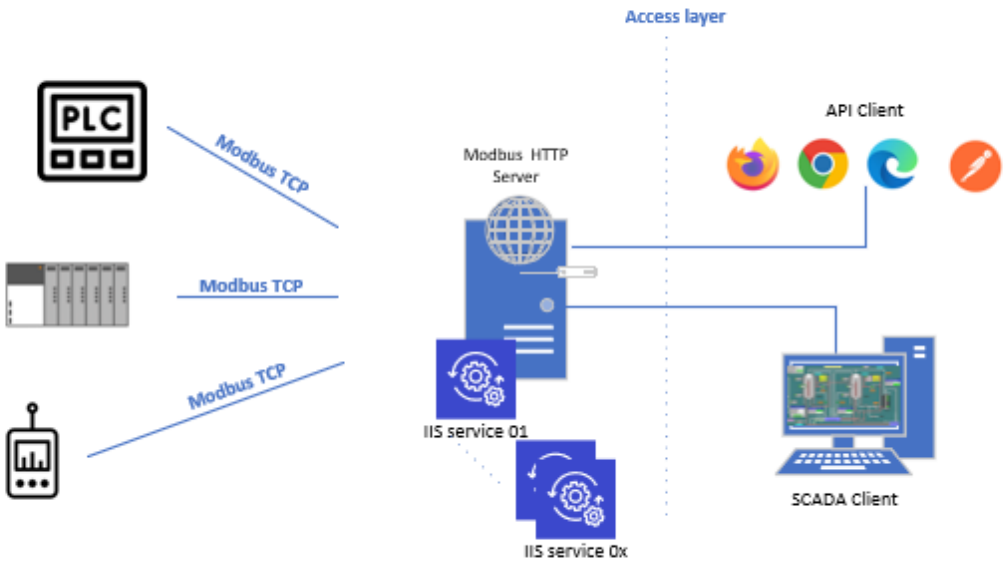
Modbus TCP to Restful API

User guide

Contents

Modbus TCP to Restful API

Modbus TCP to API is a IIs service to convert the Modus TCP to RESTful API where can IT team can access the industrial devices and also to add security for accessing the IOT devise based on username and password.



1. System Requirements

Operating System Requirements:

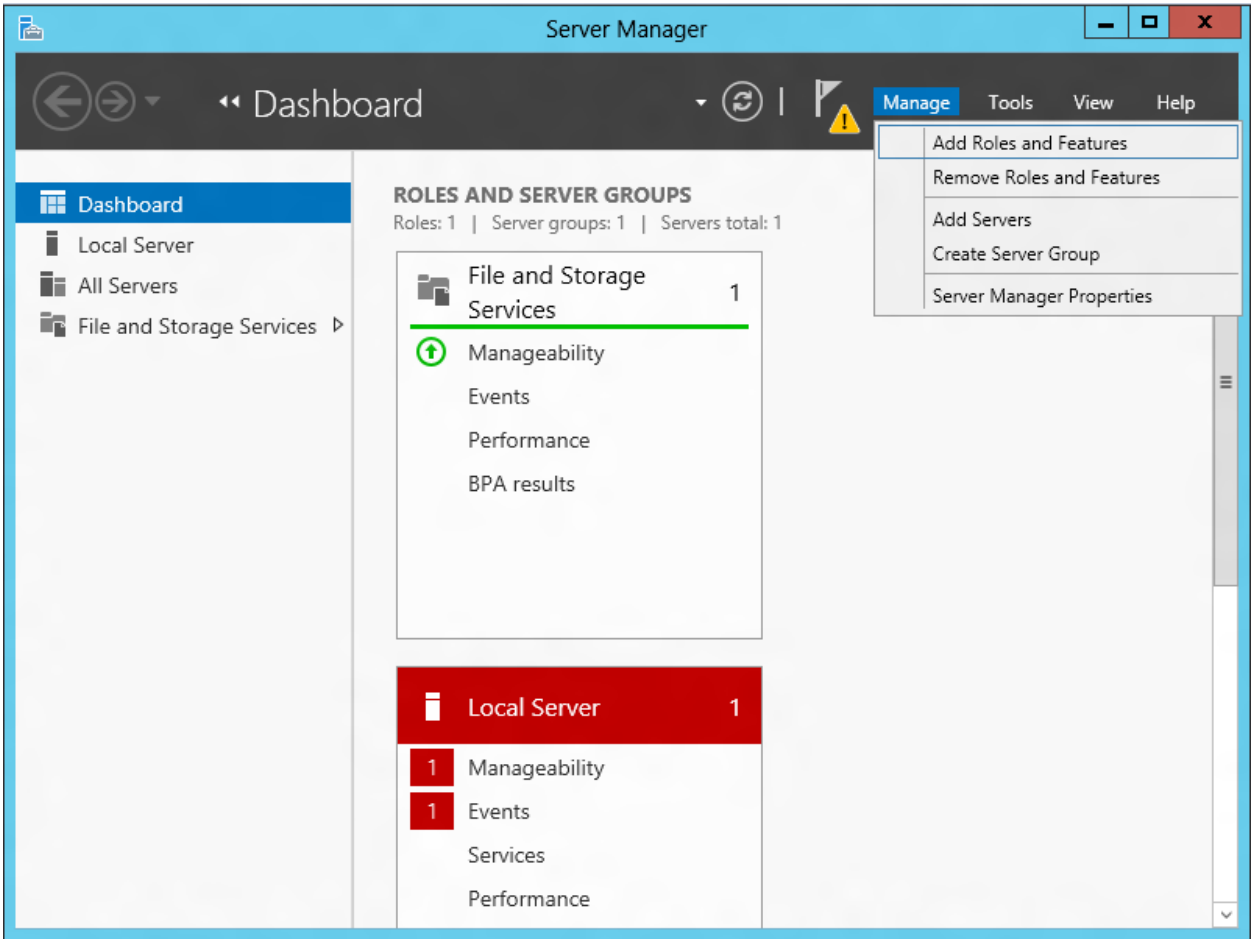
- Windows Server 2012
- Windows Server 2019
- Windows Server 2022
- Windows 10
- Windows 11

2. Installation steps

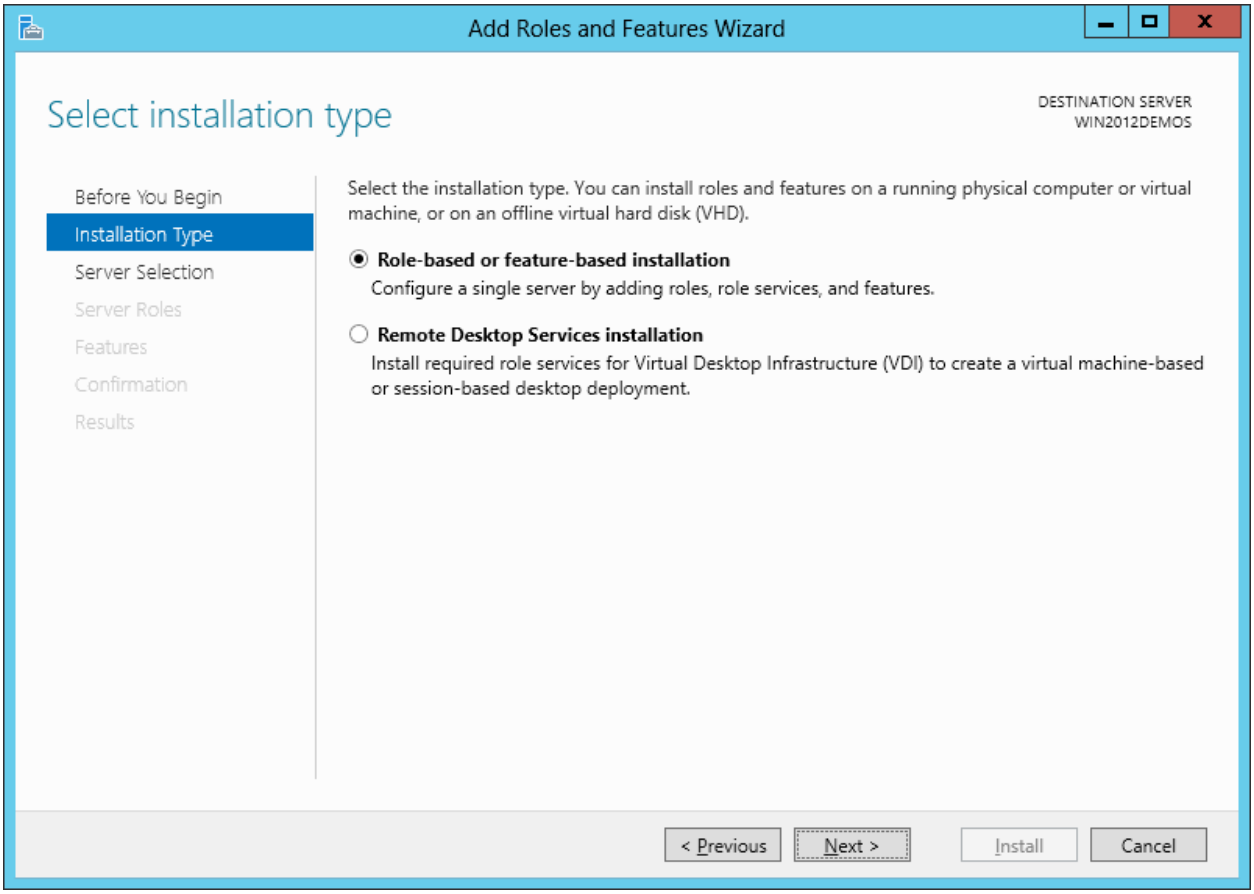
2.1. Install IIS on the windows server or on window professional.

