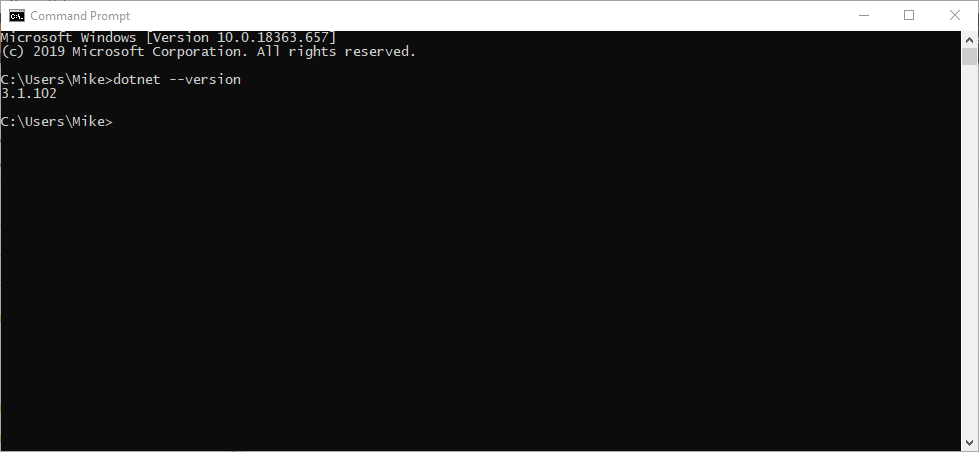
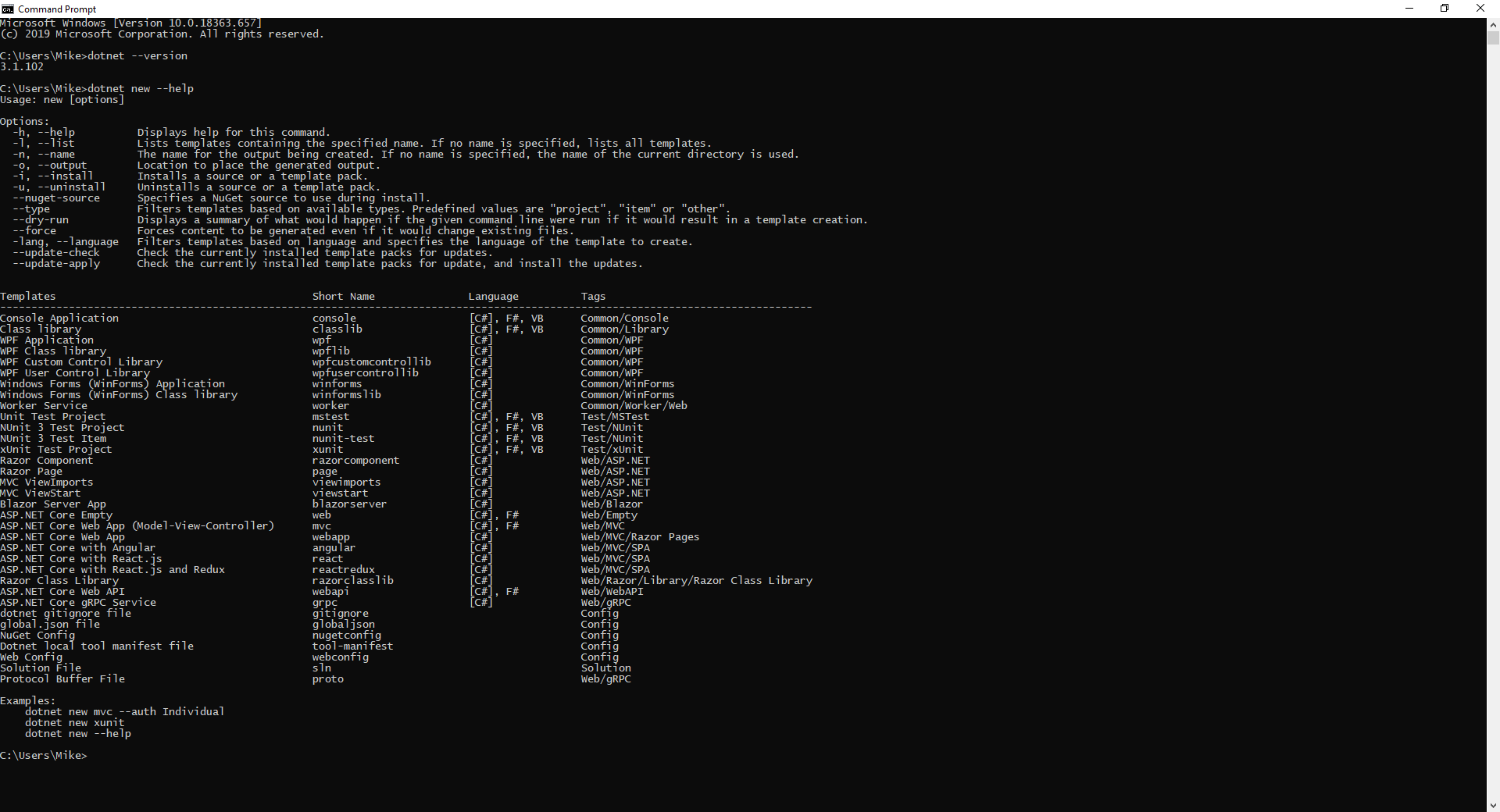
**Exercise 1 - Install 3.1 .NET Core SDK**

1. Install .NET Core 3.1 **SDK** from [https://dotnet.microsoft.com/download (Links to an external site.)](https://dotnet.microsoft.com/download)
2. Run **dotnet --version** to make sure you have the dotnet CLI installed. You may need to close and reopen a command/shell window first.



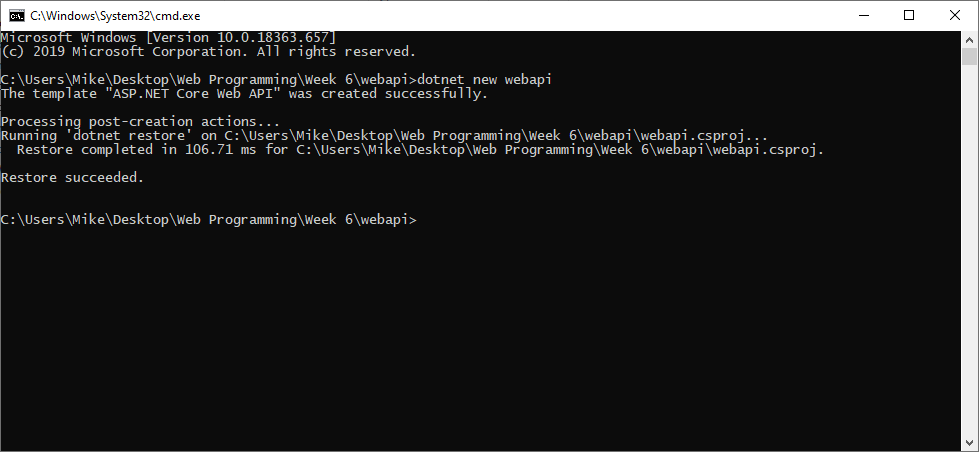
1. Run **dotnet new --help** to see all of the types of things you can create using the CLI.



**Exercise 2 - Create the sample .NET Core WebApi (REST Service)**

1. Make a week 6 lab folder by coping your week 5 folder. This will take a bit of time as the node\_module folder contents is quite large.
2. In a terminal/command window change to the week 6 lab folder you created in step 1.  Create a folder named **webapi**.
3. Change to the **webapi** folder and create the sample WebApi/REST Service by running the command:

**dotnet new webapi**

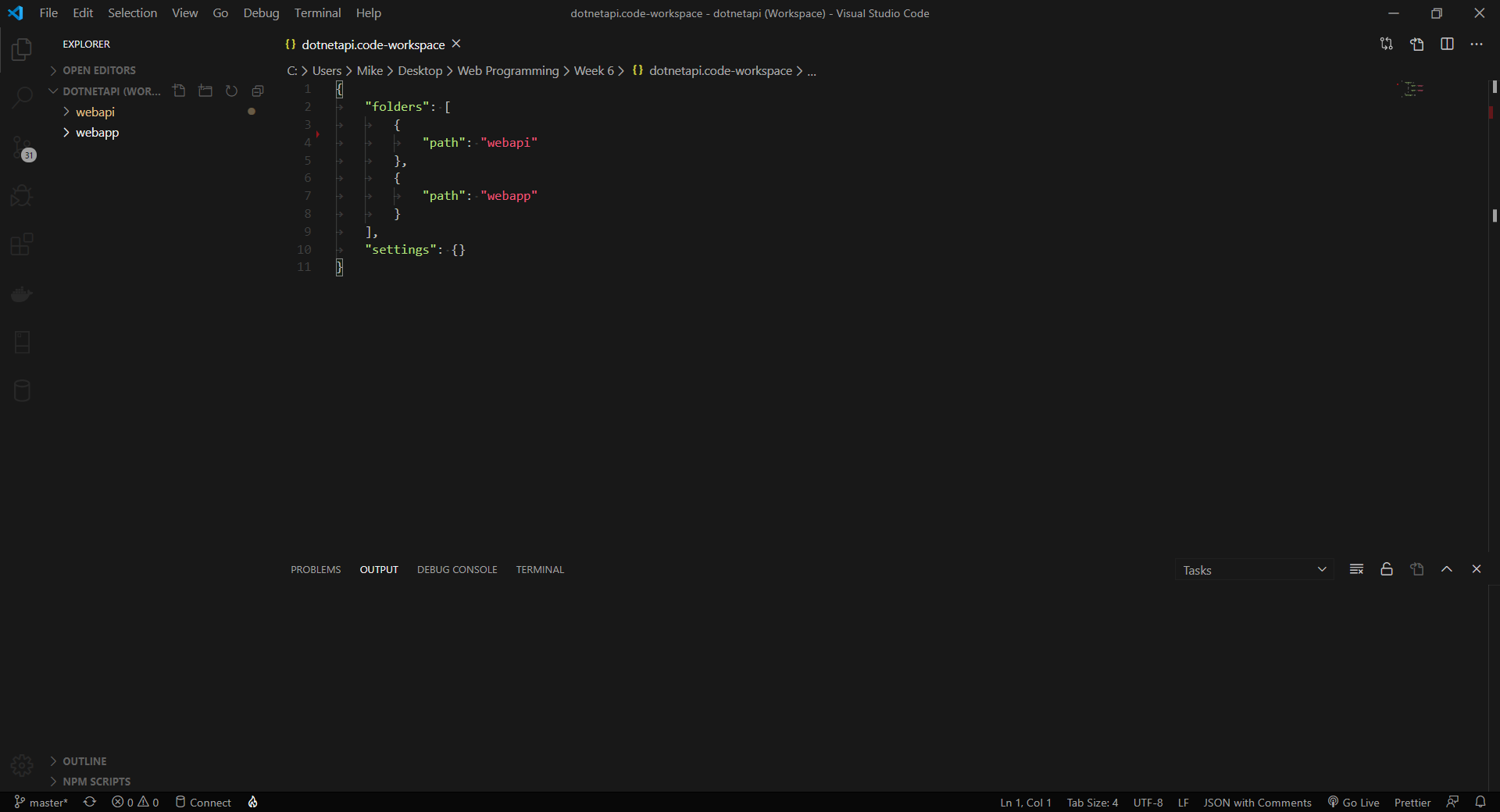


**Exercise 3 - Create a VSCode workspace to contain both the webapp and the webapi projects. (Optional if you are not using VSCode)**

1. Start up VSCode and open the week 6 lab folder.  Add the two folders (webapp and webapi) to a workspace by using the menu item "Add Folder to Workspace ..." shown below:

1. Save the workspace in the week6 folder using the menu item "Save Workspace As ..." shown below:

This will allow you to open both projects in VSCode by just opening the workspace file you just created.

