

Filière

Systèmes industriels

Orientation Infotronics

Rapport intermédiaire

Diplôme 2025

Marcelin Puiinne

SoftPLC pour l'IoT

Professeur
HEI-Vs, Prof. Métrailler Christopher, christopher.metrailler@hevs.ch

Expert
WAGO Contact SA, Stéphane Rey, stephane.rey@wago.com

Date de soumission
17 February 2025



Informations sur ce rapport

Coordonnées

Auteur: Marcelin Puippe
Bachelor Étudiante
HEI-Vs
E-Mail: marcelin.puippe@hevs.ch

Déclaration sur l'honneur

Je soussigné(e), déclare par la présente que le travail soumis est le résultat d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à d'autres formes de fraude. Toutes les sources d'information utilisées et les citations d'auteurs ont été clairement mentionnées.

Lieu, date: Sion, 24.06.2025

Signature:



Contenu

Remerciements	1
Donnée de travail de bachelor	2
Développement durable	4
1 Introduction	5
1.1 Point de départ	5
1.2 Objectif	5
1.3 Application de démonstration pour maison connectée	6
2 WDx	7
2.1 Accéder IO	7
2.1.1 Sans wda	7
2.1.2 Avec wda	7
3 Etat de l'art	9
3.1 n8n [1]	9
3.2 total js [2]	11
3.3 Analyse critique du code existant	11
3.3.1 Ajouter des blocs simples	12
3.3.2 Erreurs non gérées	14
4 Conception	16
4.1 Environnement de développement	17
4.2 Aperçu du système	18
4.3 Bloc de haut niveau	19
4.4 User Interface (UI) Design	20
4.4.1 Vue programmation	20
4.4.2 Vue WebSocket	20
4.5 Gestion et stockage des données	23
4.5.1 Node MQTT	23
4.5.2 Node HTTP client	24
4.5.3 Node HTTP serveur	25
4.5.4 Node webSocket output	25
4.5.5 WebSocket data	26
4.6 Conclusion	27
5 Implémentation	28
5.1 WDA	29
5.1.1 inputUpdate.go	29
5.1.2 outputUpdate.go	30
5.2 Interface webSocket	30
5.3 Intégration des blocs logiques simples	30
5.3.1 Bloc Bool to String	30
5.3.2 Bloc String to Bool	31
5.4 Intégration des blocs logiques complexes	34
5.4.1 MQTT	34
5.4.2 WebSocket	34

5.4.3 HTTP	34
5.5 Conclusion	34
6 Validation	35
6.1 Section 1	36
6.2 Section 2	36
6.3 Conclusion	36
7. Conclusion	37
7.1. Résumé du projet	37
7.1.1. Fonctionnalités développées :	37
7.1.2. changement de l'interface	37
7.2. Comparaison avec les objectifs initial	38
7.3. Difficultés rencontrées	38
7.4. Perspectives d'avenir	39
7.4.1. Idées d'amélioration et extensions du frontend web	39
8 Annexe	40
8.1 Planning	41
8.2 WDA Monitoring Lists	46
8.2.1 Création	46
8.2.2 Utilisation	47
8.3 WDA access mode	52
8.4 WDA I/O	53
8.4.1 Configuration des DIO (activation des Outputs) via WDA	53
8.4.2 Activation d'une Output via WDA	54
Glossaire	55

Remerciements

Je tiens à remercier mon répondant de travail de bachelor, M. Christopher Métrailler ainsi que son assitant Michael Clausen qui ont répondu à mes différentes questions, pour leurs conseilles et pour avoir pris de leur temps pour mon travail. Je remercie aussi Arnaud Ducrey pour son TB documenté.

Donnée de travail de bachelor

HES-SO Valais

SYND	ETE	TEVI
X	X	X

Données du travail de bachelor Aufgabenstellung der Bachelorarbeit

FO 1.2.02.07.FB
che/13.03.2024

Filière / Studiengang SYND	Année académique / Studienjahr 2024-25	No TB / Nr. BA IT/2025/5
Mandant / Auftraggeber <input type="checkbox"/> HES—SO Valais-Wallis <input checked="" type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>	Etudiant-e / Student/in Marcelin Puipe	Lieu d'exécution / Ausführungsort <input checked="" type="checkbox"/> HES—SO Valais-Wallis <input type="checkbox"/> Industrie <input type="checkbox"/> Etablissement partenaire <i>Partnerinstitution</i>
Travail confidentiel / vertrauliche Arbeit <input type="checkbox"/> oui / ja <input checked="" type="checkbox"/> non / nein	Expert-e / Experte/Expertin (nom, prénom, E-Mail / Name, Vorname, E-Mail) Stéphane Rey, WAGO Contact SA, stephane.rey@wago.com	

Titre SoftPLC pour IoT
<p>Description</p> <p>L'entreprise WAGO commercialise des automates programmables industriels basés Linux, également équipé d'un environnement SoftPLC basé Codesys. Un PoC d'un environnement SoftPLC basé Web a été développé dans un précédent projet. Ce travail de diplôme consiste à développer un nouveau HAL permettant d'intégrer les nouvelles interfaces des PLC WAGO CC100 (751-9401 et 751-9402), tel qu'une interface CAN. L'environnement SoftPLC doit être complété afin d'intégrer des modules I/O plus complexes. De nouveaux composants doivent être développés afin d'utiliser ces derniers de manière graphique dans le SoftPLC. Ces développements doivent permettre d'intégrer aisément de nouvelles interfaces et capteurs au PLC WAGO dans le futur, comme des IoT disponibles dans des maisons connectées (CAN, HTTP/MQTT, WiFi, etc.). Un banc de test équipé de plusieurs capteurs permettra de valider le bon fonctionnement des développements.</p> <p>Objectifs du projet individuel (Pr4)</p> <ul style="list-style-type: none"> — Étude et analyse du projet SoftPLC existant. Prise en main de l'environnement de développement et des outils Docker pour le déploiement d'application sur les PLC Wago CC100 — Développement d'applications de démonstration basées Golang (backend) et Javascript/TypeScript (frontend) pour le système existant en utilisant le banc de test existant comme base <p>Objectifs du travail de Bachelor (TB)</p> <ul style="list-style-type: none"> — Modification du Hardware Abstraction Layer (HAL) pour les nouveaux automates CC100. Développement d'un nouveau HAL pour les I/O et le module CAN en utilisant l'API VDX — Définition et implémentation de nouveaux blocs SoftPLC comprenant des blocs de haut niveau, tels que client MQTT, CAN, WebServer, client/serveur HTTP, etc. Extension du système SoftPLC existant et de l'interface web pour permettre l'ajout de ces nouveaux composants — Amélioration et extensions du frontend web avec contrôle du type d'entrées/sorties, des types des signaux. Amélioration de l'interface et de l'expérience utilisateur avec un meilleur visuel et des contrôles automatisés — Développement d'un banc de test physique et d'une application de démonstration pour une maison connectée. Documentation et tests et rédaction du rapport, poster et présentation.

Délais / Termine	
Démarrage du projet individuel (Pr4) <i>Start des individuellen Projekts (Pr4)</i>	Remise du rapport final / <i>Abgabe des Schlussberichts:</i> 14.08.2025, 12:00
17.02.2025	Expositions et Pitch / <i>Ausstellungen und Pitch der Bachelorarbeiten:</i> 22.08.2025 – HEI 25.08.2025 – Monthey 28.08.2025 – Visp
Présentation du projet individuel (Pr4) <i>Präsentation:des individuellen Projekts (Pr4)</i>	Défense orale / <i>Mündliche Verfechtung:</i> Semaines/Wochen 36-37 (01-12.09.2025)
02.05.2025	
Démarrage du travail de bachelor <i>Start der Bachelorarbeit:</i>	
19.05.2025	

Signature ou visa / <i>Unterschrift oder Visum</i>	
Responsable de l'orientation / <i>Leiter/in der Vertiefungsrichtung:</i> 	¹ Etudiant·e / Student/in: 

¹ Par sa signature, l'étudiant·e s'engage à respecter strictement la directive DI.1.2.02.07 « Travail de bachelor ». Durch seine Unterschrift verpflichtet sich er/die Student/in, sich an die Richtlinie DI.1.2.02.07 „Bachelorarbeit“ zu halten.

Développement durable

Traduction de l'anglais par chatgpt [3] :

L'Agenda 2030 pour le développement durable, adopté par l'ensemble des États membres des Nations Unies en 2015, constitue une feuille de route commune pour la paix et la prospérité des peuples et de la planète, aujourd'hui et pour l'avenir. Au cœur de cet agenda se trouvent les 17 Objectifs de Développement Durable (ODD) [4], un appel urgent à l'action lancé à tous les pays – développés comme en développement – dans le cadre d'un partenariat mondial. Ces objectifs reconnaissent que l'éradication de la pauvreté et d'autres privations doit aller de pair avec des stratégies visant à améliorer la santé et l'éducation, à réduire les inégalités et à stimuler la croissance économique – tout en luttant contre les changements climatiques et en œuvrant à la préservation des océans et des forêts.



Fig. 1. – 17 Objectifs de Développement Durable (ODD) [4]

Le projet consiste en la création d'un [Hardware Abstraction Layer \(HAL\)](#) de développement d'automate. Il est donc très probable que le projet soit utiliser dans des applications allant dans le sens d'un développement durable. En effet, l'automatisation des processus industriels peut contribuer à la durabilité en améliorant l'efficacité énergétique (ODD 7), en réduisant les déchets et en optimisant l'utilisation des ressources. De plus, l'automatisation peut également aider à surveiller et à contrôler les émissions de gaz à effet de serre, ce qui contribue à la lutte contre le changement climatique. L'automatisation favorise également la croissance économique (ODD 8) en augmentant la productivité et en réduisant les coûts de production. En outre, l'automatisation peut également améliorer la sécurité au travail (ODD 3) en réduisant le risque d'accidents liés à des tâches dangereuses.

Cependant, le programme peut également être utilisé pour des applications moins durables. Par exemple, il peut être utilisé pour automatiser des processus industriels qui ont un impact environnemental lourd (ODD 13), comme l'extraction de combustibles fossiles ou la production de déchets toxiques (ODD 14-15).

1 | Introduction

Ce document présente les travaux réalisés ainsi que les perspectives à venir pour le projet **PLCSofT** destiné à **WAGO**.

Ce projet consiste en l'ajout de nouvelles fonctionnalités à un **HAL** (Hardware Abstraction Layer) pour le développement d'automates via une interface web. La programmation se fait de manière graphique et modulaire sur une page web, en reliant les différentes fonctions disponibles entre elles. Le but étant de permettre une programmation simple et intuitive. L'objectif principal du projet est l'ajout de nouveaux blocs fonctionnels pouvant être utilisés dans ce cadre de développement et plus spécifiquement l'ajout de blocs de communication réseau. En effet, la communication réseau est essentielle pour le développement de programmes automates modernes et le développement dans des systèmes **Internet of Things (IoT)**, comme nous l'explique l'article [5].

« Scientists claim that the potential benefit derived from this technology will sprout a foreseeable future where the smart objects sense, think and act. Internet of Things is the trending technology and embodies various concepts such as fog computing, edge computing, communication protocols, electronic devices, sensors, geo-location etc. »

Il y a également l'article [6] qui explique l'importance de l'**IoT** dans de nombreux domaines comme la santé, l'agriculture, l'industrie 4.0, la domotique, etc. Il est donc important pour **WAGO** de permettre le développement de programmes automates pour ces systèmes.

1.1 Point de départ

Le projet se construit sur la base de deux programmes déjà développés, lors du TB 2024 :

- **Softplcui-main** : Gérant l'interface web côté PC.
- **Softplc-main** : Gérant l'activation des entrées / sorties selon le programme build depuis PLC UI.

Le travail effectué précédemment nous prouve la faisabilité du développement d'un tel **HAL**. Les fonctionnalités implémentées par ce précédent projet sont :

- Digital Input / output
- Analogue Input / output
- Ton (timer retardé à la montée)

1.2 Objectif

L'objectif est l'amélioration et l'implémentation de nouvelles fonctionnalités. Les tâches devant être réalisées sont les suivantes :

- Modifier les programmes actuels pour utiliser la nouvelle interface REST WDA pour piloter les nouveaux automates.
- L'implémentation de nouveaux blocs de haut niveau comme CAN, MQTT, Web-Server, client/serveur HTTP et autres bloc. Il faudra trouver une solution pour faire ces tâches par programmation en bloc.
- Amélioration et extensions du frontend web.

- Développement d'un banc de test physique et d'une application de démonstration pour une maison connectée.
- Documentation, tests et rédaction du rapport, poster et présentation.

1.3 Application de démonstration pour maison connectée

L'application de démonstration d'une maison connectée Fig. 2 a pour but de démontrer les capacités du HAL qui sera développé dans le cadre de ce projet. Il s'agit de réaliser un programme automate permettant de contrôler et de surveiller les différents équipements d'une maison connectée, tels que les lumières, les prises électriques, les capteurs de température, etc. Une page web doit permettre de visualiser l'état des équipements et de les contrôler.

La fonctionnalité de l'application de démonstration qui a été choisie et qui sera développée est la suivante : Depuis une interface web, il sera possible de contrôler une lampe en réglant son intensité et sa couleur. Depuis cette même interface, il sera également possible de régler une consigne de température. Un capteur de température enverra la température et une prise électrique s'activera si la température est trop basse.

