Cours .Net
Windows Communication Foundation

Lemettre Arnaud
Arnaud.lemettre@gmail.com

EPITA
ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

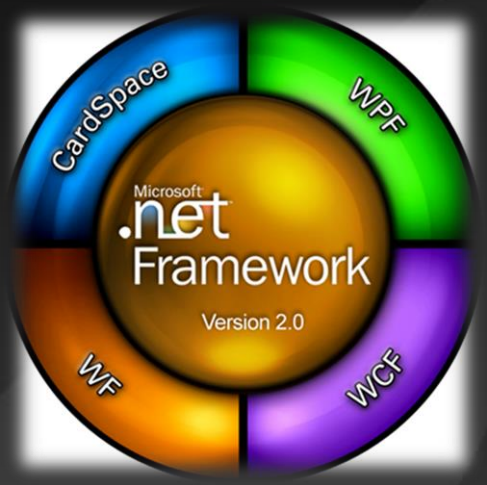
Microsoft® .NET

Sommaire

- Introduction
- Pourquoi WCF ?
- Les bases de WCF
- Création d'un Flux RSS

Introduction

- Windows Communication Foundation (WCF) est une brique du framework .Net 3.0.



Introduction

- Retour en arrière …
  - Avant WCF, aucun moyen unique pour communiquer entre les applications.

  Par exemple :

  Com+, Dcom, MSQM, .Net Remoting, Web Service, entreprise library, TCP/IP …

Pourquoi WCF donc ?

- Il va permettre d'unifier tous ces moyens de communication. De plus cette technologie est directement intégrée maintenant dans les systèmes d'exploitations de Microsoft.(Windows XP SP3

  et vista, … Seven). Dans les versions précédentes il faut installer les briques de communication à la main.

Pourquoi WCF donc ?

- Ceci permet aux développeurs de se concentrer sur les fonctionnalités sans avoir à faire de paramétrage pour la communication des applications. Toutes ces opérations sont laissées aux administrateurs système, grâce aux fichiers de configuration.
- Une partie importante de WCF est l'intégration de mécanisme de sécurité et le rajout de protocoles exemple le P2P.

Le Programme

- Aujourd'hui :
  - Création de services de bases
  - Configuration des connexions
  - Utilisation dans un cas particulier
- Dans un prochain cours :
  - La sécurisation

Windows Communication Foundation

Les Bases

- Pour comprendre les mécanismes nous allons commencer par le « hello world » des services : la calculette !

Les Bases

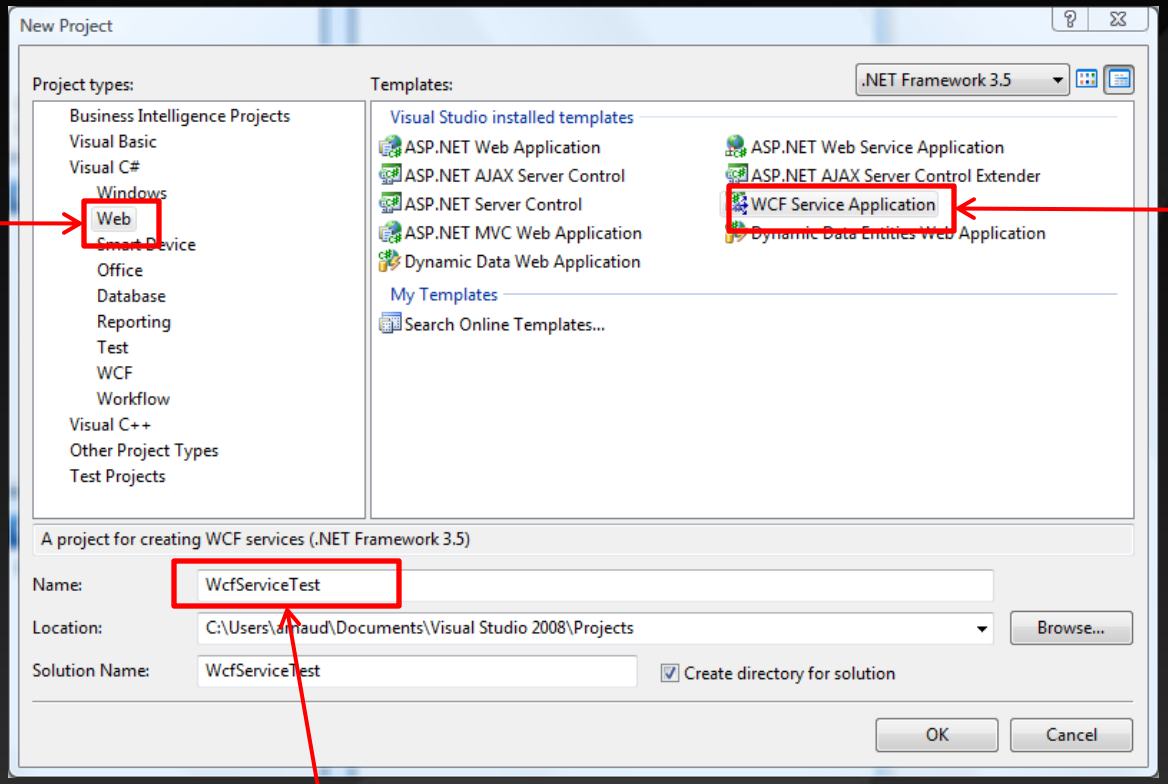
- Une application communicante signifie donc application client – serveur.
- Le serveur possédera une méthode addition qui sera exposée sur le Web, pour que le client console puisse l'utiliser.
- La méthode prendra en paramètre 2 int et retournera un int.

