Communication between programmable logic controllers and the warehouse control system

Ilia Reshetov

Haaga-Helia University of Applied Sciences
Business Information Technology
Bachelor's Thesis
2022

Abstract

Author
Ilia Reshetov

Degree
Business Information Technology

Thesis Title
Communication between programmable logic controllers and the warehouse control system

Number of pages and appendix pages
36 + 5

Modern industrial automation systems involve dozens of programmable logic controllers (PLCs). These controllers are monitored and controlled by an execution software system that is hosted on an external server. Depending on the arrangement of a certain project's requirements, it may employ PLC from many manufacturers. The process of connecting all of the PLCs to the execution systems becomes fairly difficult as there are so many different communication protocols being used by different manufacturers. Kepware is one company that has developed a software solution called KEPServerEX that addresses this issue. This comprehensive connectivity platform has implemented over 150 PLC drivers.

The purpose of this thesis is to investigate the feasibility of the connectivity software for the commissioning company. To achieve this goal, a test framework that enabled testing of communication was designed. During the course of the project, a legacy application was described to highlight the shortcomings of the existing system. It was indicated that some particular components are required to remain unchanged throughout the migration in order to maintain backward compatibility.

PLC simulation was developed with the help of the Beckhoff TwinCAT software system, and subsequently linked to KEPServerEX with the help of one of the available connection drivers. The IoT Gateway plug-in was then configured to enable REST web service, which allowed communication tags to be read and written into PLC memory. Following that, a Node.js REST client application was developed, which implemented the IoT Gateway API. This application made it possible to browse for the list of available communication tags, as well as read and write into them. All of the names and values of the tags were saved to a relational database so that they would be available for use in business logic of an external software system.

The completion of this thesis led to the development of a functional test framework, which the commission party might eventually use in the process of integrating KEPServerEX into its own software ecosystem. It was found out that the integration procedure for the organization can end up being pretty straightforward since the kepware solution only requires a minimal number of configurations and has a graphical user interface that is easy to comprehend.

Keywords
Programmable logic controller, Warehouse control system, Manufacturing execution system KEPServerEX, IoT Gateway

Table of contents

Abbreviations

PLC	Programmable Logic Controller
WCS	Warehouse Control System
MES	Manufacturing Execution System
PC	Personal Computer
ERP	Enterprise Resource Planning
OSI	Open System Interconnection
ISO	International Organization for Standardization
IEEE	The Institute of Electrical and Electronics Engineer
LAN	Local Area Network
MAC	Media Access Control
REST	Representational State Transfer
CPU	Central Processing Unit
ORDBMS	Object-Relational Database Management System
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
AWS	Amazon Web Services
URL	Uniform Resource Locator
BI	Business Intelegence

1 Introduction

This thesis work was commissioned for the company Cimcorp Oy which specialises in the automation of material flows in tire plants and distribution centres. This work is aimed to examine how migrating to a kepware solution would improve and speed up the delivery time of customers' projects. During the past few years, the author has been working for a commissioning company as a software developer and has attended on-site commissioning activities multiple times. Therefore, he has relevant knowledge and experience of the company's legacy software, understands the relevance of the thesis topic well, and appreciates the opportunity to apply theoretical knowledge in practice and contribute to the common good of the company's business.

1.1 Case Company

Cimcorp Oy is the parent business of Cimcorp Group, which manufactures tyre automation systems. The company focuses on robotics-based automation in the tyre, food and beverage, and distribution sectors. It develops automation software solutions and assembles robots, with the bulk of them going to other countries. Ulvila, Finland, is home to Cimcorp Oy's headquarters. Cimcorp is a subsidiary of the Japanese company Murata Machinery, Ltd. (Muratec).

Cimcorp has sold its products to over 40 countries on six continents by 2019. Continental, Goodyear, Cordiant, and Tigar Tyres are among the company's customers. A tyre production line automated using Cimcorp robots reduces the number of people required; when formerly 50 plant workers were required to maintain a line, just around 20 engineers are required after automation. ("Cimcorp - Wikipedia," 2021)

1.2 Background

The Warehouse Control System (WCS) is a centralised control system that interfaces with a wide variety of material handling equipment. It has the ability to gather equipment information as well as control equipment in real-time. WCS performs the following activities from a functional standpoint: interacting with equipment, gathering equipment data, executing, and regulating material flow, and monitoring and controlling equipment. In this work, WCS described as a system that manages and controls all equipment and facilities in a warehouse environment. (Son, Chang and Kim, 2015)

Cimcorp's Warehouse Control System (WCS) software suite is a combination of Manufacturing Execution System (MES) and Warehouse Management System (WMS). In distribution centres, WCS is in charge of handling the various warehouse tasks, which include order picking and material flow management, among others. The WCS system gets all of the required data from the customer's host system, including goods, order lines, volumes, priority, and delivery deadlines. WCS may also

manage order consolidation and dispatch planning, ensuring that orders are delivered to the shipping dock in reverse drop sequence, ready to be loaded into delivery trucks as they are received. (*Cimcorp WCS for complete control - Cimcorp*, 2022)

The term Programmable Logic Controllers (PLC) refers to a special-purpose computer that is extensively used in industry for the control, automation, and monitoring of machines and processes. PLC is a cost-effective and dependable option for regulating complex systems. In order to run increasingly advanced algorithms, however, substantial computer capacity is required, which is often unavailable in PLCs. In this circumstance, the usage of an external control system that runs on a PC is preferable. Unfortunately, there is no standard that would allow all field devices to communicate in a common protocol. A protocol is a language that digital systems communicate in, according to a predefined program. Different vendors in order to obtain their own market share have developed a variety of communication protocols. (Walters and Bryla, 2016)

The total performance of plant automation systems is heavily reliant on communication between WCS and PLC. While legacy software in use today still fits the standards for which it was created, various flaws have been uncovered over time. For example, every new project's configuration and programme setup requires a significant amount of human labour, slowing down the development process as a whole. However, owing to the limited variety of new equipment in new projects, the connection programme setup procedure has been determined to be a repetitive activity that may be automated by switching to the third-party connectivity solution KEPServerEX. This modification is designed to boost development department efficiency by reducing wasted time, eliminating human mistakes, and freeing up resources to focus on more important software developments.

1.3 Benefits

Migrating towards commercial connectivity platform Kepware – KEPServerEX might bring significant benefits to the commissioning party. It helps to abstract away implementation of PLC drivers and rather focus more on the acquisition and processing of data gathered from the devices on a factory floor. It provides a user management feature along with secure login over the application programming interface (API). Advances IoT Gateway that comes along with software suite allows not just to interconnect the PLC and WCS which is the main aim of the project but also it allows to connect to cloud IoT hub where comprehensive Business Intelligence (BI) tools become available.

1.4 Objectives

During the course of this work legacy PLC to WCS communication software will be described in order to point out which elements have to persist during migration to KEPServerEX in order to properly maintain backwards compatibility.

KEPServerEX must be able to provide the connectivity between the company's in-house built software suite Warehouse Control System (WCS) and Programmable Logic Controllers (PLC) which are responsible to provide control over filed devices which are at the very bottom level are collections of sensors and actuators.

The author intended to implement a Node.js application. It is a test project to see whether KEPServerEX migration is feasible. Several API methods call must be implemented to enable interaction with KEPServerEX. The browse method returns the list of normalized control tags exposed by the PLC. The list must be persisted in a relational database whereas it is required to map device tags between PLC and WCS. Read method will be executed as a polling routine in order to acquire tag values and persist them in the database. Write method will allow to update tag value on the PLC side.

An oversimplified version of the interface between PLC and WCS looks as follows:

- Read command returns a list of requested tag values to WCS. Based on those values WCS business logic gets executed.
- Write command is used by WCS to command PLC to execute certain routines, for instance to initialize transfer from one conveyor to another.

The main objectives of the thesis are:

- To investigate the suitability of the commercial KEPServerEX software and of advanced KEPServerEX plug-in IoT Gateway.
- To implement a Node.js application where communication can be tested.

1.5 Limitations

In this work apart from the need to develop a Node.js application PLC simulation is required. The author is most familiar with the TwinCAT software system and decided to use it to create a simple PLC program. Other PLC vendors will not be included in the scope of the thesis.

2 Theoretical framework

The basic ideas, fundamental terminology, and theoretical foundation of this thesis are presented in the following chapter. Figure 1 shows common industrial automation levels. This thesis only considers communication between planning on the control level. Because there was no standard protocol available when the first PLC was introduced, numerous different proprietary communication protocols have been created by PLC manufacturers. This is because the first PLC was introduced many years before the public internet. The job of intercommunication between planning and control levels is made more difficult as a result of this diversity. In the following paragraphs, the primary software that is used over the course of this study will be discussed.

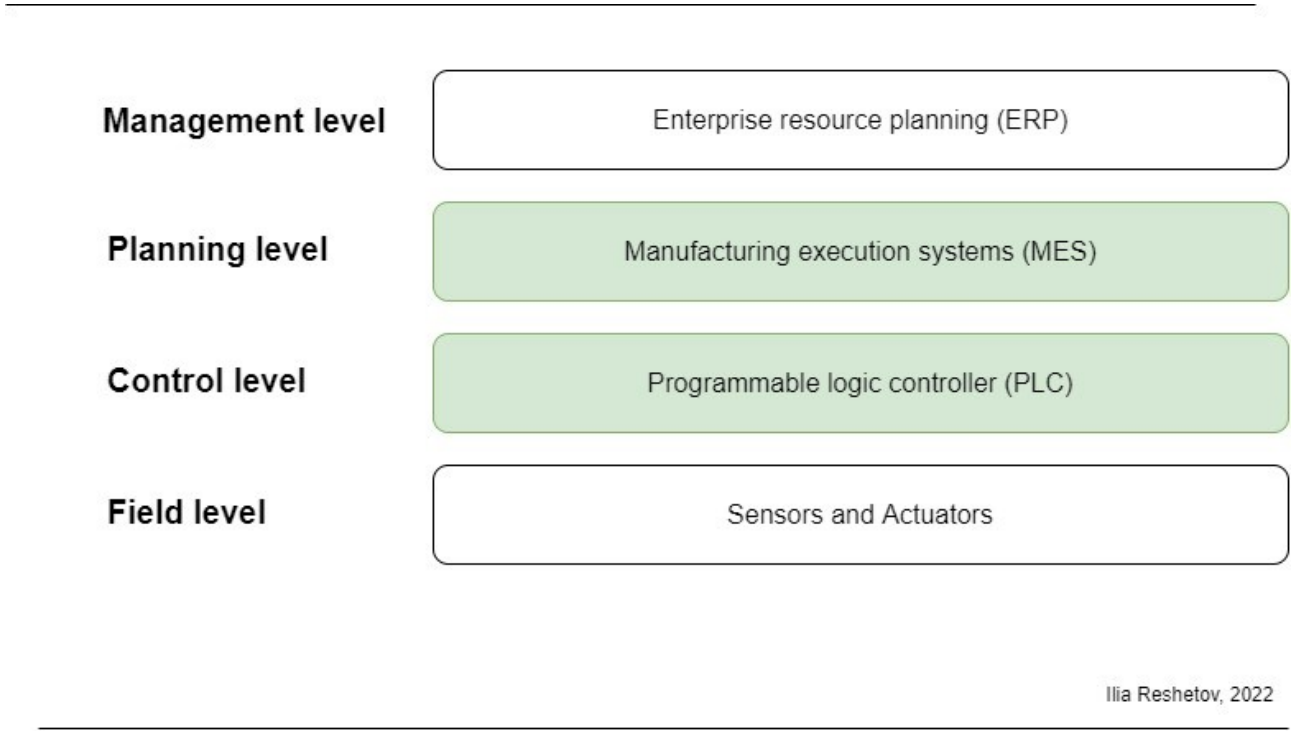


Figure 1 – Common Industrial automation framework

2.1 Programmable logic controllers

A programmable logic controller (PLC) is computer-based equipment used in a range of processing plants and manufacturing applications to conduct discrete or continuous control operations. Originally designed to replace relays in the automotive sector, the PLC is today utilised in almost every industry. Though they were often referred to as personal computers before 1980, the abbreviation PLC became the preferred acronym for programmable logic controllers as the name "PC" grew associated with personal computers in the subsequent decades. (Liptak, 2018a)

