

Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Mettre en œuvre une solution
e-commerce »

Développer des Modules - PrestaShop

Apprentissage

Mise en pratique

Evaluation

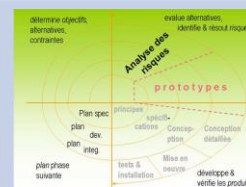
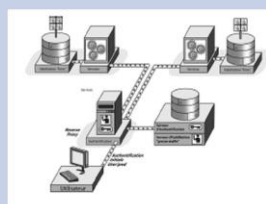


TABLE DES MATIERES

Table des matières	2
1. Module pour afficher les horaires d'ouverture.....	3
1.1 Créer la structure physique pour stocker les composants du module	3
1.2 Ecrire le code PHP du module	3
1.3 Ecrire le code permettant l'installation et la désinstallation du module par le Back-office 3	
1.4 Logo du module	3
1.5 Ecrire le code permettant l'affichage en Front-office.....	4
1.6 Ecrire les composants de vue du module	4
1.7 Tests intermédiaires.....	4
1.8 Ecrire le code permettant le paramétrage du message	4
1.9 Tester.....	5
1.10 Finaliser le développement	5
2. Message spécifique pour les professionnels	6
2.1 Explorez PrestaShop pour comprendre comment est géré le groupe de clients	6
2.1.1 Dans le Back-office.....	6
2.1.2 Dans la Base de données MySQL	7
2.1.3 Dans le modèle de données PHP	7
2.2 Adapter le code du module pour afficher le message complémentaire conditionnel	8
2.3 Finaliser le développement.....	8
2.4 Extension possible	9
3. Exportation des e-mails pour les clients enregistrés.....	9
3.1 Créer la structure physique pour stocker les composants du module	9
3.2 Ecrire le code PHP du module	10
3.3 Ecrire le code permettant l'installation et la désinstallation du module par le Back-office 10	
3.4 Logo du module	10
3.5 Ecrire le code permettant le paramétrage du module en Back-office	10
3.6 Compléter avec le code permettant la génération du fichier texte des adresses e-mail	11
3.7 Tester.....	11
3.8 Finaliser le développement.....	12
3.9 Extensions possibles.....	12

MISE EN PRATIQUE : DEVELOPPER DES MODULES PRESTASHOP

1. MODULE POUR AFFICHER LES HORAIRES D'OUVERTURE

Il s'agit ici de créer un premier module simple permettant d'afficher les horaires d'ouverture des points de vente de la boutique ; le message viendra s'insérer dans la barre de navigation standard.

1.1 CREER LA STRUCTURE PHYSIQUE POUR STOCKER LES COMPOSANTS DU MODULE

Dans le dossier de votre boutique PrestaShop, créez un dossier `openinghours` dans le dossier `modules`.

Ajouter les sous-dossiers nécessaires selon la structure recommandée PrestaShop 1.6.

1.2 ECRIRE LE CODE PHP DU MODULE

Avec un éditeur de code adapté (NotePad++...), écrivez le contenu essentiel du module en langage PHP (fichier `openinghours.php`) :

- Code préliminaire de sécurité pour éloigner les indiscrets.
- Définition de la classe de module `OpeningHours`.
- Code du constructeur de la classe définissant les paramètres généraux dont les propriétés `name`, `tab`, `version`, `author`, `ps_versions_compliancy`, `bootstrap`, `displayName`, `description` et `confirmUninstall`.

1.3 ECRIRE LE CODE PERMETTANT L'INSTALLATION ET LA DESINSTALLATION DU MODULE PAR LE BACK-OFFICE

Ajoutez le code PHP des méthodes `install()` et `uninstall()` en suivant la structure préconisée.

Ce module aura besoin d'une feuille de styles spécifique et il faudra l'associer au *hook* `displayNav` pour que son affichage apparaisse dans la barre de navigation.

Ce module affichera un message court ('Ouvert de 9h à 19h') stocké (tout comme le nom du module) dans les tables de paramètres de PrestaShop.

La procédure de désinstallation devra bien entendu supprimer ces 2 informations de la base de données.

