<https://www.youtube.com/watch?v=MIJJCR3ndQQ>

**dotnet new webapi -o helloDotnet5 --no-https**

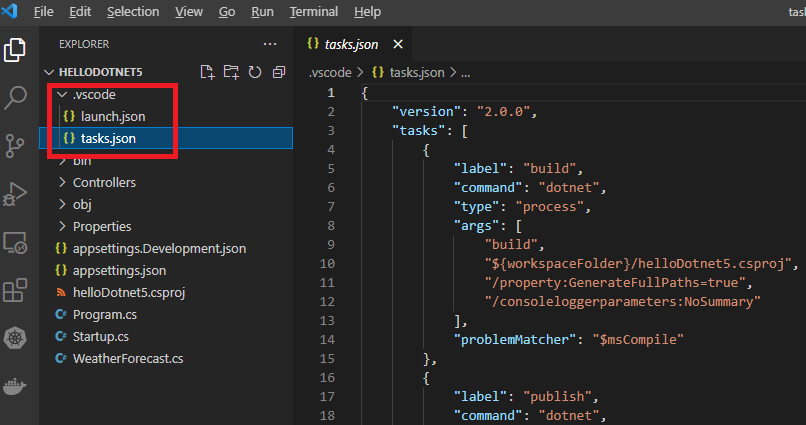
-o = output directory

–no-https = in microservice ssl is a concern usually delegated to infrastructure components like api gateway so microservice can safety avoid dealing with it.

**cd helloDotnet5**

**code .**

**Click Yes for adding c# files in VSCode**

****

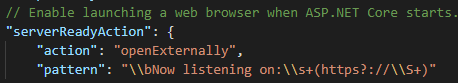
In **tasks.json** one of the tasks is **“build**” which run dotnet build (compile) with some arguments.

In **launch.json** we define how we launch the project from VSCode and one of the pre tasks is the **build** task in **tasks.json**

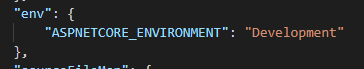
****

The task after that is launching the program

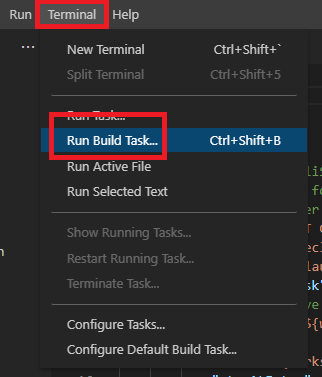
The last task is when the host is ready open a browser

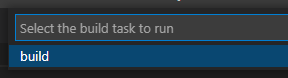


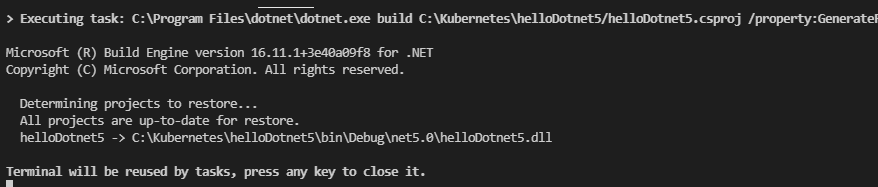
We also describe the asp.net core **environment** which is **Development** and not Production.

****

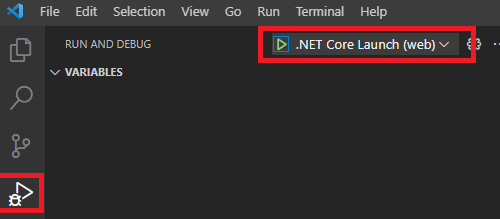
**Build the project (CTRL+SHIFT+B to open the Build task)**

