**Walkthrough: Creating a Request-Level HTTP Module By Using Native Code**

1. Article
2. 03/09/2018

**In this article**

1. [Prerequisites](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/walkthrough-creating-a-request-level-http-module-by-using-native-code#prerequisites)
2. [Creating the Module](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/walkthrough-creating-a-request-level-http-module-by-using-native-code#creating-the-module)
3. [Adding the Code and Source Files](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/walkthrough-creating-a-request-level-http-module-by-using-native-code#adding-the-code-and-source-files)
4. [Compiling and Testing the Module](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/walkthrough-creating-a-request-level-http-module-by-using-native-code#compiling-and-testing-the-module)

Show 2 more

This walkthrough demonstrates how to use C++ to create a sample request-level HTTP module that implements the new request-processing architecture in IIS 7. This new architecture extends the capabilities of native-code programming when you are writing IIS applications over earlier versions of ASP.NET HTTP modules and ISAPI filters or extensions. For more information about designing HTTP modules using the new request-processing architecture, see [Designing Native-Code HTTP Modules](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/designing-native-code-http-modules).

In this walkthrough, you will create a C++ project for your HTTP module, add the required code for a "Hello World" project, and then compile and test the module.

**Prerequisites**

The following software is required to complete the steps in the example:

* IIS 7.
* Visual Studio 2005.
* Windows Software Development Kit (SDK).

**Note** You can use Visual Studio .NET 2003 or earlier, although the walkthrough steps may not be identical.

**Creating the Module**

In this part of the walkthrough, you will create an empty C++ DLL project for your HTTP module.

**To create a new C++ DLL project**

1. Open Visual Studio 2005.
2. Verify that the global options have all the right paths to the SDK include files:
   1. On the **Tools** menu, click **Options**.
   2. Expand the **Projects and Solutions** node in the tree view, and then click **VC++ Directories**.
   3. In the **Show directories for** drop-down box, select **Include files**.
   4. Verify that the path where you installed the Windows SDK include files is listed. If the path is not listed, click the **New Line** icon, and then add the path where you installed the SDK include files. The default installation directory is $(VCInstallDir)PlatformSDK\bin.
   5. Click **OK**.
3. Create a new C++ project:
   1. On the **File** menu, point to **New**, and then click **Project**.

The **New Project** dialog box opens.

* 1. In the **Project Types** pane, expand the **Visual C++** node, and then click **Win32**.
  2. In the **Templates** pane, select **Win32 Project**.
  3. In the **Name** box, type **HelloWorld**.
  4. In the **Location** box, type the path for the sample.
  5. Click **OK**.

The **Win32 Application Wizard** opens.

* 1. Click **Application Settings**.
  2. Under **Application type**, click **DLL**.
  3. Under **Additional options**, click **Empty project**.
  4. Click **Finish**.

**Adding the Code and Source Files**

The next step is to add the required C++ and module-definition files to the project.

**To add the source files to the project**

1. Create the module-definition file to export the [RegisterModule](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/pfn-registermodule-function) function:
   1. In Solution Explorer, right-click **Source Files**, point to **Add**, and then click **New Item**.

The **Add New Item** dialog box opens.

* 1. Expand the **Visual C++** node in the **Categories** pane, and then click **Code**.
  2. In the **Templates** pane, select the **Module-Definition File** template.
  3. In the **Name** box, type **HelloWorld**, and leave the default path for the file in the **Location** box.
  4. Click **Add**.
  5. Add a line with EXPORTS and RegisterModule. Your file should look like the code below:

Copy

LIBRARY"HelloWorld"

EXPORTS

RegisterModule

**Note**

Instead of creating a module-definition file, you can export the **[RegisterModule](https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/pfn-registermodule-function)** function by using the **/EXPORT:RegisterModule** switch.

1. Create the C++ file:
   1. In Solution Explorer, right-click **Source Files**, point to **Add**, and then click **New Item**.

The **Add New Item** dialog box opens.

* 1. Expand the **Visual C++** node in the **Categories** pane, and then click **Code**.
  2. In the **Templates** pane, select the **C++ File** template.
  3. In the **Name** box, type **HelloWorld**, and leave the default path for the file in the **Location** box.
  4. Click **Add**.
  5. Add the following code:

c++Copy

#define \_WINSOCKAPI\_

#include <windows.h>

#include <sal.h>

#include <httpserv.h>

// Create the module class.

class CHelloWorld : public CHttpModule

{

public:

REQUEST\_NOTIFICATION\_STATUS

OnBeginRequest(

IN IHttpContext \* pHttpContext,

IN IHttpEventProvider \* pProvider

)

{

UNREFERENCED\_PARAMETER( pProvider );

// Create an HRESULT to receive return values from methods.

HRESULT hr;

// Retrieve a pointer to the response.

IHttpResponse \* pHttpResponse = pHttpContext->GetResponse();

// Test for an error.

if (pHttpResponse != NULL)

{

// Clear the existing response.

pHttpResponse->Clear();

// Set the MIME type to plain text.

pHttpResponse->SetHeader(

HttpHeaderContentType,"text/plain",

(USHORT)strlen("text/plain"),TRUE);

// Create a string with the response.

PCSTR pszBuffer = "Hello World!";

// Create a data chunk.

HTTP\_DATA\_CHUNK dataChunk;

// Set the chunk to a chunk in memory.

dataChunk.DataChunkType = HttpDataChunkFromMemory;

// Buffer for bytes written of data chunk.

