

Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Mettre en œuvre une solution
e-commerce »

Développer des Gabarits de Mise en Page
PrestaShop

Apprentissage

Mise en pratique

Evaluation

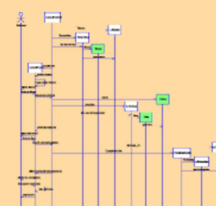
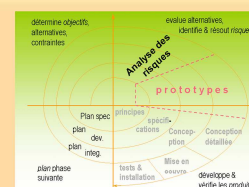
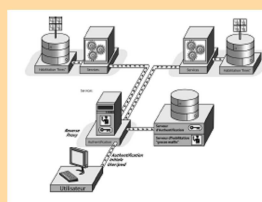


TABLE DES MATIERES

| | |
|---|----|
| Table des matières | 2 |
| 1. Introduction | 5 |
| 2. Architecture générale de PrestaShop..... | 5 |
| 2.1 Noyau PrestaShop et templates Smarty..... | 5 |
| 2.2 Structure d'une boutique PrestaShop..... | 6 |
| 3. Les templates Smarty | 8 |
| 3.1 Découverte..... | 8 |
| 3.2 Le langage Smarty | 9 |
| 3.3 La passation de variables depuis PHP | 10 |
| 4. contrôleurs, modules et templates PrestaShop | 11 |
| 4.1 Notion de surcharge de contrôleur | 11 |
| 4.2 Modifier le template associé à un contrôleur | 12 |
| 4.3 Structure d'un module standard Prestashop..... | 13 |
| 4.4 Modifier le gabarit associé à un module | 15 |
| 4.4.1 Intervenir sur le bon gabarit | 15 |
| 4.4.2 Exemple guidé à réaliser : | 15 |
| 4.5 Création de nouveaux gabarit | 17 |
| 5. En guise de synthèse sur les gabarits de mise en page PrestaShop | 18 |

Objectifs

A l'issue de cette séance, le stagiaire sera capable de comprendre le fonctionnement des contrôleurs, modules et gabarits PrestaShop ; il sera capable d'intervenir sur des modèles de présentation existants afin de modifier/personnaliser un page ou portion de page d'une boutique PrestaShop.

Concernant le système de template Smarty, l'objectif se limite à la compréhension du système et à la découverte des principales instructions du langage Smarty.

Pré requis

Avoir pris en main le Front-office et le Back-office de PrestaShop.

Maîtriser les techniques de base du développement Web, HTML, CSS et programmation PHP orientée objet.

Outils de développement

Un navigateur doté d'un débogueur complet.

Un éditeur de code source non-WYSIWYG, adapté au développement en techniques Internet (NotePad ++, CodeLobster, Brackets, Eclipse...).

Méthodologie

Ce support propose une première approche dans la problématique de la maintenance et du développement pour PrestaShop, en se focalisant sur les aspects de présentation des résultats via le système de templates Smarty.

Les apports théoriques sont illustrés et mis en pratique sur des composants très standards d'une boutique PrestaShop.

Ce document complète la documentation de référence de PrestaShop en Français (disponible en ligne sur <http://www.prestashop.com>).

NB : les apports donnés ici se basent sur la version 1.6.xx du logiciel.

Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

Ressources

La documentation de référence PrestaShop : <http://doc.prestashop.com>

Pour aller plus loin, la documentation de référence Smarty : <http://www.smarty.net/docsv2/fr/>

Lectures conseillées

1. INTRODUCTION

L'utilisation courante du Back-office PrestaShop permet de se familiariser avec la notion de module et d'en administrer certains directement dans les pages du Back-office. Quelques modules proposent un formulaire de paramétrage (message, couleur...) depuis le Back-office, ce qui permet alors de personnaliser l'affichage sans rentrer dans la programmation. C'est déjà un premier niveau de travail sur les modules, plus spécialement destiné aux non-informaticiens.

Si le besoin de personnalisation ne concerne par les traitements mais uniquement la présentation des résultats, l'informaticien développeur pourra intervenir sur le composant de présentation du module concerné, le '*template*', sans risque d'effet de bord sur le reste du module et sans risque de perturbation de la boutique. C'est tout l'intérêt de l'architecture '3 tiers' adoptée par les auteurs du logiciel.

Mais comment tout cela est-il agencé ? Où et comment intervenir sur la présentation de la boutique en tant que développeur ? C'est bien l'enjeu de la suite de ce document.

2. ARCHITECTURE GENERALE DE PRESTASHOP

2.1 NOYAU PRESTASHOP ET TEMPLATES SMARTY

Le cœur de PrestaShop est composé d'un ensemble complexe de classes PHP dont les classes 'Métier' qui représentent et gèrent les données (client, produit, catégorie...) et les classes 'Contrôleur' qui gèrent tous les traitements. Il s'agit bien là de code 100% PHP.

