

Cours	Nouvelles technologies Web
Auditoire	4 ^{ème} année Ingénieur en Réseau Télécommunication (RT4)
Etablissement	Institut National des Sciences Appliquées et de Technologies
Responsable du cours	Aymen SELLAOUTI
Années Universitaires	2015-2016

Objectifs :

Ce cours offre à l'étudiant une panoplie de nouvelles technologies Web. A L'issue de ce cours, l'étudiant devrait être capable de:

- Créer des documents HTML5 et d'établir le formatage adéquat pour raffiner la présentation en utilisant le CSS3.
- Utiliser un des Framework PHP les plus utilisés, à savoir Symfony2.
- Réaliser une application Web MVC2.
- Appréhender les ORM.
- Manipuler les « Router ».
- Utiliser les moteurs de Template.

Pré requis : Aucun

Plan du Cours :

1. **HTML5**
 - 1.1 **Introduction**
 - 1.1.1 Une brève histoire du WEB
 - 1.1.2 Le fonctionnement du WEB
 - 1.2 **Qu'est-ce que HTML 5 ?**
 - 1.3 **Les éléments basiques d'une page HTML**
 - 1.3.1 Introduction au langage HTML
 - 1.3.2 Le doctype
 - 1.3.3 Les balises usuelles
 - 1.3.4 Les images
 - 1.3.5 Les liens
 - 1.3.6 Les listes
 - 1.3.7 Les tableaux
 - 1.4 **Formulaire**
 - 1.5 **Les grandes nouveautés du HTML 5**
 - 1.5.1 Les balises sémantiques
 - 1.5.2 Les balises multimédias (audio et vidéo)
 - 1.5.3 Les formulaires en HTML5

2. CSS3

- 2.1 Principe de base
- 2.2 Formatage du texte
- 2.3 La couleur du fond
- 2.4 Les bordures et les ombres
- 2.5 Les sélecteurs
- 2.6 Notion de class et id
- 2.7 Les modèles de boîtes
- 2.8 Flux et positionnement
- 2.9 Les Flexbox

3. Symfony2 Introduction

- 3.1 Introduction
- 3.2 Qu'est-ce qu'un Framework ?
- 3.3 Architecture d'un projet Symfony2
- 3.4 Le contrôleur frontal
- 3.5 Architecture MVC2
- 3.6 Traitement d'une requête au sein de Symfony2
- 3.7 Les Bundles
- 3.8 Installation d'un projet Symfony2

4. Les contrôleurs

- 4.1 Introduction
- 4.2 Exemple d'un contrôleur
- 4.3 Lien entre la root et le contrôleur
- 4.4 Les rôles du contrôleur

5. Rooting

- 5.1 Introduction
- 5.2 Format de gestion du Routing
- 5.3 Externalisation des fichiers de rooting
- 5.4 Squelette d'une root
- 5.5 Paramétrage d'une root
- 5.6 Ordre de traitement des roots
- 5.7 Rooting avec les annotations
- 5.8 Débogage des routes

6. Les Twigs

- 6.1 Introduction
- 6.2 Les bases du langage du moteur Twig
- 6.3 Twig et l'affichage
- 6.4 Les filtres
- 6.5 Les structures conditionnelles et itératives
- 6.6 Héritage
- 6.7 Inclusion et génération des liens
- 6.8 Surcharge de template

7. Doctrine

- 1. Introduction
- 2. Les entités

3. Mapping : les annotations
4. Génération des entités
5. Gestion des bases de données
6. Le service EntityManager
7. Le Repository
8. Création de requêtes DQL et QueryBuilder
9. Externalisation des requêtes
10. Gestion des relations entre les entités

8. Les formulaires

1. Introduction
2. Définition
3. Création d'un formulaire
4. Affichage du formulaire dans le Twig
5. Les composants du formulaire
6. Gestion de la soumission du formulaire
7. Externalisation de la définition du formulaire
8. Les différents types dans le formulaire
9. Les validateurs

Bibliographie :

- [1] Cours de Mme. Mouna Laroussi, 2013/ Département Informatique INSAT.
- [2] Symfony, The Book, SensioLabs Version Aout2015.
- [3] Rodolphe Rimelé, HTML 5, Une référence pour le développeur Web, Ed. Eyrolles, 2011.
- [4] Jon Duckett, HTML&CSSDesign et création Web, Ed. Pearson, 2012.



HTML5

A person is shown holding a blue cube, which is part of a larger 3D structure.

Evolution

Une brève histoire du WEB

- 1991 : HTML 1
- A partir de 1995, tout s'accélère, création de:
 - Javascript,
 - HTML 2.0,
 - Internet explorer, 1^{ère} version.
- 1996 CSS 1,
- 1997 HTML 4,
- 1998 CSS 2,
- 2000 XHTML 1,
- 2008 HTML 5.

Plan de la présentation

Introduction

- Une brève histoire du WEB
- Le fonctionnement du WEB

Qu'est ce que HTML 5 ?

- HTML 5 : un ensemble de technologies (HTML 5, CSS 3, JavaScript)

Les éléments basiques d'une page HTML

- Introduction au langage HTML
- Le doctype
- Les balises usuelles
- Les images
- Les liens
- Les listes
- Les tableaux

Formulaire

Les grandes nouveautés du HTML 5

- Les balise sémantiques
- Les balises multimédias (audio et vidéo)
- Les formulaires en HTML5

Le fonctionnement du WEB

Le fonctionnement du WEB

Les pages sont enregistrées sur un serveur.

C'est un ordinateur (souvent très gros) accessible par n'importe quel internaute. Il se trouve généralement chez les fournisseurs d'accès.
C'est lui qui contient tout votre site web




A l'origine du WEB : Tim berners-Lee

Une brève histoire du WEB

- L'initiateur du premier site **internet** et du premier **navigateur** (WWW) en 1990 est **Tim berners-Lee**
- Il est à l'origine des 3 technologies principales du WEB: **Les adresses web**, le protocole **HTTP** et le **HTML**
- Il est le fondateur et le président du **W3C** depuis 1994.
- Le W3C est un organisme indépendant qui émet des standards pour le WEB.

Comment ça marche ?

Le fonctionnement du WEB

La personne (ou plutôt l'ordinateur... ou plutôt le logiciel) qui veut afficher une page web s'appelle le client.



Comment ça marche ?

Le fonctionnement du WEB

Lorsque le client veut afficher une page web, (c'est-à-dire quand on tape une adresse) Il envoie une requête au serveur

Client → Serveur : http://www.monsite.tn

Client : Cher serveur, Peux-tu m'envoyer la page demandée ?

Serveur : Mon site web

7

Comment ça marche ?

Le fonctionnement du WEB

Pour résumer

Etape 1 : Commander une page

Etape 2 : Récupérer la page

Etape 3 : Afficher la page

10

Comment ça marche ?

Le fonctionnement du WEB

Le serveur cherche la page.

- S'il la trouve, il la renvoie
- S'il ne la trouve pas, il renvoie une erreur

Client → Serveur

Serveur : Mon site web

Ok

8

HTML 5 : HTML 5, CSS 3 et JavaScript

Qu'est ce que HTML 5 ?

- Le **HTML 5** est fondamentalement le langage HTML dans sa 5^{ème} version...
- ...Mais il regroupe, par un abus de langage justifié, le **HTML 5**, le **CSS 3** et le **JavaScript**. **Le HTML 5** est donc la pierre angulaire de cet l'édifice.

On va donc avoir :

- **HTML** -> **fond du document**
- **CSS** -> **forme du document**
- **JavaScript** -> **dynamisme du document**

11

Comment ça marche ?

Le fonctionnement du WEB

La page est alors décodée et affichée par un logiciel : c'est le **navigateur**.

Serveur : Mon site web

Client : Navigateur

Le navigateur est un logiciel qui interprète la page HTML et renvoie une sortie sur l'écran.

9

Introduction au langage HTML

Premier document HTML

- HTML (**Hypertext Markup Language**) est le format de données structurant une page web.
- HTML est initialement dérivé du **SGML** (**Standard Generalized Markup Language**) dont la première publication date de 1986.
- C'est un langage de **balisage** : <ballese>Contenu</ballese>

```
<html>
  <head>
    Les en-têtes du document
  </head>
  <body>
    Le corps du document
  </body>
</html>
```

- Exemple de document HTML avec les premières balises **html**, **head** et **body**

12

Introduction au langage HTML

Structure d'une page HTML5

Une entête

```
<!DOCTYPE html>
<html lang="fr">
```

Un corps

```
<head>
<title>Pages personnelles...</title>
</head>

<body>
<p>Bonjour !!!</p>
</body>
```

HTML

13

Les balises usuelles

Les balises peuvent être de **deux natures différentes**

➤ **Les balises « block »** : Elles englobent le contenu. Elles contiennent des paragraphes, des titres, etc.

HTML

```
<p> Ceci est un paragraphe </p>
```

➤ **Les balises « inline »** : Elles sont dans le texte. (mettre en gras, souligner, etc...)

HTML

```
<p> On va écrire un texte <strong>en gras</strong> !!!</p>
```

16

Introduction au langage HTML

Avec quoi fait-on une page web ?

Page Web

Le code compliqué qui décrit la page s'appelle l'**HTML**
C'est ce que nous allons apprendre à faire

14

Les balises usuelles

L'entête

Placée entre les balises **<head>** et **</head>**
et marquer le titre de la page

Cette balise permet de dire que le contenu est écrit avec la **norme utf-8** (avec les accents é-à-é-é-...)

HTML

```
<html>
<head>
<title>Filippo - Download Free Software</title>
</head>
<body>
<p>Ceci est un paragraphe</p>
</body>
</html>
```

17

Le Doctype avant tout

Le Doctype

- Le **doctype** est le préambule d'une page HTML, il spécifie le type de document.
- Doctype HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```
- Doctype XHTML 1.1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```
- Doctype HTML 5

```
<!DOCTYPE html>
```

15

Les balises TITLE

L'entête

- Son contenu est le titre apparaît dans la barre de titre du navigateur ainsi que dans les signets lorsque vous mettez le site en favoris.

Exemple:

```
<title>Cours HTML5 - Apprenez les bases du HTML5</title>
```

18

L'entête

Les balises Méta

- Les balises meta sont obligatoirement placées dans la balise <head>
- Ils permettent de renseigner diverses informations sur la page en cours, comme son auteur, une courte description du document, des mots-clefs, etc.

```
<meta name="Nom" content="contenu">
```

Exemple : la meta author : Renseigne le prénom et nom de l'auteur de la page courante. S'il y a plusieurs auteurs, les séparer d'une virgule.

```
<meta name="author" content="Mona LAROUSSI,  
Aymen SELLAOUTI">
```

19

L'entête

Les balises Méta

- meta language: Langue de la page sous la forme d'une abréviation (2 lettres).
- Cette zone n'est pas obligatoire et sans doute inutile (elle ne semble pas prise en compte par les moteurs).
- Exemples de valeurs possibles : "fr" (Français), "en" (Anglais), "de" (Allemand), "es" (Espagnol), "it" (Italien).

```
<META NAME="Language" CONTENT="fr">
```

22

L'entête

Les balises Méta

- meta keywords : Contient une liste de mots-clefs sur lesquels les moteurs de recherche vont se référer.
- Les mots-clefs sont séparés par une virgule.
- Avoir trop de mots-clefs peut être considéré par les moteurs comme du spam, donc ne pas dépasser les 500 caractères.

```
<meta name="keywords" content="cours html,  
balise meta, formulaires, listes, tableaux, cadres">
```

20

L'entête

Les balises Méta

- meta Description : Description de la page. **Zone très utile.**
- Chaque page DOIT avoir une description indépendante (entre 100 et 1000 caractères).
- Cette description est fréquemment reprise par les annuaires qui indexent votre site.
- Évitez** donc d'y faire figurer une suite de mots clés sans attrait pour les internautes.

```
<META NAME="Description" CONTENT="Cette  
description est faite dans notre cours de Technologies  
Web">
```

23

L'entête

Les balises Méta

meta category: Contient une liste de mots qui décrivent la catégorie de la page. Elle est utile pour les annuaires et elle n'est pas obligatoire.

```
<META NAME="Category" CONTENT= "Webmastering,  
Gestion de sites">
```

S'il y a plusieurs valeurs possibles, alors elles seront séparées par des virgules)

21

L'entête

Les balises Méta

- <META NAME="Copyright" CONTENT="Indiquez ici les copyrights.">
- <META NAME="Generator" CONTENT="Listez ici à la façon des mots clés les différents logiciels utilisés.">
- <META NAME="Date" CONTENT="Inscrivez ici la date de création de votre page.">

Exemple :

```
<META NAME="Copyright" CONTENT="© 1999 Hugo ETIEVANT">  
<META NAME="Generator" CONTENT="HomeSite v1.2,Paint Shop Pro  
5.0,Animation Shop,Netscape Communicator 4.0">  
<META NAME="Date" CONTENT="Mon, 14 Sep 1998 08:00:00">
```

24

L'entête

Les balises Méta

Les métas http-equiv

- Elles s'utilisent comme ceci :
`<meta http-equiv="mot clé ici" content="valeur ici" />`
- Elles nous permettent de préciser l'encodage utilisé pour la page grâce au mot clé **Content-Type**.
Exemple pour un encodage au format **UTF-8** :
`<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />`

Pour les pages en **arabre** on peut utiliser l'encodage :
ISO-8859-6

25

Les paragraphes

Les balises usuelles

HTML

```
> <p> ... </p> : Balise de paragraphes (balise la plus utile !) [Block]
```

HTML

```
<body>
<p>
    Thomas Andrews (7 février 1873 - 15 avril 1912) est un architecte naval britannique qui a travaillé à la construction de plusieurs paquebots. Il grandit au sein d'une famille aisée, puis devient apprenti dans les chantiers navals Harland & Wolff de Belfast dont le président, Lord Pirrie, est son oncle. Pendant quelques années, Andrews gravit progressivement les échelons avant d'être nommé directeur général de l'entreprise. La réparation du paquebot de la White Star Line Suevic, à laquelle il participe, fait de lui un des architectes navals les plus renommés de son époque.
</p>
</body>
```

28

L'entête

Les balises Méta

- meta Refresh. Permet de :
 - Recharger (réactualiser) automatiquement la page toutes les n secondes.
`<META http-equiv="Refresh" content="10">` (pour 10 secondes)
 - Rediriger automatiquement le navigateur vers une autre page au bout de n secondes
`<META http-equiv="Refresh" content="10; URL=http://www.fst.rnu.tn">`

26

 : Retour à la ligne

Les balises usuelles

HTML

```
> <br/> : Retour à la ligne
```

HTML

```
<body>
<p>
    Ligne 1 <br/>
    ligne 2
    <br/> Ligne 3
    toujours ligne 3
</p>
</body>
```

29

Le corps

Les balises usuelles

HTML

```
<html>
  <head>
  </head>
  <body> ←
    Placé entre les balises <body> et
    termine par </body>
    C'est là que va figurer tout le contenu
    de la page
  </body>
</html>
```

27

Exercice d'application

Les balises usuelles

- Ouvrir l'éditeur du texte Notepad++
- Écrire le code HTML permettant d' :
 - Ajouter le titre « Ma première page »
 - Afficher Bonjour HTML5!
 - Ajouter le texte « et Bienvenue Chez SMARTFUTUR » dans une nouvelle ligne
- Enregistrer le fichier créé sous le nom EX1.html
- Aller au menu Exécution et choisir un navigateur (exp : Chrome)

HTML

```
<!DOCTYPE HTML>
<html>
<head>
<title>Ma première page Web</title>
</head>
<body>
    Salut ! <br/> et Bienvenue chez SMARTFUTUR
</body>
</html>
```

Google Chrome

Favoris Sites suggérés Get

Ma première page Web

Salut! et Bienvenue chez SMARTFUTUR

30

Les titres

Les balises usuelles

```

<h1> ... </h1> Titre TRES important
<h2> ... </h2> Titre important
<h3> ... </h3> Titre moyen
<h4> ... </h4> Titre pas important
<h5> ... </h5> Titre pas du tout important

```

HTML

```

<body>
    <h1> Chapitre 1</h1>
        <h2> Section 1 </h2>
            <p> texte texte....</p>

        <h2> Section 2 </h2>
            <p> camion </p>

    <h1> Chapitre 2 </h1>
        <p> texte texte texte .... </p>
</body>

```

31

Les listes

Les listes

- ... Liste non triée, liste à puces
- ... Liste triée, liste à numéros
- <dl>...</dl> Liste des définitions

34

Les images

Les images

Inline



HTML

```

<body>
    <p>
        Je suis content !!
        
    </p>

```

• Soit un **chemin absolue** de l'image (commençant par « [http://](#) ») pour pointer vers un autre site, par exemple...

32

Liste non triée

Les listes

- ... : Début et fin d'une liste
- ... : Début et fin d'un des points de la liste

Une liste non ordonnée ressemble à ceci :

HTML

```

<ul>
    <li>Fraises</li>
    <li>Framboises</li>
    <li>Cerises</li>
</ul>

```

! Retenez donc ces deux balises délimite toute la liste ;
**** délimite un élément de la liste (une puce)

35

Les liens

Les liens

Inline

<a>... : Faire un lien vers une image ou un fichier.

texte à cliquer

HTML

```

<body>
    <p>
        Pour plus d'informations,
        <a href="http://www.smartfutur.com"> cliquez ici ! </a>
    </p>

```

Lien vers un mail

Ecrivez-moi !

33

Les liste triée

Les listes

- ... : Début et fin d'une liste
- ... : Début et fin d'un des points de la liste

Une liste ordonnée ressemble à ceci :

HTML

```

<ol>
    <li>Ma journée</li>
    <li>Je me lève.</li>
    <li>Je mange et je bois</li>
    <li>Je retourne me coucher</li>
</ol>

```

! Retenez donc ces deux balises délimite toute la liste ;
**** délimite un élément de la liste (chiffre français)

36

Exercices d'application

Ecrire les balises html permettant d'obtenir le résultat suivant :

```
<ul>
  <li>avril</li>
  <li>mai</li>
  <li>juin</li>
</ul>
```

37

Attribut : rowspan

Extension de cellule sur L lignes :

```
<td rowspan="L">
```

1	1
2	2
3	3
4	4
5	5

40

Les tableaux

Les balises `<td>...</td>` indiquent le début et la fin d'une colonne

```
<body>
  <table>
    <tr>
      <td>Ligne 1 - Colonne 1</td>
      <td>Ligne 2 - Colonne 2</td>
    </tr>
    <tr>
      <td>Ligne 2 - Colonne 1</td>
      <td>Ligne 2 - Colonne 2</td>
    </tr>
  </table>
</body>
```

39

Attribut : colspan

Extension de cellule sur C colonnes :

```
<td colspan="C">
```

1	1	
2	2	4
3	3	5

41

Les tableaux

Tableau : `<table> ... </table>`
Ligne : `<tr> ... </tr>`
Cellule : `<td> ... </td>`
Cellule d'en-tête : `<th> ... </th>`
Colonnes = nombre maxi de `<td>` par `<tr>`

```
<table>
  <tr><td>1</td><td>2</td></tr>
  <tr><td>3</td><td>4</td><td>5</td></tr>
</table>
```

```
<table>
  <tr><td>1</td><td>2</td></tr>
  <tr><td>3</td><td>4</td><td>5</td></tr>
</table>
```

Equivalent au précédent

39

Exercice d'application

But du jeu : créer une page – CV avec :

- Un tableau 1 ligne/ 2 colonnes avec :
 - Votre nom en titre 1 dans la colonne de droite
 - Une photo de vous dans la colonne de gauche
- Un paragraphe « état Civil » avec les informations sous forme de liste
- Un paragraphe « formations » avec des liens vers les écoles/universités, ...

Foulen ben Foulen

Moi

- Nom : Foulen
- Adresse : Tunis
- Email : Foulen@gmail.com

Ma vie

- 1987 : Fin d'étude primaire
- 1995 : Baccalauréat
- 1999 : Maîtrise

42

Formulaire

Insertion d'un formulaire

- Un formulaire est inséré dans une page HTML par la balise double <FORM>
- En plus de ses éléments principaux, le formulaire peut contenir du texte
- Toutes les mises en forme peuvent être appliquées aux éléments du formulaire
- Plusieurs formulaires peuvent être insérés dans la même page

46

Formulaire

Fonctionnement

Attribut : ACTION

```
<FORM ACTION="script" METHOD="post" NAME ="nom_form">
```

- **ACTION:**
 - URL du script ou prog à exécuter Chemin du script à exécuter sur les données du formulaire
ACTION= "chemin_script"

47

Formulaire

Traitement des formulaires

- Comment utiliser un formulaire ?

Réalisation du formulaire (Simple)

En HTML

Exploitation du formulaire par un script

- CGI
- PHP
- JSP
- ASP
- ...

45

Attribut : METHOD

```
<FORM ACTION="script" METHOD="post" NAME ="nom_form">
```

- **METHOD:** méthode d'envoi des données **GET** ou **POST**
 - GET : données du form envoyées avec l'URL du script (limitée)
 - POST: données du form envoyées séparément de l'URL (meilleure)

NAME : nom du formulaire

48

La balise : <input>

Formulaire

➤<INPUT> : - crée une zone de saisie
- Balise monolithique

➤Principaux attributs

- NAME : nom du champ
- SIZE : largeur du cadre réservé au champ
- MAXLENGTH: longueur du texte
- TYPE : (text, password, radio, checkbox, submit, reset, file, ...)
- VALUE : La valeur du champ

49

Les libellés

Formulaire

➤Cette zone de texte est bien jolie mais si votre visiteur tombe dessus, il ne sait pas ce qu'il doit écrire. C'est justement le rôle de la balise <label> :

```
<label> Votre nom : </label> <input type="text" name="nom" size=15 maxlength=30>
```

➤Pour lier le label au champ, il faut lui donner un attribut for qui a la même valeur que l'id du champ

```
<label for="nom" > Votre nom: </label> <input type="text" name="nom" id="nom" size=15 maxlength=30>
```

52

Zone de saisie : Exemple

Formulaire

HTML

```
Votre nom : <input type="text" name="nom" size=15 maxlength=30> <br> <br>
Votre prénom : <input type="text" name="prenom" size=15 maxlength=30>
```



50

Envoyer des données d'un formulaire

Formulaire

Pour envoyer le formulaire, il suffit de cliquer avec la souris sur un bouton spécial créé par la balise <input>

```
<input type="submit" value="Valider">
```

La valeur "submit" de l'attribut TYPE caractérise le bouton d'envoi
La valeur "Valider" de l'attribut VALUE définit le texte affiché sur le bouton d'envoi



53

Zone de texte : <TEXTAREA>

Formulaire

➤La balise <TEXTAREA> permet de créer une zone de saisie de texte

➤Principaux attributs : NAME, COLS, ROWS

COLS : nombre de colonnes
ROWS : nombre de lignes

Exemple :

Votre commentaire :
<TEXTAREA NAME="comment" rows=5 cols=40></TEXTAREA>



51

Effacer les données d'un formulaire

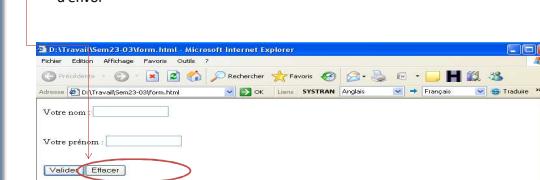
Formulaire

Après avoir rempli les données du formulaire, nous pouvons les effacer tous avec un bouton créé par la balise <input>

Avant l'envoi du formulaire

```
<input type="reset" value="Effacer">
```

La valeur "reset" de l'attribut TYPE caractérise le bouton de modification
La valeur "Effacer" de l'attribut VALUE définit le texte affiché sur le bouton d'envoi



Saisie d'un mot de passe

➤ L'attribut TYPE avec la valeur "password" permet de masquer un champs lors de la saisie, ses caractères sont remplacés par des points ou des étoiles

```
<label for="passwd" > Votre mot de passe: </label> <input type="password" name="passwd" id="passwd" size=15maxlength=30>
```

Authentification
Votre login :
Votre mot de passe : (highlighted by a red oval)
[Valider] [Modifier]
[Terminé] Poste de travail

55

Les zones d'option : CHECKED

➤ L'attribut CHECKED permet de cocher par défaut l'un des boutons

```
<input type="radio" name="nom_radio" value="valeur1" checked> texte du bouton1  
<input type="radio" name="nom_radio" value="valeur2"> texte du bouton2
```

Exemple

Notre association vous remercie pour votre participation

Votre nom :
Votre prénom :
Votre sexe : Homme Femme
Etes-vous fumeur ? : Oui Non
[Valider] [Modifier]

58

Formulaire

Les éléments d'option

➤ Ce sont des éléments qui demandent au visiteur de faire un choix parmi une liste de possibilités. On trouve :

- les zones d'options ;
- les cases à cocher ;
- les listes déroulantes.

56

Formulaire

Les cases à cocher

➤ Notation :

```
<input type="checkbox" name="nom_cases" value="valeur1"> texte case1  
<input type="checkbox" name="nom_cases" value="valeur2"> texte case2  
<input type="checkbox" name="nom_cases" value="valeur3"> texte case3
```

Notre association vous remercie pour votre participation

Votre nom :
Votre prénom :
Votre sexe : Homme Femme
Etes-vous fumeur ? : Oui Non
Vos proches sont-ils fumeurs ?
 Votre père
 Votre mère
 Un frère ou une soeur
 Votre meilleure(e) amie(r)e
[Valider] [Modifier]

59

Formulaire

Formulaire

➤ Notation

```
<input type="radio" name="nom_radio" value="valeur1"> texte du bouton1  
<input type="radio" name="nom_radio" value="valeur2"> texte du bouton2
```

➤ La valeur de l'attribut NAME est commune à toutes les zones

Votre sexe : <input type="radio" name="sex" value="Homme" checked=""> Homme <input type="radio" name="sex" value="Femme" checked=""> Femme

Notre association vous remercie pour votre participation
Votre nom :
Votre prénom :
Votre sexe : Homme Femme
[Valider] [Modifier]

57

Formulaire

Les zones à cocher

➤ Sélectionner un élément dans une liste

➤ Cocher avec la souris

➤ Possibilité de cocher plusieurs éléments

➤ Un clique sélectionne l'élément

➤ le 2 ème clique annule la sélection

Notre association vous remercie pour votre participation

Votre nom :
Votre prénom :
Votre sexe : Homme Femme
Etes-vous fumeur ? : Oui Non
Vos proches sont-ils fumeurs ?
 Votre père
 Votre mère
 Un frère ou une soeur
 Votre meilleure(e) amie(r)e
[Valider] [Modifier]

60

Formulaire

Les listes déroulantes

- Balise **<SELECT>**
- Principaux attributs : NAME, SIZE, MULTIPLE
- Sélectionner un élément dans un menu
- Sélection avec la souris

61

Formulaire

<FIELDSET> et <LEGEND>

- Ces deux balises permettent de structurer les formulaires.

```

<fieldset>
    <legend>Informations personnelles</legend>
        Nom :<input type='text'>
        Prénom :<input type='text'>
</fieldset>
<fieldset>
    <legend>Favoris</legend>
        Auteur favori :<input type='text'>
        Livre favori :<input type='text'>
</fieldset>

```

Informations personnelles
 Nom : Prénom :
 Favoris
 Auteur favori : Livre favori :

64

Formulaire

Les listes déroulantes

Exemple

```

Ville(s) visités :
<SELECT NAME="ville" >
<OPTION VALUE="RA"> Rabat
<OPTION VALUE="FE"> Fès
<OPTION VALUE="MA"> Marrakech
<OPTION VALUE="AG"> Agadir
<OPTION VALUE="TA"> Tanger
</SELECT>

```

Notez association vous remercie pour votre participation
 Votre nom :
 Votre prénom :
 Votre sexe : Homme Femme
 Êtes-vous fumeur ? Oui Non
 Vous prenez soin de fumer ?
 Vapez régulièrement
 Vapez occasionnellement
 L'avez arrêté
 Votre tabac(e) mal(e)
 Votre âge :
 Cas :
 Valider

62

Formulaire

Exercice d'application

Réaliser le formulaire ci-dessous :

- Les zones de saisies sont obligatoires

Vos commentaires

Aidez-nous à améliorer notre site:

Votre nom :

Vos commentaires :

Votre adresse de courrier électronique :

Indiquez les éléments du site que vous aimez:

Design du site Les liens Facilité de navigation Les images

65

Formulaire

Les listes déroulantes : attribut SIZE

- SIZE : indique le nombre de lignes visibles du menu
- Si SIZE < nb d'éléments du menu alors une barre de défilement apparaît
- Exemple

Notez association vous remercie pour votre participation
 Votre nom :
 Votre prénom :
 Votre sexe : Homme Femme
 Êtes-vous fumeur ? Oui Non
 Vous prenez soin de fumer ?
 Vapez régulièrement
 Vapez occasionnellement
 L'avez arrêté
 Votre tabac(e) mal(e)
 Votre âge :
 Cas :
 Valider

63

Formulaire

Les grandes nouveautés HTML5

- Une gestion des **formulaires** plus poussée
- Les nouvelles balises sémantiques (**<header>**, **<footer>** ...)
- Les balises multimédias (**<audio>** et **<vidéo>**)

66

HTML 5

Formulaire : nouveau champs

Les grandes nouveautés HTML 5

- **email** : pour saisir une adresse E-mail
- **url** : pour saisir une adresse absolue
- **tel** : Ce champ est dédié à la saisie de numéros de téléphone
- **number** : pour saisir un nombre entier
- **range** : permet de sélectionner un nombre avec un curseur
- **date** : pour saisir une date
- **search** : pour créer un champ de recherche

67

HTML 5

Formulaire : les attributs

Les grandes nouveautés HTML 5

- **required** : zone obligatoire
- **autofocus** : qui placera le focus dans le premier champ de saisie dès l'ouverture de la page
- **placeholder** : permet d'afficher un texte par défaut en grisé autocomplete
- **pattern** : permet de contrôler les données saisies

70

HTML 5

Formulaire : javascript inutile?

Les grandes nouveautés HTML 5

- <input type="email" />
- <input type="url" />
- <input type="tel" />
- <input type="number" />
- <input type="range" />
- email
- url ! Veuillez inclure "@" dans l'adresse e-mail. Il manque un symbole "@" dans "as".
- tel
- range
- number

68

HTML 5

Les nouvelles balises sémantiques

Les grandes nouveautés HTML 5

Les nouvelles balises sémantiques :

- <**header**>
- <**nav**>
- <**article**>
- <**section**>
- <**aside**>
- <**footer**>

71

HTML 5

Formulaire : manipuler une date

Les grandes nouveautés HTML 5

- <input type="date" />
- date : pour la date (05/08/1985 par exemple)
- time : pour l'heure (13:37 par exemple)
- week : pour la semaine
- month : pour le mois

69

HTML 5

Les nouvelles balises sémantiques

Les balises sémantiques

AVANT

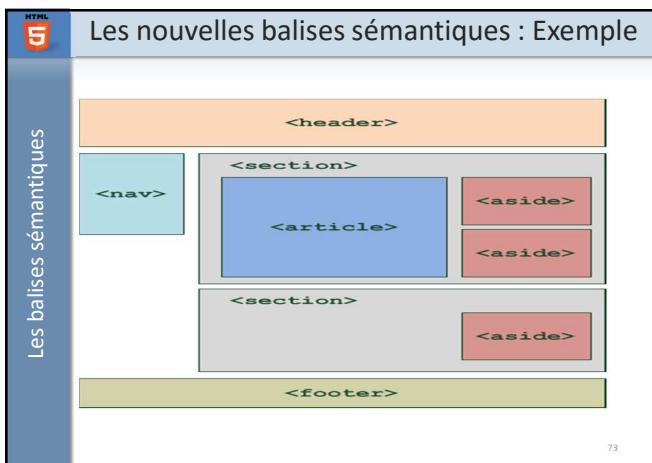
```
<body>
<div id="header"></div>
<div id="nav"></div>
<div class="article">
    <div class="section"></div>
    <div class="section"></div>
</div>
<div id="aside"></div>
```

Trop de DIV tue le DIV !

APRÈS

```
<body>
<header></header>
<nav></nav>
<article>
    <section></section>
    <section></section>
</article>
<aside></aside>
<footer></footer>
</body>
```

72



Header

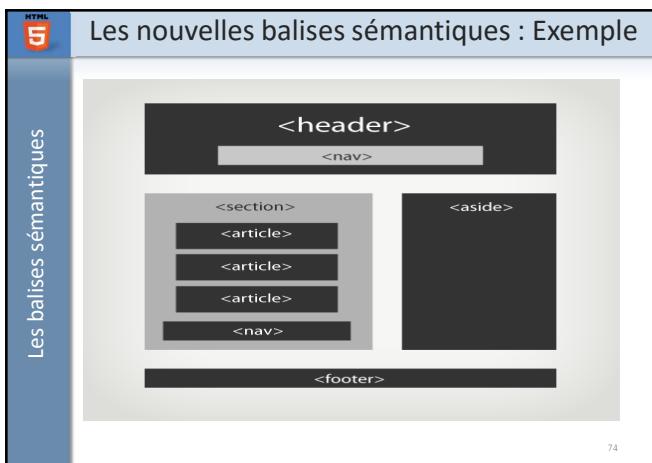
En-tête d'une section (section, article...) ou d'une page entière.

➤Sert à contenir un logo, un ou plusieurs titres, d'autres informations d'introduction, une navigation, un formulaire de recherche général.

```

<article>
  <header>
    <h1>Titre de l'article</h1>
    <p>Auteur : bidule</p>
  </header>
  <p>Contenu de l'article</p>
</article>
    
```

76



Footer

Pied de page, ou bien la conclusion d'une section.

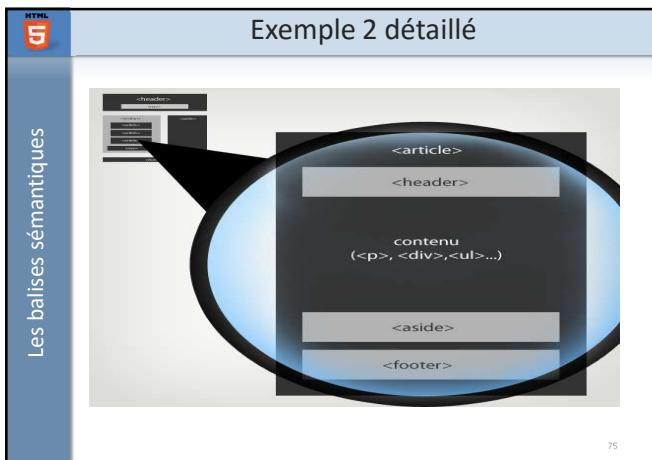
➤Peut contenir les informations concernant :

- l'auteur,
- une navigation ou une pagination (en combinaison avec <nav>),
- un logo de rappel,
- des coordonnées,
- des dates de publication.

```

<article> ...
<footer>
  <p>Posté par Foulon, le <time datetime="2012-10-12">
    12 octobre 2012</time></p>
</footer>
</article>
    
```

77



NAV

La balise <nav> doit regrouper tous les principaux liens de navigation du site. Vous y placerez par exemple le menu principal de votre site.

➤ Généralement, le menu est réalisé sous forme de liste à puces à l'intérieur de la balise <nav> :

```

<nav>
  <ul>
    <li><a href="index.html">Accueil</a></li>
    <li><a href="forum.html">Forum</a></li>
    <li><a href="contact.html">Contact</a></li>
  </ul>
</nav>
    
```

78

HTML 5

ARTICLE

C'est une spécialisation de section vise à baliser des blocs de contenu utiles que l'on pourrait extraire du document tout en conservant leur sens et leurs informations.

- C'est le cas par exemple des actualités (articles de journaux ou de blogs)
- une plus forte valeur sémantique.

```
<article>
<h1>titre de l'article</h1>
<p>Contenu de l'article</p>
</article>
```

79

HTML 5

Exemple

Les balises sémantiques

```
<body>
<header>
<h1>Recettes cuisine</h1>
<h2>Cuisine Tunisienne</h2>
</header>
<nav>
<ul>
<li><a href="#">Gâteaux</a></li>
<li><a href="#"> Pattes</a></li>
<li><a href="#"> SaladeS</a> </li>
</ul>
</nav>
<section>
<aside>
<h1>À propos de l'auteur</h1>
<p>C'est moi, Foulen ! Je suis né un 14 Janvier 1994.</p>
</aside>
```

Les balises sémantiques

```
<article>
<h1>Je suis un grand Chef de cuisine</h1>
<p>texte de l'article texte de l'article</p>
</article>
</section>
</footer>
<p>Copyright Foulen - Tous droits réservés<br />
<a href="#">Me contacter !</a>
</p>
</footer>
</body>
</html>
```

80

HTML 5

SECTION

C'est la plus générique

La section regroupe un ensemble de contenu d'une même thématique,

- La section englobe généralement une portion du contenu au centre de la page. Il contient naturellement un titre d'introduction

```
<section>
<h1>Articles</h1>
<article>
<p>texte texte texte texte </p>
</article>
</section>
```

80

HTML 5

Les nouvelles balises `<audio>` et `<video>`

Les balises audio et vidéo

La lecture audio et vidéo avec `<audio>` et `<video>`




83

HTML 5

ASIDE

Les balises sémantiques

Contient des informations complémentaires au document que l'on visualise

Par exemple :

- la définition d'un terme,
- préciser des sources,
- une liste d'articles en relation...

```
<aside>
<h4>Sources de l'article</h4>
<ul>
<li><a href="#">Lien 1</a></li>
<li><a href="#">Lien 2</a></li>
<li><a href="#">Lien 3</a></li>
</ul>
</aside>
```

81

HTML 5

Les nouvelles balises `<audio>` et `<video>`

Les balises audio et vidéo

- Une des **grandes nouveautés** de l'HTML 5 est la prise en charge **sans plugins**, de la lecture des flux **audio** et **vidéo**
- Fondamentalement, les nouvelles balises `<audio>` et `<video>` se comportent de la même manière et s'utilisent quasiment de la **même façon** que l'inclusion d'une simple image !

- syntaxe minimale pour la lecture d'un fichier **vidéo** :

```
<video src="video.webm"></video>
```
- syntaxe minimale pour la lecture d'un fichier **audio** :

```
<audio src="audio.mp3" controls></audio>
```
- syntaxe minimale pour l'inclusion d'une **image** :

```

```

84

HTML 5

La nouvelle balise `<video>`

Les balises audio et vidéo

- La simple syntaxe `<video src="video.webm"></video>` nous offre donc la possibilité de lire une vidéo directement dans notre navigateur :



Aperçus dans Google Chrome

- Cependant, l'absence de l'attribut **controls** implique que la lecture, la pause et le stop doivent s'effectuer avec le clique droit ce qui n'est pas très pratique....

85

HTML 5

La nouvelle balise `<audio>`

Les balises audio et vidéo

- La syntaxe `<audio src="audio.mp3" controls></audio>` nous offre la possibilité de lire un fichier audio directement dans notre navigateur tout comme pour la vidéo.

- Aperçus dans les principaux navigateurs :



88

HTML 5

La nouvelle balise `<video>`

Les balises audio et vidéo

- Code source d'un élément `<video>` plus complet avec le contrôle, la taille, l'image d'intro, les différentes sources et le texte alternatif.

```
<video width="360" height="640" poster="image.jpg" controls>
<source src="video.mp4" type="video/mp4" />
<source src="video.webm" type="video/webm" />
<source src="video.ogv" type="video/ogg" />
l'alternative à la vidéo : un lien de téléchargement, un message, etc.
</video>
```



Aperçus dans Mozilla Firefox

- Exactement la même vidéo que précédemment mais avec l'**image d'intro** et le **contrôleur** visible par le client.