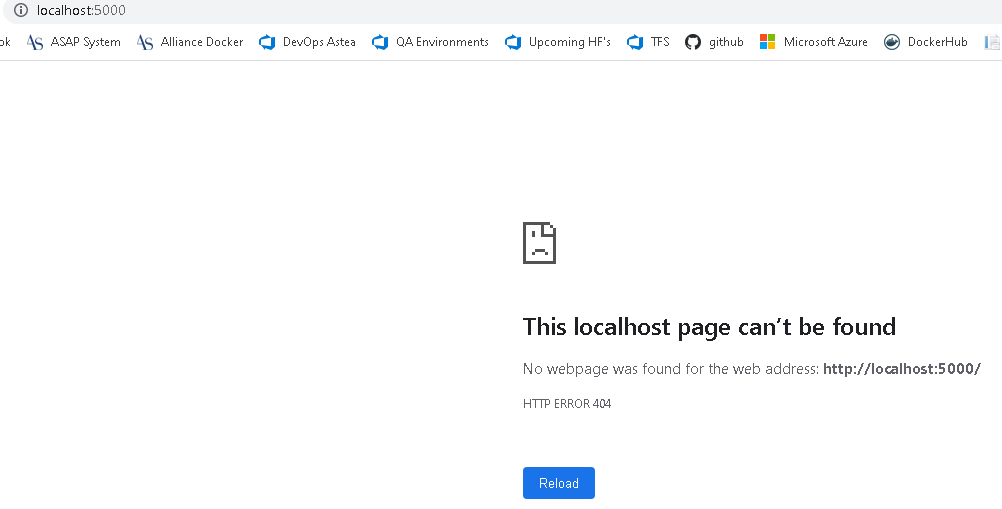
****

****

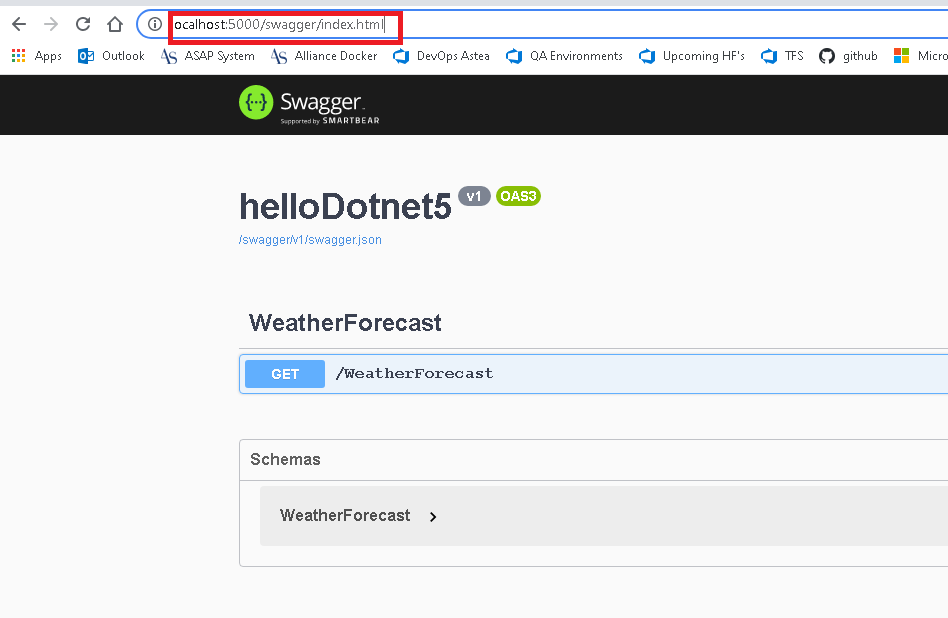
****

**Run the project:**

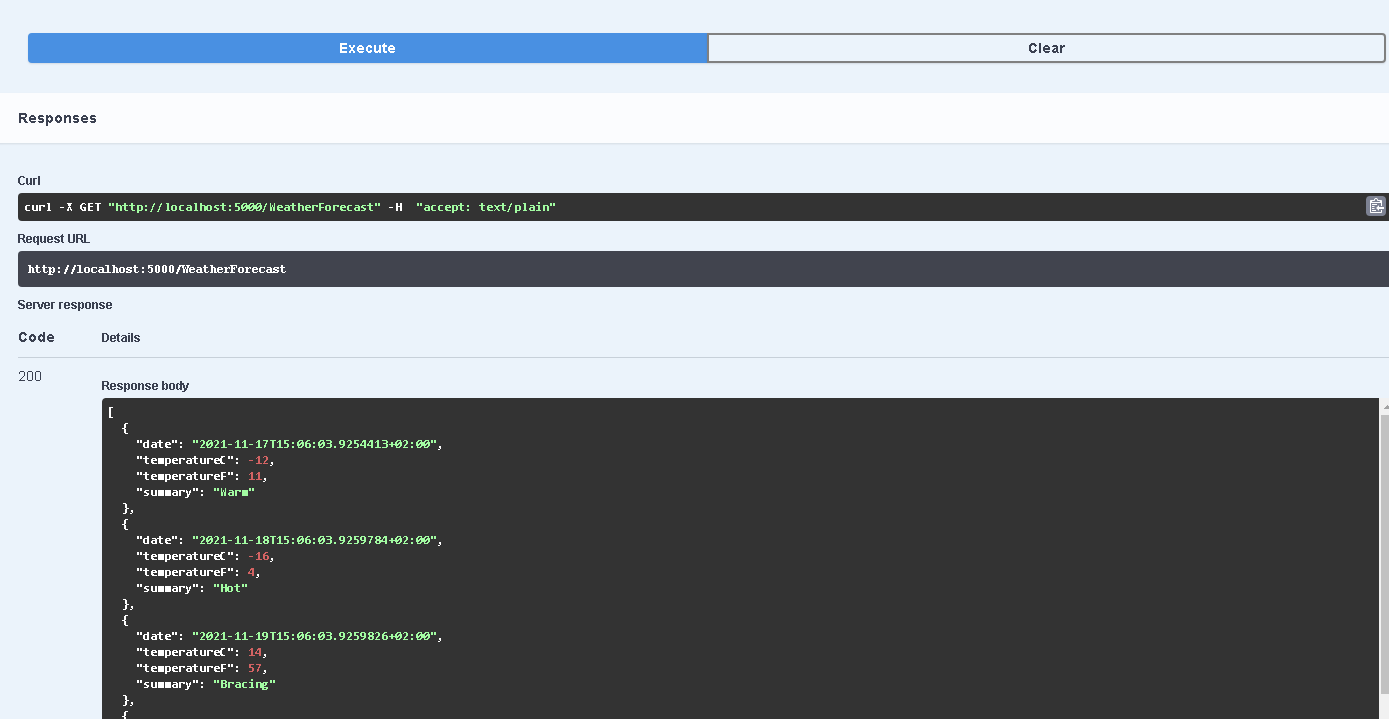
****

****

[**http://localhost:5000/swagger/index.html**](http://localhost:5000/swagger/index.html)

****

**Click Get + Try it out + Execute**

****

**To make the build a default task when use Run Build Task add group code as below in tasks.json:**

"problemMatcher": "$msCompile",

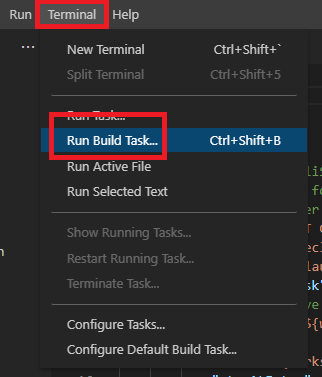
 "group": {

                "kind": "build",

                "isDefault": true

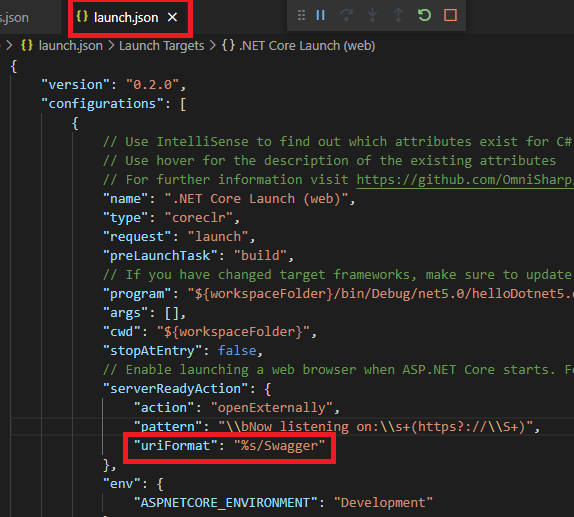
            }

**CTRL+ SHIFt + B or as below will build the project**

****

**Now it will go ahead and will not pop up the build window**

**In order to tell VSCode where you want to start when you launch – add the uriFormat as below**

****

 "uriFormat": "%s/Swagger"