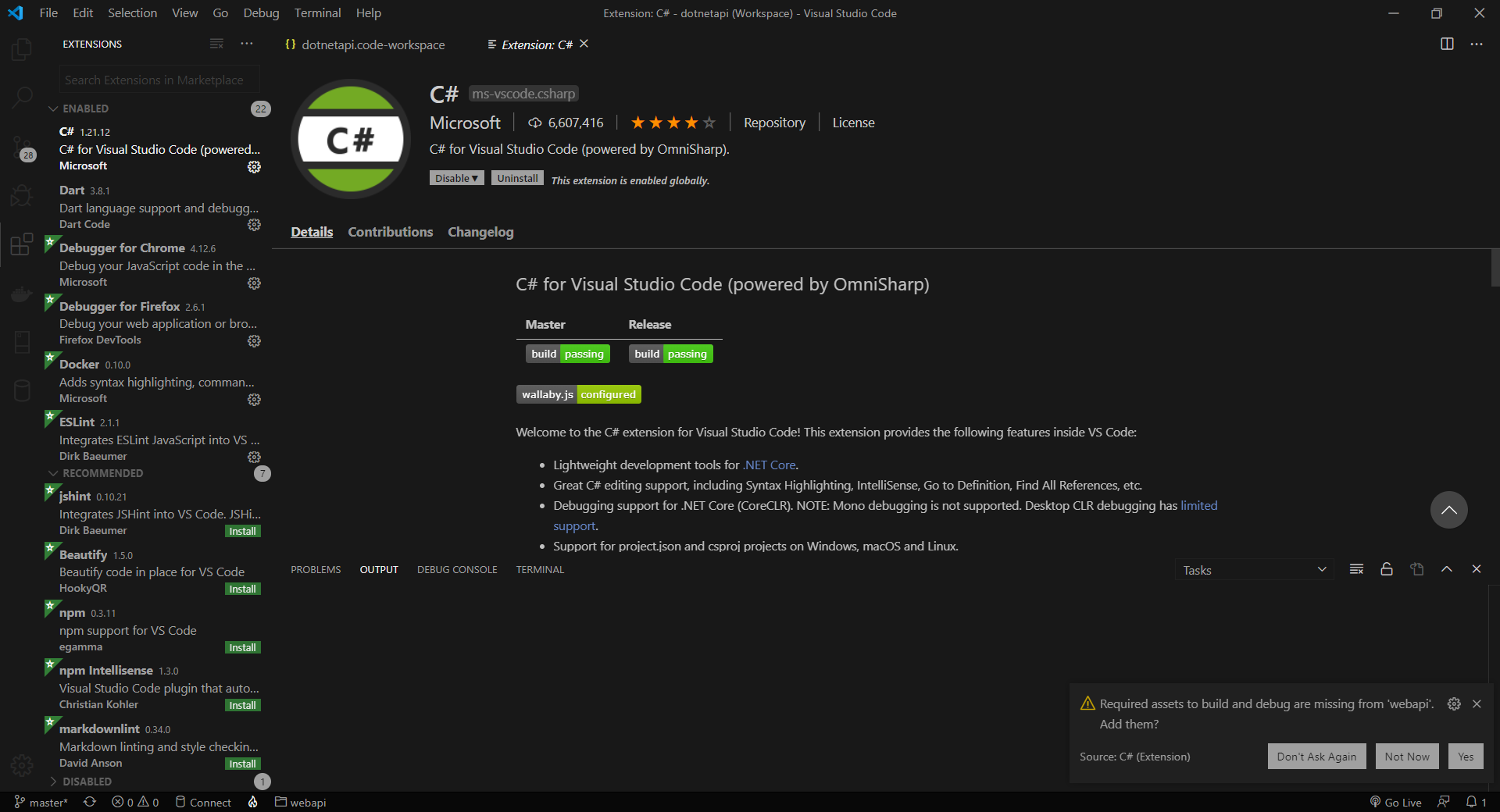


1. If you see the following dialog answer yes.

**Exercise 4 - Run the sample WebAPI/REST Service**

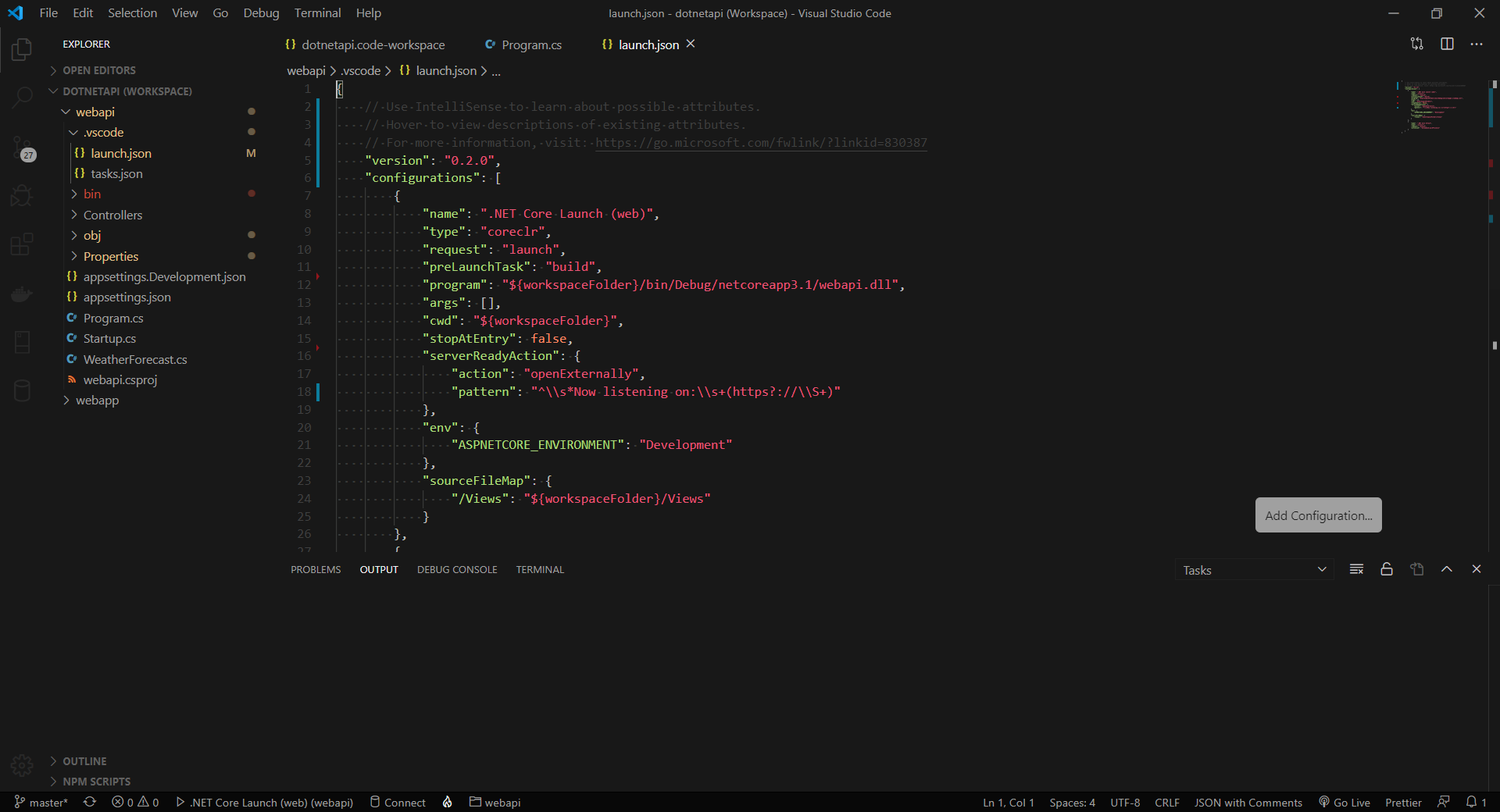
1. Install C# Extension for VS Code

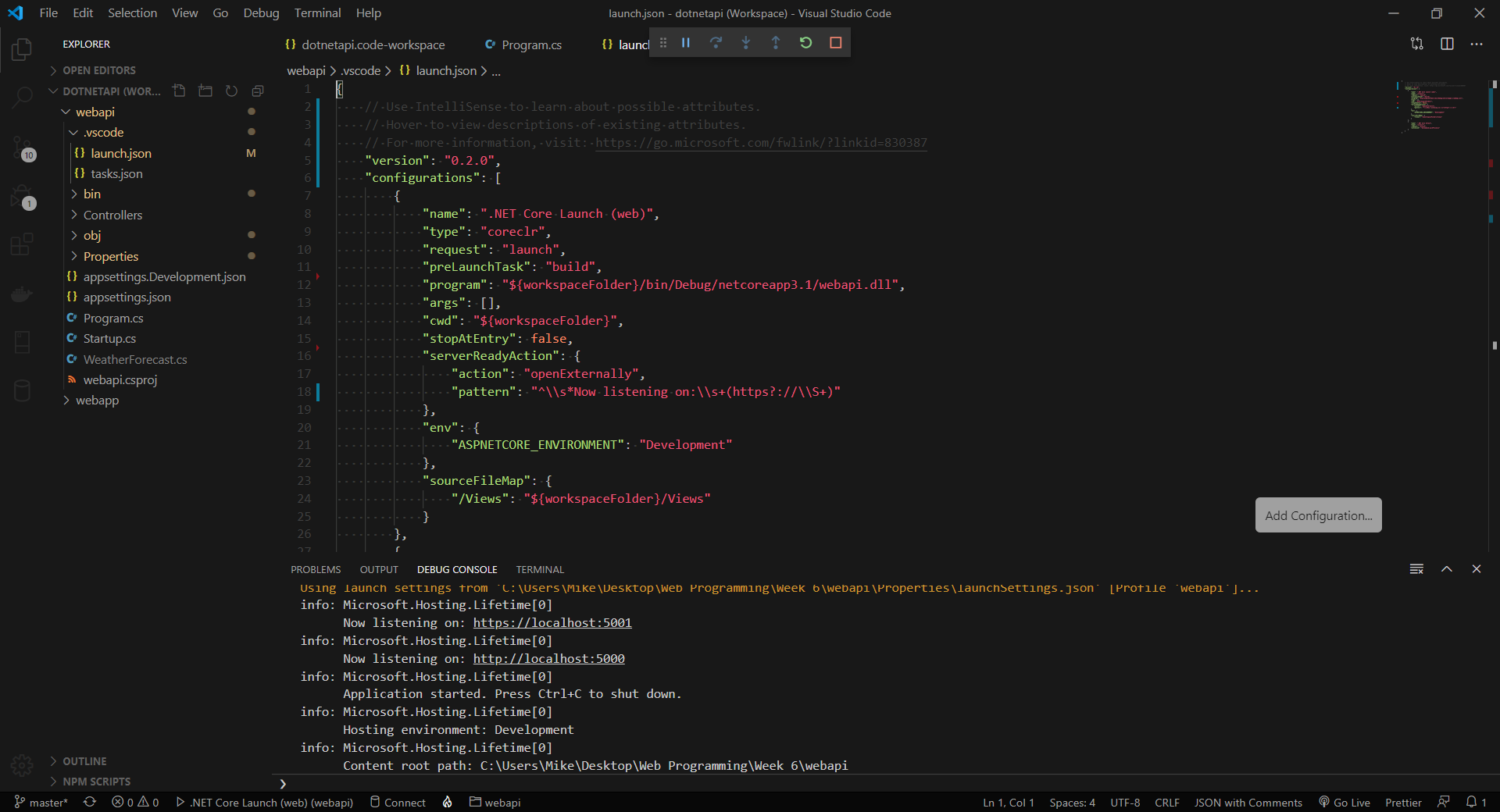
Install from [https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp (Links to an external site.)](https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp)



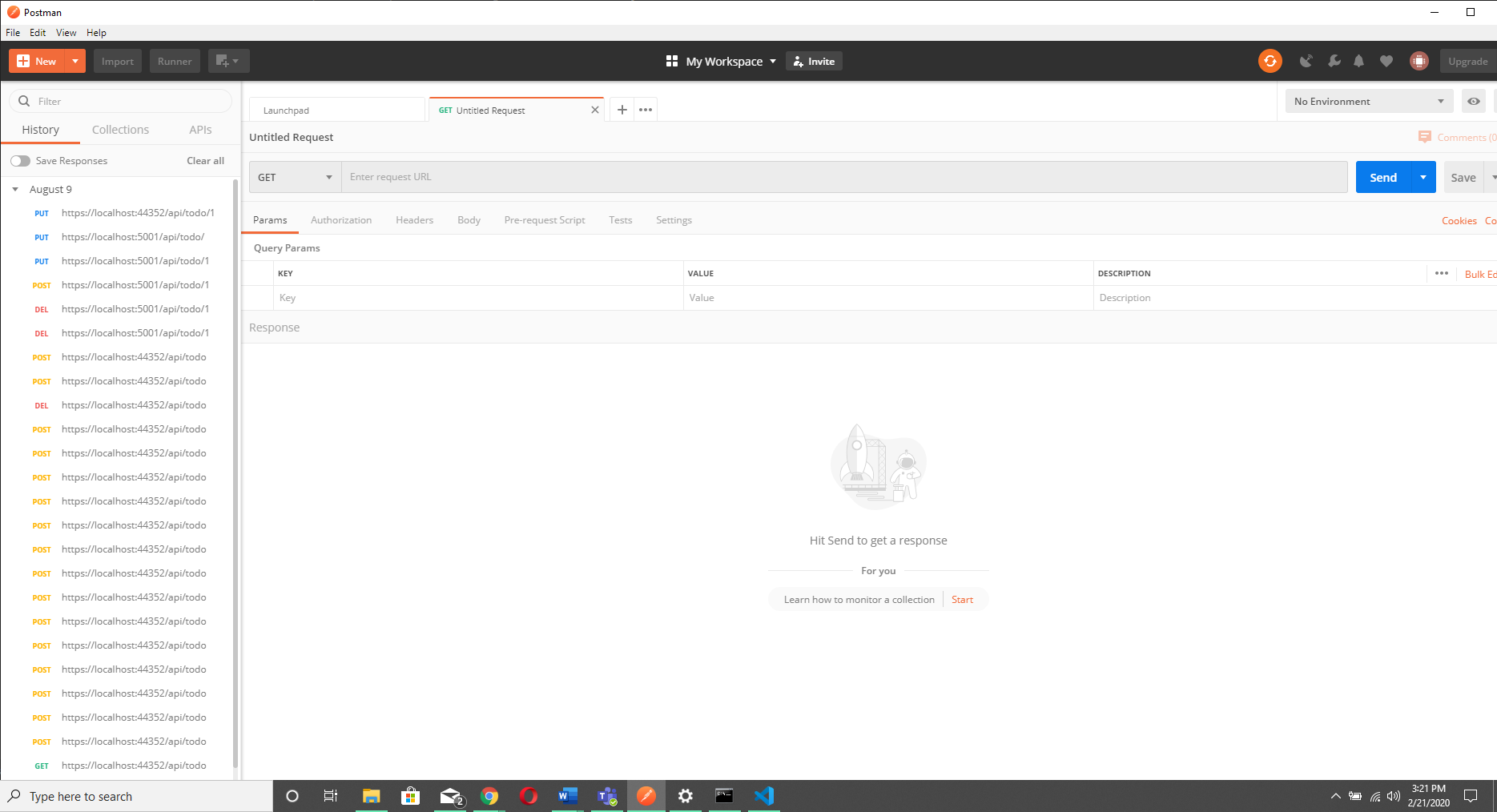
1. Select **webapi**from the VSCode Explorer and then **"Start Without Debugging"** from the **"Debug"** menu. This will display a list of runtime in a small menu.

