Kieli | suomi |
| Sivumäärä | 45 |
| Ohjaaja | Jukka Matila (VAMK), Toni Laiho (Danfoss) |

---

Tässä työssä on tarkoituksena tehdä sovellus, jonka avulla saadaan kerättyä dataa automaattisen pakkaussolun toiminnasta OPC UA -protokollan avulla. Tämän avulla saadaan tärkeää tietoa pakkaussolun toiminnasta ja toimivuudesta.

Tavoitteena on luoda ohjelma C# -ohjelmointikielellä, jonka avulla saadaan Siemens S7-1500 -logiikalta tarvittavaa tietoa pakkaussolun toimivuudesta, OPC UA -standardia käyttäen.

Kun käytetään OPC UA -standardia teollisuuden automaatiossa, saadaan hyödyllistä dataa kerättyä talteen. Tarvittaessa saadaan tietoa myös mahdollisista ongelmista, joita voidaan analysoida datan perusteella.

---

Avainsanat OPC UA, automaatio, IoT, ohjelmointi, pakkaus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

## ABSTRACT

| | |
|---|---|
| Author | Toni Kivipuro |
| Title | Automated Packing Cell Data Management Using OPC UA Protocol |
| Year | 2024 |
| Language | Finnish |
| Pages | 45 |
| Name of Supervisor | Jukka Matila (VAMK), Toni Laiho (Danfoss) |

The purpose of this thesis was to develop software that can collect data from the automated packing cell using the OPC UA -protocol. This provides important information about how this automated packing cell works.

The goal was to create a program with the C# -programming language using OPC UA standard, which makes it possible to receive data from Siemens S7-1500 logic.

Using the OPC UA machine-to-machine communication protocol makes it possible to collect and store data from industrial automated systems. Then it also gives the possibility to analyze error data from automated systems or robots.

Keywords OPC UA, automation, IoT, programming, packing

## SISÄLLYS

**KUVA- JA TAULUKKOLUETTELO**

**KÄSITE -JA LYHENNELUETTELO**

| | |
|---|---|
| C# | Ohjelmointikieli |
| COM | Component Object Model |
| DCOM | Distributed Component Object Model |
| HMI | Human Machine Interface |
| IEC | International Electrotechnical Comission |
| Lähetin | Elektroninen laite, joka lähettää dataa |
| MariaDB | Relaatiotietokanta |
| OLE | Object Linking and Embedding |
| OPC UA | OPC Unified Architecture. |
| OPC | Open Platform Communications |
| PLC | Programmable Logic Controller |
| PRI | Uniikki avain tietokannassa |
| S7-1500 | Siemens simatic logiikkasarja |
| SAP ME | SAP Manufacturing Excecution |
| Vastaanotin | Elektroninen laite, joka vastaanottaa dataa |
| WPF | Windows Presentation Foundation |
| XAML | Extensible Application Markup Language |

# 1 JOHDANTO

Tässä työssä on tarkoituksena luoda ohjelma C# -ohjelmointikielellä. Manuaalisesti täytettävän Excelin sijaan ohjelmalla saadaan automaattisen pakkasolun dataa sen toimivuudesta automaattisesti tietokantaan. Standardi, jota hyödynnetään, on OPC UA. Pakkaussolun datan kerääminen talteen on todella tärkeää, koska sen avulla pystytään näkemään sen käytettävyys ja toimivuus, sekä ennakoimaan ongelmia. Lisäksi sen avulla voidaan tehdä tarvittavia toimenpiteitä. Teollisuudessa automaation yleistyessä monien eri asioiden seuraaminen datan perusteella on mahdollista ja dataa pystytään hyödyntämään ennakointiin. Lisäksi suuri osa manuaalisia työvaiheita saadaan automatisoitua.

Pakkaussolu toimii kahdessa eri vuorossa ja se pakkaa noin 300 laitetta per vuoro, jolloin sen tarpeellisuus on todella suuri. Se helpottaa tuotantoa keskittymään muihin työtehtäviin, mikä oli yksi suurimmista syistä, miksi nähtiin tarpeelliseksi saada ohjelma automaattiselle datan keräämiselle.

OPC UA tarjoaa mahdollisuudet automaation kehittämiseen, jolloin saadaan kaikki mahdollisuudet irti automaatiosta. Kyseisellä ratkaisulla löytyy myös valmiita ohjelmistoja, joilla voidaan seurata tai kerätä dataa, mutta siitä haluttiin tehdä räätälöity versio Danfossille. Kyseinen sovellus tullaan testaamaan vain testiympäristössä, koska täydet valmiudet testaamiseen ei ole vielä pakkaussolun ympäristössä valmiina. Tarkoituksena on saada kokemusta enemmän OPC UA -standardista ja sen hyödyntämisestä automaattisissa ratkaisuissa.

# 2 DANFOSS

Danfoss on yritys, joka tuottaa teollisuuteen ja kuluttajalle erilaisia ratkaisuja (taajuusmuuttajia, termostaatteja, venttiilejä ja korkeapainepumppuja). Vaasaan syntyi yhtiö Danfoss Drives vuonna 2014, kun tanskalainen yritys Danfoss osti koko Vaconin osakekannan. Danfoss Drives keskittyy täysin taajuusmuuttajien tuottamiseen ja siihen liittyviin ratkaisuihin. Danfoss Drivesin Suomen pääkonttori sijaitsee Vaasassa. Kyseisessä toimipisteessä on tuotekehitys -ja tuotanto eritehoisille taajuusmuuttajille (kuva 1).


**Kuva 1.** Vaconin taajuusmuuttajien tehdas Vaasassa.

## 2.1 Vaconin historia

Vaasassa syntyi uusi yritys vuonna 1993, kun 13 insinööriä aloitti työskentelyn asiakkaan tarpeisiin vastaavien taajuusmuuttajien parissa. Osa heistä oli työskennellyt taajuusmuuttajien parissa eri osa-alueilla. Yrityksen nimeksi syntyi Vaasa Control, joka lanseerasi ensimmäisen tuotesukupolven kaksi vuotta myöhemmin. Tähän samaan aikaan Suomessa elettiin vaikeaa lama-aikaa, jonka aikana suunniteltiin erittäin laadukas ja toimiva tuote.

Yhtiö listautui Helsingin pörssiin vuonna 2000 jolloin kyseisestä yrityksestä muodostui tunnettu Vacon Oy.

Vaconista tuli globaali yritys, jossa työskenteli 1600 henkilöä. Parhaimmillaan liikevaihto oli 409 miljoonaa euroa. Taajuusmuuttajia myytiin ympäri maailmaa ja toimintaa pystyttiin laajentamaan. Vacon perustui omistautumiseen ja suurella intohimolla työskentelemiseen.

## 2.2 Danfoss

Danfoss on tanskalaisen perheen perustama yritys, joka toimii teollisuuden eri osa-alueilla. Tunnetuimpia tuotteita ovat Danfoss-termostaatit, joita on monissa kodeissa ympäri maailman. Toiminta koostuu taajuusmuuttajista, lämmitys/jäähdytys -ratkaisuista ja voimansiirrosta. Yrityksellä on 97 tehdasta yli 20 eri maassa ja se työllistää noin 42 000 henkilöä. Liikevaihto vuonna 2022 oli noin 10,3 miljardia euroa. (Danfoss, 2024)

## 2.3 Taajuusmuuttaja

Taajuusmuuttajia käytetään sähkömoottorien nopeuden säätämisessä. Niiden avulla saadaan tarvittava nopeus moottorille eri käyttökohteissa. Taajuusmuuttajan käytössä on monia erilaisia hyötyjä. Ne pidentävät sähkömoottorien/sovellusten ikää mekaanisesti, parantavat niiden ohjattavuutta ja vähentävät energiankulutusta. Taajuusmuuttajat ovat suuri osa ihmisten arkea, mutta ne eivät ole aina näkyvillä. Danfoss Drives tarjoaa monipuolisia ratkaisuja eri asiakkaille (kuva 2).

**Kuva 2.** VACON®- ja VLT®-tuotteet (Danfoss Drives, 2024).

# 3 AUTOMAATTINEN PAKKAUSSOLU

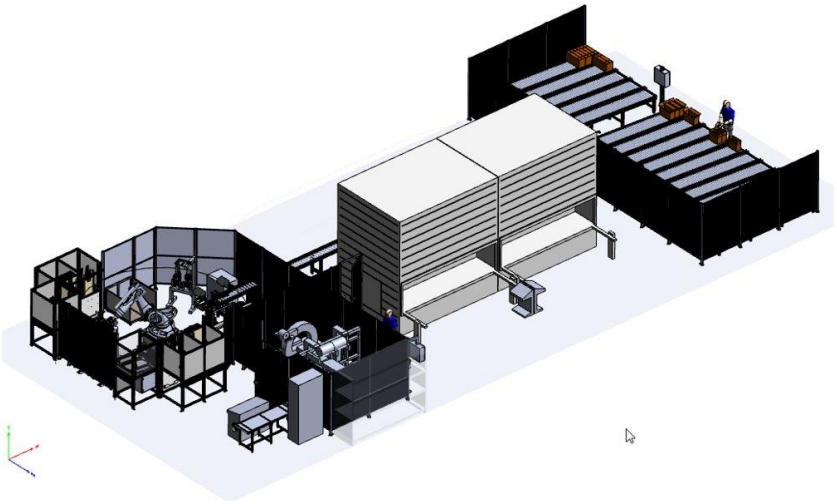
Tässä luvussa käydään läpi automaattista pakkaussolua, jolle tämä sovellus on tarkoitus tehdä. Kyseinen täysin automaattinen pakkaussolu pakkaa parhaimmillaan 600 laitetta päivässä, ja yhden laitteen läpimenoaika on 2 minuuttia.

