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 4

Les Architectures Distribuées

Plan Du Chapitre

- Introduction
- Les architectures 2-tiers
- Les architectures n-tiers
- Étude de Cas
- Préparation

Introduction

Introduction

- Dans une entreprise on a souvent besoin de produire des projets informatiques
  - **Développement en interne** (‘*in-house development*’)
  - **Développement externalisé** (‘*outsourcing*’)

- **Analyse des besoins (Requirements Analysis):** Rassembler toutes les informations nécessaires à la compréhension du problème

- **Spécification des besoins (Requirements specifications):** Spécifier de manière claire et précise le résultat de l’analyse en utilisant des méthodes de modélisation

Introduction

- **Les besoins**
  - **Fonctionnels**: ce que le client attend en terme de fonctionnalités.
  - **Non fonctionnels**: conformité aux standards, contraintes sous lesquelles le système doit rester opérationnel.
- **Les besoins doivent être analysés afin de munir la conception du système.**
- **A l'issue de la conception, on aura une architecture logicielle à mettre en œuvre.**

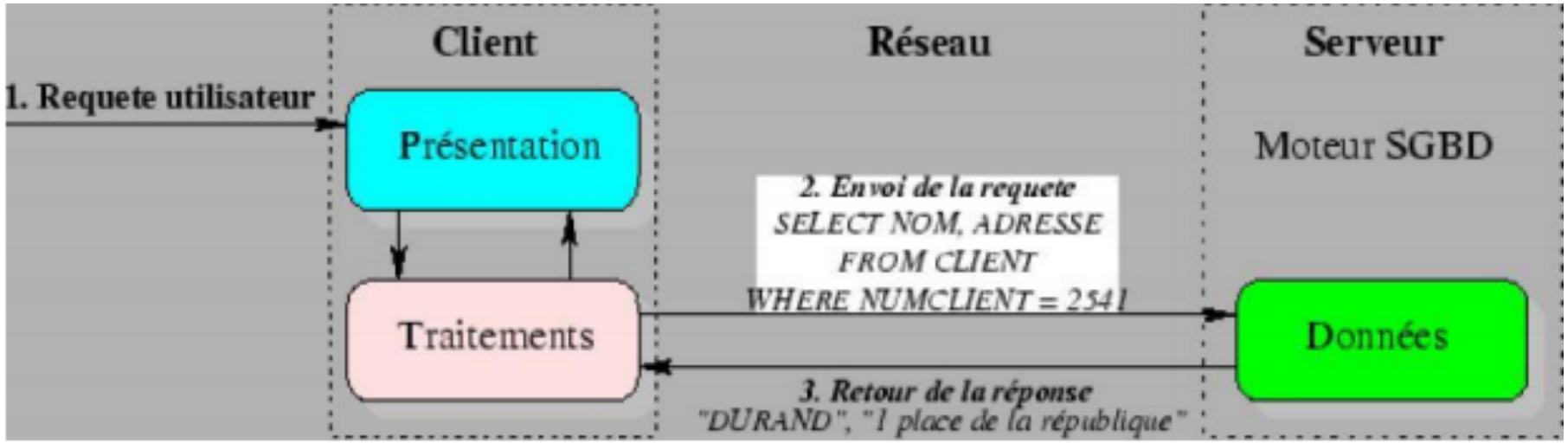
- L'**architecture logicielle** décrit les différents éléments d'un ou de plusieurs systèmes informatiques, leurs interrelations et leurs interactions.
- L'analyse décrit le « quoi faire »
- L'architecture décrit le « **comment le faire** »!

Les Architectures 2-tiers

Les Architectures 2-tiers

- Dans une architecture 2-tiers, (« **client-serveur** »), le poste client se contente de déléguer la gestion des données à un système spécialisé.

- Un SGBD centralisé sur un serveur dédié interrogé en utilisant un langage de requête (SQL).



Les Architectures 2-tiers

Avantages	Inconvénients
Gestion centralisée des données: • Le serveur est au centre du réseau • Les ressources (BDD) sont communes à tous les utilisateurs • Garantit la cohérence des données, la non-redondance et la non-contradiction	**1 seul serveur effectue tous les traitements** • Charge élevée au niveau du serveur: dégradation de performance avec un nombre élevé de clients
Exploiter la puissance des ordinateurs déployés en réseau	La charge sur le poste client est élevée de point de vue utilisation du réseau
	Les coûts de la maintenance du code et de l'extension des fonctionnalités sont élevés
	Le code n'est pas très réutilisable
	La sécurité

Les Architectures n-tiers

Les Architectures n-tiers

- Pour résoudre les limitations de l'architecture 2-tiers, tout en conservant ses avantages on a recours aux '**architectures distribuées**'
  - Répartition des traitements
  - Exemple: Architecture 3-tiers

- **Couche vs. Tiers**
  - Une **couche** décrit le regroupement **logique** des fonctionnalités et des composants
  - Un **tiers** décrit la distribution **physique** des fonctionnalités et des composants à travers les serveurs, ordinateurs, réseaux, etc.
  - ***Plusieurs couches peuvent résider sur un même tiers.***

