

Programmation avancée en c#.NET

Ing.Meryem OUARRACHI

Plan du module

Programmation WEB

- ☐ Généralités outils Web
- ☐ ASP MVC
- ☐ ASP.Net core
- ☐ Angular

Génie logiciel en .Net

BI en Self Service

Programmation distribuée avancée

- ☐ Web API
- ☐ GraphQL

CHAPITRE 4:

ASP Web API

ASP.Net Web API

C'est quoi API (Interface de programmation d'application)?

- Ensemble de définitions de sous-programmes, de protocoles et d'outils pour la construction de logiciels et d'applications.
- Donc API est une sorte d'interface qui a un ensemble de fonctions qui permettent aux programmeurs d'accéder à des fonctionnalités spécifiques ou des données d'une application, d'un système d'exploitation ou d'autres services.
- L'API Web, comme son nom l'indique, est une API sur le Web accessible via le protocole HTTP. C'est un concept et non une technologie. Nous pouvons créer une API Web en utilisant différentes technologies telles que Java, .NET, etc.

ASP.Net Web API

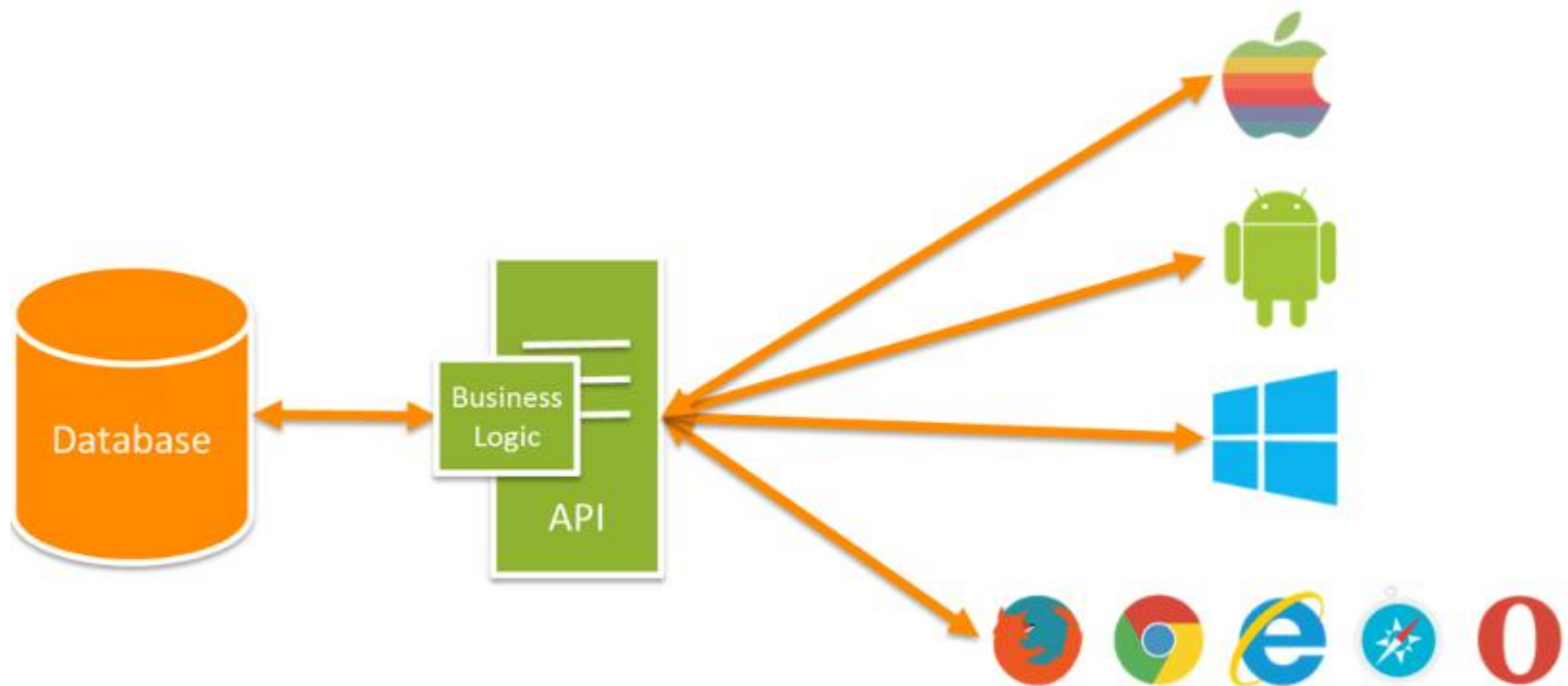
- **En général:**

C'est un concept permettant de créer une couche de services càd un ensemble de méthodes peuvent être utiliser par d'autres applications —————> On parle alors d'une application de service web

- **Plus précisément:**

ASP.NET Web API est un framework qui facilite la création de services HTTP pouvant atteindre un large éventail de clients, y compris les navigateurs et les appareils mobiles. Il utilise les services de RESTful.

ASP.Net Web API



ASP.Net Web API vs WCF

- ✱ WCF utilise les services SOAP et le format XML
- ✱ Web API utilise les service RESTful et le format Json+Autres formats

-L'API Web est beaucoup plus simple et facile à utiliser:car il suffit de taper l'adresse de service pour qu'on puisse l'utiliser par contre en wcf il y'a un proxy à mettre en place

-L'API Web est légère et convient parfaitement aux périphériques qui ont un nombre limité de bande passante comme les smartphones:car coté taille json est plus léger

ASP.Net Web API vs WCF

-WCF est beaucoup plus polyvalent car il peut fonctionner avec plusieurs canaux de transport tcp udp → on l'utilise si on veut créer un service qui doit prendre en charge des scénarios spéciaux tels que la messagerie unidirectionnelle, les files d'attente de messages, la communication duplex, etc.

-On choisit l'API Web lorsque on veut créer des services orientés ressources via HTTP qui peuvent utiliser toutes les fonctionnalités de HTTP (comme les URI, les en-têtes de requête / réponse, la mise en cache, la gestion des versions, divers formats de contenu).

Web API vs WCF Rest

- Web API est une continuité du projet WCF Rest.
- Ce dernier a été conçu pour utiliser les services Restful mais il ne supporte que le Get et le Post les autres verbs nécessitent une configuration spéciale dans le serveur web(IIS)
- Alors ASP Web API était une solution pour utiliser toutes les services d'une manière simple et facile sans passer par des configurations supplémentaires
- WCF--→WCF Rest --→ASP Web API

Restful(Representational State Transfer)

- En web API Le protocole http utilise les règles de Rest afin d'assurer l'échange de message dans un environnement distribué .
- Rest est un style d'architecture qui repose sur le protocole HTTP.Il accède à une ressource (par son URI unique) pour procéder à diverses opérations supportées par HTTP.
- RESTful est généralement utilisé pour désigner les services Web implémentant l'architecture Rest

Les méthodes de requêtes Http

-HTTP définit un ensemble des méthodes de requête qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée. Ces méthodes sont souvent appelées verbes HTTP.

Méthode	Rôle	Type de requête
Get	utilisée afin de récupérer des données	Requête select
Post	utilisée pour envoyer une entité vers la ressource indiquée	Requête Create
PUT	remplace toutes les représentations actuelles de la ressource visée par le contenu de la requête	Requête Update
DELETE	supprime la ressource indiquée	Requête Delete

Le protocole http

- **Requête HTTP:**

- **Header:**

- Méthode à exécuter (get, post...)
- *Accept*: Format de réponse (xml ou json, html...)
- *Content-Type*: Format de requête (xml ou json...)

- **Body:**

Contient les données à envoyer avec les requêtes

L e protocole http

- **Reponse HTTP:**

- **Header:** Contient des informations sur le serveur
- **Body**
- **Status Code:** Les codes d'état de réponse HTTP indiquent si une requête HTTP spécifique a été exécutée avec succès ou non.

Code	Signification	Exemple
200	Ok:La demande a réussi	GET: La ressource a été récupérée et transmise dans le corps du message.
201	créé	La demande a réussi et une nouvelle ressource a été créée à la suite de cela. C'est généralement la réponse envoyée après une requête POST, ou après certaines requêtes PUT
4XX	il y'a un problème	Le serveur ne peut pas trouver la ressource demandée. (401)

Création de projet ASP Web API

- En VS Créer nouveau projet de type ASP web API
- Nouveau controleur→Choisir controleur API vide→nommé Service1
- Ce controleur hériter la classe ApiController

```
public class Service1Controller : ApiController
{
    public string Get()
    {
        return "valeur1";
    }
}
```

Routage de projet ASP Web API

-Par la suite dans l'url on met <http://localhost:2331/api/Service1>

On a la possibilité de configurer le controlleur de démarrage dans le fichier App_Start / WebApiConfig

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

        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "Service1Api",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { controller = "Service1", id = RouteParameter.Optional }
        );
    }
}
```

-Aussi on peut mettre <http://localhost:2331/api> car on a définit Service1 comme contrôleur de démarrage

Routage de projet ASP Web API

-Il y'a aussi la possibilité de définir le routage lors de la définition de l'action via l'attribut Route de la manière suivante:

```
public class Service1Controller : ApiController
{
    [Route("api/MonPremierService/Action1")]
    public string Get()
    {
        return "valeur1";
    }
}
```

-Pour l'appel on met:

<http://localhost:2331/api/MonPremierService/Action1>

-Résultat d'exécution

```
<string>valeur1</string>
```


Création des méthodes Http

Après qu'on définit le routage, on commence à écrire les méthodes http(Get, Post, Put, Delete) qui vont être utilisées par les clients afin de faire des traitements sur les ressources concernées par le web API crée.

-Les actions doivent commencer par Get, post put ou delete sinon il faut précéder le nom de l'action par les attributs [HttpGet] ou [HttpPost]....

MediaTypeFormatter

- Les MediaTypeFormatter sont des classes chargées de sérialiser les données de requête / réponse afin que l'API Web puisse comprendre le format de données de la requête et envoyer les données dans le format attendu par le client.
- Au cas d'utilisation d'EF n'oublier pas d'ajouter les attributs [DataMember] et [DataContract]
- Le format xml est pris par défaut ,pour basculer à json
`config.Formatters.Remove(config.Formatters.XmlFormatter);`

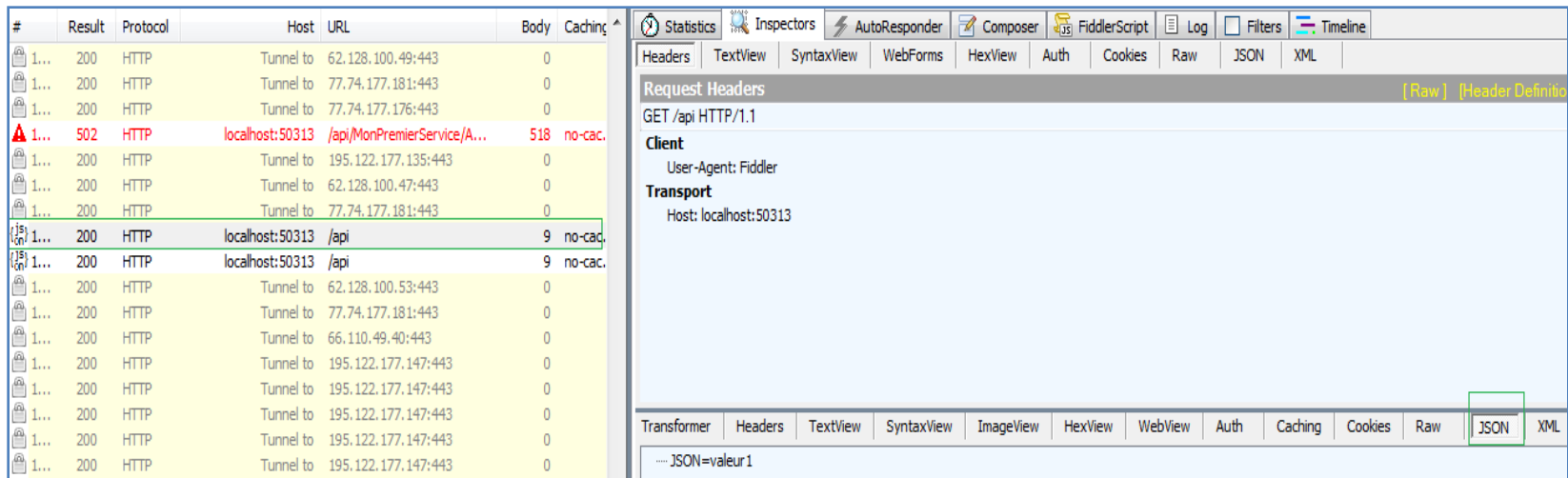
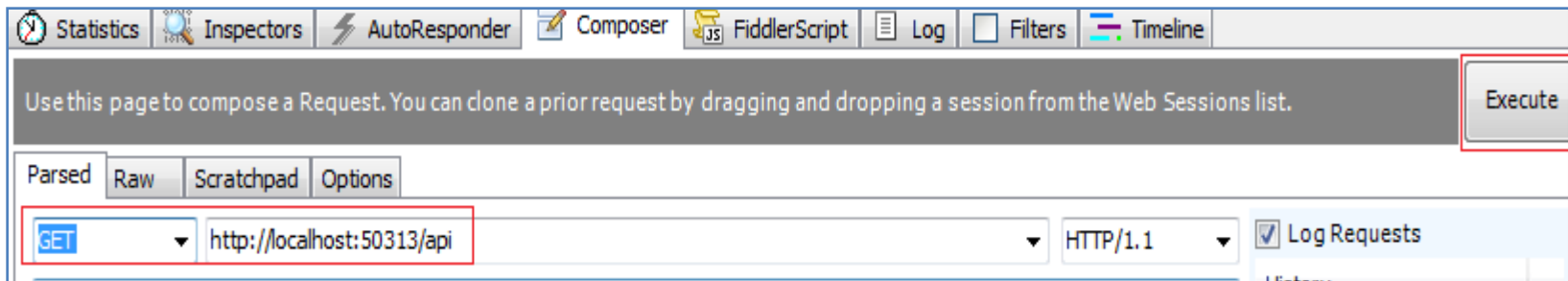
Test de projet ASP Web API

On aura besoin d'un outil qui nous permet de tester en local le détail de la requête et la réponse http développé (équivalent de test Client en WCF)

- **Fiddler**: C'est un outil gratuit de débogage Web qui enregistre tout le trafic HTTP en inspectant le trafic et manipulant les données entrantes et sortantes.
- **Installation**: <https://www.telerik.com/download/fiddler>

Test de projet ASP Web API

-Test de l'action précédente sous Fiddler:ouvrir fiddler→onglet Composer→écrire l'uri de l'action à exécuter



Liaison des paramètres

Q:Comment lier les paramètres des actions avec les requêtes http?

R:Web API lie les paramètres des actions soit à travers les queryString de l'URL,soit avec le corps de la requête(Body Request) en suivant la règle par défaut suivante:

Type primitif	→	Liaison avec Query String
Type complexe	→	Liaison avec Body Request

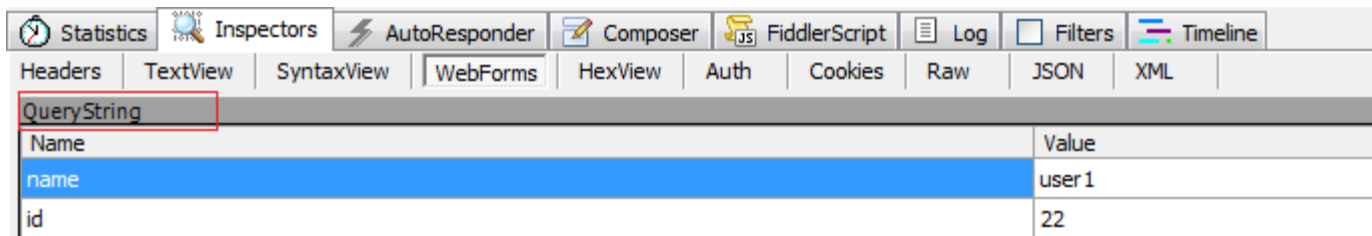
Liaison des paramètres

Exemple1: QueryString

```
public string Get2(string name,int id)
{
    return "Nom="+name +"Identifiant="+id;
}
```

-Appel:

<http://localhost:50313/api/Service1?name=user1&id=22>



The screenshot shows the Fiddler web debugging tool interface. The 'Inspectors' pane is active, and the 'QueryString' tab is selected. It displays a table of query string parameters. The 'name' parameter has a value of 'user1' and the 'id' parameter has a value of '22'.

Name	Value
name	user1
id	22

Liaison des paramètres

Remarques:

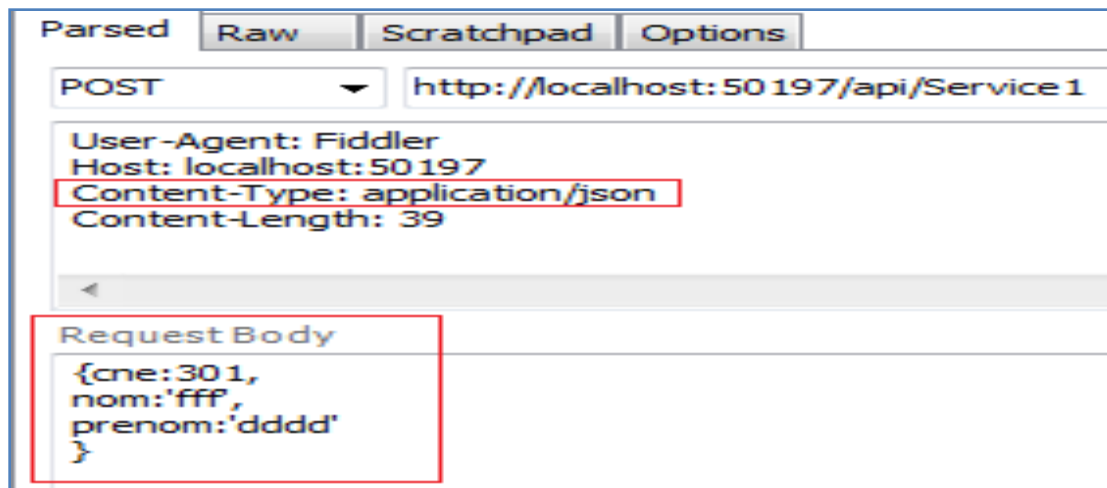
- Les noms des paramètres doivent être identiques à celle utilisé lors d'appel en url sans être obligé de respecter la casse
- Lors de l'appel ce n'est pas obligé de respecter l'ordre des paramètres.

Liaison des paramètres

Exemple2: BodyRequest

```
public void Postetudiant(etudiant etudiant)
{
    db.etudiants.Add(etudiant);
    db.SaveChanges();
}
```

-Appel: <http://localhost:50313/api/Service1>



Liaison des paramètres

Si on veut changer le comportement par défaut de web api dans la liaison des données on utilise les attributs [FromUri] et [FromBody]

[FromUri]	→	Lire un type Complexe à partir le query String
[FromBody]	→	Lire un type primitif à partir le BodyRequest

Liaison des paramètres

- **Exemple:** [FromUri]

```
public void Postetudiant([FromUri]etudiant etudiant)
{
    db.etudiants.Add(etudiant);
    db.SaveChanges();
}
```

-Appel: <localhost:50313/api/Service1?cne=1&nom=ddd1&prenom=gg>

- **Exempl2:** [FromBody]

```
public void Post([FromBody]int cne)
{
}
}
```

Liaison des paramètres

- Remarques:

- BodyRequest ne supporte qu'un seul paramètre.

- Le verbe GET peut prendre des paramètres de requête uniquement à partir des chaînes de requête.

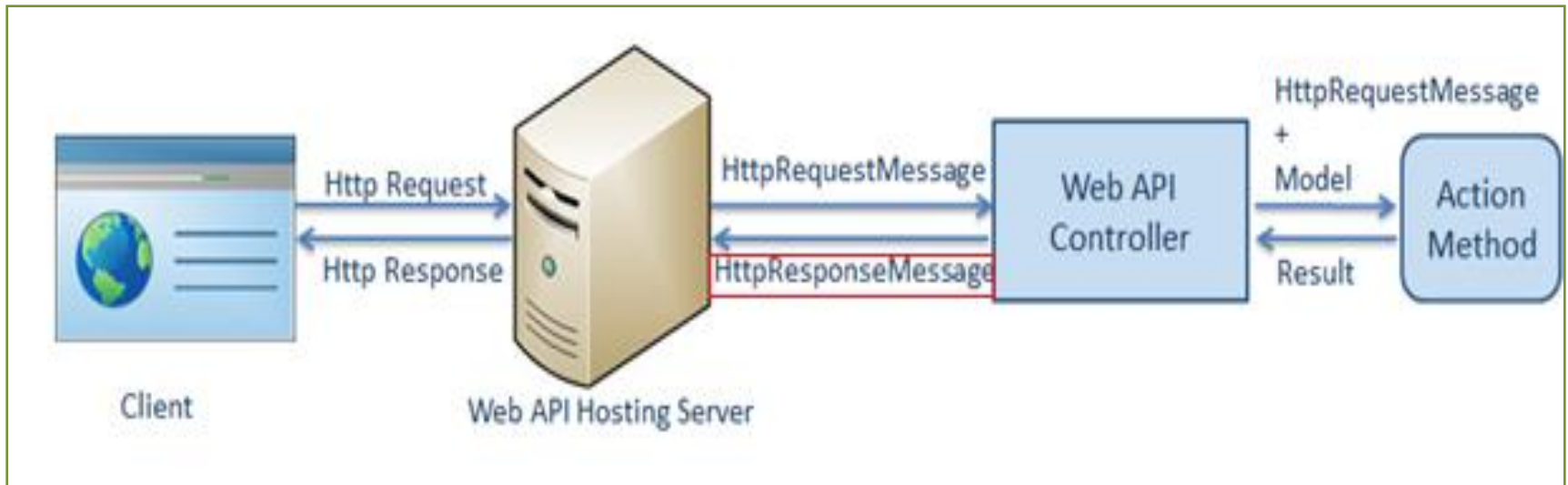
Les valeurs de retour

Les actions de web API peuvent retourner:

- Void
- Type primitifs/Complexe
- HttpResponseMessage
- IHttpActionResult

Les valeurs de retour

- HttpResponseMessage



L'avantage d'envoyer HttpResponseMessage à partir d'une méthode d'action permet de configurer une réponse comme on souhaite. On peut définir le code d'état, le contenu ou le message d'erreur (le cas échéant) selon nos besoins.

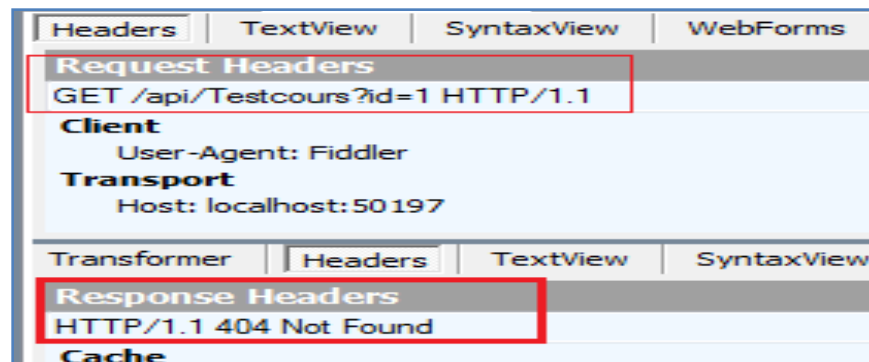
Les valeurs de retour

- HttpResponseMessage:

-Exemple1:

```
public HttpResponseMessage Getetudiant(int id)
{
    etudiant etudiant = db.etudiants.Find(id);
    if (etudiant == null)
    {
        return Request.CreateResponse(HttpStatusCode.NotFound, id);
    }
    return Request.CreateResponse(HttpStatusCode.OK, etudiant);
}
```

-Test sous Fiddler:



Les valeurs de retour

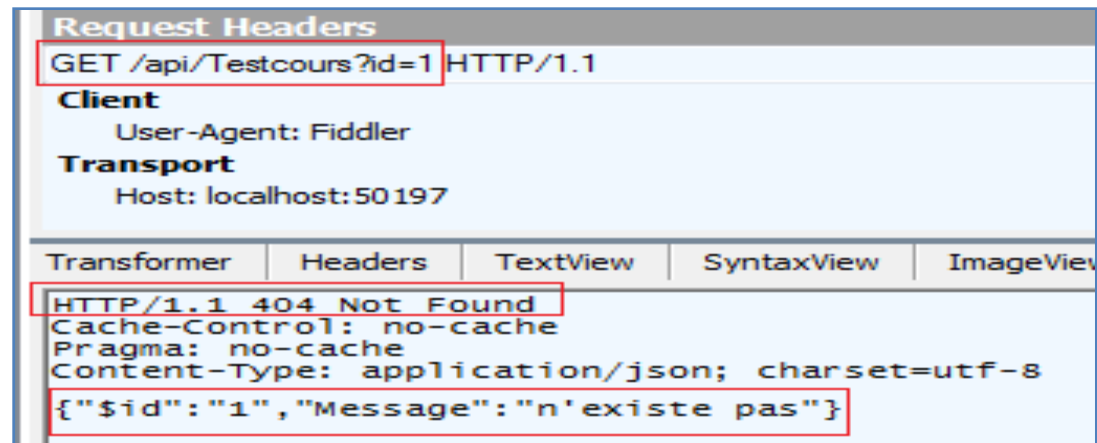
- HttpResponseMessage:

-Exemple2:

```
public HttpResponseMessage Getetudiant(int id)
{
    etudiant etudiant = db.etudiants.Find(id);
    if (etudiant == null)
    {
        return Request.CreateErrorResponse(HttpStatusCode.NotFound, "n'existe pas");
    }

    return Request.CreateResponse(HttpStatusCode.OK, etudiant);
}
```

-Test sous Fiddler:



Les valeurs de retour

- IHttpActionResult:

-Il a été introduit dans Web API 2 . Une méthode d'action dans Web API 2 peut renvoyer une implémentation de la classe IHttpActionResult qui est similaire à la classe ActionResult dans ASP.NET MVC.

-Il s'agit d'une interface qui définit un ensemble des méthodes permettant d'envoyer des réponses HttpResponseMessage avec moins de code.

Les valeurs de retour

- IHttpActionResult:

Méthode ApiController	Description
BadRequest()	Crée un BadRequest Result .
Conflict()	Crée un Conflict Result (Conflit 409).
Content()	Crée un NegotiatedContent Result avec les valeurs spécifiées.
CreatedAtRoute()	Crée un CreatedAtRouteNegotiatedContent Result (201 Créé) avec les valeurs spécifiées.
NotFound()	Crée un NotFound Result .
Ok()	Crée un Ok Result (200 OK).
ResponseMessage()	Crée un ResponseMessage Result avec la réponse spécifiée.
StatusCode()	Crée un StatusCode Result avec le code d'état spécifié.

Les valeurs de retour

- IHttpActionResult:

-Exemple:

```
public IHttpActionResult Get(int id)
{
    etudiant etudiant = db.etudiants.Find(id);
    if (etudiant == null)
    {
        return NotFound();
    }

    return Ok(etudiant);
}
```

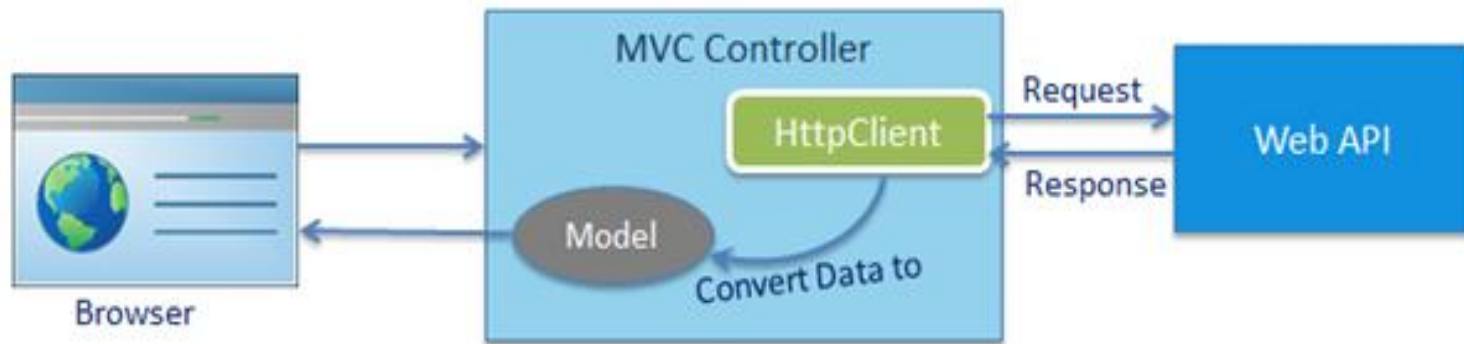
-Resultat sous Fiddler:

Transformer	Headers	TextView	SyntaxView
HTTP/1.1 404 Not Found			
Cache-Control: no-cache			
Pragma: no-cache			

Transformer	Headers	TextView
JSON		
cne=11		
nom=Falahi		
prenom=noure		
sexe=f		
tof=/Content/images/e1.jpg		

Consommation de Web API

Etapes de consommation



1. Dans le contrôleur MVC Utiliser **HttpClient** qui est responsable d'envoyer la requête à WebAPI puis il reçoit la réponse.
2. Effectuer les conversions nécessaires des données de réponse provenant de l'API Web en modèle.
3. Afficher la réponse dans la vue.

Consommation de méthode Get

-Exemple de méthode Get

