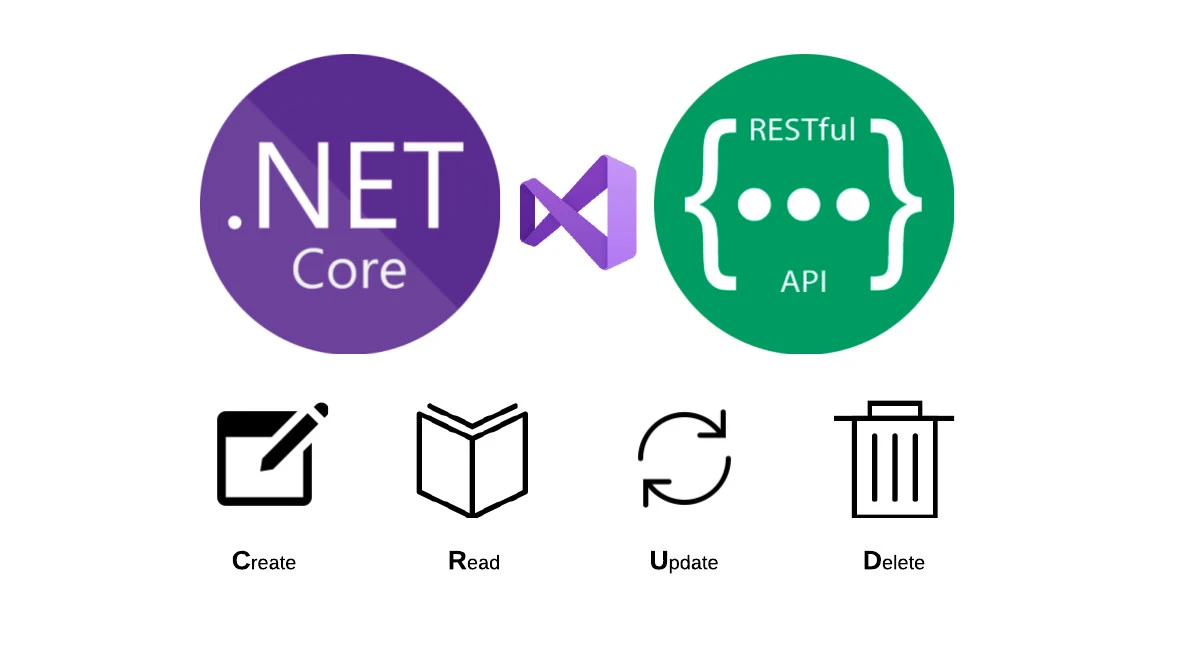
## 



Create a Web API with ASP.Net Core

Name : Dut Phearak

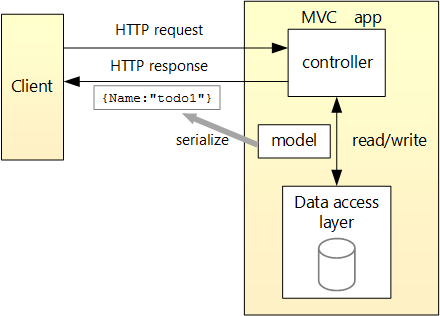
Subject : Web Application

Room : E6

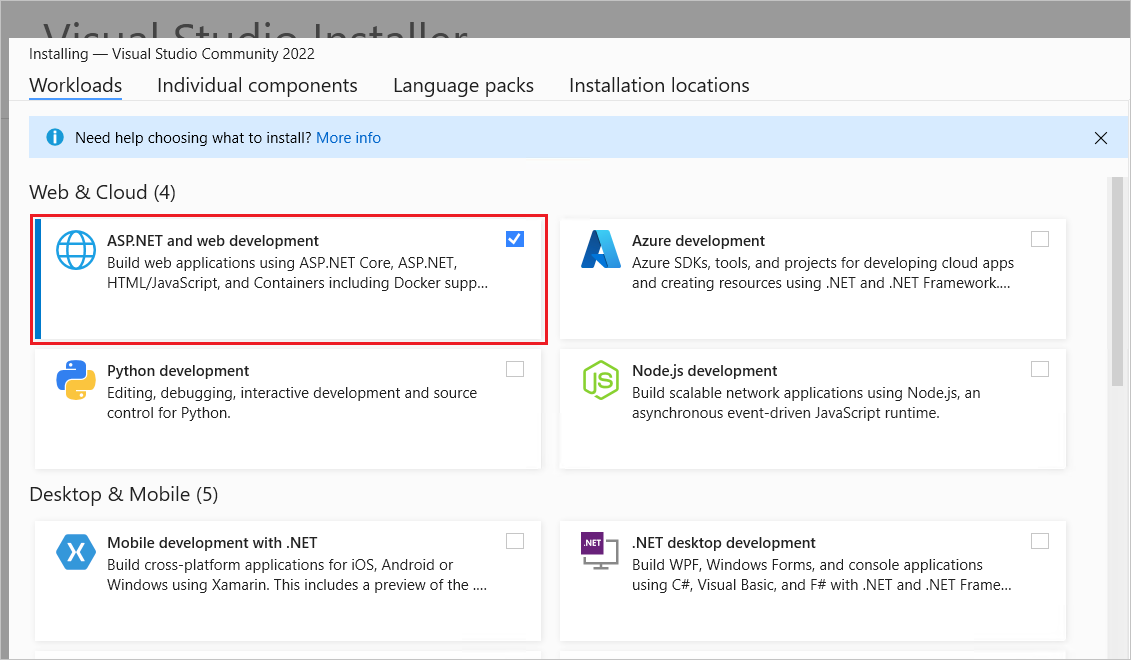
Lecture : Chamroeun Thou

# + Overview

GET , GET BY PARAMS, POST, PUT (UPDATE), DELETE



# + Start Create Web Project

-Open Visual Studio 2022 with the ASP.NET and web development workload.

-From the File menu, select New > Project.

-Enter Web API in the search box.

-Select the ASP.NET Core Web API template and select Next.

-In the Configure your new project dialog, name the project TodoApi and select Next.

-In the Additional information dialog:

+Confirm the Framework is .NET 7.0 (or later).

+Confirm the checkbox for Use controllers(uncheck to use minimal APIs) is

checked.

+Select Create.



# + Test the project

The project template creates a **WeatherForecast** API with support for Swagger.

