Développement d'un prototype système SCADA multiplateforme pour la supervision d'un microréseau DC avec simulation FCS-MPC intégrée: architecture, implémentation et validation

par

Elhadji Omar YOUM

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE GÉNIE ÉLECTRIQUE
M. Sc. A.

MONTRÉAL, LE 3 JUILLET 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

Prof. Ambrish Chandra, directeur de mémoire
Département de génie électrique à École de technologie supérieure

Prof. Miloud Rezkallah, codirecteur
Département de génie électrique à École de technologie supérieure

Prof. Adrian Ilinca, président du jury
Département de génie mécanique à École de technologie supérieure

Prof. Ricardo Izquierdo, examinateur externe
Département de génie électrique à École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 15 JUIN 2026

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à remercier mes directeurs de recherche, Prof. Chandra et Prof. Rezkallah, pour leur encadrement, leurs conseils et leur soutien tout au long de ce projet de maîtrise.

Je remercie également l'École de technologie supérieure pour les ressources et les infrastructures mises à disposition.

Enfin, je remercie ma famille et mes amis pour leur soutien constant.

Développement d'un prototype système SCADA multiplateforme pour la supervision d'un microréseau DC avec simulation FCS-MPC intégrée: architecture, implémentation et validation

Elhadji Omar YOUM

RÉSUMÉ

Ce mémoire présente la conception, le développement et la validation d'un système en deux parties complémentaires : 1)une simulation FCS-MPC (Finite Control Set Model Predictive Control) pour microréseau à courant continu sous MATLAB/Simulink, utilisant les algorithmes publiés dans la thèse de Dubuisson (2023), et 2)un prototype de système SCADA multi plateforme supervisant cette simulation.

La simulation FCS-MPC réimplémente les algorithmes et l'architecture de microréseau proposés par Dubuisson (2023), celle-ci est composée de cinq convertisseurs : un inverseur de source triphasé à 8 états de commutation avec régulation de tension, un convertisseur réversible buck-boost pour la batterie, un boost photovoltaïque, un boost pour une éolienne à vitesse variable basée sur une génératrice synchrone à aimant permanent (PMSG), et un buck pour le générateur diesel, le tout sur un bus à courant continu commun régulé à 150 V.

Le prototype SCADA utilise le cadre de développement .NET MAUI (Multi-platform App UI). L'architecture intègre la communication Modbus TCP/IP standard pour l'acquisition de données. Une double stratégie d'historisation : locale avec SQLite et distante avec API REST sur Raspberry Pi est utilisée. Le système comprend des interfaces utilisateur pour la supervision (dashboard interactif, diagramme de circuit, graphiques historiques), la gestion d'alarmes avec niveaux de priorité, et l'exportation de données.

Le système a été validé selon deux configurations avec un mode de fonctionnement complémentaire : la journalisation SQLite directe avec sous-échantillonnage $\times 20$ et groupement à 1000 échantillons, et l'architecture Modbus distribuée avec un bloc Modbus Simulink, le serveur diagslave, un script d'historisation Python et REST API sur Raspberry Pi.

Mots-clés: scada, microréseau dc, fcs-mpc, commande prédictive, .net maui, modbus tcp/ip, prototype logiciel, supervision, systèmes distribués, mppt

Development of a Cross-Platform SCADA Prototype for DC Microgrid Supervision with Integrated FCS-MPC Simulation

Elhadji Omar YOUM

ABSTRACT

This thesis presents the design, development, and validation of a system consisting of two complementary parts : 1) an FCS-MPC (Finite Control Set Model Predictive Control) simulation for a DC microgrid in MATLAB/Simulink, using the algorithms published in the thesis by Dubuisson (2023), and 2) a multi-platform SCADA system prototype that monitors this simulation.

The FCS-MPC simulation reimplements the algorithms and microgrid architecture proposed by Dubuisson (2023) ; it consists of five converters : an 8-state three-phase source inverter with voltage regulation, a reversible buck-boost converter for the battery, a photovoltaic boost converter, a boost converter for a variable-speed wind turbine based on a permanent magnet synchronous generator (PMSG), and a buck converter for the diesel generator, all connected to a common DC bus regulated at 150 V.

The SCADA prototype uses the .NET MAUI (Multi-platform App UI) development framework. The architecture incorporates standard Modbus TCP/IP communication for data acquisition. A dual data logging strategy is employed : local logging using SQLite and remote logging via a REST API on a Raspberry Pi. The system includes user interfaces for monitoring (interactive dashboard, circuit diagram, historical graphs), alarm management with priority levels, and data export.

The system has been validated in two configurations with complementary operating modes : direct SQLite logging with 20x downsampling and grouping in blocks of 1,000 samples, and a distributed Modbus architecture using a Simulink Modbus block, the diagslave server, a Python logger script, and REST on a Raspberry Pi.

Keywords: scada, dc microgrid, fcs-mpc, model predictive control, .net maui, modbus tcp/ip, software prototype, monitoring, distributed systems, mppt

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AC	Alternating Current (Courant Alternatif)
API	Application Programming Interface
BESS	Battery Energy Storage System (Système de Stockage par Batterie)
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CRUD	Create, Read, Update, Delete
DC	Direct Current (Courant Continu)
DG	Diesel Generator (Générateur Diesel)
EMS	Energy Management System
FCS-MPC	Finite Control Set Model Predictive Control (Commande Prédictive à Ensemble Fini d'États)
HMI	Human-Machine Interface
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
MAUI	Multi-platform App UI
MPPT	Maximum Power Point Tracking (Poursuite du Point de Puissance Maximale)
MPC	Model Predictive Control (Commande Prédictive par Modèle)
MTBF	Mean Time Between Failures
MVVM	Model-View-ViewModel
ORM	Object-Relational Mapping
P&O	Perturbation & Observation (algorithme MPPT)
PMSG	Permanent Magnet Synchronous Generator (Génératrice Synchrone à Aimants Permanents)
PV	Photovoltaic (Photovoltaïque)

PWM	Pulse Width Modulation (Modulation de Largeur d'Impulsion)
REST	Representational State Transfer
RTU	Remote Terminal Unit
SCADA	Supervisory Control And Data Acquisition
SoC	State of Charge (État de Charge)
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
THD	Total Harmonic Distortion (Distorsion Harmonique Totale)
UI	User Interface
ViewModel	View Model (MVVM pattern)
VSI	Voltage Source Inverter (Onduleur à Source de Tension)
WT	Wind Turbine (Éolienne)
ANN	Artificial Neural Network (Réseau de Neurones Artificiels)
CSV	Comma-Separated Values
DI	Dependency Injection (Injection de Dépendances)
DSP	Digital Signal Processor
FPGA	Field-Programmable Gate Array
HiL	Hardware-in-the-Loop
RAM	Random Access Memory
RBAC	Role-Based Access Control
RMSE	Root Mean Square Error
ROI	Return On Investment
SiL	Software-in-the-Loop
SLA	Service Level Agreement
TLS	Transport Layer Security
TRL	Technology Readiness Level
WAL	Write-Ahead Logging (SQLite)

XAML	eXtensible Application Markup Language
TTFB	Time To First Bit

LISTE DES SYMBOLES ET UNITÉS DE MESURE

f_r	Fréquence de rafraîchissement [Hz]
f_{sw}	Fréquence de commutation effective [kHz]
i	Courant [A]
i_L	Courant dans l'inductance [A]
I_{MPP}	Courant au point de puissance maximale [A]
I_{ref}	Courant de référence de l'algorithme MPPT [A]
J	Fonction de coût FCS-MPC
K_e	Constante de force contre-électromotrice de la PMSG [V·s/rad]
K_v	Gain de régulation de tension du bus DC [A/V]
L_{net}	Latence réseau [ms]
M_{RAM}	Consommation mémoire [MB]
n	Nombre d'échantillons
P	Puissance [W]
$R_{reconnect}$	Taux de reconnexion [%]
S_a, S_b, S_c	États de commutation du VSI (0 ou 1)
SoC	État de charge de la batterie [%]
t	Temps [s]
T_{CPU}	Charge processeur [%]
T_s	Période d'échantillonnage du contrôleur [µs]
T_{sample}	Période d'échantillonnage SCADA [ms]
v	Tension [V]
v_C	Tension aux bornes du condensateur de filtre [V]
V_{bus}	Tension du bus DC [V]
V_{ref}	Tension de référence du bus DC [V]
ω_m	Vitesse mécanique de la turbine éolienne [rad/s]

C_f	Capacité du filtre LC de sortie [F]
G_{load}	Conductance de charge [S]
L_f	Inductance du filtre LC de sortie [H]
M_p	Dépassement [%]
R_f	Résistance du filtre [Ω]
R_{load}	Résistance de charge [Ω]
t_r	Temps de montée [s]
t_s	Temps d'établissement [s]
η	Efficacité énergétique [%]

INTRODUCTION

0.1 Mise en contexte

L'intégration croissante des sources d'énergie renouvelables dans les réseaux électriques souligne l'importance des microréseaux DC comme une solution énergétique pour la combinaison de plusieurs sources sur un même système. Faire la supervision des éléments critiques de cette infrastructure énergétique tels que les batteries nécessite un système SCADA (*Supervisory Control And Data Acquisition*) pouvant faire l'historisation et la visualisation des données dans une interface utilisateur fluide. Boyer (2010); Sharma, Chatterjee & Rakshit (2021).

Les solutions commerciales disponibles sur le marché comme Ignition, Wonderware, Siemens WinCC sont onéreuses et propriétaires dans la majorité des cas, ce qui rend leur utilisation dans le cadre de la recherche académique et des déploiements de petite échelle ardue.

L'apparition des cadres de développement multi plateformes modernes comme .NET MAUI Microsoft (2023); Hermes (2023) ouvre donc une possibilité de développer des prototypes de SCADA à moindre coût, applications qui seront déployable sur Windows, Android, iOS sur une base de code unique.

0.2 Problématique

La supervision d'un microréseau DC pose des difficultés opérationnelles spécifiques. Le microréseau combine plusieurs sources hétérogènes (batteries, panneaux photovoltaïques, convertisseurs DC/DC, charges variables) avec des états changeants, ce qui rend la surveillance manuelle inadéquate et expose les équipements à des dégradations rapides en cas de mauvaise détection.

Sans système de supervision adapté, l'opérateur doit surveiller individuellement l'état des batteries, des convertisseurs et des charges physiquement ou en consultant plusieurs interfaces hétérogènes (logiciels constructeurs), ce qui rend la détection d'anomalies tardive et l'analyse des incidents fastidieuse. L'absence d'historisation centralisée empêche la corrélation entre les événements observés et les conditions de fonctionnement.

Les difficultés spécifiques rencontrées par le personnel technique lors de la supervision d'un microréseau DC sont les suivantes :

Visibilité limitée sur l'état du microréseau : sans interface unifiée, le technicien ne dispose pas d'une vue d'ensemble en temps réel des tensions du bus DC, des courants, des états de charge des batteries et des alarmes, ce qui rend difficile la prise de décision rapide en cas de comportement anormal des sources ou des convertisseurs.

Diagnostic des états transitoires : les phénomènes rapides propres au microréseau DC comme la commutation des convertisseurs, la variation brusque de charge, le basculement entre sources ; ne peuvent pas être analysés sans historisation à fréquence suffisante pour comprendre les causes d'une défaillance ou d'un comportement inattendu.

Coût et rigidité des solutions SCADA industrielles : les plateformes commerciales (Ignition, WinCC, Wonderware) sont coûteuses, peu modulaires et orientées vers les architectures AC industrielles, alors que la supervision d'un microréseau DC nécessite des points de mesure, des seuils et des indicateurs spécifiques (état de charge, tension de bus DC, modes de fonctionnement) Patrascu, Muntean, Cornea *et al.* (2016); Jaloudi (2024).

Mobilité de la supervision : le technicien doit pouvoir consulter l'état du microréseau depuis un poste fixe, une tablette ou un téléphone lors de ses interventions à proximité des équipements.

Historisation fiable sur matériel embarqué : l'enregistrement haute fréquence des mesures du microréseau DC sur une plateforme embarquée pour éviter la perte de données.

Un système SCADA multiplateforme et adapté au microréseau DC permettrait de répondre directement à ces difficultés avec une interface unique de visualisation des grandeurs DC, un mécanisme d'alarmes automatisé sur les seuils critiques du bus DC, une historisation centralisée pour le diagnostic des transitoires, et un accès mobile aux indicateurs du microréseau, tout en s'intégrant aux contrôleurs et aux convertisseurs via Modbus.

0.3 Objectif général

L'objectif général est de faire la conception, le développement et la validation d'un prototype de système SCADA multiplateforme en utilisant les technologies open-source et les licences académiques disponibles pour la supervision d'un microréseau DC, démontrant ainsi la faisabilité technique d'une telle alternative aux solutions commerciales.

0.4 Objectifs spécifiques

Pour atteindre l'objectif général, le travail est décomposé en objectifs spécifiques qui couvrent les principaux défis techniques liés à la supervision du microréseau DC simulé du choix du cadre logiciel jusqu'à la validation des performances de l'historisation et l'estimation du compromis coût/fonctionnalités du prototype.

OS1 : Évaluer la capacité d'un framework multiplateforme moderne (.NET MAUI) à satisfaire les exigences fonctionnelles d'un système SCADA de supervision (non-contrôle critique) pour un microréseau DC.

OS2 : Définir et valider des stratégies de conception et d'optimisation d'une telle plateforme embarquée pour un microréseau spécifique.

OS3 : Caractériser le rapport coût/fonctionnalités d'un prototype SCADA basé sur des technologies open-source comparé aux solutions industrielles propriétaires.

0.5 Organisation du mémoire

Le Chapitre 1 présente la revue de littérature des systèmes SCADA industriels, protocoles de communication (Modbus, OPC-UA), architectures distribuées, frameworks multiplateformes (.NET MAUI), ainsi que la commande prédictive FCS-MPC appliquée aux microréseaux. Le Chapitre 2 décrit la méthodologie et l'architecture globale du prototype. Le Chapitre 3 décrit en détail l'implémentation de l'ensemble du prototype commençant par la simulation FCS-MPC réimplémentant les algorithmes de Dubuisson (2023) ensuite l'implémentation des services SCADA (Modbus, historisation, interfaces .NET MAUI). Le Chapitre 4 présente la validation expérimentale : tests fonctionnels de la communication Modbus, mesures de performance de l'historisation SQLite, déploiement distribué sur Raspberry Pi, et validation de l'interface. La Conclusion synthétise les contributions du prototype, évalue l'atteinte des objectifs, et propose des perspectives de recherche future, la validation Hardware-in-the-Loop, et l'apprentissage automatique.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Systèmes SCADA pour réseaux électriques

1.1.1 Définition et architecture

Le SCADA (Supervisory Control And Data Acquisition) est la technologie qui permet aux utilisateurs de collecter des données d'un ou plusieurs endroits et d'envoyer des instructions de control limitées à ces endroits, Boyer (2010). Dans le secteur énergétique, les systèmes SCADA assurent le suivi en temps réel des grandeurs électriques, l'historisation des données, la gestion des alarmes, et l'interfaçage entre opérateur et système, Sharma *et al.* (2021).

L'architecture SCADA classique comporte trois niveaux hiérarchiques :

1. **Niveau terrain :** capteurs, actionneurs, RTUs (Remote Terminal Units)
2. **Niveau communication :** réseaux industriels, protocoles de terrain
3. **Niveau supervision :** serveurs, bases de données historiques, HMI (Human-Machine Interface)

1.1.2 Solutions SCADA commerciales

Le marché est dominé par des solutions propriétaires établies :

- **Ignition par Inductive Automation**
- **Wonderware (AVEVA)**
- **Siemens WinCC**
- **GE iFIX**
- **Thingsboard**

Ces systèmes offrent une robustesse et des fonctionnalités variées, mais leur coût (licences + maintenance + formation) représente un obstacle majeur pour la recherche académique et les installations de petite échelle Pasztor (2018); Patrascu *et al.* (2016).

1.1.3 Protocoles de communication industrielle

1.1.3.1 Modbus TCP/IP

Modbus, introduit en 1979, demeure largement utilisé pour sa simplicité, Modbus Organization (2012). La variante Modbus TCP/IP englobe les trames Modbus dans des paquets TCP, permettant communication via réseaux Ethernet standard, Si, Korada, Ayyanar & Lei (2021); Rey, Gómez, Duarte *et al.* (2024). Le protocole supporte :

- Lecture/écriture de registres holding (16-bit)
- Lecture de registres input (lecture seule)
- Manipulation de coils (bits digitaux)
- Simplicité d'implémentation avec de nombreuses bibliothèques libres d'accès

Il est cependant limité par une absence de sécurité native et pas d'horodatage natif des données.

1.1.3.2 OPC UA

OPC Unified Architecture représente l'évolution moderne des standards OPC, offrant :

- Architecture orientée services (SOA)
- Modèle de données riche (objets, méthodes, événements)
- Sécurité intégrée (authentification, chiffrement, signatures)
- Plateforme-indépendant (implémentations Windows, Linux, embarqué)

Toutefois, comparé à Modbus, la complexité est accrue nécessitant donc des serveurs OPC et une configuration plus lourde.

1.1.3.3 Autres protocoles

- **DNP3 :** Principalement réseaux électriques nord-américains, horodatage précis
- **IEC 61850 :** Standard substations électriques, complexité élevée
- **MQTT :** Pub/sub léger pour IoT, de plus en plus adopté en IoT

Pour ce prototype, Modbus TCP/IP a été retenu pour sa simplicité, son support universel dans MATLAB/Simulink, et ses nombreuses bibliothèques open-source.

1.2 Cadre de développement multiplateforme

Les technologies de développement multi plateforme sont nombreuses. Parmi elles on distingue les approches basée web comme Electron, Tauri, Flutter, React Native ou encore Xamarin/MAUI qui a été utilisé dans ce mémoire. Ce choix découle d'une base déjà présente dans la programmation en C#.

Avantages pour SCADA :

- Langage unique (C# 11)
- Accès direct aux APIs système : fichiers, réseau, bases de données
- Écosystème NuGet riche (Modbus, SQLite, charting)
- Hot reload pour développement rapide

Limitations :

- Jeune maturité (bugs, documentation incomplète 2022-2024)
- Écosystème moins développé que WPF/WinForms
- Support Linux desktop expérimental (non utilisé dans ce projet)

1.2.1 Modèle MVVM (Model-View-ViewModel)

Le modèle MVVM est le modèle d'architecture recommandée pour les applications MAUI, séparant :

- **Model :** Données et logique métier (Données capteur, Événements d'alarme)
- **View :** Interface utilisateur XAML, data binding
- **ViewModel :** Logique présentation, commandes, propriétés observables

Les avantages de ce modèle sont : testabilité, réutilisabilité, séparation des differents services, support des outils Microsoft.

1.3 Gestion de données embarquées

1.3.1 Bases de données pour systèmes contraints

Pour l'historisation sur plateformes embarquées (Raspberry Pi), plusieurs options existent :

- **SQLite :** Format fichier unique, pas de serveur, intégration simple, performances acceptables, Owens (2020); SQLite.org (2024)
- **InfluxDB :** Base de données temporelles optimisée, mais empreinte mémoire importante
- **TimescaleDB :** Extension PostgreSQL, excellent pour requêtes analytiques, mais complexité accrue

Le choix s'est porté sur le SQLite car supporté par le .NET avec des fonctions et bibliothèques natives, une empreinte mémoire minimale, une simplicité de mise en œuvre et un format compatible avec plusieurs plateformes.

1.3.2 Stratégies d'optimisation SQLite

Les écritures SQLite peuvent causer des ralentissements pour historisation haute fréquence, SQLite.org (2024). Les strategies identifiées dans la littérature pour regler ce probleme :

1. **Transactions groupées :** N insertions dans une transaction au lieu de N transactions
2. **WAL mode (Write-Ahead Logging) :** Écritures en paarallèle aux lectures pour améliorer le débit
3. **Pragma optimizations :** synchronous=NORMAL, cache_size augmenté
4. **Buffering applicatif :** Accumulation en mémoire, insertion périodique
5. **Downsampling :** Réduction fréquence pour données historiques (moyenne, min, max)

1.4 Commande prédictive à ensemble fini d'états (FCS-MPC) pour microréseaux

1.4.1 Principe du FCS-MPC

Le contrôle prédictif à ensemble fini d'états (FCS-MPC, *Finite Control Set Model Predictive Control*) est une stratégie de commande discrète adaptée aux convertisseurs de puissance à commutation, Rodriguez *et al.* (2013); Kouro, Cortés, Vargas, Ammann & Rodriguez (2009). Contrairement au MPC classique à ensemble continu, le FCS-MPC exploite le caractère naturellement discret des états de commutation ($S \in \{0, 1\}$) pour énumérer tous les vecteurs de tension ou d'état possibles, sélectionner celui qui minimise une fonction de coût, et l'appliquer directement sans bloc modulateur (PWM) intermédiaire, Karamanakos & Geyer (2020); Dragicevic (2019).

Le cycle de contrôle FCS-MPC se déroule en trois étapes à chaque période T_s qui sont la mesure des grandeurs, la prédiction via le modèle discret et l'optimisation minimisant le coût.

La fonction de coût conventionnelle du FCS-MPC compare les prédictions à la référence. Cependant, Dubuisson (2023) propose de combiner le FCS-MPC avec la théorie de stabilité de

Lyapunov : au lieu de comparer les 8 prédictions à la référence, on dérive une *tension requise* V_{req} garantissant la convergence de l'erreur vers zéro, puis on compare cette tension requise aux tensions possibles du convertisseur. Cette approche garantit la stabilité du système et réduit la charge computationnelle, Dubuisson (2023).

Pour chaque convertisseur, la démarche Lyapunov–FCS-MPC suit quatre étapes :

1. Définir l'erreur de suivi : $e(k+1) = x(k+1) - x_{ref}(k+1)$
2. Proposer une fonction de Lyapunov : $LF(e(k)) = \frac{1}{2}e(k)^2$, toujours positive et nulle quand $e = 0$
3. Exiger que le taux de variation $\Delta LF = LF(k+1) - LF(k) < 0$ pour garantir la convergence
4. Résoudre $e(k+1) = 0$ pour obtenir la tension requise V_{req}

La fonction de coût modifiée par Lyapunov pour un convertisseur DC-DC s'écrit :

$$g = |V_{req} - d \cdot V_{DC}| \tag{1.1}$$

où $d \in \{0, 1\}$ est l'état de commutation candidat. Pour le VSI, la fonction de coût en repère $\alpha\beta$ est :

$$g_{VSI} = |V_{i,req,\alpha} - V_{i,\alpha}| + |V_{i,req,\beta} - V_{i,\beta}| \tag{1.2}$$

où $V_{i,req}$ est la tension requise dérivée de la stabilité de Lyapunov et $V_{i,\alpha}, V_{i,\beta}$ les tensions de sortie possibles du VSI.

La Figure 1.1 résume le cycle de calcul exécuté à chaque période d'échantillonnage T_s.