The first PLC for General Motors Corporation was built in 1968 to minimise the expensive discarding of assembly-line relays during model changeovers. It brought a fundamental benefit in that they employed programming rather than rewiring to configure for a new application. PLC programming might be completed considerably more quickly than old rewiring techniques. (Liptak, 2018b)

Control duties (which may be time sensitive) are often executed entirely by the PLC, while monitoring and supervisory functions are handled by the personal computer (PC). There is a clear distinction between the activities in this manner, and the PC–PLC connection may be broken without causing damage to the plant or machine under management. (Liptak, 2018b)

There are three ways to connect the PC to the PLC:

- Serial (9.6 to 38.4 Kbps)
- Fieldbus (9.6 Kbps to 12 Mbps depending on which fieldbus is used)
- Ethernet TCP/IP (100 Mbps to 1 Gbps)

2.1.1 PLC to PC Serial connection

Historically, point-to-point serial link connections have been widely utilised due to their ease of use and affordable price. In terms of hardware, two interface boards and a connecting wire are all that is required to make such a connection. It is only essential to equip the PLC with a suitable board since the PC is normally equipped with a serial interface. The RS-2321 and RS-4222 serial point-to-point connections are the most common. With regard to the International Standards Organization's (ISO) model for Open System Interconnection (OSI), these two standards have nothing to do with other aspects of a link's functionality. Normal values for these connections are 9.6 Kbps, 19.2 Kbps, and 38.4 Kbps in terms of transmission speed. Because of this, only limited quantities of data may be sent across such networks without slowing down the whole transmission system. Finally, serial connections can only transmit across a few tens of metres of distance. (Liptak, 2018b)

2.1.2 PLC to PC connection over communication networks

The Institute of Electrical and Electronics Engineers (IEEE) 802 committee standardised local area network (LAN), and the International Standards Organization eventually acknowledged the findings in its 802 families of standards. Despite the fact that the IEEE committee provides numerous options, practically all LAN systems are based on the media access control (MAC) described by the IEEE 802.3 subcommittee. This MAC is often referred to as Ethernet, and it is also used for PC–PLC communications. Ethernet interface boards for both PCs and PLCs are accessible out of the box and frequently utilised. (Liptak, 2018b)

Fieldbuses are computer networks that are used to connect devices and controllers in plant-wide automation systems. Fieldbuses enable the implementation of control loops by connecting a controller to its sensor (transmitter) and final control element (valve or actuator) at the "device" (lowest) level of plant-wide automation systems. Fieldbuses link intelligent devices at the system or intermediate level. In contrast to LANs, where Ethernet is the most extensively used PC–PLC link, there is no dominant product among the many fieldbuses.

Because each fieldbus product has its own interface board, and all interface boards for all of these systems are not always accessible, LANs are a better choice for PC–PLC communications than fieldbuses. This is most certainly related to the relatively slow rate of international standardisation. As a result, a variety of proprietary fieldbus solutions are available on the market today, including DeviceNet, ControlNet, Fieldbus Foundation, Interbus, Profibus, and WorldFIP. All of these items are now covered by a variety of European and international standards, including EN50170,6, EN50254,7, EN50325,8, and IEC61158.9. However, using the same fieldbus across the plant assures that the connections at the device level of industrial automation systems are compatible. Fieldbuses are often utilised due to this reason, as well as due to their high performance. Compared to serial connectivity, communication networks provide a number of benefits. (Liptak, 2018b)

Communication networks, for an instance, are quicker in terms of transmission speed and are capable of transmitting across greater distances. Additionally, they may offer multilayer protocols that improve the efficiency of data transmission sessions. Ethernet transmission speeds are typically 100 Mbps. The maximum range between a station and a switch is 100 metres; however, numerous available network components enable the implementation of extremely complex configurations, extending the range between PC and PLC to several thousands of metres. (Liptak, 2018b)

Additionally, 1 Gbps Ethernet components are currently accessible, and the migration to 10 Gbps is projected to occur very shortly. The speed of the fieldbus transmission is product-dependent. For example, Profibus operates at a data rate of 12 Mbps, Interbus operates at a data rate of 2 Mbps, ControlNet operates at a data rate of 5 Mbps, and DeviceNet operates at a data rate of 500 Mbps. Although the following speeds are slower than those of Ethernet, it must be recognised that fieldbus protocols are generally more efficient, allowing for the similarly great result of fieldbus-based PC–PLC connections. The transmission rate of a fieldbus is determined by the manufacturer, although nearly all of the existing standards are capable of transmitting across several hundred metres. (Liptak, 2018b)

2.2 Manufacturing execution system

At the end of the 1990s, it became clear that better and quicker product information systems were required. Initially, it was thought that incorporating the automation level into enterprise resource planning (ERP) would make a separate production management level obsolete. The outcomes were underwhelming. This is also acceptable given that the dependable but time-consuming management and settlement level does not fit the real-time results-oriented world of production. Production discontent with real-time information in a substantial number of production businesses led to the development of production standards in stages.

One such example is the International Standards on Auditing (ISA) and its guidelines. The name manufacturing execution system (MES) originated with the Manufacturing Enterprise Solution Association's (MESA) development of 11 functions for a production system in the early 1990s. (Meyer, Fuchs and Thiel, 2009)

All of these functions take place at various levels throughout the workshop in order to accommodate the requirements of each stakeholder in the manufacturing process. A MES software is not required to fulfil all of ISA95 features in order to generate uniformity in the offer. The 11 functions are defined by ISA95 (INFODREAM INC., 2022):

- Data acquisition
- Scheduling
- Staff management
- Resource management
- Production tracking and dispatch
- Product traceability
- Quality management
- Process management
- Performance analysis
- Document management
- Maintenance management (INFODREAM INC., 2022)

A MES allows data to flow between the organisational level, which is often backed by an Enterprise Resource Planning (ERP), and the shop floor management systems, which are typically made up of many separate, highly specialised software programmes. (D’Antonio, Bedolla and Chiabert, 2017)

A MES serves two primary functions. Top-down data flow is first and foremost a concern for this system, which means that needs and necessities offered by organisational levels must be translated into an ideal sequence planning that meets such goals. There are a number of restrictions that must

be taken into consideration in order to determine the optimal sequence for using the available resources (such as personnel, machinery, materials, and inventories).(D'Antonio, Bedolla and Chiabert, 2017)

The second goal of a MES is to control the data flow from the bottom up. The shop floor can collect data on process performance and product quality; the role of MES is to collect these data, analyse them using appropriate mathematical techniques, and extract synthetic information to provide the business level with a complete picture of the current state of the process. The analysis may need to be done in real-time in order to make the appropriate choices to regulate the process. There has been a recent explosion in the use of monitoring systems to evaluate the quality of products and the efficiency of manufacturing processes and to help drive such improvements forward.

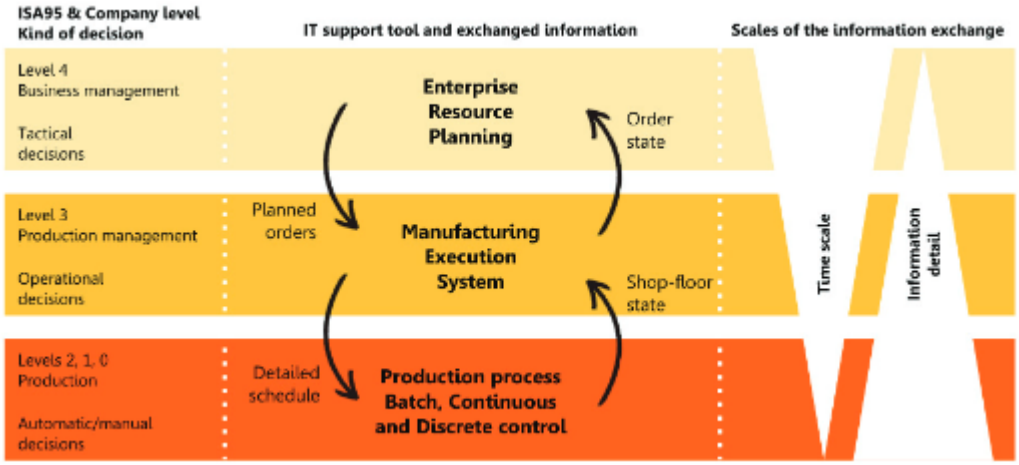


Figure 2 - MES placement within an industrial context (D'Antonio, Bedolla and Chiabert, 2017)

2.3 Kepware – KEPServerEX connectivity platform

Kepware Technologies is a software business of PTC Incorporated, headquartered in Portland, Maine. Kepware's main software product is KEPServerEX, which was founded in 1995 and is currently deployed in over 100 countries, assisting tens of thousands of organisations in enhancing their operations and decision-making. (*KEPServerEX Connectivity Platform | OPC Server | Kepware*, 2022)

KEPServerEX is one of the best connectivity platforms in the industry. It serves as a single gateway between industrial automation and your business applications. The platform architecture enables users to connect, manage, monitor, and control various automation devices through a single user interface. (*KEPServerEX Connectivity Platform | OPC Server | Kepware*, 2022)

KEPServerEX makes leading automation, Big Data, and analytics software accessible through OPC, proprietary protocols, and IT (Information Technologies) protocols (including MQTT, REST, ODBC, and SNMP). (*KEPServerEX Connectivity Platform | OPC Server | Kepware*, 2022)

2.3.1 KEPServerEX - IoT Gateway is an advanced plug-in

The capabilities of the KEPServerEX connection platform may be further expanded with the help of a sophisticated plug-in called the IoT Gateway. It allows data streams to be sent from field divisions over REST services into management systems as well as into the cloud, which in turn provides business analytics and interactive dashboards. (*Stream Industrial Data with the IoT Gateway | Kepware*, 2022)

2.4 PostgreSQL - relational database

PostgreSQL is an Object-Relational Database Management System (ORDBMS) that has been under development since 1977 in different variants. It all started with the Ingres project at the University of California, Berkeley. Ingres was eventually commercialised by Relational Technologies/Ingres Corporation. In 1986, another team headed by Berkeley's Michael Stonebraker continued the development of the Ingres code to build Postgres, an object-relational database system. After a short period as Postgres95, Postgres was renamed PostgreSQL in 1996 as a result of a new open-source initiative and the software's improved capabilities. (Worsley, 2002)

The PostgreSQL project is still being actively developed globally by a group of open-source volunteers. PostgreSQL is largely regarded as the most technologically sophisticated open-source database system in existence today. It has numerous capabilities that have historically been found solely in commercial solutions of enterprise-level software. (Worsley, 2002)

2.5 TwinCAT – software platform

One of the benefits that Beckhoff has made available is the capability to transform any personal computer into a programmable logic controller, which can then be used to operate field equipment. (*TwinCAT | Automation software | Beckhoff Worldwide*, 2022)

One of TwinCAT’s 3 key goals is to make software engineering easier. It is evident that integrating into popular and current software development environments is preferable to designing own independent solutions. Microsoft Visual Studio is the development environment for TwinCAT 3. Beckhoff provides the user with an extendable and future-proof platform by integrating TwinCAT 3 as an extension into Visual Studio. (*TwinCAT | Automation software | Beckhoff Worldwide*, 2022)