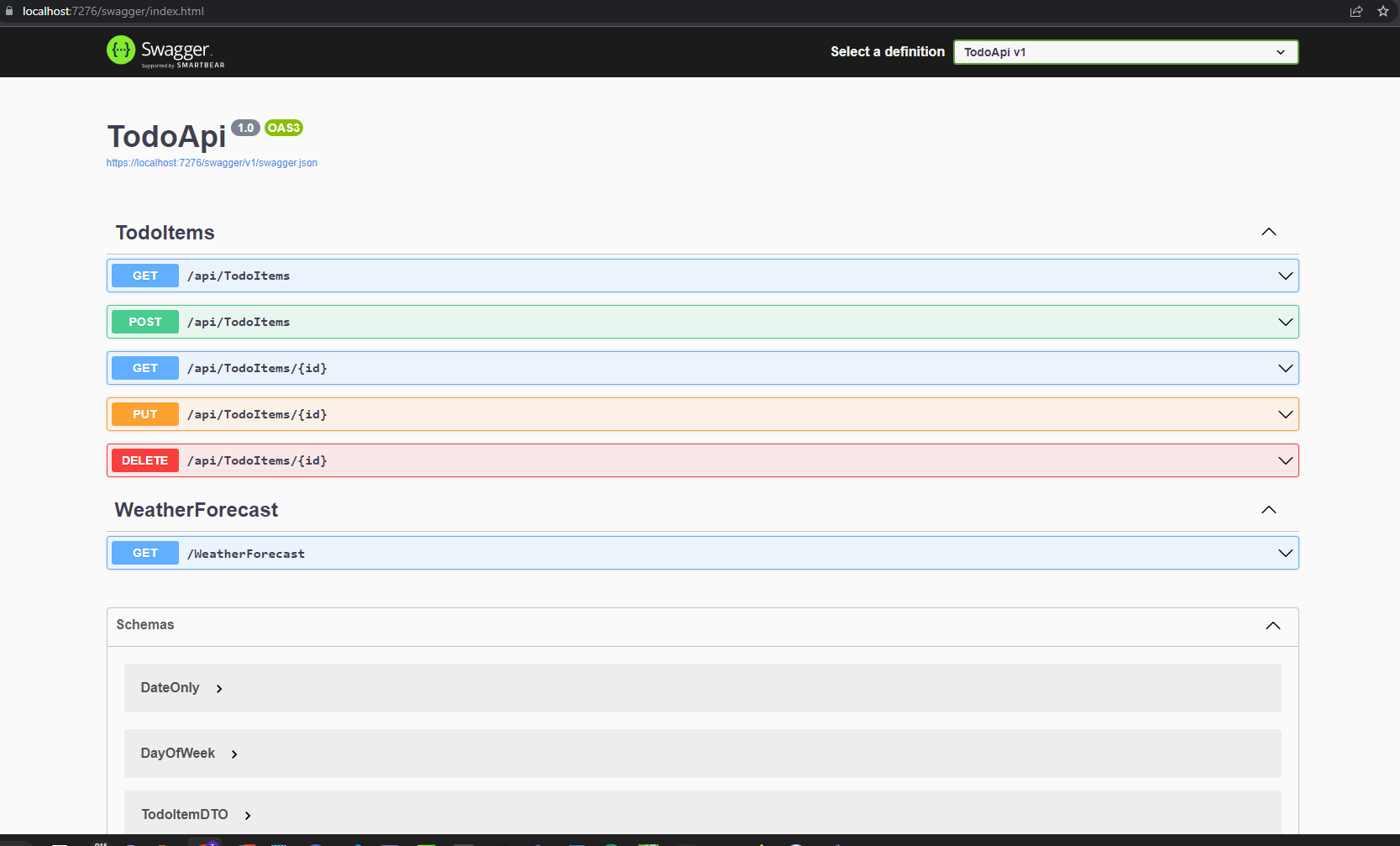
Swagger (OpenAPI) is a language-agnostic specification for describing REST APIs. It allows both computers and humans to understand the capabilities of a REST API without direct access to the source code. Its main goals are to:

-Minimize the amount of work needed to connect decoupled services.

-Reduce the amount of time needed to accurately document a service.

Visual Studio launches the default browser and navigates to **https://localhost:<port>/swagger/index.html**, where <port> is a randomly chosen port number.

The result displayed:



Select GET > Try it out > Execute. The page displays:

-The Curl command to test the WeatherForecast API.

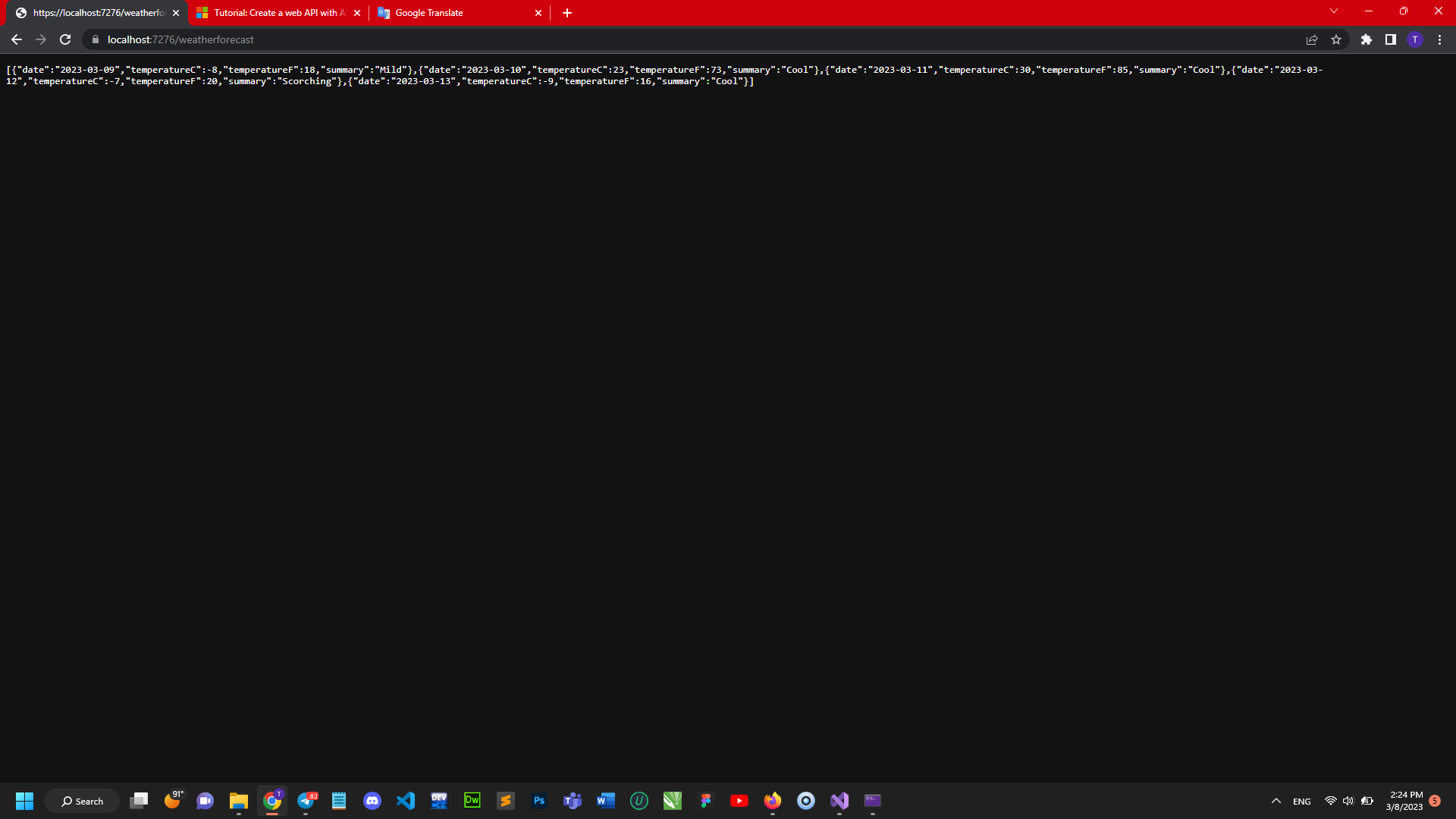
-The URL to test the WeatherForecast API.