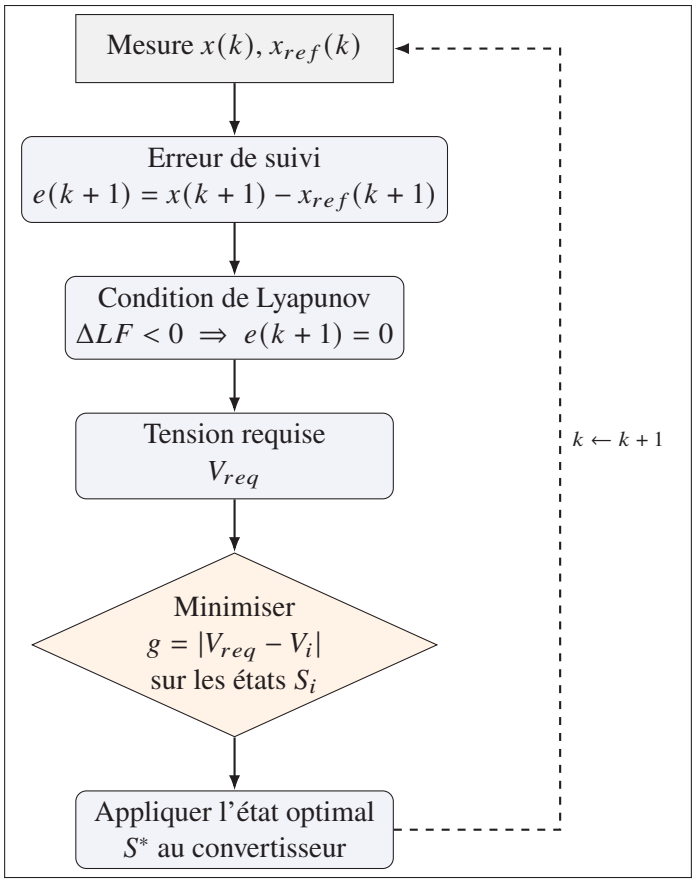

Figure 1.1 Cycle de contrôle Lyapunov–FCS-MPC

1.4.2 Application à l'onduleur inverseur de source de tension

L'onduleur inverseur de source de tension (VSI, Voltage Source Inverter) constitue l'interface principale entre le bus DC et la charge AC dans une architecture de microréseau, Chen, Abdel-Rahim, Peng & Wang (2020); Villalón, Muñoz, Muñoz & Rivera (2023). Avec un filtre LC de sortie (L_f, C_f), le modèle d'état continu du VSI est :

$$\frac{d}{dt}\begin{bmatrix} i_f \\ v_C \end{bmatrix} = \underbrace{\begin{bmatrix} -r_f/L_f & -1/L_f \\ 1/C_f & 0 \end{bmatrix}}_{A}\begin{bmatrix} i_f \\ v_C \end{bmatrix} + \underbrace{\begin{bmatrix} 1/L_f & 0 \\ 0 & -1/C_f \end{bmatrix}}_{B}\begin{bmatrix} V_i \\ I_L \end{bmatrix} \tag{1.3}$$

où i_f est le courant du filtre, v_C la tension du condensateur de sortie, r_f la résistance du filtre, L_f et C_f respectivement l'inductance et la capacité du filtre LC, V_i la tension d'entrée (sortie du VSI), I_L le courant de charge, et A, B les matrices d'état et d'entrée du modèle continu. Le modèle continu est transformé en modèle discret par exponentielle matricielle, Dubuisson (2023) :

$$\begin{bmatrix} i_f(k+1) \\ v_C(k+1) \end{bmatrix} = \Phi \begin{bmatrix} i_f(k) \\ v_C(k) \end{bmatrix} + \Gamma \begin{bmatrix} V_i(k) \\ I_L(k) \end{bmatrix} \tag{1.4}$$

$$\Phi = e^{AT_s}, \quad \Gamma = A^{-1}(\Phi - I_{2\times 2})B \tag{1.5}$$

où Φ est la matrice de transition d'état discrète obtenue par exponentielle matricielle de A sur la période d'échantillonnage T_s, Γ est la matrice d'entrée discrète associée au vecteur d'entrée $[V_i, I_L]^T$, T_s la période d'échantillonnage du contrôleur, $I_{2\times 2}$ la matrice identité d'ordre 2, et k l'indice d'échantillonnage discret.

Pour un VSI triphasé à deux niveaux, 8 vecteurs de tension discrets sont candidats ($[S_a, S_b, S_c] \in \{0, 1\}^3$), où S_a, S_b et S_c désignent les états de commutation des trois bras de l'onduleur. L'analyse de stabilité de Lyapunov permet de dériver la tension requise $V_{i,req}$ qui est ensuite comparée aux 8 tensions possibles via la fonction de coût, réduisant ainsi la charge computationnelle par rapport au FCS-MPC conventionnel, Dubuisson (2023).

1.4.3 Convertisseurs DC-DC avec FCS-MPC

Les convertisseurs DC-DC (boost, buck-boost, buck) présents dans un microréseau multi-sources s'adaptent naturellement au FCS-MPC avec un ensemble fini de deux états ($d \in \{0, 1\}$) Dubuisson (2023). En appliquant la loi de Kirchhoff et la méthode d'Euler, le modèle du

convertisseur et les prédictions de courant sont obtenues. Pour un convertisseur boost :

$$V_{in} = L\frac{dI}{dt} + d \cdot V_{DC} \tag{1.6}$$

$$I(k+1) = \frac{T_s}{L}\left(V_{in}(k) - d \cdot V_{DC}(k)\right) + I(k) \tag{1.7}$$

où V_{in} est la tension d'entrée du convertisseur, L l'inductance de stockage, I le courant dans l'inductance, $d \in \{0, 1\}$ l'état de commutation de l'interrupteur principal, V_{DC} la tension du bus DC, T_s la période d'échantillonnage et k l'indice d'échantillonnage discret.

L'erreur entre la prédiction et la référence est définie comme $I_{error}(k+1) = I(k+1) - I_{ref}(k+1)$, où I_{ref} désigne le courant de référence du convertisseur. La fonction de Lyapunov $LF = \frac{1}{2}(I_{error}(k))^2$ est utilisée pour étudier la convergence de l'erreur, $\Delta LF = LF(k+1) - LF(k)$ représentant son taux de variation discret. Avec $\Delta LF < 0$, on obtient la tension requise :

$$V_{req} = d \cdot V_{DC}(k) = \frac{L}{T_s}\left(I_{ref}(k+1) - I(k)\right) + V_{in}(k) \tag{1.8}$$

où V_{req} est la tension requise garantissant la convergence de l'erreur vers zéro pour le Lyapunov. La fonction de coût modifiée $g = |V_{req} - d \cdot V_{DC}|$ sélectionne l'état de commutation d optimal.

1.4.4 MPPT par perturbation & observation (P&O)

Pour les sources renouvelables (PV, WT), le MPPT est couplé au FCS-MPC pour fournir la référence de courant I_{ref} :

- **Éolienne (WT) :** Algorithme P&O, Esram & Chapman (2007) ; la référence de vitesse est déduite du rapport de boost $i_{ref} = I_{out,MPP} \times V_{bus}/V_{in}$
- **Panneau PV :** Modèle simplifié estimant le courant MPP en fonction de l'irradiance : $I_{MPP} = 0{,}9 \times I_{sc,nom} \times (G/1000)$

1.4.5 Régulation du bus DC par la batterie (BESS)

La batterie (BESS) joue le rôle de tampon énergétique, régulant la tension du bus DC à $V_{ref} = 150$ V via un convertisseur réversible buck-boost avec FCS-MPC, Dubuisson (2023). La référence de courant est dérivée du bilan de puissance au bus DC, en considérant les contributions de toutes les sources et la charge :

$$I_{B,ref} = \frac{V_{DC}}{V_B}\left[\frac{C_{DC}}{T_s}(V_{DC,ref} - V_{DC}) - \frac{V_{PV}}{V_{DC}}I_{PV} - \frac{V_{WT}}{V_{DC}}I_{WT} - \frac{V_R}{V_{DC}}I_{DG} + I_{L,rms}\right] \tag{1.9}$$

où V_B est la tension de la batterie, C_{DC} la capacité du bus DC, et les termes $V_x I_x / V_{DC}$ représentent les contributions de chaque source ramenées au bus DC, Dubuisson (2023).

La tension requise pour le convertisseur buck-boost, dérivée de la stabilité de Lyapunov, est :

$$V_{req,B} = d \cdot V_{DC}(k) = \frac{L_B}{T_s}\left(I_{B,ref}(k+1) - I_B(k)\right) + V_B(k) \tag{1.10}$$

et la fonction de coût associée est $g_B = |V_{req,B} - d \cdot V_{DC}|$.

1.4.6 Intégration au système SCADA

La simulation FCS-MPC constitue la source de données supervisée par le SCADA. Du côté simulation MATLAB/Simulink, dans un premier temps, une fonction exporte 29 paramètres (tensions, courants, puissances, SoC) vers une base SQLite via un mécanisme de buffering (1000 échantillons) avec sous-échantillonnage ×20 depuis la période de contrôle de $T_s = 50\,\mu$s. Ces données sont ensuite accessibles en temps réel via le client SCADA .NET MAUI. Dans un second temps les données sont transférées de MATLAB/Simulink vers un serveur modbus (Diagslave) via un bloc de communication ModbusClient ; les données sont ensuite récupérées par un script de journalisation en python qui les insère dans le serveur FastAPI sur Raspberry Pi.

1.5 Architectures SCADA pour microréseaux

La littérature sur SCADA spécifiques microréseaux reste limitée comparée aux applications industrielles traditionnelles, Zhaoxia, Zhijun, Guerrero *et al.* (2017); Beus, Herenčić, Pandžić *et al.* (2022). Plusieurs travaux récents explorent :

- Intégration OPC UA pour interopérabilité entre niveaux EMS/SCADA, Zerk, Ouassaid & Zidani (2023)
- Architectures cloud-edge pour systèmes distribués, Beus & Sirovina (2024)
- SCADA web-based pour accès distant (HTML5, WebSocket), Jaloudi (2024)
- Intégration machine learning pour détection anomalies et maintenance prédictive

Lacune observée : Peu de travaux documentent la mise en œuvre complète d'un SCADA de recherche à l'aide de frameworks multiplateformes modernes (.NET MAUI, Flutter) sur matériel embarqué à faible coût.

1.6 Réseaux d'énergie cognitive

1.6.1 Définition et concepts fondamentaux

Un réseau d'énergie cognitive (CEN, *Cognitive Energy Network*), est un réseau électrique qui intègre l'intelligence artificielle pour percevoir, raisonner et agir de façon autonome. Le système collecte des données capteurs et les analyse en temps réel avec des algorithmes d'IA, puis prend des décisions de contrôle ou d'alerte sans intervention humaine. Ce système s'appuie sur trois concept fondamentaux : l'intelligence (estimation d'état, prédiction), la perception (détection de défauts, diagnostic), et la sécurité (cybersécurité, intégrité des données). Barbalho, Aquino & Lopes (2025) regroupe les techniques d'apprentissage par renforcement appliquées au contrôle de microréseaux. Les techniques basées sur l'IA selon cette étude surpassent les méthodes conventionnelles dans la majorité des scénarios testés, autant en termes de stabilité que d'efficacité énergétique.

1.6.2 Architecture des réseaux d'énergie cognitive

L'architecture d'un CEN est structuré en couches :

- Couche physique qui génère les données (capteurs, convertisseurs, réseau électrique)
- Couche de communication avec le protocoles industriels, le réseau IP
- Couche de traitement edge/cloud pour le prétraitement des données.
- Couche cognitive pour les algorithmes d'IA, et la prise de décision.

La figure 1.2 illustre cette architecture en couches appliquée au contexte d'un microréseau DC supervisé.

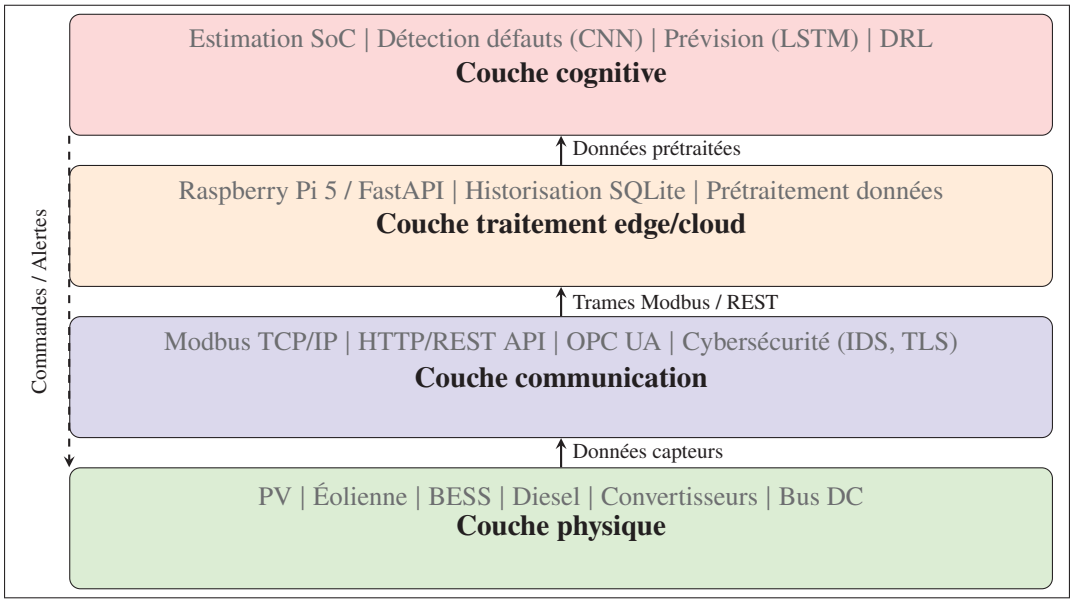

Figure 1.2 Architecture en couches d'un réseau d'énergie cognitive appliqué à un microréseau DC

Cette architecture en couches est inspirée des modèles présentés par Yıldırım *et al.* (2025) et Barbalho *et al.* (2025). Yıldırım *et al.* (2025) présentent une revue complète du edge computing appliqué aux microréseaux. Barbalho *et al.* (2025) souligne également que l'architecture des agents d'apprentissage par renforcement dans un microréseau doit tenir compte de la distribution géographique des sources et des contraintes de communication.

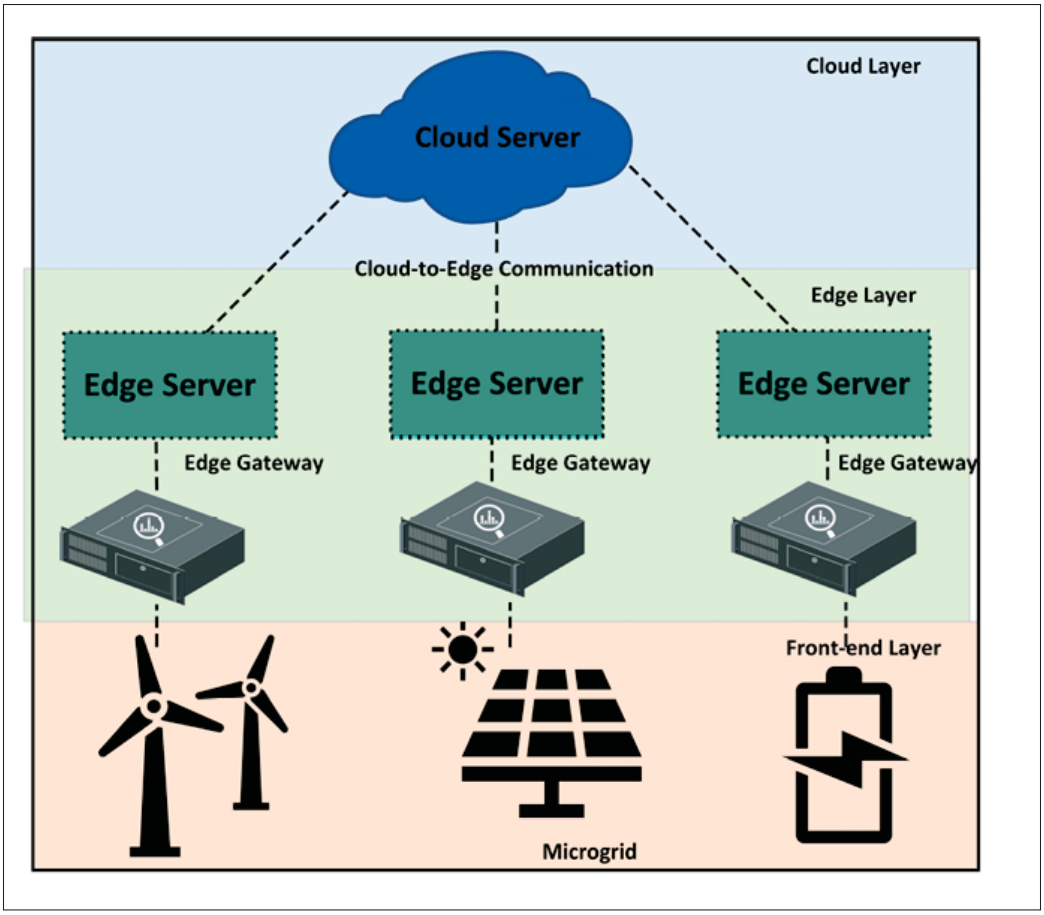

Figure 1.3 Architecture edge computing pour les réseaux électriques (tirée de Yıldırım *et al.* (2025)).

1.6.3 Intégration des modules intelligents

Les modules intelligents permettent de réaliser plusieurs actions comme la détection, les calculs et estimation, la gestion de l'énergie etc. La figure 1.4 présente l'interaction entre les modules intelligents dans le contexte de la détection de défauts d'un microréseau supervisé par SCADA.

Plusieurs méthodes sont utilisées pour l'estimation comme le filtre de Kalman étendu (EKF) qui est une méthode classique mais avec des limitations (sensibilité aux conditions initiales, hypothèse de linéarité locale), la recherche propose alors des approches hybrides. Dar & Singh (2025) combinent un réseau BiLSTM (bidirectinal Long term Short term Memory) avec un EKF

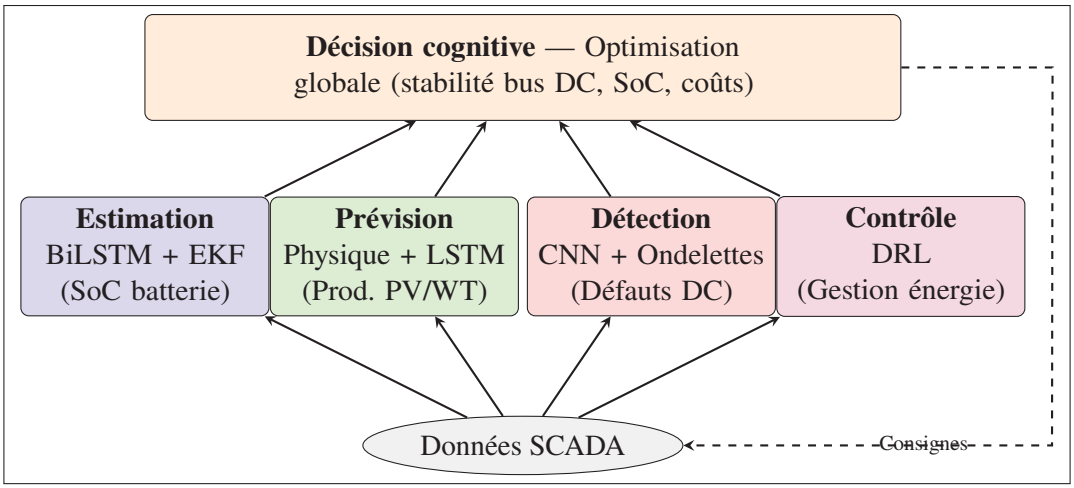

Figure 1.4 Intégration des modules cognitifs dans un microréseau supervisé

innovant pour l'estimation du SoC des batteries lithium-ion, atteignant des erreurs inférieures à 1% sur des profils de décharge variés.

La gestion optimale de l'énergie avec l'apprentissage par renforcement profond (DRL) a l'avantage de faire le contrôle à partir des données sans modèle analytique explicite. Xiong, Li, Li & Wu (2025) montre une structure qui gère simultanément les sources renouvelables et l'intégration de véhicules électriques.

La detection de defauts dans Arévalo, Cano & Jurado (2025) est basée sur des réseaux de neurones convolutifs (CNN) optimisés, combinée à la transformée en ondelettes.

1.6.4 Intégration de la cybersécurité

La cybersécurité constitue un enjeu critique pour tout système SCADA. Alomari, Alshraideh, Al Khawaldah & Arman (2025) montrent de façon précise les menaces sur les réseaux intelligents comme les : attaques man-in-the-middle, injection de fausses données (*false data injection*), déni de service (DoS). Leur analyse montre que la surface d'attaque augmente significativement avec l'adoption de protocoles IP dans les systèmes SCADA industriels. Ils identifient trois axes de protection essentiels : la protection des communications (chiffrement

TLS/SSL, VPN), la détection d'anomalies par IA (analyse du trafic réseau), et le chiffrement des données SCADA au repos et en transit.

1.6.5 Intégration des contraintes climatiques

La production des sources renouvelables (PV, éolienne) est liée aux conditions climatiques, ce qui cause une variabilité dans le bilan énergétique du microréseau. Le travail de Flórez, Correa-Flórez & Montoya (2025) développe un framework hybride combinant un modèle physique des équations aérodynamiques de la turbine éolienne et un réseau LSTM pour la prédiction de la puissance éolienne. Dans un système scada, ce type de prévision permet d'anticiper les variations climatiques et de faire la gestion de la batterie. Le système au lieu de subir les conditions climatique, les prédit et s'adapte au besoin.

1.6.6 Interopérabilité et API

Kotsiopoulos, Sarigiannidis, Ioannidis & Tzovaras (2025) montrent dans leur travail qu'une augmentation du nombre de protocoles supporté par le système entraîne une augmentation de la surface d'attaque car chaque interface devient un point d'entrée potentiel.
Yıldırım *et al.* (2025) soulignent que les architectures edge computing comme montré dans 1.3 pour les réseaux électriques doivent avoir des couches d'abstraction pour assurer la traduction entre protocoles.

1.6.7 Optimisation globale du système

L'optimisation globale d'un CEN vise à maximiser l'efficacité énergétique tout en maintenant la fiabilité et la sécurité. Dans Barbalho *et al.* (2025), les approches par apprentissage sont particulièrement adaptées à cette tâche. En effet, l'apprentissage inclut une politique de contrôle globale intégrant les objectifs comme la minimisation des coûts, le maintien de la stabilité du bus DC et le respect des contraintes opérationnelles (limites de SoC, puissance maximale des convertisseurs).

Xiong *et al.* (2025) quant a eux démontrent qu'un agent DRL unique peut gérer l'ensemble des décisions d'un microréseau avec sources renouvelables et stockage.

