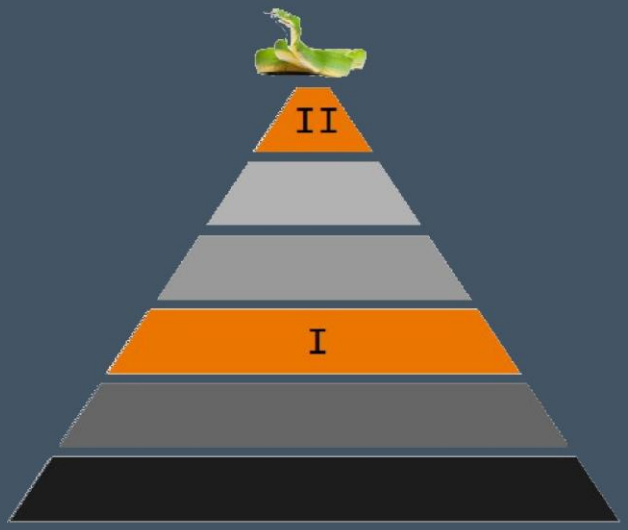
Programming II

Capstone Project



Dr James Garland

Dr Diarmuid Ó Briain

Version: 3.1

Ollscoil Teicneolaiochta an Oirdheiscirt

South East Technological University

Table of Contents

1 Capstone project introduction 5
2 The testbed 5
3 Virtual Environment 6
4 Running the Server 6
Running the server. 6
5 Capstone requirement – Part A (40%) 7
5.1 Startup Information (5%) 7
5.2 Running the client with IP details 7
5.3 Read values 8
5.4 Write values 9
5.5 Creating or Updating a Username and Password 10
5.6 Error options 10
5.7 Exiting the application 10
5.8 Other application activity 10
6 Capstone requirement – Part B (30%) 11
6.1 RESTful API 11
6.2 Production Webserver 12
7 Capstone requirement – Part C (15%) 12
7.1 Virtual Environment 12
7.2 Network capture 12
7.3 Scan (5%) 13
8 Capstone requirement – Part D (15%) 13
8.1 Additional feature 13
9 Submission 13
10 Assessment notes 13

Illustration Index

Figure 1: The capstone testbed 5
Figure 2: Mozilla RESTer, RESTful API browser 11
Figure 3: Wireshark capture 12

This page is intentionally blank

1 Capstone project introduction

The goal of this capstone project is to allow you to explore Python3 via a project to embed the learning from the lectures and exercises in the context of an Industrial Networking problem. It is expected that you contribute to the development of a solution based on the learnings from the overall course and beyond. Use the python documentation (https://docs.python.org/3/).