1.4 LOGO DU MODULE

Créez ou récupérez un logo pour ce module (format PNG, 32 x 32 pixels) et stockez-le au bon endroit dans la structure de dossiers et documents du module.

1.5 ECRIRE LE CODE PERMETTANT L’AFFICHAGE EN FRONT-OFFICE

Ajoutez au module le code de la méthode définissant les modalités d’affichage lorsque le *hook* `displayNav` est déclenché. Il s’agit :

- De récupérer la valeur du message depuis la table des paramètres PrestaShop.
- De transmettre cette valeur au moteur Smarty.
- De provoquer l’exécution du template Smarty correspondant `openingHours.tpl` (qui n’est pas encore écrit).

Ajouter aussi le code de la méthode permettant de provoquer l’appel de la feuille de styles CSS correspondante `openingHours.css` (qui n’est pas encore écrite non-plus).

1.6 ECRIRE LES COMPOSANTS DE VUE DU MODULE

Ecrivez le template `openingHours.tpl` destiné au Front-office et stockez-le dans le sous-dossier adéquat.

Ce template très court contient essentiellement le code *commenté* permettant d’afficher dans un élément HTML `<div>` bien *identifié* le contenu de la variable que le module a transmis au moteur Smarty.

Ecrivez la feuille de styles CSS nécessaire et stockez-la dans le sous-dossier adéquat. Elle définit essentiellement une règle de styles pour l’élément `<div>` du template afin de préciser sa mise en forme (police, taille, couleur, marges, flottement...)

1.7 TESTS INTERMEDIAIRES

Vérifiez bien vos codes et enregistrez le tout.

Dans le Back-office PrestaShop, affichez la liste des modules, retrouvez votre module `openinghours` et procédez à son installation.

Dans le Front-office, affichez ou rafraîchissez une page quelconque comme la page d’accueil et vérifiez la présence et l’apparence du message en barre de navigation. Effectuez au besoin quelques ajustements CSS sans toutefois rechercher la perfection car là n’est pas le sujet.

1.8 ECRIRE LE CODE PERMETTANT LE PARAMETRAGE DU MESSAGE

Ajoutez à votre module les 3 méthodes nécessaires pour présenter à l’utilisateur du Back-office un formulaire simple permettant de modifier le message des horaires d’ouverture.

- Méthode `getContent()` qui, en cas de soumission du formulaire, doit récupérer et contrôler la valeur saisie par l’utilisateur, la mettre à jour en table de paramètres PrestaShop et préparer le message de succès, et qui, dans tous les cas, demande l’affichage du formulaire de saisie par la méthode `displayFormOpeningHours()`.
- Méthode `displayFormOpeningHours()` qui instancie le générateur de form PrestaShop, définit son contenu grâce à la 3^e méthode `formOpeningHours()` (qui

reste à écrire), définit les paramètres généraux du form dont la langue, le titre et les boutons de sauvegarde et de retour et fait générer le code HTML par le helper.

- Méthode `formOpeningHours()` qui définit le contenu du formulaire HTML de saisie, principalement un titre, un label et une zone de texte, et un bouton de soumission.

1.9 TESTER

Vérifiez bien vos codes et enregistrez le tout.

Dans le Back-office PrestaShop, affichez la liste des modules, retrouvez votre module `openinghours` et demandez sa configuration ; saisissez un nouveau message court, enregistrez et testez dans le Front-office (veillez à ajuster les paramètres de performance de PrestaShop).

Procédez à la mise au point de votre code si nécessaire jusqu'à obtention du résultat voulu.

Commentez bien votre code.

1.10 FINALISER LE DEVELOPPEMENT

Dans le Back-office PrestaShop, procédez à la traduction du module, en langue anglaise par exemple.

Dans le Back-office, changez les préférences de langue de l'utilisateur courant de manière à afficher le Back-office en Anglais. Effectuez les traductions éventuellement nécessaires.

Dans le Front-office testez les affichages en changeant de langue.

Ajustez au besoin votre code de manière à obtenir un bon résultat dans les 2 langues.