1.6.8 Synchronisation avec les architectures SCADA

La synchronisation entre un CEN et un SCADA signifie que les modules cognitifs, tels que l'estimation, la détection et la prévision, sont intégrés au processus de collecte et de supervision des données. Cependant, dans un CEN, ces données servent également à alimenter des modules d'intelligence artificielle qui génèrent des diagnostics, des prévisions et des recommandations de contrôle.

Yıldırım *et al.* (2025) identifient le edge computing comme moyen de synchronisation de ces éléments. Il permet d'exécuter les algorithmes d'intelligence artificielle en local avec une latence minimale. Alomari *et al.* (2025) ajoutent que cette intégration doit se faire tout en préservant la sécurité des flux de données. Les résultats des modules cognitifs passent par les mêmes canaux que les données brutes SCADA et doivent donc bénéficier des mêmes protections, comme l'authentification, le chiffrement et la journalisation.

Notre prototype, basé sur une architecture Raspberry Pi 5, FastAPI et .NET MAUI, constitue une base sur laquelle ces modules cognitifs pourraient être ajoutés dans le futur.

CHAPITRE 2

MÉTHODOLOGIE ET ARCHITECTURE

Ce chapitre présente la démarche suivie pour concevoir et développer le prototype SCADA, ainsi que l'architecture globale qui en découle. Il commence par l'approche méthodologique en six phases qui détaille la structure logicielle et matérielle, les flux de données entre la simulation, la passerelle Modbus, le serveur Raspberry Pi et l'application, et enfin le plan de tests qui servira de base à la validation expérimentale.

2.1 Approche méthodologique

Le développement suit une méthodologie itérative et incrémentale, structurée en six phases successives.

2.1.1 Conception de l'architecture.

Cette première phase fixe les fondations du système. Les composants logiciels et matériels sont identifiés (simulation MATLAB, passerelle Modbus, serveur Raspberry Pi, application .NET MAUI), les interfaces sont définies, et les exigences fonctionnelles établies.

2.1.2 MVP (Minimum Viable Product).

Un prototype minimal est mis en place pour valider la chaîne de bout en bout. Il se limite à la communication Modbus TCP entre la simulation et le client SCADA, et à une visualisation basique des grandeurs principales pour démontrer la faisabilité.

2.1.3 Extension fonctionnelle.

Sur la base du MVP, les fonctionnalités complémentaires d'un SCADA sont ajoutées progressivement comme l'historisation locale des mesures, la gestion des alarmes, et la création des pages additionnelles de l'interface pour des vues détaillées.

2.1.4 Optimisation.

Cette phase couvre l'optimisation des écritures SQLite (transactions groupées, mode WAL), les reconnexions Modbus en cas de coupure, et l'amélioration de la réactivité de l'interface utilisateur.

2.1.5 Déploiement distribué.

Le Raspberry Pi héberge le serveur d'historisation et expose les données via une API REST (FastAPI), pendant que les clients .NET MAUI reçoivent ces données sur l'ordinateur, validant ainsi la séparation entre acquisition, lecture et présentation des données.

2.1.6 Validation.

Consiste à faire des tests fonctionnels de la communication Modbus, mesures de performance de l'historisation et de l'API, vérification du comportement de l'interface, et rédaction de la documentation associée.

2.2 Architecture globale du système

2.2.1 Vue niveau système

Le système intègre deux composants majeurs : (1) une simulation du micro-réseau DC sur MATLAB/Simulink, et (2) un système SCADA supervisant cette simulation. L'architecture globale s'articule sur quatre niveaux, et a été validée selon deux configurations complémentaires.

2.2.1.1 Configuration de test 1 : Pipeline Modbus distribué.

Cette configuration valide l'architecture distribuée, où les données transitent par le protocole Modbus TCP et le serveur Raspberry Pi :

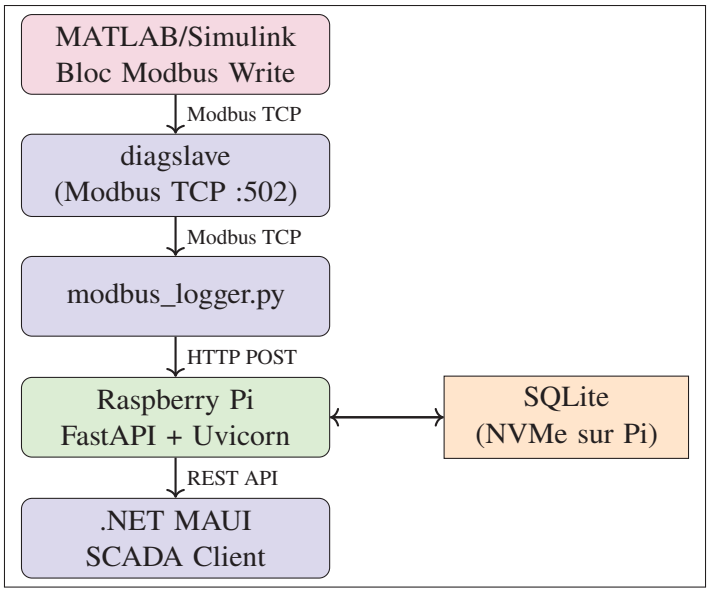

Figure 2.1 Configuration de test 1 : Pipeline Modbus distribué

2.2.1.2 Configuration de test 2 : Journalisation SQLite directe.

Cette configuration exploite la fonction `log_data.m` crée a cet effet pour écrire directement dans une base SQLite locale, lue ensuite par l'application MAUI. Plus rapide à mettre en œuvre et offrant un débit de données plus élevé, elle s'est avérée mieux adaptée au développement itératif de l'interface, en particulier pour valider le comportement des fenêtres glissantes dans les graphiques historiques (ChartsPage) :

Niveau 0 : Simulation FCS-MPC : Le microréseau DC est simulé en MATLAB avec un pas de temps $T_s = 50\,\mu$s. Cinq convertisseurs de puissance contrôlés par FCS-MPC sont présents : l'onduleur VSI triphasé (8 états de commutation), le convertisseur buck-boost BESS, le boost PV avec MPPT, le boost WT avec MPPT P&O, et le buck DG.

Niveau 1 : Exportation de données : Deux chemins d'exportation coexistent. En *configuration 2* (SQLite directe), la fonction `log_data.m` avec sous-échantillonnage (×20) et buffering (1000 points) exporte les 29 grandeurs mesurées vers SQLite local. En *configuration 1* (Modbus distribué), un bloc `Modbus Client Write` dans Simulink écrit ces grandeurs comme registres

holding vers le serveur externe diagslave (port 502), où elles sont ensuite lues par le script `modbus_logger.py`.

Niveaux 2-3 : SCADA : Le client .NET MAUI supervise par lecture directe sur la base de données ou via l'API REST du Raspberry Pi.

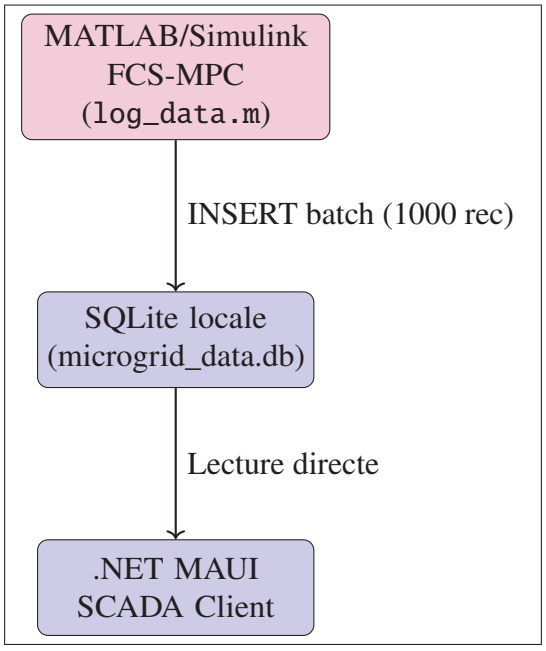

Figure 2.2 Configuration de test 2 : Journalisation SQLite directe

2.2.2 Modes opérationnels

Les deux configurations décrites ci-dessus correspondent à deux modes opérationnels intégrés dans l'application. Le mode Local SQLite (configuration 2) offre une meilleur rapidité dans l'acquisition des données depuis Matlab et un déploiement simplifié (c'est cette configuration qui a permis de valider efficacement les fenêtres glissantes des graphiques historiques). Le mode Distant (configuration 1) valide l'architecture distribuée cible avec le Raspberry Pi, le serveur modbus et le logger. Le système supporte ces deux modes de manière configurable :

1. **Mode Local SQLite :**

   Dans ce mode de fonctionnement, comme montré sur la figure 2.2, la simulation MATLAB écrit directement sur la base de données SQLite locale à l'aide d'une fonction. Ensuite l'application MAUI vient lire directement depuis la base de données pour l'affichage. Cette configuration ne nécessite pas de serveur et est donc adaptée aux test de développement et démonstrations hors ligne.

2. **Mode Distant (Pi) :**

   Dans cette configuration 2.1, la simulation MATLAB utilise un bloc Modbus Write pour écrire sur un serveur Modbus tampon (Diagslave) sur le port 502 ; le script d'historisation Python lit de façon périodique les données sur le serveur Modbus et les envoie sur le Raspberry via une requête HTTP. Les données sont ensuite stockées dans la base de données SQLite sur la carte. Pour afficher les données, l'application interroge Pi via une requête HTTP (Authentification basique) ; cette configuration est adaptée aux déploiements distribués sur ordinateur.

2.3 Architecture logicielle .NET MAUI

2.3.1 Architecture MVVM implémentée

Composants clés :

- **Views (XAML) :** MicrogridDashboard, ChartsPage, DataHistorianPage, AlarmsAndEventsPage, etc.
- **ViewModels :** INotifyPropertyChanged,
- **Models :** SensorData (données microréseau), AlarmEvent (alarmes), DieselMetrics(paramètres moteur)
- **Services :** ModbusService, SqliteDataService, PiRemoteDataService, AlarmService, ConnectionManager etc

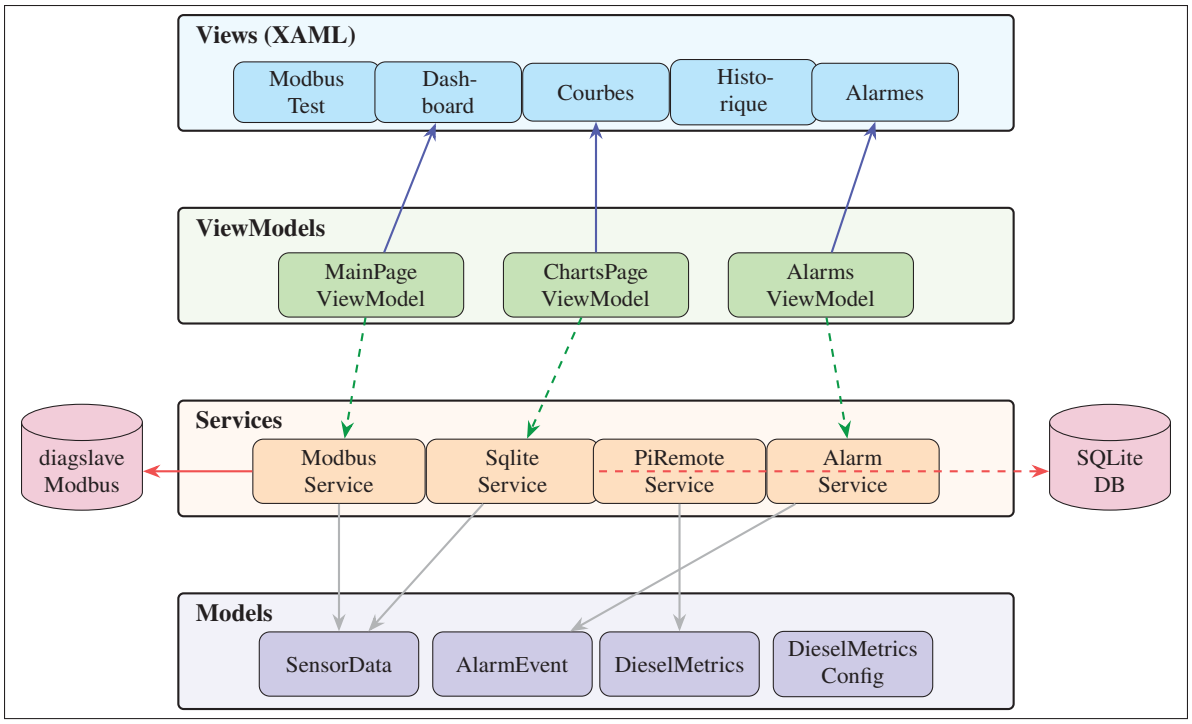

Figure 2.3 Architecture MVVM implémentée

2.3.2 Services

2.3.2.1 ConnectionManager

Le `ConnectionManager` est un fichier qui gère l'état global de la connexion avec la variable Bool `IsConnected`, qui indique si le système est connecté ou déconnecté, en déclenchant l'événement `ConnectionStatusChanged`, en fournissant les méthodes `ConnectAsync()` et `Disconnect()`, et en assurant périodiquement la vérification et le maintien de la connexion. Le modal de connexion indique pour la premiere connexion le mode de fonctionnement local ou distant (III-3 ou III-2).

2.3.2.2 ModbusService

Le ModbusService est un fichier permetant l'incorporation de la bibliothèque NModbus, Si *et al.* (2021); Véliz, Riffo, Restrepo, Rivera *et al.* (2023). Utilisé uniquement sur la page test au depart du développement dont les principales fonctions sont :

- La connexion TcpClient + ModbusMaster (NModbus 3.x)
- La gestion des temps morts (timeout), reconnexion sur le port
- Conversion Big Endian IEEE 754 (2 registres en Double 32-bit), Modbus Organization (2012)

2.3.2.3 SqliteDataService (IDataService)

Service permettant l'implémentation locale de la base de données SQLite en configuration 2 :

- SQLiteAsyncConnection (sqlite-net-pcl)
- Table SensorData : timestamp, tensions, courants, puissances, SoC, etc
- Méthodes CRUD (Create Read Update Delete) asynchrones
- Stratégies optimisation (détaillées section suivante)
- Exportation en CSV

2.3.2.4 PiRemoteDataService (IDataService)

Permet d'utiliser `HttpClient` avec une authentification basique basée sur une table de la base de données pour communiquer avec les endpoints REST `/api/data`, `/api/data/latest` et `/api/stats`, s'appuie sur la sérialisation JSON via `System.Text.Json`, et garantit ainsi une compatibilité avec une autre implémentation reposant sur la même interface `IDataService`.

2.3.2.5 AlarmService

Permet la gestion des alarmes avec des degrés de priorité variables (CRITICAL, HIGH, WARN, INFO).

- Table SQLite : AlarmEvent (ID, Timestamp, Priority, Message, IsActive, IsAcknowledged)
- Priorités : CRITICAL, HIGH, WARN, INFO

2.3.2.6 AutoAlarmService

Fournit la logique de génération automatique des alarmes :

- Surveillance seuils configurables (tensions, courants, SoC batterie, températures)
- Comparaison avec SensorData
- Déclenchement/résolution automatique

2.3.3 Interfaces utilisateur principales

1. **ConnectionPopup :** Modal pour le choix Local/Pi, permet la configuration IP/port
2. **SystemOverview :** montre une Vue synthétique des indicateurs de performance, du statut sources, production, consommation
3. **MicrogridDashboard :** Schéma synoptique animé, gauges, indicateurs visuels
4. **ChartsPage :** Graphiques historiques (LiveChartsCore), zoom, périodes de visualisation configurables
5. **DataHistorianPage :** Tableau données, filtrés par date/heure, exportation CSV
6. **AlarmsAndEventsPage :** Liste alarmes actives, acquittement, filtrés par priorité
7. **DieselGeneratorPage :** Métriques diesel spécifiques, historique démarrages, analyse environnementale
8. **CircuitDiagramPage :** Schéma bloc animé avec flux puissance
9. **ModbusTestPage :** Outil diagnostic : lecture/écriture registres manuels Modbus et SQLite

2.4 Implémentation côté serveur

2.4.1 Enregistrements (Logging)

Les enregistrements ou logging des données se font par connexion du serveur Modbus TCP (diagslave) vers MATLAB (localhost :502),ainsi que la lecture cyclique des registres, conversion des registres en dictionnaire Python par le script logger, envoi des données en JSON par requête POST vers le Raspberry Pi (/api/batch), et gestion des erreurs niveau réseau et logique de nouvelle tentative de reconnexion.

2.4.2 FastAPI sur Raspberry Pi

Le système fonctionne grâce à un serveur REST léger, créé avec FastAPI et Uvicorn sur le port 8000, qui utilise une base de données SQLite stockée sur le SSD NVMe. Il expose des points de terminaisons (endpoints) spécifiques pour consulter les données du microréseau, insérer des données par lots, obtenir des statistiques de la base ainsi que suivre l'état du serveur.

- Framework : FastAPI + Uvicorn (port 8000)
- Base données : SQLite (/data/scada/microgrid.db sur NVMe SSD)
- Endpoints :
  - GET /api/microgrid ?limit=N : Derniers N enregistrements (authentification basique)
  - POST /api/microgrid/batch : Insertion batch (depuis script Python)
  - GET /api/microgrid/stats : Statistiques DB
  - GET /api/health : État du serveur
- Authentification : Basic Auth (variables environnement)

2.4.3 Structure de la base de données SQLite

L'historisation des données du microréseau utilise une base de donnée SQLite de forme identique dans les deux configurations mises en place (`microgrid_data.db`) dont la table principale `sensor_data` contient les colonnes suivantes :

Tableau 2.1 Structure de la table `sensor_data` (colonnes principales)

Colonne	Type	Description
id	INTEGER	Clé primaire auto-incrémentée
timestamp	REAL	Timestamp Unix
vbus	REAL	Tension du bus DC (V)
pbus	REAL	Puissance du bus DC (W)
ibus	REAL	Courant du bus DC (A)
va_dg, vb_dg, vc_dg	REAL	Tensions de phase GD (V)
ia_dg, ib_dg, ic_dg	REAL	Courants de phase GD (A)
iout_dg, pout_dg	REAL	Courant/puissance sortie GD (A, W)
va_load, vb_load, vc_load	REAL	Tensions de phase charge AC (V)
ia_load, ib_load, ic_load	REAL	Courants de phase charge AC (A)
g_load, p_load	REAL	Conductance (S) / puissance charge (W)
iout_bess, pout_bess	REAL	Courant/puissance BESS (A, W)
soc_bess, preq_bess	REAL	SoC (%) / puissance requise BESS (W)
iout_wt, pout_wt, omega_wt	REAL	WT : courant (A), puissance (W), vitesse (rad/s)
iout_pv, pout_pv, g_irr	REAL	PV : courant (A), puissance (W), irradiance (W/m^2)

Fonctionnalités du service SqliteDataService : Regroupe les fonctions permettant de traiter les données de la base de donnée SQLite. Pour chacune le role est :

- `GetRecentDataAsync(seconds)` : Données des N dernières secondes (basé sur timestamp Unix)
- `GetLatestRecordsByTimestampAsync(count)` : donne les N derniers enregistrements ordonnés par timestamp décroissant
- `GetLatestDataAsync()` : Dernier enregistrement uniquement
- `CleanupOldDataAsync(hours)` : execute la suppression données obsolètes (gestion taille DB)
- `GetDataCountAsync()` : Nombre total d'enregistrements (statistiques)

Le modèle C# `SensorData` utilise les attributs SQLite-net (`[Table]`, `[Column]`, `[PrimaryKey]`) pour la cartographie objet-relationnel automatique. La base peut être partagée entre les entrées de données depuis MATLAB/Simulink et l'application SCADA (lecture asynchrone), permettant une historisation continue durant les simulations.

2.5 Tests et validation

2.5.1 Environnement de test

- PC développement : Razer Blade 14 : Windows 11, AMD Ryzen, GPU : NVIDIA GeForce RTX 3070, 16 GB RAM
- Raspberry Pi 5 Model B (8 GB RAM, Raspberry Pi OS Debian Trixie 64-bit)
- Réseau local Ethernet Gigabit via switch TP-Link TL-SG105 (LAN direct, pas de Wi-Fi)
- MATLAB R2024b avec Simulink (29 signaux à 100 Hz via bloc Modbus)
- Serveur Modbus : diagslave sur localhost :502 (recevant les données MATLAB)

2.5.2 Scénarios de test fonctionnels

1. **Test connexion Modbus :**
    - Connexion réussie vers serveur test
    - Lecture registres multiples
    - Gestion déconnexion/reconnexion
2. **Test historisation SQLite locale :**
    - Écriture 100 enregistrements/s pendant 5 minutes
    - Vérification intégrité données
    - Mesure charge CPU/disque
3. **Test mode distant Pi :**
    - Logger POST vers Pi (100 requêtes)
    - MAUI GET depuis Pi

- Validation cohérence données bout-en-bout

4. **Test UI réactivité :**

   - Rafraîchissement dashboard (mesure fps)
   - Charts streaming temps réel (1000 points)
   - Navigation entre pages (latence)

5. **Test alarmes :**

   - Génération d'alarmes avec priorités variées
   - Acquittement des alarmes, test filtres
   - Persistance après redémarrage app

Ce chapitre a posé les bases méthodologiques et architecturales du prototype. La démarche en six phases permet d'enchaîner conception, MVP, extension fonctionnelle, optimisation, déploiement distribué et validation. L'architecture retenue déclinée en deux configurations de test définit clairement les rôles de la simulation, du serveur Raspberry Pi et de l'application .NET MAUI, ainsi que le plan de tests qui sera développé au chapitre suivant pour guider l'implémentation détaillée du prototype.

CHAPITRE 3

IMPLÉMENTATION

L'implémentation de la méthodologie abordée au chapitre précédent appliquée au microréseau de Dubuisson (2023) est illustré dans ce chapitre. Elle commence par la simulation du microréseau ensuite le développement et l'optimisation de l'application SCADA.

3.1 Développement de l'application MAUI

Le développement de l'application s'est fait avec le logiciel Visual Studio Community 2022 v 17.13.2 de Microsoft avec les packages installés : Développement .Net Desktop, développement .NET Multi-Platform App UI (MAUI). Après installation, créer un nouveau projet .Net MAUI, un code passe-partout(boilerplate) est alors généré qui sera modifié aux besoins.

3.1.1 Configuration

Fichier MauiApp1.csproj :

Fichier de configuration du système indiquant au compilateur la plateforme ciblée (Android, Windows ou macOS). Permet la configuration entre autres des polices de caractères, images, logo, autres ressources et des bibliothèques utilisées.

