

Tutoriel : Créer une API web avec ASP.NET Core MVC

04/02/2019 • 28 minutes de lecture • Contributeurs   

Dans cet article

[Vue d'ensemble](#)

[Prérequis](#)

[Créer un projet web](#)

[Ajouter une classe de modèle](#)

[Ajouter un contexte de base de données](#)

[Inscrire le contexte de base de données](#)

[Ajouter un contrôleur](#)

[Ajouter des méthodes Get](#)

[Routage et chemins d'URL](#)

[Valeurs de retour](#)

[Tester la méthode GetTodoItems](#)

[Ajouter une méthode Create](#)

[Ajouter une méthode PutTodoItem](#)

[Ajouter une méthode DeleteTodoItem](#)

[Appeler l'API avec jQuery](#)

[Ressources supplémentaires](#)

[Étapes suivantes](#)

Par [Rick Anderson](#) et [Mike Wasson](#)

Ce tutoriel décrit les principes fondamentaux liés à la génération d'une API web avec ASP.NET Core.

Dans ce didacticiel, vous apprendrez à :

- ✓ Créer un projet d'API web.
- ✓ Ajouter une classe de modèle
- ✓ Créer le contexte de base de données
- ✓ Inscrire le contexte de base de données
- ✓ Ajouter un contrôleur
- ✓ Ajouter les méthodes CRUD

- ✓ Configurer le routage et les chemins d'URL
- ✓ Spécifier des valeurs de retour
- ✓ Appeler l'API web avec Postman
- ✓ Appeler l'API web avec jQuery.

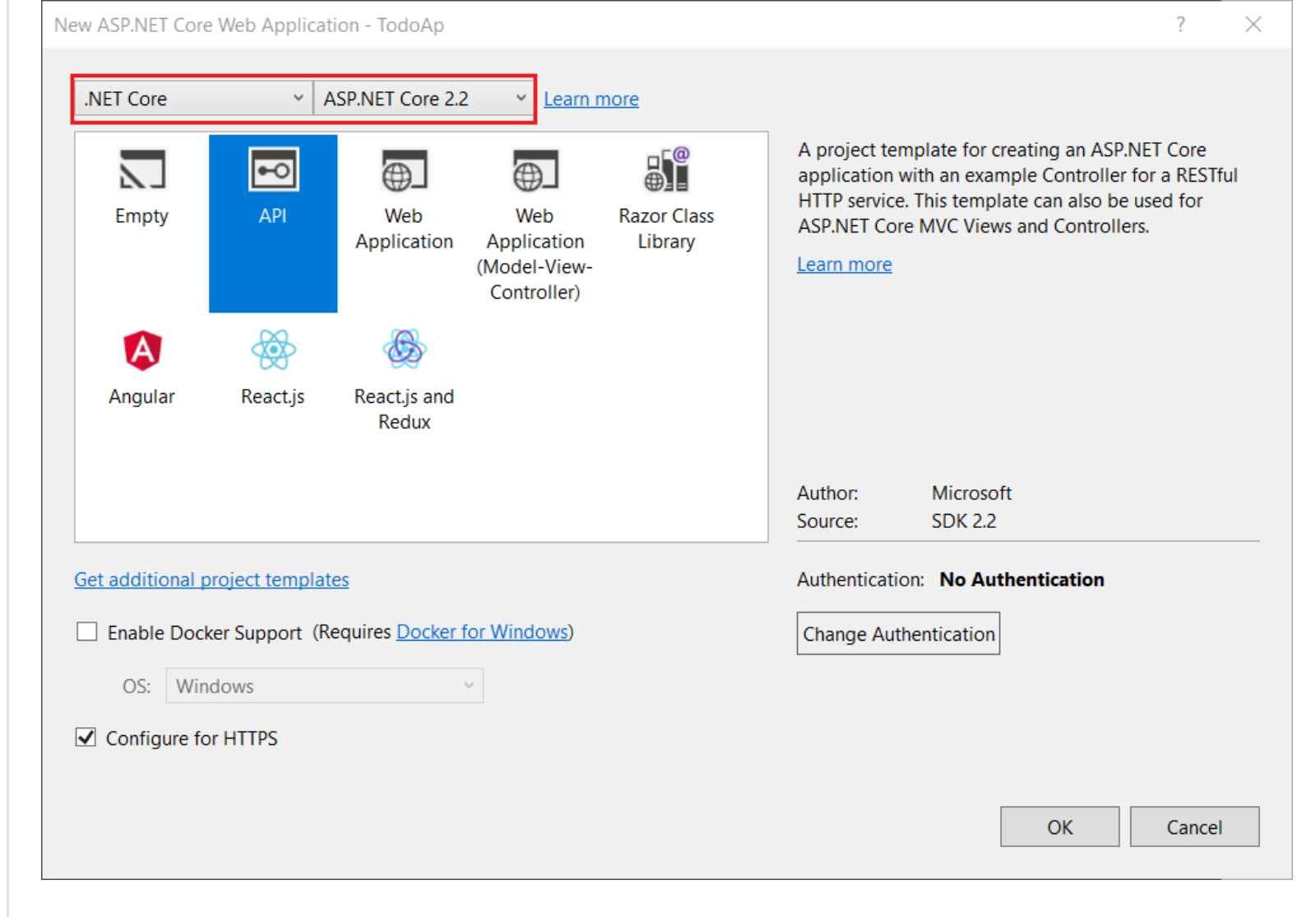
À la fin, vous disposez d'une API web qui peut gérer des tâches stockées dans une base de données relationnelle.

Vue d'ensemble

Ce didacticiel crée l'API suivante :

API	Description	Corps de la requête	Corps de réponse
GET /api/todo	Obtenir toutes les tâches	Aucun.	Tableau de tâches
GET /api/todo/{id}	Obtenir un élément par ID	Aucun.	Tâche
POST /api/todo	Ajouter un nouvel élément	Tâche	Tâche
PUT /api/todo/{id}	Mettre à jour un élément existant	Tâche	Aucun.
DELETE /api/todo/{id}	Supprimer un élément	Aucun.	Aucun.

