

# Projet Réseaux locaux industriels

---

## Travail réalisé par:

- Imen Laouirine
- Ahmed Belaaj
- Omar Chaabouni

Enseignant: Dr. Sofiane OUNI

2021/2022

## OBJECTIF

Réalisation d'une application qui permet avec une interface graphique de créer des requêtes Modbus en Mode TCP.

## ENVIRONNEMENT DE TRAVAIL, OUTILS ET BIBLIOTHÈQUES

### VS code :

Visual Studio Code qui est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégré.

### Le langage de programmation:

Pour le langage de programmation, nous avons utilisé python (version 3.9.9)

### La bibliothèque PyModbus :

Pymodbus est une implémentation complète du protocole Modbus utilisant twisted/torndoo/asyncio pour son noyau de communication asynchrone. Il peut également être utilisé sans aucune dépendance tierce (à part pyserial) si un projet plus simple est nécessaire. De plus, il devrait fonctionner correctement sous toute version de python > 2.7.

## Le framework Flask :

Flask est un micro framework open-source de développement web en Python. Il est classé comme microframework car il est très léger.<sup>3</sup> Flask a pour objectif de garder un noyau simple mais extensible. Il n'intègre pas de système d'authentification, pas de couche d'abstraction de base de données, ni d'outil de validation de formulaires. Cependant, de nombreuses extensions permettent d'ajouter facilement des fonctionnalités.<sup>4</sup> Il est distribué sous licence BSD 5.

## DÉMARCHE ET MÉTHODOLOGIE DE TRAVAIL

Nous avons réalisé deux applications. La première représente le côté serveur et la deuxième représente le côté client qui va lancer les requêtes.

## Les fonctions à traiter :

Pour chaque requête nous allons extraire les champs correspondants aux variables nécessaires(l'adresse, le nombre de registre à lire, ...). Les fonctions supportées pour les requêtes sont les suivantes :

- **La fonction du code 01 : Lire les coils**

Cette fonction permet de lire l'état de 1 à 2000 bits dans un dispositif distant. La PDU(Protocol Data Unit) de demande spécifie l'adresse de départ, c'est-à-dire l'adresse de la première bit spécifié, et le nombre de bits. Dans le PDU, les bits sont adressés en commençant par zéro.

Exemple de requête :

**0B 01 001D 001F ED6E**

- ❖ **0B : Adresse de l'esclave (0B hex = adresse 11)**
  - ❖ **01 : Le code de fonction 1 (lecture des bits)**
  - ❖ **001D : L'adresse du premier bit à lire (001D hex = 29, +1 offset = bit #30).**
  - ❖ **001F : Le nombre total des bits demandés (25 hex = 31, entrées 30 à 60).**
  - ❖ **ED6E : Le CRC (Cyclic Redundancy Check) pour la vérification des erreurs.**
- **La fonction du code 03 : lire des registres**

Il est utilisé pour lire le contenu d'un bloc contigu de registres. Le PDU de requête spécifie l'adresse de départ du registre et le nombre de registres. Dans le PDU, les registres sont adressés en commençant par

zéro. Par conséquent, les registres numérotés de 1 à 16 sont adressés de 0 à 15.

Exemple de requête :

**0B 03 006F 0003 357C**

- ❖ **0B** : Adresse de l'esclave (0B hex = adresse 11)
- ❖ **03** : Le code de fonction 3 (lecture des registres)
- ❖ **006F** : L'adresse du premier registre demandé (006F hex = 111, +40001 offset = entrée #40112).
- ❖ **0003** : Le nombre total de registres demandés (lire 3 registres 40112 à 40114).
- ❖ **357C** : Le CRC (Cyclic Redundancy Check) pour la vérification des erreurs.

- **La fonction du code 05 : écrire un seul coil**

Le PDU de demande spécifie l'adresse du bit à forcer. Les bits sont adressés à partir de zéro. Par conséquent, le bit numéroté 1 est adressé comme 0. L'état On/Off demandé est spécifié par une constante dans le champ Coil Value. Une valeur de 0xFF00 demande que le bit soit activé (1). Une valeur de 0x0000 demande que le bit soit désactivé (0). Toutes les autres valeurs sont illégales et n'affectent pas le bit.