-The response code, body, and headers.

-A drop-down list box with media types and the example value and schema.

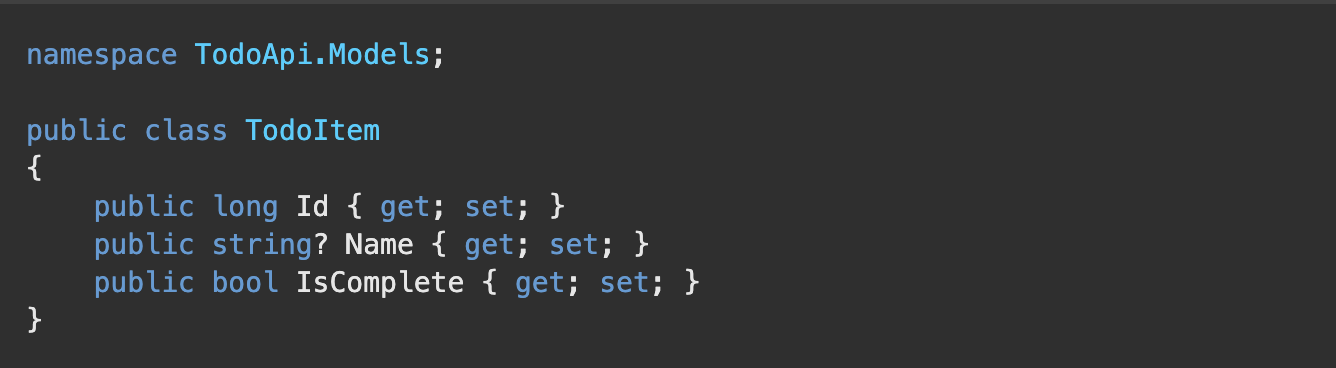
Copy and paste the Request URL in the browser: https://localhost:<port>/weatherforecast

JSON is returned:



# + Add a Model Class

* In Solution Explorer, right-click the project. Select Add > New Folder. Name the folder Models.
* Right-click the Models folder and select Add > Class. Name the class TodoItem and select Add.
* Replace the template code with the following:



# + Add a Database Context

The database context is the main class that coordinates Entity Framework functionality for a data model. This class is created by deriving from the Microsoft.EntityFrameworkCore.DbContext class.

**Add NuGet packages**

-From the Tools menu, select NuGet Package Manager > Manage NuGet Packages for Solution.

-Select the Browse tab, and then enter **Microsoft.EntityFrameworkCore.InMemory** in the search box.

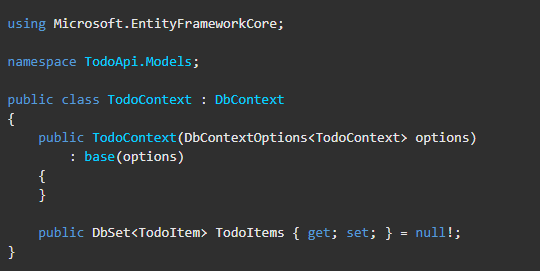
-Select **Microsoft.EntityFrameworkCore.InMemory** in the left pane.

-Select the Project checkbox in the right pane and then select Install.

**Add the TodoContext database context**

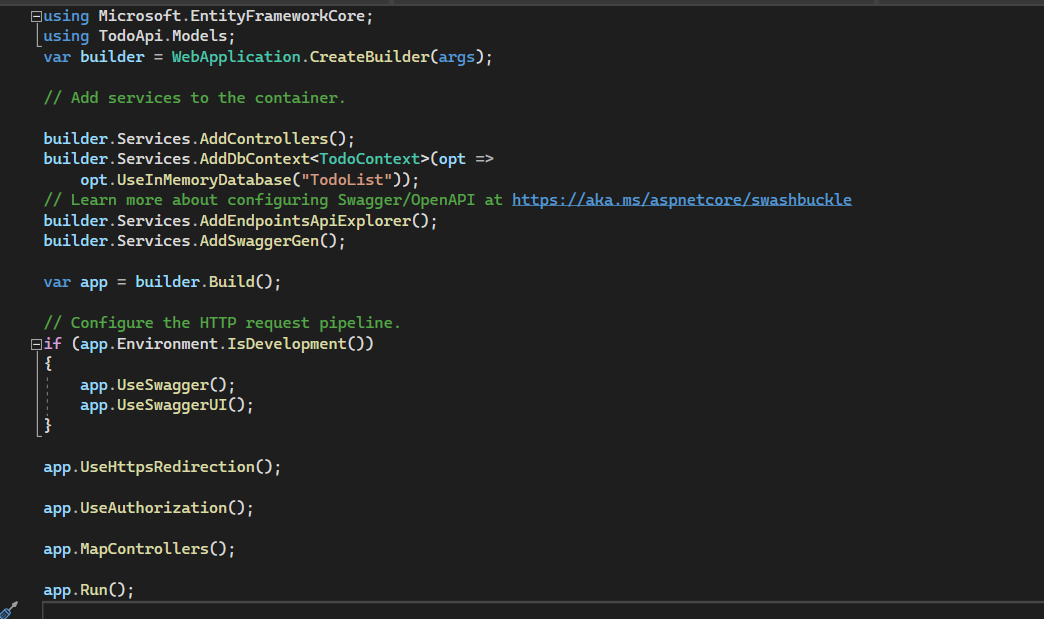
-Right-click the **Models** folder and select Add > Class. Name the class TodoContext and click Add.

-Enter the following code:



**+ Register the Database Context**

-Update **Program.cs**



# + Scaffold a Controller

-Right-click the Controllers folder.

-Select **Add** > **New Scaffolded Item**.

-Select **API Controller with actions, using Entity Framework,** and then select **Add**.

In the **Add API Controller with actions, using Entity Framework** dialog:

-Select **TodoItem (TodoApi.Models)** in the Model class.

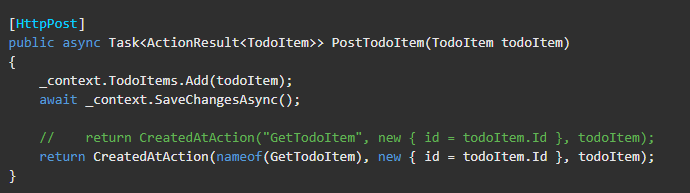
-Select **TodoContext (TodoApi.Models)** in the Data context class.