TwinCAT 3 Runtime provides a real-time environment for loading, executing, and administering TwinCAT modules. The separate modules must not be generated using the same Compiler, allowing them to be independently written by different manufacturers or developers. Furthermore, it is irrelevant whether the modules are PLC, NC, CNC, or C-Code produced. (TwinCAT | Automation software | Beckhoff Worldwide, 2022)

Multicore CPU capability is another feature that distinguishes TwinCAT 3. Individual TwinCAT Tasks may be assigned to distinct CPU cores. Hence the optimal performance of the most recent multicore Industrial- and Embedded PCs may be realised. (TwinCAT | Automation software | Beckhoff Worldwide, 2022)

2.6 Node.js

Node.js is a fascinating framework for creating web apps, application servers, network servers and clients, and general-purpose programming. Through a clever blend of server-side JavaScript, asynchronous I/O, and asynchronous programming, it is intended for great scalability in networked applications. Despite being just 10 years old, Node.js has swiftly gained notoriety and is already playing an important role. Companies of various sizes are adopting it for big and small-scale initiatives. PayPal, for example, has switched several Java services to Node.js. The Node.js architecture deviates from a common decision made by other application platforms. Whereas threads are often used to grow a programme to occupy the CPU, Node.js avoids threads due to their inherent complexity. It is claimed that single-thread event-driven systems provide a low memory footprint, fast throughput, a superior latency profile under load, and a simpler programming style. The Node.js platform is seeing significant development, and many consider it as an appealing alternative to classic web application frameworks such as Java, PHP, Python, or Ruby on Rails. (Herron, 2020)

The Node.js core modules are broad enough to create any kind of server that uses any TCP or UDP protocol, such as Domain Name System (DNS), Hypertext Transfer Protocol (HTTP), Internet Relay Chat (IRC), or File Transfer Protocol (FTP). While it can be used to build internet servers or clients, its main application is for regular website development, either in place of technology like an Apache/PHP or Rails stack, or to supplement existing websites—for example, adding real-time chat or monitoring existing websites is simple with the Socket.IO library for Node.js. Because of its lightweight and high-performance characteristics, Node.js is often utilised as a glue service. (Herron, 2020)

The Node.js design is known to have less overhead than thread-based architectures since it is built on a single execution thread with an inventive event-oriented, asynchronous-programming approach

and a fast JavaScript engine. Other concurrency systems have memory expense and complexity, whereas Node.js does not. Microservices are a hot topic in software development right now. Microservices are designed to break down a huge web application into smaller, more focused services that can be built quickly by small teams. While the microservice design is not technically new—more it is of a reframe of previous client–server computing models—it works well with agile project management methodologies and allows for more granular application deployment. (Herron, 2020)

3 Implementation of communication test framework

The previous section has discussed technologies utilised during the project. This section is dedicated to defining a test framework that will allow verifying of whether the kepware solution is suitable for commissioning company needs. It has several parts. In the first part legacy software architecture is discussed to identify known flows that new solution has to overcome. The second part will define the test framework in detail as it is quite an important task before jumping into actual software implementation. The last part contains walkthrough instructions in regards to how to configure KEPServerEX as well as the IoT gateway plugin.

3.1 Project requirements

The main goal of the project is to verify that commercial Kepware – KEPServerEX software could fulfil the company's requirements in regards to WCS to PLC communication. The task will be accomplished by developing a Node.js application that is going to implement the client-side of IoT gateway API with the following commands available: browse, read and write. Another crucial part of the application would be the persistence layer that would connect to the database on the start-up of the application and create the required schema, table, and index if those do not exist yet.

3.2 Description of a legacy communication software

In the past Java application has been developed to maintain connectivity between the control system and PLC. Figure 3 shows the simplified architecture of the tag communicator. The main functionality of the software is to read tags. It has three different drivers available which are enough to support standard devices that are commissioned to the customers' projects. However, some projects might go well beyond the scope of standard devices and the need to ingrate some third-party equipment might appear and therefore new driver's implementation would require. Implementation of the new driver is very time-consuming for software developers. Another drawback that was reported by project teams while using the Java application is that it does not have a graphical user interface thus every new project has to be configured by using old sophisticated scripts that will import control tags definitions from an Excel spreadsheet.

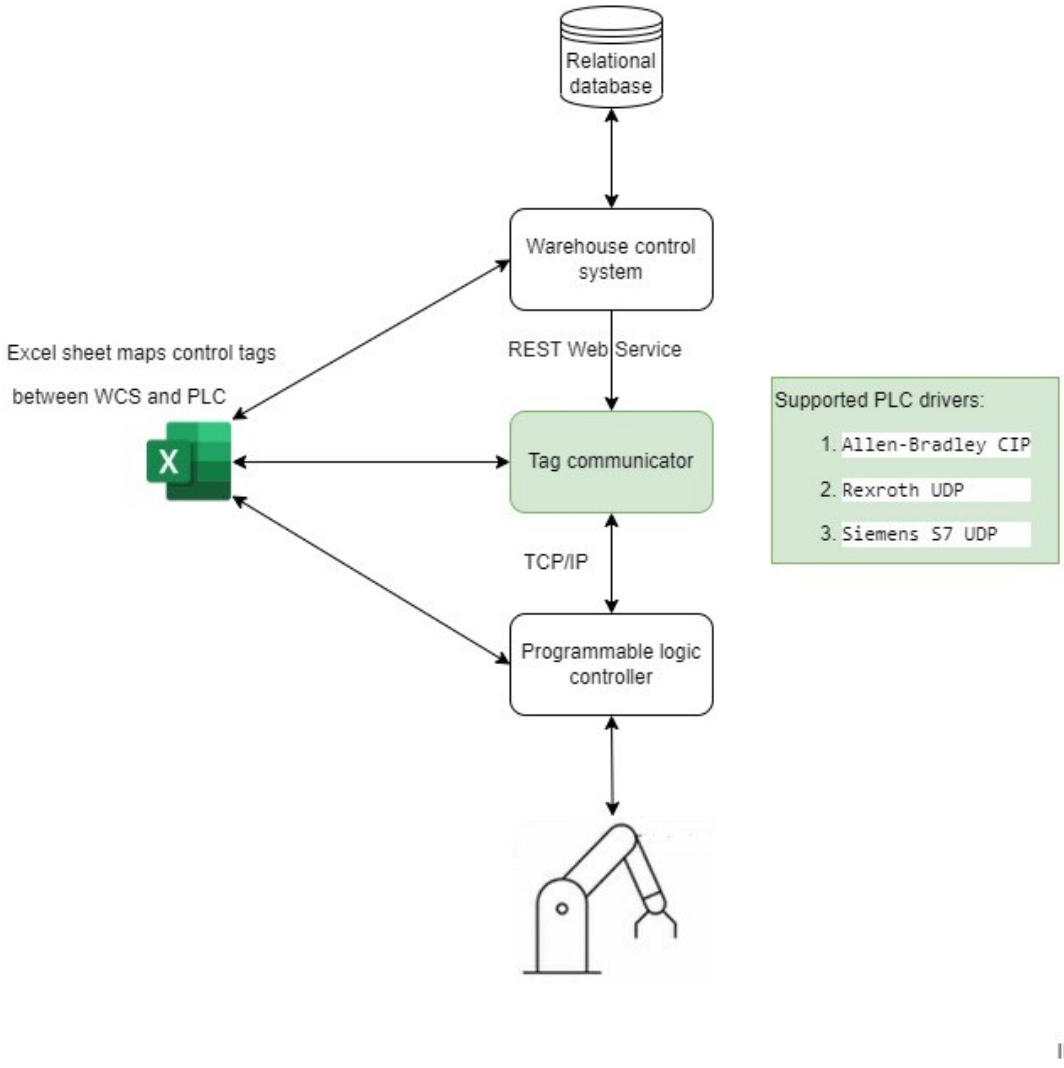


Figure 3 - The tag communicator architecture diagram

3.3 Defining test framework

As per the contract agreement with the commissioning company the author has to build an environment where communication could be tested. This thesis might also be used as setup instructions.

Beckhoff TwinCAT 3 version 3.1.4024.22 development environment that has integration with Microsoft Visual Studio used during the course of this project in order to simulate PLC. Since Beckhoff is a PC-based PLC thus the actual device is not required to test the communication.

KEPServerEX software kit has an IoT gateway plugin available out of the box as well as over 150 vendor-specific PLC drivers. The test was carried out with KEPServerEX version 6.11.718.0.

Node.js version used in this project is 16.13.0. Node application dependencies are defined in file called package.json as follows:

- "axios": "^0.27.2" promise based HTTP client
- "dotenv": "^16.0.0", module that loads environment variables
- "node-fetch": "^3.2.4", a light-weight module that brings Fetch API to Node.js.
- "pg": "^8.7.3", non-blocking PostgreSQL client for Node.js.

A diagram in Figure 4 describes the planned framework. It was decided to go with the TwinCAT software package since the author has the greatest experience with it. Another advantage that comes with using TwinCAT is that it makes it possible to convert any PC into a PLC. As a result, there is no longer a need to acquire specialised hardware in order to put the idea into action.

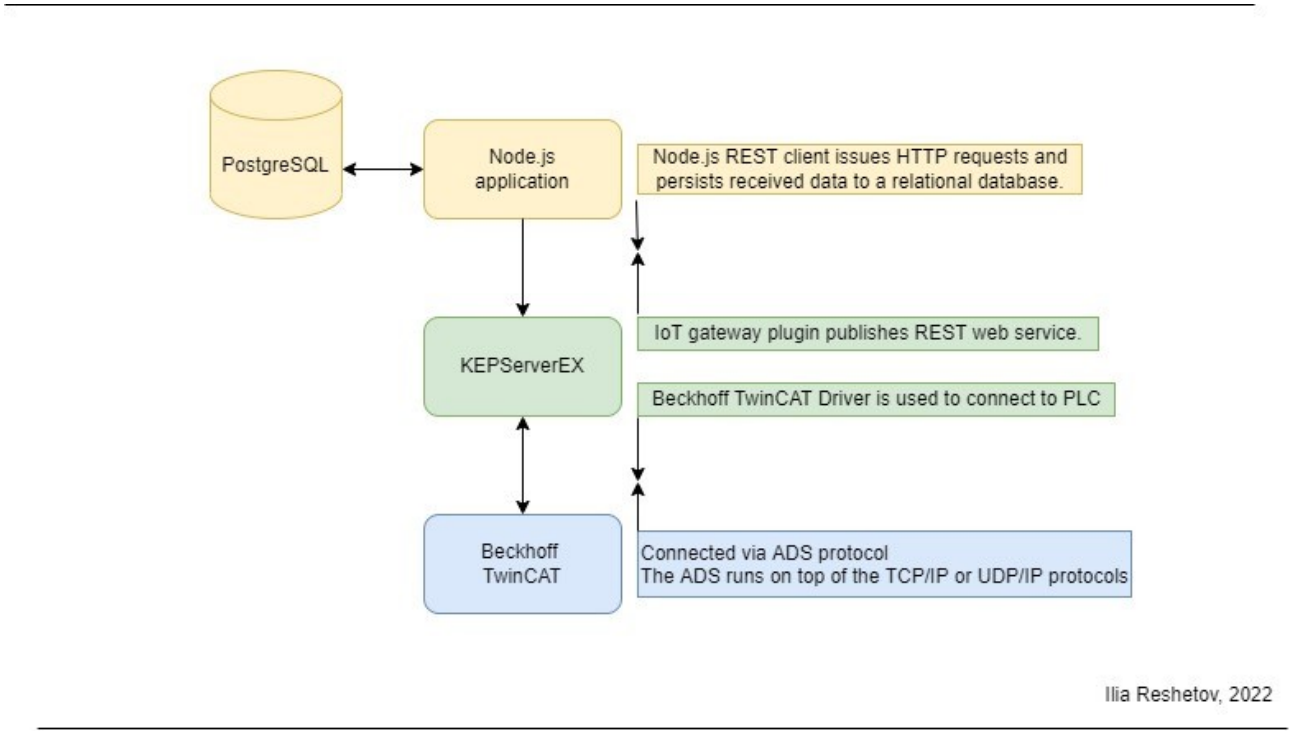


Figure 4 – Communication test framework

3.4 Implementation details

Both TwinCAT and KEPServerEX are windows based solutions and therefore windows operating system is required. To begin, one needs to install PostgreSQL and Node.js, both of which are open-source and freely accessible. TwinCAT and KEPServerEX offer a free development version with certain limits that meets the needs of this project, hence a commercial version is not required.

