**HTTP (HyperText Transport Protocol)**

**HTTP (HyperText Transport Protocol)**

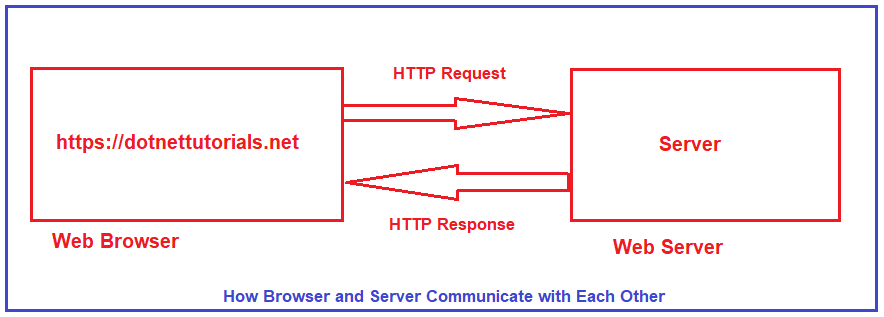
In this article, I am going to discuss everything you need to know about **HTTP (HyperText Transport Protocol)**i.e. HTTP Verbs or Methods, HTTP Status Codes, HTTP Requests, and Responses. As part of this article, we are going to discuss the following pointers in detail.

1. **How browser and Server Communicate with Each Other?**
2. **What is HTTP (HyperText Transport Protocol)?**
3. **Understanding HTTP Request and Response.**
4. **HTTP Verbs or HTTP Methods**
5. **HTTP Status Codes**

**How browser and Server Communicate with Each Other?**

HTTP stands for HyperText Transport Protocol. HTTP is used for communication between the client and server. Let us understand what does it mean by client and server with an example.

Suppose, you open the web browser, type the URL in the browser and press the enter button. As soon as you press the enter, a request is going to the server (called a web server). Whatever data you are sending from the web browser to the webserver is called a Request and whatever data we are receiving from the webserver is called a Response. This is how the browser and web server communicate with each other in the form of Request and Response. And this type of communication is only possible by using HTTP Protocol. So, the request can be termed as HTTP Request and the response can be called HTTP Response. For better understanding, please have a look at the following image.



Browser is not the only client. For example, if you are using a mobile application, then your mobile is a client. If you are calling APIs using tools like Postman and Fiddler, then Postman and Fiddlers are also the clients.

**What is HTTP?**

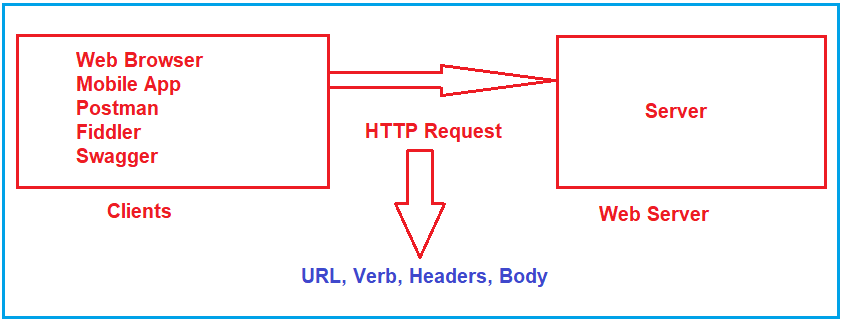
I have got the following two definitions of HTTP (HyperText Transport Protocol) from Wikipedia,

1. Hypertext Transfer Protocol (often abbreviated to HTTP) is a communications protocol. It is used to send and receive web pages and files on the internet. It was developed by Tim Berners-Lee and is now coordinated by the W3C. HTTP version 1.1 is the most commonly used version today. It is defined in RFC 2616.
2. The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

**HTTP Request Components:**

When we send something from the client (browser, mobile, postman, fiddler, etc.) to the server (webserver) is called a Request. The request is formed with a couple of components. They are as follows:

1. **URL**: Each Request must have a unique URL
2. **Verb (Method):** Each Request must have an HTTP Verb.
3. **Header(s):** Each Request can contain one or more Headers.
4. **Body**: Each request can have a body. The body contains the data that we want to send to the server.

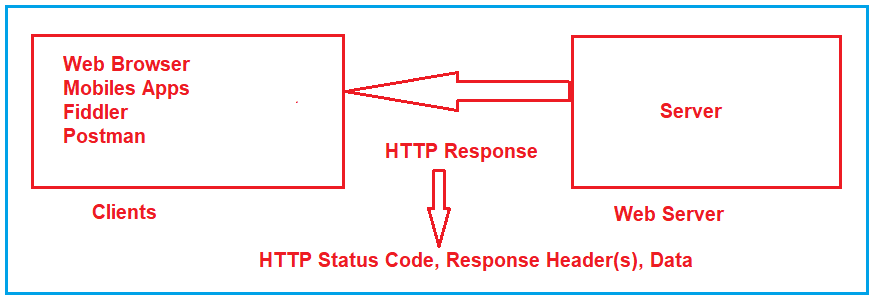


**HTTP Response Components:**

Whatever we get from the webserver to the clients called HTTP Response. The HTTP response contains the following components.

1. **HTTP Status Code:** It must have a Status Code.
2. **Response Headers:** It can have one or more response headers.
3. **Data**: Response can have data i.e. return to the client.

Along with the above three components, other components are also there. But the above three components are the important components in an HTTP Response.



**HTTP Verbs or HTTP Methods:**

We have already discussed that each HTTP Request must have an HTTP Verb. Let us understand what all HTTP Methods or Verbs are available.

**GET Method:**

The GET HTTP Method is used to Retrieve the Data. The HTTP GET method requests a representation of the specified resource. Requests using GET should only be used to request data (they shouldn’t include data). For example, you want to search something like you want to get the list of employees, list of products, you want to retrieve a book by id, etc. So, whenever you are expecting some data from the server, you need to use GET HTTP Verb.

In the case of the HTTP Get Method, the Web API may be expecting some data from the client, but the main purpose of the GET method is to fetch or retrieve data from the server. If you want to implement some kind of search functionality then the Web API may expect some data to filter out the results. In this case, the clients need to send the data.

**POST Method:**

The POST request is used to make a new entry in the database. It is not only specific to a database, whenever you want to create a new resource in your application, then you need to use the POST method.

What does it mean by adding a new Resource? It means if you want to add a new Employee, or you want to add a new product. So, here the main concept is not the database, the main concept is adding a new resource.

**PUT Method:**

PUT is also one of the HTTP Verb. The PUT method is used to update all the properties of the current resource in the database. What does it mean? For Example, we have a table called Product in our database. If we want to update all properties of a particular product (i.e. current resource) then we need to use PUT HTTP Request. So, whenever you want to update all the properties (column) of a resource (existing record in the database), then you need to use PUT Method. You cannot add a new resource using the PUT method.

**PATCH Method:**

There are some situations where you don’t want to update all the properties of an existing resource instead you want to update a few of the properties, then you need to use the PATCH method. So, the PATCH method is similar to the PUT method, but it is used to update few properties of the current resource in the database. For example, if you want to update few properties (columns) of an existing product then you need to use the PATCH method. That means if your Product table contains 10 columns, and you want to update only four columns of an existing product, then you need to use the PATCH method.

**DELETE Method:**

The DELETE method is used to delete the resource from the database. That means you are removing or deleting an existing entity from your database. In modern applications, we use two concepts for delete. One is Soft Delete and another one is Hard Delete.

Soft Delete: In your table, if you have some column like IsDeleted or IsActive, or something similar to this and you just want to update that column, then you cannot use Delete Method. In that case, you need to use the PATCH method. This is because you are not deleting the record from the database, you just update the record.

Hard Delete: If you want to remove the existing entity from the table, then you need to use the DELETE method. For example, Delete an existing product from the Product table in the database, etc.

There are other HTTP Methods available, but these are the most used HTTP Methods, we generally used in our applications.

**HTTP Status Codes:**

The HyperText Transport Protocol status code is also one of the important components of HTTP Response. The Status code is issued from the server and they give information about the response. Whenever we get any response from the server to the client, in that HTTP Response, we must have one HTTP Status code. All the HTTP Status codes are divided into five categories. They are as follows. Here, XX will represent the actual number.

1. **1XX**: Informational Response (Ex: 100, 101, 102, etc.)
2. **2XX**: Successful, whenever you get 2XX as the response code, it means the request is successful. (Ex. 200, 201, 203, etc.)
3. **3XX**: Redirection, whenever you get 3XX as the response code, it means it is re-directional i.e. some re-directional is happening on the server. (Ex. 300, 301, etc.)
4. **4XX**: Client Error, whenever you get 4XX as the response code, it means there is some problem with your request. (Ex: 400, 404, etc.)
5. **5XX**: Server Error. Whenever you get 5XX as the response code, it means there is some problem in the server. (Ex: 500, etc.).

**Frequently used HTTP Status Codes:**

The following are some of the frequently used Status codes.

1. **100: 100 means Continue.** The HTTP 100 Continue informational status response code indicates that everything so far is OK and that the client should continue with the request or ignore it if it is already finished.
2. **200: 200 means OK.** The HTTP 200 OK success status response code indicates that the request has succeeded. If you are searching for some data and you got the data properly. That means the request is successful and in that case, you will get 200 OK as the HTTP status code.
3. **201: 201 means a new resource created.** The HTTP 201 Created success status response code indicates that the request has succeeded and has led to the creation of a resource. The new resource is effectively created before this response is sent back and the new resource is returned in the body of the message, its location being either the URL of the request or the content of the Location header. If you are adding successfully a new resource by using the HTTP Post method, then in that case you will get 201 as the Status code.
4. **204: 204 means No Content.** The HTTP 204 No Content success status response code indicates that a request has succeeded, but that the client doesn’t need to navigate away from its current page. If the server processed the request successfully and it is not returning any content, then in that case you will get a 204 response status code.
5. **301: 301 means Moved Permanently.** If you are getting 301 as a status code from the server, it means the resource you are looking for is moved permanently to the URL given by the Location headers.
6. **302: 302 means Found.** If you are getting 302 as a status code from the server, it means the resource you are looking for is moved temporarily to the URL given by the Location headers.
7. **400: 400 means Bad Request.** If you are getting 400 as the status code from the server, then the issue is with the client request. If the request contains some wrong data such as malformed request syntax, invalid request message framing, or deceptive request routing, then we will get this 400 Bad Request status code.
8. **401: 401 means Unauthorized.** If you are trying to access the resource for which you don’t have access (Invalid authentication credentials), then you will get a 401 unauthorized status code from the server.
9. **404: 404 means Not Found**. If you are looking for a resource that does not exist, then you will get this 404 Not Found status code from the server. Links that lead to a 404 page are often called broken or dead links.
10. **405: 405 means Method Not Allowed.** The 405 Method Not Allowed response status code indicates that the request method is known by the server but is not supported by the target resource. For example, we have one method which is a POST method in the server and we trying to access that method from the client using GET Verb, then, in that case, you will get a 405 status code.
11. **500: 500 means Internal Server Error**. If there is some error in the server, then you will get a 500 Internal Server Error status code.
12. **503: 503 means Service Unavailable.** The 503 Service Unavailable server error response code indicates that the server is not ready to handle the request. If the server is down for maintenance or the server is overloaded then in that case, you will get the 503 Service Unavailable Status code.
13. **504: 504 means Gateway Timeout.** The 504 Gateway Timeout server error response code indicates that the server while acting as a gateway or proxy, did not get a response in time from the upstream server that is needed in order to complete the request.

**Environment Setup for ASP.NET Core Web API Development**

**Environment Setup for ASP.NET Core Web API Development**

In this article, I am going to discuss the **Environment Setup required for developing ASP.NET Core Web API** Applications. Please read our previous article where we discussed [**HTTP Protocols**](https://dotnettutorials.net/lesson/hypertext-transport-protocol/). Tools and Software Requires for the development of ASP.NET Core Web API Applications.

1. **Operating System**: (Windows, Mac, Linux)
2. **Editor**: Visual Studio 2019 (latest version), Visual Studio Code.
3. **Dot Net Core SDK 5** (Latest Version as of this writing): This is the software development KIT and this KIT is helpful for the development and running of the application in the system.
4. **Database**: SQL Server 2019
5. **SSMS** (SQL Server Management Studio): To interact with SQL Server database
6. **API Client:** Postman, Fiddler. These are client tools and are used to test the Web APIs

**Operating System:**

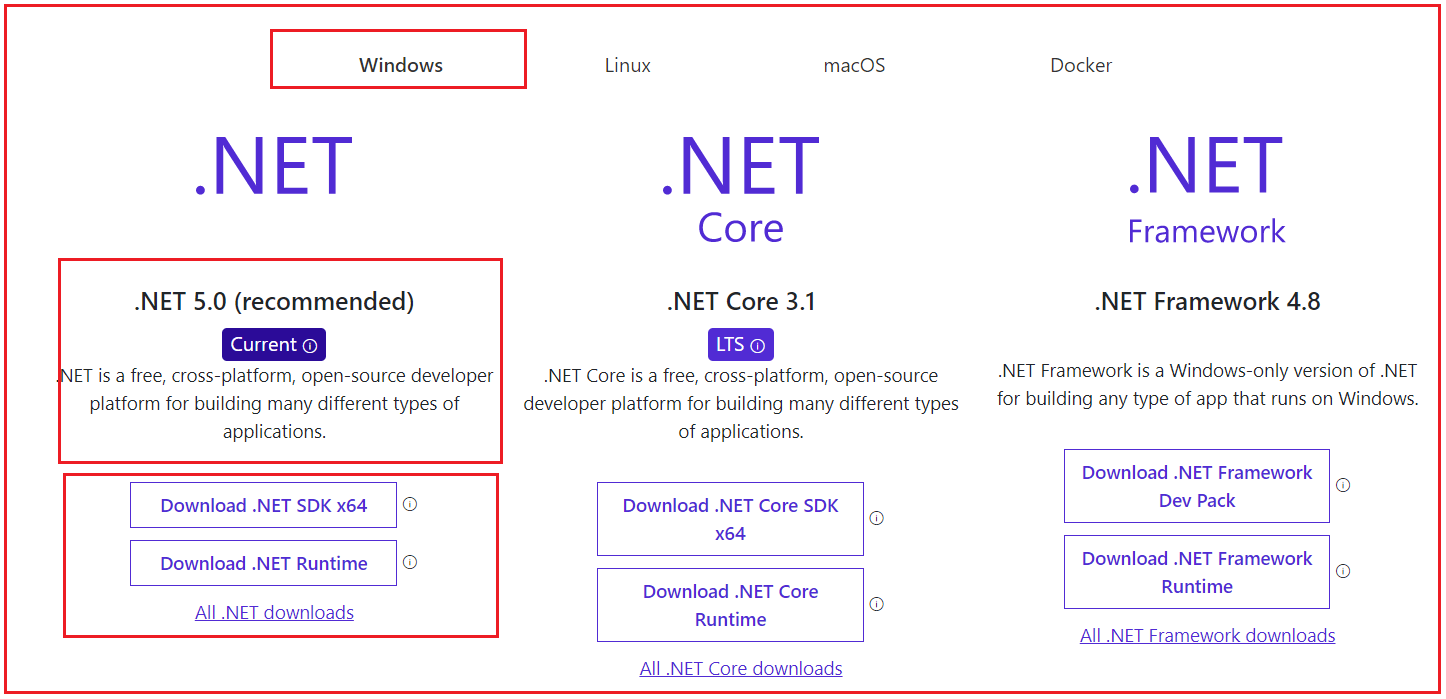
You are free to use any operating system as per your choice. .NET Core is an open-source and cross-platform framework. So, you can use any type of Operating System such as Windows, Mac, and Linus for ASP.NET Core Web API Application development. I am going to use Windows Operating System.