DWORD cbSent;

// Set the chunk to the buffer.

dataChunk.FromMemory.pBuffer =

(PVOID) pszBuffer;

// Set the chunk size to the buffer size.

dataChunk.FromMemory.BufferLength =

(USHORT) strlen(pszBuffer);

// Insert the data chunk into the response.

hr = pHttpResponse->WriteEntityChunks(

&dataChunk,1,FALSE,TRUE,&cbSent);

// Test for an error.

if (FAILED(hr))

{

// Set the HTTP status.

pHttpResponse->SetStatus(500,"Server Error",0,hr);

}

// End additional processing.

return RQ\_NOTIFICATION\_FINISH\_REQUEST;

}

// Return processing to the pipeline.

return RQ\_NOTIFICATION\_CONTINUE;

}

};

// Create the module's class factory.

class CHelloWorldFactory : public IHttpModuleFactory

{

public:

HRESULT

GetHttpModule(

OUT CHttpModule \*\* ppModule,

IN IModuleAllocator \* pAllocator

)

{

UNREFERENCED\_PARAMETER( pAllocator );

// Create a new instance.

CHelloWorld \* pModule = new CHelloWorld;

// Test for an error.

if (!pModule)

{

// Return an error if the factory cannot create the instance.

return HRESULT\_FROM\_WIN32( ERROR\_NOT\_ENOUGH\_MEMORY );

}

else

{

// Return a pointer to the module.

\*ppModule = pModule;

pModule = NULL;

// Return a success status.

return S\_OK;

}

}

void

Terminate()

{

// Remove the class from memory.

delete this;

}

};

// Create the module's exported registration function.

HRESULT

\_\_stdcall

RegisterModule(

DWORD dwServerVersion,

IHttpModuleRegistrationInfo \* pModuleInfo,

IHttpServer \* pGlobalInfo

)

{

UNREFERENCED\_PARAMETER( dwServerVersion );

UNREFERENCED\_PARAMETER( pGlobalInfo );

// Set the request notifications and exit.

return pModuleInfo->SetRequestNotifications(

new CHelloWorldFactory,

RQ\_BEGIN\_REQUEST,

0

);

}

**Compiling and Testing the Module**

**To compile and test the project**

1. Compile the HTTP module:
   1. On the **Build** menu, click **Build Solution**.
   2. Verify that Visual Studio did not return any errors or warnings.
   3. Add the HelloWorld.dll module (with the complete path) to the globalModules section of %windir%\system32\inetsrv\config\applicationHost.config file.
2. Use Internet Explorer to browse to your Web site; you should see "Begin Request sample " with the request count displayed.

**Note**

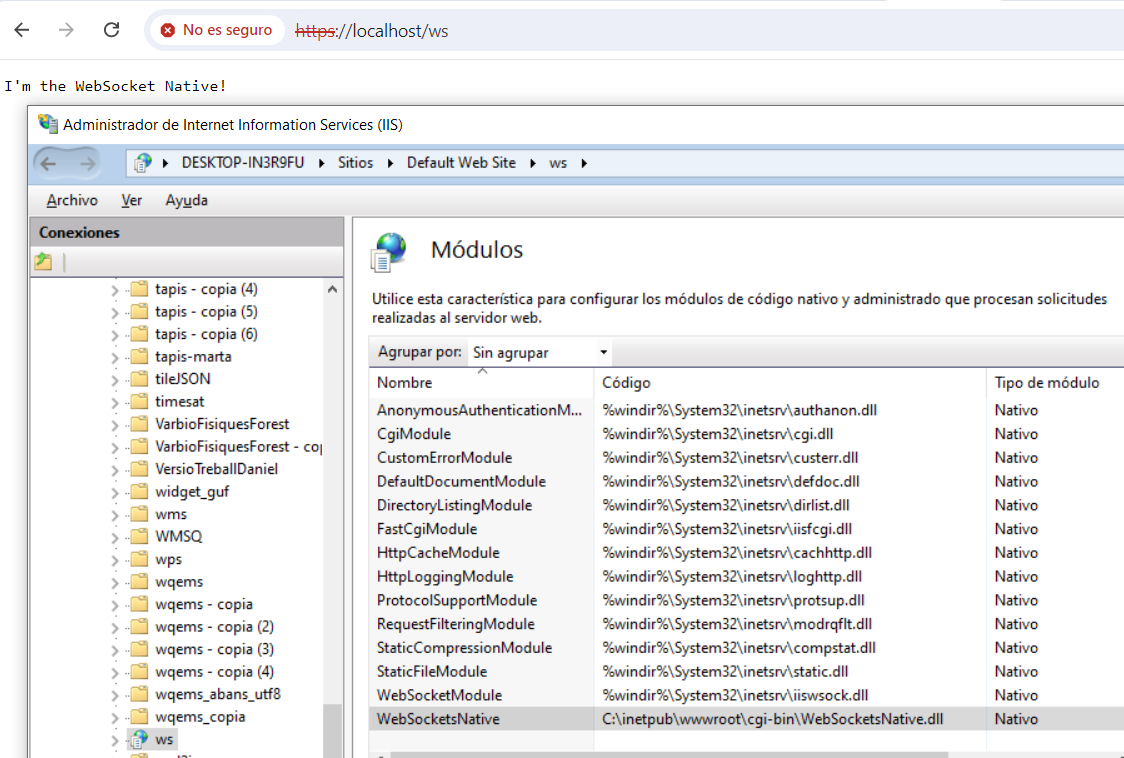
You will need to stop IIS before you link your project on subsequent builds.