Les Architectures n-tiers

	2 tiers	3 et n tiers
Administration du système	**Complexe**	**Moins complexe**
Sécurité	**Faible** (sécurité au niveau des données)	**Elevée** (raffinée au niveau des services ou des méthodes)
Encapsulation des données	**Faible** (les tables de données sont directement accessibles)	**Elevée** (le client fait appel à des services ou méthodes)
Performance	**Faible**	**Bonne**
Extensibilité	**Faible** (gestion limitée des liens réseaux avec le client)	**Excellente** (possibilité de répartir dynamiquement la charge sur plusieurs serveurs)
Réutilisation	**Faible**	**Excellente**
Facilité de développement	**Elevée**	**En progression**
Sources de données hétérogènes	**Non**	**Oui** (utiliser plusieurs bases de données dans la même transaction)
Flexibilité d'architecture matérielle	**Limitée**	**Excellente** (résider les couches 2 et 3 sur une ou plusieurs machines)
Relève en cas de pannes	**Faible**	**Excellente**

Les Architectures n-tiers

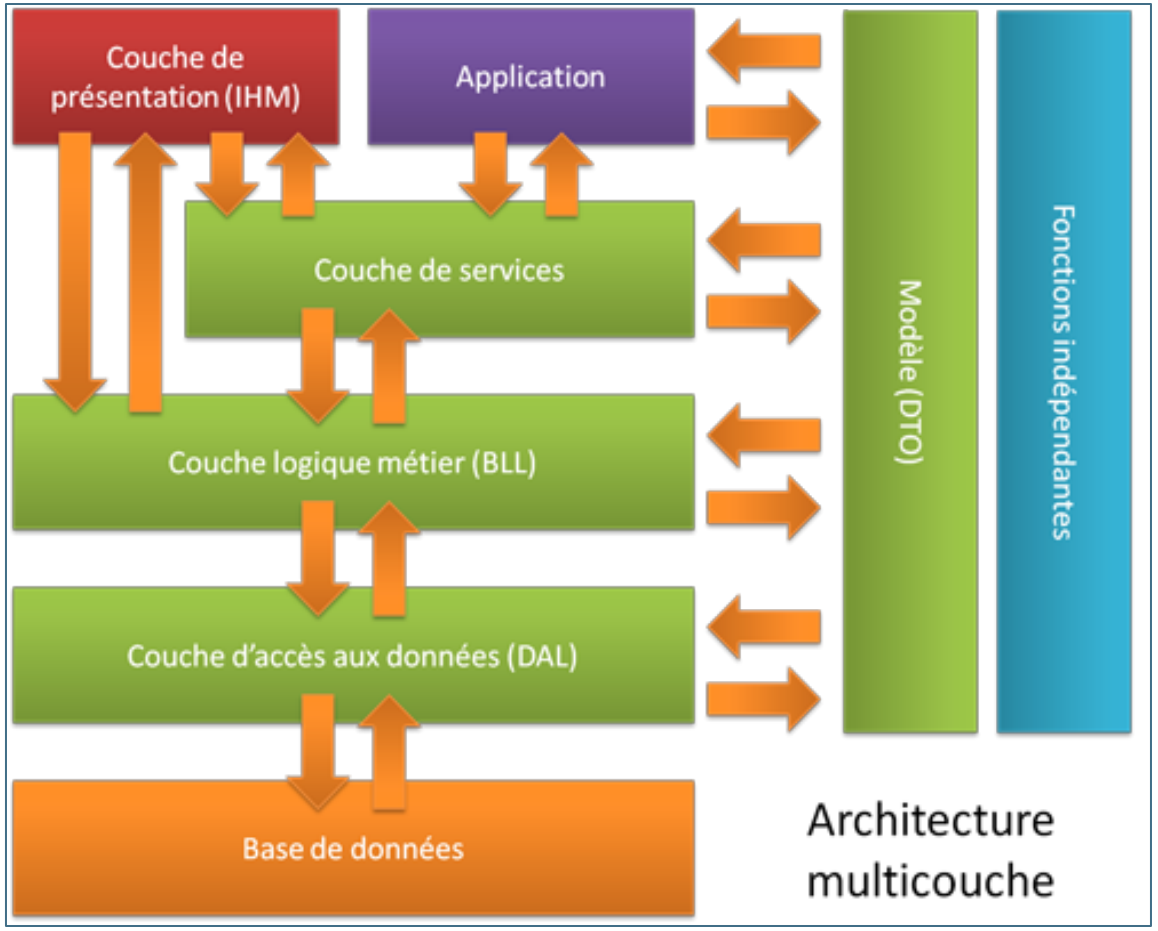
Une application peut être divisée en 3 couches

- **«Presentation Layer» (PL)**
  - couche de présentation (IHM, GUI)
  - Responsable de l'affichage sur l'interface utilisateur
  - traitement locaux: contrôles de saisies, mises en forme des données

- **« Business Logic » ou « Business Layer » (BL)**
  - Couche 'métier'
  - traitements applicatifs globaux
  - partie fonctionnelle de l'application
  - opère sur les données en fonction des requêtes utilisateurs, effectuées à travers la PL