**Download and install .NET Core SDK 5 (Latest Version)**

Once you decide Operating System, next you need to install .NET Core SDK on that machine. In order to download .NET Core SDK, please go to the below Website.

[**https://dotnet.microsoft.com/download**](https://dotnet.microsoft.com/download)

Once you go to the above website, you will find the following webpage. As per your operating system choose the appropriate .NET Core SDK. The latest version as of this writing is .NET 5.0 and this is the recommended one as shown in the below image.

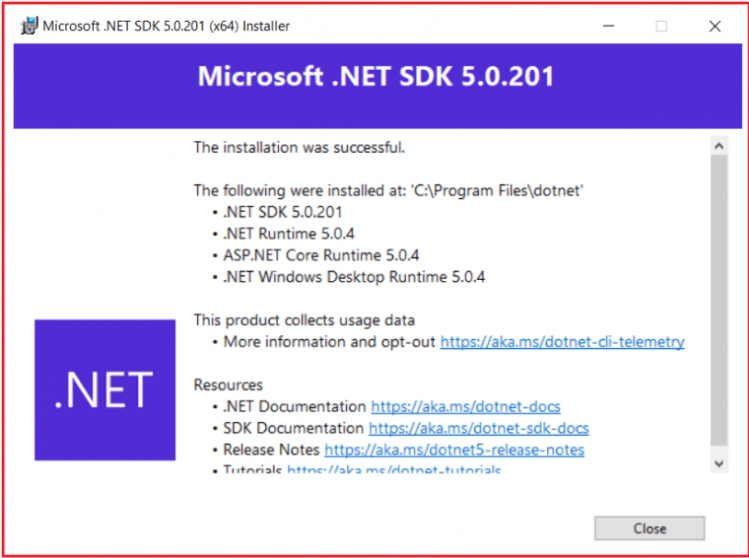


**Further, in .NET 5, there are three options to download the .NET Core SDK which are as follows:**

1. **.NET Runtime:** The .NET Runtime is required only to run .NET Core applications. The .NET Runtime just contains the resources or libraries that are required to run existing .NET Core applications.
2. **.NET SDK:** If you want to develop, build, run, test, and deploy your .NET Core Applications, then you need to download the **.**NET SDK (Software Development Kit). The .NET SDK also contains.NET Runtime. So, if you installed the .NET SDK, then there is no need to install .NET Runtime separately.
3. **All .NET downloads:** If you want an older version of .NET (like .NET Core 3.1), then you need to click on this link and it will navigate to another page, from where you can download the .NET core Framework version as per your choice.

As we are here to develop, build, run and deploy .NET Core Applications, so we need to click on the **Download .NET SDK x64** link. Once you click on the **Download .NET SDK x64**button it will download the .NET Core SDK.

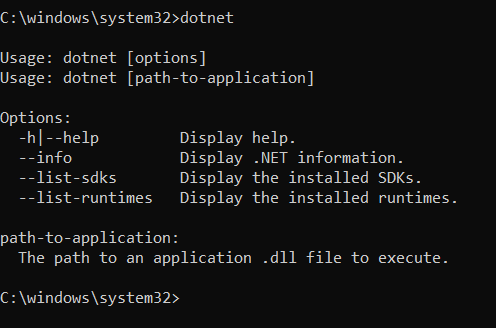
Once you download the .NET SDK, then install it by following the required steps. If everything fine, once the installation completed, you will get the following.



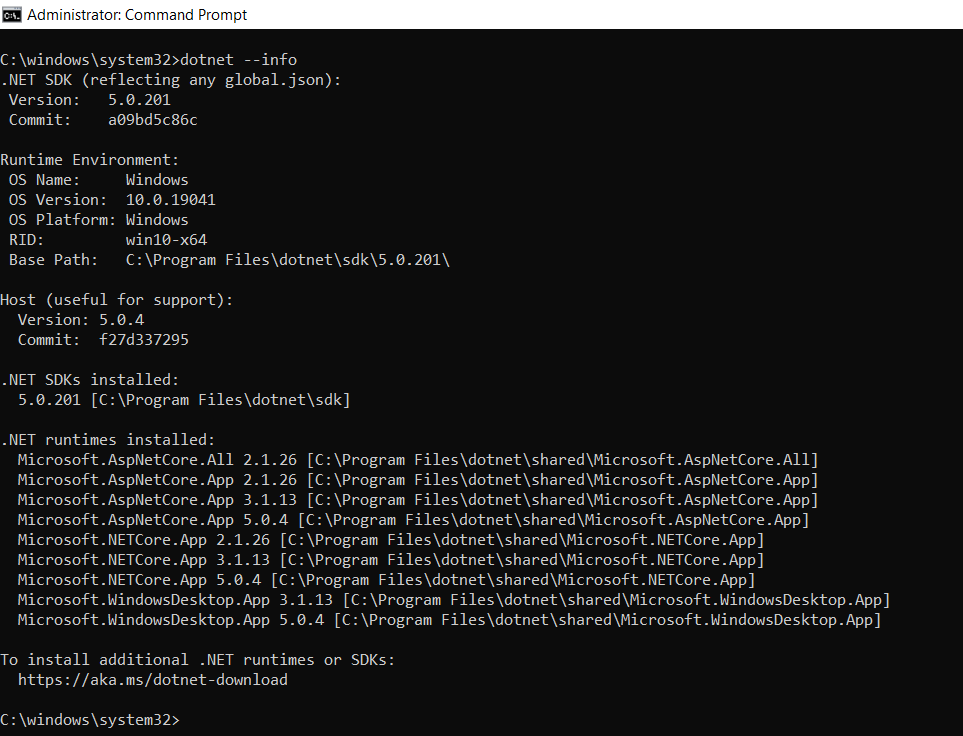
**Note:** If you installed .NET Core SDK on your machine, then there is no need to install .NET Runtime separately as .NET Core SDK automatically the .NET Runtime. The .NET Runtime is a part of .NET SDK. Another point that you need to keep in mind is that, once you installed .NET Core SDK, then automatically ASP.NET Core gets installed.

**Verifying the Installation:**

Once you’ve installed .NET SDK, open the command prompt and type **dotnet** and press the enter key as shown in the below image. If the installation succeeded, then you should get the following output.



Now in order to verify what versions installed on your machine, type the **dotnet –info** command in the command prompt and press enter as shown in the below image.



As you can see in the above image, in my machine, .NET SDK version 5.0.201 is installed. Along the way, it also displays the OS Name, Version, etc. Further, if you notice it also installed .NET Runtimes.

**Note:** The .NET Core SDK will give a complete environment to develop, build, test and deploy the ASP.NET Core Web API Applications.

**Integrated Development Environment (IDE) for ASP.NET Core Web API Development:**

You can use any of the following IDEs to develop the ASP.NET Core Web API Applications.

1. **Visual Studio 2019**
2. **Visual Studio Code**
3. **.Net Core CLI**

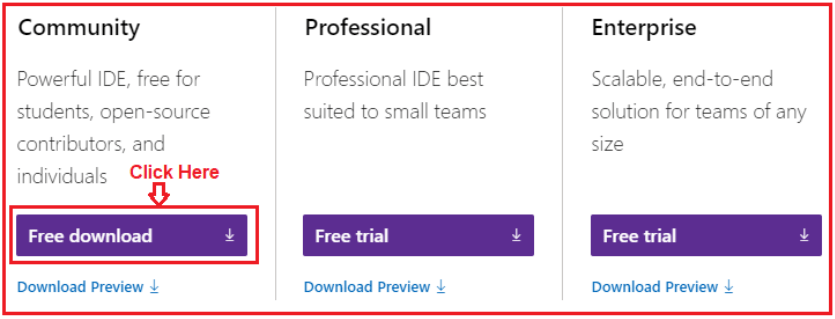
We will discuss using all the above options to develop the ASP.NET Core Web API Applications. But from the beginning, we are going to use Visual Studio 2019 as the IDE for ASP.NET Core Web API development.

**Download Visual Studio 2019:**

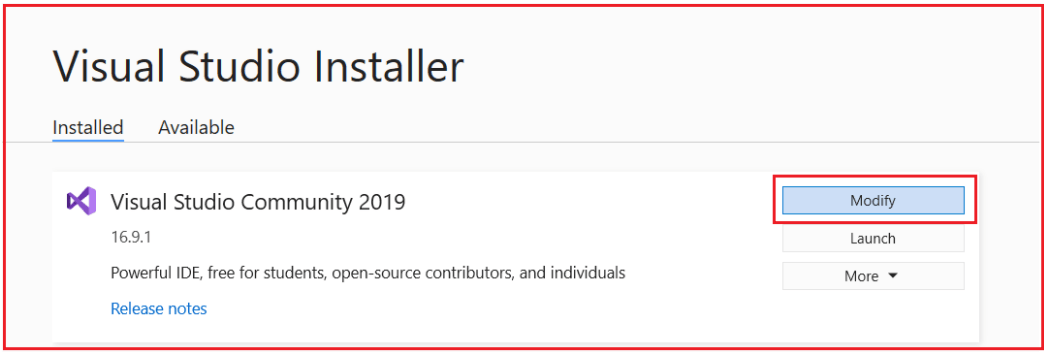
If you have not installed Visual Studio 2019, then please download and installed Visual From the below URL.

[**https://visualstudio.microsoft.com/downloads/**](https://visualstudio.microsoft.com/downloads/)

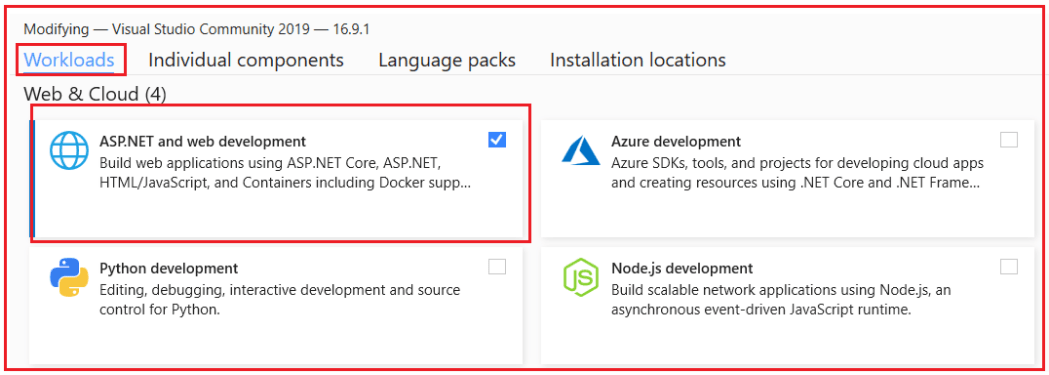
Once you navigate to the above URL, and if you are a student and you just want to learn ASP.NET Core Web API, then I recommended you to choose the Community edition of Visual Studio 2019 which is free as shown in the below image and the community version has all the features of the enterprise edition has.



While installing Visual Studio 2019, please make sure ASP.NET and Web development workload is installed. To verify this, whether you have installed this workload or not, Open Visual Studio Installer and then click on the Modify button as shown in the below image.



Once you click on the Modify button, the following screen window will open. Here select the Workloads option and then make sure the ASP.NET and Web Development checkbox to be checked as shown in the below image.

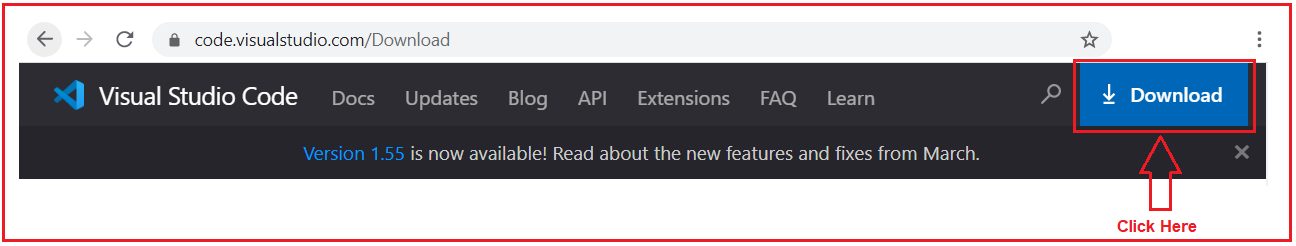


**How to Download and Install Visual Studio Code**

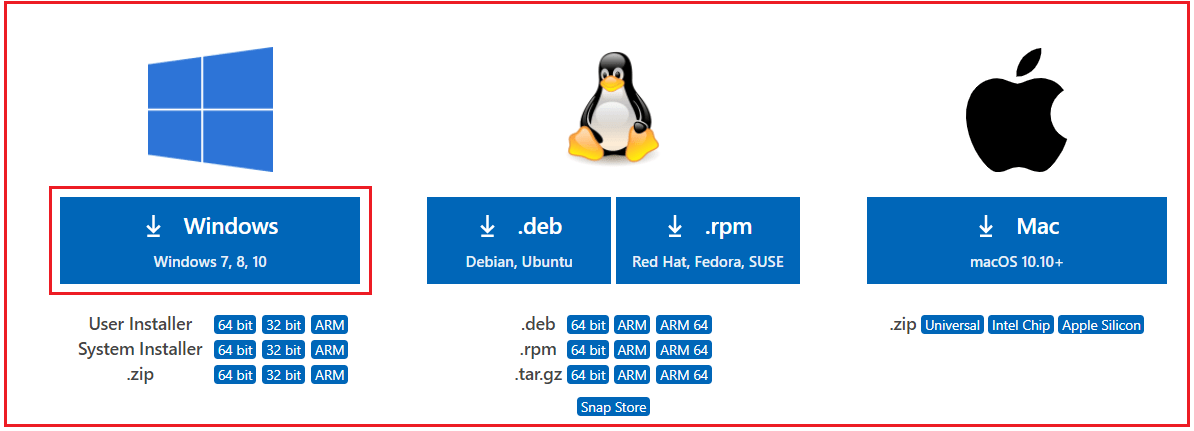
Let us see how to download and install Visual Studio Code. In order to download Visual Studio Code, please click on the following link.

[**https://code.visualstudio.com/**](https://code.visualstudio.com/)

Once you click on the above link, it will open the below website, from this webpage just click on the Download button which you can see at the top of the website as shown below.



Once you click on the Download button, it will open the following page. Based on your operating system choose the appropriate visual studio. I have windows 10 installed on my machine, so I installed the Windows option as shown in the below image.



Once the download completed, click on the setup file follow the on-screen steps to install Visual Studio on your machine.