**Troubleshooting Your Settings**

If your module does not compile or does not work as expected, here are several areas that you can check:

* Ensure that you have specified \_\_stdcall for your exported functions, or that you have configured compilation by using the \_\_stdcall (/Gz) calling convention.
* Ensure that IIS has loaded HelloWorld.dll:
  1. In IIS Manager, click **Default Web Site** in the **Connections** pane.
  2. In the workspace (the center pane), select **Features View**.
  3. In the **Group by** box, select **Category**.
  4. In the **Server Components** category, double-click **Modules**.
  5. Verify that HelloWorld module is listed.
* Ensure that you have added the correct RegisterModule export to your definition file.
* Ensure that you have added the definition file to the project settings. To add the file to the project settings, complete the following steps:
  1. On the **Project** menu, click **Properties**.
  2. Expand the **Configuration Properties** node in the tree view, expand the **Linker** node, and then click **Input**.
  3. For the **Module Definition File** settings, ensure that your definition file is listed.

És més fàcil del que semblava:



Les instruccions d'aquí són molt precises i funcionen.

<https://learn.microsoft.com/en-us/previous-versions/iis/smooth-streaming-client/walkthrough-creating-a-request-level-http-module-by-using-native-code>

El més peculiar és que s'ha d'instal·lar com un mòdul que es carrega sempre invoquis la plana que invoquis. Per això l'he hagut de posar en un directori que és una aplicació i que no té res a dins.

El pas més important és:

1. Add the HelloWorld.dll module (with the complete path) to the globalModules section of %windir%\system32\inetsrv\config\applicationHost.config file.

Si t'equivoques de nom d'aplicació retorna un 503 i cau tot el sistema d'execució d'aplicacions i queda documentat com un supererror al visor de successos:

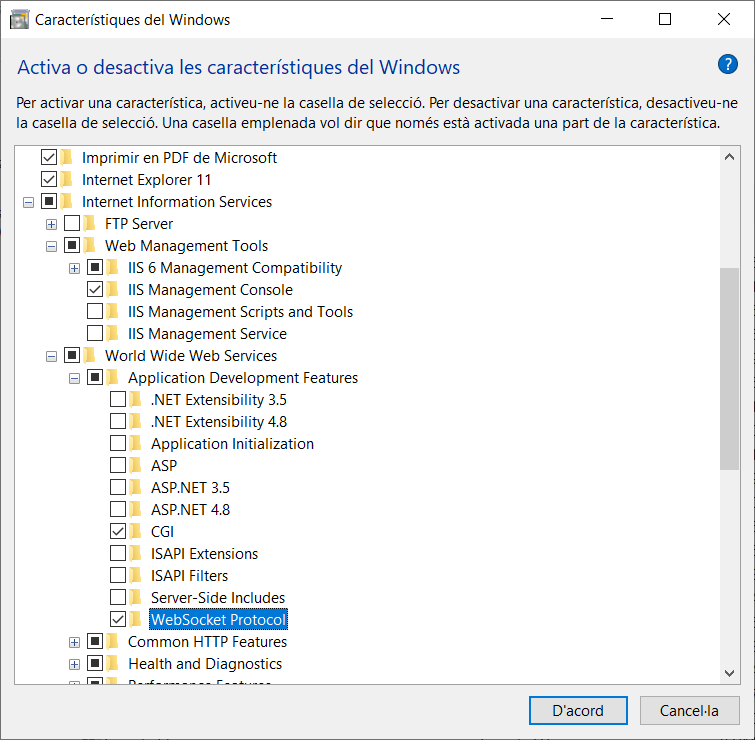


Ara vaig a provar de retornar un canvi de protocol a veure que passa!.

