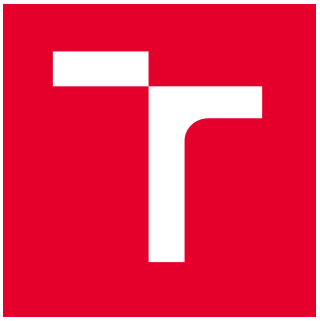
BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

BACHELOR'S THESIS

Brno, 2025 Karim Ayyad



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

HMI DEMONSTRATIONAL APPLICATION BASED ON ANDROID DEVICE

HMI DEMONSTRATIONAL APPLICATION BASED ON ANDROID DEVICE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — Karim Ayyad
AUTOR PRÁCE

SUPERVISOR — doc. Ing. Jakub Arm, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Bachelor's Thesis

Bachelor's study program **Electrical Engineering**
specialization Power Systems and Automation
Department of Control and Instrumentation

Student: Karim Ayyad ***ID:*** 247332

Year of study: 3 ***Academic year:*** 2024/25

TITLE OF THESIS:

HMI demonstrational application based on Android device

INSTRUCTION:

The task is to design a demonstration PLC/HMI control system for a selected industrial machine/line, which will be based on the use of an Android OS capable device. Partial tasks are ensuring the timeliness of communication between the HMI device and the PLC, implementing basic security techniques, and showing the advantages of deploying an Android application for interaction with the operator.

1. Research of relevant technologies and libraries.
2. Design a demonstration application.
3. Design an Android-based HMI subsystem.
4. Implement SW for PLC and HMI (Android Application).
5. Evaluate the achieved properties of the system.

RECOMMENDED LITERATURE:

[1] S. TANELLA. Android OPCUA Client. 2019. [Online]. Available: https://github.com/SimoneTinella/Android_OPCUA_Client.

[2] J. HURNÍK. HMI aplikace v prostředí Android komunikující pomocí OPC UA. Vysoké učení technické v Brně, Brno, 2019.

Date of project specification: 10.2.2025 ***Deadline for submission:*** 28.5.2025

Supervisor: doc. Ing. Jakub Arm, Ph.D.

doc. Ing. Miloslav Steinbauer, Ph.D.
Chair of study program board

Abstract

The thesis aims to find a solution for integrating Android OS into industrial automation by managing a system of PLC and HMI with an android mobile application. Demonstration and illustration of the project will be described in the theoretical part. For the practical part of thesis, a demonstrational system will be implemented and tested for the PLC and linked with the application. A copy of the source code for the whole project will be supplemented in the appendices.

Keywords

TwinCAT 3, Android, JAVA, OPC UA, PLC, HMI Application.

Bibliographic citation

AYYAD, Karim. HMI demonstrational application based on Android device. Online, bachelor's Thesis. Jakub ARM (supervisor). Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, 2025. Available at: https://www.vut.cz/en/students/final-thesis/detail/169683.

Author’s Declaration

Author:	Karim Ayyad
Author’s ID:	247332
Paper type:	Bachelor’s thesis
Academic year:	2024/25
Topic:	HMI demonstrational application based on Android device

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the project and listed in the comprehensive bibliography at the end of the project.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone’s personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation S 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno, ----------

author’s signature

Acknowledgement

I would like to thank the advisor of my thesis doc. Ing. Jakub Arm Ph.D. for his valuable comments. I am also very grateful to my family for their constant encouragement and support.

Brno, ----------

Author’s signature

Contents

FIGURES

INTRODUCTION

The main task of this bachelor's thesis is to design and implement a control system, using Programmable-Logic Controller (PLC) and Human Machine Interface (HMI). The thesis will focus on creating an Android application to work, remotely, as a real-time interface between the operator and the system, ensuring the timeliness of the communication between the PLC and the HMI, implementing basic security technologies and demonstrating the prospective advantages of integrating an Android interface to the system.

The thesis consists of two main parts. The first part is about the structure of the whole project, including the developed PLC and HMI system and the Android application among other modules used to establish the communication, a theoretical introduction for the challenges, which my thesis will supposedly solve, and followed by research and exploration of the relevant technologies and libraries to achieve that and the foundational knowledge needed to integrate Android systems into the industry.

I will later follow the practical part, demonstrating each technology used in the solution and the reasoning for choosing them. For example, the programming language that I'll use to develop the Android application, which is JAVA, or the IDEs. In addition, focus will be mainly spotted on the biggest challenge for the solution, that is remote communication, as the intention was to make it secure, stable and real-time control. The final choice was using tunnelling as the need to publish the OPC UA in public aroused while developing the solution. Based on inspiration from an open-source project "An Android OPC UA Client based on official OPC Foundation Java Stack. [1] ", I chose OPC UA communication to be implemented in the Android application with a suitable open-source library and OPC UA server configuration on the PLC.

In the appendices, you can find a version of the source code for both the PLC project and the Android project.

1. PROJECT'S DESIGN

1.1 Aims

First and foremost, I had set the objectives and goals of the thesis, which may also evolve throughout the process of developing. These objectives are defined based on the assignment of the bachelor's thesis, but with more details and specifications.

- Creating a demonstration project with TwinCAT3 and configuring the PLC.
- Designing the program for the PLC.
- Designing and configuring the HMI screen.
- Creating a project in Android Studio and designing it according to the architecture.
- Implementing communication using OPC UA.
- Publishing the OPC UA server securely for public access, in order to communicate with the Android application.
- Designing and implementing the user interface of the Android application.
- Testing the functionality of individual parts of the projects, including the PLC program, the HMI screens, the communication and last the application.

1.2 Project requirements

By the end of the thesis, I expect the project to meet the following functional and non-functional requirements, which are based on the aims of the project:

- Functional requirements:
  - Minimized latency for reading/writing processes.
  - Secured publication of the OPC UA server.
  - Accurate control for the PLC logic, HMI interface and Android application.
- Non-functional requirements:
  - Reliability under heavy duty performance.
  - User friendly and interactive user interface (UI).
  - Compatibility with most Android systems.
  - A larger margin for further development in the future.

1.3 Proposed solution

In this section, I'll outline my plan to implement the solution to meet the requirements and the assignment. The assignment species that I need to implement the HMI interface to be Android-based. Typically, the PLC's web server (in this case, it would be OPC UA server) or the HMI server can be accessed through web browser on devices connected on the same local network (Figure 1-1).

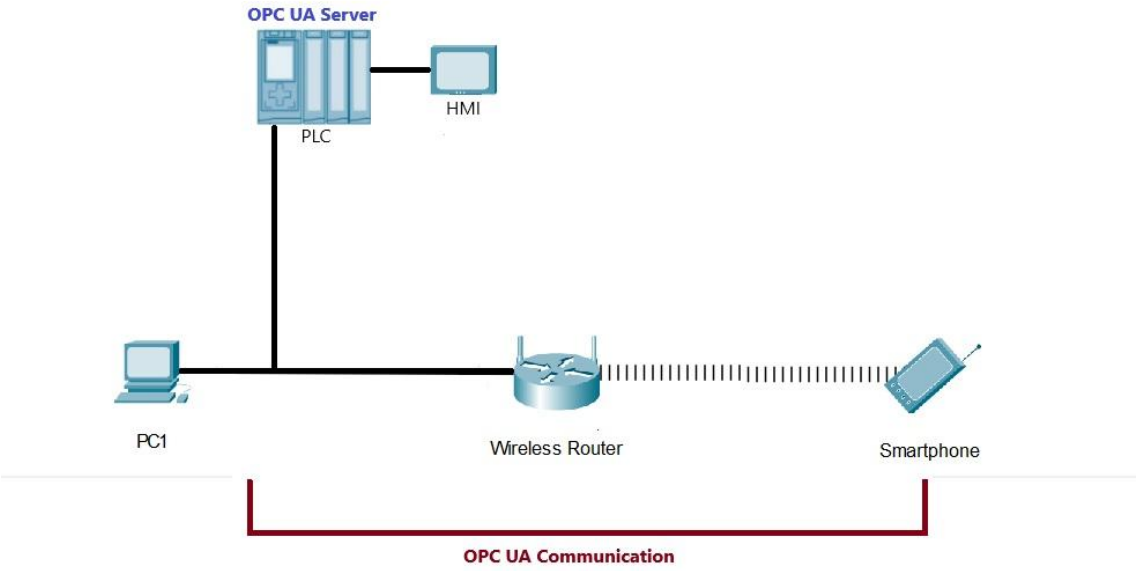


Figure 1-1 An example of ordinary communication over the same network. [2]

However, the challenge was to enable the communication even remotely via a designed application, and without requiring a direct connection to the PLC or HMI local network. Thus, a secure and real-time solution for communicating remotely in public was necessary to be found. One approach I considered was to use an external server to establish communication between the PLC and/or HMI and the mobile application, provided it fulfils the requirements. Due to limitations in resources, I wanted to look for a free open-source alternative.

The final proposed solution involves utilizing tunnelling protocol, in order to make the OPC UA server from the PLC to the web accessible over the internet from the mobile application, which has internet access throughout the communication process (Figure 1-2). In this way, we'll be able to access the PLC and HMI externally from any location, provided we pass the correct tunnelled URL to the application.

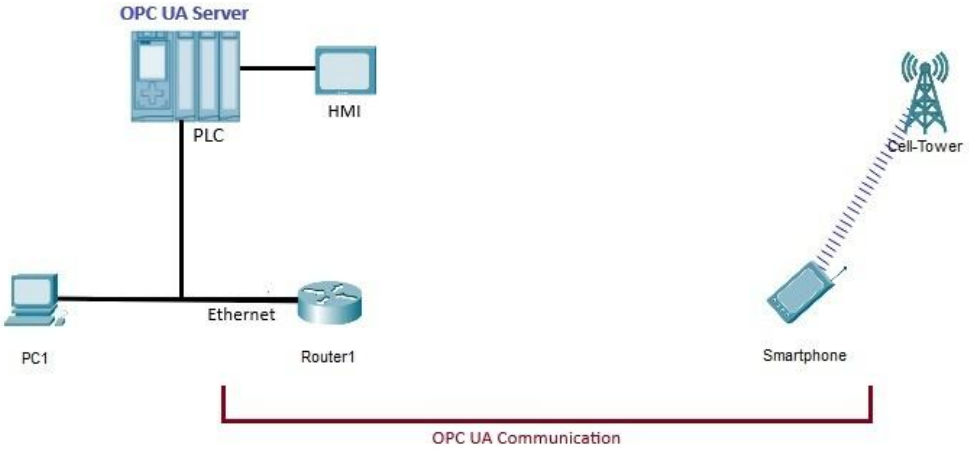


Figure 1-2 Topology of the proposed solution to communicate remotely

2. USED TECHNOLOGIES

In this section, I will go through all the technologies used in my solution and provide an overview for each of them and the reasons for choosing them.

2.1 PLC & HMI

A Programmable Logic Controller (PLC) is a modern and powerful digital controller, which is widely used in industrial automation systems and takes input data, processes it according to logic and generates the necessary output to perform the system's functionality. A typical PLC consists of a power supply, a processor with integrated memory and communication modules in addition to I/O modules [3]. Some PLCs are designed compact as one unit, incorporating all these components into a single unit. Figure 2-1 shows an illustration for a basic PLC system and how its process and communication flow.

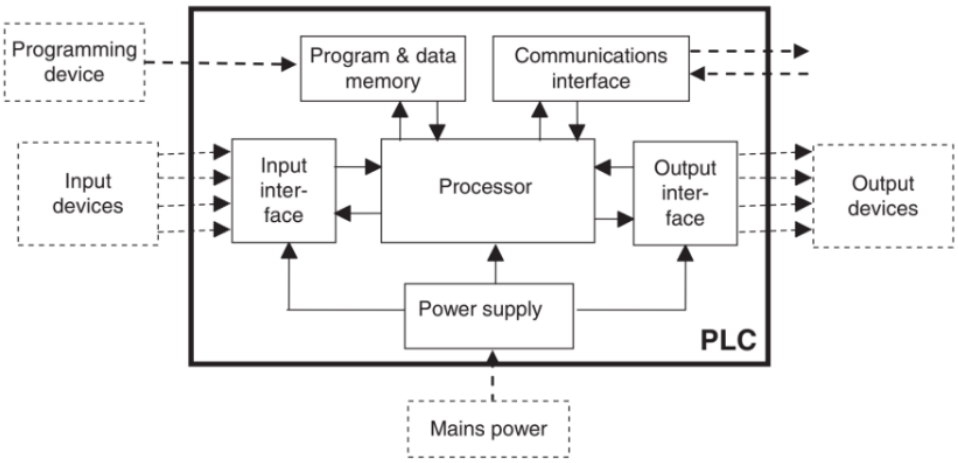


Figure 2-1 PLC System. [3]

Beckhoff controller (Embedded PC CX family) was selected for the solution. It offers PC-based automation with high reliability, a compact design and straightforward programming using TwinCAT3 software with a significant variety of communication interfaces [4].

