



Titre
Architecte Logiciel

Niveau I

2015

POUTORD Maxence

Titre du Mémoire : Enjeux d'une architecture orientée services dans le cadre du développement d'une application web

Soutenu par : Maxence POUTORD

Entreprise : GDF SUEZ SA

Tuteur : Franck DUBOIS

Date de remise du document :

Remerciements

En préambule de ce mémoire, je souhaite adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire.

Je tiens tout d'abord à remercier mon tuteur Franck DUBOIS sa bienveillance, ses conseils et tous nos autres moments d'échanges qui vont me manquer. Ces quelques lignes sont peu de choses par rapport à tout ce qu'il m'a apporté.

Mes remerciements s'adressent également à ma chef de projet Hélène WALLYN pour la confiance qu'elle m'a accordé.

Je ne saurais oublier Pascal GUCHET à qui j'adresse toute ma reconnaissance pour sa patience et les conseils qu'il m'a prodigués tout au long de mon alternance.

Enfin je tiens à remercier toutes les personnes avec qui j'ai pu travailler à Energy Formation. Je ne peux pas toutes les citer mais je pense qu'elles se reconnaitront à travers ces quelques lignes.

Merci à toutes et à tous.

Sommaire

Remerciements	2
Introduction	5
I Fonctionnement global de l'application	8
I.1 SOA, service et services web	8
I.2 Services web du site interne	11
I.3 Propositions de fonctionnement pour la mise à jour des données de stage	15
I.3.a. Proposition 1.1 : Création d'un service web et demande de prestation	15
I.3.b. Proposition 1.2 : Un formulaire unique pour les deux systèmes	17
I.3.c. Proposition 1.3 : Délocaliser la base de données	19
I.4 Propositions de fonctionnement pour l'affichage de l'offre de formation	20
I.4.a. Proposition 2.1 : Utiliser les services web du site interne	20
I.4.b. Proposition 2.2 : Reconstruire la page dynamiquement	21
I.4.c. Proposition 2.3 : Utiliser les données délocalisées	23
I.5 Solution retenue	24
II Outils et composants	26
II.1 Côté serveur	26
II.1.a. Symfony2, un framework PHP	26
II.1.b. Fonctionnement de Symfony2	27
II.1.c. Symfony2 et les services	29
II.1.d. Twig, un moteur de templates	32
II.1.e. Doctrine pour assurer la persistance des objets	33
II.2 Côté client	34
II.2.a. Utiliser un framework JavaScript MVC ?	34
II.2.b. Framework CSS	37
II.3 Outils annexes	39
II.3.a. Qualité du code	39
II.3.b. Génération de la documentation technique	41
II.3.c. Automatisation des tâches	41
II.3.d. Gestion des versions	42
II.3.e. Analyse d'audience internet	43
III Mise en place	45
III.1 Echange de données avec le site interne	45
III.1.a. Le protocole SOAP	45
III.1.b. Fonctionnement	47
III.1.c. Insertion et récupération	51
III.2 Echange de données avec le site externe	55
III.2.a. Génération des classes métiers	55
III.2.b. Classe de conversion	56
III.3 Récupération des données de l'application par des applications tierces	57
III.3.a. Le protocole HTTP	57
III.3.b. L'architecture REST	59
III.3.c. De l'API HTTP à l'API RESTful : Les 4 niveaux du modèle de maturité de Richardson	60
III.3.d. La place des services web de l'application dans le modèle de Richardson	67
III.4 Aperçu global de l'application	72

<i>Retour d'expérience</i>	<i>74</i>
<i>Conclusion.....</i>	<i>75</i>
<i>Bibliographie.....</i>	<i>78</i>
<i>Table des illustrations.....</i>	<i>79</i>
<i>Annexes</i>	<i>81</i>

Introduction

Energy Formation, centre de formation aux métiers du gaz du groupe GDF SUEZ propose un catalogue de 350 stages à deux populations différentes : des clients internes au groupe et des clients externes. Certains de ces stages sont réservés à une population, tandis que d'autres sont proposés aux deux populations.

Chaque profil (client interne et externe) a son offre sur un site web différent. L'offre des clients internes se trouve sur l'intranet du groupe, et celle destinée aux clients externes est accessible sur internet.

Le site intranet est en réalité une application permettant de gérer diverses activités d'Energy Formation comme la facturation, l'envoi de convocations, le suivi des inscriptions, ... et un portail web. Toute l'offre est présente dans la base de données de cette application, même si elle n'est pas forcément visible par les clients.

Le site internet, lui, ne sert que de vitrine. Les stages sont gérés via une interface d'administration.

Le site interne repose sur la technologie ASP.NET. Il est hébergé chez le prestataire et Energy Formation n'a pas accès au serveur. Cependant, le prestataire a développé des services web permettant d'interagir avec les données de l'application (à la fois en lecture et en écriture).

Le site externe est conçu avec la technologie PHP. Il est hébergé sur un serveur Apache avec MySQL comme Système de Gestion de Base de Données. Contrairement au site interne, il ne dispose pas de services web. En revanche, Energy Formation dispose d'un accès à ce serveur (pour y déposer des scripts) et au serveur de base de données MySQL.

Les 350 stages du catalogue sont régulièrement mis à jour en fonction de diverses variables (évolutions réglementaires, du matériel, des méthodes utilisées), si bien que pour chaque création ou modification de stage, les assistantes doivent assurer la mise à jour sur le site intranet et éventuellement sur le site extranet.

Schématiquement, le processus pourrait-être représenté comme ceci :

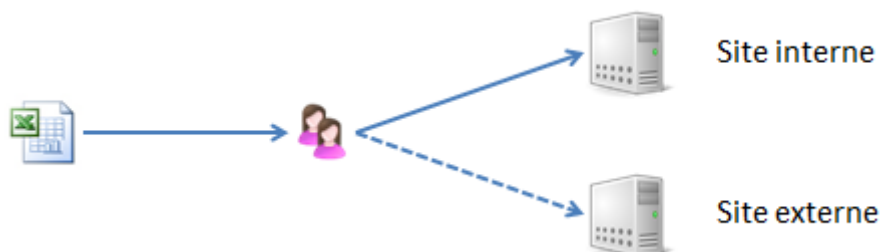


Figure 1 - Processus pour la mise à jour des données relatives aux stages

L'affichage de l'offre de formation ne donne pas satisfaction avec les outils du portail web du site interne car l'outil est très limité : pages d'erreurs non personnalisées, absence d'URL permettant d'identifier clairement les pages, recherche limitée de stage (seulement sur le libellé), ...

De ces deux constats est née l'idée de se doter d'une application permettant à la fois de fiabiliser la saisie des données relatives aux stages, et d'améliorer la communication vers ses clients.

L'objectif de cette application est donc double :

- Mettre à jour les données des stages sur le site interne et externe (si ce stage est proposé au public externe de l'entreprise) ;
- Améliorer la communication d'Energy Formation vers ses clients internes.

Pour répondre à ces objectifs, l'application devra être de type client-serveur avec une architecture à 3 niveaux :

- Le client : représente le programme exécuté par l'utilisateur. On distingue principalement 2 types de clients :
 - client lourd : la majeure partie du programme fonctionnera sur le client ;
 - client léger : la philosophie est l'inverse de celle du client lourd. Le client léger propose de déléguer la majeure partie des traitements au serveur, y compris la génération de l'interface.
- Le serveur d'application : fait le lien entre la demande du client et la source de données ;
- Le serveur de base de données : représente la source de données.

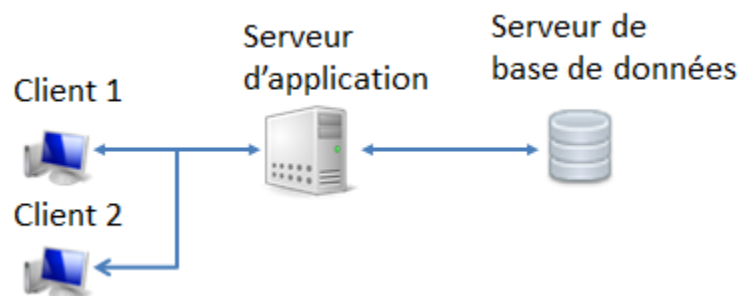


Figure 2 - Client-serveur avec une architecture à 3 niveaux

Avantage de ce modèle : plusieurs personnes peuvent se connecter de façon simultanée à une application ; ce qui correspond au principe de toutes les applications web.

Il existe certaines limites techniques imposées par l'entreprise, parmi lesquelles :

Type d'application : choix du type de client

L'environnement informatique de GDF SUEZ est, par mesure de sécurité, très restrictif. Créer une partie de l'application de type « client lourd » relève de l'impossible. De plus, le client lourd ne peut apporter une réponse satisfaisante au problème dans la mesure où le catalogue créé doit être sous forme web.

Le client sera donc de type « client léger », une application fonctionnant sur un navigateur web.



Dans le groupe GDF SUEZ, le navigateur web de référence est **Microsoft Internet Explorer 7** (abrégié IE7). Du fait de son ancienneté, ce navigateur est en général peu apprécié des développeurs car les

**Figure 3 -
Logo d'IE7**

standards actuels du web ne sont pas tous respectés. Conscient que le navigateur utilisé est désuet, le groupe GDF SUEZ laisse la possibilité de télécharger sur l'intranet du groupe des navigateurs plus récents (Google Chrome et Mozilla Firefox).

L'application devra donc être accessible aussi bien sur des navigateurs récents que sur des navigateurs plus anciens comme IE7.

En complément, comme beaucoup d'entreprises, GDF SUEZ dispose d'une charte graphique. La mise en production d'une application web à destination du grand public et des salariés du groupe nécessite le respect de cette charte.

Type d'application : choix du type de serveur

Le serveur utilisé sera celui sur lequel est hébergé le site externe. Il faudra donc utiliser la technologie PHP (en version 5.4) et utiliser MySQL comme Système de Gestion de Base de Données.

Utiliser ce serveur plutôt qu'un autre offre certains avantages :

- Le serveur est connu du groupe (les acquisitions de serveur ne sont pas choses simples) ;
- Le serveur est en liste blanche¹ pour tout le groupe et celui d'EDF (cette procédure a été très longue) ;
- L'accès à base de données du site externe n'est possible que depuis ce serveur (c'est une contrainte de l'hébergeur).

Ces différents éléments orientent l'architecture de la future application vers une architecture orientée services. A l'heure où les DSI (Directeurs des Systèmes d'Information) cherchent à rationaliser leurs coûts en mutualisant et en réutilisant l'existant, ces architectures sont très en vogue car elles répondent de manière pertinente à ces problématiques.

Il est donc légitime de se poser la question des enjeux des architectures orientées services dans le cadre d'applications web.

Ce mémoire a pour but d'y répondre au travers du projet préalablement introduit. Tout comme pour construire une maison, avant de se lancer dans la construction, il est primordial d'analyser l'environnement pour ensuite choisir les matériaux adéquats. La réponse à la problématique sera semblable à cet exemple : une première partie portera sur l'analyse des différentes possibilités d'architecture, la seconde portera sur le choix des outils et enfin la dernière fera un focus sur les différents services utilisés et leur mise en œuvre dans le système.

¹ http://fr.wikipedia.org/wiki/Liste_blanche

I Fonctionnement global de l'application

Avant de s'intéresser au fonctionnement des services web du site interne, il est nécessaire de comprendre quelques principes inhérents aux architectures orientées services.

I.1 SOA, service et services web

Les systèmes d'information (SI) des entreprises sont bâtis autour d'une multitude d'applications. Pour accéder à ses progiciels, l'utilisateur passe par des interfaces client/serveur (client lourd), et via son navigateur pour d'autres applications.

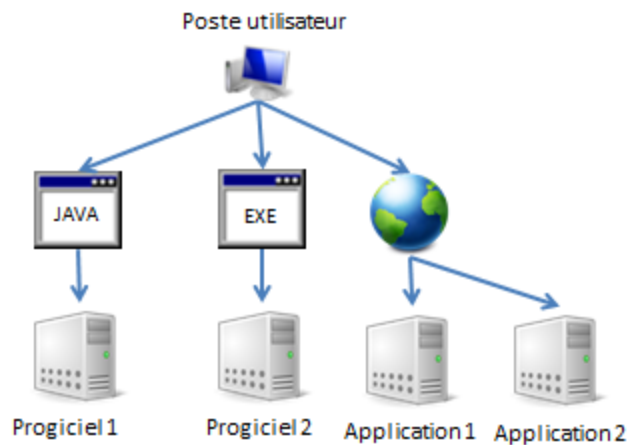


Figure 4 - Entropie du système d'information

Ce type d'architecture de système d'information pose un problème majeur : les applications ne communiquent pas entre elles. Par conséquent, il y a une redondance d'informations. Recopier l'information d'une application A sur une application B, c'est prendre le risque que cette dernière ne soit pas à jour. Dans certains cas, cela peut poser problème (par exemple si la base de données d'un logiciel de paie n'est pas à jour).

Au fil du temps, l'informatique a pris de plus en plus de place dans l'entreprise jusqu'à devenir omniprésente. Il est alors devenu primordial pour les DSI (Directeurs des Systèmes d'Information) de maîtriser cette hétérogénéité d'applications.

C'est alors que les **architectures orientées services** (ou **SOA** pour Service Oriented Architectures) ont vu le jour dans les années 2000. Cette forme d'architecture favorise l'échange d'informations (via des services) entre plusieurs applications. Le système qui fournit le service est le fournisseur et celui qui le reçoit est le consommateur.

Un **service** est en fait une fonction logicielle qui met à disposition un accès vers une fonction métier. Son objectif est d'être simple d'utilisation et réutilisable.

Un **service web** (ou web service en anglais) est un programme qui permet l'échange et la manipulation de données applicatives. Cet échange se fait par le biais d'internet ou d'intranet via le protocole HTTP.

Pour illustrer ce qui précède, prenons l'exemple d'un magasin ayant mis en place une gouvernance SOA au sein de son système d'information.

Le client qui souhaite consulter le prix d'un produit peut soit :

- Se rendre en magasin s'adresser à un vendeur (qui regardera sur son logiciel) ;
- Se rendre sur le site du magasin ;
- Ou bien sur l'application mobile du magasin.

Dans chaque cas, le même service sera appelé.

De même, quand ce client ira sur un comparateur de prix, il utilisera le même service produit. Ce service, disponible depuis le web, est un service web.

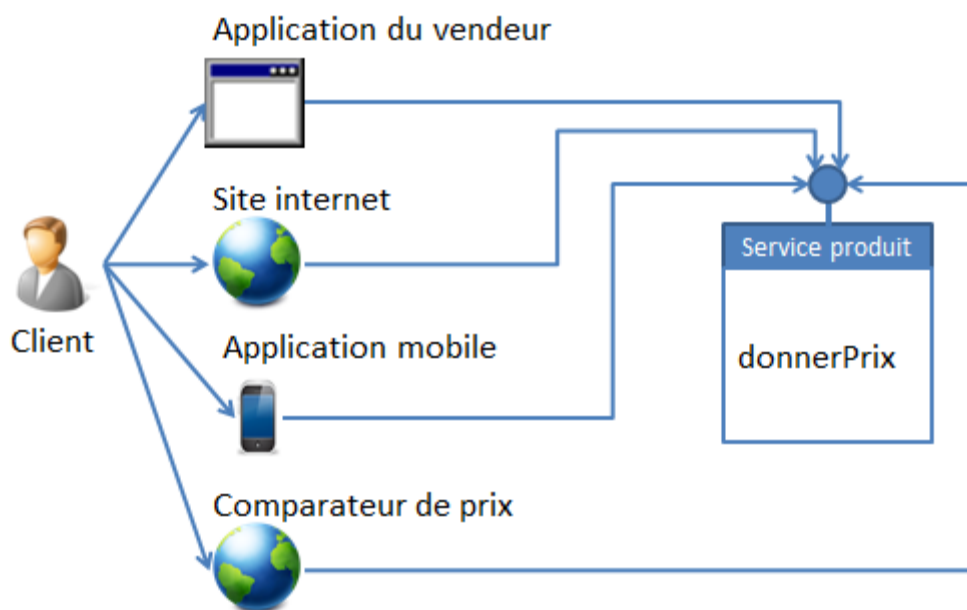


Figure 5 – Un intérêt de SOA, la mutualisation des services

Avantage pour le comparateur de prix : il n'a pas besoin de connaître la logique métier du magasin en question pour afficher ses prix.

Les services peuvent aussi appeler un autre service. En reprenant l'exemple ci-dessus, il est possible d'imaginer un service de commande permettant de s'adresser directement auprès du magasin. Pour calculer le montant de la commande, le service commande utiliserait le service produit afin de déterminer le coût unitaire d'un produit.

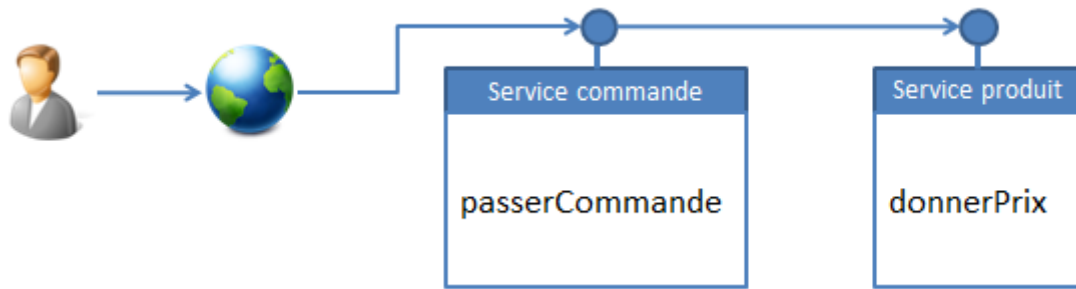


Figure 6 - Des services interopérables

Un service n'est pas qu'un élément destiné à être utilisé à l'extérieur de l'application. Il peut aussi l'être au sein même de l'application. Par exemple, il est possible de créer un service de mail. Dans l'application dès qu'une opération nécessitera l'envoi de mail, ce service sera appelé.

La réponse peut se faire sous plusieurs formats : XML, JSON et YAML. Voici pour un même objet Commande, ses différentes déclinaisons :

XML (*Extensible Markup Language* ou « *langage de balisage extensible* » en français) :

```

<commande id="20140601-35" date="01-06-2014">
  <articles>
    <article id="1" libelle="clé USB 32Go" prix="14.85" />
    <article id="2" libelle="clavier" prix="10.00" />
    <article id="3" libelle="souris" prix="12.82" />
  </articles>
</commande>

```

JSON (*JavaScript Object Notation*) :

```

{
  "commande": {
    "id": "20140601-35",
    "date": "01-06-2014",
    "articles": {
      "article": [
        { "id": "1", "libelle": "clé USB 32Go", "prix": "14.85" },
        { "id": "2", "libelle": "clavier", "prix": "10.00" },
        { "id": "3", "libelle": "souris", "prix": "12.82" }
      ]
    }
  }
}

```

YAML² (YAML Ain't Markup Language ou « YAML n'est pas un langage de balisage » en français) :

```
commande:
  id: 20140601-35
  date: 01-06-2014
  articles:
    article:
      id: 1
      libelle: clé USB 32Go
      prix: 14.85
    article:
      id: 2
      libelle: clavier
      prix: 10.00
    article:
      id: 2
      libelle: souris
      prix: 12.82
```

Maintenant que ces termes sont définis, il est temps de s'intéresser au fonctionnement global de l'application.

Cependant, avant d'imaginer comment l'application pourrait fonctionner, la première chose à faire est de dresser un état des lieux de l'existant pour ensuite décider de l'architecture qui répondrait le mieux aux besoins.

1.2 Services web du site interne

Pour bien comprendre les services web, il est nécessaire de savoir comment fonctionne l'application interne de manière globale.

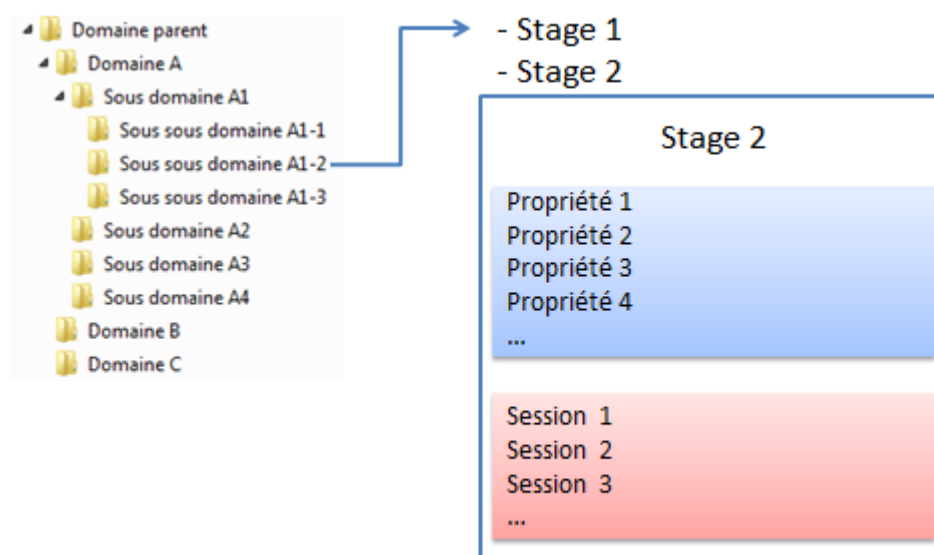


Figure 7 - Organisation des stages dans le site interne et structure d'un stage

² <http://www.yaml.org>

Les stages (aussi appelés modules) sont rangés dans des domaines. Ces domaines peuvent s'apparenter à des répertoires Windows. Le stage est lui-même composé de plusieurs propriétés tel que sa durée, ses prérequis, l'objectif de formation... Ces **propriétés** sont, de par la structure de l'application, dynamiques. C'est-à-dire qu'un administrateur de l'application peut rajouter si besoin un champ formulaire de saisie des fiches de stage, et lui affecter un type de valeur (chaîne de caractères, texte, entier naturel, liste déroulante...).

Autre composante d'un stage : les **sessions** de formation qui lui sont affectées. Les gestionnaires de formation des clients d'Energy Formation ont directement accès au site interne pour pouvoir inscrire leurs employés aux formations. Pour qu'un gestionnaire ait accès à une session, il faut la dupliquer (une duplication par gestionnaire). La base de données liée aux sessions de stage est assez conséquente car une session est ouverte à une dizaine de gestionnaires environ (le nombre varie en fonction du stage).

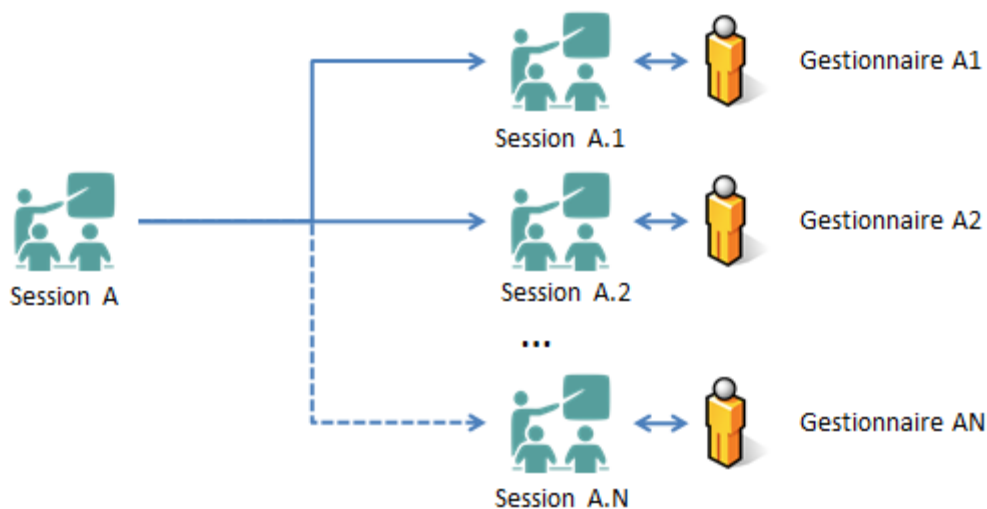


Figure 8 - Les sessions sont dupliquées pour l'accès aux gestionnaires

Chaque stage comporte une trentaine de propriétés et peut avoir entre deux et une dizaine de sessions.

Deux catégories de services web peuvent interagir avec ces données : les services Formation et les services Session (au sens session de connexion).

Services Web Formation :

Fonction	Description
AddModuleXml (AddStageXml)	Permet d'ajouter un stage dans l'application. Le stage sera « vide », seule la structure sera créée.
GetModuleXml (GetStageXml)	Permet d'accéder à un stage de l'application ainsi qu'à ses propriétés et une partie des informations sur les sessions.
GetProprieteXml	Permet d'accéder à la valeur d'une propriété d'un stage.
SetProprieteXml	Permet de renseigner la valeur d'une propriété de stage
GetSessionXml	Permet d'accéder au détail d'une session de formation d'un stage (nombre de stagiaires, place disponibles, lieu de formation...)
GetDomaineXml	Permet d'obtenir la liste des domaines
GetOffreXml	Permet d'obtenir l'ensemble des stages d'un domaine Pour chaque stage on obtient son enveloppe et une partie des informations des sessions. Par contre, GetOffreXml va limiter la recherche au domaine passé en paramètre. La fonction ne renverra pas les stages du sous domaine.

La terminaison XML signifie que la valeur de retour sera au format XML.
Pour une meilleure compréhension, le mot Module sera remplacé par Stage par la suite.

Services Web Session :

Fonction	Description
InitSession	Permet d'initier une session de connexion. Renverra un identifiant unique de connexion.
CheckSession	Permet de vérifier si l'identifiant unique de connexion est toujours valide.
EndSession	Permet de clore la connexion. Détruit l'identifiant de connexion.

Les différents services ne sont pas suffixés par le mot clé « Xml », cependant tous renvoient des documents sous forme XML.

Focus sur quelques services :

- GetStageXml :

La méthode GetStageXml permet d'obtenir le contenu d'un stage dont l'identifiant est passé en paramètres.

