Nano framework

Nano MVC (Français)

- Configuration
- Erreur et logs
- Routage, rewrite, controlleur
- Modèle de données
- Vues, multilingue
- Cache, twig, Less
- Mail

Nano sur http://nano.local

```
<?php

    /**
    * Nano Mvc
    *
    * Necessite PHP 5.3 à 7.2 et le
mod_rewrite actif.
    *
    * "app/cache", "app/log" et "public/css"
    * doivent disposer des droits d'ecriture.
    */
    run();

?>
```

Configuration

La configuration est uniquement déclarée sous forme de fichier. Vous la trouverez dans votre dossier de configuration de l'application "app/config"

La configuration minimale requiert les fichiers *"global.php"* et *"route.php"*.

Chaque fichier de configuration peut etre surchargé avec la configuration d'un domaine. Les fichiers ne sont pas remplacés, mais fusionné en faveur de la configuration de domaine.

Tout autre fichier supplementaire sera chargé automatiquement, à la demande.

Pour récuperer manuellement un parametre de configuration depuis un fichier, vous pouvez utiliser la commande suivante:

```
\Nano\Config::get('global.error.show');
```

La nom du parametre de configuration s'articule de manière uniforme, le nom du fichier de configuration, suivi des nom de parametres de chaque niveau de tableau défini, le tout séparé par des points. Si aucune valeur ne correspond au paramètre demandé, la commande retournera "null".

Erreurs et log

Le gestionnaire d'erreur est de log son par defaut géré de manière automatique, configurable via le fichier de configurarion "app/config/global.php"

```
return [
    'error' => [
        'show'  => true, // Afficher les erreurs
        'log'   => true  // Faire un trace log automatique lors d'une erreur
    ],
    [...]
];
```

Il est egalement possible de demander une trace en log manuellement à partir de la commande suivante :

```
\Nano\Log::trace('mon message');
```

Les logs sont ecrits automatiquements sur le chemin "app/log/[année]/[mois]/[jour].php"

Routage, rewriting, controlleur

Pour des question pratiques et de simplicité, les mecaniques de routages ont été simplfiés. Les routes sont definies dans le fichier de configuration *"app/config/route.php"*

```
return [
    '__404__'                                   => 'e404/index',
    '/'                                         => 'home/index',
    '/rewrite-([0-9a-z]+)'                      => 'home/index',
    '/rewrite-([0-9a-z]+)-([0-9a-z]+)'          => 'home/index',
    '/rewrite-([0-9a-z]+)-([0-9a-z]+)-([0-9a-z]+)' => 'home/index'
];
```

La routage se limite à une liste de clés/valeurs. La route à pour clé une expression regulière correspondante aux regles de rewrite. La valeur de la route correspond au chemin d'accès au controller, terminé par le nom de la methode désirée.

Les routes encadrés par des "__", tel que "__404__" correspond à des routes spéciales qui ne seront pas interprétées comme des expression regulières.

Pour toute les autres routes, les parametres de rewrites sont automatiquement injecté dans leur ordre d'apparition vers la methode de destination du routage comme on peut le voir dans le controlleur utilisé pour la page en cours.

```
class Home{
    public function index($param1=null, $param2=null, $param3=null){
        return new \Nano\View\Twig('index.twig', ['p1'=>$param1,
'p2'=>$param2, 'p3'=>$param3], 200, []);
    }
}
```

! Attention: Afin que le routage puisse fonction en presence ou absence des parametres de routages, les parametres definis dans la methode du controlleur devront tous disposer d'une valeur par defaut (ici "null")

```php
class Home{


    public function get_index(){
        return
new\Nano\View\Twig('index.twig',
[], 200, []);
    }
    public function post_index()
{
        return new
\Nano\View\Twig('index.twig',
[], 200, []);
    }
    public function index(){
        return new
\Nano\View\Twig('index.twig',
[], 200, []);
    }


}
```

Il est egalement possible implicitement de definir le routage à partir de la methode http utilisée, via le controller, comme sur l'exemple suivant :

Dans le cas ci-contre, le routeur cherchera en priorité une methode [MethodeHttp]_[MethodeController], et sinon [MethodeController].

Dans le cas présent ou la page est accédée via une methode HTTP "GET", la methode "get_index" sera executée, ou en son absence, la methode "index".