Exemple de requête :

**0B 05 00BF 0000 FC84**

- ❖ **0B** : Adresse de l'esclave (0B hex = adresse 11).
- ❖ **05** : Le code de fonction 5 (écriture d'un seul bit).
- ❖ **00BF** : L'adresse du bit (bit# 192 - 1 offset = 191 = BF hex).
- ❖ **0000** : L'état à écrire (FF00 = ON, 0000 = OFF).
- ❖ **FC84** : Le CRC (Cyclic Redundancy Check) pour la vérification des erreurs.

- **La fonction du code 06 : écrire un seul registre**

Ce code de fonction est utilisé pour écrire un seul registre de maintien dans un dispositif distant. Le PDU de demande spécifie l'adresse du registre à écrire. Les registres sont adressés en commençant par zéro.

Exemple de requête :

**0B 06 0004 ABCD 7604**

- ❖ **0B** : Adresse de l'esclave (0B hex = adresse 11).
- ❖ **06** : Code de fonction 6 (écriture d'un seul holding registre).
- ❖ **0004** : L'adresse des données du registre (0004 hex = 4, + 40001 offset = registre #40005).
- ❖ **ABCD** : La valeur à écrire.
- ❖ **7604** : Le CRC (Cyclic Redundancy Check) pour le contrôle des erreurs.

- **La fonction du code 15 : écrire plusieurs coils**

Ce code de fonction est utilisé pour forcer chaque bit d'une séquence de bits à être activé ou désactivé dans un dispositif distant. Le PDU de demande spécifie les références des bit à forcer. Les bits sont adressés à partir de zéro. Les états On/Off demandés sont spécifiés par le contenu du champ de données de la demande. Un "1" logique dans une position binaire du champ demande que la sortie correspondante soit activée. Un "0" logique demande à ce qu'elle soit désactivée.

Exemple de requête :

**0B 0F 001B 0009 02 4D01 6CA7**

- ❖ **0B** : Adresse de l'esclave (0B hex = adresse 11).
- ❖ **0F** : Le code de fonction 15 (écriture de bit multiples, 15 = 0F hex).
- ❖ **001B** : L'adresse du premier bit (001B hex = 27, + 1 offset = bobine #28).
- ❖ **0009** : Le nombre de bit à écrire (09 hex = 9).
- ❖ **02** : Le nombre d'octets de données à suivre (9 bits = 1 octet + 1 bit + 7 supports d'espacement = 2 octets).
- ❖ **4D** : Bits 35 - 28 (0100 1101).
- ❖ **01** : 7 supports d'espace et bit 36 (0000 0001).
- ❖ **6CA7** : Le CRC (Cyclic Redundancy Check) pour le contrôle des erreurs.

- **La fonction du code 16 : écrire plusieurs registres**

Ce code de fonction est utilisé pour écrire un bloc de registres contigus (1 à 123 registres) dans un dispositif distant. Les valeurs à écrire sont spécifiées dans le champ de données de la requête. Les données sont emballées sous forme de deux octets par registre.

Exemple de requête :

**0B 10 0012 0002 04 0B0A C102 A0D5**

- ❖ **0B** : Adresse de l'esclave (0B hex = adresse 11).
- ❖ **10** : Le code de fonction 16 (écriture de holding registres multiples, 16 = 10 hex).
- ❖ **0012** : L'adresse du premier registre (0012 hex = 18, +40001 offset = registre #40019).
- ❖ **0002** : Le nombre de registres à écrire.
- ❖ **04** : Le nombre d'octets de données à suivre (2 registres x 2 octets chacun = 4 octets).
- ❖ **0B0A** : La valeur à écrire dans le registre 40019.
- ❖ **C102** : La valeur à écrire dans le registre 40020.
- ❖ **A0D5** : Le CRC (Cyclic Redundancy Check) pour le contrôle des erreurs.

- **La fonction du code 23 : écrire/lire des registres**

Ce code de fonction est utilisé pour écrire et lire des blocs de registres contigus (1 à 123 registres) dans un dispositif distant en même temps. Le PDU de requête spécifie l'adresse de départ du registre et le nombre de registres et les valeurs à écrire sont spécifiées dans le champ de données de la requête. Les données sont emballées sous forme de deux octets par registre.