Select **.NET Core**. This will then create a launch.json file to enable the launching of the application. This only needs to be done once. The webapi should then start. If not try selecting **"Start Without Debugging"** again. Once started successfully, a browser window should then appear.  You can ignore the brower window as we'll access the API in the next step using Postman.  If you scroll through the output in the VSCode "Debug Console" you'll see that the application is listening on two ports.  One for http (5000) and one for https (5001).  You'll use the http port below.

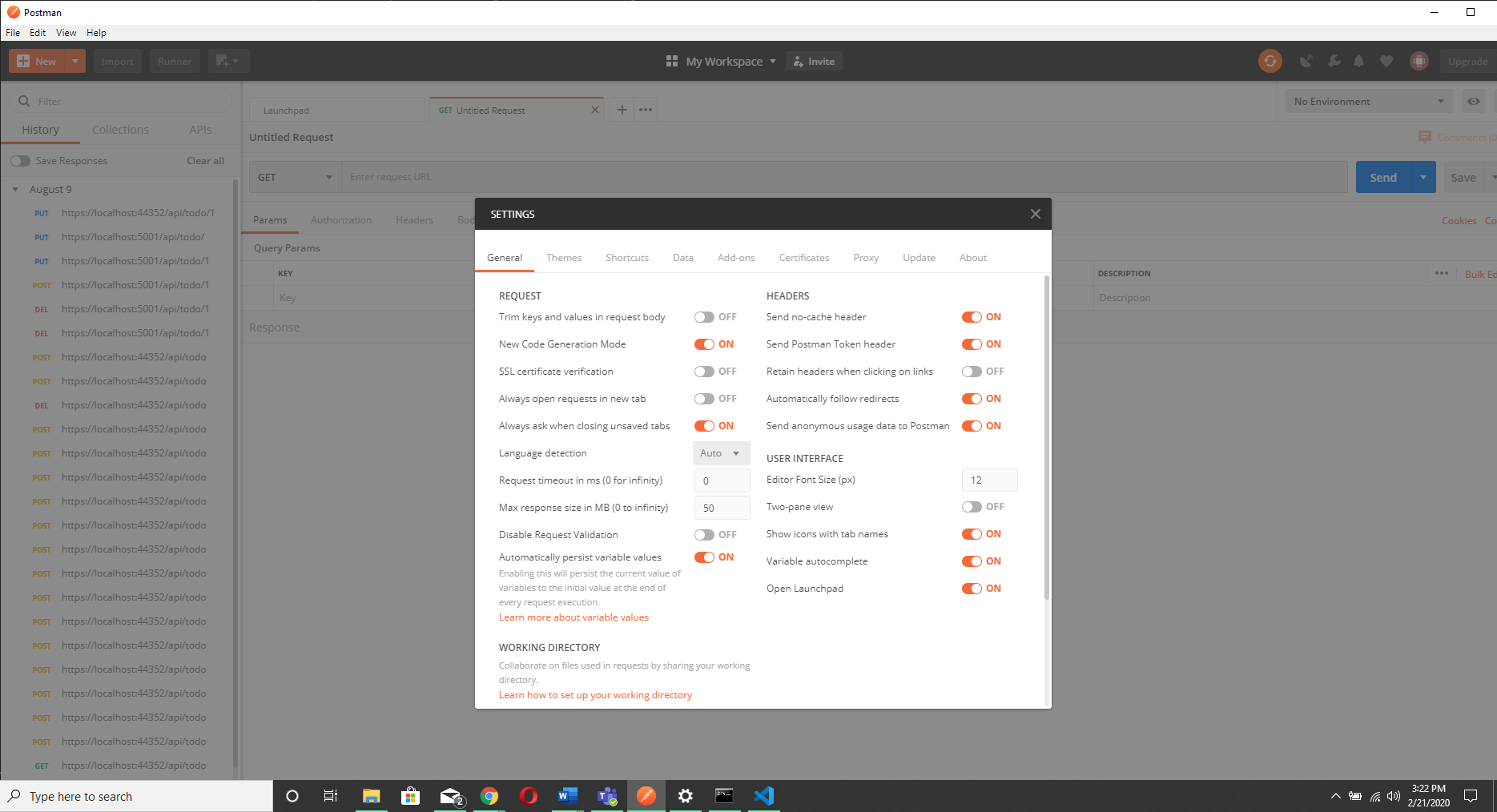


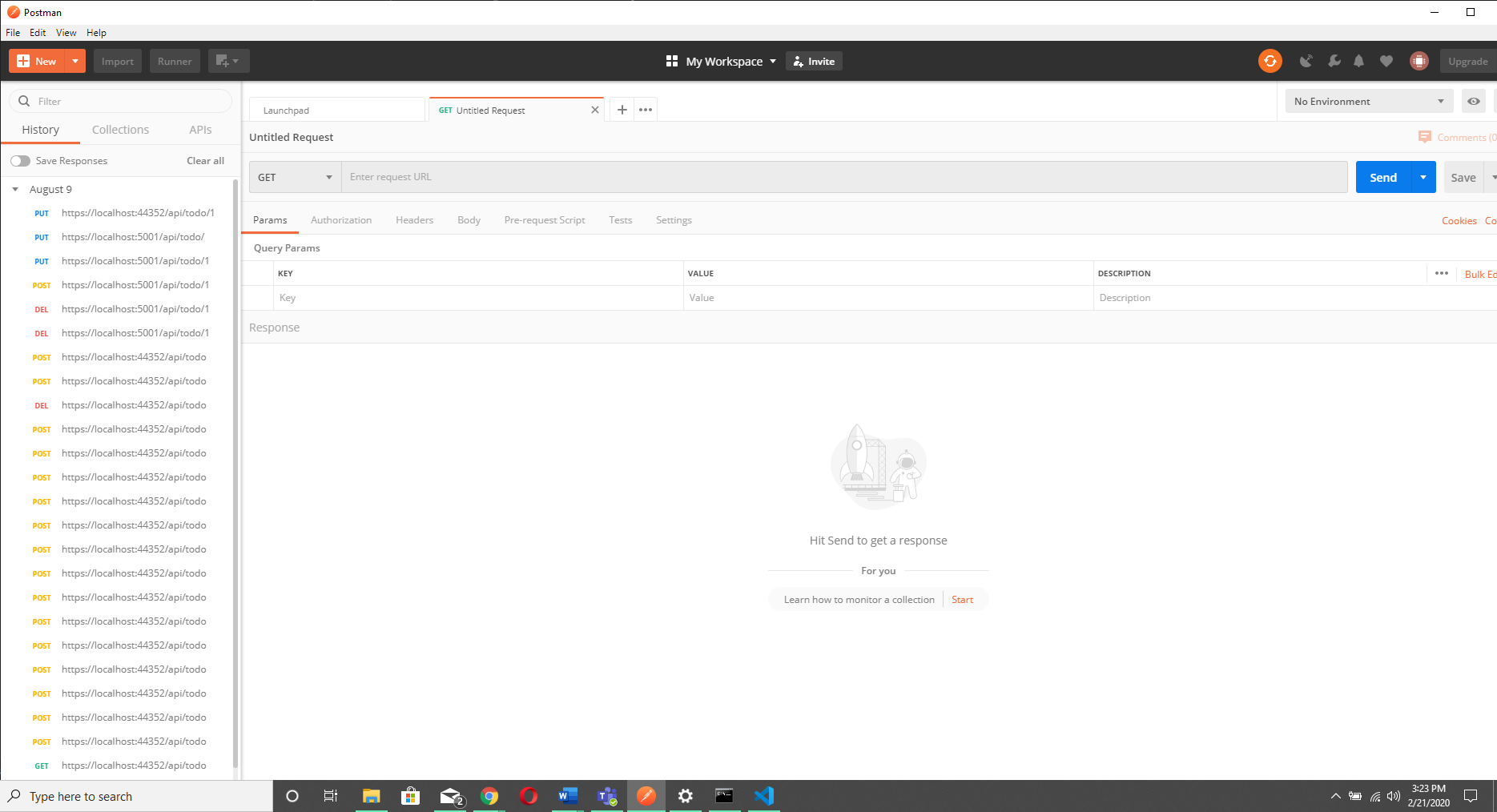


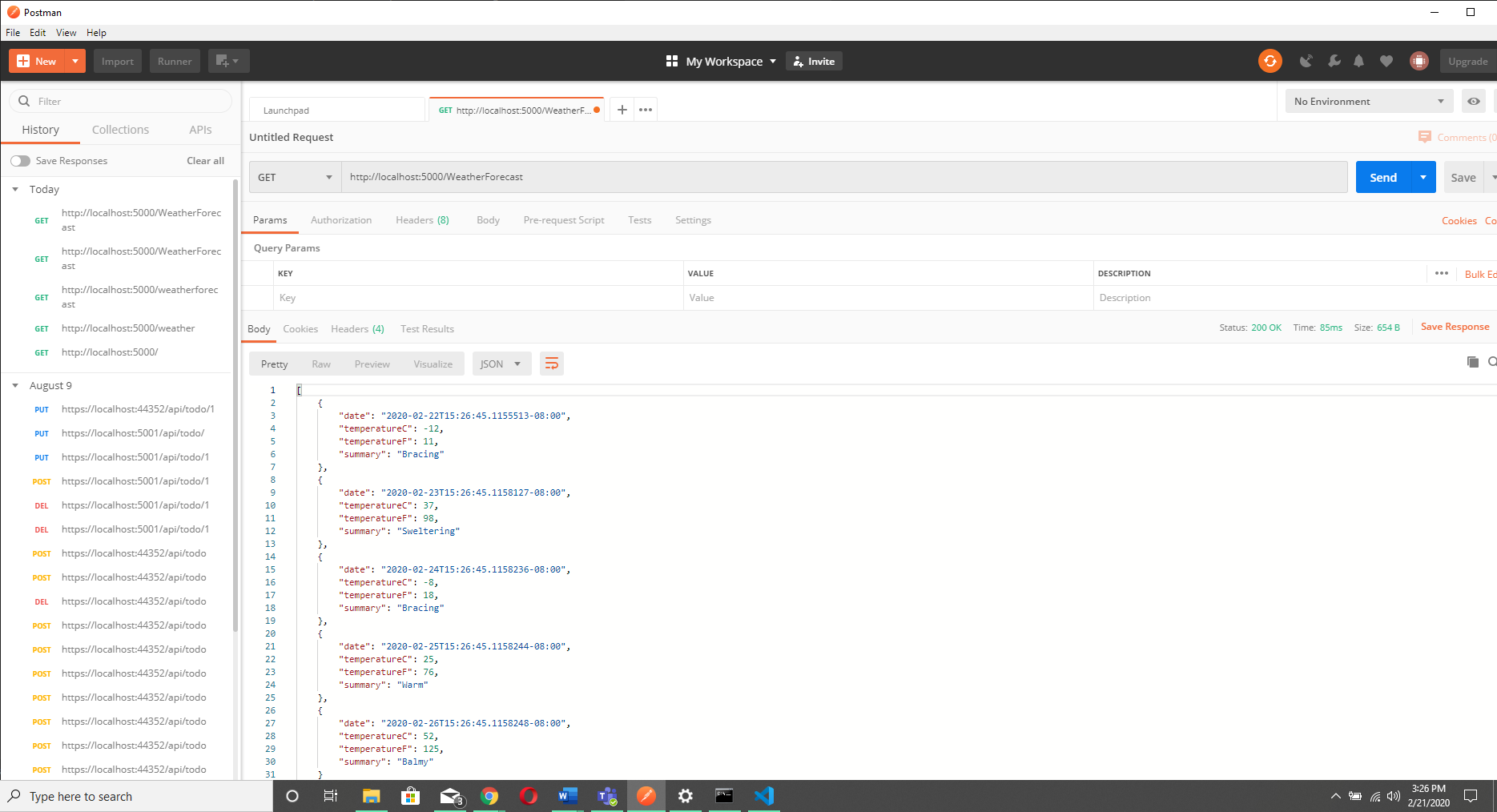
1. Install Postman from [https://www.getpostman.com/ (Links to an external site.)](https://www.getpostman.com/) if you haven't already installed it.  Startup Postman.



1. Configure it to turn off SSL certificate verification. If you don't do this Postman will display a security error. You get to the screen below by clicking on the wrench in the top right corner of Postman and selecting "Settings". Then make sure "SSL certificate verification" is OFF.