## 3.1 Pakkaussolu

Automaation yleistyessä on huomattu teollisuudessa, että useita yksinkertaisia työvaiheita pystytään helposti automatisoimaan. Tämän ansiosta pystytään saamaan kustannustehokkaita ratkaisuja tuotantoon, mikä auttaa myös työntekijöitä keskittymään työhön, jota ei saada helposti automatisoitua. Tämä on johtanut siihen, että Danfossin Vaasan tehtaalle on rakennettu automaattinen pakkaussolu, joka pakkaa tehtaan pienitehoisia taajuusmuuttajia jopa 600 kpl päivässä. Tähän kokonaisuuteen kuuluu:

- mobiilirobotti
- KUKA KR90
- KUKA KR8.

Mobiilirobotti kuljettaa laitteen tuotantolinjalta pakkaussoluun, minkä jälkeen solussa oleva lukija lukee tarvittavat tiedot laitteesta, sen jälkeen SAP ME kertoo robotille, minkälainen laite on kyseessä ja robotti pakkaa kyseisen laitteen (kuva 3).



**Kuva 3.** 3D-malli pakkaussolusta.

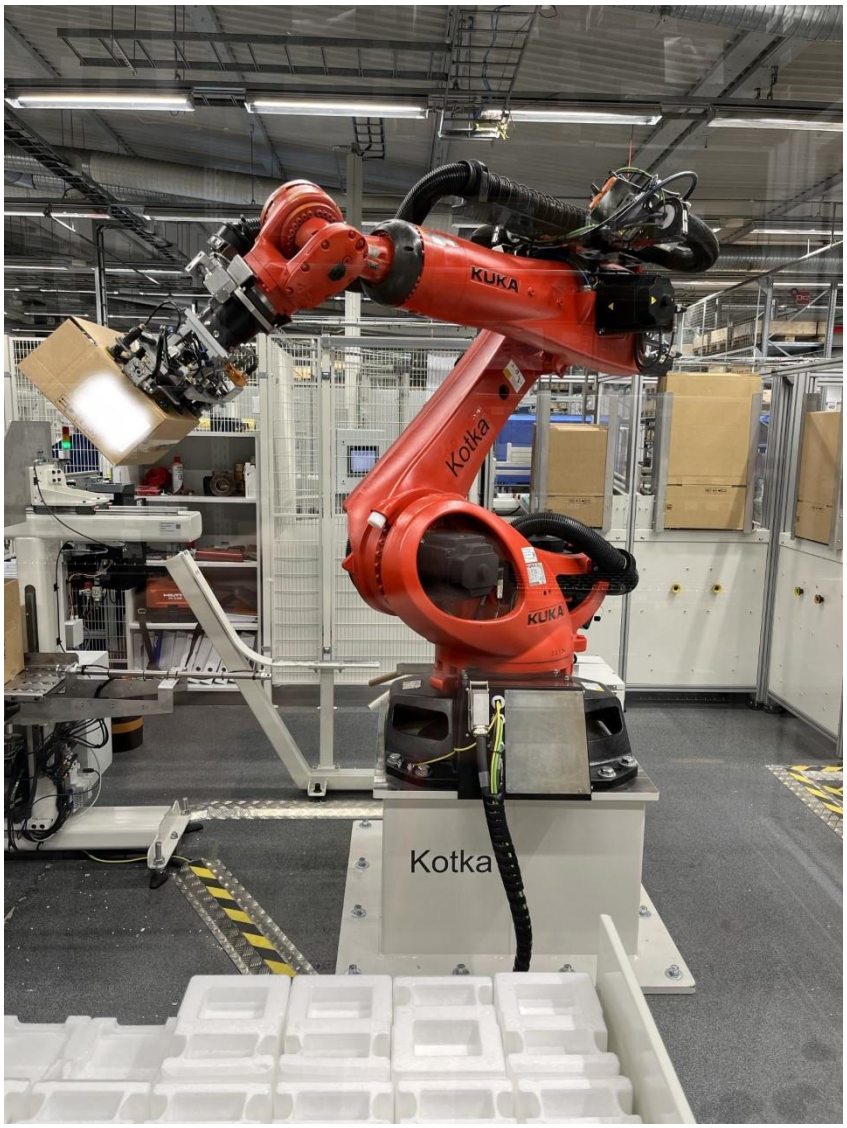
### 3.1.1 Mobiilirobotti

Laitteiden kuljetus tuotantolinjalta tapahtuu mobiilirobotin (kuva 4) avulla mobiilirobotti noutaa syöttöpisteeltä laitteita ja tuo ne pakkaussolun syöttöpisteelle. Kokonaisuudessaan mobiilirobotti kulkee noin 300 metrin matkan. Tähän kokonaisuuteen on lisätty syöttöpisteelle ja mobiilirobotille läpäisevän säteen optinen anturi, jonka lähetin ja vastaanotin kommunikoivat keskenään. Tämän ansiosta kommunikaatio toimii molempiin suuntiin. Läpäisevä optinen anturi on laite, joka käyttää valoa esineen läsnäolon tai puuttumisen havaitsemiseen. (Stykemain, 2023.)

**Kuva 4.** Mobiilirobotti.

### 3.1.2 KUKA KR90

Paketin kokoaminen ja laitteen paketoiminen tapahtuu KUKA KR90 -robotin avulla (kuva 5). Kyseinen robotti on saanut tarvittavan tiedon laitteesta SAP ME:stä, jolloin robotti osaa tehdä juuri kyseiselle laitteelle oikeanlaisen pakkauksen. Pakkauksen tekemisen jälkeen robotti asettaa laitteen pakettiin ja tämän jälkeen paketti laitetaan eteenpäin tarvittavien dokumenttien ja osien lisäämistä varten.




**Kuva 5.** KUKA KR90 -robotti

### 3.1.3 KUKA KR8

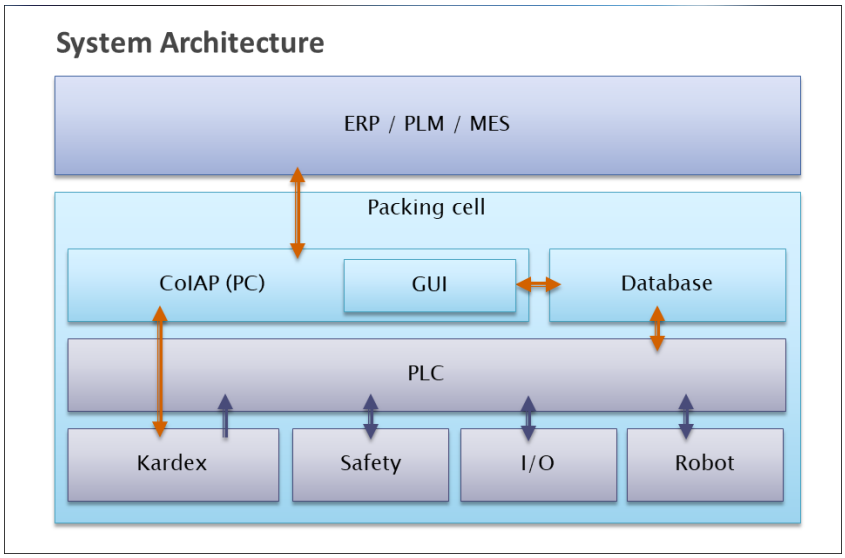
Nyt laite tulee paketoinnin viimeistelyyn, eli KUKA KR8 -robotille (kuva 6). Tämä kyseinen robotti lisää laitteen mukaan tarvittavat dokumentit ja mahdolliset muut lisäosat, jos laite sellaisia tarvitsee.


**Kuva 6.** KUKA KR8 -robotti.

## 3.2 Pakkaussolun arkkitehtuuri

Tällä hetkellä pakkaussolun data jää talteen ainoastaan tietokanta-PC:n muuttujiin, jotka on määritelty tallettamaan dataa 30 päivää (kuva 7). Tämä tarkoittaa sitä, että jos halutaan saada talteen dataa, se täytyy hoitaa manuaalisesti. Tähän asti pakkaussolun dataa on virheiden osalta lisätty Excel tiedostoon, jonka avulla voidaan tarkastella virheiden toistuvuutta ja mahdollisesti tehdä ennakoituja toimenpiteitä virheiden vähentämiseksi. Kyseinen ratkaisu ei ole paras mahdollinen, koska se pystytään hoitamaan automaattisesti ja siten saadaan karsittua manuaalista työtä. Vaihtoehdoksi valittiin ohjelma (OPC UA Client), jonka avulla saadaan

yhteys PLC-dataan (OPC UA Server). Asiakkaan (Client) avulla voidaan ottaa yhteys palvelimeen (Server) ja sen avulla saadaan haettua muuttujille dataa palvelimelta.




**Kuva 7.** Pakkassolun arkkitehtuuri.

### 3.3 Pakkaussolun data

Yksi tärkeimmistä asioista automaatiossa on kyseisen kokonaisuuden seuranta, jonka avulla pystytään tarkkailemaan prosessin toimivuutta. Tähän tarkoitukseen on yleensä luotu mahdollisuus saada tarvittavaa dataa ulos laitteistosta, jolloin pystytään analysoimaan mahdollisia virheitä tai mikä on sen kokonaiskäyttöaika. Taulukossa 1 on mainittu tämän pakkaussolun keräämä data, joka tällä hetkellä on näkyvillä vain pakkaussolussa olevissa Siemens SIMATIC HMI -näytöillä (kuva 8).

