Pour ce TP, nous allons construire un pseudo catalogue de produits.

La mise en œuvre permettra de couvrir la création de service WCF, la consommation de WebService en .NET et également la création de services REST.

Exercice 1 : Création d'un Service WCF

1. Créer un projet de type « Application de service WCF » nommé DotNet.Wcf.TP dans Visual Studio.

2. Ajouter au projet une classe Product dans un dossier DataContracts contenant les **propriétés** suivantes :
   - Id : int
   - Label : string
   - Description : string
   - Price : double

3. Ajouter à cette classe les éléments nécessaires pour quelle soie visible dans le contrat de service.

4. Ajouter au projet un élément de type « WCF Service » nommé ProductServices

5. Dans le contrat de service (IProductServices), déclarer les méthodes suivantes :

Methode	Retour	Description
GetAllProducts()	List<Product>	Renvoie la liste de tous les produits
GetProductById(int id)	Product	Renvoie le produit en fonction de l'id en paramètre.
SaveProduct(Product product)	Int	Sauvegarde le produit. Dans le cas où l'id est égal à 0, le service ajoute un nouveau produit sinon le produit est mis à jour. Le retour de la méthode est l'id du produit ajouté ou mis à jour

6. Dans la classe d'implémentation du service (ProductServices.svc.cs), ajouter le code suivant :

```
        private static List<Product> products = new List<Product>()
        {
            new Product() { Id = 1, Label = "Produit1", Description = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit.", Price = 49.99d },
            new Product() { Id = 2, Label = "Produit2", Description = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit.", Price = 24.90d },
            new Product() { Id = 3, Label = "Produit3", Description = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit.", Price = 100d },
            new Product() { Id = 4, Label = "Produit4", Description = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit.", Price = 67.53d },
            new Product() { Id = 5, Label = "Produit5", Description = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit.", Price = 42d }
        };
```

Cette liste fera office de base de données pour notre exemple.

7. Implémenter la méthode GetAllProducts puis lancer le debug puis compiler (ctrl+shift+B)

8. Lancer l'application en debug avec F5 (vous pouvez également lancer l'application sans debug avec Ctrl+F5).
9. L'application WCF Test Client se lance. Tester la méthode GetAllProducts dans cette application
10. Copier l'adresse du service puis ouvrez l'url dans un navigateur

11. Copier l'adresse du WSDL puis tester votre service dans SoapUI.

12. Implémenter les 2 autres méthodes du service puis les tester.

Exercice 2 : Création d'un client consommant un webservice

1. Dans SoapUI, créer un Mock_Service depuis le projet créé dans l'exercice précédent, implémenter une réponse puis le démarrer.

2. Dans une nouvelle instance de Visual Studio, créer un nouveau Projet de type « Application Console »

3. Ajouter la référence de service au service SoapUI

4. Dans la méthode Main de la classe Program, implémenter l'appel au client et afficher l'ensemble des informations produit dans la console.

Tips :

- Pour afficher un élément dans la console, on utilise `Console.WriteLine`
- Utiliser $ avant l'ouverture des guillemets d'une string permet d'afficher une variable entre {}
  Ex : `$"maVar: {maVar}"`
- Utiliser `Console.ReadKey();` pour éviter que l'application ne se ferme à la fin de l'exécution

Exercice 3 : Création d'un service ASP.NET Core Web API

1. Dans une instance Visual Studio, créer un projet de type « ASP.NET Core Web Application » nommé DotNet.WebApi

2. Sélectionner le template API puis cliquer sur OK

3. Au niveau de la solution, ajouter un projet de type « Bibliothèque de classe (.NET Standard)» nommé DotNet.WebApi.Service puis supprimer Class1.cs

4. Dans le projet DotNet.WebApi.Service, ajouter une classe nommée ProductManager

5. Ajouter dans le fichier le code en Annexe1

6. Dans le projet DotNet.WebApi, référencer le projet DotNet.WebApi.Service dans les dépendances.

7. Faire un clique-droit sur le projet DotNet.WebApi puis sélectionner « Gérer les packages Nuget »

8. Dans la partie Browse, chercher et installer AutoMapper

9. Créer le dossier Models puis dans ce dernier, créer la classe ProductModel implémentant les mêmes propriétés que Product. (copier/coller)

10. Dans le fichier Startup.cs, dans la méthode Configure, ajouter le code suivant :

```
AutoMapper.Mapper.Initialize(cfg =>
{
    cfg.CreateMap<Business.Product, Models.ProductModel>();
    cfg.CreateMap<Models.ProductModel, Business.Product>();
});
```

11. Dans le dossier Controller, ajouter un contrôleur API Controller (avec read/write actions) nommé ProductController

12. Implémenter le contenu de la méthode Get puis lancer l'application

Tips :

On passe de Product à ProductModel via Automapper avec le code suivant :

```
AutoMapper.Mapper.Map<ProductModel>(entity);
```

Remarque : Automapper gère également les type List<T>

13. Tester l'application avec SoapUI.

14. Implémenter le reste des méthodes et les tester

Exercice 4 : Création d'un client

Créer un client consommant l'API REST avec la technologie de votre choix

Annexes

Annexe1

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace DotNet.WebApi.Business
{
    public class Product
    {
        public int Id { get; set; }

        public string Label { get; set; }

        public string Description { get; set; }

        public double Price { get; set; }

    }

    public static class ProductManager
    {
        private static List<Product> products = new List<Product>()
        {
            new Product() { Id = 1, Label = "Produit1", Description = "Lorem ipsum dolor sit amet, consectetur
adipiscing elit.", Price = 49.99d },
            new Product() { Id = 2, Label = "Produit2", Description = "Lorem ipsum dolor sit amet, consectetur
adipiscing elit.", Price = 24.90d },
            new Product() { Id = 3, Label = "Produit3", Description = "Lorem ipsum dolor sit amet, consectetur
adipiscing elit.", Price = 100d },
            new Product() { Id = 4, Label = "Produit4", Description = "Lorem ipsum dolor sit amet, consectetur
adipiscing elit.", Price = 67.53d },
            new Product() { Id = 5, Label = "Produit5", Description = "Lorem ipsum dolor sit amet, consectetur
adipiscing elit.", Price = 42d }
        };

        public static List<Product> GetAll()
        {
            return products;
        }

        public static Product GetById(int id)
        {
            return products.FirstOrDefault(x => x.Id == id);
        }

        public static void Add(Product product)
        {
            int id = products.Max(x => x.Id) + 1;
            product.Id = id;
            products.Add(product);
        }

        public static void Update(Product product)
        {
            Product entity = products.FirstOrDefault(x => x.Id == product.Id);

            if (entity == null)
                throw new ArgumentOutOfRangeException("L'id du produit à mettre à jour n'existe pas");

            products.Remove(entity);
            products.Add(product);

        }

        public static void Delete(int id)
        {
            Product entity = products.FirstOrDefault(x => x.Id == id);

            if (entity == null)
                throw new ArgumentOutOfRangeException("L'id du produit à mettre à jour n'existe pas");

            products.Remove(entity);
        }
    }
}
```