La balise `<video>` possède également l'attribut **autoplay**, **preload** et **loop**

86

HTML 5

La nouvelle balise `<audio>`

Les balises audio et vidéo

- De même que pour l'élément `<video>`, nous pouvons avec l'élément `<audio>` gérer plusieurs sources pour répondre à l'incompatibilité des navigateurs.

```
<audio controls>
<source src="audio.oga" type="audio/ogg" />
<source src="audio.mp3" type="audio/mpeg" />
<source src="audio.acc" type="audio/mp4" />
l'alternative à l'audio: un lien de téléchargement, un message, etc.
</audio>
```

La balise `<audio>` possède également l'attribut **autoplay**, **preload** et **loop**

89

HTML 5

Les attributs de la balise `<video>`

Les balises audio et vidéo

- **controls** : pour ajouter les boutons « Lecture », « Pause » et la barre de défilement
- **width**, **height**.
- **loop** : (jouer en boucle.)
- **autoplay** : la musique sera jouée dès le chargement de la page.
- **preload** (on ne peut pas le forcer, ça dépend du nav.)
 - **auto** (par défaut)
 - **metadata** : charge uniquement les métadonnées (durée, dimensions, etc.).
 - **none** : pas de préchargement. Utile si vous nous voulez pas gaspiller de bande passante sur votre site.

87

HTML 5

Les attributs de la balise `<audio>`

Les balises audio et vidéo

- **controls** : pour ajouter les boutons « Lecture », « Pause » et la barre de défilement
- **width** : pour modifier la largeur de l'outil de lecture audio.
- **loop** : la musique sera jouée en boucle.
- **autoplay** : la musique sera jouée dès le chargement de la page.
- **preload** (on ne peut pas le forcer, ça dépend du nav.)
 - **auto** (par défaut)
 - **metadata** : charge uniquement les métadonnées (durée, dimensions, etc.).
 - **none** : pas de préchargement. Utile si vous nous voulez pas gaspiller de bande passante sur votre site.

90

AYMEN SELLAOUTI
aymen.sellaouti@gmail.com

The slide has a light blue header section. The title 'CSS 3' is centered in a large, bold, dark blue font. Below the title is a white rectangular area containing a large blue icon. The icon features the letters 'CSS' in a black sans-serif font above a stylized blue '3' composed of three nested, rounded blue shapes. To the right of this white area is a vertical column with a light blue background. A hand is visible on the right side, holding a pen and drawing a hierarchical tree structure on a whiteboard.

Principe de CSS

Pages personnelles de David NOFL

Pages personnelles de David NOFL

- ACCUEIL
- CSS
- INFORMATIONS
- CONTACT
- Actualités
- Archives
- Liens utiles
- Enseignement
- Recherche
- CV

Rechercher

Sur le site de l'ÉTSQCS

Français

David NOFL enseignant en Masterinfo à l'ÉTSQCS du Campus Nancy - Identification du comportement d'échange de balises (XHTML et applications de leur mise en évidence) Exemples :
 2005-2011 Maître de l'Ecole Doctorale de Nancy - Coordonnées de l'Institut et de l'Université
 2006-2009 Membre de l'Agence nationale de la recherche (ANR) - Coordination de la recherche de fonds pour le projet "Modélisation et optimisation de l'interprétation et de l'exploitation des connaissances dans les systèmes de lecture et d'apprentissage".
 2006-2010 Membre de l'Ecole Doctorale Supérieure de l'Université de Lorraine - Coordonnées de l'Institut et de l'Université
 2006-2011 Membre de l'Ecole Doctorale Supérieure de l'Université de Lorraine - Coordonnées de l'Institut et de l'Université
 2010 Stage en recherche à l'ÉTSQCS (Nancy) - Modélisation du comportement d'échange des balises (XHTML)
 2005-2011 Responsable de l'ETCSQCS Nancy - encadrement de projets de conception et en cours 5202 Datez aussi
 2012 Ingénierie Conceptuelle au sein de la plate-forme d'ETCSQCS

Page sans CSS

Pages personnelles de David NOFL

Pages personnelles de David NOFL

ACCUEIL

CSS

INFORMATIONS

CONTACT

Actualités

Archives

Liens utiles

Enseignement

Recherche

CV

Rechercher

Sur le site de l'ÉTSQCS

Français

David NOFL enseignant en Masterinfo à l'École Doctorale de Nancy - Identification du comportement d'échange de balises (XHTML et applications de leur mise en évidence) Exemples :
 2005-2011 Maître de l'Ecole Doctorale de Nancy - Coordonnées de l'Institut et de l'Université
 2006-2009 Membre de l'Agence nationale de la recherche (ANR) - Coordination de la recherche de fonds pour le projet "Modélisation et optimisation de l'interprétation et de l'exploitation des connaissances dans les systèmes de lecture et d'apprentissage".
 2006-2010 Membre de l'Ecole Doctorale Supérieure de l'Université de Lorraine - Coordonnées de l'Institut et de l'Université
 2006-2011 Membre de l'Ecole Doctorale Supérieure de l'Université de Lorraine - Coordonnées de l'Institut et de l'Université
 2010 Stage en recherche à l'ÉTSQCS (Nancy) - Modélisation du comportement d'échange des balises (XHTML)
 2005-2011 Responsable de l'ETCSQCS Nancy - encadrement de projets de conception et en cours 5202 Datez aussi
 2012 Ingénierie Conceptuelle au sein de la plate-forme d'ETCSQCS

Page avec CSS



Plan de la présentation

Plan

- Principe de base
- Où écris on le css?
- Formatage du texte
- La couleur du fond
- Les bordures et les ombres
- Les sélecteurs
- Notion de class et id
- Pseudo calss et pseudo élément
- Les modèle de boîtes
- Flux et positionnement





Principe de CSS

- Le CSS est un langage permettant la mise en page d'une page web.
- Attention à ne pas confondre :
 - **Le HTML** : dit le **contenu** de ce qu'il y a sur la page
 - Je veux un titre,*
 - Je veux une image,*
 - Je veux un paragraphe qui dit ça, ça, ça...*
 - **Le CSS** : dit **comment afficher** ce qu'il y a sur la page
 - Je veux que les titres soient centrés,*
 - Je veux qu'une image soit encadrée en vert,*
 - Je veux que le fond soit rouge...*



Les sélecteurs

Principe de CSS

Les sélecteurs sont des caractères alphanumériques qui identifient la règle.

- Sélecteur HTML
- La classe
- L'ID

 Règle CSS : présentation du code

Principe de CSS

- Présentation possible :

```
h2 {color: red; text-align: center ;}
```

- Présentation conseillée (avec indentation et passage à la ligne) :

```
h2 {
  color: red;
  text-align: center ;
}
```

Bonne Pratique

7

 Du CSS à travers un fichier CSS

Où écrit on le CSS

- Pour qu'une page HTML puisse utiliser un fichier CSS il faut le référencer dans la partie <head> avec la balise **link**

```
<link rel="stylesheet" href="style.css" />
```

10

 Les commentaires

Principe de CSS

Comme en HTML, il est possible de mettre des **commentaires**.

Pour faire un commentaire, on tape :

```
/*suivi du commentaire */
```

Vos commentaires peuvent être sur une ou plusieurs lignes.

Par exemple :

```
/* essai.css
   Par Flen
   Fichier créé le jour j */
```

```
p { color: blue; /* Les paragraphes seront bleus */ }
```

8

 Du CSS dans le <head> du fichier HTML

Où écrit on le CSS

- On peut intégrer à l'aide de la balise head du code CSS sans passer par une page CSS

```
<head>
  <meta charset="utf-8" />
  <style>
    h1
    {
      color: red;
    }
  </style>
</head>
```

11

 titre

Où écrit on le CSS

On peut introduire un code en langage CSS de 3 façons :

- ✓ Dans un fichier CSS ;
- ✓ Dans l'en-tête du fichier HTML (<head>) ;
- ✓ Directement dans les balises via l'attribut style.

9

 Du CSS directement dans les balises

Où écrit on le CSS

- C'est la manière la moins recommandée

```
<html>
  <body>
    <p style="color:red;">Je ne suis pas recommandé</p>
  </body>
</html>
```

12



Formatage du texte

Formatage du texte

- La taille
- La police
- Styles du texte
- L'alignement
- Les flottants

13

Une valeur relative



- C'est la méthode recommandée car le texte s'adapte alors plus facilement aux préférences de tous les visiteurs.

valeur relative :

xx-small : minuscule ;
x-small : très petit ;
small : petit ;
medium : moyen ;
large : grand ;
x-large : très grand ;
xx-large : gigantesque.

Il existe que 7 tailles ?!!!

16



La taille : font-size

Formatage du texte

Balise { font-size : valeur }

- **taille absolue** : valeur en pixels, en centimètres ou millimètres.
- **taille relative** : valeur en pourcentage, « em » ou « ex »,

14



Une valeur relative

La taille

il existe d'autres moyens ..

- **em** :
1em, le texte a une taille normale.
p { font-size: 0.8em; }
- **ex** : qui fonctionne sur le même principe que le em mais qui est plus petit de base
- **le pourcentage** : (80%, 130%...).

17

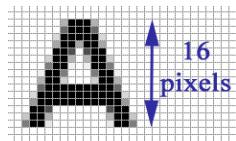


Taille absolue

La taille

Balise{ font-size : valeur px }

Exemple : font-size: 16px;



Une lettre de 16 pixels de hauteur

15



La police : font-family

La police

balise { font-family: police; }

- pour éviter les problèmes si l'internaute n'a pas la même police que vous, on précise en général plusieurs noms de police, séparés par des virgules :

balise { font-family: police1, police2, police3, police4; }

- En général, on indique en tout dernier **serif** ou **sans-serif** (police par défaut)

18

Liste des polices

➤ liste de polices qui fonctionnent bien sur la plupart des navigateurs

Arial ; Arial Black ; Comic Sans MS ; Courier New ; Georgia ; → Impact ; Times New Roman ; Trebuchet MS ; Verdana.	Texte en Arial Texte en Arial Black Texte en Comic Sans MS Texte en Courier New Texte en Georgia Texte en Impact Texte en Times New Roman Texte en Trebuchet MS Texte en Verdana
---	--

Alignement : à gauche, centré, à droite et justifié

text-align : left| center| right| justify

➤ Valeur :

- left : le texte sera aligné à gauche (par défaut).
- center : le texte sera centré.
- right : le texte sera aligné à droite.
- justify : le texte sera « justifié ».

➤ Exemple :

```
h1 { text-align: center; }
p { text-align: justify; }
```

22

Exemple

La police

```
p { font-family: Impact, "Arial Black", Arial, Verdana, sans-serif; }
```

Entre " " car il y'a un espace

si rien n'a marché, mets une police standard sans-serif

20

Les flottants

La police

➤ Le CSS nous permet de faire flotter un élément autour du texte.

➤ On dit aussi qu'on fait un « **habillage** ».

Exemple :

une image flottante entourée par du texte :



...Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec vitae lorem imperdiet lacus molestie molestie. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec eu purus. Phasellus metus lorem, blandit et, posuere quis, tincidunt vitae, ante. Vivamus consequat mauris a diam. Vivamus nibh erat, hendrerit nec, aliquet ut, hendrerit quis, nunc. Vestibulum et turpis et elit tempor euismod.

23

Italique, gras, souligné

Styles du texte

➤ Mettre en italique
font-style : normal| oblique| italic

➤ Mettre en gras
font-weight : normal| bold

➤ Soulignement et autres décos
text-decoration : underline |line-through| overline| none

21

Propriétés float et clear

Les flottants

➤ **Float** peut prendre deux valeurs très simples :

- left : l'élément flottera à gauche.
- right : l'élément flottera à droite

Exemple

```
img {
  float: left
}
```

➤ **clear**, une propriété qui permet de dire : « *Stop, ce texte doit être en-dessous du flottant et non plus à côté* ». Peut prendre trois valeurs :

- left : le texte se poursuit en-dessous après un **float: left**;
- right : le texte se poursuit en-dessous après un **float: right**;
- both : le texte se poursuit en-dessous, que ce soit après un **float: left**; ou après un **float:right**.

24



La couleur et le fond

La couleur et le fond

- La couleur du texte : *color*
- La couleur du fond : *background-color*
- Image du fond : *background-image*

25

Les bordures standards : border

Les bordures et les ombres

- Border peut utiliser jusqu'à 3 valeurs pour modifier l'apparence de la bordure :
- **La largeur** : en pixels (comme 2px).
- **La couleur** : nom de couleur (black, red,...), soit une valeur hexadécimal (#FF0000), soit une valeur RGB (rgb(198, 212, 37)).
- **Le type de bordure** : (none | solid| dotted|double.....)

➤ Exemple :

```
h1 { border: 3px blue dashed; }
```

28



Les seize noms de couleurs utilisables en CSS

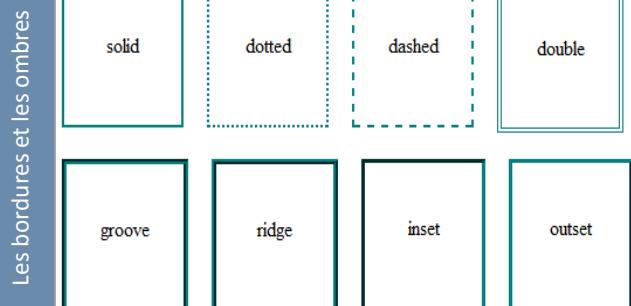
La couleur et le fond

white	
silver	
gray	
black	
red	
maroon	
lime	
green	
yellow	
olive	
blue	
navy	
fuchsia	
purple	
aqua	
teal	

26



Les différents types de bordure



29



Color, background-color et background-image

La couleur et le fond

- **Color** : définit la couleur du texte par exemple en hexadécimal
`H3 {color: #000080;}`
- **background-color** : définit la couleur de l'arrière-plan
`H1 {background-color: #000000;}`
- **background-image** : définit l'image de l'arrière-plan URL de l'image
`Body {background-image: image.gif;}`

27



Exercice d'application

Les bordures et les ombres

- Ecrire le propriétés css permettant de :
 - ❖ Augmenter la taille des paragraphes à 150%
 - ❖ Encadrer les titres de niveau 1 avec un trait simple de couleur rouge
 - ❖ Souligner les titres de niveau 2 et changer la couleur du texte en vert. –

30

border : en haut, à droite, à gauche, en bas

Les bordures et les ombres

- Les quatre cotés :
 - border-top** : bordure du haut ;
 - border-bottom** : bordure du bas ;
 - border-left** : bordure de gauche ;
 - border-right** : bordure de droite.

➤ Exemple :

```
p { border-left: 2px solid black;
    border-right: 2px solid black; }
```

➤ On peut modifier les bordures d'une image

31

Nouveauté de CSS3 : L'ombre du texte

Les bordures et les ombres

- **text-shadow** : l'ombre du texte
Avec **text-shadow**, vous pouvez ajouter une ombre directement sur les lettres de votre texte
Les valeurs fonctionnent exactement de la même façon que **box-shadow** : décalage, adoucissement et couleur.

Code : CSS

```
p { text-shadow: 2px 2px 4px black; }
```

Résultat :

34

Nouveauté de CSS3: Bordures arrondies

Les bordures et les ombres

- **border-radius** : arrondir facilement les angles de n'importe quel élément. Il suffit d'indiquer la taille (« l'importance ») de l'arrondi en pixels :

```
p { border-radius: 10px; }
```

```
p { border-radius: 10px 5px 10px 5px; }
```

Les valeurs correspondent aux angles suivants dans cet ordre : en haut à gauche ;
en haut à droite ;
en bas à droite ;
en bas à gauche ;
en bas à gauche.

32

Les sélecteurs

Les sélecteurs

Les principaux sélecteurs sont :

- **Nom d'élément** : **h1**
- **Sélecteur multiple** (regroupement de sélecteurs) : **h1, h2, p**
- **Sélecteur contextuel** (combinaison d'éléments imbriqués) **h1 a**
- **Identifiant** : **p#intro1**
#intro1
- **Classe** : **p.intro**
.intro
- **Pseudo-classe** : **a:hover**
- **Sélecteur universel** : *****

35

Nouveauté de CSS3 : Les ombres des boîtes

Les bordures et les ombres

- La propriété **box-shadow** s'applique à tout le bloc et prend quatre valeurs dans l'ordre suivant :
 - le décalage horizontal de l'ombre ;
 - le décalage vertical de l'ombre ;
 - l'adoucissement du dégradé ;
 - la couleur de l'ombre.

➤ Exemple

```
p { box-shadow: 6px 6px 0px black; }
```

33

Nom de l'élément

Les sélecteurs

La règle css s'applique à toutes les balises HTML indiquées

Exemples :

```
h1 {
  color: red;
  font-size: 2em;
}
```

```
h2 {
  color: red;
  font-size: 1.5em;
}
```

```
p {
  color: black;
}
```

36

 Regroupement de sélecteurs du HTML (avec virgule)

Les sélecteurs

```

h1 {color:red;
    font-size:1,2em}      ↔      h1, h2, p {color:red;}
h2 {color:red;}          h1 {font-size:1,2em;}
p {color:red;}           
```

37

 Les sélecteurs d'enfants

A >B : une balise enfant direct de l'autre

Exemple: **ol>li**
Li enfant direct de Ol

Dans la partie CSS
ol>li {color :red}

Dans le body

```

<ol>
  <li>titre1</li>
    <ul> <li>titre1.1</li>
      <li>titre1.2</li>
    </ul>
  </ol>
  
```

40

 Sélecteur universel *

Les sélecteurs

* remplace tout élément du document
Exemple :
*** {color: black}**

Tous les éléments du document (h1, h2, p, div, ul...) auront une couleur de police noire.

38

 Sélecteur adjacent

A + B : une première balise B qui suit A

Exemple : **h3+ p**
Sélectionne la première balise **p** située après un titre **h3**..

Dans la partie CSS
h3+p {color: red;}

Dans le body

```

<h3>Titre</h3>
<p>Paragraphe1</p>
<p>Paragraphe2</p>
  
```

41

 Les sélecteurs descendants

Les sélecteurs

A B : une balise contenue dans une autre

Exemple : **h1 em**
h1 est un ancêtre de em, même lointain.
em est un descendant de h1, même lointain.

Dans la partie CSS
h1 em {color: red;}

Dans le body

```

<h1>Titre A1</h1>
<p>Texte A </p>
<h1>Titre B1 <em> Titre C1</em>Titre D1 </h1>
<p>Texte B <em> Texte C</em> Texte D </p>
  
```

39

 Les sélecteurs d'attribut

A[attribut] : une balise qui possède un attribut

Exemple : **a[title]**
Sélectionne tous les liens **a** qui possèdent un attribut title.

Dans la partie CSS
a[title]{color: red;}

Dans le body

```

<a href="http://site1.com" title="Infobulle"> Lien1</a>
<br>
<a href="http://site2.com" > Lien2</a>
  
```

42

Sélecteur de valeur d'attribut

A[attribut="Valeur"] : une balise, un attribut et une valeur exacte

Exemple : `a[title="Cliquez ici"]`

Idem, mais l'attribut doit en plus avoir exactement pour valeur « Cliquez ici »

Dans la partie CSS

```
a[title="Cliquez ici"] {color: red;}
```

Dans le body

```
<a href="http://site.com" title="Infobulle " >Lien1</a>
<a href="http://site1.com" title="Cliquez ici">Lien2</a>
```

43

Syntaxe générale d'une classe CSS

Les sélecteurs

46

Sélecteur de valeur d'attribut

A[attribut="Valeur"] : une balise, un attribut et une valeur exacte

Exemple : `a[title*="ici"]`

Idem, l'attribut doit cette fois contenir dans sa valeur le mot « ici » (peu importe sa position).

Dans la partie CSS

```
a[title*="ici"] {color: red;}
```

Dans le body

```
<a href="http://site.com" title="Quelque part par ici"> Lien</a>
```

44

La notion de class

Les sélecteurs

- Class est un attribut "générique" qui s'applique à toutes sortes d'éléments.
- Une classe permet de définir un sous-ensemble.
- Elle peut s'appliquer à plusieurs éléments.

Elle s'ajoute dans une balise du document html :

```
<h1 class="intro" >Titre 1</p>
<p class="intro">Texte 2</p>
<p class="intro">Texte 3</p>
<img class="logo" src=image/img1.jpg>
```

Utilisation dans une règle CSS :

```
.intro {font-style: italic; color: brown}
h1.intro {text-align: center;}
.logo {border: 1px solid red ;}
```

47

L'identifiant id

Les sélecteurs

- id est un attribut "générique" qui s'applique à toutes sortes d'éléments.
- Il sert à identifier une balise précise.
- Il doit être unique dans un document.

Elle s'ajoute dans une balise du document html :

```
<p id="intro">texte d'introduction</p>

```

Utilisation dans une règle CSS :

```
#intro {font-style: italic; text-align: center;}
#logo {
    /* Mettez les propriétés CSS ici */
}
```

48

Les sélecteurs

`.nav ul ul{color:red;}`

```
<div class="nav">
    <ul>
        <li>nom
            <ul>
                <li>CHERIF</li>
                <li>Ben Saleh</li>
            </ul>
        </li>
        <li>prenom
            <ul>
                <li>Lina</li>
                <li>Mohamed</li>
            </ul>
        </li>
    </ul>
</div>
```

48

Bilan

Les sélecteurs

HTML

```
<BODY>
<ul class="niveau1">
    <li>Exemple 1</li>
    <li>Liste simple</li>
</ul>
<div class="italic_rouge">
    Exemple 2
</div>
</BODY>
```

.HTML

Le Résultat

- Exemple 1
- Liste simple

Sous Class dépendante

CSS

```
ul.niveau1
{
list-style-type:square;
color:blue;
}
```

Sous Class indépendante

CSS

```
.italic_rouge
{
    color:red;
    font-style:italic;
}
```

49

DIV : Exemple

Les sélecteurs

HTML

```
<HEAD>
<STYLE>
.zone{
    font-size: x-small;
}
</STYLE>
</HEAD>
<BODY>
    La balise DIV
    <DIV class=zone>
        <P>Commentaire :</P>
        <P>N'oubliez pas l'attribut class!</P>
    </DIV>
</BODY>
</HTML>
```

52

Bilan

Les sélecteurs

HTML

```
<BODY>
<ul class="niveau1">
    <li>Exemple 1</li>
    <li>Liste simple</li>
</ul>
<div class="italic_rouge">
    Exemple 2
</div>
</BODY>
```

.HTML

Le Résultat

- Exemple 1
- Liste simple
- Exemple 2**

Sous Class dépendante

CSS

```
ul.niveau1
{
list-style-type:square;
color:blue;
}
```

Sous Class indépendante

CSS

```
.italic_rouge
{
    color:red;
    font-style:italic;
}
```

50

SPAN: Exemple

Les sélecteurs

HTML

```
<!DOCTYPE Html >
<HTML>
<HEAD>
    <TITLE>14 Janvier</TITLE>
    <STYLE>
        SPAN { float: left;
            font-family: Arial;
            font-size: 40pt }
    </STYLE>
</HEAD>
<BODY>
    <H2> Les Tunisiens fêtent le 14 Janvier sur fond de crise</H2>
    <SPAN>L</SPAN>e centre-ville de Tunis a été envahi par la foule,
    samedi, dès les premières heures de la journée. A chacun sa
    petite musique. .....
</BODY>
</HTML>
```

53

DIV et SPAN

Les sélecteurs

➤ **div et span** sont deux balises neutres.

➤ **div et span** sont en général associés à un id ou une class.

➤ **div** : balise de bloc

```
<ul><li>.....</li>...</ul>
<div id="piedpage">
    <p>.....</p>
    <p>.....</p>
</div>
```

➤ **span** ("petite étendue") : balise en-ligne, délimite une partie de texte

```
<p>Il prend un pot de <span
id="rouge">peinture rouge</span> et
un pinceau.</p>
```

51

Les pseudo classes

Les sélecteurs

➤ Règles affectant les liens :

A:link { color: red } /*liens non visité*/

A:active { color: maroon } /*lien pendant que l'on click,dessus)

A:visited { color: green } /*liens visités*/

A:hover {font-size:300%} /*Au survol*/

A:focus /*(état d'un élément qui a reçu «l'attention», par exemple un lien lorsqu'on y accède grâce au clavier, ou un champ texte d'un formulaire lorsqu'on clique dedans);*/

54

Les pseudo éléments

➤ Règles affectant le 1er caractère ou la 1ère ligne de chaque paragraphe :
*/*lettrine" taille 3x sup. au texte*/*

```
P:first-letter
{ font-size: 300% ;
  float: left ;
  color: green
}

/*1ère ligne en petites majuscules */

P:first-line { font-style: small-caps }
```

55

Ajouter les bordures : border

Les styles du tableau

Nom	Age	Pays
Imen	33ans	Sfax
Ahmed	26 ans	Mahdia

table,td,tr {

```
border: 1px solid black;
/* auront une bordure de 1px */}
```

56

Exercice d'application

Chapitre : CSS

Donner le code HTML et CSS permettent d'obtenir ce résultat

Les sélecteurs

- partie 1 : Les sélecteurs
 - Les selecteurs simples
 - les selecteurs Combinés
 - notion de class et id
- partie 2 : DIV et SPAN
 - La balise DIV
 - Exemple
 - La balise Span
- partie 3 : pseudo-classes
 - Règles affectant les liens
 - Exemple
 - Règles affectant les paragraphes

56

Coller les bordures : border-collapse

Les styles du tableau

Nom	Age	Pays
Imen	33ans	Sfax
Ahmed	26 ans	Mahdia

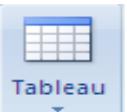
th,td{

```
border: 1px solid black;
}
table {
  border-collapse: collapse;
```

56

Les styles du tableau

➤ Margin
 ➤ Padding
 ➤ Background-color , background-image
 ➤ Color,font-size, text-decoration,font-family....
 ➤ width, height
 ➤ Border
 ➤ Border-collapse
 ➤ Caption-side



Les styles du tableau

Le style du titre du tableau: caption-side

Les styles du tableau

Nom	Age	Pays
Imen	33ans	Sfax
Ahmed	26 ans	Mahdia

caption{ caption-side :bottom; }

56

Les sélecteurs

Exercice d'application

- Ecrire les codes HTML et CSS permettant d'obtenir le résultat suivant:

61



Exemple (2/6)

Marge interne et marge externe

```
h1 {  
margin: 30px ;color:#000000;  
border-right: 5px solid black ;  
border-top: 5px solid #000000 ;  
padding-left: 40%;  
margin-right: 10px ;  
}  
  
belle dans mon miroir je suis la plus belle dans mon miroir je suis la plus belle dans mon miroir je suis la plus belle dans mon miroir  
  
je suis la plus belle dans mon miroir
```

The diagram illustrates the relationship between CSS margin and padding properties and the layout of a container block. It features a large blue dashed rectangle labeled "Bloc Conteneur". Inside this rectangle, there is a smaller black dashed rectangle representing the element's content area. The space between the content area and the outer border of the container is labeled "Marge externe (Margin-top)" at the top and "Marge externe (Margin-left)" on the left, each indicated by a blue double-headed arrow. Within the content area, there are three rows of text: "Css Css Css Css Css Css" repeated three times. The space between these rows is labeled "Marge interne (Padding-right)" on the right, indicated by a red double-headed arrow. At the bottom of the content area, there is a red double-headed arrow labeled "Marge interne (Padding-bottom)".

Marge interne et marge externe

Bloc Conteneur

Marge externe (Margin-top)

Marge externe (Margin-left)

Marge interne (Padding-right)

Marge interne (Padding-bottom)

 CSS

Exemple (4/6)

Marge interne et marge externe

```
h3 {  
margin-left: 15px ;  
padding-left: 5px ;  
border-bottom: 1px solid #9b2 ;  
border-left: 3px solid #9b2 ;  
color: #9b2 ;  
}
```

mon miroir

je suis la plus belle dans mon miroir

Marge interne et marge externe

Exemple (5/6)

margin: 50px 100px ;

Exemple

Image de fond en CSS

```
<html>
<head>
<style>
html {
background-color: black;
background-image: url('roger_rabbit.jpg');
color: white;
}
</style>
</head>
<body>
<p>Roger Rabbit: Yeah ...</p>
</body>
</html>
```

70

Marge interne et marge externe

Exemple (6/6)

padding-right: 200px ;

background-repeat

Image de fond en CSS

- Permet de définir si l'image doit être répété ou pas
 - repeat, repeat-x, repeat-y, no-repeat

```
body {
background-color: black;
background-image: url(roger_rabbit.jpg);
background-repeat: no-repeat;
color:white;
}
```

71

Image de fond en CSS

Image de fond en CSS

- **background-image** : définit l'image de l'arrière-plan URL de l'image

```
Body {background-image: url (image.gif)}
```

Positionner une image de fond: background-position

Image de fond en CSS

- Permet de définir la position de l'image dans l'élément
 - <pourcentage x> <pourcentage y>
 - <unité x> <unité y> (ex: 30px 50px)
 - top | center | bottom et left | center | right (ex: top left)

```
body {
background-color: black;
background-image: url(roger_rabbit.jpg);
background-repeat: no-repeat;
background-position: 30px 150px;
color:blue;
}
```

72

Image de fond en CSS

Positionner une image de fond: `background-position`

- Si l'on veut qu'elle se place en haut à droite, on rajoutera le code :
`background-position: right top;`
 ou
`background-position:100% 0%;`
- Pour une position au centre de la page, cela donnera :
`background-position:center center;`
 ou
`background-position: 50% 50%;`
- Pour une position en bas à droite :
`background-position:right bottom;`
 ou
`background-position:100% 100%;`

73

Image de fond en CSS

Nouveauté en CSS3 : Multiple background

- La propriété `background-image` n'acceptait, en CSS 2.1, qu'une seule image.
- Les CSS3 permettent des background multiples
- Exemple :
Une image aux quatre coins d'une page

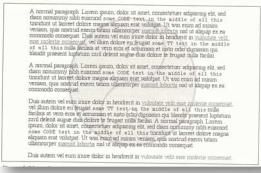
```
Body {
    background: url(images/houx.gif) no-repeat left top,
                url(images/houx.gif) no-repeat right top,
                url(images/houx.gif) no-repeat left bottom,
                url(images/houx.gif) no-repeat right bottom;
}
```

74

Image de fond en CSS

Fond de page fixe : `background-attachment`

- Pour que le fond de page devienne fixe, on rajoutera dans le code :
`background-attachment: fixed;`



→ L'image demeure au centre même lorsque l'on déroule le texte

- Valeur par défaut
`background-attachment: scroll`

74

Image de fond en CSS

Changement d'image de fond au survol du lien

```
#menu a:hover
{
    background-image:url(images/gris_anim.gif);
    background-repeat:no-repeat;
}
```

/*Pour pouvoir combiner une image de fond (ici gris_anim.gif) et un caractère (ici un guillemet), on peut utiliser l'attribut content et le pseudo élément before*/

```
#menu a:hover:before
{
    content:" » ";
}
```



75

Image de fond en CSS

Background

- On peut aussi cumuler ces différentes informations dans un unique background.
- Exemple :

```
body
{
    background: white url(image_de_fond.png) no-repeat
                right bottom;
```

75

Positionnement et flux

Positionnement et flux

- Flux : ordre d'affichage des éléments
- 4 méthodes de positionnement
 - normal
 - float
 - relatif
 - absolu
- Quels sont les différences ?

76

Positionnement et flux

Le flux normal

```
<BODY>
<div class="conteneur">
<div class="normalA">
  Bloc A
</div>
<div class="normalB">
  Bloc B
</div>
</div>
</BODY>
```

```
.normalA {
width:150px;
height:150px;
background-color:red;
border:1px solid black;
}

.normalB {
width:250px ;
height:100px ;
background-color:green;
border:1px solid black;
}
```

Bloc conteneur

Bloc A

Bloc B

Noter bien

Positionnement et flux

➤ CSS:

```
.bloc_relatif
{position:relative;left:30px;}
```

➤ HTML:

Sous le pont Mirabeau

```
<div class="bloc_relatif"> div class="bloc_relatif" </div>
La joie venait toujours après la peine
</div>
```

Sous le pont Mirabeau

```
<span class="bloc_relatif"> span class="bloc_relatif" </span>
La joie venait toujours après la peine
</div>
```

82

Positionnement et flux

Le flux normal en bloc

```
<BODY>
<div class="conteneur">
<div class="normalA">
  Bloc A
</div>
<div class="normalB">
  Bloc B
</div>
</div>
</BODY>
```

```
.normalA {
width:150px;
height:150px;
background-color:red;
border:1px solid black;
}

.normalB {
width:250px ;
height:100px ;
background-color:green;
border:1px solid black;
}
```

Bloc conteneur

Bloc A

Bloc B

Flux normal en bloc : Succession verticale

Noter bien la différence : inline et bloc

Positionnement et flux

Sous le pont Mirabeau code la Seine Et nos amours Faut-il qu'il m'en souvienne **div class="bloc_relatif"**
La joie venait toujours après la peine

Sous le pont Mirabeau code la Seine Et nos amours Faut-il qu'il m'en souvienne **span class="bloc_relatif"** le venait toujours après la peine

Succession horizontale

Succession verticale

83

Positionnement et flux

Le flux normal en ligne

```
<BODY>
<div class="conteneur">
<span class="normalC">
  Bloc C
</span>
<span class="normalD">
  Bloc D
</span>
</div>
</BODY>
```

```
.normalC {
width:150px;
height:150px;
background-color:red;
border:1px solid black;
}

.normalD {
width:250px ;
height:100px ;
background-color:green;
border:1px solid black;
}
```

Bloc conteneur

Bloc C

Bloc D

Flux normal en ligne : Succession horizontale

Bloc D

Noter bien la différence : inline et bloc

Positionnement et flux

Bienvenue à mon site.

Blog, cv, info...

À propos de l'auteur
Je suis Asas Tarek, je suis né le 23 octobre 2011.
Je suis un grand voyageur.
Des articles, des cours et des tutoriels à télécharger, ...

Accueil
Blog
CV
Glossaire
Download
Liens utiles
Contact

Copyright Asas Tarek - 2012. Tous droits réservés - [Me contacter](#).

Positionnement et flux

Noter bien la différence : inline et bloc

```
<html>
<head>
<title>Site Web</title>
<style>
header{
border : 5px inset red;
padding : 5px 5px;
}
nav{
float: left;
width: 150px;
border: 3px solid black;
}
section{
margin-left: 170px;
border: 3px solid blue;
padding : 5px 5px 5px 5px;
}
footer{
margin-left: 170px;
border: 3px outset green;
text-align: center;
}
</style>
</head>
```

Positionnement et flux

Le positionnement absolu

- Le bloc doit être positionné tout en bas à droite (0 pixel par rapport à la droite de la page, 0 par rapport au bas de la page).

```
nav{
position: absolute;
right: 0px;
bottom: 0px; }
```

88

Positionnement et flux

Exemple

Point d'origine (0,0)

top

left

right

bottom

Positionnement absolu de l'élément sur la page

Positionnement et flux

Positionnement fixe

- **Le positionnement fixe:** Le principe est exactement le même que pour le positionnement absolu sauf que, cette fois, le bloc reste fixe à sa position, même si on descend plus bas dans la page.

```
element{
position: fixed;
right: 0px;
bottom: 0px; }
```

89

Positionnement et flux

Noter bien la différence : inline et bloc

- ❖ **Le positionnement absolu** permet de placer un élément (réellement) n'importe où sur la page.
- ❖ Pour effectuer un positionnement absolu, on doit écrire :
`Element{ position: absolute;}`
- ❖ On a dit qu'on voulait un positionnement absolu, mais encore faut-il dire où l'on veut que le bloc soit positionné sur la page.
- ❖ Pour ce faire, on va utiliser quatre propriétés CSS :
 - ✓ `left` : position par rapport à la gauche de la page ;
 - ✓ `right` : position par rapport à la droite de la page ;
 - ✓ `top` : position par rapport au haut de la page ;
 - ✓ `bottom` : position par rapport au bas de la page.
- ❖ On peut leur donner une valeur en pixels, comme 14px, ou bien une valeur en pourcentage, comme 50%.

Positionnement et flux

Le positionnement absolu

- Le positionnement relatif permet d'effectuer des «ajustements» : l'élément est décalé par rapport à sa position initiale.
- Par exemple un texte important, situé entre deux balises ``.
- Pour commencer, je le mets sur fond rouge pour qu'on puisse mieux le repérer :

```
>strong{ background-color: red; /* Fond rouge */
>color: yellow; /* Texte de couleur jaune */ }
```

Point d'origine(0,0)

C'est moi, **bonjour les amis!** Je suis né un 23 novembre 2005.

C'est moi, **bonjour les amis!** Je suis né un 23 novembre 2005.



Positionnement et flux

Le positionnement relatif

➤ Si vous faites un **position: relative;** le texte sur fond rouge va se déplacer par rapport à la position où il se trouve.

➤ Je veux que mon texte se décale de 55 pixels vers la droite et de 10 pixels vers le bas.

➤ Je vais donc demander à ce qu'il soit décalé de 55 pixels par rapport au « bord gauche » et de 10 pixels par rapport au « bord haut ».

```
Strong{background-color: red;
color: yellow;
position: relative;
left: 55px;
top: 10px;}
```

C'est moi, → 55 px
↓ 10 px Le suis né un 23 novembre 2005.
bonjour les amis!



Positionnement et flux

Le flux flottant

```
<BODY>
<div class="conteneur">
<div class="flotteA">
Boîte A
</div>
<div class="flotteB">
Boîte B
</div>
</div>
</BODY>
```

```
.conteneur {
width:800px;
border:1px solid black;
}
```

```
.flotteA {
float:left;
width:650px;
background-color:yellow;
border:1px solid black;
}
```

```
.flotteB {
float:left;
width:100px;
background-color:blue;
border:1px solid black;
}
```

Bloc conteneur - 800px de large

Boîte A - 650px



Positionnement et flux

Le flux flottant

```
<BODY>
<div class="conteneur">
<div class="flotteA">
Boîte A
</div>
<p>
Texte...blabla ...
</p>
</div>
</BODY>
```

```
.flotteA {
float:left;
width:500px;
background-color:yellow;
border:1px solid black;
}
```

Bloc conteneur
Boîte A - float:left



Positionnement et flux

Le flux flottant

```
<BODY>
<div class="conteneur">
<div class="flotteA">
Boîte A
</div>
<div class="flotteB">
Boîte B
</div>
</div>
</BODY>
```

```
.conteneur {
width:800px;
border:1px solid black;
}
```

```
.flotteA {
float:left;
width:650px;
background-color:yellow;
border:1px solid black;
}
```

```
.flotteB {
float:left;
width:100px;
background-color:blue;
border:1px solid black;
}
```

Bloc conteneur - 800px de large

Boîte B - 100px



Positionnement et flux

Le flux flottant

```
<BODY>
<div class="conteneur">
<div class="flotteA">
Boîte A
</div>
<p>
Texte...blabla ...
</p>
</div>
</BODY>
```

```
.flotteA {
float:left;
width:500px;
background-color:yellow;
border:1px solid black;
}
```

Bloc conteneur



Positionnement et flux

Le flux flottant

```
<BODY>
<div class="conteneur">
<div class="flotteA">
Boîte A
</div>
<div class="flotteB">
Boîte B
</div>
</div>
</BODY>
```

```
.conteneur {
width:800px;
border:1px solid black;
}
```

```
.flotteA {
float:left;
width:650px;
background-color:yellow;
border:1px solid black;
}
```

```
.flotteB {
float:left;
width:600px;
background-color:blue;
border:1px solid black;
}
```

Bloc conteneur - 800px de large

Boîte B - 600px

Positionnement et flux

Le flux flottant

`<BODY>`

```

<div class="conteneur">
  <div class="flotteA">
    Boite A
  </div>
  <div class="flotteB">
    Boite B
  </div>
</BODY>

```

`.conteneur { width:800px; border:1px solid black; }`

`.flotteA { float:left; width:650px; background-color:yellow; border:1px solid black; }`

`.flotteB { float:left; width:600px; background-color:blue; border:1px solid black; }`

Positionnement et flux

Flux absolu

`<BODY>`

```

<div class="conteneur" style="width:800px; border:1px solid black; position: relative; height: 100px; margin-bottom: 10px; background-color: #f0f0f0; overflow: hidden;">
  <div class="bloc_absolu" style="position: absolute; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center;">
    Bloc Aaaaaaa
  </div>
  dddddd....dddd
</div>
</BODY>

```

`<div class="bloc_absolu" style="position: absolute; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center;">Bloc Aaaaaaa`

Positionnement et flux

Le flux relatif

`<BODY>`

```

<div class="conteneur">
  <div class="bloc_relatif" style="position: relative; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center;">
    Bloc A
  </div>
  <div class="bloc_relatif" style="position: relative; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center;">
    Bloc B
  </div>
</div>

```

`.conteneur { width:800px; border:1px solid black; }`

`.bloc_relatif { position: relative; width:300px; margin-top:20px; margin-left:30px; border:1px solid black; }`

Positionnement

Flexbox

Principe : conteneur + éléments

Sans Flexbox

```

<div id="conteneur">
  <div class="element1">Element 1</div>
  <div class="element2">Element 2</div>
  <div class="element3">Element 3</div>
</div>

```

Avec Flexbox

```

<div>
  <div>Element 1</div>
  <div>Element 2</div>
  <div>Element 3</div>
</div>

```

`<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <link href="css/flextest.css" rel="stylesheet">
 <title>Illustration de la mise en page avec flex</title>
</head>
<body>`

`<div id="conteneur">
 <div class="element1">Element 1</div>
 <div class="element2">Element 2</div>
 <div class="element3">Element 3</div>
</div>`

`<div>
 <div>Element 1</div>
 <div>Element 2</div>
 <div>Element 3</div>
</div>`

101

Positionnement et flux

Flux absolu

`<BODY>`

```

<div class="conteneur">
  <div class="bloc_absolu" style="position: absolute; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center;">
    Bloc A
  </div>
  <div class="bloc_absolu" style="position: absolute; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center;">
    Bloc B
  </div>
</div>

```

`.conteneur { width:800px; border:1px solid black; }`

`.bloc_absolu { position: absolute; width: 300px; margin-top: 20px; margin-left: 30px; border: 1px solid black; background-color: #fff; z-index: 1; font-size: 0.8em; padding: 5px; color: #000; text-align: center; }`

Positionnement : Direction

Flex offre une flexibilité dans l'ordonnancement des éléments via la propriété `flex-direction` qui peut prendre les valeurs suivantes:

- `row` : organisés sur une ligne. C'est la valeur par défaut
- `column` : organisés sur une colonne
- `row-reverse` : sur une ligne, mais en ordre inversé
- `column-reverse` : sur une colonne, mais en ordre inversé

102

Flexbox

Positionnement : Direction

Flexbox

WRAP

nowrap		Les éléments se resserrent tant qu'ils peuvent.
wrap		Les éléments passent à la ligne.
wrap-reverse		Les éléments passent à la ligne à l'envers.