**Taulukko 1.** PLC-datan kuvaus ja datatyyppi.

| **Kuvaus** | **Datatyyppi** |
|---|---|
| Aloitetut tuotteet | INT |
| Tuotteet yhteensä | INT |
| Hyvät tuotteet | INT |
| Vialliset tuotteet | INT |
| Viimeisin tahtiaika | float |
| Vuoron tahtiaika | float |
| Käynnissäoloaika | INT |
| Käynnissäoloaika % | float |
| Käytettävyysaika | INT |
| OEE | float |
| OE | float |
| YIELD | float |
| Suorituskyky | float |
| Automaattinen käytettävyys | float |
| Käytettävyysaika | float |


**Kuva 8.** Pakkaussolun data HMI näytöllä.

# 4 OPC-STANDARDI

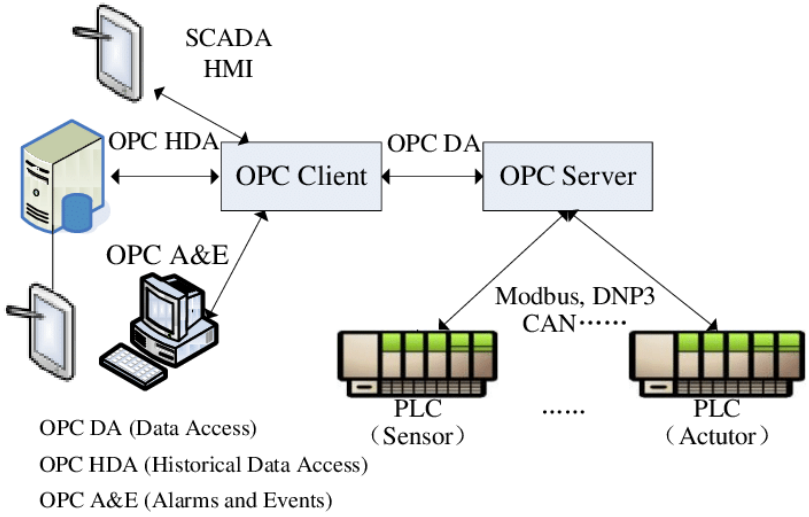
Tässä luvussa käydään läpi lyhyesti hieman historiaa, minkälaisia ominaisuuksia OPC Classic ja OPC UA sisältävät.

## 4.1 OPC Classic

OPC-standardin tarina alkaa 1990-luvulta, jolloin PC:t ja Microsoft Windows olivat kovassa nosteessa. Sen juuret perustuvat Microsoftin teknologioihin, jotka ovat OLE, COM ja DCOM. Näiden avulla eri ohjelmat pystyvät vaihtamaan tietoja keskenään. OPC:n tarkoituksena on tarjota standardirajapinta, jonka avulla erilaiset sovellukset pystyvät tarjoamaan tarvittavaa mittaus- ja ohjaustietoa eri sovelluksille, riippumatta valmistajista tai erilaisista teollisuusprotokollista. Aikoinaan, kun OPC oli käytössä, sen määrittelyt olivat seuraavanlaiset:

- DA
- A&E
- HDA.

Näillä määrittelyillä pystyttiin tuomaan tarvittavaa tietoa teollisuuden järjestelmästä. Tosiaikaisen datan siirtoon käytettiin DA:ta, hälytys- ja tapahtumatietojen välittämiseen A&E:tä ja historiatietojen siirtoon HDA:ta. Nykypäivänä perinteinen OPC kulkee nimellä OPC Classic (kuva 9). (Suomen automaatioseura ry, 2023.)



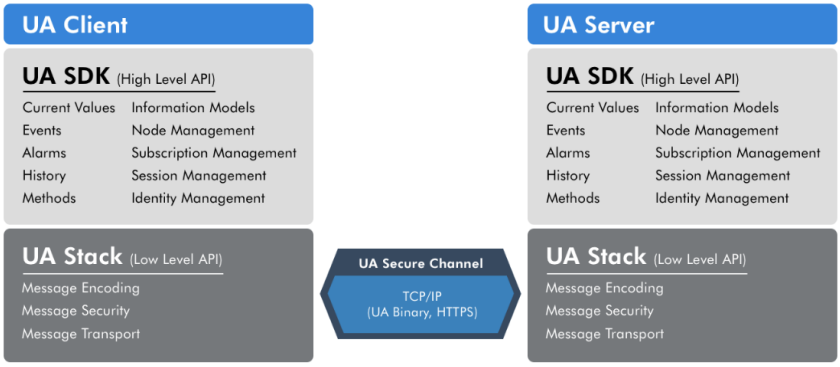

**Kuva 9.** OPC Classic-arkkitehtuuri (ResearchGate, 2014).

### 4.2 OPC UA

OPC UA mahdollistaa laitteistojen välisen kommunikoinnin teollisuuden automaatiossa alustasta riippumatta. Se mahdollistaa esimerkiksi datan siirron logiikalta tietokantaan ja siitä mahdollisesti eteenpäin. OPC UA–standardi julkaistiin vuonna 2008. Kyseinen kokonaisuus on alustariippumaton arkkitehtuuri, joka sisältää kaikki yksittäiset määrittelyt, jotka ovat käytössä OPC Classic -versiossa (kuva 10). OPC UA perustuu IEC-standardiin 62541. OPC UA -standardin edut ovat seuraavanlaisia (OPC Foundation, 2023):

- Toiminallinen vastaavuus
    - OPC UA sisältää COM-määrittelyt, jotka ovat OPC Classic-versiossa.
- Alustariippumaton
    - Mahdollistaa toiminnan kaikissa käyttöjärjestelmissä.
- Tietoturva
    - Yhteys on salattu, vaatii todennuksen ja turvatarkastuksen järjestelmien välillä.

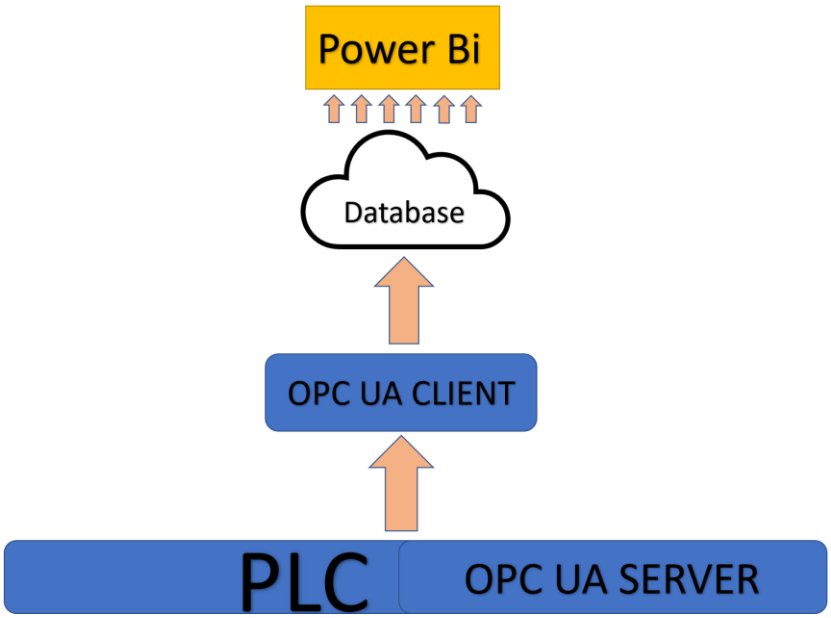
- Laajennettavuus
    - Mahdollistaa uusien ominaisuuksien lisäämisen, vaikuttamatta olemassa oleviin sovelluksiin.
- Kattava tiedon mallinnus
    - Pystytään määrittelemään monimutkaista tietoa.




**Kuva 10.** OPC UA -arkkitehtuuri (TheOPCFoundation, 2020).

# 5 X4MANCLIENT-SOVELLUS

Tässä luvussa käydään läpi X4MANClient-sovellus ja sen tarpeelliset toiminnot. Kyseisen sovelluksen tarkoituksena on hakea tarvittavaa dataa OPC UA -palvelimelta ja siirtää se datatietokantaan, jolloin tarvittaessa voidaan analysoida pakkaussolun käytettävyyttä (kuva 11).



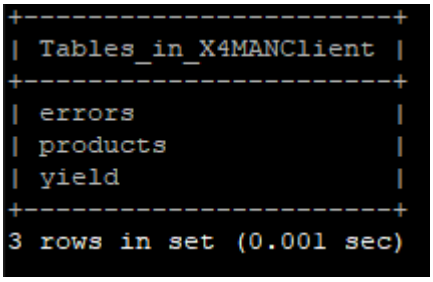

**Kuva 11.** Pakkaussolun tavoitekokonaisuus.

## 5.1 Tietokanta

Tietokanta tätä sovellusta varten toteutettiin omalle Rasberry Pi 3 model b -minitietokoneelle, joka sopii täydellisesti pienimuotoiseen palvelintoteutukseen. Käyttöjärjestelmänä toimii Raspberry Pi OS (Raspberry Pi, 2024.).

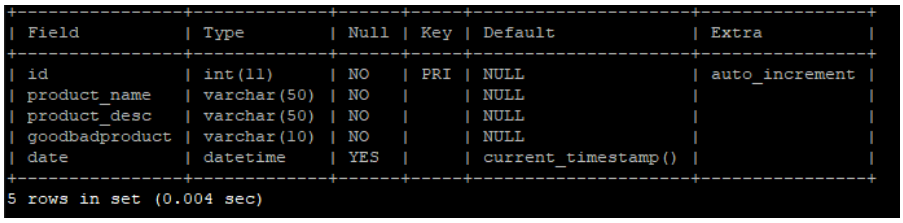
Tietokantana käytetään MariaDB relaatiotietokantajärjestelmää, joka on vapaan lähdekoodin järjestelmä. Se on yksi maailman suosituimmista tietokannoista. (MariaDB, 2024.)