2 The testbed

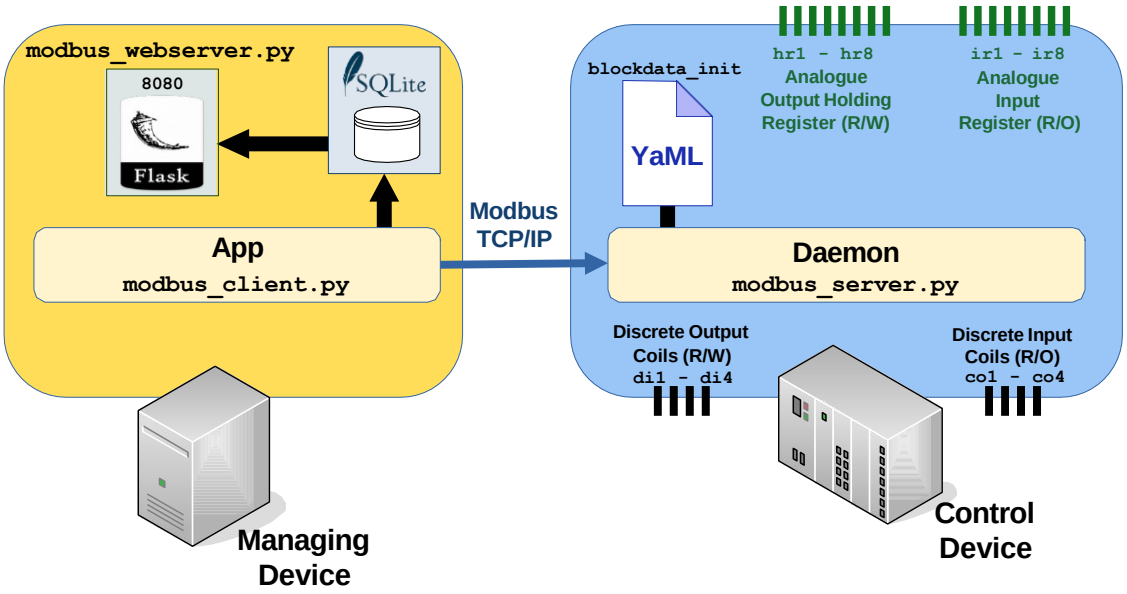


Figure 1: The capstone testbed

The testbed in Figure 1 consists of a control device, which will be represented by a given program called **`modbus_server.py`**. This control device has four sets of datablocks, namely:

Discrete Input Contacts

- A set of four input, Read/Only coils.
- These contacts are either activated or not and can be logic “0” or logic “1”.

Discrete Output Coils

- A set of four output, Read/Write coils.
- These coils are either activated or not and can be logic “0” or logic “1”.

Analogue Input Register

- An 8-byte, output, Read/Only register.
- These registers contain any integer value between 0-9.

Analogue Output Holding Register

- An 8-byte, output, Read/Write register.
- These registers can take any integer value between 0-9.

3 Virtual Environment

All work for this assignment must be carried out within a Python virtual environment called **.pymod**, and this is important as you must submit a **requirements.txt** file for the environment to trap the packages necessary to run it.

4 Running the Server

On a computer simulating the Control Device, run the **modbus_server.py** program. This program will import a set of default settings from a configuration file called **blockdata_init.yaml**, which you will also be given.

```
(.pymod)~$ cat blockdata_init.yaml
# Initial Data for Modbus Server

di: [1,1,0,0]             # Discrete Input Contacts
co: [0,0,1,1]             # Discrete Output Coils
ir: [0,0,0,0,4,4,4,4]     # Analogue Input Register
hr: [3,3,3,3,0,0,0,0]     # Analogue Output Holding Register
```

Running the server.

```
(.pymod)~$ ./modbus_server.py
Running with Effective User ID (euid): 1000
Switching to root privileges as euid: 0
[sudo] password for alovelace: adapass

  Vendor Name : SETU Modbus Laboratory
  Vendor URL  : https://pymodbus.readthedocs.io/
  Product Name: Modbus Server
  Model Name  : Python Modbus
  Revision    : 3.0

  Initial State
  -----------------

  Discrete Input Contacts           :  True, True, False, False
  Discrete Output Coils             :  False, False, True, True
  Analogue Input Register           : [ 0, 0, 0, 0, 4, 4, 4, 4 ]
  Analogue Output Holding Register  : [ 3, 3, 3, 3, 0, 0, 0, 0 ]

  Listening on 127.0.1.1:502 for 0.0.0.0:*
  Initiating Modbus daemon...
```

Upon running the server, the default data is read in, and the coils, contacts and registers are populated with the block data.

5 Capstone requirement – Part A (40%)

Write a complementary **`modbus_client.py`** that operates as follows:

5.1 Startup Information (5%)

Suppose the user attempts to run the program, without adequately providing information to connect to the running server. In that case, a suitable help structure such as this must be provided as help.

```
(.pymod)~$ ./modbus_client.py
usage: modbus_client.py [-h] [-i IP] [-p PORT]

Modbus client

optional arguments:
  -h, --help            show this help message and exit
  -i IP, --ip IP        Modbus server IP address
  -p PORT, --port PORT  Modbus server TCP port
```

5.2 Running the client with IP details

Giving at least the IP address but with the option to supply a port number, the program will also present a menu of options, as displayed below. Each page will be preceded by the Vendor Name, Product Code, and Revision, all of which must be extracted from the Modbus server—the values, in this case, are to be coloured green on the terminal.

```
(.pymod)~$ ./modbus_client.py -i 127.0.0.1 -p 502

(.pymod)~$ ./modbus_client.py -i 127.0.0.1

  Vendor Name    : SETU Modbus Laboratory
  Product Code   : PM
  Revision       : 3.0

  ------------------------------------------------------

  1. Discrete Input Contact values
  2. Discrete Output Coil values
  3. Analogue Input Register values
  4. Analogue Output Holding Register values
  5. Display all Discrete and Register values
  6. Write values to the Discrete Output Coils
  7. Write to the Analogue Output Holding Register
  8. Create/update a username and password
  9. Exit/Quit

  Select an activity:
```