Les Bases

- Le serveur :
  - Il faut d'abord créer un projet Web->WCF Service Application.
  - C'est un template qui permet d'avoir une pré configuration, pour faire fonctionner des services.

Les Bases

Type de la solution

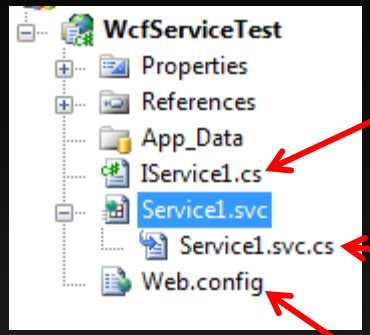


Projet WCF

Nom du projet

Les Bases

Composition d'un projet



Une interface qui décrit les méthodes qui seront exposées sur le web. (Dans l'exemple ci-contre)

Implémentation de l'interface :
c'est le service concrètement.

Fichier de configuration des protocoles et des connexions

Les Bases

Dans le fichier d'interface il faut décrire les méthodes dont vous avez besoins.
Par exemple :

```
[ServiceContract]
public interface IServiceCalc
{
    [OperationContract]
    int add(int a, int b);
}
```

Spécifie que l'interface est un contrat WCF

Nom du contrat

Spécifie que l'on va exposer la méthode, si on ne veut pas, il suffit de ne pas mettre l'attribut

Le prototype de la fonction

On peut mettre autant de fonctions que l'on veut dans un contrat, cependant il ne faut pas oublier de rajouter [OperationContract]

Les Bases

Maintenant il faut implémenter notre contrat dans le service:

Nom du service

```
public class ServiceCalc : IServiceCalc
{
    public int add(int a, int b)
    {
        return a + b;
    }
}
```

Nom du contrat à implémenter

Méthodes implémentées

Les Bases

```
<system.serviceModel>
  <services>
<!--name : nom du service doit mettre
    namespace.nomService-->
    <service
    behaviorConfiguration="WcfServiceTest.ServiceCalcBehav
    ior"
      name="WcfServiceTest.ServiceCalc">
      <endpoint address="" binding="wsHttpBinding"
    contract="WcfServiceTest.IServiceCalc" />
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <!--permet de spécifier certaines options-->
      <behavior name="WcfServiceTest.ServiceCalcBehavior">
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug
    includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Fichier de configuration du service. De base il est configuré par défaut.

Les Bases

Qu'est ce que l'ABC :

- Address : du serveur qui héberge le service
- Binding : façon dont sera exposé le service
- Contract : les méthodes exposées