-Select **Add**.

If the scaffolding operation fails, select **Add** to try scaffolding a second time.

# + Update the PostTodoItem Create Method

Update the return statement in the **PostTodoItem** to use the **nameof** operator:



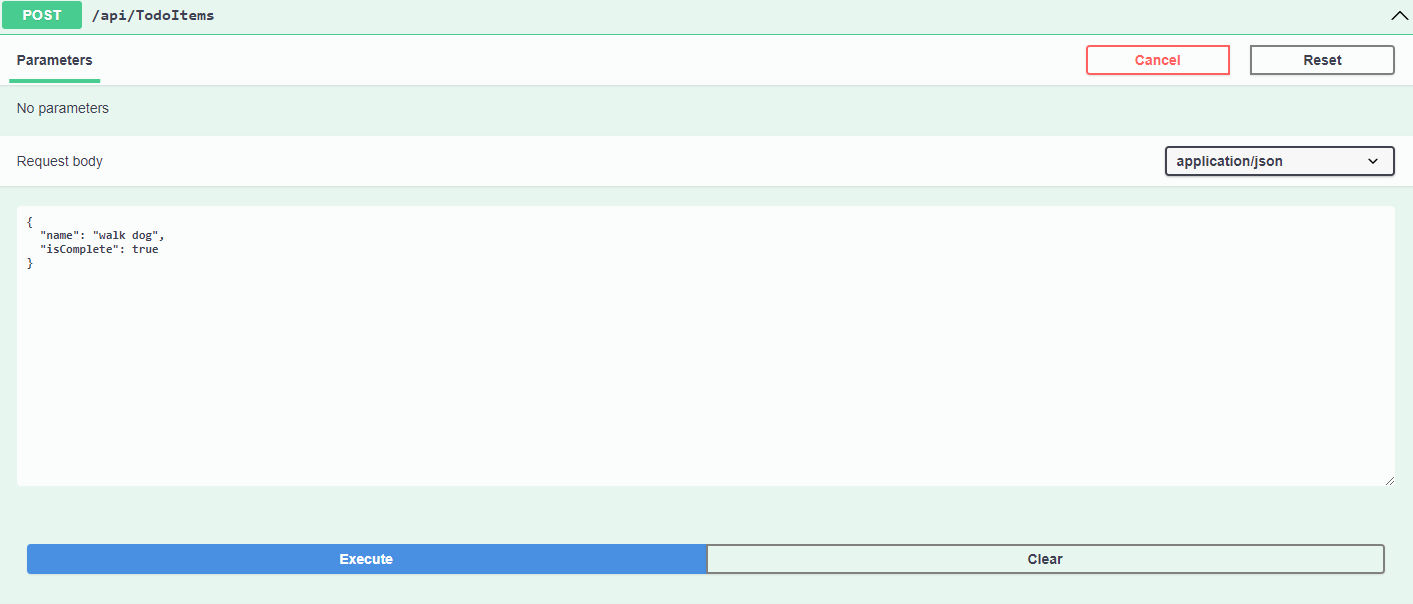
The preceding code is an **HTTP POST method**, as indicated by the **[HttpPost]** attribute. The method gets the value of the TodoItemfrom the body of the **HTTP request**Test PostTodoItemTest the Location Header URI Examine the GET Methods

**+ Test PostTodoItem**

-Press Ctrl+F5 to run the app.

-In the Swagger browser window, select **POST /api/TodoItems**, and then select Try it out.

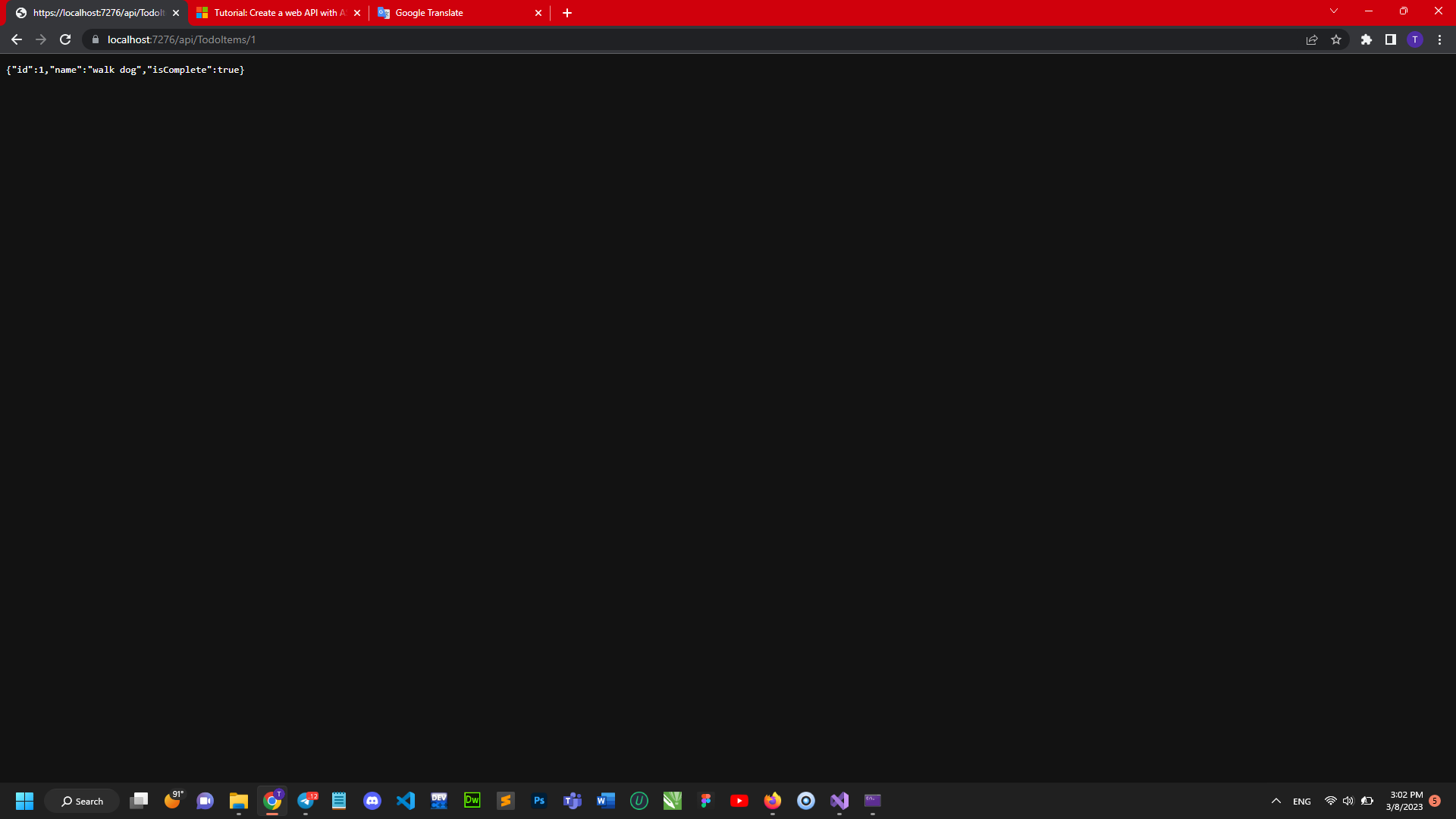
-In the **Request body** input window, update the JSON. For example,