2.1.1 TwinCAT3 IDE

The Windows Control and Automation Technology (TwinCAT3) is an integrated development environment (IDE) from Beckhoff, which is built on Visual Studio shell. It's used mostly for configuring and programming the PLC and its relevant HMI in multiple languages, such as Ladder (LD), Structured Text (ST) or Function Block

Diagram (FBD), In addition, it supports C/C++, MATLAB and Simulink [5].

One of TwinCAT3's strengths is the real-time-capable runtime, which is critical for simulation and testing purposes. Additionally, it offers various functions, such as TwinCAT HMI and TwinCAT OPC UA, which can be activated and licensed through the Package Manager.

2.1.2 HMI

Human-Machine Interface (HMI) refers to an interface between the operator and the control system, where communication with the controller is essential, providing a graphical visualization of the system, including its real-time data, control commands and feedback, such as alarms.

HMIs have been constantly developed over the years, until it became easily configurable using integrated softwares, e.g., TwinCAT HMI, and able to communicate with the PLC using different communication interfaces, for example OPC UA, securely with minimum latency. Furthermore, they can be built based on current web technologies, such as HTML5 and JavaScript, and integrated with other systems, including Android-based platforms.

2.2 Android OS

Android is an open-source operating system that was first released by Google in 2008 and is based on Linux kernel and developed primarily for mobile devices, such as smartphones, tablets, smart glasses, smartwatches or other personal mobile computing devices. It's considered one of the most widely used mobile operating systems globally with 73.51% [9]. Android is written in many languages, namely Java, Kotlin, C++, Rust and others. The architecture of Android consists of a Linux kernel, hardware abstraction layers, a runtime environment, application frameworks and a large set of supported libraries for different functionalities. [6]

Due to its variety and widespread adoption, Android OS was selected for this project as it's suitable for creating various applications for industrial systems. It also provides plenty of robust and secure APIs for communication, UI design, and hardware integration, making it one of the easiest developable operating systems in the world.

2.2.1 JAVA

Java is a high-level, class-based, general-purpose and object-oriented programming language that was first created by Sun Microsystems in 1995 and later acquired by Oracle. It's well-known for its platform independence, through its Java Virtual Machine (JVM), which makes it commonly used as the primary development language for many applications, including Android.

To develop a Java-based application, a Java Development Kit (JDK) needs to be installed, which includes a set of tools and libraries that facilitate the development

process. The Java programming language also shares a lot of similarities with C and C++ in syntax, but with having a different structure, especially in the storage management by avoiding explicit deallocation (as in C's *free* or C++'s *delete*). This provides higher performance garbage-collected implementations, reducing the risk of memory leaks [7].

In addition to Java's robust security framework, which is critical in our application, responsive user-interfaces can be also easily created with the aid of Android's built-in widgets and layouts. Furthermore, networking configuration using Java's HTTP and WebSocket libraries establishes seamless communication between the application and external devices or servers and this is the core for IoT and automation projects.

2.2.2 Android Studio IDE

Android Studio is the official integrated development environment (IDE) particularly for Android applications. It was created by Google in collaboration with JetBrains, as it's built on IntelliJ IDEA software, which eases the integration and use of different extensions and tools that offer smoothness of the development and release process.

Android Studio was selected as the IDE for the implementation project due to its solid features [8], such as:

- The Code editor, which supports code generation, refactoring and debugging.
- An emulator that enables testing the applications based on the Android version or the device configurations.
- A layout editor, which is used for customizing and previewing the user interface.
- Build system using Gradle plugin to manage dependencies, automate build and generate APKs (Android Package) for deployment.

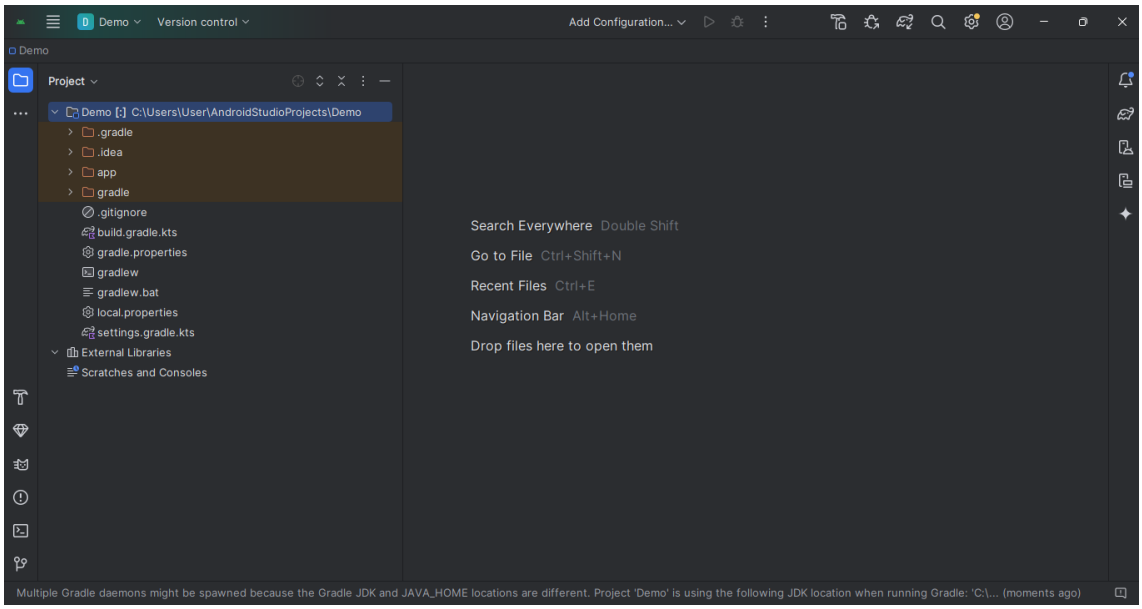


Figure 2-2 Android Studio

2.3 OPC UA

OPC Unified Architecture (OPC UA) is an industrial, cross-platform communication protocol for field devices and PLCs to communicate with cloud applications, developed by OPC Foundation and released in 2008. OPC UA supports client-server (see Figure 2-3) and publish-subscribe communication models, leading to efficient data transfer in IoT applications.

OPC UA has many defining features, such as its robust security by utilizing strict extensive controls, e.g., authentication through X509 certificates, session encryptions for different encryption levels, and user access control to validate the login credentials and certificate to make sure of providing the appropriate permissions [10]. Besides its security, it's communication-protocol independent, which means it can be mapped to other protocols such as TCP/IP, UDP/IP, MQTT or WebSocket. This will later help in publishing the OPC UA server on the web and access it from the Android application.

OPC UA has been selected for this project for all its features that offer seamless and secure communication between the PLC and the Android-based HMI application.

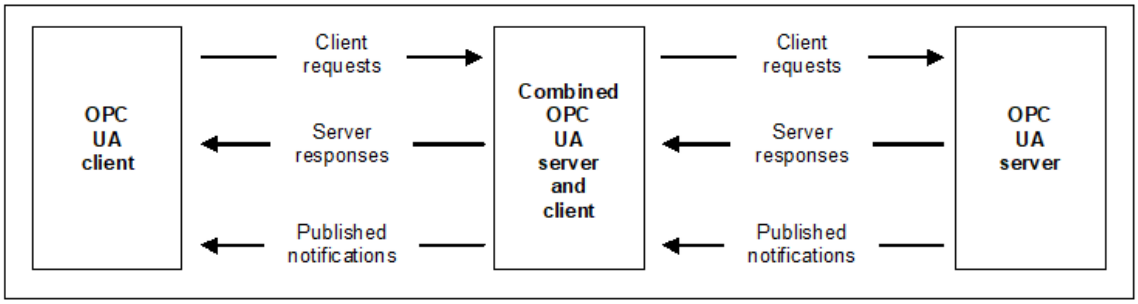


Figure 2-3 OPC UA system architecture. [11]

2.3.1 TwinCAT 3's TF6100 OPC UA Package

To implement OPC UA in the PLC in our project, a specialized function from Beckhoff is used and installed with TwinCAT Package Manager. This function is given a number TF6100 and is called TwinCAT3 OPC UA. Using the client-server model and the security features of OPC UA like using certificates and user authentication, the function allows the PLC to act as OPC UA server or client, to exchange data with other OPC UA clients or servers, respectively, seamlessly and securely. This function also supports data access by providing real-time read/write operations for different data types, which is considered the core of my project, historical data access for analysis and reporting, and alarms and system conditions for diagnostics [12]. It offers graphical implementation tools for configuring servers (TwinCAT OPC UA Configurator) and for a sample client (TwinCAT OPC UA Sample Client), which helps in testing and debugging.

2.3.2 OPC UA Configurator Overview

The OPC UA Configurator is an integrated tool in Beckhoff's function TF6100 TwinCAT 3 OPC UA that simplifies the setup and management of OPC UA communication for the

non-expert users. [13].

It allows the users to define the OPC UA server's address and map PLC variables to OPC UA nodes and create the relevant endpoints. It also enables the setup of user authentication and security policies, including the used encryption and access permissions. Furthermore, the user can use the tool to configure alarms, different events, and monitor historical data. Figure 2-4 shows the user interface of OPC UA Configurator.

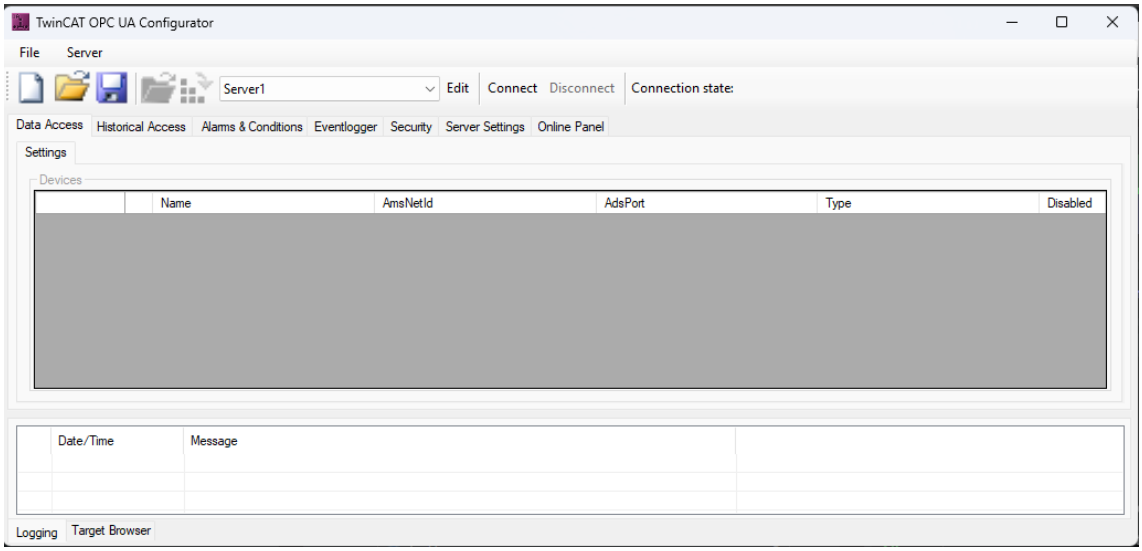


Figure 2-4 OPC UA Configurator

2.4 Tunnelling

After setting up the OPC UA server and linking the PLC with it, we'll need to publish the real-time data into the public network, i.e. the Internet, so it can be accessible by other permitted applications, which is the Android application in my project. To achieve this, multiple solutions were considered, such as using an external cloud server. But due to resources limitations, the choice was to use Tunnelling.

Tunnelling in networks refers to a concept of redirecting the data across the network from different protocols, such as OPC and TCP, through HTTPS. Figure 2-5 shows an illustrative example of tunnelling, where local network can be accessed remotely by using a tunnel.

Tunnelling consists of multiple processes as follows:

- Encryption, where the packets (small pieces of data) are encrypted for security and privacy
- Encapsulation, at which the encrypted packets are encapsulated in other packets over public network protocol, for example HTTP.
- Header definition to define the source and destination of the packet.
- Transmission of the encapsulated packets.
- Decapsulation.
- Decryption

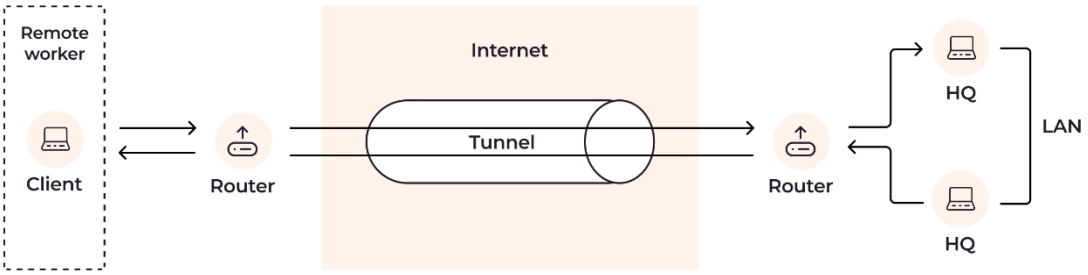


Figure 2-5 The tunnelling process. [14]

One popular application of tunnelling is Virtual Private Network (VPN), where it can be used to extend the access from private networks to off-site users, who aren't in the same network.