Le diagramme suivant illustre la conception de l'application.



Tester l'API

Le modèle de projet crée une API `values`. Appelez la méthode `Get` à partir d'un navigateur pour tester l'application.

Visual Studio Visual Studio Code Visual Studio pour Mac

Appuyez sur Ctrl+F5 pour exécuter l'application. Visual Studio lance un navigateur et accède à `https://localhost:<port>/api/values`, où `<port>` est un numéro de port choisi de manière aléatoire.

Si une boîte de dialogue apparaît vous demandant si vous devez approuver le certificat IIS Express, sélectionnez **Oui**. Dans la boîte de dialogue **Avertissement de sécurité** qui s'affiche ensuite, sélectionnez **Oui**.

Le code JSON suivant est retourné :

JSON

Copier

JSON

Copier

```
["value1","value2"]
```

Ajouter une classe de modèle

Un *modèle* est un ensemble de classes qui représentent les données gérées par l'application. Le modèle pour cette application est une classe `TodoItem` unique.

Visual Studio

Visual Studio Code

Visual Studio pour Mac

- Dans l'**Explorateur de solutions**, cliquez avec le bouton droit sur le projet. Sélectionnez **Ajouter > Nouveau dossier**. Nommez le dossier *Models*.
- Cliquez avec le bouton droit sur le dossier *Models* et sélectionnez **Ajouter > Classe**. Nommez la classe *TodoItem* et sélectionnez sur **Ajouter**.
- Remplacez le code du modèle par le code suivant :

C#

 Copier

C#

 Copier

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

La propriété `Id` fonctionne comme la clé unique dans une base de données relationnelle.

Vous pouvez placer des classes de modèle n'importe où dans le projet, mais le dossier *Models* est utilisé par convention.

Ajouter un contexte de base de données

Le *contexte de base de données* est la classe principale qui coordonne les

fonctionnalités d'Entity Framework pour un modèle de données. Cette classe est créée en dérivant de la classe `Microsoft.EntityFrameworkCore.DbContext`.

Visual Studio

Visual Studio Code / Visual Studio pour Mac

- Cliquez avec le bouton droit sur le dossier *Models* et sélectionnez **Ajouter > Classe**. Nommez la classe *TodoContext* et cliquez sur **Ajouter**.

- Remplacez le code du modèle par le code suivant :

C#

 Copier

C#

 Copier

```
using Microsoft.EntityFrameworkCore;

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

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

Inscrire le contexte de base de données

Dans ASP.NET Core, les services tels que le contexte de base de données doivent être inscrits auprès du [conteneur d'injection de dépendances](#). Le conteneur fournit le service aux contrôleurs.

Mettez à jour *Startup.cs* avec le code en surbrillance suivant :

C#

 Copier

C#

 Copier

```
// Unused usings removed
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using TodoApi.Models;

namespace TodoApi
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method
        // to add services to the
        // container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<TodoContext>(opt =>
                opt.UseInMemoryDatabase("TodoList"));
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
        }

        // This method gets called by the runtime. Use this method
        // to configure the HTTP
        // request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                // The default HSTS value is 30 days. You may want
                // to change this for
                // production scenarios, see https://aka.ms/aspnet-
                // core-hsts.
                app.UseHsts();
            }
        }
    }
}
```

```

    app.UseHttpsRedirection();
    app.UseMvc();
}
}
}

```

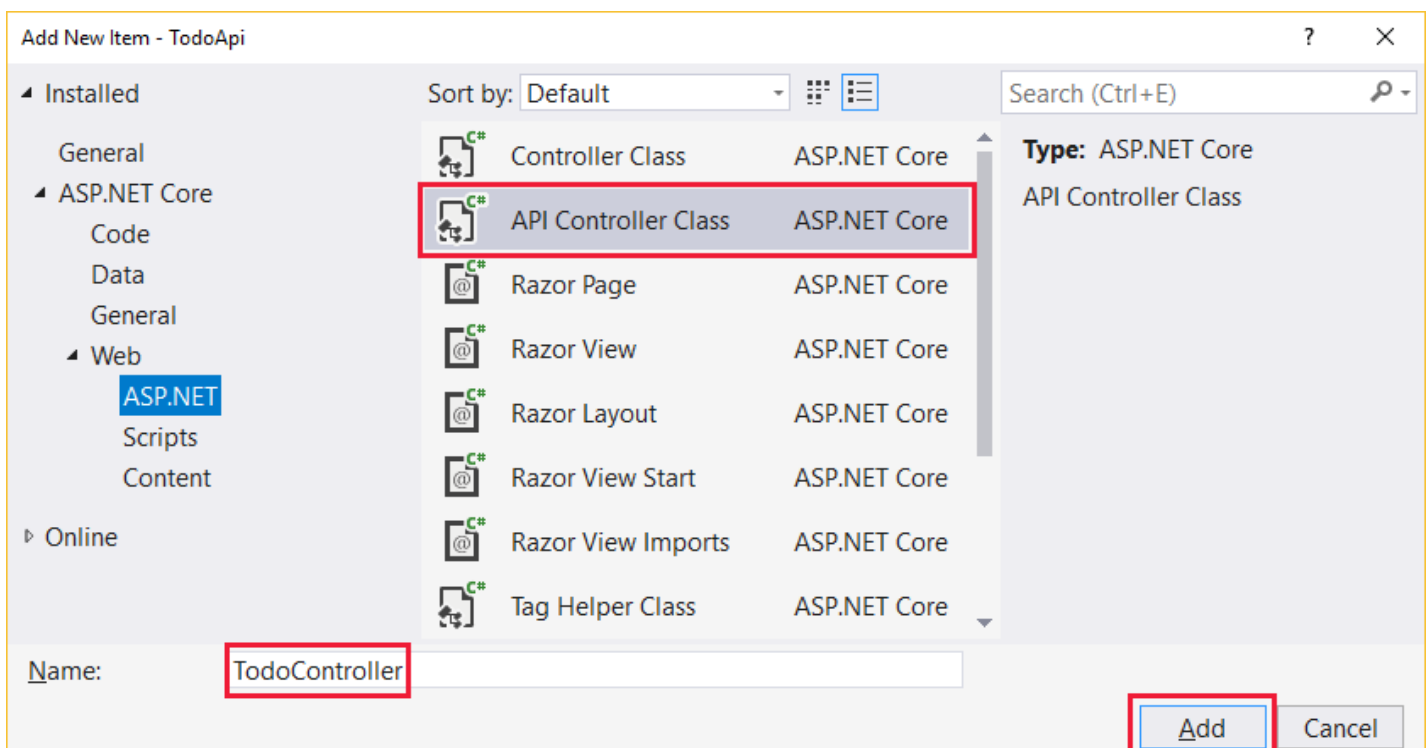
Le code précédent :

- Supprime les déclarations `using` inutilisées.
- Ajoute le contexte de base de données au conteneur d'injection de dépendances.
- Spécifie que le contexte de base de données utilise une base de données en mémoire.

Ajouter un contrôleur

Visual Studio Visual Studio Code / Visual Studio pour Mac

- Cliquez avec le bouton droit sur le dossier *Contrôleurs*.
- Sélectionnez **Ajouter > Nouvel élément**.
- Dans la boîte de dialogue **Ajouter un nouvel élément**, sélectionnez le modèle **Classe de contrôleur d'API**.
- Nommez la classe *TodoController* et sélectionnez **Ajouter**.



- Remplacez le code du modèle par le code suivant :

C#

 Copier

C#

 Copier

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

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

        public TodoController(TodoContext context)
        {
            _context = context;

            if (_context.TODOItems.Count() == 0)
            {
                // Create a new TodoItem if collection is empty,
                // which means you can't delete all TODOItems.
                _context.TODOItems.Add(new TodoItem { Name =
                "Item1" });
                _context.SaveChanges();
            }
        }
    }
}
```