Tietokannan nimeksi tuli X4MANClient ja sille tuli kolme erilaista taulua, products, errors ja yield (kuva 12).




**Kuva 12.** X4MANClient-tietokannan taulut.

Taulussa products ovat seuraavat sarakkeet Id, product_name, product_desc, goodbadproduct ja date. Id-sarake on pääavain (PRI), joka on yksilöllinen tunniste kullekin riville. Se kasvaa automaattisesti aina, kun uusi rivi lisätään tietokantaan. Id-sarake on määritelty kokonaislukuna ja sen maksimipituus on 11. Product_name ja product_desc -sarakkeet on simuloinnissa satunnaisesti arvottu, ne kertovat mikä tuote ja minkä niminen on kyseessä. Molemmat sarakkeet on määritelty merkkijonotyyppinä ja niiden pituus on 50 merkkiä. Goodbadproduct -sarake on myös simuloinnissa satunnaisesti määritelty arvo tuotteelle, joka kuvitteellisesti kertoo, onko tuote mennyt pakkaussolusta läpi hyvänä vai huonona -tuotteena. Se on myös määritelty merkkijonona, jonka maksimipituus on 10 merkkiä. Viimeisenä sarakkeena kyseisessä taulussa on date, johon määritellään sen hetken päivämäärä ja kellonaika, milloin rivi on lisätty tauluun (kuva 13).

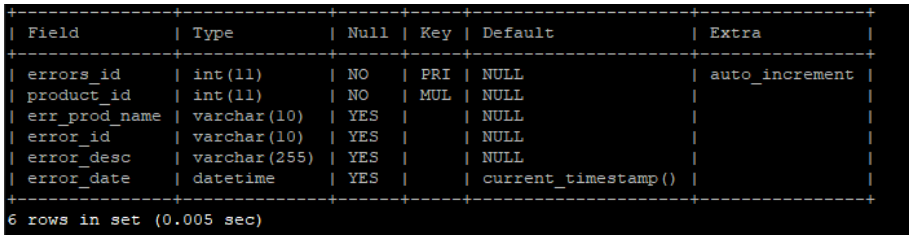


| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | int(11) | NO | PRI | NULL | auto_increment |
| product_name | varchar(50) | NO | | NULL | |
| product_desc | varchar(50) | NO | | NULL | |
| goodbadproduct | varchar(10) | NO | | NULL | |
| date | datetime | YES | | current_timestamp() | |

5 rows in set (0.004 sec)

**Kuva 13.** Products-taulu.

Errors-taulussa on sarakkeet errors_id, product_id, err_prod_name, error_id, error_desc ja error_date. Tässä taulussa errors_id -sarake on pääavain (PRI). Errors_id-sarake on määritelty kokonaislukuna ja sen maksimipituus on 11. Seuraava sarake product_id on viite products taulun id -sarakkeeseen ja tällä voidaan liittää kyseinen virhe oikeaan tuotteeseen. Tällä sarakkeella on samat määrittelyt kuin errors_id:llä. Sarake err_prod_name on sama kuin product_name products taulussa. Error_id on tunniste virheelle ja siihen liittyy suoraan sarake error_desc. Error_id arvotaan satunnaisesti, jos tuote menee virheellisesti läpi. Error_id on merkkijono ja sen pituus on 10 merkkiä, kun taas error_desc merkkijono on pituudeltaan 255 merkkiä. Myös tässä taulussa on käytössä date -sarake ja se on samanlainen kuin products taulussa (kuva 14).



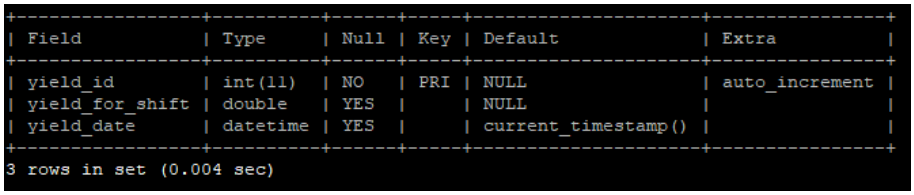
| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| errors_id | int(11) | NO | PRI | NULL | auto_increment |
| product_id | int(11) | NO | MUL | NULL | |
| err_prod_name | varchar(10) | YES | | NULL | |
| error_id | varchar(10) | YES | | NULL | |
| error_desc | varchar(255) | YES | | NULL | |
| error_date | datetime | YES | | current_timestamp() | |

6 rows in set (0.005 sec)

**Kuva 14.** Errors-taulu.

Viimeisenä tauluna on yield. Tähän tauluun on määritelty sarakkeet yield_id, yield_for_shift ja yield_date. Sarake yield_id on vastaava yksilöllinen tunniste kuin id ja errors_id. Yield_for_shift-sarakkeeseen tuodaan arvo prosentteina, joka tallennetaan tauluun vuoron lopussa. Tämä arvo kertoo, kuinka monta prosenttia

tuotteista on mennyt ”hyvänä” läpi. Kyseinen sarake on määritelty desimaalilukuna tallennettavaksi. Viimeisenä sarakkeena on yield_date, johon määritellään päivämäärä ja aika, jolloin kyseinen yield_for_shift -rivi on tallennettu tietokantaan (kuva 15).



```
+-----------------+----------+------+-----+---------------------+----------------+
| Field           | Type     | Null | Key | Default             | Extra          |
+-----------------+----------+------+-----+---------------------+----------------+
| yield_id        | int(11)  | NO   | PRI | NULL                | auto_increment |
| yield_for_shift | double   | YES  |     | NULL                |                |
| yield_date      | datetime | YES  |     | current_timestamp() |                |
+-----------------+----------+------+-----+---------------------+----------------+
3 rows in set (0.004 sec)
```

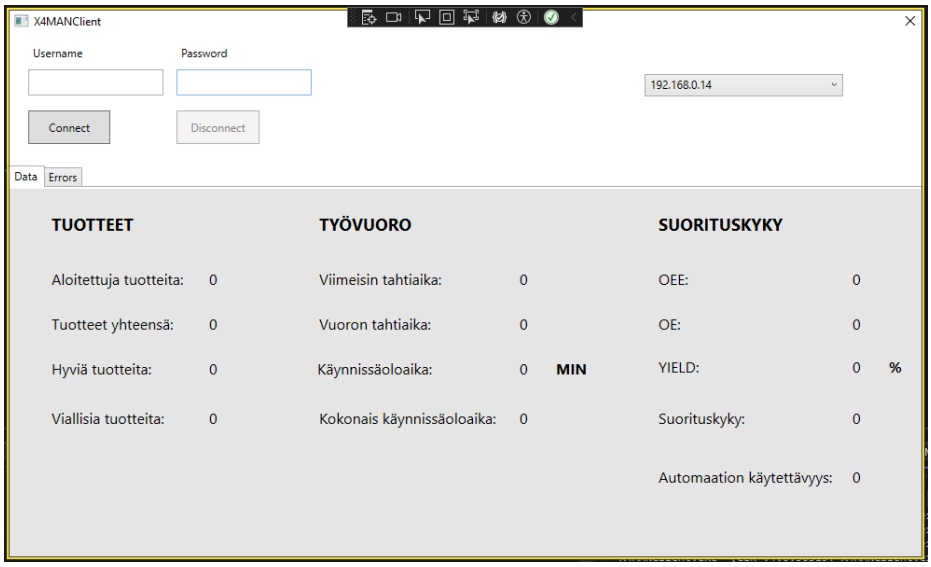
**Kuva 15.** Yield-taulu.

Tietokanta on valmiina simulointia varten ja sinne voidaan nyt tallettaa tarvittavia tietoja.

### 5.2 Käyttöliittymän toteutus

Työkaluna ohjelmoinnissa on käytetty Visual Studio 2022:aa ja ohjelmointikielenä tässä työssä käytetään C# -kieltä. Kyseinen ohjelmointikieli on oliopohjainen, mikä mahdollistaa myös visuaalisten ohjelmien luomisen. Tarkoituksena on saada luotua ohjelma, joka sisältää tarvittavan datan hakemisen OPC UA -asiakkaalla OPC UA -palvelimelta ja viedä data tietokantaan talteen.

Ohjelmoinnissa käytetään OPC UA -liitännäisiä kirjastoja, jotka mahdollistavat tarvittavien ominaisuuksien luomisen ohjelmalle. Käyttöliittymä on toteutettu WPF-käyttöliittymäkehyksellä, joka on mahdollistanut visuaalisesti näyttävän ja monipuolisen käyttöliittymän, se määritellään yleensä XAML-kielellä (kuva 16).




**Kuva 16.** X4MANClient-käyttöliittymä.

## 5.3 X4MANClient ohjelmakoodi

Sovelluksen lähdekoodi on jaettu useampaan luokkaan ja XML-tiedostoon. Luokat ja tiedostot sisältävät sovellukselle tarvittavat ominaisuudet myös demoympäristön simuloinnissa käytettäville arvojen määrittelyille. Niihin sisältyy myös OPC UA -liitännäiset metodit, jotka määrittelevät esiasetukset tarvittaville ominaisuuksille, jolloin voidaan luoda yhteys OPC UA -palvelimelle. Tässä sovelluksessa on myös lisäkirjastot, jotka ovat seuraavat:

• Opc.Ua.Client

• Opc.Ua.Core

Kirjastojen avulla saadaan tarvittavat metodit, joilla voidaan määritellä OPC UA -asetuksia.