```
//dataContext généré par EF
private tp1ADOEntities db = new tp1ADOEntities();

// GET: api/etudiants
public IEnumerable<etudiant> Getetudiants()
{
    return db.etudiants;
}
```

Consommation de méthode Get

-Consommation de méthode Get

```
public ActionResult Index()
{
    IEnumerable<etudiant> x;

    //1. Instancier un objet de HttpClient responsable de communication avec le webApi
    HttpClient client = new HttpClient();
    //2. Déterminer l'adresse de WebApi
    client.BaseAddress = new Uri("http://localhost:50197");

    //3. Préciser le format de réponse souhaité
    client.DefaultRequestHeaders.Accept.
        Add(new System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));

    //4. GetAsync:récupérer les résultats
    HttpResponseMessage response = client.GetAsync("api/etudiants").Result;

    //5. ReadAsAsync:lire le résultat
    x = response.Content.ReadAsAsync<IEnumerable<etudiant>>().Result;
    return View( x);
}
```

Consommation de méthode Post

-Exemple de méthode Post

```
public IHttpActionResult Postetudiant(etudiant etudiant)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.etudiants.Add(etudiant);
    db.SaveChanges();
    return CreatedAtRoute("DefaultApi", new { id = etudiant.cne }, etudiant);
}
```

Méthode Post

-Exemple de méthode Post: Plus simple

```
public HttpResponseMessage PostEtudiant(etudiant et)
{
    try
    {
        db.etudiants.Add(et);
        db.SaveChanges();
        HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.Created);
        return response;
    }
    catch (Exception ex)
    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex.Message);
    }
}
```


Méthode Post

-Consommation de méthode Post

```
public ActionResult Creer(etudiant etudiant)
{
    //1. Instancier un objet de HttpClient
    HttpClient client = new HttpClient();
    client.BaseAddress = new Uri("http://localhost:50197");

    client.DefaultRequestHeaders.Accept
        .Add(new System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));

    //2. PostAsJsonAsync: exécuter la méthode post, lui passer les paramètres dans le 2ème arg
    var message = client.PostAsJsonAsync("api/etudiants", etudiant).Result;

    return RedirectToAction("index");
}
```

Méthode Put

-Exemple de méthode Put:

```
public HttpResponseMessage PutEtudiant(etudiant etudiant)
{
    db.Entry(etudiant).State = System.Data.Entity.EntityState.Modified;
    db.SaveChanges();
    HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK);
    return response;
}
```

Méthode Put

-Consommation de méthode Put:

```
public ActionResult Modifier(etudiant etudiant)
{
    int id = etudiant.cne;
    HttpClient client = new HttpClient();
    client.BaseAddress = new Uri("http://localhost:50197");

    HttpResponseMessage response = client.PutAsJsonAsync("api/etudiants", etudiant).Result;

    return RedirectToAction("Index");
}
```

Méthode Delete

-Exemple de méthode Delete:

```
public IHttpActionResult Deleteetudiant(int id)
{
    etudiant etudiant = db.etudiants.Find(id);
    if (etudiant == null)
    {
        return NotFound();
    }

    db.etudiants.Remove(etudiant);
    db.SaveChanges();

    return Ok(etudiant);
}
```

Méthode Delete

-Consommation de méthode Delete:

```
public ActionResult Delete(int id)
{
    HttpClient client = new HttpClient();
    client.BaseAddress = new Uri("http://localhost:50197/");

    //HTTP DELETE
    var message = client.DeleteAsync("api/etudiants/" + id.ToString()).Result;

    if (message.IsSuccessStatusCode)
    {
        return RedirectToAction("Index");
    }

    return RedirectToAction("Index");
}
```