SOAP

Objectifs

- Protocole SOAP
- Exemple d’implantation en C#
- Rendre son application résiliente

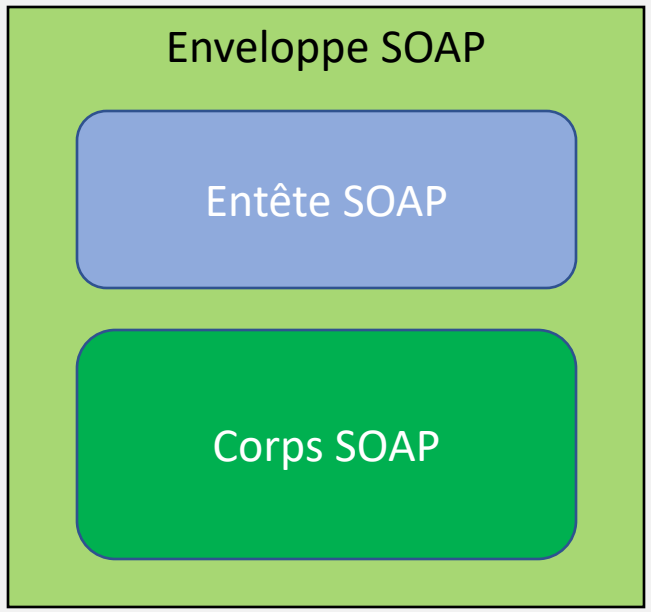
SOAP

- SOAP (*Simple* Object Access Protocol) : protocole d'échanges de données se basant sur XML :
  - Une enveloppe qui définit la structure du message et comment l'interpréter
  - Des règles et des définitions de types
  - Une convention de description d'appels de fonctions et de valeur de retour
- 3 principales caractéristiques :
  - Extensible : beaucoup d'extension de type WS-XYZ comme WS-Security, WS-AT (ACID), etc.
  - Neutre : peut fonction au dessus de TCP, d'UDP, HTTP, etc.
  - Indépendant : ne nécessite pas de technologie/langage de programmation particuliers

https://en.wikipedia.org/wiki/SOAP

SOAP

- On parle souvent d'appel de méthodes distantes ou Remote Procedure Call (RPC)
- Structure




```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
               xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>

  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Web Services Description Language - WSDL

- Description
  - Des types
  - Format des messages
  - Des fonctions d'un service
- Écrit en XML

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="h
xmlns:http="http://schemas.microsoft.com/ws/06/2004/policy/http" xmlns:msc="http:
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsu="http://docs.o
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" targetNamespace="http:
  <wsdl:types>
    <xs:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org
xmlns:ser="http://schemas.microsoft.com/2003/10/Serialization/">
      <xs:import namespace="http://schemas.datacontract.org/2004/07/Models" />
      <xs:import namespace="http://schemas.datacontract.org/2004/07/System" />
      <xs:element name="Ping">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="1" name="s" nillable="true" type="xs:string" /
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="PingResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="1" name="PingResult" nillable="true" type="xs:
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="PingComplexModel">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="1" name="inputModel" nillable="true" xmlns:q1=
```

Implantation en C# - Serveur

- Créez un projet ASP.Net MVC core
- Ajoutez le package Nuget « SoapCore »
- Créez une interface par service dans un projet séparé afin de les partager avec la partie cliente
- Implantez ces interfaces
- Déclarez les correspondances entre les interfaces et les classes concrètes dans la méthode « ConfigureServices » de la classe « StartUp »
- Dans notre cas nous allons utiliser le protocole HTTP

Implantation en C# - Serveur

```
[ServiceContract]
public interface IEchoService
{
    [OperationContract]
    string Echo(string p_message);
    [OperationContract]
    decimal CalculInteretAnnuel(decimal p_montant, decimal p_taux);
}
```