-select **Execute**

**+ Test the location header URI**

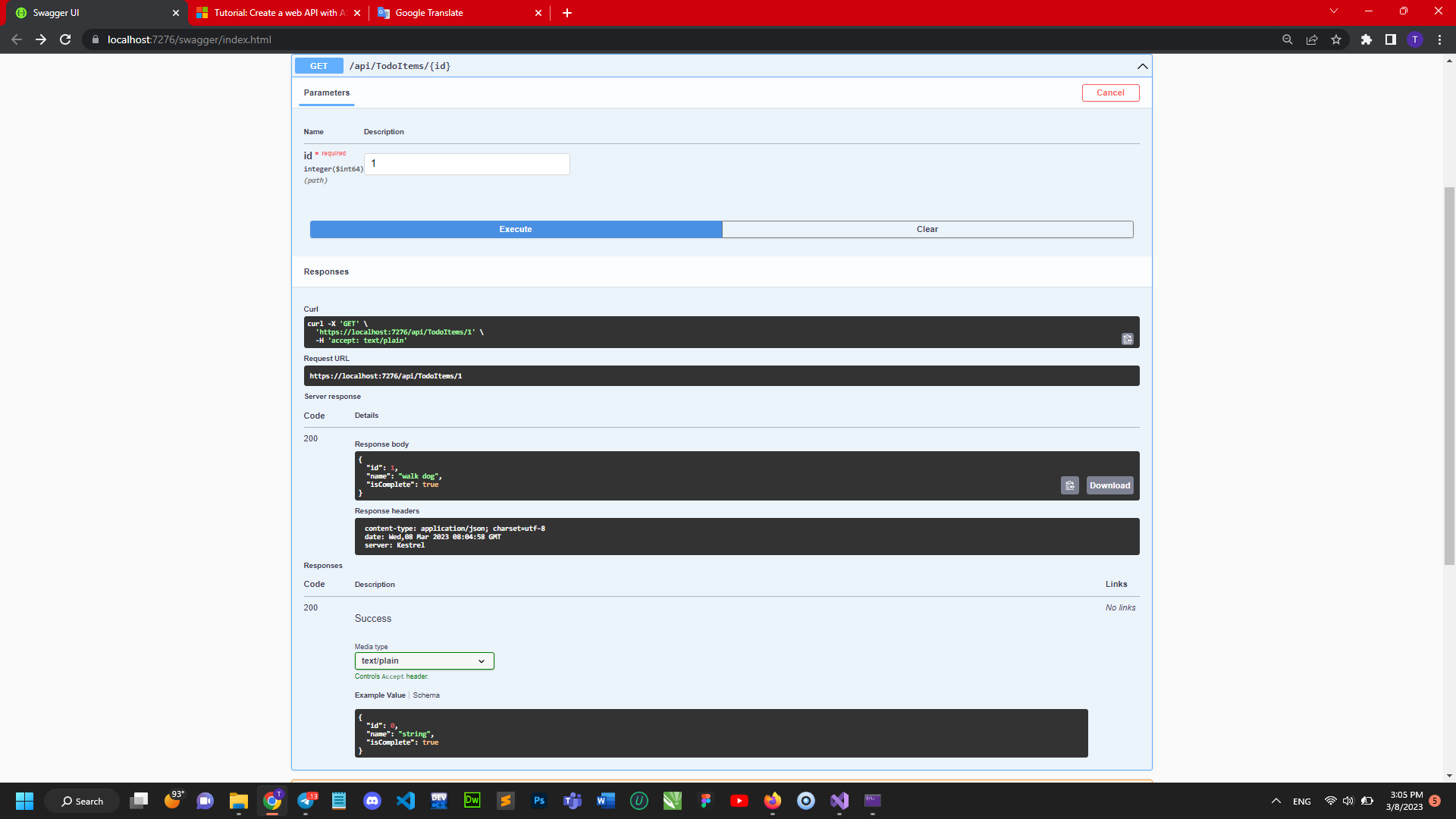
In the preceding POST, the Swagger UI shows the location header under Response headers. For example, location: **https://localhost:7276/api/TodoItems/1**. The location header shows the URI to the created resource.



**To test the location header:**

-In the Swagger browser window, select GET /api/TodoItems/{id}, and then select Try it out.

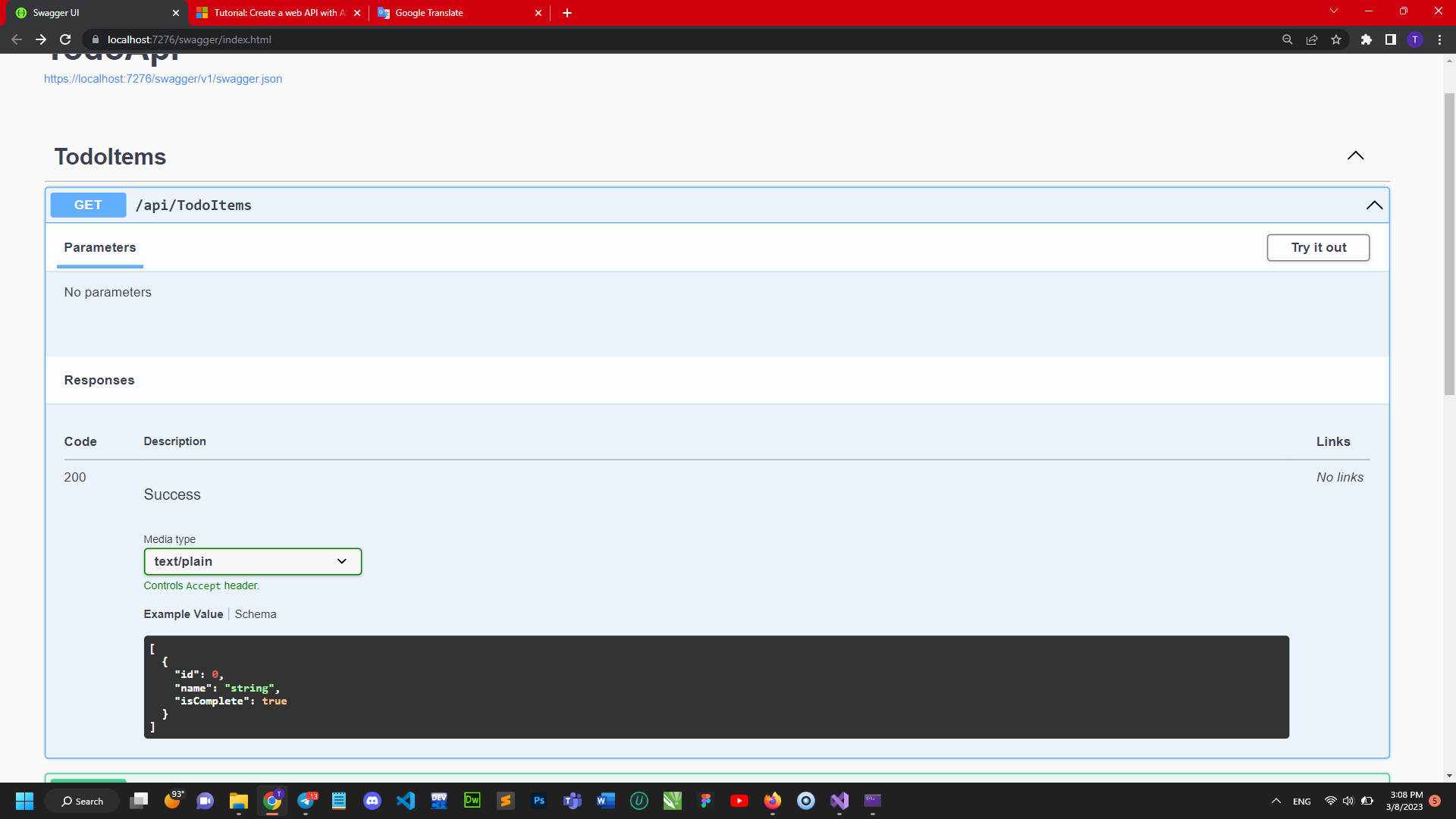
-Enter 1 in the id input box, and then select Execute.