5.3 Read values

The following output is expected from options 1 to 5. Note the colouring required for bool returns, such as True and False.

```
1. Discrete Input Contacts values
Select an activity: 1

  Discrete Input Contacts [R/O]

  Contact 0: True
  Contact 1: True
  Contact 2: False
  Contact 3: False

  Main menu, press Enter:

2. Discrete Output Coils values
Select an activity: 2

  Discrete Output Coil [R/W]

  Coil 0: False
  Coil 1: False
  Coil 2: True
  Coil 3: True

  Main menu, press Enter:

3. Analogue Input Register values
Select an activity: 3

  Analogue Input Register [R/O]

  Register 0: 0
  Register 1: 0
  Register 2: 0
  Register 3: 0
  Register 4: 4
  Register 5: 4
  Register 6: 4
  Register 7: 4

  Main menu, press Enter:


4. Analogue Output Holding Register values
Select an activity: 4

  Analogue Output Holding Register [R/W]

  Register 0: 3
  Register 1: 3
  Register 2: 3
  Register 3: 3
  Register 4: 0
  Register 5: 0
  Register 6: 0
  Register 7: 0

  Main menu, press Enter:
```

```
5. Display all Discrete and Register values
Select an activity: 5

  Display all Discrete and Register values

   Discrete Output Contacts            True, True, False, False
   Discrete Input Coils                False, False, True True
   Analogue Input Registers           [ 0, 0, 0, 0, 4, 4, 4, 4 ]
   Analogue Output Holding Registers  [ 3, 3, 3, 3, 0, 0, 0, 0 ]

  Main menu, press Enter:
```

5.4 Write values

The following output is expected from options 6 to 7. Note the colouring required for bool returns, such as True and False. In option 6, values given other than 1 or 0 should be ignored, and if the user does not give enough values, then the remaining values should be with 0 to complete the coil or register. Similarly, for option 7, values given outside 0 – 9 should be discarded, and padding of 0 should be applied where necessary to complete the eight slots in the register.

```
6. Write values to the Discrete Output Coils
Select an activity: 6

  Enter comma separated
  list of up to 4 bool values: 1,2,1,1,1

  Values such as '2' are not valid, skipping.

  Accepting these values' 1, 1, 1, 1'.

  Post write, Discrete Output Coil values:

  Success: True

  Coil 0: True
  Coil 1: True
  Coil 2: True
  Coil 3: True

  Main menu, press Enter:


7. Write to the Analogue Output Holding Register
Select an activity: 7

  Enter comma separated
  list of up to 8 integer values: 2,3,4,5,6

  Accepting these values' 2, 3, 4, 5, 6, 0, 0, 0'. Post

  write, Analogue Output Holding Register values:

  Success: True

  Register 0: 2
  Register 1: 3
  Register 2: 4
  Register 3: 5
  Register 4: 6
  Register 5: 0
  Register 6: 0
  Register 7: 0

  Main menu, press Enter:
```

5.5 Creating or Updating a Username and Password

Tokens are required for user access to the RESTful API. Option 8 presents the option to enter username and password for storage in an appropriate database table. Note that the password should not be shown to the terminal as it is typed.

```
 8. Create/update a username and password

  Select an activity: 8

  Enter a username: ada
  Enter a password:

    User ‘ada’ and the associated password added
```

5.6 Error options

Any illegal option received at the menu, for example, 10, should result in a message informing the user of their error and return them to the menu options.

```
Select an activity: 10

Not a valid choice, try again
```

5.7 Exiting the application

The final option, option 9, should present the following message to the user for a short period and then return the shell to the clear state.

```
 8. Exit/Quit

  Select an activity: 9

Goodbye, the Modbus Server will be there when you return

  (.pymod)~$
```