### 5.3.1 Sovelluksen käynnistys

Sovelluksen käynnistyessä ensimmäisenä ladataan OPC UA -asetuksia, jotka on määritelty AppConfigurations.cs -luokassa:

```
/// <summary>
/// Constructor initializes configuration and resets values to default.
/// </summary>
public AppConfigurations()
{
// Sets all configuration fields to default values
   ResetValues();
// Initializes application configuration
   x_appConfiguration = CreateApplicationConfiguration();
}
```

Ensimmäinen metodi ResetValues() nollaa kyseisessä kaikki asetukset luokassa, jotka liittyvät OPC UA -asiakkaan yhteyden luomiseksi OPC UA -palvelimelle. Seuraavaksi metodilla CreateApplicationConfiguration() ladataan OPC UA sovellusta varten applikaatioasetukset, jotka on määritelty erillisessä XML-tiedostossa nimeltä X4MANClient.Config.xml:

```
// Loads the OPC UA application configuration from a file
ApplicationConfiguration x_appConfiguration = ApplicationConfigura-
tion.Load("X4MAN", ApplicationType.Client);
```

Tämän jälkeen käydään katsomassa, onko sovelluksella sertifikaattia varastossa ja jos ei, niin luodaan uusi sertifikaatti sovellukselle X4MANClient:

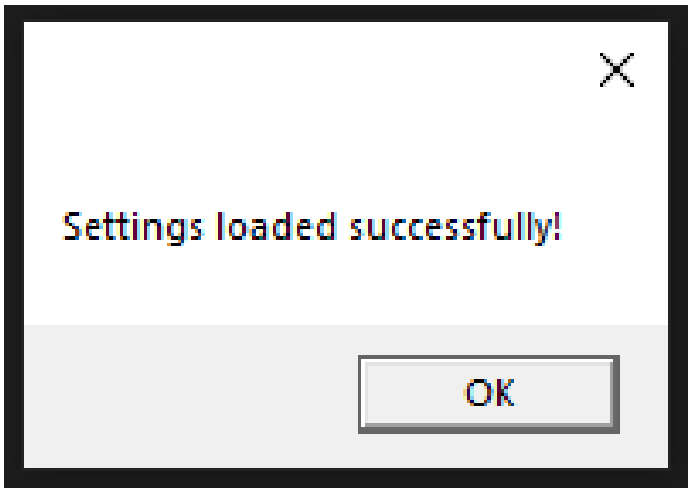
```
// Finds an existing client certificate, or generates one if not found
 X509Certificate2 clientCertificate = x_appConfiguration.SecurityConfigura-
tion.ApplicationCertificate.Find(true);

 if (clientCertificate == null)
 {
    List<string> localIp = GetLocalIp();
          clientCertificate = CertificateFactory.CreateCertificate(
```

```
            x_appConfiguration.SecurityConfiguration.ApplicationCertificate.Store-
            Type,
            x_appConfiguration.SecurityConfiguration.ApplicationCertificate.Store-
            Path,
        x_appConfiguration.ApplicationUri,
        x_appConfiguration.ApplicationName,
        null,
        localIp,
        2048, // Key size for the certification
        600, // Certificate validity period in months
        1); // Issuer type
    }
```

Seuraavaksi ladataan ennalta määritellyt NodeID -arvot ja niiden nimet XML-tiedostosta. Sille on luotuna oma luokka NodeIDsFromxml.cs, jonka sisällä on metodi LoadSettings(). Tämän kyseisen metodin sisälle on määritelty omat objektilistat, joihin ne ladataan muistiin:

```
// Initialize lists to hold node IDs and display names
nodeIDList = new List<string>();
nodeDispNameList = new List<string>();
```

Tämän jälkeen, jos lataus on tehty onnistuneesti, avautuu ikkuna, jossa lukee: *Settings loaded successfully!* (kuva 17).




**Kuva 17.** NodeID-määrittelyt ladattu onnistuneesti.

Kun edellisestä vaiheesta painetaan OK, ladataan simulointia varten tarvittavat tuotteet ja virheet omiin sanakirjoihin, joissa tuotteiden ja virheiden avaimet ovat merkkijonoina. Myös arvot ovat merkkijonoina luokassa Products.cs:

```
/// <summary>
/// Populates the product dictionary with predefined product names and descrip-
tions.
/// </summary>
private void ProductsToDictionary()
{
    productList = new Dictionary<string, string>
    {
        {"MRX1", "Low voltage drive x" },
        {"MRX2", "Low voltage drive y" },
        {"MRX3", "Medium voltage drive " },
        {"MRX4", "High voltage drive x" },
        {"MRX5", "High voltage drive y" }
    };
}
```

```
/// <summary>
/// Populates the error dictionary with predefined error codes and descriptions.
/// </summary>
private void ErrorsToDictionary()
{
    errorList = new Dictionary<string, string>
    {
        {"E01", "Cannot get or pick cardboard from stock" },
        {"E02", "Cannot get or pick styrofoam from stock" },
        {"E03", "Manual stock is empty" },
        {"E04", "Tape station is empty" },
        {"E05", "Robot cannot pick packing materials properly" }
    };
}
```

Lopuksi sovellus suorittaa seuraavat metodit ennen käynnistymistä:

- InitializeTimer()
- IpsToCombobox()
- ErrorTable()

Metodi InitializeTimer() alustaa ja määrittää uuden ajastimen, jota käytetään tässä työssä simulointiin.

Metodi IpsToCombobox() lisää kaikki paikalliset IP-osoitteet, jotka löytyvät ja lisää ne pudotusvalikkoon (Combobox). Valikosta valitaan tarvittava IP-osoite yhdistämiseen OPC UA -palvelimelle.

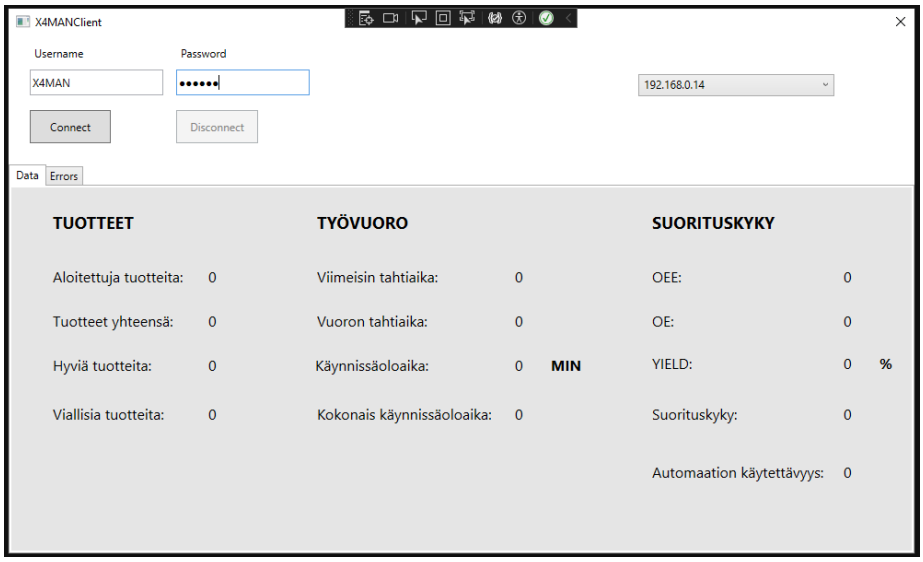
Metodi ErrorTable() alustaa ja määritelee käyttöliittymän virhetaulukon, johon lisätään simuloinnissa tulevia virheitä. Tämän jälkeen sovellus on käynnistynyt onnistuneesti.

### 5.3.2 Yhdistäminen OPC UA -palvelimelle

OPC UA -palvelimena tässä työssä käytettiin Prosys OPC UA Simulation serversovellusta (kuva 18). Kyseinen sovellus on suomalaisen Prosys OPC-yrityksen kehittämä. Yritys tarjoaa erilaisia palveluja ja auttaa tehtaita digitalisoitumaan OPC UA -standardin avulla (Prosys OPC, 2024).

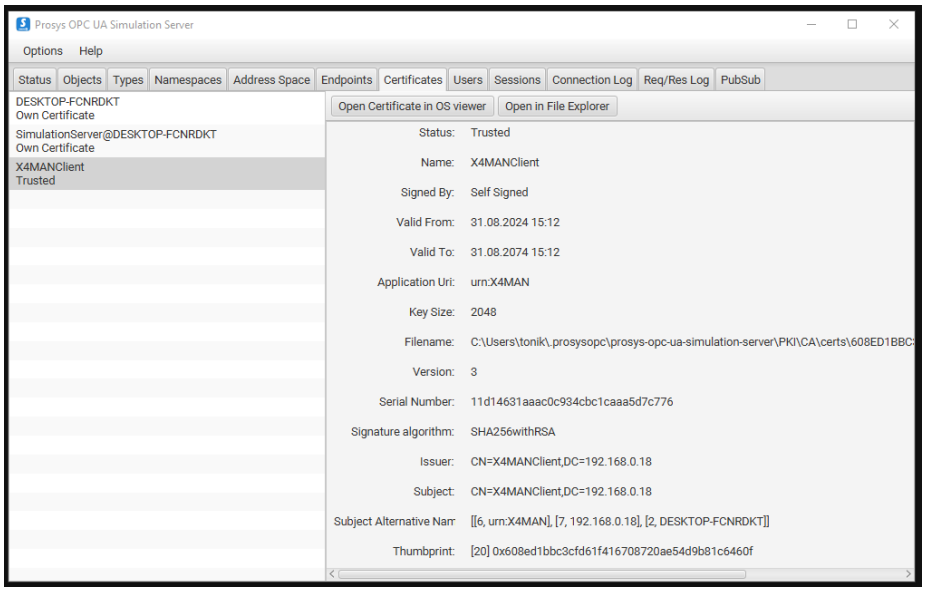

**Kuva 18.** Prosys OPC UA simulation server.

Kun palvelin sovellus on käynnistetty ja statuksena on Running, voidaan luoda yhteys X4MANClient-sovelluksella (kuva 19).