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

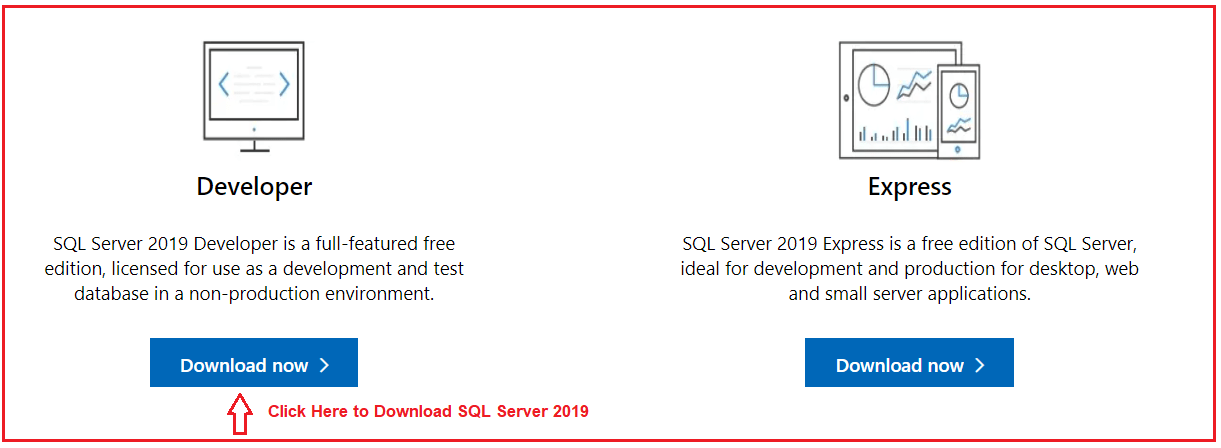
**Note:** Visual Studio and Visual Studio Code are editors. They will help us to write the code quickly. But you are free to use any editor as per your choice such as Notepad ++, Atom, Sublime, etc.

**Install SQL Server 2019:**

Next, we are going to install SQL Server 2019. Remember we are going to develop Web APIs and to work with Wen APIs, we need some data. You can hard code the data in memory. But in Real-time, we need to work with a database. As a database, in this tutorial, we are going to use SQL Server. If you have not installed SQL server 2019 on your machine, then please download and installed it from the below URL.

[**https://www.microsoft.com/en-ie/sql-server/sql-server-downloads**](https://www.microsoft.com/en-ie/sql-server/sql-server-downloads)

Once you click on the above link, it will open the below page. Here we have many options to use SQL Server database. Here, click on the Download now button which is below the Developer tab as shown in the below image.



SQL Server 2019 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment. Once you download, then install it by following the on-screen steps.

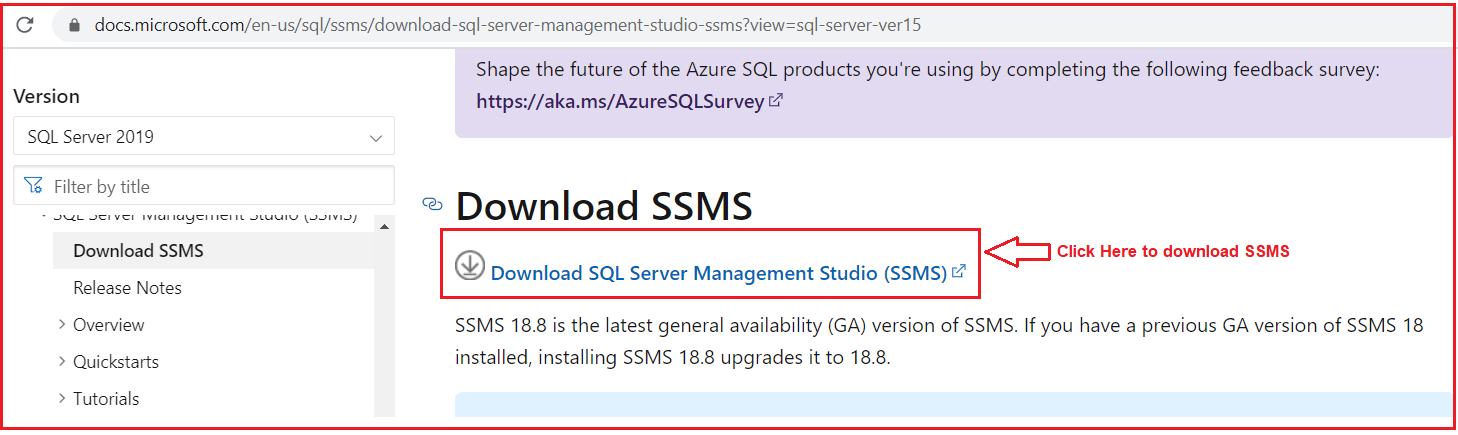
SQL Server is a DBMS and provides the complete environment to save and work with the data of an application. SQL Server stores the data in form of Tables. We can create relationships between the tables. You can use SQL Server from any other server from your local machine.

**Install SQL Server Management Studio:**

Once you have installed the SQL Server database, then you need an interface to interact with the SQL Server Database. And for that, we are going to use SQL Server Management Studio (SSMS) to interact with the SQL Server database. If you have not installed SSMS on your machine, then please download and installed the SQL Server Management Studio from the below link.

[**https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms**](https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms)

Once you go to the above link, it will open the below webpage and simply click on the Download SQL Server Management Studio (SSMS) link as shown in the below image.



Once you download SSMS, then install it. SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to Azure SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL Server and databases. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, and build queries and scripts.

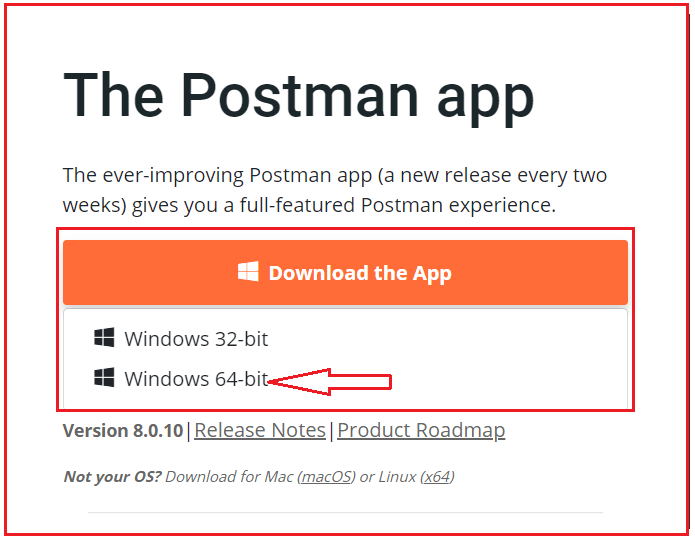
Use SSMS to query, design, and manage your databases and data warehouses, wherever they are – on your local computer, or in the cloud.

**Install Postman:**

As we are going to work with the Web API, we also need one Web API client tool. And for the client tool, we are going to use Postman. In order to install Postman, click on the below link.

[**https://www.postman.com/downloads/**](https://www.postman.com/downloads/)

Once you click on the above link, it will open the below webpage. From the below page, click on the Download the App link to download postman on your machine. Depending upon your operating system, you can choose either 32-bit or 64-bit. I have installed a 64-bit operating system and so, I choose Windows 64 bit as shown in the below image.



Once you download, then install it. A postman is an API tool. Postman provides a complete and nice interface to work with APIs.

That’s it. We are ready with our environment to develop ASP.NET Core Web API Applications. In the next article, I am going to discuss [**How to Create, Build, Run, and Test ASP.NET Core Web API Application using .NET Core CLI**](https://dotnettutorials.net/lesson/creating-asp-net-core-web-api-project-using-net-core-cli/). Here, in this article, I explain the **Environment Setup for ASP.NET Core Web API Application Development** and I hope you enjoy this ASP.NET Core Web API Environment setup for Development article.

**Creating ASP.NET Core Web API Project using .NET Core CLI**

**Creating ASP.NET Core Web API project using .NET Core CLI**

In this article, I am going to discuss **how to create a Web API project using ASP.NET Core 5**. We will also discuss how to build and run that project. And finally, we will see, how to test the API which is by default provided by that project using Swagger, Browser, and Postman. Please read our previous article where we discussed the [**Environment Setup for ASP.NET Core Web API**](https://dotnettutorials.net/lesson/environment-setup-asp-net-core-web-api/) Application development.

**Creating ASP.NET Core Web API Project**

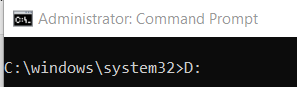
In .NET Core, there are two ways to create a project. They are as follows:

1. **Using .NET Core CLI (Command Line Interface)**
2. **Using Visual Studio 2019**

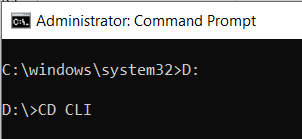
**Note**: In this article, I will show you how to create an ASP.NET Core Web API project using .NET Core CLI (Command Line Interface) and in the next article, I will show you how to create an ASP.NET Core Web API project using Visual Studio 2019.

**ASP.NET Core Web API project using .NET Core CLI**

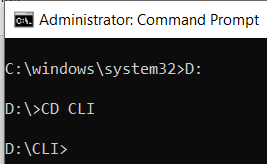
First, we need to select the folder location where we need to create the Web API Project. Suppose we want to create the ASP.NET Core Web API Project in the **D:\CLI** folder. Then open the command prompt in Administrator mode and change the directory location to D drive in the command prompt by typing **D:** and then press the enter key as shown below.



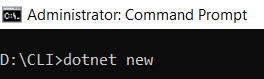
Once you type **D:** and press enter button, then it will change the directory path to the D drive. Then decide inside which folder you to create the project. I am going to create the project inside the CLI folder. So, change the directory location to the CLI folder by typing **CD CLI** and press the enter as shown below.



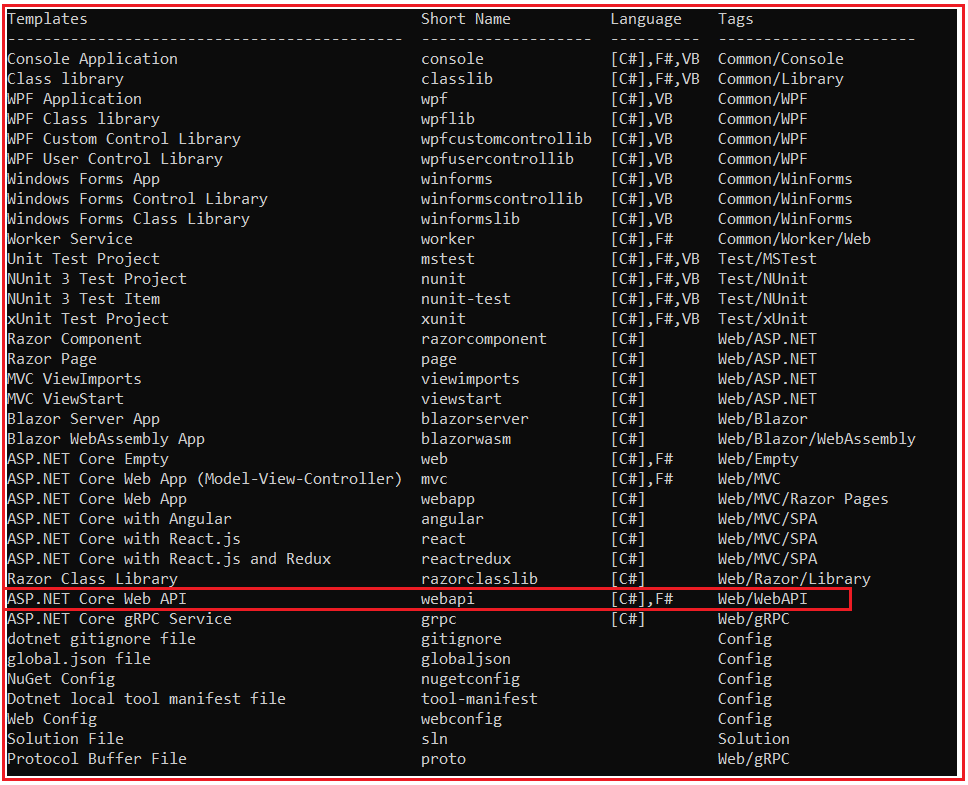
Once you type **CD CLI**and press the enter key, it will change the directory path to the CLI folder as shown in the below image.



Now let see the command to create a new project using .NET Core CLI. First, type **dotnet new** and press the enter button as shown in the below image.



Once you type **dotnet new** and press the enter button, it will display the following. As you can see in the below image, we have the templates and the short name for the template as well as the default language for the template.

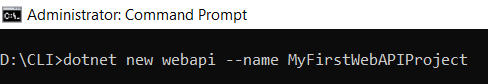


As you can see in the above image, it provides one template called ASP.NET Core Web API, and to create that project we can use the command webapi (Short Name of the template). The default programming language for this Web API Project is C# and apart from C# language, you can also use F# language.

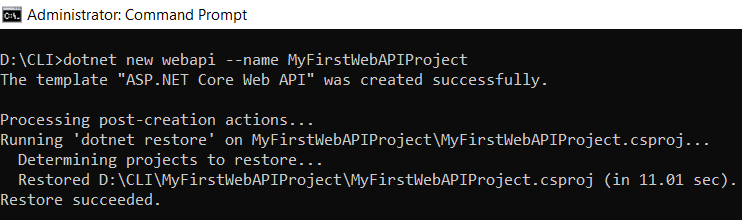
Let us create the web API project. You can create a web API project using two ways. They are as follows:

1. **dotnet new webapi:** If you only type **dotnet new webapi**, then a new project will be created inside the CLI folder with the default name.
2. **dotnet new webapi –name MyFirstWebAPIProject**: With this command, the .NET Core CLI will create a project inside the CLI folder with the name MyFirstWebAPIProject.

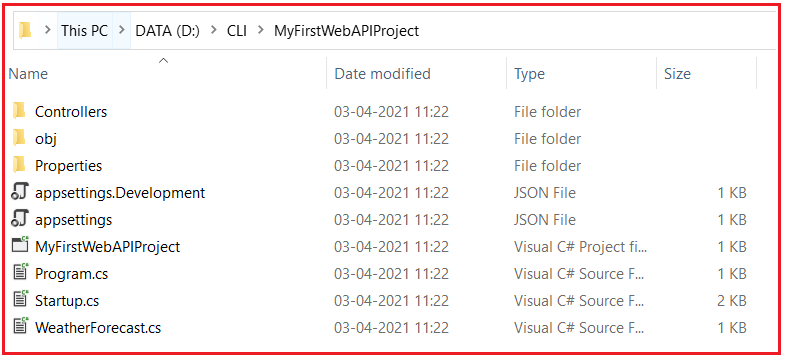
So, let us use the second option to create the ASP.NET Core Web API Project. Type **dotnet new webapi –name MyFirstWebAPIProject** command and press enter in the command prompt as shown in the below image.



Once you type dotnet new webapi –name MyFirstWebAPIProject and press enter, you will get the below output saying that the template ASP.NET Core Web API was created successfully.

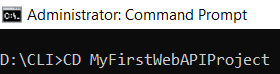


Now, inside the CLI Folder, you can see a folder with your project name (in my case the folder name is **MyFirstWebAPIProject**) and inside that folder, you can see the following default project files and folders provided by the ASP.NET Core Web API template as shown in the below image.

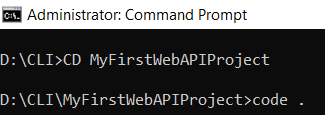


**Opening ASP.NET Core Web API project in Visual Studio Code:**

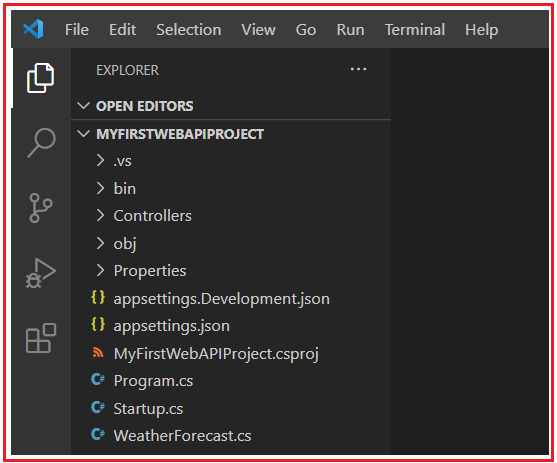
Now let us see how to open the above project in Visual Studio Code. To do so, first change the directory in the command prompt to the project directory (CD MyFirstWebAPIProject) as shown below.