The implementation phase is divided into four modules:

- Create TwinCAT project to enable PLC simulation.

- Configure KEPServerEX.
- Connect IoT Gateway to TwinCAT simulation and configure REST service server
- Develop and test the Node.js application.

3.4.1 TwinCAT simulation setup

TwinCAT default boilerplate is used to create generic simulation PLC project. Following the generation of the project, the global variable list needs to be specified. (Figure 5). These variables are accessible to the outside world, and they play a vital part in making it possible for higher-level systems to govern PLC by reading from and writing to those variables. It is essential to be aware that time-sensitive activities, such as safety functions, should not depend on commands from an external system; rather, the PLC should be fully responsible for controlling these activities. When deciding which variables to publish, this is something that should be kept in mind at all times.

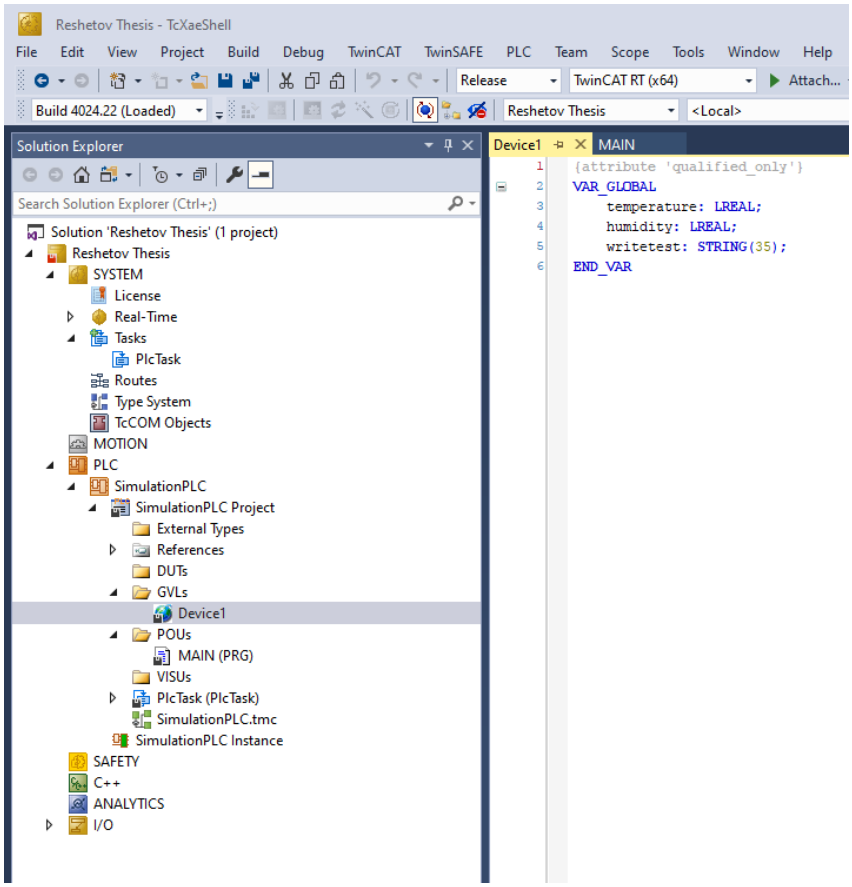


Figure 5 - TwinCAT global variable list

The next step would be to implement PLCs main program that would generate random values and assign them to humidity and temperature variables on every program scan (Figure 6). This would

emulate situations that occur in the real world when the values of communications tags are frequently modified. The Beckhoff system library includes a function block that allows you to generate a pseudo random number of type LREAL. It needs the input of the seed value, which is the beginning value for specifying the random number series. With double precision, function block output produces a pseudo-random value between 0.0 and 1.0. When generated the value assigned sequentially to global variables. Values get updated on every program scan. This configuration is sufficient to demonstrate that the developed Node.js client application is able to read and reliably persist PLC communication tag values to a relational database without experiencing substantial delays.

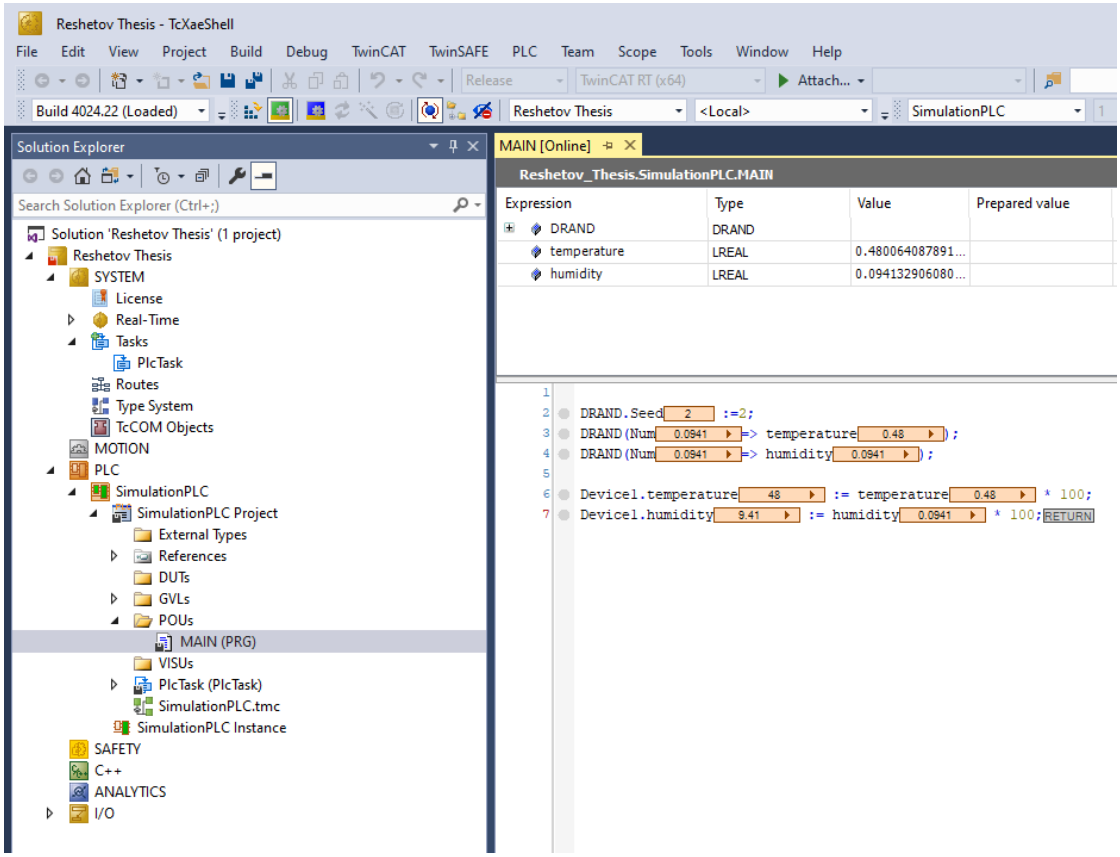


Figure 6 - TwinCAT main program

3.4.2 KEPServerEX - add a channel

In comparison to the legacy application that commission organization has used in the past, the user interface of KEPServerEX is quite straightforward. The author was able to configure the necessary setup on their own with no assistance other than the user manual that was supplied by the vendor.

Before connecting new PLC devices, the user should keep in mind that each connection channel in KEPServerEX has its own execution thread. Utilizing one channel for each PLC is advised to provide the highest possible performance.

When configuring the connectivity channel user is prompted with the extensive wizard that walks through every step. In the event that the production environment is not known, it is possible that some of the channel configuration specific parameters might be kept at their default settings.

In order to complete the work required for this thesis, the localhost instance of TwinCAT is utilised. When one is making a channel, they need to make certain that they choose the appropriate PLC driver, which in this instance is known as Beckhoff TwinCAT. If a personal computer has more than one network adapter, the user is responsible for ensuring that the appropriate one is picked. Within this particular scenario, the default network adapter was used. (Figure 7Figure 7).

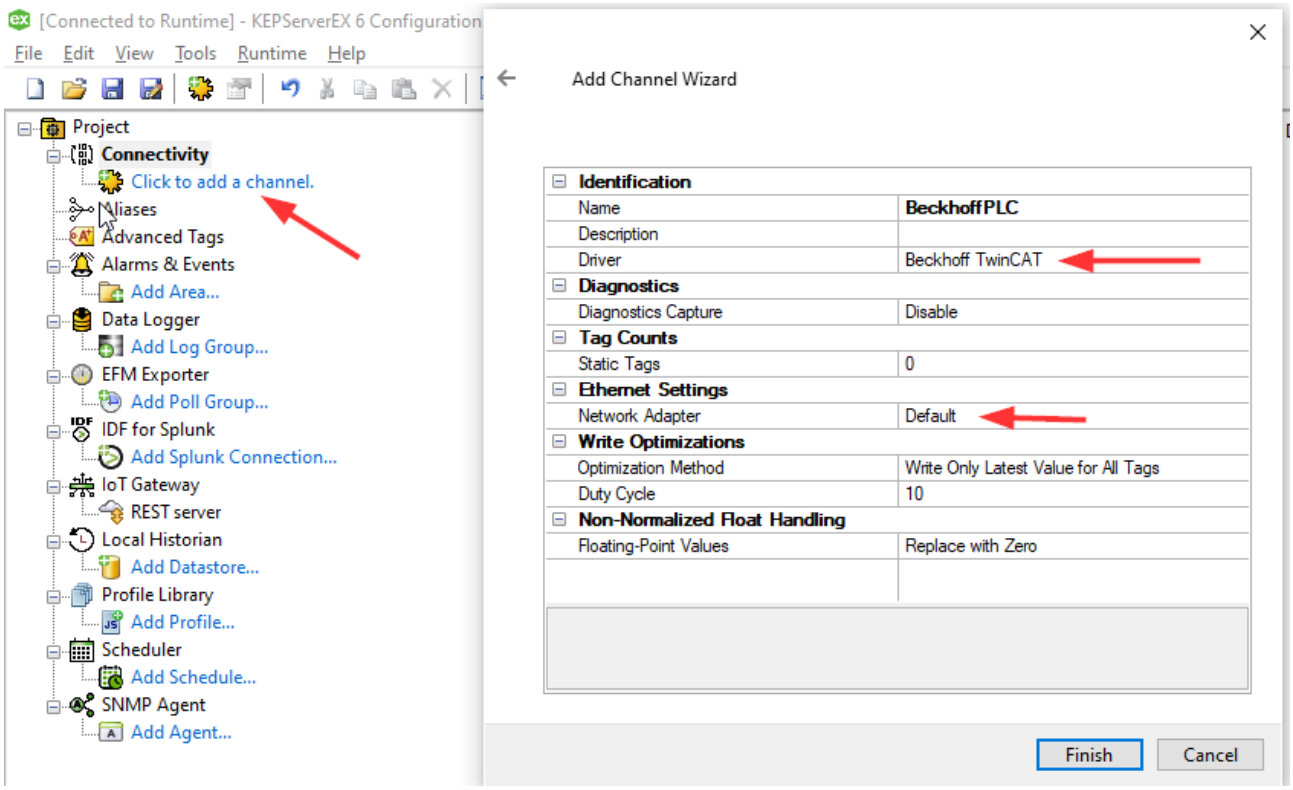


Figure 7 - Configuring connectivity channel to Beckhoff TwinCAT

3.4.3 KEPServerEX - add a device

Similarly, to the preceding section, the bulk of the options may be left at their default values. In the next paragraphs, the emphasis will be placed more on the configurations that are necessary for the completion of the project. It is necessary to pick the appropriate driver for the Beckhoff TwinCAT yet again.