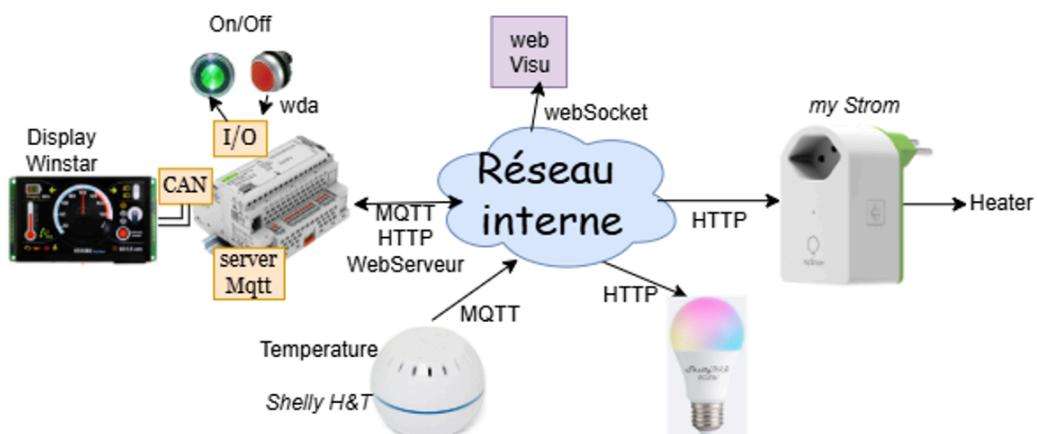


Fig. 2. – Application de démonstration pour **maison connectée**

2 | WDx

Documentation API : <https://192.168.37.134/openapi/wda.openapi.html>

Documentation : <https://downloadcenter.wago.com/wago/null/details/m2ddbfyuihrn44rp2h>

Json Parameter : [https://192.168.37.134/wda/parameter-definitions?page\[limit\]=20000](https://192.168.37.134/wda/parameter-definitions?page[limit]=20000)

2.1 Accéder IO

2.1.1 Sans wda

Lien pour accéder sur page web : 192.168.37.134:8888/api/v1/hal/io

Résultat affiché sur la page web :

```
{« di »:[false,false,false,false,false,false],« do »:[false,false,false,false],« ai »:[0.336,0.343],« ao »:[0,0],« temp »:[16778.26508951407,16778.26508951407]}
```

Activer output :

Dans le fichier « OutputUpdate.go » de softplc-main. Pour active DO1 : PUT <http://192.168.37.134:8888/api/v1/hal/do/0>

```

47     req, err := http.NewRequest(http.MethodPut, url, data)    err: nil    req: Put http://192.168.37.134:8888/api/v1/hal/do/0
48     if err != nil {
49         fmt.Println(err)
50     }
51     req.Header.Set(key: "Content-Type", value: "application/json")
52
53     client := &http.Client{} client: *http.Client | 0xc000202230
54     resp, err := client.Do(req)  resp: *http.Response | 0xc000490120
55     if err != nil {
56

```

Fig. 3. – programme : OutputUpdate.go

C'est à partir de la ligne 54 qu'on a la lampe allumé. Plus de détaille dans dossier « autre » puis dossier « Request ».

```

GET /api/v1/hal/do/0 HTTP/1.1
Host: 192.168.37.134:8888
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Vary: Origin
Date: Fri, 11 Apr 2025 09:46:07 GMT
Content-Length: 162

{"di": [true, false, false, false, false, false], "do": [false, false, false, false], "ai": [0.338, 0.324], "ao": [0, 0], "temp": [16778.26508951407, 16778.26508951407]}

PUT /api/v1/hal/do/0 HTTP/1.1
Host: 192.168.37.134:8888
User-Agent: Go-http-client/1.1
Content-Length: 4
Content-Type: application/json
Accept-Encoding: gzip

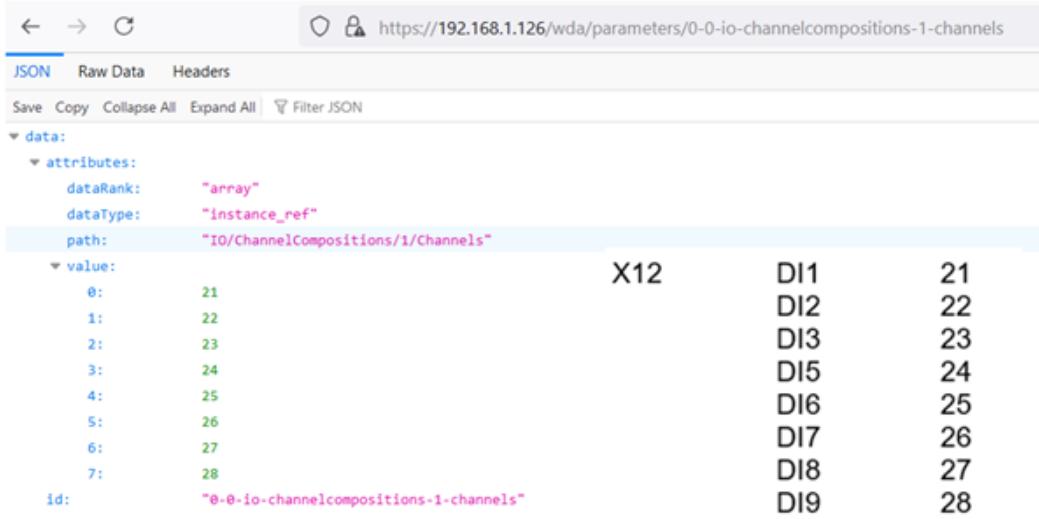
trueHTTP/1.1 204 No Content
Vary: Origin
Date: Fri, 11 Apr 2025 09:46:07 GMT

```

Fig. 4. – wireShark Http stream

2.1.2 Avec wda

751-9402 : <https://192.168.37.134/wda/parameters/0-0-io-channelcompositions-1-channels>



The screenshot shows a JSON editor interface with the URL <https://192.168.1.126/wda/parameters/0-0-io-channelcompositions-1-channels>. The JSON structure is as follows:

```
JSON Raw Data Headers  
Save Copy Collapse All Expand All Filter JSON  
▼ data:  
  ▼ attributes:  
    dataRank: "array"  
    dataType: "instance_ref"  
    path: "IO/ChannelCompositions/1/Channels"  
  ▼ value:  
    X12      DI1      21  
    0: 21      DI2      22  
    1: 22      DI3      23  
    2: 23      DI5      24  
    3: 24      DI6      25  
    4: 25      DI7      26  
    5: 26      DI8      27  
    6: 27      DI9      28  
    7: 28  
  id: "0-0-io-channelcompositions-1-channels"
```

Fig. 5. – automate 751-9402 accéder aux IO

C'est possible avec un automate 751-9402 [7], cependant on a le modèle 751-9401 [8]. Cela ne signifie pas que c'est impossible, mais que ce n'est pas documenté.

3 | Etat de l'art

Durant le cours projet 4, plusieurs ressources pouvant être utile pour la suite ont été trouvées. Cette section présente aussi le travail effectué durant le TB 2024.

3.1 n8n [1]

C'est un logiciel d'automation présent en ligne sur GitHub. Il permet une programmation en no-code sur page web comme ce qu'on l'on essaie de faire. Il est surtout conçu pour l'automation de tâche simple. Il permet notamment l'automation de chat alimenté par l'IA, c'est-à-dire des réponses automatiques. Ce n'est pas ce qui nous intéresse mais cela peut nous aider à avoir des idées.

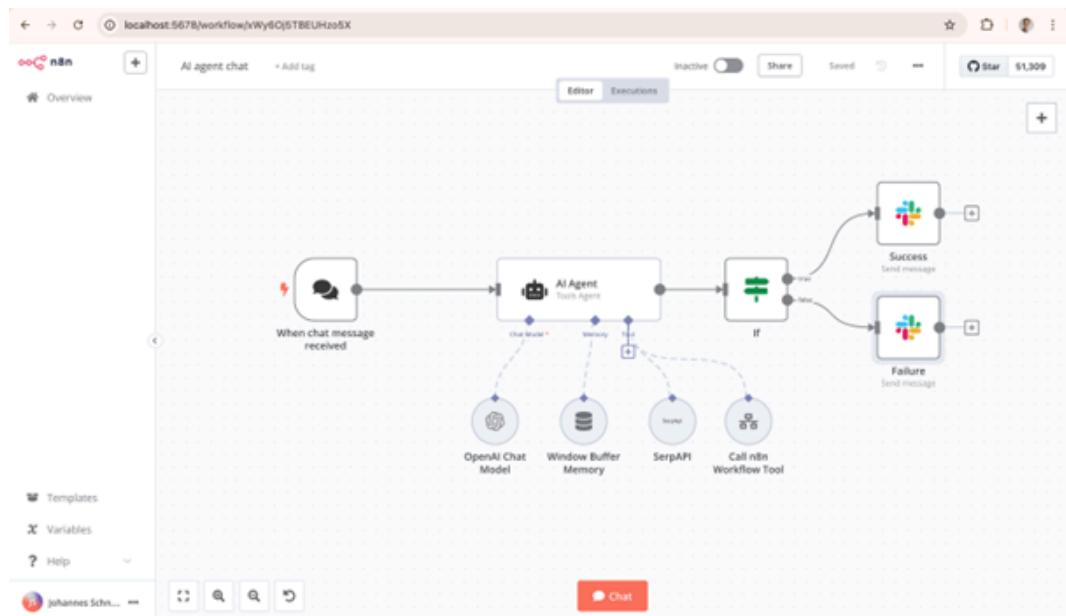


Fig. 6. – Exemple GitHub n8n

L'activation d'une output en n8n peut se faire de la manière suivante. Il suffit d'un bloc HTTP Request1 qu'on configure.

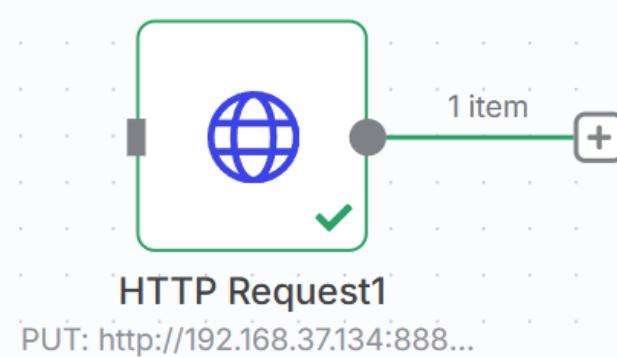


Fig. 7. – Bloc HTTP Request1 en n8n

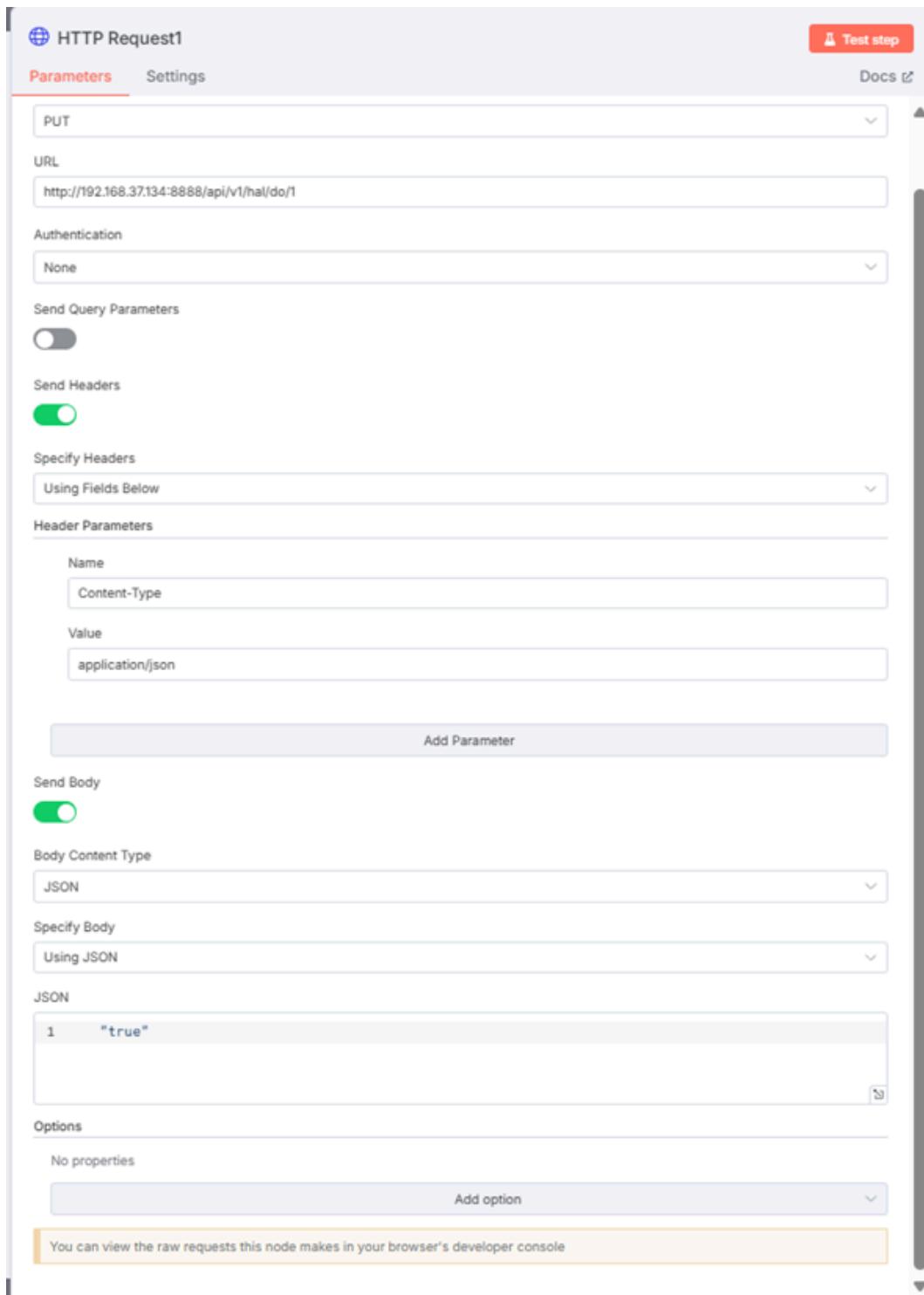


Fig. 8. – configuration HTTP Request1 en n8n pour activation output 2

Le code de n8n est disponible sur GitHub [1]. Une analyse a été faite, mais il utilise une librairie différente de celle que nous utilisons. Nous ne pouvons donc pas nous en inspirer directement. Cependant, il est possible de s'en inspirer pour la création de l'interface graphique et de la logique de programmation.

3.2 totaljs [2]

C'est un logiciel de programmation en no-code. Il est possible de faire des programmes en JS, mais il y a aussi une interface graphique.

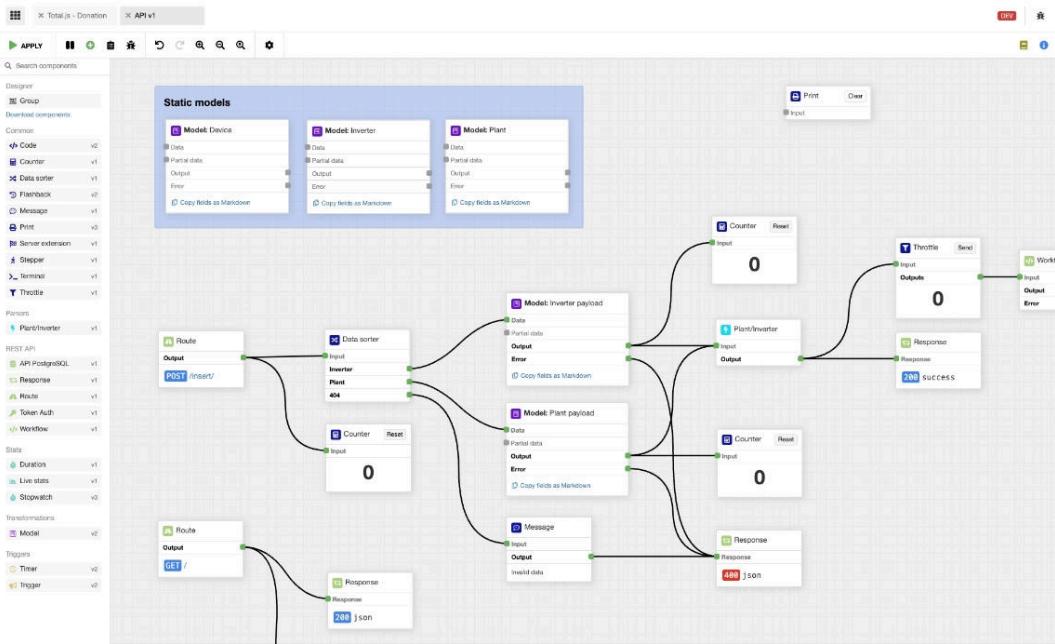


Fig. 9. – Exemple totalJS

3.3 Analyse critique du code existant

Le code existant est un bon point de départ pour la création d'un [HAL](#) de développement d'automate. Cependant, il y a plusieurs points à améliorer.

