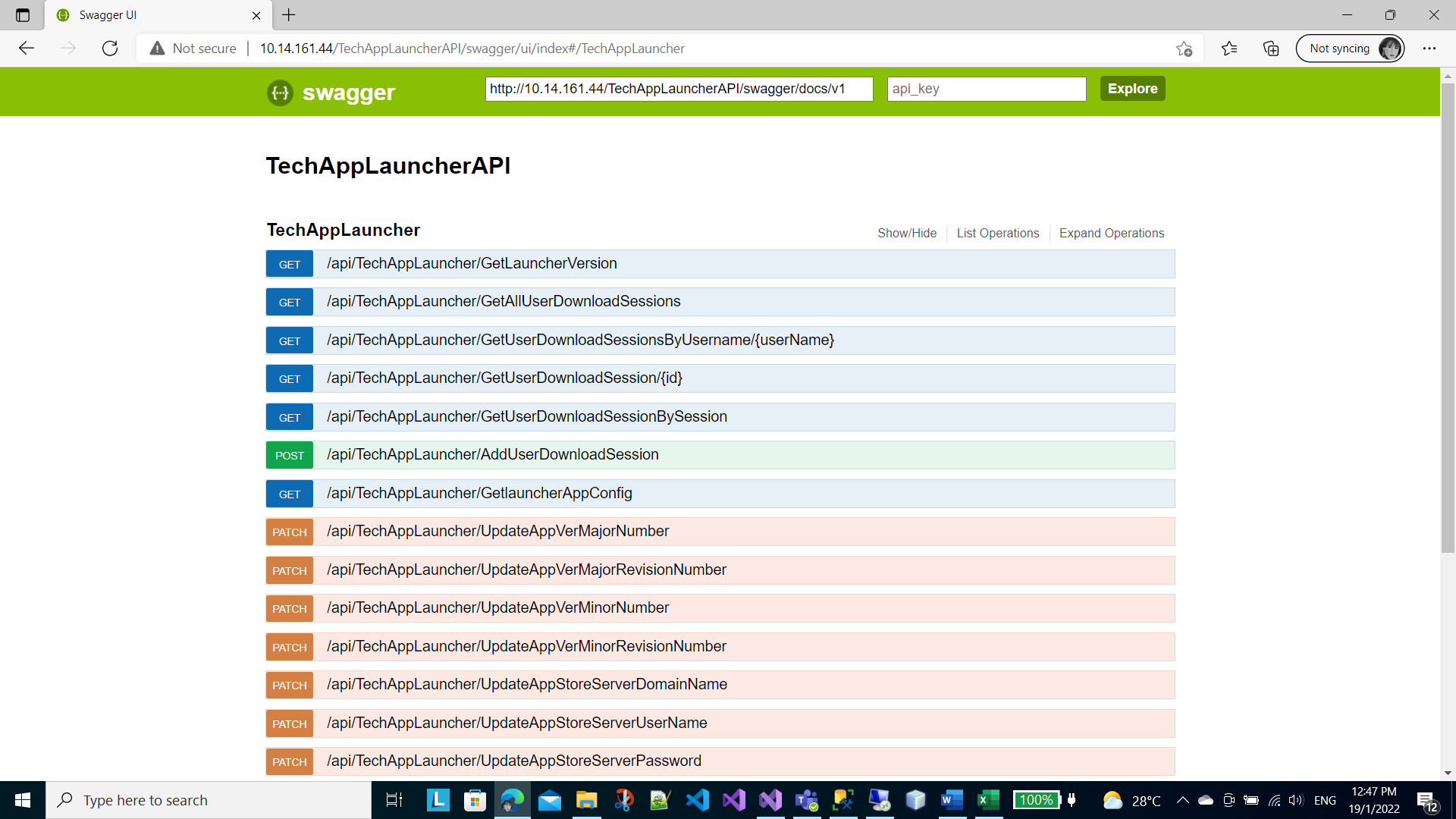
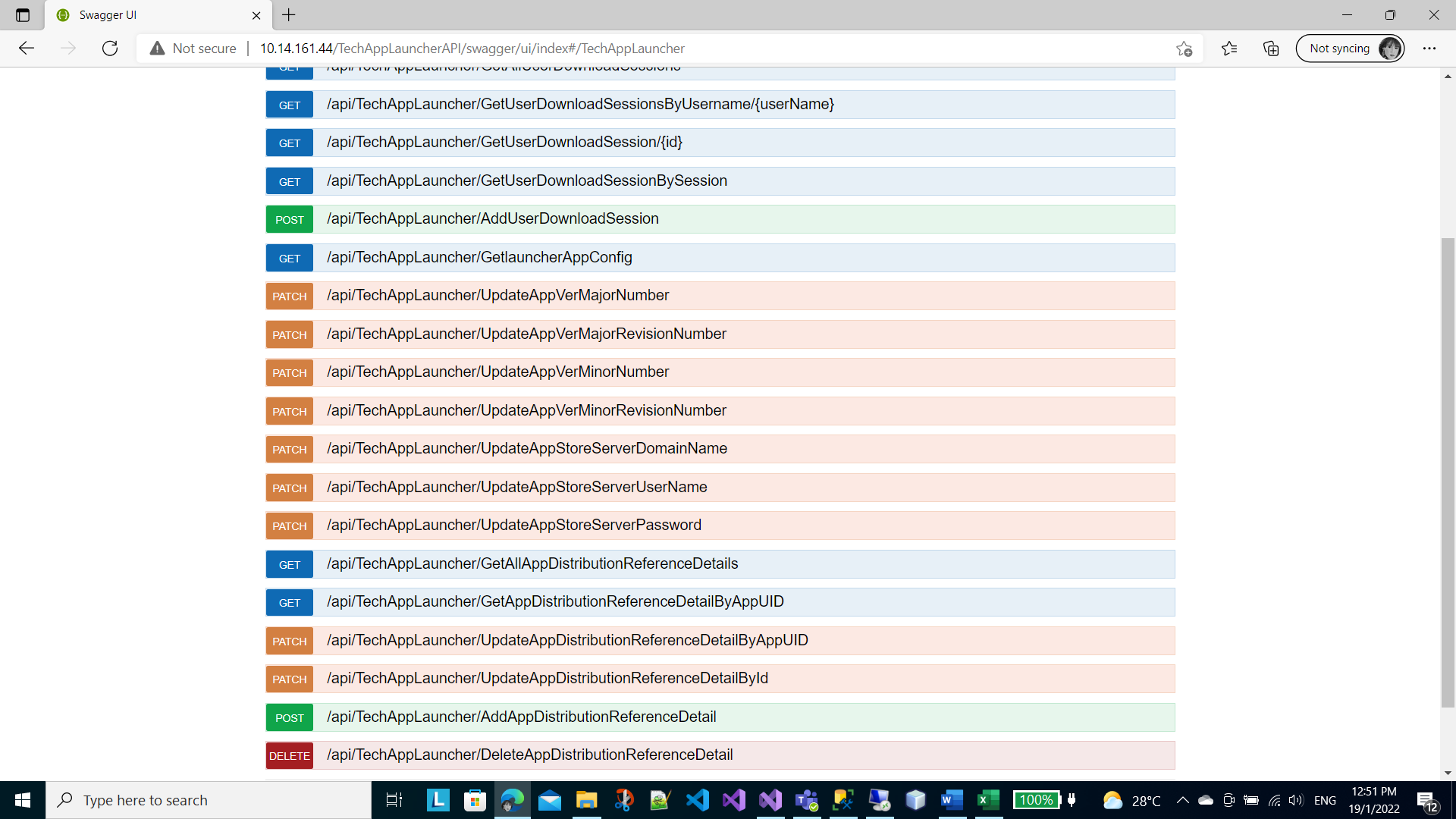
**TechAppLauncherAPI**

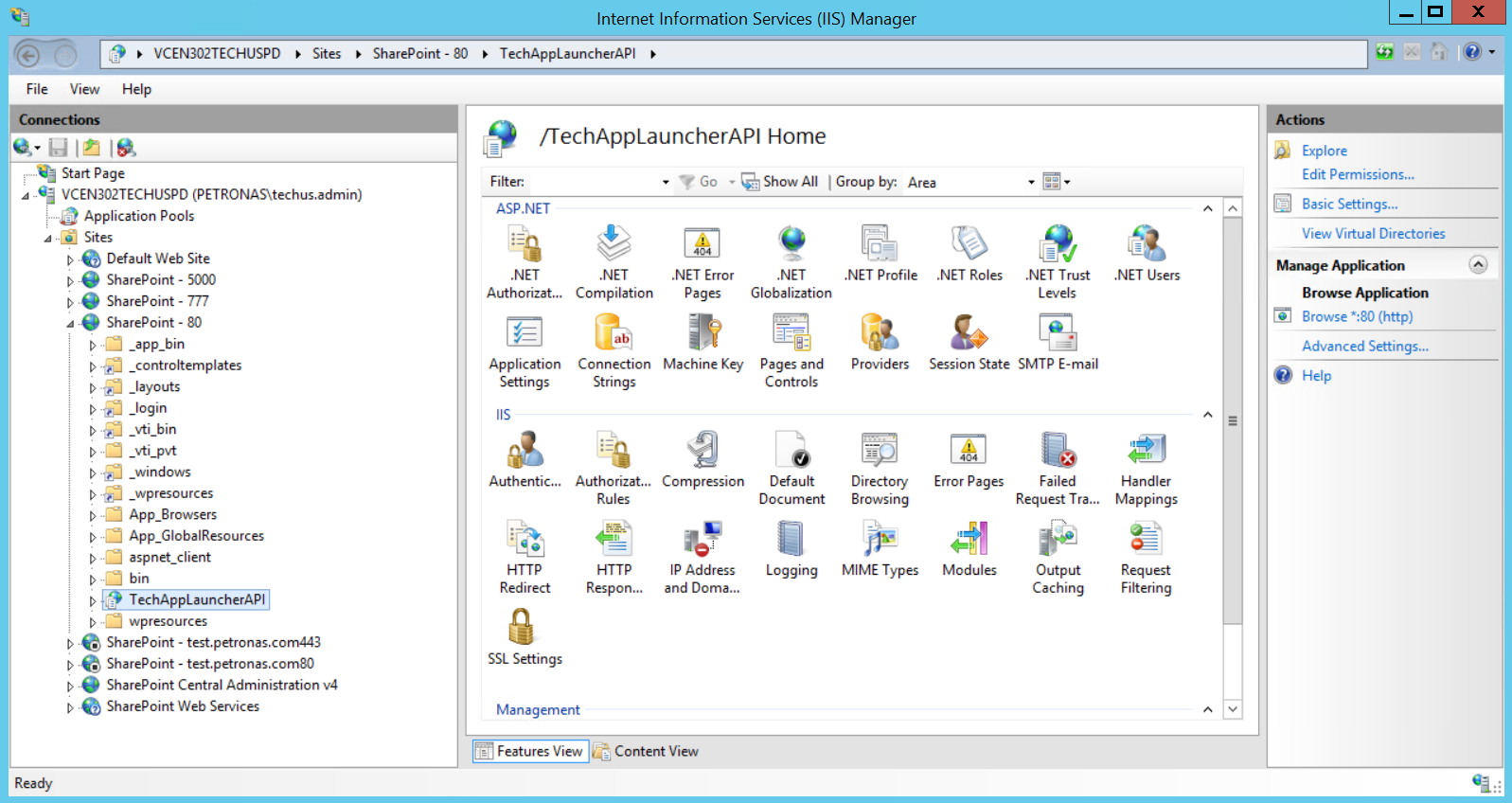
|  |  |
| --- | --- |
| Application Name | TechAppLauncherAPI |
| Application Framework | .Net Framework 4.7.2 |
| UI Framework | ASP.net WebAPI |
| IDE | Visual Studio 2019 |
| Repository | <https://github.com/azzulhisham/Petronas_TechAppLauncherAPI.git> |
| Application Url | <http://10.14.161.44/TechAppLauncherAPI/swagger/ui/index> |
| Host | IIS |

**Overview**

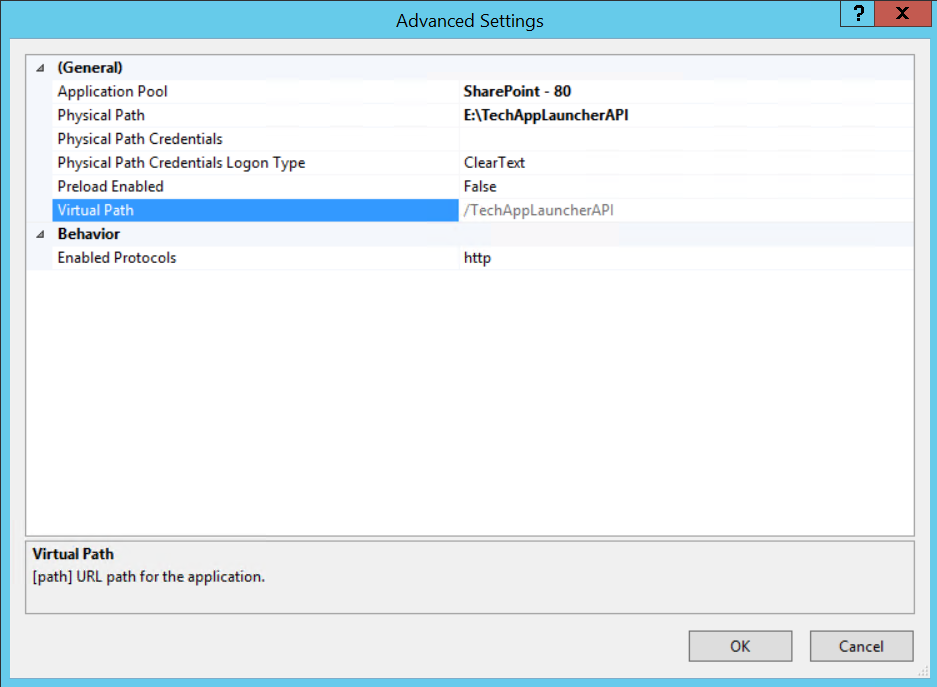


The link [Swagger UI](http://10.14.161.44/TechAppLauncherAPI/swagger/ui/index#/TechAppLauncher) provided above is the application API content which can be viewed through Swagger UI.

