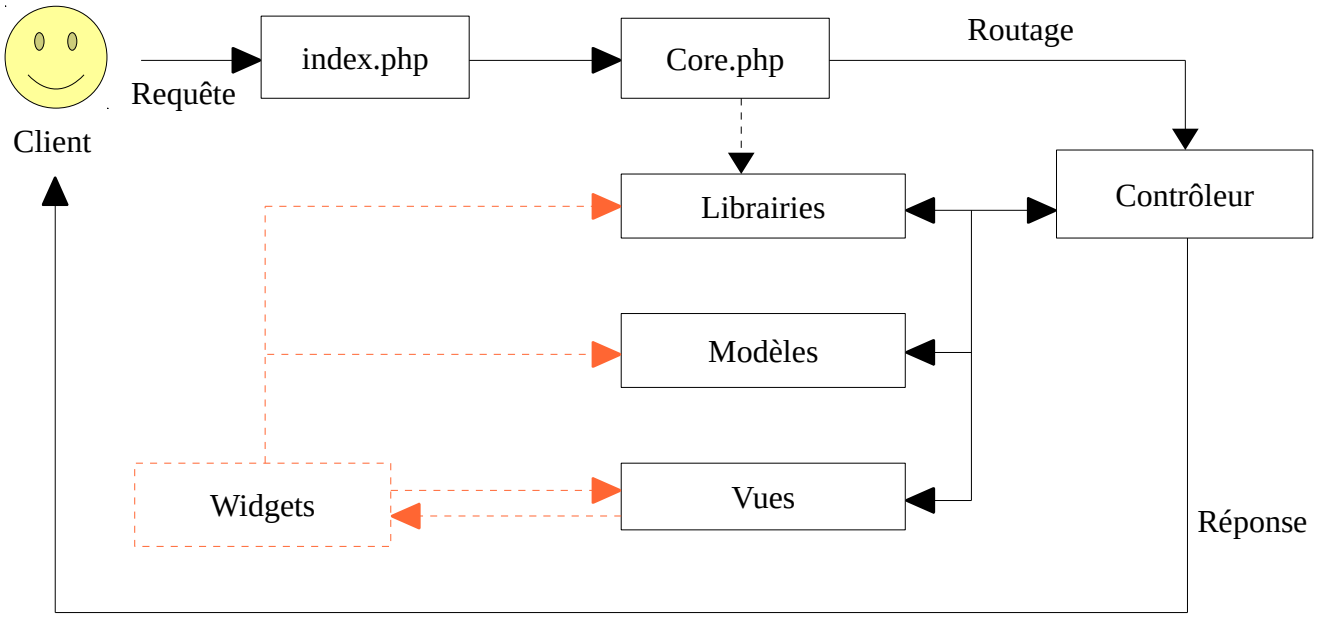
Documentation Framework TimeLine

Sommaire :

I. L'architecture :

1. Schéma global simplifier du framework :



Le fonctionnent réelle de l'application diffère du schéma ci-dessus mais le schéma permet néanmoins de comprendre l'idée de fonctionnent du framework.

La partie en orange sur le schéma signifie qu'elle n'est pas encore implémenté dans le système.

2. Présentation des dossiers :

Répertoire courant : ./

apps : Contient tous les dossiers et fichiers relatifs à l'application (front-end).

public : Contient tous les dossiers et fichiers que l'utilisateur peut accéder directement (exemple : images, feuille de style, javascript …).

system : Contient les dossiers et fichiers relatifs au fonctionnent du système ou aux ressources disponible pour l'application (back-end).

Réperoire courrant : ./apps/

configs : Contient les fichiers de configurations statiques pour l'application (voir VI).

hooks : Contient les hooks de l'application

langs : Contient les fichiers de localisation pour l'application (voir VIII).

models : Contient les modèles pour l'application (voir III).

modules : Contient les contrôleurs de l'application (voir II).

routes : Contient les fichiers gérant les routes personnalisés (indisponible).

views : Contient les vues pour l'application (voir IV).

widgets : Contient les widgets de l'application (voir VII) (indisponible).

Répertoire courant : ./system/

c**aches :** Contient les fichiers caches (indisponible).

configs : Contient les fichiers de configurations statiques du système.

helpers : Contient les helpers disponible (voir V).

includes : Contient des fichiers supplémentaire qui peuvent être inclus dans le système.

libraries : Contient les librairies disponible (voir IX).

II. Les contrôleurs/modules :

1. Créer un nouveau contrôleur :

Pour créer un nouveau contrôleur, commencer pas vous placez dans le dossier apps/modules du framework puis créer un fichier qui portera le nom de votre module avec un majuscule (par exemple si je veut créer un contrôleur pour un module de d'utilisateur, je vais créer un nouveau fichier et le nommé Utilisateurs.php ou Users.php).

Le fichier créé doit par la suite contenir une classe qui porte le même nom que le fichier (sans le .php) et qui étend la classe « Controller ». Cette classe doit contenir au moins la méthode par défaut défini dans le fichier de configuration système (system/configs/settings.php). Par défaut il s'agit de la méthode « index ».

Si je reprend mon exemple, voici le contenu minimum du contrôleur « Utilisateurs » :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Utilisateurs extends Controller {

    public function index() {
        echo 'Hello world';
    }

}

/* End of file Utilisateurs.php */
/* Location : ./apps/modules/Utilisateurs.php */
```

Pour accéder au contrôleur depuis le navigateur il faudra préciser les paramètres « m » et « a » (par défauts).

Le paramètre « m » correspond au nom du module et donc du contrôleur (toujours sans le .php), mais contrairement au contrôleur, le nom du module peut être écrit en minuscule. Si le paramètre « m » n'est pas préciser dans l'url alors c'est le contrôleur par défaut qui sera chargé.

Le paramètre « a » correspond au nom de la méthode à appelé dans le contrôleur. Si le paramètre n'est pas préciser alors la méthode par défaut sera exécuter (index).

Si l'on prend le contrôleur suivant :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Utilisateurs extends Controller {

    public function index() {
        $this->hello();
    }

    public function hello() {
        echo 'Hello world';
    }

    public function bonjour() {
        echo 'Bonjour tous le monde';
    }

}

/* End of file Utilisateurs.php */
/* Location : ./apps/modules/Utilisateurs.php */
```

Si l'url est la suivante : http://example.fr/index.php?m=utilisateurs&a=hello
Alors la page va afficher « Hello world ».

Si l'url est la suivant : http://example.fr/index.php?m=utilisateurs&a=bonjour
Alors la page va afficher « Bonjour tous le monde ».

Si l'url est la suivante : http://example.fr/index.php?m=utilisateurs
Alors la page va afficher « Hello world ».

1. Gérer l'authentification et les droits d'accès *(prochainement)*

2. Créer des routes personnalisées *(prochainement)*

III. Les modèles :

1. Comment créer un nouveau modèle :

Pour créer un nouveau modèle, commencer par vous placez dans le répertoire « apps/models » de l'application puis créer un fichier PHP qui portera le nom du modèle que vous souhaiter créer (avec une majuscule au début) suivit de _model (par exemple si je veut créer un modèle Utilisateurs je vais créer un nouveau fichier Utilisateurs_model.php dans le dossier apps/models).

Le fichier doit par la suite contenir une classe du même nom que le fichier et qui étend la classe « Model ». Contrairement au contrôleur, la classe n'a pas besoin d'avoir de méthode par défaut.

Par la suite vous pouvez ajouter simplement vos méthodes dans le modèle.

Si je reprend mon exemple, voici le contenu du modèle « Utilisateurs » :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Utilisateurs_model extends Model {

    public function ajouterDeux($a) {
        return $a + 2;
    }

}

/* End of file Utilisateurs_model.php */
/* Location : ./apps/models/Utilisateurs_model.php */
```

2. Charger et utiliser un modèle depuis le contrôleur :

Pour utiliser un modèle depuis le contrôleur, ouvrer le fichier de votre contrôleur et placez vous à l'endroit où vous voulez utiliser le modèle.

Avant de pouvoir utiliser le modèle il faut le charger, pour cela nous allons utiliser la fonction « model » de la librairie « load » qui possède la spécification suivant :

*Load ::model(**$model**:string [, **$alias**:string])*

$model : Le nom du model que vous souhaiter charger (avec ou sans majuscule).

$alias : Le nom de la variable dans lequel sera instancié le modèle.

Pour utiliser cette fonction dans le contrôleur il faut taper :

$this->load->model('model', 'alias') ;

Si aucun alias n'est spécifier alors le nom de la variable sera le contenu du premier paramètre.

Si je reprend mon exemple, voici le contenu du contrôleur « Utilisateurs » qui fait appel au modèle « Utilisateurs » :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Home extends Controller {

    public function index() {
        $this->load->model('utilisateurs_model', 'utilisateursMng');

        $a = 1;
        $a = $this->utilisateursMng->ajouterDeux($a);

        echo $a;
    }

}
```

Si l'on accède a la méthode « index » du contrôleur alors cela va afficher « 3 ».

3. Utiliser PDO dans un modèle :

Ont utilise principalement les modèles pour communiquer avec la base de donnée, pour cela il est possible d'utiliser PDO grâce à la librairie « Pdodb » du framework.

Dans un premier temps assurez vous d'avoir configuré les informations de connexion à la base de donnée dans le fichier « system/configs/database.php ». Par la suite charger la librairie « Pdodb » dans le contrôleur (ou dans la mesure où il s'agit d'une librairie qui peut-être utilisé à chaque page, il peut-être intéressant de la rajouter à l'autoloader).

$this->load->library('Pdodb') ;

Une fois la librairie chargé, utiliser la function « loadPDO » de la librairie « Pdodb » dans la fonction du modèle où vous voulez utiliser PDO pour récupérer une référence d'une instance de PDO correspond à votre base de donnée que vous stockerez dans une variable.

$pdo =& $this->pdodb->loadPDO() ;

Vous pourrez par la suite utiliser la variable « $pdo » pour faire appel au fonction de PDO.

Remarque : Par convention, un modèle est égal à une table dans la base de donnée et le nom du modèle porte le nom de la table dans la base de donnée avec une majuscule en début.

Exemple d'un modèle qui utilise PDO :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Utilisateurs_model extends Model {

    public function readUtilisateurs() {
        $pdo =& $this->pdodb->loadPDO();

        $req = $pdo->query('SELECT id, name, email FROM utilisateurs');
        return $req->fetchAll(PDO::FETCH_CLASS);
    }

}

/* End of file Utilisateurs_model.php */
/* Location : ./apps/models/Utilisateurs_model.php */
```

Le contrôleur :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Home extends Controller {

    public function index() {
        $this->load->library('pdodb');
        $this->load->model('utilisateurs_model', 'utilisateursMng');

        $data = $this->utilisateursMng->readUtilisateurs();

        foreach ($data as $value) {
            echo $value->id . ' - ' . $value->name . ' - ' . $value->email . '<br />';
        }
    }

}
```

Le résultat :

1 - Jean - jean.paul@gmail.com
2 - Claude - claude.gilbert@yahoo.fr

IV. Les vues :

1. Comment créer une nouvelle vue :

Pour créer une vue, placez vous dans le dossier « views » du répertoire « apps » puis créer un nouveau fichier PHP. Ce fichier PHP devra contenir essentiellement du code HTML. Il ne doit posséder aucune déclaration de fonction néanmoins il est possible d'utiliser avec modération les conditions ainsi que les boucles ainsi que des « echo ». Il peut arriver que l'on est besoin de réalisé des actions plus complexes, dans ce cas il faut utiliser les helpers (voir V).

Exemple d'une vue :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed'); ?>

<!DOCTYPE HTML>
<html>
<head>
    <title>Liste des utilisateurs</title>
    <meta charset="utf-8" />
</head>
<body>
    <table style="border:1px solid #000;">
        <tr><th>ID</td><th>Nom</th><th>E-mail</th></tr>
        <?php foreach ($data as $value) { ?>
            <tr>
                <td><?php echo htmlentities($value->id); ?></td>
                <td><?php echo htmlentities($value->name); ?></td>
                <td><?php echo htmlentities($value->email); ?></td>
            </tr>
        <?php } ?>
    </table>
</body>
</html>
```

Remarque : Dans la vue il est autoriser d'inclure d'autres vues avec les fonctions « include » et « include_once ».

2. Charger une vue depuis le contrôleur :

Pour charger une vue depuis le contrôleur il faut utiliser la méthode « load » de la librairie « view » dont la spécification est la suivant :

*View::load(**$_file**:string [, **$_data**:array [, **$_dir**:string]])*

$_file : Le nom du fichier à charger (avec le .php).

$_data : Les variables à passer à la vue (la vue est chargé en dehors du contrôleur donc si l'on veut afficher des données récupérer dans le contrôleur il faut passer ces données par un tableau.

$_dir : Le répertoire où ce trouve les vues. (par défaut : apps/views)

Pour charger la librairie utiliser la commande : *$this->load->library('view') ;*

En reprenant les exemples précédant, voici un exemple de contrôleur :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Home extends Controller {

    public function index() {
        $this->load->library('pdodb');
        $this->load->model('utilisateurs_model', 'utilisateursMng');

        $data = $this->utilisateursMng->readUtilisateurs();

        $this->load->library('view');
        $this->view->load('utilisateur_index.php', array('data' => $data));
    }

}
```

Le résultat après exécution :

ID	Nom	E-mail
1	Jean	jean.paul@gmail.com
2	Claude	claude.gilbert@yahoo.fr

V. Les helpers :

1. Qu'est ce qu'un helper :

Un helper est un fichier PHP qui va contenir des fonctions permettant d'alléger la taille du code et d'éviter la redondance. Concrètement les helpers contiennent des fonctions utile auxquels on y fait généralement souvent appels. Les helpers permettent aussi d'aider la vue à traiter les informations.

2. Comment créer un nouveau helper :

Pour créer un nouveau helper, placez vous dans le dossier « helpers » du répertoire « system »puis créer un fichier PHP du nom de votre helper. Les noms des fichiers doivent être en minuscule. Une fois le fichier créer il ne vous reste plus cas rajouter les fonctions que vous souhaiter.

Lorsque vous créez une nouvelle fonction, penser à vérifier s'il est n'existe pas déjà avec la fonction « function_exists(...) ».

Contrairement aux contrôleurs et aux modèles vous pouvez directement accéder aux helpers depuis la vue.

Il est possible que depuis une fonction d'un helper vous souhaiter utiliser une librairie du framework, pour cela il existe la fonction « getInstance() » qui renvoie la référence de l'objet globale du framework. Ainsi en faisant la commande :

$FW =& getInstance() ;

Vous allez pouvoir utiliser les librairies chargées auparavant, par exemple :

$FW->config->get('secureKey') ;

Exemple d'un helper :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

if ( ! function_exists('lang')) {

    function lang($line) {
        $FW =& getInstance();

        if ( ! $FW->load->isLoaded('lang'))
            $FW->load->library('lang');

        return $FW->lang->line($line);
    }

}

if ( ! function_exists('encUTF8')) {

    function encUTF8($string) {
        return utf8_encode($string);
    }

}

/* End of file lang.php */
/* Location : ./system/helpers/lang.php */
```

3. Charger un helper depuis le contrôleur :

Pour charger un helper depuis le contrôleur il suffit d'utiliser la méthode « helper » de la librairie « load » dont la spécification est :

Load::helper(**$helper**:string)

$helper : Le nom de l'helper à charger (sans le .php).

Exemple : *$this->load->helper('lang') ;*

Une fois l'helper chargé, vous pouvez faire appel aux fonctions qui sont à l'intérieur depuis n'importe quelle partie du code.

Attention, ne pas confondre les helpers et les modèles.

VI. Les fichiers de configurations statiques :

1. Créer un nouveau fichier de configuration :

Il existe deux types de fichier de configuration statique, les fichiers servant à la configuration système (Framework, librairies, helpers …) et les fichiers servant à la configuration de l'application (modules, vues, modèles …). Dans le premiers cas il faudra ce placer dans le dossier « system/configs » alors que dans le deuxième cas il faudra aller dans le dossier « apps/configs ».

Pour créer un nouveau fichier de configuration il vous faudra donc vous placez dans un premier temps dans le répertoire approprié. Une fois dans le répertoire, créer un nouveau fichier PHP, le nom du fichier doit être en minuscule et ne doit pas déjà être utilisé par un autre fichier de configuration.

Dans le fichier de configuration, commencer par créer une variable « $config » dans lequel vous déclarez un nouveau tableau. Par la suite il ne vous reste plus cas affecter des cases de ce tableau avec vos valeurs de configuration.

Exemple d'un fichier de configuration :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

$config = array();

/**
 * Date format
 * @see http://php.net/manual/fr/function.date.php
 * @default $config['format'] = 'm/d/Y';
 */

$config['day'] = 'm/d/Y';

/**
 * Hour format
 * @see http://php.net/manual/fr/function.date.php
 * @default $config['hour'] = 'H:i';
 */

$config['hour'] = 'H:i';

/* End of file date.php */
/* Location : ./apps/configs/date.php */
```

2. Charger et utiliser les fichiers de configuration depuis le contrôleur :

Pour utiliser un fichier de configuration depuis le contrôleur il faut d'abord chargé le fichier de configuration, pour cela placez vous dans le contrôleur puis utiliser la méthode « load » de la librairie « Config » dont la spécification est la suivante :

*Config::load(**$file**:string [, **$alias**:string])*

$file : Correspond au nom du fichier de configuration (sans .php) que vous voulez charger dans le répertoire « configs ».

$alias : Pour éviter le risque de conflit entre les fichiers de configuration il est possible de spécifier des alias. Si aucun alias n'est spécifié alors l'alias sera « setting ».

Une fois le fichier de configuration chargé, il faut utilisé la méthode « get » de la même librairie pour récupérer la valeur d'un paramètre de configuration.

*Config::get(**$name**:string [, **$alias**:string])*

$name : Nom du paramètre à récupérer.

$alias : Alias dans lequel le paramètre est stocké.

Si le paramètre n'existe pas alors la fonction renvoie une chaine de caractère vide.

Enfin il est aussi possible de redéfinir les paramètres de configuration à la volée grâce à la méthode « set » dont la spécification est la suivante :

*Config::set(**$name**:string, **$value**:mixed [, **$alias**:string])*

$name : Nom du paramètre à récupérer.

$value : La nouvelle valeur que vous voulez affecter.

$alias : Alias dans lequel le paramètre est stocké.

Exemple d'un contrôleur utilisant les fichiers de configurations statique :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Home extends Controller {

    public function index() {
        $this->config->load('date', 'timeformat');

        echo 'We are the ' . date($this->config->get('day', 'timeformat'))
            . ' and it\'s ' . date($this->config->get('hour', 'timeformat')) . '<br />';

        echo 'La langue par défaut est : ' . $this->config->get('localization') . '<br />';

        $this->config->set('day', 'd/m/Y', 'timeformat');

        echo 'Nous somme le ' . date($this->config->get('day', 'timeformat'))
            . ' et il est ' . date($this->config->get('hour', 'timeformat'));
    }

}
```

Résultat :

We are the 10/28/2014 and it's 12:37
La langue par défaut est : french
Nous somme le 28/10/2014 et il est 12:37

VII. Les widgets :

1. Comment créer un nouveau widget *(prochainement) :*

2. Comment charger un widget depuis la vue *(prochainement) :*

VIII. Les localisations :

1. Comment créer un fichier le localisation *(prochainement) :*

2. Charger et utiliser les localisations (*prochainement) :*

IX. Les librairies :

1. Créer une nouvelle librairie :

Pour créer une nouvelle libraire, placez-vous dans le dossier « libraries » du répertoire « system » et créer un nouveau fichier PHP qui portera le nom de votre libraire (avec une majuscule en début). Ouvrer le fichier puis créer une classe qui porte le même nom que le fichier (sans le .php) puis écrivez les méthodes de votre librairie dans cette classe.

Exemple de librairie :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Dateformat {

    // Instance du framework
    private $FW;

    public function __construct() {
        $this->FW =& getInstance();
        $this->FW->config->load('date', 'timeformat');
    }

    public function getDay() {
        return date($this->FW->config->get('day', 'timeformat'));
    }

    public function getHour() {
        return date($this->FW->config->get('hour', 'timeformat'));
    }

}

/* End of file DateFormat.php */
/* Location : ./system/libraries/DateFormat.php */
```

2. Charger et utiliser une librairie depuis le contrôleur :

Pour utiliser une libraire il faut commencer par la chargé. Pour cela on utilise la méthode « library » de la libraire « load » (chargé par défaut) dont la spécification est la suivante :

*Load::library(**$library**:string [, **$params**:array [, **$name**:string]])*

$name : Nom de la librairie à chargé (en minuscule).

$params : Tableau associatif contenant les paramètres du constructeur de la classe.

$name : Nom de la variable ou sera instancié la libraire (par défaut il prend le nom de la librairie en minuscule).

Une fois la libraire chargé il vous suffit de faire appel au méthodes de la librairie.

Exemple d'utilisation d'une librairie :

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Home extends Controller {

    public function index() {
        $this->load->library('dateformat');

        echo 'We are the ' . $this->dateformat->getDay()
            . ' and it\'s ' . $this->dateformat->getHour() . '<br />';

        echo 'La langue par défaut est : ' . $this->config->get('localization') . '<br />';

        $this->config->set('day', 'd/m/Y', 'timeformat');

        echo 'Nous somme le ' . $this->dateformat->getDay()
            . ' et il est ' . $this->dateformat->getHour();
    }

}
```

X. Validation des formulaires :

1. Vérifier les données d'un formulaires :

Lorsque l'on veux utilisé les données dans formulaires, il faut filtrer ces données, vérifier quel contiennent bien ce qui est attendu. Pour réalisé les vérifications il existe la librairie « vForm », cette librairie permet de vérifier les champs d'un formulaire, de gérer les éventuelles erreurs et de filtrer les données des champs.

Pour charger la librairie utiliser la commande « $this->load->library('vform') » dans le contrôleur. Une fois la librairie chargée, la fonction « addRules » de cette librairie vous permet de définir des règles de validation.

*Vform::addRules(**$field**:string [, **$label**:string, **$rules**:string, **$prep**:string]) ;*

$field : Le nom du champ auquel la règle fait référence.

$label : L'appellation du champ (c'est le nom du champ qui sera afficher en cas d'erreur).

$rule : Les règles à appliquer au champ.

$prep : Les filtres à appliquer au champ.

Une fois les règles définies, il faut utiliser la fonction « run » de la librairie pour lancer les vérifications. Cette fonction renverra « TRUE » si le formulaire est conforme aux règles définies et « FALSE » s'il existe des erreurs.

*Vform::run([**$method**:string) ;*

$method : Le nom de la méthode d'envoi du formulaire (« POST » par défaut).

Exemple d'une vérification **sans vForm** :

```php
public function index() {

    $error = array();
    if (isset($_POST['email'])) {
            if (strlen($_POST['email']) < 124) {
                if ( ! preg_match('/^[a-zA-Z0-9-, _]+@[a-zA-Z0-9-, _.]+$/', $_POST['age'])) {
                    $error['email'] = 'Le champ E-mail n\'est pas une adresse email valide.';
                }
            }
            else {
                $error['email'] = 'Le champ E-mail doit faire moins de 124 caractères.';
            }
    }
    else {
        $error['email'] = 'Le champ E-mail est requis.';
    }

    if (isset($_POST['age'])) {
        if (is_numeric($_POST['age']) && floatval($_POST['age']) == intval($_POST['age'])) {
            if (intval($_POST['age']) < 13) {
                $error['age'] = 'Le champ Age doit être supérieur ou égale à 13.';
            }
        }
        else {
            $error['age'] = 'Le champ Age doit être une valeur entière.';
        }
    }
    else {
        $error['age'] = 'Le champ Age est requis.';
    }

    if (count($error) == 0) {
        // Formulaire correcte
    }
    else {
        // Formulaire incorrecte
    }
```

Exemple d'une vérification **avec vForm** :

```php
public function index() {
    $this->load->library('vform');

    $this->vform->addRules('email', 'E-mail', 'required|maxLength[124]|email');
    $this->vform->addRules('age', 'Age', 'required|isInt|gt[12]');

    if ($this->vform->run()) {
        // Formulaire correcte
    }
    else {
        // Formulaire incorrecte
    }
```

2. Les règles :

La librairie « vForm » inclus des règles prédéfinies pour la vérification des formulaires. Lorsque vous entrez les règles dans la fonction « addRules », vous devez les séparer d'un « | ». Il est possible que les règles demandes un ou plusieurs paramètres, pour cela il faut mettre les paramètres entre crochet « [param] », si un règle demande plusieurs paramètres alors il faut séparer les paramètres par une virgule « [param1,param2] ».

Liste des règles disponibles :

Nom	Description
required	Vérifie que le champ est présent dans le formulaire.
notEmpty	Vérifie que le champ n'est pas vide.
matches[string]	Vérifie que les données du champ correspondes aux données d'un autre champ passé en paramètre.
minLength[int]	Vérifie que les données du champ font au moins x caractères.
maxLength[int]	Vérifie que les données du champ font moins de x caractères.
btwLength[int, int]	Vérifie que les données du champ sont comprises entres x et x caractères.
exactLength[int]	Vérifie que les données du champ font x caractère.
isNumeric	Vérifie que les données du champ sont numériques.
isInt	Vérifie que les données du champ sont numériques et qu'il sagit d'une valeur entière.
gt[int]	Vérifie que les données du champ sont supérieurs à x.
lt[int]	Vérifie que les données du champ sont inférieurs à x.
alpha	Vérifie que les données du champ ne contiennent que des lettres.
alphaNum	Vérifie que les données du champ ne contiennent que des lettres et des chiffres.
alphaExt	Comme alphaNum mais avec les caractères « - », « » et « _ ».
email	Vérifie que le champ contient une adresse email.
check[string]	Vérifie que les données du champ correspondes à un regexp passé en paramètre.

3. L'affichage des erreurs :

Pouvoir vérifier les formulaires est une chose mais il faut aussi pouvoir informer l'utilisateur des éventuelles erreurs qu'il aurai commis dans le formulaire.

Pour cela il existe deux fonctions. La première est « hasError » de la librairie, cette fonction retourne « TRUE » s'il existe des erreurs et « FALSE » s'il n'y a pas d'erreur.

Vform::hasError():bool

La deuxième est la fonction « getError » de la librairie, si le champ est précisé, elle retourne l'erreur sur le champ précisé sinon elle retourne toute les erreurs dans le formulaire. Les erreurs sont

retourné sous forme de chaîne de caractère précédé d'un préfixe et suffixe (modifiable à l'aide des fonctions « setErrorPrefix » et « setErrorSuffix »). S'il n'y a pas d'erreur, alors la fonction retourne une chaîne de caractère vide.

*Vform::getError([**$field**:string]):string*

Exemple :

```php
public function index() {
    $this->load->library('vform');

    $this->vform->addRules('email', 'E-mail', 'required|maxLength[124]|email');
    $this->vform->addRules('age', 'Age', 'required|isInt|gt[12]');

    if ($this->vform->run()) {
        echo 'Aucune erreur !';
    }
    else {
        if ($this->vform->hasError())
            echo $this->vform->getError();
    }
```

Pour afficher les erreurs dans la vue on peut passer le résultat des fonctions « hasError » et « getError » à la vue (déconseillé) où alors simplement utiliser l'helper « vform » qui est chargé automatiquement lorsque l'on charge la librarie « vform ». Dans le deuxième cas, il faudra utiliser la fonction « hasErrorForm » (alias de hasError) et la fonction « getErrorForm » (alias de getError).

Exemple :

```php
<!DOCTYPE HTML>
<html>
<head>
    <title>Ajouter un groupe</title>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Ajout d'un groupe</h1>
    <?php if (hasErrorForm()) echo getErrorForm(); ?>
```

Dans l'helper « vform » il existe une autre fonction utile : « getPostData », cette fonction retourne la valeur du champ en paramètre (si le champ existe et qu'il n'y a pas d'erreur dessus). Cette fonction est pratique pour ré-remplir les champs d'un formulaire après une erreur.

*getPostData(**$field**:string [, **$escaped**:bool]):string*