API REST

Développement web REST API en C#

L'objectif de cette présentation est de présenter les concepts clés du développement web REST API en C#.

Cette présentation est divisée en plusieurs parties :

- Qu'est-ce que REST API
- Architecture REST API
- Requêtes HTTP
- Création d'une API REST

Qu'est-ce que REST API

REST signifie Representational State Transfer, et une API REST est une interface de programmation de l'application qui utilise les protocoles HTTP pour obtenir ou envoyer des données.

Toutes les API REST utilisent les protocoles HTTP pour envoyer et recevoir des données.

Dans une API REST, les données sont représentées sous forme de ressources.

Concepts clés

Les concepts clés d'une API REST incluent :

- Utilisation des protocoles HTTP pour envoyer et recevoir des données.
- Utilisation d'URIs pour identifier les ressources.
- Utilisation de différents types de requêtes HTTP pour effectuer des opérations différentes sur les ressources (GET, POST, PUT, DELETE).

Utilisation dans le développement web

Les API REST sont largement utilisées dans le développement web pour intégrer différents services et applications.

Elles permettent de communiquer entre les différentes parties sans avoir besoin d'une connaissance précise des systèmes sous-jacents.

Architecture REST API

Structure des URL

La structure des URL est un élément clé pour une API REST.

Les URIs doivent être conçues de manière à identifier clairement les ressources et à effectuer les opérations appropriées sur celles-ci.

Types de requêtes HTTP

Les API REST utilisent différents types de requêtes HTTP pour effectuer des opérations différentes sur les ressources.

Les types les plus couramment utilisés sont :

- GET : obtenir une ressource.
- POST : créer une ressource.
- PUT : mettre à jour une ressource.
- DELETE : supprimer une ressource.

Mise en place de l'architecture REST API

Voici les étapes à suivre :

- Création du projet.
- Structure du projet.
- Les URI.
- Contrôleurs.
- Modèles.
- Base de données.

Création du projet

Pour créer un projet de développement web REST API en C#, nous allons utiliser Visual Studio Code.

Il est possible de créer un projet de développement web REST API en C# en tapant la commande suivante dans le terminal :

```
dotnet new webapi -o API
```

Structure du projet

```
├──API
│    ├──Controllers
│    ├──Data
│    ├──Models
│    └──Properties
├──Ressources
│    └──Theme
└──Test
      └──API
```

Les URI

Les URI sont utilisées pour identifier les ressources.

Les URI doivent être conçues de manière à identifier clairement les ressources et à effectuer les opérations appropriées sur celles-ci.

Une URI doit être unique et doit être facile à comprendre.

Contrôleurs

Les contrôleurs sont des classes qui contiennent les actions qui sont appelées lorsqu'une requête HTTP est reçue.

La seule différence entre les contrôleurs asp.net et les contrôleurs web API est que les contrôleurs web API ne retournent pas de vue.

Ils retournent directement des données sous forme de JSON.

Les JSON

Les JSON sont organisés sous forme de paires clé-valeur.

Exemple :

```
{
    "nom": "Dupont",
    "prenom": "Jean",
    "age": 25
}
```

Les JSON

JSON signifie JavaScript Object Notation.

Il s'agit d'un format de données textuel qui est utilisé pour échanger des données entre les systèmes.

Son utilisation est très courante dans le développement web, il représente une alternative aux formats XML.

Modèles

Pour transmettre des données entre les différentes parties, nous allons utiliser des modèles.

Exemple :

```
public class Personne
{
        public string Nom { get; set; }
        public string Prenom { get; set; }
        public int Age { get; set; }
}
```

Modèles

Dans le cadre de ce projet, nous pouvons etre amenés à renvoyer des classes de modèles sous forme de JSON.

Pour cela, nous allons utiliser l'attribut [ApiController] qui permet de renvoyer des données sous forme de JSON.

Cet attribut doit être ajouté au début de la classe.

Tout ce qui est renvoyé par les actions du contrôleur sera automatiquement converti en JSON.

Création des contrôleurs pour les requêtes HTTP

Une fois le projet créé, nous allons créer les contrôleurs pour les requêtes HTTP.

Chaque contrôleur doit contenir une action pour chaque type de requête HTTP.

Si une requête HTTP est reçue pour une URI qui correspond à une action d'un contrôleur, l'action correspondante est appelée.

Exemple de contrôleur pour les requêtes HTTP

```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
                // ...
        }
}
```

Ajout des actions pour les requêtes HTTP

Les actions pour les requêtes HTTP doivent être ajoutées dans les contrôleurs.

Elles représentent les opérations qui seront effectuées sur les ressources a l'aide des requêtes HTTP.