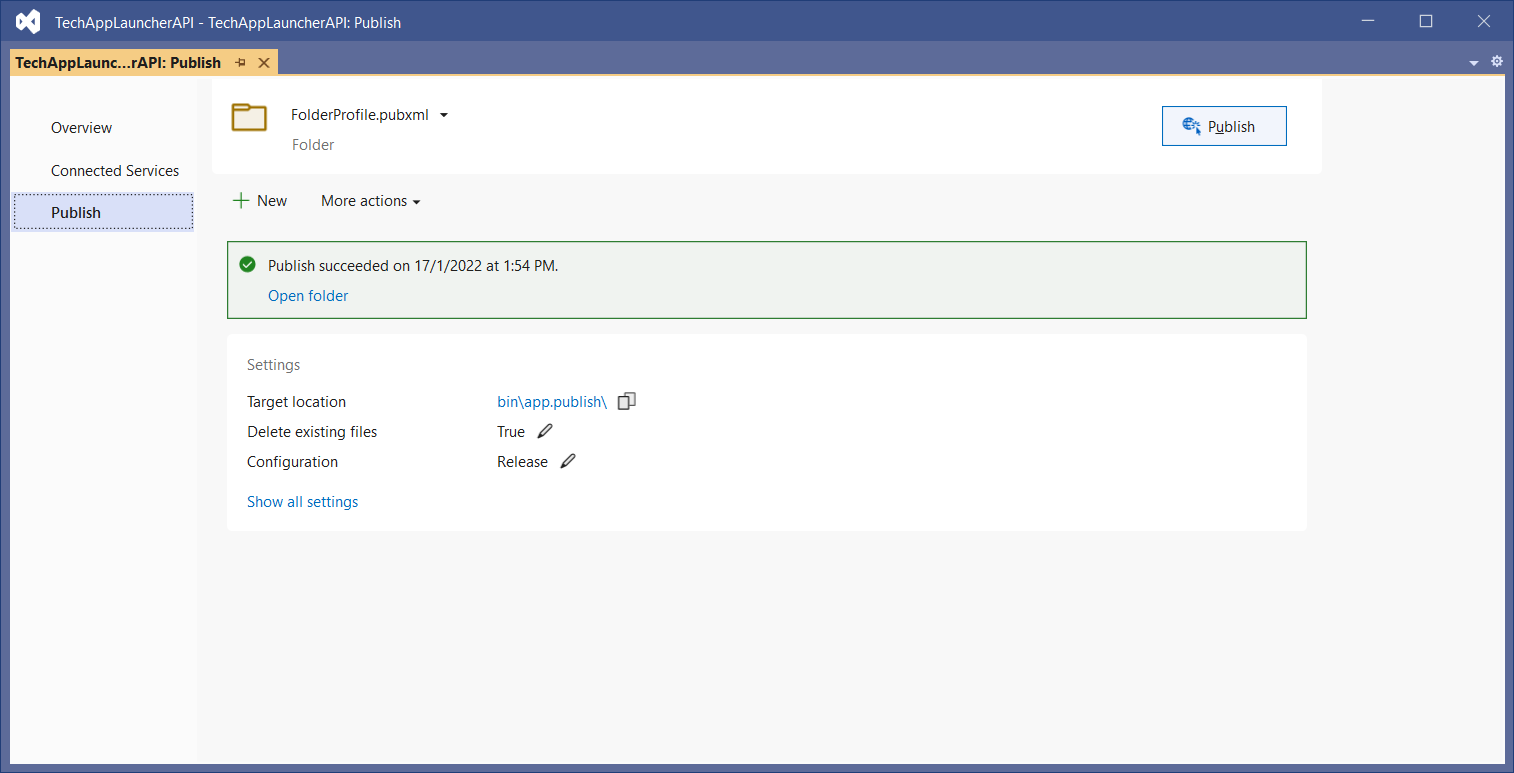
It is hosted by IIS Server at **10.14.161.44**. On top of that it is connecting to the MS-SQL server at **10.14.162.74**.



The physical path of the application is located at ‘**E:\TechAppLauncherAPI**’



**Deploying New Version**



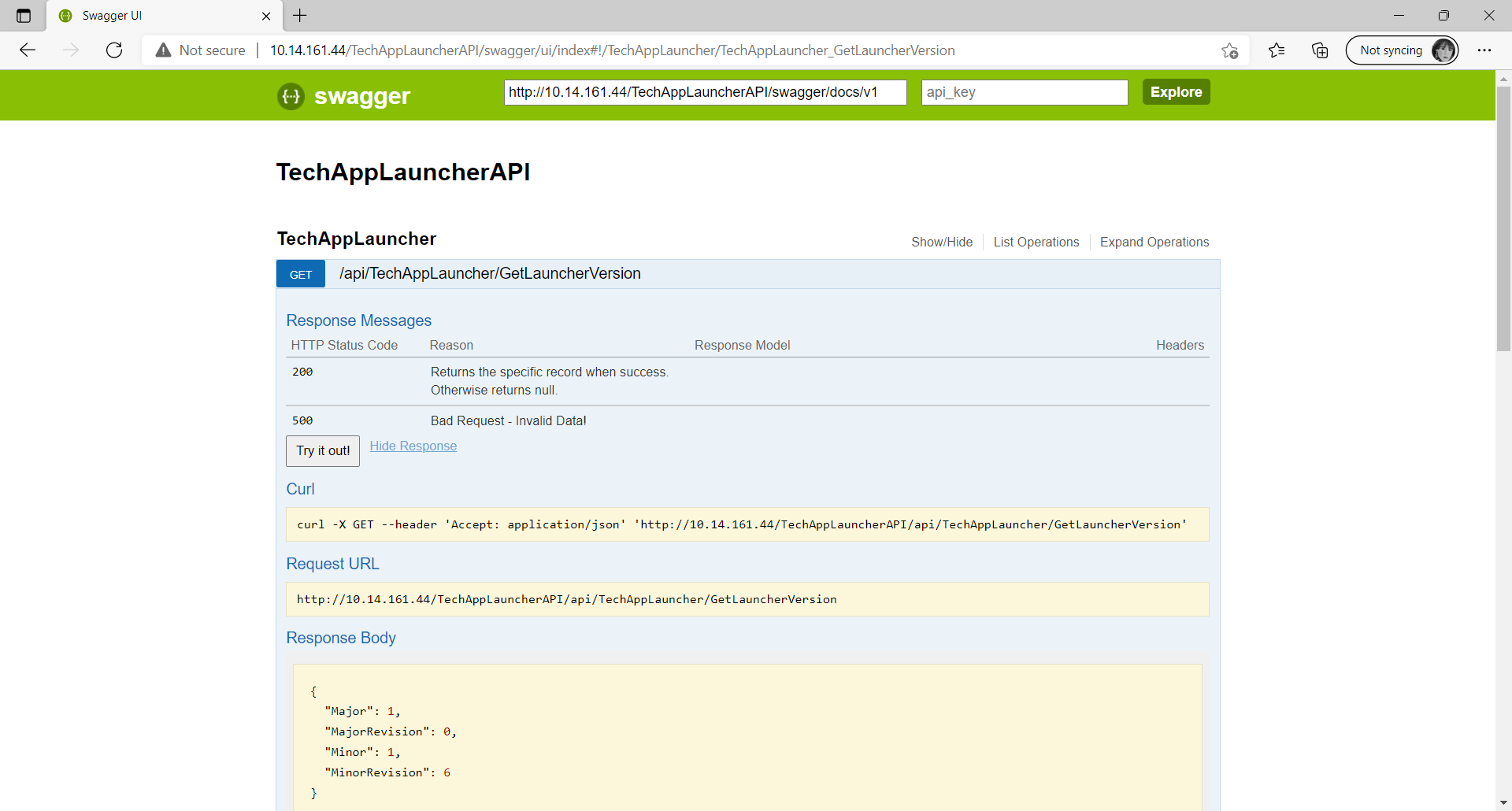
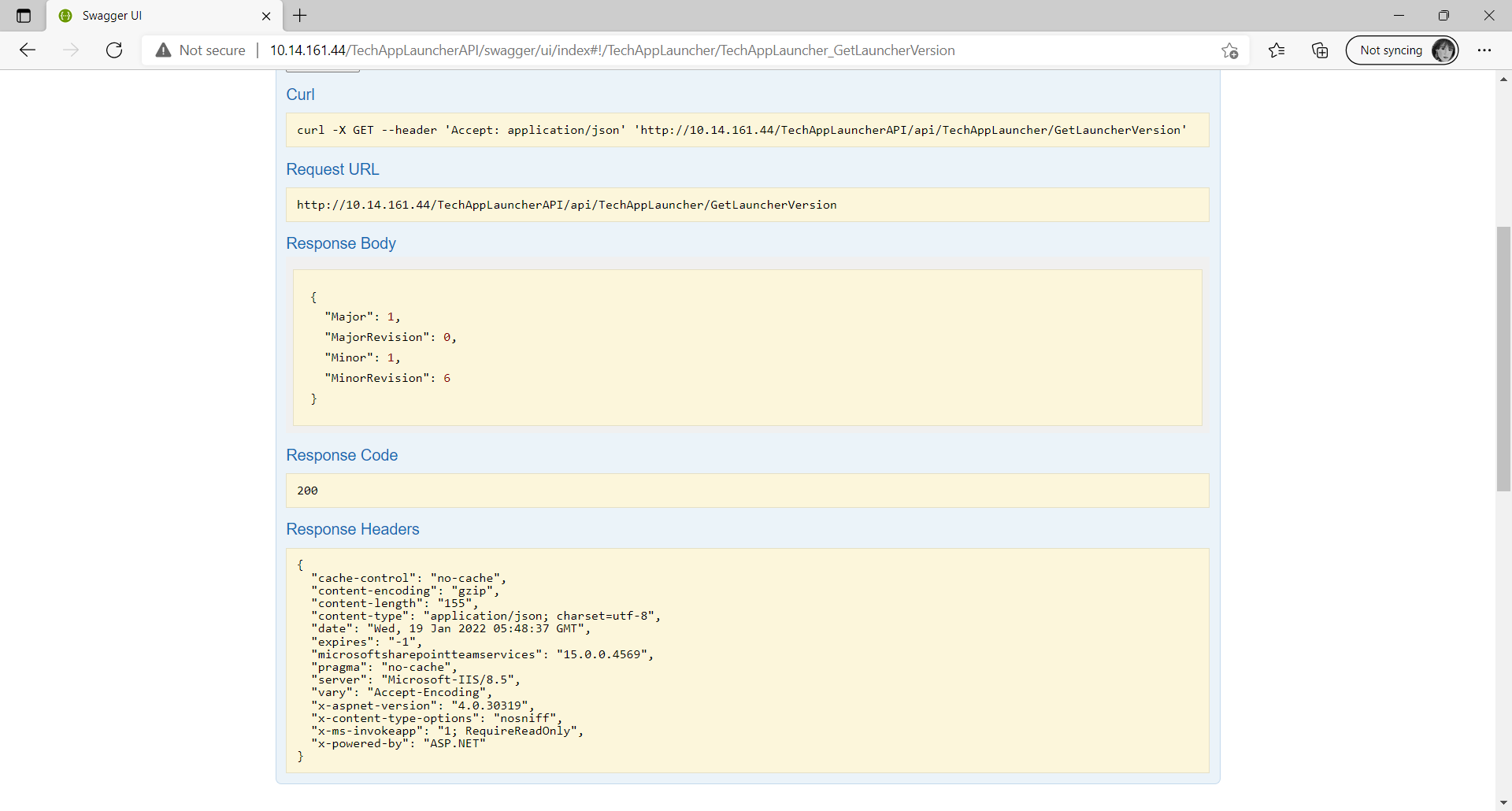
2

1

To publish a newer version of the app, right-click on the project in the solution explorer and then choose ‘**Publish…**’ button[1]. The wizard of publishing a newer version of the application will appear. Click on the ‘**Publish**’ button to proceed. Upon successful, the package of the application is located at ‘**bin\app.publish**’[2].

**To deploy the API to the server** – copy the entire content of the folder ‘**bin\app.publish**’ to the folder ‘**E:\TechAppLauncherAPI**’ at the server ’**10.14.161.44**’.

**About Swagger UI**



1

2

3

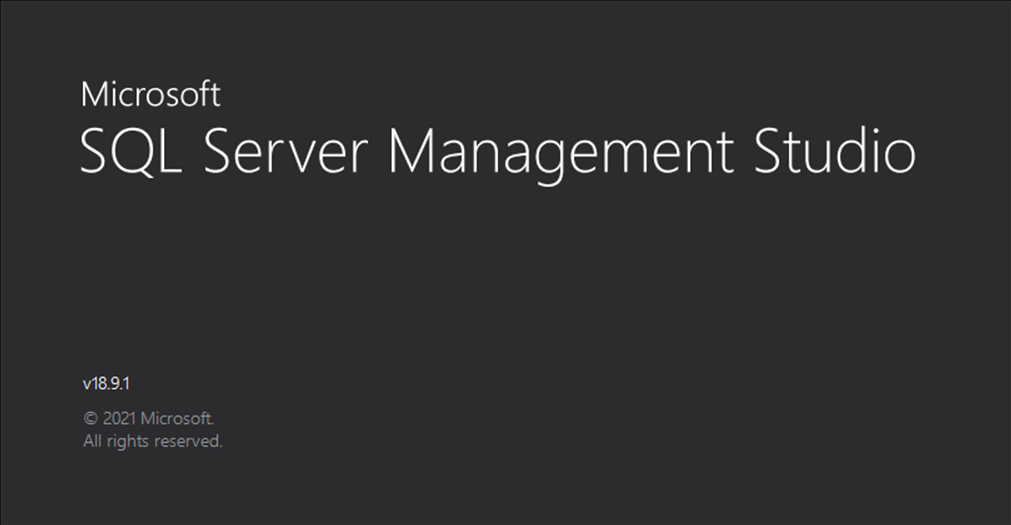
4

5

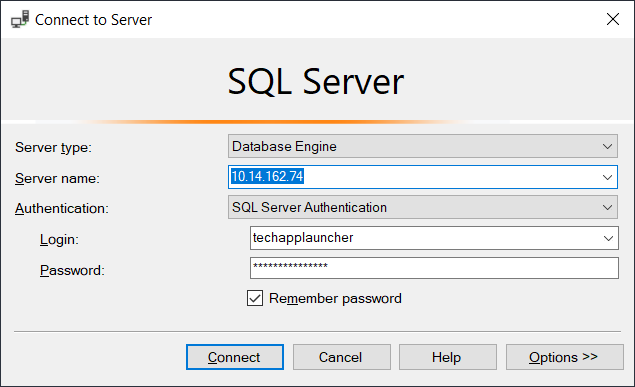
6

1. Shows the endpoint of the API
2. Provides the summary of the HTTP Response Message.
3. The button provides the feature to test the endpoint.
4. Shows the fully qualified URL for the specific endpoint.
5. Shows the HTTP Response body when success.
6. Shows the HTTP Response Status Code.