NB : la traduction du message stocké en table de configuration peut poser quelques problèmes... Il serait possible de prévoir 2 messages différents, stockés sous 2 noms différents, mais cela nécessiterait de tester la langue de l'utilisateur pour déterminer quelle variable de configuration prendre en compte. L'exercice suivant permet d'envisager une solution pour effectuer ce test...

2. MESSAGE SPECIFIQUE POUR LES PROFESSIONNELS

Assurez-vous que vous avez déterminé plusieurs groupes de clients pour votre boutique, par exemple, les 'particuliers' et les 'professionnels', et que vos clients sont bien rattachés à un groupe par défaut.

Il s'agit maintenant de préciser dans notre message que, par exemple, les points de vente sont 'ouverts aux professionnels dès 8h' ; ce message ne doit s'afficher qu'aux clients du groupe professionnels.

Exemple en Français pour un particulier connecté :



Exemple en Français pour un professionnel connecté :



Reste à récupérer dans notre module les informations sur le client.

2.1 EXPLOREZ PRESTASHOP POUR COMPRENDRE COMMENT EST GERE LE GROUPE DE CLIENTS

2.1.1 Dans le Back-office

Un client peut être associé à plusieurs groupes dont un groupe par défaut comme le montre l'écran de mise à jour des clients :

L'écran de mise à jour d'un client dans l'interface back-office de PrestaShop. Les champs de saisie incluent : Prénom (Prérempli : Professionnel), Nom (Un), Adresse e-mail (professionnel.un@un.fr), Mot de passe (avec un bouton pour générer un mot de passe), et Date de naissance (3 Mars 1997). Il y a des boutons 'OUI' et 'NON' pour 'Activé', 'Lettre d'informations', et 'Opt-In'. Une section 'Accès des groupes' contient un tableau avec des cases à cocher pour sélectionner les groupes auxquels le client appartient. Le groupe 'Professionnels' est sélectionné. En bas, un menu déroulant 'Groupe de clients par défaut' est réglé sur 'Professionnels' et est entouré d'un cercle noir.

ID	Nom du groupe
1	Visiteur
2	Invité
<input checked="" type="checkbox"/>	3 Client
<input checked="" type="checkbox"/>	4 Professionnels
<input type="checkbox"/>	5 Particuliers
<input type="checkbox"/>	6 particulier
<input type="checkbox"/>	7 professionnel