Le code précédent :

- Définit une classe de contrôleur d'API sans méthodes.
- Décorez la classe avec l'attribut `[ApiController]`. Cet attribut indique que le contrôleur répond aux requêtes de l'API web. Pour plus d'informations sur les comportements spécifiques que permet l'attribut, consultez [Annotation avec attribut ApiController](#).
- Utilise l'injection de dépendances pour injecter le contexte de base de données

(`TodoContext`) dans le contrôleur. Le contexte de base de données est utilisé dans chacune des méthodes la [CRUD](#) du contrôleur.

- Ajoute un élément nommé `Item1` à la base de données si celle-ci est vide. Ce code se trouvant dans le constructeur, il s'exécute chaque fois qu'une nouvelle requête HTTP existe. Si vous supprimez tous les éléments, le constructeur recrée `Item1` au prochain appel d'une méthode d'API. Ainsi, il peut vous sembler à tort que la suppression n'a pas fonctionné.

Ajouter des méthodes Get

Pour fournir une API qui récupère les tâches, ajoutez les méthodes suivantes à la classe `TodoController` :

C#

 Copier

C#

 Copier

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

// GET: api/ToDo/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;
}
```

Ces méthodes implémentent deux points de terminaison GET :

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

Testez l'application en appelant les deux points de terminaison à partir d'un navigateur. Par exemple :

- `https://localhost:<port>/api/todo`
- `https://localhost:<port>/api/todo/1`



La réponse HTTP suivante est générée par l'appel à `GetTodoItems` :

JSON	 Copier
JSON	 Copier
<pre>[{ "id": 1, "name": "Item1", "isComplete": false }]</pre>	

Routage et chemins d'URL

L'attribut `[HttpGet]` désigne une méthode qui répond à une requête HTTP GET. Le chemin d'URL pour chaque méthode est construit comme suit :

- Partez de la chaîne de modèle dans l'attribut `Route` du contrôleur :

C#	 Copier
C#	 Copier
<pre>namespace TodoApi.Controllers { [Route("api/[controller]")] [ApiController] public class TodoController : ControllerBase { private readonly TodoContext _context;</pre>	


- Remplacez `[controller]` par le nom du contrôleur qui, par convention, est le nom de la classe du contrôleur sans le suffixe « Controller ». Pour cet exemple, le nom de classe du contrôleur étant `TodoController`, le nom du contrôleur est

« todo ». Le [routage](#) d'ASP.NET Core ne respecte pas la casse.

- Si l'attribut `[HttpGet]` a un modèle de route (par exemple, `[HttpGet("products")]`), ajoutez-le au chemin. Cet exemple n'utilise pas de modèle. Pour plus d'informations, consultez [Routage par attributs avec des attributs Http\[Verbe\]](#).

Dans la méthode `GetTodoItem` suivante, `"{id}"` est une variable d'espace réservé pour l'identificateur unique de la tâche. Quand `GetTodoItem` est appelée, la valeur de `"{id}"` dans l'URL est fournie à la méthode dans son paramètre `id`.

C#

 Copier

C#

 Copier