**Kuva 19.** X4MANClient yhdistäminen OPC UA -palvelimelle.

Simulation server -sovelluksella luotiin uusi käyttäjänimi X4MAN ja sille määriteltiin salasana. Sovelluksessa myös luotettiin X4MANClient-sovelluksen luomaan sertifikaattiin suojattua yhteyttä varten (kuva 20). Seuraavaksi valitaan IP-osoite valikosta, joka tässä tapauksessa on 192.168.0.14. Tämän jälkeen painetaan Connect ja sovellus lähtee suorittamaan tarvittavia toimintoja yhteyden luomiseksi.




**Kuva 20.** X4MANClient-sertifikaatti.

Ensimmäiseksi valitaan IP-osoite palvelimelle, joka tulee pudotusvalikosta, joka on sillä hetkellä valittuna ja se tallennetaan merkkijonomuuttujaan localIp:

*localIp = IpCombo.SelectedItem.ToString();*

Tämän jälkeen suoritetaan rivi, joka määritellään antamaan ilmoituksia yhteydestä palvelimeen:

*x_appConfigurations.x_keepAliveEventHandler += new KeepAliveEventHandler(KeepAliveNotification);*

Tämä keepAliveEventHandler -tapahtuma kutsuu käynnistyessään aina metodia KeepAliveNotification(), johon ei ole määritelty tarkemmin mitään sisältöä, mutta metodin mukana menee sinne Session session ja KeepAliveEventArgs e.

Seuraavaksi suoritetaan metodi Connect(), joka on AppConfigurations.cs-luokassa. Tämän metodin mukana menevät määritelyt localIp, xUsername.Text, xPassword.Password, jotka sisältävät IP-osoitteen, käyttäjänimen ja salasanan OPC UA -palvelimelle yhteyden muodostamista varten. Connect-metodin sisällä haetaan

AplicationConfiguration -objekti, jossa on määritelty palvelinyhteyteen määritellyt asetukset:

```
ApplicationConfiguration x_config = x_appConfiguration;
```

Tämän jälkeen luodaan uusi käyttäjätunnusobjekti tietojen perusteella. Tiedot tulevat aikaisemmista määrittelyistä:

```
UserIdentity uid;
uid = User(x_user, x_password);
```

Nyt luodaan sertifikaatin validointitapahtuma CertificateValidationNotification -taphtuman metodiin, jolla käsitellään mahdolliset sertifikaattiongelmat:

```
// Sets up certificate validation event
x_config.CertificateValidator.CertificateValidation+= CertificateValidationNotification;
```

Tämän jälkeen määritellään päätepisteen kuvaus x_endpointDescription, jonka mukana menee IP-osoite. Lisäksi määritellään päätepisteen konfiguraatio x_endpointConfiguration, jonka avulla tekniset asetukset hoidetaan, esimerkiksi tietoturva ja aikakatkaisut. Nyt luodaan vielä konfiguroitu päätepiste x_configuredEndpoint, johon asetetaan luotu päätepiste ja konfiguraatio:

```
// Initializes endpoint, endpoint configuration, and configured endpoint
x_endpointDescription = CreateEndpointDescription(ipAddressFromBox);
x_endpointConfiguration = CreateEndpointConfiguration(x_config);
x_configuredEndpoint = new ConfiguredEndpoint(null, x_endpointDescription, x_endpointConfiguration);
```

Lopuksi päivitetään sertifikaattikomponentti CertificateValidator ja luodaan istunto x_session yhteyttä varten. Istunnon luomista varten tarvitaan applikaa-

tiokonfiguraatio x_config, konfiguroitu päätepiste x_configuredEndpoint, istunnon nimi x_sessionName ja käyttäjätunnus uid. Samassa määritellään istunnolle KeepAlive -tapahtuma, jolloin yhteys pysyy aktiivisena:

```
x_config.CertificateValidator.Update(x_config);
// Creates a session with keep-alive support
x_session = Session.Create(x_config, x_configuredEndpoint, true, x_sessionName,
(uint)x_sessionTimeout, uid, null);
x_session.KeepAlive += new KeepAliveEventHandler(KeepAliveNotification);
```

Nyt meillä on tarvittavat määrittelyt tehtynä ja silloin metodi Connect palauttaa arvon TRUE serverConnect -muuttujalle ja yksi vaihe OPC UA -palvelinyhteyteen on valmis:

```
serverConnect = x_appConfigurations.Connect(localIp, xUsername.Text, xPass-
word.Password);
```

Yhteyden luominen vaatii NodeID-arvot ja niiden nimet sekä tilauksen (Subscription) luomisen palvelimelle. NodeID-arvot ja nimet tuodaan omiin listoihin node ja dispNames NodeIDsFromxml.cs -luokasta:

```
// Retrieve node IDs and display names for monitored items
 List<string> node = x_nodeIDsFromxml.Nodes;
 List<string> dispNames = x_nodeIDsFromxml.NodeDispName;
```

Seuraavaksi luodaan uusi tilaus palvelimelle:

```
x_appConfigurations.xSub = x_appConfigurations.subCreate;
```

Tässä kutsutaan metodia Sub() luokasta AppConfigurations.cs. Metodi katsoo ensin, onko luotu x_session arvolla NULL tai jos on olemassa oleva tilaus x_sub, niin se palauttaa metodiarvon NULL:

```
if (x_session == null || x_sub != null)
```

```
{
   return null;
}
```

Muuten luodaan uusi tilaus palvelinyhteyttä varten:

```
x_sub = new Subscription(x_session.DefaultSubscription);
x_sub.PublishingEnabled = true;
x_sub.PublishingInterval = 1; // Sets the publishing interval to 1 ms
x_session.AddSubscription(x_sub);
x_sub.Create(); // Creates the subscription on the server
```

Kun määrittelyt tilausta varten on tehty, lisätään tilaukselle NodeID-arvot ja niiden nimet:

```
// Subscribe to each monitored item by node ID and display name
for (int i = 0; i < node.Count; i++)
{
   dispName = dispNames[i];
   nodeID = node[i];
   x_appConfigurations.monitoredItems(x_appConfigurations.xSub,  nodeID,
dispName);
}
```

Nyt lisätään palvelimelta saatujen arvojen mahdollisia muutoksia varten MonitoredItemNotificationEventHandler(ClientValue), jonka avulla ClientValue()-metodi ilmoittaa mahdollisesta arvon muuttumisesta palvelimella:

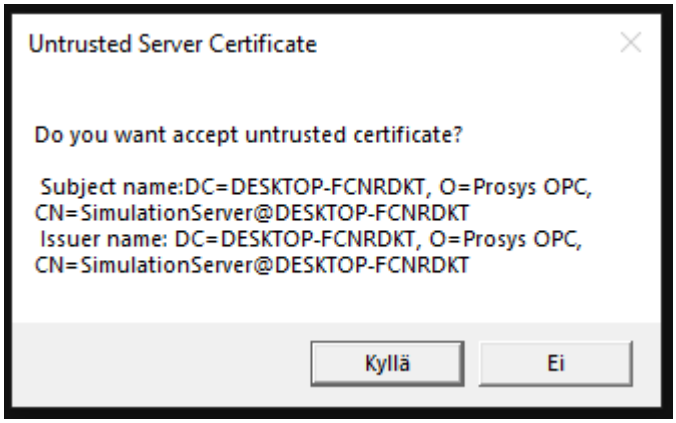
```
// Set up notification handler for changes in monitored items
x_appConfigurations.x_ItemsChangeNotification  +=  new  MonitoredItemNotifi-
cationEventHandler(ClientValue);
```

Nyt vielä asetetaan objektille x_sub arvo NULL, joka nollaa kyseisen tilauksen:

*x_appConfigurations.xSub = null;*

Tämän jälkeen suoritetaan metodi StartTimer(), joka käynnistää ajastimen palvelimelta saatujen arvojen simulointia varten.

Lopuksi sovellus kysyy, haluaako käyttäjä luottaa ei-luotettaviin palvelimen sertifikaatteihin (kuva 21). Jos painaa Kyllä, on sen jälkeen onnistunut yhteys OPC UA -palvelimelle.




**Kuva 21.** Ei-luotettavat palvelimen sertifikaatit.

### 5.3.3 Yhteys MariaDB tietokantaan

Tietokantayhteys MariaDB -relaatiotietokantaan on helposti toteutettavissa. Itse tietokantaan täytyy määritellä käyttäjätunnus ja salasana, sekä IP-osoitteet / portit, jotka mahdollistavat yhteyden tietokantaan.