- TargetFrameworks : net8.0-windows10.0.19041.0 ; net8.0-android ; net8.0-ios
- OutputType : Exe
- UseMaui : true
- SingleProject : true

Packages NuGet :

Bibliotheques NuGet installées depuis le gestionnaire de package NuGet :

- NModbus (3.0.81) : pour la communication Modbus TCP/IP
- sqlite-net-pcl (1.9.172) : Historisation SQLite
- LiveChartsCore.SkiaSharpView.Maui (2.0.0-rc2) : visualisation graphiques temps réel

- SkiaSharp.Views.Maui.Controls (2.88.8) : Rendu graphique
- SQLitePCLRaw.bundle_green (2.1.10) : Fournisseur SQLite natif

3.1.2 Configuration centralisée

`DatabaseConfig` est une classe statique qui regroupe le chemin de la base de données et le mode de connexion actif (`LocalSqlite` dans la configuration 2 ou `PiServer` dans la configuration 1). La propriété `CurrentDbPath` retourne le chemin SQLite local ou une chaîne vide selon le mode choisi, ce qui permet de basculer entre les deux configurations d'intégration de manière transparente. Le choix de l'utilisateur via le modal `ConnectionPopup` est sauvegardé dans les `Preferences` MAUI pour une connection future plus rapide.

3.1.3 Service Modbus thread-safe

Le `ModbusMaster` de NModbus n'est pas thread-safe, Si *et al.* (2021) : quand plusieurs pages lancent des lectures en même temps, cela provoquait des corruptions du flux TCP. La solution retenue sérialise toutes les requêtes Modbus, garantissant ainsi qu'une seule lecture ou écriture traverse à tout instant. C'est ce choix qui a été préféré à l'alternative de créer un `ModbusMaster` par page, ce qui aurait multiplié les connexions TCP.

3.1.4 Conversion registres Modbus

MATLAB écrit les valeurs flottantes 32-bit sur deux registres Modbus consécutifs en format Big Endian, Modbus Organization (2012). La méthode `ReadHoldingRegistersAsFloatAsync` lit 2n registres de 16-bit, puis reconstruit chaque réel en réarrangeant les octets : le mot de poids fort (registre pair) est placé avant le mot de poids faible (registre impair), avec inversion des octets au sein de chaque mot pour respecter l'endianness .NET (Little Endian). La conversion utilise `BitConverter.ToSingle`.

3.2 Optimisation de l'historisation SQLite

3.2.1 Problème initial

Les premiers essais d'insertion (1 insertion par mesure) s'avérait inefficace sur la plateforme et entrainaient une latence perceptible dans l'interface utilisateur, augmentation de la charge du CPU(pouvant aller à environs 40%) et une perte de données. L'écriture naïve 10 Hz (1 INSERT par mesure) s'avérait inefficace sur plateforme embarquée. Des stratégies ont été mises en place pour régler ces problèmes.

3.2.2 Stratégie 1 : Buffering avec transactions groupées

Le principe est de faire l'accumulation de N échantillons en mémoire (BUFFER_SIZE = 1000), puis les écrire en une seule fois sur SQLite. Chaque appel ajoute au buffer ; lorsque celui-ci est plein en une transaction unique.

Les résultats montrent que la charge CPU mesurée avec le gestionnaire de tâches de l'ordinateur diminue, mais risque de perte buffer s'il y a arrêt brusque de l'application avant téléversement vers le serveur.

3.2.3 Stratégie 2 : Downsampling intelligent (sous-échantillonnage)

Pour les données de plus d'une heure d'ancienneté, un processus de sous-échantillonnage remplace les enregistrements individuels par une moyenne par secondes. La méthode `DownsampleOldDataAsync` regroupe les données anciennes par secondes (groupées sur le timestamp correspondant), calcule la moyenne de chaque variable, supprime les originaux et insère les agrégats le tout dans une transaction unique pour garantir la cohérence.

On constate une réduction de la taille de la base de données de 50%, et amélioration des performances des requêtes historiques sur de longues périodes.

3.2.4 Stratégie 3 : WAL mode + Pragma optimization

Il s'agit de faire des opérations de lecture et d'écriture concourantes (WAL) et gérer la cache pour augmenter le débit.

```
private async Task InitAsync()
{
    _database = new SQLiteAsyncConnection(_dbPath);
    await _database.ExecuteAsync("PRAGMA journal_mode=WAL");
    await _database.ExecuteAsync("PRAGMA synchronous=NORMAL");
    await _database.ExecuteAsync("PRAGMA cache_size=10000");
    await _database.CreateTableAsync<SensorData>();
}
```

- WAL : Lectures concurrentes pendant écritures
- synchronous=NORMAL : Assure l'équilibre fiabilité/performance
- cache_size : 10 000 pages (40 MB) pour réduire les flush disque

3.2.5 Stratégie 4 : Écriture asynchrone dédiée (Service en arrière plan)

Un thread dédié utilise une file en arrière-plan. Les données sont ajoutées via `QueueInsertAsync` sans causer de blocage ou ralentissement dans le thread UI ; le thread écrivain traite les éléments de facon séquentielle et les insère dans SQLite.

On voit que l'interface utilisateur est totalement découplée des opérations d'écriture, donc aucune interruption sèche perceptible.

3.2.6 Comparaison stratégies

Seule la stratégie 1 (buffering avec transactions groupées) est implémentée dans le code du SCADA MAUI. Les stratégies 2,3 et 4 sont des alternatives étudiées mais dans le déploiement

final, l'historisation est faite coté serveur : le script logger Python envoie des groupes de 1000 échantillons via HTTP POST au FastAPI sur Pi 5 avec NVMe SSD.

Stratégie finale retenue : Buffering batch (1000 échantillons côté Python logger) + transactions groupées côté FastAPI.

3.3 Implémentation API REST Raspberry Pi

3.3.1 Structure FastAPI

Le serveur REST est implémenté en Python avec FastAPI et Uvicorn. L'application utilise quatre endpoints (points de terminaison) principaux : `GET /api/microgrid` (pour la lecture des pages avec authentification basique), `POST /api/microgrid/batch` (pour l'insertion par lots depuis le logger Python), `GET /api/microgrid/stats` (pour les statistiques de la base de donnée) et `GET /api/health` (pour avoir l'état du serveur). L'authentification utilise `HTTPBasic` avec comparaison sécurisée. Les données sont stockées dans SQLite sur le NVMe SSD monté au niveau de `/data/scada/`.

3.3.2 Déploiement systemd

Le serveur déployé comme un service systemd (`scada-api.service`) avec redémarrage automatique (`Restart=always`) pour assurer une reprise des opérations en cas de coupure brève. L'activation se fait par `systemctl enable/start`. La configuration complète du service est documentée en Annexe A.

3.4 Interfaces utilisateur avancées

3.4.1 Tableau de bord

La page `MicrogridDashboard.xaml` constitue l'interface principale de supervision. Elle utilise une grille adaptative à trois lignes (en-tête, synoptique, pied de page) avec des cadres

pour chaque source d'énergie. Les valeurs sont liées aux propriétés du ViewModel avec une liaison des données MVVM (`{Binding PvPower}`, `{Binding BusVoltage}`, etc.).Le menu permettant la navigation se situe sur le coté gauche. Le bus DC central occupe visuellement le centre de la mise en page montrant clairement la tension du bus lue sur la base de donnée comme le montrela figure 3.1. On distingue aussi clairement les puissances engagées sur le réseau, les alarmes récentes et autres diagrammes explicatifs.

Fonctionnalités visuelles :

- Mise en page adaptée aux différentes tailles d'écran
- Jauges circulaires personnalisées avec SkiaSharp
- Animations de flux de puissance
- Mises à jour réactives via `INotifyPropertyChanged` pour toujours montrer les valeurs récentes

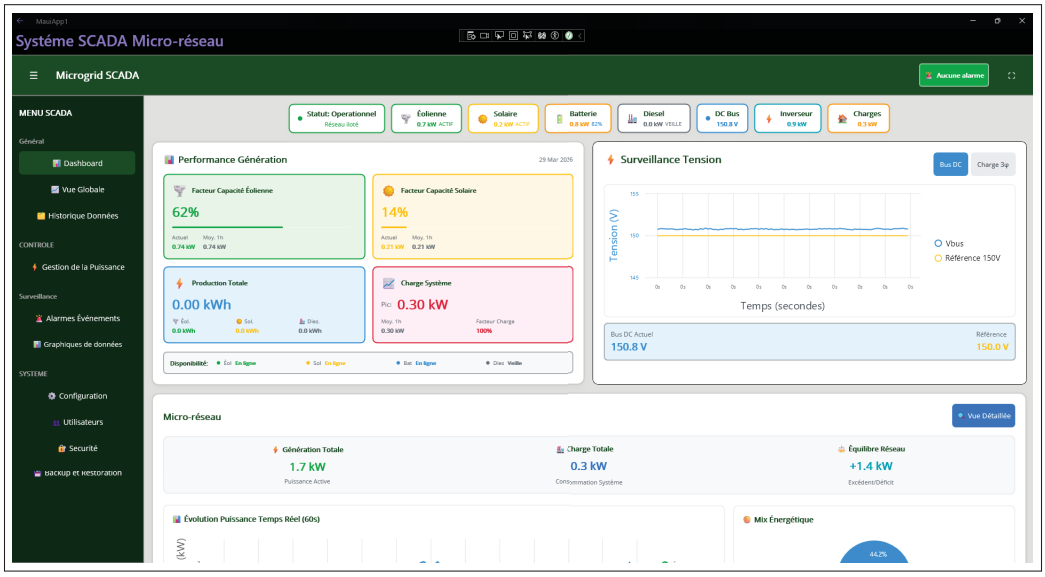

Figure 3.1 Dashboard principal de l'application SCADA

3.4.2 Graphiques historiques

La page `ChartsPage` utilise la bibliothèque LiveChartsCore.SkiaSharpView.Maui pour le rendu de graphiques vectoriels performants. Le ViewModel expose une collection liée au composant

`CartesianChart` en XAML. Pour chaque diagramme, chaque série est configurée comme `LineSeries<double>` avec rendu SkiaSharp (trait coloré, sans remplissage, sans marqueurs géométriques pour la fluidité). La page montre une repartition en onglets 3.2 pour la tension, le courant, la puissance et le constructeur de diagramme.

Fonctionnalités :

- Zoom/pan tactile et tooltips interactifs
- Axes temporels automatiques avec périodes configurables
- Séries multiples superposées (tensions, courants, puissances)
- Fenêtre glissante (100 points)
- Constructeur de diagramme pour observation de diagrammes en particulier

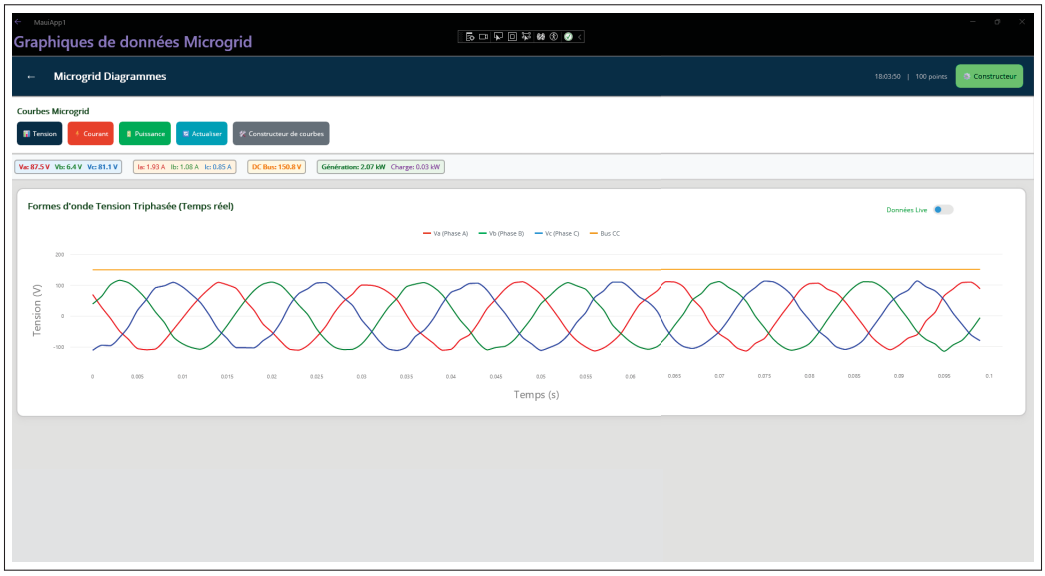

Figure 3.2 Page de visualisation des graphiques temporels SCADA

3.4.3 Exportation en CSV

La page d'historisation se présente comme montre 3.3 permet de visualiser les données historiques sur des plages de temps différentes au besoin. La méthode `ExportDataAsync` génère un fichier CSV horodaté 3.4 (`data_historian_yyyyMMdd_HHmmss.csv`) contenant 8 colonnes : timestamp ISO 8601, puissances éolienne/solaire/diesel/charge (kW), SoC

batterie (%), puissance BESS (kW), et tension bus DC (V). L'export utilise `StreamWriter` standard sans dépendance externe, garantissant la compatibilité cross-platform (Windows, Android, iOS). Les timestamps Unix stockés dans SQLite sont convertis en dates lisibles via `DateTimeOffset.FromUnixTimeSeconds`. Les fichiers produits s'ouvrent directement dans Excel ou LibreOffice Calc comme le montre la figure 3.4.

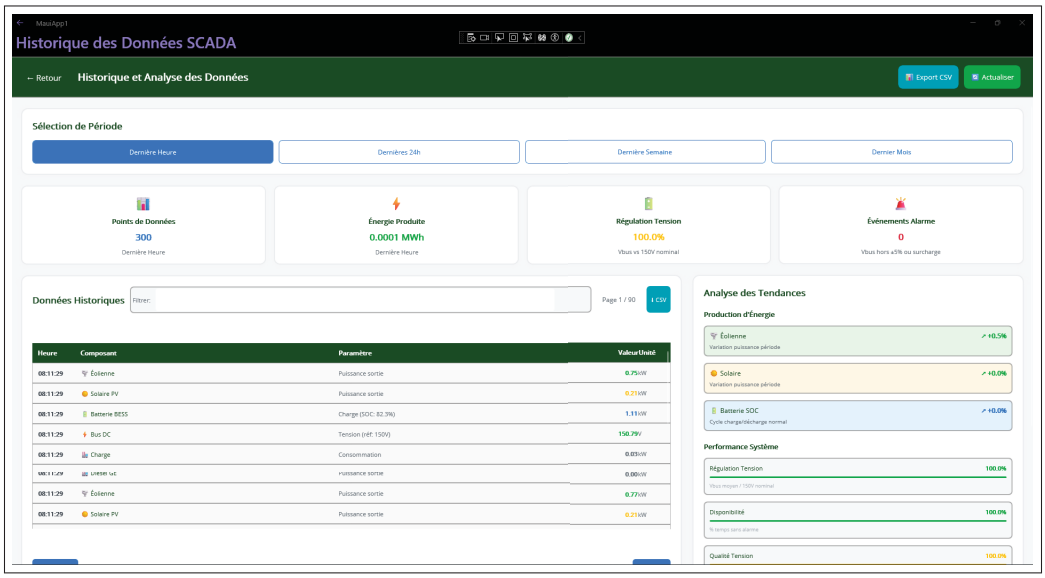

Figure 3.3 Page d'historisation des données SCADA

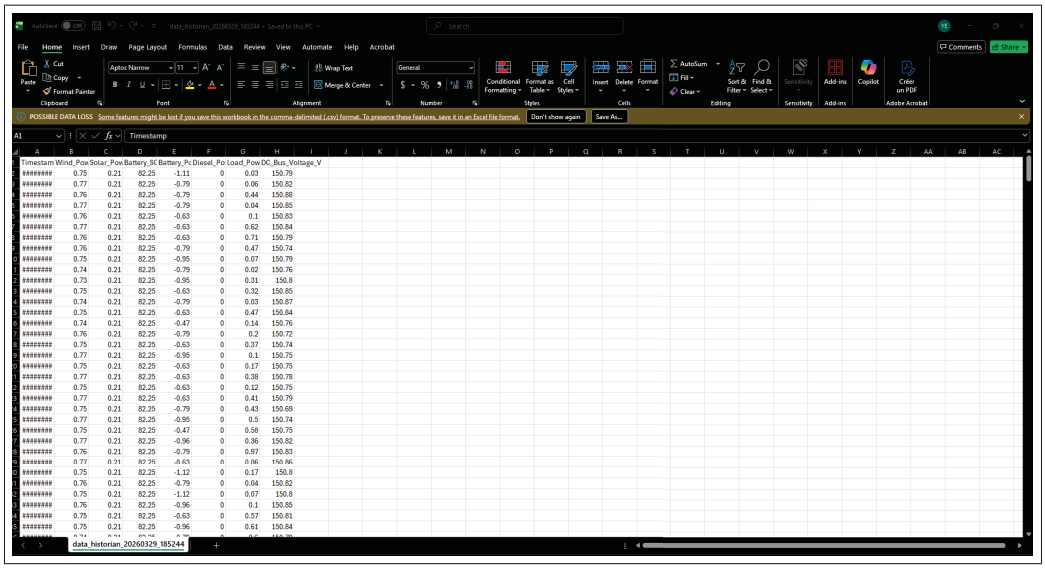

Figure 3.4 Fichier CSV généré pour historisation

3.5 Simulation FCS-MPC (MATLAB/Simulink)

3.5.1 Vue d'ensemble

Cette section décrit la simulation du microréseau DC, réimplémentant sous MATLAB/Simulink les algorithmes et la topologie proposés dans la thèse de Dubuisson (2023). Notre contribution spécifique consiste à (1) transposer ces algorithmes en bloc fonctions MATLAB, (2) intégrer la journalisation vers SQLite (directe et à travers Modbus vers Pi), et (3) les connecter à l'infrastructure SCADA développée dans ce mémoire. Les algorithmes de contrôle eux-mêmes, les paramètres des convertisseurs, et quelques scénarios de test sont issus de ces travaux.

3.5.2 Architecture de simulation

La simulation du microréseau DC est développée sous MATLAB Simulink avec des blocs fonctions MATLAB liés avec un pas de temps $T_s = 50\,\mu$s. Le microréseau comprend cinq sources/charges interconnectées par un bus DC à 150 V :

- **VSI triphasé (onduleur) :** Interface DC/AC vers la charge résistive triphasée, contrôlé par FCS-MPC avec 8 états de commutation
- **Convertisseur boost PV :** Alimentation DC depuis les panneaux photovoltaïques, FCS-MPC + MPPT
- **Convertisseur boost WT :** Alimentation DC depuis la génératrice synchrone éolienne (PMSG), FCS-MPC + MPPT P&O
- **Convertisseur buck DG :** Alimentation DC depuis le générateur diesel (400 V), FCS-MPC avec référence de courant configurable
- **Convertisseur réversible buck-boost BESS :** Stockage/restitution d'énergie avec la batterie (108 V, SoC initial 80%), FCS-MPC + régulation du bus DC

Les 29 variables d'état et de puissance journalisées constituent le jeu de données supervisé par le SCADA, décrit dans le tableau 3.1.

Tableau 3.1 Variables de simulation exportées vers le SCADA (29 paramètres)

Variable	Description	Composant	Unité
V_{bus}	Tension bus DC	Bus central	V
P_{bus}, I_{bus}	Puissance/courant bus	Bus central	W, A
$V_{a,b,c,DG}$	Tensions de phase DG	Diesel	V
$I_{a,b,c,DG}$, $I_{out,DG}$, P_{DG}	Courants/puissance DG	Diesel	A, W
$V_{a,b,c,load}$	Tensions de phase charge	Charge	V
$I_{a,b,c,load}$, G_{load}, P_{load}	Courants/conductance/puissance	Charge	A, S, W
$I_{out,BESS}$, P_{BESS}, SoC, $P_{req,BESS}$	BESS : courant, puissance, SoC, ref.	Batterie	A, W, %
$I_{out,WT}$, P_{WT}, ω_{WT}	WT : courant, puissance, vitesse	Éolienne	A, W, rad/s
$I_{out,PV}$, P_{PV}, G_{irr}	PV : courant, puissance, irradiance	Solaire	A, W, W/m^2

3.5.3 Scénarios de simulation

Les scénarios de test dérivent des figures présentées par Dubuisson (2023) (Fig. 4, Fig. 5, etc.) et étendus pour les besoins de démonstration SCADA (scénarios 10–11). Onze scénarios permettent de tester différents scénarios opérationnels du microréseau :

Tableau 3.2 Scénarios de simulation FCS-MPC disponibles

Scénario	Description	Durée typique
1	Rampe WT (démarrage éolienne)	1 s
2	Échelon d'irradiance PV (5001000 W/m^2)	1 s
3	Délestage charge (déconnexion à 0,5 s)	1 s
4	Connexion générateur diesel avec PV bas	1 s
5	Charge batterie (génération excédentaire)	1 s
6	Décharge batterie (génération faible)	1 s
7	Scénario multi-sources variable (DG intermittent)	5 s
8	PV variable + délestage charge (Fig. 4 article)	1 s
9	Connexion/déconnexion DG + délestage (Fig. 5 article)	1 s
10	Scénario réaliste 24 heures	86400 s
11	Comparaison avec/sans DG côte à côte	variable
12	Test journalisation longue durée SCADA (10 s)	10 s

3.5.4 Journalisation directe (configuration 2) vers le SCADA

La fonction `log_data.m` constitue l'interface entre la simulation et le SCADA. Elle utilise une stratégie de double optimisation :

- **Sous-échantillonnage (facteur** = 20) **:** Seul 1 échantillon sur 20 est retenu, soit une fréquence effective de 50 μs $\times$ 20 = 1 ms = 1 kHz. Acceptable pour la supervision SCADA
- **Buffering (1000 échantillons, 29 variables) :** Les données sont accumulées dans un buffer de taille 1000×29, puis transportées vers SQLite en une seule transaction lorsque le buffer est plein.(Section 3.2)

```
function log_data(vbus, pbus, ibus, ...)
    persistent counter buffer bufferIdx bufferSize DOWNSAMPLE_FACTOR
    if isempty(counter)
        counter = 0;
        bufferSize = 1000;
        buffer = zeros(bufferSize, 29);
        bufferIdx = 1;
        DOWNSAMPLE_FACTOR = 20;
    end
    counter = counter + 1;
    if mod(counter, DOWNSAMPLE_FACTOR) == 0
        buffer(bufferIdx, :) = [vbus, pbus, ibus, ...];
        bufferIdx = bufferIdx + 1;
        if bufferIdx > bufferSize
            log_to_sqlite_bulk(buffer);
            bufferIdx = 1;
        end
    end
end
```

L'utilisation du modificateur `coder.extrinsic('log_to_sqlite_bulk')` permet d'appeler la fonction MATLAB depuis le code généré (Simulink Coder) sans interruption de la simulation, ce qui préserve la compatibilité avec les appels de fonctions externes (SQLite, Modbus).