Once you type CD MyFirstWebAPIProject and press the enter button, it will change the directory. Then to open the project files and folders in Visual Studio Code, type “**code .**” (code space dot) and press the enter button as shown in the below image.



Once you type “**code .**” (code space dot) and press the enter button, it will open the project in Visual Studio code as shown in the below image.



In our upcoming articles, we will discuss each of the above files and folders in detail.

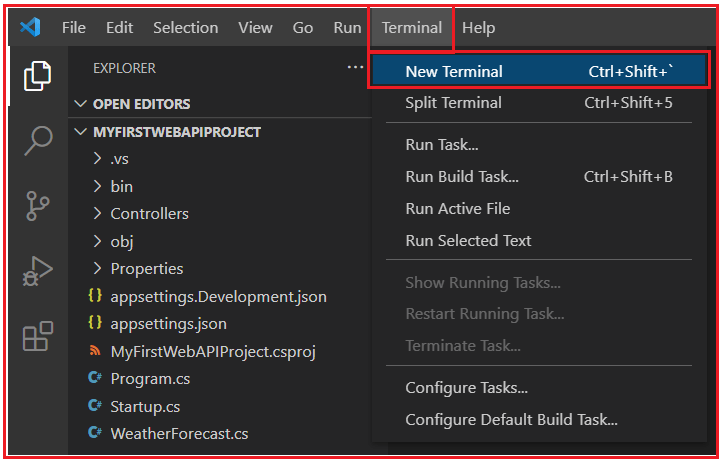
**How to build an ASP.NET Core Web API project using .NET Core CLI?**

Now let us see how to build an ASP.NET Core Web API project using .NET Core CLI. You can build the project using two ways.

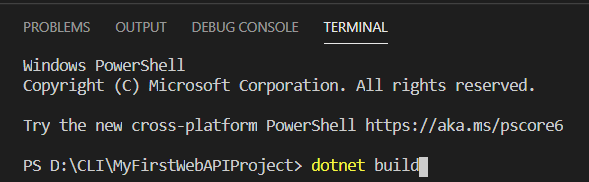
1. **Using Visual Studio Code Terminal**
2. **Using Command Prompt**

**Build ASP.NET Core Web API Project Using Visual Studio Code Terminal**

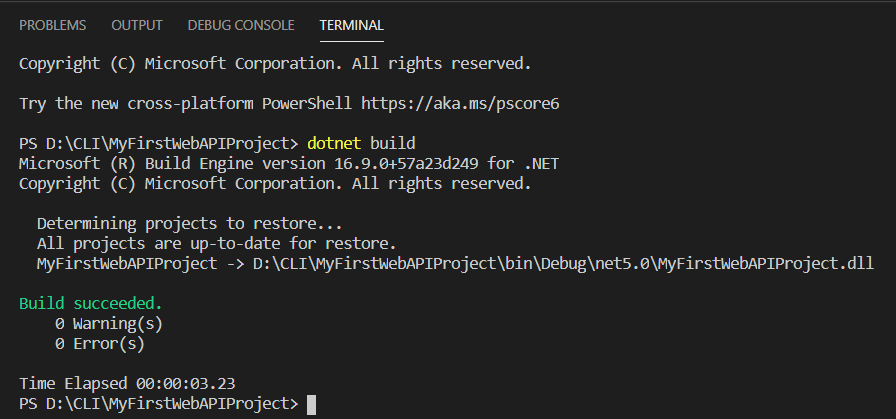
First, open a new terminal. To do so, click on the **Terminal** Menu and then select the **New Terminal** option as shown in the below image.



Once you click on the New Terminal option, it will open the terminal window as shown in the below image. In the terminal, type **dotnet build and press the enter button** as shown in the below image. This command is used to build an existing project.



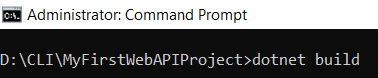
Once you type dotnet build and press the enter button, it will build the project and you get the following message.



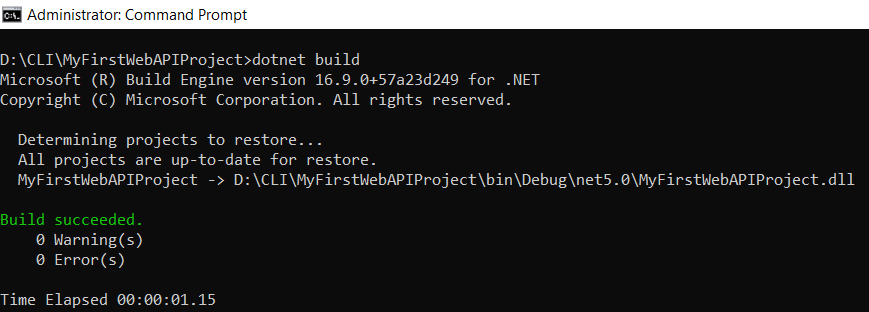
As you can see in the above image, the build is successful and there is no error and no warning.

**Build ASP.Net Core Web API Project Using Command Prompt:**

In the command prompt, first set the directory to your project folder and then type dotnet build and press the enter button as shown in the below image.

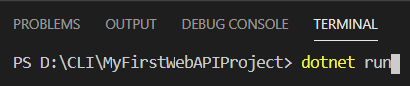


Once you type dotnet build and press the enter button, it will build your project and you will get the below message.

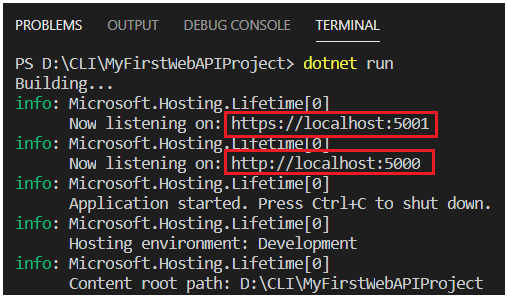


**How to run the ASP.NET Core Web API project using .NET Core CLI?**

Now let us see how to run the above ASP.NET Core Web API Project. The .NET Core CLI provides the run command to run the ASP.NET Core Web API Application. So, in the terminal type dotnet run and press the enter button as shown in the below image.

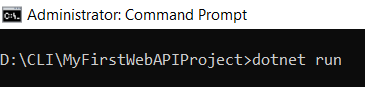


Once you type dotnet run and press the enter button, you will get the below message.

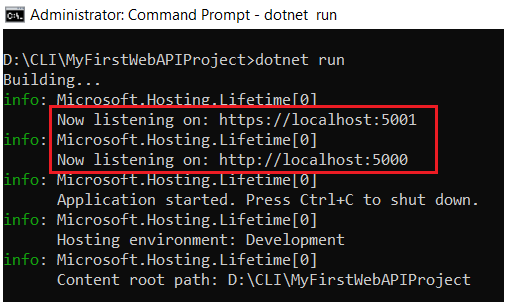


**How to run ASP.NET Core Web API Project using Command Prompt?**

In the command prompt, first set the directory to your project folder and then type **dotnet run** command and press the enter button as shown in the below image.



Once you type dotnet run and press the enter button, it will run your project and you will get the below message.

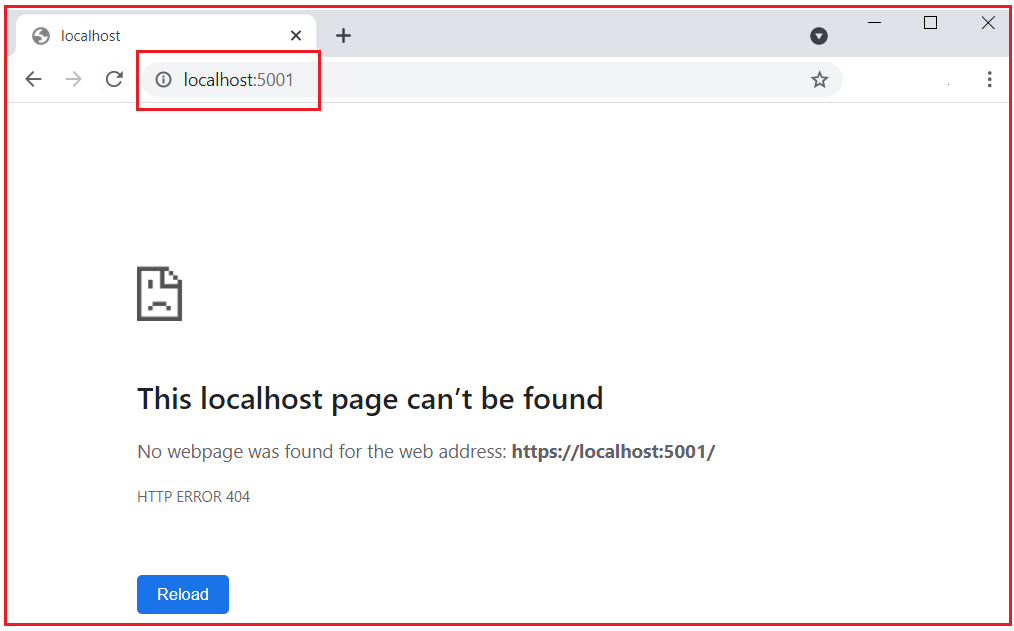


As you can see in the above image, our ASP.NET Core Web API Application is running on two different ports. They are as follows:

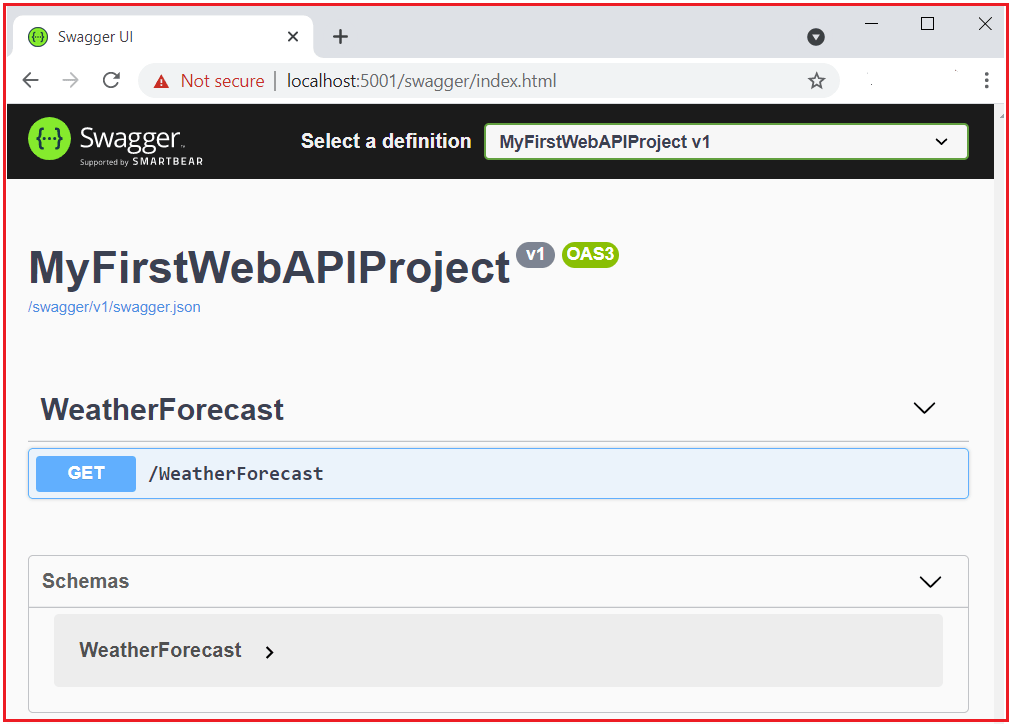
**https://localhost:5001**

**http://localhost:5000**

Now, open the above URL in any of your browsers and you will get a 404 error.



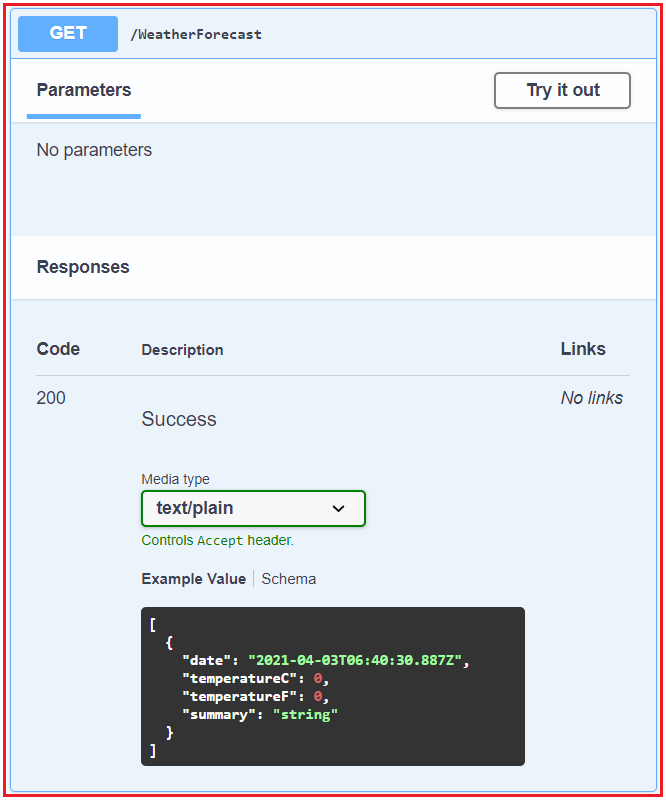
Don’t worry. Just type **swagger** at the end of the URL and press enter and you will get the following webpage.



The swagger will display the details of all the Web APIs available in your project. As you can see in the above image, it showing one API i.e. /WeatherForecast and the type is Get. Now click on the /WeatherForecast API to see details as shown in the below image.



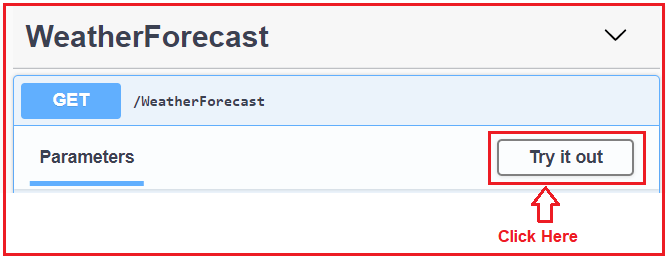
Once you click on the /WeatherForecast API, then it will show you the details of this API as shown in the below image.



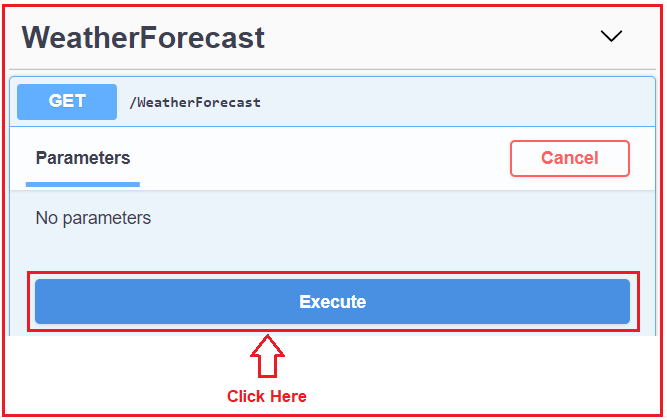
Note: Swagger is also a Client API Tool and using swagger we can also test the Web APIs. If you are using the default ASP.NET Core Web API project, then by default swagger is installed into the project.