Ohjelman puolella lisättiin kirjasto MySql.Data.MySqlClient;, jonka avulla voidaan käyttää metodeita yhteyden luomiseen. Tämän lisäksi määrittelin x_sqlServerUrl merkkijonon, jota tarvitaan yhteyden muodostamiseen:

*x_sqlServerUrl = @"Server=192.168.0.19;Port=3306;Database=X4MANClient;User ID=X4MAN;Password=696969;";*

Metodi tietokantayhteyden luomiselle on ConnectToSql(), joka on luokassa AppConfigurations.cs. Se palauttaa muuttujalle sqlConnect arvon TRUE tai FALSE:

```
sqlConnect = x_appConfigurations.ConnectToSql();
```

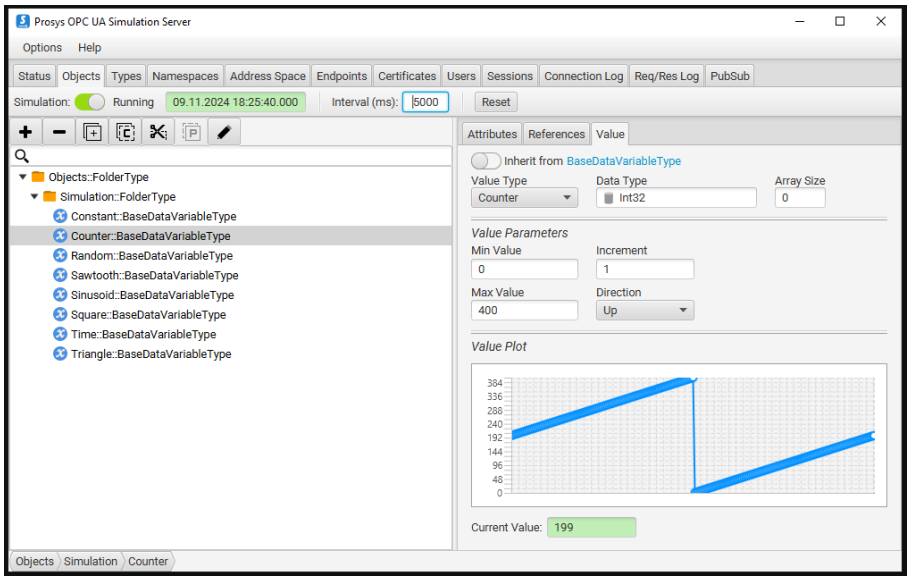
Yhteys luodaan ja avataan seuraavilla komentoriveillä:

```
x_connection = new MySqlConnection(x_sqlServerUrl); x_connection.Open();
```

Jos yhteyden muodostaminen palautetaan arvo TRUE. Sen jälkeen on yhteys tietokantaan ja sovellus pystyy tallentamaan dataa eri tauluihin tietokannassa.

### 5.3.4 Simulointi

Nyt kun tarvittavat toimenpiteet on tehty yhteyksien saavuttamiseksi, voidaan käydä simulointiin tarvittavat metodit ja toiminnallisuudet. Simuloinnissa OPC UA -palvelimelta haetaan yhtä arvoa ja sen tyyppi on Counter. Tämä arvo on määritelty päivittymään viiden sekunnin välein palvelimella, jolloin lisätään edelliseen arvoon numero 1. Simulointi alkaa arvosta 0 ja sen maksimiarvo on 400 (kuva 22). Loput tuotteiden simuloinnista on toteutettu sovelluksessa.




**Kuva 22.** Simulointi palvelimella.

Sovelluksen puolella kutsutaan metodia ClientValue(), joka sisältää objektit MonitoredItem ja MonitoredItemNotificationEventArgs. Seuraavaksi määritellään objekti MonitoredItemNotification, joka luodaan e.NotificationValue -tapahtumailmoituksen perusteella:

```
MonitoredItemNotification monitoredItemNotification = e.NotificationValue as MonitoredItemNotification;
```

Tämän jälkeen katsotaan, onko monitoredItemNotification eri suuri kuin NULL ja jos on, määritellään NodeID-nimi merkkijonomuuttujaan monItemDispName ja arvo muuttujaan val:

```
monItemDispName = monitoredItem.DisplayName.ToString();
val = Utils.Format("{0}", monitoredItemNotification.Value.WrappedValue);
```

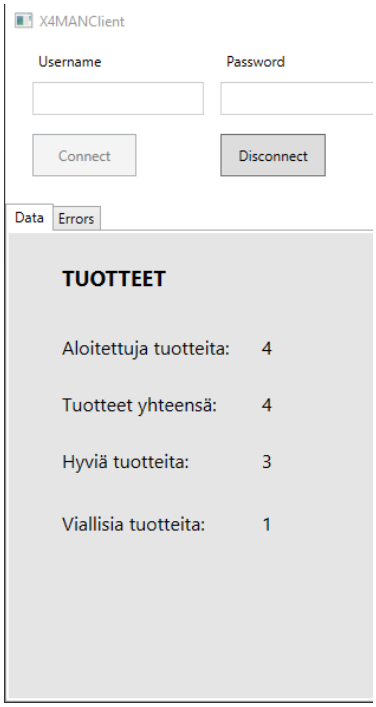
Tämän jälkeen katsotaan, onko monItemDispName merkkijonon arvo Counter ja jos on, muutetaan merkkijono val kokonaisluvuksi parsedValue:

```
if (monItemDispName == "Counter")
{
   Int32.TryParse(val, out parsedValue);
```

*Seuraavaksi katsotaan, onko parsedValue -arvo suurempi kuin valueTotal -arvo:*

```
if (valueTotal < parsedValue)
{
   startedProductsValue.Content = val;
   totalProductsValue.Content = val;
   randomValueForProduct = RandomBadGoodProduct();
   x_product.RandomProduct();
```

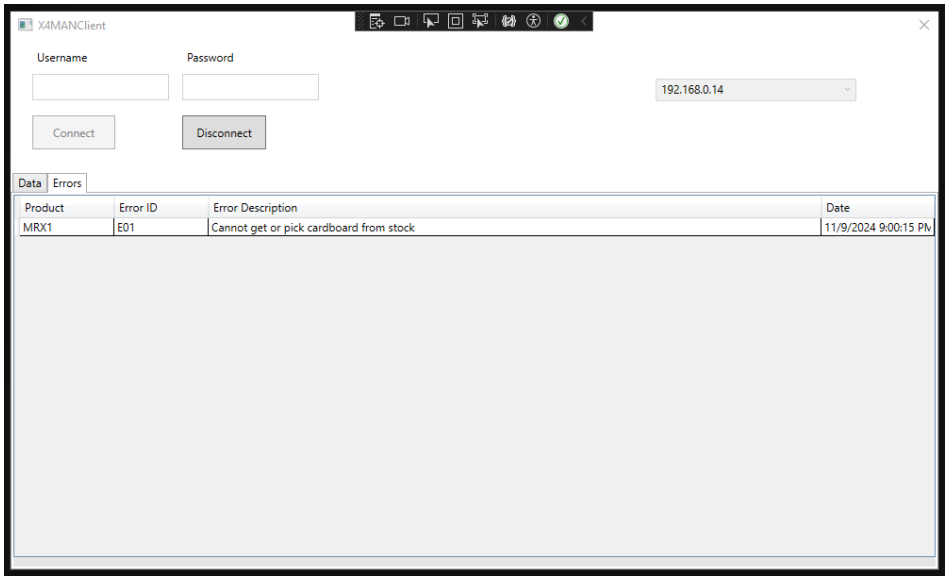
Jos ehto toteutuu, lisätään merkkijono val näkyville aloitettuihin tuotteisiin ja kokonaistuotteisiin (kuva 23).



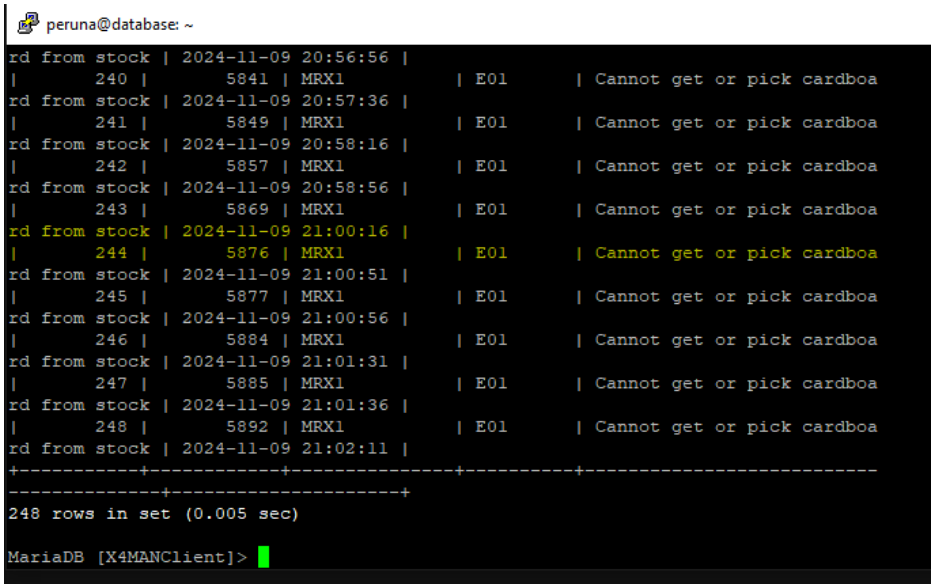

**Kuva 23.** Kaikki tuotteet.

Samassa arvotaan metodeilla RandomBadGoodProduct()- ja RandomProduct()-tuote ja onko se hyvä vai huono tuote. Jos tuote on huono, muuttujaa valueFailed kasvatetaan ja myös lisätään sovellukseen näkyville tuo arvo. Tuotteelle arvotaan myös virhe, joka sitten määritellään tuotteelle virhetauluun (kuva 24) ja kun tuote lisätään tietokantaan (kuva 25). Lopuksi kasvatetaan myös muuttujan valueTotal -arvoa:

```
if (!randomValueForProduct) // If the randomized Product failed
{
    valueFailed++;
    failedProductsValue.Content = valueFailed.ToString();

    x_product.RandomError();
    errorTable.Rows.Add(x_product.ProductNames, x_product.ErrorIDs, x_product.ErrorDescriptions, DateTime.Now.ToString());
    x_product.ProductToDatabase("bad", x_appConfigurations.GetConnection(), roundForDB, yieldPercentage); valueTotal++;
}
```