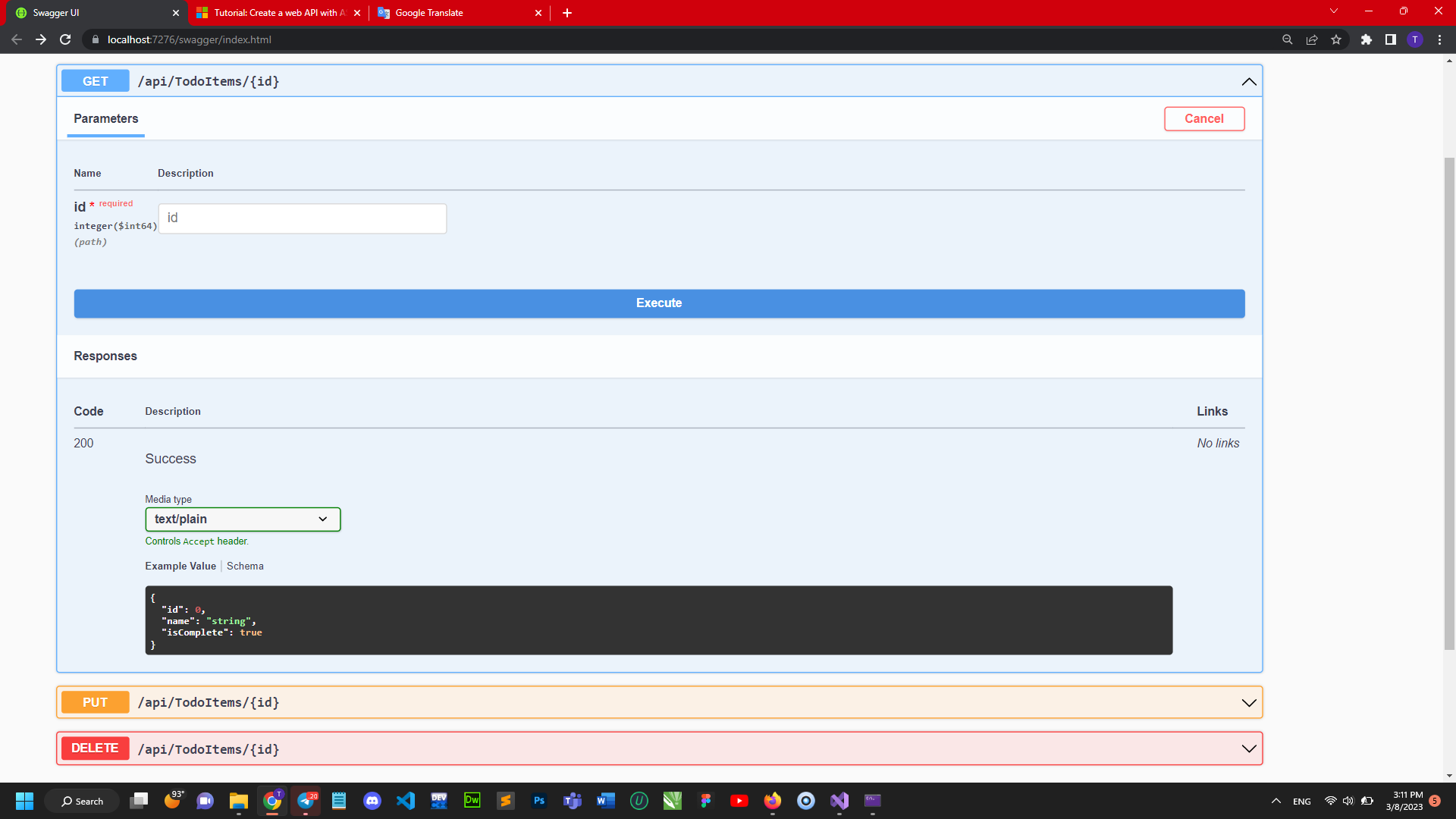
**+ Examine the GET method**

Two GET endpoints are implemented:

**-GET /api/todoitems**

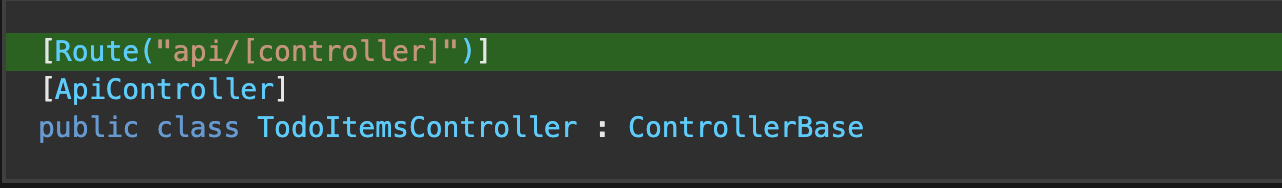
****

**-GET /api/todoitems/{id}**

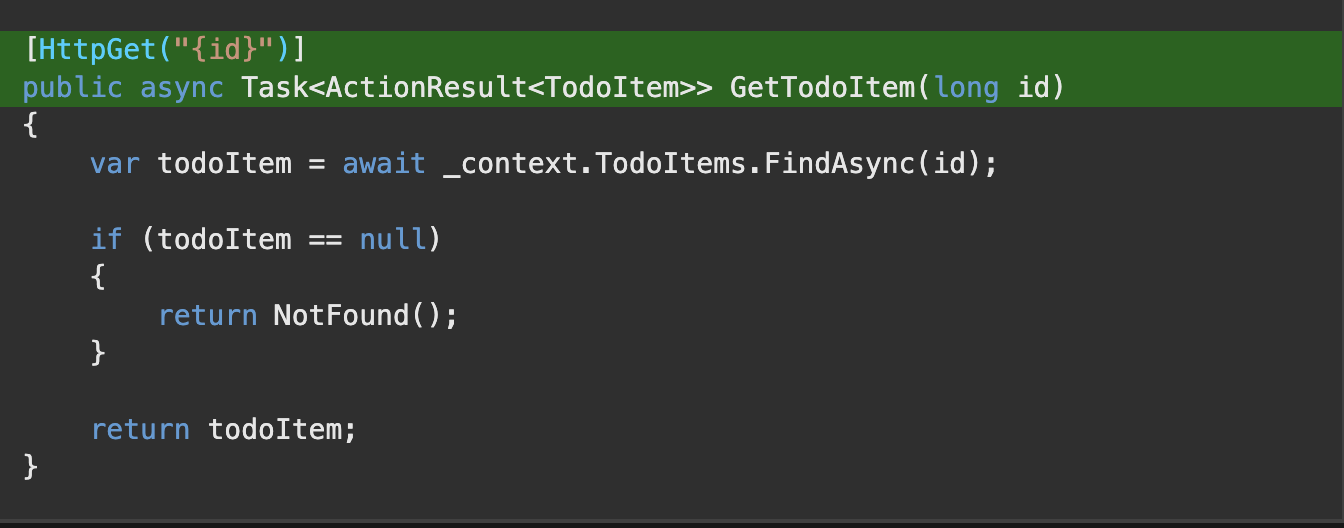


# + Routing and URL paths

### Start with the template string in the controller's Route attribute:



### GetTodoItem method, "{id}" is a placeholder variable for the unique identifier of the to-do item. When GetTodoItem is invoked, the value of "{id}" in the URL is provided to the method in its id parameter.



# + Return Values

* If no item matches the requested ID, the method returns a 404 status NotFound error code.
* Otherwise, the method returns 200 with a JSON response body. Returning item results in an HTTP 200 response.