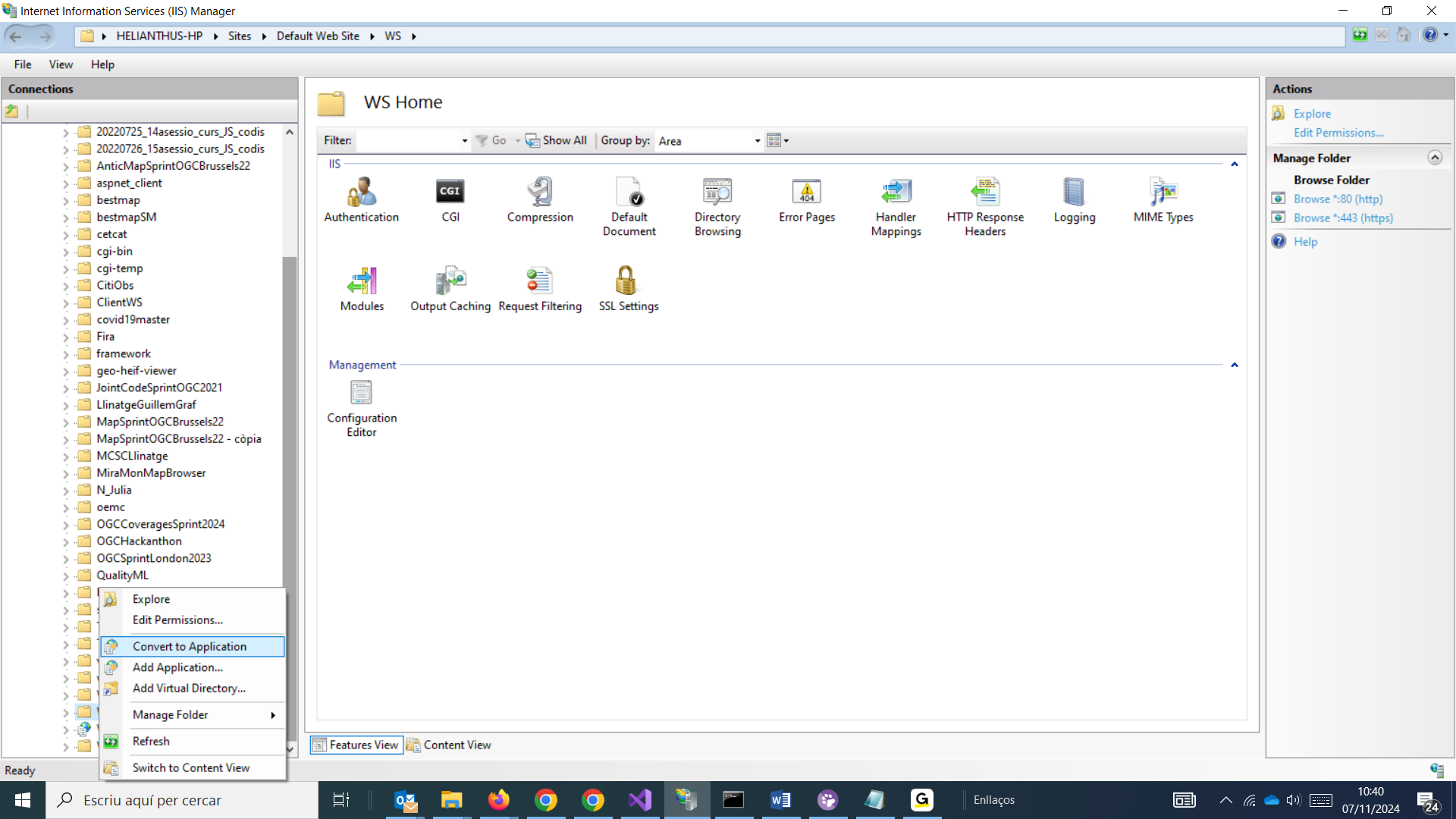
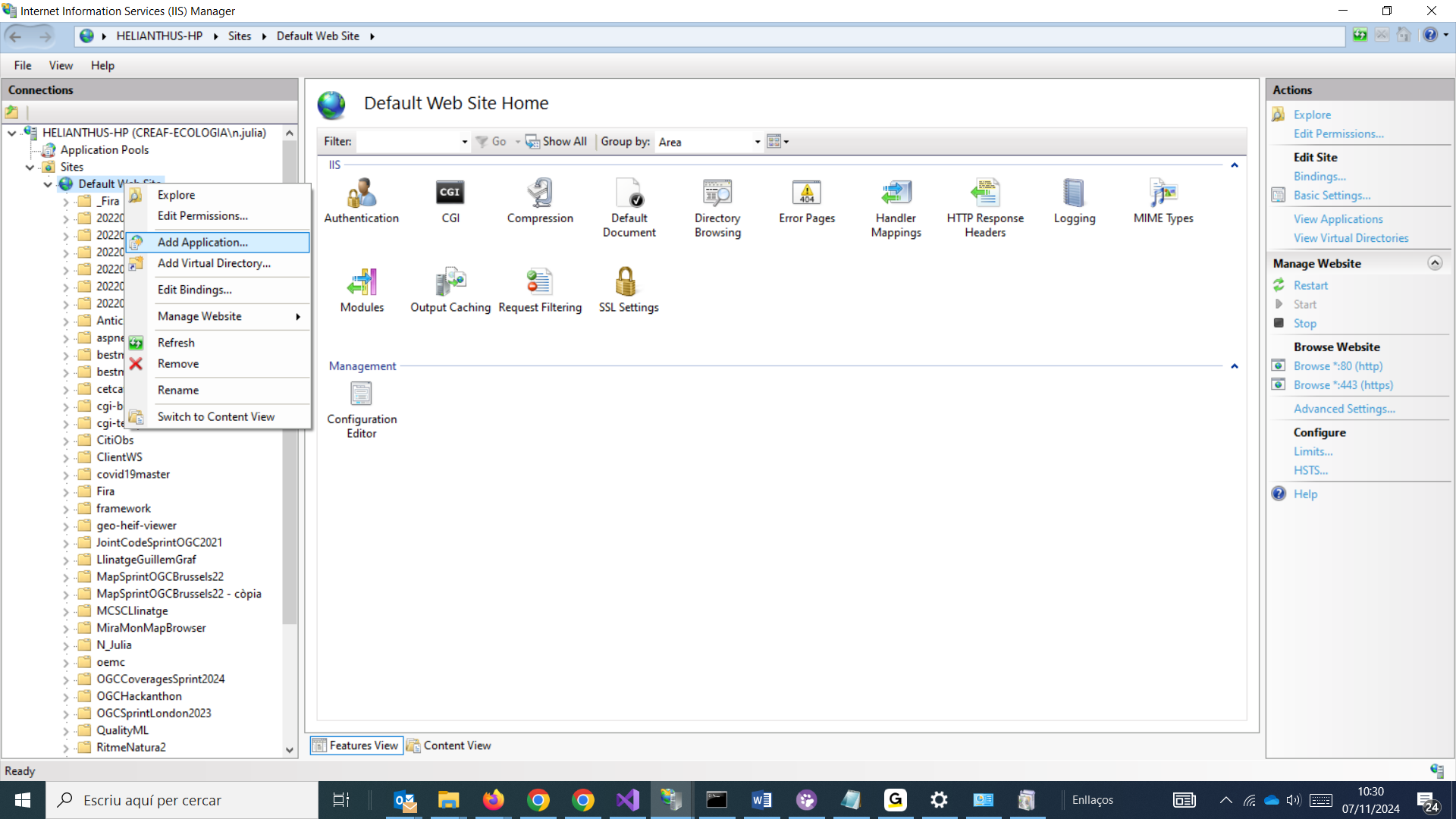
<https://www.isdecisions.com/products/userlock/help/agents/iis_agent/iis_agent_configuration.htm>

