ubiquity-framework Documentation

phpmv

juin 03, 2023

Quick-starts

CHAPITRE 1

Quick start avec la console

Note : Si vous n'aimez pas le mode console, vous pouvez passer au quick-start avec les *webtools (UbiquityMyAdmin)*.

1.1 Composer installation

ubiquity utilise Composer pour gérer ses dépendances. Vous devrez donc vous assurer que vous avez Composer installé sur votre machine.

1.2 Installation d'Ubiquity devtools

Télécharger l'installeur Ubiquity-devtools avec Composer.

```
composer global require phpmv/ubiquity-devtools
```

Testez votre installation en faisant :

```
Ubiquity version
```

```
· PHP 7.2.15-0ubuntu0.18.04.1
· Ubiquity devtools (1.1.3)
```

Vous pouvez obtenir à tout moment de l'aide sur une commande en tapant : `Ubiquity help` suivi de ce que vous cherchez.

Exemple :

```
Ubiquity help project
```

1.3 Création de projet

Créer le projet **quick-start**

```
Ubiquity new quick-start
```

1.4 Structure des dossiers

Le projet créé dans le dossier **quick-start** a une structure simple et lisible :

Le dossier **app** contient le code de votre future application :

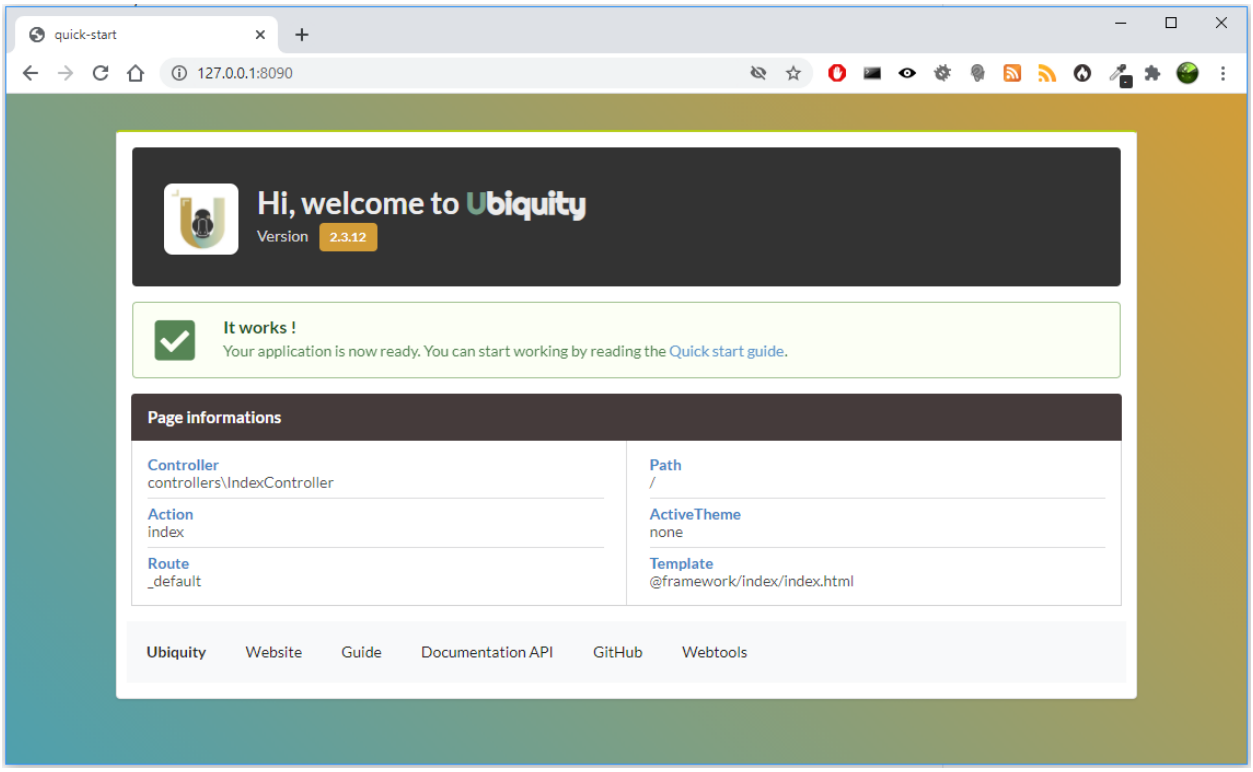
```
app
  ├ cache
  ├ config
  ├ controllers
  ├ models
  └ views
```

1.5 Démarrage

Allez dans le dossier nouvellement créé **quick-start** et démarrez le serveur php intégré :

```
Ubiquity serve
```

Vérifier l'opération à l'adresse **http ://127.0.0.1 :8090** :



Note : Si le port 8090 est occupé, vous pouvez démarrer le serveur en utilisant un autre port en utilisant l'option -p.

```
Ubiquity serve -p=8095
```

1.6 Contrôleur

L'application en mode console **devtools** permet de gagner du temps dans les opérations répétitives. Utilisons la pour créer un contrôleur.

```
Ubiquity controller DefaultController
```

```
   · The project folder is /var/www/html/quick-start

■ success : Controller creation
   · Creation of the Controller DefaultController at the location app/controllers/DefaultController.php
```

Vous pouvez éditer le fichier `app/controllers/DefaultController` dans votre IDE :

Code source 1 – app/controllers/DefaultController.php

```
1  namespace controllers;
2   /**
3    * Controller DefaultController
4    */
5  class DefaultController extends ControllerBase{
6       public function index(){}
7  }
```

Ajouter le message traditionnel, et tester votre page à l'adresse `http://127.0.0.1:8090/DefaultController`

Code source 2 – app/controllers/DefaultController.php

```
class DefaultController extends ControllerBase{

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

}
```

Pour l'instant, nous n'avons pas défini de routes, L'accès à l'application se fait donc selon le schéma suivant : `controllerName/actionName/param`

L'action par défaut est la méthode **index**, elle n'a pas besoin d'être spécifiée dans l'url.

1.7 Route

Important : Le routage est défini avec l'attribut `Route` (avec php>8) ou l'annotation `@route` et n'est pas fait dans un fichier de configuration : c'est un choix de conception.

Le paramètre **automated** mis à **true** permet aux méthodes de notre classe d'être définies comme des sous routes de la route principale `/hello`.

Avec annotations :

Code source 3 – app/controllers/DefaultController.php

```
namespace controllers;
/**
 * Controller DefaultController
 * @route("/hello","automated"=>true)
 */
class DefaultController extends ControllerBase{

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

}
```

Avec attributs (php>8) :

Code source 4 – app/controllers/DefaultController.php

```
namespace controllers;
use Ubiquity\attributes\items\router\Route;

#[Route('/hello', automated: true)]
class DefaultController extends ControllerBase{

    public function index(){
```

(suite sur la page suivante)

(suite de la page précédente)

```
8           echo 'Hello world!';
9       }
10
11  }
```

1.7.1 Cache du routeur

Important : Aucune modification sur les routes n'est effective sans initialiser le cache. Les annotations ne sont jamais lues au moment de l'exécution. C'est également un choix de conception.

Nous pouvons utiliser la console pour ré-initialiser le cache :

```
Ubiquity init-cache
```

```
■ success : init-cache:all
   · cache directory is /var/www/html/quick-start/app/cache/
   · Models directory is /var/www/html/quick-start/app/models
   · Models cache reset
   · Controllers directory is /var/www/html/quick-start/app/controllers
   · Router cache reset
   · Controllers directory is /var/www/html/quick-start/app/controllers
   · Rest cache reset
```

Vérifions que la route existe :

```
Ubiquity info:routes
```

path	controller	action	parameters
/hello/(index/)?	controllers\DefaultController	index	[]

```
   · 1 routes (routes)
```

Nous pouvons maintenant tester la page `http://127.0.0.1:8090/hello`

1.8 Action et route avec paramètres

Nous allons maintenant créer une action (sayHello) avec un paramètre (name), et la route associée (to) : La route utilisera le paramètre **name** de l'action :

```
Ubiquity action DefaultController.sayHello -p=name -r=to/{name}/
```

```
■ info
   · You need to re-init Router cache to apply this update with init-cache command

■ info : Creation
   · The action sayHello is created in controller controllers\DefaultController
```

Après ré-initialisation du cache (commande **init-cache**), la commande **info :routes** devrait afficher :

path	controller	action	parameters
/hello/(index/)?	controllers\DefaultController	index	[]
/hello/to/(.+?)/		sayHello	[name*]

```
   · 2 routes (routes)
```

Changer le code dans votre IDE : l'action doit dire hello à quelqu'un (somebody)...

Code source 5 – app/controllers/DefaultController.php

```php
    /**
     * @route("to/{name}/")
     */
    public function sayHello($name){
            echo 'Hello '.$name.'!';
    }
```

et tester la page `http://127.0.0.1:8090/hello/to/Mr SMITH`

1.9 Action, route, paramètres et vue

Nous allons maintenant créer une action (information) avec deux paramètres (titre et message), la route associée (info), et une vue pour afficher le message : La route utilisera les deux paramètres de l'action.

```
Ubiquity action DefaultController.information -p=title,message='nothing' -r=info/{title}/
↪{message} -v
```

Note : Le paramètre -v (–view) est utiliser pour créer la vue associée à l'action.

Après ré-initialisation du cache, nous devrions avoir 3 routes :

path	controller	action	parameters
/hello/(index/)?	controllers\DefaultController	index	[]
/hello/to/(.+?)/		sayHello	[name*]
/hello/info/(.+?)/(.*?)		information	[title*,message]

· 3 routes (routes)

Retournons à notre environnement de développement pour voir le code généré :

Code source 6 – app/controllers/DefaultController.php

```
    /**
     * @route("info/{title}/{message}")
     */
    public function information($title,$message='nothing'){
            $this->loadView('DefaultController/information.html');
    }
```

Nous devons passer les 2 variables à la vue :

```
/**
 * @route("info/{title}/{message}")
 */
public function information($title,$message='nothing'){
        $this->loadView('DefaultController/information.html',compact('title','message'));
}
```

Et utiliser ces 2 variables dans la vue twig associée :

Code source 7 – app/views/DefaultController/information.html

```
    <h1>{{title}}</h1>
    <div>{{message | raw}}</div>
```

Nous pouvons tester votre page à l'addresse `http://127.0.0.1:8090/hello/info/Quick start/Ubiquity est super simple` une évidence.

CHAPITRE 2

Quick start avec les Webtools

2.1 Installation de Composer

ubiquity utilise Composer pour gérer ses dépendances. Vous devrez donc vous assurer que vous avez Composer installé sur votre machine.

2.2 Installation d'Ubiquity-devtools

Télécharger l'installeur Ubiquity-devtools avec Composer.

```
composer global require phpmv/ubiquity-devtools
```

Testez votre installation en faisant :

```
Ubiquity version
```

```
· PHP 7.2.15-0ubuntu0.18.04.1
· Ubiquity devtools (1.1.3)
```

Vous pouvez obtenir à tout moment de l'aide sur une commande en tapant : `Ubiquity help` suivi de ce que vous cherchez.

Exemple :

```
Ubiquity help project
```

2.3 Création de projet

Créer le projet **quick-start** incluant les **Webtools** (avec l'option **-a**)

```
Ubiquity new quick-start -a
```

2.4 Structure des dossiers

Le projet créé dans le dossier **quick-start** a une structure simple et lisible :

Le dossier **app** contient le code de votre future application :

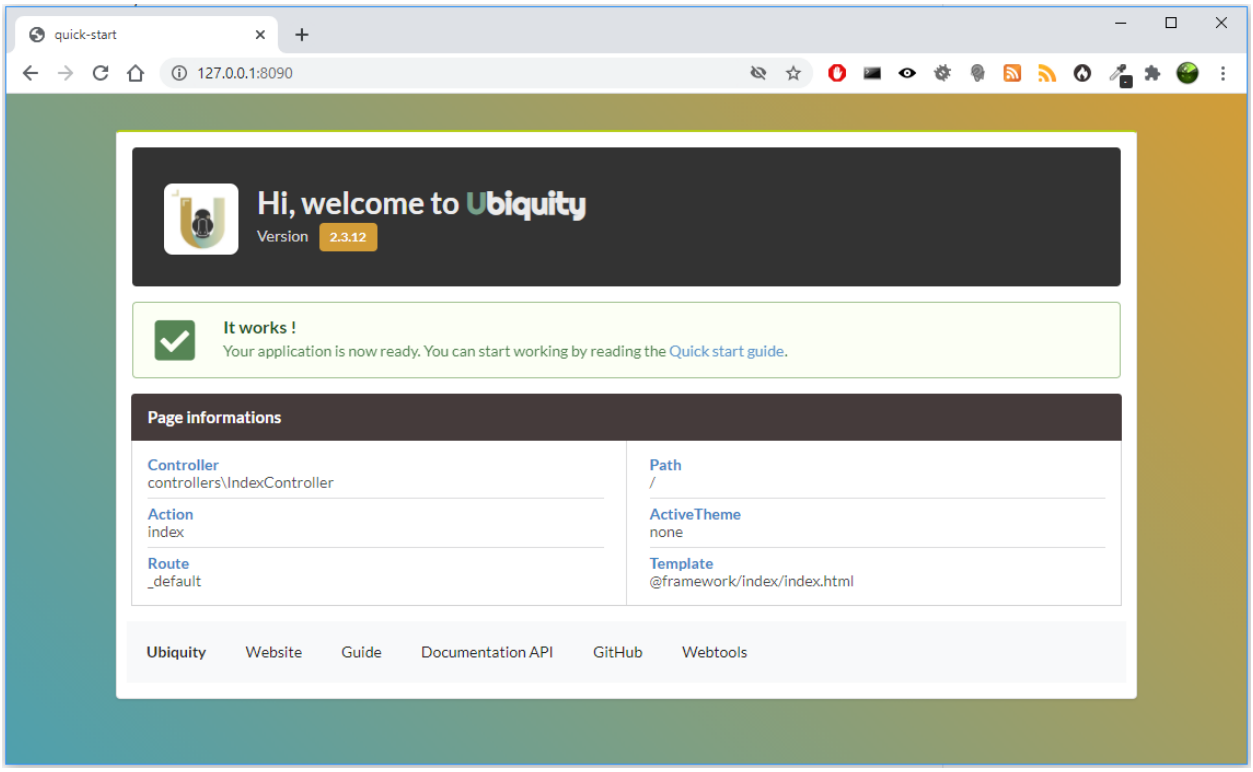
```
app
  ├ cache
  ├ config
  ├ controllers
  ├ models
  └ views
```

2.5 Démarrage

Allez dans le dossier nouvellement créé **quick-start** et démarrez le serveur php intégré :

```
Ubiquity serve
```

Vérifier l'opération à l'adresse **http ://127.0.0.1 :8090** :

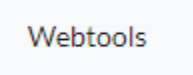


Note : Si le port 8090 est occupé, vous pouvez démarrer le serveur en utilisant un autre port en utilisant l'option -p.

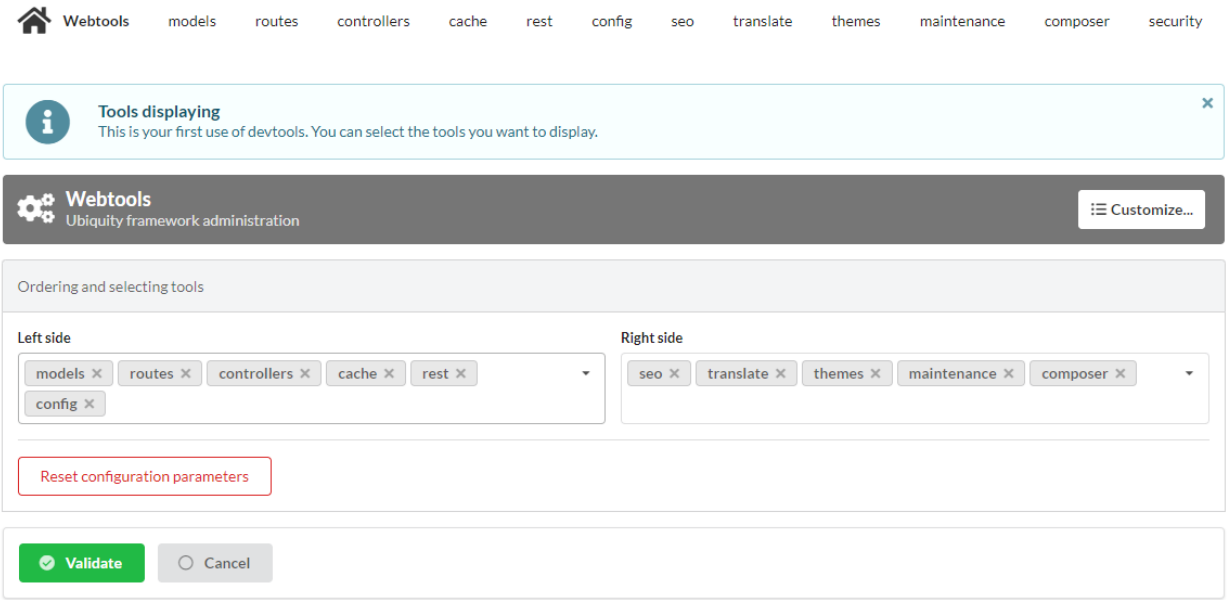
```
Ubiquity serve -p=8095
```

2.6 Contrôleur

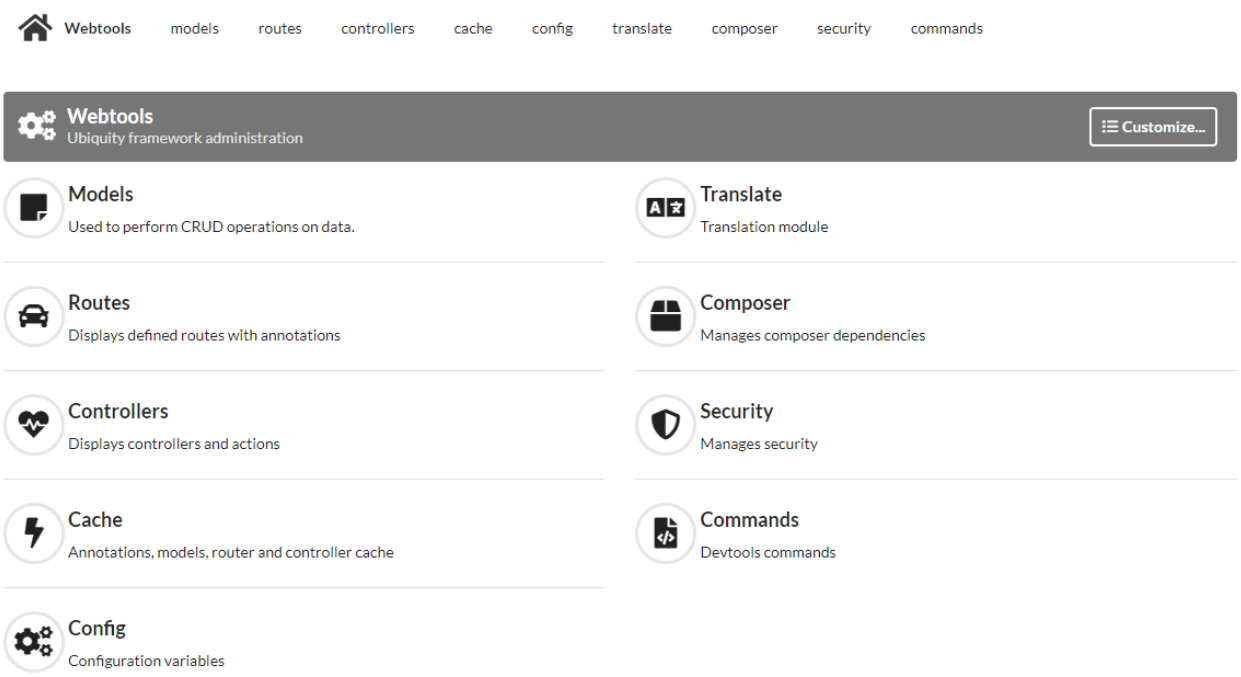
Naviguer vers l'interface d'administration en cliquant sur le bouton **Webtools** :



Sélectionner les outils dont vous avez besoin :



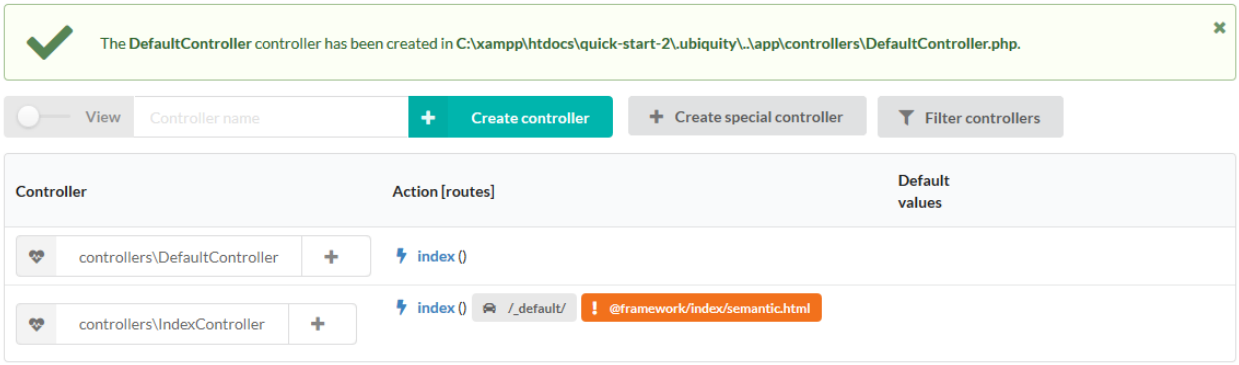
L'application web **Webtools** permet de gagner du temps sur les opérations répétitives.



Nous allons les utiliser pour créer un contrôleur.

Aller dans la partie **Controllers**, saisir **DefaultController** dans la zone **controllerName** et créer le contrôleur :

Le contrôleur **DefaultController** est créé :



Vous pouvez éditer le fichier `app/controllers/DefaultController` dans votre IDE :

Code source 1 – app/controllers/DefaultController.php

```
1 namespace controllers;
2  /**
3  * Controller DefaultController
4  **/
5 class DefaultController extends ControllerBase{
6      public function index(){}
7 }
```

Ajouter le traditionnel **Hello world**, et tester votre page à l'adresse `http://127.0.0.1:8090/DefaultController`

Code source 2 – app/controllers/DefaultController.php

```
class DefaultController extends ControllerBase{

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

}
```

Pour l'instant, nous n'avons pas défini de routes, L'accès à l'application se fait donc selon le schéma suivant : `controllerName/actionName/param`

L'action par défaut est la méthode **index**, elle n'a pas besoin d'être spécifiée dans l'url.

2.7 Route

Important : Le routage est défini avec l'attribut `Route` (avec php>8) ou l'annotation `@route` et n'est pas fait dans un fichier de configuration : c'est un choix de conception.

Le paramètre **automated** mis à **true** permet aux méthodes de notre classe d'être définies comme des sous routes de la route principale `/hello`.

Code source 3 – app/controllers/DefaultController.php

```
1  namespace controllers;
2   /**
3   * Controller DefaultController
4   * @route("/hello","automated"=>true)
5   **/
6  class DefaultController extends ControllerBase{
7
8          public function index(){
9                  echo 'Hello world!';
10         }
11
12 }
```

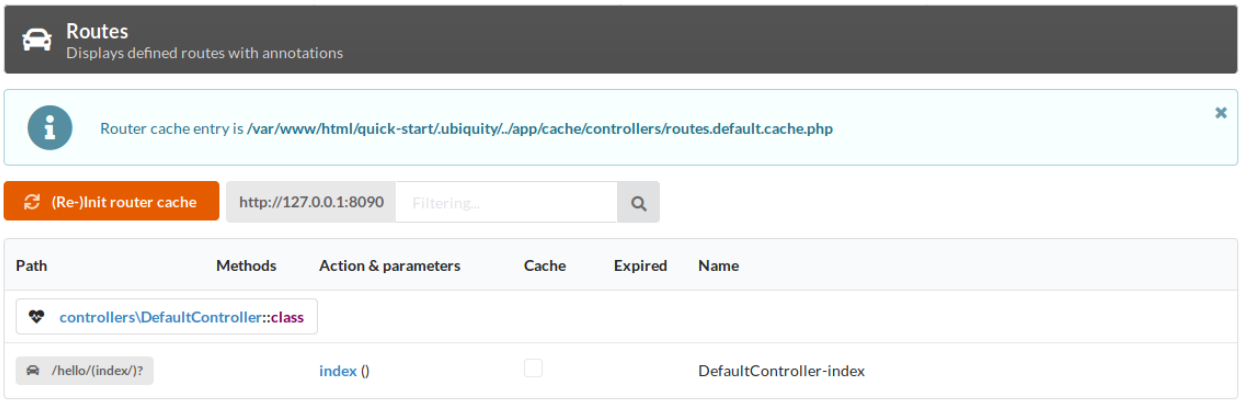
2.7.1 Cache du routeur

Important : Aucune modification sur les routes n'est effective sans initialiser le cache. Les annotations ne sont jamais lues au moment de l'exécution. C'est également un choix de conception.

Nous pouvons utiliser les webtools pour ré-initialiser le cache :

Allez dans la partie **Routes** et cliquer sur le bouton **re-init cache**

La route est maintenant listée dans l'interface :



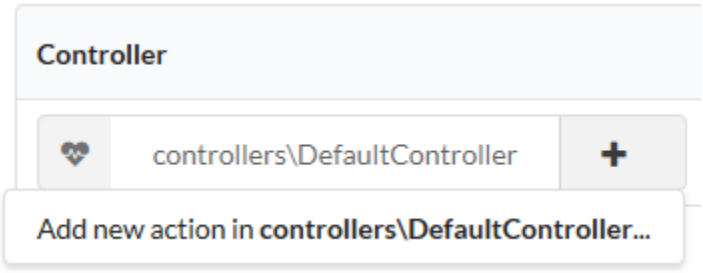
Nous pouvons maintenant tester la page en cliquant sur le bouton **GET** ou en allant à l'adresse `http://127.0.0.1:8090/hello`.

2.8 Action & route avec paramètres

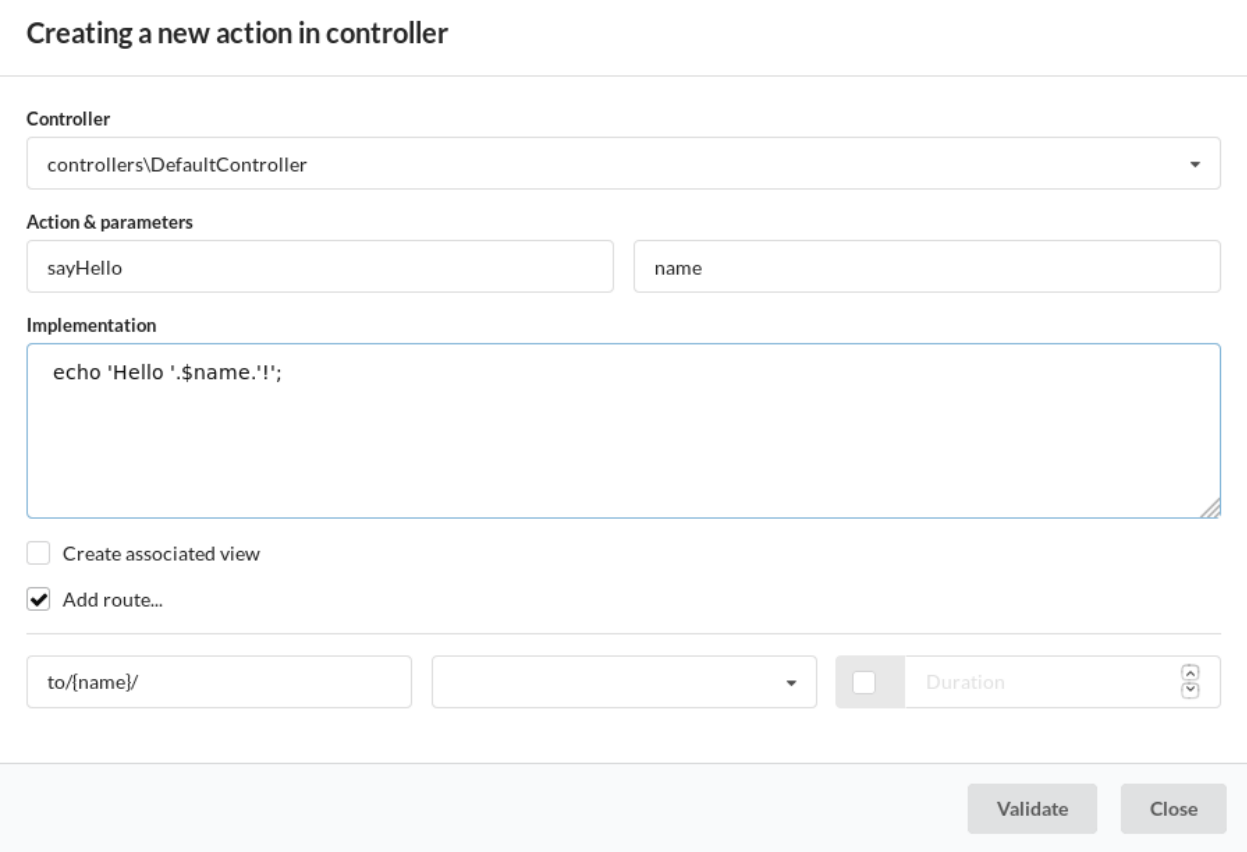
Nous allons maintenant créer une action (sayHello) avec un paramètre (name), et la route associée (to) : La route utilisera le paramètre **name** de l'action :

Aller à la section **Controllers** :

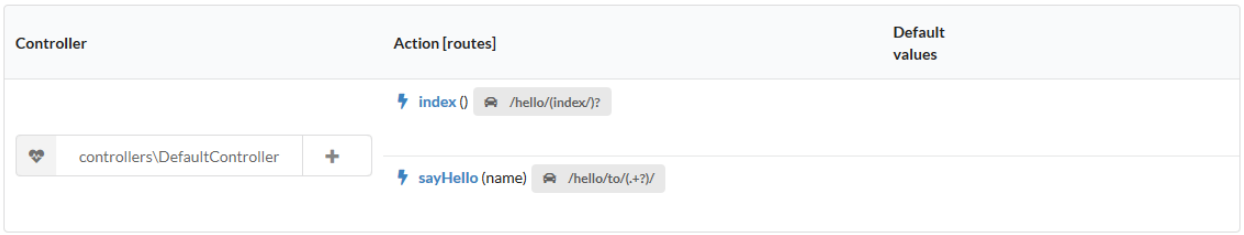
- cliquer sur le bouton + associé à DefaultController,
- puis sélectionner l'élément **Add new action in..**,



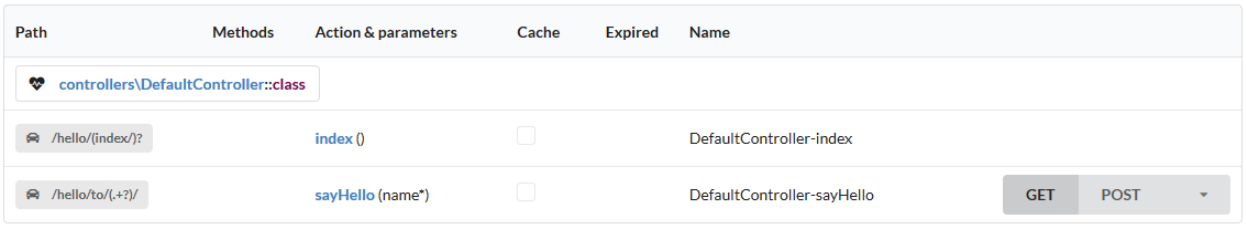
Saisir les caractéristiques de l'action dans le formulaire suivant :



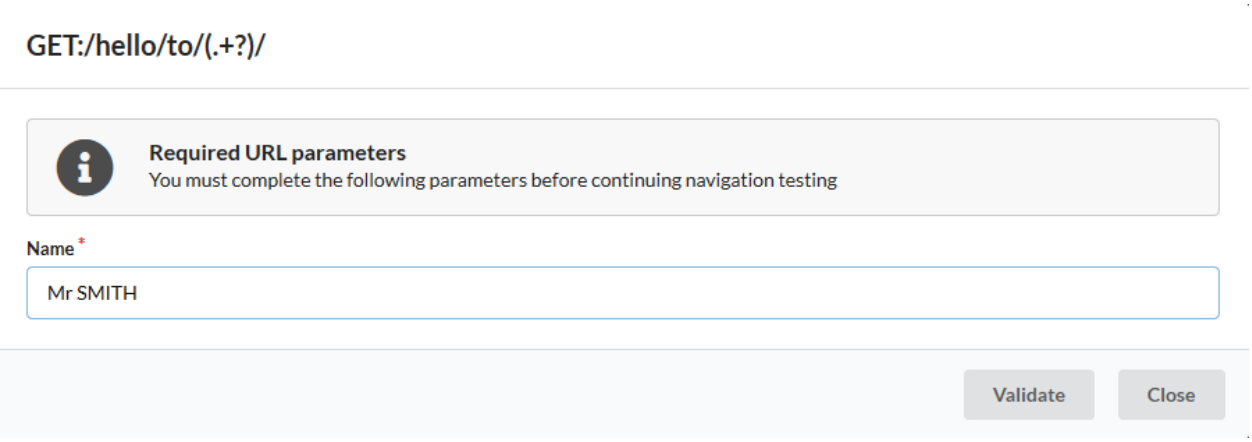
Après avoir réinitialisé le cache avec le bouton orange, nous pouvons voir la nouvelle route **hello/to/{name}** :



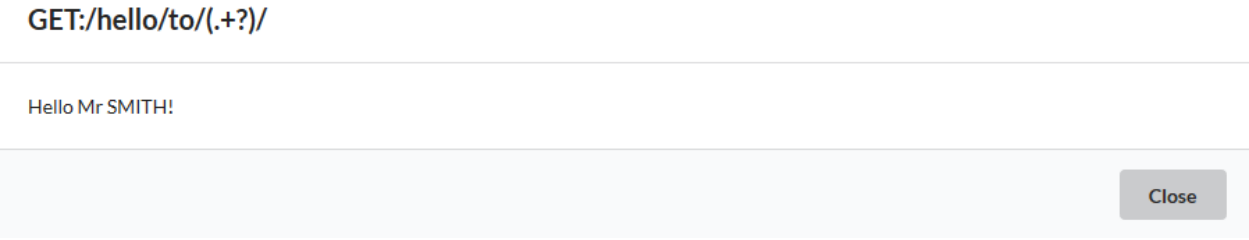
Vérifiez la création de la route en allant dans la section Routes :



Nous pouvons maintenant tester la page en cliquant sur le bouton **GET** :



Nous pouvons voir le résultat :

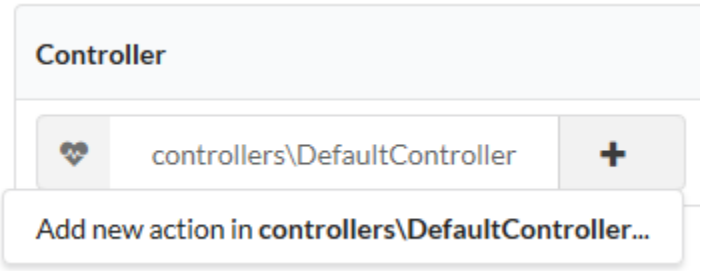


Aller directement à l'adresse « http://127.0.0.1:8090/hello/to/Mr SMITH » pour tester

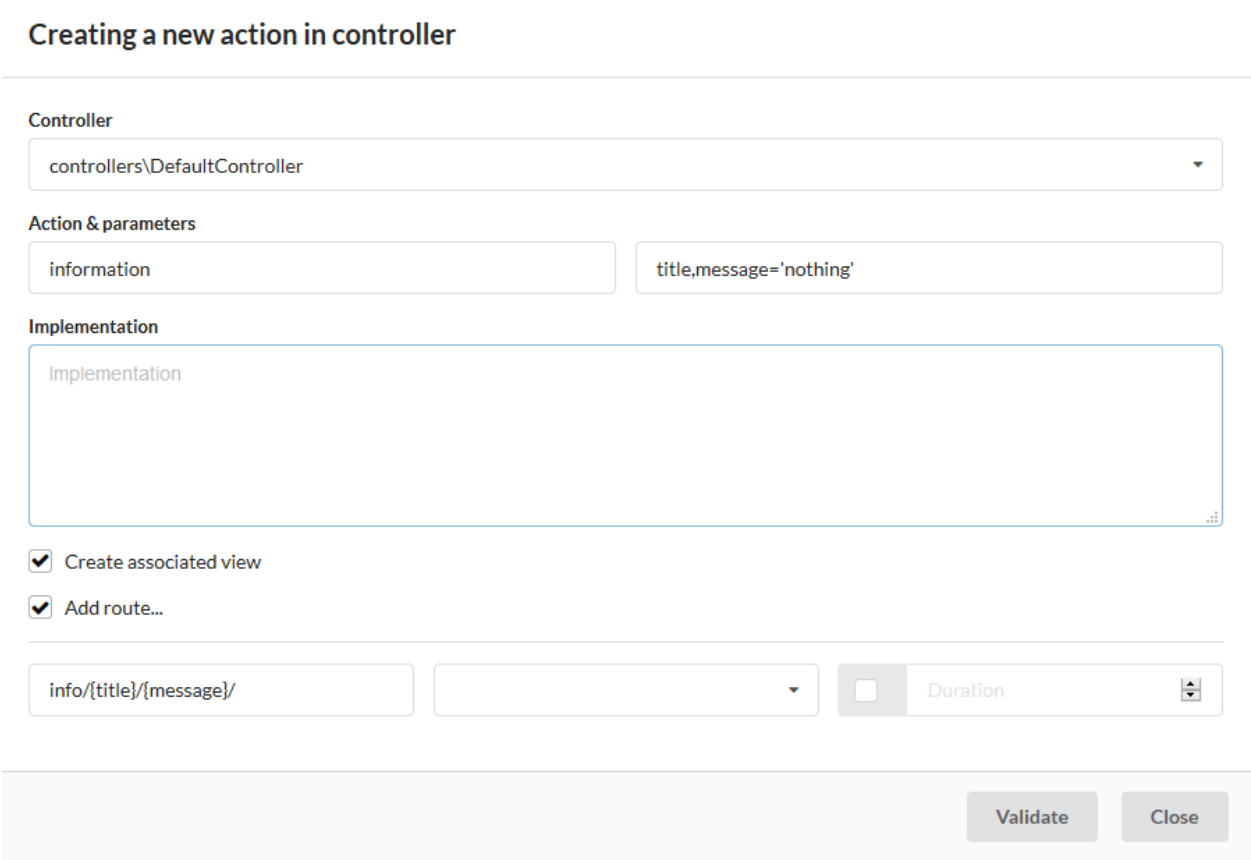
2.9 Action, paramètres de route & vue

Nous allons maintenant créer une action (information) avec deux paramètres (titre et message), la route associée (info), et une vue pour afficher le message : La route utilisera les deux paramètres de l'action.

Dans la section **Contrôleurs**, créer une autre action dans **DefaultController** :

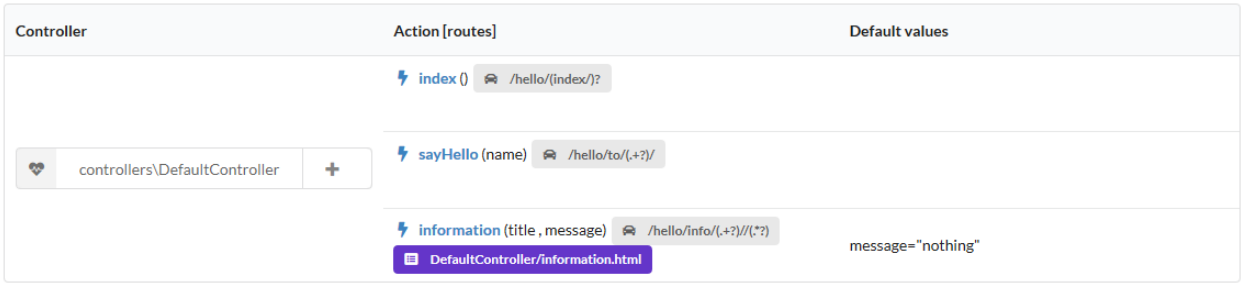


Saisir les caractéristiques de l'action dans le formulaire suivant :



Note : La case à cocher **view** d'activer la création de la vue associée à l'action.

Après avoir réinitialisé le cache, nous avons maintenant 3 routes :



Retournons dans notre environnement de développement et voyons le code généré :

Code source 4 – app/controllers/DefaultController.php

```
    /**
     *@route("info/{title}/{message}")
    **/
    public function information($title,$message='nothing'){
            $this->loadView('DefaultController/information.html');
    }
```

Nous devons passer les 2 variables à la vue :

```php
/**
 *@route("info/{title}/{message}")
**/
public function information($title,$message='nothing'){
        $this->loadView('DefaultController/information.html',compact('title','message'));
}
```

Et nous utilisons nos 2 variables dans la vue twig associée :

Code source 5 – app/views/DefaultController/information.html

```twig
    <h1>{{title}}</h1>
    <div>{{message | raw}}</div>
```

Nous pouvons tester notre page à l'adresse `http://127.0.0.1:8090/hello/info/Quick start/Ubiquity is quiet simple`;-)

New in documentation

- *Security*
- *Async platforms*
- *Commands module*
- *Composer module*
- *OAuth client module*
- Mailer module
- *Servers configuration*
- *Database connection*
- *Optimization*
- *Rich client*
- *REST module*
- *Data transformers*
- *Dependency injection*
- *Events*
- *Views and themes*
- *Contributing*
- *Quick start avec les webtools*
- Generating models :
  - with webtools (UbiquityMyAdmin)
  - with console (devtools)

CHAPITRE 3

Installation d'ubiquity-devtools

3.1 Composer installation

ubiquity utilise Composer pour gérer ses dépendances. Vous devrez donc vous assurer que vous avez Composer installé sur votre machine.

3.2 Installer Ubiquity-devtools

Téléchargez l'installateur Ubiquity-devtools en utilisant Composer.

```
composer global require phpmv/ubiquity-devtools
```

Assurez-vous de placer le répertoire `~/.composer/vendor/bin` dans votre PATH afin que l'exécutable **Ubiquity** puisse être localisé par votre système.

Une fois installé, la commande `Ubiquity new` crée une nouvelle installation d'Ubiquity dans le répertoire que vous avez spécifié. Par exemple, `Ubiquity new blog` crée un répertoire nommé **blog** contenant un projet Ubiquity :

```
Ubiquity new blog
```

L'option semantic ajoute Semantic-UI pour le front-end.

Vous pouvez voir plus d'options sur l'installation en lisant la section *Création de projet*.

CHAPITRE 4

Création de projet

Après avoir installé *Installation d'ubiquity-devtools*, dans votre terminal, utilisez la commande *new* depuis le dossier racine de votre serveur web :

4.1 Exemples

Un simple projet

```
Ubiquity new projectName
```

Un projet avec les webtools

```
Ubiquity new projectName -a
```

Un projet avec les thèmes Bootstrap et Semantic-ui installés

```
Ubiquity new projectName --themes bootstrap,semantic
```

4.2 Arguments de l'installeur

nom court	nom	rôle	valeur par défaut	valeurs autorisées	Depuis les devtools
b	dbName	Définit le nom de la base de données			
s	serverName	Définit l'adresse du serveur de base de données	*127.0.0.1*		
p	port	Définit le port de la base de données	*3306*		
u	user	Définit l'utilisateur de la base de données	*root*		
w	password	Définit le mot de passe d'accès à la base de données	*""*		
h	themes	Installe les thèmes.		semantic,bootstrap,foundation	
m	allmodels	Crée l'ensemble des modèles depuis la base de données.	*false*		
a	admin	Ajoute les webtools.	*false*		
i	siteUrl	Définit l'URL du site.	*http ://127.0.0.1/{projectname}*		1.2.6
e	rewriteBase	Définit la base de la ré-écriture d'urls.	*/{projectname}/*		1.2.6

4.3 Usage des arguments

4.3.1 noms courts

Exemple de création du projet **blog**, connecté à la base de données **blogDb**, avec génération de tous les modèles

```
Ubiquity new blog -b=blogDb -m=true
```

4.3.2 noms longs

Exemple de création du projet **blog**, connecté à la base de données **bogDb**, avec génération de tous les modèles et intégration du thème sémantic

```
Ubiquity new blog --dbName=blogDb --all-models=true --themes=semantic
```

4.4 Lancement

Pour démarrer le serveur web intégré et tester vos pages, exécutez à partir du dossier racine de l'application :

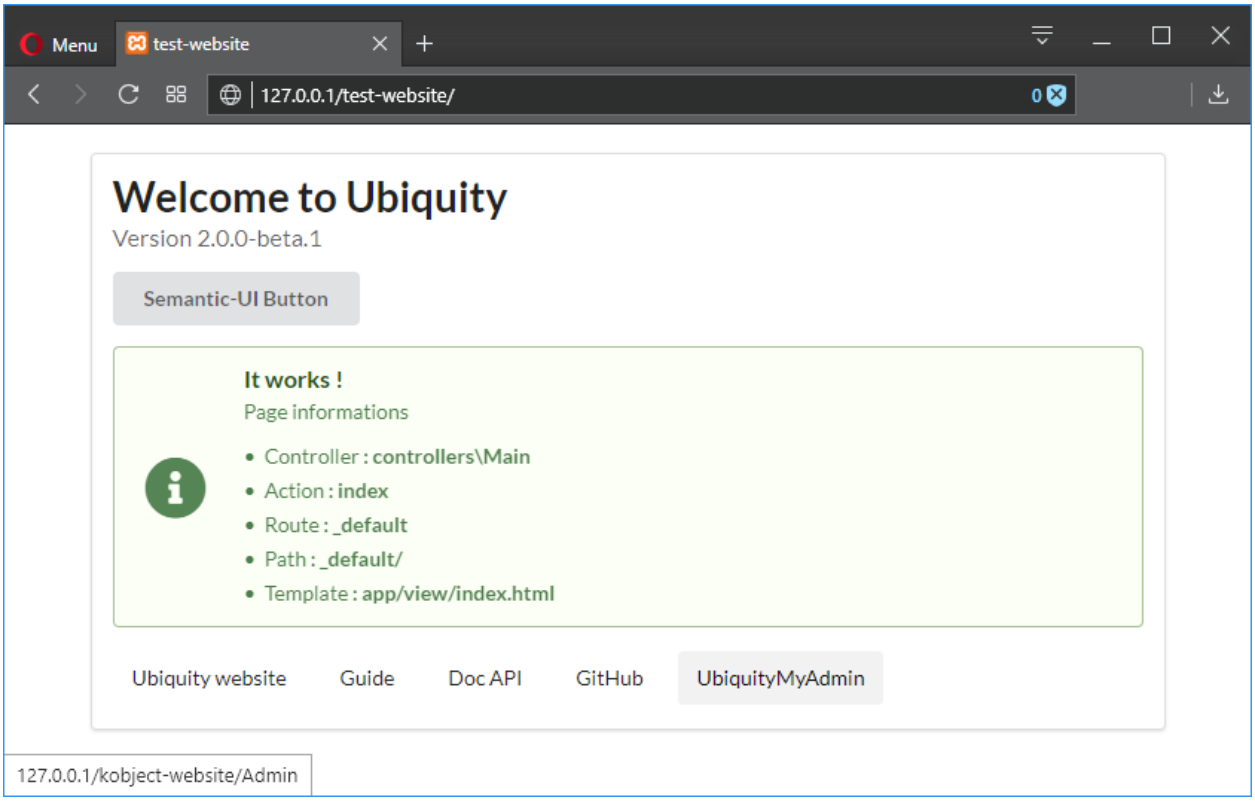
```
Ubiquity serve
```

Le serveur web est démarré à l'adresse `127.0.0.1:8090`

CHAPITRE 5

Configuration de projet

Normalement, le programme d'installation limite les modifications à apporter aux fichiers de configuration et votre application est opérationnelle après l'installation.



5.1 Configuration principale

La configuration principale d'un projet est localisée dans le fichier `app/conf/config.php`.

Code source 1 – app/conf/config.php

```php
return array(
            "siteUrl"=>"%siteUrl%",
            "database"=>[
                            "dbName"=>"%dbName%",
                            "serverName"=>"%serverName%",
                            "port"=>"%port%",
                            "user"=>"%user%",
                            "password"=>"%password%"
            ],
            "namespaces"=>[],
            "templateEngine"=>'Ubiquity\views\engine\twig\Twig',
            "templateEngineOptions"=>array("cache"=>false),
            "test"=>false,
            "debug"=>false,
            "di"=>[%injections%],
            "cacheDirectory"=>"cache/",
            "mvcNS"=>["models"=>"models","controllers"=>"controllers"]
);
```

5.2 Configuration des services

Les services chargés au démarrage sont configurés dans le fichier `app/conf/services.php`.

Code source 2 – app/conf/services.php

```php
    use Ubiquity\controllers\Router;

    try{
            \Ubiquity\cache\CacheManager::startProd($config);
    }catch(Exception $e){
            //Do something
    }
    \Ubiquity\orm\DAO::startDatabase($config);
    Router::start();
    Router::addRoute("_default", "controllers\\IndexController");
```

5.3 URLs sympas

5.3.1 Apache

Le framework est livré avec un fichier **.htaccess** qui est utilisé pour autoriser les URLs sans index.php. Si vous utilisez Apache comme serveur pour votre application Ubiquity, assurez-vous d'activer le module **mod_rewrite**.

Code source 3 – .htaccess

```
AddDefaultCharset UTF-8
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /blog/
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{HTTP_ACCEPT} !(.*images.*)
    RewriteRule ^(.*)$ index.php?c=$1 [L,QSA]
</IfModule>
```

Voir Apache configuration pour des détails supplémentaires.

5.3.2 Nginx

Avec Nginx, la directive suivante dans la configuration de votre site autorisera les URL « sympas » :

```
location /{
     rewrite ^/(.*)$ /index.php?c=$1 last;
}
```

Voir NginX configuration pour des informations supplémentaires.

5.3.3 Driver Laravel Valet

Valet est un environnement de développement php pour les adeptes Mac. Pas de `Vagrant`, pas de fichier `/etc/hosts`. Vous pouvez même partager vos sites publiquement en utilisant des tunnels locaux.

Laravel Valet configure votre Mac pour qu'il exécute toujours `Nginx` en arrière-plan au démarrage de votre machine. Ensuite, en utilisant `DnsMasq`, Valet renvoie toutes les requêtes sur le domaine `*.test` pour pointer vers des sites installés sur votre machine locale.

Plus d'informations sur Laravel Valet

Créez `UbiquityValetDriver.php` sous `~/.config/valet/Drivers/` ajoutez le code php ci-dessous et sauvegardez-le.

```php
<?php

class UbiquityValetDriver extends BasicValetDriver{

        /**
        * Determine if the driver serves the request.
        *
        * @param  string  $sitePath
        * @param  string  $siteName
        * @param  string  $uri
        * @return bool
        */
        public function serves($sitePath, $siteName, $uri){
                if(is_dir($sitePath . DIRECTORY_SEPARATOR . '.ubiquity')) {
                        return true;
                }
                return false;
        }

        public function isStaticFile($sitePath, $siteName, $uri){
                if(is_file($sitePath . $uri)) {
                        return $sitePath . $uri;
                }
                return false;
        }

        /**
        * Get the fully resolved path to the application's front controller.
        *
        * @param  string  $sitePath
        * @param  string  $siteName
        * @param  string  $uri
        * @return string
        */
        public function frontControllerPath($sitePath, $siteName, $uri){
                $sitePath.='/public';
                $_SERVER['DOCUMENT_ROOT'] = $sitePath;
                $_SERVER['SCRIPT_NAME'] = '/index.php';
                $_SERVER['SCRIPT_FILENAME'] = $sitePath . '/index.php';
                $_SERVER['DOCUMENT_URI'] = $sitePath . '/index.php';
                $_SERVER['PHP_SELF'] = '/index.php';
```

(suite sur la page suivante)

(suite de la page précédente)

```
                $_GET['c'] = '';

                if($uri) {
                        $_GET['c'] = ltrim($uri, '/');
                        $_SERVER['PHP_SELF'] = $_SERVER['PHP_SELF']. $uri;
                        $_SERVER['PATH_INFO'] = $uri;
                }

                $indexPath = $sitePath . '/index.php';

                if(file_exists($indexPath)) {
                        return $indexPath;
                }
        }
}
```

CHAPITRE 6

Utilisation des devtools

6.1 Création de projet

Voir *Création de projet* pour créer un projet.

Astuce : Pour toutes les autres commandes, vous devez vous trouver dans le dossier de votre projet ou dans l'un de ses sous-dossiers.

Important : Le dossier `.ubiquity` créé automatiquement avec le projet permet aux devtools de trouver le dossier racine du projet. S'il a été supprimé ou n'est plus présent, vous devez recréer ce dossier vide.