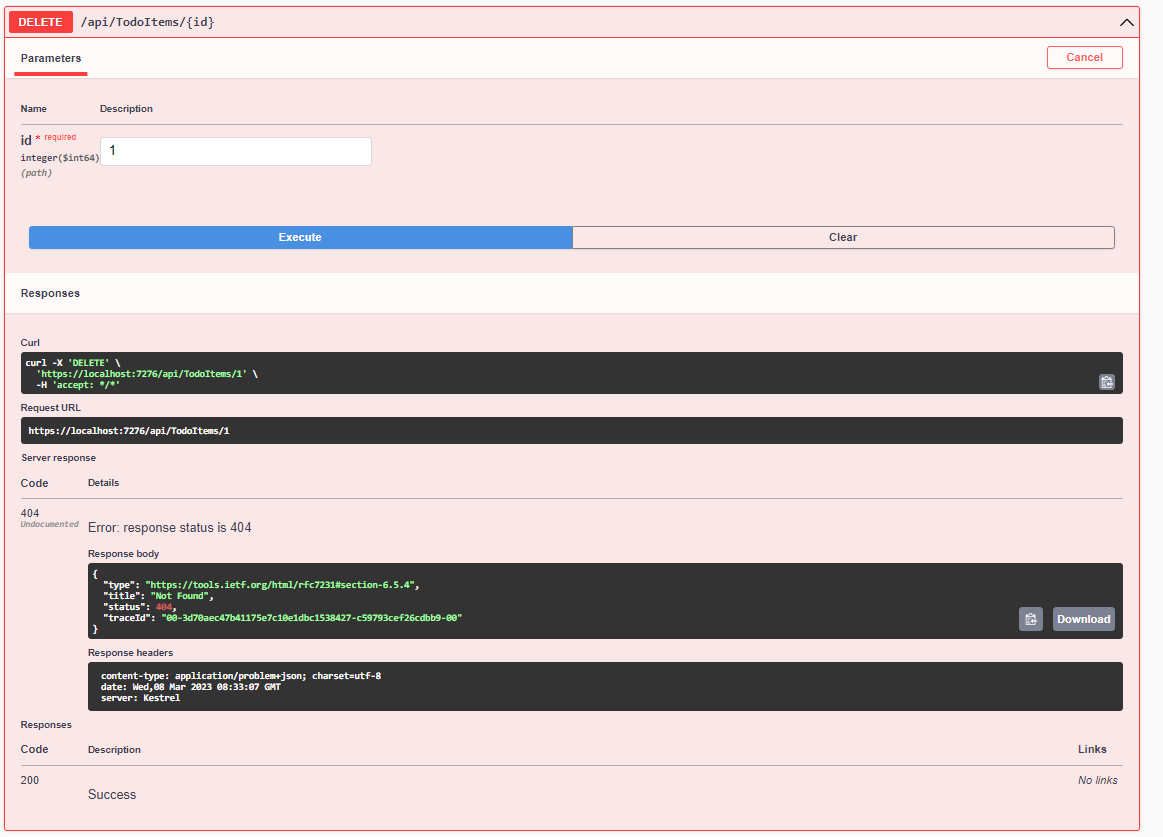
# + PutTodoItem Method

# 

# 

# 

# + DeleteTodoItem Method

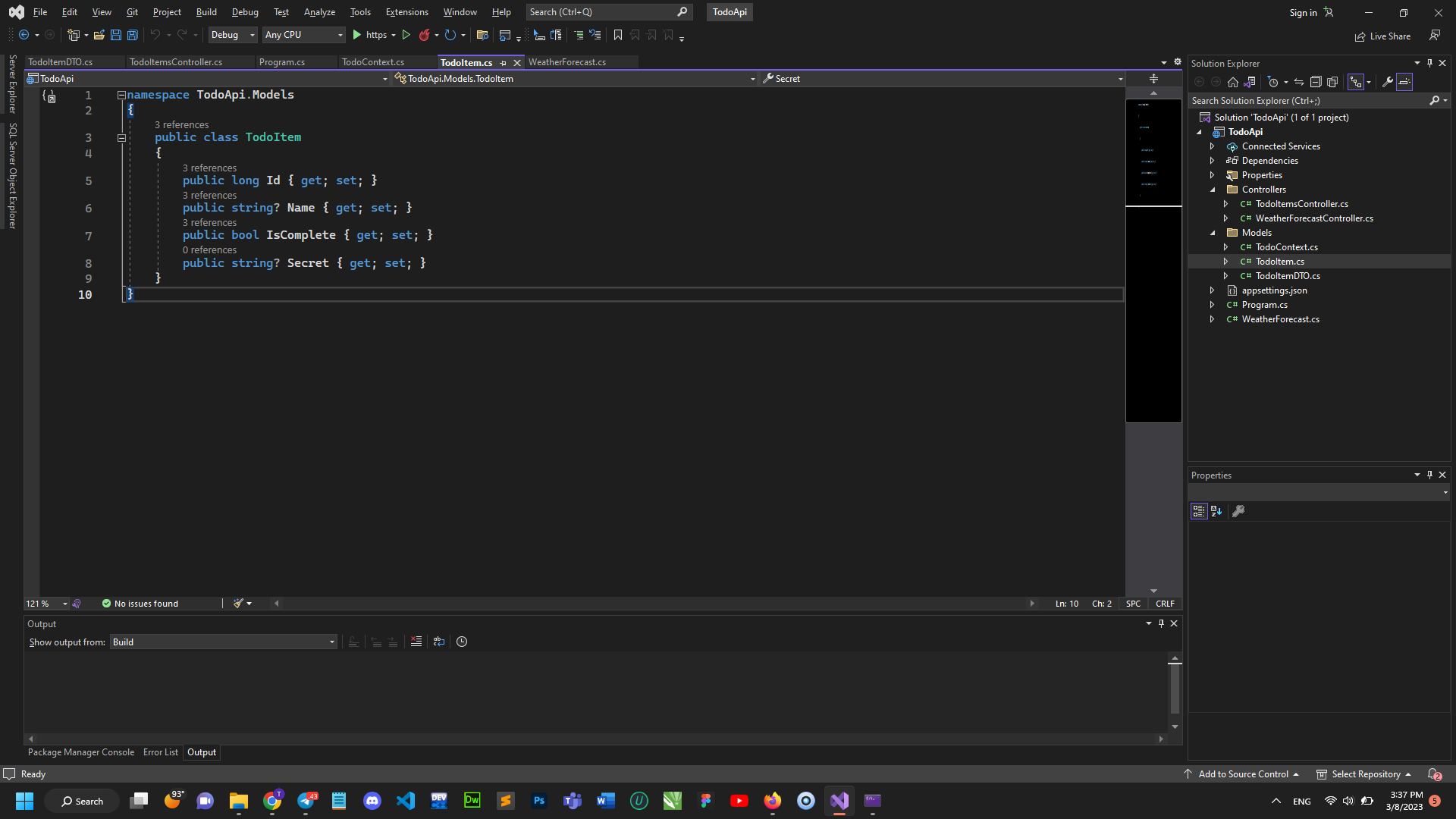


# + Prevent Over-Posting

### **A DTO may be used to :**

* Prevent over-posting.
* Hide properties that clients are not supposed to view.
* Omit some properties in order to reduce payload size.
* Flatten object graphs that contain nested objects. Flattened object graphs can be more convenient for clients.

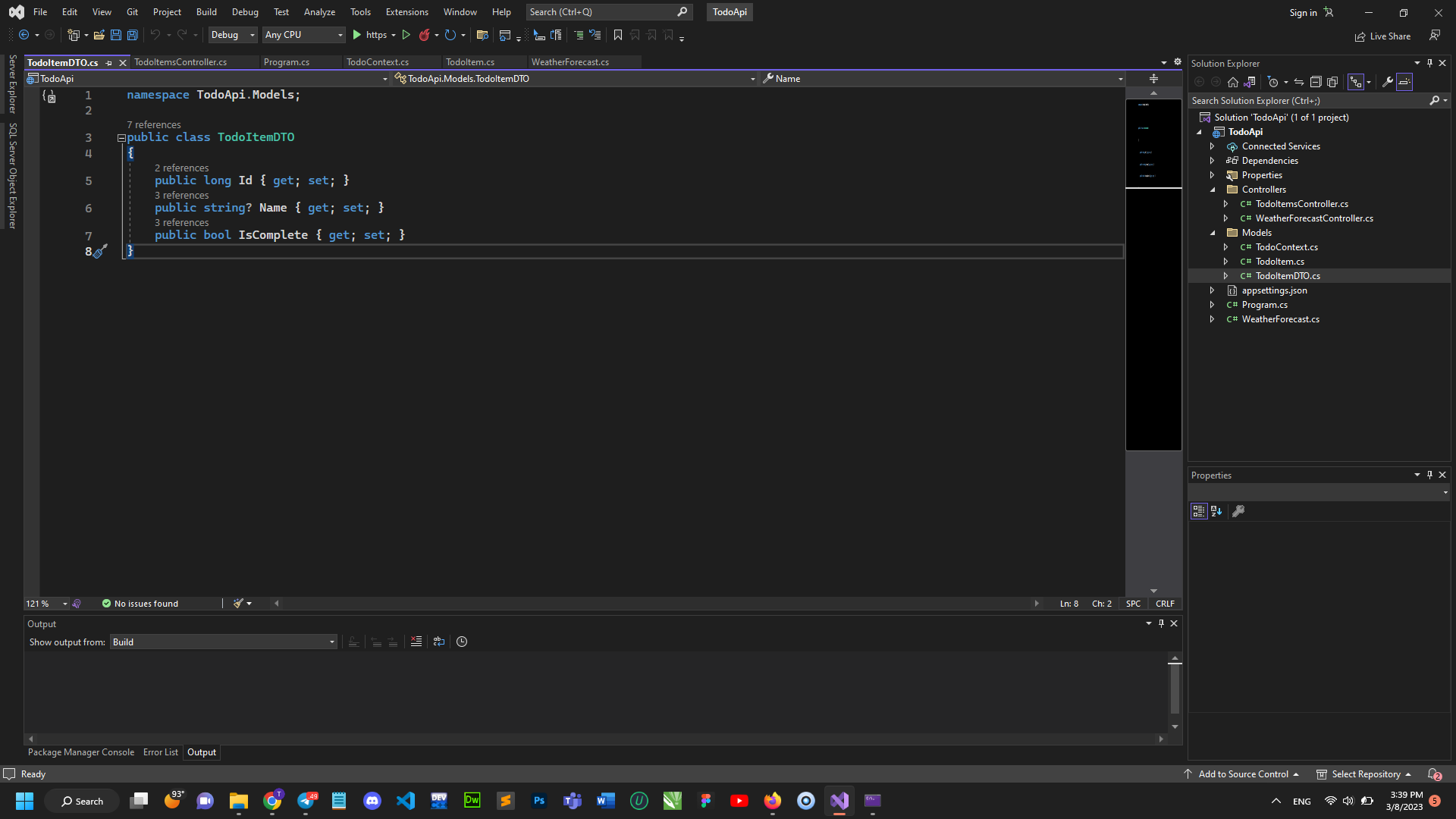
To demonstrate the DTO approach, update the **TodoItem** class to include a secret field:



The secret field needs to be hidden from this app, but an administrative app could choose to expose it.

Verify you can post and get the secret field.

Create a DTO model:



Update the **TodoItemsController** to use **TodoItemDTO**:

