ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

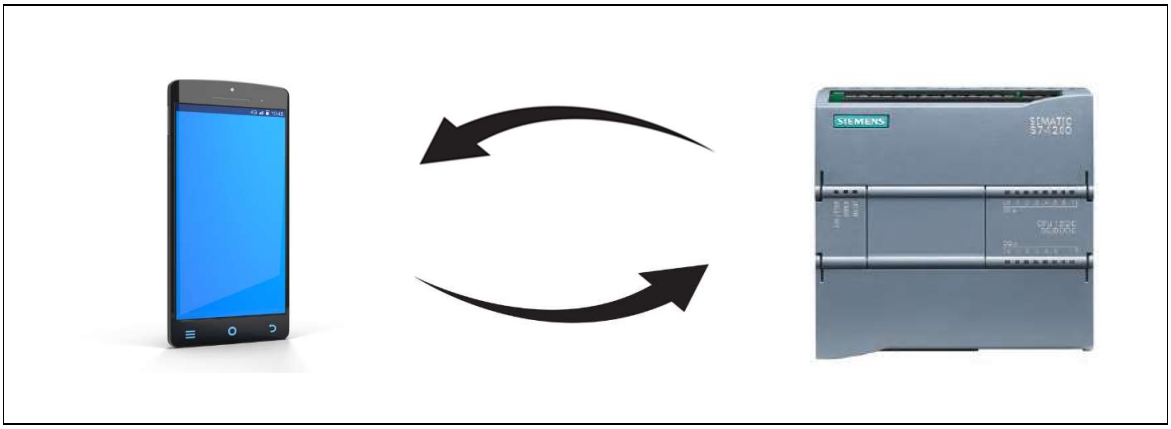
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΕΦΑΡΜΟΣΜΕΝΑ ΗΛΕΚΤΡΟΝΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη SCADA για διαχείριση δεδομένων από PLC σε Android περιβάλλον»



Του φοιτητή
Αποστόλου Βασίλειου
Αρ. Μητρώου: 52003Μ

Επιβλέπων
Ονοματεπώνυμο: Τσαγκάρης Απόστολος
Βαθμίδα: Καθηγητής

Ημερομηνία 20/5/2022

Τίτλος Δ.Ε. Ανάπτυξη SCADA για διαχείριση δεδομένων από PLC σε Android περιβάλλον
Κωδικός Δ.Ε. 21158
Ονοματεπώνυμο φοιτητή Αποστόλου Βασίλειος
Ονοματεπώνυμο εισηγητή Τσαγκάρης Απόστολος
Ημερομηνία ανάληψης Δ.Ε. 20/10/2021
Ημερομηνία περάτωσης Δ.Ε. 20/5/2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Μεταπτυχιακό Πρόγραμμα Σπουδών «Εφαρμοσμένα Ηλεκτρονικά Συστήματα» στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Στους αγαπημένους μου»

Πρόλογος

Στην παρούσα εργασία αναλύεται η επικοινωνία μεταξύ ενός PLC και μιας Android εφαρμογής. Μαζί με αυτήν, παρουσιάζεται και ο τρόπος λειτουργίας κάθε συσκευής ξεχωριστά, το οποίο αποτελεί πρόκληση από μόνο του. Τα οφέλη της ενασχόλησης με αυτό το αντικείμενο, χωρίζονται σε τρία μέρη. Πρώτον, η ενασχόληση με PLC οδήγησε σε βαθύτερη κατανόηση των λειτουργιών και των δυνατοτήτων του. Δεύτερον, ο προγραμματισμός της Android εφαρμογής αποτέλεσε πρόκληση, η οποία οδήγησε στην καλύτερη εξοικείωση με τη γλώσσα προγραμματισμού java, με το περιβάλλον του Android Studio και κατ' επέκταση με το ίδιο το λειτουργικό. Τέλος, η επικοινωνία των δύο ήταν μια ευκαιρία για περαιτέρω έρευνα των πρωτόκολλων επικοινωνίας και πως αυτά πρέπει να προσαρμοστούν σε κάθε περίπτωση. Συνεπώς, επιτεύχθηκε ο στόχος της διπλωματικής εργασίας με την εφαρμογή των γνώσεων, της συνθετικής ικανότητας και της δυνατότητας επίλυσης προβλημάτων.

Περίληψη

Η διπλωματική εργασία πραγματεύεται την υλοποίηση ενός SCADA συστήματος σε Android περιβάλλον, από το οποίο γίνεται η διαχείριση δεδομένων ενός PLC. Αρχικά, γίνεται μια εισαγωγή στις εξελίξεις και τις ανάγκες της βιομηχανίας, η οποίες αφορούν το συγκεκριμένο αντικείμενο. Παρουσιάζονται οι διάφορες μέθοδοι που ακολουθούνται από ερευνητές για την υλοποίηση ενός παρόμοιου συστήματος, τα οφέλη της χρήσης PLC και SCADA συστημάτων και οι κίνδυνοι που κρύβει η απομακρυσμένη επικοινωνία. Έπειτα, αναλύεται η ροή της εργασίας, ο σκοπός της, το κάθε κομμάτι που πρόκειται να αναλυθεί και η γενική λειτουργία του συστήματος, μαζί με χρήσιμες πληροφορίες για την καλύτερη κατανόηση του. Ακολουθεί η εκτενής ανάλυση των λειτουργιών του PLC, οι λεπτομέρειες υλοποίησης του συστήματος, ο τρόπος λειτουργίας και η παρουσίαση των μερών που αποτελούν τον κώδικα. Επίσης, γίνεται αναφορά στην μεταφορά των λειτουργιών σε μια οθόνη ελέγχου και του πίνακα αυτοματισμού που συνοδεύει το σύστημα και των μερών που τον αποτελούν. Παρακάτω, παρουσιάζεται η βασική δομή της Android εφαρμογής με εκτενής ανάλυση του κώδικα που χρειάζεται για τη λειτουργία κάθε μέρους. Γίνεται παράθεση καίριων σημείων του κώδικα μαζί με την κατάλληλη επεξήγηση. Τέλος, παρουσιάζονται τα συμπεράσματα που εξάγονται από την υλοποίηση του συστήματος, τα οφέλη της εργασίας και οι προοπτικές βελτίωσης.

«Development of SCADA control system in Android environment for data management from PLC»

«Apostolou Vasileios»

Abstract

The main purpose of this thesis is to develop and implement a method of communication between a PLC and an Android application, which remotely controls a system used for liquid food storage and distribution. After a quick review of the main topics of the thesis, the different approaches of SCADA communication and the benefits of the said methods along with the ever-present risks of the remote communication, the workflow of the PLC is presented. Firstly, after the explanation of the various Function and Data Blocks used, the main ladder diagram is presented. Every Network is discussed in detail, explaining its role in the overall program flow, including the HMI configuration settings. Then, the flow chart of the ESP32 code is presented, along with the functions and the amplifier's circuit used for the analog input. The electrical schematic diagram and the basic functions of the automation electrical board are presented, leading to conclusions of the second chapter. In the next chapter follows the analysis of the development of the android application. Flow charts and code samples from each activity are presented for a better understanding of the different aspects of the application, along with detailed analysis. Finally, the conclusions, the possible developments, and the different kinds of use of the overall application are listed and discussed in the final chapter.

Ευχαριστίες

Με την ολοκλήρωση της εργασίας θα ήθελα να ευχαριστήσω τον Γιάννη Κατρανίτσα για την πολύτιμη βοήθειά του σχετική με θέματα του PLC, τον Βασίλειο Καρβωνίδη για τη συνεχή στήριξη του σε θέματα σχετικά με τη γλώσσα Java και τον Νικόλαο Τόπη για την διάθεση του εξοπλισμού του έτσι ώστε να γίνουν δοκιμές απομακρυσμένου ελέγχου του συστήματος.

Περιεχόμενα

Πρόλογος vi
Περίληψη vii
Abstract viii
Ευχαριστίες ix
Περιεχόμενα x
Κατάλογος Σχημάτων xii
Κατάλογος Εικόνων xii
Συντομογραφίες xiv
Κεφάλαιο 1ο: Εισαγωγή και Βασικές Λειτουργίες 1
1.1 Εισαγωγή 1
1.2 Προγραμματιζόμενος Λογικός Ελεγκτής (PLC) 1
1.3 Μέθοδοι εφαρμογής SCADA 4
1.4 Περιγραφή βασικής λειτουργίας 7
Κεφάλαιο 2ο: Περιγραφή του Αυτοματισμού 12
2.1 Εισαγωγή 12
2.2 Αρχές Λειτουργίας 12
2.2.1 Θεωρητική Ανάλυση 12
2.2.2 Βασικές Λειτουργίες 13
2.2.3 Ανάλυση Τρόπου Λειτουργίας 15
2.2.4 Ανάλυση HMI (Human Machine Interface) και του Πίνακα Αυτοματισμού 23
2.3 Επίλογος 30
Κεφάλαιο 3ο: Περιγραφή της Android Εφαρμογής 31
3.1 Εισαγωγή 31
3.2 Αρχές Λειτουργίας 31
3.2.1 Δομή και Βασική Ροή 31
3.2.2 Ανάλυση Λειτουργίας 31
3.3 Επίλογος 50
Κεφάλαιο 4ο: Συμπεράσματα 51
ΒΙΒΛΙΟΓΡΑΦΙΑ 53
ΠΑΡΑΡΤΗΜΑ Α : ΔΙΑΓΡΑΜΜΑΑ LADDER 56
ΠΑΡΑΡΤΗΜΑ Β : ΚΩΔΙΚΑΣ ANDROID ΕΦΑΡΜΟΓΗΣ 68

ΠΑΡΑΡΤΗΜΑ Γ : ΚΩΔΙΚΑΣ ESP32.....110
ΠΑΡΑΡΤΗΜΑ Δ : ΗΛΕΚΤΡΟΛΟΓΙΚΟ ΣΧΕΔΙΟ ΠΙΝΑΚΑ.....115

Κατάλογος Σχημάτων

Σχήμα 2.1: Το διάγραμμα ροής της Ladder..... 14
Σχήμα 3.1: Το διάγραμμα ροής του Login Activity.....33
Σχήμα 3.2: Το διάγραμμα ροής του IP Activity.....35
Σχήμα 3.3: Το διάγραμμα ροής του Read/Write Activity.....38
Σχήμα 3.4: Το διάγραμμα ροής του I/O Control Activity.....38
Σχήμα 3.5: Το διάγραμμα ροής του Emergency Stop Triggered Acitivity.....47
Σχήμα 3.6: Το διάγραμμα ροής του Settings Activity.....49

Κατάλογος Εικόνων

Εικόνα 1.1: Το εργαλείο DUC.....9
Εικόνα 1.2: Η επιλογή "Edit Hosts".....9
Εικόνα 1.3: Η ρύθμιση του DDNS.....10
Εικόνα 1.4: Η ρύθμιση Port Forwarding.....11
Εικόνα 2.1: Τα Program Blocks του project.....15
Εικόνα 2.2: Το Data Block που περιέχεται μέσα στο FC1.....16
Εικόνα 2.3: Το Data Block που περιέχεται μέσα στο FC2.....17
Εικόνα 2.4: Η επιλογή στην καρτέλα "Access Level".....17
Εικόνα 2.5: Η επιλογή στην καρτέλα "Connection Mechanism".....18
Εικόνα 2.6: Η λειτουργία του FC3.....18
Εικόνα 2.7: Η λειτουργία του FC3 μέσα στο Main Block.....19
Εικόνα 2.8: Η προσομοίωση της HMI.....23
Εικόνα 2.9α: Η ρύθμιση "SetBit".....24
Εικόνα 2.9β: Η ρύθμιση "ResetBit".....24
Εικόνα 2.10α: Η ρύθμιση για τη λήψη της θερμοκρασίας στο αντίστοιχο πεδίο.....24
Εικόνα 2.10β: Η ρύθμιση του ενδεικτικού της στάθμης υγρού.....25
Εικόνα 2.10γ: Η ρύθμιση του ενδεικτικού της κατάστασης ψύξης.....25
Εικόνα 2.11: Η ρύθμιση του πεδίου "Temperature Point".....25
Εικόνα 2.12: Το αισθητήριο θερμοκρασίας DHT11 και η ένδειξη θερμοκρασίας και υγρασίας.....27
Εικόνα 2.13: Η συνάρτηση για την ανάγνωση θερμοκρασίας.....27
Εικόνα 2.14: Η συνάρτηση για την ανάγνωση της υγρασίας.....28
Εικόνα 2.15: Ο κώδικας για τον υπολογισμό και τη μετάδοση της αναλογικής τιμής.....29
Εικόνα 2.16: Ο συνδεσμολογία του κυκλώματος στο PSPICE.....29
Εικόνα 2.17: Η έξοδος της ενισχυτικής βαθμίδας σε σχέση με την είσοδό της.....29
Εικόνα 2.18: Ο πίνακας αυτοματισμού.....30
Εικόνα 3.1: Τα πεδία του "LoginActivity".....32
Εικόνα 3.2: Ο κώδικας της ρουτίνας "validate".....32
Εικόνα 3.3: Η εκτέλεση της "validate" με το πάτημα του "LOGIN".....32
Εικόνα 3.4: Το πεδίο εισαγωγής της IP διεύθυνσης.....33
Εικόνα 3.5: Ο κώδικας που εκτελείται πιέζοντας το "CONNECT".....34
Εικόνα 3.6: Η αποθήκευση της διεύθυνσης IP.....34
Εικόνα 3.7: Το κομμάτι κώδικα που αφορά την ανάγνωση πληροφοριών από το PLC.....36

Εικόνα 3.8: Ο κώδικας που αφορά την εμφάνιση δεδομένων στα αντίστοιχα πεδία.....37
Εικόνα 3.9: Ο κώδικας για την αποστολή του setpoint θερμοκρασίας προς το PLC.....37
Εικόνα 3.10: Το "Read/WriteActivity".....39
Εικόνα 3.11: Έλεγχος της κατάστασης του στοπ έκτακτης ανάγκης (Emergency Stop) 40
Εικόνα 3.12: Ο κώδικας ελέγχου κατάστασης εξόδων PLC και στάθμης δεξαμενής.....40
Εικόνα 3.13: Οι ενέργειες του κώδικα μετά τον αρχικό έλεγχο.....41
Εικόνα 3.14α: Οι ενδείξεις σε κανονική λειτουργία.....42
Εικόνα 3.14β: Η ένδειξη χαμηλής στάθμης.....43
Εικόνα 3.15α: Η λειτουργία του κώδικα που αφορά την εκκίνηση του αναδευτήρα.....44
Εικόνα 3.15β: Η επαναφορά της κατάστασης του "START" από "TRUE" σε "FALSE".....45
Εικόνα 3.16: Ο κώδικας ο οποίος αφορά τη λειτουργία του Emergency Stop.....46
Εικόνα 3.17: Η οθόνη στο "ESTriggeredActivity".....46
Εικόνα 3.18: Ο κώδικας που ελέγχει την επαναφορά του φυσικού Emergency Stop.....47
Εικόνα 3.19: Ο κώδικας που επαναφέρει τη θέση του Emergency Stop στο PLC σε "FALSE".....48
Εικόνα 3.20: Τα πεδία στο "SettingsActivity".....49
Εικόνα 3.21: Ο κώδικας του "SettingsActivity".....50

Συντομογραφίες

DDNS	Dynamic Domain Name Server
PLC	Programmable Logic Controller – Προγραμματιζόμενος Λογικός Ελεγκτής
LD	Ladder Diagram
SFC	Sequential Function Chart
ST	Structured Text
IL	Instruction List
HMI	Human Machine Interface
SCADA	Supervisory Control And Data Acquisition
VPN	Virtual Private Network
TIA Portal	Total Integrated Automation Portal
FC	Function Block
DB	Data Block
°C	Βαθμοί Κελσίου

Κεφάλαιο 1ο: Εισαγωγή και Βασικές Λειτουργίες

1.1 Εισαγωγή

Η εξέλιξη της βιομηχανίας και του αυτοματισμού των διαδικασιών της, προσφέρουν τη δυνατότητα της μαζικής και ταχύτερης παραγωγής εδώ και αρκετά χρόνια. Οι λύσεις που προσφέρονται αφορούν το γενικό σύνολο της βιομηχανίας κι έτσι δεν έχει σημασία ο τομέας της εκάστοτε βιομηχανίας, καθώς οι λύσεις αυτές μπορούν να προσαρμοστούν και να εφαρμοστούν εξίσου σε κάθε ανάγκη. Αυτές οι λύσεις και εφαρμογές μπορούν να αφορούν την διαδικασία αυτοματισμού της παραγωγής, την βελτίωση της ταχύτητας της, την πρόβλεψη και την επίβλεψη (monitoring) του συνόλου για πιθανές βλάβες και την πρόληψή τους. Το τελευταίο κομμάτι της επίβλεψης, πρόληψης και καταγραφής, αλλά και η γενική διαδικασία του αυτοματισμού, είναι θέματα τα οποία αναλύονται και παρουσιάζονται σε αυτήν τη διπλωματική εργασία, με την παρουσίαση ενός μοντέλου το οποίο καταγράφει και επιστρέφει μέσω Android εφαρμογής, μετρήσεις ενός PLC (Programmable Logic Controller). Επίσης, επιτρέπει την εξ' αποστάσεως εκκίνηση και παύση διάφορων λειτουργιών ενός συστήματος αυτοματισμού, λεπτομέρειες του οποίου θα αναλυθούν παρακάτω σε αυτό το κεφάλαιο.

Παρακάτω αναλύεται η λειτουργία ενός PLC και η ανάπτυξη αυτού του συστήματος, το οποίο αποτελεί μια μορφή SCADA (Supervisory Control And Data Acquisition). Το SCADA παίζει κρίσιμο ρόλο στην επίβλεψη και τον ορθό έλεγχο της παραγωγής, εξασφαλίζοντας ταχύτητα και ασφάλεια στη συμπεριφορά του συστήματος, ενώ ταυτόχρονα εκτελεί τις ενέργειες για τις οποίες έχει προγραμματιστεί με ταχύτητα και ακρίβεια. Αποτελεί έναν τομέα ο οποίος συνεχώς εξελίσσεται, κυρίως για τη μείωση του κόστους εφαρμογής του, την επίτευξη μικρότερων ταχυτήτων και μεγαλύτερης αποδοτικότητας. Ένας σημαντικός παράγοντας αποτελεί και η ασφαλής επικοινωνία που εξασφαλίζει μεταξύ των συσκευών.

1.2 Προγραμματιζόμενος Λογικός Ελεγκτής (PLC)

Η εφαρμογή των προγραμματιζόμενων λογικών ελεγκτών στις βιομηχανικές εφαρμογές άλλαξε εξ' ολοκλήρου τις διαδικασίες της. Αρχικά, η υλοποίηση συστημάτων αυτοματισμού γινόταν αποκλειστικά με τη χρήση ηλεκτρονόμων (Relays), χρονικών (Timers) και διάφορων άλλων εξαρτημάτων, τα οποία χρησιμοποιούνται ακόμη και σήμερα. Ωστόσο, ένα PLC μπορεί να προσομοιώσει τη λειτουργία αυτών των ηλεκτρονόμων και των χρονικών και μαζί άλλες λειτουργίες. Ενώ δηλαδή στην πρώτη περίπτωση μπορεί να απαιτούνταν δεκάδες ή εκατοντάδες εξαρτήματα για να υλοποιηθεί ένα σύστημα αυτοματισμού, ανάλογα τις απαιτήσεις κάθε περίπτωσης, σχεδόν όλα αυτά μπορούν να προσομοιωθούν με ένα PLC το οποίο μπορεί να προγραμματιστεί ανάλογα. Αυτό προσφέρει ταχύτητα, εξοικονόμηση χώρου και χρόνου, αλλά και σταθερότητα. Ένας τέτοιος ελεγκτής αποτελείται από μια κεντρική μονάδα επεξεργασίας (CPU), από ψηφιακές εισόδους και εξόδους, ενώ μπορούν να προστεθούν και επιπλέον κάρτες εισόδων και εξόδων για την επέκταση του. Επίσης, έχουν αναπτυχθεί και διάφορες κάρτες οι οποίες εξυπηρετούν ειδικούς σκοπούς, όπως η διαχείριση αναλογικών σημάτων, τα οποία μπορούν να είναι αναλογικών εισόδων ή εξόδων, ή σημάτων υψηλής ταχύτητας, όπως οι κάρτες (high-speed) οι οποίες χρησιμοποιούνται για τη λήψη και επεξεργασία του σήματος που προέρχεται από ένα resolver. [23][25][27]

Ο προγραμματισμός ενός PLC απαιτεί ένα περιβάλλον υπολογιστή, το οποίο εξαρτάται από τον κατασκευαστή της συσκευής. Ενδεικτικά, ένα PLC της Siemens μπορεί να προγραμματιστεί μέσω του TIA Portal, ενώ ένα αντίστοιχο της Allen & Bradley, μέσω του RSLogix 5000. Ωστόσο, η γλώσσα

προγραμματισμού δε διαφέρει, ενώ υπάρχουν πέντε επιλογές γλώσσας προγραμματισμού, σύμφωνα με τον κανονισμό IEC 61131-3. Αυτή μπορεί να είναι είτε η Ladder ή η FBD (Function Block Diagram), είτε η SFC (Sequential function chart), η Structured Text (ST) ή η Instruction List (IL). Η Ladder συνήθως προτιμάται, διότι αποτελείται από σύμβολα, παρόμοια με αυτά ενός ηλεκτρολογικού σχεδίου, κάτι το οποίο κάνει πιο εύκολη την κατανόηση της. Χρησιμοποιήθηκε αρχικά επειδή ακριβώς μοιάζει τόσο με ένα ηλεκτρολογικό σχέδιο και ήταν πιο άμεση και γρήγορη η επαφή που αποκτούσαν όσοι είχαν γνώσεις ηλεκτρολογικού σχεδίου και κλασσικού αυτοματισμού. Ωστόσο, έχει και αυτή κάποιο βαθμό δυσκολίας, καθώς πέρα από τα σύμβολα, χρησιμοποιούνται και blocks τα οποία απαιτούν πολλές φορές αρκετές και περίπλοκες ρυθμίσεις. [25]

Το διάγραμμα Ladder μοιάζει με αυτό του ηλεκτρολογικού σχεδίου, όπως αναφέρθηκε και χρησιμοποιεί κι αυτό κανονικά ανοικτές (Normally Open – NO), κανονικά κλειστές (Normally Closed – NC) επαφές και πηνία (Coils), ανάμεσα σε πολλά άλλα, τα οποία συνδέονται παράλληλα ή σε σειρά. Επίσης, δίνει την επιλογή χρήσης χρονικών, τα οποία μπορούν να αφορούν καθυστέρηση κατά την ενεργοποίηση τους ή την απενεργοποίηση τους και άλλα. Ακόμα, παρέχει έτοιμα μπλοκ συναρτήσεων, τα οποία εκτελούν διάφορες λειτουργίες, όπως τη λήψη και επεξεργασία αναλογικών τιμών, ή PID έλεγχο.

Η χρήση των PLC είναι ευρεία και αφορά πάρα πολλές εφαρμογές της βιομηχανίας. Μια τέτοια συσκευή μπορεί να χρησιμοποιηθεί για τον έλεγχο και τη λειτουργία ενός συστήματος το οποίο συσκευάζει τρόφιμα, όπως το ρύζι. Πρώτα πρέπει να γίνει το ζύγισμά του, το οποίο γίνεται από κυψέλες και το PLC, ανάλογα την αναλογική τιμή που λαμβάνει και μετατρέπει, προβαίνει στις αντίστοιχες ενέργειες. Δηλαδή, μπορεί να ενεργοποιήσει ένα πνευματικό σύστημα, μέσω του οποίου θα ανοίξει το δοχείο ζύγισης και το ρύζι θα πέσει μέσα στο σακουλάκι. Αυτό με τη σειρά του θα σφραγιστεί με δύο πλάκες, οι οποίες έχουν την κατάλληλη θερμοκρασία για να λιώσουν το πλαστικό, τόσο όσο χρειάζεται, ώστε να σφραγιστεί η συσκευασία, χωρίς να καταστραφεί το πλαστικό. Η θερμοκρασία ελέγχεται από το PLC, ανάλογα τη ρύθμιση που δίνει ο χρήστης. Τέλος, ένα μαχαίρι ανάμεσα στις πλάκες θα κόψει το κολλημένο πλαστικό από το υπόλοιπο ρολό, διαμορφώνοντας έτσι μια έτοιμη συσκευασία ρυζιού, η οποία είναι έτοιμη να προχωρήσει στο επόμενο στάδιο της συσκευασίας.

Ένα ακόμα παράδειγμα αφορά τη ζήτηση αέρα από ένα δίκτυο αέρα σε μια βιομηχανία. Η λειτουργία του αεροσυμπιεστή καθορίζεται από έναν ελεγκτή, ο οποίος συγκρίνει την ζητούμενη πίεση, η οποία εισάγεται από το χρήστη, με την πραγματική, η οποία μετριέται από έναν αναλογικό αισθητήρα πίεσης, ο οποίος είναι τοποθετημένος στο κατάλληλο σημείο του δικτύου. Η σύγκριση οδηγεί τον ελεγκτή να προβεί σε ενέργειες. Αν η πραγματική πίεση είναι χαμηλότερη από τη ζητούμενη, τότε ξεκινάει τον αεροσυμπιεστή, ή αυξάνει τις στροφές του κινητήρα, αν πρόκειται για αεροσυμπιεστή του οποίου οι στροφές ελέγχονται από inverter. Αν στην αντίθετη περίπτωση η πραγματική πίεση είναι μεγαλύτερη από τη ζητούμενη, τότε η λειτουργία προσαρμόζεται ανάλογα. Οι χρήσεις των PLC μπορούν να αφορούν πιο περίπλοκες εφαρμογές, όπως ο αυτοματισμός της διαδικασίας της παραγωγής ψωμιού, από την αυτόματη ζύμωση του, μέχρι τη συσκευασία του. Αυτό προϋποθέτει το συγχρονισμό όλων των συσκευών για την επίτευξη της συνεχούς και καλής ροής της γραμμής, έτσι ώστε να επιτευχθεί και η καλύτερη δυνατή ταχύτητα παραγωγής σε συνδυασμό με την καλύτερη δυνατή ποιότητα προϊόντος. [25][27]

Για να χρησιμοποιηθεί ένας προγραμματιζόμενος λογικός ελεγκτής σε διάφορους τομείς της βιομηχανίας, θα πρέπει να έχει κάποια χαρακτηριστικά. Οι απαιτήσεις ανάμεσα σε πολλές, είναι το να είναι γρήγορος, εύκολος στον προγραμματισμό του, στη χρήση του και ανθεκτικος σε αντίξοες

συνθήκες. Οι απαιτήσεις αυτές είναι δικαιολογημένες, καθώς οι εφαρμογές μπορεί να αφορούν είτε απλές εφαρμογές, αλλά και κρίσιμες εφαρμογές, όπως η διαχείριση καυσίμων, στρατιωτικές εγκαταστάσεις, ή μονάδες ηλεκτροδότησης ή ελέγχου ποιότητας νερού.

Η αντοχή των ελεγκτών σε αντίξοες συνθήκες εργασίας, είναι ένα από τα σημαντικότερα χαρακτηριστικά. Επίσης, εφόσον προσομοιώνουν τη λειτουργία ηλεκτρονόμων, χρονικών και άλλων εξαρτημάτων, εξοικονομούν χώρο και μειώνεται η πιθανότητα παρουσίας βλαβών, καθώς δε γίνεται χρήση τέτοιων εξαρτημάτων, τα οποία αποτελούνται από μηχανικά μέρη. Ακόμα ένα χαρακτηριστικό πλεονέκτημα των PLC είναι η ευελιξία που προσφέρουν στον προγραμματισμό τους. Ο χρήστης έχει τη δυνατότητα να επιλέξει ανάμεσα σε πέντε διαφορετικές γλώσσες προγραμματισμού, όπως αναφέρθηκε, ενώ χρειάζεται να γνωρίζει έστω μια, για να μπορέσει να προγραμματίσει ένα PLC. Επίσης, η εύκολη διαχείριση σφαλμάτων είναι ένα χαρακτηριστικό τέτοιων ελεγκτών, καθώς μαζί με το κυρίως πρόγραμμα, εκτελούνται και διαγνωστικά, τα οποία επιτηρούν την ορθή λειτουργία του. Η ταχύτητα που εκτελείται το κυρίως πρόγραμμα παίζει επίσης σημαντικό ρόλο. Όπως είναι λογικό, εφόσον απαιτούνται μικροί χρόνοι αντίδρασης, τότε ο κύκλος του προγράμματος πρέπει να είναι μικρός. Αυτό εξαρτάται από τις δυνατότητες του επεξεργαστή κάθε συσκευής και από το μέγεθος του προγράμματος. Ωστόσο ο χρόνος είναι της τάξης των milliseconds (ms), και ο κύκλος ξεκινάει με τη σάρωση των εισόδων, την εκτέλεση του προγράμματος και των διαγνωστικών και την ανανέωση των εξόδων, ανάλογα. Τα παραπάνω πλεονεκτήματα, μαζί με άλλα, όπως η μικρή κατανάλωση ισχύος, η εκτέλεση περίπλοκων λογικών πράξεων και ο μεγάλος όγκος πληροφορίας γύρω από τον προγραμματισμό και τη διαχείριση προγραμματιζόμενων λογικών ελεγκτών, έχουν κάνει τα PLC πρωταγωνιστές στην παγκόσμια βιομηχανία. Αποτελούν ελεγκτές οι οποίοι μπορούν να προσαρμοστούν σε κάθε ανάγκη, είναι ευέλικτοι, ανθεκτικοί και εύκολοι στη διαχείριση τους και για αυτό έχουν συμβάλει στην εξέλιξη της βιομηχανίας, με ότι αυτό συνεπάγεται. [23][25][27]

Από την άλλη πλευρά, παρά τα ξεκάθαρα πλεονεκτήματα των προγραμματιζόμενων λογικών ελεγκτών, δεν παύουν να υπάρχουν και μειονεκτήματα. Αυτά μπορούν να αφορούν τη συσκευή και τους περιορισμούς που μπορεί να παρουσιάζει, όπως επίσης και τις εφαρμογές που χρησιμοποιείται. Η λειτουργία ενός PLC δε διαφέρει κατά πολύ από αυτή ενός συμβατικού ηλεκτρονικού υπολογιστή (PC). Η χρήση και των δύο είναι αναγκαία στη βιομηχανία, παρόλα αυτά υπάρχουν κάποιες βασικές διαφορές στον τρόπο που εφαρμόζονται. Ωστόσο, ενώ το PLC είναι ένα από τους κύριους παράγοντες της σωστής λειτουργίας ενός συστήματος αυτοματισμού, σε σύγκριση με έναν υπολογιστή είναι κατώτερο. Πρώτον, η συνεχής ανανέωση και αναβάθμιση των ηλεκτρονικών υπολογιστών, έχουν καταστήσει δυνατές μεγάλες επεξεργαστικές δυνατότητες και αρκετά μεγάλη χωρητικότητα μνήμης, κάτι που δε συναντάται στην πλευρά του PLC. Επίσης, δεν είναι απαραίτητο πως η χρήση ενός PLC αποδεικνύεται αποτελεσματική σε ένα σύστημα. Πιο συγκεκριμένα, σε μελέτη που έγινε το 2013 και παρατίθεται στη βιβλιογραφία [34], αποδεικνύεται πως η χρήση ενός PLC για ένα σύστημα για την καλύτερη αποδοτικότητα του δημοτικού δικτύου νερού στην Καλιφόρνια, δεν ήταν αποδοτική. Αντίθετα, φαίνεται πως η χρήση του μπορούσε να προκαλέσει μεγαλύτερη φθορά στην αποδοτικότητα του συστήματος, παρά να τη βελτιώσει. Επίσης, σε διαφορετική μελέτη, η οποία παρατίθεται στη βιβλιογραφία [35] και αφορά την ανάπτυξη μιας μικρής ανεμογεννήτριας, ο συγγραφέας καταλήγει στο συμπέρασμα πως ενώ υπήρχε η σκέψη της χρήσης PLC, επειδή από τη μια η απαιτούμενη επεξεργαστική δύναμη για το σύστημα είναι μικρή και από την άλλη το κόστος ενός τέτοιου ελεγκτή υψηλό, προτίμησε τη χρήση ενός μικροελεγκτή PIC16F877. [23][34][35]

1.3 Μέθοδοι εφαρμογής SCADA

Ένα σύστημα SCADA αποτελείται από πολλά υποσυστήματα, τα οποία ελέγχονται το κάθε ένα ξεχωριστά, από ξεχωριστό ελεγκτή. Ωστόσο, είναι πιθανό να υπάρχει επικοινωνία μεταξύ των συσκευών, μέσω του SCADA ή μέσω απευθείας σύνδεσης μεταξύ τους. Όταν πρόκειται για ένα μεγάλο σύστημα, το οποίο ελέγχεται από ένα κεντρικό σημείο, τότε το δεύτερο δεν προτείνεται. Δηλαδή, όλες οι επικοινωνίες πρέπει να καταλήγουν στο ίδιο SCADA, για την καλύτερη λειτουργία. Κάθε υποσύστημα, πρέπει να περιέχει έναν ελεγκτή PLC, μια κεντρική μονάδα δηλαδή, στην οποία καταλήγουν όλα τα σήματα και συνδέονται όλες οι συσκευές που ανήκουν στο υποσύστημα. Ο ελεγκτής και οι διάφορες αυτές συσκευές επικοινωνούν μεταξύ τους μέσω διάφορων βιομηχανικών πρωτοκόλλων επικοινωνίας, τα οποία έχουν αναπτυχθεί ειδικά για αυτού του είδους τις εφαρμογές, όπως το PROFIBUS. Ταυτόχρονα, το PLC επικοινωνεί με τον κεντρικό έλεγχο, στέλνοντας απαραίτητα και χρήσιμα δεδομένα. Άλλωστε, όπως αναφέρθηκε και υποδεικνύει το όνομα του, ένα SCADA σύστημα είναι εκείνο που εκτελεί τον κεντρικό έλεγχο και τη συγκέντρωση και διαχείριση δεδομένων. Ένα σύστημα SCADA μπορεί να εφαρμοστεί σχεδόν σε όλους τους τομείς της βιομηχανίας, όπως στη βιομηχανία τροφίμων, στις τηλεπικοινωνίες, στη διανομή αερίου, ακόμα και σε στρατιωτικές υποδομές. [20][29]

Ο ελεγκτής που χρησιμοποιείται σε ένα SCADA μπορεί να είναι ένας προγραμματιζόμενος λογικός ελεγκτής (PLC). Μέσω αυτού του ελεγκτή και με βάση τον κώδικα που περιέχει, καθορίζονται όλες οι λειτουργίες του συστήματος. Κάθε συσκευή μπορεί να αποτελεί μια είσοδο ή μια έξοδο του και να αποστέλλει ή να λαμβάνει αντίστοιχα το κατάλληλο σήμα από το PLC. Επίσης, η επικοινωνία μεταξύ των συσκευών μπορεί να υλοποιηθεί μέσω του PLC και αποτελεί ένα κρίσιμο κομμάτι της λειτουργίας του συστήματος. Η επικοινωνία παρέχει και αποστέλλει χρήσιμες πληροφορίες ανάμεσα στον ελεγκτή και τις συσκευές που περιέχονται στο σύστημα, ενώ ταυτόχρονα μπορεί να παρέχει χρήσιμες πληροφορίες στο χρήστη για την κατάσταση του συστήματος, για πιθανές βλάβες ή προειδοποιήσεις και επιτρέπει στον ίδιο να προβεί σε αλλαγές και ρυθμίσεις, για την επίτευξη της επιθυμητής λειτουργίας. Συμπεραίνεται λοιπόν πως η επικοινωνία αποτελεί το σημαντικότερο μέρος ενός συστήματος αυτοματισμού. Οι τρόποι επικοινωνίας διαφέρουν, ενώ έχουν αναπτυχθεί συγκεκριμένα βιομηχανικά πρωτόκολλα. Ακόμη, έχει γίνει δυνατή η απομακρυσμένη επίβλεψη και ρύθμιση ενός συστήματος, ακόμα και από συμβατικές συσκευές, όπως ένα κινητό τηλέφωνο. Αυτό έχει δώσει ακόμη μεγαλύτερη ευελιξία και ταχύτητα, ωστόσο περιέχει περισσότερους κινδύνους. Ενώ παλαιότερα συστήματα SCADA επικοινωνούσαν μέσω ενός απομονωμένου δικτύου, τα σημερινά χρησιμοποιούν και το διαδίκτυο για την επικοινωνία τους. Συνεπώς, τα παλαιότερα συστήματα ήταν καλύτερα προστατευμένα από εξωτερικές απειλές, κάτι που δε συμβαίνει στα νεότερα συστήματα τα οποία χρησιμοποιούν το διαδίκτυο, καθώς εκτίθενται σε κινδύνους στους οποίους εκτίθεται οποιαδήποτε συσκευή συνδέεται στο διαδίκτυο. Η κρισιμότητα ορισμένων εφαρμογών ωστόσο, όπως ο έλεγχος ηλεκτροδότησης, υδροδότησης ή ακόμα και μια στρατιωτική υποδομή, κάνουν αναγκαία την ανάπτυξη πρωτόκολλων προστασίας τέτοιων συστημάτων. Για αυτόν το λόγο, παρουσιάζονται συνεχώς, από ερευνητές και από εταιρείες, νέες έρευνες και μέθοδοι οι οποίες προτείνονται για τη καλύτερη και μεγαλύτερη προστασία τέτοιων συστημάτων. Η σύνδεση στο διαδίκτυο ωστόσο, αλλά ακόμα και άλλες τεχνολογίες επικοινωνίας, όπως αυτή μέσω Bluetooth, παρά τους κινδύνους, έχουν καταστήσει δυνατή την πιο γρήγορη και άμεση την επικοινωνία μεταξύ συσκευών και χρηστών. [26][29]

Οι προσπάθειες όλων όσων ασχολούνται με τη βελτίωση της τεχνολογίας των συστημάτων SCADA, αφορούν τη μείωση του κόστους και τους τρόπους επικοινωνίας. Σύμφωνα με πηγές που παρατίθενται στη Βιβλιογραφία, ένας από τους τρόπους ανάπτυξης ενός SCADA για απομακρυσμένο

έλεγχο, μπορεί να αφορά την ανάπτυξη συστημάτων, τα οποία χρησιμοποιούν την τεχνολογία Bluetooth και μέσω ενός μικροελεγκτή (Arduino Uno, ESP32 και άλλους) μπορούν να μεταδώσουν σε μια Android συσκευή και να λάβουν πληροφορία από αυτή, η οποία μπορεί να επικοινωνεί μέσω εξατομικευμένης εφαρμογής. Η εφαρμογή μπορεί να έχει υλοποιηθεί με τη χρήση κώδικα, ή μέσω έτοιμων εργαλείων, τα οποία απαιτούν σε μεγαλύτερο βαθμό ρυθμίσεις παρά κώδικα. Επίσης, οι μικροελεγκτές χρησιμοποιούνται και για τον έλεγχο ηλεκτρονόμων, οι οποίοι αποτελούν τις εισόδους ή τις εξόδους του συστήματος, ανάλογα την εφαρμογή. Η χρήση του Bluetooth ωστόσο περιορίζει τη δυνατότητα ελέγχου από μεγάλη απόσταση. Επίσης, ένα τέτοιο σύστημα μπορεί να βασίζεται σε έναν web server, μέσω του οποίου μεταδίδονται και λαμβάνονται πληροφορίες και ο οποίος μπορεί να επικοινωνεί είτε μόνο μέσα στο τοπικό δίκτυο, είτε απομακρυσμένα από άλλο δίκτυο ή και τα δύο. Ακόμα, η ανάπτυξη τέτοιων εφαρμογών μπορεί να γίνει και με το συνδυασμό των παραπάνω. Δηλαδή, μπορεί να γίνει χρήση ενός web server, ενός ελεγκτή και μιας android εφαρμογής, με την επικοινωνία των δύο να είναι δυνατή και εντός και εκτός τοπικού δικτύου, με την κατάλληλη υποδομή. [10][11][12][32][33]

Η απομακρυσμένη επικοινωνία αποτελεί έναν πολύ σημαντικό παράγοντα στην ανάπτυξη ενός SCADA. Οι μέθοδοι που εφαρμόζονται έχουν, όπως είναι λογικό, κάποια πλεονεκτήματα και κάποια μειονεκτήματα. Πιο συγκεκριμένα, η πρώτη μέθοδος που αναφέρθηκε στην προηγούμενη παράγραφο και αφορά τη χρήση Bluetooth και μικροελεγκτή, αποτελεί μια λύση χαμηλού κόστους και απλής υλοποίησης. Ωστόσο, ο χρήστης περιορίζεται από την επιτρεπόμενη απόσταση επικοινωνίας, η οποία είναι μικρή, καθώς πρόκειται για Bluetooth τεχνολογία, και από τις δυνατότητες του μικροελεγκτή, ειδικά αν πρόκειται για κάποιους οι οποίοι χρησιμοποιούνται σε μικρά project, όπως το Arduino Uno. Ακόμα, οι δυνατότητες της εφαρμογής μπορούν επίσης να περιορίσουν την απόδοση του συστήματος, ειδικά αν πρόκειται για έτοιμο κώδικα και ρυθμίσεις. Παρόλα αυτά, αποτελεί μια λύση για εφαρμογές χωρίς ιδιαίτερες απαιτήσεις ή για την υλοποίηση υποσυστημάτων, τα οποία δεν αποτελούν μέρος του κύριου συστήματος, αλλά επικοινωνούν μαζί του, ανταλλάζοντας πληροφορίες. [10][22][24]

Η δεύτερη μέθοδος χρησιμοποιείται σε μεγάλο βαθμό και αποτελεί ίσως την αμέσως επόμενη καλύτερη λύση, καθώς η χρήση ενός web server παρέχει πλεονεκτήματα τα οποία δε μπορούν να υπάρξουν στην πρώτη λύση. Το κυριότερο είναι η απόσταση, καθώς είτε πρόκειται για επικοινωνία σε επίπεδο τοπικού ή απομονωμένου δικτύου, είτε για απομακρυσμένο έλεγχο μέσω διαδικτύου, τότε αυτός μπορεί να γίνει μέσω ηλεκτρονικού υπολογιστή (Η/Υ) και της ιστοσελίδας που μπορεί να έχει αναπτυχθεί από τον χρήστη ή παρέχεται έτοιμη από τους ίδιους τους κατασκευαστές. Οι δυνατότητες εξαρτώνται από τη συσκευή ελέγχου (PLC) και από τη δυνατότητα σταθερής σύνδεσης. Είναι μια μέθοδος η οποία χρησιμοποιείται ευρέως σε βιομηχανικές εγκαταστάσεις, στις οποίες όλο το SCADA φαίνεται στην αντίστοιχη σελίδα που έχει υλοποιηθεί και μπορεί να χειριστεί μέσω αυτής. [11][22][24]

Ωστόσο, η επιλογή της android εφαρμογής είναι αυτή που μπορεί να κριθεί ως η επικρατέστερη πλέον, λόγω των πλεονεκτημάτων της. Αρχικά, αν η υλοποίηση της εφαρμογής γίνεται μέσω κώδικα, όπως στην περίπτωση της παρούσας εργασίας, μπορεί να προσαρμοστεί καλύτερα στις απαιτήσεις του συστήματος και να γίνει καλύτερη εξατομίκευση, η οποία βοηθά στον καλύτερο έλεγχο. Η ταχύτητα εξαρτάται από τη σταθερότητα της σύνδεσης, ωστόσο, όσον αφορά την απομακρυσμένη επίβλεψη ενός συστήματος, είναι φυσιολογικό να υπάρχουν καθυστερήσεις. Η χρήση της android εφαρμογής μπορεί παρόλα αυτά να αποδειχθεί η πιο αποτελεσματική. Πιο συγκεκριμένα, η λύση του web server, απαιτεί τη χρήση ενός Η/Υ και συνεπώς προστίθεται η δυσκολία στη φορητότητα, ένα πρόβλημα το οποίο έχει αντιμετωπιστεί με τη χρήση φορητών Η/Υ (laptop). Παρόλα αυτά, επειδή η χρήση ‘’έξυπνων’’ κινητών συσκευών (Smartphones) είναι πλέον ευρεία, ο χρήστης μπορεί να έχει στο χέρι

του τη διαχείριση του συστήματος, ανά πάσα ώρα και στιγμή και να λαμβάνει πληροφορίες ή να παρεμβαίνει σε λειτουργίες, χωρίς να βρίσκεται αναγκαστικά κοντά στο σύστημα. Συνεπώς, μειώνεται αποτελεσματικά ο χρόνος προειδοποίησης και αντίδρασης, κάτι που έχει ως αποτέλεσμα να προβλέπονται επικείμενες βλάβες, να ελέγχονται και να επιβλέπονται διάφορες λειτουργίες του συστήματος χωρίς φυσική παρουσία από το χρήστη και να γίνεται πιο εύκολος και γρήγορος ο τρόπος διαχείρισης του συστήματος, συνολικά. Επίσης, η χρήση Android συσκευών αποτελεί πλεονέκτημα, καθώς προσφέρουν μεγάλη ποικιλία επιλογών εξατομίκευσης, πολλές επιλογές εισόδων, είτε από το ενσωματωμένο πληκτρολόγιο είτε μέσω ακόμα και φωνητικών εντολών και έναν αρκετά μεγάλο χώρο αποθήκευσης, ο οποίος μπορεί να προσαρμοστεί, ανάλογα τις απαιτήσεις. Το android λειτουργικό είναι βασισμένο στο λογισμικό Linux, το οποίο είναι ένα λογισμικό ανοικτού πηγαίου κώδικα (Open-Source) και η πιο συχνή επιλογή για τον προγραμματισμό των android εφαρμογών είναι η γλώσσα προγραμματισμού Java. Τα παραπάνω προσφέρουν ένα αρκετά σταθερό λογισμικό, για το οποίο υπάρχουν αρκετές δωρεάν και καλής ποιότητας εφαρμογές, ανοικτού πηγαίου κώδικα, οι οποίες μπορούν να αφορούν τη δημιουργία ενός web server, μιας βάσης δεδομένων ή τη χρήση εργαλείων για την υλοποίηση της εκάστοτε εφαρμογής. Επίσης, η συνδεσιμότητα των android συσκευών παίζει, όπως αναφέρθηκε, μεγάλο ρόλο στην επίτευξη επικοινωνίας μεταξύ συσκευών, είτε σε τοπικό δίκτυο (Ethernet, USB, WiFi) είτε απομακρυσμένα (GPRS, 3G/4G/5G). Πολύ συχνά χρησιμοποιείται ο συνδυασμός των παραπάνω. Δηλαδή, ένα σύστημα επιτηρείται από έναν server και από μια android εφαρμογή ταυτόχρονα. Η πρόσβαση και των δύο μπορεί να αφορά το τοπικό δίκτυο αλλά και μια εξωτερική απομακρυσμένη σύνδεση για συνεχή επίβλεψη και παρέμβαση. Για να επιτευχθεί μια απομακρυσμένη σύνδεση ωστόσο, θα πρέπει ο κεντρικός ελεγκτής να είναι συνδεδεμένος σε ένα τοπικό δίκτυο, το οποίο παρέχει πρόσβαση στο διαδίκτυο. Από εκεί, θα πρέπει να ακολουθηθεί η μέθοδος του Port Forwarding, μέσω της οποίας ανοίγεται ένας δίαυλος επικοινωνίας μέσω διαδικτύου. Τα βήματα αυτής της μεθόδου περιγράφονται σε παρακάτω ενότητα του παρόντος κεφαλαίου. Όμως, καθώς οι προτεινόμενες μέθοδοι επιλύουν τα προβλήματα της απομακρυσμένης επικοινωνίας και επίβλεψης, η εφαρμογή τους μπορεί να επιφέρει προβλήματα συμβατότητας ή αστάθειας με μερικές συσκευές, ανάλογα τη μελέτη και την ανάπτυξη τους. [11][24][32][33]

Από την πλευρά τους, οι περισσότεροι κατασκευαστές, για να επιλύσουν τα παραπάνω προβλήματα, μαζί με άλλα που προκύπτουν και να προσφέρουν τις πλέον επαγγελματικές λύσεις, έχουν καταφέρει να αναπτύξουν συσκευές επικοινωνίας οι οποίες προσθέτουν σταθερότητα και ασφάλεια στο σύστημα, μεταδίδοντας και λαμβάνοντας δεδομένα ακόμα και κάτω από αντίξοες συνθήκες, ακολουθώντας κάποια πρότυπα. Ωστόσο, όπως κάθε συσκευή που είναι συνδεδεμένη στο διαδίκτυο, έτσι και τα συστήματα SCADA απειλούνται από τους ίδιους κινδύνους. Εξίσου σημαντική με την απώλεια δεδομένων αποτελεί και η δυσλειτουργία του συστήματος που μπορεί να προκαλέσει μια τέτοια παραβίαση, με ότι συνεπάγεται αυτό για τη λειτουργία, την ταχύτητα και την αποδοτικότητα. Για αυτό, οι περισσότερες συνδέσεις που απαιτούν επικοινωνία εκτός τοπικού δικτύου, μέσω ίντερνετ γίνονται μέσω εικονικού ιδιωτικού δικτύου (Virtual Private Network – VPN). Προϋπόθεση για την απομακρυσμένη επικοινωνία αποτελεί επίσης η διαδικασία του Port Forwarding, μέσω της οποίας ανοίγεται ένας δίαυλος επικοινωνίας ανάμεσα σε μια συσκευή που ανήκει σε κάποιο τοπικό δίκτυο και σε μια άλλη, η οποία ανήκει σε διαφορετικό. Με τη χρήση VPN και την προστασία των εξυπηρετητών (servers), προστίθεται συνεπώς ένα ακόμη στρώμα ασφάλειας, μέσω του οποίου κρυπτογραφείται η επικοινωνία και τα δεδομένα που ανταλλάσσονται μεταξύ συσκευών, οι οποίες ανήκουν σε αυτό το δίκτυο. Επιπλέον, σε τέτοιου είδους επικοινωνίες εφαρμόζονται και κάποια πρότυπα, όπως IEEE 1402, ISO 17799 και άλλα, τα οποία εξασφαλίζουν την αντιμετώπιση απειλών και παραβιάσεων. Ωστόσο, η προσπάθεια αντιμετώπισης τέτοιων απειλών είναι συνεχής και ακόμα

και τα πρότυπα που ορίζονται έχουν κάποιες αδυναμίες, οι οποίες διορθώνονται συνεχώς, έτσι ώστε να προσφέρεται η μεγαλύτερη δυνατή ασφάλεια, σε κάθε περίπτωση. [20][21][26][29]

Όπως προκύπτει από τα παραπάνω και από τη βιβλιογραφική έρευνα, η επικοινωνία αποτελεί το σημαντικότερο κομμάτι σε ένα σύστημα SCADA. Από την μεριά του τοπικού δικτύου, αυτό θα πρέπει να είναι σταθερό, γρήγορο και να μην επηρεάζεται από παρεμβολές οι οποίες υπάρχουν μέσα σε ένα βιομηχανικό περιβάλλον. Επίσης, οι συσκευές θα πρέπει να είναι συμβατές με το πρωτόκολλο επικοινωνίας που θα επιλεγεί σε κάθε εφαρμογή. Οπότε, απαιτείται ιδιαίτερη μελέτη και προσοχή για τα σωστά χαρακτηριστικά του συστήματος και τις συσκευές οι οποίες θα επιλεγούν. Ιδιαίτερη μελέτη χρειάζεται και για τον τρόπο επικοινωνίας ο οποίος θα επιλεγεί, ο οποίος πρέπει να βασίζεται στις υποδομές, τις συσκευές και τις απαιτήσεις του συστήματος.

Από την άλλη πλευρά, όσον αφορά την απομακρυσμένη επικοινωνία, ισχύουν οι ίδιες προϋποθέσεις που αφορούν και το τοπικό δίκτυο, μόνο που σε αυτήν την περίπτωση πρέπει να λαμβάνονται υπόψιν και όλοι οι κίνδυνοι που απειλούν οποιαδήποτε συσκευή είναι συνδεδεμένη στο διαδίκτυο. Θα πρέπει δηλαδή να γίνεται η χρήση όλων των βιομηχανικών πρωτοκόλλων επικοινωνίας και προστασίας από απειλές, καθώς αυτές μπορεί να αποβούν μοιραίες για την ανθρώπινη ζωή και για το σύστημα. Τα πρωτόκολλα αυτά έχουν αναπτυχθεί και συνεχίζουν να μελετώνται από ερευνητές και βιομηχανικούς κολοσσούς, κάτι που κάνει την ανάπτυξη τους όλο και πιο σταθερή και αξιόπιστη, βασισμένη σε πραγματικές ανάγκες και υπάρχοντες κινδύνους. Η έρευνα για όλους τους κατασκευαστές αποτελεί επένδυση και συνεχή αγώνα, για την καλύτερη προστασία, τη σταθερότητα και την αξιοπιστία των συστημάτων τους. Για αυτό και όπως φαίνεται από τη βιβλιογραφία, πολλές έρευνες έχουν πραγματική και άμεση εφαρμογή στο πεδίο, ανεξάρτητα από τον τομέα της βιομηχανίας, καθώς η προστασία δεδομένων και άλλων σημαντικών πληροφοριών από κακόβουλο λογισμικό αφορά τους πάντες. Όπως γίνεται κατανοητό λοιπόν, η κρισιμότητα τέτοιων πρωτοκόλλων έγκειται στην προστασία που προσφέρουν, πρωτίστως στην ανθρώπινη ζωή και έπειτα στην εκάστοτε εγκατάσταση και τα δεδομένα της.

1.4 Περιγραφή βασικής λειτουργίας

Το σύστημα αυτοματισμού που αναλύεται στην παρούσα εργασία, αποτελείται από τρία μέρη:

- Το μέρος του αυτοματισμού από την πλευρά του PLC.
- Το μέρος της λειτουργίας της Android εφαρμογής.
- Τον προγραμματισμό και την επικοινωνία των δύο παραπάνω.

Οι λεπτομέρειες των παραπάνω αναφέρονται και αναλύονται στα παρακάτω κεφάλαια και γίνεται εκτενής αναφορά στα διαγράμματα ροής που ακολουθεί κάθε πλευρά, στον κώδικα που κάνει εφικτή τη σωστή λειτουργία τους, καθώς και το υλικό που συνοδεύει το καθένα. Πρώτα όμως, είναι αναγκαίο να γίνει επεξήγηση της βασικής λειτουργίας του συνόλου της εφαρμογής, την ακολουθία των διαδικασιών και το γενικό πλαίσιο, στο οποίο κινούνται όλα τα μέρη της.

Η βασική ιδέα αφορά τον αυτοματισμό της λειτουργίας μιας δεξαμενής υγρού τρόφιμου, το οποίο χρησιμοποιείται στη βιομηχανία τροφίμων. Στη σωστή λειτουργία συμβάλλουν πολλοί παράγοντες, όπως η σωστή θερμοκρασία του υγρού (ψύξη συνήθως κοντά στους 5°C), η σωστή ανάδευση του, ο τακτικός καθαρισμός της δεξαμενής και ο χρόνος καθαρισμού. Επίσης, το υγρό πρέπει να διαμοιράζεται στην παραγωγή για τη χρήση σε κάθε γραμμή και θα πρέπει να γίνεται πλήρωση της δεξαμενής όταν το υγρό φτάνει στο κάτω όριο. Άρα, όλες αυτές οι λειτουργίες πρέπει να ρυθμίζονται από τον εκάστοτε χρήστη και να εκτελούνται από ένα PLC. Στις ίδιες λειτουργίες μπορεί να παρέμβει

και μέσω Android εφαρμογής, η οποία του δίνει πρόσβαση σε όλες τις παραπάνω δυνατότητες. Ο χρήστης έχει πρόσβαση στην επιλογή της επιθυμητής θερμοκρασίας, στο χρόνο ανάδευσης και στο χρόνο καθαρισμού. Επίσης, μπορεί να εκκινήσει τον αναδευτήρα, την αντλία αναπλήρωσης και το σύστημα καθαρισμού της δεξαμενής, υπό κάποιες προϋποθέσεις, οι οποίες θα εξηγηθούν αναλυτικά σε παρακάτω κεφάλαιο. Ακόμη, μπορεί να ειδοποιηθεί για την ανάγκη αναπλήρωσης, όταν το υγρό φτάνει σε χαμηλό επίπεδο, από τον αντίστοιχο αισθητήρα ή για κάποια έκτακτη ανάγκη του συστήματος. Συνεπώς, ο χρήστης αποφασίζει και έχει τη δυνατότητα να ρυθμίσει αρκετές λειτουργίες του αυτοματισμού για την προσαρμογή της εφαρμογής, ανάλογα τις ανάγκες κάθε εγκατάστασης. Για να γίνουν πιο κατανοητές οι βασικές έννοιες που αναλύονται παρακάτω, στις επόμενες παραγράφους παρατίθενται λεπτομέρειες του συστήματος.

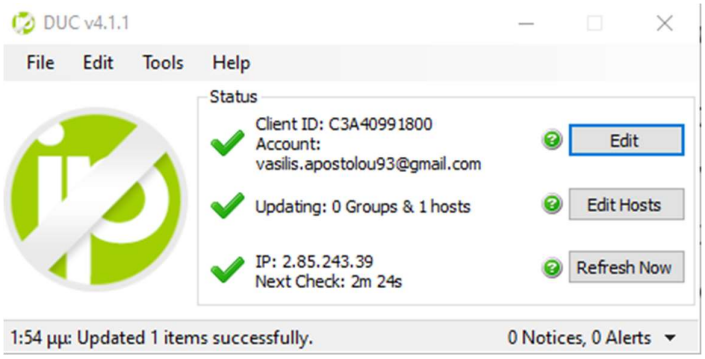
Πιο συγκεκριμένα, το σύστημα αυτοματισμού ελέγχεται από το PLC και χωρίζεται σε τρία κομμάτια. Το πρώτο είναι το ίδιο το PLC, το δεύτερο η οθόνη HMI (Human Machine Interface) η οποία χρησιμοποιείται για τη διεπαφή χρήστη και το τελευταίο ο πίνακας αυτοματισμού. Στο πρώτο μέρος, το PLC δέχεται δεδομένα στις ψηφιακές εισόδους του από τα πλήκτρα ελέγχου και στην αναλογική είσοδο από το αισθητήριο θερμοκρασίας. Η ενεργοποίηση των ψηφιακών εξόδων εξαρτάται από τον κώδικα, ο οποίος έχει αναπτυχθεί σε Ladder. Οι έξοδοι με τη σειρά τους, ενεργοποιούν από έναν ηλεκτρονόμο (ρελέ – relay) η κάθε μια, οι λειτουργίες των οποίων εξηγούνται παρακάτω. Στην οθόνη απεικονίζονται χρήσιμες πληροφορίες για το σύστημα, ενώ δίνεται η δυνατότητα στο χρήστη να ελέγχει τη συμπεριφορά του συστήματος. Τέλος, ο πίνακας αυτοματισμού αποτελείται από πλήκτρα ελέγχου, το PLC και την τροφοδοσία, ενώ συνδεδεμένος σε αυτόν είναι και ο υποπίνακας, μέσα στον οποίο περιέχεται ο ESP32, το αισθητήριο θερμοκρασίας και το κύκλωμα ενίσχυσης. Τα παραπάνω χρησιμεύουν στη λήψη της θερμοκρασίας από το DHT11 προς το ESP32, την επεξεργασία της μέσα στον κώδικα του μικροελεγκτή και την αποστολή της στη βαθμίδα ενίσχυσης, η οποία με τη σειρά της την αποστέλλει στην αναλογική είσοδο του PLC.

Από την άλλη πλευρά, η Android εφαρμογή αναπτύχθηκε στο Android Studio και χωρίζεται σε έξι ενότητες, ή αλλιώς Activities. Κάθε Activity εξυπηρετεί και σε μια συγκεκριμένη λειτουργία. Η ροή της ξεκινάει με την είσοδο του χρήστη, εισάγοντας τα κατάλληλα διαπιστευτήρια. Στη συνέχεια, εισάγεται η διεύθυνση του PLC με το οποίο επιθυμεί ο χρήστης να επικοινωνήσει και αν η σύνδεση είναι επιτυχής, εμφανίζονται σε διαφορετικό Activity διάφορα δεδομένα που λαμβάνονται, όπως η πραγματική θερμοκρασία, η κατάσταση της ψύξης και άλλα. Ταυτόχρονα, ο χρήστης έχει τη δυνατότητα να αποστείλει δεδομένα προς το PLC, αλλάζοντας μεταβλητές σε συγκεκριμένες θέσεις, οι οποίες επηρεάζουν μια ή παραπάνω λειτουργίες του συστήματος. Σε επόμενο Activity, μπορούν να ελεγχθούν οι έξοδοι του PLC και να ληφθούν πληροφορίες για το επίπεδο της στάθμης και την κατάσταση των εξόδων. Αν έχει ενεργοποιηθεί το στοπ έκτακτης ανάγκης (Emergency Stop), τότε η εφαρμογή εμφανίζει το κατάλληλο Activity, το οποίο προειδοποιεί τον χρήστη. Ανάλογα την κατάσταση του συστήματος, ο χρήστης μπορεί να επαναφέρει το στοπ μόνο από την εφαρμογή, αν αυτό δεν αφορά το στοπ που υπάρχει στον πίνακα αυτοματισμού και αν το σφάλμα δεν παραμένει. Τέλος, ο χρήστης έχει τη δυνατότητα να ορίσει εξατομικευμένα διαπιστευτήρια, στο κατάλληλο Activity.

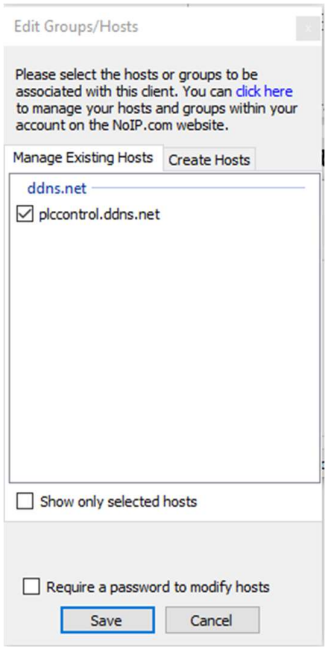
Όσον αφορά το κομμάτι της απομακρυσμένης επικοινωνίας εκτός τοπικού δικτύου, απαιτείται η χρήση της μεθόδου του Port Forwarding. Δηλαδή, της διάνοιξης ενός διαύλου επικοινωνίας της συσκευής που βρίσκεται μέσα σε ένα τοπικό δίκτυο, με το διαδίκτυο. Βέβαια, η μέθοδος αυτή ενέχει κινδύνους, οι οποίοι αναφέρθηκαν σε προηγούμενη ενότητα και θα πρέπει να γίνεται με προσοχή, κυρίως με τη χρήση ενός εικονικού ιδιωτικού δικτύου (VPN).

Επειδή η IP διεύθυνση ενός δρομολογητή (router) αλλάζει ανά τακτά χρονικά διαστήματα, εφόσον είναι δυναμική και όχι στατική, ανάλογα τις ρυθμίσεις και τις παροχές της εκάστοτε εταιρείας τηλεφωνίας, για να γίνει εφικτή η συνεχής επικοινωνία δύο συσκευών, οι οποίες δεν ανήκουν στο ίδιο τοπικό δίκτυο, προτείνεται η χρήση μιας σταθερής διεύθυνσης μέσω του εργαλείου No-IP (www.no-ip.com). Εκεί, ακολουθώντας τα βήματα των οδηγιών χρήσης της ιστοσελίδας, μπορεί να υλοποιηθεί μια δωρεάν ιστοσελίδα (ή ένας server) της οποίας το όνομα δίνεται από τον χρήστη, παραμένει σταθερό, και έχει συγκεκριμένες καταλήξεις, όπως ddns.net, χρησιμοποιώντας την εξωτερική IP του δρομολογητή του εκάστοτε δικτύου. Δηλαδή, μια τέτοια σελίδα μπορεί να υλοποιηθεί και σε ένα οικιακό δίκτυο. Επίσης, για να λειτουργήσει η σελίδα θα πρέπει να γίνει η χρήση ενός εργαλείου που προσφέρεται δωρεάν μέσω του No-IP και αντίστοιχες ρυθμίσεις στον δρομολογητή.

Το εργαλείο ονομάζεται DUC (Εικόνα 1.1), εγκαθίσταται σε έναν υπολογιστή και είναι στην ουσία η εφαρμογή η οποία ελέγχει ανά τακτά χρονικά διαστήματα αν έχει αλλάξει η εξωτερική διεύθυνση IP του δρομολογητή και ανανεώνεται αυτόματα, χωρίς να χρειάζεται ο χρήστης να φτιάξει τη σελίδα από την αρχή, για να χρησιμοποιήσει τη νέα IP. Επίσης, μέσω του εργαλείου μπορεί να γίνει η διαχείρηση ενός ή παραπάνω σελίδων που έχουν δημιουργηθεί από το χρήστη, μέσω της επιλογής ‘’Edit Hosts’’ (Εικόνα 1.2).



Εικόνα 1.1: Το εργαλείο DUC.

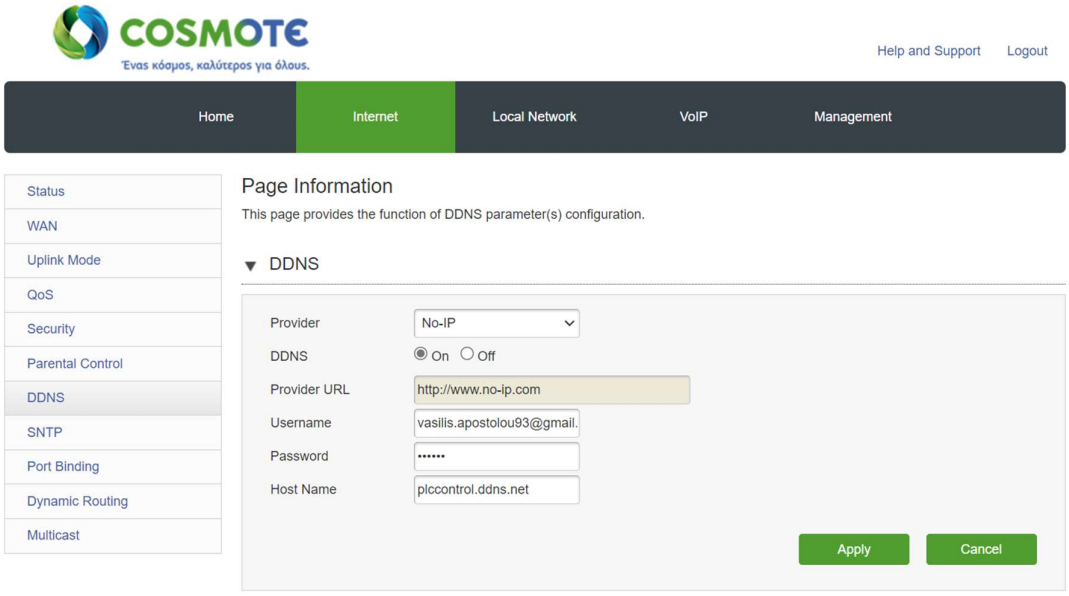


Εικόνα 1.2: Η επιλογή ‘’Edit Hosts’’.

Όπως αναφέρθηκε, μαζί με τα παραπάνω θα πρέπει να γίνουν και κάποιες συγκεκριμένες ρυθμίσεις στο δρομολογητή. Αυτές αφορούν το Port Forwarding, όπως το προτείνει ο εκάστοτε πάροχος τηλεφωνίας, ο οποίος στη συγκεκριμένη περίπτωση είναι η COSMOTE. Σύμφωνα λοιπόν με τις οδηγίες οι οποίες βρίσκονται στην επίσημη ιστοσελίδα του παρόχου, θα πρέπει να γίνουν οι παρακάτω ρυθμίσεις στο δρομολογητή:

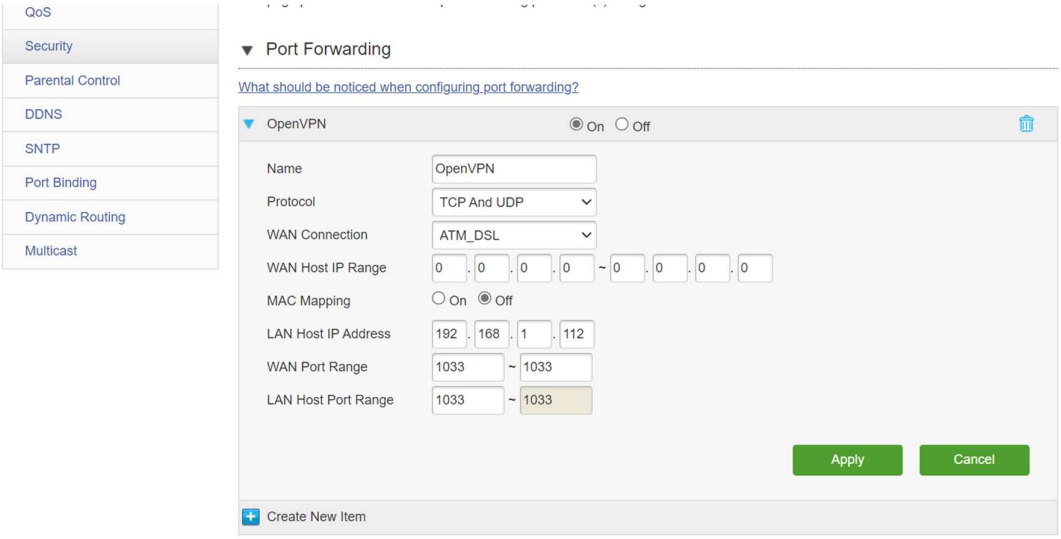
- Ρύθμιση DDNS
- Ρύθμιση Port Forwarding

Οι παραπάνω ρυθμίσεις γίνονται στο περιβάλλον του δρομολογητή, το οποίο είναι προσβάσιμο συνήθως μέσω της διεύθυνσης 192.168.1.1. Η διεύθυνση αυτή οδηγεί στη σελίδα του δρομολογητή, όπου ζητούνται τα διαπιστευτήρια του χρήστη. Μετά τη σύνδεσή του, εμφανίζεται το περιβάλλον μέσω του οποίου μπορούν να γίνουν διάφορες ρυθμίσεις. Στην περίπτωση της COSMOTE, η ρύθμιση του DDNS βρίσκεται στην καρτέλα ‘’Internet’’, στην επιλογή ‘’DDNS’’, όπως φαίνεται στην Εικόνα 1.3. Μέσω αυτής της επιλογής μπορεί να επικοινωνήσει ο δρομολογητής με τη σελίδα η οποία υλοποιήθηκε μέσω του No-IP.



Εικόνα 1.3: Η ρύθμιση του DDNS.

Η επόμενη ρύθμιση του Port Forwarding βρίσκεται στην καρτέλα ‘’Internet’’, στην επιλογή ‘’Security’’ και την υποεπιλογή ‘’Port Forwarding’’, όπως φαίνεται στην Εικόνα 1.4. Εκεί ο χρήστης μπορεί να δώσει ένα οποιοδήποτε όνομα, επιλέγει το πρωτόκολλο επικοινωνίας και στο πεδίο ‘’LAN Host IP Address’’ επιλέγει την εσωτερική διεύθυνση της συσκευής, την οποία θέλει να προωθήσει στο διαδίκτυο. Στη συγκεκριμένη περίπτωση είναι η IP διεύθυνση του PLC. Τέλος, στις δύο τελευταίες επιλογές επιλέγεται η εξωτερική και εσωτερική πόρτα αντίστοιχα, μέσω των οποίων γίνεται δυνατή η προώθηση της συσκευής στο διαδίκτυο.



Εικόνα 1.4: Η ρύθμιση Port Forwarding.

Κεφάλαιο 2ο: Περιγραφή του Αυτοματισμού

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναλυθούν η γενική ιδέα του αυτοματισμού από τη μεριά του PLC, η ροή που ακολουθεί ο κώδικας και θα γίνει εκτενής περιγραφή της λειτουργίας των επιμέρους πλευρών. Ακόμη, θα γίνει αναφορά στη διαδικασία υλοποίησης του πίνακα αυτοματισμού που συνοδεύει την εφαρμογή και ανάλυσή του. Τέλος, θα παρουσιαστούν τα αποτελέσματα και τα συμπεράσματα.

2.2 Αρχές Λειτουργίας

2.2.1 Θεωρητική Ανάλυση

Όπως ήδη αναφέρθηκε στο προηγούμενο κεφάλαιο, η εφαρμογή αυτοματισμού που αναλύεται στην παρούσα εργασία, αφορά τον απομακρυσμένο έλεγχο ενός PLC, το οποίο με τη σειρά του ελέγχει μια σειρά λειτουργιών που αφορούν τη διατήρηση και το διαμοιρασμό ενός υγρού τρόφιμου (προζύμι ή μαγιά) σε μια ή περισσότερες γραμμές παραγωγής καθώς και την επίβλεψη ολόκληρου του συστήματος. Παρακάτω, φαίνεται μια λίστα με τις βασικές λειτουργίες οι οποίες πρέπει να επιτηρούνται και να ακολουθούνται για την απρόσκοπτη λειτουργία του συστήματος και την εξασφάλιση της ποιότητας του προϊόντος:

- Έλεγχος για έκτακτη ανάγκη ή ενεργοποίησης Emergency Stop.
- Έλεγχος ύπαρξης υγρού μέσα στη δεξαμενή.
- Επίβλεψη της θερμοκρασίας και ανάλογη χρήση της ψύξης.
- Ανάδευση σε τακτά χρονικά διαστήματα.
- Έλεγχος του επιπέδου του υγρού και προειδοποίηση για την ανάγκη αναπλήρωσης.
- Καθαρισμός της δεξαμενής, μόνο αν έχει αδειάσει πλήρως.

Για κάθε μία από τις παραπάνω λειτουργίες πρέπει να τηρούνται κάποιοι κανόνες και να πληρούνται κάποιες προϋποθέσεις, έτσι ώστε να εξασφαλίζεται και η ασφάλεια του χρήστη, αλλά και η διατήρηση της ποιότητας. Η ποιότητα δεν αφορά μόνο το προϊόν της δεξαμενής, αλλά ολόκληρο το σύστημα, καθώς οι καταπονήσεις των κινητήρων, ή η κακή μεταχείριση των αισθητηρίων λόγω κακής χρήσης ή παράλειψης καθαρισμού, μπορούν να επηρεάσουν τη λειτουργία ή να οδηγήσουν ακόμα και σε ανεπιθύμητες βλάβες, οι οποίες θα μπορούσαν να είχαν προβλεφθεί. Προς αποφυγή, λοιπόν, τέτοιων φαινομένων, η λειτουργία του συστήματος βασίζεται σε κάποιους κανόνες, οι οποίοι εφαρμόζονται και μέσα στον κώδικα.

Για να γίνει πιο κατανοητή η λειτουργία του αυτοματισμού, πρέπει πρώτα να αναφερθούν τα επιμέρους κομμάτια, τα οποία συνδυάζονται για την επίτευξη του τελικού αποτελέσματος. Ο χρήστης, λοιπόν, μπορεί να ελέγξει αρκετές παραμέτρους του συστήματος, όπως είναι η θερμοκρασία που διατηρείται το υγρό μέσα στη δεξαμενή, η εκκίνηση και ο χρόνος ανάδευσης, η εκκίνηση και ο χρόνος καθαρισμού και η έναρξη ή διακοπή της αναπλήρωσης της δεξαμενής.

Κάθε μια ακολουθία που αφορά τον έλεγχο της κάθε συσκευής ξεχωριστά, τον έλεγχο θερμοκρασίας, τον έλεγχο στάθμης, τον έλεγχο χρονικών, πράξεων μεταβλητών και την ανάγνωση ή καταγραφή δεδομένων, περιέχεται σε διαφορετικό κομμάτι κώδικα (Network) μέσα στο TIA Portal, το οποίο είναι το περιβάλλον προγραμματισμού της Siemens, καθώς το PLC που χρησιμοποιείται στην παρούσα εργασία, είναι της ίδιας εταιρίας. Αυτό βοηθάει στο διαχωρισμό και στον καλύτερο προγραμματισμό κάθε λειτουργίας ξεχωριστά οι οποίες θα αναλυθούν παρακάτω. [1][5][6]

2.2.2 Βασικές Λειτουργίες

Σε αυτήν την ενότητα αναλύεται το γενικό διάγραμμα ροής και η λειτουργία του κώδικα, ο οποίος γράφτηκε σε γλώσσα Ladder, με τη χρήση του περιβάλλοντος TIA Portal, το οποίο έχει αναπτυχθεί από τη Siemens, καθώς και το PLC που χρησιμοποιείται είναι της ίδιας εταιρίας.

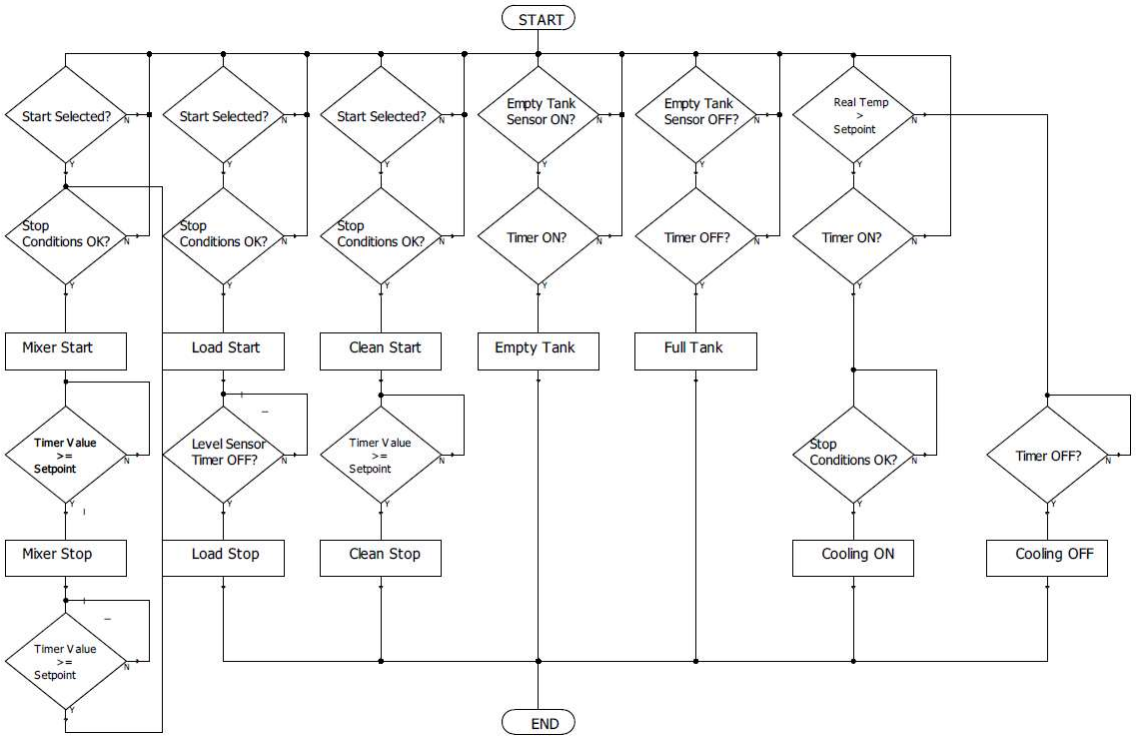
Στη γενική του ροή, ο κώδικας ελέγχει αν υπάρχει κάποια έκτακτη ανάγκη, αν έχει σταματήσει κάποια λειτουργία ενώ δεν έπρεπε ή αν έχει πιεστεί κάποιο πλήκτρο έκτακτης ανάγκης (Emergency Stop). Αν όλες οι προϋποθέσεις τηρούνται, τότε η λειτουργία συνεχίζεται κανονικά. Γίνεται έλεγχος για τον αν υπάρχει υγρό μέσα στη δεξαμενή, με βάση το σήμα ενός αισθητήρα, ο οποίος ‘’αντιλαμβάνεται’’ την ύπαρξη ή όχι του υγρού και στέλνει το αντίστοιχο σήμα στο PLC. Αν δεν υπάρχει υγρό, τότε μπορεί να ξεκινήσει η διαδικασία της αναπλήρωσης ή του καθαρισμού, αλλά όχι και των δύο ταυτόχρονα. Αν υπάρχει υγρό στη δεξαμενή, τότε ξεκινά ο έλεγχος της θερμοκρασίας της δεξαμενής, και γίνεται σύγκριση της με το απαιτούμενο όριο το οποίο τίθεται από τον χρήστη. Αν η πραγματική θερμοκρασία είναι κάτω από το όριο, τότε η ψύξη σταματά. Αν είναι μεγαλύτερη από το όριο, τότε η ψύξη ξεκινάει.

Η πραγματική θερμοκρασία προέρχεται από δεδομένα που μεταδίδονται από έναν DHT11 αισθητήρα θερμοκρασίας, σε ένα ESP32. Από εκεί, μια έξοδος του ESP32 προγραμματίζεται αναλόγως, έτσι ώστε το αποτέλεσμα της να είναι 0 Volts για 0°C και 3,3 Volts για 50°C, καθώς η θερμοκρασία της δεξαμενής πρέπει πάντα να κυμαίνεται ανάμεσα σε ένα εύρος τιμών από 1°C μέχρι 20°C. Η τάση εξόδου του μικροελεγκτή ωστόσο δεν επαρκεί για να εισαχθεί απευθείας στην αναλογική είσοδο του PLC και πρέπει πρώτα να ενισχυθεί από μια ενισχυτική βαθμίδα, η οποία αποτελείται από έναν LM358P και κατάλληλες αντιστάσεις, και η έξοδος της οποίας είναι 0 – 10 Volts. Στις παρακάτω ενότητες γίνεται αναλυτική περιγραφή των παραπάνω σταδίων και του κώδικα που απαιτείται για την εξαγωγή σωστών αποτελεσμάτων από τον ESP32, καθώς και τους υπολογισμούς για τη σωστή τελική έξοδο της ενισχυτικής βαθμίδας.

Τέσσερις έξοδοι του PLC χρησιμοποιούνται για τον έλεγχο της ανάδευσης, της αναπλήρωσης, του καθαρισμού και της ψύξης. Κάθε μια από αυτές ενεργοποιείται και απενεργοποιείται με βάση κάποιες προϋποθέσεις και αυτό μπορεί να γίνει είτε από τον πίνακα αυτοματισμού, είτε από την οθόνη HMI (Human Machine Interface) η οποία λειτουργεί μέσω προσομοίωσης του TIA Portal, είτε από την android εφαρμογή. Η οθόνη HMI και η android εφαρμογή δουλεύουν παράλληλα και οι εντολές μπορούν να φτάσουν είτε από τη μια είτε από την άλλη, με τις αντίστοιχες τιμές να ανανεώνονται και στα δύο πεδία. Τα πεδία αυτά είναι η ενεργοποίηση των τριών εξόδων, του αναδευτήρα, της αναπλήρωσης και του καθαρισμού, ενώ η έξοδος της ψύξης ενεργοποιείται μόνο εσωτερικά από το PLC, με βάση τη σύγκριση ανάμεσα στην πραγματική θερμοκρασία και τη ζητούμενη. Οι δύο αυτές θερμοκρασίες μαζί με τους αντίστοιχους χρόνους καθυστέρησης για την ανάμειξη και τον καθαρισμό, βρίσκονται σε αντίστοιχα πεδία. Επίσης, υπάρχει προειδοποίηση για χαμηλό επίπεδο υγρού στη δεξαμενή και η ένδειξη της κατάστασης της ψύξης (ενεργοποιημένη ή όχι).

Ο αναδευτήρας μπορεί να ενεργοποιηθεί από START είτε της εφαρμογής, είτε της HMI, είτε του πίνακα αυτοματισμού, αν δεν έχει πιεστεί κάποιο STOP ή Emergency Stop, και αν δεν είναι ενεργοποιημένη η αντλία αναπλήρωσης. Επίσης, η έξοδος παραμένει ενεργοποιημένη με αυτοσυγκράτηση για συγκεκριμένο χρόνο, τον οποίο καθορίζει ο χρήστης, έπειτα απενεργοποιείται για τον ίδιο χρόνο και ενεργοποιείται ξανά. Η διαδικασία συνεχίζει κυκλικά, μέχρι να πιεστεί κάποιο στοπ. Ο προκαθορισμένος χρόνος μέσα στον κώδικα είναι δέκα δευτερόλεπτα, έτσι ώστε να ελέγχεται τουλάχιστον η σωστή λειτουργία. Ο χρήστης θα πρέπει να ορίσει τον επιθυμητό χρόνο για τον οποίο θα παραμένει ενεργοποιημένη η έξοδος της ανάδευσης. Ο αυτοματισμός αναπλήρωσης,

ενεργοποιείται κι αυτός με START το οποίο μπορεί να προέρχεται είτε από την εφαρμογή, είτε από τον πίνακα, είτε από την HMI και να μην έχει επιλεγεί κανένα STOP ή Emergency Stop. Επίσης, προϋποθέσεις αποτελούν και το να μην έχει ενεργοποιηθεί ο καθαρισμός και το αισθητήριο χαμηλής στάθμης να δείχνει την αντίστοιχη προειδοποίηση. Ενεργοποιείται και συγκρατείται με αυτοσυγκράτηση και απενεργοποιείται όταν απενεργοποιηθεί η ένδειξη του αισθητηρίου χαμηλής στάθμης, μετά από ένα χρόνο πέντε δευτερολέπτων. Ο λόγος που χρησιμοποιείται η καθυστέρηση είναι γιατί όσο η δεξαμενή θα γεμίζει και το υγρό θα κυματίζει, υπάρχει μεγάλη πιθανότητα το αισθητήριο να απενεργοποιείται στιγμιαία και να έχει ως αποτέλεσμα να απενεργοποιείται ο αυτοματισμός πλήρωσης, χωρίς το υγρό να έχει φτάσει στο επιθυμητό σημείο. Σημειώνεται επίσης, ότι η ίδια καθυστέρηση πέντε δευτερολέπτων εφαρμόζεται και όταν το υγρό φτάσει στη χαμηλή στάθμη, για να μην υπάρχουν στιγμιαίες διακοπές και εκκινήσεις του αυτοματισμού. Επίσης, ο χρόνος των καθυστερήσεων επιλέχθηκε να είναι της τάξης των μερικών δευτερολέπτων για λόγους γρήγορης προσομοίωσης. Οι χρόνοι μπορούν να μεταβληθούν μέσα στον κώδικα και να προσαρμοστούν στις εκάστοτε ανάγκες. Τέλος, το σύστημα καθαρισμού ενεργοποιείται επίσης είτε από START του πίνακα αυτοματισμού, είτε από την HMI, είτε από την εφαρμογή και αν δεν έχει πιεστεί κάποιο STOP ή Emergency Stop. Ακόμη, πρέπει αντίστοιχα να μην έχει ενεργοποιηθεί ο αυτοματισμός αναπλήρωσης και ο αισθητήρας χαμηλής στάθμης να έχει ενεργοποιηθεί, υποδεικνύοντας ότι δεν υπάρχει υπολογίσιμη ποσότητα υγρού στη δεξαμενή και δεν υπάρχει κίνδυνος απώλειας προϊόντος. Ο καθαρισμός διαρκεί κάποια δευτερόλεπτα, τα οποία καθορίζονται από τον χρήστη, όπως στην περίπτωση της ανάδευσης, ενώ ο προκαθορισμένος χρόνος είναι επίσης δέκα δευτερόλεπτα. [1][5][6]



Σχήμα 2.1: Το διάγραμμα ροής της Ladder.

Η έξοδος της ψύξης, ενεργοποιείται με βάση τη σύγκριση ανάμεσα στην πραγματική τιμή της θερμοκρασίας όπως λαμβάνεται από την αναλογική είσοδο και στο όριο που καθορίζει ο χρήστης. Η μετατροπή της θερμοκρασίας και η βελτίωση των τιμών της αναλύονται παρακάτω. Η προκαθορισμένη τιμή είναι οι 5°C. Και σε αυτήν την περίπτωση εφαρμόζεται μια καθυστέρηση των

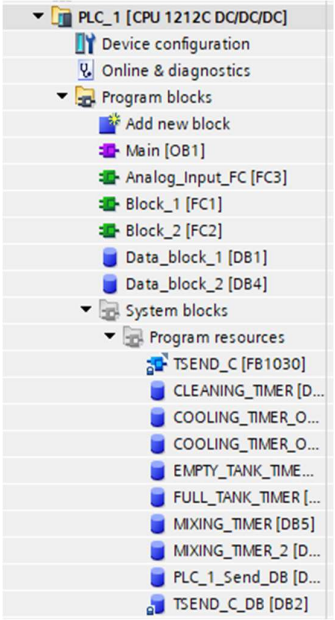
πέντε δευτερολέπτων, η οποία φροντίζει έτσι ώστε να αποφεύγονται στιγμιαίες εκκινήσεις και παύσεις, οι οποίες προκαλούν φθορά στον αυτοματισμό (βλάβη κινητήρα, φθορά σε ηλεκτρονόμους κ.λπ.). Η ενεργοποίηση της ψύξης διατηρείται με αυτοσυγκράτηση, απενεργοποιείται μόλις ενεργοποιηθεί το χρονικό που ενεργοποιείται όταν η πραγματική θερμοκρασία βρεθεί κάτω από το ζητούμενο όριο, ενώ ταυτόχρονα δε θα πρέπει να έχει πιεστεί κανένα Emergency Stop, είτε από τον πίνακα, είτε από την εφαρμογή.

Τέλος, ο κώδικας αποτελείται κι από άλλα Networks, τα οποία θα αναλυθούν λεπτομερώς στην επόμενη ενότητα, όπου γίνεται επεξήγηση κάθε Network ξεχωριστά, για την καλύτερη κατανόηση της λειτουργίας. Κάθε βασική λειτουργία έχει, όπως φαίνεται από τα παραπάνω, ένα δικό της διάγραμμα ροής, το οποίο συνδυάζεται με τα υπόλοιπα για την εύρυθμη λειτουργία του συστήματος. Ένα γενικό διάγραμμα ροής που ακολουθείται από το PLC, φαίνεται παραπάνω στο Σχήμα 2.1.

2.2.3 Ανάλυση Τρόπου Λειτουργίας

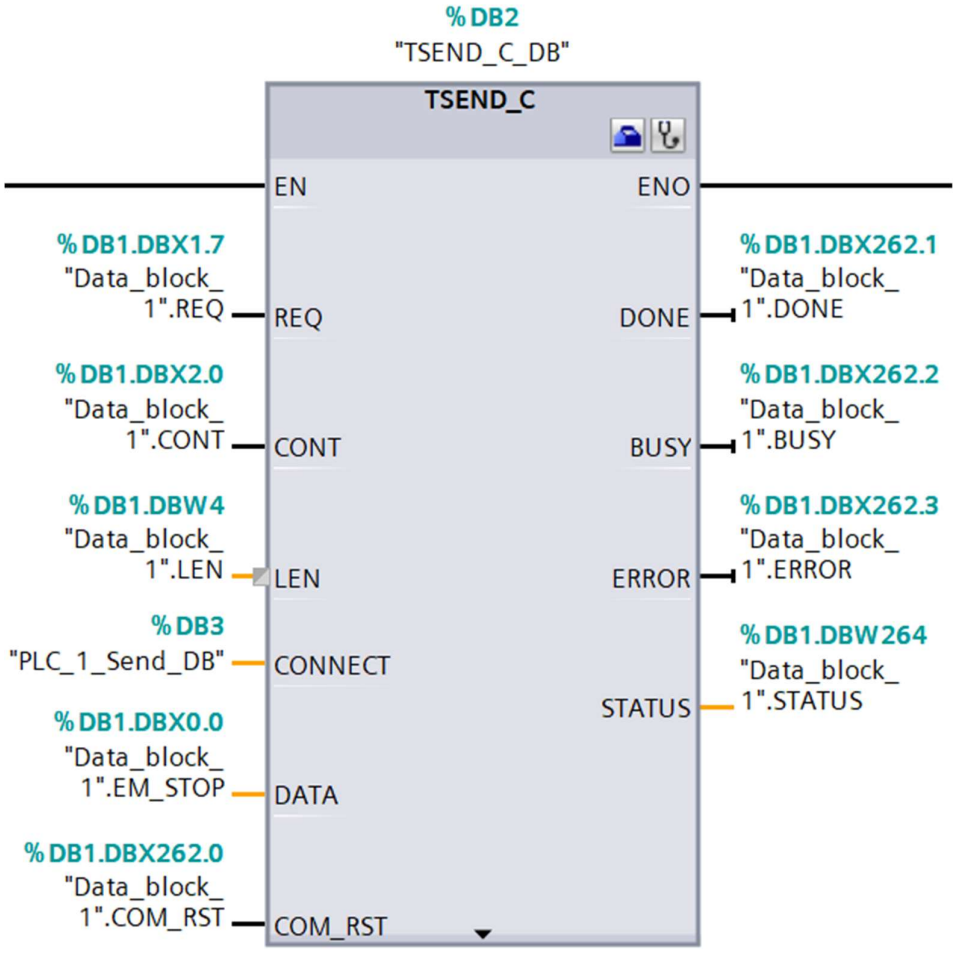
Στην προηγούμενη ενότητα έγινε ανάλυση του τρόπου λειτουργίας των βασικών μερών του συστήματος, της επιρροής που ασκούν το ένα στο άλλο και του τρόπου ελέγχου κάθε βαθμίδας. Σε αυτήν την ενότητα γίνεται ανάλυση όλων των μερών του κώδικα Ladder, ο οποίος χωρίζεται σε Networks, τα οποία διευκολύνουν την κωδικοποίηση. Άλλωστε και η ίδια η Ladder δεν είναι ακριβώς γλώσσα προγραμματισμού με τη συμβατική έννοια, αλλά περισσότερο μια γλώσσα συμβόλων. Ωστόσο είναι ένα χρήσιμο εργαλείο και λόγω του ότι χρησιμοποιεί σύμβολα, παρόμοια με αυτά ενός ηλεκτρολογικού σχεδίου, όπως φαίνεται παρακάτω, μπορεί να διευκολύνει τη μετατροπή ενός ηλεκτρολογικού σχεδίου σε κώδικα. [5][6]

Αφού γίνουν οι αρχικές ρυθμίσεις και ξεκινήσει ένα νέο project στο TIA Portal, μπορεί επίσης να ξεκινήσει και ο προγραμματισμός. Ο κώδικας του συγκεκριμένου project, χωρίζεται σε Program Blocks, τα οποία βρίσκονται στον αντίστοιχο φάκελο με την ίδια ονομασία, όπως φαίνεται στην Εικόνα 2.1. Στο Main Block βρίσκεται ο κυρίως κώδικας, οι βασικές λειτουργίες που αναλύθηκαν στην προηγούμενη ενότητα, ανάμεσα σε άλλες, των οποίων η περιγραφή θα γίνει σε αυτήν την ενότητα.

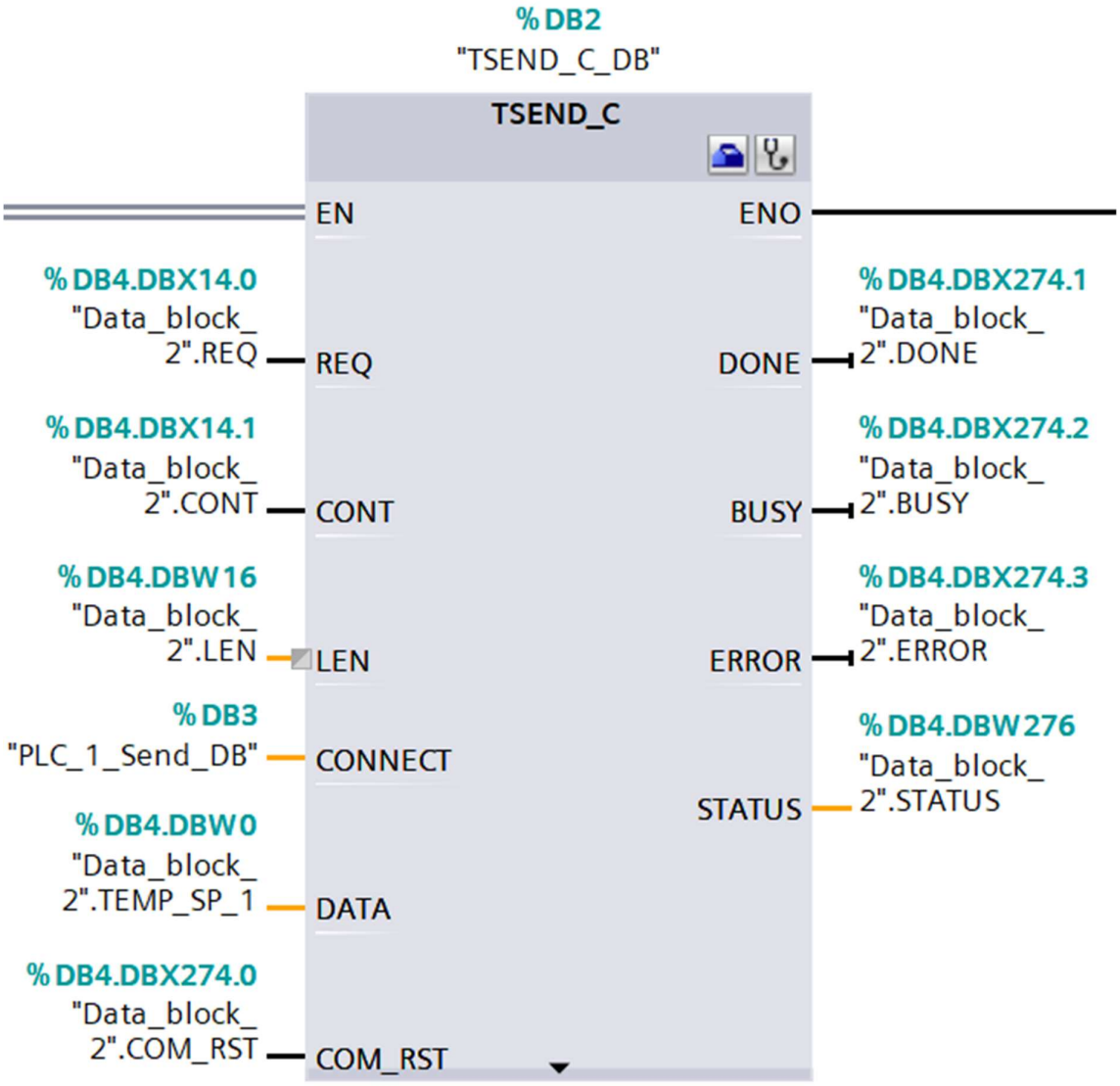


Εικόνα 2.1: Τα Program Blocks του project.

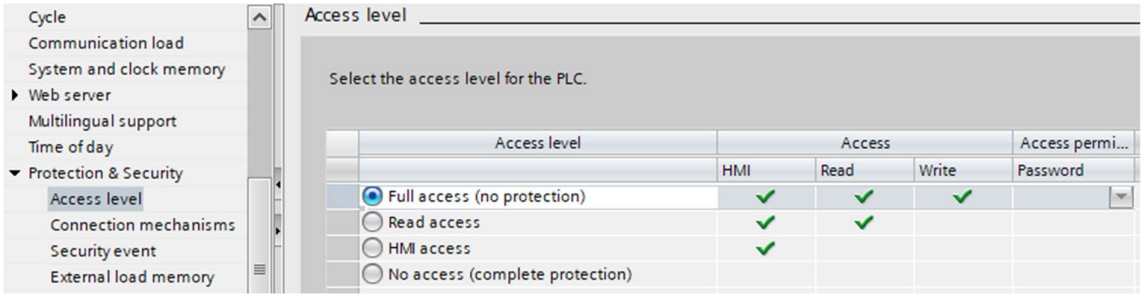
Τα Blocks FC1, FC2 και FC3, ονομάζονται Function Blocks, και περιέχουν κομμάτια κώδικα (ή υπορουτίνες) τα οποία μπορούν να κληθούν από το κυρίως πρόγραμμα και να εκτελεστούν. Πιο συγκεκριμένα, το FC1 περιέχει το Data Block (DB), το οποίο φαίνεται στην Εικόνα 2.2, ονομάζεται TSEND_C και μέσω αυτού γίνεται η σύνδεση και στέλνονται δεδομένα του Data Block 1 προς τον web server ή προς την android εφαρμογή, όπως θα εξηγηθεί παρακάτω. Οι μεταβλητές που χρησιμοποιούνται στη συγκεκριμένη περίπτωση, δεν είναι μεταβλητές οι οποίες έχουν κάποια χρήση στο κυρίως πρόγραμμα. Η ονομασία και η χρήση τους εξυπηρετούν απλά τη σωστή λειτουργία του Block. Το μόνο που αξίζει να σημειωθεί, είναι πως οι μεταβλητές πρέπει να βρίσκονται στο Data Block 1, έτσι ώστε να γνωρίζει από που να διαβάσει τις μεταβλητές. Επίσης, τα ίδια ακριβώς ισχύουν και για το FC2, το οποίο κι αυτό περιέχει το TSEND_DB, όπως φαίνεται στην Εικόνα 2.3, αλλά σε αυτήν την περίπτωση, οι μεταβλητές προέρχονται από το Data Block 4. Ωστόσο, για να εδραιωθεί μια σύνδεση TCP/IP, έτσι ώστε να επιτρέψει την απομακρυσμένη ανάγνωση δεδομένων και την καταγραφή νέων σε συγκεκριμένες θέσεις, θα πρέπει να γίνουν μερικές ακόμη ρυθμίσεις. Τα δύο αυτά Function Blocks, θα πρέπει να καλούνται συνεχώς από το κυρίως πρόγραμμα και συνεπώς θα πρέπει να ανήκουν σε ένα Network το καθένα, όπως θα εξηγηθεί στη συνέχεια. Τέλος, στην αρχική διαμόρφωση της συσκευής (Device Configuration), επιλέγοντας τις ιδιότητες της συσκευής και στην επιλογή ‘’Protection & Security’’ του μενού, στην καρτέλα ‘’Access Level’’, πρέπει να επιλεγεί η πλήρης πρόσβαση (‘’Full Access’’) και στην καρτέλα ‘’Connection Mechanisms’’ η επιλογή PUT/GET, όπως φαίνεται στις Εικόνες 2.4 και 2.5 αντίστοιχα. [5][6]



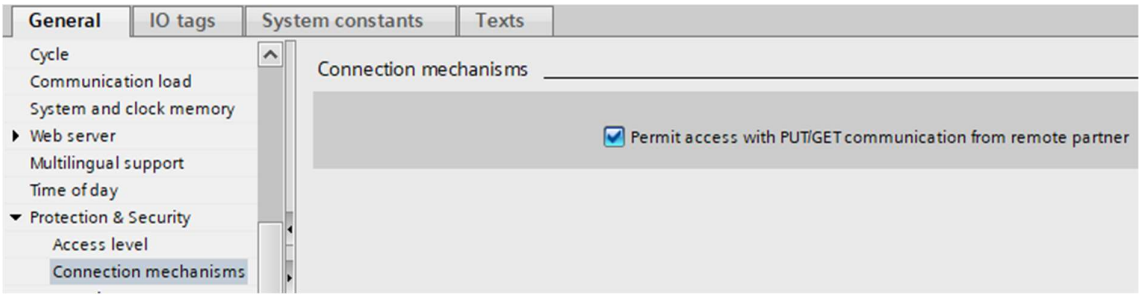
Εικόνα 2.2: Το Data Block που περιέχεται μέσα στο FC1.



Εικόνα 2.3: Το Data Block που περιέχεται μέσα στο FC2.

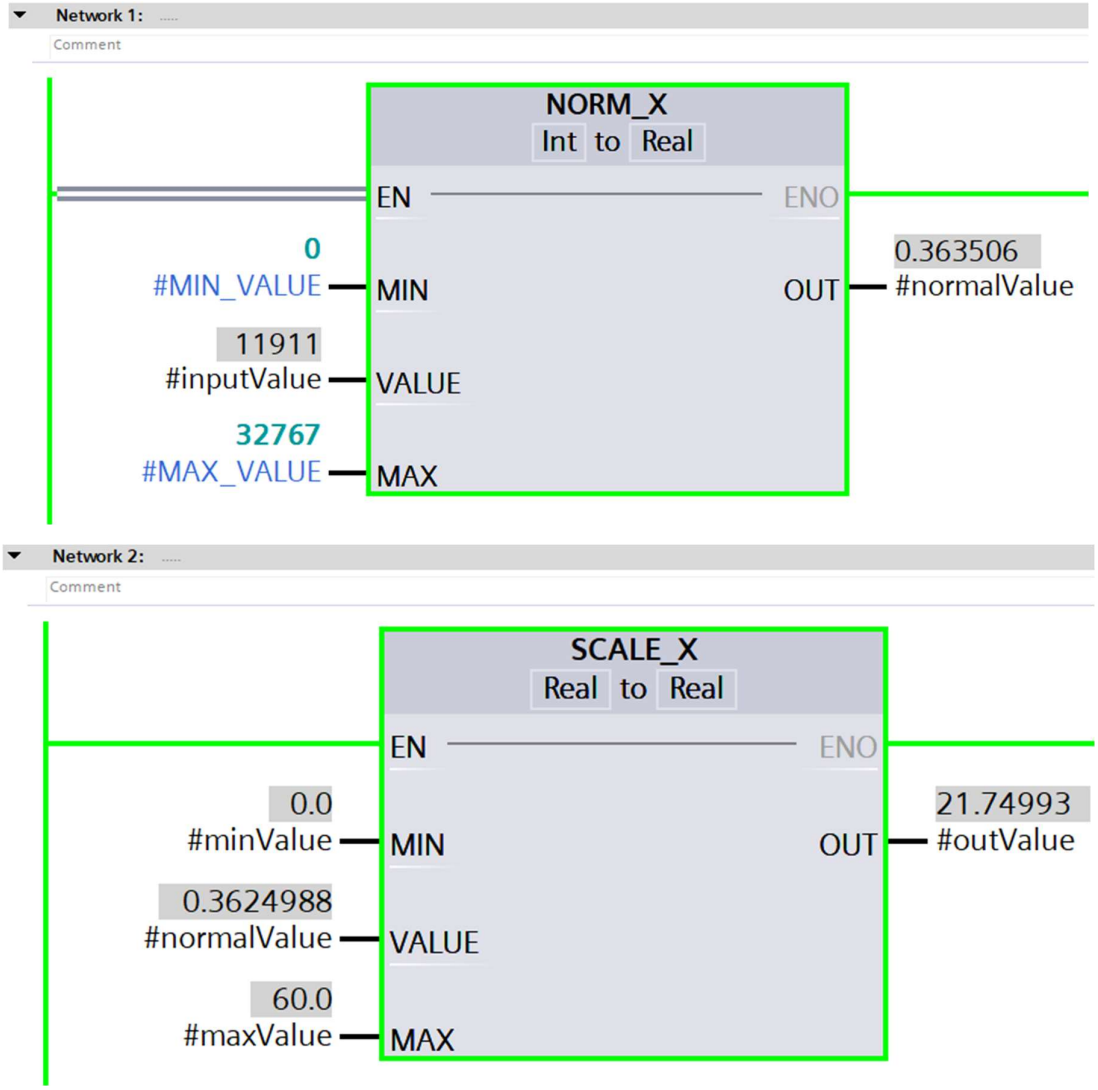


Εικόνα 2.4: Η επιλογή στην καρτέλα ‘’Access Level’’



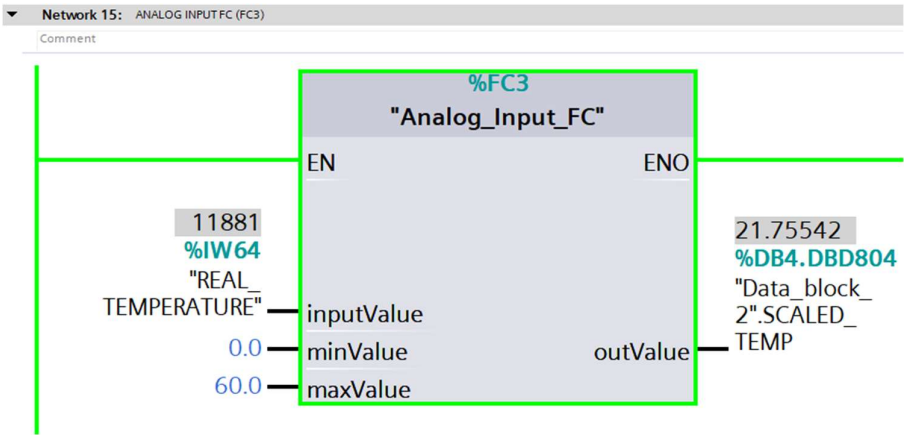
Εικόνα 2.5: Η επιλογή στην καρτέλα ‘’Connection Mechanism’’.

Το FC3 αποτελεί το μπλοκ στο οποίο γίνεται η μετατροπή της τιμής της αναλογικής εισόδου σε θερμοκρασία, έτσι ώστε να γίνεται η ανάγνωση της. Όπως φαίνεται στην Εικόνα 2.6, αποτελείται από δύο networks τα οποία περιέχουν από ένα στοιχείο το καθένα. Στο πρώτο γίνεται η μετατροπή της τιμής σε ‘’κανονική’’ τιμή. Δηλαδή, η τιμή που προκύπτει πρέπει να κυμαίνεται ανάμεσα σε κάποια όρια και αυτά καθορίζονται βάση την ανάλυση (resolution) της αναλογικής εισόδου. [1][6]



Εικόνα 2.6: Η λειτουργία του FC3.

Στη συγκεκριμένη περίπτωση έχει επιλεγεί, όπως φαίνεται, η μέγιστη τιμή ‘’#MAX_VALUE’’ 32.767, καθώς, σύμφωνα με τα δεδομένα της συσκευής που χρησιμοποιήθηκε, η ανάλυση της αναλογικής εισόδου είναι 15 bits (2^{15} = 32.768). Η τιμή αυτή καταλήγει στη θέση IW64, η οποία καθορίζεται από το PLC για τη μια από τις δύο αναλογικές θέσεις της συσκευής. Μπαίνοντας λοιπόν η τιμή στα όρια που ορίζονται από το στοιχείο NORM_X, προκύπτει η τιμή ‘’#normalValue’’ η οποία εισάγεται στο δεύτερο στοιχείο SCALE_X. Εκεί γίνεται ο καθορισμός του εύρους τιμών, ανάμεσα στις οποίες πρέπει να κινείται η συγκεκριμένη τιμή, ενώ οι τιμές ‘’#minValue’’ και ‘’#maxValue’’ δίνονται στο κυρίως πρόγραμμα, όπως φαίνεται στην Εικόνα 2.7. Οι τιμές 0 έως 60, αντιπροσωπεύουν την ελάχιστη και μέγιστη τιμή θερμοκρασίας που χρειάζονται για το συγκεκριμένο σύστημα. Η μέγιστη τιμή διαφέρει κατά 10°C από αυτήν που έχει οριστεί στον κώδικα του ESP32 ως μέγιστη (50°C), λόγω του ότι υπάρχουν μικρές διαφορές ανάμεσα στην αναλογική έξοδο του ESP32, την ενισχυτική βαθμίδα και την αναλογική είσοδο του PLC. Ωστόσο, αυτή η διαφοροποίηση δεν επηρεάζει πουθενά τη λειτουργία κανενός εξαρτήματος ή του κώδικα γενικότερα κι έτσι η αύξηση της μέγιστης επιθυμητής τιμής στον κώδικα του PLC, βοηθάει στην επίτευξη ακρίβειας στις μετρήσεις, καθώς οι τιμές που παράγονται μέσα στον κώδικα, δε διαφέρουν σχεδόν καθόλου από την τιμή που λαμβάνεται από τον αισθητήρα DHT11 και φαίνεται στην OLED, όπως φαίνεται σε παρακάτω παράγραφο. Τέλος, η τιμή που προκύπτει είναι η πραγματική θερμοκρασία η οποία αποθηκεύεται στη μεταβλητή SCALED_TEMP μέσα στο Data Block 4, από όπου μπορεί να διαβαστεί από την HMI και την android εφαρμογή.



Εικόνα 2.7: Η λειτουργία του FC3 μέσα στο Main Block.

Τα επόμενα δύο μπλοκ κώδικα, όπως φαίνεται στην Εικόνα 2.1, είναι τα Data Block 1 (DB1) και Data Block 2 (DB4). Σε αυτού του είδους μπλοκ αποθηκεύονται μεταβλητές δεδομένων όλων των τύπων, οι οποίες χρησιμοποιούνται στο κυρίως πρόγραμμα (Main Block). Στη συγκεκριμένη περίπτωση στο Data Block 1 έχουν δημιουργηθεί και αποθηκευτεί μεταβλητές, οι οποίες αφορούν τα απομακρυσμένα START, STOP και EMERGENCY STOP και χρησιμοποιούνται σαν επαφές σε συνδυασμό με τις επαφές που προέρχονται από τον πίνακα αυτοματισμού. Ακόμη, έχουν δημιουργηθεί μεταβλητές οι οποίες αποτελούν διαγνωστικά για ορισμένες συσκευές, στις οποίες αποθηκεύεται η κατάσταση κάθε μιας και μπορεί να αναγνωστεί από την android εφαρμογή και από την HMI. Τέλος, έχουν δημιουργηθεί και οι μεταβλητές οι οποίες χρησιμοποιούνται στο FC1, όπως αναφέρθηκε παραπάνω. Η λίστα των μεταβλητών του DB1 φαίνεται στον Πίνακα 2.1. Σε παρακάτω ενότητες εξηγείται το πως λειτουργούν αυτές οι μεταβλητές μέσα στον κώδικα, αλλά και πως διαβάζονται απομακρυσμένα.

Πίνακας 2.1: Ο πίνακας μεταβλητών του Data Block 1.

Name	Data Type	Offset	Start value
EM_STOP	Bool	0.0	false
START_MIXER	Bool	0.1	false
STOP_MIXER	Bool	0.2	false
START_LOAD	Bool	0.3	false
STOP_LOAD	Bool	0.4	false
START_CLEAN	Bool	0.5	false
STOP_CLEAN	Bool	0.6	false
MIXER_OK	Bool	0.7	false
LOAD_OK	Bool	1.0	false
CLEAN_OK	Bool	1.1	false
COOLING_OK	Bool	1.2	false
EMERGENCY_ACTIVE	Bool	1.3	false
EMPTY_TANK	Bool	1.4	false
COOLER_ON	Bool	1.5	false
COOLER_OFF	Bool	1.6	false
REQ	Bool	1.7	false
CONT	Bool	2.0	false
LEN	UInt	4.0	0
DATA_1	String	6.0	‘’
COM_RST	Bool	262.0	false
DONE	Bool	262.1	false
BUSY	Bool	262.2	false
ERROR	Bool	262.3	false
STATUS	Word	264.0	16#0

Αντίστοιχα, το Data Block 2 (DB4) περιέχει επίσης μεταβλητές οι οποίες χρησιμοποιούνται στο κυρίως πρόγραμμα και αποτελούν τα στοιχεία που μεταδίδονται στην android εφαρμογή και την HMI. Πιο συγκεκριμένα, όπως φαίνεται στον Πίνακα 2.2 στον οποίο φαίνεται η λίστα των μεταβλητών, οι μεταβλητές που περιέχονται σε αυτό το Data Block είναι το σημείο ρύθμισης (Setpoint) της θερμοκρασίας, ο χρόνος καθαρισμού και ο χρόνος μίξης, στις οποίες δίνεται μια αρχική τιμή 5°C στο σημείο ρύθμισης και 10 δευτερόλεπτα στους χρόνους καθαρισμού και μίξης, για την καλύτερη λειτουργία του κώδικα και για να υπάρχει η βεβαιότητα ότι η HMI και η android εφαρμογή μπορούν να επικοινωνήσουν και να διαβάσουν τιμές από το PLC. Ακόμα, περιέχονται μεταβλητές στις οποίες γίνεται προσωρινή ή μόνιμη αποθήκευση τιμών καθώς και οι αντίστοιχες μεταβλητές που απαιτούνται για τη σωστή λειτουργία του FC2.

Πίνακας 2.2: Ο πίνακας μεταβλητών του Data Block 2.

Name	Data Type	Offset	Start Value
TEMP_SP1	Word	0.0	16#5
CLEANING_TIME	Word	2.0	16#A
MIXING_TIME	Word	4.0	16#A
REAL_TEMP	Real	6.0	0.0

REAL_TEMP2	DInt	10.0	0
REQ	Bool	14.0	false
CONT	Bool	14.1	false
LEN	UInt	16.0	0
DATA_1	String	18.0	‘’
COM_RST	Bool	274.0	false
DONE	Bool	274.1	false
BUSY	Bool	274.2	false
ERROR	Bool	274.3	false
STATUS	Word	276.0	16#0
TEMP_SP	String	278.0	‘’
MIX_TIME	String	534.0	‘’
MIX_INT	Int	790.0	0
MIX_MILLIS	Int	792.0	0
TEMP_INT	Int	794.0	0
TEMP_REAL	Real	796.0	0.0
CLEAN_INT	Int	800.0	0
CLEAN_MILLIS	Int	802.0	0
SCALED_TEMP	Real	804.0	0.0

Παρόμοια λειτουργία με τα Data Blocks που αναφέρθηκαν έχει και το πεδίο PLC tags, μέσα στο οποίο περιέχονται οι ετικέτες (ή ονομασίες) για κάθε είσοδο, έξοδο, θέση μνήμης που έχει χρησιμοποιηθεί στο πρόγραμμα. Η λίστα φαίνεται στον Πίνακα 2.3 και κάθε μια ετικέτα ονομάζεται διαφορετικά, έτσι ώστε να εξηγείται καλύτερα η λειτουργία της. Αυτές οι ετικέτες μπορούν να αντιπροσωπεύουν τις εισόδους ή τις εξόδους του PLC, όπως φαίνεται, ή θέσεις μνήμης οι οποίες χρησιμοποιούνται μέσα στο πρόγραμμα. Ωστόσο, δεν είναι δυνατή η απευθείας ανάγνωση τους από την android εφαρμογή, κάτι που έκανε αναγκαία την ύπαρξη των Data Blocks και την ανάμιξη τους στις λειτουργίες του προγράμματος. [6][8]

Πίνακας 2.3: Η λίστα PLC Tags.

Name	Tag Table	Data Type	Address
EM_STOP_EXT	Default	Bool	%I0.0
START_MIXER_EXT	Default	Bool	%I0.1
STOP_MIXER_EXT	Default	Bool	%I0.2
START_LOAD_EXT	Default	Bool	%I0.3
STOP_LOAD_EXT	Default	Bool	%I0.4
START_CLEAN_EXT	Default	Bool	%I0.5
STOP_CLEAN_EXT	Default	Bool	%I0.6
EMPTY_TANK_SENSOR	Default	Bool	%I0.7
REAL_TEMPERATURE	Default	Int	%IW64
MIXER_STARTED	Default	Bool	%Q0.0
LOADING_STARTED	Default	Bool	%Q0.1
CLEANING_STARTED	Default	Bool	%Q0.2
COOLING_ON	Default	Bool	%Q0.3

MIXER_TIMER_ON	Default	Bool	%M36.3
EMPTY_TANK_TIMER_ON	Default	Bool	%M36.4
CLEANING_TIMER_ON	Default	Bool	%M36.5
MIXING_INTERCHANGE	Default	Bool	%M36.7
MIXER_TIMER_OFF	Default	Bool	%M37.0

Τέλος, στα μπλοκ περιέχονται και τα μπλοκ συστήματος, όπως φαίνεται στην Εικόνα 2.1, τα οποία δημιουργούνται αυτόματα, μόλις δημιουργηθεί κάποιος Timer για παράδειγμα και παίρνουν την ονομασία που έχει επιλεγεί για τον κάθε Timer. Στο συγκεκριμένο πρόγραμμα δεν έχει γίνει κάποια αλλαγή μέσα σε αυτά τα μπλοκ, καθώς οι Timers που έχουν χρησιμοποιηθεί έχουν ρυθμιστεί απευθείας μέσα από το κυρίως πρόγραμμα και η λειτουργία τους θα εξηγηθεί παρακάτω.

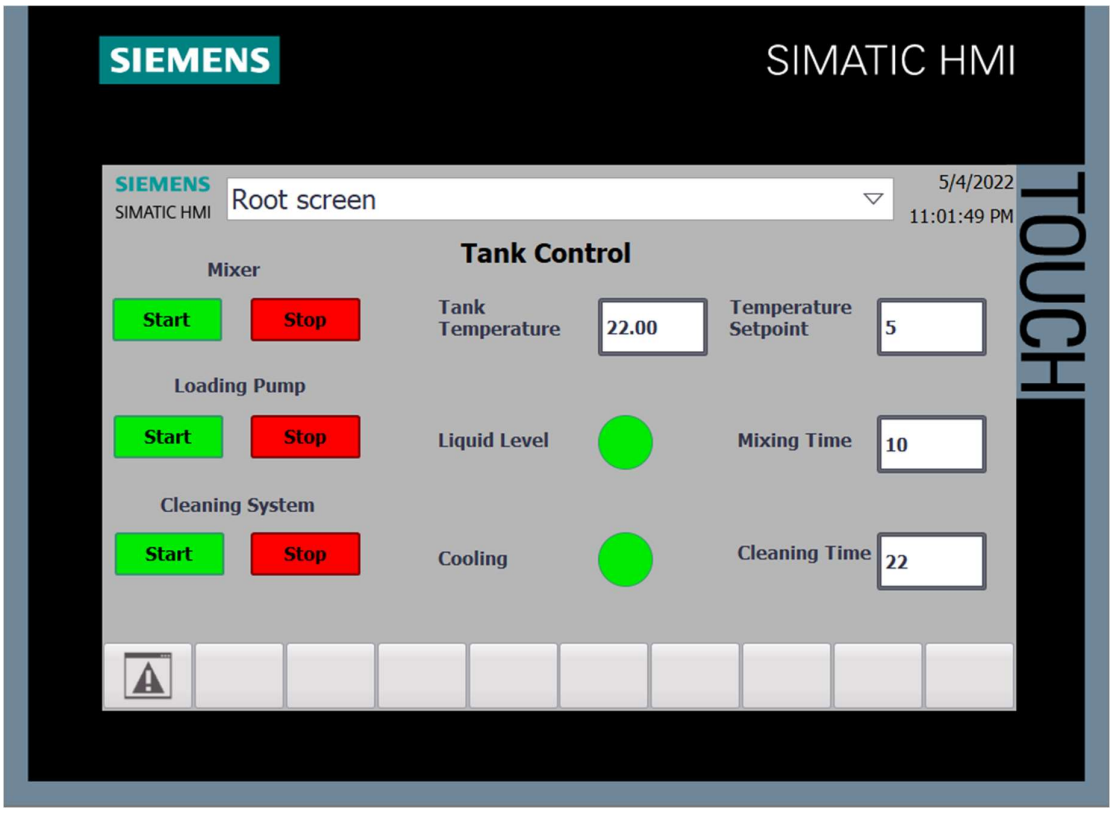
Στο Main Block βρίσκεται ο κυρίως κώδικας, ο οποίος περιέχει τη μορφοποίηση του αυτοματισμού, αλλά και τα blocks τα οποία εξηγήθηκαν σε προηγούμενες παραγράφους. Στο Παράρτημα Α βρίσκεται όλος ο κώδικας του Main Block, όπου φαίνονται όλα τα Networks, μερικά από τα οποία εξηγήθηκαν παραπάνω. Παρακάτω σε αυτήν την ενότητα ωστόσο, υπάρχουν εικόνες στις οποίες φαίνονται σημαντικά σημεία του κώδικα, τα οποία δεν εξηγήθηκαν έως τώρα. Στις επόμενες παραγράφους γίνεται αναφορά σε όλα τα Networks που περιέχει το Main Block.

Το Network 1 περιέχει τον κώδικα που αφορά την εκκίνηση του αναδευτήρα, ενώ το Network 2 αφορά την εκκίνηση του αυτοματισμού αναπλήρωσης της δεξαμενής και το Network 3 αυτήν του συστήματος καθαρισμού. Όπως φαίνεται λαμβάνονται υπόψιν όλοι οι περιορισμοί και κανόνες που αναφέρθηκαν παραπάνω. Τα Networks 4 και 5 αφορούν την ενεργοποίηση των δύο Function Blocks, FC1 και FC2, τα οποία αφορούν την TCP/IP επικοινωνία και των οποίων οι ιδιότητες αναφέρθηκαν ήδη. Στα Networks 6 μέχρι 11 φαίνεται η μετατροπή του επιθυμητού χρόνου που εισάγει ο χρήστης σε ακέραιο, ο πολλαπλασιασμός του για να μετατραπεί σε milliseconds και στη συνέχεια η μετατροπή σε μονάδα χρόνου, έτσι ώστε να εφαρμοστεί στα χρονικά ‘’MIXING TIMER’’ και ‘’MIXING TIMER 2’’. Με το πρώτο ‘’START’’ ενεργοποιείται το πρώτο χρονικό, λειτουργεί για όσα δευτερόλεπτα έχει προκαθοριστεί και μετά το πέρας του χρόνου απενεργοποιεί τον αναδευτήρα, ενώ ταυτόχρονα ενεργοποιεί το δεύτερο χρονικό. Στη δεύτερη περίπτωση, για τον ίδιο προκαθορισμένο χρόνο, ο αναδευτήρας παραμένει απενεργοποιημένος και ξεκινά εκ νέου μετά το πέρας της καθυστέρησης. Τα χρονικά λειτουργούν κυκλικά, μέχρι να πιεστεί κάποιο στοπ. Στο Network 11 περιέχεται η συνδεσμολογία για την ενεργοποίηση μιας θέσης μνήμης, η οποία ενεργοποιείται με το πρώτο ‘’START’’ και παραμένει ενεργοποιημένη, έτσι ώστε μετά την εναλλαγή των χρονικών, η λειτουργία του αναδευτήρα να ξεκινά αυτόματα. Η απενεργοποίηση της εξαρτάται από το αν πιεστεί κάποιο από τα στοπ. Στο επόμενο Network (Network 12), με την ενεργοποίηση του Stop έκτακτης ανάγκης (Emergency Stop), ενεργοποιείται επίσης η μεταβλητή ‘’EMERGENCY ACTIVE’’ η οποία βρίσκεται στο Data Block 1 και μέσω αυτής αποστέλλεται η πληροφορία ότι έχει πιεστεί το συγκεκριμένο πλήκτρο, στην android εφαρμογή. Τα Network 13 και 14 αφορούν την καθυστέρηση που εφαρμόζεται αφού ενεργοποιηθεί ο αισθητήρας χαμηλής στάθμης. Η καθυστέρηση αυτή χρησιμοποιείται, όπως αναφέρθηκε, για να αποφεύγονται στιγμιαίες ενεργοποιήσεις ή απενεργοποιήσεις των blocks τα οποία σχετίζονται με τη στάθμη της δεξαμενής. Η μεταβλητή ‘’EMPTY TANK’’ η οποία βρίσκεται κι αυτή στο Data Block 1 χρησιμοποιείται για να ενημερώνει την οθόνη HMI και την android εφαρμογή ταυτόχρονα. Στη συνέχεια, γίνεται η εισαγωγή της πραγματικής θερμοκρασίας από τις μετρήσεις του αισθητήρα, στο Network 15, με την εισαγωγή και αρχικοποίηση του FC3. Η τιμή εξόδου του, η οποία αποτελεί την πραγματική θερμοκρασία μεταφέρεται σε μια νέα μεταβλητή στο Network 16, μέσω του

block ‘’MOVE’’ και η οποία χρησιμοποιείται για τη σύγκριση με το ζητούμενο όριο, αλλά και για τη μετάδοση της πραγματικής θερμοκρασίας στην HMI και στην εφαρμογή. Στο Network 17, γίνεται η μετατροπή του ορίου από τον χρήστη, σε ‘’Real’’ μεταβλητή, έτσι ώστε να μπορέσει να συγκριθεί με την πραγματική θερμοκρασία. Έπειτα, τα Networks 18, 19 και 20 αφορούν την ενεργοποίηση της ψύξης, ανάλογα το αποτέλεσμα της σύγκρισης της πραγματικής θερμοκρασίας, με τη ζητούμενη από το χρήστη. Η καθυστέρηση που χρησιμοποιείται εξυπηρετεί τον αντίστοιχο σκοπό που εξυπηρετεί κι αυτή που χρησιμοποιείται στην προειδοποίηση χαμηλής στάθμης, όπως εξηγήθηκε παραπάνω. Τέλος, στα Networks 21 με 24, η τιμή του χρόνου που ορίζει ο χρήστης για το σύστημα καθαρισμού μετατρέπεται σε ακέραιο, πολλαπλασιάζεται για να μετατραπεί σε milliseconds και έπειτα μετατρέπεται σε τιμή χρόνου, η οποία εισάγεται στον timer ‘’CLEANING TIMER’’. [6][9]

2.2.4 Ανάλυση HMI (Human Machine Interface) και του Πίνακα Αυτοματισμού

Υπό κανονικές συνθήκες, η HMI θα ήταν μέρος του πίνακα αυτοματισμού και για αυτό θα αναλυθούν και τα δύο στην ίδια ενότητα. Στην περίπτωση της παρούσας εργασίας παρουσιάζεται μόνο η προσομοίωση μιας τέτοιας οθόνης. Στην HMI έχουν συμπεριληφθεί οι εκκινήσεις των τριών εξόδων, η απεικόνιση της θερμοκρασίας, της κατάστασης της ψύξης (ON/OFF), η προειδοποίηση χαμηλής στάθμης, καθώς και τα τρία πεδία εισαγωγής τιμών, θερμοκρασίας και χρόνων καθυστέρησης.



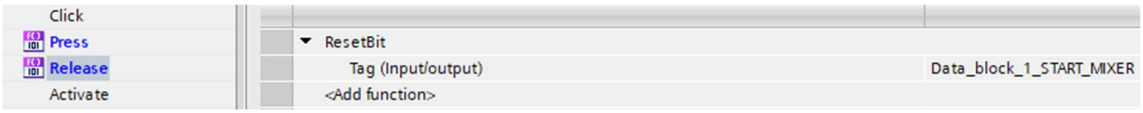
Εικόνα 2.8: Η προσομοίωση της HMI.

Η HMI επικοινωνεί με το PLC παράλληλα με τα εξωτερικά πλήκτρα και την android εφαρμογή. Στα αριστερά της φαίνονται τα πλήκτρα ‘’START’’ και ‘’STOP’’, τα οποία αφορούν τις τρείς από τις τέσσερις εξόδους του συστήματος. Κάθε ένα πλήκτρο λειτουργεί σαν στιγμιαίο, κάτι που σημαίνει πως οι ιδιότητες τους πρέπει να προσαρμοστούν αναλόγως. Πιο συγκεκριμένα, στην καρτέλα ‘’Properties’’ για κάθε ένα πλήκτρο, δίνονται τα συγκεκριμένα χαρακτηριστικά μορφοποίησης της

εμφάνισης του πλήκτρου στην οθόνη. Στην καρτέλα ‘’Events’’ καθορίζεται η λειτουργία του πλήκτρου. Για να λειτουργεί σαν στιγμιαίο, τότε πρέπει να επιλεγούν οι δύο ιδιότητες ‘’Press’’ και ‘’Release’’, με τις οποίες καθορίζεται η συμπεριφορά του πλήκτρου όταν πιεστεί και όταν ελευθερωθεί. Αφού λοιπόν γίνουν οι απαραίτητες ρυθμίσεις για το χρώμα, το μέγεθος και το μέγεθος κειμένου του κάθε πλήκτρου, στην καρτέλα ‘’Events’’, επιλέγοντας το ‘’Press’’, δίνεται η δυνατότητα να γίνουν διάφορες επιλογές λειτουργιών. Στη συγκεκριμένη περίπτωση, η επιλογή πρέπει να είναι η ‘’SetBit’’, μέσω της οποίας μπορεί να καθοριστεί το ποια θέση θα γίνει αληθής (‘’TRUE’’), όπως φαίνεται στην Εικόνα 2.9α. [7]

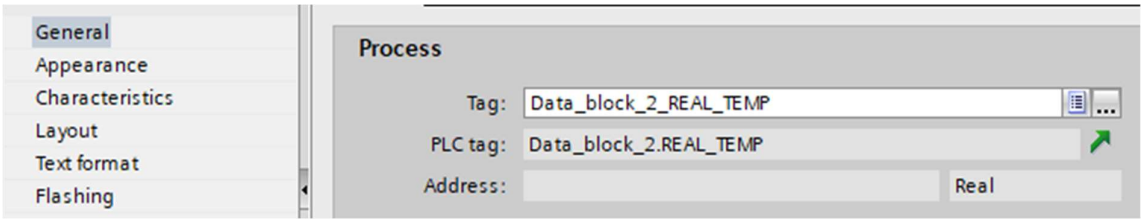
Εικόνα 2.9α: Η ρύθμιση ‘’SetBit’’.

Για να επανέλθει η θέση που έχει επιλεγεί ξανά στην αρχική της κατάσταση (‘’FALSE’’), τότε στο ‘’Release’’ πρέπει να επιλεγεί η λειτουργία ‘’ResetBit’’, όπως φαίνεται στην Εικόνα 2.9β, η οποία λειτουργεί με αυτόν ακριβώς τον τρόπο. Η επαναφορά στην αρχική κατάσταση για κάθε θέση είναι κρίσιμη, καθώς δεν πρέπει να παραμένει αληθής η τιμή ενός ‘’STOP’’, αφού επιλεγεί, γιατί αυτό θα εμποδίσει την εκ νέου επανεκκίνηση της εξόδου, καθώς το στοπ θα παραμένει ενεργοποιημένο. Επίσης, στην περίπτωση που κάποιο ‘’START’’ παραμείνει ενεργοποιημένο, τότε μπορεί μετά από επαναφορά κάποιου στοπ ή του στοπ έκτακτης ανάγκης να υπάρξει ανεπιθύμητη εκκίνηση του συστήματος, κάτι που αποτελεί κίνδυνο και για τον χρήστη και των ατόμων κοντά στη δεξαμενή, αλλά και για ολόκληρο το σύστημα. Και τα έξι πλήκτρα που φαίνονται, λειτουργούν με τον ίδιο τρόπο που αναλύθηκε. [7]

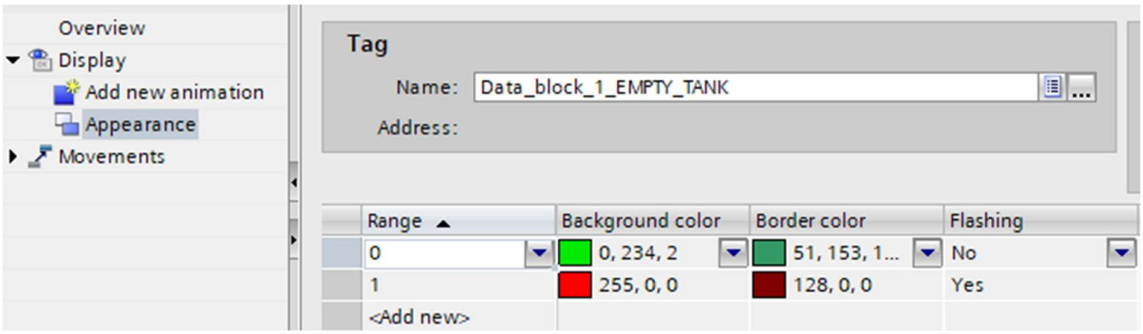


Εικόνα 2.9β: Η ρύθμιση ‘’ResetBit’’.

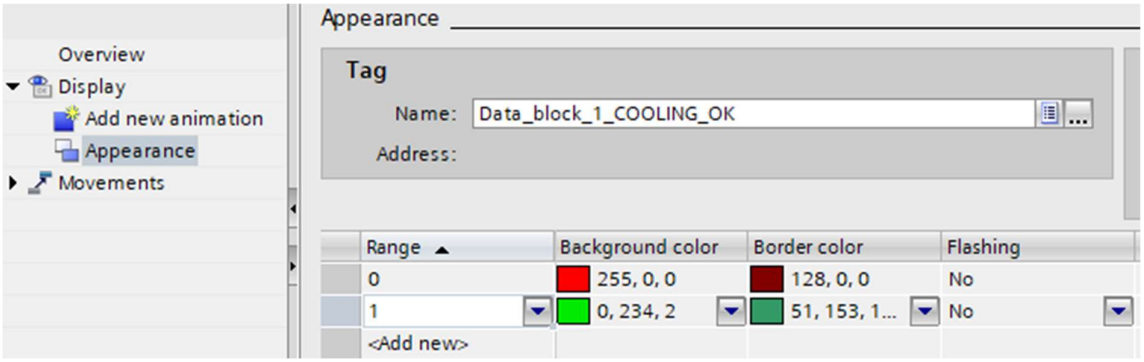
Στη μέση της HMI φαίνεται το πεδίο απεικόνισης της θερμοκρασίας, καθώς και τα πεδία επιπέδου στάθμης και κατάστασης της ψύξης. Για το πεδίο της θερμοκρασίας ορίζεται μια θέση από την οποία πρέπει να λαμβάνει την πληροφορία της τιμής της θερμοκρασίας και αυτή καθορίζεται στην καρτέλα ‘’Properties’’ στην επιλογή ‘’General’’, όπως φαίνεται στην Εικόνα 2.10α. Ο τύπος ορίζεται σαν ‘’Output’’ καθώς δε λαμβάνεται κάποια τιμή από τον χρήστη σε αυτό το πεδίο. Στη συνέχεια, τα πεδία της στάθμης και της κατάστασης της ψύξης λειτουργούν με τον ίδιο τρόπο. Δηλαδή, όπως φαίνεται στην Εικόνα 2.10β και 2.10γ, στην καρτέλα ‘’Animations’’ στην επιλογή ‘’Display’’ καθορίζεται η θέση από την οποία λαμβάνει κάθε πεδίο την κατάσταση του (‘’TRUE’’ ή ‘’FALSE’’) και παρακάτω ρυθμίζονται οι ιδιότητες τους στις δύο περιπτώσεις, όταν η τιμή της θέσης που λαμβάνεται είναι ‘’0’’ και όταν είναι ‘’1’’. [7]



Εικόνα 2.10α: Η ρύθμιση για την λήψη της θερμοκρασίας στο αντίστοιχο πεδίο.

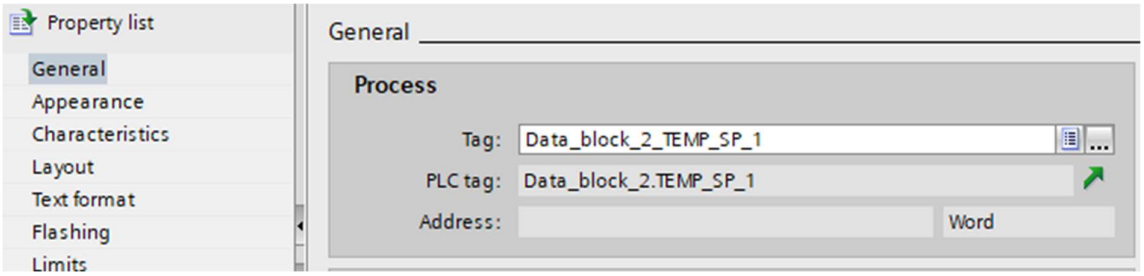


Εικόνα 2.10β: Η ρύθμιση του ενδεικτικού της στάθμης υγρού.



Εικόνα 2.10γ: Η ρύθμιση του ενδεικτικού της κατάστασης της ψύξης.

Τέλος, στα δεξιά της οθόνης υπάρχουν τρία πεδία, στα οποία απεικονίζεται η τιμή του καθενός και ταυτόχρονα ο χρήστης μπορεί να δώσει τις επιθυμητές τιμές οι οποίες εγγράφονται στις αντίστοιχες θέσεις. Το πρώτο πεδίο αφορά τη ρύθμιση του επιθυμητού ορίου θερμοκρασίας της δεξαμενής, το δεύτερο το χρονικό όριο ανάδευσης και το τρίτο το αντίστοιχο χρονικό όριο καθαρισμού. Στην Εικόνα 2.11 φαίνεται η ρύθμιση για το πρώτο πεδίο, η οποία είναι ίδια και για τα άλλα δύο πεδία, με το μόνο που διαφοροποιείται να είναι η θέση από την οποία λαμβάνονται και στην οποία εγγράφονται οι τιμές. Στη ρύθμιση ‘’Type’’ θα πρέπει να είναι επιλεγμένο το ‘’Input/Output’’ έτσι ώστε το πεδίο να λειτουργεί και με τους δύο τρόπους. [7]



Εικόνα 2.11: Η ρύθμιση του πεδίου ‘’Temprature Setpoint’’.

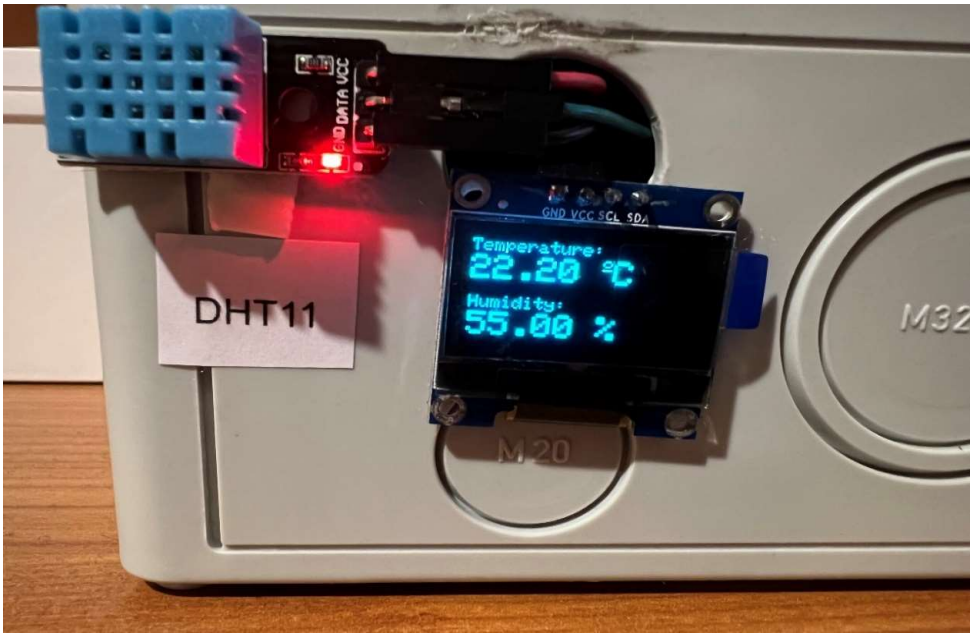
Σύμφωνα με τις απαιτούμενες λειτουργίες υλοποιήθηκε και ο πίνακας αυτοματισμού (Εικόνα 2.18). Αποτελείται από ηλεκτρονόμους (ρελέ), πλήκτρα (buttons ή μπουτόν) την τροφοδοσία του PLC, καθώς και από έναν υπο-πίνακα (Εικόνα 2.12) στον οποίο βρίσκεται το κύκλωμα του ESP32 και της ενισχυτικής βαθμίδας. Εκτός πίνακα βρίσκεται και το αισθητήριο χαμηλής στάθμης, το οποίο στην πραγματικότητα είναι ένα φωτοκύτταρο με τάση λειτουργίας 230V, το οποίο λειτουργεί σαν διακόπτης ON/OFF (‘’0’’ και ‘’1’’). Να σημειωθεί εδώ πως η υλοποίηση του πίνακα έγινε για να προσομοιωθεί μια αντίστοιχη λειτουργία, η οποία θα αφορούσε πραγματική εφαρμογή. Μπορούν να

εφαρμοστούν αρκετές βελτιώσεις, όπως αναφέρεται και σε επόμενο κεφάλαιο, ωστόσο το κόστος ήταν ο κύριος παράγοντας ο οποίος εμπόδισε τη χρήση αναλογικού αισθητήρα για τον υπολογισμό των λίτρων και στάθμης, τη χρήση καλύτερου συστήματος μέτρησης και απεικόνισης της θερμοκρασίας, όπως και τη χρήση μιας οθόνης για την απεικόνιση ολόκληρου του συστήματος. Παρόλα αυτά, η εφαρμογή αποδίδει το επιθυμητό αποτέλεσμα και αποτελεί μια αρκετά καλή προσομοίωση ενός πραγματικού συστήματος.

Στην Εικόνα 2.18 φαίνεται ο πίνακας αυτοματισμού και τα εξαρτήματα που τον απαρτίζουν, ενώ το ηλεκτρολογικό του σχέδιο παρατίθεται στο Παράρτημα Δ. Η τροφοδοσία είναι μονοφασική, τη τάξης των 230V και καταλήγει στον μικροαυτόματο F1, ο οποίος παρέχει ασφάλεια σε όλο το κύκλωμα. Από εκεί τροφοδοτείται και το τροφοδοτικό PSU, το οποίο μετατρέπει την τάση σε 24VDC για την τροφοδοσία του PLC (CPU 1). Υπάρχουν τρεις επιλογές, ‘’MIXER’’, ‘’LOAD PUMP’’ και ‘’CLEANING SYSTEM’’ με δύο πλήκτρα το καθένα ‘’START’’ και ‘’STOP’’, χρώματος πράσινου και κόκκινου αντίστοιχα. Αποτελούν τα πλήκτρα μέσω των οποίων γίνεται η εκκίνηση και η παύση κάθε εξόδου αντίστοιχα. Κάθε ένα πλήκτρο καταλήγει σε μια συγκεκριμένη είσοδο του PLC, κάτι που φαίνεται στον Πίνακα 2.3, στο μπλοκ των PLC Tags. Οι έξοδοι του PLC αντίστοιχα, όταν ενεργοποιούνται, ενεργοποιούν και έναν ηλεκτρονόμο (ρελέ), με τάση πηνίου 24VDC, ανάλογα την έξοδο που έχει ενεργοποιηθεί. Δηλαδή, ο ηλεκτρονόμος K1 ενεργοποιείται από την έξοδο Q0.0, ο οποίος αφορά τον έλεγχο του αναδευτήρα (‘’MIXER’’). Ο K2 λαμβάνει το σήμα για την ενεργοποίηση του από την έξοδο Q0.1 και αφορά τον έλεγχο του συστήματος αναπλήρωσης, ενώ ο K3 ενεργοποιείται από την έξοδο Q0.2 του PLC και ελέγχει το σύστημα καθαρισμού. Η έξοδος Q0.3 ενεργοποιείται από την εκκίνηση της ψύξης μέσα στον κώδικα, ωστόσο δεν ενεργοποιείται κάποιος εξωτερικός ηλεκτρονόμος. Ο ηλεκτρονόμος K4 ενεργοποιείται από το φωτοκύτταρο που προσομοιώνει το αισθητήριο στάθμης και μια από τις επαφές του χρησιμοποιείται για να σταλεί το σήμα ενεργοποίησης του στην αντίστοιχη είσοδο του PLC. Επίσης, μια κανονικά ανοικτή (NO – Normally Open) και μια κανονικά κλειστή (NC – Normally Closed) επαφή καθενός ηλεκτρονόμου, χρησιμοποιείται για την ενεργοποίηση των LED λυχνίων, οι οποίες βρίσκονται στο πλήκτρο ‘’START’’ και ‘’STOP’’ αντίστοιχα. Τέλος, το στοπ έκτακτης ανάγκης (‘’EMERGENCY STOP’’) καταλήγει στην είσοδο I0.0 του PLC.

Η μετατροπή της θερμοκρασίας σε τάση και η μετάδοση της στην αναλογική είσοδο του PLC, γίνεται από ένα ESP32 και μια ενισχυτική βαθμίδα. Στο Παράρτημα Γ περιέχεται ο κώδικας του ESP32 και παρακάτω γίνεται ανάλυση της ροής και των σημαντικών κομματιών του.

Αρχικά, μετά τη δήλωση των βιβλιοθηκών, οι οποίες χρειάζεται να συμπεριληφθούν, και τον απαραίτητο καθορισμό των pins για την ανάγνωση και μετάδοση τιμών, ακολουθούν οι δύο συναρτήσεις, οποίες φαίνονται στην Εικόνα 2.13 και Εικόνα 2.14, οι οποίες επιστρέφουν την τιμή της θερμοκρασίας και της υγρασίας αντίστοιχα. Μέσω της βιβλιοθήκης του DHT11, η οποία περιλαμβάνεται μέσα στο Arduino IDE, μπορεί να γίνει ανάγνωση θερμοκρασίας και υγρασίας με τις εντολές dht.readTemperature() και dht.readHumidity() αντίστοιχα. [15]



Εικόνα 2.12: Το αισθητήριο θερμοκρασίας DHT11 και η ένδειξη θερμοκρασίας και υγρασίας.

Έπειτα, μέσα στην void setup() γίνεται η αρχικοποίηση του αισθητήρα και της οθόνης και στη συνέχεια, μέσα στην void loop() βρίσκεται ο κώδικας που επαναλαμβάνεται. Εκεί, οι τιμές θερμοκρασίας και υγρασίας που επιστρέφονται από τις δύο συναρτήσεις, αποθηκεύονται σε δύο ξεχωριστές μεταβλητές, οι οποίες τυπώνονται στην OLED οθόνη, όπως φαίνεται στην Εικόνα 2.15β. [15]

```
String readDHTTemperature() {
  // Read temperature as Celsius
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}
```

Εικόνα 2.13: Η συνάρτηση για την ανάγνωση θερμοκρασίας.

```
String readDHTHumidity() {
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(h);
    return String(h);
  }
}
```

Εικόνα 2.14: Η συνάρτηση για την ανάγνωση υγρασίας.

Η τιμή της θερμοκρασίας είναι αυτή που καθορίζει την τιμή της αναλογικής εξόδου του ESP32, η οποία λαμβάνεται από το pin 25. Ο υπολογισμός της αναλύεται στην παρακάτω σχέση, όπου ισχύει ότι:

$$m = \frac{255}{50} = 5.1$$

$$y_1 = 0, x_1 = 0$$

$$y - y_1 = m \cdot (x - x_1) \Rightarrow y - 0 = 5.1 \cdot (x - 0) \Rightarrow y = 5.1 \cdot x \quad (2.1)$$

Η μεταβλητή m είναι ίση με τη διαίρεση των δύο μέγιστων τιμών, αφενός των bit που μεταδίδονται από την αναλογική έξοδο και της θερμοκρασίας. Η μέγιστη τιμή της αναλογικής εξόδου καθορίζεται από την ανάλυση (resolution) της αναλογικής εξόδου. Η έξοδος του ESP32 έχει ανάλυση 8 bit και συνεπώς η μέγιστη τιμή είναι 2^8. Επίσης, οι τιμές y_1 και x_1 αποτελούν τις ελάχιστες τιμές των δύο μεγεθών, ανάλυσης και θερμοκρασίας. Το αποτέλεσμα που προκύπτει αποθηκεύεται σε μια μεταβλητή, όπως φαίνεται στον κώδικα της Εικόνας 2.17 και μεταδίδεται στην αναλογική έξοδο με την εντολή dacWrite(ADC, Value), η οποία στη σύνταξη της απαιτεί το pin (ADC) που είναι ορισμένη η αναλογική έξοδος, σε αυτή την περίπτωση το pin 26 και την τιμή (Value) που πρέπει να μεταδοθεί.

$$V_{OUT} = \alpha \cdot V_{IN} + \beta \Rightarrow 10VDC = \alpha \cdot 3.2VDC + 0 \Rightarrow \alpha = \frac{10VDC}{3.2VDC} \Rightarrow \alpha = 3.1 \quad (2.2)$$

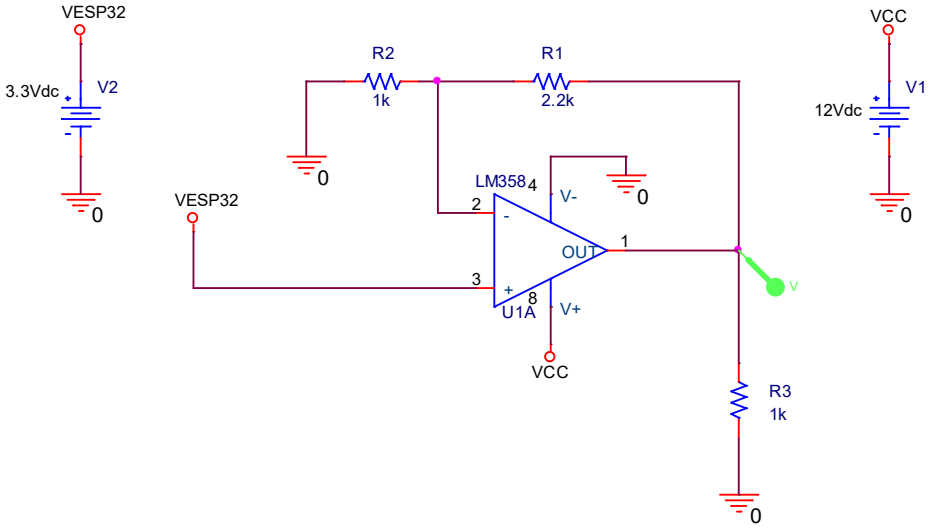
$$\alpha = 1 + \frac{R1}{R2} \Rightarrow 3.1 = 1 + \frac{R1}{1k\Omega} \Rightarrow 3.1 - 1 = \frac{R1}{1k\Omega} \Rightarrow R1 = 2.1k\Omega \quad (2.3)$$

Η αναλογική τιμή κυμαίνεται από 0V έως περίπου 3,3V, τα οποία πρέπει να ενισχυθούν, έτσι ώστε η ελάχιστη τιμή να είναι τα 0V αλλά η μέγιστη να φτάνει τα 10V. Αυτό γίνεται από την ενισχυτική βαθμίδα, η συνδεσμολογία της οποίας φαίνεται στην Εικόνα 2.16. Ο σχεδιασμός έγινε με τη χρήση του PSPICE και ο υπολογισμός των αντιστάσεων έγινε με βάση τη δεδομένη είσοδο και την επιθυμητή έξοδο και φαίνεται στις σχέσεις 2.2 και 2.3. Επίσης, χρησιμοποιήθηκε εξωτερική

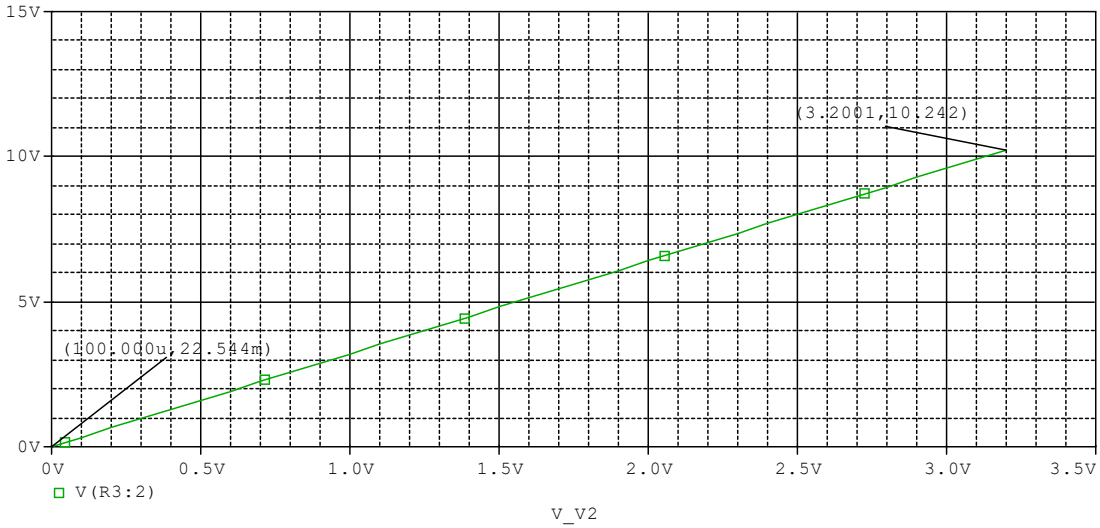
τροφοδοσία 12VDC για τον LMP358, έτσι ώστε να μπορέσει να αποδώσει τη ζητούμενη έξοδο. [2][15]

```
float y = 5.1*t;

dacWrite(26,y);
delay(1000);
```

Εικόνα 2.15: Ο κώδικας για τον υπολογισμό και τη μετάδοση της αναλογικής τιμής.



Εικόνα 2.16: Η συνδεσμολογία του κυκλώματος στο PSPICE.



Εικόνα 2.17: Η έξοδος της ενισχυτικής βαθμίδας σε σχέση με την είσοδο της.

2.3 Επίλογος

Σε αυτό το κεφάλαιο έγινε ανάλυση του τρόπου λειτουργίας του κώδικα του PLC, των κανόνων που ισχύουν καθώς και των προϋποθέσεων οι οποίες πρέπει να τηρούνται, έτσι ώστε να εξασφαλίζεται η ορθή λειτουργία. Έπειτα, κάθε κομμάτι (network) κώδικα αναλύθηκε ξεχωριστά, για την καλύτερη κατανόηση του αυτοματισμού. Αναλύθηκε επίσης η δομή της HMI – Human Machine Interface, η οποία επικοινωνεί με το πρόγραμμα παράλληλα με τον πίνακα αυτοματισμού και την android εφαρμογή. Τέλος, έγινε παρουσίαση και επεξήγηση του πίνακα, η ανάλυση της λειτουργίας των επιμέρους υλικών και η ανάλυση του κώδικα του ESP32, οι τιμές που παρέχει και η λειτουργία της ενισχυτικής βαθμίδας και ο υπολογισμός της, έτσι ώστε η εφαρμογή να γίνει κατανοητή σε όλα τα επίπεδα.

Εικόνα 2.18: Ο πίνακας αυτοματισμού.

Κεφάλαιο 3ο: Περιγραφή της Android Εφαρμογής

3.1 Εισαγωγή

Όπως έχει ήδη αναφερθεί, το PLC ελέγχεται και από απόσταση, μέσω android εφαρμογής, μέσω της οποίας ο χρήστης μπορεί να ελέγχει τις λειτουργίες του PLC, να παρακολουθεί τη λειτουργία του συστήματος διαβάζοντας τιμές που τον αφορούν και να ρυθμίζει τα επιθυμητά όρια (setpoints) της θερμοκρασίας και των χρόνων λειτουργίας του αναδευτήρα και του συστήματος καθαρισμού. Σε αυτήν την ενότητα γίνεται περιγραφή και ανάλυση των δυνατοτήτων της εφαρμογής και παρουσιάζεται λεπτομερώς η δομή του κώδικα, ο οποίος παρατίθεται στο Παράρτημα Β.

3.2 Αρχές Λειτουργίας

3.2.1 Δομή και Βασική Ροή

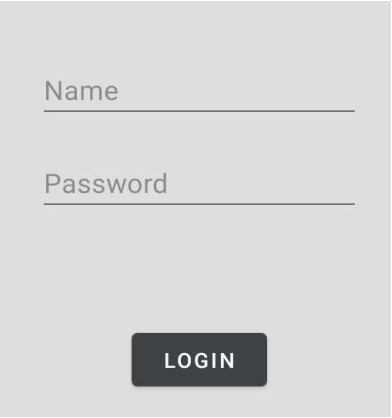
Η εφαρμογή υλοποιήθηκε στο Android Studio, σε γλώσσα Java και χωρίζεται σε Activities, μέσω των οποίων προγραμματίζεται ξεχωριστά η κάθε λειτουργία της εφαρμογής, οι οποίες λειτουργίες εξαρτώνται από το σε ποια οθόνη βρίσκεται ο χρήστης. Για κάθε Activity υπάρχει ο κώδικας που αφορά τη μορφοποίηση κάθε οθόνης, το που θα βρίσκονται τα εικονίδια, τι περιορισμούς θα έχουν σε σχέση με τα υπόλοιπα αντικείμενα που βρίσκονται επάνω στην οθόνη, καθώς και το τι ιδιότητες θα έχει κάθε ένα και έχει μορφή .xml. Υπάρχει επίσης και ο κυρίως κώδικας, στο οποίο γράφονται οι εντολές που καθορίζουν τη λειτουργία του κάθε Activity, όπως θα εξηγηθεί παρακάτω. [3][4]

Η εφαρμογή χωρίζεται σε έξι Activities και κάθε ένα αποτελεί διαφορετική λειτουργία, το οποίο όμως επικοινωνεί σε κάποιες περιπτώσεις με τα υπόλοιπα, για την ανταλλαγή πληροφοριών. Η ροή της εφαρμογής ακολουθεί τη λογική. Πιο συγκεκριμένα, με την εκκίνηση της εφαρμογής, το πρώτο Activity που εμφανίζεται, είναι αυτό στο οποίο ο χρήστης πρέπει να εισάγει τα διαπιστευτήρια του για να προχωρήσει στο χειρισμό της εφαρμογής. Αν αυτά είναι σωστά, προχωράει στην επόμενη οθόνη, στην οποία μπορεί να εισάγει την IP του PLC με το οποίο θέλει να συνδεθεί. Αν το πεδίο δεν είναι κενό, τότε προχωράει στην επόμενη οθόνη, η οποία αφορά την ανάγνωση και τον ορισμό δεδομένων. Αν η σύνδεση με το PLC είναι επιτυχής, θα μπορεί να διαβάσει τις ανάλογες τιμές. Στη συγκεκριμένη οθόνη, ο χρήστης έχει την επιλογή να αποσυνδεθεί, να αλλάξει Activity και να προχωρήσει είτε στον έλεγχο των εξόδων, είτε στην αλλαγή IP, είτε στην οθόνη που μπορεί να καθορίσει τα διαπιστευτήρια του. Στην πρώτη περίπτωση, αν αποσυνδεθεί, οδηγείται και πάλι στο πρώτο Activity, έτσι ώστε να μπορέσει να συνδεθεί εκ νέου. Αν επιλέξει την δεύτερη εναλλακτική, μπορεί να προχωρήσει στο Activity για τον έλεγχο των εξόδων του συστήματος. Σε αυτήν την περίπτωση, εμφανίζεται μια νέα οθόνη, η οποία αφορά των έλεγχο εκκίνησης και παύσης του αναδευτήρα, του συστήματος αναπλήρωσης και αυτού του καθαρισμού, μαζί με άλλες πληροφορίες, όπως εξηγείται αναλυτικά παρακάτω. Στην τρίτη περίπτωση, αν επιλέξει να αλλάξει IP διεύθυνση, τότε οδηγείται και πάλι στο αντίστοιχο Activity για να ορίσει νέα διεύθυνση. Τέλος, αν επιλέξει να αλλάξει διαπιστευτήρια, οδηγείται στο Activity το οποίο αφορά τη συγκεκριμένη εργασία.

3.2.2 Ανάλυση Λειτουργίας

Η ανάλυση της εφαρμογής παρακάτω γίνεται με βάση τη σειρά που εμφανίζεται κάθε Activity. Το πρώτο, λοιπόν, είναι το ‘’LoginActivity’’, το οποίο φαίνεται στην Εικόνα 3.1 και αφορά τη σύνδεση του χρήστη στην εφαρμογή. Αποτελείται από δύο πεδία, στα οποία ο χρήστης μπορεί να

πληκτρολογήσει το όνομα χρήστη και τον κωδικό και ένα πλήκτρο το οποίο πρέπει να πιέσει για να λειτουργήσει ο κώδικας που αφορά τον έλεγχο των διαπιστευτηρίων.



Εικόνα 3.1: Τα πεδία του ''LoginActivity''

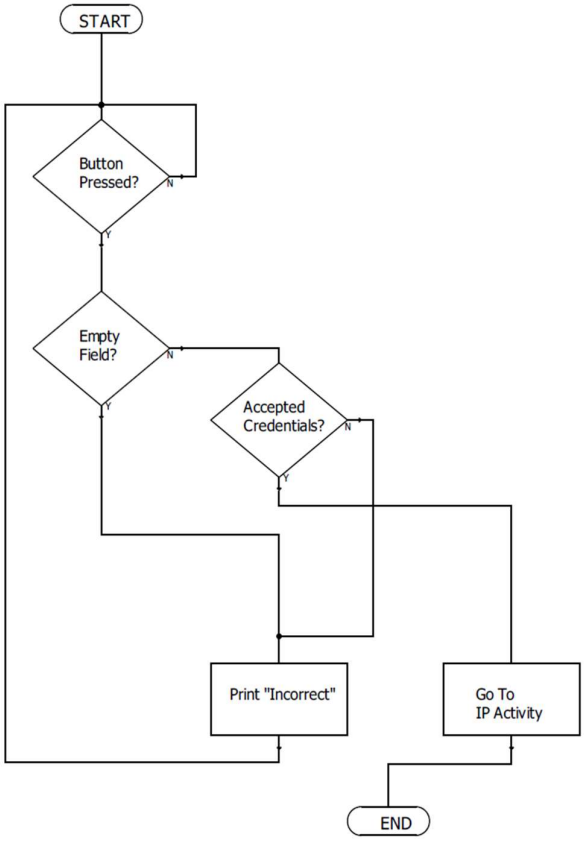
Αν πρόκειται για την πρώτη σύνδεση του χρήστη, τότε πρέπει να χρησιμοποιήσει τα ''Admin'' και ''1234'' για όνομα χρήστη και κωδικό, αντίστοιχα, τα οποία καθορίζονται μέσα στον κώδικα. Αξίζει να σημειωθεί πως τα παραπάνω επιλέχθηκαν για την ευκολότερη σύνδεση στην εφαρμογή και πως αν επρόκειτο η εφαρμογή να χρησιμοποιηθεί σε πραγματικές συνθήκες, τότε τα διαπιστευτήρια θα μετονομάζονταν έτσι ώστε να είναι δύσκολη η παραβίαση τους και ο χρήστης θα ενημερωνόταν εκ των προτέρων για τα προκαθορισμένα διαπιστευτήρια και πως να τα χρησιμοποιήσει και να τα αλλάξει στη συνέχεια. Εφόσον ο χρήστης πληκτρολογήσει τα σωστά διαπιστευτήρια, τότε οδηγείται στο επόμενο Activity. Αν όχι, τότε εμφανίζεται μήνυμα το οποίο τον ενημερώνει πως τα στοιχεία που παρείχε ήταν λανθασμένα. Η σύγκριση γίνεται μέσα στην ρουτίνα ''validate'', όπως φαίνεται στην Εικόνα 3.2, όπου φαίνεται ένα μέρος του κώδικα του ''LoginActivity'' και εφαρμόζεται με την επιλογή του κουμπιού ''LOGIN''.

```
public void validate (String userName, String userPassword){
    if (userName.isEmpty() || userPassword.isEmpty()){
        Toast.makeText(LoginActivity.this, "Empty Field",
Toast.LENGTH_LONG).show();  //Check for empty field
    }
    else {
        if ((userName.equals(loginName)) &&
(userPassword.equals(loginPass)) || (userName.equals("Admin")) &&
(userPassword.equals("1234"))) {
            Intent intent = new Intent(LoginActivity.this,
IPActivity.class);
            startActivity(intent);
        } else {
            Toast.makeText(LoginActivity.this, "Incorrect",
Toast.LENGTH_LONG).show();
        }
    }  //Check if credentials are the right ones
}
```

Εικόνα 3.2: Ο κώδικας της ρουτίνας ''validate''.

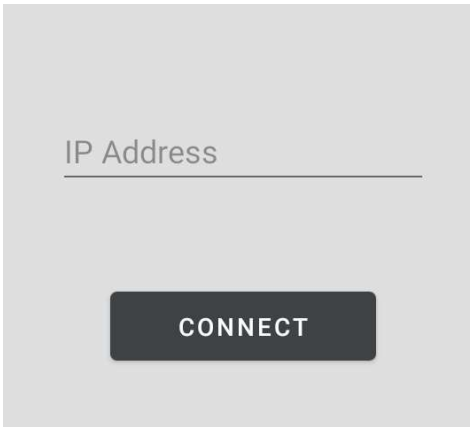
```
login.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        validate(name.getText().toString(), password.getText().toString());
    }
});  //Function call to check credentials
```

Εικόνα 3.3: Η εκτέλεση της ''validate'' με το πάτημα του ''LOGIN''.



Σχήμα 3.1: Το διάγραμμα ροής του ‘’LoginActivity’’.

Το επόμενο ‘’IPActivity’’ εμφανίζεται, με την προϋπόθεση τα διαπιστευτήρια που έχει εισάγει ο χρήστης, είναι σωστά. Εδώ, ο χρήστης μπορεί να πληκτρολογήσει την διεύθυνση IP του PLC με το οποίο επιθυμεί να συνδεθεί, όπως φαίνεται στην Εικόνα 3.4. Πιέζοντας το πλήκτρο ‘’CONNECT’’, αν το πεδίο είναι άδειο, τότε εμφανίζεται το μήνυμα που ενημερώνει το χρήστη να προσπαθήσει ξανά. Σε διαφορετική περίπτωση, η εφαρμογή ξεκινάει την προσπάθεια σύνδεσης στη συγκεκριμένη IP και ταυτόχρονα αποθηκεύει αυτήν την IP σε μια static String ‘’ipAddress’’, η οποία μπορεί να χρησιμοποιηθεί και στα υπόλοιπα Activity για την επικοινωνία με το PLC, καλώντας την ‘’getAddress’’, όπως φαίνεται στην Εικόνα 3.6.



Εικόνα 3.4: Το πεδίο εισαγωγής της IP διεύθυνσης.

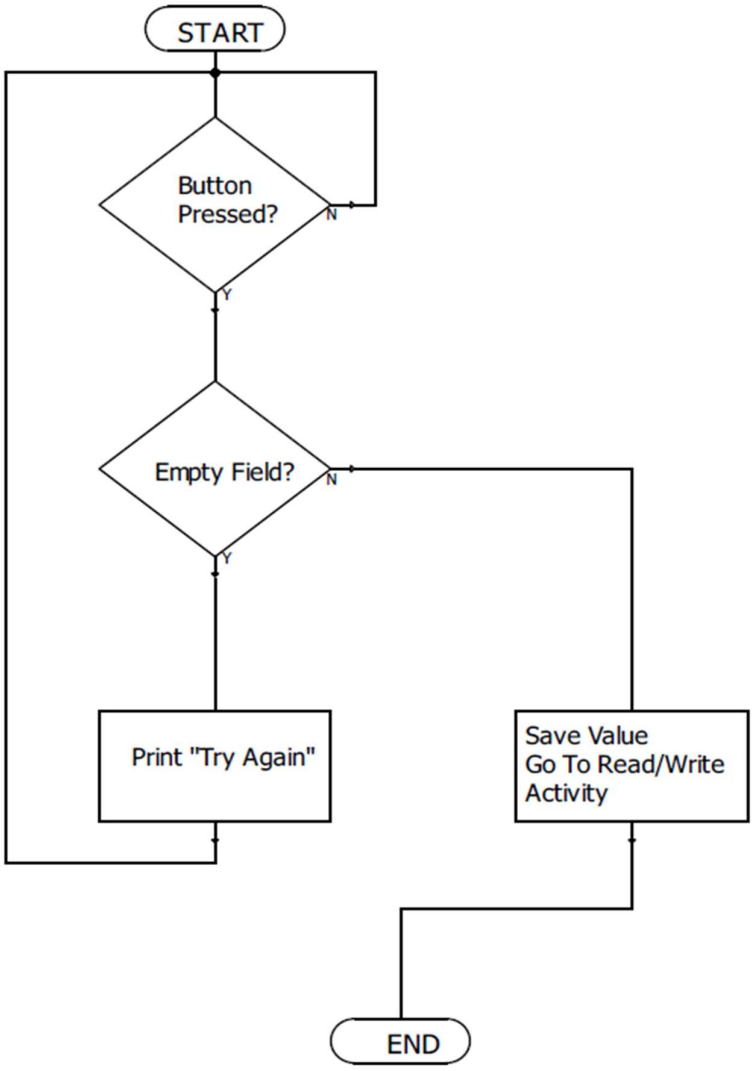
```
connect.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        ipcheck = address.getText().toString();

        if(TextUtils.isEmpty(ipcheck)) {
            Toast.makeText(IPActivity.this, "Try Again",
Toast.LENGTH_LONG).show();  //Check for empty field
        }

        else {

            ipAddress = ipcheck;
            Intent intent = new Intent(IPActivity.this,
ReadWriteActivity.class);
            startActivity(intent);
        }

    }

});
```

Εικόνα 3.5: Ο κώδικας που εκτελείται πιέζοντας το ‘’CONNECT’’.

```
public static String getAddress()
{

    return ipAddress;

}  //IP Address stored for other Activities to use
```

Εικόνα 3.6: Η αποθήκευση της διεύθυνσης IP.

Και πάλι, αν το πεδίο δεν είναι κενό, ο χρήστης οδηγείται σε ένα νέο Activity, μέσω του οποίου μπορεί να λάβει πληροφορίες από το PLC, οι οποίες αφορούν την πραγματική θερμοκρασία της δεξαμενής, το χρόνο που έχει καθοριστεί για την ανάδευση και το χρόνο για τον καθαρισμό. Μπορεί ακόμα να λάβει πληροφορίες για το αν η ψύξη λειτουργεί ή όχι και ποιο είναι το ζητούμενο όριο (setpoint) της θερμοκρασίας, στα αντίστοιχα πεδία. Ο κώδικας που αφορά την παροχή πληροφοριών από το PLC φαίνεται στην Εικόνα 3.7.



Σχήμα 3.2: Το διάγραμμα ροής του ‘’IPActivity’’.

Στον παρακάτω κώδικα, αρχικά γίνεται η σύνδεση στο PLC, μέσω της IP διεύθυνσης που έχει ορίσει ο χρήστης στο προηγούμενο IPActivity και η οποία στο συγκεκριμένο Activity είναι αποθηκευμένη στη μεταβλητή ‘’SetAddress’’. Η σύνδεση στο PLC γίνεται μέσω της βιβλιοθήκης Moka7 η οποία χρησιμοποιήθηκε για την επικοινωνία και ο τρόπος φαίνεται στις δύο πρώτες γραμμές του κώδικα της παρακάτω εικόνας. Η μεταβλητή ‘’readcheck’’ γίνεται ‘’0’’ όταν η σύνδεση είναι επιτυχής, και αυτή είναι η προϋπόθεση για την επιτυχημένη επικοινωνία. Σε διαφορετική περίπτωση, εκτελείται ο κώδικας που περιέχεται στην else, μέσω της οποίας τυπώνεται το σφάλμα που αφορά την αποτυχία σύνδεσης. Μέσα στην if δηλώνεται ένας buffer ‘’data1’’, στον οποίο αποθηκεύονται οι τιμές που λαμβάνονται από το PLC. Έπειτα, δηλώνεται το Data Block στο οποίο περιέχονται οι πληροφορίες που πρέπει να ληφθούν, από ποια θέση του να ξεκινήσει η ανάγνωση δεδομένων, το πλήθος τους και το που θα αποθηκευτούν. [4][8][17]

```
client.SetConnectionType(S7.S7_BASIC);

int readcheck = client.ConnectTo("" + SetAddress, 0, 1);//connect to given
address

if (readcheck == 0)  //check connection
{
    byte[] data1 = new byte[256];  //data buffer
    client.ReadArea(S7.S7AreaDB, 4, 0, 12, data1);  //where to read
    readdata1 = S7.GetFloatAt(data1, 6) + " °C";  //Temperature reading
    readdata2 = S7.GetWordAt(data1, 2) + " Seconds";//Mixing time reading
    readdata3 = S7.GetWordAt(data1, 4) + " Seconds";//Cleaning time reading
    readdata4 = S7.GetWordAt(data1,0) + " °C";  //Temperature setpoint
reading


    byte[] data2 = new byte[256];
    client.ReadArea(S7.S7AreaDB, 1, 0, 12, data2);
    CoolingBitCheck = S7.GetBitAt(data2,1,2);  //Check if cooling is on
}

else
{  //error printing
    readdata1 = "ERR: " + S7Client.ErrorText(readcheck);
    readdata2 = "ERR: " + S7Client.ErrorText(readcheck);
    readdata3 = "ERR: " + S7Client.ErrorText(readcheck);
    coolingState = "ERR: " + S7Client.ErrorText(readcheck);
}
client.Disconnect();
```

Εικόνα 3.7: Το κομμάτι κώδικα που αφορά την ανάγνωση πληροφοριών από το PLC.

Οι τιμές που λαμβάνονται αποθηκεύονται σε πέντε μεταβλητές, μια για κάθε τιμή και στην συνέχεια οι μεταβλητές αυτές χρησιμοποιούνται για την τύπωση των τιμών στα αντίστοιχα πεδία. Στην ‘’readdata1’’ αποθηκεύεται η τιμή της πραγματικής θερμοκρασίας, στην ‘’readdata2’’ η τιμή του χρόνου ανάδευσης, στην ‘’readdata3’’ η τιμή του χρόνου καθαρισμού σε δευτερόλεπτα, ενώ στη μεταβλητή ‘’readdata4’’ το setpoint θερμοκρασίας που έχει οριστεί. Τέλος, σε έναν διαφορετικό buffer ‘’data2’’, αποθηκεύεται η τιμή που αφορά την κατάσταση της ψύξης, η οποία βρίσκεται στο Data Block 1 και επιστρέφει ‘’0’’ ή ‘’1’’.

Η θέση που ορίζεται στον κώδικα, αφορά τη θέση στην οποία βρίσκεται η τιμή στον πίνακα του Data Block. Πιο συγκεκριμένα, στην Εικόνα 2.9 του προηγούμενου κεφαλαίου, στον πίνακα μεταβλητών του Data Block 2 φαίνεται η στήλη ‘’Offset’’. Ο αριθμός πριν την υποδιαστολή, είναι αυτός που ορίζει τη θέση. Για παράδειγμα, στον παραπάνω κώδικα στην εντολή ανάγνωσης της πραγματικής θερμοκρασίας η θέση καθορίζεται σε ‘’Pos: 6’’. Στην Εικόνα 2.9, η τιμή της πραγματικής θερμοκρασίας αποθηκεύεται σε μια ‘’Real’’ μεταβλητή με όνομα ‘’REAL_TEMP’’, στο Offset της οποίας αναγράφεται το ‘’6.0’’. Αυτό το ‘’6’’ είναι η θέση η οποία καθορίζεται και στον κώδικα παραπάνω. Αυτή η μέθοδος ακολουθείται σε όλη την διάρκεια του κώδικα, είτε πρόκειται για λήψη δεδομένων, είτε για αποστολή τους, όπως θα εξηγηθεί παρακάτω. [8]

```
protected void onPostExecute(String result)
{  //printing the values
    TextView textout1 = (TextView) findViewById(R.id.readdata1);
    textout1.setText(readdata1);

    TextView textout2 = (TextView) findViewById(R.id.readdata2);
    textout2.setText(readdata2);
```

```
    TextView textout3 = (TextView) findViewById(R.id.readdata3);
    textout3.setText(readdata3);

    TextView textout7 = (TextView) findViewById(R.id.readdata4);
    textout7.setText((readdata4));

    if (CoolingBitCheck == null){
        TextView textout4 = (TextView) findViewById(R.id.coolingState);
        textout4.setText(coolingState);
    }

    if (CoolingBitCheck != null) {
        if (CoolingBitCheck == true) {
            TextView textout5 = (TextView) findViewById(R.id.coolingState);
            CoolingOnOff = "Cooling is on";
            textout5.setText(CoolingOnOff);
        } else {
            TextView textout6 = (TextView) findViewById(R.id.coolingState);
            CoolingOnOff = "Cooling is off";
            textout6.setText(CoolingOnOff);
        }
    }
}
```

Εικόνα 3.8: Ο κώδικας που αφορά την εμφάνιση των δεδομένων στα αντίστοιχα πεδία.

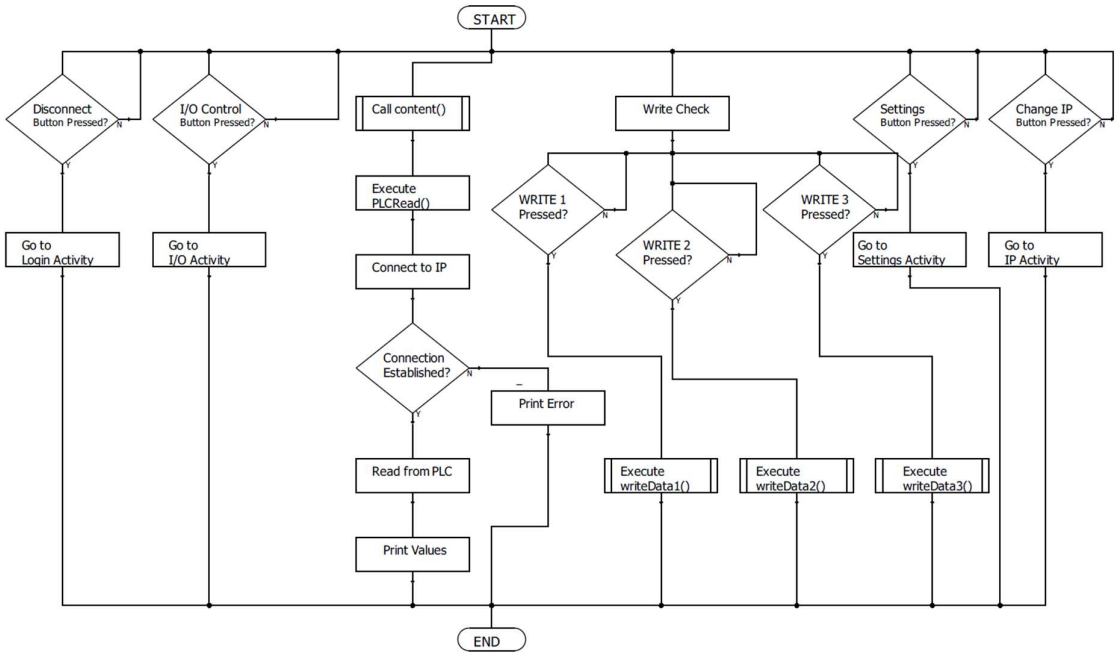
Στην παραπάνω εικόνα φαίνεται το κομμάτι κώδικα το οποίο εμφανίζει τα δεδομένα στα αντίστοιχα πεδία. Επίσης, η μεταβλητή ‘’CoolingBitCheck’’ ελέγχεται αρχικά για το αν περιέχει κάποια τιμή ή αν είναι κενή και στη συνέχεια για το αν η τιμή είναι αληθής ή ψευδής, με βάση την οποία εκτελείται ο αντίστοιχος κώδικας. Τα δεδομένα ανανεώνονται αυτόματα κάθε δέκα δευτερόλεπτα, όπως καθορίζει η ρουτίνα καθυστέρησης ‘’refresh’’.

Πιο κάτω στο ίδιο Activity, ο χρήστης μπορεί να γράψει τις τιμές που επιθυμεί για το όριο θερμοκρασίας, το χρόνο ανάδευσης και το χρόνο καθαρισμού και να τις στείλει στο PLC, στις αντίστοιχες θέσεις.

```
if (writecheck1 == 0) {
    EditText writedata1 = findViewById(R.id.etWrite1);
    String data1 = writedata1.getText().toString();  //Value field
    writedata1.getText().clear();  //Clear value field after writing

    byte[] dataBuffer1 = new byte[256];  //data buffer
    S7.SetWordAt(dataBuffer1, 0, Integer.parseInt(data1));
    client.WriteArea(S7.S7AreaDB, 4,0,2,dataBuffer1);  //where to write
}
```

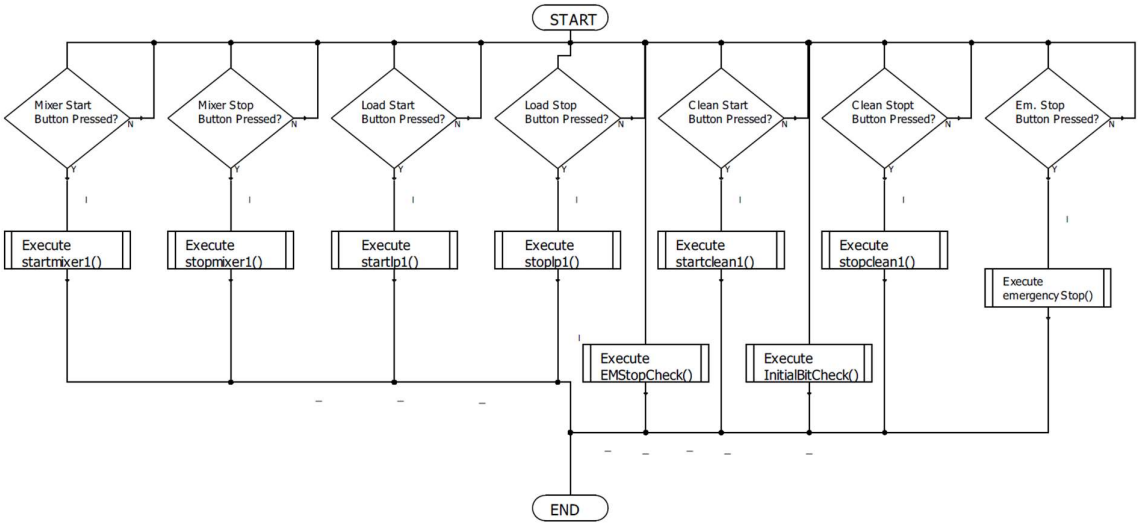
Εικόνα 3.9: Ο κώδικας για την αποστολή του setpoint θερμοκρασίας προς το PLC.

Στον παραπάνω κώδικα, για τη σύνδεση στο PLC ακολουθείται η ίδια μέθοδος που έχει αναφερθεί ήδη πιο πάνω. Η μεταβλητή ‘’writecheck1’’ είναι αυτή στην οποία αποθηκεύεται η τιμή από την οποία καθορίζεται η επιτυχημένη σύνδεση. Το ‘’writedata1’’ είναι η ονομασία του πρώτου πεδίου, στο οποίο γράφεται η θερμοκρασία και έπειτα αποθηκεύεται σε μια string μεταβλητή ‘’data1’’. Αυτή μετατρέπεται και αποθηκεύεται στον buffer ‘’dataBuffer1’’ στη θέση ‘’Pos: 0’’. Έπειτα, αποστέλλεται στην αντίστοιχη θέση στην οποία θέλουμε να μεταδοθεί στο PLC και καθορίζεται με βάση τον τρόπο που αναφέρθηκε παραπάνω. Τέλος, γίνεται η δήλωση του τι πρέπει να εγγραφεί και που. Η ίδια διαδικασία ακολουθείται και για τα υπόλοιπα δύο πεδία, με τη μόνη διαφορά να είναι η θέση στην οποία γράφεται κάθε πληροφορία. [8]

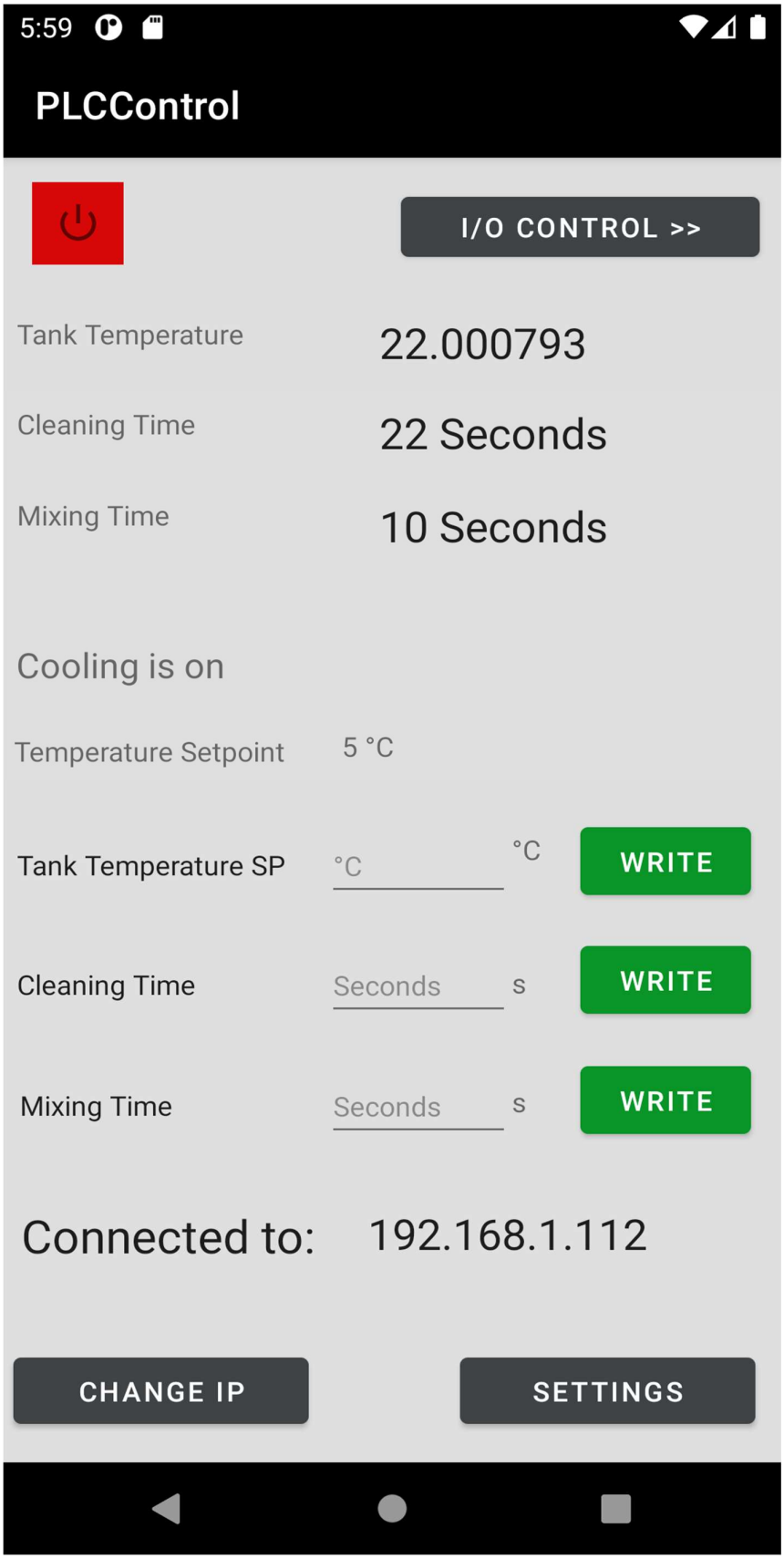


Σχήμα 3.3: Το διάγραμμα ροής του Read/Write Activity.

Στο ‘’ReadWriteActivity’’ το οποίο αναλύθηκε παραπάνω, υπάρχουν κάποιες επιλογές που δίνονται στο χρήστη, όπως αναφέρθηκε και οι οποίες φαίνονται στην Εικόνα 3.10. Με την επιλογή ‘’CHANGE IP’’ ο χρήστης μεταφέρεται στο ‘’IPActivity’’ όπου μπορεί εκ νέου να πληκτρολογήσει μια IP διεύθυνση. Με την επιλογή ‘’SETTINGS’’ μεταφέρεται στο ‘’SettingsActivity’’, μέσω του οποίο μπορεί να αλλάξει διαπιστευτήρια για τη σύνδεση στην εφαρμογή και το οποίο θα αναλυθεί παρακάτω. Με την επιλογή του πλήκτρου αποσύνδεσης, μεταφέρεται στο ‘’LoginActivity’’, όπου μπορεί να συνδεθεί και πάλι στην εφαρμογή. Τέλος, με την επιλογή ‘’I/O CONTROL’’ μεταφέρεται στο ‘’ioControlActivity’’, μέσω του οποίου μπορεί να χειριστεί τις εξόδους του PLC και το οποίο εξηγείται παρακάτω.



Σχήμα 3.4: Το διάγραμμα ροής του I/O Control Activity.



Εικόνα 3.10: Το ‘’ReadWriteActivity’’.

```
int check = client.ConnectTo("" + SetAddress,0,1);
if(check == 0){
    client.ReadArea(S7.S7AreaDB,1,0,10, buttonData);
    EMStopBitCheck = S7.GetBitAt(buttonData, 1, 3);
}
```

Εικόνα 3.11: Έλεγχος κατάστασης του στοπ έκτακτης ανάγκης (Emergency Stop).

Το ‘’ioControlActivity’’ αφορά τον έλεγχο των εξόδων του PLC, και την λήψη πληροφορίας για τη στάθμη της δεξαμενής. Με την εκκίνηση του συγκεκριμένου Activity γίνεται η αρχική λήψη πληροφοριών, οι οποίες αφορούν την κατάσταση των εξόδων, καθώς και του στοπ έκτακτης ανάγκης. Ο έλεγχος του τελευταίου, γίνεται κάθε τρία δευτερόλεπτα, έτσι ώστε αν πιεστεί το στοπ έκτακτης ανάγκης που υπάρχει στον πίνακα, η πληροφορία να φτάσει στην εφαρμογή και να ενημερωθεί ο χρήστης. Επίσης, γίνεται έλεγχος κάθε ένα δευτερόλεπτο και για την κατάσταση των υπόλοιπων εξόδων και της στάθμης, έτσι ώστε αν πιεστεί κάποιο μπουτόν ή η στάθμη βρεθεί σε χαμηλό επίπεδο, να ενημερωθούν τα δεδομένα της εφαρμογής, ειδοποιώντας το χρήστη.

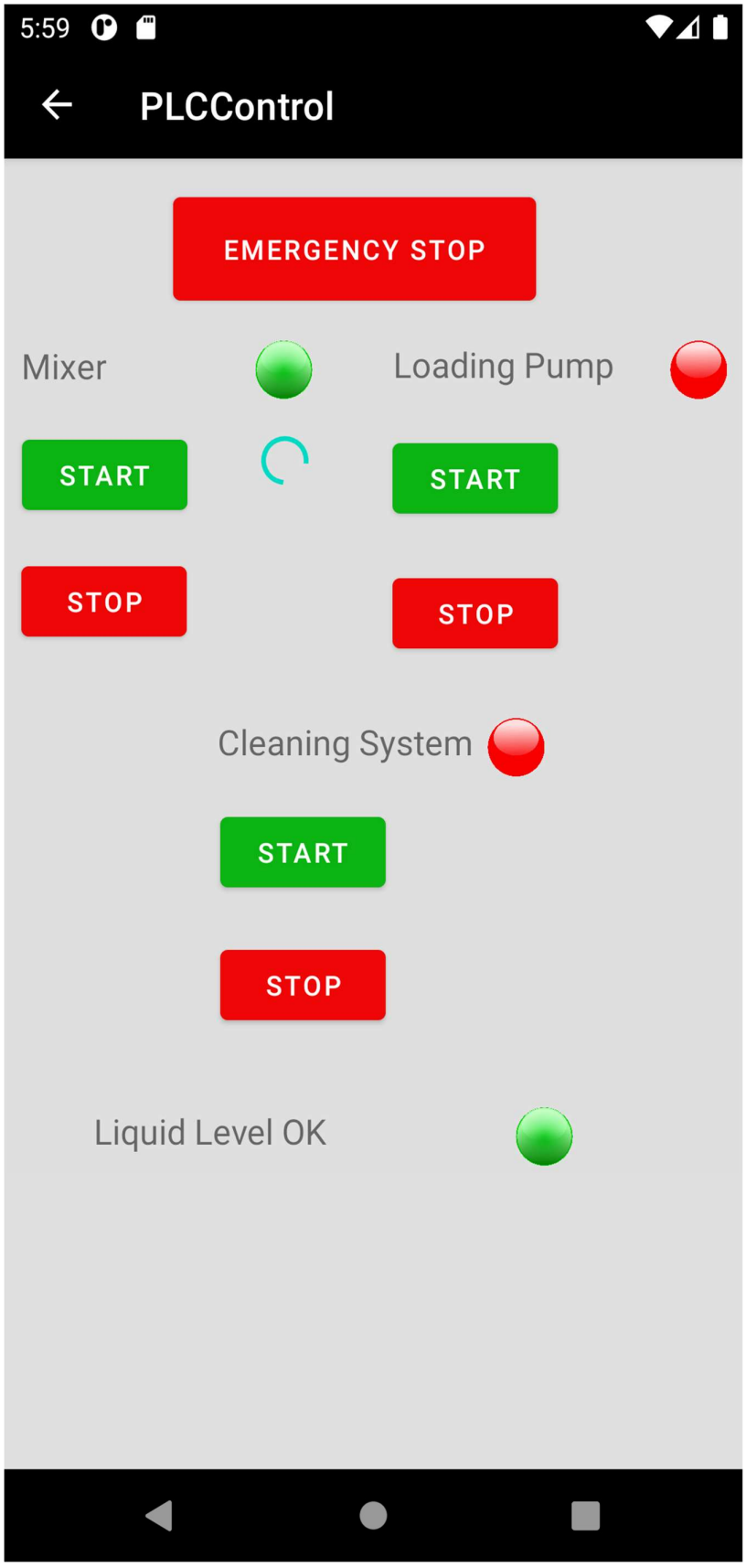
```
client.SetConnectionType(Moka7.S7.S7_BASIC);

int check = client.ConnectTo("" + SetAddress, 0, 1);

if (check == 0){

    client.ReadArea(Moka7.S7.S7AreaDB, 1, 0, 10, buttonData);
    MixerBitCheck = Moka7.S7.GetBitAt(buttonData, 0, 7);
    LoadBitCheck = Moka7.S7.GetBitAt(buttonData, 1,0);
    CleanBitCheck = Moka7.S7.GetBitAt(buttonData,1,1);
    EmptyBitCheck = S7.GetBitAt(buttonData,1,4);

}
```

Εικόνα 3.12: Ο κώδικας ελέγχου κατάστασης εξόδων PLC και στάθμης δεξαμενής.

Αφού γίνει ο έλεγχος, τότε καθορίζονται τα αντίστοιχα πεδία στην εφαρμογή. Συγκεκριμένα, όπως φαίνεται στην παρακάτω εικόνα, κάθε μεταβλητή ελέγχεται αρχικά για το αν περιέχει κάποια τιμή και ύστερα για το ποια είναι αυτή η τιμή. Ανάλογα το αποτέλεσμα, γίνονται και οι αντίστοιχες κινήσεις, οι οποίες περιέχονται είτε στην if, είτε στην else, σε κάθε περίπτωση. Όταν μια έξοδος είναι σε λειτουργία, τότε το ενδεικτικό δίπλα από τα κουμπιά ελέγχου, γίνεται από κόκκινο, πράσινο και εμφανίζεται μια μπάρα η οποία κινείται κυκλικά και υποδεικνύει τη λειτουργία της εξόδου. Όταν η λειτουργία σταματήσει, τότε η μπάρα σταματάει και το ενδεικτικό γίνεται ξανά κόκκινο. Το ίδιο ενδεικτικό υπάρχει και στα υπόλοιπα πλήκτρα ελέγχου. Τέλος, αν η στάθμη είναι σε κανονικά επίπεδα, δηλαδή δεν έχει ενεργοποιηθεί το φωτοκύτταρο, τότε στο κάτω μέρος της συγκεκριμένης οθόνης εμφανίζεται η ένδειξη ‘’Liquid Level OK’’ με το ενδεικτικό από δίπλα να είναι πράσινο, όπως φαίνεται στην Εικόνα 3.14α. Στην Εικόνα 3.14β απεικονίζεται η αντίθετη περίπτωση, όταν δηλαδή η στάθμη είναι χαμηλή. Τότε το ενδεικτικό γίνεται κόκκινο και η ένδειξη αλλάζει σε ‘’Empty Tank Warning’’. Η ακολουθία του κώδικα φαίνεται και στο κομμάτι που παρατίθεται στην Εικόνα 3.13.

```
protected void onPostExecute(String result) {

      if (MixerBitCheck != null) {
          if (MixerBitCheck == true) {
              stateImageMixer1.setImageResource(R.drawable.green_light);
              progressBar1.setVisibility(VISIBLE);
          } else {
              stateImageMixer1.setImageResource(R.drawable.red_light);
              progressBar1.setVisibility(GONE);
          }
      }

      if (LoadBitCheck != null) {
          if (LoadBitCheck == true) {
              stateImageLoad1.setImageResource(R.drawable.green_light);
              progressBar2.setVisibility(VISIBLE);
          } else {
              stateImageLoad1.setImageResource(R.drawable.red_light);
              progressBar2.setVisibility(GONE);
          }
      }

      if (CleanBitCheck != null) {
          if (CleanBitCheck == true) {
              stateImageClean1.setImageResource(R.drawable.green_light);
              progressBar3.setVisibility(VISIBLE);
          } else {
              stateImageClean1.setImageResource(R.drawable.red_light);
              progressBar3.setVisibility(GONE);
          }
      }

      if (EmptyBitCheck != null) {
          if (EmptyBitCheck == true) {
              TextView textout1 = (TextView) findViewById(R.id.textView14);
              EmptyState = "Empty Tank Warning";
              textout1.setText(EmptyState);
              stateImageEmpty.setImageResource(R.drawable.red_light);
          }
          else {
              TextView textout1 = (TextView) findViewById(R.id.textView14);
              EmptyState = "Liquid Level OK";
              textout1.setText(EmptyState);
              stateImageEmpty.setImageResource(R.drawable.green_light);
          }
      }
    }
}  //Initial Bit Check Function
```

Εικόνα 3.13: Οι ενέργειες του κώδικα μετά τον αρχικό έλεγχο.



Εικόνα 3.14α: Οι ενδείξεις σε κανονική λειτουργία.

Εικόνα 3.14β: Η ένδειξη χαμηλής στάθμης.

Για κάθε πλήκτρο ‘’START’’ και ‘’STOP’’ για κάθε μια από τις τρείς λειτουργίες, όπως έχει αναφερθεί, αποστέλλεται ξεχωριστή πληροφορία στο PLC, σε συγκεκριμένη θέση, όπως έχει οριστεί στον κώδικα του PLC, ο οποίος αναλύθηκε στο προηγούμενο κεφάλαιο. Πιο συγκεκριμένα, με την επιλογή ενός πλήκτρου, αποστέλλεται μέσω του κώδικα της android εφαρμογής η εκάστοτε πληροφορία στη συγκεκριμένη μεταβλητή που έχει οριστεί στο PLC και αλλάζει την τιμή από ‘’TRUE’’ σε ‘’FALSE’’ ή το αντίστροφο. Ωστόσο, θα αποτελούσε πρόβλημα αν μια εντολή παρέμενε αληθής. Για παράδειγμα, αν επιλεγεί η εκκίνηση του αναδευτήρα μέσω του αντίστοιχου ‘’START’’, τότε η αντίστοιχη θέση στο PLC θα γινόταν αληθής και ο αναδευτήρας θα εκκινούσε. Αν παρόλα αυτά, γινόταν προσπάθεια παύσης του με το πλήκτρο ‘’STOP’’, ο αναδευτήρας θα σταματούσε, ωστόσο η θέση ‘’STOP’’ στο PLC θα παρέμενε αληθής, κάτι που σημαίνει ότι δε θα μπορούσε να εκκινήσει εκ νέου ο αναδευτήρας. Το πρόβλημα λύθηκε με τη χρήση του κώδικα που φαίνεται στις παρακάτω εικόνες.

```
private class StartMixer1 extends AsyncTask <String, Void, String>{
    Boolean MixerBitCheck;
    @Override
    protected String doInBackground(String...params){
        try{
            client.SetConnectionType(Moka7.S7.S7_BASIC);
            int check = client.ConnectTo("" + SetAddress, 0, 1);
            if (check == 0){  //connection successful
                Moka7.S7.SetBitAt(buttonData, 0, 1, true);  //position to
write
                client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
                MixerBitCheck = Moka7.S7.GetBitAt(buttonData, 0, 7);
//position to check
            }
            else {

            }
            client.Disconnect();
        }
        catch (Exception e) {
            Thread.interrupted();
        }

        return "execute";
    }

    @Override
    protected void onPostExecute(String result) {
        if (MixerBitCheck != null) {  //check if empty
            if (MixerBitCheck == true) {  //check if true
                stateImageMixer1.setImageResource(R.drawable.green_light);
            } else {
                stateImageMixer1.setImageResource(R.drawable.red_light);
            }
        }
    }
}  //Mixer Start Class
```

Εικόνα 3.15α: Η λειτουργία του κώδικα που αφορά την εκκίνηση του αναδευτήρα.

Στην Εικόνα 3.15α παρουσιάζεται το κομμάτι του κώδικα που αφορά την επικοινωνία της εφαρμογής με το PLC για την εκκίνηση του αναδευτήρα. Αρχικά, γίνεται η σύνδεση με την απομακρυσμένη συσκευή, με τον τρόπο ο οποίος έχει αναλυθεί σε προηγούμενες παραγράφους, αποστέλλεται η τιμή ‘’TRUE’’ με την εντολή SetBitAt της βιβλιοθήκης Moka7 και στη συνέχεια

δηλώνεται το Data Block στο οποίο βρίσκεται η μεταβλητή, της οποίας η τιμή πρέπει να αλλαχθεί. Έπειτα, η εφαρμογή λαμβάνει με την εντολή GetBitAt, την κατάσταση μιας διαφορετικής μεταβλητής από το PLC, η οποία γίνεται αληθής αν έχει ενεργοποιηθεί η έξοδος που αφορά τον αναδευτήρα. Με βάση τον έλεγχο αυτής της μεταβλητής και της τιμής που επιστρέφει στην εφαρμογή, τότε είτε το ενδεικτικό γίνεται πράσινο, κάτι που σημαίνει ότι ο αναδευτήρας λειτουργεί, είτε παραμένει κόκκινο, που σημαίνει ότι κάτι πήγε στραβά και ο αναδευτήρας δεν ξεκίνησε. [8]

```
private class ResetStartMixer1 extends AsyncTask <String, Void, String>{
    @Override
    protected String doInBackground(String...params){
        try{
            client.SetConnectionType(Moka7.S7.S7_BASIC);
            int check = client.ConnectTo("" + SetAddress, 0, 1);
            if (check == 0){
                //byte[] data = new byte[256];
                Moka7.S7.SetBitAt(buttonData, 0, 1, false);
                client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
            }
            else {

            }
            client.Disconnect();
        }
        catch (Exception e) {
            Thread.interrupted();
        }
        return "execute";
    }
}  //Start Value Reset class so the button works as a momentary push button
```

Εικόνα 3.15β: Η επαναφορά της κατάστασης του ‘’START’’ από ‘’TRUE’’ σε ‘’FALSE’’.

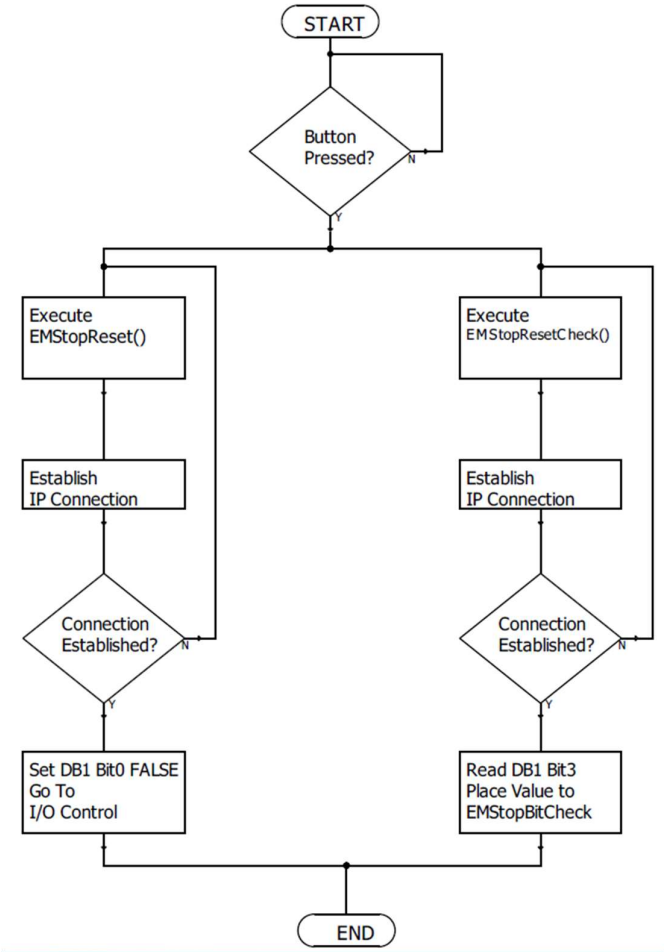
Αφού εκτελεστεί ο κώδικας που φαίνεται στην Εικόνα 3.15α, τότε η κατάσταση του bit στο οποίο βρίσκεται η μεταβλητή ‘’START’’ στη μεριά του PLC, θα παραμείνει αληθής. Ωστόσο, αυτό δημιουργεί προβλήματα, καθώς μπορεί επαναφέροντας κάποιο stop, να ξεκινήσει αναπάντεχα κάποια λειτουργία του συστήματος, αφού το απομακρυσμένο ‘’START’’ δουλεύει παράλληλα με το πλήκτρο ‘’START’’ το οποίο υπάρχει στον πίνακα αυτοματισμού και στην HMI. Αυτό το πρόβλημα λύθηκε με τη χρήση μια διαφορετικής κλάσης, μέσα στην οποία περιέχεται το κομμάτι κώδικα, το οποίο με την ίδια μέθοδο, όπως παραπάνω, επαναφέρει την τιμή του ‘’START’’ σε ‘’FALSE’’ και λειτουργεί με καθυστέρηση ενός δευτερολέπτου, αφού εκτελεστεί ο κώδικας της Εικόνας 3.15α. Έτσι, το ‘’START’’ του αναδευτήρα στη συγκεκριμένη περίπτωση, λειτουργεί ως στιγμιαίο πλήκτρο, το οποίο επαναφέρεται αυτόματα στην αρχική του θέση, όπως συμβαίνει και με ένα στιγμιαίο πλήκτρο το οποίο βρίσκεται στον πίνακα αυτοματισμού ή στην HMI. Με την ίδια ακριβώς μέθοδο έχουν προγραμματιστεί να λειτουργούν όλα τα πλήκτρα ‘’START’’ και ‘’STOP’’, τα οποία βρίσκονται στο συγκεκριμένο Activity, εκτός από το στοπ έκτακτης ανάγκης (‘’EMERGENCY STOP’’). Το μόνο που αλλάζει σε κάθε περίπτωση, είναι η θέση της αντίστοιχης μεταβλητής στο PLC, στην οποία πρέπει να μεταφερθεί η πληροφορία.

```
public void emergencyStop (View v){
    new emStop().execute("");
    Intent intentES = new Intent(ioControlActivity.this,
ESTriggeredActivity.class);
    startActivity(intentES);

}  //If Em. Stop is pushed, go to ESTriggered Activity

public class emStop extends AsyncTask <String, Void, String>{
    @Override
    protected String doInBackground(String...params){
        try{
            client.SetConnectionType(S7.S7_BASIC);
            int check = client.ConnectTo("" + SetAddress,0,1);
            if(check == 0){
                S7.SetBitAt(buttonData,0,0,true);
                client.WriteArea(S7.S7AreaDB,1,0,12, buttonData);
            }
            else{

            }
            client.Disconnect();
        } catch (Exception e){
            Thread.interrupted();
        }
        return "execute";
    }
}  //Class for Em. Stop
```

Εικόνα 3.16: Ο κώδικας ο οποίος αφορά τη λειτουργία του Emergency Stop.

Το στοπ έκτακτης ανάγκης αποτελεί το πλήκτρο το οποίο πιέζεται μόνο σε ιδιαίτερες περιπτώσεις, κυρίως όταν υπάρχει κίνδυνος από τη συμπεριφορά του συστήματος, είτε για τον άνθρωπο, είτε για το ίδιο το σύστημα. Η χρήση του στην εφαρμογή είναι παρόμοια με αυτήν που έχει το φυσικό πλήκτρο, επάνω στον πίνακα αυτοματισμού. Με την ίδια μέθοδο όπως και παραπάνω, με την επιλογή του πλήκτρου ‘’EMERGENCY STOP’’, τότε η εφαρμογή αποστέλλει την αντίστοιχη πληροφορία στην αντίστοιχη θέση του PLC, για να σταματήσουν όλες οι λειτουργίες. Μετά την εκτέλεση του κώδικα, η εφαρμογή μεταφέρει τον χρήστη σε ένα νέο ‘’ESTriggeredActivity’’, στο οποίο παρουσιάζεται η πληροφορία ότι έχει πιεστεί το στοπ έκτακτης ανάγκης, μαζί με ένα πλήκτρο ‘’RESET’’. Το συγκεκριμένο Activity φαίνεται στην Εικόνα 3.17 και κομμάτια του κώδικά του στις Εικόνες 3.18 και 3.19.

Εικόνα 3.17: Η οθόνη στο ‘’ESTriggeredActivity’’.



Σχήμα 3.5: Το διάγραμμα ροής του Emergency Stop Triggered Activity.

```
private class EMStopResetCheck extends AsyncTask<String, Void, String>{
    Boolean EMStopBitCheck;
    @Override
    protected String doInBackground(String...params){
        try{
            client.SetConnectionType(Moka7.S7.S7_BASIC);
            int check = client.ConnectTo("" + SetAddress,0,1);
            if(check == 0){
                client.ReadArea(Moka7.S7.S7AreaDB,1,0,10, buttonData);
                EMStopBitCheck = Moka7.S7.GetBitAt(buttonData, 1, 3);
//Check if Em. Stop position is Reset
            }
            else{

            }
            client.Disconnect();
        }
        catch (Exception e){
            Thread.interrupted();
        }
        return "execute";
    }

}  //Check if Em. Stop has been reset
```

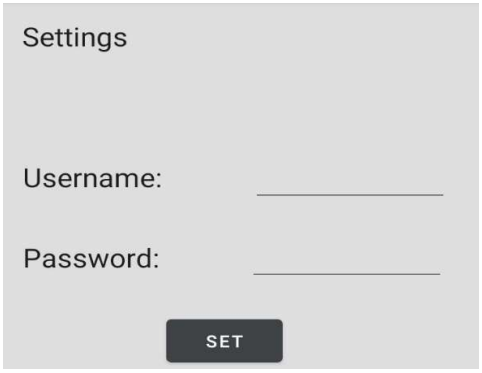
Εικόνα 3.18: Ο κώδικας που ελέγχει την επαναφορά του φυσικού ‘’Emergency Stop’’.

Όταν πιεστεί ένα από το δύο στοπ έκτακτης ανάγκης, δηλαδή αυτό που υπάρχει στον πίνακα αυτοματισμού και αυτό που υπάρχει στην εφαρμογή, τότε η εφαρμογή μεταφέρεται στο ‘’ESTriggeredActivity’’. Σε αυτό το Activity γίνεται έλεγχος αφενός για το αν έχει γίνει επαναφορά του στοπ, το οποίο βρίσκεται στον πίνακα αυτοματισμού και αφετέρου για το αν έχει πιεστεί το πλήκτρο ‘’RESET’’. Στην πρώτη περίπτωση, μόλις πιεστεί το ‘’RESET’’, ελέγχεται η κατάσταση της μεταβλητής που αφορά το εξωτερικό στοπ, με την εντολή GetBitAt και δηλώνοντας τη θέση και το Data Block που βρίσκεται, στη μεριά του PLC. Αν δεν έχει γίνει επαναφορά του φυσικού στοπ, τότε η εφαρμογή παραμένει στο συγκεκριμένο Activity. Αν έχει γίνει επαναφορά του φυσικού στοπ, τότε η εφαρμογή μπορεί να επανέλθει στο προηγούμενο Activity, αφού πρώτα έχει στείλει την εντολή επαναφοράς της κατάστασης της θέσης που αφορά το στοπ της εφαρμογής, από ‘’TRUE’’ σε ‘’FALSE’’. Έτσι το σύστημα επανέρχεται στην αρχική του κατάσταση και μπορεί να λειτουργήσει και πάλι.

```
public class EMStopReset extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String...params){
        try{
            client.SetConnectionType(Moka7.S7.S7_BASIC);
            int check = client.ConnectTo("" + SetAddress,0,1);
            if(check == 0){
                Moka7.S7.SetBitAt(buttonData,0,0,false);  //If Reset is
pressed, change the Em. Stop value to PLC
                client.WriteArea(Moka7.S7.S7AreaDB,1,0,12, buttonData);
                Intent intentReset = new Intent(ESTriggeredActivity.this,
ioControlActivity.class);  //Go to I/O Control
                startActivity(intentReset);
            }
            else{

            }
            client.Disconnect();
        } catch (Exception e){
            Thread.interrupted();
        }
        return "execute";
    }
}  //Check if Reset is pressed
```

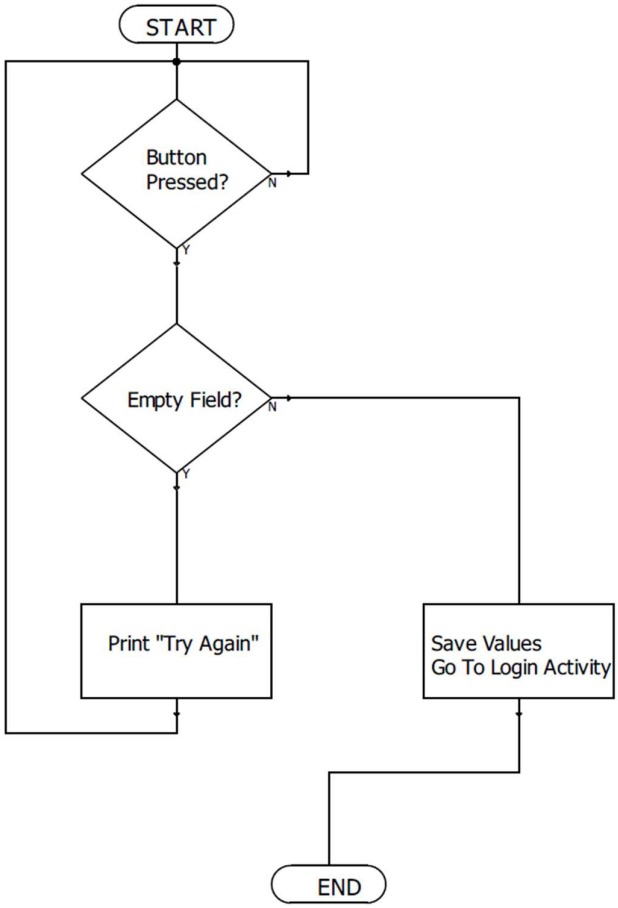
Εικόνα 3.19: Ο κώδικας που επαναφέρει τη θέση του ‘’Emergency Stop’’ στο PLC σε ‘’FALSE’’.

Όπως αναλύθηκε στο ‘’ReadWriteActivity’’, μια από τις επιλογές του χρήστη είναι η επιλογή ‘’SETTINGS’’ (Εικόνα 3.10), η οποία τον μεταφέρει σε ένα νέο Activity, με όνομα ‘’SettingsActivity’’ και το οποίο φαίνεται στην Εικόνα 3.20.



Εικόνα 3.20: Τα πεδία στο ‘’SettingsActivity’’.

Εδώ ο χρήστης μπορεί να αλλάξει τα διαπιστευτήρια, τα οποία χρησιμοποιεί για τη σύνδεση του στην εφαρμογή. Μπορεί να επιλέξει το όνομα χρήστη και τον κωδικό που επιθυμεί και πιέζοντας το ‘’SET’’ οι τιμές αυτές αποθηκεύονται σε μεταβλητές οι οποίες μπορούν να χρησιμοποιηθούν πλέον στο ‘’LoginActivity’’, εκεί όπου μεταφέρεται μετά την αποθήκευση των μεταβλητών, για να εισέλθει με τα νέα διαπιστευτήρια. Ο κώδικας φαίνεται στην παρακάτω εικόνα.



Σχήμα 3.6: Το διάγραμμα ροής του Settings Activity.

```
setbtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            changedName = username.getText().toString();
            changedPassword = password.getText().toString();
            if(TextUtils.isEmpty(changedName) ||
TextUtils.isEmpty(changedPassword)) {
                Toast.makeText(SettingsActivity.this, "Try Again",
Toast.LENGTH_LONG).show();  //Check for empty field
            }
            else {
                Intent intentCredSet = new Intent(SettingsActivity.this,
LoginActivity.class);  //Go to Login Activity
                startActivity(intentCredSet);
            }
        }
    });  //After credentials are set, go to Login Activity
}
```

```
public static String getChangedName(){

    return changedName;
}  //Name stored

public static String getChangedPassword(){

    return changedPassword;
}  //Password stored
```

Εικόνα 3.21: Ο κώδικας του ‘’SettingsActivity’’.

Στον παραπάνω κώδικα, χρησιμοποιήθηκε η συνάρτηση SetOnClickListener() και η μεταβλητή ‘’setbtn’’ είναι αυτή που σχετίζεται με το πλήκτρο ‘’SET’’. Το όνομα χρήστη και ο κωδικός λαμβάνονται από τα αντίστοιχα πεδία, και μετατρέπονται σε string μεταβλητές. Αν ένα από τα πεδία είναι άδεια, ο χρήστης ειδοποιείται. Αν υπάρχουν τιμές στις μεταβλητές, τότε αυτές αποθηκεύονται και η εφαρμογή γυρίζει στο ‘’LoginActivity’’. Οι δύο συναρτήσεις ‘’getChangedName() και getChangedPassword() εξυπηρετούν στη μεταφορά των δύο μεταβλητών στο ‘’LoginActivity’’. [4][17]

3.3 Επίλογος

Στο Κεφάλαιο 3 παρουσιάστηκαν οι λειτουργίες της Android εφαρμογής, μαζί με τα κομμάτια κώδικα που τις συνοδεύουν. Με τη χρήση των σωστών διαπιστευτηρίων ο χρήστης μπορεί να εισέλθει στην εφαρμογή, να πληκτρολογήσει την IP διεύθυνση, έτσι ώστε να συνδεθεί στο PLC και στη συνέχεια να διαβάσει τα δεδομένα που μεταδίδονται, εφόσον η σύνδεση είναι επιτυχής. Επίσης, μπορεί να αλλάξει τα πεδία που ορίζονται με τις επιθυμητές τιμές σε κάθε περίπτωση, ενώ του δίνεται η δυνατότητα είτε να αποσυνδεθεί, είτε να μεταφερθεί σε άλλες λειτουργίες. Μπορεί επίσης να ελέγξει την κατάσταση και τη λειτουργία των εξόδων του συστήματος, ενώ ενημερώνεται για πιθανές δυσλειτουργίες του συστήματος. Τέλος, μπορεί να αλλάξει τα διαπιστευτήρια του και να συνδεθεί εκ νέου στην εφαρμογή, ακολουθώντας την ίδια διαδικασία.

Οι λειτουργίες της εφαρμογής κινούνται στο ίδιο πλαίσιο με τις αντίστοιχες του PLC, για την καλύτερη επικοινωνία και ανταλλαγή πληροφοριών. Επίσης, όπως αναφέρθηκε λήφθηκαν τα απαραίτητα μέτρα για την αποφυγή ανεπιθύμητων συμπεριφορών του συστήματος. Τέλος, οι χρόνοι που καθορίζονται είναι ενδεικτικοί, για την καλύτερη προσομοίωση και κατανόηση των λειτουργιών της εφαρμογής, όπως ισχύει και για τον αυτοματισμό του PLC και φυσικά μπορούν να μεταβληθούν μέσα από τον κώδικα.

Κεφάλαιο 4ο: Συμπεράσματα

Κλείνοντας την παρούσα εργασία παρουσιάζονται τα συμπεράσματα τα οποία προκύπτουν από την ανάλυση των μερών που την αποτελούν, την επικοινωνία τους και των διάφορων λειτουργιών. Επίσης, αναφέρονται χρήσιμες πληροφορίες και βελτιώσεις της.

Αρχικά, στο δεύτερο κεφάλαιο παρουσιάζεται η λειτουργία του αυτοματισμού. Αυτός αποτελείται από το PLC, το οποίο αποτελεί τον κεντρικό ελεγκτή του συστήματος και με βάση κάποιες συγκεκριμένες προϋποθέσεις ρυθμίζει τις λειτουργίες για τις οποίες έχει προγραμματιστεί και επικοινωνεί με την εφαρμογή και την οθόνη HMI. Επίσης, κομμάτι του αυτοματισμού αποτελεί και η οθόνη HMI, η οποία στην προκειμένη περίπτωση υπάρχει μόνο σε προσομοίωση, όπως αναφέρθηκε στην αντίστοιχη ενότητα. Μέσω αυτής ο χρήστης μπορεί να χειριστεί το σύστημα και να λάβει χρήσιμες πληροφορίες για αυτό. Ο προγραμματισμός των παραπάνω και η προσομοίωση της οθόνης γίνεται μέσω του περιβάλλοντος TIA Portal της εταιρείας Siemens. Τέλος, το τρίτο κομμάτι του αυτοματισμού αποτελείται από τον πίνακα αυτοματισμού. Εκεί περιέχεται, μαζί με τη συσκευή PLC, η τροφοδοσία, οι ηλεκτρονόμοι και τα διάφορα πλήκτρα, τα οποία επικοινωνούν τον ελεγκτή. Το σχέδιο του πίνακα σχεδιάστηκε με το πρόγραμμα QElectroTech, το οποίο είναι ένα δωρεάν λογισμικό για τη σχεδίαση ηλεκτρολογικών και ηλεκτρονικών σχεδίων. Εξωτερικά του πίνακα υπάρχει το φωτοκύτταρο στάθμης, όπως αναφέρθηκε και ο υποπίνακας που περιέχει το ESP32 και την ενισχυτική βαθμίδα. Ο προγραμματισμός του ESP32 έγινε μέσω του Arduino IDE και ο σχεδιασμός και υπολογισμός της ενισχυτικής βαθμίδας στο περιβάλλον PSPICE.

Πρόκειται λοιπόν για μια κατασκευή ερασιτεχνικού επιπέδου, όπως είναι φυσικό. Ωστόσο, μέσω των προτύπων και των προϋποθέσεων που τηρήθηκαν και εφαρμόστηκαν, η λειτουργία και η κατασκευή συνολικά τηρούν όλους τους κανόνες ασφαλείας, οι οποίοι ισχύουν στη βιομηχανία. Στην ουσία, όλη η κατασκευή και η υλοποίηση του συστήματος κινήθηκε με τη σκέψη ότι όντως πρόκειται να χρησιμοποιηθεί επαγγελματικά, για αυτό και προκύπτει αυτό το αποτέλεσμα. Οι βελτιώσεις οι οποίες μπορούν να εφαρμοστούν, αφορούν και τα υλικά και το λογισμικό. Ωστόσο, εφόσον ο πίνακας αυτοματισμού αποτελεί τη βασική προσομοίωση του συστήματος, κάθε βελτίωση εξαρτάται από την εφαρμογή, το περιβάλλον και το μέγεθος του συστήματος. Επίσης, θα μπορούσαν να γίνονται καλύτερες μετρήσεις στάθμης και θερμοκρασίας, με τη χρήση αναλογικών αισθητηρίων βιομηχανικού τύπου, ωστόσο λόγω κόστους δεν έγινε χρήση τους.

Στο τρίτο κεφάλαιο γίνεται η ανάλυση της Android εφαρμογής, του κώδικα για κάθε λειτουργία της και της επικοινωνίας της με το PLC. Χωρίζεται σε έξι κομμάτια, τα οποία αποτελούν διαφορετικό περιβάλλον για το χρήστη και παρέχουν διαφορετικές πληροφορίες. Η υλοποίηση της έγινε στο περιβάλλον του Android Studio, σε γλώσσα Java. Και στην περίπτωση της εφαρμογής τηρήθηκαν υψηλά πρότυπα, βασισμένα στην ίδια φιλοσοφία στην οποία κινήθηκε και η υλοποίηση του αυτοματισμού. Οι βελτιώσεις που επιδέχεται η εφαρμογή αφορούν περισσότερο στη μορφοποίηση της, στην εξατομίκευση της για κάθε χρήστη ή για κάθε εταιρεία ξεχωριστά και στον τρόπο παρουσίασης των δεδομένων. Ωστόσο, το τελευταίο εξαρτάται και από το σε ποια εφαρμογή θα χρησιμοποιηθεί μια τέτοια εφαρμογή. Οπότε, γίνεται κατανοητό πως η κάθε βελτίωση εξαρτάται και από τις απαιτήσεις που έχει κάθε σύστημα από μια τέτοια εφαρμογή και στον τρόπο που χρησιμεύει.

Μια ακόμη βελτίωση η οποία θα μπορούσε να γίνει, είναι η απομακρυσμένη επικοινωνία από διαφορετικό δίκτυο. Πιο συγκεκριμένα, εφόσον το PLC είναι συνδεδεμένο σε ένα τοπικό δίκτυο, μπορεί μέσω του Port Forwarding να επικοινωνήσει και εκτός τοπικού δικτύου. Η μέθοδος του Port

Forwarding αναλύθηκε στο πρώτο κεφάλαιο, μαζί με τη χρήση ενός DDNS, έτσι ώστε η διεύθυνση να παραμένει σταθερή, ακόμα και όταν η εξωτερική διεύθυνση του δρομολογητή αλλάξει, αν δεν είναι στατική. Όπως αντίστοιχα στο τοπικό δίκτυο η διεύθυνση είναι της μορφής 192.168.1.112, έτσι και στην περίπτωση απομακρυσμένου ελέγχου, ο χρήστης μπορεί να πληκτρολογήσει στην εφαρμογή, στην οθόνη που εισάγεται η IP διεύθυνση, το όνομα της ιστοσελίδας που έχει υλοποιηθεί μέσω του No-IP, μαζί με τον αριθμό της πόρτας που έχει ανοιχτεί μέσω της μεθόδου του Port Forwarding. Δηλαδή, αυτό που θα πληκτρολογήσει ο χρήστης στην android εφαρμογή, θα πρέπει να είναι της μορφής ‘’www.plccontrol.ddns.net:445’’, όπου φαίνεται το όνομα της σελίδας, και η πόρτα 445 η οποία έχει ανοίξει. Ωστόσο, λόγω δυσκολίας επικοινωνίας με τον πάροχο τηλεφωνίας του οικιακού δικτύου, οι δοκιμές έγιναν μόνο σε τοπικό δίκτυο. Η λειτουργία της εφαρμογής δε διαφέρει καθόλου, είτε πρόκειται για τοπικό, είτε για απομακρυσμένο έλεγχο, παρά μόνο στον τρόπο σύνδεσης και πιθανόν στο χρόνο απόκρισης, λόγω διαφορετικού δικτύου.

Κλείνοντας, τα συμπεράσματα που προκύπτουν αφορούν τόσο το όφελος της ενασχόλησης με ένα τέτοιου είδους σύστημα, όσο και τις βελτιώσεις που μπορούν να εφαρμοστούν. Η έρευνα και η εφαρμογή των γνώσεων στη συγκεκριμένη εργασία βοήθησε στην καλύτερη κατανόηση του αυτοματισμού από τη μια, με τη μελέτη των λειτουργιών του PLC, το συνδυασμό του με την οθόνη και τον πίνακα αυτοματισμού και τις τελικές αποφάσεις για τη λειτουργία του κάθε μέρους. Επίσης, έγιναν περισσότερο κατανοητές οι δυνατότητες και οι λειτουργίες του android λογισμικού. Από την άλλη, οι βελτιώσεις που μπορούν να εφαρμοστούν εξαρτώνται από τον τρόπο λειτουργίας του συστήματος, όπως αναφέρθηκε. Σε κάθε περίπτωση, μπορούν να γίνουν βελτιώσεις και παρεμβάσεις στα μέρη του συστήματος, τα οποία θα βοηθήσουν στην καλύτερη λειτουργία. Συνεπώς, η παρούσα κατάσταση του συστήματος που παρουσιάστηκε αποτελεί ένα δείγμα για τις δυνατότητες του και μια προσομοίωση τους, παρόλο τους περιορισμούς λόγω κόστους και εργαλείων υλοποίησης, παρά κάτι που απευθύνεται σε κάτι συγκεκριμένο και εξατομικευμένο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[1] Γ. Κρανάς και Ε. Δασκαλοπούλου, *Βιομηχανικοί Αυτοματισμοί & Προγραμματιζόμενοι λογικοί ελεγκτές (PLC),* Αθήνα, Εκδόσεις ΙΩΝ, 2008

[2] Σπάσος Ν. Μιχάλης, *Αναλογική Επεξεργασία Σημάτων Αισθητηρίων*, Εκδόσεις ΑΪΒΑΖΗ, 2018

[3] H. Schildt, *Java A Beginner's Guide,* New York, McGraw – Hill Education, 2019.

[4] B. Burd, J.P. Mueller, *Android Application Development All-In-One,* New Jersey, John Wiley & Sons Inc., 2020.

Manuals

[5] Siemens, Getting Started with S7-1200.

[6] Siemens, S7 – 1200 Programmable Controller.

[7] Siemens, HMI Devices Comfort Panels.

[8] Davide Nardella, ''Moka7 Reference Manual'', December 2016.

Papers

[9] Acromag, ''How to Connect to an Ethernet Device for Communication'', Michigan, Acromag, 2020.

[10] Raihan Bin Mofidul, Shahadath Hossain Sabbir, Amit Kumer Podder and Mohammad Shaifur Rahman, ''Design and Implementation of Remote Controlling and Monitoring System for Automatic PLC Based Packaging Industry'', *1st International Conference on Advances in Science, Engineering and Robotics Technology*, 2019.

[11] Cemal Yılmaz, Ercan Nurcan Yılmaz, Mehmet Fatih Isık, Mehmet Ali Sinan Usalan, Yusuf Sonmez, Veysel Ozdemir, ''Design and Implementation of Real-Time Monitoring and Control System Supported with IOS/Android Application for Industrial Furnaces'', *John Wiley & Sons, Inc*, 2018.

[12] Bin Qiu, Hoay Beng Gooi, Yilu Liu, Eng Kiat Chan, ''Internet-based SCADA Display System'', *IEEE Computer Applications in Power*, January 2002.

[13] Lawrence O. Aghenta, M. Tariq Iqbal, ''Development of an IoT Based Open Source SCADA System for PV System Monitoring'', *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, 2019.

Data Sheets

[14] Espressif Systems, ''ESP32 Datasheet Series'', ESP32 Datasheet Version 3.9, 2022.

[15] Texas Instruments, ''Industry-Standard Dual Operational Amplifiers datasheet (Rev. AA)'', LM358P Datasheet, Mar. 2022 (Revision).

[16] Adafruit, ‘’DHT11, DHT22 and AM2302 Sensors’’. DHT11 Datasheet, Dec. 2021.

[17] Vishay, ‘’OLED-128O064D-BPP3N00000 Display Datasheet’’, OLED Display Datasheet, Dec. 2016.

Internet Sites

[18] Google, ‘’Android Studio User Guide’’, *Google*, 2021 [Online]. Available: www.developer.android.com

[19] Google, ‘’Async Task Object in Android’’, *Google*, 2021 [Online]. Available: www.developer.android.com

Journal Articles

[20] Sagarika Ghosh, Srinivas Sampalli, ‘’A Survey of Security in SCADA Networks: Current Issues and Future Challenges’’, *Access IEEE*, vol. 8, pp. 135812-135831, July 2019.

[21] Dimitrios Pliatsios, Panagiotis Sarigiannidis, Thomas Lagkas and Antonios G. Sarigiannidis, ‘’A Survey on SCADA Systems: Secure Protocols, Incidents, Threats and Tactics’’, *IEEE Communications Surveys & Tutorials*, vol. 22, pp. 1942-1976, April 2020.

[22] Mehmet Fatih Isik, Mustafa Resit Haboglu, Cemal Yilmaz, Ercan Nurcan Yilmaz, ‘’Design and Implementation of Real Time Monitoring and Control System for Distributed Robotic Systems Supported with IOS/Android Application’’, *Tehnicki Vjesnik*, pp 423-428, 2018.

[23] Ephrem Ryan Alphonsus, Mohammad Omar Abdullah, ‘’A review on the applications of programmable logic controllers (PLCs)’’, *Renewable and Sustainable Energy Reviews*, pp. 1185–1205, February 2016.

[24] Eki Ahmad Zaki Hamidi, Tiara Gustiana, Mufid Ridlo Effendi, Pajar Abdul Malik Hambali ‘’Design and Implementation Supervisory Control and Data Acquisition (SCADA) of Flocculation Process of Water Treatment Plant (WTP) Using Raspberry Pi’’, *2019 IEEE 5th International Conference on Wireless and Telematics (ICWT)*, pp. 1 – 5, 2019.

[25] Rahul Pawar, Dr. N. R. Bhasme, ‘’Application of PLC’s for Automation of Processes in Industries’’, *Int. Journal of Engineering Research and Applications*, vol. 6, pp. 53 - 59, June 2016.

[26] Nikita Shingre, Reema Nagwekar, Rupa Roy, Trupti Shendkar, Rupinder Kaur, ‘’Review On Mobile Phone Based SCADA For Industrial Automation’’, *International Journal of Technical Research and Applications*, vol. 39, pp 1-4, March 2016.

[27] G. Zhu and J. Lee, ‘’Development of Imitated-Handwriting Systems Using PLC-Controlled CoreXY Mechanisms’’, *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1117-1121, 2019.

[28] N. -V. Truong and D. -L. Vu, ‘’Remote monitoring and control of industrial process via wireless network and Android platform’’, *2012 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pp. 340-343, 2012.

[29] Carlos E. Pereira, Peter Neumann, ‘’Industrial Communication Protocols’’, *Springer Handbook of Automation*, pp. 981–999, 2009.

[30] A. Drumea, ‘’Control of industrial systems using Android-based devices’’, *Proceedings of the 36th International Spring Seminar on Electronics Technology*, pp. 405-408, 2013.

[31] T. J. Alexander, ‘’An implementation of mobile control room environment in android platform for industrial applications’’, *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pp. 1-4, 2015.

[32] S. G. Hegde, S. R. Desai, D. R. Gajanan, S. B. Kowligi and Sachin RC, ‘’Implementation of SCADA in industries using wireless technologies’’, *2015 International Conference on Industrial Instrumentation and Control (ICIC)*, pp. 143-148, 2015.

[33] Josif Tomic, Miodrag Kusljevic, Milan Vidakovic, Vladimir Rajs, ‘’Smart SCADA system for urban air pollution monitoring’’, *Measurement*, vol. 58, pp. 138-146, September 2014.

[34] A. DeBenedictis, B. Haley, C.K. Woo, E. Cutter, ‘’Operational energy-efficiency improvement of municipal water pumping in California’’, *Energy*, vol. 53, pp. 237-243, March 2013.

[35] R. Ahshan, M.T. Iqbal, George K.I. Mann, ‘’Controller for a small induction-generator based wind-turbine’’, *Applied Energy*, vol. 85, pp. 218–227, 2008.

ΠΑΡΑΡΤΗΜΑ Α : LADDER DIAGRAM

Παρακάτω φαίνεται το διάγραμμα Ladder, το οποίο αποτελεί τον κώδικα του PLC. Η υλοποίηση του έγινε με τη δωρεάν δοκιμαστική έκδοση του TIA Portal της εταιρείας Siemens. Κάθε επικεφαλίδα του κάθε Network περιγράφει τη λειτουργία του, ενώ οι μεταβλητές και τα αντίστοιχα blocks έχουν ονομαστεί κατάλληλα, για την καλύτερη κατανόηση της λειτουργίας του.

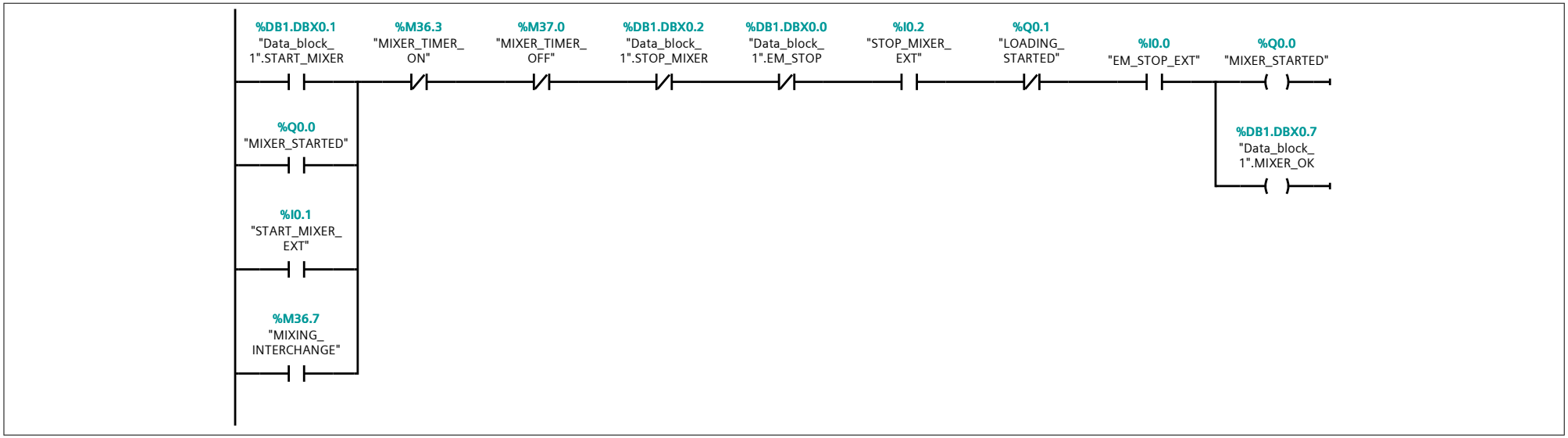
AndroidComm / PLC_1 [CPU 1212C DC/DC/DC] / Program blocks

Main [OB1]

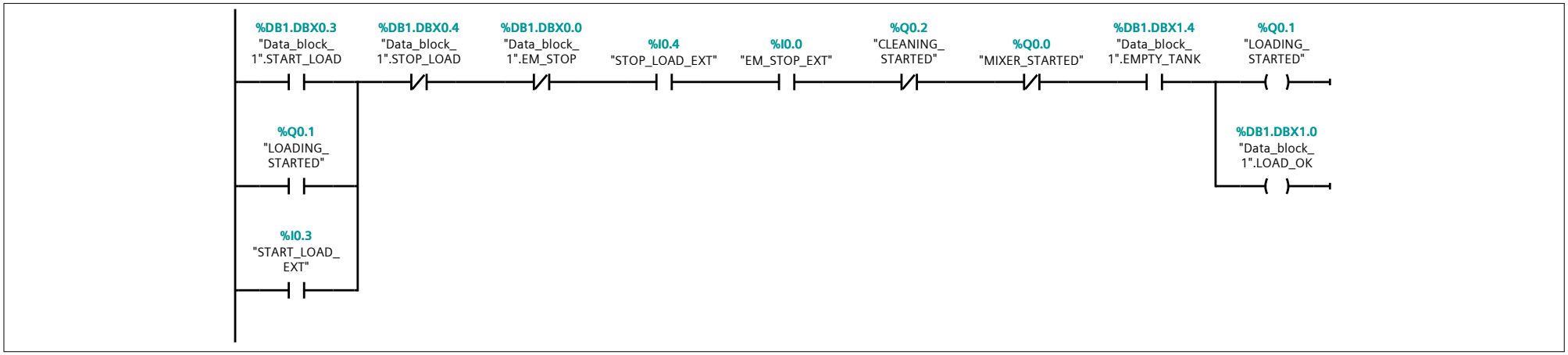
Main Properties							
General							
Name	Main	**Number**	1	**Type**	OB	**Language**	LAD
Numbering	Automatic						
Information							
Title	"Main Program Sweep (Cycle)"	**Author**		**Comment**		**Family**	
Version	0.1	**User-defined ID**					

Main			
Name	**Data type**	**Default value**	**Comment**
▼ Input			
Initial_Call	Bool		Initial call of this OB
Remanence	Bool		=True, if remanent data are available
Temp			
Constant			

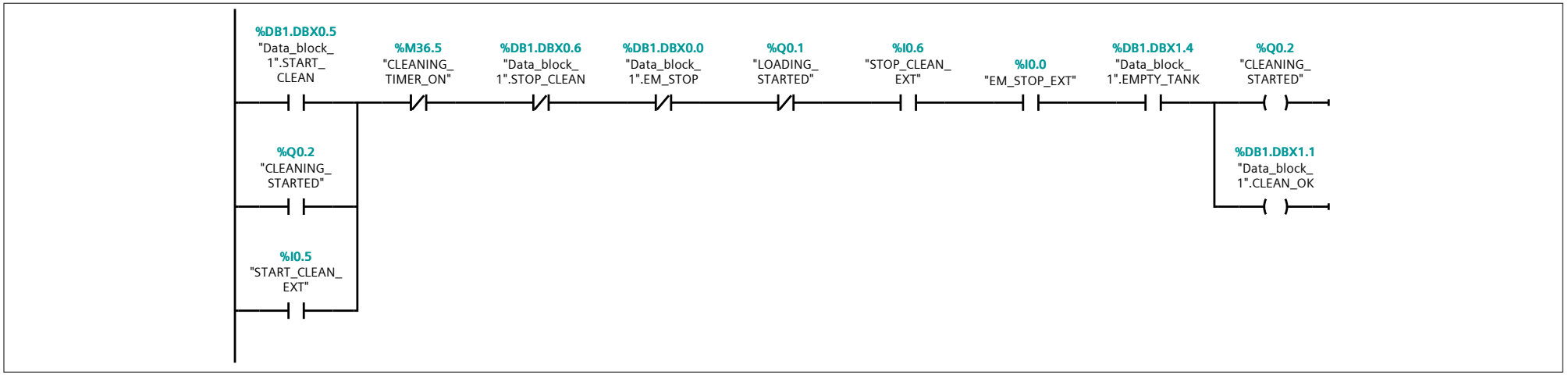
Network 1: MIXER



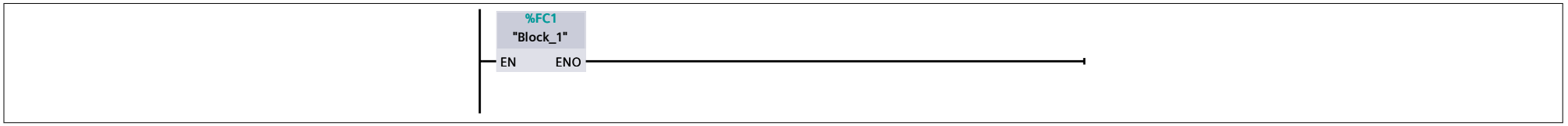
Network 2: LOADING PUMP



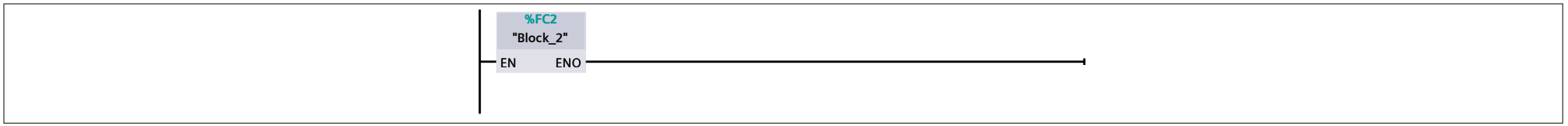
Network 3: CLEANING SYSTEM



Network 4: FC1 COMMUNICATION ENABLED

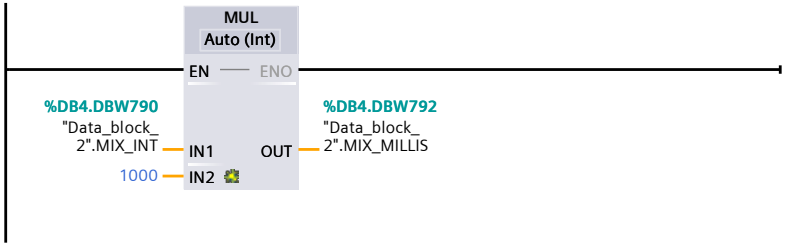


Network 5: FC2 COMMUNICATION ENABLED

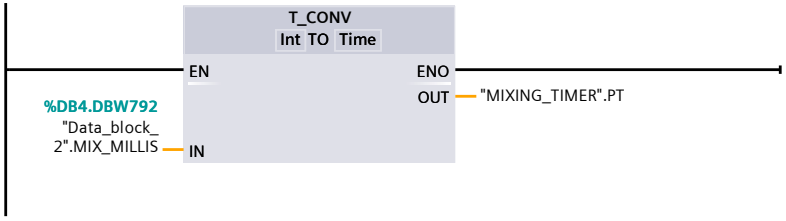


Network 6: MIXING TIME SP FROM USER TO INT CONVERSION

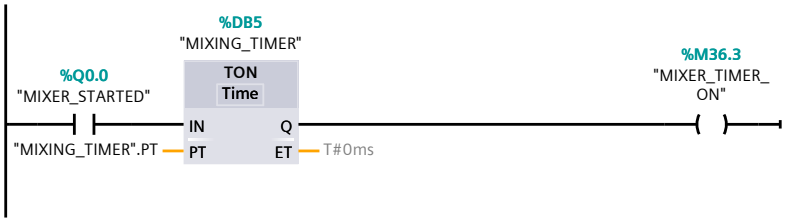
Network 7: MIXING INT TO MILLIS



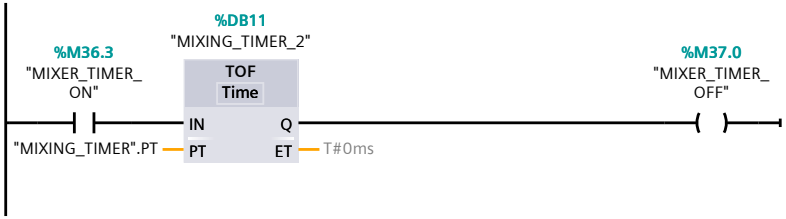
Network 8: MIXING MILLIS TO TIMER VALUE



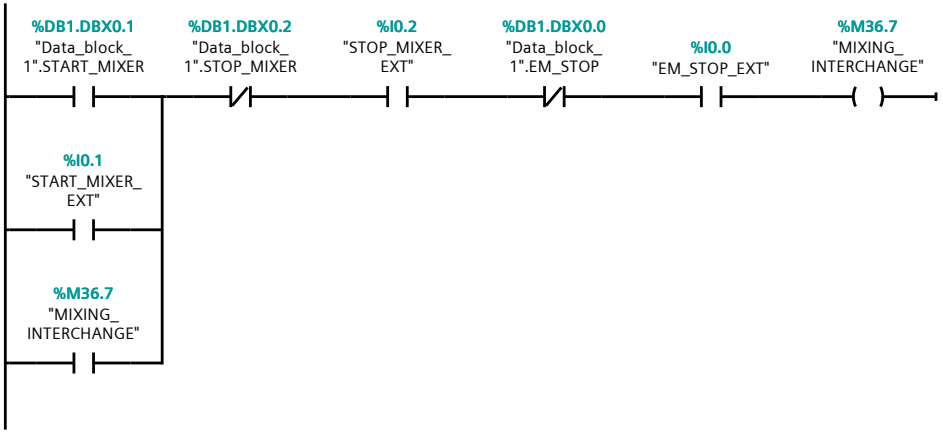
Network 9: MIXING TIMER



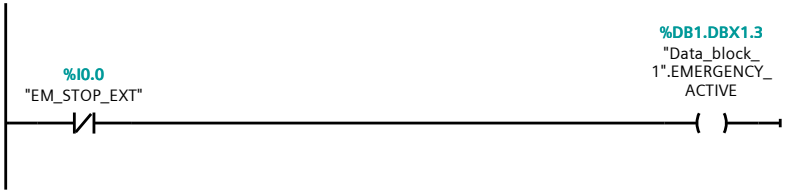
Network 10:



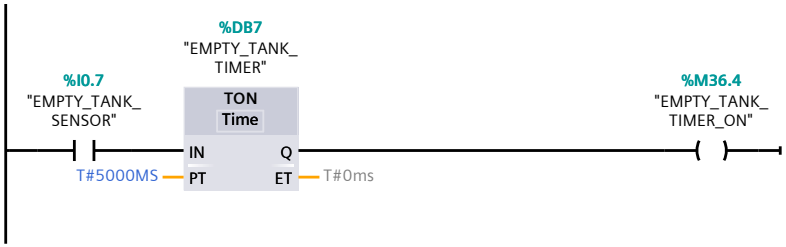
Network 11:



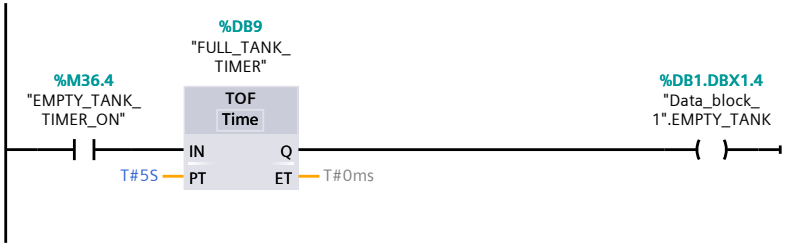
Network 12: EM. STOP BIT



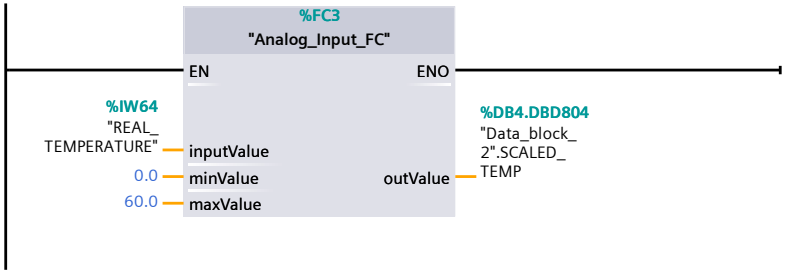
Network 13: EMPTY TANK TIMER



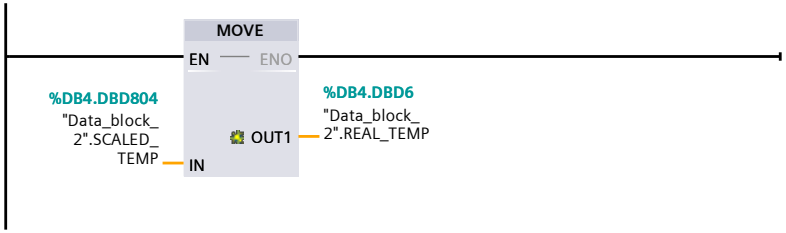
Network 14: FULL TANK TIMER



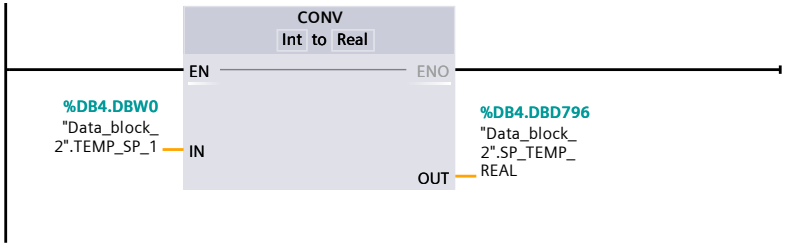
Network 15: ANALOG INPUT FC (FC3)



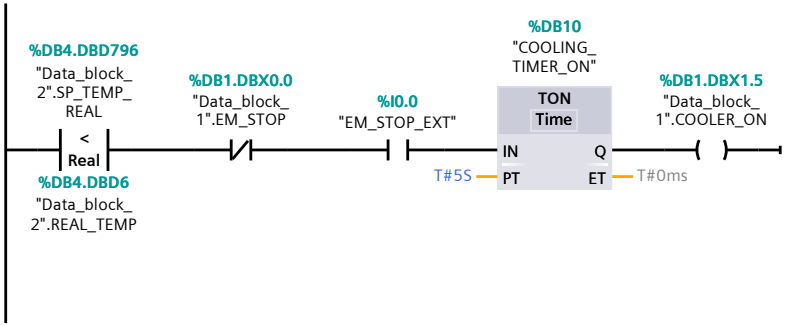
Network 16: SCALED TEMP TO REAL TEMP VARIABLE MOVE



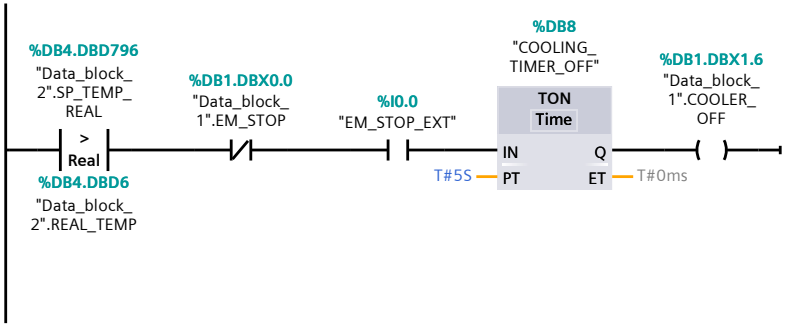
Network 17: TEMPERATURE SP FROM USER TO REAL CONVERSION



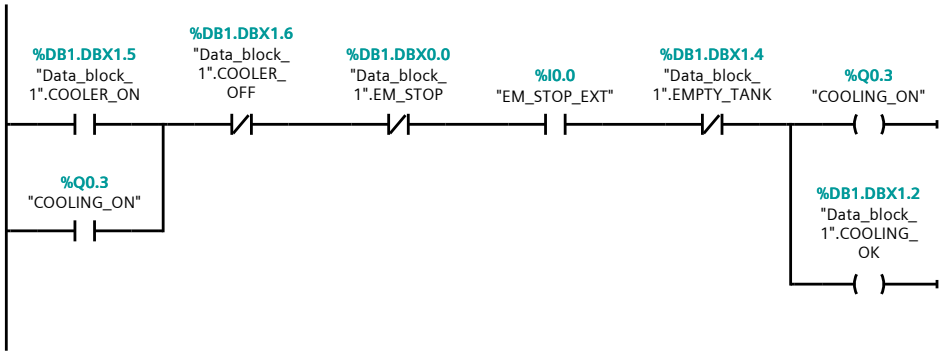
Network 18: TEMPERATURE COMPARISON AND COOLING ACTIVATION



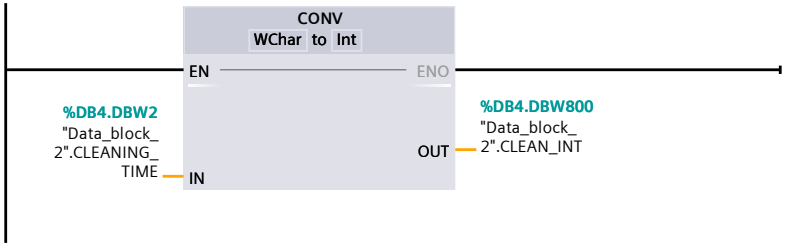
Network 19: TEMPERATURE COMPARISON AND COOLING DEACTIVATION



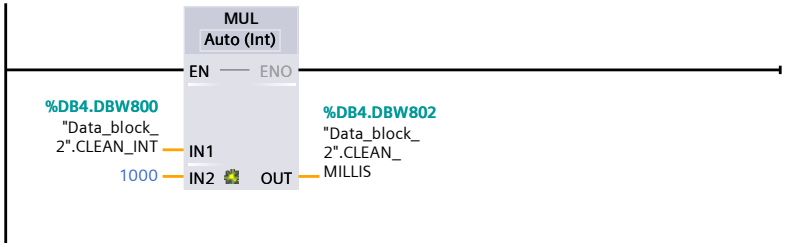
Network 20: COOLING



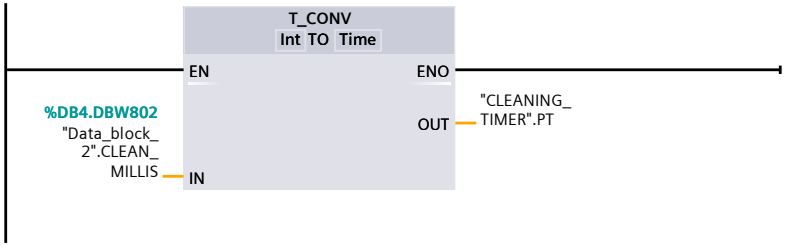
Network 21: USER CLEANING TIME SP TO INT



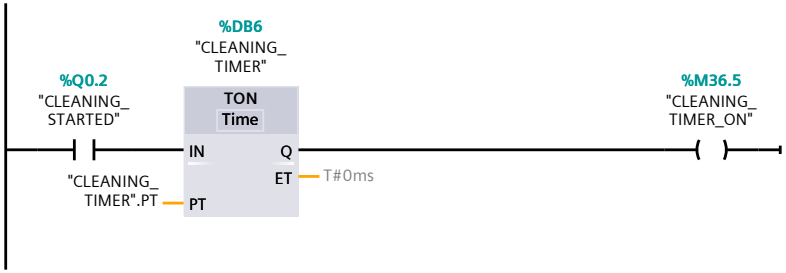
Network 22: CLEANING INT TO MILLIS



Network 23: CLEANING MILLIS TO TIME



Network 24: CLEANING TIMER



ΠΑΡΑΡΤΗΜΑ Β : ΚΩΔΙΚΑΣ ANDROID ΕΦΑΡΜΟΓΗΣ

Σε αυτό το παράρτημα παρατίθεται ο κώδικας της Android εφαρμογής, ο οποίος υλοποιήθηκε με τη χρήση του περιβάλλοντος του Android Studio, το οποίο αποτελεί το επίσημο περιβάλλον ανάπτυξης Android εφαρμογών και υποστηρίζεται από την Google. Η γλώσσα είναι βασισμένη σε Java με κάποιες διαφοροποιήσεις οι οποίες εφαρμόζονται για την προσαρμογή της στο android περιβάλλον. Στις παρακάτω σελίδες, τα κομμάτια κώδικα που φαίνονται ακολουθούν την ίδια σειρά με την οποία τα Activities εμφανίζονται κατά τη χρήση της εφαρμογής. Πιο συγκεκριμένα, πρώτα φαίνεται ο κώδικας του Login Activity, έπειτα του IP Activity, του Read/Write Activity, του I/O Control Activity, του Emergency Stop Triggered Activity και τέλος του Settings Activity. Στην επικεφαλίδα κάθε σελίδας, αναφέρεται το όνομα του Activity στο οποίο ανήκει ο κώδικας.

```java
package com.example.plccontrol;

import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class LoginActivity extends AppCompatActivity {

    public EditText name;
    public EditText password;
    public Button login;
    String loginName = SettingsActivity.getChangedName();
    String loginPass = SettingsActivity.getChangedPassword();


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login2);


        name = (EditText) findViewById(R.id.etName);
        password = (EditText) findViewById(R.id.etPassword);
        login = (Button) findViewById(R.id.loginbtn);

```

```
31          login.setOnClickListener(new View.OnClickListener() {
32              @Override
33              public void onClick(View v) {
34                  validate(name.getText().toString(), password.getText().toString());
35              }
36          });  //Function call to check credentials
37      }
38
39      public void validate (String userName, String userPassword){
40          if (userName.isEmpty() || userPassword.isEmpty()){
41              Toast.makeText(LoginActivity.this, "Empty Field", Toast.LENGTH_LONG).show();  //
    Check for empty field
42          }
43          else {
44              if ((userName.equals(loginName)) && (userPassword.equals(loginPass)) || (userName.
    equals("Admin")) && (userPassword.equals("1234"))) {
45                  Intent intent = new Intent(LoginActivity.this, IPActivity.class);
46                  startActivity(intent);
47              } else {
48                  Toast.makeText(LoginActivity.this, "Incorrect", Toast.LENGTH_LONG).show();
49              }
50          }  //Check if credentials are the right ones
51      }
52
53  }
```

```java
package com.example.plccontrol;

import org.w3c.dom.Text;

import android.app.Activity;
import android.content.Intent;
import android.os.AsyncTask;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import Moka7.S7;
import Moka7.S7Client;

public class IPActivity extends AppCompatActivity {

    public EditText address;
    public Button connect;
    public static String ipAddress;
    public String ipcheck;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ipactivity);

        address = (EditText) findViewById(R.id.etIP);
```

```
31          connect = (Button) findViewById(R.id.connectbtn);
32
33          connect.setOnClickListener(new View.OnClickListener() {
34              @Override
35              public void onClick(View v) {
36                  ipcheck = address.getText().toString();
37                  if(TextUtils.isEmpty(ipcheck)) {
38                      Toast.makeText(IPActivity.this, "Try Again", Toast.LENGTH_LONG).show();
    //Check for empty field
39                  }
40                  else {
41                      ipAddress = ipcheck;
42                      Intent intent = new Intent(IPActivity.this, ReadWriteActivity.class);
43                      startActivity(intent);
44                  }
45              }
46          });
47      }
48
49      public static String getAddress(){
50
51          return ipAddress;
52      }  //IP Address stored for other Activities to use
53  }
```

```java
package com.example.plccontrol;

import Moka7.S7;
import Moka7.S7Client;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Handler;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;


public class ReadWriteActivity extends AppCompatActivity {

    public Button btnnextpage;
    public ImageButton disconnect;
    public Button settings;
    public Button changeIP;
    public TextView Address;
    public String SetAddress = IPActivity.getAddress();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.readwriteactivity);

```

```
31          btnnextpage = findViewById(R.id.btnnxtpg);
32
33          btnnextpage.setOnClickListener(new View.OnClickListener(){
34                  @Override
35                  public void onClick(View v) {
36              Intent intentNext = new Intent(ReadWriteActivity.this, ioControlActivity.class);
37              startActivity(intentNext);
38          }
39          });  //If I/O Control is pressed, go to ioControl Activity
40
41          disconnect = findViewById(R.id.disconnectbtn);
42
43          disconnect.setOnClickListener(new View.OnClickListener() {
44              @Override
45              public void onClick(View v) {
46                  Intent intentDisconnect = new Intent(ReadWriteActivity.this, LoginActivity.
    class);
47                  startActivity(intentDisconnect);
48              }
49          });  //If Disconnect is pressed, go to Login Acitivity
50
51          settings = findViewById(R.id.settingsbtn);
52
53          settings.setOnClickListener(new View.OnClickListener() {
54              @Override
55              public void onClick(View v) {
56                  Intent intentSettings = new Intent( ReadWriteActivity.this, SettingsActivity.
    class);
57                  startActivity(intentSettings);
58              }
```

```
59          });  //If Settings is pressed, go to Settings Activity
60
61          changeIP = findViewById(R.id.changeipbtn);
62
63          changeIP.setOnClickListener(new View.OnClickListener() {
64              @Override
65              public void onClick(View v) {
66                  Intent intentIP = new Intent(ReadWriteActivity.this, IPActivity.class);
67                  startActivity(intentIP);
68              }
69          });  //if Change IP is pressed, go to IP Activity
70
71          Address = findViewById(R.id.tvAddress);
72          Address.setText(SetAddress);
73
74          content();  //Function call for reading initial values
75
76      }
77
78      S7Client client = new S7Client();
79
80      public void content(){
81          new PLCRead().execute("");
82
83          refresh(10000);
84      }
85
86      private void refresh(int milliseconds) {
87          final Handler handler = new Handler();
88          final Runnable runnable = new Runnable() {
```

```
89              @Override
90              public void run() {
91                  content();
92              }
93          };
94          handler.postDelayed(runnable, milliseconds);
95      }  //Refresh delay of the readings
96
97      public class PLCRead extends AsyncTask <String, Void, String>
98      {
99          String readdata1;
100         String readdata2;
101         String readdata3;
102         String readdata4;
103         String coolingState;
104         String CoolingOnOff;
105         Boolean CoolingBitCheck = null;
106
107         @Override
108         protected String doInBackground(String... params)
109         {
110             try
111             {
112                 client.SetConnectionType(S7.S7_BASIC);
113                 int readcheck = client.ConnectTo("" + SetAddress, 0, 1);  //connect to given
    address
114                 if (readcheck == 0)  //check connection
115                 {
116                     byte[] data1 = new byte[256];  //data buffer
117                     client.ReadArea(S7.S7AreaDB, 4, 0, 12, data1);  //where to read
```

```
118                     readdata1 = S7.GetFloatAt(data1, 6) + " °C";  //Temperature reading
119                     readdata2 = S7.GetWordAt(data1, 2) + " Seconds";  //Mixing time reading
120                     readdata3 = S7.GetWordAt(data1, 4) + " Seconds";  //Cleaning time reading
121                     readdata4 = S7.GetWordAt(data1,0) + " °C";  //Temperature setpoint
    reading
122
123
124                     byte[] data2 = new byte[256];
125                     client.ReadArea(S7.S7AreaDB, 1, 0, 12, data2);
126                     CoolingBitCheck = S7.GetBitAt(data2,1,2);  //Check if cooling is on
127                 }
128                 else
129                 {  //error printing
130                     readdata1 = "ERR: " + S7Client.ErrorText(readcheck);
131                     readdata2 = "ERR: " + S7Client.ErrorText(readcheck);
132                     readdata3 = "ERR: " + S7Client.ErrorText(readcheck);
133                     coolingState = "ERR: " + S7Client.ErrorText(readcheck);
134                 }
135                 client.Disconnect();
136             }
137             catch (Exception e)
138             {
139                 Thread.interrupted();
140             }
141             return "executed";
142         }
143
144         @Override
145         protected void onPostExecute(String result)
146         {  //printing the values
```

```
147             TextView textout1 = (TextView) findViewById(R.id.readdata1);
148             textout1.setText(readdata1);
149
150             TextView textout2 = (TextView) findViewById(R.id.readdata2);
151             textout2.setText(readdata2);
152
153             TextView textout3 = (TextView) findViewById(R.id.readdata3);
154             textout3.setText(readdata3);
155
156             TextView textout7 = (TextView) findViewById(R.id.readdata4);
157             textout7.setText((readdata4));
158
159             if (CoolingBitCheck == null){
160                 TextView textout4 = (TextView) findViewById(R.id.coolingState);
161                 textout4.setText(coolingState);
162             }
163
164             if (CoolingBitCheck != null) {
165                 if (CoolingBitCheck == true) {
166                     TextView textout5 = (TextView) findViewById(R.id.coolingState);
167                     CoolingOnOff = "Cooling is on";
168                     textout5.setText(CoolingOnOff);
169                 } else {
170                     TextView textout6 = (TextView) findViewById(R.id.coolingState);
171                     CoolingOnOff = "Cooling is off";
172                     textout6.setText(CoolingOnOff);
173                 }
174             }
175         }
176     }
```

```
177
178     public void writeData1(View v)
179     {
180
181         new PLCWrite1().execute("");
182     }
183
184     public class PLCWrite1 extends AsyncTask<String, Void, String>
185     {
186         @Override
187         protected String doInBackground(String... params)
188         {
189             try
190             {
191                 client.SetConnectionType(S7.S7_BASIC);
192                 int writecheck1 = client.ConnectTo("" + SetAddress, 0, 1);
193
194                 if (writecheck1 == 0) {
195                     EditText writedata1 = findViewById(R.id.etWrite1);
196                     String data1 = writedata1.getText().toString();  //Value field
197                     writedata1.getText().clear();  //Clear value field after writing
198
199                     byte[] dataBuffer1 = new byte[256];  //data buffer
200                     S7.SetWordAt(dataBuffer1, 0, Integer.parseInt(data1));
201                     client.WriteArea(S7.S7AreaDB, 4,0,2,dataBuffer1);  //where to write
202                 }
203
204                 else {
205
206                 }
```

```
207
208                 client.Disconnect();
209             }
210
211             catch(Exception e){
212                 Thread.interrupted();
213             }
214
215             return "execute";
216
217         }
218     }  //Class for writing Temperature Setpoint
219
220     public void writeData2(View v)
221     {
222
223         new PLCWrite2().execute("");
224     }
225
226     public class PLCWrite2 extends AsyncTask<String, Void, String>
227     {
228         @Override
229         protected String doInBackground(String... params)
230         {
231             try
232             {
233                 client.SetConnectionType(S7.S7_BASIC);
234                 int writecheck2 = client.ConnectTo("" + SetAddress, 0, 1);
235
236                 if (writecheck2 == 0) {
```

```
237                     EditText writedata2 = findViewById(R.id.etWrite2);
238                     String data2 = writedata2.getText().toString();
239                     writedata2.getText().clear();
240
241                     byte[] dataBuffer2 = new byte[256];
242                     S7.SetWordAt(dataBuffer2, 0, Integer.parseInt(data2));
243                     client.WriteArea(S7.S7AreaDB, 4,2,2,dataBuffer2);
244                 }
245
246                 else {
247
248                 }
249
250                 client.Disconnect();
251             }
252
253             catch(Exception e){
254                 Thread.interrupted();
255             }
256
257             return "execute";
258         }
259     }  //Class for writing the Mixing Time
260
261     public void writeData3(View v){
262
263         new PLCWrite3().execute("");
264     }
265
266     public class PLCWrite3 extends AsyncTask<String, Void, String>{
```

```
267         @Override
268         protected String doInBackground(String... params){
269             try {
270                 client.SetConnectionType(S7.S7_BASIC);
271                 int writecheck3 = client.ConnectTo("" + SetAddress, 0, 1);
272 
273                 if (writecheck3 == 0) {
274                     EditText writedata3 = findViewById(R.id.etWrite3);
275                     String data3 = writedata3.getText().toString();
276                     writedata3.getText().clear();
277 
278                     byte[] dataBuffer3 = new byte[256];
279                     S7.SetWordAt(dataBuffer3, 0, Integer.parseInt(data3));
280                     client.WriteArea(S7.S7AreaDB, 4,4,2,dataBuffer3);
281                 }
282 
283                 else {
284 
285                 }
286 
287                 client.Disconnect();
288             }
289 
290             catch(Exception e){
291                 Thread.interrupted();
292             }
293 
294             return "execute";
295         }
296     }  //Class for writing the Cleaning Time
```

```
297 }
```

```java
package com.example.plccontrol;

import static android.view.View.GONE;
import static android.view.View.VISIBLE;

import Moka7.S7;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Handler;
import android.os.Looper;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;


public class ioControlActivity extends AppCompatActivity {

    public String SetAddress = IPActivity.getAddress();
    Moka7.S7Client client = new Moka7.S7Client();
    public Handler dHandler = new Handler();
    public Runnable runnable;
    public Button strmix1;
    public Button stpmix1;
    public Button strloadpump1;
    public Button stploadpump1;
    public Button strclean1;
```

```
31      public Button stpclean1;
32      public Button emstop;
33      public ImageView stateImageMixer1;
34      public ImageView stateImageLoad1;
35      public ImageView stateImageClean1;
36      public ImageView stateImageEmpty;
37      public ProgressBar progressBar1;
38      public ProgressBar progressBar2;
39      public ProgressBar progressBar3;
40      byte buttonData[] = new byte[256];
41
42      @Override
43      protected void onCreate(Bundle savedInstanceState) {
44          super.onCreate(savedInstanceState);
45          setContentView(R.layout.iocontrolactivity);
46
47          strmix1 = findViewById(R.id.strmix1);
48          stpmix1 = findViewById(R.id.stpmix1);
49
50          strloadpump1 = findViewById(R.id.strloadpump1);
51          stploadpump1 = findViewById(R.id.stploadpump1);
52
53          strclean1 = findViewById(R.id.strclean1);
54          stpclean1 = findViewById(R.id.stpclean1);
55
56          emstop = findViewById(R.id.emstop);
57
58          stateImageMixer1 = findViewById(R.id.ivmixer1);
59          stateImageLoad1 = findViewById(R.id.ivloadpump1);
60          stateImageClean1 = findViewById(R.id.ivclean1);
```

```
61          stateImageEmpty = findViewById(R.id.ivemptystate);
62
63          progressBar1 = findViewById(R.id.progressBar1);
64          progressBar2 = findViewById(R.id.progressBar2);
65          progressBar3 = findViewById(R.id.progressBar3);
66
67          InitialBitCheck();  //Initial Check Function Call
68          EMStopCheck();     //Em. Stop Pressed Check Function Call
69
70      }
71
72      public void EMStopCheck(){
73          new EMStopInitialCheck().execute("");
74          EMStopRefresh(3000);
75      }
76
77      private void EMStopRefresh(int milliseconds) {
78          runnable = new Runnable() {
79              @Override
80              public void run() {
81
82                  EMStopCheck();
83              }
84          };
85          dHandler.postDelayed(runnable, milliseconds);
86      }  //Em. Stop Check Delay Function
87
88      private class EMStopInitialCheck extends AsyncTask<String, Void, String>{
89          Boolean EMStopBitCheck;
90          @Override
```

```
 91          protected String doInBackground(String...params){
 92              try{
 93                  client.SetConnectionType(S7.S7_BASIC);
 94                  int check = client.ConnectTo("" + SetAddress,0,1);
 95                  if(check == 0){
 96                      client.ReadArea(S7.S7AreaDB,1,0,10, buttonData);
 97                      EMStopBitCheck = S7.GetBitAt(buttonData, 1, 3);
 98                  }
 99                  else{
100
101                  }
102                  client.Disconnect();
103              }
104              catch (Exception e){
105                  Thread.interrupted();
106              }
107              return "execute";
108          }
109
110          @Override
111          protected void onPostExecute(String result) {
112              if(EMStopBitCheck != null) {
113                  if (EMStopBitCheck == true) {
114                      dHandler.removeCallbacks(runnable);
115                      Intent intentEStop = new Intent(ioControlActivity.this,
    ESTriggeredActivity.class);
116                      startActivity(intentEStop);
117                  }
118              }
119          }
```

```
120
121     } //Em. Stop Check if Pressed Function
122
123
124     public void InitialBitCheck(){
125
126         new initialCheck().execute("");
127
128         CheckRefresh(1000);
129     }
130
131     private void CheckRefresh(int milliseconds) {
132         final Handler handler = new Handler();
133         final Runnable runnable = new Runnable() {
134             @Override
135             public void run() {
136
137                 InitialBitCheck();
138             }
139         };
140         handler.postDelayed(runnable, milliseconds);
141     }  //Initial Check Delay Function
142
143     private class initialCheck extends AsyncTask<String, Void, String>{
144
145         Boolean MixerBitCheck;
146         Boolean LoadBitCheck;
147         Boolean CleanBitCheck;
148         Boolean EmptyBitCheck;
149         String EmptyState;
```

```
150
151         @Override
152         protected String doInBackground(String...params){
153             try{
154                 client.SetConnectionType(Moka7.S7.S7_BASIC);
155                 int check = client.ConnectTo("" + SetAddress, 0, 1);
156                 if (check == 0){
157                     client.ReadArea(Moka7.S7.S7AreaDB, 1, 0, 10, buttonData);
158                     MixerBitCheck = Moka7.S7.GetBitAt(buttonData, 0, 7);
159                     LoadBitCheck = Moka7.S7.GetBitAt(buttonData, 1,0);
160                     CleanBitCheck = Moka7.S7.GetBitAt(buttonData,1,1);
161                     EmptyBitCheck = S7.GetBitAt(buttonData,1,4);
162                 }
163                 else {
164
165                 }
166                 client.Disconnect();
167             }
168             catch (Exception e) {
169                 Thread.interrupted();
170             }
171
172             return "execute";
173         }
174
175           @Override
176           protected void onPostExecute(String result) {
177             if (MixerBitCheck != null) {
178                 if (MixerBitCheck == true) {
179                     stateImageMixer1.setImageResource(R.drawable.green_light);
```

```
                    progressBar1.setVisibility(VISIBLE);
                } else {
                    stateImageMixer1.setImageResource(R.drawable.red_light);
                    progressBar1.setVisibility(GONE);
                }
            }
            if (LoadBitCheck != null) {
                if (LoadBitCheck == true) {
                    stateImageLoad1.setImageResource(R.drawable.green_light);
                    progressBar2.setVisibility(VISIBLE);
                } else {
                    stateImageLoad1.setImageResource(R.drawable.red_light);
                    progressBar2.setVisibility(GONE);
                }
            }
            if (CleanBitCheck != null) {
                if (CleanBitCheck == true) {
                    stateImageClean1.setImageResource(R.drawable.green_light);
                    progressBar3.setVisibility(VISIBLE);
                } else {
                    stateImageClean1.setImageResource(R.drawable.red_light);
                    progressBar3.setVisibility(GONE);
                }
            }
            if (EmptyBitCheck != null) {
                if (EmptyBitCheck == true) {
                    TextView textout1 = (TextView) findViewById(R.id.textView14);
                    EmptyState = "Empty Tank Warning";
                    textout1.setText(EmptyState);
                    stateImageEmpty.setImageResource(R.drawable.red_light);
```

```
210                 } else {
211                     TextView textout1 = (TextView) findViewById(R.id.textView14);
212                     EmptyState = "Liquid Level OK";
213                     textout1.setText(EmptyState);
214                     stateImageEmpty.setImageResource(R.drawable.green_light);
215                 }
216             }
217         }
218     }  //Initial Bit Check Function
219
220     public void startmixer1(View v){
221
222         new StartMixer1().execute("");
223
224         final Handler handler = new Handler(Looper.getMainLooper());
225         handler.postDelayed(new Runnable() {
226             @Override
227             public void run() {
228                 new ResetStartMixer1().execute("");
229             }
230         }, 1000);
231     }  //Function executed after ''MIXER START'' is pressed
232
233     private class StartMixer1 extends AsyncTask <String, Void, String>{
234         Boolean MixerBitCheck;
235         @Override
236         protected String doInBackground(String...params){
237             try{
238                 client.SetConnectionType(Moka7.S7.S7_BASIC);
239                 int check = client.ConnectTo("" + SetAddress, 0, 1);
```

```
240                 if (check == 0){  //connection successful
241                     Moka7.S7.SetBitAt(buttonData, 0, 1, true);  //position to write
242                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
243                     MixerBitCheck = Moka7.S7.GetBitAt(buttonData, 0, 7);  //position to check
244                 }
245                 else {
246
247                 }
248                 client.Disconnect();
249             }
250             catch (Exception e) {
251                 Thread.interrupted();
252             }
253
254             return "execute";
255         }
256
257         @Override
258         protected void onPostExecute(String result) {
259             if (MixerBitCheck != null) {  //check if empty
260                 if (MixerBitCheck == true) {  //check if true
261                     stateImageMixer1.setImageResource(R.drawable.green_light);
262                 } else {
263                     stateImageMixer1.setImageResource(R.drawable.red_light);
264                 }
265             }
266         }
267     }  //Mixer Start Class
268
269     private class ResetStartMixer1 extends AsyncTask <String, Void, String>{
```

```
270         @Override
271         protected String doInBackground(String...params){
272             try{
273                 client.SetConnectionType(Moka7.S7.S7_BASIC);
274                 int check = client.ConnectTo("" + SetAddress, 0, 1);
275                 if (check == 0){
276                     //byte[] data = new byte[256];
277                     Moka7.S7.SetBitAt(buttonData, 0, 1, false);
278                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
279                 }
280                 else {
281 
282                 }
283                 client.Disconnect();
284             }
285             catch (Exception e) {
286                 Thread.interrupted();
287             }
288 
289             return "execute";
290 
291         }
292     }  //Start Value Reset class so the button works as a momentary push button
293 
294     public void stopmixer1(View v){
295 
296         new StopMixer1().execute("");
297 
298         final Handler handler = new Handler(Looper.getMainLooper());
299         handler.postDelayed(new Runnable() {
```

```
300             @Override
301             public void run() {
302                 new ResetStopMixer1().execute("");
303             }
304         }, 1000);
305     }
306
307     public class StopMixer1 extends AsyncTask <String, Void, String>{
308         @Override
309         protected String doInBackground(String...params){
310             try{
311                 client.SetConnectionType(Moka7.S7.S7_BASIC);
312                 int check = client.ConnectTo("" + SetAddress, 0, 1);
313                 if (check == 0){
314                     //byte[] stop1 = new byte[256];
315                     Moka7.S7.SetBitAt(buttonData, 0, 2, true);
316                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
317                     stateImageMixer1.setImageResource(R.drawable.red_light);
318                 }
319                 else {
320
321                 }
322                 client.Disconnect();
323             }
324             catch (Exception e) {
325                 Thread.interrupted();
326             }
327             return "execute";
328         }
329     }
```

```
330
331     public class ResetStopMixer1 extends AsyncTask <String, Void, String>{
332         @Override
333         protected String doInBackground(String...params){
334             try{
335                 client.SetConnectionType(Moka7.S7.S7_BASIC);
336                 int check = client.ConnectTo("" + SetAddress, 0, 1);
337                 if (check == 0){
338                     //byte[] stop1 = new byte[256];
339                     Moka7.S7.SetBitAt(buttonData, 0, 2, false);
340                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
341                 }
342                 else {
343
344                 }
345                 client.Disconnect();
346             }
347             catch (Exception e) {
348                 Thread.interrupted();
349             }
350             return "execute";
351         }
352     }
353
354     public void startlp1(View v){
355
356         new StartLP1().execute("");
357
358         final Handler handler = new Handler(Looper.getMainLooper());
359         handler.postDelayed(new Runnable() {
```

```
360             @Override
361             public void run() {
362                 new ResetStartLP1().execute("");
363             }
364         }, 1000);
365     }
366
367     public class StartLP1 extends AsyncTask <String, Void, String>{
368         Boolean LoadBitCheck;
369         @Override
370         protected String doInBackground(String...params){
371             try{
372                 client.SetConnectionType(Moka7.S7.S7_BASIC);
373                 int check = client.ConnectTo("" + SetAddress, 0, 1);
374                 if (check == 0){
375                     //byte[] start2 = new byte[256];
376                     Moka7.S7.SetBitAt(buttonData, 0, 3, true);
377                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
378                     LoadBitCheck = Moka7.S7.GetBitAt(buttonData,1,0);
379                 }
380                 else {
381
382                 }
383                 client.Disconnect();
384             }
385             catch (Exception e) {
386                 Thread.interrupted();
387             }
388             return "execute";
389         }
```

```
390
391         @Override
392         protected void onPostExecute(String result) {
393             if (LoadBitCheck != null) {
394                 if (LoadBitCheck == true) {
395                     stateImageLoad1.setImageResource(R.drawable.green_light);
396                 } else {
397                     stateImageLoad1.setImageResource(R.drawable.red_light);
398                 }
399             }
400         }
401
402     }
403
404     public class ResetStartLP1 extends AsyncTask <String, Void, String>{
405         @Override
406         protected String doInBackground(String...params){
407             try{
408                 client.SetConnectionType(Moka7.S7.S7_BASIC);
409                 int check = client.ConnectTo("" + SetAddress, 0, 1);
410                 if (check == 0){
411                     //byte[] start2 = new byte[256];
412                     Moka7.S7.SetBitAt(buttonData, 0, 3, false);
413                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
414                 }
415                 else {
416
417                 }
418                 client.Disconnect();
419             }
```

```
            catch (Exception e) {
                Thread.interrupted();
            }
            return "execute";
        }

    }

    public void stoplp1(View v){

        new StopLP1().execute("");

        final Handler handler = new Handler(Looper.getMainLooper());
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                new ResetStopLP1().execute("");
            }
        }, 1000);
    }


    public class StopLP1 extends AsyncTask <String, Void, String>{
        @Override
        protected String doInBackground(String...params) {
            try {
                client.SetConnectionType(Moka7.S7.S7_BASIC);
                int check = client.ConnectTo("" + SetAddress, 0, 1);
                if (check == 0) {
                    Moka7.S7.SetBitAt(buttonData, 0, 4, true);
```

```
450                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
451                     stateImageLoad1.setImageResource(R.drawable.red_light);
452                 }
453                 else{
454 
455                 }
456 
457                 client.Disconnect();
458             } catch (Exception e) {
459                 Thread.interrupted();
460             }
461             return "execute";
462         }
463 
464     }
465 
466     public class ResetStopLP1 extends AsyncTask <String, Void, String>{
467         @Override
468         protected String doInBackground(String...params) {
469             try {
470                 client.SetConnectionType(Moka7.S7.S7_BASIC);
471                 int check = client.ConnectTo("" + SetAddress, 0, 1);
472                 if (check == 0) {
473                     Moka7.S7.SetBitAt(buttonData, 0, 4, false);
474                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
475                 }
476                 else{
477 
478                 }
479 
```

```
480                 client.Disconnect();
481             } catch (Exception e) {
482                 Thread.interrupted();
483             }
484             return "execute";
485         }
486
487     }
488
489     public void startclean1(View v){
490
491         new StartClean1().execute("");
492
493         final Handler handler = new Handler(Looper.getMainLooper());
494         handler.postDelayed(new Runnable() {
495             @Override
496             public void run() {
497                 new ResetStartClean1().execute("");
498             }
499         }, 1000);
500     }
501
502     public class StartClean1 extends AsyncTask <String, Void, String>{
503         Boolean CleanBitCheck;
504         @Override
505         protected String doInBackground(String...params){
506             try{
507                 client.SetConnectionType(Moka7.S7.S7_BASIC);
508                 int check = client.ConnectTo("" + SetAddress, 0, 1);
509                 if (check == 0){
```

```
510                     //byte[] start2 = new byte[256];
511                     Moka7.S7.SetBitAt(buttonData, 0, 5, true);
512                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
513                     CleanBitCheck = Moka7.S7.GetBitAt(buttonData, 1,1);
514                 }
515                 else {
516 
517                 }
518                 client.Disconnect();
519             }
520             catch (Exception e) {
521                 Thread.interrupted();
522             }
523             return "execute";
524         }
525 
526         @Override
527         protected void onPostExecute(String result) {
528             if (CleanBitCheck != null) {
529                 if (CleanBitCheck == true) {
530                     stateImageClean1.setImageResource(R.drawable.green_light);
531                 } else {
532                     stateImageClean1.setImageResource(R.drawable.red_light);
533                 }
534             }
535         }
536 
537     }
538 
539     public class ResetStartClean1 extends AsyncTask <String, Void, String>{
```

```
        @Override
        protected String doInBackground(String...params){
            try{
                client.SetConnectionType(Moka7.S7.S7_BASIC);
                int check = client.ConnectTo("" + SetAddress, 0, 1);
                if (check == 0){
                    //byte[] start2 = new byte[256];
                    Moka7.S7.SetBitAt(buttonData, 0, 5, false);
                    client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
                }
                else {

                }
                client.Disconnect();
            }
            catch (Exception e) {
                Thread.interrupted();
            }
            return "execute";
        }

    }

    public void stopclean1(View v){

        new StopClean1().execute("");

        final Handler handler = new Handler(Looper.getMainLooper());
        handler.postDelayed(new Runnable() {
            @Override
```

```
            public void run() {
                new ResetStopClean1().execute("");
            }
        }, 1000);
    }


    public class StopClean1 extends AsyncTask <String, Void, String>{
        @Override
        protected String doInBackground(String...params) {
            try {
                client.SetConnectionType(Moka7.S7.S7_BASIC);
                int check = client.ConnectTo("" + SetAddress, 0, 1);
                if (check == 0) {
                    Moka7.S7.SetBitAt(buttonData, 0, 6, true);
                    client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
                    stateImageClean1.setImageResource(R.drawable.red_light);
                }
                else{

                }

                client.Disconnect();
            } catch (Exception e) {
                Thread.interrupted();
            }
            return "execute";
        }

    }
```

```
600
601     public class ResetStopClean1 extends AsyncTask <String, Void, String>{
602         @Override
603         protected String doInBackground(String...params) {
604             try {
605                 client.SetConnectionType(Moka7.S7.S7_BASIC);
606                 int check = client.ConnectTo("" + SetAddress, 0, 1);
607                 if (check == 0) {
608                     Moka7.S7.SetBitAt(buttonData, 0, 6, false);
609                     client.WriteArea(Moka7.S7.S7AreaDB, 1, 0, 12, buttonData);
610                 }
611                 else{
612
613                 }
614
615                 client.Disconnect();
616             } catch (Exception e) {
617                 Thread.interrupted();
618             }
619             return "execute";
620         }
621
622     }
623
624     public void emergencyStop (View v){
625         new emStop().execute("");
626         Intent intentES = new Intent(ioControlActivity.this, ESTriggeredActivity.class);
627         startActivity(intentES);
628
629     }  //If Em. Stop is pushed, go to ESTriggered Activity
```

```
630
631     public class emStop extends AsyncTask <String, Void, String>{
632         @Override
633         protected String doInBackground(String...params){
634             try{
635                 client.SetConnectionType(S7.S7_BASIC);
636                 int check = client.ConnectTo("" + SetAddress,0,1);
637                 if(check == 0){
638                     S7.SetBitAt(buttonData,0,0,true);
639                     client.WriteArea(S7.S7AreaDB,1,0,12, buttonData);
640                 }
641                 else{
642
643                 }
644                 client.Disconnect();
645             } catch (Exception e){
646                 Thread.interrupted();
647             }
648             return "execute";
649         }
650     }  //Class for Em. Stop
651 }
```

```java
 package com.example.plccontrol;

import android.content.Intent;
import android.os.AsyncTask;
import android.view.View;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class ESTriggeredActivity extends AppCompatActivity {

    public String SetAddress = IPActivity.getAddress();
    Moka7.S7Client client = new Moka7.S7Client();
    byte buttonData[] = new byte[256];


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_estriggered);

    }

    public void EmergencyStopReset (View v){
        new EMStopResetCheck().execute("");
        new EMStopReset().execute("");
    }

    private class EMStopResetCheck extends AsyncTask<String, Void, String>{
        Boolean EMStopBitCheck;
        @Override
```

```
31          protected String doInBackground(String...params){
32              try{
33                  client.SetConnectionType(Moka7.S7.S7_BASIC);
34                  int check = client.ConnectTo("" + SetAddress,0,1);
35                  if(check == 0){
36                      client.ReadArea(Moka7.S7.S7AreaDB,1,0,10, buttonData);
37                      EMStopBitCheck = Moka7.S7.GetBitAt(buttonData, 1, 3);  //Check if Em. Stop
    position is Reset
38                  }
39                  else{
40
41                  }
42                  client.Disconnect();
43              }
44              catch (Exception e){
45                  Thread.interrupted();
46              }
47              return "execute";
48          }
49
50      }  //Check if Em. Stop is reset
51
52      public class EMStopReset extends AsyncTask<String, Void, String> {
53          @Override
54          protected String doInBackground(String...params){
55              try{
56                  client.SetConnectionType(Moka7.S7.S7_BASIC);
57                  int check = client.ConnectTo("" + SetAddress,0,1);
58                  if(check == 0){
59                      Moka7.S7.SetBitAt(buttonData,0,0,false);  //If Reset is pressed, change
```

```
59 the Em. Stop value to PLC
60                     client.WriteArea(Moka7.S7.S7AreaDB,1,0,12, buttonData);
61                     Intent intentReset = new Intent(ESTriggeredActivity.this,
   ioControlActivity.class);  //Go to I/O Control
62                     startActivity(intentReset);
63                 }
64                 else{
65
66                 }
67                 client.Disconnect();
68             } catch (Exception e){
69                 Thread.interrupted();
70             }
71             return "execute";
72         }
73     }  //Check if Reset is pressed
74 }
75
76
```

```java
 package com.example.plccontrol;

import android.content.Intent;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class SettingsActivity extends AppCompatActivity {

    public static String changedName;
    public static String changedPassword;
    EditText username;
    EditText password;
    Button setbtn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        username = (EditText) findViewById(R.id.etNameChange);
        password = (EditText) findViewById(R.id.etPasswordChange);
        setbtn = (Button) findViewById(R.id.credsetbtn);


        setbtn.setOnClickListener(new View.OnClickListener() {
```

```
31              @Override
32              public void onClick(View v) {
33                  changedName = username.getText().toString();
34                  changedPassword = password.getText().toString();
35                  if(TextUtils.isEmpty(changedName) || TextUtils.isEmpty(changedPassword)) {
36                      Toast.makeText(SettingsActivity.this, "Try Again", Toast.LENGTH_LONG).show
   ();  //Check for empty field
37                  }
38                  else {
39                      Intent intentCredSet = new Intent(SettingsActivity.this, LoginActivity.
   class);  //Go to Login Activity
40                      startActivity(intentCredSet);
41                  }
42              }
43          });  //After credentials are set, go to Login Activity
44
45      }
46      public static String getChangedName(){
47
48          return changedName;
49      }  //Name stored
50
51      public static String getChangedPassword(){
52
53          return changedPassword;
54      }  //Password stored
55  }
```

ΠΑΡΑΡΤΗΜΑ Γ : ΚΩΔΙΚΑΣ ESP32

Για τη λήψη και τη μετάδοση της θερμοκρασίας χρησιμοποιήθηκε το ESP32 σε συνδυασμό με έναν αισθητήρα DHT11, μια οθόνη OLED και μια ενισχυτική βαθμίδα, όπως εξηγήθηκε σε αντίστοιχο κεφάλαιο. Ο προγραμματισμός έγινε στο Arduino IDE, με τη χρήση κατάλληλων βιβλιοθηκών οι οποίες εξυπηρετούν στη λήψη και τη μετάδοση της θερμοκρασίας.

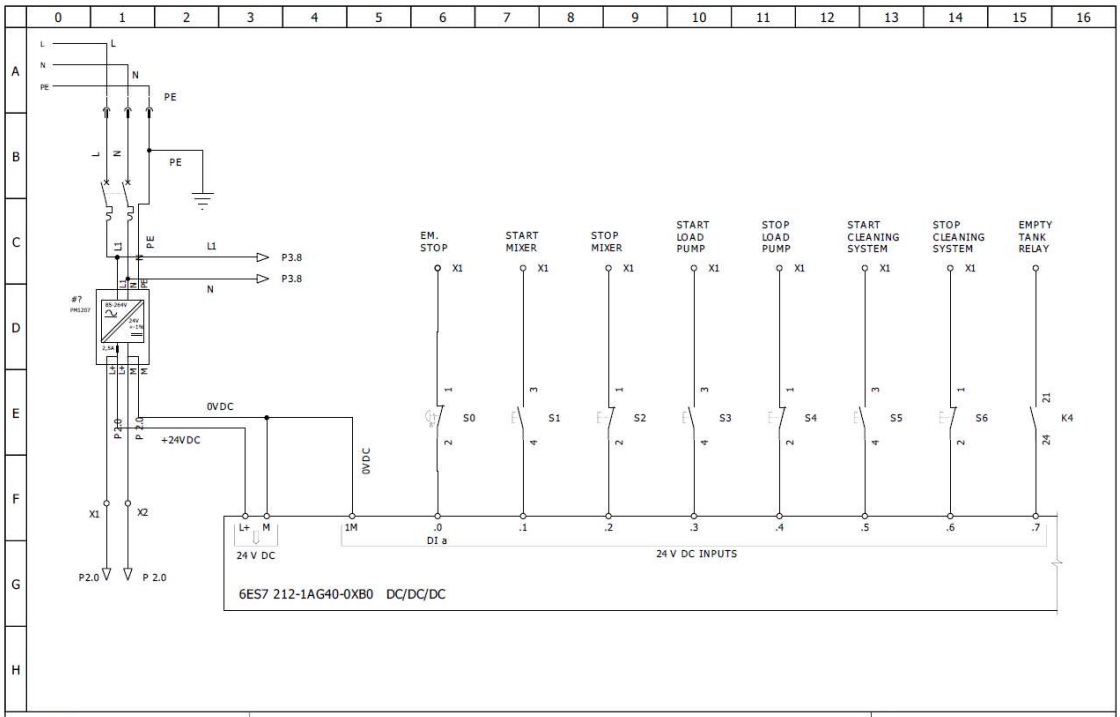
```
#include <Wire.h>
#include<Adafruit_GFX.h>
#include<Adafruit_SSD1306.h>
#include<Adafruit_Sensor.h>
#include <DHT.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define DHTPIN 25    // Digital pin connected to the DHT sensor

#define DHTTYPE    DHT11     // Sensor used: DHT 11

DHT dht(DHTPIN, DHTTYPE);

String readDHTTemperature() {
  // Read temperature as Celsius
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}

String readDHTHumidity() {
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
```

```
  else {
    Serial.println(h);
    return String(h);
  }
}

void setup() {
 Serial.begin(115200);

 dht.begin();

 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
   Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
 delay(2000);
 display.clearDisplay();
 display.setTextColor(WHITE);

}

void loop() {
 delay(5000);

 //Read temperature and humidity
 float t = dht.readTemperature();
 float h = dht.readHumidity();
 if (isnan(h) || isnan(t)) {
   Serial.println("Failed to read from DHT sensor!");
  }
 // clear display
 display.clearDisplay();

 // display temperature
 display.setTextSize(1);
 display.setCursor(0,0);
 display.print("Temperature: ");
 display.setTextSize(2);
 display.setCursor(0,10);
```

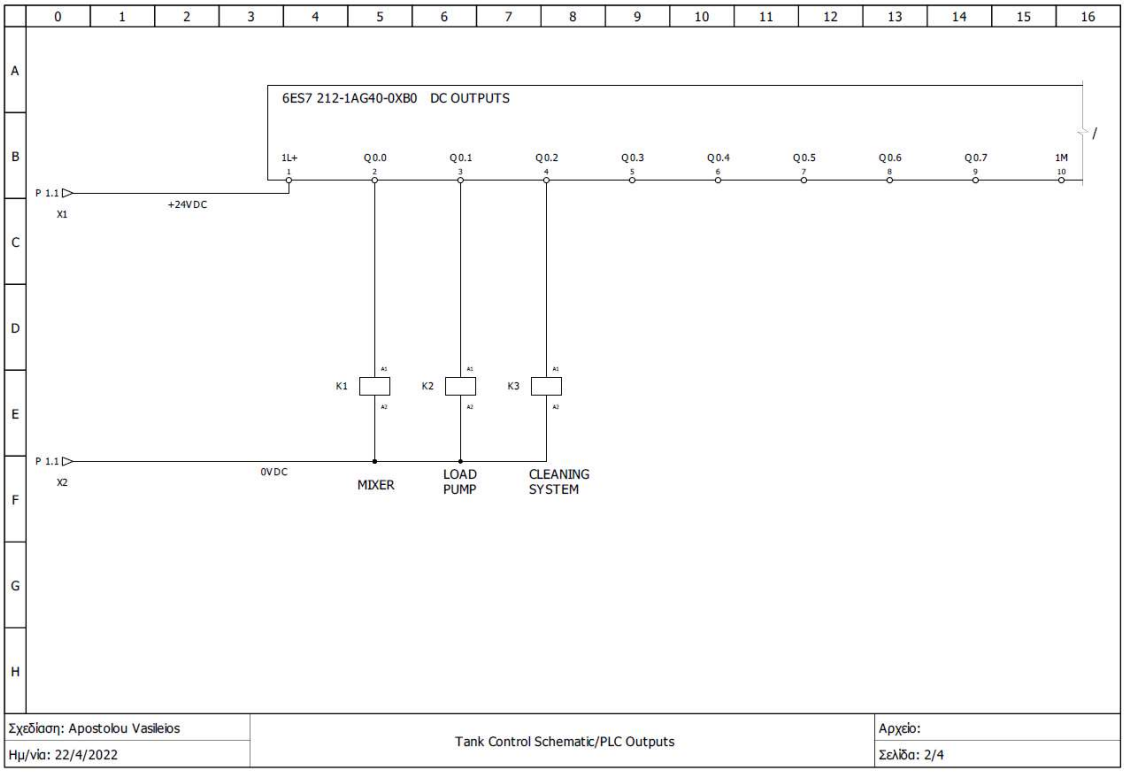
```
 display.print(t);
 Serial.println(t);
  display.print(" ");
display.setTextSize(1);
 display.cp437(true);
 display.write(167);
display.setTextSize(2);
 display.print("C");

  // display humidity
display.setTextSize(1);
 display.setCursor(0, 35);
 display.print("Humidity: ");
display.setTextSize(2);
 display.setCursor(0, 45);
 display.print(h);
  display.print(" %");

 display.display();

  // Value calculation for DAC
  float y = 5.1*t;

  // DAC Transmission
 dacWrite(26,y);
 delay(1000);
}
```

ΠΑΡΑΡΤΗΜΑ Δ : ΗΛΕΚΤΡΟΛΟΓΙΚΟ ΣΧΕΔΙΟ ΠΙΝΑΚΑ

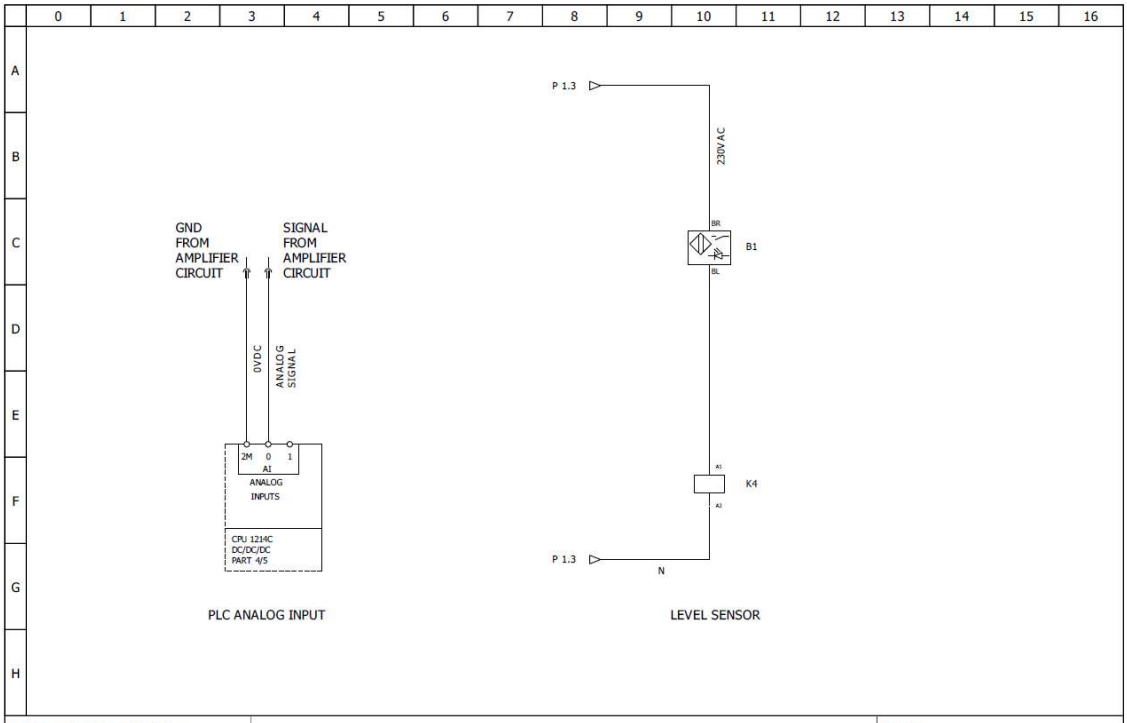
Παρακάτω φαίνεται το ηλεκτρολογικό σχέδιο του πίνακα αυτοματισμού του συστήματος. Απεικονίζονται όλες οι εξωτερικές λειτουργίες της εφαρμογής και η διασύνδεση μεταξύ των στοιχείων. Η σχεδίαση του έγινε με τη χρήση του προγράμματος QElectroTech, το οποίο είναι δωρεάν και Open Source λογισμικό.



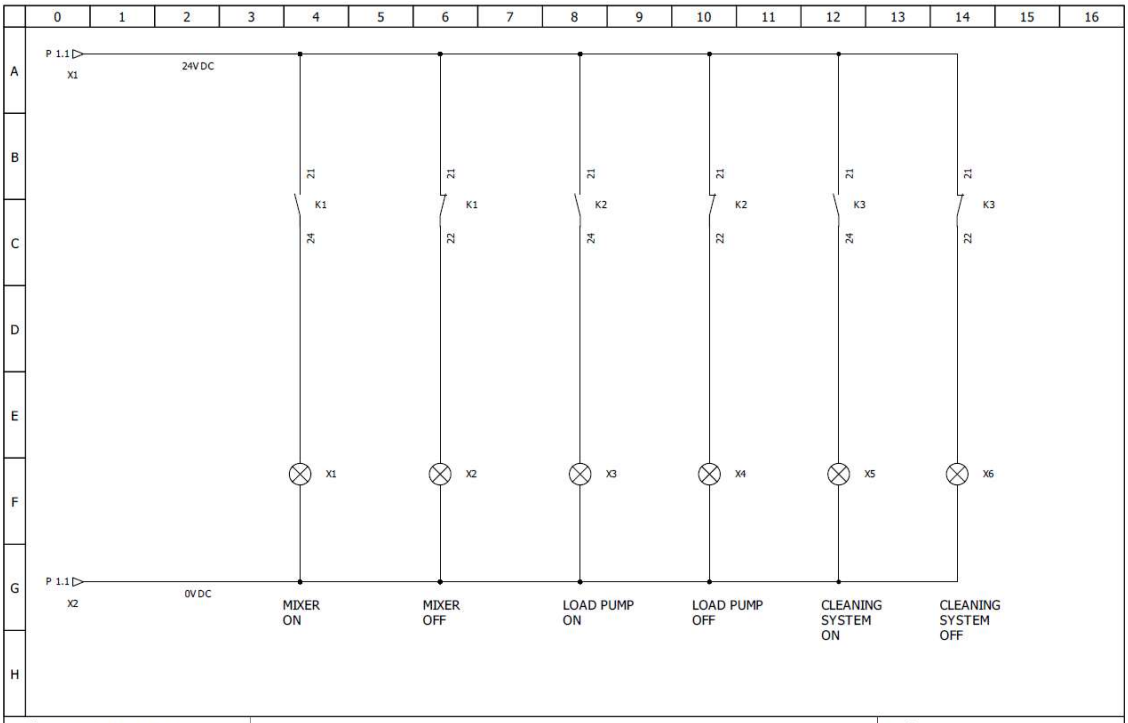
Σχεδίαση: Apostolou Vasileios	Tank Control Schematic/Main & PLC Inputs	Αρχείο:
Ημ/νία: 22/4/2022		Σελίδα: 1/4



Σχεδίαση: Apostolou Vasileios	Tank Control Schematic/PLC Outputs	Αρχείο:
Ημ/νία: 22/4/2022		Σελίδα: 2/4



Σχεδίαση: Apostolou Vasileios	Tank Control/Analog Input & Level Input	Αρχείο:
Ημ/νία: 22/4/2022		Σελίδα: 3/4



Σχεδίαση: Apostolou Vasileios	Tank Control/Indicators	Αρχείο:
Ημ/νία: 22/4/2022		Σελίδα: 4/4