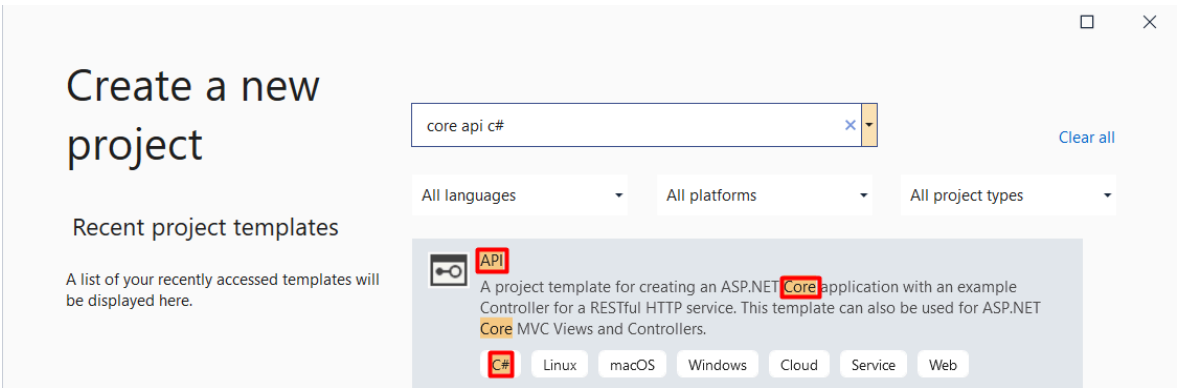
Tutorial: Create an ASP.NET Core web API project

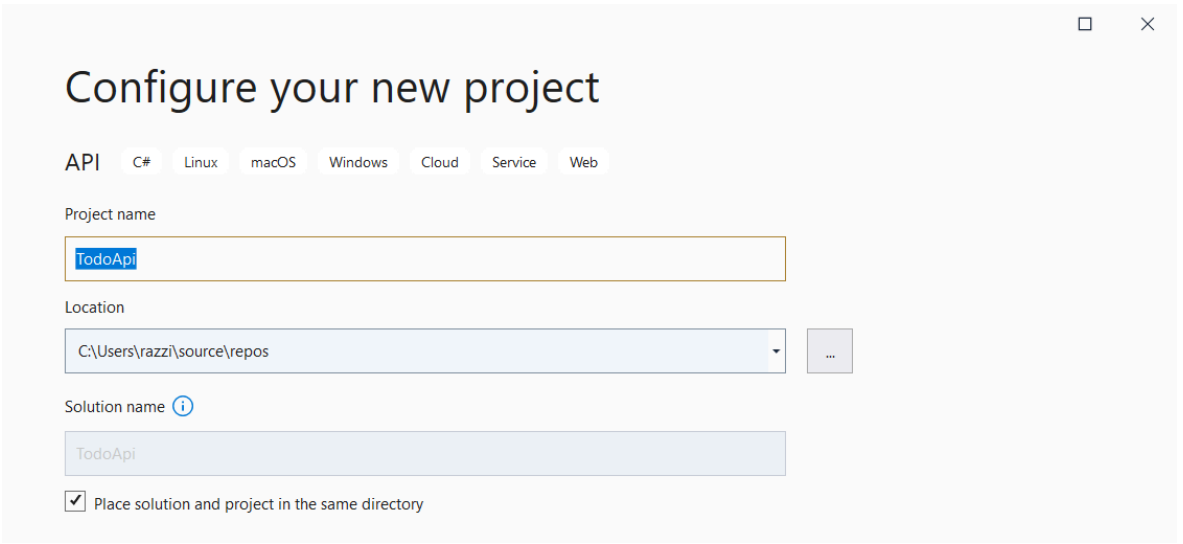
⚠️ This tutorial is based on Visual Studio 2019 16.11 and ASP.Net Core 3.1

Create Web API Project

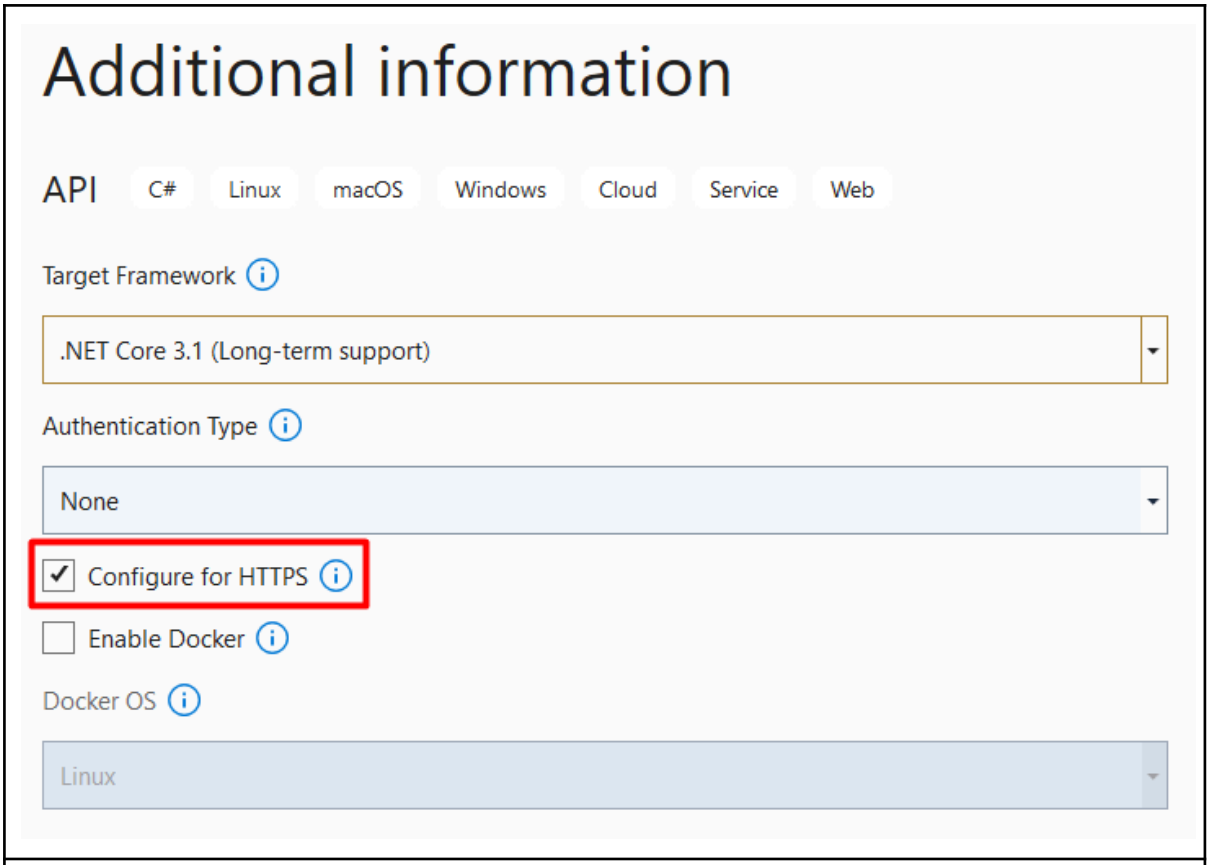
Search for the template using the keywords core api c#