Pour assurer la présentation du Front-office comme du Back-office, c'est-à-dire pour générer le code final HTML/JavaScript/CSS à adresser au navigateur de l'utilisateur, il serait encore possible d'utiliser de simples scripts PHP classiques. Les auteurs de PrestaShop ont fait le choix d'un outil intermédiaire, Smarty, logiciel serveur capable de personnaliser des modèles de pages composés de code HTML/JavaScript/CSS et d'instructions en langage spécifique. Ces **modèles de mise en page**, ces **gabarits**, sont appelés '**templates**'. Au moment de servir une page, Smarty fusionne un ensemble de données spécifiques (des variables PHP) avec le gabarit (un peu comme le fait un traitement de textes dans une fonction de mailing) pour générer le code final à adresser au navigateur.

Le rendu final, par le navigateur, du code HTML reçu est assuré par un ensemble de **feuilles de styles CSS**. Les 'thèmes' PrestaShop installés définissent tous les styles utilisés par les pages standards du logiciel ; un thème est en fait essentiellement constitué de feuilles de styles CSS.

Au-delà de ces besoins standards, chaque module peut encore ajouter ses propres gabarits et feuilles de styles CSS.



Dans cette architecture '3-tiers', templates Smarty et scripts CSS constituent donc les '**vues**' du logiciel ; ils sont en général stockés dans des sous-dossiers '**views**'.

2.2 STRUCTURE D'UNE BOUTIQUE PRESTASHOP

Une boutique PrestaShop est constituée d'une base de données MySQL et d'une multitude de fichiers et dossiers stockés dans un dossier publié par le serveur Web :

| Nom | |
|---------------------|---|
| admin5134 | ← les composants du back-office |
| cache | |
| classes | ← les 'classes Métier' représentant les données PrestaShop (le 'Modèle' de MVC) |
| config | |
| controllers | ← les classes 'Contrôleur' assurant la production des pages de la boutique |
| css | ← les feuilles de styles globales (communes aux différents thèmes) |
| docs | |
| download | |
| img | ← les images globales de la boutique (communes aux différents thèmes) |
| js | ← les scripts JavaScript globaux (communs aux différents thèmes) |
| localization | |
| log | |
| mails | |
| modules | ← les composants des modules installés (contrôleur et vue de MVC) |
| override | les 'surcharges' de classes |
| pdf | |
| themes | ← les composants des thèmes installés |
| tools | |
| translations | |
| upload | |
| webservice | |
| .htaccess | |
| { } CONTRIBUTING.md | |
| { } CONTRIBUTORS.md | |
| error500.html | |
| footer.php | |
| header.php | |
| images.inc.php | |
| index.php | ← le 'contrôleur principal' qui distribue les traitements à réaliser |
| init.php | |
| modules.txt | |
| { } README.md | |
| sitemap.xml | |

⚡ Toutes les demandes de pages sont adressées au contrôleur principal (le 'dispatcher') `index.php` qui redirige la requête vers le bon contrôleur.

Chaque contrôleur PrestaShop se charge de générer une page-type de la boutique et il fait appel au code PHP (= contrôleur du modèle MVC) des différents modules greffés sur la page.

Développer des gabarits de mise en page - PrestaShop

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

Par exemple, l'URL suivante :

`http://localhost:8080/prestashop16/index.php?id_category=14&controller=category&id_lang=1`
appelle le contrôleur `CategoryController.php` avec les paramètres supplémentaires de langue et de numéro de catégorie demandée.

Le script `CategoryController.php` définit une **classe contrôleur** `CategoryControllerCore` (dérivée de la classe contrôleur de base pour Front-office) :

```
class CategoryControllerCore extends FrontController
{
```

Un contrôleur ne s'occupe que de la 'logique applicative', c'est-à-dire des traitements à effectuer et **fait le lien avec les composants de mise en page (les 'templates') et les éventuels scripts CSS et JavaScript** ; dans l'exemple ci-dessous, la méthode `setMedia()` fait le lien avec les CSS spécifiques à l'affichage des catégories.

Extrait de la classe `CategoryControllerCore` :

La méthode `setMedia()` fait le lien avec les feuilles de styles CSS nécessaires :

```
public function setMedia()
{
    parent::setMedia();


    if (!$this->useMobileTheme())
    {
        //TODO : check why cluetip css is include without js file
        $this->addCSS(array(
            _THEME_CSS_DIR_.'scenes.css' => 'all',
            _THEME_CSS_DIR_.'category.css' => 'all',
            _THEME_CSS_DIR_.'product_list.css' => 'all',
        ));
    }
}
```

Et la méthode `initContent()` fait le lien avec le template, ici stocké dans le dossier du thème courant :

```
public function initContent()
{
    parent::initContent();

    $this->setTemplate(_PS_THEME_DIR_.'category.tpl');
}
```

NB : Les templates Smarty ont par défaut une extension `.tpl`.

 Toute la présentation (vue de MVC) est reléguée dans les scripts du template et des feuilles de styles CSS.

3. LES TEMPLATES SMARTY

3.1 DECOUVERTE

PrestaShop a choisi, pour assurer la présentation des pages, le 'moteur de template' Smarty. Il s'agit d'un logiciel complémentaire (gratuit) installé sur le serveur Web qui génère le code final HTML/JavaScript/CSS à envoyer au navigateur à partir de modèles de pages (templates) et de variables PHP reçues par le template.

Pour assurer la fusion des variables dans le gabarit, Smarty propose un langage spécifique relativement simple, très proche du PHP, et assez puissant. Il faudra donc aussi se familiariser avec le fonctionnement et le langage Smarty pour intervenir sur les affichages fournis ou en créer de nouveaux ; en voici déjà les premiers principes illustrés par l'exemple des catégories.



Ouvrir (depuis le dossier d'un thème) et observer le code du template category.tpl.

On remarque immédiatement qu'il s'agit bien d'une partie du `<body>` d'une page HTML constituée d'éléments `<div>` imbriqués.

On identifie aisément les instructions en langage Smarty qui sont délimitées par des accolades.

Extrait de code du template category.tpl :

```
{include file="$tpl_dir./errors.tpl"}
{if isset($category)}
    {if $category->id AND $category->active}
        {if $scenes || $category->description || $category->id_image}
            <div class="content_scene_cat">
                {if $scenes}
                    <div class="content_scene">
                        <!-- Scenes -->
                        {include file="$tpl_dir./scenes.tpl" scenes=$scenes}
                        {if $category->description}
                            <div class="cat_desc rte">
                                {if Tools::strlen($category->description) > 350}
                                    <div id="category_description_short">{$description_short}</div>
                                    <div id="category_description_full" class="unvisible">{$category->description}</div>
                                    <a href="{ $link->getCategoryLink($category->id_category, $category->id_image) }">{$category->name}</a>
                                {else}
                                    <div>{$category->description}</div>
                                {/if}
                            </div>
                        {/if}
                    </div>
                {else}
                    <!-- Category image -->
                {/if}
            </div>
        {/if}
    {/if}
{/if}
```

Annotations :

- `{include file=...}` : inclusion d'un autre script
- `<div class="content_scene">` : il s'agit bien de générer du code HTML
- `{include file="$tpl_dir./scenes.tpl" scenes=$scenes}` : appel de script PHP
- `{if...} ... {else} {/if}` : test
- `{if Tools::strlen($category->description) > 350}` : utilisation de variables PHP (simples ou tableaux associatifs) ; ici, insertion du contenu de la variable
- `<!-- Category image -->` : Commentaire HTML qui aidera à se repérer dans le debugger du navigateur



Dans PrestaShop, la mise en page est assurée par une imbrication complexe d'éléments HTML `<div>`, bien souvent optionnels selon le paramétrage du Back-office. Chaque template ne s'occupe que de sa ou ses `<div>` spécifiques.

Plus loin, on peut remarquer une boucle en langage Smarty :

```
<ul class="clearfix">
{foreach from=$subcategories item=subcategory}
    <li>
        <div class="subcategory-image">
            <a href="{ $link->getCategoryLink... }" >
                {if $subcategory.id_image}
                    
        </div>
        <h5><a class="subcategory-name" href="{ $:
            {if $subcategory.description}
                <div class="cat_desc">{$subcategory
            {/if}
        </h5>
    </li>
{foreach}
</ul>
```

{foreach from=... item=...} ... {/foreach} :
boucle de parcours d'une liste/collection

Enfin, on peut remarquer le système de traduction Smarty similaire à celui de PrestaShop :

```
<!-- Subcategories -->
<div id="subcategories">
    <p class="subcategory-heading">{l s='Subcategories'}</p>
```

{l s='...'} : chaîne de caractères à traduire dans la
langue courante
NB : devrait contenir aussi le paramètre mod='...'

3.2 LE LANGAGE SMARTY

Pour résumer, un template Smarty :

- contient le **code HTML constant** à adresser au navigateur,
- contient des **instructions de contrôle** (test, boucle) en langage Smarty écrites entre **{...}**,
- peut être découpé en plusieurs scripts assemblés par l'instruction Smarty **{include file='...'} (écriture modulaire facilitant la maintenance, externalisation du code JavaScript) ;**
- peut manipuler des variables PHP,
- **insère très simplement le contenu d'une variable PHP { \$... }** (similaire à l'ordre PHP echo),
- peut faire appel à d'autres scripts PHP par appel direct,
- **peut inclure des chaînes de caractères à traduire** selon la langue courante de l'utilisateur **{ l s='...' mod='...' },**
- peut contenir du commentaire HTML (qui sera adressé au navigateur) **<!-- ... -->** ou même du commentaire Smarty spécifique (non adressé au navigateur) **{* ... *}**

Développer des gabarits de mise en page - PrestaShop

De plus cette variante de Smarty offre de nombreuses variables d'environnement adaptées aux données manipulées par PrestaShop (images de produits...) ainsi que des ordres de formatage des données ('filtres') ou d'aide à la mise au point. Par exemple :

- `{$maVariable | truncate:120 | escape:'htmlall'}` ne prend que les 120 premiers caractères de la variable et transforme les caractères spéciaux HTML comme le signe < ,
- `{$monTableau | @count}` restitue le nombre d'occurrences du tableau,
- `{$maVariable | @print_r}` et `{$maVariable | @debug_print_var}` insèrent des informations de débogage dans la page.
- `{debug}` insère dans le code HTML envoyé au navigateur l'ensemble des variables connues de la page en cours.

NB : La séance consacrée à la création de modules PrestaShop permettra de mettre en pratique le langage Smarty.

Pour aller plus loin dans l'étude du langage Smarty, on peut se référer à la documentation de référence (<http://www.smarty.net/docsv2/fr/>) au fur et à mesure des besoins lors de la réalisation de templates.

3.3 LA PASSATION DE VARIABLES DEPUIS PHP

Le moteur Smarty peut accéder aux variables globales PHP (et donc aux variables globales PrestaShop) mais il est préférable de transmettre les données sous forme de paramètres, depuis le contrôleur ou le module vers le template. **C'est le rôle de la méthode `assign()` de l'objet smarty du contexte des contrôleurs.** Ainsi dans la suite de la méthode `initcontent()` de l'exemple précédent, contrôleur `CategoryController.php` :

```
$this->context->smarty->assign(array(
    'category' => $this->category,
    'description_short' => Tools::truncateString($this->category->description, 350),
    'products' => (isset($this->cat_products) && $this->cat_products) ? $this->cat_p
    'id_category' => (int)$this->category->id,
    'id_category_parent' => (int)$this->category->id_parent,
    'return_category_name' => Tools::safeOutput($this->category->name),
    'path' => Tools::getPath($this->category->id),
    'add_prod_display' => Configuration::get('PS_ATTRIBUTE_CATEGORY_DISPLAY'),
    'categorySize' => Image::getSize(ImageType::getFormattedName('category')),
    'mediumSize' => Image::getSize(ImageType::getFormattedName('medium')),
    'thumbSceneSize' => Image::getSize(ImageType::getFormattedName('m_scene')),
    'homeSize' => Image::getSize(ImageType::getFormattedName('home')),
    'allow_oosp' => (int)Configuration::get('PS_ORDER_OUT_OF_STOCK'),
    'comparator_max_item' => (int)Configuration::get('PS_COMPARATOR_MAX_ITEM'),
    'suppliers' => Supplier::getSuppliers(),
    'body_classes' => array($this->php_self.'-'. $this->category->id, $this->php_self
));
```

Dès lors, le template Smarty peut manipuler les variables `$category`, `$description_short`, `$products`... comme on l'a vu dans les extraits du template ci-dessus.



Enfin, la programmation de la présentation dans PrestaShop revient essentiellement à établir le/les gabarits de pages ou portions de page HTML dynamiques Smarty et les portions de feuilles de styles nécessaires.

Reste à comprendre où trouver les scripts à modifier et où stocker les nouveaux scripts.

4. CONTROLEURS, MODULES ET TEMPLATES PRESTASHOP

4.1 NOTION DE SURCHARGE DE CONTROLEUR

Un grand principe, et un gros avantage des versions actuelles de PrestaShop, réside dans la notion de **surcharge de classe, applicable aux Contrôleurs PrestaShop** :



Dans PrestaShop, tout contrôleur peut être réécrit par duplication sans intervenir sur le code initial. Dès lors, la mise à jour du contrôleur d'origine lors du changement de version n'a aucune incidence sur le code réécrit.

Le mécanisme de surcharge de classe fait l'objet de la séance consacrée au développement de modules. Ici, seule la présentation nous intéresse ; on déroulera le processus en se focalisant sur les mécanismes d'affichage via les templates.

Prenons l'exemple du contrôleur des catégories déjà évoqué plus haut. Pour cet exemple (qui n'a d'autre intérêt que de mettre en évidence le mécanisme), nous voulons afficher un message totalement différent quand ce contrôleur est invoqué.

Le contrôleur des catégories est appelé par le contrôleur principal `index.php` lorsque l'utilisateur choisit une catégorie dans le Front-office ; la demande de page est alors de type :

`http://localhost:8080/prestashop16/index.php?id_category=14&controller=category&id_lang=1`



Le contrôleur d'origine est bien situé dans le dossier `controllers/front` de la boutique ; recopier le fichier `CategoryController.php` dans le dossier `override/controllers/front` de la boutique.

Ouvrir cette copie à l'aide de l'éditeur de code (NotePad ++...) pour modifier uniquement la déclaration de classe :

```
class CategoryController extends FrontController {...
```

Il suffit de supprimer le suffixe `Core` pour surcharger le contrôleur !

4.2 MODIFIER LE TEMPLATE ASSOCIE A UN CONTROLEUR

Pour l'instant, les 2 contrôleurs ont exactement le même comportement. Nous voulons simplement modifier l'affichage, il s'agit donc d'intervenir sur le bon template ; c'est bien évidemment la méthode `initContent()` qui fait le lien avec le template utilisé :

```
public function initContent()
{
    parent::initContent();

    $this->setTemplate(_PS_THEME_DIR_.'category.tpl');
...
}
```



Rechercher le fichier `category.tpl` dans le dossier du thème de la boutique.

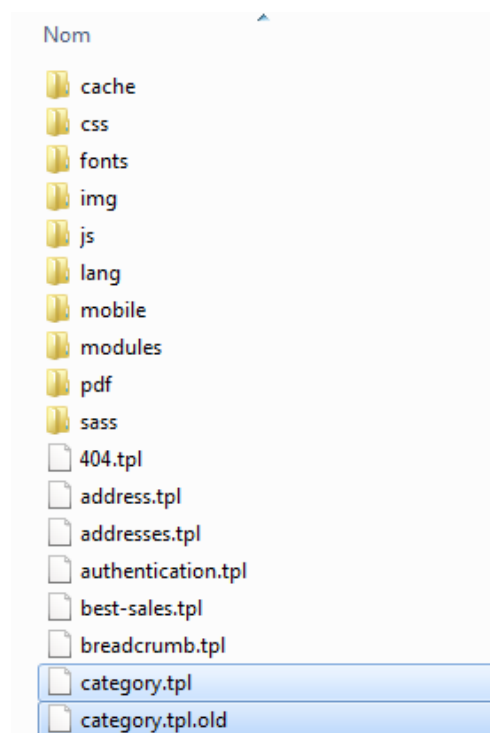
Les templates ne peuvent être surchargés ; effectuer une copie du fichier. L'original peut être différencié en y ajoutant l'extension `.old`, la copie conserve le même nom :

Ouvrir `category.tpl` dans l'éditeur de code et supprimer tout le contenu pour ne laisser qu'un message simple de type :

```
{*
* redéfinition du template category.tpl
pour exemple
*}
```

Exemple de surcharge du contrôleur de catégories.

Voilà la surcharge du contrôleur effectuée ; c'est immédiat, il suffit de recharger la page d'accueil et de choisir une catégorie pour voir le nouveau message dans le corps de la page :



NB : on aurait aussi bien pu créer un nouveau fichier template et l'enregistrer sous un nouveau nom ; il aurait alors été nécessaire de modifier l'appel de la méthode `setTemplate()` dans la méthode `initContent()`.

NB : noter que cette manipulation n'a concerné que le thème choisi et que l'on retrouve un comportement standard en activant un autre thème.

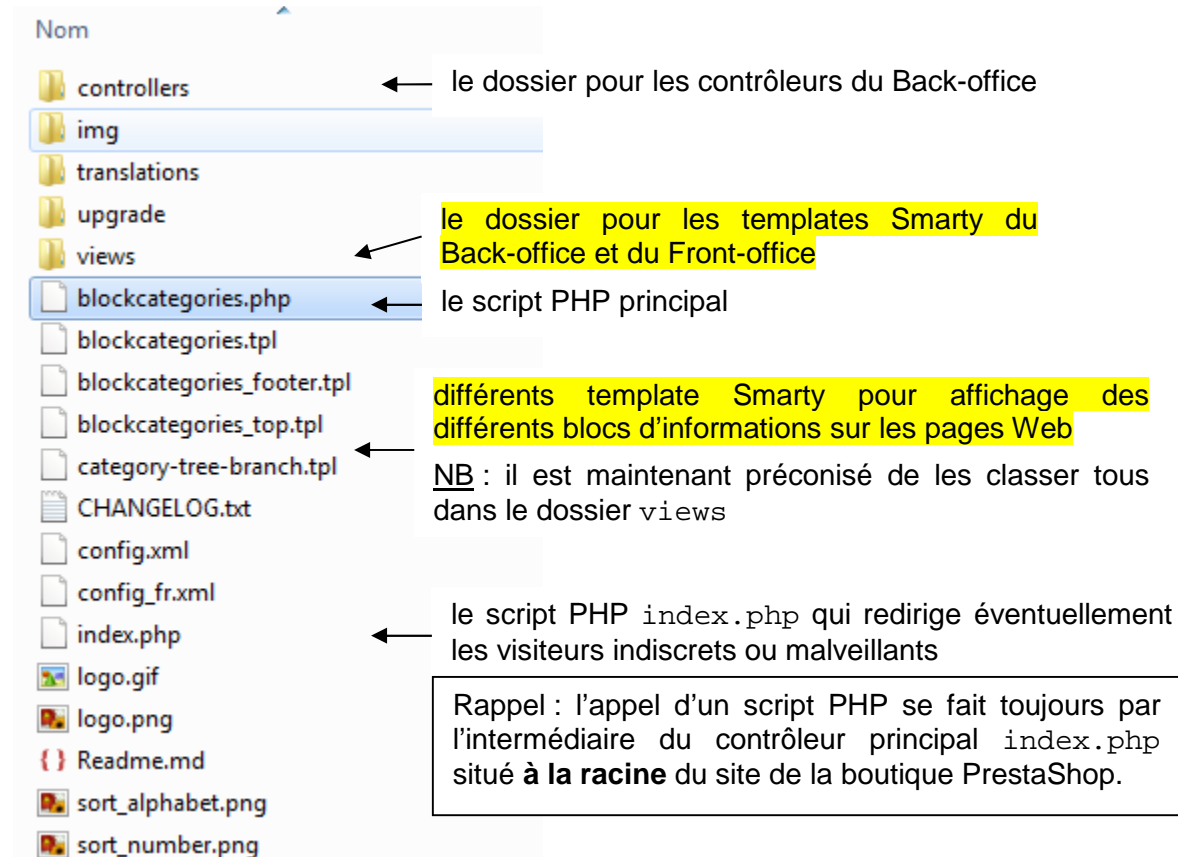
Pour rétablir le fonctionnement initial, il suffit de supprimer (ou de renommer) le contrôleur et le template créés (`CategoryController.php` et `category.tpl`) et de rétablir le nom initial du template d'origine. Grâce au mécanisme de surcharge de PrestaShop, rien n'a été perdu !

4.3 STRUCTURE D'UN MODULE STANDARD PRESTASHOP

Au-delà des contrôleurs standards qui assurent la génération des pages d'une boutique PrestaShop, il est possible d'ajouter de multiples modules complémentaires qui assurent chacun une fonction bien délimitée et qui se greffent en général dans des pages du Front-Office ou du Back-Office. Le Back-office PrestaShop et le site des 'add-on' (<http://addons.prestashop.com/>) aident à la personnalisation des fonctionnalités de la boutique par ajout/activation/paramétrage de ces modules.

La création de modules fait partie du rôle d'un développeur mais ce n'est pas encore le sujet. Pour l'instant, il s'agit de s'intéresser au fonctionnement des modules générant un affichage sur une page standard.

Un module PrestaShop est constitué d'un ensemble de scripts stockés dans un sous-dossier du dossier `modules` de la boutique. Ainsi pour le module `blockcategories` qui affiche les catégories :



***NB :** PrestaShop a déjà beaucoup évolué (et évoluera encore) ; en particulier, l'organisation des composants se structure de plus en plus depuis la version 1.5. Par exemple, il est vivement préconisé de stocker tous les templates dans les sous-dossiers du dossier `views`. Mais les modules livrés en standards ne sont pas tous réécrits selon cette nouvelle convention ; c'est pourquoi, dans cet exemple, les templates sont stockés 'en vrac' dans le dossier principal...*

4.4 MODIFIER LE GABARIT ASSOCIE A UN MODULE

4.4.1 Intervenir sur le bon gabarit

Quand on y regarde de plus près, on constate de multiples template et feuilles de styles pour une même fonctionnalité, portant le même nom mais situé dans des dossiers différents.

Ainsi, le template `blockcategories.tpl` se trouve dans le dossier des modules de la boutique mais aussi dans chacun des dossiers de thème de la boutique.

Comment s'y retrouver pour savoir où intervenir ?

Une bonne méthode est d'utiliser tout d'abord le **débogueur du navigateur** qui permet de :

- Consulter la structure HTML de la page, et ses commentaires par blocs, issus des templates des modules,
- Consulter les styles CSS appliqués de manière à identifier l'ensemble des feuilles CSS qui interviennent dans le rendu final.



4.4.2 Exemple guidé à réaliser :

The screenshot shows a PrestaShop storefront with a sidebar menu, a welcome message, and a product list. Annotations include:

- A box pointing to the product list with the text: "Identifier les attributs CSS et feuilles de styles".
- A box pointing to the HTML structure in the developer tools with the text: "Identifier le bloc donc le module".
- Developer tools showing the HTML structure of the product list, highlighting the `<ul class="product_list row list">` block.
- Developer tools showing the CSS styles applied to the product list, including `index.php grid.scss:12`, `index.php type.scss:103`, and `index.php utilities.scss:24`.

Les fonctionnalités de mise en page du Back-office (Live Edit, Positions des modules) peuvent aussi aider à cette identification.

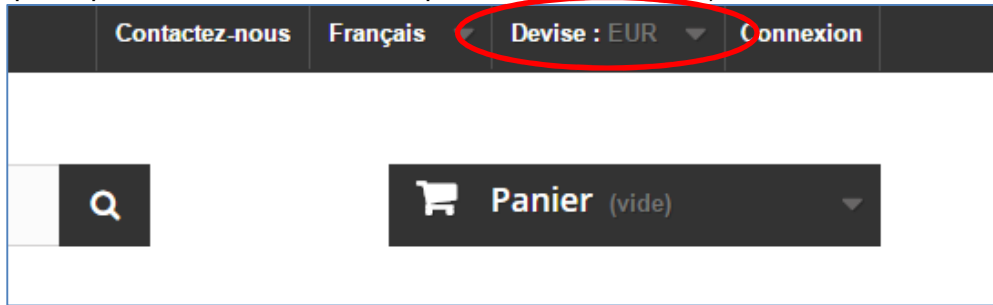
Appliquons tout ceci à l'affichage des devises en barre de navigation : nous voulons modifier le libellé et les couleurs standards pour un thème particulier.

- Vérifier que le 'Bloc devises' est activé dans le Back-office et qu'il s'affiche bien dans la barre de navigation (il peut être nécessaire d'ajouter une devise car ce bloc ne s'affiche

Développer des gabarits de mise en page - PrestaShop

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

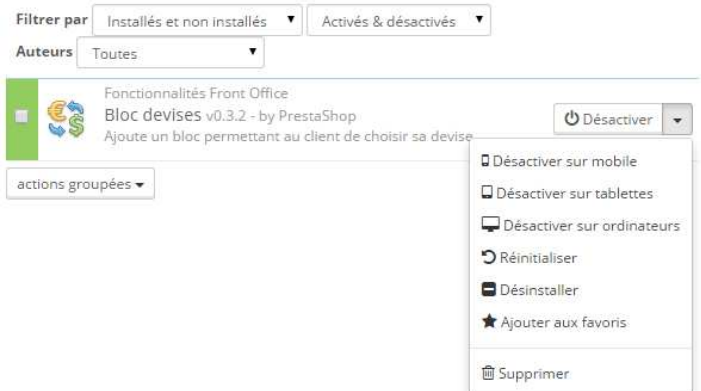
que si plusieurs devises sont disponibles pour le client) :



- Observer le module 'Bloc devises' en Back-office : il n'offre aucun moyen de paramétrage :

Il faudra donc plonger dans le code mais avant cela, il faut identifier précisément le bloc d'affichage et le module concernés.

- Live Edit renseigne peu sur le module car il permet simplement d'ajuster son positionnement.
- Le **débogueur du navigateur** permet d'identifier le **bloc HTML** donc le **nom du module** qui le génère et les **éléments HTML** où intervenir :



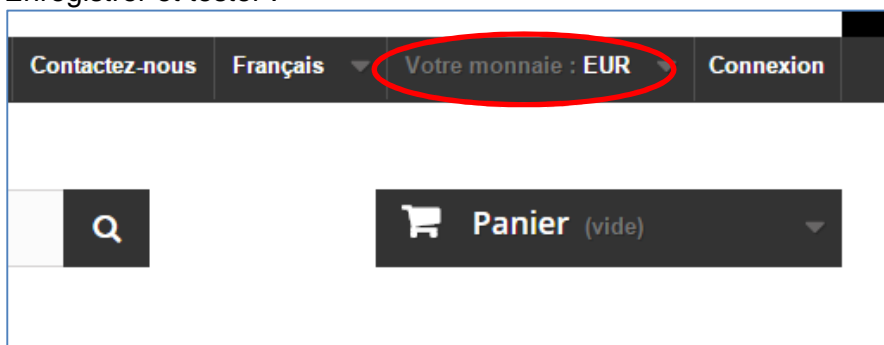
- Dans le dossier du thème en cours, rechercher le sous-dossier du module 'blockcurrencies' : il contient le template **blockcurrencies.tpl** ; c'est là que ça se passe !