2.1.1. Install IIS on the windows server.

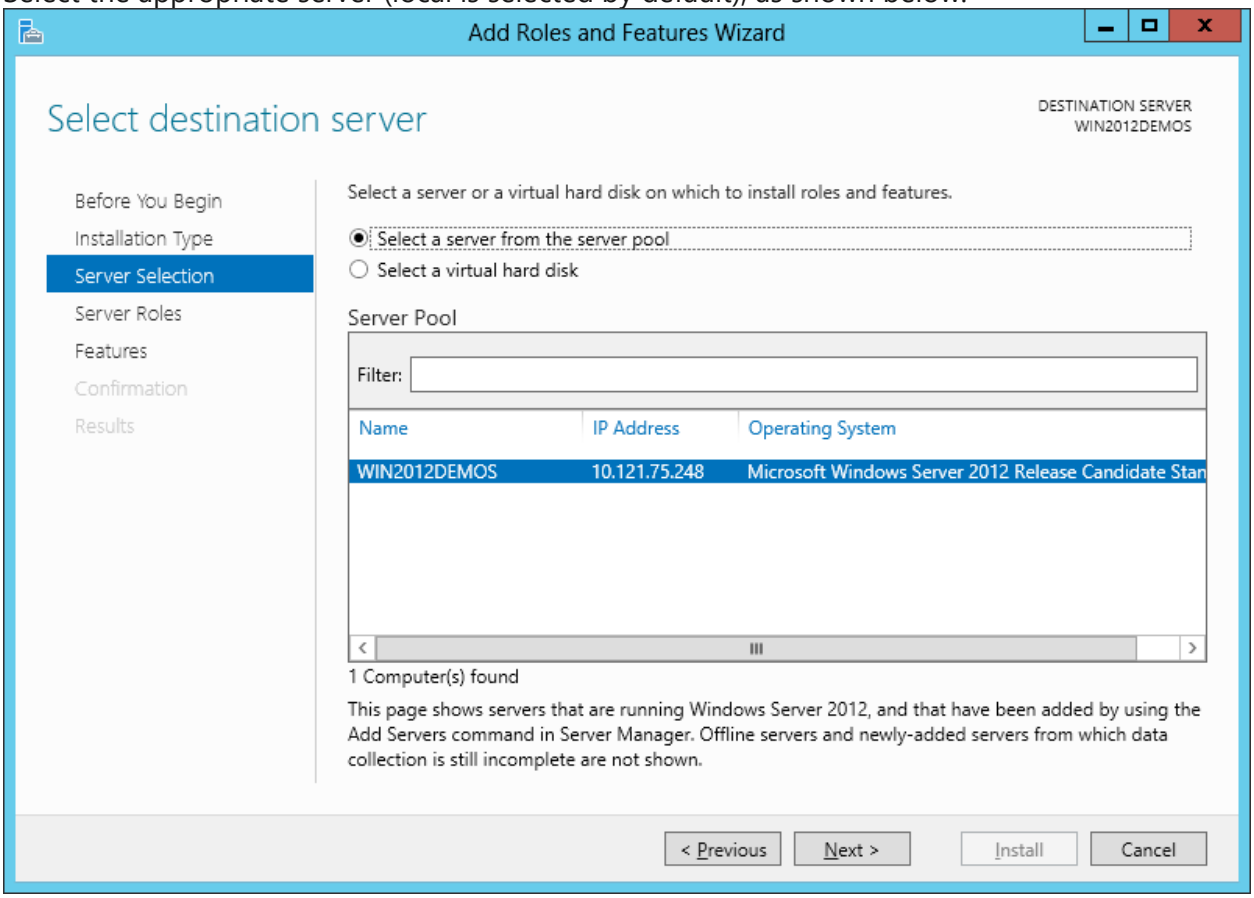
1. Open **Server Manager**.
2. Under **Manage** menu, select **Add Roles and Features**:



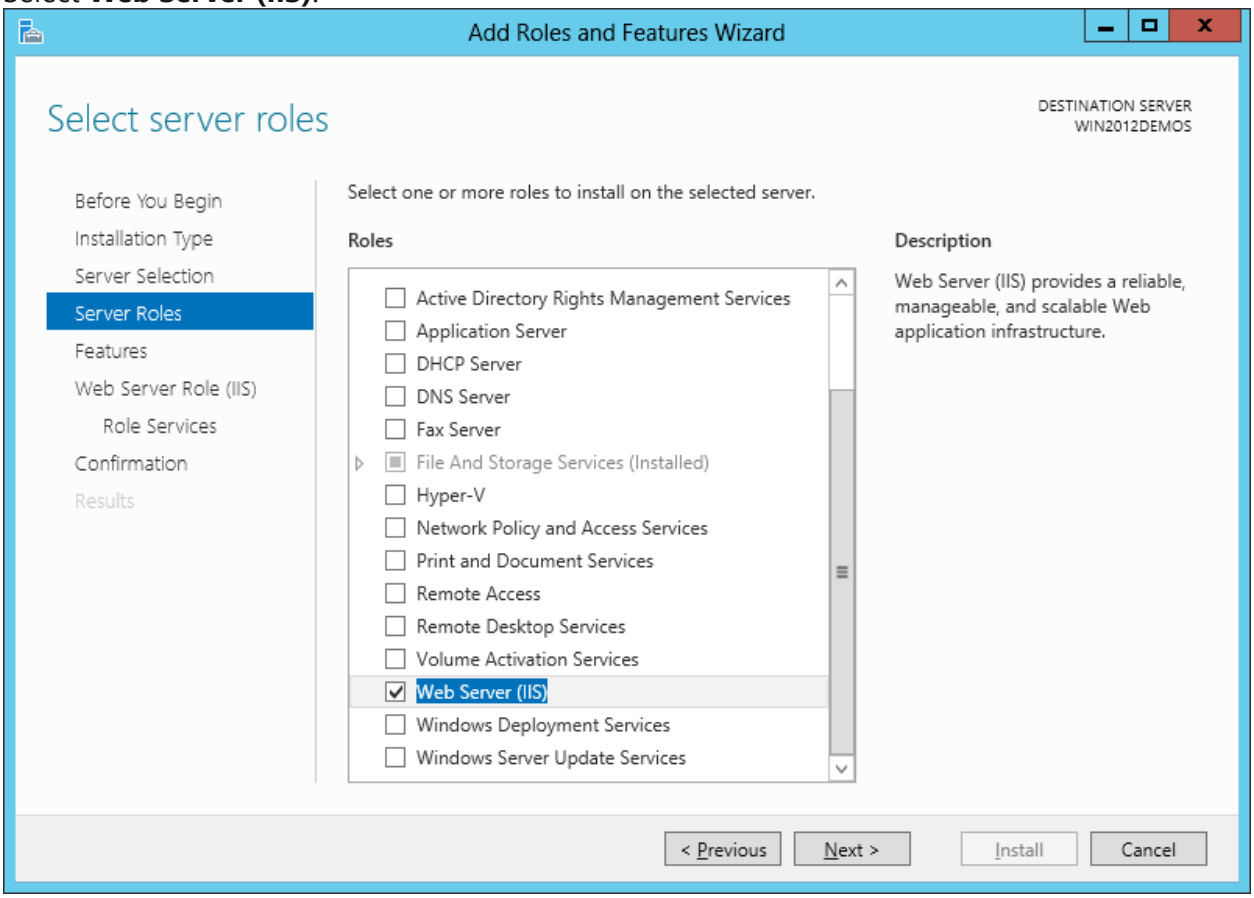
3. Select **Role-based or Feature-based Installation**:



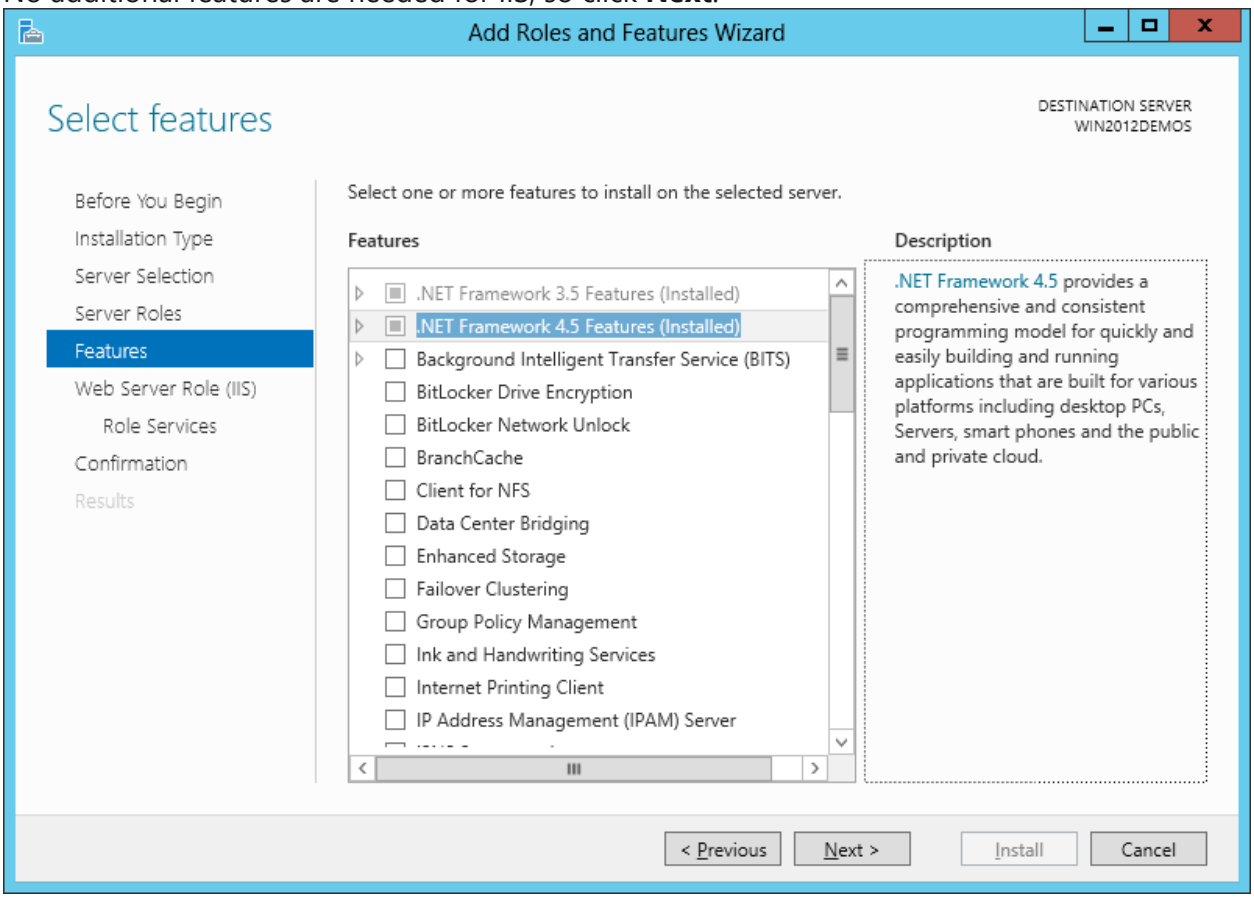
4. Select the appropriate server (local is selected by default), as shown below:



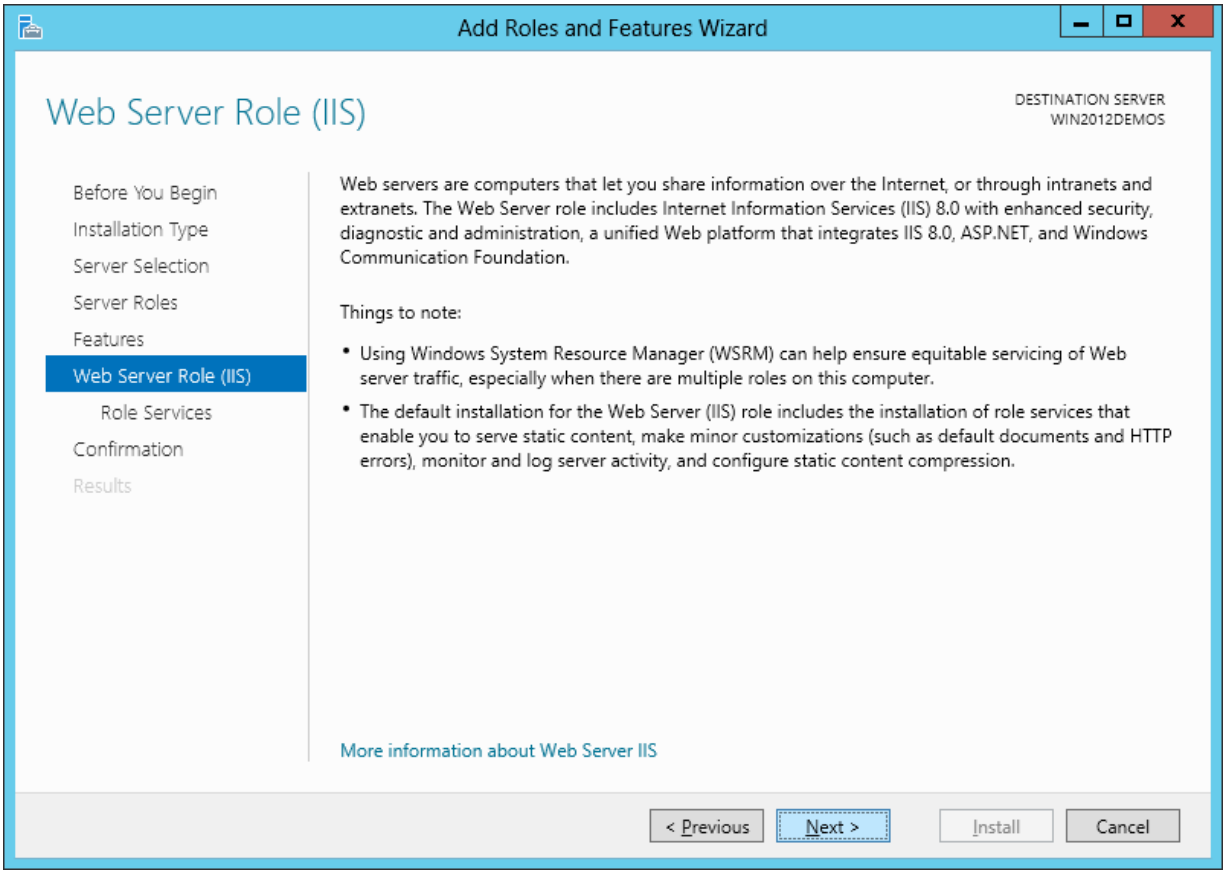
5. Select **Web Server (IIS)**:



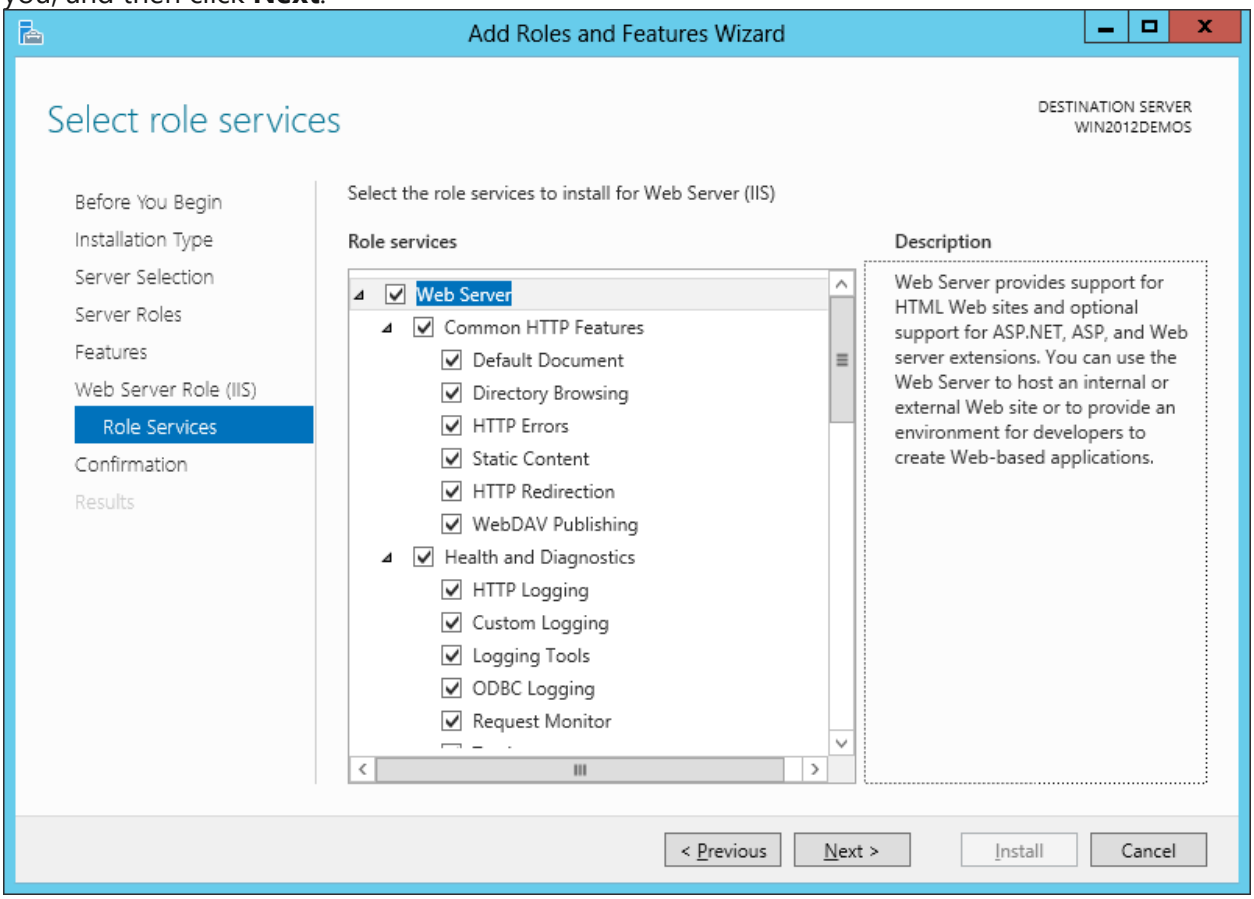
6. No additional features are needed for IIS, so click **Next**:



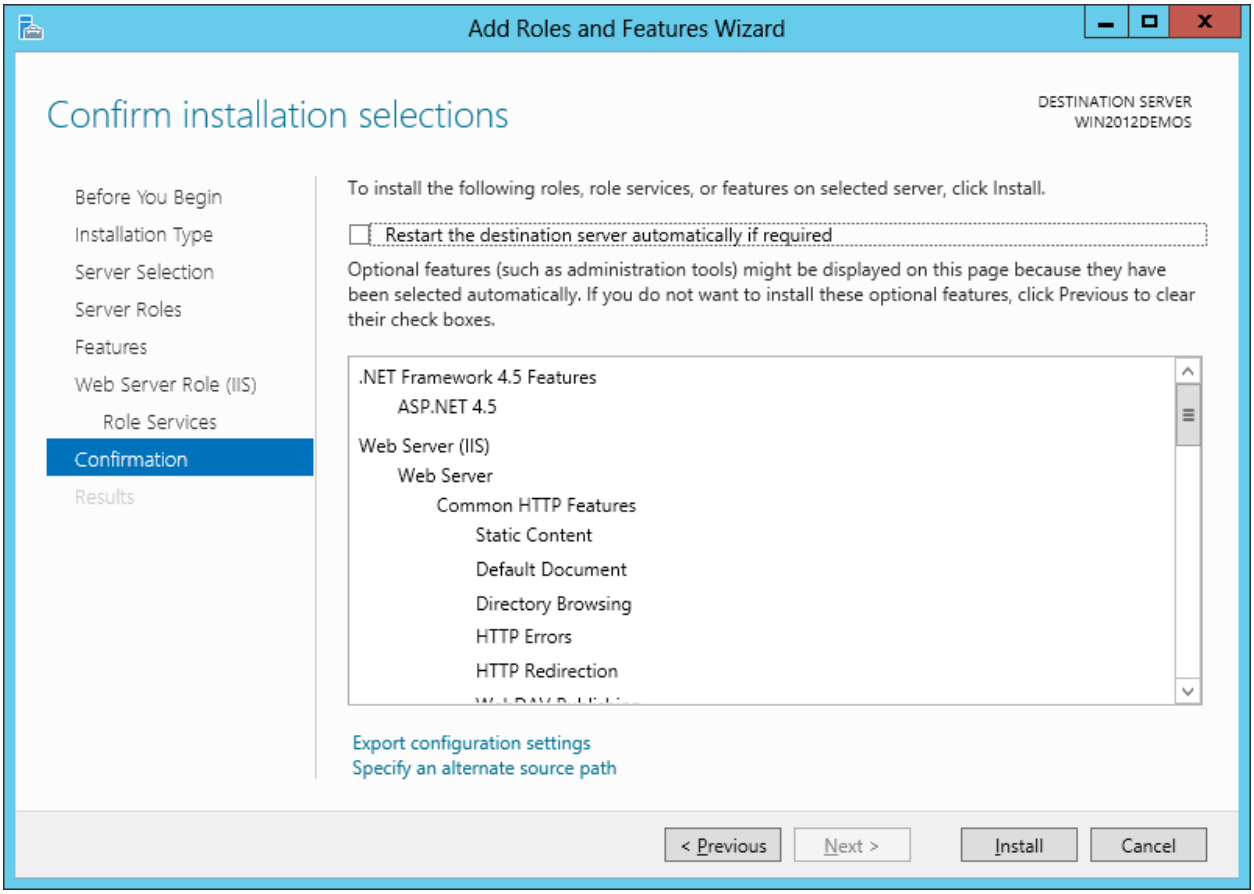
7. Click **Next**:


Add Roles and Features Wizard
Web Server Role (IIS)
DESTINATION SERVER
WIN2012DEMOS
Before You Begin
Installation Type
Server Selection
Server Roles
Features
Web Server Role (IIS)
Role Services
Confirmation
Results
Web servers are computers that let you share information over the Internet, or through intranets and extranets. The Web Server role includes Internet Information Services (IIS) 8.0 with enhanced security, diagnostic and administration, a unified Web platform that integrates IIS 8.0, ASP.NET, and Windows Communication Foundation.
Things to note:
Using Windows System Resource Manager (WSRM) can help ensure equitable servicing of Web server traffic, especially when there are multiple roles on this computer.
The default installation for the Web Server (IIS) role includes the installation of role services that enable you to serve static content, make minor customizations (such as default documents and HTTP errors), monitor and log server activity, and configure static content compression.
More information about Web Server IIS
< Previous
Next >
Install
Cancel

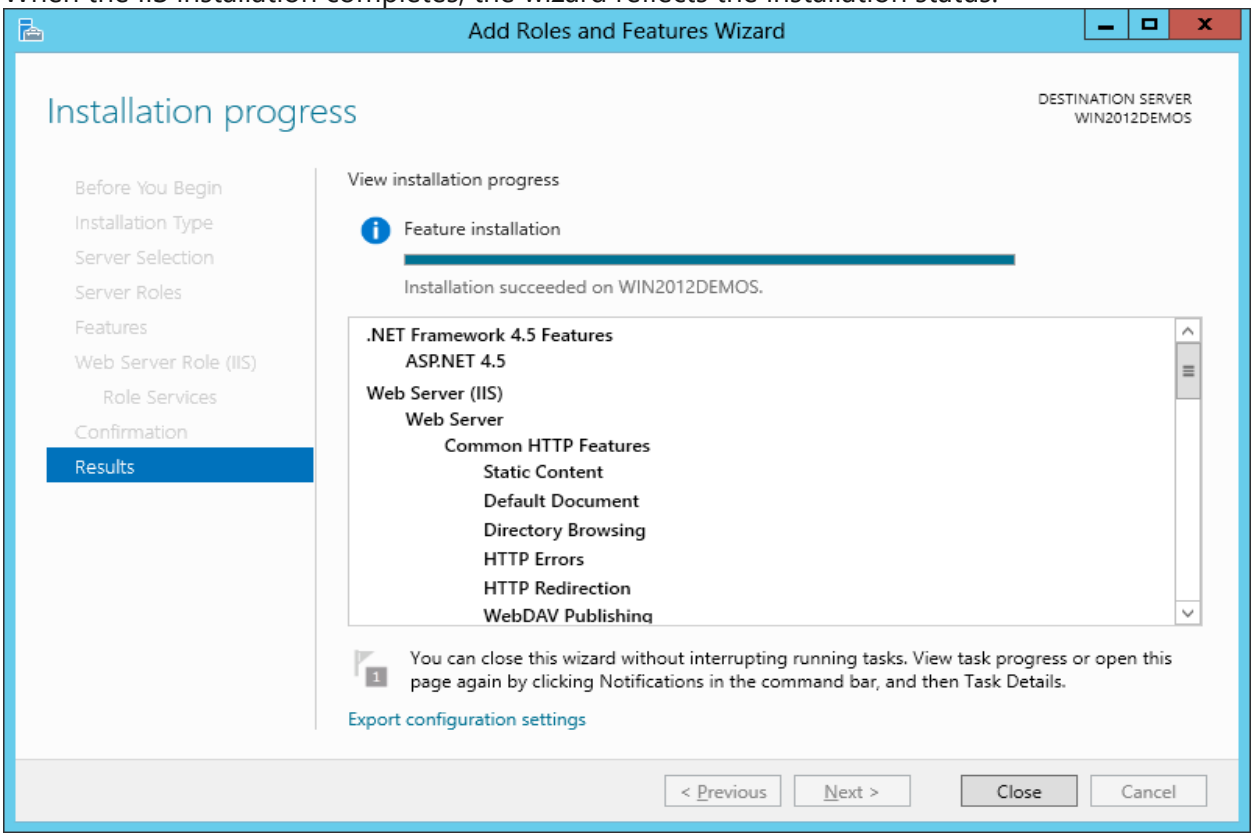
8. Customize your installation of IIS, or accept the default settings that have already been selected for you, and then click **Next**:



9. Click **Install**:



10. When the IIS installation completes, the wizard reflects the installation status:

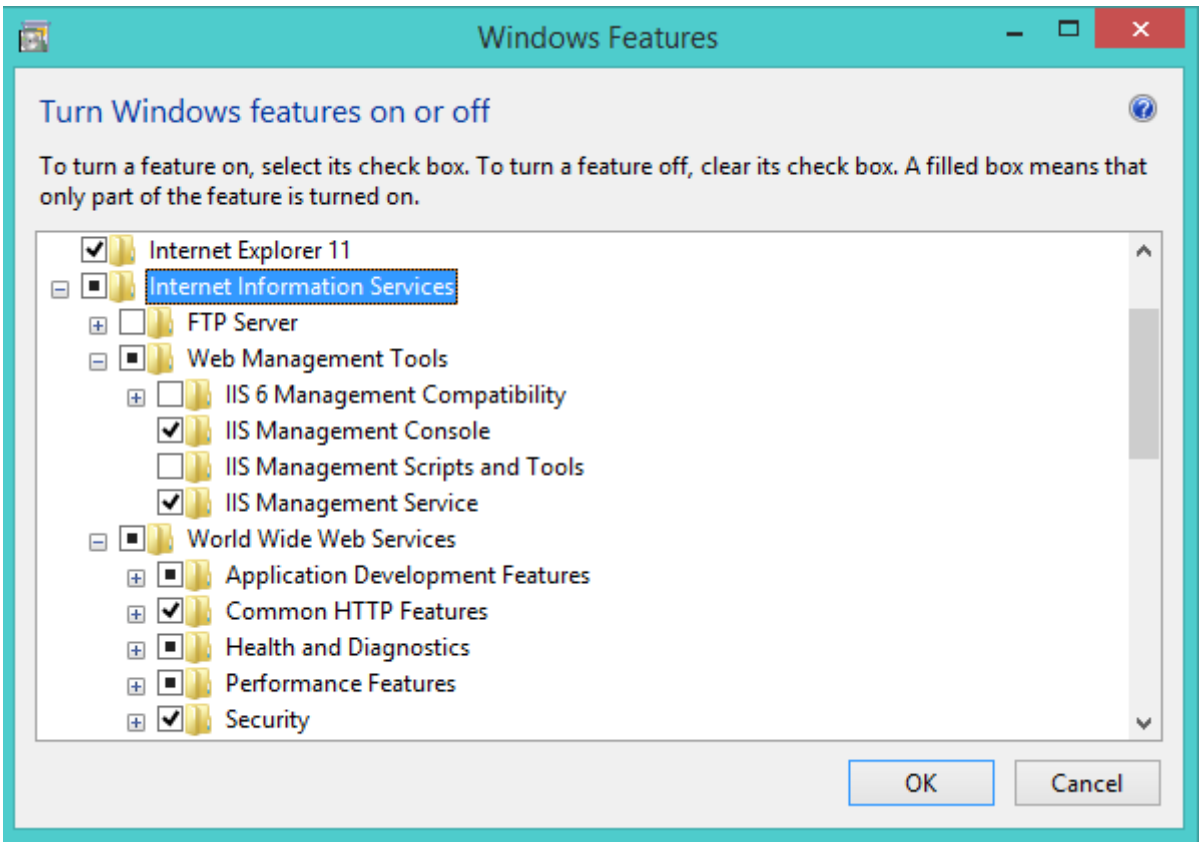


11. Click **Close** to exit the wizard.

2.1.2. install IIS on Windows 10

1. Open the Control Panel and click on "Programs."
2. Click on "Turn Windows features on or off."
3. Scroll down and find "Internet Information Services" and check the box.
4. Click on "OK" and wait for the installation to complete.
5. Once the installation is finished, open your web browser and type "localhost" to verify that IIS is running.

By following these steps, you can easily install IIS on Windows 10 and start hosting your websites.



2.1.3. Setup .Net core

You can download the two packages from the following links or can be found in the prerequisite folder in the setup package

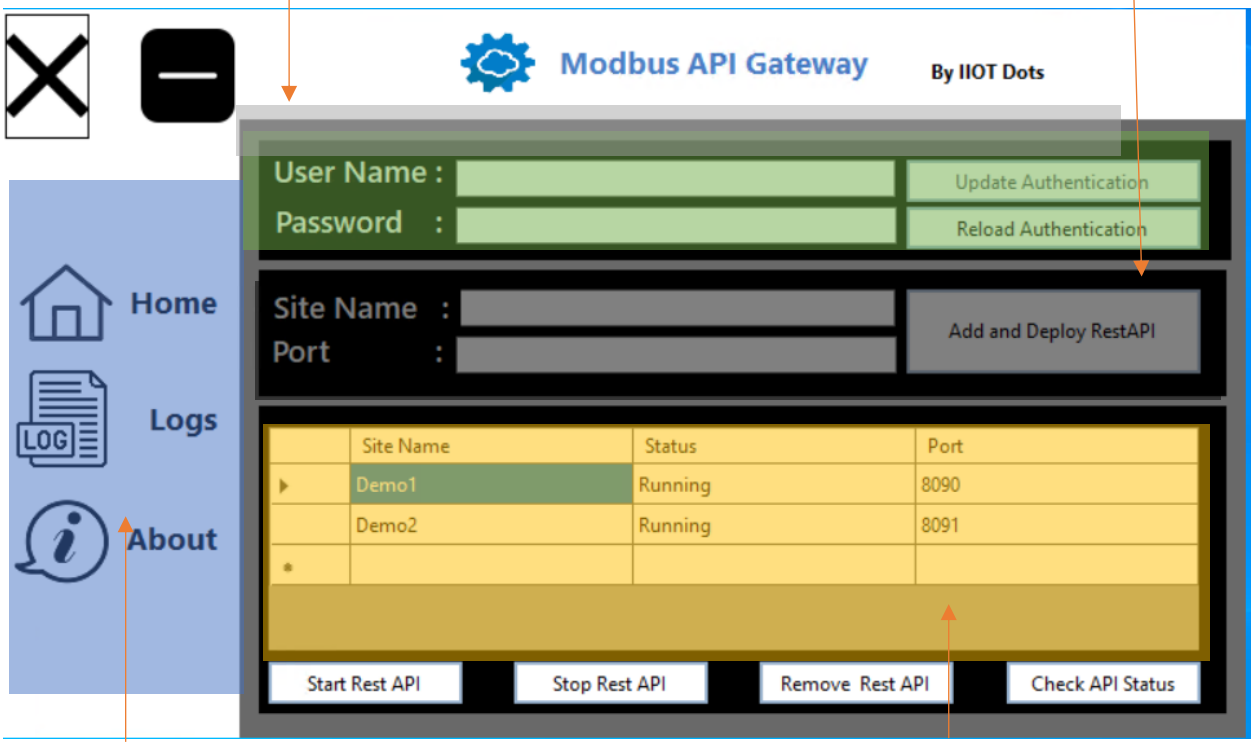
1- Download the .net 8 hosting bundle
https://dotnet.microsoft.com/permalink/dotnetcore-current-windows-runtime-bundle-installer
2- .NET 8.0 Desktop Runtime (v8.0.8) x64 to be installed and can be downloaded from the following link:

https://download.visualstudio.microsoft.com/download/pr/27bcdd70-ce64-4049-ba24-2b14f9267729/d4a435e55182ce5424a7204c2cf2b3ea/windowsdesktop-runtime-8.0.11-win-x64.exe

3. Application Overview

Authentication configuration

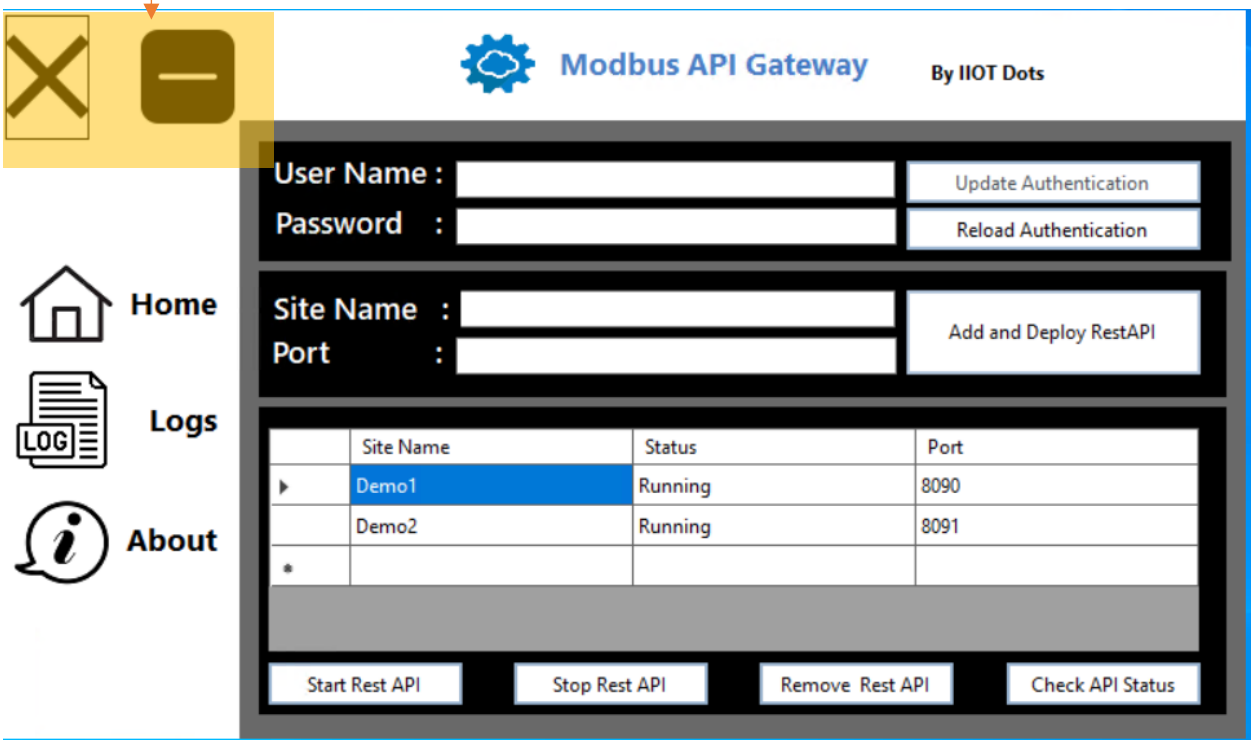
Add new API



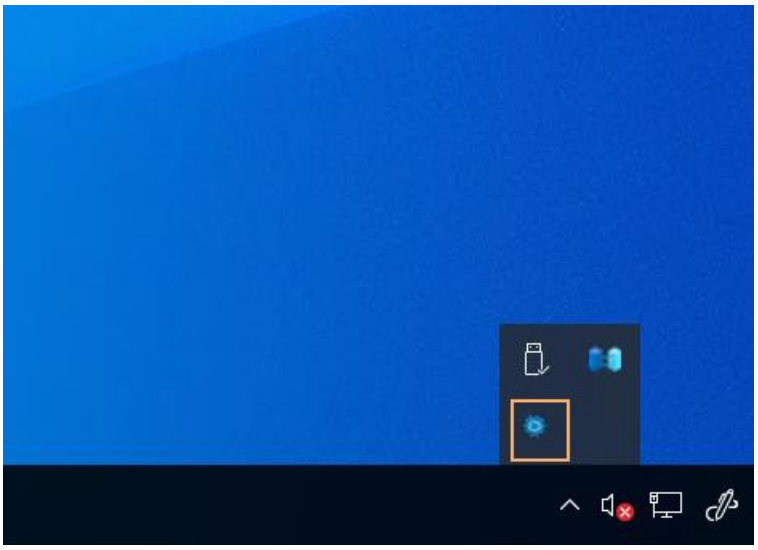
Side Menu Area

Configured RESTful APIs Area and control

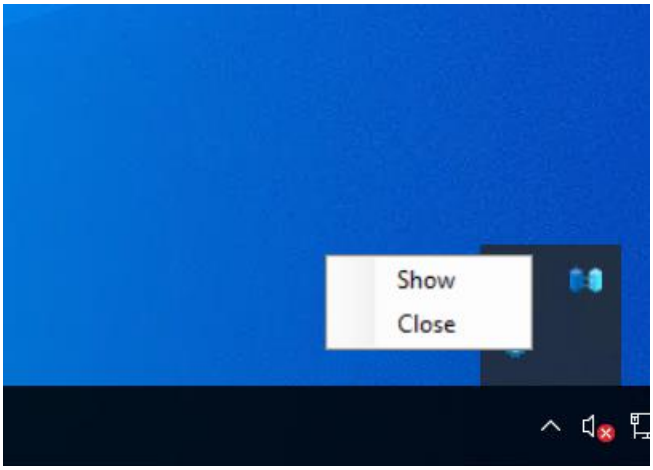
Minimize -Close window.



When the window closed the application will be minimized in the notification menu