The page number is 106.

Flexbox

Positionnement : Direction

Flexbox

Alignment

Les blocs sont alignés sur deux axes : **axe principale** (défini par `flex-direction`) et **axe secondaire**. L'**axe secondaire** représente l'axe **opposé** l'axe que vous utilisez initialement.

- Si vos éléments sont organisés **horizontalement**, l'**axe secondaire** est l'**axe vertical**.
- Si vos éléments sont organisés **verticalement**, l'**axe secondaire** est l'**axe horizontal**.

The page number is 107.

Flexbox

WRAP

Le traitement par défaut des blocs consiste à toujours essayer de ce placer sur la même ligne.

La propriété `flex-wrap` permet de gérer l'alignement des blocs au sein même de ma ligne.

- `nowrap` : pas de retour à la ligne c'est la valeur par défaut
- `wrap` : les éléments vont à la ligne lorsqu'il n'y a plus la place
- `wrap-reverse` : les éléments vont à la ligne lorsqu'il n'y a plus la place en sens inverse

The page number is 105.

Flexbox

Alignement sur l'axe principal

L'alignement sur l'axe principal se fait avec la propriété `justify-content`

- `flex-start` : alignés au début qui est la valeur par défaut
- `flex-end` : alignés à la fin
- `center` : alignés au centre
- `space-between` : les éléments sont étirés sur tout l'axe sans laisser d'espace entre eux
- `space-around` : idem, les éléments sont étirés sur tout l'axe, mais ils laissent aussi de l'espace sur les extrémités

The page number is 108.

Flexbox

Alignement sur l'axe principal

flex-start
flex-end
center
space-between
space-around

109

Flexbox

La propriété `flex` permet de pondérer la taille des éléments.

Si un élément possède la propriété `flex` et la valeur `1`, il prendra tout l'espace restant libre

Si plusieurs éléments possèdent cette propriété alors la valeur représentera le poids de la taille de cet élément

flex : 1;
flex : 2;
flex : 1;

112

Flexbox

Alignment sur l'axe secondaire

L'alignement sur l'axe secondaire se fait avec la propriété `align-items`

- `stretch` : les éléments sont étirés sur tout l'axe ce qui représente la valeur par défaut
- `flex-start` : alignés au début
- `flex-end` : alignés à la fin
- `center` : alignés au centre
- `baseline` : alignés sur la ligne de base ce qui est semblable à `flex-start`

110

INSAT 2015/2016

HTML

AYMEN SELLAOUTI
aymen.sellaouti@gmail.com

CSS3

113

Flexbox

Alignment sur l'axe secondaire d'un seul élément

L'alignement sur l'axe secondaire peut être fait sur un seul élément en utilisant `align-self`

```

#conteneur1 {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
}

.element1 {
    width: 25px;
    height: 50px;
    border: solid 2px blue;
}

.element2 {
    width: 25px;
    height: 50px;
    border: solid 2px darkred;
    align-self: flex-end;
}

.element3 {
    width: 25px;
    height: 50px;
    border: solid 2px greenyellow;
}

```

111

Symfony2 Introduction

AYMEN SELLAYOUTI

Pourquoi utiliser un Framework

Productivité : ensemble de composants déjà prêt à l'emploi

Propreté du code => maintenabilité et évolutivité

Basée sur une architecture => facilite le travail en équipe

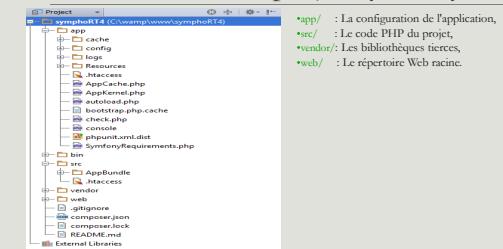
Communauté et documentation

C'est quoi symfony ?

« Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony. » [site Official Symfony]

- ✓ Ensemble de composants PHP
- ✓ Framework pour les applications web
- ✓ Basé sur des composants
- ✓ Structuration du code
- ✓ Maintenabilité
- ✓ Une philosophie
- ✓ Les bonnes pratiques
- ✓ standardisation
- ✓ Une très grande communauté

Architecture d'un projet Symfony2



Framework

FrameWork : Cadre De Travail (Boîte à outils)

Ensemble de composants servant à créer :

- Fondation
- Architecture

Architecture d'un projet Symfony2 app/

Configuration (app/config)

Cache(app/cache)

Logs(app/log)

Ressources(app/ressources)

Architecture d'un projet Symfony2 src/

Code source php du projet

Organisé en Bundle(module, brique logiciel)

Architecture d'un projet Symfony2 vendor/

Bibliothèques externes de l'application

Exemple :

- symfony2
- Doctrine
- Twig
- SwiftMailer

Le contrôleur frontal (1)

Il joue le rôle de dispatcheur :

- Intercepte les requêtes
- Appelle le noyau de symfony (AppKernel.php)
- Le noyau prépare la réponse à rendre

Architecture d'un projet Symfony2 web/

Contient les fichiers statiques et publiés :

- Images
 - Feuilles de styles
 - Javascript,...
- Contient les deux contrôleurs frontaux
- app.php (pour le mode production)
 - app_dev.php (pour le mode développement)

Le contrôleur frontal (2)

Le contrôleur principal s'occupe de gérer les requêtes, mais cela signifie plus que simplement déterminer une action à exécuter.

En fait, il exécute le code qui est commun à chaque action, soit les étapes suivantes :

- Definir les constantes
- Déterminer les chemins des bibliothèques Symfony.
- Charger et initialiser les classes du cœur du framework.
- Changer la configuration.
- Décoder la requête URL afin d'identifier les actions à effectuer et les paramètres de la requête.
- Si l'action n'existe pas, faire une redirection sur l'action 404 error.
- Activer les filtres (par exemple, si les requêtes nécessitent une authentification).
- Exécuter les filtres une première fois.
- Exécuter l'action et générer la présentation.
- Exécuter les filtres une seconde fois.
- Renvoyer la réponse.

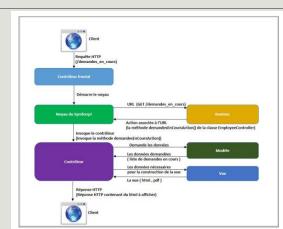
Architecture MVC

Symfony respecte l'architecture MVC2

M: Model

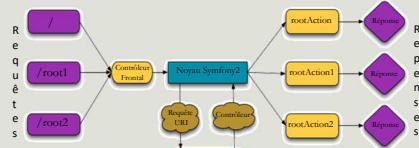
V: Vue

C: Contrôleur



Avec l'utilisation d'un contrôleur frontal

Traitement d'une requête au sein de Symfony2



Structure d'un Bundle

Controller : C'est le dossier qui contient les contrôleurs du Bundle.

Entity : Dossier Qui contient les entités du Bundle

Form : Dossier contenant la définition des formulaires du Bundle

Test : Dossier pour les test unitaires du Bundle

Ressources : Dossier contenant les fichiers nécessaires à votre bundle à savoir :

➤ les fichiers de configurations dans le sous dossier **config**

➤ Les vues du Bundle dans le sous dossier **views**

➤ Les fichiers Css et Js dans le sous dossier **public**

...

La décomposition en Bundle (1)

Un bundle peut être défini comme étant un :

- Une brique logicielle,
- un module fonctionnel
- un morceau d'application.

Exemple : Site internet sur les Jeux Vidéos

Nous pouvons avoir les bundles suivants :

- Forum
- Newsletter
- Actualité...

Les Bundles les plus connus de la communauté

FOSUserBundle : Gestion des utilisateurs

SonataAdminBundle : Espace d'administration

KnpMenuBundle : Gestion des menus

KnpPaginatorBundle : Pagination

<http://knplbundles.com/> ; <http://symfony.com/blog/the-30-most-useful-symfony-bundles-and-making-them-even-better>
<https://packagist.org>

La décomposition en Bundle (2)

Un Bundle peut être

- Indépendant et réutilisable
- Dépendant d'autres bundles

Un Bundle permet de découper l'application en fonctionnalités. Il a une architecture bien définie.

Un Bundle peut être utilisé dans plusieurs applications.

Installation de Symfony2 (1)

Il existe plusieurs méthodes afin d'installer symfony2
(<http://symfony.com/fr/doc/current/book/installation.html>)

Pré requis :

Serveur Web contenant PHP (Apache).

Pour vérifier la bonne mise en place des pré requis utiliser la commande suivante :

`php app/check.php`

<http://symfony.com/fr/doc/current/reference/requirements.html> ;

Installation de Symfony2 (2)

Méthode 1

Télécharger la version souhaitée (<http://symfony.com/download>)

Méthode 2

Télécharger et installer composer (<https://getcomposer.org/download/>)

Ouvrir la ligne de commande et placer vous sur le dossier d'installation

Tapez la commande suivante : **composer create-project symfony/framework-standard-edition path/ "2.8.*"** (Ici nous avons mentionné la dernière version lors de la réalisation de ce cours à savoir la version 2.7.*)

Méthode 3

Installer le symfony installer avec la commande suivante :

c:\> php -r "readfile('http://symfony.com/installer');" > symfony

Copier le fichier dans le dossier père de votre dossier d'installation

Taper la commande suivante : **php symfony new nomProjet**

Génération du premier Bundle

2 Méthodes peuvent générer un Bundle

> Méthode manuelle (non recommandée)

> Méthode automatique avec la console

commande : **php app/console generate:bundle**

Propriétés de la commande :

namespace : Espace de nom il peut avoir n'importe quel nom qui se termine par Bundle les bonnes pratiques nous apprennent qu'il est préférable de le décomposer en trois parties

> Namespace racine qui représente généralement l'organisme (e.g. nom d'un particulier, d'une entreprise)

> Nom du Bundle

> Le suffixe Bundle

Exemple : RT4/TestBundle

Configuration de votre projet

Afin de configurer votre projet il y a deux solutions :

Solution 1 : app/config/parameters.yml

Solution 2 : <http://127.0.0.1/nomDossier/web/config.php>



Génération du premier Bundle (2)

Propriétés de la commande :

Nom du bundle : Le nom est automatiquement suggéré en utilisant le namespace sans le /

Destination : le répertoire de destination par défaut c'est un dossier portant le nom du bundle sous le répertoire source

Format de configuration : Représente la forme avec laquelle Symfony va configurer le Bundle. Les plus utilisées sont YML et Annotation. Nous utilisons comme départ YML.

Ma première page

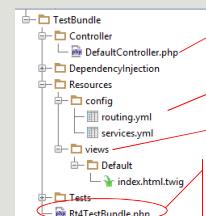
Tester ce lien en mettant le nom que vous avez choisi pour votre application : http://127.0.0.1/NomDossier/web/app_dev.php/app/example



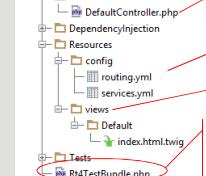
Passez maintenant en mode prod

<http://127.0.0.1/NomDossier/web/app.php/app/example> Qu'est ce qui a changé ?

Génération du premier Bundle (3)



Les contrôleurs du bundle



Les routes du bundle

Les vues du bundle

Le seul fichier obligatoire pour un bundle il est donc nécessaire de le créer lors d'une création manuelle d'un bundle. Il va juste hériter de la classe bundle

Enregistrement du Bundle

Afin de charger les bundles d'un projet, Symfony enregistre ces derniers dans son noyau.

Le fichier AppKernel se trouve dans le dossier app

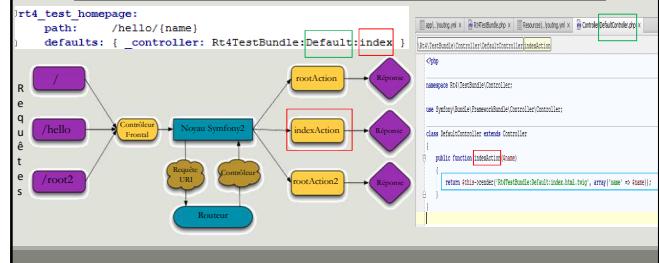
L'enregistrement du Bundle se fait automatiquement avec la console.

Pour la création manuelle du Bundle il faut l'ajouter manuellement dans le tableau de bundle de la méthode registerBundles() du appKernel

Ajout du bundle test

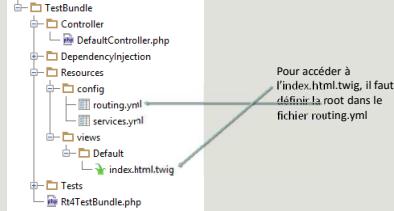
```
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle(),
            new Sensio\Bundle\DistributionBundle\SensioDistributionBundle(),
            new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new AppBundle\AppBundle(),
            new Rt4TestBundle\Rt4TestBundle(),
            new Stash\StashBundle\StashBundle(),
        );
    }
}
```

Enregistrement des roots du Bundle (3)



Enregistrement des roots du Bundle (1)

Afin d'accéder au différentes vues d'un bundles nous devons enregistrer les roots qui correspondent.

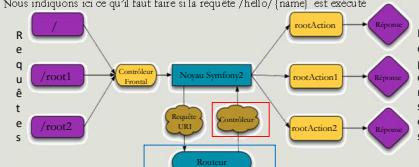


aymen.sellaouti@gmail.com

Enregistrement des roots du Bundle (2)

```
rt4_test_homepage:
    path: /hello/{name}
    defaults: { _controller: Rt4TestBundle:Default:index }
```

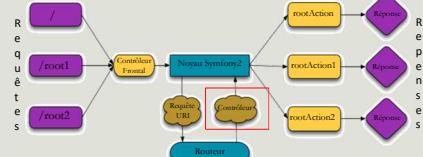
Nous indiquons ici ce qu'il faut faire si la requête `/hello/{name}` est exécuté



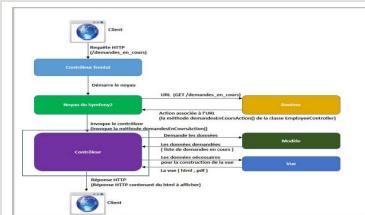
Symfony2 Les contrôleurs

AYMEN SELLAYOUTI

Introduction (3)



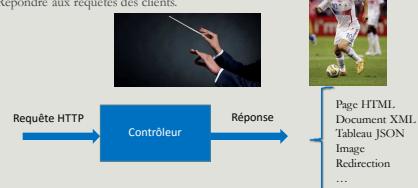
Introduction (1)



Introduction (4)

Fonction PHP (action)

Rôle : Répondre aux requêtes des clients.



Introduction (2)

Système permettant de

- gérer les liens internes du projet
- avoir des URLs plus explicites
- associer une URL à une action

Exemple d'un contrôleur

```
<?php
namespace RT4\TestBundle\Controller; // espace de nom
use Symfony\Bundle\FrameworkBundle\Controller\Controller; // on importe la classe Controller
use Symfony\Component\HttpFoundation\Response;
class HelloController extends Controller // Si on veut que notre contrôleur hérite de la classe afin de pouvoir utiliser
// ces méthodes helper (http://api.symfony.com/2.7/Symfony/Bundle/FrameworkBundle/Controller.html)
{ // Par convention chaque contrôleur (action) se termine par le mot clé Action
    public function indexAction() // le nom index est l'identifiant de l'action c'est avec ce nom que nous
        // l'appelons dans la root
    { // On crée un objet Response puis sur la retourne c'est le rôle du contrôleur
        $resp = new Response('<html><body>Bonjour RT4!</body></html>');
        return $resp;
    }
}
```

Fonctions de base de la classe Contoller

Méthode	fonctionnalité	Valeur de retour
generateUrl(string \$route, array() \$parameters)	Génère une URL à partir de la route	String
forward(String Controller, array () \$parameters)	Forward la requête vers un autre contrôleur	Response
Redirect(string \$url int \$statut)	Redirige vers une url	RedirectResponse
Render(string \$view array \$parameters)	Affiche une vue	Response
getRequest()	Raccourci pour récupérer le service request	Request
Get(string \$id)	Retourne un service à travers son id	object
createNotFoundException(String \$messag)	Retourne une NotFoundException	NotFoundException

<http://api.symfony.com/2.7/Symfony/Bundle/FrameworkBundle/Controller/Controller.html>

Récupérer les paramètres de la requête (1)

Afin de récupérer l'objet Request dans le contrôleur il suffit de le déclarer dans l'entête du contrôleur en question.

Exemple

```
public function indexAction(Request $req)
{
    ...
}
```

Lien entre la root et le contrôleur Création de la root

Le Contrôleur exemple se trouvant dans la classe controller Exemple et s'appelant action la root menant vers ce contrôleur doit suivre la syntaxe suivante :

```
_controller:NomBundle:NomClasseController:NomAction

rt4_bjr:
path: /bonjour
defaults: { _controller: Rt4TestBundle:Exemple:index}
```

Récupérer les paramètres de la requête (2)

L'objet Request permet de récupérer l'ensemble des attributs passés dans la requête

Type de paramètres	Méthode Symfony2	Méthode traditionnelle	Exemple
Variables d'URL	\$request->query	\$_GET	\$request->query->get('tag')
Variables de formulaire	\$request->request	\$_POST	\$request->request->get('tag')
Variables de cookie	\$request->cookies	\$_COOKIE	\$request->cookies->get('tag')

Lien entre la root et le contrôleur Passage de paramétré : root vers contrôleur

Afin de récupérer les paramètres de la root dans le contrôleur nous utilisons les noms des paramètres.

Exemple

```
rt4_bjr:
path: /bonjour/{section}
defaults: { _controller: Rt4TestBundle:Exemple:index}
public function indexAction($section)
{
    $resp = new Response('<html><body>Bonjour'. $section.'!</body></html>');
    return $resp;
}
```

Récupérer les paramètres de la requête (3)

Exemple : pour l'url suivante :
http://127.0.0.1/symfonyRT4/web/app_dev.php/test/bonjour/RT4?groupe=1
Pour récupérer le groupe passé dans l'url (donc du Get) on devra récupérer le request puis utiliser \$request->query->get('tag')

Exemple

```
public function indexAction($section,Request $req)
{
    $groupe = $request->query->get('groupe');
    return new Response(' Bonjour '.$section.' groupe '.$groupe);
}
```

Récupérer les paramètres de la requête (4)

La classe Request offre plusieurs informations concernant la requête HTTP à travers un ensemble de méthodes (<http://api.symfony.com/2.5/Symfony/Component/HttpFoundation/Request.html>)

getMethod() : retourne la méthode de ma requête

getLocale() : retourne la locale de la requête (langue)

isXmlHttpRequest() : retourne vrai si la requête est de type XMLHttpRequest

...

Réponse aux requêtes Redirection

Rôle : Redirection vers une deuxième page

Méthode :

- Sans passer par les helpers de la classe Controller :
 - \$url = \$this->get('router')->generate('notre_route');
 - Return new RedirectResponse(\$url);
- En utilisant les helpers :
 - \$url = \$this->get('router')->generate('notre_route');
 - Return \$this->redirect(\$url);

Réponse aux requêtes Renvoi (1)

Le traitement se fait à travers le service templating

Rôle : créer la réponse et la retourner

Méthode :

- Sans passer par les helpers de la classe Controller :
 - \$this->get('templating')->renderResponse('l'url', 'les paramètres à transférer');
- En utilisant les helpers :
 - \$this->render('l'url', 'les paramètres à transférer');

Réponse aux requêtes Forwarding

Rôle : Forwarder vers un autre contrôleur

Méthode :

- Sans passer par les helpers de la classe Controller en utilisant le httpKernel :
 - \$httpKernel = \$this->container->get('http_kernel');Return new RedirectResponse(\$url);
 - \$response = \$httpKernel->forward('NomBundle:NomController:NomAction', array('label1' => \$param1, 'label2' => \$label2,));
 - return(\$response);
- En utilisant les helpers :
 - \$response = \$this->forward('NomBundle:NomController:NomAction', array('label1' => \$param1, 'label2' => \$label2,));
 - return(\$response);

Réponse aux requêtes Renvoi (2)

Exemple :

```
public function indexAction($section, Request $req)  
{ $groupe = $request->query->get('groupe');  
return $this->get('templating')->renderResponse('Ri4AsBundle:Default:index.html.twig', array('section'=>$section,  
'groupe'=>$groupe));  
}  
  
public function indexAction($section, Request $req)  
{ $groupe = $request->query->get('groupe');  
return $this->render('Ri4AsBundle:Default:index.html.twig', array('section'=>$section,  
'groupe'=>$groupe));  
}
```

aymen.sellaouti@gmail.com

Symfony2 Routing

AYMEN SELLAUTI

Format de gestion du routing

➤ YAML

➤ XML

➤ PHP

➤ Annotations

Introduction

Système permettant de

- gérer les liens internes du projet
- avoir des URLs plus explicites
- associer une URL à une action

Routing : fichier principal

Emplacement : app/config/routing.yml

Squelette d'une route :

```
Lynda_back:
    resource: "@LyndaBackBundle/Resources/config/routing.yml"
    prefix: /
```

Référence au fichier routing de LyndaBackBundle

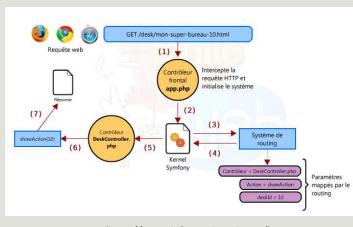
```
Lynda_front:
    resource: "@LyndaFrontBundle/Resources/config/routing.yml"
    prefix: /
```

```
rtt_main:
    resource: "@RTTMainBundle/Resources/config/routing.yml"
    prefix: /as
```

```
rtt_test:
    resource: "@RTTTestBundle/Resources/config/routing.yml"
    prefix: /test
```

```
app:
    resource: "AppBundle/Controller/"
    type: annotation
```

Introduction



Routing (<http://www.lafemeduweb.net/>)

Externalisation des fichiers de routing (1)

Externalisation des fichiers de routing (1)

```
moi.your.yml x
wellalot.routing:
    resource: "Wellalot1RTTRouting2Bundle/Resources/config/routing.yml"
    prefix: /
```

```
wellalot_rtt_routing:
    resource: "Wellalot1RTTRouting2Bundle/Resources/config/routing.yml"
    prefix: /
```

```
app:
    resource: "AppBundle/Controller/"
    type: annotation
```

```
wellalot_rtt_routing1_homepage:
    path: "/hello/{name}"
    defaults: { _controller: Wellalot1RTTRouting2Bundle:Default:index }
```

```
wellalot_rtt_routing2_homepage:
    path: "/hello/{name}"
    defaults: { _controller: Wellalot1RTTRouting2Bundle:Default:index }
```

Externalisation des fichiers de routing (2)

```

SensioRT4
  - RoutingBundle
    - Controller
    - DependencyInjection
    - Resources
      - config
        - routing.yml
      - doc
      - translations
      - views
    - Tests
    - SellosoutiRT4RoutingBundle.php
    - Resources
      - config
        - routing.yml
      - doc
      - translations
      - views

```

```

resources:
  - {resource: '@SellosoutiRT4RoutingBundle/Resources/config/routing.yml'}
  prefix:   /front
  - {resource: '@SellosoutiRT4RoutingBundle/Resources/config/routing.xml'}
  prefix:   /front

sellosouti_rt4_routing.xml:
<?xml version="1.0" encoding="UTF-8" ?>
<routes xmlns="http://symfony.com/schema/routing"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://symfony.com/schema/routing http://symfony.com/schema/routing/routing.xsd">
  <route id="blog" path="/blog/{page}">
    <default key="_controller">AcmeBlogBundle:Blog:index</default>
    <requirement key="page">\d+</requirement>
  </route>
</routes>
```

```

resources:
  - {resource: '@SellosoutiRT4RoutingBundle/Resources/config/routing.yml'}
  type: annotation
  - {resource: '@AppBundle/Controller/'}

sellosouti_rt4_routing.yml:
blog:
  path: /hello/{name}
  defaults: { _controller: SellosoutiRT4RoutingBundle:Default:index }
```

Squelette d'une root (3)

```

class DefaultController extends Controller
{
    /**
     * @Route("/app/example", name="homepage") Exemple de routing utilisant les annotations
     */
    public function indexAction()
    {
        return $this->render('default/index.html.twig');
    }
}
```

<http://symfony.com/fr/doc/current/book/routing.html>

Squelette d'une root (1)

```

YAML XML PHP
1 blog
2   path: /blog/{page}
3   defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }
4   requirements:
5     page: \d+

```

```

<?xml version="1.0" encoding="UTF-8" ?>
<routes xmlns="http://symfony.com/schema/routing"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://symfony.com/schema/routing http://symfony.com/schema/routing/routing.xsd">
  <route id="blog" path="/blog/{page}">
    <default key="_controller">AcmeBlogBundle:Blog:index</default>
    <requirement key="page">\d+</requirement>
  </route>
</routes>
```

<http://symfony.com/fr/doc/current/book/routing.html>

Le choix du format des fichiers

Sensio recommande sans concession l'utilisation du format Yaml au sein des applications. Les bundles développés sont partagés entre deux formats : le XML et le Yaml.

Sensio a décidé de recommander Yaml car celui-ci est plus « *user-friendly* ».

L'utilisation d'un autre format ne changerait rien au fonctionnement de votre application.

Squelette d'une root (2)

```

YAML XML PHP
1 us: Symfony\Component\Routing\RouteCollection;
2 us: Symfony\Component\Routing\Route;
3
4 $collection = new RouteCollection();
5 $collection->add('blog', new Route('/blog/{page}', array(
6   '_controller' => 'AcmeBlogBundle:Blog:index',
7   'page' => 3,
8 ), array(
9   'page' => '\d+',
10 )));
11
12 return $collection;

```

<http://symfony.com/fr/doc/current/book/routing.html>

Paramétrage de la root (1)

```

front_homepage:
  path: /hello/{name}
  defaults: { _controller: LyndaFrontBundle:Default:index }

En intégrant le paramètre {name}, les urls valables sont les roots de type /hello/*
Exemple :
/hello/
/hello/1
/hello/1475
/hello/abcde

/hello est-elle une URL valide ?
```

Paramétrage de la root (2)

Nous pouvons ajouter autant de paramètre dans la route

Les séparateurs entre les paramètres sont '/' et ':'

Exemple

```
front_article:  
path: /article/{year}/{langue}/{slug}.{format}  
defaults: {_controller: LyndaFrontBundle:Default:index }
```

Ici l'url doit contenir l'année de l'article {year}, la langue de l'article {langue} les mots clefs {slug} ainsi que le format {format}

Astuce

Pour la langue utiliser la variable fourni par symfony2 _locale permettant ainsi de modifier en même temps la locale de la page

Pour le format utiliser le _format qui modifie le content type envoyé au navigateur

Paramétrage de la root (5)

Pour contrôler les paramètres de la route on utilise les requirements

```
Syntaxe  
front_article:  
path: /article/{year}/{_locale}/{slug}.{format}  
defaults: {_controller: LyndaFrontBundle:Default:index, variable:valeurParDefaut }  
requirements:  
    nomParamètre: condition
```

Exemple

```
front_article:  
path: /article/{year}/{langue}/{slug}.{format}  
defaults: {_controller: LyndaFrontBundle:Default:index, _format: html}  
requirements:  
    year:\d{4} // l'année sera obligatoirement un chiffre composé de 4 entiers  
    langue: fr|en // ici la langue ne pourra être qu'anglais ou français
```

Paramétrage de la root (3)

Exercice :

Est-ce que ces URL sont valides ou non selon la root défini et pourquoi ?

Que contient chaque variable ?

```
front_article:  
path: /article/{year}/{langue}/{slug}.{format}  
defaults: {_controller: LyndaFrontBundle:Default:index }
```

1- http://127.0.0.1/symfoRT4/web/app_dev.php/article/2005/fr/page.twig-symfony-routing.twig.html

2- http://127.0.0.1/symfoRT4/web/app_dev.php/article/a/2005/fr/page.twig-symfony-routing.twig.html

3- http://127.0.0.1/symfoRT4/web/app_dev.php/article/2005/fr/twig-symfony/-routing.twig.html

Paramétrage de la root (5)

\d équivalence à \d{1}

\d+ ensemble d'entiers

Gestion des méthodes http :

```
_method: POST // ceci signifie que la root ne s'exécuteira qu'à travers la méthode POST  
options les plus utilisées :  
➤ GET  
➤ POST
```

Paramétrage de la root (4)

Afin d'avoir des valeurs par défauts nous utilisons la syntaxe suivante :

```
Syntaxe  
front_article:  
path: /article/{year}/{_locale}/{slug}.{_format}  
defaults: {_controller: LyndaFrontBundle:Default:index, variable:valeurParDefaut }
```

Exemple

```
front_article:  
path: /article/{year}/{langue}/{slug}.{format}  
defaults: {_controller: LyndaFrontBundle:Default:index, _format: html}
```

Important : Seul un paramètre optionnel terminant la root pourra être absent de l'URL.

Ordre de traitement des roots (1)

Le traitement des roots se fait de la première root vers la dernière.

Attention à l'ordre d'écriture de vos roots.

Exemple

```
front:  
    path: /front/{page}  
    defaults: {_controller: FrontBundle:Default:index}  
front_pages:  
    path: /front/{Keys}  
    defaults: {_controller: FrontBundle:Default:show }  
Comment accéder au path front_pages ? Quel est le problème avec ces 2 roots
```

Ordre de traitement des roots (2)

```
front:  
    path: /front/{page}  
    defaults: { _controller: FrontBundle:Defaultindex}  
  
front_pages:  
    path: /front/{Keys}  
    defaults: { _controller: FrontBundle:Defaultshow }  
  
Les deux roots sont de la forme /front/* donc n'importe quel root de cette forme sera  
automatiquement transféré au Default controller pour executer l'index action.
```

Proposer une solution

Ordre de traitement des roots (5)

Les Routes précédentes Gagnent toujours

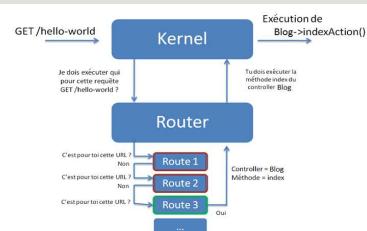
L'ordre des routes est très important.

En utilisant un ordre clair et intelligent, vous pouvez accomplir tout ce que vous voulez.
<http://symfony.com/fr/doc/current/book/routing.html>

Ordre de traitement des roots (3)

```
front:  
    path: /front/{page}  
    defaults: { _controller: FrontBundle:Defaultindex}  
  
requirements:  
    page: \d+  
  
front_pages:  
    path: /front/{Keys}  
    defaults: { _controller: FrontBundle:Defaultshow }  
  
Tester le fonctionnement des routes suivantes : /front/1234  
/front/test-ordre-de-rooting
```

Ordre de traitement des roots (6)



Ordre de traitement des roots (4)

Que se passe t-il si on inverse l'ordre des deux roots ? Est-ce que la solution persiste?

```
front_pages:  
    path: /front/{page}  
    defaults: { _controller: FrontBundle:Defaultshow }  
  
front:  
    path: /front/{page}  
    defaults: { _controller: FrontBundle:Defaultindex}  
  
requirements:  
    page: \d+  
  
Tester le fonctionnement des routes suivantes : /front/1234  
/front/test-ordre-de-rooting
```

Routing avec les annotations

Les annotations :

- Méta données de configuration ajoutés directement au code source
 - Alternative aux fichiers de configuration
 - Peuvent être ajoutées aux classes méthodes attributs ...
- <http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>

Routing avec les annotations

Définir une root

```
1 blog_home
  pattern: /
  defaults: {_controller: SensioBlingBundle:Post:index}

1 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
2
3 class PostController extends Controller
4 {
5     /**
6      * @Route("/")
7      */
8     public function indexAction()
9     {
10        // ...
11    }
12 }
```

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>

YML

Annotation

Routing avec les annotations

Nom de la route (2)

La route définie par l'annotation `@Route` a un nom généré par défaut

Le nom de la route peut être surchargé en utilisant l'attribut `name`

```
1 /**
2  * @Route("/", name="blog_home")
3  */
4 public function indexAction()
5 {
6     // ...
7 }
```

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>

Routing avec les annotations

Valeur par défaut

```
1 /**
2  * @Route("/{id}", requirements={"id" = "\d+"}, defaults={"foo" = "bar"})
3  */
4 public function showAction($id)
5 {
6 }
```

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>

Routing avec les annotations

Préfixe

Une annotation Route sur une classe Contrôleur définit une un préfixe sur toutes les routes des actions de ce contrôleur

```
1 /**
2  * @Route("/blog")
3  */
4 class PostController extends Controller
5 {
6     /**
7      * @Route("/{id}")
8      */
9     public function showAction($id)
10    {
11    }
12 }
```

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>

Routing avec les annotations

Nom de la route (1)

La route définie par l'annotation `@Route` a un nom généré par défaut

NomeRoute= NomBundle_NomContrôleur_NomAction

Quel nom a cette route sachant qu'elle se trouve dans le bundle sensio?

```
1 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
2
3 class PostController extends Controller
4 {
5     /**
6      * @Route("/")
7      */
8     public function indexAction()
9     {
10        // ...
11    }
12 }
```

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>

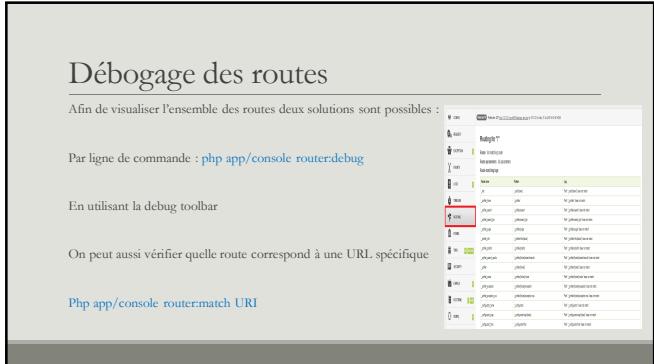
Routing avec les annotations

Méthodes HTTP

Pour spécifier la méthode HTTP gérée par la route il faut utiliser l'annotation `@Method`

```
1 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
2 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
3
4 /**
5  * @Route("/blog")
6  */
7 class PostController extends Controller
8 {
9     /**
10      * @Route("/{id}")
11      * @Method("GET")
12      */
13     public function editAction($id)
14     {
15     }
16 }
```

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/routing.html>



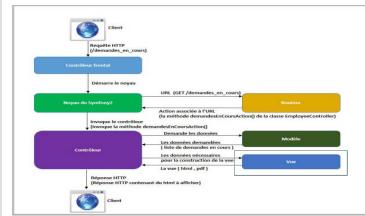
Symfony2 Les TWIGS

AYMEN SELLAYOUTI

Introduction (3)

- Permet de gérer de l'héritage entre Templates et layout
- L'objectif principal de Twig est de permettre aux développeurs de séparer la couche de présentation (Vue du MVC) dans des Templates dédiés, afin de favoriser la maintenabilité du code.
- Idéal pour les graphistes qui ne connaissent pas forcément le langage PHP et qui s'accommoderont parfaitement des instructions natives du moteur, relativement simples à maîtriser.
- Il y a quelques fonctionnalités en plus, comme l'héritage de templates.
- Il sécurise vos variables automatiquement (`htmlentities()`, `addslashes()`)

Introduction (1)



Les bases du langage du moteur Twig

`{{ maVar }}`: Les doubles accolades (équivalent de l'echo dans php ou du `<?=%>` pour les jsp) permettent d'imprimer une valeur, le résultat d'une fonction...

`{% code %}`: Les accolades pourcentage permettent d'exécuter une fonction, définir un bloc...

`{# Les commentaires #}`: Les commentaires.

Introduction (2)

- Moteur de template PHP.
- Développé par l'équipe de Sensio Labs et intégré directement dans le framework **Symfony2**.
- Directement intégré dans Symfony2 (pas besoin de l'installer).

Comment afficher une variable dans les TWIGS (1)

Fonctionnalité	Exemple Twig	Équivalent PHP
Afficher une variable	Variable : {{ MaVariable }}	Pseudo : <?php echo \$MaVariable; ?>
Afficher le contenu d'une case d'un tableau	Identifiant : {{ tab['id'] }}	Identifiant : <?php echo \$tab['id']; ?>
Afficher l'attribut d'un objet, dont le getter respecte la convention <code>\$objet->getAttribut()</code>	Identifiant : {{ user.id }}	Identifiant : <?php echo \$user->getId(); ?>
Afficher une variable en lui appliquant un filtre, ici, « upper » met tout en majuscules :	MaVariable majus : {{ MaVariable upper }}	MaVariable majus: <?php echo strtoupper(\$MaVariable); ?>

Comment afficher une variable dans les TWIGS (2)		
Fonctionnalité	Exemple Twig	Équivalent PHP
Afficher une variable en combinant les filtres. « stripTags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule.	Message : {{ news.texte stripTags title }}	Message : <?php echo ucwords(strip_tags(\$news->getTexte())); ?>
Utiliser un filtre avec des arguments. Attention, il faut que date soit un objet de type Datetime ici.	Date : {{ date date('d/m/Y') }}	Date : <?php echo \$date->format('d/m/Y'); ?>
Concaténer	Identité : {{ nom ~ " " ~ prenom }}	Identité : <?php echo \$nom.' '.\$prenom; ?>

Twig et l'affichage d'un attribut	
Lorsqu'on veut accéder à un attribut d'un objet avec twig on a le fonctionnement suivant pour l'exemple {{objet.attribut}} :	
✓ Vérification si l'objet est un tableau si oui vérifier que nous avons un index valide dans ce cas elle affiche le contenu de object['attribut']	
✓ Simon, si objet est un objet, elle vérifie si attribut est un attribut valide public. Si c'est le cas, elle affiche objet->attribut	
✓ Simon, si objet est un objet, elle vérifie si attribut() est une méthode valide publique. Si c'est le cas, elle affiche object->attribut()	
✓ Simon, si objet est un objet, elle vérifie si getAttribut() est une méthode valide. Si c'est le cas, elle affiche object->getAttribut()	
✓ Simon, et si objet est un objet, elle vérifie si isAttribut() est une méthode valide. Si c'est le cas, elle affiche object->isAttribut()	
✓ Sinon, elle n'affiche rien et retourne null.	

