NOVA SCHOOL OF SCIENCE & TECHNOLOGY

MDE

Labwork2 – Class 4

MQTT inside Prolog

Prolog HTTP Server

Java App Integration

2024 - 2025

Lab2 (Class 4)

- MQTT inside Prolog
  - MQTT broker installation and experiments [OPTIONAL]
  - Using MQTT inside Prolog
  - Examples

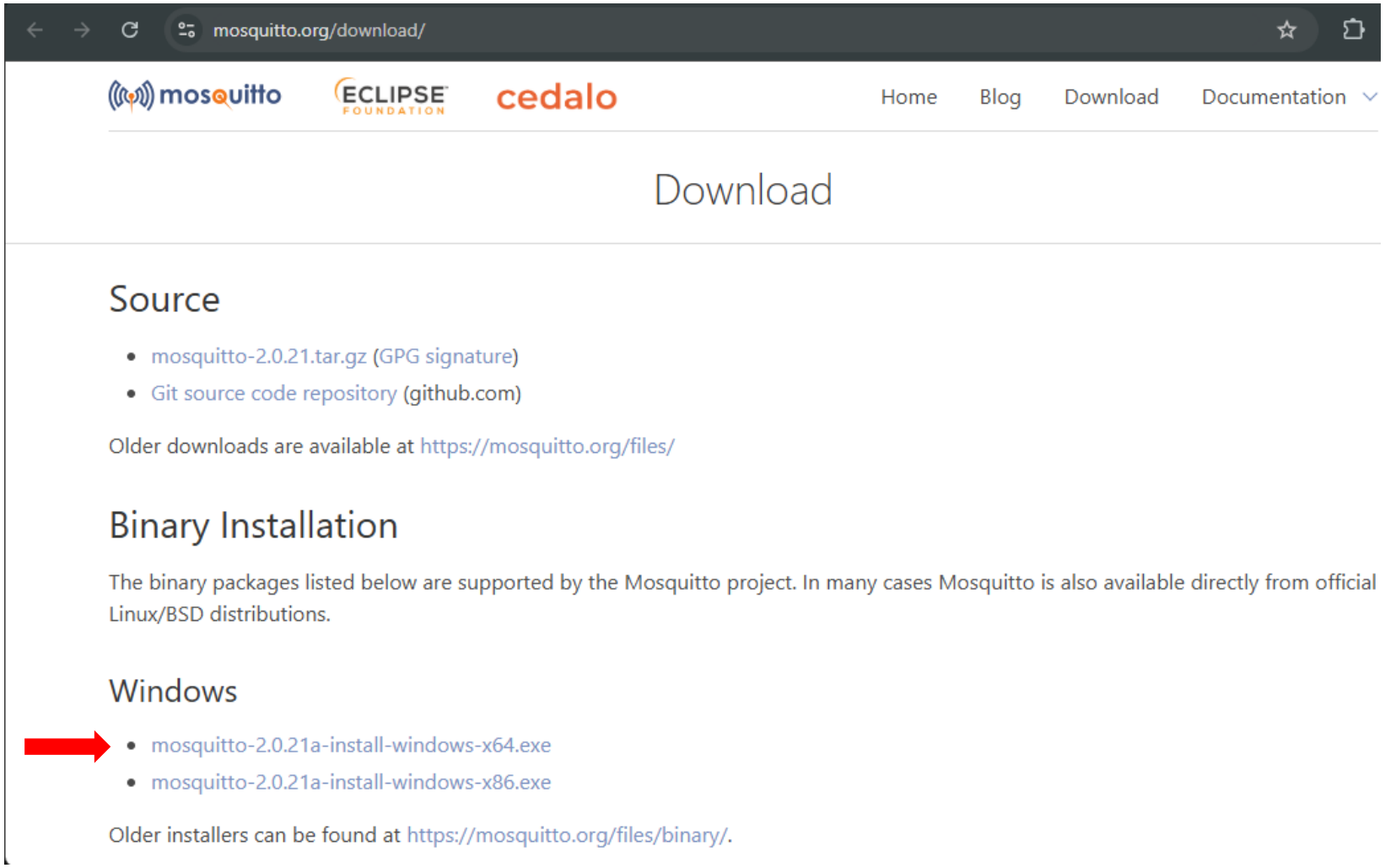
- Prolog HTTP Server, integration with JAVA using JSON:
  - Prolog Http Server
  - Java Http Client
  - Examples

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

MQTT inside PROLOG

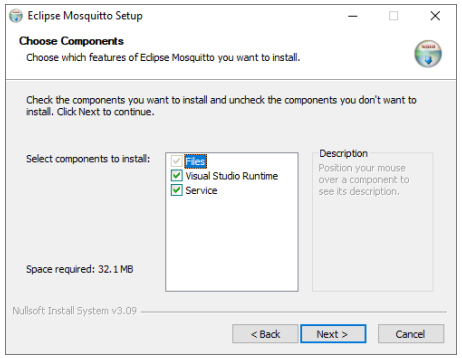
Mosquitto Broker Install (1) [OPTIONAL]

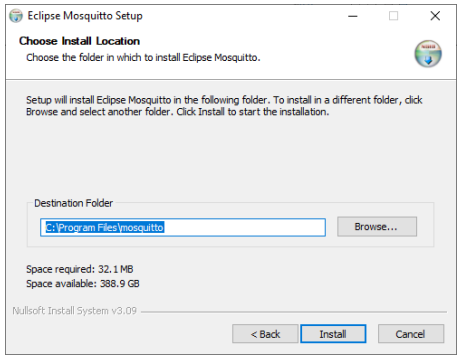
- https://mosquitto.org/download/

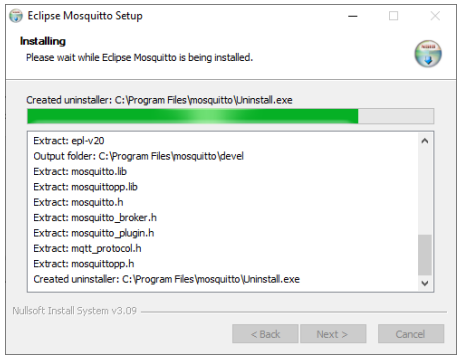


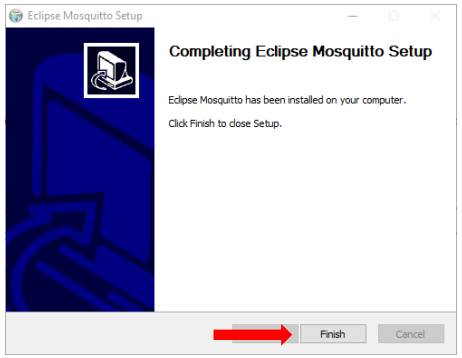
Mosquitto Broker Install (2) [OPTIONAL]

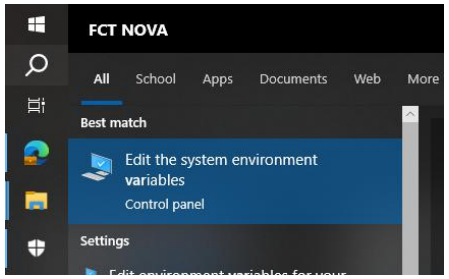
- Install mosquitto as illustrated in the pictures.
- Add the mosquitto folder to the PATH system variables as shown in the pictures below.
- Close all the dialogs regarding the PATH creation, by clicking OK in all them.
- It is rare, in some windows we need to restart the system (probably if the next slide fails).