6.2 Création de contrôleur

6.2.1 Spécifications

— commande : `controller`
— Argument : `controller-name`
— aliases : `create-controller`

6.2.2 Paramètres

nom court	nom	rôle	valeur par défaut	valeurs autorisées
v	view	Crée la vue index associée.	true	true, false

6.2.3 Exemples :

Crée la classe contrôleur `controllers\ClientController` dans `app/controllers/ClientController.php` :

```
Ubiquity controller ClientController
```

Crée la classe contrôleur `controllers\ClientController` dans `app/controllers/ClientController.php` et la vue associée dans `app/views/ClientController/index.html` :

```
Ubiquity controller ClientController -v
```

6.3 Création d'action

6.3.1 Spécifications

— commande : `action`
— Argument : `controller-name.action-name`
— aliases : `new-action`

6.3.2 Paramètres

nom court	nom	rôle	valeur par défaut	valeurs autorisées
p	params	Les paramètres (ou arguments) de l'action.		a,b=5 or $a,$b,$c
r	route	Le path de route associé		/path/to/route
v	create-view	Crée la vue associée.	false	true,false

6.3.3 Exemples :

Ajoute l'action « all » dans le contrôleur « Users » :

```
Ubiquity action Users.all
```

Résultat :

Code source 1 – app/controllers/Users.php

```
1 namespace controllers;
2 /**
3  * Controller Users
```

(suite sur la page suivante)

(suite de la page précédente)

```
 */
class Users extends ControllerBase{

   public function index(){}

   public function all(){

   }

}
```

Ajoute l'action `display` dans le contrôleur `Users` avec un paramètre :

```
Ubiquity action Users.display -p=idUser
```

Résultat :

Code source 2 – app/controllers/Users.php

```
class Users extends ControllerBase{

   public function index(){}

   public function display($idUser){

   }
}
```

Ajoute l'action « display » avec une route associée :

```
Ubiquity action Users.display -p=idUser -r=/users/display/{idUser}
```

Résultat :

Attributs

Code source 3 – app/controllers/Users.php

```
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class Users extends ControllerBase{

   public function index(){}

   #[Route('/users/display/{idUser}')]
   public function display($idUser){

   }
}
```

Annotations

Code source 4 – app/controllers/Users.php

```php
namespace controllers;

class Users extends ControllerBase{

   public function index(){}

   /**
    *@route("/users/display/{idUser}")
    */
   public function display($idUser){

   }
}
```

Ajoute l'action `search` avec plusieurs paramètres :

```
Ubiquity action Users.search -p=name,address=''
```

Résultat :

Attributs

Code source 5 – app/controllers/Users.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class Users extends ControllerBase{

   public function index(){}

   #[Route('/users/display/{idUser}')]
   public function display($idUser){

   }

   public function search($name,$address=''){

   }
}
```

Annotations

Code source 6 – app/controllers/Users.php

```php
namespace controllers;

class Users extends ControllerBase{

   public function index(){}

```

(suite sur la page suivante)

(suite de la page précédente)

```
7     /**
8      * @route("/users/display/{idUser}")
9      */
10    public function display($idUser){
11
12    }
13
14    public function search($name,$address=''){
15
16    }
17 }
```

Ajoute l'action `search` et crée la vue associée :

```
Ubiquity action Users.search -p name,address -v
```

6.4 Création de modèle

Note : Vous pouvez vérifier les paramètres de connexion à la base de données dans le fichier app/config/config.php avant d'exécuter ces commandes.

Pour générer un modèle correspondant à la table **user** dans la base de données :

```
Ubiquity model user
```

6.5 Création de tous les modèles

Pour générer tous les modèles à partir de la base de données :

```
Ubiquity all-models
```

6.6 Initialisation du cache

Pour initialiser le cache du routeur (basé sur les annotations dans les contrôleurs) et de l'ORM (basé sur les annotations dans les modèles) :

```
Ubiquity init-cache
```

CHAPITRE 7

URLs

Comme de nombreux autres frameworks, si vous utilisez le routeur avec son comportement par défaut, il existe une relation biunivoque entre une chaîne d'URL et la classe/méthode de contrôleur correspondante. Les segments d'une URI suivent normalement ce modèle : : :

```
example.com/controller/method/param
example.com/controller/method/param1/param2
```

7.1 Méthode par défaut

Lorsque l'URL est composée d'une seule partie, correspondant au nom d'un contrôleur, la méthode index du contrôleur est automatiquement appelée :

URL :

```
example.com/Products
example.com/Products/index
```

Controller :

Code source 1 – app/controllers/Products.php

```
class Products extends ControllerBase{
    public function index(){
        //Default action
    }
}
```

7.2 Paramètres requis

Si la méthode sollicitée a des paramètres obligatoires, ils doivent être passés dans l'URL :

Controller :

Code source 2 – app/controllers/Products.php

```
class Products extends ControllerBase{
    public function display($id){}
}
```

Urls valides :

```
example.com/Products/display/1
example.com/Products/display/10/
example.com/Products/display/ECS
```

7.3 Paramètres optionnels

La méthode appelée peut avoir des paramètres optionnels.

Si un paramètre n'est pas présent dans l'URL, la valeur par défaut du paramètre est utilisée.

Controller :

Code source 3 – app/controllers/Products.php

```
class Products extends ControllerBase{
    public function sort($field, $order='ASC'){}
}
```

Urls valides :

```
example.com/Products/sort/name (uses "ASC" for the second parameter)
example.com/Products/sort/name/DESC
example.com/Products/sort/name/ASC
```

7.4 Sensibilité à la casse

Sur les systèmes Unix, le nom des contrôleurs est sensible à la casse.

Controller :

Code source 4 – app/controllers/Products.php

```
class Products extends ControllerBase{
    public function caseInsensitive(){}
}
```

Urls :

```
example.com/Products/caseInsensitive (valid)
example.com/Products/caseinsensitive (valid because the method names are case␣
↪insensitive)
example.com/products/caseInsensitive (invalid since the products controller does not␣
↪exist)
```

7.5 Personnalisation du routage

Le *Routeur* et les attributs ou annotations dans les classes contrôleur permettent de personnaliser les URLs.

CHAPITRE 8

Routeur

Le routage peut être utilisé en plus du mécanisme par défaut qui associe `controller/action/{paramètres}` à une url.

8.1 Routes dynamiques

Les routes dynamiques sont définies à l'exécution. Il est possible de les déclarer dans le fichier **app/config/services.php**.

Important : Les routes dynamiques ne devraient être utilisées que si la situation l'exige :
- Dans le cas d'une micro-application
- Si une route doit être définie dynamiquement

Dans tous les autres cas, il est conseillé de déclarer les routes avec des annotations ou attributs, afin de bénéficier du cache.

8.1.1 Callback routes

Les routes Ubiquity les plus basiques sont définies à partir d'une Closure. Dans le contexte des micro-applications, cette méthode évite de devoir créer un contrôleur.

Code source 1 – app/config/services.php

```
1  use Ubiquity\controllers\Router;
2
3  Router::get("foo", function(){
4          echo 'Hello world!';
5  });
```

Les routes de type callback peuvent être définies pour toutes les méthodes http :
- Router : :post
- Router : :put

— Router : :delete
— Router : :patch
— Router : :options

8.1.2 Controller routes

Les routes peuvent aussi être définies de manière plus conventionnelle à partir de l'action d'un contrôleur :

Code source 2 – app/config/services.php

```
1  use Ubiquity\controllers\Router;
2
3  Router::addRoute('bar', \controllers\FooController::class,'index');
```

La méthode `FooController::index()` sera accessible via l'url `/bar`.

Dans ce cas, le contrôleur **FooController** doit hériter de **Ubiquity\controllers\Controller** ou l'un de ces dérivés, et doit implémenter la méthode **index** :

Code source 3 – app/controllers/FooController.php

```
1  namespace controllers;
2
3  class FooController extends ControllerBase{
4
5          public function index(){
6                  echo 'Hello from foo';
7          }
8  }
```

8.1.3 Route par défaut

La route par défaut correspond au path **/**. Elle peut être définie en utilisant le path réservé **_default**

Code source 4 – app/config/services.php

```
1  use Ubiquity\controllers\Router;
2
3  Router::addRoute("_default", \controllers\FooController::class,'bar');
```

8.2 Routes statiques

Les routes statiques sont définies via annotations ou en utilisant les attribut php natifs depuis `Ubiquity 2.4.0`.

Note : Ces annotations ou attributs ne sont jamais lus à l'exécution. Il est nécessaire de ré-initialiser le cache du router pour prendre en compte les changements effectués sur les routes.

8.2.1 Création

Attributs

Code source 5 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{

    #[Route('products')]
    public function index(){}

}
```

Annotations

Code source 6 – app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{

    /**
     * @route("products")
     */
    public function index(){}

}
```

La méthode `Products::index()` sera accessible par l'url `/products`.

Note :

Les slash initial ou terminal sont ignorés dans le path. Les routes suivantes sont donc équivalentes :

- `#[Route('products')]`
- `#[Route('/products')]`
- `#[Route('/products/')]`

8.2.2 Paramètres de routes

Une route peut avoir des paramètres :

Attributs

Code source 7 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

```

(suite sur la page suivante)

(suite de la page précédente)

```
class ProductsController extends ControllerBase{
...
     #[Route('products/{value}')]
     public function search($value){
          // $value will equal the dynamic part of the URL
          // e.g. at /products/brocolis, then $value='brocolis'
          // ...
     }
}
```

Annotations

Code source 8 – app/controllers/ProductsController.php

```
namespace controllers;

class ProductsController extends ControllerBase{
...
    /**
     * @route("products/{value}")
     */
     public function search($value){
        // $value will equal the dynamic part of the URL
        // e.g. at /products/brocolis, then $value='brocolis'
        // ...
     }
}
```

8.2.3 Paramètres optionnels des routes

Une route peut définir des paramètres optionnels, si la méthode associée a des arguments optionnels :

Attributs

Code source 9 – app/controllers/ProductsController.php

```
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{
   ...
   #[Route('products/all/{pageNum}/{countPerPage}')]
   public function list($pageNum,$countPerPage=50){
      // ...
   }
}
```

Annotations

Code source 10 – app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{
    ...
    /**
     * @route("products/all/{pageNum}/{countPerPage}")
     */
    public function list($pageNum,$countPerPage=50){
        // ...
    }
}
```

8.2.4 Route requirements

Il est possible d'ajouter des spécifications sur les variables passées dans l'url via l'attribut **requirements**.

Attributs

Code source 11 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{
    ...
    #[Route('products/all/{pageNum}/{countPerPage}',requirements: ["pageNum"=>"\d+",
↪"countPerPage"=>"\d?"])]
    public function list($pageNum,$countPerPage=50){
        // ...
    }
}
```

Annotations

Code source 12 – app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{
    ...
    /**
     * @route("products/all/{pageNum}/{countPerPage}","requirements"=>["pageNum"=>"\d+",
↪"countPerPage"=>"\d?"])
     */
    public function list($pageNum,$countPerPage=50){
        // ...
    }
}
```

La route définie correspond à ces urls :

— products/all/1/20
— products/all/5/

mais pas avec celle-ci :

— products/all/test

8.2.5 Types des paramètres

La déclaration de route prend en compte les types de données passés à l'action, ce qui évite d'ajouter des requirements pour les types simples (int, bool, float).

Attributs

Code source 13 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{
    ...
    #[Route('products/{productNumber}')]
    public function one(int $productNumber){
        // ...
    }
}
```

Annotations

Code source 14 – app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{
    ...
    /**
     * @route("products/{productNumber}")
     */
    public function one(int $productNumber){
        // ...
    }
}
```

La route définie correspond à ces urls :

— products/1
— products/20

mais pas avec celle-ci :

— products/test

Valeurs possibles par type de données :

— int : 1...
— bool : 0 or 1
— float : 1 1.0 ...

8.2.6 Méthodes http des routes

Il est possible de spécifier la ou les méthodes http associées à une route :

Attributs

Code source 15 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{

    #[Route('products',methods: ['get','post'])]
    public function index(){}

}
```

Annotations

Code source 16 – app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{

    /**
     * @route("products","methods"=>["get","post"])
     */
    public function index(){}

}
```

L'attribut **methods** peut accepter plusieurs méthodes : `@route("testMethods","methods"=>["get","post","delete"])` `#[Route('testMethods', methods: ['get','post','delete'])]`

L'annotation **@route** ou l'attribut **Route** correspondent à l'ensemble des méthodes http. Une annotation spécifique existe pour chaque méthode http :

- **@get => Get**
- **@post => Post**
- **@put => Put**
- **@patch => Patch**
- **@delete => Delete**
- **@head => Head**
- **@options => Options**

Attributs

Code source 17 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Get;

class ProductsController extends ControllerBase{
```

(suite sur la page suivante)

(suite de la page précédente)

```
6
7       #[Get('products')]
8       public function index(){}
9
10  }
```

Annotations

Code source 18 – app/controllers/ProductsController.php

```
1   namespace controllers;
2
3   class ProductsController extends ControllerBase{
4
5       /**
6        * @get("products")
7        */
8       public function index(){}
9
10  }
```

8.2.7 Nom de route

Il est possible de spécifier le nom **name** d'une route pour faciliter l'accès à l'url associée. Si l'attribut **name** n'est pas spécifié, chaque route a un nom par défaut, basé sur le pattern **controllerName.methodName**.

Attributs

Code source 19 – app/controllers/ProductsController.php

```
1   namespace controllers;
2
3   use Ubiquity\attributes\items\router\Route;
4
5   class ProductsController extends ControllerBase{
6
7       #[Route('products',name: 'products.index')]
8       public function index(){}
9
10  }
```

Annotations

Code source 20 – app/controllers/ProductsController.php

```
1   namespace controllers;
2
3   class ProductsController extends ControllerBase{
4
5       /**
6        * @route("products","name"=>"products.index")
7        */
```

(suite sur la page suivante)

(suite de la page précédente)

```
    public function index(){}

}
```

8.2.8 Génération d'URL

Les routes names peuvent être utilisés pour générer les URLs.

Liens vers une route en Twig

```
<a href="{{ path('products.index') }}">Products</a>
```

8.2.9 Route globale

L'annotation **@route** peut être utilisée sur une classe contrôleur :

Attributs

Code source 21 – app/controllers/ProductsController.php

```
namespace controllers;

use Ubiquity\attributes\items\router\Route;

#[Route('products')]
class ProductsController extends ControllerBase{
    ...
    #[Route('/all')]
    public function display(){}

}
```

Annotations

Code source 22 – app/controllers/ProductsController.php

```
namespace controllers;
/**
 * @route("/product")
 */
class ProductsController extends ControllerBase{

    ...
    /**
     * @route("/all")
     */
    public function display(){}

}
```

Dans ce cas, la route définie sur le contrôleur est utilisée en tant que préfixe pour toutes les routes du contrôleur : La route générée pour l'action **display** est `/product/all`

Routes automatiques

Si une route globale est définie, il est possible d'ajouter toutes les actions du contrôleur en tant que routes (en utilisant le préfixe global), en spécifiant le paramètre **automated** :

Attributs

Code source 23 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

#[Route('/products',automated: true)]
class ProductsController extends ControllerBase{

    public function index(){}

    public function generate(){}

    public function display($id){}

}
```

Annotations

Code source 24 – app/controllers/ProductsController.php

```php
namespace controllers;
/**
 * @route("/product","automated"=>true)
 */
class ProductsController extends ControllerBase{

    public function index(){}

    public function generate(){}

    public function display($id){}

}
```

L'attribut automated définit 3 routes depuis ProductsController :

- */product/(index/)?*
- */product/generate*
- */product/display/{id}*

Routes héritées

Avec l'attribut **inherited**, il est possible de générer les routes déclarées dans la classe de base, ou de générer automatiquement les routes associées aux actions de la classe de base, si l'attribut **automated** est mis à true en même temps.

Classe de base :

Attributs

Code source 25 – app/controllers/ProductsBase.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

abstract class ProductsBase extends ControllerBase{

    #[Route('(index/)?')]
    public function index(){}

    #[Route('sort/{name}')]
    public function sortBy($name){}

}
```

Annotations

Code source 26 – app/controllers/ProductsBase.php

```php
namespace controllers;

abstract class ProductsBase extends ControllerBase{

   /**
    *@route("(index/)?")
    */
   public function index(){}

   /**
    * @route("sort/{name}")
    */
   public function sortBy($name){}

}
```

La classe dérivée utilisant les membres hérités :

Attributs

Code source 27 – app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

```

(suite sur la page suivante)

(suite de la page précédente)

```
#[Route('/product',inherited: true)]
class ProductsController extends ProductsBase{

   public function display(){}

}
```

Annotations

Code source 28 – app/controllers/ProductsController.php

```
namespace controllers;
/**
 * @route("/product","inherited"=>true)
 */
class ProductsController extends ProductsBase{

   public function display(){}

}
```

L'attribut inherited définit les 2 routes de ProductsBase :

- */products/(index/)?*
- */products/sort/{name}*

Si les attributs **automated** et **inherited** sont utilisés en conjonction, les actions de la classe de base sont également ajoutées aux routes.

Paramètres globaux de routes

La partie globale d'une route peut définir des paramètres, qui seront passés dans toutes les routes générées. Ces paramètres peuvent être récupérés par le biais d'un membre de données public :

Attributs

Code source 29 – app/controllers/FooController.php

```
namespace controllers;

use Ubiquity\attributes\items\router\Route;

#[Route('/foo/{bar}',automated: true)]
class FooController {

   public string $bar;

   public function display(){
       echo $this->bar;
   }

}
```

Annotations

Code source 30 – app/controllers/FooController.php

```php
namespace controllers;

/**
 * @route("/foo/{bar}","automated"=>true)
 */
class FooController {

   public string $bar;

   public function display(){
       echo $this->bar;
   }

}
```

L'accès à l'url `/foo/bar/display` affiche le contenu du membre bar.

Routes avec préfixe global

Si la route globale est définie sur un contrôleur, toutes les routes générées dans ce contrôleur sont précédées du préfixe. Il est possible d'introduire explicitement des exceptions sur certaines routes, en utilisant le préfixe `#/`.

Attributs

Code source 31 – app/controllers/FooController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

#[Route('/foo',automated: true)]
class FooController {

   #[Route('#/noRoot')]
   public function noRoot(){}

}
```

Annotations

Code source 32 – app/controllers/FooController.php

```php
namespace controllers;

/**
 * @route("/foo","automated"=>true)
 */
class FooController {

  /**
   * @route("#/noRoot")
```

(suite sur la page suivante)

(suite de la page précédente)

```
    */
    public function noRoot(){}

}
```

Le contrôleur définit l'url `/noRoot` url, qui n'est pas préfixée par `/foo`.

8.2.10 Priorité des routes

Le paramètre **prority** d'une route permet à celle ci d'être résolue avec une plus ou moins grande priorité.

Plus le paramètre de priorité est élevé, plus la route sera définie au début de la pile des routes en cache.

Dans l'exemple ci-dessous, la route **produits/all** sera définie avant la route **/produits**.

Attributs

Code source 33 – app/controllers/ProductsController.php

```
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{

    #[Route('products', priority: 1)]
    public function index(){}

    #[Route('products/all', priority: 10)]
    public function all(){}

}
```

Annotations

Code source 34 – app/controllers/ProductsController.php

```
namespace controllers;

class ProductsController extends ControllerBase{

    /**
     * @route("products","priority"=>1)
     */
    public function index(){}

    /**
     * @route("products/all","priority"=>10)
     */
    public function all(){}

}
```

La valeur par défaut pour priority est `0`.

8.3 Mise en cache de réponse d'une route

Il est possible de mettre en cache la réponse produite par une route :

Dans ce cas, la réponse est en cache et n'est plus générée dynamiquement.

Attributs

```
#[Route('products/all', cache: true)]
public function all(){}
```

Annotations

```
/**
 * @route("products/all","cache"=>true)
 */
public function all(){}
```

8.3.1 Durée du cache

La **duration** est exprimée en secondes, si elle est omise, la validité du cache est infinie.

Attributs

```
#[Route('products/all', cache: true, duration: 3600)]
public function all(){}
```

Annotations

```
/**
 * @route("products/all","cache"=>true,"duration"=>3600)
 */
public function all(){}
```

8.3.2 Expiration du cache

Il est possible de forcer le rechargement de la réponse en supprimant le cache associé.

```
Router::setExpired("products/all");
```

8.4 Mise en cache des routes dynamiques

Les routes dynamiques peuvent également être mises en cache.

Important : Cette possibilité n'a d'intérêt que si cette mise en cache n'est pas faite en production, mais au moment de l'initialisation du cache.

```
Router::get("foo", function(){
   echo 'Hello world!';
});

Router::addRoute("string", \controllers\Main::class,"index");
CacheManager::storeDynamicRoutes(false);
```

Vérification des routes avec les devtools :

```
Ubiquity info:routes
```

```
· The project folder is C:\xampp7.4.4\htdocs\quick-start
```

path	controller	action	parameters
'/_default/'	'controllers\\IndexController'	'index'	
'/string/'	≡	≡	≡
'/foo/'	(x)=>{}	''	≡

```
· 3 routes (routes)
```

8.5 Gestion des erreurs (404 & 500)

8.5.1 Routage par défaut

Avec le système de routage par défaut (le couple contrôleur+action définissant une route), un gestionnaire d'erreurs peut être redéfini pour personnaliser la gestion des erreurs.

Dans le fichier de configuration **app/config/config.php**, ajoutez la clé **onError**, associée à un callback définissant les messages d'erreur :

```
"onError"=>function ($code, $message = null,$controller=null){
   switch($code){
      case 404:
         $init=($controller==null);
         \Ubiquity\controllers\Startup::forward('IndexController/p404',$init,$init);
         break;
   }
}
```

Implémente l'action sollicitée **p404** dans **IndexController** :

Code source 35 – app/controllers/IndexController.php

```
...

public function p404(){
   echo "<div class='ui error message'><div class='header'>404</div>The page you are␣
↪looking for doesn't exist!</div>";
}
```

8.5.2 Routage avec annotations

Il suffit dans ce cas d'ajouter une dernière route désactivant le système de routage par défaut, et correspondant à la gestion de l'erreur 404 :

Attributs

Code source 36 – app/controllers/IndexController.php

```
...

#[Route('{url}', priority: -1000)]
public function p404($url){
   echo "<div class='ui error message'><div class='header'>404</div>The page `$url` you␣
↪are looking for doesn't exist!</div>";
}
```

Annotations

Code source 37 – app/controllers/IndexController.php

```
...

/**
 * @route("{url}","priority"=>-1000)
 */
public function p404($url){
   echo "<div class='ui error message'><div class='header'>404</div>The page `$url` you␣
↪are looking for doesn't exist!</div>";
}
```

CHAPITRE 9

Contrôleurs

Un contrôleur est une classe PHP héritant de `Ubiquity\controllers\Controller` et fournissant un point d'entrée dans l'application. Les contrôleurs et leurs méthodes définissent les URLs accessibles.

9.1 Création de contrôleur

La méthode la plus facile pour créer un contrôleur est de le faire depuis les devtools.

A partir de l'invite de commande, aller dans le dossier du projet. Pour créer le contrôleur Products, utiliser la commande :

```
Ubiquity controller Products
```

Le contrôleur `Products.php` créé dans le dossier `app/controllers` du projet.

Code source 1 – app/controllers/Products.php

```
1 namespace controllers;
2 /**
3  * Controller Products
4  */
5 class Products extends ControllerBase{
6
7 public function index(){}
8
9 }
```

Il est maintenant possible d'accéder à l'URL (la méthode `index` est sollicitée par défaut) :

```
example.com/Products
example.com/Products/index
```

Note : Un contrôleur peut être créé manuellement. Dans ce cas, il doit respecter les règles suivantes :
- La classe doit être définie dans le dossier **app/controllers**
- Le nom de la classe doit correspondre au nom du fichier php
- La classe doit hériter de **ControllerBase** et doit être définie dans le namespace **controllers**
- Elle doit surdéfinir la méthode abstraite **index**

9.2 Méthodes

9.2.1 public

Le second segment de l'URL détermine la méthode publique du contrôleur sollicitée. La méthode « index » est appelée par défaut, si le second segment est vide.

Code source 2 – app/controllers/First.php

```
1 namespace controllers;
2 class First extends ControllerBase{
3
4    public function hello(){
5       echo "Hello world!";
6    }
7
8 }
```

La méthode `hello` du contrôleur `First` met à disposition les URL suivantes :

```
example.com/First/hello
```

9.2.2 Arguments de méthode

Les arguments d'une méthode doivent être passés dans l'url, excepté s'ils sont optionnels.

Code source 3 – app/controllers/First.php

```
namespace controllers;
class First extends ControllerBase{

public function says($what,$who='world') {
   echo $what.' '.$who;
}

}
```

La méthode `hello` du contrôleur `First` met à disposition les URL suivantes :

```
example.com/First/says/hello (says hello world)
example.com/First/says/Hi/everyone (says Hi everyone)
```

9.2.3 private

Les méthodes privées ou protégées ne sont pas accessibles depuis l'URL.

9.3 Contrôleur par défaut

Le contrôleur par défaut peut être défini par le biais du Router, dans le fichier `services.php`

Code source 4 – app/config/services.php

```
Router::start();
Router::addRoute("_default", "controllers\First");
```

Dans ce cas, l'accès à l'URL « exemple.com/ » charge le contrôleur **First** et appelle la méthode par défaut **index**.

9.4 Chargement des vues

9.4.1 chargement

Les vues sont stockées dans le dossier `app/views`. Elles sont chargées à partir des méthodes du contrôleur. Par défaut, il est possible de créer des vues en php, ou avec twig. Twig est le moteur de template par défaut pour les fichiers html.

Chargement des vues php

Si l'extension du fichier n'est pas spécifiée, la méthode **loadView** charge un fichier php.

Code source 5 – app/controllers/First.php

```
namespace controllers;
class First extends ControllerBase{
   public function displayPHP(){
      //loads the view app/views/index.php
      $this->loadView('index');
   }
}
```

Chargement des vues twig

Si l'extension du fichier est html, la méthode **loadView** charge un fichier twig html.

Code source 6 – app/controllers/First.php

```
namespace controllers;
class First extends ControllerBase{
   public function displayTwig(){
      //loads the view app/views/index.html
      $this->loadView("index.html");
   }
}
```

Chargement par défaut de vue

Si vous utilisez la méthode de dénomination des vues par défaut : La vue par défaut associée à une action dans un contrôleur est située dans le dossier `views/controller-name/action-name` :

```
views
     |
     └ Users
         └ info.html
```

Code source 7 – app/controllers/Users.php

```php
namespace controllers;

class Users extends BaseController{
   ...
   public function info(){
      $this->loadDefaultView();
   }
}
```

9.4.2 Variables associées à une vue

Une des missions du contrôleur est de passer des variables à la vue. Ceci peut être fait au chargement de la vue, à partir d'un tableau associatif :

Code source 8 – app/controllers/First.php

```php
class First extends ControllerBase{
   public function displayTwigWithVar($name){
      $message="hello";
      //loads the view app/views/index.html
      $this->loadView('index.html', ['recipient'=>$name, 'message'=>$message]);
   }
}
```

Les clés du tableau associatif créent des variables du même nom dans la vue. Utilisation de ces variables dans Twig :

Code source 9 – app/views/index.html

```
<h1>{{message}} {{recipient}}</h1>
```

Les variables peuvent également être passées avant le chargement de la vue :

```php
//passing one variable
$this->view->setVar('title','Message');
//passing an array of 2 variables
$this->view->setVars(['message'=>$message,'recipient'=>$name]);
//loading the view that now contains 3 variables
$this->loadView('First/index.html');
```

9.4.3 retour de l'affichage d'une vue dans une chaîne

Il est possible de charger une vue, et de retourner le résultat dans une chaîne, en mettant à true le 3ème paramètre de la méthode loadview :

```
$viewResult=$this->loadView("First/index.html",[],true);
echo $viewResult;
```

9.4.4 chargement de plusieurs vues

Une action peut charger plusieurs vues

Code source 10 – app/controllers/Products.php

```
namespace controllers;
class Products extends ControllerBase{
   public function all(){
      $this->loadView('Main/header.html', ['title'=>'Products']);
      $this->loadView('Products/index.html',['products'=>$this->products]);
      $this->loadView('Main/footer.html');
   }
}
```

Important : Une vue est souvent partielle. Il est donc important de ne pas intégrer systématiquement les balises **html** et **body** définissant une page html complète.

9.4.5 organisation des vues

Il est conseillé d'organiser les vues en dossiers. La méthode la plus recommandée est de créer un dossier par contrôleur, et d'y stocker les vues associées. Pour charger la vue `index.html`, stockée dans `app/views/First` :

```
$this->loadView("First/index.html");
```

9.5 initialize et finalize

La méthode **initialize** est automatiquement appelée avant chaque action sollicitée, la méthode **finalize** après chaque action.

Exemple d'utilisation des méthodes initialize et finalize créées automatiquement au sein d'un nouveau projet :

Code source 11 – app/controllers/ControllerBase.php

```
namespace controllers;

use Ubiquity\controllers\Controller;
use Ubiquity\utils\http\URequest;

/**
```

(suite sur la page suivante)

(suite de la page précédente)

```
 * ControllerBase.
 */
abstract class ControllerBase extends Controller{
   protected $headerView = "@activeTheme/main/vHeader.html";
   protected $footerView = "@activeTheme/main/vFooter.html";

   public function initialize() {
      if (! URequest::isAjax ()) {
         $this->loadView ( $this->headerView );
      }
   }

   public function finalize() {
      if (! URequest::isAjax ()) {
         $this->loadView ( $this->footerView );
      }
   }
}
```

9.6 Contrôle d'accès

Le contrôle d'accès à un contrôleur peut être effectué manuellement, à l'aide des méthodes *isValid* et *onInvalidControl*.

La méthode *isValid* doit retourner un booléen permettant de définir si l'action sollicitée passée en paramètre est accessible.

Dans l'exemple suivant, l'accès aux actions du contrôleur **IndexController** n'est possible que si une variable de session **activeUser** existe :

Code source 12 – app/controllers/IndexController.php

```
class IndexController extends ControllerBase{
...
   public function isValid($action){
      return USession::exists('activeUser');
   }
}
```

Si la variable **activeUser** n'existe pas, une erreur **unauthorized 401** est renvoyée.

La méthode *onInvalidControl* permet de personnalisé le comportement de l'accès non autorisé :

Code source 13 – app/controllers/IndexController.php

```
class IndexController extends ControllerBase{
   ...
   public function isValid($action){
      return USession::exists('activeUser');
   }

   public function onInvalidControl(){
      $this->initialize();
```

(suite sur la page suivante)

(suite de la page précédente)

```
        $this->loadView('unauthorized.html');
        $this->finalize();
    }
}
```

Code source 14 – app/views/unauthorized.html

```
<div class="ui container">
    <div class="ui brown icon message">
        <i class="ui ban icon"></i>
        <div class="content">
            <div class="header">
                Error 401
            </div>
            <p>You are not authorized to access to <b>{{app.getController() ~ "::" ~ app.
↪getAction()}}</b>.</p>
        </div>
    </div>
</div>
```

Il est aussi possible de gérer le contrôle d'accès à partir des *AuthControllers*

9.7 Redirection

Une redirection n'est pas un simple appel à une action d'un contrôleur. La redirection sollicite également les méthodes *initialize* et *finalize*, ainsi que le contrôle d'accès.

La méthode **forward** peut être invoquée sans la sollicitation des méthodes *initialize* et *finalize*.

Il est possible d'effectuer une redirection vers une route par son nom.

9.8 Injection de dépendances

Voir *Dependency injection*

9.9 namespaces

Le namespace des contrôleurs est défini par défaut à *controllers* dans le fichier *app/config/config.php*.

9.10 Classe de base

L'héritage peut être utiliser pour factoriser le comportement des contrôleurs. La classe *BaseController* est créée par défaut au sein des nouveaux projets dans cette logique.

9.11 Classe de base contrôleur spécifiques

Classe contrôleur	rôle
Contrôleur	Classe de base pour tous les contrôleurs
SimpleViewController	Classe de base associée à un moteur de template php basique (pour utilisation avec des micro-services)
SimpleViewAsyncController	Classe de base associée avec un moteur de template php pour les serveurs asynchrones

CHAPITRE 10

Evènements

Note : Le module Events utilise la classe statique **EventsManager** pour gérer les évènements.

10.1 Framework core events

Ubiquity émet des événements lors des différentes phases de soumission d'une requête. Ces événements sont relativement peu nombreux, afin de limiter leur impact sur les performances.

Partie	Nom d'évènement	Paramètres	Se produit quand
ViewEvents	BEFORE_RENDER	viewname, parameters	Avant d'afficher une vue
ViewEvents	AFTER_RENDER	viewname, parameters	Après l'affichage d'une vue
DAOEvents	GET_ALL	objects, classname	Après le chargement d'un ensemble d'objets
DAOEvents	GET_ONE	object, classname	Après le chargement d'un objet
DAOEvents	BEFORE_UPDATE	instance	Avant la mise à jour d'un objet
DAOEvents	AFTER_UPDATE	instance, result	Après mise à jour d'un objet
DAOEvents	BEFORE_INSERT	instance	Avant l'insertion d'un objet
DAOEvents	AFTER_INSERT	instance, result	Après l'insertion d'un objet
RestEvents	BEFORE_INSERT	instance	Avant l'insertion d'un objet
RestEvents	BEFORE_UPDATE	instance	Avant la mise à jour d'un objet

Note : Il n'y a pas d'événement **BeforeAction** et **AfterAction**, puisque les méthodes **initialize** et **finalize** de la classe contrôleur effectuent ces opérations.

10.2 Ecouter un évènement

Exemple 1 :

Ajout d'une propriété **_updated** sur les instances modifiées dans la base de données :

Code source 1 – app/config/services.php

```
use Ubiquity\events\EventsManager;
use Ubiquity\events\DAOEvents;

...

    EventsManager::addListener(DAOEvents::AFTER_UPDATE, function($instance,$result){
            if($result==1){
                    $instance->_updated=true;
            }
    });
```

Note : Les paramètres passés à la fonction de callback varient en fonction de l'évènement écouté.

Exemple 2 :

Modification de l'affichage de la vue

Code source 2 – app/config/services.php

```
use Ubiquity\events\EventsManager;
use Ubiquity\events\ViewEvents;

...

    EventsManager::addListener(ViewEvents::AFTER_RENDER,function(&$render,$viewname,
↪$datas){
            $render='<h1>'.$viewname.'</h1>'.$render;
    });
```

10.3 Créer ses propres évènements

Exemple :

Création d'un évènement pour compter et mémoriser le nombre d'affichages par action :

Code source 3 – app/eventListener/TracePageEventListener.php

```
    namespace eventListener;

    use Ubiquity\events\EventListenerInterface;
    use Ubiquity\utils\base\UArray;

    class TracePageEventListener implements EventListenerInterface {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        const EVENT_NAME = 'tracePage';

        public function on(&...$params) {
                $filename = \ROOT . \DS . 'config\stats.php';
                $stats = [ ];
                if (file_exists ( $filename )) {
                        $stats = include $filename;
                }
                $page = $params [0] . '::' . $params [1];
                $value = $stats [$page] ?? 0;
                $value ++;
                $stats [$page] = $value;
                UArray::save ( $stats, $filename );
        }
    }
```

10.4 Enregistrement d'évènements

Enregistrement de l'évènement **TracePageEventListener** dans `services.php` :

Code source 4 – app/config/services.php

```
    use Ubiquity\events\EventsManager;
    use eventListener\TracePageEventListener;

    ...

    EventsManager::addListener(TracePageEventListener::EVENT_NAME,␣
↪TracePageEventListener::class);
```

10.5 Déclencher des évènements

Un évènement peut être déclenché depuis n'importe quel contexte, mais il est plus logique de le faire depuis la méthode **initialize** du contrôleur de base.

Code source 5 – app/controllers/ControllerBase.php

```
    namespace controllers;

    use Ubiquity\controllers\Controller;
    use Ubiquity\utils\http\URequest;
    use Ubiquity\events\EventsManager;
    use eventListener\TracePageEventListener;
    use Ubiquity\controllers\Startup;

    /**
     * ControllerBase.
     **/
```

(suite sur la page suivante)

(suite de la page précédente)

```
12      abstract class ControllerBase extends Controller{
13              protected $headerView = "@activeTheme/main/vHeader.html";
14              protected $footerView = "@activeTheme/main/vFooter.html";
15              public function initialize() {
16                      $controller=Startup::getController();
17                      $action=Startup::getAction();
18                      EventsManager::trigger(TracePageEventListener::EVENT_NAME,
    ↪$controller,$action);
19                      if (! URequest::isAjax ()) {
20                              $this->loadView ( $this->headerView );
21                      }
22              }
23              public function finalize() {
24                      if (! URequest::isAjax ()) {
25                              $this->loadView ( $this->footerView );
26                      }
27              }
28      }
```

Le résultat dans app/config/stats.php :

Code source 6 – app/config/stats.php

```
return array(
            "controllers\\IndexController::index"=>5,
            "controllers\\IndexController::ct"=>1,
            "controllers\\NewController::index"=>1,
            "controllers\\TestUCookieController::index"=>1
    );
```

10.6 Optimisation de l'enregistrement d'évènements

Il est préférable de mettre en cache l'enregistrement des écouteurs, afin d'optimiser leur temps de chargement :

Créer un script client, ou une action contrôleur (non accessible en mode production) :

```
use Ubiquity\events\EventsManager;

public function initEvents(){
        EventsManager::start();
        EventsManager::addListener(DAOEvents::AFTER_UPDATE, function($instance,$result){
                if($result==1){
                        $instance->_updated=true;
                }
        });
        EventsManager::addListener(TracePageEventListener::EVENT_NAME,␣
    ↪TracePageEventListener::class);
        EventsManager::store();
}
```

Après exécution, le cache est généré dans le fichier `app/cache/events/events.cache.php`.

Une fois le cache généré, le fichier `services.php` doit juste inclure cette ligne :

```
\Ubiquity\events\EventsManager::start();
```

CHAPITRE 11

Injection de dépendances

Note : Pour des raisons de performances, l'injection de dépendances n'est pas utilisée dans le cœur du framework.

L'injection de dépendances (DI) est un modèle de conception utilisé pour mettre en œuvre l'IoC. Il permet de créer des objets dépendants en dehors d'une classe et de fournir ces objets à une classe de différentes manières. En utilisant l'injection de dépendances, nous déplaçons la création et la liaison des objets dépendants en dehors de la classe qui en dépend.

Note : Ubiquity ne supporte que l'injection de propriétés, afin de ne pas recourir à l'introspection à l'exécution. Seuls les contrôleurs supportent l'injection de dépendances.

11.1 Autowiring de service

11.1.1 Création de service

Créer un service

Code source 1 – app/services/Service.php

```
1  namespace services;
2
3      class Service{
4          public function __construct($ctrl){
5              echo 'Service instanciation in '.get_class($ctrl);
6          }
7
8          public function do($someThink=""){
9              echo 'do '.$someThink ."in service";
```

(suite sur la page suivante)

(suite de la page précédente)

```
        }
    }
```

11.1.2 Autowiring dans un contrôleur

Créer un contrôleur utilisant le service

Code source 2 – app/services/Service.php

```
namespace controllers;

 /**
 * Controller Client
 **/
class ClientController extends ControllerBase{

        /**
         * @autowired
         * @var services\Service
         */
        private $service;

        public function index(){}

        /**
         * @param \services\Service $service
         */
        public function setService($service) {
                $this->service = $service;
        }
}
```

Dans l'exemple ci-dessus, Ubiquity recherche et injecte **$service** lorsque **ClientController** est créé.

L'annotation @autowired nécessite que :

- Le type à instancier soit déclaré avec l'annotation **@var**
- La propriété **$service** ait un setteur, ou soit déclarée publique

Etant donné que les annotations ne sont jamais lues à l'exécution, il est nécessaire de générer le cache des contrôleurs :

```
Ubiquity init-cache -t=controllers
```

Reste à vérifier que le service est bien injecté, en allant à l'adresse `/ClientController`.

11.2 Injection de service

11.2.1 Service

Créons maintenant un deuxième service, nécessitant une initialisation spéciale.

Code source 3 – app/services/ServiceWithInit.php

```
1  class ServiceWithInit{
2          private $init;
3
4          public function init(){
5                  $this->init=true;
6          }
7
8          public function do(){
9                  if($this->init){
10                         echo 'init well initialized!';
11                 }else{
12                         echo 'Service not initialized';
13                 }
14         }
15 }
```

11.2.2 Injection dans le contrôleur

Code source 4 – app/controllers/ClientController.php

```
1  namespace controllers;
2
3      /**
4       * Controller Client
5       **/
6      class ClientController extends ControllerBase{
7
8              /**
9               * @autowired
10              * @var \services\Service
11              */
12             private $service;
13
14             /**
15              * @injected
16              */
17             private $serviceToInit;
18
19             public function index(){
20                     $this->serviceToInit->do();
21             }
22
23             /**
```

(suite sur la page suivante)

(suite de la page précédente)

```
24              * @param \services\Service $service
25              */
26             public function setService($service) {
27                     $this->service = $service;
28             }
29
30             /**
31              * @param mixed $serviceToInit
32              */
33             public function setServiceToInit($serviceToInit) {
34                     $this->serviceToInit = $serviceToInit;
35             }
36
37     }
```

11.2.3 Di déclaration

Dans `app/config/config.php`, créez une nouvelle clé pour la propriété **serviceToInit** à injecter dans la partie **di**.

```
"di"=>["ClientController.serviceToInit"=>function(){
                        $service=new \services\ServiceWithInit();
                        $service->init();
                        return $service;
                }
        ]
```

générer le cache des contrôleurs :

```
Ubiquity init-cache -t=controllers
```

Vérifier que le service est bien injecté en allant à l'adresse `/ClientController`.

Note : Si le même service doit être utilisé dans plusieurs contrôleurs, utilisez la notation avec joker :

```
"di"=>["*.serviceToInit"=>function(){
                        $service=new \services\ServiceWithInit();
                        $service->init();
                        return $service;
                }
        ]
```

11.2.4 Injection avec un nom

Si le nom du service à injecter est différent de la clé du tableau **di**, il est possible d'utiliser l'attribut name de l'annotation **@injected**.

Dans `app/config/config.php`, créez une nouvelle clé pour la propriété **serviceToInit** à injecter dans la partie **di**.

```
"di"=>["*.service"=>function(){
                        $service=new \services\ServiceWithInit();
                        $service->init();
                        return $service;
                    }
        ]
```

```
/**
 * @injected("service")
 */
private $serviceToInit;
```

11.3 Injection de service à l'exécution

Il est possible d'injecter des services à l'exécution, sans qu'ils aient été précédemment déclarés dans une classe contrôleur.

Code source 5 – app/services/RuntimeService.php

```
1  namespace services;
2
3      class RuntimeService{
4          public function __construct($ctrl){
5              echo 'Service instanciation in '.get_class($ctrl);
6          }
7      }
```

Dans `app/config/config.php`, ajouter la clé **@exec** dans la partie **di**.

```
"di"=>["@exec"=>"rService"=>function($ctrl){
                        return new \services\RuntimeService($ctrl);
                    }
        ]
```

Avec cette déclaration, le membre **$rService**, instance de **RuntimeService**, est injecté dans tous les contrôleurs. Il est donc conseillé d'utiliser les commentaires de javadoc pour déclarer **$rService** dans les contrôleurs qui l'utilisent (pour obtenir la complétion de code sur **$rService** dans votre IDE).

Code source 6 – app/controllers/MyController.php

```
1  namespace controllers;
2
3      /**
4      * Controller Client
5      * property services\RuntimeService $rService
```

(suite sur la page suivante)

(suite de la page précédente)

```
6       **/
7       class MyController extends ControllerBase{
8
9               public function index(){
10                      $this->rService->do();
11              }
12      }
```

CHAPITRE 12

Contrôleurs CRUD

Les contrôleurs CRUD vous permettent d'effectuer des opérations de base sur une classe Modèle :

- Create
- Read
- Update
- Delete
- …

Note :

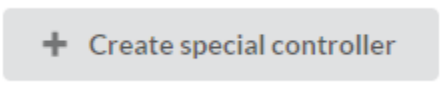
Depuis la version 2.4.6, deux types de CrudController existent :

- *ResourceCrudController* associé à un modèle
- *MultiResourceCRUDController*, affichant un index et permettant de naviguer entre les modèles.

12.1 ResourceCrudController

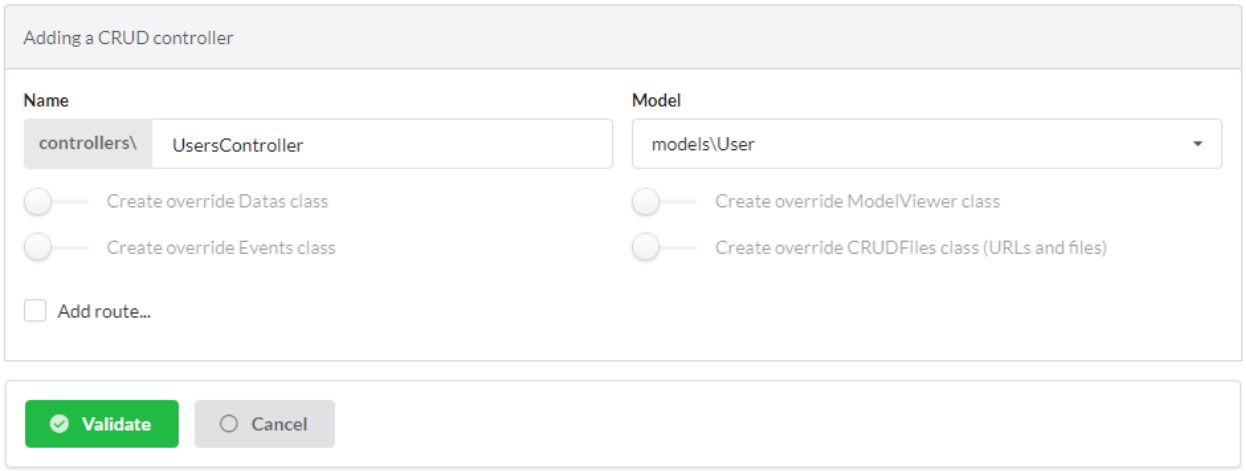
12.1.1 Création

Dans l'interface d'administration (webtools), activez la partie **Controllers**, et choisissez **Resource Crud controller** :



Remplissez ensuite le formulaire :

- Entrez le nom du contrôleur
- Sélectionnez le modèle associé
- Cliquez ensuite sur le bouton de validation



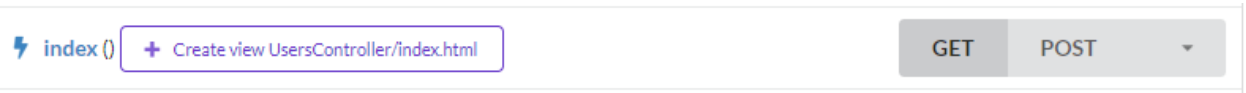
12.1.2 Description des caractéristiques

Le contrôleur généré :

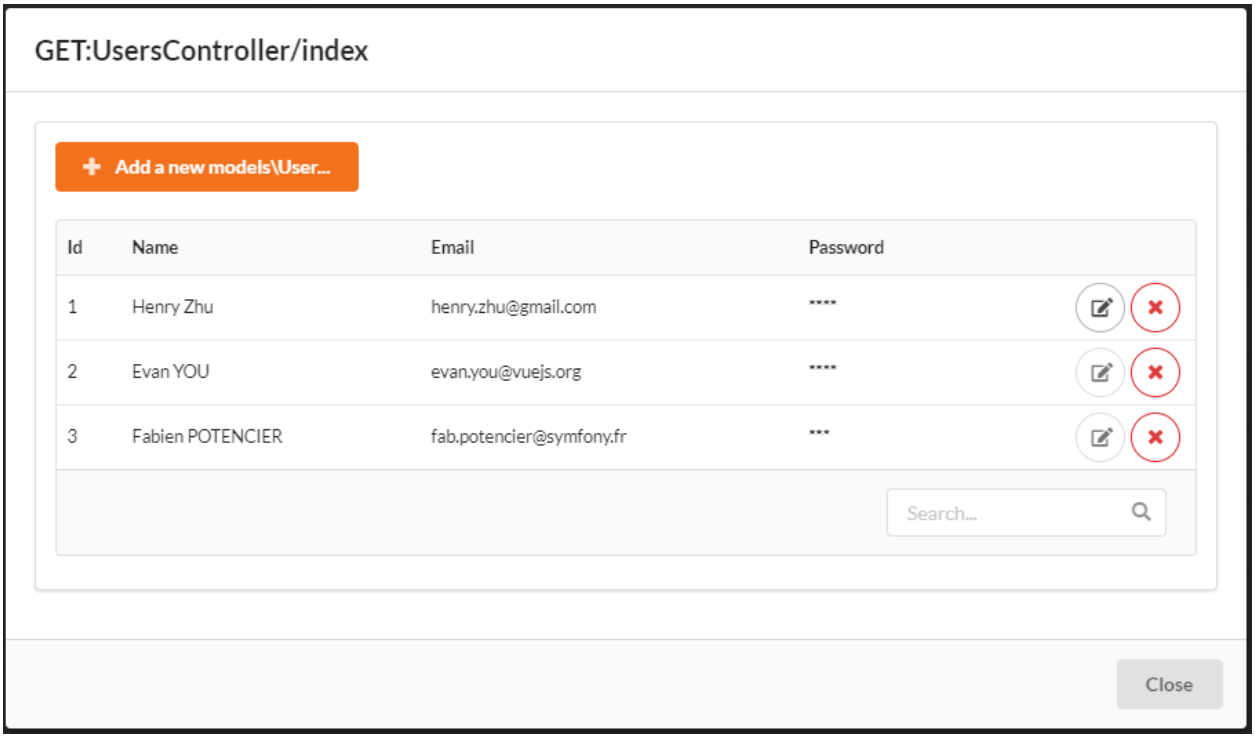
Code source 1 – app/controllers/Products.php

```php
<?php
namespace controllers;

 /**
 * CRUD Controller UsersController
 **/
class UsersController extends \Ubiquity\controllers\crud\CRUDController{

    public function __construct(){
            parent::__construct();
            $this->model= models\User::class;
    }

    public function _getBaseRoute():string {
            return 'UsersController';
    }
}
```

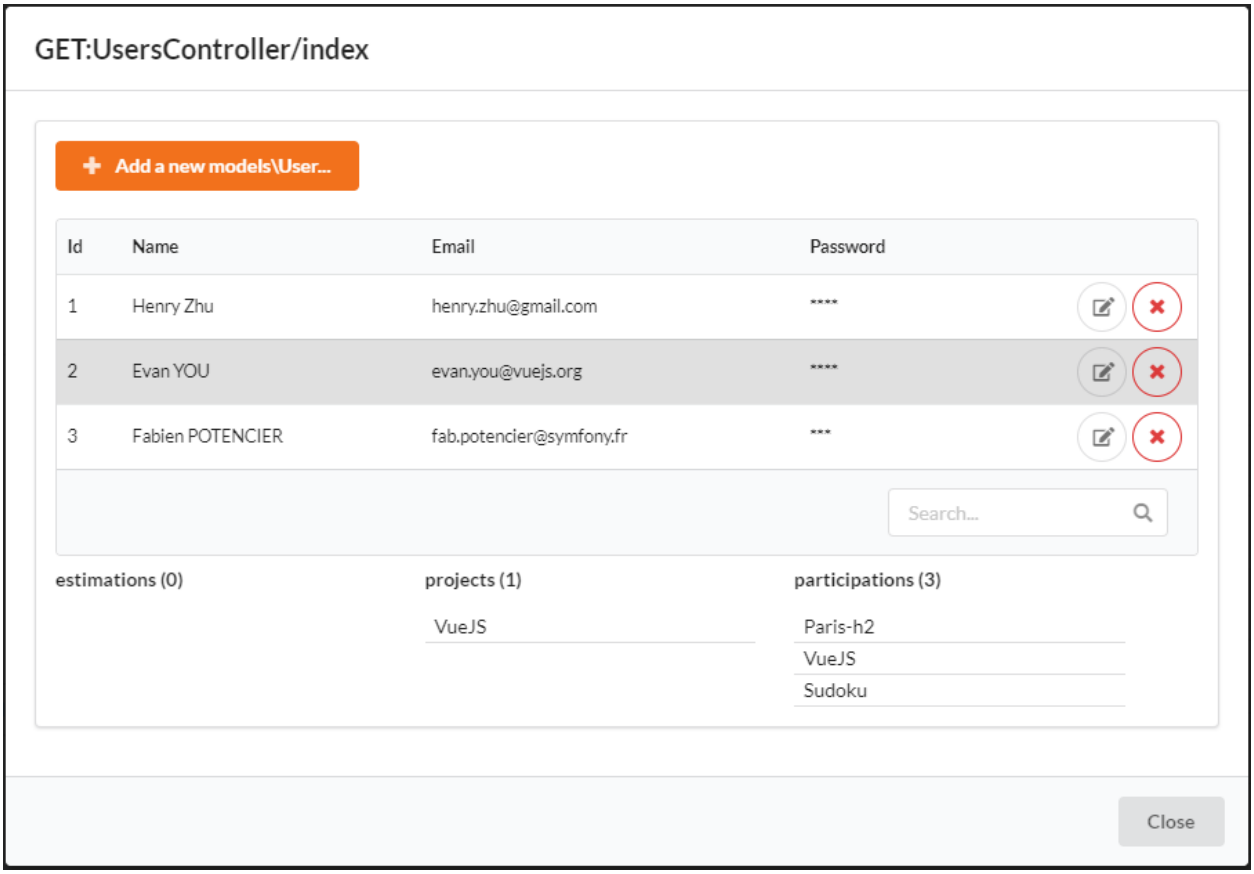
Testez le contrôleur créé en cliquant sur le bouton « get » devant l'action **index** :



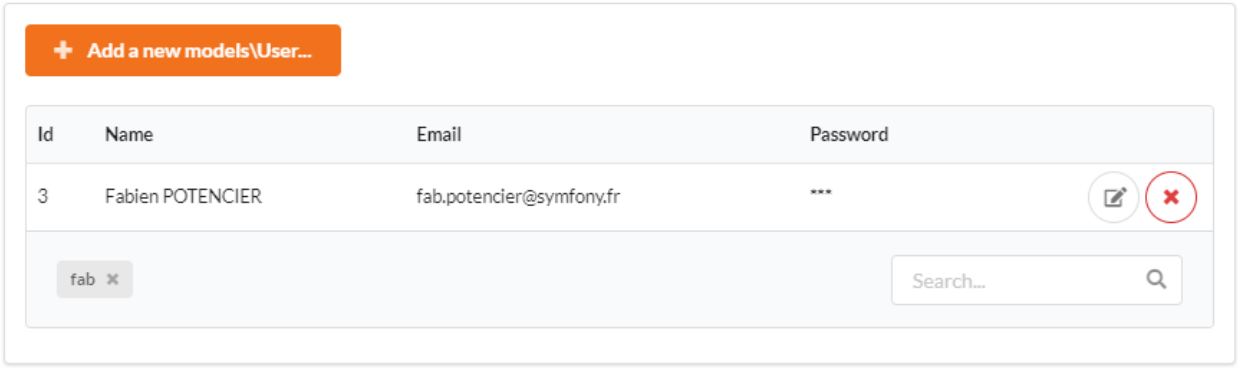
Lecture (action index)



En cliquant sur une ligne de la dataTable (instance), on affiche les objets associés à l'instance (action **details**) :



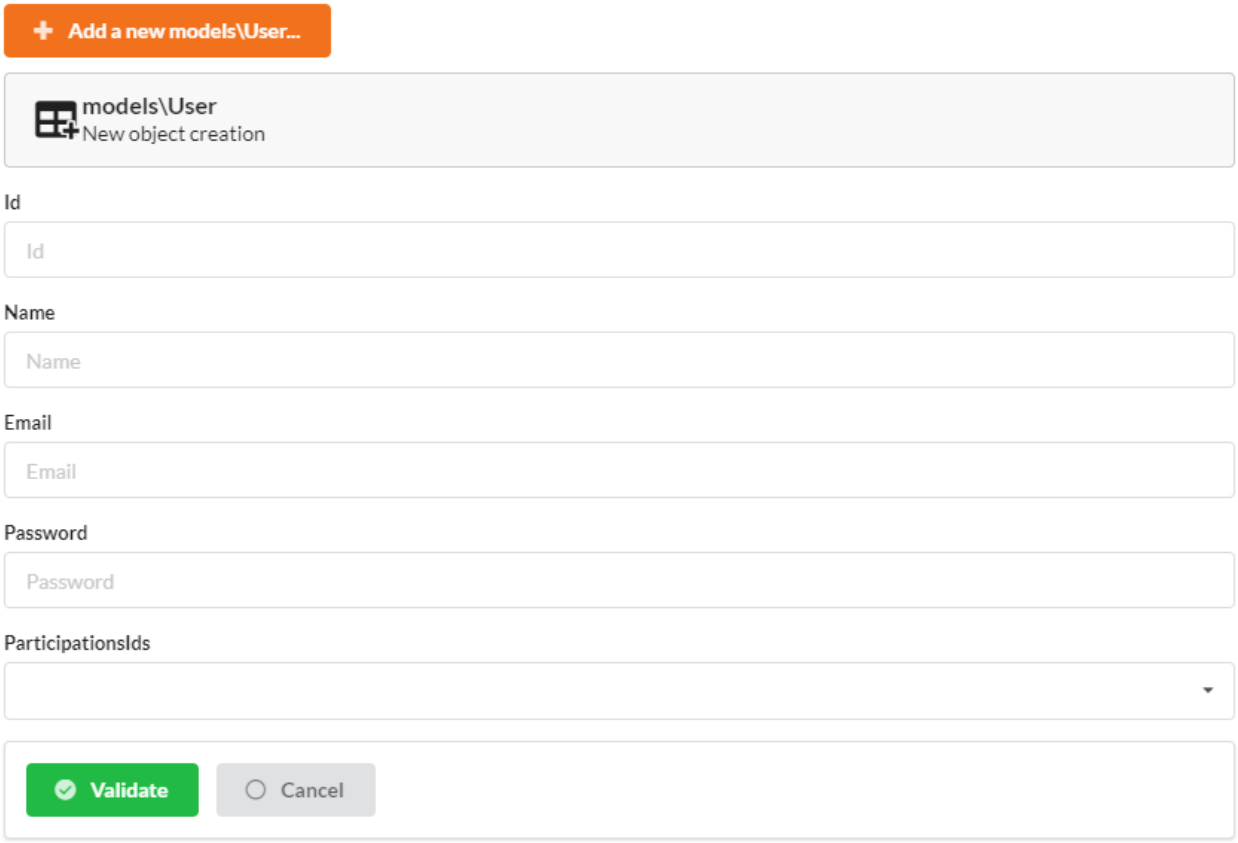
Utilisation de la zone de recherche :



Création (action newModel)

Il est possible de créer une instance en cliquant sur le bouton « add ».