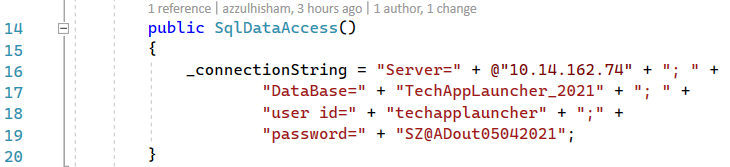
**Managing SQL Server**

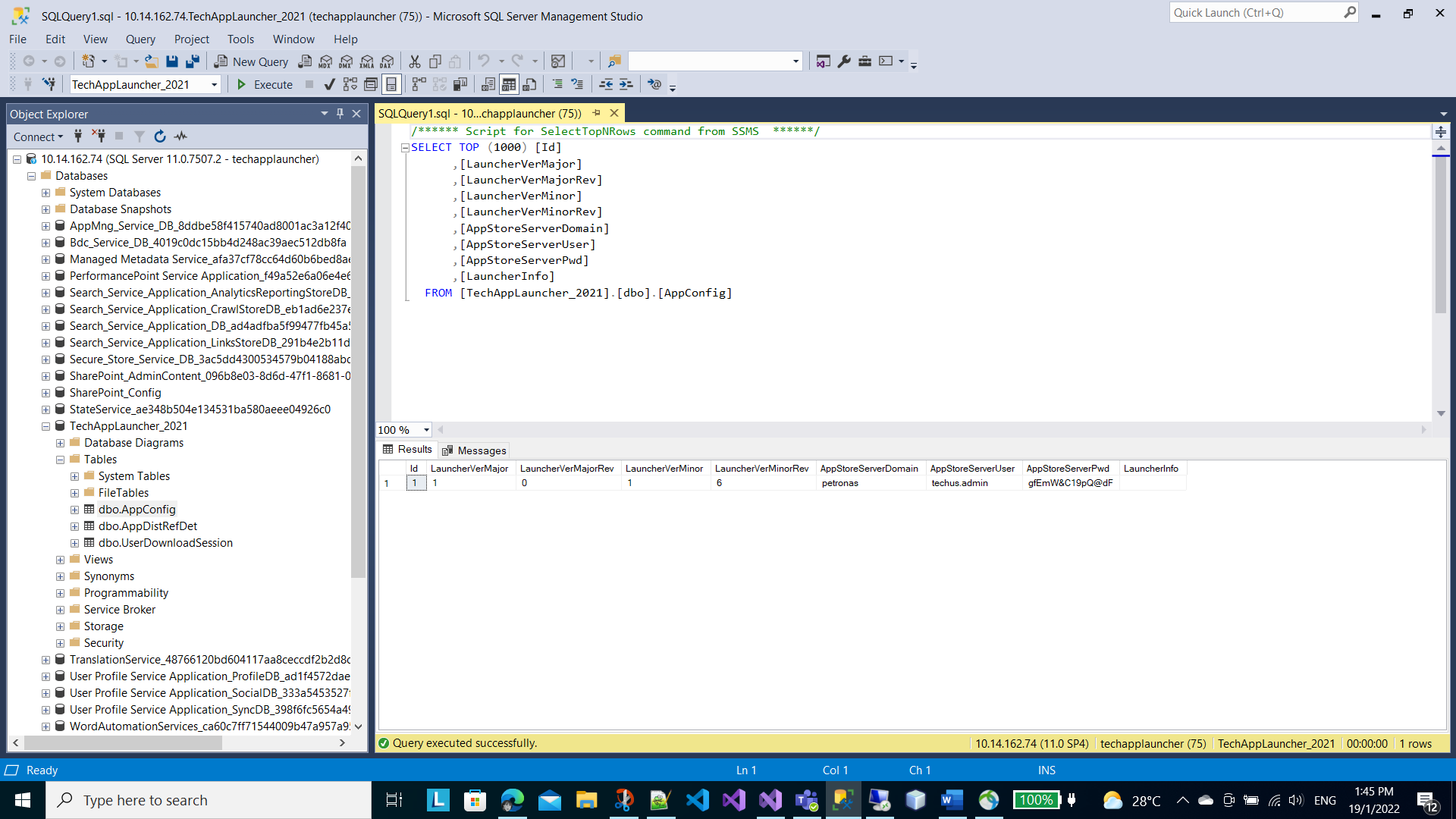


It is recommended to use the **Microsoft SQL Server Management Studio** in order to manage the SQL database for TechAppLauncher. It can download from the Microsoft official website: <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>



The **username** and the **password** as well as the database being used by the application can be found in the repository of the application.





**Microsoft SQL Database Server**

You try to download a free version of SQL server (**SQL Express**) and set it up on your system in order to get familiar with it. The link below is the official website to download the **SQL Express**.

<https://www.microsoft.com/en-gb/download/details.aspx?id=101064>

**Database Tables**

There are 3 tables currently in used in this project.

1. AppConfig
2. AppDistRefDet
3. UserDownloadSession

The script below is to create ‘**AppConfig**’ table:

CREATE TABLE [AppConfig](

[Id] [bigint] IDENTITY(1,1) NOT NULL,

[LauncherVerMajor] [int] NOT NULL,

[LauncherVerMajorRev] [int] NOT NULL,

[LauncherVerMinor] [int] NOT NULL,

[LauncherVerMinorRev] [int] NOT NULL,

[AppStoreServerDomain] [nvarchar](32) NOT NULL,

[AppStoreServerUser] [nvarchar](32) NOT NULL,

[AppStoreServerPwd] [nvarchar](16) NOT NULL,

[LauncherInfo] [nvarchar](50) NULL

) ON [PRIMARY]

The script below is used to create ‘**AppDistRefDet**’:

CREATE TABLE [AppDistRefDet](

[Id] [bigint] IDENTITY(1,1) NOT NULL,

[AppUID] [nvarchar](64) NOT NULL,

[LinkID] [nvarchar](150) NOT NULL,

[AgentName] [nvarchar](50) NOT NULL,

[Description] [nvarchar](250) NULL,

CONSTRAINT [uc\_AppUID] UNIQUE NONCLUSTERED

(

[AppUID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

And the script below is used to create the ‘**UserDownloadSession**’:

CREATE TABLE [UserDownloadSession](

[Id] [bigint] IDENTITY(1,1) NOT NULL,

[AppId] [bigint] NULL,

[AppUID] [nvarchar](50) NOT NULL,

[Title] [nvarchar](250) NULL,

[Status] [nvarchar](50) NULL,

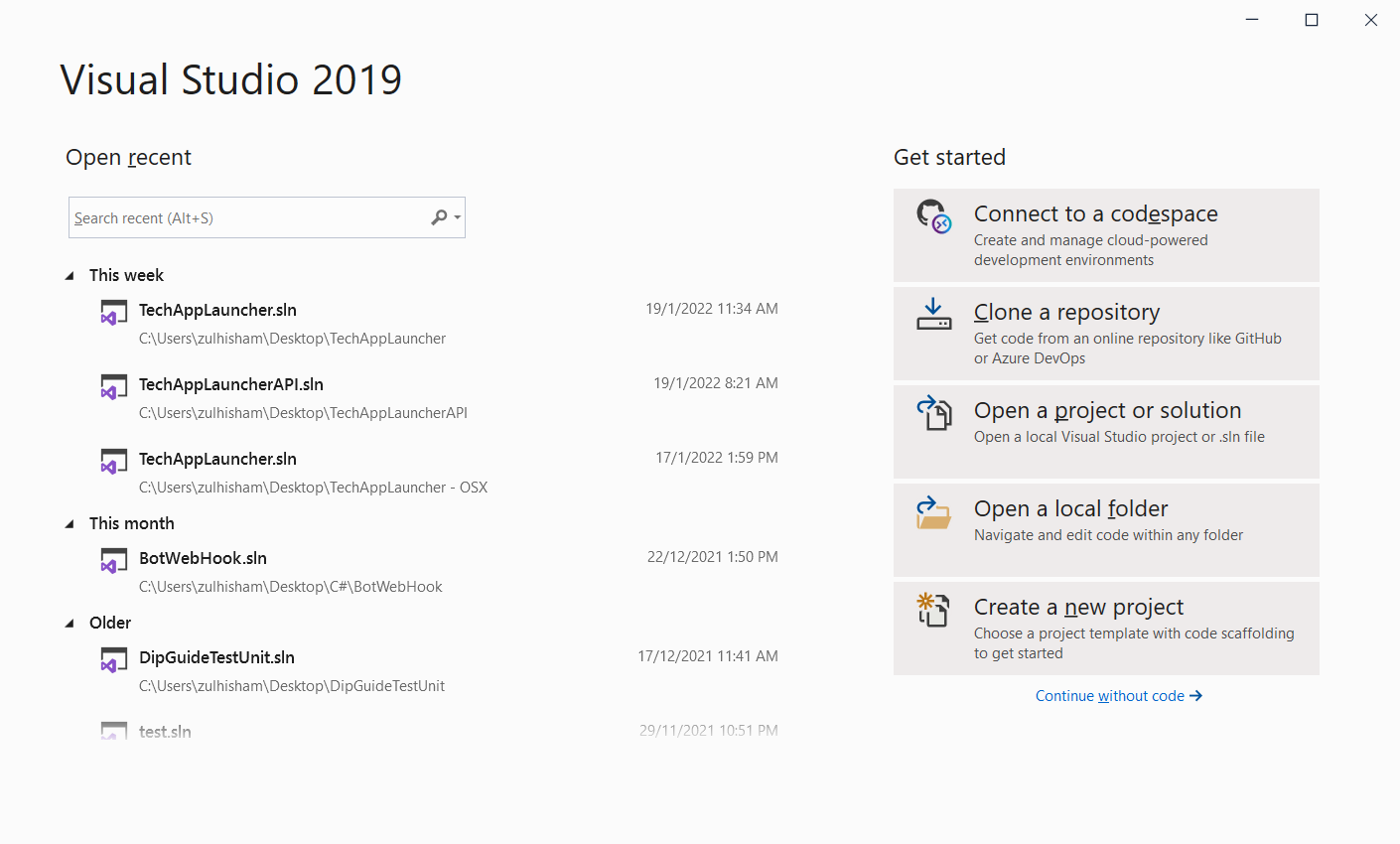
[UserName] [nvarchar](150) NOT NULL,

[InstallTimeStamp] [datetime] NOT NULL,

[Remark] [nvarchar](50) NULL

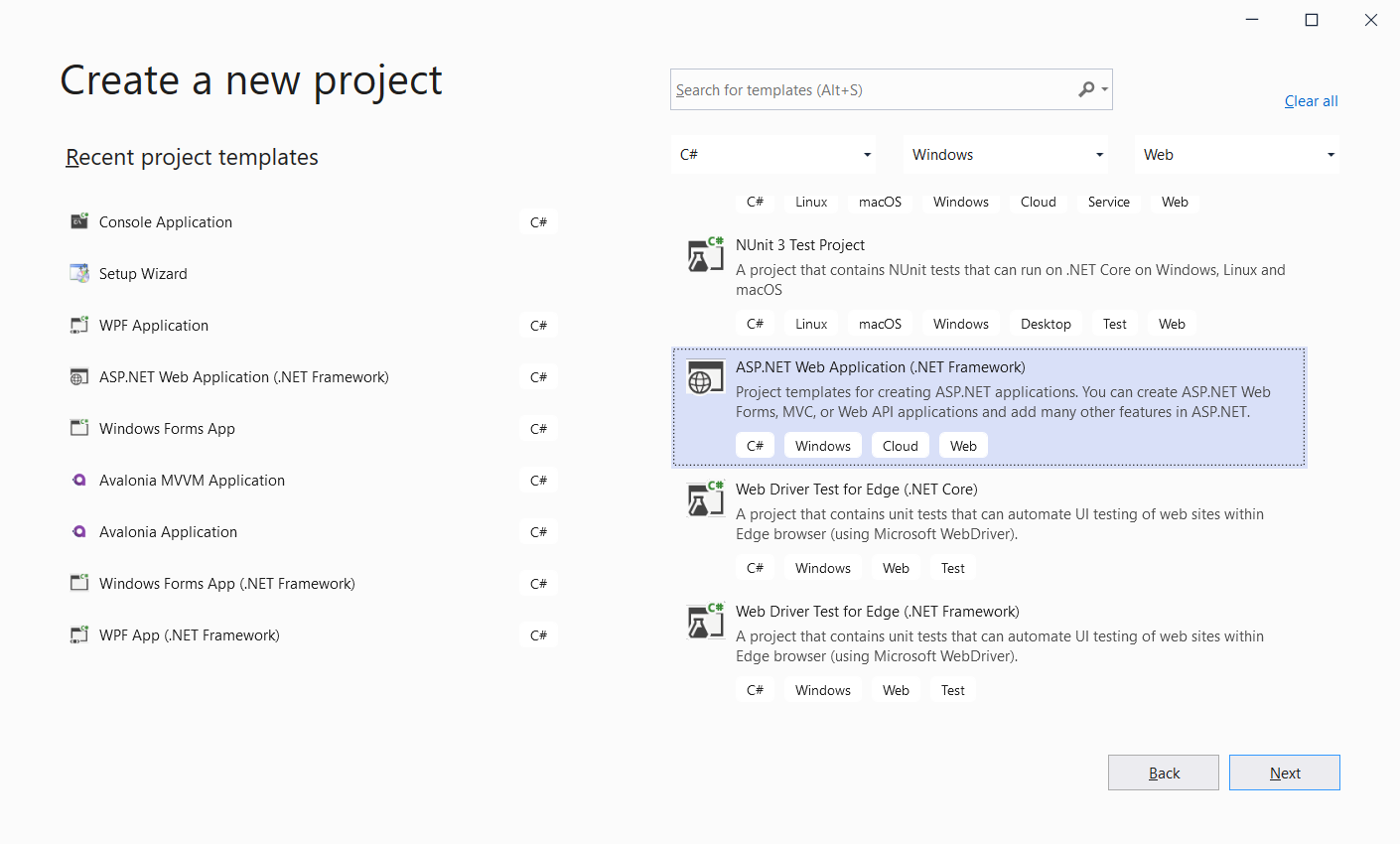
) ON [PRIMARY]