**Test ASP.NET Core Web API using Swagger:**

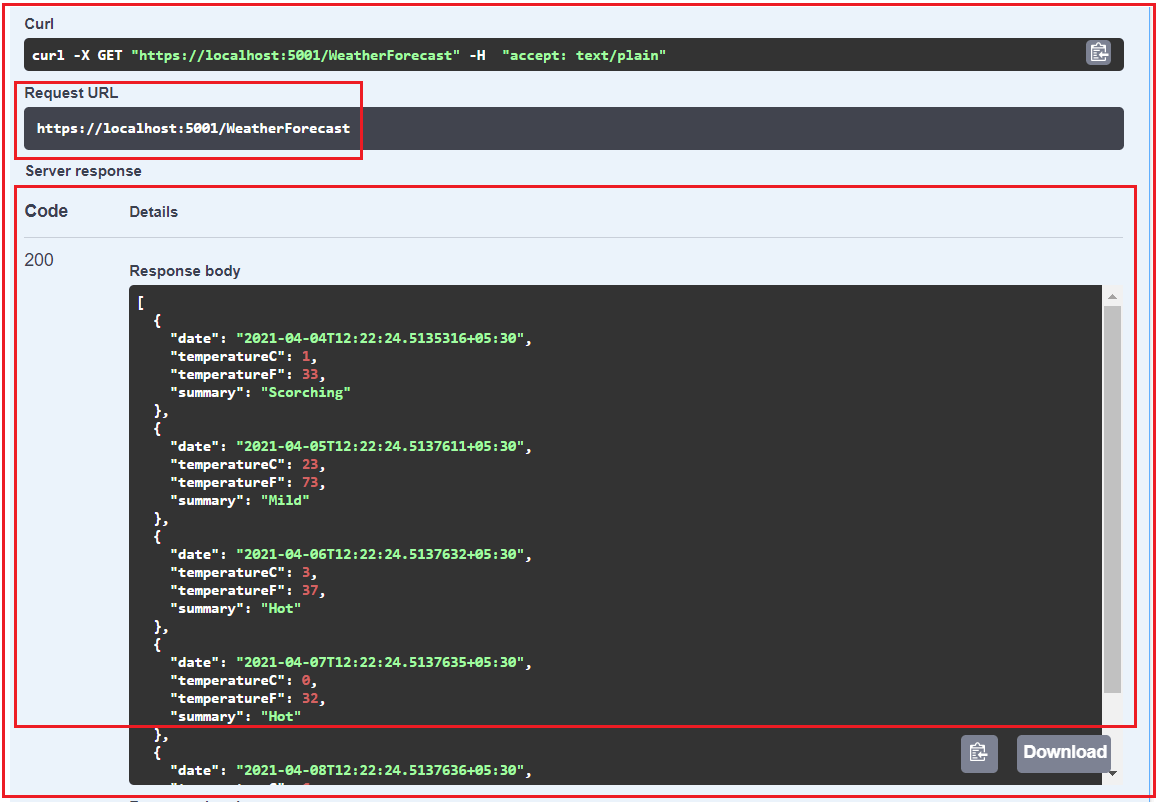
Now let us see how to test the API i.e. WeatherForecast API using swagger. To test the API using swagger, first, click on the **try it out** button as shown in the below image.



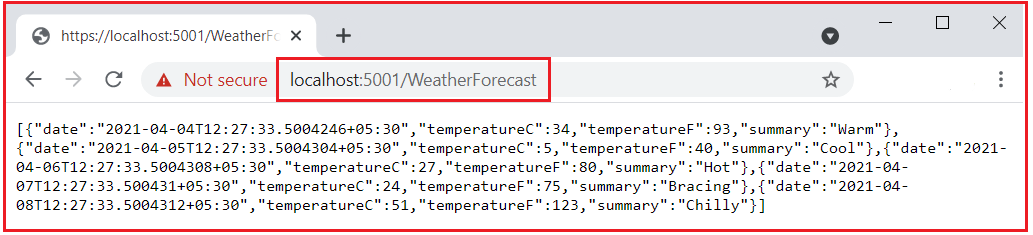
Once you click on the Try it Out button, it will open the below, and again here click on the Execute button as shown in the below image.



Once you click on the Execute button, it will give you the response as shown in the below image. Here, you can find the request URL, the response body, the response status code, and response headers.



In our upcoming article, we will discuss where this response is coming from. Even if you want you directly call the Request URL in any of your browsers and you will get the response as expected as shown in the below image.



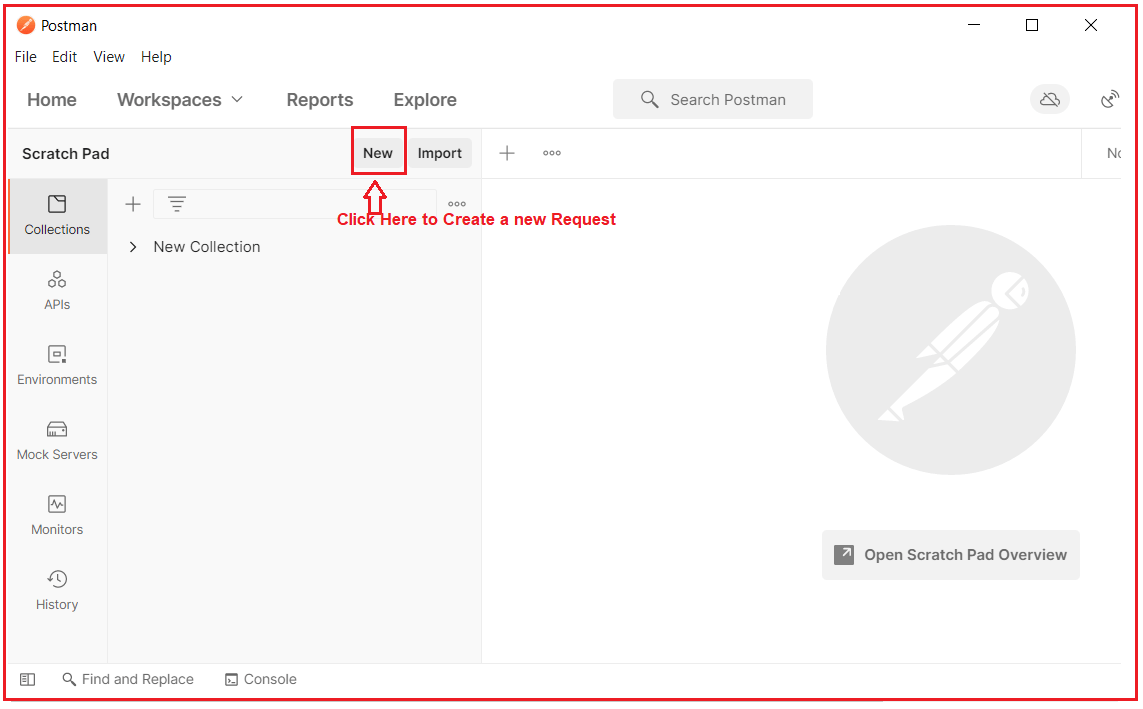
# How to Test ASP.NET Core Web API using Postman

## ****How to Test ASP.NET Core Web API using Postman?****

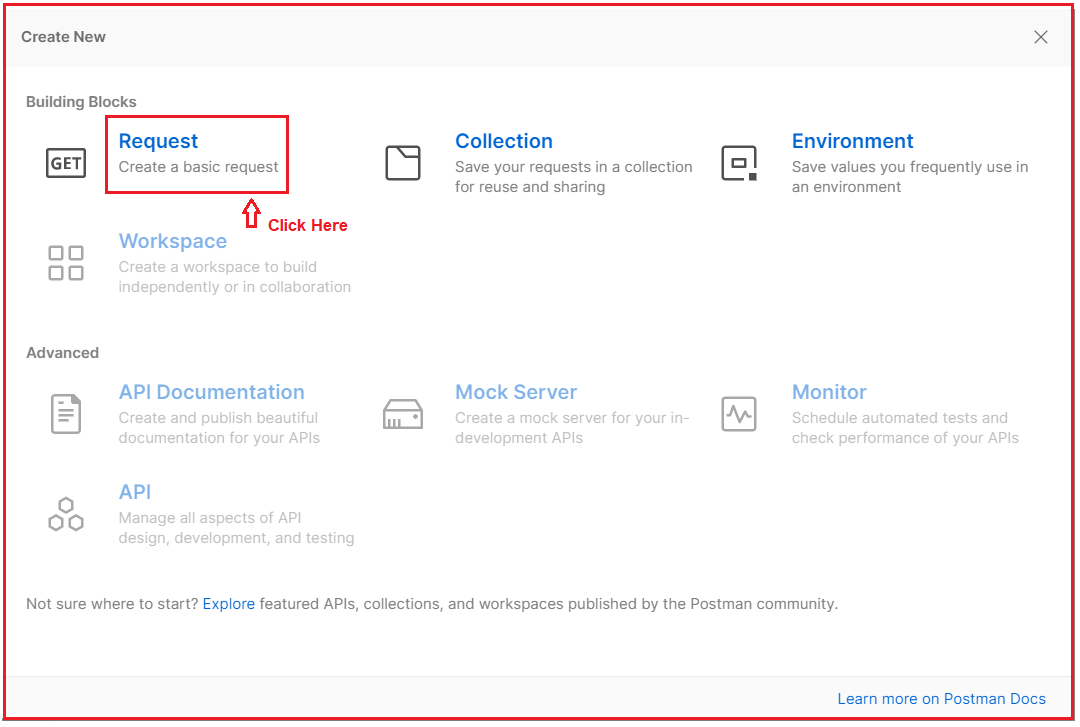
In this article, I am going to discuss **How to Test ASP.NET Core Web API using Postman**. Please read our previous article where we discussed [**How to Create, Build, Run and Test ASP.NET Core Web API Application**](https://dotnettutorials.net/lesson/creating-asp-net-core-web-api-project-using-net-core-cli/). In fact, this is a continuation part to our previous article. A postman is a client tool used for testing Restful APIs.

##### ****How to Test ASP.NET Core Web API using Postman?****

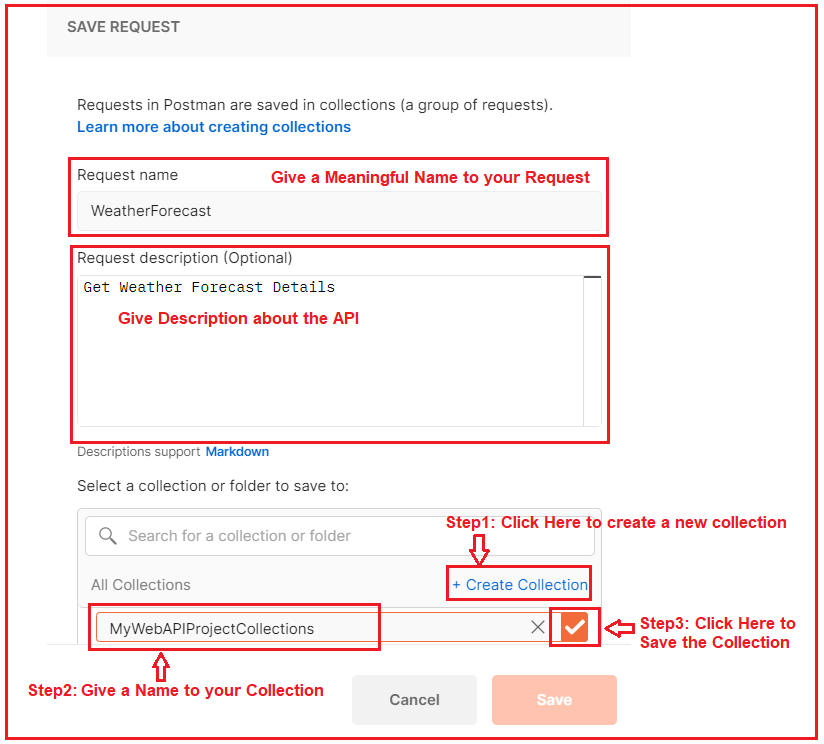
Now let us see, how to test ASP.NET Core Web API using Postman. First open Postman. Once you open the Postman, then click on the New button as shown in the below image to create a new request.



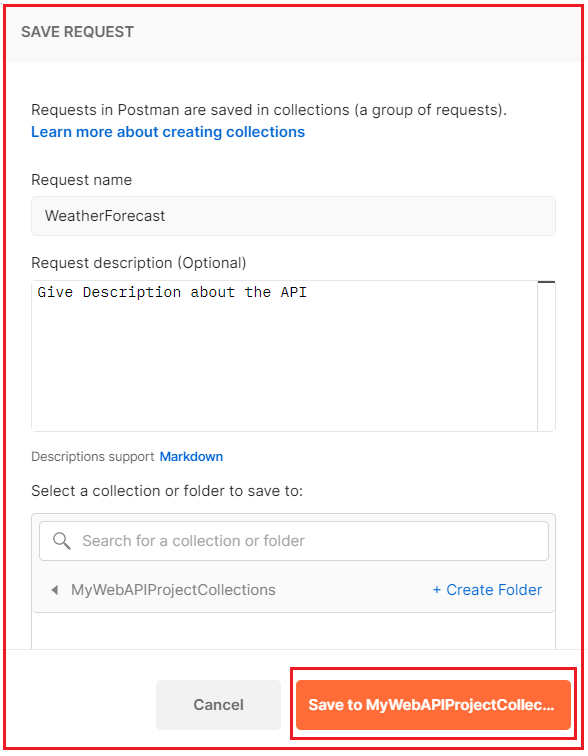
Once you click on the New button, it will open the following Create New window. Here, simply click on the Request tab as shown in the below image.



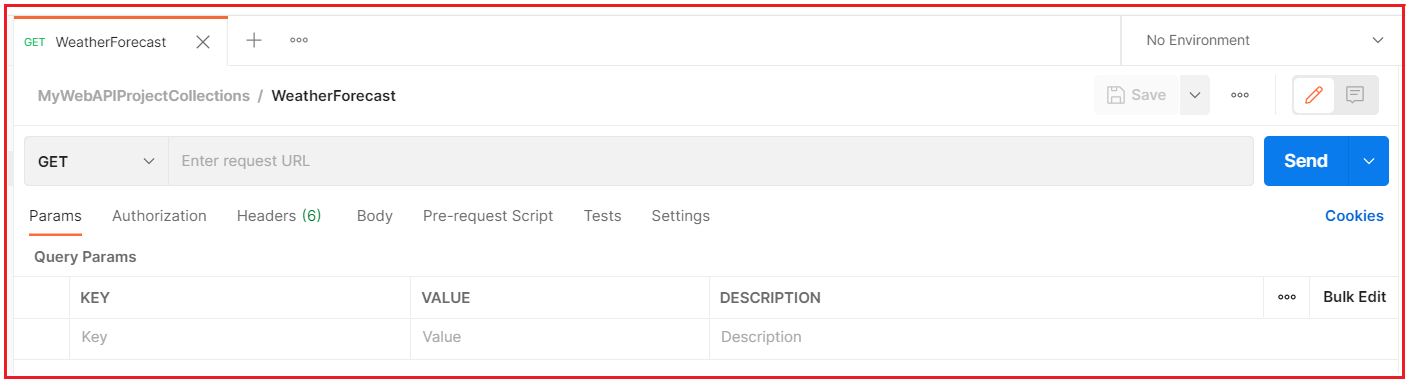
Once you click on the Request button, it will open the below Save Request window. Here, you need to do a couple of things. First, give a meaningful name to your API Request and provide a description of the API. Then you need to create one collection where you can create any of the Requests or you can use an existing collection to save this request. As we are doing for the first name, so, I am creating a new collection as shown in the below image.