```
<service
    behaviorConfiguration="WcfServiceTest.ServiceCalcBehavior"
      name="WcfServiceTest.ServiceCalc">
      <endpoint address="" binding="wsHttpBinding"
    contract="WcfServiceTest.IServiceCalc" />
    </service>
```

behaviorConfiguration : permet de spécifier des options pour le service
Name : nom du service. Cela correspond au namespace.NomService

Les Bases

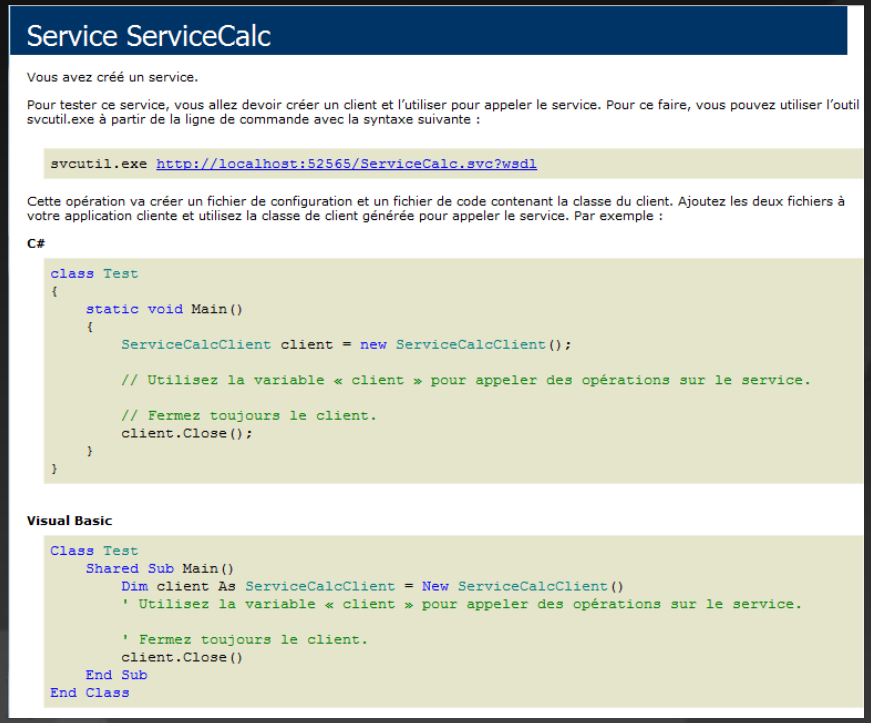
- Pour la configuration du service, il faut passer par les binding. Il y a un certain nombre de binding à connaître :

Binding	Protocol	Encodage	Description
basicHttpBinding	HTTP	XML 1.0	Interopérable
wsHttpBinding	HTTP	XML 1.0	Sécurité,
wsDualHttpBinding	HTTP	XML 1.0	Sécurité, communication double sens
netTcpBinding	TCP	Binaire	Sécurité, communication double sens
netNamedPipeBinding	tubes nommés	Binaire	Local seulement, communication double sens
netMsmqBinding	Message windows	Binaire	Windows seulement

Les Bases

- Si jamais toutes ces configurations ne répondent pas à vos besoins, vous pouvez en créer vous-même.

- Maintenant en faisant F5 on peu lancer le serveur. Si tout se passe bien on obtient ceci :



Service ServiceCalc

Vous avez créé un service.

Pour tester ce service, vous allez devoir créer un client et l'utiliser pour appeler le service. Pour ce faire, vous pouvez utiliser l'outil svcutil.exe à partir de la ligne de commande avec la syntaxe suivante :

```
svcutil.exe http://localhost:52565/ServiceCalc.svc?wsdl
```

Cette opération va créer un fichier de configuration et un fichier de code contenant la classe du client. Ajoutez les deux fichiers à votre application cliente et utilisez la classe de client générée pour appeler le service. Par exemple :

C#

```
class Test
{
    static void Main()
    {
        ServiceCalcClient client = new ServiceCalcClient();

        // Utilisez la variable « client » pour appeler des opérations sur le service.

        // Fermez toujours le client.
        client.Close();
    }
}
```

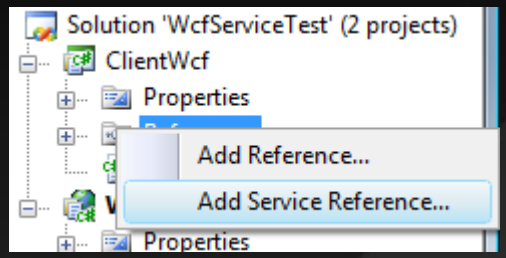
Visual Basic

```
Class Test
    Shared Sub Main()
        Dim client As ServiceCalcClient = New ServiceCalcClient()
        ' Utilisez la variable « client » pour appeler des opérations sur le service.

        ' Fermez toujours le client.
        client.Close()
    End Sub
End Class
```

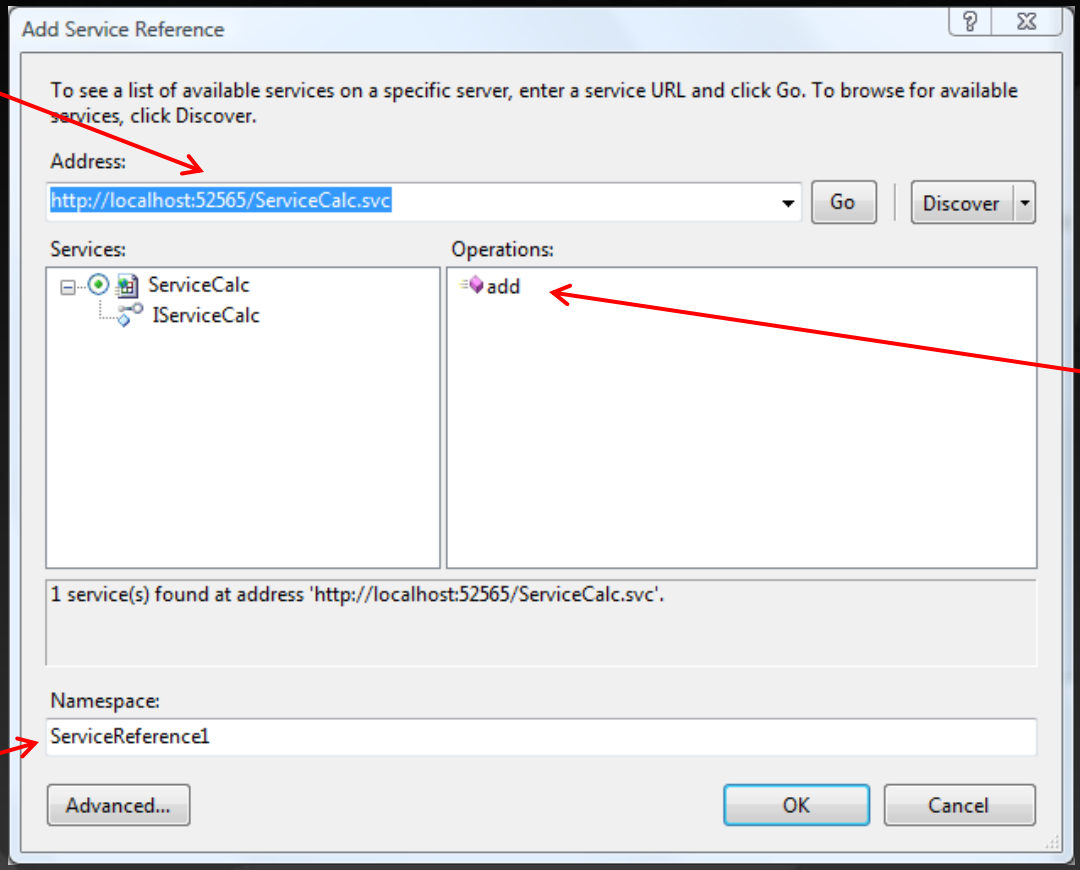
Les Bases

- Le client:
- Pour utiliser un service il faut l'ajouter aux références du projet



Les Bases

Adresse du service



Méthode sur le service

Nom par lequel vous accéderez au service dans le code C#

Les Bases

Utilisation en code C# :

Namespace du service

Proxy pour accéder aux méthodes

```
try
{
    ServiceReference1.ServiceCalcClient client = new ServiceReference1.ServiceCalcClient();
    int res = client.add(1, 2);
    Console.WriteLine(res);
}
catch (Exception ex)
{
    throw;
}
```

Appel de la méthode add

Les Bases