**A Quick Tutorial To Get The WebAPI Project Start Up By Using Visual Studio 2019**



1

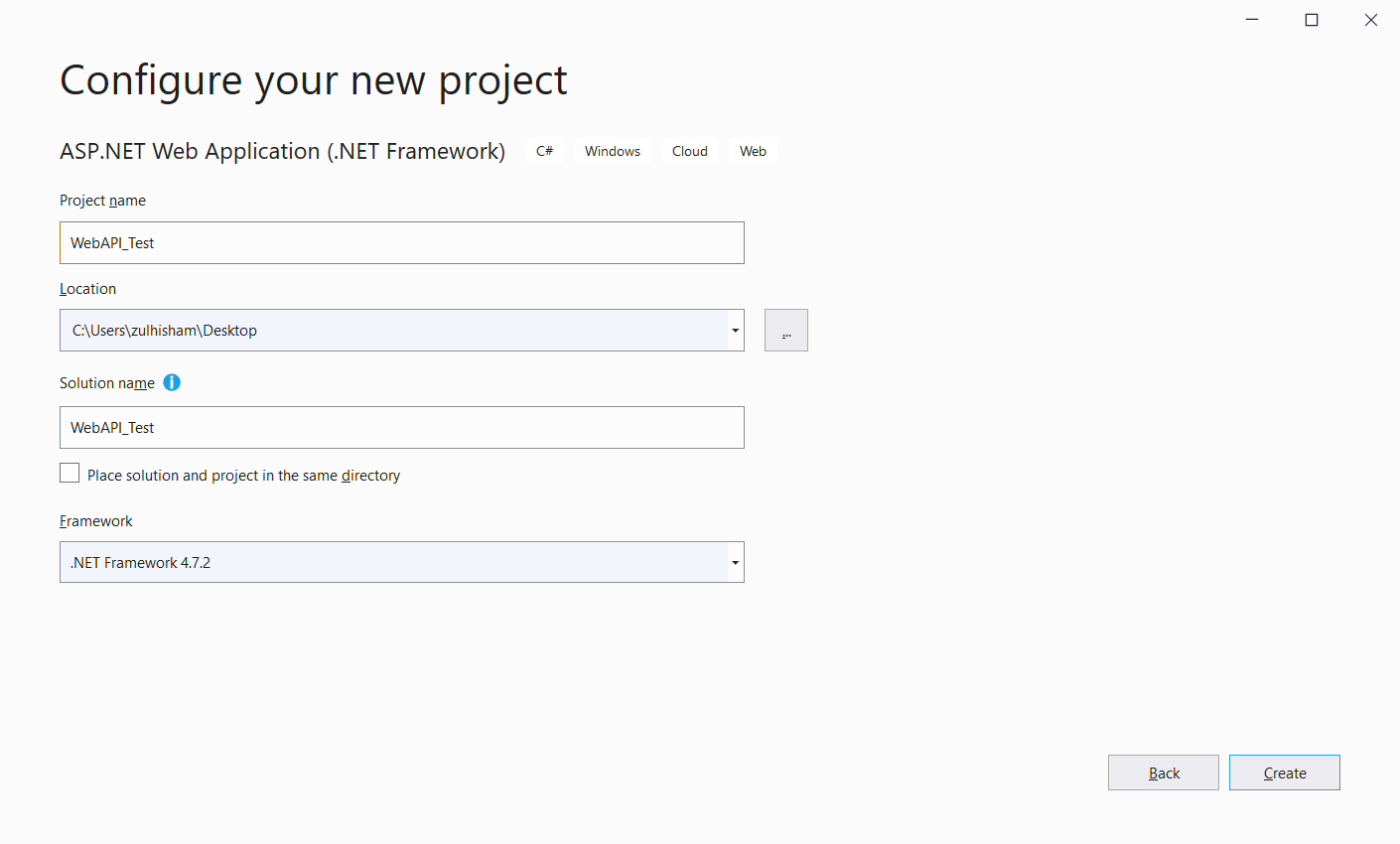
Now… let get your Visual Studio 2019 up and run. In the ‘**Get started**’ page, choose the ‘**Create a new project**’[1] as shown in the screenshot above.



3

2

In the ‘**Create a new project**’ page, search for ‘**ASP.NET Web Application (.Net Framework)**’[2], then click the ‘**Next**’ button[3] to proceed.



8

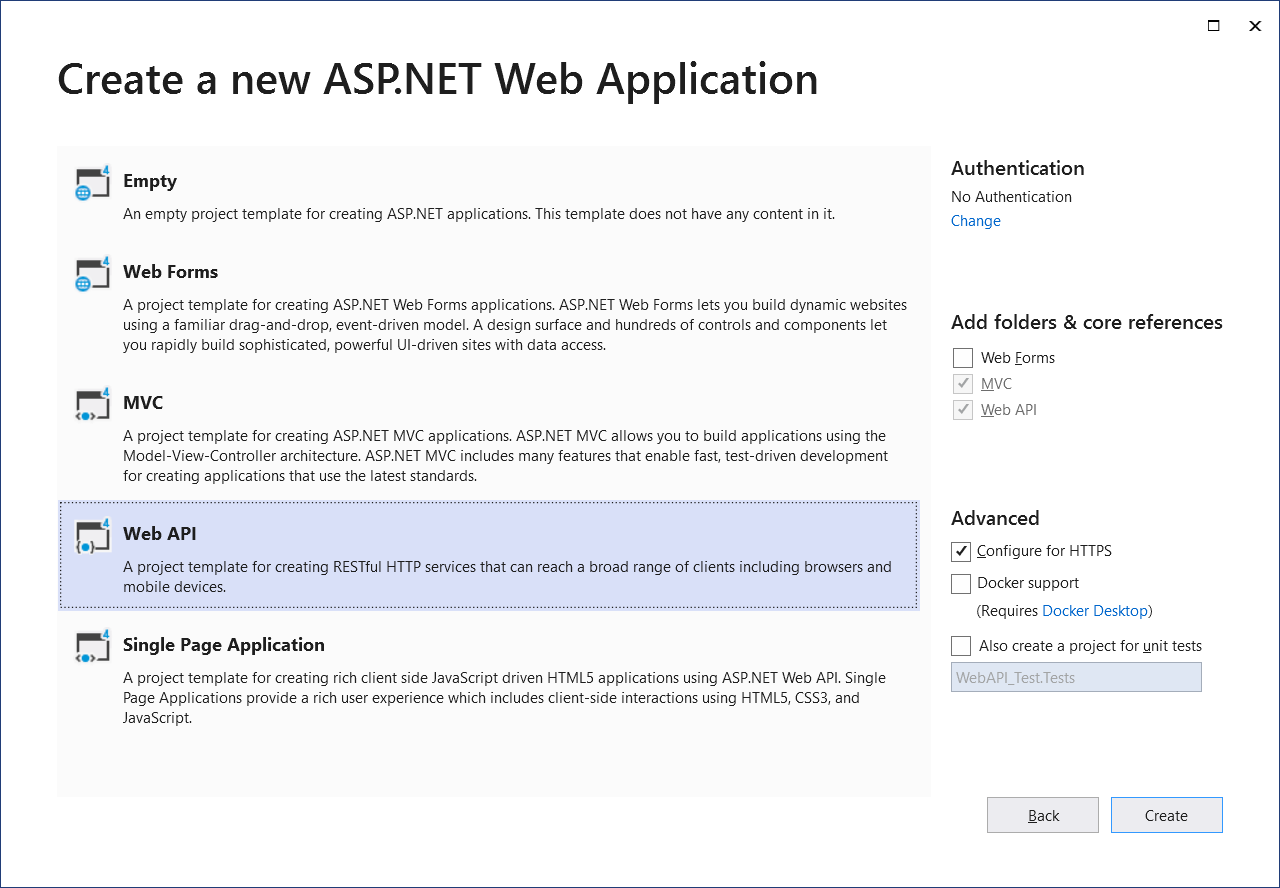
7

6

5

4

In the ‘**Configure your new project**’ page, give a name for your project[4]. Then choose a location or path to place your project files[5]. You can leave the **Solution Name**[6] as the Visual Studio provided to you. Finally, select the appropriate **.net framework**[7] for your project. In this example, we will use ‘**.Net Framework 4.7.2**’. Once done, click the ‘**Create**’ button[8] to let Visual Studio to start creating your project.

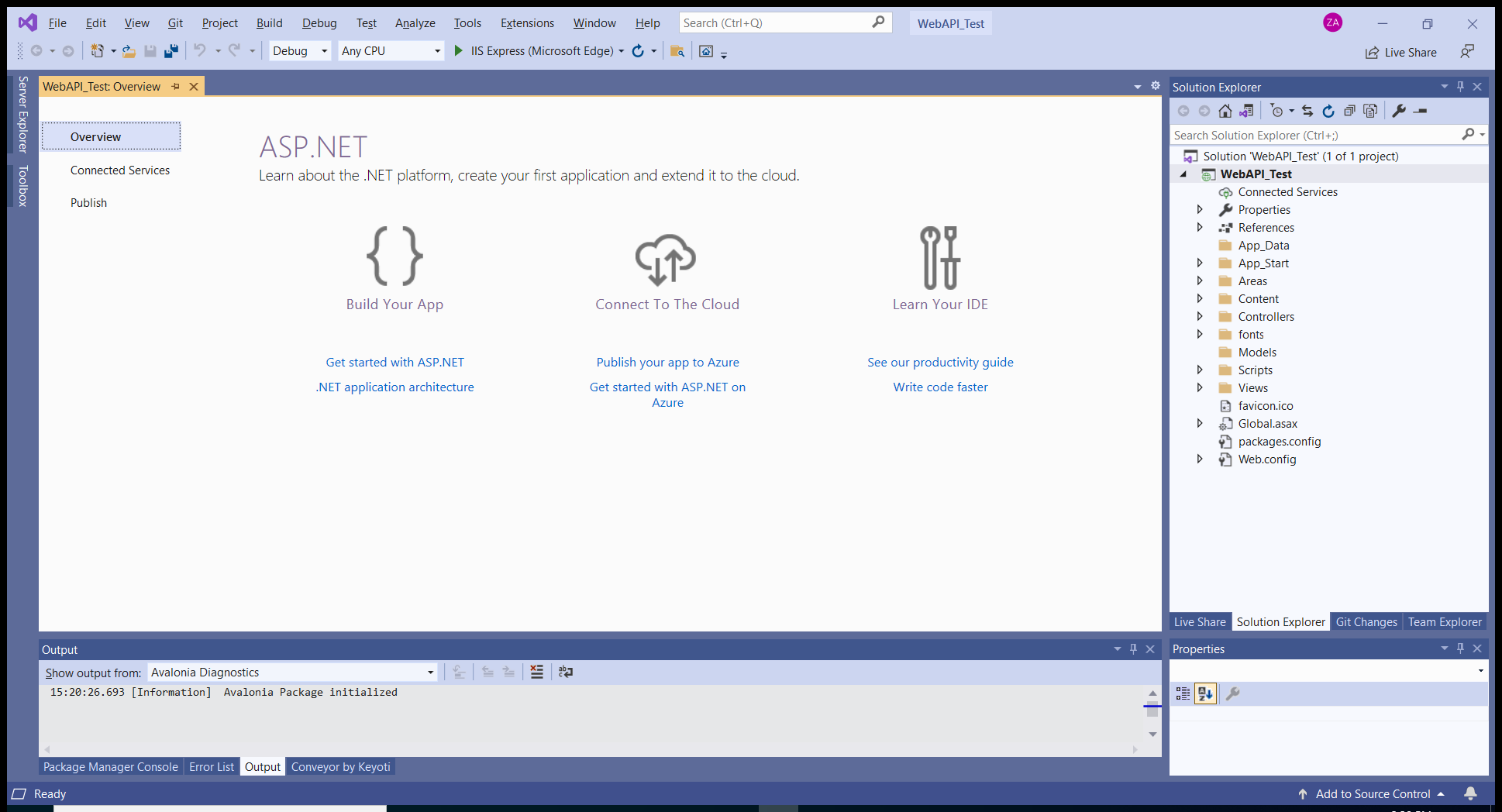


11

10

9

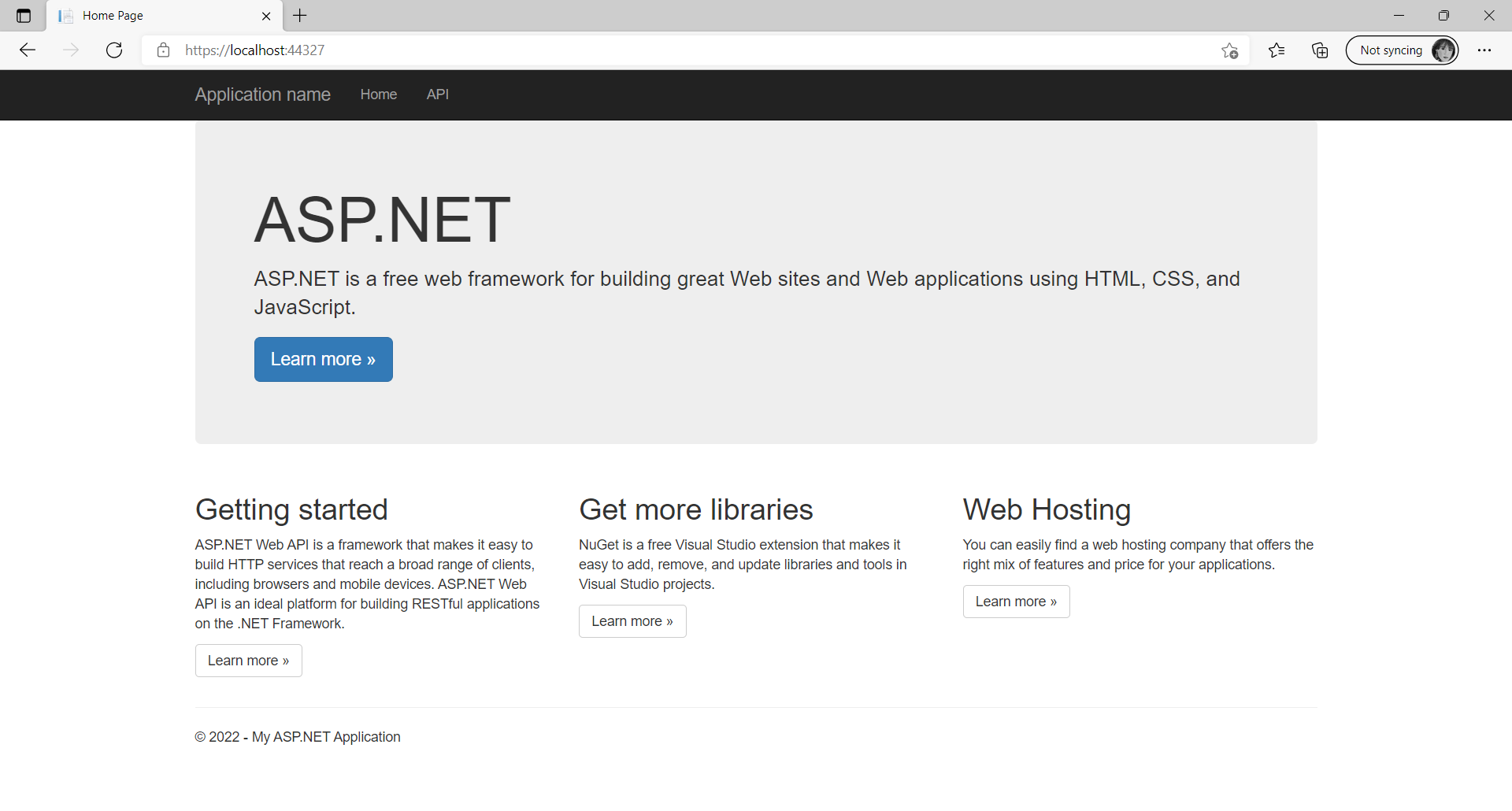
In the ‘**Create a new ASP.NET Web Application**’ page, choose ‘**WebAPI**’ template[9]. And leave the **Authentication** with ‘**No Authentication**’ as well as others setting remain as default[10]. The authentication is another huge topic to discuss. Click ‘**Create**’ button[11] to start create your project.