Once you click on the right arrow button, the collection will be created and the Save button is renamed with the collection name and also that button is now enabled. Simply click on that button as shown in the below image.



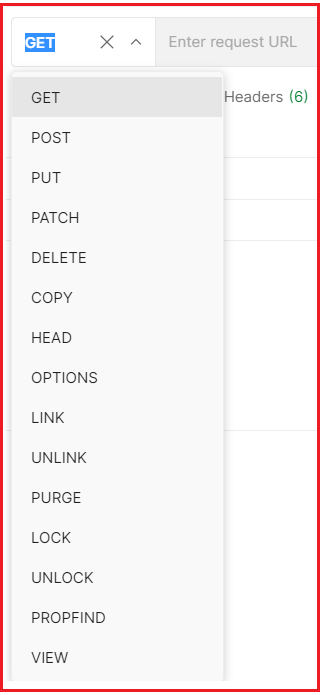
Once you click on the Save to Collection button, it will launch the following window.



Now let us understand the different components of the above Request.

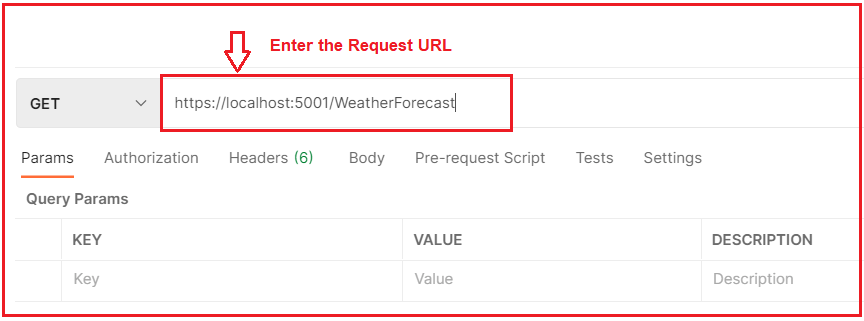
##### ****Selecting HTTP Method:****

You need to select the appropriate method of the API that you want to access. Here, in the dropdown, you can find the list of available HTTP Methods. As per your need to you select GET, POST, PUT, PATCH, DELETE, etc. In our example, the WeatherForecast API is of type GET, so, here we need to select the GET HTTP Method.



##### ****Enter the Request URL:****

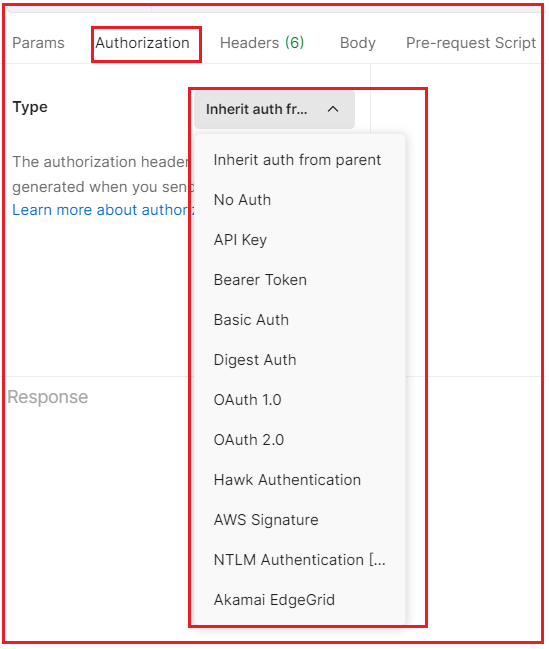
In the Enter Request URL text, you need to provide the API URL that you want to access as shown in the below image. As we are going to access the WeatherForecast API, So, I enter the URL as https://localhost:5001/WeatherForecast.



**Params:** If you want to send any parameters then you need to set those parameters as key-value pairs in the Params tab. In our example, we not going to pass any parameters to the Web API, so we keep this empty.

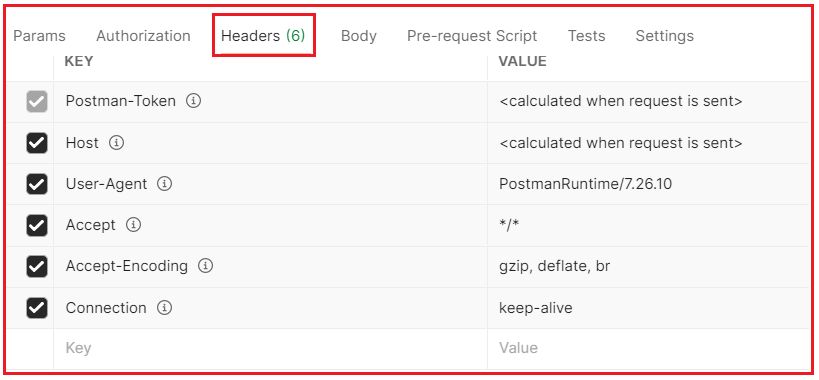
##### ****Authorization:****

If your Web API needs some kind of authorization, then you need to provide such authorization here. In the type tab, you need to select the appropriate authorization and you also need to provide the value for the same. In our example, we don’t require any authorization, so we also keep this tab empty.



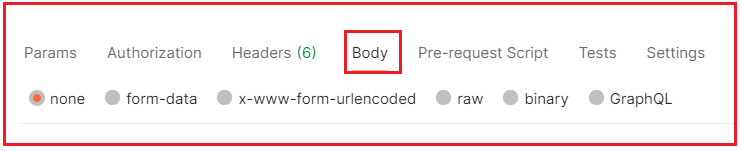
##### ****Headers:****

The next one is the Headers tab. As you can see, the postman by default sends the following headers to the Web API. If your API needs some additional headers, then you can set those headers here only in the form of Key-Value Pairs. In our example, the API does not require any additional headers. So, we are only going with the default headers.

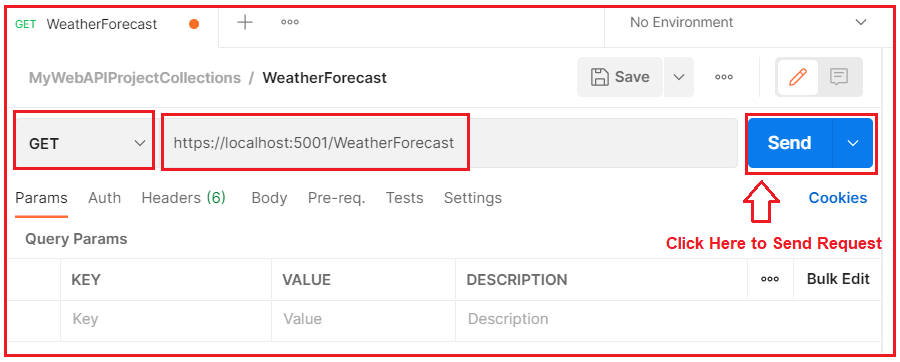


##### ****Body:****

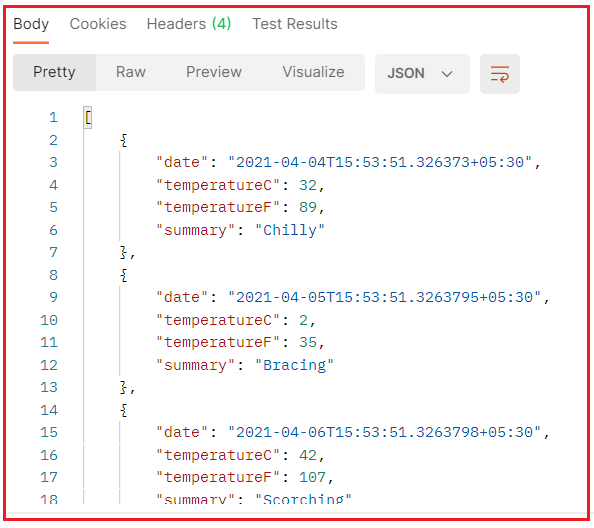
If your API accepts some data in the body, then you can send such data in the body tab. Further depending upon the type of data you can choose form-data, raw, binary, etc as shown in the below image. As our API is a GET request, as it doesn’t require any data, so none is selected by default.



Now, click on the Send button, which will send the request to the Web API server as shown in the below image.



Once you click on the Send button, it will make a request to the Web API and the Web API sends the response back to the Client. And you can see the response in the response body as shown in the below image.



In the response, you can also check the HTTP status code, the time it takes, and the size of the data it received as shown in the below image.

How to Test ASP.NET Core Web API using Postman

This is how you can use Postman to test your Web APIs. In the next article, I am going to discuss [**how to create, build, run, and test the ASP.NET Core Web API project using Visual Studio 2019**](https://dotnettutorials.net/lesson/asp-net-core-web-api-project-in-visual-studio-2019/). Here, in this article, I try to explain **How to use Postman to test ASP.NET Core Web APIs** and I hope you enjoy this article.

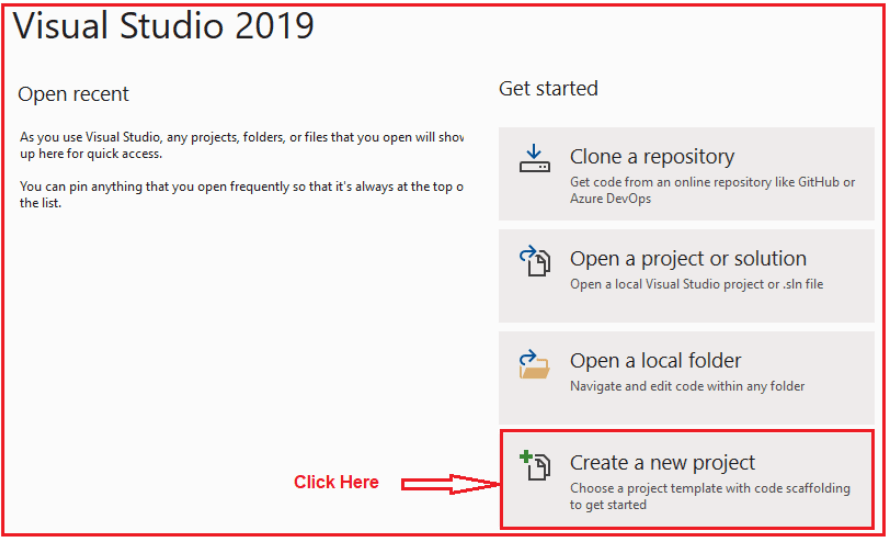
# Creating ASP.NET Core Web API Project in Visual Studio 2019

## ****Creating ASP.NET Core Web API project in Visual Studio 2019****

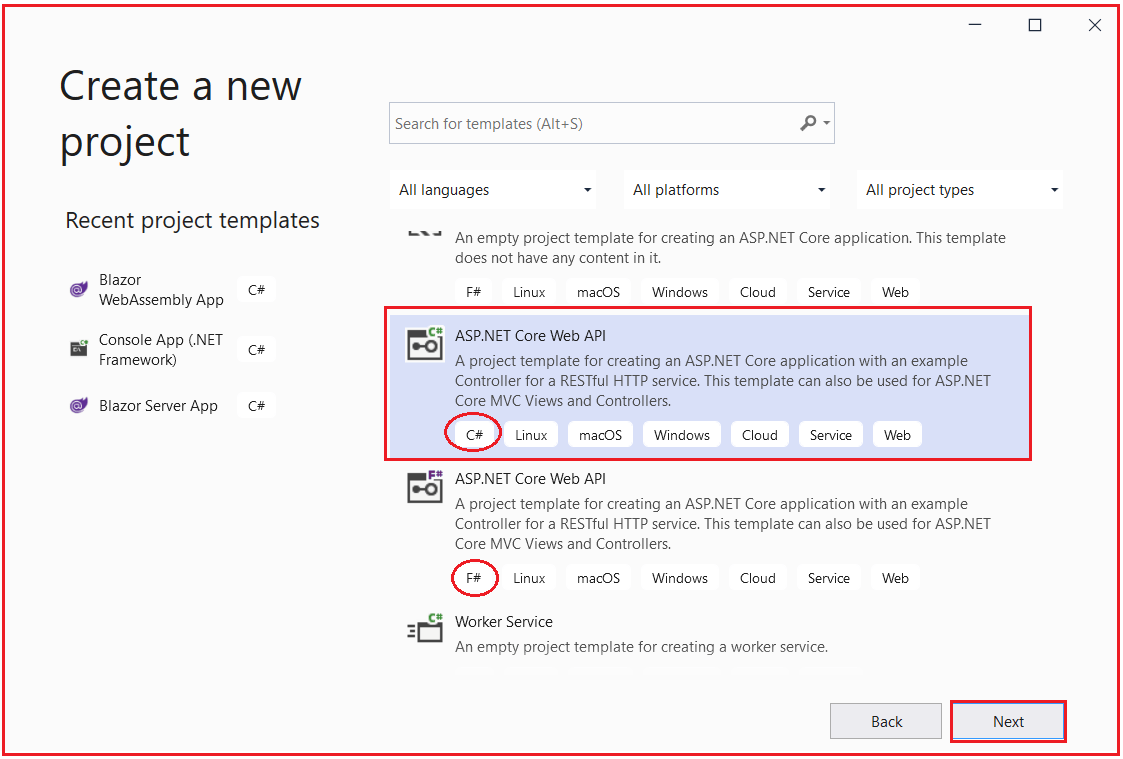
In this article, I am going to discuss **How to Create, Build, Run, and Test the ASP.NET Core Web API project in Visual Studio 2019**. Please read our previous article, where we discussed [**How to Create, Build, Run, and Test ASP.NET Core Web API project using .NET Core CLI**](https://dotnettutorials.net/lesson/creating-asp-net-core-web-api-project-using-net-core-cli/) and Visual Studio Code.

##### ****Creating ASP.NET Core Web API Project Using Visual Studio 2019****

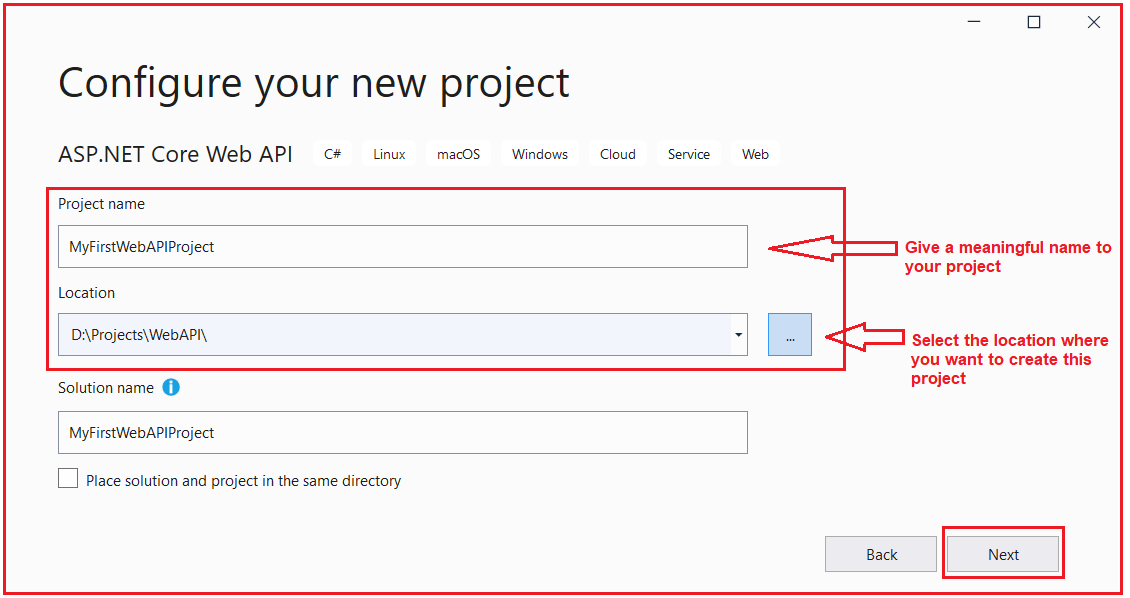
Now, we are going to create the ASP.NET Core Web API project using Visual Studio 2019. So, open Visual Studio 2019 and then click on the Create a new project option as shown in the below image.