Ci-dessous, le rendu du service avec le stage « 7337 - Devenir Salarié d'Intervention de Sécurité » :

```
<?xml version="1.0" encoding="UTF-8"?>
<stage id="603" idparent="0" referentiel="FPC">
  <code><![CDATA[7337]]></code>
  <nom><![CDATA[Devenir Salarié d'Intervention de Sécurité]]></nom>
  <datemodification><![CDATA[20140717T103216]]></datemodification>
  <createur id="14" type="personne">
    <nom><![CDATA[DOE John]]></nom>
  </createur>
  <diffusionofl><![CDATA[12 participants.]]></diffusionofl>
  <sessions>
    <session id="144073" code="2014-7337-9-ZG" typeprocess="mono">
      <nom><![CDATA[Devenir salarié d'intervention de sécurité]]></nom>
      <datedebut><![CDATA[20141201]]></datedebut>
      <datefin><![CDATA[20141205]]></datefin>
      <datescompletes><![CDATA[Du 01/12/2014 au
05/12/2014]]></datescompletes>
      <lieuformation><![CDATA[GDF SUEZ ENERGY FORMATION-
GENNEVILLIERS]]></lieuformation>
      <lieuhebergement><![CDATA[]]></lieuhebergement>
    </session>
    ...
  </sessions>
  <champs>
    <champ id="862" typechamp="fiche" typedonnee="MEM">
      <code><![CDATA[PUBLIC]]></code>
      <nom><![CDATA[Population concernée]]></nom>
      <valeur><![CDATA[Salariés devant assurer l'intervention de sécurité
gaz]]></valeur>
      <valeurAffichee><![CDATA[Salariés devant assurer l'intervention de
sécurité gaz]]></valeurAffichee>
      <idvaleur><![CDATA[]]></idvaleur>
    </champ>
    ...
  </champs>
</stage>
```

Le fichier ci-dessus a été tronqué et seuls une session et un champ ont été gardés. La ressource retournée par le serveur contient 1600 lignes et 107 000 caractères (le fichier pèse un peu plus de 100ko) répartis de la façon suivante :

- 130 sessions (réellement 9) ;
- 59 propriétés (seules 30 sont utilisées).

- AddStageXml :

La méthode AddStageXml, permet de créer un stage dans la base de données. Les données retournées sont en XML et ne contiennent que l'identifiant du stage créé.

```
<?xml version="1.0" encoding="UTF-8"?>
<addStage>
  <Stage id="1964"/>
</addStage>
```

Les premiers tests permettent de faire le constat suivant : les services web du site interne sont rapides en écriture. En revanche, ils sont lents en lecture car le serveur doit générer un gros fichier et l'envoyer au consommateur du service.

I.3 Propositions de fonctionnement pour la mise à jour des données de stage

I.3.a. Proposition 1.1 : Création d'un service web et demande de prestation

Dans les schémas de cette partie, les traits bleus désignent les flux d'informations maîtrisés et les traits rouges désignent des flux d'informations non maîtrisés (car gérés par une autre application).

Parmi les solutions imaginables, il y en a une qui consiste à demander au prestataire du site interne d'exécuter un appel à un service web (du site externe). Le travail de l'application se limiterait alors à la création d'un service web, à moins que ce dernier ne soit développé par le prestataire du site externe.

Schématiquement, la partie modification des données des stages ressemblerait à ceci :

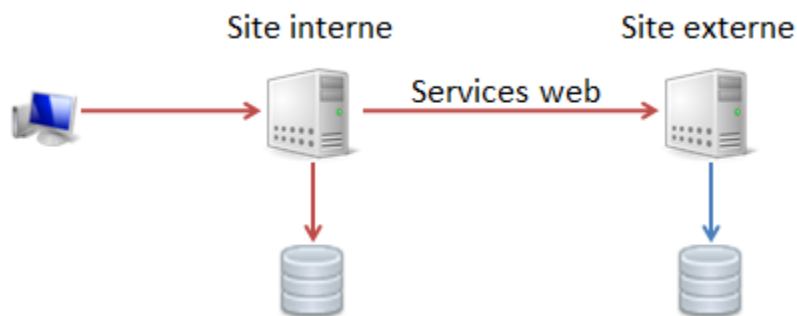


Figure 9 - Modification des données (proposition 1.1)

Un bon moyen d'analyser la pertinence d'une idée, est de recourir à la matrice SWOT³. SWOT est un acronyme issu de l'anglais : **S**trengths (forces), **W**eaknesses (faiblesses), **O**pportunities (opportunités), **T**hreats (menaces). Cet outil d'analyse et d'aide à la décision, s'il ne donne pas la solution idéale, offre une bonne vue d'ensemble d'une proposition de projet, de ses atouts et faiblesses.

Cette matrice se regarde de deux façons :

- Verticalement : avec à gauche, les éléments positifs et à droite les négatifs ;
- Horizontalement : avec en haut les éléments internes à l'entreprise et en bas, les éléments externes à l'entreprise.

³ <http://fr.wikipedia.org/wiki/SWOT>

	Positif	Négatif
Interne	Forces <ul style="list-style-type: none"> • Temps de réalisation moindre, les ressources peuvent-être mises sur d'autres projets • Les données ne sont pas dupliquées. 	Faiblesses <ul style="list-style-type: none"> • Accès très lent aux données des stages • Evite de multiplier les applications au sein du SI* de l'entreprise
Externe	Opportunités <ul style="list-style-type: none"> • Maintenable par le prestataire 	Menaces <ul style="list-style-type: none"> • Coûts élevés du prestataire • Le prestataire peut augmenter ses coûts pour la maintenance

*SI = Système d'Information

Sur l'écran de l'utilisateur, il a été imaginé d'ajouter en bas du formulaire existant une case à cocher. Si elle est cochée, alors un appel est lancé vers le web service.

Z115-Formation intensive à la communication de crise - Membres de Direction

Dernière modification : le 28/08/2012 par MARIE [REDACTED]

Général | Propriété | Inscription(s) | Fiche | Offre en ligne | Facturation | Achats | Progression pédagogique | Séquence | Notes

Informations générales

Code : Z115

Intitulé : Formation intensive à la communication de crise - Membres de Direction

Créateur : [REDACTED] Alexis

Sélection active : Oui

Visible dans le catalogue pdf : Oui

Modèle(s) de fiche : Modèle par défaut

Code Nomenclature : Sélectionnez une valeur...

Il y a actuellement 33 process pour cet élément.

Mettre à jour sur le site externe : ☒ 

Valider

Figure 10 - Proposition de maquette réalisée pour la proposition 1.1

I.3.b. Proposition 1.2 : Un formulaire unique pour les deux systèmes

Cette solution consiste à mettre à jour les données via des services web. L'utilisateur saisira les informations relative aux stages sur le formulaire de l'application et l'application se chargera de déverser les informations dans les deux systèmes d'information.

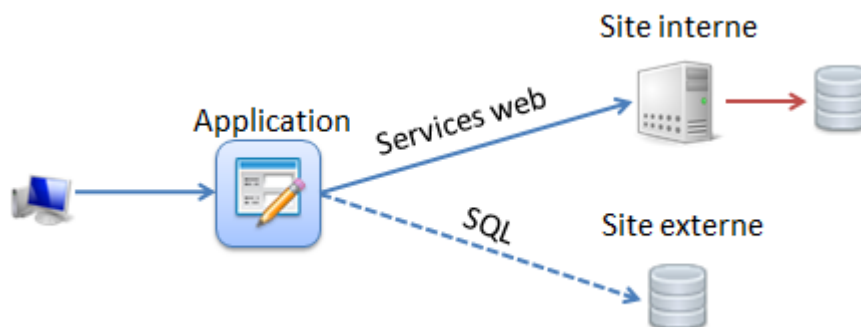


Figure 11 - Modification des données (proposition 1.2)

Cette proposition repose entièrement sur les services web. Comme en témoigne le diagramme ci-dessous, la modification d'un stage est particulièrement fastidieuse. Pour une meilleure lecture, les requêtes entre le navigateur et l'application ont été simplifiées.

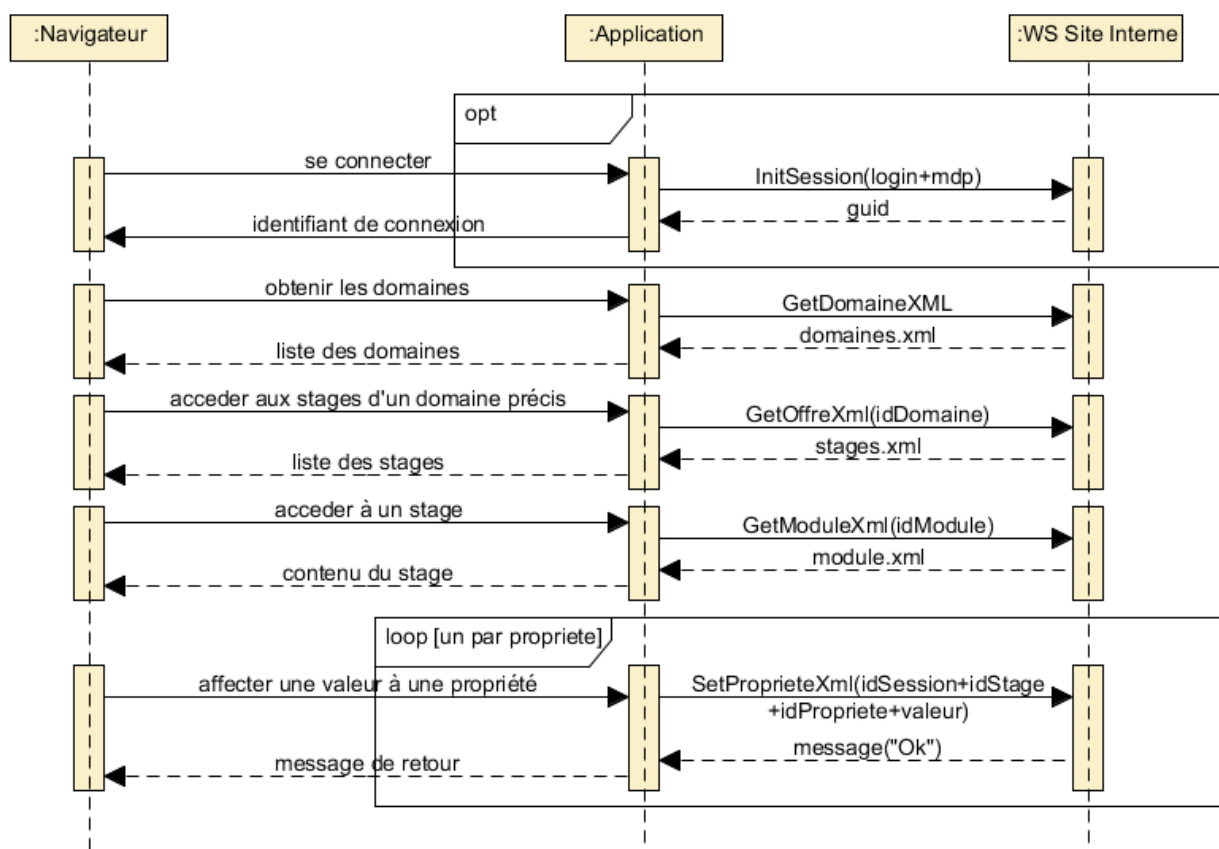


Figure 12 – Ensemble des requêtes nécessaires pour modifier un stage

Voici les différentes étapes nécessaires à la modification d'un stage :

1. Si ce n'est déjà fait, l'utilisateur se connecte avec ses identifiants. Son identifiant de connexion (guid) lui est restitué.
(Nota : il est aussi possible de stocker cet identifiant dans une variable de session du serveur).
2. Pour rechercher un stage, l'utilisateur :
 - a. parcourt l'arborescence
 - b. demande à afficher l'ensemble des stages d'un domaine
 - c. sélectionne son stage
3. Ensuite, le stage est récupéré. Le formulaire à l'écran de l'utilisateur est prérempli avec les informations que l'application est allée chercher.
4. Une fois le formulaire validé, une requête est envoyée au serveur pour enregistrer chaque propriété modifiée.

Le nombre de requêtes n'est en soit pas trop problématique dans la mesure où la communication entre deux serveurs est plutôt rapide. En revanche, les informations envoyées représentent une masse de données très importante et non négligeable, car synonyme d'un délai d'attente plus important pour le client. En effet, plus l'information qui transite est importante, plus le serveur mettra du temps à générer les données formatées, les envoyer et les interpréter.

Un épuisement de délai (timeout en anglais) peut aussi se produire, notamment lorsque la durée entre la requête et la réponse du serveur cible est trop importante. Ce phénomène se produit quand la réponse met beaucoup de temps à se construire ou bien quand la connexion est mauvaise entre les deux parties.

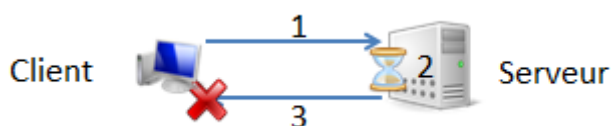


Figure 13 - Epuisements de délai

	Positif	Négatif
Interne	Forces <ul style="list-style-type: none"> • Les données ne sont pas dupliquées. Les données affichées sont celles du site interne • Temps de développement moyen • Temps de saisie plus court (utilisation de plugins plus modernes que ceux présents dans l'application) 	Faiblesses <ul style="list-style-type: none"> • Accès très lent aux données des stages • Risque d'épuisement de délai
Externe	Opportunités	Menaces <ul style="list-style-type: none"> • Risque de perdre le client

I.3.c. Proposition 1.3 : Délocaliser la base de données

Cette solution consiste à délocaliser les données de la base de données de référence vers une nouvelle base. Cette base serait alors le référentiel d'Energy Formation.

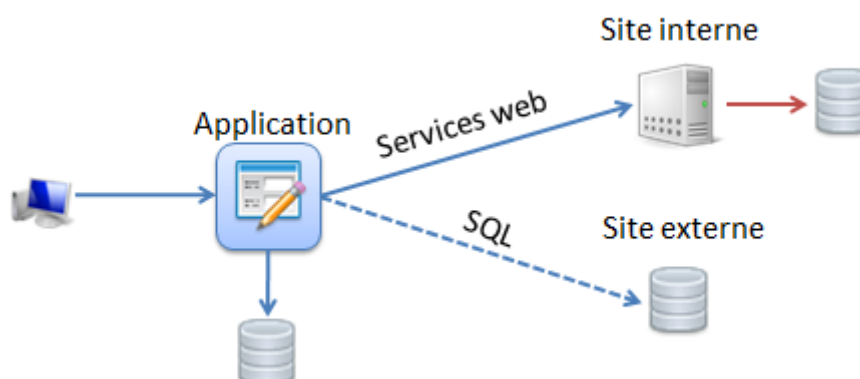


Figure 14 - Modification des données pour la proposition 1.3

Pour la partie catalogue, la source de données est la même que celle de l'application.

	Positif	Négatif
Interne	Forces <ul style="list-style-type: none"> • Accès aux données plus rapide que dans la solution 1.2 (si l'identifiant du stage sur le site interne est sauvegardé dans la base de données) • Temps de saisie plus court (utilisation de plugins plus modernes que ceux présents dans l'application) 	Faiblesses <ul style="list-style-type: none"> • Ajout d'une 3^e source de données • Temps de développement long • Risque de divergence entre les données de l'application et celles des autres applications
Externe	Opportunités	Menaces

I.4 Propositions de fonctionnement pour l'affichage de l'offre de formation

I.4.a. Proposition 2.1 : Utiliser les services web du site interne

L'accès à l'offre de formation se ferait par l'intermédiaire des services web.

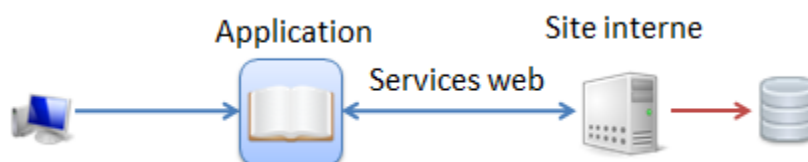


Figure 15 - Accès à l'offre de formation (proposition 2.1)

Comme en témoigne le diagramme de séquence ci-dessous, l'inconvénient majeur de cette proposition est le nombre très important de requêtes qui peuvent transiter entre l'application et le serveur de services web.

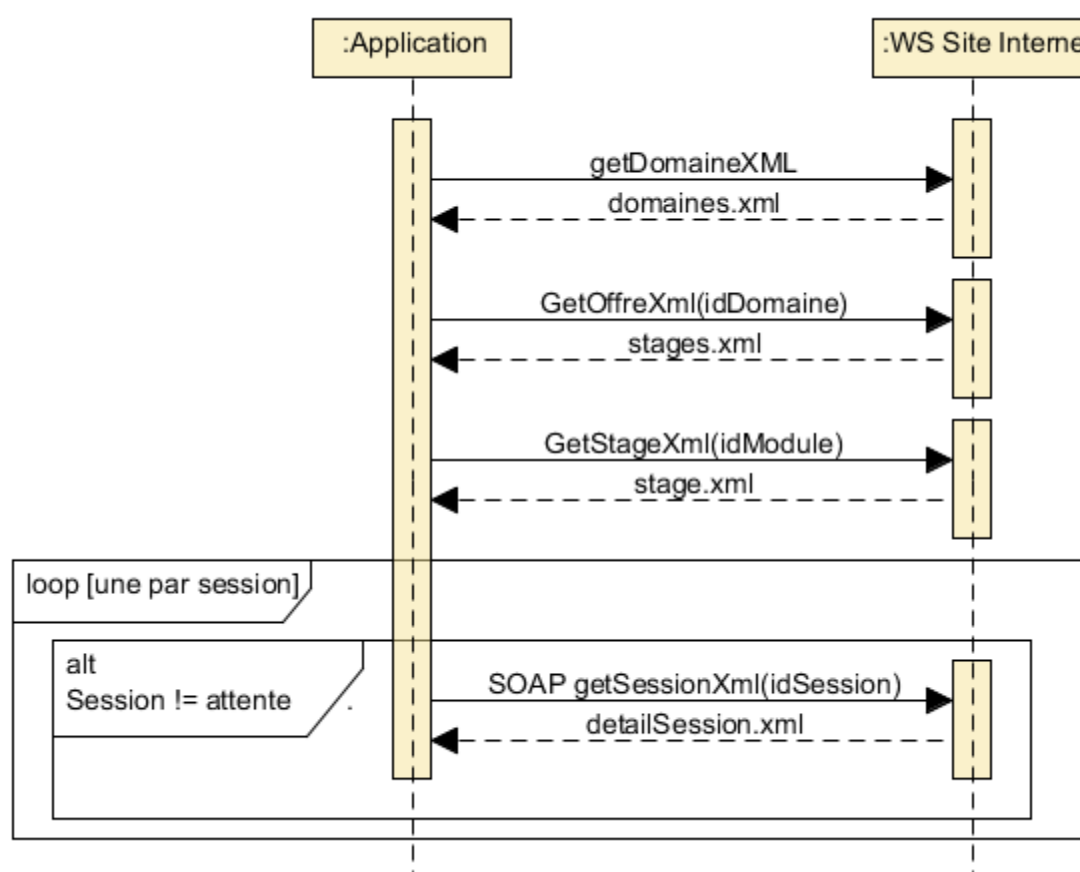


Figure 16 – Ensemble des services appelés pour accéder aux informations complètes d'un stage

Voici les différentes étapes pour afficher le contenu d'un stage :

1. Parcours de l'arborescence
2. Afficher l'ensemble des stages d'un domaine
3. Affichage du stage sélectionné
4. L'affichage est complété avec les sessions du stage sélectionné, sachant que, sur l'offre de formation, n'apparaissent pas les sessions en attente.

	Positif	Négatif
Interne	Forces <ul style="list-style-type: none"> • Fiabilité : les données à l'écran sont celles de la source de donnée cible • Temps de développement court • Pas besoin de dupliquer la base de données 	Faiblesses <ul style="list-style-type: none"> • Temps d'affichage très long (une requête peut mettre plus d'une minute à s'exécuter). • Ne résout pas le problème des fonctionnalités manquantes • Navigation très limitée • Epuisement de délai
Externe	Opportunités	Menaces <ul style="list-style-type: none"> • Risque de perdre le client en cours de route

Cette proposition est assujettie aux mêmes volumes de données que la proposition 1.2 et donc aux mêmes risques : latences et épuisements de délai.

I.4.b. Proposition 2.2 : Reconstruire la page dynamiquement

L'accès à l'offre de formation se ferait par l'intermédiaire des services web.

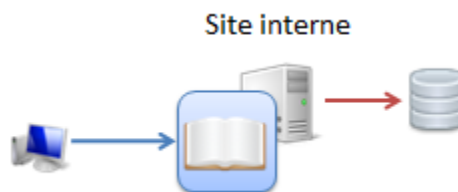


Figure 17 - Accès à l'offre de formation (proposition 2.2)

Pour améliorer le catalogue, une autre solution envisagée consisterait à reconstruire la page une fois chargée via du code JavaScript.

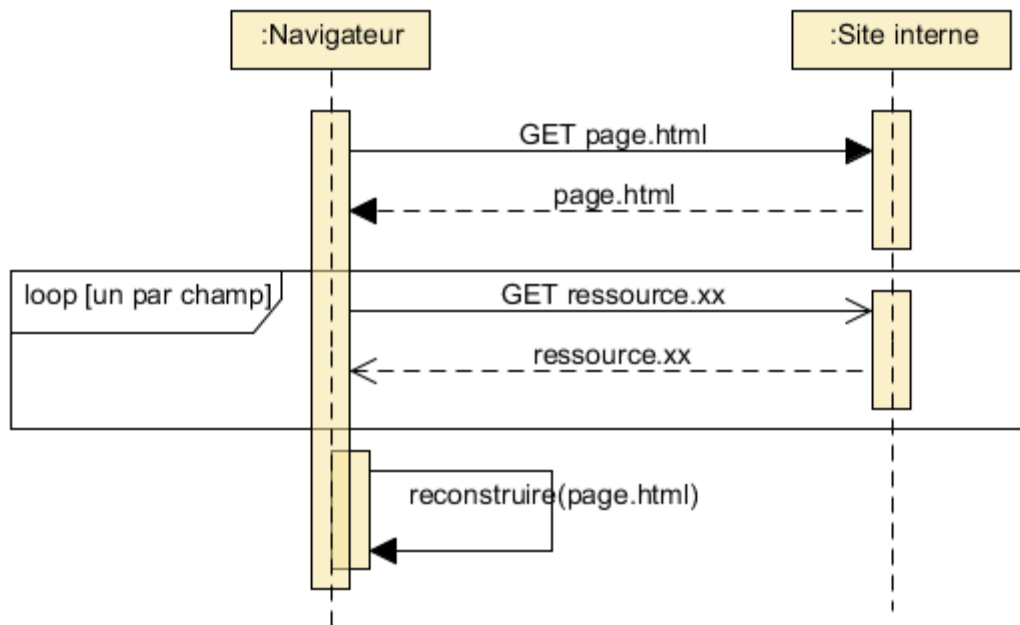


Figure 18 - Reconstruction dynamique d'une page web

Voici les différentes étapes du processus de reconstruction :

1. Le client demande la page ;
2. Une fois la page reçue, le navigateur l'affiche. Cependant, elle est dissociée de ses ressources ;
Le navigateur va télécharger de façon asynchrone toutes les ressources nécessaires de la page (scripts, feuilles de style et images) ;
3. Une fois complètement chargée (3-6 secondes), un script reconstruit la page avec les éléments qui la composent.

Le site interne repose sur la technologie web forms. Le code HTML produit par cette technologie est aussi lourd qu'indigeste, rendant toute modification de ce dernier presque impossible.

De plus cette solution ne serait qu'une modification esthétique de l'existant. Elle ne réglerait pas le problème des fonctionnalités manquantes.

	Positif	Négatif
Interne	Forces <ul style="list-style-type: none"> • Utilise les données du site • Pas dépendant des services web. Il n'y aura pas de maintenance en cas d'évolution. 	Faiblesses <ul style="list-style-type: none"> • Temps de développement long • Ne résout pas le problème des fonctionnalités manquantes • Navigation toujours très limitée • Maintenance lourde • Temps d'attente avant de reconstruire la page.
Externe	Opportunités	Menaces

I.4.c. Proposition 2.3 : Utiliser les données délocalisées

Cette solution consiste à afficher l'offre de formation avec les informations situées sur sa propre source de données.

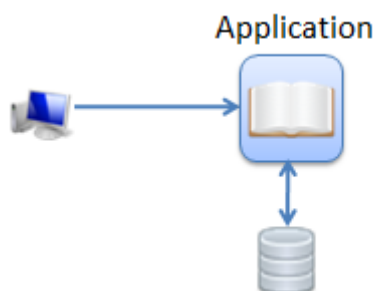


Figure 19 – Accès à l'offre de formation (proposition 2.3)

Nota : Cette solution n'est réalisable que si le choix est fait de délocaliser la base de données (proposition 1.3).

	Positif	Négatif
Interne	Forces <ul style="list-style-type: none"> • Catalogue hautement personnalisable • Possibilité de palier les fonctionnalités manquantes au catalogue existant (recherche sur n'importe quel élément) • Accès rapide aux données 	Faiblesses <ul style="list-style-type: none"> • Nécessite l'ajout d'une 3^e source de données • Temps de développement long
Externe	Opportunités	Menaces

1.5 Solution retenue

Les choix d'architecture influant sur la réussite ou l'échec d'un projet, il convient d'y apporter une attention toute particulière.

Préférant réaliser l'application en son sein, Energy Formation n'a pas souhaité demander au prestataire de réaliser ce qui figure dans la proposition 1.1.

En lecture, les services web sont très – voire trop – longs. Si l'on impose trop de temps d'attente entre plusieurs pages, l'utilisateur final risque, à juste titre, de voir cette application comme une régression et non comme une évolution.

Susciter la désapprobation des utilisateurs, c'est compromettre la réussite du projet et faire échouer à coups sûr une phase si importante qu'est la conduite du changement.

Après analyse des scénarios présentés ci-dessus, le choix s'est porté sur les solutions 1.3 et 2.3.

En concaténant les deux propositions, voici schématiquement à quoi ressemblera l'application.