Because of its concepts of encryption and encapsulation, it offers enhanced security and privacy for the data and simple transmission by bypassing network restrictions, such as firewalls. In addition, its segmentation and encapsulation provide optimized network performance.

2.4.1 ngrok

ngrok is an environment independent software for tunneling the local endpoints into web service [15].

The way it works is simple, as it creates a secure, temporary tunnel connection between a public URL, that can be customized with paid plans, and a local web server by its port number. This public URL is used to direct the requests from end users to the local server through ngrok servers, which are responsible for sending the response in return to the end users [16]. Figure 2-6 shows an example of how ngrok works.

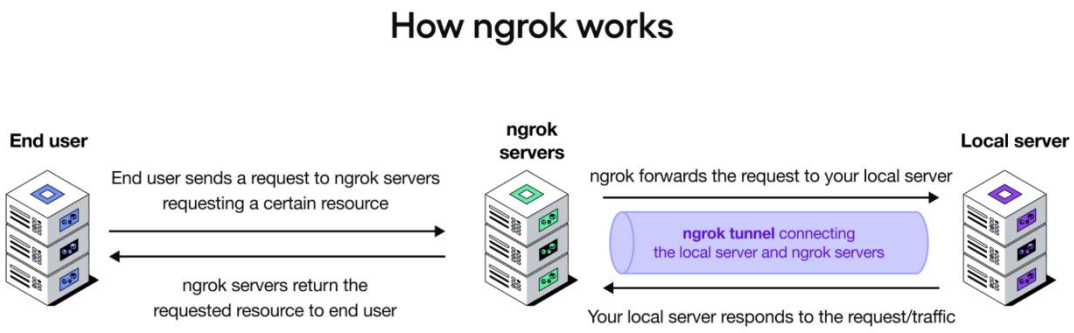


Figure 2-6 How ngrok works. [16]

Because it’s a user-friendly and free tool, ngrok is used in my project to expose the PLC OPC UA server that runs locally to the Android HMI application, enabling seamless and secure communication.

2.5 Used Libraries

2.5.1 Eclipse Milo

Eclipse Milo is an open-source cross-platform library developed by Eclipse Foundation for implementing and interacting with OPC UA protocol. It can work in a real-time client-server model, adhering to OPC UA standards, including the security compliance of supporting X509 certificates, encryption, and user authentication [17].

The library was a proper choice for implementing OPC UA client in the Android application, due to its ease of integration and the above-mentioned features.

2.5.2 AndroidX

AndroidX is a new set of libraries by Google to improve the development process for Android application, replacing the Support Library. It offers better organization and long-term support, and therefore enhanced functionality [18].

Two of the key features of AndroidX is the backward compatibility, to ensure the applications can run on older Android versions, and its modular design, where the developers can choose and use libraries when needed, resulting in better size optimization.

Specific AndroidX UI-related libraries are used in my project, because of its efficient responsiveness and easy integration.

3. SYSTEM ARCHITECTURE AND DESIGN

In this section, I will discuss thoroughly the system architecture and design for PLC and the Android application.

3.1 Overall System Architecture

The system architecture is shown in Figure 3-1 Overall system architecture., which illustrates the full system architecture, highlighting the communication paths and key components. For testing and demonstration, I have chosen the system to be an application for tank control, where the PLC will control the tank filling and emptying processes through the valves and based on temperature and water level sensors. Output status tags are also implemented to reflect the status of sensors and actuators, providing clear image of the system's operational status. Alarms for high and low level, and more, are also implemented to be tested on the android application.

The OPC UA Server will play an important role in ensuring stable communication between both the PLC and the Android application, in order to be able to control and monitor the PLC and therefore create the desired HMI screens. This communication is established via tunnelling service, depending on TCP protocol, to the local OPC UA server, which can be configured with TF6100 OPC UA server configurator, or in the TwinCAT3.

At the same time, an HMI might be configured in TwinCAT3 and linked directly to the PLC. This HMI can be also tunnelled and accessed from any web browser, regardless of the device type or the OS, based on HTTP protocol.

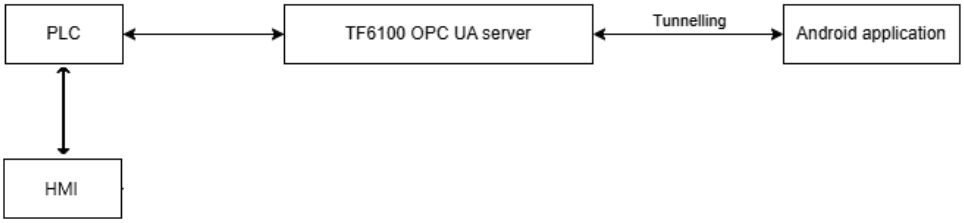


Figure 3-1 Overall system architecture.

In Figure 3-2, a demonstration for the TwinCAT3 PLC project's structure is shown. The project consists of 3 main Global Variable Lists (GVL): GVL_IO, GVL_Process, and GVL_AlarmsAndStates. This separation aims to contain separately the variables based on their function and improves the readability, scalability and maintainability. All these GVLs are processed in two separate Program Organization Units (POU) of type

Program (PRG), implemented in Structured Text (ST) language. These two POUs are run cyclically within the MAIN POU.

This approach eases the developers to extend or modify specific parts of the system, such as adding new sensors, actuators, or alarms, with isolating them without affecting the unrelated sections of the code, which leads subsequently to improved troubleshooting and will consume less testing time during commissioning phase. In addition, it ensures complete robustness in handling industrial communication challenges with OPC UA tunnelling, which was proved through stress testing with more than hundred concurrent OPA UA client subscription from the Android application, where it maintained stable and seamless communication.

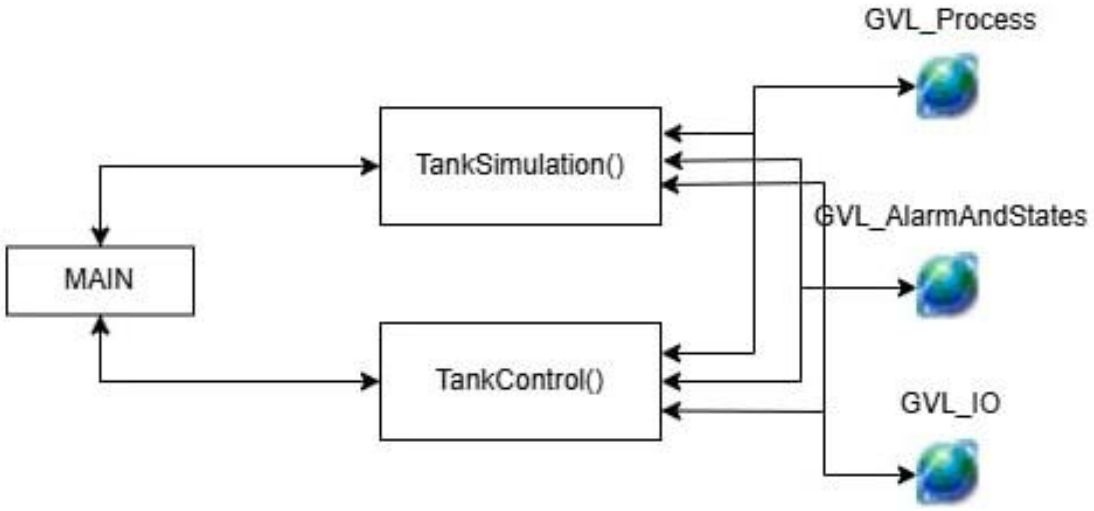


Figure 3-2 TwinCAT3 PLC project structure.

3.2 Application Design

The Android application has undergone several months in designing and developing, but its core structure is shown in Figure 3-4. The application was designed to be responsive, interactive, and reliable, to be able to serve the user with the basic defined functionalities.

The chosen architecture to design the application was Model-View-ViewModel (MVVM) (see Figure 3-3), which is widely used to separate the UI from the back-end logic. Model layer handles data from PLC (TwinCAT), alarm monitoring, and OPC UA communication. Whereas the View layer handles the UI rendering and user interfaces, the ViewModel layer works as mediator between the Model and View layers, handling the business logic. This model has plenty of advantages such as its simplicity to develop and test, and therefore maintenance.

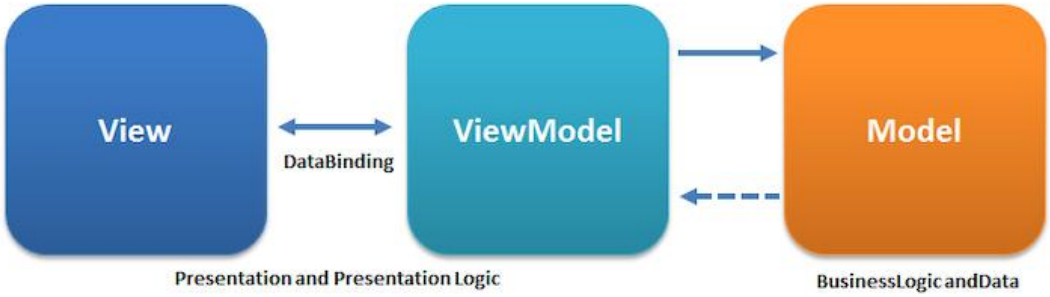


Figure 3-3 MVVM software pattern. [19]

The UI of the application mainly consists of 4 activities. The first one, which is the home page, is defined in class MainActivity.java, where it's linked to a layout resource activity_main.xml. The main and lone role of this page is to collect the tunnelled URL of OPC UA server from the user and navigate to the next activity page, that's CredentialsPage.java. As described in its name, this page is dedicated to get the username and password from the user and to connect to the OPC UA server, based on the given credentials.

In case of establishing a successful connection to the server, the application navigates the user to the third page of the application, which is defined as showing_rt_data.xml page in class BrowseData.java. This page will retrieve all the PLC variables, which were exposed to the OPC UA server when being defined in TwinCAT3, showing the real-time values of each of them and enabling the user to write a different value.

From showing_rt_data.xml page we can navigate to create the new customizable HMI screen in a new page (activity_custom_hmi.xml) defined in class GraphicalHmiActivity.java. This activity can work as abstract factory pattern to dynamically generate the desired HMI components, enabling noticeable flexibility during testing with the ability to successfully creating custom dashboards for more than 20 PLC variables without requiring any further code changes. Alarms can be shown and handled in a pop-up window in the same page, with priority queue that sorts alerts by severity.

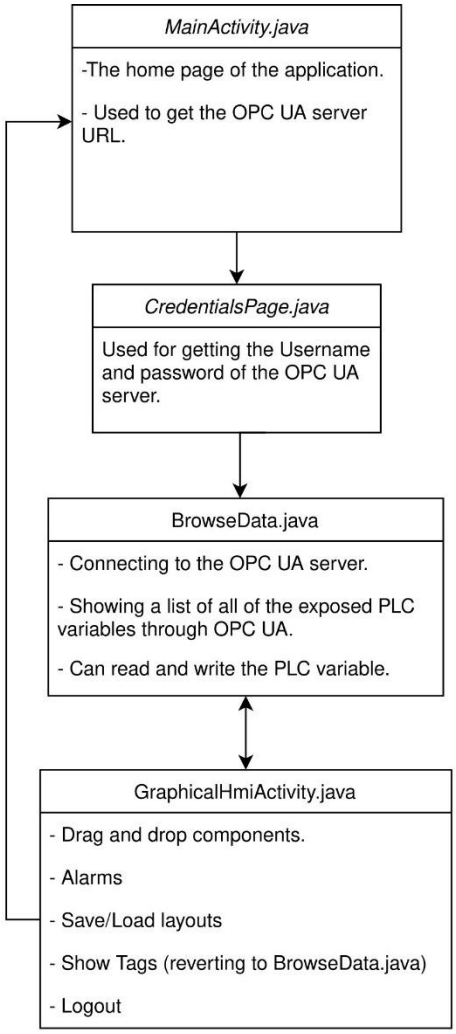


Figure 3-4 Application architecture.

