

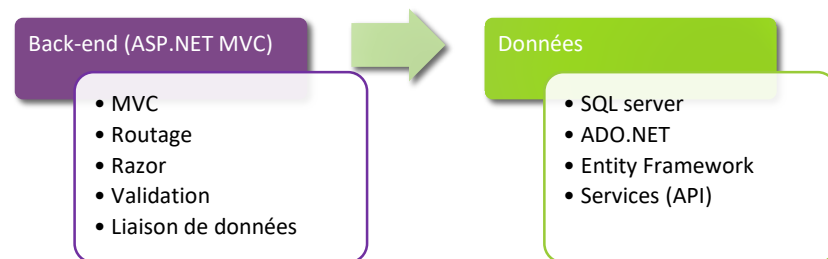
Les services Web avec ASP.NET Web API

MOHAMED AIROUCHE

Les parties du cours

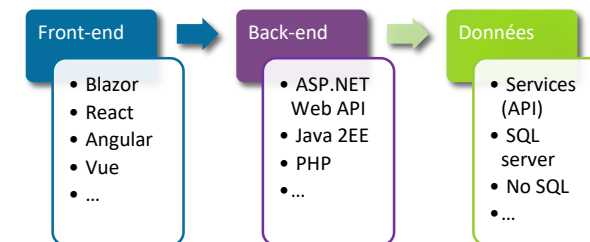
Dans ce cours, nous nous intéresserons à:

- ✓ La logique de développement MVC avec ASP.NET (architecture MVC, routage, validation et liaison de données).
- ✓ La logique d'accès aux bases de données (ADO.NET, LINQ, DbContext, l'ORM Entity Framework).
- Les logiques de développement avec ASP.NET Core (modularité, injection de dépendance, MVC, Pages Razor, Web API).
- Utilisation d'Identity Framework pour la sécurité de l'application (authentification et autorisation).
- La logique de développement de service Web (API) du côté serveur et Blazor du côté client en utilisant juste la technologie .NET.



<https://dotnet.microsoft.com/learn/aspnet>

<https://dotnet.microsoft.com/learn/aspnet/architecture>



ASP.NET Web API

- ASP.NET Web API est un Framework permettant de créer des services HTTP accessibles à partir de n'importe quel client, y compris les navigateurs et les appareils mobiles. C'est une plate-forme idéale pour créer des applications RESTful avec .NET Framework.
- ASP.NET Web API utilise des modèles et des contrôleurs avec des actions pour réaliser les opérations CRUD dans une base de données.

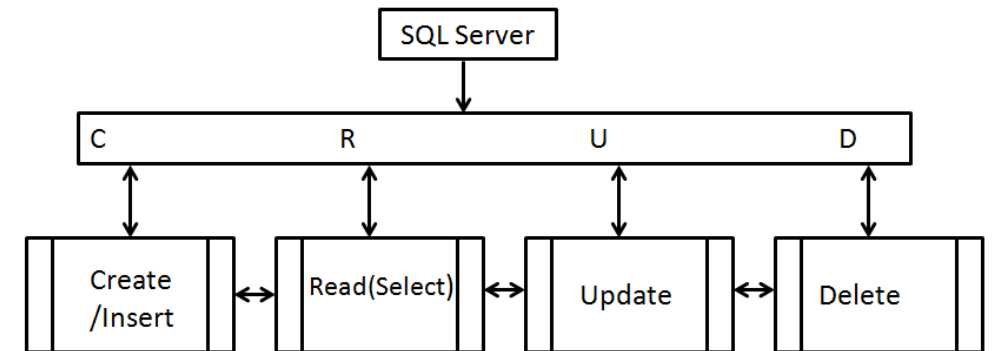
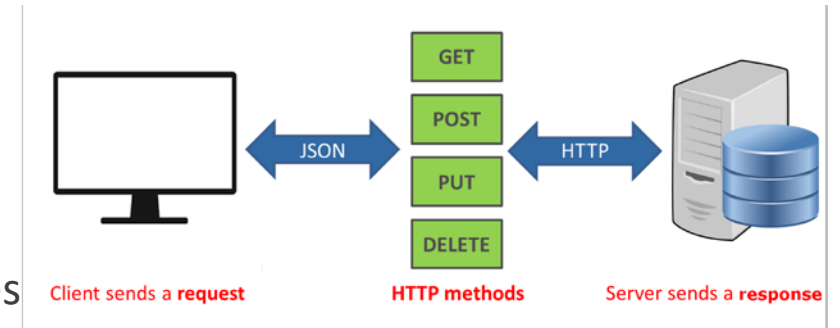


<https://www.tutorialsteacher.com/webapi/web-api-tutorials>

Le Web API

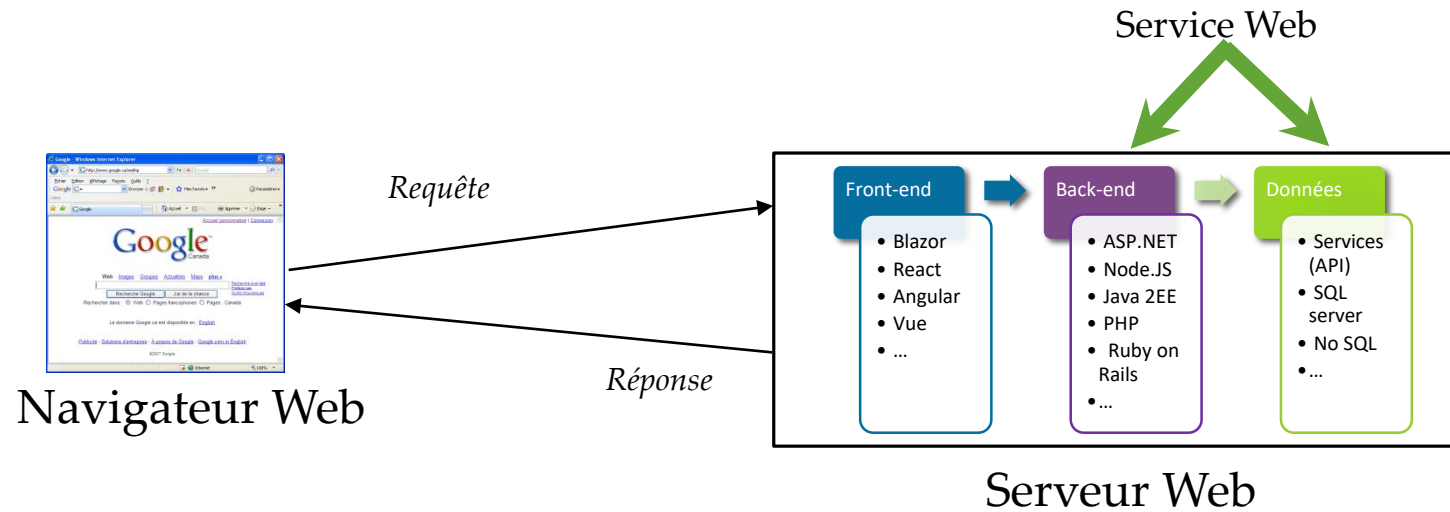
- Visual Studio 2017 permet d'ajouter un contrôleur pour un Web API avec des fonctionnalités de génération automatique des méthodes d'actions suivantes :

- Post : la méthode d'action pour insérer des données (insert).
- Get : la méthode d'action pour lire un enregistrement (Select).
- Put : la méthode d'action pour la mise à jour des données (update).
- Delete : la méthode d'action pour supprimer un enregistrement.



<https://www.tutorialsteacher.com/webapi/web-api-tutorials>

Architecture d'une application Web



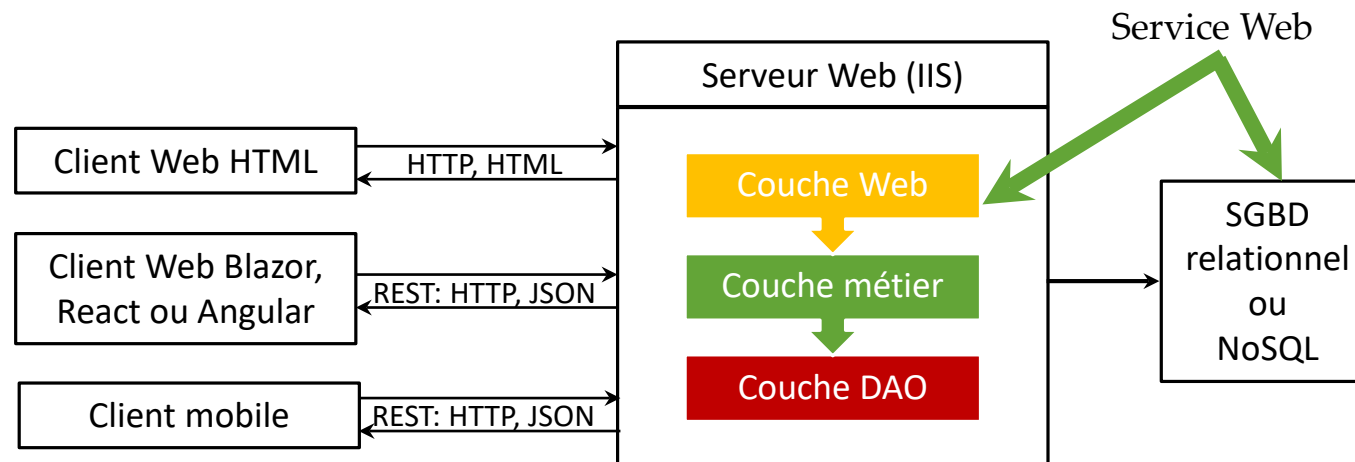
Une application Web moderne peut être composée principalement par les trois parties suivantes :

- **Partie Front-end** : contient la logique permettant de créer l'interface utilisateur et le script exécuté du côté client.
- **Partie Back-end** : contient la logique et le code exécuté du côté serveur comme la logique des couches métier et accès aux données.
- **Partie des données** : contient généralement les serveurs des données sous forme SQL ou non SQL.

Architecture d'une application Web

Une application Web utilisant une architecture en couches est souvent composée de trois couches :

- La couche Web
- La couche métier
- La couche d'accès aux bases de données



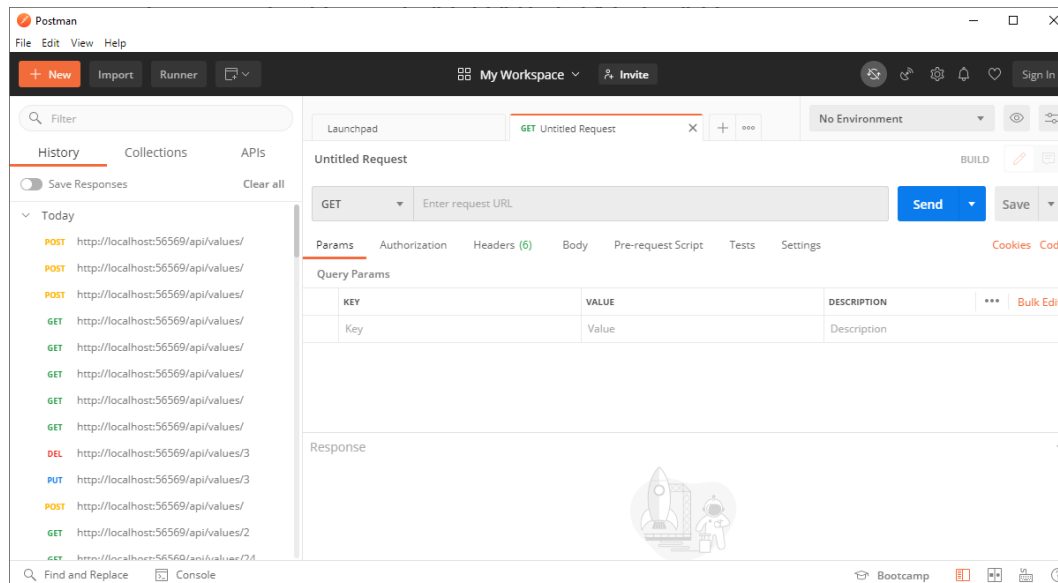
<https://docs.microsoft.com/fr-ca/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>

Outil Postman

Postman

Parmi les nombreuses solutions pour interroger ou tester les services Web (API), Postman propose de nombreuses fonctionnalités, une prise en main rapide et une interface graphique agréable.

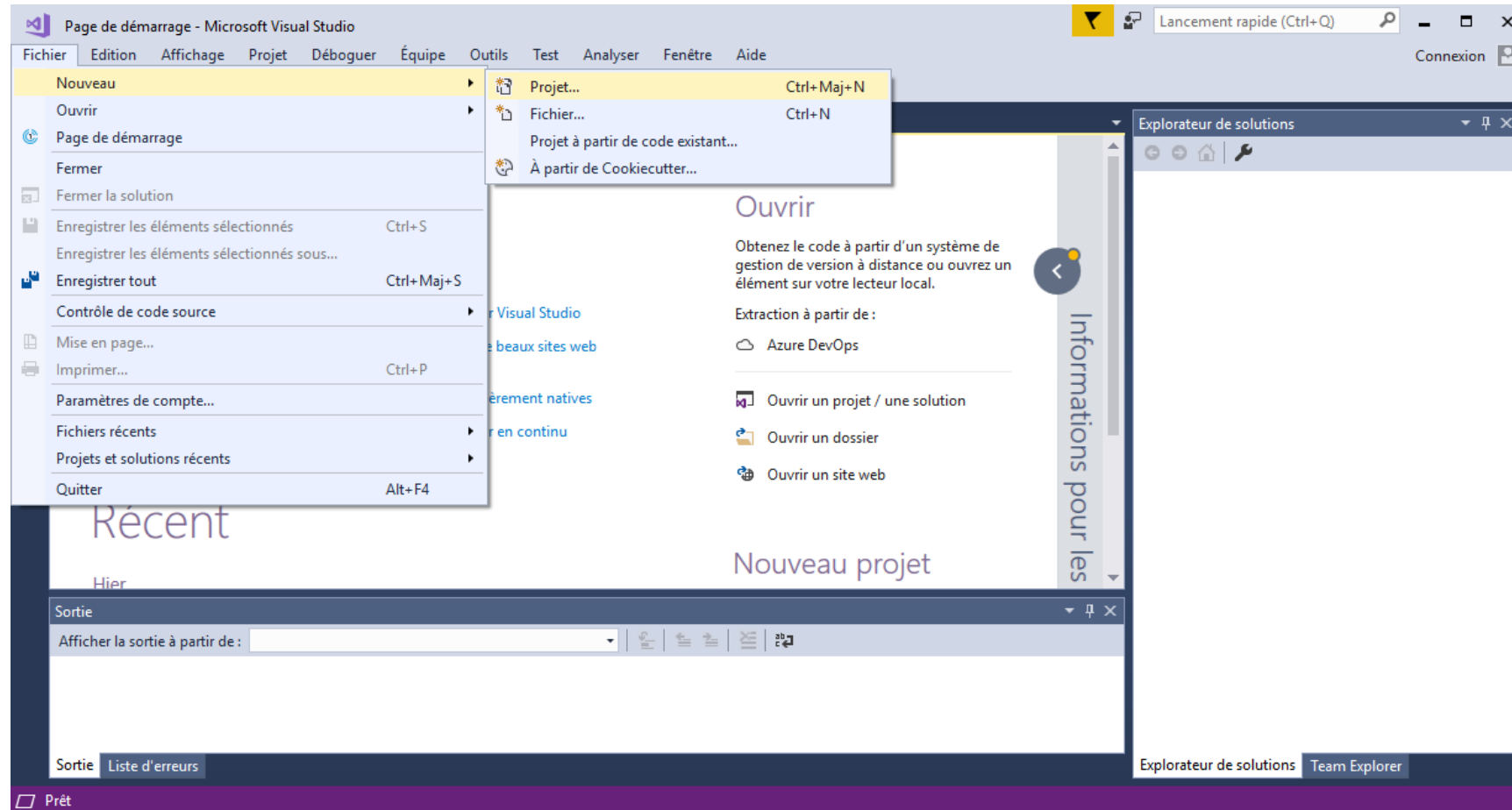
Postman permet de construire et d'exécuter des requêtes HTTP, de les stocker dans un historique afin de pouvoir les rejouer, mais surtout de les organiser en Collections. Cette classification permet notamment de regrouper des requêtes de façon «fonctionnelle». <https://blog.webnet.fr/presentation-de-postman-outil-multifonction-pour-api-web/>
Postman est un outil utilisé pour tester les services Web.



