Digidream

Rapport d’activité

Stage du 29 mars 2021 au 30 avril 2021

Olivier KEITH
02/05/2021

Table des matières

Présentation de la société

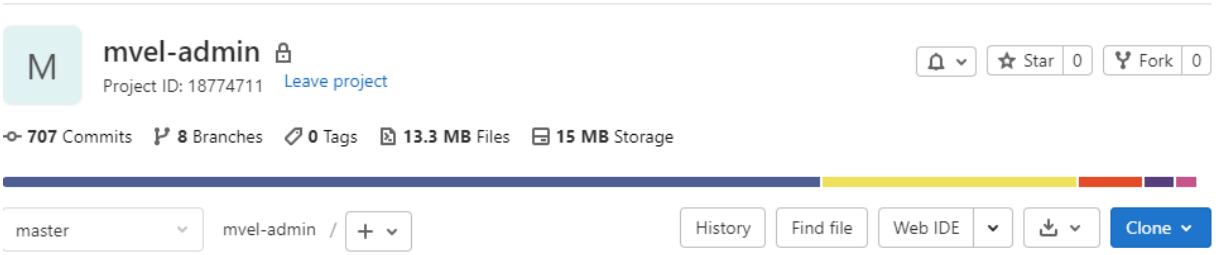
Digidream est une société basée à Strasbourg dont les activités principales sont :

- La publicité sur les réseaux sociaux
- La création de site internet vitrine
- La création de site marchand click & collect et E-commerce

C'est sur cette dernière activité que je suis intervenu dans le cadre de mon stage de deuxième année.

Présentation du projet

Mvel (mes ventes en ligne) est une application essentiellement écrite en PHP et basée sur la Framework Koseven (Koseven a Kohana Framework fork compatible with PHP7 | koseven.ga).



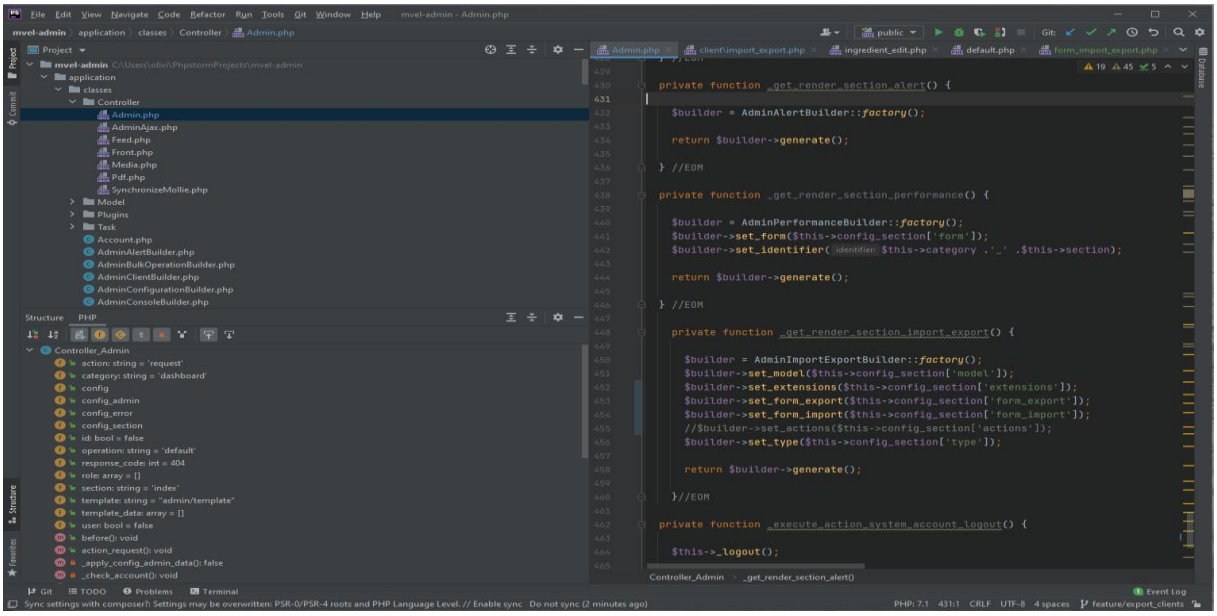
Le projet est hébergé sur la forge en ligne Gitlab.com. Le travail est distribué sous forme de ticket.

Travail effectué

Le stage a été réalisé seul et en télétravail. La communication se faisait par visioconférence et partage d'écran ou par message (Slack). Le code a été versionné à l'aide de Git et envoyé sur la forge en ligne GitLab.