The AMS Net ID and port number are the two identifiers that are used in the process of implementing the unique identification of TwinCAT devices. The ID is set when TwinCAT runtime is configured. Within the TwinCAT network, the address of the local computer is referred to as the AMS Net ID. The dot notation is used to denote the AMS Net ID, which is comprised of 6 bytes altogether. The "Net IDs" are required to be given out by the project planner, and duplicates inside the TwinCAT network are not allowed. The IP address of the system is added to "1.1" to produce an AMS Net ID during the installation. This is the option that is used by default. The AMS Net ID on the test PC was determined to be 172.21.52.65.1.1. Port 851 is being used for connectivity in this project (Figure 8).

When KEPServerEX is connected to a PLC, it has the capability of automatically generating communication tags, which is an additional highly useful function that KEPServerEX offers. In addition to enabling the generation of tags, it also permits the normalisation of data. Tags get created in the local database. The local database is updated with the newly added tags. That might be exported as a configuration file and used as a template for setting up software at a later time.

The job of software engineers is made easier by the use of this kind of framework. When modifications need to be made to the communication tags, they just need to be modified in a single location inside the PLC program, and then the changes are propagated to the systems that are upstream. Because of this improvement, both the likelihood of a mistake being made by a person and the quantity of work that has to be done by hand have been greatly reduced.

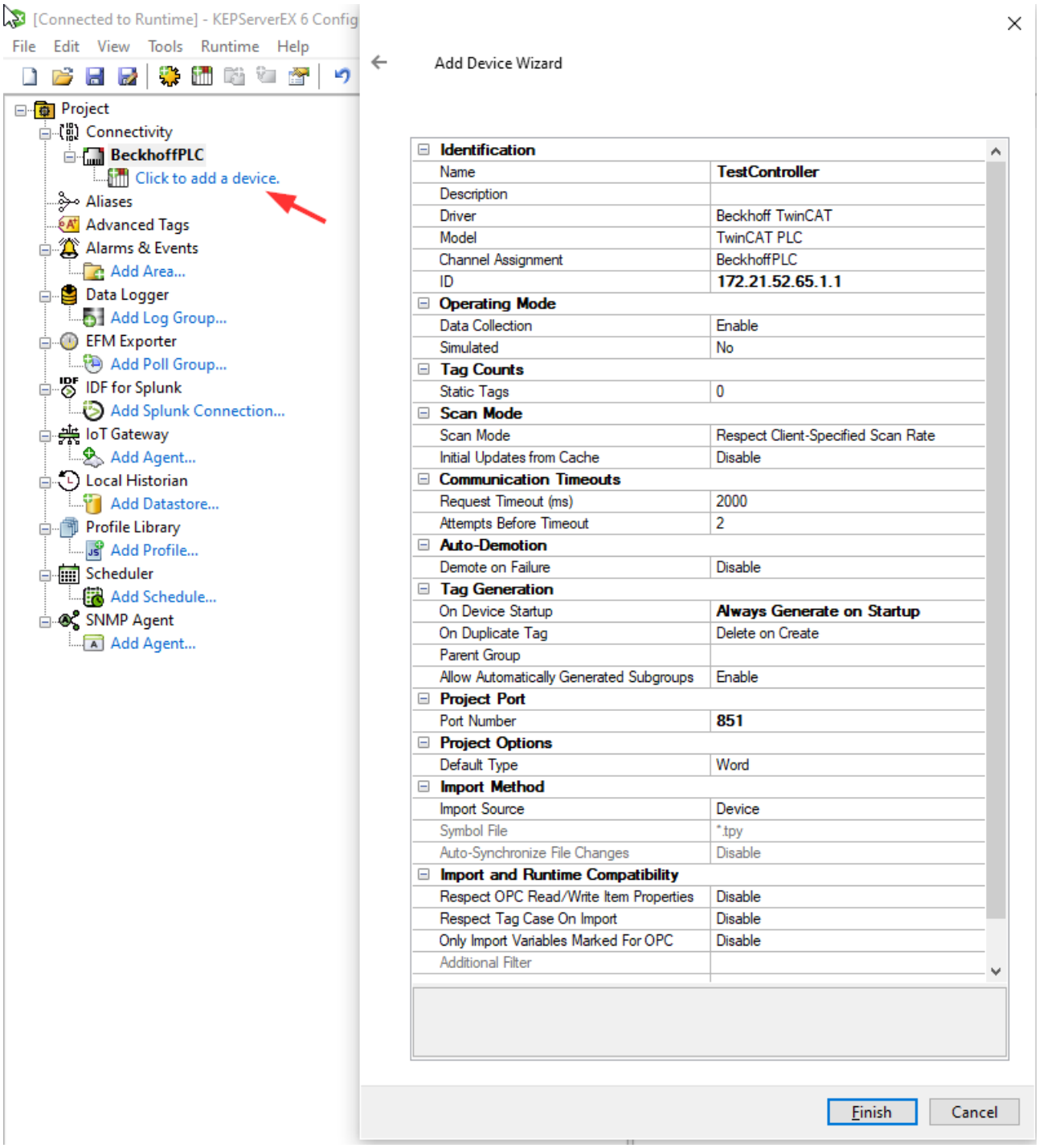


Figure 8 - Configuring connectivity channel to Beckhoff TwinCAT

3.4.4 OPC Quick Client – test connectivity

The building of OPC applications is simplified with the help of the OPC Quick Client. It is a comprehensive OPC client that covers every action that can be performed by an OPC client programme. Users get access to all of the data that is stored in the server application when they make use of the OPC Quick Client. This includes all of the system tags, diagnostic tags, and user-defined tags. Users are granted the ability to read and write data, carry out structured test suites, and test the performance of the server using the OPC Quick Client. Assisting in the diagnosis of typical OPC client/server difficulties is its thorough error reporting, which offers specific feedback about any OPC faults that are supplied by the server. (Support Center | Customer Resources | Kepware, 2022)

After the connection channel and device have been added to the project, OPC Quick Client may be used to test communication. Figure 9 shows that the connection was successfully established and Device1's tags were updated with a scan rate of 100 milliseconds. Both the temperature and the humidity tags contain values that are randomly generated and are actively being updated. They are going to be put to use in a test that will evaluate the read capabilities of the Node.js application. Because it is intended to be used to test the writing capacity of the Node.js client application, the device tag that is now named as "writetest" does not yet have any value associated with it.

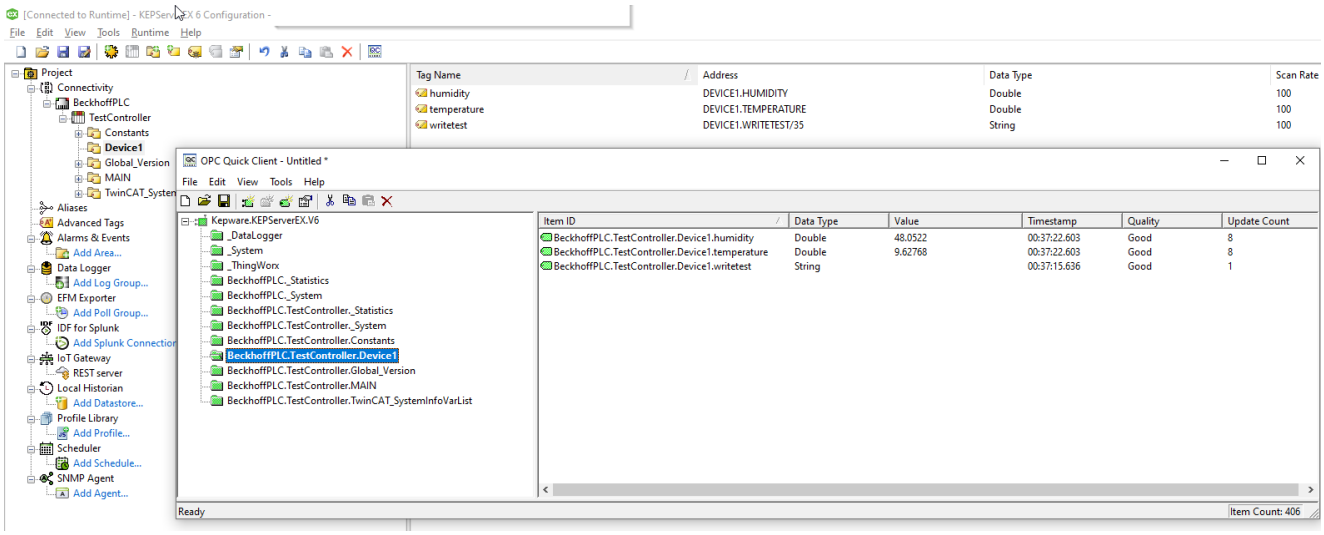


Figure 9 - Test connectivity with OPC Quick Client

3.4.5 IoT gateway plugin, add agent

The IoT Gateway for KEPServerEX makes use of the web-friendly protocol HTTP to enable the delivery of data points from programmable logic controllers and other devices to third-party endpoints and other clients. The data that is delivered is a payload that adheres to the JSON protocol and contains the item ID, value, quality, and timestamp.

It is essential to point out that the installation of a 32-bit Java JRE version 7 or later is necessary for the IoT plugin to function properly.

This project makes use of HTTP since the development environment is well-suited to the protocol's requirements. In a production setting, however, only the HTTPS protocol should be utilised since it encrypts the data sent between the remote client and server.

The IoT plugin has a user management function that restricts access to authorised users only. Authentication credentials in the header of any client connection must match a valid account in the User Manager or Security Policy Plug-in by default. In this project, anonymous login is utilised to facilitate development.

Figure 10 shows configuration wizards for setting up the REST server agent, which is required to enable the write endpoint and for anonymous login. If one were to copy the URL: http://127.0.0.1:39320/iotgateway/ address that was produced, they could then paste it into a web browser to see the API documentation.