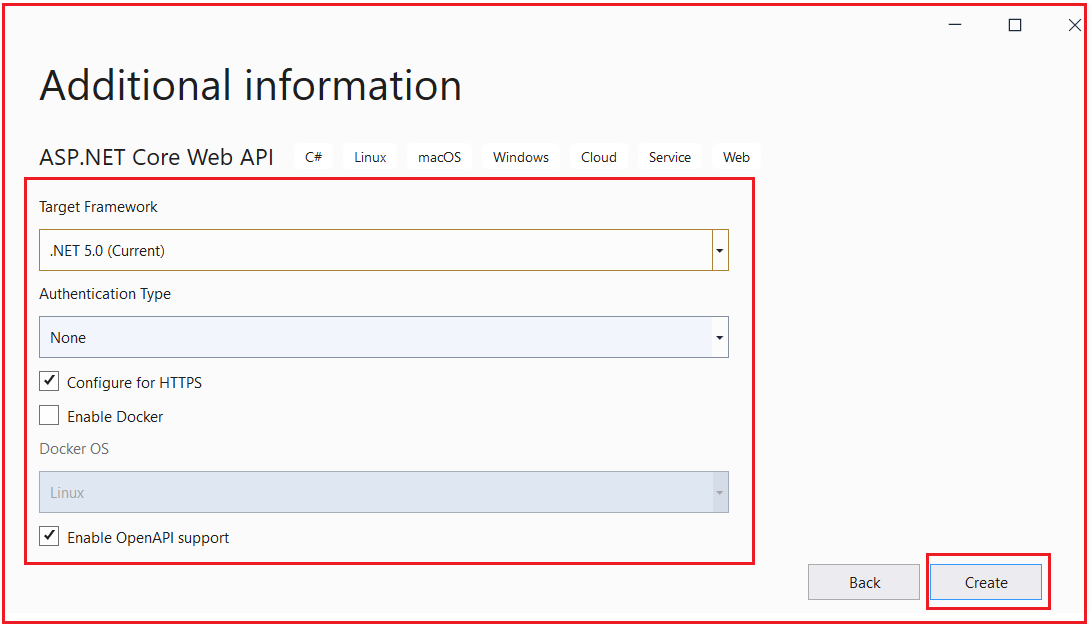
Once you click on the Create a new project option, the following Create a new project window will open. Here, you can find two projects template for creating the **ASP.NET Core Web API**project. One is using C# language and the other one is using F# language. I am going to use C# as the programming language, so I select the project template which uses C# Language as shown in the below image.



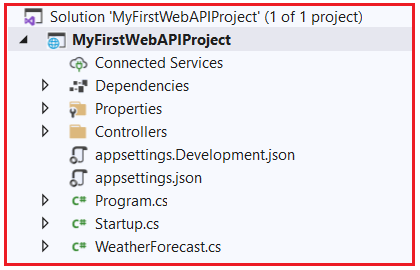
Once you click on the **Next** button, then the configure your new project window will open. Here, you need to specify the Project name (MyFirstWebAPIProject) and the location where you want to create the project. And finally, click on the **Next**button as shown in the below image.



Once you click on the Next button, it will open the Additional Information window. Here, you need to select the Target .NET Framework version. The authentication Types. Whether you want to configure HTTPS and enable Docker. Select .NET 5, select authentication type as None, check the Configure for HTTPS and uncheck the Enable Docker checkboxes and then click on the Create button as shown in the below image.

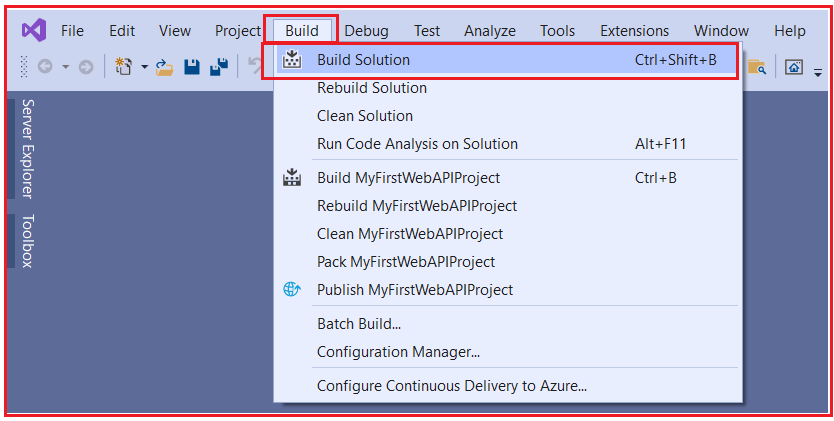


Once you click on the Create button, it will create the ASP.NET Core Web API project with the following file and folder structure. In our next article, we will discuss all these files and folders in detail.

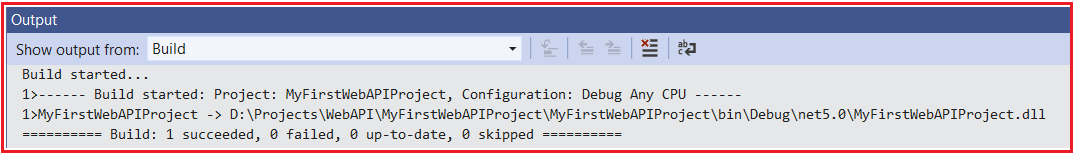


##### ****How to Build the ASP.NET Core Web API project in Visual Studio 2019?****

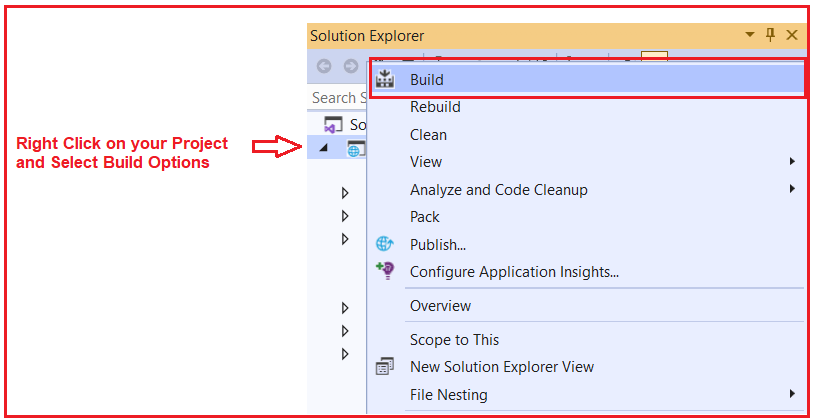
We can build the ASP.NET Core Web API Project in many ways. So, let us discuss them. From the menus, select Build => Build Solution as shown in the below image.



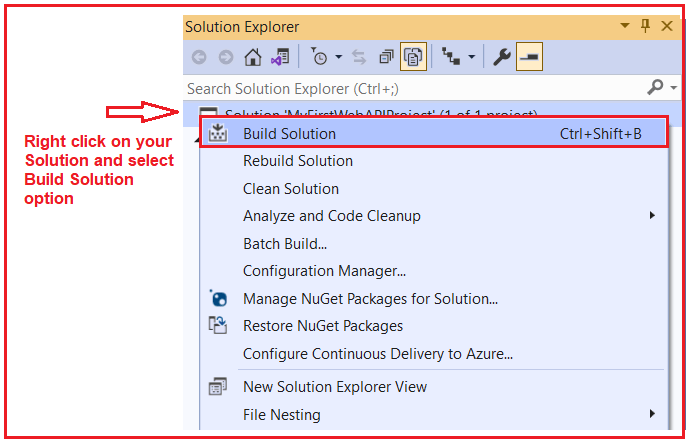
Once you select Build => Build Solution, then it will build all the projects that are there in the solution. Once the build is successful, you will get the below message in the output.



You can also build your project by right-clicking on your project and then selecting the build option from the context menus as shown in the below images.



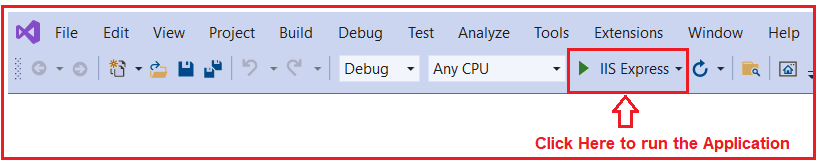
You can also right-click on your solution and select the Build Solution option to build all your projects as shown in the below image.



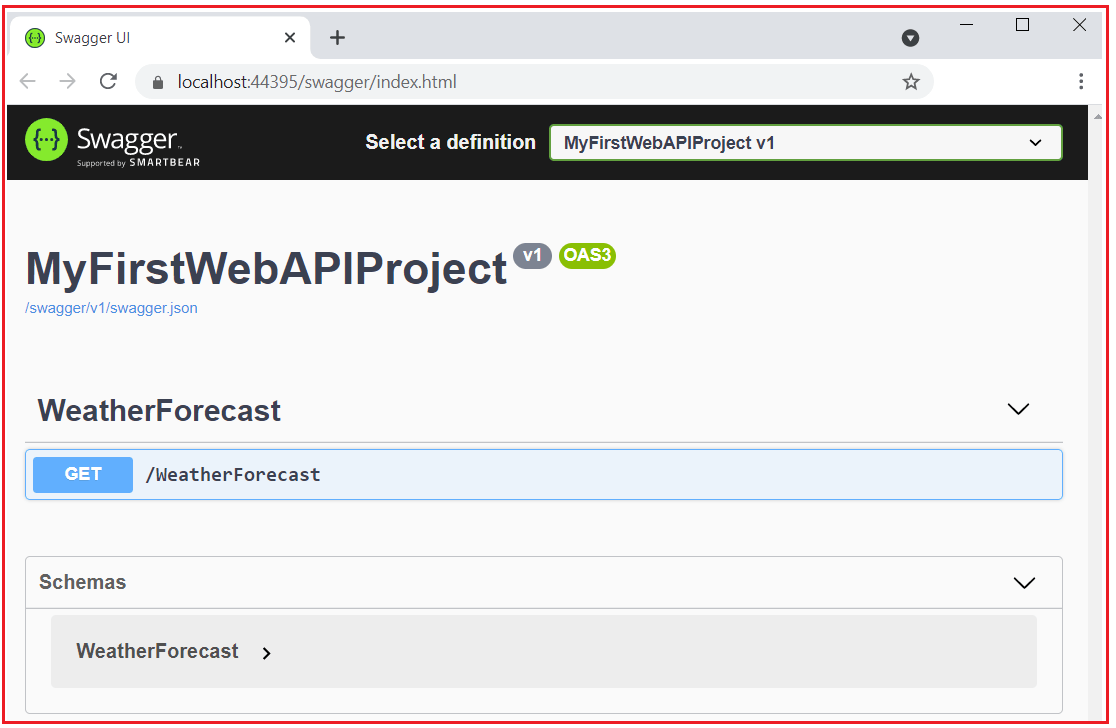
**Note:** If you use the Build Solution option, then it will build all your projects that are present inside the solution. If you want to build a particular project, then simply right-click on the Project and select the Build option which will build that project only. The last option to build projects in Visual Studio is a Keyboard shortcut. You can use **Ctrl+Shift+B** to build your solution.

##### ****How to run the ASP.NET Core Web API Application in Visual Studio 2019?****

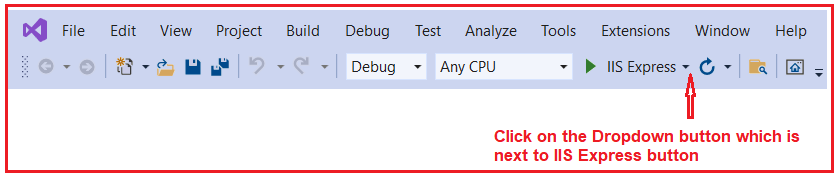
To run the application, simply click on the IIS Express green button as shown in the below image.



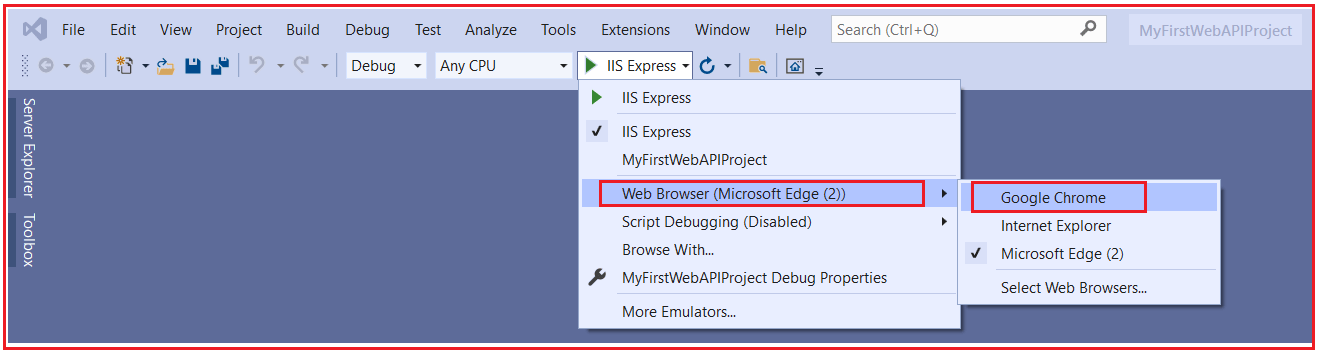
Once you click on the IIS Express button, it will run the application and you will get the following swagger page in the browser. Please have a look at the Port number (44395) on which the application is running.



Here, the above swagger opens in the default Microsoft Edge browser. If you want then you can also change the default browser to something else like Google Chrome. To do so, simply click on the dropdown button which is appeared next to the IIS Express button as shown in the below image.

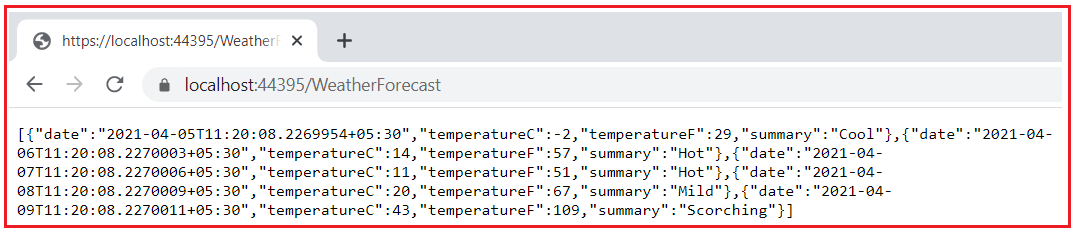


Then select Google Chrome as the Web Browser as shown in the below image.



With the above changes in place, now run the application and you must the web browser as Google Chrome.

Now, have look at the port number on which the application is running. My application is running on port number 44395. Now you test the API using the same approach that we have discussed in our previous two articles. So, I am not going to discuss the same here. You just try yourself and if you face any problem, then let us know by putting a comment in the comment box. To simplify, I am just hitting the URL (https://localhost:44395/WeatherForecast) from my web browser and I am getting the result as expected as shown in the below image.