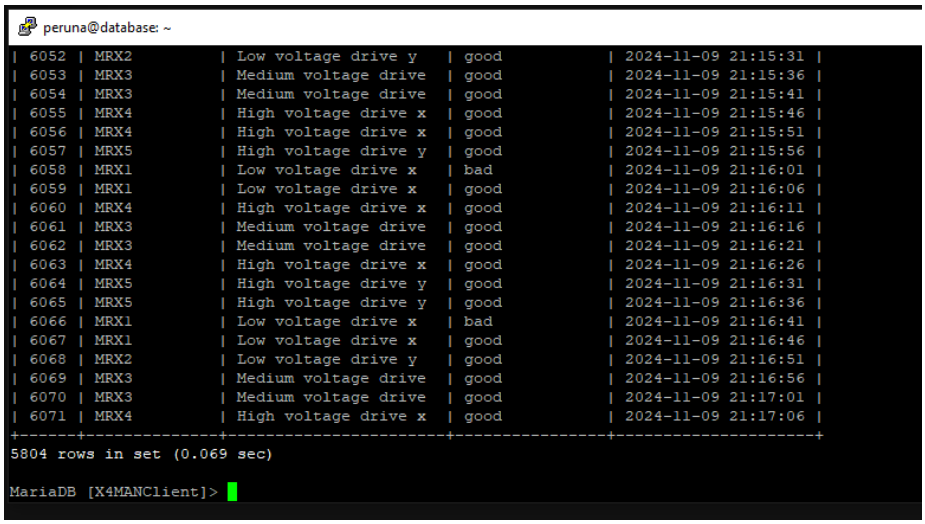

**Kuva 24.** Huono tuote virhelistalla.




**Kuva 25.** Huono tuote tietokannassa.

Kun tuote on hyvä, kasvatetaan muuttujan valueGood -arvoa ja lisätään tuote tietokantaan hyvänä (kuva 26). Tämän jälkeen kasvatetaan muuttujan valueTotal -arvoa:

```
else // Product succeeded
{
   valueGood++;
   goodProductsValue.Content = valueGood.ToString();
   x_product.ProductToDatabase("good", x_appConfigurations.GetConnec  tion(),
roundForDB, yieldPercentage);
   valueTotal++;

}
```



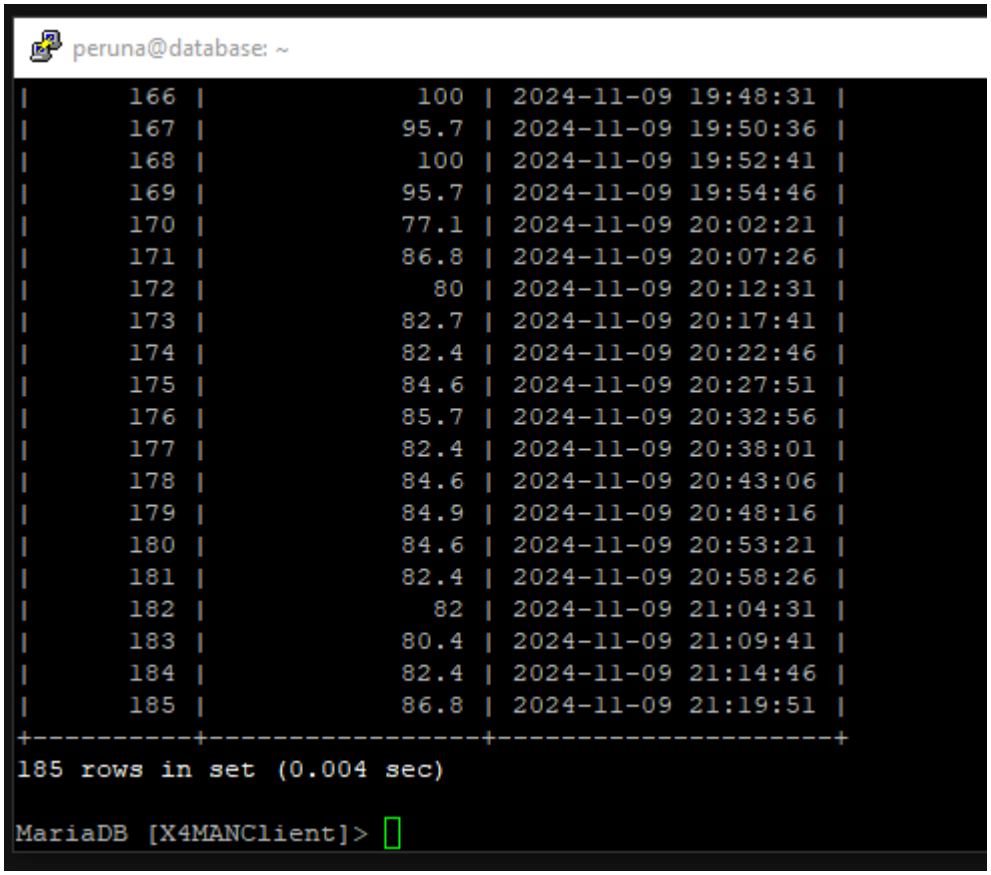

**Kuva 26.** Hyvät ja huonot tuotteet tietokannassa.

Simuloinnissa lasketaan myös Yield-arvo prosentteina, joka kertoo, kuinka monta prosenttia tuotteista on mennyt hyvänä läpi (kuva 27). Tämä arvo nollaantuu aina viiden minuutin välein, kun simulointi vaihtaa vuoroa. Lisäksi tuo kyseinen arvo lisätään tietokantaan (kuva 28) vuoron päätteeksi:

```
if (minutes == 5) // Simulation "restart" based on timer
{
    roundForDB++;
    x_product.ProductToDatabase(null, x_appConfigurations.GetConnection(),
roundForDB, yieldPercentage);
    UIReset();
    StopTimer();
    StartTimer();
    val = null;
}
```

| TYÖVUORO | | | SUORITUSKYKY | |
|---|---|---|---|---|
| Viimeisin tahtiaika: | 0 | | OEE: | 0 |
| Vuoron tahtiaika: | 0 | | OE: | 0 |
| Käynnissäoloaika: | 3 | **MIN** | YIELD: | 75,7 **%** |
| Kokonais käynnissäoloaika: | 0 | | Suorituskyky: | 0 |
| | | | Automaation käytettävyys: | 0 |

**Kuva 27.** Simuloinnin käynnissäoloaika ja YIELD.




**Kuva 28.** YIELD arvot tietokannassa.

# 6 YHTEENVETO

Kokonaisuudessaan työ oli sopivan kokoinen opinnäytetyöksi. Haasteellisuutta tuli ohjelmoinnin ja erilaisten toiminnallisuuksien kautta. Opin Raspberryn käytöstä ja myös tietokannoista. C# -ohjelmointikieltä olen harjoittanut yksittäisellä kurssilla, muuten oppiminen on tapahtunut omien henkilökohtaisten projektien kautta. Yksi tärkeimmistä asioista tässä projektissa kuitenkin oli OPC UA -standardiin perehtyminen, joka auttaa työpaikalla tulevaisuuden projekteissa, kun automaatiota lisätään huomattavasti.

Harmillisesti tätä kyseistä sovellusta ei saatu toteutettua suoraan pakkaussolulle, koska ajallisesti se olisi ollut liian haastavaa nykyhetkessä. Valmiudet käyttöönottoon kuitenkin on, kun lisenssit ja muut tarvittavat ovat valmiina. Todennäköisesti kyseinen ohjelma tullaan uusimaan uusien kirjastojen kera, jolla saadaan yksinkertaistettua ohjelmakoodia. Se helpottaa myös ohjelmakoodin opettamista muille kyseisen projektin mukana olleille.

Henkilökohtaisesti löysin myös paljon parannettavaa ohjelmointiprojektien hallinnassa ja itse ohjelmoinnissa. Tämän projektin kohdalla hallinta meni omalla kovalevyllä ja OneDriveen tallentamiseen ajan ja päivämäärän mukaan. Itse ohjelmoinnissa opin parempia debuggaustaitoja ja ymmärrystä, kuinka ohjelmakoodia on helpompi lukea.

## LÄHTEET

Danfoss Drives. (2024). *VACON®- ja VLT®-tuotteet* [kuva]. Noudettu 1.12.2024 osoitteesta https://www.danfoss.com/fi-fi/products/dds/

Danfoss. (2024). *Danfoss annual report.* Noudettu 1.12.2024 osoitteesta https://assets.danfoss.com/documents/latest/403512/AH493132301547en-000101.pdf

MariaDB. (2024). *Open source relational database*. Noudettu 6.11.2024 osoitteesta https://mariadb.org/

OPC Foundation. (2023). *OPC UA, Unified Architecture.* Noudettu 21.5.2023 osoitteesta https://opcfoundation.org/about/opc-technologies/opc-ua/

Prosys OPC. (2024). *Prosys OPC lyhyesti.* Noudettu 5.11.2024 osoitteesta https://prosysopc.com/about-us/

Raspberry Pi. (2024). *Raspberry Pi OS.* Noudettu 6.11.2024 osoitteesta https://www.raspberrypi.com/software/

ResearchGate. (2014). *Xiong qi: OPC Classic architecture* [kuva]. Noudettu 18.11.2024 osoitteesta https://www.researchgate.net/figure/Classical-OPC-architecture_fig1_273197353

Stykemain A. (2021, 21. kesäkuuta). *Photoelectric Sensor Explained.* Realpars. Noudettu 15.5.2023 osoitteesta https://realpars.com/photoelectric-sensor/

Suomen automaatioseura ry. (2023). *OPC-toimikunta, OPC historiaa*. Noudettu 18.5.2023 osoitteesta https://www.automaatioseura.fi/sas/jaostot/opc/

TheOPCFoundation. (2020, 5. heinäkuuta). *OPC UA Getting Started* [kuva]. Youtube. Noudettu 18.11.2024 osoitteesta https://youtu.be/EewQBi-JrJU?t=574