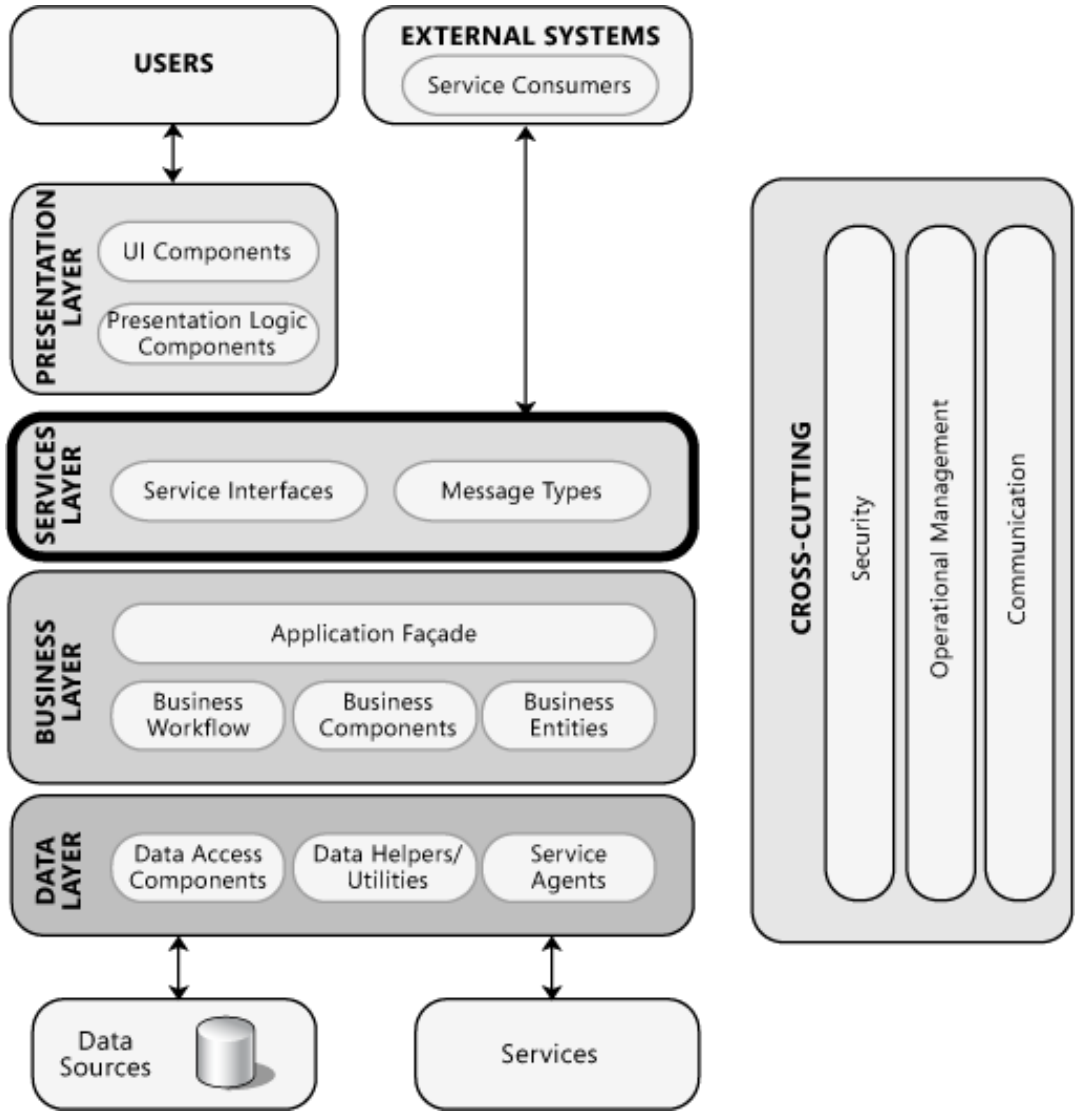
- **« Data Layer » ou « Data Access Layer » (DAL)**
  - La partie gérant l'accès aux données du système (à la BDD – peut importe son type)

Les Architectures n-tiers – Aller plus loin



- Chaque couche regroupe les composants (objets, méthodes) partageant les mêmes fonctionnalités, les mêmes rôles.
  - Ainsi, on ne devrait avoir aucune requête SQL dans une page Web
- Chaque couche ne peut communiquer qu'avec celle qui lui est immédiatement voisine
- les couches communiquent entre elles en se transmettant des objets de transfert de données: les **DTO** (‘*Data Transfer Objects*’)

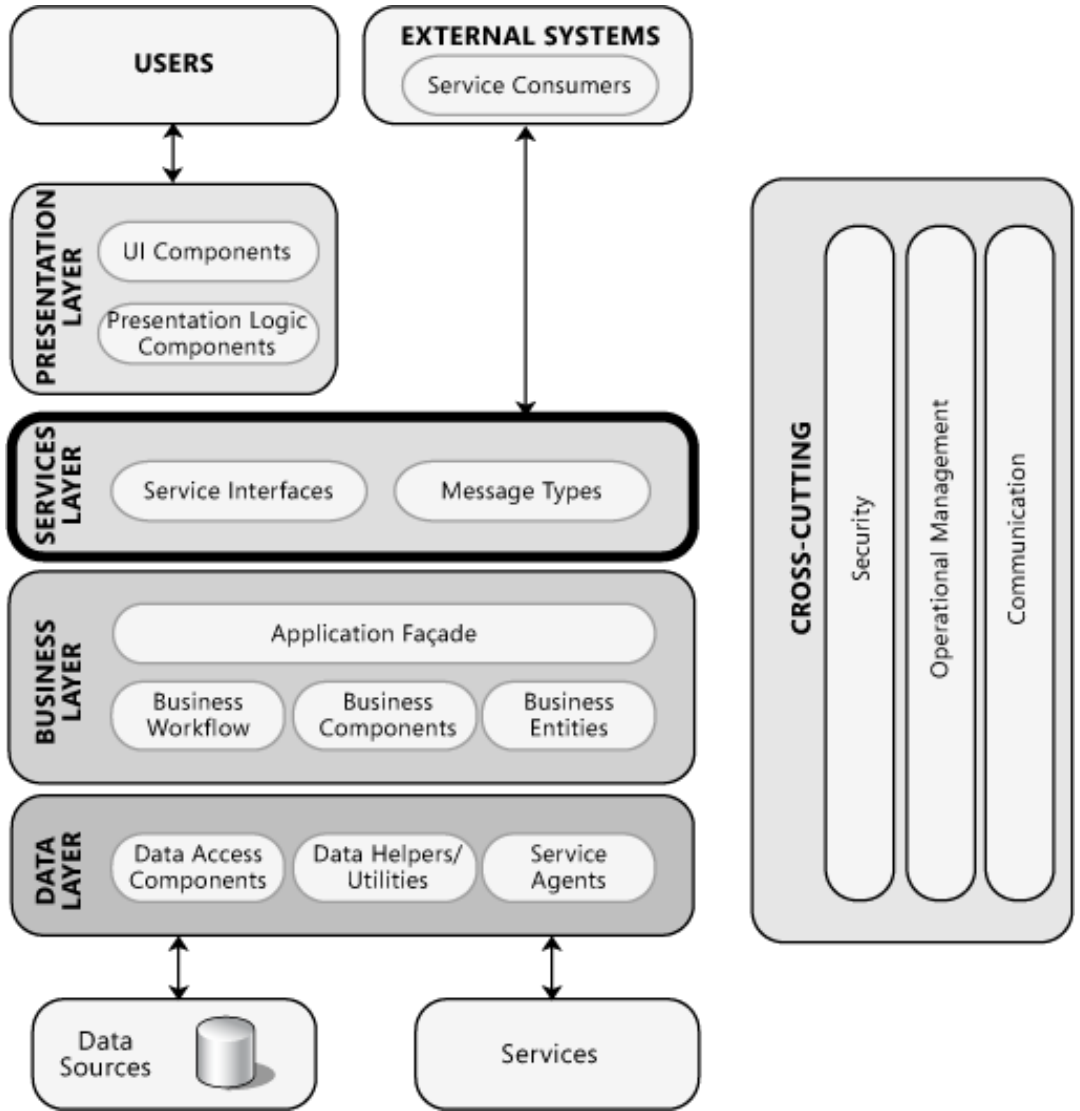
Les Architectures n-tiers – Aller plus loin