Exemple de requête :

**0B 17 0000 0002 0FFF 0002 04 ABCD 1234 4025**

- ❖ **0B** : Adresse de l'esclave (0B hex = adresse 11).
- ❖ **17** : Le code de fonction 23 (écriture/lecture des registres multiples, 23= 17 hex).
- ❖ **0000** : L'adresse du premier registre demandé
- ❖ **0002** : Le nombre des registres à lire
- ❖ **0FFF** : L'adresse du premier registre.

- ❖ 0002 : Le nombre de registres à écrire.
- ❖ 04 : Le nombre d'octets de données à suivre (2 registres x 2 octets chacun = 4 octets).
- ❖ 0B0A : La valeur à écrire dans le registre 44096.
- ❖ C102 : La valeur à écrire dans le registre 44097.
- ❖ 4025 : Le CRC (Cyclic Redundancy Check) pour le contrôle des erreurs.

## L'interface graphique de l'application :

En ce qui concerne l'interface graphique nous avons utilisé le framework Flask et nous avons réalisé une interface contenant un espace de texte pour que le client puisse lancer sa requête écrite en hexadécimal et un autre espace pour afficher la réponse du serveur.

- Nous allons tout d'abord lancer notre application sur le port 200 :

```
C:\Windows\System32\cmd.exe - python app.py
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

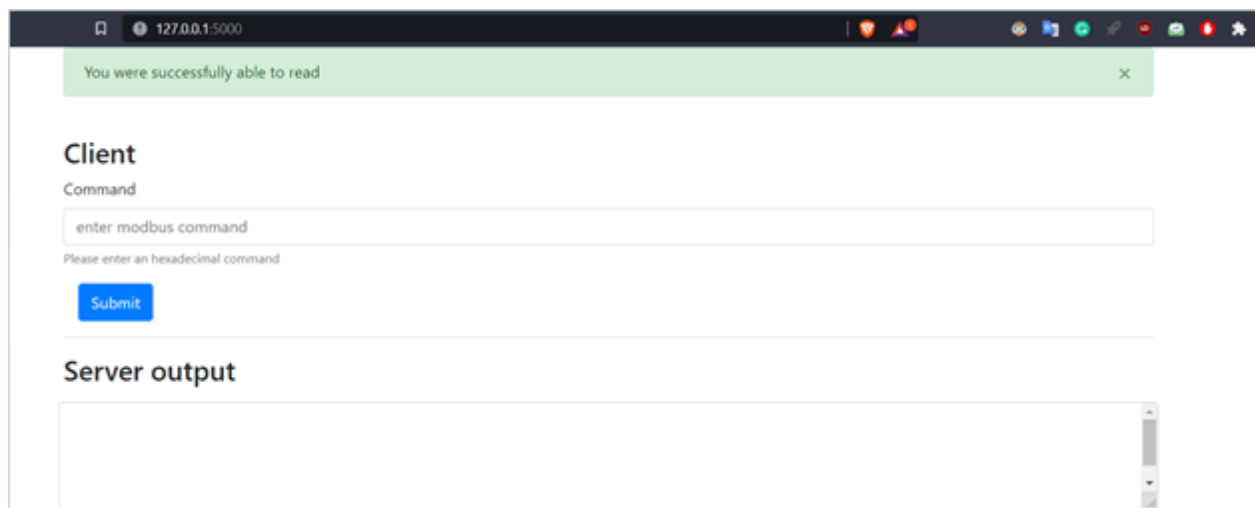
C:\Users\Omar\OneDrive\Bureau\pymodbus>python app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 108-060-789
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [16/Dec/2021 16:32:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:32:05] "GET /static/js/bootstrap.js HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:32:05] "GET /static/css/bootstrap.css HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:32:05] "GET /static/js/popper.min.js HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:32:05] "GET /static/js/jquery.min.js HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:32:06] "GET /favicon.ico HTTP/1.1" 404 -
[11, 15, 27, 9, 2, 19714, 27815]
Slave adress 11
Write multiple coils, single bit access
127.0.0.1 - - [16/Dec/2021 16:35:35] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:35:35] "GET /static/css/bootstrap.css HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:35:35] "GET /static/js/jquery.min.js HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:35:35] "GET /static/js/popper.min.js HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:35:35] "GET /static/js/bootstrap.js HTTP/1.1" 304 -
[1, 1, 27, 15, 60782]
Slave adress 1
Read coils, single bit access
127.0.0.1 - - [16/Dec/2021 16:36:36] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:36:36] "GET /static/css/bootstrap.css HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:36:36] "GET /static/js/jquery.min.js HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:36:36] "GET /static/js/popper.min.js HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:36:36] "GET /static/js/bootstrap.js HTTP/1.1" 304 -
[11, 16, 18, 2, 4, 5, 112, 41173]
Slave adress 11
Write multiple registers, 16 bit access
127.0.0.1 - - [16/Dec/2021 16:44:24] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [16/Dec/2021 16:44:24] "GET /static/js/jquery.min.js HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:44:24] "GET /static/css/bootstrap.css HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:44:24] "GET /static/js/popper.min.js HTTP/1.1" 304 -
127.0.0.1 - - [16/Dec/2021 16:44:24] "GET /static/js/bootstrap.js HTTP/1.1" 304 -
```