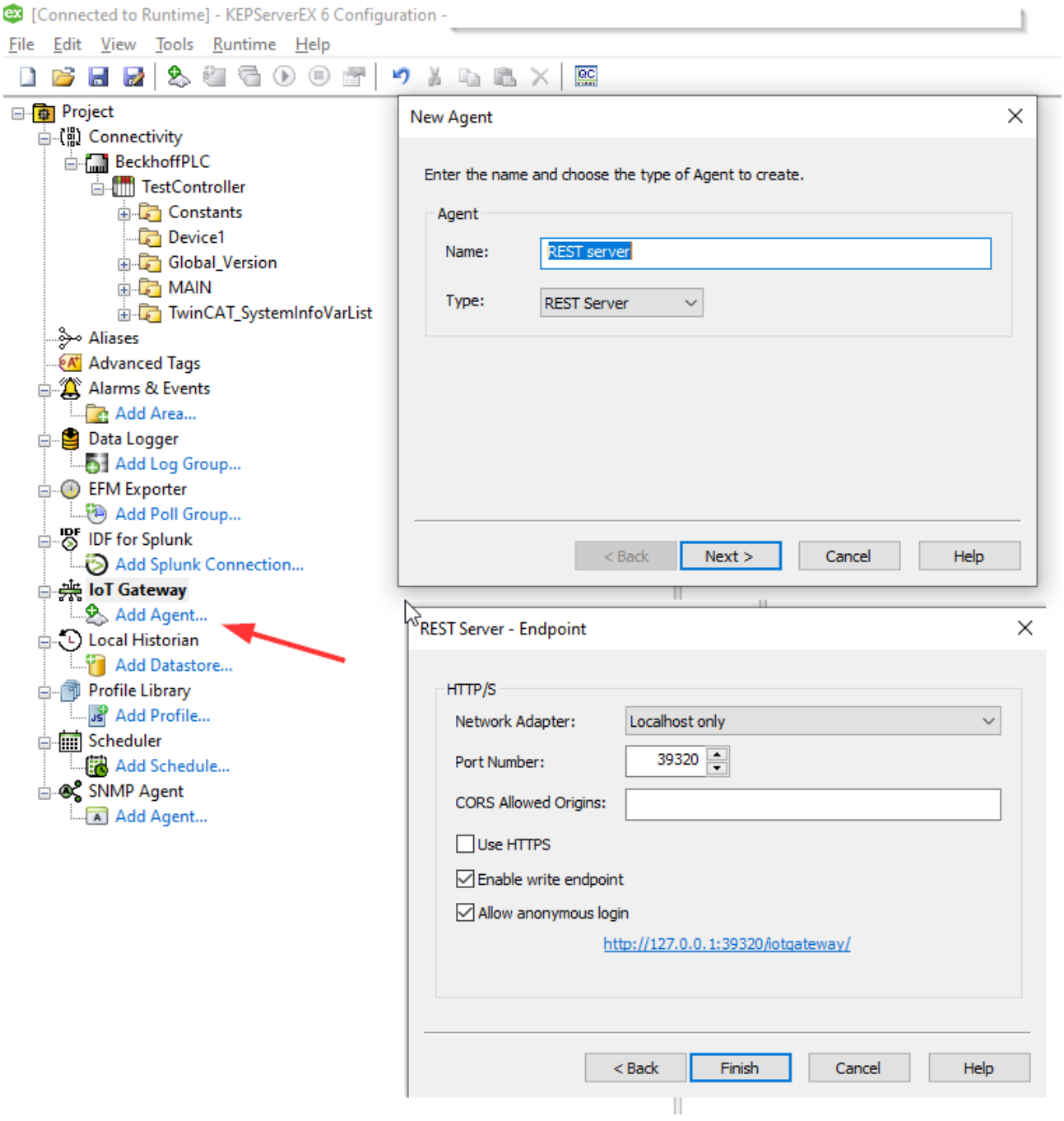


Figure 10 - Adding IoT gateway agent

3.4.6 IoT gateway plugin, add server tags

The agent generates a server item reference based on each defined tag, and just like any other client, it polls for data at the scan rate that was previously set. The scan speeds may be customised for each tag individually.

IoT Gateway system tags disclose certain status information from the IoT Gateway. These tags are updated every five seconds. By writing any value to them, they may be reset to zero. System tags count towards the total licenced amount of tags when setup as a tag under an IoT Gateway agent.

It is necessary to keep in mind that the price of the licence is determined by the number of tags the user is using; thus, it is essential to carefully choose the things that are broadcasted via the gateway. That will deliver the most significant advantages to the company's business operations.

After being setup, an agent is prepared for the addition of tags. The last step, which takes place after the REST server agent has been established, is to explicitly add any server tags that you want to make available via the web service. At this point in the project, the only requirements are for humidity, temperature, and write tags. (Figure 11).

When this is finished, the REST web service will be completely available, and we will be able to continue implementing the client application using Node.js.

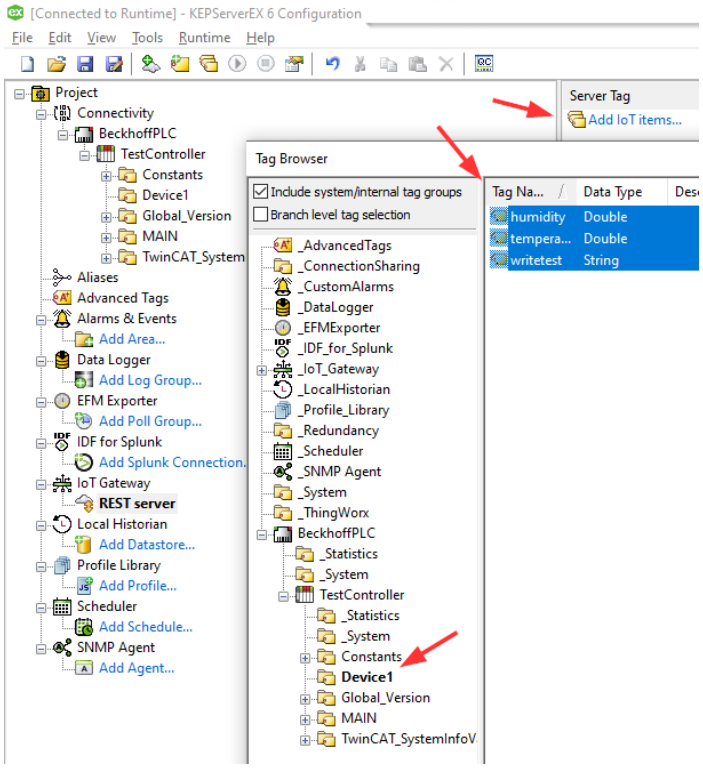


Figure 11 - Adding server tags to IoT gateway

3.4.7 Setting up Node.js application

Node.js is a technology that allows developers to construct JavaScript-based server applications that are both quick and scalable. A module that communicates with the database and another module that serves as a client for the HTTP service are going to be developed as part of this project. The database module will be responsible for storing tags in the database and retrieving them, while the HTTP module will be responsible for sending requests and processing responses in accordance with those requests.

One of the files that must be present for a Node.js application to function properly is called package.json. This file defines all of the scripts and dependencies that are necessary to execute the project. The contents of the package.json are detailed below.

```
{
  "name": "kepserverex-iot-gateway-rest-client",
  "version": "1.0.0",
  "description": "REST client for KEPServerEX IoT Gateway plug-in",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "nodemon index.js",
    "start-log-to-file": "nodemon index.js  > app.log 2>&1"
  },
  "author": "Ilia Reshetov",
  "license": "MIT",
  "devDependencies": {
    "nodemon": "^2.0.16"
  },
  "dependencies": {
    "axios": "^0.27.2",
    "dotenv": "^16.0.0",
    "node-fetch": "^3.2.4",
    "pg": "^8.7.3"
  }
}
```

To launch the application, type npm run start on the command line. It triggers script defined in package.json. In order to facilitate development, the dependency nodemon is used. It is the responsibility of this component to automatically deploy any modifications that were made while the application was being running, eliminating the need to restart the application each and every time.

Dotenv is another significant requirement that is used to assist one of the best practises for developing software, which requires the configuration to be kept strictly separate from the source code. Two configurations are required for this project. The connection string DATABASE_URI is a

query string that provides connection-specific parameters and enables database connection. Second setting is the KEPWARE_IOT_URL base URL, which is where the REST web service listens for incoming requests.

Axios and node-fetch, the last two dependencies, are used to make HTTP requests easier. Each library has more than 20 million weekly downloads on the node package management website.

All of the source code is provided in appendices 1 and 2. Following that, we will conduct an inspection of the various building parts of code. The application consists of two different parts. The initial component of this project is the development of the REST client in accordance with the kepware documentation. The second step involves developing a persistence layer by using PostgreSQL, which is a relational database.

3.4.8 Node.js REST persistence layer implementation

Application by design is heavily relying on database connectivity and therefore it is crucial to initialize schema before executing any other functions. On start-up of the application at first connection to database is established. Then schema, table and index must be created. Figure 12 shows entity relationship diagram for the table that was produced.

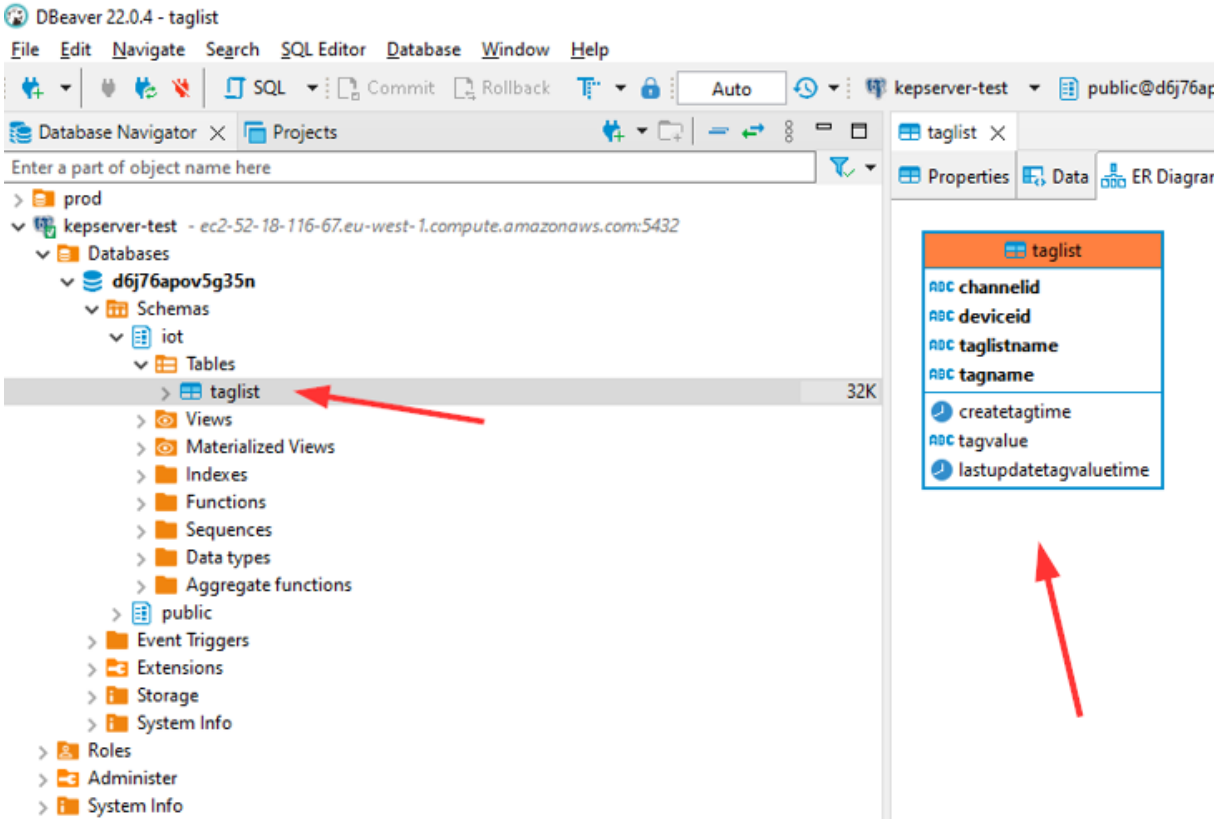


Figure 12 - PostgreSQL schema diagram

3.4.9 Node.js REST client implementation

IoT Gateway can handle three different types of HTTP requests. Every type of request requires its own function implementation inside the Node.js source code.

A Browse command returns a JSON array of all the tags that are set up for particular REST Server instance. These identifiers must be stored in a database for future usage. Browse command has to be executed only ones on application startup. The browse command can only be run once, and that is at the beginning of the program cycle. This will guarantee that the database always has the most recent version of the tag list. The following text demonstrate implantation of function called browse:

```
async function browse() {
  try {
    console.log(`INFO: Execute GET:${BROWSE_REST_REQUEST}`);
    const response = await fetch(BROWSE_REST_REQUEST);
    const tags = await response.json();
    tags.browseResults.forEach((tag) => {
      const tag_array = tag.id.split(".", 4);
      initializeTag(tag_array[0], tag_array[1], tag_array[2], tag_array[3]);
    });
    console.log("INFO: BROWSE has been successfully executed");
    console.log(tags);
  } catch (error) {
    console.log(error);
  }
}
```

Because Node.js makes use of a single-threaded event loop, an asynchronous function is required in order to send a GET request to REST API. After response has been received it must be parsed into object. When it is ready need to access array available by the property browseResults. While iterating over the array initializeTag function is called which responsible to save tag name to database. The tag name is divided into four sections, each of which is indented to enhance future accessibility by allowing the database to be queried for unique devices.

