Micro-Framework

Le framework décrit ci-dessous est à finalité pédagogique : il permet uniquement de faciliter et d'accélérer le développement, en respectant le design pattern MVC.

Vous pouvez également consulter la documentation des classes déjà existantes (Micro-framework & Cloud) : Documentation API

-- Installation pour tests

- Télécharger ou cloner le projet https://github.com/jcheron/Cloud
- Copier les fichiers dans le dossier **htdocs** de votre serveur.
- Renommer éventuellement le dossier **Cloud-master** en **Cloud**

-- Configuration

1. Exécuter le script **database/cloud.sql** dans phpmyadmin pour créer la base de données
2. Editer le fichier de configuration app/config.php, et mettez éventuellement à jour les paramètres (siteUrl) :

```
<?php
return array(
        "siteUrl"=>"http://127.0.0.1/Cloud/",
        "documentRoot"=>"Accueil",
        "database"=>[
                "dbName"=>"cloud",
                "serverName"=>"127.0.0.1",
                "port"=>"3306",
                "user"=>"root",
                "password"=>""
        ],
        "directories"=>["my"]
);
```

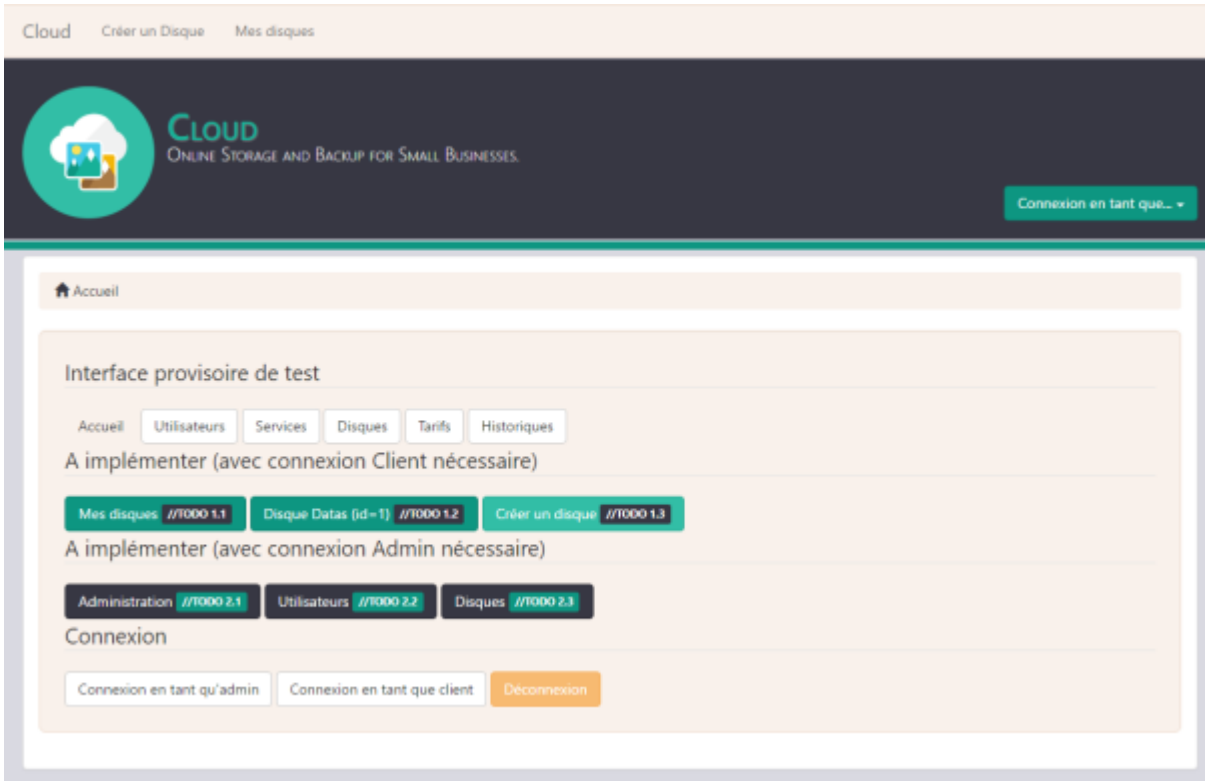
Vérifier également le paramètre **RewriteBase** du fichier **.htaccess** :

```
AddDefaultCharset UTF-8

<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /Cloud/
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{HTTP_ACCEPT} !(.*images.*)
    RewriteRule ^(.*)$ app/index.php?c=$1 [L]
</IfModule>
```