```
// GET: api/Todo/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;
}
```

Valeurs de retour

Le type de retour des méthodes `GetTodoItems` et `GetTodoItem` est [type `ActionResult<T>`](#). ASP.NET Core sérialise automatiquement l'objet en [JSON](#) et écrit le JSON dans le corps du message de réponse. Le code de réponse pour ce type de retour est 200, en supposant qu'il n'existe pas d'exception non gérée. Les exceptions non gérées sont converties en erreurs 5xx.

Les types de retour `ActionResult` peuvent représenter une large plage de codes d'état HTTP. Par exemple, `GetTodoItem` peut retourner deux valeurs d'état différentes :

- Si aucun élément ne correspond à l'ID demandé, la méthode retourne un code

d'erreur 404 [introuvable](#).

- Sinon, la méthode retourne 200 avec un corps de réponse JSON. Le retour de `item` entraîne une réponse HTTP 200.

Tester la méthode GetTodoItems

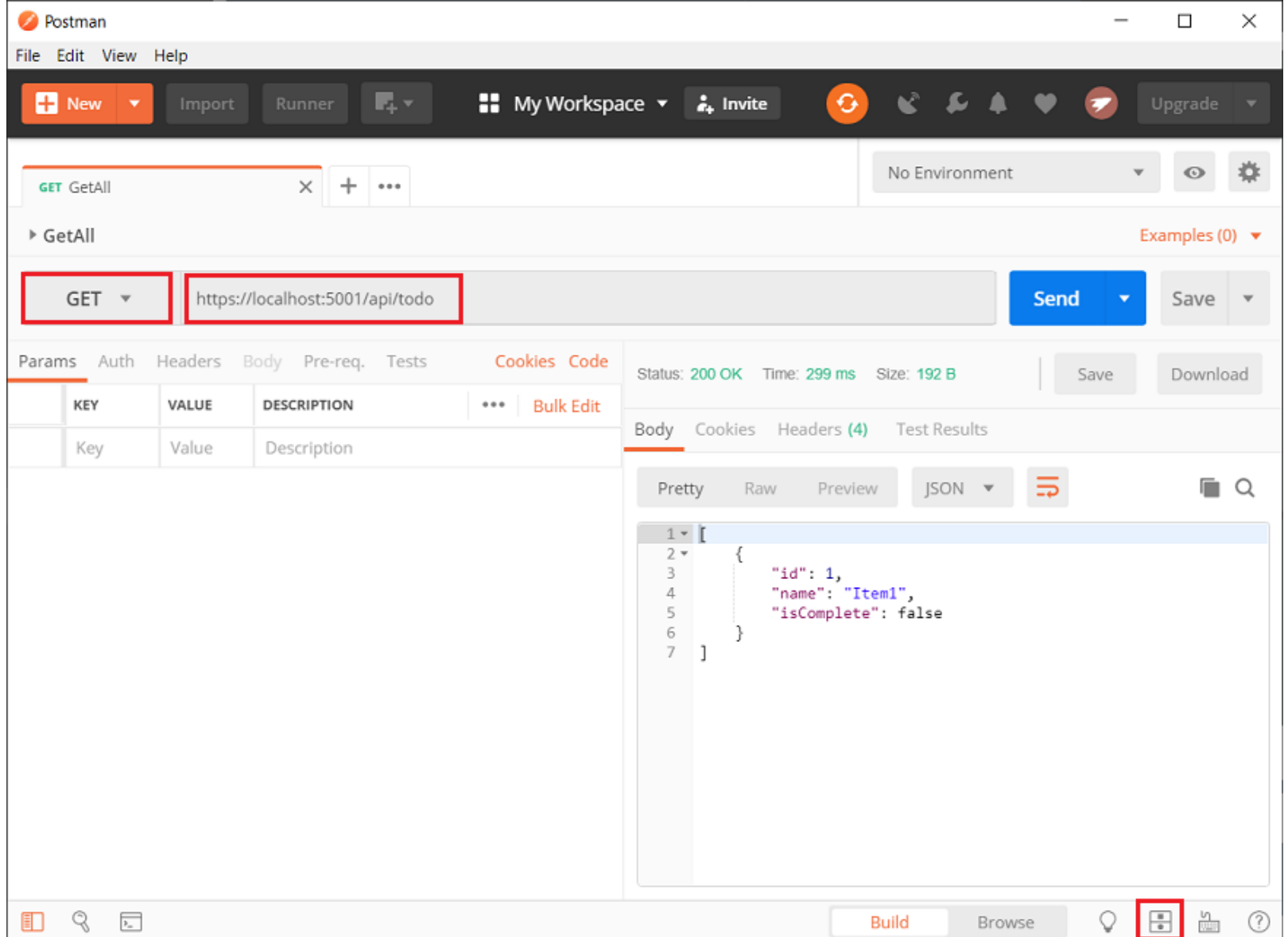
Ce tutoriel utilise Postman pour tester l'API web.

- Installez [Postman](#).
- Démarrez l'application web.
- Démarrez Postman.
- Désactivez la **vérification du certificat SSL**.
 - À partir de **Fichier > Paramètres** (onglet **Général*), désactivez **Vérification du certificat SSL**.

Avertissement

Réactivez la vérification du certificat SSL après avoir testé le contrôleur.

- Créez une requête.
 - Définissez la méthode HTTP sur **GET**.
 - Définissez l'URL de la requête sur `https://localhost:<port>/api/todo`.
Par exemple, `https://localhost:5001/api/todo`.
- Définissez l'**affichage à deux volets** dans Postman.
- Sélectionnez **Send** (Envoyer).



Ajouter une méthode Create

Ajoutez la méthode `PostTodoItem` suivante :

C#

 Copier

C#

 Copier

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

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

Le code précédent est une méthode HTTP POST, comme indiqué par l'attribut [\[HttpPost\]](#). La méthode obtient la valeur de la tâche dans le corps de la requête

HTTP.

La méthode `CreatedAtAction` :

- Retourne un code d'état HTTP 201 en cas de réussite. HTTP 201 est la réponse standard d'une méthode HTTP POST qui crée une ressource sur le serveur.
- Ajoute un en-tête `Location` à la réponse. L'en-tête `Location` spécifie l'URI de l'élément d'action qui vient d'être créé. Pour plus d'informations, consultez la section [10.2.2 201 Created](#).
- Fait référence à l'action `GetTodoItem` pour créer l'URI `Location` de l'en-tête. Le mot clé `nameof` C# est utilisé pour éviter de coder en dur le nom de l'action dans l'appel `CreatedAtAction`.

C#

 Copier

C#

 Copier