Comment afficher une variable dans les TWIGS : Exemple :	
<pre>(# set permet de déclarer des variables #) (% set MaVariable = 'test' %) Bonjour ({{ MaVariable }})!
 I On affiche l'index du tableau tab envoyé par le contrôleur# Identifiant : {{ tab[1] }}
 (# Application de quelques filtres # MaVariable majus : {{ MaVariable upper }}
 I set texte = '<i>text</i>' % Message sans les filtres : {{ texte }}
 Message avec les filtres : {{ texte stripTags title }}
 I nous nous donne la date système # {{ "now" date("d/m/Y") })
 (# Concaténer # concat : {{ MaVariable ~"~" texte }})</pre>	

Les filtres		
Filtre	Description	Exemple Twig
Upper	Met toutes les lettres en majuscules.	{{ var upper }}
StripTags	Supprime toutes les balises XML.	{{ var stripTags }}
Date	Formatte la date selon le format donné en argument. La variable en entrée doit être une instance de Datetime.	{{ date date('d/m/Y') }} Date d'aujourd'hui : {{ "now" date('d/m/Y') }}
Format	Insère des variables dans un texte, équivalent à printf.	{{ "Il y a %s pommes et %s poires" format(153, nb_poires) }}
Length	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	Longueur de la variable : {{ texte length }} Nombre d'éléments du tableau : {{ tableau length }}

Comment afficher une variable dans les TWIGS : Exemple :	
	

Tests	
is defined l'équivalent de isset en php Rôle vérifie l'existence d'une variable Exemple : {{ if MaVariable is defined }} J'existe {{ endif }}	
even Rôle vérifie si la variable est pair Exemple : {{ if MaVariable is even }} Pair {{ endif }}	odd Rôle vérifie si la variable est impair Exemple : {{ if MaVariable is odd }} Impair {{ endif }}

Structures de contrôle

Les structures que ce soit séquentielles ou itératives sont souvent très proches du langage naturel.

Elles sont introduites entre {%% %%}

Généralement elle se termine par une expression de fin de block

Structure itérative

{% for valeur in valeurs %}

 block à répéter

{% else %}

 Traitements à faire si il n'y
 a aucune valeur

{% endfor %}

Exemple

La formation de l'équipe A est :

{% for joueur in joueurs %}

 {{player.nom}}

{% else %}

 La liste n'a pas encore été renseignée

{% endfor %}

Structure conditionnelle

IF

Syntaxe :

{% if cond %}

Block de traitement

{%endif%}

Exemple

{% if employee.salaire < 250 %}

ce salaire est inférieur au smig

{%endif%}

Structure itérative

La boucle for définit une variable loop ayant les attributs suivants :

Variable	Description
{loop.index}	Le numéro de l'itération courante (en commençant par 1).
{loop.index0}	Le numéro de l'itération courante (en commençant par 0).
{loop.revindex}	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 1).
{loop.revindex0}	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 0).
{loop.first}	true si c'est la première itération, false sinon.
{loop.last}	true si c'est la dernière itération, false sinon.
{loop.length}	Le nombre total d'itérations dans la boucle.

Structure conditionnelle

IF else elseif

Syntaxe :

{% if cond %}

block traitement

{% elseif cond2 %}

block traitement

{% else %}

block traitement

{% endif %}

Exemple

```
{% if maison.temperaturer <0%}
Très Froid
{% elseif maison.temperaturer <10 %}
Froid
{% else %}
Bonne température
{% endif %}
```

Accès aux Template

Les Templates doivent se trouver dans l'un des deux emplacements suivants :

> app/resources/views

> /resources/views d'un bundle

Afin d'accéder aux Template, une convention de nommage est définie :

NomBundle:DossierFichier:page.twig

Exemples

:index.html.twig // ce fichier se trouve directement dans app/resources/views

RT4Bundle:index.html.twig // ce fichier se trouve dans src/RT4Bundle/resources/views

RT4Bundle:Multimedia:index.html.twig // ce fichier se trouve dans src/RT4Bundle/Multimedia/resources/views

Nommage des pages TWIG

Par convention le nommage des vues dans symfony se fait selon la convention suivante :
NomPage.FormatFinal.MoteurdeTemplate
Exemple
index.html.twig
Le nom du fichier est index, le format final sera du html et le moteur de template suivi est le TWIG

Héritage (3)

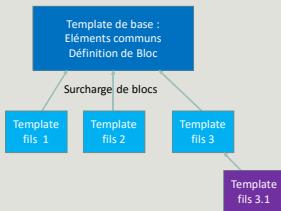
= Afin d'hériter d'un Template père il faut étendre ce dernier avec la syntaxe suivante :

```
= {% extends '::TemplateDeBase' %}  
= Exemple :  
= {% extends '::base.html.twig' %}  
= Si le fils ne surcharge aucun des blocs et n'ajoute rien on aura le même affichage que pour la base
```



Héritage (1)

= Vision proche de l'héritage dans l'orienté objet



Héritage (4)

= L'élément de base de l'héritage est le **bloc**

```
= Un bloc est défini comme suit : {% block nomBloc%} contenu du bloc{%endblock%}  
= Pour changer le contenu de la page il faut surcharger obligatoirement le block body (le block central qu'elle que soit son nom)  
| extends '::base.html.twig'  
| block body | Bonjour j'ai pris ma indépendance et j'ai défini mon propre body (| endblock |)  
= Si on ne surcharge pas le body on aura l'erreur suivante  
| extends '::base.html.twig'  
| block body | Bonjour j'ai pris non indépendance et j'ai défini mon propre body  
| endblock |  
| endblock | A template that extends another one cannot have a body in BlockBodyD  
| endblock | BlockBodyD  
| endblock |
```



Héritage (2)

= Exemple de Template de base

```
<?php  
namespace App\\Controller;  
use Symfony\\Component\\HttpFoundation\\Response;  
use Symfony\\Component\\Templating\\TemplateEngineInterface;  
use Symfony\\Component\\Templating\\TemplateTrait;  
  
class BaseController extends Controller  
{  
    use TemplateTrait;  
  
    public function index(): Response  
    {  
        $this->render('base.html.twig');  
        return $this->render('base.html.twig');  
    }  
}
```



Héritage (5)

= Pour récupérer le contenu d'un bloc père il suffit d'utiliser la méthode parent()

```
| extends '::base.html.twig' |  
| block body |  
| { parent() } |> bonjour j'ai pris mon indépendance et j'ai défini mon propre body  
| endblock |  
| endblock |
```

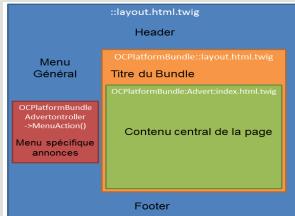


= Ceci peut être aussi appliquée pour toute la hiérarchie

```
| extends 'base.html.twig' |  
| block body |  
| { parent() } |> bonjour j'ai pris mon indépendance et j'ai défini mon propre body  
| endblock |
```



Héritage (6)



<https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/le-moteur-de-templates-twig>

Inclusion Template

Afin d'inclure un Template ou un fragment de code dans un autre template on utilise la syntaxe suivante :

```
 {{ include('NomBundle:Dossier:nomTemplate', {'labelParam1': param1,'labelParam2': param2,... }) }}
```

Exemple :

```
 {{ include('Rt4AsBundle:Default:section.html.twig', {'Section': section})}}
```

Inclusion Contrôleur

Que faire si on veut inclure un template dynamique ? Un template des meilleurs vente un template des articles les plus vus les derniers cours postés ...

Solution :

Inclure un contrôleur sans ou avec des paramètres

Syntaxe :

```
 {{ render(controller('NomBundle:NomController:NomAction', {'labelParam1': param1,'labelParam2': param2,...})) }}
```

E-1

[Glossary](#) | [Feedback](#) | [Report a Problem](#) | [Study Group Tools](#) | [Syllabus](#) | [Help](#)

Génération de liens avec TWIG (1)

Twig permet de générer des liens à partir des noms de root en utilisant la fonction `path`.

```
rt4_as_homepage:
    path: /hello/{name}
    defaults: { _controller: Rta4sBundle:Default:index }

rt4_as_base:
    path: /base
    defaults: { _controller: Rta4sBundle:Default:base }

rt4_as_file:
    path: /files
    defaults: { _controller: Rta4sBundle:Default:file }

rt4_as_pfifile:
    path: /pfifiles
    defaults: { _controller: Rta4sBundle:Default:pfifitxt } 
```

[!] entend 'Rta4sBundle:Default:fila.html.twig'

[!] block body

[[parent()]])>

Revoir J'ai pris un aussi une indépendance de je suis le parent fil et j'ai défini mon propre body des deux fils

Q: {{ fil }}{{ fil }} et la base {{ base }} son grand père est {{ grandPere }}

A: {{ fil }}{{ fil }} et la file {{ file }} ne gère pas les {{ file }}

[!] endblock

Génération de liens avec TWIG (2)

Passage de paramètres avec path

Syntaxe

```

  { path:'root', { param1: param1, param2: param2, ... } } }

rt4a_base:
  path: '/base'
  defaults: { controller: controller, Rt4aBundle:Default:index }

rt4a_base:
  path: '/base'
  defaults: { controller: controller, Rt4aBundle:Default:base }

rt4a_files:
  path: '/files'
  defaults: { controller: controller, Rt4aBundle:Default:file }

rt4a_pf_files:
  path: '/pf/files'
  controller: controller, Rt4aBundle:Default:rootFile [ ! block ]

```

Génération de liens avec TWIG (3)

Pour générer l'url absolue nous utilisons `url()`

Syntaxe

Génération de liens avec TWIG (4)

Pour générer le path d'un fichier (img, css, js,...) nous utilisons la fonction asset.

Cette fonction permet la portabilité de l'application en permettant une génération automatique du path du fichier indépendamment de l'emplacement de l'application.

Exemple :

➤ Si l'application est hébergé directement dans la racine de l'hôte alors le path des images est /images/nomImg.

➤ Si l'application est hébergé dans un sous répertoire de l'hôte alors le path des images est /nomApp/images/nomImg.

Syntaxe :

asset('ressource')

Exemples

```

```

```
<link href="{{ asset('css/blog.css') }}" rel="stylesheet" type="text/css" />
```

Surcharge de Template

Afin de surcharger les Template d'un Bundle, nous nous basons sur le fonctionnement de base de symfony2.

Lorsque le contrôleur fait appel à un Template dans un Bundle Symfony2 cherche dans 2 emplacements selon l'ordre suivant :

- 1) app/Resources/NomBundle/views/DossierTemplate/nomTemplate.html.twig
- 2) src/NomAppli/NomBundle/Resources/views/DossierTemplate/nomTemplate.html.twig

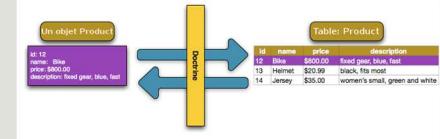
Pour surcharger le template il suffit donc de copier ce dernier vers le dossier app/Resources/NomBundle/views/DossierTemplate/ que vous devez créer vous-même et personnaliser le Template à votre guise.

aymen.sellaouti@gmail.com

Symfony2 L'ORM DOCTRINE

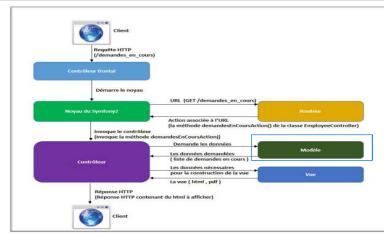
AYMEN SELLAYOUTI

Introduction (3)



- Doctrine : ORM le plus utilisé avec Symfony2
- Associe des classes PHP avec les tables de votre BD (mapping)

Introduction (1)



2

Les entités (1)

- Objet PHP
- Les entités représentent les objets PHP équivalents à une table de la base de données.
- Une entité est généralement composée par les attributs de la table ainsi que leurs getters et setters
- Manipulé par l'ORM

5

Introduction (2)

- ORM : Object Relation Mapper
- Couche d'abstraction
- Gérer la persistance des données
- Mapper les tables de la base de données relationnelle avec des objets
- Crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.
- Propose des méthodes prédéfinies



3

Les entités (2) Exemple

```

<?php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Student
 * @ORM\Table(name="student")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\StudentRepository")
 */
class Student
{
    /**
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @ORM\Column(name="name", type="string", length=255)
     */
    private $name;

    /**
     * @ORM\Column(name="age", type="integer")
     */
    private $age;

    /**
     * @ORM\Column(name="email", type="string", length=255)
     */
    private $email;
}

private $student;

public function __construct()
{
    $this->student = new \Doctrine\Common\Collections\ArrayCollection();
}

public function addStudent(Student $student)
{
    $this->student[] = $student;
}

public function removeStudent(Student $student)
{
    $this->student->removeElement($student);
}

public function getStudent()
{
    return $this->student;
}
  
```

6

Configuration des entités

Configuration Externe : YAML, XML, PHP

Configuration Interne : annotations

Choix de la configuration ?

Deux Visions :

Pro-Externe

Séparation complète des fonctionnalités spécialement lorsque l'entité est conséquente

Pro-Interne

Plus facile et agréable de chercher dans un seul fichier l'ensemble des informations, plus de visibilité

?

Mapping : Les annotations (3) Table

Permet de spécifier le nom de la table dans la base de données à associer à l'entité

Applicable sur une classe et placée avant la définition de la classe en PHP

Fauteuille sans cette annotation le nom de la table sera automatiquement le nom de l'entité

Généralement utilisable pour ajouter des préfixes ou pour forcer la première lettre de la table en minuscule

Syntaxe : `@ORM\Table()`

Exemple :

```
/**  
 *  
 * @ORM\Table("animal")  
 * @ORM\Entity(repositoryClass="Rt4\AsBundle\Entity\animalRepository")  
 */
```

10

Mapping : Les annotations (1)

Rôle : Faire le lien entre les entités et les tables de la base de données

Lien à travers les métadonnées

Remarque : Un seul format par bundle (impossibilité de mélanger)

Syntaxe :

```
/**  
 * les différentes annotations  
 */
```

8

Mapping : Les annotations (4) Column

Permet de définir les caractéristiques de la colonne concernée

Applicable sur un attribut de classe juste avant la définition PHP de l'attribut concerné.

Syntaxe : `@ORM\Column()`

Exemple :

```
/**  
 *  
 * @ORM\Column(param1="valParam1", param2="valParam2")  
 */
```

11

Mapping : Les annotations (2) Entity

Permet de définir un objet comme une entité

Applicable sur une classe

Placée avant la définition de la classe en PHP

Syntaxe :

`@ORM\Entity`

Paramètres :

repositoryClass (facultatif). Permet de préciser le namespace complet du repository qui gère cette entité.

Exemple :

`@ORM\Entity(repositoryClass="Rt4\AsBundle\Entity\animalRepository")`

9

Mapping : Les annotations (5) Les paramètres de Column

Paramètre	Valeur par défaut	Utilisation
type	string	Définit le type de colonne comme nous venons de le voir.
name	Nom de l'attribut	Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet
length	255	Définit la longueur de la colonne (pour les strings).
unique	false	Définit la colonne comme unique (Exemple : email).
nullable	false	Permet à la colonne de contenir des NULL.
precision	0	Définit la précision d'un nombre à virgule(decimal)
scale	0	le nombre de chiffres après la virgule (decimal)

12

Mapping : Les annotations (6)

Les types de Column

Type Doctrine	Type SQL	Type PHP	Utilisation
string	VARCHAR	string	Toutes les chaînes de caractères jusqu'à 255 caractères.
integer	INT	integer	Tous les nombres jusqu'à 2 147 483 647.
smallint	SMALLINT	integer	Tous les nombres jusqu'à 32 767.
bigint	BIGINT	string	Tous les nombres jusqu'à 9 223 372 036 854 775 807.
boolean	BOOLEAN	boolean	Les valeurs booléennes true et false.
decimal	DECIMAL	double	Les nombres à virgule.

13

Génération des entités

Deux méthodes pour générer les entités :

- Méthode manuelle (non recommandée)
 - Créer la classe
 - ajouter le mapping
 - ajouter les getters et les setters (manuellement ou en utilisant la commande suivante : `php app/console doctrine:generate:entities` (elle crée les getters et les setters de toutes les entités))
- Méthode en utilisant les commandes
 - Il suffit de lancer la commande suivante :
 - Ajouter les attributs ainsi que les paramètres qui vont avec
 - Une fois terminé, Doctrine génère l'entité avec toutes les métadonnées de mapping

16

Mapping : Les annotations (7)

Les types de Column

Type Doctrine	Type SQL	Type PHP	Utilisation
date ou datetime	DATETIME	objet DateTime	Toutes les dates et heures.
time	TIME	objet DateTime	Toutes les heures.
text	CLOB	string	Les chaînes de caractères de plus de 255 caractères.
object	CLOB	Type de l'objet stocké	Stocke un objet PHP en utilisant serialize/unserialize.
array	CLOB	array	Stocke un tableau PHP en utilisant serialize/unserialize.
float	FLOAT	double	Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur.

14

Gestion de la base de données (1)

Configuration de l'application

Afin de configurer la base de données de l'application il faut renseigner les champs dans le fichier `parameters.yml`

```
parameters:
    database_host: 127.0.0.1,
    database_port: null,
    database_name: rt4,
    database_user: root,
    database_password: null,
    mailer_transport: smtp,
    mailer_host: 127.0.0.1,
    mailer_user: null,
    mailer_password: null,
    secret: e965bc083397f3b669db685fecdd03398d4b783,
    database_driver: pdo_mysql,
    database_path: null }
```

17

Mapping : Les annotations (8)

Conventions de Nommage

Même s'il reste facultatif, le champ « name » doit être modifié afin de respecter les conventions de nommage qui diffèrent entre ceux de la base de données et ceux de la programmation OO.

➤ Les noms de classes sont écrites en « Pascal Case » TheEntity.

➤ Les attributs de classes sont écrites en « camel Case » oneAttribute.

➤ Les noms des tables et des colonnes en SQL sont écrites en minuscules, les mots sont séparés par « _ » one_table, one_column.

15

Gestion de la base de données (2)

Création de base de données

Afin de créer la base de données du projet 2 méthodes sont utilisées :

- Manuelle en utilisant le SGBD (le nom de la BD doit être le même que celui mentionné dans le fichier `parameters.yml`)

➤ En utilisant la ligne de command avec la commande suivante :

- `php app/console doctrine:database:create`
- Une base de données avec les propriétés mentionnées dans `parameters.yml` sera automatiquement générée

18

Gestion de la base de données (3)

Création des tables de la base de données

Afin de créer les tables de la base de données doctrine se base sur les entités placées dans les différentes dossier Entity des différents bundles de l'application.

- Deux commandes permettent de créer les tables de la BD :
 - `php app/console doctrine:schema:create`
 - `php app/console doctrine:schema:update --force` // utilisable pour la création et pour la mise à jour d'une table
- Astuce :
 - `php app/console doctrine:schema:update --dump-sql` // Affiche les requêtes SQL à exécuter pour la BD

Cette astuce permet de vérifier la requête à exécuter avant la mise à jour de la table.

19

Gestion de la base de données (5)

Le service EntityManager (2)

Les repositories (dépôts)

Des classes PHP dont le rôle est de permettre à l'utilisateur de récupérer des entités d'une classe donnée.

Syntaxe :

```
Pour accéder au repository de la classe MaClasse on utilise l'EntityManager  
$repo = $entityManager->getRepository(e. Bundle:MAClasse);  
Quelques méthodes offertes par le repository :  
$repository->findAll(); // récupère tous les entités (enregistrements) relatifs à l'entité associé au repository  
$repository->find($id); // requête sur la clé primaire  
$repository->findBy(); // retourne un ensemble d'entités avec un filtrage sur plusieurs critères (nbre donné)  
$repository->findOneBy(); // même principe que findBy mais une seule entité  
$repository->findOneByNomPropriété(); $repository->findOneByNomPropriété();
```

22

Gestion de la base de données (4)

Le service Doctrine

Rôle : permet la gestion des données dans la BD : [persistance](#) des données et [consultation](#) des données.

Méthode :

- `$this->get('doctrine');`
- `$this->getDoctrine();` //helper (raccourcie de la classe Controller)

20

Gestion de la base de données (5)

Le service EntityManager (3)

findAll()

Rôle : retourne l'ensemble des entités qui correspondent à l'entité associé au repository. Le format du retour est un Array

Exemple :

```
//On récupère le repository de l'entity manager correspondant à l'entité Etudiant  
$repository = $this  
->getDoctrine()  
->getManager()  
->getRepository('OCPlatformBundle:Advert') ;  
//On récupère la liste des étudiants  
$listAdverts = $repository->findAll();  
Généralement le tableau obtenu est passé à la vue (TWIG) et est affiché en utilisant un foreach
```

23

Gestion de la base de données (5)

Le service EntityManager (1)

Rôle : L'interface ORM proposée par doctrine offrant des méthodes prédéfini pour persister dans la base ou pour chercher mettre à jour ou supprimer une entité.

Méthode :

- `$entityManager = $this->get('doctrine.orm.entity_manager')`
- `$this->getDoctrine()->getManager();` //helper (raccourcie de la classe Controller)

Remarque il y a aussi la méthode `getEntityManager()` mais elle est devenu obsolète.

21

Gestion de la base de données (5)

Le service EntityManager (4)

find(\$id)

Rôle : retourne l'entité qui correspond à la clé primaire passé en argument. Généralement cette clé est l'id.

Exemple :

```
//On récupère le repository de l'entity manager correspondant à l'entité Etudiant  
$repo = $this  
->getDoctrine()  
->getManager()  
->getRepository('Rt4AsBundle:Etudiant');  
//on lance la requête sur l'étudiant d'id 2  
$etudiant = $repository->find(2);
```

24

Gestion de la base de données (5) Le service EntityManager (5)

findBy()

Rôle : retourne l'ensemble des entités qui correspondent à l'entité associé au répositorie comme `findAll` sauf qu'elle permet d'[effectuer un filtrage](#) sur un ensemble de critères passés dans un Array. Elle offre la possibilité de [trier](#) les entités sélectionnées et facilite la [pagination](#) en offrant un nombre de valeur de retour.

Syntaxe:

```
$repository->findBy( array $criteria, array $orderBy = null, $limit = null, $offset = null);  
  
Exemple :  
  
$repository = $this->getDoctrine()->getManager() ->getRepository('Rt4AsBundle:Etudiant');  
  
$listeEtudiants = $repository->findBy(array('section' => 'RT4','nom' => 'Mohamed'),  
array('date' => 'desc'),10, 0);
```

25

Gestion de la base de données (5) Le service EntityManager (8)

findOneByPropriété()

Rôle : En remplaçant le mot [Propriété](#) par le [nom d'une des propriétés de l'entité](#), la fonction va faire le même rôle que `findOneBy` mais avec un seul critère.

Exemple :

```
$repository = $this->getDoctrine()->getManager() ->getRepository('Rt4AsBundle:Etudiant');  
  
$listeEtudiants = $repository->findOneByNom('Aymen');
```

26

Gestion de la base de données (5) Le service EntityManager (6)

findOneBy()

Rôle : Même principe que `FindBy` mais en retournant une seule entité ce qui élimine automatiquement les paramètres d'ordre de limite et d'offset

Exemple :

```
$repository = $this->getDoctrine()->getManager() ->getRepository('Rt4AsBundle:Etudiant');  
  
$Etud = $repository->findOneBy(array('section' => 'RT4','nom' => 'Mohamed'));
```

26

Gestion de la base de données (5) Le service EntityManager (9)

Enregistrement des données

Etant un ORM, Doctrine traite les objets PHP

Pour enrégistrer des données dans la BD il faut préparer les objets contenant ces données la

La méthode [persist\(\)](#) de l'entityManager permet d'[associer les objets à persister avec Doctrine](#)

Afin d'[exécuter les requêtes sur la BD](#) (enrégistrer les données dans la base) il faut utiliser la méthode [flush\(\)](#)

L'utilisation de flush permet de profiter de la force de Doctrine qui utilise les [Transactions](#)

[La persistance agit de la même façon avec l'ajout \(insert\) ou la mise à jour \(update\)](#)

29

Gestion de la base de données (5) Le service EntityManager (7)

findByPropriété()

Rôle : En remplaçant le mot [Propriété](#) par le [nom d'une des propriétés de l'entité](#), la fonction va faire le même rôle que `findBy` mais avec un seul critère qui est le nom de la propriété et sans les options.

Exemple :

```
$repository = $this->getDoctrine()->getManager() ->getRepository('Rt4AsBundle:Etudiant');  
  
$listeEtudiants = $repository->findByNom('Aymen');
```

27

Gestion de la base de données (5) Le service EntityManager (9)

```
public function AddUpdateAction($id)  
{  
    // on récupère notre entity manager  
    $em = $this->getDoctrine()->getManager();  
    // on récupère le repository  
    $repo = $em->getRepository('Rt4AsBundle:Etudiant');  
    // on récupère l'entité  
    $etudiant = new Etudiant();  
    $etudiant->setNom('nouvel etudiant')  
    ->setCin('123456789')  
    ->setDateNaissance(new \DateTime())  
    ->setPrenom('new prenom')  
    ->setNumEtudiant(1234);  
    // on persiste l'entité  
    // on récupère une entité qui est automatiquement associé à Doctrine plus besoin de la persister  
    $etud = $repo->find($id);  
    if ($etud){  
        $etud->setCin(2782);  
        $em->flush();  
        // on récupère l'ensemble des entités et on le renvoie à la page d'affichage  
        $etudiants = $em->getRepository('Rt4AsBundle:Etudiant')->findAll();  
        return $this->render('Rt4AsBundle:Default:liste.html.twig', array(  
            'etudiants' => $etudiants,  
        ));  
    }  
}
```

Enregistrement et mise
à jour des données
(Exemple)

30

Gestion de la base de données (5) Le service EntityManager (9)

Suppression d'une entité

La méthode `remove()` permet de supprimer une entité

```
public function DeleteAction($id)
{
    // on récupère notre entity manager et notre repository
    $em = $this->getDoctrine()->getRepository();
    $repo = $em->getRepository('Rt4AsBundle:Etudiant');
    //on récupère une entité qui est automatiquement associé à Doctrine plus besoin de la persister
    $etud = $repo->find(410);
    if ($etud) {
        $em->remove($etud);
        $em->flush();
    }
    //on récupère l'ensemble des entités et on le transmet à la page d'affichage
    $etudiants = $em->getRepository("Rt4AsBundle:Etudiant")->findAll();
    return $this->render('Rt4AsBundle:Default:list.html.twig', array(
        'etudiants' => $etudiants,
    ));
}
```

31

Gestion de la base de données (8) QueryBuilder

Constructeur de requête DOCTRINE Alternative au DQL

Accessible via le [Repository](#)

Le résultat fourni par la méthode `getQuery` du [QueryBuilder](#) permet de générer la requête en DQL.

De même que le `createQuery`, une fois la requête créée la méthode `getResult()` permet de récupérer un tableau de résultat.

```
$repository=$this->getEntityManager()->getRepository('Rt4AsBundle:Etudiant');
$query = $repository->createQueryBuilder('e')
    ->where('e.age > :age')
    ->setParameter('age', 22)
    ->orderBy('e.age', 'ASC')
    ->getQuery();
$etudiants = $query->getResult();
```

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>

34

Gestion de la base de données (6) Création de requêtes

Les requêtes de doctrine sont écrites en utilisant le langage de doctrine le Doctrine Query Language [DQL](#), ou en utilisant un Objet créateur de requêtes le [CreateQueryBuilder](#)

`createQuery` : Méthode de l'Entity Manager

```
/*query = $em->createQuery()
 *SELECT e
 *FROM Rt4AsBundle:Etudiant e
 *WHERE e.age > :age
 *ORDER BY e.age ASC*/
*$setParameter('age', '22');
$etudiants = $query->getResult();
```

➤ `CreateQueryBuilder` : Méthode du répository

```
getRepository($this->getEntityManager())->getRepository('Rt4AsBundle:Etudiant');
$query = $repository->createQueryBuilder('e')
    ->where('e.age > :age')
    ->setParameter('age', '22')
    ->orderBy('e.age', 'asc')
    ->getQuery();
$etudiants = $query->getResult();
```

32

Externalisation des requêtes dans un dépôt personnalisé pour chaque entité

Bu :

➤ Isoler la couche modèle

➤ Réutilisabilité

Façabilité :

Ajouter le dépôt dans le mapping de l'entité (@ORM\Entity(repositoryClass=<> NotreRepository </>)

Exemple :

```
/** 
 * Etudiant
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="Rt4AsBundle\Entity\EtudiantRepository")
 */
class Etudiant
{
```

35

Gestion de la base de données (7) CreateQuery et DQL

Le DQL peut être défini comme une adaptation du SQL, adapté à l'orienté objet et donc à DOCTRINE

La requête est défini sous forme d'une chaîne de caractère

Afin de créer une requête DQL il faut utiliser la méthode `createQuery()` de l'Entity Manager

La méthode `setParameter('label','valeur')` permet de définir un paramètre de la requête

Pour définir plusieurs paramètres ou bien utiliser `setParameter` plusieurs fois ou bien la méthode `setParameters(array('label1','valeur1', 'label2','valeur2'),...,'labelN','valeurN')`

Une fois la requête créée la méthode `getResult()` permet de récupérer un tableau de résultat

Le langage DQL est explicité dans le lien suivant :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html>

33

Gestion des relations entre les entités (1) Les types de relation

Les entités de la BD présentent des relations d'association :

- A **OneToOne** B : à une entité A on associe une entité de B et inversement
- A **ManyToOne** B : à une entité B on associe plusieurs entité de A et à une entité de A on associe une entité de B
- A **ManyToMany** B : à une entité de A on associe plusieurs entité de B et inversement

36

Gestion des relations entre les entités (2) Relation unidirectionnelle et bidirectionnelle

La notion de navigabilité de UML est la source de la notion de relation unidirectionnelle ou bidirectionnelle

Une relation est dite navigable dans les deux sens si les deux entité doivent avoir une trace de la relation.

Exemple : Supposons que nous avons les deux classes CandidatPrésidentielle et Electeur.

L'électeur doit savoir à qui il a voté donc il doit sauvegarder cette information par contre le candidat pour cause d'anonymat de vote ne doit pas connaître les personnes qui ont voté pour lui.

On aura donc un attribut Candidat dans la table Electeur mais pas de collection ou tableau nommé électeur dans la table CandidatPrésidentielle. Ici on a une relation **unidirectionnelle**.

37

Gestion des relations entre les entités (5) ManyToOne Unidirectionnelle

➤ Relation **unidirectionnelle** puisque Section ne référence pas Etudiant

➤ L'annotation `@ORM\JoinColumn` n'est pas obligatoire ici vu que par défaut il va référencer l'id de la table avec laquelle il a une jointure

```
/**Entity */
Class Etudiant
{
    ...
    /**
     * @ORM\ManyToOne(targetEntity=>Section)
     * @ORM\JoinColumn(name=> section_id",referencedColumnName="id")
     */
    private $section;
}
//-
//-
//-
//-
//-
//-
}
```

40

Gestion des relations entre les entités (3) OneToOne unidirectionnelle

➤ Relation **unidirectionnelle** puisque Media ne référence pas Etudiant

➤ L'annotation `@ORM\JoinColumn` n'est pas obligatoire ici vu que par défaut il va référencer l'id de la table avec laquelle il a une jointure

```
/**Entity */
Class Etudiant
{
    ...
    /**
     * @ORM\OneToOne(targetEntity=> Media)
     * @ORM\JoinColumn(name=> media_id",
     referencedColumnName="id")
     */
    private $media;
}
//-
//-
//-
//-
//-
//-
}
```

38

Gestion des relations entre les entités (6) OneToMany Bidirectionnelle

➤ Si nous voulons connaître dans l'objet section l'ensemble des étudiants qui lui sont affectés alors on doit avoir une relation bidirectionnelle

➤ Section aussi doit référencer Etudiant

➤ On aura une relation **OneToMany** coté Section puisqu'à « One » Section on a « Many » Etudiants

➤ On doit ajouter l'attribut `mappedBy` côté `OneToOne` et `inversedBy` côté `ManyToOne`

➤ On doit spécifier dans le `constructeur` du `OneToMany` que l'attribut mappé est de type `ArrayList` en l'instantiant

```
/**Entity */
Class Section
{
    ...
    /**
     * @ORM\OneToMany(targetEntity=> Etudiant")
     */
    private $etudiants;
    ...
    public function __construct()
    {
        $this->etudiants = new ArrayCollection();
    }
}
/**Entity */
Class Etudiant
{
    ...
    /**
     * @ORM\ManyToOne(targetEntity=> Section)
     * @ORM\JoinColumn(name=> section_id",referencedColumnName="id")
     */
    private $section;
}
//-
//-
//-
//-
//-
//-
}
```

41

Gestion des relations entre les entités (4) OneToOne Bidirectionnelle

➤ Si nous voulons qu'à partir du media on peut directement savoir à quel étudiant il appartient nous devons faire une relation bidirectionnelle

➤ Media aussi doit référencer Etudiant

```
/**Entity */
Class Etudiant
{
    ...
    /**
     * @ORM\OneToOne(targetEntity=> Media)
     * @ORM\JoinColumn(name=> media_id",referencedColumnName="id")
     */
    private $media;
}
//-
//-
//-
//-
//-
//-

```

39

Gestion des relations entre les entités (7) ManyToMany

➤ Relation **unidirectionnelle** puisque Cours ne référence pas Prof

➤ Ici on peut savoir quels sont les cours de chaque étudiant mais pas l'inverse (on peut l'extraire via une requête)

```
/**Entity */
Class Cours
{
    ...
}
/**Entity */
Class Etudiant
{
    ...
    /**
     * @ORM\ManyToMany(targetEntity=> "Cours")
     * @ORM\JoinTable(name=> Prof_cours")
     */
    private $cours;
}
//-
//-
//-
//-
//-
//-

```

40

Gestion des relations entre les entités (8) ManyToMany Bidirectionnelle

➤ Ici on pourra directement connaître les prof qui enseignent chaque cours et les cours de chaque prof

➤ On doit spécifier dans les deux constructeurs que l'attribut mappé est de type ArrayCollection en l'instanciant

➤ Cette relation peut être traduite à deux relation OneToMany/ManyToOne entre les 3 classes participantes.

```
/**Entity**/
Class Prof
{
    ...
    /**
     * @ORM\ManyToMany(targetEntity="Insat\RelationBundle\Entity\Cours", inversedBy="prof")
     * @ORM\JoinColumn(nullable=false)
     */
    private $cours;
}

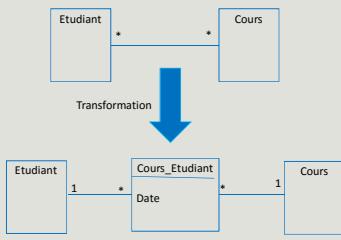
/**Entity**/
Class Cours
{
    ...
    /**
     * @ORM\ManyToMany(targetEntity="Insat\RelationBundle\Entity\Prof",mappedBy="cours")
     */
    private $prof;
}

private $cours;
function __construct(){ $this->cours = new ArrayCollection(); }

private $prof;
function __construct(){ $this->prof = new ArrayCollection(); }
```

aymen.sellaouti@gmail.com

Gestion des relations entre les entités (9) Exemple ManyToMany (1)



44

Gestion des relations entre les entités (9) Exemple ManyToMany (2)

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

    /**
     * @var integer
     * @ORM\Column(name="annee", type="integer")
     */
    private $annee;

    /**
     * @var Date
     * @ORM\ManyToOne(targetEntity="Insat\RelationBundle\Entity\Date")
     * @ORM\JoinColumn(nullable=false)
     */
    private $date;

    /**
     * @var ArrayCollection
     * @ORM\ManyToMany(targetEntity="Insat\RelationBundle\Entity\Cours", inversedBy="cours")
     */
    private $cours;

    /**
     * @var Cours
     * @ORM\ManyToMany(targetEntity="Insat\RelationBundle\Entity\Etudiant", inversedBy="etudiants")
     * @ORM\JoinColumn(nullable=false)
     */
    private $etudiants;
}

class Cours
{
    /**
     * @var integer
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
}

class Date
{
    /**
     * @var integer
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
}
```

45

Symfony2 Les formulaires

AYMEN SELLAYOUTI

Références

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

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

Qu'est ce qu'un formulaire symfony2

La philosophie de Symfony2 pour les formulaires est la suivante :

- Un formulaire est l'image d'un objet existant
- Le formulaire sert à alimenter cet objet.

Classe Exemple
{
private \$id;
private \$nom;
private \$age;
}



Nom

Age

Introduction

- Rôle très important dans le web
- Vitrine, interface entre les visiteurs du site web et le contenu du site
- Généralement traité en utilisant du html <form> ... </form>
- Symfony2 et les formulaires : [le composant Form](#)
- Bibliothèque dédiée aux formulaires

Comment créer un formulaire (1)

La création d'un formulaire se fait à travers le Constructeur de formulaire FormBuilder

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($objetImage)
```

Pour indiquer les champs à ajouter au formulaire on utilise la méthode `add` du `FromBuilder`

La méthode `add` contient 3 paramètres :

- 1) le nom du champ dans le formulaire
- 2) le type du champ <http://symfony.com/fr/doc/current/reference/forms/types.html>
- 3) un array qui contient des options spécifiques au type du champ

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)->add('nom', 'text')  
->add('age', 'integer')
```

Comment créer un formulaire (2)

Pour récupérer le formulaire créé, il faut utiliser la méthode `getForm()`

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)->add('nom','text')  
->add('age','integer')  
->getForm();
```

Comment créer un formulaire (3)

La création du formulaire se fait de 2 façons différentes :

- 1) Dans le contrôleur qui va utiliser le formulaire
- 2) En externalisant la définition dans un fichier

?

Affichage du formulaire dans TWIG(1)

Afin d'afficher le formulaire créé, il faut transmettre la vue de ce formulaire à la page Twig qui doit l'accueillir.

La méthode `createView` de l'objet `Form` permet de créer cette vue

Il ne reste plus qu'à l'envoyer à la page twig en question

Exemple :

```
$form= $this->createFormBuilder($exemple)->add('nom','text')
        ->add('age','integer')
        ->getForm();
return $this->render('Rt4AppBundle:Default:myform.html.twig', array(
    'form'= >$form->createView());
```

8

Affichage du formulaire dans TWIG (3) Les composants du formulaire (1)

`form_start()` affiche la balise d'ouverture du formulaire HTML, soit `<form>`. Il faut passer la variable du formulaire en premier argument, et les paramètres en deuxième argument. L'index attr des paramètres, et cela s'appliquera à toutes les fonctions suivantes, représente les attributs à ajouter à la balise générée, ici le `<form>`. Il nous permet d'appliquer une classe CSS au formulaire, ici `form-horizontal`.

Exemple : `{ { form_start(form, {attr: {class: 'form-horizontal'}}) } }`

`form_errors()` affiche les erreurs attachées au champ donné en argument.

`form_label()` affiche le label HTML du champ donné en argument. Le deuxième argument est le contenu du label.

10

Affichage du formulaire dans TWIG (3) Les composants du formulaire (2)

`form_widget()` affiche le champ HTML lui-même (que ce soit `<input>`, `<select>`, etc).

Exemple : `{ { form_widget(form.title, {attr: {class: 'form-control'}}) } }`

`form_row()` affiche le label, les erreurs et le champ.

`form_rest()` affiche tous les champs manquants du formulaire (dans notre cas, juste le champ CSRF puisque nous avons déjà affiché à la main tous les autres champs). `form_end()` affiche la balise de fermeture du formulaire HTML, soit `</form>`.

Remarque : Certains types de champ ont des options d'affichage supplémentaires qui peuvent être passées au widget. Ces options sont documentées avec chaque type, mais l'option `attr` est commune à tous les types et vous permet de modifier les attributs d'un élément de formulaire.