Installation et découverte

Les premiers jours ont été accès sur la découverte du code et de Koseven, ainsi que la mise en place sur mon serveur Ubuntu de l’application.



Premier ticket

Pour essayer de me familiariser avec ce Framework, mon premier ticket a été la recherche d'une faute de frappe dans la partie du backend administratif. L'ensemble de l'application est un immense puzzle, il y a des « morceaux » de code partout avec des utilités différentes. Par exemple, il n'existe qu'un seul bouton dont le « morceau » de code est réutilisé à chaque fois que nous avons besoin d'un « bouton ». Le but étant surtout de pouvoir tout réutiliser. Mais comme je n'avais pas la vision globale du code, la tâche n'a pas été facile.

Une fois le bon fichier trouvé, la correction a été simple, il suffisait d'enlever le « \ » en trop dans le texte descriptif

```
70      'description' => __("Les groupes d'ingrédients vous permettent d'organiser comme bon vous semble vos ingrédients."),
71      'icon' => 'info',
72      'title' => __('Informations'),
73      'type' => 'info',
```

Export de clients

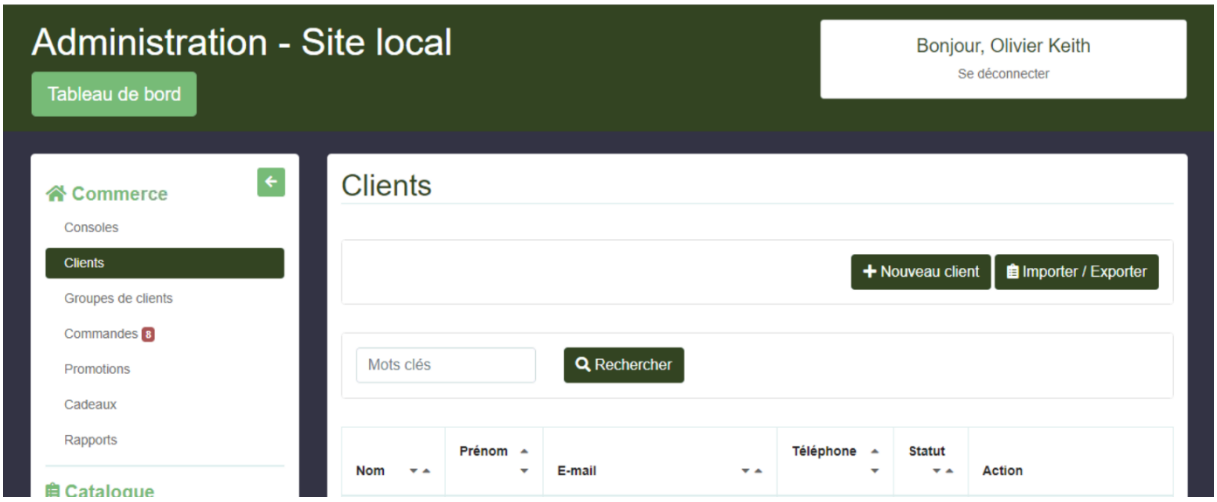
Le travail suivant avait pour objectif la mise en place d'une nouvelle fonctionnalité au site : la possibilité de pouvoir importer ou exporter de la base de données « les clients ». Le travail devait se faire en deux étapes. La première étape était la mise la place de l'export des clients, avec la possibilité de pouvoir choisir le ou les champs à exporter selon la liste de champs suivante : noms, prénoms, identifiant, adresses mails, numéro de téléphone et accord sur le RGPD des clients.

Le bouton « import / Export »

J'ai tout d'abord mis en place le bouton « import/export » et donc trouver le bon fichier de configuration pour y ajouter la configuration de notre bouton.

```php
[
    'color' => 'primary',
    'full_width' => false,
    'icon' => 'clipboard-list',
    'icon_only' => false,
    'id' => false,
    'label' => __("Importer / Exporter"),
    'category' => 'trading',
    'section' => 'client',
    'operation' => 'import_export',
    'permissions' => [
      'client_import_export',
    ],
    'type' => 'request_admin'
]
```

Et nous obtenons le résultat suivant dans le backoffice : le bouton « import / Export » est placé.



Ensuite, il a fallu mettre en place la fonctionnalité d'export des clients de la base de données en fonction des champs choisis pour l'export.