```
// GET: api/Todo/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;
}
```

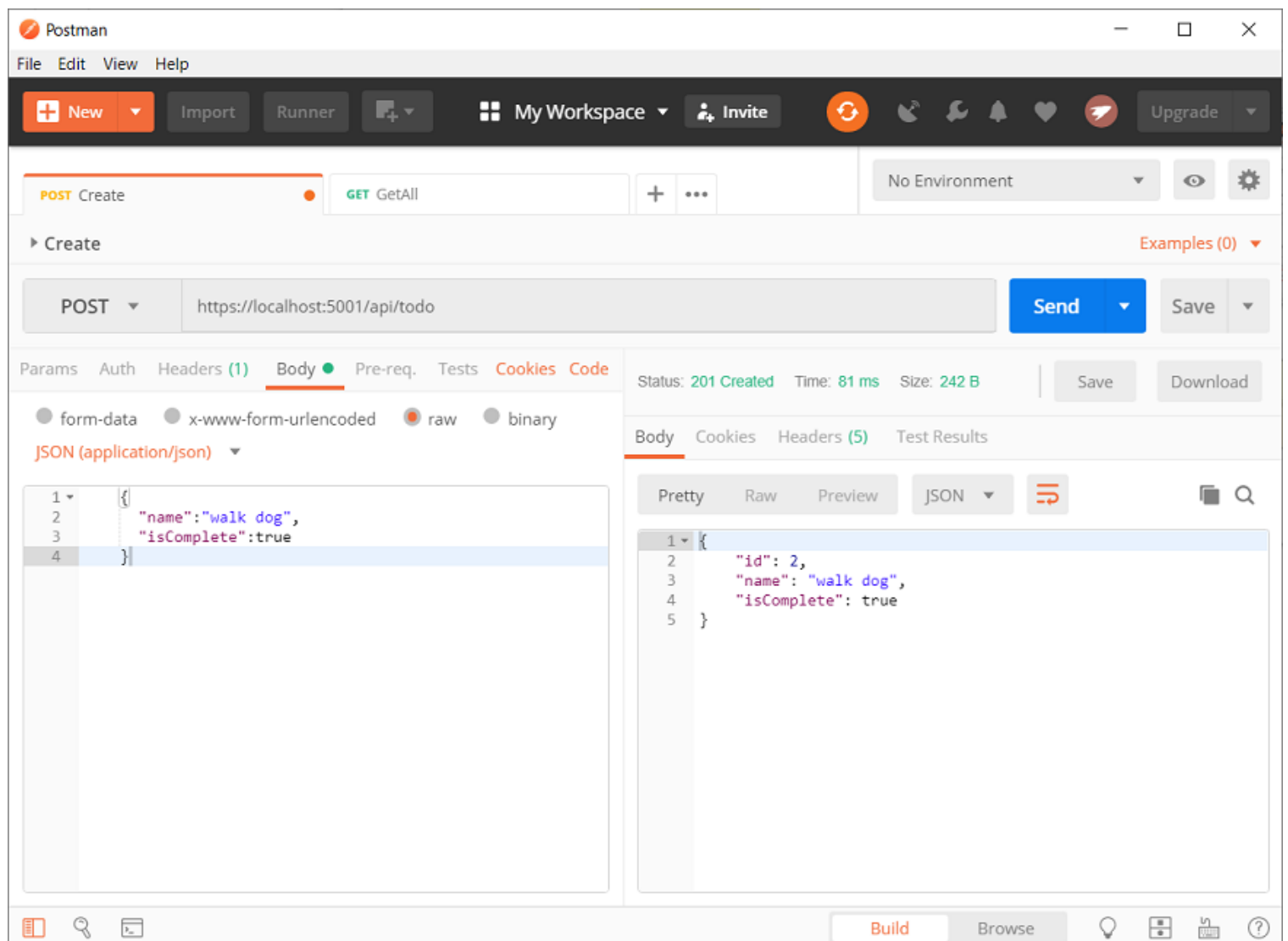
Tester la méthode PostTodoItem

- Générez le projet.
- Dans Postman, définissez la méthode HTTP sur `POST`.
- Sélectionnez l'onglet **Body** (Corps).
- Sélectionnez la case d'option **raw** (données brutes).
- Définissez le type sur **JSON** (`application/json`).

- Dans le corps de la demande, entrez la syntaxe JSON d'une tâche :

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

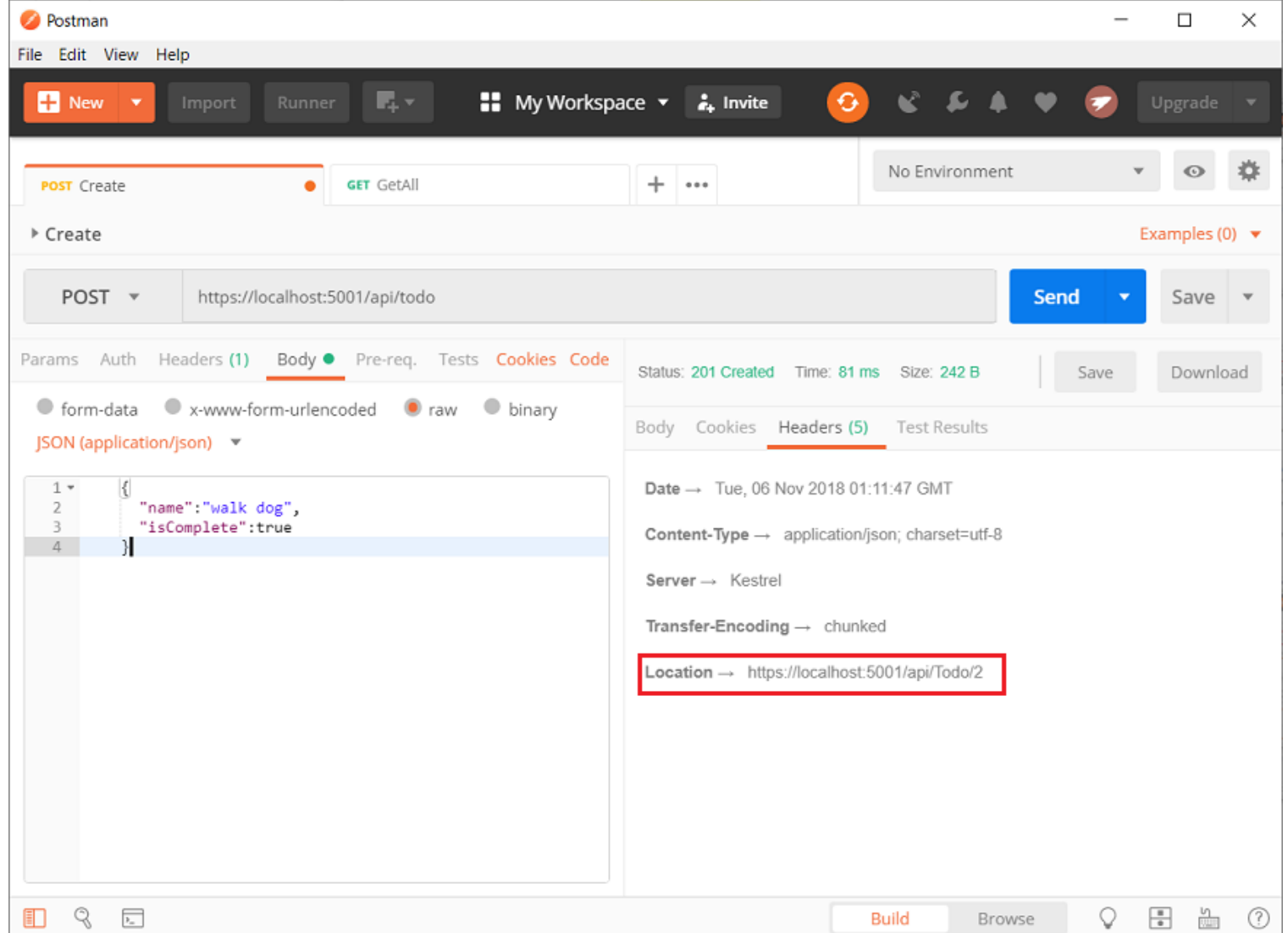
- Sélectionnez **Send** (Envoyer).



Si vous obtenez une erreur 405 Méthode non autorisée, il est probable que le projet n'ait pas été compilé après l'ajout de la méthode `PostTodoItem`.

Tester l'URI de l'en-tête d'emplacement

- Sélectionnez l'onglet **Headers** (En-têtes) dans le volet **Response** (Réponse).
- Copiez la valeur d'en-tête **Location** (Emplacement) :



- Définissez la méthode sur GET.
- Collez l'URI (par exemple, `https://localhost:5001/api/ToDo/2`).
- Sélectionnez **Send** (Envoyer).

Ajouter une méthode PutTodoItem

Ajoutez la méthode `PutTodoItem` suivante :

C#

Copier

C#

Copier