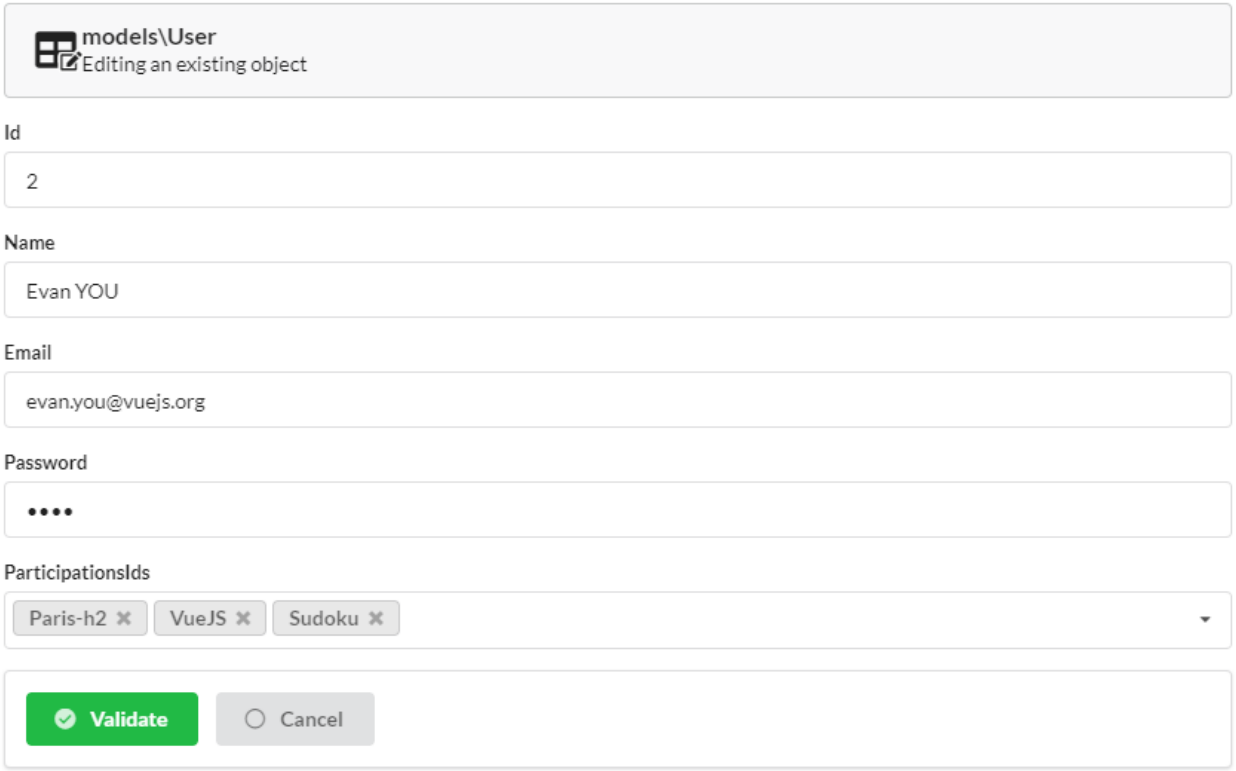
Le formulaire par défaut pour ajouter une instance de User :



Mise à jour (action update)

Le bouton d'édition sur chaque ligne vous permet d'éditer une instance.

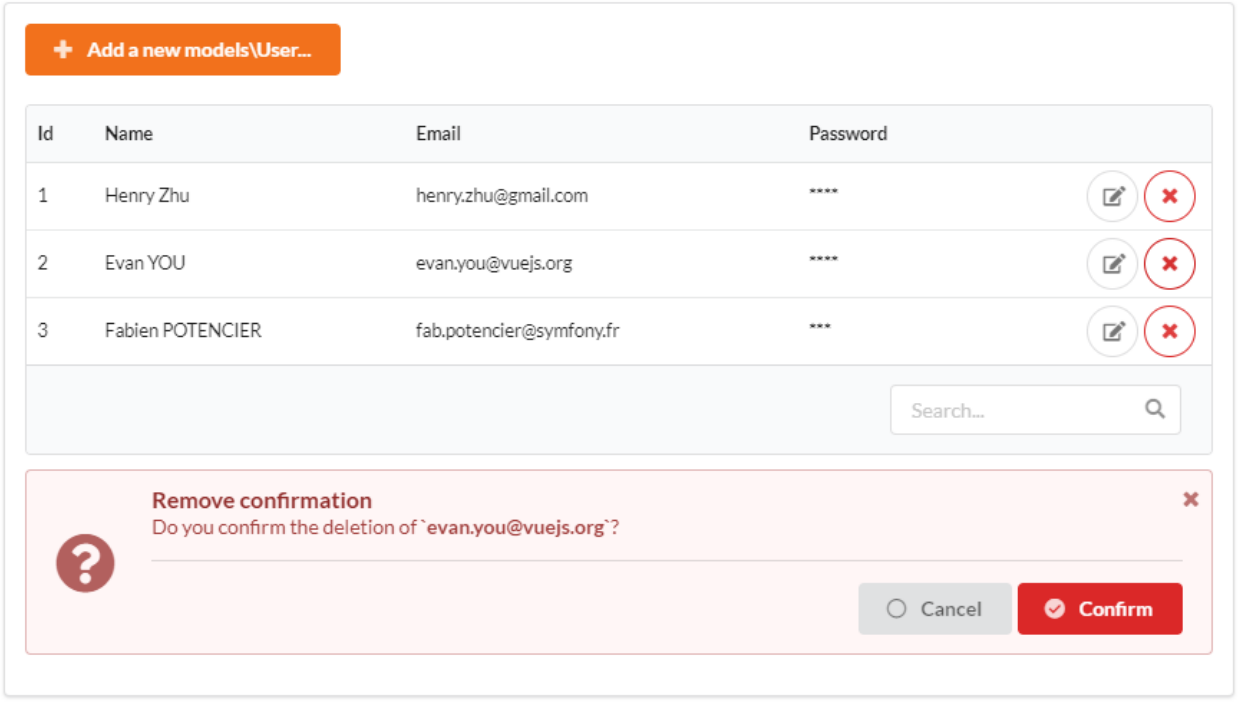
Le formulaire par défaut pour ajouter une instance de User :



Suppression (action delete)

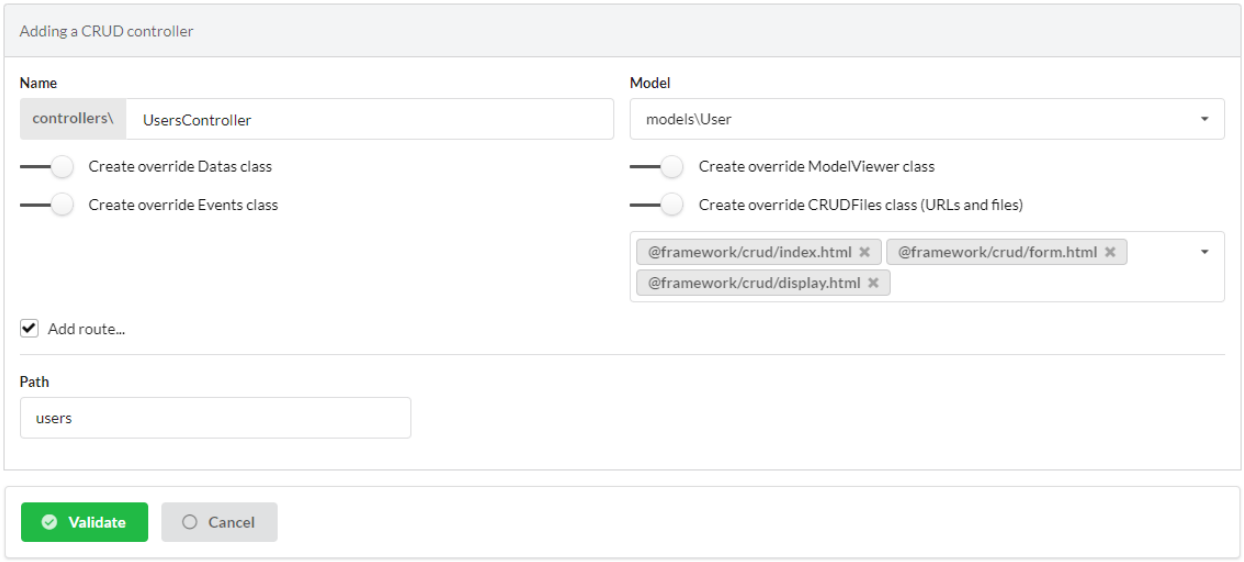
Le bouton de suppression sur chaque ligne vous permet de supprimer une instance.

Affichage du message de confirmation avant la suppression :



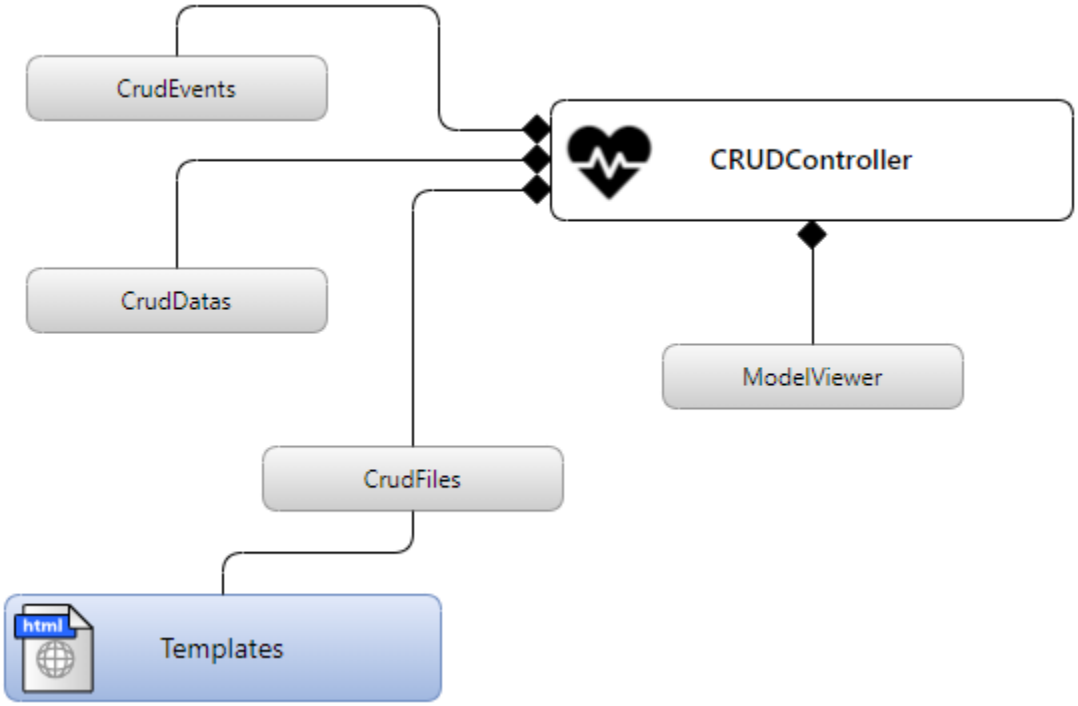
12.1.3 Personnalisation

Créez à nouveau un ResourceCrudController à partir de l'interface d'administration :



Il est désormais possible de personnaliser le module en utilisant la surdéfinition.

Aperçu



Surdéfinition des classes

Méthodes de ResourceCRUDController à surdéfinir

Méthode	Signification	Retour par défaut
routes		
index()	Default page : liste de toutes les instances	
edit($modal= »no », $ids= » »)	Edite une instance	
newModel($modal= »no »)	Crée une instance	
display($modal= »no »,$ids= » »)	Affiche une instance	
delete($ids)	Supprime une instance	
update()	Affiche le résultat de la mise à jour d'une instance	
showDetail($ids)	Affiche les membres associés avec des clés étrangères	
refresh_()	Rafraîchit la zone correspondant à la DataTable (#lv)	
refreshTable($id=null)	//TO COMMENT	

Méthodes de ModelViewer à surdéfinir

Méthode	Signification
	route
getModelDataTable($instances, $model,$totalCount,$page=1)	Crée la dataTable et ajoute son comportement
getDataTableInstance($instances,$model,$totalCount,$page=1)	Crée la dataTable
recordsPerPage($model,$totalCount=0)	Retourne le nombre de lignes à afficher (si null, il n'y a pa
getGroupByFields()	Renvoie un tableau de membres sur lequel on peut effectu
getDataTableRowButtons()	Retourne un tableau de boutons à afficher pour chaque lig
onDataTableRowButton(HtmlButton $bt, ?string $name)	À surdéfinir pour modifier les boutons des lignes de la tab
getCaptions($captions, $className)	Retourne les légendes des en-têtes de colonne
	route
showDetailsOnDataTableClick()	À remplacer pour s'assurer que le détail d'un objet cliqué
onDisplayFkElementListDetails($element,$member,$className,$object)	A modifier pour l'affichage de chaque élément dans un co
getFkHeaderElementDetails($member, $className, $object)	Renvoie l'en-tête d'un objet étranger (issue de ManyToOn
getFkElementDetails($member, $className, $object)	Renvoie un composant permettant d'afficher un objet étra
getFkHeaderListDetails($member, $className, $list)	Renvoie l'en-tête d'une liste d'objets étrangers (oneToMa
getFkListDetails($member, $className, $list)	Renvoie un composant liste permettant d'afficher une coll
	routes
getForm($identifier, $instance)	Renvoie le formulaire d'ajout ou de modification d'un obj
formHasMessage()	Détermine si le formulaire a un titre de type message
getFormModalTitle($instance)	Renvoie le titre de la modale du formulaire
onFormModalButtons($btOkay, $btCancel)	Hook pour modifier les boutons modaux
getFormTitle($form,$instance)	Renvoie un tableau associatif définissant le titre du messa
setFormFieldsComponent(DataForm $form,$fieldTypes)	Définit le composant pour chaque champ
onGenerateFormField($field)	Pour faire quelque chose lorsque $field est généré dans le
isModal($objects, $model)	Condition pour déterminer si le formulaire de modificatio
getFormCaptions($captions, $className, $instance)	Renvoie les légendes des champs de formulaire
	route
getModelDataElement($instance,$model,$modal)	Renvoie un objet DataElement pour l'affichage de l'instan
getElementCaptions($captions, $className, $instance)	Renvoie les légendes des champs du DataElement
	route
onConfirmButtons(HtmlButton $confirmBtn,HtmlButton $cancelBtn)	A surdéfinir pour modifier les boutons de confirmation de

Méthodes CRUDDatas à surdéfinir

Méthode	Signification	Retour par défaut
route **index**		
_getInstancesFilter($model)	Ajoute une condition pour filtrer les instances affichées dans dataTable	1=1
getFieldNames($model)	Retourne les champs à afficher dans l'action **index** pour $model	tous les noms des membres
getSearchFieldNames($model)	Renvoie les champs à utiliser dans les requêtes de recherche.	tous les noms des membres
routes **edit** et **newModel**		
getFormFieldNames($model,$instance)	Renvoie les champs à mettre à jour dans les actions **edit** et **newModel** pour $model.	tous les noms des membres
getManyToOneDatas($fkClass,$instance,$member)	Renvoie une liste (filtrée) d'objets $fkClass à afficher dans une liste html	toutes les instances de $fkClass
getOneToManyDatas($fkClass,$instance,$member)	Renvoie une liste (filtrée) d'objets $fkClass à afficher dans une liste html	toutes les instances de $fkClass
getManyToManyDatas($fkClass,$instance,$member)	Renvoie une liste (filtrée) d'objets $fkClass à afficher dans une liste html	toutes les instances de $fkClass
route **display**		
getElementFieldNames($model)	Retourne les champs à afficher dans l'action **display** pour $model	tous les noms des membres

Méthodes CRUDEvents à surdéfinir

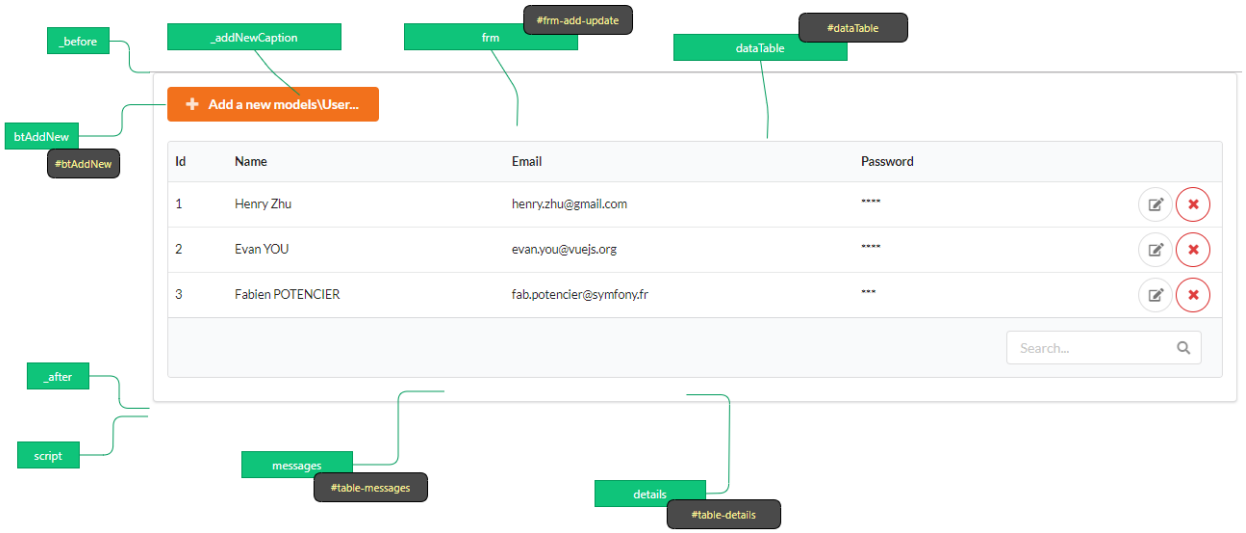
Méthode	Signification	Retour par défaut
route **index**		
onConfDeleteMessage(CRUDMessage $message,$instance)	Renvoie le message de confirmation affiché avant la suppression d'une instance.	CRUDMessage
onSuccessDeleteMessage(CRUDMessage $message,$instance)	Renvoie le message affiché après une suppression	CRUDMessage
onErrorDeleteMessage(CRUDMessage $message,$instance)	Renvoie le message affiché lorsqu'une erreur s'est produite lors de la suppression.	CRUDMessage
routes **edit** et **newModel**		
onSuccessUpdateMessage(CRUDMessage $message)	Renvoie le message affiché lorsqu'une instance est ajoutée ou insérée.	CRUDMessage
onErrorUpdateMessage(CRUDMessage $message)	Renvoie le message affiché lorsqu'une erreur s'est produite lors de la mise à jour ou de l'insertion.	CRUDMessage
onNewInstance(object $instance)	Déclenché après la création d'une nouvelle instance	
onBeforeUpdate(object $instance, bool $isNew)	Déclenché avant la mise à jour de l'instance	
Toutes les routes		
onNotFoundMessage(CRUDMessage $message,$ids)	Renvoie le message affiché lorsqu'une instance n'existe pas.	
onDisplayElements($dataTable,$objects,$refresh)	Déclenché après l'affichage des objets dans le dataTable	

Méthodes CRUDFiles à surdéfinir

Méthode	Signification	Retour par défaut
Fichiers template		
getViewBaseTemplate()	Renvoie le template de base pour toutes les actions Crud si getBaseTemplate renvoie un nom de fichier de template de base.	@framework/crud/baseTemplate.html
getViewIndex()	Retourne le template pour la route **index**.	@framework/crud/index.html
getViewForm()	Retourne le modèle pour les routes **edit** et **newInstance**.	@framework/crud/form.html
getViewDisplay()	Retourne le modèle pour la route **display**.	@framework/crud/display.html
Urls		
getRouteRefresh()	Retourne la route pour rafraîchir la route index.	/**refresh_**
getRouteDetails()	Renvoie la route de la route detail, lorsque l'utilisateur clique sur une ligne du dataTable.	/showDetail
getRouteDelete()	Retourne la route pour la suppression d'une instance.	/delete
getRouteEdit()	Retourne la route pour éditer une instance	/edit
getRouteDisplay()	Retourne la route pour l'affichage d'une instance	/display
getRouteRefreshTable()	Renvoie la route pour rafraîchir le dataTable.	/refreshTable
getDetailClickURL($model)	Retourne la route associée à une instance de clé étrangère dans la liste	« «

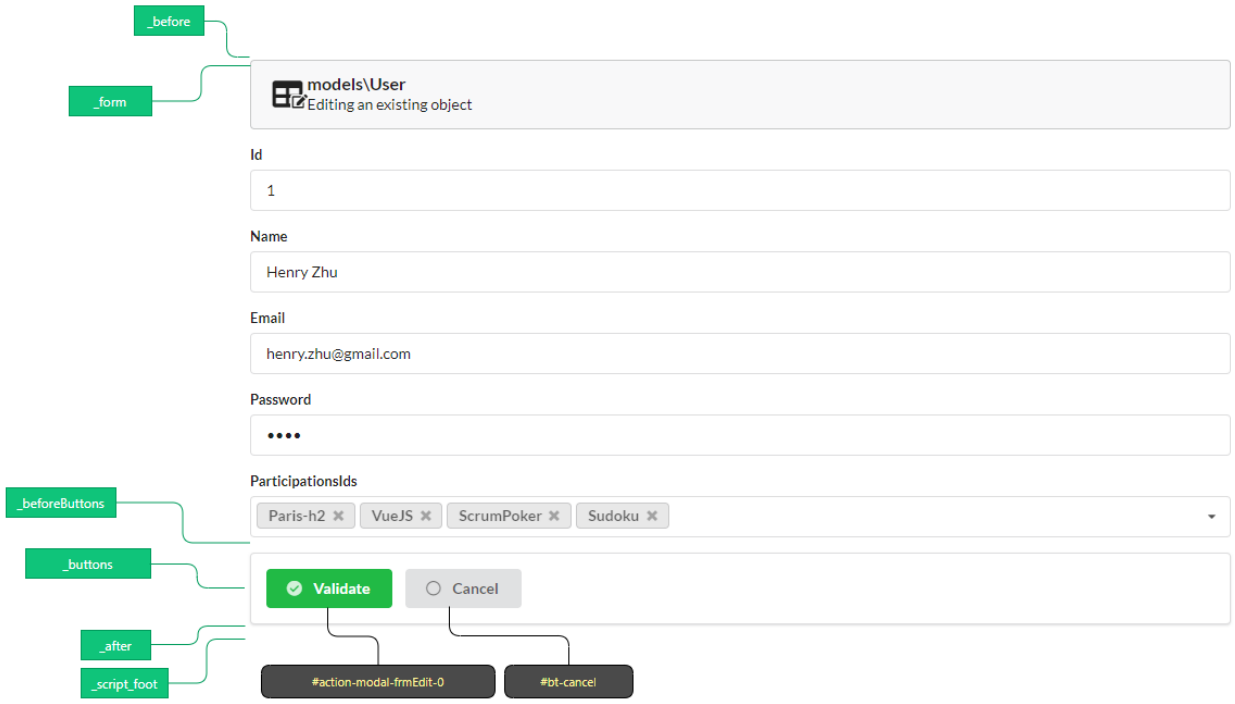
Structure des templates Twig

index.html



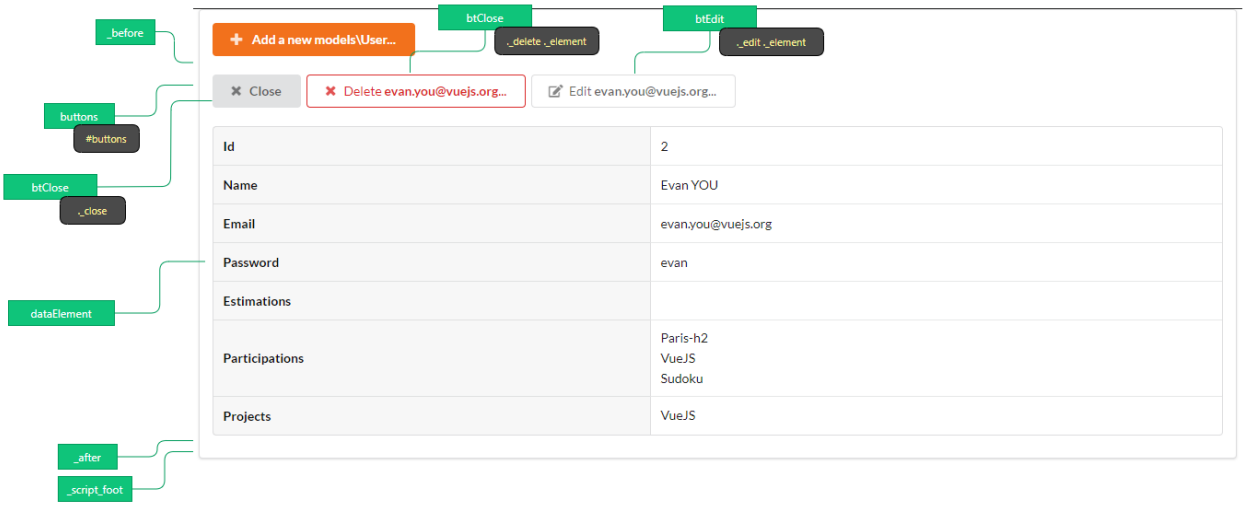
form.html

Affiché dans le bloc **frm**



display.html

Affiché dans le bloc **frm**

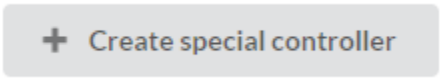


12.2 MultiResourceCrudController

Note : *MultiResourceCRUDController* affiche un index permettant de naviguer entre les CRUDs des modèles.

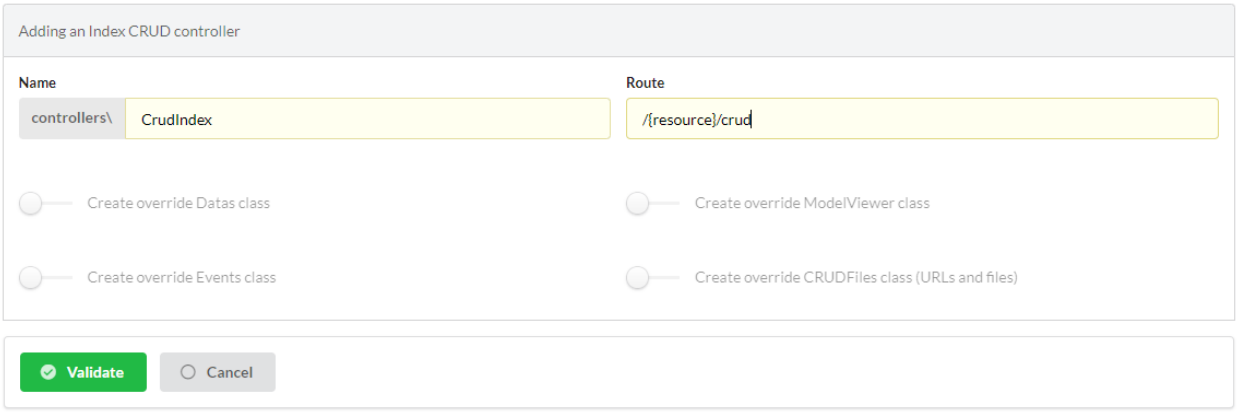
12.2.1 Création

Dans l'interface d'administration (webtools), activez la partie **Controllers**, et choisissez **Index Crud controller** :



Remplissez ensuite le formulaire :

— Entrez le nom du contrôleur
— Le chemin de la route (qui doit contenir la partie variable *{resource}*)
— Cliquez ensuite sur le bouton de validation



12.2.2 Description des caractéristiques

Le contrôleur généré :

Code source 2 – app/controllers/CrudIndex.php

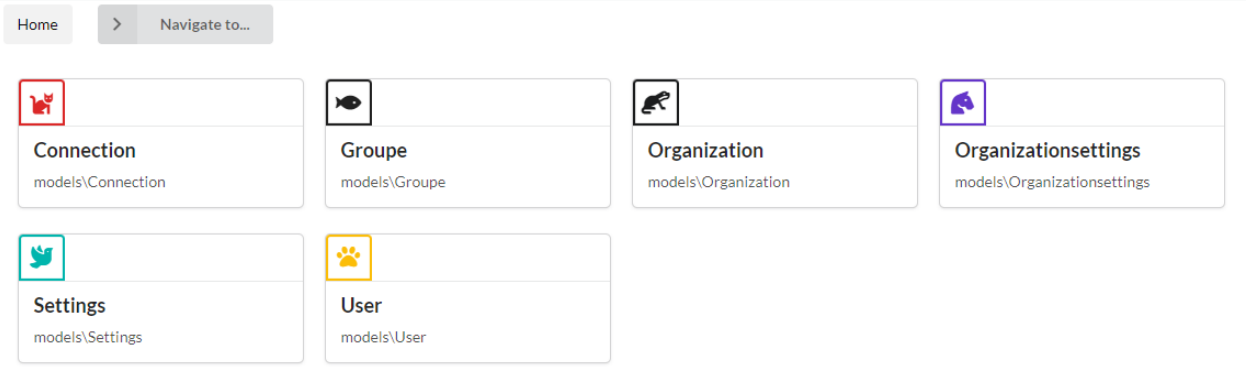
```
1  <?php
2  namespace controllers;
3  use Ubiquity\attributes\items\router\Route;
4
5  #[Route(path: "/{resource}/crud",inherited: true,automated: true)]
6  class CrudIndex extends \Ubiquity\controllers\crud\MultiResourceCRUDController{
7
8              #[Route(name: "crud.index",priority: -1)]
9              public function index() {
10                     parent::index();
11             }
12
```

(suite sur la page suivante)

(suite de la page précédente)

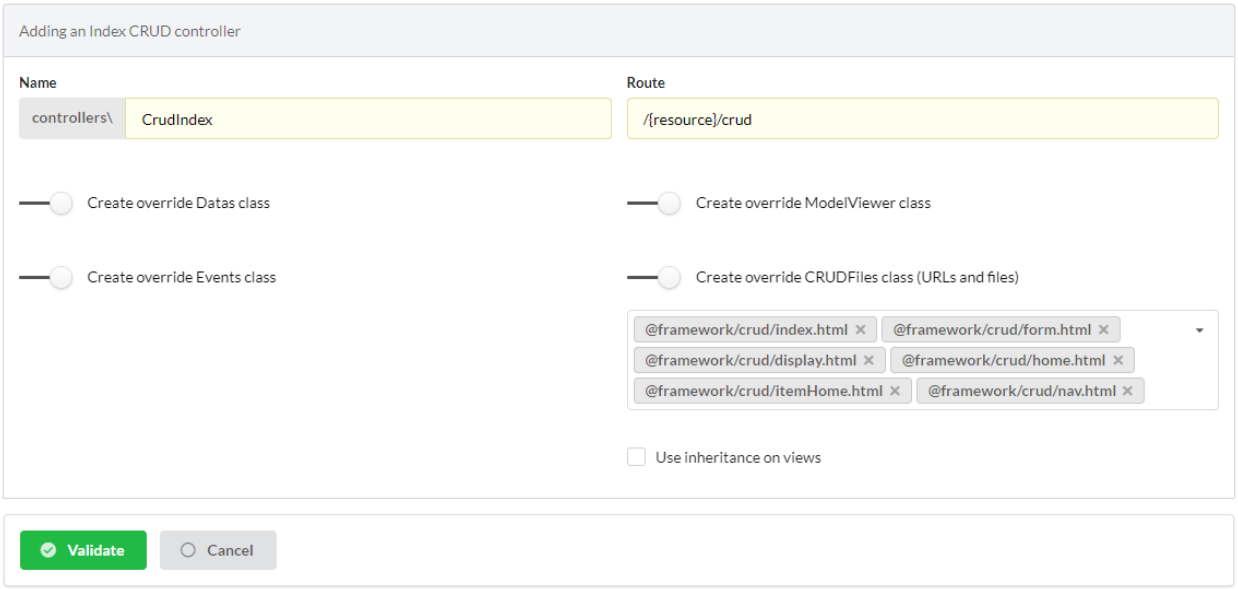
```
        #[Route(path: "#//home/crud",name: "crud.home",priority: 100)]
        public function home(){
                parent::home();
        }

        protected function getIndexType():array {
                return ['four link cards','card'];
        }

        public function _getBaseRoute():string {
                return "/".$this->resource."/crud";
        }

}
```

Testez le contrôleur créé à l'url */home/crud* :



12.2.3 Personnalisation

Créez à nouveau un *MultiResourceCrudController* à partir de l'interface d'administration :



Il est maintenant possible de personnaliser le module en utilisant la surdéfinition comme pour les *ResourceCRUDControllers*.

Classes spécifiques à surdéfinir

Méthodes de MultiResourceCRUDController à surdéfinir

Méthode	Signification	Retour par défaut
routes		
home ()	Page d'accueil : liste des modèles	
—	Toutes les routes de « CRUDController ».	
Evènements		
onRenderView(array &$data)	Avant le rendu de la page d'accueil	
Configuration		
hasNavigation()	Renvoie la valeur True pour l'affichage du menu déroulant de navigation.	True
getIndexModels()	Renvoie la liste des modèles disponibles à afficher	modèles à partir de la base de données par défaut
getIndexModelsDetails()	Retourne un tableau associatif (titre, icon, url) pour chaque modèle.	[]
getIndexDefaultIcon(string $resource)	Renvoie l'icône d'un modèle	Un animal au hasard
getIndexDefaultTitle(string $resource)	Retourne le titre d'un modèle	Le nom de la ressource
getIndexDefaultDesc(string $modelClass)	Renvoie la description d'un modèle	Le nom complet de la classe
getIndexDefaultUrl(string $resource)	Retourne l'url associée à un modèle	Le chemin de la route
getIndexDefaultMeta(string $modelClass)	Retourne la partie méta pour un modèle	
getIndexType()	Définit les classes css du composant de l'index	cards
getModelName()	Renvoie le nom complet du modèle pour $this->resource	A partir du NS par défaut des modèles

Méthodes CRUDFiles à surdéfinir

Méthode	Signification	Retour par défaut
Fichiers template		
getViewHome()	Retourne le template de base pour la vue d'accueil	@framework/crud/home.html
getViewItemHome()	Renvoie le template d'un élément de la route home.	@framework/crud/itemHome.html
getViewNav()	Renvoie le template pour l'affichage des modèles dans une liste déroulante.	@framework/crud/nav.html

Note : Toutes les autres méthodes des classes `CRUDController`, `CRUDFiles`, `CRUDEvents` et `CRUDDatas` peuvent être surdéfinies comme pour le `ResourceCRUDController`.

CHAPITRE 13

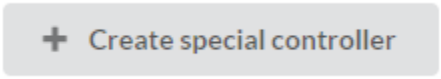
Auth Contrôleurs

Les contrôleurs Auth vous permettent d'effectuer une authentification de base avec :

- se connecter avec un compte
- création de compte
- déconnexion
- contrôleurs avec l'authentification requise

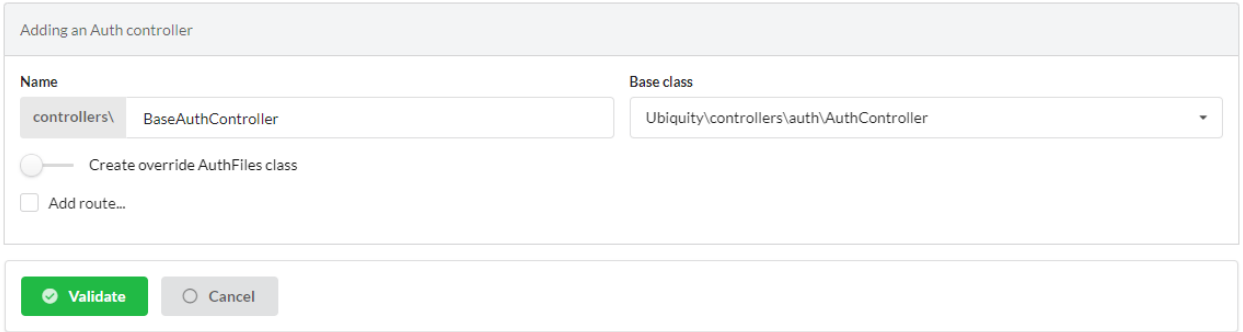
13.1 Création

Dans l'interface d'administration (webtools), activez la partie **Controllers**, et choisissez de créer un **Auth controller** :



Remplissez ensuite le formulaire :

- Entrez le nom du contrôleur (BaseAuthController dans ce cas)



Le contrôleur généré :

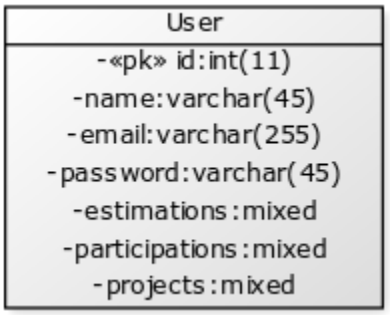
Code source 1 – app/controllers/BaseAuthController.php

```php
/**
 * Auth Controller BaseAuthController
 **/
class BaseAuthController extends \Ubiquity\controllers\auth\AuthController{

	protected function onConnect($connected) {
		$urlParts=$this->getOriginalURL();
		USession::set($this->_getUserSessionKey(), $connected);
		if(isset($urlParts)){
			Startup::forward(implode("/",$urlParts));
		}else{
			//TODO
			//Forwarding to the default controller/action
		}
	}

	protected function _connect() {
		if(URequest::isPost()){
			$email=URequest::post($this->_getLoginInputName());
			$password=URequest::post($this->_getPasswordInputName());
			//TODO
			//Loading from the database the user corresponding to the parameters
			//Checking user creditentials
			//Returning the user
		}
		return;
	}

	/**
	 * {@inheritDoc}
	 * @see \Ubiquity\controllers\auth\AuthController::isValidUser()
	 */
	public function _isValidUser($action=null): bool {
		return USession::exists($this->_getUserSessionKey());
	}

	public function _getBaseRoute(): string {
		return 'BaseAuthController';
	}
}
```

13.2 Implémentation de l'authentification

Exemple d'implémentation avec l'interface d'administration : Nous allons ajouter un contrôle d'authentification sur l'interface d'administration.

L'authentification est basée sur la vérification du couple email/password sur le modèle **User** :



13.2.1 Modification de BaseAuthController

Code source 2 – app/controllers/BaseAuthController.php

```
1  /**
2  * Auth Controller BaseAuthController
3  **/
4  class BaseAuthController extends \Ubiquity\controllers\auth\AuthController{
5
6      protected function onConnect($connected) {
7              $urlParts=$this->getOriginalURL();
8              USession::set($this->_getUserSessionKey(), $connected);
9              if(isset($urlParts)){
10                     Startup::forward(implode("/",$urlParts));
11             }else{
12                     Startup::forward("admin");
13             }
14     }
15
16     protected function _connect() {
17             if(URequest::isPost()){
18                     $email=URequest::post($this->_getLoginInputName());
19                     $password=URequest::post($this->_getPasswordInputName());
20                     return DAO::uGetOne(User::class, "email=? and password= ?",false,[
↪$email,$password]);
21             }
22             return;
23     }
24
25     /**
26      * {@inheritDoc}
27      * @see \Ubiquity\controllers\auth\AuthController::isValidUser()
28      */
29     public function _isValidUser($action=null): bool {
```

(suite sur la page suivante)

(suite de la page précédente)

```
30              return USession::exists($this->_getUserSessionKey());
31      }
32
33      public function _getBaseRoute(): string {
34              return 'BaseAuthController';
35      }
36      /**
37       * {@inheritDoc}
38       * @see \Ubiquity\controllers\auth\AuthController::_getLoginInputName()
39       */
40      public function _getLoginInputName(): string {
41              return "email";
42      }
43  }
```

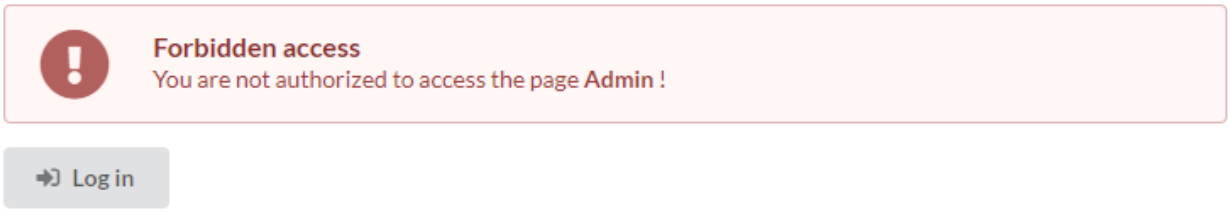
13.2.2 Modification du contrôleur Admin

Modifiez le contrôleur d'administration pour utiliser BaseAuthController :

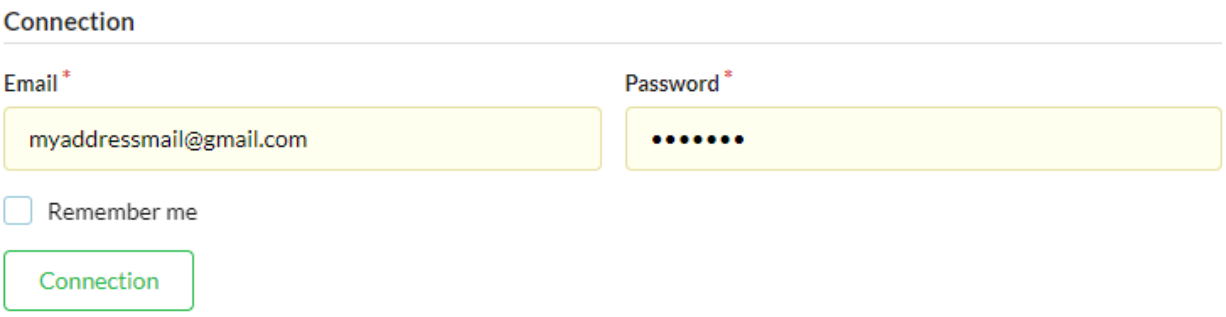
Code source 3 – app/controllers/Admin.php

```
1  class Admin extends UbiquityMyAdminBaseController{
2      use WithAuthTrait;
3      protected function getAuthController(): AuthController {
4              return $this->_auth ??= new BaseAuthController($this);
5      }
6  }
```

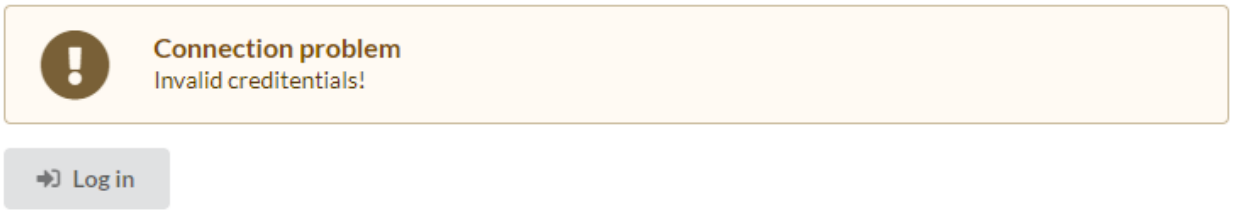
Tester l'interface d'administration à l'adresse **/admin** :



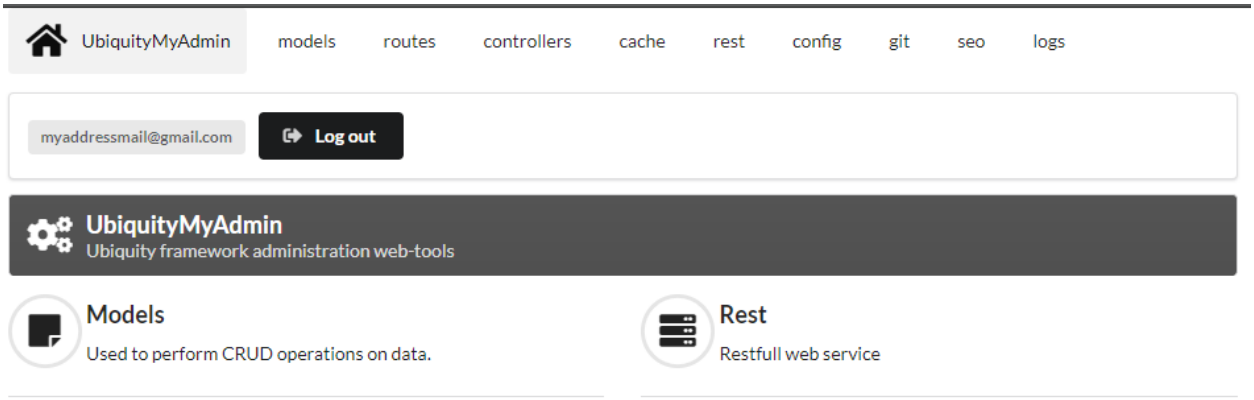
Après avoir cliqué sur **login** :



Si les données d'authentification sont invalides :



Si les données d'authentification sont valides :



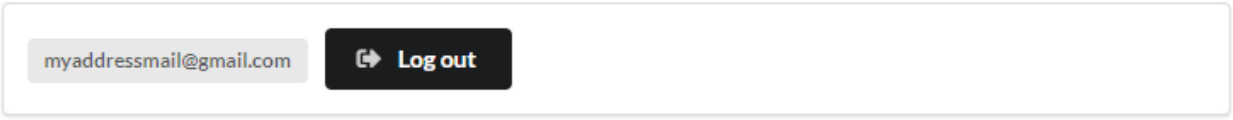
13.2.3 Attachement de la zone info-user

Modifier le contrôleur **BaseAuthController** :

Code source 4 – app/controllers/BaseAuthController.php

```
1  /**
2   * Auth Controller BaseAuthController
3   **/
4  class BaseAuthController extends \Ubiquity\controllers\auth\AuthController{
5  ...
6      public function _displayInfoAsString(): bool {
7              return true;
8      }
9  }
```

La zone **_userInfo** est désormais présente sur toutes les pages de l'administration :



Elle peut être affichée dans n'importe quelle vue twig :

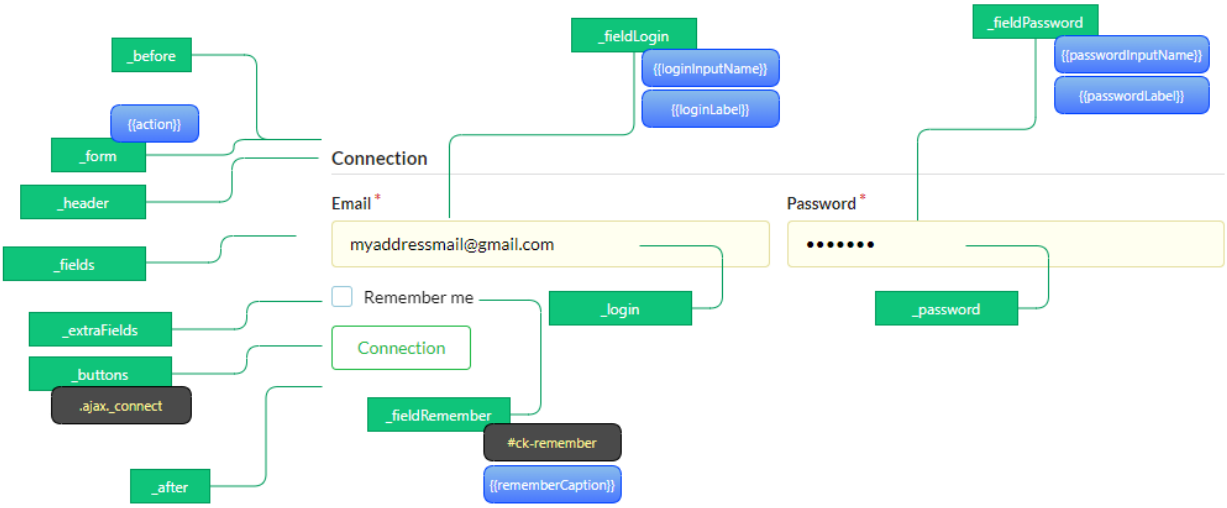
```
{{ _userInfo | raw }}
```

13.3 Description des caractéristiques

13.3.1 Personnalisation des templates

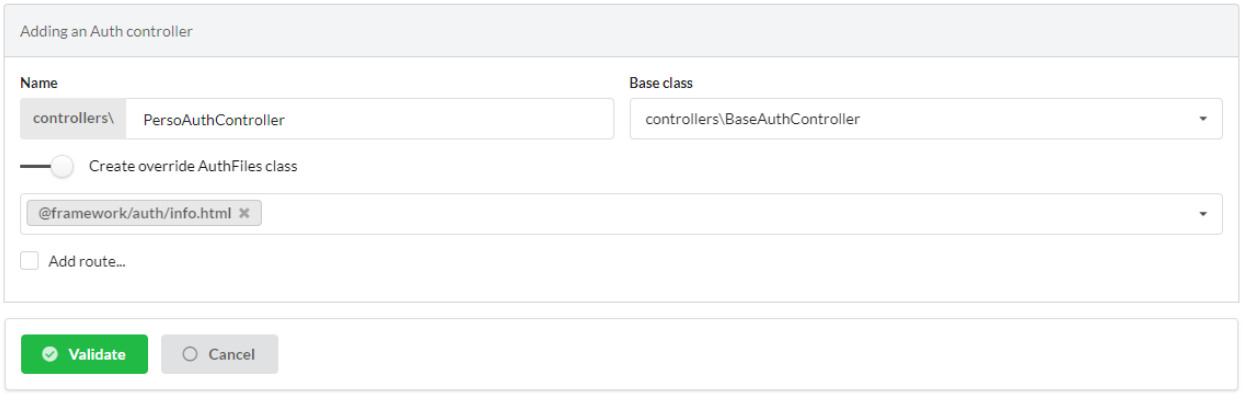
template index.html

Le template index.html gère la connexion :



Exemple avec la zone **_userInfo** :

Créez un nouveau AuthController nommé **PersoAuthController** :



Modifiez le template **app/views/PersoAuthController/info.html**.

Code source 5 – app/views/PersoAuthController/info.html

```
{% extends "@framework/auth/info.html" %}
{% block _before %}
    <div class="ui tertiary inverted red segment">
{% endblock %}
{% block _userInfo %}
    {{ parent() }}
{% endblock %}
{% block _logoutButton %}
```

(suite sur la page suivante)

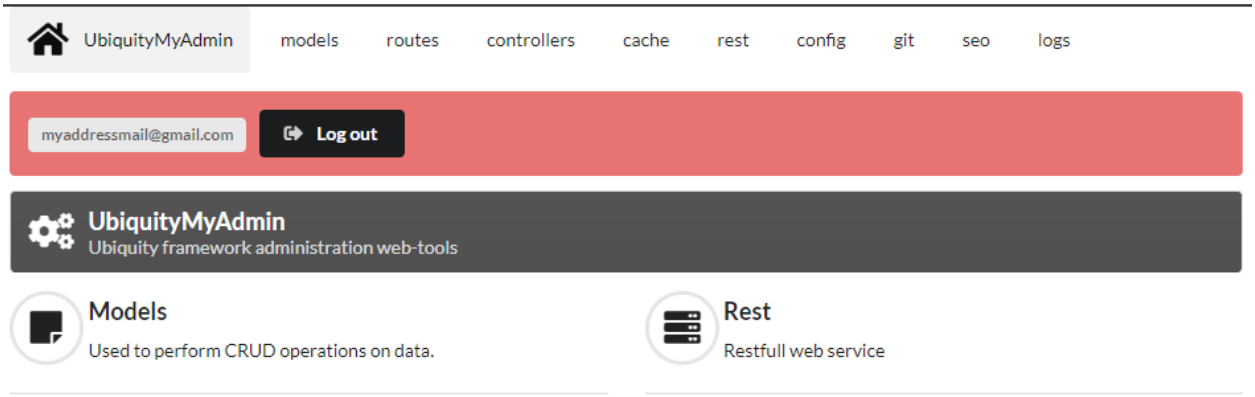
(suite de la page précédente)

```
    {{ parent() }}
{% endblock %}
{% block _logoutCaption %}
    {{ parent() }}
{% endblock %}
{% block _loginButton %}
    {{ parent() }}
{% endblock %}
{% block _loginCaption %}
    {{ parent() }}
{% endblock %}
{% block _after %}
            </div>
{% endblock %}
```

Changez le contrôleur AuthController **Admin** :

Code source 6 – app/controllers/Admin.php

```
class Admin extends UbiquityMyAdminBaseController{
    use WithAuthTrait;
    protected function getAuthController(): AuthController {
            return $this->_auth ??= new PersoAuthController($this);
    }
}
```



13.3.2 Personnalisation des messages

Code source 7 – app/controllers/PersoAuthController.php

```
class PersoAuthController extends \controllers\BaseAuth{
...
 /**
  * {@inheritDoc}
  * @see \Ubiquity\controllers\auth\AuthController::badLoginMessage()
  */
 protected function badLoginMessage(\Ubiquity\utils\flash\FlashMessage $fMessage) {
     $fMessage->setTitle("Erreur d'authentification");
```

(suite sur la page suivante)

(suite de la page précédente)

```
        $fMessage->setContent("Login ou mot de passe incorrects !");
        $this->_setLoginCaption("Essayer à nouveau");

    }
...
}
```

13.3.3 Auto-vérification

Code source 8 – app/controllers/PersoAuthController.php

```
class PersoAuthController extends \controllers\BaseAuth{
...
    /**
     * {@inheritDoc}
     * @see \Ubiquity\controllers\auth\AuthController::_checkConnectionTimeout()
     */
    public function _checkConnectionTimeout() {
        return 10000;
    }
...
}
```

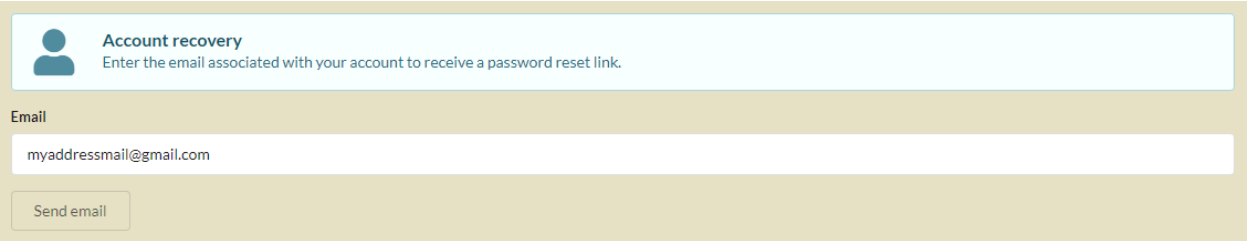
13.3.4 Limitation du nombre de tentatives

Code source 9 – app/controllers/PersoAuthController.php

```php
class PersoAuthController extends \controllers\BaseAuth{
...
 /**
  * {@inheritDoc}
  * @see \Ubiquity\controllers\auth\AuthController::attemptsNumber()
  */
 protected function attemptsNumber(): int {
     return 3;
 }
...
}
```

13.3.5 Récupération de compte

La récupération de compte se fait en réinitialisant le mot de passe du compte. Un e-mail de réinitialisation du mot de passe est envoyé, à une adresse e-mail correspondant à un compte actif.