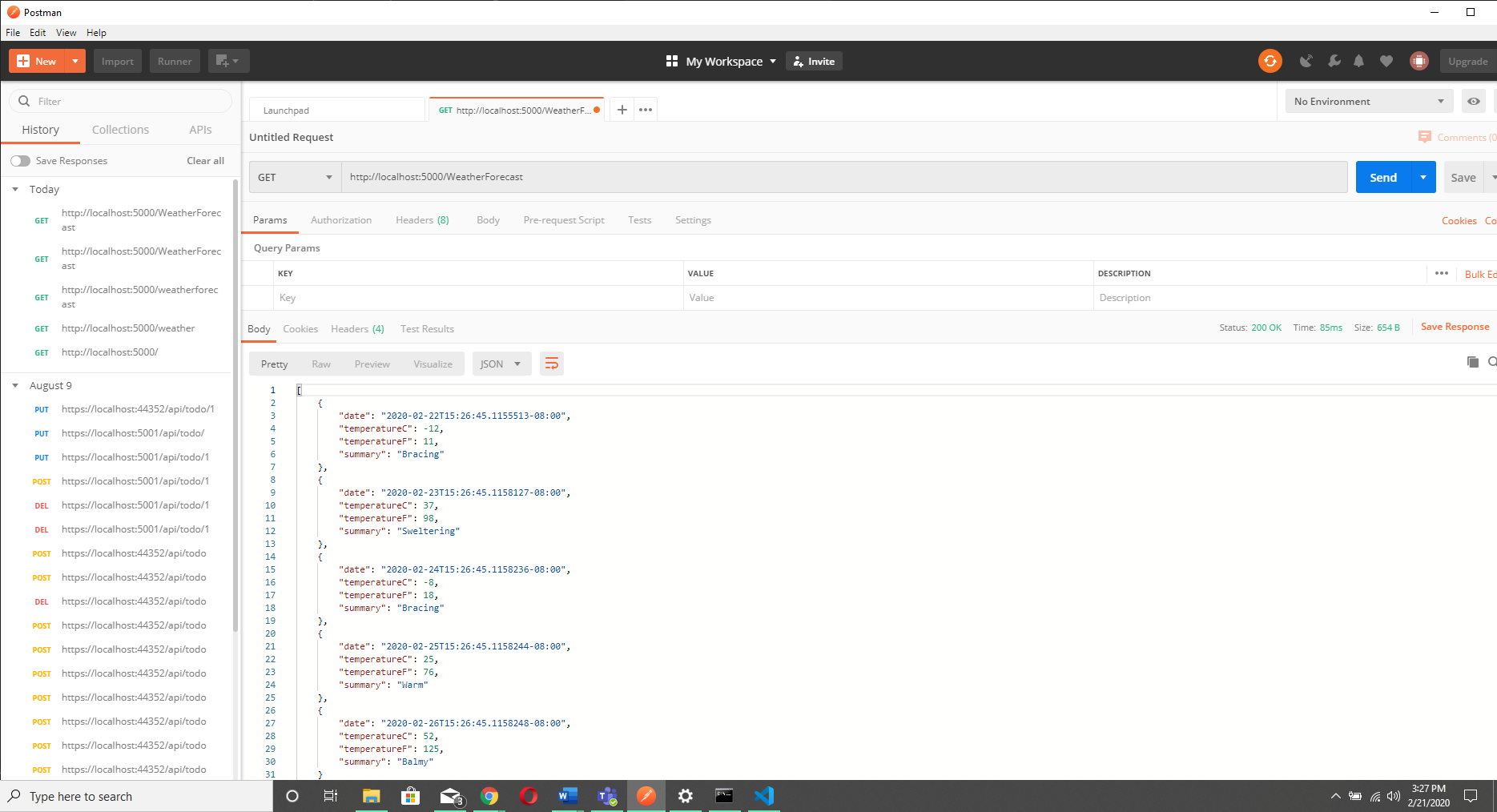


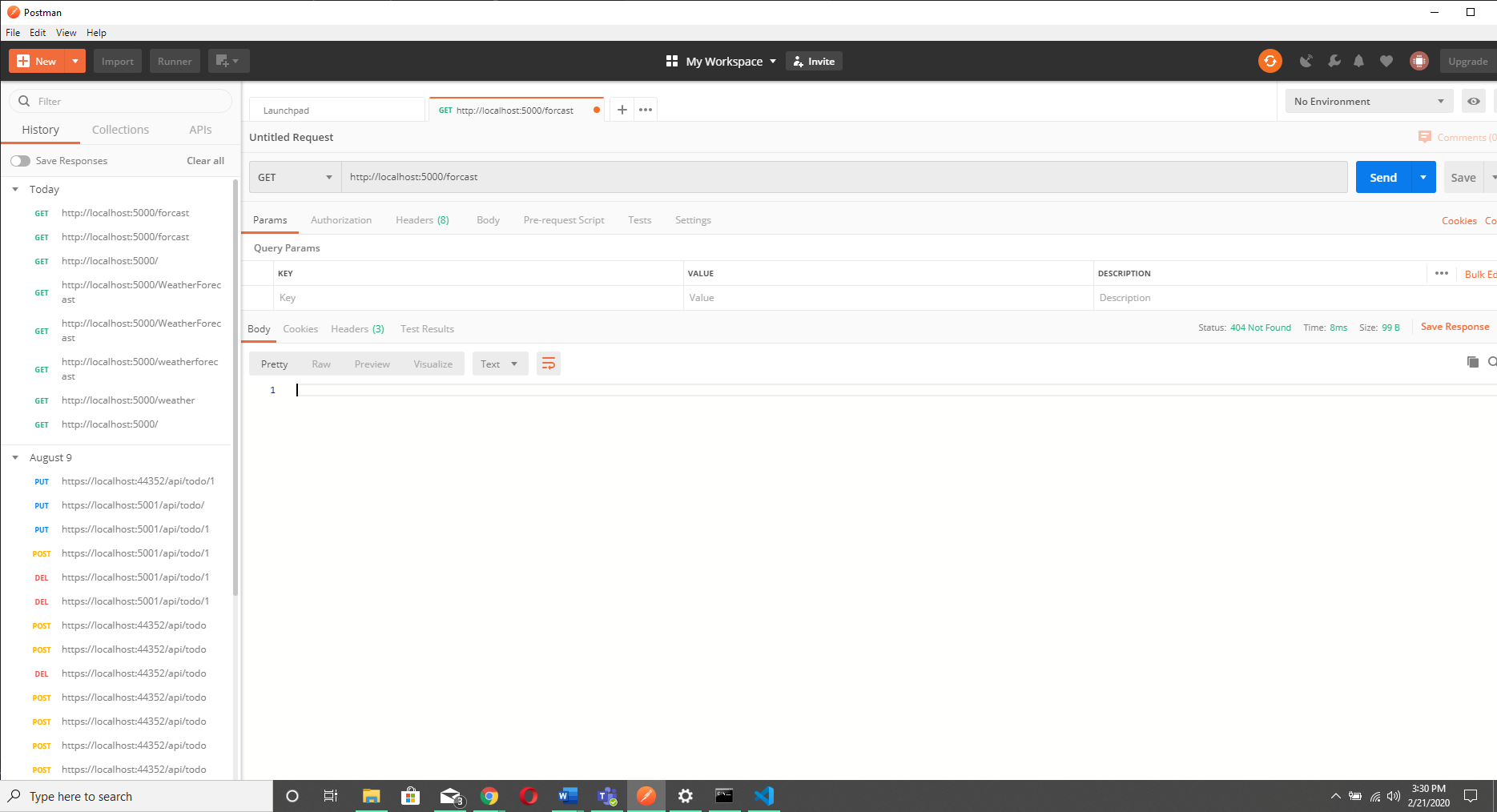
1. In Postman click on the "+" tab to get see the following displayed:  
     
   
2. Enter the URL as shown below:

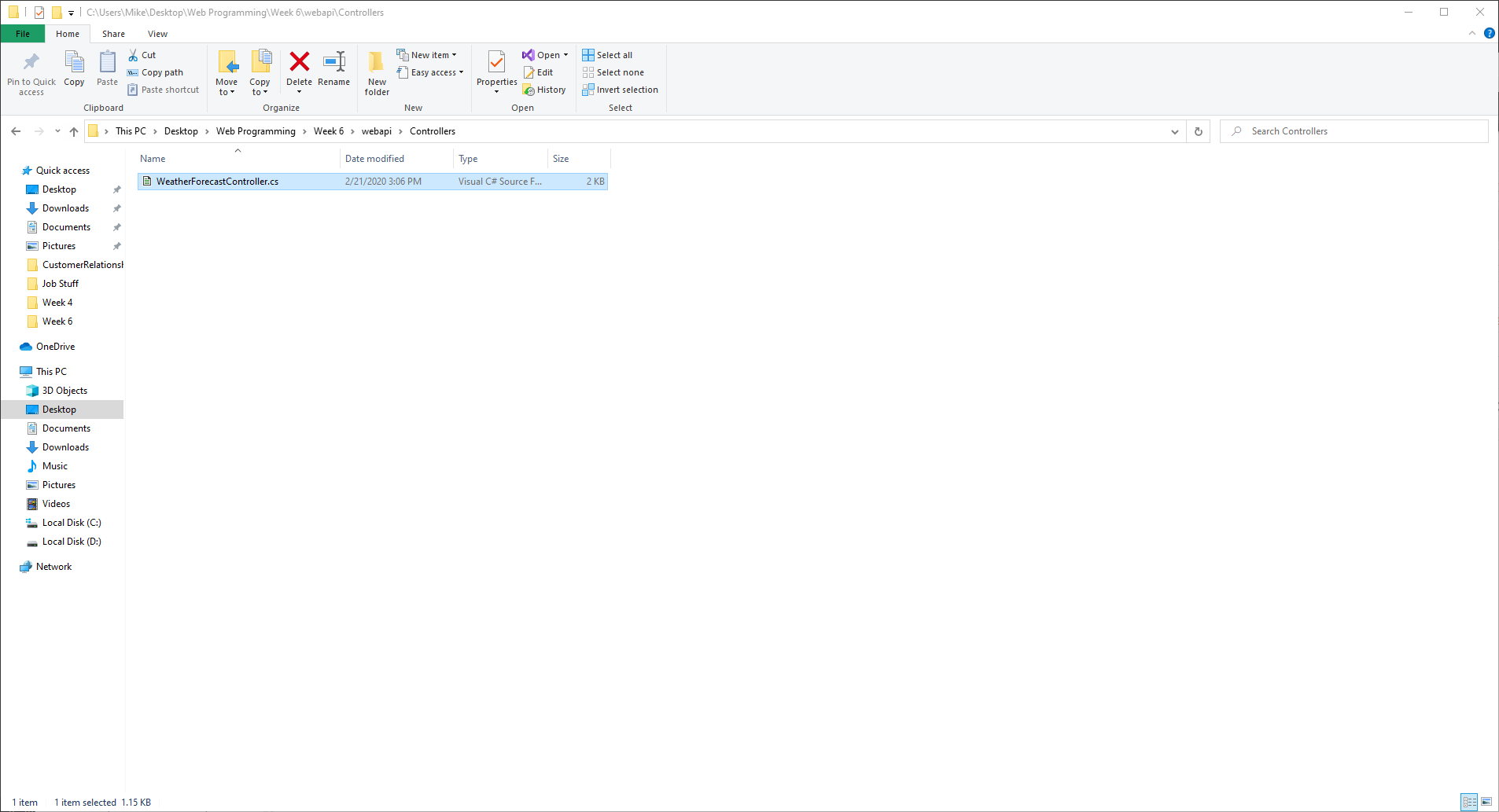
  
  
You can save this request in Postman if you want.

1. You should see the return values in the bottom part of Postman shown below:

See the status (200) returned by the REST api and displayed in Postman.  If you don't see the JSON output check the status returned.  If it is a 404 (not found) check the URL to make sure you entered it correctly.