5.8 Other application activity

The application should populate a database in the background, which will persistently hold information for the RESTful API in a suitable table. For example, the activities above should make entries similar to the following in the database.

```
(.pymod)~$ sqlite3 modbus_db.sqlite
SQLite version 3.31.1 2020-01-27 19:55:54
Enter “.help” for usage hints.
sqlite> select * from modbus;
127.0.0.1|1,1,0,0|1,1,1,1|0,0,0,0,4,4,4,4|2,3,4,5,6,7,8,9
```

6 Capstone requirement – Part B (30%)

6.1 RESTful API

Write a complementary **`Flask`** application that provides a secure RESTful API using a name such as **`modbus_rest.py`**. The application should return information held in the database for any IP address in JSON form, providing a valid token from the database, which is included in the curl request.

```
(.pymod)~$ ./modbus_rest.py
 * Serving Flask app 'modbus_rest' (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
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 144-821-704
```

```
(.pymod)~$ curl -X GET -H "Authorization: Token secret1"
http://127.0.0.1:5000/127.0.0.1
{"ip": "127.0.0.1", "di": "1,1,0,0", "co": "1,1,1,1", "ir":
"0,0,0,0,4,4,4,4", "hr": "2,3,4,5,6,7,8,9"}
```

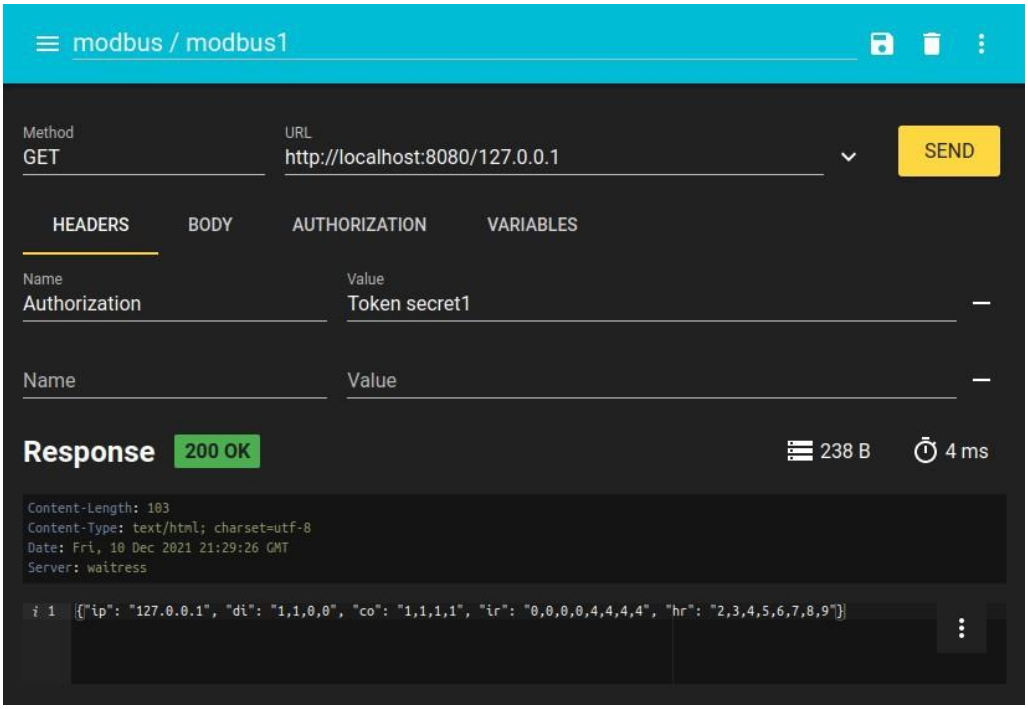


Figure 2: Mozilla RESTer, RESTful API browser

6.2 Production Webserver

Put the newly developed RESTful API app into production on TCP port 8080 using **`waitress`**.

```
(.pymod)~$ ./modbus_webserver.py

(.pymod)~$ curl -X GET -H "Authorization: Token secret1"
http://127.0.0.1:8080/200.1.1.1
{"ip": "127.0.0.1", "di": "1,1,0,0", "co": "1,1,1,1", "ir":
"0,0,0,0,4,4,4,4", "hr": "2,3,4,5,6,7,8,9"}
```

7 Capstone requirement – Part C (15%)

7.1 Virtual Environment

A **`requirements.txt`** file containing the information to automatically establish a Python virtual environment will be developed as part of the solution package.