**%s = append**

**/swagger = the path that we want to add to the URL -http://localhost:5000**

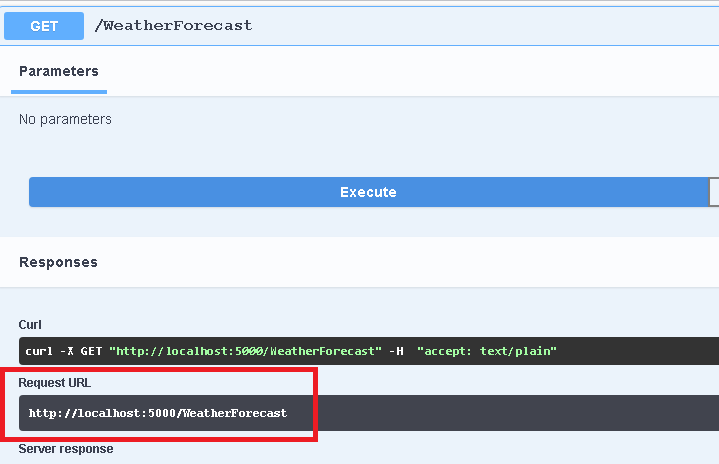
**F5 – Run the project**

**SHIFT+F5 = Stop the project**

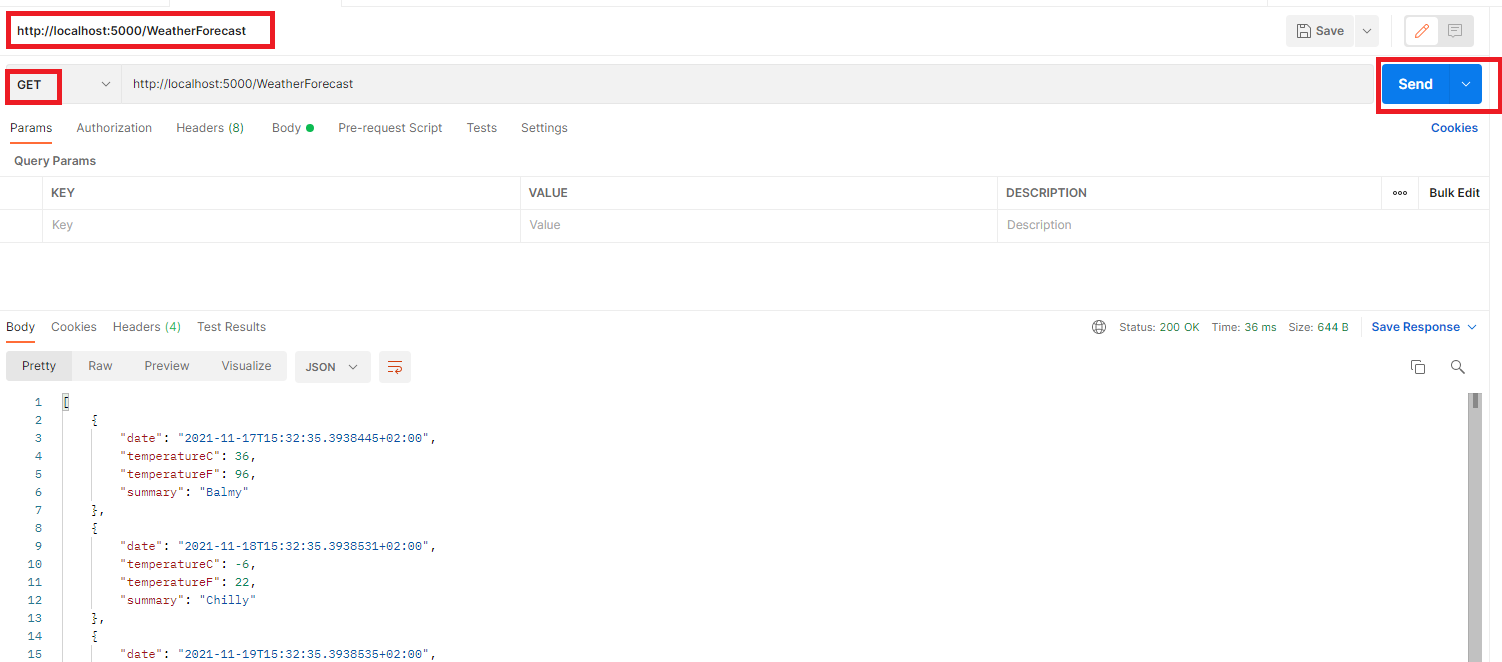
**In order to use postman instead of open the browser every time when we run the project run:**

**Run Postman-win64-8.12.4-Setup.exe:**

**Select Get and paste the request URL -** [http://localhost:5000/WeathterForecas](http://localhost:5000/WeatherForecast) that we executed in the browser.