This layered design pattern in the application offers a clean separation of logic, simplifies debugging, and supports future scalability, such as adding more advanced HMI features or implementing offline data logging and historical reports will be required.

Eventually, this dual adherence to both Android best practices and modern industry standards represents the key innovation of my design, making the application a diagnostic tool for control engineers and an operational user-friendly interface for plant personnel.

4. IMPLEMENTATION

The practical implementation of the project, the application, and the communication between them took place in several major steps and will be described in this chapter, where a brief demonstration of how the system components were configured and programmed in both TwinCAT 3 for PLC and Android Studio for the Android application to fulfil the requirements outlined in the first chapter. This implementation aims to transform the theoretical design into functioning prototype.

4.1 TwinCAT 3 Implementation

As discussed previously in Chapter 2.1.1, TwinCAT 3 from Beckhoff is a modern and flexible IDE for industrial automation, that complies with IEC 61131-3 standard for different PLC programming languages. For this project, TwinCAT 3 was chosen to program the tank system, simulate its process and expose IOs through OPC UA server. In order to download and install TwinCAT3 and all the relevant packages, TwinCAT Package Manager was used. The PLC project was structured systematically to ensure modularity, readability and scalability.

4.1.1 Package Manager

TwinCAT Package Manager plays major role in managing third-party libraries, software modules, and system extensions. It downloads packages and their dependencies from one or more so-called feeds. In the context of my project, the Package Manager was utilized to install and configure the TF6100 OPC UA server package, which enables OPC UA communication on the PLC by offering all necessary services and functions to expose the PLC's internal variables as OPC UA nodes, that can be accessed externally.

The configuration of Package Manager required signing in with myBeckhoff account, which must be created initially on Beckhoff's official website. In addition, it will require being integrated into Visual Studio or eXtended Automation Engineering (XAE) shell. A list of available packages is shown afterwards to be downloaded and installed or updated. Package Manager also accepts "tcpck" commands for the command line to manage different TwinCAT components or individual packages.

Figure 4-1 illustrates Package Manager after logging in, to download and install different packages.

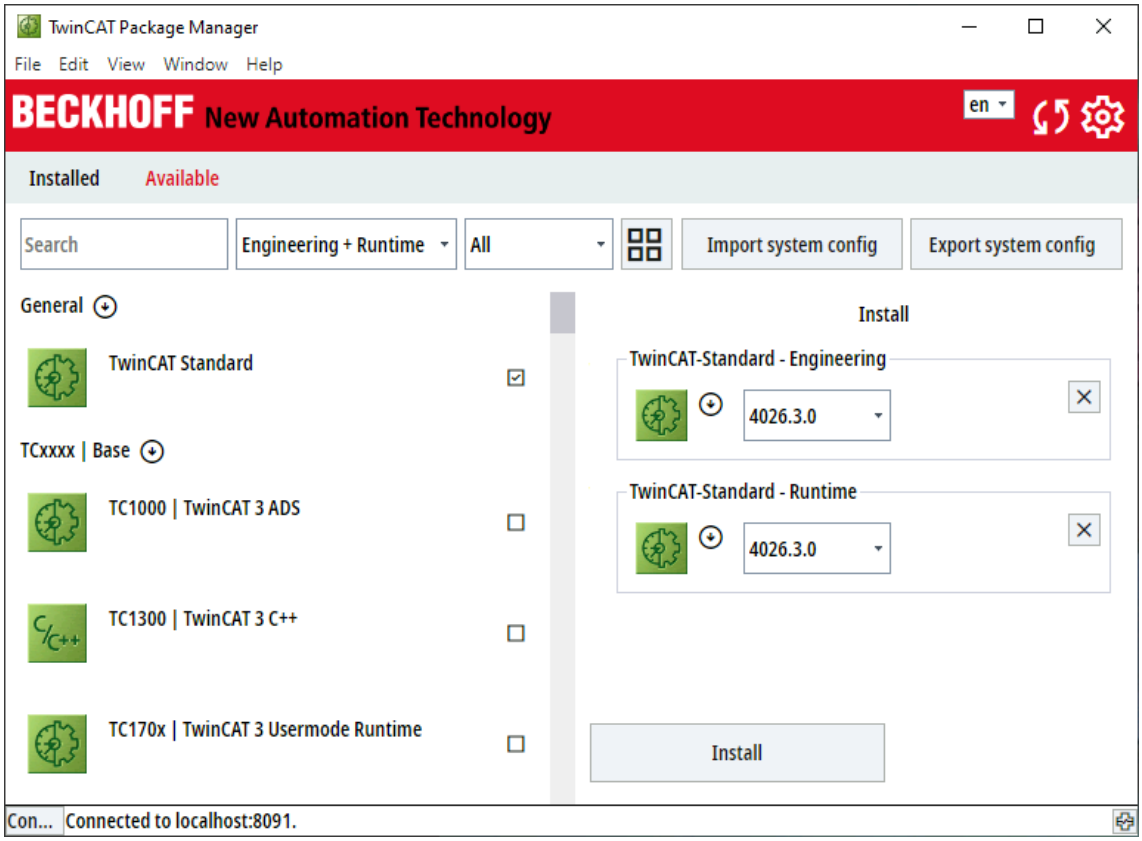


Figure 4-1 TwinCAT Package Manager.

4.1.2 Program's Logic

First, a new XAE project must be created, and a standard PLC Project should be added to it in the PLC section. In addition, a free 7 Day Trial License must be guaranteed to the selected packages and functions, such as TC1200 (TC3 PLC) and TF6100 (TC3 OPC-UA). Further configurations regarding setting up the OPC UA communication are discussed in the next sub-section.

Figure 4-2Figure 3-2 illustrates the project tree in TwinCAT 3. Separation of concerns and promoting modularity were critical by dividing the program into multiple POUs and storing the variables in multiple GVLs, based on their usage. A conventional naming starting with a letter, which represents the datatype, is used to ease the process of choosing the proper HMI component for the operator in the application.

The first major GVL, GVL_IO, contains all input and output variables that reflect the physical signals, such as Start and Stop buttons and Low- and High-Level sensors for inputs, or output actuators such as Pump Run bit to indicate the status of the pump or Valves bits. By centralizing all IO variables in one GVL, they would be easily referenced across several POUs.

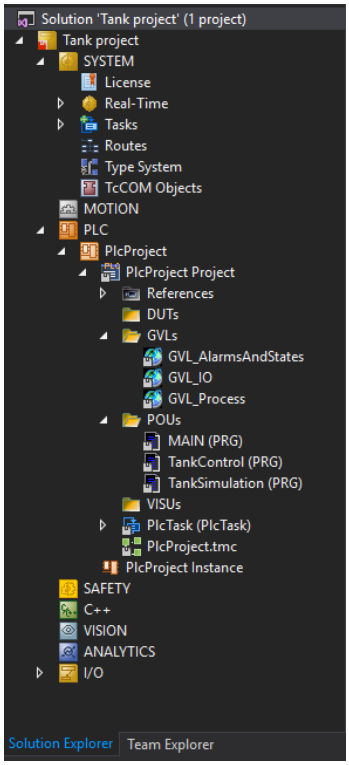


Figure 4-2 Project tree in TwinCAT3.

The second GVL is GVL_Process and it holds variables that represent the internal process states and calculations. This includes flow rates, setpoints, and intermediate variables to simulate the filling and emptying processes. These variables are updated every scan cycle, to reflect the process over time. Separating such variables from IO variables helps to prevent mixing the hardware configuration logic with process modelling.

The third and final GVL is GVL_AlarmsAndStates and it's responsible for all alarms and system states variables, including flags for high- and low-level alarms, over-temperature alarm and system health status. These alarms are configured in the OPC UA server and defined in an AlarmCondition according to their values, severity and alarm texts. Although these alarms are triggered only on the server side, their values are still handled and updated in the POUs, such as TankControl POU in my project.

The process control POU (TankSimulation) is regarded as the heart of the simulation, as it continuously updates the tank level based on the inflow and outflow rates and the inlet and outlet valves. For calculations, I decided to make them time-based, where the change in tank level for every PLC scan cycle is updated and calculated as the net inflow minus outflow times the cycle time. Such continuous update approach offered robust and smooth simulation and prevented any sudden and undesired jumps in processing the values.

In control POU (TankControl), the logic for managing the tank's physical operations was implemented, where it monitors sensor inputs, executes necessary control decisions,

and updating the actuator outputs. The alarm management in this POU works by monitoring processed variables from GVL_Process continuously, comparing them against defined thresholds in GVL_AlarmsAndStates, and triggering the flags if alarms were set.

Finally, and most importantly, the entire logic was successfully integrated with the OPC UA communication layer, by selecting particular variables from all three GVLs and exposing them to the OPC UA server using the TwinCAT attribute "*{attribute 'OPC.UA.DA' := '1'}*", as Figure 4-3 shows giving access to the OPC UA server for every variable. Further attribute declaration can be used, if necessary, to configure the read/write permissions of every selected variable.

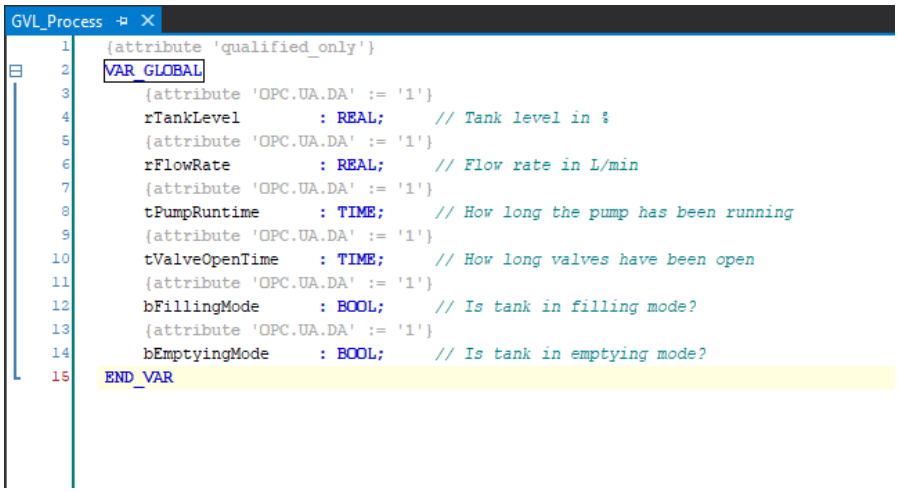


Figure 4-3 Variables declaration in GVL_Process.

4.2 OPC UA Configuration

The implementation of OPC UA communication in my project was the cornerstone of it, as it was the chosen communication between the PLC and the Android application, due to its various advantages, which was described briefly in 2.3. Key aspects of the configuration include selecting and preparing the OPC UA nodes, which will refer to the PLC variables, and configuring the security settings to satisfy the application requirements.

4.2.1 Security Settings

Security was crucial in my project and occupied significant care during OPC UA configuration. One of many reasons behind the selection of OPC UA communication was its robust security model, which uses X509 certificates [20], encryption and access control. In this project, I have chosen a security policy Basic256Sha256, with message security mode of type SignAndEncrypt, to guarantee maximum confidentiality, integrity, and reliability. This type of security policy is well-known for its strong level of encryption

using SHA256 (Secure Hash Algorithm) encryption algorithm with message signing, alongside using the binary encoding over TCP (opc.tcp) for maximum efficiency and fast message exchange between the PLC and the Android application.

The security configuration was done in several steps. First, and during the initial logging into the local server in TwinCAT OPC UA configurator, user authentication is configured with assigning a username and password to log in the server. Then after successfully connecting to the server, the security policies for the server's endpoints can be configured from the tab "Server Settings". After finishing all the security configurations, it's essential to "Activate On Target" to apply the changes on the locally running server.

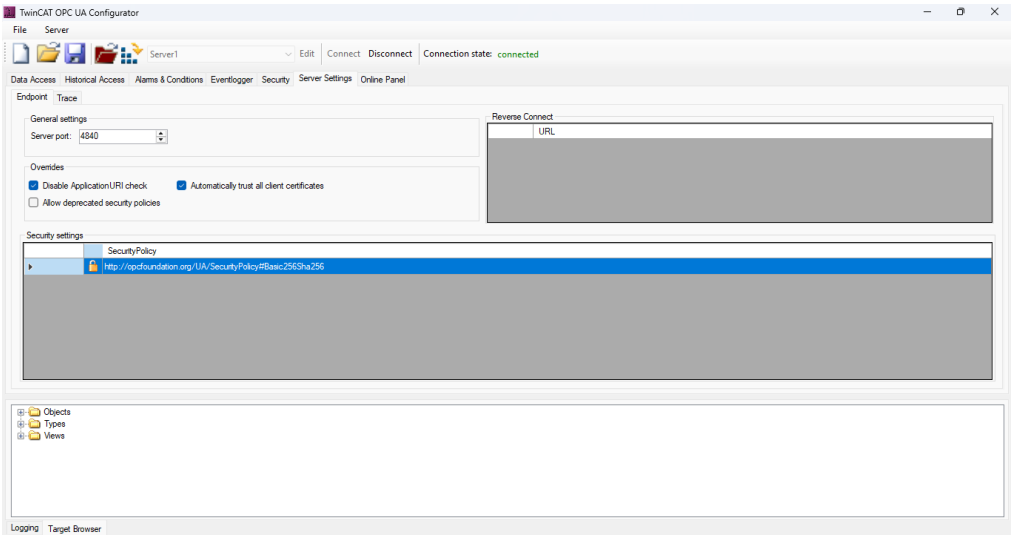


Figure 4-4 Security Settings in OPC UA configurator.