Page de configuration

Tout d'abord, il a fallu mettre en place la page de la configuration de la liste des champs à exporter

```php
<?php

return [
    'fields' => [
        'id' => __("Identifiant"),
        'lastname' => __("Nom"),
        'firstname' =>  __("Prénom"),
        'phone' => __("Téléphone"),
        'email' => __("Email"),
        'rgpd_accepted' => __("Accord RGPD"),
    ],
    'permissions' => [
        'client',
        'client_import_export',
    ],
    'model' => 'Client',
    'request_id' => false,
    'template' => 'sections/form_import_export',
    'title' => __("Importer / exporter des clients"),
    'type' => 'import_export',
];
```

Comme le montre la capture précèdente, il a fallu mettre en place un nouveau Template : « sections/form_import_export », ceci afin de nous appuyer sur une classe qui existe déjà et qui gère tous les formulaires de l'application.

Configuration du formulaire de l'import/export

```php
'form_export' => [
    'actions' => [
        [
            'color' => 'primary',
            'full_width' => false,
            'icon' => 'cog',
            'icon_only' => false,
            'label' => __("Exporter"),
            'type' => 'submit'
        ],
    ],
```

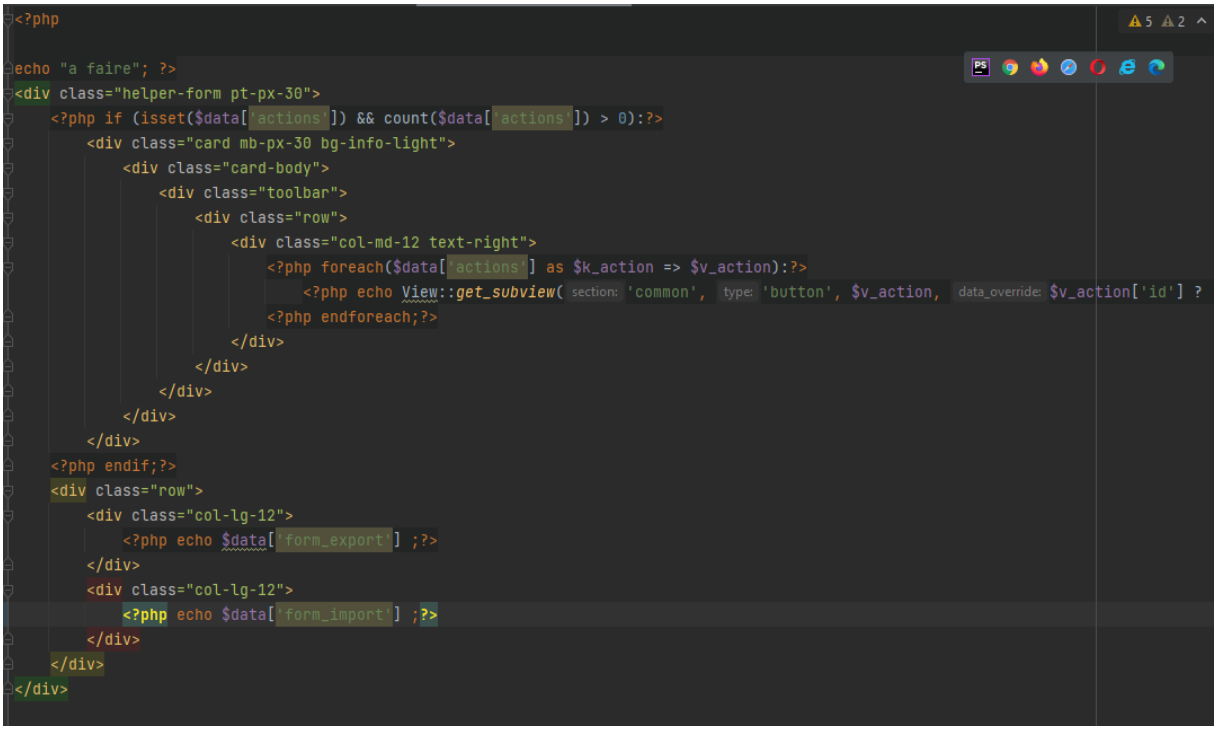
J'ai commencé par ajouter une « action », un bouton permettant de valider le formulaire (ci-contre)

J'ai ensuite défini tous les autres champs qui formerons le formulaire comme déterminé par le pattern du site, sans les champs qui ont déjà été définis dans notre page de configuration.