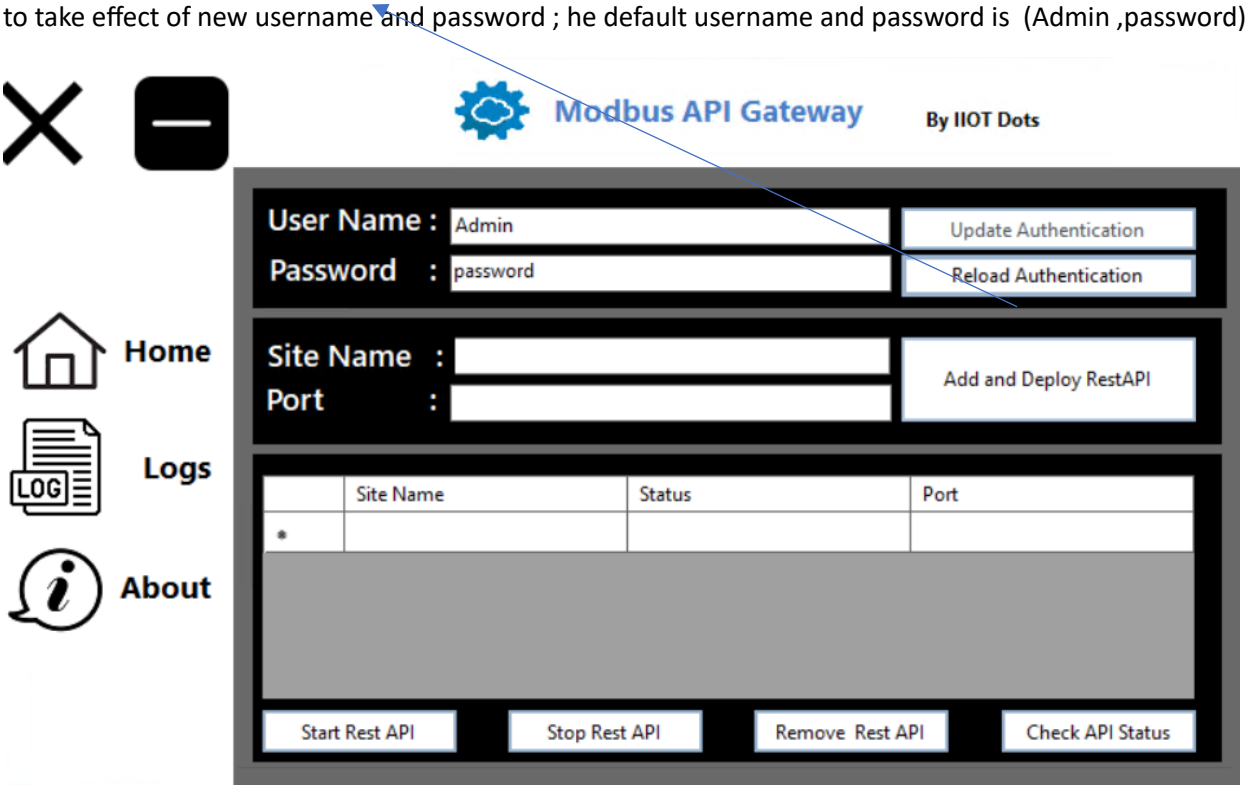


By write click on it , you can close the program or to open the window

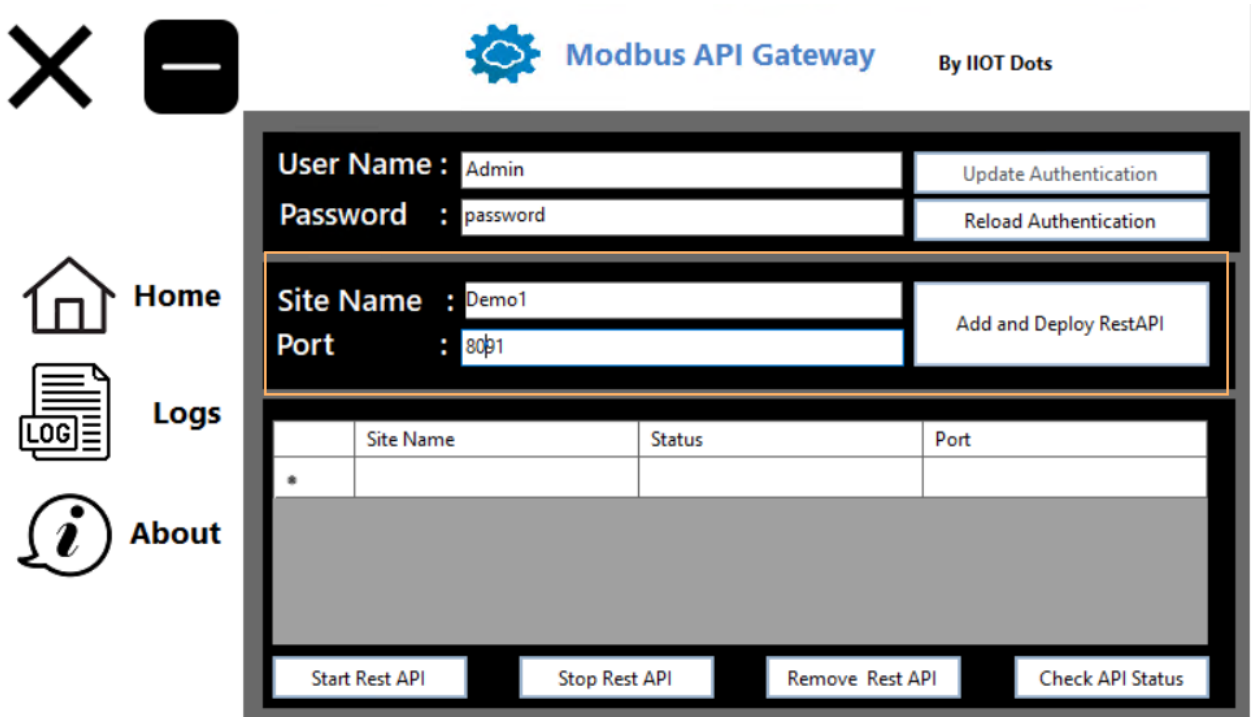


4. Quick Start up

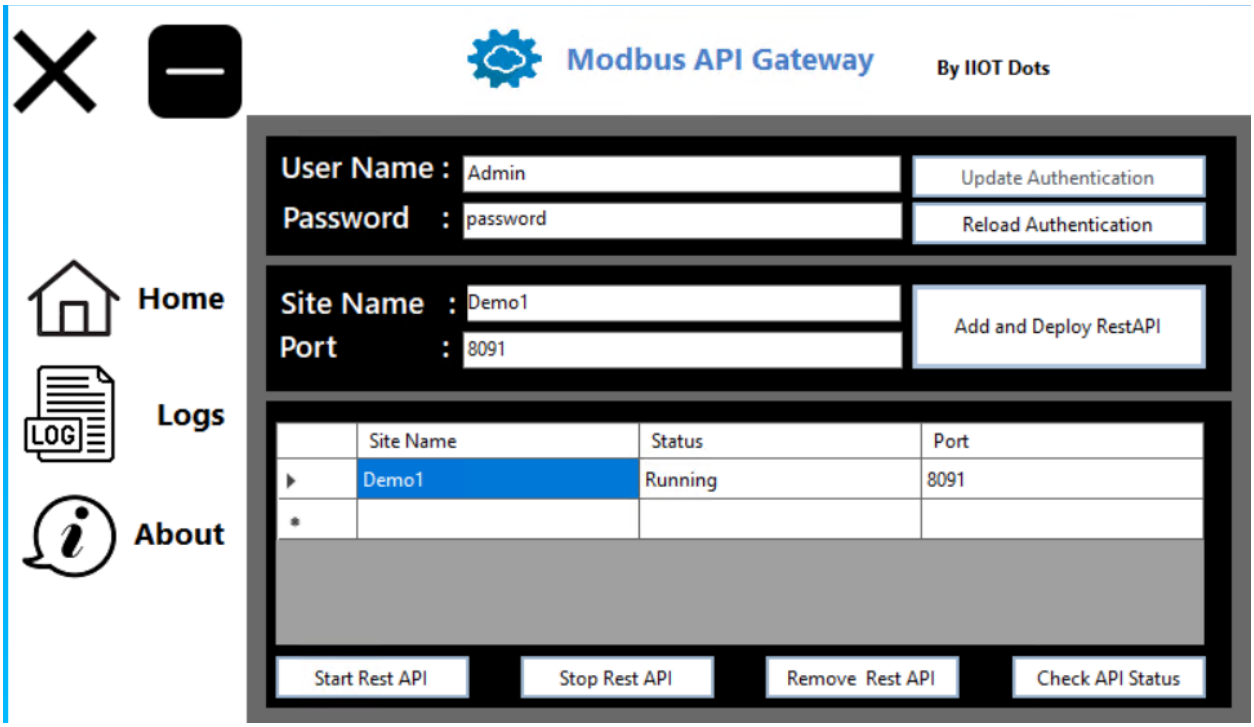
Step 1:retrieve the default authentication and you can change it if required , but you have to restart the windows to take effect of new username and password ; he default username and password is (Admin ,password)



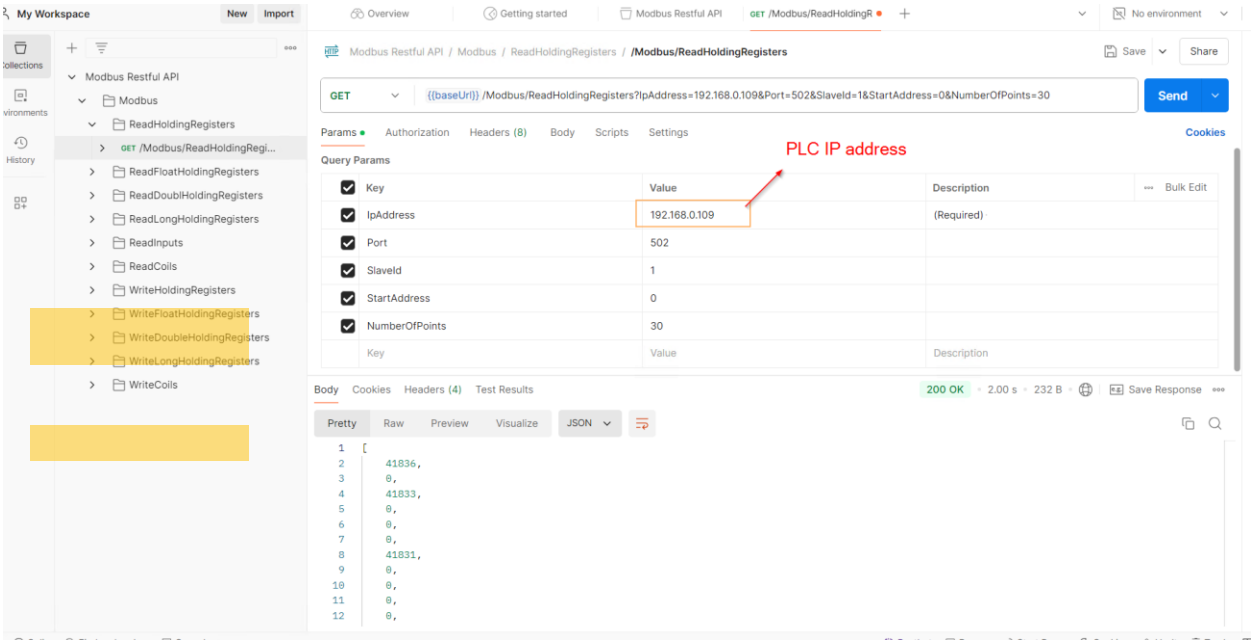
Step 2:Add new API and deploy it , if the IP address used before a warning message will be displayed



Step 3:After adding , it should be running and you can add many RESTful APIs to distribute the load