12

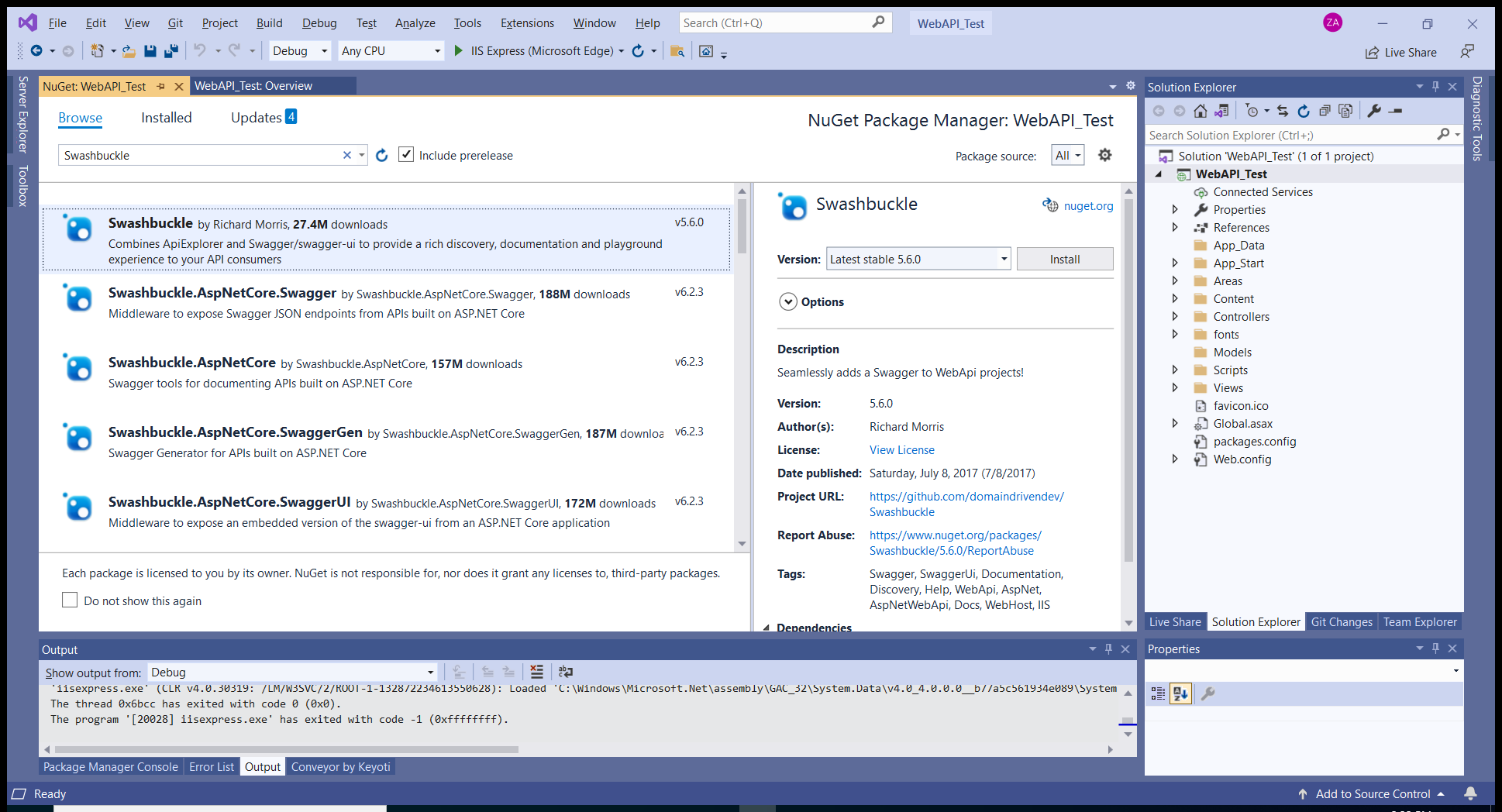
Visual Studio will takes awhile to generate your project files. Once it is done, you should see something similar to the screenshot above.

Now, try to run your first ASP.NET application by click on the ‘**Run**’ button[12] as shown in the screenshot above or press ‘**F5**’ to start the Web Application.



And if you see something similar shown in the screenshot just above in your internet browser, that’s mean your ASP.NET application is up and run. **Hurray……. Congratulation……………!**

Now, let add the **Swagger** page into your project. Go to the ‘**NuGet Package Manager**’ by right-click on your project in the ‘**Solution Explorer**’, then choose ‘**Manage NuGet Packages…**’.



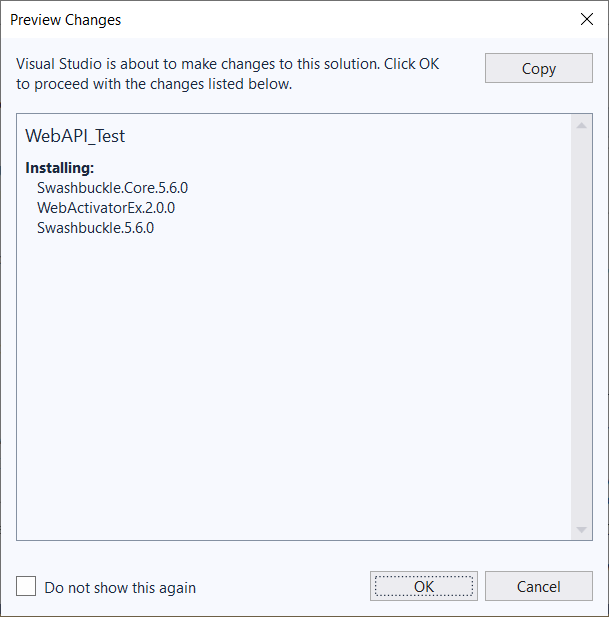
16

15

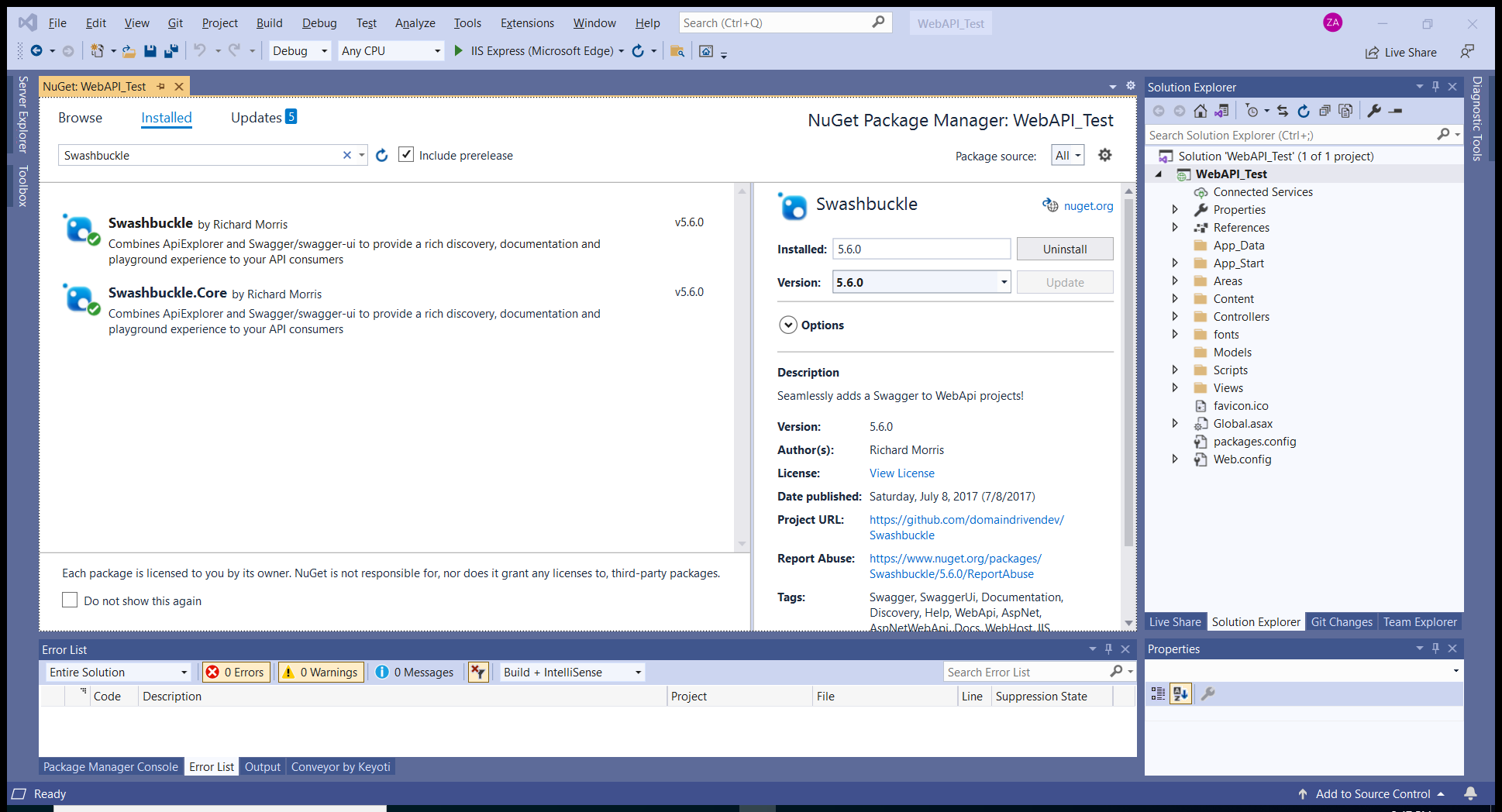
13

14

In the ‘**NuGet Package Manager**’, select ‘**Browse**’[13] tab and search for ‘**Swashbuckle**’[14]. Then select the ‘**Swashbuckle**’ package[15] to install it by click on the ‘**Install**’ button[16].



Visual Studio will prompt you the changes that it is going to make to your project file as shown in the screenshot above. Click ‘**OK**’ to proceed.