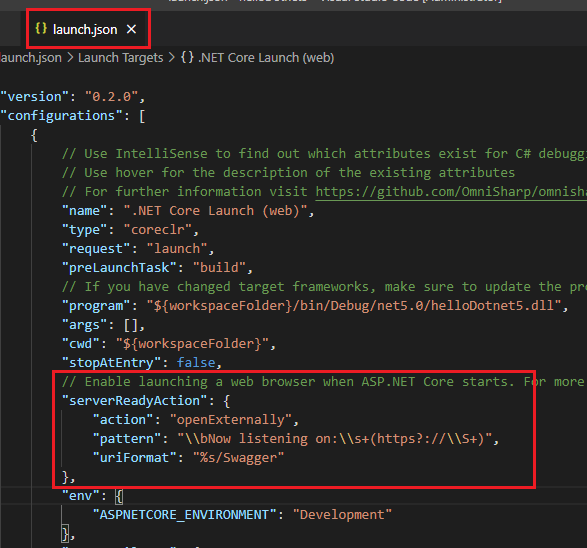
****

**Paste the request URL in postman and click Get.**

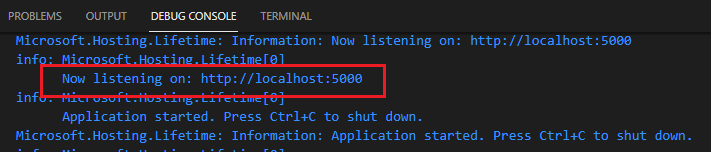
****

**We have the response.**

**Now in order to avoid opening the browser each time when we test the api – remove the lines below from launch.json:**

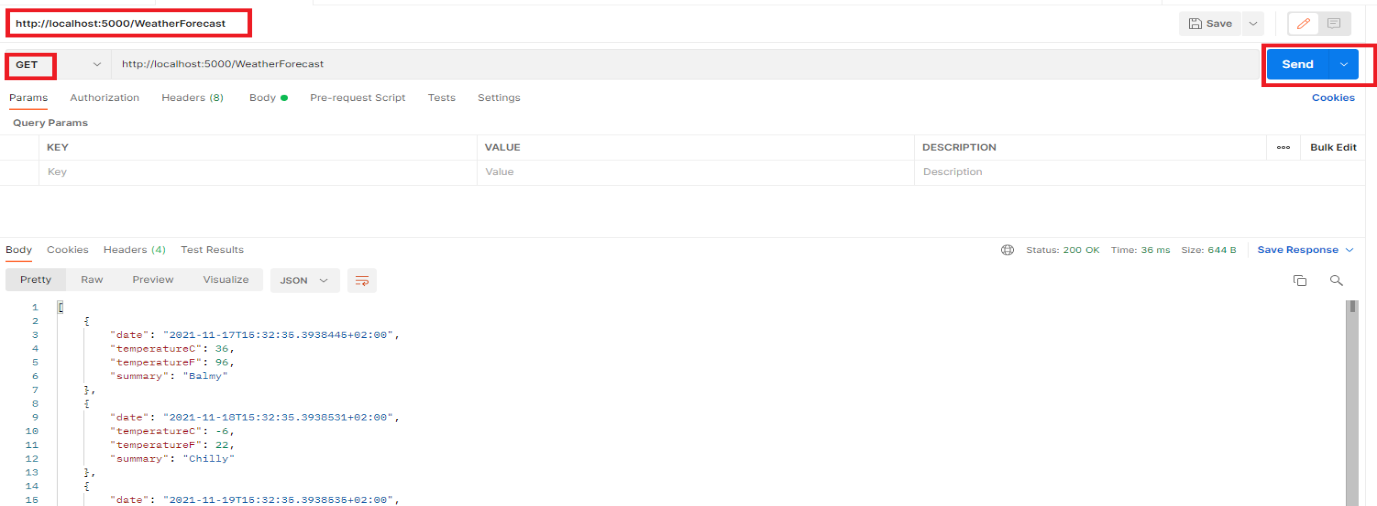
****

**CTRL+SHIFT+B -> F5**

****

**Now it says that the host is listing to http//localhost:5000**

**Paste the request URL in postman and click Get.** [http://localhost:5000/WeathterForecas](http://localhost:5000/WeatherForecast)

****

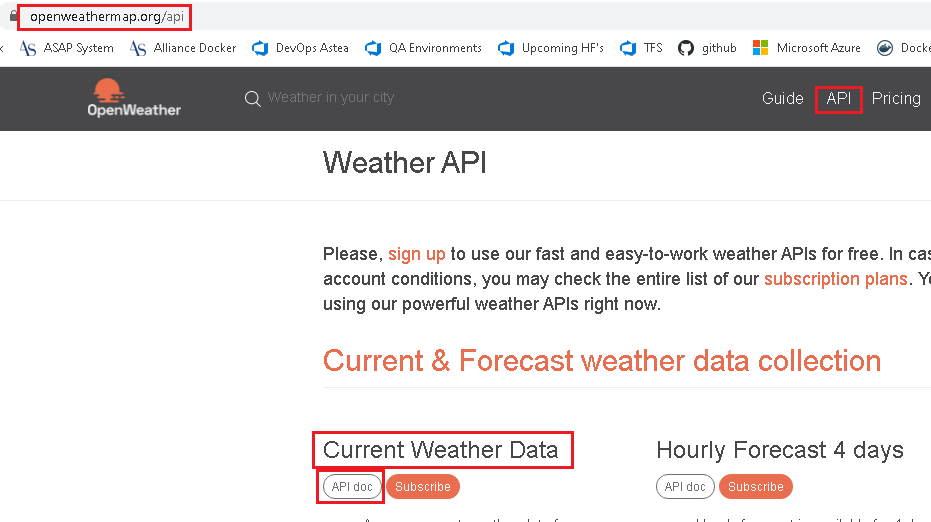
**We have the response.**

**Start programing**

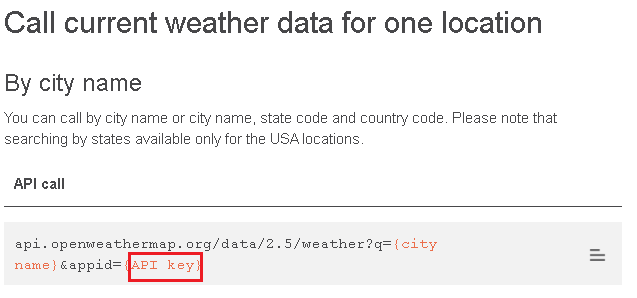
Make the program more reality and query some weather information from external service so we can tell the weather of some city.

We will use an external service called “open weather” - <https://openweathermap.org/>

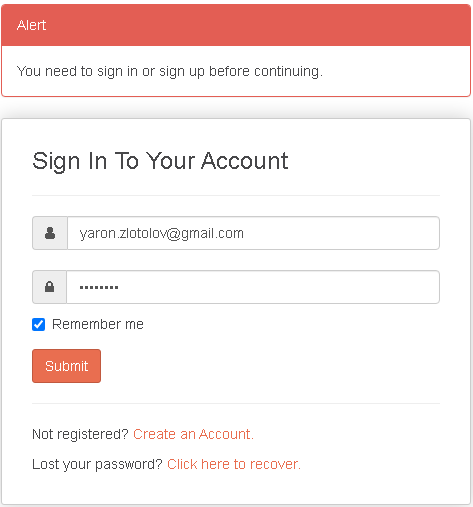
This service allows to query a bunch of api’s – we will use “Current Weather API”



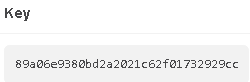
Click the **api doc** -> we can use the service below to query the weather of some city and for that we need to get the **API KEY** available



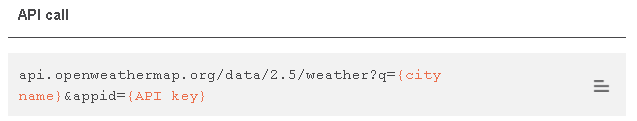
Click the API KEY will ask to sign in or create an account (it’s free)