3.5.5 Journalisation avec Modbus TCP via bloc Simulink

En complément de la journalisation SQLite directe, une autre configuration d'intégration a été mise en place pour valider l'architecture. Dans cette configuration, les données de simulation passent par le protocole Modbus TCP selon le chemin suivant :

1. **Bloc Modbus Write (MATLAB/Simulink) :** Un bloc `Modbus Client Write` intégré au modèle Simulink écrit les 29 variables de simulation dans des registres holding (format float 32-bit Big Endian, 2 registres par variable) sur le serveur à l'adresse localhost :502 à chaque pas de sous-échantillonnage. Ainsi qu'un registre supplémentaire qui servira à indiquer au script Logger si une simulation est en cours.
2. **diagslave (serveur Modbus TCP) :** Le logiciel `diagslave` écoute sur le port 502 et stocke les registres écrits par MATLAB. Il agit comme serveur Modbus intermédiaire, faisant la séparation entre la simulation et l'infrastructure SCADA.
3. **modbus_logger.py (pont Python) :** Le script lit de façon cyclique les registres depuis diagslave (100 Hz), reconstruit les valeurs à partir des paires de registres, et envoie les données par HTTP POST (batch JSON) vers le serveur FastAPI sur le Raspberry Pi lorsque le buffer est plein (1000 échantillons).
4. **FastAPI sur Raspberry Pi :** Le serveur reçoit les lots, les insère dans SQLite, et les expose via l'API REST pour l'application SCADA.

Cette configuration est plus longue à mettre en place que la journalisation SQLite directe, mais elle valide le chemin de données complet tel qu'il serait déployé dans un environnement de production avec un microréseau physique.

Ce chapitre a détaillé l'implémentation complète du prototype, de l'application à la chaîne d'acquisition des données côté simulation. Les choix techniques retenus à chaque étape

(architecture MVVM, transactions SQLite groupées, batch HTTP, registres holding 32 bits) préparent les mesures de performance et les tests fonctionnels qui font l'objet du chapitre suivant.

CHAPITRE 4

VALIDATION EXPÉRIMENTALE ET RÉSULTATS

Ce chapitre présente la validation expérimentale du prototype SCADA développé dans les chapitres précédents. La configuration matérielle et logicielle utilisée pour les essais est d'abord définie et les résultats sont organisés autour des principaux objectifs spécifiques : tests fonctionnels de la communication Modbus et de l'API REST, mesures de performance de l'historisation SQLite sur Raspberry Pi.

4.1 Configuration expérimentale

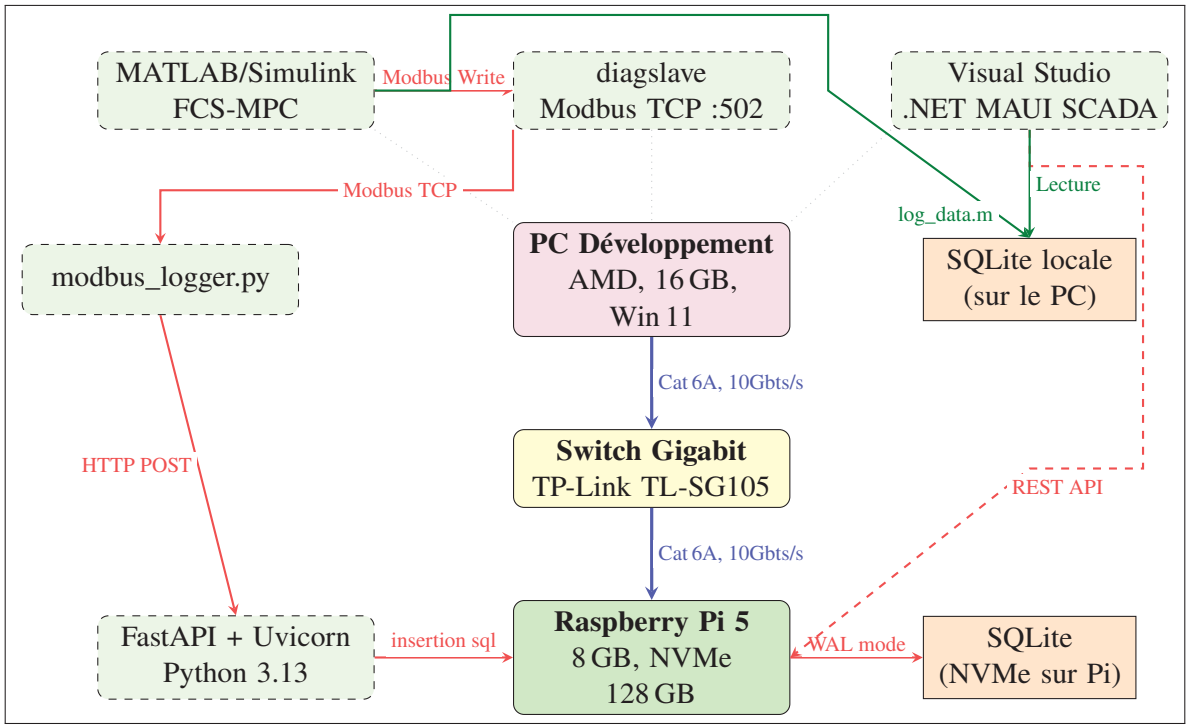

Figure 4.1 Banc de test — matériel et logiciels. **Rouge** = Config. 1 (Modbus distribué), **Vert** = Config. 2 (SQLite directe).

4.1.1 Matériel utilisé

- **PC développement :**
  - CPU : AMD Ryzen 9 5900hx (8 cores, 16 threads,3.3 GHz)

  - GPU : NVIDIA DeForce RTX 3070
  - RAM : 16 GB DDR4
  - SSD : NVMe 1TB GB
  - OS : Windows 11
  - Carte réseau : Ethernet Gigabit + Wi-Fi 6
- **Raspberry Pi 5 Model B :**
  - RAM : 8 GB LPDDR4X
  - Storage : NVMe SSD 128 GB (monté sur /data) + GeekPi PoE Hat
  - OS : Raspberry Pi OS 64-bit (Debian Trixie)
  - Python : 3.13 + FastAPI 0.128 + Uvicorn 0.40
  - Connectivité : Ethernet Gigabit (direct LAN)
- **Réseau :**
  - Switch Gigabit TP-Link TL-SG105 (5 ports, non managé)
  - Connexion LAN directe PC-Pi (pas de routeur Wi-Fi)
  - Latence réseau LAN mesurée : < 1 ms (ping moyen)
  - Cable CAT RJ45 10Gbit/s

4.1.2 Logiciels

- Visual Studio 2022 Community (17.13.2)
- .NET 8.0 SDK
- MATLAB R2024b (tests Modbus futurs)
- DiagSlave 3.4 (serveur Modbus test)
- Python 3.13 + FastAPI 0.128 + Uvicorn 0.40
- Sqlite Studio pour SQLite (audit bases de données)
- Wireshark (analyse trafic Modbus)

4.2 Tests fonctionnels

4.2.1 Configurations d'intégration testées

Les tests fonctionnels sont réalisés selon deux configurations d'intégration complémentaires, décrites en section 2.2.1 :

- **Configuration 1 (Modbus distribué) :** Le flux de données va de MATLAB (bloc Modbus Write) vers diagslave :502 ensuite modbus_logger.py puis transféré au Raspberry Pi FastAPI et enfin récupéré par l'application .NET MAUI (mode Pi). Cette configuration est utilisée principalement pour les tests de communication Modbus (Test 1) et de mode distant de bout en bout (Test 3), validant l'architecture distribuée complète.
- **Configuration 2 (SQLite directe) :** le flux de données va de MATLAB (log_data.m) vers la base de données SQLite locale et récupéré par .NET MAUI (mode Local). Cette configuration est utilisée pour les tests d'historisation (Test 2), de réactivité UI (Test 5) et de graphiques historiques. Cette configuration est plus rapide à mettre en place et s'est avérée adaptée pour valider le fonctionnement des fenêtres glissantes dans les graphiques (ChartsPage), grâce au débit élevé de la journalisation directe.

Pour tous les test nécessitant une injection de données dans la base de données directement, l'application SQLiteStudio est utilisée. L'injection peut s'y faire directement de façon native dans l'interface de l'application ou par exécution de query dans le langage SQL avec l'éditeur embarqué. Les requêtes HTTP sont faites sur le terminal par CURL vers le Pi pour les tests.

4.2.2 Test 1 : Communication Modbus TCP/IP

On cherche à Valider la connexion, les lecture registres, et la gestion d'erreurs.

Le test commence par démarrer DiagSlave sur localhost :502 avec 10 holding registers initialisés ; puis la simulation sur MATLAB. Après cela Il faut connecter l'application MAUI :

ConnectAsync("127.0.0.1", 502) et faire la lecture sur les registres 0-9. Enfin arrêter et recommencer Diagslave pour observer la reconnexion automatique

Résultats :

- Connexion réussie < 200 ms
- Lecture registres : latence 1 ms (Modbus localhost via diagslave)
- Conversion float correcte (validation avec valeurs connues sur scope Matlab)
- Détection déconnexion en < 3 secondes (timeout TCP)
- Reconnexion automatique après 5 secondes (heartbeat) mais arret de la simulation sur Matlab car le bloc ModbusClient nécessite toujours un serveur Modbus actif sur ce port

4.2.3 Test 2 : Historisation SQLite

L'objectif ici est de Mesurer la performance en écriture et en lecture, et la charge que ces opérations ont sur le système.

Ce test consiste dans un premier temps à générer 10 000 enregistrements SensorData synthétiques (valeurs aléatoires) avec une requête SQLite ensuite les insérer avec la stratégie d'agrégation (buffering) + transactions groupées ; mesurer le temps total d'insertion, la charge CPU, et la taille de la base de données et enfin vérifier l'intégrité en s'assurant du compte des entrées insérées. Ce test se fait d'abord avec une insertion des données depuis l'ordinateur vers la base de données SQLite et ensuite du Pi vers la base de données.

Résultats (Windows PC) :

- Insertions uniques (chaque valeur) : 10 000 insertions en 51.6 s = 194 enreg/s
- Insertions groupées (batch=1000) : 10 000 insertions en 0.13 s = 79 622 enreg/s
- Lecture 10 000 lignes : 49.0 ms
- Vérification compte (COUNT) : 10 000 validée
- Aucune perte donnée détectée

Résultats (Raspberry Pi 5, NVMe SSD) :

- Insertions uniques : 18 503 enreg/s
- Insertions groupées (batch=1000) : 82 580 enreg/s
- Température après charge : 42.8°C
- Avec FastAPI HTTP (localhost) : 37 225 enreg/s
- Avec FastAPI HTTP (LAN PC vers Pi) : 24 400 enreg/s

4.2.4 Test 3 : Mode distant Pi end-to-end

L'objectif de ce test est de faire la Validation de la chaîne complète des données de MATLAB vers diagslave vers le Logger ensuite le Pi API enfin l'application MAUI.

Le test commence en démarrant diagslave sur l'adresse localhost :502, puis lancer la simulation MATLAB avec le bloc Modbus Write alimentant diagslave en registres holding contenant les variables du microréseau. Ensuite démarrer FastAPI sur Pi (port 8000), lancer le modbus_logger.py qui lit diagslave, et fait les requêtes POST vers Pi et laisser tourner le système. Enfin se connecter en mode Pi sur l application, afficher le dashboard et vérifier cohérence données.

Résultats :

- Latence HTTP GET 100 rec : 190 ms TTFB (Time To First Bit)
- Latence HTTP GET 10 rec : 88 ms TTFB
- Authentification Basic Auth fonctionnelle (identifiant et mot de passe)
- Dashboard MAUI affiche des valeurs cohérentes
- Aucune corruption données détectée
- Observations : 6 POST échoués (timeouts réseau ponctuels), retry automatique opérationnel

4.2.5 Test 4 : Alarmes et événements

Ce test a pour objectif de vérifier la validité du système d'alarme automatique.

Le test débute par la configuration des seuils dans `AutoAlarmService` (par exemple $V_{bus} <$ 140 V classé comme CRITIQUE), puis par l'injection dans la table `SensorData` d'une valeur

$V_{bus} = 135$ V afin de provoquer le déclenchement. On vérifie ensuite que l'alarme correspondante est générée automatiquement avec la bonne priorité, puis on l'acquitte directement dans l'application. La tension est ensuite restaurée à $V_{bus} = 150$ V pour observer l'auto-résolution de l'alarme. Enfin, l'application est redémarrée pour confirmer la persistance des alarmes et de leur état d'acquittement en base de données.

Résultats :

- Alarme générée en < 500 ms après le seuil franchi
- Priorité CRITICAL correctement assignée
- Acquittement sauvegardé en base de donnée
- Auto-résolution après retour de la valeur normale
- Persistance confirmée après redémarrage
- Filtres par priorité fonctionnels

4.2.6 Test 5 : Performance UI et réactivité

On cherche ici à mesurer la fluidité de l'interface et la latence de mises à jour.

Le dashboard est laissé en lecture Modbus continue pendant que l'on observe la fluidité du rafraîchissement et l'évolution de la consommation mémoire de l'application. Une série de 50 navigations successives entre la page *Tableau de Bord* et la page *Courbes* est ensuite exécutée afin d'évaluer la latence perçue lors des changements de vue et la stabilité de l'interface dans la durée.

Résultats :

- Interface fluide, pas de saccade perceptible lors du rafraîchissement
- Latence de perception : imperceptible (< 100 ms subjectivement)
- Navigation entre pages : instantanée (< 300 ms perçu)
- Mémoire app : ~85 MB (observé dans le Gestionnaire de Taches, pas de fuite détectée)

4.2.7 Test 6 : Exportation des données

L'objectif ici est de valider l'exportation en fichier avec extension .CSV.

Une plage temporelle est sélectionnée dans la page d'historique de l'application, puis l'exportation CSV est déclenchée depuis l'interface utilisateur. Le fichier produit est ensuite ouvert dans Excel pour vérifier le nombre et l'ordre des colonnes exportées, le format des horodatages (ISO 8601), la cohérence des unités physiques (kW, V, etc) et la lisibilité globale des données.

Résultats :

- Export CSV fonctionnel sur Windows
- Fichier .csv valide, ouvrable dans Excel
- 8 colonnes exportées (Timestamp, 7 grandeurs physiques en kW,V,etc)
- Formatage des colonnes correct
- Timestamps sous le format ISO 8601 (YYYY-MM-DDTHH :mm :ssZ)

4.3 Validation des résultats de simulation FCS-MPC

Les résultats de simulation Simulink ont été validés. Deux scénarios critiques issus des travaux de Dubuisson (2023) ont été reproduits et analysés.

4.3.1 Régulation tension bus DC à 150 V

Conditions de test :

- Éolienne active (ω_m = 188,5 rad/s), PV actif (irradiance 1000 W/m^2)
- Générateur diesel connecté, BESS actif pour équilibrage de puissance
- Charge AC connectée

Résultats : La tension du bus DC est stable autour de 150 V comme le montre la figure 4.2 avec une variation très faible (entre 149,95 V et 150,15 V), confirmant l'efficacité de la régulation par la BESS via FCS-MPC. Le partage de puissance entre PV, éolienne et GD fonctionne comme

prévu, avec la BESS qui charge lorsque $P_{PV} + P_{WT} + P_{DG} > P_{Charge}$ et décharge dans le cas inverse.

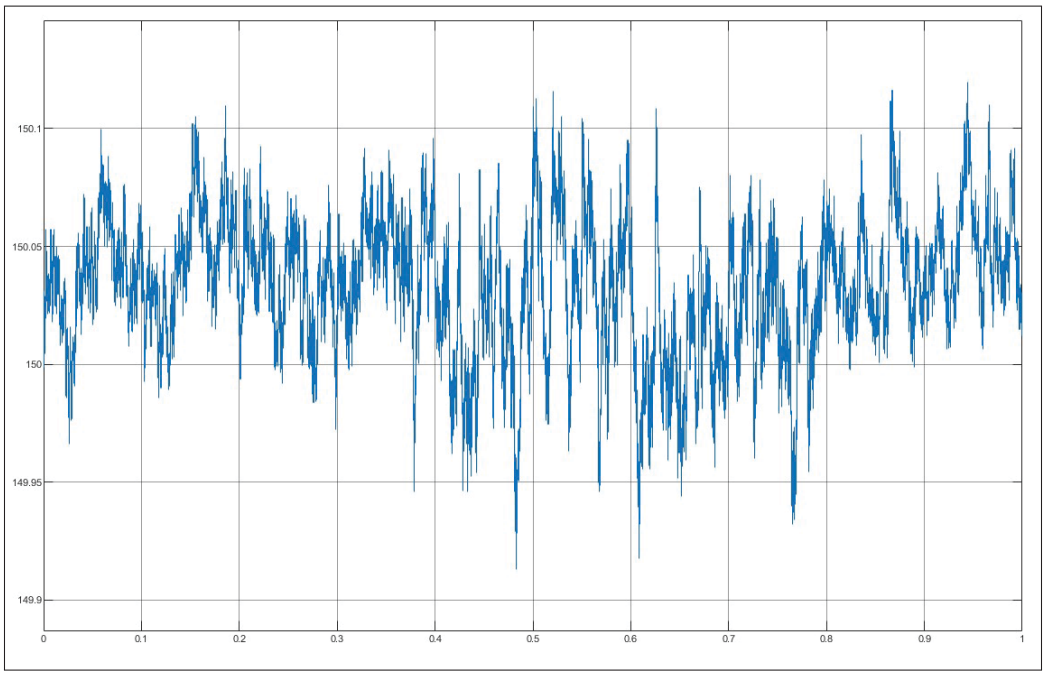

Figure 4.2 Régulation de la tension du bus DC en fonctionnement normal avec toutes les sources actives.

4.3.2 Scénario 1 : Irradiance PV variable avec BESS

Conditions de test : Éolienne et GD arrêtés, PV actif avec irradiance variable, BESS actif.

Temps de simulation :

- $t < 0{,}2$ s : Irradiance faible (200 W/m^2), charge connectée, BESS décharge
- $t = 0{,}2$ s : Irradiance augmente à 600 W/m^2, I_{PV} augmente
- $t = 0{,}4$ s : Irradiance = 1000 W/m^2 (max), I_{PV} maximale
- $t = 0{,}5$ s : Déconnexion charge ($I_L = 0$), BESS passe en mode charge ($I_B < 0$)
- $t = 0{,}7$ s : Reconnexion charge, I_L retourne
- $t = 0{,}8$ s : Irradiance diminue à 400 W/m^2, I_{PV} diminue, BESS compense

Résultats : V_{DC} stable ≈ 150 V durant toutes les transitions comme le montre la figure 4.3. Le courant PV suit fidèlement le profil d'irradiance et la BESS s'adapte automatiquement au bilan énergétique.

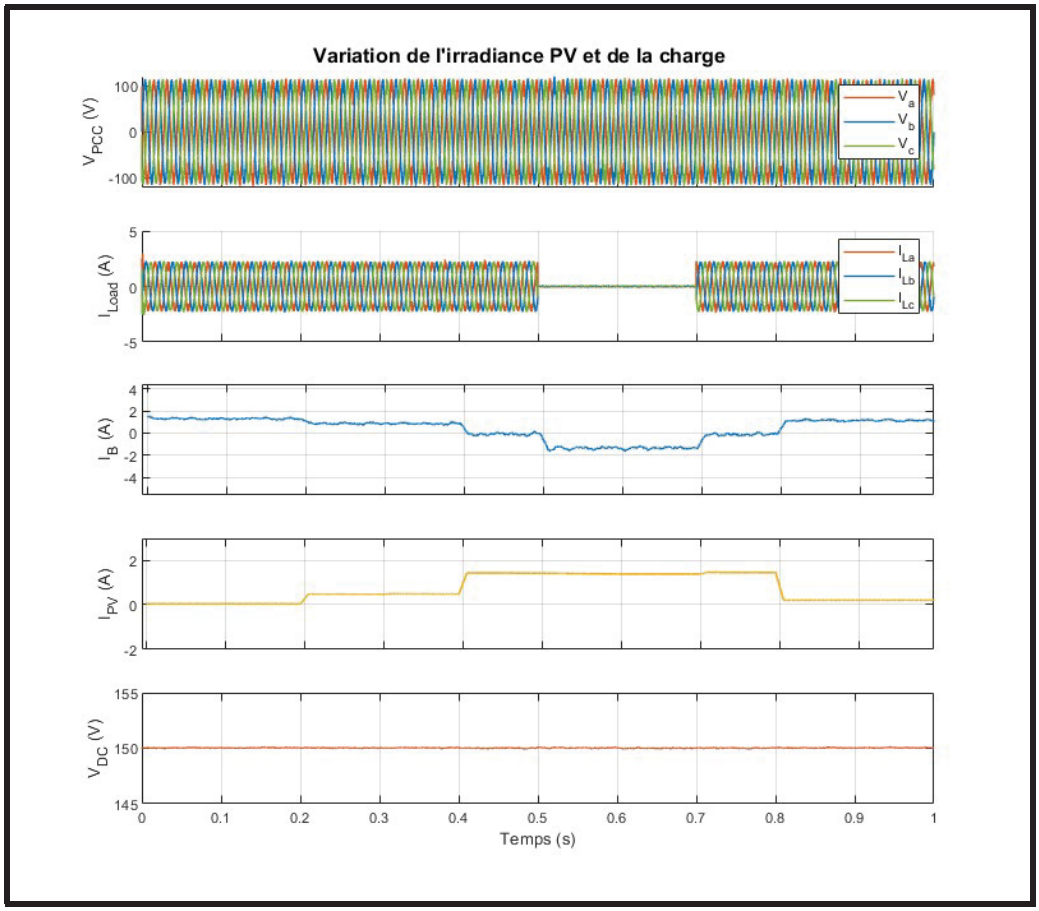

Figure 4.3 Résultats de simulation pour le scénario test 1 : irradiance PV variable avec gestion BESS adaptative.

4.3.3 Scénario 2 : Connexion/déconnexion GD avec BESS

Conditions de test : Éolienne et PV arrêtés, GD avec connexion/déconnexion dynamique, BESS actif.

Temps de simulation :

- $t < 0{,}4$ s : BESS seul alimente la charge ($I_B > 0$ décharge, $I_{DG} = 0$)

- t = 0,4 s : Connexion GD, I_{DG} augmente jusqu'à ≈ 6 A
- t = 0,5 s : Déconnexion charge ($I_L = 0$), GD continue de fournir
- t = 0,7 s : Reconnexion charge, I_L retourne
- t = 0,8 s : Déconnexion GD ($I_{DG} = 0$)
- t > 0,8 s : BESS seul alimente à nouveau la charge

Résultats : V_{DC} stable ≈ 150 V durant toutes les transitions de connexion/déconnexion du GD. La BESS s'adapte automatiquement en augmentant sa décharge lorsque le GD est déconnecté et en la réduisant lorsque le GD est actif comme illustré dans la figure 4.4.