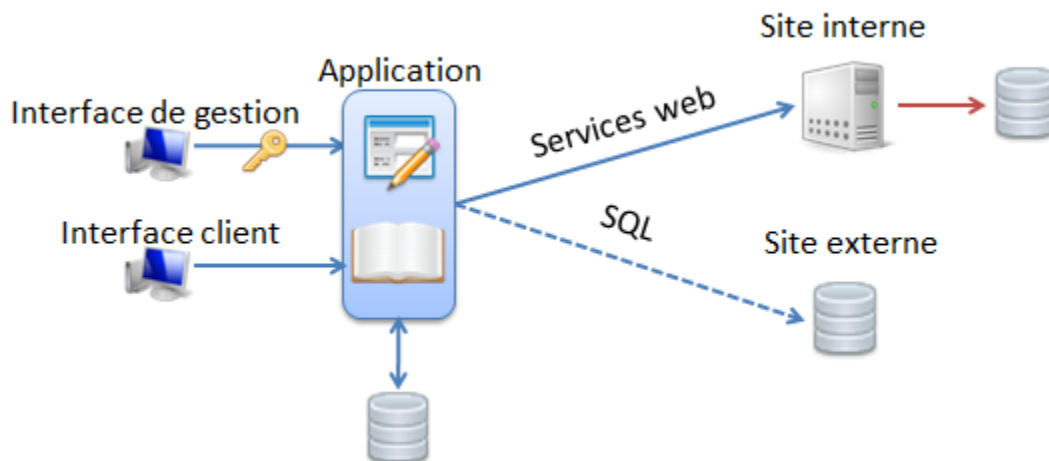


Figure 20 - Fonctionnement global validé

Une application séparée en deux parties :

- Une partie Catalogue : où le client pourra visualiser l'offre de formation qui se servira exclusivement des données de sa propre source (à l'exception des données sur les sessions).
- Une partie Administration : où les assistantes d'Energy Formation pourront, une fois connectées (d'où la clef), gérer à la fois le catalogue et l'insertion des données dans les environnements cibles.

Voyant les avantages qu'ils pouvaient avoir à posséder une base de données des stages maîtrisée, Energy Formation a **ajouté une nouvelle contrainte au projet : l'application devra être accessible depuis d'autres en lecture et écriture**. Cette nouvelle contrainte ne posait pas de problèmes à ce stade du projet dans la mesure où :

- Le développement n'avait pas commencé ;
- L'accès aux données ne remettait pas en cause l'architecture globale validée.

Cependant, en plus d'ajouter un degré de complexité supplémentaire à l'application, cette contrainte retardera le temps de développement et donc la mise en ligne.

L'architecture globale de l'application étant validée, il est maintenant temps de définir les bons outils.

II Outils et composants

II.1 Côté serveur

Utiliser un framework ?

Tout d'abord, qu'est-ce qu'un framework ?

Voici la définition qu'en donne Wikipedia⁴ : « un **framework** est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel ». Il faut voir le framework comme une boîte à outil qui évite développeur de « réinventer la roue » à chaque projet.

Les avantages liés à l'utilisation d'un framework sont multiples :

- Communauté : dans l'hypothèse où le framework est connu et reconnu ;
- Sécurité : les composants sont testés et approuvés par d'autres ;
- Structure : permet de mieux structurer le code ;
- Vitesse : la quasi-totalité du code produit est destiné à l'application. Ainsi, il n'y a pas à se soucier de développer des outils de bas niveau (auto chargement des classes, gestion du cache, ...).

En complément, l'apprentissage du framework en lui-même et de ses composants nécessite un temps qui peut devenir conséquent.

II.1.a. Symfony2, un framework PHP



Figure 21 - Logo de Symfony

Enormément de framework existent dans l'écosystème PHP. Parmi eux, Symfony2, un framework libre écrit en PHP5. Il est développé par l'entreprise Française Sensio Labs dont le créateur est Fabien POTENCIER.

Voici quelques-uns des points forts de Symfony2 :

Sécurité :

Chaque formulaire est nativement protégé des failles de type CSRF⁵. Leur traitement est, lui, protégé des attaques de type injection SQL⁶...

Communauté :

Symfony2 est un framework très utilisé et reconnu (Dailymotion, Yahoo!, BBC News, ...) avec une communauté active. En plus de sa documentation plutôt conséquente⁷ (et traduite en plusieurs langues), il existe de nombreuses formations en ligne pour apprendre à l'utiliser⁸.

⁴ <http://fr.wikipedia.org/wiki/Framework>

⁵ http://fr.wikipedia.org/wiki/Cross-Site_Request_Forgery

⁶ <http://php.net/manual/fr/security.database.sql-injection.php>

⁷ <http://symfony.com/doc/current/index.html>

⁸ <http://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2>

Partage :

Symfony2 découpe les fonctionnalités applicatives (utilisateur, création de PDF, ...) en **bundles**. Concept qui abordé plus loin.

Interface en ligne de commande :

Symfony2 dispose d'un mode console très puissant permettant, entre autres, de : générer la base de données (ou une partie), générer du code, déboguer, ... La génération de code est très appréciée des développeurs : une fois une entité créée, il est possible d'ajouter en une ligne de commande ses contrôleurs et ses vues. On obtient ainsi un modèle CRUD prêt à l'emploi. L'acronyme CRUD désigne les quatre opérations de persistance d'un objet : créer (**C**reate), lire (**R**ead), mettre à jour (**U**psdate) et supprimer (**D**eleter).

Le recours à ces générateurs permet de gagner du temps et de se concentrer sur des éléments plus complexes.

Barre d'outils de débogage (debug):

En mode développement, une barre d'outils est présente sur chaque page de l'application.

Voici ce à quoi elle ressemble :



Figure 22 - Barre d'outils de débogage du framework Symfony2

Elle fournit des informations intéressantes pour le développeur telles que :

- 1 : Le statut HTTP renvoyé par le serveur (200),
- 2 : Le temps pour générer la page (32114 millisecondes),
- 3 : Le nombre de requêtes vers la base de données (4).

Il est par ailleurs possible d'obtenir un détail en cliquant sur la zone souhaitée.

II.1.b. Fonctionnement de Symfony2

Symfony2 et MVC :

Ce framework utilise l'architecture MVC ou Modèle-Vue-Contrôleur, un architecture ayant pour objectif de scinder les classes métiers (modèle), des affichages sur le poste client (vue) et les traitements (contrôleur). Ainsi on évite de mélanger dans le même script : les appels de base de données, le code HTML et la logique métier. L'avantage d'un tel découpage est qu'il permet une plus grande maintenabilité et évolutivité du produit.

Pour mieux comprendre le fonctionnement de ce framework, voici le cheminement d'une requête :

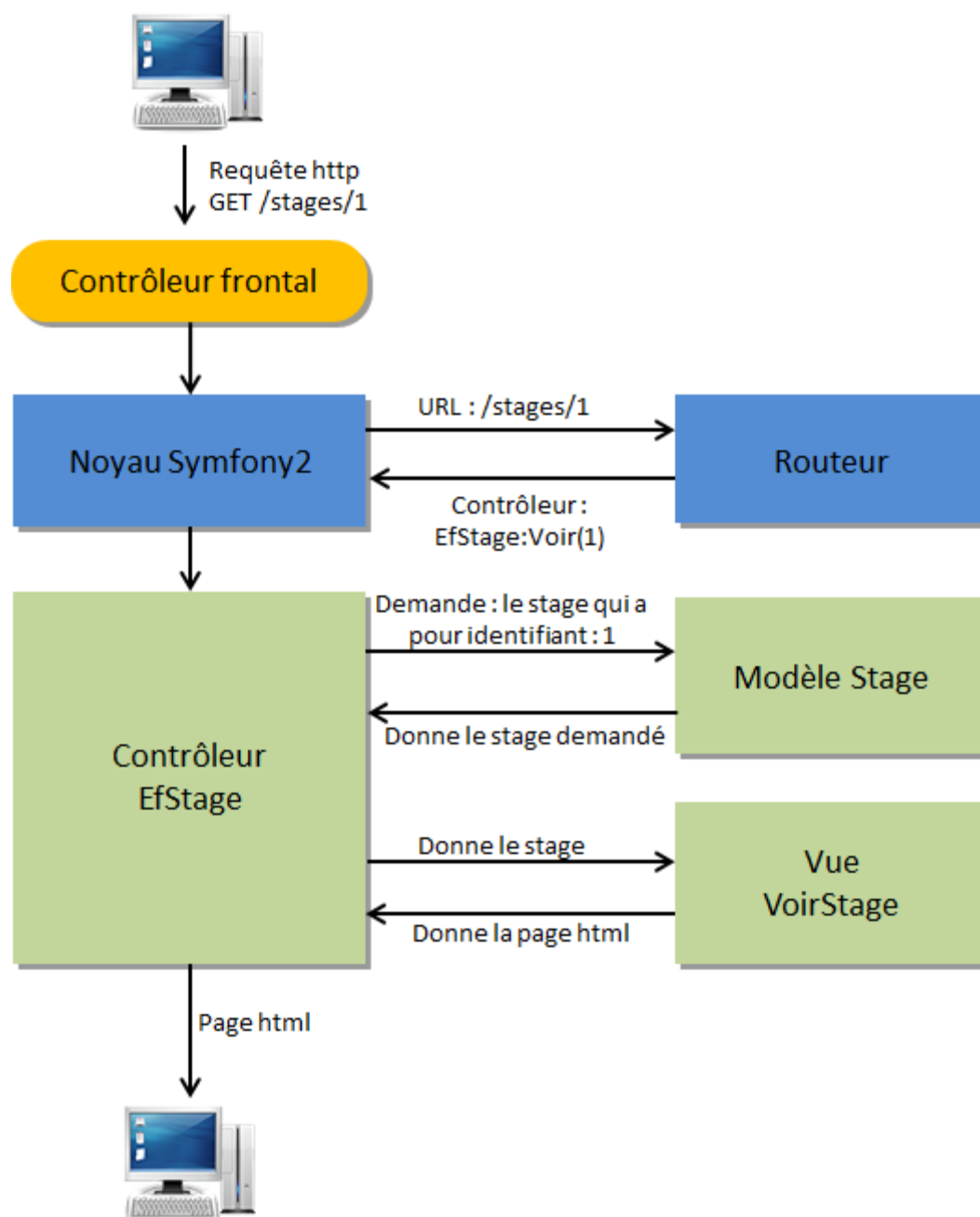


Figure 23 - Cheminement d'une requête dans une application Symfony2⁹

⁹ Schéma inspiré de la formation Symfony2 du site OpenClassrooms : <http://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2>

1. Le poste client demande la page /stages/1 ;
2. La requête est passée dans le contrôleur frontal qu'il enverra au noyau ;
3. Le noyau demande au routeur de résoudre cette URL ;
En retour, le routeur lui répond qu'il faut exécuter la méthode Voir() du contrôleur EfStage et lui passer "1" en paramètre ;
4. Le contrôleur va consulter le modèle Stage pour lui demander le stage qui a pour identifiant "1" ;
5. Le contrôleur va demander à la vue VoirStage de lui fournir le rendu HTML en fonction de l'objet Stage passé en paramètre.
6. Le contrôleur retourne au poste client le résultat de sa requête.

Concept de bundle (plugin) :

La spécificité de Symfony2 s'appuie sur le concept du "tout module". Chaque application est considérée comme un ensemble de "bundle". Ces bundles peuvent-être comparés à des plugins.

Imaginons un développeur ayant à concevoir deux applications :

- Une application X qui permet à des utilisateurs d'acheter des produits et d'imprimer les factures au format PDF.
- Une application Y qui permet à des utilisateurs d'écrire des articles, de commenter ceux des autres et de les imprimer au format PDF.

En utilisant le framework Symfony2, le développeur découpera ses applications de la façon suivante :

- Pour l'application X : UtilisateurBundle, ProduitBundle et PDFBundle ;
- Pour l'application Y : UtilisateurBundle, ArticleBundle et PDFBundle.

Le développeur pourra réutiliser les bundles utilisateur et PDF, s'évitant ainsi une réécriture.

La communauté Symfony2 étant très active, nombreux sont les développeurs qui partagent leurs bundles.

Si le bundle publié ne couvre que 80% des besoins, le développeur a deux solutions :

- Soit il ajoute au bundle les fonctionnalités restantes et propose éventuellement de les partager à la communauté via des plateformes communautaires telles que GitHub ;
- Soit il surcharge les classes du bundle (via le mécanisme d'héritage) pour que le code couvre l'intégralité du besoin.

II.1.c. Symfony2 et les services

Symfony2 permet d'avoir une architecture orientée services. Les bonnes pratiques recommandent d'utiliser des services ; ceux-ci sont très largement présents dans le framework de base.

Le framework Symfony2 possède un conteneur de services : le service container¹⁰. Son rôle est de gérer les services de l'application en les instanciant et en les fournissant au contrôleur demandé.

¹⁰ http://symfony.com/fr/doc/current/book/service_container.html

Ainsi, tous les services enregistrés dans l'application se trouveront dans ce conteneur. Pour enregistrer un service, il faut noter dans un fichier de configuration où il est accessible et lui donner un nom qui lui sera unique. Ensuite, pour utiliser un service souhaité, il faudra le demander au conteneur de services.

Pour illustrer ces notions, voici un schéma montrant comment fonctionne ce conteneur de services au moment où un contrôleur demande un service (Service1), sachant que ce dernier dépend d'un autre (Service2).

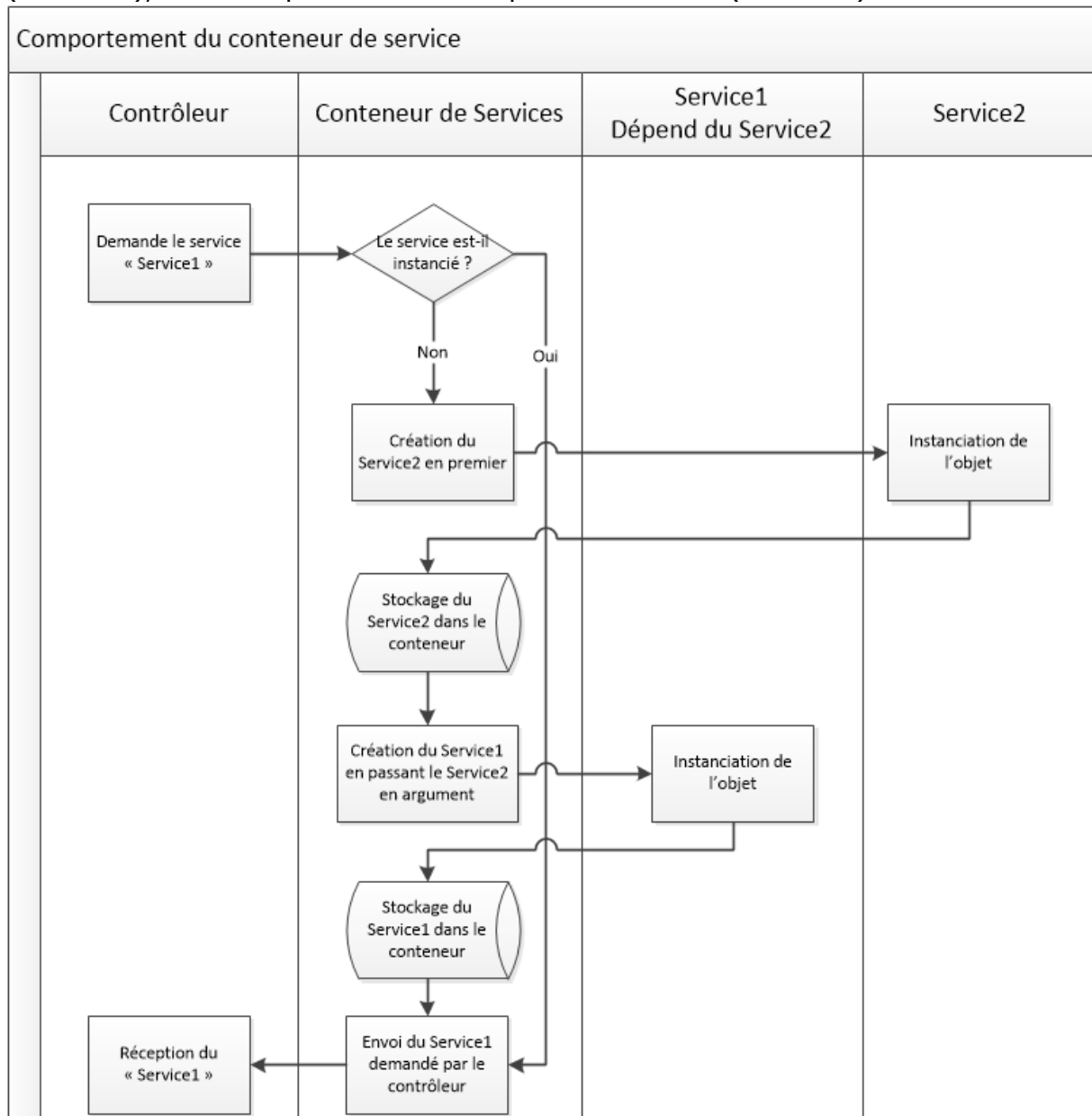


Figure 24 - Fonctionnement du conteneur de service¹¹

Avantage des services : la classe du service n'est instanciée que lorsqu'elle est appelée dans le code. Ainsi, une application qui possède 1 service et une application qui en possède 1 000 fonctionneront exactement pareil. De plus, la

¹¹ Source : <http://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/les-services-theorie-et-creation>

classe n'est instanciée qu'une seule fois lorsque le service est utilisé plusieurs fois.

L'unique instanciation d'un service se fait par le biais du patron de conception : Singleton. Ce patron de conception¹² (design pattern en anglais) permet de ne récupérer qu'une seule et unique instance de la classe de service en question. Son comportement est très simple, ce qui fait de lui le patron de conception le plus utilisé.

Le « Singleton » :

Soit une classe Singleton ayant :

- un attribut \$instance (en PHP les variables sont précédées d'un symbole dollar) ;
- un constructeur vide ;
- une méthode getInstance() ;
- et d'autres méthodes.

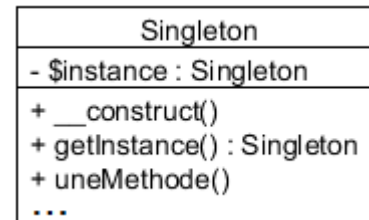


Figure 25 - Diagramme UML d'un Singleton

L'attribut \$instance va stocker l'instance de cette classe. Lorsque la méthode getInstance() s'exécutera, elle vérifiera si la variable est non nulle. Si elle est nulle, la classe va s'instancier. La méthode retournera ensuite la variable \$instance.

L'injection de dépendances :

Dans le développement d'applications conséquentes et dans les framework, un grand nombre d'objets sont manipulés.

- Un objet User pour l'utilisateur connecté ;
- Un objet Session pour gérer la session de l'utilisateur ;
- Un objet Mail pour gérer l'envoi de mail ;
- Un objet Log pour gérer les logs (fichiers journaux de l'application) de l'application
- Un objet DBal pour gérer l'accès à la base de données ;
- ...

Parmi tous les services, figureront un service logger (qui se chargera des logs) et un service mailer (qui se chargera des mails). Le premier faisant appel au second, pour envoyer par mail les fichiers journaux.

Pour cette réalisation, il serait tentant de mettre dans le constructeur du service logger, un constructeur du service mailer, ce qui apporterait les problèmes suivants :

- Le changement de nom de la classe Mail nécessiterait de parcourir tout le code pour modifier les appels ;
- Plusieurs instances du service mailer pourraient être créées.

Une solution consiste alors à injecter le service mailer dans le service de logs. Cela se fait par l'utilisation d'un patron de conception, celui de l'injection de

¹² http://fr.wikipedia.org/wiki/Patron_de_conception

dépendance (*Dependency Injection* en anglais). Ce patron s'illustre par le principe d'Hollywood : « *Ne nous appelez pas, nous vous appellerons* ». En effet, comme le montre le fichier de configuration ci-dessous, ce n'est pas le service logger qui va se charger de créer le service mailer mais bien le conteneur de services qui va passer le service mailer (qu'il aura préalablement instancié).

```
<services>
  <service id="mailer" class="EF\UtilBundle\Mailer">
    <argument>sendmail</argument>
  </service>

  <service id="logger" class="EF\UtilBundle\Logger">
    <argument>@mailer</argument>
  </service>
</services>
```

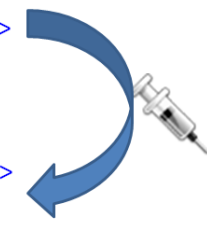


Figure 26 - Injection de service dans un autre

II.1.d. Twig, un moteur de templates



Twig est un gestionnaire de templates créé par SensioLabs, autrement dit, le même créateur que le framework Symfony. Un template est un patron (comme en couture) qui va s'occuper de structurer la page.

Le langage utilisé par les moteurs de templates est propre à chaque moteur. Twig a pour avantage d'être très simple, facilement intégrable dans Symfony, et le code est plus clair que l'inclusion de PHP dans une page HTML.

Figure 27 - Logo de TWIG

A titre de comparaison, voici un exemple pour afficher une variable (appelée nom) dans un paragraphe (balise p) :

PHP	<code><p>Bonjour <?php \$nom; ?> !</p></code>
TWIG	<code><p>Bonjour {{ nom }} !</p></code>

La syntaxe utilisée par Twig est beaucoup plus courte et par la même occasion plus compréhensible.

Les autres avantages de Twig sont :

- Facilité des héritages de templates : cela évite par exemple de recopier le contenu HTML des en-têtes et pieds de page sur chacune des pages de l'application.
- L'affichage des variables est sécurisé, ceci afin de protéger l'application des failles XSS¹³ (Cross-Site Scripting).
- Travail avec un graphiste : les graphistes ne connaissent pas tous PHP, la syntaxe de Twig est plus simple à comprendre.

¹³ http://fr.wikipedia.org/wiki/Cross-site_scripting

II.1.e. Doctrine pour assurer la persistance des objets



**Figure 28 -
Logo de
Doctrine**

Doctrine est un framework de mapping objet-relation (ou ORM pour object-relational mapping) pour PHP. C'est l'équivalent d'Entity Framework (technologies .NET) et d'Hibernate (technologies JAVA).

Cet ORM fournit la persistance des objets PHP en toute transparence.

Il consiste à associer une classe avec une table et chaque attribut avec un champ de la table. Ainsi le développeur se concentre uniquement sur ses objets et Doctrine se charge de :

- Convertir les objets métier en requêtes SQL et les exécuter ;
- Transformer les données de la base de données en objets métier PHP.

Pour que Doctrine puisse faire le lien entre une classe métier et la base de données, il faudra mapper cet objet, c'est-à-dire faire correspondre chaque propriété de l'objet à un champ dans la base de données. Le mapping se fait au travers d'annotations.

```
/**
 * @ORM\Table(name="ef_page")
 * @ORM\Entity()
 */
class Page
{
    /**
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @ORM\Column(name="titre", type="string", length=100, unique=true)
     */
    private $titre;
```

Figure 29 - Un objet mappé avec la base de données

L'exemple ci-dessus, montre un objet Page mappé avec la base de données. La table correspondante sera ef_page, la propriété de l'objet id jouera le rôle de clé primaire (avec un auto-incrément) et le champ titre sera un type string de 100 caractères et unique.

Sans ces annotations permettant de relier l'objet à la base de données, le framework ne peut pas savoir le type de variable affectée à cette dernière. En effet, le langage PHP n'est pas typé.

Doctrine permet aussi de faire des requêtes personnalisées avec son propre langage : le DQL (pour Doctrine Query Language). L'exemple ci-dessous montre la faible différence qu'il y a entre le DQL et le SQL :

Langage DQL	Langage SQL
SELECT u FROM MonProjet\Model\Utilisateur u WHERE u.age > 20	SELECT * FROM Utilisateur AS u WHERE u.age > 20

Des classes POPO :

« *La simplicité est la sophistication suprême* » - Léonard de Vinci

Doctrine persiste les entités de l'application qui sont sous la forme POPO. POPO est l'acronyme de Plain Old PHP Object (« bon vieil objet PHP »), un POPO est une classe n'implémentant pas d'interfaces spécifiques à un framework.

Ce terme a été inventé par Martin Fowler, Rebecca Parsons et Josh MacKenzie en 2000 : « *Nous nous sommes demandés pourquoi tout le monde était autant contre l'utilisation d'objets simples dans leurs systèmes et nous avons conclu que c'était parce que ces objets simples n'avaient pas un nom sympa. Alors, on leur en a donné un et ça a pris tout de suite.* »¹⁴.

En effet, bien que le paradigme objet remonte aux années 60, l'âge d'or de la programmation orientée objet remonte aux années 1990. Voyant la puissance de l'objet, beaucoup de développeurs se mirent à développer des objets les plus complexes les uns que les autres. C'est alors que sont apparues les classes POPO, allant en quelque sorte à contrecourant de ces tendances.

Le concept des classes POPO repose donc sur le principe KISS : *Keep It Simple, Stupid*, ce qui donne mot à mot en français : « Garde ça simple, stupide ». Effectivement, plus la conception d'une application est simple, plus sa maintenance et sa compréhension le seront.

Dans le patron de conception Modèle-Vue-Contrôleur (MVC), les classes correspondantes aux modèles, seront des classes POPO.

II.2 Côté client

II.2.a. Utiliser un framework JavaScript MVC ?

JavaScript est un langage objet destiné à ajouter de l'interactivité dans les pages internet. Depuis quelques années de nombreux frameworks JavaScript se popularisent.

Ces frameworks permettent de faire des applications web dites "monopages". Le principal avantage réside dans le fait que le chargement d'une page n'est que partiel et le temps de chargement est donc réduit.

¹⁴ <http://www.martinfowler.com/bliki/POJO.html>

Tout d'abord, voici à quoi ressemblerait l'application sans utiliser de framework JavaScript :

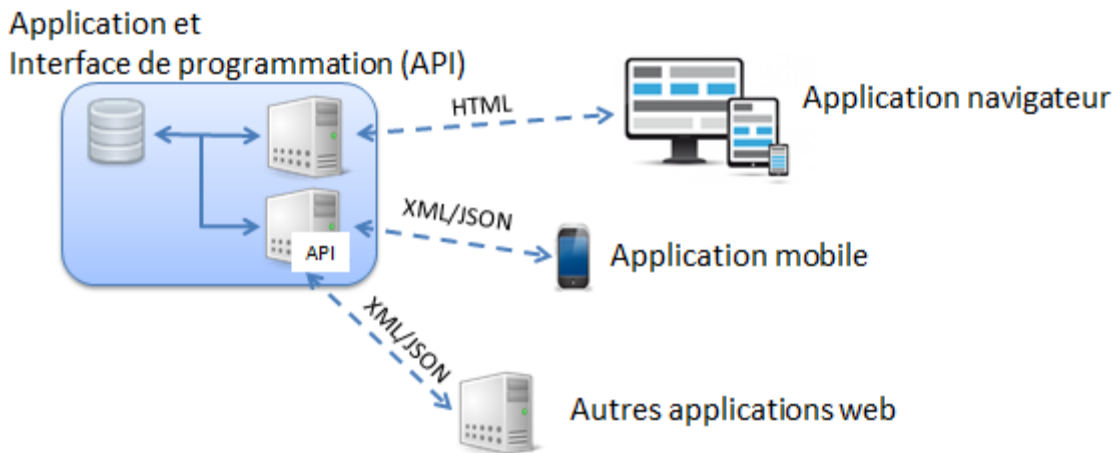


Figure 30 – Echange de données entre serveur et différents périphériques

Il faudrait dans ce cas scinder l'application en deux parties :

- Une partie application web permettant d'envoyer des pages HTML et de les traiter ;
- Une partie interface de programmation (API) permettant d'envoyer des données et de les traiter.