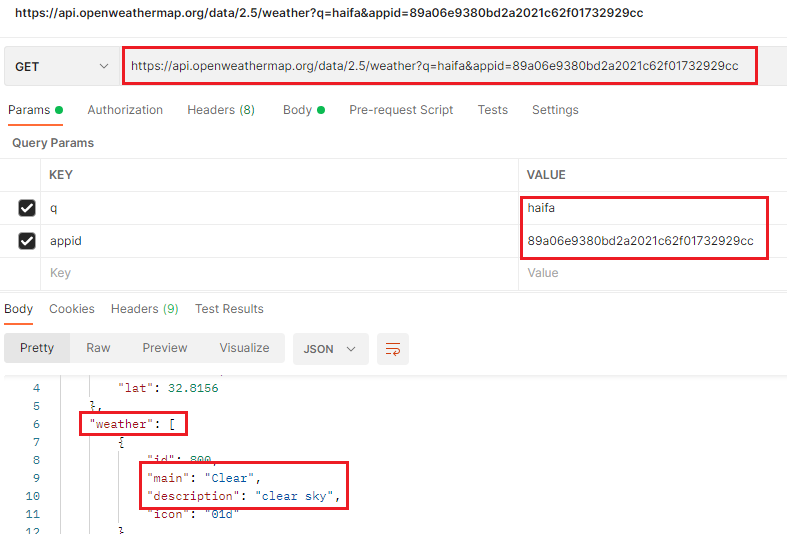
Go to API Keys and copy the key:

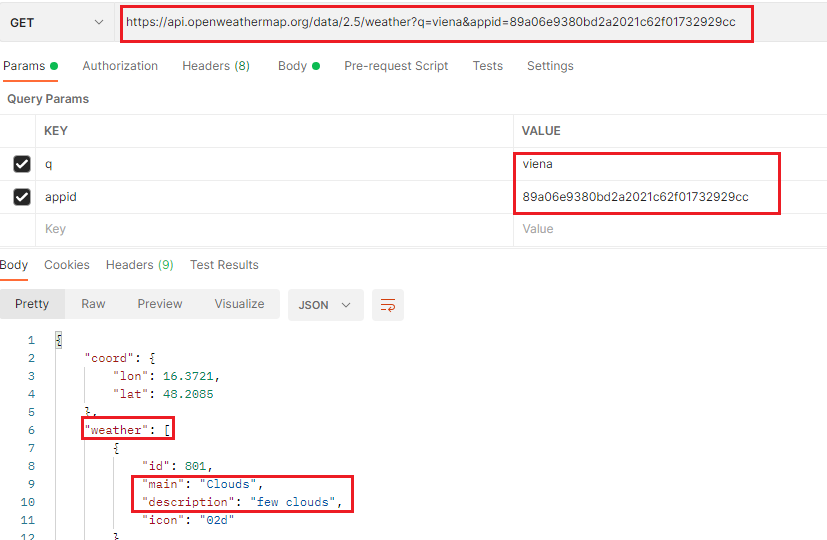


Now we will try the api by copy the query and paste it in postman with a city and the API Key.



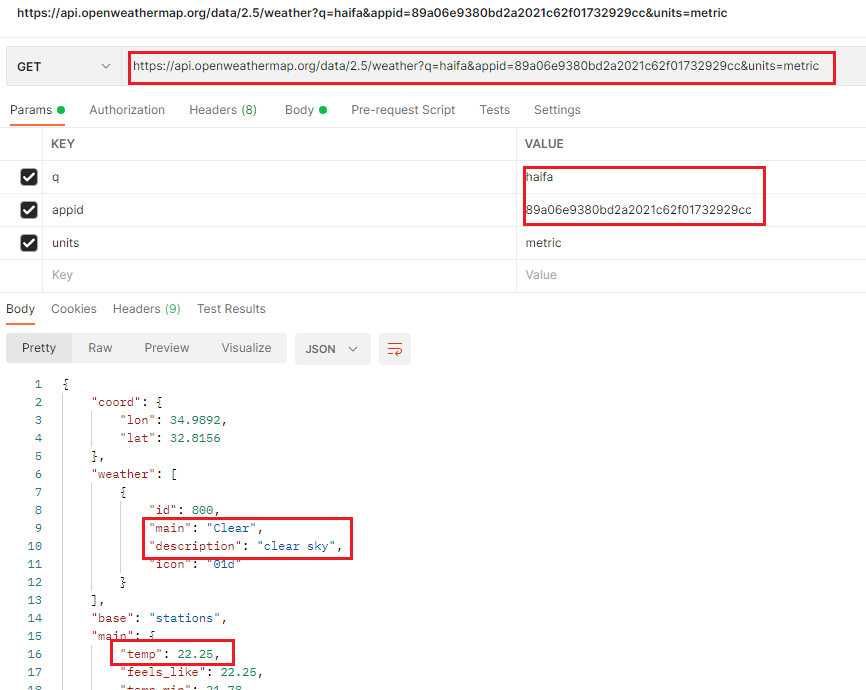
<https://api.openweathermap.org/data/2.5/weather?q=haifa&appid=89a06e9380bd2a2021c62f01732929cc>





\*In order to see the temperature in Celsius we will append &units=metric to the query

https://api.openweathermap.org/data/2.5/weather?q=haifa&appid=89a06e9380bd2a2021c62f01732929cc&units=metric



For our project we need to know:

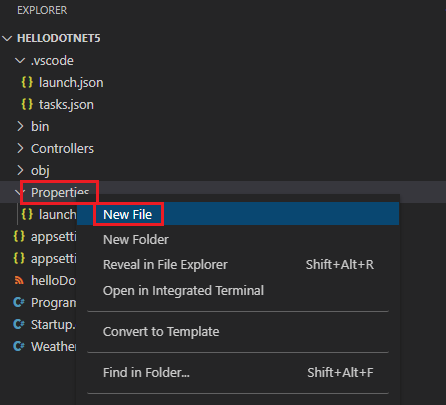
The endpoint (the location of the API) - <https://api.openweathermap.org>

The API KEY - 89a06e9380bd2a2021c62f01732929cc

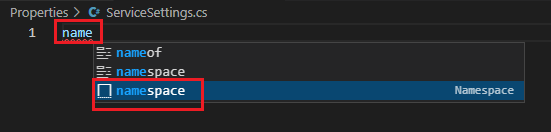
These parameters should not be hard coded so we don’t need to touch the application.

