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.

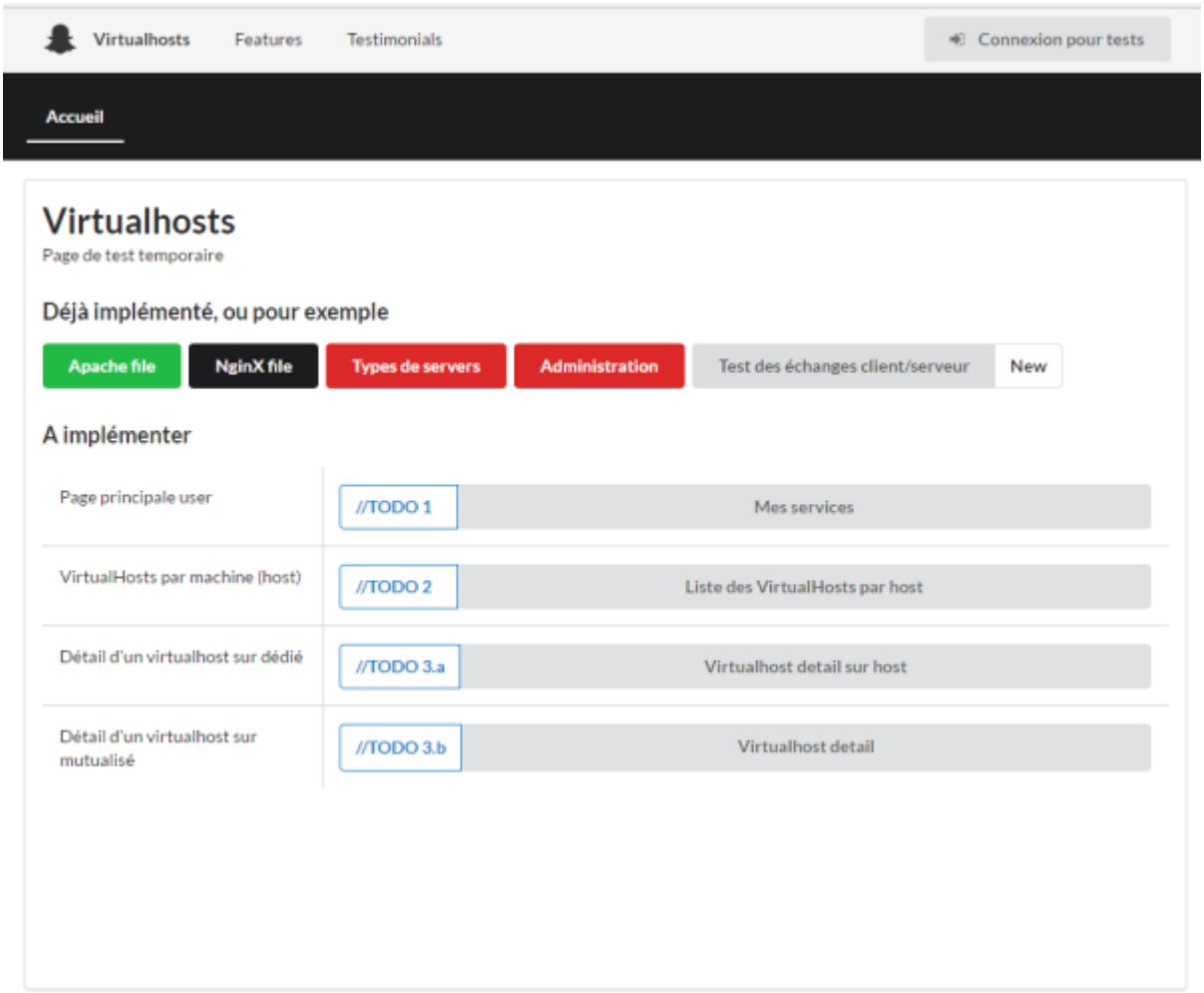
Vous pouvez également consulter la documentation des classes déjà existantes (Micro-framework & Virtualhosts) :
Documentation API

-- Installation pour tests

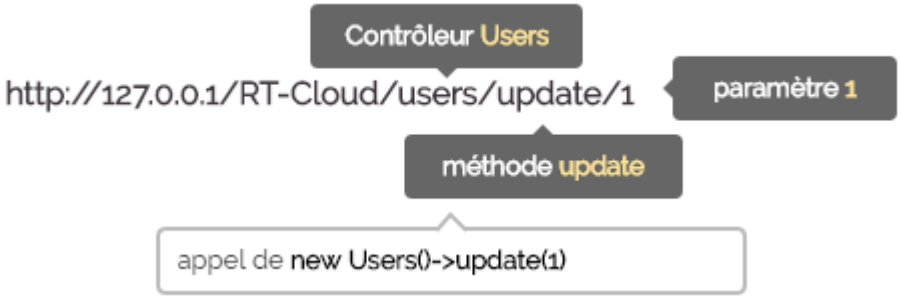
- Télécharger ou cloner le projet https://github.com/jcheron/micro-virtualhosts
- Copier les fichiers dans le dossier **htdocs** de votre serveur.
- Renommer éventuellement le dossier **micro-virtualhosts-master** en **micro-virtualhosts**

-- Configuration

1. Exécuter le script **app/database/virtualhosts.sql** dans phpmyadmin pour créer la base de données,
2. Editer le fichier de configuration app/config.php, et mettez éventuellement à jour le paramètre **siteUrl** en fonction du dossier où se trouve votre projet,
3. Editer le fichier de **.htaccess, et mettez éventuellement à jour le paramètre** RewriteBase **de la même façon. <sxh bash>composer update<sxh> <sxh php;title:app/config.php> <?php return array(“siteUrl”⇒“http://127.0.0.1/micro-virtualhosts/”, “documentRoot”⇒“Test”, “database”⇒[“dbName”⇒“virtualhosts”, “serverName”⇒“127.0.0.1”, “port”⇒“3306”, “user”⇒“root”, “password”⇒“”], “onStartup”⇒function($action){ }, “directories”⇒[“libraries”], “templateEngine”⇒'micro\views\engine\Twig', “templateEngineOptions”⇒array(“cache”⇒false), “test”⇒false, “debug”⇒false, “di”⇒[“jquery”⇒function(){ $jquery=new Ajax\php\micro\JsUtils([“defer”⇒true]); $jquery→semantic(new Ajax\Semantic()); return $jquery; }]); </sxh> Vérifier également le paramètre** RewriteBase **du fichier** .htaccess **: <sxh php;title:.htaccess;highlight:[5]> AddDefaultCharset UTF-8 <IfModule mod_rewrite.c> RewriteEngine On RewriteBase /micro-virtualhosts/ RewriteCond %{REQUEST_FILENAME} !-f RewriteCond %{HTTP_ACCEPT} !(.*images.*) RewriteRule ^(.*)$ app/index.php?c=$1 [L] </IfModule> </sxh>**
   Le module **rewrite** doit être activé sur le serveur web apache.
   Tester l'installation en allant à l'adresse : http://127.0.0.1/micro-virtualhosts/



===== - Structure ===== |< 100% >| ^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/libraries | 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 :



|< 100% >| ^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 <sxh php;title:app/controllers/ExempleController> class ExempleController extends BaseController{ public function index(){ $users=DAO::getAll("User"); … } } </sxh> ==== - Affichage d'une ou plusieurs vues ==== Ou d'afficher des vues : Exemple : Chargement de la vue** vHeader.php **<sxh php;title:app/controllers/ExempleController;highlight:[4]> class ExempleController extends BaseController{ public function index(){ … $this→loadView("main/vHeader"); } } </sxh> ==== - Passage de données à une vue ==== …D'afficher des vues en leur passant des données… === Passage d'une variable === <sxh php;title:app/controllers/ExempleController;highlight:[4]> class ExempleController extends BaseController{ public function index(){ $users=DAO::getAll("User"); $this→loadView("main/vUsers",$users); } } </sxh> === Récupération d'une variable dans la vue === <sxh php;title:app/views/vUsers.php> print_r($data); </sxh> === Passage de plusieurs variables === <sxh php;title:app/controllers/ExempleController;highlight:[4]> class ExempleController extends BaseController{ public function index(){ $users=DAO::getAll("User"); $this→loadView("main/vUsers",array("users"⇒$users,"title"⇒"Liste des utilisateurs"); } } </sxh> === Récupération de plusieurs variables dans la vue === Les clefs du tableau associatif passé correspondent aux variables accessibles depuis la vue : <sxh php;title:app/views/vUsers.php> echo "<h1>".$title."</h1>" print_r($users); </sxh> ===== - 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 **<sxh php;title:app/controllers/ExempleController;highlight:[4]> class ExempleController extends BaseController{ public function isValid(){ return isset($_SESSION["user"]); } } </sxh> Si la méthode** isValid **retourne false, la méthode** onInvalidControl **est automatiquement appelée : <sxh php;title:app/controllers/ExempleController;highlight:[4]> class ExempleController extends BaseController{ public function onInvalidControl(){ echo "Accès interdit"; exit; } } </sxh> ===== - Accès aux données ===== ==== - Lecture de données ==== === - Chargement d'un enregistrement === Chargement de l'Host d'id égal à 1 : <sxh php;title:app/controllers/ExempleController> use micro\orm\DAO; $host=DAO::getOne("Host",1); </sxh> Les données uniques associées à un objet chargé depuis la base sont accessibles : <sxh php;title:app/controllers/ExempleController>** ***Utilisateur associé à l'Host echo $host→getUser(); </sxh> Les données multiples associées à un objet chargé depuis la base doivent être explicitement chargées : Chargement des servers installés sur un Host : <sxh php;title:app/controllers/ExempleController> $servers=DAO::getOneToMany($host, "servers"); </sxh> Chargement conditionnel d'un Host : <sxh php;title:app/controllers/ExempleController> $ticket=DAO::getOne("Host","name='srv1'"); </sxh> === - Chargement de listes d'objets === Chargement de tous les hosts : <sxh php;title:app/controllers/ExempleController> $hosts=DAO::getAll("Host"); </sxh> La méthode getAll retourne un tableau qu'il est possible de parcourir : <sxh php;title:app/controllers/ExempleController> $hosts=DAO::getAll("User"); foreach($hosts as $host){ echo $host."
"; } </sxh> Chargement conditionnel des hosts d'un utilisateur d'id 2 : <sxh php;title:app/controllers/ExempleController> $hosts=DAO::getAll("Ticket","idUser=2"); </sxh> Chargement avec classement par ordre du nom : <sxh php;title:app/controllers/ExempleController> $hosts=DAO::getAll("Host","1=1 ORDER BY name ASC"); </sxh> Chargement des 5 premiers enregistrements : <sxh php;title:app/controllers/ExempleController> $hosts=DAO::getAll("Host","1=1 LIMIT 5"); </sxh> ==== - Mise à jour de données ==== === - Insertion === <sxh***

php;title:app/controllers/ExempleController;highlight:[6]> $user=new User(); $user→setLogin(“jDoe”); $user→setMail(“jdoe@local.fr”); $user→setPassword(“wzrtb”); DAO::insert($user); </sxh> Il est préférable de gérer l'impossibilité de l'ajout et les erreurs avec une gestion des exception (try…catch) : <sxh php;title:app/controllers/ExempleController;highlight:[6]> $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”; } </sxh> === - Mise à jour === La mise à jour nécessite que l'objet à mettre à jour ait été chargé depuis la base de données : <sxh php;title:app/controllers/ExempleController;highlight:[4]> $user=DAO::getOne(“User”,5); $user→setLogin(“johnDoe”); try{ DAO::update($user); echo “Utilisateur modifié”; }catch(Exception $e){ echo “Erreur lors de la modification”; } </sxh> === - Suppression === La suppression nécessite que l'objet à supprimer ait été chargé depuis la base de données : <sxh php;title:app/controllers/ExempleController;highlight:[3]> $user=DAO::getOne(“User”,5); try{ DAO::delete($user); echo “Utilisateur supprimé”; }catch(Exception $e){ echo “Erreur lors de la suppression”; } </sxh>