<https://www.postman.com/>

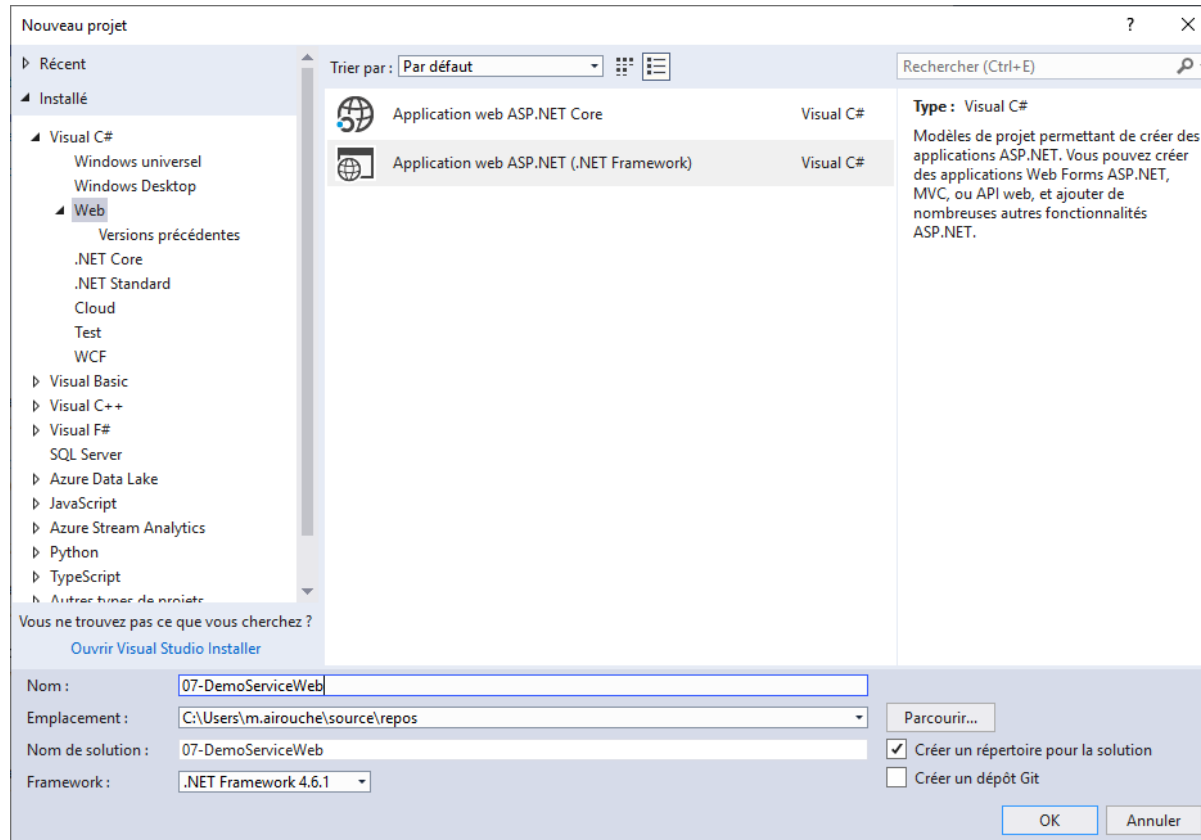
Exemple avec ASP.NET Web API

La création d'un projet ASP.NET Web API s'effectue via le menu **Fichier-Nouveau Projet**



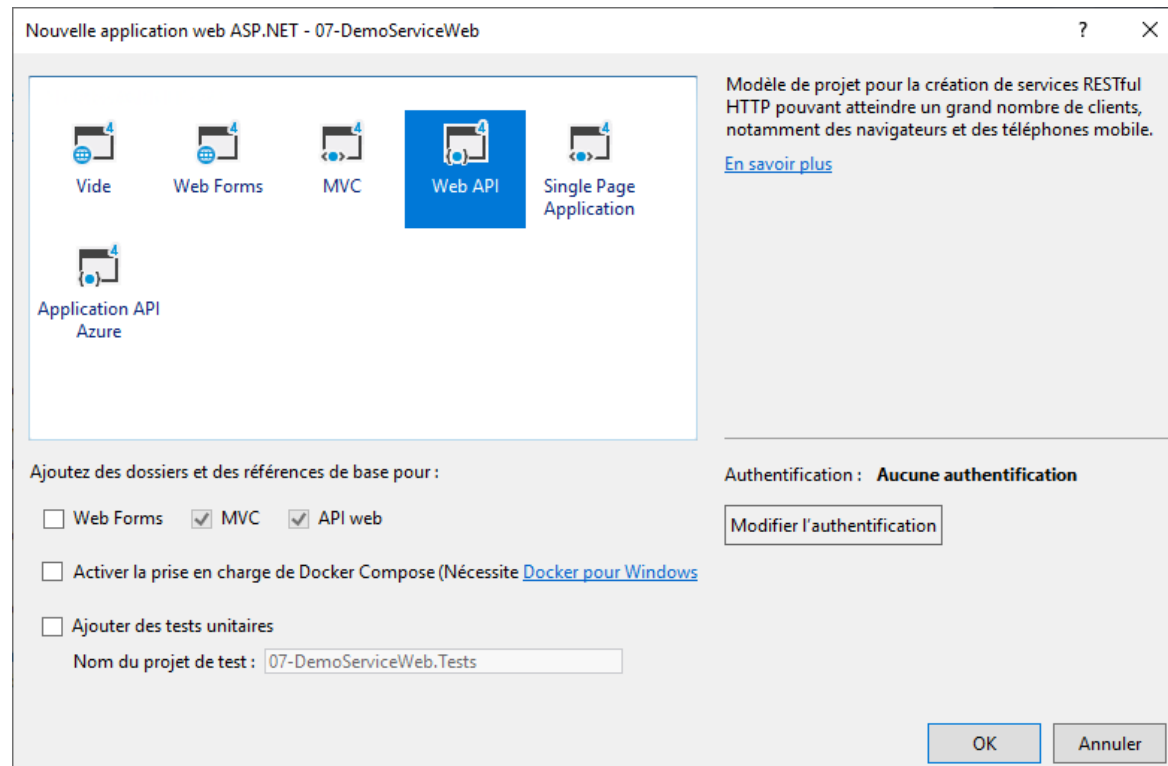
Exemple avec ASP.NET Web API

Sélectionnez (.NET Framework) et nommez le projet : **07-DemoServiceWeb**



Exemple avec ASP.NET Web API

Sélectionnez le modèle Web API et les deux options: MVC et API web

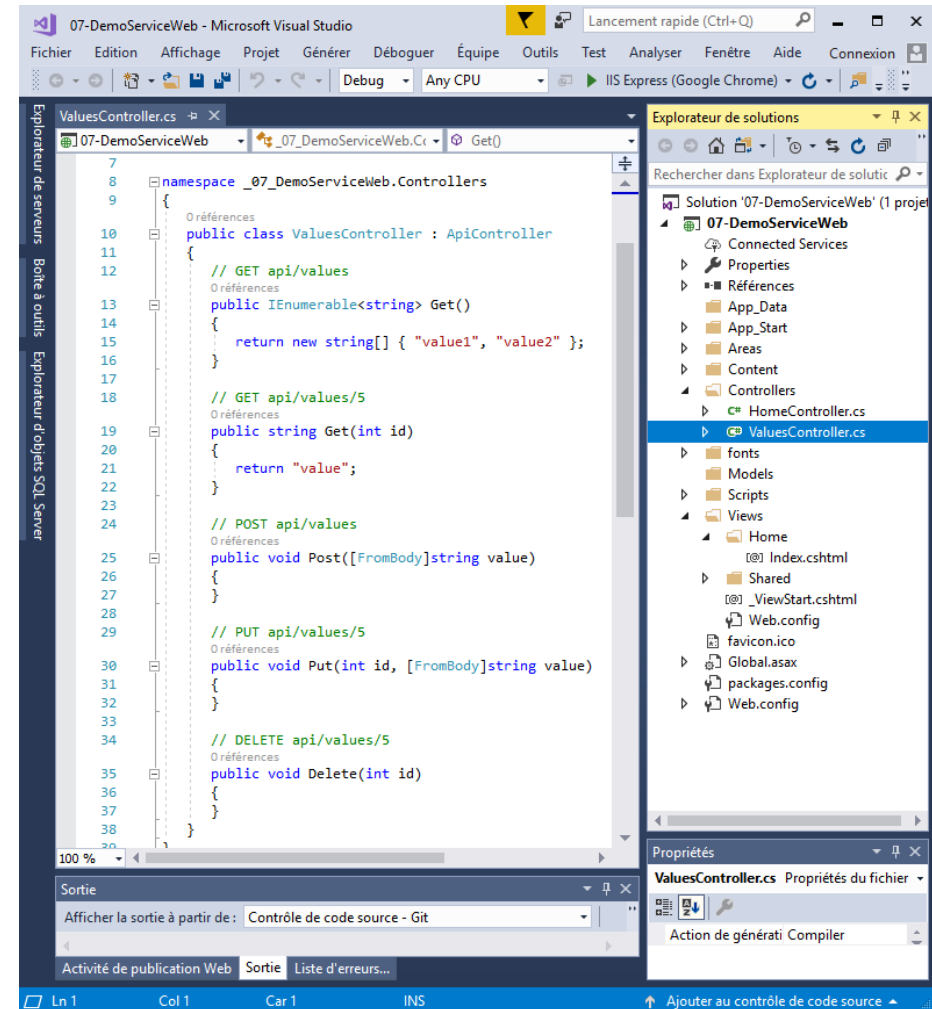


Exemple avec ASP.NET Web API

Visual studio génère une solution du projet similaire aux projets MVC générés précédemment dans ce cours.

On peut distinguer les composantes suivantes :

- Un contrôleur Home qui hérite de Controller.
- Un contrôleur Values qui hérite de ApiController.



Exemple avec ASP.NET Web API

Un contrôleur Values qui hérite de ApiController.

Si vous souhaitez écrire des méthodes qui ne commencent pas par un verbe HTTP, vous pouvez appliquer l'attribut de verbe http approprié sur la méthode telle que HttpGet, HttpPost, HttpPut etc. comme le contrôleur MVC.

<https://www.tutorialsteacher.com/webapi/web-api-controller>

Type de retour de méthode d'action

La méthode d'action de l'API Web peut avoir les types de retour suivants.

- ✓ Void
- ✓ Type primitif ou type complexe
- ✓ HttpResponseMessage
- ✓ IHttpActionResult

<https://www.tutorialsteacher.com/webapi/action-method-return-type-in-web-api>

```
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // GET api/values/5
    public string Get(int id)
    {
        return "value";
    }

    // POST api/values
    public void Post([FromBody]string value)
    {
    }

    // PUT api/values/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/values/5
    public void Delete(int id)
    {
    }
}
```

Exemple avec ASP.NET Web API

Liaison des données

Les méthodes d'action dans les contrôleurs Web API peuvent avoir un ou plusieurs paramètres de types différents. Il peut être de type primitif ou complexe. Le Web API lie les paramètres de la méthode d'action avec la chaîne de requête de l'URL ou avec le corps de la requête en fonction du type de paramètre. Par défaut, si le type de paramètre est de type primitif .NET tel que int, bool, double, string, GUID, DateTime, decimal ou tout autre type pouvant être converti à partir du type chaîne, il définit la valeur d'un paramètre à partir de la chaîne de requête. Et si le type de paramètre est de type complexe, le Web API essaie par défaut d'obtenir la valeur du corps de la requête.

Le tableau suivant répertorie les règles par défaut pour la liaison de paramètres.