```
// PUT: api/ToDo/5
[HttpPut("{id}")]
public async Task<IActionResult> PutTodoItem(long id, TodoItem
item)
{
    if (id != item.Id)
    {
        return BadRequest();
    }
}
```

```
    _context.Entry(item).State = EntityState.Modified;
    await _context.SaveChangesAsync();

    return NoContent();
}
```

`PutTodoItem` est similaire à `PostTodoItem`, à la différence près qu'il utilise HTTP PUT. La réponse est [204 \(Pas de contenu\)](#). D'après la spécification HTTP, une requête PUT nécessite que le client envoie toute l'entité mise à jour, et pas seulement les changements. Pour prendre en charge les mises à jour partielles, utilisez [HTTP PATCH](#).

Si vous obtenez une erreur en appelant `PutTodoItem`, appelez `GET` pour vérifier que la base de données contient un élément.

Tester la méthode `PutTodoItem`

Cet exemple utilise une base de données en mémoire qui doit être initialisée à chaque démarrage de l'application. La base de données doit contenir un élément avant que vous ne passiez un appel PUT. Appelez GET afin de garantir qu'un élément existe dans la base de données avant d'effectuer un appel PUT.

Mettez à jour la tâche dont l'id est 1 en définissant son nom sur « feed fish » :

JSON

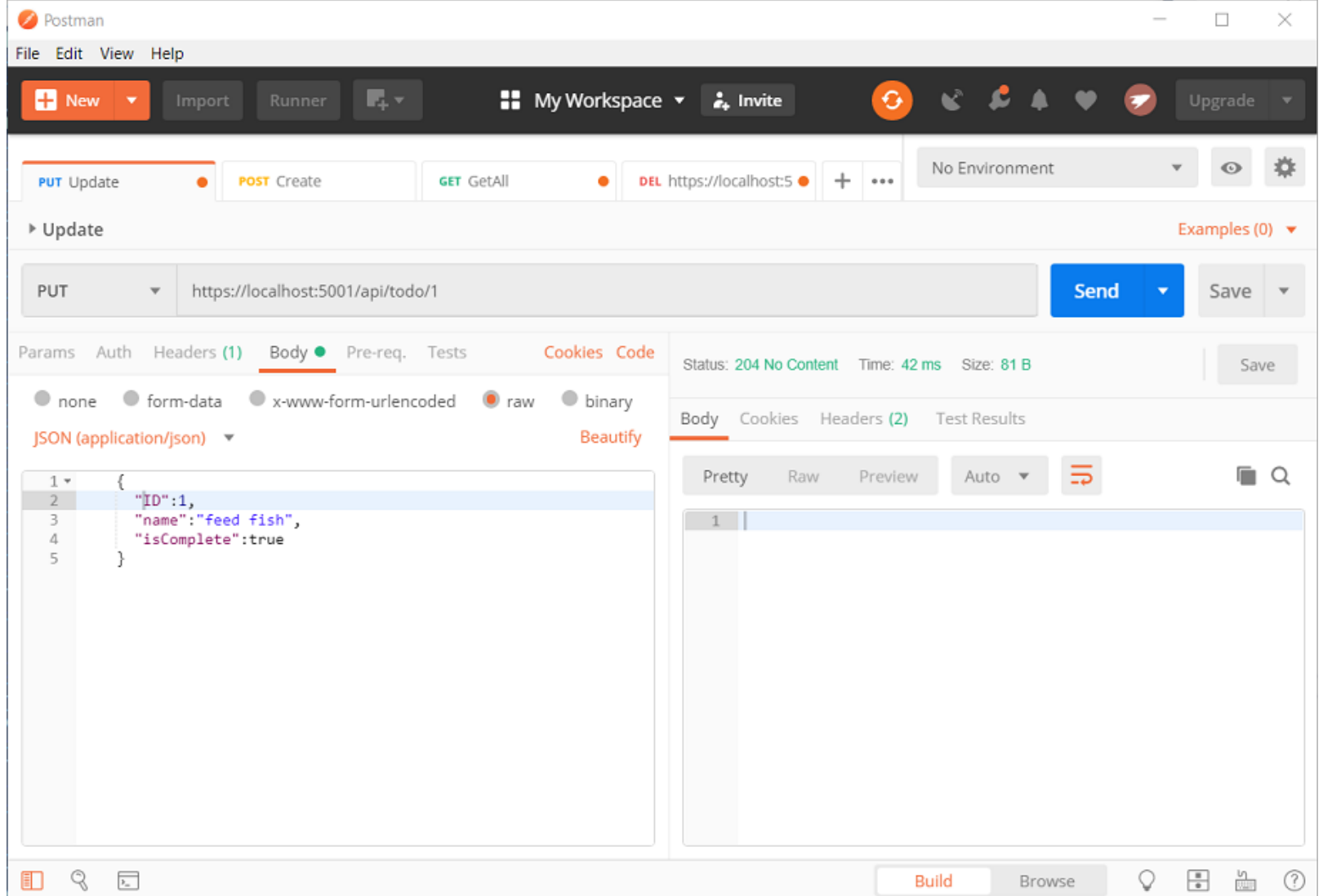
 Copier

JSON

 Copier

```
{
  "ID":1,
  "name":"feed fish",
  "isComplete":true
}
```

L'image suivante montre la mise à jour Postman :



Ajouter une méthode DeleteTodoItem

Ajoutez la méthode `DeleteTodoItem` suivante :

C#

 Copier

C#

 Copier

```
// DELETE: api/ToDo/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTodoItem(long id)
{
    var todoItem = await _context.TODOItems.FindAsync(id);

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

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

    return NoContent();
}
```

La réponse `DeleteTodoItem` est [204 \(Pas de contenu\)](#).

Tester la méthode `DeleteTodoItem`

Utilisez Postman pour supprimer une tâche :

- Définissez la méthode sur `DELETE` .
- Définissez l'URI de l'objet à supprimer, par exemple `https://localhost:5001/api/todo/1` .
- Sélectionnez **Send**.

L'exemple d'application vous permet de supprimer tous les éléments, mais quand le dernier élément est supprimé, un nouveau est créé par le constructeur de classe de modèle au prochain appel de l'API.

Appeler l'API avec jQuery

Dans cette section, une page HTML qui utilise jQuery pour appeler l'API web est ajoutée. jQuery lance la requête et met à jour la page avec les détails de la réponse de l'API.

Configurez l'application pour [traiter les fichiers statiques](#) et [activer le mappage de fichier par défaut](#) :

C#

 Copier

C#

 Copier

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        // The default HSTS value is 30 days. You may want to
change this for
        // production scenarios, see https://aka.ms/aspnetcore-
hsts.
        app.UseHsts();
    }
}
```

```
app.UseDefaultFiles();
app.UseStaticFiles();
app.UseHttpsRedirection();
app.UseMvc();
}
```


Créez un dossier *wwwroot* dans le répertoire du projet.

Ajoutez un fichier HTML nommé *index.html* au répertoire *wwwroot*. Remplacez son contenu par le balisage suivant :

HTML

 Copier

HTML

 Copier

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>To-do CRUD</title>
  <style>
    input[type='submit'], button, [aria-label] {
      cursor: pointer;
    }

    #spoiler {
      display: none;
    }

    table {
      font-family: Arial, sans-serif;
      border: 1px solid;
      border-collapse: collapse;
    }

    th {
      background-color: #0066CC;
      color: white;
    }

    td {
      border: 1px solid;
      padding: 5px;
    }
  </style>
</head>
<body>
  <h1>To-do CRUD</h1>
```

```

<h3>Add</h3>
<form action="javascript:void(0);" method="POST" onsubmit="add-
Item()">
    <input type="text" id="add-name" placeholder="New to-do">
    <input type="submit" value="Add">
</form>

<div id="spoiler">
    <h3>Edit</h3>
    <form class="my-form">
        <input type="hidden" id="edit-id">
        <input type="checkbox" id="edit-isComplete">
        <input type="text" id="edit-name">
        <input type="submit" value="Save">
        <a onclick="closeInput()" aria-label="Close">#10006;
</a>
    </form>
</div>

<p id="counter"></p>

<table>
    <tr>
        <th>Is Complete</th>
        <th>Name</th>
        <th></th>
        <th></th>
    </tr>
    <tbody id="todos"></tbody>
</table>

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
    integrity="sha256-FgpCb/KJQlLNf0u91ta32o/NMZxltwRo8-
QtmkMRdAu8="
    crossorigin="anonymous"></script>
<script src="site.js"></script>
</body>
</html>

```

Ajoutez un fichier JavaScript nommé *site.js* au répertoire *wwwroot*. Remplacez son contenu par le code suivant :

JavaScript

 Copier

JavaScript

 Copier