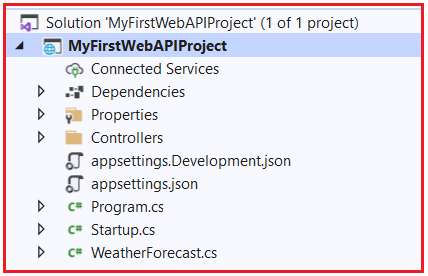
**ASP.NET Core Web API Files and Folders**

**ASP.NET Core Web API Files and Folders**

In this article, I am going to discuss the **Files and Folders which are created by default when we create a new ASP.NET Core Web API** Application. Please read our previous article where we discussed[**How to Create, Build, and Run ASP.NET Core Web API**](https://dotnettutorials.net/lesson/asp-net-core-web-api-project-in-visual-studio-2019/) in Visual Studio 2019.

**Folders and Files in ASP.NET Core Web API:**

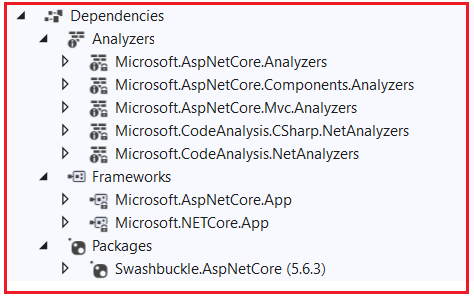
In our previous article, we created the first ASP.NET Core Web API Project and we also see that the project is created with the following files and folder structure.



Now let us proceed and understand the above files and folders in detail.

**Dependencies:**

The Dependencies contains all the packages and SDKs that are installed in this project. Further, if you expand, it contains three files (Analyzers, Frameworks, and Packages) as shown in the below image.



**Packages:**

At the moment the Packages folder contains one package i.e. Swashbuckle.AspNetCore. If you remember when we run the application, then it opens the swagger page. This package is basically for Swagger. Later if you add any new packages to your project, then that package will be shown here.

**Frameworks:**

The framework contains two folders i.e. **Microsoft.AspNetCore.App** and **Microsoft.NETCore.App**. All the packages that are required to run ASP.NET Core Application will be there inside Microsoft.AspNetCore.App file. The ASP.NET Core is a framework that is written on top of the .NET Core. So, all those packages that are required for the .NET core are available inside Microsoft.NETCore.App. So, the point that you need to remember is .NET Core and ASP.NET Core are two different things. All the packages which are specific to ASP.NET Core will reside inside **Microsoft.AspNetCore.App** and all the package which are specific to .NET Core will reside inside the **Microsoft.NETCore.App.**

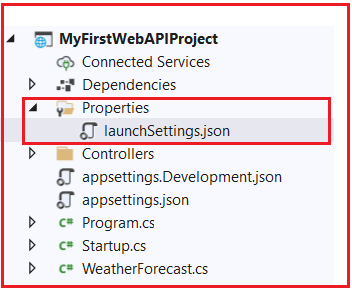
**Analyzers:**

The ASP.NET Core 2.2 and later provides analyzer packages intended for use with Web API projects. The analyzers will work with controllers annotated with ApiControllerAttribute while building on web API conventions. The analyzers package notifies you of any controller action that:

1. Returns an undeclared status code.
2. Returns an undeclared success result.
3. Documents a status code that isn’t returned.
4. Includes an explicit model validation check.

**Properties:**

The Properties Folder in ASP.NET Core Web Application by default contains one JSON file called as launchsettings.json file as shown in the below image.



The launchsettings.json file contains some settings that are going to be used by .NET Core Framework when we run the application either from Visual Studio or by using .NET Core CLI. Another point that you need to keep in mind, the launchSettings.json file is only used within the local development machine. So, this file is not required when we publishing our ASP.NET Core Web API application into the production server. Now, open the launchSettings.json file, by default you will see the following settings.



If you remember, when we are running our application using .NET Core CLI, we are getting two URLs (**https://localhost:5001** and **http://localhost:5000**). These two URLs are coming from this MyFirstWebAPIProject Profile.

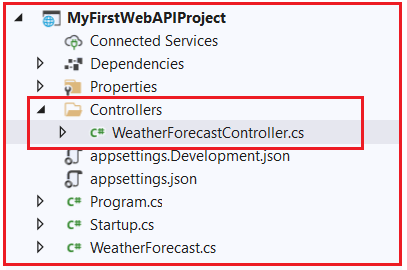
Further, if you remember when we run our application using Visual Studio, the application run on port number 44395. For IIS Express Profile, it uses **iissettings**. If you notice the iisSettings, the applicationUrl is **http://localhost:63044** and sslPort as **44395**. That means if you are running the application using the HTTP protocol, then it will use the **63044** port number and if you run the application using HTTPS protocol then it will use the 44395 port number.

**Note**: If you are running your application from the visual studio then IIS Express Profile will be used (for HTTP the port number will be 63044 and for HTTPS the port number will be 44395). On the other hand, if you are running your application using .NET Core CLI, then MyFirstWebAPIProject profile will be used which is nothing but using Kestrel Web Server and for HTTP protocol it uses the port number 5000 and for HTTPS protocol it uses the port number 5001.

[**Learn More about the launchsettings.json file.**](https://dotnettutorials.net/lesson/asp-net-core-launchsettings-json-file/)

**Controllers:**

The ASP.NET Core Web API is a controller-based approach. All the controllers of your ASP.NET Core Web API Application should and must reside inside the Controllers folder. Here, by default one controller is there inside the Controllers folder as shown in the below image.



If you open the WeatherForecastController file, then you will find the following code in it. The Controller class us inherited from the ControllerBase class and decorated with ApiController and Route attribute. We will discuss all these things in our upcoming article.

**using** *Microsoft.AspNetCore.Mvc;*

**using** *Microsoft.Extensions.Logging;*

**using** *System;*

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**using** *System.Threading.Tasks;*

**namespace** *MyFirstWebAPIProject.Controllers*

**{**

**[**ApiController**]**

**[**Route**(**"[controller]"**)]**

**public** **class** WeatherForecastController : ControllerBase

**{**

**private** **static** **readonly** string**[]** Summaries = new**[]**

**{**

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

**}**;

**private** **readonly** ILogger**<**WeatherForecastController**>** \_logger;

**public** WeatherForecastController**(**ILogger**<**WeatherForecastController**>** logger**)**

**{**

\_logger = logger;

**}**

**[**HttpGet**]**

**public** IEnumerable**<**WeatherForecast**>** Get**()**

**{**

var rng = new Random**()**;

**return** Enumerable.Range**(**1, 5**)**.Select**(**index =**>** new WeatherForecast

**{**

Date = DateTime.Now.AddDays**(**index**)**,

TemperatureC = rng.Next**(**-20, 55**)**,

Summary = Summaries**[**rng.Next**(**Summaries.Length**)]**

**})**

.ToArray**()**;

**}**

**}**

**}**

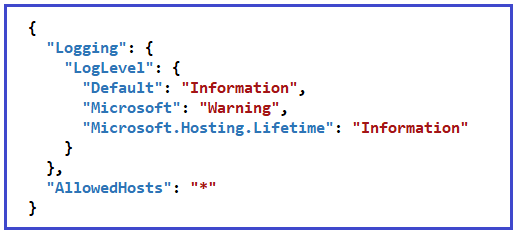
As you can in the above image, we have one Get method which is decorated with the HttpGet attribute. And when we call the following URL using any client (Swagger, Postman, and Browser), this is the method going to return the data. We will discuss Controller in detail in our upcoming articles.

**https://localhost:44395/WeatherForecast**

**appsettings.json file:**

The next file that we are going to discuss is the appsettings.json file. This is the same as web.config or app.config of our traditional .NET Application. The appsettings.json file is the application configuration file in ASP.NET Core Web Application used to store the configuration settings such as database connections strings, any application scope global variables, etc.

If you open the appsettings.json file, then you see the following code by default which is created by .NET Core Framework when we created the ASP.NET Core Web API Application.



[**Learn More about the appsettings.json file.**](https://dotnettutorials.net/lesson/asp-net-core-appsettings-json-file/)

**appsettings.Development.json:**

If you want to configure some settings based on the environments then you can do such settings in appsettings.{Environment}.json file. You can create n number of environments like development, staging, production, etc. I

If you set some settings in the appsettings.Development.json file, then such settings can only be used in the development environment, can not be used in other environments. If you open the appsettings.Development.json file, then you will get the following code in it. In our upcoming articles, we will see how to create and use different environment appSettings file in ASP.NET Core Web API Application.

**{**

"Logging": **{**

"LogLevel": **{**

"Default": "Information"**,**

"Microsoft": "Warning"**,**

"Microsoft.Hosting.Lifetime": "Information"

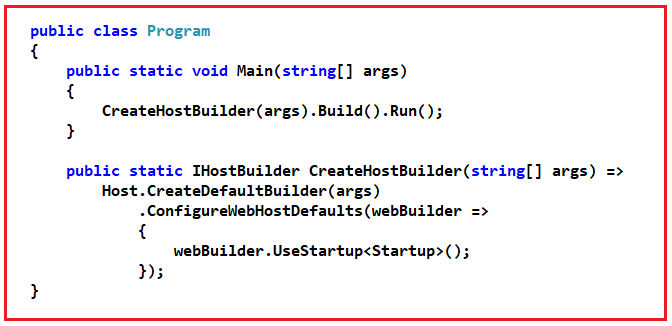
**}**

**}**

**}**

**Program.cs class file:**

The Program.cs class file of our ASP.NET Core Webb API Application contains the following code.



As shown in the above Program class code, it has a **public static void Main()**method. The Main method is the entry point of our Application.

Each ASp.NET Core Web API Application initially starts as a Console Application and the Main() method is the entry point to the application. So, when we execute the ASP.NET Core Web API application, it first looks for the Main() methodand this is the method from where the execution starts for the application. The Main() method then configures ASP.NET Coreand starts it. At this point, the application becomes an ASP.NET Core Web API application.

[**Learn More about ASP.NET Core Main Method.**](https://dotnettutorials.net/lesson/asp-net-core-main-method/)

**Startup.cs class file:**

The Startup class is like the Global.asax file of our traditional .NET application. As the name suggests, it is executed when the application starts. You will find the following code in your Startup class.

**namespace** *MyFirstWebAPIProject*

**{**

**public** **class** Startup

**{**

**public** Startup**(**IConfiguration configuration**)**

**{**

Configuration = configuration;

**}**

**public** IConfiguration Configuration **{** **get**; **}**

// This method gets called by the runtime. Use this method to add services to the container.

**public** **void** ConfigureServices**(**IServiceCollection services**)**

**{**

services.AddControllers**()**;

services.AddSwaggerGen**(**c =**>**

**{**

c.SwaggerDoc**(**"v1", new OpenApiInfo **{** Title = "MyFirstWebAPIProject", Version = "v1" **})**;

**})**;

**}**

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

**public** **void** Configure**(**IApplicationBuilder app, IWebHostEnvironment env**)**

**{**

**if** **(**env.IsDevelopment**())**

**{**

app.UseDeveloperExceptionPage**()**;

app.UseSwagger**()**;

app.UseSwaggerUI**(**c =**>** c.SwaggerEndpoint**(**"/swagger/v1/swagger.json", "MyFirstWebAPIProject v1"**))**;

**}**

app.UseHttpsRedirection**()**;

app.UseRouting**()**;

app.UseAuthorization**()**;

app.UseEndpoints**(**endpoints =**>**

**{**

endpoints.MapControllers**()**;

**})**;

**}**

**}**

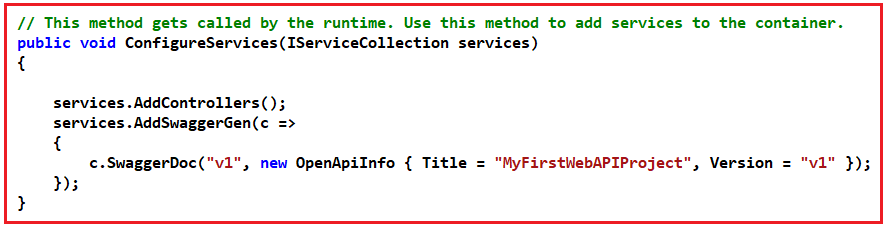
**}**

As you can see in the above code, the Startup class includes two public methods: ConfigureServices and Configure.

**ConfigureServices() method:**

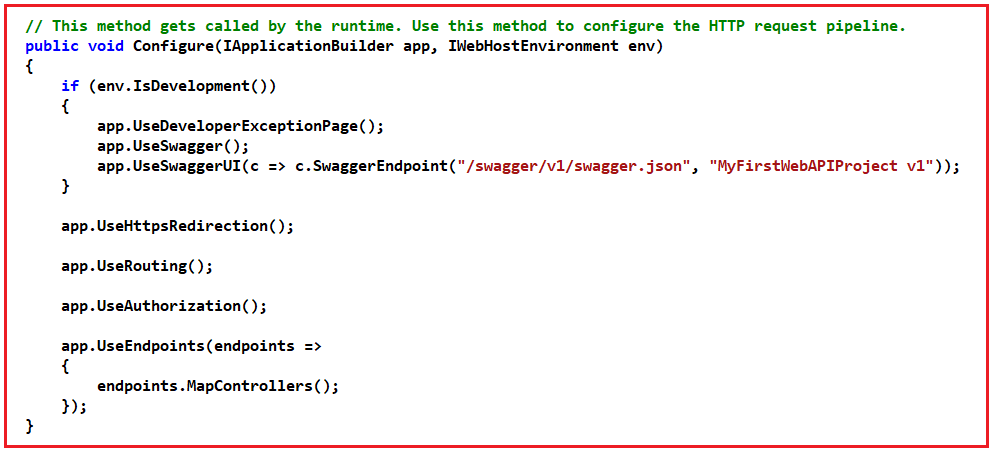
The ConfigureServices method of the Startup class is the place where we can register our dependent classes with the built-in IoC container. Once we register the dependent classes, they can be used anywhere within the application. The ConfigureServices method includes the IServiceCollection parameter to register services to the IoC container.

If you notice, currently it adds two services i.e. AddControllers and AddSwaggerGen to the build-in IoC Container as shown in the below image.



**Configure() method:**

The Configure method of the Startup class is the place where we configure the application request pipeline using the IApplicationBuilder instance that is provided by the built-in IoC container. ASP.NET Core introduced the middleware components to define a request pipeline, which will be executed on every request. If you look at the Configure method, it registered UseDeveloperExceptionPage, UseSwagger, UseSwaggerUI, UseHttpsRedirection, UseRouting, UseAuthorization, and UseEndpoints middleware components to the request processing pipeline as shown in the below image. We will discuss each middleware component in detail in our upcoming articles.



[**Learn more about the ASP.NET Core Startup class.**](https://dotnettutorials.net/lesson/asp-net-core-satrtup-class/)

**WeatherForecast.cs class file:**

This is the model class and you can find the following code in it.

**public** **class** WeatherForecast

**{**

**public** DateTime Date **{** **get**; **set**; **}**

**public** **int** TemperatureC **{** **get**; **set**; **}**

**public** **int** TemperatureF =**>** 32 + **(int)(**TemperatureC / 0.5556**)**;

**public** string Summary **{** **get**; **set**; **}**

**}**

**Note:** The Model can be placed anywhere. It can be placed inside the Models folder. Directly at the project level. Even they can be crated as separate class library projects.