Modèle de données

La configuration des connexion de base de données sont dans le fichier *"app/config/database.php"*

```php
return [
    'connexion1' => [
        'dsn'      => 'mysql:host=localhost;dbname=Nano',
        'user'     => 'root',
        'password' => '',
        'timeout'  => 3
    ]
];
```

Il est possible de definir autant de connexion que l'on souhaite, elle ne seront exploité qu'a la demande.

Une fois les connexion definies, il est possible de créer des abstractions de tables dans le dossier "app/model", afin de regrouper toute les manipulations de données dans un même fichier, sous la forme suivante :

```php
class MyTable extends \Nano\Model{

    // Nom de la connexion utilisé (defini en configuration)
    protected static $connexion = 'connexion1';

    public static function getAll(){
        return static::query('SELECT * FROM mytable');
    }
}
```

Enfin, cette abstraction est exploitable depuis n'importe quel controlleur :

```php
namespace App\Controller;

    use \App\Model\MyTable as MyTableModel;

    class Home{

        public function index(){
            $data = MyTableModel::getAll();
            return new \Nano\View\Twig('index.twig', ['data' => $data], 200,
[]);
        }

    }

}
```

Vues, multilangue

Nano propose 3 type de rendu de vues, mise à disposition de manière natives :

- Rendu de flux de données : \Nano\View
- Rendu de JSON : \Nano\View\Json
- Rendu de Vues Twig : \Nano\View\Twig

Il est fortement conseillé d'exploité les vues twig pour les sorties html afin de profiter de la prise en charge du cache natif, ainsi que pour les facilités de mise en formes, et des fonctionnalités supplementaires proposées, comme les fonctions suivantes :

- {{ basepath() }} : Retourne le domaine courant (utilisé pour former les url absolues des assets
- {{ translate('clé.de.traduction') }} : Traduit le mot clé fourni selon la langue et les fichier de traduction fournis.
- {{ config('clé.de.configuration') }} : Retourne la configuration associée.

Les traductions sont stockées sous forme de fichier dans le dossier "app/lang/", chaque fichier porte les 2 premières lettres de la locale de la langue qu'ils representent. L'appel aux traductions suivent le formalisme des fichiers de configuration :

```php
return [
    'lang'  => 'Français',
    'ui'    => [
        'title' => 'Mon titre'
    ]
];
```

Les traduction et les configurations sont appelable alors depuis un controlleur ou une vue

```php
// Dans un controlleur
\Nano\Lang::translate('ui.title');
\Nano\Config::get('global.lang.default'
```

```
<!-- Dans une vue !-->
{{ translate('ui.title'); }}
{{
config('global.lang.default');
}}
```

La langue est detectée par defaut via le parametrage du navigateur, sinon reviens à la langue par defaut. En cas de changement de langue, elle peut être forcé via le controlleur :

```
// Dans un controlleur
\Nano\Lang::setLang('fr');
```

Cache, Twig, Less, compilation

Nano supporte l'utilisation de la librairie **Less** et **Twig** pour faciliter l'exploitation et la maintenance de feuille de style et des templates.

Les fichiers **Twig** (.twig) sont directement appelable en temps que vue depuis le controlleur. La gestion du cache se fait depuis le fichier de configuration de l'application "app/config/global.php"

Les fichiers **Less** (.less) sont regroupés dans les assets "public/asset/less/"
Afin de spécifier au système les fichiers less à compiler, il faut les renseigner dans le fichier de configuration de l'application "app/config/global.php"

La gestion de tous les **caches** sont géléralisé dans une seule variable en fichier de configuration de l'application "app/config/global.php"

```
'view' => [
    'cache' => false, // Activer le cache twig et less
    'less' => [
        // Liste les fichiers less à compiler
        // (gauche) Le chemin du fichier less en absolu à partir du dossier
public/asset/less
        // (droite) Le chemin du fichier css en absolu à partir du dossier
public/asset/css
        'home/home.less' => 'home.css',
    ]
],
```

N.B. Less ayant encore un bug sur les "@import" sur un dossier parent, les chemins d'accès associés à la commande "@import" sont attendu en absolue à partir du dossier racine "public/asset/less/"