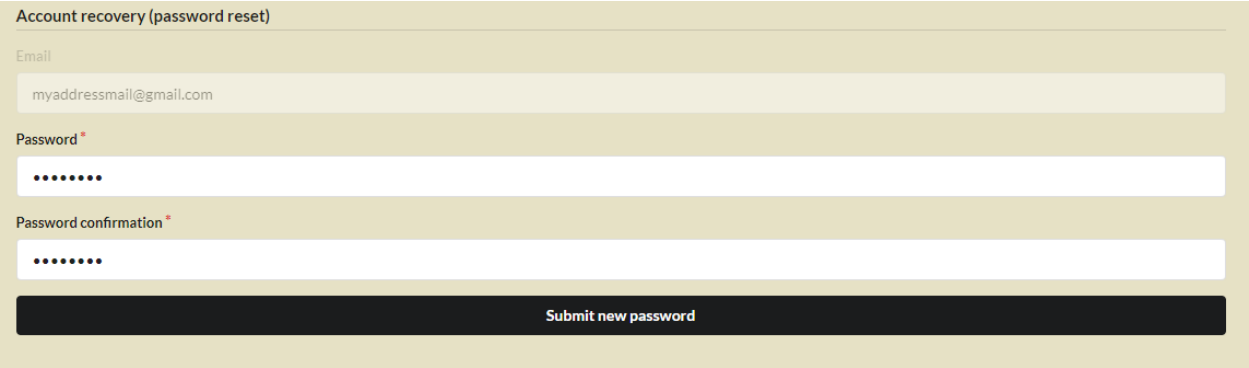
Code source 10 – app/controllers/PersoAuthController.php

```php
class PersoAuthController extends \controllers\BaseAuth{
...
 protected function hasAccountRecovery():bool{
     return true;
 }

 protected function _sendEmailAccountRecovery(string $email,string $validationURL,string
↪$expire):bool {
     MailerManager::start();
     $mail=new AuthAccountRecoveryMail();
     $mail->to($connected->getEmail());
     $mail->setUrl($validationURL);
     $mail->setExpire($expire);
     return MailerManager::send($mail);
 }

 protected function passwordResetAction(string $email,string $newPasswordHash):bool {
     //To implement for modifying the user password
 }

 protected function isValidEmailForRecovery(string $email):bool {
     //To implement: return true if a valid account match with this email
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }
}
```



Note : Par défaut, le lien ne peut être utilisé que sur la même machine, dans une période de temps prédéterminée (qui peut être modifiée en surchargeant la méthode `accountRecoveryDuration`).

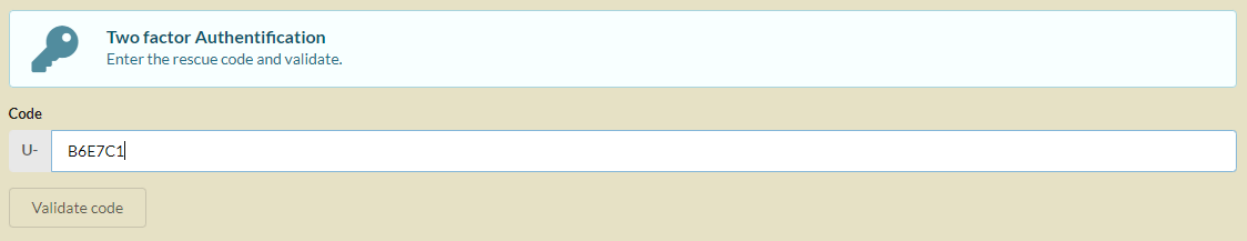
13.3.6 Activation de MFA/2FA

L'authentification multi-facteurs peut être activée de manière conditionnelle, sur la base des informations de l'utilisateur préalablement connecté.

Note : La phase 2 de l'authentification est réalisée dans l'exemple ci-dessous en envoyant un code aléatoire par email. La classe AuthMailerClass est disponible dans le paquet `Ubiquity-mailer`.

Code source 11 – app/controllers/PersoAuthController.php

```
class PersoAuthController extends \controllers\BaseAuth{
...
 /**
  * {@inheritDoc}
  * @see \Ubiquity\controllers\auth\AuthController::has2FA()
  */
 protected function has2FA($accountValue=null):bool{
     return true;
 }

 protected function _send2FACode(string $code, $connected):void {
     MailerManager::start();
     $mail=new AuthMailerClass();
     $mail->to($connected->getEmail());
     $mail->setCode($code);
     MailerManager::send($mail);
 }
...
}
```

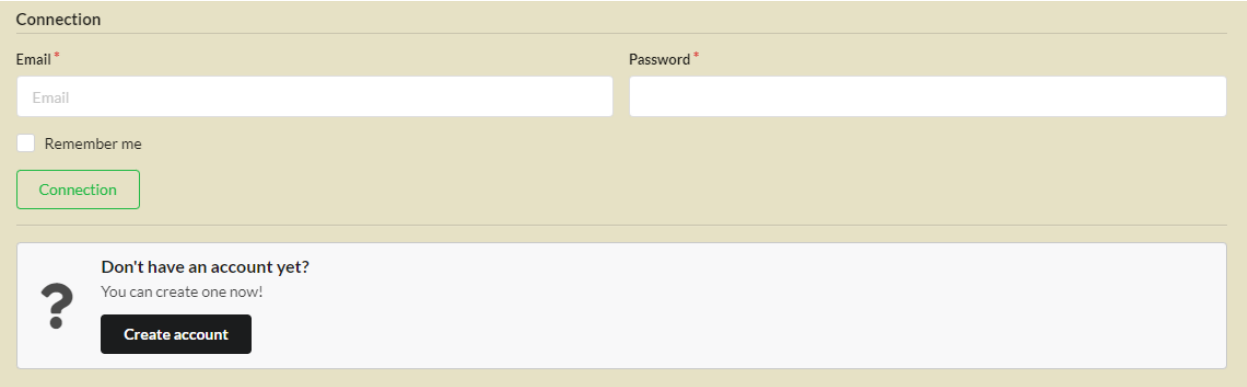


Note : Il est possible de personnaliser la création du code généré, ainsi que le préfixe utilisé. L'exemple ci-dessous est implémenté avec la bibliothèque `robthree/twofactorauth`.

```
protected function generate2FACode():string{
        $tfa=new TwoFactorAuth();
        return $tfa->createSecret();
}

protected function towFACodePrefix():string{
        return 'U-';
}
```

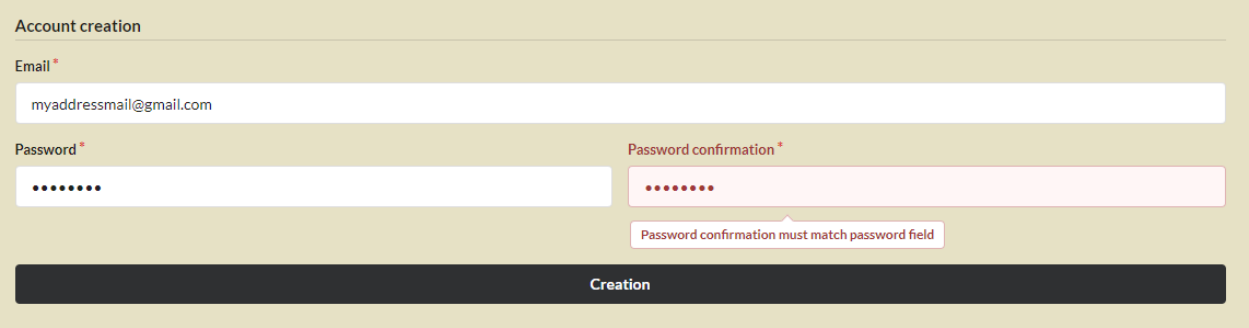
13.3.7 Création de compte

L'activation de la création du compte est également optionnelle :



Code source 12 – app/controllers/PersoAuthController.php

```php
class PersoAuthController extends \controllers\BaseAuth{
...
 protected function hasAccountCreation():bool{
     return true;
 }
...
}
```

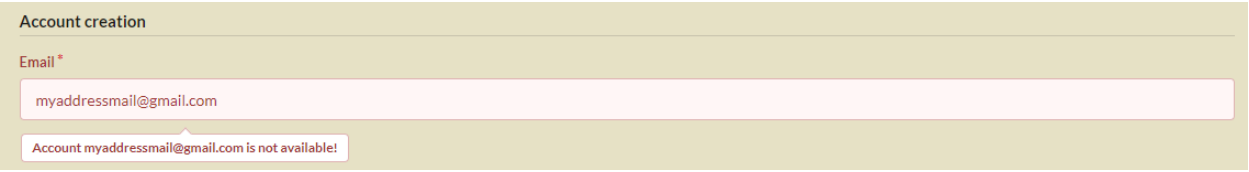


Dans ce cas, la méthode _create doit être surchargée afin de créer le compte :

```php
protected function _create(string $login, string $password): ?bool {
        if(!DAO::exists(User::class,'login= ?',[$login])){
                $user=new User();
                $user->setLogin($login);
                $user->setPassword($password);
                URequest::setValuesToObject($user);//for the others params in the POST.
                return DAO::insert($user);
        }
        return false;
}
```

Vous pouvez vérifier la validité/disponibilité du login avant de valider le formulaire de création de compte :

```php
protected function newAccountCreationRule(string $accountName): ?bool {
        return !DAO::exists(User::class,'login= ?',[$accountName]);
}
```



Une action de confirmation (vérification par courriel) peut être demandée à l'utilisateur :

```php
protected function hasEmailValidation(): bool {
        return true;
}

protected function _sendEmailValidation(string $email,string $validationURL,string
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪$expire):void {
        MailerManager::start();
        $mail=new AuthEmailValidationMail();
        $mail->to($connected->getEmail());
        $mail->setUrl($validationURL);
        $mail->setExpire($expire);
        MailerManager::send($mail);
}
```

Note : Il est possible de personnaliser ces parties en surchargeant les méthodes associées, ou en modifiant les interfaces dans les modèles concernés.

CHAPITRE 14

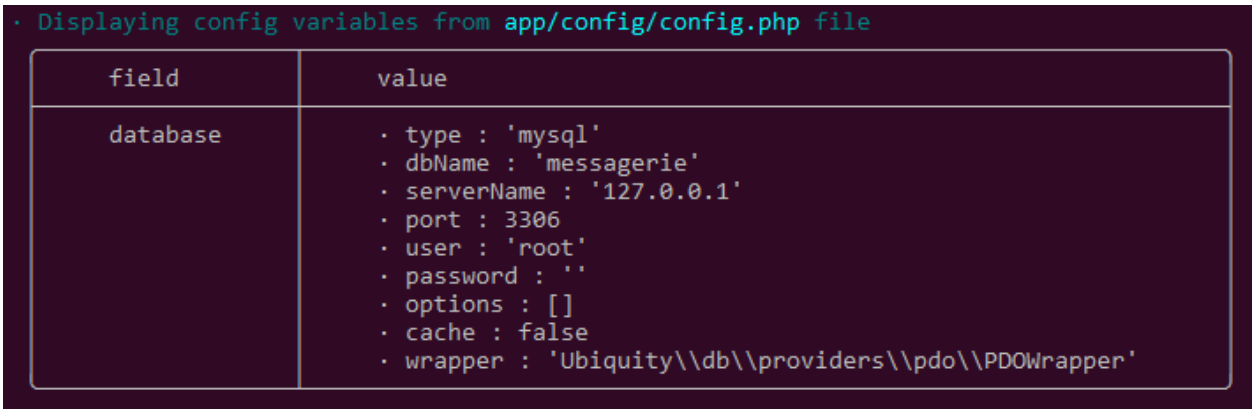
Base de données

La classe **DAO** est en charge des opérations de chargement et de persistance des modèles :

14.1 Connexion à la base de données

Vérifiez que les paramètres de connexion à la base de données sont correctement renseignés dans le fichier de configuration :

```
Ubiquity config -f=database
```



14.1.1 Connexion transparente

Depuis Ubiquity 2.3.0, la connexion à la base de données se fait automatiquement à la première requête :

```
use Ubiquity\orm\DAO;

$firstUser=DAO::getById(User::class,1);//Automatically start the database
```

C'est le cas de toutes les méthodes de la classe **DAO** utilisées pour effectuer des opérations CRUD.

14.1.2 Connexion explicite

Dans certains cas, cependant, il peut être utile d'établir une connexion explicite à la base de données, notamment pour vérifier la connexion.

```
use Ubiquity\orm\DAO;
use Ubiquity\controllers\Startup;
...
try{
    $config=\Ubiquity\controllers\Startup::getConfig();
    DAO::startDatabase($config);
    $users=DAO::getAll(User::class,'');
}catch(Exception $e){
    echo $e->getMessage();
}
```

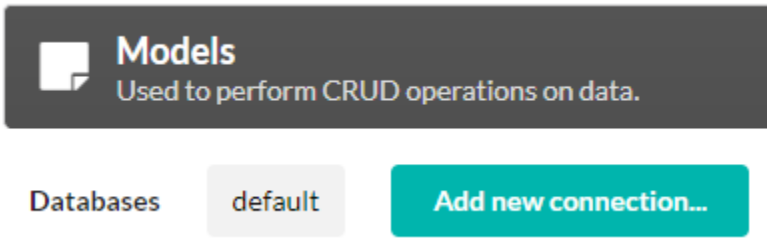
14.2 Connexions multiples

14.2.1 Ajout d'une nouvelle connexion

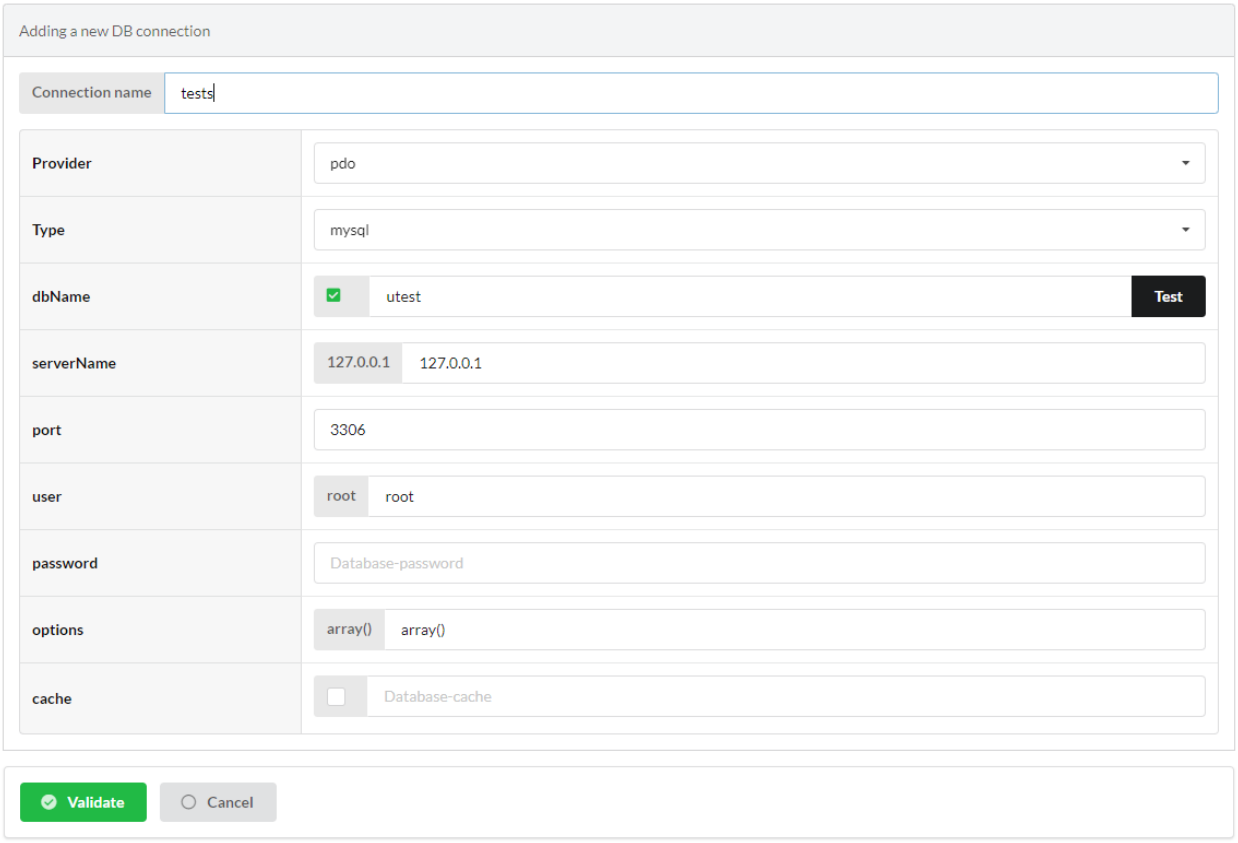
Ubiquity vous permet de gérer plusieurs connexions à des bases de données.

Avec les webtools

Dans la partie **Modèles**, choisissez le bouton **Add new connection** :



Définir les paramètres de configuration de la connexion :



Générer des modèles pour la nouvelle connexion : Les modèles générés incluent l'annotation `@database` ou l'attribut `Database` mentionnant leur lien avec la connexion.

Attributs

```php
<?php
namespace models\tests;
use Ubiquity\attributes\items\Database;
use Ubiquity\attributes\items\Table;

#[Database('tests')]
#[Table('groupe')]
class Groupe{
    ...
}
```

Annotations

```php
<?php
namespace models\tests;
/**
 * @database('tests')
 * @table('groupe')
 */
class Groupe{
    ...
}
```

Les modèles sont générés dans un sous-dossier de `models`.

Avec plusieurs connexions, n'oubliez pas d'ajouter la ligne suivante au fichier `services.php` :

```
\Ubiquity\orm\DAO::start();
```

La méthode `start` effectue la correspondance entre chaque modèle et la connexion associée.

CHAPITRE 15

Génération des modèles

15.1 A partir d'une base de données existante

— avec les devtools
— avec les webtools

15.2 A partir de zéro

— création des modèles avec les devtools
— migrations

CHAPITRE 16

ORM

Note : si vous voulez générer automatiquement les modèles, consultez la partie *génération des modèles* .

Une classe de type modèle est juste un bon vieil objet php sans héritage. Les modèles sont situés par défaut dans le dossier **appmodels**. Le mappage relationnel d'objets (ORM) repose sur les annotations ou les attributs des membres (depuis PHP8) dans la classe du modèle.

16.1 Définition de modèles

16.1.1 Un modèle basique

— Un modèle doit définir sa clé primaire en utilisant l'annotation **@id** sur les membres concernés.
— Les membres sérialisés doivent avoir des getters et setters.
— Sans autre annotation, une classe correspond à une table du même nom dans la base de données, chaque membre correspond à un champ de cette table.

Attributs

Code source 1 – app/models/User.php

```
1  namespace models;
2
3  use Ubiquity\attributes\items\Id;
4
5  class User{
6
7      #[Id]
8      private $id;
9
10     private $firstname;
```

(suite sur la page suivante)

(suite de la page précédente)

```
    public function getFirstname(){
        return $this->firstname;
    }
    public function setFirstname($firstname){
        $this->firstname=$firstname;
    }
}
```

Annotations

Code source 2 – app/models/User.php

```
namespace models;

class User{
    /**
     * @id
     */
    private $id;

    private $firstname;

    public function getFirstname(){
        return $this->firstname;
    }
    public function setFirstname($firstname){
        $this->firstname=$firstname;
    }
}
```

16.1.2 Mappage

Table->Classe

Si le nom de la table est différent du nom de la classe, l'annotation **@table** permet de préciser le nom de la table.

Attributs

Code source 3 – app/models/User.php

```
namespace models;

use Ubiquity\attributes\items\Table;
use Ubiquity\attributes\items\Id;

#[Table('user')]
class User{

    #[Id]
    private $id;

```

(suite sur la page suivante)

(suite de la page précédente)

```
12      private $firstname;
13
14      public function getFirstname(){
15          return $this->firstname;
16      }
17      public function setFirstname($firstname){
18          $this->firstname=$firstname;
19      }
20  }
```

Annotations

Code source 4 – app/models/User.php

```
1   namespace models;
2
3   /**
4    * @table("name"=>"user")
5    */
6   class User{
7       /**
8        * @id
9        */
10      private $id;
11
12      private $firstname;
13
14      public function getFirstname(){
15          return $this->firstname;
16      }
17      public function setFirstname($firstname){
18          $this->firstname=$firstname;
19      }
20  }
```

Champ->Membre

Si le nom d'un champ est différent du nom du membre de la classe associé, l'annotation **@column** permet de spécifier un nom de champ différent.

Attributs

Code source 5 – app/models/User.php

```
1   namespace models;
2
3   use Ubiquity\attributes\items\Table;
4   use Ubiquity\attributes\items\Id;
5   use Ubiquity\attributes\items\Column;
6
7   #[Table('user')
8   class User{
```

(suite sur la page suivante)

(suite de la page précédente)

```
9
10      #[Id]
11      private $id;
12
13      #[Column('column_name')]
14      private $firstname;
15
16      public function getFirstname(){
17          return $this->firstname;
18      }
19      public function setFirstname($firstname){
20          $this->firstname=$firstname;
21      }
22  }
```

Annotations

Code source 6 – app/models/User.php

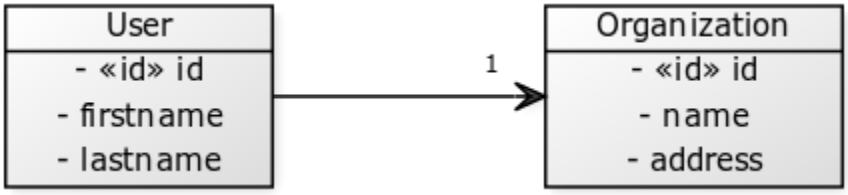
```
1   namespace models;
2
3   /**
4    * @table("user")
5    */
6   class User{
7       /**
8        * @id
9        */
10      private $id;
11
12      /**
13       * column("user_name")
14       */
15      private $firstname;
16
17      public function getFirstname(){
18          return $this->firstname;
19      }
20      public function setFirstname($firstname){
21          $this->firstname=$firstname;
22      }
23  }
```

16.1.3 Associations

Note : Convention de nommage Les noms des champs des clés étrangères sont constitués du nom de la clé primaire de la table référencée suivi du nom de la table référencée dont la première lettre est en majuscule. Exemple : `idUser` pour la table `user` dont la clé primaire est `id`.

ManyToOne

Un **utilisateur** appartient à une **organisation** :



Attributs

Code source 7 – app/models/User.php

```php
namespace models;

use Ubiquity\attributes\items\ManyToOne;
use Ubiquity\attributes\items\Id;
use Ubiquity\attributes\items\JoinColumn;

class User{

   #[Id]
   private $id;

   private $firstname;

   #[ManyToOne]
   #[JoinColumn(className: \models\Organization::class, name: 'idOrganization',␣
↪nullable: false)]
   private $organization;

   public function getOrganization(){
      return $this->organization;
   }

   public function setOrganization($organization){
      $this->organization=$organization;
   }
}
```

Annotations

Code source 8 – app/models/User.php

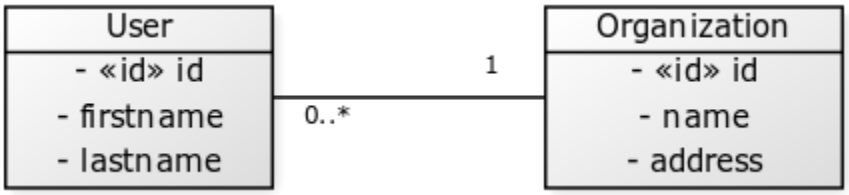
```php
namespace models;

class User{
  /**
   * @id
   */
  private $id;

  private $firstname;

  /**
   * @manyToOne
   * @joinColumn("className"=>"models\\Organization","name"=>"idOrganization","nullable"=>false)
   */
  private $organization;

  public function getOrganization(){
     return $this->organization;
  }

  public function setOrganization($organization){
     $this->organization=$organization;
  }
}
```

L'annotation **@joinColumn** ou l'attribut **JoinColumn** le spécifie :

- Le membre **$organisation** est une instance de **modelsOrganization**.
- La table **user** possède une clé étrangère **idOrganization** faisant référence à la clé primaire de l'organisation.
- Cette clé étrangère n'est pas nulle => un utilisateur aura toujours une organisation.

OneToMany

Une **organisation** a de nombreux **utilisateurs** :



Attributs

Code source 9 – app/models/Organization.php

```php
namespace models;

use Ubiquity\attributes\items\OneToMany;
use Ubiquity\attributes\items\Id;
```

(suite sur la page suivante)

(suite de la page précédente)

```
5
6  class Organization{
7
8      #[Id]
9      private $id;
10
11     private $name;
12
13     #[OneToMany(mappedBy: 'organization', className: \models\User::class)]
14     private $users;
15 }
```

Annotation

Code source 10 – app/models/Organization.php

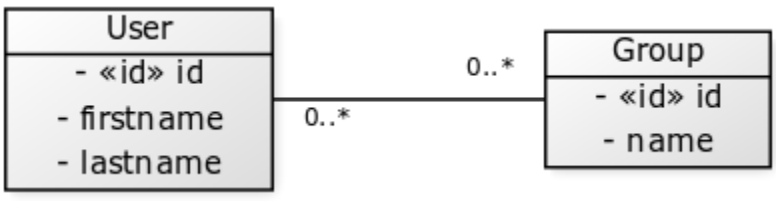
```
1  namespace models;
2
3  class Organization{
4      /**
5       * @id
6       */
7      private $id;
8
9      private $name;
10
11     /**
12      * @oneToMany("mappedBy"=>"organization","className"=>"models\\User")
13      */
14     private $users;
15 }
```

Dans ce cas, l'association est bidirectionnelle. L'annotation **@oneToMany** doit juste préciser :

- La classe de chaque utilisateur dans le tableau users : **modelsUser**.
- la valeur de **@mappedBy** est le nom de l'attribut association-mapping du côté propriétaire : **$organisation** dans la classe **User**.

ManyToMany

- Un **utilisateur** peut appartenir à des **groupes**.
- Un **groupe** se compose de plusieurs **utilisateurs**.



Attributs

Code source 11 – app/models/User.php

```php
namespace models;

use Ubiquity\attributes\items\ManyToMany;
use Ubiquity\attributes\items\Id;
use Ubiquity\attributes\items\JoinTable;

class User{

    #[Id]
    private $id;

    private $firstname;

    #[ManyToMany(targetEntity: \models\Group::class, inversedBy: 'users')]
    #[JoinTable(name: 'groupusers')]
    private $groups;

}
```

Annotations

Code source 12 – app/models/User.php

```php
namespace models;

class User{
    /**
     * @id
     */
    private $id;

    private $firstname;

    /**
     * @manyToMany("targetEntity"=>"models\\Group","inversedBy"=>"users")
     * @joinTable("name"=>"groupusers")
     */
    private $groups;

}
```

Attributs

Code source 13 – app/models/Group.php

```php
namespace models;

use Ubiquity\attributes\items\ManyToMany;
use Ubiquity\attributes\items\Id;
use Ubiquity\attributes\items\JoinTable;

```

(suite sur la page suivante)

(suite de la page précédente)

```
class Group{

   #[Id]
   private $id;

   private $name;

   #[ManyToMany(targetEntity: \models\User::class, inversedBy: 'groups')]
   #[JoinTable(name: 'groupusers')]
   private $users;

}
```

Annotations

Code source 14 – app/models/Group.php

```
namespace models;

class Group{
   /**
    * @id
    */
   private $id;

   private $name;

   /**
    * @manyToMany("targetEntity"=>"models\\User","inversedBy"=>"groups")
    * @joinTable("name"=>"groupusers")
    */
   private $users;

}
```

Si les conventions de nommage ne sont pas respectées pour les clés étrangères, il est possible de spécifier les champs associés.

Attributs

Code source 15 – app/models/Group.php

```
namespace models;

use Ubiquity\attributes\items\ManyToMany;
use Ubiquity\attributes\items\Id;
use Ubiquity\attributes\items\JoinTable;

class Group{

   #[Id]
   private $id;

```

(suite sur la page suivante)

(suite de la page précédente)

```
    private $name;

    #[ManyToMany(targetEntity: \models\User::class, inversedBy: 'groupes')]
    #[JoinTable(name: 'groupeusers',
    joinColumns: ['name'=>'id_groupe','referencedColumnName'=>'id'],
    inverseJoinColumns: ['name'=>'id_user','referencedColumnName'=>'id'])]
    private $users;

}
```

Annotations

Code source 16 – app/models/Group.php

```
namespace models;

class Group{
    /**
     * @id
     */
    private $id;

    private $name;

    /**
     * @manyToMany("targetEntity"=>"models\\User","inversedBy"=>"groupes")
     * @joinTable("name"=>"groupeusers",
     * "joinColumns"=>["name"=>"id_groupe","referencedColumnName"=>"id"],
     * "inverseJoinColumns"=>["name"=>"id_user","referencedColumnName"=>"id"])
     */
    private $users;

}
```

16.2 Annotations ORM

16.2.1 Annotations pour les classes

@annotation	rôle	propriétés	rôle
@database	Définit l'offset de la base de données associée (défini dans le fichier de configuration)		
@table	Définit le nom de la table associée.		

16.2.2 Annotations pour les membres

@annotation	rôle	propriétés	rôle
@id	Définit la ou les clés primaires.		
@column	Spécifie les caractéristiques du champ associé.	name	Nom du champ associé
		nullable	true si la valeur peut être nulle
		dbType	Type du champ dans la base de données
@transient	Indique que le champ n'est pas persistant.		

16.2.3 Associations

@annotation (extends)	rôle	properties [optional]	rôle
@manyToOne	Définit une association à valeur unique avec une autre classe d'entité.		
@joinColumn (@column)	Indique la clé étrangère dans l'association manyToOne.	className	Classe du membre
		[referencedColumnName]	Nom de la colonne associée
@oneToMany	Définit une association à valeurs multiples avec une autre classe d'entité.	className	Classe des objets du membre
		[mappedBy]	Nom de l'attribut de mappage d'association du côté propriétaire.
@manyToMany	Définit une association à valeurs multiples avec une multiplicité de plusieurs à plusieurs.	targetEntity	Classe des objets du membre
		[inversedBy]	Nom du membre associé du côté inverse
		[mappedBy]	Nom du membre associé du côté propriétaire
@joinTable	Définit la table d'association pour l'association many-to-many.	name	Le nom de la table d'association
		[joinColumns]	@column => name et referencedColumnName pour ce côté
		[inverseJoinColumns]	@column => name et referencedColumnName pour l'autre côté

CHAPITRE 17

DAO

La classe **DAO** est en charge des opérations de chargement et de persistance des modèles :

17.1 Connexion à la base de données

Vérifiez que les paramètres de connexion à la base de données sont correctement renseignés dans le fichier de configuration :

```
Ubiquity config -f=database
```

Depuis la version 2.3.0

Le démarrage de la base de données avec `DAO::startDatabase($config)` dans le fichier services.php est inutile, nul besoin de démarrer la base de données, la connexion est faite automatiquement à la première requête. Utilisez `DAO::start()` dans le fichier **app/config/services.php** lorsque vous utilisez plusieurs bases de données (avec la fonctionnalité multi db).

17.2 Chargement de données

17.2.1 Chargement d'une instance

Chargement d'une instance de la classe *modelsUser* avec l'identifiant *5*.

```
use Ubiquity\orm\DAO;
use models\User;

$user=DAO::getById(User::class, 5);
```

Chargement d'une instance en utilisant une condition :

```php
use Ubiquity\orm\DAO;
use models\User;

DAO::getOne(User::class, 'name= ?',false,['DOE']);
```

Chargement de BelongsTo

Par défaut, les membres définis par une relation **belongsTo** sont automatiquement chargés.

Chaque utilisateur n'appartient qu'à une seule catégorie :

```php
$user=DAO::getById(User::class,5);
echo $user->getCategory()->getName();
```

Il est possible d'empêcher ce chargement par défaut ; le troisième paramètre permet de charger ou non les membres belongsTo :

```php
$user=DAO::getOne(User::class,5, false);
echo $user->getCategory();// NULL
```

Chargement de HasMany

Le chargement des membres **hasMany** doit toujours être explicite ; le troisième paramètre permet le chargement explicite des membres.

Chaque utilisateur fait partie de plusieurs groupes :

```php
$user=DAO::getOne(User::class,5,['groupes']);
foreach($user->getGroupes() as $groupe){
    echo $groupe->getName().'<br>';
}
```

Clé primaire composite

Soit le modèle *ProductDetail* correspondant à un produit commandé sur une commande et dont la clé primaire est composite :

Attributs

Code source 1 – app/models/ProductDetail.php

```php
namespace models;

use Ubiquity\attributes\items\Id;

class ProductDetail{

  #[Id]
  private $idProduct;

  #[Id]
```

(suite sur la page suivante)

(suite de la page précédente)

```
    private $idCommand;

    ...
}
```

Annotations

Code source 2 – app/models/ProductDetail.php

```
namespace models;

class ProductDetail{
    /**
     * @id
     */
    private $idProduct;

    /**
     * @id
     */
    private $idCommand;

    ...
}
```

Le deuxième paramètre *$keyValues* peut être un tableau si la clé primaire est composite :

```
$productDetail=DAO::getOne(ProductDetail::class,[18,'BF327']);
echo 'Command:'.$productDetail->getCommande().'<br>';
echo 'Product:'.$productDetail->getProduct().'<br>';
```

17.2.2 Chargement de plusieurs objets

Chargement d'instances de la classe *User* :

```
$users=DAO::getAll(User::class);
foreach($users as $user){
    echo $user->getName()."<br>";
}
```

chargement des membres en relation

Chargement d'instances de la classe *User* avec sa catégorie et ses groupes :

```
$users=DAO::getAll(User::class,['groupes','category']);
foreach($users as $user){
    echo "<h2>".$user->getName()."</h2>";
    echo $user->getCategory()."<br>";
    echo "<h3>Groups</h3>";
    echo "<ul>";
```

(suite sur la page suivante)

(suite de la page précédente)

```
    foreach($user->getGroupes() as $groupe){
        echo "<li>".$groupe->getName()."</li>";
    }
    echo "</ul>";
}
```

Descendre dans la hiérarchie des objets liés : Chargement des instances de la classe *User* avec sa catégorie, ses groupes et l'organisation de chaque groupe :

```
$users=DAO::getAll(User::class,['groupes.organization','category']);
foreach($users as $user){
    echo "<h2>".$user->getName()."</h2>";
    echo $user->getCategory()."<br>";
    echo "<h3>Groups</h3>";
    echo "<ul>";
    foreach($user->getGroupes() as $groupe){
        echo "<li>".$groupe->getName()."<br>";
        echo "<li>".$groupe->getOrganization()->getName()."</li>";
    }
    echo "</ul>";
}
```

Utilisation de caractères de remplacement :

Chargement des instances de la classe *User* avec sa catégorie, ses groupes et tous les membres en relation de chaque groupe :

```
$users=DAO::getAll(User::class,['groupes.*','category']);
```

17.2.3 Requêtage utilisant des conditions

Requêtes simples

Le paramètre *condition* est équivalent à la partie WHERE d'une instruction SQL :

```
$users=DAO::getAll(User::class,'firstName like "bren%" and not suspended',false);
```

Pour éviter les injections SQL et bénéficier de la préparation des statements, il est préférable d'effectuer une requête paramétrée :

```
$users=DAO::getAll(User::class,'firstName like ? and suspended= ?',false,['bren%',
↪false]);
```

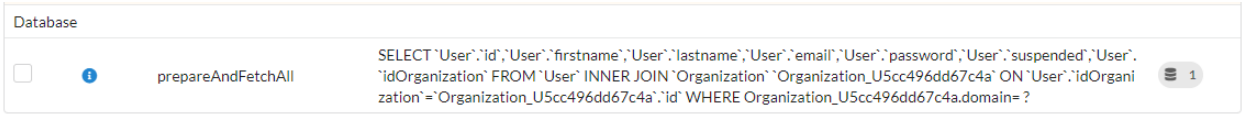
UQueries

L'utilisation de **U-queries** permet de poser des conditions sur les membres associés :

Sélection des utilisateurs dont l'organisation possède le domaine **lecnam.net** :

```
$users=DAO::uGetAll(User::class,'organization.domain= ?',false,['lecnam.net']);
```

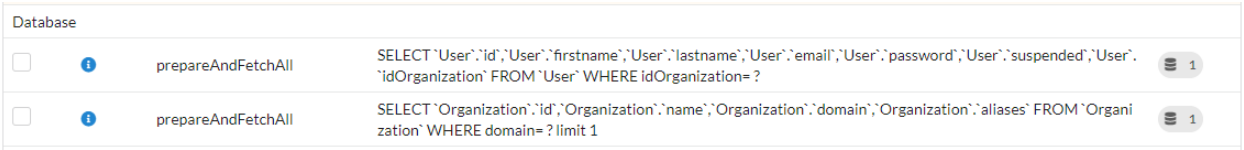
Il est possible de visualiser la requête générée dans les logs (si le logging est activé) :



Le résultat peut être vérifié en sélectionnant tous les utilisateurs de cette organisation :

```
$organization=DAO::getOne(Organization::class,'domain= ?',['users'],['lecnam.net']);
$users=$organization->getUsers();
```

Les logs correspondants :



17.2.4 Comptage

Test de l'existance

```
if(DAO::exists(User::class,'lastname like ?',['SMITH'])){
    //there's a Mr SMITH
}
```

Comptage

Pour compter les instances, ce qu'il ne faut pas faire, si les utilisateurs ne sont pas déjà chargés :

```
$users=DAO::getAll(User::class);
echo "there are ". \count($users) ." users";
```

Ce qui doit être fait :

```
$count=DAO::count(User::class);
echo "there are $count users";
```

Avec une condition :

```
$notSuspendedCount=DAO::count(User::class, 'suspended = ?', [false]);
```

Avec une condition sur les objets associés :

Nombre d'utilisateurs appartenant à l'organisation nommée **OTAN**.

```
$count=DAO::uCount(User::class,'organization.name= ?',['OTAN']);
```

17.3 Modification de données

17.3.1 Ajout d'une instance

Ajout d'une organisation :

```
$orga=new Organization();
$orga->setName('Foo');
$orga->setDomain('foo.net');
if(DAO::save($orga)){
  echo $orga.' added in database';
}
```

Ajout d'une instance d'utilisateur, dans une organisation :

```
$orga=DAO::getById(Organization::class, 1);
$user=new User();
$user->setFirstname('DOE');
$user->setLastname('John');
$user->setEmail('doe@bar.net');
$user->setOrganization($orga);
if(DAO::save($user)){
  echo $user.' added in database in '.$orga;
}
```

17.3.2 Mise à jour d'une instance

Dans un premier temps, l'instance doit être chargée :

```
$orga=DAO::getOne(Organization::class,'domain= ?',false,['foo.net']);
$orga->setAliases('foo.org');
if(DAO::save($orga)){
  echo $orga.' updated in database';
}
```

17.3.3 Suppression d'une instance

Si l'instance est déjà chargée depuis la base de données :

```
$orga=DAO::getById(Organization::class,5,false);
if(DAO::remove($orga)){
  echo $orga.' deleted from database';
}
```

Si l'instance n'est pas chargée, il est plus approprié d'utiliser la méthode *delete* :

```
if(DAO::delete(Organization::class,5)){
  echo 'Organization deleted from database';
}
```

17.4 Suppression de plusieurs instances

Suppression de plusieurs instances sans chargement préalable :

```
if($res=DAO::deleteAll(models\User::class, 'id in (?,?,?)',[1,2,3])){
    echo "$res elements deleted";
}
```

17.5 Requêtes en masse (bulk)

Les requêtes en masse permettent d'effectuer plusieurs opérations (insertion, modification ou suppression) en une seule requête, ce qui contribue à améliorer les performances.

17.5.1 Insertions en masse

Exemple d'insertion :

```
$u = new User();
$u->setName('Martin1');
DAO::toInsert($u);
$u = new User();
$u->setName('Martin2');
DAO::toInsert($u);
//Perform inserts
DAO::flushInserts();
```

17.5.2 Mises à jour en masse

Exemple de mise à jour :

```
$users = DAO::getAll(User::class, 'name like ?', false, [
   'Martin%'
]);
foreach ($users as $user) {
   $user->setName(\strtoupper($user->getName()));
   DAO::toUpdate($user);
}
DAO::flushUpdates();
```

17.5.3 Suppressions en masse

Exemple de suppression :

```
$users = DAO::getAll(User::class, 'name like ?', false, [
     'BULK%'
]);
DAO::toDeletes($users);
DAO::flushDeletes();
```

La méthode *DAO : :flush()* peut être appelée si des insertions, des mises à jour ou des suppressions sont en attente.

17.6 Transactions

17.6.1 Transactions explicites

Toutes les opérations DAO peuvent être insérées dans une transaction, ce qui permet d'atomiser une série de changements :

```
try{
   DAO::beginTransaction();
   $orga=new Organization();
   $orga->setName('Foo');
   DAO::save($orga);

   $user=new User();
   $user->setFirstname('DOE');
   $user->setOrganization($orga);
   DAO::save($user);
   DAO::commit();
}catch (\Exception $e){
   DAO::rollBack();
}
```

En cas de bases de données multiples définies dans la configuration, les méthodes liées aux transactions peuvent prendre l'offset de base de données défini en paramètre.

```
DAO::beginTransaction('db-messagerie');
//some DAO operations on messagerie models
DAO::commit('db-messagerie');
```

17.6.2 Transactions implicites

Certaines méthodes DAO utilisent implicitement les transactions pour regrouper les opérations d'insertion, de mise à jour ou de suppression.

```
$users=DAO::getAll(User::class);
foreach ($users as $user){
    $user->setSuspended(true);
    DAO::toUpdate($user);
```

(suite sur la page suivante)

(suite de la page précédente)

```
}
DAO::updateGroups();//Perform updates in a transaction
```

17.7 Classe SDAO

La classe **SDAO** accélère les opérations CRUD pour les classes métier sans relations.

Les modèles doivent dans ce cas déclarer uniquement des membres publics, et ne pas respecter l'encapsulation habituelle.

Code source 3 – app/models/Product.php

```
namespace models;
class Product{
  /**
   * @id
   */
  public $id;

  public $name;

  ...
}
```

La classe **SDAO** hérite de **DAO** et possède les mêmes méthodes pour effectuer des opérations CRUD.

```
use Ubiquity\orm\DAO;

$product=DAO::getById(Product::class, 5);
```

17.8 Requêtes DAO préparées

La préparation de certaines requêtes peut améliorer les performances avec les serveurs Swoole, Workerman ou Roadrunner. Cette préparation initialise les objets qui seront ensuite utilisés pour exécuter la requête. Cette initialisation est effectuée au démarrage du serveur, ou au démarrage de chaque worker, si un tel événement existe.

17.8.1 Exemple Swoole

Préparation

Code source 4 – app/config/swooleServices.php

```
$swooleServer->on('workerStart', function ($srv) use (&$config) {
   \Ubiquity\orm\DAO::startDatabase($config);
   \Ubiquity\orm\DAO::prepareGetById('user', User::class);
   \Ubiquity\orm\DAO::prepareGetAll('productsByName', Product::class,'name like ?');
});
```

Utilisation

Code source 5 – app/controllers/UsersController.php

```
public function displayUser($idUser){
   $user=DAO::executePrepared('user',[1]);
   echo $user->getName();
}

public function displayProducts($name){
   $products=DAO::executePrepared('productsByName',[$name]);
   ...
}
```

CHAPITRE 18

Requête

Note : Pour toutes les fonctionnalités Http, Ubiquity utilise des classes techniques contenant des méthodes statiques. C'est un choix de conception pour éviter l'injection de dépendances qui dégraderait les performances.

La classe **URequest** offre des fonctionnalités supplémentaires permettant de manipuler plus facilement les tableaux php natifs **$_POST** et **$_GET**.

18.1 Récupération de données

18.1.1 A partir de la méthode get

La méthode **get** renvoie la valeur *null* si la clé **name** n'existe pas dans les variables get.

```
use Ubiquity\utils\http\URequest;

$name=URequest::get("name");
```

La méthode **get** peut être appelée avec le second paramètre facultatif qui renvoie une valeur si la clé n'existe pas dans les variables get.

```
$name=URequest::get("name",1);
```

18.1.2 A partir de la méthode post

La méthode **post** renvoie la valeur *null* si la clé **name** n'existe pas dans les variables post.

```
use Ubiquity\utils\http\URequest;

$name=URequest::post("name");
```

La méthode **post** peut être appelée avec le second paramètre facultatif qui renvoie une valeur si la clé n'existe pas dans les variables post.

```
$name=URequest::post("name",1);
```

La méthode **getPost** applique un callback aux éléments du tableau $_POST et les retourne (callback par défaut : **htmlEntities**) :

```
$protectedValues=URequest::getPost();
```

18.2 Récupération et affectation de données multiples

Il est courant d'affecter les valeurs d'un tableau associatif aux membres d'un objet. C'est le cas par exemple lors de la validation d'un formulaire de modification d'objet.

La méthode **setValuesToObject** effectue cette opération :

Considérons une classe **User** :

```
class User {
	private $id;
	private $firstname;
	private $lastname;

	public function setId($id){
		$this->id=$id;
	}
	public function getId(){
		return $this->id;
	}

	public function setFirstname($firstname){
		$this->firstname=$firstname;
	}
	public function getFirstname(){
		return $this->firstname;
	}

	public function setLastname($lastname){
		$this->lastname=$lastname;
	}
	public function getLastname(){
		return $this->lastname;
	}
}
```

Considérons un formulaire pour modifier un utilisateur :

```
<form method="post" action="Users/update">
 <input type="hidden" name="id" value="{{user.id}}">
     <label for="firstname">Firstname:</label>
     <input type="text" id="firstname" name="firstname" value="{{user.firstname}}">
     <label for="lastname">Lastname:</label>
     <input type="text" id="lastname" name="lastname" value="{{user.lastname}}">
     <input type="submit" value="validate modifications">
</form>
```

L'action **update** du contrôleur **Users** doit mettre à jour l'instance utilisateur à partir des valeurs POST. L'utilisation de la méthode **setPostValuesToObject** permet d'éviter l'affectation des variables postées une à une aux membres de l'objet. Il est également possible d'utiliser la méthode **setGetValuesToObject** pour la méthode **get**, ou **setValuesToObject** pour affecter les valeurs de tout tableau associatif à un objet.

Code source 1 – app/controllers/Users.php

```
1  namespace controllers;
2
3  use Ubiquity\orm\DAO;
4  use Uniquity\utils\http\URequest;
5
6  class Users extends BaseController{
7      ...
8      public function update(){
9              $user=DAO::getOne("models\User",URequest::post("id"));
10             URequest::setPostValuesToObject($user);
11             DAO::update($user);
12     }
13 }
```

Note : Les méthodes **SetValuesToObject** utilisent des setters pour modifier les membres d'un objet. La classe concernée doit donc implémenter des setters pour tous les membres modifiables.

18.3 Test de la requête

18.3.1 isPost

La méthode **isPost** renvoie *true* si la requête a été soumise via la méthode POST : Dans le cas ci-dessous, la méthode *initialize* ne charge la vue *vHeader.html* que si la requête n'est pas une requête Ajax.

Code source 2 – app/controllers/Users.php

```
1  namespace controllers;
2
3  use Ubiquity\orm\DAO;
4  use Ubiquity\utils\http\URequest;
5
6  class Users extends BaseController{
```

(suite sur la page suivante)

(suite de la page précédente)

```
    ...
    public function update(){
            if(URequest::isPost()){
                    $user=DAO::getOne("models\User",URequest::post("id"));
                    URequest::setPostValuesToObject($user);
                    DAO::update($user);
            }
    }
}
```

18.3.2 isAjax

La méthode **isAjax** renvoie *true* si la requête est une requête Ajax :

Code source 3 – app/controllers/Users.php

```
...
    public function initialize(){
            if(!URequest::isAjax()){
                    $this->loadView("main/vHeader.html");
            }
    }
    ...
```

18.3.3 isCrossSite

La méthode **isCrossSite** vérifie que la requête n'est pas cross-site.

CHAPITRE 19

Réponse

Note : Pour toutes les fonctionnalités Http, Ubiquity utilise des classes techniques contenant des méthodes statiques. C'est un choix de conception pour éviter l'injection de dépendances qui dégraderait les performances.

La classe **UResponse** ne gère que les en-têtes, pas le corps de la réponse, qui est conventionnellement fourni par le contenu affiché par les appels utilisés pour écrire des données (echo, print...).

La classe **UResponse** offre des fonctionnalités supplémentaires permettant de manipuler plus facilement les en-têtes de réponse.

19.1 Ajout ou modification d'en-têtes

```
use Ubiquity\utils\http\UResponse;
$animal='camel';
UResponse::header('Animal',$animal);
```

Forcer plusieurs en-têtes du même type :

```
UResponse::header('Animal','monkey',false);
```

Force le code de réponse HTTP à la valeur spécifiée :

```
UResponse::header('Messages',$message,false,500);
```

19.2 Définition d'en-têtes spécifiques

19.2.1 content-type

Définir le type de contenu de la réponse à **application/json** :

```
UResponse::asJSON();
```

Définir le type de contenu de la réponse à **text/html** :

```
UResponse::asHtml();
```

Définir le type de contenu de la réponse à **plain/text** :

```
UResponse::asText();
```

Définir le type de contenu de la réponse à **application/xml** :

```
UResponse::asXml();
```

Définition d'un encodage spécifique (la valeur par défaut est toujours **utf-8**) :

```
UResponse::asHtml('iso-8859-1');
```

19.3 Cache

Forcer la désactivation du cache du navigateur :

```
UResponse::noCache();
```

19.4 Accept

Définit les types de contenu, exprimés sous forme de types MIME, que le client est en mesure de comprendre. Voir Acceptation des valeurs par défaut.

```
UResponse::setAccept('text/html');
```

19.5 En-têtes de réponse CORS

Cross-Origin Resource Sharing (CORS) est un mécanisme qui utilise des en-têtes HTTP supplémentaires pour indiquer au navigateur que votre application Web exécutée à une origine (domaine) a la permission d'accéder à des ressources sélectionnées sur un serveur à une origine différente.

19.5.1 Access-Control-Allow-Origin

Réglage de l'origine autorisée :

```
UResponse::setAccessControlOrigin('http://myDomain/');
```

19.5.2 Access-Control-Allow-methods

Définir les méthodes autorisées :

```
UResponse::setAccessControlMethods('GET, POST, PUT, DELETE, PATCH, OPTIONS');
```

19.5.3 Access-Control-Allow-headers

Définir les en-têtes autorisés :

```
UResponse::setAccessControlHeaders('X-Requested-With, Content-Type, Accept, Origin,␣
↪Authorization');
```

19.5.4 Activation globale de CORS

activer CORS pour un domaine avec des valeurs par défaut :
- Méthodes autorisées : « GET, POST, PUT, DELETE, PATCH, OPTIONS ».
- En-têtes autorisés : « X-Requested-With, Content-Type, Accept, Origin, Authorization ».

```
UResponse::enableCors('http://myDomain/');
```

19.6 Test des en-têtes de réponse

Vérifier si les en-têtes ont été envoyés :

```
if(!UResponse::isSent()){
    //do something if headers are not send
}
```

Test si le type de contenu de la réponse est **application/json** :

Important : Cette méthode ne fonctionne que si vous avez utilisé la classe UResponse pour définir les en-têtes.

```
if(UResponse::isJSON()){
    //do something if response is a JSON response
}
```

CHAPITRE 20

Session

Note : Pour toutes les fonctionnalités Http, Ubiquity utilise des classes techniques contenant des méthodes statiques. C'est un choix de conception pour éviter l'injection de dépendances qui dégraderait les performances.

La classe **USession** fournit des fonctionnalités supplémentaires pour manipuler plus facilement le tableau natif **$_SESSION** de php.

20.1 Démarrer la session

La session Http est lancée automatiquement si la clé **sessionName** est renseignée dans le fichier de configuration **app/config.php** :

```
<?php
return array(
            ...
            "sessionName"=>"key-for-app",
            ...
 );
```

Si la clé sessionName n'est pas renseignée, il est nécessaire de démarrer la session explicitement pour l'utiliser :

```
use Ubiquity\utils\http\USession;
...
USession::start("key-for-app");
```

Note : Le paramètre **name** est facultatif mais recommandé pour éviter les conflits de variables.

20.2 Création ou modification d'une variable de session

```
use Ubiquity\utils\http\USession;

USession::set("name","SMITH");
USession::set("activeUser",$user);
```

20.3 Récupération de données

La méthode **get** renvoie la valeur *null* si la clé **name** n'existe pas dans les variables de session.

```
use Ubiquity\utils\http\USession;

$name=USession::get("name");
```

La méthode **get** peut être appelée avec le second paramètre facultatif qui renvoie une valeur si la clé n'existe pas dans les variables de session.

```
$name=USession::get("page",1);
```

Note : La méthode **session** est un alias de la méthode **get**.

La méthode **getAll** renvoie toutes les variables de la session :

```
$sessionVars=USession::getAll();
```

20.4 Test

La méthode **exists** teste l'existence d'une variable en session.

```
if(USession::exists("name")){
     //do something when name key exists in session
}
```

La méthode **isStarted** vérifie que la session est démarrée.

```
if(USession::isStarted()){
     //do something if the session is started
}
```

20.5 Suppression de variables

La méthode **delete** permet de supprimer une variable de session :

```
USession::delete("name");
```

20.6 Clôture explicite de la session

La méthode **terminate** ferme correctement la session et supprime toutes les variables de session créées :

```
USession::terminate();
```

CHAPITRE 21

Cookie

Note : Pour toutes les fonctionnalités Http, Ubiquity utilise des classes techniques contenant des méthodes statiques. C'est un choix de conception pour éviter l'injection de dépendances qui dégraderait les performances.

La classe **UCookie** fournit des fonctionnalités supplémentaires permettant de manipuler plus facilement le tableau php natif **$_COOKIES**.

21.1 Création ou modification de Cookie

```php
use Ubiquity\utils\http\UCookie;

$cookie_name = 'user';
$cookie_value = 'John Doe';
UCookie::set($cookie_name, $cookie_value);//duration : 1 day
```

Création d'un cookie d'une durée de 5 jours :

```php
UCookie::set($cookie_name, $cookie_value,5*60*60*24);
```

Sur un domaine particulier :

```php
UCookie::set($cookie_name, $cookie_value,5*60*60*24,'/admin');
```

Envoi d'un cookie sans url-encodage de la valeur du cookie :

```php
UCookie::setRaw($cookie_name, $cookie_value);
```

Test de la création du cookie :

```
if(UCookie::setRaw($cookie_name, $cookie_value)){
     //cookie created
}
```

21.2 Récupération d'un cookie

```
$userName=UCookie::get('user');
```

21.2.1 Vérification de l'existence

```
if(UCookie::exists('user')){
     //do something if cookie user exists
}
```

21.2.2 Utilisation d'une valeur par défaut

Si le cookie nommé page n'existe pas, la valeur par défaut 1 est renvoyée :

```
$page=UCookie::get('page',1);
```

21.3 Suppression d'un cookie

Suppression du cookie avec le nom **page** :

```
UCookie::delete('page');
```

21.4 Suppression de tous les cookies

Suppression de tous les cookies d'un domaine entier :

```
UCookie::deleteAll();
```

Suppression de tous les cookies du domaine **admin** :

```
UCookie::deleteAll('/admin');
```

CHAPITRE 22

Vues

Ubiquity utilise Twig comme moteur de template par défaut (voir Documentation Twig). Les vues sont situées dans le dossier **app/views**. Elles doivent avoir l'extension **.html** pour être interprétées par Twig.

Ubiquity peut également être utilisé avec un système de vues PHP, pour obtenir de meilleures performances, ou simplement pour permettre l'utilisation de php dans les vues.