HTTP Method	Query String	Request Body
GET	Primitive Type, Complex Type	NA
POST	Primitive Type	Complex Type
PUT	Primitive Type	Complex Type
PATCH	Primitive Type	Complex Type
DELETE	Primitive Type, Complex Type	NA

<https://www.tutorialsteacher.com/webapi/parameter-binding-in-web-api>

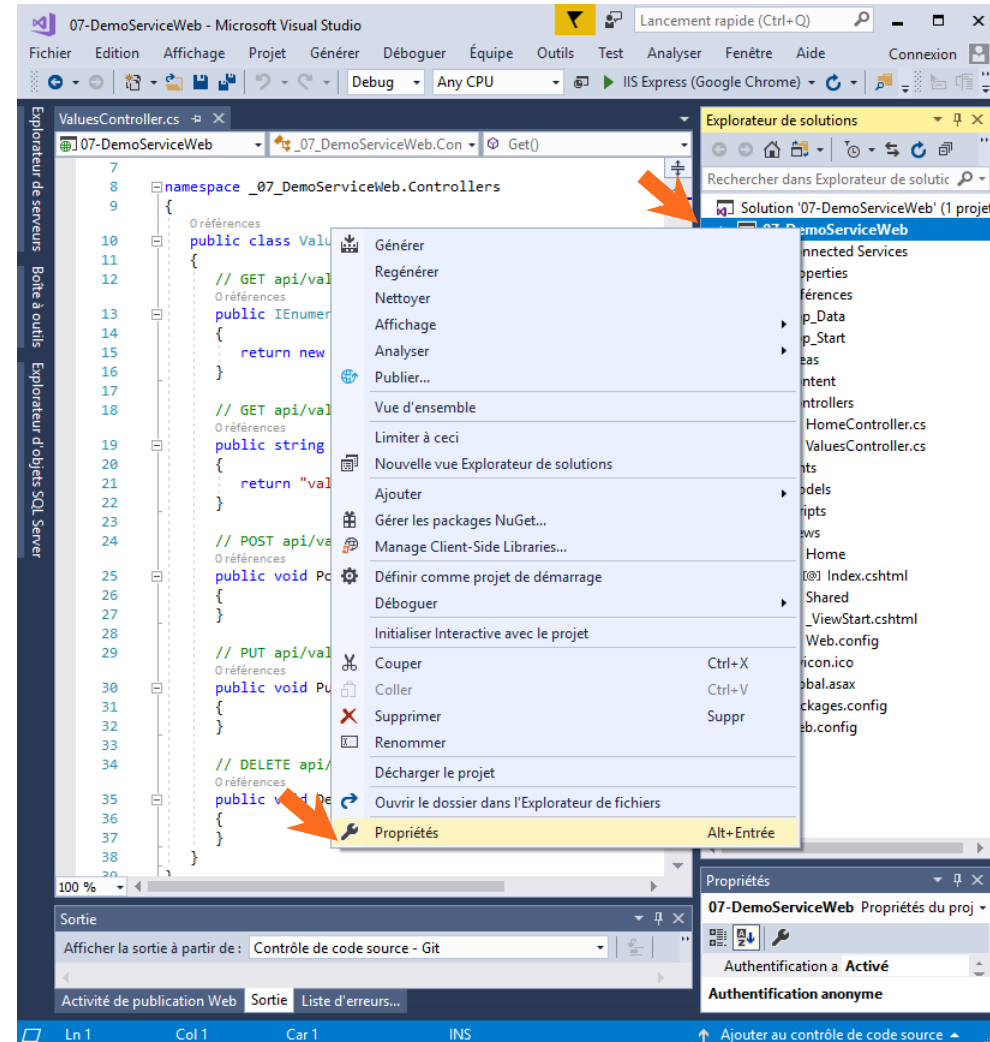
Exemple avec ASP.NET Web API

Changer l'adresse URL du projet

Par défaut, Visual studio affecte une adresse URL au projet à utiliser avec IIS Express sous la forme suivante :

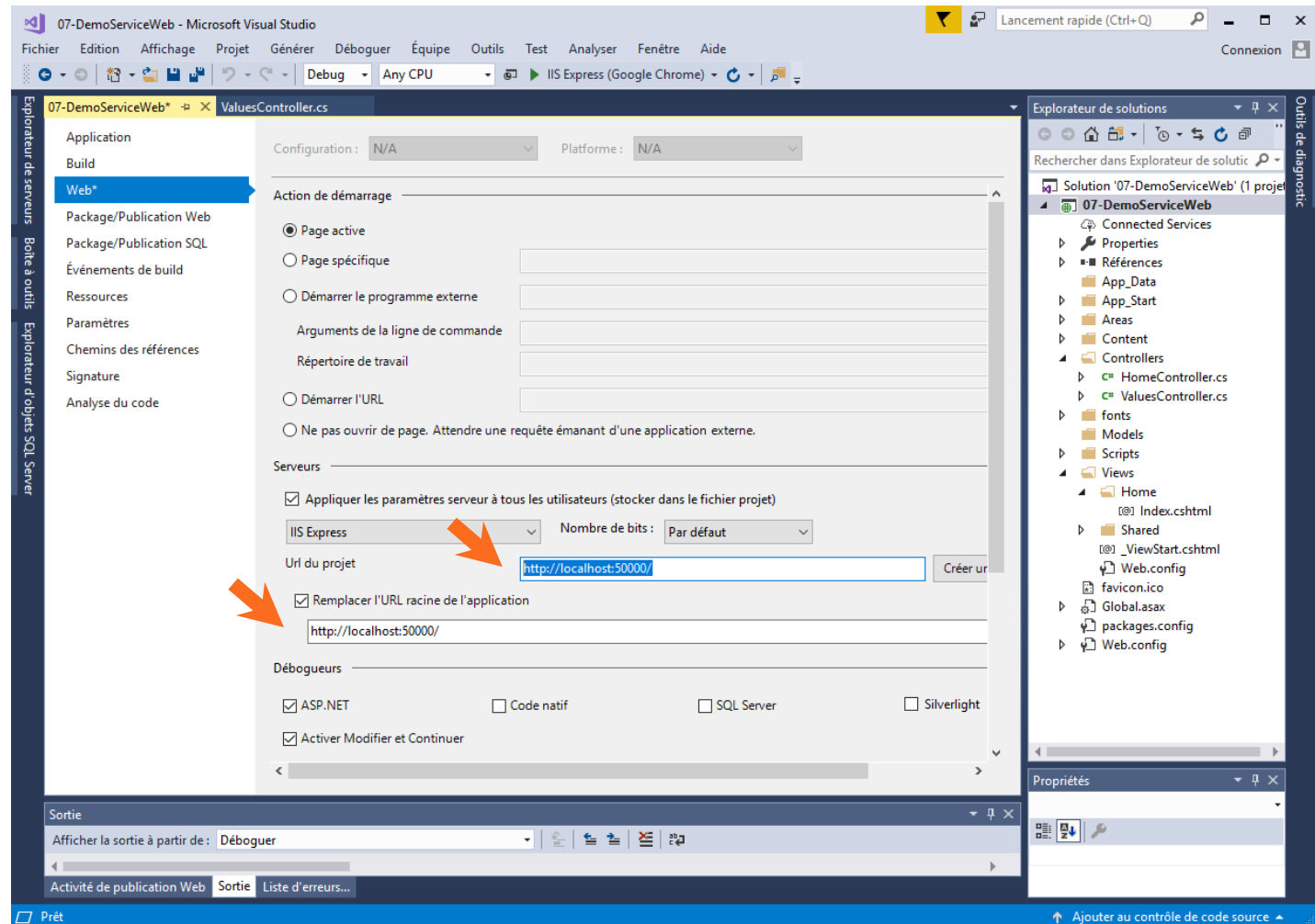
<http://localhost:NoPort/>

Avec Visual studio, on peut changer l'URL par défaut d'un projet en changeant le numéro de port comme le montrent les figures suivantes.



Exemple avec ASP.NET Web API

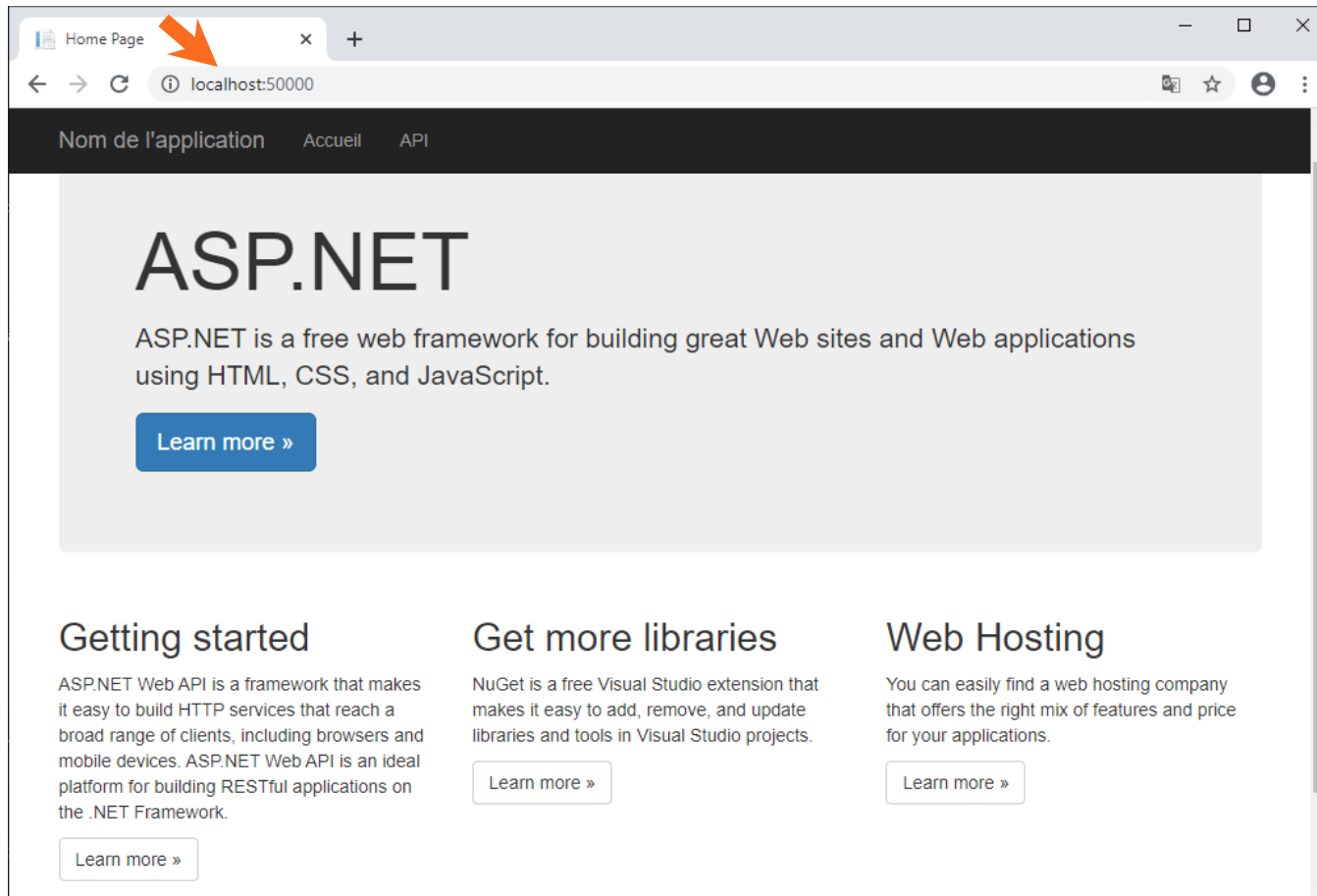
Changer l'adresse URL du projet



Exemple avec ASP.NET Web API

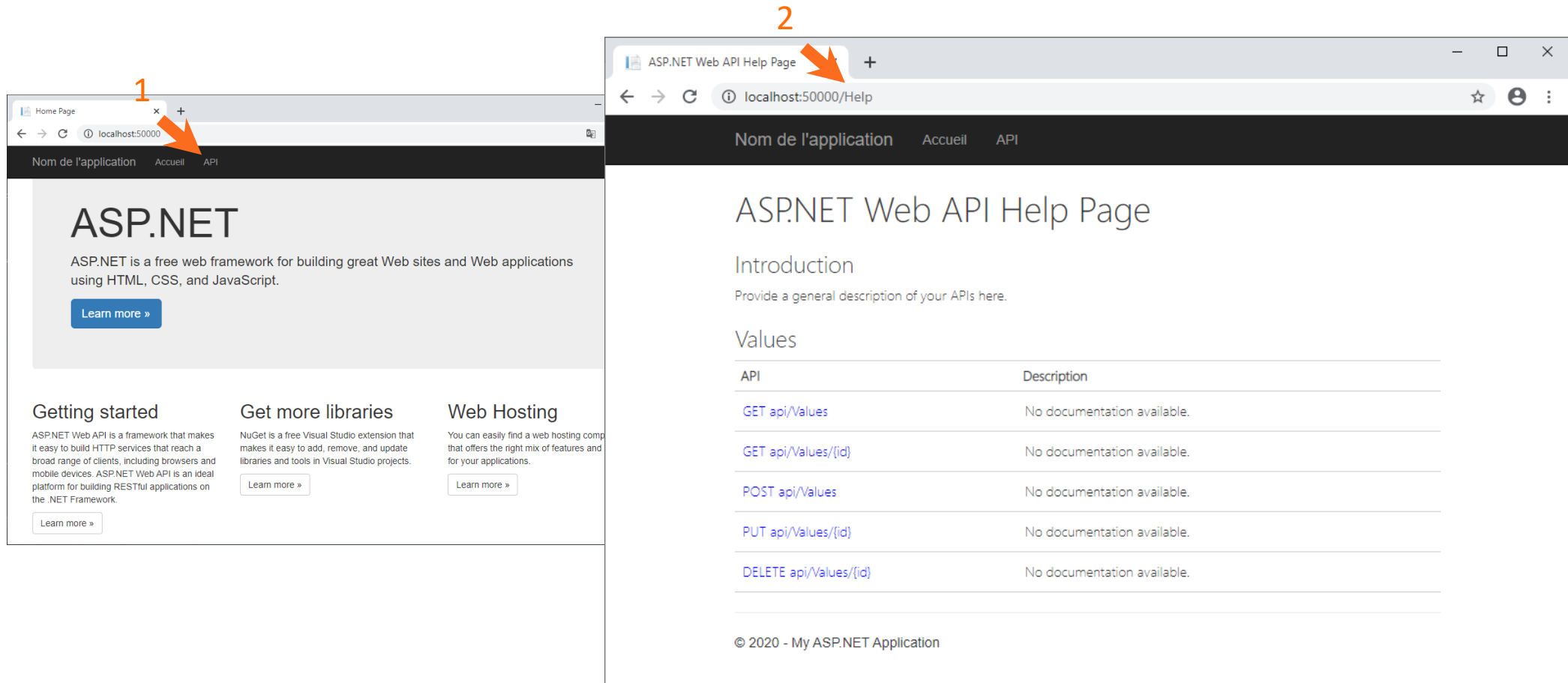
Exécution

L'exécution de l'application donne la page de la figure suivante.



