dFramework

Table des matières

Bienvenue chez nous

Bienvenue sur dFramework

dFramerork est un framework de développement d'applications - une boîte à outils - destiné aux personnes qui construisent des applications et sites Web à l'aide de PHP. Son objectif est de vous permettre de développer des projets beaucoup plus rapidement que si vous écriviez du code à partir de rien, en fournissant un ensemble riche de bibliothèques pour les tâches courantes, ainsi qu'une interface simple et une structure logique pour accéder à ces bibliothèques. dFramework vous permet de vous concentrer de manière créative sur votre projet en minimisant la quantité de code nécessaire pour une tâche donnée.

A qui s'adresse dFramework

dFramework est la boîte à outils qu'il vous faut si:

- Vous voulez un cadre avec une petite empreinte.
- Vous avez besoin d'une performance exceptionnelle.
- Vous voulez un framework qui nécessite une configuration presque nulle.
- Vous voulez un framework qui ne vous oblige pas à utiliser la ligne de commande.
- Vous voulez un cadre qui ne vous oblige pas à respecter des règles de codage restrictives.
- Vous n'êtes pas intéressé par les bibliothèques monolithiques à grande échelle comme PEAR.
- Vous ne voulez pas être obligé d'apprendre un langage de templates.
- Vous évitez la complexité, privilégiez les solutions simples.
- Vous avez besoin d'une documentation claire et complète.

Préface

Mot de l'auteur

dFramework est le diminutif de Dimtrov Framework. Il a été initialement conçu pour aider à la réalisation de mon projet de deuxième année du cycle d'ingénieur à l'Institut Africain d'Informatique représentation du Cameroun.

Une fois le projet réalisé, j'ai voulu l'utiliser dans mes différents projets futur afin de ne plus avoir à refaire toutes les opérations de routines qu'on rencontre dans la création des sites et applications web (routage, traitement des formulaires, etc) tout en gardant une maîtrise totale de la structure et du fonctionnement du système.

Le framework fonctionnait mais avait de gros manquements et était trop basique. J'ai donc décidé de l'améliorer considérablement pour l'utiliser dans mes projets personnel et faire en profiter mes amis de classe (qui n'étaient pas très alaise en développement web).

Par ailleurs, je suis un partisan de l'initiative GNU et l'open source en général, c'est pourquoi, je m'applique dans la réalisation de cette œuvre pour apporter ma modeste contribution à l'essor de l'informatique en espérant vraiment que ceci pourra être utile à plus d'une personne.

Crédits

dFramework a été développé par Dimitric Sitchet Tomkeu (Étudiant en Génie Logiciel a l'IAI-Cameroun et CEO de Dimtrov Sarl). Dans la réalisation de ce travail, il a utilisé plusieurs ressources extérieurs:

- Au niveau des cours et tutoriels
    - La POO en PHP (www.grafikart.fr)
    - Mis en pratique de la POO en PHP (www.grafikart.fr)

- Au niveau des codes sources
    - CodeIgniter (https://github.com/bcit-ci/CodeIgniter)

    - CakePHP (http://cakephp.org)

- Au niveau de l'élaboration du manuel d'utilisation
    - CodeIgniter (https://github.com/bcit-ci/CodeIgniter)

Remerciements

Licence

Licence d'utilisation

Le guide d'utilisation

Ce document, considéré comme le guide d'utilisation du dFramework a été écrit par Dimitric Sitchet Tomkeu. Il est mis librement et gratuitement à la disposition de tout un chacun suivant les termes d'utilisation de la licence **Creative Commons BY-NC-SA**. En utilisant ce document vous devez garder à l'esprit que dFramework est un projet opensource; de ce fait, vous êtes libre d'utiliser ce guide, de le modifier et de le partager à condition expresse de respecter les termes du contrat ci-après:

- **Attribution**: En utilisant, en modifiant ou en partageant ce document, vous reconnaissez qu'il a été initialement écrit par Dimitric Sitchet Tomkeu (https://www.facebook.com/dimtrovich). En cas de redistribution, le nom de ce dernier devra systématiquement apparaître dans toutes les copies.
- **Non commercial**: Interdiction formelle de tirer un profit commercial (gain direct ou plus-value commerciale) en utilisant cette œuvre ou son adaptation.
- **Partage dans les mêmes conditions**: Si vous décidez de partager cette

œuvre (adaptation ou original), vous le rediffuserai obligatoirement selon la même licence ou avec une licence similaire (version ultérieure par exemple).

Le code source

dFramework est un ensemble de classes et de fonctions créées pour faciliter le développement des sites web et applications écrits en PHP. C'est un projet opensource distribué sous la licence MPL-2. Son code source, hébergé sur GitLab (https://gitlab.com/dimtrov/dframework) et GitHub, est actuellement maintenu par Dimitric Sitchet Tomkeu (Mai 2020).

Avant d'utiliser dFramework pour réaliser votre projet, pensez à bien prendre en compte les termes d'utilisations spécifiés ci-dessous:

Copyright (c) 2019-2020 Dimitric Sitchet Tomkeu

Une autorisation est accordée gratuitement à toute personne qui obtient une copie
de ce logiciel et les fichiers de documentation associés (le "Logiciel"), pour traiter
dans le Logiciel sans restriction, y compris sans limitation les droits
pour utiliser, copier, modifier, fusionner, publier, distribuer, sous-licence et/ou vendre
des copies du Logiciel, et de permettre aux personnes à qui le Logiciel est
pour le faire, sous réserve des conditions suivantes :

Les avis de droit d'auteur ci-dessus et ce préavis d'autorisation sont inclus dans tous les
copies ou des portions substantielles du Logiciel.

LE LOGICIEL COUVERT EST FOURNI SOUS CETTE LICENCE « TEL QUEL », SANS GARANTIE D'AUCUNE SORTE, EXPRESSE, IMPLICITE OU

STATUTAIRE, Y COMPRIS SANS LIMITATIONS, LES GARANTIES QUE LE LOGICIEL COUVERT EST EXEMPT DE DEFAUTS, COMMERCIALISABLE, ADAPTÉ A UN PARTICULIER BUT OU NON-CONTREFACON. VOUS ÊTES ENTIEREMENT RESPONSABLE DE LA QUALITÉ ET DES PERFORMANCES DU LOGICIEL COUVERT. SI UN LOGICIEL COUVERT S'AVÉRAIT DÉFECTUEUX A QUELQUE ÉGARD QUE CE SOIT, VOUS (PAS TOUT CONTRIBUTEUR) ASSUMEZ LE COÛT DE TOUT SERVICE, RÉPARATION OU CORRECTION NÉCESSAIRE. CETTE EXCLUSION DE GARANTIE CONSTITUE UNE PARTIE ESSENTIELLE DE CETTE LICENCE, AUCUNE UTILISATION D'UN LOGICIEL COUVERT N'EST AUTORISÉE EN VERTU DE CETTE LICENCE, SAUF EN VERTUE DE CETTE CLAUSE DE NON-RESPONSABILITÉ.

Pour plus d'informations sur la licence de dFramework, veuillez vous rendre à l'adresse https://dimtrov.hebfree.org/works/licenses/dframework

Change Log

Changements et améliorations apportés

Version 2.0

Date de début de mis à jour: 15/10/2019

- Ajout du routeur d'URL en complément au dispatcheur
- Création du système de capture de bloc d'une vue pour le layout
- Ajout des librairies Crypto, Paginator, Mail, Upload, Zipper, Validator, File
- Intégration des objets *request* et *response*

Version 2.0.1

Date de début de mis à jour: 12/12/2019

- Tous les contrôleurs doivent désormais être suffixé de *Controller*
- Ajout du fichier de configuration *layout.php* pour définir les fichiers css et js à

charger automatiquement dans le Layout
- Possibilité de charger les fichiers css et/ou js directement à partir du fichier de vue qui les utilisera
- Ajout des librairies Api, Debug
- Amélioration de la librairie Browser qui devient par la même occasion Ua

Version 2.1

Date de début de mis à jour: 28/12/2019

- Mise en place du système de cache
- Intégration de l'utilitaire de migration des base de données
- Ajout des librairies Populate, Parser

Version 2.2

Date de début de mis à jour: 25/01/2020

- Chargement automatique des objets *Request* et *Response*. Plus besoin de les importer au préalable
- Intégration des gestionnaires de session et de failles CSRF
- Pris en compte des suffixes d'URL
- Ajout des librairies Form

Version 3.0

Date de début de mis à jour: 15/02/2020

- Intégration du moteur de template Smarty
- Restructuration des dossiers du framework
- Organisation des dossiers de dépendances selon leurs structures initiales (retrait de l'espace de nom *dframework\dependencies* et conservation de l'espace de nom propre à la dépendance)
- Ajout d'une "class-mapper" pour auto-charger les classes de dépendances spécifiquement à leurs espaces de nom
- Ajout de l'utilitaire interne de gestion de mots de passe (génération, hachage, vérification d'égalité)
- Ajout de l'utilitaire d'authentification (composant Login, composant Register)

- Les configurations ne sont plus tous chargées au démarrage de l'application. Chaque configuration est désormais chargée lorsque le besoin de l'utiliser se fait ressentir
- Intégration de la dépendance flip\whoops pour mieux afficher les erreurs de code
- Ajout d'un utilitaire natif pour capturer et sauvegarder les erreurs et exceptions dans des fichiers journaux
- Ajout d'un système de remappage interne des routes (équivalent de la méthode _remap() de CodeIgniter)
- Remplacement de la librairie Validator en Checker. Ajout d'une méthode de vérification de numero de téléphone (Cameroun et France uniquement)
- La librairie Validator est désormais utilisée a d'autres fin et s'appuie sur la dépendance vlucas\valitron
- Ajout des librairie Image, Dom et Pdf

Version 3.1

Date de début de mis à jour: 29/04/2020

- Possibilité de charger un contrôleur dans un autre
- Ajout de l'utilitaire de création des web services (RestController)
- Ajout d'un mini ORM natif pour la gestion des entités
- Possibilité de mapper des classes spécifique de l'application : indépendamment des dépendances systèmes
- Intégration de l'outils de profilling "Tracy"
- Ajout des utilitaires de génération automatique des entités, des modèles et des contrôleurs
- Ajout de l'interface de migration de base de données via l'invite de commande

Exigences du système

Que faut-il pour utiliser dFramewrok ?

Pour fonctionner un projet utilisant dFramework doit être lancé sur un serveur web ayant les propriétés suivantes

- Une version de PHP >= 7.1
- L'extension PDO doit être activée
- L'extension OpenSSL doit être activée
- L'extension Reflection doit être activée
- L'extension DOM doit être activée
- L'extension XML doit être activée
- L'extension FileInfo doit être activée

Installation

Guide d'installation

Télécharger les sources

dFramework est disponible sous forme d'archive zip. Vous pouvez le télécharger à tout moment et gratuitement sur son site officiel, (https://dimtrov.hebfree.org/works/dframework) ou sur le dépôt GitHub du groupe Dimtrov (https://github.com/Dimtrov).

Liens de téléchargements directs

- https://github.com/Dimtrov/dframework/master.zip

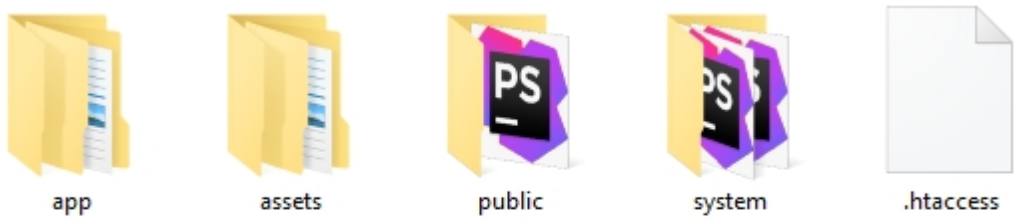
Installation

dFramework est très simple à installer. Le processus d'installation du framework se fait en quatre étapes.

1. Décompressez l'archive téléchargé.
2. Déposez les dossier et fichiers issus de l'archive dans la racine de votre projet (*ex: C:\xampp\htdocs\myproject*)
3. Ouvrez le fichiers *app/config/general.php* et entrez votre URL de base ($general['base_url']) (*ex: http://localhost/myproject*). Si vous ne renseignez pas cette variable, le framework essayera de détecter automatiquement l'URLde base.
4. Ouvrez enfin le fichier *app/config/data.php* et définissez votre clé de chiffrement ($data['encryption']['key']). Cette clé doit être bien choisie et rester secrète car elle sera utiliser pour chiffrer vos données de sessions et cookies et est utiliser par la bibliothèque Crypto

Ça y est... Votre projet est prêt. Vous pouvez désormais utiliser toutes les fonctionnalités offertes par dFramework pour développer votre application.

Architecture physique d'un projet dFramework



Un projet dFramework a à sa racine **au moins quatre dossiers et un fichier .htaccess.** Ceux-ci étant indispensable pour le fonctionnement de votre application.

1. **Le fichier ".htaccess"**

Pour ceux qui ne le connaissent pas, c'est un ficher de configuration du serveur Apache. Dans le processus de dFramework, il définit le fichier servant de point d'entré de l'application et retransmet toutes les URLs de l'utilisateur vers ce fichier.

2. **Le dossier "public"**

Généralement, le point d'entré des sites web c'est le fichier index.php qui se

trouve à la racine de votre projet. Cette pratique n'est pas bonne et on recommande de déplacer ce fichier vers un sous dossier. Le dossier public est donc un simple dossier contenant uniquement le point d'entré de votre application et les fichiers statiques de votre applications repartis dans quatre principaux dossiers à savoir

- *css* pour stocker vos fichiers css
- *img* pour vos images
- *js* pour stocker vos fichiers javascript
- *lib* pour accueillir les fichiers des plugins que vous utiliser pour votre front-end (bootstrap, jquery, dataTable, etc.)

Vous pouvez ajouter vos propres dossiers si vous le souhaitez mais, ces quatre dossiers doivent exister sinon vous ne pourriez pas utiliser les fonctions du helper assets.

3. **Le dossier "system"**

Comme son nom l'indique, c'est le dossier du système. Il contient toute les sources propres au framework. Vous ne devez donc le toucher sous aucun prétexte (sauf ci vous savez vraiment ce que vous faites) au risque d'endommager le framework et provoquer le disfonctionnement de votre application.

Le dossier *system* contient à sa racine entre autre les fichiers et les dossiers suivant

- *Autoloader.php:* C'est l'auto-loader des classes dFramework uniquement. Vous pouvez créer l'auto-loader de vos propres classe manuellement ou en utilisant *composer.*
  - *Par défaut (si vous ne supprimez pas le fichier composer.json et le dossier vendors contenu dans l'archive lors du téléchargement), vous pouvez placer vos classes applicatives dans le dossier app/class et les donner le namespace* ***App***
- *components:* Ce dossier regorge les composants du framework
- *constants:* Ce dossier contient les fichiers statiques utilisés par les composants propres du framework
- *core:* Ce dossier est le cœur du framework. Tous les fichiers principaux du framework s'y trouvent
- *dependencies:* Comme sont nom l'indique, ce dossier stocke toutes les dépendances internes du framework

- *helpers:* On y retrouve les différents helpers du framework
- *libraries:* Ce dossier contient toutes les librairies (bibliothèques) natives du framework

4. **Le dossier app**

C'est votre dossier. Tous les fichiers propres à votre applications (configurations, contrôleurs, vues, modèles, etc.) se trouvent dans ce dossier. C'est ici que vous passerez la majeur partie de votre temps c'est pourquoi, des explications détaillés sur le fonctionnement de ce dossier seront données dans des sections appropriées.

Fonctionnalités

Architectures dFramework

Organisation des répertoires

Modèle MVC

Bootstraping de l'application

Les URL dFramerwork

Le système d'URL dFramerwork

Introduction

Par défaut, les URL dans dFramework sont conçues pour être conviviales pour les moteurs de recherche. Plutôt que d'utiliser l'approche standard de «chaîne de requête» pour les URL qui est synonyme de systèmes dynamiques, nous utilisons une approche basée sur les segments : *exemple . com /actualités/article/my_article*

Segments d'URI

En suivant l'approche MVC, les segments dans l'URL, représentent généralement: *site.com/classe/fonction/parametres*

1. Le premier segment représente la **classe de contrôleur** qui doit être invoquée.
2. Le deuxième segment représente la **fonction de classe** , ou méthode, qui doit être appelée.
3. Le troisième, et tous les segments supplémentaires, représentent toutes les variables qui seront transmises au contrôleur.

Le Helper URL contient des fonctions qui facilitent l'utilisation de vos données URI. De plus, vos URL peuvent être remappées à l'aide de la fonction de routage URI pour plus de flexibilité.

Ajout d'un suffixe d'URL

Vous pouvez spécifier un suffixe qui sera ajouté à toutes les URL générées par dFramework. Par exemple, si une URL est la suivante: *exemple.com/produits/voir/chaussures*, vous pouvez éventuellement ajouter un suffixe, comme **.html**, faisant apparaître la page d'un certain type: *exemple.com/produits/voir/chaussures.html*

Pour cela, vous devez modifier la valeur du paramètre *$general['url_suffix']* dans votre fichier */app/config/general.php*

Les contrôleurs

Introduction

Les contrôleurs sont au cœur de votre application, car ils déterminent comment les requêtes HTTP doivent être traitées.

Un contrôleur est simplement un fichier de classe qui est nommé d'une manière et qui peut être associée à un URI.

Considérons l'URI *exemple.com/blog*. dFramework tenterait de trouver un fichier nommé *BlogController.php* dans le dossier */app/controllers/* et le chargerait. Lorsque le nom d'un contrôleur correspond au premier segment d'un URI, il est automatiquement chargé.

Hello World !

Créons un simple contrôleur pour que vous puissiez le voir en action. À l'aide de votre éditeur de texte, créez un fichier appelé *BlogController.php* dans votre dossier */app/controllers/* et insérez-y le code suivant:

```
<?php
        class BlogController extends
dFramework\core\Controller {
                public function index()
                {
                   echo 'Hello World!';
                }
        }
```

Le nom du fichier doit commencer par une LETTRE MAJUSCULE (B) et doit être suffixé par Controller ('C' majuscule). Le fichier doit contenir une

classe ayant le même nom que le fichier (BlogController) qui hérite de la classe dFramework\core\Controller.

Si vous vous rendez à l'adresse *exemple.com/blog*, vous verrez votre message s'afficher sur votre écran.

Les actions ou méthodes

Dans l'exemple ci-dessus, le nom de l'action est index(). L'action «index» est toujours chargée par défaut si le **deuxième segment** de l'URI est vide. Une autre façon d'afficher votre message «Hello World» serait donc : *exemple.com/blog/index*

Le deuxième segment de l'URI détermine quelle action du contrôleur est exécutée.

Essayons. Ajoutez une nouvelle méthode à votre contrôleur:

```
<?php
        class            BlogController            extends
dFramework\core\Controller {
                public function index()
                {
                  echo 'Hello World!';
                }
                public function comments()
                {
                  echo 'Un simple commentaire';
                }
        }
```

Chargez maintenant l'URL *exemple.com/blog/comments* pour exécuter l'action comments(). Vous verrez alors votre nouveau message s'afficher.

Passer des argument à votre action

Si votre URI contient plus de deux segments, ils seront transmis à votre action comme en tant que paramètres.

Par exemple, supposons que vous ayez l'URI *exemple.com/membres/profils/14/dimitri*. Votre action passera en paramètres les segments 3 et 4 ("14" et "dimitri").

```
    <?php
         class MembresController extends
dFramework\core\Controller {
              public function profils($id, $name)
              {
                 echo 'Salut ' . $name . ' ! votre
identifiant est : ' . $id;
              }
         }
```

Accédez maintenant à l'adresse *exemple.com/membres/profils/15/john* pour vérifier le résultat.

Définition d'un contrôleur par défaut

dFramework peut charger un contrôleur par défaut en l'absence d'URI, comme ce sera le cas lorsque seule l'URL racine de votre site est demandée. Pour spécifier un contrôleur par défaut, vous devez modifier la valeur du paramètre *$route['default_controller']* de votre fichier */app/config/route.php*.

Par exemple, en mettant `$route['default_controller'] = 'Blog'`, le contrôleur "BlogController" sera automatiquement chargé (et l'action "index" sera exécutée) si on accède a la racine de votre site : *exemple.com*

Méthodes privées

Dans certains cas, vous souhaiterez peut-être que certaines actions ne soient pas accessibles au public. Pour ce faire, il vous suffit de déclarer la méthode comme étant **privée** ou **protégée** et elle ne sera pas diffusée via une requête URL. Par exemple, si vous deviez avoir une méthode comme celle ci-dessous, elle ne pourra pas être appelée via une URL:

```
private function my_protected_action()
{
    // some code
}
```

Organisez vos contrôleurs en sous répertoires

Si vous créez une grande application, vous souhaiterez peut-être organiser ou structurer hiérarchiquement vos contrôleurs en sous-répertoires. dFramework vous autorise à le faire.

Créez simplement des sous-dossiers dans le répertoire */app/controllers/* et placez-y vos classes de contrôleur.

Lorsque vous organisez vos classes de cette maniere, le premier segment de votre URI doit spécifier le dossier. Par exemple, supposons que vous ayez un contrôleur situé dans "*/app/controllers/produits/Chaussures.php*"

Pour appeler le contrôleur ci-dessus, votre URI ressemblera à "exemple.com/produits/chaussures/voir/123

Chacun de vos sous-répertoires peut contenir un contrôleur par défaut qui sera appelé si l'URL ne contient que le sous-répertoire. Placez simplement un contrôleur qui correspond au nom de votre 'default_controller' comme spécifié dans

votre fichier */app/config/route.php*

dFramework vous permet également de remapper vos URI à l'aide de son gestionnaire de route fonction de routage d'URI . vous créez une grande application, vous souhaiterez peut-être organiser ou structurer hiérarchiquement vos contrôleurs en

Constructeur de classe

Si vous avez l'intention d'utiliser un constructeur dans l'un de vos contrôleurs, vous DEVEZ y placer la ligne de code suivante: `parent::__construct();`

La raison pour laquelle cette ligne est nécessaire est que votre constructeur local remplacera celui de la classe de contrôleur parent, nous devons donc l'appeler manuellement.

```
<?php
        class              BlogController              extends
dFramework\core\Controller {
                public function __construct()
                {
                   parent::__construct();
                         // Ecrivez les instructions de votre
constructeur ici
                }
        }
```

Les constructeurs sont utiles si vous devez définir des valeurs par défaut ou exécuter un processus par défaut lorsque votre classe est instanciée. Les constructeurs ne peuvent pas retourner de valeur, mais ils peuvent effectuer un travail par défaut.

Noms de méthodes réservés

Étant donné que vos classes de contrôleur étendront le contrôleur d'application principal, vous devez faire attention de ne pas nommer vos méthodes de manière identique à celles utilisées par cette classe, sinon vos fonctions locales les remplaceront. Voir Noms réservés pour une liste complète

Vous ne devez également jamais avoir une méthode nommée de manière identique à son nom de classe. Si vous le faites, et qu'il n'y a pas de méthode *__construct()* dans la même classe, alors votre méthode *Index::index()* sera exécutée en tant que constructeur de classe!

La vue

Les vues simples

Introduction

Une vue est une simple page web ou un fragment de page comme par exemple *le header, le footer ou la sidebar*. Elles peuvent facilement être imbriquées dans d'autres vues pour créer une hierarchie.

Les vues ne sont jamais appelées directement, elles doivent être charger par un contrôleur. Souvenez-vous que dans les framework MVC, le contrôleur est le chef d'orchestre du processus, donc c'est lui qui est responsable de la récupération d'une vue particulière. Si vous n'avez pas lu le chapitre sur le contrôleur, nous vous recommandons d'y jeter un coup d'oeuil avant de continuer.

Nous allons utilisé l'exemple de contrôleur crée dans le chapitre relatif au contrôleur, pour y ajouter une vue.

Création d'une vue

Utilisez votre éditeur de texte et créer un fichier nommé *blogview.php*, enregistrer le dans le dossier */app/views/blog* et insérer y le code HTML suivant. (*vous remarquez que le dossier "blog" est à créer dans le répertoire /app/views*)

```
<html>
<head>
     <title>Mon Blog</title>
</head>
<body>
     <h1>Bienvenu sur mon Blog!</h1>
     <p>Vous trouverez ici toutes mes informations et
activités</p>
</body>
</html>
```

Chargement d'une vue

Pour charger une vue particulière, vous devez utiliser la méthode suivante.

```
$this->view('nom_vue')->render();
```

Ici, *nom_vue* represente le nom de votre fichier de vue.

L'extension du fichier ne doit pas être spécifiée. Le framework se chargera dé récupérer le bon fichier et l'afficher

Maintenant, ouvrez votre fichier de contrôleur (BlogController.php) et remplacez l'instruction `echo` de l'action index() par la méthode de chargement d'une vue comme le montre le code suivant :

```
<?php
     class BlogController extends
dFramework\core\Controller {
          public function index() {
             $this->view('blogview')->render();
          }
     }
```

Si vous ouvrez votre navigateur avec l'URL appropriée, vous verez une page correspondant au code entré dans votre fichier de vue. L'URL peut être par exemple *http://localhost/my-df-app/blog*

Chargement de plusieurs vues

dFramework gérera intelligemment plusieurs appels à view() à partir d'un contrôleur. Si plusieurs appels se produisent, ils seront ajoutés ensemble. Par exemple, vous pouvez souhaiter avoir une vue d'en-tête, une vue de menu, une vue de contenu et une vue de pied de page. Cela pourrait ressembler à ceci:

```
<?php
        class PageController extends
dFramework\core\Controller {
            public function index()
            {
               $this->view('header')->render();
               $this->view('menu')->render();
               $this->view('content')->render();
               $this->view('footer')->render();
            }
        }
```

Structurez vos dossiers de vues

Dossiers relatifs et dossiers absolus

dFramework gérera intelligemment plusieurs appels à view() à partir d'un contrôleur. Si plusieurs appels se produisent, ils seront ajoutés ensemble. Par exemple, vous pouvez souhaiter avoir une vue d'en-tête, une vue de menu, une vue de contenu et une vue de pied de page. Cela pourrait ressembler à ceci:

[code]

Placez vos vues dans des sous répertoires

Vos fichiers de vue peuvent également être stockés dans des sous-

répertoires si vous préférez ce type d'organisation. Pour ce faire, vous devrez inclure le nom du répertoire chargeant la vue. Exemple:

```
$this->view('sous_dossier/fichier')->render();
```

Ajout des données dynamiques à la vue

Les données sont transmises du contrôleur à la vue au moyen d'un tableau dans le deuxième paramètre de la méthode de chargement de la vue. Voici un exemple simple:

```
$data = array(
    'title' => 'Mon titre',
    'heading' => 'Mon entete',
    'message' => 'Mon message'
);
$this->view('blogview', $data)->render();
```

Allez dans votre contrôleur d'exemple et modifiez le comme ceci

```
<?php
     class BlogController extends
dFramework\core\Controller {
          public function index()
          {
             $data['title'] = "Mon vrai titre";
             $data['heading'] = "L'entete plus realiste";
             $this->view('blogview', $data)->render();
          }
     }
```

Puis dans votre fichier de vue, essayez le code suivant

```
<html>
<head>
     <title><?php echo $title; ?></title>
</head>
<body>
     <h1><?php echo $heading; ?></h1>
```

```
</body>
</html>
```

Création de boucles

Le tableau de données que vous transmettez à vos fichiers d'affichage n'est pas limité à de simples variables. Vous pouvez passer des tableaux multidimensionnels, qui peuvent être bouclés pour générer plusieurs lignes. Par exemple, si vous extrayez des données de votre base de données, elles se présentent généralement sous la forme d'un tableau multidimensionnel.

Voici un exemple simple. Ajoutez ceci à votre contrôleur:

```
<?php
     class BlogController extends
dFramework\core\Controller {
          public function index()
          {
             $data['todo_list'] = array('Netoyer la
maison', 'Appeler maman', 'Visionner');
             $data['title'] = "Mon vrai titre";
             $data['heading'] = "L'entete plus realiste";
             $this->view('blogview', $data)->render();
          }
     }
```

Dans votre fichier de vue, il ne vous restera plus qu'a faire un truc du genre...

```
<html>
     <head>
          <title><?php echo $title; ?></title>
     </head>
     <body>
          <h1><?php echo $heading; ?></h1>
          <h3>Ma liste de taches</h3>
          <ul>
          <?php foreach ($todo_list as $item): ?>
             <li><?php echo $item; ?></li>
          <?php endforeach;?>
          </ul>
     </body>
```

```
</html>
```

Renvoyer une vue comme une donnée

L'objet View possède une deuxième méthode qui vous permet de renvoyer des données sous forme de chaîne plutôt que de les envoyer à votre navigateur. Cela peut être utile si vous souhaitez traiter les données d'une manière ou d'une autre (utilisation du cache par fichier, utile pour le design des mails a envoyer). Si vous utilisez la méthode get() à la place de la méthode render(), elle renverra des données. Vous pouvez toujours envoyer une variable à la méthode view() si vous souhaitez que les données soient renvoyées:

```
$string = $this->view('myfile')->get();
```


Le modèle

Les modèles

Introduction

Les modèles sont des classes PHP conçues pour fonctionner avec les informations de votre base de données. Par exemple, supposons que vous utilisez dFramework pour gérer un blog. Vous pouvez avoir une classe de modèle qui contient des fonctions pour insérer, mettre à jour et récupérer les données de votre blog. Voici un exemple de ce à quoi pourrait ressembler une telle classe de modèle:

[code]

Par souci de simplicité dans cet exemple, nous utilisons $_POST directement. Il s'agit généralement d'une mauvaise pratique et une approche

plus courante consisterait à utiliser la bibliothèque de données `$this->data->post('title')`.

Anatomie d'un modèle

Les classes de modèles sont stockées dans votre répertoire */app/models/*. Ils peuvent être imbriqués dans des sous-répertoires si vous souhaitez ce type d'organisation.

Le prototype de base d'une classe de modèle est le suivant:

```
<html>
<head>
     <title>Mon Blog</title>
</head>
<body>
     <h1>Bienvenu sur mon Blog!</h1>
     <p>Vous trouverez ici toutes mes informations et
activités</p>
</body>
</html>
```

Où **NameModel** est le nom de votre classe. Les noms de classe **doivent** avoir la première lettre en majuscule et être suffixé de **Model**. Assurez-vous que votre classe étend la classe **dFramework\core\Model**.

Le nom de fichier doit correspondre au nom de la classe. Par exemple, si c'est votre classe:

[code]

Alors votre fichier sera le suivant: */app/models/UserModel.php*

Chargement d'un modèle

Vos modèles seront généralement chargés et appelés à partir de vos méthodes de contrôleur . Pour charger un modèle, vous utiliserez la méthode suivante:

```
$this->loadModel('model_name');
```

Si votre modèle se trouve dans un sous-répertoire, incluez le chemin relatif de votre répertoire de modèles. Par exemple, si vous avez un modèle situé dans */app/models/blog/QueriesModel.php*, vous le *chargerez en* utilisant:

```
$this->loadModel('blog/Queries');
```

Une fois chargé, vous accéderez à vos méthodes de modèle en utilisant un objet du même nom que votre classe:

```
$this->loadModel('model_name');
$this->model_name->my_method();
```

Si vous souhaitez que votre modèle soit affecté à un nom d'objet différent, vous pouvez le spécifier via le deuxième paramètre de la méthode de chargement:

```
$this->loadModel('model_name' ,  'foobar');
$this->foobar->my_method();
```

```
<?php
    class BlogController extends
dFramework\core\Controller {
        public function index() {
            $this->view('blogview')->render();
        }
    }
```

Si vous ouvrez votre navigateur avec l'URL appropriée, vous verez une page correspondant au code entré dans votre fichier de vue. L'URL peut être par exemple *http://localhost/my-df-app/blog*

Chargement de plusieurs vues

dFramework gérera intelligemment plusieurs appels à view() à partir d'un contrôleur. Si plusieurs appels se produisent, ils seront ajoutés ensemble. Par exemple, vous pouvez souhaiter avoir une vue d'en-tête, une vue de menu, une vue de contenu et une vue de pied de page. Cela pourrait ressembler à ceci:

```
    <?php
        class PageController extends
dFramework\core\Controller {
            public function index()
            {
               $this->view('header')->render();
               $this->view('menu')->render();
               $this->view('content')->render();
               $this->view('footer')->render();
            }
        }
```

Structurez vos dossiers de vues

Dossiers relatifs et dossiers absolus

dFramework gérera intelligemment plusieurs appels à view() à partir d'un contrôleur. Si plusieurs appels se produisent, ils seront ajoutés ensemble. Par exemple, vous pouvez souhaiter avoir une vue d'en-tête, une vue de menu, une vue de contenu et une vue de pied de page. Cela pourrait ressembler à ceci:

[code]

Placez vos vues dans des sous répertoires

Vos fichiers de vue peuvent également être stockés dans des sous-répertoires si vous préférez ce type d'organisation. Pour ce faire, vous devrez inclure le nom du répertoire chargeant la vue. Exemple:

```
$this->view('sous_dossier/fichier')->render();
```

Ajout des données dynamiques à la vue

Les données sont transmises du contrôleur à la vue au moyen d'un tableau dans le deuxième paramètre de la méthode de chargement de la vue. Voici un exemple simple:

```
$data = array(
    'title' => 'Mon titre',
    'heading' => 'Mon entete',
    'message' => 'Mon message'
);
$this->view('blogview', $data)->render();
```

Allez dans votre contrôleur d'exemple et modifiez le comme ceci

```
<?php
     class BlogController extends
dFramework\core\Controller {
          public function index()
          {
             $data['title'] = "Mon vrai titre";
             $data['heading'] = "L'entete plus realiste";
             $this->view('blogview', $data)->render();
          }
     }
```

Puis dans votre fichier de vue, essayez le code suivant

```
<html>
<head>
     <title><?php echo $title; ?></title>
</head>
<body>
     <h1><?php echo $heading; ?></h1>
</body>
</html>
```

Création de boucles

Le tableau de données que vous transmettez à vos fichiers d'affichage n'est

pas limité à de simples variables. Vous pouvez passer des tableaux multidimensionnels, qui peuvent être bouclés pour générer plusieurs lignes. Par exemple, si vous extrayez des données de votre base de données, elles se présentent généralement sous la forme d'un tableau multidimensionnel.

Voici un exemple simple. Ajoutez ceci à votre contrôleur:

```
<?php
        class BlogController extends
dFramework\core\Controller {
                public function index()
                {
                   $data['todo_list'] = array('Netoyer la
maison', 'Appeler maman', 'Visionner');
                   $data['title'] = "Mon vrai titre";
                   $data['heading'] = "L'entete plus realiste";
                   $this->view('blogview', $data)->render();
                }
        }
```

Dans votre fichier de vue, il ne vous restera plus qu'a faire un truc du genre...

```
<html>
     <head>
          <title><?php echo $title; ?></title>
     </head>
     <body>
          <h1><?php echo $heading; ?></h1>
          <h3>Ma liste de taches</h3>
          <ul>
          <?php foreach ($todo_list as $item): ?>
             <li><?php echo $item; ?></li>
          <?php endforeach;?>
          </ul>
     </body>
</html>
```

Renvoyer une vue comme une donnée

L'objet View possède une deuxième méthode qui vous permet de renvoyer

des données sous forme de chaîne plutôt que de les envoyer à votre navigateur. Cela peut être utile si vous souhaitez traiter les données d'une manière ou d'une autre (utilisation du cache par fichier, utile pour le design des mails a envoyer). Si vous utilisez la méthode get() à la place de la méthode render(), elle renverra des données. Vous pouvez toujours envoyer une variable à la méthode view() si vous souhaitez que les données soient renvoyées:

```
$string = $this->view('myfile')->get();
```


Gestion des routes

Les helpers

Utilisation des librairies

Créer vos librairies

Le rendu final

Le système de Layout

Gestion du cache

Smarty

Gestion de la base de données

Configuration

Le constructeur de requêtes

FluentPDO

Hydratation et migrations

Libraires

Api

La bibliothèque de communications externes : Api

dFramework implémente une mini bibliothèque de communication avec des services externes à l'aide de la librairie php-requests (https://github.com/rmccue/Requests). Le framework offre une couche d'abstraction vous permettant d'effectuer des appels aux APIs très simplement. En une seule ligne, émettez une requête sécurisée vers une Api quelconque.

Utilisation

Importer la classe

Comme la majorités des bibliothèques de dFramework, la bibliothèque **Api** est initialisée dans votre contrôleur à l'aide de la commande `$this->loadLlibrary()` comme le montre le code suivant:

```
$this->loadLibrary('Api');
```

Une fois chargé, l'objet de la bibliothèque Api sera disponible en utilisant:

```
$this->api;
```

Définition de l'URL de base

En principe, une fois que vous avez chargé la bibliothèque Api, vous n'avez plus rien à faire. Nous vous recommandons cependant d'initialiser l'URL de base du service que vous voulez appeler. Cela vous permettre de ne plus à avoir à entrer toute l'URL en absolue lors des appels mais juste la partie relative. Ceci est utile si vous devez appeler plusieurs URL du même service (nom de domaine identique).

Pour définir une URL de base pour vos appels à l'Api, vous devez utiliser la méthode baseUrl() de l'objet Api qui prend en paramètre une chaîne de caractère représentant l'Url de base du service externe au quel vous accéderez .

Supposons que vous utilisez le service *http://www.api.com* pour gérer vos clients et produits (ajout, modification, suppression, affichage). En tant normal, vous devez accéder aux URLs *http://www.api.com/votre_cle/clients* et *http://www.api.com/votre_cle/produits* respectivement pour gérer les clients et les produits. En utilisant la méthode baseUrl(), vous n'auriez plus qu'à spécifier la dernière partie de l'URL, le reste sera automatiquement rajouté

```
$this->api->baseUrl('http://www.api.com/votre_cle');
```

Une fois ceci fait, vous n'auriez plus qu'à faire

```
$this->api->get('clients');
$this->api->get('produits');
```

au lieu de

```
$this->api->get('http://www.api.com/votre_cle/clients');
$this->api->get('http://www.api.com/votre_cle/produits');
```

Généralement, la méthode baseUrl() doit être lancée dans le constructeur de votre contrôleur pour qu'elle agisse partout

Spécification du format du résultat

Vous pouvez également utiliser la méthode returnType() pour définir le format du résultat de l'appel des services externes.Il admet quatre valeurs spécifiant le type de formatage données renvoyées.

Si vous choisissez de formater le résultat (données + entêtes), vous obtiendrez un tableau contenant d'une part les métas données (code *(code de statut)*, success *(état de la requête)*, content-type *(le type de données renvoyés)*) et les données proprement dit. Ci dessous, nous avons un exemple de résultat renvoyé

```
[
     'meta' => [
          'code' => 200,
          'success' => true,
          'content-type' => 'application/json'
     ],
     'data' => [
          {
               "id" : 1,
               "name" : "dframework"
          },
          {
               "id" : 2,
               "name" : "codeigniter"
          }
     ]
]
```

Si vous ne précisez pas que les données seront formatés, vous obtiendrez un résultat brut que vous devriez déboguer (avec la fonction *var_dump()* par exemple) pour récupérer ce qui vous intéresse.

Références de classe

Propriétés et constantes

- ***const* BRUT**

  A utiliser comme paramètre de la méthode returnType() pour spécifier que le résultat des appels aux Api doivent renvoyé de façon brute (toutes les données relatives à la requête effectuée)

- ***const* FORMAT**

  A utiliser comme paramètre de la méthode returnType() pour spécifier que le résultat des appels aux Api doivent être formatés (tableau contenant les métas données basiques et les données issues du service web)

- ***const* DATAS**

  A utiliser comme paramètre de la méthode returnType() pour spécifier que le résultat des appels aux Api doivent contenir uniquement les données fournies par le service web. **C'est le comportement par défaut**

- ***const* METAS**

  A utiliser comme paramètre de la méthode returnType() pour spécifier que le résultat des appels aux Api ne doivent contenir que des métas données simples (code de statut, état de la requête, type de données renvoyés)

- ***const* HEADERS**

  A utiliser comme paramètre de la méthode returnType() pour spécifier que le résultat des appels aux Api doivent contenir uniquement la liste de toutes les entêtes renvoyées par le service web

Méthodes

- **`baseUrl`**`(string $url) : void`
  - Description: Définit l'url de base des appels aux services externes.
  - Paramètres
    - $url *(string)* - L'URL de base du service externe
  - Type de retour : void
  - Résultat : Rien

  Exemple :

  ```
  $this->api->baseUrl('http://192.168.137.1/ssm/api');
  ```

- **`returnType`**`(int $type) : void`
  - Description: Définit le type de formatage des données issus de l'appel d'un service externe.
  - Paramètres
    - $type *(dF_Api::FORMAT | dF_Api::BRUT)* - Le type de formatage à appliquer au résultat de l'appel au service externe.
  - Type de retour : void
  - Résultat : Rien

  Exemple :

  ```
  $this->api->returnType(dF_Api::BRUT);
  ```

  - Note: Par défaut les données sont formatés. Donc vous n'utiliserai jamais la constante *dF_Api::FORMAT*. Quand à la constante *dF_Api::BRUT*, elle est généralement utilisée pour le débogage.

- **`get`**`(string $url [, $headers = array()[, $options = array()] ]) : mixed`
  - Description: Envoie une requête de type GET à un service externe.
  - Paramètres
    - $url *(string)* - L'URL du service à appeler
    - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête

        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Un tableau avec les données retourner par le service web ou un tableau contenant des informations spécifique en fonction de la constante utilisée comme paramètre de ma méthode returnType().

    - **Note :** En cas d'utilisation de la constante *dF_Api::BRUT* comme paramètre de la méthode returnType(), un objet de type *Request_response* vous sera renvoyé et vous devez alors le déboguer (avec la fonction *var_dump()* par exemple) pour extraire les données qui vous seront utile

    Exemple :

```
$result = $this->api->get('http://192.168.137.1/ssm/api/articles');
     Si vous avez défini l'URL de base, vous pouvez simplement faire
$result = $this->api->get('articles');
```

- `post(string $url [, $headers = array()[, $data = array()[, $options = array()] ] ]) : mixed`
    - Description: Envoie une requête de type POST à un service externe.
    - Paramètres
        - $url *(string)* - L'URL du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $data *(array|null)* - Les données à envoyer aux service appelé
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

    Exemple :

```
$this->api->post('articles', null, [
```

```
        'nomArticle' => 'Laptop Dell Core 2Duo',
        'prix' => 150000,
        'reference' => 'LDC2D'
    ]);
```

- **head**(string $url [, $headers = array()[, $options = array()]]) : mixed
    - Description: Envoie une requête de type HEAD à un service externe.
    - Paramètres
        - $url *(string)* - L'URL du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

    Exemple :

    ```
    $this->api->head('articles', null, ['timeout' => 60]);
    ```

- **delete**(string $url [, $headers = array()[, $options = array()]]) : mixed
    - Description: Envoie une requête de type DELETE à un service externe.
    - Paramètres
        - $url *(string)* - L'URL du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

    Exemple :

    ```
    $this->api->delete('articles?id=3', null, [
    ```

```
        'auth' => ['username', 'password']
    ]);
```

- **trace**(string $url [, $headers = array()[, $options = array()]]) : mixed
    - Description: Envoie une requête de type TRACE à un service externe.
    - Paramètres
        - $url *(string)* - L'url du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

- **put**(string $url [, $headers = array()[, $data = array()[, $options = array()]]]) : mixed
    - Description: Envoie une requête de type PUT à un service externe.
    - Paramètres
        - $url *(string)* - L'URL du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $data *(array|null)* - Les données à envoyer aux service appelé
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

    Exemple :

    ```
    $this->api->put('articles', null, [
        'idArticle' => 1
        'prix' => 175000
    ], ['useragent' => 'dFramework/v2']);
    ```

- `patch(string $url [, $headers = array()[, $data = array()[, $options = array()] ] ]) : mixed`
    - Description: Envoie une requête de type PATCH à un service externe.
    - Paramètres
        - $url *(string)* - L'URL du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $data *(array|null)* - Les données à envoyer aux service appelé
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

- `options(string $url [, $headers = array()[, $data = array()[, $options = array()] ] ]) : mixed`
    - Description: Envoie une requête de type OPTIONS à un service externe.
    - Paramètres
        - $url *(string)* - L'URL du service à appeler
        - $headers *(array|null)* - Les entêtes supplémentaires à envoyer avec la requête
        - $data *(array|null)* - Les données à envoyer aux service appelé
        - $options *(array|null)* - Les options de la requête. Les options possibles seront donnés ci-dessous
    - Type de retour : array|Request_response
    - Résultat : Voir résultat de la méthode get()

Options pris en charge

Le tableau ci-dessous donne la liste des options pris en compte par les méthodes d'appel aux services. Pour plus d'informations, vous pouvez vous rendre sur l'API de notre framework (https://dimtrov.hebfree.org/docs/dframework/api)

Options	Valeur par défaut	Description	Valeur acceptées
timeout	10	Temps d'attente de la réponse en seconde	float. *Vous pouvez entrer des secondes avec une précision en milliseconde (ex: 0.01)*
connect_timeout	10	Durée à attendre avant de retenter un nouvel appel au service externe	float. *Vous pouvez entrer des secondes avec une précision en milliseconde (ex: 0.01)*
useragent	dframework/$version	Useragent à envoyer au serveur	string
follow_redirects	true		bool
redirects	10	Nombre de redirection autorisée avant la génération d'erreur	int
blocking	true		bool
auth	false	Instance de la classe Requests_Auth ou tableau (user et password) à utiliser pour l'authentification Basic	false/array/Requests_Auth
proxy	false	Detqils de proxy pour le proxy by-passing et l'authentification	false/array/string/Requests_proxy
max_bytes	false	Limite de la taille du corps de la reponse	false/int
idn	true	Active l'IDN parsing	bool
verify	/system/dependencies/rmccue/requests/src/Requests/Transport/cacert.p	Spécifie si on doit vérifier le certificat SSL. Vous pouvez passer le	string/bool

	em	chemin vers le fichier contenant le certificat à vérifier	
verifyname	true	Spécifie si on doit vérifier le nom du propriétaire dans le certificat SSL	bool

La librairie API est une interface à la dépendance php-requests. Pour utiliser toutes les fonctionnalités offerte par cette dépendance, vous pouvez visiter http://requests.ryanmccue.info/api/

Captcha

Crypto

La bibliothèque de chiffrement : Crypto

NE PAS utiliser cette bibliothèque de chiffrement ni aucune autre pour le stockage du mot de passe de l'utilisateur! Les mots de passe doivent être hachés à la place, et vous devez le faire via la propre extension PHP Hachage de mots de passe

Notre bibliothèque de chiffrement fournit un chiffrement de données bidirectionnel. Pour ce faire, il utilise des extensions PHP malheureusement pas toujours disponibles sur tous les systèmes. Vous devez avoir l'une des dépendances suivantes pour utiliser cette bibliothèque:

- OpenSSL

Si aucune des dépendances ci-dessus n'est remplie, nous ne pourrons pas vous proposer une implémentation suffisamment bonne pour répondre aux normes élevées requises pour une cryptographie correcte.

Utilisation

Importer la classe

Comme la majorités des bibliothèques de dFramework, la bibliothèque **Crypto** est initialisée dans votre contrôleur à l'aide de la commande `$this->loadLlibrary()` comme le montre le code suivant:

```
$this->loadLibrary('Crypto');
```

Une fois chargé, l'objet de la bibliothèque Crypto sera disponible en utilisant:

```
$this->crypto;
```

Comportement par défaut

Par défaut, la bibliothèque Crypto utilise le chiffrement AES-128 en mode CBC, à l'aide de votre *clé de chiffrement* configurée et de l'authentification SHA512

HMAC.

Cependant, la *clé de chiffrement* n'est pas utilisée telle quelle. En effet, un HMAC nécessite également une clé secrète et l'utilisation de la même clé à la fois pour le cryptage et l'authentification est une mauvaise pratique. De ce fait, deux clés distinctes sont dérivées de votre *clé de chiffrement* déjà configurée : une pour le chiffrement et une pour l'authentification. Cela se fait via une technique appelée HKDF (Key Derivation Function).

Configurez vos paramètres par défauts

Une *clé de chiffrement* est une information qui contrôle le processus de cryptographie et permet de chiffrer une chaîne de texte en clair, puis de la déchiffrer. C'est l'ingrédient secret de l'ensemble du processus qui vous permet d'être le seul à pouvoir déchiffrer les données que vous avez décidé de cacher aux yeux du public. Une fois qu'une clé est utilisée pour chiffrer des données, cette même clé constitue le **seul** moyen de le déchiffrer. Vous devez donc non seulement en choisir une avec soin, mais vous ne devez pas la perdre, sinon vous perdrez également l'accès aux données.

Votre clé de cryptage **doit** être aussi longue que l'algorithme de cryptage utilisé le permet. Pour AES-128, 128 bits ou 16 octets (caractères) sont longs. Vous trouverez ci-dessous un tableau indiquant les longueurs de clé prises en charge par différents chiffrements.

La clé doit être aussi aléatoire que possible et ne **doit pas** être une chaîne de texte normale, ni la sortie d'une fonction de hachage, etc. Pour créer une clé appropriée, vous devez utiliser la méthode genKey() de la bibliothèque Crypto.

```
// $key se verra attribuer une clé aléatoire de 16 octets (128 bits)
$key = $this->crypto->genKey(16);
```

La clé générée peut être stockée dans votre *app / config / data.php* pour être stockée et la transmise de manière dynamique lors du cryptage / décryptage. Pour enregistrer votre clé dans votre fichier *app / config / data.php* , ouvrez le fichier et définissez:

```
$config['encryption'] = [
    'key' => 'VOTRE_CLE'
];
```

La méthode genKey() génère des données binaires, difficilement traitable (par exemple, un copier-coller peut l' endommager). Vous pouvez utiliser bin2hex() ou hex2bin() pour travailler avec la clé d'une manière plus conviviale. Par exemple:

```
// Obtient une représentation codée en hexadécimal de la clé:
$key = bin2hex($this->crypto->genKey(16));

// Mettez la même valeur dans votre configuration avec hex2bin (),
// pour qu'elle soit toujours transmise sous forme binaire à la bibliothèque:
$config['encryption'] = [
    'key' => hex2bin(< votre clé codée en hexadécimal >)
];
```

Méthodes de chiffrement pris en charge

Les différentes méthodes de cryptage ont des caractéristiques différentes et servent à des fins différentes. Certains sont plus forts que d'autres, certains sont plus rapides et certains offrent des fonctionnalités supplémentaires. Vous trouverez ci-dessous la liste des méthodes de chiffrements pris en charge par dFramework:
AES-128-CBC, AES-128-CFB, AES-128-CFB1, AES-128-CFB8, AES-128-OFB, AES-192-CBC, AES-192-CFB, AES-192-CFB1, AES-192-CFB8, AES-192-OFB, AES-256-CBC, AES-256-CFB, AES-256-CFB1, AES-256-CFB8, AES-256-OFB, BF-CBC, BF-CFB, BF-OFB, CAST5-CBC, CAST5-CFB, CAST5-OFB, IDEA-CBC, IDEA-CFB, IDEA-OFB

- **Longueur du message**

  Il est important pour vous de savoir qu'une chaîne cryptée est généralement plus longue que la chaîne d'origine en texte brut (selon le chiffrement). Ceci est influencé par l'algorithme de chiffrement lui-même, l'IV ajouté au texte chiffré et le message d'authentification HMAC également ajouté. En outre, le message crypté est également codé en Base64, de sorte qu'il est sécurisé pour le stockage et la transmission, quel que soit le jeu de caractères utilisé.
  Gardez ces informations à l'esprit lors de la sélection de votre mécanisme de stockage de données. Les cookies, par exemple, ne peuvent contenir que 4K d'informations.

- **Configurer la librairie**

  Comme indiqué dans la section «Comportement par défaut» ci-dessus, vous devez utiliser le sélecteur AES-128 en mode CBC et votre `$data['encryption']['key']`.

Les paramètres de chiffrement globaux sont définis via le fichier *app / config / data.php* à travers la syntaxe suivante

```
$data['encryption'] = [
     'key'     => 'VOTRE_CLE',
     'algo'    => 'ALGORITHME_A_UTILISER',
     'add_hmac => true/false
];
```

Ces paramètres, seront automatiquement utiliser pour tous vos processus de chiffrement/de-chiffrement.
Toutefois, vous pouvez changer cela pour une section particulière en utilisant la méthode `set()` pour spécifier les options de cryptage souhaitées. Elle accepte un tableau associatif de paramètres, qui sont tous facultatifs:

Options	Valeur par défaut	Description	Valeur acceptées
algo	AES-128-CBC	La méthode de	Toutes les méthodes de

		chiffrement à utiliser	chiffrement énumérées ci-dessus
key	$data['encryption'] ['key']	La clé de chiffrement à utiliser	chaîne de caractères
add_hmac	true	Spécifié si on doit ajouter un message d'authentification HMAC à la chaîne chiffrée	true/false

Par exemple, vous pouvez modifier l'algorithme de chiffrement, la clé et refuser l'ajout de message d'authentification en procédez comme suit:

```
    $this->crypto->set([
        'algo'         =>  'AES-256-OFB',
        'add_hmac'     =>  false ,
         'key'         =>  '<chaîne aléatoire de 32
caractères>'
    ]);
```

Comme indiqué précédemment, il est important que vous choisissiez une clé de taille appropriée pour l'algorithme utilisé.

Crypter et décrypter les données

Le chiffrement et le déchiffrement des données avec les paramètres de bibliothèque déjà configurés est simple. Il suffit simplement de passer la chaîne aux méthodes `encrypt()` et / ou `decrypt()` en fonction de l'opération souhaitée :

```
    $plain_text  =  'Ceci est un message en texte brut!' ;
    $ciphertext  =  $this->crypto->encrypt($plain_text);

    echo   $this->crypto->decrypt($ciphertext); // Résultats:
Ceci est un message en texte brut!
```

Et c'est tout! La bibliothèque de cryptage fera tout le nécessaire pour que le processus entier soit sécurisé de manière cryptographique. Vous n'avez pas besoin de vous en préoccuper.

Les deux méthodes renverront FALSE en cas d'erreur. Bien qu'avec encrypt(), cela ne puisse signifier qu'une configuration est incorrecte, vous devez toujours vérifier la valeur de retour du decrypt() en production.

Références de classe

Propriétés

Cette librairie n'a aucune propriété accessible

Méthodes

- **`encrypt`**`(string $data) : string`
  - Description: Crypte les données d'entrée et retourne son texte chiffré.
  - Paramètres
    - $data *(string)* - Données à chiffrer
  - Type de retour : string
  - Résultat : Données crypter ou False en cas d'erreur

Exemple :

```
$ciphertext = $this->crypto->encrypt('Mon message secret');
```

- **decrypt**(string $data) : string
  - Description: Déchiffre les données d'entrée et les renvoie en texte brut.
  - Paramètres
    - $data *(string)* - Données à décrypter
  - Type de retour : string
  - Résultat : Données déchiffrées ou FALSE en cas d'échec

Exemple :

```
echo $this->crypto->decrypt($ciphertext);
```

- **genKey**(int $length) : string
  - Description: Crée une clé cryptographique en récupérant des données aléatoires à partir des sources du système d'exploitation.
  - Paramètres
    - $length *(string)* - Longueur de sortie
  - Type de retour : string
  - Résultat : Une clé cryptographique pseudo-aléatoire de longueur spécifiée ou FALSE si une erreur survient

Exemple :

```
$key = $this->crypto->genKey(16);
```

- **set**(array $params) : void
  - Description: Initialise (configure) la bibliothèque pour utiliser une configuration (méthode ou une clé de chiffrement) différente de la configuration par défaut.
  - Paramètres :
    - $params *(array)* - Paramètres de configuration
  - Type de retour : void

Exemple :

```
$this->crypto->set(['algo' => 'AES-256-CBC']);
```

- Note: Veuillez vous reporter à la section "Configurer la librairie" pour des informations détaillées.

Mail

La bibliothèque d'envoi d'email : Mail

dFramework a une bibliothèque d'envoi d'email assez robuste. Elle est basée sur la classe PHPMailer (https://github.com/PHPMailer/PHPMailer) et possède les fonctionnalités suivantes:

- Plusieurs protocoles: Mail, SendMail, QMail et SMTP
- Cryptage TLS et SSL pour SMTP
- Destinataires multiples
- CC et BCC
- Courriel HTML ou Plaintext
- Les pièces jointes
- Les priorités
- Outils de débogage d'email

Utilisation

Importer la classe

Comme la majorités des bibliothèques de dFramework, la bibliothèque **Mail** est initialisée dans votre contrôleur à l'aide de la commande `$this->loadLlibrary()` comme le montre le code suivant:

```
$this->loadLibrary('Mail');
```

Une fois chargé, l'objet de la bibliothèque Mail sera disponible en utilisant:

```
$this->mail;
```

Connexion au serveur de messagerie

Avant d'envoyer un mail, il faut d'abord se connecter à un serveur de messagerie. Cela se fait par l'utilisation de la méthode `connect()` de l'objet Mail qui prend en paramètres un tableau associatif contenant les informations de connexion au serveur. L'exemple ci-dessous est un cas pratique de connexion au serveur de messagerie de gmail

```
$this->mail->connect([
    'host'      => 'smtp.gmail.com',
    'username'  => 'user@gmail.com',
    'password'  => 'userpassword'
]);
```

Le tableau ci-dessous présente la liste exhaustive des paramètres de connexion supportées

Options	Valeur par défaut	Description	Valeur acceptées

host	/	L'adresse du serveur de messagerie auquel on souhaite se connecter	chaîne de caractères
username	/	L'utilisateur qui souhaite se connecter	chaîne de caractères
password	/	Le mot de passe de l'utilisateur	chaîne de caractères
port	587	Le port d'écoute du serveur mail	entier
debug	false	Spécifie si on renvoi un debug serveur après l'envoie du mail	true/false

Envoi d'email

Une fois connecté à un serveur de messagerie. Vous pouvez envoyer vos emails en toute sérénité. La bibliothèque Mail utilise des noms de méthodes parlante et le principe de fluent (chaînage) pour vous faciliter la tache. Ainsi, pour envoyer un mail, vous pouvez utiliser le code suivant (*après avoir été connecté à un serveur)*.

```
$this->mail
     ->from('dframework@dimtrov.com', 'dFramework')
     ->to('dev.dimitrisitchet@gmail.com',        'Dimitric
Sitchet Tomkeu')
     ->subject('Test de fonctionnalité')
     ->message("Un  petit  test  d'envoi  d'email  avec  la
bibliotheque <b>Mail</b>");
if($this->mail->send()) {
     echo 'Message envoyé avec succès';
}
else {
     echo 'Echec d\'envoi';
}
```

Plusieurs autres méthodes sont disponible pour la construction d'un mail. Veuillez vous referez à la référence de classe pour plus de détails.

Préférences de messagerie

Il existe 5 préférences différentes pour personnaliser le mode d'envoi de vos e-mails. Les préférences sont définies en transmettant un tableau de valeurs à la méthode set() de l'objet Mail. Voici un exemple de la façon dont vous pourriez définir certaines préférences:

```
$this->mail->set([
        'charset' => 'utf-8',
        'method' => 'qmail',
        'timeout' => 60
]);
```

Toutes nos préférences ont des valeurs par défaut qui seront utilisées si vous ne les définissez pas.

- **Définition des préférences de messagerie dans un fichier de configuration**

  Si vous préférez ne pas définir de préférences à l'aide de la méthode ci-dessus, vous pouvez utiliser le fichier de configuration *app / config / email.php*. Mettez tous vos préférences dans le tableau $email['nom_de_la_config']['set'] et vous n'auriez plus qu'à faire `$this->mail->use('nom_de_la_config');` pour que celles-ci soient automatiquement utilisées à chaque fois que vous enverrez un e-mail. Bien-sur, vous pouvez modifier une préférence précise lors de d'un envoi particulier. Vous pouvez également défini les paramètres de connexion au serveur dans ce fichier. Cela vous permettra de ne plus le faire au prochain envoi. Pour cela, utiliser le tableau $email['nom_de_la_config']['connect'] pour définir ces paramètres.

- **Préférences**

Vous trouverez ci-dessous une liste de toutes les préférences pouvant être définies lors de l'envoi d'un courrier électronique.

Options	Valeur par défaut	Description	Valeur acceptées
method	SMTP	Le protocole à utiliser pour l'envoi du mail	SMTP, MAIL, QMAIL, SENDMAIL
timeout	300	Délai d'attente SMTP (en secondes)	entier
charset	utf-8	Jeu de caractères	utf-8, iso-8859-1, etc.
priority	null	Priorité courriel. 1 = plus élevé. 5 = le plus bas. 3 = normal.	1, 3, 5
encryption	tls	Cryptage SMTP	tls, ssl

Références de classe

Propriétés et constantes

Cette librairie n'a aucune propriété et constante accessible

Méthodes

- `from(string $address [, string $name = '']) : dF_Mail`
  - Description: Définit l'adresse email et le nom de la personne qui envoie l'e-mail.
  - Paramètres
    - $address *(string)* - L'adresse e-mail
    - $name *(string)* - Le nom à afficher
  - Type de retour : dF_Mail
  - Résultat : Instance dF_Mail (chaînage de méthodes)

Exemple :

```
$this->mail->from('me@example.com', 'John Doe');
```

- **to**(**$address** [, **string** **$name** = '']) : **dF_Mail**
  - Description: Définit la ou les adresses électroniques du ou des destinataires. Peut être un simple courrier électronique, un courrier et un nom de destinataire ou un tableau associatif ayant les adresses de destinataires comme clés et leurs nom comme valeur.
  - Paramètres
    - $address *(string|array)* - L'adresse e-mail ou tableau d'adresse des destinataires
    - $name *(string)* - Le nom à afficher
  - Type de retour : dF_Mail
  - Résultat : Instance dF_Mail (chaînage de méthodes)

  Exemple :

```
$this->mail->to('you@example.com', 'Doe John');
$this->mail->to([
     'mark@facebook.com' => 'Mark Zurkemberg',
     'lary@google.com' => 'Lary Page',
     'bill@microsoft.com => 'Bill Gates'
]);
```

  - Note: Lorsque le premier paramètre est un tableau, le deuxième n'est plus pris en compte. On peut donc l'ignorer

- **reply**(**$address** [, **string** **$name** = '']) : **dF_Mail**
  - Description: Définit la ou les adresses électroniques du ou des destinataires. Peut être un simple courrier électronique, un courrier et un nom de destinataire ou un tableau associatif ayant les adresses de destinataires comme clés et leurs nom comme valeur.
  - Paramètres
    - $address *(string|array)* - L'adresse e-mail ou tableau d'adresse des

destinataires
    - $name *(string)* - Le nom à afficher
  - Type de retour : dF_Mail
  - Résultat : Instance dF_Mail (chaînage de méthodes)

  Exemple :

```
$this->mail->reply('you@example.com', 'Doe John');
$this->mail->reply([
     'mark@facebook.com' => 'Mark Zurkemberg',
     'lary@google.com' => 'Lary Page',
     'bill@microsoft.com => 'Bill Gates'
]);
```

  - Note: Lorsque le premier paramètre est un tableau, le deuxième n'est plus pris en compte. On peut donc l'ignorer

- `cc($address [, string $name = '']) : dF_Mail`
  - Description: Définit la ou les adresses électroniques du ou des destinataires. Peut être un simple courrier électronique, un courrier et un nom de destinataire ou un tableau associatif ayant les adresses de destinataires comme clés et leurs nom comme valeur.
  - Paramètres
    - $address *(string|array)* - L'adresse e-mail ou tableau d'adresse des destinataires
    - $name *(string)* - Le nom à afficher
  - Type de retour : dF_Mail
  - Résultat : Instance dF_Mail (chaînage de méthodes)

  Exemple :

```
     $this->mail->reply('you@example.com', 'Doe John');
     $this->mail->reply([
          'mark@facebook.com' => 'Mark Zurkemberg',
          'lary@google.com' => 'Lary Page',
          'bill@microsoft.com => 'Bill Gates'
```

```
]);
```

- Note: Lorsque le premier paramètre est un tableau, le deuxième n'est plus pris en compte. On peut donc l'ignorer

- `connect(array $params) : void`
  - Description: Définie les paramètres de connexion au serveur de messagerie de façon locale. Utile si l'on souhaite utiliser un paramètre différents de ceux spécifiés dans le fichier de configuration.
  - Paramètres
    - $params *(array)* - Tableau associatif contenant les paramètres de connexion
  - Type de retour : void

- `set(array $params) : void`
  - Description: Définie les options de traitement de l'envoi d'email de façon locale. Utile si l'on souhaite utiliser un paramètre différents de ceux spécifiés dans le fichier de configuration.
  - Paramètres
    - $params *(array)* - Tableau associatif contenant les options de traitement
  - Type de retour : void

- `use(string $name) : void`
  - Description: Spécifie la configuration à prendre dans le fichier de configuration d'email pour la connexion au serveur de messagerie et pour le processus d'envoi d'email. Utile si on a configuré plusieurs paramètres.
  - Paramètres :
    - $params *(string)* - Nom du paramètres de configuration que l'on souhaite utiliser
  - Type de retour : void

Paginator

La bibliothèque de pagination : Paginator

NE PAS utiliser cette bibliothèque de *chiffrement* ni aucune autre pour le stockage du mot de passe de l'utilisateur! Les mots de passe doivent être *hachés à la* place, et vous devez le faire via la propre extension PHP Hachage de mots de passe

Notre bibliothèque de chiffrement fournit un chiffrement de données bidirectionnel. Pour ce faire, il utilise des extensions PHP malheureusement pas toujours disponibles sur tous les systèmes. Vous devez avoir l'une des dépendances suivantes pour utiliser cette bibliothèque:

- OpenSSL

Si aucune des dépendances ci-dessus n'est remplie, nous ne pourrons pas vous proposer une implémentation suffisamment bonne pour répondre aux normes élevées requises pour une cryptographie correcte.

- Utilisation
  - Importer la classe
  - Comportement par défaut
  - Configurer vos paramètres de chiffrement par défaut
  - Méthodes de chiffrement pris en charge
- Crypter et décrypter les données
- Référence de classe

Utilisation

Importer la classe

Comme la majorités des bibliothèques de dFramework, la bibliothèque **Crypto** est initialisée dans votre contrôleur à l'aide de la commande `$this->loadLlibrary()` comme le montre le code suivant:

```
$this->loadLibrary('Crypto');
```

Une fois chargé, l'objet de la bibliothèque Crypto sera disponible en utilisant:

```
$this->crypto;
```

Comportement par défaut

Par défaut, la bibliothèque Crypto utilise le chiffrement AES-128 en mode CBC, à l'aide de votre *clé de chiffrement* configurée et de l'authentification SHA512 HMAC.

Cependant, la *clé de chiffrement* n'est pas utilisée telle quelle. En effet, un HMAC nécessite également une clé secrète et l'utilisation de la même clé à la fois pour le cryptage et l'authentification est une mauvaise pratique. De ce fait, deux clés distinctes sont dérivées de votre *clé de chiffrement* déjà configurée : une pour le chiffrement et une pour l'authentification. Cela se fait via une technique appelée HKDF (Key Derivation Function).

Configurez vos paramètres par défauts

Une *clé de chiffrement* est une information qui contrôle le processus de cryptographie et permet de chiffrer une chaîne de texte en clair, puis de la déchiffrer. C'est l'ingrédient secret de l'ensemble du processus qui vous permet d'être le seul à pouvoir déchiffrer les données que vous avez décidé de cacher aux yeux du public. Une fois qu'une clé est utilisée pour chiffrer des données, cette même clé constitue le **seul** moyen de le déchiffrer. Vous devez donc non seulement en choisir une avec soin, mais vous ne devez pas la perdre, sinon vous perdrez également l'accès aux données.

Votre clé de cryptage **doit** être aussi longue que l'algorithme de cryptage utilisé le permet. Pour AES-128, 128 bits ou 16 octets (caractères) sont longs. Vous trouverez ci-dessous un tableau indiquant les longueurs de clé prises en charge par différents chiffrements.
La clé doit être aussi aléatoire que possible et ne **doit pas** être une chaîne de texte normale, ni la sortie d'une fonction de hachage, etc. Pour créer une clé appropriée, vous devez utiliser la méthode genKey() de la bibliothèque Crypto.

```
// $key se verra attribuer une clé aléatoire de 16 octets (128 bits)
$key = $this->crypto->genKey(16);
```

La clé générée peut être stockée dans votre *app / config / data.php* pour être stockée et la transmise de manière dynamique lors du cryptage / décryptage. Pour enregistrer votre clé dans votre fichier *app / config / data.php* , ouvrez le fichier et définissez:

```
$config['data']['encryption'] = [
    'key' => 'VOTRE_CLE'
];
```

La méthode genKey() génère des données binaires, difficilement traitable (par exemple, un copier-coller peut l' endommager). Vous pouvez utiliser bin2hex() ou hex2bin() pour travailler avec la clé d'une manière plus conviviale. Par exemple:

```
// Obtient une représentation codée en hexadécimal de la clé:
 $key  =  bin2hex($this->crypto->genKey(16));

// Mettez la même valeur dans votre configuration avec hex2bin (),
  // pour qu'elle soit toujours transmise sous forme binaire à la bibliothèque:
 $config['data']['encryption'] = [
    'key' =>  hex2bin(< votre  clé codée en hexadécimal
```

```
>)
    ];
```

Méthodes de chiffrement pris en charge

Les différentes méthodes de cryptage ont des caractéristiques différentes et servent à des fins différentes. Certains sont plus forts que d'autres, certains sont plus rapides et certains offrent des fonctionnalités supplémentaires. Vous trouverez ci-dessous la liste des méthodes de chiffrements pris en charge par dFramework:
AES-128-CBC, AES-128-CFB, AES-128-CFB1, AES-128-CFB8, AES-128-OFB, AES-192-CBC, AES-192-CFB, AES-192-CFB1, AES-192-CFB8, AES-192-OFB, AES-256-CBC, AES-256-CFB, AES-256-CFB1, AES-256-CFB8, AES-256-OFB, BF-CBC, BF-CFB, BF-OFB, CAST5-CBC, CAST5-CFB, CAST5-OFB, IDEA-CBC, IDEA-CFB, IDEA-OFB

- **Longueur du message**

  Il est important pour vous de savoir qu'une chaîne cryptée est généralement plus longue que la chaîne d'origine en texte brut (selon le chiffrement). Ceci est influencé par l'algorithme de chiffrement lui-même, l'IV ajouté au texte chiffré et le message d'authentification HMAC également ajouté. En outre, le message crypté est également codé en Base64, de sorte qu'il est sécurisé pour le stockage et la transmission, quel que soit le jeu de caractères utilisé.
  Gardez ces informations à l'esprit lors de la sélection de votre mécanisme de stockage de données. Les cookies, par exemple, ne peuvent contenir que 4K d'informations.
- **Configurer la librairie**

  Comme indiqué dans la section «Comportement par défaut» ci-dessus, vous devez utiliser le sélecteur AES-128 en mode CBC et votre $config['encryption']['key'].

Les paramètres de chiffrement globaux sont définis via le fichier *app / config / data.php* à travers la syntaxe suivante

```
$config['data']['encryption'] = [
     'key'     => 'VOTRE_CLE',
     'algo'    => 'ALGORITHME_A_UTILISER',
     'add_hmac => true/false
];
```

Ces paramètres, seront automatiquement utiliser pour tous vos processus de chiffrement/de-chiffrement.

Toutefois, vous pouvez changer cela pour une section particulière en utilisant la méthode set() pour spécifier les options de cryptage souhaitées. Elle accepte un tableau associatif de paramètres, qui sont tous facultatifs:

Options	Valeur par défaut	Description	Valeur acceptées
algo	AES-128-CBC	La méthode de chiffrement à utiliser	Toutes les méthodes de chiffrement énumérées ci-dessus
key	$config['data']['encryption']['key']	La clé de chiffrement à utiliser	chaîne de caractères
add_hmac	true	Spécifié si on doit ajouter un message d'authentification HMAC à la chaîne chiffrée	true/false

Par exemple, vous pouvez modifier l'algorithme de chiffrement, la clé et refuser l'ajout de message d'authentification en procédez comme suit:

```
$this->crypto->set([
     'algo'         =>  'AES-256-OFB',
     'add_hmac'     =>  false ,
      'key'         =>  '<chaîne aléatoire de 32
caractères>'
 ]);
```

Comme indiqué précédemment, il est important que vous choisissiez une clé de taille appropriée pour l'algorithme utilisé.

Populate

La bibliothèque d'alimentation des bases de données : Populate

Lorsqu'on développe des sites web, on a souvent besoin de données pour tester le rendu final de notre application. dFramework met à la disposition des développeurs une bibliothèque leur permettant d'alimenter leurs bases de données avec des informations proche de la réalité afin de réaliser des tests se rapprochant le plus des situations concrètes. Pour cela le framework a intégré dans son noyau la librairie Faker (https://github.com/fzaninotto/Faker). En quelques lignes seulement, vous disposerez d'une base de données pleine d'informations qui vous permettra de tester les choses de façon plus réelle.

Utilisation

Importer la classe

Comme la majorités des bibliothèques de dFramework, la bibliothèque **Populate** est initialisée dans votre contrôleur à l'aide de la commande `$this->loadLlibrary()` comme le montre le code suivant:

```
$this->loadLibrary('Populate');
```

Une fois chargé, l'objet de la bibliothèque Populate sera disponible en utilisant:

```
$this->populate;
```

Génération du contenu

Après avoir chargé la bibliothèque Populate, vous devez spécifier le type d'information à mettre dans chaque champs de votre table. Pour cela, vous devez utiliser la méthode generate() de l'objet Populate qui prend deux paramètres en entrée. Le premier étant le champ à remplir dans la base de donnée et le second, la fonction à appliquer pour générer la donnée à insérer.

Le code qui suit montre un exemple basic de génération de données pour le nom, le slug, la date de création et le contenu des articles pour un blog

```
$this->populate
     ->generate('title', 'sentence')
     ->generate('slug', 'slug')
     ->generate('create_at', 'date')
     ->generate('content', 'paragraph[6]');
```

Vous devez par la suite appliquer la méthode table() pour spécifier la table à remplir. Une fois ceci fait, vous n'auriez plus qu'à appeler la méthode run() pour démarrer le remplissage

```
$this->populate
     ->generate([
          'title' => 'sentence'
          'content' => 'paragraph[6]'
          ...
     ])
```

```
        ->table('articles')
        ->run();
```

Vous remarquez que la méthode generate() peut prendre un tableau associatif en premier paramètre, représentant les couples champs/fonctions. Le second paramètre sera alors ignoré dans ce cas.

Fonctions prises en charge pour la génération du contenu

Pour générer un contenu fictif, vous disposer de plusieurs fonctions produisant chacune un contenu spécifique. La liste ci-dessous présente quelque fonctions couramment utilisées.

Fonctions	**Paramètres**	**Données générées aléatoirement**	**Exemple**
randomDigit	/	Chiffre	randomDigit => 7
randomDigitNot	- entier : chiffre à ignorer	Chiffre différent de celui passé en paramètre	randomDigitNot[4] => 0, 1, 2, 3, 5, 6, 7, 8, ou 9
randomNumber	+ entier : nombre de chiffre à générer	Nombre entier comprenant n chiffre	randomNumber[5] => 70274
randomFloat	+ entier : nombre de chiffre maximal après la virgule (précision)	Nombre décimal	randomFloat[4] => 84.3892
	+ entier : valeur minimale du nombre à générer		
	+ entier : valeur maximale du nombre à générer		
numberBetween	- entier : valeur minimale	Nombre compris entre les deux nombre passés	numberBetween[1000, 3000] => 2530

	- entier : valeur maximale	en paramètres	
randomLetter	/	Lettre	randomLetter => d
randomElement	- tableau : tableau des éléments à partir desquels on va récupérer des valeurs aléatoires	Élément tiré du tableau passé en paramètre	randomElement[array('a','b','c','d')] => 'c'
shuffle	- chaîne : chaîne de caractères à mélanger	Chaîne mélangée	shuffle['hello, world'] => 'rlo,h eoldlw'
word	/	Mot	word => 'aut'
sentence	+ entier : nombre de mot que doit contenir la phrase + booléen : spécifie si les mots générés doivent avoir un nombre de caractères variable	Phrase content le nombre de mot spécifié	sentence[6, true] => 'Sit vitae voluptas sint non voluptates.'
paragraph	+ entier : nombre de phrase que doit contenir le paragraphe + booléen : spécifie si les phrases générées doivent avoir un nombre de mots variable	Paragraphe content le nombre de phrase spécifié	paragraph[3, true] => 'Ut ab voluptas sed a nam. Sint autem inventore aut officia aut aut blanditiis. Ducimus eos odit amet et est ut eum.'
text	+ entier : nombre maximal de caracteres	Texte ayant au plus 'n' caracteres	text[200] => 'Fuga totam reiciendis qui architecto fugiat nemo. Consequatur recusandae qui cupiditate eos quod.'
title	+(null | 'male' | 'female') :	Titre	title => 'Ms.'

	genre		
titleMale	/	Titre masculin	titleMale => 'Mr.'
titleFemale	/	Titre feminin	titleFemale => 'Ms.'
suffix	/	Préfixe quelconque	suffix => 'Jr.'
name	+(null | 'male' | 'female') : genre	Nom et prénom (peut être préfixe)	name => 'Dr. Zane Stroman'
firstName	+(null | 'male' | 'female') : genre	Prénom	firstName => 'Maynard'
firstNameMale	/	Prénom masculin	firstNameMale => 'Maynard'
firstNameFemale	/	Prénom feminin	firstNameFemale => 'Rachel'
lastName	/	Nom propre	lastName => 'Zulauf'

Références de classe

Propriétés et constantes

Cette librairie n'a aucune propriété et constante accessible

Méthodes

- `locale(string $locale) : dF_Populate`
  - Description: Spécifie la langue à utiliser pour générer le contenu. Par défaut le contenu est généré en français (fr_FR)
  - Paramètres
    - $locale *(string)* - La locale à utiliser
  - Type de retour : dF_Populate

- Résultat : Instance dF_Populate (Chaînage des méthodes)

Exemple :

```
$this->populate->locale('en_US');
```

- **table**(string $table, string $use_db = 'default') : dF_Populate
  - Description: Spécifie la table à remplir avec les données générées.
  - Paramètres
    - $table *(string)* - La table à remplir.
    - $use_db *(string)* - La configuration de base de données à utiliser.
  - Type de retour : dF_Populate
  - Résultat : Instance dF_Populate (Chaînage des méthodes)

- **generate**($field [, string $function = null]) : dF_Populate
  - Description: Défini les type de données à générer pour chaque colonne qu'on souhaite remplir dans la base de données.
  - Paramètres
    - $field *(string|array)* - Le champ qu'on souhaite remplir dans la table
    - $function *(string|null)* - La fonction désignant le type de donnés à remplir ci-dessous (Cf. Fonctions prises en charges)
  - Type de retour : dF_Populate
  - Résultat : Instance dF_Populate (Chaînage des méthodes)

  - Note : Cette fonction accepte en premier paramètre un tableau associatif représentant les champs souhaitées et les fonctions à appliquer pour les remplir. Dans ce cas, le second paramètres sera ignoré.

- **rows**(int $row) : dF_Populate
  - Description: Spécifie le nombre de ligne à insérer dans la base de données.
  - Paramètres
    - $row *(int)* - Le nombre d'enregistrement que l'on souhaite insérer dans la table

- Type de retour : dF_Populate
- Résultat : Instance dF_Populate (Chaînage des méthodes)

- **`run() : void`**
  - Description: Lance la génération des données et le remplissage de la base de données.
  - Type de retour : Void
  - Résultat : Rien

- **`join(string $merge, array $tables [, string $use_db = 'default' ]) : void`**
  - Description: Associe de façon aléatoire les clés de différentes tables dans une table d'association.
  - Paramètres
    - $merge *(string)* - La table d'association dans laquelle doit migrer les clés
    - $tables *(array)* - Tableau associatif contenant en clé le nom des champs des tables principales et comme valeurs, le nom des champs dans la table d'association
    - $use_db *(string|null)* - La configuration de la base de données à utiliser
  - Type de retour : Void
  - Résultat : Rien

  - Note : Les clés du tableau doivent être constituées du nom de la table et du champ a faire migrer. Les deux séparés par un point. Par ailleurs, cette fonction ne gère que les migrations d'associations binaires (2 tables)

  Exemple :

```
$this->populate->join('categories_articles', [
     'articles.id' => 'id_article',
     'categories.id' => 'id_categorie'
]);
```

Dans cet exemple, les tables "articles" et "categories" migreront les valeurs de leurs champs "id" respectivement dans les champs "id_article" et "id_categorie"

de la table d'association "categories_articles". Toutes les combinaisons seront faite de façon totalement aléatoire.

Ua

La bibliothèque des informations sur l'équipement de l'utilisateur : Ua

Grâce à la classe Browser développée par Chris Schuld (http://chrisschuld.com), dFramework vous offre un outils puissant de détection d'agent utilisateur. A travers la librairie **Ua** vous obtiendrez toutes les informations relative à l'équipement utilisé par vos utilisateurs pour accéder à votre application.

Utilisation

Importer la classe

Comme la majorités des bibliothèques de dFramework, la bibliothèque **Ua** est initialisée dans votre contrôleur à l'aide de la commande `$this->loadLlibrary()` comme le montre le code suivant:

```
$this->loadLibrary('Ua');
```

Une fois chargé, l'objet de la bibliothèque Ua sera disponible en utilisant:

```
$this->ua;
```

Références de classe

Propriétés et constantes

Cette librairie n'a aucune propriété et constante accessible

Méthodes

- **reset**() : void
  - Description: Réinitialise les propriétés de internes
  - Type de retour : void
  - Résultat : Rien

- **isBrowser**(string $browserName) : bool
  - Description: Vérifie si une chaîne donnée correspond au navigateur actuellement utilisé par le visiteur
  - Paramètres
    - $browserName *(string)* - Le nom du navigateur à vérifier
  - Type de retour : bool
  - Résultat : true si la chaîne passée en paramètre correspond au navigateur du visiteur et false sinon

  Exemple :
  ```
  var_dump($this->ua->isBrowser('Google Chrome')); //
  ```

```
bool(true)
var_dump($this->ua->isBrowser('Opera')); // bool(false)
```

- **browser**() : string
  - Description: Renvoi le nom du navigateur utilisé
  - Type de retour : string
  - Résultat : Nom du navigateur

- **platform**() : string
  - Description: Renvoi le nom de la plateforme utilisée par le visiteur
  - Type de retour : string
  - Résultat : Nom de la pateforme

  Exemple :

```
var_dump($this->ua->platform()); // string(7) "Windows"
```

- **version**() : string
  - Description: Renvoi la version du navigateur utilisée par le visiteur
  - Type de retour : string
  - Résultat : Version du navigateur

- **isAol**() : bool
  - Description: Vérifie si le navigateur provient de AOL
  - Type de retour : bool
  - Résultat : true si le navigateur utilisé provient de AOL et false sinon

- **isFacebook**() : bool
  - Description: Vérifie si le navigateur provient de Facebook
  - Type de retour : bool
  - Résultat : true si le navigateur utilisé provient de Facebook et false sinon

- **isMobile**() : bool

  - Description: Vérifie si l'équipement utilisé pour naviguer est un téléphone
  - Type de retour : bool
  - Résultat : true si le visiteur utilise un téléphone pour naviguer sur votre site et false sinon

- **`isTablet() : bool`**
  - Description: Vérifie si l'équipement utilisé pour naviguer est une tablette
  - Type de retour : bool
  - Résultat : true si le visiteur utilise une tablette pour naviguer sur votre site et false sinon

- **`isRobot() : bool`**
  - Description: Vérifie si l'agent utilisateur utilisé est un robot (ex: Slurp, GoogleBot)
  - Type de retour : bool
  - Résultat : true si le visiteur utilise une tablette pour naviguer sur votre site et false sinon

- **`userAgent() : string`**
  - Description: Récupère l'agent utilisateur utilisé par le visiteur
  - Type de retour : string
  - Résultat : Le useragent à partir des entêtes renvoyées

  Exemple :

```
echo $this->ua->userAgent(); // Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.0.4) Gecko/20060613 Camino/1.0.2
```

- **`setUserAgent(string $agent_string) : void`**
  - Description: Modifie la valeur de l'agent utilisateur
  - Paramètres
    - $agent_string *(string)* - Le nouvel agent utilisateur

- Type de retour : void
- Résultat : Rien
- Note: Cette méthode est utile lorsque vous avez des usersagents enregistrés dans un fichier ou une base de données et que vous voulez récupérer des informations à partir d'eux. donc l'ignorer

Exemple :

```
$this->loadModel('user', 'donnees_utilisateurs');
$details = $this->donnees_utilisateurs->getDetails();
foreach($details As $detail) {
     $this->ua->setUserAgent($details['user_agent']);
     echo '<b>'.$details['id_user'].'</b> : ';
     echo $this->ua->browser().' -- '.$this->ua->version().'<br><br>';
}
```

- Note: On notera ici qu'on a au préalable récupéré l'agent utilisateur avec la méthode userAgent() et l'enregistré en base de données

- **isReferral**() : bool
  - Description: Verifie si l'utilisateur provenait d'un autre site.
  - Type de retour : bool
  - Résultat : true si le navigateur était sur un autre site avent d'arriver sur le vôtre et false sinon

- **referrer**() : ?string
  - Description: Renvoie l'adresse du site où provient l'utilisateur
  - Type de retour : string/null
  - Résultat : l'URL du site de provenance de l'utilisateur ou null s'il ne provenait pas d'un autre site

- **languages**() : array
  - Description: Renvoie la liste des langues acceptées par le navigateur de

l'utilisateur
    - Type de retour : array
    - Résultat : tableau contenant la liste des langues acceptées par le visiteur

- **accept_lang**([string $lang = 'en']) : bool
    - Description: Vérifie si une langue spécifique fait partir des langues acceptées par le navigateur de l'utilisateur
    - Paramètres
        - $lang *(string)* - Code de la langue à vérifier
    - Type de retour : bool
    - Résultat : true si la langue spécifiée fait partie des langues pris en compte par le navigateur du visiteur et false sinon

    Exemple :

```
if($this->ua->accept_lang('fr')) {
     $this->view('index')->render();
}
else {
     echo "Desolez notre site ne peut être accessible qu'en français";
     die();
}
```

- **charsets**() : **array**
    - Description: Renvoie la liste des encodages acceptés par le navigateur de l'utilisateur
    - Type de retour : array
    - Résultat : tableau contenant la liste des encodages acceptés par le visiteur

- **accept_charset**([string $charset = 'utf-8']) : bool
    - Description: Vérifie si un encodage spécifique fait partir des encodages acceptés par le navigateur de l'utilisateur
    - Paramètres

- $lang *(string)* - Encodage à vérifier

- Type de retour : bool
- Résultat : true si la l'encodage spécifié fait partie des encodages pris en compte par le navigateur du visiteur et false sinon

Validator

Nouveau chapitre