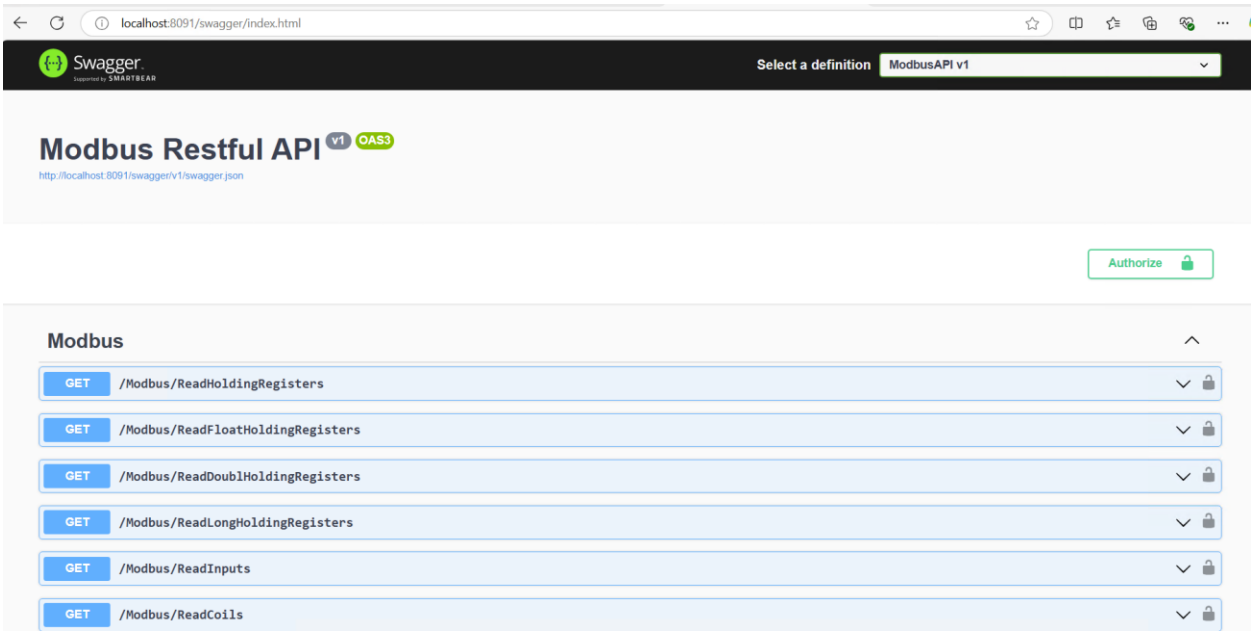


Step4 :You can verify the connection using postman using the shared collection in the support field

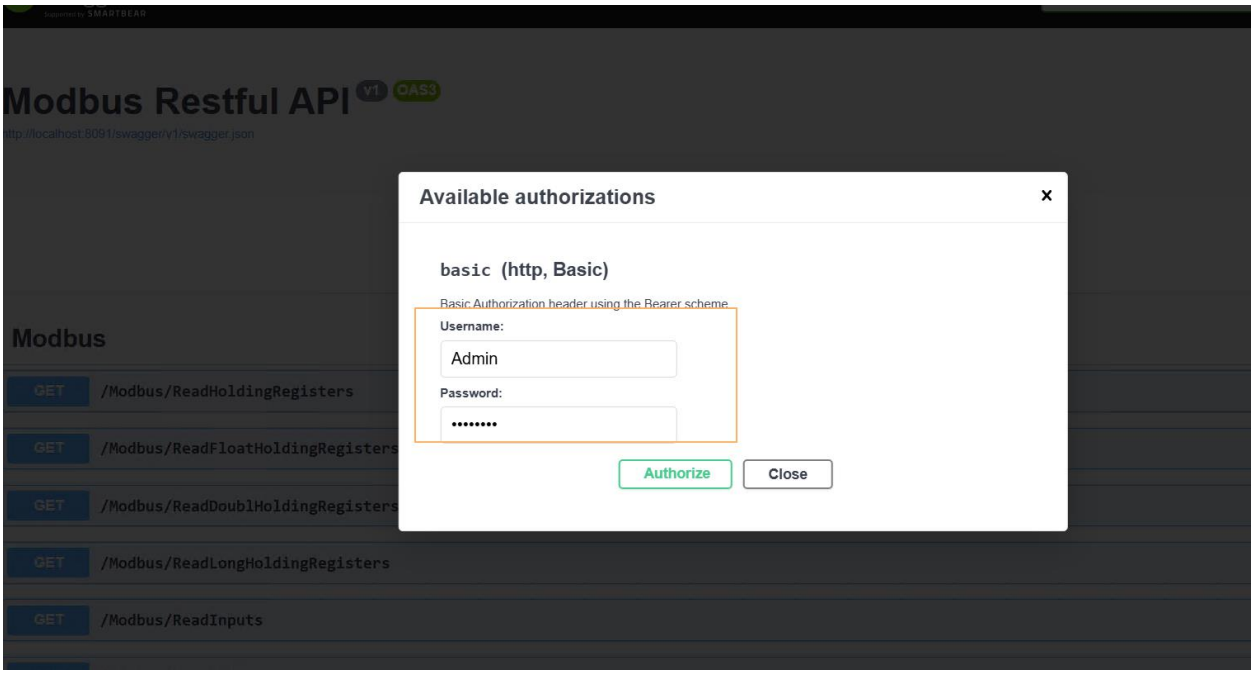


Or by sing the swagger from web browser as the following and to use the configured IP address

http://localhost:8091/swagger/index.html



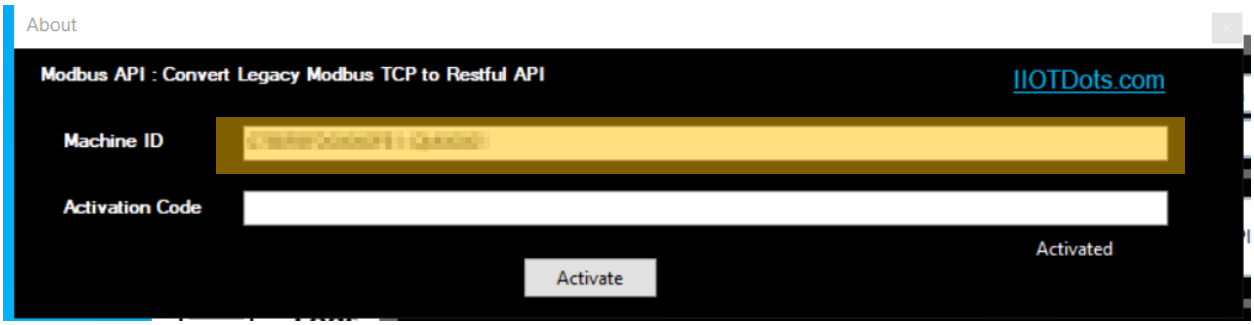
you have to authorize first before using it.


Modbus Restful API
v1
OAS3
http://localhost:8091/swagger/v1/swagger.json
Available authorizations
basic (http, Basic)
Basic Authorization header using the Bearer scheme
Username:
Admin
Password:
Authorize
Close
Modbus
GET /Modbus/ReadHoldingRegisters
GET /Modbus/ReadFloatHoldingRegisters
GET /Modbus/ReadDoublHoldingRegisters
GET /Modbus/ReadLongHoldingRegisters
GET /Modbus/ReadInputs

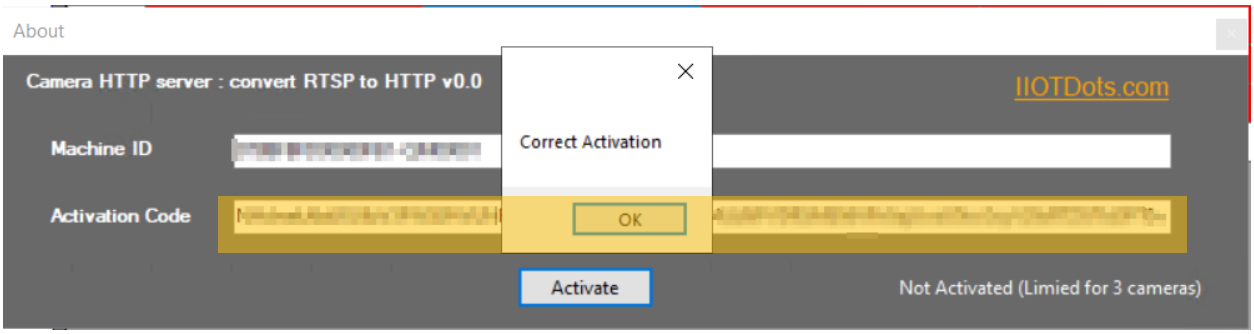
5. Program Activation

To able to have more than 10 request , you will need to activate the program, by sending the machine ID to:

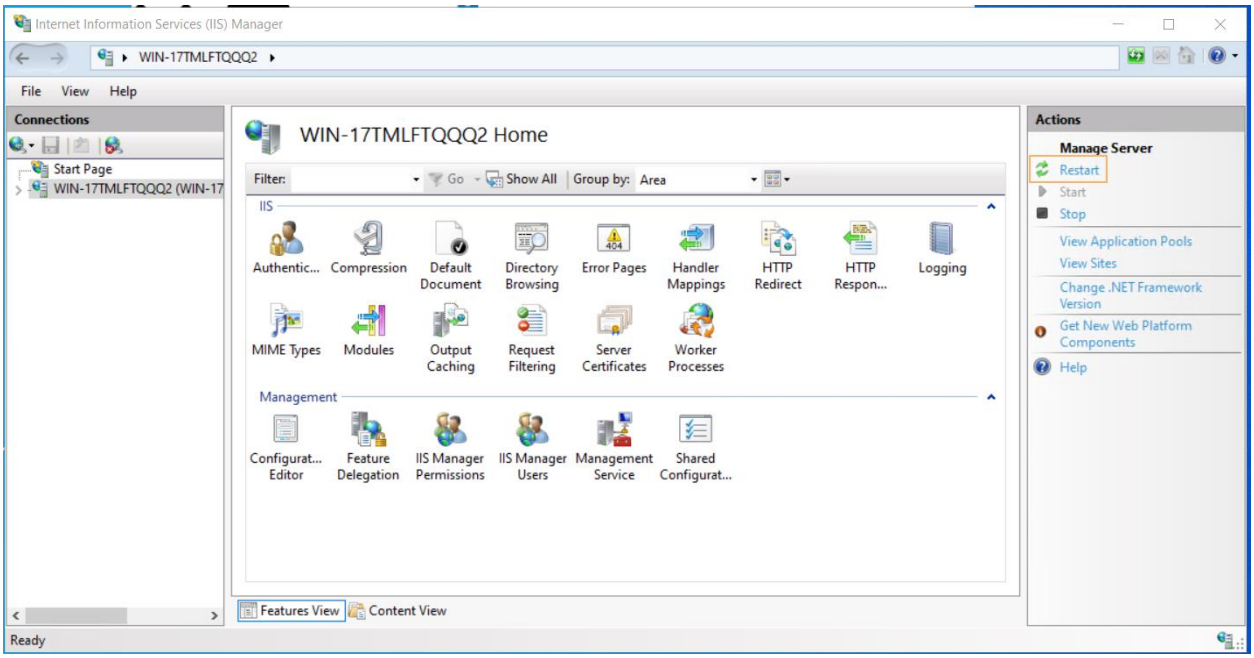
sales@IIOTdots.com



After getting the activation code , you can add it and activate the program



The IIS service need to be restarted after activation.


