XML et Architectures Logicielles Distribuées

Baabda :
M. E. ABOU ZEID
M. E. MATTA

Zahlé :
Mlle. C. SAAD

Mejdelaya:
M. R. ABI NADER

PLAN DU COURS

- Chapitre 1: Le Langage XML
- Chapitre 2: XPath et APIs XML
- **Chapitre 3: Les services web**
- Chapitre 4: Les architectures distribuées
- Chapitre 5: Le Schéma XML

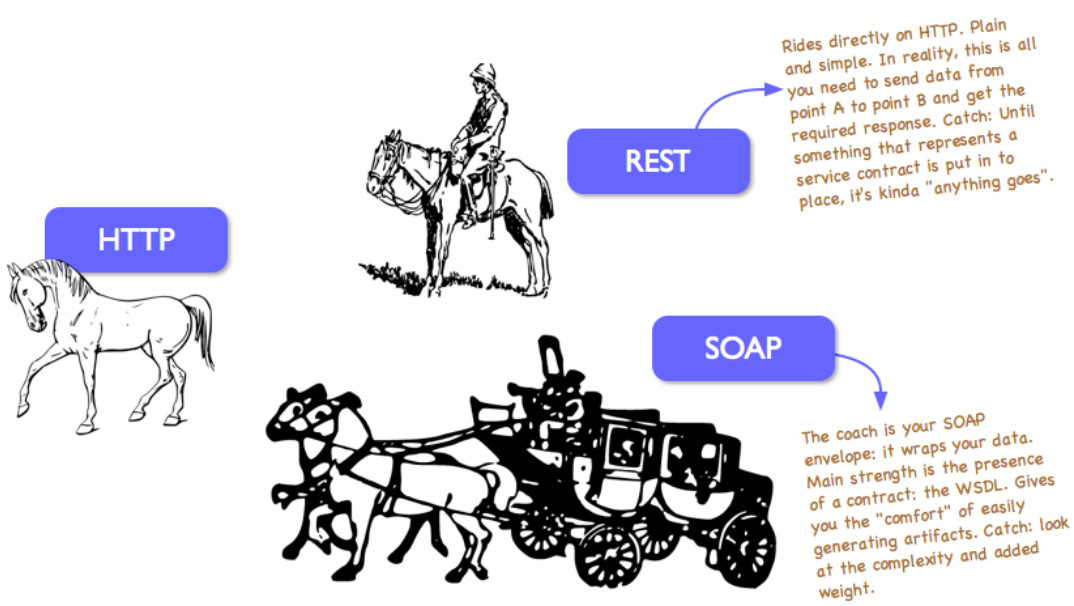
CHAPITRE 3

Les Services Web

Plan Du Chapitre

- Vue globale sur les services web
- MVC Web API
- Le Routage dans MVC Web API
- Étude de Cas

Vue globale sur les services web



Vue globale sur les services web

- Applications **logicielles, un mécanisme**
  - Interopérable et indépendant de tout langage de programmation et de toute plate-forme d'exécution
  - utilisant le protocole HTTP comme *moyen de transport*
- Permettent la communication **à distance** entre des applications **via le réseau Internet**
- Identification par un **URI**
- Base pour construire des **systèmes distribués**
- **Peuvent être:**
  - **Publiées**: pour permettre l'accès au serveur pour les utilisateurs distants
  - **Localisées**: découvertes
  - **Invoquées**: consommées

Vue globale sur les services web

	REST	SOAP
Signification	**RE**presentational **S**tate **T**ransfer	**S**imple **O**bject **A**ccess **P**rotocol
Définition	Style Architectural (client-serveur)	Protocole
Formats d'échange	JSON, XML, etc.	XML uniquement
Transport	Utilise le protocole HTTP simple	Utilise l'enveloppe SOAP puis HTTP pour le transfert de données – "encapsulation"
Méthodes HTTP	POST, GET, PUT, DELETE	POST
Sécurité	• Ne propose pas de fonctions de sécurité, cryptage, gestion des sessions, fiabilité, etc. • Alternatives: utiliser des méthodes personnalisées pour l'authentification/HTTPS/ etc.	• Fournit des caractéristiques WS-*: Adressage, Sécurité, fiabilité, etc.
Autres	• Stateless • Performance • Extensibilité • Caching • Très répandu et utilisé	• Stateless • Performance inférieure • Extensibilité complexe • Pas de Caching • Utilisé lorsque REST n'est pas utilisable

Vue globale sur les services web

SAMPLE: Common operations on news item object	SOAP approach	RESTful approach
Create news item	CreateNewsItem(string id, string title)	/news.svc/{id} HTTP METHOD: **PUT**
Update news item	UpdateNewsItem(string id)	/news.svc/{id} HTTP METHOD: **PUT**
Get news item	GetNewsItem(string id, string title)	/news.svc/{id} HTTP METHOD: **GET**
Get news items	GetNewsItems()	/news.svc/ HTTP METHOD: **GET**
Delete news item	DeleteNewsItem(string id)	/news.svc/{id} HTTP METHOD: **DELETE**

Exposition de la logique métier

- **REST** fournit un **accès à des ressources** via le **URI** et la méthode employée (POST, GET, DELETE, etc.)
  - *Similaire aux opérations CRUD*
- Avec **SOAP** on emploie des **interfaces définissant des fonctions** avec leurs signatures
  - *Contrat WSDL entre client et service*