Utiliser un framework JavaScript MVC conduit à repenser la façon de construire son application du côté serveur. Le serveur se limite à une simple interface de programmation (ou API pour *Application Programming Interface*). Cette API envoie les données au format XML/JSON et l'application navigateur convertit ces données en données HTML, plus digestes pour l'utilisateur final.

Ainsi, si demain l'entreprise souhaite développer une application mobile (type Android, iPhone ou Windows Phone), seule la partie mobile sera à réaliser. Le traitement serveur sera le même ainsi que les requêtes envoyées à celui-ci.

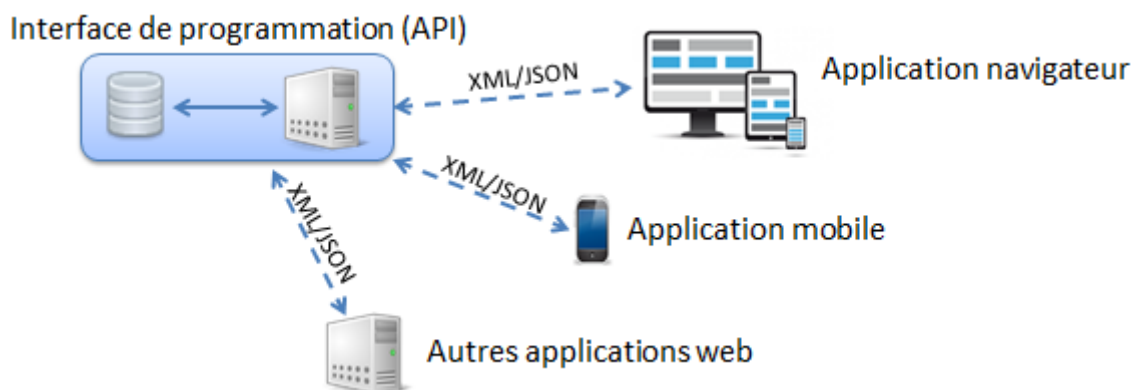


Figure 31 - Intérêt des framework JavaScript MVC

Avec ce type d'architecture, le client léger s'alourdit. On parle alors de client riche.

Cependant les frameworks JavaScript MVC les plus populaires (comme AngularJS, Backbone.js et Ember.js) ne sont pas compatibles avec Microsoft Internet Explorer 7. Souvent, la compatibilité est assurée à partir de la version 9.

L'idée d'utiliser un framework JavaScript MVC a alors été abandonnée. L'application fonctionnera à l'instar de la figure précédente, avec une application web et une interface de programmation pour pouvoir interagir avec les données.

Pour accroître l'expérience utilisateur, il est néanmoins conseillé d'utiliser JavaScript. Et bien que l'idée d'utiliser un framework ait été abandonnée, il existe cependant de nombreuses bibliothèques sur internet permettant de rendre les applications plus conviviales pour l'utilisateur final.



Figure 32 - Logo de jQuery

C'est le cas de jQuery, une bibliothèque JavaScript libre développée initialement par John RESIG. Cette bibliothèque permet de dynamiser et d'améliorer l'ergonomie d'une page web. L'avantage d'utiliser jQuery plutôt que de coder en JavaScript pur est que le code est compatible avec les plus grands

navigateurs du marché (dont le navigateur IE7), ce qui n'est pas le cas du langage JavaScript.

jQuery permet notamment de :

- Parcourir et manipuler le DOM (la structure de la page HTML) ;
- Gérer des événements utilisateurs (clic, double clic, focus, ...) ;
- Ajouter des effets visuels (fondu d'entrée, glissement, wizz, ...) ;
- Modifier le style de la page (CSS) ;
- Ajouter des interactions AJAX (« *Asynchronous JavaScript And XML* » soit en français « *JavaScript Asynchrone et XML* »). La technologie AJAX permet au navigateur d'envoyer et de recevoir des informations sans avoir à recharger la page.

La figure ci-dessous montre un avantage lié à AJAX : un champ de saisié « auto-complétant » alimenté de façon asynchrone.

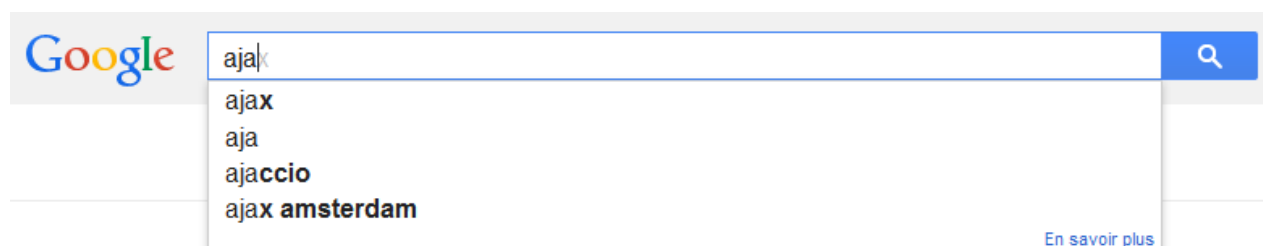


Figure 33 - Le moteur de recherche Google et la technologie AJAX

Selon W3Techs ¹⁵, jQuery représenterait 94,3% de parts de marché des bibliothèques JavaScript présentes sur la toile en octobre 2014.

Cette bibliothèque doit sa renommée au fait qu'elle est légère, bien documentée et simple d'utilisation.

¹⁵ <http://w3techs.com>

Avec des technologies comme jQuery et AJAX, il est possible de simuler le fonctionnement d'un framework JavaScript MVC. Cependant, jQuery n'est pas un framework mais bien une bibliothèque. L'application serait un conglomérat d'écouteurs d'évènements (ou *event listener* en anglais) complètement lié à la structure HTML de la page. La maintenance serait aussi périlleuse. On parle du syndrome du plat de spaghettis¹⁶.

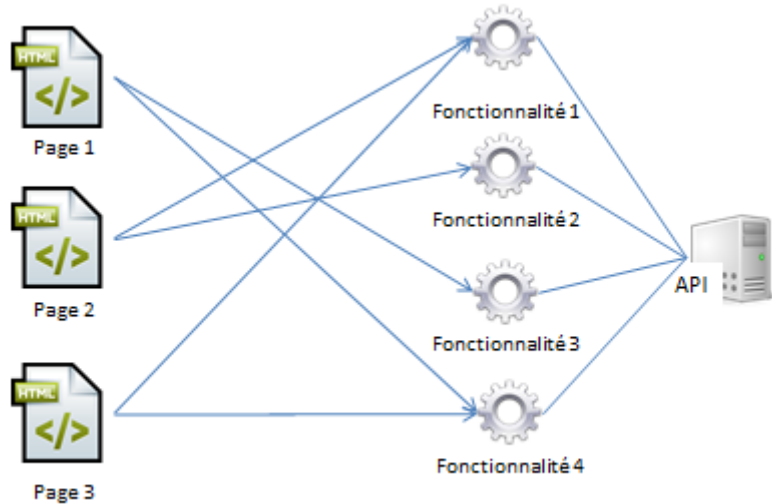


Figure 34 - Syndrome du plat de spaghettis

Il est donc fortement déconseillé de procéder ainsi. Ce n'est pas pour autant qu'il est déconseillé d'utiliser les fonctionnalités AJAX de jQuery, bien au contraire.

II.2.b. Framework CSS

Les feuilles de styles constituent un autre composant majeur des pages web. On parle de CSS (pour *Cascading Style Sheet*).



Figure 35 -
Logo de
Bootstrap

Tout comme les langages serveur et les langages clients, les feuilles de styles ont elles aussi des frameworks. Twitter Bootstrap est le plus connu. Comme son nom l'indique, il a été développé par Twitter Inc., l'entreprise détentrice du réseau social Twitter.

A l'instar de jQuery, Bootstrap est libre de droits, bien fourni et simple d'utilisation, ce qui fait de ce framework, l'un des plus populaires sur la plateforme de développement GitHub¹⁷. Un autre point fort de Bootstrap, il est personnalisable, et donc adaptable à la charte graphique imposée par le groupe GDF SUEZ.

Dans le cadre du développement, la dernière version (v3) a volontairement été délaissée au profit d'une version plus ancienne (version 2.3.2). En effet, depuis la version 3, la compatibilité avec le navigateur Microsoft Internet Explorer 7 n'est plus assurée. La dernière version à assurer la compatibilité avec ce navigateur est la version 2.3.2.

¹⁶ http://fr.wikipedia.org/wiki/Syndrome_du_plat_de_spaghettis

¹⁷ Projet Bootstrap sur la plateforme GitHub : <https://github.com/twbs/bootstrap>

Parmi les composants du framework, on retrouve une barre de menus. La facilité d'utilisation est telle qu'il n'y a pas besoin d'être graphiste de métier pour avoir une application esthétique.

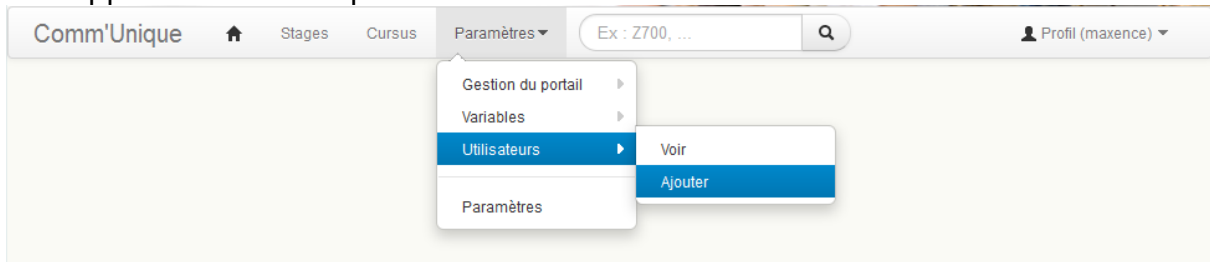


Figure 36 - Exemple de composants disponibles avec Bootstrap

Un autre élément important est que Bootstrap permet de créer en toute simplicité des sites web adaptatifs (aussi connus sous le nom de responsive web design). C'est-à-dire que contenu de la page s'adapte à l'écran de l'utilisateur : écran d'ordinateur, tablette tactile ou ordiphone (smartphone).

Là où les sites web adaptatifs tirent leur épingle du jeu par rapport aux sites qui ont une déclinaison ordiphone et/ou tablette tactile, c'est qu'il n'y a pas besoin de dupliquer les vues. Avec les sites web adaptatifs, une seule page html est renvoyée au navigateur. C'est ensuite au navigateur de l'adapter en fonction de l'écran qui consulte l'application web.

Comme une seule vue est envoyée au client, il n'y en a qu'une seule à maintenir.

Voici comment est réagencée une page web sur plusieurs périphériques :

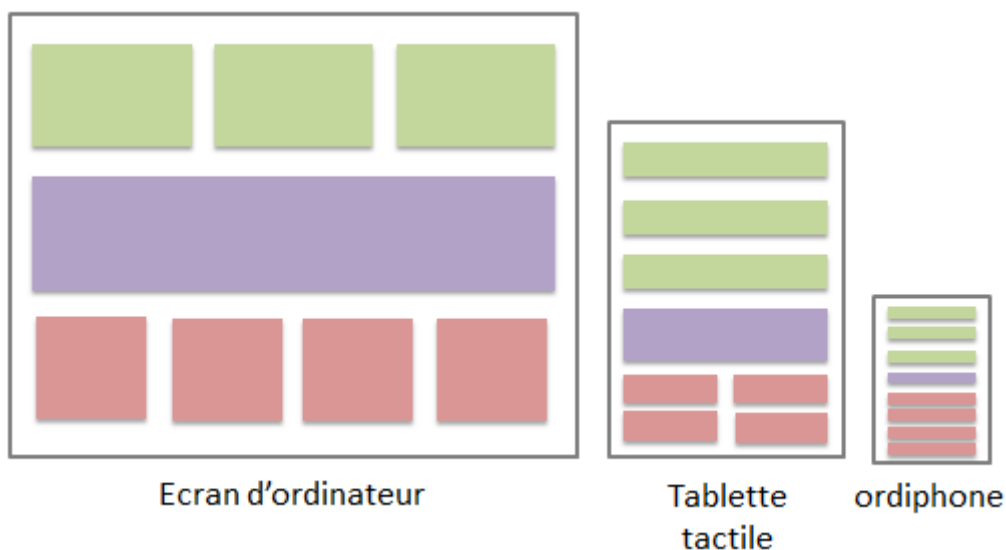


Figure 37 - Agencement d'une page web adaptative sur plusieurs appareils

En conception de sites web adaptatifs, il faut toujours penser à l'ordiphone. Compte tenu de la taille plus réduite de l'écran, le contenu d'une page nécessite d'être épuré afin de ne pas perturber l'utilisateur.

II.3 Outils annexes

II.3.a. Qualité du code

Qu'il s'agisse d'un langage client ou un langage serveur, il est primordial de mesurer la qualité de son propre code. C'est-à-dire s'assurer de fournir une application de qualité et maintenable par d'autres développeurs. Il n'y a rien de plus personnel que du code. Coder en utilisant les standards limite la personnalisation du code et le rend plus maintenable par autrui. L'application s'inscrit ainsi de manière plus pérenne dans le temps.

Par ailleurs, un code structuré, clair et commenté est gage d'économies substantielles au regard des corrections d'erreurs de programmation, comme l'illustre le tableau ci-dessous¹⁸ représentant l'évolution du coût de correction d'une erreur :

		Moment où l'erreur est détectée				
Coût de réparation d'une erreur	Identification des besoins	Identification des besoins	Définition de l'architecture	Développement	Tests	Après déploiement
	Identification des besoins	1×	3×	5–10×	10×	10–100×
Moment où la fonctionnalité est introduite	Définition de l'architecture	–	1×	10×	15×	25–100×
	Développement	–	–	1×	10×	10–25×

En effet, au fur et à mesure des étapes d'un projet, le coût de réparation d'une erreur croît de façon exponentielle. De même, plus une fonctionnalité est ajoutée tardivement dans le projet, plus son coût de réparation sera important.

Cependant, comme le souligne Tom DeMarco : *"You can't control what you can't measure"*¹⁹ dont la traduction littérale est : "vous ne pouvez pas contrôler ce que vous ne mesurez pas". Sans mesure, il est difficile d'avoir un avis sur la qualité du code produit.

C'est là qu'interviennent les outils de mesure de qualité comme PHP_CodeSniffer. Cet outil permet d'auditer son code et de s'assurer de la bonne mise en œuvre des bonnes pratiques du langage et du framework. En effet, le framework Symfony2 possède lui aussi son propre référentiel de bonnes pratiques, elles aussi mesurables.

Les normes PHP les plus connues sont les normes PSR-N, avec N correspondant au niveau de qualité qui s'échelonne de 0 à 7 (7 étant le plus restrictif). PSR signifiant « PHP Standard Recommendation ». Elles sont définies par la communauté « PHP Framework Interoperability Group » (FIG en abrégé).

¹⁸ Traduction du tableau de la page Wikipedia sur les tests logiciels (http://en.wikipedia.org/wiki/Software_testing). Ce tableau est le fruit d'une étude faite par le NIST (*National Institute of Standards and Technology*)

¹⁹ Tom DeMarco, *Controlling Software Projects: Management, Measurement, and Estimates* (1982)

Les recommandations PSR²⁰ sont les suivantes :

- (PSR-0) cette norme qui gérât l'auto-chargement des classes est aujourd'hui dépréciée (au profit de la PSR-4).
- PSR-1 : pose les conditions minimales de codage (ex : les noms des constantes doivent être en majuscule, ...).
- PSR-2 : pose le style d'écriture du code (ex : l'indentation par espace, définition de la forme d'une structure de contrôle, ...).
- PSR-3 : pose les règles pour les interfaces des fichiers journaux. Cette norme est pour les créateurs de frameworks PHP.
- PSR-4 : pose les conditions pour l'auto chargement des classes.
- PSR-7 : pose les conditions pour les interfaces de messages HTTP.

Les normes qui intéressent les développeurs utilisateurs de framework sont les PSR-1 et PSR-2. Les autres ont été créées pour gérer l'interopérabilité de certains frameworks.

Voici un exemple comparatif entre ces deux normes :

Structure de contrôle

```
1 <?php
2 if ($expression) {
3     // code
4 } else {
5     // code
6 }
```

```
1 <?php
2 if ($expression)
3 {
4     // code
5 }
6 else
7 {
8     // code
9 }
```

Validité PSR-1 ? Oui.

Validité PSR-2 ? Oui.

Oui.

Non (il ne faut pas avoir de saut de ligne au niveau des accolades).

L'utilisation de la norme PSR-2 a été adoptée car la PSR-1 a été jugée trop laxiste (notamment au niveau de l'indentation du code).

Pour rendre compatible des parties de codes non compatibles PSR-2, PHP-CS-Fixer a été utilisé. Cet outil, développé par SensioLabs, permet de réparer (en grande partie) les erreurs trouvées par PHP_CodeSniffer. Il peut aussi vérifier et corriger du code qui ne respecte pas les bonnes pratiques du framework Symfony2. Cet outil s'utilise via l'invite de commandes.

Enfin, pour détecter du code mort (fonctions ou variables déclarées mais qui ne seront jamais appelées), l'outil PHP Mess Detector (PHPMD) a été utilisé. Mais comme son nom l'indique, il ne répare pas, il détecte.



**Figure 38 - Logo
de JSHint**

Pour la qualité du code JavaScript, c'est JSHint qui a été utilisé. Parmi la liste des utilisateurs de JSHint, se trouvent de grands noms tels que Mozilla, Facebook et Yahoo!.

A l'inverse de PHP_CodeSniffer, JSHint va plus se

²⁰ <https://github.com/php-fig/fig-standards/tree/master/accepted>

concentrer sur le fond que sur la forme du code (comme le ferait un Environnement de développement intégré (comme Eclipse ou Netbeans). Mais il est aussi possible de paramétrer l'outil pour qu'il scanne le style d'écriture. Cet outil s'utilise lui aussi en ligne de commande.

II.3.b. Génération de la documentation technique

Comme l'imposent les normes PSR, toute fonctionnalité se doit d'être commentée avec des annotations. Avec l'ensemble des commentaires ajoutés, une documentation est générée.

SAMI est un générateur de documentation technique conçu par le créateur de Symfony2, Fabien POTENTIER. Ce générateur parcourt tous les fichiers qui ont l'extension .php et crée une documentation au format HTML à l'aide des commentaires présents dans le code sous forme d'annotation.

The screenshot displays the SAMI-generated documentation for the Symfony2 API. On the left, a sidebar shows a tree view of the project structure, including bundles like CoursusBundle, SiteBundle, StageBundle, and UserBundle, with the 'UserController' class selected under the 'Controller' directory. The main content area shows the details for the 'UserController' class, which extends 'Controller'. It includes a brief description 'User controller.' and a list of methods with their annotations and descriptions:

- `vue indexAction()`: Liste tous les utilisateurs
- `vue createAction(Request $request)`: Affiche le formulaire de création d'une entité User.
- `vue newAction()`: Affiche un formulaire pour créer une nouvelle entité User
- `vue editAction(int $id)`: Affiche le formulaire pour modifier un User existant.
- `vue updateAction(Request $request, int $id)`: Modifie une entité utilisateur existante.
- `Response resetPwdAction(int $id)`: Remplace le mot de passe de l'utilisateur par un nouveau mot de passe.

Figure 39 - Exemple de documentation générée par SAMI

II.3.c. Automatisation des tâches

Dans le développement d'application, il existe de nombreuses tâches redondantes qui, lorsqu'elles sont faites manuellement par le développeur, sont chronophages.

Parmi ces tâches, on retrouve : la minification du code (qui consiste à concaténer tout le code sur une seule ligne pour réduire le poids des fichiers), concaténer des fichiers, vider le contenu d'un répertoire, exécuter plusieurs scripts pour réinstaller la base de données...



**Figure 40 - Logo
de GruntJS**

GruntJS palie ces inconvénients. Fort de ses 3500 plugins, cet outil permet d'effectuer une multitude de tâches via l'interface de commandes.

Ainsi, la minifications de plusieurs fichiers ne se fera que depuis une seule ligne de commande.

Il est aussi possible, avec cet outil, de concaténer plusieurs commandes. On peut alors créer une seule et même commande de déploiement qui effectuera plusieurs actions de manière successives.

Par exemple :

1. Vider le cache ;
2. Vérifier que le code PHP est bien PSR-2 ;
3. Vérifier que le code JavaScript est bien conforme aux règles définies par JSHint ;
4. Générer les fichiers CSS et JavaScript minifiés ;
5. Générer la documentation technique ;
6. Déployer le code sur le serveur.

II.3.d. Gestion des versions

Un logiciel de gestion de versions permet, comme son nom l'indique, de gérer les différentes versions du code source d'un logiciel.

Utiliser un logiciel de gestion de versions offre plusieurs avantages :

1. Il permet, pour un fichier, de revenir à un état précédent ou de réintégrer dans le projet un fichier précédemment supprimé.
2. Les outils de gestion de versions permettent de travailler en mode collaboratif. Les développeurs peuvent alors travailler à plusieurs et de façon simultanée sur la même application et en local (pas sur le serveur, mais sur leurs machines). Néanmoins, s'ils modifient le même fichier en même temps, le gestionnaire alertera le développeur et l'invitera à résoudre le conflit de versions.



**Figure 41 -
Logo de git**

Git fait partie d'eux. Ce logiciel a été conçu par Linus Torvalds, créateur du noyau Linux. Git a d'ailleurs été créé pour le développement de ce noyau.

Git fonctionne avec le principe de branche. Ce mécanisme, permet d'isoler le code source pour travailler sur une partie spécifique du code.

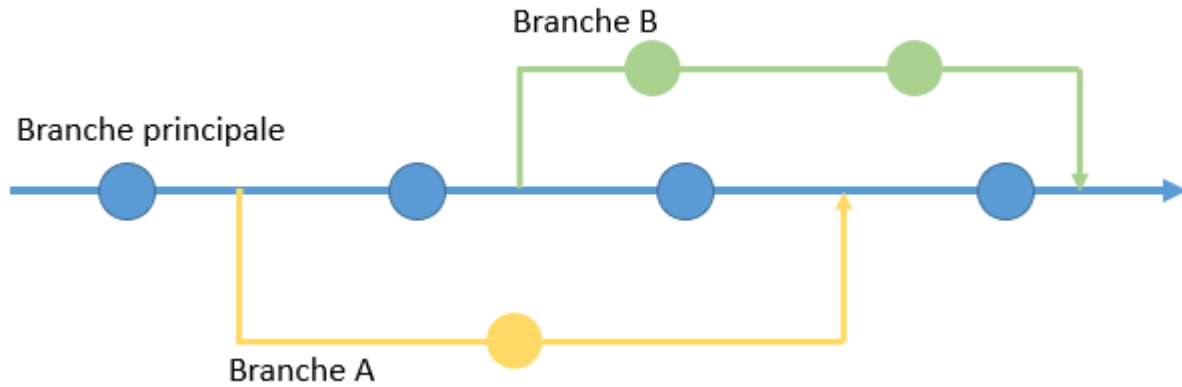


Figure 42 - La gestion des branches par Git

Dans l'exemple ci-dessus, la branche principale (bleue) correspond au code source de l'application en production. A un instant T , on a demandé à un développeur de travailler sur une nouvelle fonctionnalité. Pour ne pas que les développeurs se « marchent sur les pieds », la fonctionnalité a été créée sur une autre branche (Branche A) de telle sorte que le développement ne perturbe pas le bon fonctionnement de l'application. A un autre instant $T+1$, il a été demandé à un autre développeur de travailler sur une autre fonctionnalité. Ce développeur a donc créé une branche B dans la même optique que pour la branche A. A l'issue du développement, le code sera rapatrié sur la branche principale. Il existe énormément de façons d'utiliser ces branches. Le cas ci-dessus n'est qu'un exemple parmi tant d'autres.

Cet outil comporte une multitude de commandes disponibles. Certaines sont à utiliser avec une extrême précaution afin de ne pas perdre du code.

II.3.e. Analyse d'audience internet

Mesurer l'audience d'un site internet sert avant tout à évaluer sa popularité. Les outils d'analyse d'audience internet permettent aussi de capter les tendances et de prendre des décisions sur les prochaines montées de version (privilégier le mobile, continuer d'adapter le site pour certains navigateurs...).

PIWIK

**Figure 43 -
Logo de
Piwik**

Piwik fait partie de ces logiciels. Contrairement à son concurrent direct Google Analytics, il est libre et open source. Ce logiciel s'installe sur un serveur PHP/MySQL et a besoin, pour fonctionner correctement, d'un bout de code JavaScript sur chacune des pages à analyser.

Il permet de :

- Quantifier les visites sur une page donnée à un instant t ;
- Obtenir le type de navigateur ;
- Calculer le Temps moyen passé par utilisateur sur l'application.

Voici un exemple de rapport que peut fournir un outil d'audience internet tel que Piwik :

Rapport

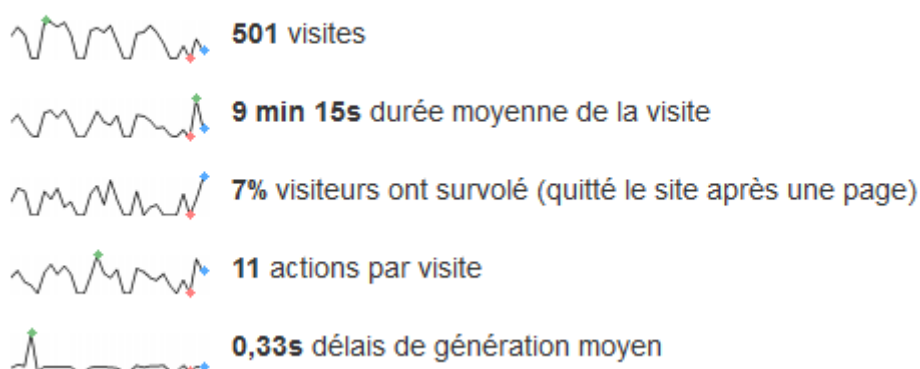


Figure 44 - Rapport d'audience d'un site réalisé avec l'outil Piwik

Contrairement à son concurrent, Piwik permet de ne pas pister les visiteurs. Ce qui l'exempte d'afficher un message tel que celui ci-dessous :²¹

En poursuivant votre navigation sur ce site, vous acceptez l'utilisation de cookies pour réaliser des statistiques de visites anonyme. [En savoir plus.](#)

Figure 45 - Piwik permet de ne pas avoir de bandeau requérant le consentement du visiteur à recevoir des cookies

Cependant ce paramétrage est optionnel et n'est pas actif par défaut. Certains logiciels anti-traçage bloquent cet outil. Par conséquent, ces visites ne seront pas comptabilisées. A l'inverse, les visites de certains robots peuvent être comptabilisés comme celles d'humains.