Create a class that represent that information in the project:

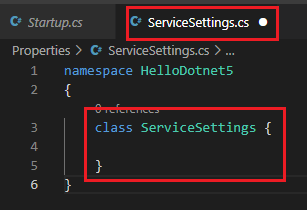
1. Properties > New File > **ServiceSettings.cs**



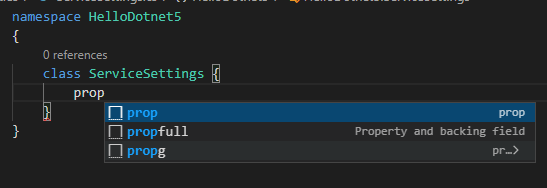
1. Put it in the same namespace as everything else which is **HelloDotnet5**

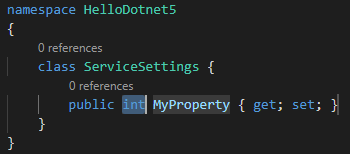


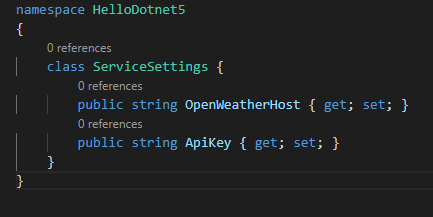
1. Do our class which is - **ServiceSettings**



1. Add a couple of properties by enter prop + TAB



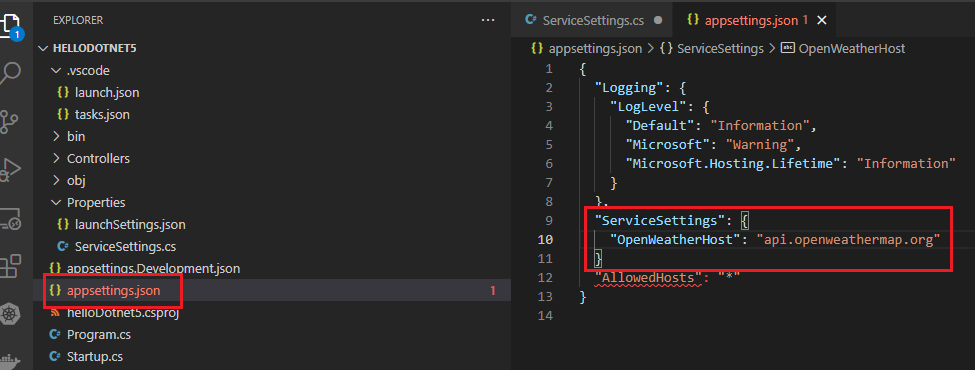




OpenWeatherHost property represent the host - <https://api.openweathermap.org>

ApiKey property represents the API KEY - 89a06e9380bd2a2021c62f01732929cc

1. We need to configure these parameters out of the code safety and for this we will use **appsettings.json** for **OpenWeatherHost** property:





appsettings.json – file settings to the project

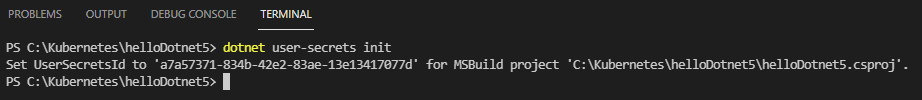
ServiceSettings – is the name of the class

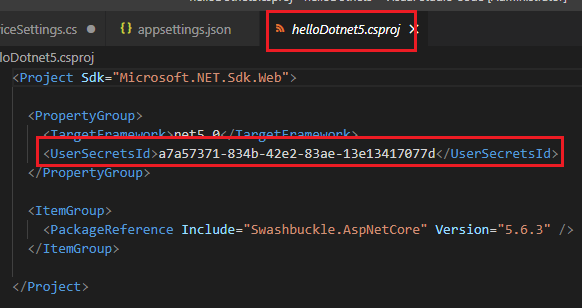
OpenWeatherHost – is the name of the property

api.openweathermap.org – is the URL value (host)

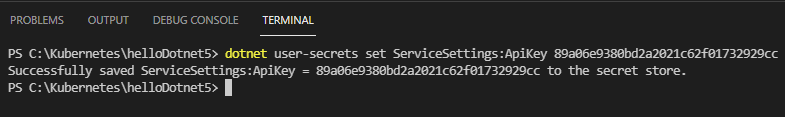
1. The ApiKey property should be in secret and not in appsettings.json

**dotnet user-secrets init**





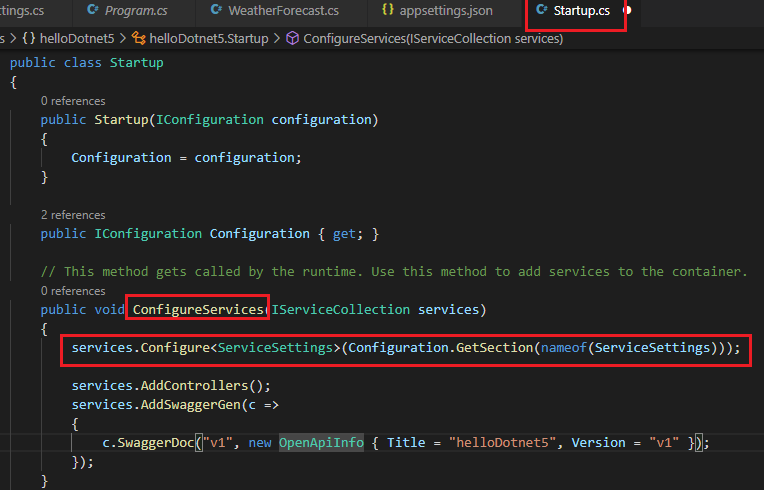
**dotnet user-secrets set ServiceSettings:ApiKey 89a06e9380bd2a2021c62f01732929cc**

****

The secret of the ApiKey value is stored somewhere in the box

1. Now we need to tell the application how to find and register these configurations (in appsettings.json and in secret) and for that we will use **Startup.cs** by adding this configuration code (dependency injection) so when the app is loaded it will find configuration settings in both appsettings.json and in secret store

**services.Configure<ServiceSettings>(Configuration.GetSection(nameof(ServiceSettings)));**



**services.Configure<ServiceSettings>(Configuration.GetSection(nameof(ServiceSettings)));**

**services.Configure** – **services** is a collection

<**ServiceSettings**> – is the class name that we created

(**Configuration. .GetSection** – **Configuration** is the first property in **Startup.cs**  “public IConfiguration **Configuration** { get; }”