Editer le template **blockcurrencies.tpl** → rechercher le `` correspondant :
`<div class="current">`


```
<input type="hidden" name="id_currency" id="id_currency" value=""/>
<input type="hidden" name="SubmitCurrency" value="" />
<span class="cur-label">{l s='Currency' mod='blockcurrencies'} :</span>
```

C'est bien là que ça se passe ; comme il s'agit d'une chaîne traduite, modifier la traduction du thème dans le Back-office : **'Votre monnaie'** (pas d'intervention dans le code pour cette fois).

- Il reste à modifier les couleurs, c'est l'affaire de CSS. On peut éventuellement tester dans le débogueur du navigateur en modifiant les valeurs de la propriété `color` des éléments. Dans le sous-dossier du module des CSS du thème **.../css/modules/blockcurrencies**, éditer la feuille de styles **blockcurrencies.css** pour inverser les couleurs (`#777777` et `#fff`).
- Enregistrer et tester :



***NB** : cette modification est applicable uniquement pour ce thème. Pour une modification valable pour tous les thèmes, il faudrait intervenir sur les éléments de base du module, stockés dans le dossier `/modules/blockcurrencies` de la boutique ; on y retrouve bien le même template et la même feuille de styles, qui s'appliquent à tous les thèmes, sauf si c'est contredit dans un thème particulier, comme ici.*

 Ces quelques manipulations ont simplement pour but de mettre en évidence les différents points d'intervention et les différents moyens du développeur :

- Live Edit et configuration de module sont à privilégier.
- A tout moment, privilégier l'intervention grâce aux fonctionnalités Back-office PrestaShop plutôt que de modifier le code (système de traduction, formulaires du Back-office...).
- Pour plonger dans le code, il est nécessaire de bien **identifier l'endroit où intervenir** grâce au débogueur du navigateur et à l'exploration des composants de code source.
- La structure très modulaire des composants de PrestaShop aide à la tâche et limite les effets de bord (possiblement désastreux...).

4.5 CREATION DE NOUVEAUX GABARIT

Tous les principes abordés lors de cette séance vont pouvoir s'appliquer en cas de création d'un nouveau module. Mais un module doit aussi assurer un traitement qu'il faut décrire en langage PHP. C'est l'objet d'une autre séance...

5. EN GUISE DE SYNTHÈSE SUR LES GABARITS DE MISE EN PAGE PRESTASHOP

- Dans PrestaShop, un gabarit de mise en page est établi sous la forme d'un template Smarty.
- Un template Smarty contient du code HTML et des instructions spécifiques permettant de fusionner des variables PHP reçues au moment de son exécution.
- En général, un template Smarty assure la génération personnalisée d'une portion de page (un ensemble de <div> imbriqués).
- Les instructions spécifiques en langage Smarty sont indiquées entre accolades.
- Le contenu d'une variable PHP est positionné dans le code HTML du template en précisant simplement son nom de variable PHP entre accolades.
- Le langage Smarty permet de formater le contenu des variables à fusionner (transformations, troncature...).
- Le langage Smarty offre au développeur des instructions de test et de boucle afin de pouvoir parfaitement contrôler le code HTML d'affichage généré.
- PrestaShop adopte une architecture 'Modèle-Vue-Contrôleur' ; les templates et feuilles de styles CSS constituent la couche 'Vue'.
- Les traitements fournissant les pages standards de PrestaShop sont assurés par des contrôleurs écrits en PHP et stockés dans le dossier `/controllers` de la boutique ; ces contrôleurs peuvent associer les feuilles de styles nécessaires grâce à leur méthode `addCSS()` appelée par défaut par la méthode `setMedia()` ; ils peuvent associer le template nécessaire grâce à leur méthode `setTemplate()` appelée par défaut par la méthode `initContent()`.
- Il est préférable de surcharger un contrôleur plutôt que de le modifier de manière à assurer les évolutions futures de PrestaShop (copie dans le dossier `/override` de la boutique et modification du nom de classe).
- Un module PrestaShop assure une fonction particulière bien délimitée, bien souvent l'affichage d'une portion de page de la boutique ; un module est associé à la page par le mécanisme des crochets (Hook).
- Un module PrestaShop est lui aussi construit en architecture MVC : le contrôleur assure les traitements nécessaires en PHP, les vues sont constituées des templates et feuilles CSS nécessaires à l'affichage de la portion de page.
- Un module ou un contrôleur peuvent utiliser des templates et CSS différents selon le thème activé pour la boutique. Il est important de bien localiser les templates et CSS à modifier selon la portée voulue de la modification.

CREDITS

ŒUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)

Benoit Hézard - formateur

Chantal Perrachon – Ingénieure de formation

Date de mise à jour : 29/02/16

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Développer des gabarits de mise en page - PrestaShop

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »