

Criar uma API Web com o ASP.NET Core e o Visual Studio para Windows

📅 15/08/2017 • ⌚ 16 minutos para ler • Colaboradores  

Neste artigo

[Visão geral](#)

[Pré-requisitos](#)

[Criar o projeto](#)

[Registrar o contexto de banco de dados](#)

[Obtendo itens de tarefas pendentes](#)

[Implementar as outras operações CRUD](#)

[Próximas etapas](#)

Por [Rick Anderson](#) e [Mike Wasson](#)

Este tutorial compilará uma API Web para gerenciar uma lista de itens de “tarefas pendentes”. Uma interface do usuário (UI) não será criada.

Há três versões deste tutorial:

- Windows: API Web com o Visual Studio para Windows (este tutorial)
- macOS: [API Web com o Visual Studio para Mac](#)
- macOS, Linux, Windows: [API Web com o Visual Studio Code](#)

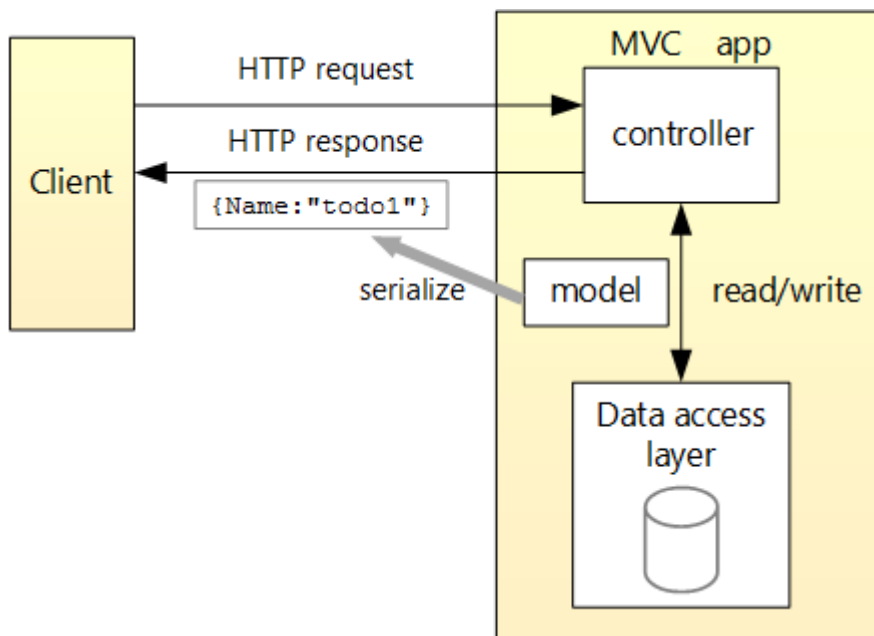
Visão geral

Este tutorial cria a seguinte API:

API	Descrição	Corpo da solicitação	Corpo da resposta
GET /api/todo	Obter todos os itens de tarefas pendentes	Nenhum	Matriz de itens de tarefas pendentes
GET /api/todo/{id}	Obter um item por ID	Nenhum	Item de tarefas pendentes
POST /api/todo	Adicionar um novo item	Item de tarefas pendentes	Item de tarefas pendentes

API	Descrição	Corpo da solicitação	Corpo da resposta
PUT /api/todo/{id}	Atualizar um item existente	Item de tarefas pendentes	Nenhum
DELETE /api/todo/{id}	Excluir um item	Nenhum	Nenhum

O diagrama a seguir mostra o design básico do aplicativo.



- O cliente é tudo o que consome a API Web (aplicativo móvel, navegador, etc.). Este tutorial não cria um cliente. [Postman](#) ou [curl](#) são usados como clientes para testar o aplicativo.
- Um *modelo* é um objeto que representa os dados no aplicativo. Nesse caso, o único modelo é um item de tarefas pendentes. Modelos são representados como classes `c#`, também conhecidos como **Plain Old C# Objeto** (POCOs).
- Um *controlador* é um objeto que manipula solicitações HTTP e cria a resposta HTTP. Este aplicativo tem um único controlador.
- Para manter o tutorial simples, o aplicativo não usa um banco de dados persistente. O aplicativo de exemplo armazena os itens de tarefas pendentes em um banco de dados em memória.

Pré-requisitos

Instale o seguinte:

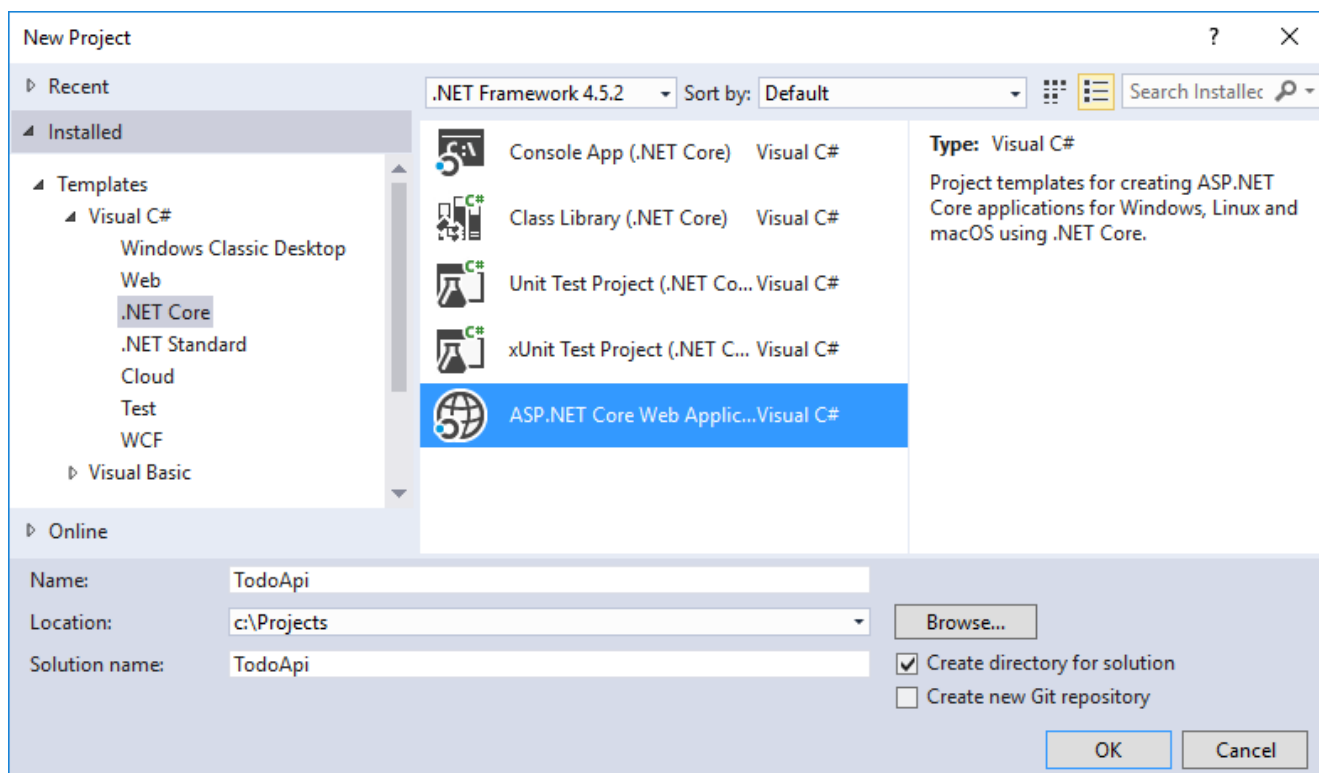
- [SDK do .NET Core 2.0.0](#) ou posterior.
- [Visual Studio 2017](#) versão 15.3 ou posterior com a carga de trabalho do **ASP.NET e desenvolvimento para a Web**.

Consulte [este PDF](#) para o ASP.NET Core versão 1.1.

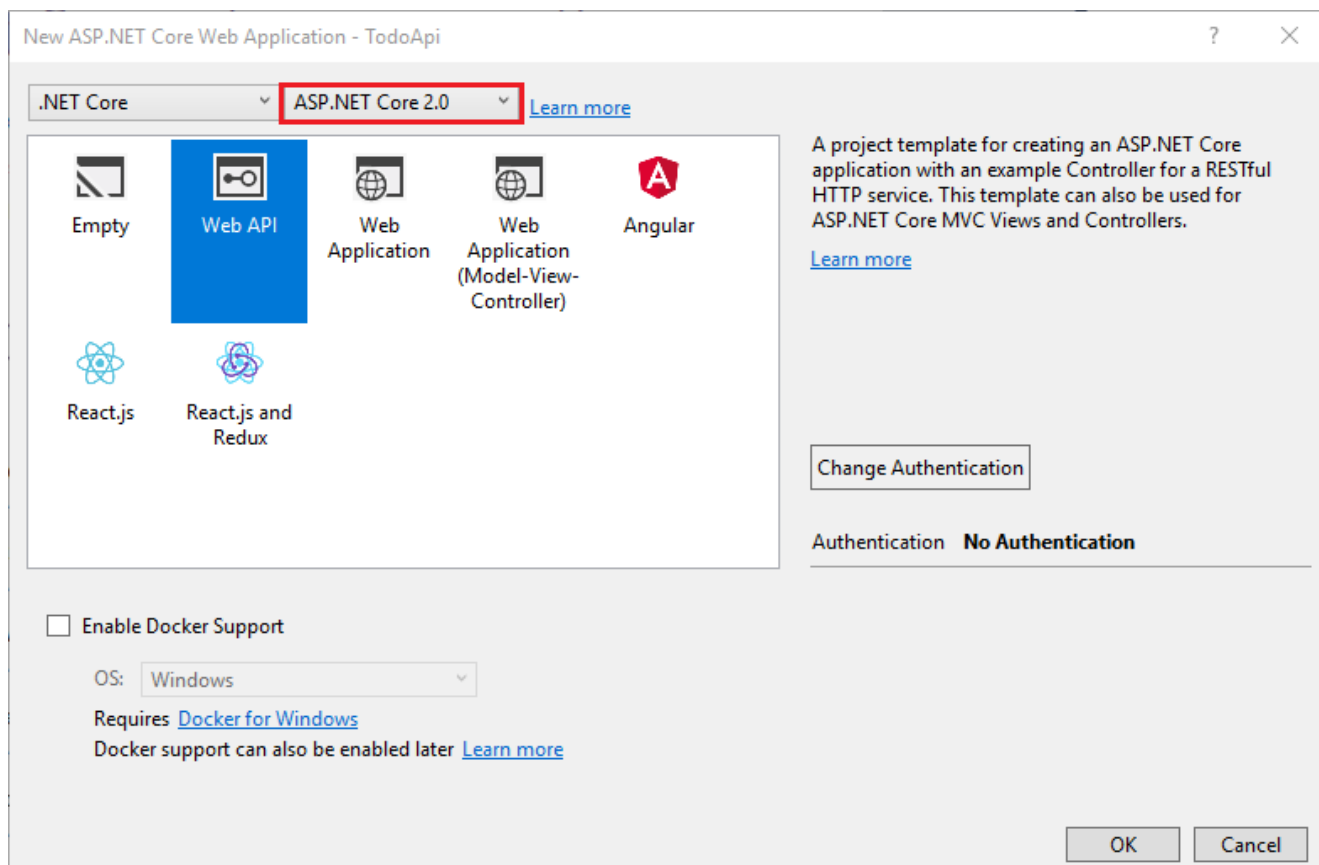
Criar o projeto

No Visual Studio, selecione o menu **Arquivo**, > **Novo** > **Projeto**.

Selecione o modelo de projeto **Aplicativo Web ASP.NET Core (.NET Core)**. Nomeie o projeto e selecione **OK**.




Na caixa de diálogo **Novo aplicativo Web ASP.NET Core – TodoApi**, selecione o modelo **API Web**. Selecione **OK**. **Não** selecione **Habilitar Suporte ao Docker**.



Iniciar o aplicativo

No Visual Studio, pressione CTRL + F5 para iniciar o aplicativo. O Visual Studio inicia um navegador e navega para `http://localhost:port/api/values`, em que *porta* é um número da porta escolhido aleatoriamente. O Chrome, Microsoft Edge e Firefox exibem a seguinte saída:

	 Copiar
["value1","value2"]	

Adicionar uma classe de modelo

Um modelo é um objeto que representa os dados no aplicativo. Nesse caso, o único modelo é um item de tarefas pendentes.

Adicione uma pasta denominada "Modelos". No Gerenciador de Soluções, clique com o botão direito do mouse no projeto. Selecione **Adicionar** > **Nova Pasta**. Nomeie a pasta *Models*.

Observação: as classes de modelo entram em qualquer lugar no projeto. A pasta *Modelos* é usada por convenção para as classes de modelo.

Adicione uma classe `TodoItem`. Clique com o botão direito do mouse na pasta *Modelos* e selecione **Adicionar** > **Classe**. Nomeie a classe `TodoItem` e, em seguida, selecione

Adicionar.

Atualize a classe `TodoItem` com o código a seguir:

C# Copiar

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

O banco de dados gera o `Id` quando um `TodoItem` é criado.

Criar o contexto de banco de dados

O *contexto de banco de dados* é a classe principal que coordena a funcionalidade do Entity Framework para um determinado modelo de dados. Essa classe é criada derivando-a da classe `Microsoft.EntityFrameworkCore.DbContext`.

Adicione uma classe `TodoContext`. Clique com o botão direito do mouse na pasta *Modelos* e selecione **Adicionar** > **Classe**. Nomeie a classe `TodoContext` e, em seguida, selecione **Adicionar**.

Substitua a classe pelo código a seguir:

C# Copiar

```
using Microsoft.EntityFrameworkCore;

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

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

Registrar o contexto de banco de dados

Nesta etapa, o contexto do banco de dados é registrado com o contêiner [injeção de dependência](#). Os serviços (como o contexto do banco de dados) que são registrados com o contêiner de injeção de dependência (DI) estão disponíveis para os controladores.

Registre o contexto de banco de dados com o contêiner de serviço usando o suporte interno para [injeção de dependência](#). Substitua o conteúdo do arquivo *Startup.cs* pelo seguinte código:

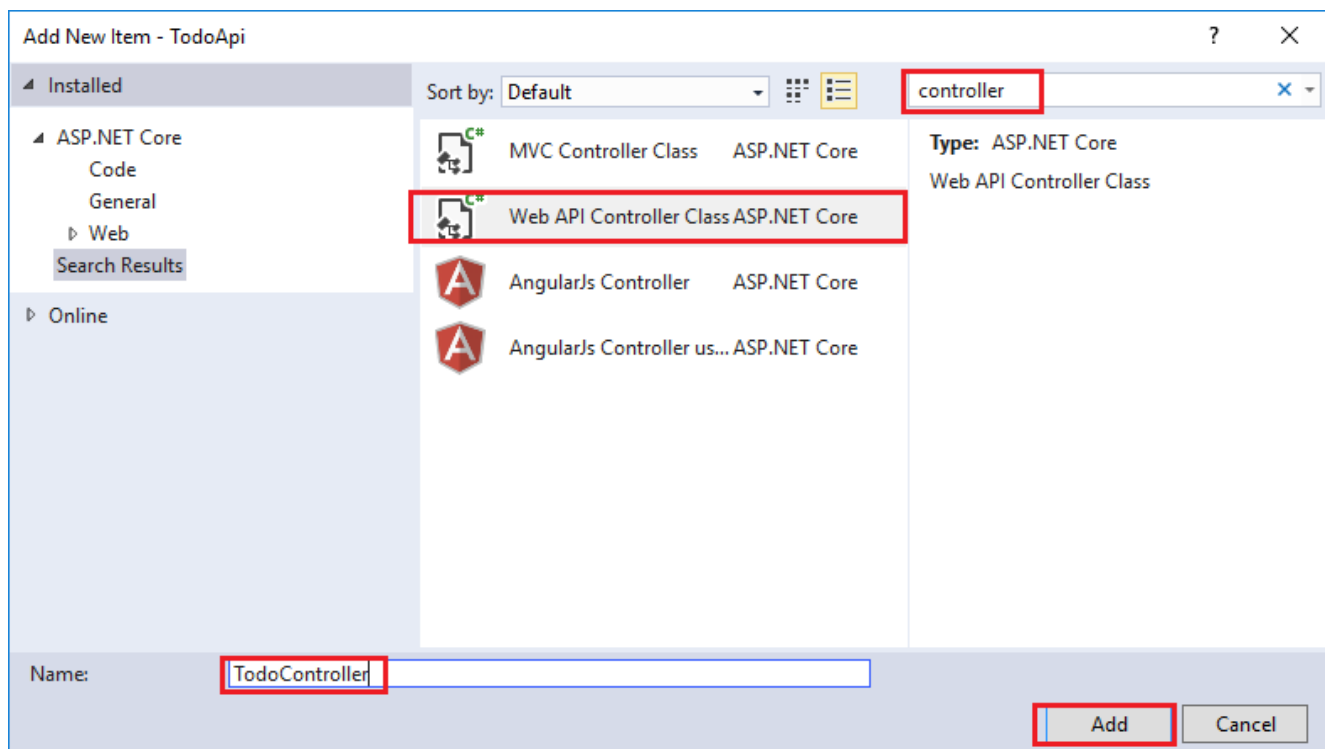
```
C# Copiar  
  
using Microsoft.AspNetCore.Builder;  
using Microsoft.EntityFrameworkCore;  
using Microsoft.Extensions.DependencyInjection;  
using TodoApi.Models;  
  
namespace TodoApi  
{  
    public class Startup  
    {  
        public void ConfigureServices(IServiceCollection services)  
        {  
            services.AddDbContext<TodoContext>(opt => opt.UseInMemoryDatabase("TodoList"));  
            services.AddMvc();  
        }  
  
        public void Configure(IApplicationBuilder app)  
        {  
            app.UseMvc();  
        }  
    }  
}
```

O código anterior:

- Remove o código que não é usado.
- Especifica que um banco de dados em memória é injetado no contêiner de serviço.

Adicionar um controlador

No Gerenciador de Soluções, clique com o botão direito do mouse na pasta *Controladores*. Selecione **Adicionar** > **Novo Item**. Na caixa de diálogo **Adicionar Novo Item**, selecione o modelo **Classe do Controlador de API Web**. Nomeie a classe `TodoController`.



Substitua a classe pelo código a seguir:

```
C# Copiar

using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using TodoApi.Models;
using System.Linq;

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

        public TodoController(TodoContext context)
        {
            _context = context;

            if (_context.TODOItems.Count() == 0)
            {
                _context.TODOItems.Add(new TodoItem { Name = "Item1" });
                _context.SaveChanges();
            }
        }
    }
}
```

O código anterior:

- Define uma classe de controlador vazia. Nas próximas seções, os métodos serão adicionados para implementar a API.
- O construtor usa a [Injeção de Dependência](#) para injetar o contexto de banco de dados (`TodoContext`) no controlador. O contexto de banco de dados é usado em cada um dos métodos [CRUD](#) no controlador.
- O construtor adiciona um item no banco de dados em memória, caso ele não exista.

Obtendo itens de tarefas pendentes

Para obter os itens de tarefas pendentes, adicione os métodos a seguir à classe

`TodoController` .

C#

 Copiar

```
[HttpGet]
public IEnumerable<TodoItem> GetAll()
{
    return _context.TODOItems.ToList();
}

[HttpGet("{id}", Name = "GetTodo")]
public IActionResult GetById(long id)
{
    var item = _context.TODOItems.FirstOrDefault(t => t.Id == id);
    if (item == null)
    {
        return NotFound();
    }
    return new ObjectResult(item);
}
```

Esses métodos de implementam os dois métodos GET:

- `GET /api/todo`
- `GET /api/todo/{id}`

Esta é uma resposta HTTP de exemplo para o método `GetAll` :

 Copiar

```
[
  {
    "id": 1,
    "name": "Item1",
    "isComplete": false
  }
]
```


Mais adiante no tutorial, mostrarei como a resposta HTTP pode ser visualizada com o [Postman](#) ou o [curl](#).

Roteamento e caminhos de URL

O atributo `[HttpGet]` especifica um método HTTP GET. O caminho da URL de cada método é construído da seguinte maneira:

- Use a cadeia de caracteres de modelo no atributo `Route` do controlador:

```
C#Copiar  
  
namespace TodoApi.Controllers  
{  
    [Route("api/[controller]")]  
    public class TodoController : Controller  
    {  
        private readonly TodoContext _context;  
    }  
}
```

- Substitua `[controller]` com o nome do controlador, que é o nome de classe do controlador menos o sufixo "Controlador". Para esta amostra, o nome de classe do controlador é **Todo**Controller e o nome da raiz é "todo". O roteamento do ASP.NET Core não diferencia maiúsculas de minúsculas.
- Se o atributo `[HttpGet]` tiver um modelo de rota (como `[HttpGet("/products")]`), acrescente isso ao caminho. Esta amostra não usa um modelo. Consulte [Roteamento de atributo com atributos Http\[verb\]](#) para obter mais informações.

No método `GetById`:

```
C#Copiar  
  
[HttpGet("{id}", Name = "GetTodo")]  
public IActionResult GetById(long id)  
{  
    var item = _context.TodoItems.FirstOrDefault(t => t.Id == id);  
    if (item == null)  
    {  
        return NotFound();  
    }  
    return new ObjectResult(item);  
}
```

`"{id}"` é uma variável de espaço reservado para a ID do item `todo`. Quando `GetById` é invocado, ele atribui o valor `"{id}"` na URL ao parâmetro `id` do método.

`Name = "GetTodo"` cria uma rota nomeada. Rotas nomeadas:

- permitem que o aplicativo crie um link HTTP usando o nome da rota.
- Serão explicadas posteriormente no tutorial.

Valores de retorno

O método `GetAll` retorna um `IEnumerable`. O MVC serializa automaticamente o objeto em [JSON](#) e grava o JSON no corpo da mensagem de resposta. O código de resposta para esse método é 200, supondo que não haja nenhuma exceção sem tratamento. (Exceções sem tratamento são convertidas em erros 5xx.)

Por outro lado, o método `GetById` retorna o tipo `ActionResult` mais geral, que representa uma ampla variedade de tipos de retorno. `GetById` tem dois tipos retornados diferentes:

- Se nenhum item corresponder à ID solicitada, o método retornará um erro 404. Retornar `NotFound` retorna uma resposta HTTP 404.
- Caso contrário, o método retornará 200 com um corpo de resposta JSON. Retornar `ObjectResult` retorna uma resposta HTTP 200.

Iniciar o aplicativo

No Visual Studio, pressione CTRL + F5 para iniciar o aplicativo. O Visual Studio inicia um navegador e navega para `http://localhost:port/api/values`, em que *porta* é um número da porta escolhido aleatoriamente. Navegue até o controlador `Todo` no `http://localhost:port/api/todo`.

Implementar as outras operações CRUD

Nas próximas seções, os métodos `Create`, `Update` e `Delete` serão adicionados ao controlador.

Create

Adicione o seguinte método `Create`.

C#

 Copiar

```
[HttpPost]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }
}
```

```

    }

    _context.TODOItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
}

```

O código anterior é um método HTTP POST, indicado pelo atributo `[HttpPost]`. O atributo `[FromBody]` informa ao MVC para obter o valor do item de tarefas pendentes do corpo da solicitação HTTP.

O método `CreatedAtRoute`:

- Retorna uma resposta 201. HTTP 201 é a resposta padrão para um método HTTP POST que cria um novo recurso no servidor.
- Adiciona um cabeçalho Local à resposta. O cabeçalho Location especifica o URI do item de tarefas pendentes recém-criado. Consulte [10.2.2 201 criado](#).
- Usa a rota chamada "GetTodo" para criar a URL. A rota chamada "GetTodo" é definida em `GetById`:

C#

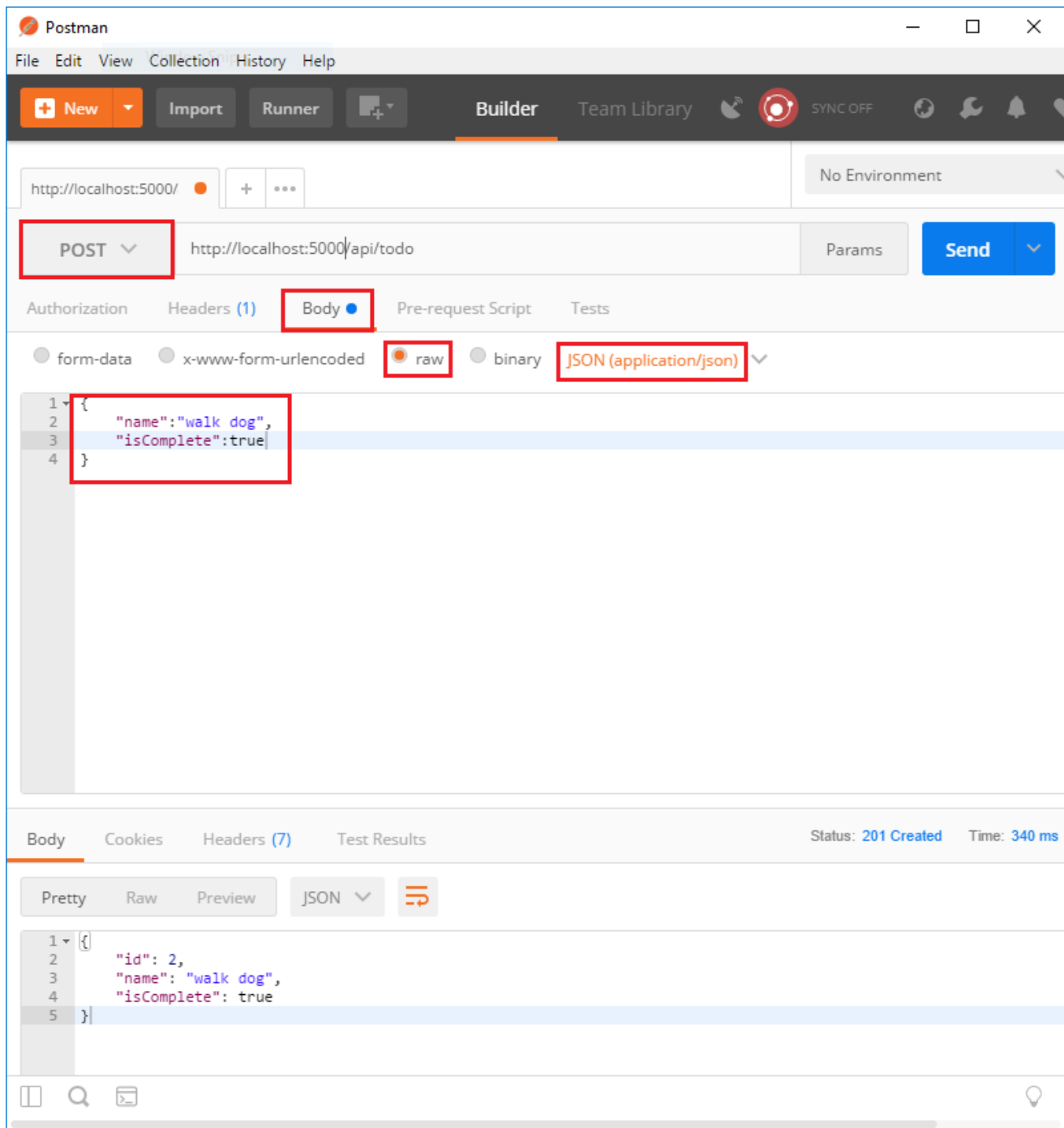
 Copiar

```


[HttpGet("{id}", Name = "GetTodo")]
public IActionResult GetById(long id)
{
    var item = _context.TODOItems.FirstOrDefault(t => t.Id == id);
    if (item == null)
    {
        return NotFound();
    }
    return new ObjectResult(item);
}

```

Usar o Postman para enviar uma solicitação Create

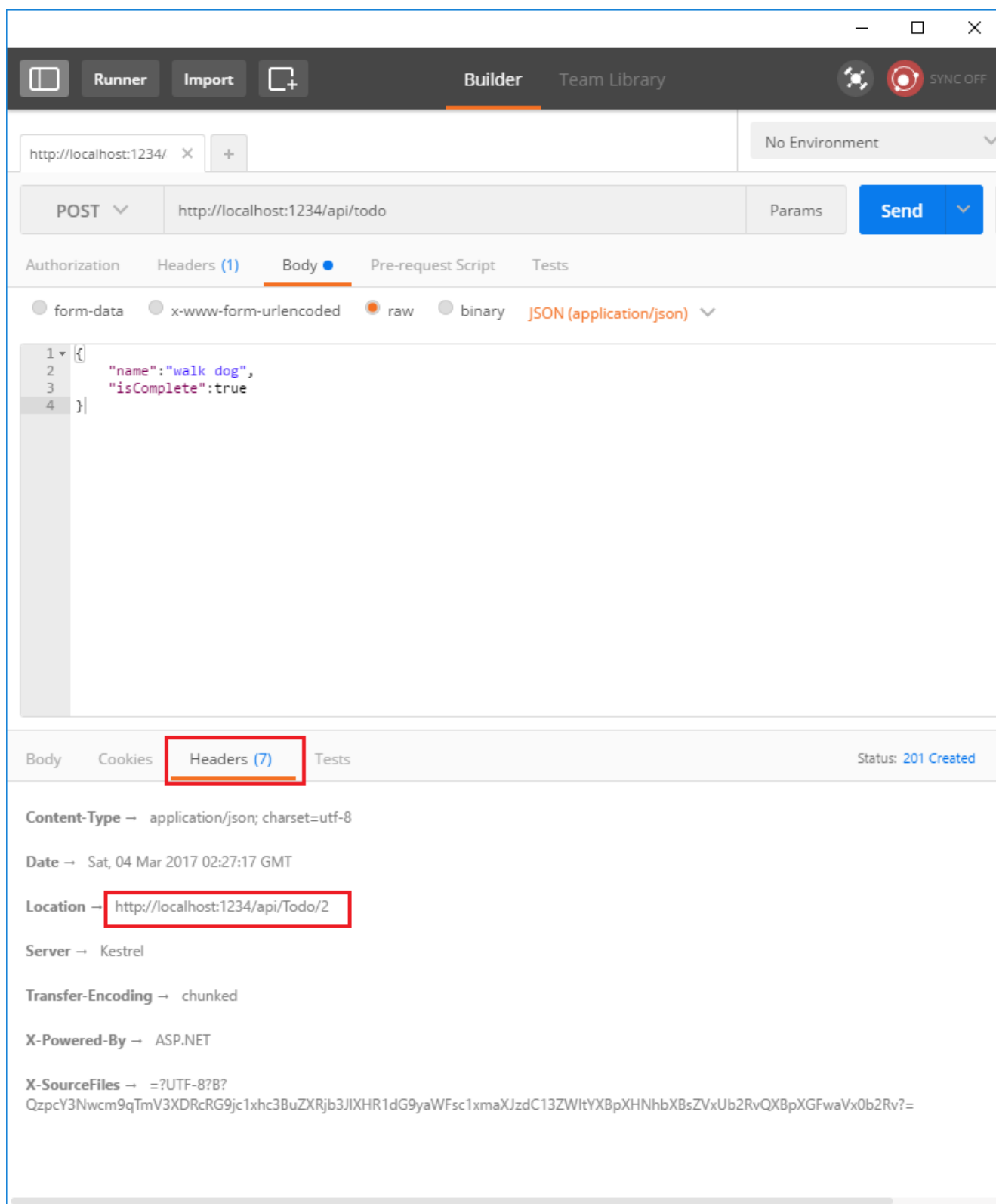


- Defina o método HTTP como **POST**
- Selecione o botão de opção **Corpo**
- Selecione o botão de opção **bruto**
- Definir o tipo como JSON
- No editor de chave-valor, insira um item de tarefas pendentes, como

JSON	 Copiar
<pre>{ "name": "walk dog", "isComplete": true }</pre>	

- Selecione **Enviar**


- Selecione a guia Cabeçalhos no painel inferior e copie o cabeçalho **Location**:



O URI do cabeçalho Local pode ser usado para acessar o novo item.

Atualização

Adicione o seguinte método `Update` :

C#	 Copiar
----	--