- Une couche de services (‘services layer’) peut également être utilisée pour exposer la logique métier aux systèmes externes
  - On gagne ainsi un support à un plus large éventail de types de clients

- Dans certains cas, la PL peut communiquer avec le BL via SL, spécialement lorsque PL et BL ne résident pas sur le même tiers.

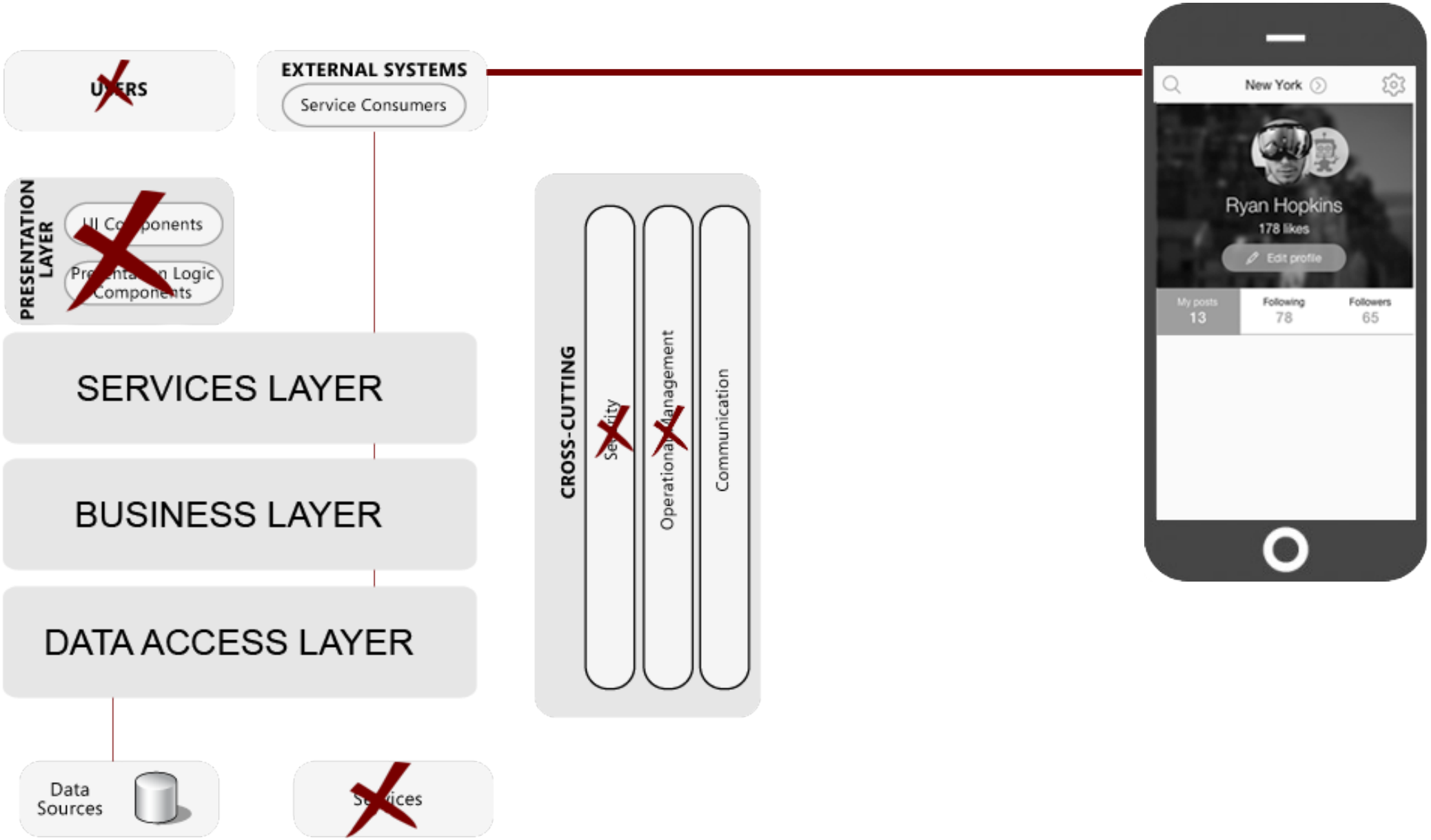
Les Architectures n-tiers – Aller plus loin