Implantation en C# - Serveur

```
public class EchoService : IEchoService
{
    public decimal CalculInteretAnnuel(decimal p_montant, decimal p_taux) {
        if (p_montant < 0.0m) {
            throw new ArgumentOutOfRangeException(nameof(p_montant),
                                                  "Le montant ne doit pas être négatif");
        }

        if (p_taux < 0.0m || p_taux > 1.0m) {
            throw new ArgumentOutOfRangeException(nameof(p_taux),
                                                  "Le taux doit être compris entre 0.0 et 1.0");
        }

        return p_montant * p_taux;
    }


    public string Echo(string p_message) {
        return p_message;
    }
}
```

Implantation en C# - Serveur

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IEchoService, EchoService>();
        services.AddSoapCore();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseEndpoints(endpoints => {
                            endpoints.UseSoapEndpoint<IEchoService>(opt =>
                            {
                                    opt.Path = "/EchoService.asmx »;
                                    opt.SoapSerializer = SoapSerializer.DataContractSerializer;
                            });
        });         // ...
    }
}
```

Implantation en C# - Client

```
Binding binding = new BasicHttpBinding();
EndpointAddress endpoint = new EndpointAddress(new Uri("http://localhost:5000/EchoService.asmx"));
ChannelFactory<IEchoService> channelFactory = new ChannelFactory<IEchoService>(binding, endpoint);
IEchoService echoService = channelFactory.CreateChannel();
```

Implantation en C# - Client

```
string echo = echoService.Echo("Bonjour DSED !");
try
{
    echoService.CalculInteretAnnuel(-1, 0);
}
catch (Exception ex)
{
    Console.Error.WriteLine(ex.Message);
}
```

Applications distribuées et tolérance aux échecs

- Les applications distribuées dépendent de réseaux pour communiquer avec divers services
- Les réseaux introduisent une probabilité accrue de coupures et d’échecs de connexion
- Pour gérer les interruptions et les erreurs de connexion, il est courant d'utiliser des patrons de conception
- Deux solutions principales : **Politique de réessai** et **Disjoncteur**
- Implantation :
  - Code : manuel ou bibliothèque (Ex. Polly en C#)
  - Infrastructure : utilisation d’un proxy (Ex. sidecar dans Kubernetes (Dapr))

Politique de réessai (Retry Policy)

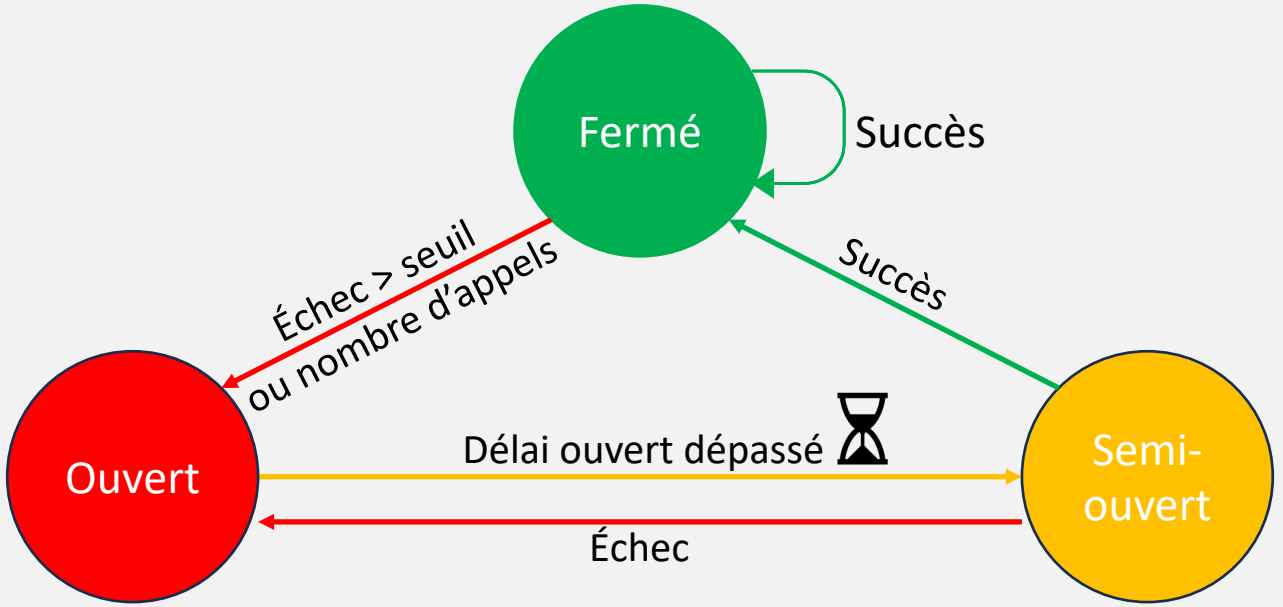
- La politique de réessai tente de répéter l’appel d’une opération en cas d’échec
- Utile pour les erreurs temporaires ou les problèmes de latence réseau

Implantation politique réessai – en C# – Client