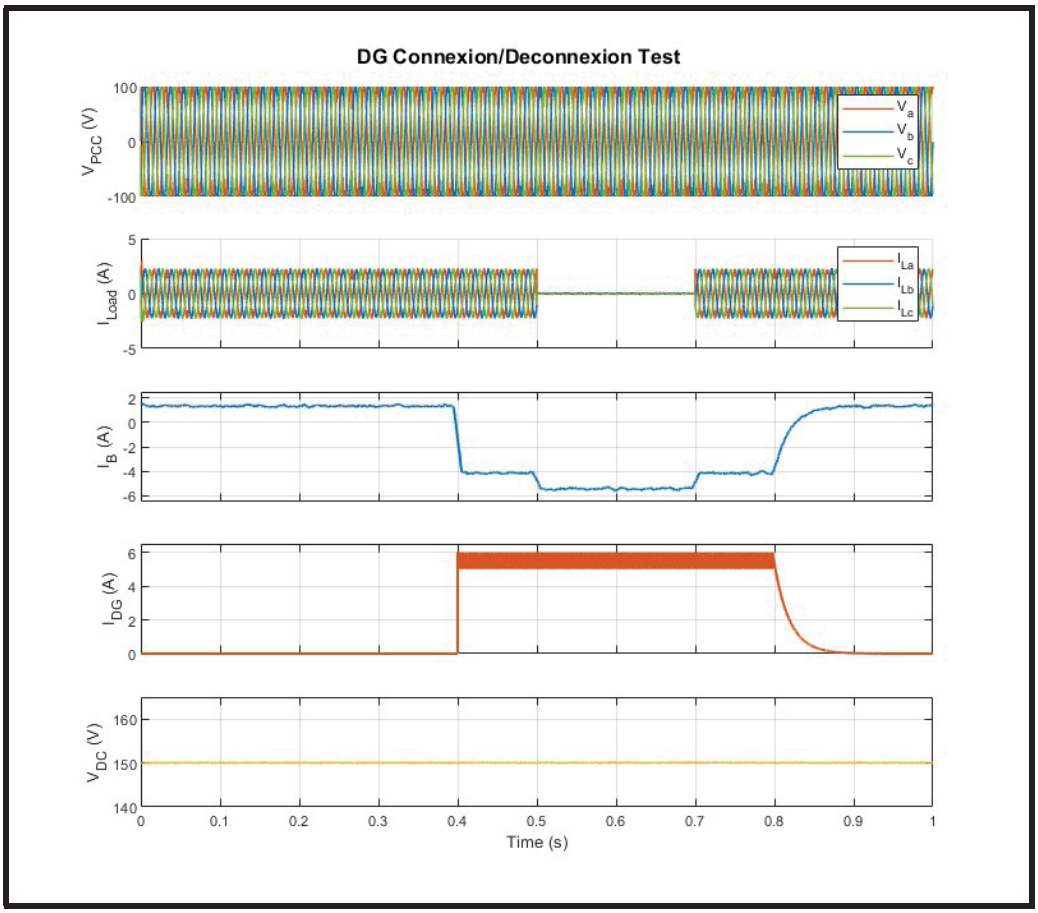

Figure 4.4 Résultats de simulation pour le scénario test 2 : transitions connexion/déconnexion du générateur diesel.

4.3.4 Validation communication SCADA

Les données de simulation ont été transmises avec succès au SCADA via les deux configurations de test. La figure 4.5 montre les variables du bus DC(tension, courant, puissance) capturées dans l'interface de test, et la figure 4.6 montre les tensions et courants triphasés.

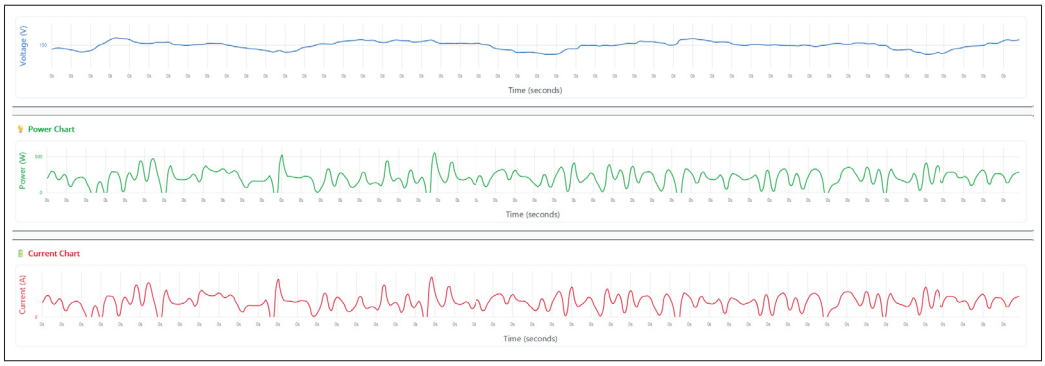

Figure 4.5 Résultats dans la page test SCADA : Pbus, Vbus, Ibus (graphiques LiveCharts).

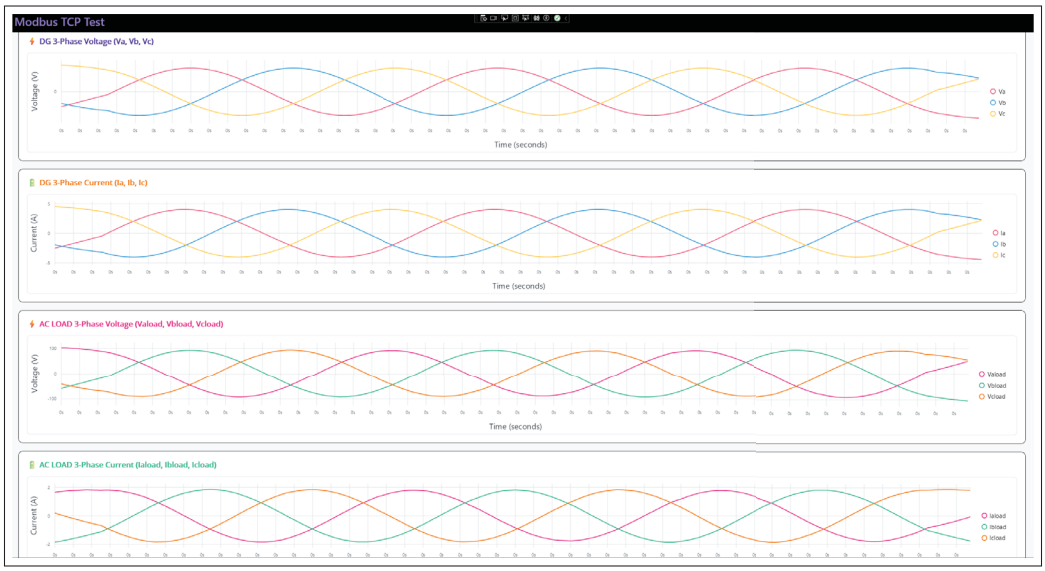

Figure 4.6 Résultats dans la page test SCADA : tensions et courants triphasés (VabcLoad, IabcLoad, VabcDG, IabcDG).

4.4 Récapitulatif mesures de performance quantitatives

4.4.1 Performance d'historisation

Ce tableau résume les performances observées pour les insertions de données au niveau des bases de données.

Tableau 4.1 Débit écriture SQLite mesuré (10 000 insertions, 30 colonnes)

Configuration	Débit (enreg/s)	Note
Windows, Insertion individuelles	194	très lent
Windows, Insertions par lot (batch=1000)	79 622	transaction groupée
Pi 5 NVMe, Insertion individuelles	18 503	NVMe rapide
Pi 5 NVMe, Insertions par lot (batch=1000)	82 580	transaction groupée
Pi 5, via FastAPI+SQLite (localhost)	37 225	HTTP
Pi 5, via FastAPI+SQLite (LAN)	24 400	PC vers Pi

4.4.2 Fiabilité et disponibilité

- Service systemd `scada-api` : actif, 0 redémarrage automatique observé (NRestarts=0)
- Uptime : test longue durée (≥ 72h) non complété dans cette itération
- Taux reconnexion après perte réseau : à mesurer

4.5 Analyse comparative coût-bénéfice

4.5.1 Coûts de développement

Le tableau 4.2 montre les estimations pour le coût de développement de l'application comparé au solutions commerciales sur le marché. Cet exercice consiste à établir une plage sur chaque catégorie considérée pour souligner la pertinence de la solution développée.

Tableau 4.2 Estimations : coûts prototype vs. SCADA commercial

Poste	Prototype(estimations)	SCADA Commercial
Licences logicielles	0$ (open-source)	15 000$ - 50 000$
Développement (300h @ 50$/h)	15 000$	Inclus
Formation	500$	5 000$ - 10 000$
Matériel (Pi 5 + NVMe + switch)	200$	0$ (inclus)
Maintenance annuelle	1 000$	3 000$ - 10 000$
Total première année	**16 600$**	**23 000$ - 70 000$**
Total 5 ans	**20 600$**	**35 000$ - 110 000$**

4.5.2 Comparaison des fonctionnalités

Le prototype, bien que dans un stade précoce et n'ayant pas encore le nombre d'options que les logiciels commerciaux, arrive à assurer des fonctionnalités demandées dans le milieu industriel pour un fonctionnement de principe (tableau 4.3).

Tableau 4.3 Couverture de fonctionnalités

Fonctionnalité	Prototype	SCADA Ignition
Communication Modbus TCP	oui	oui
OPC UA	non	oui
Historisation	oui	oui
Graphiques historiques	oui	oui
Dashboard customisable	non	oui
Alarmes + acquittement	oui	oui
Export CSV	oui	oui
Multi-plateforme natif	oui	oui
Scripting Python/C#	partiel	inconnu
Redondance serveurs	non	oui
Audit cybersécurité IEC 62443	non	oui
Support professionnel	non	oui

4.6 Défis et Mitigation

4.6.1 Intégration FCS-MPC haute fréquence

La simulation FCS-MPC, opérant à $T_s = 50\,\mu$s (20 kHz), cette fréquence très élevée ne s'avère pas compatible avec le scada (1 a 10 Hz en général)

Solution : Faire un sous-échantillonnage $\times 20$ qui la réduit à 1 kHz puis faire des lots de 1000 lignes de 29 variables, envoyer le lot en une transaction.

4.6.2 Performance écritures SQLite sur Pi

Les performance d'écriture avec l'approche individuelle (1 insertion/mesure) montre une saturation au niveau du Pi.

Solutions testées :

1. Transactions groupées : amélioration
2. WAL mode : amélioration additionnelle
3. Background thread : découplage UI totalement

La solution finale retenue est une combinaison des trois.

4.6.3 Conversion Modbus float Big Endian

MATLAB écrit les réels 32-bit en Big Endian, la lecture simple produisait des valeurs erronées.

Solution : Inversion manuelle des bytes avec BitConverter, validation avec valeurs connues.

4.7 Limitations de l'étude

4.7.1 Limitations techniques

1. **Protocole unique :** Seul Modbus TCP supporté. Extension OPC UA/MQTT nécessiterait développement significatif.
2. **Pas de contrôle temps réel :** Latences 10-100 ms inadéquates pour boucles contrôle rapides. Prototype limité à la supervision.
3. **Cybersécurité basique :** Authentification Basic Auth
4. **Redondance absente :** Single point of failure (serveur Pi). SCADA industriels offrent une redondance des serveurs.
5. **Scalabilité non testée :** Validation limitée à 1 microréseau simulé. Comportement avec 10+ sources inconnu.
6. **Tests Android non réalisés sur device physique :** L'application compile pour Android (`net8.0-android`) mais n'a pas été testée sur un appareil physique dans cette itération. Tests Android prévus pour version future.

4.7.2 Limitations méthodologiques

1. **Absence validation microréseau réel :** Tests avec serveurs Modbus simulés uniquement. Validation sur hardware physique (panneaux PV, batteries, convertisseurs) reste nécessaire.
2. **Pas de comparaison directe SCADA commercial :** Analyse coût/fonctionnalités basée sur spécifications publiques, estimations et citations.
3. **Tests longue durée limités :** 72h maximum. Comportement sur semaines/mois (dégradations mémoire, corruption DB) non évalué.
4. **Charge utilisateur unique :** Tests single-user. Comportement multi-utilisateurs simultanés (mobile + PC) non testé.

Ce chapitre documente les tests fonctionnels et de mesures de performance du prototype. Les résultats montrent que la communication Modbus, l'API REST et l'historisation SQLite sur

Raspberry Pi fonctionnent pour un système SCADA de supervision, et que l'interface .NET MAUI prend et utilise les données de manière cohérente.

CONCLUSION ET RECOMMANDATIONS

Ce travail s'interesse à la supervision d'un microréseau DC dans le cadre de la recherche académique dans un laboratoire universitaire, sans recourir aux plateformes SCADA industrielles. Le prototype conçu à cet effet, qui intègre une simulation MATLAB/Simulink (cinq convertisseurs FCS-MPC d'après Dubuisson (2023)) et une application .NET MAUI communiquant par Modbus TCP/IP, HTTP FastAPI s'est avérée techniquement viable, quoique partielle.

Le cadre de développement .NET MAUI a permis de construire une application multi plateforme pour faire la surveillance du réseau. L'architecture conçue avec le serveur FastAPI sur Raspberry Pi5 a fourni des résultats acceptables. Les stratégies d'optimisation du traitement des données au niveau du SCADA ont permis d'assurer une fluidité d'interface graphique ainsi qu'une rapidité accrue dans le traitement des données.

Du côté de la simulation, l'intégration du sous-échantillonnage $\times 20$ et du groupement par 1000 échantillons pour exportation vers SQLite par la fonction `log_data.m` a permis de résoudre le problème d'échelle entre la période de contrôle ($T_s = 50\,\mu$s) et la fréquence SCADA (1–10 Hz) mais aussi a permis de réduire le temps de simulation et la taille de la base de donnée.

Perspectives de recherche future

Extensions court terme (6-12 mois) Au cours de cette periode, il serait possible de pouvoir dans un premier temps élargir les protocoles supportés (OPC UA client, MQTT pub, DNP3), d'augmenter le nombre de fonctionnalités avancées (Templates dashboards configurables, Scripting Python embarqué, Rapports automatiques PDF, Notifications push), de renforcer la cybersécurité (Authentification multi-facteurs, Roles-based access control, Audit logs conformité)

Validation Hardware-in-the-Loop (long terme 12-24 mois)

Objectif : Valider le système complet (SCADA + FCS-MPC + microréseau) sur plateforme temps réel.

Plateforme HiL envisagée :

- OPAL-RT OP4510 ou OP5707 (simulateur temps réel) Gonzalez-Candelario, Darbali-Zamora *et al.* (2023)
- Modèle Simulink microréseau compilé pour temps réel Villalón *et al.* (2023)
- FCS-MPC implémenté en C, déployé sur OPAL-RT
- SCADA MAUI connecté via Modbus TCP pour supervision Beus *et al.* (2022); Cintuglu, Elsayed *et al.* (2015)

ANNEXE I

CONFIGURATIONS ET PARAMÈTRES

1. Configuration .NET MAUI

1.1 Fichier MauiApp1.csproj

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>net8.0-windows10.0.19041.0;
                      net8.0-android;net8.0-ios</TargetFrameworks>
    <OutputType>Exe</OutputType>
    <RootNamespace>MauiApp1</RootNamespace>
    <UseMaui>true</UseMaui>
    <SingleProject>true</SingleProject>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <ApplicationTitle>Microgrid SCADA</ApplicationTitle>
    <ApplicationId>com.ets.microgridscada</ApplicationId>
    <ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
    <ApplicationVersion>1</ApplicationVersion>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="NModbus" Version="3.0.81" />
    <PackageReference Include="sqlite-net-pcl" Version="1.9.172" />
    <PackageReference Include="LiveChartsCore.SkiaSharpView.Maui"
                      Version="2.0.0-rc2" />
    <PackageReference Include="SkiaSharp.Views.Maui.Controls"
```

```
                    Version="2.88.8" />
    <PackageReference Include="SQLitePCLRaw.bundle_green"
                    Version="2.1.10" />
  </ItemGroup>
</Project>
```

2. Configuration Raspberry Pi

2.1 Installation Python dependencies

```
sudo apt update
sudo apt install python3-pip python3-venv
python3 -m venv ~/scada_env
source ~/scada_env/bin/activate
pip install fastapi uvicorn pymodbus aiosqlite python-multipart
```

2.2 Configuration systemd service

Fichier `/etc/systemd/system/scada-api.service` :

```
[Unit]
Description=SCADA Microgrid FastAPI Service
After=network.target

[Service]
User=pi
WorkingDirectory=/data/scada
ExecStart=/usr/bin/python3 -m uvicorn temp_main:app
          --host 0.0.0.0 --port 8000
Restart=always
```

```
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Activation :

```
sudo systemctl daemon-reload
sudo systemctl enable microgrid-api
sudo systemctl start microgrid-api
sudo systemctl status microgrid-api
```

ANNEXE II

EXTRAITS CODE SOURCE

1. Service Modbus thread-safe

Fichier : Services/ModbusService.cs

```
using NModbus;
using System.Net.Sockets;

namespace MauiApp1.Services
{
  public class ModbusService : IDisposable
  {
    private TcpClient? _tcpClient;
    private IModbusMaster? _modbusMaster;
    private readonly SemaphoreSlim _modbusLock = new(1, 1);

    public async Task<float[]> ReadHoldingRegistersAsFloatAsync(
      byte slaveId, ushort startAddress, ushort floatCount)
    {
      await _modbusLock.WaitAsync();
      try
      {
        if (!IsConnected)
          throw new InvalidOperationException("Not connected");

        ushort regCount = (ushort)(floatCount * 2);
        var registers = await _modbusMaster!
          .ReadHoldingRegistersAsync(slaveId, startAddress, regCount);
```

```
      float[] floats = new float[floatCount];
      for (int i = 0; i < floatCount; i++)
      {
        byte[] bytes = new byte[4];
        byte[] highWord = BitConverter.GetBytes(registers[i*2]);
        byte[] lowWord = BitConverter.GetBytes(registers[i*2+1]);

        // Big Endian
        bytes[0] = highWord[1];
        bytes[1] = highWord[0];
        bytes[2] = lowWord[1];
        bytes[3] = lowWord[0];

        floats[i] = BitConverter.ToSingle(bytes, 0);
      }
      return floats;
    }
    finally
    {
      _modbusLock.Release();
    }
  }

  public void Dispose()
  {
    _modbusLock?.Dispose();
    _modbusMaster?.Dispose();
    _tcpClient?.Dispose();
```

```
    }
  }
}
```

2. Historisation optimisée Raspberry Pi

Fichier : Services/SqliteDataService.cs (extraits)

```
public class SqliteDataService : IDataService
{
  private SQLiteAsyncConnection? _database;
  private readonly List<SensorData> _buffer = new();
  private const int BUFFER_SIZE = 50;
  private readonly BlockingCollection<SensorData> _writeQueue = new();

  private async Task InitAsync()
  {
    _database = new SQLiteAsyncConnection(_dbPath);


    await _database.ExecuteAsync("PRAGMA journal_mode=WAL");
    await _database.ExecuteAsync("PRAGMA synchronous=NORMAL");
    await _database.ExecuteAsync("PRAGMA cache_size=10000");

    await _database.CreateTableAsync<SensorData>();

    // Background writer thread
    Task.Run(BackgroundWriterAsync);
  }
```

```
public void QueueInsertAsync(SensorData data)
{
  _writeQueue.Add(data);
}

private async Task BackgroundWriterAsync()
{
  foreach (var data in _writeQueue.GetConsumingEnumerable())
  {
    _buffer.Add(data);

    if (_buffer.Count >= BUFFER_SIZE)
    {
      await FlushBufferAsync();
    }
  }
}

private async Task FlushBufferAsync()
{
  if (_buffer.Count == 0) return;

  var toInsert = _buffer.ToList();
  _buffer.Clear();

  await Database.RunInTransactionAsync(tran => {
    foreach (var data in toInsert)
    {
      tran.Insert(data);
```

```
        }
      });
    }
  }
```

ANNEXE III

CAPTURES D'ÉCRAN

1. Application Windows

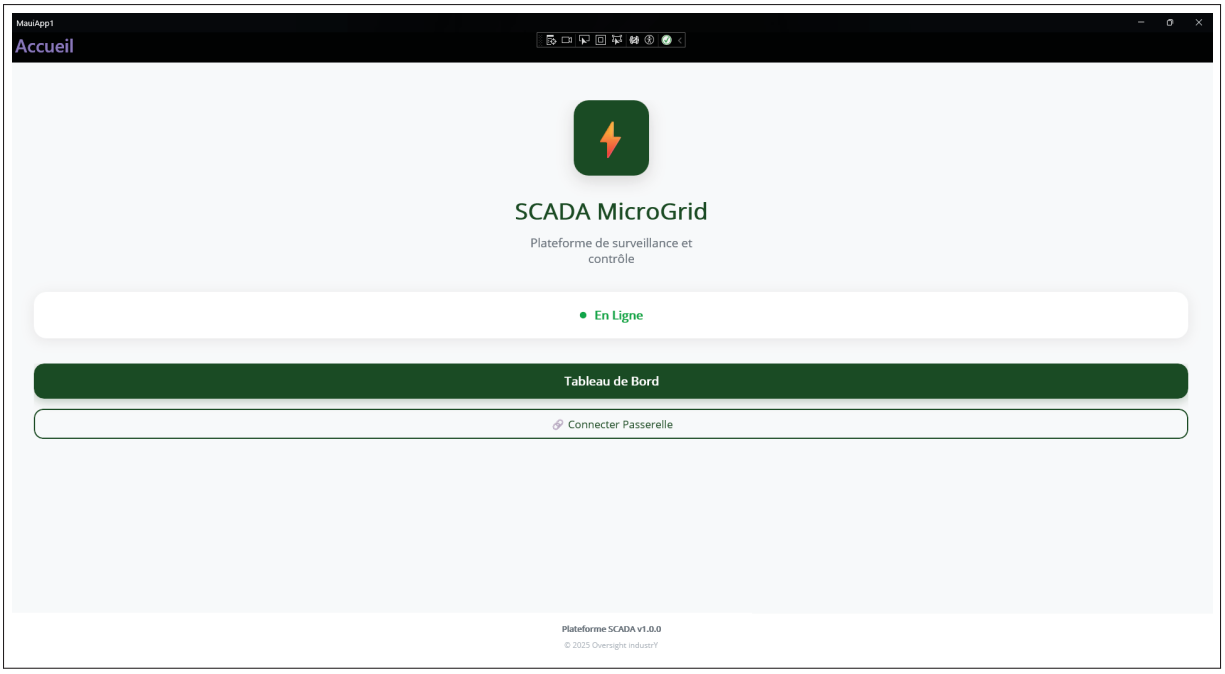

Figure-A III-1 Page d'Accueil SCADA avec option de connexion.