Le code est presque totalement fonctionnel malgré quelques petites erreurs. Cependant, la structure actuelle ne permet pas l'intégration de blocs plus complexes que ce qui a été fait. En effet, par exemple, nous pouvons recevoir et transmettre qu'un nombre très limité de paramètres, et la structure de ceux-ci n'est pas très flexible. Le typage des blocs est également très limité. Il utilise le type **float64** pour tous les blocs, ce qui n'est pas adapté à la transmission de données plus complexes. Il serait préférable d'utiliser un type plus flexible, comme un type any, un type générique ou au moins un type **string**. De plus, le code est très difficile à lire et à comprendre car il utilise beaucoup d'imbrications de boucles **for** et **if**. Il manque des commentaires. Il n'a également pas prévu la possibilité d'avoir **plusieurs output** pour un bloc.

Au niveau visuel, la structure doit être améliorée. En effet, on ne peut pas continuer à mettre tous les blocs logiques dans le même fichier. Il faudrait au moins créer des fichiers séparés pour chaque type de bloc logique. La taille des blocs n'est également pas adaptée aux besoins.

Le code ne permet pas de changer le type d'un bloc de manière dynamique, car c'est le programme backend qui l'envoie à l'initialisation du programme. Cela est donc impossible sans de grosses modifications.

Il n'y a pas non plus de méthode permettant de réinitialiser les nodes, ce qui est nécessaire pour pouvoir créer des blocs plus complexes.

3.3.1 Ajouter des blocs simples

Il y a également des points positifs. La structure des « Nodes » est facile à utiliser. Pour ajouter un bloc simple, il suffit de créer un fichier dans le dossier **nodes** du programme backend (**softplc-main**). On peut, par exemple, copier-coller puis renommer un fichier existant selon le type de bloc à ajouter. Il existe trois types de blocs : **LogicalNode**, **OutputNode** et **InputNode**.



renommer un fichier : il peut parfois y avoir des problèmes si le nom du bloc commence par une lettre en début d'alphabet.

Vous pouvez copier le fichier **OrNode**, puis le modifier. Pour cela, utilisez la fonctionnalité *rechercher/remplacer* afin de remplacer "**Or**" par le nom de votre nouveau bloc. Il faut le faire deux fois :

- une fois avec "**Or**" (majuscule)
- une fois avec "**or**" (minuscule)

Les éléments suivants doivent ensuite être adaptés :

1. La fonction **ProcessLogic**, où vous écrirez la logique spécifique de votre bloc.
2. La variable de type de la structure **nodeDescription**, dont le lien avec la partie visuelle est montré dans la figure Fig. 10.

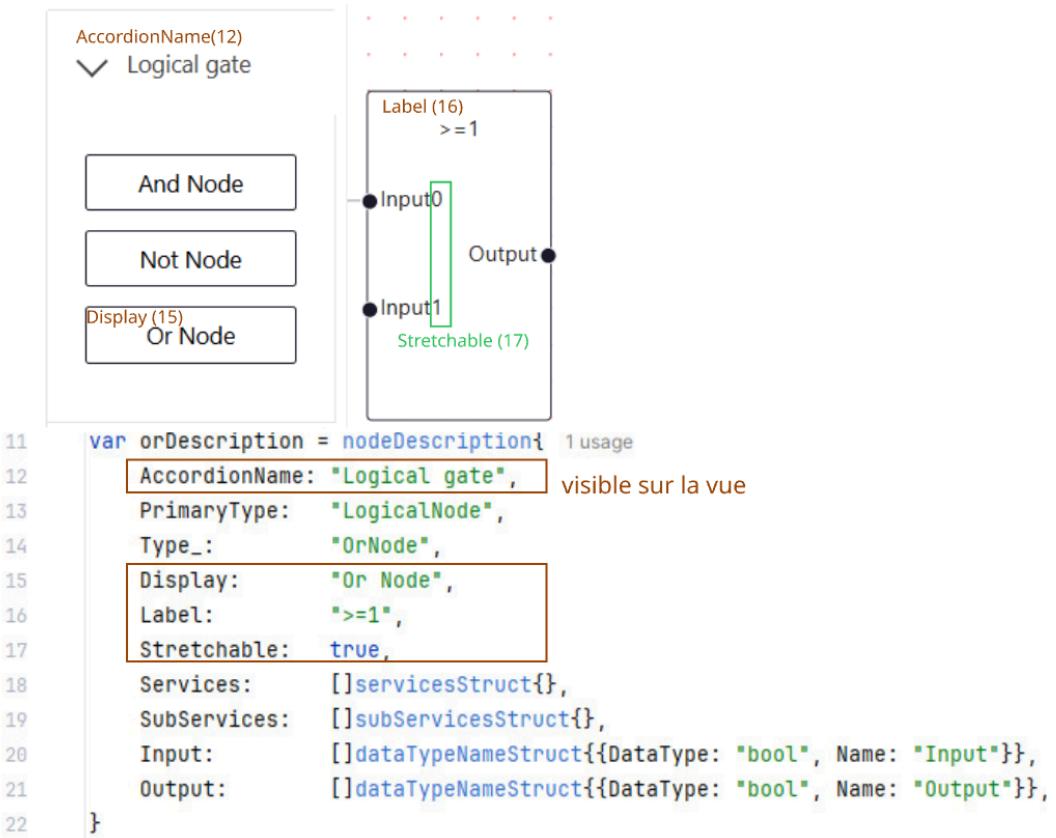


Fig. 10. – **nodeDescription** VS vue – modifier le fichier **OrNode** pour créer un nouveau bloc

3.3.2 Erreurs non gérées



Certaines erreurs n'ont pas été traitées lors de l'ancien TB. Cette section présente ces erreurs qui devront être réglées.

3.3.2.1 Manque lien

Le problème est que le programme plcSoft plante au lieu d'afficher simplement une erreur et de ne pas Build le programme dans l'automate.

Cependant, le save est possible et le restore peut être fait après avoir relancé le programme.

Remarque Bloc bleu : il y a des bloc bleu qui est le résultat d'un test explication dans la synthèse.

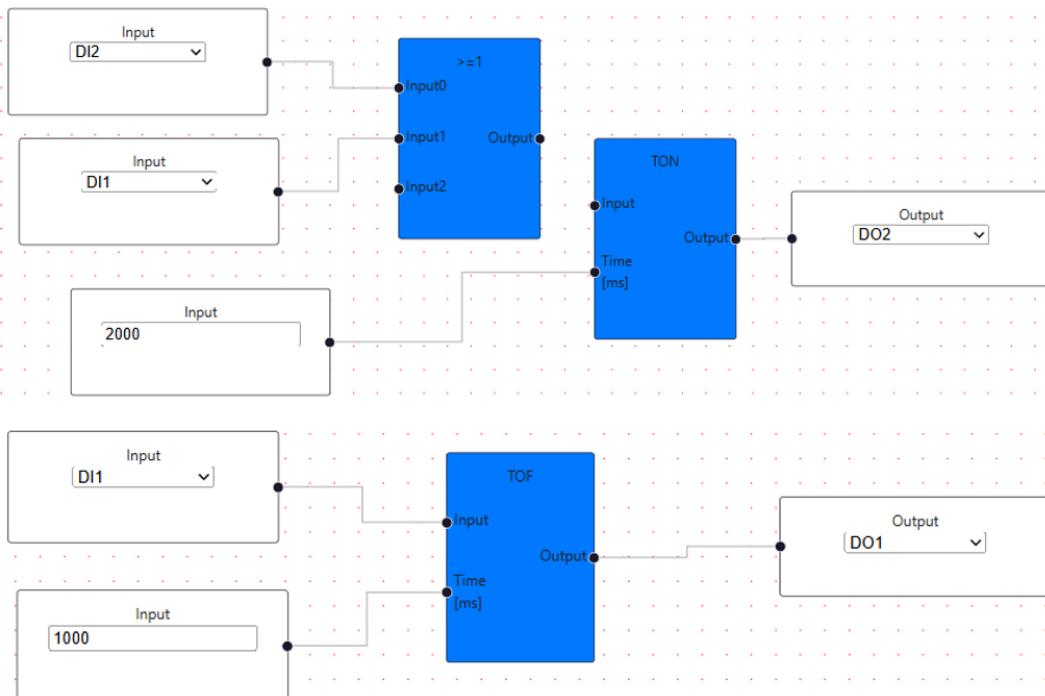


Fig. 11. – Défaut manque lien entre OR et TON

```
OutputNodes:
&{3 digitalOutput [{ D01 {0xc00026f050 Input bool}}]}
&{6 digitalOutput [{ D02 {0xc00026f140 Input bool}}]}
LogicalNode:
&{0 TOFNode [{0xc000228018 Input bool} {0xc00046ae78 Time [ms] value}] [{0 Output bool}] {<nil> false} false 0 false}
&{7 TONNode [{<nil> Input bool} {0xc00041cbbb Time [ms] value}] [{0 Output bool}] {<nil> false} false 0 false}
InputNodes:
&{2 digitalInput [{ DI1 {0xc000228018 Output bool}}]}
&{4 constantInput [{ {0xc00046ae78 Output value}}]}
&{8 constantInput [{ {0xc00041cbbb Output value}}]}
Const:
{4 1000}
{8 2000}
```

Fig. 12. – message plcsoft save défaut

```

panic: runtime error: invalid memory address or nil pointer dereference
[signal 0xc0000005 code=0x0 addr=0x0 pc=0xc1a412]

goroutine 1 [running]:
SoftPLC/nodes.(*TONNode).ProcessLogic(0xc0001813b0)
    C:/Users/puiipp/Desktop/TB_PLCSoft_Wago/Software/Go/softplc-main/softplc-main/nodes/tonNode.go:89 +0x1d2
main.main()
    C:/Users/puiipp/Desktop/TB_PLCSoft_Wago/Software/Go/softplc-main/softPLC.go:31 +0x13c

Process finished with the exit code 2

```

Fig. 13. – Erreur Build manque lien entre OR et TON

3.3.2.2 Plusieurs Output pour un Input



Cela devrait être faisable, pourtant c'est interdit.

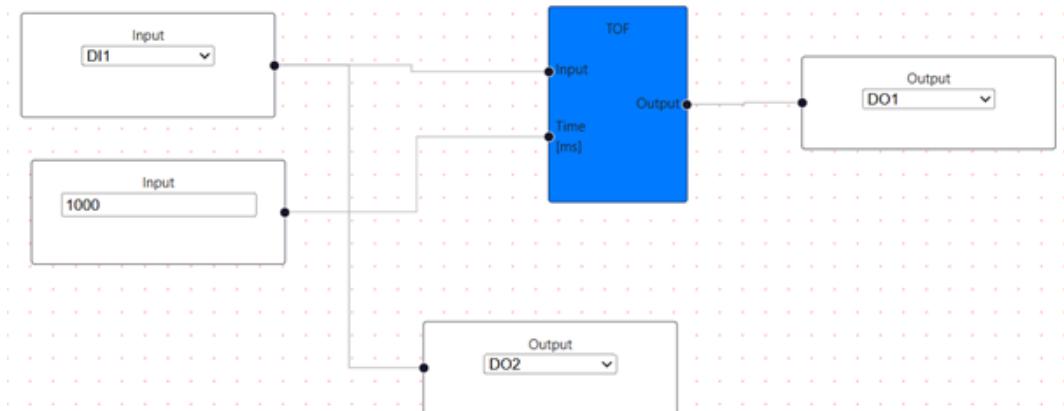


Fig. 14. – Erreur 2 output pour un input, avec une output directement sur l'input

4 | Conception

 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequre doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Contenu

4.1 Environnement de développement	17
4.2 Aperçu du système	18
4.3 Bloc de haut niveau	19
4.4 User Interface (UI) Design	20
4.4.1 Vue programmation	20
4.4.2 Vue WebSocket	20
4.5 Gestion et stockage des données	23
4.5.1 Node MQTT	23
4.5.2 Node HTTP client	24
4.5.3 Node HTTP serveur	25
4.5.4 Node webSocket output	25
4.5.5 WebSocket data	26
4.6 Conclusion	27

4.1 Environnement de développement

Appareil	Adresse IP	Utiliser dans soft
Automate WAGO CC100	192.168.37.134	softplc-main
PC	localhost	softplcui-main

La salle **23.N320** utilisé pour connecté l'automate WAGO CC100 sur le réseau.

Logiciel	Appareil	Commentaires
Goland	PC	Développement logiciel
Umlet	PC	Réalisation de schéma basé développement
Firefox	PC	Permettant meilleure visualisation de WDx
HTTPPie	PC	Permettant de tester les requêtes HTTP
miro	site	Réalisation de schéma, prise de note, réflexion, analyse
WDx	Automate	Dénomination générale pour parler de WDM + WDD + WDA : <ul style="list-style-type: none"> • WDM = WAGO Device Model (standard utilisé pour les WDD) • WDD = WAGO Device Description (manifeste décrivant ce que le produit met à disposition) • WDA = WAGO Device Access (accès aux paramètres et IO)
WDA	Automate	Nouvelle librairie, interface REST, WAGO accessible par web en JSON. Permet la récupération des informations de l'automate et le contrôle des entrées/sorties par requête HTTP en format JSON. https://192.168.37.134/wda/parameters?page[limit]=20000 https://192.168.1.126/wda/parameter-definitions?page[limit]=20000

4.2 Aperçu du système

L'aperçu du système est présenté dans le schéma de principe Fig. 15. Il montre les différentes parties du système et comment elles interagissent entre elles. Le schéma n'est pas exhaustif, mais il donne une idée générale des grands principes du système.