Exemple avec ASP.NET Web API

Exécution



Exemple avec ASP.NET Web API

Le routage pour Web API

Le routage est le processus consistant à diriger une demande HTTP vers une action d'un contrôleur.

Le fichier **RouteConfig.cs** dans le dossier App_start est la partie de votre application, dans laquelle vous définissez l'itinéraire pour votre application pour la partie **MVC**.

Le fichier **WebApiConfig.cs** dans le dossier App_start est la partie de votre application, dans laquelle vous définissez l'itinéraire pour votre application pour la partie **Web API**.

Le [Front Controller] devait router une URL vers l'action chargée de la traiter. Une route sert à faire le lien entre un modèle d'URL et une action. Ces routes sont définies dans le dossier [App_Start] du projet par les classes [FilterConfig, **WebApiConfig**, **RouteConfig**, BundleConfig] :

<https://www.tutorialsteacher.com/mvc/routing-in-mvc>

Exemple avec ASP.NET Web API

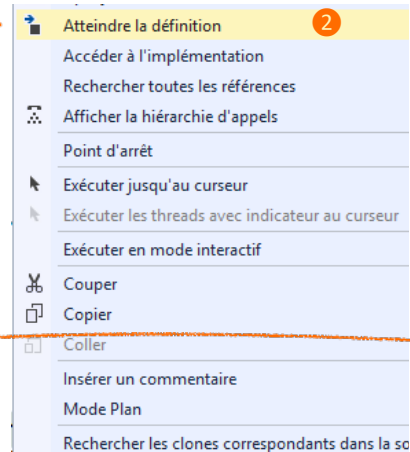
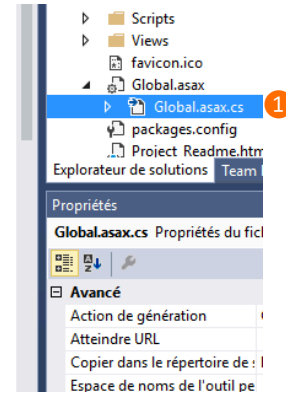
Comment les routes par défaut sont-elles configurées pour les Web API?

0 références

```
public class WebApiApplication : System.Web.HttpApplication
{
    0 références
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        GlobalConfiguration.Configure(WebApiConfig.Register);
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

```
public class ValuesController : ApiController
```

localhost:50000/api/values



App_Start

- BundleConfig.cs
- FilterConfig.cs
- RouteConfig.cs
- WebApiConfig.cs

```
public static class WebApiConfig
{
    1 référence
    public static void Register(HttpConfiguration config)
    {
        // Configuration et services API Web

        // Itinéraires de l'API Web
        config.MapHttpAttributeRoutes();

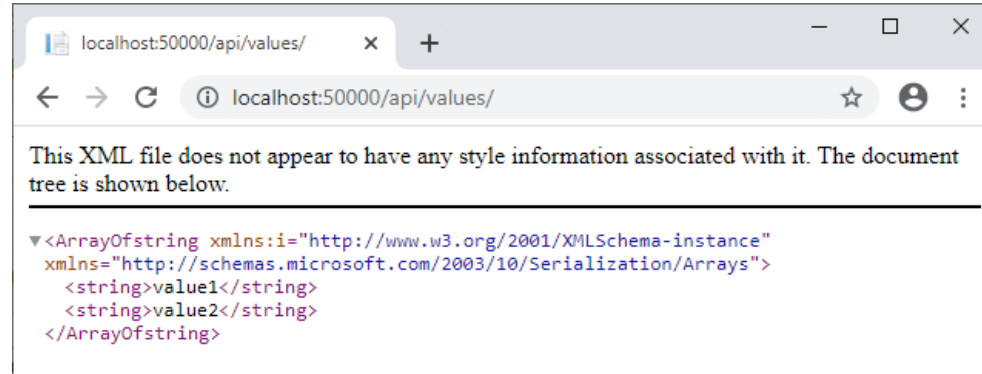
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Exemple avec ASP.NET Web API

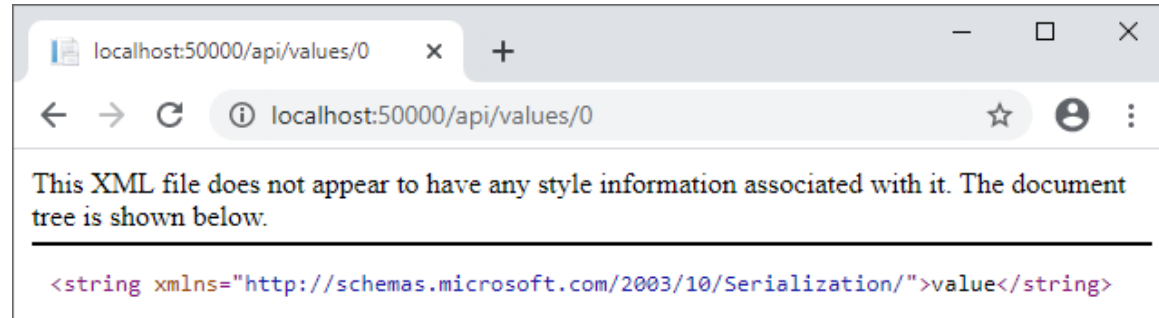
Les méthodes GET de l'API

On peut interroger la collection, ou une ressource spécifique.

API
GET api/Values
GET api/Values/{id}
POST api/Values
PUT api/Values/{id}
DELETE api/Values/{id}




```
<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
  <string>value1</string>
  <string>value2</string>
</ArrayOfstring>
```



```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">value</string>
```

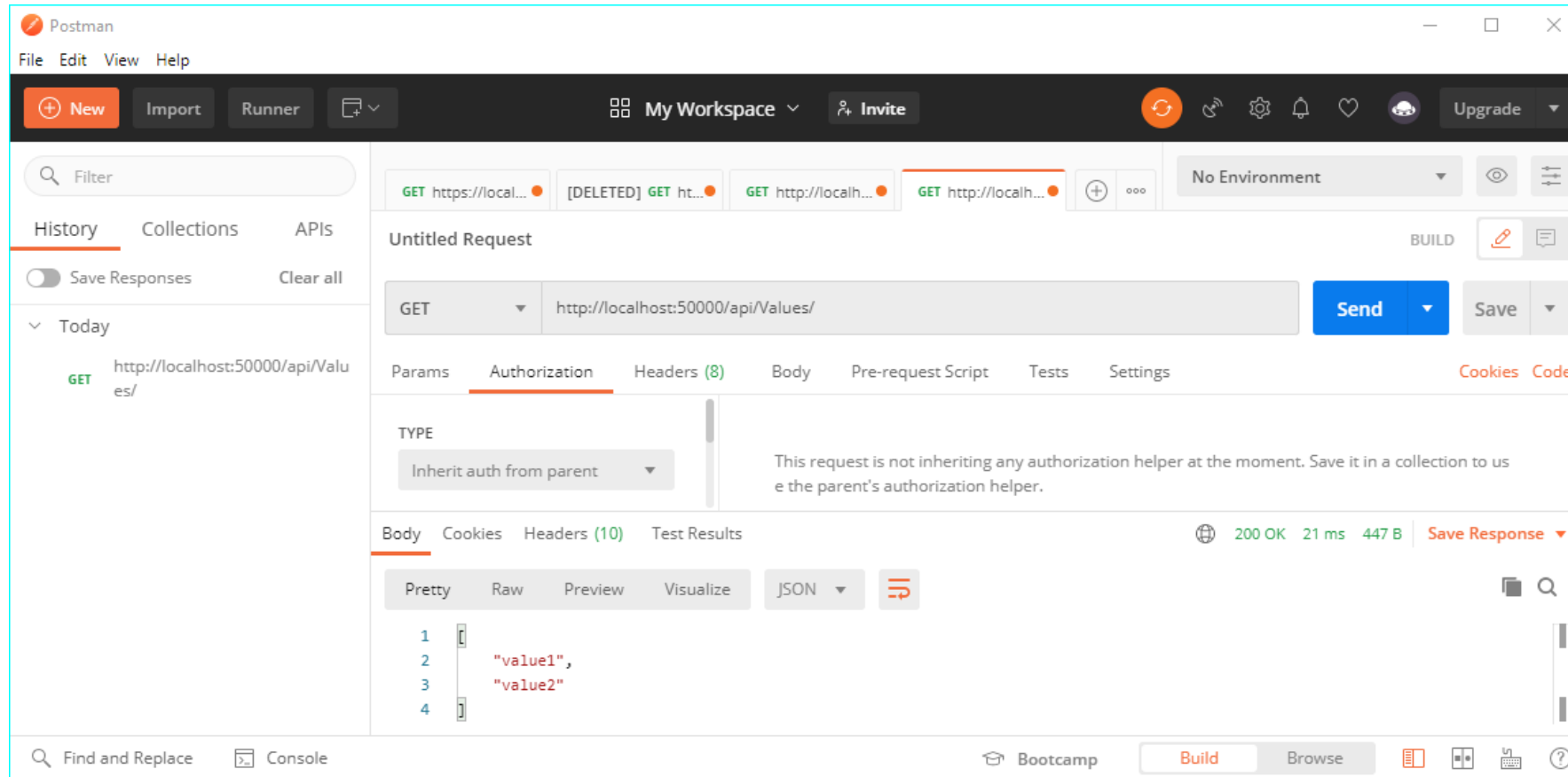
Exemple avec ASP.NET Web API

Outil – Postman pour tester l'API

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Settings	Cookies	Code
Headers  Hide auto-generated headers								
	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets		
<input checked="" type="checkbox"/>	Cache-Control ⓘ	no-cache						
<input checked="" type="checkbox"/>	Postman-Token ⓘ	<calculated when request is sent>						
<input checked="" type="checkbox"/>	Host ⓘ	<calculated when request is sent>						
<input checked="" type="checkbox"/>	User-Agent ⓘ	PostmanRuntime/7.26.3						
<input checked="" type="checkbox"/>	Accept ⓘ	*/*						
<input checked="" type="checkbox"/>	Accept-Encoding ⓘ	gzip, deflate, br						
<input checked="" type="checkbox"/>	Connection ⓘ	keep-alive						

Outils — Postman (json)

Les méthodes GET de l'API



Outils — Postman (xml)

Les méthodes GET de l'API

The image displays two screenshots of the Postman application interface, illustrating the setup for sending a GET request with an XML response.

Left Screenshot: Request Setup

- Method:** GET
- URL:** `http://localhost:50000/api/Values/`
- Headers (8):**

KEY	VALUE
Cache-Control	no-cache
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.26.3
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Accept	application/xml

Right Screenshot: Response Body

- Body:** `ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays"><string>value1</string><string>value2</string></ArrayOfstring>`
- View:** XML

Orange arrows highlight the **Send** button, the **Accept** header, and the **XML** view.

Exemple avec ASP.NET Web API

Modifions le contrôleur Values et ajoutons du code pour toutes les méthodes

```
public class ValuesController : ApiController
{
    static List<string> strings = new List<string>()
        { "value0", "value1", "value2" };

    // GET api/values
    public IEnumerable<string> Get()
    {
        return strings;
    }

    // GET api/values/5
    public string Get(int id)
    {
        return strings[id];
    }

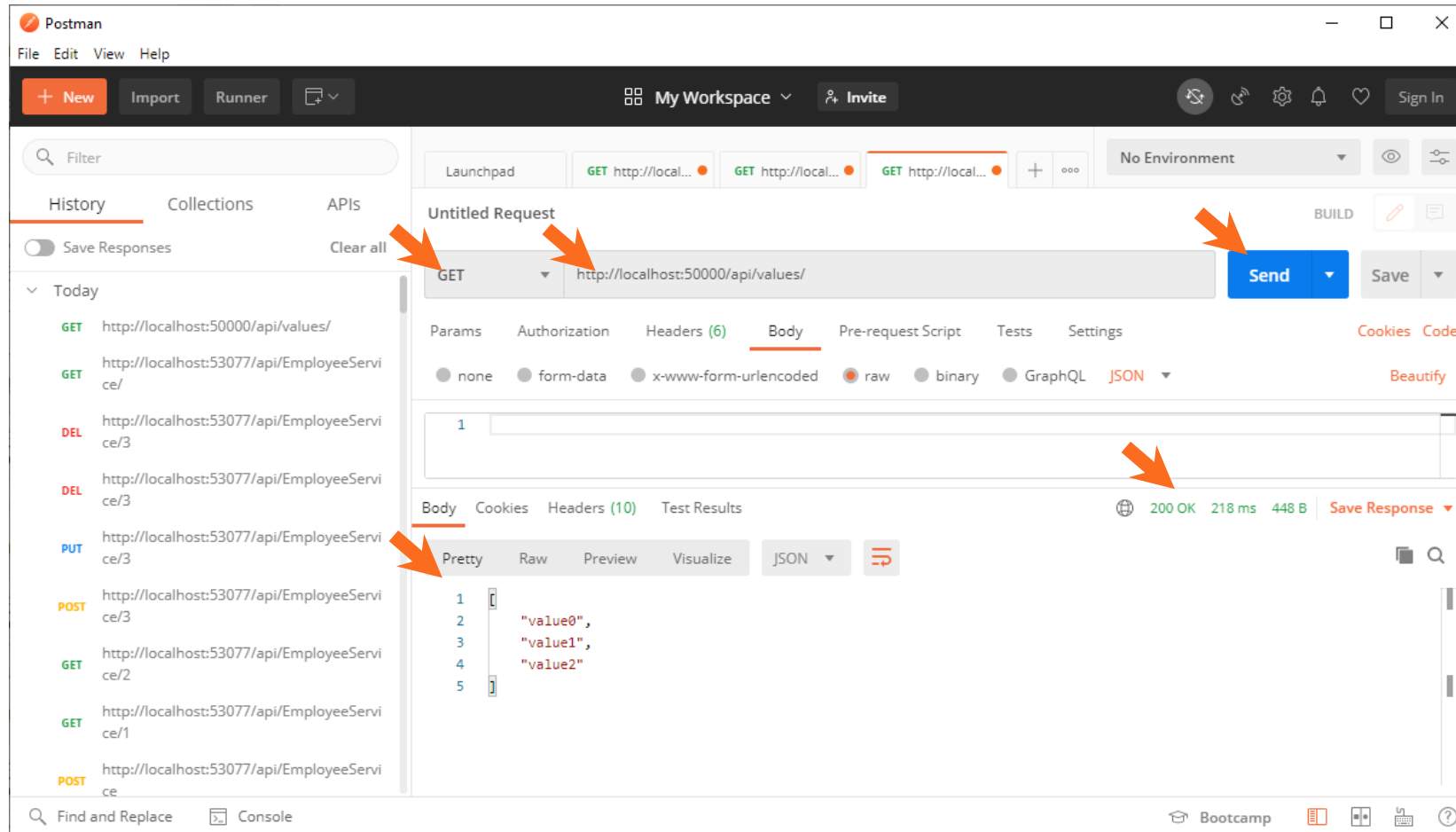
    // POST api/values
    public void Post([FromBody]string value)
    {
        strings.Add(value);
    }
}
```

```
// La suite de contrôleur
// PUT api/values/5
public void Put(int id, [FromBody]string value)
{
    strings[id] = value;
}

// DELETE api/values/5
public void Delete(int id)
{
    strings.RemoveAt(id);
}
}
```

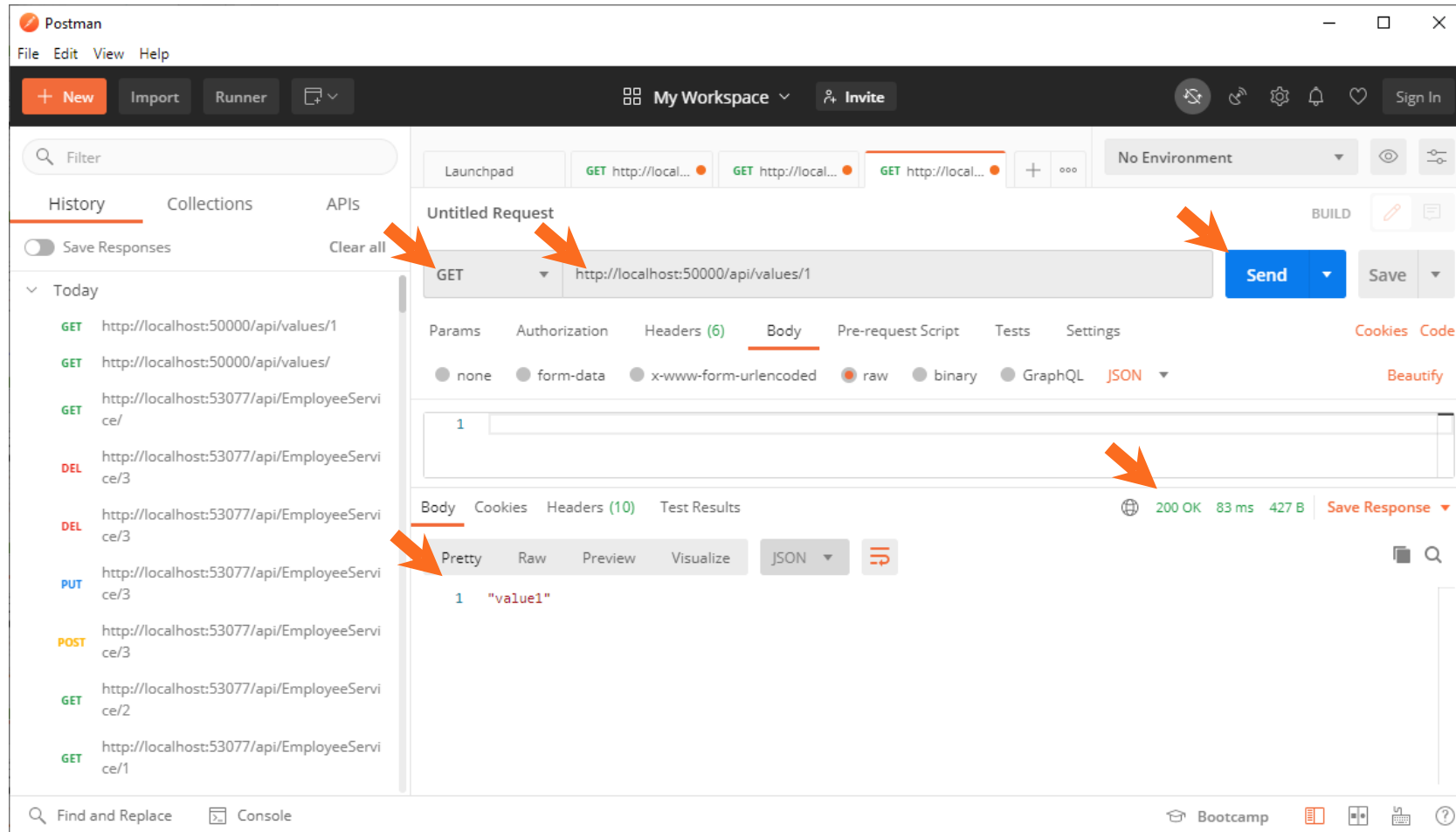

Outils — Postman

Outil – Postman pour tester la méthode GET de l'API



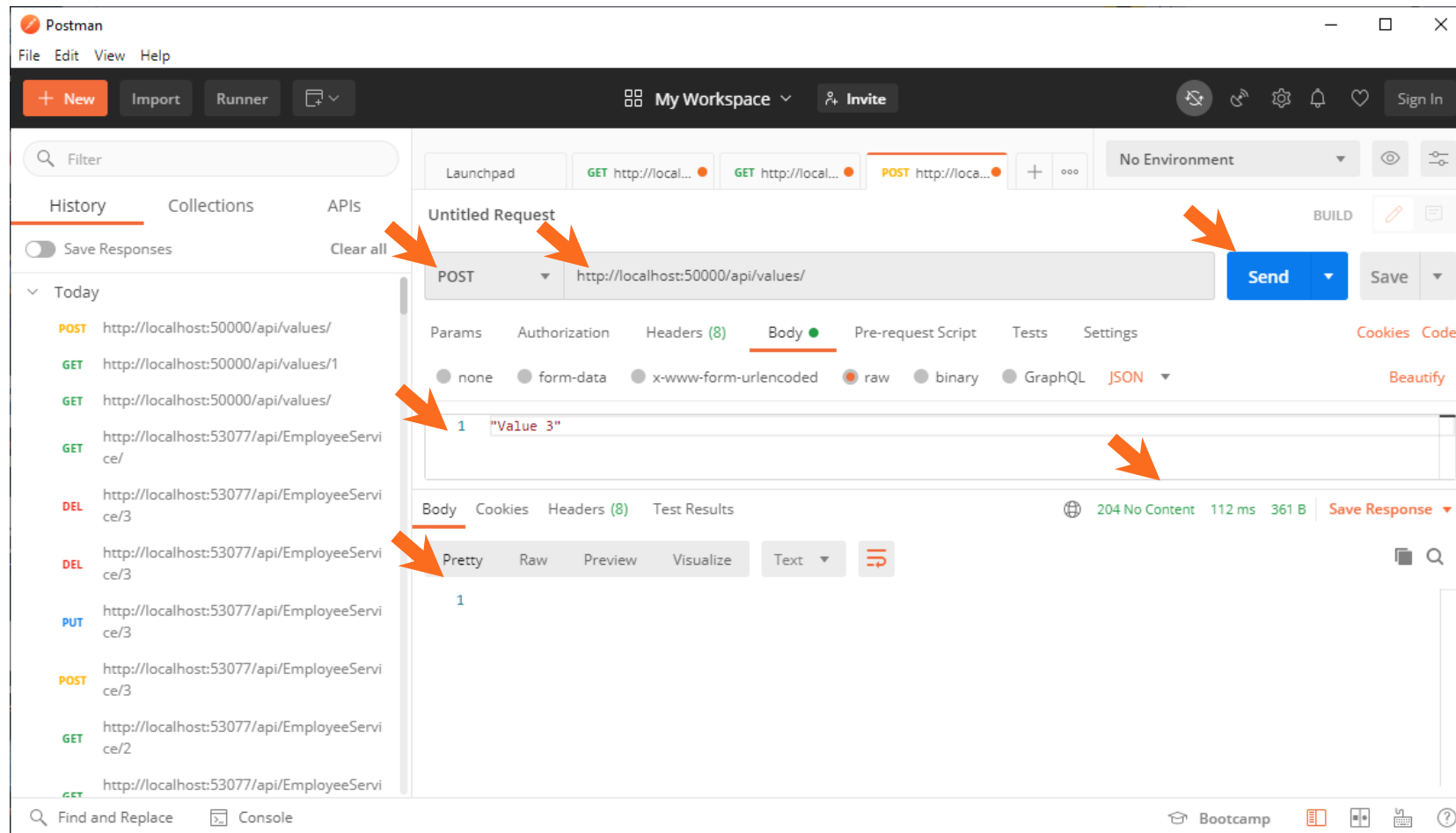
Outils — Postman

Outil – Postman pour tester la méthode GET avec id de l'API



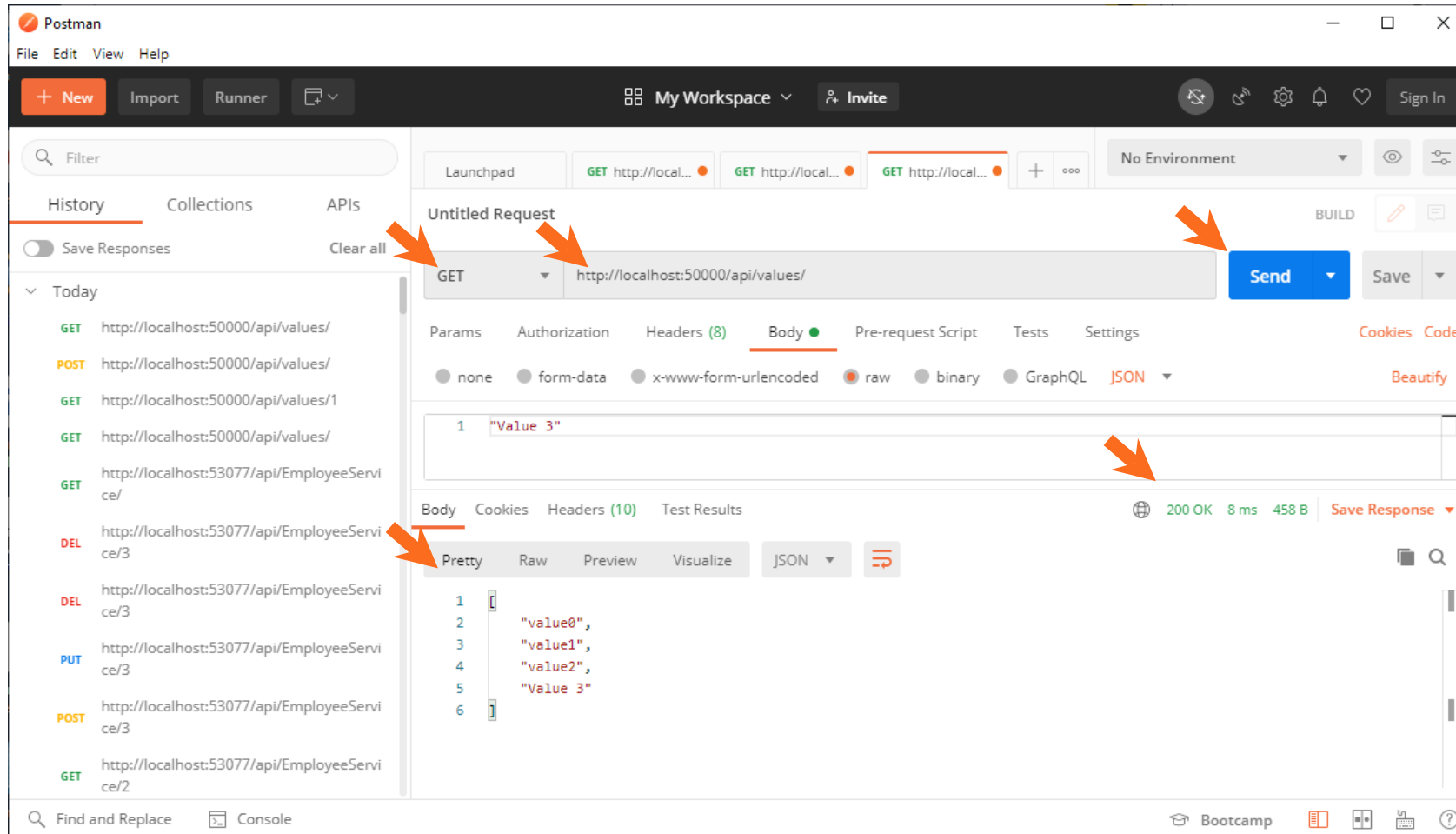
Outils — Postman

Outil – Postman pour tester la méthode POST de l'API



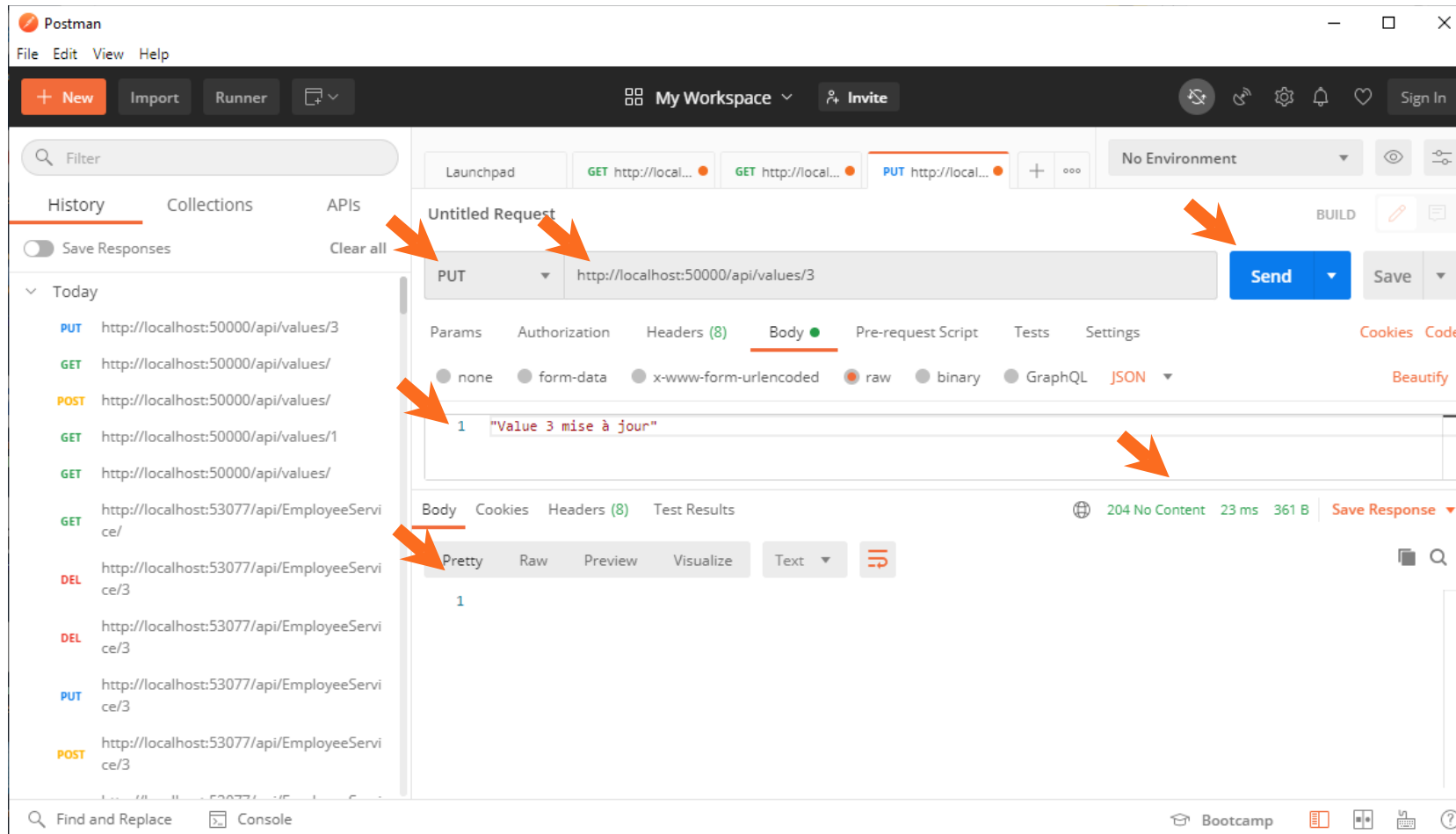
Outils — Postman

Outil – Postman pour confirmer l'ajout de l'élément avec la méthode GET de l'API



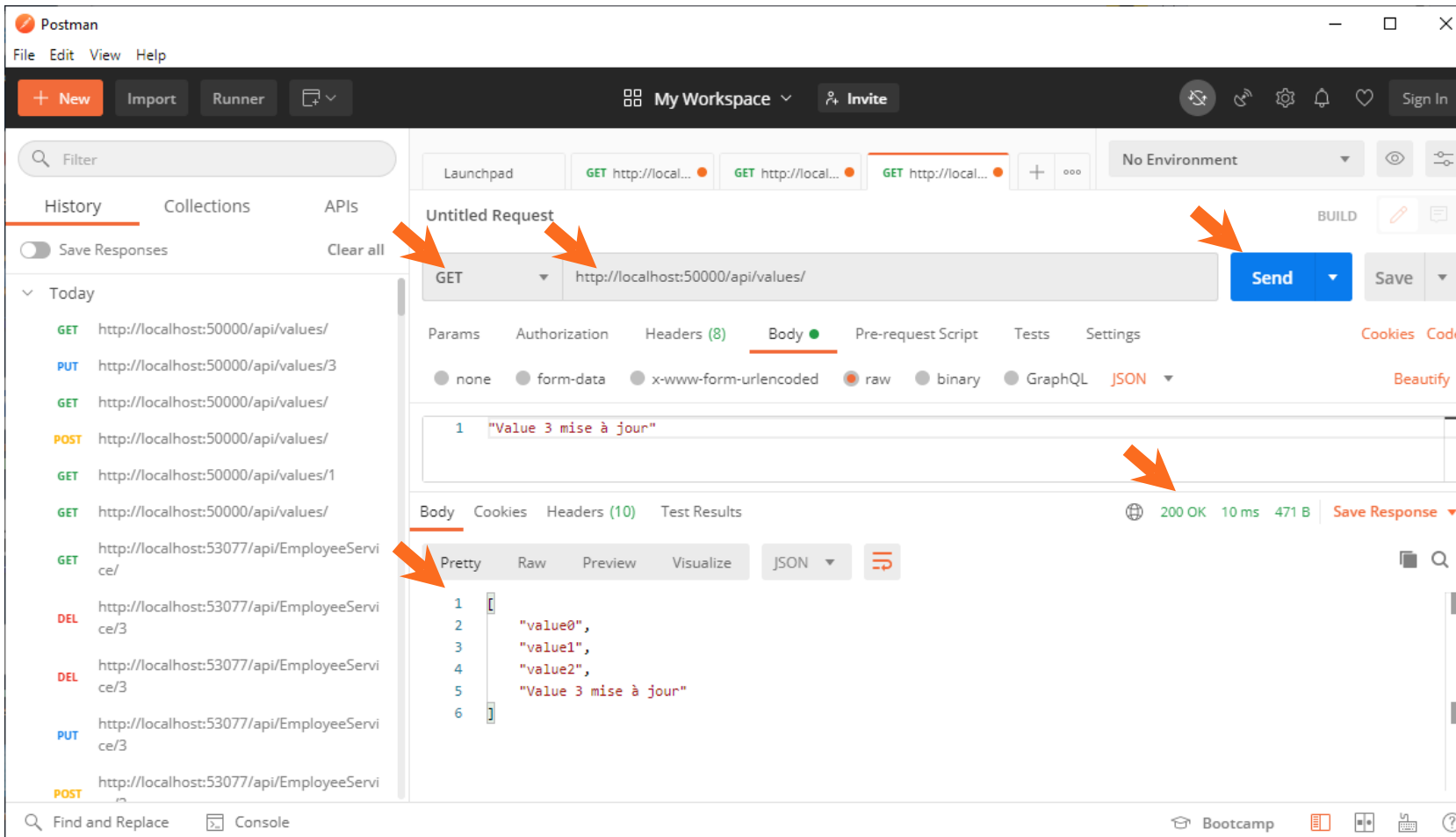
Outils — Postman

Outil – Postman pour tester la méthode PUT de l'API



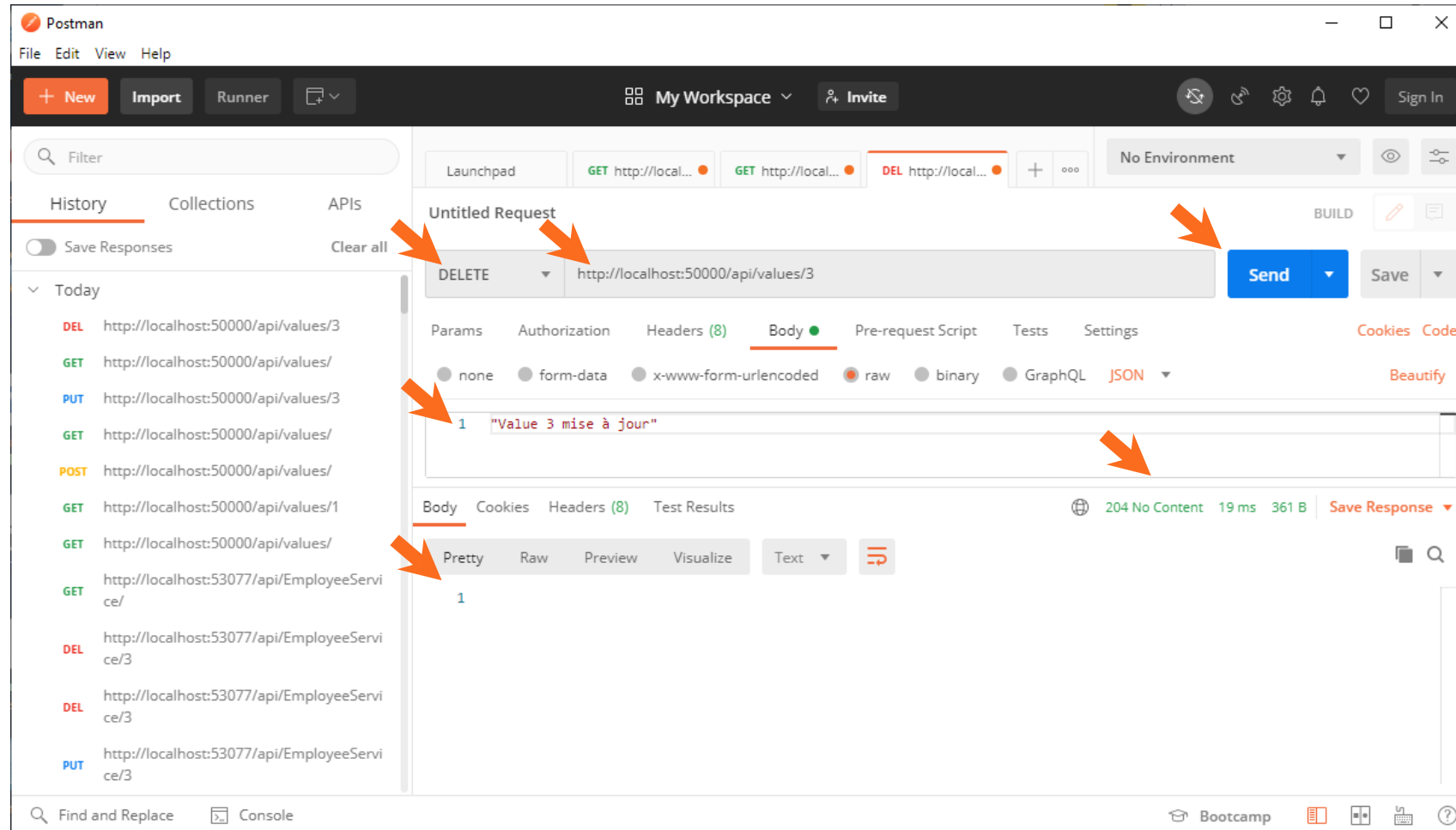
Outils — Postman

Outil – Postman pour valider la mise à jour de l'élément avec la méthode GET de l'API



Outils — Postman

Outil – Postman pour tester la méthode DELETE de l'API



HTTP Status codes

HTTP Status Codes

<http://www.restapitutorial.com/httpstatuscodes.html>

This page is created from HTTP status code information found at ietf.org and Wikipedia. Click on the **category heading** or the **status code** link to read more.

1xx Informational

100 Continue

2xx Success

★ 200 OK

203 Non-Authoritative Information

206 Partial Content

226 IM Used

3xx Redirection

300 Multiple Choices

303 See Other

306 (Unused)

4xx Client Error

★ 400 Bad Request

★ 403 Forbidden

406 Not Acceptable

★ 409 Conflict

412 Precondition Failed

415 Unsupported Media Type

418 I'm a teapot (RFC 2324)

423 Locked (WebDAV)

426 Upgrade Required

431 Request Header Fields Too Large

450 Blocked by Windows Parental Controls (Microsoft)

101 Switching Protocols

★ 201 Created

★ 204 No Content

207 Multi-Status (WebDAV)

301 Moved Permanently

★ 304 Not Modified

307 Temporary Redirect

★ 401 Unauthorized

★ 404 Not Found

407 Proxy Authentication Required

410 Gone

413 Request Entity Too Large

416 Requested Range Not Satisfiable

420 Enhance Your Calm (Twitter)

424 Failed Dependency (WebDAV)

428 Precondition Required

444 No Response (Nginx)

451 Unavailable For Legal Reasons

102 Processing (WebDAV)

202 Accepted

205 Reset Content

208 Already Reported (WebDAV)

302 Found

305 Use Proxy

308 Permanent Redirect (experimental)

402 Payment Required

405 Method Not Allowed

408 Request Timeout

411 Length Required

414 Request-URI Too Long

417 Expectation Failed

422 Unprocessable Entity (WebDAV)

425 Reserved for WebDAV

429 Too Many Requests

449 Retry With (Microsoft)

499 Client Closed Request (Nginx)

5xx Server Error

★ 500 Internal Server Error

503 Service Unavailable

506 Variant Also Negotiates (Experimental)

509 Bandwidth Limit Exceeded (Apache)

598 Network read timeout error

501 Not Implemented

504 Gateway Timeout

507 Insufficient Storage (WebDAV)

510 Not Extended

599 Network connect timeout error

502 Bad Gateway

505 HTTP Version Not Supported

508 Loop Detected (WebDAV)

511 Network Authentication Required

★ "Top 10" HTTP Status Code. More REST service-specific information is contained in the entry.

Méthodes

Invoque la méthode correspondante au verbe HTTP :

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	404 (Not Found), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	404 (Not Found), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	404 (Not Found), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Exemple 2

Ajout d'un service Web à l'application qui permet de gérer des employées :

On souhaite ajouter un service Web (API) à l'application qui permet de gérer des employées. Chaque employé est défini par son identifiant, son nom, son prénom, son sexe, son département et sa ville.

Le service web doit permettre de :

- Retourner tous les employés.
- Retourner les détails d'un employé.
- Ajouter un nouvel employé.
- Modifier un employé.
- Supprimer un employé.

Le service Web doit utiliser la même logique d'accès aux données développée dans l'application MVC dans le cours précédent. L'API doit gérer :

- ✓ Les employés stockés dans une collection d'employés de type *Dictionary* dont la clé de chaque employé est représentée par son identifiant.
- ✓ Les employés stockés dans une base de données locale SQL Server Express.

Exemple 2

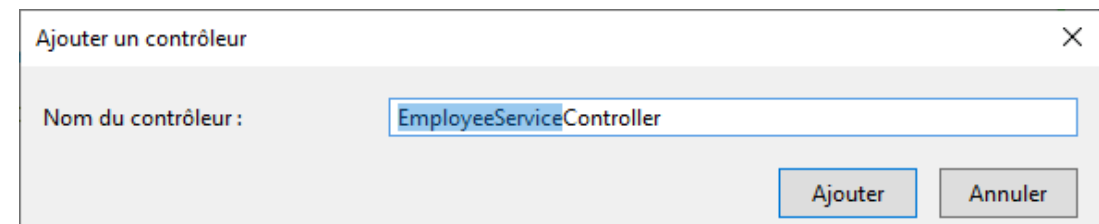
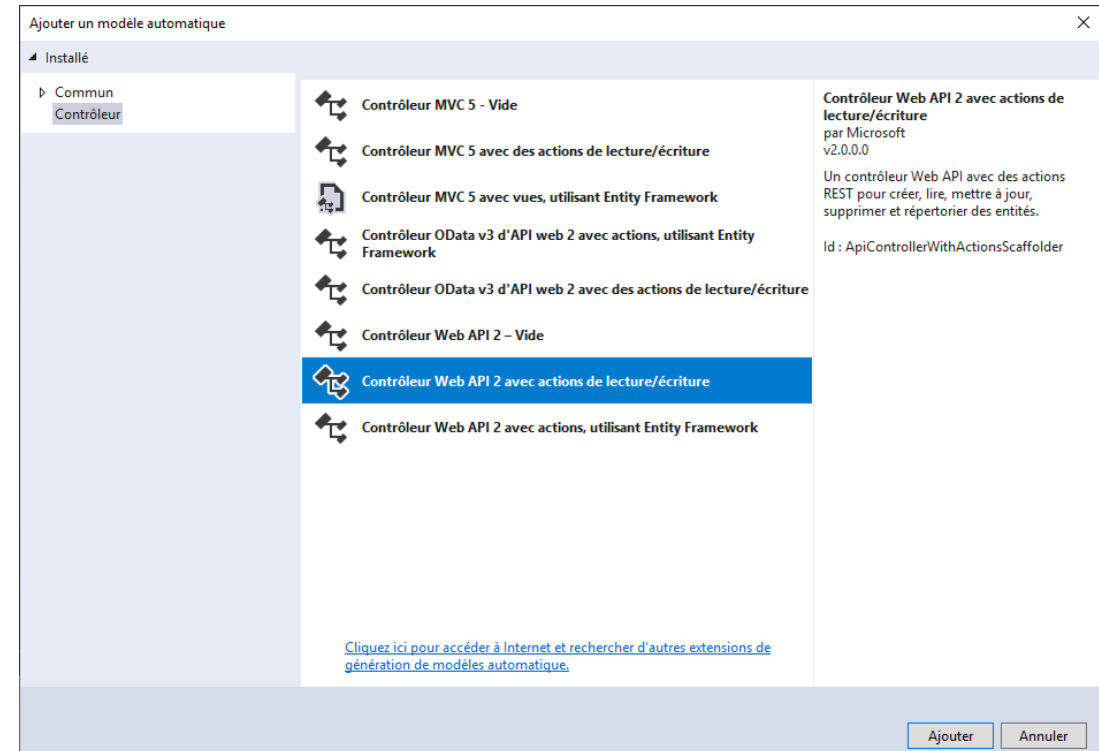
Ajout d'un autre contrôleur à l'application :

Faire un clic droit sur le dossier «Controllers» du projet >> Ajouter >> Contrôleur.

Sélectionner le modèle Contrôleur Web API 2 avec des actions de lecture/écriture.

Visual studio 2017 vas créer automatiquement les méthodes d'actions suivantes :

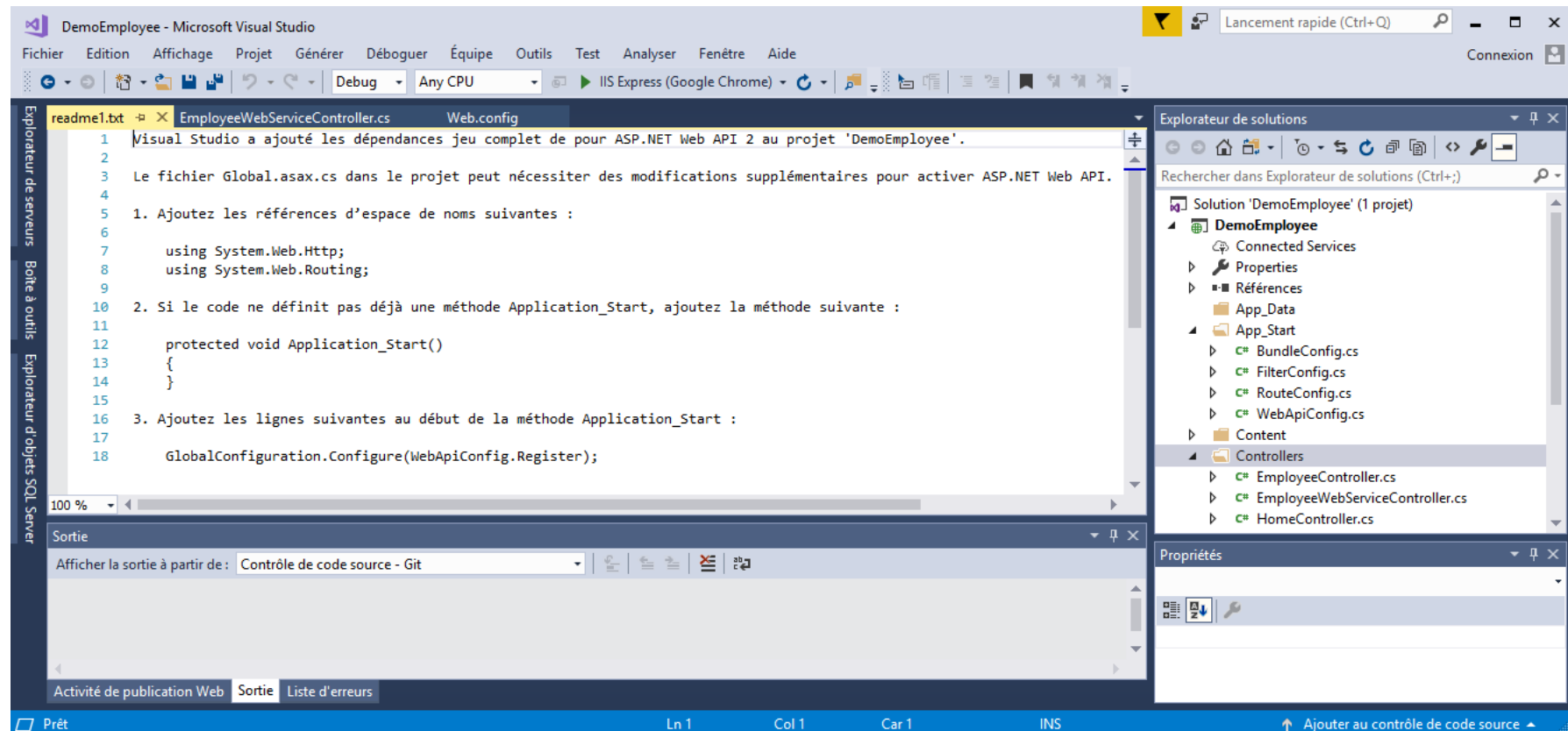
- IEnumerable<string> Get()
- string Get(int id)
- Void Post([FromBody]string value)
- Void Put(int id, [FromBody]string value)
- void Delete(int id)



Exemple 2

Ajout d'un autre contrôleur à l'application :

Visual studio affiche un fichier readme1.txt pour indiquer ce qui nous devons ajouter pour faire fonctionner le contrôleur ajouté.



Exemple 2

Ajoutons au début de la méthode `Application_Start()` la ligne suivante :

```
GlobalConfiguration.Configure(WebApiConfig.Register);

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace WebApiDemo
{
    public class WebApiApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            GlobalConfiguration.Configure(WebApiConfig.Register);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

Exemple 2

Modifions le contrôleur EmployeeService et ajoutons du code pour toutes les méthodes

```
public class EmployeeServiceController : ApiController
{
    IEmployeeDataAccess employeeDataAccess { get; set; }
    public EmployeeServiceController()
    {
        employeeDataAccess = new EmployeeDataAccessImp1();
    }
    // GET: api/EmployeeService
    public IHttpActionResult Get()
    {
        IList <Employee> employees = employeeDataAccess.GetAllEmployees().ToList();
        if(employees.Count == 0)
        {
            return Content(HttpStatusCode.NotFound, "Employees not found");
        }
        return Ok(employees);
    }
    // GET: api/EmployeeService/5
    public IHttpActionResult Get(int id)
    {
        Employee employee = null;
        employee=employeeDataAccess.GetEmployeeData(id);
        if (employee == null)
            return Content(HttpStatusCode.NotFound, "Employee not found");
        return Ok(employee);
    }
}
```

Exemple 2

```
// Suite de contrôleur
// POST: api/EmployeeService
public IActionResult Post([FromBody]Employee employee)
{
    if (!ModelState.IsValid || employee == null)
        return BadRequest("Invalid data.");
    return Ok(employeeDataAccess.AddEmployee(employee));
}

// PUT: api/EmployeeService/5
public IActionResult Put(int id, [FromBody]Employee employee)
{
    if (!ModelState.IsValid)
        return BadRequest("Not a valid model.");
    Employee emp = null;
    emp = employeeDataAccess.GetEmployeeData(id);
    if (emp != null)
        employeeDataAccess.UpdateEmployee(employee);
    else
        return NotFound();
    return Ok();
}

// Suite de contrôleur
// DELETE: api/EmployeeService/5
public IActionResult Delete(int id)
{
    if (id <= 0)
        return BadRequest("Not a valid employee id");
    Employee emp = null;
    emp = employeeDataAccess.GetEmployeeData(id);
    if (emp != null)
        employeeDataAccess.DeleteEmployee(id);
    else
        return NotFound();
    return Ok();
}
```

Exemple 2

Une version simplifiée du contrôleur EmployeeService2 sans validation est aussi ajoutée dans le projet

```
public class EmployeeService2Controller : ApiController
{
    IEmployeeDataAccess employeeDataAccess { get; set; }
    public EmployeeService2Controller()
    {
        employeeDataAccess = new EmployeeDataAccessImp1();
    }
    // GET: api/EmployeeService2
    public IEnumerable<Employee> Get()
    {
        return employeeDataAccess.GetAllEmployees().ToList();
    }

    // GET: api/EmployeeService2/5
    public Employee Get(int id)
    {
        return employeeDataAccess.GetEmployeeData(id);
    }

    // POST: api/EmployeeService2
    public void Post([FromBody]Employee employee)
    {
        employeeDataAccess.AddEmployee(employee);
    }

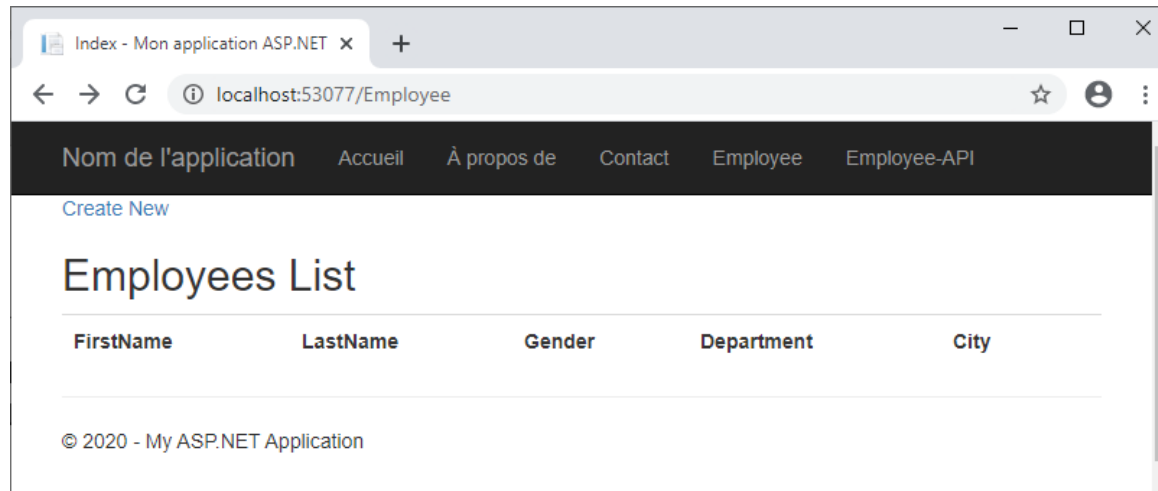
    // Suite de contrôleur
    // PUT: api/EmployeeService2/5
    public void Put(int id, [FromBody]Employee employee)
    {
        employeeDataAccess.UpdateEmployee(employee);
    }