Voici donc les champs dont la classe « Form » a besoin pour fonctionner

```
    'fields' => [
        [
            'icon' => 'list-alt',
            'label' => __("Exporter"),
            'fieldsets' => [
                [
                    'fields' => [
                        [
                            'label' => __("Champs à exporter"),
                            'name' => 'export_fields',
                            'options' => [
                                'type' => 'custom',
                            ],
                            'type' => 'checkboxes',
                            'value' => '',
                        ],
                        [
                            'label' => __("Format de l'export"),
                            'name' => 'extensions',
                            'options' => [
                                'method' => 'get_options_extensions',
                                'type' => 'config',
                            ],
                            'type' => 'select',
                            'value' => '',
                        ],
                    ],
                ],
            ],
        ],
    ],
],
```

Le « Template » ainsi configuré, il fallait mettre en place un outil dans la « Vue » : un « widget » qui va recevoir toutes les informations du contrôleur pour pouvoir les afficher.



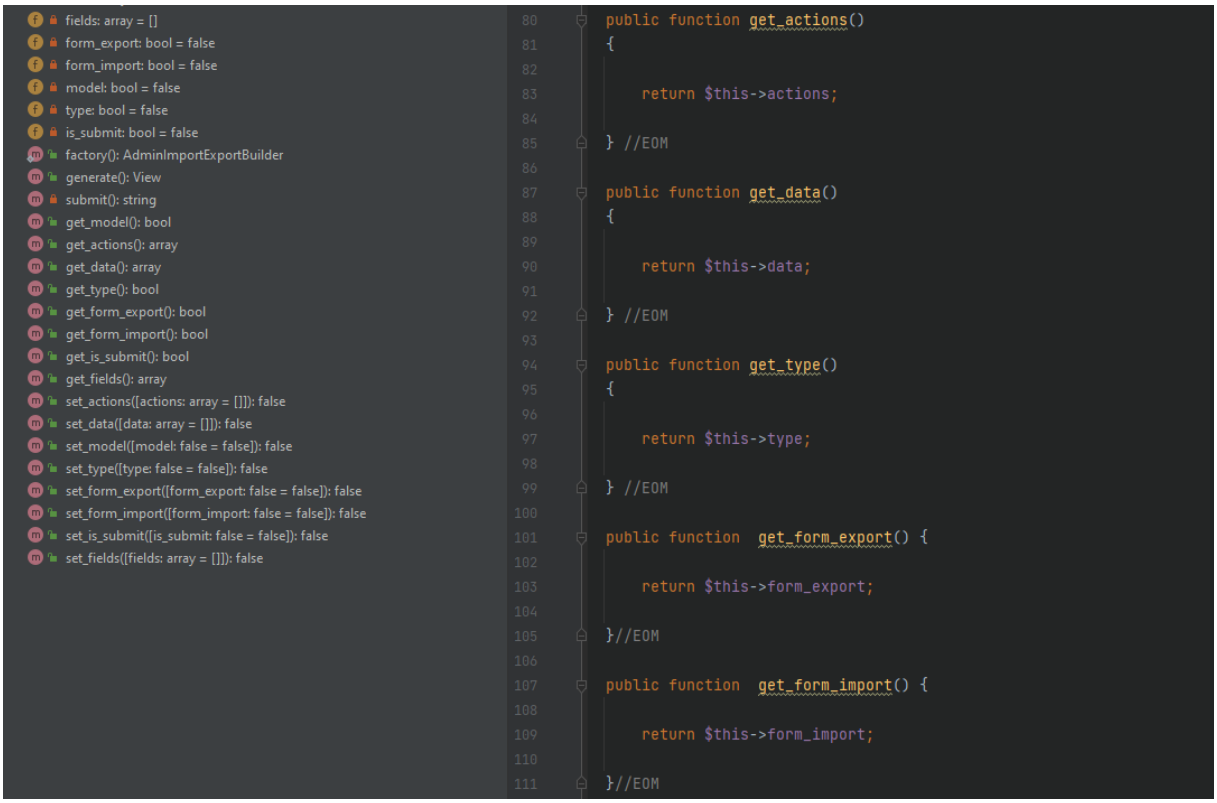
```php
<?php

echo "a faire"; ?>
<div class="helper-form pt-px-30">
    <?php if (isset($data['actions']) && count($data['actions']) > 0):?>
        <div class="card mb-px-30 bg-info-light">
            <div class="card-body">
                <div class="toolbar">
                    <div class="row">
                        <div class="col-md-12 text-right">
                            <?php foreach($data['actions'] as $k_action => $v_action):?>
                                <?php echo View::get_subview( section: 'common', type: 'button', $v_action, data_override: $v_action['id'] ?
                            <?php endforeach;?>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    <?php endif;?>
    <div class="row">
        <div class="col-lg-12">
            <?php echo $data['form_export'] ;?>
        </div>
        <div class="col-lg-12">
            <?php echo $data['form_import'] ;?>
        </div>
    </div>
</div>
```

La classe AdminImportExportBuilder

J'ai créé une classe qui va s'occuper de récupérer les différentes informations reparties dans les différents Template et fichier de configuration. J'ai mis en place les différentes propriétés privées ainsi que les « getters » et les « setters ».