```

const uri = "api/todo";
let todos = null;
function getCount(data) {

```

```
const el = $("#counter");
let name = "to-do";
if (data) {
  if (data > 1) {
    name = "to-dos";
  }
  el.text(data + " " + name);
} else {
  el.text("No " + name);
}
}

$(document).ready(function() {
  getData();
});

function getData() {
  $.ajax({
    type: "GET",
    url: uri,
    cache: false,
    success: function(data) {
      const tBody = $("#todos");

      $(tBody).empty();

      getCount(data.length);

      $.each(data, function(key, item) {
        const tr = $("|  |  |
| --- | --- |
|</tr>")
          .append(
            $(" </td>").append(               $("</td>").append(               $("Edit</button>").on("click", function() {                 editItem(item.id);               })             )           )           .append(             $(" </td>").append(               $("Delete</button>").on("click", function() {                 deleteItem(item.id);               })             )           )       });     }   }); } | |

```

```

        })
    )
    );

    tr.appendTo(tBody);
});

    todos = data;
}
});
}

function addItem() {
    const item = {
        name: $("#add-name").val(),
        isComplete: false
    };

    $.ajax({
        type: "POST",
        accepts: "application/json",
        url: uri,
        contentType: "application/json",
        data: JSON.stringify(item),
        error: function(jqXHR, textStatus, errorThrown) {
            alert("Something went wrong!");
        },
        success: function(result) {
            getData();
            $("#add-name").val("");
        }
    });
}

function deleteItem(id) {
    $.ajax({
        url: uri + "/" + id,
        type: "DELETE",
        success: function(result) {
            getData();
        }
    });
}

function editItem(id) {
    $.each(todos, function(key, item) {
        if (item.id === id) {
            $("#edit-name").val(item.name);
            $("#edit-id").val(item.id);
            $("#edit-isComplete")[0].checked = item.isComplete;
        }
    });
}

```



```

});
$("#spoiler").css({ display: "block" });
}

$(".my-form").on("submit", function() {
  const item = {
    name: $("#edit-name").val(),
    isComplete: $("#edit-isComplete").is(":checked"),
    id: $("#edit-id").val()
  };

  $.ajax({
    url: uri + "/" + $("#edit-id").val(),
    type: "PUT",
    accepts: "application/json",
    contentType: "application/json",
    data: JSON.stringify(item),
    success: function(result) {
      getData();
    }
  });

  closeInput();
  return false;
});

function closeInput() {
  $("#spoiler").css({ display: "none" });
}

```

Vous devrez peut-être changer les paramètres de lancement du projet ASP.NET Core pour tester la page HTML localement :

- Ouvrez *Properties\launchSettings.json*.
- Supprimez la propriété `launchUrl` pour forcer l'utilisation du fichier *index.html* (fichier par défaut du projet) à l'ouverture de l'application.

Il existe plusieurs façons d'obtenir jQuery. Dans l'extrait précédent, la bibliothèque est chargée à partir d'un CDN.

Cet exemple appelle toutes les méthodes CRUD de l'API. Les explications suivantes traitent des appels à l'API.

Obtenir une liste de tâches

La fonction JQuery [ajax](#) envoie une requête `GET` à l'API, qui retourne du code JSON

success

 Copier

 Copier

```
$(document).ready(function() {
  getData();
});

function getData() {
  $.ajax({
    type: "GET",
    url: uri,
    cache: false,
    success: function(data) {
      const tbody = $("#todos");

      $(tbody).empty();

      getCount(data.length);

      $.each(data, function(key, item) {
        const tr = $("|  |  |
| --- | --- |
|</tr>")
          .append(
            $(" </td>").append(               $("</td>").append(               $("Edit</button>").on("click", function() {                 editItem(item.id);               })             )           )           .append(             $(" </td>").append(               $("Delete</button>").on("click", function() {                 deleteItem(item.id);               })             )           )       });     }   }); } | |

```

```

    );

    tr.appendTo(tBody);
  });

  todos = data;
}
});
}

```


Ajouter une tâche

La fonction [ajax](#) envoie une requête `POST` dont le corps indique la tâche. Les options `accepts` et `contentType` sont définies avec la valeur `application/json` pour spécifier le type de média qui est reçu et envoyé. La tâche est convertie au format JSON à l'aide de [JSON.stringify](#). Quand l'API retourne un code d'état de réussite, la fonction `getData` est appelée pour mettre à jour la table HTML.

JavaScript

 Copier

JavaScript

 Copier

```

function addItem() {
  const item = {
    name: $("#add-name").val(),
    isComplete: false
  };

  $.ajax({
    type: "POST",
    accepts: "application/json",
    url: uri,
    contentType: "application/json",
    data: JSON.stringify(item),
    error: function(jqXHR, textStatus, errorThrown) {
      alert("Something went wrong!");
    },
    success: function(result) {
      getData();
      $("#add-name").val("");
    }
  });
}

```

Mettre à jour une tâche

La mise à jour d'une tâche est similaire à l'ajout d'une tâche. L'identificateur unique de la tâche est ajouté à l' `url` et le `type` est `PUT` .

JavaScript

 Copier

JavaScript

 Copier

```
$.ajax({
  url: uri + "/" + $("#edit-id").val(),
  type: "PUT",
  accepts: "application/json",
  contentType: "application/json",
  data: JSON.stringify(item),
  success: function(result) {
    getData();
  }
});
```

Supprimer une tâche

Pour supprimer une tâche, vous devez définir le `type` sur l'appel AJAX avec la valeur `DELETE` et spécifier l'identificateur unique de l'élément dans l'URL.

JavaScript

 Copier

JavaScript

 Copier

```
$.ajax({
  url: uri + "/" + id,
  type: "DELETE",
  success: function(result) {
    getData();
  }
});
```

Ressources supplémentaires

[Affichez ou téléchargez l'exemple de code de ce tutoriel.](#) Consultez [Guide pratique pour télécharger](#).

Pour plus d'informations, reportez-vous aux ressources suivantes :

- [Créer des API web avec ASP.NET Core](#)