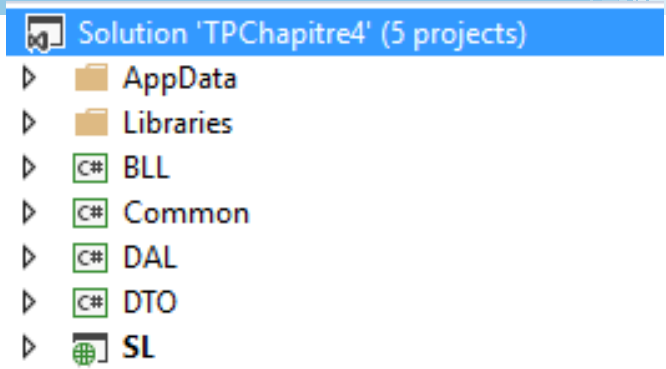


- **Les fonctionnalités communes ou transversales (‘*cross-cutting concerns*’)**
  - Des fonctionnalités **centralisées communes aux différentes couches**
  - **Entre autres:**
    - Communication (messages)
    - Gestion d’erreurs
    - Authentification
    - Caching
    - Logging
    - Validation

Étude de Cas

Étude de Cas


EXTERNAL SYSTEMS
Service Consumers
PRESENTATION LAYER
SERVICES LAYER
BUSINESS LAYER
DATA ACCESS LAYER
CROSS-CUTTING
Communication
Data Sources
New York
Ryan Hopkins
178 likes
Edit profile
My posts
13
Following
78
Followers
65



Étude de Cas

- Il faut créer les projets suivants pour construire l'architecture en question:

Nom du projet	Fonctionnalité	Type
SL	Services Layer	Web Project
BLL	Business Layer	Class Library
DAL	Data Access Layer	
DTO	Data Transfer Objects	
Common	Cross-cutting concerns (dans notre cas, pour les codes et messages d'état)	

Étude de Cas

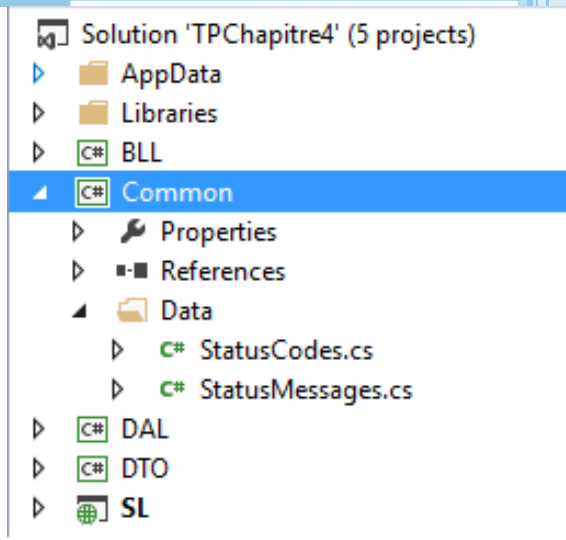
- Common

Codes

```
public static class StatusCodes
{
    public const int Success = 0;
    public const int InexistantUser = 1;
    public const int Error = 2;
}
```

Messages

```
public static class StatusMessages
{
    public const string Success = "Successful";
    public const string InexistantUser = "Inexistant User";
    public const string Error = "Server Error";
}
```



Étude de Cas

- DTO

```
[Serializable]
[DataContract]
7 references
public class GetProfileRequestDTO
{
    [DataMember]
    3 references
    public int UserID { get; set; }
}
```

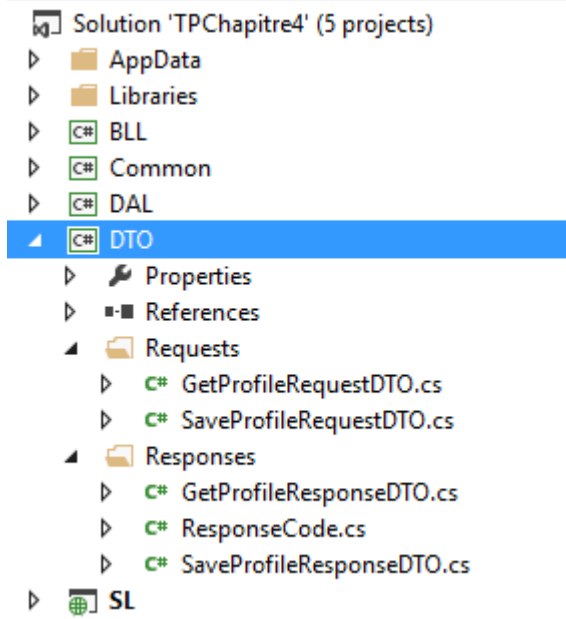
DTO de la requête

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

Classe Users

```
[Serializable]
    [DataContract]
    public class GetProfileResponseDTO
    {
        [DataMember]
        [XmlAttribute]
        public int Id { get; set; }
        [DataMember]
        public ResponseCode Status { 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]
        [XmlElement]
        public int TotalFollowers { get; set; }
        [DataMember]
        [XmlElement]
        public string ImageURL { get; set; }
        [DataMember]
        [XmlElement]
        public string City { get; set; }
    }
```

DTO de la réponse




```
public class ResponseCode
{
    5 references
    public int Code { get; set; }
    5 references
    public string Message { get; set; }
}
```

Classe ResponseCode

Étude de Cas

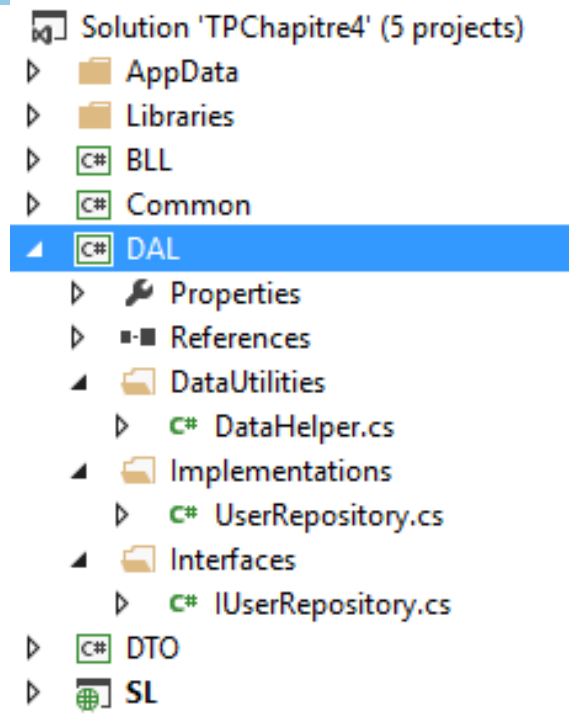
- DAL

IUserRepository

```
public interface IUserRepository
{
    2 references
    GetProfileResponseDTO FetchProfile(GetProfileRequestDTO objLoginRequest);
    2 references
    SaveProfileResponseDTO SaveProfile(SaveProfileRequestDTO objProfileRequest);
}
```

UserRepository

```
public class UserRepository: IUserRepository
{
    DataHelper dh = new DataHelper();
    2 references
    public GetProfileResponseDTO FetchProfile(GetProfileRequestDTO objProfileRequest)
    {
        GetProfileResponseDTO objProfileResponse = dh.GetProfileQuery(objProfileRequest);
        return objProfileResponse;
    }
    2 references
    public SaveProfileResponseDTO SaveProfile(SaveProfileRequestDTO objProfileRequest)
    {
        SaveProfileResponseDTO objProfileResponse = dh.SaveProfileQuery(objProfileRequest);
        return objProfileResponse;
    }
}
```



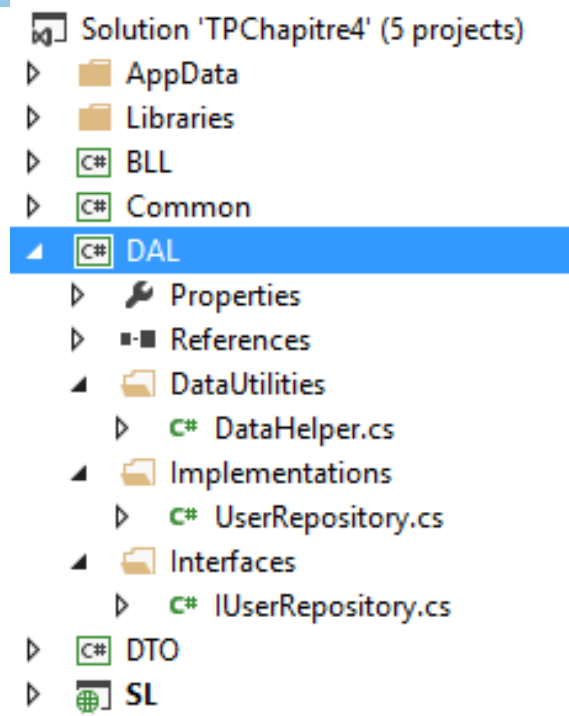
Étude de Cas

- DAL

DataHelper

```
1 reference
public GetProfileResponseDTO GetProfileQuery(GetProfileRequestDTO objProfileRequest)
{
    GetProfileResponseDTO objGetProfileResponse;
    try
    {
        XElement xElement = XElement.Load(Path);

        objGetProfileResponse = (from u in xElement.Elements("user")
                where (string)u.Attribute("id") == objProfileRequest.UserID.ToString()
                select new GetProfileResponseDTO
                 {
                     Id = objProfileRequest.UserID,
                     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();
```