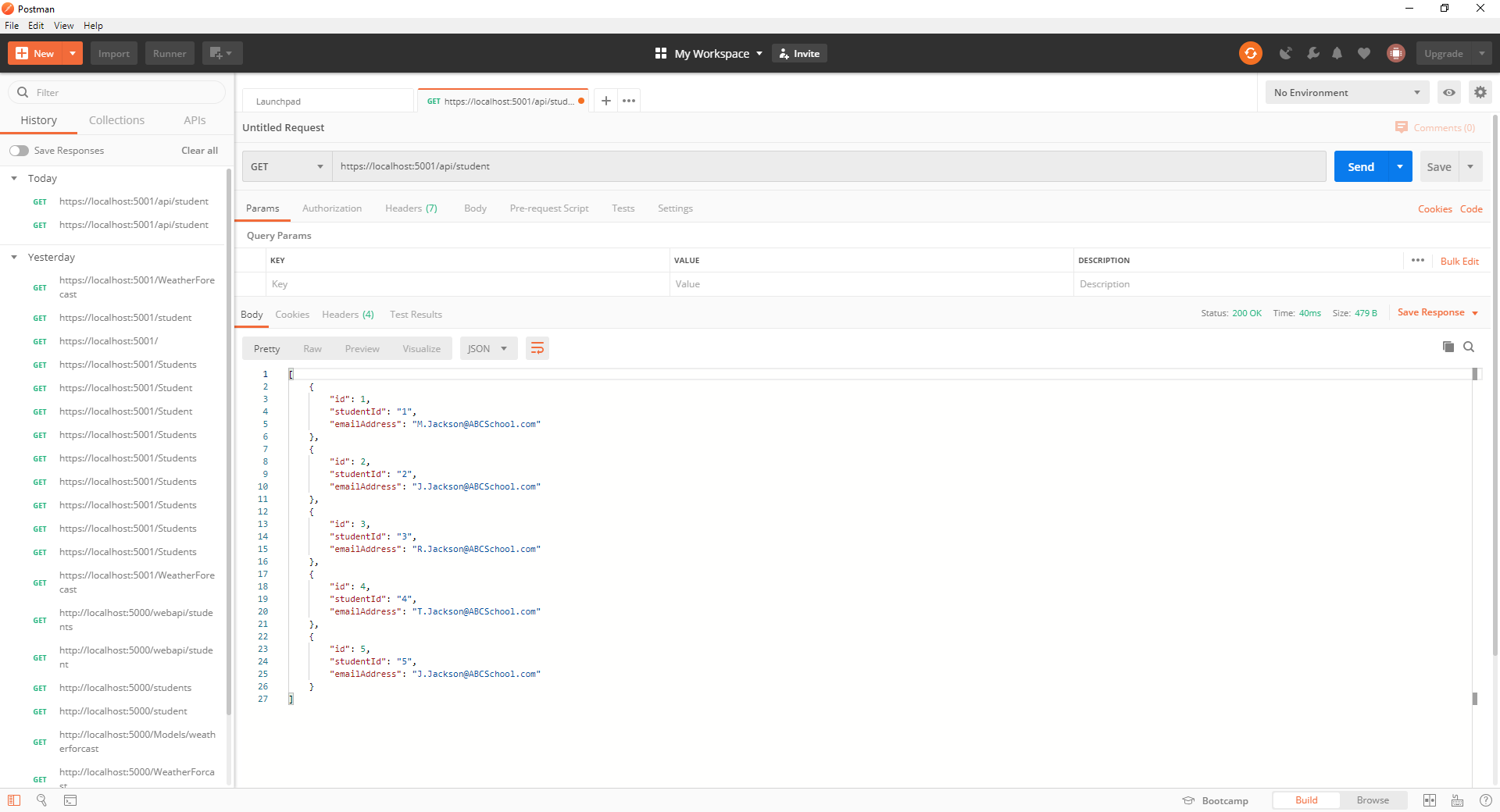


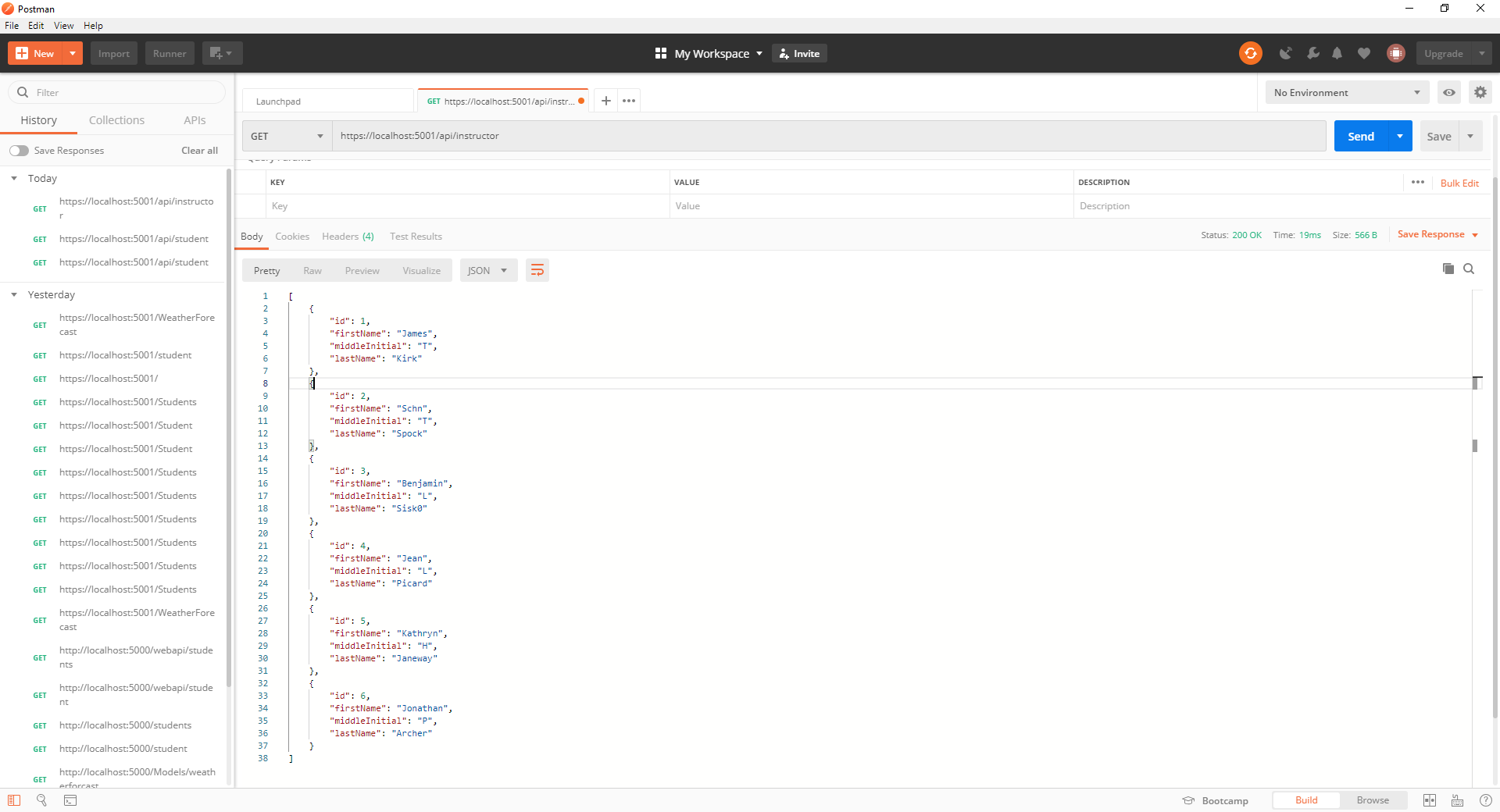
1. Change the URL to be invalid (e.g. remove the word "Weather") and resubmit the request.  You should get a 404.
2. 
3. Find the WeatherForecastController in the webapi project code where the output is being generated.