**.GetSection(nameof(ServiceSettings))); - ServiceSettings** is the section that we declared in appsettings.json

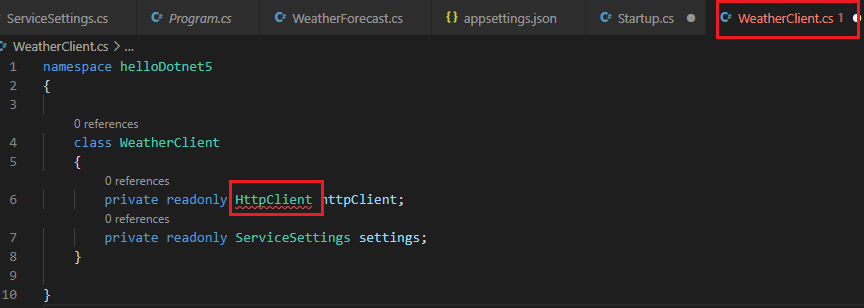
**services.Configure<classname>(Configuration.GetSection(nameof(section-in appsettings.json)));**

**By doing this (service instance) we can access the configuration settings anywhere we need in the application. When the app is loaded the Configuration engine is Statup.cs is going to find the settings in both appsettings.json and in secret with the dependency injection engine (Configuration engine) of .net5**

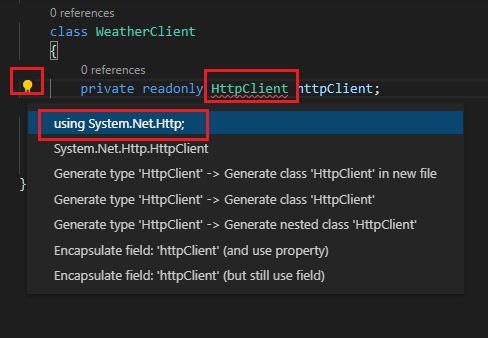
1. Now we need to write a code to access api. For that we will create a service client supported by “**IHttpClient**” object in .net5.

For that we will create a file called **WeatherClient.cs**

1. add to properties (objects )as below

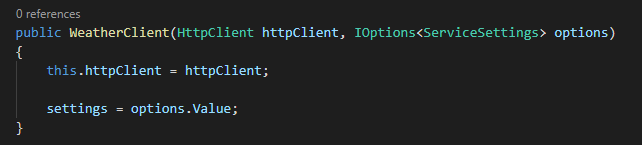


Add reference to HttpClient object:



**HttpClient** is a key object in .NET5 to access external endpoints like we need to call now

**ServiceSettings** is the service that declared for the **ServiceSettings.cs** class we created that holds the setting information to the endpoint (host).



Add constructor – in this case our client. We injecting an instance of **HttpClient** (**HttpClient httpClient**) that’s somehow it constructed in startup and it should be available here and also the **IOptions** object, this is how we are able to receive the configuration that has been registered in stratup.cs –

**services.Configure<ServiceSettings>(Configuration.GetSection(nameof(ServiceSettings)));** (done above)

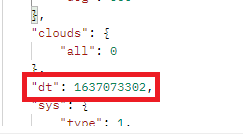
Then we grab both of them into properties **this.httpClient = HttpClient and settings = options.Value;**

**Options** has a property calledValue that actually has the service settings we care about.

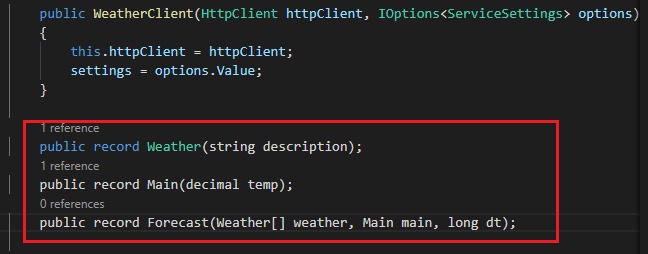
1. Now will look at postman and select the objects that we want to send as a response when we send request to the external host api.openweathermap.org:

https://api.openweathermap.org/data/2.5/weather?q=haifa&appid=89a06e9380bd2a2021c62f01732929cc&units=metric



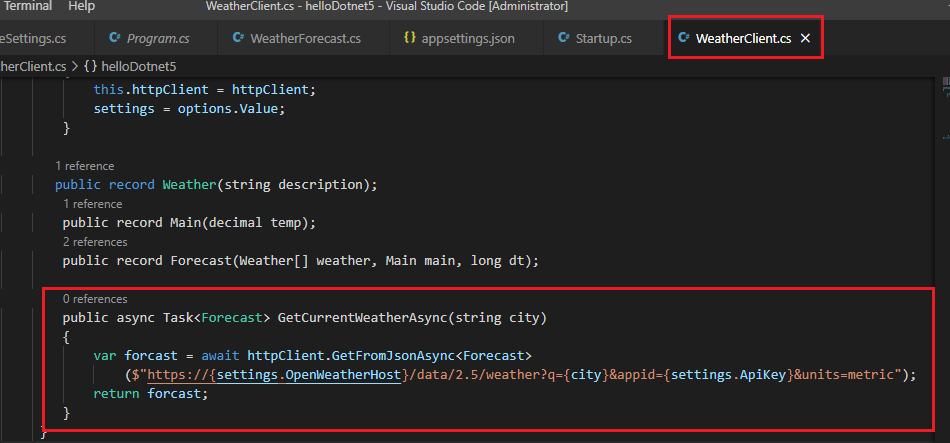


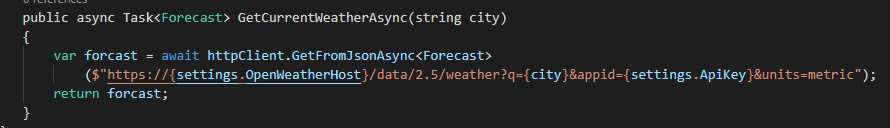
We choose **weather** object with **description** inside and **main** object with **temp** and **dt** object and for that we will use record object that will grab the objects that we wants:



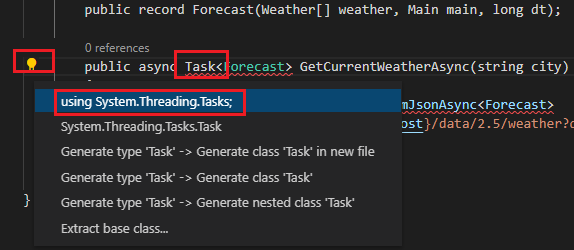
The last record type **Forecast** is an overall record that we are going to use and contains all objects we need from the external host api which are **weather** collection, **main** and **dt** (datetime).