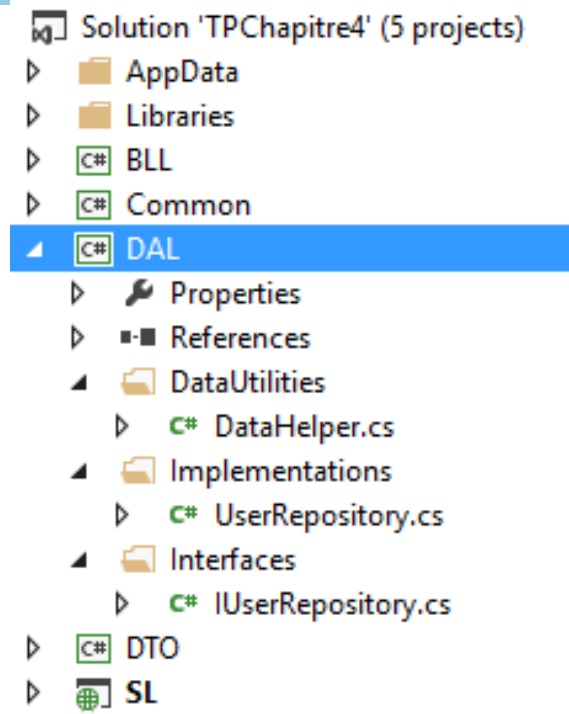


Étude de Cas

- DAL

DataHelper

```
        if (objGetProfileResponse != null)
        {
            objGetProfileResponse.Status = new ResponseCode()
            {
                Code = StatusCodes.Success,
                Message = StatusMessages.Success
            };
        }
        else
        {
            objGetProfileResponse = new GetProfileResponseDTO()
            {
                Status = new ResponseCode()
                {
                    Code = StatusCodes.InexistantUser,
                    Message = StatusMessages.InexistantUser
                }
            };
        }
    }
    catch (Exception ex)
    {
        objGetProfileResponse = new GetProfileResponseDTO()
        {
            Status = new ResponseCode()
            {
                Code = StatusCodes.Error,
                Message = StatusMessages.Error
            }
        };
    }
    return objGetProfileResponse;
}
```



Étude de Cas

- BLL

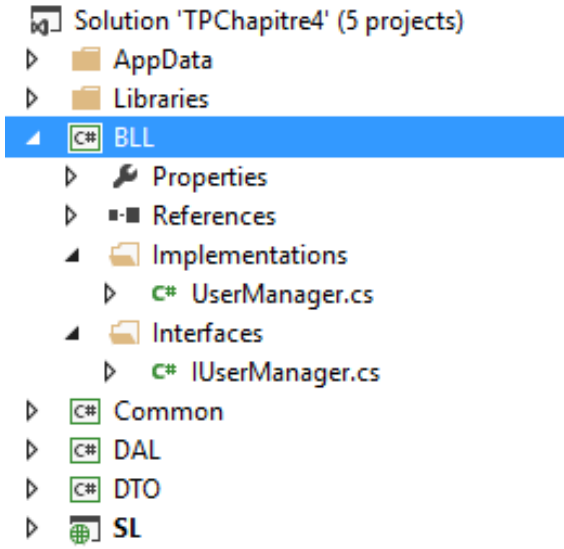
IUserManager

```
public interface IUserManager
{
    2 references
    GetProfileResponseDTO FetchProfile(GetProfileRequestDTO objLoginRequest);
    2 references
    SaveProfileResponseDTO SaveProfile(SaveProfileRequestDTO obj);
}
```

UserManager

```
public class UserManager : IUserManager
{
    IUserRepository userRepository;

    2 references
    public GetProfileResponseDTO FetchProfile(GetProfileRequestDTO objProfileRequest)
    {
        userRepository = new UserRepository();
        GetProfileResponseDTO objFetchProfileResponse = new GetProfileResponseDTO();
        objFetchProfileResponse = this.userRepository.FetchProfile(objProfileRequest);
        return objFetchProfileResponse;
    }
```