22.1 Chargement

Les vues sont chargées depuis les contrôleurs :

Code source 1 – app/controllers/Users.php

```php
namespace controllers;

class Users extends BaseController{
    ...
    public function index(){
                    $this->loadView("index.html");
            }
    }
}
```

22.1.1 Chargement de la vue par défaut

Si vous utilisez la méthode de dénomination des vues par défaut : La vue par défaut associée à une action dans un contrôleur est située dans le dossier `views/controller-name/action-name` :

```
views
     |
     └ Users
        └ info.html
```

Code source 2 – app/controllers/Users.php

```php
namespace controllers;

class Users extends BaseController{
    ...
    public function info(){
                    $this->loadDefaultView();
            }
    }
}
```

22.2 Chargement et passage de variables

Les variables sont transmises à la vue dans un tableau associatif. Chaque clé crée une variable du même nom dans la vue.

Code source 3 – app/controllers/Users.php

```php
namespace controllers;

class Users extends BaseController{
    ...
    public function display($message,$type){
                    $this->loadView("users/display.html",["message"=>$message,"type"=>
↪$type]);
            }
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }
}
```

Dans ce cas, il est utile d'appeler `compact` pour créer un tableau contenant des variables et leurs valeurs :

Code source 4 – app/controllers/Users.php

```
namespace controllers;

class Users extends BaseController{
    ...
    public function display($message,$type){
                    $this->loadView("users/display.html",compact("message","type"));
            }
    }
}
```

22.3 Affichage dans une vue

La vue peut alors afficher les variables :

Code source 5 – users/display.html

```
<h2>{{type}}</h2>
<div>{{message}}</div>
```

Les variables peuvent également avoir des attributs ou des éléments auxquels vous pouvez accéder.

Vous pouvez utiliser un point (.) pour accéder aux attributs d'une variable (méthodes ou propriétés d'un objet PHP, ou éléments d'un tableau PHP), ou la syntaxe dite « d'indice » ([]) :

```
{{ foo.bar }}
{{ foo['bar'] }}
```

22.4 Fonctions supplémentaires Ubiquity

La variable globale `app` donne accès à des fonctionnalités prédéfinies d'Ubiquity Twig :

— `app` est une instance de la classe `Framework` et donne accès aux méthodes publiques de cette classe.

Obtenir la version installée du framework :

```
{{ app.version() }}
```

Renvoie les noms du contrôleur actif et de l'action :

```
{{ app.getController() }}
{{ app.getAction() }}
```

Retourne les wrapper classes globales :

Pour la requête :

```
{{ app.getRequest().isAjax() }}
```

Pour la session :

```
{{ app.getSession().get('homePage','index') }}
```

voir Classeframework dans API pour plus d'informations.

22.5 Chargement des vue PHP

Désactivez si nécessaire Twig dans le fichier de configuration en supprimant la clé **templateEngine**.

Créez ensuite un contrôleur qui hérite de « SimpleViewController », ou de « SimpleViewAsyncController » si vous utilisez **Swoole** ou **Workerman** :

Code source 6 – app/controllers/Users.php

```
1  namespace controllers;
2
3  use Ubiquity\controllers\SimpleViewController;
4
5  class Users extends SimpleViewController{
6      ...
7      public function display($message,$type){
8                      $this->loadView("users/display.php",compact("message","type"));
9              }
10     }
11 }
```

Note : Dans ce cas, les fonctions de chargement des assets et des thèmes ne sont pas prises en charge.

CHAPITRE 23

Assets

Les assets correspondent aux fichiers javascript, feuilles de style, polices de caractères, images à inclure dans votre application. Ils se trouvent dans le dossier **public/assets**. Il est préférable de séparer les ressources en sous-dossiers par type.

```
public/assets
      ├ css
      │   ├ style.css
      │   └ semantic.min.css
      └ js
          └ jquery.min.js
```

Intégration de fichiers css ou js :

```
{{ css('css/style.css') }}
{{ css('css/semantic.min.css') }}

{{ js('js/jquery.min.js') }}
```

```
{{ css('https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/semantic.min.css') }}

{{ js('https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/semantic.min.js') }}
```

CDN avec des paramètres supplémentaires :

```
{{ css('https://cdn.jsdelivr.net/npm/foundation-sites@6.5.3/dist/css/foundation.min.css',
↪{crossorigin: 'anonymous',integrity: 'sha256-/PFxCnsMh+...'}) }}
```

CHAPITRE 24

Thèmes

Note : Les thèmes sont totalement inutiles si vous n'avez qu'une seule présentation à appliquer.

Ubiquity supporte des thèmes qui peuvent avoir leurs propres assets et vues selon le modèle du thème à afficher par le contrôleur. Chaque action du contrôleur peut utiliser un thème spécifique, ou peut utiliser le thème par défaut configuré dans le fichier *config.php* dans `templateEngineOptions => array("activeTheme" => "semantic")`.

Ubiquity est livré avec 3 thèmes par défaut : **Bootstrap**, **Foundation** et **Semantic-UI**.

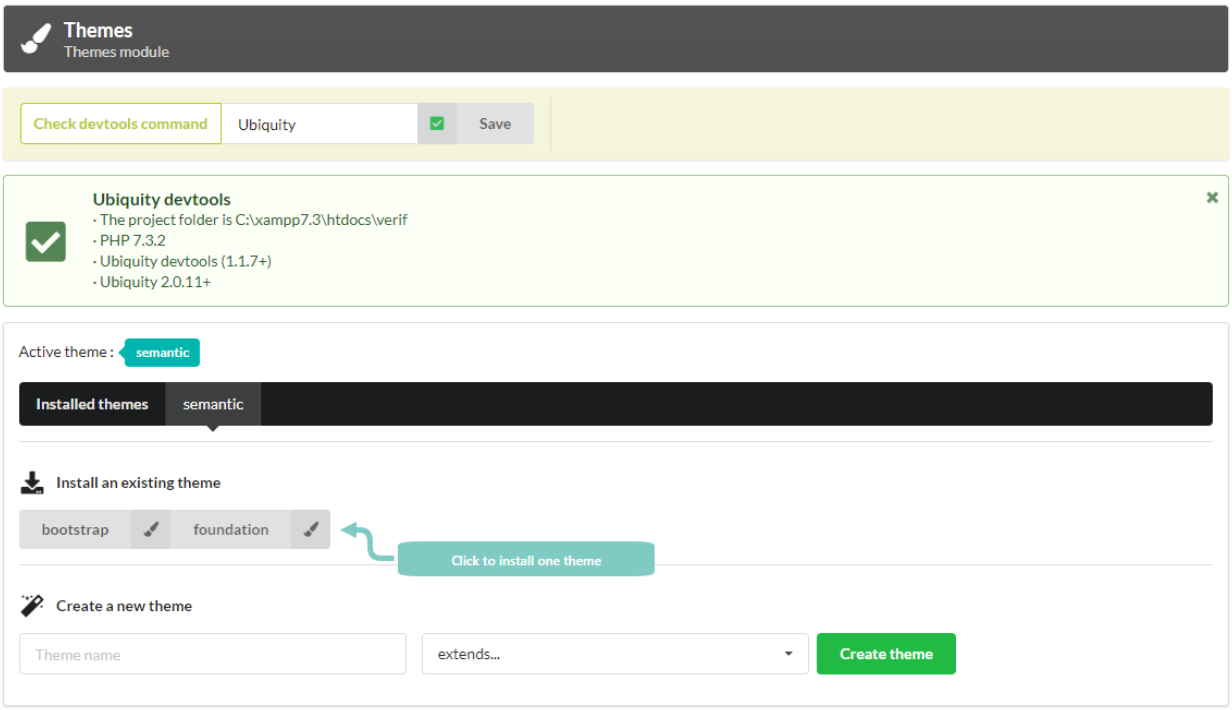
24.1 Installation d'un thème

Avec les devtools, exécuter :

```
Ubiquity install-theme bootstrap
```

Le thème installé est l'un des suivants : **bootstrap**, **foundation** ou **semantic**.

Avec **webtools**, vous pouvez faire de même, à condition que les **devtools** soient installés et accessibles (dossier Ubiquity ajouté dans le chemin du système) :



24.2 Création d'un nouveau thème

Avec les devtools, exécuter :

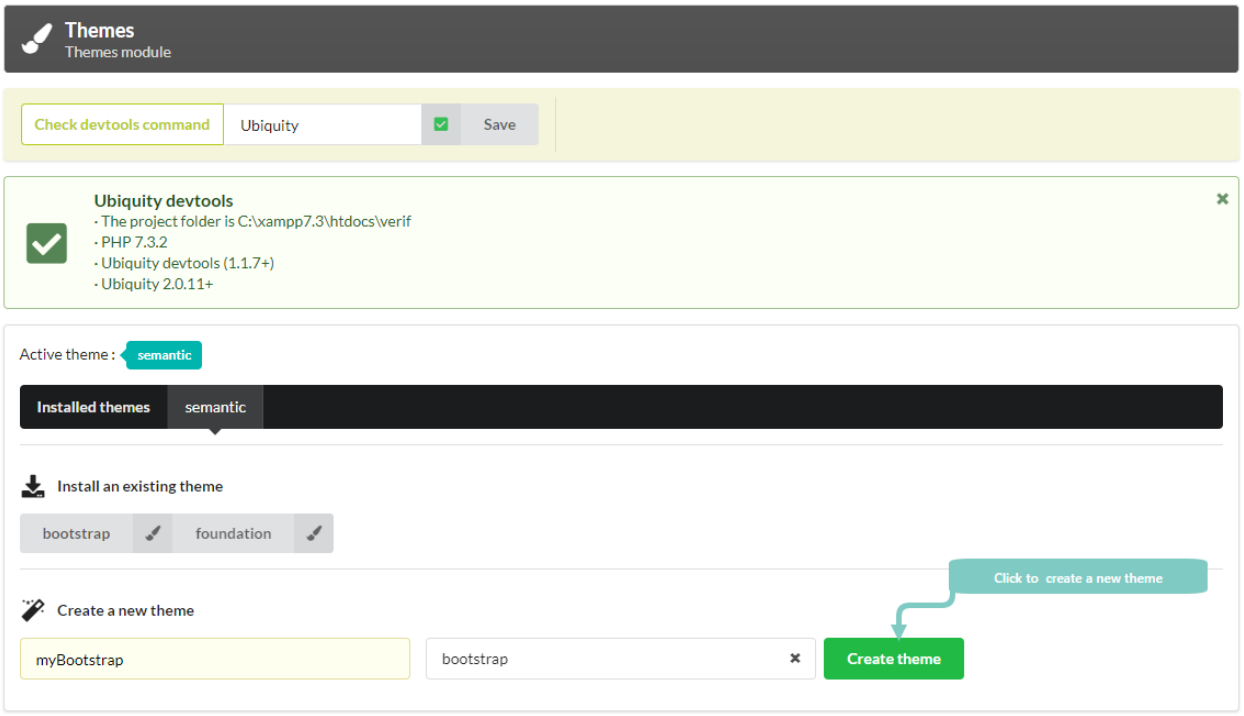
```
Ubiquity create-theme myTheme
```

Création d'un nouveau thème à partir de Bootstrap, Semantic…

Avec les devtools, exécuter :

```
Ubiquity create-theme myBootstrap -x=bootstrap
```

Avec les **webtools** :



24.3 Structure et fonctionnement des thèmes

24.3.1 Structure

Dossier vues d'un thème

Les vues d'un thème sont situées dans le dossier **app/views/themes/theme-name**.

```
app/views
        └ themes
              ├ bootstrap
              │        └ main
              │             ├ vHeader.html
              │             └ vFooter.html
              └ semantic
                       └ main
                            ├ vHeader.html
                            └ vFooter.html
```

La classe de base contrôleur est responsable du chargement des vues pour définir l'en-tête et le pied de page de chaque page :

Code source 1 – app/controllers/ControllerBase.php

```
1  <?php
2  namespace controllers;
3
4  use Ubiquity\controllers\Controller;
5  use Ubiquity\utils\http\URequest;
```

(suite sur la page suivante)

(suite de la page précédente)

```
6
7       /**
8        * ControllerBase.
9        **/
10      abstract class ControllerBase extends Controller{
11              protected $headerView = "@activeTheme/main/vHeader.html";
12              protected $footerView = "@activeTheme/main/vFooter.html";
13
14              public function initialize() {
15                      if (! URequest::isAjax ()) {
16                              $this->loadView ( $this->headerView );
17                      }
18              }
19              public function finalize() {
20                      if (! URequest::isAjax ()) {
21                              $this->loadView ( $this->footerView );
22                      }
23              }
24      }
```

Dossier assets d'un thème

Les assets d'un thème sont créés dans le dossier `public/assets/theme-name`.

La structure du dossier des assets est souvent la suivante :

```
public/assets/bootstrap
                                ├ css
                                │  ├ style.css
                                │  └ all.min.css
                                ├ scss
                                │  ├ myVariables.scss
                                │  └ app.scss
                                ├ webfonts
                                │
                                └ img
```

24.4 Changement du thème actif

24.4.1 Changement persistant

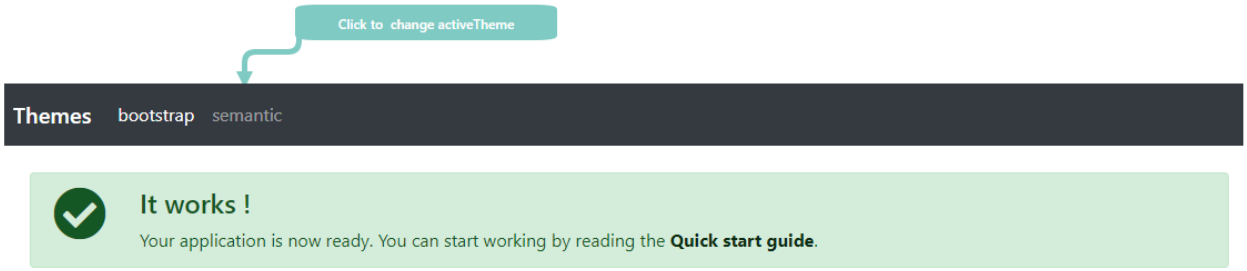
activeTheme est défini dans `app/config/config.php` avec `templateEngineOptions => array("activeTheme" => "semantic")`

Le thème actif peut être modifié avec les **devtools** :

```
Ubiquity config:set --templateEngineOptions.activeTheme=bootstrap
```

Il est également possible de le faire à partir de la page d'accueil, ou avec les **webtools** :

Depuis la page d'accueil :



Depuis les webtools :

Cette modification peut également être effectuée au moment de l'exécution :

Depuis un contrôleur :

```
ThemeManager::saveActiveTheme('bootstrap');
```

24.4.2 Changement local non persistant

Pour définir un thème spécifique pour toutes les actions d'un contrôleur, la méthode la plus simple consiste à surdéfinir la méthode **initialize** du contrôleur :

Code source 2 – app/controllers/Users.php

```
1  namespace controllers;
2
3  use \Ubiquity\themes\ThemesManager;
4
5  class Users extends BaseController{
6
7          public function initialize(){
8              parent::intialize();
9              ThemesManager::setActiveTheme('bootstrap');
10         }
11     }
```

Ou si le changement ne doit concerner qu'une seule action :

Code source 3 – app/controllers/Users.php

```
1  namespace controllers;
2
3  use \Ubiquity\themes\ThemesManager;
4
5  class Users extends BaseController{
6
```

(suite sur la page suivante)

(suite de la page précédente)

```
        public function doStuff(){
            ThemesManager::setActiveTheme('bootstrap');
            ...
        }
    }
```

Changement de thème conditionnel, indépendamment du contrôleur :

Exemple avec une modification du thème en fonction d'une variable passée dans l'URL

Code source 4 – app/config/services.php

```
use Ubiquity\themes\ThemesManager;
use Ubiquity\utils\http\URequest;

...

ThemesManager::onBeforeRender(function(){
            if(URequest::get("th")=='bootstrap'){
                    ThemesManager::setActiveTheme("bootstrap");
            }
    });
```

24.4.3 Support pour appareils mobiles

Ajouter un outil de détection des appareils mobiles. Installer MobileDetect :

```
composer require mobiledetect/mobiledetectlib
```

Il est généralement plus facile de créer des vues différentes par appareil.

Créez un thème spécifique pour la partie mobile (en créant un dossier `views/themes/mobile` et en y plaçant les vues spécifiques aux appareils mobiles). Il est important dans ce cas d'utiliser les mêmes noms de fichiers pour la partie mobile et la partie non-mobile.

Il est également conseillé dans ce cas que tous les chargements de vues utilisent l'espace de noms **@activeTheme** :

```
$this->loadView("@activeTheme/index.html");
```

index.html doit être disponible dans ce cas dans les dossiers `views` et `views/themes/mobile`.

Détection globale des mobiles (à partir de services.php)

Code source 5 – app/config/services.php

```
use Ubiquity\themes\ThemesManager;

...

ThemesManager::onBeforeRender(function () {
    $mb = new \Mobile_Detect();
    if ($mb->isMobile()) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
            ThemesManager::setActiveTheme('mobile');
        }
});
```

Détection locale (depuis un contrôleur)

Code source 6 – app/controllers/FooController.php

```
use Ubiquity\themes\ThemesManager;

...

    public function initialize() {
            $mb = new \Mobile_Detect();
            if ($mb->isMobile()) {
                    ThemesManager::setActiveTheme('mobile');
            }
            parent::initialize();
    }
```

24.5 Chargement de vues et d'assets

24.5.1 Vues

Pour charger une vue à partir du dossier **activeTheme**, vous pouvez utiliser l'espace de nom **@activeTheme** :

Code source 7 – app/controllers/Users.php

```php
namespace controllers;

class Users extends BaseController{

        public function action(){
            $this->loadView('@activeTheme/action.html');
            ...
        }
    }
```

Si le **activeTheme** est **bootstrap**, la vue chargée est `app/views/themes/bootstrap/action.html`.

24.5.2 Vue par défaut

Si vous suivez les conventions de nommage des vues Ubiquity, la vue par défaut chargée pour une action dans un contrôleur lorsqu'un thème est actif est : `app/views/themes/theme-name/controller-name/action-name.html`.

Par exemple, si le thème actif est bootstrap, la vue par défaut pour l'action display dans le contrôleur Users doit se trouver dans le fichier `app/views/themes/bootstrap/Users/display.html`.

Code source 8 – app/controllers/Users.php

```php
namespace controllers;

class Users extends BaseController{

        public function display(){
            $this->loadDefaultView();
            ...
        }
    }
```

Note : Les commandes devtools pour créer un contrôleur ou une action et leur vue associée utilisent le dossier **@activeTheme** si un thème est actif.

```
Ubiquity controller Users -v

Ubiquity action Users.display -v
```

24.6 Chargement des assets

Le mécanisme est le même que pour les vues : l'espace de nom `@activeTheme` fait référence au dossier `public/assets/theme-name/`.

```
{{ css('@activeTheme/css/style.css') }}

{{ js('@activeTheme/js/scripts.js') }}

{{ img('@activeTheme/img/image-name.png', {alt: 'Image Alt Name', class: 'css-class'}) }}
```

Si le thème **bootstrap** est actif, le dossier des assets est `public/assets/bootstrap/`.

24.7 Compilation Css

Pour Bootstrap ou foundation, installez sass :

```
npm install -g sass
```

Ensuite, exécutez depuis le dossier racine du projet :

Pour bootstrap :

```
ssass public/assets/bootstrap/scss/app.scss public/assets/bootstrap/css/style.css --load-
↪path=vendor
```

Pour foundation :

```
ssass public/assets/foundation/scss/app.scss public/assets/foundation/css/style.css --
↪load-path=vendor
```

CHAPITRE 25

jQuery Semantic-UI

Par défaut, Ubiquity utilise la bibliothèque phpMv-UI pour la partie UI. **PhpMv-UI** permet de créer des composants basés sur Semantic-UI ou Bootstrap et de générer des scripts jQuery en PHP.

Cette bibliothèque est utilisée pour l'interface d'administration **webtools**.

25.1 Intégration

Par défaut, une variable **$jquery** est injectée dans les contrôleurs au moment de l'exécution.

Cette opération se fait par injection de dépendance, dans `app/config.php` :

Code source 1 – app/config.php

```
...
"di"=>array(
            "@exec"=>array(
                            "jquery"=>function ($controller){
                                    return \Ajax\php\ubiquity\JsUtils::diSemantic(
↪$controller);
                                    }
                            )
            )
...
```

Il n'y a donc rien à faire, mais pour faciliter son utilisation et permettre la complétion de code dans un contrôleur, il est recommandé d'ajouter la documentation de code suivante :

Code source 2 – app/controllers/FooController.php

```
/**
* Controller FooController
```

(suite sur la page suivante)

(suite de la page précédente)

```
 * @property \Ajax\php\ubiquity\JsUtils $jquery
 **/
class FooController extends ControllerBase{

	public function index(){}
}
```

25.2 jQuery

25.2.1 Href en requêtes ajax

Créez un nouveau contrôleur et sa vue associée, puis définissez les routes suivantes :

Code source 3 – app/controllers/FooController.php

```
namespace controllers;

class FooController extends ControllerBase {

	public function index() {
		$this->loadview("FooController/index.html");
	}

	/**
	 *
	 *@get("a","name"=>"action.a")
	 */
	public function aAction() {
		echo "a";
	}

	/**
	 *
	 *@get("b","name"=>"action.b")
	 */
	public function bAction() {
		echo "b";
	}
}
```

La vue associée :

Code source 4 – app/views/FooController/index.html

```
	<a href="{{path('action.a')}}">Action a</a>
	<a href="{{path('action.b')}}">Action b</a>
```

Initialiser le cache du routeur :

```
Ubiquity init:cache -t=controllers
```

Testez cette page dans votre navigateur à l'adresse `http://127.0.0.1:8090/FooController`.

Transformation des requêtes en requêtes Ajax

Le résultat de chaque requête ajax doit être affiché dans une zone de la page définie par son sélecteur jQuery (`.result span`).

Code source 5 – app/controllers/FooController.php

```php
namespace controllers;

/**
 * @property \Ajax\php\ubiquity\JsUtils $jquery
 */
class FooController extends ControllerBase {

    public function index() {
            $this->jquery->getHref('a','.result span');
            $this->jquery->renderView("FooController/index.html");
    }
    ...
}
```

Code source 6 – app/views/FooController/index.html

```
    <a href="{{path('action.a')}}">Action a</a>
    <a href="{{path('action.b')}}">Action b</a>
<div class='result'>
    Selected action:
    <span>No One</span>
</div>
{{ script_foot | raw }}
```

Note : La variable `script_foot` contient le script jquery généré par la méthode **renderView**. Le filtre **raw** marque la valeur comme étant « sûre », ce qui signifie que dans un environnement où l'échappement automatique est activé, cette variable ne sera pas échappée.

Ajoutons un peu de css pour le rendre plus professionnel :

Code source 7 – app/views/FooController/index.html

```
<div class="ui buttons">
    <a class="ui button" href="{{path('action.a')}}">Action a</a>
    <a class="ui button" href="{{path('action.b')}}">Action b</a>
</div>
<div class='ui segment result'>
    Selected action:
    <span class="ui label">No One</span>
</div>
{{ script_foot | raw }}
```

Si nous voulons ajouter un nouveau lien dont le résultat doit être affiché dans une autre zone, il est possible de le

spécifier via l'attribut **data-target**.

La nouvelle action

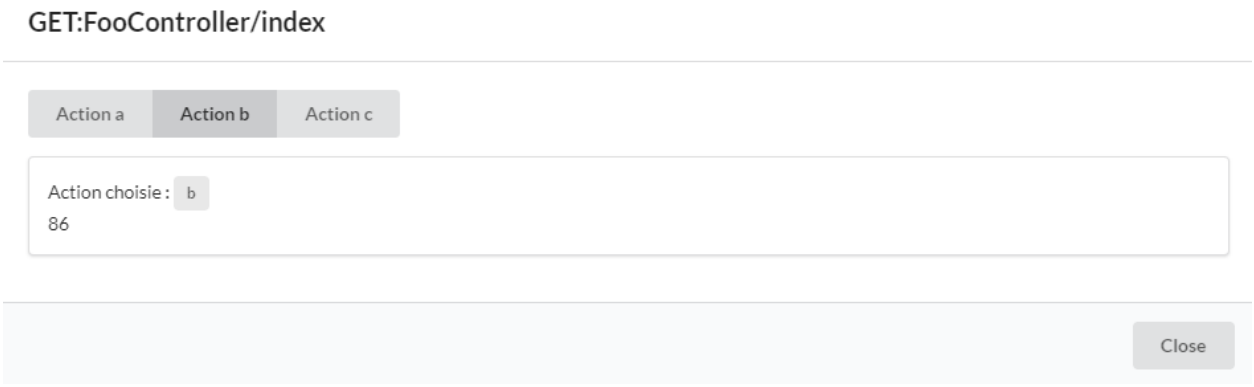
Code source 8 – app/controllers/FooController.php

```php
namespace controllers;

class FooController extends ControllerBase {
    ...
    /**
     *@get("c","name"=>"action.c")
     */
    public function cAction() {
            echo \rand(0, 1000);
    }
}
```

La vue associée :

Code source 9 – app/views/FooController/index.html

```
<div class="ui buttons">
    <a class="ui button" href="{{path('action.a')}}">Action a</a>
    <a class="ui button" href="{{path('action.b')}}">Action b</a>
    <a class="ui button" href="{{path('action.c')}}" data-target=".result p">Action c</
↪a>
</div>
<div class='ui segment result'>
    Selected action:
    <span class="ui label">No One</span>
    <p></p>
</div>
{{ script_foot | raw }}
```



Définition des attributs de la requête ajax :

Dans l'exemple suivant, les paramètres passés à la variable **attributs** de la méthode `getHref` :
- supprimer l'historique de la navigation,
- rendre le loader ajax interne au bouton cliqué.

Code source 10 – app/controllers/FooController.php

```php
namespace controllers;

/**
 * @property \Ajax\php\ubiquity\JsUtils $jquery
 */
class FooController extends ControllerBase {

	public function index() {
		$this->jquery->getHref('a','.result span', [
			'hasLoader' => 'internal',
			'historize' => false
		]);
		$this->jquery->renderView("FooController/index.html");
	}
	...
}
```

Note : Il est possible d'utiliser la méthode `postHref` pour utiliser le **POST** http.

25.2.2 Requêtes ajax classiques

Pour cet exemple, créez la base de données suivante :

```sql
CREATE DATABASE `uguide` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `uguide`;

CREATE TABLE `user` (
  `id` int(11) NOT NULL,
  `firstname` varchar(30) NOT NULL,
  `lastname` varchar(30) NOT NULL,
  `password` varchar(30) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `user` (`id`, `firstname`, `lastname`) VALUES
(1, 'You', 'Evan'),
(2, 'Potencier', 'Fabien'),
(3, 'Otwell', 'Taylor');

ALTER TABLE `user` ADD PRIMARY KEY (`id`);
ALTER TABLE `user`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

Connectez l'application à la base de données, et générez la classe *User* :

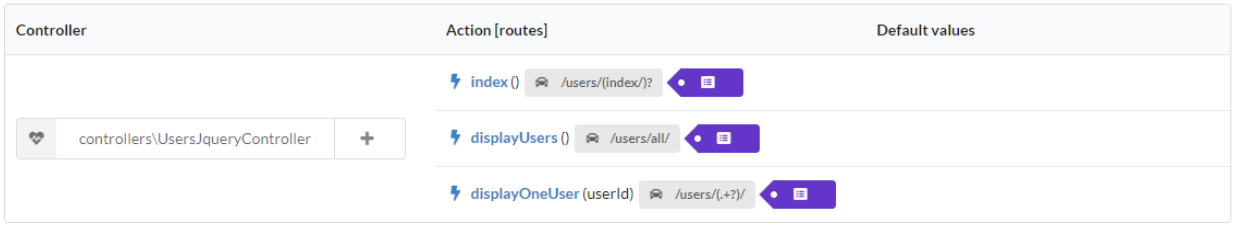
Avec les devtools :

```
Ubiquity config:set --database.dbName uguide
Ubiquity all-models
```

Créer un nouveau contrôleur *UsersJqueryController*.

```
Ubiquity controller UsersJqueryController -v
```

Créez les actions suivantes dans *UsersJqueryController* :



Action index

L'action *index* doit afficher un bouton pour obtenir la liste des utilisateurs, chargée via une requête ajax :

Code source 11 – app/controllers/UsersJqueryController.php

```
1  namespace controllers;
2
3  /**
4   * Controller UsersJqueryController
5   *
6   * @property \Ajax\php\ubiquity\JsUtils $jquery
7   * @route("users")
8   */
9  class UsersJqueryController extends ControllerBase {
10
11      /**
12       *
13       * {@inheritdoc}
14       * @see \Ubiquity\controllers\Controller::index()
15       * @get
16       */
17      public function index() {
18              $this->jquery->getOnClick('#users-bt', Router::path('display.users'), '
    ↪#users', [
19                      'hasLoader' => 'internal'
20              ]);
21              $this->jquery->renderDefaultView();
22      }
23 }
```

La vue par défaut associée à l'action *index* :

Code source 12 – app/views/UsersJqueryController/index.html

```
<div class="ui container">
    <div id="users-bt" class="ui button">
            <i class="ui users icon"></i>
            Display <b>users</b>
    </div>
    <p></p>
    <div id="users">
    </div>
</div>
{{ script_foot | raw }}
```

Action displayUsers

Tous les utilisateurs sont affichés, et un clic sur un utilisateur doit afficher les détails de l'utilisateur via une requête ajax postée :

Code source 13 – app/controllers/UsersJqueryController.php

```
1  namespace controllers;
2
3  /**
4   * Controller UsersJqueryController
5   *
6   * @property \Ajax\php\ubiquity\JsUtils $jquery
7   * @route("users")
8   */
9  class UsersJqueryController extends ControllerBase {
10 ...
11     /**
12      *
13      * @get("all","name"=>"display.users","cache"=>true)
14      */
15     public function displayUsers() {
16             $users = DAO::getAll(User::class);
17             $this->jquery->click('#close-bt', '$("#users").html("");');
18             $this->jquery->postOnClick('li[data-ajax]', Router::path('display.one.user',
   ↪ [
19                     ""
20             ]), '{}', '#user-detail', [
21                     'attr' => 'data-ajax',
22                     'hasLoader' => false
23             ]);
24             $this->jquery->renderDefaultView([
25                     'users' => $users
26             ]);
27     }
```

La vue associée à l'action *displayUsers* :

Code source 14 – app/views/UsersJqueryController/displayUsers.html

```
<div class="ui top attached header">
     <i class="users circular icon"></i>
     <div class="content">Users</div>
</div>
<div class="ui attached segment">
     <ul id='users-content'>
     {% for user in users %}
             <li data-ajax="{{user.id}}">{{user.firstname }} {{user.lastname}}</li>
     {% endfor %}
     </ul>
     <div id='user-detail'></div>
</div>
<div class="ui bottom attached inverted segment">
<div id="close-bt" class="ui inverted button">Close</div>
</div>
{{ script_foot | raw }}
```

Action displayOneUser

Code source 15 – app/controllers/UsersJqueryController.php

```
1  namespace controllers;
2
3  /**
4   * Controller UsersJqueryController
5   *
6   * @property \Ajax\php\ubiquity\JsUtils $jquery
7   * @route("users")
8   */
9  class UsersJqueryController extends ControllerBase {
10 ...
11      /**
12       *
13       * @post("{userId}","name"=>"display.one.user","cache"=>true,"duration"=>3600)
14       */
15      public function displayOneUser($userId) {
16              $user = DAO::getById(User::class, $userId);
17              $this->jquery->hide('#users-content', '', '', true);
18              $this->jquery->click('#close-user-bt', '$("#user-detail").html("");$("
   ↪#users-content").show();');
19              $this->jquery->renderDefaultView([
20                      'user' => $user
21              ]);
22      }
```

La vue associée à l'action *displayOneUser* :

Code source 16 – app/views/UsersJqueryController/displayUsers.html

```
<div class="ui label">
     <i class="ui user icon"></i>
     Id
     <div class="detail">{{user.id}}</div>
</div>
<div class="ui label">
     Firstname
     <div class="detail">{{user.firstname}}</div>
</div>
<div class="ui label">
     Lastname
     <div class="detail">{{user.lastname}}</div>
</div>
<p></p>
<div id="close-user-bt" class="ui black button">
     <i class="ui users icon"></i>
     Return to users
</div>
{{ script_foot | raw }}
```

25.3 Composants Semantic

Nous allons ensuite réaliser un contrôleur implémentant les mêmes fonctionnalités que précédemment, mais en utilisant des composants **PhpMv-UI** (partie sémantique).

25.3.1 Exemple HtmlButton

Créer un nouveau contrôleur *UsersJqueryController*.

```
Ubiquity controller UsersCompoController -v
```

Code source 17 – app/controllers/UsersJqueryController.php

```
1  namespace controllers;
2
3  use Ubiquity\controllers\Router;
4
5  /**
6   * Controller UsersCompoController
7   *
8   * @property \Ajax\php\ubiquity\JsUtils $jquery
9   * @route("users-compo")
10  */
11 class UsersCompoController extends ControllerBase {
12
13      private function semantic() {
14              return $this->jquery->semantic();
15      }
```

(suite sur la page suivante)

(suite de la page précédente)

```
16
17      /**
18       *
19       * @get
20       */
21      public function index() {
22              $bt = $this->semantic()->htmlButton('users-bt', 'Display users');
23              $bt->addIcon('users');
24              $bt->getOnClick(Router::path('display.compo.users'), '#users', [
25                      'hasLoader' => 'internal'
26              ]);
27              $this->jquery->renderDefaultView();
28      }
```

Note : L'appel à renderView ou renderDefaultView sur l'objet JQuery effectue la compilation du composant et génère le HTML et le JS correspondants.

La vue associée intègre le composant bouton avec le tableau *q* disponible dans la vue :

Code source 18 – app/views/UsersCompoController/index.html

```
<div class="ui container">
    {{ q['users-bt'] | raw }}
    <p></p>
    <div id="users">
    </div>
</div>
{{ script_foot | raw }}
```

//todo DataTable sample ++++++++++++++++++

CHAPITRE 26

Normalizers

Note : Le module Normalizer utilise la classe statique **NormalizersManager** pour gérer la normalisation.

CHAPITRE 27

Validateurs

Note : Le module Validators utilise la classe statique **ValidatorsManager** pour gérer la validation.

La validation permet de vérifier que les données membres d'un objet respectent certaines contraintes.

27.1 Ajout de validators

Soit la classe **Author** que nous souhaitons utiliser dans notre application :

Code source 1 – app/models/Author.php

```
1  namespace models;
2
3  class Author {
4          /**
5           * @var string
6           * @validator("notEmpty")
7           */
8          private $name;
9
10         public function getName(){
11                 return $this->name;
12         }
13
14         public function setName($name){
15                 $this->name=$name;
16         }
17 }
```

Une contrainte de validation a été ajoutée sur le membre **name** avec l'annotation **@validator**, de façon à rendre sa saisie obligatoire.

27.2 Génération du cache

Exécuter cette commande dans la console pour générer le cache de la classe **Author** :

```
Ubiquity init-cache -t=models
```

Le cache du validateur est généré dans le fichier `app/cache/contents/validators/models/Author.cache.php`.

27.3 Validation d'instances

27.3.1 Une instance

```
public function testValidateAuthor(){
        $author=new Author();
        //Do something with $author
        $violations=ValidatorsManager::validate($author);
        if(sizeof($violations)>0){
                echo implode('<br>', ValidatorsManager::validate($author));
        }else{
                echo 'The author is valid!';
        }
}
```

Si le **name** de l'instance de author est vide, l'action devrait afficher :

```
name : This value should not be empty
```

La méthode **validate** retroune un tableau d'instances de **ConstraintViolation** .

27.3.2 Plusieurs instances

```
public function testValidateAuthors(){
        $authors=DAO::getAll(Author::class);
        $violations=ValidatorsManager::validateInstances($author);
        foreach($violations as $violation){
                echo $violation.'<br>';
        }
}
```

27.4 Génération des models avec validateurs par défaut

Lorsque les classes sont générées automatiquement à partir de la base de données, des validateurs par défaut sont associés aux membres, basés su les méta-données des champs.

```
Ubiquity create-model User
```

Code source 2 – app/models/Author.php

```
1   namespace models;
2   class User{
3           /**
4            * @id
5            * @column("name"=>"id","nullable"=>false,"dbType"=>"int(11)")
6            * @validator("id","constraints"=>array("autoinc"=>true))
7           **/
8           private $id;
9
10          /**
11           * @column("name"=>"firstname","nullable"=>false,"dbType"=>"varchar(65)")
12           * @validator("length","constraints"=>array("max"=>65,"notNull"=>true))
13          **/
14          private $firstname;
15
16          /**
17           * @column("name"=>"lastname","nullable"=>false,"dbType"=>"varchar(65)")
18           * @validator("length","constraints"=>array("max"=>65,"notNull"=>true))
19          **/
20          private $lastname;
21
22          /**
23           * @column("name"=>"email","nullable"=>false,"dbType"=>"varchar(255)")
24           * @validator("email","constraints"=>array("notNull"=>true))
25           * @validator("length","constraints"=>array("max"=>255))
26          **/
27          private $email;
28
29          /**
30           * @column("name"=>"password","nullable"=>true,"dbType"=>"varchar(255)")
31           * @validator("length","constraints"=>array("max"=>255))
32          **/
33          private $password;
34
35          /**
36           * @column("name"=>"suspended","nullable"=>true,"dbType"=>"tinyint(1)")
37           * @validator("isBool")
38          **/
39          private $suspended;
40  }
```

Ces validateurs peuvent être modifiés. Les modifications doivent toujours être suivies d'une ré-initialisation du cache des models.

```
Ubiquity init-cache -t=models
```

Les informations relatives aux validateurs existants sur les models peuvent être affichées avec les devtools :

```
Ubiquity info:validation -m=User
```

```
· The project folder is C:\xampp7.3\htdocs\verif
```

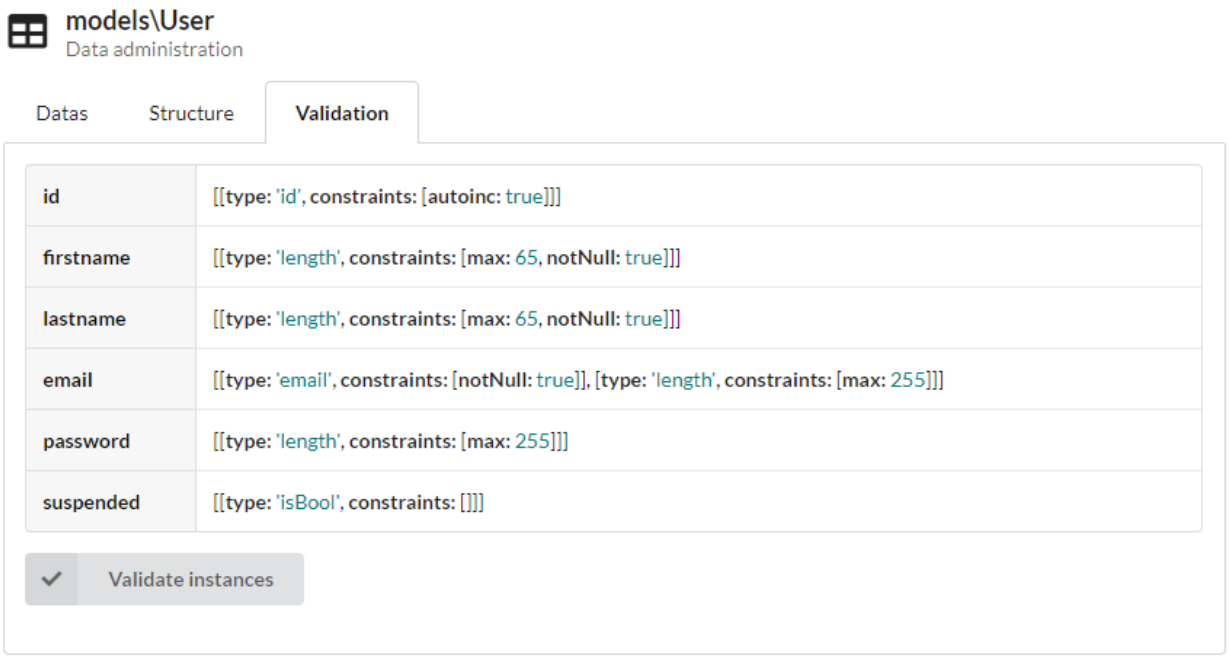
field	value
id	· [type: 'id',constraints: [autoinc: true]]
firstname lastname	· [type: 'length',constraints: [max: 65,notNull: true]] ≡
email	· [type: 'email',constraints: [notNull: true]] · [type: 'length',constraints: [max: 255]]
password	· [type: 'length',constraints: [max: 255]]
suspended	· [type: 'isBool',constraints: []]

Retourne les validateurs associés au champ email :

```
Ubiquity info:validation email -m=User
```

```
· email
  · type : 'email'
  · constraints : [notNull: true]
  · type : 'length'
  · constraints : [max: 255]
```

Les informations de validation sont également accessibles depuis la partie **models** des webtools :



27.5 Types de validators

27.5.1 Basic

Validator	Rôles	Contraintes	Valeurs acceptées
isBool	Vérifie si la valeur est un booléen		true,false,0,1
isEmpty	Vérifie si la valeur n'est pas vide		"",null
isFalse	Vérifie si la valeur est false		false,"false",0,"0"
isNull	Vérifie si la valeur est nulle		null
isTrue	Vérifie si la valeur est true		true,"true",1,"1"
notEmpty	Vérifie si la valeur n'est pas vide		!null && !""
notNull	Vérifie si la valeur n'est pas nulle		!null
type	Vérifie que la valeur est du type {type}	{type}	

27.5.2 Comparison

27.5.3 Dates

27.5.4 Multiples

27.5.5 Strings

CHAPITRE 28

Transformers

Note : Le module Transformers utilise la classe statique **TransformersManager** pour gérer les transformations.

Les Transformers sont utilisés pour transformer les données après leur chargement depuis une base de données, ou avant leur affichage dans une vue.

28.1 Ajouter des transformers

Soit la classe **Author** que nous souhaitons utiliser dans notre application :

Attributs

Code source 1 – app/models/Author.php

```php
namespace models;

use Ubiquity\attributes\items\Transformer;

class Author {

    #[Transformer('upper')]
    private $name;

    public function getName(){
        return $this->name;
    }

    public function setName($name){
        $this->name=$name;
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }
}
```

Annotations

Code source 2 – app/models/Author.php

```php
namespace models;

class Author {
    /**
     * @var string
     * @transformer("upper")
     */
    private $name;

    public function getName(){
        return $this->name;
    }

    public function setName($name){
        $this->name=$name;
    }
}
```

Un transformer a été ajouté sur le membre **name** avec l'annotation **@transformer** , de façon à mettre en majuscules le nom dans les vues.

28.2 Génération du cache

Exécuter la commande suivante dans la console pour générer le cache de la classe **Author** :

```
Ubiquity init-cache -t=models
```

Le cache des transformers est généré avec les méta-données des models dans le dossier `app/cache/models/Author.cache.php`.

Les informations relatives aux transormers peuvent être affichées à partir des devtools :

```
Ubiquity info:model -m=Author -f=#transformers
```

field	value
#transformers	· toView : [name: 'Ubiquity\\contents\\transformation\\transformers\\UpperCase']

28.3 Utilisation des transformers

Démarrer le manager **TransformersManager** depuis le fichier *app/config/services.php* :

Code source 3 – app/config/services.php

```
\Ubiquity\contents\transformation\TransformersManager::startProd();
```

Le résultat peut être testé depuis l'interface d'administration :

Id	Name
1	JOHN GRISHAM
2	JOANNE ROWLING
3	STEPHEN EDWIN KING

ou en créant un contrôleur

Code source 4 – app/controllers/Authors.php

```
1  namespace controllers;
2
3  class Authors {
4
5     public function index(){
6        DAO::transformersOp='toView';
7        $authors=DAO::getAll(Author::class);
8        $this->loadDefaultView(['authors'=>$authors]);
9     }
10
11 }
```

Code source 5 – app/views/Authors/index.html

```
<ul>
   {% for author in authors %}
      <li>{{ author.name }}</li>
   {% endfor %}
</ul>
```

28.4 Types de transformers

28.4.1 transform

Le type **transform** est basé sur l'interface **TransformerInterface**. Il est utilisé lorsque les données doivent être transformées en objet. Le transormer **DateTime** est un bon exemple d'un tel transformer :

— Au chargement des données, le transformer convertit les données de la base en une instance de php DateTime.
— La méthode **reverse** effectue l'opération inverse (php date vers date compatible avec la base de données).

28.4.2 toView

Le type **toView** est basé sur l'interface **TransformerViewInterface**. Il est utilisé dans le cas où les données transformées doivent être affichées dans une vue.

28.4.3 toForm

Le type **toForm** est basé sur l'interface **TransformerFormInterface**. Il est utilisé lorsque les données doivent être affichées dans un formulaire.

28.5 Utilisation des transformers

28.5.1 Transformation sur chargement des données

En cas d'omission, l'opération par défaut **transformerOp** est **transform**

```
$authors=DAO::getAll(Author::class);
```

Définit transformerOp à **toView**

```
DAO::transformersOp='toView';
$authors=DAO::getAll(Author::class);
```

28.5.2 Transformation après chargement

Retourne la valeur du membre transformée

```
TransformersManager::transform($author, 'name','toView');
```

Retourne une valeur transformée :

```
TransformersManager::applyTransformer($author, 'name','john doe','toView');
```

Transforme une instance en lui appliquant tous les transformers définis :

```
TransformersManager::transformInstance($author,'toView');
```

28.6 Transformers existants

Transformer	Type(s)	Description
datetime	transform, toView, toForm	Transforme une date issue de la base de données en une instance de DateTime php
upper	toView	Met la valeur du membre en majuscule
lower	toView	Met la valeur du membre en minuscule
firstUpper	toView	Met une majuscule sur la première lettre de la valeur du membre
password	toView	Masque les caractères de la valeur du membre
md5	toView	Hash la valeur en md5

28.7 Créer votre propre transformer

28.7.1 Création

Crée un transformer pour afficher un nom d'utilisateur comme une adresse email locale :

Code source 6 – app/transformers/toLocalEmail.php

```
1  namespace transformers;
2  use Ubiquity\contents\transformation\TransformerViewInterface;
3
4  class ToLocalEmail implements TransformerViewInterface{
5
6     public static function toView($value) {
7        if($value!=null) {
8           return sprintf('%s@mydomain.local',strtolower($value));
9        }
10    }
11
12 }
```

28.7.2 Enregistrement

Enregistrer le transformer en exécutant le script suivant :

```
TransformersManager::registerClassAndSave('localEmail',\transformers\
↪ToLocalEmail::class);
```

28.7.3 Utilisation

Attributs

Code source 7 – app/models/User.php

```
1  namespace models;
2
3  use Ubiquity\attributes\items\Transformer;
4
5  class User {
6
7     #[Transformer('localEmail')]
8     private $name;
9
10    public function getName(){
11       return $this->name;
12    }
13
14    public function setName($name){
15       $this->name=$name;
16    }
17 }
```

Annotations

Code source 8 – app/models/User.php

```
1  namespace models;
2
3  class User {
4     /**
5      * @var string
6      * @transformer("localEmail")
7      */
8     private $name;
9
10    public function getName(){
11       return $this->name;
12    }
13
14    public function setName($name){
15       $this->name=$name;
16    }
17 }
```

```
DAO::transformersOp='toView';
$user=DAO::getOne(User::class,"name='Smith'");
echo $user->getName();
```

Smith user name will be displayed as **smith@mydomain.local**.

CHAPITRE 29

Module de traduction

Note : Le module de traduction utilise la classe statique **TranslatorManager** pour gérer les traductions.

29.1 Structure du module

Les traductions sont regroupées par **domaine**, dans une **locale** :

Dans le répertoire racine des traductions (par défaut **app/translations**) :

- Chaque locale correspond à un sous-dossier.
- Pour chaque locale, dans un sous-dossier, un domaine correspond à un fichier php.

```
translations
     ├ en_EN
     │     ├ messages.php
     │     └ blog.php
     └ fr_FR
           ├ messages.php
           └ blog.php
```

- chaque fichier de domaine contient un tableau associatif de traductions **key-> valeur de la traduction**.
- **Chaque clé peut être associée à**
  - une traduction
  - une traduction contenant des variables (entre % et %)
  - un tableau de traductions pour la pluralisation

Code source 1 – app/translations/en_EN/messages.php

```
return [
     'okayBtn'=>'Okay',
     'cancelBtn'=>'Cancel',
```

(suite sur la page suivante)

(suite de la page précédente)

```
    'deleteMessage'=>['No message to delete!','1 message to delete.','%count% messages␣
↪to delete.']
];
```

29.2 Démarrage du module

Le démarrage du module se fait logiquement dans le fichier **services.php**.

Code source 2 – app/config/services.php

```
1 Ubiquity\cache\CacheManager::startProd($config);
2 Ubiquity\translation\TranslatorManager::start();
```

En l'absence de paramètres, l'appel de la méthode **start** utilise la locale **en_EN**, sans fallbacklocale.

Important : Le module de traduction doit être lancé après le démarrage du cache.

29.2.1 Définition de la locale

Changement de la locale au démarrage du gestionnaire :

Code source 3 – app/config/services.php

```
1 Ubiquity\cache\CacheManager::startProd($config);
2 Ubiquity\translation\TranslatorManager::start('fr_FR');
```

Changement de la locale après le chargement du gestionnaire :

```
TranslatorManager::setLocale('fr_FR');
```

29.2.2 Définition de la fallbackLocale

La locale **fr_EN** sera utilisée si **fr_FR** n'est pas trouvée :

Code source 4 – app/config/services.php

```
1 Ubiquity\cache\CacheManager::startProd($config);
2 Ubiquity\translation\TranslatorManager::start('fr_FR','en_EN');
```

29.3 Définition du répertoire racine des traductions

Si le paramètre **rootDir** est absent, le répertoire utilisé par défaut est « app/translations ».

Code source 5 – app/config/services.php

```
1 Ubiquity\cache\CacheManager::startProd($config);
2 Ubiquity\translation\TranslatorManager::start('fr_FR','en_EN','myTranslations');
```

29.4 Faire une traduction

29.4.1 Avec php

Traduction de la clé **okayBtn** dans la locale par défaut (spécifiée lors du démarrage du gestionnaire) :

```
$okBtnCaption=TranslatorManager::trans('okayBtn');
```

Sans paramètres, l'appel de la méthode **trans** utilise la locale par défaut, le domaine **messages**.

Traduction de la clé **message** utilisant une variable :

```
$okBtnCaption=TranslatorManager::trans('message',['user'=>$user]);
```

Dans ce cas, le fichier de traduction doit contenir une référence à la variable **user** pour la clé **message** :

Code source 6 – app/translations/en_EN/messages.php

```
['message'=>'Hello %user%!',...];
```

29.4.2 Dans les vues twig :

Traduction de la clé **okayBtn** dans la locale par défaut (spécifiée lors du démarrage du gestionnaire) :

```
{{ t('okayBtn') }}
```

Traduction de la clé **message** utilisant une variable :

```
{{ t('message',parameters) }}
```

CHAPITRE 30

Sécurité

30.1 Principes directeurs

30.1.1 Validation des formulaires

Validation côté client

Il est préférable d'effectuer une validation initiale côté client pour éviter de soumettre des données invalides au serveur.

Exemple de la création d'un formulaire dans l'action d'un contrôleur (cette partie pourrait être située dans un service dédié pour une meilleure séparation des couches) :

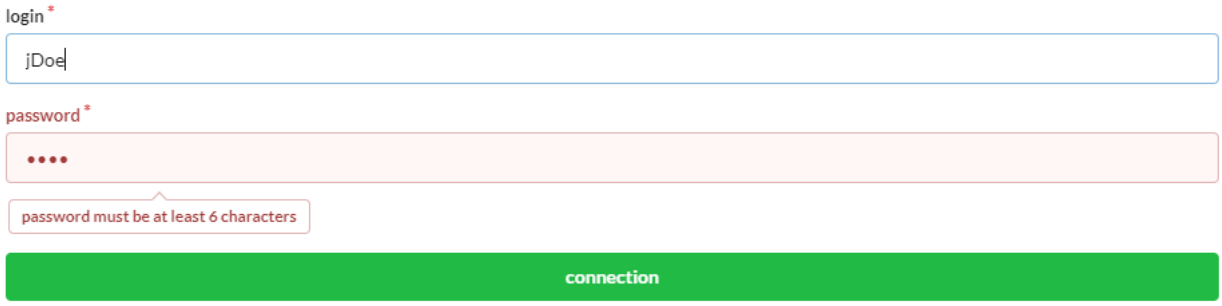
Code source 1 – app/controllers/UsersManagement.php

```php
public function index(){
    $frm=$this->jquery->semantic()->dataForm('frm-user',new User());
    $frm->setFields(['login','password','connection']);
    $frm->fieldAsInput('login',
        ['rules'=>'empty']
    );
    $frm->fieldAsInput('password',
        [
            'inputType'=>'password',
            'rules'=>['empty','minLength[6]']
        ]
    );
    $frm->setValidationParams(['on'=>'blur','inline'=>true]);
    $frm->fieldAsSubmit('connection','fluid green','/submit','#response');
    $this->jquery->renderDefaultView();
}
```

Ma vue associée :

Code source 2 – app/views/UsersManagement/index.html

```
{{ q['frm-user'] | raw }}
{{ script_foot | raw }}
<div id="response"></div>
```



Note : Les contrôleurs CRUD intègrent automatiquement cette validation côté client en utilisant les validateurs attachés aux membres des modèles.

```
#[Column(name: "password",nullable: true,dbType: "varchar(255)")]
#[Validator(type: "length",constraints: ["max"=>20,"min"=>6])]
#[Transformer(name: "password")]
private $password;
```

Validation côté serveur

Il est préférable de restreindre les URLs autorisées à modifier les données. Au préalable, en spécifiant la méthode Http dans les routes, et en testant la requête :

```
#[Post(path: "/submit")]
public function submitUser(){
   if(!URequest::isCrossSite() && URequest::isAjax()){
      $datas=URequest::getPost();//post with htmlEntities
      //Do something with $datas
   }
}
```

Note : Le module **Ubiquity-security** offre un contrôle supplémentaire pour éviter les requêtes intersites.

Après avoir modifié un objet, il est possible de vérifier sa validité, grâce aux validateurs attachés aux membres du Modèle associé :

```
#[Post(path: "/submit")]
public function submitUser(){
   if(!URequest::isCrossSite()){
      $datas=URequest::getPost();//post with htmlEntities
      $user=new User();
```

(suite sur la page suivante)

(suite de la page précédente)

```
        URequest::setValuesToObject($user,$datas);

        $violations=ValidatorsManager::validate($user);
        if(\count($violations)==0){
            //do something with this valid user
        } else {
            //Display violations...
        }
    }
}
```

30.1.2 Opérations DAO

Il est toujours recommandé d'utiliser des requêtes paramétrées, quelles que soient les opérations effectuées sur les données :

- Pour éviter les injections SQL.
- Pour permettre l'utilisation de requêtes préparées, accélérant le traitement.

```
$googleUsers=DAO::getAll(User::class,'email like ?',false,['%@gmail.com']);
```

```
$countActiveUsers=DAO::count(User::class,'active= ?',[true]);
```

Note : Les opérations DAO qui prennent des objets comme paramètres utilisent ce mécanisme par défaut.

```
DAO::save($user);
```

30.1.3 Gestion des mots de passe

Le transformateur `Password` permet à un champ d'être du type mot de passe lorsqu'il est affiché dans un formulaire CRUD généré automatiquement.