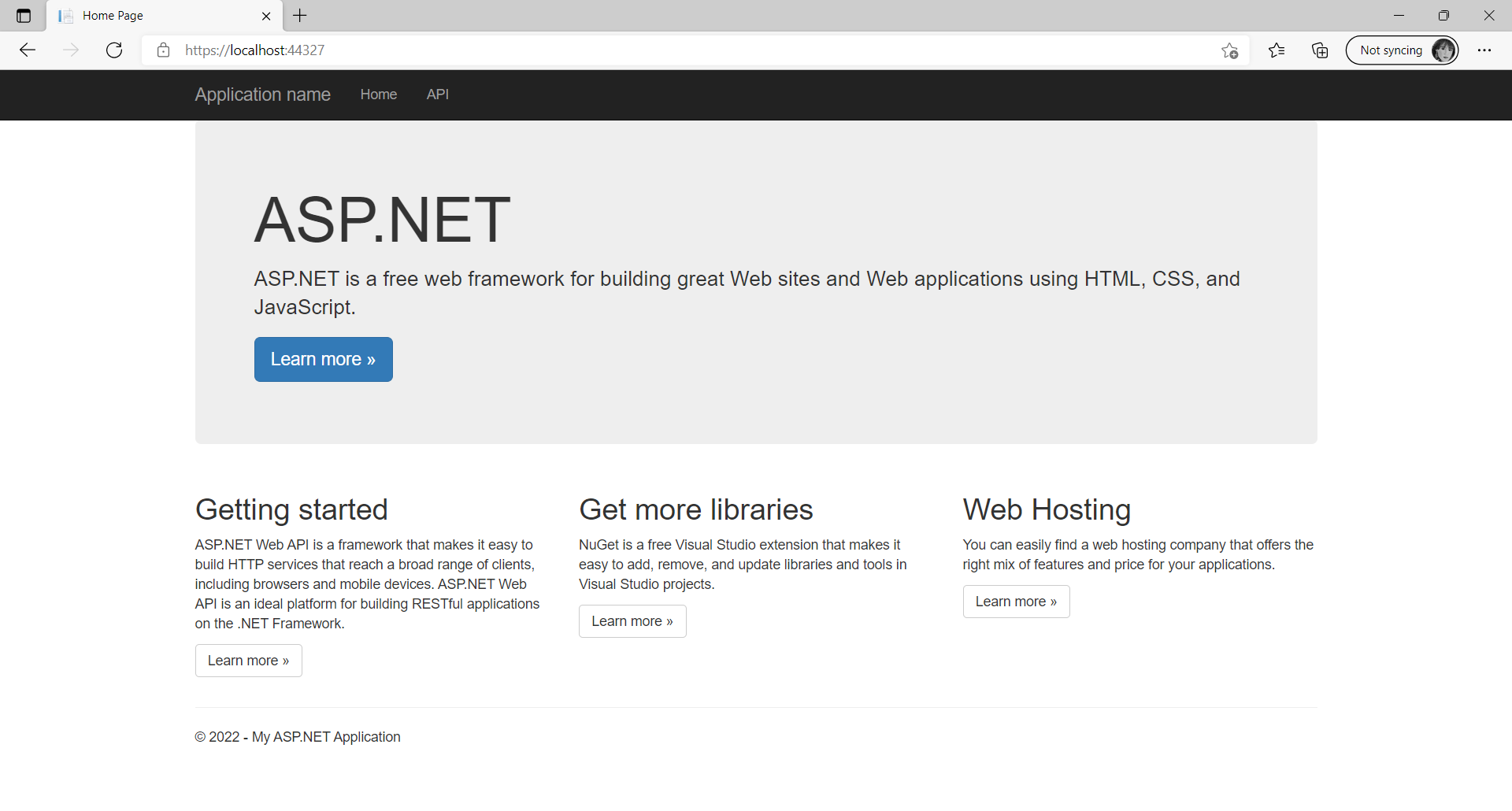


18

17

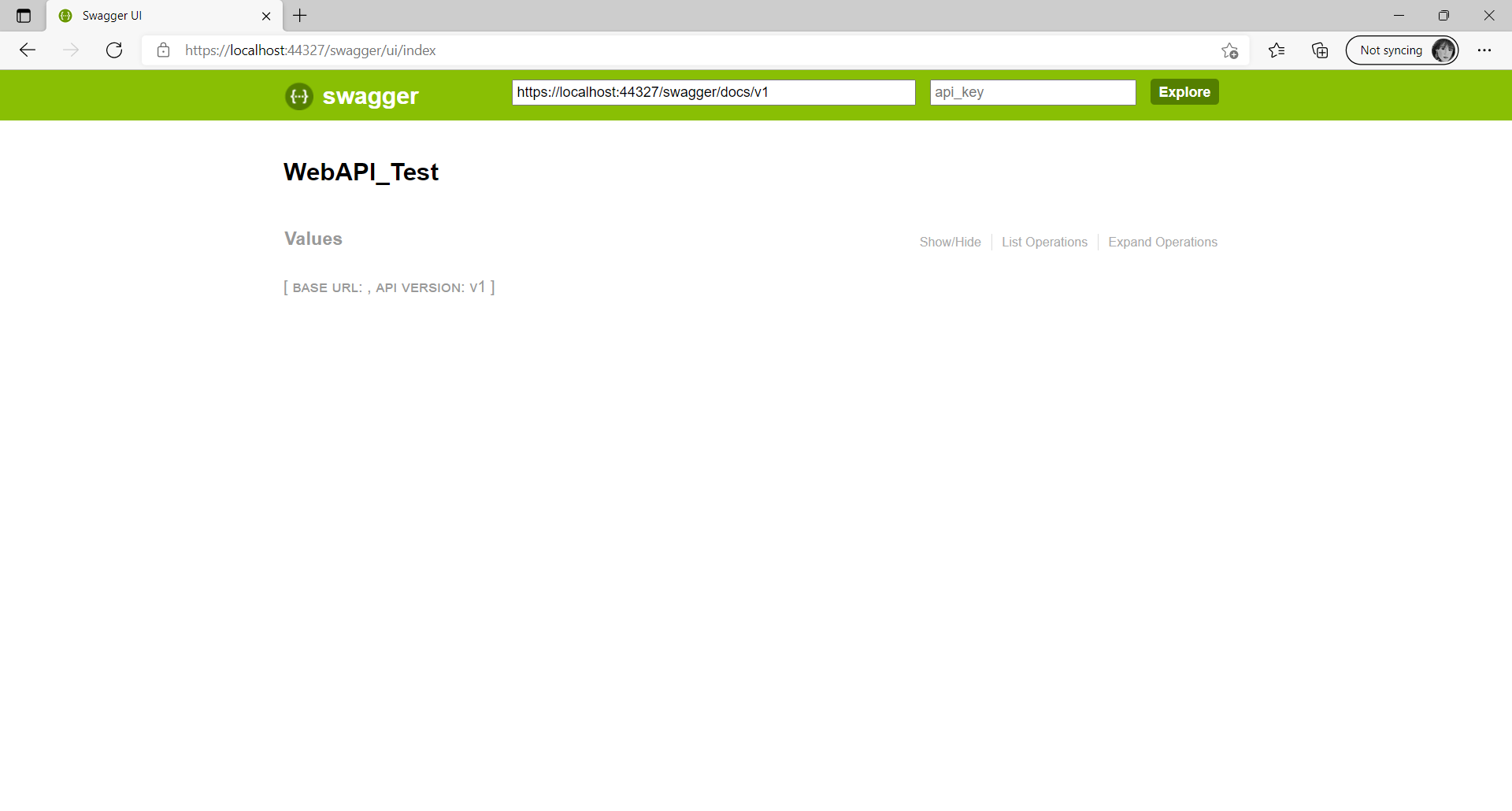
Once done, switch to ‘**Installed**’ tab[17], the ‘**Swashbuckle**’ should have installed to your project. From here, you can remove it if you don’t like Swagger page later on. Just select the package and then click the ‘**Uninstall**’ button[18] in order to remove it.

Now, you can close the ‘**NuGet Package Manager**’ and then try to run the project again.



Well, nothing has been changed. You will see the web page just as before. **Damn…….! Where is the Swagger page?**

Well, don’t panic….! Try to add ‘**/swagger**’ after the ‘**https://localhost:xxxxx**’ at the address bar and then hit ‘**Enter**’ key. **Bomb…. The swagger page is loaded!**

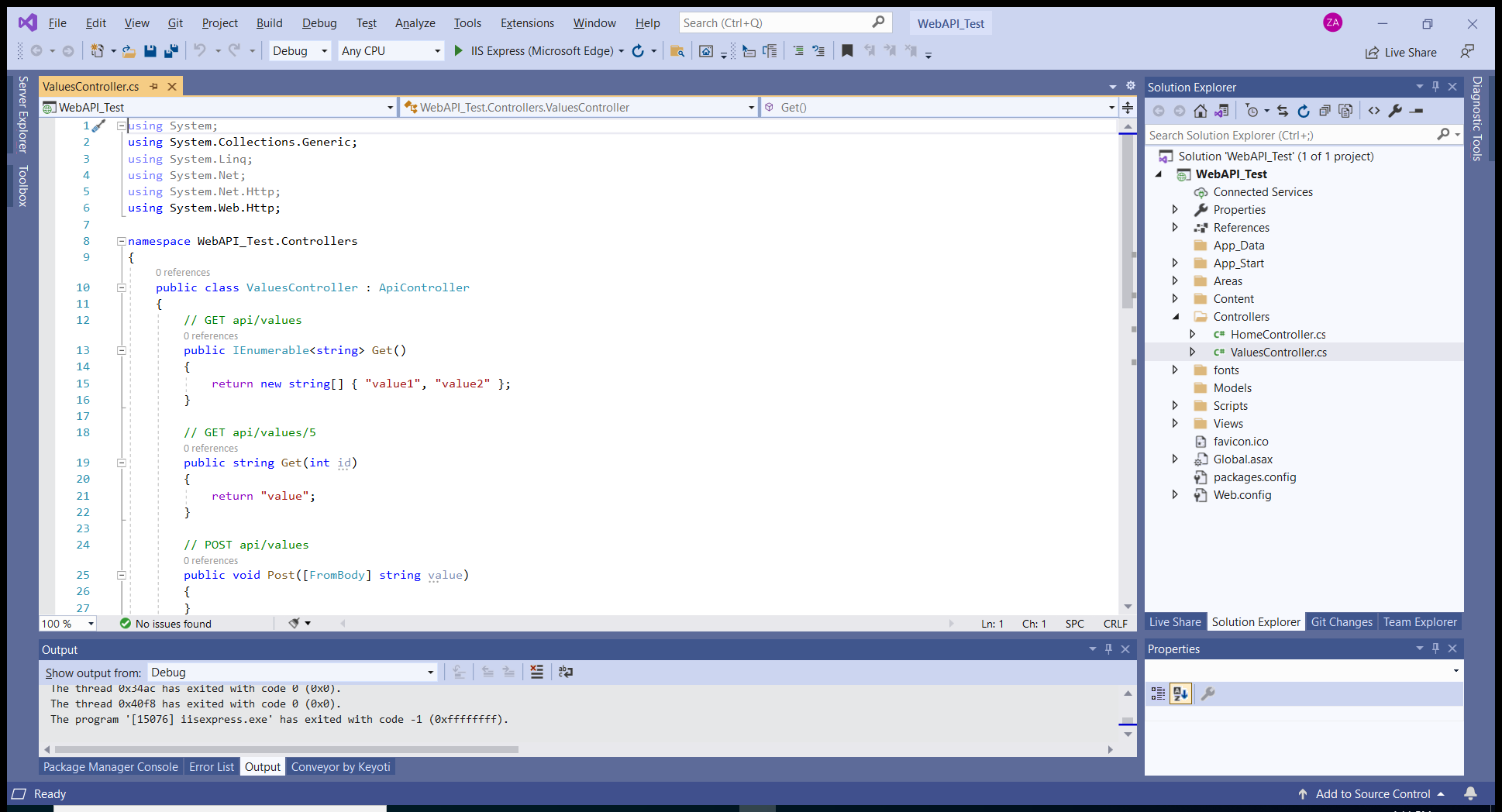


19

Now, expend the ‘**Values**’ API[19]. You notice some endpoints is already exists. But where does the C# code for these endpoints?

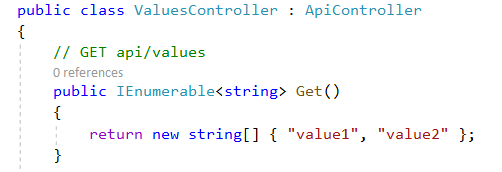


Go back to Visual Studio……You can also close the browser in order to stop the application or press the ‘**Stop**’ button at Visual Studio to stop it.

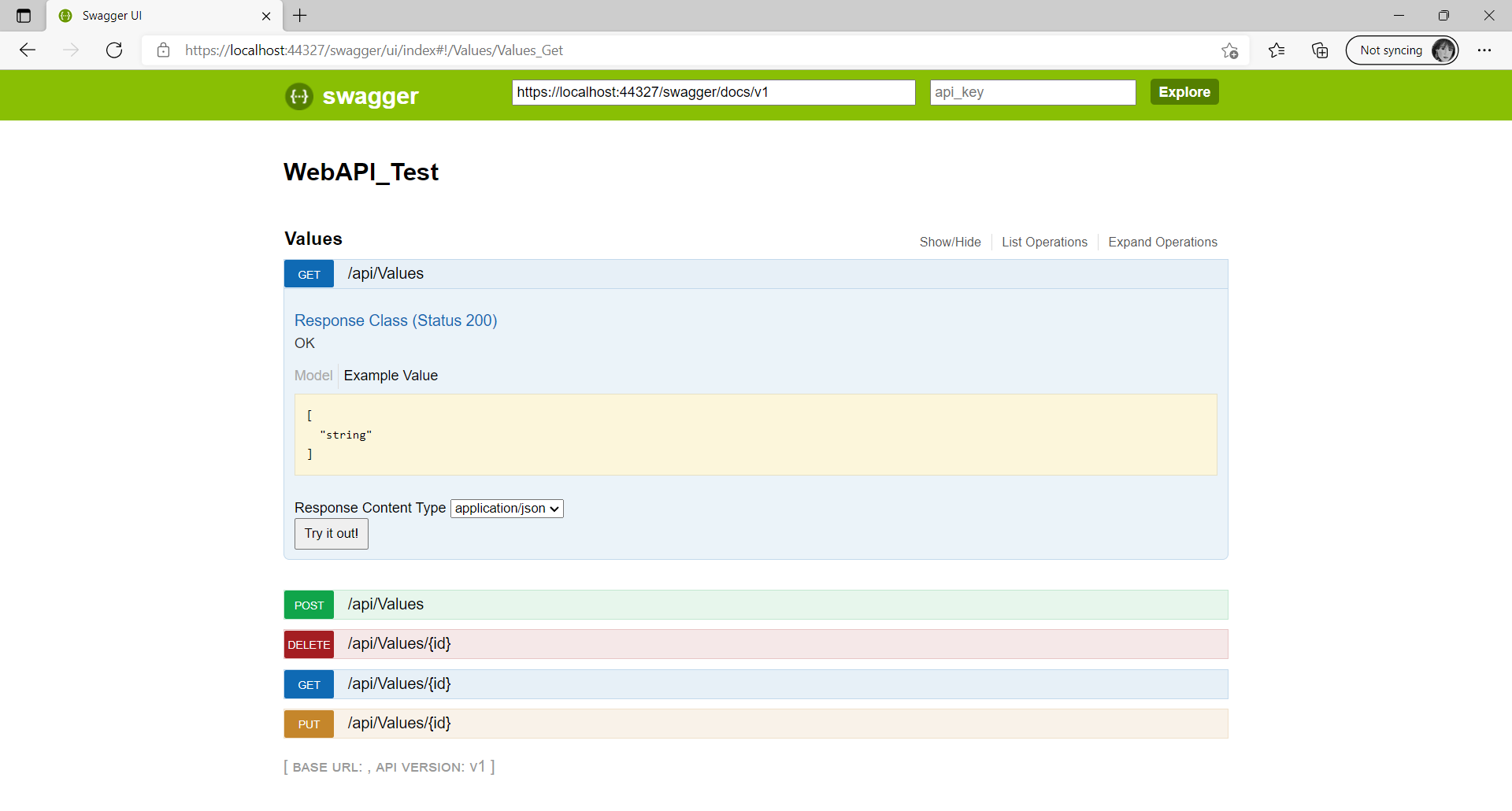


20

Expend the ‘**Controllers**’ folder[20] in the ‘**Solution Explorer**’ as shown above. And then double-click on the ‘**ValuesController.cs**’ file, all the C# code for the ‘**Values**’ API are located in this C# file.



Let look at the code above, which is a ‘**Get**’ endpoint. The code in this endpoint is just returning an array of string whenever it is being hit. Now, let examine this by starting up your web application.



21

Once the web application is up and run, go to its **swagger page** and then hit the ‘**Try it out**’ button[21] as shown in the screen above.



23

22

You will notice that the response body is an **array of string which is in json format**[22]. Compare to the C# code, the response body is exactly what the C# code returns.

While the web application is still running, you can try test the endpoint using ‘**Postman**’. Before start up your ‘**Postman**’, make a copy of the Request URL[23].



27

26

25

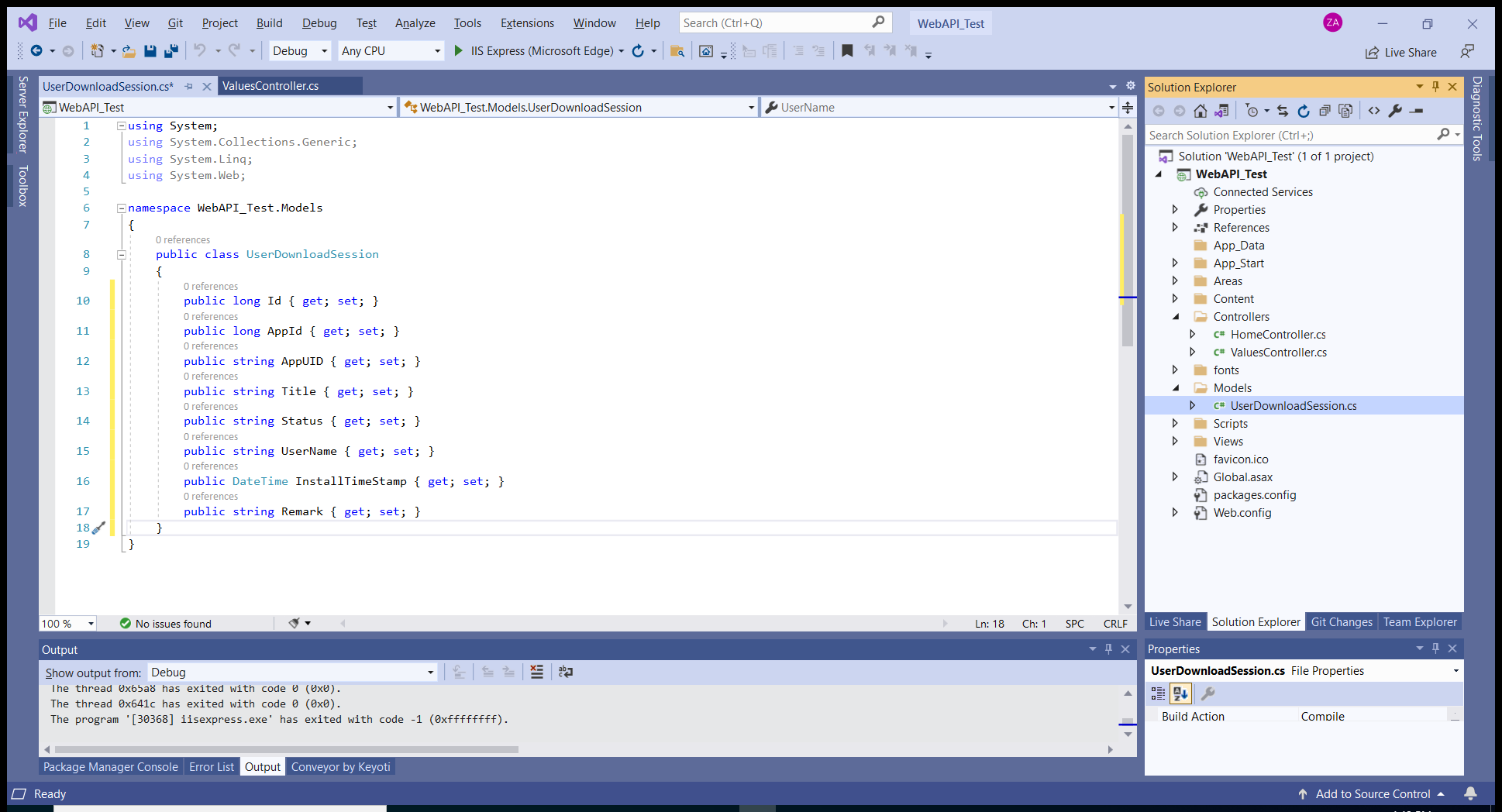
24

Paste the URL to the address bar[24]. And change the HTTP request to ‘**Get**’[25] method, then press ‘**Send**’ button[26]. You will notice the response body[27] is same as the swagger page shows to you. This is because when you press the ‘**Send**’ button, the same endpoint is being hit, and the same C# code is running at the backend. Hence you got the same response as in the swagger page.