11

Affichage du formulaire dans TWIG (2)

Deux méthodes permettent d'afficher le formulaire dans Twig :

- 1) Afficher directement la totalité du formulaire avec la méthode `form`

```
{ { form(nomDuFormulaire) } }
```

- 2) Afficher les composants du formulaire séparément un à un (généralement lorsqu'on veut personnaliser les différents champs)

9

Gestion de la soumission des Formulaires (1)

La gestion de la soumission des formulaires se fait à l'aide de la méthode `handleRequest($request)`

`HandleRequest` vérifie si la requête est de type POST. Si c'est le cas, elle va mapper les données du formulaire avec l'objet affecté au formulaire en utilisant les setters de cet objet.

```
public function testFormation()
{
    $tache = new Tache();
    $tache->setTitre('terminer les formulaires');
    $tache->setDescription('Générer les formulaires');
    $form = $this->createFormBuilder($tache)
        ->add('action',$this->generateUrl('rt4_1st_Form'))
        ->add('tache','text')
        ->add('matache','text')
        ->add('date','date')
        ->add('submit','submit');
    $form->handleRequest($request);
    if($form->isValid())
    {
        $em=&tache->getDoctrine()->getManager();
        $em->persist($tache);
        $em->flush();
        return $this->render('Rt4AppBundle:Default:succesTache.html.twig');
    }
    return $this->render('Rt4AppBundle:Default:myform.html.twig',array('form'= >$form->createView()));
}
```

12

Gestion de la soumission des Formulaires (2)

```
public function testFormulation()
{
    $tache = new Tache();
    $form = $this->createFormBuilder($tache)
        ->setAction($this->generateUrl('rt4_add_form'))
        ->setMethod('POST')
        ->add('text', 'text')
        ->add('date', 'date')
        ->getForm();
    return $this->render('Rt4AsBundle:Default:myform.html.twig', array('form'=>$form->createView()));
}

public function addFormAction(Request $request)
{
    $tache = new Tache();
    $form = $this->createFormBuilder($tache)
        ->setAction($this->generateUrl('rt4_add_form'))
        ->setMethod('POST')
        ->add('text', 'text')
        ->add('date', 'date')
        ->getForm();
    $form->handleRequest($request);
    $em = $this->getDoctrine()->getManager();
    $em->persist($tache);
    $em->flush();
    return $this->render('Rt4AsBundle:Default:successTache.html.twig');
}
```

13

Externalisation de la définition des formulaires (3)

```
/**
 * Param OptionsResolverInterface $resolver
 */
public function setDefaultOptions(OptionsResolverInterface $resolver)
{
    $resolver->setDefaults(array(
        'data_class' => 'Rt4AsBundle\Entity\Tache'
    ));
}

/**
 * Returns string
 */
public function getLabel()
{
    return 'rt4_asbundle_tache';
}
```

16

Externalisation de la définition des formulaires (1)

Afin de rendre les formulaires réutilisables, Symfony permet l'externalisation des formulaires en des objets.

- **Convention de nommage :** L'objet du formulaire doit être nommé comme suit `NomObjetType`
 - Cet objet doit obligatoirement étendre la classe `AbstractType`
 - Deux méthodes doivent obligatoirement être implémentées :
- `buildForm(FormBuilderInterface $builder, array $options)` qui est la méthode qui va permettre la création et la définition du formulaire
- `getName()` qui retourne un identifiant unique pour le formulaire. Par convention c'est le nom du bundle et le nom de l'entité séparé avec des '_' et sans majuscule
- Il y a aussi la méthode `setDefaultOptions` qui est facultative et qui permet de définir l'objet associé au formulaire.

14

Externalisation de la définition des formulaires (4)

Ne pouvons pas accéder dans la classe `AbstractType` à la méthode `generateUrl` afin de modifier l'action du formulaire, il faut donc procéder ainsi :

- Générer l'url au niveau du formulaire
- Passer l'url dans le troisième paramètre optionnel de la méthode `createForm`
- Récupérer l'url dans l'objet du formulaire dans le tableau `$options` via son `label`

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->setAction($options['action'])
        ->setMethod('POST')
        ->add('tache')
        ->add('date')
    ;
}

$form = $this->createForm(new TacheType(), $tache, array(
    'action' => $this->generateUrl('rt4_add_form')
));
;
```

17

Externalisation de la définition des formulaires (2)

Doctrine permet d'automatiquement générer la classe du formulaire spécifique à une entité en utilisant la commande suivante :

`php app/console doctrine:generate:form BundleName:EntityName`

Exemple :

`php app/console doctrine:generate:form Rt4AsBundle:Tache`

La récupération du formulaire au niveau des contrôleur devient beaucoup plus facile :

```
$form = $this->createForm(new TacheType(), $tache);
```

15

Les propriétés d'un champ dans le formulaire

Le troisième paramètre de la méthode `add` est un tableau d'options pour les attributs du formulaire
Parmi les options communes à la majorité des champs nous citons :

- `label` : pour le label du champ si cette option n'est pas mentionné alors le label sera le nom du champ
- `required` : Permet de dire si le champ est obligatoire ou non (Par défaut l'option required est défini à true)

18

Les principaux types dans le formulaire (1)

Les formulaires sont composés d'un ensemble de champs

Chaque champ possède un nom, un type et des options

Symfony propose une grande panoplie de types de champ

Texte	Choix	Date et temps	Divers	Multiple	Caché
text	choice	date	checkbox	collection	hidden
textarea	entity	datetime	file	repeated	csrf
email	country	time	radio		
integer	language	birthday			
money	locale				
number	timezone				
password					
percent					
search					
url					

19

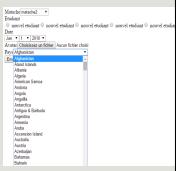
Les principaux types dans le formulaire (4) Le type country

Affiche la liste des pays du monde

La langue d'affichage est celle de la locale de votre application (config.yml)

Exemple

```
->add('pays','country')
```



<http://symfony.com/fr/doc/current/reference/forms/types/country.html>

22

Les principaux types dans le formulaire (2) Le type choice

Type spécifique aux champs optionnels (select, boutons radio, checkboxes)

Pour spécifier le **type d'options** qu'on veut avoir il faut utiliser le paramètre `expanded`. S'il est à `false` (valeur par défaut) alors nous aurons **une liste déroulante**. S'il est à `true` alors nous aurons des **boutons radio ou des checkbox** qui dépendra du paramètre `multiple`

Exemple :

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('nom', 'choice', [
            'expanded' => true,
            'multiple' => false,
            'choices' => [
                'matche1' => 'matche1',
                'matche2' => 'matche2',
                'matche3' => 'matche3'
            ]
        ])
        ->add('date', 'date', [
            'expanded' => false,
            'multiple' => false
        ]);
}
```

Expanded=true

Expanded=false

<http://symfony.com/fr/doc/current/reference/forms/types/choice.html>

20

Les principaux types dans le formulaire (4) Le type file

Le type `file` permet l'upload de n'importe quel type de fichier

Le champ permet de récupérer un **objet** de type `file` contenant le **path** de l'objet à uploader

Pour pouvoir gérer cet objet il faut la copier dans le **répertoire web de votre projet** et de préférence dans un dossier `upload`

Attribuer un nom unique à votre fichier pour ne pas avoir de problème lors de l'ajout de fichier ayant le même nom (vous pouvez utiliser la méthode suivante `md5($uniqueId)`)

Pour déplacer votre fichier utiliser la méthode `move($pathsrc,$pathdest)` de votre objet `file`

`__DIR__` vous donne le **path de l'endroit où vous l'utilisez**

<http://symfony.com/fr/doc/current/reference/forms/types/file.html>

23

Les principaux types dans le formulaire (3) Le type Entity

Champ choice spécial

Les choices (les options) seront chargés à partir des éléments d'une entité Doctrine

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('nom')
        ->add('age')
        ->add('enseignant', 'entity', [
            'class' => FormationFormationBundle\Entity\Enseignant,
            'property' => 'nom',
        ])
}
```

Balise HTML	expanded	multiple
Liste déroulante	false	false
Liste déroulante (avec attribut <code>multiple</code>)	false	true
Boutons radio	true	false
Cases à cocher	true	true

<http://symfony.com/fr/doc/current/reference/forms/types/entity.html>

21

Les validateurs (1)

Le validateur est conçu pour valider les objets selon des *contraintes*.

Le validateur de symfony est utilisé pour attribuer des contraintes sur les formulaires

La validation peut être faite de plusieurs façons :

YAML ([dans le fichier validation.yml dans le dossier /Resources/config du Bundle en question](#))

Annotations

XML

PHP

La méthode `isValid()` du FORM déclenche le processus de validation

<http://symfony.com/fr/doc/current/reference/constraints.html>

24

Les validateurs

Les annotations (1)

Afin de pouvoir utiliser les annotations de validation il faut importer la class Constraints

```
use Symfony\Component\Validator\Constraints as Assert;

Syntaxe :
@Assert\MaContrainte(option1="valeur1", option2="valeur2", ...)

Exemples :
@Assert\NotBlank( message = "Ce champ ne doit pas être vide ")
@Assert\Length(min=4, message="Le login doit contenir au moins {{ limit }} caractères.")
@Assert\Url()

Afin de tester vos validateurs, désactiver la validation HTML.
{{ form(form, {attr : {'novalidate' : 'novalidate'}}) }} ou en ajoutant novalidate à <form>
```

25

Les validateurs

Les annotations (File)

Contrainte	Rôle	Options
File	La contrainte File vérifie que la valeur est un fichier valide, c'est-à-dire soit une chaîne de caractères qui pointe vers un fichier existant, soit une instance de la classe File (ce qui inclut Uploadedfile).	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. mimeType : mimeType(s) que le fichier doit avoir.
Image	La contrainte Image vérifie que la valeur est valide selon la contrainte précédente File (dont elle hérite les options), sauf que les mimeType acceptés sont automatiquement définis comme ceux de fichiers images. Il est également possible de mettre des contraintes sur la hauteur max ou la largeur max de l'image.	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. minWidth /maxWidth : la largeur minimale et maximale que doit respecter l'image. minHeight /maxHeight : la hauteur minimale et maximale que doit respecter l'image.

26

Les validateurs

Les annotations (Les contraintes de base)

Contrainte	Rôle	Options
NotBlank Blank	La contrainte NotBlank vérifie que la valeur soumise n'est ni une chaîne de caractères vide, ni NULL. La contrainte Blank fait l'inverse.	-
True False	La contrainte True vérifie que la valeur vaut true, 1 ou "1". La contrainte False vérifie que la valeur vaut false, 0 ou "0".	-
NotNull Null	La contrainte NotNull vérifie que la valeur est strictement différente de null.	-
Type	La contrainte Type vérifie que la valeur est bien du type donné en argument.	type (option par défaut) : le type duquel doit être la valeur, parmi array, bool, int, object

<https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/validez-vos donnees>

26

aymen.sellaouti@gmail.com

Les validateurs

Les annotations (Nombre, date)

Contrainte	Rôle	Options
Range	La contrainte Range vérifie que la valeur ne dépasse pas X, ou qu'elle dépasse Y.	min : nbre de car minimum max : nbre de car maximum minMessage : msg erreur nbre de car min maxMessage : msg erreur nbre de car max invalidMessage : msg erreur si non nombre
Date	vérifie que la valeur est un objet de type Datetime, ou une chaîne de type YYYY-MM-DD.	-
Time	vérifie que c'est un objet de type Datetime, ou une chaîne type HH:MM:SS.	-
DateTime	vérifie que c'est un objet de type Datetime, ou une chaîne de caractères du type YYYY-MM-DD HH:MM:SS.	-

27

Atelier HTML5

Mise en Forme sans CSS

Les attributs de la balise body :

bgcolor="couleur" : Couleur qui sera utilisé pour l'arrière-plan.

background="fichierimage" : Image d'arrière-plan (BMP, JPG ou GIF).

bgproperties = " fixed" fixe l'arrière plan de votre document.

link="couleur" : Couleur des liens.

alink="couleur" : Couleur des liens déjà exploités.

vlink="couleur" : Couleur des liens lors du clic.

- Les couleurs peuvent être écrites de deux manières :

- En hexadécimal de type RVB (Rouge, Vert, Bleu) et précédées d'un dièse (#),

Exemples:

#ff0000 => Rouge, #00ff00 => Vert, #0000ff => Bleu,

#ffffff => Blanc, #8000ff => Violet, #ffff00 => Jaune,

#000000 => Noir.

- En textuelles en anglais,

Exemples : red, yellow, pink.

La mise en forme du texte:

- Le texte en gras: Pour mettre le texte en gras on l'encadre de la balise ****.

Exemple:Ce texte s'affichera en gras.

- Le texte en italique: Pour mettre le texte en italique on l'encadre de la balise **<i>**.

Exemple:<i>Ce texte s'affichera en italique.</i>

- Le texte souligné: Pour souligner le texte on l'encadre de la balise **<u>**.

Exemple: <u>Ce texte sera souligné.</u>

- Le texte **barré**: Pour barrer le texte on l'encadre de la balise **<s>**.

Exemple:<s>Ce texte sera barré.</s>

- Le texte en **exposant**: Pour placer le texte en exposant on l'encadre de la balise **<sup>**.

Exemple:^{Ce texte sera en exposant.}

- Le texte en **indice**: Pour placer le texte en indice on l'encadre de la balise **<sub>**.

Exemple: _{Ce texte sera en indice.}

- Modifier la **couleur du texte**: Pour modifier la couleur du texte on utilise l'**attribut color** de la balise ****.

Exemple:

Ce texte sera en rouge.

- Modifier la police du texte: Pour modifier la police du texte on utilise l'**attribut face** de la balise ****.

Exemple:

Ce texte sera en verdana.

Remarque: On a tendance à écrire plusieurs police plutôt qu'une seule.

La police utilisé peut ne peut existé.

On sépare alors les différentes polices par une virgule dans l'ordre de sélection le cas où la précédente ne peut pas être affichée.

Exemple: Ce texte sera en verdana ou en sans-serif si verdana n'est pas installée.

- Modification de la taille du texte:

Pour modifier la **taille du texte** on utilise l'**attribut size** de la balise ****.

Exemple: Ce texte sera en taille 5.

Par défaut, la valeur de l'attribut **size vaut "3"**.

La valeur de cet attribut et les valeurs possibles sont les entiers **de "1" à "7"**.

Remarque: Il est bien évidemment possible de renseigner ces trois attributs (color, face, size) à la fois dans la même balise ****.

l'attribut « **align** » pour spécifier la position du paragraphe dans la page. Il peut prendre les valeurs left, right, center.

Les caractères spéciaux:

- < **Signe inférieur** : < ;
 - ▶ 37 est < à 42 ▶ 37 est < ; à 42
- > **Signe supérieur** : > ;
 - ▶ 42 est > à 37 ▶ 42 est > ; à 37
- & **Et commercial** : & ;
 - ▶ John Wily & Co. ▶ John Wily & ; Co.
- " **Double cotes** : " ;
 - ▶ “La vie est un recommencement éternel”
 - ▶ " ;La vie est un recommencement éternel" ;
- **Espace** : ;
 - ▶ Il dit : “N’ayez pas peur ! Je vais tout régler.”
 - ▶ Il dit ; : ;" ;N’ayez pas
peur ; ! ;Je vais tout régler." ;

Remplacer # par une lettre

- **Accent grave** : &grave ;
- **Accent aigu** : ´ ;
- **Accent circonflexe** : &circ ;
- **Tréma** : ¨ ;
- **Cédille** : ¸ ;

Nécessaires si la balise métâ http-equiv n'est pas renseignée !

- **Exemple** : C'était les accents français en HTML
 - ▶ C'´ ;tait les accents franç ;ais en HTML

Exercice 1 :

1. Créer une page HTML nommée « page1.htm », qui contient le texte suivant.

Comment effectuer une recherche ?

Pour vos recherches documentaires, vous allez être amenés à chercher et utiliser plusieurs sortes de **documents** :(ouvrages, revues, usuels, manuels, CD-ROM, DVD, documents en ligne, etc.), disponibles sur différents **supports** (papier, numérique...) pour y trouver l'information dont vous avez besoin.

2. Ajouter le code HTML qui permet d'afficher le titre « Introduction » dans la barre du navigateur.
3. Ajouter une balise méta qui permet de dire qui a réalisé la page.
4. Ajouter une balise méta qui permet un meilleur référencement des pages.
5. Créer une deuxième page html nommée « page2.htm », qui contient le texte suivant.

Comment mieux référencer mon site ?

Pour améliorer la qualité du référencement de nos pages nous avons utilisés la balise meta qui pour l'attribut name prend la valeur « keywords » et pour l'attribut content des mots que nous avons choisis et qui reflètent au mieux le contenu de la page.

6. Utiliser la meta nécessaire qui permet de recharger la page chaque 5s pour chacune des deux pages.
7. Modifier l'un des attributs de cette balise afin de permettre à chaque page de passer à l'autre après 10s.
8. Regroupez les deux paragraphes dans un seul fichier
9. Aligner le 1^{er} paragraphe à gauche et le 2^{ème} à droite.
10. Ajouter les balises nécessaires pour mettre en gras les mots correspondants.
11. Ajouter des commentaires avant chaque paragraphe.
12. Ajouter un 3^{ème} paragraphe aligné au centre qui contient le texte suivant :

Le **support** désigne la nature matérielle du document. On parle de supports papier, supports numériques..., pour parler des documents imprimés, numériques. Le **support**, c'est la base, la matière du document.

Partie 2

- Mettre en forme l'article précédent tout en répondant aux questions suivantes :
1. Mettre le titre du document en police **Courier** et en **vert**.
 2. Mettre les mots « recherches documentaires » en **italique** et en **bleu**.
 3. Mettre le mot « CD-ROM » en **italique** et en **rouge**, et lui appliquer **la taille 5**.
 4. Ajouter un retour à la ligne après la fermeture de la parenthèse qui contient le mot CD-ROM.
 5. Placer une barre horizontale en fin de l'article.
 6. Ajouter à sa fin, après la barre, la date « 12/12/2012 », dans un paragraphe aligné à droite.
 7. Ajouter les balises meta nécessaires à votre document

Les liens hypertextes

Les liens permettent de construire un hypertexte et peuvent être de différents types:

- **externes**: un pointeur du document mène vers un autre document
- **internes**: un pointeur renvoie à une section du même document

Un **lien** se définit par la balise `<a>` suivi du paramètre **Href="URL"** qui définit **l'adresse du document** vers lequel le lien conduit.

Les liens internes permettent de construire des tables de matières et des renvois à l'intérieur d'un texte. Un lien interne pointe vers une ancre, c'est à dire un endroit à l'intérieur d'un document défini par un nom. Il faut définir deux choses pour un lien interne.

- L'ancre interne

``

Exemple : ``

- Le lien vers l'ancre : Le lien proprement dit se définit avec la balise `<a>` suivie de :

`mots_sensibles`

Exemple : `Aller en bas`

Il est également possible d'utiliser les ancrés dans les liens externes. Il faut alors spécifier l'ancré vers laquelle pointe le lien en ajoutant #nom à la fin de l'URL.

Exemple : Ancres et liens externes

```
<a href = "http://tecfa.unige.ch/cours/exemple.html#partie2">Deuxième Partie</a>
```

Et dans le document exemple.html on trouve la définition de l'ancré suivante:

```
<a name="partie2"> </a>
```

Une deuxième façon d'utiliser les liens interne est l'utilisation de l'attribut **id** de la façon suivante :

- Définition dans une page : attribut id

Exemple : <h1 id="menu">Menu</h1>

- Référence depuis la même page

Exemple : Aller au menu

- Référence depuis une autre page

Exemple :

```
<a href="page.html#menu">Retour au menu</a>
```

Exercice 2 :

1. Créez la page index.html ayant l'allure suivante :

Présentation sera un lien vers la page présentation.html

Services sera un lien vers la page service.html

Contact sera un lien vers votre mail

index - Microsoft Internet Explorer

Fichier Edition Affichage Favoris Outils ?

Précédente Suivante Arrêter Actualiser Démarrage

Adresse C:\siteentreprise\index.htm

[Présentation](#)
[services](#)
[Contact](#)

2. La page présentation aura l'allure ci-dessous :

L'image sera un lien vers la page siège.html

3. La page Service aura l'allure ci-dessous :

Service 1 et Service 2 seront respectivement des liens internes vers le premier et le deuxième service et retour seront des liens vers le haut de la page

4. ajoutez dans chaque page des liens vers les autres pages

5. Ajoutez dans la page présentation le contenu suivant

Nous offrons les services suivants :

Service1 et Service 2 qui seront respectivement des liens vers le premier et le deuxième service de la page service.html

Présentation - Microsoft Internet Explorer

Fichier Edition Affichage Favoris Outils ?

Précédente Suivante Arrêter Actualiser Démarrage

Adresse C:\siteentreprise\présentation.htm

Créé en 1990, notre entreprise possède 3 unités:

- Une unité de filature
- Une unité de teinture
- Une unité de traitement sur pièces

Vous pouvez visiter notre siège



[accueil](#)

[Services](#)

[Contact](#)

services - Microsoft Internet Explorer

Fichier Edition Affichage Favoris Outils ?

Précédente Suivante Arrêter Actualiser Démarrage

Adresse C:\siteentreprise\services.htm

[Service 1](#)

[Service 2](#)

Notre entreprise offre le service de filature

[retour](#)

Notre entreprise offre le service de confection

[retour](#)

Exercice 3 :

Reproduire les pages suivantes :

Voici ma page Perso							
Mon Avatar	 <i>Mon Avatar</i>						
Nom	Votre nom						
Prenom	Votre prenom						
Age	Votre Age						
Diplomes	<table><tr><td>Bac</td><td>2011</td></tr><tr><td>Licence</td><td>2014</td></tr></table>			Bac	2011	Licence	2014
Bac	2011						
Licence	2014						
Hobbies	<ul style="list-style-type: none">I. hobbie 1<ul style="list-style-type: none">a. desc 1b. desc 2II. hobbie 2III. hobbie 3						

2)

Population	Moyenne	
	Hauteur en cm	Espérance de vie
Sexe	Masculin	1,78
	Féminin	1,70

Ajouter une colonne pour indiquer le poids moyen : par exemple pour Masculin 75 kg et Féminin 55 kg

3)

Inscription en licence

Informations personnelles	
Votre nom	votre nom?
Votre prénom	votre Prénom?
Votre age	votre age?
Votre adresse?	
Votre adresse	
Mot de passe	
Retapez votre mot de passe	
Gestion des groupes et des options	
Votre groupe : <input type="radio"/> Groupe 1 <input type="radio"/> Groupe 2 <input type="radio"/> Groupe 3	
Numéro de carte d'étudiant	Numéro de carte d'étudiant?
<input type="checkbox"/> J2ee <input type="checkbox"/> Android <input type="checkbox"/> .Net <input type="checkbox"/> HTML5 <input type="checkbox"/> Flex	
Choisissez les options que vous désirez	
<input type="button" value="valider"/> <input type="button" value="réinitialiser"/>	

Reprendre le formulaire précédent en y incluant les contraintes suivantes :

L'age devra être obligatoirement un entier.

Les champs de mot de passe doivent être obligatoires

Ajouter deux champs permettant d'ajouter l'@mail ainsi que l'URL du site de l'étudiant s'il y en a.

Ajouter un champs représentant la date de naissance de l'étudiant.

Afin d'éviter les scripts pirates, ajouter un zone texte qui n'accepte que les mots suivant l'expression régulière suivante : [I][N][S][A][T][0-9]{4}

4)

En cas de validation du formulaire, l'internaute sera redirigé vers la page suivante :



INSAT

Hymne de l'école

Vidéo présentatrice de l'école

Reproduire cette page en essayant de démarrer le fichier audio à l'ouverture de la page.

Les Frames

- Les cadres servent à séparer la fenêtre en différents morceaux indépendants qui peuvent chacun contenir une page HTML différente, ces morceaux ayant la capacité de communiquer entre eux.
- On utilise au minimum deux cadres dans une fenêtre, en général un cadre pour le menu à gauche et un autre pour le contenu du site. Parfois on rajoute un cadre en haut pour un bandeau fixe.
- Deux balises sont utilisées : FRAMESET et FRAME

<FRAMESET> fractionnement de la fenêtre active.

<FRAMESET ROWS ="30%, 70%"> partage horizontal.

<FRAMESET COLS ="30%, 70%"> partage vertical.

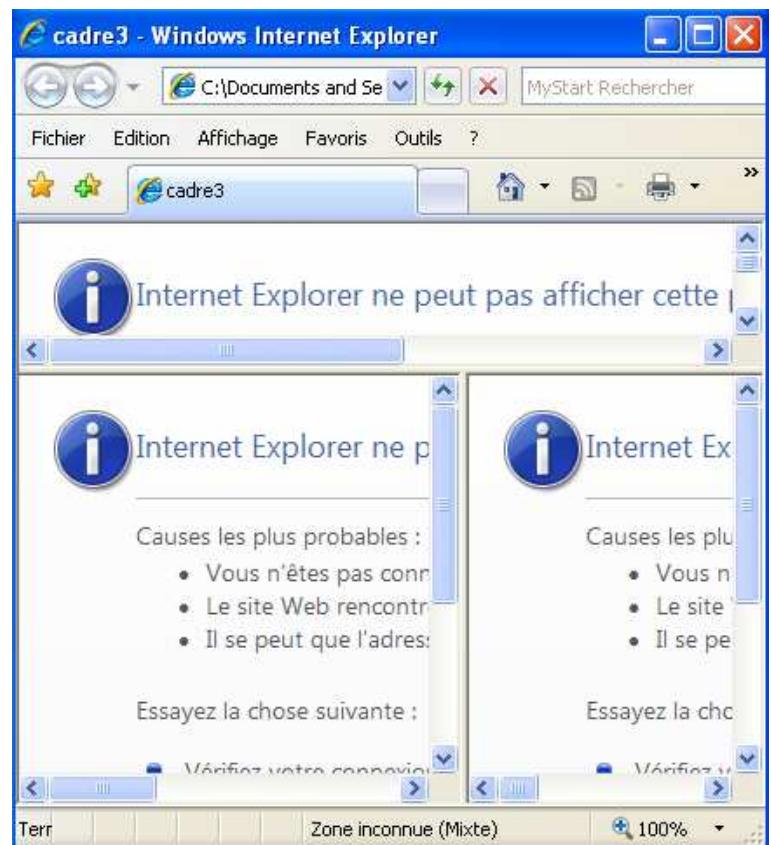
<FRAME> contenu du cadre.

<FRAME SRC="un.htm"> le fichier un.htm sera chargé dans ce cadre.

- Remarque : la balise <FRAMESET> remplace la balise <BODY>.

Exemple 1 :

```
<HTML>
  <HEAD>
    <TITLE>cadre3</TITLE>
  </HEAD>
  <FRAMESET ROWS="25%,*">
    <FRAME SRC="un.htm">
  <FRAMESET COLS="60%,*">
    <FRAME SRC="deux.htm">
    <FRAME SRC="trois.htm">
  </FRAMESET>
</FRAMESET>
</HTML>
```



L'attribut frameborder :

Cet attribut permet de déterminer si les cadres auront ou n'auront pas de bordure. Il a deux valeurs yes ou no.

frameborder=yes ou **frameborder=no**

L'attribut border :

Cet attribut permet de déterminer la taille des bordures entourant les cadres. Il peut prendre plusieurs valeurs ex: **border=n**, n étant une valeur en pixels définissant la taille de la bordure. La valeur 0 indique aucune bordure.

border=0 ou par exemple **border=5**

L'attribut bordercolor

Cet attribut permet de déterminer la couleur de l'ensemble des bordures des cadres. Il suffit pour cela de spécifier une couleur sous forme de nom ou de sa valeur hexadécimale.

bordercolor="red" ou **bordercolor="#ff0000"**

L'attribut noresize

En utilisant cet attribut, vous interdisez à l'utilisateur de redimensionner les cadres. Par défaut les cadres peuvent être redimensionnés.

L'attribut scrolling

Cet attribut permet d'attribuer ou non une barre de défilement (scrollbar) à un cadre. Il possède trois valeurs :

- yes : Indique que la barre de défilement sera toujours visible
- no : Indique que la barre de défilement ne sera jamais visible (à tester avant de l'utiliser)
- auto : Indique que le navigateur déterminera si la barre de défilement est nécessaire

Exemple: <frame src="tableau.html" scrolling=**auto** >

Les attributs marginwidth et marginheight :

* **marginwidth** permet de spécifier la grandeur des marges de gauche et de droite du cadre créé, la valeur doit être exprimée en pixels, elle peut avoir comme valeur 0.

* **marginheight** permet de spécifier la grandeur des marges de haut et de bas du cadre créé, la valeur doit être exprimée en pixels, elle peut avoir comme valeur 0.

Liens vers d'autres Frames

Lorsqu'on active un lien se trouvant dans une Frame, la page appelée par ce dernier s'affichent dans la Frame en question. L'attribut **target** permet d'appeler un autre cadre par le nom défini avec l'attribut **name** précédemment.

Exemple :

Principale utilisation : Les sommaires

L'attribut target peut prendre les valeurs suivantes :

- _blank qui indique au browser qu'il doit créer une nouvelle fenêtre afin d'y afficher le fichier. Dans ce cas, vous ouvrez en fait un nouveau browser.
- _self qui indique que le fichier sera chargé dans la même fenêtre que celle dans laquelle se trouve le lien.
- _top qui implique l'affichage du fichier sur toute la surface de la fenêtre du browser.

Exercice :

On veut créer un site pour le cours du langage html qui suit la description suivante :

La première page contient trois cadres horizontales

1. Le premier cadre contient le titre de la page
2. Le deuxième cadre contient les chapitres : Introduction, Tableau, Image, Lien
3. Le troisième cadre contient une page vide

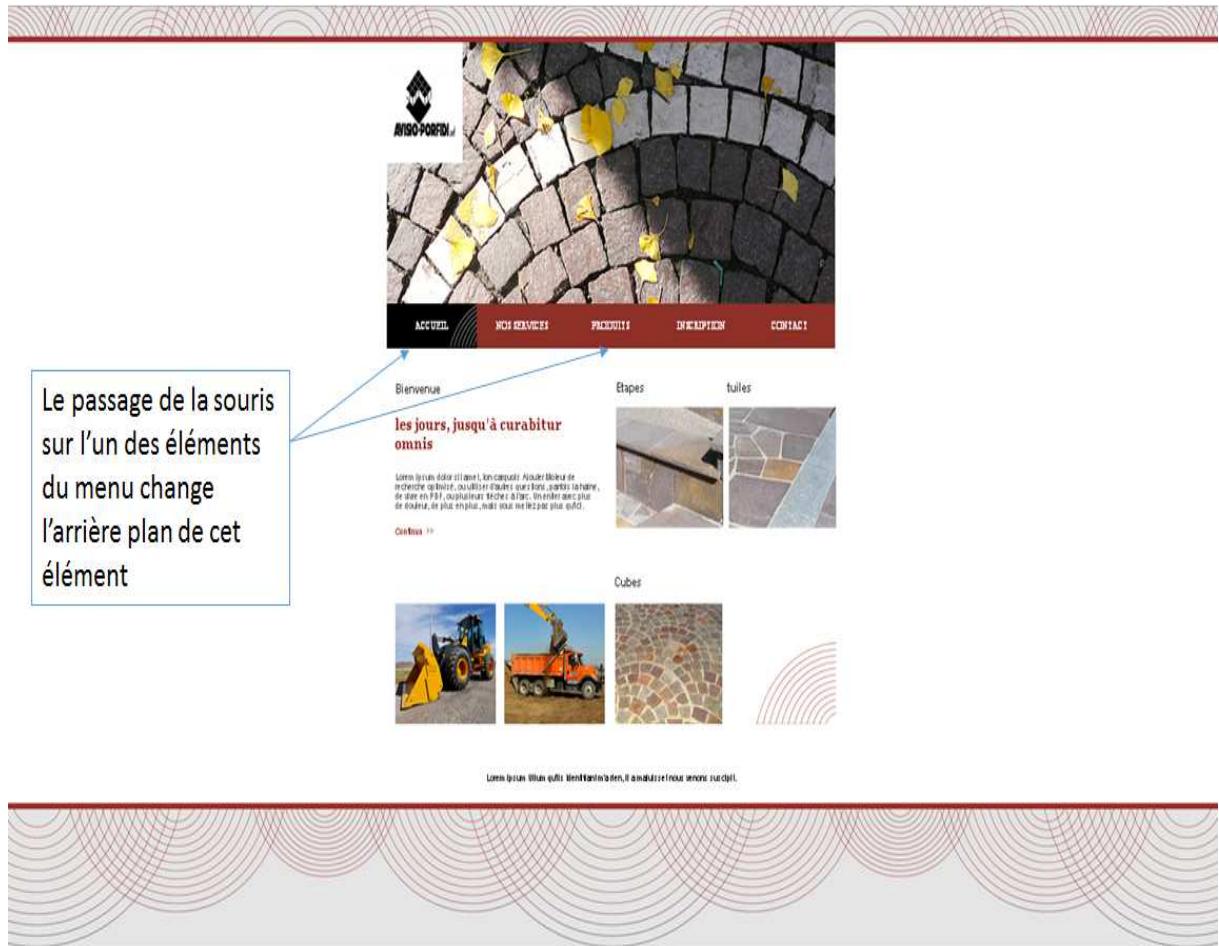
Si on veut visualiser le contenu d'un chapitre :

- On clique sur un chapitre (lien qui se trouve dans le cadre 2)
 - Le contenu s'affiche dans le dernier cadre
1. Ecrire le code correspondant
 2. Le chapitre lien contient deux parties : lien interne et lien externe, chaque partie sera développée dans une page à part.

Atelier CSS

Le but de cet atelier est de s'acclimater au feuilles de styles CSS.

Nous commençons par un premier exercice qui permettra d'utiliser le HTML5 et les feuilles de style CSS afin de reproduire la page suivante :



Le CSS représente un langage qui permet la mise en page d'une page web. Les feuilles de styles CSS se présentent comme une succession de définition des électeurs avec leur mise en page.

Les sélecteurs sont des caractères alphanumériques qui identifient la règle.

Chaque sélecteur est présenté comme suit :

```

Selecteur {  

    attribut1 : valeur1;  

    attribut2 : valeur2 ;  

}

```

Exemple :

```

H1 {  

    color: blue;  

    text-align: right ;  

}

```

Nous présentons ici un récapitulatif des différentes fonctionnalités offertes par le CSS.

A. Les propriétés des caractères

Propriété	Valeurs (exemples)	Description
font-family	serif, sans-serif, times	Nom de police
@font-face	<i>Nom et source de la police</i>	Police personnalisée
font-size	1.3em, 16px, 120%...	Taille du texte
font-weight	bold, normal	Gras
font-style	italic, oblique, normal	Italique
text-decoration	underline, overline, line-through, blink, none	Soulignement, ligne au-dessus, barré ou clignotant
font-variant	small-caps, normal	Petites capitales
text-transform	capitalize, lowercase, uppercase	Capitales
Font	-	Super propriété de police. Combine : font-weight, font-style, font-size, font-variant, font-family.
text-align	left, center, right, justify	Alignement horizontal
vertical-align	baseline, middle, sub, super, top, bottom	Alignement vertical (cellules de tableau ou éléments inline-block uniquement)
line-height	18px, 120%, normal...	Hauteur de ligne
text-indent	25px	Alinéa
white-space	pre, nowrap, normal	Césure
word-wrap	break-word, normal	Césure forcée
text-shadow	5px 5px 2px blue <i>(horizontale, verticale, fondu, couleur)</i>	Ombre de texte

B. Propriétés de couleur et de fond

Propriété	Valeurs (exemples)	Description
color	<i>nom</i> , <code>rgb(rouge,vert,bleu)</code> , <code>rgba(rouge,vert,bleu,transparence)</code> , <code>#CF1A20...</code>	Couleur du texte
background-color	<i>Identique à color</i>	Couleur de fond
background-image	<code>url('image.png')</code>	Image de fond
background-attachment	<code>fixed</code> , <code>scroll</code>	Fond fixe
background-repeat	<code>repeat-x</code> , <code>repeat-y</code> , <code>no-repeat</code> , <code>repeat</code>	Répétition du fond
background-position	<i>(x y)</i> , <code>top</code> , <code>center</code> , <code>bottom</code> , <code>left</code> , <code>right</code>	Position du fond
Background	-	Super propriété du fond. Combine : <code>background-image</code> , <code>background-repeat</code> , <code>background-attachment</code> , <code>background-position</code>
opacity	0.5	Transparence

C. Propriétés des boîtes

Propriété	Valeurs (exemples)	Description
width	150px, 80%...	Largeur
height	150px, 80%...	Hauteur
min-width	150px, 80%...	Largeur minimale
max-width	150px, 80%...	Largeur maximale
min-height	150px, 80%...	Hauteur minimale
max-height	150px, 80%...	Hauteur maximale
margin-top	23px	Marge en haut
margin-left	23px	Marge à gauche
margin-right	23px	Marge à droite
margin-bottom	23px	Marge en bas
margin	23px 5px 23px 5px <i>(haut, droite, bas, gauche)</i>	Super-propriété de marge. Combine : <code>margin-top</code> , <code>margin-right</code> , <code>margin-bottom</code> , <code>margin-left</code> .
padding-left	23px	Marge intérieure à gauche
padding-right	23px	Marge intérieure à droite
padding-bottom	23px	Marge intérieure en bas
padding-top	23px	Marge intérieure en haut
padding	23px 5px 23px 5px <i>(haut, droite, bas, gauche)</i>	Super-propriété de marge intérieure. Combine : <code>padding-top</code> , <code>padding-right</code> , <code>padding-bottom</code> , <code>padding-left</code> .
border-width	3px	Épaisseur de bordure
border-color	<i>nom</i> , <code>rgb(rouge,vert,bleu)</code> ,	Couleur de bordure

	rgba(rouge,vert,bleu,transparence), #CF1A20...	
border-style	solid, dotted, dashed, double, groove, ridge, inset, outset	Type de bordure
Border	3px solid black	Super-propriété de bordure. Combine border-width, border-color, border-style.
border-radius	5px	Bordure arrondie
box-shadow	6px 6px 0px black (horizontale, verticale, fondu, couleur)	Ombre de boîte

D. Propriétés de positionnement et d'affichage

Propriété	Valeurs (exemples)	Description
display	block, inline, inline-block, table, table-cell, none...	Type d'élément (block, inline, inline-block, none...) inline Permet de mettre les éléments verticalement
visibility	visible, hidden	Visibilité
clip	rect (0px, 60px, 30px, 0px) rect (haut, droite, bas, gauche)	Affichage d'une partie de l'élément
overflow	auto, scroll, visible, hidden	Comportement en cas de dépassement
float	left, right, none	Flottant
clear	left, right, both, none	Arrêt d'un flottant
position	relative, absolute, static	Positionnement
top	20px	Position par rapport au haut
bottom	20px	Position par rapport au bas
left	20px	Position par rapport à la gauche
right	20px	Position par rapport à la droite
z-index	10	Ordre d'affichage en cas de superposition. La plus grande valeur est affichée par-dessus les autres.

E. Propriétés des listes

Propriété	Valeurs (exemples)	Description
list-style-type	disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none	Type de liste
list-style-position	inside, outside	Position en retrait
list-style-image	url('puce.png')	Puce personnalisée
list-style	-	Super-propriété de liste. Combine list-style-type, list-style/-position, list-style-image.

F. Propriétés des tableaux

Propriété	Valeurs (exemples)	Description
border-collapse	collapse, separate	Fusion des bordures
empty-cells	hide, show	Affichage des cellules vides
caption-side	bottom, top	Position du titre du tableau

G. Autres propriétés

Propriété	Valeurs (exemple)	Description
cursor	crosshair, default, help, move, pointer, progress, text, wait, e-resize, ne-resize, auto...	Curseur de souris

Menu avec les apports de CSS3

Le but de cet exemple est de construire étape par étape un menu en utilisant les propriétés du CSS3.

Tout d'abord il faut préciser qu'il existe des balises non encore reconnu directement par les navigateurs d'où le besoin d'ajouter des préfixes permettant cette identification :

- ✓ -o- pour Opera
- ✓ -moz- pour Gecko (Mozilla)
- ✓ -webkit- pour Webkit (Chrome, Safari, Android...)
- ✓ -ms- pour Microsoft (Internet Explorer)
- ✓ -khtml- pour KHTML (Konqueror)

Le menu que nous voulons reconstruire est le suivant :



Commençons par écrire le code HTML permettant d'obtenir une liste.

```
<nav>  
  
<ul>  
  <li><a href="#1">Presentation</a></li>  
  <li><a href="#2">HTML5</a></li>  
  <li><a href="#3">CSS3</a></li>  
  <li><a href="#4">PHP5</a></li>
```

```
<li><a href="#5">Projet</a></li>  
</ul>  
</nav>
```

Afin de supprimer la décoration des li et de rendre la liste sous forme d'un tableau on utilise le code CSS suivant :

```
ul {  
    display: table; /* Permet de traiter la liste comme un tableau */  
    width: 500px;  
    margin: 50px auto;  
    padding: 0;  
    font: 1.1em sans-serif;  
}  
  
ul li {  
    display: table-cell; /* afficher les li comme des éléments d'un tableau */  
    background: dodgerblue;  
}
```

Nous obtenons alors le résultat suivant :

[Presentation](#) [HTML5](#) [CSS3](#) [PHP5](#) [Projet](#)

Nous passons ensuite à l'arrondissement des coins en utilisant l'attribut suivant dans l'**ul** :

border-radius: 10px;
qui arrondira les 4 coins.

On passe ensuite aux ombres.

text-shadow: 2px 2px 5px black;

Ajoute une (ou plusieurs) ombre(s) sur le texte

`box-shadow: 2px 2px 5px (5px) black (inset);`

Ajoute une (ou plusieurs) ombre(s) sur une boite

L'ombre peut être interne: `inset`

L'ombre peut être plus grande ou plus petite (4ème valeur numérique).

Ajoutons le code suivant pour l'`ul`

`box-shadow: 0 1px 3px rgba(0,0,0,.3),`

`0 5px 10px rgba(0,0,0,.25),`

`0 20px 20px rgba(0,0,0,.15) ;`

Avec `rgba` représentant la notion de transparence dans les couleurs.

Ensute on modifie la couleur du fond en la transformant en une couleur transparente en utilisant l'attribut `linear-gradient`

Exemple : `linear-gradient((to left), red, green);`

La direction est optionnelle. La valeur par défaut est `to bottom`

Nombre de couleurs "illimité"

Par précaution on peut ajouter les préfixes comme suit :

`background: dodgerblue;`

`background: -webkit-linear-gradient(deepskyblue,dodgerblue);`

`background: -moz-linear-gradient(deepskyblue,dodgerblue);`

`background: -ms-linear-gradient(deepskyblue,dodgerblue);`

`background: -o-linear-gradient(deepskyblue,dodgerblue);`

`background: linear-gradient(deepskyblue,dodgerblue);`

On ajoutera ceci dans le `li`.

Passons maintenant au traitement des liens. Le but est de rendre tous le bloc cliquable, d'enlever le soulignement des liens et de décorer un peu tout ça. Nous procérons comme suit :

`ul li a{`

`display: block; /* Afin que tous le bloc devienne cliquable */`

`text-align: center;`

`text-decoration: none; /* Pour enlever le soulignement des liens*/`

`padding: 8px 8px 17px 8px;`

`color: black;`

```

color: rgba(0,0,0,.7); /* ajouter un effet transparent */
text-shadow: 0 1px 0 rgba(255,255,255,.4); /* ombrer le texte */
box-shadow: 0 1px 0 rgba(255,255,255,.7) inset, /* ombrer le box */
            0 -1px 0 hsl(210,100%,32%) inset,
            0 -2px 0 hsl(210,100%,38%) inset,
            0 -3px 0 hsl(210,100%,44%) inset,
            0 -4px 0 hsl(210,100%,50%) inset,
            0 -5px 0 hsl(210,100%,60%) inset;
}

```

Nous passons ensuite à la gestion des actions sur les liens.

```

ul li a:hover,ul li a:focus{
background: rgba(255,255,255,.2);padding:8px 25px 17px 25px;
box-shadow: 0 1px 0 rgba(255,255,255,.7) inset,
            0 -1px 0 hsl(210,100%,42%) inset,
            0 -2px 0 hsl(210,100%,48%) inset,
            0 -3px 0 hsl(210,100%,54%) inset,
            0 -4px 0 hsl(210,100%,60%) inset,
            0 -5px 0 hsl(210,100%,70%) inset;
}

```

}



```

ul li a:active{
background: rgba(0,0,0,.15);
background: -webkit-linear-gradient(rgba(0,0,0,.2),rgba(0,0,0,.1));
background: -moz-linear-gradient(rgba(0,0,0,.2),rgba(0,0,0,.1));
background: -ms-linear-gradient(rgba(0,0,0,.2),rgba(0,0,0,.1));
background: -o-linear-gradient(rgba(0,0,0,.2),rgba(0,0,0,.1));
background: linear-gradient(rgba(0,0,0,.2),rgba(0,0,0,.1));
box-shadow: 0 0 2px rgba(0,0,0,.3) inset;
}

```

Presentation

HTML5

CSS3

PHP5

Projet

Atelier Symfony2 : Introduction Bundle et Contrôleur

OBJECTIF :

Le but de cet atelier est de s'acclimater avec le Framework Symfony2.

Liens utiles :

- <http://symfony.com/>
- <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/>
- <http://www.doctrine-project.org/>

Livre

- Symfony, The Book, SensioLabs Version Aout2015.

Présentation de symfony2 :

Symfony2 est un Framework PHP basé sur l'architecture MVC2. C'est un cadre de travail (Framework) offrant une panoplie de composants permettant de modéliser les fondations et l'architecture d'un logiciel. Il a pour but d'uniformiser l'architecture des applications web PHP et d'améliorer la productivité des développeurs.

Les avantages de Symfony2 :

- ✓ Gain en productivité.
- ✓ Organisation du code : l'architecture de symfony2 doit être obligatoirement suivie par les développeurs ce qui fait que le code est bien organisé et facilement maintenable et évolutif.
- ✓ Briques logiciels déjà prêtées : (Génération de bundle, d'entités, de la base de données, des formulaires)
- ✓ Architecture MVC2
- ✓ Gestion automatique de la réécriture des URL (url rewriting)

- ✓ Grande communauté
- ✓ Documentation de qualité et régulièrement mise à jour ;

Installation de symfony2 :

Pré requis :

Serveur Web contenant PHP (Apache).

Après l'installation et pour vérifier la bonne mise en place des prérequis, utiliser la commande suivante :

php app/check.php

<http://symfony.com/fr/doc/current/reference/requirements.html> ;

Il existe plusieurs méthodes afin d'installer symfony2
(<http://symfony.com/fr/doc/current/book/installation.html>)

Méthode 1

Télécharger la version souhaitée (<http://symfony.com/download>)

Méthode 2

Télécharger et installer composer (<https://getcomposer.org/download/>)

Ouvrir la ligne de commande et placer vous sur le dossier d'installation

Tapez la commande suivante : **composer create-project symfony/framework-standard-edition path/ "2.7.*"** (Ici nous avons mentionné la dernière version lors de la réalisation de cet atelier en ne mentionnant aucune version vous aurez automatiquement la dernière version)

Méthode 3

Installer le symfony installer avec la commande suivante :

c:> php -r "readfile('http://symfony.com/installer');" > symfony

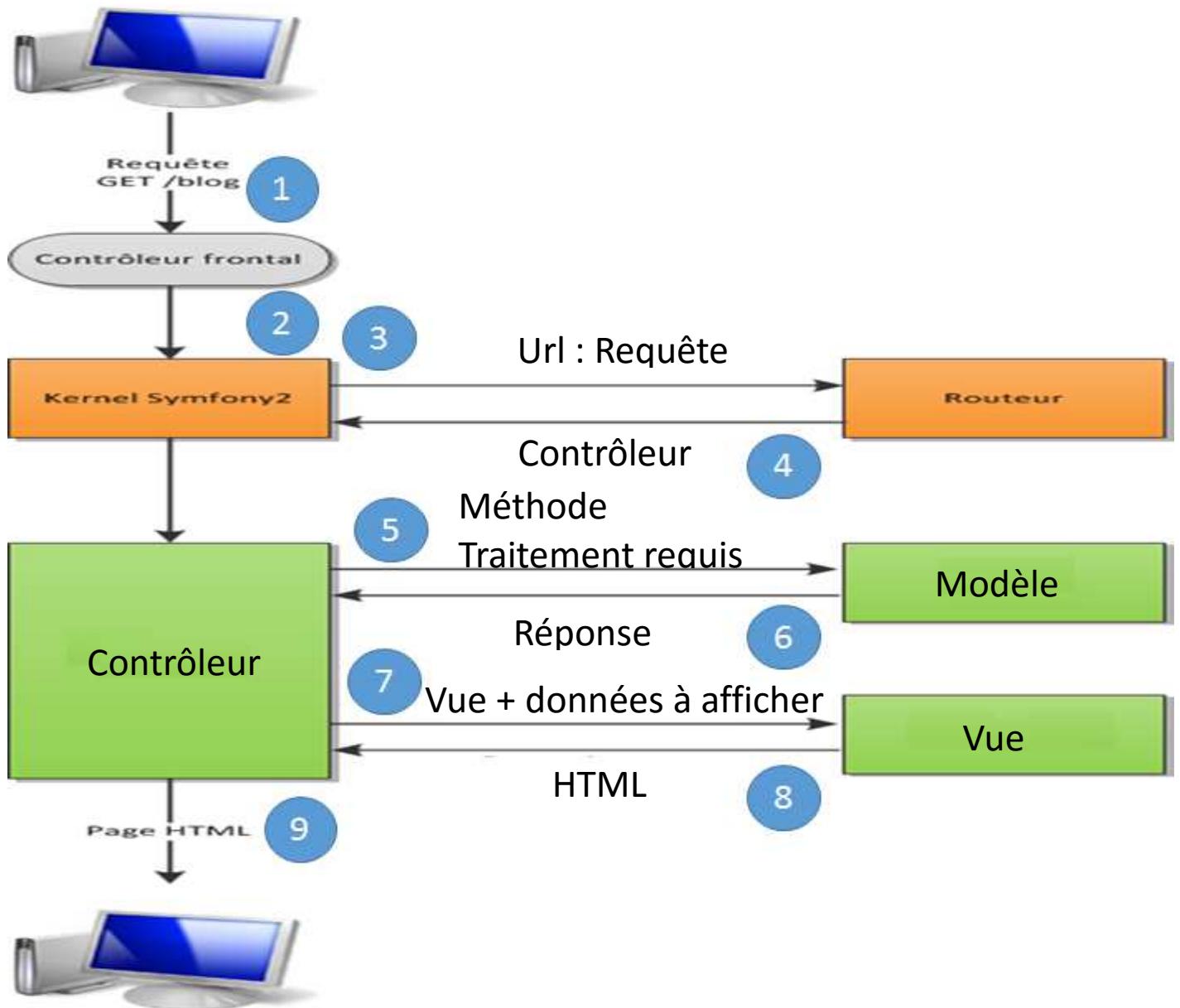
Copier le fichier dans le dossier père de votre dossier d'installation

Taper la commande suivante : **php symfony new nomProjet**

Notez bien : N'oubliez pas de vérifier la version PHP utilisée, une version inférieure à la version 5.3.2 ne sera pas fonctionnelle.

N'oubliez pas d'ajouter vos variables d'environnement. Voici les étapes nécessaires :

- Allez dans les paramètres système avancés du panneau de configuration



frontal app.php. Il est plus rapide et ne retourne aucune information concernant les erreurs.

Les deux contrôleurs frontaux offre une abstraction par rapport au noyau de symfony2 qui traite les requêtes des utilisateurs. Les contrôleurs ne font que transmettre la requête au noyau de symfony2. Le déroulement du traitement des requêtes utilisateur est illustré par la figure 1.

Figure 1 : Architecture MVC2 et traitement des requêtes dans Symfony2

Une fois l'acheminement des requêtes assimilés, nous présentons maintenant la façon avec laquelle symfony2 organise les différents modules fonctionnels (fonctionnalités comme par exemple la gestion des utilisateurs, la gestion des articles,...). Ceci se fait à travers une décomposition en **bundles**.

Un bundle est une brique de l'application. Symfony2 utilise ce concept regroupant dans un même endroit (i.e, le bundle) tout ce qui concerne une même fonctionnalité. Cette organisation permet un découpage naturel des fonctionnalités. Ce découpage en bundle permet une transportabilité de ces briques permettant ainsi le partage et la réutilisabilité des différentes briques déjà développés.

Le site référence des bundles symfony2 est <http://knpbundles.com/>.

Les bundles les plus connus sont :

- ✓ FOSUserBundle : il permet la gestion des utilisateurs (e.g, connexion, inscription, déconnexion, édition d'un utilisateur) et fournit aussi les vues qui vont avec.
- ✓ FOSCommentBundle : il permet la gestion des commentaires
- ✓ GravatarBundle : il permet la gestion des avatars.

Tous les bundles ont l'arborescence conventionnelle suivante mais qui ne reste pas figée et qui peut être enrichie :

- ✓ Controller : dossier qui contient vos contrôleurs
- ✓ DependencyInjection : dossier qui contient des informations sur votre bundle
- ✓ Entity : dossier qui contient vos modèles
- ✓ Form : dossier qui contient vos éventuels formulaires
- ✓ Resources : dossier qui contient les ressources de votre appli, il possède l'arborescence suivante :
 - config dossier qui contient les fichiers de configuration de votre bundle, e.g., les routes
 - public dossier qui contient les fichiers publics de votre bundle, e.g., fichiers CSS et JavaScript, images, etc.
 - views : dossier qui contient les vues du bundle, les templates Twig
- ✓ Tests : dossier qui contient d'éventuels tests unitaires et fonctionnels.

Création de bundle :

Même si l'arborescence d'un bundle est connue, il reste pesant de la créer manuellement à chaque nouveau bundle. Dans ce sens, symfony2 offre un générateur de bundle qui permet la création automatique d'un bundle ainsi que de son contenu.

Placez-vous avec la ligne de commande dans le dossier de votre application (sur windows vous pouvez le faire en accédant au dossier puis en appuyant sur le clic droit et le bouton shift). A partir de ce dossier, nous allons exécuter des fichiers PHP, ceci est possible grâce à la variable d'environnement précédemment créée. Le fichier que nous allons utiliser est le fichier [app/console](#). Pour ce faire, exécutez la commande `php app/console`. Cette commande est une bénédiction et un point fort de symfony2 vu qu'elle permet de GENERER DU CODE

AUTOMATIQUEMENT. Elle permet aussi de gérer la base de données, de gérer les utilisateurs de la base, de vider le cache et pleins d'autres fonctionnalités.

Pour générer un bundle on utilise donc la commande suivante : `php app/console generate:bundle`. Une succession de questions/réponses vous permettra de créer votre bundle. Une explication est associée à chaque question.

Vous devez commencer par votre espace de nom (**namespace**) qui devra se terminer par le mot clef **bundle**. Il est généralement composé de 3 parties :

- ✓ le namespace racine : généralement représentatif du pseudo ou le nom du site ;
 - Une séparation avec \
- ✓ le nom du bundle (la fonctionnalité) ;
- ✓ « Bundle » est le suffixe obligatoire.

Pour bien suivre ce tutoriel appeler votre namespace `Insat/TutoBundle`

Ensuite, vous devez saisir le nom du bundle, par convention c'est le même que le namespace mais sans la séparation \

Ensuite, c'est la destination (le dossier source), il propose automatiquement le dossier par convention (le dossier src).

Ensuite, le format du fichier de configuration, travaillons avec **YAML**

Ensuite, la structure, sélectionnez le tout. Puis validez toutes les autres questions.

Votre bundle est maintenant prêt à l'emploi avec un constructeur exemple nommé `helloController`. Testez-le avec l'url suivante en indiquant le nom de votre projet : `http://localhost/Nom_Projet/web/app_dev.php/hello/World`.

PS : les dernières versions n'implémentent pas toutes cette action

Adaptation avec les différentes entités de l'architecture de symfony2 (le Routeur, les contrôleurs et les moteurs de templates twig) : Création d'une page en suivant l'architecture de symfony2

Le routeur

Comme nous l'avons vu dans les différentes étapes de traitement d'une requête par symfony2, la première étape de traitement est la capture de la requête par le contrôleur frontal qui la dispache vers le noyau de symfony2. Ce dernier l'envoie vers le routeur qui va identifier le contrôleur susceptible de traiter cette requête. Ce routeur va étudier l'url associé à la requête et en identifier l'action du contrôleur à exécuter.

Prenons l'exemple de l'url suivante : 127.0.0.1/tutosym/web/app_dev.php/Bonjour

Le routeur devra traiter cette url en indiquant à symfony2 quel traitement il doit exécuter à la réception de cette url (requête).

Visualisons le contenu du fichier routing dans le bundle que vous avez créé. Allez dans le dossier de votre bundle puis dans le dossier Ressources puis config. Voici l'arborescence permettant d'accéder à votre fichier routing :

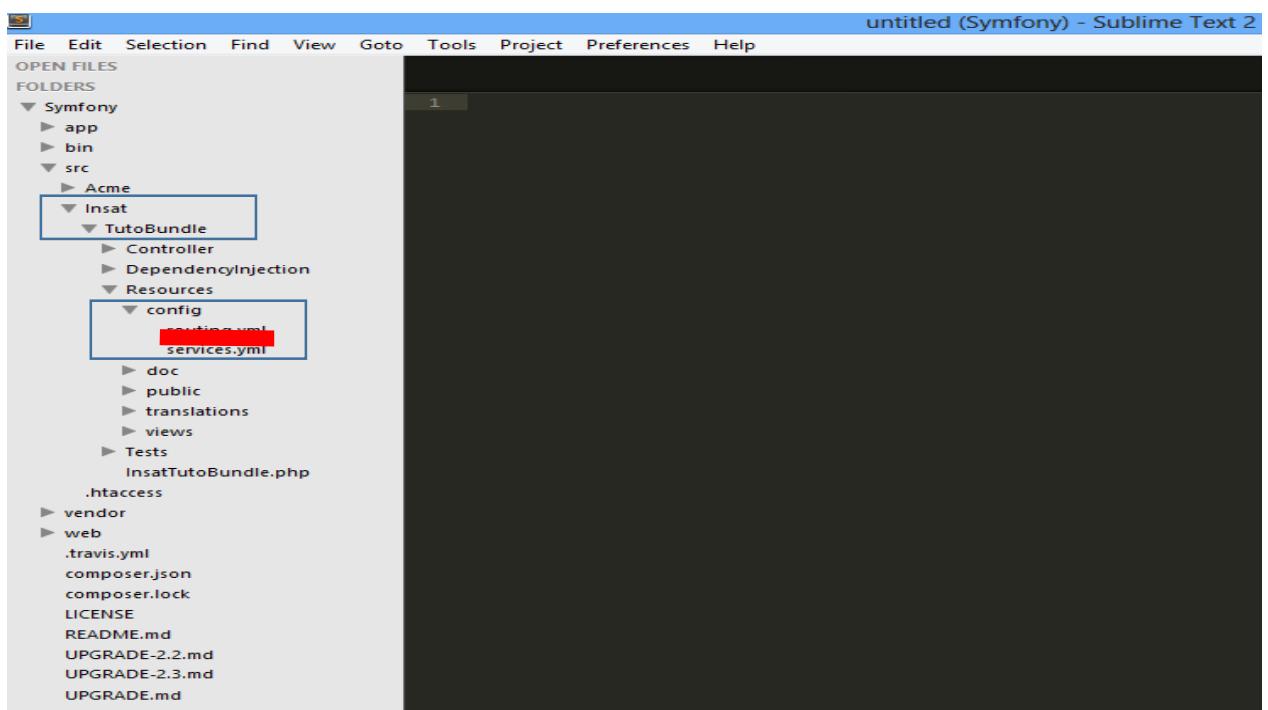


Figure 2 : Arborescence vers le rooting

Visualisez maintenant le contenu de ce fichier. Normalement vous devez avoir quelque chose comme ça.

insat_tuto_homepage:

pattern: /hello/{name}

defaults: { _controller: InsatTutoBundle:Default:index }

Il contient une seule route, à savoir, celle de l'homepage. Elle est composée de 3 parties :

- ✓ **Le nom de la route** qui représente l'alias de la route et qui permettra d'y accéder
- ✓ **Le pattern** qui représente la partie utilisable dans l'url pour accéder à cette route
- ✓ **Le default** qui permet d'associer cette route au contrôleur qui va exécuter la requête

Donc, lorsque l'on exécute l'url suivante :

http://127.0.0.1/Symfony/web/app_dev.php/hello/apprenant, le routeur ira chercher la route correspondante au pattern hello/apprenant. Une fois trouvé, il ira chercher le contrôleur correspondant se trouvant dans la partie defaults.

Décortiquons maintenant le defaults : { _controller: InsatTutoBundle:Default:index }

Cette ligne nous informe qu'il faudra aller chercher l'action (contrôleur) index du groupement de contrôleur Default qui se trouve dans le bundle Tuto de l'espace de nommage Insat. La figure 3 illustre cette décomposition de la requête pas le routeur.

The screenshot shows a Sublime Text 2 interface with the following details:

- File Path:** C:\wamp\www\Symfony\src\Insat\TutoBundle\Controller\DefaultController.php (Symfony)
- File Structure:** The left sidebar shows the project structure:
 - OPEN FILES:** Symfony
 - FOLDERS:** app, bin, src
 - Acme** (highlighted)
 - Insat** (highlighted)
 - TutoBundle** (highlighted)
 - Controller** (highlighted)
 - DefaultController.php** (highlighted)
 - DependencyInjection**
 - Resources**
 - config (routing.yml, services.yml)
 - doc
 - public
 - translations
 - views
 - Tests
 - InsatTutoBundle.php
 - .htaccess
- Code View:** The main pane displays the PHP code of DefaultController.php:

```
<?php
namespace Insat\TutoBundle\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
class DefaultController extends Controller
{
    public function indexAction($name)
    {
        return $this->render('InsatTutoBundle:Default:index.html.twig', array('name' => $name));
    }
}
```

Figure 3 : Le contrôleur symfony2

Pour l'attribut name du pattern, il sera utilisé comme paramètre d'entrée de l'action index. Nous verrons ultérieurement quelle sont les propriétés d'un contrôleur et pourquoi on parle d'action.

Une fois la route traitée par le retour, il fera appel au contrôleur pour qu'il exécute l'action souhaitée.

Le contrôleur

Comme nous l'avons vu pour les routes, les contrôleurs spécifiques au bundle se trouvent dans notre dossier TutoBundle, dans le dossier Controller. Ce dossier contiendra tous les contrôleurs de ce bundle.

Un contrôleur est composé par un ensemble d'actions. Ces actions sont les vrais contrôleurs symfony2. Chaque action est spécifique à un traitement. Le contrôleur a pour rôle de gérer le dispatching des données et traitement entre les couches vue et modèle.

Un groupe de contrôleurs symfony2 doit avoir un nom qui se termine par le suffixe **Controller**. Afin de bénéficier des méthodes prédefinies « helpers » des contrôleurs, il doit hériter de la classe Controller.

Décortiquons maintenant le DefaultController de la figure 2 proposé par symfony2.

- Ligne 3 : C'est le namespace des contrôleurs de notre bundle. Astuce : c'est la structure des répertoires dans lequel se trouve le contrôleur.
- Ligne 5 : Comme nous l'avons mentionné, chaque contrôleur hérite du contrôleur de base de symfony2 d'où l'importation de la classe en utilisant le mot clef use
- Ligne 7 : le nom de notre contrôleur respecte le nom du fichier pour que l'*autoload* fonctionne.
- Ligne 9 : ici nous définissons le contrôleur (la méthode) indexAction(). Chaque contrôleur doit se terminer par le suffixe Action. Ceci est une convention de symfony2. Le noyau cherchera directement le nom de l'action passé dans la root
- Ligne 11 : On retourne la page voulu avec comme argument l'attribut name qu'on passe à travers un tableau.

Exercice

Si vous n'avez pas encore créé votre projet Symfony2 et votre premier Bundle, fait le.

Dans votre dossier Controller de votre Bundle créer le fichier TestController suivant :

```
<?php  
  
namespace Insat\formaBundle\Controller;  
  
use Symfony\Bundle\FrameworkBundle\Controller\Controller;  
  
class TestController extends Controller  
  
{  
  
    public function premierAction()  
  
    {  
  
        // Contenu du contrôleur à préparer  
  
    }  
  
}
```

- 1) Modifier le contenu du contrôleur « premierAction » de sorte qu'il retourne du code Html. Souhaitons bonjour à l'utilisateur et lui informons que c'est notre premier contrôleur. Utiliser l'objet Response (Symfony\Component\HttpFoundation\Response) qui prend en paramètre le code html à retourner.

PS : n'oublier pas de gérer la route de votre contrôleur

La figure 4 illustre le résultat souhaité.



Figure 4 : Affichage du premier exemple

2) Créer l'action (contrôleur) cvAction

Préparer des variables permettant de créer un mini Portfolio. Le contenu de ces variables devra vous permettre d'afficher vos nom et prénom, votre âge et votre Section.

Ensuite, utiliser la méthode `render` afin d'invoquer votre page **TWIG** en lui passant les variables que vous venez de préparer.

Sachant que pour afficher une variable dans TWIG il suffit de l'entourer de {{ nomVariable }}, créer une page TWIG « cv.html.twig » dans un dossier « Premier» que vous créerez dans le dossier « views » du dossier « Resources » de votre Bundle. Cette page devra afficher les données transmises par votre contrôleur comme l'illustre la Figure 5.



Figure 5 : Affichage du mini Portfolio

Atelier Symfony2 : Le rooter de symfony et le moteur de template TWIG

OBJECTIF :

Le but de cet atelier est de comprendre et manipuler :

1. Le rooter de Symfony2
2. Le moteur de template TWIG.

Références :

Liens utiles

- <http://symfony.com/>
 - <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/>
 - <http://www.doctrine-project.org/>
- Livre
- Symfony, The Book, SensioLabs Version Aout2015.

I- Le rooter de Symfony

Le rooter de Symfony a pour but de gérer les liens interne du projet. Il remplace le « .haccess » et le fameux rewrite afin de permettre d'avoir de jolies URL plus simple et plus explicite favorisant ainsi le référencement. Son rôle majeur et d'associer une url à un traitement qui est généralement une action.

I.1 - Format de gestion de rooting

Plusieurs formats permettent la gestion des root. YAML, XML PHP ou encore les annotations permettent d'accomplir cette tâche. Dans ce Tutoriel, nous nous consacrerons à YAML.

I.2 – Fichier principal de rooting

Le fichier de rooting de base se trouve dans le dossier config du fichier app de votre projet :

app/config/routing.yml

Généralement dans ce fichier, nous trouvons un ensemble de root ainsi que des références aux fichiers roots spécifiques aux différents bundle présent dans l'application.

I.3 - Squelette d'une root

insat_tuto_homepage:

```
pattern: /hello/{name}  
defaults: { _controller: InsatTutoBundle:Default:index }
```

Il contient une seule route, à savoir, celle de [l'homepage](#). Elle est composée de 3 parties :

- ✓ [Le nom de la route](#) qui représente l'alias de la route et qui permettra d'y accéder
- ✓ [Le pattern](#) qui représente la partie utilisable dans l'url pour accéder à cette route
- ✓ [Le default](#) qui permet d'associer cette route au contrôleur qui va exécuter la requête

Ceci représente le squelette de base d'une root.

I.4 – Référencement d'un fichier route d'un bundle

Pour avoir une bonne visibilité des routes et une meilleure maintenabilité, une externalisation des roots des bundles est généralement utilisée. Dans le fichier principal, une référence au fichier externalisé est mentionnée pour dire au rooter d'aller chercher la route souhaitée dans ces fichiers.

Voici la structure d'un référencement à un fichier root d'un bundle

NomReferenc:

```
resource: "@NomBundle/Resources/config/routing.yml"
```

```
prefix: /LePrefixDeL'ensembleDesRootsDeCeFichier
```

Exemple :

```
insatforma:
```

```
resource: "@InsatformaBundle/Resources/config/routing.yml"
```

```
prefix: /forma
```

I.5 Paramétrer une root

Une root peut contenir plusieurs paramètres qui représentent des variables utilisables par le contrôleur.

Les paramètres sont ajoutés une à la fin de la root. Chaque paramètre est insérer entre {}.
Les paramètres sont séparés par des '/' ou par ':'.

Exemple :

```
insatforma_homepage:
```

```
path: /article/{year}/{langue}/{slug}.{format}
```

```
defaults: { _controller: InsatformaBundle:Default:index }
```

Ici l'url doit contenir l'année de l'article {year}, la langue de l'article {langue} les mots clefs {slug} ainsi que le format {format}

Astuce

Pour la langue utiliser la variable fourni par symfony2 `_locale` permettant ainsi de modifier en même temps la locale de la page. Pour le format utiliser le `_format` qui modifie le content-type envoyé au navigateur.

Exercice :

Ecrire le contrôleur ainsi que le fichier TWIG permettant d'afficher les données récupérées dans la root.

Vérifier les urls suivantes et dire si elles sont valides. Expliquer les cas où l'url n'est pas valide et proposer une correction. N'oublier pas de changer le nom du dossier pour qu'il corresponde au nom de votre projet.

- a) http://127.0.0.1/forminsat/web/app_dev.php/article/2005/fr/page.twig-symfony-routing.twig.html
- b) http://127.0.0.1/forminsat/web/app_dev.php/article.a/2005/fr/page.twig-symfony-routing.twig.html
- c) http://127.0.0.1/forminsat/web/app_dev.php/article/2005/fr/twig-symfony/-routing.twig.html

I.6 Valeurs par défaut des paramètres d'une root

Nous pouvons spécifier la valeur d'un paramètre d'une root comme valeur par défaut dans le cas de l'omission de cette dernière. Le format par exemple peut être mis par défaut à « html » l'utilisateur n'aura plus à le mentionner dans ce cas.

Syntaxe

`front_article:`

```
path: /article/{year}/{_locale}/{slug}.{_format}
defaults: { _controller: InsatformaBundle:Default:index, variable:valeurParDefaut }
```

Exemple

`front_article:`

```
path: /article/{year}/{langue}/{slug}.{format}
defaults: { _controller: InsatformaBundle:Default:index, _format: html }
```

Important : Seul un paramètre optionnel terminant la root pourra être absent de l'URL

a) Sans valeur par défaut l'url

http://127.0.0.1/forminsat/web/app_dev.php/article/2005/fr/page.twig-symfony-routing.twig.html

est invalide. Devient-elle valide avec l'ajout de la valeur par défaut.

I.7 Ajouter des contraintes sur les paramètres de la root

Le contrôle des paramètres de la root se fait à travers les [requirements](#).

Syntaxe :

```
front_article:  
    path: /article/{year}/{_locale}/{slug}.{_format}  
    defaults: { _controller: InsatformaBundle:Default:index, variable:valeurParDefaut }  
    requirements :  
        nomParamètre: condition
```

Exemple

```
front_article:  
    path: /article/{year}/{langue}/{slug}.{format}  
    defaults: { _controller: InsatformaBundle:Default:index, _format: html }  
    requirements:  
        year:\d{4} // l'année sera obligatoirement un chiffre composé de 4 entiers  
        langue: fr|en // ici la langue ne pourra être qu'anglais ou français
```

Les [requirements](#) les plus utilisés sont les suivants :

\d équivalente à \d{1}

\d+ ensemble d'entiers

Gestion des méthodes http :

_method: POST ceci signifie que la root ne s'exécutera qu'à travers la méthode POST

Nous pouvons aussi utiliser le GET

Exercice

Reprendre l'exercice de l'atelier numéro 1 dont l'énoncé est la suivant :

Préparer des variables permettant de créer un mini Portfolio. Le contenu de ces variables devra vous permettre d'afficher vos nom et prénom, votre âge et votre Section.

Ensuite utiliser la méthode `render` afin d'invoquer votre page **TWIG** en lui passant les variables que vous venez de préparer.

Sachant que pour afficher une variable dans TWIG il suffit de l'entourer de `{} nomVariable {}`, créer une page TWIG « cv.html.twig » dans un dossier « Premier» que vous créerez dans le dossier « views » du dossier « Resources » de votre Bundle. Cette page devra afficher les données transmises par votre contrôleur comme l'illustre la Figure 1.



Figure 1 : Affichage du mini Portfolio

Remarque :

Les données ne devront plus être récupérées dans le contrôleur mais à travers la root.

Contrôler que l'âge soit un entier ne dépassant pas les 99 ans, que la section soit GL ou RT. La valeur par défaut du format est html. La langue de la page doit être anglais ou français.

Essayer de trouver une astuce pour modifier la langue de l'affichage selon la valeur de la langue récupérée.

II- Le moteur de template TWIG

Les TWIGs représentent le V (Vue) du modèle MVC. La majorité des projets Symfony2 utilise le moteur de Template TWIG pour les différents avantages qu'il offre.

II.1 Objectif

L'objectif principal de Twig est de permettre aux développeurs de séparer la couche de présentation (Vue du MVC) dans des templates dédiés, afin de favoriser la maintenabilité du code.

Il représente une alternative idéale pour les graphistes qui ne connaissent pas forcément le langage PHP et qui s'accommoderont parfaitement des instructions natives du moteur, relativement simples à maîtriser.

II.2 Les bases du langage TWIG

II.2.1 Afficher le contenu d'une variable

Syntaxe :

`{{ maVar }}` C'est l'équivalent de l'echo dans php ou du `<%= %>` pour les jsp. Les doubles accolades permettent d'imprimer une valeur, le résultat d'une fonction...

Exemple : Le nom de l'utilisateur est {{ nom }}
 il a {{ age }} années.

Fonctionnalité	Exemple Twig
Afficher une variable	Variable : {{ MaVari }}
Afficher le contenu d'une case d'un tableau	Identifiant : {{ tab['id'] }}
Afficher l'attribut d'un objet, dont le getter respecte la convention \$objet->getAttribut()	Identifiant : {{ user.id }}
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	MaVariable majus : {{ MaVariable upper }}
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule.	Message : {{ news.texte striptags title }}
Utiliser un filtre avec des arguments. Attention, il faut que date soit un objet de type Datetime ici.	Date : {{ date date('d/m/Y') }}
Concaténer	Identité : {{ nom ~ " " ~ prenom }}

II.2.2 Ecriture de code

Syntaxe :

{% code %}

Les accolades pourcentage permettent d'exécuter une fonction, définir un bloc...

Exemple :

{% set maVariable = « test » %}

II.2.3 Les commentaires

Syntaxe :

{# commentaires #}

Permettent de définir un bloc comme étant un commentaire donc non interprétable

Exemple :

{# ceci est un commentaire #}

III Les filtres dans TWIG

Les filtres permettent d'effectuer des traitements sur les variables très utiles. Pour appliquer le filtre il faut ajouter ‘|’ après la variable suivie du filtre.

Filtre	Description	Exemple Twig
Upper/lower	Met toutes les lettres en majuscules/minuscules.	<pre>{{ var upper }}</pre> <pre>{{ var lower }}</pre>
Striptags	Supprime toutes les balises XML.	<pre>{{ var striptags }}</pre>
Date	<p>Formate la date selon le format donné en argument.</p> <p>La variable en entrée doit être une instance de Datetime.</p>	<pre>{{ date date('d/m/Y') }}</pre> <p>Date d'aujourd'hui : {{ "now" date('d/m/Y') }}</p>
Format	Insère des variables dans un texte, équivalent à printf .	<pre>{{ "Il y a %s pommes et %s poires" format(153, nb_poires) }}</pre>
Length	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	<p>Longueur de la variable :</p> <pre>{{ texte length }}</pre> <p>Nombre d'éléments du tableau :</p> <pre>{{ tableau length }}</pre>

abs	Permet de s'assurer d'avoir l'affichage d'une valeur positive sur sa vue	<code>{{ number abs }}</code>
Capitalize	La chaîne de caractère commence par une lettre en capitale.	<code>{{ 'Je commence par une majiscule' capitalize }}</code>
convert_encoding	Modifie un encodage par un autre	<code>{{ data convert_encoding('UTF-8', 'iso-2022-jp') }}</code>
date_modify	Permet de modifier une date	Dans 3 jours nous serons le : <code>{{ "now" date_modify("+3 days") date("m/d/Y") }}</code>
default	Retourner la valeur par défaut d'une variable si elle est indéfinie ou vide	<code>{{ test default('la variable test n'est pas définie') }}</code>
Join	Concaténer des éléments avec un séparateur : join()	<code>{{ ['a','y','m','e','n'] join(':') }}</code>
Keys	Retourner les index d'une séquence	<code>% set aymen=['a','y','m','e','n'] %</code> <code>% for key in aymen keys %</code> <code> {{ key }}
</code> <code>% endfor %</code>
Merge	Ajouter des éléments à une séquence	<code>% set pair = [2,4,6]%</code> <code>% set impair = [1,3,5]%</code> <code>% set pairimpair = pair merge(impair) %</code> <code>% for chiffre in pairimpair %</code> <code> {{ chiffre }}
</code> <code>% endfor %</code>
Reverse	Renverser une séquence ou une chaîne de caractère	<code>% set aymen=['a','y','m','e','n'] %</code> <code>{{ ['a','y','m','e','n'] join(':') reverse }}</code>

Sort	Trier une séquence en conservant l'association des index	{% for chiffre in pairimpair sort %} {{ chiffre }} {% endfor %}
------	--	---

Pour plus de filtres veuillez consulter l'url suivante :

<http://twig.sensiolabs.org/doc/filters/index.html>

Pour les tests nous utilisons

is defined l'équivalent de **isset** en php

Rôle vérifie l'existance d'une variable

Exemple: { % if nom is defined % } J'existe { % endif% }

even

Rôle vérifie si la variable est pair

Exemple : { % if age is even % }Pair{ % endif% }

odd

Rôle vérifie si la variable est impair

Exemple : { % if age is odd% }Impair{ % endif% }

Exercice d'application :

Reprendre l'exemple précédent et appliquer les filtres suivants sur l'affichage :

Les nom et prénoms doivent être en majuscule

L'âge doit être positif

Donner des valeurs par défaut aux variables

IV Les structures de contrôle

Les structures que ce soit séquentielles ou itératives sont souvent très proches du langage naturel. Elles sont introduites entre { % % }. Généralement elle se termine par une expression de fin de block.

IV.1 Les structures conditionnelles

« if »

Syntaxe :

{% **if** cnd %}

Block de traitement

{%**endif**%}

Exemple

{% if employee.salaire < 250 %}

ce salaire est inférieur au smig

{%**endif**%}

« if elseif else »

Syntaxe :

{% **if** cnd %}

block traitement

{% **elseif** cnd2 %}

block traitement

{% **else** %}

block traitement

{% **endif** %}

Exemple

{% **if** maison.tempertature <0% }

Très Froid

{% **elseif** maison.tempertature <10 % }

Froid

```
{% else %}
```

Bonne température

```
{% endif %}
```

IV.2 Les structures itératives

Syntaxe :

```
{% for valeur in valeurs %}
```

block à répéter

```
{% else %}
```

Traitements à faire si il n'y a aucune valeur

```
{% endfor %}
```

Exemple

La formation de l'équipe A est :


```
<ol>
```

```
{% for joueur in joueurs %}
```

```
    <li> {{player.nom}} </li>
```

```
{% else %}
```

La liste n'a pas encore été renseignée

```
{% endfor %}
```

```
</ol>
```

V Accès aux Templates

Les Templates doivent se trouvent dans l'un des deux emplacements suivants :

- app/ressources/views
- /ressources/views d'un bundle

Afin d'accéder aux Template une convention de nommage est définit :

`NomBundle:DossierFichier:pagetwig`

Exemples

- ✓ `::index.html.twig` // ce fichier se trouve directement dans app/ressources/views
- ✓ `RT4Bundle::index.html.twig` // ce fichier se trouve dans src/RT4Bundle/ressources/views
- ✓ `RT4Bundle:Multimedia:index.html.twig` // ce fichier se trouve dans src/RT4Bundle/Multimedia/ressources/views

VI Convention de nommage des pages TWIG

Par convention le nommage des vues dans symfony se fait selon la convention suivante :

`NomPAge.FormatFinal.MoteurdeTemplate`

Exemple

`index.html.twig`

Le nom du fichier est index, le format final sera du html et le moteur de Template suivi est le TWIG

VII L'héritage dans TWIG

Le templating dans TWIG se base sur la notion d'héritage qui se rapproche du concept d'héritage de l'orienté objet.

La figure 2 illustre la vision d'héritage des templates.

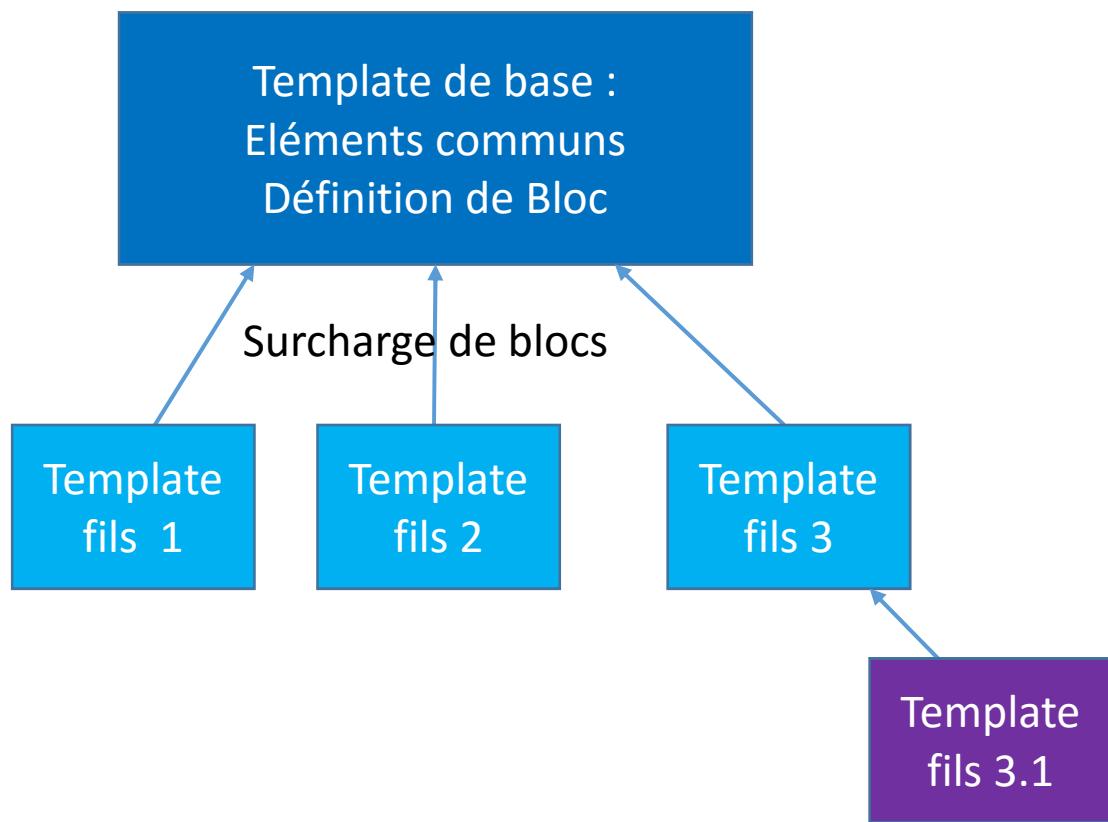


Figure 2 : Modélisation du concept d'héritage de Template

Généralement Un Template de base modélise le formalise et la structure générique des pages du projet ou du bundle. Ce Template est décomposé en un ensemble de bloc qui va permettre cette notion d'héritage. Chaque bloc peut être surchargé par les templates fils. Les parties non surchargées et les parties non surcharchables seront héritées.

Le fichier de base fourni par Symfony est le suivant :

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>
            {% block title %}Bonjour je suis le bloc principal!{% endblock %}
        </title>
        {% block stylesheets %}
            {#ici je vais définir les fichiers css communs à l'ensemble des pages de l'application#}
            {% endblock %}
            <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
            {# ici c'est le favicon de mon appli asset permet de définir le path de ce fichier #}
        </head>
        <body>
            {% block body %}
                {#ici c'est le contenu du Body#}
                Bienvenu dans le body du bloc principal
            {% endblock %}
            {% block javascripts %} {#ici je vais définir mes fichiers js#} {% endblock %}
        </body>
    </html>

```

Afin d'étendre un template on utilise le 'extends'

Exemple

```
{% extends '::base.html.twig' %}
```

La figure * montre un petit exemple d'un page qui a surchargé le block body du fichier père.

```
{% extends '::base.html.twig' %}
```

```
{% block body %} Bonjour J'ai pris mon indépendance et j'ai défini mon propre body {% endblock %}
```



Bonjour J'ai pris mon indépendance et j'ai défini mon propre body

Figure 3 : Exemple de surcharge d'un block body

Remarque : Si on n'avait pas mentionné le block body on aura eu une erreur

Dans les templates fils qui vont étendre ce template de base, il faut généralement surcharger le block body.

Si on veut **ajouter une partie à un block** tout en **gardant le contenu** du template père de ce block, il suffit de faire appel à la méthode **parent()**.

La Figure 4 illustre un exemple d'utilisation du parent().



```
{% extends '::base.html.twig' %}
```

```
{% block body %}
```

```
 {{ parent() }}<br>
```

Bonjour J'ai pris mon indépendance et j'ai défini mon propre body

```
{% endblock %}
```



```
{% extends 'Rt4AsBundle:Default:fils.html.twig' %}
```

```
{% block body %}
```

```
 {{ parent() }}<br>
```

Bonjour J'ai pris moi aussi mon indépendance je suis le petit fils et j'ai défini mon propre body

```
{% endblock %}
```

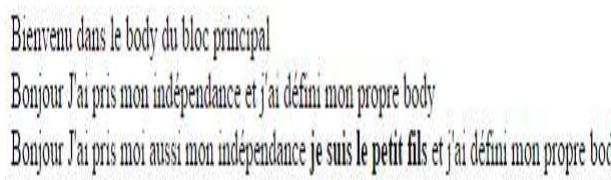


Figure 4 : Illustration de l'utilisation de parent()

La figure 5 représente une illustration de ce qu'on pourra faire afin de bien utiliser ce concept d'héritage offert par TWIG.

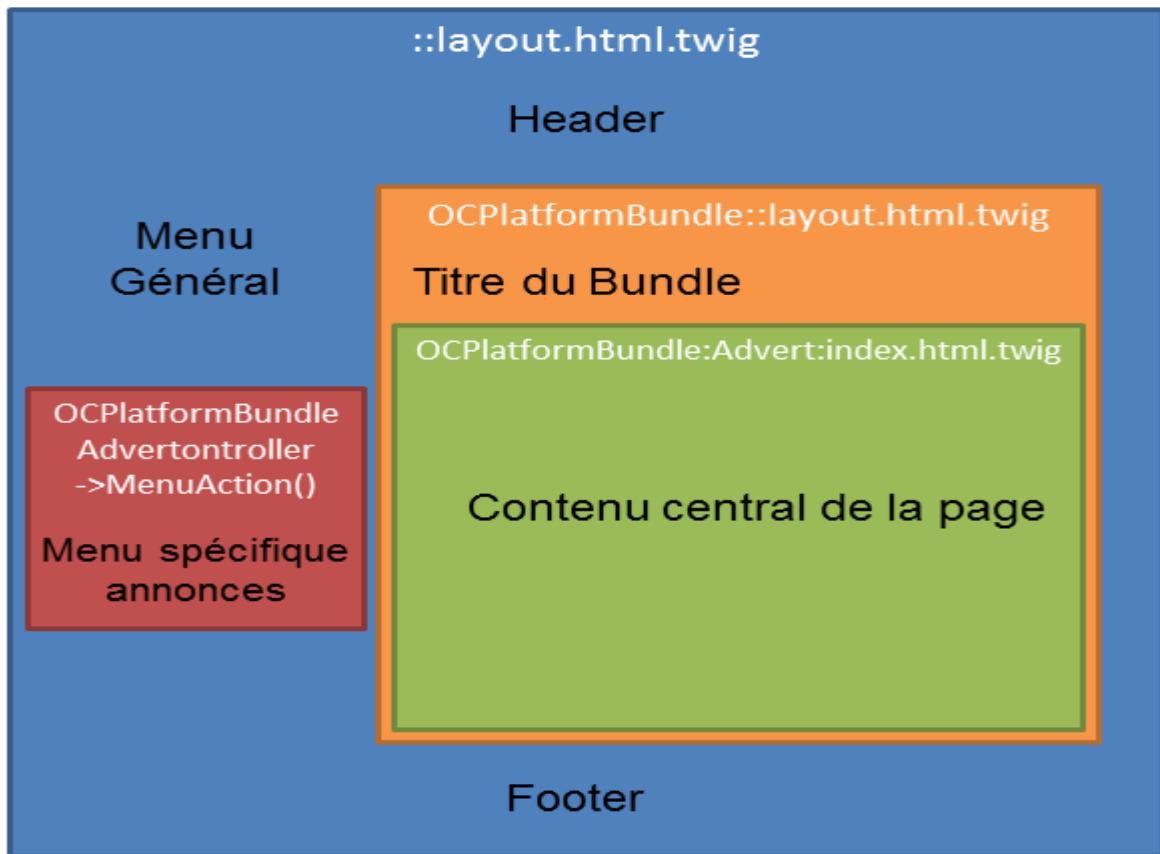


Figure 5 Découpage en bloc d'un Template

(<https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/le-moteur-de-templates-twig>)

VIII L'inclusion de Template

Afin d'inclure un template dynamique (le contenu dépend de certains paramètre) tel qu'un Template des meilleures ventes, un Template des articles les plus vus, les derniers cours postés. Il faut inclure un contrôleur sans ou avec des paramètres.

Syntaxe :

```
{ { render(controller('NomBundle:NomController:NomAction',           {'labelParam1': param1,'labelParam2': param2,... })) } }
```

Exemple

```
{ { render(controller('Rt4AsBundle:Student:Top', { 'nb': 10 })) } }
```

IX Génération des liens avec les TWIG

TWIG permet de générer les liens automatiquement à partir des noms de root en utilisant la fonction `path`. La Figure illustre le fonctionnement de `path`.

```
|rt4_as_homepage:  
|    path:      /hello/{name}                                {{ extends 'Rt4AsBundle:Default:fils.html.twig' }}  
|    defaults: { _controller: Rt4AsBundle:Default:index }  
  
|rt4_as_base:  
|    path:      /base  
|    defaults: { _controller: Rt4AsBundle:Default:base }  
  
|rt4_as_fils:  
|    path:      /fils  
|    defaults: { _controller: Rt4AsBundle:Default:fils }  
  
|rt4_as_ptifils:  
|    path:      /ptifils  
|    defaults: { controller: Rt4AsBundle:Default:petitFils }  
  
|{{ block body }}  
  
{{ parent() }}<br>  
Bonjour J'ai pris moi aussi mon indépendance <b>je suis le petit fils</b> et j'ai défini mon propre body <br>  
  
<a href="{{ path('rt4_as_base') }}">mon grand père est ici</a><br>  
<a href="{{ path('rt4_as_fils') }}"> mon père est ici </a><br>  
{{ endblock }}
```

Figure 6 : Exemple d'utilisation de path

Pour les root qui contiennent des paramètres en entrée, `path` permet de les introduire en utilisant la syntaxe suivante :

Syntaxe :

```
{ { path('root', { param1: valparam1, param2 : valparam2,... }) } }
```

La Figure 7 illustre le fonctionnement de path avec les paramètres.

```

rt4_as_homepage:
    path:      /hello/{name}
    defaults: { _controller: Rt4AsBundle:Default:index }
    extends: 'Rt4AsBundle:Default:fils.html.twig'
    block body

rt4_as_base:
    path:      /base
    defaults: { _controller: Rt4AsBundle:Default:base }
    extends: 'parent()'<br>
    block body
        Bonjour J'ai pris moi aussi mon indépendance <b>je suis le petit fils</b> et j'ai défini mon propre body <br>

rt4_as_fils:
    path:      /fils
    defaults: { _controller: Rt4AsBundle:Default:fils }
    extends: 'parent()'<br>
    block body
        <a href="{{ path('rt4_as_base') }}>mon grand père est ici</a><br>
        <a href="{{ path('rt4_as_fils') }}> mon père est ici </a><br>
        <a href="{{ path('rt4_as_homepage', {name: 'aymen'}) }}> Bonjour </a><br>

rt4_as_ptifils:
    path:      /ptifils
    defaults: { _controller: Rt4AsBundle:Default:petitFils }
    extends: 'parent()'
    block body
        <a href="{{ path('rt4_as_base') }}>mon grand père est ici</a><br>
        <a href="{{ path('rt4_as_fils') }}> mon père est ici </a><br>
        <a href="{{ path('rt4_as_homepage', {name: 'aymen'}) }}> Bonjour </a><br>

```

Figure 7 : Exemple d'utilisation de path avec paramètres

Pour générer le path d'un fichier (img, css, js,...) nous utilisons la fonction asset.

Cette fonction permet la portabilité de l'application en permettant une génération automatique du path du fichier indépendamment de l'emplacement de l'application.

Exemple :

- Si l'application est hébergé directement dans la racine de l'hôte alors le path des images est /images/nomImg.

➤ Si l'application est hébergé dans un sous répertoire de l'hôte alors le path des images est /nomApp/images/nomImg.

Syntaxe :

```
asset('ressource')
```

Exemples

```

```

```
<link href="{{ asset('css/blog.css') }}" rel="stylesheet" type="text/css" />
```

Cette fonction asset est très utilisée lors de l'intégration des templates. Dans notre fichier base qui contiendra les déclarations des fichiers css et js on utilisera cette fonction pour définir la source de ces fichiers.

X Surcharge de Template

Afin de surcharger les Template d'un Bundle, nous nous basons sur le fonctionnement de base de symfony2.

Lorsque le contrôleur fait appel à un Template dans un Bundle, Symfony2 cherche dans 2 emplacements selon l'ordre suivant :

- 1) app/Resources/NomBundle/views/DossierTemplate/nomTemplate.html.twig
- 2) src/NomAppli/NomBundle/Resources/views/DossierTemplate/nomTemplate.html.twig

Pour surcharger le template il suffit donc de copier ce dernier vers le dossier app/Resources/NomBundle/views/DossierTemplate/ que vous devez créer vous-même et personnaliser le Template à votre guise en gardant le nom du fichier.

Exercice

- 1) Télécharger le Template Bootstrap SB admin à partir du lien suivant :
<http://startbootstrap.com/template-overviews/sb-admin/>
- 2) Intégrer ce Template à votre projet le Template père doit contenir le contenu de la page blank-page. La figure 8 introduit le résultat estompé.

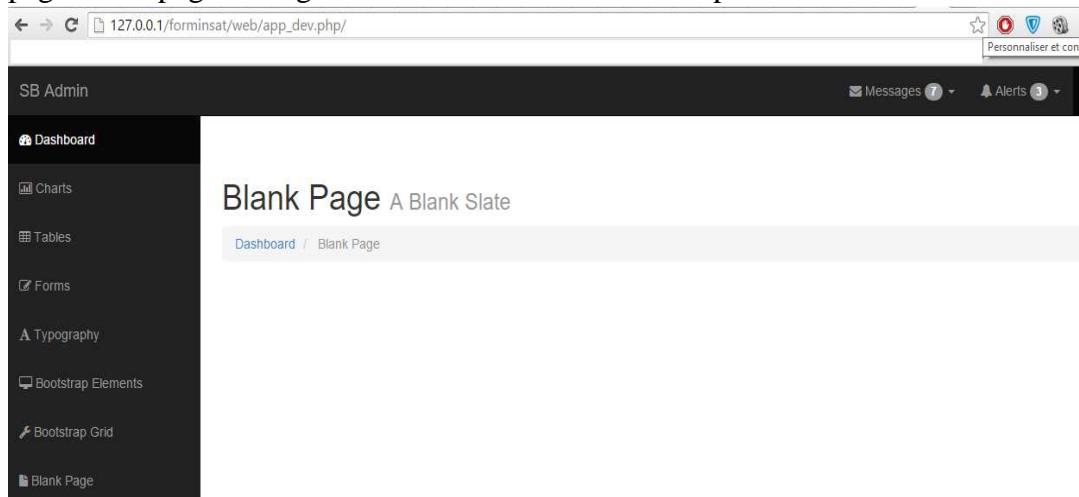


Figure 8 : Intégration du Template

3) Récupérer à partir du template edmin fourni avec la formation les parties manquantes afin de récupérer de la page lister-all la partie spécifique à un utilisateur et intégrer la dans une page cv.html.twig. N'oublier pas d'ajouter la root ainsi que l'action nécessaire pour l'affichage de la page illustré par la figure 9. Prenez en compte la gestion des parties encadrés en rouge dans l'image Définissez les blocs nécessaire pour faire en sorte de rendre ces parties paramétrables dans les Templates fils.

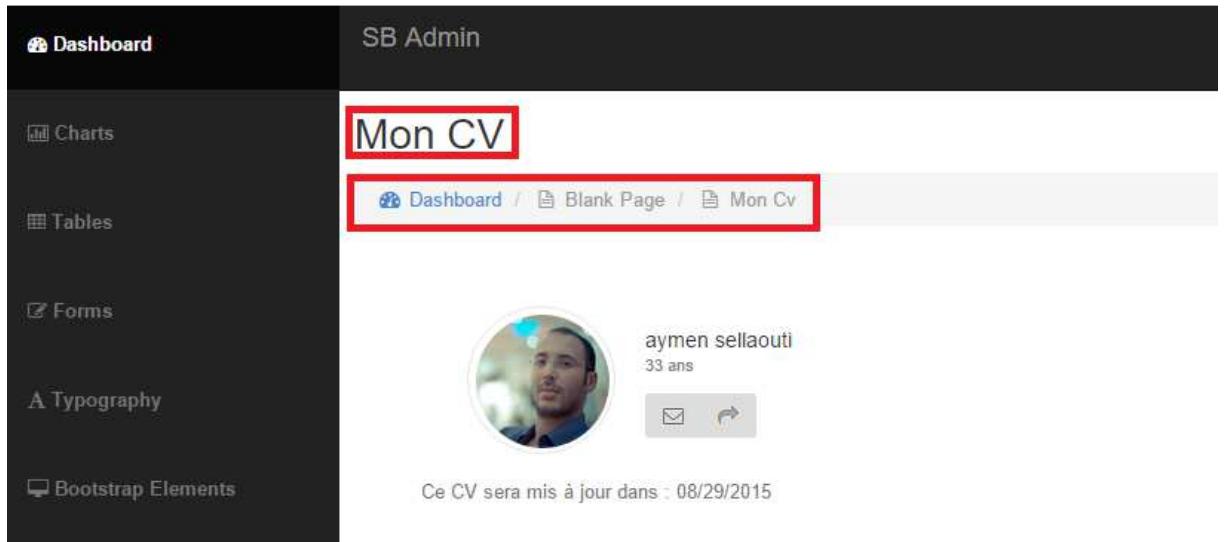


Figure 9 : La page mon CV intégré dans le Template

4) Nous voulons maintenant afficher un groupe de cv dans une nouvelle page cvs.html.twig. Commencer par créer une classe Personne contenant comme attributs nom, prenom, age et path. Créer un contrôleur dans lequel instancier quelques objets personnes. Envoyer ces personnes à la page cvs.html.twig et afficher les comme le montre la Figure 10.

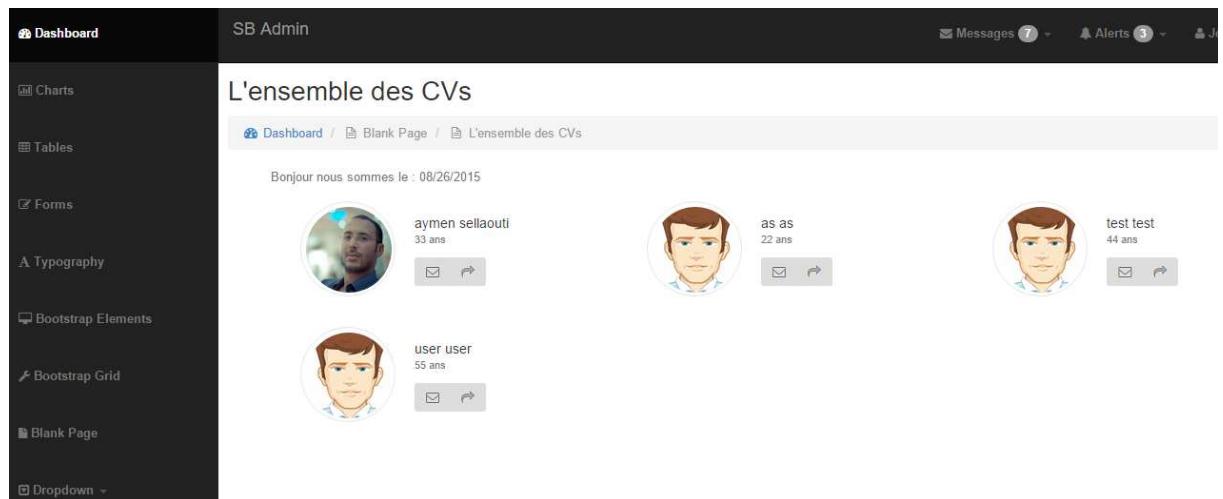


Figure 10 : Liste des CVs

- 5) Modifier le code de sorte que lorsque la personne ne possède pas de photo on lui affecte une photo par défaut que vous définirez. Dans l'exemple de la Figure 10 nous avons pris la photo user.png du template edmin bootstrap ;
 - 6) Appliquer quelques filtres TWIG afin d'améliorer l'affichage (nom prenom qui commence toujours par une majuscule, un affichage dans le cas où il n'y a aucun cv, ...).
 - 7) Si l'utilisateur s'appelle aymen, alors le bouton flèche de son post-it (celui qui est à côté du bouton enveloppe) devra l'envoyer vers la page « cv.html.twig ».
- Ps : pour comparer 2 variables string en TWIG on utilise la syntaxe suivante :

`var1 is sameas(var2)`

Atelier Symfony2 : Les entités et Doctrine

OBJECTIF :

Le but de cet atelier est de comprendre et manipuler :

1. Les entités
2. L'ORM Doctrine
3. Les relations entre les entités

Référence :

Liens utiles

- <http://symfony.com/>
- <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/>
- <http://www.doctrine-project.org/>

Livre

- Symfony, The Book, SensioLabs Version Aout2015.

I- Introduction

Dans cette partie nous allons aborder la M (Modèle) du MVC. La gestion de cette couche sera faite en utilisant un « Object Relation Mapping (ORM)» qui permettra d'avoir une abstraction avec la base de données et permettra de gérer la persistance des données dans la base sans contact direct avec la base de données et sans passer par le langage SQL.

L'ORM permettra essentiellement de mapper les tables de la BD relationnelle avec des objets donnant l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. Afin de permettre cette illusion, l'ORM utilisé par la communauté Symfony est Doctrine. Le mapping se fait avec des objets appelés Entité (Entity). La figure 1 illustre le rôle de Doctrine.

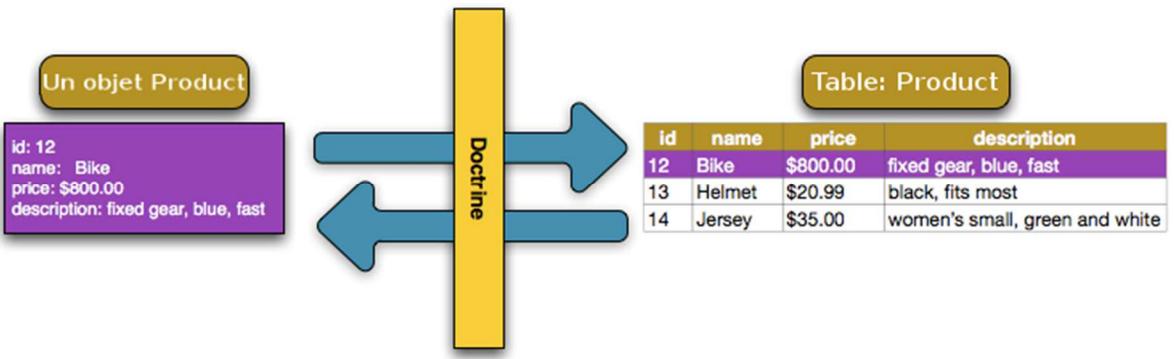


Figure 1 : Rôle de Doctrine

II- Les entités

II.1 Définition

Une entité de point de vue Symfony représente les objets PHP équivalents à une table de la base de données. Une entité est généralement composée par les attributs de la table ainsi que leurs getters et setters. Elle est l'élément de base de l'ORM.

II.1 Mapping à base d'annotation

Le [mapping](#) a pour rôle de faire le [lien](#) entre les [entités](#) et [les tables de la base de données](#).

Le mapping est fait à travers des [métadonnées](#) pouvant être associés directement aux attributs dans l'objet entité directement à travers l'utilisation des [annotations](#) ou en utilisant des fichiers externes en YAML ou en XML. Nous nous intéressons ici particulièrement aux annotations.

Lien à travers les métadonnées

[Remarque](#) : Dans un même bundle vous devez utiliser le même format de mapping. JE vous conseille de le faire pour tout le projet.

[Syntaxe](#) :

```
/*
 * les différentes annotations
 */
```

Voici les différentes annotations utilisées pour effectuer le mapping d'une entité.

@ORM\Entity

Cette annotation permet de définir un **objet** comme une **entité**. Elle est applicable sur une **classe** en la plaçant avant la définition de la classe en PHP

Syntaxe :

```
@ORM\Entity
```

Paramètres :

repositoryClass (facultatif). Permet de préciser le namespace complet du repository qui gère cette entité.

Exemple :

```
@ORM\Entity(repositoryClass="Insat\formaBundle\Entity\PersonneRepository")
```

@ORM\Table()

Permet de **spécifier le nom de la table dans la base de données à associer à l'entité**. Il est applicable sur une classe et placée avant la définition de la classe en PHP

Cette annotation est **facultative**. En effet, sans cette annotation le nom de la table sera automatiquement le nom de l'entité. Elle est généralement utilisable pour ajouter des préfixes ou pour forcer la première lettre de la table en minuscule.

Syntaxe : @ORM\Table()

Exemple :

```
/**  
 * Personne  
 *  
 * @ORM\Table()  
 * ORM\Entity(repositoryClass="Insat\formaBundle\Entity\PersonneRepository")  
 */
```

@ORM\Column()

C'est l'une des annotations les plus importantes. Elle permet de définir les caractéristiques de la colonne concernée. Elle est applicable sur un attribut de classe juste avant la définition PHP de l'attribut concerné.

Syntaxe : @ORM\Column()

Exemple :

```
/**  
 * @var string  
 *  
 * @ORM\Column(name="nom", type="string", length=255)  
 */  
  
private $nom;
```

L'un des paramètres de l'annotation `@ORM\Column` est le paramètre `type` qui introduit le type de la variable utilisée. Les types de colonnes utilisées par Doctrine ne sont pas les mêmes que ceux utilisés par SQL. Le tableau suivant illustre la correspondance entre les Types de Doctrine et ceux de SQL.

Type Doctrine	Type SQL	Type PHP	Utilisation
String	VARCHAR	string	Toutes les chaînes de caractères jusqu'à 255 caractères.
Integer	INT	integer	Tous les nombres jusqu'à 2 147 483 647.
Smallint	SMALLINT	integer	Tous les nombres jusqu'à 32 767.
Bigint	BIGINT	string	Tous les nombres jusqu'à 9 223 372 036 854 775 807.
Boolean	BOOLEAN	boolean	Les valeurs booléennes true et false.
Decimal	DECIMAL	double	Les nombres à virgule.
date ou datetime	DATETIME	objet DateTime	Toutes les dates et heures.

Time	TIME	objet DateTime	Toutes les heures.
Text	CLOB	string	Les chaînes de caractères de plus de 255 caractères.
Object	CLOB	Type de l'objet stocké	Stocke un objet PHP en utilisant serialize/unserialize.
Array	CLOB	array	Stocke un tableau PHP en utilisant serialize/unserialize.
Float	FLOAT	double	Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur.

Les autres paramètres de cette annotation sont les suivants :

Paramètre	Valeur par défaut	Utilisation
Type	string	Définit le type de colonne comme nous venons de le voir.
Name	Nom de l'attribut	Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet
Length	255	Définit la longueur de la colonne (pour les strings).
Unique	false	Définit la colonne comme unique (Exemple : email).
Nullable	false	Permet à la colonne de contenir des NULL.
Precision	0	Définit la précision d'un nombre à virgule (decimal)
Scale	0	le nombre de chiffres après la virgule (decimal)

<https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/la-couche-metier-les-entites>

Pour l'attribut id, qui est en mode auto génération, il est modélisée comme suit :

```
/**  
 * @var integer  
 *  
 * @ORM\Column(name="id", type="integer")  
 * @ORM\Id  
 * @ORM\GeneratedValue(strategy="AUTO")  
 */  
  
private $id;
```

Deux méthodes permettent de générer les entités. Les entités d'un bundle sont placées dans un dossier Entity directement sous le bundle (au même niveau que les dossiers Controller, Resources,...).

Méthode manuelle (non recommandée)

- Créer la classe
- ajouter le mapping
- ajouter les getters et les setters (manuellement ou en utilisant la commande suivante :

php app/console doctrine:generate:entities (elle crée les getters et les setters de toutes les entités)

Méthode en utilisant les commandes

- Il suffit de lancer la commande suivante :

php app/console doctrine:generate:entity

- Ajouter les attributs ainsi que les paramètres qui vont avec
- Une fois terminé, Doctrine génère l'entité avec toutes les métadonnées de mapping

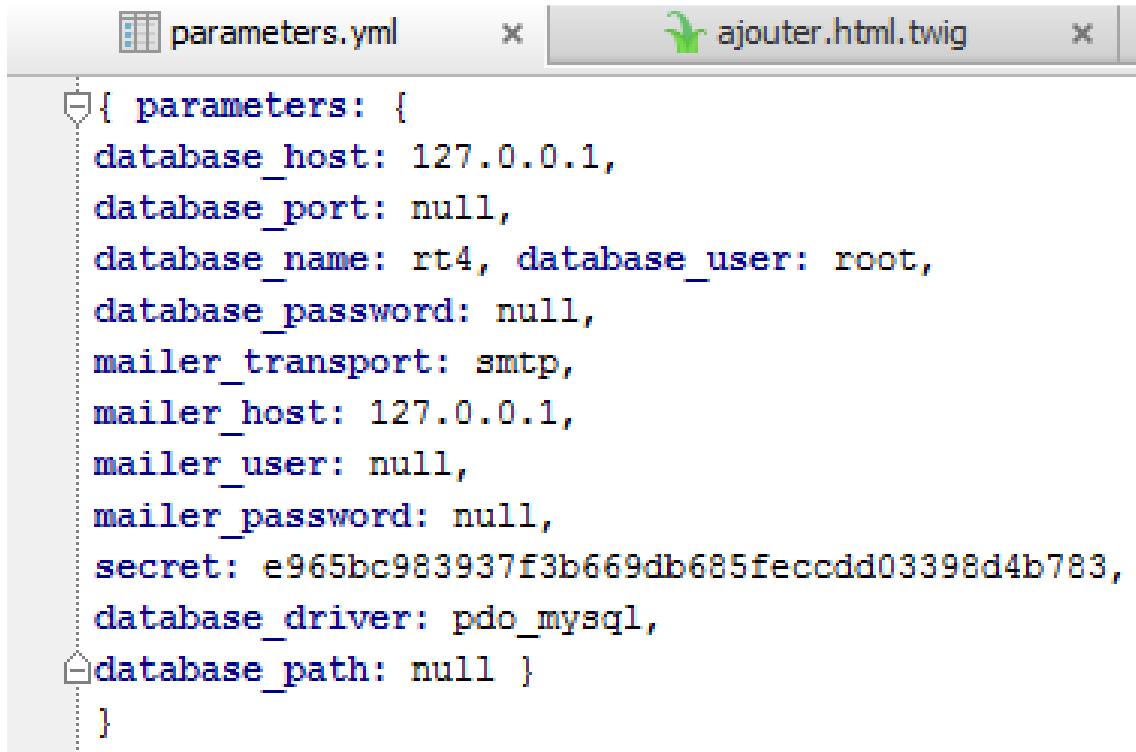
Exercice :

Utiliser les deux méthodes pour générer votre Entité Personne utilisée dans le tutoriel sur les TWIG. Pour rappel, cette entité contient un attribut id, un attribut nom, prenom, age et path. Pour la première version appelé l'entité Personne1.

III- Gestion de la base de données avec Doctrine

III.1 Crédation de la base de données

Afin de créer la base de données sans passer par votre SGBD, il suffit de configurer la BD de votre application à travers le fichier parameters.yml se trouvant sous app/config. La figure 2 montre les différentes propriétés à renseigner.



The screenshot shows a code editor with two tabs: 'parameters.yml' and 'ajouter.html.twig'. The 'parameters.yml' tab is active and displays the following YAML configuration:

```
{ parameters: { database_host: 127.0.0.1, database_port: null, database_name: rt4, database_user: root, database_password: null, mailer_transport: smtp, mailer_host: 127.0.0.1, mailer_user: null, mailer_password: null, secret: e965bc983937f3b669db685fecdd03398d4b783, database_driver: pdo_mysql, database_path: null } }
```

Figure 2 : parameters.yml

Une fois le fichier configuré, vous devez exécuter la commande suivante :

```
php app/console doctrine:database:create
```

Une base de données avec les propriétés mentionnées dans parameters.yml sera automatiquement générée.

III.2 Crédation de la base de données

Afin de créer les tables de la base de données doctrine se base sur les entités placées dans les différents dossiers Entity des différents bundles de l'application.

Deux commandes permettent de créer les tables de la BD :

- ✓ `php app/console doctrine:schema:create`
- ✓ `php app/console doctrine:schema:update --force` // utilisable pour la création et pour la mise à jour d'une table

Astuce :

php app/console doctrine:schema:update --dump-sql // Affiche les requêtes SQL à exécuter pour la BD

Exercice :

Créer votre base de données et générer vos deux tables Personne et Personne1. Vérifier qu'elles sont toutes les deux conformes.

La figure 3 illustre le résultat voulu.

The screenshot shows the phpMyAdmin interface for the 'formainsat' database. On the left, the database structure is visible with 'Nouvelle base de données' and several tables like 'animaux', 'animaux2', 'ecodocf', 'formation', 'information_schema', 'mysql', 'performance_schema', and 'test'. The 'personne' table is selected and highlighted with a red box. The main panel displays the 'Structure' of the 'personne' table, which has five columns: 'id' (int(11)), 'nom' (varchar(255)), 'prenom' (varchar(255)), 'age' (int(11)), and 'path' (varchar(100)). Each column has its type, length, and various settings like 'Null', 'Défaut', and 'Extra' (e.g., AUTO_INCREMENT). Action buttons for 'Modifier', 'Supprimer', 'Primaire', 'Unique', and 'Index' are shown for each column. At the bottom, there are buttons for 'Ajouter' (Add), 'Exécuter' (Execute), and other database operations.

Figure 3 : Crédation de la base de données

III.3 Les services de Doctrine

Doctrine offre des [services](#) permettant la gestion des enregistrements dans la BD tel que la persistance et la consultation.

La méthode permettant d'accéder à ce service est la suivante :

```
$this->get('doctrine');  
  
$this->getDoctrine(); // helper (raccourcie de la classe Controller)
```

A partir de ce service Doctrine, nous pouvons accéder à l'[EntityManager](#) qui comme son nom l'indique va manager nos entités. Cette interface ORM offre des [méthodes prédefinies](#) pour [persister](#) dans la base ou pour [chercher, mettre à jour ou supprimer](#) une entité.

```
$EntityManager = $this->get('doctrineorm.entity_manager')
```

```
$this->getDoctrine().getManager(); // helper (raccourcie de la classe Controller)
```

Remarque il y a aussi la méthode `getEntityManager()` mais elle est devenu obsolète

Une fois le manager instancié, deux catégories de traitement se distinguent, la [modification, la suppression et la persistance](#) d'un côté, et la [recherche](#) de l'autre. Pour

la recherche, nous avons besoin d'associer l'entité à traiter à notre `EntityManager` en récupérant le `Repository` qui lui est associé.

III.3 Ajout Modification, Suppression dans la BD

Etant un ORM, Doctrine traite les objets PHP. Donc l'enregistrement se fera automatiquement sur un objet qui est l'entité. Pour enregistrer des données dans la BD il faut préparer les objets contenant ces données-là.

La méthode `persist()` de l'`EntityManager` permet d'associer les objets à persister avec Doctrine. Afin d'exécuter les requêtes sur la BD (enregistrer les données dans la base) il faut utiliser la méthode `flush()` qui permet de « commiter » l'ensemble des actions que nous avons persister.

L'utilisation de `flush` permet de profiter de la force de Doctrine qui utilise les Transactions. La persistance agit de la même façon avec l'ajout (`insert`) ou la mise à jour (`update`).

Afin de persister un objet (l'ajouter ou associer la modification), nous utilisons donc la méthode `persist()` de l'`EntityManger`.

Pour la suppression, nous utilisons la méthode `remove()`.

Pour « commiter » les actions effectuées, nous utilisons la méthode `flush()` de l'`EntityManager`.

III.4 La recherche dans la BD : Les repositories (dépôts)

Les repositories sont des classes PHP dont le rôle est de permettre à l'utilisateur de récupérer des entités d'une classe donnée.

Syntaxe :

Pour accéder au repository de la classe MaClasse on utilise l'`EntityManager`

```
$repo = $EntityManager->getRepository('Bundle:MaClasse');
```

Exemple :

```
$em = $this->getDoctrine()->getManager();
```

```
$repository = $em->getRepository('FormationFormationBundle:Enseignant');
```

Quelques méthodes offertes par le repository :

1) `$repository->findAll();` // récupère tous les entités (enregistrements) relatifs à l'entité associé au repository

2) `$repository->find($id);` // requête sur la clé primaire

- 3) \$repository->findBy(array \$criteria, array \$orderBy = null, \$limit = null, \$offset = null); // retourne un ensemble d'entités avec un filtrage sur plusieurs critères (nbre donné)
- 4) \$repository->findOneBy(); // même principe que findBy mais une seule entité
- 5) \$repository->findByNomPropriété(); \$repository->findOneByNomPropriété();

Exemples :

- 1) Pour récupérer tous les Personnes

```
$listPersonne = $repository->findAll();
```

Généralement le tableau obtenu est passé à la vue (TWIG) et est affiché en utilisant un foreach

- 2) Pour récupérer la Personne d'id 2 : \$personne = \$repository->find(2);

3) [findBy\(\)](#) a un comportement assez particulier. En effet il retourne l'ensemble des entités qui correspondent à l'entité associé au repository comme findAll sauf qu'elle permet d'effectuer un filtrage sur un ensemble de critères passés dans un tableau(Array). Elle offre la possibilité de trier les entités sélectionnées et facilite la pagination en offrant un nombre de valeur de retour.

```
$listePersonnes = $repository->findBy(array('nom' => 'sellaouti'), array('age' => 'desc'), 10, 0);
```

- 4) \$Perosnne = \$repository->findOneBy(array('prenom'=>'ahmed','age' => '24'));

```
5) $listePersonnes = $repository->findByAge(25);
```

Exercice :

Nous nous proposons maintenant à partir de la classe personne de créer un CRUD, permettant d'ajouter une personne, de la modifier, de la supprimer et enfin de faire quelques exemples de recherche.

La façon avec laquelle nous traitons le crud est loin d'être conventionnelle mais elle nous permettra de très bien pratiquer ce que nous avons vu jusqu'à maintenant.

1) Commencer par créer un contrôleur permettant l'ajout d'une personne. Les données seront transmises via la root. Gérer le routing la vue et le contrôleur associé. Une fois l'entité ajoutée, une page affichant la personne sera affichée avec un message de succès. L'ensemble des actions seront regroupées dans la classe PersonneController.

Remarque : seul les .jpeg, .jpg et .png sont acceptées.

L'âge devra être entre 10 et 99 ans

Vérifier la validité de ces données avec les contrôles sur les routes.

Si l'image n'est pas introduite dans le path prévoyez une valeur par défaut.

N'oubliez pas que le ‘.’ est traité comme le ‘/’ c'est un séparateur.

La Figure 4 introduit le résultat après l'exécution.

The screenshot shows a web browser window with the URL `127.0.0.1/forminsat/web/app_dev.php/addPers/sellaouti/aymen/33/as.jpg`. The page title is "SB Admin". On the left, there's a sidebar with links: "Dashboard", "Charts", "Tables", "Forms", "A Typography", and "Bootstrap Elements". The main content area is titled "Mon CV" and displays a message: "Bonjour nous sommes le Thu/Aug/2015 Votre entité a été ajouté avec succès :)" followed by a user profile picture of a man, the name "aymen sellaouti", and the age "33 ans". There are also "Email" and "Back" buttons.

Figure 4 : Ajout d'une entité

La figure 5 illustre le résultat de l'insertion de l'entité dans la base de données

The screenshot shows the "phpMyAdmin" interface. The left sidebar lists databases and tables: "Nouvelle base de données", "animaux", "animaux2", "ecodocf", "forminsat", "formation", "information_schema", "mysql", "performance_schema", and "test". The "personne" table under "forminsat" is selected. The main panel shows a SQL query: "SELECT * FROM `personne`" and a results grid with one row: id=1, nom="sellaouti", prenom="aymen", age=33, path="images/as.jpg". There are buttons for "Afficher", "Structure", "SQL", "Rechercher", "Insérer", and "Exporter".

Figure 5 : L'entité insérée dans la base de données

- 2) Passez maintenant à la mise à jour. Nous gardons le même principe de passage des variables à travers la root sauf que pour la mise à jour, vous devez ajouter l'id de la personne à modifier.

La Figure 6 présente le résultat de l'exécution de la mise à jour de la personne d'id 1 avec une modification de l'âge.

The screenshot shows a web application interface. The URL in the browser is 127.0.0.1/forminsat/web/app_dev.php/updatePersonne/1/ellaouti/aymen/25/as.jpg. The page title is 'SB Admin'. On the left, there's a sidebar with 'Dashboard', 'Charts', 'Tables', 'Forms', 'A Typography', and 'Bootstrap Elements'. The main content area is titled 'Mon CV' and shows a success message: 'Bonjour nous sommes le Thu/Aug/2015 Votre entité a été ajouté avec succès :)'. It displays a circular profile picture of a man, the name 'aymen sellaouti', and the age '25 ans'. There are also 'Email' and 'Refresh' buttons.

Figure 6 : Modification d'une entité

La Figure 7 montre que la base de données a été bien modifiée.

The screenshot shows the 'phpMyAdmin' interface. The left sidebar lists databases like 'animaux', 'animaux2', 'ecodocf', 'forminsat', and 'test'. The 'personne' table under 'forminsat' is selected. The right panel shows the 'Structure' tab with a green checkmark indicating 'Affichage des lignes 0 - 0 (total de 1, Traitement en 0.0005 sec)'. Below it is a SQL query: 'SELECT * FROM `personne`'. A table below shows one row: id (1), nom ('sellaouti'), prenom ('aymen'), age (25), and path ('images/as.jpg'). There are buttons for 'Modifier', 'Copier', 'Effacer', 'Tout cocher', 'Pour la sélection :', 'Modifier', 'Effacer', and 'Exporter'.

Figure 7 : L'entité modifiée dans la base de données (l'âge est passé à 25 ans)

3) Insérer maintenant un ensemble de personne et vérifier les propriétés que vous avez ajouté à vos roots tels que les contraintes sur les extensions de l'image, l'âge qui doit osciller entre 10 et 99 ans et l'image par défaut qui doit être ajouté si l'image à insérer n'est pas trouvée.

4) Ajouter maintenant une page qui va afficher l'ensemble des personnes de la Base de données. La route devra s'appeler findAll. N'oublier pas la page 'cvs' que vous avez réalisé dans le tutoriel précédent elle vous sera très utile. Chaque personne affichée devra avoir son bouton suivant qui l'envoie vers sa page perso. N'oublier pas de transférer les paramètres dans le path. La figure 8 montre l'ensemble des personnes ajouté à la base de données de la Figure 9.

Figure 8 : Le résultat du findAll

Figure 9 : La base de données après l'insertion d'un ensemble de données (il y a l'image par défaut)

5) Tester les autres méthodes de recherche que nous avons introduites.

III.5 Création de requêtes

Les requêtes de doctrine sont écrite en utilisant le langage de doctrine [le Doctrine Query Language \(DQL\)](#) ou en utilisant un Objet créateur de requêtes le [CreateQueryBuilder](#).

Voici deux exemples illustrant la même requête.

[createQuery](#) : Méthode de l'Entity Manager

```
public function findAgeAction()
{
    $em=$this->getDoctrine()->getManager();
    $query=$em->createQuery(
        'Select p
         From InsatformaBundle:Personne p
         where p.age > :age
        ')>>setParameter('age','22');
    $personnes=$query->getResult();
    return $this->render('InsatformaBundle:Personne:cvs.html.twig', array(
        'personnes'=>$personnes,
    ));
}
```

[CreateQueryBuilder](#) : Méthode du répositorie

```
public function findAge2Action()
{
    $em=$this->getDoctrine()->getManager();
    $repository=$em->getRepository('InsatformaBundle:Personne');
    $query=$repository->createQueryBuilder('p')
        ->where('p.age> :age')
        ->setParameter('age','20')
        ->getQuery();
    $personnes=$query->getResult();
    return $this->render('InsatformaBundle:Personne:cvs.html.twig', array(
        'personnes'=>$personnes,
    ));
}
```

Nous remarquons que la syntaxe n'est pas éloignée l'une de l'autre.

Le langage DQL est explicité dans le lien suivant :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html>

Pour le queryBuilder, vous pouvez consulter le lien suivant :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>

Afin de suivre les bonnes pratiques Symfony2, les requêtes sont externalisées dans un repository spécifique à l'entité traitée. Ce traitement a pour but d'isoler la couche modèle. Il permet aussi la réutilisabilité des requêtes définies.

Pour ce faire, il faut ajouter le dépôt dans le mapping de l'entité puis dans le dépôt définir les méthodes à appeler.

Maintenant il suffit de faire appel à votre méthode à partir du repository.

Exercice

Reprenez la méthode de recherche par âge et externaliser la en mettant l'âge comme paramètre de la fonction.

III.5 Relations entre les entités

La gestion des relations entre les différentes entités de la base de données est une partie très importante de toute application. Pour ce faire, Doctrine permet de modéliser les relations avec l'utilisation des annotations.

Les trois relations entre les entités sont les suivantes :

- ✓ A **OneToOne** B : à une entité A on associe une entité de B et inversement
- ✓ A **ManyToOne** B : à une entité B on associe plusieurs entités de A et à une entité de A on associe une entité de B
- ✓ A **ManyToMany** B : à une entité de A on associe plusieurs entités de B et inversement

Les relations entre deux entités peuvent être unidirectionnelles ou bidirectionnelles.

La notion de navigabilité d'UML est la source de la notion de relation unidirectionnelle ou bidirectionnelle. Une relation est dite navigable dans les deux sens si les deux entités doivent avoir une trace de la relation.

Exemple : Supposons que nous avons les deux classes CandidatPrésidentielle et Electeur.

L'électeur doit savoir à qui il a voté donc il doit sauvegarder cette information par contre le candidat pour cause d'anonymat de vote ne doit pas connaître les personnes qui ont voté pour lui.

On aura donc un attribut Candidat dans la table Electeur mais pas de collection ou tableau nommé électeur dans la table CandidatPrésidentielle. Ici on a une relation unidirectionnelle.

Prenons l'exemple suivant afin de traiter les 3 associations :

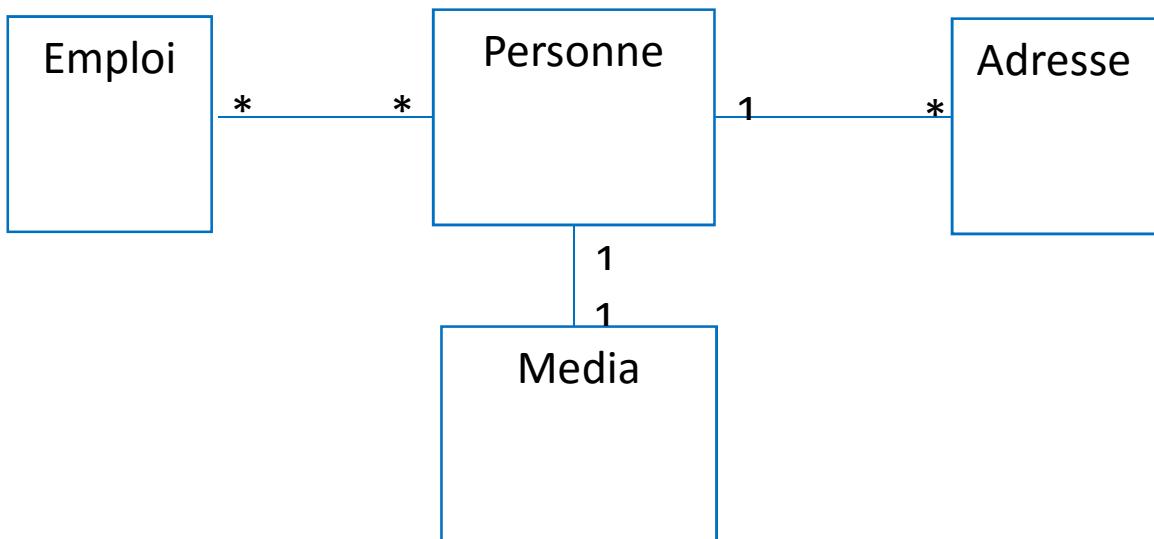


Figure 10 : Modèle contenant les 3 relations

Les relations de ce modèle :

- ✓ Adress ManyToOne Personne
- ✓ Personne OneToOne Media
- ✓ Personne ManyToMany Emploi

Relation OneToOne unidirectionnelle :

Relation unidirectionnelle puisque Media ne référence pas Personne.

Syntaxe :

```

/** 
 * @OneToOne(targetEntity= « EntitéCible»)
 * @JoinColumn(name= « labelDeLaJointure», referencedColumnName=«champsDeJointure»)
 */
  
```

Ici, l'annotation @joinColumn n'est pas obligatoire vu que par défaut il va référencer l'id de la table avec laquelle il a une jointure

Pour notre exemple :

```
Class Personne
{
    // ...
    /**
     * @ORM\OneToOne(targetEntity= "Insat\formaBundle\Entity\Media")
     * @ORM\JoinColumn(name= "Media_id",referencedColumnName="id")
     */
    private $media;
    // ...
}
```

Relation ManyToOne bidirectionnelle :

Relation bidirectionnelle puisque Personne va aussi référencer Adresse. Si nous voulons connaitre dans l'objet Personne l'ensemble de ses adresses qui lui sont alors nous devons avoir une relation bidirectionnelle

Adresse aussi doit référencer Etudiant ce qui fait que nous aurons une relation OneToMany côté Personne puisqu'à « One » Personne on a « Many » Adresses.

Nous devons ajouter l'attribut **mappedBy** côté OneToMany et **inversedBy** côté ManyToOne

Nous devons spécifier dans le constructeur du OneToMany, à savoir Personne, que l'attribut mappé est de type ArrayCollection en l'instanciant afin d'avoir une liste qui permettra de gérer les adresses de la personne.

L'annotation @joinColumn n'est pas obligatoire ici vu que par défaut il va référencer l'id de la table avec laquelle il a une jointure

Syntaxe :

```
/**
 * @ORM\ManyToOne(targetEntity= «EntitéCible»)
 * @ORM\JoinColumn(name=«lablIDJointure»,referencedColumnName=«champDeJointure»)
 */
```

Et dans l'autre classe :

```
/**
 * @ORM\OneToMany(targetEntity= «EntitéCible, mappedby(«AttribMappéDsL'autreClass»)
 */
```

Pour notre exemple :

Classe OneToMany (Personne)

```
Class Personne
{
//...
/***
 * @ORM\OneToMany(targetEntity="Insat\formaBundle\Entity\Adresse",mappedBy="personne")
 */
private $adresses;

function __construct()
{
    $this->adresses = new ArrayCollection();
}
//..
}

Classe ManyToOne (Personne)
```

Class Adresse

```
{
//...
/***
 * @ORM\ManyToOne(targetEntity="Insat\formaBundle\Entity\Personne", inversedBy="adresses")
 * @ORM\JoinColumn(name="personne_id",referencedColumnName="id")
 */
private $personne;
//..
}
```

Relation ManyToMany

Cette relation peut être uni ou bidirectionnelles selon le besoin. Si nous ne voulons pas connaitre les personnes associées à un emploi donné alors elle sera unidirectionnelle, sinon elle sera bidirectionnelle.

Nous optons pour le cas de la relation bidirectionnelle.

Class Personne

```
{
//....
/***
 * @ORM\ManyToMany(targetEntity="Insat\formaBundle\Entity\Emploi" ,
inversedBy="personnes")
 * @ORM\JoinColumn(nullable=false)
*/
private $emplois;
```

// N'oubliez pas d'associer un ArrayCollection dans le constructeur à l'attribut emplois
}

```

Class Emploi
{
//....
 /**
 * ORM\ManyToMany(targetEntity="InsaformaBundle\Entity\Personne",mappedBy="emplois")
 */
private $personnes;

// N'oubliez pas d'associer un ArrayCollection dans le constructeur à l'attribut personnes
}

```

Remarque : Pour générer les getters et setters essentiels à toutes les classes utiliser la commande : [php app/console generate:doctrine:entities InsaformaBundle:EntityName](#)

Le schéma final après l'ajout des relations est illustré par la Figure 11.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
adresse	Afficher Structure Rechercher Insérer Vider Supprimer	~0	InnoDB	utf8_unicode_ci	32 Kio	-
emploi	Afficher Structure Rechercher Insérer Vider Supprimer	~0	InnoDB	utf8_unicode_ci	16 Kio	-
media	Afficher Structure Rechercher Insérer Vider Supprimer	~0	InnoDB	utf8_unicode_ci	16 Kio	-
personne	Afficher Structure Rechercher Insérer Vider Supprimer	~0	InnoDB	utf8_unicode_ci	32 Kio	-
personne_emploi	Afficher Structure Rechercher Insérer Vider Supprimer	~0	InnoDB	utf8_unicode_ci	48 Kio	-
5 table(s)	Somme	~0	InnoDB	latin1_swedish_ci	144 Kio	0.0

Figure 11 : Schéma de la BD après applications des relations

Nous remarquons la création automatique de la table personne_emploi générée à cause de la relation ManyToMany entre les entités Personne et Emploi.

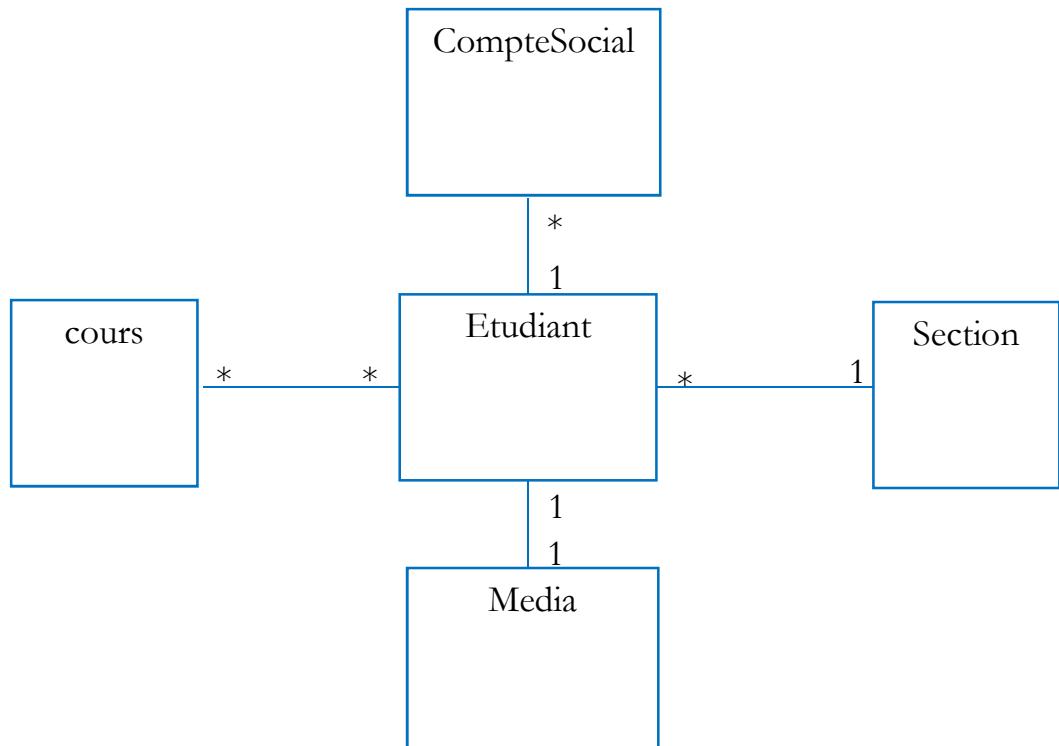
La Figure 12 montre l'ajout de l'attribut Media_id dans la table Personne.

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	id	int(11)			Non	Aucune	AUTO_INCREMENT	
2	nom	varchar(255)	utf8_unicode_ci		Non	Aucune		
3	prenom	varchar(255)	utf8_unicode_ci		Non	Aucune		
4	age	int(11)			Non	Aucune		
5	path	varchar(100)	utf8_unicode_ci		Non	Aucune		
6	Media_id	int(11)			Oui	NULL		

Figure 12 : Migration de l'id dû à la relation OneToOne

Exercice

Générer les tables et les relations de ce schéma. Choisissez les attributs qui vous semblent pertinents



Atelier Symfony2 : Les formulaires

OBJECTIF :

Le but de cet atelier est de comprendre et manipuler les formulaires de Symfony2 à travers l'Objet FORM.

Références :

- <http://symfony.com/>
- <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/>
- <http://www.doctrine-project.org/>

Livre

- Symfony, The Book, SensioLabs Version Aout2015.

I Introduction

Les formulaires représentent un composant essentiel pour les applications web. Il représente une vitrine et une interface entre l'application ou le site Web et ces utilisateurs. L'utilisation standard du formulaire se fait à travers la balise **form**. Symfony2 a révolutionné l'utilisation des formulaires à travers son composant FORM qui est une bibliothèque dédiée aux formulaires.

II Modélisation des formulaires avec Symfony

La philosophie de Symfony2 pour les formulaires est la suivante :

- ✓ Un formulaire est l'image d'un objet existant
- ✓ Le formulaire sert à alimenter cet objet.

La figure 1 illustre la modélisation des formulaires de point de vue symfony2.

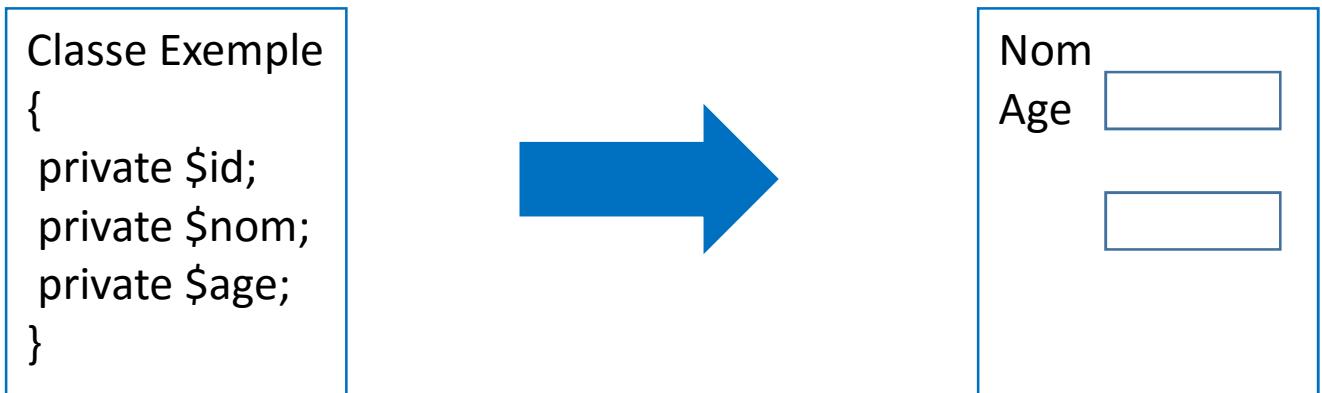


Figure 1 : Modélisation des formulaires de point de vue Symfony2

III Création de formulaire

La création d'un formulaire se fait à travers le Constructeur de formulair [FormBuilder](#)

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($objetImage)
```

Pour indiquer les champs à ajouter au formulaire on utilise la méthode [add](#) du FromBuilder. Cette méthode contient 3 paramètres :

- 1) le nom du champ dans le formulaire
- 2) le type du champ
- 3) un array qui contient des options spécifiques au type du champ

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)->add('nom','text')
                                         ->add('age','integer')
```

Pour récupérer le formulaire créé, il faut utiliser la méthode [getForm\(\)](#)

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)->add('nom','text')
```

```
->add('age','integer')  
->getForm();
```

La création du formulaire se fait de 2 façons différentes :

- 1) Dans le contrôleur qui va utiliser le formulaire
- 2) En externalisant la définition la définition dans un fichier

IV Crédit de formulaire

Afin d'afficher le formulaire créé, il faut transmettre la vue de ce formulaire à la page twig qui doit l'accueillir. Ceci se fait à travers la méthode `createView` de l'objet `Form`.

Il ne reste plus qu'à l'envoyer à la page twig en question

Exemple :

```
$form= $this->createFormBuilder($exemple)->add('nom','text')  
        ->add('age','integer')  
        ->getForm();  
return $this->render('Rt4AsBundle:Default:myform.html.twig',array(  
    'form'=>$form->createView()  
))  
);
```

Deux méthodes permettent d'afficher le formulaire dans Twig :

- 1) Afficher directement la totalité du formulaire avec la méthode `form`

```
{ { form(nomDuFormulaire) } }
```

- 2) Afficher les composants du formulaire séparément un à un (généralement lorsqu'on veut personnaliser les différents champs)

IV.1 Les composants du formulaire

Les composants du formulaire FORM sont :

`form_start()` affiche la balise d'ouverture du formulaire HTML, soit `<form>`. Il faut passer la variable du formulaire en premier argument, et les paramètres en deuxième argument. L'index attr des paramètres, et cela s'appliquera à toutes les fonctions suivantes, représente les attributs à ajouter à la balise générée, ici le `<form>`. Il nous permet d'appliquer une classe CSS au formulaire, ici `form-horizontal`.

Exemple : {{ form_start(form, {'attr': {'class': 'form-horizontal'}}) }}

`form_errors()` affiche les erreurs attachées au champ donné en argument. Nous verrons les erreurs de validation dans le prochain chapitre.

`form_label()` affiche le label HTML du champ donné en argument. Le deuxième argument est le contenu du label.

`form_widget()` affiche le champ HTML lui-même (que ce soit <input>, <select>, etc.).

Exemple : {{ form_widget(form.title, {'attr': {'class': 'form-control'}}) }}

`form_row()` affiche le label, les erreurs et le champ.

`form_rest()` affiche tous les champs manquants du formulaire (dans notre cas, juste le champ CSRF puisque nous avons déjà affiché à la main tous les autres champs).`form_end()` affiche la balise de fermeture du formulaire HTML, soit </form>.

Remarque : Certains types de champ ont des options d'affichage supplémentaires qui peuvent être passées au widget. Ces options sont documentées avec chaque type, mais l'option `attr` est commune à tous les types et vous permet de modifier les attributs d'un élément de formulaire.

IV.2 Gestion de la soumission des Formulaires

La gestion de la soumission des formulaires se fait à l'aide de la méthode `handleRequest($request)`.

L'action qui doit faire appel à cette méthode doit avoir comme paramètre l'objet `Request`. N'oublier donc pas le « `use Symfony\Component\HttpFoundation\Request` ».

`handleRequest` vérifie si la requête est de type `POST`. Si c'est le cas, elle va mapper les données du formulaire avec l'objet affecté au formulaire en utilisant les setters de cet objet.

IV.3 Externalisation de la définition des formulaires

Afin de rendre les formulaires réutilisables, Symfony permet l'externalisation des formulaires en des objets.

Par convention de nommage, l'objet du formulaire doit être nommé comme suit `NomObjetType`. Par exemple `PersonneType`.

Cet objet doit obligatoirement étendre la classe `AbstractType`. Deux méthodes doivent obligatoirement être implémentées :

- `buildForm(FormBuilderInterface $builder, array $options)` qui est la méthode qui va permettre la création et la définition du formulaire
- `getName()` qui retourne un identifiant unique pour le formulaire. Par convention c'est le nom du bundle et le nom de la méthode séparé avec des ‘_’ et sans majuscule
- Il y a aussi la méthode `setDefaultOptions()` qui est facultative et qui permet de définir l'objet associé au formulaire

Doctrine permet d'automatiquement générer la classe du formulaire spécifique à une entité en utilisant la commande suivante :

```
php app/console doctrine:generate:form BundleName:EntiyName
```

Exemple :

```
php app/console doctrine:generate:form Rt4AsBundle:Tache
```

La récupération du formulaire au niveau des contrôleurs devient beaucoup plus facile :

```
$form = $this->createForm(new TacheType(), $tache);
```

Voici un exemple de traitement d'un formulaire :

Fichier TacheType :

```
<?php

namespace Rt4\AsBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class TacheType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->setMethod('POST')
            ->add('matache','choice', array(
                'choices'=>array('choix1' => 'matache1', 'choix2' => 'matache2', 'choix3' =>
'matache3'),
                'preferred_choices'=> array('choix2'),
                'expanded'=> false,
            ))
            ->add('etudiant', 'entity', array(
                'class' => 'Rt4AsBundle:Etudiant',
                'property' => 'nom',
                'expanded'=> true,
            ))
            ->add('date')
            ->add('avatar','file',array(
                'data_class' => null
            ))
            ->add('envoyer','submit') ;
    }

    /**
     * @param OptionsResolverInterface $resolver
     */
    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'Rt4\AsBundle\Entity\Tache'));
    }

    /**
     * @return string
     */
    public function getName()
    {
        return 'rt4_asbundle_tache';
    }
}
```

Action testFormAction qui traite le fichier

```
public function testFormAction(){
    $tache = new Tache();
    $form = $this->createForm(new TacheType(),$tache);
    // ici on va activer le handle request
    $form->handleRequest($request);
    // si le formulaire est valide et si la méthode est POST on traite l'entité sinon c'est le
    // premier accès via un GET et on affiche le formulaire
    if($form->isValid())
    {
        $em=$this->getDoctrine()->getManager();
        $em->persist($tache);
        $em->flush();
        return $this->render('Rt4AsBundle:Default:successTache.html.twig');
    }
}
return $this->render('Rt4AsBundle:Default:myform.html.twig',array(
    'form'=>$form->createView()));
}
```

La vue qui va afficher le formulaire est la suivante :

```
<html>
<body>
<form action="{{ path('rt4_add_Form') }}" method="post" {{ form_enctype(form) }}>
    {{ form_widget(form) }}
</form>
</body>
</html>
```

Nous remarquons ici que pour les attributs **action** et **method** du formulaire ont été spécifié dans le fichier TWIG. Nous aurons pu le faire dans le TacheType sauf que ceci nécessite certaines règles.

Ne pouvons pas accéder dans la classe AbstractType à la méthode generateUrl afin de modifier l'action du formulaire, il faut donc procéder ainsi :

- Générer l'url au niveau du formulaire
- Passer l'url dans le troisième paramètre optionnel de la méthode createForm
- Récupérer l'url dans l'objet du formulaire dans le tableau \$options via son label

L'exemple des figures 3 et 4 illustre les changements à faire par rapport au premier exemple.

```
$form = $this->createForm(new TacheType(), $tache, array(
    'action' => $this->generateUrl('rt4_add_form')
));
;
```

Figure 2 : Passage du path dans le paramètre options

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->setAction($options['action'])
        ->setMethod('POST')
        ->add('matache')
        ->add('date')
    ;
}
```

Figure 3 : Utilisation du path de l'option dans le builder

V Propriétés d'un champ dans le formulaire

Les champs du formulaire sont paramétrable à travers le troisième paramètre de la méthode add est un tableau d'options pour les attributs du formulaire. Parmi les options communes à la majorité des champs nous citons :

- **label** : pour le label du champ si l'attribut **option** n'est pas mentionné alors le label sera le nom du champ.
- **required** : Permet de dire si le champ est obligatoire ou non (Par défaut l'option required est défini à true)

Les formulaires sont composés d'un ensemble de champs. Chaque champ possède un nom, un type et des options. Symfony propose une grande panoplie de types de champ introduite dans le tableau I.

Texte	Choix	Date et temps	Divers	Multiple	Caché
text	choice	date	checkbox	collection	hidden
textarea	entity	datetime	file	repeated	csrf
email	country	time	radio		
integer	language	birthday			
money	locale				
number	timezone				
password					
percent					
search					
url					

Tableau 1 : Les différents types des champs du FORM

V.1 Le type choice

Ce type est très important, il est spécifique aux champs optionnels (select, boutons radio, checkboxes) qui sont très utilisés. Pour spécifier le type d'options qu'on veut avoir il faut utiliser le paramètre `expanded`. S'il est à `false` (valeur par défaut) alors nous aurons une [liste déroulante](#). S'il est à `true` alors nous aurons [des boutons radio ou des checkbox](#) qui dépendront du paramètre `multiple`

Les différentes combinaisons de ces deux paramètres nous donnent le résultat introduit par le Tableau 2 <http://symfony.com/fr/doc/current/reference/forms/types/choice.html>.

Balise HTML	expanded	multiple
Liste déroulante	false	false
Liste déroulante (avec attribut <code>multiple</code>)	false	true
Boutons radio	true	false
Cases à cocher	true	true

Tableau 2 : Effet de l'utilisation des attributs `expanded` et `multiple` sur le type choice

La Figure 4 illustre différentes utilisations du type `choice` avec le paramètre `expanded`.

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->setAction($options['action'])
        ->setMethod('POST')
        ->add('matache', 'choice', array(
            'choices'=>array('choix1' => 'matache1', 'choix2' => 'matache2', 'choix3' => 'matache3'),
            'preferred_choices' => array('choix2'),
            'expanded'=>'false',
        ))
        ->add('date')
    ;
}
```

Matache

matache2 matache1 matache3

Date

Aug ▾ 21 ▾ 2015 ▾

Valider

Expanded=true

Matache matache2 ▾

Date matache2

Aug ▾ matache1

Valider matache3

Expanded=false

Figure 4 : illustration du type choice

V.2 Le type entity

C'est une extension très intéressante et très pratique du type `choice`. En effet, les choices (les options) seront chargés à partir des éléments d'une entité Doctrine.

L'exemple de la Figure 5 introduit l'utilisation de ce type. Dans le tableau d'options de ce type nous devons préciser la ‘class’ qui va être chargée ainsi que la ‘property’ à afficher.

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('nom')
        ->add('duree')
        ->add('enseignant', 'entity', array(
            'class' => 'FormationFormationBundle:Enseignant',
            'property' => 'nom',
        ))
}
```

Figure 5 : illustration du type entity

V.3 Le type country

C'est une extension très sympathique qui affiche l'ensemble des pays monde. La langue d'affichage est celle de la locale (config.yml). La figure 6 illustre l'utilisation de ce type.

Exemple :

```
->add('pays','country')
```

The screenshot shows a Symfony form interface. At the top, there are several input fields: a file upload field labeled 'Matache' with a dropdown arrow, a date field labeled 'Etudiant' with a date range from 'Jan 1 2010', and a file selection field labeled 'Avatar' with the placeholder 'Choisissez un fichier'. Below these is a dropdown menu labeled 'Pays' which is currently set to 'Afghanistan'. A scroll bar is visible on the right side of the dropdown menu, indicating it contains a large list of countries. The list includes: Afghanistan, Aland Islands, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antarctica, Antigua & Barbuda, Argentina, Armenia, Aruba, Ascension Island, Australia, Austria, Azerbaijan, Bahamas, Bahrain.

Figure 6 exemple de l'utilisation de l'attribut Country

V.4 Le type file

Le type file permet l'upload de n'importe quel type de fichier

(<http://symfony.com/fr/doc/current/reference/forms/types/file.html>)

Le champ permet de récupérer un objet de type file contenant le path de l'objet à uploader. Pour pouvoir gérer cet objet, il faut le copier dans le répertoire web de votre projet et de préférence dans un fichier upload. Ensuite, attribuer un nom unique à votre fichier pour ne pas avoir de problème lors de l'ajout de fichier ayant le même nom (vous pouvez utiliser la méthode suivante md5(uiniqueid())).

Pour déplacer votre fichier utiliser la méthode move(\$pathsrc,\$pathdest) de votre objet file

__DIR__ vous donne le path de l'endroit à partir duquel vous faites appel à cette méthode (ici votre controller).

Exercice :

Nous projetons dans cet exercice de modéliser les opérations de CRUD du modèle conceptuel de la Figure 7.

Créer les entités et ajouter les relations entre ces derniers.

Créer la base de données et les tables.

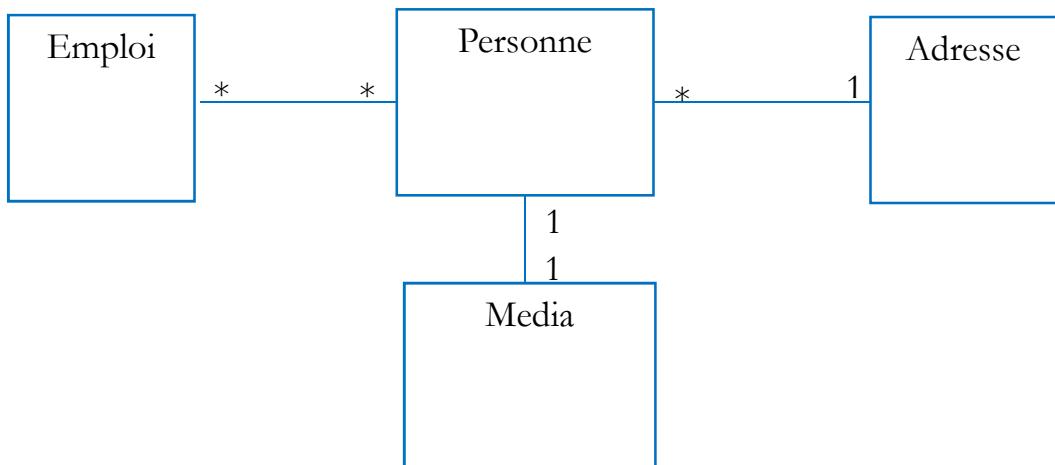


Figure 7 : Modèle conceptuel

Pour chaque entité crée le CRUD correspondant. La page de gestion de chaque entité devra afficher la liste des éléments de la base de données et pour chaque enregistrement les opérations susceptible d'être effectuée sur lui à savoir la modification la suppression ou l'aperçu. La Figure 8 illustre un exemple de gestion sur l'entité Personne. Utilisez le tableau offert dans le sb-admin.

La capture d'écran montre l'interface de gestion des personnes dans l'application SB Admin. Le menu latéral gauche contient les options : Dashboard, Charts, Tables, Forms, Typography, Bootstrap Elements, Bootstrap Grid, et Blank Page. La partie centrale affiche le titre "Gestion des personnes" et le sous-titre "Liste des personnes". Un tableau liste les personnes avec les colonnes : Nom, Prenom, Age, Emplois, et Action. Chaque ligne du tableau contient des boutons pour "Modifier", "Supprimer" et "Aperçu".

Nom	Prenom	Age	Emplois	Action
as	as	24	enseignant Ingénieur	Modifier Supprimer Aperçu
sellaouti	aymen	33	enseignant Ingénieur	Modifier Supprimer Aperçu
ieee	student	24	Pas d'emploi	Modifier Supprimer Aperçu
ieee	student	24	Pas d'emploi	Modifier Supprimer Aperçu
ieee	student	24	Pas d'emploi	Modifier Supprimer Aperçu
ieee	student	24	Pas d'emploi	Modifier Supprimer Aperçu
ieee	student	24	Pas d'emploi	Modifier Supprimer Aperçu
ieee	student	24	Pas d'emploi	Modifier Supprimer Aperçu

Figure 8 Gestion des personnes

VI Les validateurs de formulaire

Afin de pouvoir utiliser les annotations de validation il faut importer la class Constraints
use Symfony\Component\Validator\Constraints as Assert;

Syntaxe :

@Assert\MaContrainte(option1="valeur1", option2="valeur2", ...)

Exemples :

@Assert\NotBlank(message = " Ce champ ne doit pas être vide ")

@Assert\Length(min=4, message="Le login doit contenir au moins {{ limit }} caractères.")

@Assert\Url()

Afin de tester vos validateurs, désactiver la validation HTML.

{{ form(form, { 'attr' : { 'novalidate' : 'novalidate' } }) }} ou en ajoutant novalidate à
<form>

Voici les différentes contraintes classé selon le site
openclassrooms <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/validez-vos-donnees> :

Les contraintes de base :

Contrainte	Rôle	Options
NotBlank Blank	La contrainte NotBlank vérifie que la valeur soumise n'est ni une chaîne de caractères vide, ni NULL. La contrainte Blank fait l'inverse.	-
True False	La contrainte True vérifie que la valeur vaut true, 1 ou "1". La contrainte False vérifie que la valeur vaut false, 0 ou "0".	-

<u>NotNull</u> Null	La contrainte NotNull vérifie que la valeur est strictement différente de null.	-
Type	La contrainte Type vérifie que la valeur est bien du type donné en argument.	type (option par défaut) : le type duquel doit être la valeur, parmi array, bool,int, object

Nombre et date

Contrainte	Rôle	Options
Range	La contrainte Range vérifie que la valeur ne dépasse pas X, ou qu'elle dépasse Y.	min : nbre de car minimum max : nbre de car maximum minMessage : msg erreur nbre de car min maxMessage : msg erreur nbre de car max invalidMessage : msg erreur si non nmbre
Date	vérifie que la valeur est un objet de type Datetime, ou une chaîne de type YYYY-MM-DD.	-
Time	vérifie que c'est un objet de type Datetime, ou une chaîne type HH:MM:SS.	-
DateTime	vérifie que c'est un objet de type Datetime, ou une chaîne de caractères du type YYYY-MM-DD HH:MM:SS.	-

Les validateurs de File

Contrainte	Rôle	Options
File	La contrainte File vérifie que la valeur est un fichier valide, c'est-à-dire soit une chaîne de caractères qui pointe vers un fichier existant, soit une instance de la classe File (ce qui inclut UploadedFile).	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. mimeType(s) : mimeType(s) que le fichier doit avoir.
Image	La contrainte Image vérifie que la valeur est valide selon la contrainte précédente File (dont elle hérite les options), sauf que les mimeType acceptés sont automatiquement définis comme ceux de fichiers images. Il est également possible de mettre des contraintes sur la hauteur max ou la largeur max de l'image.	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. minWidth / maxWidth : la largeur minimale et maximale que doit respecter l'image. minHeight / maxHeight : la hauteur minimale et maximale que doit respecter l'image.

Exercice

Désactiver la validation HTML5 sur vos formulaires et mettez vos propres validateurs. Le Media devra avoir des formats et une taille max. Limiter l'intervalle de l'âge. Gérer la taille du nom, prenom et designation. Les champs ne doivent pas être vides. Ajouter d'autres validateurs.