```
#[Transformer(name: "password")]
private $password;
```

Après soumission d'un formulaire, il est possible de crypter un mot de passe à partir de la classe URequest :

```
$encryptedPassword=URequest::password_hash('password');
$user->setPassword($encryptedPassword);
DAO::save($user);
```

L'algorithme utilisé dans ce cas est défini par le paramètre php `PASSWORD_DEFAULT`.

Il est également possible de vérifier de la même manière un mot de passe saisi par un utilisateur, en le comparant à un hachage :

```
if(URequest::password_verify('password', $existingPasswordHash)){
    //password is ok
}
```

Important : Configurez Https pour éviter d'envoyer les mots de passe en clair.

30.2 Module de sécurité/ Gestion des ACL

En plus de ces quelques règles, vous pouvez procéder à des installations si nécessaire :

- *Ubiquity-acl*
- *Ubiquity-security*

CHAPITRE 31

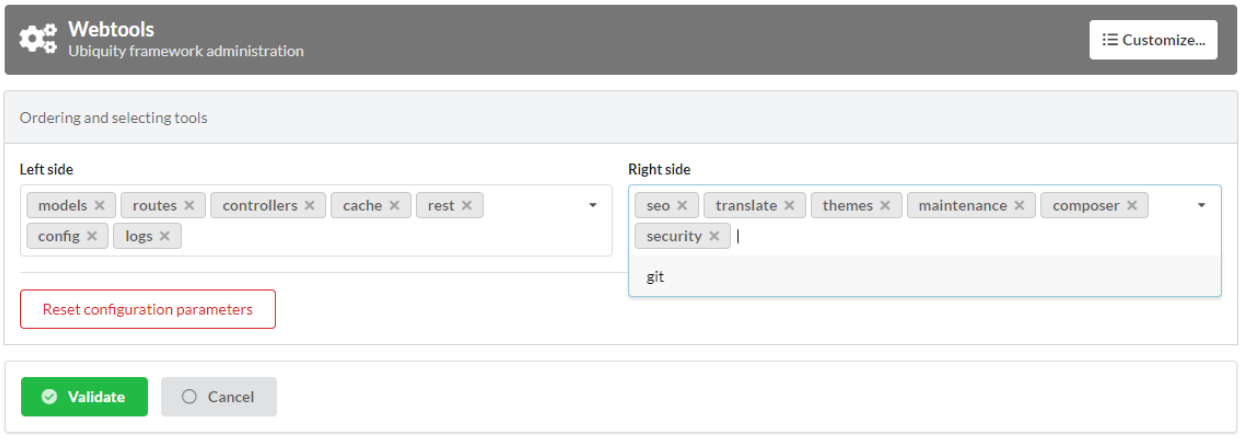
Module sécurité

31.1 Installation

Installer le module Ubiquity-security à partir de l'invite de commande ou des **Webtools** (partie Composer).

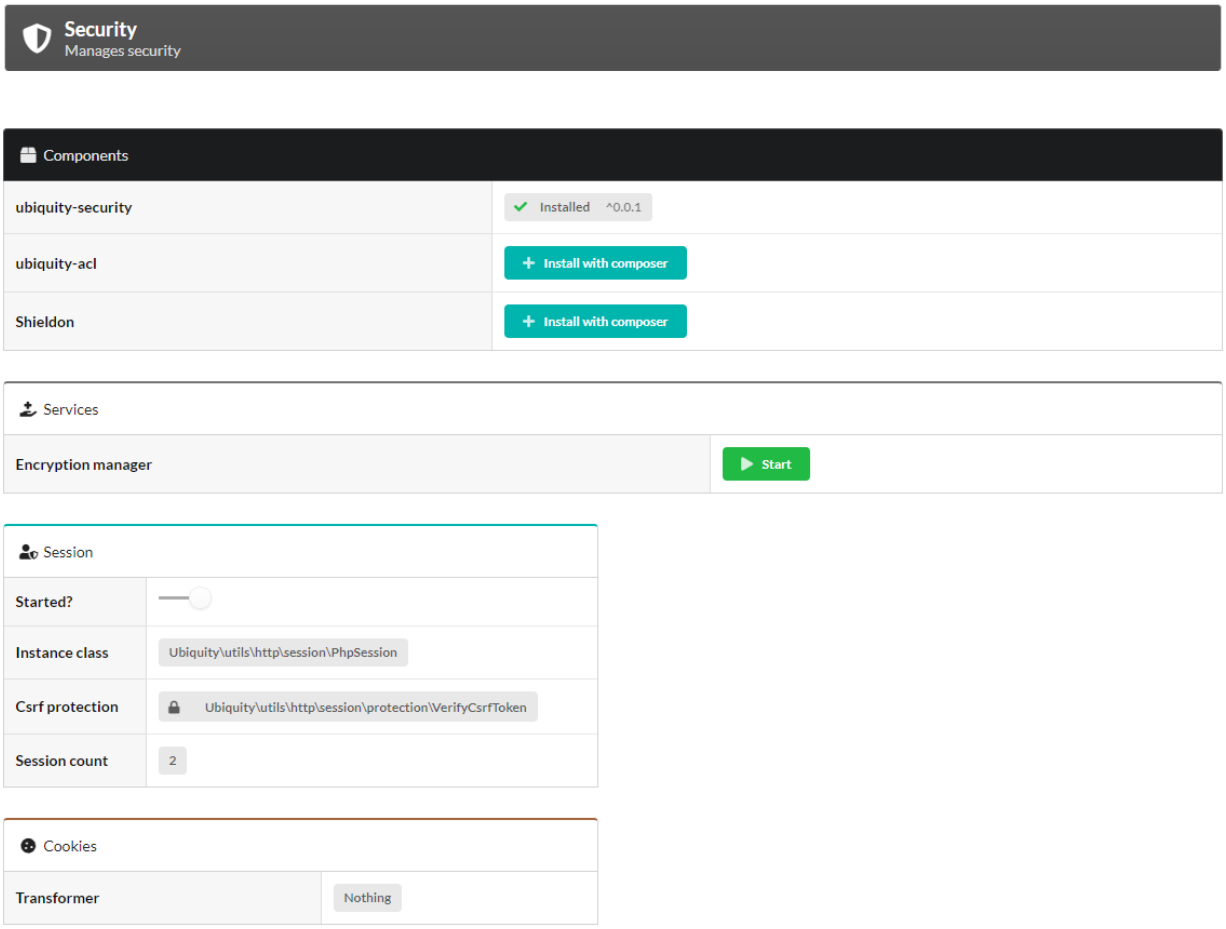
```
composer require phpmv/ubiquity-security
```

Activez ensuite l'affichage de la partie Sécurité dans les **Webtools** :



31.2 Session CSRF

La session est par défaut protégée contre les attaques CSRF via la classe `VerifyCsrfToken` (même sans le module **Ubiquity-security**). |Une instance de jeton (`CSRFToken`) est générée au démarrage de la session. La validité du jeton est ensuite vérifiée via un cookie à chaque requête.



Cette protection peut être personnalisée en créant une classe implémentant `VerifySessionCsrfInterface`.

Code source 1 – app/session/MyCsrfProtection.php

```
class MyCsrfProtection implements VerifySessionCsrfInterface {
   private AbstractSession $sessionInstance;

   public function __construct(AbstractSession $sessionInstance) {
      $this->sessionInstance = $sessionInstance;
   }

   public function init() {
      //TODO when the session starts
   }

   public function clear() {
      //TODO when the session ends
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }

    public function start() {
        //TODO When the session starts or is resumed
    }

    public static function getLevel() {
        return 1; //An integer to appreciate the level of security
    }
}
```

Démarrer la protection personnalisée dans les services :

Code source 2 – app/config/services.php

```
use Ubiquity\utils\http\session\PhpSession;
use Ubiquity\controllers\Startup;
use app\session\MyCsrfProtection;

Startup::setSessionInstance(new PhpSession(new MyCsrfProtection()));
```

31.2.1 Désactiver la protection

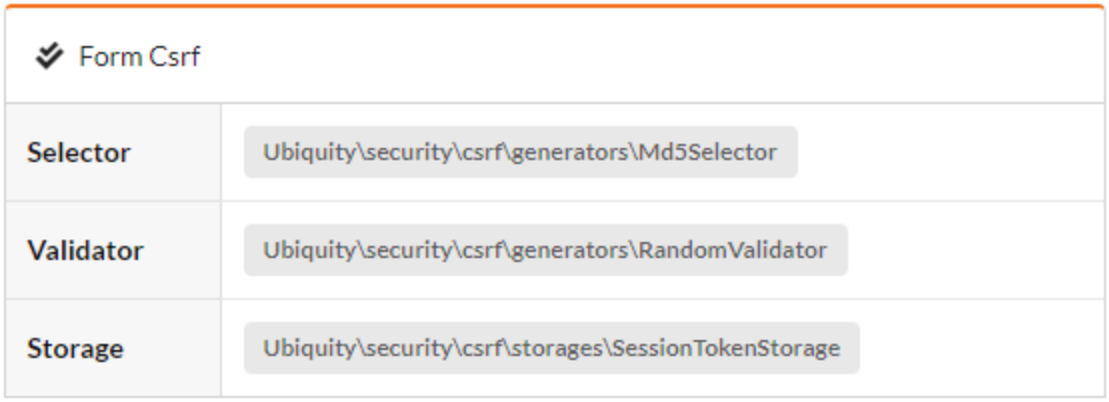
Si vous n'avez pas besoin de protéger votre session contre les attaques Csrf, démarrez la session avec la classe « NoCsrfProtection ».

Code source 3 – app/config/services.php

```
use Ubiquity\utils\http\session\PhpSession;
use Ubiquity\controllers\Startup;
use Ubiquity\utils\http\session\protection\NoCsrfProtection;

Startup::setSessionInstance(new PhpSession(new NoCsrfProtection()));
```

31.3 Gestion CSRF

Le service **CsrfManager** peut être démarré directement depuis l'interface **webtools**. |Son rôle est de fournir des outils pour protéger les routes sensibles des attaques Csrf (celles qui permettent la validation des formulaires par exemple).



— Le service est démarré dans le fichier `services.php`.

Code source 4 – app/config/services.php

```
\Ubiquity\security\csrf\CsrfManager::start();
```

31.3.1 Exemple de protection de formulaire :

La vue du formulaire :

```
<form id="frm-bar" action='/submit' method='post'>
   {{ csrf('frm-bar') }}
   <input type='text' id='sensitiveData' name='sensitiveData'>
</form>
```

La méthode `csrf` génère un jeton pour le formulaire (En ajoutant un champ caché dans le formulaire correspondant au jeton.).

La soumission du formulaire dans un contrôleur :

```
use Ubiquity\security\csrf\UCsrfHttp;

#[Post('/submit')]
public function submit(){
   if(UCsrfHttp::isValidPost('frm-bar')){
      //Token is valid! => do something with post datas
   }
}
```

Note : Il est également possible de gérer cette protection via un cookie.

31.3.2 Exemple de protection avec ajax :

Le champ méta `csrf-token` est généré sur toutes les pages.

Code source 5 – app/controllers/BaseController.php

```
abstract class ControllerBase extends Controller{
   protected $headerView = "@activeTheme/main/vHeader.html";
   protected $footerView = "@activeTheme/main/vFooter.html";

   public function initialize() {
      if (! URequest::isAjax ()) {
         $meta=UCsrfHttp::getTokenMeta('postAjax');
         $this->loadView ( $this->headerView,['meta'=>$meta] );
      }
   }
}
```

Ce champ est ajouté dans la headerView :

Code source 6 – app/views/main/vHeader.html

```
{% block header %}
   <base href="{{config["siteUrl"]}}">
   <meta charset="UTF-8">
   <link rel="icon" href="data:;base64,iVBORw0KGgo=">
   {{meta | raw}}
   <title>Tests</title>
{% endblock %}
```

Exemple avec un bouton qui envoie des données via ajax. Le paramètre `csrf` est mis à true. Donc quand la requête est postée, le `csrf-token` est envoyé dans les headers de la requête.

```
#[Get(path: "/ajax")]
public function ajax(){
   $this->jquery->postOnClick('#bt','/postAjax','{id:55}','#myResponse',['csrf'=>true]);
   $this->jquery->renderDefaultView();
}
```

La route de soumission peut vérifier la présence et la validité du jeton :

```
#[Post(path: "postAjax")]
public function postAjax(){
   if(UCsrfHttp::isValidMeta('postAjax')){
      var_dump($_POST);
   }else{
      echo 'invalid or absent meta csrf-token';
   }
}
```

31.4 Gestionnaire de cryptage

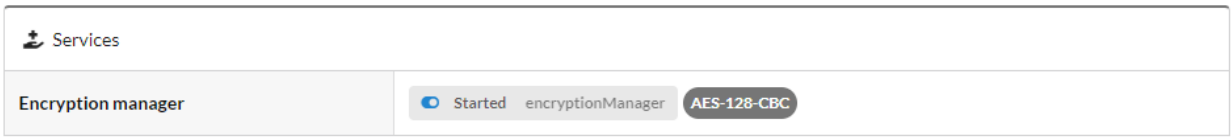
Le service **EncryptionManager** peut être lancé directement à partir de l'interface **webtools**.

— Dans ce cas, une clé est générée dans le fichier de configuration « app/config/config.php ».
— Le service est démarré dans le fichier `services.php`.

Code source 7 – app/config/services.php

```
\Ubiquity\security\data\EncryptionManager::start($config);
```

Note : Par défaut, le cryptage est effectué en `AES-128`.



31.4.1 Changement du cypher code :

Mise à niveau vers AES-256 :

Code source 8 – app/config/services.php

```
\Ubiquity\security\data\EncryptionManager::startProd($config, Encryption::AES256);
```

Générer une nouvelle clé :

```
Ubiquity new:key 256
```

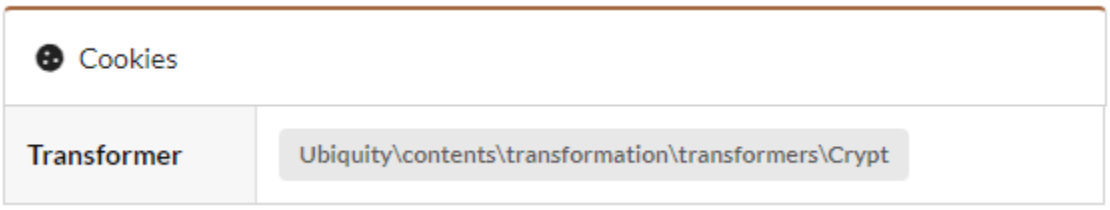
La nouvelle clé est générée dans le fichier `app/config/config.php`.

Cryptage des cookies

Les cookies peuvent être chiffrés par défaut, en ajoutant ceci dans `services.php` :

Code source 9 – app/config/services.php

```
use Ubiquity\utils\http\UCookie;
use Ubiquity\contents\transformation\transformers\Crypt;

UCookie::setTransformer(new Crypt());
```



Cryptage des données des modèles

Le transformateur `Crypt` peut aussi être utilisé sur les membres d'un modèle :

Code source 10 – app/models/User.php

```
class Foo{
    #[Transformer(name: "crypt")]
    private $secret;
    ...
}
```

Usage :

```
$o=new Foo();
$o->setSecret('bar');
TransformersManager::transformInstance($o);// secret member is encrypted
```

Cryptage des données génériques

Cryptage des chaînes :

```
$encryptedBar=EncryptionManager::encryptString('bar');
```

Pour ensuite le décrypter :

```
echo EncryptionManager::decryptString($encryptedBar);
```

Il est possible de crypter tout type de données :

```
$encryptedUser=EncryptionManager::encrypt($user);
```

Pour ensuite le décrypter, avec éventuellement sérialisation/désérialisation s'il s'agit d'un objet :

```
$user=EncryptionManager::decrypt($encryptedUser);
```

31.5 Gestionnaire des politiques de sécurité du contenu

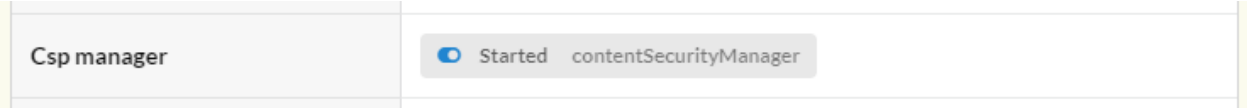
Le service **ContentSecurityManager** peut être lancé directement à partir de l'interface **webtools**.

— Le service est démarré dans le fichier `services.php`.

Code source 11 – app/config/services.php

```
\Ubiquity\security\csp\ContentSecurityManager::start(reportOnly: true,onNonce: function(
↪$name,$value){
    if($name==='jsUtils') {
            \Ubiquity\security\csp\ContentSecurityManager::defaultUbiquityDebug()->
↪addNonce($value, \Ubiquity\security\csp\CspDirectives::SCRIPT_SRC)->
↪addHeaderToResponse();
    }
});
```

Note : Avec cette configuration par défaut, un nonce est ajouté aux scripts jquery générés avec phpmv-ui. Le contrôle du CSP se fait en mode Report-only .



31.5.1 Ajout d'un nonce

Exemple d'ajout de nonce sur les pages d'en-tête et de pied de page :

Mise à jour du contrôleur de base

Code source 12 – app/controllers/ControllerBase.php

```php
namespace controllers;

use Ubiquity\controllers\Controller;
use Ubiquity\security\csp\ContentSecurityManager;
use Ubiquity\utils\http\URequest;

/**
 * controllers$ControllerBase
 */
abstract class ControllerBase extends Controller {

	protected $headerView = "@activeTheme/main/vHeader.html";

	protected $footerView = "@activeTheme/main/vFooter.html";

	protected $nonce;

	public function initialize() {
		$this->nonce=ContentSecurityManager::getNonce('jsUtils');
		if (! URequest::isAjax()) {
			$this->loadView($this->headerView,['nonce'=>$this->nonce]);
		}
	}

	public function finalize() {
		if (! URequest::isAjax()) {
			$this->loadView($this->footerView,['nonce'=>$this->nonce]);
		}
	}
}
```

Ajout du nonce dans les vues d'en-tête et de pied de page

Code source 13 – app/views/main/vHeader.html

```
{% block css %}
    {{ css('https://cdn.jsdelivr.net/npm/fomantic-ui@2.8.8/dist/semantic.min.css',[
↪'nonce'=>nonce]) }}
    {{css('css/style.css',['nonce'=>nonce])}}
{% endblock %}
```

Code source 14 – app/views/main/vFooter.html

```
{% block scripts %}
    {{ js('https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js',['nonce'=>
↪nonce]) }}
    {{ js('https://cdn.jsdelivr.net/npm/fomantic-ui@2.8.8/dist/semantic.min.js',['nonce
↪'=>nonce]) }}
{% endblock %}
```

31.6 Gestion des mots de passe

31.7 Jeton d'utilisateurs

CHAPITRE 32

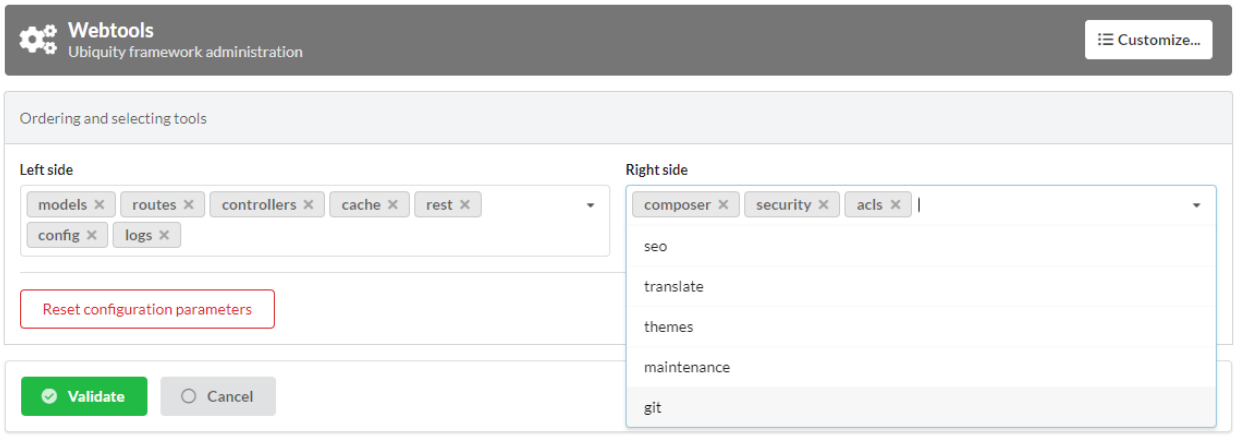
Gestion des ACL

32.1 Installation

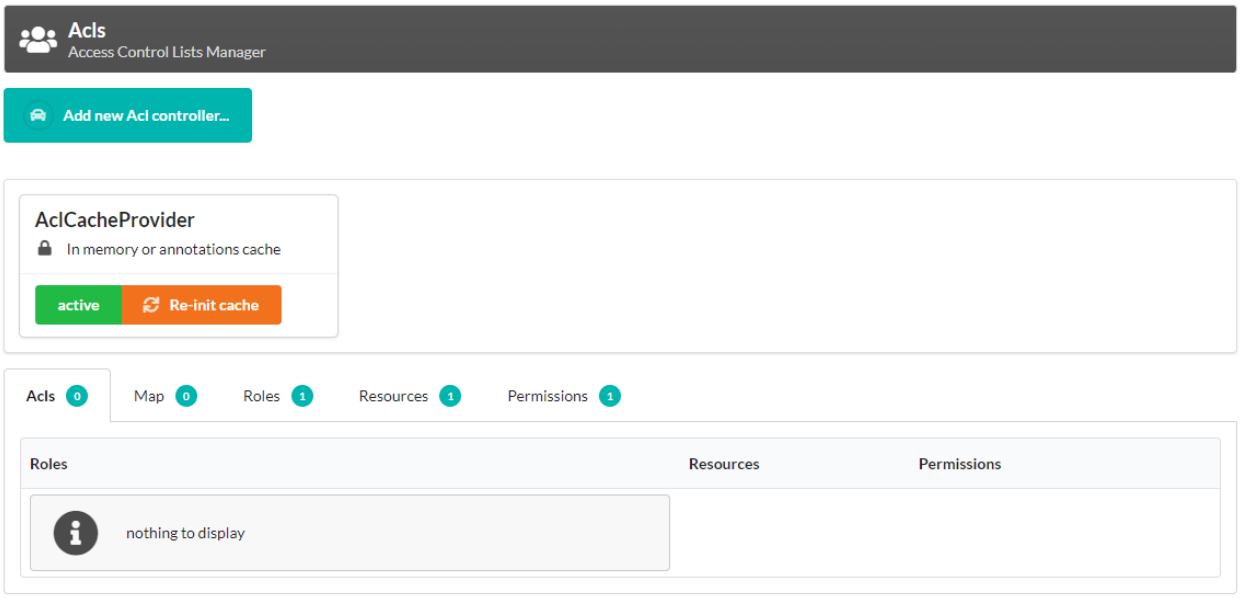
Installez le module **Ubiquity-acl** à partir de l'invite de commande ou des **Webtools** (partie Composer).

```
composer require phpmv/ubiquity-acl
```

Activez ensuite l'affichage de la partie Acl dans les **Webtools** :



Interface ACL dans les **webtools** :



32.2 Règles Acl

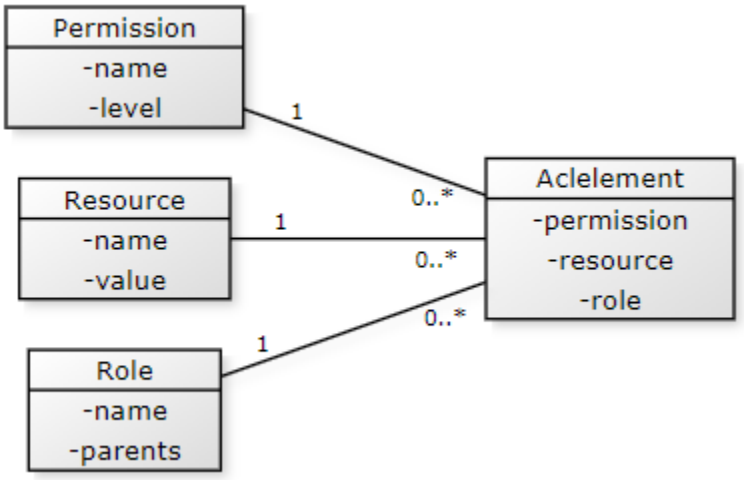
Les ACLs sont utilisées pour définir l'accès à une application Ubiquity. Elles sont définies selon les principes suivants :

Une application Ubiquity est composée de :

- **Ressources** (éventuellement des contrôleurs, ou des actions de ces contrôleurs)
- **Rôles**, éventuellement attribués à des utilisateurs. Chaque **Rôle** peut hériter de rôles parents.
- **Permissions**, qui correspondent à un droit de faire. Chaque permission a un niveau (représenté par une valeur entière).

Règles supplémentaires :

- Un AclElement (**Allow**) accorde une permission à un rôle sur une ressource.
- Chaque rôle hérite des autorisations de ses parents, en plus des siennes.
- Si un rôle a un certain niveau d'autorisation d'accès à une ressource, il aura également toutes les autorisations d'un niveau inférieur sur cette ressource.
- L'association d'une ressource et d'une permission à un contrôleur ou à une action de contrôleur définit un élément **map**.



Règles de nommage :

— Rôle, en lettres capitales, commençant par une arobase (@USER, @ADMIN, @ALL…).
— Permissions, en majuscules, nommées à l'aide d'un verbe (READ, WRITE, OPEN…).
— Ressource, majuscule à la première lettre (Produits, Clients…)

32.3 Démarrage des ACLs

Le service **AclManager** peut être démarré directement depuis l'interface **webtools**, dans la partie **Security**.

— Le service est démarré dans le fichier `services.php`.

Code source 1 – app/config/services.php

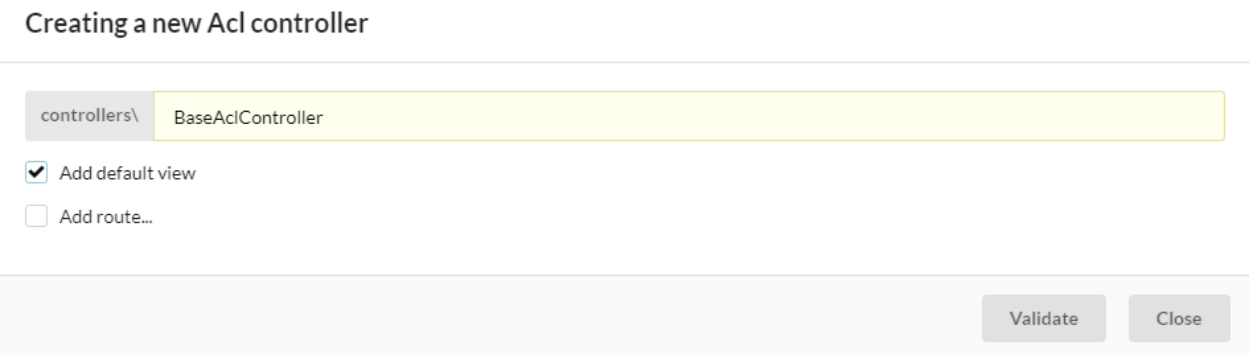
```
\Ubiquity\security\acl\AclManager::startWithCacheProvider();
```

32.3.1 ACLCacheProvider

Ce fournisseur par défaut vous permet de gérer les ACL définies par des attributs ou des annotations.

AclController

Un AclController permet de gérer automatiquement les accès à ses propres ressources en se basant sur des ACL. Il est possible de les créer automatiquement à partir des **webtools**.



Mais il ne s'agit que d'un contrôleur de base, utilisant le trait AclControllerTrait.

Ce contrôleur va juste redéfinir la méthode `_getRole`, pour qu'elle renvoie le rôle de l'utilisateur actif, par exemple.

Code source 2 – app/controllers/BaseAclController.php

```
<?php
namespace controllers;

use Ubiquity\controllers\Controller;
use Ubiquity\security\acl\controllers\AclControllerTrait;
use Ubiquity\attributes\items\acl\Allow;

class BaseAclController extends Controller {
use AclControllerTrait;

```

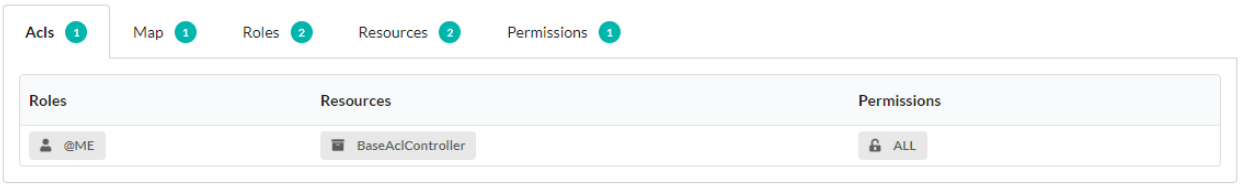
(suite sur la page suivante)

(suite de la page précédente)

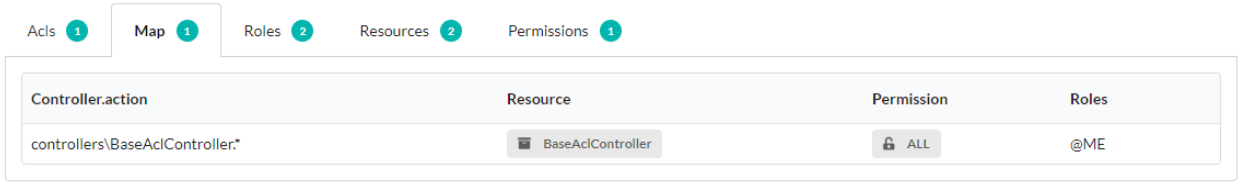
```
    #[Allow('@ME')]
    public function index() {
        $this->loadView("BaseAclController/index.html");
    }

    public function _getRole() {
        $_GET['role']??'@ME';//Just for testing: logically, this is the active user's role
    }

    /**
     * {@inheritdoc}
     * @see \Ubiquity\controllers\Controller::onInvalidControl()
     */
    public function onInvalidControl() {
        echo $this->_getRole() . ' is not allowed!';
    }
}
```

L'autorisation a été accordée pour la ressource :

— Sans spécifier la ressource, chaque action du contrôleur est définie comme une ressource.
— Sans spécifier la permission, la permission « ALL » est utilisée.



Et cette association est présente dans la map des Acls :



AclController avec authentication

Note : L'utilisation à la fois de `WithAuthTrait` et de ``AclControllerTrait`` nécessite de lever l'ambiguïté sur la méthode ``isValid``.

Code source 3 – app/controllers/BaseAclController.php

```
class BaseAclController extends Controller {
    use AclControllerTrait,WithAuthTrait{
        WithAuthTrait::isValid insteadof AclControllerTrait;
        AclControllerTrait::isValid as isValidAcl;
    }
```

(suite sur la page suivante)

(suite de la page précédente)

```
    public function isValid($action){
        return parent::isValid($action)&& $this->isValidAcl($action);
    }
}
```

Allow avec Rôle, ressource et permission

Allow sans création préalable :

`@USER` est autorisé à accéder à la ressource `Foo` avec la permission `READ`.

Code source 4 – app/controllers/BaseAclController.php

```
use Ubiquity\attributes\items\acl\Allow;

class BaseAclController extends Controller {
use AclControllerTrait;
    ...

    #[Allow('@USER','Foo', 'READ')]
    public function foo(){
        echo 'foo page allowed for @USER and @ME';
    }
}
```

Note : Le rôle, la ressource et la permission sont automatiquement créés dès qu'ils sont invoqués avec `Allow`.

Allow avec création explicite :

Code source 5 – app/controllers/BaseAclController.php

```
use Ubiquity\attributes\items\acl\Allow;
use Ubiquity\attributes\items\acl\Permission;

class BaseAclController extends Controller {
use AclControllerTrait;
   ...

   #[Permission('READ',500)]
   #[Allow('@USER','Foo', 'READ')]
   public function foo(){
      echo 'foo page allowed for @USER and @ME';
   }
}
```

Ajout d'ACL à l'exécution

Que ce soit dans un contrôleur ou dans un service, il est possible d'ajouter des rôles, des ressources, des permissions et des autorisations au moment de l'exécution :

Par exemple :\N- Ajouter un rôle `@USER` héritant de `@GUEST`.

```
use Ubiquity\security\acl\AclManager;

AclManager::addRole('@GUEST');
AclManager::addRole('@USER',['@GUEST']);
```

Définir les ACLs avec une base de données

Les ACL définies dans la base de données s'ajoutent aux ACL définies via les annotations ou les attributs.

32.3.2 Initialisation

L'initialisation permet de créer les tables associées aux ACLs (Role, Resource, Permission, AclElement). Elle ne doit être faite qu'une seule fois, et en mode dev uniquement.

A placer par exemple dans le fichier `app/config/bootstrap.php` :

```
use Ubiquity\controllers\Startup;
use Ubiquity\security\acl\AclManager;

$config=Startup::$config;
AclManager::initializeDAOProvider($config, 'default');
```

32.3.3 Démarrage

Dans le fichier `app/config/services.php` :

```
use Ubiquity\security\acl\AclManager;
use Ubiquity\security\acl\persistence\AclCacheProvider;
use Ubiquity\security\acl\persistence\AclDAOProvider;
use Ubiquity\orm\DAO;

DAO::start();//Optional, to use only if dbOffset is not default

AclManager::start();
AclManager::initFromProviders([
    new AclCacheProvider(), new AclDAOProvider($config)
]);
```

32.4 Stratégies de définition des ACL

32.4.1 Avec peu de ressources :

Définir les autorisations pour chaque action du contrôleur ou chaque groupe d'actions :

Les ressources correspondent logiquement aux contrôleurs, et les permissions aux actions. Mais cette règle peut ne pas être respectée, et une action peut être définie comme une ressource, selon les besoins.

La seule règle obligatoire est qu'une paire contrôleur/action ne peut correspondre qu'à une seule paire ressource/permission (pas nécessairement unique).

Code source 6 – app/controllers/BaseAclController.php

```
namespace controllers;

use Ubiquity\controllers\Controller;
use Ubiquity\security\acl\controllers\AclControllerTrait;
use Ubiquity\attributes\items\acl\Permission;
use Ubiquity\attributes\items\acl\Resource;

#[Resource('Foo')]
#[Allow('@ADMIN')]
class FooController extends Controller {
   use AclControllerTrait;

   #[Allow('@NONE')]
   public function index() {
      echo 'index';
   }

   #[Allow('@USER')]
   public function read() {
      echo 'read';
   }
```

(suite sur la page suivante)

(suite de la page précédente)

```
    #[Allow('@USER')]
    public function write() {
        echo 'write';
    }

    public function admin() {
        echo 'admin';
    }

    public function _getRole() {
        return $_GET['role']??'@NONE';
    }

    /**
     * {@inheritdoc}
     * @see \Ubiquity\controllers\Controller::onInvalidControl()
     */
    public function onInvalidControl() {
        echo $this->_getRole() . ' is not allowed!';
    }

}
```

32.4.2 Avec plus de ressources :

Code source 7 – app/controllers/BaseAclController.php

```
namespace controllers;

use Ubiquity\controllers\Controller;
use Ubiquity\security\acl\controllers\AclControllerTrait;
use Ubiquity\attributes\items\acl\Permission;
use Ubiquity\attributes\items\acl\Resource;

#[Resource('Foo')]
class FooController extends Controller {
    use AclControllerTrait;

    #[Permission('INDEX',1)]
    public function index() {
        echo 'index';
    }

    #[Permission('READ',2)]
    public function read() {
        echo 'read';
    }

    #[Permission('WRITE',3)]
    public function write() {
```

(suite sur la page suivante)

(suite de la page précédente)

```
        echo 'write';
    }

    #[Permission('ADMIN',10)]
    public function admin() {
        echo 'admin';
    }

    public function _getRole() {
        return $_GET['role']??'NONE';
    }

    /**
     * {@inheritdoc}
     * @see \Ubiquity\controllers\Controller::onInvalidControl()
     */
    public function onInvalidControl() {
        echo $this->_getRole() . ' is not allowed!';
    }

}
```

CHAPITRE 33

Rest

Le module REST implémente un CRUD de base, avec un système d'authentification, directement testable dans la partie administration.

33.1 REST et routage

Le routeur est essentiel au module REST, puisque REST (Representation State Transfer) est basé sur les URL et les méthodes HTTP.

Note : Pour des raisons de performance, les routes REST sont mises en cache indépendamment des autres routes. Il est donc nécessaire de démarrer le routeur d'une manière particulière pour activer les routes REST et ne pas obtenir une erreur 404 récurrente.

Le routeur est démarré dans le fichier `services.php`.

Sans l'activation des routes REST :

Code source 1 – app/config/services.php

```
...
Router::start();
```

Pour activer les routes REST dans une application qui a également une partie non-REST :

Code source 2 – app/config/services.php

```
...
Router::startAll();
```

Pour activer uniquement les routes REST :

```
Router::startRest();
```

Il est possible de lancer le routage de manière conditionnelle (cette méthode ne sera efficace que si le nombre de routes est important dans l'une ou l'autre partie) :

Code source 3 – app/config/services.php

```
...
    if($config['isRest']()){
            Router::startRest();
    }else{
            Router::start();
    }
```

33.2 Ressource REST

Un contrôleur REST peut être directement associé à un modèle.

Note : Si vous n'avez pas de base de données Mysql sous la main, vous pouvez télécharger celle-ci : `messagerie.sql`

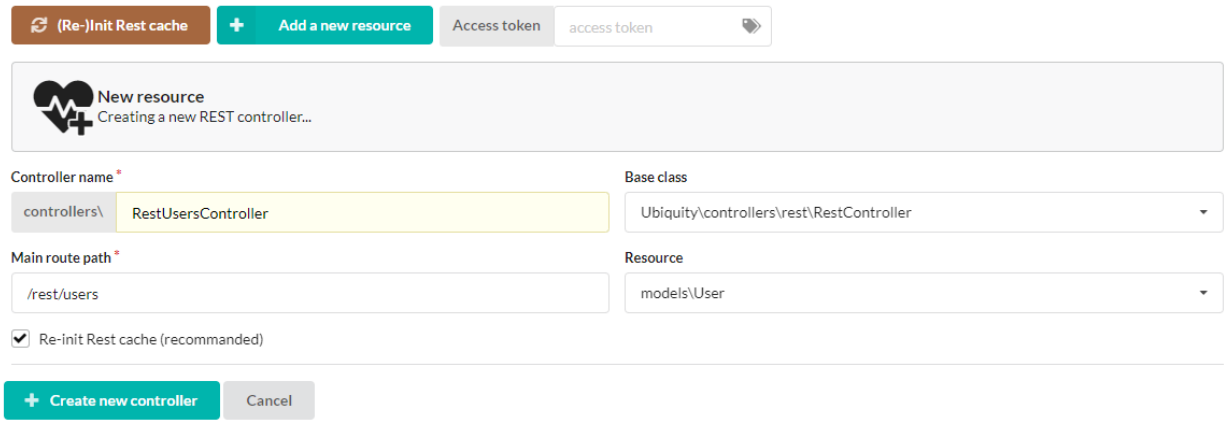
33.2.1 Création

Avec les devtools :

```
Ubiquity rest RestUsersController -r=User -p=/rest/users
```

Ou avec les webtools :

Allez dans la partie **REST** puis choisir **Add a new resource** :



Le contrôleur créé :

Code source 4 – app/controllers/RestUsersController.php

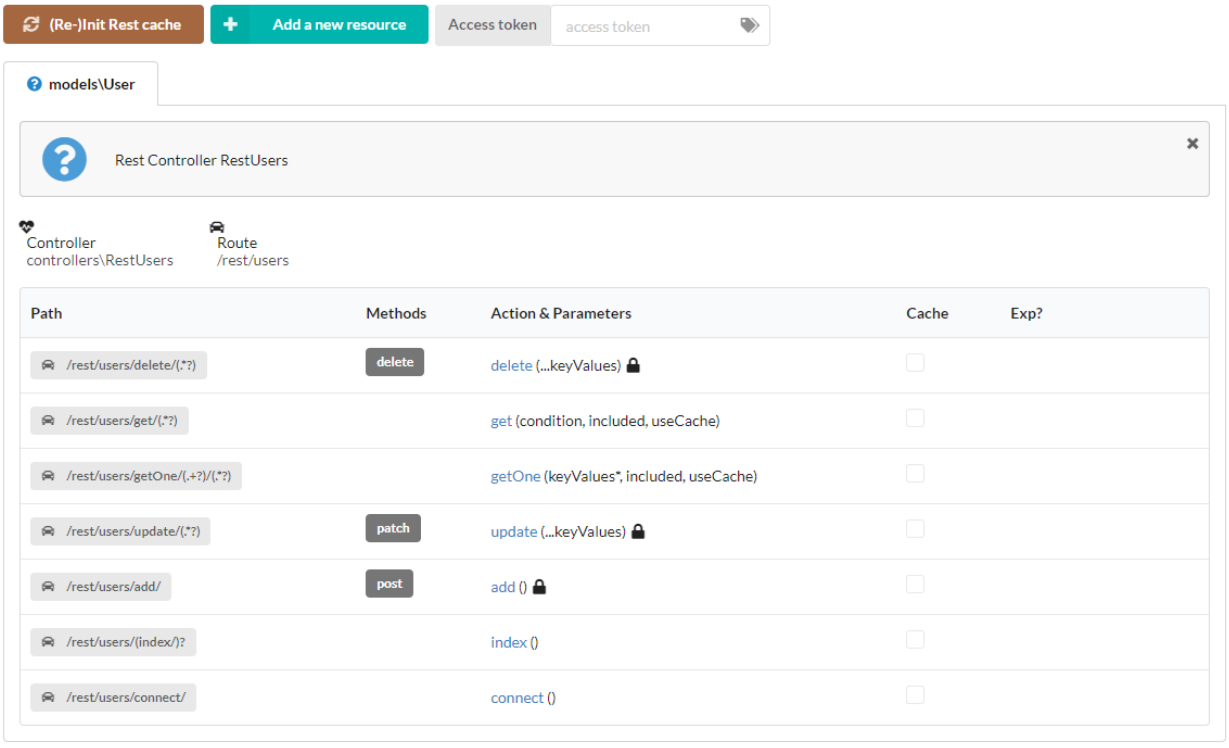
```php
namespace controllers;

/**
 * Rest Controller RestUsersController
 * @route("/rest/users","inherited"=>true,"automated"=>true)
 * @rest("resource"=>"models\\User")
 */
class RestUsersController extends \Ubiquity\controllers\rest\RestController {