Now… Let do something interesting! Stop the application if the Web application is still running. Right-click the project in the solution explorer and then create a new folder call ‘**Models**’. If it is already exists, then you can skip this step to create the folder.

Add a new C# class file into the ‘**Models**’ folder and name the file as ‘**UserDownloadSession.cs**’. Then add the following C# code into the file.

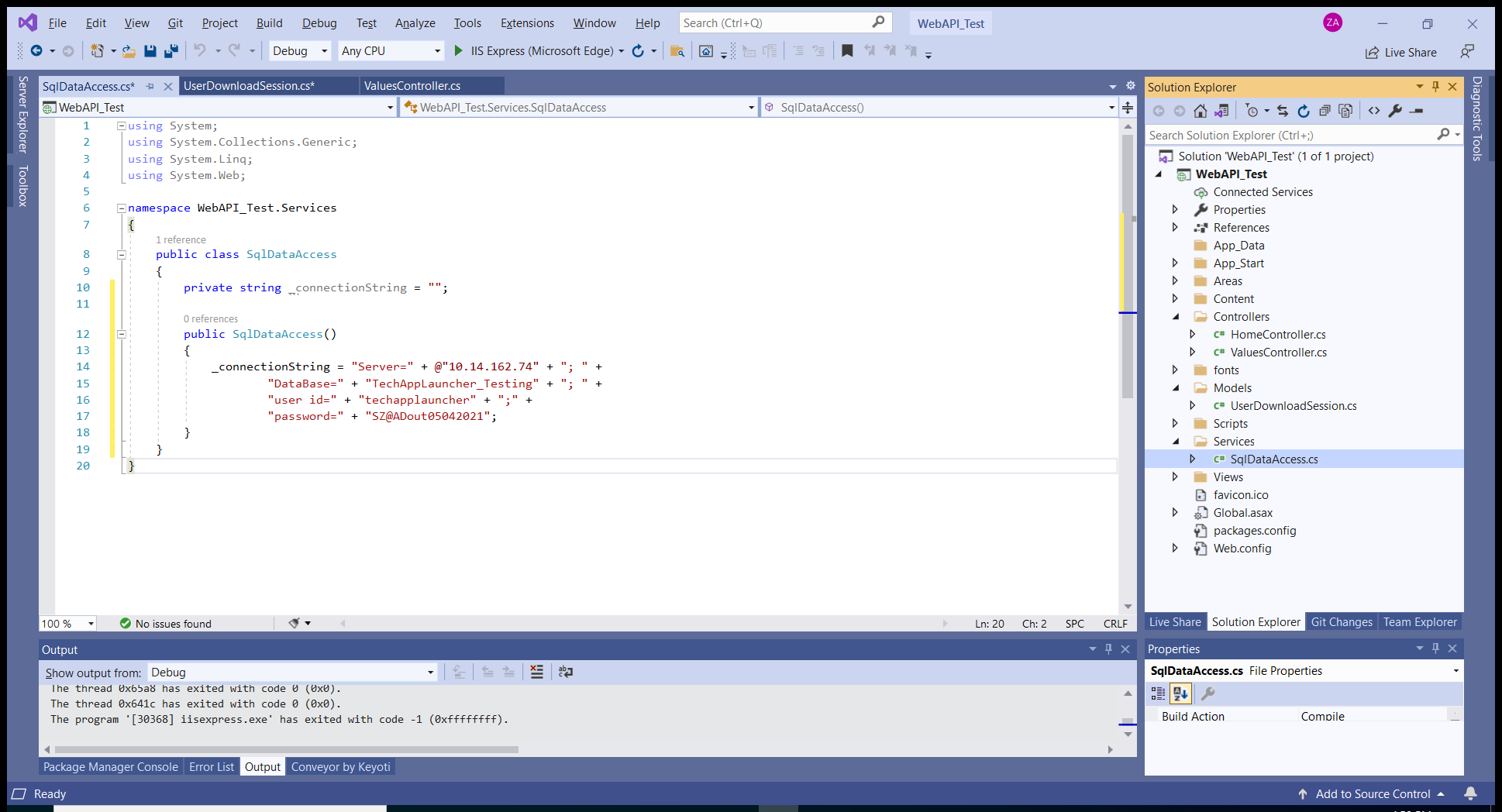




Your entire project files should have something similar to the screenshot above.

Next, right-click the project in the solution explorer and then create a new folder call ‘**Services**’. Then add a new C# class file into the ‘**Services**’ folder and name the file as ‘**SqlDataAccess.cs**’. Also, add the following code to the file. Please note that the connection string is redirecting to the testing database ‘**TechAppLauncher\_Testing**’.





Your entire project files now should have something similar to the screenshot above.

Now, copy the following C# code to the ‘**SqlDataAccess.cs**’ file. It is also necessary for you to add the 2 lines of codes below at top of the C# file.

using System.Data.SqlClient;

using WebAPI\_Test.Models;

public List<UserDownloadSession> GetUserDownloadSessions()

{

SqlConnection dbConnection = new SqlConnection(\_connectionString);

string \_qry = "SELECT \* FROM UserDownloadSession ORDER BY InstallTimeStamp DESC";

List<UserDownloadSession> userDownloadSessions = null;

try

{

dbConnection.Open();

SqlCommand \_qrycmd = new SqlCommand(\_qry, dbConnection);

//\_qrycmd.ExecuteNonQuery();

SqlDataReader Reader = \_qrycmd.ExecuteReader();

if (Reader.HasRows)

{

userDownloadSessions = new List<UserDownloadSession>();

while (Reader.Read())

{

userDownloadSessions.Add(new UserDownloadSession

{

Id = Reader.GetInt64(0),

AppId = Reader.IsDBNull(1) ? 0 : Reader.GetInt64(1),

AppUID = Reader.GetString(2),

Title = Reader.IsDBNull(3) ? "" : Reader.GetString(3),

Status = Reader.IsDBNull(4) ? "" : Reader.GetString(4),

UserName = Reader.GetString(5),

InstallTimeStamp = Reader.GetDateTime(6),

Remark = Reader.IsDBNull(7) ? "" : Reader.GetString(7)

});

}

}

}

catch (Exception Ex)

{

string msg = Ex.Message;

}

finally

{

dbConnection.Close();

}

return userDownloadSessions;

}

The code above is actually add a function ‘GetUserDownloadSessions()’ which return a list of ‘UserDownloadSession’ in the ‘**SqlDataAccess.cs**’ file.

Again… copy the entire code below to the ‘**SqlDataAccess.cs**’ file.

public List<UserDownloadSession> AddUserDownloadSession(UserDownloadSession userDownloadSession)

{

int \_ret = 0;

SqlConnection dbConnection = new SqlConnection(\_connectionString);

string \_qry = "INSERT INTO UserDownloadSession VALUES (@AppId, @AppUID, @Title, @Status, @UserName, @InstallTimeStamp, @Remark)";

List<UserDownloadSession> userDownloadSessions = null;

try

{

dbConnection.Open();

SqlCommand \_qrycmd = new SqlCommand(\_qry, dbConnection);

\_qrycmd.CommandText = \_qry;

\_qrycmd.Parameters.AddWithValue("@AppId", userDownloadSession.AppId);

\_qrycmd.Parameters.AddWithValue("@AppUID", userDownloadSession.AppUID);

\_qrycmd.Parameters.AddWithValue("@Title", userDownloadSession.Title);

\_qrycmd.Parameters.AddWithValue("@Status", userDownloadSession.Status);

\_qrycmd.Parameters.AddWithValue("@UserName", userDownloadSession.UserName);

\_qrycmd.Parameters.AddWithValue("@InstallTimeStamp", userDownloadSession.InstallTimeStamp);

\_qrycmd.Parameters.AddWithValue("@Remark", userDownloadSession.Remark);

//\_qrycmd.ExecuteNonQuery();

SqlDataReader Reader = \_qrycmd.ExecuteReader();

\_ret = Reader.RecordsAffected;

if (\_ret > 0)

{

userDownloadSessions = GetUserDownloadSessions();

}

}

catch (Exception Ex)

{

string msg = Ex.Message;

}

finally

{

dbConnection.Close();

}

return userDownloadSessions;

}

Last but not least, add the following code into the ‘**ValuesController.cs**’ file.

private SqlDataAccess \_sqlDataAccess;

public ValuesController()

{

\_sqlDataAccess = new SqlDataAccess();

}

[HttpGet]

[Route("api/TechAppLauncher/GetAllUserDownloadSessions")]

[SwaggerResponse(HttpStatusCode.OK, "Returns the list of records when success. Otherwise returns null.")]

[SwaggerResponse(HttpStatusCode.InternalServerError, "Bad Request - Invalid Data!")]

public IEnumerable<UserDownloadSession> GetAllUserDownloadSessions()

{

return \_sqlDataAccess.GetUserDownloadSessions();

}

[HttpPost]

[Route("api/TechAppLauncher/AddUserDownloadSession")]

[SwaggerResponse(HttpStatusCode.OK, "Returns the specific record when success. Otherwise returns null.")]

[SwaggerResponse(HttpStatusCode.InternalServerError, "Bad Request - Invalid Data!")]

public IEnumerable<UserDownloadSession> AddUserDownloadSession([FromBody] UserDownloadSession userDownloadSession)

{

return \_sqlDataAccess.AddUserDownloadSession(userDownloadSession);

}

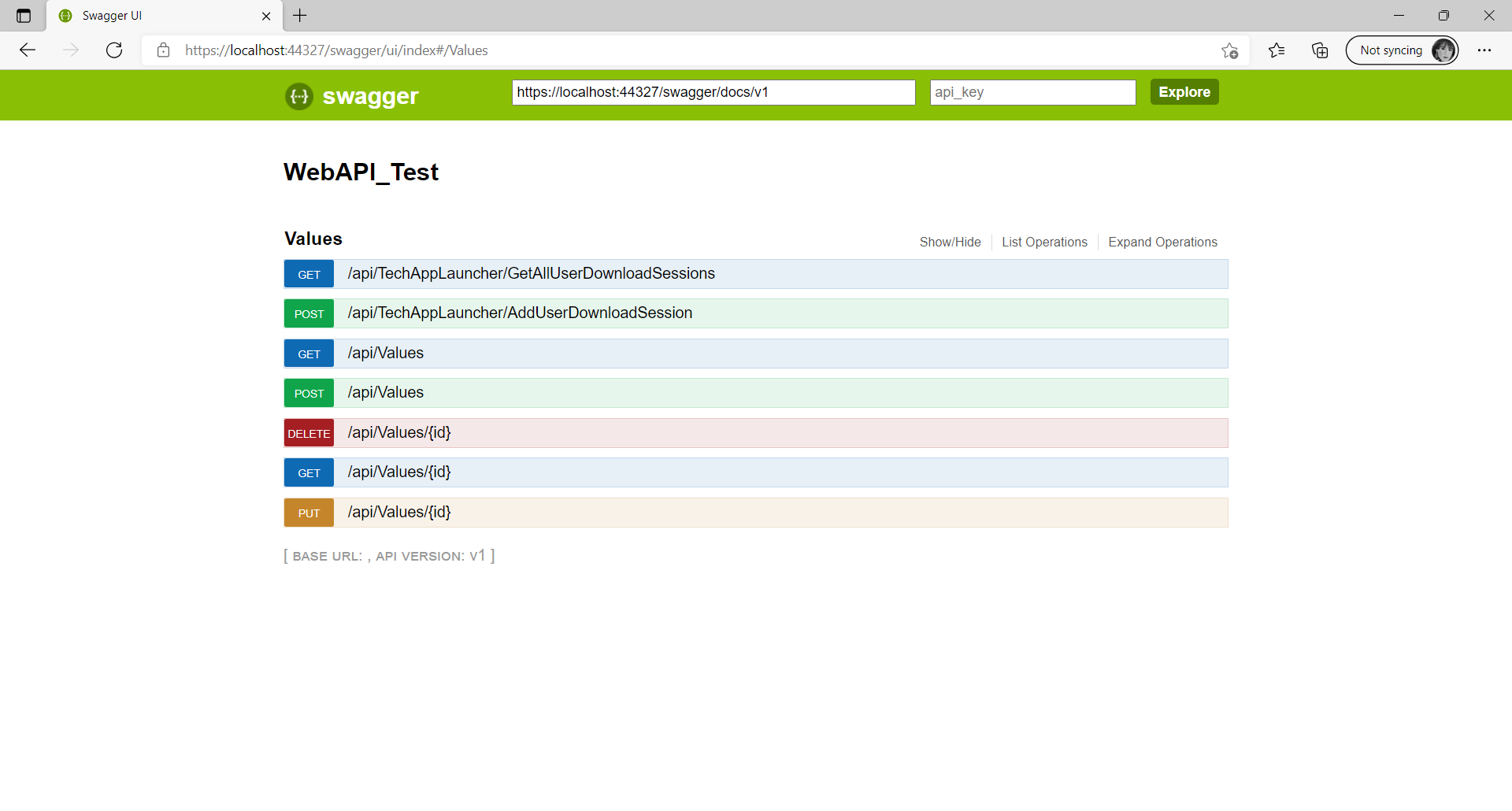
You may facing a number of errors after copy the code into the controller file. What you need to do is just add the 3 lines of code below to the top of the file.

using WebAPI\_Test.Models;

using WebAPI\_Test.Services;