Une fois les outils sélectionnés, il est maintenant temps de se consacrer à la mise en place de l'application.

²¹ <http://www.cnil.fr/vos-obligations/sites-web-cookies-et-autres-traceurs/>

III Mise en place

L'application est donc au cœur de nombreux échanges applicatifs.

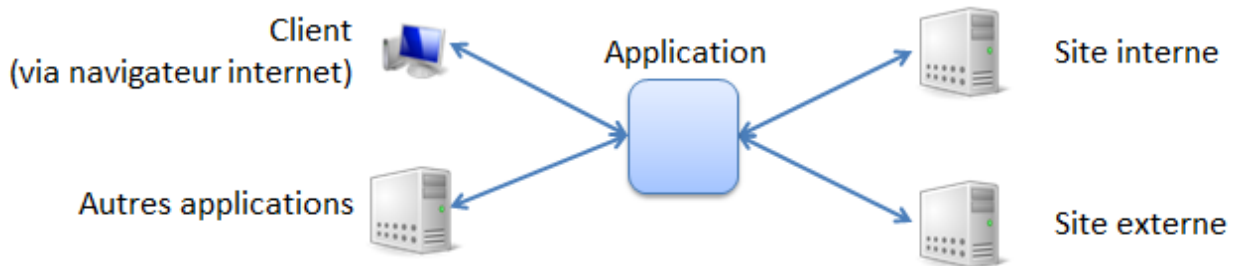


Figure 46 - Une application au centre des échanges

L'objectif de cette partie est d'analyser la nature et le fonctionnement des échanges entre l'application et : le site interne, le site externe et enfin d'éventuelles autres applications.

Le système est à la fois fournisseur de services et consommateur.

III.1 Echange de données avec le site interne

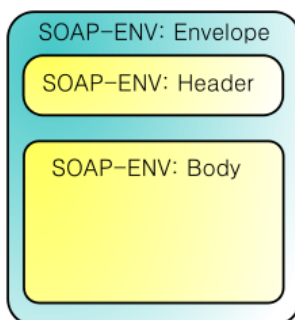
III.1.a. Le protocole SOAP

SOAP permet l'échange d'objets (données) sérialisés entre deux serveurs. Sérialiser un objet, c'est le sauvegarder à un instant t , pour le stocker, transférer ou bien le récupérer ultérieurement.

Par exemple, lors de la sauvegarde d'un document Word, l'objet document est sérialisé. Ce fichier peut alors être transféré sur d'autres applications. A l'ouverture, l'objet se désérialisera pour reprendre sa forme antérieure.

Avec protocole SOAP, les requêtes et les réponses sont sous la forme XML.

Un message SOAP est composé de deux parties :



- La partie Header (entête) : comporte des informations complémentaires pour le traitement des données (identification de l'émetteur du message, règle de sécurité, algorithme de chiffrement, ...)
- La partie Body (corps) : contient les données propres au message.

Figure 47 - Structure SOAP (source : <http://fr.wikipedia.org/wiki/SOAP>)

Voici un exemple permettant de voir à quoi peut ressembler la requête et la réponse d'un service web permettant d'initialiser une connexion. Il s'agit ici de la méthode `InitSession` des Web Services Session :

Requête	<pre><?xml version="1.0" encoding="utf-8"?> <soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"> <soap12:Body> <InitSession xmlns="http://tempuri.org/SiteInterne/Session"> <login>mpoutord</login> <password>UHGFQ9ZUcvDOv7diMjtwCw==</password> </InitSession> </soap12:Body> </soap12:Envelope></pre>
Réponse	<pre><?xml version="1.0" encoding="utf-8"?> <soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"> <soap12:Body> <InitSessionResponse xmlns="http://tempuri.org/SiteInterne/Session"> <InitSessionResult>3d48d8b8-6075-4508-a9c6- 00fd7ac73d69</InitSessionResult> </InitSessionResponse> </soap12:Body> </soap12:Envelope></pre>

Avec SOAP, les entrées et les sorties sont encapsulées dans une enveloppe SOAP. Le plus souvent, le transfert se fait via le protocole HTTP.

Chaque service web est décrit par WSDL (acronyme de *Web Service Description Language*). Comme son nom l'indique, WSDL est chargé de décrire le mode de fonctionnement d'un ou plusieurs services web. Cette description est faite par le langage XML. Comme en témoigne l'annexe 2 (WSDL du service web `initsession` présenté dans l'encadré ci-dessus), la structure de WSDL est particulièrement lourde pour un simple service web. Il est donc rare d'écrire ce fichier à la main, aussi est-il le plus souvent généré.

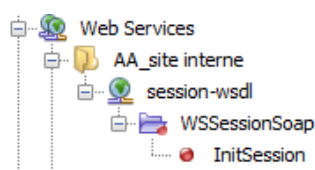


Figure 48 - WSDL est compris par les IDE (exemple avec NetBeans)

Cependant, du fait de leur structure commune, les principaux environnements de développement (IDE) comme Microsoft Visual Studio, Eclipse et Netbeans sont capables de le comprendre.

Il existe aussi des registres UDDI (Universal Description, Discovery and Integration) qui ont pour but de référencer les services web sur le réseau d'une entreprise. Les services web sont alors centralisés au sein du SI.

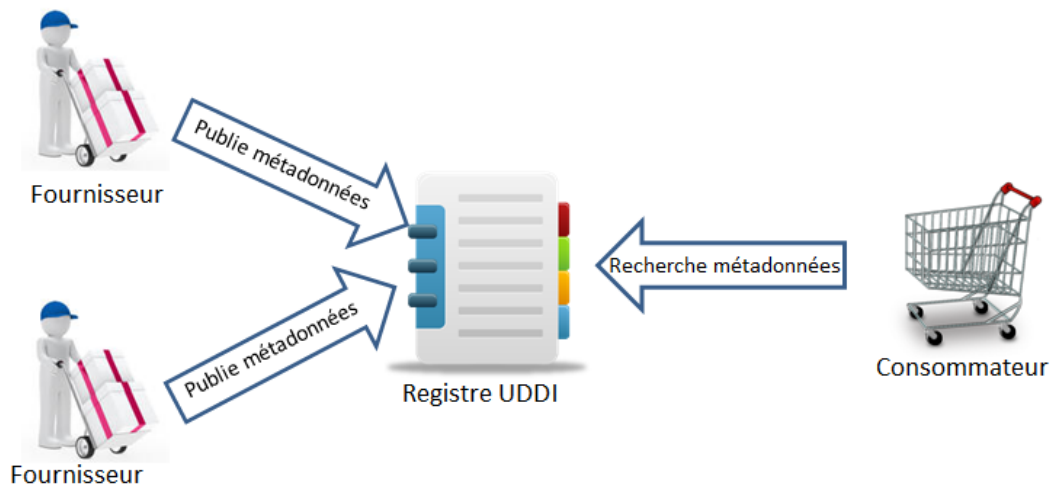


Figure 49 - Utilité d'un registre UDDI

Lorsque le consommateur du service contacte le registre pour un service. Et le registre lui donne le fournisseur (le registre ne donne pas le service, juste son emplacement). Par conséquent, si un fournisseur change, tout le code source des applications consommatrices n'est pas à revoir.

Il est possible de séparer les services en trois catégories : registre d'entreprise (pour l'entreprise), registre partenaire (pour les entreprises partenaires) et registre publique (pour qui le veut).

SOAP et WSDL et UDDI sont normalisés par le W3C²² (World Wide Web Consortium), l'organisme de normalisation du web.

III.1.b. Fonctionnement

Dans le fonctionnement standard de Symfony2, la persistance des objets est assurée par le contrôleur qui va appeler un service doctrine. Doctrine va faire la liaison entre la classe de l'objet et la base de données avec un fichier de mappage.

Pour assurer la persistance sur le site interne, l'idée était de faire quelque chose de semblable de façon à ce que l'utilisation des services web du site interne soit la plus transparente possible.

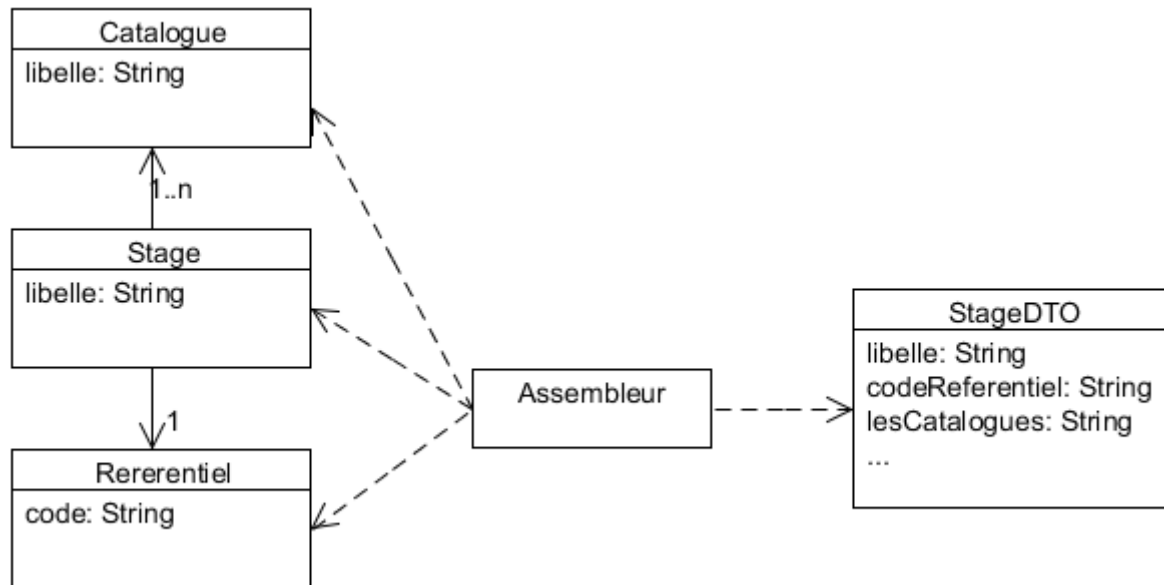
Objets de transfert de données (DTO) :

Les objets qui seront persistés en base ne seront pas les objets Stages de l'application. Ce sera des objets de transfert de données (ou DTO²³ pour Data Transfer Object). L'avantage d'utiliser un objet DTO plutôt que l'objet d'origine c'est qu'il fait un conteneur simple d'utilisation et léger.

De plus, le DTO favorise la séparation entre logique métier et logique de persistance.

²² <http://www.w3.org>

²³ <http://martinfowler.com/eaCatalog/dataTransferObject.html>

Figure 50 - Utilité d'un DTO²⁴

Le diagramme ci-dessus montre bien qu'il est plus simple de manipuler l'objet StageDTO que l'objet Stage, surtout si l'on souhaite accéder au code du référentiel du stage.

La propriété lesCatalogues de l'objet StageDTO est une chaîne de caractères qui correspond à la liste des catalogues dont les éléments sont séparés par une virgule.

Fichier de mappage :

Le fichier de mappage fait la jointure entre la propriété de l'objet StageDTO et le champ du site interne. Ce fichier est écrit en YAML :

```

varInterne:
  NB_JOUR:
    prop: nbJours
    famille: ChampsDynamiques
    libelle: Nombre de jours
  OBJECTIFS:
    prop: objFormation
    famille: ChampsDynamiques
    libelle: objectifs de la formation
    ...
fixeInterne:
  NBJOURLIMINS:
    prop: nbjourlimins
    famille: ChampsDynamiques
    libelle: Délai d'inscription (en j.)
  
```

Ce fichier comporte 2 types de variables :

- Les variables internes : chaque champ du site interne correspond à une propriété d'un objet. Par exemple, le champ du site interne NB_JOUR, correspondra à la propriété nbJours de l'objet DTOStage.
- Les variables fixes : ce sont les variables qui ne sont pas dans l'objet DTOStage mais dans le fichier de paramétrage général de l'application.

²⁴ Inspiré du site : <http://martinfowler.com/eaCatalog/dataTransferObject.html>

Couche d'accès aux données (DAL) :

La couche d'accès aux données ou DAL (pour Database Access Layer) est comme son nom l'indique, une couche qui permet d'accéder aux données en lecture et écriture. C'est cette couche qui appellera les fonctions SOAP.

Il existe un objet DAL par WSDL. Cet objet est constitué d'un client SOAP (classe SOAPClient²⁵ en PHP). Pour ne pas utiliser plusieurs clients SOAP et bien un seul, le patron de conception Singleton a été utilisé.

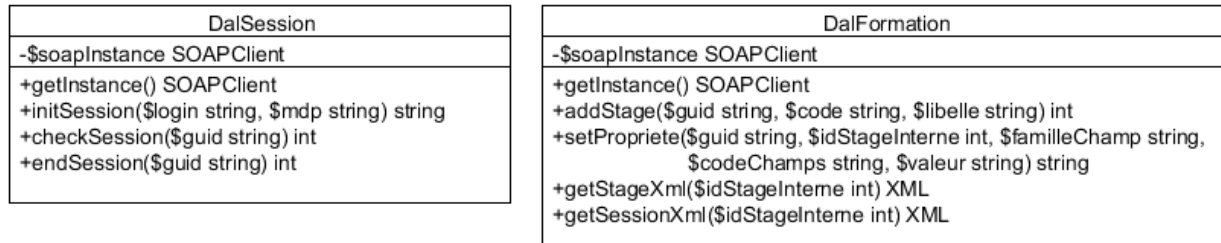


Figure 51 – Diagramme des classes d'accès aux données (DAL) de l'application

Les services :

Tandis que les objets DAL répondent aux besoins de la logique métier de l'application interne, les objets services répondent à ceux de l'application.

Pour les opérations sur les stages (créer et mettre à jour), le ServiceStage convertit les objets Stage en StageDTO et les données de sessions reçues en SessionDTO. Ensuite le service va appeler autant de méthodes de la classe DalFormation que nécessaire.

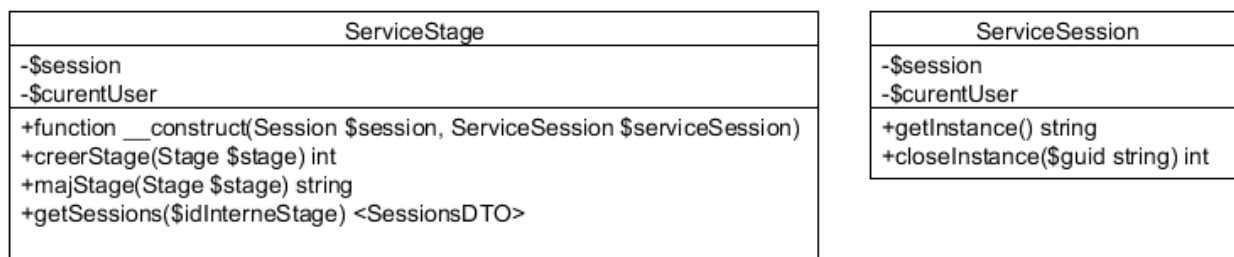


Figure 52 - Diagramme des classes des services de l'application

On peut voir qu'à son instantiation, le service stage embarque une instance du service session. En effet, le service session est injecté dans le service stage. Le contrôleur du stage (StageController) n'a pas à se soucier du service session lorsqu'il utilise le service stage pour accéder ou persister les données.

L'objet Session présent dans le ServiceStage est un élément du framework Symfony2. Cet objet Session permet l'utilisation de messages flash²⁶ pour stocker des messages temporaires dont la durée de vie va jusqu'à la restitution du message. Ainsi, dès qu'une transaction a lieu vers le site interne, l'état est sauvegardé dans un message flash. A la fin de la requête, l'utilisateur accède au message renvoyé pour chacun des services web appelés.

²⁵ <http://php.net/manual/fr/class.soapclient.php>

²⁶ http://symfony.com/fr/doc/current/components/http_foundation/sessions.html#messages-flash

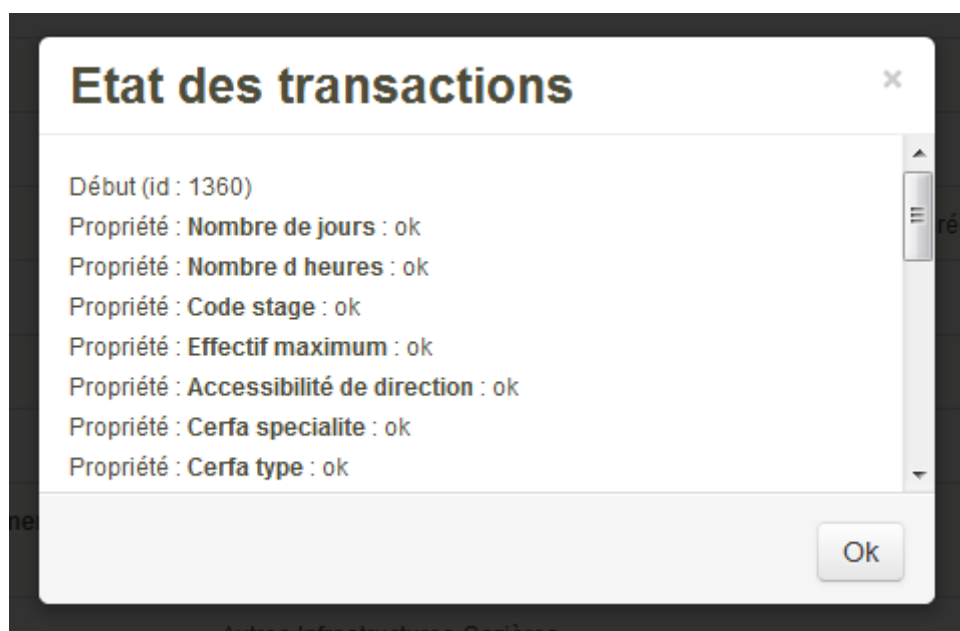
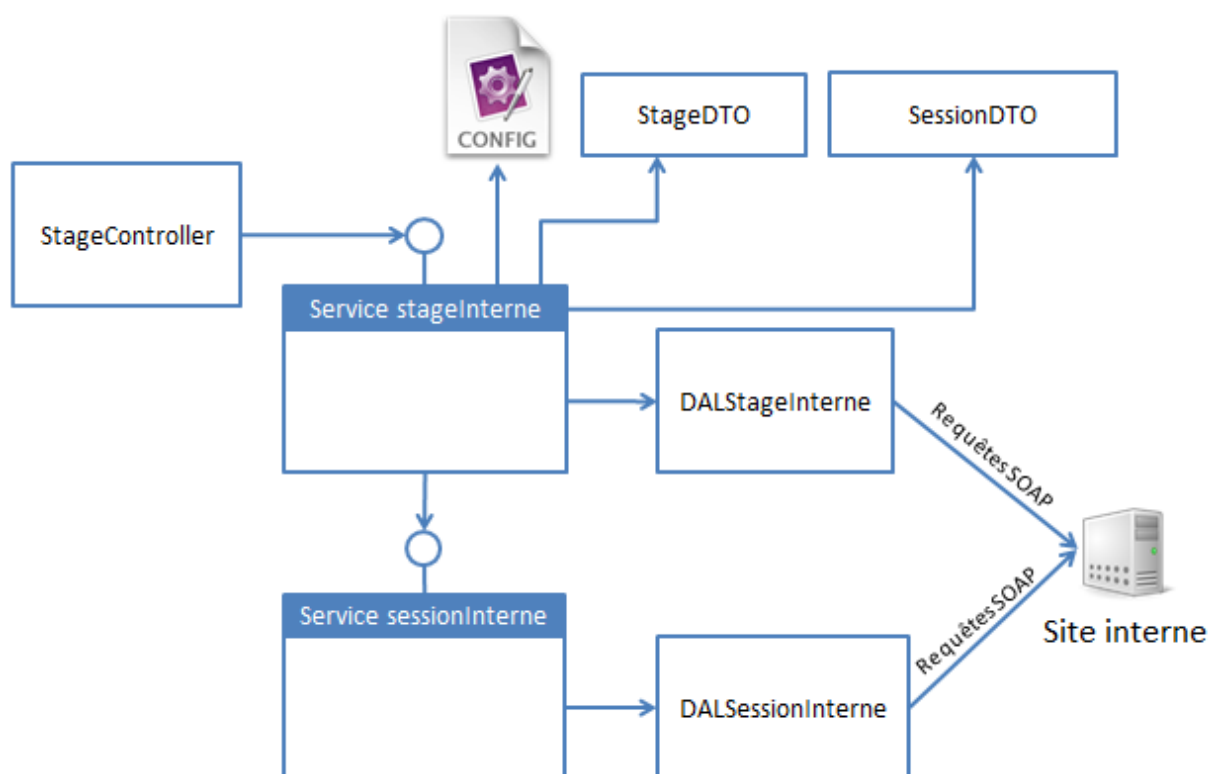


Figure 53 - Les messages flash permettent de stocker de l'information dans divers endroits et moments de la requête pour ensuite la restituer au client

Vue d'ensemble :



Nota : l'objet SessionDTO concerne les sessions au sens session de formation.

Figure 54 - Fonctionnement des services pour le site interne

La partie précédente présentait les avantages des registres UDDI. Cependant, mettre en place un registre UDDI pour si peu de services web serait superflu.

III.1.c. Insertion et récupération

Bien que l'utilisation des services web du site interne soit principalement en écriture, elle en utilise aussi en lecture.

Insertion d'un stage :

Pour une meilleure clarté, ce processus va être scindé en deux : le premier traitera la récupération d'une instance vers les Services Web Session et le second portera sur l'insertion d'un objet Stage dans la base du site interne.

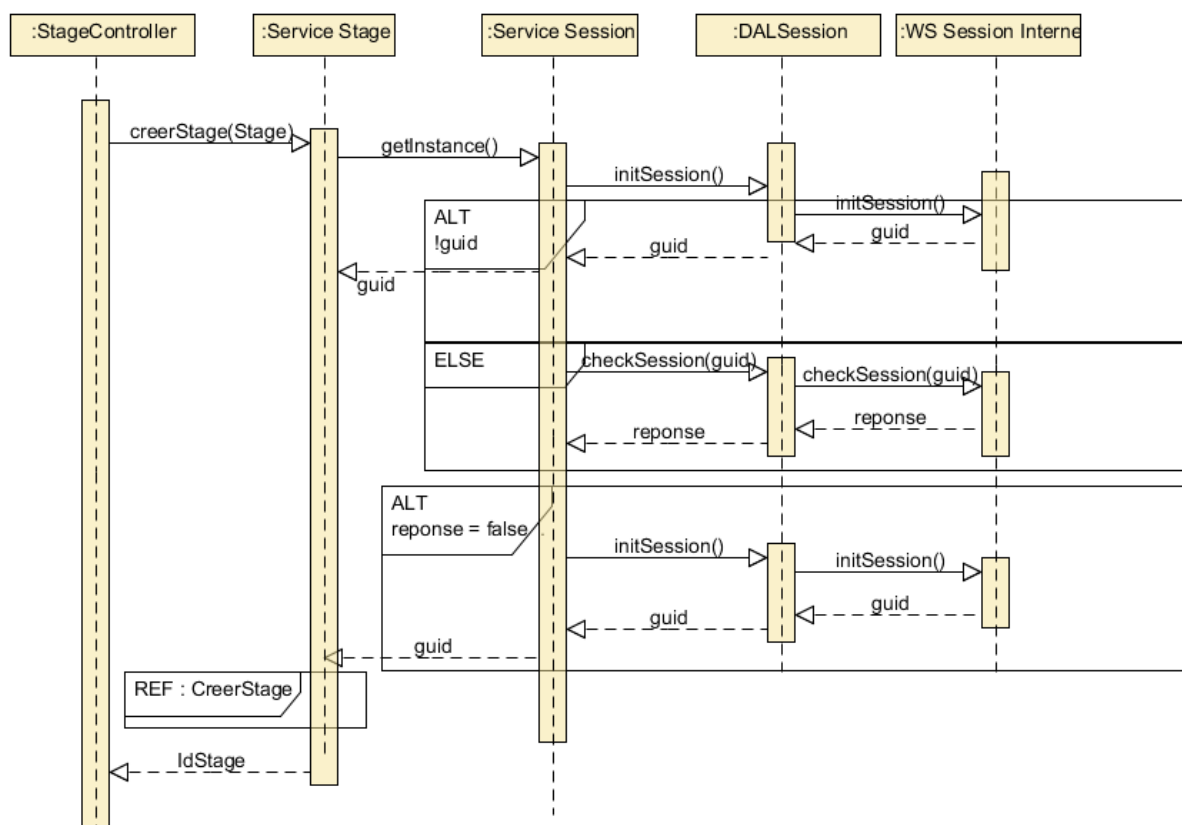


Figure 55 – Processus de récupération d'un jeton de session (guid)

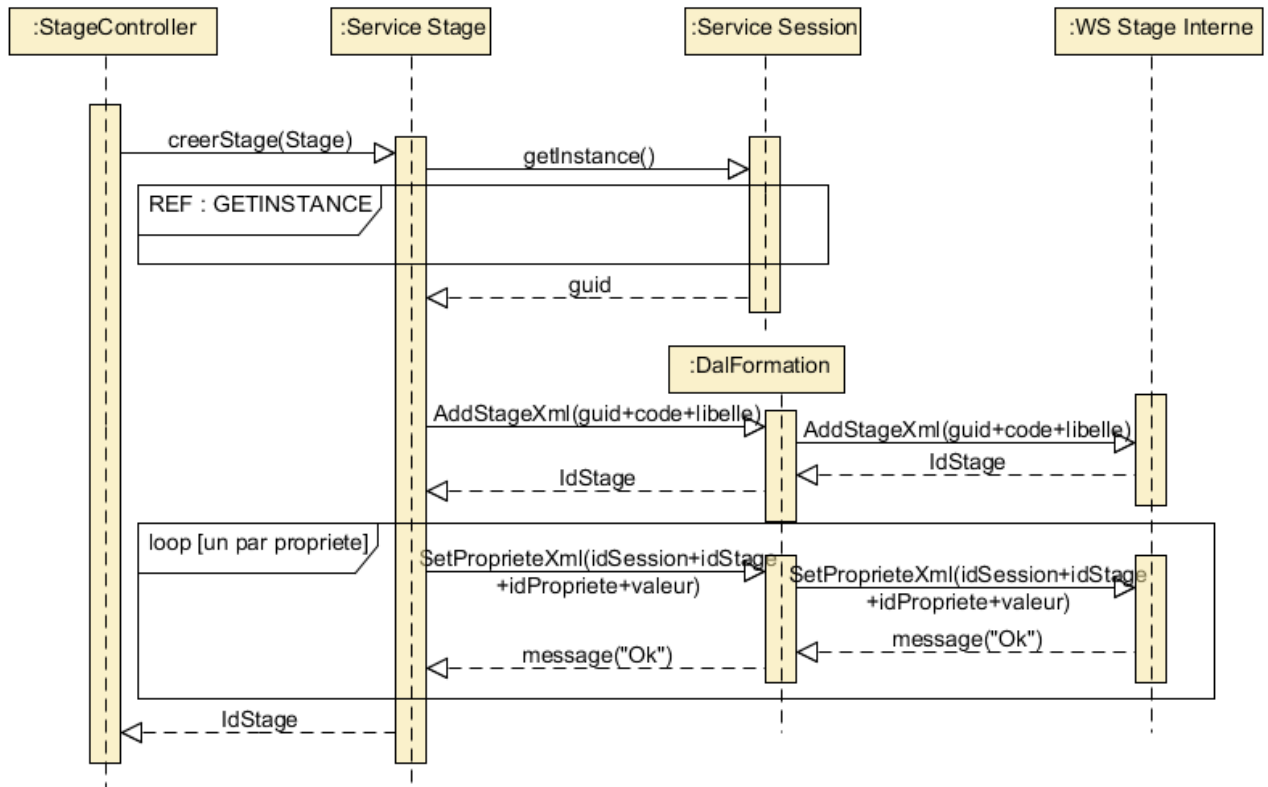


Figure 56 - Processus de création d'un stage en utilisant les services du site interne

Dans le second diagramme, il y a des messages « ok » renvoyés en cas de succès. Ces messages seront ensuite restitués à l'utilisateur sur une fenêtre. De type « popup »

Il faut 37 requêtes nécessaires pour créer un stage :

- 1 pour récupérer un jeton de connexion ;
- 1 pour créer l'enveloppe du stage ;
- 35 pour mettre à jour les propriétés du stage (il y en a 35).