```

[HttpPut("{id}")]
public IActionResult Update(long id, [FromBody] TodoItem item)
{
    if (item == null || item.Id != id)
    {
        return BadRequest();
    }

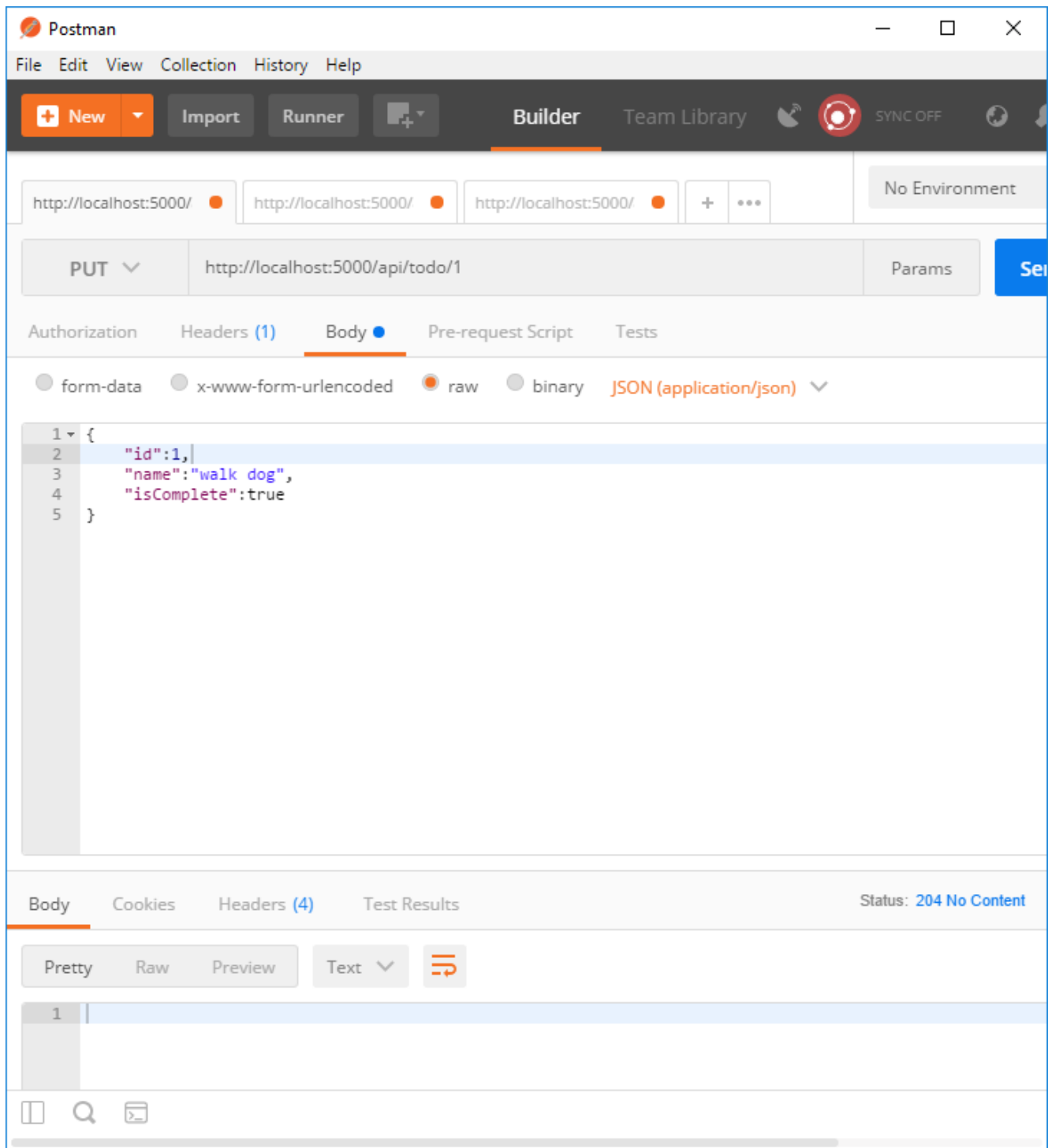
    var todo = _context.TODOItems.FirstOrDefault(t => t.Id == id);
    if (todo == null)
    {
        return NotFound();
    }