    // DELETE: api/EmployeeService2/5
    public void Delete(int id)
    {
        employeeDataAccess.DeleteEmployee(id);
    }
}
```

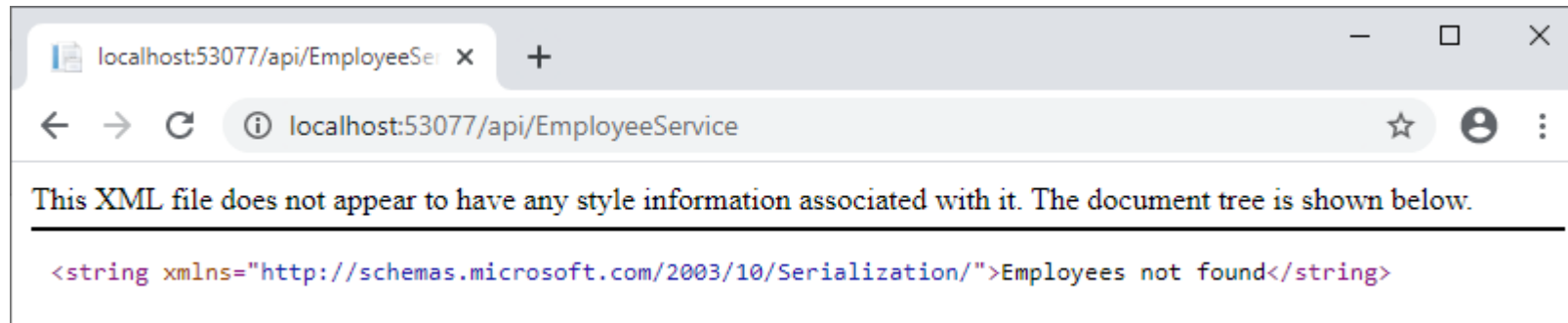

Exemple 2

Exécution

L'exécution de l'application avec URL `http://localhost:53077/Employee` donne la page de la figure suivante.

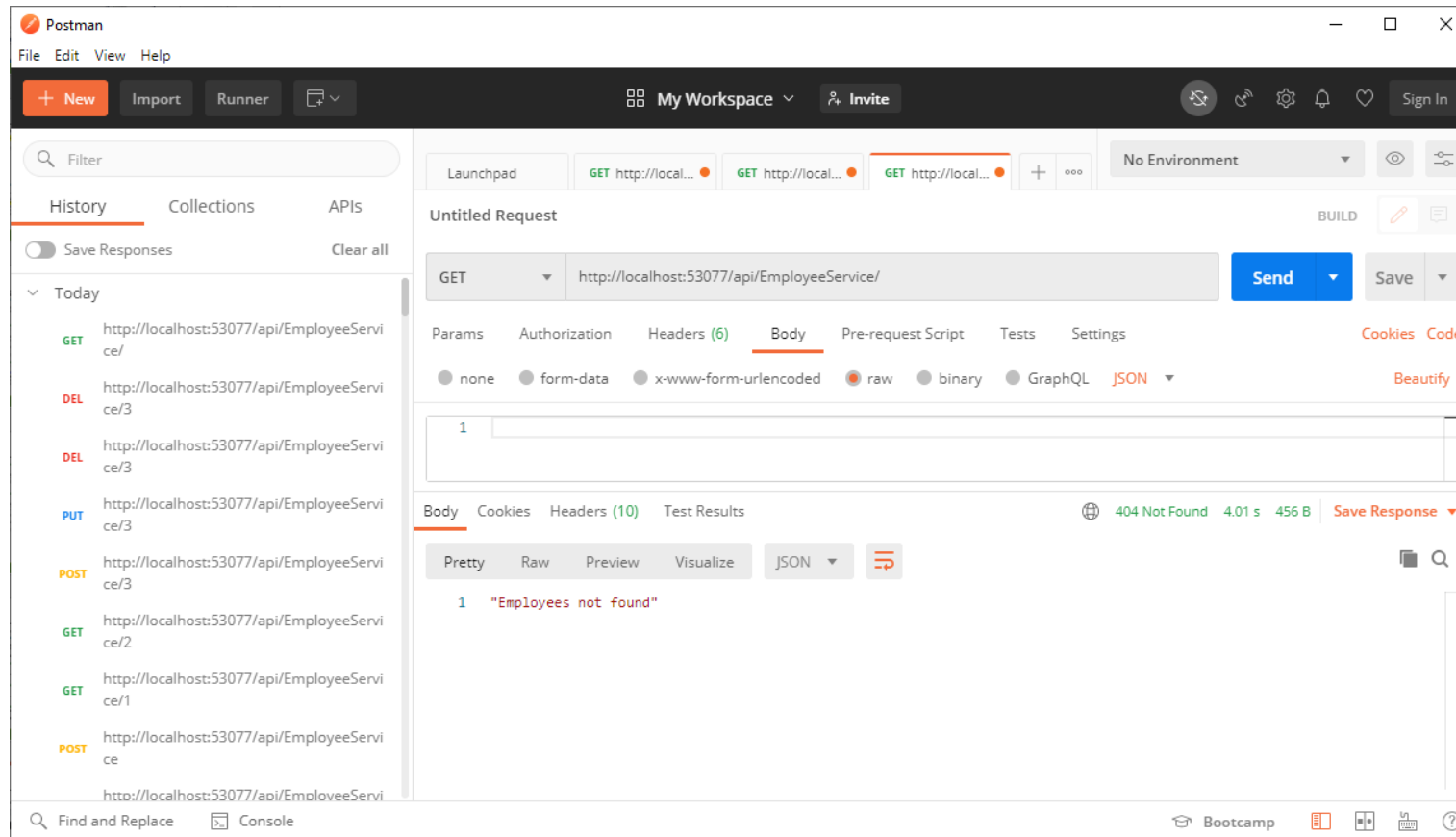


L'exécution de l'application avec URL `http://localhost:53077/api/EmployeeService` donne la page de la figure suivante.



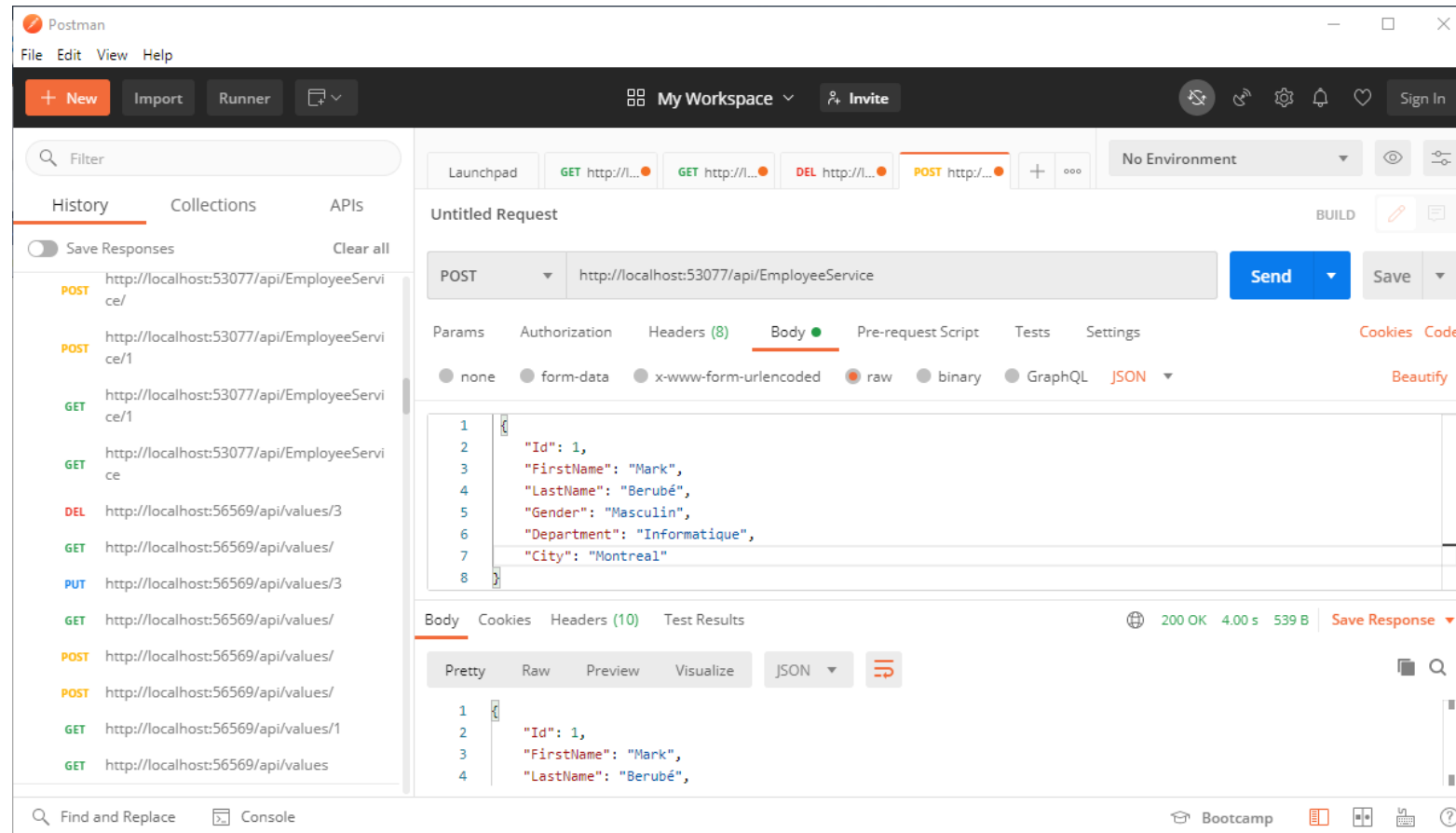
Outils — Postman

Outil – Postman pour tester la méthode GET de l'API



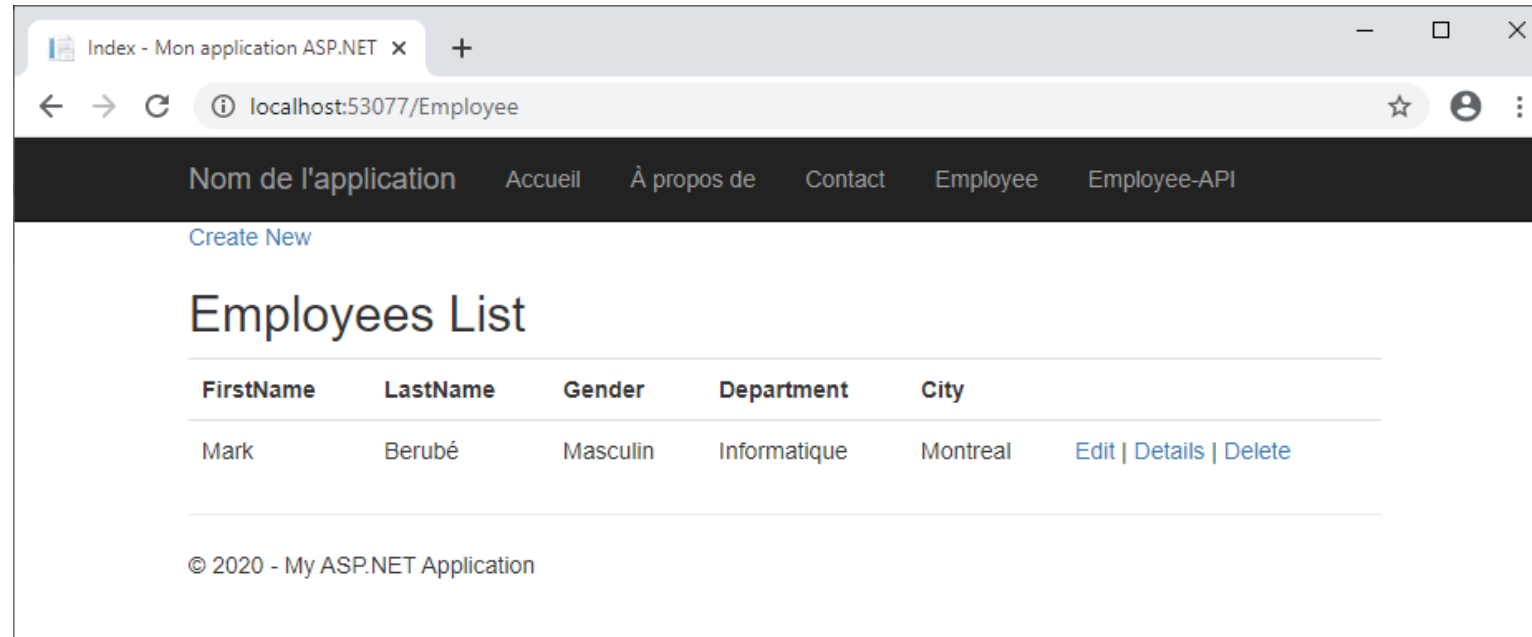
Outils — Postman

Outil – Postman pour tester la méthode POST de l'API



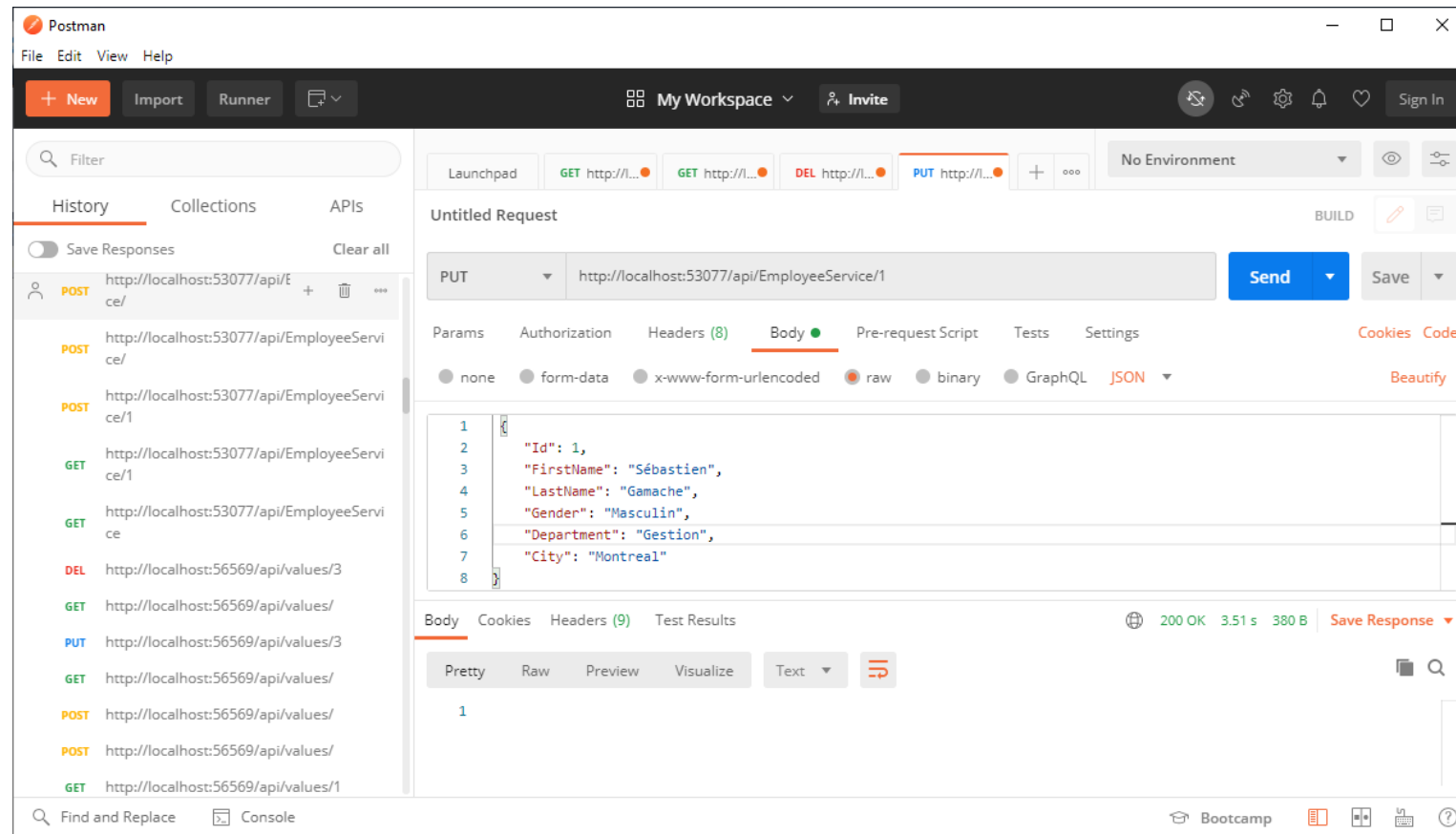
Outils — Postman

Validation d'ajout avec l'application MVC



Outils — Postman

Outil – Postman pour tester la méthode PUT de l'API



Outils — Postman

Validation de modification avec l'application MVC