MVC Web API

MVC Web API

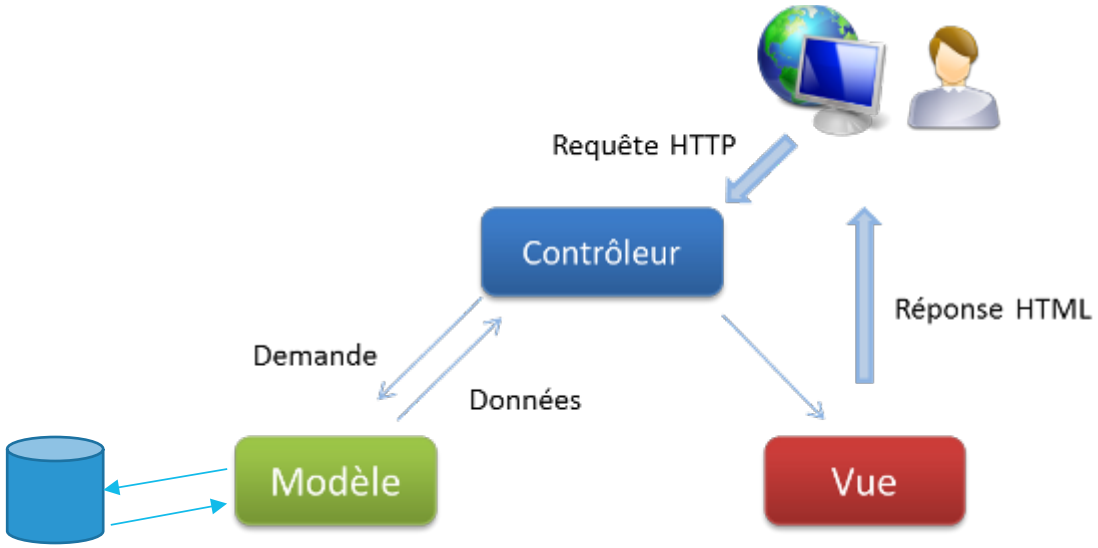
Le patron d'architecture « MVC »

- Patron d'architecture logicielle Modèle-Vue-Contrôleur
- «***Model-View-Controller***»

- Modèle destiné à répondre aux besoins des applications interactives **en séparant les problématiques** liées aux différents composants au sein de leur architecture respective
  - «***separation of concerns***»

Avantages
✓ séparation des problématiques et conception plus efficace
✓ éléments plus simples à réutiliser qui ne dépendent que d'un contexte
✓ maintenance et extensibilité facilitées
✓ Développement multi-développeurs plus aisée
✓ Etc.

MVC Web API



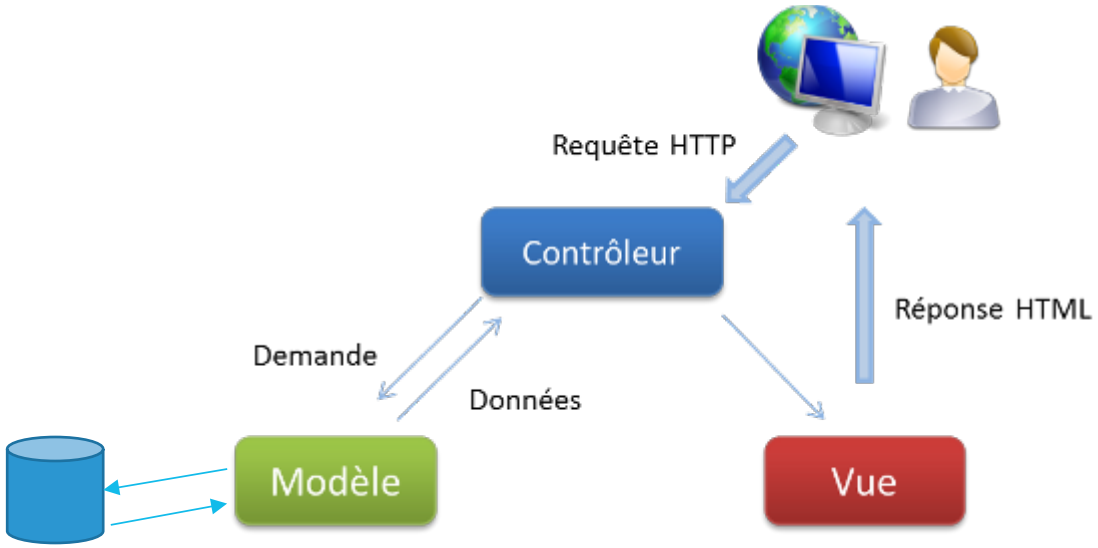
Le patron d'architecture « MVC »

- **Se divise en trois composants:**
  1. **Modèle:**
     - ✓ Définit les modèles de données (classes) requis pour l'affichage dans la 'Vue'
     - ✓ Fournit les données au contrôleur suite à une communication avec la base de données
     - ✓ Notifie le contrôleur lorsque les données sont modifiées

MVC Web API

Le patron d'architecture « MVC »