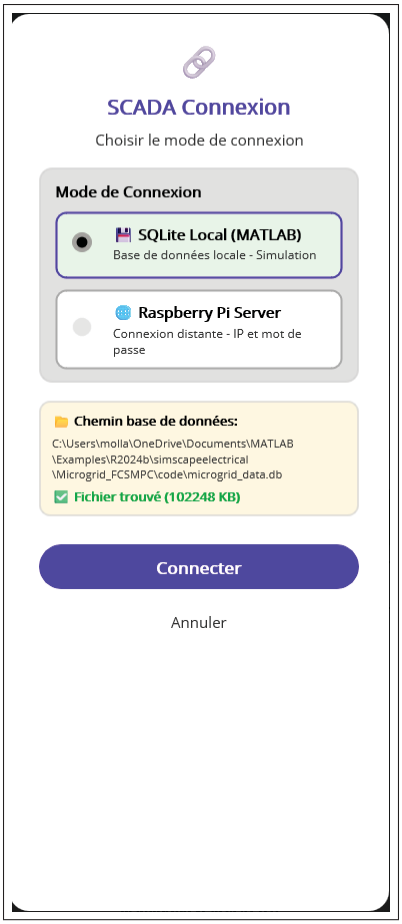

Figure-A III-2 Page de connexion.

2. Pages de supervision et diagnostic

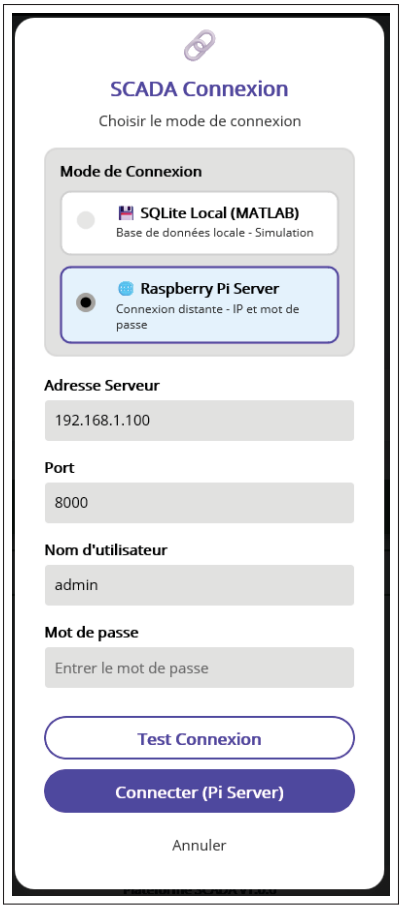

Figure-A III-3 Page de connexion sur Raspberry Pi.

3. Vue d'ensemble Simulink

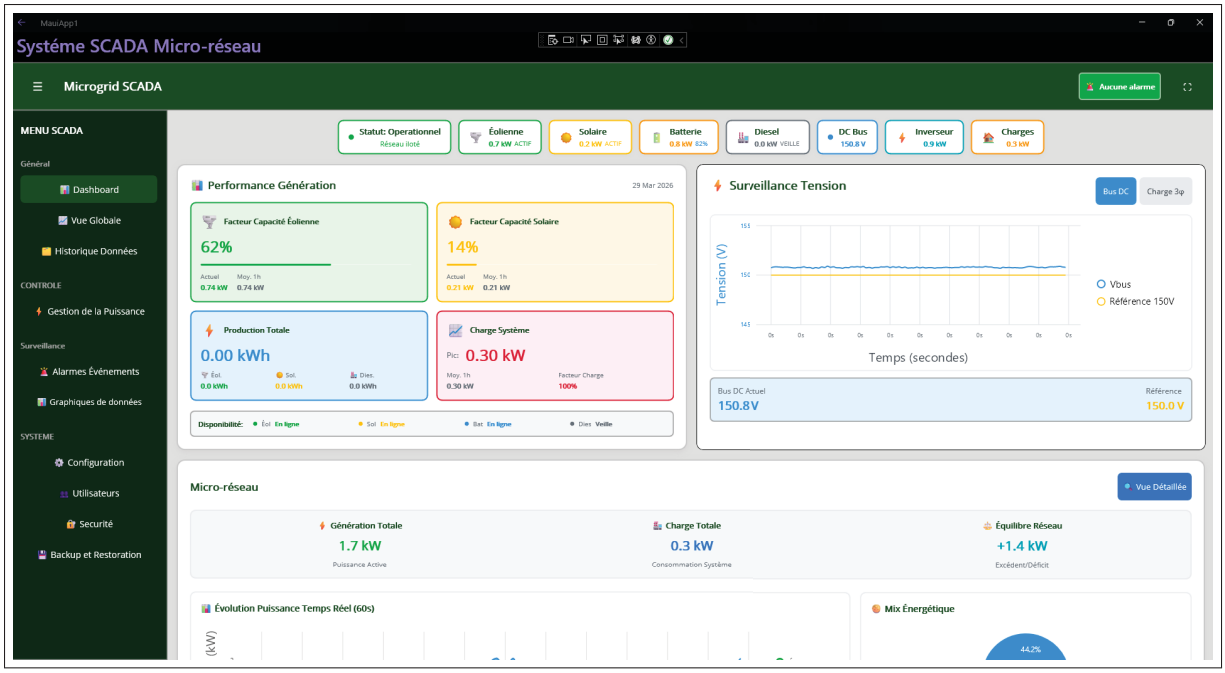

Figure-A III-4 Page principale du dashboard SCADA avec menu de navigation latéral et indicateurs de performance du microréseau.

4. Autres Applications utilisées

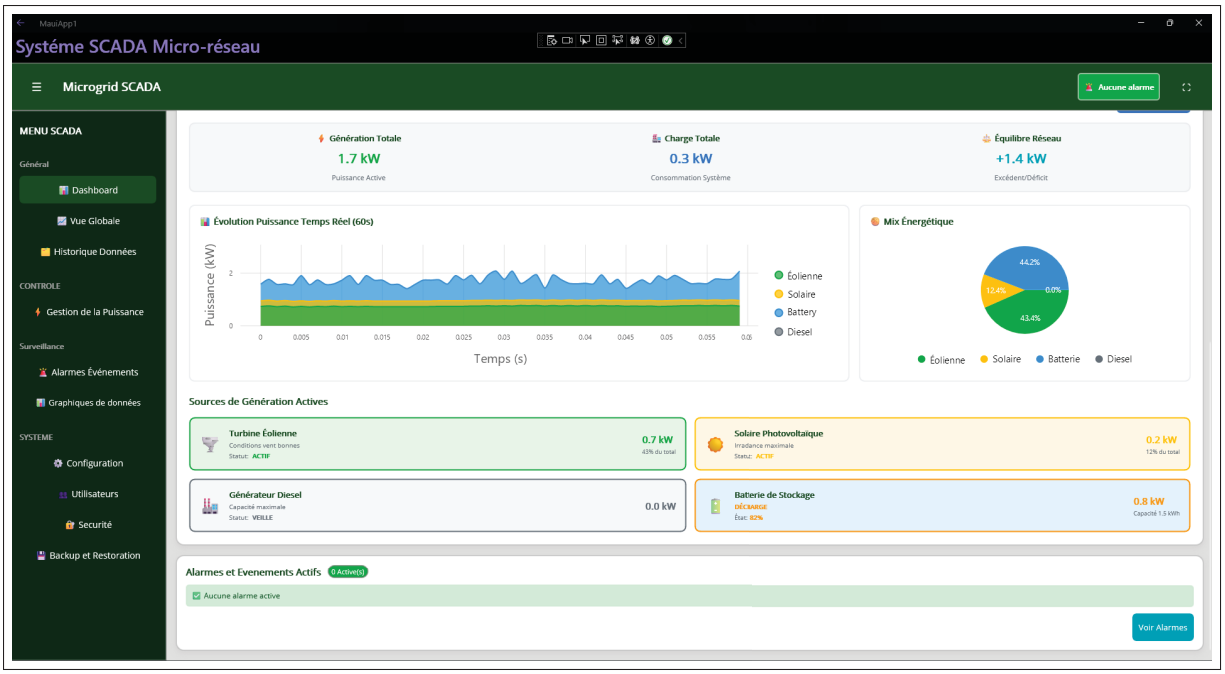

Figure-A III-5 Page principale du dashboard SCADA (suite).

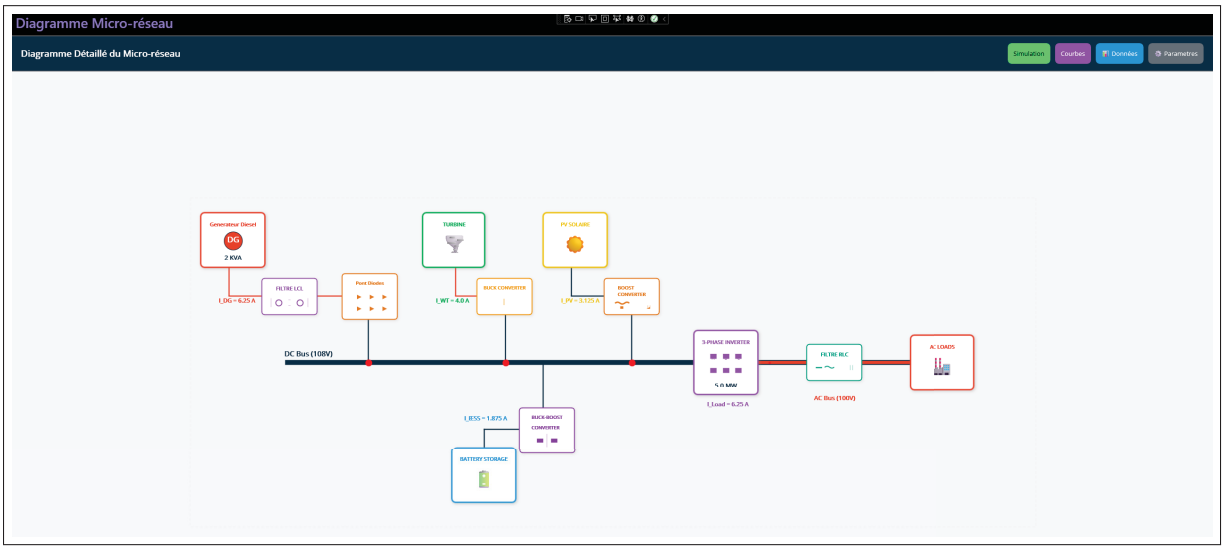

Figure-A III-6 Vue détaillée et interactive du microréseau avec éléments cliquables pour accéder aux informations de chaque composant.

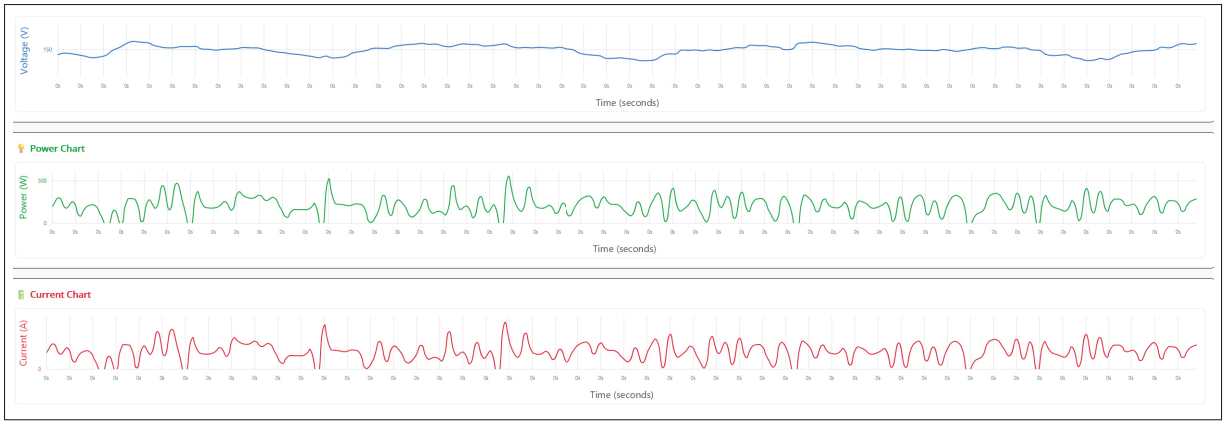

Figure-A III-7 Graphiques temps réel des variables du bus DC (Pbus, Vbus, Ibus) capturées via communication Modbus TCP.

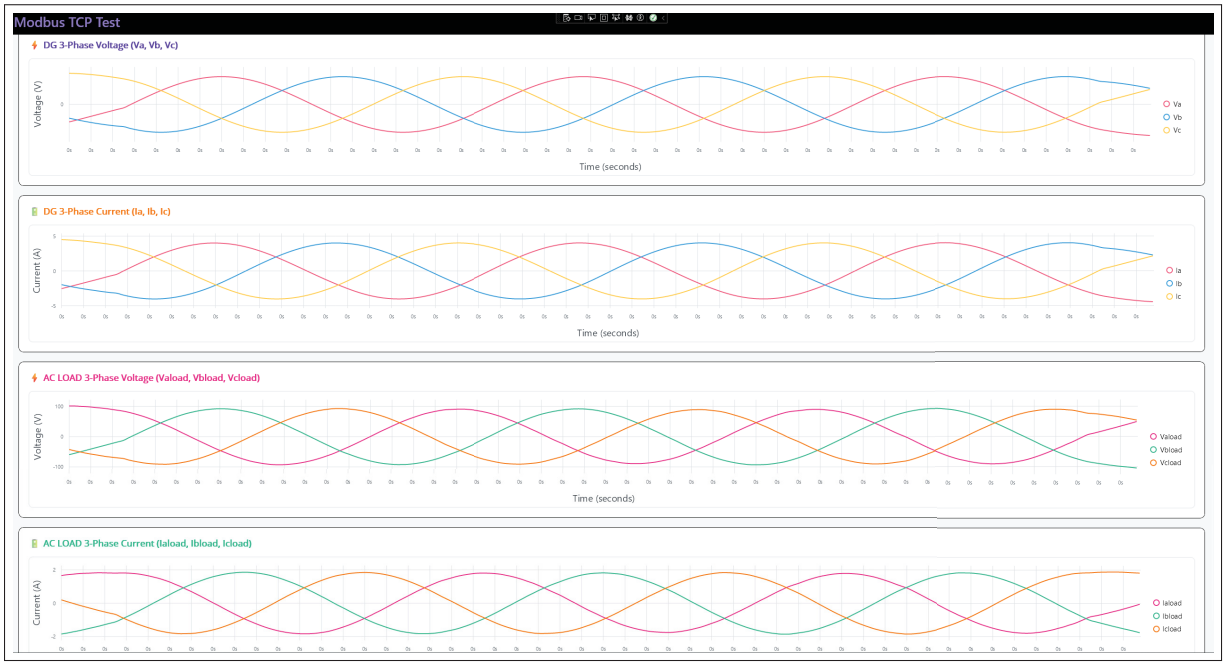

Figure-A III-8 Graphiques temps réel des tensions et courants triphasés (VabcLoad, IabcLoad, VabcDG, IabcDG).

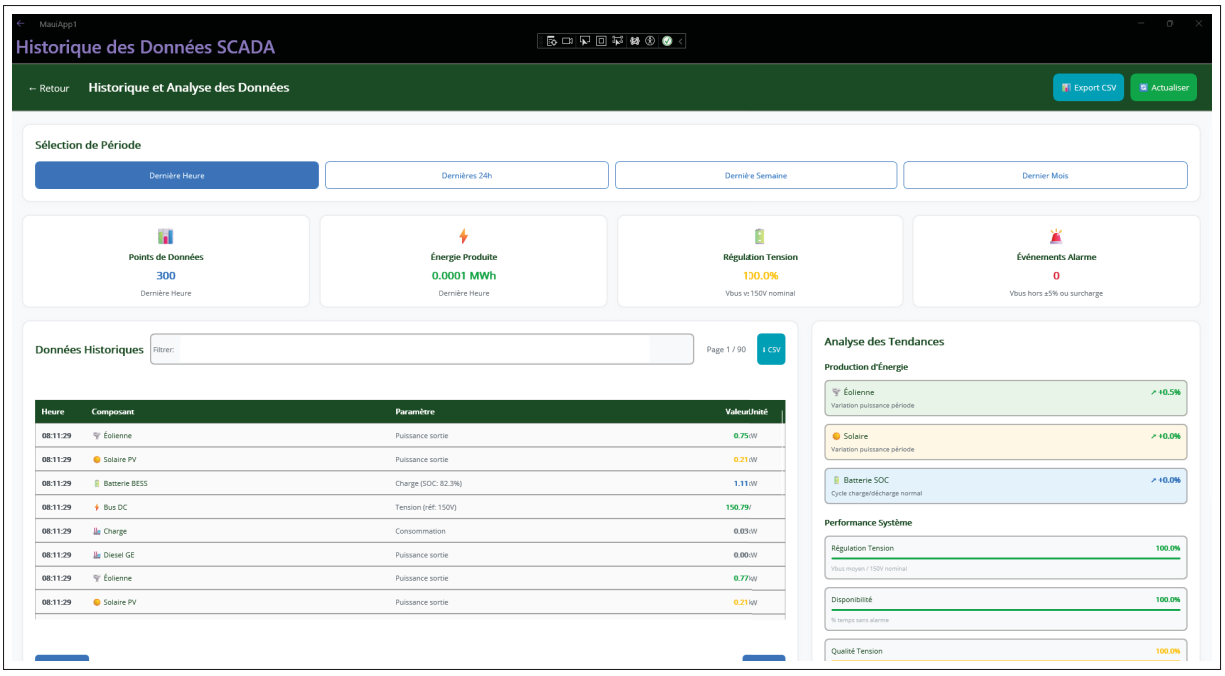

Figure-A III-9 Page d'historique des données (DataHistorianPage) avec consultation de la base SQLite et filtres temporels.

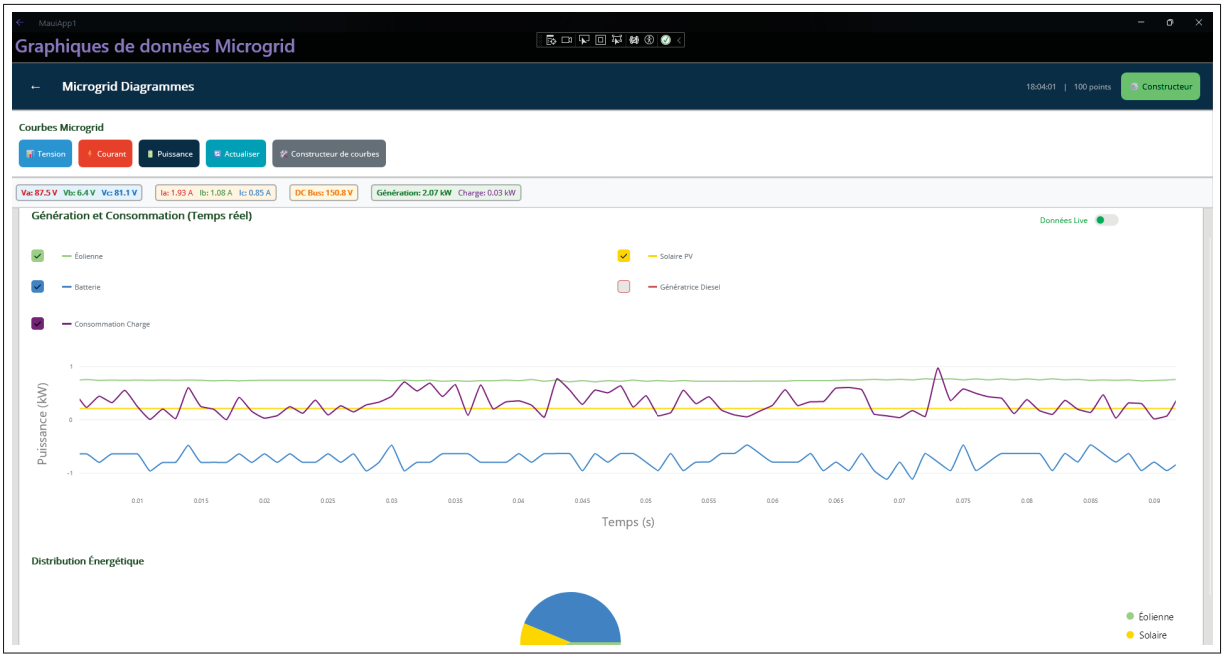

Figure-A III-10 Page de visualisation des courbes de puissance et répartition énergétique du microréseau.

Figure-A III-11 Page de visualisation du constructeur de courbes de puissance et répartition énergétique du microréseau.

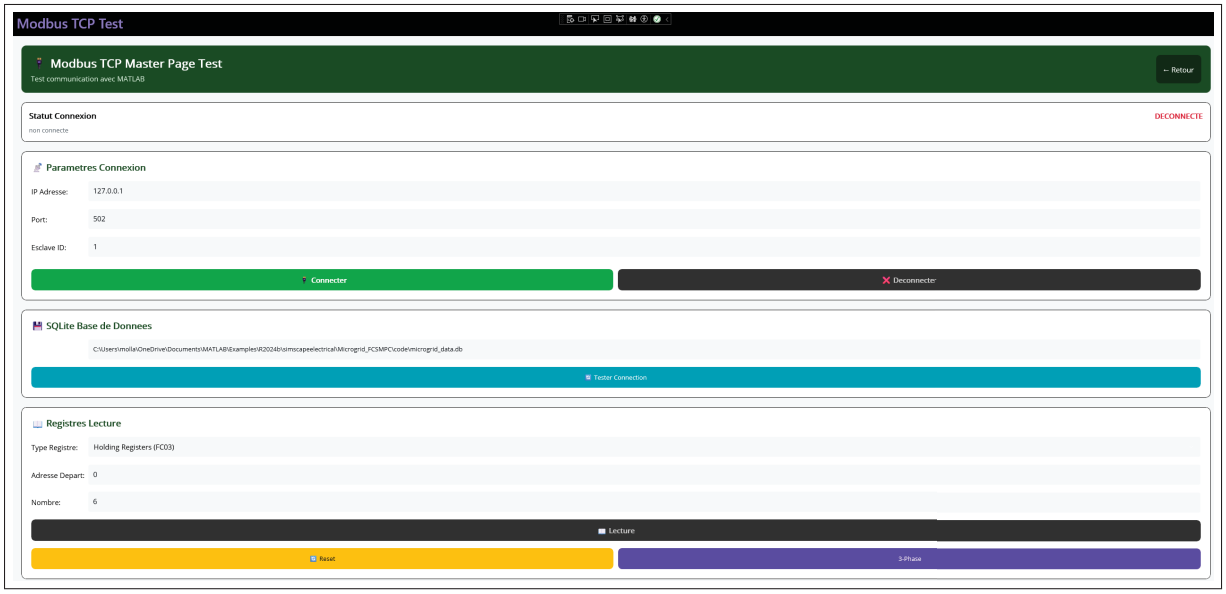

Figure-A III-12 Page de test Modbus (ModbusTestPage) : configuration de la connexion, paramètres de lecture et diagnostic de la communication.

