Créer un module pour le tableau de bord

Contenu

Créer un module pour le tableau de bord

La version 1.6 de PrestaShop offre un tout nouveau tableau de bord (ou "Dashboard"), fait de blocs de contenus qui peuvent être réorganisés et, ce qui nous intéresse ici, auxquels vous pouvez ajouter les vôtres.

En effet, vous pouvez créer vos propres modules pour le tableau de bord (ou "module dashboard"), que vous pouvez ensuite proposer en téléchargement ou à l'achat sur PrestaShop Addons.

Les différences avec un module PrestaShop normal

Un module de tableau de bord PrestaShop est essentiellement identique à un module PrestaShop normal, avec quelques spécificités. Il est situé dans le dossier /modules, et peut utiliser les contrôleurs de PrestaShop, comme n'importe quel module.

Les spécificités

Le nom

Votre module dashboard doit avoir un nom unique. La convention établie consiste à ajoute un préfixe "dash" au nom de tout module Dashboard : les modules dashboard par défaut ont pour noms "dashproduct", "dashactivity", "dashgoals", etc.

La méthode constructeur

Comme pour tout module PrestaShop, le fichier PHP principal doit contenir une méthode `__construct()` qui déclare les variables habituelles: `name`, `displayName`, `version`, etc.

Quelques nouvelles variables sont nécessaires pour un module dashboard :

- `$this->push_filename`: le dossier où vos données de push devraient être stockées.
- `$this->allow_push`: un booléen indiquant si votre module devrait recevoir des données de push ou non.
- `$this->push_time_limit`: le nombre de secondes entre deux envois de données.

Voici un exemple de méthode `__construct()`, prise depuis le module Dashactivity :

```
public function __construct()
{
        $this->name = 'dashactivity';
        $this->displayName = 'Dashboard Activity';
        $this->tab = '';
        $this->version = '0.1';
        $this->author = 'PrestaShop';
        $this->push_filename = _PS_CACHE_DIR_.'push/activity';
        $this->allow_push = true;
        $this->push_time_limit = 180;

        parent::__construct();
}
```

Notez que la variable `tab` est vide. Elle n'est utile à l'heure actuelle, mais pourrait le devenir dans une prochaine version de PrestaShop.

La méthode install()

Comme pour tout module PrestaShop, le fichier PHP principal doit contenir une méthode `install()` qui déclare les points d'accroche habituels, tel que `displayBackOfficeHeader`.

PrestaShop 1.6 implémente de nombreux nouveaux points d'accroche propres aux modules dashboard :

- `dashboardData`: la manière dont vous souhaitez que vos données soient gérées.
- `dashboardZoneOne`: vous permet d'afficher votre contenu dans la colonne de gauche du tableau de bord.
- `dashboardZoneTwo`: vous permet d'afficher votre contenu dans la colonne centrale du tableau de bord.

Voici un exemple de méthode `install()`, tirée du module Dashproduct :

```
public function install()
{
        if (!parent::install()
                || !$this->registerHook('dashboardZoneTwo')
                || !$this->registerHook('dashboardData'))
                return false;
        return true;
}
```

Les zones

Le tableau de bord dispose de deux zones pour vos modules :

- Zone One : la colonne de gauche du tableau de bord.
- Zone Two : la colonne centrale du tableau de bord.

En fonction des points d'accroche auxquels votre module s'attache, il vous est possible d'afficher votre contenu soit sur l'une des deux colonnes, soit sur les deux si le besoin s'en fait ressentir.

> ⚠ La colonne de droite du tableau de bord n'est pas accessible pour les modules.
>
> L'utilisateur ne peut pas déplacer un module d'une colonne à l'autre, donc il est inutile d'enregistrer les deux zones dans votre code.

Chaque point d'accroche doit appeler un fichier modèle afin d'afficher votre contenu.

Voici par exemple la méthode `hookDashboardZoneTwo()` du module Dashproduct :

```
public function hookDashboardZoneTwo($params)
{
        $this->context->smarty->assign(array(
                'date_from' => Tools::displayDate($params['date_from']),
                'date_to' => Tools::displayDate($params['date_to'])));
        return $this->display(__FILE__, 'dashboard_zone_two.tpl');
}
```

Les modèles

Vous devriez nommer vos modèles en fonction de la zone dans laquelle ils s'afficheront, pour servir de rappel.

Le fichier modèle doit être stocké dans un dossier propre au module : `/modules/dashmyodule /views/templates/hook/dashboard_zone_two.tpl`

Le modèle lui-même est un modèle PrestaShop classique, avec des balises Smarty au milieu de balises HTML.
Comme vous pouvez le voir dans la méthode d'accroche, vous pouvez créer de nouvelles balises Smarty quand votre module est attaché à une zone. Par exemple, avec ce code (inspiré du module Dashactivity) :

```
public function hookDashboardZoneOne($params)
{
        $this->context->smarty->assign(array_merge(array(
                'dashactivity_config_form' => $this->renderConfigForm(),
                'date_subtitle' => $this->l('(from %s to %s)'),
                'date_format' => $this->context->language->date_format_lite
        ), $this->getConfigFieldsValues()));
        return $this->display(__FILE__, 'dashboard_zone_one.tpl');
}
```

Ce code déclare trois nouvelles balises :

- dashactivity_config_form: affiche le contenu de la méthode `renderConfigForm()` du module.
- date_subtitle: affiche une chaîne localisée ("from %s to %s").
- date_format: affiche la date dans un format léger (day.month.year).

Partant de là, vous pouvez appeler ces balises directement dans le modèle. Par exemple, `{$dashactivity_config_form}` affiche le formulaire de `renderConfigForm()`.

Le flux de données

L'intérêt d'avoir un module dashboard consiste à afficher du contenu utile pour votre utilisateur. Une fois que le tableau de bord a terminé de se charger, et chaque fois que vous modifiez les réglages temporels (au haut du tableau de bord), PrestaShop déclenche un appel au point d'accroche `dashboardData` afin de charger les données par le biais de requêtes Ajax.

Pour accrocher du code au point d'accroche `dashboardData`, vous devez créer une méthode `hookDashboardData()`, qui doit contenir les requêtes SQL nécessaire à vos données. Cette méthode doit alors renvoyer un tableau utilisant un type de données standard, chacun contenant un tableau des valeurs de ce type de données.

Par exemple, voici la méthode `hookDashboardData()` de Dashproduct :

```
public function hookDashboardData($params)
{
    $table_recent_orders = $this->getTableRecentOrders();
    $table_best_sellers = $this->getTableBestSellers($params['date_from'], $params['date_to']);
    $table_most_viewed = $this->getTableMostViewed($params['date_from'], $params['date_to']);
    $table_top_10_most_search = $this->getTableTop10MostSearch($params['date_from'], $params['date_to']);
    return array(
        'data_table' => array(
            'table_recent_orders' => $table_recent_orders,
            'table_best_sellers' => $table_best_sellers,
            'table_most_viewed' => $table_most_viewed,
            'table_top_10_most_search' => $table_top_10_most_search,
        )
    );
}
```

Construire hookDashboardData()

Votre méthode `hookDashboardData()` peut être aussi simple ou complexe qu'il est nécessaire pour votre module dashboard. Par exemple, voici la manière dont le module Dashactivity l'implémente : https://github.com/PrestaShop/PrestaShop/blob/c7fe91a0a9e02ca21183cc1c09e3e565d4de7265/modules/dashactivity/dashactivity.php#L95 Notez qu'il est important qu'elle renvoie les différentes valeurs sous la forme d'un tableau de tableaux, avec les types de données comme tableaux racines.

Voici par exemple la fin de la méthode `hookDashboardData()` de Dashactivity :

```
return array(
    'data_value' => array(
        'pending_orders' => $pending_orders,
        'return_exchanges' => $return_exchanges,
        // etc.
    ),
    'data_trends' => array(
        'orders_trends' => array('way' => 'down', 'value' => 0.42),
    ),
    'data_list_small' => array(
        'dash_traffic_source' => $this->getReferer($params['date_from'], $params['date_to']),
    ),
    'data_chart' => array(
        'dash_trends_chart1' => $this->getChartTrafficSource($params['date_from'], $params['date_to']),
    ),
);
```

PrestaShop récupère ces données à l'aide d'une requête Ajax, au format JSON :

```
{"dashactivity":{
  "data_value":{"pending_orders":"0","return_exchanges":"0"},
  "data_trends":{"orders_trends":{"way":"down","value":0.42}},
  "data_list_small":{"dash_traffic_source":{"Direct link":0}},
  "data_chart":{"dash_trends_chart1":{"chart_type":"pie_chart_trends","data":[{"key":"Direct link","y":0}]}}}
}
```

Notez que vous devez mettre les valeurs en place dans un tableau correspondant au type de données :

- Tous les tableaux Values doivent être dans le tableau `data_value`
- Tous les tableaux Trends doivent être dans le tableau `data_trends`
- etc.

Les types de données

Votre module peut utiliser 4 types de données :

- Value: une simple valeur, qui peut utiliser n'importe quel type de données (string, number, boolean, etc.)
- Trends: un type de données spécial, disponible sous la forme de d'un tableau à deux valeurs :
    - 'way' : ce peut être 'up' ou 'down', en fonction de la tendance
    - 'value' : la différence entre la valeur précédente et l'actuelle
- List Small : un tableau de valeurs.
- Chart : un tableau de deux valeurs
    - 'chart_type' : le type de graphique à utiliser (ex. : 'pie_chart_trends').
    - 'data' : un tableau de tableaux :
        - key : la clé
        - y : la valeur

Nommer et récupérer les valeurs affichées

La clé de chaque valeur est très importante, car PrestaShop se basera sur celle-ci pour mettre à jour l'affichage de votre module : à l'aide de JavaScript, le tableau de bord trouvera la balise HTML correspondant à la valeur, et mettre à jour son contenu.

Par exemple, voici le code pour le fichier `dashboard_zone_one.tpl` de Dashactivity :

```
<span class="data_value size_l">
    <span id="pending_orders"></span>
</span>
```

Ce code est lié à la valeur `pending_orders` de flux JSON par le biais d'un appel JavaScript – que vous avez vous-mêmes placé dans le code PHP de votre méthode `hookDashboardData()`.

De fait, vous devez faire attention à la manière donc vous nommez vos éléments HTML et les clés de vos données, ainsi qu'à la corrélation nette entre les deux au sein de PrestaShop.

Vos propres types de données

En plus de ceux disponibles par défaut, vous pouvez créer vos propres types de données ! Par exemple, vous pourriez avoir besoin d'un type Carrousel, ou un booléen clairement binaire. Votre code de retour pourrait alors être :

```
return array(
    'data_boolean' => array(
        'is_enabled' => getModuleState(),
        'must_auto_reload' => $auto_reload,
        // etc.
    ),
);
```

Ce qui vous donnerai la réponse JSON suivante :

```
{"dashmymodule":{
  "data_boolean":{"is_enabled":true,"must_auto_reload":false},
}
```

Les types de données par défaut voient leurs données chargées et placées grâce à des fonctions JavaScript spécifiques. Pour que les données de vos types de données personnalisés puissent eux aussi être chargés et correctement placés, vous devez ajouter votre propre fonction JavaScript, qui est appelé automatiquement lorsque `hookDashboardData()` renvoie son fichier JSON.

La fonction JavaScript doit être dans votre propre fichier `.js`, qui vous devriez ajouter à l'en-tête du thème à l'aide de la méthode `$this->context->controller->addjs('js/mymethods.js', 'all');`.

Elle devrait prendre cette forme simple :

```
function data_boolean(widget_name, data)
{
        // Ici, le code prend le tableau de données envoyé par hookDashboardData, et gérer son affichage à la bonne position.
}
```

Pour avoir un exemple de telles méthodes JavaScript, observez celles utilisées pour `data_value`, `data_trends` et les autres types par défaut dans le fichier https://github.com/PrestaShop/PrestaShop/blob/e71094283395b092ccd0b0a0bb0a0fdfe25cbabc/js/admin-dashboard.js#L112.

Le formulaire de configuration

Les modules dahsboard peuvent disposer de leur propre formulaire de configuration, que l'utilisateur peut ouvrir en un clic de souris.

Pour déclarer le formulaire de configuration, vous devez simplement :

1. Déclarer la fonction PHP qui génèrera ce formulaire,
2. Assigner cette fonction à une balise Smarty
3. Appeler la balise Smarty dans le modèle.

Voici comment le module Dashactivity s'y prend :

1 - Déclaration :

```
public function renderConfigForm()
{
                $fields_form = array(
                                'form' => array(
                                                'id_form' => 'step_carrier_general',
                                                'input' => array(),
                                                'submit' => array(
                                                                'title' => $this->l(' Save '),
                                                                'class' => 'btn btn-default submit_dash_config',
                                                                'reset' => array(
                                                                                'title' => $this->l('Cancel'),
                                                                                'class' => 'btn btn-default
cancel_dash_config',
                                                                )
                                                )
                                ),
                );

                $sub_widget = array(
                                array('label' => $this->l('Show Pending'), 'config_name' =>
'DASHACTIVITY_SHOW_PENDING'),
                                array('label' => $this->l('Show Notifications'), 'config_name' =>
'DASHACTIVITY_SHOW_NOTIFICATION'),
                                array('label' => $this->l('Show Clients'), 'config_name' =>
'DASHACTIVITY_SHOW_CUSTOMERS'),
                                array('label' => $this->l('Show Newsletters'), 'config_name' =>
'DASHACTIVITY_SHOW_NEWSLETTER'),
                                array('label' => $this->l('Show Traffic'), 'config_name' =>
'DASHACTIVITY_SHOW_TRAFFIC'),
                );

// etc.
}
```

Pour voir le code de `renderConfigForm()` en entier, ouvrir le lien https://github.com/PrestaShop/PrestaShop/blob/c7fe91a0a9e02ca21183cc1c09e3e565d4de7265/modules/dashactivity/dashactivity.php#L297.

2 - Assignation à une balise Smarty :

```
public function hookDashboardZoneOne($params)
{
$this->context->smarty->assign(array_merge(array(
                'dashactivity_config_form' => $this->renderConfigForm(),
        ), $this->getConfigFieldsValues()));
        return $this->display(__FILE__, 'dashboard_zone_one.tpl');
}
```

3 - Utilisation au sein du modèle :

```
<section id="dashactivity_config" class="dash_config hide">
        <header><i class="icon-wrench"></i> {l s='Configuration' mod='dashactivity'}</header>
        {$dashactivity_config_form}
</section>
```

Notez bien que le modèle devrait être exactement tel que présenté ici, afin de s'assurer qu'il s'intègre bien dans le design du tableau de bord.