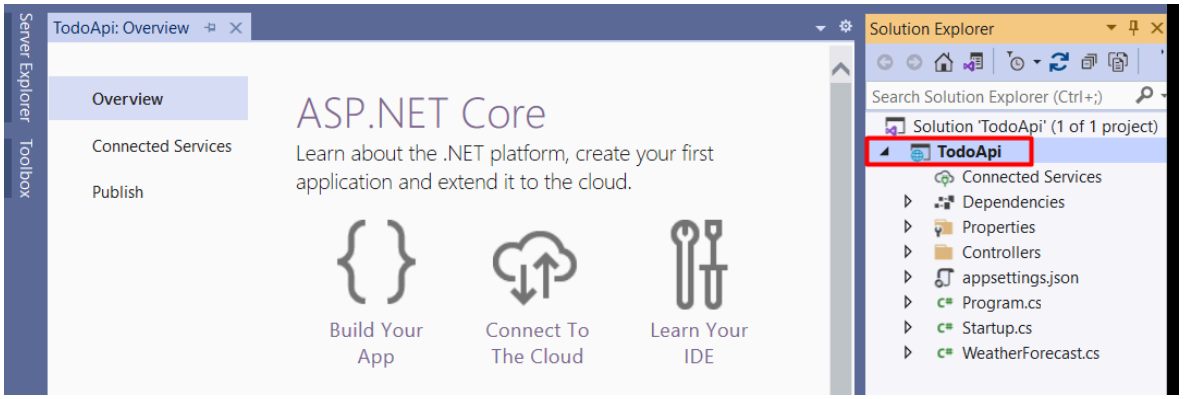


Project Name = TodoApi

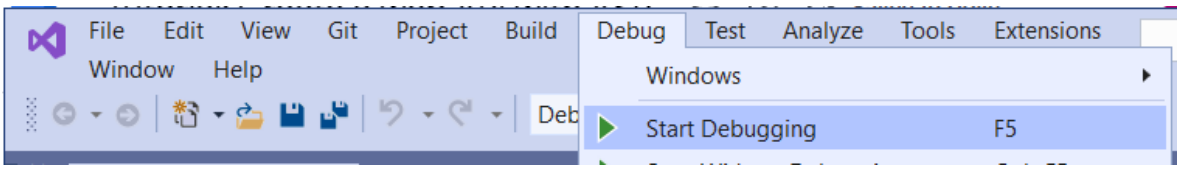


(In the Create a new ASP.NET Core Web Application dialog, confirm that .NET Core and ASP.NET Core 3.1 are selected. Select the API template and click Create.)



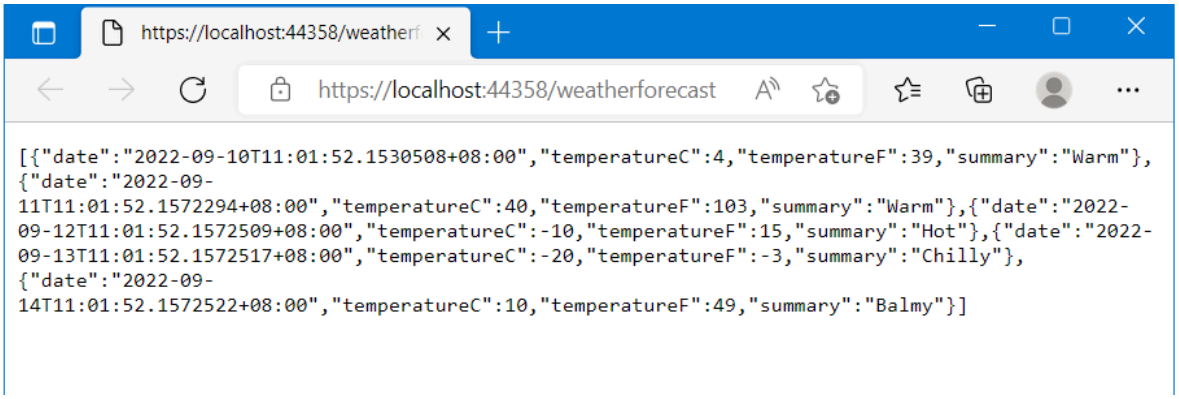


Test Run
Press Ctrl+F5 to run the app. Visual Studio launches a browser and navigates to `https://localhost:<port>/weatherforecast`, where `<port>` is a randomly chosen port number. If you get a dialog box that asks if you should trust the IIS Express certificate, select Yes. In the Security Warning dialog that appears next, select Yes.
For manual step, go to menu Debug/Start Debugging



You can also click the run button labeled with IIS

JSON similar to the following is returned:



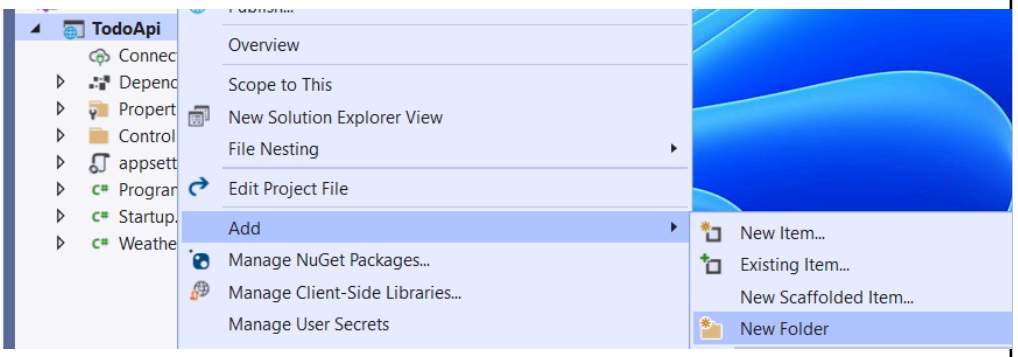

```
[
  {
    "date": "2019-07-16T19:04:05.7257911-06:00",
    "temperatureC": 52,
    "temperatureF": 125,
    "summary": "Mild"
  },
  {
    "date": "2019-07-17T19:04:05.7258461-06:00",
    "temperatureC": 36,
    "temperatureF": 96,
    "summary": "Warm"
  },
  {
    "date": "2019-07-18T19:04:05.7258467-06:00",
    "temperatureC": 39,
    "temperatureF": 102,
    "summary": "Cool"
  },
  {
    "date": "2019-07-19T19:04:05.7258471-06:00",
    "temperatureC": 10,
    "temperatureF": 49,
    "summary": "Bracing"
```

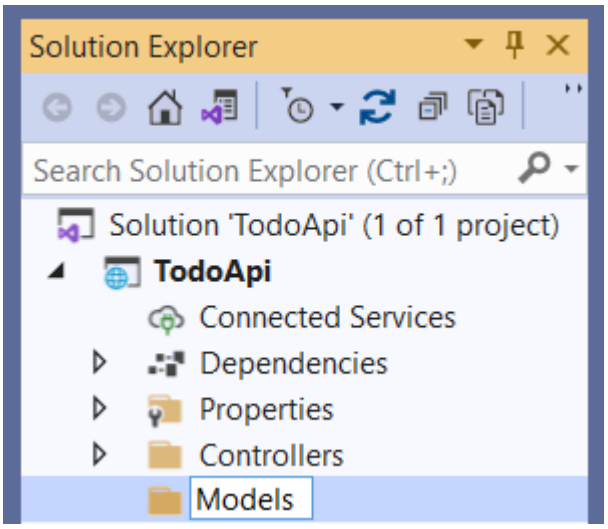
```
  },
  {
    "date": "2019-07-20T19:04:05.7258474-06:00",
    "temperatureC": -1,
    "temperatureF": 31,
    "summary": "Chilly"
  }
]
```

Add a model class

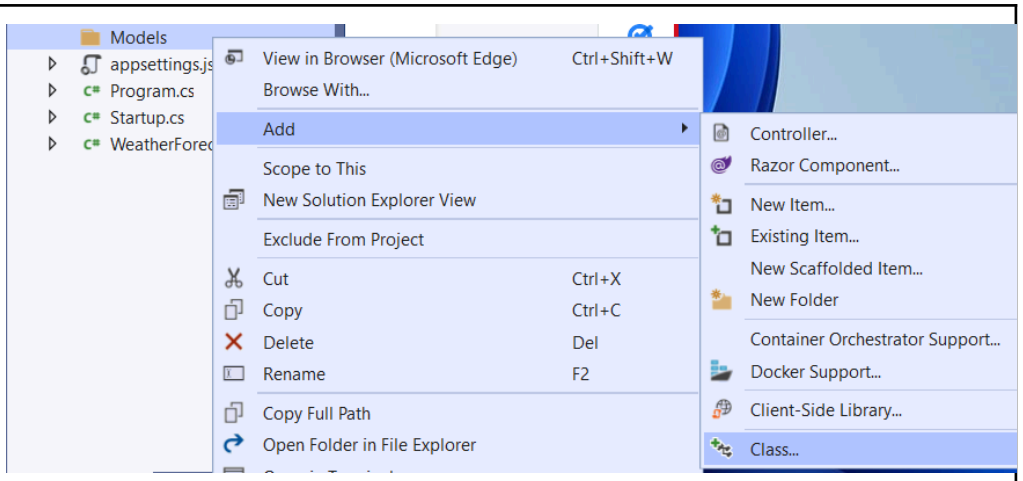
A model is a set of classes that represent the data that the app manages.
The model for this app is a single TodoItem class.