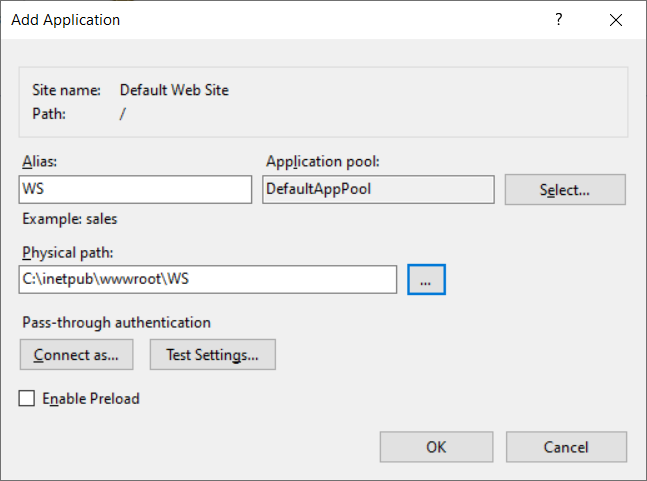
**COM CONFIGURAR EL ISS PERQUE FUNCIONI UN CODI NATIU COM A WEBSOCKET**

1. Tenir un Windows 10 o superior o un Windows Server 2019 amb IIS
2. Afegir el component WebSocket al IIS:



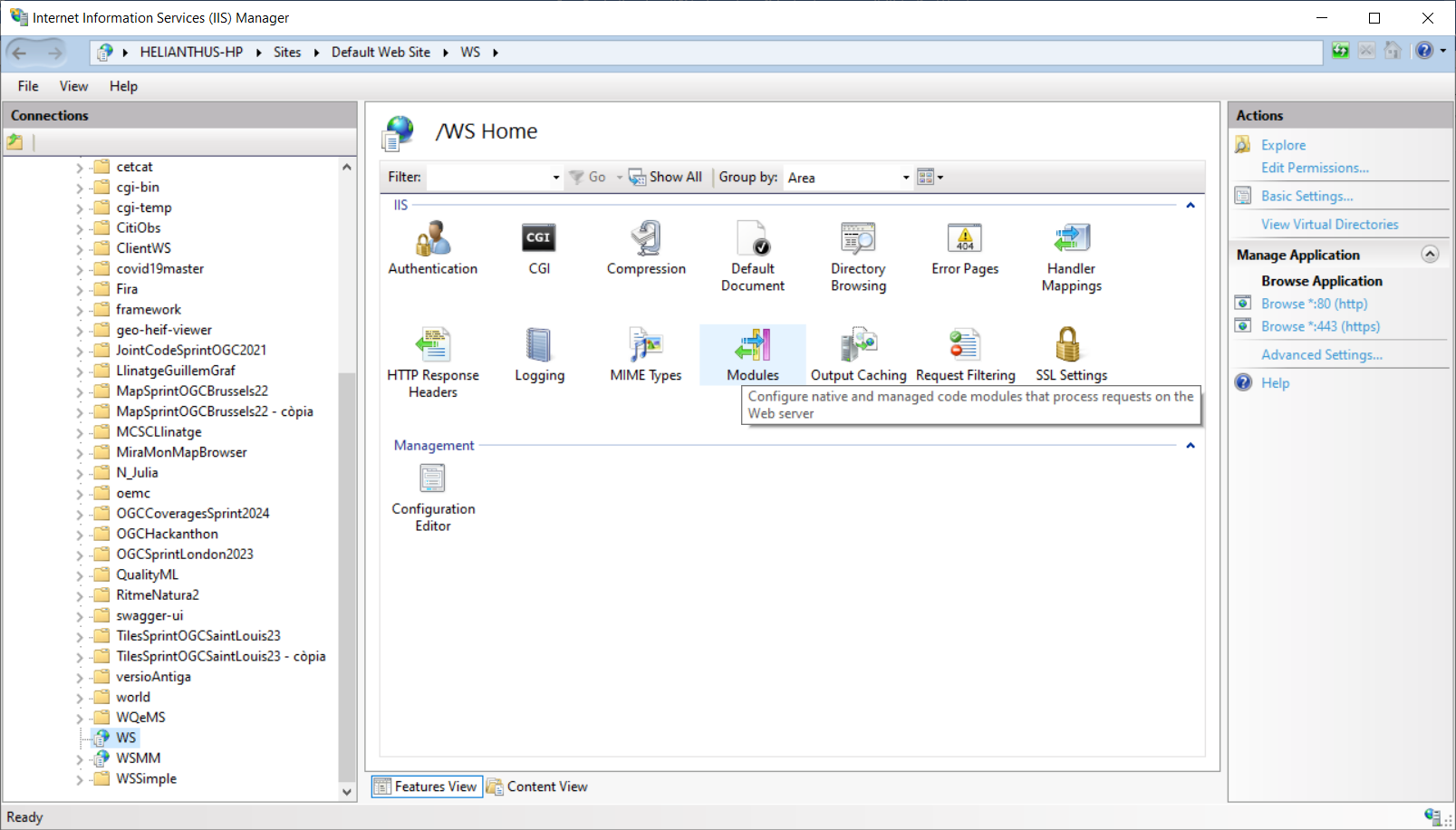
1. Cal afegir el teu modul natiu per example “WebSocketsNative.dll” (amb el path complet a dins de l’arbre de navegació, per exemple: c:\Inetpub\wwwroot\WSMM\WebSocketsNative.dll o c:\Inetpub\WSMM\WebSocketsNative.dll (en aquest cas fora de l’arbre de navegació) a la”globalModules” del fitxer %windir%\system32\inetsrv\config\applicationHost.config. (obrir amb el notepad o similar mode administrador des del CMD).
2. Ara cal crear una APLICACIÓ és a dir, convertir el directori que voleu que actui com a websocket en aplicació(Appliction Pools). Hi ha 2 maneres de fer-ho:
   1. Crear un directori buit amb el nom que voleu que tingui el vostre websocket, per exemple “c:\InetPub\wwwroot\WS”, i convertir-lo en aplicació directament fent botó dret ratolí sobre l’element WS (o bé afegir una aplicació fent botó dret des de l’arrel de navegació).



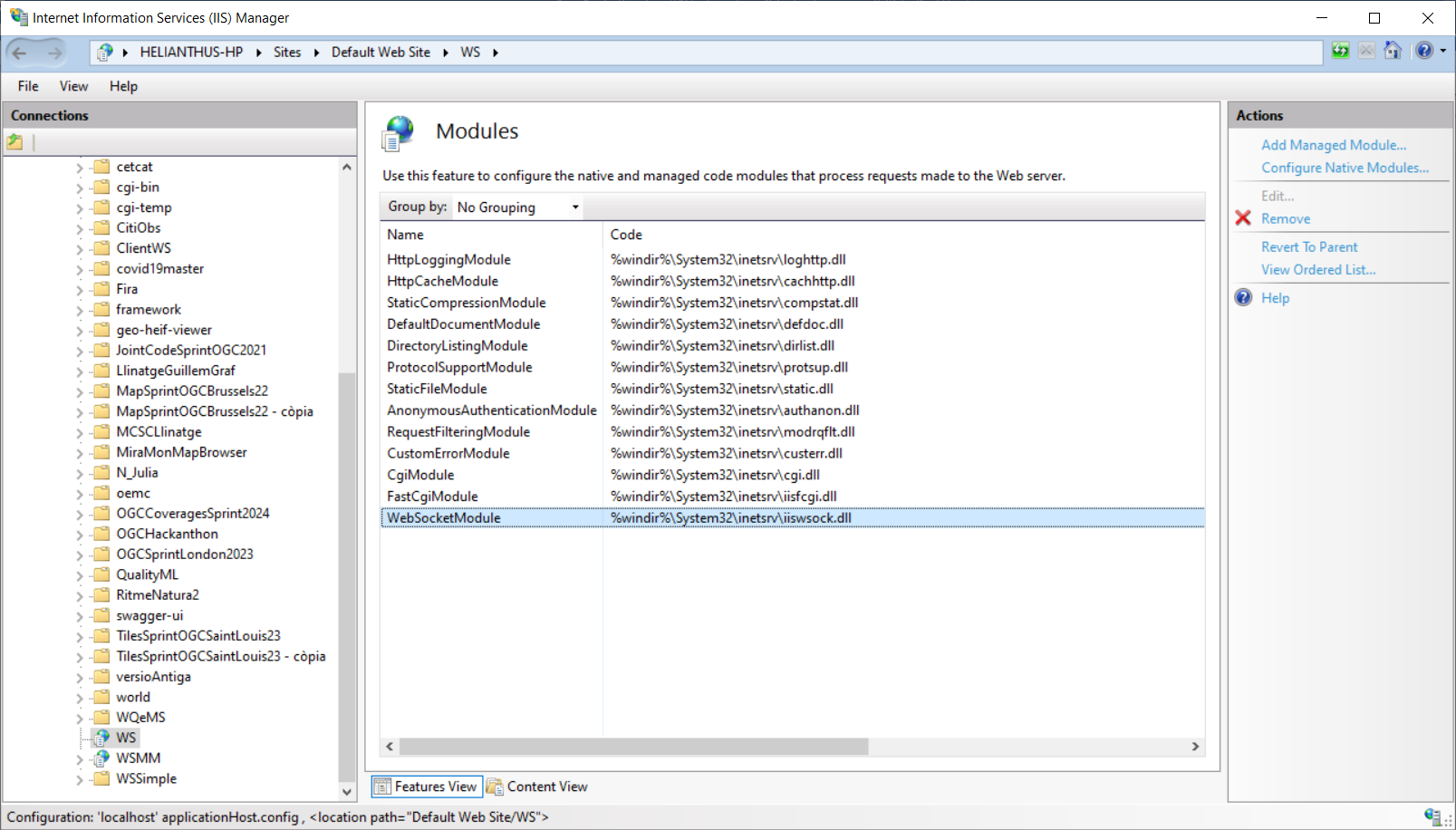


Un cop tenim això ara hem de dir-li que es comporti com un Websocket. Això es fa activan dos mòduls:

Al IIS, aneu a la carpeta “WS” de l’arbre que es mostra a l’esquerra i en les opcions d’aquesta carpeta que es mostren a la dreta seleccioneu “Modules”:

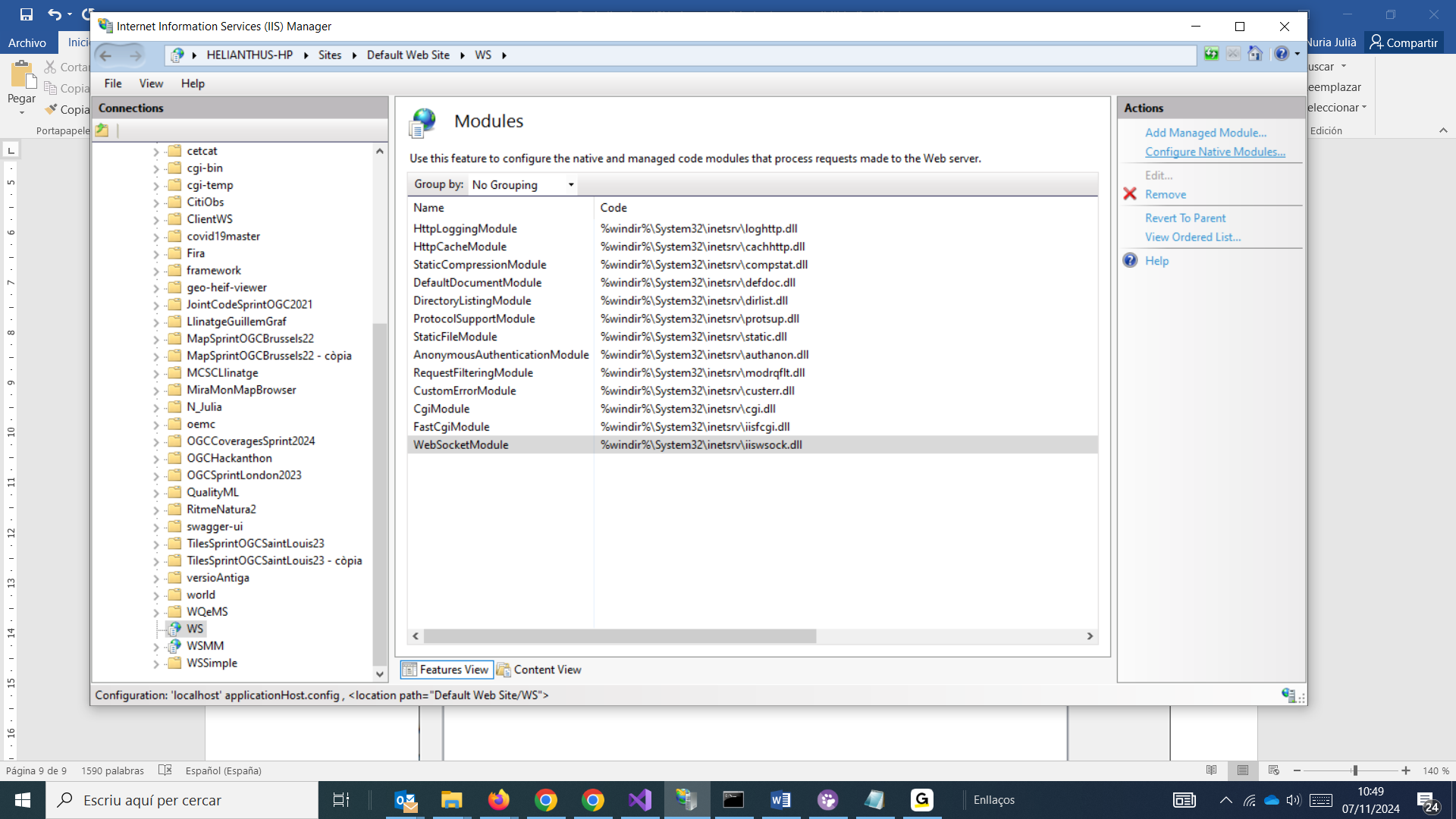
I

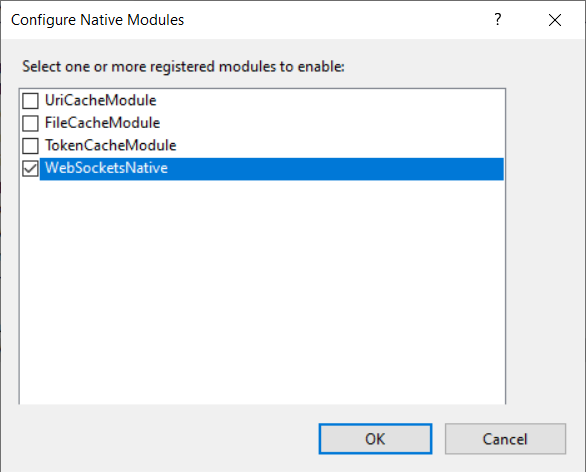
Quan l’obriu ha de tenir un aspecto com ara aquest, on apareix el modul general per fer WebSockets (gràcies a la instal·lació que hem fet del component prèviament)