L'ensemble de ces requêtes prennent entre 7 et 9 secondes à s'exécuter (en plus des requêtes SQL pour insérer l'objet dans la base de l'application).

Restitution d'une fiche stage :

La plupart des informations présentes sur la fiche d'un stage sont présentes dans la base de données. Les données relatives aux sessions n'ont pas été délocalisées dans la nouvelle application. Pour y accéder, il faut faire un certain nombre de requêtes, dont le processus est illustré ci-après :

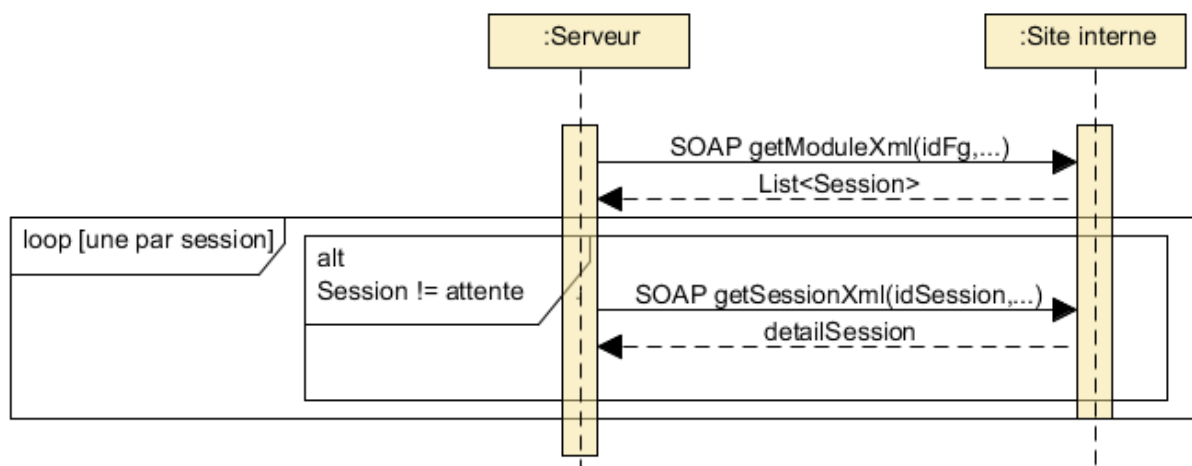


Figure 57 - Processus de récupération des données de sessions de stage

Lors du développement, une question s'est posée : faut-il nécessairement que la fiche soit complète pour la restituer au client ?

En effet, la durée nécessaire pour récupérer les informations sur les sessions dépend du nombre de sessions que possède le stage. Dans la majeure partie des cas, c'est presque instantané (maximum 2 secondes). Cependant, pour les stages plus populaires qui, de facto, possèdent plus de sessions, la durée peut atteindre 10 secondes.

Le choix s'est donc rapidement orienté vers la restitution partielle de la fiche stage. Ensuite, un code JavaScript va chercher de façon asynchrone les sessions. Pour faire patienter l'utilisateur, un GIF animé lui indique que le chargement de la page n'est pas terminé. Une fois le chargement terminé, il disparaît et les sessions sont affichées.

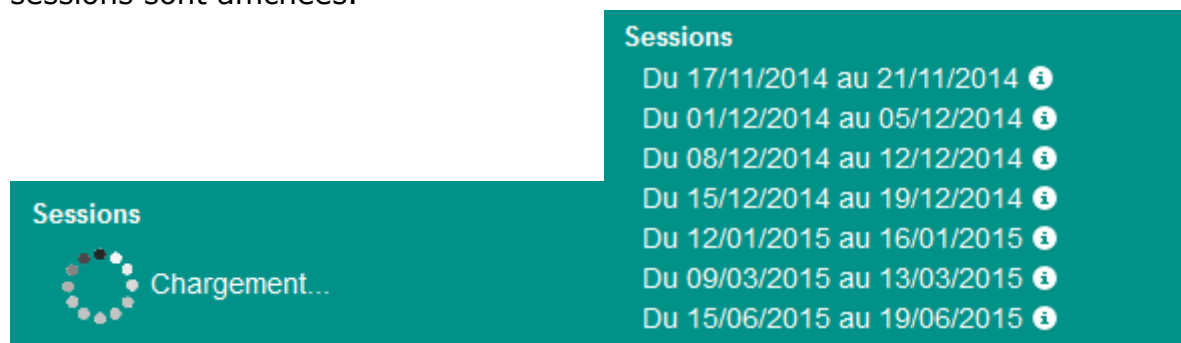


Figure 58 - Un GIF animé en attendant le chargement complet de la page

Dans l'exemple ci-dessus, la requête est particulièrement longue. Si le navigateur met en cache les ressources (images, fichiers CSS et JavaScript), le client mettra 13,9 secondes à charger la totalité de la page.

URL	Statut	Protocole	Poids	Chronologie
GET 7337	200 OK	http	3,6 KB	180ms
GET 603	200 OK	http	1,2 KB	13,25s
GET piwik.php	200 OK	http	43 B	60ms
3 requêtes			4,8 KB	13,9s (onload: 1,19s)

Figure 59 - Chronométrage d'affichage d'une fiche stage (réalisé avec FireBug)

La première requête (GET 7337) correspond au temps de chargement de la page et la seconde (GET 603) correspond à celui des sessions (la troisième correspond au chargement de Piwik, l'outil d'analyse d'audience).

L'avantage de charger la page de façon asynchrone plutôt que de façon synchrone est que le client n'a pas à attendre pour obtenir les informations dont il ne veut peut-être pas. La fonctionnalité qui permet de récupérer les sessions est elle-même un service web de l'application.

Voici comment est restituée une fiche stage au client :

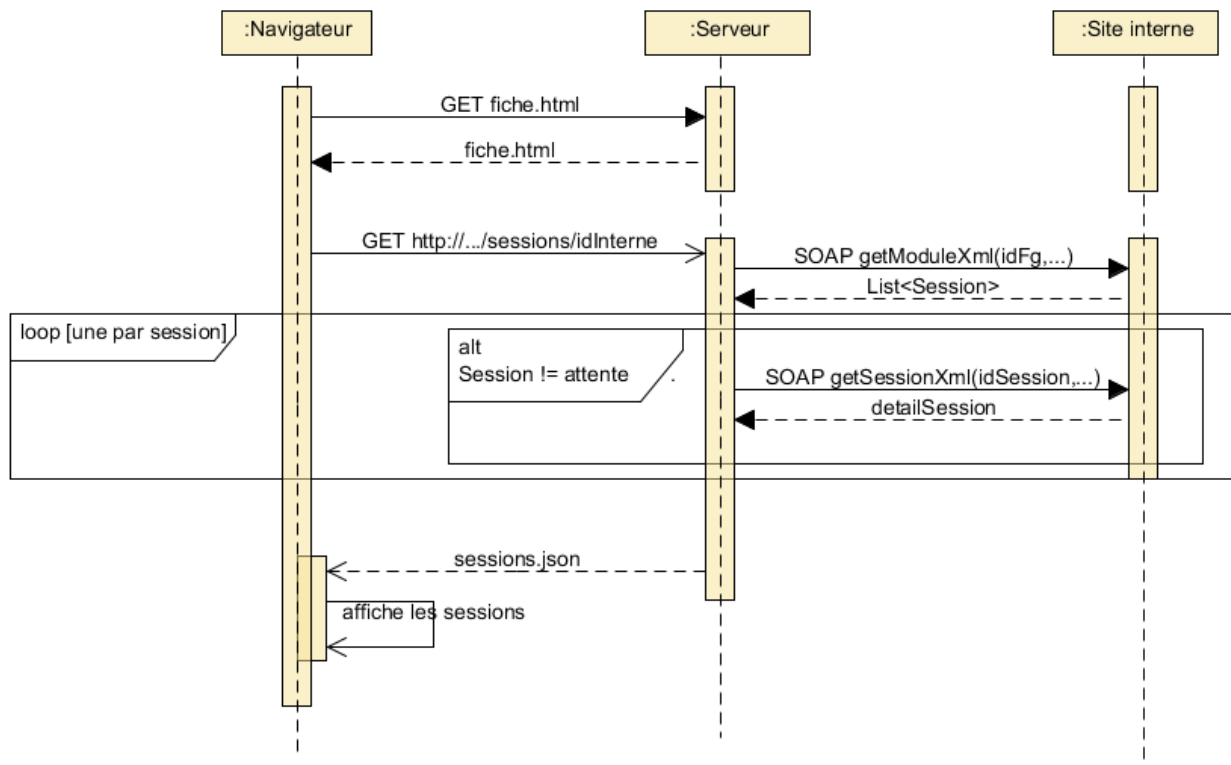


Figure 60 - Restitution d'une fiche de stage

III.2 Echange de données avec le site externe

III.2.a. Génération des classes métiers

Pour créer la couche métier correspondant au site dédié à nos clients externes au groupe, le concept de rétro-ingénierie (*Reverse engineering*) a été utilisé. Ce principe consiste « à étudier un objet pour en déterminer le fonctionnement interne ou la méthode de fabrication. » (Définition de Wikipedia²⁷).

La quasi-totalité de l'application se base sur l'approche *code-first*, c'est-à-dire qu'il faut d'abord créer les objets PHP pour ensuite les persister en base de données. L'approche de rétro-ingénierie fonctionne à l'inverse. On parle aussi d'approche *database first*. Il existe une troisième approche, celle du *model first* qui consiste à construire le modèle avec des outils de modélisation (comme UML), et à générer ensuite à la fois le code PHP et la base de données correspondante.

Ces concepts viennent du framework Entity Framework, équivalent de Doctrine pour les technologies .NET (développé par Microsoft).

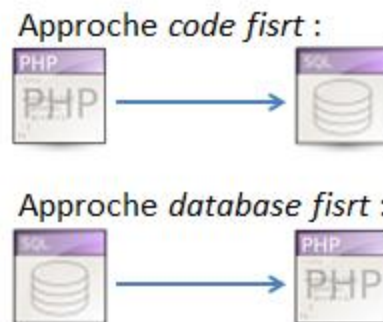


Figure 61 - Comparatif entre l'approche *code first* et l'approche *database first*

Pour générer les classes métier, il a donc fallu utiliser l'approche *database first*. L'extraction SQL contient uniquement les ordres de définition des données, permettant de recréer à l'identique la structure des tables.

Après avoir recréé cette base dans l'environnement de tests, il a fallu demander à Doctrine d'inspecter la base de données et de générer des fichiers de métadonnées pour chaque table présente.

²⁷ <http://fr.wikipedia.org/wiki/Rétroingénierie>

Ci-dessous, un exemple de fichier de métadonnées généré par Doctrine :

```
<?xml version="1.0" encoding="utf-8"?>
<doctrine-mapping>
  <entity name="AfStage" table="af_stage">
    <id name="id" type="int" column="id">
      <generator strategy="IDENTITY"/>
    </id>
    <field name="title" type="string" column="title" length="255"/>
    <field name="code" type="string" column="title" length="255"/>
    <field name="contentPrerequis" type="text" column="content_prerequis"/>
    ...
    <field name="updateDate" type="datetime" column="update_date"/>
    <lifecycle-callbacks/>
  </entity>
</doctrine-mapping>
```

C'est ensuite à partir de ces fichiers que Doctrine va se baser pour générer les classes du modèle correspondant.

Il faudra ensuite exécuter une nouvelle commande pour générer les accesseurs et mutateurs (méthodes get/set d'une propriété). L'entité générée est maintenant fin prête à l'emploi avec Doctrine.

Cette conception pose un problème : si le modèle de la base de données évolue, toutes les modifications apportées à l'entité générée seront écrasées et donc supprimées. Comme chaque système a son propre modèle d'objet d'un stage, il faut créer une classe de conversion.

III.2.b. Classe de conversion

En imaginant que l'objet Stage de l'application est un rond et que celui du site externe un triangle, l'objectif de la classe de conversion sera de transformer un rond en triangle pour pouvoir ensuite le persister en base de données. Car pour effectuer la sauvegarde, la base de données attend un triangle et non un rond.

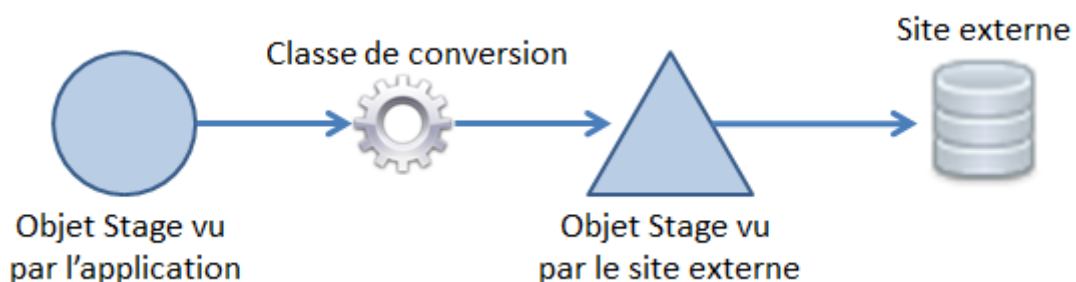


Figure 62 - Utilité d'une classe de conversion

La figure ci-dessus représente l'intérêt d'une telle classe. Le triangle est ni plus ni moins que la classe précédemment générée. La flèche, quant à elle, est à sens unique car l'application a juste besoin d'insérer les données en base.

Ensuite, la persistance de cet objet en base se fait comme les autres entités de l'application : avec Doctrine.

La méthode de conversion aurait pu être ajoutée dans l'objet stage généré par doctrine lors de la rétroingénierie. Cependant, à chaque fois que la structure de la base de données du site externe évoluera, cette méthode sera supprimée (les classes générées par Doctrine écrasent les existantes). Il faudra ensuite rechercher la classe dans le logiciel de gestion de versions pour la réintégrer.

III.3 Récupération des données de l'application par des applications tierces

Il existe deux types de services web :

- les services de type WS-* présentés précédemment, qui reposent sur les standards SOAP et WSDL ;
- les services REST qui reposent sur les standards du web : HTTP et URI.

Comme le montre cette analyse de tendances de recherche du moteur de recherche Google, REST est très en vogue, et ce, depuis quelques années.

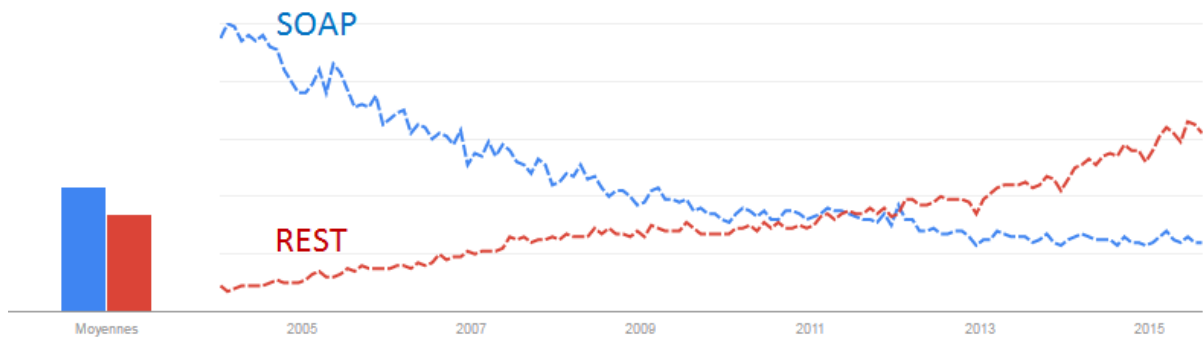


Figure 63 - Analyse de tendance sur le moteur de recherche Google (Google Trend) entre les deux types de service web SOAP et REST²⁸

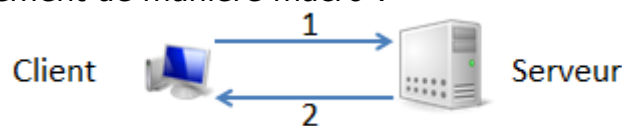
En effet, REST doit sa popularité à sa simplicité d'utilisation et de mise en œuvre.

L'application développée a été construite avec un style très proche de REST. C'est donc cette architecture qui a été choisie pour permettre à des applications tierces de partager ses données. La meilleure implémentation de REST étant HTTP, il est nécessaire de le présenter au préalable.

III.3.a. Le protocole HTTP

Le protocole HTTP (pour HyperText Transfer Protocol) est un protocole créé en 1996 en version 1 (HTTP/1.0). La version actuelle est 1.1 (HTTP/1.1). Il est très rare dans le monde du web qu'un protocole évolue si peu. S'il en est ainsi, c'est qu'il a été bien conçu dans sa version initiale et qu'il est simple.

Voici son fonctionnement de manière macro :



1. Le client demande la page

²⁸<https://www.google.fr/trends/explore#q=%2Fm%2F077dn%2C%20%2Fm%2F03nsxd&cmpt=q&tz=Etc%2FGMT-2>

```
GET /mapage.html HTTP/1.1
Host: monapplication.com
Accept: text/html
User-Agent: Mozilla/5.0
```

2. Le serveur envoie au client la ressource demandée (qui peut être générée)

```
HTTP/1.1 200 OK
Date: Sat, 11 Apr 2014 21:05:05 GMT
Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
Content-Type: text/html

<html>
  <!--contenu de mapage.html -->
</html>
```

Le mot GET présent dans la première requête est une *méthode* HTTP. Ces méthodes définissent ce que le client souhaite faire de la ressource. On parle aussi de verbes.

Voici les principales méthodes HTTP :

Méthode HTTP	Description
GET	Récupère la ressource depuis le serveur
POST	Crée une ressource sur le serveur
PUT	Met à jour la ressource (complète) sur le serveur
PATCH	Met à jour une partie d'une ressource en envoyant un différentiel
DELETE	Supprime la ressource sur le serveur
HEAD	Récupère les métadonnées de la ressource

La RFC7231²⁹ a ajouté deux notions aux méthodes HTTP :

- Les méthodes dites *sûres* ne modifient pas les données sur le serveur. Peu importe le nombre de fois qu'elles sont appelées.
- Les méthodes *idempotentes* quant à elles peuvent modifier les données lors du premier appel. Lors des suivants, la réponse sera tout le temps identique. En cas de panne/problème réseau, il est donc possible de relancer la requête sans crainte.

Tableau récapitulatif de la sûreté et de l'idempotence des méthodes :

Méthodes	Sûre ?	Idempotentes ?
GET	Oui	Oui
HEAD	Oui	Oui
POST	Non	Non
PUT	Non	Oui
PATCH	Non	Non
DELETE	Non	Oui

²⁹ <http://tools.ietf.org/html/rfc7231#section-4.2>

La mise en cache est l'une des contraintes les plus importantes des architectures web. En effet, elle permet de :

- décharger le serveur en lui évitant de réitérer la même action,
- réduire le temps de traitement d'une requête,
- ...et donc de renvoyer plus rapidement l'information au client.

Les méthodes dites *sûres* sont donc à mettre en cache.

Il est primordial d'utiliser ces méthodes à bon escient. Si par exemple, il est possible de supprimer un utilisateur via la méthode GET en effectuant une requête comme celle-ci :

```
GET api.mon-application.org/utilisateurs/supprimer?id=42
```

... le cache ne sera pas forcément actualisé si un utilisateur souhaite accéder à la ressource correspondant à l'identifiant 42 ; alors que ce serait le cas si la méthode DELETE était appelée.

En réponse, le serveur renvoie les informations demandées avec un code correspondant à un statut. On parle de code HTTP³⁰.

Voici les plus répandus, classés par famille :

Famille	Description	Exemples
1xx	Information	100 Continuer
2xx	Succès	200 OK. Requête traitée avec succès 201 Création du document réussie
3xx	Redirection	301 Ressource déplacée de façon permanente
4xx	Erreur d'origine client	403 Accès interdit 404 Ressource non trouvée
5xx	Erreur d'origine serveur	500 Erreur interne

III.3.b. L'architecture REST

REST (pour REpresentational State Transfer) n'est pas un protocole mais bien un style d'architecture. REST a été créé par Roy T. Fielding pour sa thèse de doctorat³¹. Ce style d'architecture impose de ne plus penser « page web » mais ressource.

Ainsi, le client n'accède plus à des pages web mais à des **ressources**. Par exemple, lorsque l'on tape dans barre d'URL du navigateur :

➤ <http://monapplication.com/stages/1>

Et la requête suivante va partir vers le serveur :

```
GET /stages/1 HTTP/1.1
Host: monapplication.com
Accept: text/html
```

³⁰ <http://tools.ietf.org/html/rfc7231>

³¹ <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

User-Agent: Mozilla/5.0

Le client accèdera à la ressource stage qui a pour identifiant 1 au format HTML. REST diverge d'un site classique, car si l'utilisateur avait envoyé cette requête :

```
GET /stages/1 HTTP/1.1
Host: monapplication.com
Accept: application/xml
User-Agent: Mozilla/5.0
```

Le client aurait reçu de la part du serveur, la ressource stage ayant pour identifiant 1 au format XML.

Pour ajouter une ressource sur le serveur, la requête ne serait plus de type GET mais POST et celle-ci comporterait des paramètres (les variables renseignées pour créer une ressource stage).

```
POST /stages/ HTTP/1.1
Host: monapplication.com
Accept: application/xml
User-Agent: Mozilla/5.0
```

```
libelleStage=Soudage
codeStage=1234
```

En cas de réussite, le

REST est donc un style d'architecture entièrement basé sur HTTP (méthodes, code d'erreur,...). L'approche REST consiste à accéder à des ressources via des URI (Uniform Resource Identifier, soit en français identifiant uniforme de ressource).

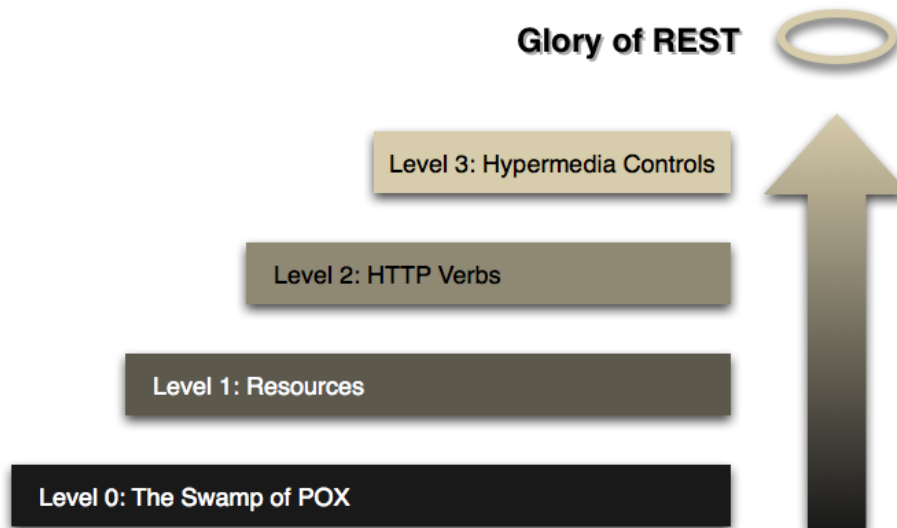
La question que soulève REST est la suivante : quelle est la différence entre une API HTTP et une API RESTful ?

III.3.c. De l'API HTTP à l'API RESTful : Les 4 niveaux du modèle de maturité de Richardson

Les schémas et exemples de cette partie proviennent ou sont très largement inspirés de l'article³² de Martin Fowler sur le Modèle de Maturité de Richardson.

En se basant sur le travail de Roy Fielding, Leonard Richardson a établi un modèle de maturité des services web REST appelé Richardson Maturity Model (RMM). Ces 4 niveaux permettent d'évaluer une API (Interface de programmation) par rapport aux contraintes REST.

³² <http://martinfowler.com/articles/richardsonMaturityModel.html>

Figure 64 - Les étapes vers la gloire de REST³³

Les niveaux vont du niveau 0 au niveau 3. Le niveau 0 étant le moins conforme à l'approche REST, et le 3 étant le niveau le plus haut niveau de conformité. On parlera alors d'une API RESTful.

Le niveau 0 : le marais des POX (« bons vieux XML »)

Le niveau 0, utilise le protocole HTTP uniquement à des fins de transport. Cependant, les réponses du serveur n'utilisent pas pleinement HTTP (comme par exemple les codes réponse HTTP, les méthodes HTTP, ...).

Les protocoles SOAP (les services web du site interne) et XML-RPC (l'ancêtre de SOAP) utilisent HTTP uniquement à des fins de transport du message. En tentant d'accéder à un service web auquel l'utilisateur n'a pas le droit, la réponse ne sera pas accompagnée d'un code d'erreur 403 : accès interdit.

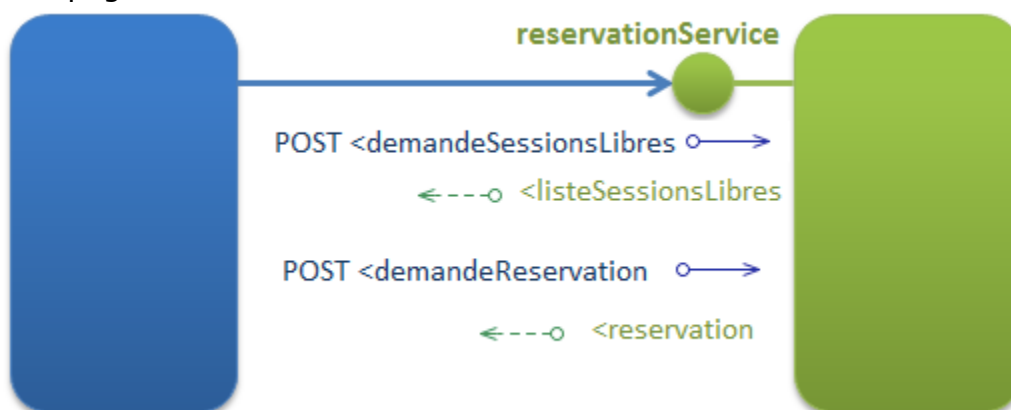


Figure 65 - Le niveau 0 du modèle de Richardson

Pour illustration, voici ce que pourrait donner la réservation d'un stage qu'un manager souhaite faire pour un de ses employés. Dans un premier temps, le manager va consulter la liste des sessions pour un stage donné, puis il va inscrire son salarié à la session souhaitée. Les informations échangées sont sous

³³ Source : <http://martinfowler.com/articles/richardsonMaturityModel.html>

la forme XML. Cependant, le format n'est pas figé, il peut aussi se trouver sous la forme JSON, YAML, CSON...

Première étape : le manager demande une place disponible pour le stage soudage au mois de juin 2014 :

```
POST /reservationService HTTP/1.1
```

```
<demandeSessionsLibres date="06-2014" stage="soudage"/>
```

En réponse, le serveur va renvoyer la liste des sessions libres pour la date demandée :

```
HTTP/1.1 200 OK
```

```
<listeSessionsLibres>
  <session du="09-06-2014" au="13-06-2014" placesLibres="2">
    <stage id="soudage"/>
  </session>
  <session du="23-06-2014" au="27-06-2014" placesLibres="1">
    <stage id="soudage"/>
  </session>
</listeSessionsLibres>
```

Le manager réserve ensuite, via son application, un emplacement sur la 2^e session pour le stagiaire johnDoe.

```
POST /reservationService HTTP/1.1
```

```
<demandeReservation>
  <session stage="soudage" du="23-06-2014" au="27-06-2014">
    <stagiaire id="johnDoe"/>
  </session>
</demandeReservation>
```

En cas de réussite, le serveur retourne un message pour signifier que la réservation est bien faite.

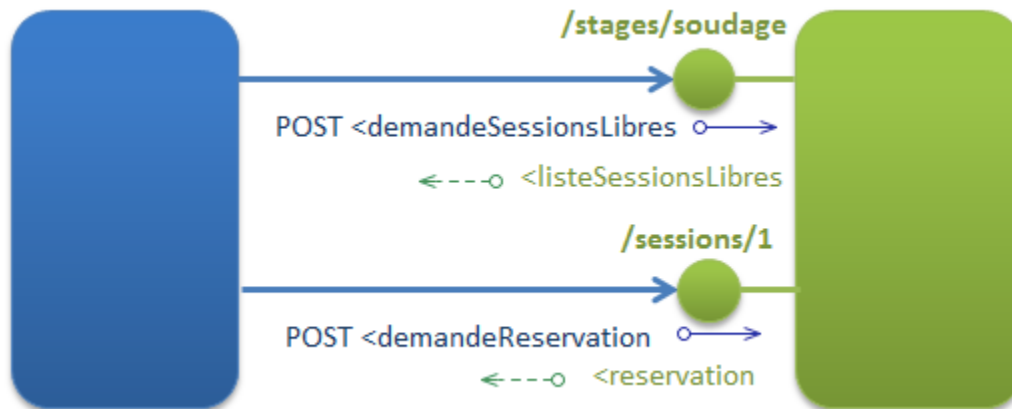
```
HTTP/1.1 200 OK
```

```
<reservation>
  <session stage="soudage" du="23-06-2014" au="27-06-2014">
    <stagiaire id="johnDoe"/>
  </session>
</reservation>
```

En cas d'échec, le serveur renvoie un message avec le motif.

```
HTTP/1.1 200 OK
```

```
<demandeReservationEchec>
  <session stage="soudage" du="23-06-2014" au="27-06-2014">
    <stagiaire id="johnDoe"/>
    <raison>Session complète.</raison>
  </session>
</demandeReservationEchec>
```

Le niveau 1 : ressources**Figure 66 - Le niveau 1 du modèle de Richardson**

La deuxième étape vers la « *gloire de REST* » (*Glory of REST*), est l'introduction d'un élément fondamental de REST : la notion de ressources.

Pour continuer dans l'exemple précédent, il n'y a plus d'URI du type : "/reservationService" qui contient tous les services de réservation mais bien une url par ressource.

L'application va interroger le stage soudage et l'interroger sur les sessions libres.

```
POST /stages/soudage HTTP/1.1
```

```
<demandeSessionsLibres date="06-2014"/>
```

Dans la réponse du serveur, un identifiant unique est placé sur chaque ressource. C'est l'attribut id.

```
HTTP/1.1 200 OK
```

```
<listeSessionsLibres>
  <session id="1" du="09-06-2014" au="13-06-2014" placesLibres="2"
    stage="soudage" />
  <session id="2" du="23-06-2014" au="27-06-2014" placesLibres="1"
    stage="soudage" />
</listeSessionsLibres>
```

Pour faire la demande de réservation, il faut maintenant contacter la ressource session ayant pour identifiant (id) 2.

```
POST /sessions/1 HTTP/1.1
```

```
<demandeReservation>
  <stagiaire id="johnDoe"/>
</demandeReservation>
```

Tout comme pour le niveau 0, le message de retour confirme la réservation.

```
HTTP/1.1 200 OK
```

```
<reservation>
  <session id="2" stage="soudage" du="23-06-2014" au="27-06-2014">
    <stagiaire id="johnDoe"/>
  </reservation>
```

Nota : En cas d'échec, le même message que celui de niveau 0 est fourni.

Le niveau 2 : méthodes HTTP

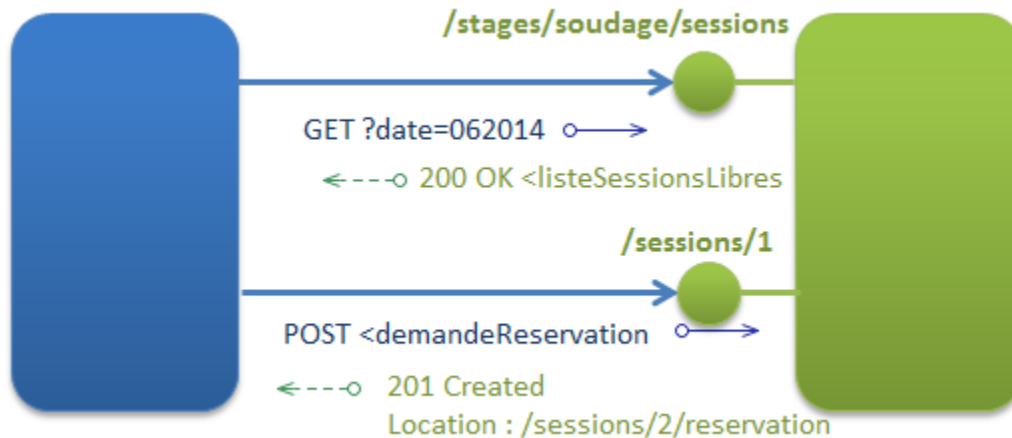


Figure 67 - Le niveau 2 du modèle de Richardson

La 3^e étape inclut la notion de méthodes HTTP et de codes.

Pour demander une ressource (en l'occurrence, une liste de sessions), la méthode HTTP adapté est GET. Pour filtrer les résultats, tout se fait via l'URI.

```
GET /stages/soudage/sessions?date=062014 HTTP/1.1
Host: monapplication.com
```

Comme pour le niveau 1, le serveur renvoie la liste des sessions.

```
HTTP/1.1 200 OK
```

```
<listeSessionsLibres>
  <session id="1" du="09-06-2014" au="13-06-2014" placesLibres="2"
    stage="soudage" />
  <session id="2" du="23-06-2014" au="27-06-2014" placesLibres="1"
    stage="soudage" />
</listeSessionsLibres>
```

Pour réserver, il faut ajouter une demande de réservation dans la ressource session 1. La méthode HTTP correspondant à l'ajout d'une ressource est POST. Si le manager souhaitait supprimer la réservation de John Doe, la requête aurait comporté la méthode HTTP DELETE.


```
POST /sessions/2 HTTP/1.1
```

```
<demandeReservation>
  <stagiaire id="johnDoe"/>
</demandeReservation>
```

Pour indiquer qu'une ressource (une réservation) a bien été faite, le serveur renvoie un code HTTP 201, indiquant que la requête a été traitée avec succès et qu'une ressource a été créée.

```
HTTP/1.1 201 Created
Location: /sessions/2/reservation
```

```
<reservation>
  <session id="2" stage="soudage" du="23-06-2014" au="27-06-2014">
    <stagiaire id="johnDoe"/>
</reservation>
```

En cas de conflit, le serveur renvoie un code 409 : conflit. Ce code signifie que la requête ne peut être traitée dans l'état actuel.

```
HTTP/1.1 409 Conflict
```

```
<reservation>
  <session id="2" stage="soudage" du="23-06-2014" au="27-06-2014">
</reservation>
```

Le niveau 3 : contrôles hypermédias

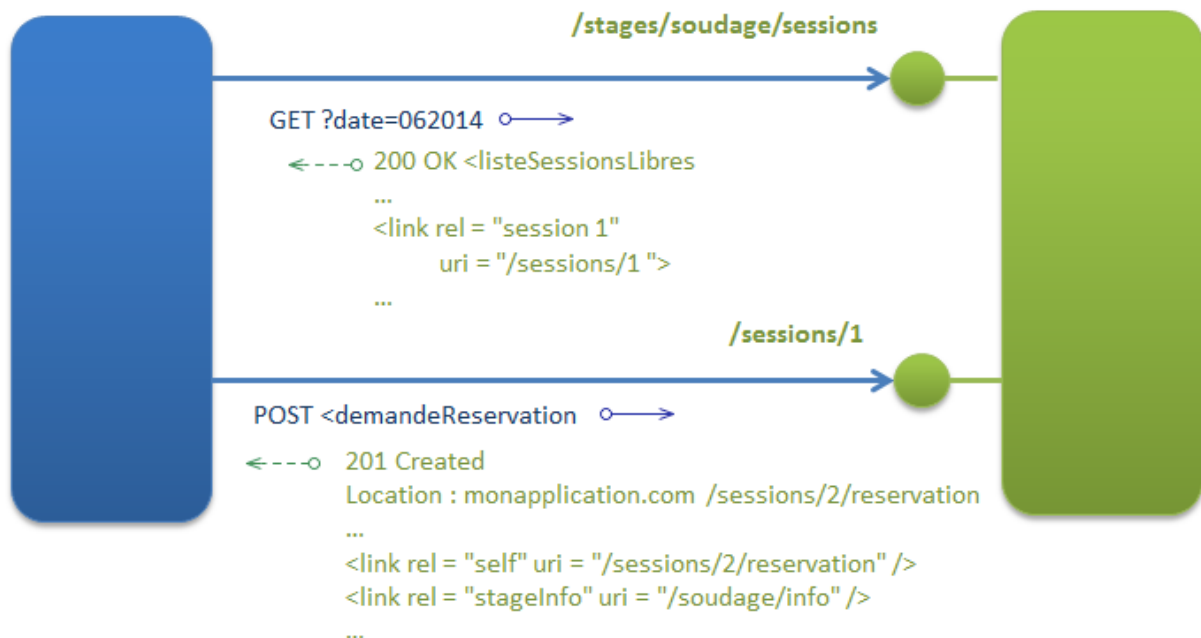


Figure 68 - Le niveau 3 du modèle de Richardson

Le dernier et plus haut niveau du modèle de Richardson introduit la notion de contrôles hypermédias. Concrètement, « un hypermédia est une extension de l'hypertexte à des données multimédia » (définition de Wikipedia³⁴).

Ce 4^e niveau est aussi connu sous l'acronyme anglo-saxon HATEOAS (*Hypertext As The Engine Of Application State* soit en français hypertext comme le moteur de l'état de l'application). Dans chaque réponse, la ressource doit indiquer la liste des actions possibles par le biais d'URI.

L'utilisation de contrôles hypermédias permet, en partie, d'auto-documenter l'API.

Pour accéder aux sessions, rien ne change depuis le niveau 2.

```
GET /stages/soudage/sessions?date=062014 HTTP/1.1
Host: monapplication.com
```

Le serveur retourne presque les mêmes informations, à l'exception d'une balise <link> (lien). Chaque lien permet d'accéder à la session en question. Désormais, le client n'aura plus à sa charge la construction d'URI.

```
HTTP/1.1 200 OK

<listeSessionsLibres>
  <session id="1" du="09-06-2014" au="13-06-2014" placesLibres="2"
    stage="soudage">
    <link rel = "session 1"
      uri = "/sessions/1"/>
    </session>
  <session id="2" du="23-06-2014" au="27-06-2014" placesLibres="1"
    stage="soudage" >
    <link rel = "session 2"
      uri = "/sessions/2"/>
    </session>
</listeSessionsLibres>
```

Comme pour les niveaux 1 et 2, l'ajout d'une réservation se fait pareillement.

```
POST /sessions/2 HTTP/1.1

<demandeReservation>
  <stagiaire id="johnDoe"/>
</demandeReservation>
```

Le serveur retourne presque le même message qu'au niveau 2, à l'exception de liens permettant au client plusieurs actions une fois la demande de réservation effectuée (obtenir des informations sur le stage, contacter l'organisme de formation et accéder aux informations du stagiaire).

³⁴ Définition de Wikipedia : <http://fr.wikipedia.org/wiki/Hypermédia>

```

HTTP/1.1 201 Created
Location: /sessions/2/reservation

<reservation>
  <session id="2" stage="soudage" du="23-06-2014" au="27-06-2014">
    <stagiaire id="johnDoe"/>

    <link rel = "self" uri = "/sessions/2/reservation" />
    <link rel = "stageInfo" uri = "stages/soudage/info" />
    <link rel = "contactInfo" uri = "/contact" />
    <link rel = "infoStagiaire" uri = "/stagiaire/johnDoe/info"/>
  </session>
</reservation>

```

Ainsi, le client peut naviguer avec les données fournies par le serveur dans sa réponse. Dans les niveaux 1 et 2 du modèle de Richardson, le client doit savoir qu'il faut concaténer l'URI "/sessions/" avec son identifiant pour pouvoir y accéder.

En plus d'informer le client des URI, ce 3^e niveau informe le client de ce qu'il peut et ne peut pas faire. Pour illustration, voici un exemple sensiblement différent de ce qui a été donné précédemment :

```

HTTP/1.1 200 OK

<listeSessions>
  <session id="1" du="09-06-2014" au="13-06-2014" placesLibres="2"
    stage="soudage">
    <link rel = "Session 1 - réserver un emplacement"
      uri = "/sessions/1/reservation"/>
  </session>
  <session id="2" du="16-06-2014" au="21-06-2014" placesLibres="0"
    stage="soudage" >
  </session>
  <session id="3" du="23-06-2014" au="27-06-2014" placesLibres="1"
    stage="soudage" >
    <link rel = "Session 3 - réserver un emplacement"
      uri = "/sessions/3/reservation"/>
  </session>
</listeSessions>

```

La session n°2 (id=2) n'ayant plus de place disponible, l'interface de programmation (API) ne proposera pas au client de réserver.

III.3.d. La place des services web de l'application dans le modèle de Richardson

Ce qui a été fait :

Symfony2 utilise le principe de réécriture d'URL. C'est ce principe qui fait que l'on ne voit jamais d'URL de la sorte :

- monapplication.com/index.php?stage=1

Mais plutôt :

- monapplication.com/stages/1

Les URI sont en quelque sorte "agnostiques" de la technologie utilisée. En complément, le choix des URI a été fait pour durer, comme le définit le principe du W3C : « *Cools URIs don't change* »³⁵, soit en français « *Les URI sympas ne changent pas* ». En effet, changer d'URI une fois l'application déployée peut-avoir un effet néfaste, si certaines personnes mettent par exemple un lien dans leurs favoris (ou pire si un site partenaire recopie le lien sur leur site), et que celui-ci ne fonctionne plus.

La fiche d'un stage est disponible sous plusieurs formats et par plusieurs moyens. Exemple pour le stage « 3UDZ » :

Format	URL
HTML	http://monapplication.com/catalogue/fiches/3UDZ
HTML	http://monapplication.com/catalogue/fiches/3UDZ.html
XML	http://monapplication.com/catalogue/fiches/3UDZ.xml
JSON	http://monapplication.com/catalogue/fiches/3UDZ.json
YAML	http://monapplication.com/catalogue/fiches/3UDZ.yml

Nota : Dans la partie visible du client, les identifiants de base de données ne sont pas visibles. On affiche alors le code stage qui est, lui aussi, unique.

Le bundle JMSSerializerBundle³⁶ est un bundle de la communauté de Symfony2. Il permet de sérialiser n'importe quel objet en différents formats : XML, JSON et YAML. Pour fonctionner, il suffit juste d'annoter les classes métiers.

L'objet Stage a été particulier dans le développement du service web. En effet, il comporte des informations propres à l'entreprise (comme celles liées à la tarification) et d'autres informations publiques (comme celles que le client voit sur la fiche du stage). Pour répondre à ce problème, il a fallu recourir au patron de conception : objets de transfert de données (DTO).

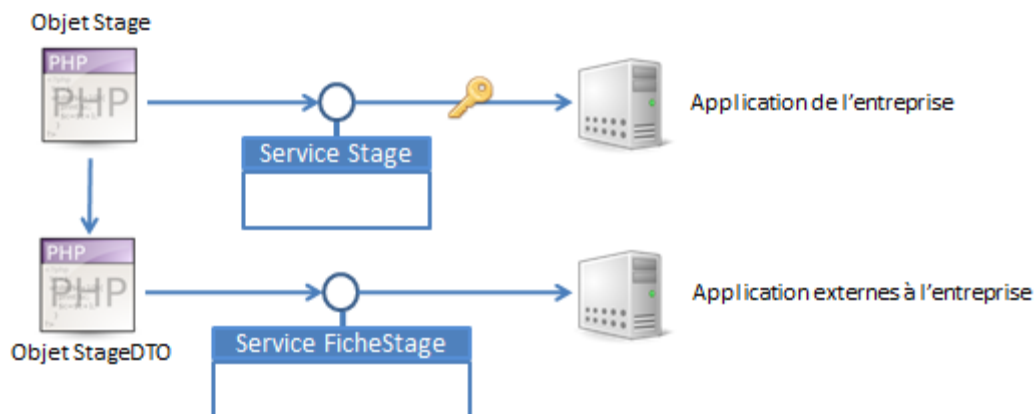


Figure 69 - Les DTO servent aussi à masquer certaines propriétés d'un objet

Ainsi, les informations sont bien séparées. Le service Stage créé à partir de l'objet du même nom, ne sera accessible que pour les applications internes à l'entreprise. Le service FicheStage créé à partir de l'objet StageDTO nom sera accessible pour les applications externes à l'entreprise.

³⁵ <http://www.w3.org/Provider/Style/URI>

³⁶ <https://github.com/schmittjoh/JMSSerializerBundle>

Le routage de Symfony2 permet d'adopter facilement une approche REST car, pour une même URI, il est possible d'affecter plusieurs contrôleurs. Cependant, le distinguo se fera au niveau de la méthode HTTP.

Pour la partie « saisie des stages », le fichier de routage est structuré ainsi :

URI	Méthode	Description
/stages/	GET	Lister les stages
/stages/nouveau	GET	Formulaire pour créer un stage
/stages/	POST	Créer un stage
/stages/{code}/modifier	GET	Formulaire pour modifier un stage
/stages/{code}	PUT	Ajout d'un stage
/stages/{code}	GET	Voir les informations du stage
/stages/{code}	DELETE	Suppression d'un stage

Nota :

- Les URI seront toutes préfixées par /administration/
- Il faut remplacer {code} par le code du stage souhaité.
- Toutes les méthodes renverront un code 200, à l'exception de la méthode PUT qui renverra un code HTTP 201, comme le préconise le W3C³⁷.

Pour la partie relative aux stages du « catalogue », la structure des URL est semblable :

URI	Méthode	Format acceptés	Description
/stages/fiche-{code}._{format}	GET	XML, JSON, YML, PDF, HTML	Accès à la fiche du stage
/stages/{code}/sessions._{format}	GET	XML, JSON, YML	Accès aux sessions du stage

Nota :

- Il faut remplacer {code} par le code du stage souhaité ;
- Il faut remplacer {format} par un format accepté.

Il est maintenant intéressant de faire le lien entre ce qui a été réalisé et sa place dans le modèle de Richardson.

La place de l'interface de programmation (API) dans le modèle de Richardson :

Dans le monde du développement informatique, il est souvent tentant pour le développeur de rendre un travail à la pointe de la technologie. Cependant, tout ceci est chronophage et n'est pas forcément utile, d'autant que le pragmatisme impose parfois de s'éloigner de modèles trop théoriques.

³⁷ <http://tools.ietf.org/html/rfc7231>

Par exemple, pour récupérer une ressource sous un certain format via l'API de l'application, il faut lui spécifier son extension dans l'URI :

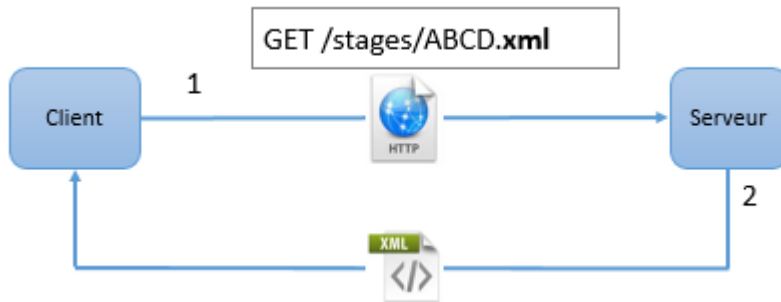


Figure 70 - Exemple d'api travaillant sans négociation de contenu

Du point de vue REST, il ne faudrait pas indiquer l'extension des ressources dans l'URI. Ce qui veut dire que la ressource identifiée par `stages/ABCD.html` n'est pas la même que `stages/ABCD.pdf`. Dans les faits, il s'agit souvent de la même ressource mais qui n'a pas la même représentation. Il faudrait donc utiliser la négociation de contenu et demander le format souhaité dans l'entête HTTP.

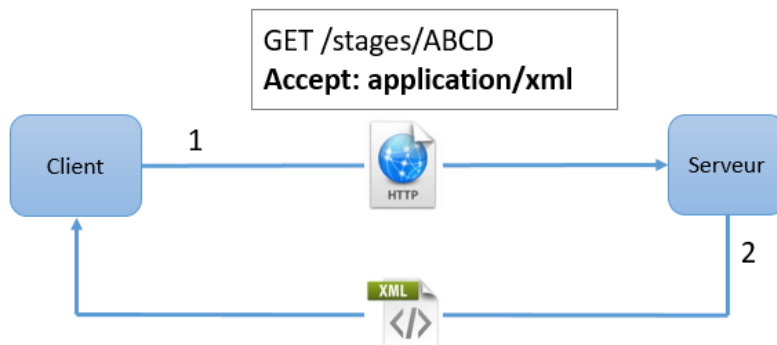


Figure 71 - Exemple d'api travaillant avec la négociation de contenu

Cela n'a pas été mis en place à cause d'une problématique de temps. En effet :

- Du côté fournisseur de service, cela implique d'ajouter un degré de complexité supplémentaire.
- Du côté consommateur de services, il faut ajouter une ligne de code lors de chaque appel pour spécifier le format souhaité.

Le fait de mettre au pluriel l'objet correspondant à la ressource n'est en aucun cas une contrainte de REST. C'est seulement une règle de style qui est conseillée dans l'élaboration du design d'URI.

Le niveau 3 inclut la notion de contrôles hypermédias. Si le tiers qui utilise l'application n'est pas compatible avec ces contrôles, cette mise en place se révélera inutile. L'interface de programmation développée sera peut-être de grande qualité, mais le développeur aura inutilement perdu du temps. De plus, l'ajout des liens alourdit le contenu du message renvoyé au client.

Pour reprendre l'exemple de la partie précédente :

```
HTTP/1.1 200 OK

<listeSessionsLibres>
  <session id="1" du="09-06-2014" au="13-06-2014" placesLibres="2"
    stage="soudage">
    <link rel = "session 1"
      uri = "/sessions/1"/>
    </session>
  <session id="2" du="23-06-2014" au="27-06-2014" placesLibres="1"
    stage="soudage" >
    <link rel = "session 2"
      uri = "/sessions/2"/>
    </session>
</listeSessionsLibres>
```

Le coût du passage du niveau 2 au niveau 3 est de 104 octets (en UTF-8, 1 caractère = 1 octet). Le message sans les liens pèse 227 octets. Dans le cas présent, l'augmentation de la taille du message est de 45%. Augmenter le poids d'une requête de 227 octet est peut-être peu significatif pour une telle ressource. Cependant, cette augmentation peut devenir très conséquente avec une ressource plus complexe.