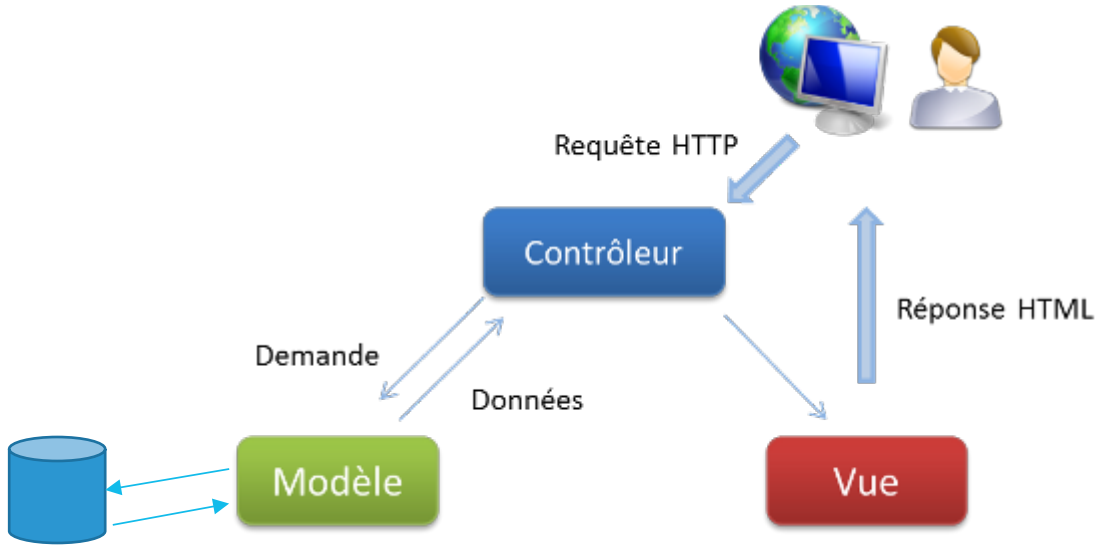
- **Se divise en trois composants:**
  - **2. Vue ('UI logic')**
    - présente les données à l'utilisateur via une interface graphique
    - Reflète l'état du modèle (l'état de ses attributs)
    - Transmet les interactions utilisateur au contrôleur



MVC Web API

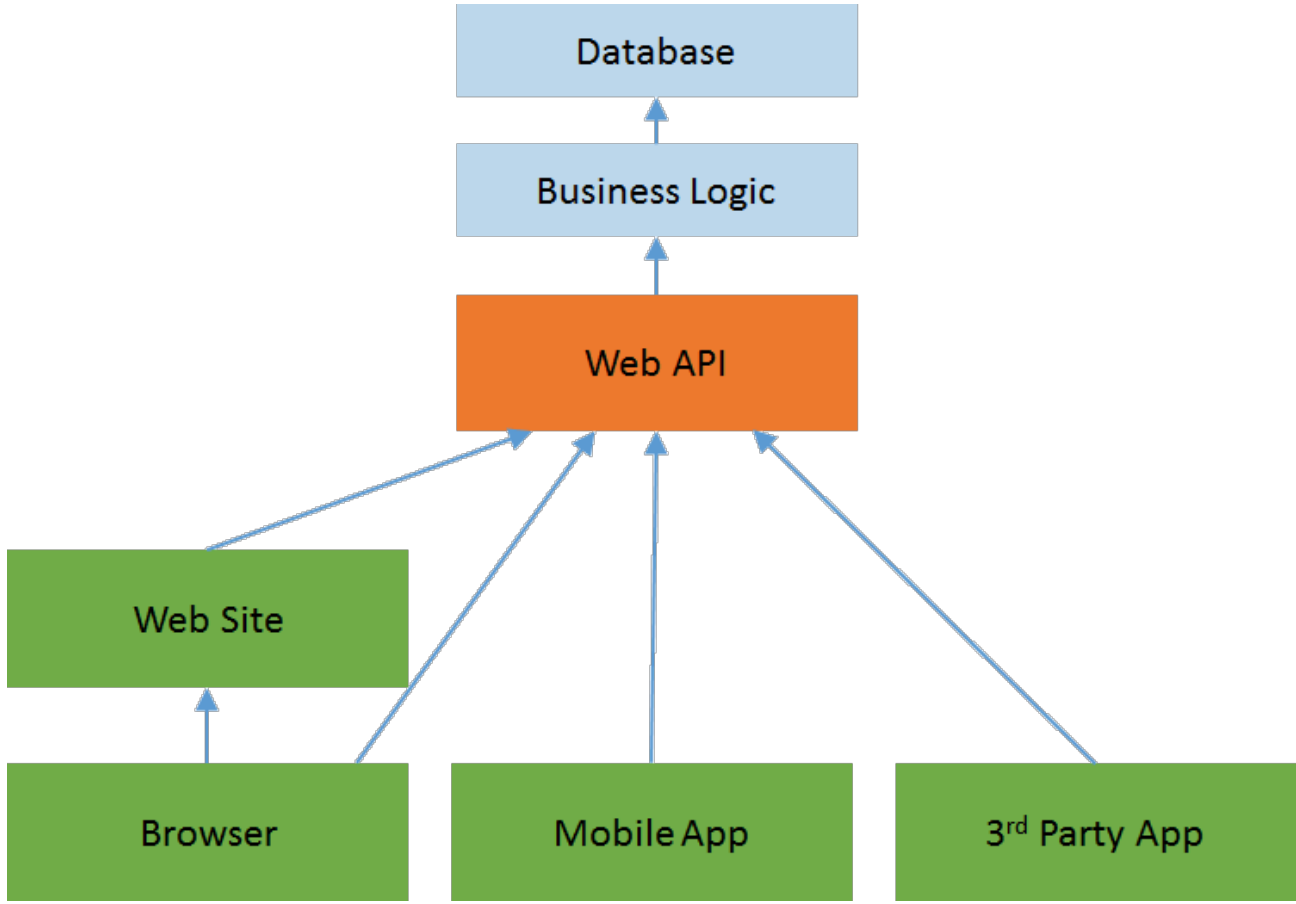
Le patron d'architecture « MVC »