Internet Information Services (IIS) Manager
WIN-17TMLFTQQQ2
File
View
Help
Connections
Start Page
WIN-17TMLFTQQQ2 (WIN-17
WIN-17TMLFTQQQ2 Home
Filter:
Go
Show All
Group by:
Area
IIS
Authentic...
Compression
Default Document
Directory Browsing
Error Pages
Handler Mappings
HTTP Redirect
HTTP Respon...
Logging
MIME Types
Modules
Output Caching
Request Filtering
Server Certificates
Worker Processes
Management
Configurat... Editor
Feature Delegation
IIS Manager Permissions
IIS Manager Users
Management Service
Shared Configurat...
Features View
Content View
Ready
Actions
Manage Server
Restart
Start
Stop
View Application Pools
View Sites
Change .NET Framework Version
Get New Web Platform Components
Help

6. API Guide

The API allows interaction with Modbus holding registers, supporting different types of data such as standard integers, floating-point numbers, and double-precision floating-point numbers. Below is a description of each endpoint and how to use them:

6.1.1. ReadHoldingRegisters

This endpoint reads standard integer holding registers.

Endpoint:

- **Query-Based**:

```
GET{{baseUrl}}/Modbus/ReadHoldingRegisters?IpAddress=127.0.0.1&Port=502&Slave
Id=1&StartAddress=0&NumberOfPoints=10
```

- **RESTful Example**:

```
GET {{baseUrl}}/Modbus/ReadHoldingRegisters/127.0.0.1/502/1/0/10
```

Parameters:

- `IpAddress`: IP address of the Modbus server.
- `Port`: Modbus server port (default: `502`).
- `SlaveId`: ID of the slave device.
- `StartAddress`: Address to start reading from.
- `NumberOfPoints`: Number of registers to read.

Output Example :

```
{
  "status": "success",
  "data": {
    "IpAddress": "127.0.0.1",
    "Port": 502,
    "SlaveId": 1,
    "StartAddress": 0,
    "Registers": [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010]
  }
}
```

6.1.2. ReadFloatHoldingRegisters

This endpoint reads floating-point numbers from the holding registers.

Endpoint:

- **Query-Based**:

```
GET{{baseUrl}}/Modbus/ReadFloatHoldingRegisters?IpAddress=127.0.0.1&Port=502&
SlaveId=1&StartAddress=10&NumberOfPoints=4&FloatReversed=false
```

- **RESTful Example**:

```
GET{{baseUrl}}/Modbus/ReadFloatHoldingRegisters/127.0.0.1/502/1/10/4/false
```

Parameters:

- `IpAddress`: IP address of the Modbus server.
- `Port`: Modbus server port.
- `SlaveId`: ID of the slave device.
- `StartAddress`: Address to start reading from.
- `NumberOfPoints`: Number of registers to read.
- `FloatReversed`: Whether float data is reversed.

Output Example:

```
{
  "status": "success",
  "data": {
    "IpAddress": "127.0.0.1",
    "Port": 502,
    "SlaveId": 1,
    "StartAddress": 10,
    "FloatValues": [12.34, 56.78, 90.12, 34.56]
  }
}
```

6.1.3. ReadDoublHoldingRegisters

This endpoint reads double-precision floating-point numbers from the holding registers.

Endpoint:

- **Query-Based**:

```
GET{{baseUrl}}/Modbus/ReadDoublHoldingRegisters?IpAddress=127.0.0.1&Port=502&
SlaveId=1&StartAddress=20&NumberOfPoints=1&DoubletReversed=false
```

- **RESTful Example**:

```
GET {{baseUrl}}/Modbus/ReadDoublHoldingRegisters/127.0.0.1/502/1/20/1/false
```

Parameters:

- `IpAddress`: IP address of the Modbus server.
- `Port`: Modbus server port.
- `SlaveId`: ID of the slave device.
- `StartAddress`: Address to start reading from.
- `NumberOfPoints`: Number of registers to read.
- `DoubletReversed`: Whether double data is reversed.

Output Example:

```
{
  "status": "success",
  "data": {
    "IpAddress": "127.0.0.1",
    "Port": 502,
    "SlaveId": 1,
    "StartAddress": 20,
    "DoubleValues": [1234567.8901]
  }
}
```

6.1.4. ReadLongHoldingRegisters

This endpoint retrieves long integers (64-bit) from the Modbus holding registers.

Endpoint:

- **Query-Based**:

```
GET
{{baseUrl}}/Modbus/ReadLongHoldingRegisters?IpAddress=127.0.0.1&Port=502&Slav
eId=1&StartAddress=40&NumberOfPoints=4&LongReversed=false
```

- **RESTful Example**:

```
GET {{baseUrl}}/Modbus/ReadLongHoldingRegisters/127.0.0.1/502/1/40/4/false
```

Parameters:

- `IpAddress`: IP address of the Modbus server.
- `Port`: Modbus server port.
- `SlaveId`: ID of the slave device.
- `StartAddress`: Address to start reading from.
- `NumberOfPoints`: Number of registers to read.
- `LongReversed`: Whether the byte order of long values is reversed.

Output Example:

```
{
  "status": "success",
  "data": {
    "IpAddress": "127.0.0.1",
    "Port": 502,
    "SlaveId": 1,
    "StartAddress": 40,
    "LongValues": [123456789012345, 987654321098765, 456789012345678,
789012345678901]
  }
}
```

6.1.5. ReadInputs

This endpoint reads input states (Boolean values) from the Modbus input registers.

Endpoint:

- **Query-Based**:

```
GET
{{baseUrl}}/Modbus/ReadInputs?IpAddress=127.0.0.1&Port=502&SlaveId=1&StartAdd
ress=0&NumberOfPoints=100
```

- **RESTful Example**:

```
GET {{baseUrl}}/Modbus/ReadInputs/127.0.0.1/502/1/0/100
```

Parameters:

- `IpAddress`: IP address of the Modbus server.
- `Port`: Modbus server port.
- `SlaveId`: ID of the slave device.
- `StartAddress`: Address to start reading from.
- `NumberOfPoints`: Number of inputs to read.

Output Example:

```
{
  "status": "success",
  "data": {
    "IpAddress": "127.0.0.1",
    "Port": 502,
    "SlaveId": 1,
    "StartAddress": 0,
    "InputStates": [true, false, true, true, false, true, false, true, false, true,
false, true]
  }
}
```

6.1.6. WriteHoldingRegisters

This endpoint writes integer values to Modbus holding registers.

Endpoint:

- **Query-Based** (not applicable for POST).
- **RESTful Example**:

```
POST {{baseUrl}}/Modbus/WriteHoldingRegisters
```

Request Body:

```
{
  "ipAddress": "127.0.0.1",
  "port": 502,
  "slaveId": 1,
  "startAddress": 5,
  "values": [100, 200, 400]
}
```

Headers:

- `Content-Type: application/json`

Output Example:

```
{
  "status": "success",
  "message": "Values written successfully",
  "data": {
    "ipAddress": "127.0.0.1",
    "port": 502,
    "slaveId": 1,
    "startAddress": 5,
    "writtenValues": [100, 200, 400]
  }
}
```

6.1.7. WriteFloatHoldingRegisters

This endpoint writes floating-point values to Modbus holding registers.

Endpoint:

- **Query-Based** (not applicable for POST).
- **RESTful Example**:

```
POST {{baseUrl}}/Modbus/WriteFloatHoldingRegisters
```

Request Body:

```
{
  "ipAddress": "127.0.0.1",
  "port": 502,
  "slaveId": 1,
  "startAddress": 10,
  "values": [4.55, 3.55],
  "floatReversed": false
}
```

Headers:

- `Content-Type: application/json`

Output Example:

```
{
  "status": "success",
  "message": "Float values written successfully",
  "data": {
    "ipAddress": "127.0.0.1",
    "port": 502,
    "slaveId": 1,
    "startAddress": 10,
    "writtenValues": [4.55, 3.55],
    "floatReversed": false
  }
}
```

6.1.8. WriteDoubleHoldingRegisters

This endpoint writes double values to Modbus holding registers.

Endpoint:

- **RESTful Example**:

```
POST {{baseUrl}}/Modbus/WriteDoubleHoldingRegisters
```

Request Body:

```
{
  "ipAddress": "127.0.0.1",
  "port": 502,
  "slaveId": 1,
  "startAddress": 20,
  "values": [55555, 6666, 999],
  "doubleReversed": false
}
```

Headers:

- `Content-Type: application/json`

Output Example:

```
{
  "status": "success",
  "message": "Double values written successfully",
  "data": {
    "ipAddress": "127.0.0.1",
    "port": 502,
    "slaveId": 1,
    "startAddress": 20,
    "writtenValues": [55555, 6666, 999],
    "doubleReversed": false
  }
}
```

6.1.9. WriteLongHoldingRegisters

This endpoint writes long values to Modbus holding registers.

Endpoint:

- **RESTful Example**:

```
POST {{baseUrl}}/Modbus/WriteLongHoldingRegisters
```

Request Body:

```
{
  "ipAddress": "127.0.0.1",
  "port": 502,
  "slaveId": 1,
  "startAddress": 40,
  "values": [50, 60],
  "longReversed": true
}
```

Headers:

- `Content-Type: application/json`

Output Example:

```
{
  "status": "success",
  "message": "Long values written successfully",
  "data": {
    "ipAddress": "127.0.0.1",
    "port": 502,
    "slaveId": 1,
    "startAddress": 40,
    "writtenValues": [50, 60],
    "longReversed": true
  }
}
```

6.1.10. WriteCoils

This endpoint writes boolean values to Modbus coils.

Endpoint:

- **RESTful Example**:

```
POST {{baseUrl}}/Modbus/WriteCoils
```

Request Body:

```
{
  "ipAddress": "127.0.0.1",
  "port": 502,
  "slaveId": 1,
  "startAddress": 0,
  "values": [true, true, false, true]
}
```

Headers:

- `Content-Type: application/json`

Output Example:

```
{
  "status": "success",
  "message": "Coil values written successfully",
  "data": {
    "ipAddress": "127.0.0.1",
    "port": 502,
    "slaveId": 1,
    "startAddress": 0,
    "writtenValues": [true, true, false, true]
  }
}
```

6.1.11. General Notes:

1. Replace `{{baseUrl}}` with the base URL of the API service.
2. Provide valid credentials in the `Authorization` header.
3. Use appropriate query parameters to customize the request according to your Modbus configuration.