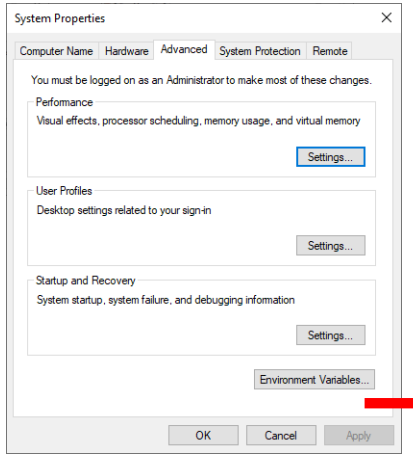


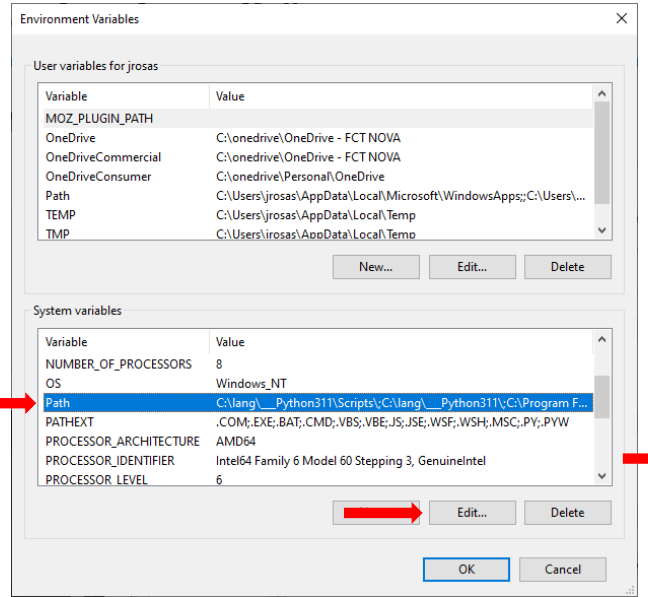


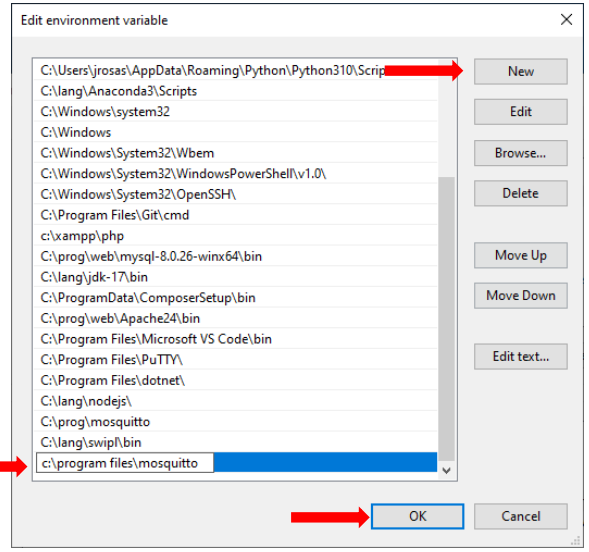






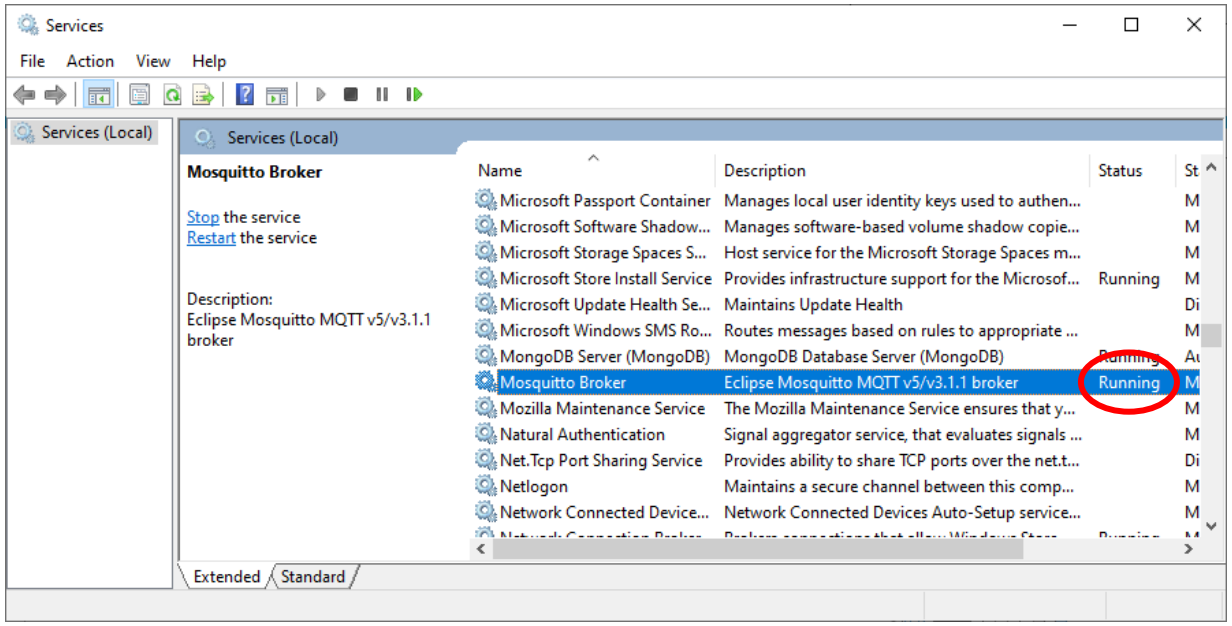




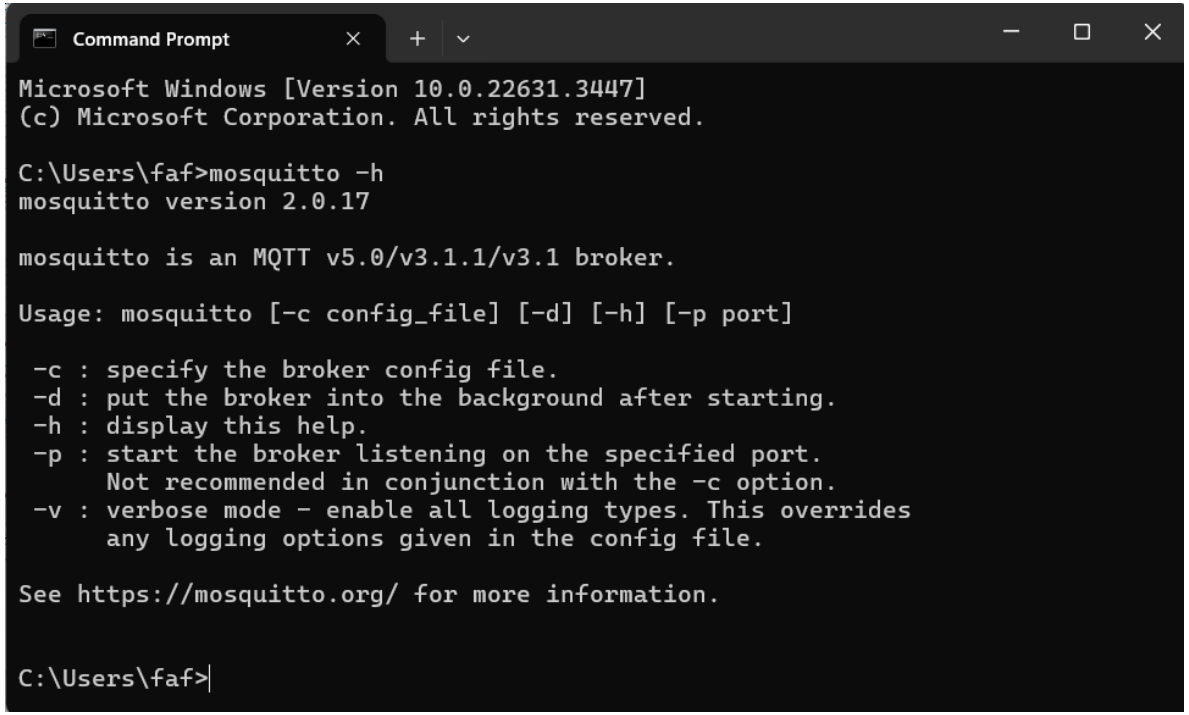


Testing MQTT [OPTIONAL]

- Verify if the service is running…



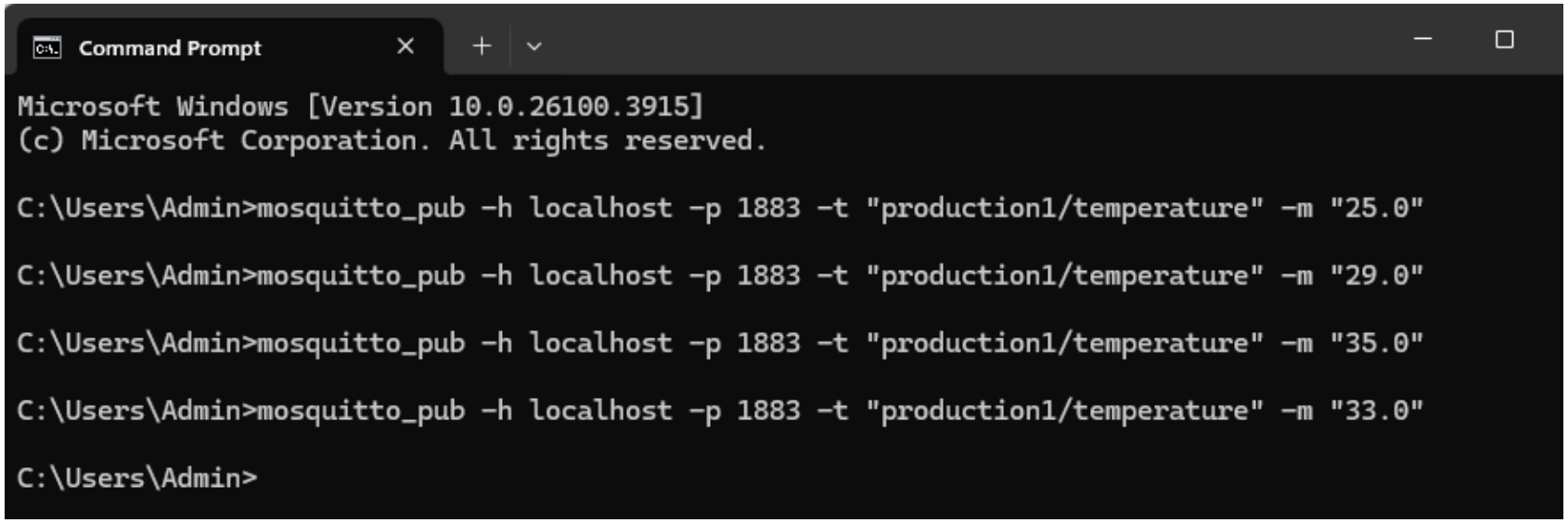
Run the command -> `mosquitto -h`



Testing Subscribe and Publish (locally…) [OPTIONAL]

For testing purposes, let's simulate the publication of temperature sensor values in production1 (see Labwork1).

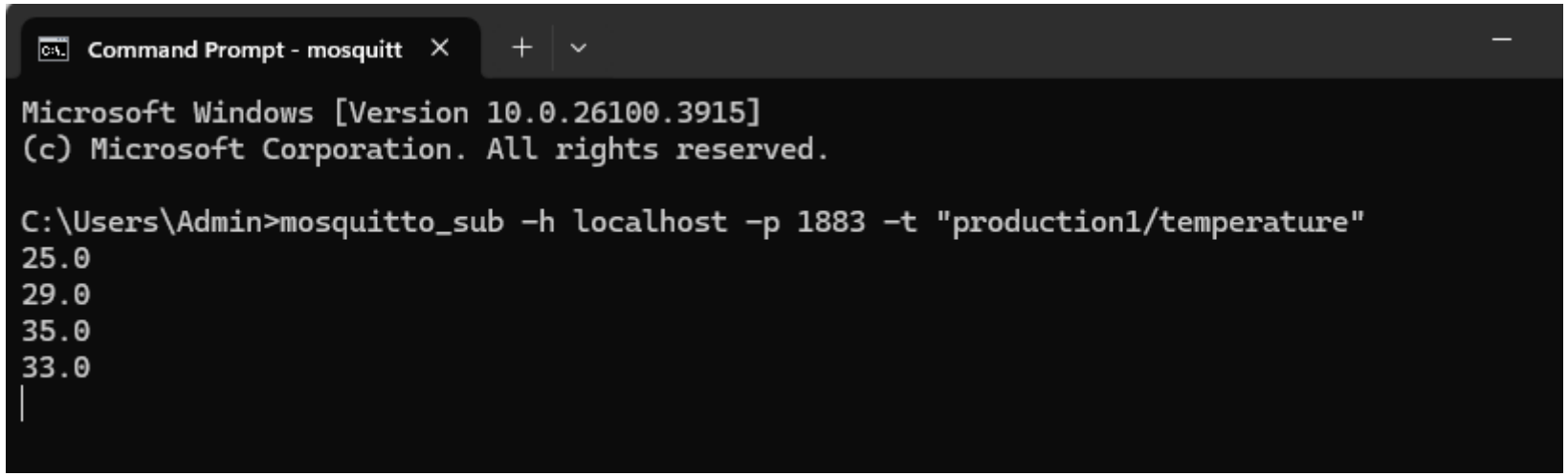
```
mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "25.0"
mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "29.0"
mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "35.0"
mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "33.0"
```



Testing Subscribe and Publish (locally…) [OPTIONAL]

Now, let's subscribe the corresponding topic:

```
mosquitto_sub -h localhost -p 1883 -t "production1/temperature"
```



Testing Subscribe (remotely to lab1.3 MQTT Broker)

Let’s test the connection to the lab 1.3 MQTT Broker and the subscribe to the corresponding topic:

```
mosquitto_sub -h 192.168.250.201 -p 1883 -t "production1/temperature"
```

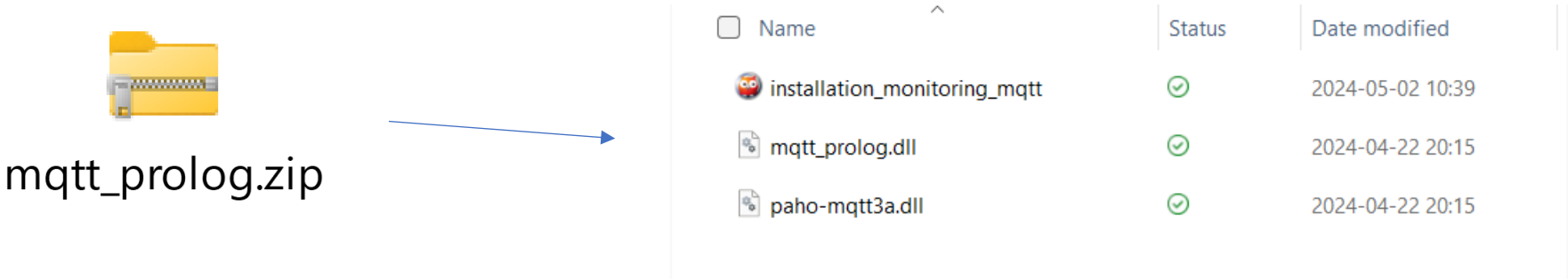
Do not forget to change your network to MDE

Creation of Prolog Module using MQTT

Now let’s create a Prolog module for using MQTT inside Prolog.

Extract the Modules for MQTT in Prolog

- Download from CLIP the zip file **mqtt_prolog**.
- Extract the zip file to your working folder



production_monitoring_mqtt.pl

- Open and edit the production_monitoring_mqtt.pl file.

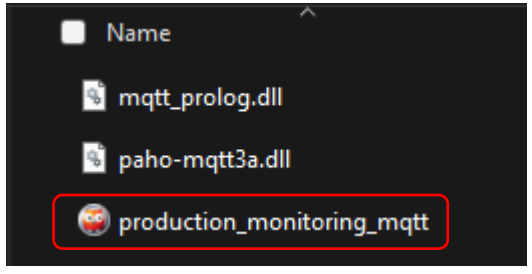
production_monitoring_mqtt.pl

File Edit Browse Compile Prolog Pce Help

production_monitoring_mqtt.pl

```
mqtt_broker('tcp://localhost:1883').      % case working locally
%mqtt_broker('tcp://192.168.250.201:1883'). % case working with lab1.3
                                            % MQTT Broker

:-dynamic mqtt_library_loaded/0.
:-dynamic mqtt_monitoring_handler/1.


load_mqtt_library:-
   not(mqtt_library_loaded),
   % loads the library mqtt_prolog.dll
   use_foreign_library(foreign(mqtt_prolog)),
   assert(mqtt_library_loaded),
   !;
   true.

create_monitoring_client:-
    load_mqtt_library,
    not(mqtt_monitoring_handler(_)),
    mqtt_broker(Broker_URL),
    mqtt_create_client(production_monitoring, Broker_URL, Handler),
    % the Handler is the C/C++ void *pointer inside the DLL.
    assert(mqtt_monitoring_handler(Handler)),
    mqtt_connect(Handler, _Result),
    !;
    true.

production_monitoring_on_connect_failure(_Handler):-
   format('failure connection of ~w~n', [production_monitoring]).

production_monitoring_on_connect_success(Handler):-
   format('success connection of ~w~n', [production_monitoring]),
   mqtt_subscribe(Handler, 'production1/temperature', 1, _Result1).

   %subscribe the other related topics

production_monitoring_on_message_arrived('production1/temperature', Mes
sage,_Handler):-
   format('mqtt topic: ~w~n',['production1/temperature']),
   format('mqtt message: ~w~n~n',[Message]).


% create the other topics on_message_arrived
```

Line: 31



SWI-Prolog -- c:/Users/Admin/OneDrive - FCT NOVA/Work/1 - FCT - U...

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free softw
are.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- edit.
true.

?-
```

production_monitoring_mqtt.pl

- The fact mqtt_broker(Broker) holds the URL address of the MQTT broker.
- Uncomment/comment whenever you are working with your local MQTT broker or 1.3 lab MQTT broker

```
%Defining the MQTT broker details
mqtt_broker('tcp://localhost:1883').
% mqtt_broker('tcp://192.168.250.201:1883').
```

- Loads the library mqtt_prolog.dll.
- The assert of fact mqtt_library_loaded is to avoid repeating the library load if we call the predicate **load_mqtt_library** more than once.

```
load_mqtt_library:-
    not(mqtt_library_loaded),
    % loads the library mqtt_prolog.dll
    use_foreign_library(foreign(mqtt_prolog)),
    assert(mqtt_library_loaded),
    !;
    true.
```

production_monitoring_mqtt.pl

- **`create_monitoring_client`** is the predicate that allows the creation of the client and the corresponding connection to the MQTT Broker:
  - The `mqtt_create_client(production_monitoring, Broker_URL, Handler)`, delivers the MQTT interface for the ***new client*** in the specific ***broker URL.***
  - The fact `mqtt_monitoring_handler(Handler)` is used to avoid repeating the client creation, if the predicate *create_monitoring_client*, is called more than once.
  - The `mqtt_connect(Handler, _Result)` predicate establishes a successful connection with the MQTT Broker in an undetermined future instant (but it is not known when).
  - But when it happens, a callback function predicate is automatically called, as shown in the next slide.

```
create_monitoring_client:-
    load_mqtt_library,
    not(mqtt_monitoring_handler(_)),
    mqtt_broker(Broker_URL),
    mqtt_create_client(production_monitoring, Broker_URL, Handler),
    % the Handler is the C/C++ void *pointer inside the DLL.
    assert(mqtt_monitoring_handler(Handler)),
    mqtt_connect(Handler, _Result),
    !;
    true.
```

production_monitoring_mqtt.pl

```
create_monitoring_client:-
    load_mqtt_library,
    not(mqtt_monitoring_handler(_)),
    mqtt_broker(Broker_URL),
    mqtt_create_client(production_monitoring, Broker_URL, Handler),
    % the Handler is the C/C++ void *pointer inside the DLL.
    assert(mqtt_monitoring_handler(Handler)),
    mqtt_connect(Handler, _Result),
    !;
    true.
```

The callbacks must have the name of the created client as prefix

```
production_monitoring_on_connect_success(Handler):-
    format('success connection of ~w~n', [production_monitoring]),
    mqtt_subscribe(Handler, 'production1/temperature', 1, _Result1).

    %subscribe the other related topics
```

production_monitoring_mqtt.pl

- When the connection is established, the `production_monitoring_on_connect_success(Handler)` predicate/callback is called, and the subscription to the topics take place.

```
production_monitoring_on_connect_success(Handler):-
    format('success connection of ~w~n', [production_monitoring]),
    mqtt_subscribe(Handler, 'production1/temperature', 1, _Result1).

    %subscribe the other related topics
```

- Once again, after subscription, future messages from the subscribed topics, are handled by corresponding callback

```
production_monitoring_on_message_arrived('production1/temperature', Message,_Handler):-
    format('mqtt topic: ~w~n',['production1/temperature']),
    format('mqtt message: ~w~n~n',[Message]).

% create the other topics on_message_arrived
```

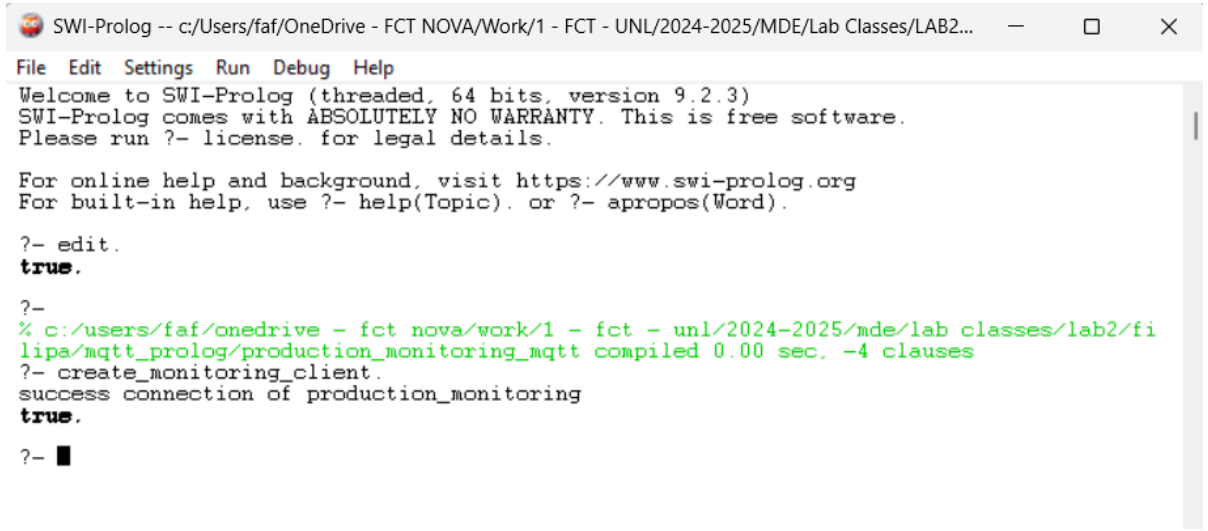
Again, the callbacks must have the name of the created client as prefix

Testing - Locally

Select the local mqtt_broker

```
%Defining the MQTT broker details
mqtt_broker('tcp://localhost:1883').          % case working locally
% mqtt_broker('tcp://192.168.250.201:1883'). % case working with lab1.3
                                              % MQTT Broker
```

- Compile and run the predicate:
  - `create_monitoring_client`



Testing - Locally

- Open a Command Prompt window, and simulate the publishing of the devices' consumption:

Command Prompt

```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\faf>mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "32.0"

C:\Users\faf>mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "22.0"

C:\Users\faf>mosquitto_pub -h localhost -p 1883 -t "production1/temperature" -m "30.0"

C:\Users\faf>
```

SWI-Prolog -- c:/Users/faf/OneDrive - FCT NOVA/Work/1 - FCT - UNL/2024-2025/MDE/Lab Classes/LAB2...

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- edit.
true.

?-
% c:/users/faf/onedrive - fct nova/work/1 - fct - unl/2024-2025/mde/lab classes/lab2/fi
lipa/mqtt_prolog/production_monitoring_mqtt compiled 0.00 sec, -4 clauses
?- create_monitoring_client.
success connection of production_monitoring
true.

?- mqtt topic: production1/temperature
mqtt message: 32.0

mqtt topic: production1/temperature
mqtt message: 22.0

mqtt topic: production1/temperature
mqtt message: 30.0
```

Testing - Remotely

- Select the remote mqtt_broker

```
%Defining the MQTT broker details
% mqtt_broker('tcp://localhost:1883').      % case working locally
mqtt_broker('tcp://192.168.250.201:1883'). % case working with lab1.3
                                             % MQTT Broker
```

- Compile and run the predicate:
  - `create_monitoring_client`

```
SWI-Prolog -- c:/Users/faf/OneDrive - FCT NOVA/Work/1 - FCT - UNL/2024-2025/MDE/Lab Classes/LAB2...
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- edit.
true.

?-
% c:/users/faf/onedrive - fct nova/work/1 - fct - unl/2024-2025/mde/lab classes/lab2/fi
lipa/mqtt_prolog/production_monitoring_mqtt compiled 0.00 sec, -4 clauses
?- create_monitoring_client.
success connection of production_monitoring
true.

?-
```

Testing - Remotely

SWI-Prolog -- c:/Users/faf/OneDrive - FCT NOVA/Work/1 - FCT - UNL/2024-2025/MDE/Lab Classes/LAB...

File Edit Settings Run Debug Help

```
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- edit.
true.

?-
% c:/users/faf/onedrive - fct nova/work/1 - fct - unl/2024-2025/mde/lab classes/lab2/fi
lipa/mqtt_prolog/production_monitoring_mqtt compiled 0.02 sec, -4 clauses
?- create_monitoring_client.
success connection of production_monitoring
true.

?- mqtt topic: production1/temperature
mqtt message: 42.85

mqtt topic: production1/temperature
mqtt message: 4.57

mqtt topic: production1/temperature
mqtt message: 12.99

mqtt topic: production1/temperature
mqtt message: 36.03

mqtt topic: production1/temperature
mqtt message: 31.95

mqtt topic: production1/temperature
mqtt message: 14.61

mqtt topic: production1/temperature
mqtt message: 14.22
```

Using a python script to simulate the devices publishing...

- You can find in CLIP the **`mqtt_publisher_script.py`** script that simulates the devices_monitoring topics publishing, by generating randomly float numbers between a min and max value.

- There you can find an example for one topic; you are invited to create other topics and publish the simulated payload.

- In case you do not have Python installed follow the guidelines (for Windows) in: https://learn.microsoft.com/en-us/windows/python/beginners

mqtt_publisher_script.py

```python
import paho.mqtt.client as mqtt_client
import random
import time

# MQTT broker details
broker_address = "127.0.0.1"
port = 1883

# Defining topics
topic_1 = "production1/temperature"
# add with other topics

def connect_mqtt():
    def on_connect(client, userdata, flags, reason_code, properties):
        if reason_code == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", reason_code)
    # Set Connecting Client ID
    client = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION2)
    client.on_connect = on_connect
    client.connect(broker_address, port)
    return client

# Function to generate random floats
def randomFloats(min=0.0, max=100.0):
    random_float = random.uniform(min, max)  # Generate a random float between min and max
    payload = "{:.2f}".format(random_float)   # Format the float to have 2 decimal places
    return payload

def publish(client):
    # Publish random floats to the MQTT topics every 5 seconds
    while True:
        payload_1 = randomFloats( min: 0.0,  max: 45.0)
        print(f"Topic: {topic_1} Publishing: {payload_1}")
        client.publish(topic_1, payload_1)  # Publish the payload to the topic_1
        # add the other topics publishing...

        time.sleep(5)  # Wait for 5 second before publishing the next float

def run():
```

Using a python script to simulate the devices publishing...

- To run the script, you should download it to your local folder, open a command prompt window and type: **`python mqtt_publisher_script.py`** as illustrated in the figure below.

```
Windows PowerShell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\faf\OneDrive - FCT NOVA\Work\1 - FCT - UNL\2023-2024\MDE\LABS\LAB2\Packages_students> python mqtt_publisher_
script.py
```

- You will probably get the following error:

```
PS C:\Users\faf\OneDrive - FCT NOVA\Work\1 - FCT - UNL\2023-2024\MDE\LABS\LAB2\Packages_students> python mqtt_publisher_
script.py
Traceback (most recent call last):
  File "C:\Users\faf\OneDrive - FCT NOVA\Work\1 - FCT - UNL\2023-2024\MDE\LABS\LAB2\Packages_students\mqtt_publisher_scr
ipt.py", line 1, in <module>
    import paho.mqtt.client as mqtt
ModuleNotFoundError: No module named 'paho'
```

Using a python script to simulate the devices publishing...

- This happens because you do not have installed the ‘paho’ module in your Python environment, to do that type: **`pip install paho-mqtt`**

```
PS C:\Users\faf\OneDrive - FCT NOVA\Work\1 - FCT - UNL\2023-2024\MDE\LABS\LAB2\Packages_students> pip install paho-mqtt
Defaulting to user installation because normal site-packages is not writeable
Collecting paho-mqtt
  Downloading paho_mqtt-2.1.0-py3-none-any.whl.metadata (23 kB)
Downloading paho_mqtt-2.1.0-py3-none-any.whl (67 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.2/67.2 kB 1.2 MB/s eta 0:00:00
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-2.1.0
```

- And run the script again:

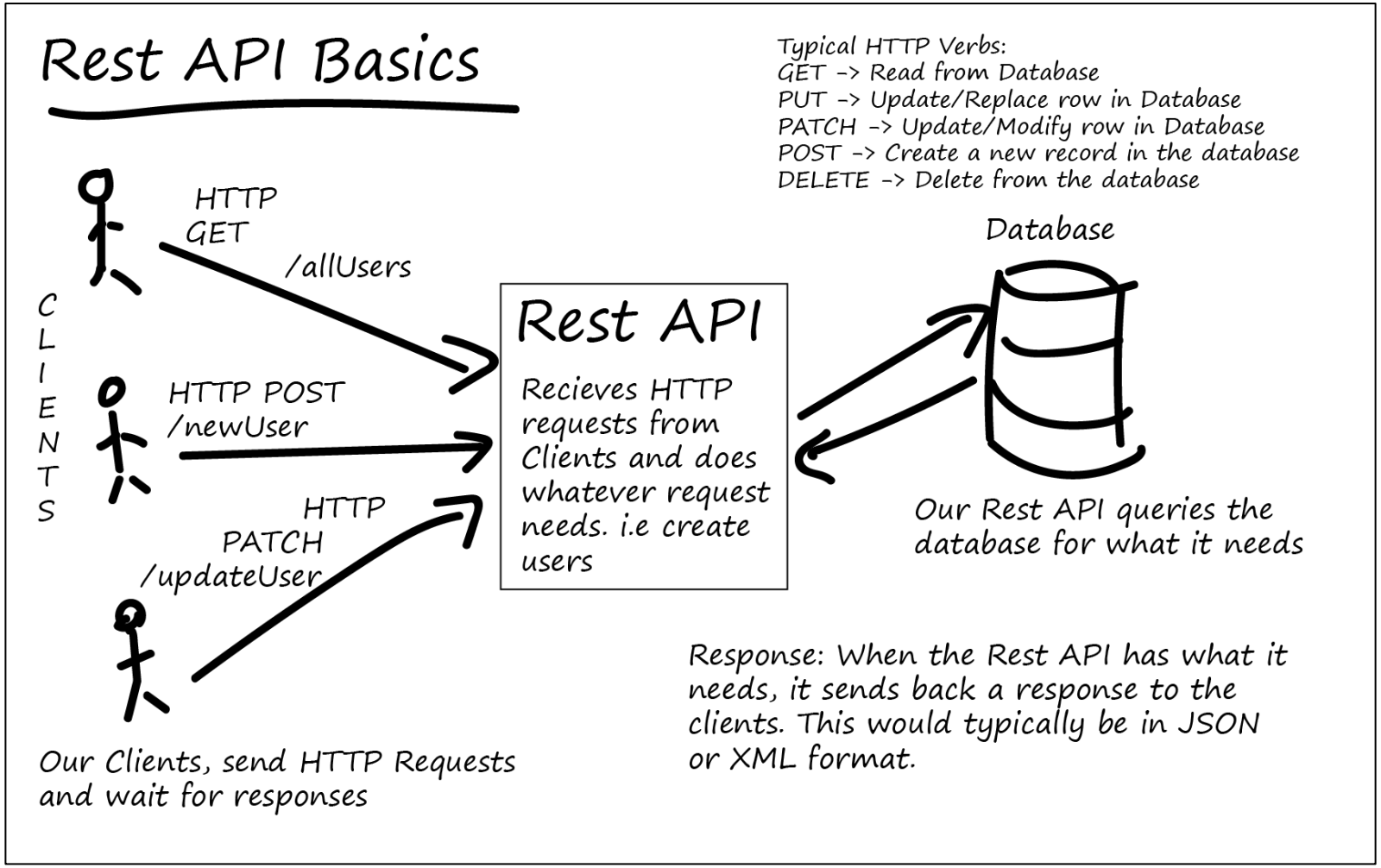
```
Windows PowerShell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PS

PS C:\Users\Admin> C:\Users\Admin\anaconda3\python.exe "C:\Users\Admin\OneDrive -
MDE\Lab Classes\LAB2\Filipa\mqtt_publisher_script.py"
Topic: production1/temperature Publishing: 38.92
Connected to MQTT Broker!
Topic: production1/temperature Publishing: 38.10
Topic: production1/temperature Publishing: 18.23
Topic: production1/temperature Publishing: 10.16
Topic: production1/temperature Publishing: 25.81
Topic: production1/temperature Publishing: 15.66
Topic: production1/temperature Publishing: 16.86
Topic: production1/temperature Publishing: 21.15
Topic: production1/temperature Publishing: 13.34
```

- Now it is easier to test your developments!! Check the subscriber -> production_monitoring !!

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

PROLOG HTTP SERVER INTEGRATION WITH JAVA (USING JSON)

HTTP Server



(https://tutorialedge.net/software-eng/what-is-a-rest-api/)

HTTP Server

- Start a new prolog file named **http_server.pl**
- Save it in the same folder where you have the installation_monitoring_mqtt.pl
- To use HTTP functionality, you need to add, in the beginning of your prolog file, the following libraries:
  - :- use_module(library(http/thread_httpd)).
  - :- use_module(library(http/http_dispatch)).
- Since we are using JSON format, add also:
  - :- use_module(library(http/http_json)).


```
% Required libraries
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/http_json)).
```

- To start a new HTTP server in SWI-Prolog, you need run the `start_server(Port)` predicate:
  - Port is the port_number where the application will be waiting for communications.

```
:- dynamic server_running/0.
% Starting our HTTP Server
start_server(Port):-
    \+ server_running,
    assert(server_running),
    http_server(http_dispatch, [port(Port)]),!;
    true.
```

HTTP Server

- Each service that you want to expose, you need to define the endpoint and the rule that will be triggered.
- So, in this example, we are creating a service that receives a name and replies with a hello message.
- Handler:

  :- http_handler('/hello', say_hello, []).

- Rule:

  say_hello(Request) :-

  member(search(Query), Request),

  memberchk(name=Name, Query),

  atom_concat("Hello: ", Name, MESSAGE),

  Response = _{ message: MESSAGE},

  reply_json(Response).

```
% Hello world (Receives a name and returns a message)
% Define the handler for the endpoint
:- http_handler('/hello', say_hello, []).

say_hello(Request) :-
    member(search(Query), Request),
    memberchk(name=Name, Query),
    atom_concat("Hello: ", Name, MESSAGE),
    Response = _{ message: MESSAGE},
    reply_json(Response).
```

HTTP Server

- Let’s run our server
  - This is done consulting the file where you have the server code:
    - File->Consult->http_server.pl
  - Running the start_server predicate with a port (in this case we used the 8001)

- To test the created service:
  - Open your browser (Chrome, Edge, Firefox, Safari, Opera, etc.)
  - And open the following link:
    - http://localhost:8001/hello?name=Andre
    - Try with your name! If your name is Andre, try with your neighbour's name!

SWI-Prolog (AMD64, Multi-threaded, version 9.2.3)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
%   library(error) compiled into error 0.00 sec, 100 clauses
%   library(option) compiled into swi_option 0.00 sec, 45 clauses
%   library(socket) compiled into socket 0.02 sec, 69 clauses
%   library(thread_pool) compiled into thread_pool 0.02 sec, 48 clauses
%   library(gensym) compiled into gensym 0.00 sec, 7 clauses
%      library(arithmetic) compiled into arithmetic 0.00 sec, 130 clauses
%     library(settings) compiled into settings 0.02 sec, 85 clauses
%    http_header compiled into http_header 0.06 sec, 777 clauses
%    http_stream compiled into http_stream 0.00 sec, 4 clauses
%    http_exception compiled into http_exception 0.00 sec, 36 clauses
%    library(broadcast) compiled into broadcast 0.00 sec, 12 clauses
%   http_wrapper compiled into httpd_wrapper 0.08 sec, 68 clauses
%   http_path compiled into http_path 0.00 sec, 45 clauses
%  library(http/thread_httpd) compiled into thread_httpd 0.13 sec, 156 clauses
%  library(http/http_dispatch) compiled into http_dispatch 0.02 sec, 198 clauses
%   library(http/http_client) compiled into http_client 0.00 sec, 53 clauses
%    library(record) compiled into record 0.02 sec, 71 clauses
%   library(http/json) compiled into json 0.03 sec, 273 clauses
%   library(memfile) compiled into memory_file 0.00 sec, 3 clauses
%  library(http/http_json) compiled into http_json 0.05 sec, 75 clauses
% c:/Users/faf/OneDrive - FCT NOVA/Work/1 - FCT - UNL/2023-2024/MDE/LABS/LAB2/Sol
uses
?- start_server(8001).
% Started server at http://localhost:8001/
true.

?-
```

localhost:8001/hello?name=An

localhost:8001/hello?name=Andre

Pretty-print ☑

```
{
  "message": "Hello: Andre"
}
```

HTTP Server

- The level of complexity, depends on the type of information we want to return:
  - String, Integer, List of Integers, List of Facts, List of Lists, etc.

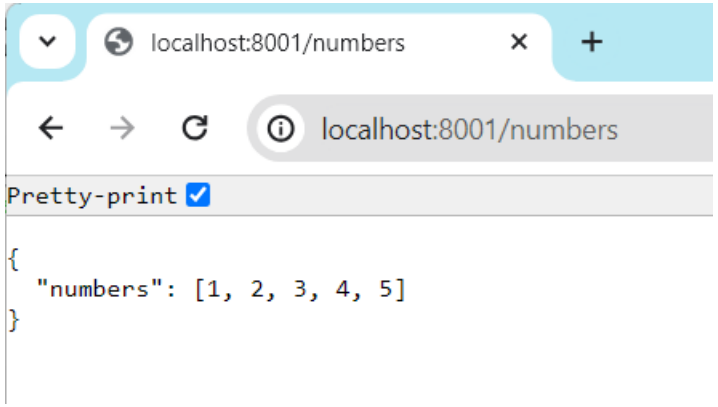
- Let's explore how can we return different types of data with the following examples:
  - List of Integers
  - List of Atoms ("Strings")
  - List of Facts
  - List of Lists

- Always using JSON.

HTTP Server

- This service returns a list of numbers:

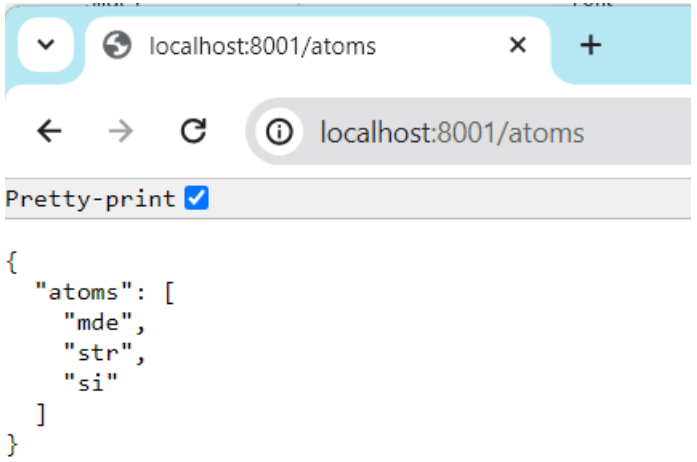
```
% List of numbers
% Define the handler for the endpoint
:- http_handler('/numbers', numbers_handler, []).

numbers_handler(_Request) :-
    % List of numbers
    Numbers = [1, 2, 3, 4, 5],
    % Create a dictionary with the list
    dict_create(JSON, _, [numbers-Numbers]),
    % Create JSON Object from the dictionary
    reply_json(JSON).
```



HTTP Server

- This service returns a list of Atoms:

```
% List of atoms
% Define the handler for the endpoint
:- http_handler('/atoms', atoms_handler, []).

atoms_handler(_Request) :-
    % List of Atoms
    Atoms = [mde, str, si],
    % Create strings from atoms
    maplist(atom_string, Atoms, Strings),
    % Create JSON Object from the dictionary
    reply_json(json([atoms=Strings])).
```



HTTP Server

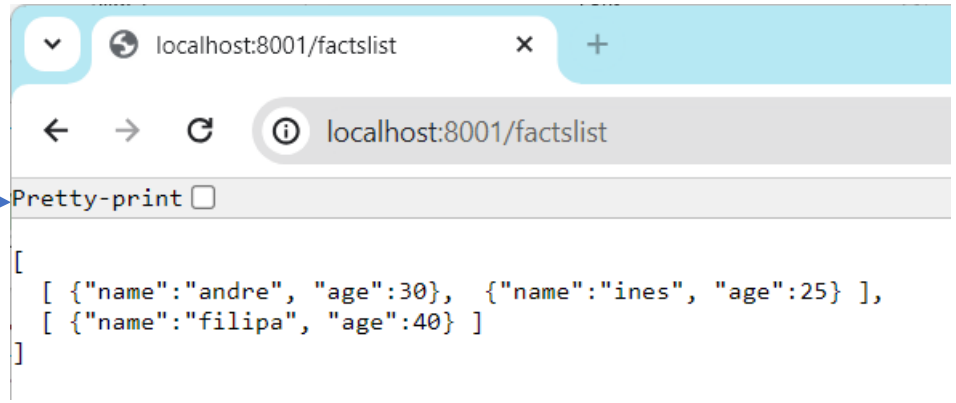
- This service returns a list of facts:
  - For this case is necessary to provide a rule to create the JSON format of the fact, in this case, `to_json` rule

```
% List of facts
% Define the handler for the endpoint
:- http_handler('/facts', facts_handler, []).

% Rule to create the JSON object
to_json(student(Name, Age), json([name=Name, age=Age])).

facts_handler(_Request) :-
    % List of facts
    FactList = [student(andre, 30), student(ines, 25), student(filipa, 40)],
    % Create a list of JSON objects from the list of facts
    maplist(to_json, FactList, JSONList),
    % Create JSON Object from the dictionary
    reply_json(json([students=JSONList])).
```

HTTP Server

- This service returns a list with lists of facts:
  - For this case is also necessary to provide a rule to create the JSON format of the fact, since it is the same type of facts, it is already coded in previous example.

```
% List of lists of facts
:- http_handler('/factslist', factslist_handler, []).

% Define the handler for the endpoint
factslist_handler(_Request) :-
    % List with lists of facts
    FactLists = [[student(andre, 30), student(ines, 25)], [student(filipa, 40)]],
    % Create lists of JSON objects from the list of facts
    maplist(maplist(to_json), FactLists, JSONLists),
    % Create the JSON object
    reply_json(JSONLists).
```



HTTP Client

- Now that you have the HTTP server ready, any application can call that services.
- Let's use a Java application to explore this capacity.
- In order to allow a Java project to call HTTP services with JSON objects, it is necessary to use additional libraries:
  - OkHTTP: This library allows a Java project to call HTTP services
  - GSON: This library allows a Java project to manipulate JSON objects

- You can find in CLIP, a Java project (HttpMaven) to open in VS Code (or IntelliJ if preferred).
- This project already has these libraries, so you will not need to add them.

- Let's call all the services to see how can we integrate a Java application with prolog.

HTTP Client

- To call the first service (hello), please add the following code within the main method (HttpMaven.java file).

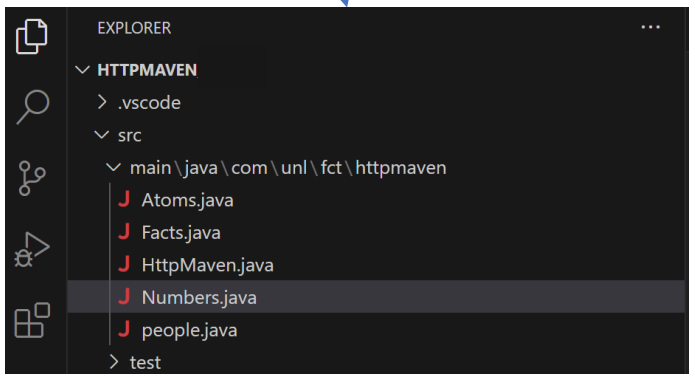
```java
public class HttpMaven {

    Run | Debug
    public static void main(String[] args) throws IOException {
        //Call service hello
        System.out.println(x:"HELLO SERVICE");
        String myName = "Andre";
        OkHttpClient clientHello = new OkHttpClient().newBuilder()
                .build();
        Request requestHello = new Request.Builder()
                .url("http://localhost:8001/hello?name=" + myName)
                .method("GET", null)
                .build();
        Response responseHello = clientHello.newCall(requestHello).execute();
        String stringHello = responseHello.body().string();
        System.out.println("Received String: " + stringHello);
        System.out.println(x:"-------------------------------------\n");
```

```
HELLO SERVICE
Received String: {"message":"Hello: Andre"}
-------------------------------------
```

HTTP Client

- To call numbers service, please add the following code within the main method (HttpMaven.java file).
  - To parse the JSON object to a Java object, we need to create a Java class with the same structure (already created in the provided project – Numbers.java class).

```java
//Call service numbers
System.out.println(x:"NUMBERS SERVICE");
OkHttpClient clientNumbers = new OkHttpClient().newBuilder()
        .build();
Request requestNumbers = new Request.Builder()
        .url("http://localhost:8001/numbers")
        .method("GET", null)
        .build();
Response responseNumbers = clientNumbers.newCall(requestNumbers).execute();
String stringNumbers = responseNumbers.body().string();
System.out.println("Received String: " + stringNumbers);
//Create the Java object from the received JSON
Gson gsonNumbers = new Gson();
Numbers numbersObject = gsonNumbers.fromJson(JsonParser.parseString(stringNumbers), classOfT:Numbers.class);
System.out.println("Array within the Numbers object: " + numbersObject.getMyNumbers());
System.out.println(x:"-------------------------------------\n");
```

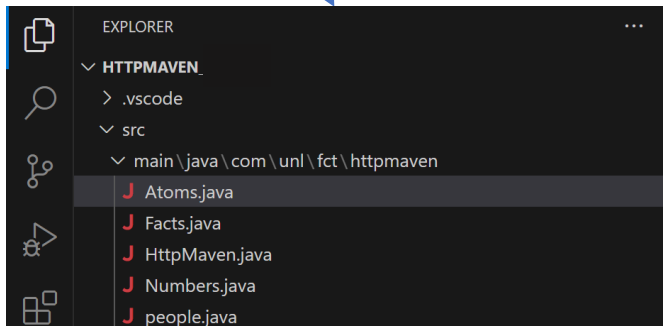



```
NUMBERS SERVICE
Received String: {"numbers": [1, 2, 3, 4, 5 ]}
Array within the Numbers object: [1, 2, 3, 4, 5]
-------------------------------------
```

HTTP Client

- To call atoms service, please add the following code within the main method (HttpMaven.java file).
  - Again, to parse the JSON object to a Java object, we need to create a Java class with the same structure (already created in the provided project – Atoms.java class).

```java
//Call service atoms
System.out.println(x:"ATOMS SERVICE");
OkHttpClient clientAtoms = new OkHttpClient().newBuilder()
        .build();
Request requestAtoms = new Request.Builder()
        .url("http://localhost:8001/atoms")
        .method("GET", null)
        .build();
Response responseAtoms = clientAtoms.newCall(requestAtoms).execute();
String stringAtoms = responseAtoms.body().string();
System.out.println("Received String: " + stringAtoms);
//Create the Java object from the received JSON
Gson gsonAtoms = new Gson();
Atoms atomsObject = gsonAtoms.fromJson(JsonParser.parseString(stringAtoms), classOfT:Atoms.class);
System.out.println("Array within the Atoms object: " + atomsObject.getMyAtoms());
System.out.println(x:"-----------------------------------\n");
```

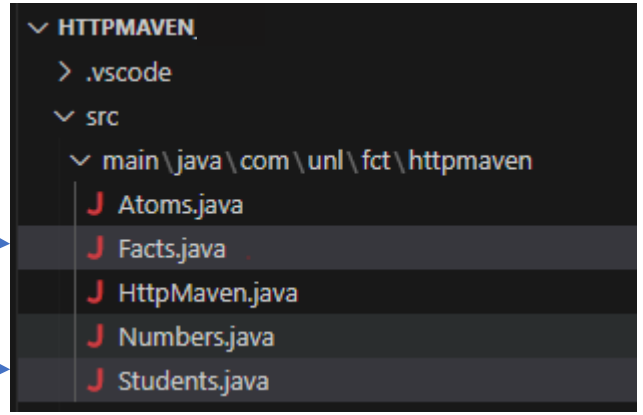



```
ATOMS SERVICE
Received String: {"atoms": ["mde", "str", "si" ]}
Array within the Atoms object: [mde, str, si]
```

HTTP Client

- To call facts service, please add the following code within the main method (HttpMaven.java file).
  - Again, to parse the JSON object to a Java object, we need to create the Java classes with the same structure (already created in the provided project – Students.java and Facts.java classes).

```java
//Call service facts
System.out.println(x:"FACTS SERVICE");
OkHttpClient clientFacts = new OkHttpClient().newBuilder()
        .build();
Request requestFacts = new Request.Builder()
        .url(arg0:"http://localhost:8001/facts")
        .method(arg0:"GET", arg1:null)
        .build();
Response responseFacts = clientFacts.newCall(requestFacts).execute();
String stringFacts = responseFacts.body().string();
//stringFacts = stringFacts.replace("\"", "'");
System.out.println("Received String: " + stringFacts);
//Create the Java object from the received JSON
Gson gsonFacts = new Gson();
// Define the type of the ArrayList<Students>
Type listType = new TypeToken<Facts>(){}.getType();
// Deserialize JSON string to Facts object
Facts factsObject = gsonFacts.fromJson(stringFacts, listType);
System.out.println(x:"Array within Facts object: ");
factsObject.printListOfStudents();
System.out.println(x:"-----------------------------------\n");
```

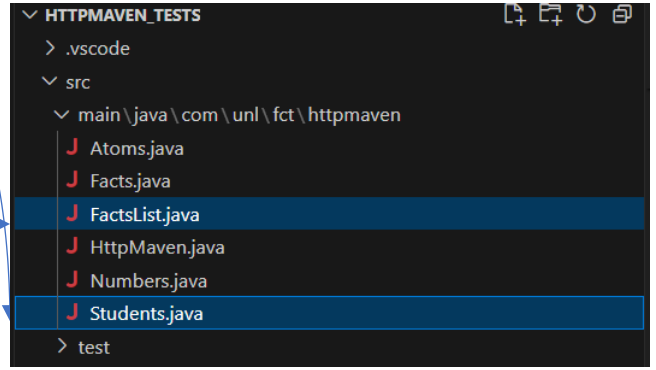



```
FACTS SERVICE
Received String: {
  "students": [
    {"name":"andre", "age":30},
    {"name":"ines", "age":25},
    {"name":"filipa", "age":40}
  ]
}
Array within Facts object:
student(andre,30)
student(ines,25)
student(filipa,40)
```

HTTP Client

- To call factsList service, please add the following code within the main method (HttpMaven.java file).
  - In this case, as we do not have a key associated to the values, we can parse the JSON Object to an ArrayList of ArrayList<Students> directly (using the class already provided in the project – Students.java) and a FactsList.java that is used as an utils class for printing the facts.

```java
//Call service factslist
System.out.println(x:"FACTSLIST SERVICE");
OkHttpClient clientFactsList = new OkHttpClient().newBuilder()
        .build();
Request requestFactsList = new Request.Builder()
        .url(arg0:"http://localhost:8001/factslist")
        .method(arg0:"GET", arg1:null)
        .build();
Response responseFactsList = clientFactsList.newCall(requestFactsList).execute();
String stringFactsList = responseFactsList.body().string();
System.out.println("Received String: " + stringFactsList);
//In this case our JSON does not have the name of the value, so we can assert directly to an ArrayList
Gson gsonFactsList = new Gson();
// Define the type of the ArrayList<ArrayList<Students>>
Type listListType = new TypeToken<ArrayList<ArrayList<Students>>>(){}.getType();
ArrayList<ArrayList<Students>> myArray = gsonFactsList.fromJson(stringFactsList, listListType);
System.out.print(s:"Array within the list of Facts: "); FactsList.printFactsList(myArray);
System.out.print(s:"First position of the list: "); FactsList.printStudents(myArray.get(index:0)); System.out.println();
System.out.print(s:"Second position of the list: "); FactsList.printStudents(myArray.get(index:1)); System.out.println();
System.out.println(x:"-------------------------------------\n");
```



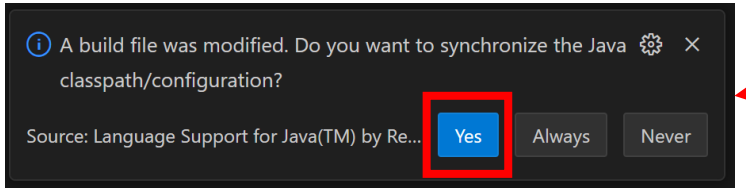

```
FACTSLIST SERVICE
Received String: [
  [ {"name":"andre", "age":30},  {"name":"ines", "age":25} ],
  [ {"name":"filipa", "age":40} ]
]
Array within the list of Facts: [[{andre,30},{ines,25}],[{filipa,40}]]
First position of the list: [{andre,30},{ines,25}]
Second position of the list: [{filipa,40}]
-------------------------------------
```

Running Issues…

- When running your java project, if you get an error like this:

```
PS C:\Users\faf\OneDrive - FCT NOVA\Work\1 - FCT - UNL\2023-2024\MDE\LABS\LAB2\Packages_students\HttpMaven>  & 'C:\Users\faf\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.4.101-hot
spot\bin\java.exe' '@C:\Users\faf\AppData\Local\Temp\cp_4yzv8603noijxt1s961dwb1u9.argfile' 'com.unl.fct.httpmaven.HttpMaven'
Error: LinkageError occurred while loading main class com.unl.fct.httpmaven.HttpMaven
        java.lang.UnsupportedClassVersionError: com/unl/fct/httpmaven/HttpMaven has been compiled by a more recent version of the Java Runtime (class file version 62.0), this version
of the Java Runtime only recognizes class file versions up to 61.0
```

- You need to update the POM file with your JDK version:
  - Usually, you can see your current JDK version in the error message…
  - You will be asked to “update” your project with the new pom configuration



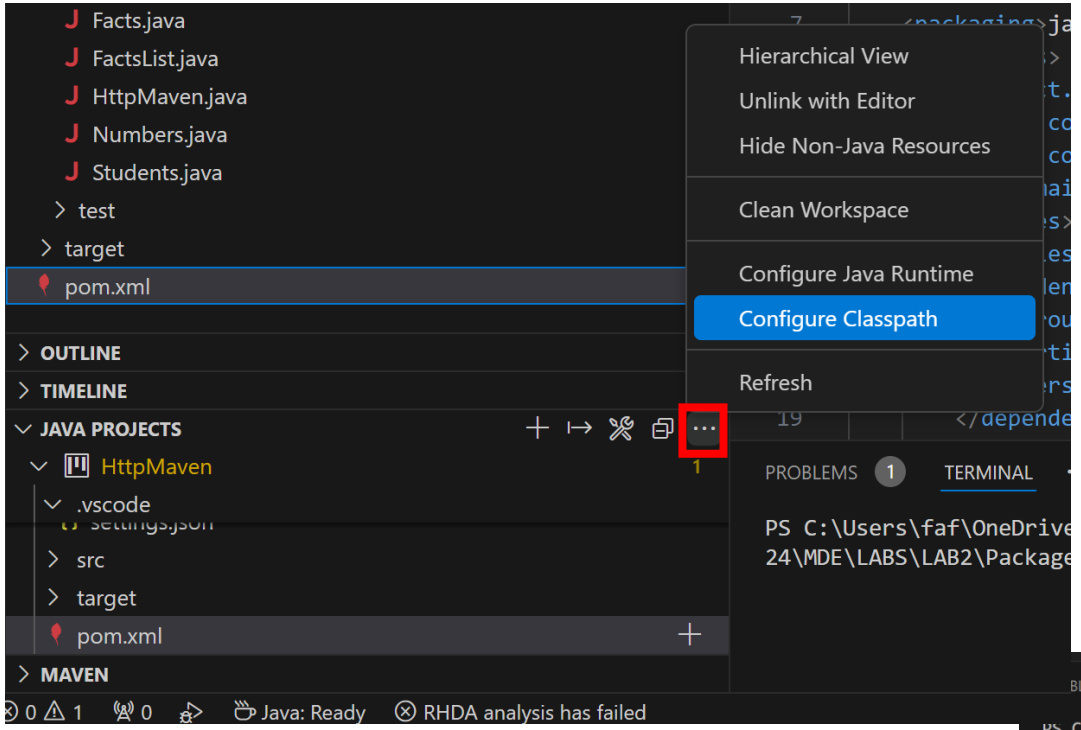
pom.xml

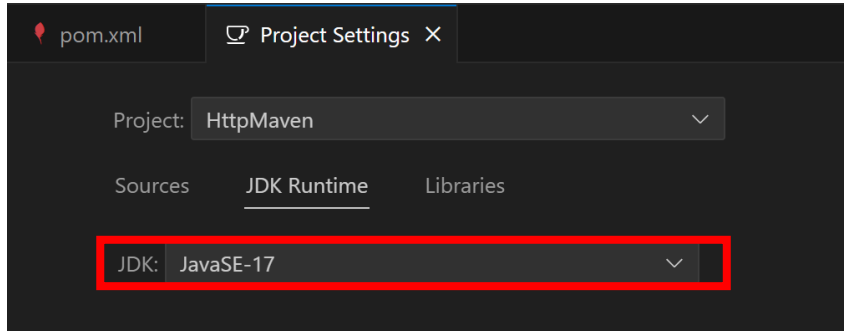
pom.xml > XSLT/XPath for Visual Studio Code > project

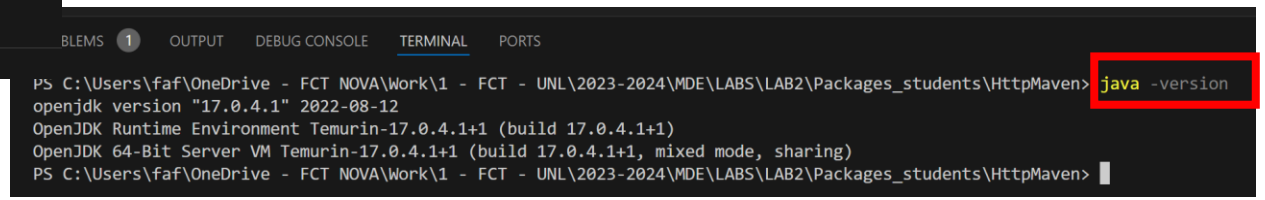
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.unl.fct</groupId>
    <artifactId>HttpMaven</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <exec.mainClass>com.unl.fct.httpmaven.HttpMaven</exec.mainClass>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
            <version>4.9.1</version>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.7</version>
        </dependency>
    </dependencies>
</project>
```

Running Issues…

- If, for any reason, you don't know your JDK, please go to JAVA PROJECTS, click on the "3 dots button" and select ***Configure Classpath***.







- Another way, is to type in terminal: java –version
- As soon as you update POM file adequately, the project will run normally.

- ✓ At this moment, you have all tools to develop your work!
- ✓ Don’t get late!!!!