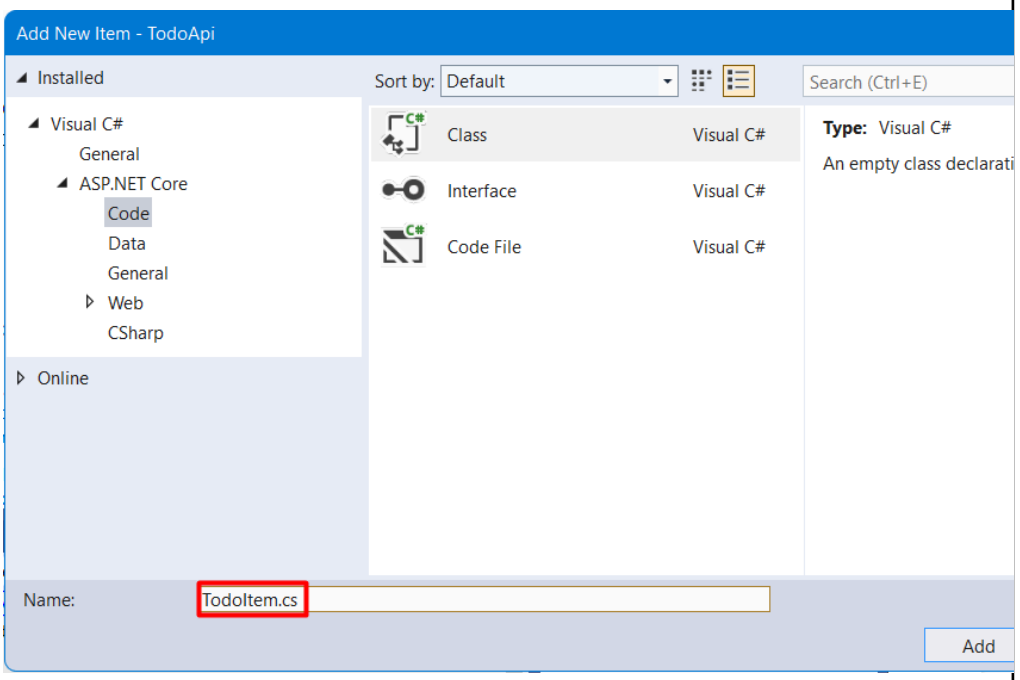
- In Solution Explorer, right-click the project. Select Add > New Folder. Name the folder `Models`.

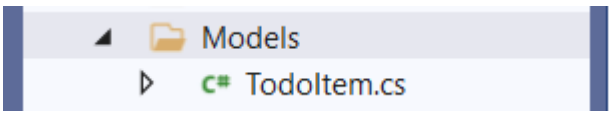




- Right-click the `Models` folder and select Add > Class. Name the class *TodoItem* and select Add.







- Replace the template code with the following code:

```
public class TodoItem
{
    public long Id { get; set; }
    public string Name { get; set; }
    public bool IsComplete { get; set; }
}
```

The complete codes for TodoItem.cs is as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TodoApi.Models
{
    public class TodoItem
    {
        public long Id { get; set; }
        public string Name { get; set; }
        public bool IsComplete { get; set; }
    }