```
int nombreEssais = 0;
int nombreMaximumEssais = 3;
bool appelEffectue = false;
decimal interet = -1m;
while (!appelEffectue && nombreEssais < nombreMaximumEssais) {
    ++nombreEssais;
    try {
        interet = echoService.CalculInteretAnnuel(100, .199m);
        appelEffectue = true;
    }
    catch (Exception) {
        if (nombreEssais >= nombreMaximumEssais) {
            throw;
        }
    }
}
```

Disjoncteur (Circuit Breaker)

- Le disjoncteur arrête temporairement les appels à un service si des erreurs répétées sont détectées
- Prévient l'épuisement des ressources en empêchant les appels inutiles jusqu'à ce que le service soit rétabli



Implantation disjoncteur – en C# – Client

```
class CircuitBreaker {
     private int echecSeuil = 3;
     private int essaisActuels = 0;
     private bool disjoncteurOuvert = false;
     private TimeSpan delaiReouverture = TimeSpan.FromSeconds(30);
     private DateTime dernieреOuverture;
     private bool enSemiOuvert = false;

     public decimal CalculInteretAnnuelAvecCB() {
          if (disjoncteurOuvert) {
               if (DateTime.Now >= dernieреOuverture + delaiReouverture)
               {
                    disjoncteurOuvert = false;
                    enSemiOuvert = true;
               } else {
                    throw new Exception("Service indisponible (Circuit
               Breaker activé).");
               }
          }
```

```
          try {
               decimal interet =
               echoService.CalculInteretAnnuel(100,
               .199m);
               essaisActuels = 0;
               enSemiOuvert = false;
               return interet;
          }
          catch (Exception) {
               if (enSemiOuvert) {
                    disjoncteurOuvert = true;
                    dernieреOuverture = DateTime.Now;
                    enSemiOuvert = false;
               } else {
                    essaisActuels++;
                    if (essaisActuels >= echecSeuil) {
                         disjoncteurOuvert = true;
                         dernieреOuverture =
                         DateTime.Now;
                    }
               }
               throw;
          }
     }
}
```

Extrait documentation Polly (Version modifiée)

```
var services = new ServiceCollection();
var optionsDefaults = new CircuitBreakerStrategyOptions();

// On définit un flux de traitement appelé "my-pipeline"
services.AddResiliencePipeline("my-pipeline", builder =>
{
    builder
    .AddRetry(new RetryStrategyOptions())
    .AddCircuitBreaker(optionsDefaults);
});

var serviceProvider = services.BuildServiceProvider();
pipeline = serviceProvider.GetRequiredKeyedService<ResiliencePipeline>("my-pipeline");

// Exemple d'utilisation du flux de travail
await pipeline.ExecuteAsync(static async token =>
{
        // Code qui peut temporairement échouer
});
```

https://www.pollydocs.org/getting-started.html

Référence

- https://en.wikipedia.org/wiki/SOAP
- https://github.com/DigDes/SoapCore
- Résilience :
  - Polly : https://www.pollydocs.org/strategies/index.html
  - Dapr : https://docs.dapr.io/concepts/resiliency-concept/