- Puis nous allons lancer le serveur qui sera à l'écoute des requêtes du client :

```
C:\Windows\System32\cmd.exe - python modbus_server.py
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

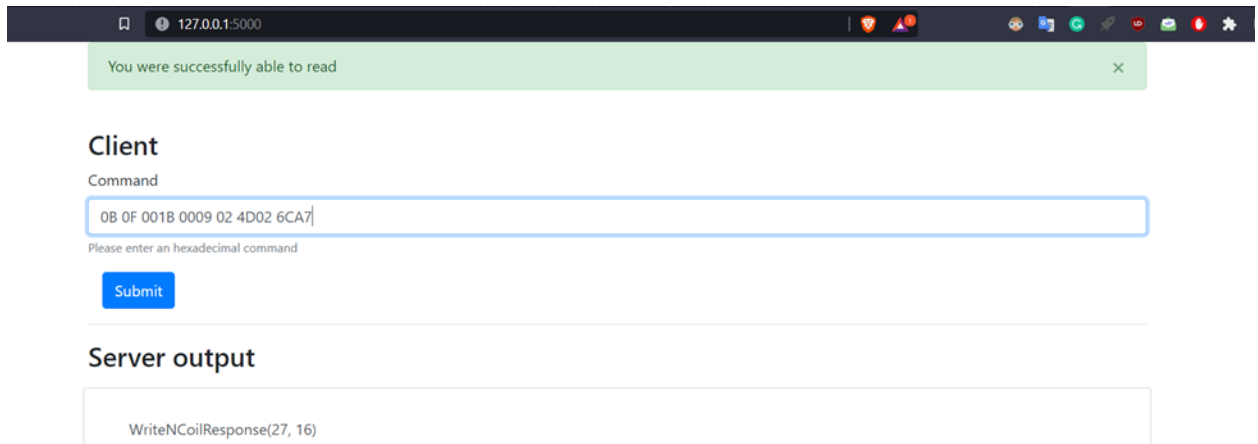
C:\Users\Omar\OneDrive\Bureau\pymodbus>python modbus_server.py
[+]Info : Server Started on IP : 127.0.0.1 and PORT : 200
```

- Une fois l'application est lancée et le serveur est sur écoute, le client peut lancer les requêtes



Et voici quelques exemples d'exécution :

- ❖ Ecriture de 9 bits à partir de l'adresse 001B :



- ❖ Lecture de 15 bits à partir de l'adresse 001B :

127.0.0.1:5000

You were successfully able to read

### Client

Command

01 01 001B 000F ED6E

Please enter an hexadecimal command

Submit

### Server output

ReadCoilsResponse(16)  
[0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

- ❖ Ecriture de deux registres de valeurs 5 et 112 à partir de l'adresse 0012 :

127.0.0.1:5000

You were successfully able to read

### Client

Command

0B 10 0012 0002 04 0005 0070 A0D5

Please enter an hexadecimal command

Submit

### Server output

WriteMultipleRegisterResponse (18,2)



❖ Lecture de deux registre à partir de l'adresse 0012:

127.0.0.1:5000

You were successfully able to read

### Client

Command

0B 03 0012 0002 7687

Please enter an hexadecimal command

Submit

### Server output

ReadHoldingRegistersResponse (2)  
[5, 112]

❖ Ecriture d'un seul bit (le bit 1 ) dans l'adresse 0007:

127.0.0.1:5000

You were successfully able to read

### Client

Command

01 05 0007 FF00 FC84

Please enter an hexadecimal command

Submit

### Server output

WriteCoilResponse(7) => 1