Dans un contexte d'entreprise où la bande passante est souvent étroite, cette augmentation de poids n'est donc pas négligeable.

L'API créée se trouve au niveau 2 du modèle de Richardson. Bien que certaines parties intègrent des liens hypermédias. En effet, sur les pages du catalogue, il y a le lien de la fiche de stage.

Après avoir vu le fonctionnement microscopique des services de l'application, cette dernière sous-partie en abordera le fonctionnement macroscopique.

III.4 Aperçu global de l'application

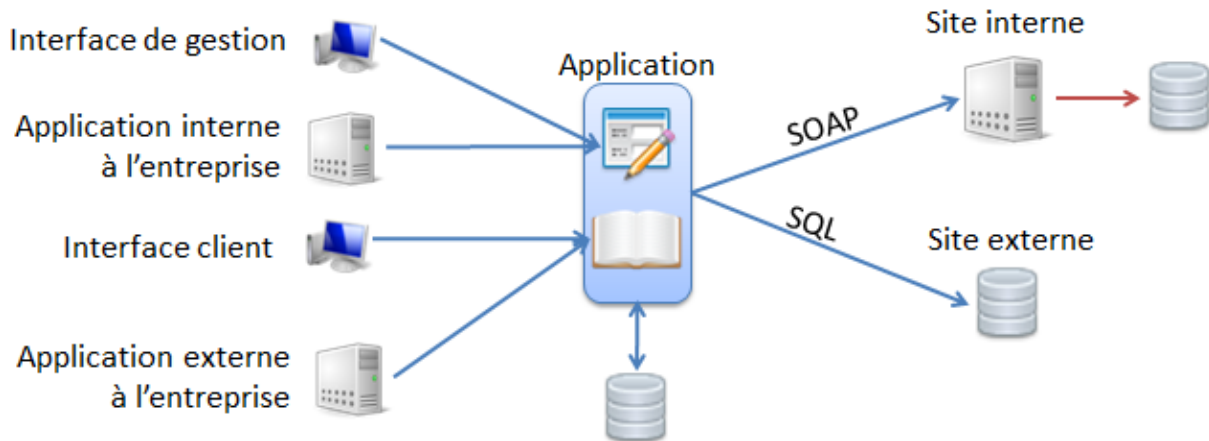


Figure 72 - Vue d'ensemble de l'application

La figure ci-dessus montre bien que l'application est au cœur de nombreux échanges d'informations.

- L'interface de gestion (partie administration) est accessible via des pages internet ;
- Les applications internes à l'entreprise accèdent à l'application via des services REST ;
- L'interface d'accès au catalogue est accessible pour les clients via des pages internet ;
- Les éventuelles applications externes à l'entreprises ou, celles qui ont uniquement besoin de voir l'offre de formation accèdent au catalogue via des services REST.

Concernant les sorties d'informations, celles-ci se font soit via le protocole SOAP (pour le site interne) ou bien directement via des ordres SQL (pour le site externe).

Sécurité :

L'**authentification** à l'application est assurée par le bundle FOSUserBundle³⁸. C'est le bundle le plus populaire de l'écosystème Symfony2. Il permet aussi de gérer toute la partie utilisateur d'une application.

Pour effectuer une opération sur l'application, il faut au préalable qu'un administrateur lui en ait donné l'**autorisation**. En effet, chaque utilisateur est associé à un ou plusieurs groupes, chaque groupe possédant des droits particuliers.

La **confidentialité** vise à empêcher l'accès aux informations à des personnes non habilitées. L'accès au site interne se fait via le protocole HTTPS. Le « S » de HTTPS veut dire *secure* ou bien sécurisé en français. Cette sécurité est assurée

³⁸ <https://github.com/FriendsOfSymfony/FOSUserBundle>

par le protocole SSL/TLS qui est un mécanisme à clef publique/clef privée. On parle alors de cryptographie asymétrique³⁹.

Enfin, la **traçabilité** de chaque action est assurée par le composant Monolog de Symfony2. Ce composant permet d'inscrire dans un fichier texte chaque action souhaitée. Il est ainsi possible de savoir qui a fait quoi à une date précise. Si une erreur importante survient, un mail de notification est automatiquement envoyé au développeur.

La **garantie d'acheminement** n'a pas pu être traitée dans le cadre du développement de cette application car il n'est pas possible d'effectuer un retour en arrière avec les services web du site interne. Cependant, en cas d'erreur, l'utilisateur de l'application est averti que l'opération a échoué.

En termes de **performance** et d'**ergonomie**, l'outil en ligne PageSpeed Insight⁴⁰ de Google permet d'évaluer la qualité d'une application internet en fonction de certains critères. A l'issue de l'évaluation, l'outil donne une note sur 100 et des conseils à suivre pour l'améliorer.

	Mobile : vitesse	Mobile : expérience utilisateur	Ordinateur : note globale
Site interne : page d'accueil	61	66	73
Site externe : page d'accueil	62	62	74
Application : page d'accueil	68	88	82
Site interne : fiche stage	Impossible*	77	Impossible*
Site externe : fiche stage	62	73	83
Application : fiche stage	74	100	87

*Le site interne ne permet pas d'identifier de manière unique une page internet. Il est par conséquent impossible d'utiliser l'outil pour une page autre que la principale. La note sur l'expérience mobile a été réalisée en exportant la page au format HTML et en la déposant sur un serveur distant.

Les notes attribuées par l'outil sont plutôt bonnes pour l'application. Cependant, la personne la plus à même d'évaluer l'application n'est autre que l'utilisateur final.

Les 37 requêtes nécessaires pour l'ajout d'un nouveau stage mettent entre 7 et 9 secondes à s'exécuter. Cette durée peut paraître importante, mais il n'est rien comparé au temps nécessaire à une assistante pour mettre à jour un stage sur les deux sites (interne et externe).

³⁹ http://fr.wikipedia.org/wiki/Cryptographie_asymétrique

⁴⁰ <https://developers.google.com/speed/pagespeed/insights/>

Retour d'expérience

Pour capitaliser l'expérience acquise au cours de la réalisation de ce projet, un retour d'expérience a été fait afin de lister les principales difficultés rencontrées et les apprentissages.

Difficultés rencontrées :

La prise en main des différentes technologies nécessite un temps et un investissement conséquent. Ce temps de mise en place est susceptible de générer un sentiment de frustration car les résultats concrets tardent à venir.

Par ailleurs, le côté « boîte noire » d'un service web peut-être gênant pour le développeur qui doit l'utiliser. De nombreux tests sont parfois nécessaires pour comprendre son fonctionnement. Il est donc primordial de bien le documenter, sans quoi, il fera perdre un temps précieux aux développeurs qui auront à l'utiliser.

Créer une API pleinement REST nécessite un temps de développement plus long que d'exposer simplement des objets sous différents formats (JSON, XML, ...). De plus, rendre son API conforme au niveau 3 du modèle de maturité de Richardson nécessite d'implémenter des fonctionnalités souvent peu utiles au regard des besoins réels des utilisateurs.

Apprentissage :

Il y a plusieurs façons de construire une application et l'écosystème PHP regorge de bundles, plus ou moins pertinents, pour faciliter le travail du développeur. Le rôle du chef de projet est d'arbitrer les choix stratégiques nécessaires. Il doit faire preuve d'assertivité et de pugnacité car, pendant la veille, il est souvent tentant d'abandonner une technologie pour une qui paraît meilleure.

La prise en main d'un outil requérant du temps, elle nécessite de la patience.

Le développeur part souvent dans l'optique que son application ne générera pas d'erreurs. Or, avec un contexte HTTP, une erreur est vite survenue (problème de bande passante, ...). Il faut bien prendre en compte ces erreurs lors de la conception.

L'apprentissage du style d'architecture REST permet, du fait de son mode de fonctionnement entièrement basé sur HTTP, de mieux comprendre celui du Web.

Les options retenues ont permis la réussite du projet.

Conclusion

La première partie du mémoire a montré l'importance de bien étudier l'environnement cible. L'étude de la structure des services web de sites tiers conduit à faire des choix que le concepteur n'aurait initialement pas imaginés. Il est donc primordial de bien prendre en compte l'architecture des environnements cibles dès le début du projet.

La seconde partie, quant à elle, traite des outils qui permettent de créer des services et d'utiliser ceux réalisés par des tiers. Cependant, la mise en place de tels composants requiert un temps de mise en place non négligeable. Le choix d'utiliser Symfony2 presque été arbitraire.

Le concept du « tout module » mis en avant par Symfony2, correspond bien aux architectures orientées services. A l'instar du système d'information, Symfony2 est composé de briques (bundles) qui s'emboîtent telles celles d'une construction Lego.

En revanche la recherche du bundle adéquat peut s'avérer longue et fastidieuse, et il est parfois nécessaire de réaliser soi-même certains composants « outils ».

La troisième et dernière partie entre plus dans le détail des technologies SOAP et REST. Ces deux technologies sont différentes, tant sur la forme que sur le fond.

Le tableau ci-dessous fait état des différences de styles entre ces technologies :

Style SOAP	Style REST
obtenirCatalogueStage	GET(URI)
mettreAJourStage(stage)	PUT(URI, ressource)
creerStage(stage)	POST(URI, ressource)
supprimerStage(idStage)	DELETE(URI)

Les avantages et inconvénients de chacune de ces technologies peuvent se résumer ainsi :

- La technologie **SOAP** repose sur des standards bien définis. Même si les requêtes SOAP transitent le plus souvent via le protocole HTTP, il est aussi possible d'en utiliser d'autres tels que SMTP. Un autre avantage de SOAP est la possibilité d'utiliser le service web de façon asynchrone. En revanche, l'omniprésence du format XML et des enveloppes, rendent le service verbeux. Un service SOAP est difficile à mettre en place sans outils adéquats et a un coût de production plus important que REST.
- Le style d'architecture **REST**, offre plus souplesse que SOAP en termes de formats supportés (JSON, XML, YAML, ...). L'utilisation de la méthode HTTP GET permet aussi la mise en cache des résultats, ce qui raccourcit le temps de réponse du serveur. Cependant, un service REST n'est disponible que depuis HTTP et sa structure est plus orientée CRUD (création, lecture, mise à jour et suppression). Il est cependant possible de sortir de ce cadre avec des URI de cette forme : `api.org/utilisateur/42/valider-mail` (pour valider le mail de l'utilisateur 42).

Si ces deux technologies sont si différentes, c'est qu'elles ne s'utilisent pas dans le même contexte. Chaque avantage et inconvénient est à prendre en compte lors du choix de technologie à utiliser.

D'une manière plus générale, les architectures orientées services (SOA) permettent de ne pas dépendre d'outils en particulier. Les applications peuvent communiquer entre-elles à condition, bien sûr, d'adopter un langage commun. Chaque application reste ainsi concentrée sur le cœur de métier de l'entreprise qui la réalise. Energy Formation a bâti son système d'information autour d'une application monolithique. Pour répondre aux demandes, le prestataire de service est parfois sorti de son périmètre métier. Le résultat de la fonctionnalité développée s'est ainsi montré inférieur aux attentes.

La mise en place d'une gouvernance SOA au sein d'un système d'information permet de ne plus dépendre d'applications ou de technologies. Le système d'information est alors plus flexible et évolutif.

L'avènement de SOA a permis l'essor des applications de types « *as a service* » (en tant que service). L'« informatique en nuage »⁴¹ (*cloud*) révolutionne le poste de travail de l'entreprise et donc le travail du développeur.

En effet, les applications de type client lourd prennent de moins en moins de place dans l'entreprise. Celle-ci préfère opter pour des applications web « en tant que service » (par exemple *SaaS*⁴²), moins sensibles aux sécurités atypiques des entreprises (pare-feu, compte administrateur désactivé ...).

Un bon exemple illustre cette nouvelle tendance : l'adaptation de la suite bureautique Microsoft Office en version navigateur sortie en juin 2011. Un document Microsoft Office Online est ainsi accessible depuis n'importe quel ordinateur qui possède un navigateur internet. Le portage de l'application sur les tablettes permet également à un très grand nombre d'écrans, d'accéder à un document.

Si selon Abraham Maslow « *Tout ressemble à un clou pour qui ne possède qu'un marteau* », il ne faut pas voir l'approche SOA comme le marteau doré permettant de résoudre n'importe quel problème. SOA ne peut rien face aux problèmes tels que le code mal optimisé, vieillissant, trop monolithique et peu performant. De plus sa mise en place est particulièrement coûteuse en temps.

SOA doit donc plutôt être envisagé comme un outil de synergie, permettant de collaborer efficacement dans l'objectif de répondre aux enjeux métiers de l'entreprise.

Depuis quelques années, un nouveau paradigme émerge : celui des micro-services⁴³. Cette nouvelle vision consiste à réduire au maximum la granularité d'un service. Un micro-service est un programme, avec une philosophie semblable à celle d'une commande Unix, qui doit : « *Ne faire qu'une seule chose, et la faire bien* » (Douglas McIlroy).

En complément, chaque micro-service est relié à sa propre base de données pour être le plus indépendant possible.

⁴¹ http://fr.wikipedia.org/wiki/Cloud_computing

⁴² Logiciel en tant que service (*Software as a Service*)

⁴³ <http://martinfowler.com/articles/microservices.html>

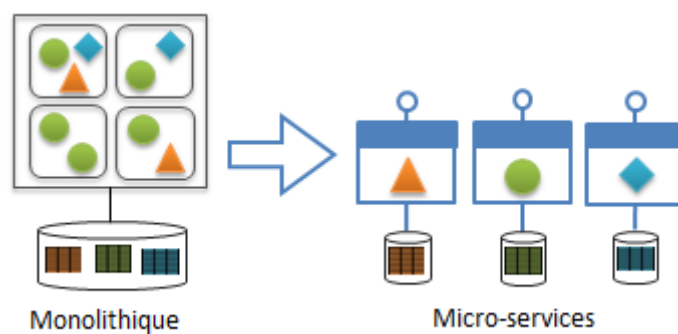


Figure 73 – Application monolithique et micro-services

Cette conception des services pose la question de l'avenir des architectures orientées services qui passera peut-être par l'utilisation de micro-services.

Bibliographie

Xavier Fournier-Morel, Pascal Grojean, Guillaume Plouin et Cyril Rognon
« SOA – Le guide de l'architecte d'un SI agile » (éditions DUNOD)

Mark Masse « REST API Design Rulebook – Designing Consistent RESTful Web
Service Interfaces » (édition O'Reilly)

Conférence de William DURAND lors du PHP Lyon Tour (AFUP) en juin 2014
« REST dans le monde Symfony »
(<https://www.youtube.com/watch?v=nm1obAL1xoo>)

Site internet de Martin FOWLER - <http://martinfowler.com>

Documentation technique de Symfony2 -
<http://symfony.com/doc/current/index.html>

Table des illustrations

Figure 1 - Processus pour la mise à jour des données relatives aux stages	5
Figure 2 - Client-serveur avec une architecture à 3 niveaux.....	6
Figure 3 - Logo d'IE7	6
Figure 4 - Entropie du système d'information.....	8
Figure 5 - Un intérêt de SOA, la mutualisation des services.....	9
Figure 6 - Des services interopérables.....	10
Figure 7 - Organisation des stages dans le site interne et structure d'un stage.	11
Figure 8 - Les sessions sont dupliquées pour l'accès aux questionnaires.....	12
Figure 9 - Modification des données (proposition 1.1)	15
Figure 10 - Proposition de maquette réalisée pour la proposition 1.1	16
Figure 11 - Modification des données (proposition 1.2).....	17
Figure 12 - Ensemble des requêtes nécessaires pour modifier un stage	17
Figure 13 - Epuisements de délai.....	18
Figure 14 - Modification des données pour la proposition 1.3	19
Figure 15 - Accès à l'offre de formation (proposition 2.1)	20
Figure 16 - Ensemble des services appelés pour accéder aux informations complètes d'un stage.....	20
Figure 17 - Accès à l'offre de formation (proposition 2.2)	21
Figure 18 - Reconstruction dynamique d'une page web	22
Figure 19 - Accès à l'offre de formation (proposition 2.3).....	23
Figure 20 - Fonctionnement global validé	24
Figure 21 - Logo de Symfony	26
Figure 22 - Barre d'outils de débogage du framework Symfony2	27
Figure 23 - Cheminement d'une requête dans une application Symfony2.....	28
Figure 24 - Fonctionnement du conteneur de service	30
Figure 25 - Diagramme UML d'un Singleton	31
Figure 26 - Injection de service dans un autre	32
Figure 27 - Logo de TWIG	32
Figure 28 - Logo de Doctrine	33
Figure 29 - Un objet mappé avec la base de données.....	33
Figure 30 - Echange de données entre serveur et différents périphériques	35
Figure 31 - Intérêt des framework JavaScript MVC	35
Figure 32 - Logo de jQuery.....	36
Figure 33 - Le moteur de recherche Google et la technologie AJAX	36
Figure 34 - Syndrome du plat de spaghettis.....	37
Figure 35 - Logo de Bootstrap.....	37
Figure 36 - Exemple de composants disponibles avec Bootstrap	38
Figure 37 - Agencement d'une page web adaptative sur plusieurs appareils.....	38
Figure 38 - Logo de JSHint	40
Figure 39 - Exemple de documentation générée par SAMI	41
Figure 40 - Logo de GruntJS	42
Figure 41 - Logo de git.....	42
Figure 42 - La gestion des branches par Git.....	43
Figure 43 - Logo de Piwik	43
Figure 44 - Rapport d'audience d'un site réalisé avec l'outil Piwik	44
Figure 45 - Piwik permet de ne pas avoir de bandeau requérant le consentement du visiteur à recevoir des cookies.....	44
Figure 46 - Une application au centre des échanges.....	45

Figure 47 - Structure SOAP (source : http://fr.wikipedia.org/wiki/SOAP).....	45
Figure 48 - WSDL est compris par les IDE (exemple avec NetBeans)	46
Figure 49 - Utilité d'un registre UDDI	47
Figure 50 - Utilité d'un DTO	48
Figure 51 - Diagramme des classes d'accès aux données (DAL) de l'application	49
Figure 52 - Diagramme des classes des services de l'application	49
Figure 53 - Les messages flash permettent de stocker de l'information dans divers endroits et moments de la requête pour ensuite la restituer au client	50
Figure 54 - Fonctionnement des services pour le site interne	50
Figure 55 - Processus de récupération d'un jeton de session (guid)	51
Figure 56 - Processus de création d'un stage en utilisant les services	52
Figure 57 - Processus de récupération des données de sessions de stage	53
Figure 58 - Un GIF animé en attendant le chargement complet de la page	53
Figure 59 - Chronométrage d'affichage d'une fiche stage (réalisé avec FireBug)	53
Figure 60 - Restitution d'une fiche de stage	54
Figure 61 - Comparatif entre l'approche <i>code first</i> et l'approche <i>database first</i> .	55
Figure 62 - Utilité d'une classe de conversion.....	56
Figure 63 - Analyse de tendance sur le moteur de recherche Google (Google Trend) entre les deux types de service web SOAP et REST	57
Figure 64 - Les étapes vers la gloire de REST.....	61
Figure 65 - Le niveau 0 du modèle de Richardson.....	61
Figure 66 - Le niveau 1 du modèle de Richardson.....	63
Figure 67 - Le niveau 2 du modèle de Richardson.....	64
Figure 68 - Le niveau 3 du modèle de Richardson.....	65
Figure 69 - Les DTO servent aussi à masquer certaines propriétés d'un objet ..	68
Figure 70 - Exemple d'api travaillant sans négociation de contenu	70
Figure 71 - Exemple d'api travaillant avec la négociation de contenu	70
Figure 72 - Vue d'ensemble de l'application	72
Figure 73 - Application monolithique et micro-services	77

Annexes

1 – Rendu d'une fiche Stage

PC Niveau 1 : Initiation aux mesures et contrôles courants

Z107

<p>Domaines et Code Transport D28</p> <p>Distribution Electricité et Gaz Autres Infrastructures Gazières</p> <p>Accessibilité TD</p> <p>Collège(s) concerné(s) Exécution Maîtrise</p> <p>Durée 35 heures réparties sur 5 jour(s)</p> <p>Lieu de réalisation Sites Energy Formation</p> <p>Effectif maxi :12</p> <p>Tarif HS4A</p> <p>Contact(s) Assistante gestion@gdfsuez.com</p> <p>Dates des sessions du 31/12/2014 au 31/12/2014 du 23/03/2015 au 27/03/2015 du 08/06/2015 au 12/06/2015 du 31/12/2015 au 31/12/2015</p>	<p>Population concernée Salariés débutants ou déjà dans le domaine de la protection cathodique, amenés à réaliser les tâches de niveau 1 définies dans la norme EN 15257. Salariés d'entreprises externes.</p> <p>Pré-requis Une période de binôme de quelques semaines avec un niveau 1 Un niveau de connaissance CAP / BEP ou un niveau équivalent est souhaité.</p> <p>Objectifs de la formation - Maîtriser des mesures courantes de protection cathodique, en effectuer une première analyse et en rendre compte - Assurer la surveillance des travaux d'installation des systèmes de protection cathodique - Assurer la maintenance des équipements - Relever et classer les résultats obtenus - Superviser les agents ayant suivi le stage "Exécution des mesures simples de protection cathodique "</p> <p>Capacités développées - Enumérer les critères de protection cathodique, décrire les éléments constitutifs d'un système de protection cathodique et les différentes mesures courantes. - Enumérer les différents types de revêtement ainsi que leurs qualités et défauts - Identifier et comprendre les paramètres de protection cathodique qui influent sur les mesures courantes. - Décrire les différentes techniques de recherche de défaut de revêtement et leurs limites. - Enumérer les normes et les textes réglementaires relatifs à la réalisation des tâches demandées. - Réussir l'évaluation théorique provisoire et/ou finale - Effectuer un premier niveau d'analyse des mesures courantes. - Réaliser l'ensemble des tâches opérationnelles liées aux mesures courantes de protection cathodique. - Assurer la surveillance des travaux d'installation des dispositifs de protection cathodique. - Réaliser des recherches de défauts de revêtement - Respecter les règles liées à la protection de l'environnement.</p>	<p>Contenu du stage PARTIE THEORIQUE Corrosion des matériaux Chimie - Electrochimie Electricité Théorie générale de la protection cathodique Courants vagabonds Technique de mesures Revêtements Connaissances des normes</p> <p>PARTIE PRATIQUE Mesures de potentiels / d'intensité Corrosivité des sols / résistivité Anodes galvaniques Poste de soutirage / déversoir Influences Raccords isolants - Fourreaux Détection de défauts Environnement : production des déchets et modifications du milieu (pH)</p> <p>Positionnement dans un cursus de professionnalisation Stage inscrit dans une offre de formation : "protection cathodique".</p> <p>Intervenant Formateur(trice) d'Energy Formation et intervenant(e) GRTGaz confirmés(es) spécialistes du métier.</p> <p>Méthodes Pédagogiques Une période de binôme : L'approche de la connaissance du métier par un accompagnement du stagiaire en unité par un agent de niveau 1 en protection cathodique. Alternance d'apports théoriques et d'échanges.</p> <p>Moyens Pédagogiques Mise en situation d'apprentissage des stagiaires sur des installations et réseaux pédagogiques des sites Energy Formation GDFSUEZ, au plus près des matériels et réseaux présents en exploitation. Documentation remise à chaque participant.</p> <p>Evaluation Une évaluation individuelle des acquis sera réalisée en fin de stage. Les résultats seront transmis à l'employeur et au stagiaire. L'apprenant devra être informé par sa hiérarchie qu'à l'issue du stage, il y aura une évaluation de ses capacités acquises (savoir, savoir-faire).</p>
--	--	--

Mise à jour le 30/08/2013

Energy Formation

GDF SUEZ

2 – Exemple WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://tempuri.org/OP/Session"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://tempuri.org/OP/Session"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://tempuri.org/OP/Session">
      <s:element name="InitSession">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="login"
              type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="password"
              type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="InitSessionSoapIn">
    <wsdl:part name="parameters" element="tns:InitSession"/>
  </wsdl:message>
  <wsdl:portType name="WSSessionSoap">
    <wsdl:operation name="InitSession">
      <wsdl:input message="tns:InitSessionSoapIn"/>
      <wsdl:output message="tns:InitSessionSoapOut"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WSSessionSoap" type="tns:WSSessionSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="InitSession">
      <soap:operation
        soapAction="http://tempuri.org/OP/Session/InitSession" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="WSSessionSoap12" type="tns:WSSessionSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="InitSession">
      <soap12:operation
        soapAction="http://tempuri.org/OP/Session/InitSession" style="document"/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

```
</wsdl:binding>
<wsdl:service name="WSSession">
  <wsdl:port name="WSSessionSoap" binding="tns:WSSessionSoap">
    <soap:address
location="https://siteinterne/opdotnet/webservices/session.asmx"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

3 – Glossaire

API : pour *Application Programing Interface* ou *interface de programmation*, est un ensemble de fonctions d'un logiciel destiné à d'autres.

Bundle : c'est une sorte de plugin pour une application Symfony2.

Cloud (ou **Cloud computing**) : consiste à délocaliser les applications et les données dans le nuage (Web). On parle aussi d'« informatique en nuage » ou bien du « nuage ».

CRUD : désigne les quatre opérations de persistance d'un objet : créer (*Create*), lire (*Read*), mettre à jour (*Update*) et supprimer (*Delete*).

Framework : ensemble de composants qui permet cadrer la construction d'une application.

HTTP : pour *HyperText Transfer Protocol*, est le protocole qu'utilise le navigateur pour accéder à une page web.

Patron de conception (ou **design pattern**) : correspond à une réponse standardisée et éprouvée d'un problème de conception logiciel.

Registre UDDI : annuaire de services web.

REST : pour *REpresentational State Transfer*, est un style d'architecture entièrement basé sur HTTP permettant l'accès et la modification de ressources.

RFC : pour Request For Comment est un document officiel détaillant un aspect d'internet (protocole, concept, ...).

Service web : programme permettant l'échange d'informations entre deux systèmes via internet.

SOAP : pour *Simple Object Access Protocol*, est un protocole bâti sur XML permettant l'échange de messages.

W3C : *World Wide Web Consortium*, l'organisme de normalisation du web

WSDL : pour *Web Service Description Language* est la description normée d'un service web SOAP.