**Exercise 5 - Create the controllers for your API/REST service**

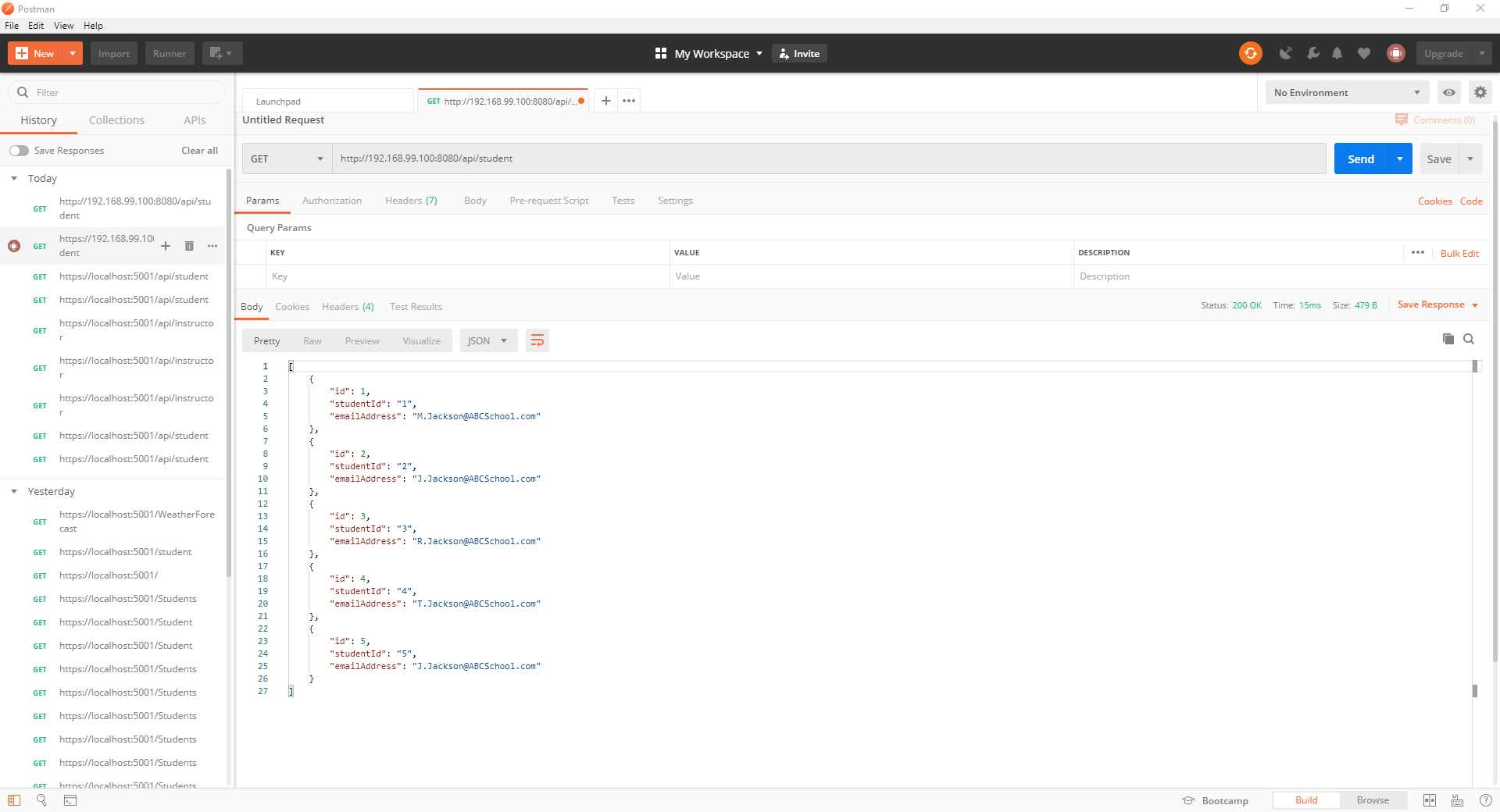
1. In the **webapi** project, create a controller like the **WeatherForecastController**for both students and instructors. You only need to include the "GET" method for now.  Don'f forget to change the class names accordingly. You can remove the WeatherForecastController if you wish.
2. Look at the example Student Controller in my GitHub repository and follow the same pattern for returning students and instructors from your two controllers.
3. Create a model for both student and instructor using the same properties you've used in previous labs. You can use my student model as an example.
4. Create an "in memory" database like the one in my example to "store" your student and instructor data. Look at the folder **week6/webapi/Database** in my example.
5. Try accessing your API from Postman to retrieve both students and instructors much like you did in exercise 4 for the weather forecast API.  **Include a screenshot of both the student and instructor JSON output shown in Postman.**

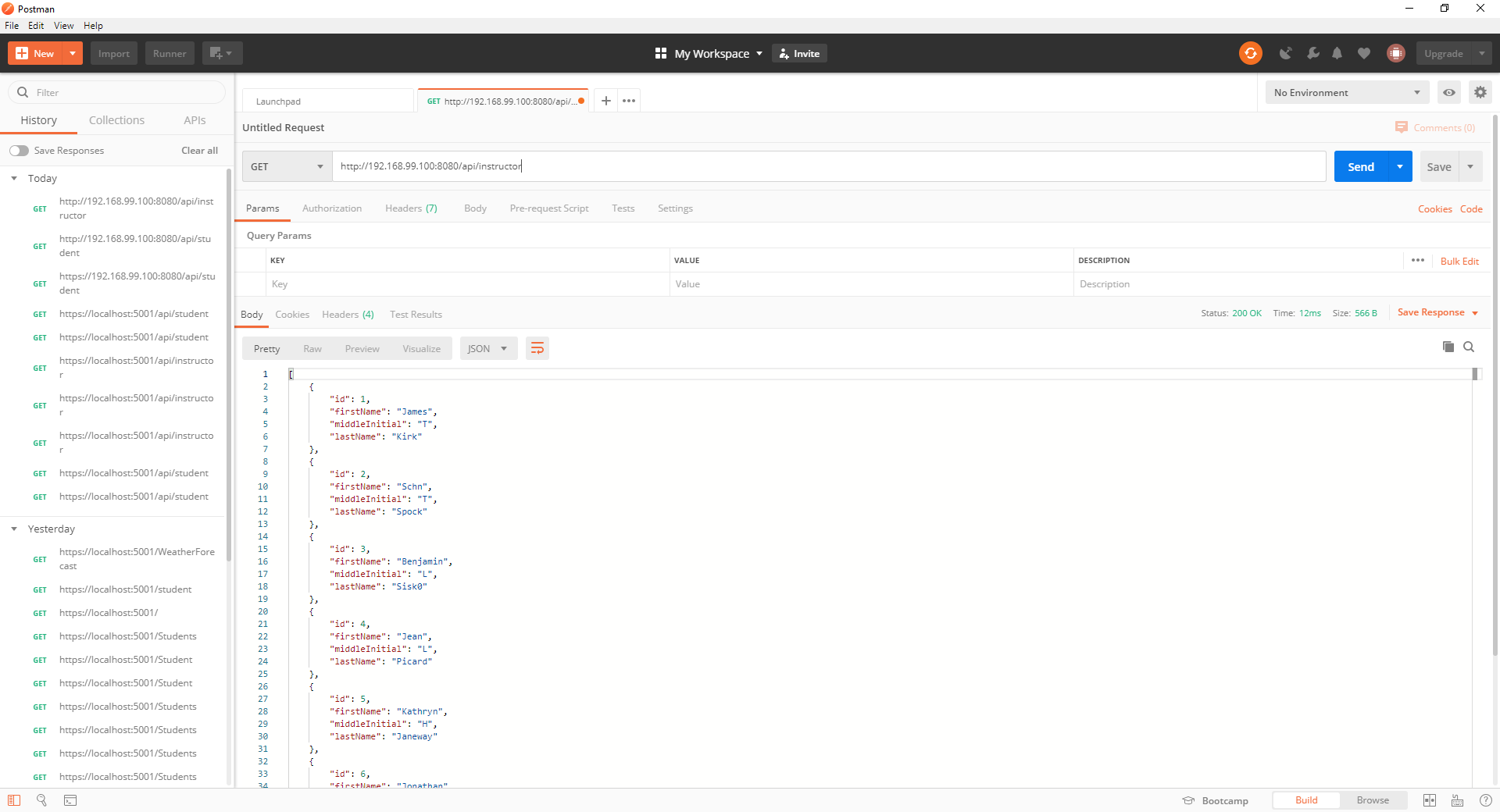




**Exercise 6 - Build and run your Webapi in a Docker container**

1. Copy the Dockerfile from my GitHub repository webapi folder to your webapi folder.  Examine the Dockerfile to see how the .NET Core app Docker image is built.  This is a two stage Docker file.   The first stage builds the application with the .NET Core SDK.  The second stage takes the output of the first stage and builds the final container image with the combination of the built .NET Core application from stage one and the .NET Core runtime.
2. Copy the .gitignore and .dockerignore files from my GitHub repository webapi folder to your webapi folder as well.
3. Build the container image and run the container like you have in previous labs.  
     
   **docker run -d --name [name your container] -p 8080:5000 [your container image name]**
4. Try accessing the student and instructor APIs at [http://localhost:8080/ (Links to an external site.)](http://localhost:8080/).  You should be able to see the same JSON output you saw in exercise 5.





**Push your Docker images to Docker Hub and push your code to GitHub.  Don't forget your screenshot document.**[(Links to an external site.)](http://www.programmableweb.com/)

[Previous](https://oit.instructure.com/courses/5294/modules/items/271445)[Next](https://oit.instructure.com/courses/5294/modules/items/271449)