Since application does not provide graphical user interface console logs are used to ensure successful program execution. Following are the log messages copied from console during software test:

```
INFO: Execute GET:http://127.0.0.1:39320/iotgateway/browse
INFO: BROWSE has been successfully executed
{
  browseResults: [
    { id: 'BeckhoffPLC.TestController.Device1.humidity' },
    { id: 'BeckhoffPLC.TestController.Device1.temperature' },
    { id: 'BeckhoffPLC.TestController.Device1.writetest' }
```

```
  ],
  succeeded: true,
  reason: ''
}
INFO: Saved to DB PLC tag definition: BeckhoffPLC.TestController.Device1.humid-
ity
INFO: Saved to DB PLC tag definition: BeckhoffPLC.TestController.De-
vice1.writetest
INFO: Saved to DB PLC tag definition: BeckhoffPLC.TestController.Device1.temper-
ature
```

There are some requirements that need to be satisfied before one may write to tags, and those requirements must be met. Write operations are only accessible when the REST Server interface has been configured to permit them and after writeable tags have been applied. In order to carry out the test, one of the tags identified as writetest that is associated with Device1 was set up so that it could be written to. Client application is able to write to one or more tags that have been specified by using the write command. Next follows the implementation details of asynchronous write function:

```
async function write() {
  const param = [
    {
      id: "BeckhoffPLC.TestController.Device1.writetest",
      v: "testValue",
    },
  ];
  console.log(`INFO: Execute POST:${WRITE_REST_REQUEST}`, param);
  axios
    .post(WRITE_REST_REQUEST, param)
    .then(function (response) {
      if (response.status === 200) {
        console.log("INFO: WRITE has been successfully executed");
        console.log(response.data);
      }
    })
    .catch(function (error) {
      console.log(error);
    });
}
```

The HTTP POST method is used in order to complete the write tag operation. It is necessary to provide the update tag id and the value that is going to be written to that tag inside the body of the POST method in order to carry out an update. In this particular case parameters object that contains id of tag and value that is going to be written since this operation is controlled by business logic,

which is beyond the scope of this project. Nonetheless, it is sufficient to test the functionality of the web service. After successful program execution the following log messages were produced:

```
INFO: Execute POST:http://127.0.0.1:39320/iotgateway/write [
  {
    id: 'BeckhoffPLC.TestController.Device1.writetest',
    v: 'testValue'
  }
]
INFO: WRITE has been successfully executed
{
  writeResults: [
    {
      id: 'BeckhoffPLC.TestController.Device1.writetest',
      s: true,
      r: ''
    }
  ]
}
```

The URL and body of the request are presented first to provide the user a clear picture of how the request is constructed. Second message notifies regarding successful execution where writeResults object has property called s and when that is set to true it means that value was written to PLC.

During the last phase of the project the read functionality was implemented. A JSON array containing the specified tag or tags is what is returned when the Read post command is executed. It utilizes HTTP GET and tag ids must be embedded within the URL command. In previous section when browse method was tested tag ids that exposed by the PLC through web service was saved the database. Details of asynchronous function implementation are shown below:

```
async function read() {
  try {
    if (!READ_REST_REQUEST) {
      const tagList = await getTagList();
      READ_REST_REQUEST = `${IOT_GATEWAY_BASE_URL}read?`;
      tagList.forEach((tag) => {
        let readRequest = `ids=${tag.channelid}.${tag.deviceid}.${tag.ta-
glistname}.${tag.tagname}&`;
        READ_REST_REQUEST = READ_REST_REQUEST.concat(readRequest);
      });
    }
    console.log(`INFO: Execute GET:${READ_REST_REQUEST}`);
    const response = await fetch(READ_REST_REQUEST);

    const tagsValues = await response.json();
    console.log("INFO: READ has been successfully executed");
```

```
    console.log(tagsValues);

    tagsValues.readResults.forEach((tag) => {
      const tag_array = tag.id.split(".", 4);
      saveTagValue(
        tag_array[0],
        tag_array[1],
        tag_array[2],
        tag_array[3],
        tag.v
      );
    });
  } catch (error) {
    console.log(error);
  }
}
```

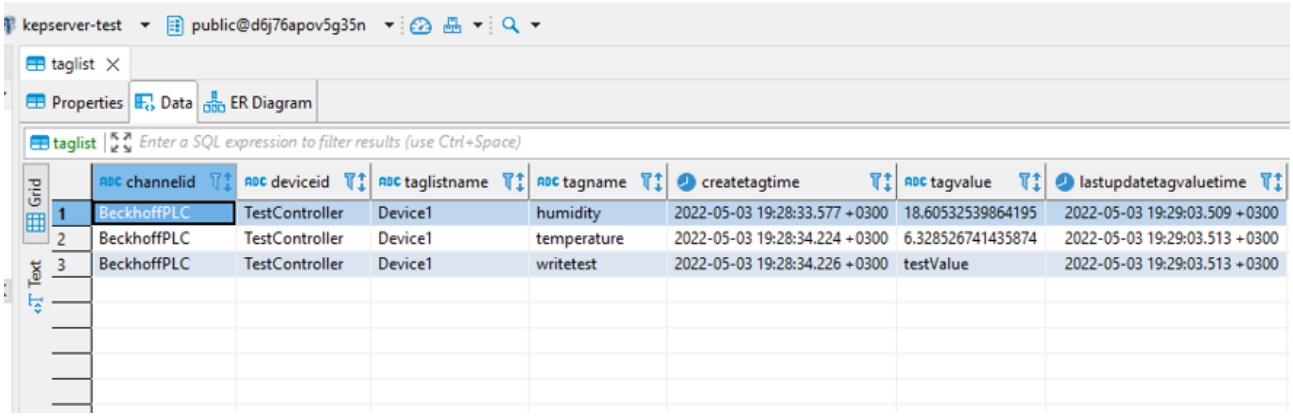
First list of tags names must be fetched from database. Fetch will return list of names in form of array while iterating over that array URL request gets contracted. This operation only happens once during launch of the application then the same request URL is used. Read method is carried out as a polled action at regular intervals to ensure that tag values are always maintained current in the database, which is important since business logic may rely on them.

```
INFO: Execute GET:http://127.0.0.1:39320/iotgateway/read?ids=Beckhoff-
PLC.TestController.Device1.humidity&ids=BeckhoffPLC.TestController.De-
vice1.writetest&ids=BeckhoffPLC.TestController.Device1.temperature&
INFO: READ has been successfully executed
{
  readResults: [
    {
      id: 'BeckhoffPLC.TestController.Device1.humidity',
      s: true,
      r: '',
      v: 87.34721904325933,
      t: 1652607018205
    },
    {
      id: 'BeckhoffPLC.TestController.Device1.writetest',
      s: true,
      r: '',
      v: 'testValue',
      t: 1652606907258
    },
    {
      id: 'BeckhoffPLC.TestController.Device1.temperature',
      s: true,
      r: '',
```

```
      v: 7.522240024414435,
      t: 1652607018205
    }
  ]
}
INFO: Saved to DB tag value: BeckhoffPLC.TestController.Device1.writetest,
value: testValue
INFO: Saved to DB tag value: BeckhoffPLC.TestController.Device1.humidity, value:
87.34721904325933
INFO: Saved to DB tag value: BeckhoffPLC.TestController.Device1.temperature,
value: 7.522240024414435
```

In the beginning of a cycle URL string was build that contains all tag names that available for the test. When it was ready HTTP GET method was called and upon response from web service values were persisted to database. The tag values are refreshed every ten seconds, and this frequency may be changed to better meet the requirements of a true production scenario.

The PostgreSQL database instance that is being utilised for the project is being hosted on the Amazon AWS (Amazon Web Services) cloud. Figure 13 shows that test was contacted successfully and tag values were updated in database with expected time interval.



	channelid	deviceid	taglistname	tagname	createtagtime	tagvalue	lastupdatetagvaluetime
1	BeckhoffPLC	TestController	Device1	humidity	2022-05-03 19:28:33.577 +0300	18.60532539864195	2022-05-03 19:29:03.509 +0300
2	BeckhoffPLC	TestController	Device1	temperature	2022-05-03 19:28:34.224 +0300	6.328526741435874	2022-05-03 19:29:03.513 +0300
3	BeckhoffPLC	TestController	Device1	writetest	2022-05-03 19:28:34.226 +0300	testValue	2022-05-03 19:29:03.513 +0300

Figure 13 - Taglist table data

4 Conclusion

This project aimed to achieve a number of objectives that were specified by the commissioning party. The main goal was to evaluate if migration to use KEPServerEX as the primary connectivity tool would be sufficient and would integrate into the current company's software infrastructure with a reasonable number of changes. Another goal was to design and implement test framework where communication setup could be tested.

In order to achieve these goals, step the author first had to acquire significant knowledge about legacy software architecture, which is used to facilitate communication between PLC and control systems on the factory floor.

After the author has gathered a sufficient amount of prior information, the process of designing the test framework started. It was chosen to employ the Beckhoff TwinCAT software system in order to bring the test setting as closely as possible to the actual production circumstances. Beckhoff is software that enables to turn any PC into PLC therefore the whole test framework was possible to build on a single laptop. Both Node.js and PostgreSQL were selected since the author was already quite acquainted with both of these technologies, which allowed for the completion of the project much more quickly.

Both aims were achieved during the course of the project. With the implemented test framework, it was proven that KEPServerEX is possible to integrate with a small amount of adjustments. The KEPServerEX IoT Gateway plug-in communicates via web service, providing great flexibility and platform independence, whereas KEPServerEX abstracts away PLC connectivity.

Migrating to KEPServerEX would minimize the time that required to setup communication for every new project and remove the burden to implement test and maintain PLC connection drivers. In the long run, it all brings great business benefits to the company because developers could spend more time working on the project development rather than spending a lot of time on repetitive and time-consuming tasks.

The author of this thesis was able to build and construct a test framework for the integration of KEPServerEX into current infrastructure. As a result of this work, it is possible to draw the conclusion that this specific commercial connectivity platform is sufficient enough to replace existing software. It not only covers existing functionality, but also adds dozens of new features that were previously unavailable in legacy software, such as a graphical user interface, over 150 PLC connection drivers out of the box, and comprehensive diagnostic tools.

It was found that by using this specific third-party solution, the business would be able to abstract away from connection to PLC and instead focus more on the development and improvement of the business logic. As a result of this migration, the firm will have a reduced requirement to assign engineers to maintain connection applications, which will, in the long run, result in cost savings.

4.1 Development proposal

Nowadays, industrial automation is not just about making things move in a consistent manner on the factory floor. The gathering of data for IoT hubs and the subsequent analysis of that data might provide significant revenues for the businesses that are the first to recognize this potential and introduce relevant services. The KEPServerEX IoT Gateway enables a variety of methods such as REST or MQTT for establishing data flow from connected devices into the cloud infrastructure. In the future, study might be conducted to determine which technique of connecting is most effective and how the data collected could be used.

4.2 Thesis process evaluation and personal learning

Due to the fact that PLC was introduced way before personal computers and the public internet, a significant number of proprietary communication protocols have been developed. Even though in recent years there have been successful attempts to develop standard protocols, a lot of legacy protocols are still in existence and require modern integration solutions. Previously, the author worked on the development of higher layer control systems and was only briefly familiar with the connection to the lower systems and only some implementation details. So it was quite beneficial to broaden the knowledge with regards to PLC to PC communication.

