Comprendre le code de l'application Diagnostic CASES

Table of Contents

Dossiers et fichiers expliqués

Dossier config

Il contient le fichier application.config.php qui sert à ajouter ou modifier des modules. Actuellement il y a deux modules présents dans l'application : 'Diagnostic' et 'Admin'.

On y trouve aussi les fichiers global.php et local.php qui contiennent les variables de configuration de l'application.

Dossier data

Ce dossier contient les modèles templates des rapports qui sont téléchargés lors d'un diagnostic. Il est possible de les modifier si l'on veut changer le contenu visuel du rapport.

Actuellement il y a les modèles anglais et français disponibles.

Dossier documentation

Il contient 3 guides distincts :

- Le guide de démarrage rapide par l'importation d'une machine virtuelle prête à l'utilisation
- Le guide technique qui contient tous les composants à installer
- Le guide d'utilisateur qui explique comment fonctionne l'application Diagnostic

Dossier language

C'est ici que sont entreposés les fichiers de traductions de l'application. Il existe actuellement la langue anglaise et la langue française. Ce sont des formats .po convertis en .mo afin d'être utilisé.

L'application utilise aussi un fichier contenant les codes des pays utilisables 'code_country.txt'ainsi que les langues actuellement utilisées 'languages.txt'.

Dossier module

C'est le dossier principal contenant tout le code de l'application. Il contient actuellement 2 modules, 'Admin' et 'Diagnostic'. Ils permettent de distinguer la partie administrateur de la partie diagnostic utilisable par l'utilisateur. Ces deux modules sont codés de la même manière, à savoir basé sur le modèle MVC.

Sur chacun des modules on retrouve :

- Un dossier config qui va permettre de faire le lien entre tous les fichiers utilisés par le module.
- Un dossier src contenant tous les fichiers utilisés par le module exceptés les vues. Il y a en l'occurrence :
    - Un dossier Controller où sont traités toutes les actions de l'utilisateur sur le Diagnostic.
    - Un dossier Form contenant les formulaires utilisés.
    - Un dossier InputFilter qui créé des conditions de validation de formulaire.
- Un dossier view contenant les vues (les pages) affichées sur le navigateur.
- Un fichier Module.php qui fait le lien avec certains fichiers.

De plus le module Diagnostic contient d'autres fichiers :

- Une vue 'layout' qui affiche les informations de base propre à chaque page de l'application (header, footer, nav).
- Un dossier Gateway qui va chercher les informations liées à la base de données concernant l'utilisateur, les questions et les catégories du Diagnostic.
- Un dossier Model qui va chercher les informations rentrées par l'utilisateur dans l'application.
- Un dossier Service qui contient divers fichiers :
    - 'CalculService.php' qui s'occupe de calculer le résultat d'un diagnostic créé en attribuant des points à chaque question en fonction de leur maturité.
    - 'TemplateProcessorService.php' qui s'occupe de créer un rapport avec les informations du diagnostic.
    - Les fichiers d'informations liés au contenu de l'application. Il y a actuellement, les informations sur les questions, les catégories, les langues, les mails ainsi que les utilisateurs.
- Un dossier Validator qui valide la puissance du mot de passe pour se connecter au diagnostic.

Dossier node_modules

Il contient les extensions utilisées par l'application, en l'occurrence il y a 'chart.js' qui sert à la création de graphique.

Dossier packer

Il contient tous les scripts liés à l'installation du diagnostic ainsi que de tous ses composants.

Dossier public

Il contient tout le nécessaire graphique qui est utilisée par l'application, à savoir : le css (avec utilisation de bootstrap), les drapeaux des langues, les images utilisées dans le diagnostic, les graphiques en javascript.

Il y a aussi le fichier 'index.php', qui sert à démarrer l'application.

Dossier scripts

Il contient tous les composants à installer pour faire fonctionner le diagnostic.

Fichier composer.json

C'est ici qu'est installée la version actuelle de Php et du framework Zend.

Fichier init_autoloader.php

Ce fichier relie l'application avec le framework Zend afin de pouvoir l'utiliser.

Fichier install.sh

Ce fichier permet l'installation du diagnostic ainsi que de tous ses composants nécessaires.

Fichier package.json

Sert à l'installation de package nécessaire au bon fonctionnement de l'application. On y retrouve entre autre le package 'chart.js' pour création de graphique.

Exemple de modification de code

Cet exemple va vous montrer et vous expliquer comment a été créée la partie catégorie présente dans l'application. Tout le code ne sera pas expliqué, mais des commentaires sont présents dans les fichiers de code afin d'expliquer au mieux les parties précises.

Les screenshots présents dans cet exemple ne sont pas des codes complets et ne sont là que pour vous faire comprendre le mieux possible comment fonctionne le code de l'application. Je vous invite à ensuite parcourir le code de l'application pour le comprendre du mieux possible.

Commençons par les vues :

Vue 'categories'

```php
<?php
$title = $this->translate('__categories');
$this->headTitle($title);
$_SESSION['nb_categories'] = 1; // Count number of category
?>

<!-- // Display confirmation if you want to delete a category -->
<script>var delete_category = <?php echo json_encode($this->translate('__delete_category')) ?></script>

<div class="row">
    <div class="col-lg-2"></div>
    <div class="col-lg-8 text-left">
        <h2><?= $this->translate('__categories') ?></h2>
        <br/>
        <?php
        $urlAdd = $this->url('admin', ['action' => 'add-category']);
        ?>
        <center>
            <a href="<?= $urlAdd ?>" style="color: #DF1D31; font-weight: bold;"><span
                        class="glyphicon glyphicon-plus-sign"></span> <?= $this->translate('__add_a_category') ?></a>
        </center>
        <br/>
        <table class="table">
            <tr>
                <th style="text-align: center"><?= $this->translate('__category'); ?></th>
                <th><?= $this->translate('__translation_key'); ?></th>
                <th style="text-align: center;"><?= $this->translate('__action'); ?></th>
            </tr>
            <?php
            foreach ($categories as $category) :
                $_SESSION['nb_categories']++;
                $urlModify = $this->url('admin', ['action' => 'modify-category', 'id' => $category->getId()]);
                $urlDelete = $this->url('admin', ['action' => 'delete-category', 'id' => $category->getId()]);
            ?>
                <tr>
                    <td align="center"><?= $this->translate($category->getTranslationKey()); ?></td>
                    <td><?= $category->getTranslationKey(); ?></td>
                    <td align="center">
                        <a href="<?= $urlModify ?>"><span class="glyphicon glyphicon-pencil"></span></a>
                          
                        <a href="<?= $urlDelete ?>" onclick='return confirm(delete_category)'><span class="glyphicon glyphicon-remove"></span></a>
                    </td>
                </tr>
            <?php
            endforeach;
            ?>
        </table>
    </div>
```

On remarque l'utilisation de Zend, qui sera présente dans tout le code de l'application.

Chaque $this->translate() fait appelle à la fonction translate qui va traduire le mot passé en paramètre dans la langue actuelle, à condition qu'il soit présent dans les fichiers de traduction.

La variable $categories est le tableau contenant toutes les catégories du diagnostic qui sont créées dans un autre fichier.

Les url d'action servent à changer de pages (ex : $this->url('admin', ['action' => 'add-category']) vous fera vous déplacer à la page 'ajouter une catégorie').

Vue ‘add-category’

```
$this->form->setAttribute('action', $this->url('admin', ['action' => 'add-category']));
$this->form->prepare(); ?>

<div class="row">
    <div class="col-lg-3 text-center"></div>
    <div class="col-lg-6 text-left">
        <h2><?= $this->translate('__add_a_category') ?></h2>
        <br />

        <?= $this->form()->openTag($form) ?>

        <table class="table table-responsive table-borderless">
            <tr>
                <td><label><?= $this->translate($form->get('translation_key')->getLabel()) ?></label></td>
                <td>
                    <?= $this->formText($form->get('translation_key')->setValue('__category' . $_SESSION['nb_categories'])) ?>
                </td>
            </tr>
```

Dans cette vue on ouvre un formulaire initialisé dans le fichier ‘CategoryForm’, qui va nous permettre de l’utiliser. La première ligne de code renvoie tout action sur un bouton au controller qui va exécuter le code correspondant. On peut ensuite créer notre formulaire en utilisant $form->get().

La vue ‘modify_category’ est faite de la même manière si ce n’est que l’on va récupérer l’id de la catégorie que l’on veut modifier.

Modèle ‘CategoryForm’

```
class CategoryForm extends Form
{
    /**
     * Categories
     *
     * @var array
     */
    protected $categories = [];

    /**
     * @return array
     */
    public function getCategories()
    {
        return $this->categories;
    }

    /**
     * @param array $categories
     * @return CategoryForm
     */
    public function setCategories($categories)
    {
        $this->categories = $categories;
        return $this;
    }
```

```
public function init()
{
    $this->add([
        'name' => 'translation_key',
        'type' => 'Text',
        'required' => true,
        'options' => [
            'label' => '__translation_key'
        ],
        'attributes' => [
            'class' => 'form-control',
        ]
    ]);

    $this->add([
        'type' => 'Csrf',
        'name' => 'csrf',
        'options' => [
            'csrf_options' => [
                'timeout' => 3600
            ]
        ]
    ]);

    $this->add([
        'name' => 'submit',
        'type' => 'Submit',
        'attributes' => [
            'value' => '__add',
            'id' => 'submitbutton',
            'class' => 'btn btn-success',
        ],
    ]);
}
```

Dans ce fichier nous avons d'une part les deux fonctions qui permettent de récupérer et de modifier les catégories du diagnostic, et d'autre part nous avons l'initialisation du formulaire de catégorie.

Modèle 'CategoryFormFactory'

```php
<?php

namespace Admin\Form;

use Admin\InputFilter\CategoryFormFilter;
use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

/**
 * Category Form Factory
 *
 * @package Admin\Factory
 * @author Romain DESJARDINS
 */
class CategoryFormFactory implements FactoryInterface
{
    /**
     * Create service
     *
     * @param ServiceLocatorInterface $serviceLocator
     * @return mixed
     */
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $categories = $serviceLocator->getServiceLocator()->get('Diagnostic\Service\CategoryService')->getCategories();

    //retrieve categories
        $tab_categories = [];
        foreach ($categories as $category) {
            $tab_categories[$category->getId()] = $category->getTranslationKey();
        }

        $form = new CategoryForm();
        $form->setCategories($tab_categories);

        $categoryFormFilter = new CategoryFormFilter($serviceLocator->getServiceLocator()->get('Zend\Db\Adapter\Adapter'));
        $form->setInputFilter($categoryFormFilter);

        return $form;
    }
}
```

C'est cette fonction qui créé le tableau des catégories et qui renvoie le formulaire correspondant.

Modèle 'CategoryFormFilter'

```php
<?php
namespace Admin\InputFilter;

use Zend\InputFilter\InputFilter;
use Zend\Validator\Hostname;

/**
 * Category Form Filter
 *
 * @package Admin\Form
 * @author Romain DESJARDINS
 */
class CategoryFormFilter extends InputFilter
{
    public function __construct($adapter)
    {
        $this->add([
            'name' => 'translation_key',
            'required' => true,
            'validators' => [
                [
                    'name' => 'StringLength',
                    'options' => [
                        'min' => 6
                    ],
                ],
            ],
        ]);
    }
}
```

Cette fonction permet de définir des conditions liées au formulaire de catégorie. En l'occurrence, le champ ne doit pas être vide et doit dépasser les 6 caractères.

Après avoir créé ces formulaires, il faut maintenant les relier aux autres fichiers du code, il faut donc modifier le fichier config du module admin.

Module 'module.config'

```
'form_elements' => [
    'factories' => [
        'AdminQuestionForm' => 'Admin\Form\QuestionFormFactory',
        'AdminCategoryForm' => 'Admin\Form\CategoryFormFactory',
        'AdminLanguageForm' => 'Admin\Form\LanguageFormFactory',
        'AdminAddTranslationForm' => 'Admin\Form\AddTranslationFormFactory',
    ],
    'invokables' => [
        'UserForm' => 'Admin\Form\UserForm',
        'NewQuestionForm' => 'Admin\Form\QuestionForm',
        'NewCategoryForm' => 'Admin\Form\CategoryForm',
        'NewLanguageForm' => 'Admin\Form\LanguageForm',
        'NewAddTranslationForm' => 'Admin\Form\AddTranslationForm',
    ],
],
```

On rajoute donc les nouveaux formulaires créés.

Ensuite, il faut récupérer les catégories de la base de données.

Modèle 'CategoryGateway'

```
class CategoryGateway extends AbstractGateway
{
    public function fetchAllWithCategories()
    {

        $select = $this->tableGateway
            ->getSql()
            ->select()
            ->columns(['id', 'translation_key']);

        $resultSet = $this->tableGateway->selectWith($select);

        return $resultSet;
    }


    public function getCategoryById($id)
    {

        $select = $this->tableGateway
            ->getSql()
            ->select()
            ->where(['id = ?' => $id]);

        $resultSet = $this->tableGateway->selectWith($select);

        return $resultSet;
    }


    public function update($id, $data)
    {
        $this->tableGateway->update([
            'translation_key' => $data['translation_key']
        ], ['id' => $id]);
    }
}
```

Ces 3 fonctions servent respectivement à récupérer les catégories dans la base de données, à récupérer une catégorie par son id et à mettre la jour la table lors d'une modification d'une catégorie.

Il nous faut ensuite créer l'entité de la catégorie, qui contiendra toutes les fonctions nécessaires afin de pouvoir l'utiliser.

Modèle 'CategoryEntity'

```php
class CategoryEntity
{
    /**
     * Id
     */
    public $id;

    /**
     * Translation Key
     */
    public $translation_key;

    /**
     * New
     */
    public $new = false;

    /**
     * @return mixed
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @param mixed $id
     * @return CategoryEntity
     */
    public function setId($id)
    {
        $this->id = $id;
        return $this;
    }

    /**
     * @return mixed
     */
    public function getTranslationKey()
    {
        return $this->translation_key;
    }
```

```php
public function setTranslationKey($translation_key)
{
    $this->translation_key = $translation_key;
    return $this;
}

/**
 * @return mixed
 */
public function getNew()
{
    return $this->new;
}

/**
 * @param mixed $new
 * @return CategoryEntity
 */
public function setNew($new)
{
    $this->new = $new;
    return $this;
}

/**
 * @param array $data
 */
public function exchangeArray($data)
{
    if (is_object($data)) {
        $data = (array)$data;
    }

    //id
    if (isset($data['id'])) {
        $this->id = $data['id'];
    } else {
        $this->id = null;
    }

    $this->translation_key = (isset($data['translation_key'])) ? $data['translation_key'] : null;
    $this->new = (isset($data['new'])) ? $data['new'] : false;
}
```

Il nous faut ensuite un fichier qui opère sur la base de données et envoie les informations à l'application.

Modèle 'CategoryService'

```php
class CategoryService extends AbstractService
{
    protected $config;

    /**
     * Fetch all with categories
     *
     * @return \Zend\Db\ResultSet\ResultSet
     * @throws \Exception
     */
    public function fetchAllWithCategories()
    {
        $tableGateway = $this->get('gateway');

        return $tableGateway->fetchAllWithCategories();
    }

    /**
     * Get category
     *
     * @return array
     * @throws \Exception
     */
    public function getCategories()
    {
        $container = new Container('diagnostic');
        if ($container->offsetExists('categories')) {
            $categories = $container->categories;
        } else {
            $questionsObject = $this->fetchAllWithCategories();

            $categories = [];
            foreach ($questionsObject as $question) {
                $categories[$question->getId()] = $question;
            }

            $container->categories = $categories;
        }

        return $categories;
    }
```

```php
public function getBddCategories()
{
    $categoriesObject = $this->fetchAllWithCategories();

    $categories = [];
    foreach ($categoriesObject as $category) {
        $categories[$category->getId()] = $category;
    }

    return $categories;
}

/**
 * Create
 *
 * @param $data
 */
public function create($data)
{
    $categoryGateway = $this->get('gateway');
    $categoryGateway->insert($data);
}

/**
 * Reset Cache
 */
public function resetCache()
{
    $container = new Container('diagnostic');
    $container->offsetUnset('categories');
}

/**
 * Get category by id
 *
 * @return \Zend\Db\ResultSet\ResultSet
 * @throws \Exception
 */
public function getCategoryById($id)
{
    $tableGateway = $this->get('gateway');

    return $tableGateway->getCategoryById($id);
}
```

```php
public function update($id, $data)
{
    unset($data['id']);
    $categoryGateway = $this->get('gateway');
    $categoryGateway->update($id, $data);
}

/**
 * Delete
 *
 * @param $id
 */
public function delete($id)
{
    $categoryGateway = $this->get('gateway');
    $categoryGateway->delete($id);
}
```

On a alors la fonction getBddCategories() qui va renvoyer le tableau contenant toutes les catégories de la base de données. Ce tableau sera ensuite utilisé par les autres fichiers de l'application.

Modèle 'CategoryServiceFactory'

```php
class CategoryServiceFactory extends AbstractServiceFactory
{
    protected $resources = [
        'gateway' => 'Diagnostic\Gateway\CategoryGateway',
        'entity' => 'Diagnostic\Model\CategoryEntity',
        'config' => 'Config',
    ];
}
```

Ce fichier permet de faire des liens entre les autres fichiers.

Enfin, changeons le controller en lui ajoutant les fonctions qui seront exécutées lors de l'appui d'un bouton dans les vues.

Controller 'IndexController'

```php
class IndexController extends AbstractController
{
    protected $userService;
    protected $userTokenService;
    protected $questionService;
    protected $categoryService;
    protected $languageService;
    protected $userForm;
    protected $adminQuestionForm;
    protected $adminCategoryForm;
    protected $adminLanguageForm;
    protected $adminAddTranslationForm;
```

```php
/**
 * Categories
 *
 * @return ViewModel
 */
public function categoriesAction()
{
    //retrieve categories
    $categoryService = $this->get('categoryService');
    $categories = $categoryService->getBddCategories();

    //send to view
    return new ViewModel([
        'categories' => $categories
    ]);
}
```

```php
public function addCategoryAction()
{
    $location_lang = '/var/www/diagnostic/language/';

    // Session value to know if the translation key already exist
    $_SESSION['erreur_exist'] = 0;

    $tabToGet = ['translation_key', 'csrf', 'submit'];

    $form = $this->get('adminCategoryForm');

    //form is post and valid
    $request = $this->getRequest();
    if ($request->isPost()) {
        $form->setData($request->getPost());

        // Determine if the translation key already exist
        $cmd='grep -c -w ' . $request->getPost('translation_key') . ' ' . $location_lang . 'en.po';
        if(exec($cmd) != 0){ $_SESSION['erreur_exist'] = 1;}

        if ($form->isValid() && $_SESSION['erreur_exist'] == 0) {
            $formData = [];
            foreach ($tabToGet as $key) {
                $formData[$key] = $form->getData()[$key];
            }
            $categoryService = $this->get('categoryService');
            $categoryService->create((array)$formData);
            $categoryService->resetCache();
```

On a d'abord la fonction categoriesAction() qui agit sur la page regroupant toutes les catégories. Afin d'afficher les catégories, elle va les chercher grâce à la fonction getBddCategories() du fichier categorySercice puis les stock dans un tableau.

Puis on a la fonction d'ajout de catégorie, qui va vérifier si les informations rentrées répondent bien aux conditions requises du formulaire avant de modifier la base de données. Les fonctions de modification et de suppression ont la même structure.

Faisons ensuite le lien avec les fichiers 'categoryService' et 'CategoryForm' afin de pouvoir les utiliser dans le controller.

Controller 'IndexControllerFactory'

```
class IndexControllerFactory extends AbstractControllerFactory
{
    protected $resources = [
        'userService' => 'Diagnostic\Service\UserService',
        'userTokenService' => 'Diagnostic\Service\UserTokenService',
        'questionService' => 'Diagnostic\Service\QuestionService',
        'categoryService' => 'Diagnostic\Service\CategoryService',
        'languageService' => 'Diagnostic\Service\LanguageService',
    ];

    protected $forms = [
        'user', 'adminQuestion', 'adminCategory', 'adminLanguage', 'adminAddTranslation'
    ];
}
```

Pour finir, il ne reste plus qu'à modifier le fichier de configuration du module diagnostic afin de rajouter les liens nécessaires au fonctionnement de l'application.

Module 'module.config'

```
'service_manager' => [
    'invokables' => [
        'Diagnostic\Model\DiagnosticEntity' => 'Diagnostic\Model\DiagnosticEntity',
        'Diagnostic\Model\InformationEntity' => 'Diagnostic\Model\InformationEntity',
        'Diagnostic\Model\QuestionEntity' => 'Diagnostic\Model\QuestionEntity',
        'Diagnostic\Model\CategoryEntity' => 'Diagnostic\Model\CategoryEntity',
        'Diagnostic\Model\UserEntity' => 'Diagnostic\Model\UserEntity',
        'Diagnostic\Model\UserTokenEntity' => 'Diagnostic\Model\UserTokenEntity',
        'Diagnostic\Service\Mime\Part' => 'Zend\Mime\Part',
        'Diagnostic\Service\Mime\Message' => 'Zend\Mime\Message',
        'Diagnostic\Service\Mail\Message' => 'Zend\Mail\Message',
        'Diagnostic\Service\Mail\Transport\Smtp' => 'Zend\Mail\Transport\Smtp',
        'Diagnostic\Service\Mail\Transport\SmtpOptions' => 'Zend\Mail\Transport\SmtpOptions',
    ],
    'abstract_factories' => [
        'Zend\Cache\Service\StorageCacheAbstractServiceFactory',
        'Zend\Log\LoggerAbstractServiceFactory',
    ],
    'factories' => [
        'translator' => 'Zend\Mvc\Service\TranslatorServiceFactory',
        'navigation' => 'Zend\Navigation\Service\DefaultNavigationFactory',
        'Diagnostic\Service\QuestionService' => 'Diagnostic\Service\QuestionServiceFactory',
        'Diagnostic\Service\CategoryService' => 'Diagnostic\Service\CategoryServiceFactory',
        'Diagnostic\Service\LanguageService' => 'Diagnostic\Service\LanguageServiceFactory',
        'Diagnostic\Service\CalculService' => 'Diagnostic\Service\CalculServiceFactory',
        'Diagnostic\Service\MailService' => 'Diagnostic\Service\MailServiceFactory',
        'Diagnostic\Service\UserService' => 'Diagnostic\Service\UserServiceFactory',
        'Diagnostic\Service\UserTokenService' => 'Diagnostic\Service\UserTokenServiceFactory',
    ],
    'shared' => [
        'Diagnostic\Model\QuestionEntity' => false,
        'Diagnostic\Model\CategoryEntity' => false,
    ],
],
```

```
'navigation' => [
    'default' => [
        [
            'label' => '__users',
            'route' => 'admin',
            'action' => 'users',
        ],
        [
            'label' => '__questions',
            'route' => 'admin',
            'action' => 'questions',
        ],
        [
            'label' => '__categories',
            'route' => 'admin',
            'action' => 'categories',
        ],
        [
            'label' => '__languages',
            'route' => 'admin',
            'action' => 'languages',
        ],
    ],
],
```

Ce screenshot ci-dessus initialise la barre de navigation principale.

Voilà, cet exemple étant terminé, vous devrez être maintenant capable de modifier le code sans trop de problèmes !