}
```

Comme les attributs **automated** et **inherited** de la route sont définis sur true, le contrôleur a les routes par défaut de la classe parente.

The base controller RestController is not standardized, it should be considered as an example for data interrogation.

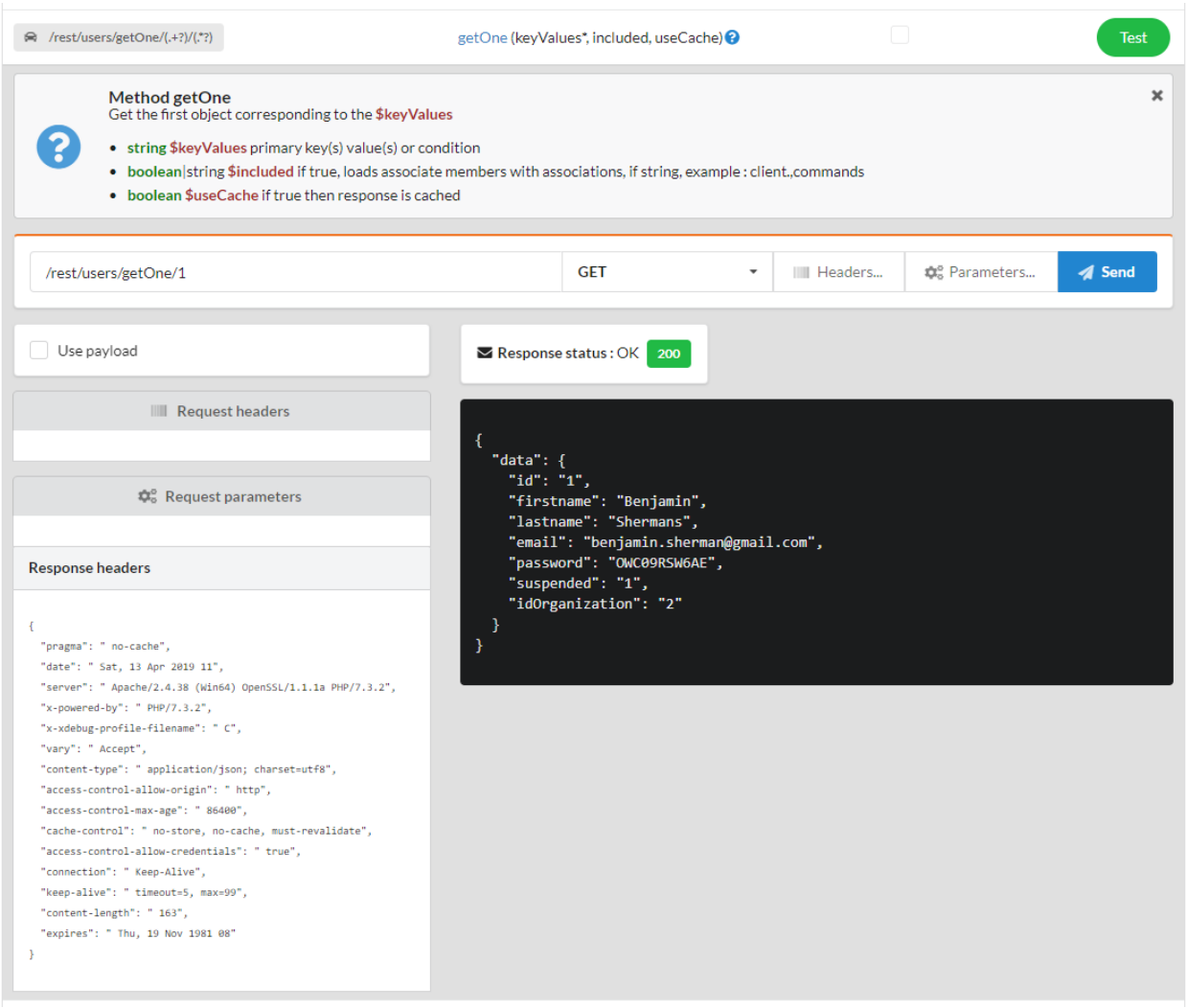
33.2.2 Interface de test

Les webtools fournissent une interface pour l'interrogation des données :

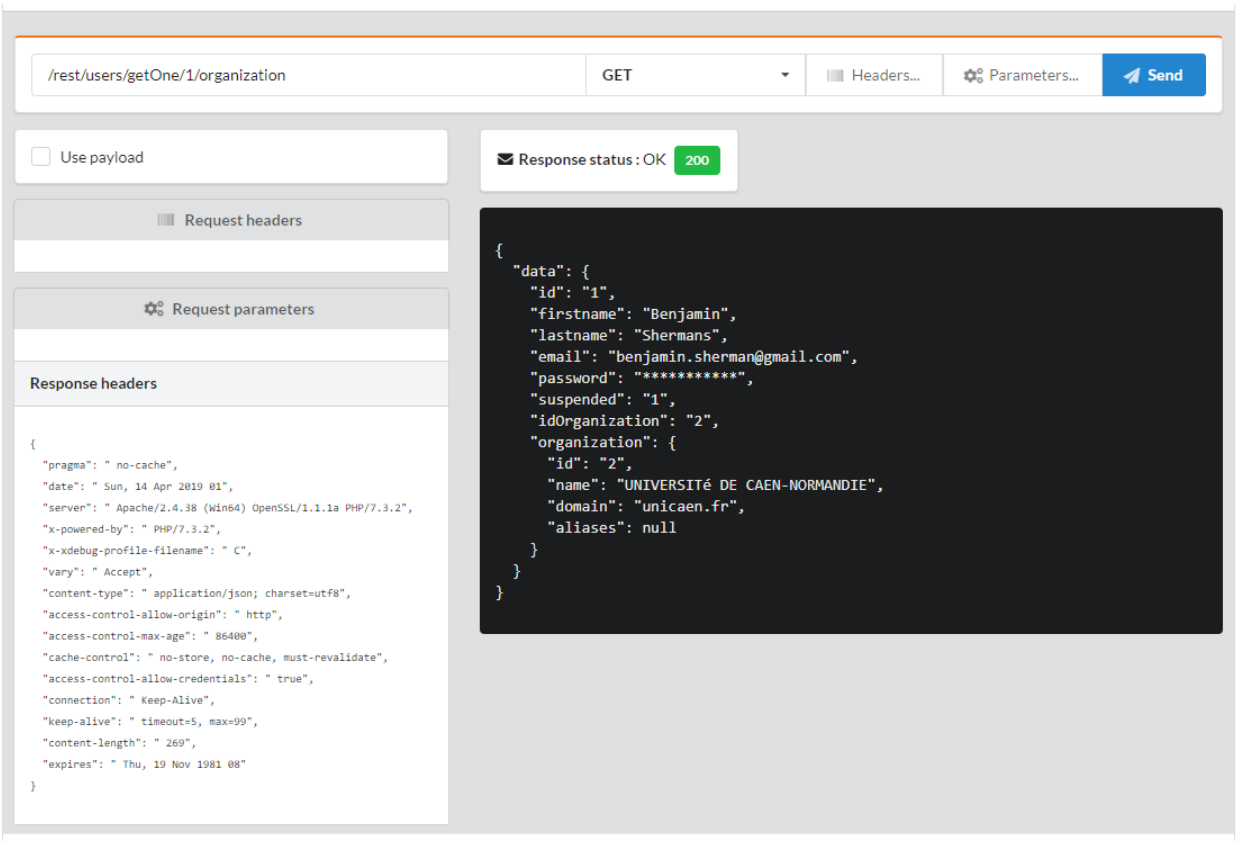


Obtenir une instance

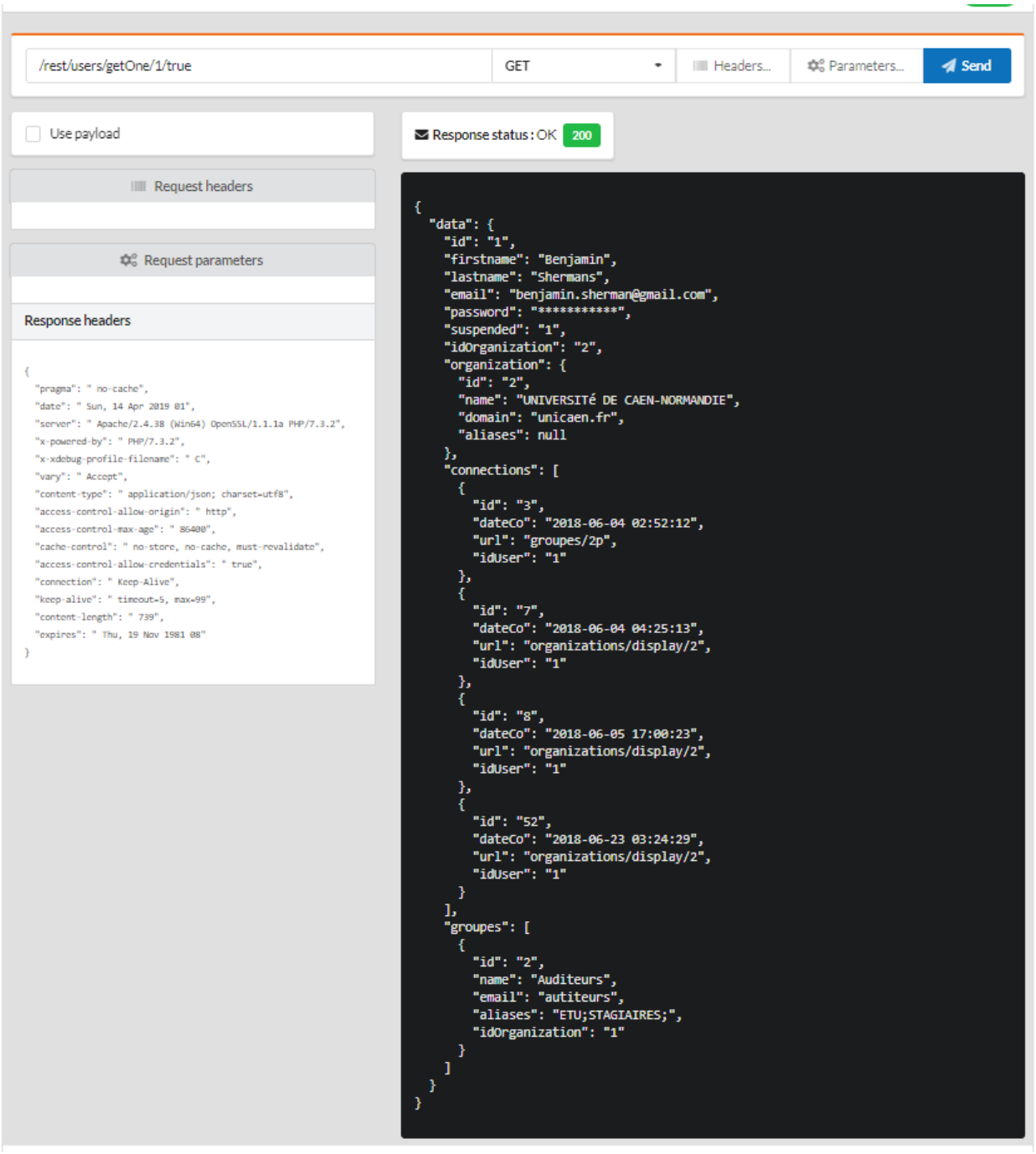
Une instance d'utilisateur est accessible par sa clé primaire (**id**) :



Inclusion des membres associés : l'organisation de l'utilisateur

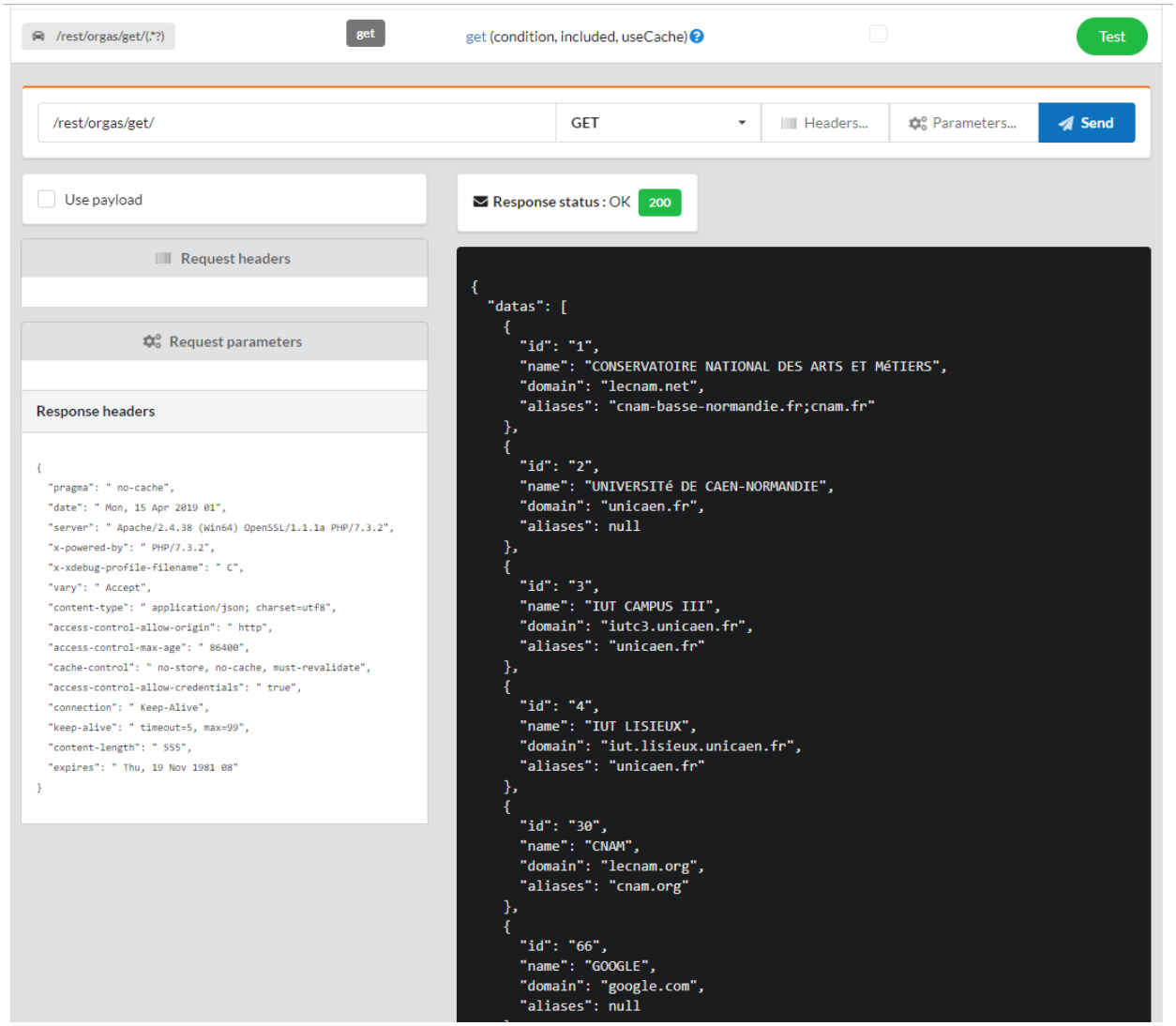


Inclusion des membres associés : organisation, connexions et groupes de l'utilisateur.

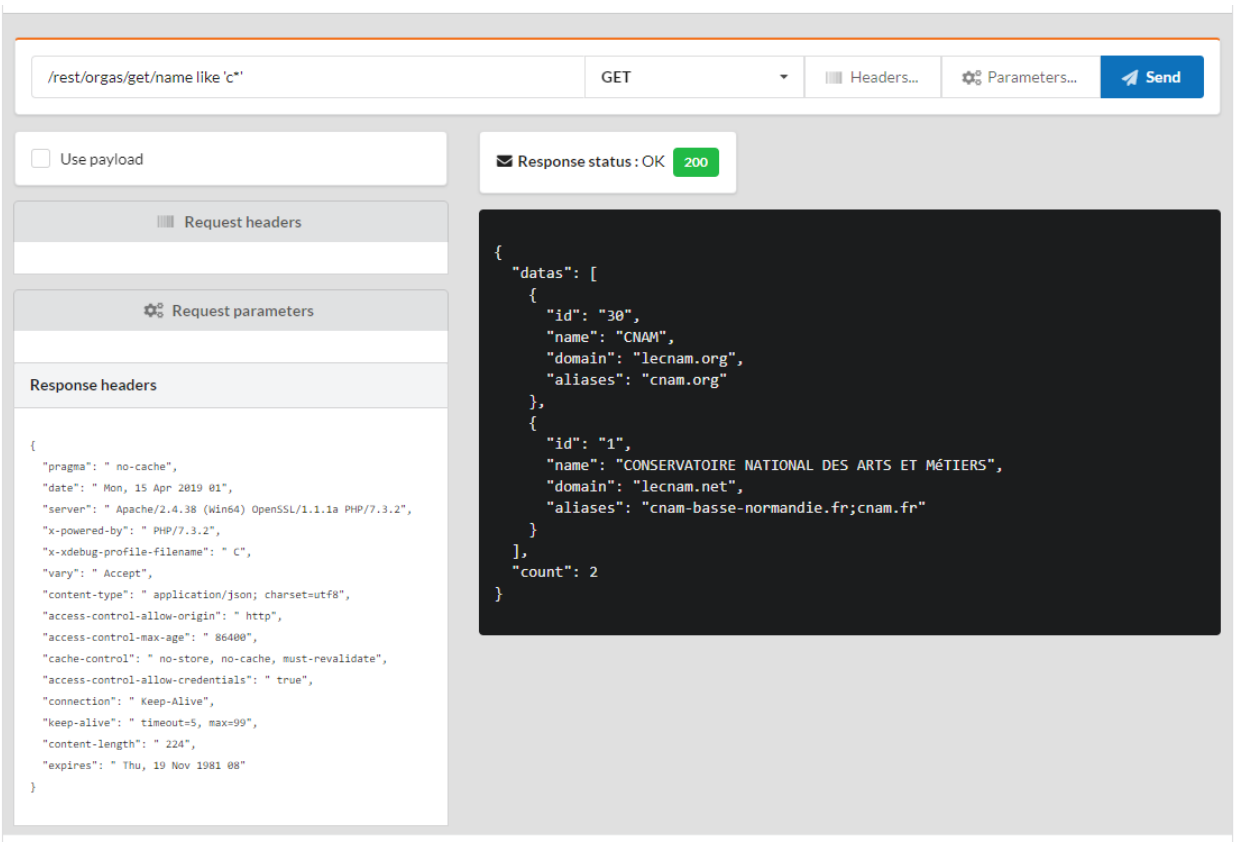

/rest/users/getOne/1/true
GET
Headers...
Parameters...
Send
Use payload
Response status : OK
200
Request headers
Request parameters
Response headers
{
"pragma": " no-cache",
"date": " Sun, 14 Apr 2019 01",
"server": " Apache/2.4.38 (Win64) OpenSSL/1.1.1a PHP/7.3.2",
"x-powered-by": " PHP/7.3.2",
"x-xdebug-profile-filename": " C",
"vary": " Accept",
"content-type": " application/json; charset=utf8",
"access-control-allow-origin": " http",
"access-control-max-age": " 86400",
"cache-control": " no-store, no-cache, must-revalidate",
"access-control-allow-credentials": " true",
"connection": " Keep-Alive",
"keep-alive": " timeout=5, max=99",
"content-length": " 739",
"expires": " Thu, 19 Nov 1981 08"
}
{
"data": {
"id": "1",
"firstname": "Benjamin",
"lastname": "Shermans",
"email": "benjamin.sherman@gmail.com",
"password": "**********",
"suspended": "1",
"idOrganization": "2",
"organization": {
"id": "2",
"name": "UNIVERSITé DE CAEN-NORMANDIE",
"domain": "unicaen.fr",
"aliases": null
},
"connections": [
{
"id": "3",
"dateCo": "2018-06-04 02:52:12",
"url": "groupes/2p",
"idUser": "1"
},
{
"id": "7",
"dateCo": "2018-06-04 04:25:13",
"url": "organizations/display/2",
"idUser": "1"
},
{
"id": "8",
"dateCo": "2018-06-05 17:00:23",
"url": "organizations/display/2",
"idUser": "1"
},
{
"id": "52",
"dateCo": "2018-06-23 03:24:29",
"url": "organizations/display/2",
"idUser": "1"
}
],
"groupes": [
{
"id": "2",
"name": "Auditeurs",
"email": "autiteurs",
"aliases": "ETU;STAGIAIRES;",
"idOrganization": "1"
}
]
}
}

Obtenir plusieurs instances

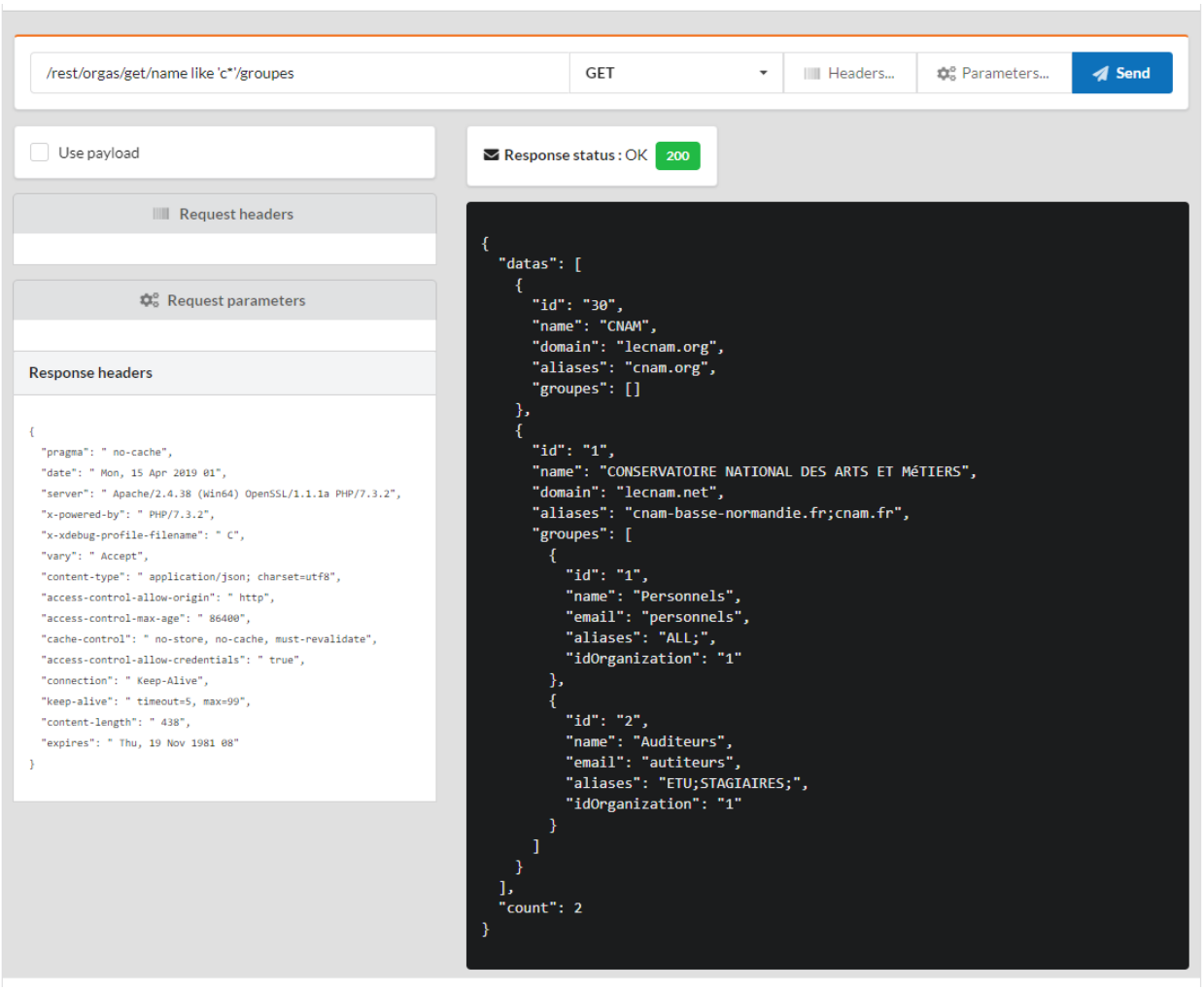
Récupérer toutes les instances :



Définir une condition :



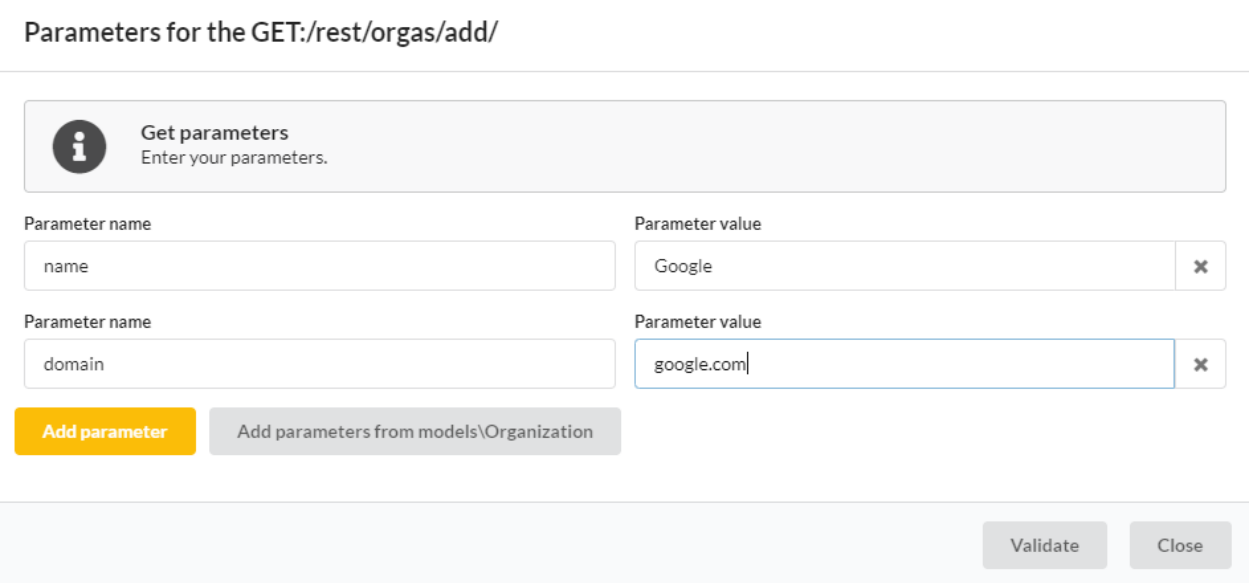
Inclure les membres associés :



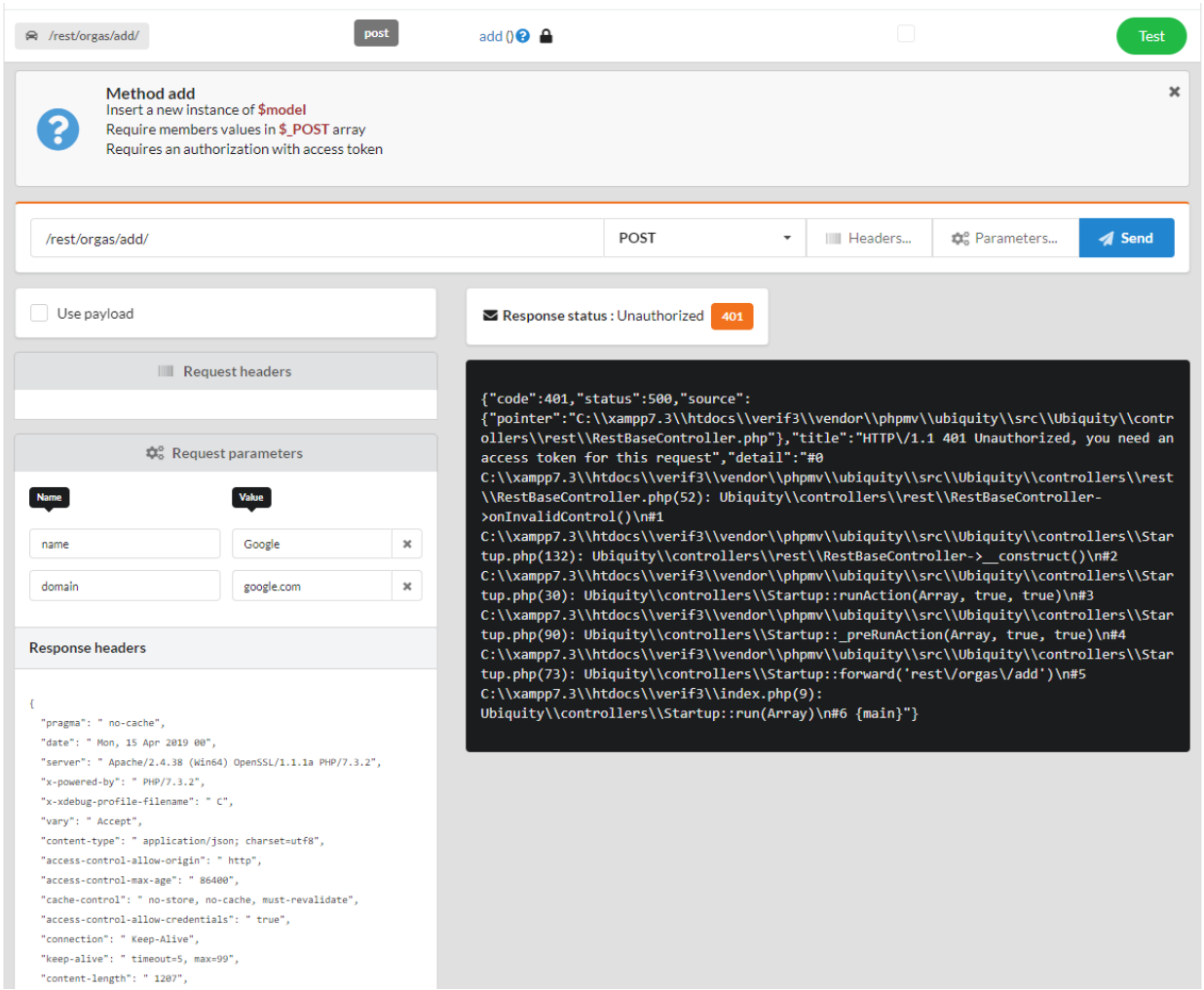
Ajout d'une instance

Les données sont envoyées par la méthode **POST**, avec un content-type défini à `application/x-www-form-urlencoded` :

Ajoutez les paramètres du nom et du domaine en cliquant sur le bouton **parameters** :

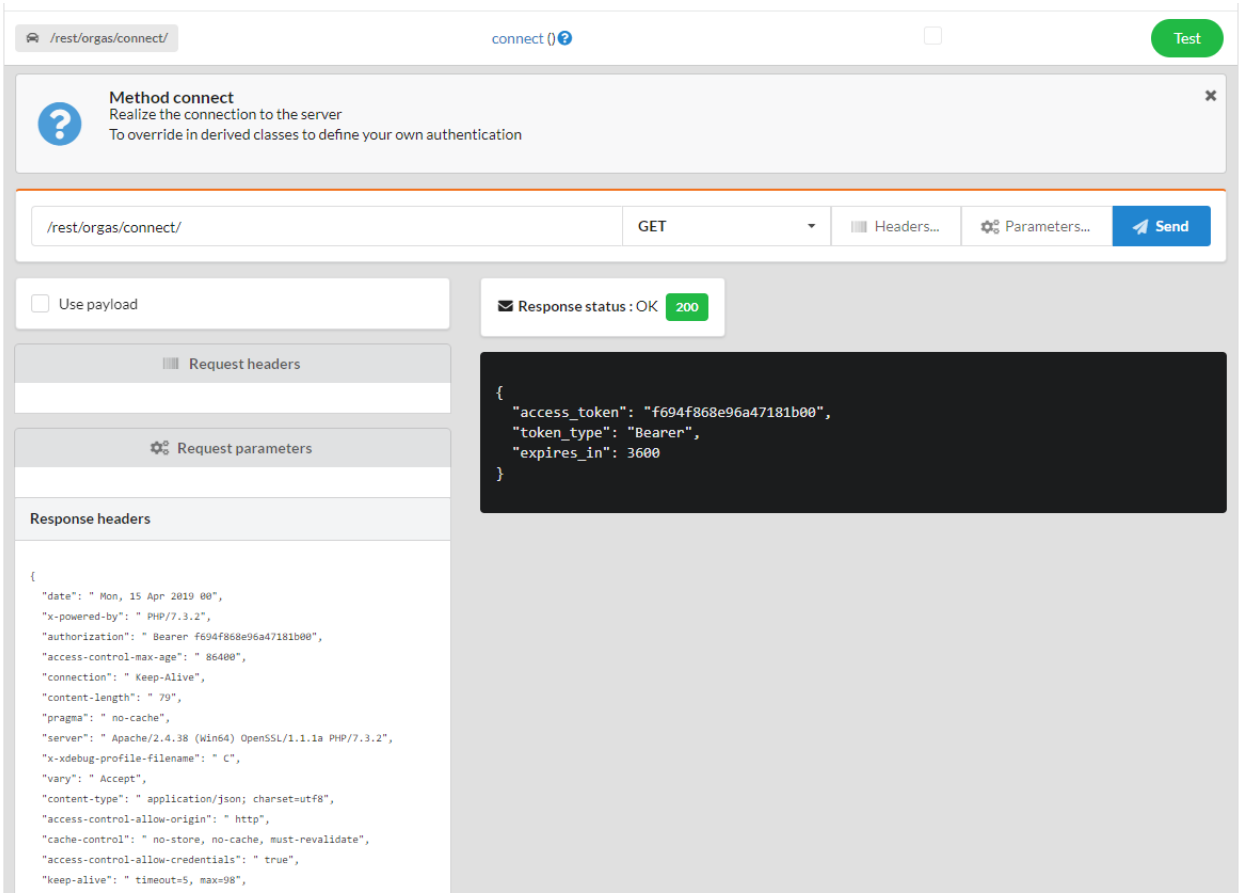


L'ajout nécessite une authentification, donc une erreur est générée, avec le statut 401 :

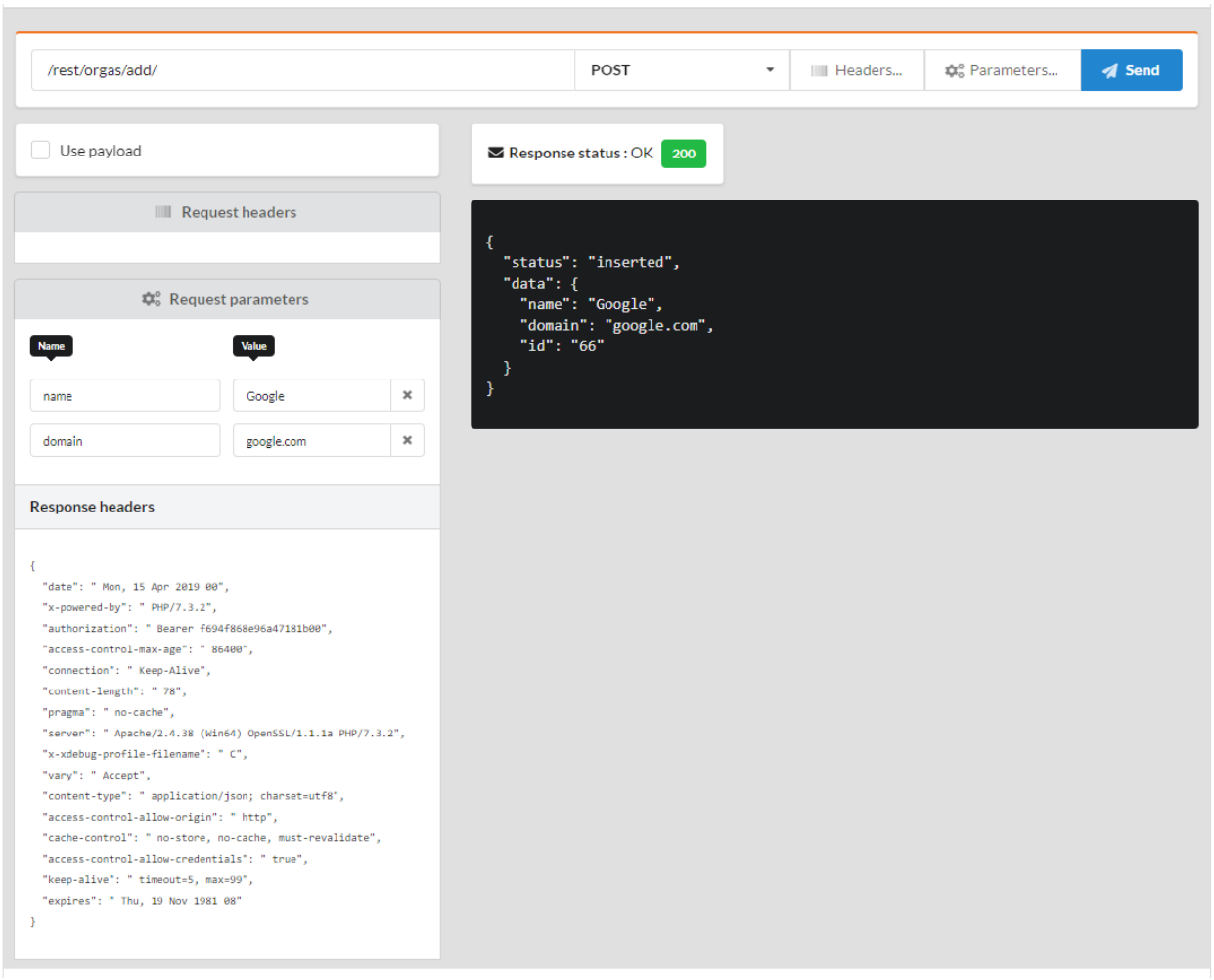


L'interface d'administration vous permet de simuler l'authentification par défaut et d'obtenir un jeton, en sollicitant la

méthode **connect** :



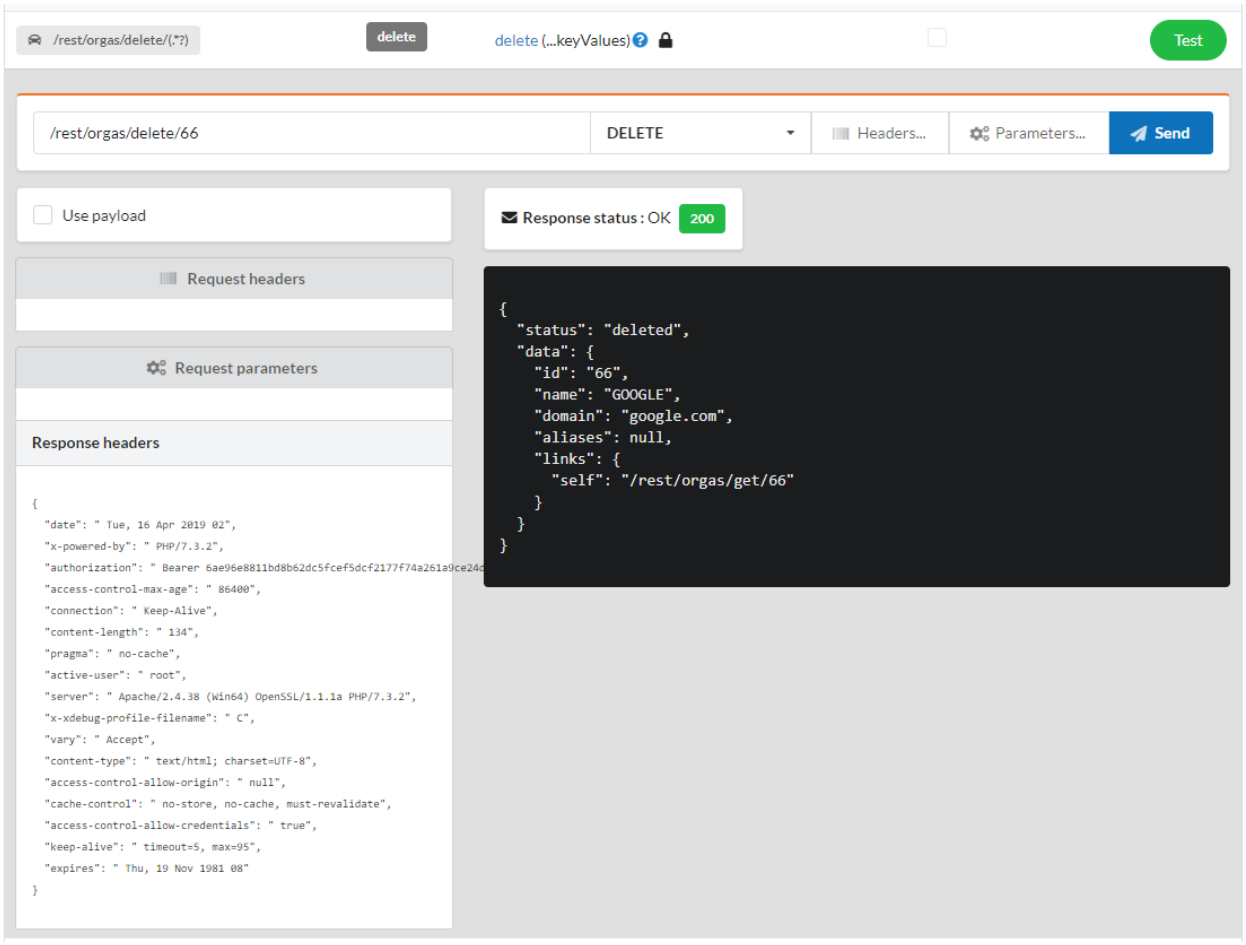
Le jeton est alors automatiquement envoyé dans les requêtes suivantes. L'enregistrement peut alors être inséré.



Mise à jour d'une instance

La mise à jour suit le même schéma que l'insertion.

Suppression d'une instance



33.2.3 Personnalisation

Routes

Il est bien sûr possible de personnaliser et de simplifier les routes. Dans ce cas, il est préférable d'utiliser l'héritage depuis la classe **RestBaseController**, et de ne pas activer les routes automatiques.

Code source 5 – app/controllers/RestOrgas.php

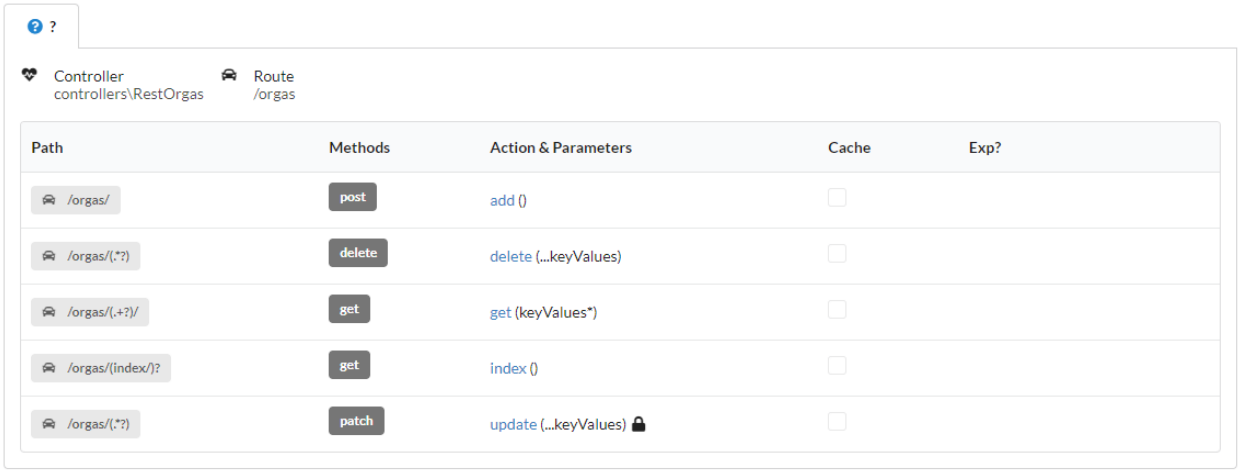
```
1  namespace controllers;
2
3  use models\Organization;
4
5  /**
6   * Rest Controller for organizations
7   *
8   * @route("/orgas")
9   * @rest
10  */
11 class RestOrgas extends \Ubiquity\controllers\rest\RestBaseController {
```

(suite sur la page suivante)

(suite de la page précédente)

```
12
13      public function initialize() {
14              $this->model = Organization::class;
15              parent::initialize();
16      }
17
18      /**
19       *
20       * @get
21       */
22      public function index() {
23              $this->_get();
24      }
25
26      /**
27       *
28       * @get("{keyValues}")
29       */
30      public function get($keyValues) {
31              $this->_getOne($keyValues);
32      }
33
34      /**
35       *
36       * @post("/")
37       */
38      public function add() {
39              $this->_add();
40      }
41
42      /**
43       *
44       * @patch("{keyValues}")
45       */
46      public function update(...$keyValues) {
47              $this->_update(...$keyValues);
48      }
49
50      /**
51       *
52       * @delete("{keyValues}")
53       */
54      public function delete(...$keyValues) {
55              $this->_delete(...$keyValues);
56      }
57  }
```

Après avoir réinitialisé le cache, l'interface de test montre les routes accessibles :



Modification des données envoyées

33.2.4 Par sur-définition

Il est possible de modifier les données envoyées aux méthodes update et add, afin d'ajouter, modifier ou supprimer la valeur des champs avant l'envoi. Soit en surdéfinissant la méthode getDatas :

Code source 6 – app/controllers/RestOrgas.php

```
...

    protected function getDatas() {
            $datas = parent::getDatas();
            unset($datas['aliases']);// Remove aliases field
            return $datas;
    }
```

33.2.5 Avec les events

Soit de manière plus globale en agissant sur les événements de repos :

Code source 7 – app/config/services.php

```
use Ubiquity\events\EventsManager;
use Ubiquity\events\RestEvents;
use Ubiquity\controllers\rest\RestBaseController;

...

EventsManager::addListener(RestEvents::BEFORE_INSERT, function ($o, array &$datas,␣
↪RestBaseController $resource) {
    unset($datas['aliases']);// Remove aliases field
});
```

33.3 Authentification

Ubiquity REST implémente une authentification Oauth2 avec des jetons Bearer. Seules les méthodes avec l'annotation `@authorization` nécessitent l'authentification, ce sont les méthodes de modification (add, update & delete).

```
            /**
             * Update an instance of $model selected by the primary key $keyValues
             * Require members values in $_POST array
             * Requires an authorization with access token
             *
             * @param array $keyValues
             * @authorization
             * @route("methods"=>["patch"])
             */
            public function update(...$keyValues) {
                    $this->_update ( ...$keyValues );
            }
```

La méthode **connect** d'un contrôleur REST établit la connexion et renvoie un nouveau jeton. Il appartient au développeur de surcharger cette méthode pour gérer une éventuelle authentification avec login et mot de passe.

```
{
  "access_token": "b641bf027617428c6eb6",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

33.3.1 Simulation d'une connexion avec login

Dans cet exemple, la connexion consiste simplement à envoyer une variable utilisateur par la méthode post. Si l'utilisateur est fourni, la méthode `connect` de l'instance `$server` retourne un jeton valide qui est stocké dans la session (la session agit ici comme une base de données).

Code source 8 – app/controllers/RestOrgas.php

```
1   namespace controllers;
2
3   use Ubiquity\utils\http\URequest;
4   use Ubiquity\utils\http\USession;
5
6   /**
7    * Rest Controller RestOrgas
8    * @route("/rest/orgas","inherited"=>true,"automated"=>true)
9    * @rest("resource"=>"models\\Organization")
10   */
11  class RestOrgas extends \Ubiquity\controllers\rest\RestController {
12
13          /**
```

(suite sur la page suivante)

(suite de la page précédente)

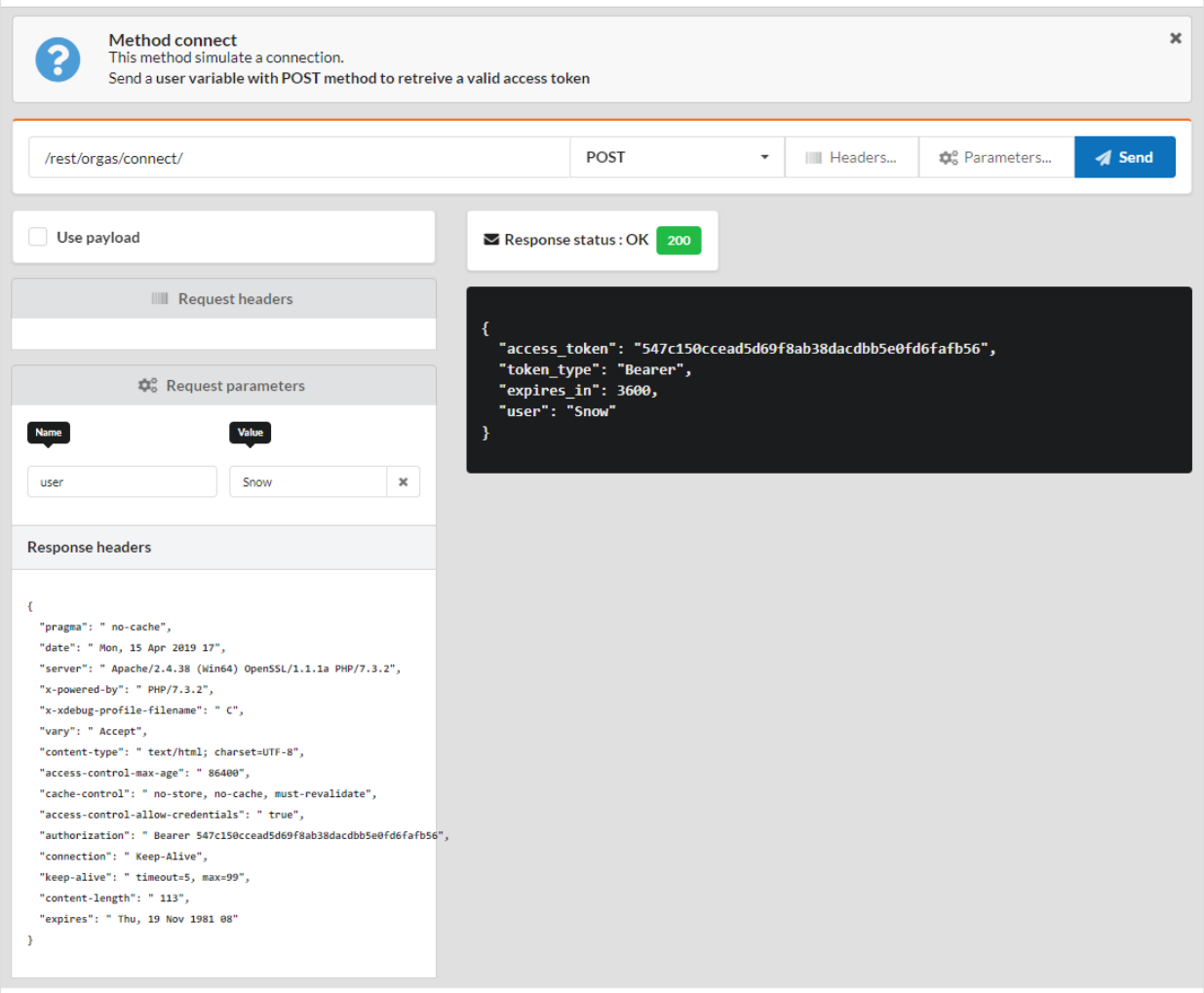
```
14              * This method simulate a connection.
15              * Send a <b>user</b> variable with <b>POST</b> method to retreive a valid␣
↪access token
16              * @route("methods"=>["post"])
17              */
18             public function connect(){
19                     if(!URequest::isCrossSite()){
20                             if(URequest::isPost()){
21                                     $user=URequest::post("user");
22                                     if(isset($user)){
23                                             $tokenInfos=$this->server->connect ();
24                                             USession::set($tokenInfos['access_token'],
↪$user);
25                                             $tokenInfos['user']=$user;
26                                             echo $this->_format($tokenInfos);
27                                             return;
28                                     }
29                             }
30                     }
31                     throw new \Exception('Unauthorized',401);
32             }
33     }
```

Pour chaque requête avec authentification, il est possible de récupérer l'utilisateur connecté (il est ajouté ici dans les en-têtes de réponse) :

Code source 9 – app/controllers/RestOrgas.php

```
1      namespace controllers;
2
3      use Ubiquity\utils\http\URequest;
4      use Ubiquity\utils\http\USession;
5
6      /**
7       * Rest Controller RestOrgas
8       * @route("/rest/orgas","inherited"=>true,"automated"=>true)
9       * @rest("resource"=>"models\\Organization")
10      */
11     class RestOrgas extends \Ubiquity\controllers\rest\RestController {
12
13             ...
14
15             public function isValid($action){
16                     $result=parent::isValid($action);
17                     if($this->requireAuth($action)){
18                             $key=$this->server->_getHeaderToken();
19                             $user=USession::get($key);
20                             $this->server->_header('active-user',$user,true);
21                     }
22                     return $result;
23             }
24     }
```

Utilisez l'interface webtools pour tester la connexion :



33.4 Personnalisation

33.4.1 Jetons d'api

Il est possible de personnaliser la génération de jetons, en surdéfinissant la méthode `getRestServer` :

Code source 10 – app/controllers/RestOrgas.php

```
1  namespace controllers;
2
3  use Ubiquity\controllers\rest\RestServer;
4  class RestOrgas extends \Ubiquity\controllers\rest\RestController {
5
6          ...
7
8          protected function getRestServer(): RestServer {
9                  $srv= new RestServer($this->config);
```

(suite sur la page suivante)

(suite de la page précédente)

```
                $srv->setTokenLength(32);
                $srv->setTokenDuration(4800);
                return $srv;
        }
}
```

33.4.2 Origines autorisées et CORS

Cross-Origin Resource Sharing (CORS)

Si vous accédez à votre api depuis un autre site, il est nécessaire de configurer **CORS**.

Dans ce cas, pour les requêtes de type `PATCH`, `PUT`, `DELETE`, votre api doit définir une route permettant à CORS d'effectuer son contrôle pré-requête en utilisant la méthode `OPTIONS`.

Code source 11 – app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        /**
         * @options('{url}')
         */
        public function options($url='') {}
}
```

Origines autorisées

Les origines autorisées permettent de définir les clients qui peuvent accéder à la ressource dans le cas d'une requête inter-domaine en définissant l'en-tête de réponse **Access-Control-Allow-Origin**. Ce champ d'en-tête est renvoyé par la méthode `OPTIONS`.

Code source 12 – app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        protected function getRestServer(): RestServer {
                $srv= new RestServer($this->config);
                $srv->setAllowedOrigin('http://mydomain/');
                return $srv;
        }
}
```

Il est possible d'autoriser plusieurs origines :

Code source 13 – app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        protected function getRestServer(): RestServer {
                $srv= new RestServer($this->config);
                $srv->setAllowedOrigins(['http://mydomain1/','http://mydomain2/']);
                return $srv;
        }
}
```

33.4.3 Réponse

Pour changer le format des réponses, il est nécessaire de créer une classe héritant de `ResponseFormatter`. Nous allons nous inspirer de **HAL**, et changer le format des réponses par :

- ajout d'un lien vers self pour chaque ressource
- ajout d'un attribut « _embedded » pour les collections
- suppression de l'attribut « data » pour les ressources uniques

Code source 14 – app/controllers/RestOrgas.php

```
namespace controllers\rest;

use Ubiquity\controllers\rest\ResponseFormatter;
use Ubiquity\orm\OrmUtils;

class MyResponseFormatter extends ResponseFormatter {

        public function cleanRestObject($o, &$classname = null) {
                $pk = OrmUtils::getFirstKeyValue ( $o );
                $r=parent::cleanRestObject($o);
                $r["links"]=["self"=>"/rest/orgas/get/".$pk];
                return $r;
        }

        public function getOne($datas) {
                return $this->format ( $this->cleanRestObject ( $datas ) );
        }

        public function get($datas, $pages = null) {
                $datas = $this->getDatas ( $datas );
                return $this->format ( [ "_embedded" => $datas,"count" => \sizeof (
↪$datas ) ] );
        }
}
```

Assignation ensuite de « MyResponseFormatter » au contrôleur REST en surchargeant la méthode « getResponseFormatter » :

Code source 15 – app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        protected function getResponseFormatter(): ResponseFormatter {
                return new MyResponseFormatter();
        }
}
```

Testez les résultats avec les méthodes getOne et get :


```
{
  "id": "1",
  "name": "CONSERVATOIRE NATIONAL DES ARTS ET MéTIERS",
  "domain": "lecnam.net",
  "aliases": "cnam-basse-normandie.fr;cnam.fr",
  "links": {
    "self": "/rest/orgas/get/1"
  }
}
```

```
{
  "_embedded": [
    {
      "id": "30",
      "name": "CNAM",
      "domain": "lecnam.org",
      "aliases": "cnam.org",
      "links": {
        "self": "/rest/orgas/get/30"
      }
    },
    {
      "id": "1",
      "name": "CONSERVATOIRE NATIONAL DES ARTS ET MéTIERS",
      "domain": "lecnam.net",
      "aliases": "cnam-basse-normandie.fr;cnam.fr",
      "links": {
        "self": "/rest/orgas/get/1"
      }
    }
  ],
  "count": 2
}
```


33.5 APIs

Contrairement aux ressources REST, les contrôleurs de type APIs sont multi-ressources.

33.5.1 SimpleRestAPI

33.5.2 JsonApi

Ubiquity implémente la spécification jsonApi avec la classe `JsonApiRestController`. JsonApi est utilisé par EmberJS et d'autres. voir https://jsonapi.org/ pour en savoir plus.

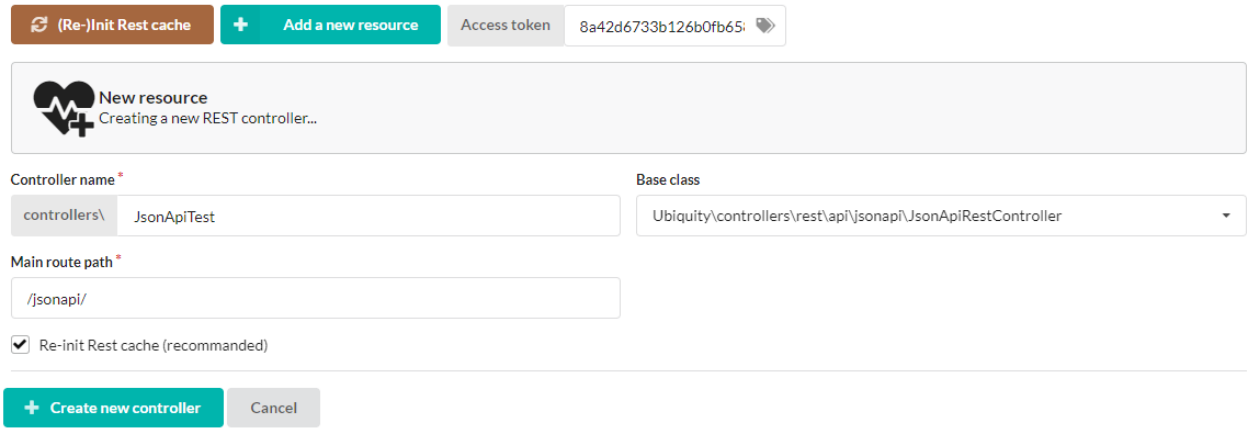
Création

Avec les devtools :

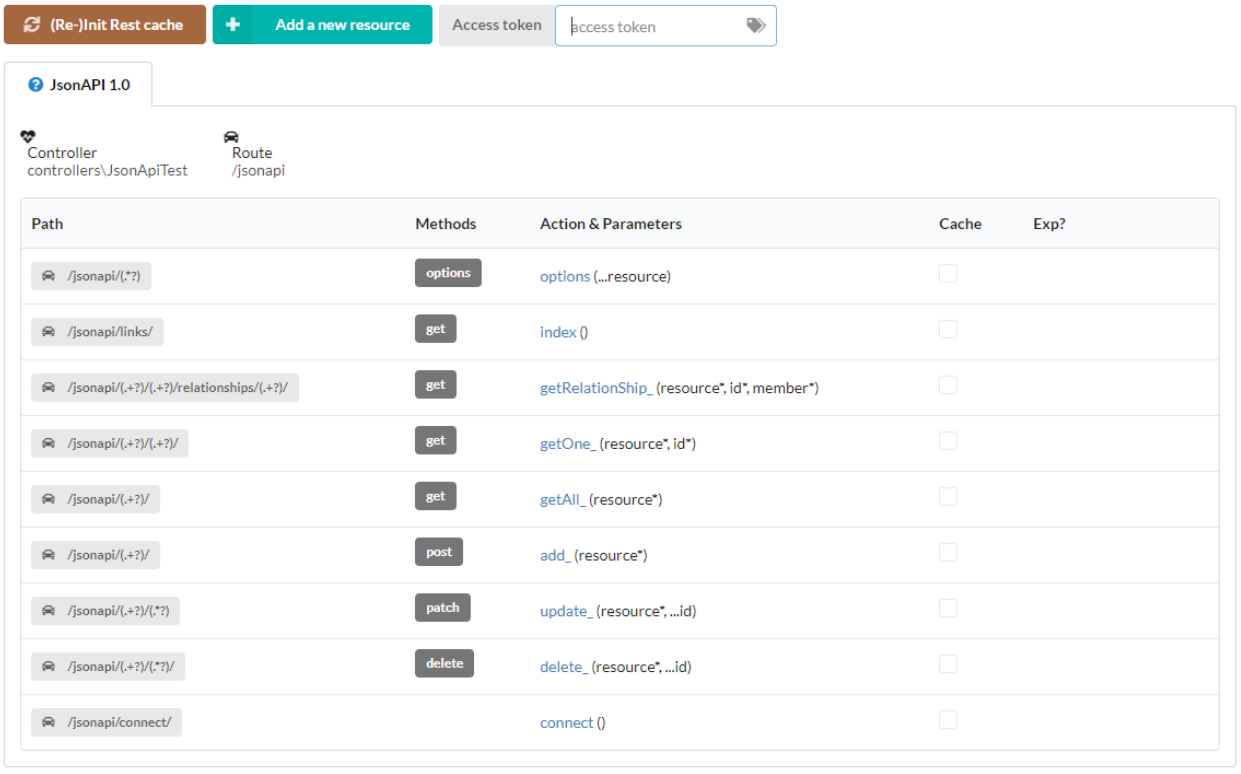
```
Ubiquity restapi JsonApiTest -p=/jsonapi
```

Ou avec les webtools :

Allez dans la partie **REST** puis choisir **Add a new resource** :

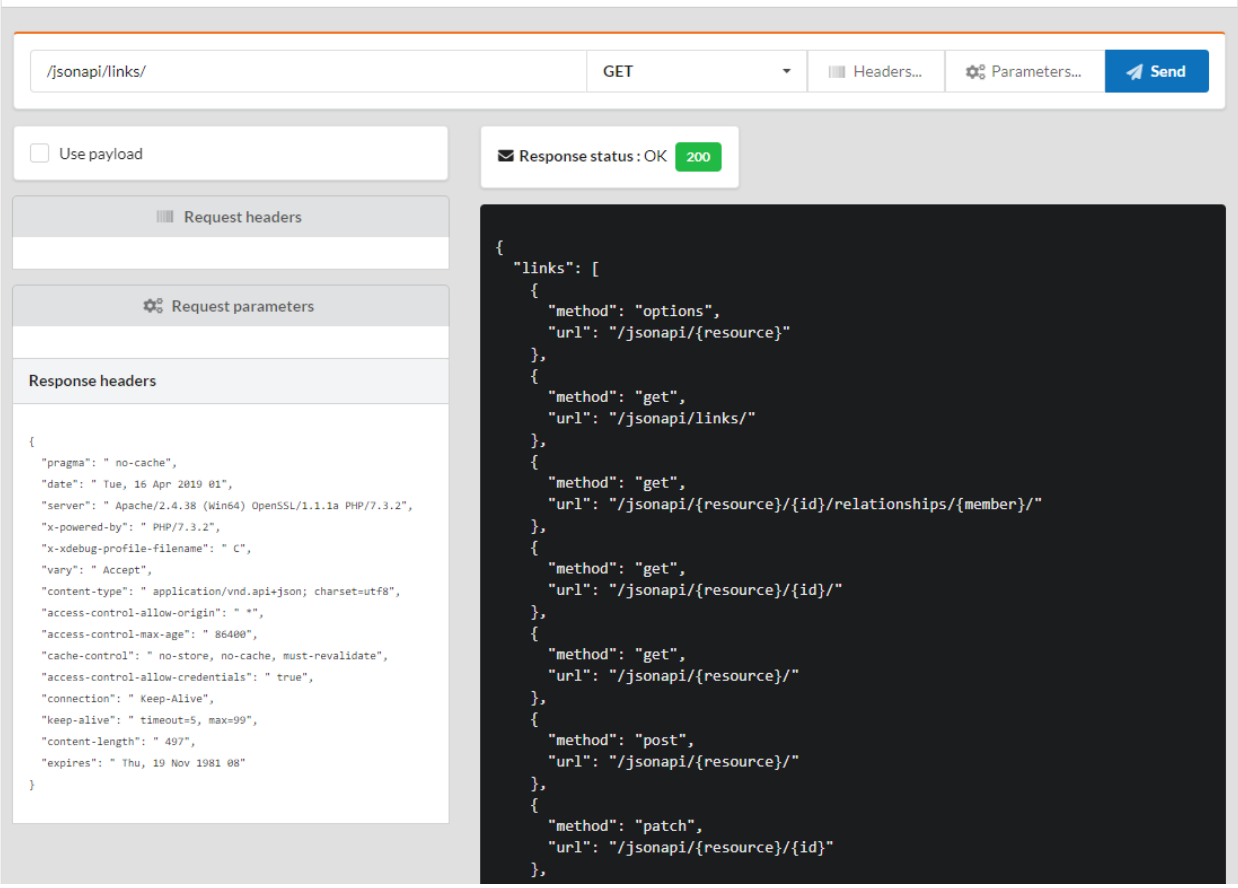


Testez l'api avec les webtools :



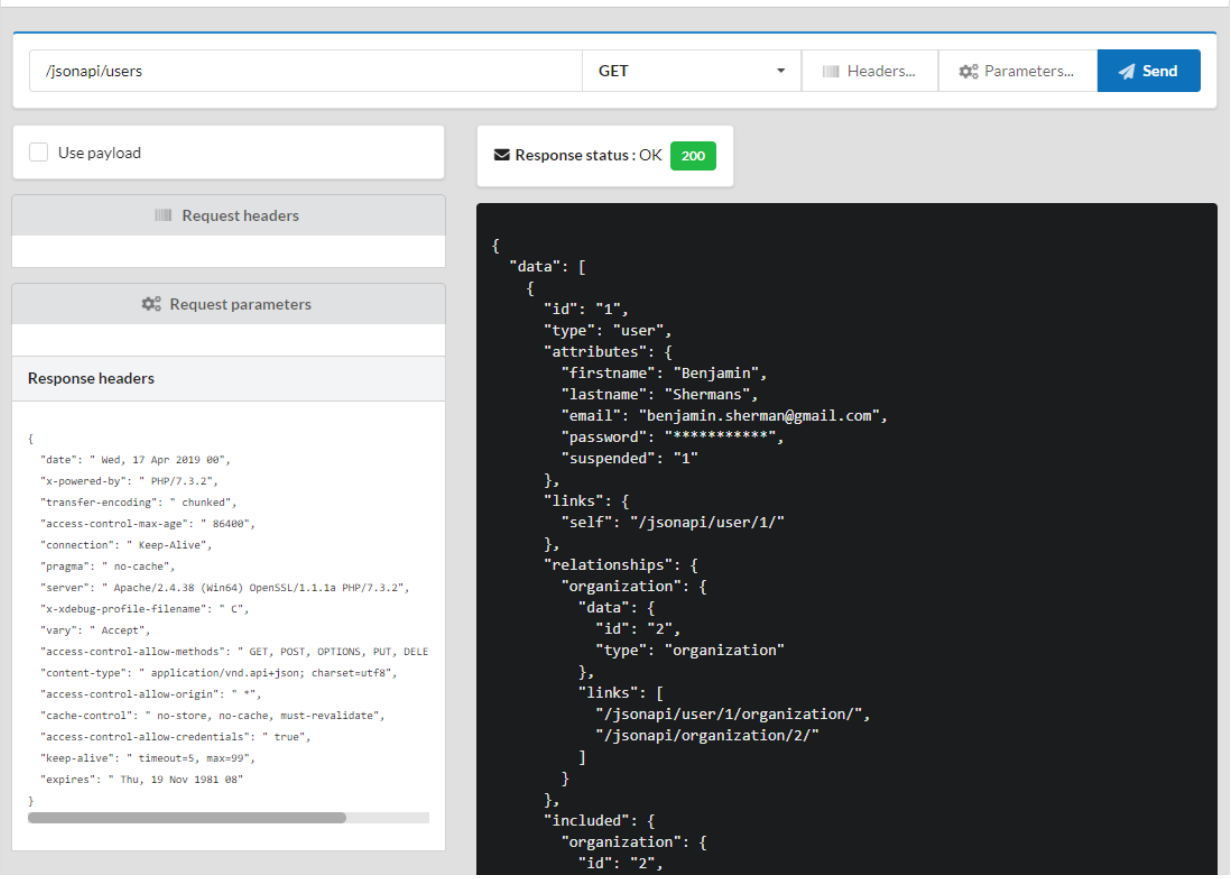
Liens

La route **links** (méthode index) renvoie la liste des urls accessibles :



Récupérer un tableau d'objets

Par défaut, tous les membres associés sont inclus :



Inclusion des membres associés

vous devez utiliser le paramètre **include** de la requête :

URL	Description
`/jsonapi/user?include=false`	Aucun membre associé n'est inclus
`/jsonapi/user?include=organization`	Inclusion de l'organisation
`/jsonapi/user?include=organization,connections`	Inclusion de l'organisation et des connexions
`/jsonapi/user?include=groupes.organization`	Inclusion des groupes, et de leur organisation

Filtrage d'instances

il est nécessaire d'utiliser le paramètre **filter** de la requête, Le paramètre **filter** correspond à la partie **where** d'une instruction SQL :

URL	Description
`/jsonapi/user?1=1`	Sans filtrage
`/jsonapi/user?firstname='Benjamin'`	Retourne tous les utilisateurs prénommés Benjamin
`/jsonapi/user?filter=firstname like 'B*'`	Retourne tous les utilisateurs dont le nom commence par un B
`/jsonapi/user?filter=suspended=0 and lastname like 'ca*'`	Retourne tous les utilisateurs suspendus dont le nom commence par « ca »

Pagination

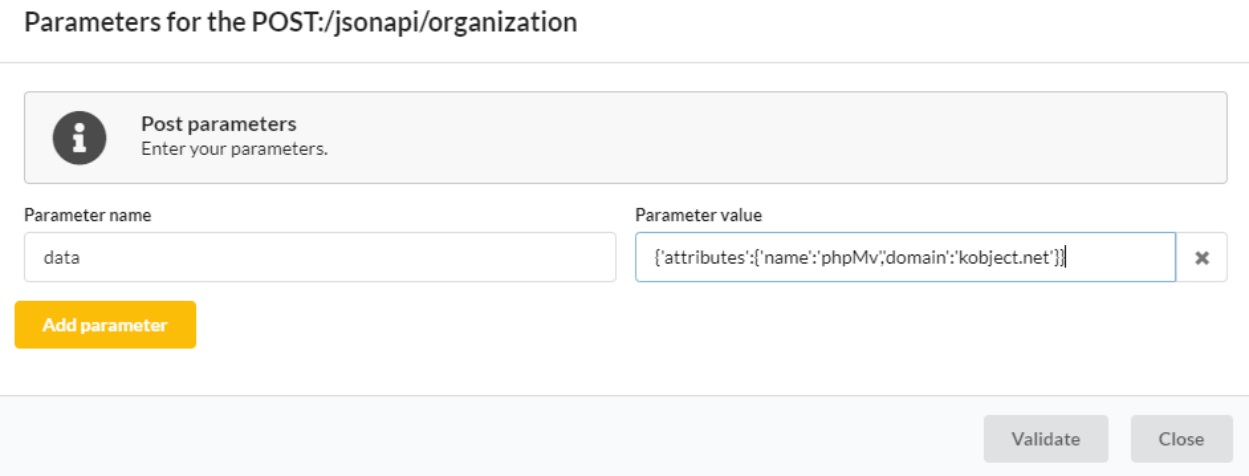
vous devez utiliser les paramètres **page[number]** et **page[size]** de la requête :

URL	Description
`/jsonapi/user`	Sans pagination
`/jsonapi/user?page[number]=1`	Affiche la première page (page size vaut 1)
`/jsonapi/user?page[number]=1&&page[size]=10`	Affiche la première page (page size vaut 10)

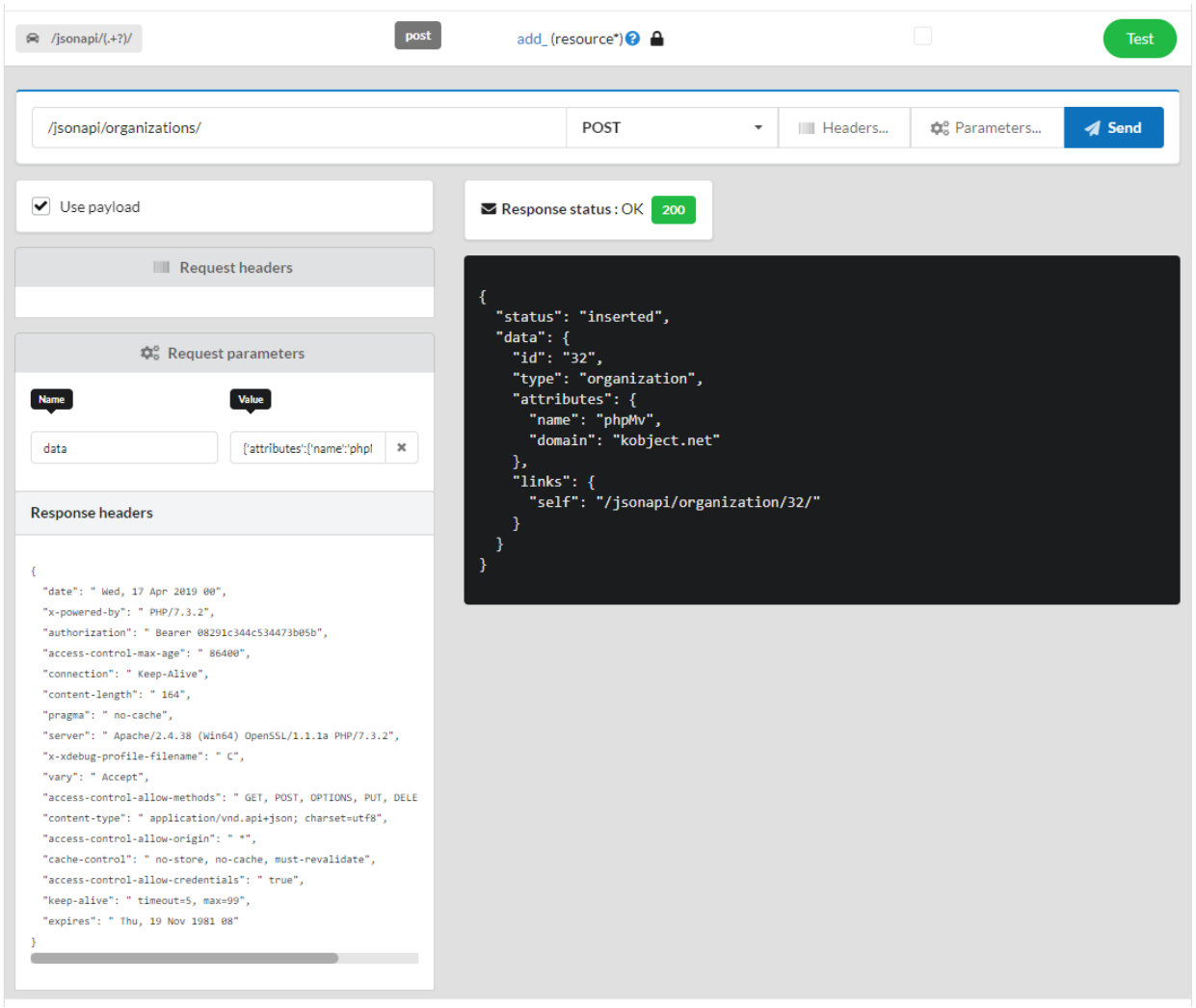
Ajout d'une instance

Les données, contenues dans `data[attributes]`, sont envoyées par la méthode **POST**, avec un type de contenu défini à `application/json ; charset=utf-8`.