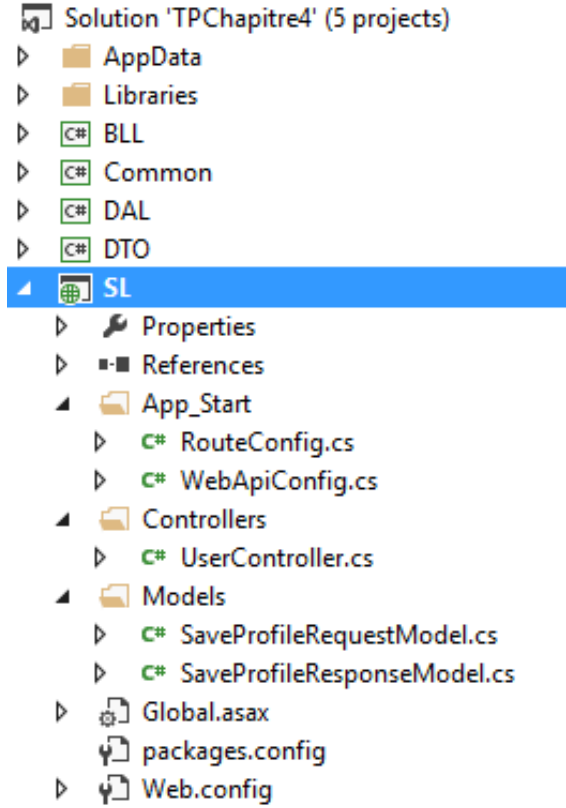


Étude de Cas

- SL

WebApiConfig

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



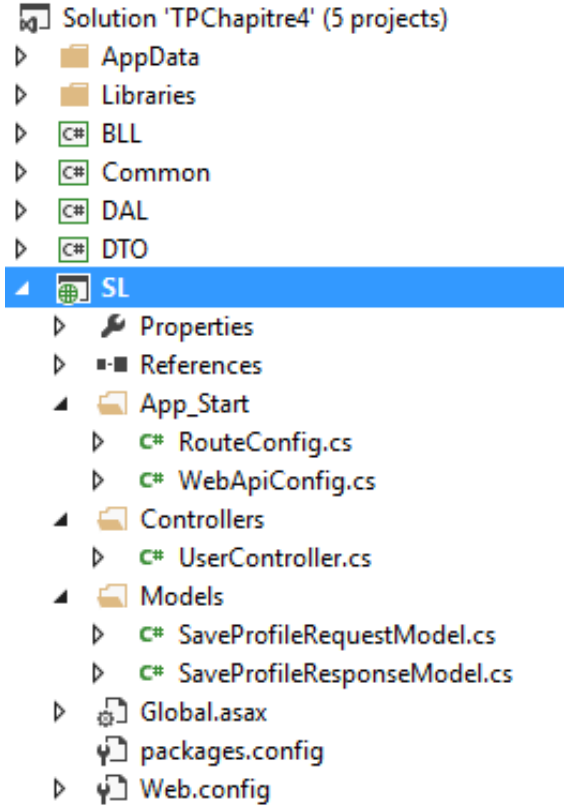
Étude de Cas

- SL

UserController

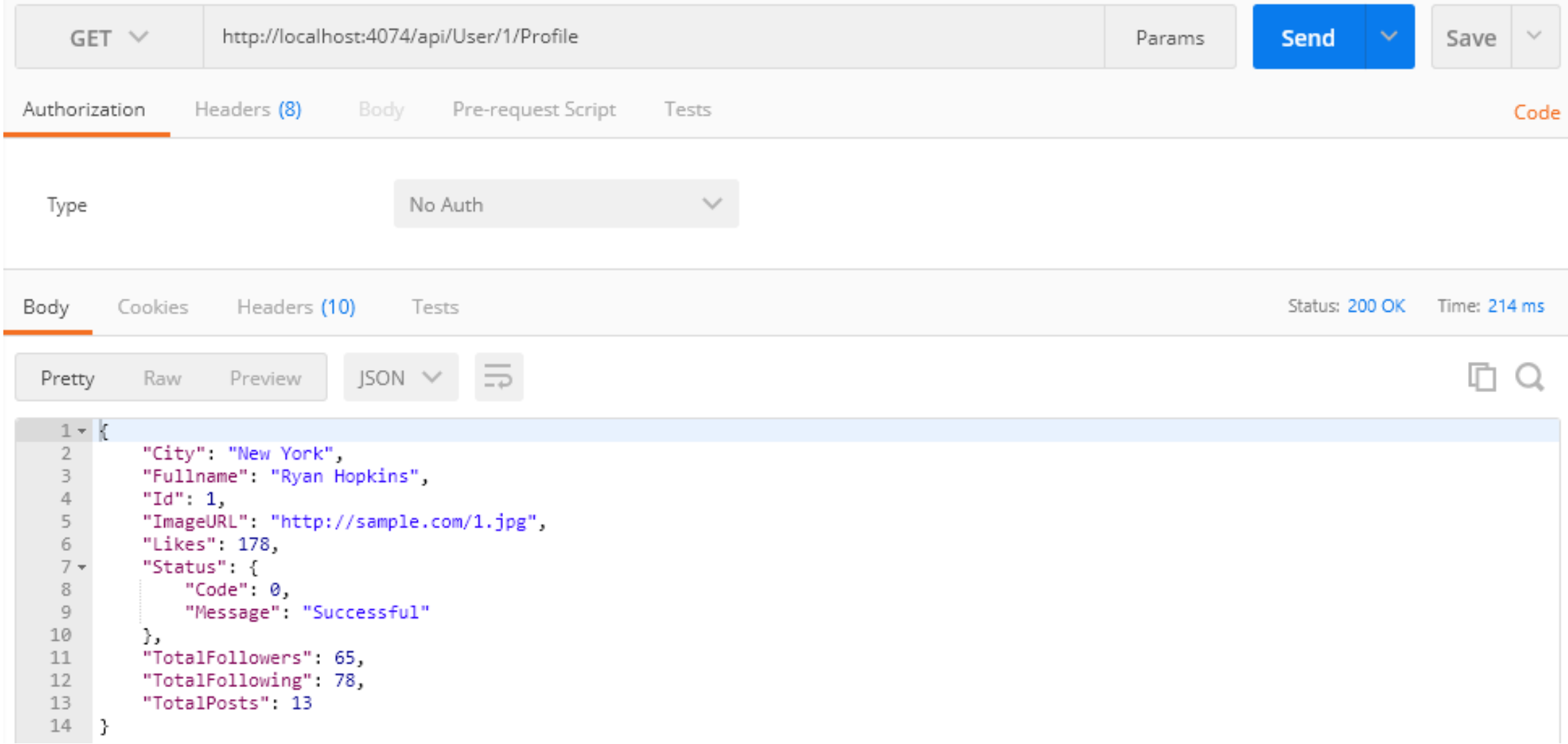
```
[RoutePrefix("api/User")]
0 references
public class UserController : ApiController
{
    IUserManager userManager;

    [Route("{id}/Profile")]
    [HttpGet]
    0 references
    public HttpResponseMessage Profile(int id)
    {
        userManager = new UserManager();
        GetProfileRequestDTO objProfileRequest = new GetProfileRequestDTO()
        {
            UserID = id
        };
        GetProfileResponseDTO objProfileResponse = userManager.FetchProfile(objProfileRequest);
        return Request.CreateResponse(HttpStatusCode.InternalServerError, objProfileResponse);
    }
```



Étude de Cas

- Postman



Préparation

Préparation

- Effectuer une opération de type [HttpPost] permettant de mettre à jour le profil d'un utilisateur.

- L'URL de la requête doit être:
  - *BaseURL/api/User/Profile*

- Le données POST doivent être suivant la structure suivante:

```
{
    "UserID" : 2,
    "FullName" : "John Smith",
    "City" : "Paris"
}
```

N.B.: Les données échangées entre le client et la SL doivent être des classes de type Model

Questions?