Ils permettront de récupérer et de définir les propriétés du fichier de configuration.

Dans la capture ci-dessous, nous avons par exemple le « getter » sur le bouton « action » ou encore celui sur le formulaire d'export.



Le problème que j'ai rencontré et de devoir prendre les « champs » de la page de configuration pour les placer dans le formulaire d'export. C'est donc ma classe qui s'en occuper

Puis on récupère le reste des informations du formulaire d'export pour générer le formulaire, à l'aide de la classe Form.

```
$form_export = Form::factory();
$form_export->set_identifier( identifier: "toto");
$form_export->set_config($config_export);
$form_export->check_submit();
```

Puis, on fabrique la vue avec tous les éléments dont nous disposons :

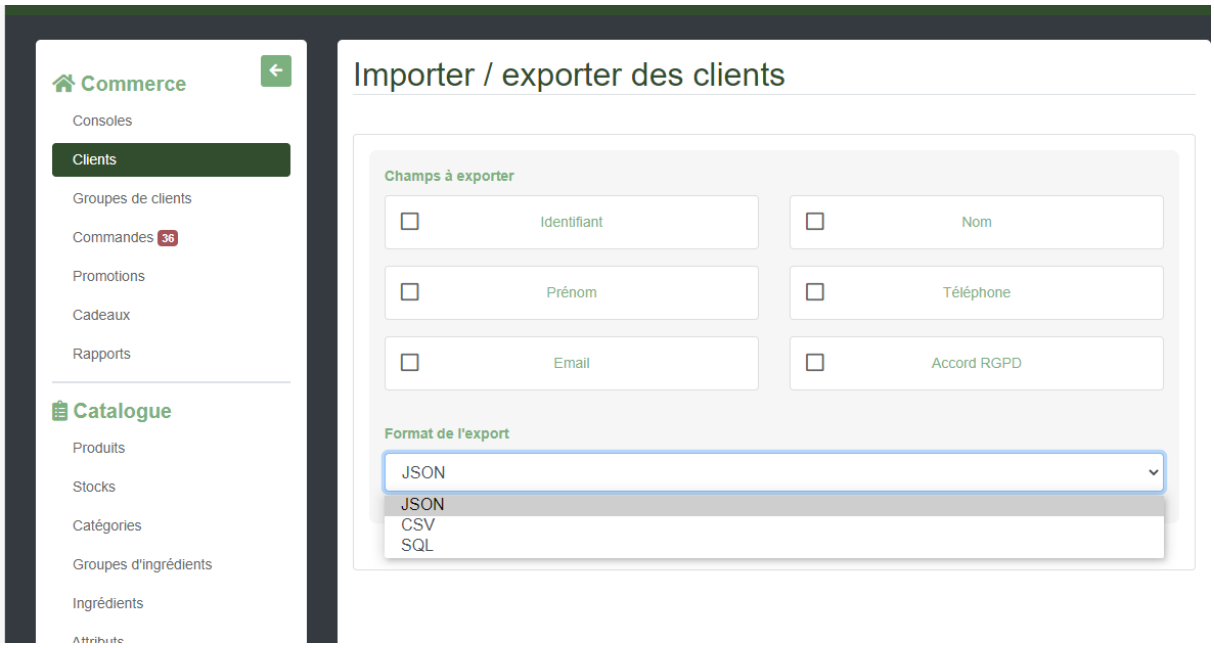
```
$data = [
    'actions' => $this->get_actions(),
    'fields' => $this->get_fields(),
    'form_export' => $form_export->render(),
    'form_import' => $form_import->render(),
];

return View::factory( file: 'admin/widget/import_export', [
    'data' => $data,
]);
```

Le contrôleur

Comme notre modèle est un modèle MVC, c'est le contrôleur qui va s'occuper d'appeler la classe AdminImporExportBuilder lorsque nous en aurons besoin. J'ai donc ajouté une fonction dans le contrôleur.

```
private function _get_render_section_import_export() {

  $builder = AdminImportExportBuilder::factory();
  $builder->set_fields($this->config_section['fields']);
  $builder->set_model($this->config_section['model']);
  $builder->set_form_export($this->config_section['form_export']);
  $builder->set_type($this->config_section['type']);

  return $builder->generate();

}//EOM
```

Voici le résultat



Et une fois les choix faits, le bouton du formulaire apparait :