    todo.IsComplete = item.IsComplete;
    todo.Name = item.Name;

    _context.TODOItems.Update(todo);
    _context.SaveChanges();
    return new NoContentResult();
}

```

`Update` é semelhante a `Create`, mas usa HTTP PUT. A resposta é [204 \(Sem conteúdo\)](#). De acordo com a especificação HTTP, uma solicitação PUT exige que o cliente envie a entidade atualizada inteira, não apenas os deltas. Para dar suporte a atualizações parciais, use HTTP PATCH.



Excluir

Adicione o seguinte método `Delete` :

```
C# Copiar

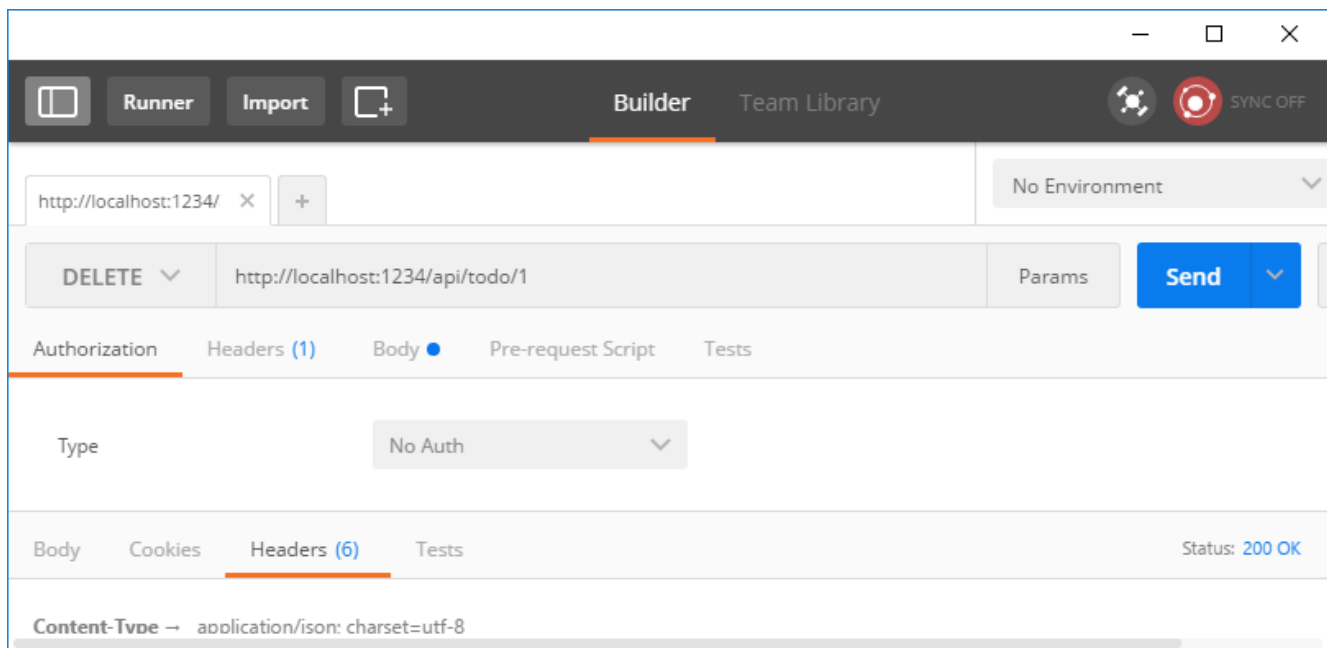
[HttpDelete("{id}")]
public IActionResult Delete(long id)
{
    var todo = _context.TODOItems.FirstOrDefault(t => t.Id == id);
    if (todo == null)
    {
        return NotFound();
    }
}
```

```
}

_context.TODOItems.Remove(todo);
_context.SaveChanges();
return new NoContentResult();
}
```

A resposta `Delete` é [204 \(Sem conteúdo\)](#).

Teste `Delete` :



Próximas etapas

- [Páginas de ajuda da API Web ASP.NET Core usando o Swagger](#)
- [Roteamento para ações do controlador](#)
- Para obter informações sobre como implantar uma API, incluindo o Serviço de Aplicativo do Azure, confira [Host e implantação](#).
- [Exibir ou baixar o código de exemplo](#). Consulte [como baixar](#).
- [Postman](#)

Note

The feedback system for this content will be changing soon. Old comments will not be carried over. If content within a comment thread is important to you, please save a copy. For more information on the upcoming change, [we invite you to read our blog post](#).