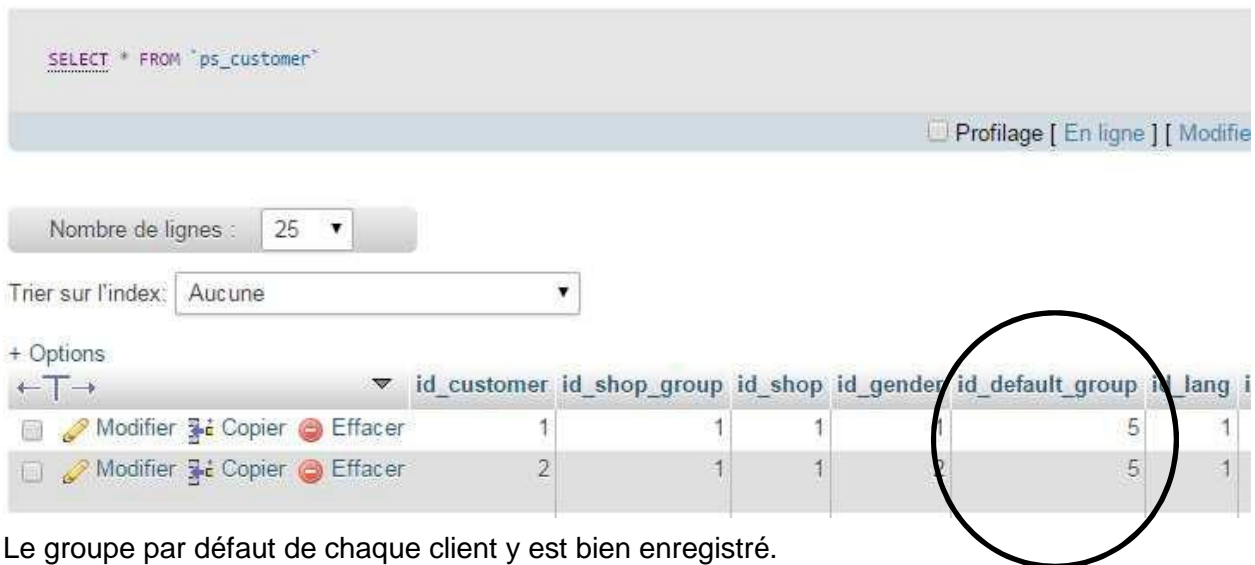
Développer des modules - PrestaShop

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

2.1.2 Dans la Base de données MySQL

Les tables standards de la boutique sont préfixées `ps_` par défaut.

Dans PhpMyAdmin, il est aisé d'y retrouver les clients (`ps_customer`) :



SELECT * FROM `ps_customer`

Profilage [En ligne] [Modifier]

Nombre de lignes : 25

Trier sur l'index: Aucune

+ Options

	id_customer	id_shop_group	id_shop	id_gender	id_default_group	id_lang	i
Modifier Copier Effacer	1	1	1	1	5	1	
Modifier Copier Effacer	2	1	1	2	5	1	

Le groupe par défaut de chaque client y est bien enregistré.

2.1.3 Dans le modèle de données PHP

Toutes les données des tables sont bien entendu représentées en mémoire par des *objets* PHP ; leur définition se trouve dans le dossier `classes` de la boutique.

L'étude des 800 lignes de code de la classe `CustomerCore` peut être rébarbative et pas très efficace pour notre besoin actuel. On peut toutefois y retrouver aisément un attribut portant le nom du champ de la table MySQL :

```
44  /** @var integer Default group ID */
45  public $id_default_group;
```

Les *contrôleurs* standards de PrestaShop mémorisent cet objet, quand l'utilisateur client s'est connecté à la boutique, dans leur variable `$this->context`. Il suffit d'insérer une simple instruction PHP de débogage pour consulter l'état de cette variable.

Modifiez le code de la méthode `hookDisplayNav()` :

```
public function hookDisplayNav($params)
{
    $openingHours = Configuration::get(...);
    $this->context->smarty->assign(array(
        'openingHours' => $openingHours
    ));
    var_dump($this->context);
    return $this->display(_FILE_, 'openingHours.tpl');
}
```

Rafraichissez l'affichage du Front-office ; la barre de navigation se trouve 'polluée' par une foule d'informations hiérarchisées ; on y retrouve entre autres l'objet `smarty` et bien évidemment un objet `customer` doté de toutes ses propriétés :

```
protected 'def' =>
    array (size=4)
        'table' => string 'cart' (length=4)
        'primary' => string 'id_cart' (length=7)
        'fields' =>
            array (size=18)
                ...
                'classname' => string 'Cart' (length=4)
protected 'update_fields' => null
public 'force_id' => boolean false
public 'customer' =>
    object(Customer)[13]
public 'id' => int 10
public 'id_shop' => string '1' (length=1)
public 'id_shop_group' => string '1' (length=1)
public 'secure_key' => string '93772261fba66fc82626be8318a94b2b' (length=32)
public 'note' => null
public 'id_gender' => string '1' (length=1)
public 'id_default_group' => string '5' (length=1)
public 'id_lang' => string '1' (length=1)
public 'lastname' => string 'Un' (length=2)
public 'firstname' => string 'Particulier' (length=11)
```

2.2 ADAPTER LE CODE DU MODULE POUR AFFICHER LE MESSAGE COMPLEMENTAIRE CONDITIONNEL

Adaptez votre module pour tester la valeur du groupe de client par défaut du client qui est connecté. Si cette valeur correspond à un professionnel (valeur 4 dans notre exemple), concaténez un message traductible défini 'en dur' dans votre code PHP avant de le transmettre au moteur Smarty :

```
if ($this->context->customer->id_default_group == ...)
    $openingHours .= $this->l(...);
```

Enregistrez le code du module ; il n'a même pas besoin d'être réinitialisé ou réinstallé. Testez en vous connectant comme client particulier et comme client professionnel.

Testez et sécurisez au besoin (cas d'un client qui n'est pas connecté...).