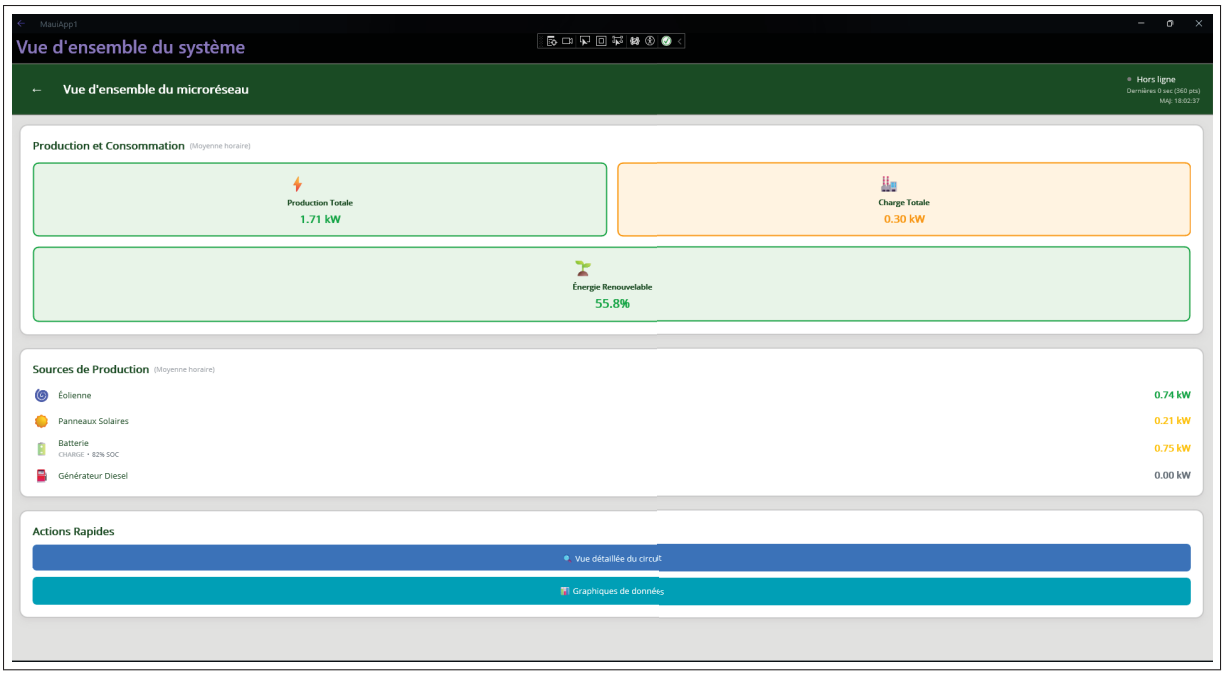

Figure-A III-13 Page de gestion de puissance et répartition énergétique entre les sources du microréseau.

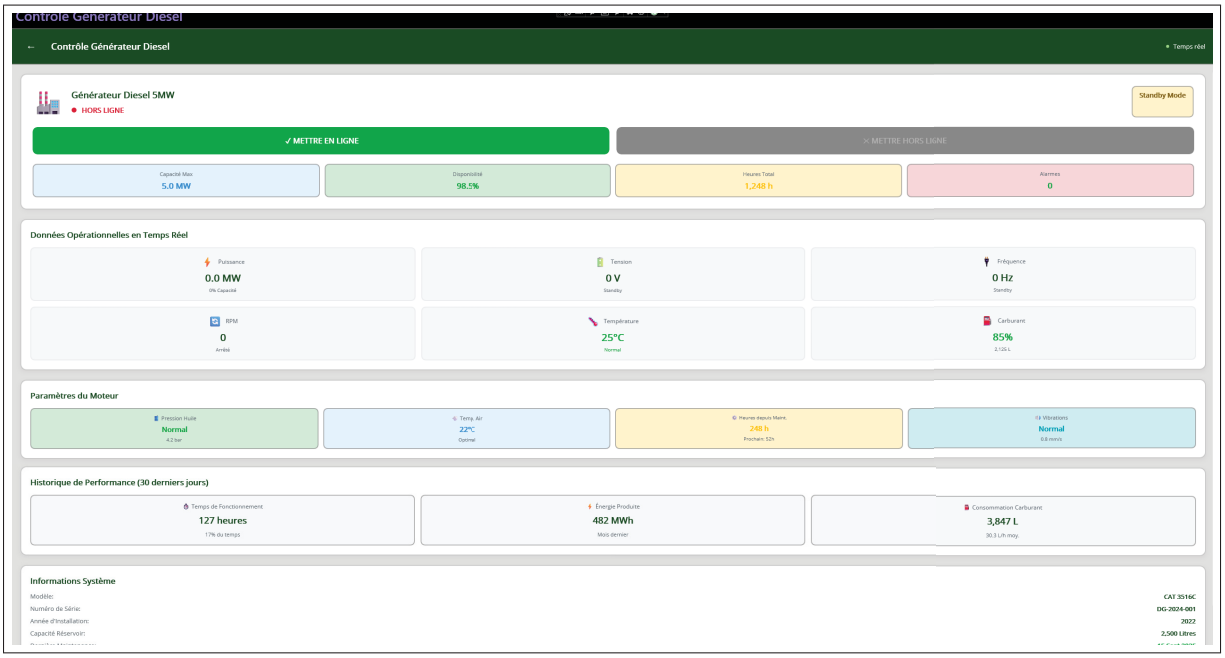

Figure-A III-14 Page de contrôle du générateur diesel : émissions CO_2, consommation carburant et graphiques comparatifs.

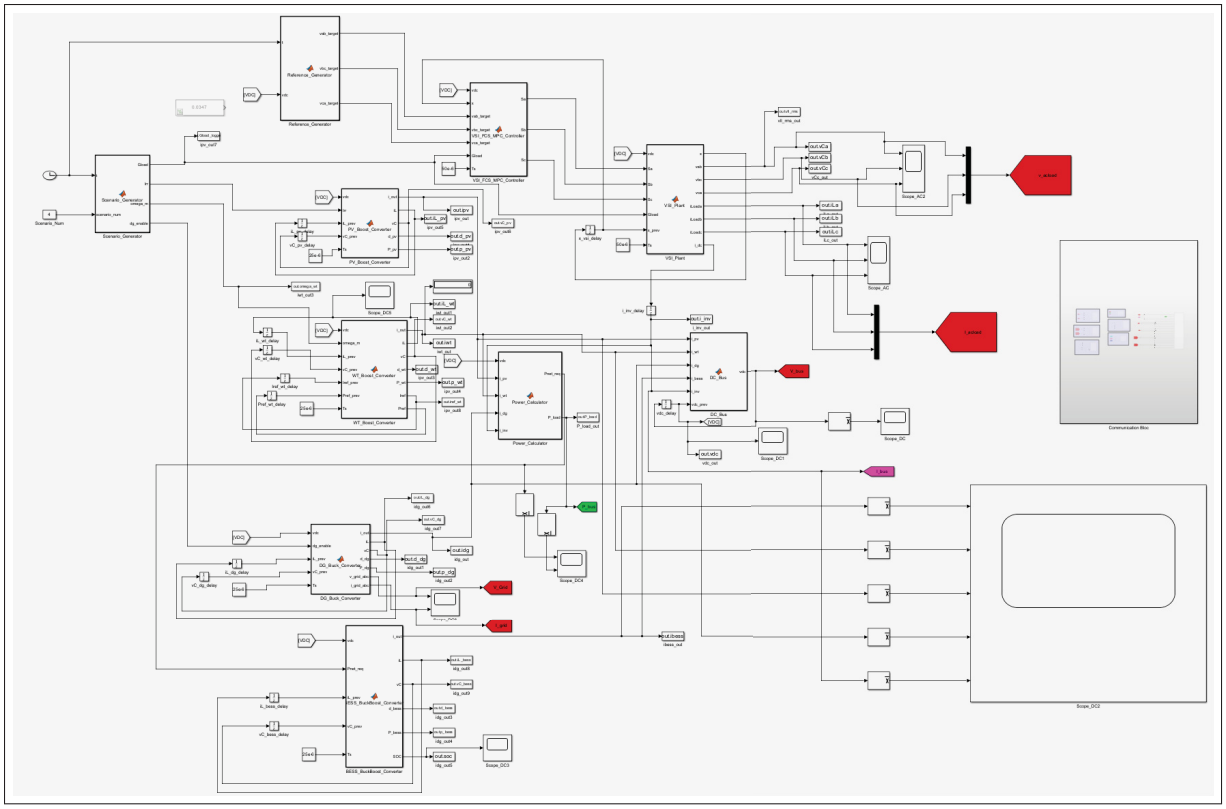

Figure-A III-15 Vue d'ensemble schématique du système microréseau dans Simulink.

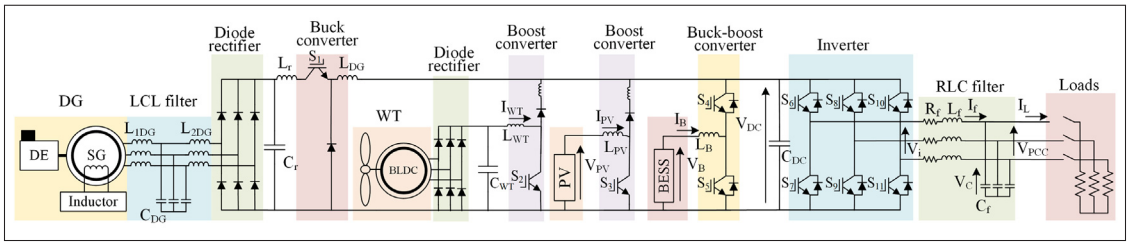

Figure-A III-16 Schéma de référence du microréseau DC multi-sources (topologie Dubuisson).

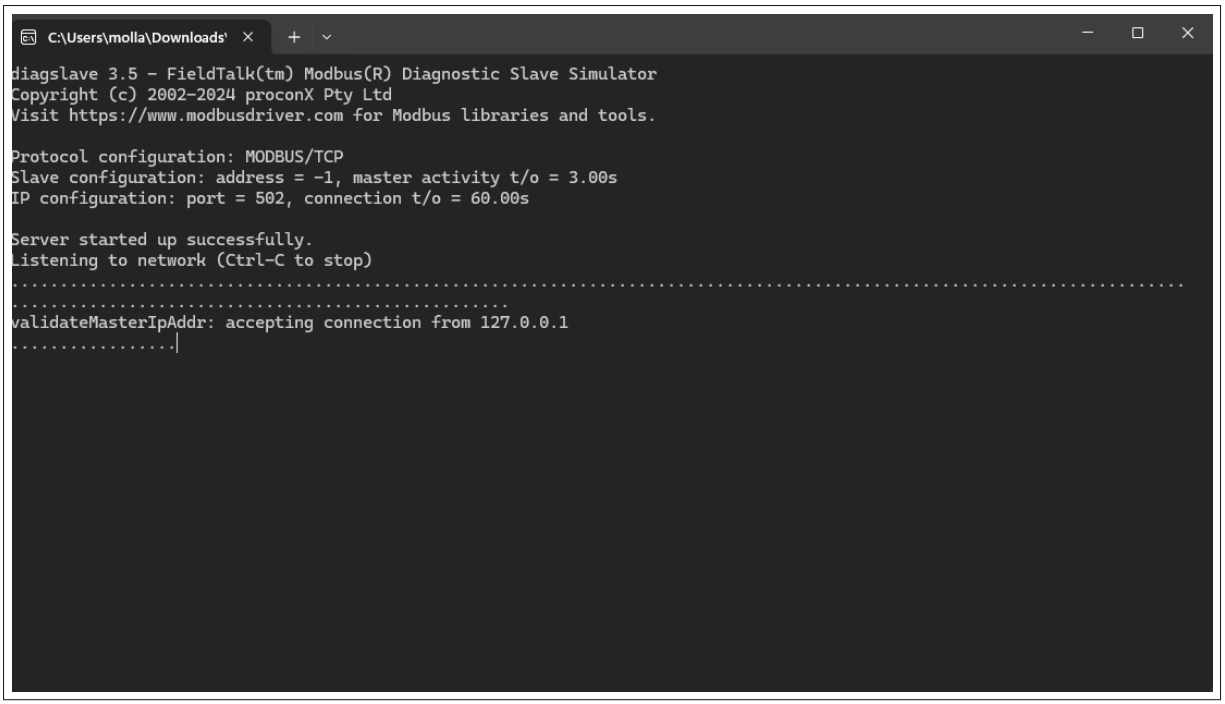

Figure-A III-17 Vue du terminal Diagslave.

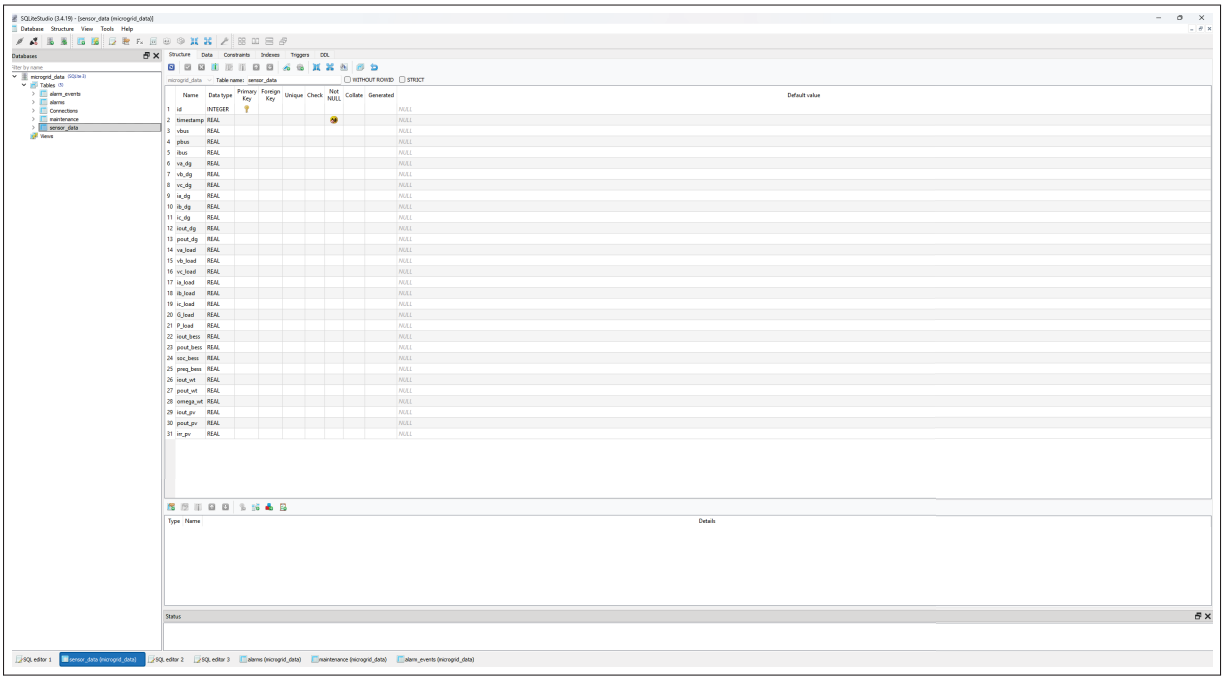

Figure-A III-18 Vue de la page d'accueil SQLite Studio.

BIBLIOGRAPHIE

Alomari, A., Alshraideh, M., Al Khawaldah, S. & Arman, N. (2025). Security of Smart Grid : Cybersecurity Issues, Potential Cyberattacks, Major Incidents, and Future Directions. *Energies*, 18(1), 141. doi : 10.3390/en18010141.

Arévalo, P., Cano, A. & Jurado, F. (2025). Data-Driven Microgrid Fault Detection Using Taguchi-Optimized Convolutional Neural Networks and Wavelet Transform. *Applied Soft Computing*, 169, 112575. doi : 10.1016/j.asoc.2024.112575.

Barbalho, R. S. M., Aquino, A. L. L. & Lopes, R. O. (2025). Reinforcement Learning Solutions for Microgrid Control : A Comprehensive Survey and Comparison With Conventional Approaches. *IEEE Access*, 13, 42851–42887. doi : 10.1109/ACCESS.2025.3549814.

Beus, M. & Sirovina, R. (2024). Design of a Generic Energy Management System (EMS) Platform for Microgrids. *9th International Conference on Smart and Sustainable Technologies*.

Beus, M., Herenčić, L., Pandžić, H. et al. (2022). Laboratory Setup for Stability and Optimization Studies of Hybrid Microgrids. *7th International Conference on Smart and Sustainable Technologies*.

Boyer, S. A. (2010). *SCADA : Supervisory Control and Data Acquisition* (éd. 4th). ISA — International Society of Automation.

Chen, T., Abdel-Rahim, O., Peng, F. & Wang, H. (2020). An Improved Finite Control Set-MPC-Based Power Sharing Control Strategy for Islanded AC Microgrids. *IEEE Access*, 8, 52676–52686.

Cintuglu, M. H., Elsayed, A. T. et al. (2015). Microgrid Automation Assisted by Synchrophasors. *IEEE Power & Energy Society General Meeting*.

Dar, M. F. & Singh, P. P. (2025). Advanced Integration of Bidirectional Long Short-Term Memory Neural Networks and Innovative Extended Kalman Filter for State of Charge Estimation of Lithium-Ion Battery. *Journal of Power Sources*, 626, 235747. doi : 10.1016/j.jpowsour.2024.235747.

Dragicevic, T. (2019). Model Predictive Control of Power Converters for Robust and Fast Operation of AC Microgrids. *IEEE Transactions on Power Electronics*, 33(7), 6304–6317.

Dubuisson, F. (2023). *Développement, étude et mise en œuvre du contrôle hiérarchique intelligent pour un microréseau hybride dédié aux stations de télécommunication isolées*. (Thèse de doctorat, École de technologie supérieure, Montréal).

Esram, T. & Chapman, P. L. (2007). Comparison of Photovoltaic Array Maximum Power Point Tracking Techniques. *IEEE Transactions on Energy Conversion*, 22(2), 439–449.

Flórez, M. A. P., Correa-Flórez, C. A. & Montoya, O. D. (2025). Hybrid Physics-LSTM Framework for Wind Power Prediction and Control in Virtual Microgrid Simulations. *IEEE Access*, 13, 94102–94117. doi : 10.1109/ACCESS.2025.3569126.

Gonzalez-Candelario, C. O., Darbali-Zamora, R. et al. (2023). Evaluation of an Autonomous Control Scheme for Interconnected DC Microgrids Using a Power Hardware-in-the-Loop Platform. *IEEE Access*.

Hermes, J. (2023). *.NET MAUI in Action*. Manning Publications.

Jaloudi, S. (2024). microIoTSCADA : A Tool to Monitor and Control Renewable Energy-Based Systems Using MODBUS. *4th Interdisciplinary Conference on Electrics and Computer (INTCEC)*.

Karamanakos, P. & Geyer, T. (2020). Guidelines for the Design of Finite Control Set Model Predictive Controllers. *IEEE Transactions on Power Electronics*, 35(7), 7434–7450.

Kotsiopoulos, T., Sarigiannidis, P., Ioannidis, D. & Tzovaras, D. (2025). Defending IIoT Against Modbus/TCP Threats : AI-Based Intrusion Detection and SDN-Based Mitigation. *International Journal of Information Security (Springer)*, 24, 88. doi : 10.1007/s10207-025-01076-2.

Kouro, S., Cortés, P., Vargas, R., Ammann, U. & Rodriguez, J. (2009). Model Predictive Control — A Simple and Powerful Method to Control Power Converters. *IEEE Transactions on Industrial Electronics*, 56(6), 1826–1838.

Microsoft. [Official .NET MAUI documentation]. (2023). .NET Multi-platform App UI (.NET MAUI) Documentation. Repéré à https://learn.microsoft.com/en-us/dotnet/maui/.

Modbus Organization. (2012). *MODBUS Application Protocol Specification V1.1b3*. Repéré à https://modbus.org.

Owens, M. (2020). *The Definitive Guide to SQLite* (éd. 2nd). Apress.

Pasztor, B. (2018). *Design and Implementation of a Supervisory Control and Data Acquisition System (SCADA) for a Microgrid Laboratory*. (Mémoire de maîtrise, Universitat Politècnica de Catalunya (UPC Barcelona)).

Patrascu, C., Muntean, N., Cornea, O. et al. (2016). Microgrid Laboratory for Educational and Research Purposes. *IEEE 16th International Conference on Environment and Electrical Engineering*.

Rey, J. M., Gómez, J., Duarte, N. et al. (2024). Design and Validation of an IoT System for an Experimental Laboratory Microgrid. *IEEE Latin America Transactions*. doi : 10.1109/TLA.2024.10810485.

Rodriguez, J., Kazmierkowski, M. P., Espinoza, J. R., Zanchetta, P., Abu-Rub, H., Young, H. A. & Rojas, C. A. (2013). State of the Art of Finite Control Set Model Predictive Control in Power Electronics. *IEEE Transactions on Industrial Informatics*, 9(2), 1003–1016.

Sharma, K. D., Chatterjee, A. & Rakshit, A. (2021). Modern SCADA Systems for Renewable Energy Integration : Challenges and Future Directions. *Renewable and Sustainable Energy Reviews*, 144, 111013.

Si, Y., Korada, N., Ayyanar, R. & Lei, Q. (2021). High-Performance Communication Architecture for a Smart Micro-Grid Testbed Using Customized Edge Intelligent Devices (EIDs) with SPI and Modbus TCP/IP Communication Protocols. *IEEE Open Journal of Power Electronics*.

SQLite.org. (2024). Performance Tuning — SQLite Documentation. Repéré à https://www.sqlite.org/speed.html.

Véliz, A., Riffo, S., Restrepo, C., Rivera, M. et al. (2023). A Review of Communication Protocols and Control Strategies in DC Microgrids : An Experimental Validation Focus. *IEEE 17th International Conference*.

Villalón, A., Muñoz, C., Muñoz, J. & Rivera, M. (2023). A Detailed dSPACE-Based Implementation of Modulated Model Predictive Control for AC Microgrids. *Sensors*, 23(14), 6288.

Xiong, Z., Li, Z., Li, Q. & Wu, Q. (2025). Deep Reinforcement Learning for Optimal Microgrid Energy Management With Renewable Energy and Electric Vehicle Integration. *Applied Soft Computing*, 172, 112878. doi : 10.1016/j.asoc.2025.112878.

Yıldırım, E., Dağdaş, S. & Karahan, M. (2025). Comprehensive Review of Edge Computing for Power Systems : State of the Art, Architecture, and Applications. *Applied Sciences*, 15(8), 4592. doi : 10.3390/app15084592.

Zerk, A. E. L., Ouassaid, M. & Zidani, Y. (2023). Development of a Real-Time Framework Between MATLAB and PLC Through OPC-UA : A Case Study of a Microgrid Energy Management System. *Scientific African*.

Zhaoxia, X., Zhijun, G., Guerrero, J. M. et al. (2017). SCADA System for Islanded DC Microgrids. *IECON 2017 — 43rd Annual Conference of the IEEE Industrial Electronics Society*.