- **Se divise en trois composants:**
  - **3. Contrôleur**
    - Gère les requêtes HTTP des clients
    - Communique les requêtes de données au Modèle
    - Notifie la Vue lorsque les données sont obtenues suite à une requête



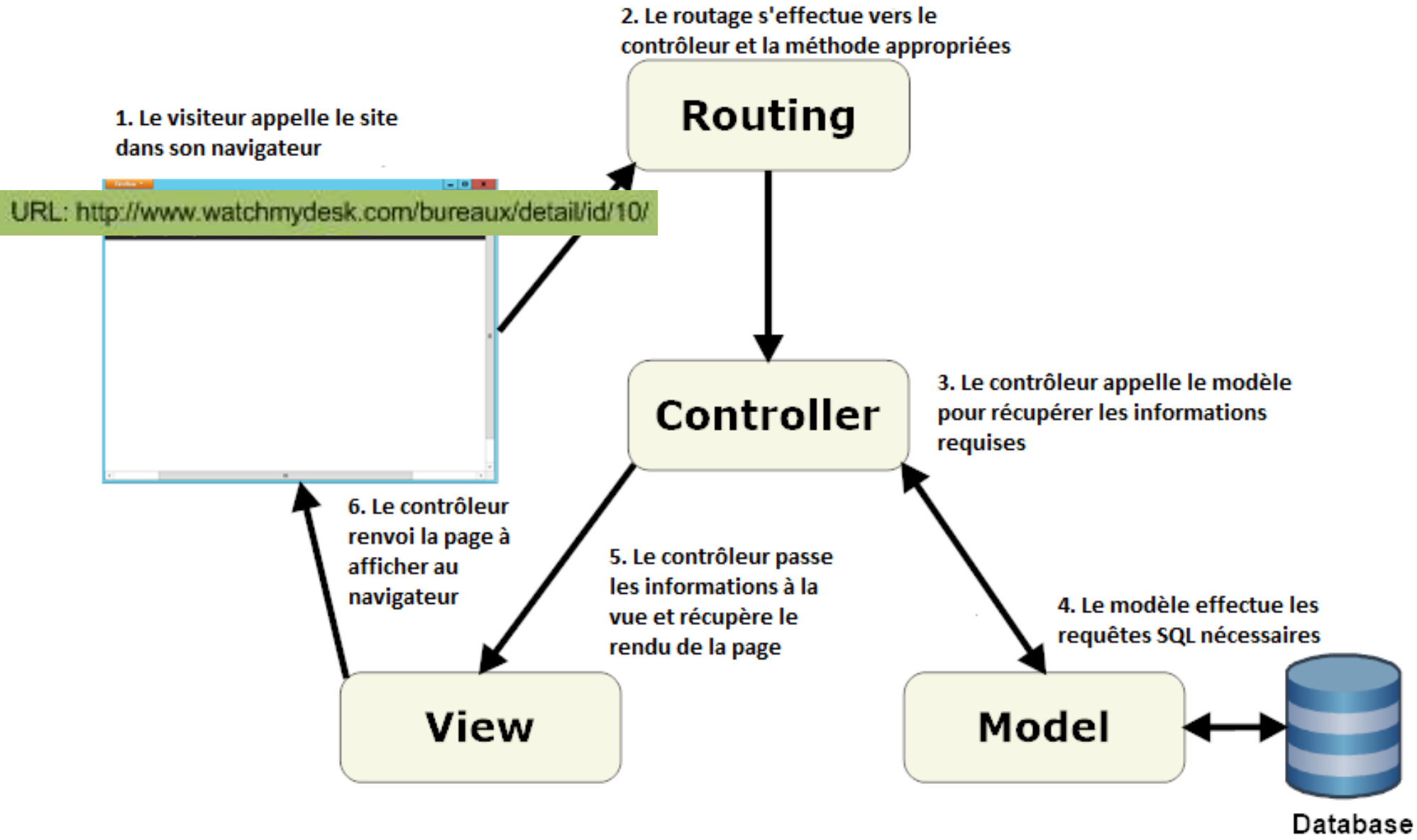
MVC Web API

- **Web API**
  - Infrastructure qui facilite le développement de services HTTP disponibles sur un large éventail de clients
    - Navigateurs
    - Appareils mobiles
    - Etc.
  - plateforme idéale pour développer des applications **RESTful** sous **.NET**



Le Routage dans MVC Web API

Le Routage dans MVC Web API



Le Routage dans MVC Web API

- **Le routage**
  - Un mécanisme permettant d'analyser une requête HTTP reçue par le service REST en fonction du contexte HTTP dans le but de déterminer quel contrôleur et quel action doivent être appelées
    - Une **action** est une méthode publique dans la classe **contrôleur**
  - La configuration du routage nécessite la configuration deux fichiers:
    1. Project\App_Start\WebApiConfig.cs
    2. Project\Global.asax
  - Deux types de routages:
    - **Le routage conventionnel** (*Conventional-Based Routing*)
    - **Le routage par attribut** (*Attribute Routing*) - MVC 5.0 et plus