}
```

The Id property functions as the unique key in a relational database.

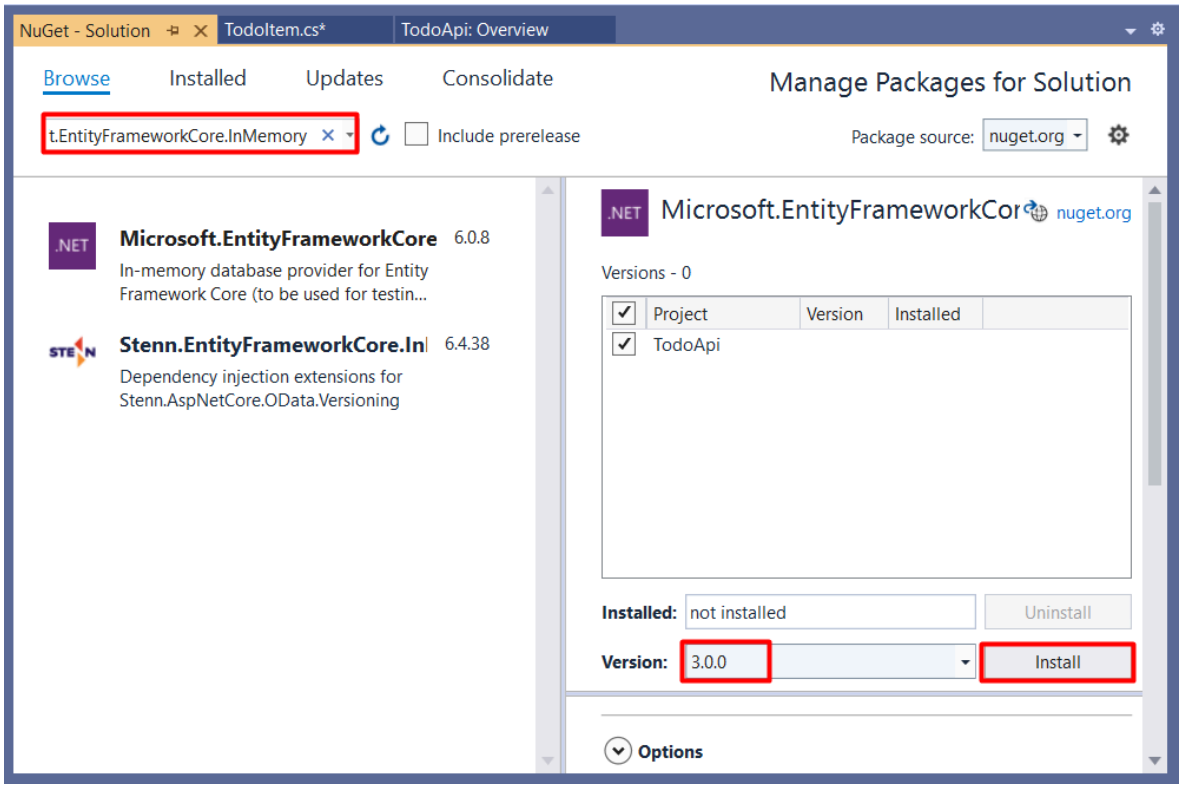
Model classes can go anywhere in the project, but the Models folder is used by convention.

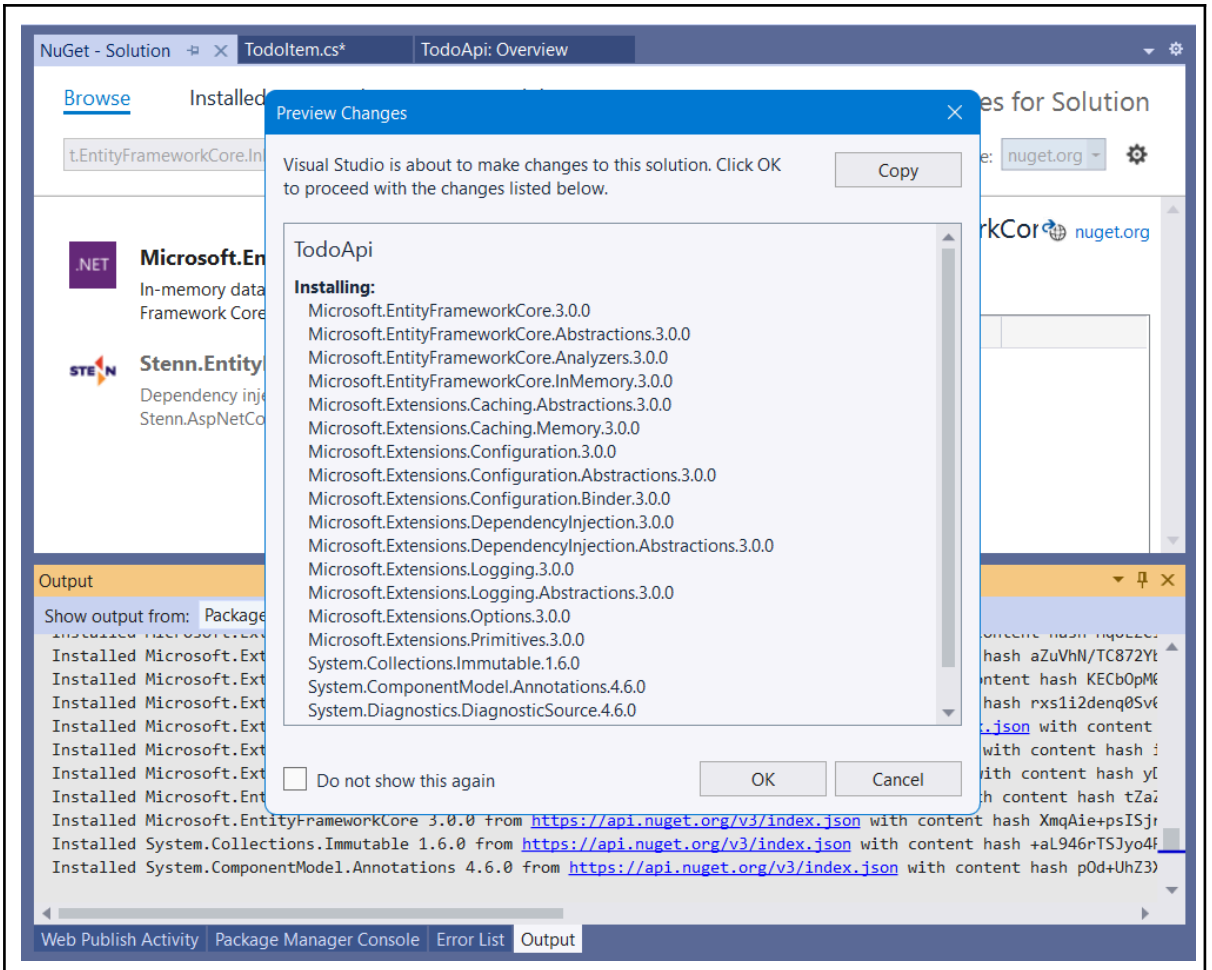
Add a database context

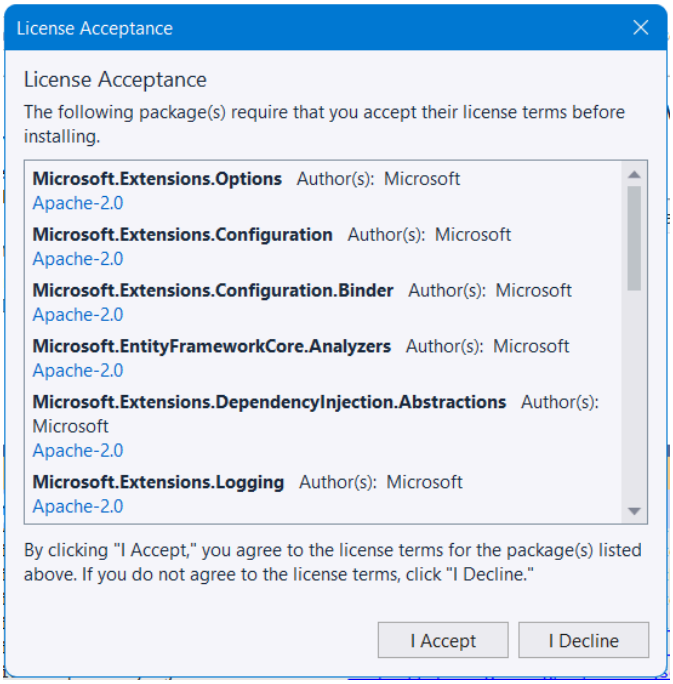
The *database context* is the main class that coordinates Entity Framework functionality for a data model.
This class is created by deriving from the `Microsoft.EntityFrameworkCore.DbContext` class.

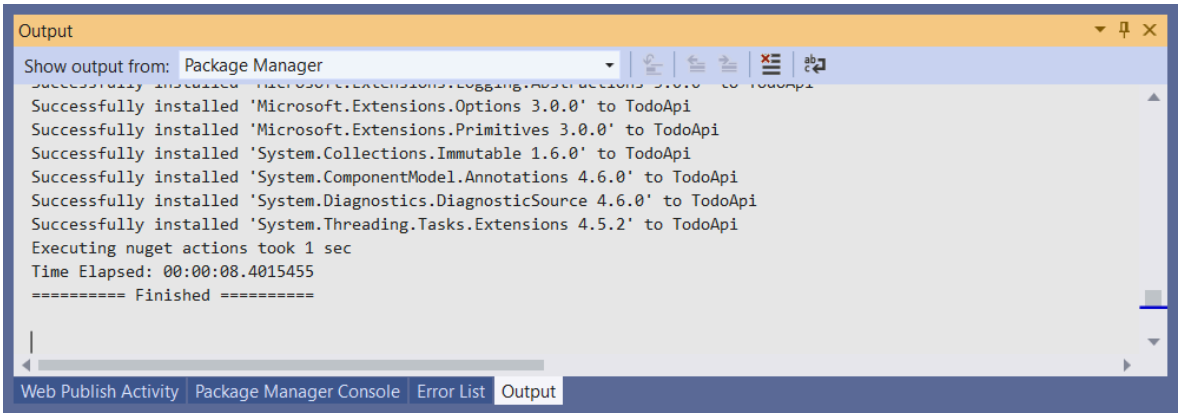
Add NuGet packages

- 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.




NuGet - Solution
TodoItem.cs*
TodoApi: Overview
Browse
Installed
es for Solution
nuget.org
Preview Changes
Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.
Copy
TodoApi
Installing:
Microsoft.EntityFrameworkCore.3.0.0
Microsoft.EntityFrameworkCore.Abstractions.3.0.0
Microsoft.EntityFrameworkCore.Analyzers.3.0.0
Microsoft.EntityFrameworkCore.InMemory.3.0.0
Microsoft.Extensions.Caching.Abstractions.3.0.0
Microsoft.Extensions.Caching.Memory.3.0.0
Microsoft.Extensions.Configuration.3.0.0
Microsoft.Extensions.Configuration.Abstractions.3.0.0
Microsoft.Extensions.Configuration.Binder.3.0.0
Microsoft.Extensions.DependencyInjection.3.0.0
Microsoft.Extensions.DependencyInjection.Abstractions.3.0.0
Microsoft.Extensions.Logging.3.0.0
Microsoft.Extensions.Logging.Abstractions.3.0.0
Microsoft.Extensions.Options.3.0.0
Microsoft.Extensions.Primitives.3.0.0
System.Collections.Immutable.1.6.0
System.ComponentModel.Annotations.4.6.0
System.Diagnostics.DiagnosticSource.4.6.0
Do not show this again
OK
Cancel
Output
Show output from:
Installed Microsoft.EntityFrameworkCore 3.0.0 from https://api.nuget.org/v3/index.json with content hash XmqAie+psISj
Installed System.Collections.Immutable 1.6.0 from https://api.nuget.org/v3/index.json with content hash +aL946rTSJyo4
Installed System.ComponentModel.Annotations 4.6.0 from https://api.nuget.org/v3/index.json with content hash pOd+UhZ3
Web Publish Activity
Package Manager Console
Error List
Output


License Acceptance
License Acceptance
The following package(s) require that you accept their license terms before installing.
Microsoft.Extensions.Options Author(s): Microsoft
Apache-2.0
Microsoft.Extensions.Configuration Author(s): Microsoft
Apache-2.0
Microsoft.Extensions.Configuration.Binder Author(s): Microsoft
Apache-2.0
Microsoft.EntityFrameworkCore.Analyzers Author(s): Microsoft
Apache-2.0
Microsoft.Extensions.DependencyInjection.Abstractions Author(s): Microsoft
Apache-2.0
Microsoft.Extensions.Logging Author(s): Microsoft
Apache-2.0
By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."
I Accept
I Decline



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:

```
using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models
{
    public class TodoContext : DbContext
    {
        public TodoContext(DbContextOptions<TodoContext> options)
            : base(options)
        {
        }

        public DbSet<TodoItem> TodoItems { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
```

```
namespace TodoApi.Models
{
    public class TodoContext : DbContext
    {
        public TodoContext(DbContextOptions<TodoContext> options)
            : base(options)
        {
        }

        public DbSet<TodoItem> TodoItems { get; set; }
    }
}
```

Register the database context

In ASP.NET Core, services such as the DB context must be registered with the dependency injection (DI) container. The container provides the service to controllers.

Update `Startup.cs` with the following highlighted code:

```
// Unused usings removed
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
~~using Microsoft.AspNetCore.HttpsPolicy;~~
~~using Microsoft.AspNetCore.Mvc;~~
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
~~using Microsoft.Extensions.Hosting;~~
~~using Microsoft.Extensions.Logging;~~
~~using System;~~
~~using System.Collections.Generic;~~
~~using System.Linq;~~
~~using System.Threading.Tasks;~~
using Microsoft.EntityFrameworkCore;
using TodoApi.Models;



namespace TodoApi
{
    public class Startup
    {
```

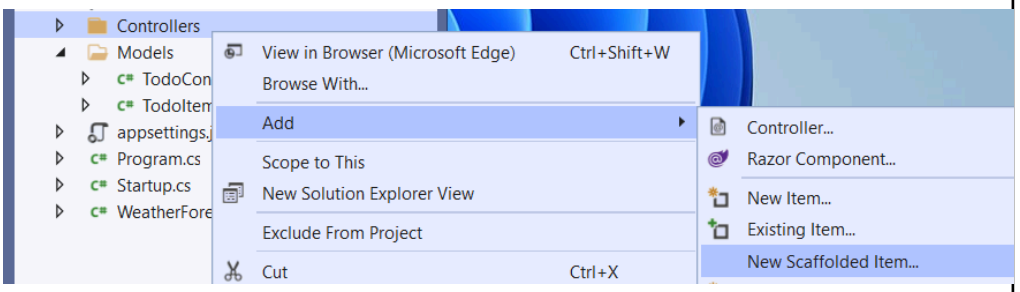
```
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<TodoContext>(opt =>
               opt.UseInMemoryDatabase("TodoList"));
            services.AddControllers();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

The preceding code:

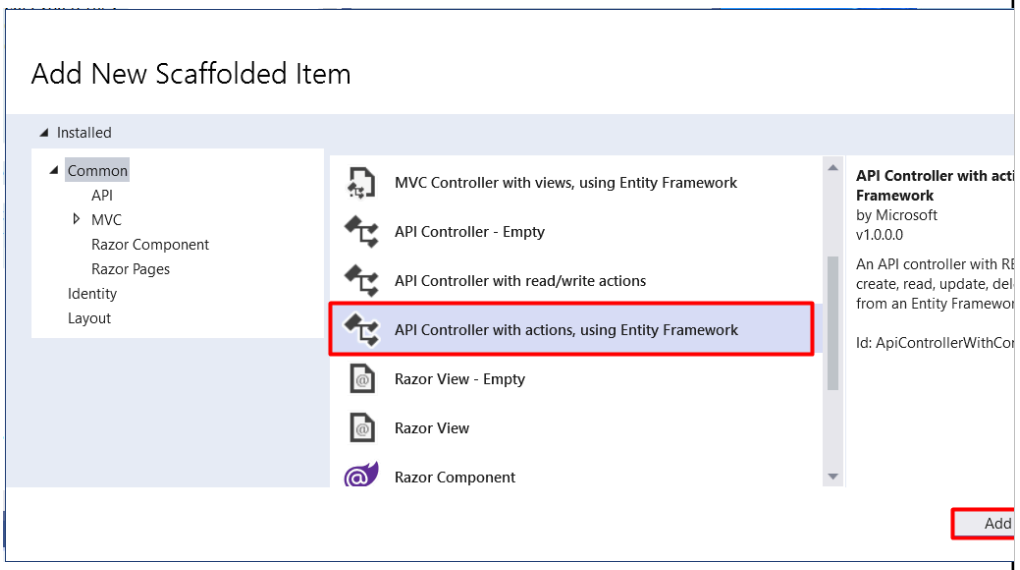
- Removes unused `using` declarations.
- Adds the database context to the DI container.
- Specifies that the database context will use an in-memory database.

Scaffold a controller

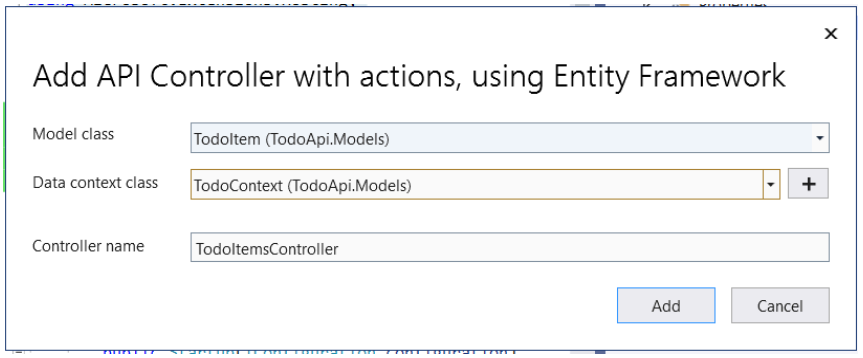
- Right-click the *Controllers* folder.
- Select Add > New Scaffolded Item.



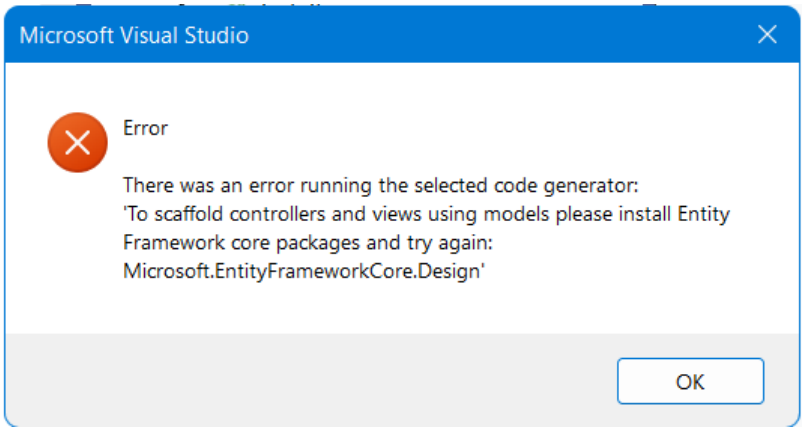
- Select API Controller with actions, using Entity Framework, and then select Add.



- 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.