2.3 FINALISER LE DEVELOPPEMENT

Mettez à jour la traduction du module, dans les 2 langues et observez à nouveau le comportement en changeant de langue dans le Front-office.

Commentez bien votre code.

2.4 EXTENSION POSSIBLE

Le contexte du contrôleur vous dit tout sur le client connecté, donc sa langue. Il reste maintenant aisé de définir plusieurs variables pour le message de base, une par langue, de les saisir dans autant de zones de texte dans le Back-office, et de récupérer depuis la table de paramètres le message rédigé dans la langue de l'utilisateur.

3. EXPORTATION DES E-MAILS POUR LES CLIENTS ENREGISTRES

Imaginons que l'entreprise qui publie la boutique dispose déjà d'un logiciel de e-mailing bien adapté à ses besoins. Il est alors inutile d'acquérir un module de e-mailing spécifique à PrestaShop et il serait tout aussi inutile de ressaisir dans le logiciel existant les adresses e-mail déjà renseignées par les clients enregistrés.

Nous devons 'simplement' récupérer ces adresses depuis la base de données de PrestaShop pour les faire importer par le logiciel existant. C'est l'objet de cet exercice.

Dans PrestaShop, un client enregistré doit renseigner une adresse e-mail stockée dans la table `ps_customer` (sous le nom de champs `email`).

Si le client s'inscrit à la newsletter, cette information est mémorisée dans la table `ps_emailsubscribe` (sous le même nom `email`).

On peut donc envisager deux traitements, l'un pour récupérer les adresses des clients (publicités) et l'autre pour récupérer les adresses concernant ceux qui sont abonnés à la newsletter (informations périodiques).

Le module à créer doit donc :

- offrir un choix à l'utilisateur du Back-office (à travers une boîte de liste de formulaire) pour déterminer quel lot d'adresses e-mail il souhaite exporter.
- Rechercher dans la base de données les adresses e-mail dans la table correspondante.
- Pour chaque adresse récupérée, recopier sa valeur sous forme d'une ligne de texte insérée dans un fichier texte daté, et stocké dans le dossier principal du module.
- Informer l'utilisateur du Back-office du bon déroulement des opérations.

Dès lors, l'administrateur de la boutique pourra importer ces adresses dans son logiciel et déclencher les envois d'e-mails avec son logiciel habituel.

3.1 CREER LA STRUCTURE PHYSIQUE POUR STOCKER LES COMPOSANTS DU MODULE

Dans le dossier de votre boutique PrestaShop, créez un dossier `exportemail` dans le dossier `modules`.

Ajouter les sous-dossiers nécessaires selon la structure recommandée PrestaShop 1.7.

3.2 ECRIRE LE CODE PHP DU MODULE

Avec un éditeur de code adapté (NotePad++...), écrivez le contenu essentiel du module en langage PHP (fichier `exportemail.php`) :

- Code préliminaire de sécurité pour éloigner les indiscrets.
- Définition de la classe de module `ExportEmail`.
- Code du constructeur de la classe définissant les paramètres généraux dont les propriétés `name`, `tab`, `version`, `author`, `ps_versions_compliancy`, `bootstrap`, `displayName`, `description` et `confirmUninstall`.

3.3 ECRIRE LE CODE PERMETTANT L'INSTALLATION ET LA DESINSTALLATION DU MODULE PAR LE BACK-OFFICE

Ajoutez le code PHP des méthodes `install()` et `uninstall()` en suivant la structure préconisée.

Ce module n'aura pas besoin de feuille de styles ni d'être associé à aucun *hook* car tout se passera dans le Back-office.

Ce module mémorise le dernier choix de table réalisé par l'utilisateur du Back-office ; ce choix est stocké dans les tables de paramètres de PrestaShop.

La procédure de désinstallation devra bien entendu supprimer cette information de la base de données.

3.4 LOGO DU MODULE

Créez ou récupérez un logo pour ce module (format PNG, 32 x 32 pixels) et stockez-le au bon endroit dans la structure de dossiers et documents du module.