Le module **rewrite** doit être activé sur le serveur web apache.

Tester l'installation en allant à l'adresse : **http://127.0.0.1/Cloud/**



-- Structure

Elément	Emplacement	Rôle
Configuration	app/config.php	Fichier de configuration
Contrôleurs	app/controllers/	Définissent les URLs et la logique applicative
Vues	app/views/	Interfaces HTML/PHP
Modèles	app/models/	Classes métier
Divers	app/my	Classes personnelles

-- Modèles et mappage objet/relationnel

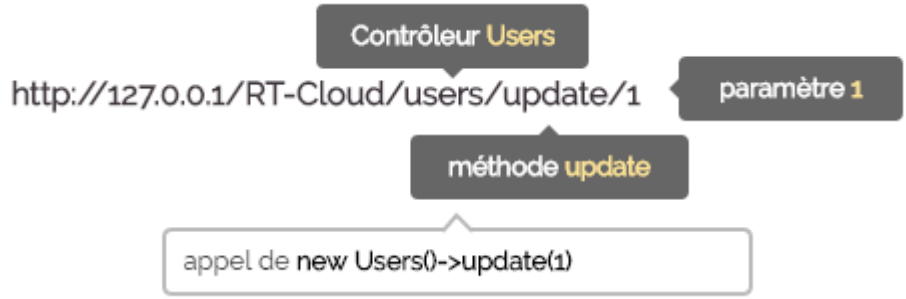
Les modèles sont les classes métiers correspondant aux tables de la base de données.
Chaque objet instancié correspond à un enregistrement de la table correspondante (table du même nom que la classe).
Chaque membre de données d'un objet correspond à un champ du même nom de la table correspondante.

Les modèles sont stockés dans le dossier **app/models**

-- Contrôleurs, vues

Un contrôleur est une classe héritant de **BaseController** et définie dans le dossier **app/controllers**.
Chaque contrôleur permet de définir un ensemble d'URL, en respectant le principe suivant :

Chaque méthode d'un contrôleur définit une ou plusieurs URL :



Url sollicitée	Appel réalisé
/controllerName/	Méthode **index** sur **ControllerName**
/controllerName/methodName/	Méthode **methodName** sur **ControllerName**
/controllerName/methodName/param1	Méthode **methodName** sur **ControllerName** avec passage du paramètre **param**

Les vues sont responsables de l'affichage des données (passées par le contrôleur), elles contiennent majoritairement du HTML (peu de PHP), et ne doivent pas effectuer de traitements.
Elles sont stockées dans le dossier **app/views**.

-- Chargement de données

Le rôle d'un contrôleur peut être de charger des données (depuis la BDD)

Exemple : chargement de tous les utilisateurs

```
class ExempleController extends BaseController{
    public function index(){
        $users=DAO::getAll("User");
        ...
    }
}
```

-- Affichage d'une ou plusieurs vues

Ou d'afficher des vues :

Exemple : Chargement de la vue **vHeader.php**

```
class ExempleController extends BaseController{
    public function index(){
       ...
       $this->loadView("main/vHeader");
    }
}
```

-- Passage de données à une vue

...D'afficher des vues en leur passant des données...

Passage d'une variable

```
class ExempleController extends BaseController{
    public function index(){
       $users=DAO::getAll("User");
       $this->loadView("main/vUsers",$users);
    }
}
```

Récupération d'une variable dans la vue

```
print_r($data);
```

Passage de plusieurs variables

```
class ExempleController extends BaseController{
    public function index(){
       $users=DAO::getAll("User");
       $this->loadView("main/vUsers",array("users"=>$users,"title"=>"Liste des
utilisateurs");
    }
}
```

Récupération de plusieurs variables dans la vue

Les clefs du tableau associatif passé correspondent aux variables accessibles depuis la vue :

```
echo "<h1>".$title."</h1>"
print_r($users);
```

-- Vues avec le moteur de template Twig

Le micro-framework peut utiliser le moteur de template Twig (son utilisation est définie dans le fichier config.php).

Il faut ensuite charger les vues en utilisant l'extension html, depuis le contrôleur.

-- Mise en place de contrôle d'accès

Pour restreindre l'accès aux URLs définies par un contrôleur :

Implémenter la méthode **isValid** du contrôleur :

On vérifie dans l'exemple suivant l'existence d'une variable de session **user**

```
class ExempleController extends BaseController{
    public function isValid(){
        return isset($_SESSION["user"]);
    }
}
```

Si la méthode **isValid** retourne false, la méthode **onInvalidControl** est automatiquement appelée :

```
class ExempleController extends BaseController{
    public function onInvalidControl(){
        echo "Accès interdit";
        exit;
    }
}
```

-- Accès aux données

-- Lecture de données

-- Chargement d'un enregistrement

Chargement du ticket d'id égal à 1 :

```
    $ticket=DAO::getOne("Ticket",1);
```

Les données uniques associées à un objet chargé depuis la base sont accessibles :

```
    //Utilisateur associé au ticket
    echo $ticket->getUser();
    //Catégorie associée au ticket
    echo $ticket->getCategorie();
```

Les données multiples associées à un objet chargé depuis la base doivent être explicitement chargées :

Chargement des messages liés au ticket :

```
    $messages=DAO::getOneToMany($ticket, "messages");
```

Chargement conditionnel d'un ticket :

```
$ticket=DAO::getOne("Ticket","title='Ecran bleu'");
```

-- Chargement de listes d'objets

Chargement de tous les tickets :

```
$tickets=DAO::getAll("Ticket");
```

La méthode **getAll** retourne un tableau qu'il est possible de parcourir :

```
$tickets=DAO::getAll("Ticket");
foreach($tickets as $ticket){
    echo $ticket->toString()."<br/>";
}
```

Chargement conditionnel des tickets de la catégorie d'id 2 :

```
$tickets=DAO::getAll("Ticket","idCategorie=2");
```

Chargement avec classement par ordre de date de creation :

```
$tickets=DAO::getAll("Ticket","1=1 ORDER BY dateCreation ASC");
```

Chargement des 5 premiers enregistrements :

```
$tickets=DAO::getAll("Ticket","1=1 LIMIT 5");
```

-- Mise à jour de données

-- Insertion

```
    $user=new User();
    $user->setLogin("jDoe");
    $user->setMail("jdoe@local.fr");
    $user->setPassword("wzrtb");
    DAO::insert($user);
```

Il est préférable de gérer l'impossibilité de l'ajout et les erreurs avec une gestion des exception (try...catch) :

```
    $user=new User();
    $user->setLogin("jDoe");
    $user->setMail("jdoe@local.fr");
    $user->setPassword("wzrtb");
    try{
        DAO::insert($user);
        echo "Utilisateur ajouté";
    }catch(Exception $e){
        echo "Erreur lors de l'ajout";
    }
```

-- Mise à jour

La mise à jour nécessite que l'objet à mettre à jour ait été chargé depuis la base de données :

```
    $user=DAO::getOne("User",5);
    $user->setLogin("johnDoe");
    try{
        DAO::update($user);
        echo "Utilisateur modifié";
    }catch(Exception $e){
        echo "Erreur lors de la modification";
    }
```