4.2.2 Integrating OPC UA to the TwinCAT 3 Project

Modifications to the PLC logic were required to utilize the OPC UA communication in the project. First, we'll need to activate the automatic download of the target file (TMC file) in the properties of the project (Figure 4-5). Exposing variables were marked using VAR_GLOBAL declarations and annotated with ATTRIB tags, such as "{attribute 'OPC.UA.DA' := '1'}" (see the example in Figure 4-3), which is used to enable the data access for OPC UA server to this variable.

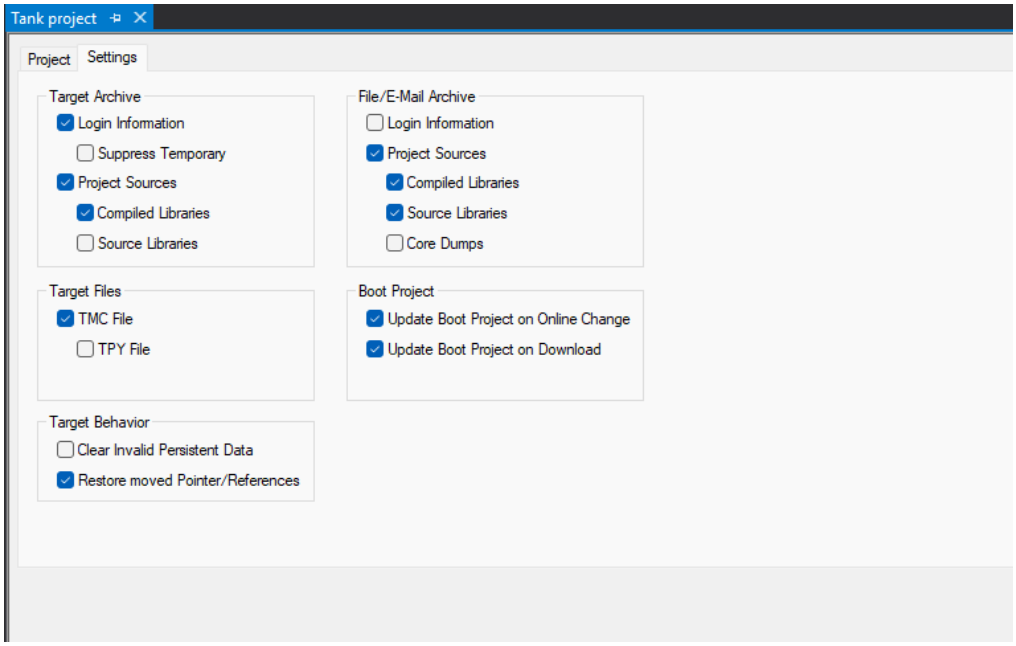


Figure 4-5 Activating TMC file in the project properties.

As part of integrating the alarm system into the server, the GVL_AlarmsAndStates variables are exposed as well into the server. A special alarm condition was created to contain all the required alarms, that were mapped from the GVL, and been configured in "Alarms & Conditions" tab, where the alarms are being configured by dragging the representative variables from the Target Browser below (see Figure 4-6).

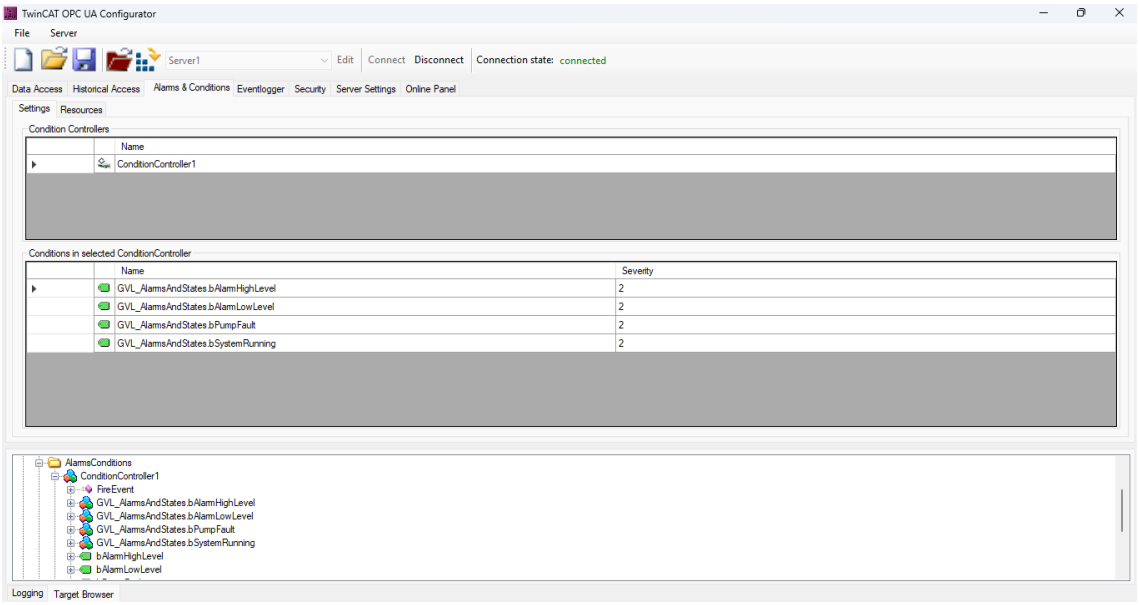


Figure 4-6 Alarms & Conditions in OPC UA Configurator.

The OPC UA Configurator proved to be a robust and user-friendly solution that simplified secured and reliable communication between the PLC and the Android HMI

application, by following the integration process, including security configuration and systematic testing that would be discussed further in 5.Testing.

4.3 Tunnelling Implementation

As remote access to the TwicCAT 3 OPC UA server from the Android application was the core of the project, tunnelling has been selected as a cost-effective and secure solution, because of the resources' limitation, as discussed in **Error! Reference source not found.**. The desired result was to have a publicly accessible URL (opc.tcp//…) that forwards responses and requests, from and to the local OPC UA server (by default: opc.tcp://localhost:4840). I chose ngrok to be the selected tunnelling interface for this task, because it's known for its simplicity, flexibility and robust security as it utilizes TLS/SSL encryption by default when creating public tunnels [21], which complements the SignAndEncrypt:Basic256Sha256 security setting that was configured on the OPC UA server. This double layer of security configuration ensures that the data transmitted is protected both at tunnelling layer and OPC UA protocol layer.

4.3.1 Setting up ngrok

To setup ngrok, the official ngrok software must be first downloaded from ngrok website. It's a portable executable file that doesn't have to be installed on the PC. In addition, an account must be created and verified to be able to tunnel any connection on the local PC.

After creating an account and downloading the ngrok software, a certain command must be entered in the command prompt, with an "Authtoken" to be added to the default ngrok.yml configuration file. In that way, I managed to link my account to the downloaded software and be able to tunnel the OPC UA server.

Next, a custom tunnel was set up using TCP protocol, as OPC UA typically uses TCP for its binary endpoint on the port 4840, the command "ngrok tcp 4840" was used to forward any incoming or outgoing traffic through this public ngrok TCP address, as shown in Figure 4-7.

Overall, the tunnelling implementation using ngrok successfully extended the range of my local PLC system, allowing me to have a global access without sacrificing or compromising neither the security nor the performance.

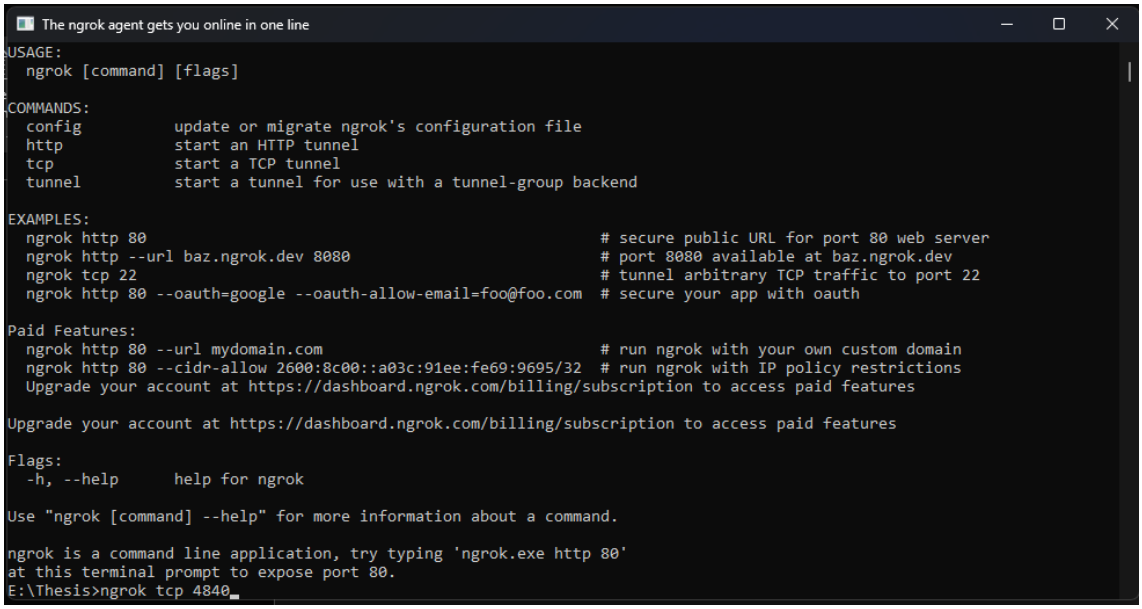


Figure 4-7 Tunnelling the OPC UA protocol over TCP port 4840.

4.4 Android OS Implementation

Implementing Android OS in my system was a very critical stage in my thesis, where it translates the application theoretical design into practical and functional application that satisfies the project's requirements. I decided to use Android Studio as the main IDE for developing, testing and releasing the application due to its powerful debugging tools, layout designer, and seamless Gradle build system. I mainly targeted and considered creating a robust, reliable, though user-friendly mobile interface to complement the PLC system seamlessly, so at some point I manage to bridge the industrial control system to the user in a secure, responsive, and intuitive manner.

4.4.1 Libraries and Packages

To achieve my task and to have the intended functionality and robustness, a combination of a third-party libraries and Android-native packages were used during my implementation of the project. These libraries have been selected because they significantly reduce the development time, while maintaining reliability, performance, and maintainability of the code, and allow the integration of networking, UI responsiveness and security into the Android application. Figure 4-8 shows a list of dependencies for the used packages and libraries, and it's defined by nature in build.gradle file.

The most important library I used in this project was the Eclipse Milo OPC UA stack. It's an open-source library with active community support and full compliance with OPC UA standards, especially for secure communication and session management. This library enabled me using the necessary API and client implementation to connect, browse, and read/write the data from the OPC UA server.

In order to implement reactive UI elements and other background tasks, I used Android Jetback components, such as appcompat and collection. Other components, for example recyclerview, ensured that the UI components are all in sync with the backend logic, especially when it comes to dealing with periodic real-time data updates. These components offered additional features such as rotational components, without causing the application to crash.