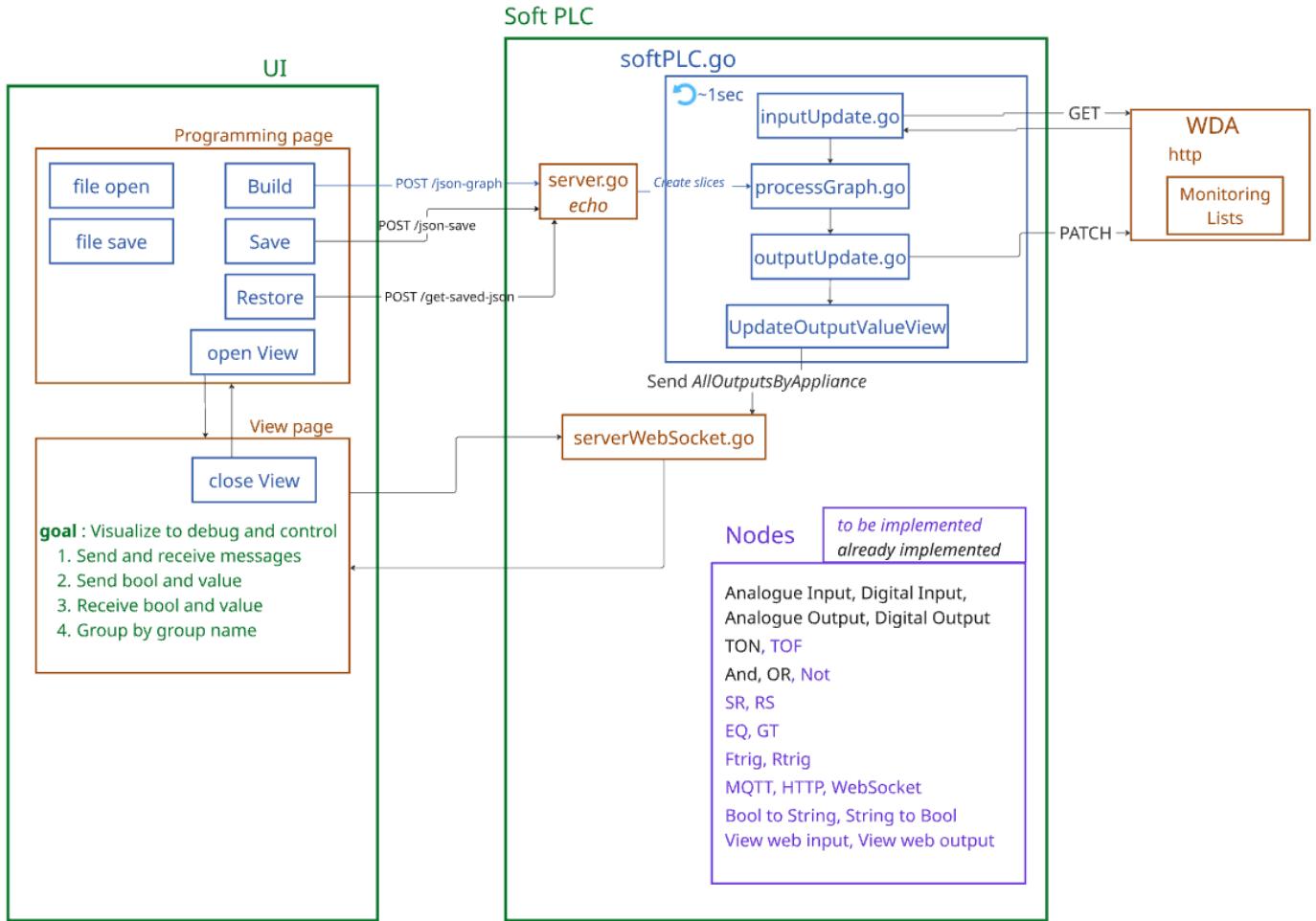


Fig. 15. – aperçu du système - schéma de principe

4.3 Bloc de haut niveau

Il existe plusieurs manière d'aborder le problème. Une des approches est de repérer les points commun entre ces blocs de haut niveau pour essayer d'en tirer une forme générique. On remarque que tous ces blocs ont pour objectif de transmettre et recevoir des données. Il faudra donc commencer par le développement de bloc commun pour une communication. Il faut également des blocs permettant de travailler avec des STRING. Le schéma figure 1 montre le concept d'une telle structure avec tous les blocs qui devront être développé autour pour pouvoir créer une communication.

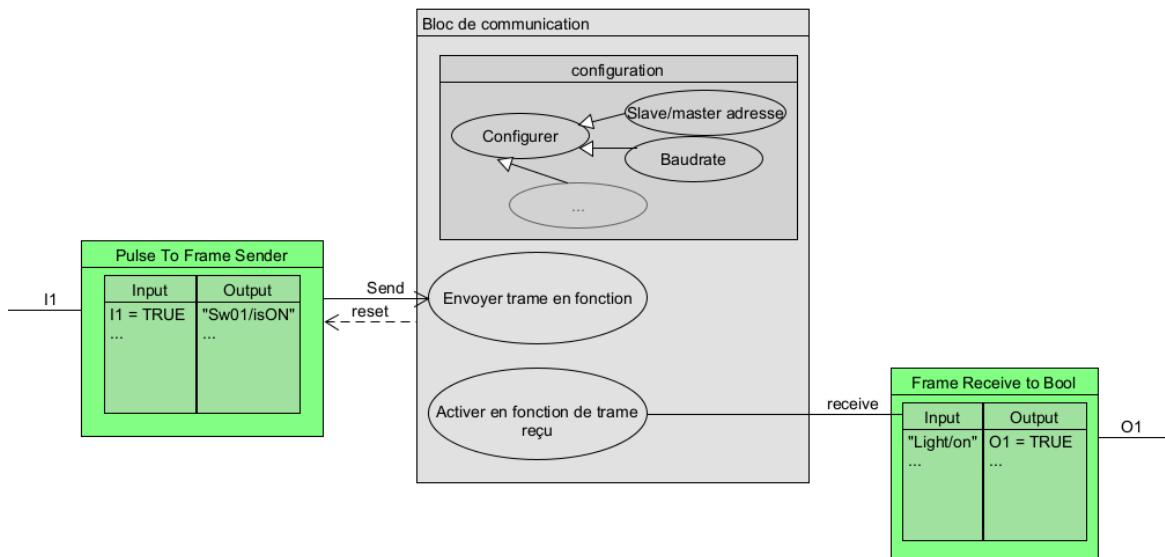


Fig. 16. – Communication principe

L'idée étant d'avoir un bloc communication qui s'occupe de la configuration étant différente pour MQTT, HTTP, CAN, etc. Sur lequel, on pourra double cliquer pour accéder à la page de configuration. Sur ce bloc de communication, on pourrait ensuite venir lier nos 2 blocs permettant la transition de boolean vers nos trame. Par le future, en mode debug, l'utilisateur pourra voir l'état de la communication grâce au « bloc de communication » et voir ce que la logique combinatoire transmet comme trame grâce au bloc en vert.



4.4 User Interface (UI) Design

4.4.1 Vue programmation

La vue programmation permet de créer des programmes PLC en utilisant une interface graphique. Elle est représentée par la « Programming Page » sur le schéma Fig. 15. Cette vue permet de créer par **drag and drop** des blocs logiques, des **Inputs** et des **Outputs**, et de les connecter entre eux pour créer un programme PLC.

4.4.2 Vue WebSocket

La vue WebSocket permet de visualiser en temps réel l'état des Inputs et Outputs spécifiques à cette vue. Sur le schéma Fig. 15, elle est représentée par la « View Page ».

Cette vue s'accompagne de blocs logiques qui permettent sa création. Ces blocs sont représentés dans la figure Fig. 17. Ils permettent de créer des Inputs et Outputs spécifiques à cette vue, permettant ainsi de visualiser l'état de n'importe quelle connexion dans le programme PLC. Un exemple de ce à quoi pourrait ressembler la vue WebSocket est présenté dans la figure Fig. 20, et le programme qui crée cette vue est présenté dans la figure Fig. 19. Les parties *recevoir* et *envoyer* les messages sont prévues pour du débogage. Les autres parties sont prévues pour contrôler et tester le programme PLC. Remarquez l'impact du champ *Appliance Name*, qui permet de créer des groupes dans la vue WebSocket. Cela facilite le regroupement des Inputs et Outputs par groupe, ce qui est très utile pour la visualisation. L'idée a été inspirée de « Remote Controller » vue en cours de Data Engineering, où une page WebSocket est générée à partir d'un fichier JSON, dont voici le lien https://cyberlearn.hes-so.ch/pluginfile.php/3312976/mod_resource/content/2/index.html.

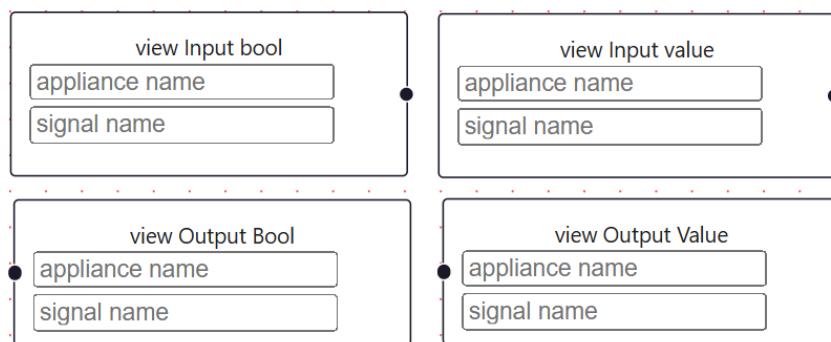


Fig. 17. – blocs vue WebSocket - Inputs et Outputs

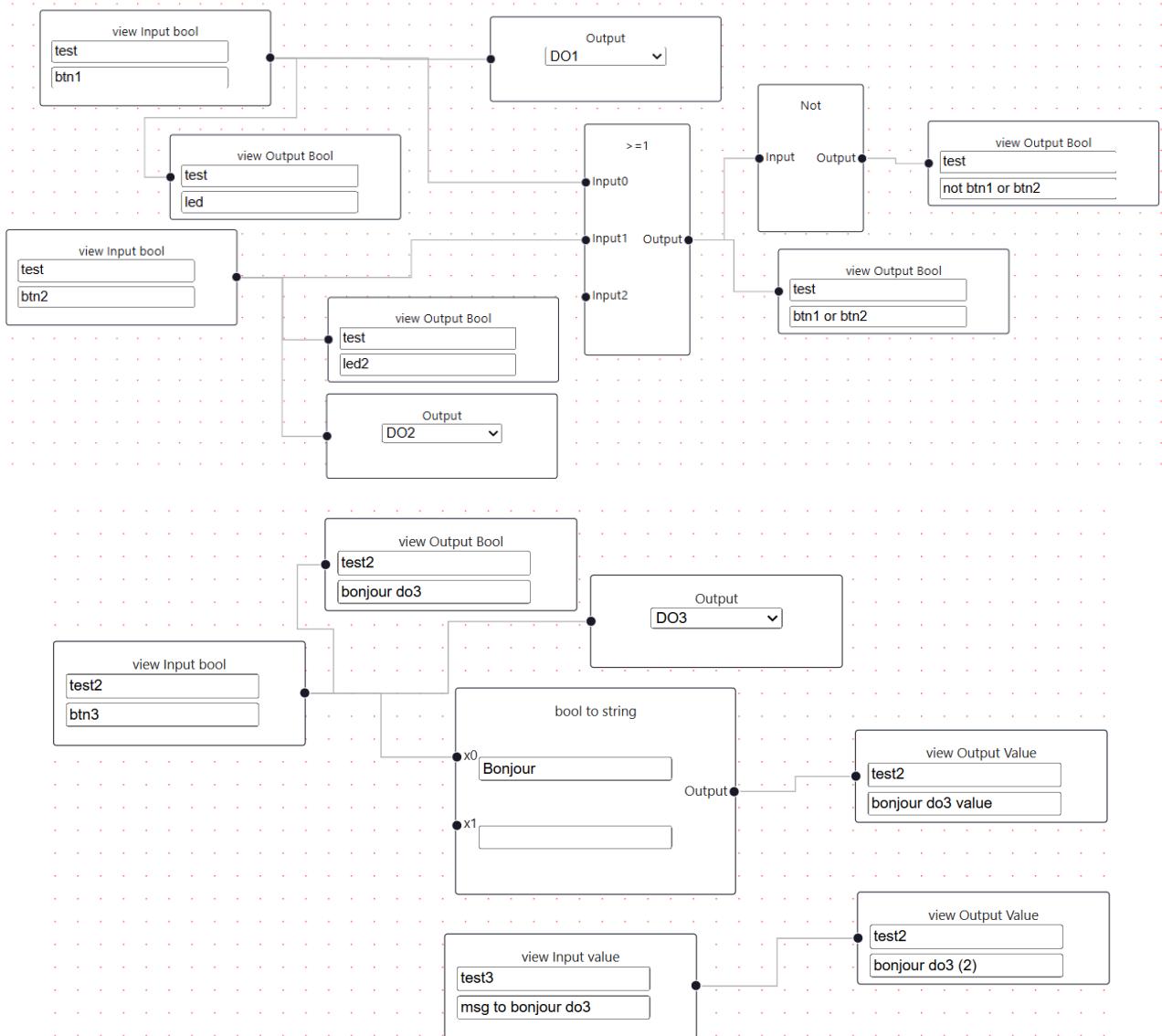


Fig. 19. – exemple programme WebSocket

[Close view](#)

WebSocket Interface

Enter message Send Envoyer les messages ici

Devices :

test

btn1 btn2 Envoyer les messages sur une input "Bool"

States :

- led : ● OFF
- led2 : ● OFF Visualiser les états des Outputs "bool"
- btn1 or btn2 : ● OFF
- not btn1 or btn2 : ● ON

test2

btn3 Envoyer les messages sur une input "Bool"

States :

- bonjour do3 : ● OFF
- bonjour do3 value : empty Visualiser les états des Inputs "value"
- bonjour do3 (2) :

test3

msg to bonjour do3: Send Envoyer les messages sur une input "value"

Messages :

Recevoir les messages ici

Fig. 20. – exemple vue WebSocket (commentaires en brun)

4.5 Gestion et stockage des données

La gestion et le stockage des données sont des aspects importantes du système. Dans notre cas, les données sont stockées et transmises en format JSON.

4.5.1 Node MQTT

Aucun des paramètres n'est obligatoire, par défaut le port est **1883** et le serveur tourne sur l'automate. Les « Settings » rentrés dans la vue sont données par **parameterValueData**.

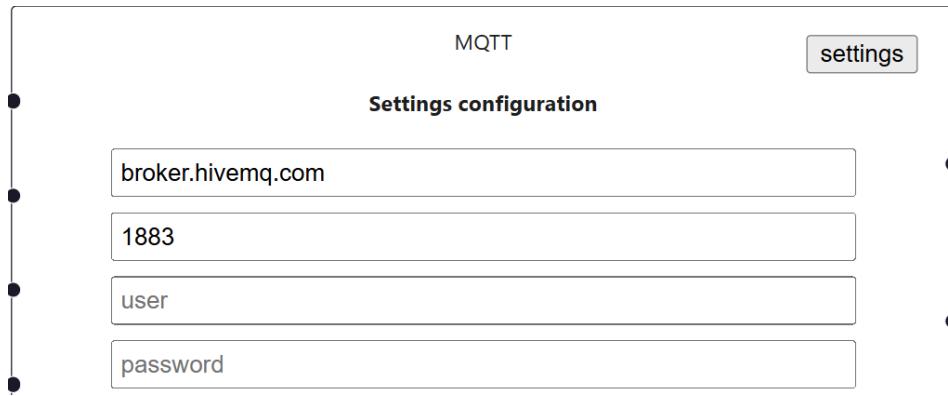


Fig. 21. – exemple de paramétrage de « Mqtt »

```

"inputHandle": [
    {
        "dataType": "bool",
        "name": "xEnable"
    },
    {
        "dataType": "value",
        "name": "topicToSend"
    },
    {
        "dataType": "value",
        "name": "msgToSend"
    },
    {
        "dataType": "value",
        "name": "topicToReceive"
    }
],
"label": "MQTT",
"outputHandle": [
    {
        "dataType": "bool",
        "name": "xDone"
    },
    {
        "dataType": "value",
        "name": "msg"
    }
],
"parameterNameData": [
    "broker",
    "port",
    "user",
    "password"
],
"parameterValueData": [
    "broker.hivemq.com",
    "1883",
    "",
    ""
]
]

```

Liste 1. – mqtt, extrait de la structure JSON d'un exemple

4.5.2 Node HTTP client

Le package Go [9] a été trouvé. Pour le Node HTTP client, il est possible de configurer les paramètres suivants :

- **URL** : l'URL de la requête HTTP.
- **user** : l'utilisateur pour l'authentification HTTP.
- **password** : le mot de passe pour l'authentification HTTP.
- **Headers** : les en-têtes HTTP à envoyer avec la requête.



les *Headers* sont des paires clé-valeur, par exemple : `{"Content-Type": "application/json"}`. Il faut donc deux paramètres pour chaque Header. De plus, il faut que ce soit possible de mettre plusieurs Headers.