References

"Cimcorp - Wikipedia" (2021). en.wikipedia.org. Available at: https://en.wikipedia.org/wiki/Cimcorp (Accessed: May 4, 2022).

Cimcorp WCS for complete control - Cimcorp (2022). Available at: https://cimcorp.com/general/cimcorp-wcs-for-complete-control/ (Accessed: May 7, 2022).

D'Antonio, G., Bedolla, J.S. and Chiabert, P. (2017) "A Novel Methodology to Integrate Manufacturing Execution Systems with the Lean Manufacturing Approach," *Procedia Manufacturing*, 11, pp. 2243–2251. doi:10.1016/j.promfg.2017.07.372.

Herron, D. (2020) *Node.js Web Development - Fifth Edition*. Edited by D. Herron. Packt Publishing, Limited. Available at: http://sfx.finna.fi/nelli21?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=4100000011375844&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

INFODREAM INC. (2022) *The functions of the MES - Functional hedging in accordance with ISA95*. Available at: https://infodreamgroup.com/what-are-the-mes-functions/ (Accessed: May 4, 2022).

KEPServerEX Connectivity Platform | OPC Server | Kepware (2022). Available at: https://www.kepware.com/en-us/products/kepserverex/ (Accessed: May 7, 2022).

Liptak, B.G. (2018a) *Instrument Engineers' Handbook, Volume Two*. CRC Press.

Liptak, B.G. (2018b) *Instrument Engineers' Handbook, Volume Two*. CRC Press.

Meyer, H., Fuchs, F. and Thiel, K. (2009) *Manufacturing Execution Systems (MES): Optimal Design, Planning, and Deployment*.

Son, D.W., Chang, Y.S. and Kim, W.R. (2015) "Design of Warehouse Control System for Real Time Management," *IFAC-PapersOnLine*, 48(3), pp. 1434–1438. doi:10.1016/J.IFACOL.2015.06.288.

Stream Industrial Data with the IoT Gateway | Kepware (2022). Available at: https://www.kepware.com/en-us/products/kepserverex/advanced-plug-ins/iot-gateway/ (Accessed: May 14, 2022).

Support Center | Customer Resources | Kepware (2022). Available at: https://www.kepware.com/en-us/support/knowledge-base/2015/what-is-the-opc-quick-client/ (Accessed: May 15, 2022).

TwinCAT | Automation software | Beckhoff Worldwide (2022). Available at: https://infosys.beckhoff.com/ (Accessed: May 7, 2022).

Walters, E.G. and Bryla, E.J. (2016) "Software architecture and framework for Programmable Logic Controllers: A case study and suggestions for research," *Machines*, 4(2). doi:10.3390/machines4020013.

Worsley, J.C. (2002) *Practical PostgreSQL*. Edited by J.C. Worsley and J.D. Drake. O'Reilly Media, Inc. Available at: http://sfx.finna.fi/nelli21?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=2670000000099553&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

"Cimcorp - Wikipedia" (2021). en.wikipedia.org. Available at: https://en.wikipedia.org/wiki/Cimcorp (Accessed: May 4, 2022).

Cimcorp WCS for complete control - Cimcorp (2022). Available at: https://cimcorp.com/general/cimcorp-wcs-for-complete-control/ (Accessed: May 7, 2022).

D'Antonio, G., Bedolla, J.S. and Chiabert, P. (2017) "A Novel Methodology to Integrate Manufacturing Execution Systems with the Lean Manufacturing Approach," *Procedia Manufacturing*, 11, pp. 2243–2251. doi:10.1016/j.promfg.2017.07.372.

Herron, D. (2020) *Node.js Web Development - Fifth Edition*. Edited by D. Herron. Packt Publishing, Limited. Available at: http://sfx.finna.fi/nelli21?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=4100000011375844&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

INFODREAM INC. (2022) *The functions of the MES - Functional hedging in accordance with ISA95*. Available at: https://infodreamgroup.com/what-are-the-mes-functions/ (Accessed: May 4, 2022).

KEPServerEX Connectivity Platform | OPC Server | Kepware (2022). Available at: https://www.kepware.com/en-us/products/kepserverex/ (Accessed: May 7, 2022).

Liptak, B.G. (2018a) *Instrument Engineers' Handbook, Volume Two*. CRC Press.

Liptak, B.G. (2018b) *Instrument Engineers' Handbook, Volume Two*. CRC Press.

Meyer, H., Fuchs, F. and Thiel, K. (2009) *Manufacturing Execution Systems (MES): Optimal Design, Planning, and Deployment*.

Son, D.W., Chang, Y.S. and Kim, W.R. (2015) "Design of Warehouse Control System for Real Time Management," *IFAC-PapersOnLine*, 48(3), pp. 1434–1438. doi:10.1016/J.IFACOL.2015.06.288.

Stream Industrial Data with the IoT Gateway | Kepware (2022). Available at: https://www.kepware.com/en-us/products/kepserverex/advanced-plug-ins/iot-gateway/ (Accessed: May 14, 2022).

Support Center | Customer Resources | Kepware (2022). Available at: https://www.kepware.com/en-us/support/knowledge-base/2015/what-is-the-opc-quick-client/ (Accessed: May 15, 2022).

TwinCAT | Automation software | Beckhoff Worldwide (2022). Available at: https://infosys.beckhoff.com/ (Accessed: May 7, 2022).

Walters, E.G. and Bryla, E.J. (2016) "Software architecture and framework for Programmable Logic Controllers: A case study and suggestions for research," *Machines*, 4(2). doi:10.3390/machines4020013.

Worsley, J.C. (2002) *Practical PostgreSQL*. Edited by J.C. Worsley and J.D. Drake. O'Reilly Media, Inc. Available at: http://sfx.finna.fi/nelli21?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=2670000000099553&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

Appendices

Appendix 1. Source code, index.js

```
import fetch from "node-fetch";
import axios from "axios";
import "dotenv/config";
import {
  initializeTag,
  getTagList,
  saveTagValue,
  initializeDataBase,
} from "./db.js";

const IOT_GATEWAY_BASE_URL = `${process.env.KEPWARE_IOT_URL}/iotgateway/`;
const BROWSE_REST_REQUEST = `${IOT_GATEWAY_BASE_URL}browse`;
const WRITE_REST_REQUEST = `${IOT_GATEWAY_BASE_URL}write`;
let READ_REST_REQUEST;

async function browse() {
  try {
    console.log(`INFO: Execute GET:${BROWSE_REST_REQUEST}`);
    const response = await fetch(BROWSE_REST_REQUEST);
    const tags = await response.json();
    tags.browseResults.forEach((tag) => {
      const tag_array = tag.id.split(".", 4);
      initializeTag(tag_array[0], tag_array[1], tag_array[2], tag_array[3]);
    });
    console.log("INFO: BROWSE has been successfully executed");
    console.log(tags);
  } catch (error) {
    console.log(error);
  }
}

async function write() {
  const param = [
    {
      id: "BeckhoffPLC.TestController.Device1.writetest",
      v: "testValue",
    },
  ];
  console.log(`INFO: Execute POST:${WRITE_REST_REQUEST}`, param);
  axios
    .post(WRITE_REST_REQUEST, param)
    .then(function (response) {
      if (response.status === 200) {
        console.log("INFO: WRITE has been successfully executed");
```

```
        console.log(response.data);
      }
    })
    .catch(function (error) {
      console.log(error);
    });
}

async function read() {
  try {
    if (!READ_REST_REQUEST) {
      const tagList = await getTagList();
      READ_REST_REQUEST = `${IOT_GATEWAY_BASE_URL}read?`;
      tagList.forEach((tag) => {
        let readRequest = `ids=${tag.channelid}.${tag.deviceid}.${tag.ta-
glistname}.${tag.tagname}&`;
        READ_REST_REQUEST = READ_REST_REQUEST.concat(readRequest);
      });
    }
    console.log(`INFO: Execute GET:${READ_REST_REQUEST}`);
    const response = await fetch(READ_REST_REQUEST);

    const tagsValues = await response.json();
    console.log("INFO: READ has been successfully executed");
    console.log(tagsValues);

    tagsValues.readResults.forEach((tag) => {
      const tag_array = tag.id.split(".", 4);
      saveTagValue(
        tag_array[0],
        tag_array[1],
        tag_array[2],
        tag_array[3],
        tag.v
      );
    });
  } catch (error) {
    console.log(error);
  }
}

function main() {
  console.log("INFO: Start main");
  browse();
  write();
  setInterval(read, 10000);
}
```

```
// START
console.log(`Base URL:${IOT_GATEWAY_BASE_URL}`);
initializeDataBase().then(() => main());
```

Appendix 2. Source code, db.js

```
import "dotenv/config";
import pg from "pg";

const pool = new pg.Pool({
  connectionString: process.env.DATABASE_URI,
  ssl: {
    rejectUnauthorized: false,
  },
});

pool.on("error", (err, client) => {
  console.error("Unexpected error on idle client", err);
  process.exit(-1);
});

const initializeDataBase = async () => {
  try {
    await pool.query("CREATE SCHEMA IF NOT EXISTS iot");
    await pool.query(
      "CREATE TABLE IF NOT EXISTS iot.taglist(channelId text NOT NULL, deviceId
text NOT NULL, tagListName text NOT NULL, tagName text NOT NULL, createTagTime
timestamptz NOT NULL, tagValue text, lastupdatetagValuetime timestamptz)"
    );
    await pool.query(
      "CREATE UNIQUE INDEX IF NOT EXISTS taglist_channelId_idx ON iot.taglist USING
btree (channelId, deviceId, tagListName, tagName)"
    );
    console.log("INFO: successfully initialized database");
  } catch (error) {
    console.log(error);
  }
};

const initializeTag = async (channelId, deviceId, tagListName, tagName) => {
  try {
    const response = await pool.query(
      "SELECT * FROM  iot.taglist WHERE channelid = $1 and deviceid = $2 and ta-
glistname = $3 and tagname = $4",
      [channelId, deviceId, tagListName, tagName]
    );

    if (!response.rows[0]) {
```

```
      const response = await pool.query(
        "INSERT INTO iot.taglist(channelid, deviceid, taglistname, tagname, cre-
atetagtime) VALUES ($1, $2, $3, $4, NOW()) RETURNING *",
        [channelId, deviceId, tagListName, tagName]
      );

      console.log(
        `INFO: Saved to DB PLC tag definition: ${response.rows[0].channelid}.${re-
sponse.rows[0].deviceid}.${response.rows[0].taglistname}.${re-
sponse.rows[0].tagname}`
      );
    } else {
      console.log(
        `WARNING: Tag already exist: ${response.rows[0].channelid}.${re-
sponse.rows[0].deviceid}.${response.rows[0].taglistname}.${re-
sponse.rows[0].tagname}  initialize tag time:${response.rows[0].createtagtime}`
      );
    }
  } catch (error) {
    console.log(error);
  }
};

const saveTagValue = async (
  channelId,
  deviceId,
  tagListName,
  tagName,
  tagValue
) => {
  const response = await pool.query(
    "UPDATE iot.taglist SET tagvalue = $5 , lastupdatetagvaluetime = NOW() WHERE
channelid = $1 and deviceid = $2 and taglistname = $3 and tagname = $4 RETURNING
*",
    [channelId, deviceId, tagListName, tagName, tagValue]
  );
  console.log(
    `INFO: Saved to DB tag value: ${response.rows[0].channelid}.${re-
sponse.rows[0].deviceid}.${response.rows[0].taglistname}.${re-
sponse.rows[0].tagname}, value: ${response.rows[0].tagvalue}`
  );
};

const getTagList = async () => {
  const response = await pool.query("SELECT * FROM  iot.taglist ", []);

  if (!response.rows[0]) {
    throw "ERROR: Tags were not initialized, call browse method first";
```

```
  } else {
    return response.rows;
  }
};

export { initializeTag, getTagList, saveTagValue, initializeDataBase };
```