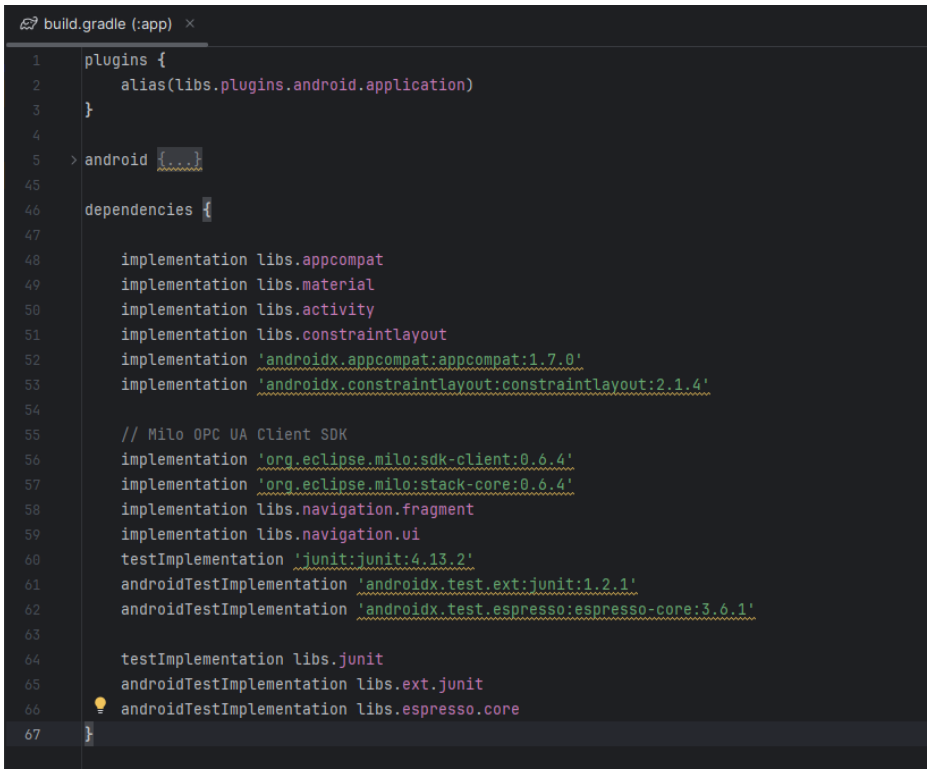


Figure 4-8 List of used dependencies.

I used several Android UI libraries as well for displaying real-time data and status information, where Material Design components, such as buttons, switches and textviews, were used for this purpose to provide a polished and intuitive interface. Other libraries were imported for very specific purposes, including JSON parsing for saving/loading the HMI layouts.

In the process of continuous developing and later testing, I needed to use logging, especially when I needed to debug unexpected bugs. Therefore, I have used Log library, assigning the messages based on needs and monitoring them, while having the application running in the emulator, from the Logcat tab in Android studio.

Eventually, I made sure before starting to develop the application, that all necessary permissions are granted for the user, namely the permission for using INTERNET. In this way, I made the choice of the libraries and packages carefully to serve the specific needs of a secure, OPC UA dependent industrial monitoring and controlling application.

4.4.2 The implementation of the application

In the phase of implementation, I divided the process into smaller stages, to ensure that every functionality is merged with the other to work flawlessly and seamlessly.

The first phase of the implementation was to implement the required pages and navigating between them in a certain sequence according to the desired UX. As per the application design, the application should contain 4 pages, Main page, Credentials page, BrowseData and Graphical HMI activity. These four pages were implemented as activities (Figure 4-9) with navigating between them using Intents in their class definitions.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".BrowsingData"
            android:label="BrowsingData" />
        <activity
            android:name=".CredentialsPage"
            android:label="CredentialsPage" />
        <activity
            android:name=".GraphicalHmiActivity"
            android:label="GraphicalHmiActivity" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figure 4-9 Defined activities in AndroidManifest.xml

The beginning was with the Main Activity, which is the application's starting page and collects the user's input for the OPC UA server tunnelled URL. Once validated, it navigates to the Credentials page, where the user enters the right username and password to log in to the server. This ensures that only the authorized users are granted to the system.

After successfully logging in, the BrowseData page is shown, and it basically a real-time data monitoring activity that shows all the available PLC variables, which were exposed to the original OPC UA server. This is done after the application's client successfully and securely start communicating asynchronously with the OPC UA server. These variables are presented in a list that is being updated periodically every 1000 ms

and can be written by the user, in case he needs to make quick modifications to any of the PLC variables. The writing operation in this page and the next page will be mapping the entered value by the user to be of the same data type as the PLC variable to prevent any mismatch of data types.

The last separate activity is responsible for the customizable HMI screen and is called “GraphicalHmiAcitivity”. It consists of blank canvas and two hidden draggable sidebars. The right sidebar offers built-in components such as Buttons, TextViews, EditTexts, Switches, as well as defined components, for example dynamic charts (both bar and line types) and indicating shapes (rectangle, triangle, and circle). Figure 4-10 shows the LinearLayout of the right sidebar, containing all necessary components for my tank system. All these components can be dynamically connected to PLC variables, as well as statically configured when needed, and provide a more visual and component-based approach to monitor and control the system.

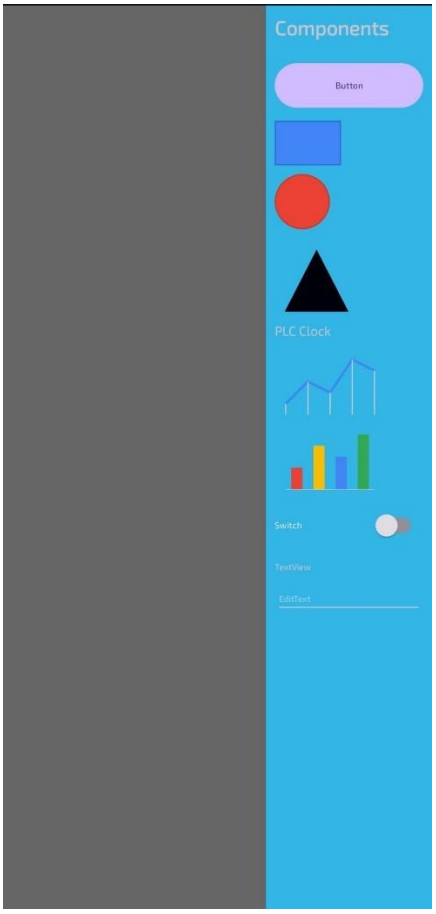


Figure 4-10 Right sidebar of customizable HMI page.

Each of these components can be dragged and dropped on the blank canvas, to create a copy of it. With a long press listener on each of them, a drop-down menu shall be shown based on the component type. However, there is common options in this menu, that include Copy, Move, Delete, Bind Variable, and other dimensional properties like width

or height, and all of them perform a certain function exclusively to the selected component.

The major trickiest components were the charts and shapes, as they, naturally, are not built-in components and had to be implemented in the code. The charts were challenging as I wanted to make them change dynamically with a selected PLC variable, and at the same time maintaining the user's ability to configure the sample rate and other graphical properties for these charts. On the other hand, shapes took plenty of effort in order to make them function both dynamically and statically. Dynamically, they can be bound to a variable and have settings based on the data type, if it's numerical (in general) or boolean, with defining the range and colours. In addition, special property was added to shape components, i.e. rotation. I found that in same use cases, it's required to have the shape component, for example a triangle, to be rotating to fit with the desired design. Figure 4-11 shows the shape_menu.xml, where I defined the needed properties of all shape components.

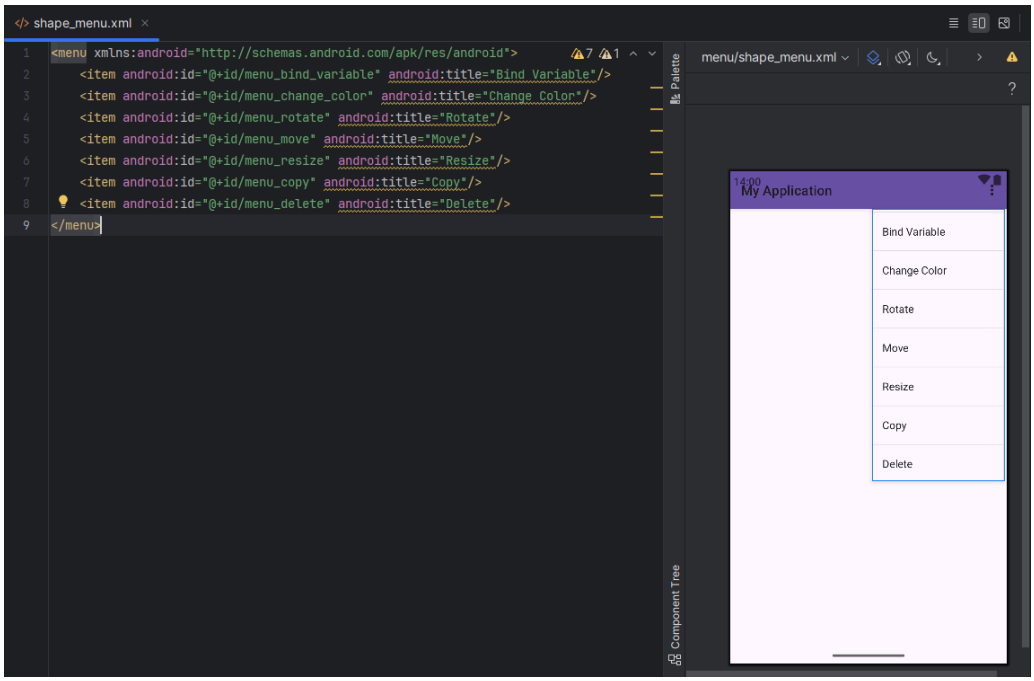


Figure 4-11 Shape menu's XML code.

Error handling and connectivity were key concerns for fully functional and stable UI. Therefore, toast messages were used in case of throwing an error, with proper texts that help the user to identify the root cause of the problem. In the meantime, it's worth mentioning that the connection to the OPC UA server should be maintained under all circumstances, to manage to consistently design and configure the desired HMI screen.

Left sidebar, shown in Figure 4-12, is more concerned with Screen management, including options such as "Show Tags", which navigates to the previous page "BrowsingData", "Save Layout" and "Load Layout" to parse the layout as JSON file, and

“Log out”, which disconnect the connection to the OPC UA server and navigates back to the home page. However, to enhance the usability, the list includes the most important feature, which is showing AlarmsConditions from the OPC UA server. Alarms were monitored using OPC UA event model and displayed as pop-up dialog, which includes the relevant tags, time stamps, and severity indicators, with an option to acknowledge the alarms.

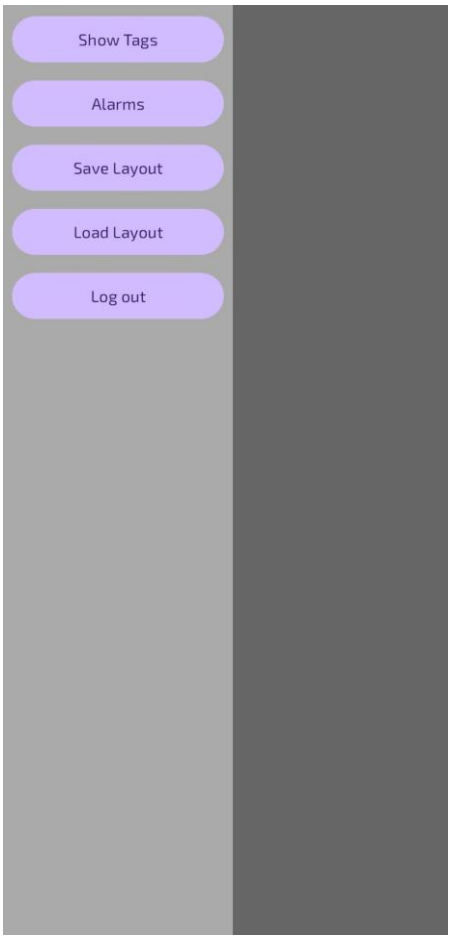


Figure 4-12 Left sidebar of customizable HMI page.

4.5 Building, Releasing, and Installing APK

After developing the application, the stage of deployment and releasing in APK (Android Package Kit) comes as an important stage in our project. It basically transforms the developed code, resources, assets into a distributable binary that can be installed on Android devices.

After carefully testing the application’s functionalities, stability and UI responsiveness according to the project’s requirements, I initiated the build process to create a signed release APK.

As part of the build process, I had to configure the Gradle build system, which manages the project’s dependencies, compilation and versioning. The appropriate settings

were defined in the build.gradle file (Figure 4-13), including the minimum SDK (Software Development Kit) version, target SDK version, and version code and name for versioning.

Next, a keystore was generated to be used in signing the release APK. The keystore is a secure file to hold the cryptographic keys used to sign the application, which is necessary to ensure the authenticity and integrity of the app when installed on user devices. This step is done on Android studio from navigating to Build from the toolbar, then “Generate Signed App Bundle / APK”.

After the APK was built and signed, it was tested on the real Android device by being installed on my personal device, after transferring the APK file to it.

For broader distribution or future updates, it’s recommended to upload the application on Google Play Store, as it will give the application higher credibility to be widely promoted and used for various systems.