```
[DataContract]
public class Coord
{
    private int _x;
    private int _y;
     private string _res;
     public string Res
     {
    get { return _x.ToString() + _y.ToString(); }
     set { _res = value; }
     }

    [DataMember]
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }

    [DataMember]
    public int Y
    {
        get { return _y; }
        set { _y = value; }
    }
    public Coord() { _x = 0; _y = 0;  }
}
```

Dans l'exemple précédent on transmet uniquement des types simples. Cependant pour certains cas on peut avoir besoin de passer des classes. Il faut donc créer des classes spéciales.

Permet de spécifier que la classe va passer par un service WCF

Spécifie les membres que l'on pourra passer par les méthodes

En .net 4.0 les attributs classes / Méthodes ne sont plus obligatoires. Si c'est utilisé de cette manière tout est transféré

Les Bases

La variable Res ne sera jamais transmise au client. De plus aucune méthode ne peut circuler dans ces classes là. Par exemple une méthode toString() devra être écrite dans le serveur et dans le client

Les Bases

Au niveau du serveur cela se traduit par :

Pour le contrat :

```
[OperationContract]
Coord add(Coord a, Coord b);
```

Au niveau du service (implémentation du contrat) :

```
public Coord add(Coord a, Coord b)
{
    return new Coord() { X = a.X + b.X, Y = a.Y + b.Y };
}
```

Les Bases

- Il y a plusieurs façons d'héberger un serveur.
  - Sur un IIS
  - Une application managée liens
- Pour les clients
  - Un peu partout (application managée, asp.net, WF, …)

Installation IIS

- Pour faire fonctionner un service sur IIS, on peut être amené à installer certaines choses (Type Mime ….)

C:\Windows\Microsoft.NET\Framework\v3.0\Windows Communication Foundation\

Puis il faut utiliser ServiceModelReg.exe -i

L'option –i permet l'installation de tous les modules pour pouvoir utiliser WCF dans IIS, -u pour la désinstallation.

Remarques

- WCF gère avec difficulté le proxy
- Il y a un portage de fait vers le monde JAVA

Remarques

- WCF 4 :
  - Les beharviors peuvent être anonymes et donc s'appliquer à l'ensemble des endpoints
  - Prise en charge de plusieurs liaisons IIS (point de terminaison pour test.com et www.test.com
  - Prise en charge de WS
    - Discovery
    - Svc facultatif pour IIS
    - Mapping des URI (uri différent des noms de services)

Remarques

- Simplification du fichier de configuration
  - Binding par défaut

```
<configuration>
    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="CalculatorServiceBehavior">
                    <serviceMetadata httpGetEnabled="True"/>
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <services>
            <service
     name="Microsoft.Samples.GettingStarted.CalculatorService"

     behaviorConfiguration="CalculatorServiceBehavior">
                <endpoint address="" binding="basicHttpBinding"
     contract="ICalculator"/>
                <endpoint address="mex" binding="mexHttpBinding"
     contract="IMetadataExchange"/>
            </service>
        </services>
    </system.serviceModel>
</configuration>
```