-- Suppression

La suppression nécessite que l'objet à supprimer ait été chargé depuis la base de données :

```
    $user=DAO::getOne("User",5);
    try{
        DAO::delete($user);
        echo "Utilisateur supprimé";
    }catch(Exception $e){
        echo "Erreur lors de la suppression";
    }
```

-- Exemples

Gestion des utilisateurs

Créer un contrôleur **CUsers** héritant de **_DefaultController** dans le dossier **app/controllers**

Implémenter le constructeur de la façon suivante :

- **$title** est le titre affiché sur la page
- **$model** la classe du modèle associé

```
<?php
class CUsers extends \_DefaultController {

    public function CUsers(){
        $this->title="Utilisateurs";
        $this->model="User";
    }

}
```

L'adresse http://127.0.0.1/helpdesk/cusers affiche maintenant la liste des utilisateurs :



Ajout/Modification

Créer la vue **app/views/cusers/vAdd.php** ; elle affiche un formulaire d'ajout ou de modification d'un utilisateur **$user** :

```
<form method="post" action="cusers/update">
<fieldset>
<legend>Ajouter/modifier un utilisateur</legend>
<div class="alert alert-info">Utilisateur : <?php echo $user->toString()?></div>
<div class="form-group">
    <input type="hidden" name="id" value="<?php echo $user->getId()?>">
    <input type="mail" name="mail" value="<?php echo $user->getMail()?>"
placeholder="Entrez l'adresse email" class="form-control">
    <input type="text" name="login" value="<?php echo $user->getLogin()?>"
placeholder="Entrez un login" class="form-control">
    <input type="password" name="password" value="<?php echo
```

```
$user->getPassword()?>" placeholder="Entrez le mot de passe" class="form-control">
    <div class="checkbox">
        <label><input type="checkbox" name="admin" <?php echo
($user->getAdmin()?"checked":"")?> value="1">Administrateur ?</label>
    </div>
</div>
<div class="form-group">
    <input type="submit" value="Valider" class="btn btn-default">
    <a class="btn btn-default" href="<?php echo
$config["siteUrl"]?>cusers">Annuler</a>
</div>
</fieldset>
</form>
```

Cette vue sera appelée sur l'action **frm** du contrôleur **CUsers** ; la méthode **frm** doit donc initialiser l'instance **$user**, puis ensuite charger la vue **vAdd.php** :

```
    public function frm($id=NULL){
        $user=$this->getInstance($id);
        $this->loadView("cusers/vAdd",array("user"=>$user));
    }
```

La méthode **getInstance** retourne l'utilisateur chargé depuis la base si **$id** est renseigné, ou un nouvel utilisateur dans le cas contraire.

L'ajout et la modification doivent maintenant fonctionner, excepté pour le champ **admin**, de type booléen, et défini par une case à cocher.
Il faut dans ce cas sur-définir la méthode **setValuesToObject** de la classe de base **_DefaultController**, pour faire en sorte que **admin** ne soit vrai que si la case **admin** du formulaire est cochée

```
    protected function setValuesToObject(&$object) {
        parent::setValuesToObject($object);
        $object->setAdmin(isset($_POST["admin"]));
    }
```

-- JavaScript/Jquery

L'introduction de scripts Jquery se fait dans les contrôleurs, par l'intermédiaire des méthodes de la classe Jquery

-- Requête ajax get vers une Url

Exécution directe :

Appel de l'url **users/frm** dont le résultat est affiché dans la zone html d'id **response**

```
    echo Jquery::get("users/frm","#response");
```

Exécution sur évènement :

Appel de l'url **users/frm** dont le résultat est affiché dans la zone html d'id **response** sur **click** du bouton d'id **btAfficher**

```
    echo Jquery::getOn("#btAfficher","click","users/frm","#response");
```

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/slam4/micro-framework?rev=1474824656

Last update: **2019/08/31 14:38**