3.5 ECRIRE LE CODE PERMETTANT LE PARAMETRAGE DU MODULE EN BACK-OFFICE

Ajoutez à votre module les 3 méthodes nécessaires pour présenter à l'utilisateur du Back-office un formulaire simple permettant de choisir entre les deux tables dans une boîte de liste.

- Méthode `getContent()` qui, en cas de soumission du formulaire, doit récupérer la valeur choisie par l'utilisateur, la mettre à jour en table de paramètres PrestaShop et préparer le message de succès ; il restera à compléter avec le traitement d'exportation proprement dit. Dans tous les cas, cette méthode se termine en redemandant l'affichage du formulaire de saisie `displayFormExportEmail()`.
- Méthode `displayFormExportEmail()` qui instancie le générateur de form PrestaShop, définit son contenu grâce à la 3^e méthode `formExportEmail()` (qui reste à écrire), définit les paramètres généraux du form dont la langue, le titre et les boutons de sauvegarde et de retour et fait générer le code HTML par le helper.
- Méthode `formExportEmail()` qui définit le contenu du formulaire HTML de saisie, principalement un titre, un label et une boîte de liste, et un bouton de soumission

3.6 COMPLETER AVEC LE CODE PERMETTANT LA GENERATION DU FICHIER TEXTE DES ADRESSES E-MAIL

Pour cet exercice, l'exportation des adresses e-mail sera effectuée par l'utilisateur du Back-office lorsque celui-ci demande de configurer le module ; l'utilisateur choisit alors la table et lance le traitement ; en fin de traitement d'exportation le formulaire de choix de table s'affiche à nouveau et un message indique le nombre d'adresses e-mail exportées.

Tout se passe donc dans la méthode `getContent()` après mémorisation de la table traitée. Il faut donc écrire le code PHP pour :

- Construire le libellé de la requête SQL permettant de récupérer le champ `email` depuis la table voulue.
- Faire exécuter cette requête SQL par la classe `PrestaShop Db`.
- Ouvrir un fichier texte, dans le dossier courant, en *écriture* et *codage binaire*, dont le nom est concaténé sous la forme `'exportemail-<table>-<date>.txt'` (exemple : `exportemail-ps_customer-03092015.txt`).
- Parcourir le résultat de la requête SQL, ligne à ligne, de manière à écrire pour chacune une ligne de texte dans le fichier, constituée tout simplement de la valeur de l'adresse récupérée suivie des caractères de saut de ligne (ne pas oublier d'effectuer à la volée le comptage des adresses récupérées).
- Fermer le fichier texte en fin de parcours des adresses récupérées et préparer le message de retour.

NB : voir au besoin les aides en ligne sur les fonctions PHP `fopen()`, `fclose()` et `fputs()`.

Exemple d'image écran réaffichant le formulaire précédé du résultat de l'exportation qui vient d'être lancée :

MODULES / EXPORTEMAIL / CONFIGURER

Configurer le module "Export e-mail Module"

Précédent Traduire Vérifier la mise à jour Points d'accroche

Settings updated - Finished : 13 email adresse(s) exported.

E-MAIL ADRESSES EXPORT

Choice a table Customer table

Please choice a table.

Process export

3.7 TESTER

Vérifiez bien votre code, commentez-le et enregistrez le tout.

Dans le Back-office PrestaShop, affichez la liste des modules, retrouvez votre module `exportemail`, installez-le et demandez sa configuration ; choisissez une table et lancer l'exportation.

Consultez le contenu du fichier texte dans le dossier du module.

Procédez à la mise au point de votre code si nécessaire jusqu'à obtention du résultat voulu.

3.8 FINALISER LE DEVELOPPEMENT

Dans le Back-office PrestaShop, procédez à la traduction du module, en langues anglaise et française.

3.9 EXTENSIONS POSSIBLES

On peut parfaire ce développement en stockant le fichier texte dans un dossier spécifique du module ou même hors des dossiers de ce module (pourquoi pas au choix de l'utilisateur du Back-office...).

On peut aussi conserver la mémoire de ces exportations (notion de journalisation) en mémorisant dans une table spécifique lors de chaque exportation date d'export, type d'export et nombre d'adresses exportées.

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 modules - PrestaShop

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