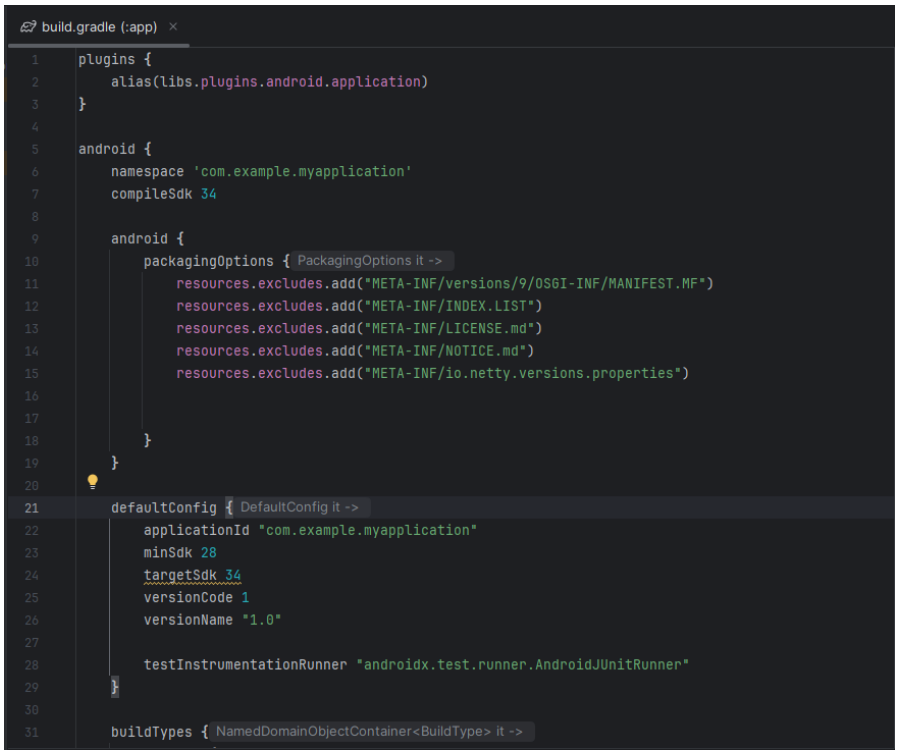


Figure 4-13 Snippet code from build.gradle file.

5. TESTING

Testing was a crucial phase of my project to validate the reliability, accuracy, and the performance of the whole project comprising TwinCAT 3 PLC project, OPC UA server and the Android application. Each of these components was tested separately before integration into the system and followed by another to examine and compare after the integration, so that I'll have objective results in the end.

5.1 TwinCAT 3 test

The testing of the PLC logic was performed in the TwinCAT 3, using the simulation. The built-in simulation allows real-time monitoring of variable values, function block outputs, and overall sequence of operations. The main task was to validate the correctness of logic, including tank control and alarm triggering based on the defined thresholds.

Each POU was tested individually and monitored during runtime. During testing, key variables, for example bStartButton and bFillingMode, were written to verify to corresponding control process.

In addition, the GVLs were also given careful attention, especially GVL_AlarmsAndStates, which is responsible for triggering the AlarmConditions in the OPC UA server.

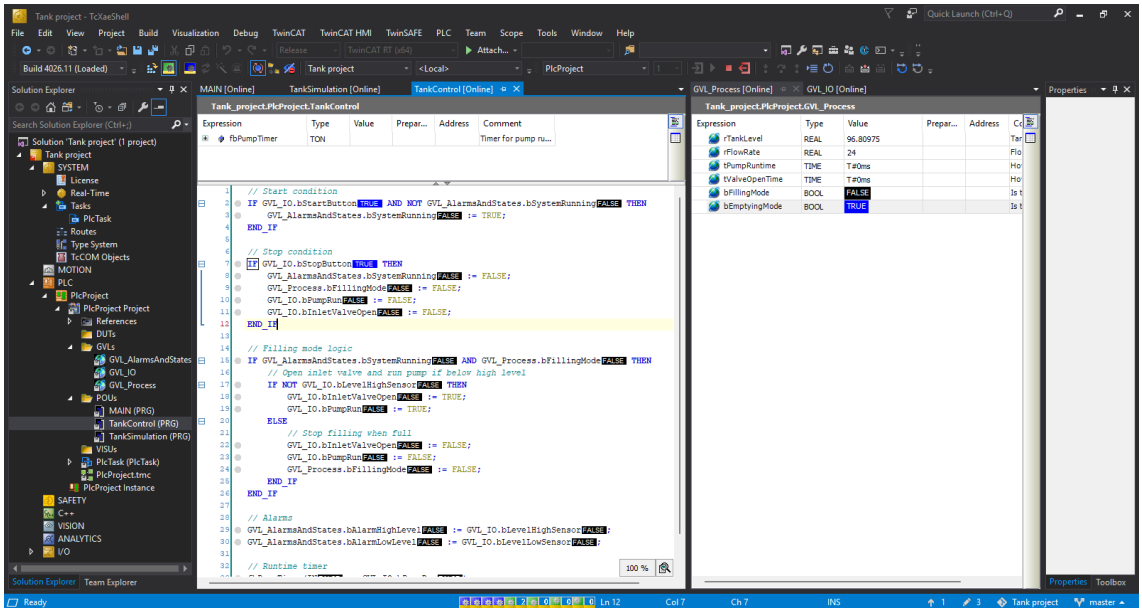


Figure 5-1 Testing GVL_Process and TankControl using Run Mode in TwinCAT 3.

Unfortunately, and because of limited resources, I didn't have the possibility to test the PLC logic with real hardware, such as sensors, actuators, and surely the Beckhoff controller.

5.2 HMI Android application test

To get reliable and consistent results, I have tested the application on multiple devices and Android Studio's emulator. The testing focused mainly on the responsiveness of the UI, data accuracy, maintaining connection, and handling various inputs.

The login functionality was tested by entering both valid and invalid credentials. The application successfully managed to login with the appropriate permissions using correct credentials and establish secure a secure session with the OPC UA server, In the contrary, it denied the access and didn't retrieve any data from the server in case of using false credentials. In both cases, the application navigated to the PLC variables page, with the expected and desired results for each. Besides, I tested logging in with several users to ensure the proper usability and connectivity to the server.

Upon clicking on "Create a new graphical screen", the application navigates to the blank canvas to design the HMI. All drag and drop components worked properly with the desired functionalities. Example of HMI screen for the system. Figure 5-2 shows an example for an HMI screen, representing basic switches, LEDs and a level indicator with indicating texts to illustrate every single component.

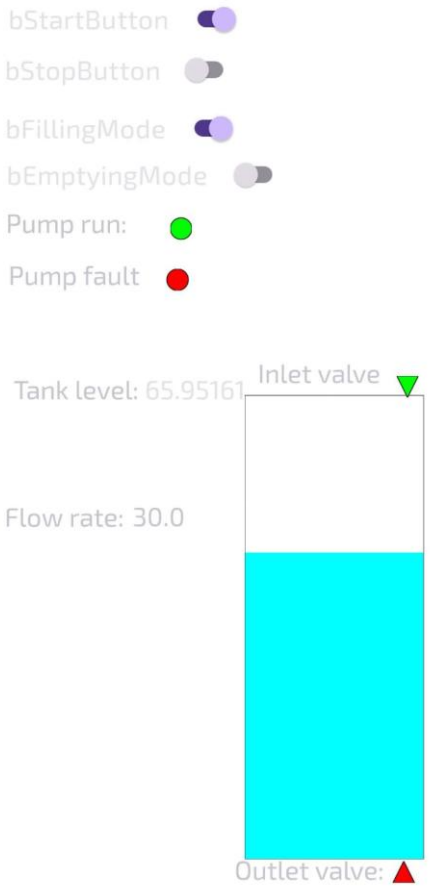


Figure 5-2 Example of HMI screen for the system.

However, it was noticed that both charts are shifted one row on the Y-axis, as shown in Figure 5-3. This might be because of the dynamic mechanism of adapting the Y-axis to the current values of the bound variable.

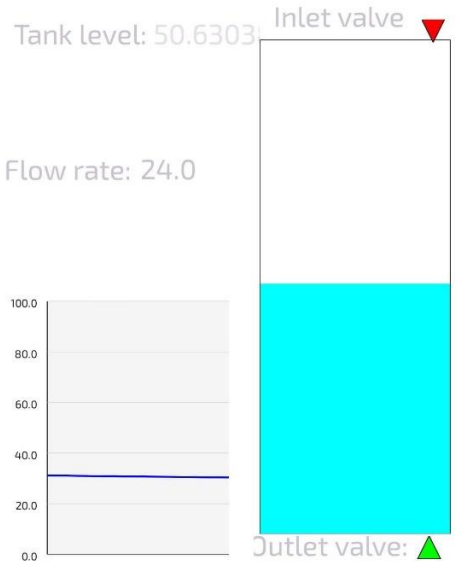


Figure 5-3 Line chart of tank level.

In addition, and most importantly, the Alarms screen had bugs in showing all alarms data, as Figure 5-4 shows (BadDeviceFailure) error, and this is likely due to a problem with the connection to the OPC UA server, fetching all the alarm's data and handling them. The Alarms screen had to be open in order to track the newly triggered alarms, as it does not retain them until being acknowledged.

Other features, such as Save Layout and Load Layout, didn't pass the tests and didn't work as expected. I tried to save the layout by saving the components types, their coordinates, and their contents in a parsed JSON file and load the same file, but unfortunately it didn't work properly and replicate the components as originally created.

In addition, error handling was properly tested by entering wrong values deliberately, which has shown toast messages with prompting the user with proper messages.

Overall, the tests were carried out with huge focus on the basic functionalities and usability. Otherwise, the bugs can be resolvable and the features are widely expandable.

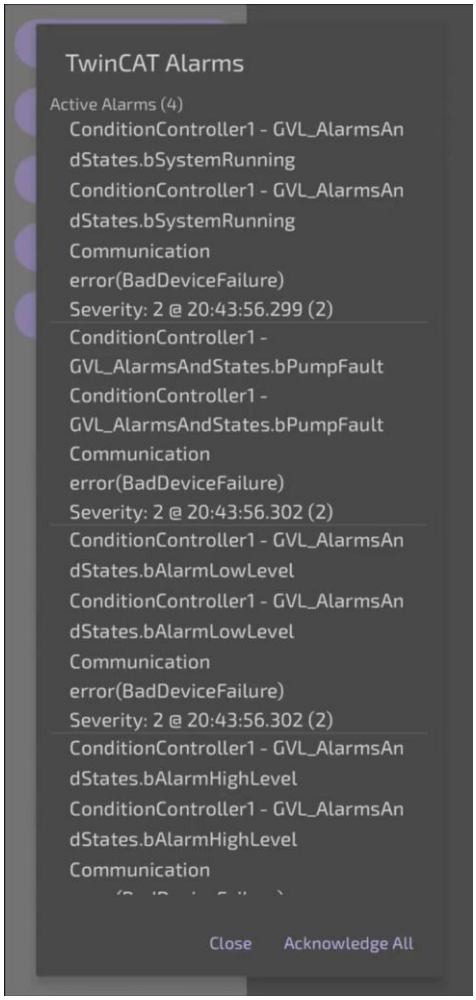


Figure 5-4 Alarm pop-up window.

5.3 Communication test

Communication testing was the most critical part of verifying the system's stability and feasibility, as it was basically the core of the project to communicate remotely between the PLC and the application. I relied in my tests on the Logcat in Android Studio to see the log of variables and events and see the communication established, with using a real physical mobile phone. Besides, I used UaExpert to track the local OPC UA server and track the timestamps of occurring changes and events from the application through the tunnel.

At first, I established the connection between the Android application, which acts in our case as OPC UA client, and the TwinCAT 3 OPC UA server. It was done by tunnelling the connection through ngrok and using its TCP address as the input into the application. The application successfully connected to the server and managed to retrieve and trust the server certificate, with confirming that encryption and trust were properly established.

The live communication was verified by changing dynamic variables, such as bStartButton and monitoring rTankLevel. For this test scenario, I changed the state of variable bStartButton from the application and monitored it through UaExpert and the Logcat of Android Studio. The latency was minimal. However, in Figure 5-5 and Figure 5-6, the change occurred on the server side (19:41:08.636) around one second before observing the handshake on the application log (19:41:09.540). The reason behind this is that I used asynchronous OPC UA writes, especially CompletableFuture type, via Eclipse Milo SDK. The sequence of events is that the client sends write request and the OPC UA server processes it immediately, then the server updates the value and sends a response and finally the Android client processes the response and logs it in Logcat. Therefore, the network stack delays such as buffering, queuing, and threading, play a role in such behaviour.

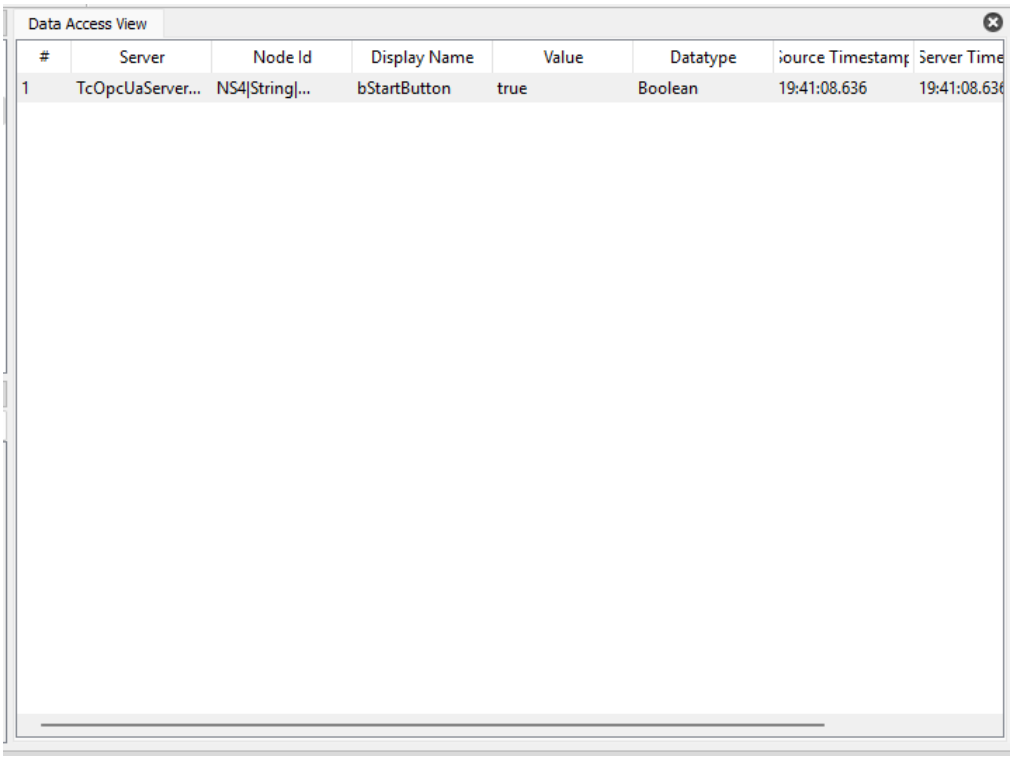


Figure 5-5 UaExpert's Data Access View window with monitoring the change event on bStartLevel.

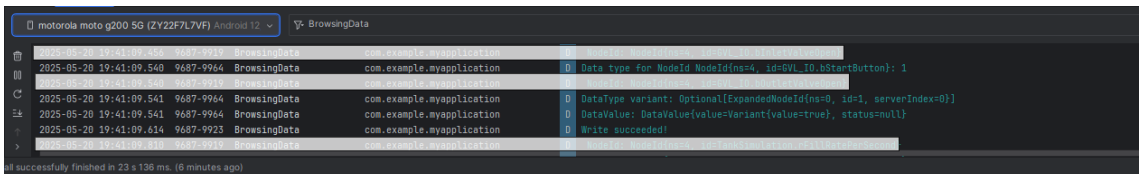


Figure 5-6 Log data of the event change on bStartLevel by the application.

In addition, time properties were examined and observed as well, such as the round-trip time for write requests from the application to the OPC UA server and back. It was tested multiple times and were of average 50-120 ms. An example is shown in Figure 5-6, where the start of the request was indicated by the message "DataValue: DataValue{value=Variant{value=true}, status=null}" and the response, represented by the message "Write succeeded!". The time difference between both were of 73ms (19:41:09.541 and 19:41:09.614 respectively). These values include all network stack overheads, encryption, and processing delays from the Android OS and the ngrok tunnel.

Overall, the latency in reading and writing was minimal and suitable for highly sensitive and demanding applications, where latency represents critical factor in the process.

Other tests were carried out to evaluate the application's behaviour under network interruptions. For example, when the ngrok endpoint was disabled or the OPC UA server was disconnected, the application handled the timeouts and retried the connection periodically. Upon server restoration, the app resumed the communication seamlessly. To achieve this, proper session timeouts were intentionally set short. In that way, the application could be monitored for how it handled expired sessions, disconnections, re-authentication, and resumed communication.

6. RESULTS

The project finally resulted in successful implementation of tank simulation and monitoring the system using TwinCAT 3 PLC software and a customizable Android HMI application. Key components of the project were integrated together, such as the control logic in the PLC, OPC UA server configuration, tunnels via ngrok, and the Android interface that's interacting with the PLC, all to achieve maximum performance and reliability.

The Beckhoff PLC system (Windows-based) interacted securely with the mobile application using OPC UA protocol. This was done by using the TF6100 OPC UA function and its relevant tools, such as TF6100 OPC UA server configurator, which allowed secure and reliable communication between both ends. Thanks to OPC UA implementation, the Android application managed to browse variables and change the values dynamically with alarms visualization and with minimal latency, which has confirmed the two-way communication of the OPC UA client-server model.

The security was, additionally, a major concern in the communication implementation. The system successfully utilized a robust security policy, that's "SignAndEncrypt:Basic256Sha256:Binary". This security policy encrypted all communications and limited them to authenticated users with the valid credentials. Besides, the tunnelling method using ngrok added extra secure layer, by isolating the internal OPC UA server from public internet, while effectively creating a VPN-like secure channel without demanding advanced networking knowledge, any complex infrastructure, or firewall changes. This has already made the application more practical and secure for small or medium-sized industrial setups.

In terms of performance, the whole system showed excellent response times at testing, with average read/write delay less than 500ms on stable network connection. Real-time data was therefore polled and updated on the application without any hindering lags.

Another concern in the design stage was the usability of the application, and the result reflected this effort. The MVVM architecture made the application scalable, maintainable and further expandable, where the modular UI allowed the users to monitor and write live data from PLC, acknowledge alarms, and of course configuring HMI components intuitively.

Functionally, all planned objectives, which were outlined at the start of the project, were met. The project demonstrated a simulation environment where the tank's fill and drain cycles could be monitored and controlled in real-time, with showcasing efficient use of OPC UA tunnelling and encryption to enable remote and secure access from the application. From development point of view, the separation of concerns between PLC control, OPC UA integration with tunnelling it, and mobile development, proved

beneficial, and the modular design facilitated debugging and performance optimization, with keeping in mind the broader scalability in the future.

Overall, the results of the project affirmed that Android OS can be integrated with modern PLC tools together into a complete control and monitoring industrial system, with ensuring secure communication standards and mobile application development.

7. CONCLUSION

This thesis' main objective was to design and implement a secure, remote-controlled HMI system for a simulated tank using TwinCAT 3 for PLC programming and Android OS for the mobile application. The system demonstrated how industrial processes could be made more accessible, user-friendly, responsive, and secure over the internet using OPC UA as the communication protocol and secure tunnelling via ngrok. Furthermore, another goal was set to explore the practical application of IoT technologies in industrial automation environments.

The project demonstrated modular approach for development, starting with PLC programming using TwinCAT 3, followed by configuring an OPC UA server to expose those PLC data securely and publishing it into the public using tunnelling, and finally a simple functional Android application to connect with the server, visualize the data, and interact with performing control actions. I took into consideration testing each of them individually before integrating them to work as a unified system and testing and validating them once again.

The significance of the application lies in its real-life applicability. As industries nowadays shift toward remote control of the systems, secure and platform-independent solutions like this application become crucial. The solution ensured scalability and flexibility of OPC UA's open standards and the cross-platform compatibility of Android OS. Besides, the tunnelling solution avoided the complexity of public IP allocation and firewall configurations, making the system feasible for small- to medium-scale systems and futuristic research applications.

I encountered several challenges in most of implementation, but one of the biggest obstacles in the beginning was establishing the communication and configuring the OPC UA server and its security settings correctly to meet the application's requirements. Although tunnelling was simple and easy method to employ in my project, it was a huge learning experience, where I understood how ngrok encapsulates and routes the traffic securely, which required a deep dive into network protocols and tunnelling principles.

Additional bugs in the Android application's code were encountered, such as bad retrieving of alarms' data or misalignment for charts. However, such bugs are easily resolvable in the future.

For future improvements and extensions, the system can go several directions. For example, the Android application can be expanded with a dashboard-based UI to include graphs and analytics of historical data and exporting logs of the system. The tunnelling mechanism itself could be replaced with a private server, to avoid limitations associated with third-party tunnelling services like ngrok.

Finally, I believe my thesis managed to introduce a functional prototype of a remotely accessible and modular industrial control system, meeting the project's initial requirements and even beyond. The knowledge and experience I gained throughout the

project’s different phases will help me in further research and development in both industrial automation and mobile IoT applications.

LITERATURE

[1] S. TANELLA, "Android OPCUA Client," 2019. [Online]. Available: https://github.com/SimoneTinella/Android_OPCUA_Client..

[2] J. HURNÍK, “HMI aplikace v prostředí Android komunikující pomocí OPC UA,” Vysoké učení technické v Brně, Brno, 2019.

[3] W. BOLTON, Programmable Logic Controllers, 6th ed., Newnes, 2015, pp. 7-20.

[4] Beckhoff Automation, “Beckhoff Information System,” [Online]. Available: https://infosys.beckhoff.com

[5] Beckhoff Automation, “TwinCAT 3: the flexible software solution for PC-based control,” [Online]. Available: https://download.beckhoff.com/download/document/catalog/Beckhoff_TwinCAT3_e.pdf.

[6] Google for Developers, "Platform architecture," Google, [Online]. Available: https://developer.android.com/guide/platform.

[7] J. GOSLING, B. JOY, G. STEELE, G. BRACHE, A. Buckley, D. Smith and G. Bierman, The Java Language Specification, 2024, [Online]. Available: https://docs.oracle.com/javase/specs/jls/se23/html/index.html

[8] Google For Developers, "Meet Android Studio," Google, [Online]. Available: https://developer.android.com/studio/intro.

[9] statcounter, "Mobile Operating System Market Share Worldwide," [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[10] OPC Foundation, "Unified Architecture – Landingpage," [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/.

[11] OPC Foundation, "OPC 10000-1: UA Part 1: Overview and Concepts," [Online]. Available: https://reference.opcfoundation.org/Core/Part1/v105/docs/.

[12] Beckhoff, "TF6100 | TwinCAT 3 OPC UA," Beckhoff Automation s.r.o., [Online]. Available: https://www.beckhoff.com/en-en/products/automation/twincat/tfxxxx-twincat-3-functions/tf6xxx-connectivity/tf6100.html.

[13] Beckhoff Automation, "TwinCAT 3 | OPC UA Configurator Manual".

[14] GCORE, “What Is Tunneling? Tunneling in Networking,” [Online]. Available: https://gcore.com/learning/what-is-tunneling/.

[15] ngrok (ngrok), “What is ngrok?,” ngrok, [Online]. Available: https://ngrok.com/docs/what-is-ngrok/.

[16] K. KIMANI, “What is ngrok? How does ngrok work?,” sendbird, [Online]. Available: https://sendbird.com/developer/tutorials/what-is-ngrok.

[17] Eclipse Foundation, “Eclipse Milo,” [Online]. Available: https://projects.eclipse.org/projects/iot.milo.

[18] Google for Developers, “AndroidX overview,” Google, [Online]. Available: https://developer.android.com/jetpack/androidx.

[19] E. G. GALLARDO, "What Is MVVM (Model-View-ViewModel)?," 30 September 2024. [Online]. Available: https://builtin.com/software-engineering-perspectives/mvvm-architecture#:~:text=View%2DViewModel)%3F-,MVVM%20(Model%2DView%2DViewModel)%20is%20a%20software%20architectural,to%20test%2C%20maintain%20and%20expand..
[20] OPC Foundation, OPC 10000-2: UA Part 2: Security, P. Hunkar, Ed., 2018.
[21] ngrok, "TLS Certificates," ngrok, [Online]. Available: https://ngrok.com/docs/universal-gateway/tls-certificates/.

SYMBOLS AND ABBREVIATIONS

Abbreviations:

FEEC	Faculty of Electrical Engineering and Communications
BUT	Brno University of Technology
UI	User Interface
UX	User Experience
OS	Operating system
PLC	Programmable Logic Controller
HMI	Human-Machine Interface
PC	Personal Computer
OPC UA	OPC Unified Architecture
POU	Program Organization Unit
GVL	Global Variable List
PRG	Program
IoT	Internet of Things

LIST OF APPENDICES

APPENDIX 1 CONTENTS OF UPLOADED ZIP FILE

/
├──MyApplication.zip..... HMI Android application
│ ├── app
│ │ ├── build..... APK
│ │ ├── libs..... Libraries
│ │ ├── src..... Code
│ │ └── build.gradle ..... Build configuration.
│ └── gradle..... Tool for build
│
└──Tank project……..... TwinCAT 3 project