Le Routage dans MVC Web API

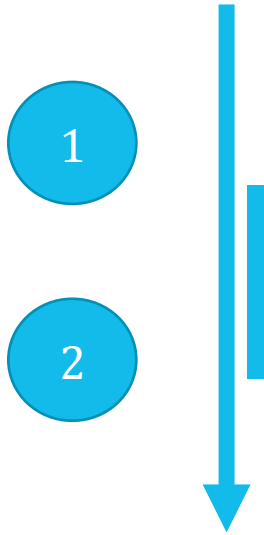
- **Le routage conventionnel**
  - Un mécanisme dans lequel on traite les requêtes reçues en se servant d'une table de routage qui permet d'effectuer les correspondances

Le Routage dans MVC Web API

- **Le routage conventionnel – Étape 1 pour la configuration**

```
public class WebApiConfig
{
    1 reference
    public static void Register(HttpConfiguration config)
    {
        config.Routes.MapHttpRoute(
            name: "Custom",
            routeTemplate: "api/{controller}/{id}/{action}",
            defaults: new { id = RouteParameter.Optional }
        );

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



La recherche de la route correspondante se fait dans l'ordre

Project\App_Start\WebApiConfig.cs

Le Routage dans MVC Web API

- **Le routage conventionnel – Étape 2 pour la configuration**

```
public class Global : System.Web.HttpApplication
{
    0 references
    protected void Application_Start(object sender, EventArgs e)
    {
        GlobalConfiguration.Configure(WebApiConfig.Register);
    }
}
```

Project\Global.asax

Le Routage dans MVC Web API

Le routage conventionnel

- **Une fois la route est trouvée dans la table de routage:**
  - Pour trouver le contrôleur, le Web API ajoute "Controller" à la valeur de la variable {controller}.
  - Pour trouver l'action, le Web API détecte la méthode HTTP en cours, puis cherche une {action} dont le nom commence par celui de la méthode HTTP.
    - *Avec une requête GET, on cherche une action qui commence par "Get ...", tels que "GetContact" ou "GetAllContacts" (valide avec GET, POST, PUT et DELETE)*
  - D'autres variables sont réservées dans le modèle de la route, tels que {id}, sont mappés à des paramètres d'action.

Le Routage dans MVC Web API

- Au lieu d'utiliser la convention de noms des méthodes HTTP dans le nom de l'action, on peut utiliser les annotations:
  - **[HttpGet]**, **[HttpPut]**, **[HttpPost]**, ou **[HttpDelete]**

```
public class ProductsController : ApiController
{
    [HttpGet]
    public Product FindProduct(id) {}
}
```

Le Routage dans MVC Web API

- On peut même modifier le nom de l'action en utilisant l'attribut "*ActionName*" qui sera le nom effectif

```
public class ProductsController : ApiController
{
    [HttpGet]
    [ActionName("Thumbnail")]
    public HttpResponseMessage GetThumbnailImage(int id);

    [HttpPost]
    [ActionName("Thumbnail")]
    public void AddThumbnailImage(int id);
}
```

Le Routage dans MVC Web API

- **Exemple**:

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

URI	http://localhost:2285/api/User/{id}/Profile
Méthode HTTP	GET
Contrôleur	*User* (UserController.cs)
Action	*Profile*
Paramètre	id (identifiant)

Le Routage dans MVC Web API

- **Le routage par attribut**
  - Un mécanisme dans lequel on utilise des attributs de routage afin de configurer **directement** les routes de l'application sur:
    - les actions des contrôleurs
    - les contrôleurs eux-mêmes
  - Ce mécanisme peut remplacer ou se combiner avec le routage par convention
  - On utilise des attributs comme:
    - **Route** : enregistrement du pattern de la route, des valeurs par défaut et des différentes contraintes

Le Routage dans MVC Web API

- **Le routage par attribut – Étape 1 pour la configuration**

```
public class WebApiConfig
{
    1 reference
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();
    }
}
```

Project\App_Start\WebApiConfig.cs

Le Routage dans MVC Web API

- **Le routage par attribut – Étape 2 pour la configuration**

```
public class Global : System.Web.HttpApplication
{
    0 references
    protected void Application_Start(object sender, EventArgs e)
    {
        GlobalConfiguration.Configure(WebApiConfig.Register);
    }
}
```

Project\Global.asax

Le Routage dans MVC Web API

- **Le routage par attribut – L'attribut Route**

```
public class BooksController : ApiController
{
    [Route("api/books")]
    public IEnumerable<Book> GetBooks() { ... }

    [Route("api/books/{id:int}")]
    public Book GetBook(int id) { ... }

    [Route("api/books")]
    [HttpPost]
    public HttpResponseMessage CreateBook(Book book) { ... }
}
```

- *N.B.: Lorsqu'on ne spécifie pas une Route au dessus de l'action, elle sera adoptée comme action par défaut pour la méthode HTTP correspondante (GET/POST/..)*

Le Routage dans MVC Web API

- **Le routage par attribut – L'attribut RoutePrefix**

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET api/books
    [Route("")]
    public IEnumerable<Book> Get() { ... }

    // GET api/books/5
    [Route("{id:int}")]
    public Book Get(int id) { ... }

    // POST api/books
    [Route("")]
    public HttpResponseMessage Post(Book book) { ... }
}
```

RoutePrefix: permet d'avoir un préfixe de route commun à toutes les méthodes définies sous le Controller

Le Routage dans MVC Web API

Le routage par attribut- Les Avantages

- Identifier rapidement la route qui mène au contrôleur, et dans des scénarios de débogage
- Plus de contrôle sur les URIs
- Créer des versions (« *versionning* »)
  - /api/v1/products
  - /api/v2/products

- Sur-définition des segments URI

/orders/1	Numéro d'ordre
/orders/pending	Mappage sur une collection

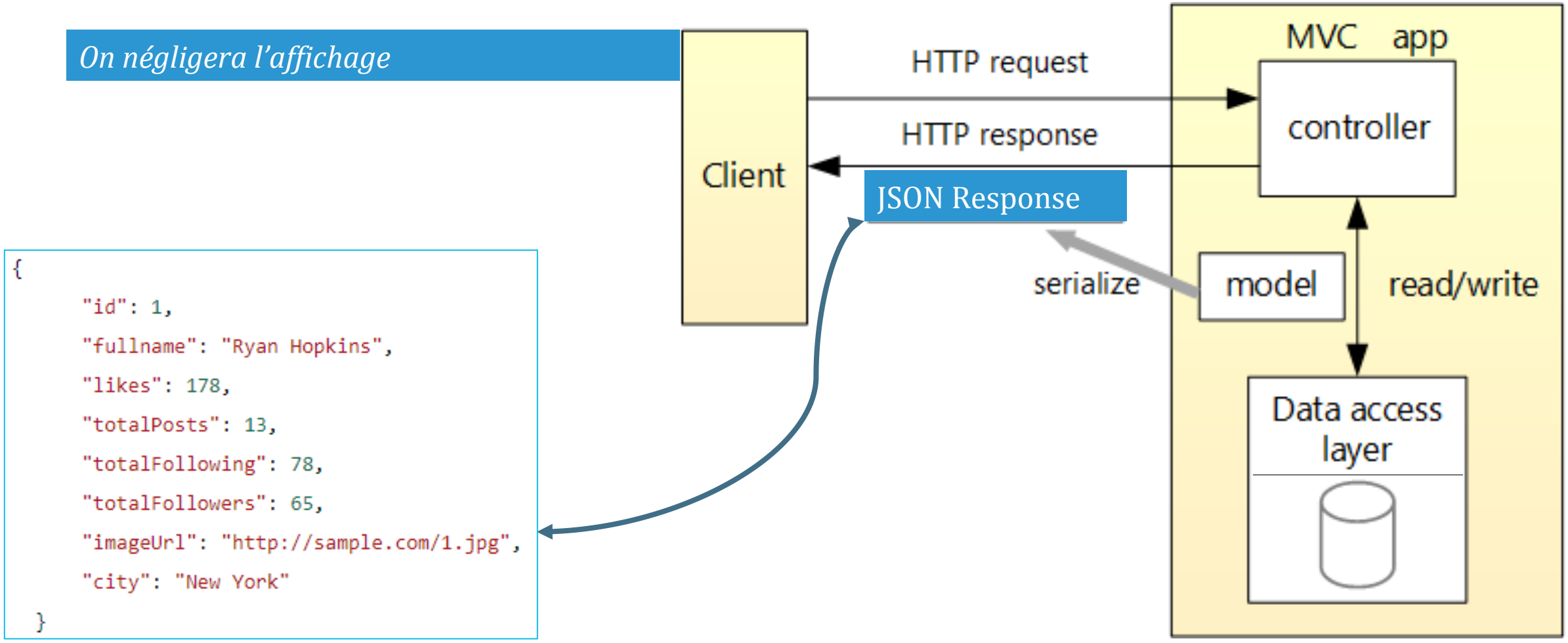
- Paramètres multiples
  - /orders/1
  - /orders/2013/06/16

Étude de Cas 1 (REST)

Construction d'un MVC Web API sous ASP.NET avec Visual Studio permettant de retourner un profil utilisateur à partir de son identifiant, suite à une requête (HTTP) initiée à partir d'une application cliente

Étude de Cas (REST): Introduction

On négligera l'affichage

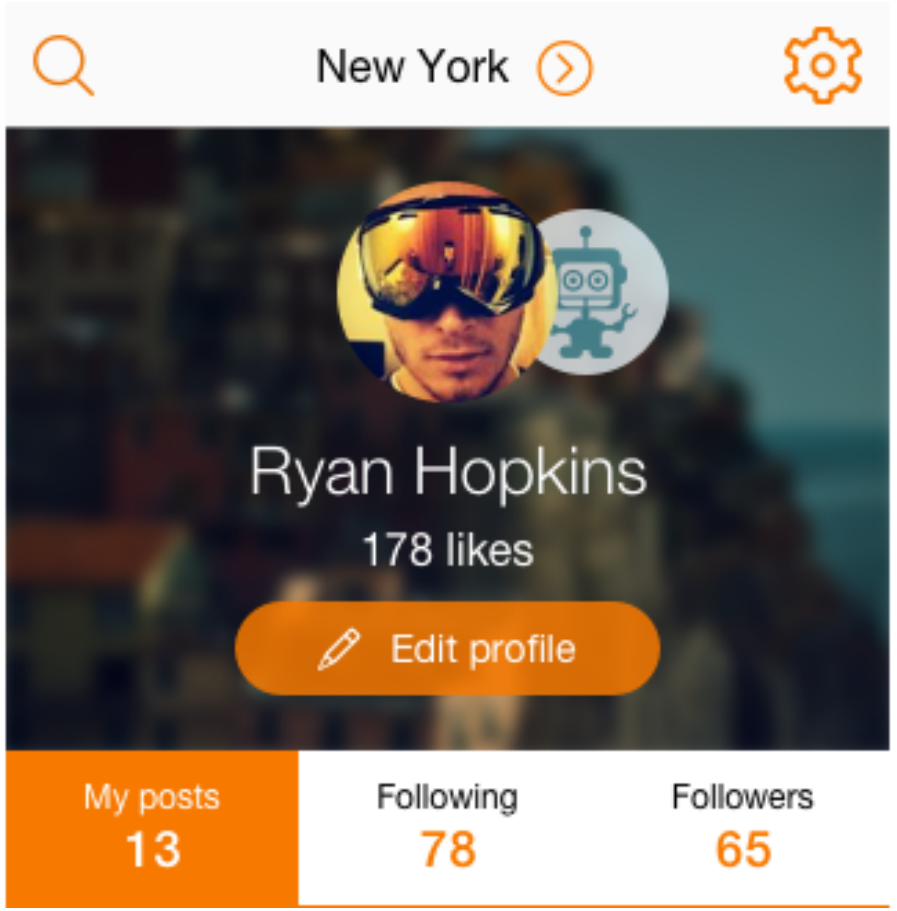



```
{
      "id": 1,
      "fullname": "Ryan Hopkins",
      "likes": 178,
      "totalPosts": 13,
      "totalFollowing": 78,
      "totalFollowers": 65,
      "imageUrl": "http://sample.com/1.jpg",
      "city": "New York"
  }
```

Étude de Cas (REST)

Pour fournir les données affichées dans la figure ci-contre, on a choisi le format JSON:

```
{
      "id": 1,
      "fullname": "Ryan Hopkins",
      "likes": 178,
      "totalPosts": 13,
      "totalFollowing": 78,
      "totalFollowers": 65,
      "imageUrl": "http://sample.com/1.jpg",
      "city": "New York"
  }
```



Méthode	API	Description	Corps de la requête	Réponse
GET	/api/User/{id}/Profile	Get a user profile by ID	Vide	Objet Profile

Étude de Cas (REST)

Étapes:

1. **Créer un projet “ExempleApi”**
   - File → New Project → Templates; Visual C#; Web → ASP.NET Web Application
   - Choisir “Empty” comme “template”
2. **Installer le module WebApi en utilisant ‘Nuget’**
   - Tools → Library Package Manager → Package Manager Console
   - Commande: Install-Package Microsoft.AspNet.WebApi
3. **Create Folders:**
   - Models
   - Controllers
   - DataAccessLayer

Étude de Cas (REST)

Étapes:

4. Configurer les routages

- Ajouter la classe WebApiConfig.cs: right click sur App_Start → Add → New Item → Class
- Ajouter le code comme suit:

```
1 reference
public class WebApiConfig
{
    1 reference
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();
    }
}
```

Étude de Cas (REST)

Étapes:

4. Configurer les routages

- Ajouter la classe Global.asax: right click sur le Projet → Add → New Item → Global Application Class
- Ajouter le code comme suit:

```
public class Global : System.Web.HttpApplication
{
    0 references
    protected void Application_Start(object sender, EventArgs e)
    {
        GlobalConfiguration.Configure(WebApiConfig.Register);
    }
}
```

Étude de Cas (REST)

Étapes:

5. Ajouter un modèle:

- Right click sur Models → Add Class
- On spécifie que les données retournées par ce Web API sont sérialisées en utilisant les annotations:
  - [Serializable]
  - [DataContract]
  - [DataMember]
  - [XmlAttribute]
  - [XmlElement]

```
[XmlRoot]
0 references
public class Users
{
    [XmlElement]
    0 references
    public List<User> User { get; set; }
}
```

```
[Serializable]
[DataContract]
public class User
{
        [DataMember]
        [XmlAttribute]
        public int Id { get; set; }
        [DataMember]
        [XmlElement]
        public string Fullname { get; set; }
        [DataMember]
        [XmlElement]
        public int Likes { get; set; }
        [DataMember]
        [XmlElement]
        public int TotalPosts { get; set; }
        [DataMember]
        [XmlElement]
        public int TotalFollowing { get; set; }
        [DataMember]
        public int TotalFollowers { get; set; }
        [DataMember]
        [XmlElement]
        public string ImageURL { get; set; }
        [DataMember]
        [XmlElement]
        public string City { get; set; }
}
```

Étude de Cas (REST)

6. Ajouter les classes gérant l'accès à la base de données dans "DataAccessLayer": Right click sur DataAccessLayer → Add Class: "IUserRepository", "UserRepository"

```
public interface IUserRepository
{
    2 references
    User GetUser(int id);
}
```

```
public class UserRepository : IUserRepository
{
    2 references
    public User GetUser(int id)
    {
        String path = System.Web.HttpContext.Current.Request.MapPath("~\\Data\\DB.xml");
        XElement xElement = XElement.Load(path);

        User user = (from u in xElement.Elements("user")
                     where (string)u.Attribute("id") == id.ToString()
                     select new User
                     {
                         Id = id,
                         Fullname = u.Element("fullname").Value.ToString(),
                         City = u.Element("city").Value.ToString(),
                         ImageURL = u.Element("imageUrl").Value.ToString(),
                         Likes = Convert.ToInt32(u.Element("likes").Value),
                         TotalFollowers = Convert.ToInt32(u.Element("totalFollowers").Value),
                         TotalFollowing = Convert.ToInt32(u.Element("totalFollowing").Value.ToString()),
                         TotalPosts = Convert.ToInt32(u.Element("totalPosts").Value.ToString())
                     }).FirstOrDefault();

        return user;
    }
}
```

Étude de Cas (REST)

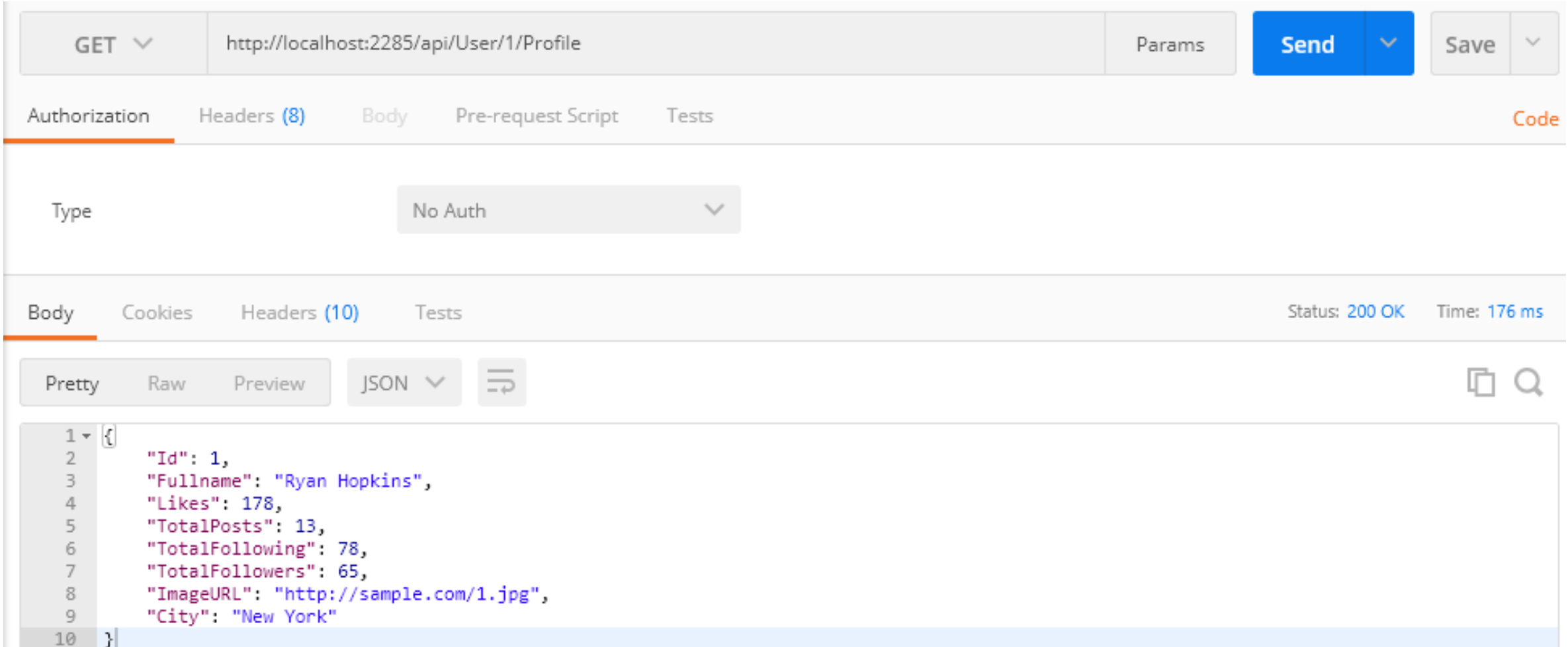
7. **Ajouter un Controller dans "Controllers":** Right click sur Controllers → Add Controller: "UserController"

```
[RoutePrefix("api/User")]
0 references
public class UserController : ApiController
{
    [Route("{id}/Profile")]
    [HttpGet]
    0 references
    public HttpResponseMessage Profile(int id)
    {
        IUserRepository UserRep = new UserRepository();
        return Request.CreateResponse(HttpStatusCode.OK, UserRep.GetUser(id));
    }
}
```

Étude de Cas (REST)

Étapes:

8. Exécuter le project
9. Tester avec “Postman”



Étude de Cas (REST)

10. Tester avec une simple requête HTTP à partir d'une application cliente qui invoque le service web créé:

```
static void GetUserProfile(int id)
{
    string url = "http://localhost:2285/api/User/" + id + "/Profile";
    HttpWebRequest httpRequest = null;
    HttpWebResponse httpResponse = null;
    try
    {
        httpRequest = (HttpWebRequest)HttpWebRequest.Create(url);
        httpRequest.Method = "GET";
        httpRequest.ContentType = "application/json";
        httpResponse = (HttpWebResponse)httpRequest.GetResponse();

        using (Stream stream = httpResponse.GetResponseStream())
        {
            StreamReader reader = new StreamReader(stream);
            User jsonDeserialized = JsonConvert.DeserializeObject<User>(reader.ReadToEnd());
            System.Diagnostics.Debug.WriteLine("{0}\t{1}", jsonDeserialized.fullname, jsonDeserialized.city);
        }
    }
    catch (Exception ex)
    {
        //handle exception
    }
    finally
    {
        httpRequest = null;
        httpResponse = null;
    }
}
```

Questions?