Si une requête HTTP est reçue pour une URI qui correspond à une action d'un contrôleur, l'action correspondante est appelée.

Test de l'API REST

Pour tester l'API REST, nous allons utiliser l'outil Postman.

Postman est un outil de test d'API REST qui permet de tester les requêtes HTTP.

Il est possible de télécharger Postman à l'adresse suivante : https://www.postman.com/downloads/

GET

Pour tester la requête GET, nous allons utiliser l'URI suivante :

```
http://localhost:5000/api/values
```

Cette URI correspond à l'action Get() du contrôleur ValuesController.

Il renvoie une liste de chaînes de caractères.

Methode GET

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


Préparation de la requête GET

Pour préparer la requête GET, nous allons utiliser l'outil Postman.

Nous allons créer une nouvelle requête GET.

Cette requête doit être envoyée à l'URI suivante :

```
http://localhost:5000/api/values
```


Les fichiers JSON

Les fichiers JSON sont des fichiers qui contiennent des données sous forme de JSON.

```
[
    {
        "nom": "Dupont",
        "prenom": "Jean",
        "age": 25
    }
]
```

POST

La requête POST permet de créer une ressource.

Dans le cas d'une requête POST, nous devons envoyer via le corps de la requête les données qui seront utilisées pour créer la ressource.

Nous pouvons utiliser plusieurs fois la même URI tant que la méthode HTTP est différente.

Methode POST

Exemple de méthode POST :

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

Le [FromBody]

L'attribut [FromBody] permet de définir a l'aide d'une annotation que les données sont reçues dans le corps de la requête.

Si l'attribut [FromBody] n'est pas utilisé, les données sont reçues dans l'URI.

L'utilisation des données reçues dans l'URI n'est pas recommandée.

Methode PUT

La requête PUT permet de mettre à jour une ressource.

Si la ressource n'existe pas, elle est créée.

Quand on utilise la requête PUT, nous devons envoyer via le corps de la requête les données qui seront utilisées pour mettre à jour la ressource.

Methode PUT

Exemple de méthode PUT :

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

Les paramètres

Les paramètres sont des données qui sont envoyées dans l'URI.

Ils permettent de définir des valeurs qui seront utilisées dans l'action.

Exemple :

```
http://localhost:5000/api/values/1
```


Swagger

Swagger est un outil qui permet de générer une documentation pour une API REST.

Swagger permet aussi de venir tester l'API REST.

Chaque action d'un contrôleur pourra être testée directement depuis la documentation.

Il suffira juste de rajouter le `/swagger` à la fin de l'URI.

La méthode DELETE

La requête DELETE permet de supprimer une ressource.

Si la ressource n'existe pas, une erreur est renvoyée.

Exemple de méthode DELETE :

```
[HttpDelete("{id}")]
public void Delete(int id)
{
        // ...
}
```

IEnumerable

L'interface IEnumerable représente une collection d'objets qui peut être parcourue.

Losqu'une méthode renvoie un objet de type IEnumerable, cela signifie que la méthode renvoie une collection d'objets.

Notre projet transformera tous les objets de type IEnumerable en JSON, nous allons donc utiliser cette interface pour renvoyer les données.

IEnumerable

Exemple de méthode qui renvoie un objet de type IEnumerable :

```
[HttpGet]
public IEnumerable<string> Get()
{
        return new string[] { "value1", "value2" };
}
```


Configuration des routes

Pour configurer les routes, nous allons utiliser l'attribut [Route] qui permet de définir l'URI de la route.

Exemple :

```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
}
```

L'arborecence des routes

L'attribut [Route] permet de définir l'URI de la route.

Il est possible de définir des routes imbriquées afin de créer une arborescence.

Cette arborescence permettra de définir une logique dans l'URI.

L'arborecence des routes

Par exemple, si nous avons des objets les uns dans les autres, nous pouvons créer une arborescence dans l'URI.

Exemple :

```
http://localhost:5000/api/clients/1/commandes/2
```


L'arborecence des routes

Exemple de configuration des routes :

```
[Route("api/[controller]")]
[ApiController]
public class ClientsController : ControllerBase
{
        [Route("{id}/commandes")]
        public IEnumerable<Commande> GetCommandes(int id)
        {
                // ...
        }
}
```

Les paramètres facultatifs

Il est possible de définir des paramètres facultatifs dans l'URI.

Pour définir un paramètre facultatif, il suffit de rajouter un `?` après le nom du paramètre.

Si le paramètre n'est pas renseigné, il prendra la valeur par défaut.

Les paramètres facultatifs

Exemple de paramètre facultatif qui prend la valeur par défaut :

```
[HttpGet]
public IEnumerable<string> Get([FromQuery] int id = 1)
{
        return new string[] { "value1", "value2" };
}
```