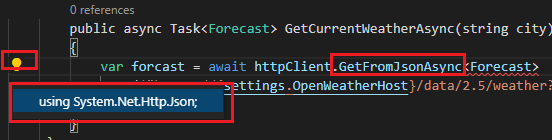
1. Now we need to call and retrieve the weather information based on some city and for that we will use a new method as below:





We need to add missing reference to namespace for **Task** and **GetFromJsonAsync**





In the command we copy and paste the URL that we tested in postman

https://api.openweathermap.org/data/2.5/weather?q=haifa&appid=89a06e9380bd2a2021c62f01732929cc&units=metric

and do string interpolation with **($)** by replacing the hard coded settings with settings that holds the configuration setting of the service **URL** and **ApiKey** and the **city** argument:

public **async** Task<**Forecast**> GetCurrentWeatherAsync(string city)

{

var forcast = **await** httpClient.GetFromJsonAsync<Forecast>

(**$**"https://**{settings.OpenWeatherHost}/**data/2.5/weather?q=**{city}**&appid=**{settings.ApiKey}**&units=metric");

return forcast;

}

For **async** call we always need to use **await** command

**<Forecast>** is used as the record type we created for retrieving information from the external host

1. Now this **WeatherClient** class needs to be registered into the series of services available in our application and for that we will go back to **Startup.cs** and in the **ConfigureServices** method we will add the command below so **WeatherClient** class will be registered into the services collection and .NET5 will know that it needs to inject an HttpClient object into that class so that we can do the actual call to the external service.

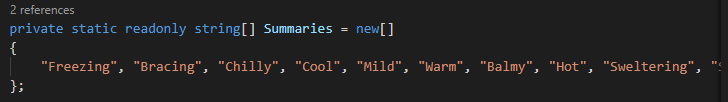


services.AddHttpClient<WeatherClient>();

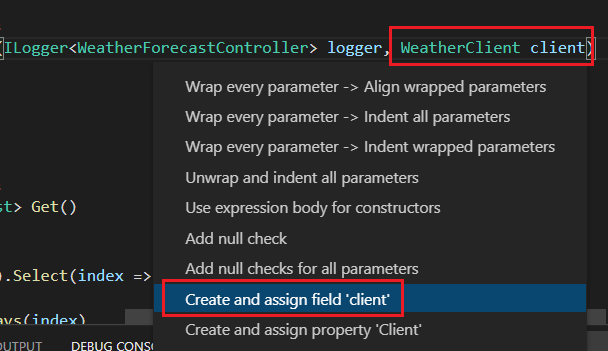
1. Now we need to change the WeatherForecastController.cs so the controller will act as a real

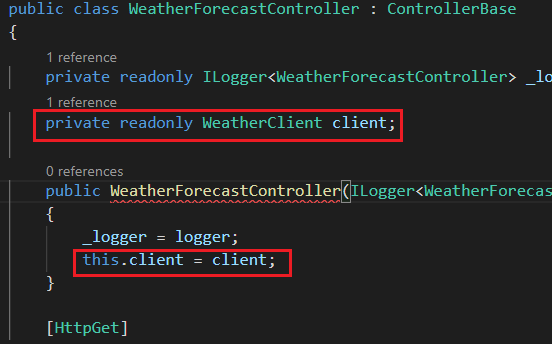
Controller as it will get the city parameter and will return the weather of that city.

First we delete the random Summaries.

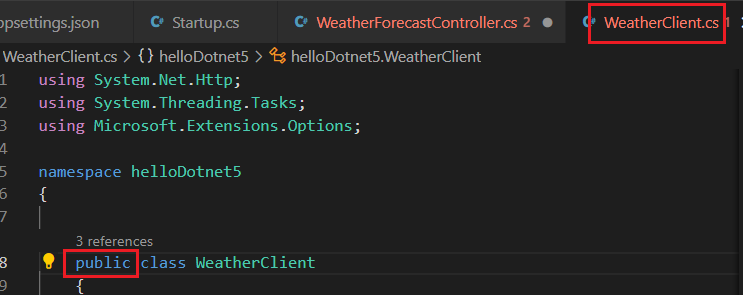


Then inject the WeatherClient class into the controller by adding **WeatherClient client** argument to the **WeatherForecastController.** After the clientpress **CTRL+.** To **create and assign filed client**:

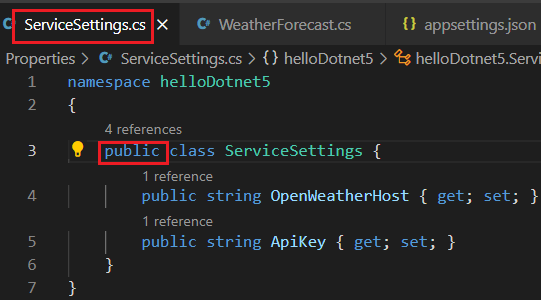




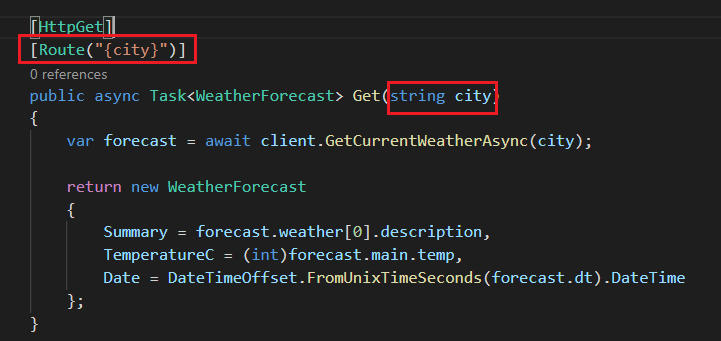
Need to add public to the **WeatherClient** class



And also to **ServiceSettings** class



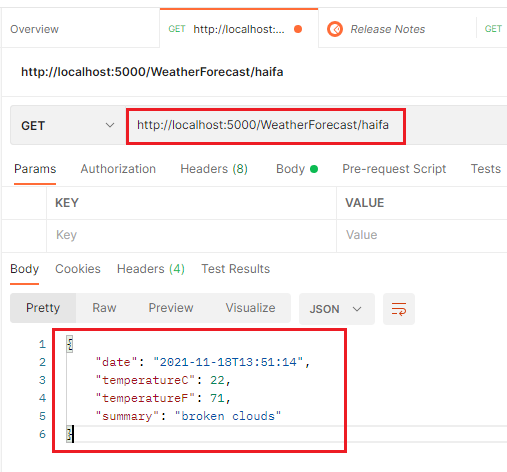
Now we change the code as below:



Postman - [http://localhost:5000/**WeatherForecast**/**haifa**](http://localhost:5000/WeatherForecast/haifa)

**WeatherForecast** comes from WeatherForecastController

**Haifa** comes from city parameter in the URL Route.

****