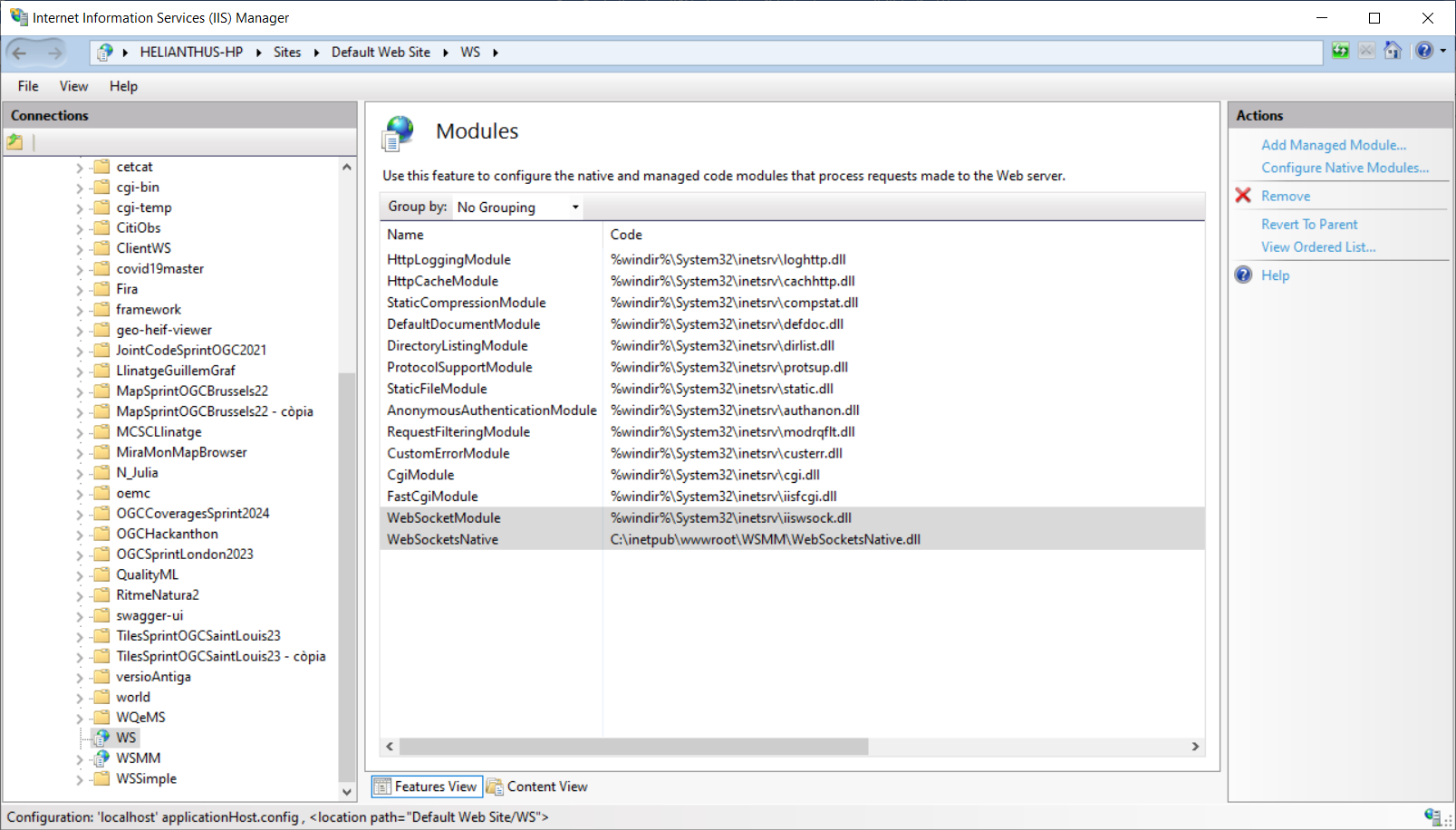
Ara hem d’afegir el nostre mòdul natiu:

Cliclant a “Configure Native Modules.” (a la dreta) apareixerà una finestra am bels mòduls natius que tenim configurats (com que hem editat el fitxer de configuració abans, ha d’apareixer el nostre modul natiu





El seleccioneu i en principi ja està configurat tot.



Ara si intenteu fer una connexió ws o wss hauria de funcionar a ws://nom\_domini/ws o bé wss://nom\_domini/ws si teniu el servidor amb un certificat https instal·lat.

* 1. Aquesta opció és bàsicament igual però usem el mateix directori on hem possat la dll i fem els mateixos pasos.

NOTEU: Que si useu l’opció a no cal que la dll estigui en el directori de navegació però si useu l’opció B si.

NOTA IMPORTANT: Sembla ser que és important que primer afegiu el WebSocketModule i després el ModulNatiu o no funcionarà res.

Al fitxer de configuración les coses s’han de veure en aquest ordre:

<globalModules>

<add name="WebSocketModule" image="%windir%\System32\inetsrv\iiswsock.dll" />

<add name="WebSocketsNative" image="C:\inetpub\wwwroot\WSMM\WebSocketsNative.dll" />

</globalModules>