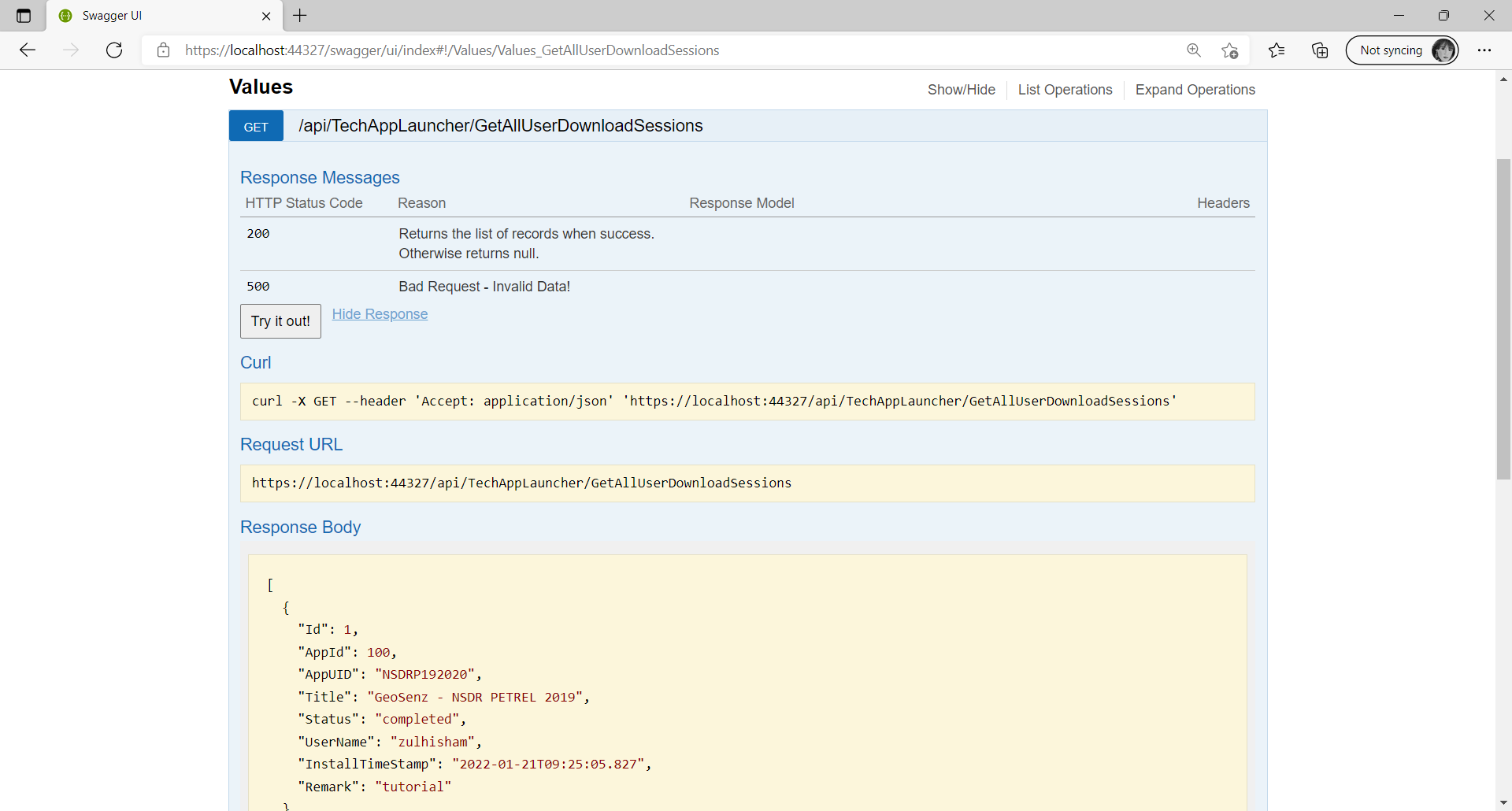
using Swashbuckle.Swagger.Annotations;

Now, you can start the Web Application. And then go to the swagger page.



28

You will notice that there are 2 newly added endpoints[28] as shown in the screen shot above. You can expend it and then click on the ‘**Try it out**’ button in order to test the ‘**/api/TechAppLauncher/GetAllUserDownloadSessions**’ API. It will returns a list of ‘**UserDownloadSession**’ in the response body[29].



29

To add a new record… Expend the ‘**/api/TechAppLauncher/AddUserDownloadSession**’ API, then add the following payload as a **parameter of** ‘**userDownloadSession**’.

{

"AppId": 100,

"AppUID": "NSDRP192020",

"Title": "GeoSenz - NSDR PETREL 2019",

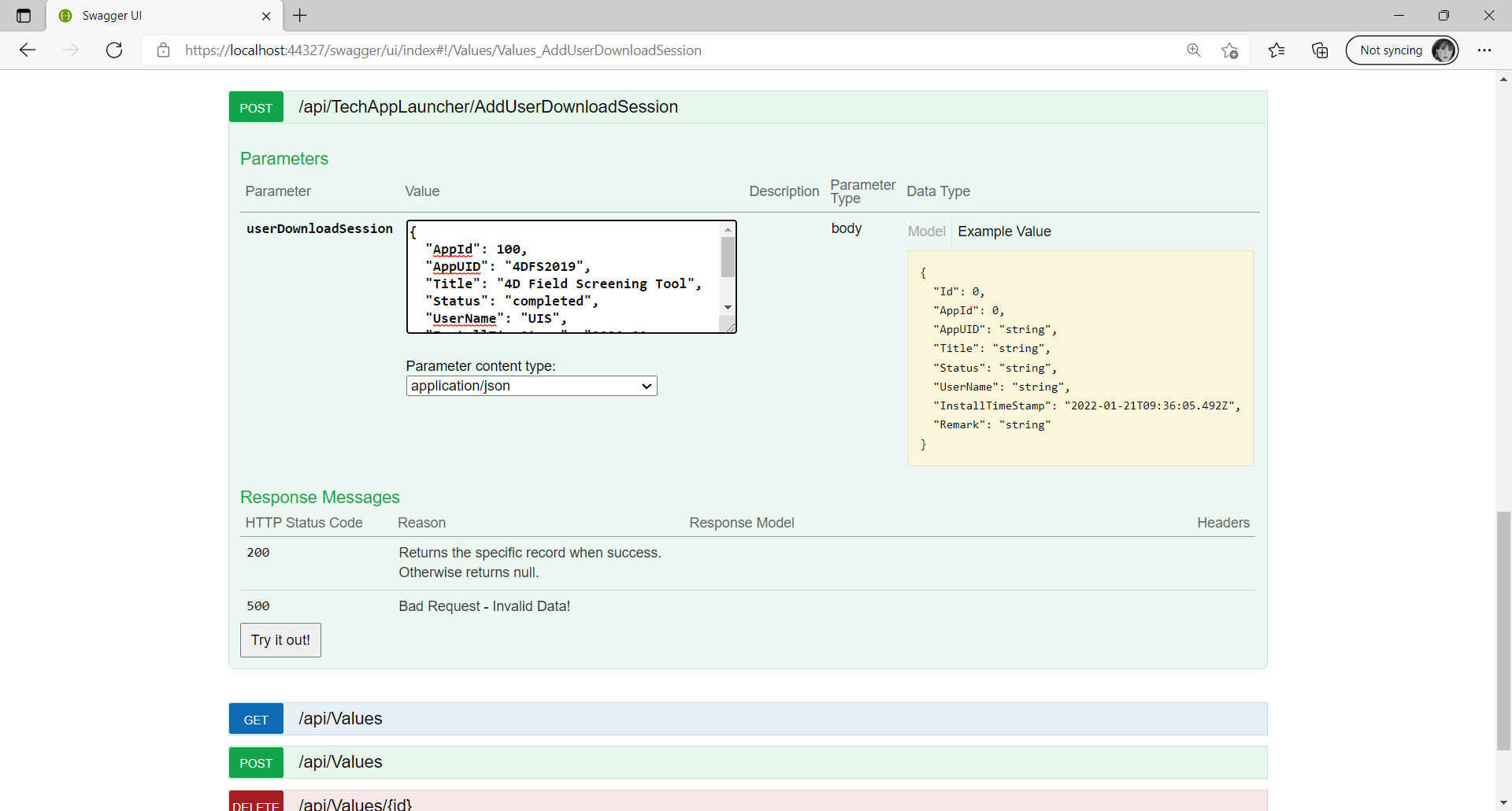
"Status": "completed",

"UserName": "zulhisham",

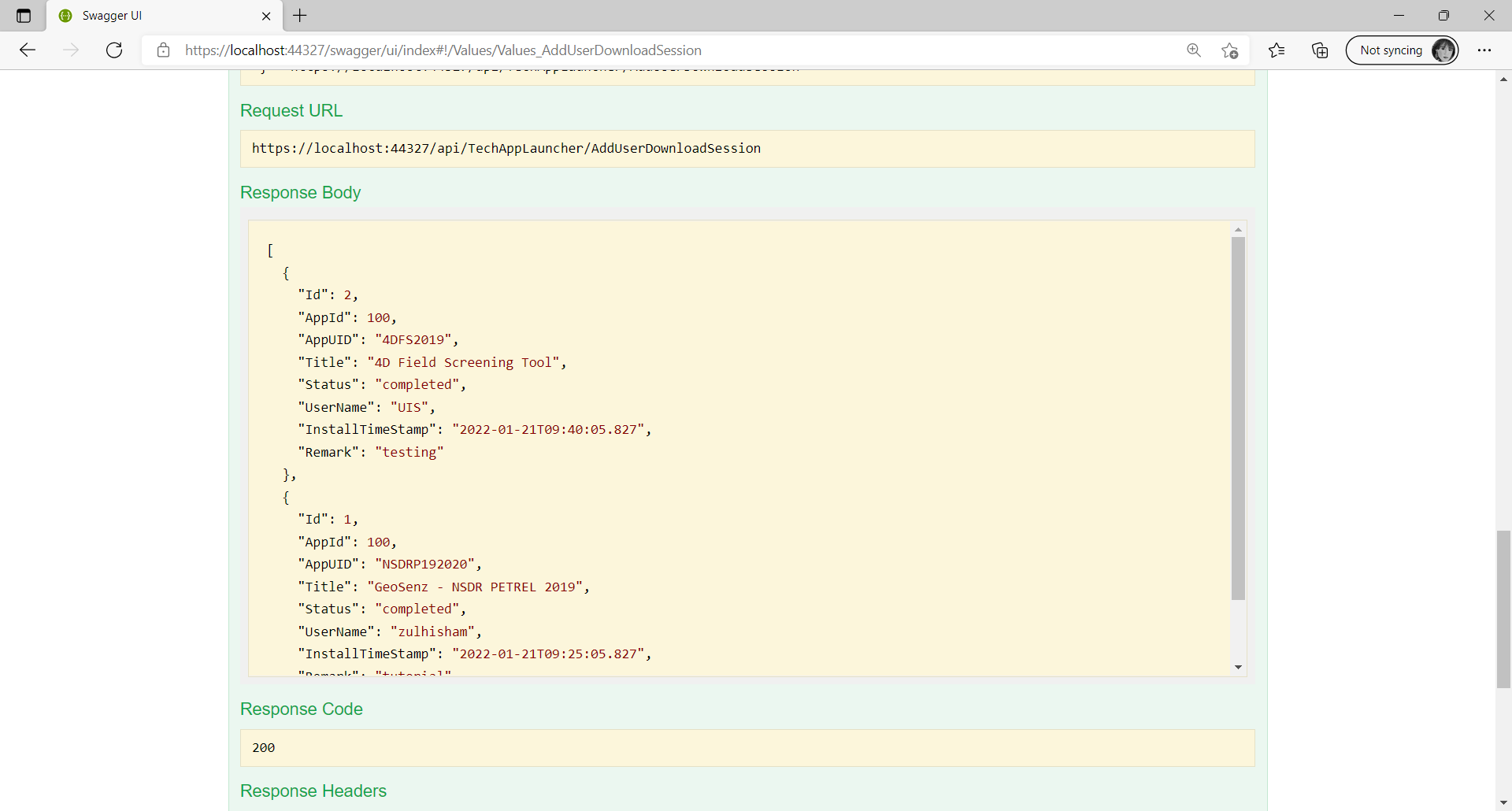
"InstallTimeStamp": "2022-01-21T09:25:05.828Z",

"Remark": "tutorial"

}



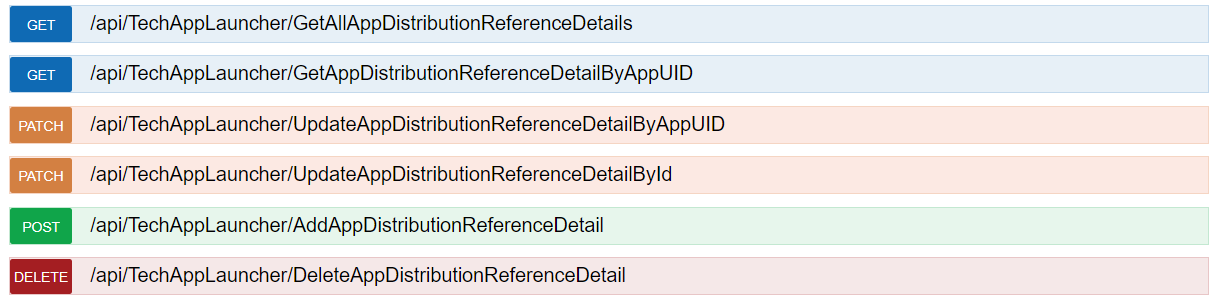
Then, click on the ‘**Try it out**’ button. You should be able to see that more and more records being added to the database as it is listed in the response body[30].



30

**DSG – AppDistributionReferenceDetail**

For DSG, it engages the same database of **Software Center**. Hence, it uses the following endpoints to manipulate the record in the database.



You should refer to the payload below as the parameter for these endpoints. The variable highlighted in ***bold italic*** is one that you need to change according to the plugin.

{

"AppUID": "***L1FDDSG2021***",

"LinkID": "-application org.eclipse.equinox.p2.director -noSplash -repository jar:file:/{pluginFilePath}!/ -installIUs {pluginUI}",

"AgentName": "dsg",

"Description": "***C:\\Users\\zulhisham\\eclipse\\committers-2021-033\\eclipse\\eclipsec.exe***"

}

* **AppUID** : is the ID you should retrieve from TechApp Store portal.
* **LinkID** : is the parameter that the eclipse command will be used.
* **AgentName** : is the name of the software to execute the command.
* **Description** : will be the executable file to execute the command.

**Best Practice**

It is recommended you shall follow some best practice, refer to the example by comparing the code below with the code above in ‘**ValuesController.cs**’ file. Line 99 to line 103 provides some summary to developer about what the API is all about. It won’t appear on the Swagger page. At line 108, instead of returns the data object directly, it is better to return a **HttpResponseMessage** object, so that the user of the API will have more clear information whenever they receive the response from the API. This is because we can create more meaning status code in the response message such as ‘**OK**’, ‘**Created**’, ‘**Accepted**’, ‘**Bad Gateway**’, ‘**Bad Request**’, ‘**Not Found**’ and etc… You can compare the code below at line 114 on how to create a response message with the code above which directly throw an exception.



By following this best practice, we have a choice to return a status code **400 – Bad Request for ‘Http POST’** method as well as return a status code **404 – Not Found for ‘Http GET’** method instead of **500 – Internal Server Error** for both of the requests which create confusion to the user of the API. So… Take it as challenge or exercise to change the code in ‘**ValuesController.cs**’ file by follow the best practice.

Well, I hope you’ve enjoy this short **ASP.NET WebAPI tutorial** | **Zulhisham Tan@2022**.