Le bloc peut prendre dynamiquement les paramètres suivants :

- **xSend** : un booléen pour envoyer lorsque qu'il passe à *true*.
- **url path** : la suite du chemin de l'URL de la requête HTTP. Il est ajouté à la suite du paramètre *URL* pour donner l'URL final.
- **Method** : la méthode HTTP à utiliser (GET, POST, PATCH, PUT, DELETE, HEAD, OPTIONS), par défaut GET.
- **Body** : le corps de la requête HTTP, qui peut être au format JSON ou autre.

Le bloc nous retournera les paramètres suivants :

- **xDone** : un booléen pour indiquer si la requête a été effectuée avec succès.
- **Response** : la réponse de la requête HTTP.

4.5.3 Node HTTP serveur

Le package Go [9] a été trouvé. L'exemple [10] permet d'en comprendre d'avantage sur la création d'un serveur HTTP en Go. Pour le déploiement d'un serveur HTTP sur Docker, il a été trouvé la documentation [11].

Le Node HTTP serveur permet de créer un serveur HTTP qui écoute les requêtes entrantes. Le but étant que l'on puisse recevoir une requête venant de n'importe où, par exemple une *appliance* HTTP qui veut activer une sortie automate. Il doit être permettre de créer une resource (POST), de modifier une resource (PUT, PATCH), de lire une resource (GET) et de supprimer une resource (DELETE).

4.5.4 Node webSocket output

Pour cette fois, nous n'utilisons pas « parameterValueData » car les outputs ont un fonctionnement différent dans le programme qui rend cette implémentation plus complexe. De plus, il n'est pas nécessaire car, généralement, les outputs n'ont pas de paramètres. Il est donc possible de se passer de cette fonctionnalité en utilisant « selectedServiceData » et « selectedSubServiceData ».

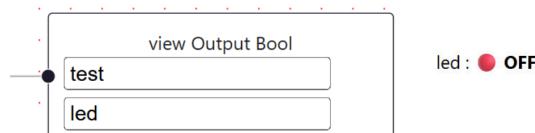


Fig. 22. – exemple de paramétrage de « view Output Bool »

```

"label": "view Output Bool",
"outputHandle": [],
"parameterNameData": null,
"primaryType": "outputNode",
"selectedFriendlyNameData": "default",
"selectedServiceData": "test",
"selectedSubServiceData": "led",

```

Liste 2. – view Output Bool, extrait de la structure JSON d'un exemple

4.5.5 WebSocket data

Le **WebSocket** est un protocole de communication bidirectionnelle qui permet d'envoyer et de recevoir des données. Il est utilisé pour la vue **WebSocket** décrite dans le Chapitre 4.4.2.

Dans les figures Fig. 23 et Fig. 24, on peut voir le principe de transmission des données. Les schémas permettent de visualiser toutes les structures de données nécessaires, ainsi que les fonctions (en brun) et les variables (en violet) les plus utiles.

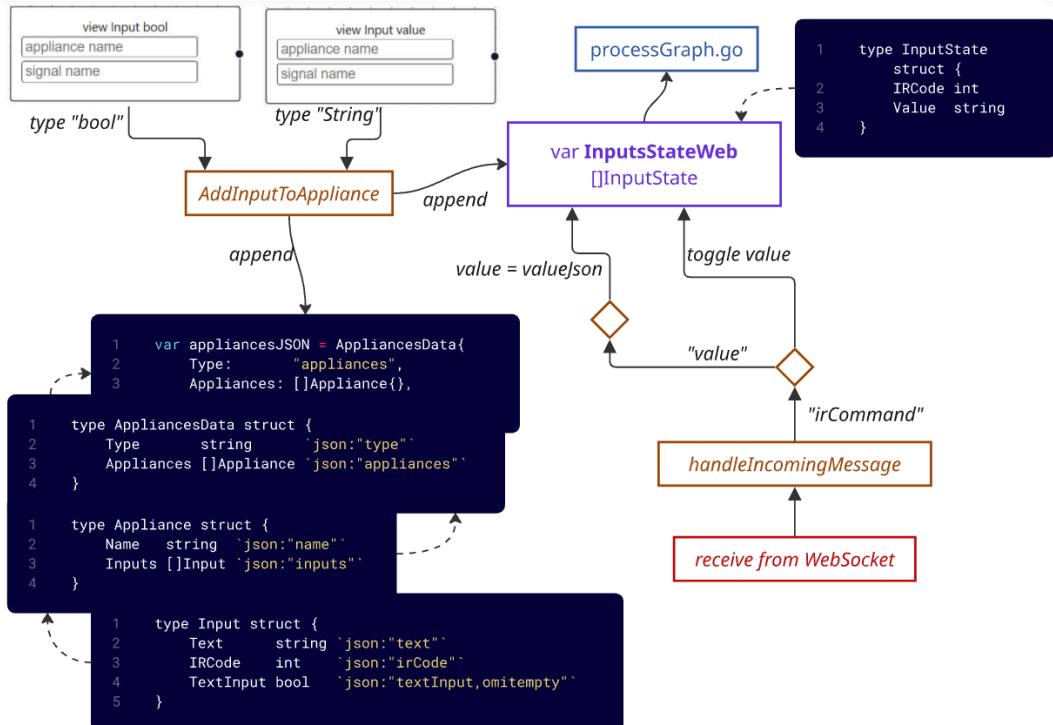


Fig. 23. – principe de transmission des données via WebSocket pour les inputs

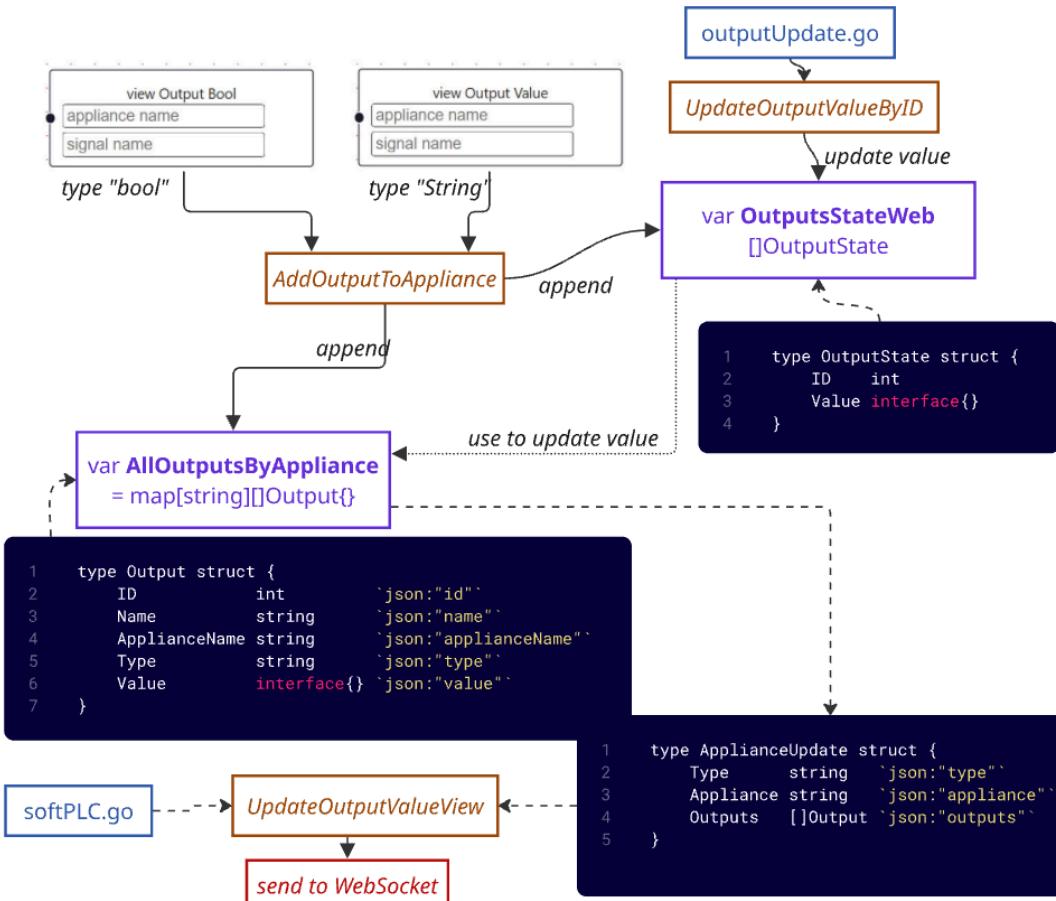


Fig. 24. – principe de transmission des données via WebSocket pour les outputs

4.6 Conclusion

5 | Implémentation

Cette section décrit les différentes étapes qui ont été implémentées durant le projet. Elle est divisée en plusieurs sous-sections, chacune abordant un aspect spécifique. Il y est notamment décrit comment les blocs logiques complexes ont été implémentés, comment WDA a été utilisé, l'ajout d'une vue webSocket ainsi que tout ce qui a dû être mis en place pour que tout cela soit réalisable.

Contenu

5.1	WDA	29
5.1.1	inputUpdate.go	29
5.1.2	outputUpdate.go	30
5.2	Interface webSocket	30
5.3	Intégration des blocs logiques simples	30
5.3.1	Bloc Bool to String	30
5.3.2	Bloc String to Bool	31
5.4	Intégration des blocs logiques complexes	34
5.4.1	MQTT	34
5.4.2	WebSocket	34
5.4.3	HTTP	34
5.5	Conclusion	34

5.1 WDA

L'intégration de WDA est une partie importante du projet. Cette section décrit comment WDA a été utilisé pour communiquer avec l'automate et comment il a été intégré dans le programme.

Pour implémenter WDA, il a fallu modifier le programme backend (**softplc-main**) et plus particulièrement les fichiers **inputUpdate.go** et **outputUpdate.go**.

Un des principaux problèmes de WDA est la vitesse, qui est très lente. Après une commande GET sur une I/O, cela peut prendre jusqu'à environ 500ms avant qu'on ait la réponse. Sachant qu'on a besoin de faire une requête GET pour chaque Input et Output, cela représente au total 22 requêtes. La première version du programme faisait une requête GET pour chaque Input et Output, ce qui prenait plusieurs secondes pour récupérer l'état de tous les Inputs et Outputs. Pour résoudre ce problème, il a été trouvé la solution d'utiliser une **Monitoring Lists** (Chapitre 8.2). Cela nous permet de récupérer les I/O en une seule requête GET pour récupérer l'état de tous les Inputs et Outputs en une seule fois. Cela permet de réduire le temps de récupération à environ 500ms.

Toutes les requêtes via WDA doivent avoir le header Fig. 25 et l'authentification Fig. 26.

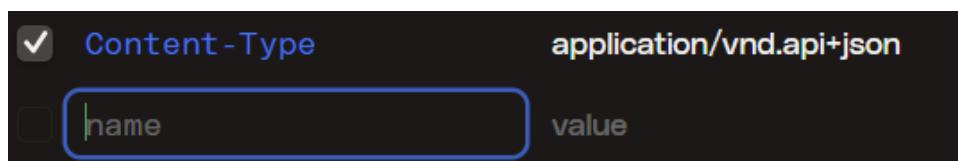


Fig. 25. – Header WDA



Fig. 26. – Authentification WDA

5.1.1 inputUpdate.go

Dans **inputUpdate.go**, l'idée étant de créer et mettre à jour la variable tableau **InputsOutputsState** de structure *InputsOutputs* dont la structure d'un élément est montré Fig. 27, afin de garder le fonctionnement du programme existant, mais cette fois-ci en utilisant WDA.

```
    3 = {inputUpdate.InputsOutputs}
      f FriendlyName = {string} ""
      f Value = {string} "0"
      f Service = {string} "DI1"
      f SubService = {string} ""
      f id = {string} ""
```

Fig. 27. – Exemple d'élément Input/Output State dans le programme backend

Au démarrage, **softPLC.go** appelle la fonction **InitInputs** pour créer le client HTTP, appeler la fonction **initClient** et initialiser la variable **InputsOutputsState**, qui est créée par la fonction **mapResultsToInputsOutputs** prenant en paramètre le résultat de la fonction **fetchValues** (Fig. 28), responsable de faire la requête GET afin de récupérer les valeurs des I/O.

La fonction **initClient** a pour rôle de **configurer les DIO** (Chapitre 8.4.1). Par défaut, ils sont configurés en **input** (value = 1 à 8). Il faut donc les configurer en **output** (value = 9 à 16) si on veut avoir que des outputs. De plus, avant de pouvoir lire et écrire des I/O, il faut modifier le **access mode** (Chapitre 8.3) en activant le control mode (value = 2). Par défaut, celui-ci est désactivé (value = 0).

```

    < v  oo results = {map[string]interface{}}
      < v  0 = 0-0-io-channels-22-divalue -> false
        10 key = {string} "0-0-io-channels-22-divalue"
        10 value = {interface{}} | bool| false
      >  1 = 0-0-io-channels-24-divalue -> false
      >  2 = 0-0-io-channels-10-dovalue -> false

```

Fig. 28. – structure valeur retornnée par la fonction **fetchValues**

Ensuite, la fonction **UpdateInputs** est appelée à chaque cycle par **softPLC.go**. La fonction **UpdateInputs** utilise la fonction **updateInputsOutputsState** pour mettre à jour la variable **InputsOutputsState**. La fonction **updateInputsOutputsState** met à jour la variable **InputsOutputsState** avec les nouvelles valeurs récupérées par la fonction **fetchValues** pour récupérer les valeurs des I/O.

En parallèle, la fonction **CreateMonitoringLists** est appelée périodiquement toutes les 600 secondes par **softPLC.go** pour recréer une monitoring lists (Chapitre 8.2).

5.1.2 **outputUpdate.go**

Dans **outputUpdate.go**, l'idée est de mettre à jour les **outputs** après que le programme ait été exécuté. La logique pour déclencher l'envoi des nouvelles valeurs des **outputs** est restée la même que pour l'ancien programme. La différence est que l'on utilise **WDA** pour envoyer les nouvelles valeurs des **outputs**. C'est-à-dire que l'on crée des requêtes **PATCH** pour les AO et DO, exemple de requête **PATCH** (Chapitre 8.4.2).