```
<configuration>
    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="">
                    <serviceMetadata httpGetEnabled="True"/>
                </behavior>
            </serviceBehaviors>
        </behaviors>
    </system.serviceModel>
</configuration>
```

Flux RSS

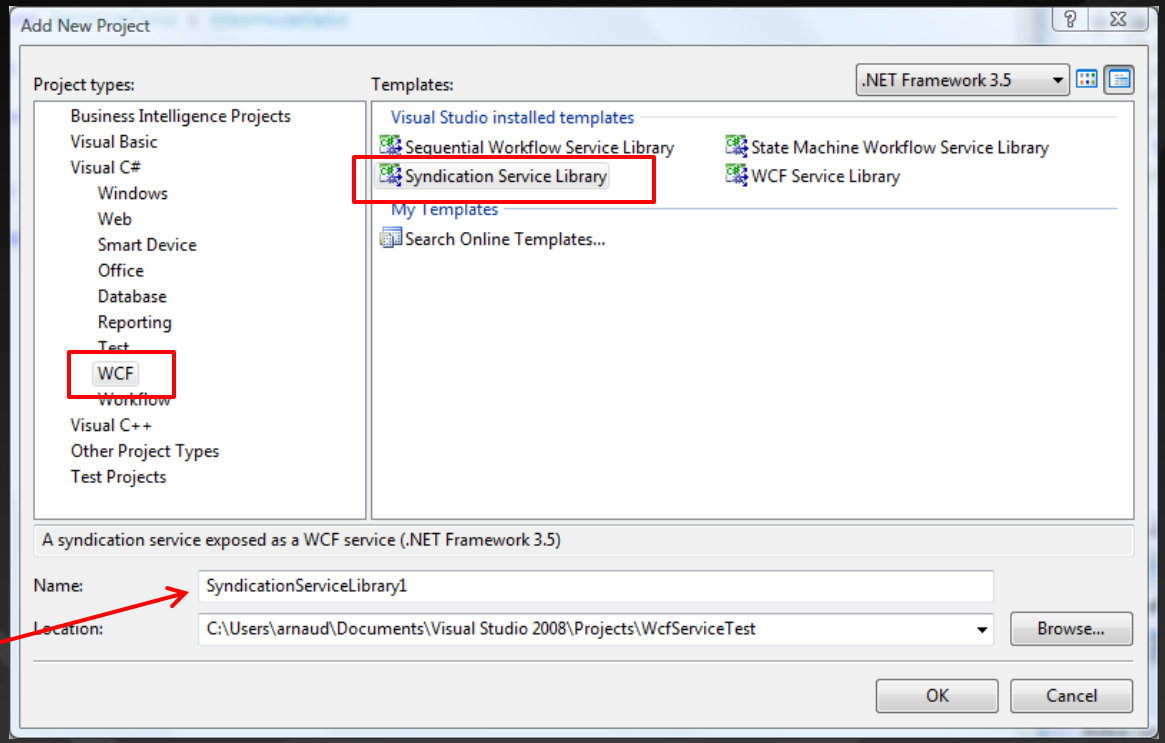
Pourquoi ?

- Les syndications Feeds ont été introduit en même temps que WCF. Donc c'est l'occasion de voir les mécanismes. Les flux RSS ne sont qu'un formatage XML.

- Mais avec WCF, cela devient beaucoup plus simple :

Création d'un Flux

- Pour cela il faut créer une nouvelle solution :
  - Syndication Service Library



Nom du futur projet

Création d'un Flux

Permet de déclarer le feed. En paramètre du constructeur il faut lui passer le titre du flux, puis sa description.

```
SyndicationFeed
```

Liste de tous les posts. Pour rajouter des posts il faut créer des SyndicationsItem

```
List<SyndicationItem> items = new List<SyndicationItem>();
SyndicationItem item = new SyndicationItem("An item", "Item
   content", null);
items.Add(item);
feed.Items = items;
```

Quand la liste est complète, il faut la mettre dans le flux.

Création d'un Flux

- L'utilisation de ce template est très utile, cependant il n'est pas prévu pour être déployé dans un IIS. Pour cela il faut faire quelques petites modifications :
  - Compiler le projet.
  - Créer une application sous IIS
  - Il faut copier et renommer app.config en web.config.
  - Déplacer le fichier nameProject.dll dans le /bin/

Création d'un Flux

Puis il faut ajouter un fichier de service :

Service.svc

Avec dedans :

```
<%@ ServiceHost Language="C#" Debug="true"
   Service="NameProject.Feed1" %>
```

Lecture d'un Flux

De la même façon, le framework .Net offre certaines facilités pour lire des flux et donc fabriquer un lecteur de flux personnalisable …

```
XmlReader reader =
   XmlReader.Create("http://www.developpez.com/rss.php");
SyndicationFeed feed = SyndicationFeed.Load(reader);
foreach (SyndicationItem item in feed.Items)
{
    Console.WriteLine(item.Title);
}
```


En 4.5

En 4.5

- Possibilité de faire du contract First génération un squelette de service à partir d'un WSDL depuis svcutil (à travers le /serviceContract)
- Le WSDL via l'option singleWsdl pourra être généré sous la forme d'un seul fichier contenant tout le schéma
- Support des WebSocket
- Simplification du fichier de configuration s'il contient les valeurs par défaut

QUESTIONS ?