7.2 Network capture

Using an appropriate network traffic monitor such as **tcpdump**, **tshark** or **Wireshark**, capture a TCP stream to include the messaging between the application and the server for one of the activities that cause a change to the values of a coil or register. With the aid of a suitable diagram, narrate the role of each packet in the interaction.

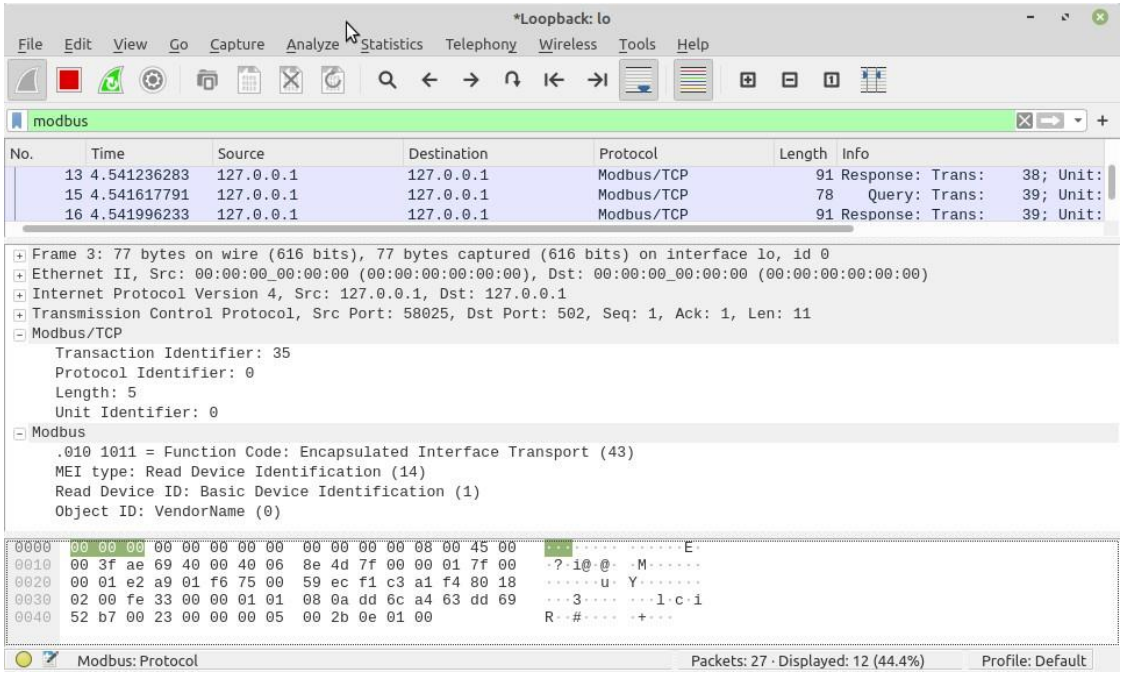


Figure 3: Wireshark capture

7.3 Scan (5%)

Carry out a Cyber reconnaissance of the **`modbus server`** to determine as much information as possible using **`nmap`**. Describe your findings and the process you used to achieve them.

8 Capstone requirement – Part D (15%)

8.1 Additional feature

Add one additional feature to your solution that you feel can improve it. Write a short description of the feature, and using your preferred desktop screen capture program, make a short video demonstrating the feature, as well as its role and function. The video must also give a brief walk-through of your code.

9 Submission

All files, plus a **`README.md`** markdown description file, are to be uploaded to your **`GitHub, GitLab, or Onedrive`** repository on or before 23:59 hrs, 2 May 2025. An email with a link to the upload to be sent to james.garland@setu.ie to confirm by the same deadline.

10 Assessment notes

Plagiarism or copying of other people's work will not be tolerated.

The examination team may run a Viva Voce with any candidate to determine the mark to be assigned. In such viva, the candidate will be quizzed on how they developed the application and how it works.

This page is intentionally blank