5.2 Interface webSocket

5.3 Intégration des blocs logiques simples

L'intégration des blocs logiques simples est une étape importante du projet. Cette section décrit comment ces blocs logiques simples ont été intégrés dans le programme.

5.3.1 Bloc **Bool to String**

Il correspond au bloc **Pulse to frame Sender** du schéma Fig. 16. Mais il est utilisable pour d'autres fonctionnalités que la communication. Le bloc **Bool to String** permet de transmettre en sortie une chaîne de caractères selon si un booléen est à *true* en face. C'est un bloc de type **stachable**, c'est-à-dire que le nombre d'entrées est dynamique. La figure Fig. 29 montre un exemple de ce à quoi pourrait ressembler le bloc **Bool to String**. Il est

possible d'écrire plusieurs lignes de texte, chaque ligne de texte se trouve en face d'une entrée booléenne. Si l'entrée booléenne est à *true*, la ligne de texte correspondante sera transmise en sortie.

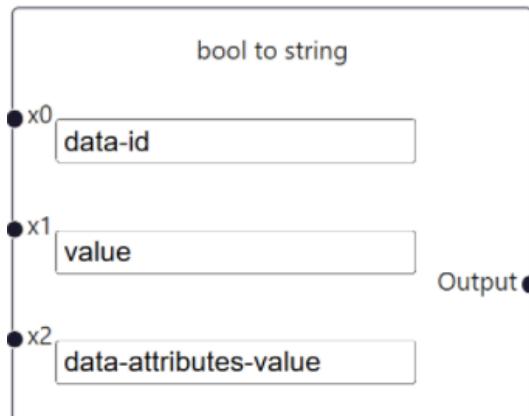


Fig. 29. – bloc Bool to String

i Il est possible d'**activer plusieurs lignes de texte** en même temps. Par exemple, si les entrées booléennes **x0**, **x1** et **x2** sont à *true*, les lignes de texte correspondantes seront transmises en sortie sous la forme d'un tableau. Cependant, la sortie est de type **String**, donc les lignes de texte seront séparées par « „, « (deux virgules entourées par des espaces).

5.3.2 Bloc String to Bool

Le bloc **String to Bool** permet de recevoir une chaîne de caractères et de la convertir en booléens. Il est représenté par le bloc **Frame to Pulse Receiver** sur le schéma Fig. 16. Il est également un bloc de type **stachable** mais au niveau des sorties. La figure Fig. 30 montre un exemple de ce à quoi pourrait ressembler le bloc **String to Bool**. Il est possible d'écrire plusieurs lignes de texte, chaque ligne de texte se trouve en face d'une sortie booléenne. Si la chaîne de caractères reçue contient la ligne de texte correspondante, la sortie booléenne sera à *true*.

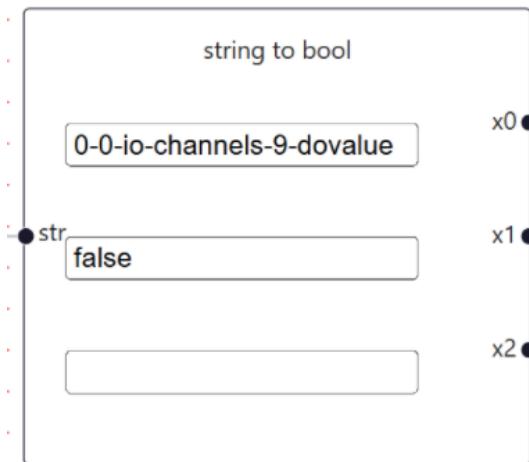


Fig. 30. – bloc String to Bool

Le fonctionnement du bloc **String to Bool** a été réfléchi de manière à ce qu'il soit le plus simple possible, mais pouvant gérer le plus de situations possible. Le schéma bloc simulant toutes les situations se trouve en Fig. 31. Des blocs se trouvent en amont, mais ce qui nous intéresse est ce qu'on retrouve en *input*. Les trois situations de fonctionnement principales sont :

1. Activer un booléen si la chaîne de caractères reçue correspond à une des lignes de texte.
2. Traiter un tableau reçu dans son ordre original. Cela est important pour les blocs de communication, car on peut donner le résultat de leurs messages dans le même ordre que l'ordre de demande de requête, ce qui permet de traiter les messages sans se soucier d'avoir plusieurs messages avec les mêmes valeurs.
3. Traiter des tableaux comme des caractères simples.

Ainsi, il doit être capable de gérer des caractères simples et des pseudos tableaux (lignes de texte séparées par « „ „). Si la chaîne de caractères reçue contient un caractère simple, la sortie booléenne correspondante sera à *true*, pour autant qu'elle existe.

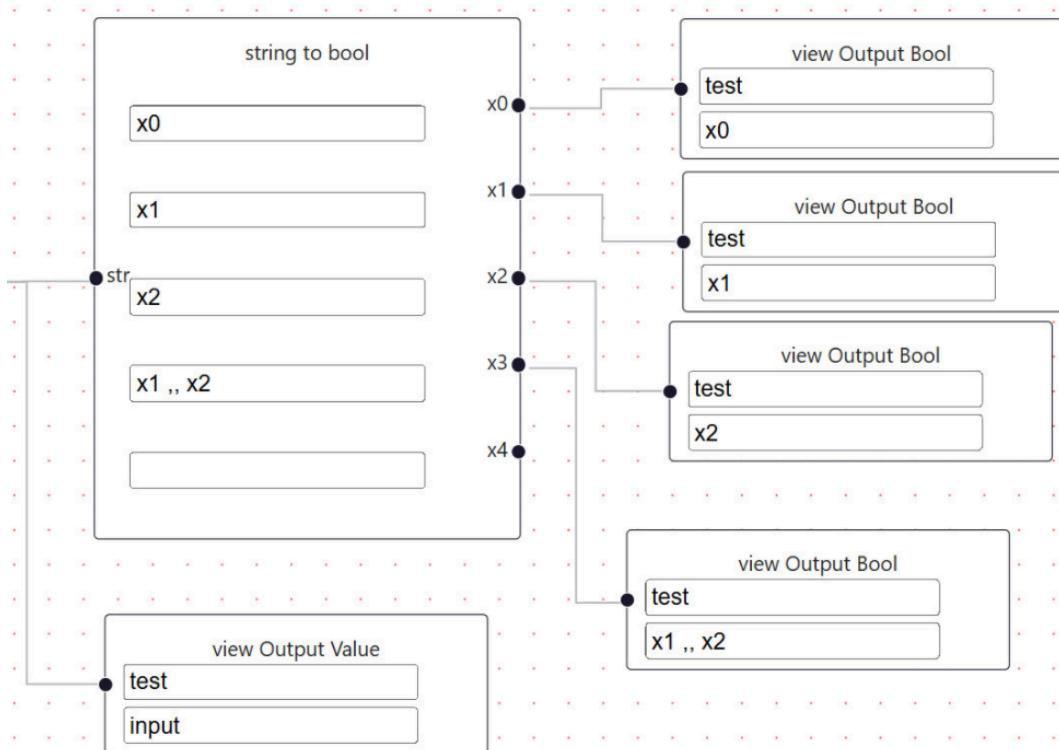


Fig. 31. – exemple - fonctionnement String to Bool

- input : **x0** input : **x1** input : **x2**
- x0 : ON x0 : OFF x0 : OFF
- x1 : OFF x1 : ON x1 : OFF
- x2 : OFF x2 : OFF x2 : ON

Fig. 32. – exemple - fonctionnement String to Bool - situation 1

input : x0 .. ABC .. x2	input : x0 .. x1 .. x2	input : x0 .. x1	• input : x0 .. x2
x0 : ON	x0 : ON	x0 : ON	• x0 : ON
x1 : OFF	x1 : ON	x1 : ON	• x1 : OFF
x2 : ON	x2 : ON	x2 : OFF	• x2 : OFF
x1 .. x2 : OFF			

Fig. 33. – exemple - fonctionnement String to Bool - situation 2

input : x1 „ x2	input : x0 „ x1 „ x2
x0 :  OFF	x0 :  ON
x1 :  OFF	x1 :  ON
x2 :  OFF	x2 :  ON
x1 „ x2 :  ON	x1 „ x2 :  OFF

Fig. 34. – exemple - fonctionnement **String to Bool** - situation 3

5.4 Intégration des blocs logiques complexes

5.4.1 MQTT

5.4.2 WebSocket

5.4.3 HTTP

5.5 Conclusion

6 | Validation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Contenu

6.1 Section 1	36
6.2 Section 2	36
6.3 Conclusion	36

6.1 Section 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequa doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

6.2 Section 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequa doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

6.3 Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequa doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

7. Conclusion

7.1. Résumé du projet

L'automate a été câblé et configuré. Il est prêt à être utilisé pour la suite du projet. Les programmes softplc-main et softplcui-main ont pu être testés et fonctionnent comme décrit dans le travail précédent de TB. Toutefois, la partie analogique n'a pas été testée, mais elle ne semble pas fonctionnelle, car elle ne figurait pas parmi les points traités dans le TB précédent.

Par ailleurs, le bloc Appliance Input ne fonctionne pas et fait planter l'interface. De nombreuses améliorations, décrites dans la partie « Chapitre 7.4.1 », restent possibles.

7.1.1. Fonctionnalités développées :

- Un nouveau bloc timer de type TOR a été ajouté. Cela prouve la faisabilité de l'ajout de blocs simples et montre que le programme est à la fois robuste et adaptable.
- Ajout de la fonctionnalité de sauvegarde/restauration via un fichier Très pratique, elle évite de devoir réécrire le code à chaque modification ou chargement du programme.
- Ajout d'une **SlideBar** [12], une fonctionnalité nécessaire pour l'ergonomie de l'interface, permet de voir les éléments plus bas.

Il a été décidé de concentrer les efforts sur l'ajout de fonctionnalités permettant de gagner du temps lors du développement et des tests.

7.1.2. changement de l'interface

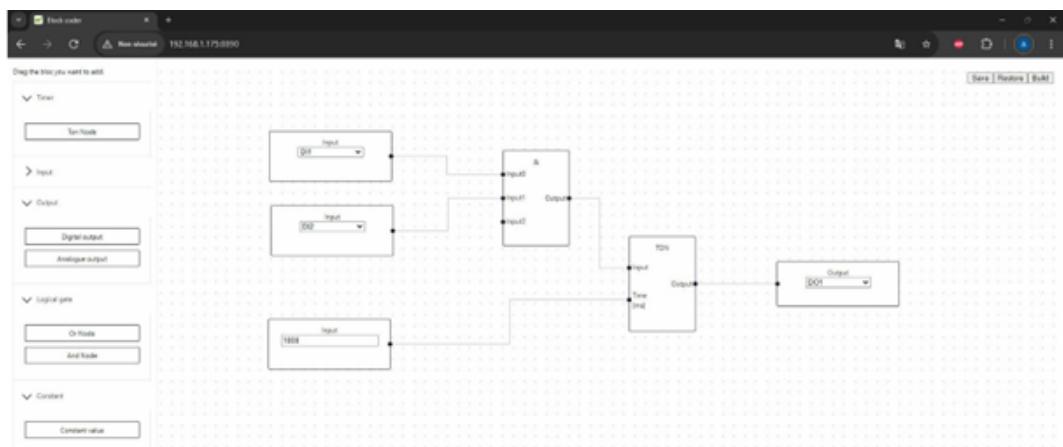


Fig. 35. – Interface avant

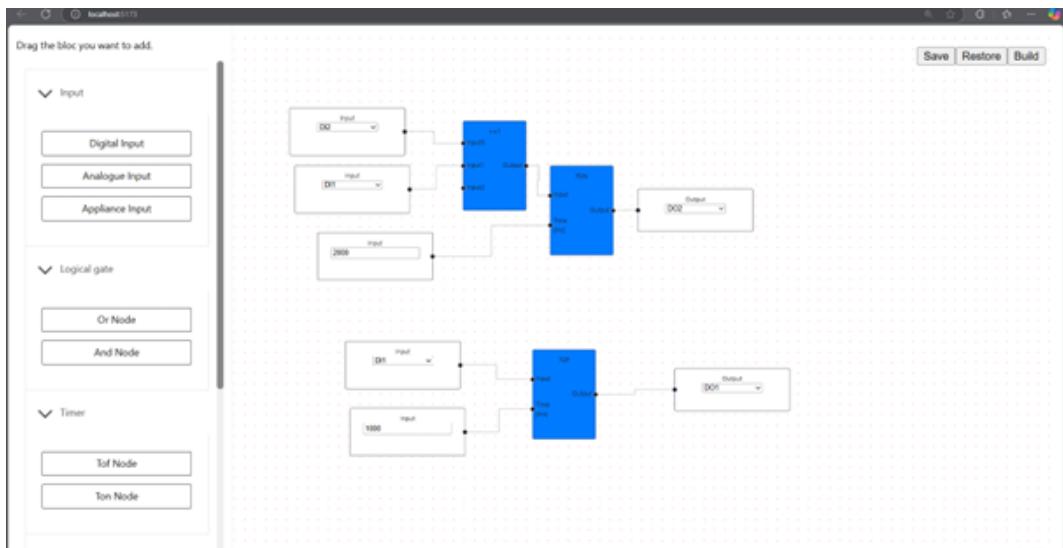


Fig. 36. – Interface après

La différence est la **slide Bar** car avant si on ouvrait tous on n'avait pas accès aux composants du bas. On voit également le nouveau bloc de type **TOR** qui a été ajouté.

Bloc bleu : il sont le résultat d'un test qui a été fait l'objectif était de voir comment était géré le style css des blocs. Le résultat est qu'il est géré par groupe. Ainsi, tous les Blocs *LogicalNode* ont le même type. Il faudra donc améliorer la structure pour rendre plus facile l'attribution de style si on veut plus personnaliser.



7.2. Comparaison avec les objectifs initial

Les objectifs fixés par pr4 sont atteints. Le programme a pu être testé et permet de créer des programmes très simples. Un nouveau bloc a été ajouté et testé sur l'automate. Le principe de fonctionnement des codes a été vus et il est possible d'ajouter de nouveaux. Cependant, le programme n'est pas parfait. Il reste des points à améliorer, notamment des erreurs .

7.3. Difficultés rencontrées

La documentation de WDA n'est pas suffisante pour comprendre le fonctionnement de la *library*. Il y a beaucoup de paramètres différents, mais on ne trouve pas ceux qui nous intéressent, la majorité d'entre eux sont pour modifier des paramètres de la configuration automatique. Cependant, il a pu être remarqué que les modèles **741-9402** et **751-9401** ne sont pas les mêmes. La documentation du 741-9402 est plus complète, et l'utilisation des entrées/sorties (I/O) y est clairement expliquée. En revanche, il n'a toujours pas été trouvé de documentation concernant l'utilisation du module CAN.

7.4. Perspectives d'avenir

7.4.1. Idées d'amélioration et extensions du frontend web

Il y a de nombreuses possibilités d'amélioration pour l'interface utilisateur.

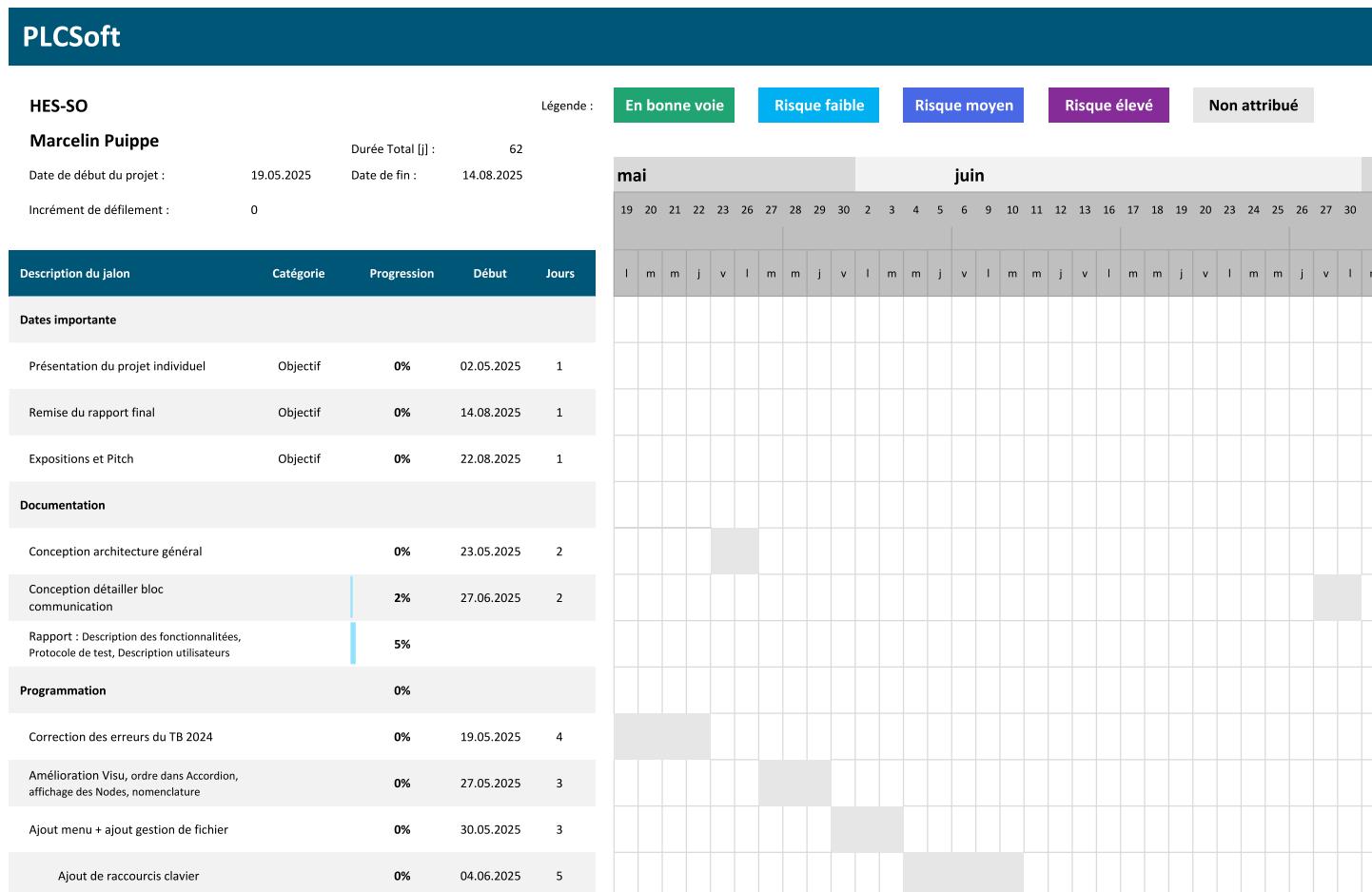
- Interdire les liens qui passent sur un bloc, ajouter une intelligence de connexion.
- Interdire les boucles de rétroaction (comme dans Codesys) ou les gérer proprement.
- Ajouter des blocs logiques contenant un champ (pour les Inputs, c'est déjà en partie fait, mais non fonctionnel, et il n'y a pas de système de seuil).
- Améliorer la nomenclature : éviter d'utiliser « Output » pour l'analogique et le digital, et « Input » pour les constantes. Une idée serait d'ajouter un menu déroulant sur le bloc pour choisir le type.
- Permettre de coder le frontend indépendamment du backend, c'est-à-dire générer les accordéons à partir d'un fichier, qui peut être mis à jour lorsqu'on est connecté.
- Ajout de raccourcis clavier :
 - Rendre la touche *Delete* fonctionnel.
 - Ctrl + C / V / A / Z / Y.
 - Clic + glisser = multi-sélection.
 - Touche O pour placer un Output, I pour un Input.
 - Touche Espace pour placer un composant identique au précédent.
 - Shift + clic gauche + glisser pour dupliquer.
 - Une autre idée intéressante : une touche (Ctrl + Alt + C) pour ajouter automatiquement tous les blocs nécessaires autour d'un bloc ou groupe sélectionné, avec des valeurs par défaut. Par exemple, on sélectionne un bloc TON, on appuie sur la touche, et le système ajoute automatiquement une constante de 1 seconde, une entrée DIO1 (ou la suivante si déjà utilisée), et une sortie DO1. Les valeurs par défaut ne sont pas obligatoires, on peut faire sans.
 - Touche dédiée pour activer ou désactiver les valeurs par défaut.
- L'ordre des Inputs, Outputs, blocs logiques, etc. dans l'accordéon n'est jamais le même, ce qui rend l'utilisation plus pénible car on ne peut pas s'habituer.
- Amélioration de l'aspect visuel : couleurs et autres éléments graphiques.
- Ajout d'une barre de menu en haut :
 - Affichage des raccourcis clavier
 - Aide
 - Choix de l'emplacement d'enregistrement par l'utilisateur
 - Ajout d'un mode Debug

8 | Annexe

Contenu

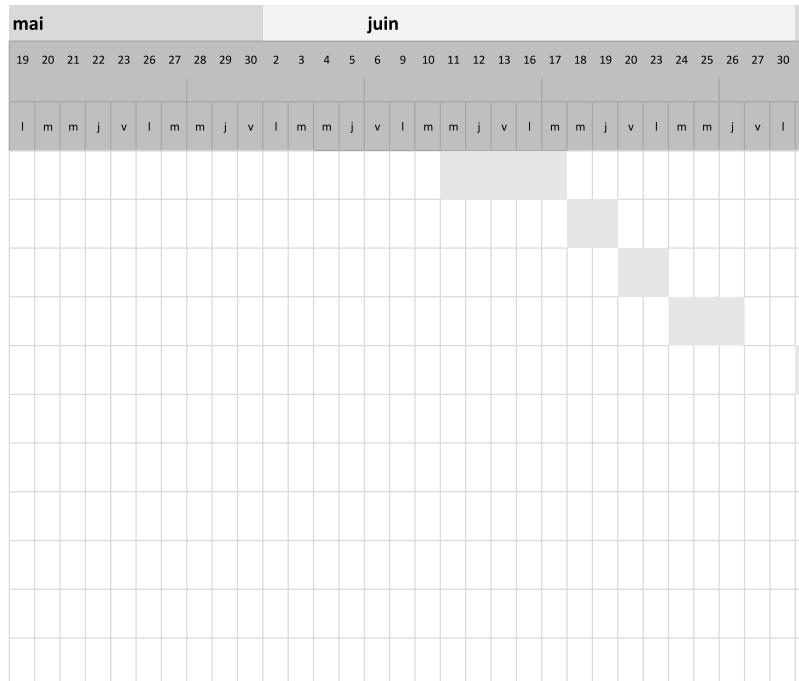
8.1 Planning	41
8.2 WDA Monitoring Lists	46
8.2.1 Création	46
8.2.2 Utilisation	47
8.3 WDA access mode	52
8.4 WDA I/O	53
8.4.1 Configuration des DIO (activation des Outputs) via WDA	53
8.4.2 Activation d'une Output via WDA	54

8.1 Planning



Date de début du projet : 19.05.2025 Date de fin : 14.08.2025
 Incrément de défilement : 0

Description du jalon	Catégorie	Progression	Début	Jours
Ajout mode Debug		0%	11.06.2025	5
Création bloc "TRIG, R_TRIG, F_TRIG"		0%	18.06.2025	2
Création bloc logiques contenant un champ + système de seuil		0%	20.06.2025	2
Création bloc "bool to frame"		0%	24.06.2025	3
Création bloc "bloc de communication MQTT"		0%	01.07.2025	10
Création bloc "bloc de communication HTTP"		0%	15.07.2025	5
Création bloc "bloc de communication WebServer"		0%	22.07.2025	5
Création bloc "bloc de communication CAN"		0%	29.07.2025	8
Debugage			08.08.2025	
Tester sur maison connectée, application de démonstration			08.08.2025	2
Reserve			12.08.2025	1



PLCSoft

HES-SO

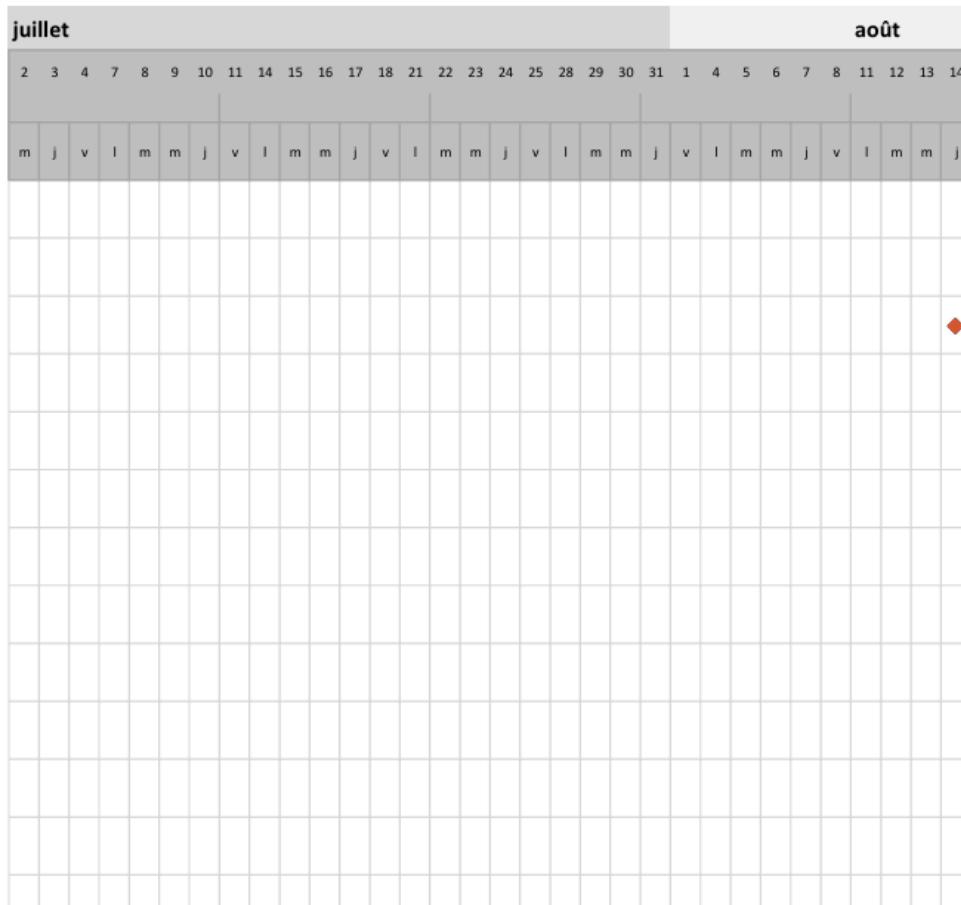
Marcelin Puippe

Durée Total [j] : 62
 Date de début du projet : 19.05.2025 Date de fin : 14.08.2025
 Incrément de défilement : 32

Légende :

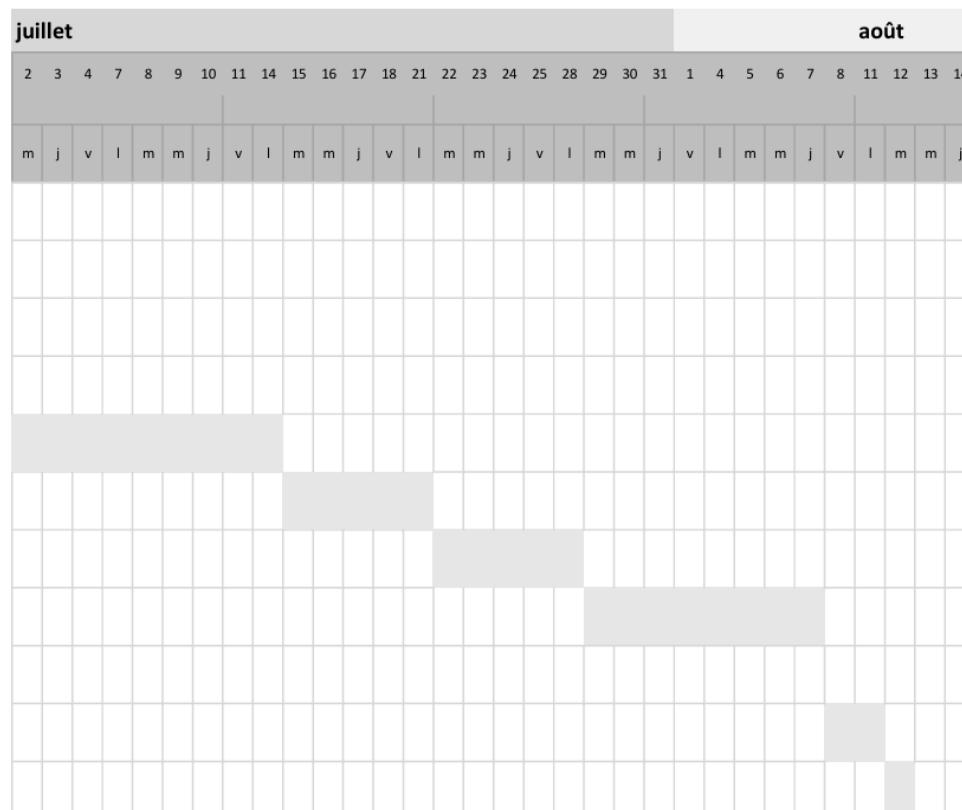
En bonne voie
Risque faible
Risque moyen
Risque élevé
Non attribué

Description du jalon	Catégorie	Progression	Début	Jours
Dates importante				
Présentation du projet individuel	Objectif	0%	02.05.2025	1
Remise du rapport final	Objectif	0%	14.08.2025	1
Expositions et Pitch	Objectif	0%	22.08.2025	1
Documentation				
Conception architecture général		0%	23.05.2025	2
Conception détailler bloc communication		2%	27.06.2025	2
Rapport : Description des fonctionnalités, Protocole de test, Description utilisateurs		5%		
Programmation				
Correction des erreurs du TB 2024		0%	19.05.2025	4
Amélioration Visu, ordre dans Accordion, affichage des Nodes, nomenclature		0%	27.05.2025	3
Ajout menu + ajout gestion de fichier		0%	30.05.2025	3
Ajout de reconnaissance visuelle		0%	04.06.2025	5



Date de début du projet : 19.05.2025 Date de fin : 14.08.2025
 Incrémentation de défilement : 32

Description du jalon	Catégorie	Progression	Début	Jours
Ajout mode Debug		0%	11.06.2025	5
Création bloc "TRIG, R_TRIG, F_TRIG"		0%	18.06.2025	2
Création bloc logiques contenant un champ + système de seuil		0%	20.06.2025	2
Création bloc "bool to frame"		0%	24.06.2025	3
Création bloc "bloc de communication MQTT"		0%	01.07.2025	10
Création bloc "bloc de communication HTTP"		0%	15.07.2025	5
Création bloc "bloc de communication WebServer"		0%	22.07.2025	5
Création bloc "bloc de communication CAN"		0%	29.07.2025	8
Debugage			08.08.2025	
Tester sur maison connectée, application de démonstration			08.08.2025	2
Reserve			12.08.2025	1



8.2 WDA Monitoring Lists

Documentation (accessible uniquement si l'automate est disponible) : <https://192.168.37.134/openapi/WDA.openapi.html#tag/Monitoring-Lists>

8.2.1 Création

i Remarquer que nous insérons un **timeout** de 600 secondes, ce qui correspond à 10 minutes. Il est possible de le modifier si besoin. A noter qu'il faudra **recréer** la Monitoring List à la fin de ce délai pour continuer à monitorer les entrées et sorties de l'automate.