Microsoft.EntityFrameworkCore.Design
Select version 3.1.3

-

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using TodoApi.Models;

namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TodoItemsController : ControllerBase
    {
        private readonly TodoContext _context;

        public TodoItemsController(TodoContext context)
        {
            _context = context;
        }

        // GET: api/TodoItems
        [HttpGet]
        public async Task<ActionResult<IEnumerable<TodoItem>>> GetTodoItems()
        {
            return await _context.TodoItems.ToListAsync();
        }

        // GET: api/TodoItems/5
        [HttpGet("{id}")]
        public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
        {
            var todoItem = await _context.TodoItems.FindAsync(id);

            if (todoItem == null)
            {
                return NotFound();
            }

            return todoItem;
```

```
        }

        // PUT: api/TodoItems/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to, for
        // more details, see https://go.microsoft.com/fwlink/?linkid=2123754.
        [HttpPut("{id}")]
        public async Task<IActionResult> PutTodoItem(long id, TodoItem todoItem)
        {
            if (id != todoItem.Id)
            {
                return BadRequest();
            }

            _context.Entry(todoItem).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!TodoItemExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/TodoItems
        // To protect from overposting attacks, enable the specific properties you want to
bind to, for
        // more details, see https://go.microsoft.com/fwlink/?linkid=2123754.
        [HttpPost]
        public async Task<ActionResult<TodoItem>> PostTodoItem(TodoItem todoItem)
        {
            _context.TodoItems.Add(todoItem);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetTodoItem", new { id = todoItem.Id }, todoItem);
        }

        // DELETE: api/TodoItems/5
        [HttpDelete("{id}")]
        public async Task<ActionResult<TodoItem>> DeleteTodoItem(long id)
        {
            var todoItem = await _context.TodoItems.FindAsync(id);
            if (todoItem == null)
            {
                return NotFound();
            }

            _context.TodoItems.Remove(todoItem);
            await _context.SaveChangesAsync();

            return todoItem;
        }

        private bool TodoItemExists(long id)
```

```
        {
            return _context.TodoItems.Any(e => e.Id == id);
        }
    }
}
```

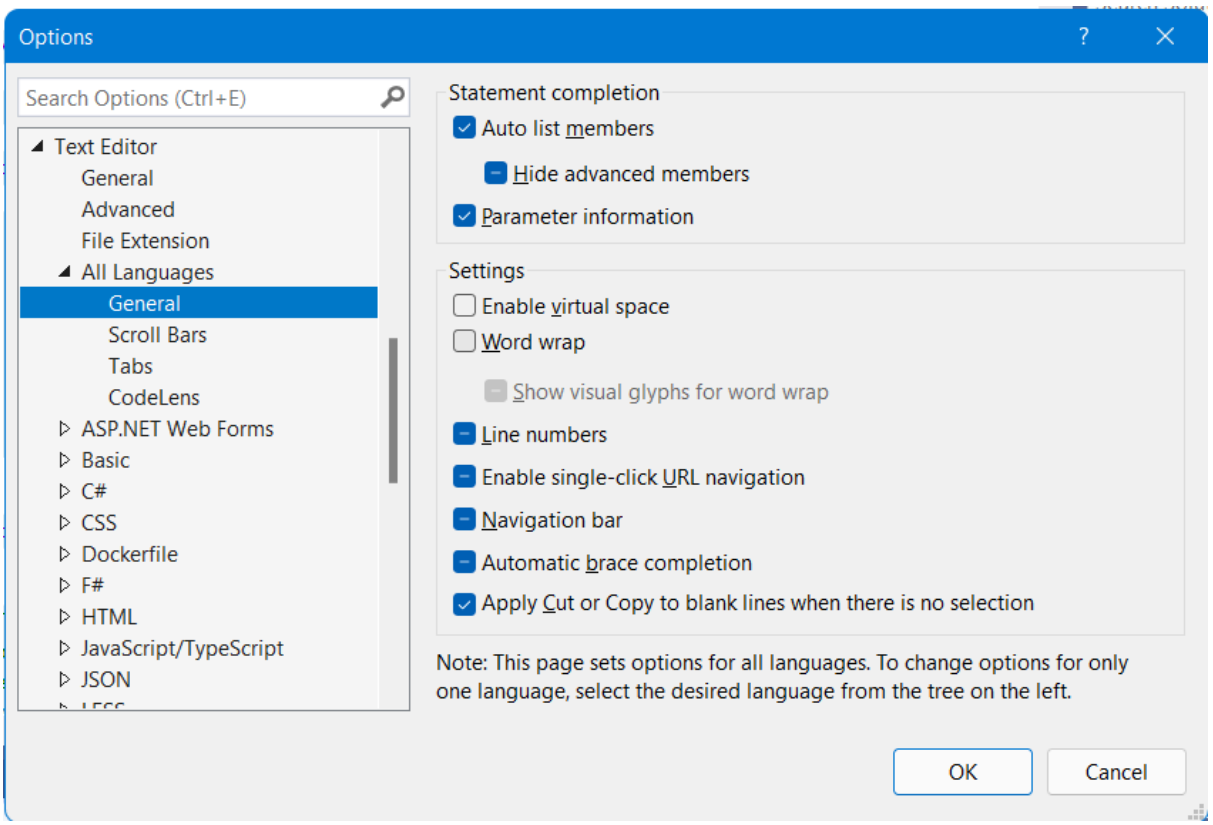
The generated code:

- Marks the class with the [ApiController] attribute. This attribute indicates that the controller responds to web API requests. For information about specific behaviors that the attribute enables, see Create web APIs with ASP.NET Core.
- Uses DI to inject the database context (`TodoContext`) into the controller. The database context is used in each of the CRUD methods in the controller.

The ASP.NET Core templates for:

- Controllers with views include `[action]` in the route template.
- API controllers don't include `[action]` in the route template.

When the `[action]` token isn't in the route template, the action name is excluded from the route. That is, the action's associated method name isn't used in the matching route.



Examine the PostTodoItem create method

Replace the return statement in the `PostTodoItem` to use the nameof operator:

```
// POST: api/TodoItems
[HttpPost]
public async Task<ActionResult<TodoItem>> PostTodoItem(TodoItem todoItem)
{
    _context.TodoItems.Add(todoItem);
    await _context.SaveChangesAsync();

    //return CreatedAtAction("GetTodoItem", new { id = todoItem.Id }, todoItem);
    return CreatedAtAction(nameof(GetTodoItem), new { id = todoItem.Id }, todoItem);
}
```

The preceding code is an HTTP POST method, as indicated by the [HttpPost] attribute. The method gets the value of the to-do item from the body of the HTTP request.

For more information, see Attribute routing with Http[Verb] attributes.

The CreatedAtAction method:

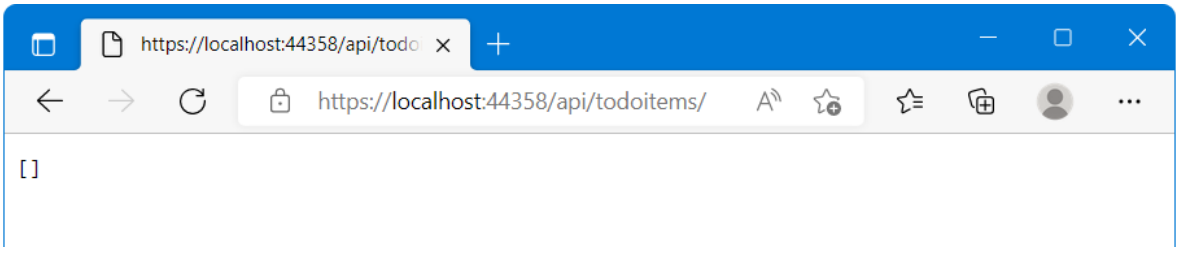
- Returns an HTTP 201 status code if successful. HTTP 201 is the standard response for an HTTP POST method that creates a new resource on the server.
- Adds a Location header to the response. The `Location` header specifies the URI of the newly created to-do item. For more information, see 10.2.2 201 Created.
- References the `GetTodoItem` action to create the `Location` header's URI. The C# `nameof` keyword is used to avoid hard-coding the action name in the `CreatedAtAction` call.

Download Sample Solution:

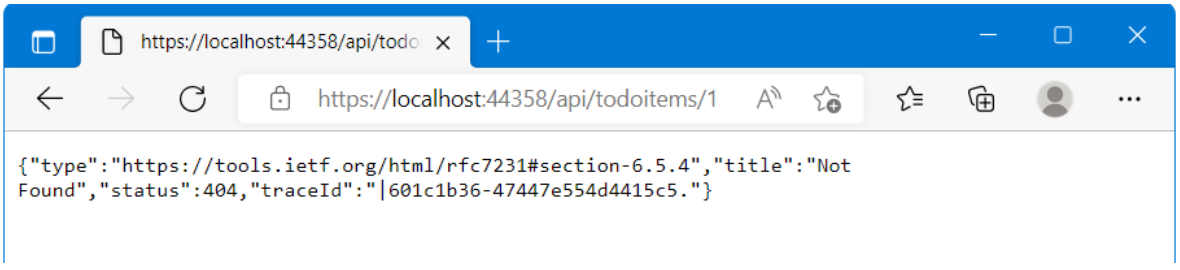
TodoApi.zip

Test Your API Endpoint:
https://localhost:<port>/api/todoitems/

At the moment the database is empty.



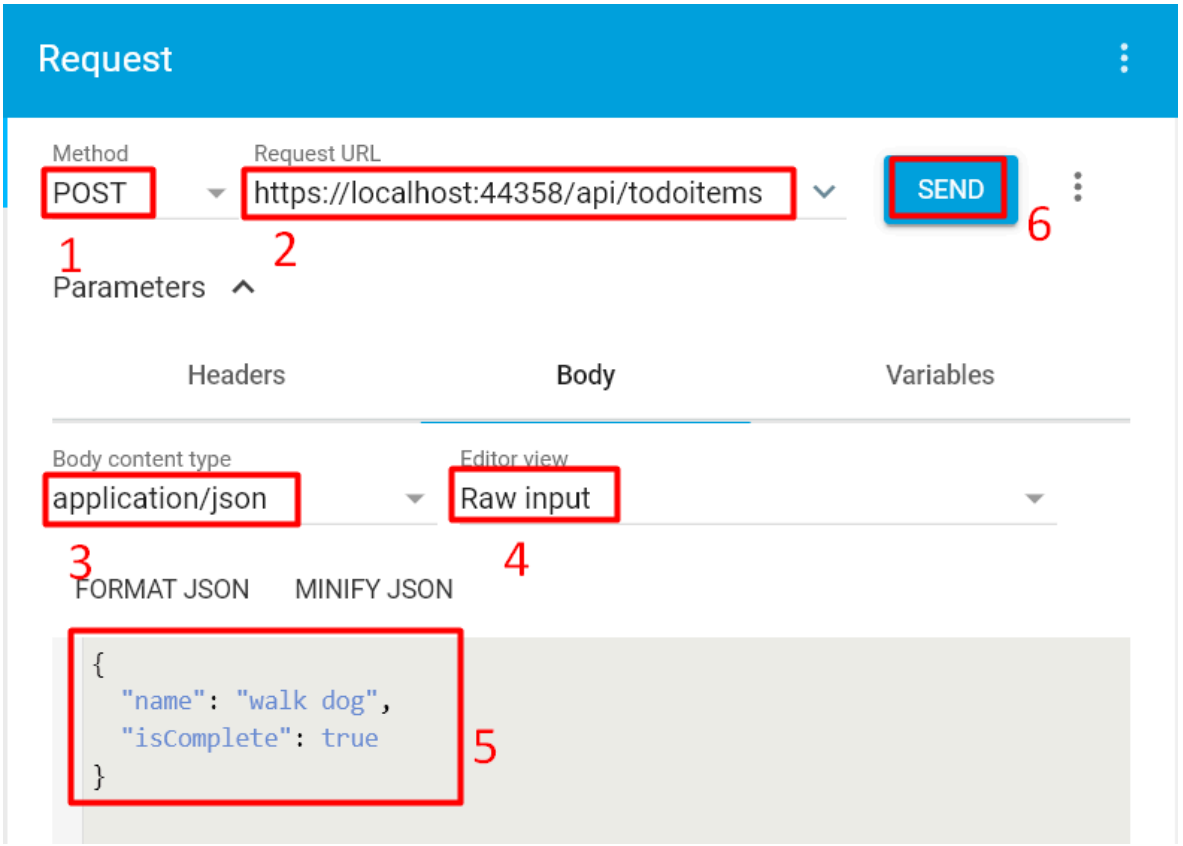
https://localhost:<port>/api/todoitems/1



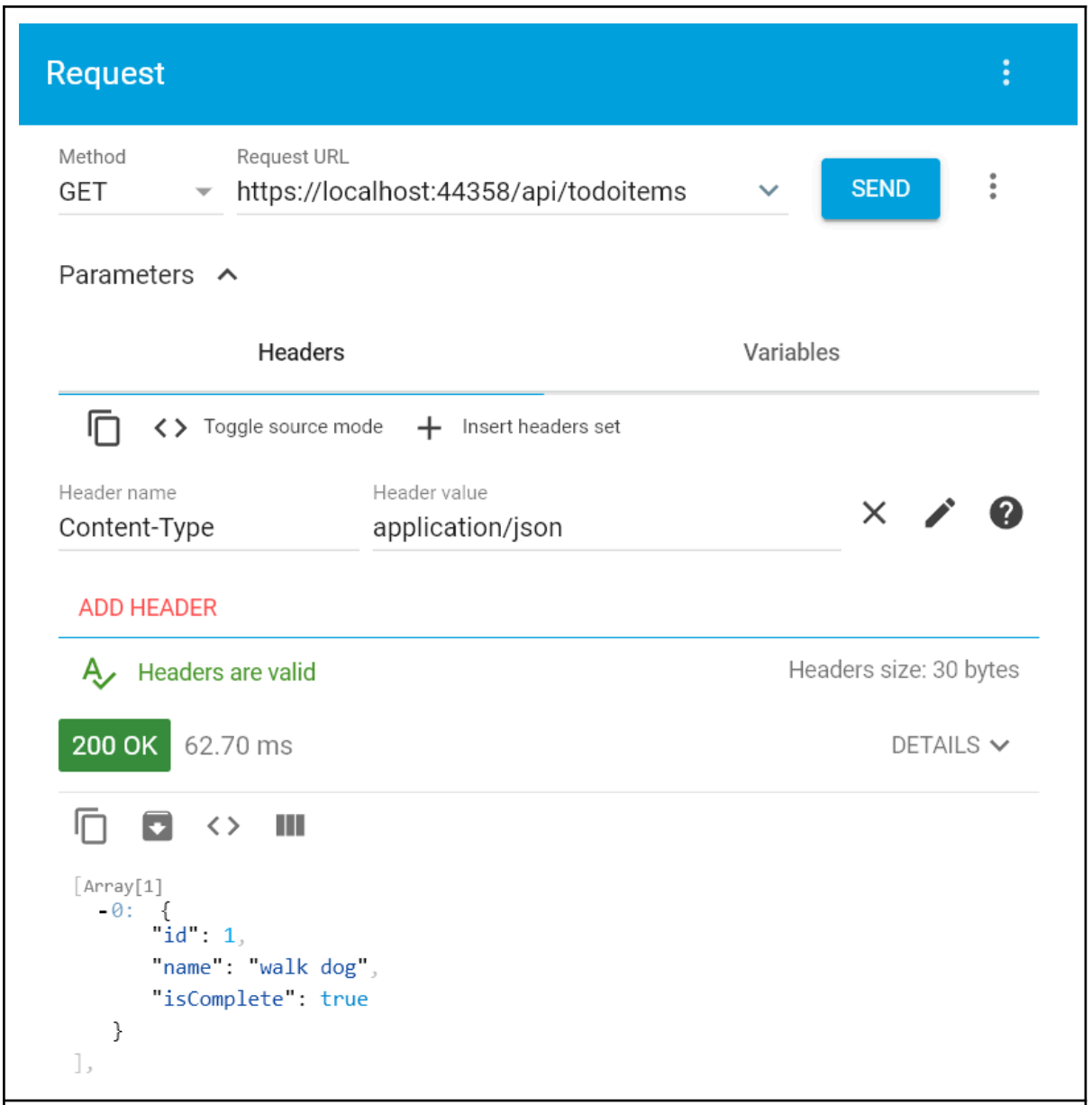
Test PostTodoItem with REST Client

Install Chrome Extension
https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo

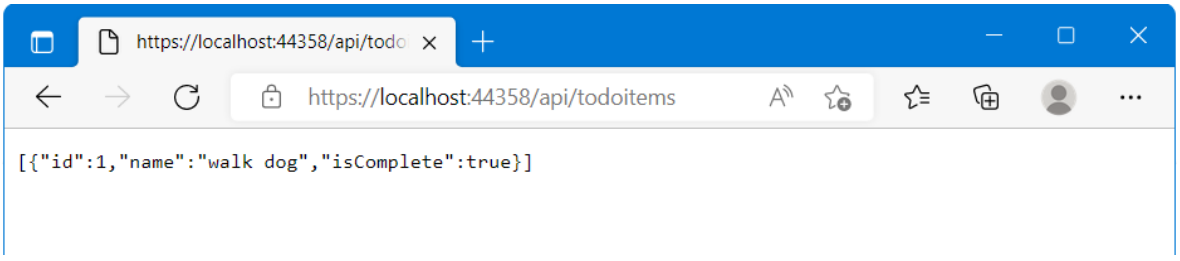
{"name":"walk dog","isComplete":true}



As a result, there will be a new item in the database.



https://localhost:<port>/api/todoitems/



Install Postman

This tutorial uses Postman to test the web API.

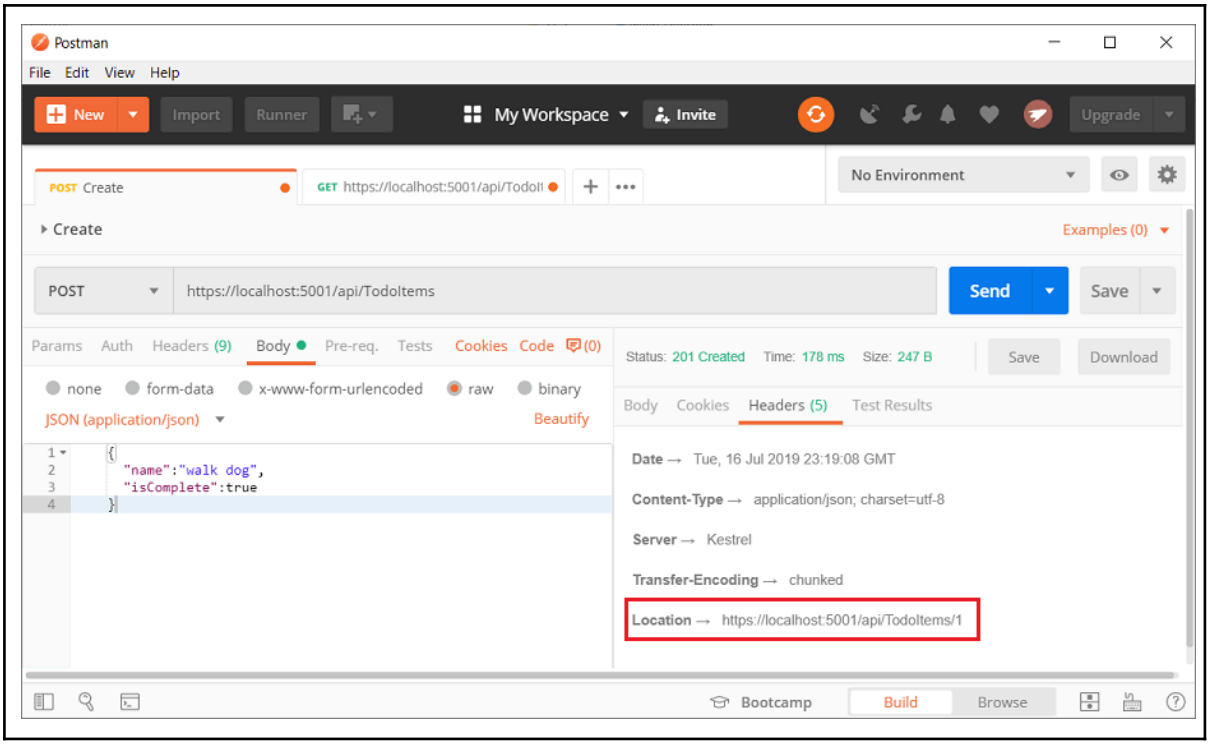
- Install Postman
- Start the web app.
- Start Postman.
- Disable SSL certificate verification:
    - Postman for Windows: Postman for Windows File > Settings (General tab), disable SSL certificate verification.
    - Postman for macOS: Postman for Windows Postman > Settings (General tab), disable SSL certificate verification.

Test PostTodoItem with Postman

- Create a new request.
- Set the HTTP method to `POST`.
- Set the URI to `https://localhost:<port>/api/todoitems`. For example, `https://localhost:5001/api/todoitems`.
- Select the Body tab.
- Select the raw radio button.
- Set the type to JSON (application/json).
- In the request body enter JSON for a to-do item:

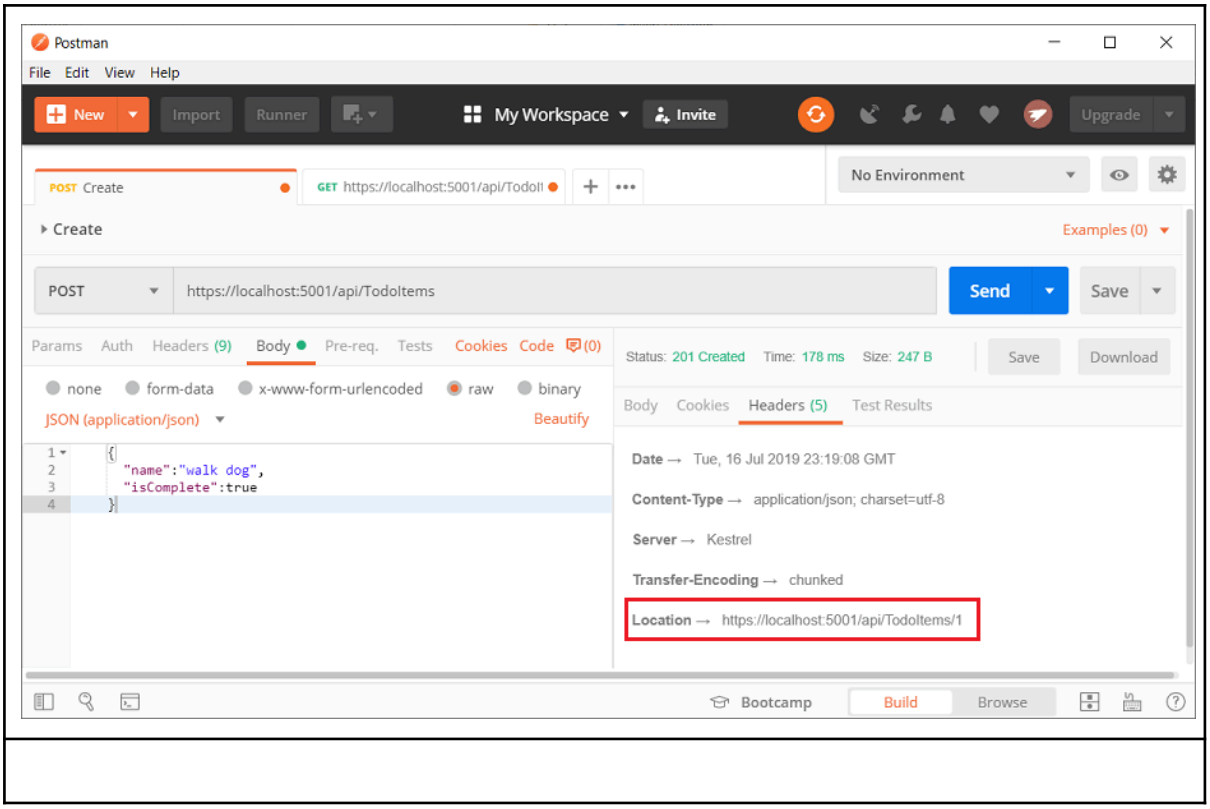
```
{
  "name":"walk dog",
  "isComplete":true
}
```

Select Send.



Test the location header URI with Postman

- Select the Headers tab in the Response pane.
- Copy the Location header value:



- Set the HTTP method to `GET`.
- Set the URI to `https://localhost:<port>/api/todoitems/1`. For example, `https://localhost:5001/api/todoitems/1`.
- Select Send.