De plus, il faudra mémoriser l'ID de la Monitoring List créée, car il sera nécessaire pour les requêtes GET et DELETE. Il est possible de créer plusieurs Monitoring Lists, mais il faudra alors faire attention à ne pas dépasser le nombre maximum autorisé par l'automate.

POST <https://192.168.37.134/wda/monitoring-lists>

Fig. 39. – commande post Monitoring List

```
{
  "data": {
    "type": "monitoring-lists",
    "attributes": {
      "timeout": 600
    },
    "relationships": {
      "parameters": {
        "data": [
          { "id": "0-0-io-channels-21-divalue", "type": "parameters" },
          { "id": "0-0-io-channels-22-divalue", "type": "parameters" },
          { "id": "0-0-io-channels-23-divalue", "type": "parameters" },
          { "id": "0-0-io-channels-9-dovalue", "type": "parameters" },
          { "id": "0-0-io-channels-10-dovalue", "type": "parameters" },
          { "id": "0-0-io-channels-11-dovalue", "type": "parameters" }
        ]
      }
    }
  }
}
```

Liste 3. – **body send**, exemple de création d'une Monitoring List pour les 3 premières Inputs et 3 premières Outputs de l'automate

```
{
  "data": {
    "attributes": {
      "timeout": 600
    },
    "id": "26",
    "links": {
      "self": "/WDA/monitoring-lists/26"
    },
    "relationships": {
      "parameters": {
        "links": {
          "related": "/WDA/monitoring-lists/26/parameters"
        }
      }
    },
    "type": "monitoring-lists"
  },
  "jsonapi": {
    "version": "1.0"
  },
  "meta": {
    "doc": "/openapi/WDA.openapi.html#operation/createMonitoringList",
    "version": "1.4.1"
  }
}
```

Liste 4. – **body response**, exemple de création d'une Monitoring List pour les 3 premières Inputs et 3 premières Outputs de l'automate

8.2.2 Utilisation

i Pour utiliser la Monitoring List, il faut faire une requête GET sur l'URL de la Monitoring List créée précédemment. Il est possible de récupérer l'état de toutes les entrées et sorties en une seule requête GET.

On remarque qu'il y a beaucoup d'informations qui ne nous intéressent pas dans la réponse. Il y a que « path » et « value » de « attributes » qui nous intéressent.

```
GET ↴ https://192.168.37.134/wda/monitoring-lists/26/parameters
```

Fig. 40. – commande GET Monitoring List

```
{
  "data": [
    {
      "attributes": {
        "dataRank": "scalar",
        "dataType": "boolean",
        "path": "IO/Channels/21/DIValue",
        "value": false
      },
      "id": "0-0-io-channels-21-divalue",
      "links": {
        "self": "/WDA/parameters/0-0-io-channels-21-divalue"
      },
      "relationships": {
        "definition": {
          "links": {
            "related": "/WDA/parameters/0-0-io-channels-21-divalue/
definition"
          }
        },
        "device": {
          "links": {
            "related": "/WDA/parameters/0-0-io-channels-21-divalue/device"
          }
        }
      },
      "type": "parameters"
    },
    {
      "attributes": {
        "dataRank": "scalar",
        "dataType": "boolean",
        "path": "IO/Channels/22/DIValue",
        "value": false
      },
      "id": "0-0-io-channels-22-divalue",
      "links": {
        "self": "/WDA/parameters/0-0-io-channels-22-divalue"
      },
      "relationships": {
        "definition": {
          "links": {
            "related": "/WDA/parameters/0-0-io-channels-22-divalue/
definition"
          }
        },
        "device": {
          "links": {
            "related": "/WDA/parameters/0-0-io-channels-22-divalue/device"
          }
        }
      },
      "type": "parameters"
    }
  ]
}
```

Liste 5. – **body response**, exemple d'utilisation d'une Monitoring List pour les 3 premières Inputs et 3 premières Outputs de l'automate

```
{
  "attributes": {
    "dataRank": "scalar",
    "dataType": "boolean",
    "path": "IO/Channels/23/DIValue",
    "value": false
  },
  "id": "0-0-io-channels-23-divalue",
  "links": {
    "self": "/WDA/parameters/0-0-io-channels-23-divalue"
  },
  "relationships": {
    "definition": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-23-divalue/
definition"
      }
    },
    "device": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-23-divalue/device"
      }
    }
  },
  "type": "parameters"
},
{
  "attributes": {
    "dataRank": "scalar",
    "dataType": "boolean",
    "path": "IO/Channels/9/D0Value",
    "value": false
  },
  "id": "0-0-io-channels-9-dovalue",
  "links": {
    "self": "/WDA/parameters/0-0-io-channels-9-dovalue"
  },
  "relationships": {
    "definition": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-9-dovalue/
definition"
      }
    },
    "device": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-9-dovalue/device"
      }
    }
  },
  "type": "parameters"
}
}
```

Liste 6. – **body response**, exemple d'utilisation d'une Monitoring List pour les 3 premières Inputs et 3 premières Outputs de l'automate

```
{
  "attributes": {
    "dataRank": "scalar",
    "dataType": "boolean",
    "path": "IO/Channels/10/D0Value",
    "value": false
  },
  "id": "0-0-io-channels-10-dovalue",
  "links": {
    "self": "/WDA/parameters/0-0-io-channels-10-dovalue"
  },
  "relationships": {
    "definition": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-10-dovalue/
definition"
      }
    },
    "device": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-10-dovalue/device"
      }
    }
  },
  "type": "parameters"
},
{
  "attributes": {
    "dataRank": "scalar",
    "dataType": "boolean",
    "path": "IO/Channels/11/D0Value",
    "value": false
  },
  "id": "0-0-io-channels-11-dovalue",
  "links": {
    "self": "/WDA/parameters/0-0-io-channels-11-dovalue"
  },
  "relationships": {
    "definition": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-11-dovalue/
definition"
      }
    },
    "device": {
      "links": {
        "related": "/WDA/parameters/0-0-io-channels-11-dovalue/device"
      }
    }
  },
  "type": "parameters"
}
]
```

Liste 7. – **body response**, exemple d'utilisation d'une Monitoring List pour les 3 premières Inputs et 3 premières Outputs de l'automate

```
"jsonapi": {  
    "version": "1.0"  
,  
    "links": {  
        "first": "/WDA/monitoring-lists/26/parameters?  
page[limit]=255&page[offset]=0",  
        "last": "/WDA/monitoring-lists/26/parameters?  
page[limit]=255&page[offset]=0",  
        "self": "/WDA/monitoring-lists/26/parameters?  
page[limit]=255&page[offset]=0"  
    },  
    "meta": {  
        "doc": "/openapi/WDA.openapi.html#operation/  
getMonitoringListParameters",  
        "version": "1.4.1"  
    }  
}
```

Liste 8. – **body response**, exemple d'utilisation d'une Monitoring List pour les 3 premières Inputs et 3 premières Outputs de l'automate

8.3 WDA access mode

Afin de pouvoir **écrire** ou **lire** les entrées et sorties de l'automate, il faut activer le mode d'accès WDA correspond. Le paramètre « Value » peut prendre une des 3 valeurs entières suivantes :

- 0 : no access (no read/write access)
- 1 : monitor mode (read-only mode)
- 2 : control mode (read/write mode)

```
PATCH 🔳 https://192.168.37.134/wda/parameters/0-0-io-iocheckaccessmode
```

Fig. 41. – commande PATCH pour changer le mode d'accès WDA

```
{
  "data": {
    "id": "0-0-io-iocheckaccessmode",
    "type": "parameters",
    "attributes": {
      "value": 2
    }
  }
}
```

Liste 9. – **body send**, exemple changer le mode d'accès WDA à « control mode » (read/write mode)

8.4 WDA I/O

8.4.1 Configuration des DIO (activation des Outputs) via WDA

Le bornier X5 de l'automate est composé de **8 DIO**. On peut donc choisir lesquels configurer en **Input** et lesquels configurer en **Output**. Cela peut se faire via une commande **PATCH** comme le montre Fig. 44.

Le paramètre « **value** » doit prendre un tableau de valeurs entières dont chaque index correspond à une **DIO**. Par exemple, si l'on veut configurer **DIO1** pour être une **Output**, on doit mettre 9 à l'index 0 du tableau. Si l'on veut que ce soit une **Input**, on doit mettre 1 à l'index 0 du tableau.

Les figures Fig. 42 et Fig. 43 montrent les valeurs d'activation des **Inputs** et **Outputs** pour chaque **DIO**.

X5	DI1	1	DO1	9
	DI2	2	DO2	10
	DI3	3	DO3	11
	DI4	4	DO4	12
	DI5	5	DO5	13
	DI6	6	DO6	14
	DI7	7	DO7	15
	DI8	8	DO8	16

Fig. 42. – valeurs d'activation des inputs Fig. 43. – valeurs d'activation des outputs

PATCH → <https://192.168.37.134/wda/parameters/0-0-io-channelcompositions-4-channels>

Fig. 44. – commande PATCH pour activer toutes les outputs via WDA

```
{
  "data": {
    "id": "0-0-io-channelcompositions-4-channels",
    "type": "parameters",
    "attributes": {
      "value": [
        9,
        10,
        11,
        12,
        13,
        14,
        15,
        16
      ]
    }
  }
}
```

Liste 10. – **body send**, exemple pour passer toutes les DIO en output

8.4.2 Activation d'une Output via WDA

```
PATCH → https://192.168.37.134/wda/parameters/0-0-io-channels-9-dovalue
```

Fig. 45. – commande PATCH pour activer une output via WDA

```
{
  "data": {
    "id": "0-0-io-channels-9-dovalue",
    "type": "parameters",
    "attributes": {
      "value": true
    }
  }
}
```

Liste 11. – **body send**, exemple pour passer la valeur d'une output à true (ici DIO1) dans l'automate via WDA

Request PATCH	Response 204
	▼ HTTP/1.1 204 No Content

Fig. 46. – **Response** : commande PATCH pour activer une output via WDA

Glossaire

Software

HAL – Hardware Abstraction Layer: A HAL is a layer of software that abstracts the hardware details of a computer system, allowing higher-level software to interact with the hardware without needing to know the specifics of the hardware implementation.
[4](#), [5](#), [6](#), [11](#)

Technology

IoT – Internet of Things: The Internet of Things (IoT) refers to the network of physical objects embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet.
[5](#)

termes généraux

stachable: Stachable fait référence à la capacité d'un bloc d'étendre son nombre d'entrées ou sorties en fonction des besoins. Cela permet une flexibilité dans la conception. C'est un paramètre pouvant être choisi lors de la création d'un bloc.
[30](#), [31](#)

Bibliographie

- [1] « N8n-Io/N8n ». Consulté le: 27 avril 2025. [En ligne]. Disponible sur: <https://github.com/n8n-io/n8n>
- [2] « JavaScript Libraries and Components for Web Development ». Consulté le: 27 avril 2025. [En ligne]. Disponible sur: <https://www.totaljs.com/>
- [3] « ChatGPT ». Consulté le: 28 février 2025. [En ligne]. Disponible sur: <https://chatgpt.com/>
- [4] « THE 17 GOALS | Sustainable Development ». Consulté le: 27 avril 2025. [En ligne]. Disponible sur: <https://sdgs.un.org/goals>
- [5] N. Sharma, M. Shamkuwar, et I. Singh, « The History, Present and Future with IoT », in *Internet of Things and Big Data Analytics for Smart Generation*, V. E. Balas, V. K. Solanki, R. Kumar, et M. Khari, Éd., Cham: Springer International Publishing, 2019, p. 27-51. doi: [10.1007/978-3-030-04203-5_3](https://doi.org/10.1007/978-3-030-04203-5_3).
- [6] R. Chataut, A. Phoummalayvane, et R. Akl, « Unleashing the Power of IoT: A Comprehensive Review of IoT Applications and Future Prospects in Healthcare, Agriculture, Smart Homes, Smart Cities, and Industry 4.0 », *Sensors*, vol. 23, n° 16, 2023, doi: [10.3390/s23167194](https://doi.org/10.3390/s23167194).
- [7] « WAGO Download Center - WAGO Device Model - 751-9402 ». Consulté le: 27 avril 2025. [En ligne]. Disponible sur: <https://downloadcenter.wago.com/wago/null/details/m2dd83m229zdbc5cau>
- [8] « WAGO Download Center - WAGO Device Model - 751-9401 ». Consulté le: 27 avril 2025. [En ligne]. Disponible sur: <https://downloadcenter.wago.com/wago/null/details/m2dbfyuihrn44rp2h>
- [9] « Http Package - Net/Http - Go Packages ». Consulté le: 18 juin 2025. [En ligne]. Disponible sur: <https://pkg.go.dev/net/http#ListenAndServe>
- [10] V. SOYSOUVANH, « Clients et Serveurs HTTP en Go : Guide Complet ». Consulté le: 17 juin 2025. [En ligne]. Disponible sur: <https://certiquizz.com/fr/cours/programmation/go/go-beyond-maitrisez-go-pour-l-innovation-cloud-ia-et-devops/http-client-et-serveur>
- [11] C. Nicholson, « Craignicholson/Simplehttp ». Consulté le: 18 juin 2025. [En ligne]. Disponible sur: <https://github.com/craignicholson/simplehttp>
- [12] « Accordion | React Bootstrap ». Consulté le: 27 avril 2025. [En ligne]. Disponible sur: <https://react-bootstrap.netlify.app/docs/components/accordion/>