Ajoutez vos paramètres en cliquant sur le bouton **paramètres** :

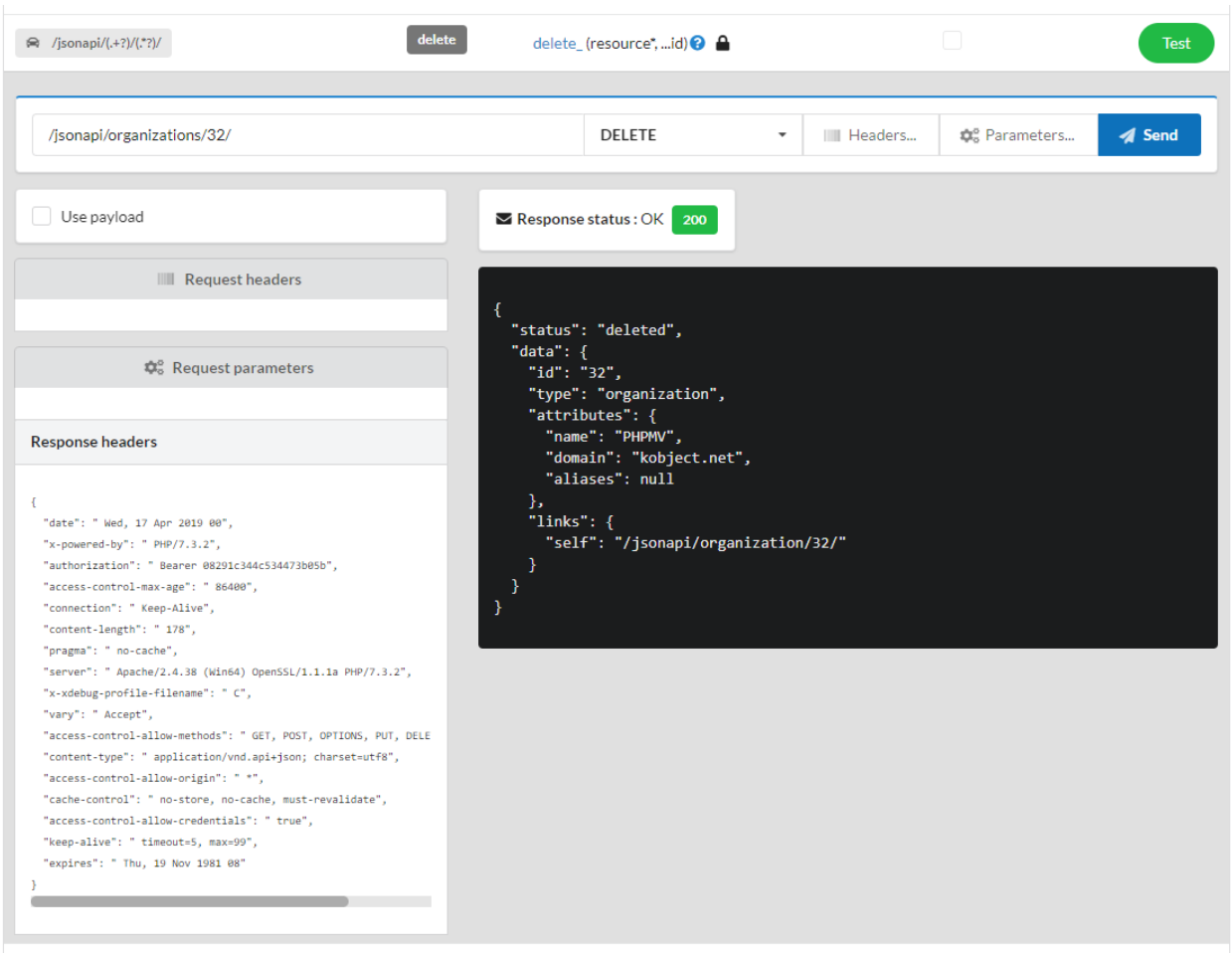


L'ajout nécessite une authentification, une erreur est donc générée, avec le statut 401 si le jeton est absent ou expiré.



Suppression d'une instance

La suppression requiert la méthode **DELETE**, et l'utilisation de l'"**id** de l'objet à supprimer :


/jsonapi/(.+?)/(.*?)/
delete
delete_(resource*, ...id)
Test
/jsonapi/organizations/32/
DELETE
Headers...
Parameters...
Send
Use payload
Request headers
Request parameters
Response headers
{
"date": " Wed, 17 Apr 2019 08",
"x-powered-by": " PHP/7.3.2",
"authorization": " Bearer 08291c344c534473b05b",
"access-control-max-age": " 86400",
"connection": " Keep-Alive",
"content-length": " 178",
"pragma": " no-cache",
"server": " Apache/2.4.38 (Win64) OpenSSL/1.1.1a PHP/7.3.2",
"x-xdebug-profile-filename": " C",
"vary": " Accept",
"access-control-allow-methods": " GET, POST, OPTIONS, PUT, DELE
"content-type": " application/vnd.api+json; charset=utf8",
"access-control-allow-origin": " *",
"cache-control": " no-store, no-cache, must-revalidate",
"access-control-allow-credentials": " true",
"keep-alive": " timeout=5, max=99",
"expires": " Thu, 19 Nov 1981 08"
}
Response status : OK
200
{
"status": "deleted",
"data": {
"id": "32",
"type": "organization",
"attributes": {
"name": "PHPMV",
"domain": "kobject.net",
"aliases": null
},
"links": {
"self": "/jsonapi/organization/32/"
}
}
}

CHAPITRE 34

Webtools

Note : Les webtools vous permettent de gérer une application Ubiquity via une interface web. Depuis **Ubiquity 2.2.0**, les webtools sont dans un repository séparé.

34.1 Installation

Mettez à jour les devtools si nécessaire pour commencer :

```
composer global update
```

34.1.1 Lors de la création d'un projet

Créer un projet avec les **webtools** (option `-a`)

```
Ubiquity new quick-start -a
```

34.1.2 Dans un projet existant

Dans une console, aller à la racine du projet et exécuter :

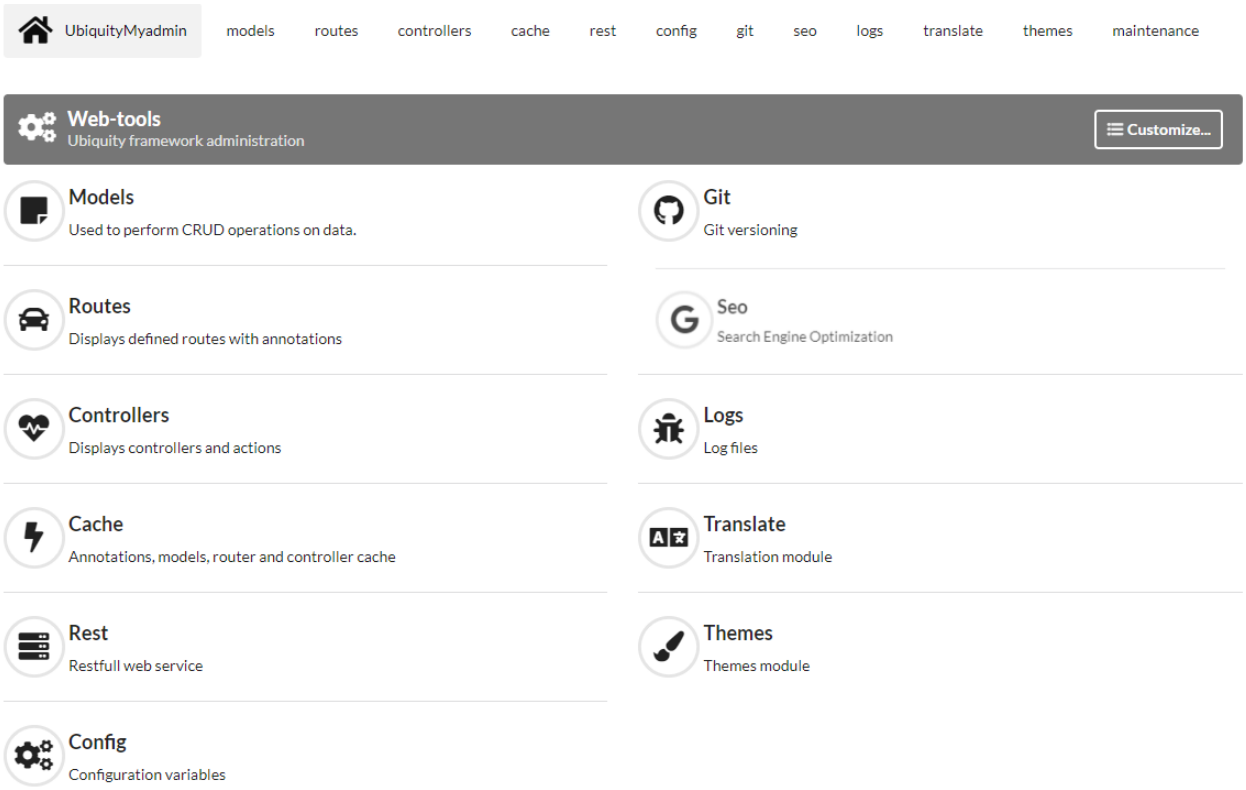
```
Ubiquity admin
```

34.2 Démarrage

Démarrez le serveur web embarqué, à partir du dossier du projet :

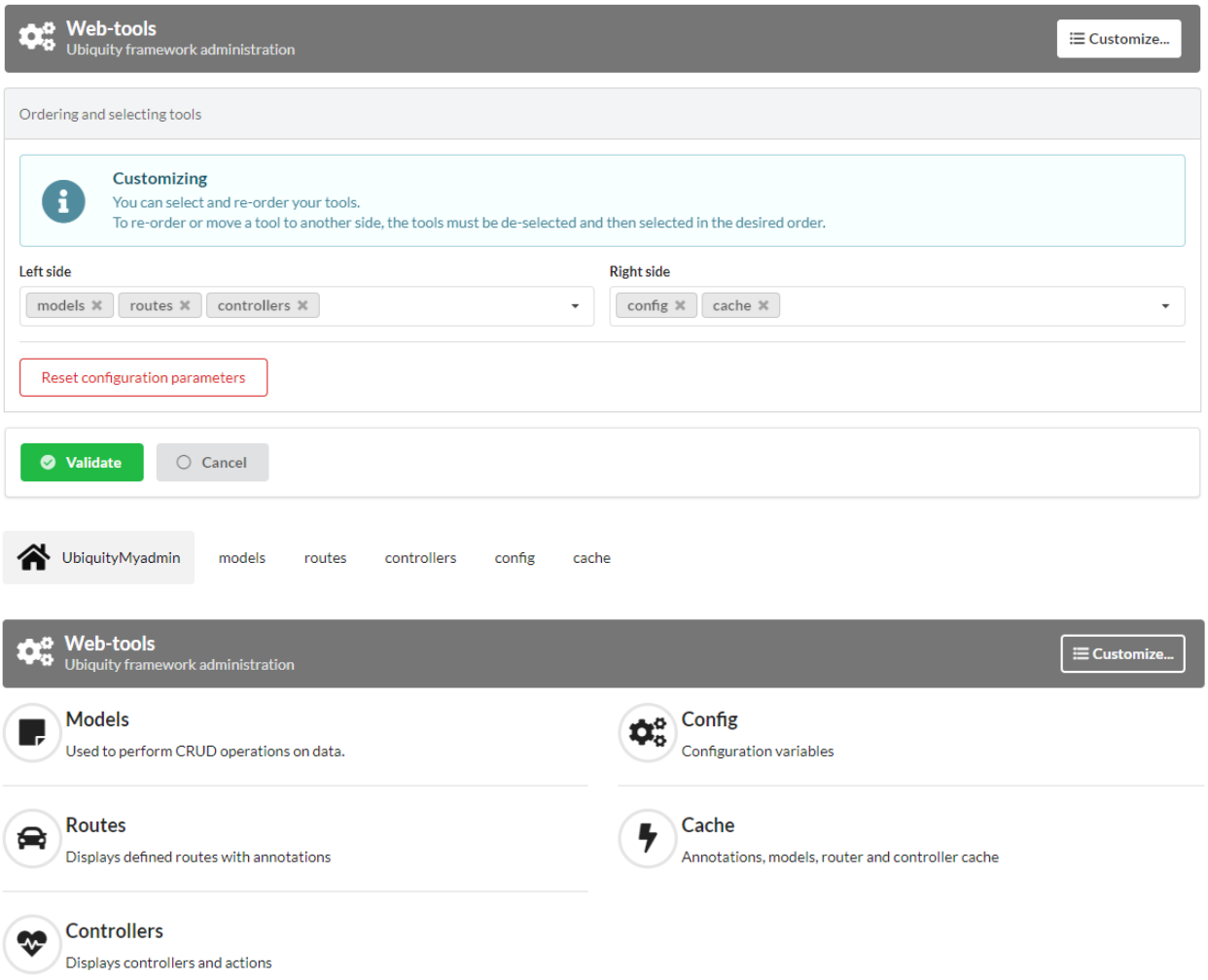
```
Ubiquity serve
```

allez à l'adresse : `http://127.0.0.1:8090/Admin`



34.3 Personnalisation

Cliquez sur **customize** pour afficher uniquement les outils que vous utilisez :



34.4 Modules webtools

34.4.1 Routes

Affiche les routes par défaut (non REST).

Operations :

— Filtrer les routes
— Tester les routes (GET, POST…)
— Initialiser le cache du routeur

34.4.2 Contrôleurs

Affiche les contrôleurs non REST.

Operations :

- Créer un contrôleur (et éventuellement la vue associée à l'action **index** par défaut)
- Créer une action dans un contrôleur (éventuellement la vue associée, la route associée)
- Créer un contrôleur spécial (CRUD ou Auth)
- Tester une action (GET, POST…)

34.4.3 Modèles

Affiche les métadonnées des modèles, permet de parcourir les entités.

Operations :

- Créer des modèles à partir d'une base de données
- Générer le cache des modèles
- Générer un script de base de données à partir de modèles existants
- Effectuer des opérations CRUD sur les modèles

34.4.4 Rest

Affiche et gère les services REST.

Operations :

- Ré-initiliser le cache Rest et les routes
- Créer un nouveau service (en utilisant une api)
- Créer une nouvelle ressource (associée à un modèle)
- Tester et interroger un service web à l'aide de méthodes http
- Effectuer des opérations CRUD sur les modèles

34.4.5 Cache

Affiche les fichiers de cache.

Operations :

- Supprimer ou réinitialiser le cache des modèles
- Supprimer ou réinitialiser le cache des contrôleurs
- Supprimer les autres fichiers de cache

34.4.6 Maintenance

Permet de gérer les modes de maintenance.

Operations :

- Créer ou mettre à jour un mode de maintenance
- Désactiver/activer un mode de maintenance
- Supprimer un mode de maintenance

34.4.7 Config

Permet d'afficher et de modifier la configuration de l'application.

34.4.8 Git

Synchronise le projet en utilisant git.

Operations :

- Configuration avec des repositories externes
- Commit
- Push
- Pull

Gère les thèmes Css.

Operations :

- Installer un thème existant
- Activer un thème
- Créer un nouveau thème (éventuellement basé sur un thème existant)

CHAPITRE 35

Contribuer

35.1 Exigences système

Avant de travailler sur Ubiquity, ajoutez à votre environnement la configuration suivante :

— Git
— PHP 7.1 ou plus.

35.2 Télécharger le code source Ubiquity

Sur le repository github Ubiquity :

— Forkez le projet Ubiquity
— Clonez votre fork localement

```
git clone git@github.com:USERNAME/ubiquity.git
```

35.3 Travaillez sur votre partie

Note : Avant de commencer, vous devez savoir que tous les correctifs que vous allez soumettre doivent être publiés sous la licence Apache 2.0, sauf si cela est explicitement spécifié dans vos commits.

35.3.1 Créez une branche dédiée

Note : Utilisez un nom descriptif pour la nommer :
— issue_xxx où xxx est le numéro de l'issue est une bonne convention pour les corrections de bugs.
— feature_name est une bonne convention pour les nouvelles fonctionnalités

```
git checkout -b NEW_BRANCH_NAME master
```

35.3.2 Travaillez sur votre partie

Travaillez sur votre code et commitez aussi souvent que nécessaire, en gardant à l'esprit les éléments suivants :
— Lire la partie *Ubiquity coding standards* ;
— Ajoutez des tests unitaires, fonctionnels ou d'acceptation pour prouver que le bug est corrigé ou que la nouvelle fonctionnalité fonctionne réellement ;
— Faites des commits atomiques et logiquement séparés (utilisez *git rebase* pour avoir un historique propre et logique) ;
— Rédigez de bons messages de commit (voir le conseil ci-dessous).
— Incrémentez les numéros de version dans tous les fichiers modifiés, en respectant les règles semver :

Étant donné un numéro de version `MAJOR.MINOR.PATCH`, incrémenter le :
— La partie MAJOR si des changements non rétro compatibles sont effectués sur l'API,
— la partie `MINOR` lorsque vous ajoutez des fonctionnalités de manière rétrocompatible, et
— la partie `PATCH` lorsque vous effectuez des corrections de bugs ou des améliorations rétro compatibles.

35.4 Soumettez vos modifications

Mettez à jour la partie [Unrelease] du fichier CHANGELOG.md en intégrant vos modifications dans les parties appropriées :
— Added
— Changed
— Removed
— Fixed
Avant de soumettre votre patch, mettez à jour votre branche (nécessaire si vous mettez du temps à terminer vos modifications) :

```
git checkout master
git fetch upstream
git merge upstream/master
git checkout NEW_BRANCH_NAME
git rebase master
```

35.5 Faire une Pull Request

Vous pouvez maintenant faire une Pull Request sur le repository github Ubiquity .

CHAPITRE 36

Guide de développement

Note : Bien que le framework soit relativement récent, veuillez noter que certaines des premières classes d'Ubiquity ne suivent pas entièrement ce guide et n'ont pas été modifiées pour des raisons de rétrocompatibilité. Néanmoins, tous les nouveaux codes doivent suivre ce guide.

36.1 Choix de conception

36.1.1 Sollicitation et utilisation de services

Injection de dépendances

Évitez d'utiliser l'injection de dépendances pour toutes les parties du framework, en interne. L'injection de dépendances est un mécanisme gourmand en ressources :

— Elle nécessite d'identifier l'élément à instancier ;
— Puis de procéder à sont instanciation :
— Pour enfin assigner le résultat à une variable.

Obtenir des services d'un conteneur

Evitez également les accès publics aux services enregistrés dans un conteneur de services. Ce type d'accès implique la manipulation d'objets dont le type de retour est inconnu, et peu facile à manipuler pour le développeur.

Par exemple, il est difficile de manipuler le retour non typé de `$this->serviceContainer->get('translator')`, comme le permettent certains frameworks, et de savoir quelles méthodes appeler dessus.

Lorsque cela est possible, et lorsque cela ne réduit pas trop la flexibilité, l'utilisation de classes statiques est conseillée :

Pour un développeur, la classe `TranslatorManager` est accessible depuis un l'ensemble du projet sans aucune instanciation d'objet. Elle expose son interface publique et permet la complétion de code :

— Le translator n'a pas besoin d'être injecté pour être utilisé ;

— Il n'a pas besoin d'être récupéré depuis un conteneur de services.

L'utilisation de classes statiques crée inévitablement une forte dépendance et affecte la flexibilité. Mais pour revenir à l'exemple du Traducteur, il n'y a aucune raison de le modifier s'il est satisfaisant. |brl Il n'est pas souhaitable de vouloir à tout prix apporter de la flexibilité quand elle n'est pas nécessaire, pour qu'en conséquence, l'utilisateur constate ensuite que son application est un peu lente.

36.2 Optimisation

L'exécution de chaque ligne de code peut avoir des répercussions importantes sur les performances. Comparez et évaluez les solutions de mise en œuvre, en particulier si le code est sollicité à plusieurs reprises :

— Identifiez ces appels répétés et coûteux avec des outils de profilage php (Blackfire profiler , Tideways . . .)
— Benchmarkez vos différentes implémentations avec phpMyBenchmarks

36.3 Qualité du code

Ubiquity utilise Scrutinizer-CI pour l'évaluation de la qualité du code.

— Pour les classes et les méthodes :
  — Les évaluations A ou B sont satisfaisantes
  — C est acceptable, mais à éviter autant que possible
  — Les notes plus basses sont à prohiber

36.3.1 Complexité du code

— Les méthodes complexes doivent être divisées en plusieurs, pour faciliter la maintenance et permettre la réutilisation ;
— Pour les classes complexes, effectuer une refactorisation de type extract-class ou extract-subclass en utilisant les traits.

36.3.2 Duplications de code

Évitez absolument la duplication du code, sauf si la duplication est minime et est justifiée par les performances.

36.3.3 Bugs

Essayez de résoudre tous les bugs signalés au fur et à mesure, sans les laisser s'accumuler.

36.4 Tests

Toute correction de bug qui n'inclut pas un test prouvant l'existence du bug corrigé, peut être suspecte. De même pour les nouvelles fonctionnalités dont on ne peut prouver qu'elles fonctionnent réellement.

Il est également important de maintenir un taux de couverture du code par les tests acceptable, qui peut baisser si une nouvelle fonctionnalité n'est pas testée.

36.5 Documentation du code

Le code actuel n'est pas encore entièrement documenté, n'hésitez pas à contribuer afin de combler cette lacune.

36.6 Standards de code

Les standards de code Ubiquity sont basés sur les standards PSR-1 , PSR-2 et PSR-4 il est donc préférable de les connaître. Les quelques exceptions aux standards sont normalement reportées dans ce guide.

36.6.1 Conventions de nommage

- Utiliser camelCase pour les variables php, les membres, les fonctions et noms de méthodes, les arguments (e.g. $modelsCacheDirectory, isStarted()) ;
- Chaque classe php doit avoir un namespace (voir PSR-4) et utiliser la convention UpperCamelCase pour son nommage (e.g. CacheManager) ;
- Préfixer toutes les classes abstraites avec Abstract, à l'exception des classes PHPUnit BaseTests :
- Suffixer les interfaces avec `Interface` ;
- Suffixer les traits avec `Trait` ;
- Suffixer les exceptions avec `Exception` ;
- Suffixer les classes principales de type manager avec `Manager` (e.g. CacheManager, TranslatorManager) ;
- Préfixer les classes utilitaires avec un `U` (e.g. UString, URequest) ;
- Utiliser UpperCamelCase pour le nommage des fichiers PHP (e.g. CacheManager.php) ;
- Utiliser les majuscules pour les constantes (e.g. const SESSION_NAME="Ubiquity").

36.6.2 Indentation, tabulations, accolades

- Utiliser des tabulations, et non des espaces ; (! PSR-2)
- Utiliser les accolades sans passage à la ligne suivante ; (! PSR-2)
- Utilisez des accolades pour indiquer le corps de la structure de contrôle, quel que soit le nombre d'instructions qu'elle contient ;

36.6.3 Classes

- Définir une classe par fichier ;
- Déclarer l'héritage de la classe et toutes les interfaces implémentées sur la même ligne que le nom de la classe ;
- Déclarer les membres de données avant les méthodes ;
- Déclarer les méthodes privées en premier, puis celles qui sont protégées pour finir par les méthodes publiques :
- Déclarer tous les arguments sur la même ligne que le nom de la méthode/fonction, quel que soit le nombre d'arguments ;
- Utiliser des parenthèses lors de l'instanciation d'une classe, même si le constructeur ne prend aucun argument ;
- Ajouter le statement use pour chaque classe qui ne fait pas partie de l'espace de noms global ;

36.6.4 Opérateurs

— Utiliser la comparaison identique ou égale lorsque vous le jugez nécessaire ;

Exemple

```php
<?php
namespace Ubiquity\namespace;

use Ubiquity\othernamespace\Foo;

/**
 * Class description.
 * Ubiquity\namespace$Example
 * This class is part of Ubiquity
 *
 * @author authorName <authorMail>
 * @version 1.0.0
 * @since Ubiquity x.x.x
 */
class Example {
        /**
         * @var int
         *
         */
        private $theInt = 1;

        /**
         * Does something from **a** and **b**
         *
         * @param int $a The a
         * @param int $b The b
         */
        function foo($a, $b) {
                switch ($a) {
                        case 0 :
                                $Other->doFoo ();
                                break;
                        default :
                                $Other->doBaz ();
                }
        }

        /**
         * Adds some values
         *
         * @param param V $v The v object
         */
        function bar($v) {
                for($i = 0; $i < 10; $i ++) {
                        $v->add ( $i );
                }
        }
}
```

Important :

Vous pouvez importer ces fichiers de standardisation intégrant ces règles dans votre IDE :

— `Eclipse`

— `PhpStorm`

Si votre IDE favori ne figure pas dans la liste, vous pouvez soumettre le fichier de normalisation associé en créant une nouvelle PR.

CHAPITRE 37

Guide de documentation

Ubiquity fourni deux espaces de documentation :

— Le guide, qui aide à apprendre les manipulations et les concepts ;

— La documentation API, qui sert de référence lors du codage.

Vous pouvez aider à améliorer les guides Ubiquity en les rendant plus cohérents, consistants ou lisibles, en ajoutant des informations manquantes, en corrigeant des erreurs factuelles, en corrigeant des fautes de frappe ou en les mettant à jour pour la dernière version d'Ubiquity.

Pour ce faire, apportez des modifications aux fichiers sources des guides Ubiquity (situés ici sur GitHub). Ensuite, ouvrez une Pull Request pour appliquer vos changements à la branche master.

Lorsque vous travaillez sur la documentation, pensez à prendre en compte les consignes.

CHAPITRE 38

Serveurs configuration

Important : Depuis la version 2.4.5, pour des raisons de sécurité et de simplification, la racine d'une application Ubiquity est localisée dans le dossier public.

38.1 Apache2

38.1.1 mod_php/PHP-CGI

Apache 2.2

Code source 1 – mydomain.conf

```
<VirtualHost *:80>
    ServerName mydomain.tld

    DocumentRoot /var/www/project/public
    DirectoryIndex /index.php

    <Directory /var/www/project/public>
        # enable the .htaccess rewrites
        AllowOverride All
        Order Allow,Deny
        Allow from All
    </Directory>

    ErrorLog /var/log/apache2/project_error.log
    CustomLog /var/log/apache2/project_access.log combined
</VirtualHost>
```

Code source 2 – mydomain.conf

```
    <VirtualHost *:80>
        ServerName mydomain.tld

        DocumentRoot /var/www/project/public
        DirectoryIndex /index.php

        <Directory /var/www/project/public>
            AllowOverride None

            # Copy .htaccess contents here

        </Directory>

        ErrorLog /var/log/apache2/project_error.log
        CustomLog /var/log/apache2/project_access.log combined
    </VirtualHost>
```

Apache 2.4

Avec Apache 2.4, `Order Allow,Deny` a été remplacé par `Require all granted`.

Code source 3 – mydomain.conf

```
    <VirtualHost *:80>
        ServerName mydomain.tld

        DocumentRoot /var/www/project/public
        DirectoryIndex /index.php

        <Directory /var/www/project/public>
            # enable the .htaccess rewrites
            AllowOverride All
            Require all granted
        </Directory>

        ErrorLog /var/log/apache2/project_error.log
        CustomLog /var/log/apache2/project_access.log combined
    </VirtualHost>
```

déplacement de index.php dans le dossier public

Si vous avez créé votre projet avec une version antérieure à la 2.4.5, vous devez modifier `index.php` et déplacer les fichiers `index.php` et `.htaccess` dans le dossier `public`.

Code source 4 – public/index.php

```
<?php
define('DS', DIRECTORY_SEPARATOR);
//Updated with index.php in public folder
```

(suite sur la page suivante)

(suite de la page précédente)

```
define('ROOT', __DIR__ . DS . '../app' . DS);
$config = include_once ROOT . 'config/config.php';
require_once ROOT . './../vendor/autoload.php';
require_once ROOT . 'config/services.php';
\Ubiquity\controllers\Startup::run($config);
```

38.1.2 PHP-FPM

Assurez-vous que les packages **libapache2-mod-fastcgi** et **php7.x-fpm** sont installés (remplacer **x** par votre version de php).

Configuration **php-pm** :

Code source 5 – php-pm.conf

```
;;;;;;;;;;;;;;;;;;;;
; Pool Definitions ;
;;;;;;;;;;;;;;;;;;;;

; Start a new pool named 'www'.
; the variable $pool can be used in any directive and will be replaced by the
; pool name ('www' here)
[www]

user = www-data
group = www-data

; use a unix domain socket
listen = /var/run/php/php7.4-fpm.sock

; or listen on a TCP socket
listen = 127.0.0.1:9000
```

Configuration **Apache 2.4** :

Code source 6 – mydomain.conf

```
<VirtualHost *:80>
...
   <FilesMatch \.php$>
        SetHandler proxy:fcgi://127.0.0.1:9000
        # for Unix sockets, Apache 2.4.10 or higher
        # SetHandler proxy:unix:/path/to/fpm.sock|fcgi://localhost/var/www/
    </FilesMatch>
 </VirtualHost>
```

38.2 nginX

Configuration **nginX** :

Code source 7 – nginx.conf

```
upstream fastcgi_backend {
    server unix:/var/run/php/php7.4-fpm.sock;
    keepalive 50;
}
server {
    server_name mydomain.tld www.mydomain.tld;
    root /var/www/project/public;
    index index.php;
    listen 8080;

 location / {
     # try to serve file directly, fallback to index.php
     try_files $uri @rewrites;
 }

 location @rewrites {
     rewrite ^/(.*)$ /index.php?c=$1 last;
 }

    location = /index.php{
        fastcgi_pass fastcgi_backend;
        fastcgi_keep_conn on;
        fastcgi_param DOCUMENT_ROOT $realpath_root;
        fastcgi_param SCRIPT_FILENAME  $document_root/index.php;
        include /etc/nginx/fastcgi_params;
    }

    # return 404 for all other php files not matching the front controller
    # this prevents access to other php files you don't want to be accessible.
    location ~ \.php$ {
        return 404;
    }

    error_log /var/log/nginx/project_error.log;
```

(suite sur la page suivante)

(suite de la page précédente)

```
    access_log /var/log/nginx/project_access.log;
}
```

38.3 Swoole

Configuration **Swoole** :

Code source 8 – .ubiquity/swoole-config.php

```
<?php
return array(
    "host" => "0.0.0.0",
    "port" => 8080,
    "options"=>[
        "worker_num" => \swoole_cpu_num() * 2,
            "reactor_num" => \swoole_cpu_num() * 2
        ]
);
```

38.4 Workerman

Configuration **Workerman** :

Code source 9 – .ubiquity/workerman-config.php

```
<?php
return array(
    "host" => "0.0.0.0",
    "port" => 8080,
    "socket"=>[
        "count" => 4,
        "reuseport" =>true
    ]
);
```

38.5 RoadRunner

Configuration **RoadRunner** :

Code source 10 – .ubiquity/.rr.yml

```
http:
  address:         ":8090"
  workers.command: "php-cgi ./.ubiquity/rr-worker.php"
  workers:
    pool:
      # Set numWorkers to 1 while debugging
```

(suite sur la page suivante)

(suite de la page précédente)

```
      numWorkers: 10
      maxJobs:    1000

# static file serving. remove this section to disable static file serving.
static:
  # root directory for static file (http would not serve .php and .htaccess files).
  dir:   "."

  # list of extensions for forbid for serving.
  forbid: [".php", ".htaccess", ".yml"]

  always: [".ico", ".html", ".css", ".js"]
```

CHAPITRE 39

Optimisation Ubiquity

Ubiquity est rapide, mais peut l'être encore plus en optimisant quelques éléments.

Note : Le serveur de test intégré (accessible par **Ubiquity serve**) utilise ses propres fichiers de configuration et de lancement (dans le dossier **.ubiquity** de votre projet). Il ne doit donc pas être utilisé pour évaluer les résultats des modifications apportées.

Testez vos pages en utilisant une configuration logicielle et matérielle similaire à celle utilisée en production. Utilisez un outil de benchmark pour évaluer vos changements au fur et à mesure (**Apache Bench** par exemple).

39.1 Cache

39.1.1 Système

Choisissez et testez parmi les différents systèmes de cache (ArrayCache, PhpFastCache, MemCached). Le système de cache est défini dans le fichier de configuration :

Code source 1 – app/config/config.php

```
    "cache" => [
            "directory" => "cache/",
            "system" => "Ubiquity\\cache\\system\\ArrayCache",
            "params" => []
    ]
```

Le cache par défaut **ArrayCache** est souvent la solution la plus optimisée.

39.1.2 Génération

Générer le routeur et le cache ORM (pensez que les annotations ne sont jamais utilisées à l'exécution) :

```
Ubiquity init-cache
```

39.1.3 Contenus statiques

Si votre application comporte des pages qui sont générées par PHP mais qui changent rarement, vous pouvez les mettre en cache :

— Le résultat de la requête (utilisant les méthodes **DAO**)
— La réponse de la route (avec l'annotation **@route**)

39.2 fichier index

Supprimez la ligne définissant le signalement des erreurs au moment de l'exécution, et assurez-vous que l'affichage des erreurs est désactivé dans **php.ini**.

Code source 2 – index.php

```
error_reporting(\E_ALL);//To be removed
```

39.3 Optimisation de la configuration

La configuration est accessible depuis le fichier `app/config/config.php`.

Ne gardez que les éléments essentiels à votre application.

key	rôle	Optimisation
siteUrl	Utilisé par les méthodes Ajax, et par les fonctions `url` et `path` de Twig.	A supprimer si ces fonctions ne sont pas utilisées
database	Utilisé par l'ORM Ubiquity	A supprimer si la partie ORM n'est pas utilisée
sessionName	Si elle est assignée, démarre ou récupère la session php pour chaque requête.	A supprimer si la session est inutile
templateEngine	S'il est assigné, instancie un nouvel objet Moteur pour chaque requête.	A supprimer si les vues ne sont pas utilisées
templateEngineOptions	Options attribuées à l'instance du moteur de templates	mettre l'option cache à `true` si Twig est utilisé
test	A enlever (déprécié)	
debug	Active ou désactive les logs	Mettre à `false` en production
logger	Définit l'instance de logger	A enlever en production
di	Définit les services à injecter	La seule clé lue à l'exécution est `@exec`
cache	Définit le chemin du cache et la classe de base du cache, utilisés par les modèles, le routeur, l'injection de dépendance.	
mvcNS	Définit les chemins ou les espaces de noms utilisés par les contrôleurs, les modèles et les contrôleurs Rest	
isRest	Définit la condition pour détecter si un chemin correspond à un contrôleur Rest	A supprimer si vous n'utilisez pas explicitement cette condition dans votre code

Exemple de configuration sans session, et sans injection de dépendances :

Code source 3 – app/config/config.php

```
<?php
return array(
            "templateEngine"=>'Ubiquity\\views\\engine\\Twig',
            "templateEngineOptions"=>array("cache"=>true),
            "debug"=>false,
            "cache"=>["directory"=>"cache/","system"=>"Ubiquity\\cache\\system\\
↪ArrayCache","params"=>[]],
            "mvcNS"=>["models"=>"models","controllers"=>"controllers","rest"=>""]
);
```

39.4 Optimisation des services

Les services chargés sont accessibles depuis le fichier `app/config/services.php`.

Comme pour le fichier de configuration, ne conservez que les éléments essentiels à votre application.

Lignes	Rôle
\Ubiquity\cache\CacheManager : :start-Prod($config)	Démarre le cache pour l'ORM, la base de données, le routeur, l'injection de dépendances.
UbiquityormDAO : :start()	A utiliser uniquement en la présence de plusieurs bases de données
Router : :start()	A utiliser uniquement si les routes sont définies avec des annotations.
Router : :addRoute(« _default », « controllers\IndexController »)	Définit la route par défaut (à supprimer en production)
\Ubiquity\assets\AssetsManager : :start($config)	Définit la variable siteUrl à partir du ThemeManager, à utiliser uniquement si les fonctions `css` et `js` de twig sont utilisées

Exemple d'un fichier Services avec une base de données et le démarrage du routeur :

Code source 4 – app/config/services.php

```
1 <?php
2 \Ubiquity\cache\CacheManager::startProd($config);
3 \Ubiquity\controllers\Router::start();
```

39.5 Optimisation de l'autoloader

En production, supprimez les dépendances utilisées uniquement en développement, et générez le fichier map des classes optimisé :

```
composer install --no-dev --classmap-authoritative
```

Si les dépendances utilisées ont déjà été supprimées et que vous souhaitez uniquement mettre à jour le fichier map (après avoir ajouté ou supprimé une classe) :

```
composer dump-autoload -o  --classmap-authoritative
```

Note : Le paramètre *–no-dev* ` supprime la dépendance `ubiquity-dev` requise par **webtools**. Si vous utilisez webtools en production, ajoutez la dépendance `phpmv/ubiquity-dev` :

```
composer require phpmv/ubiquity-dev
```

39.6 Optimisation PHP

Veuillez noter que d'autres applications peuvent utiliser les valeurs modifiées sur le même serveur.

39.6.1 OP-Cache

OPcache améliore les performances de PHP en stockant le bytecode des scripts précompilés dans la mémoire partagée, ce qui évite à PHP de devoir charger et analyser les scripts à chaque requête.

Code source 5 – php.ini

```
[opcache]
; Determines if Zend OPCache is enabled
opcache.enable=1
```

Code source 6 – php.ini

```
; The OPcache shared memory storage size.
opcache.memory_consumption=256

; The maximum number of keys (scripts) in the OPcache hash table.
; Only numbers between 200 and 1000000 are allowed.
opcache.max_accelerated_files=10000

; When disabled, you must reset the OPcache manually or restart the
; webserver for changes to the filesystem to take effect.
opcache.validate_timestamps=0

; Allow file existence override (file_exists, etc.) performance feature.
opcache.enable_file_override=1

; Enables or disables copying of PHP code (text segment) into HUGE PAGES.
; This should improve performance, but requires appropriate OS configuration.
opcache.huge_code_pages=1
```

Si vous utilisez le serveur web **ubiquity-swoole** :

Code source 7 – php.ini

```
; Determines if Zend OPCache is enabled for the CLI version of PHP
opcache.enable_cli=1
```

39.7 Pour compléter

N'oubliez pas que le framework utilisé ne fait pas tout. Vous devez également optimiser votre propre code.

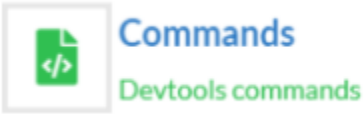
CHAPITRE 40

Ubiquity commandes

Note : Cette partie est accessible à partir des **webtools**, donc si vous avez créé votre projet avec l'option **-a** ou avec la commande **create-project**.

40.1 Commandes

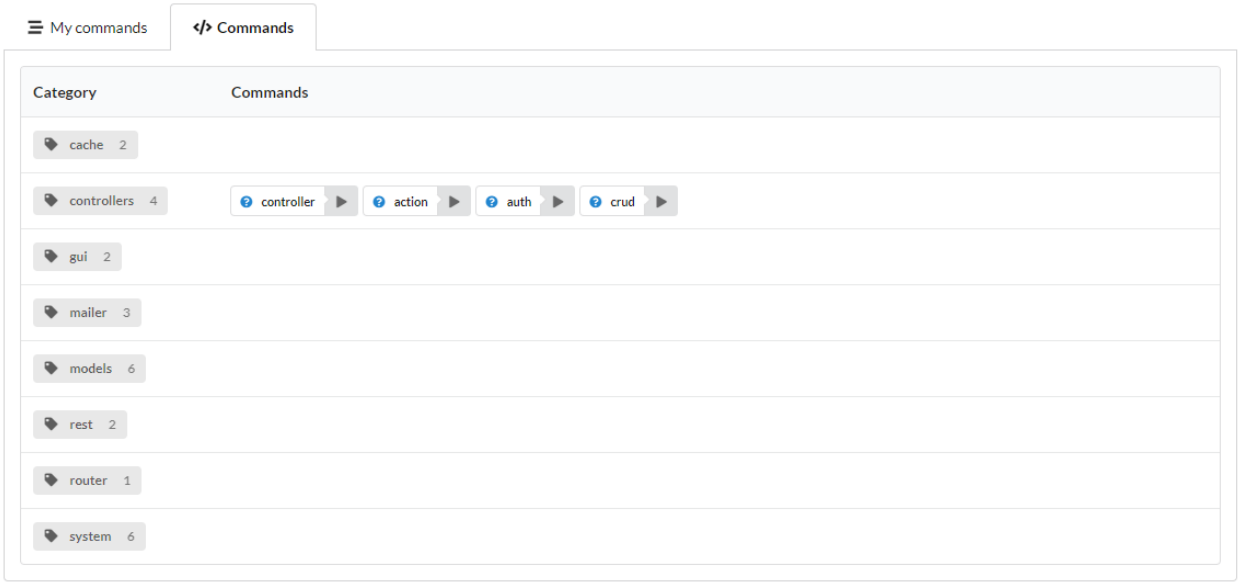
A partir des webtools, aller dans la partie **commands**,



ou directement à l'adresse `http://127.0.0.1:8090/Admin/commands`.

40.1.1 Liste des commandes

Activer l'onglet **Commands** pour obtenir la liste des commandes devtools.

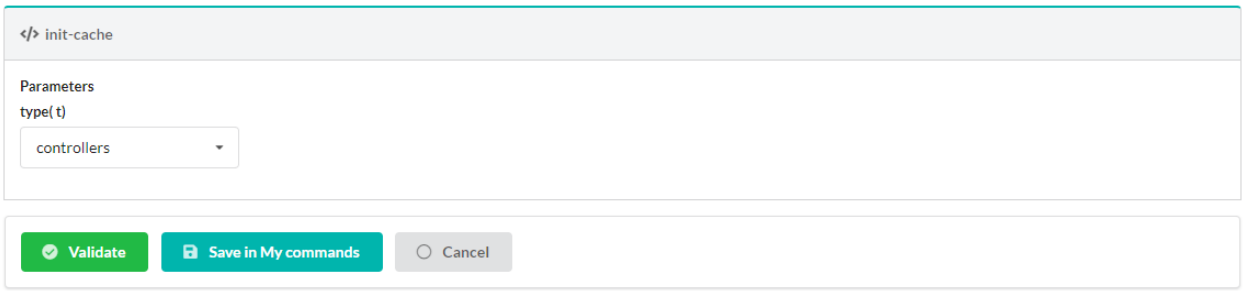


40.1.2 Informations sur une commande

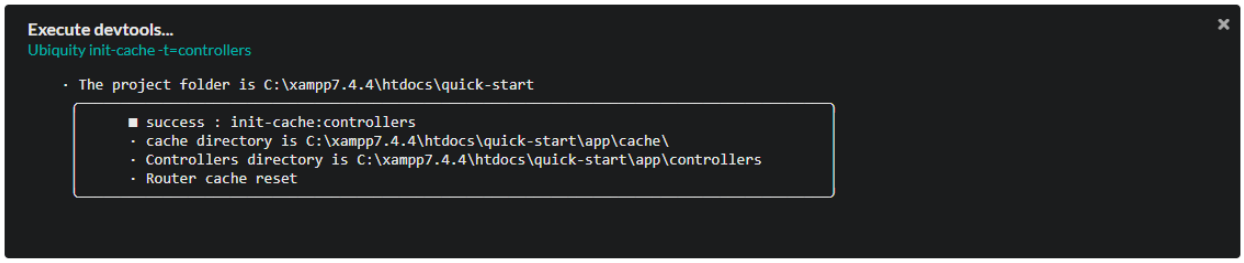
Il est possible d'obtenir de l'aide sur une commande (ce qui produit un résultat équivalent à `Ubiquity help cmdName`).

40.1.3 Exécution de commandes

Un clic sur le bouton run d'une commande affiche un formulaire pour saisir ses paramètres (ou l'exécute directement si elle n'en prend aucun).



Après avoir saisi les paramètres, l'exécution produit un résultat.

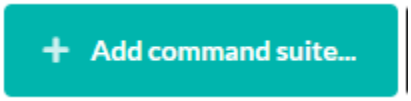


40.2 Suite de commandes

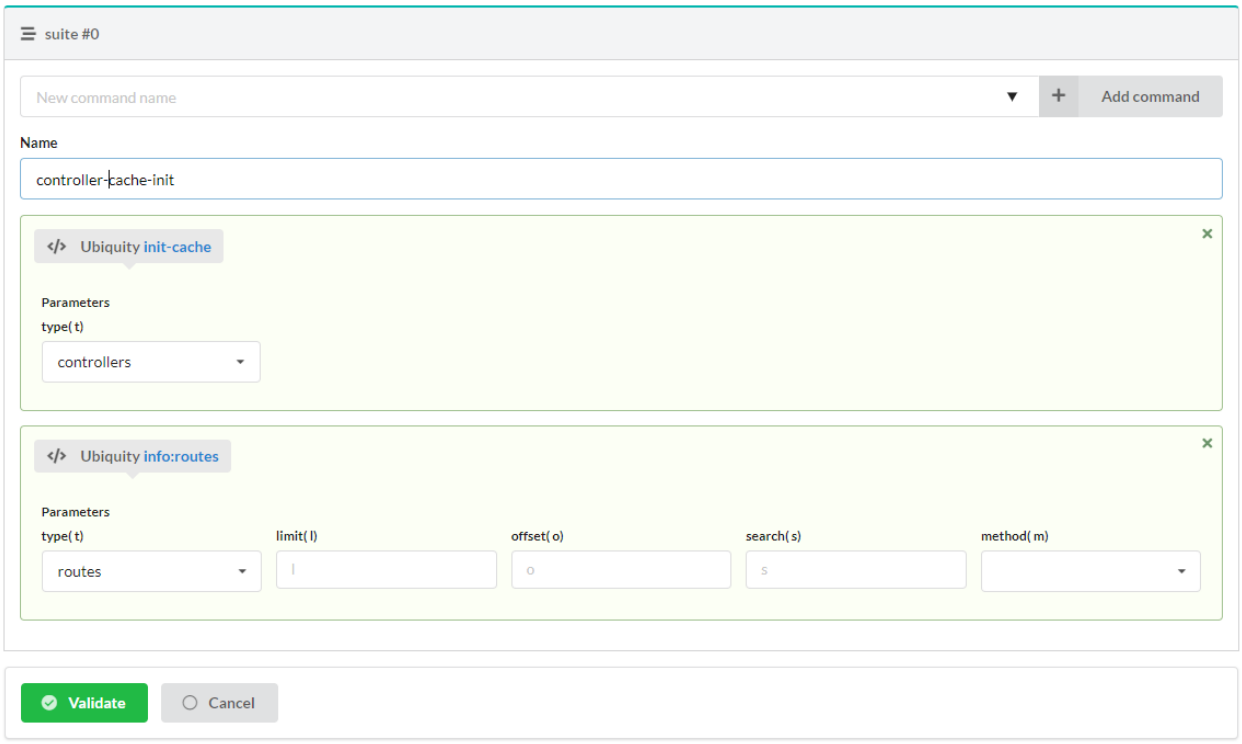
Retourner à l'onglet **My commands** : Il est possible de sauvegarder une séquence de commandes (avec des paramètres enregistrés), puis d'exécuter cette même séquence :

40.2.1 Création de suite

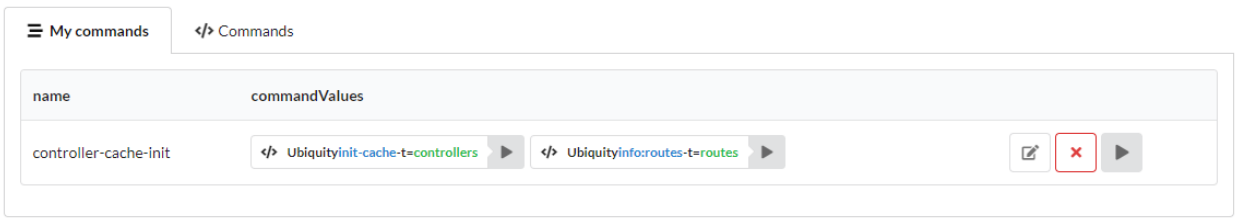
Cliquer sur **add command suite**



Ajouter les commandes souhaitées et modifier les paramètres :

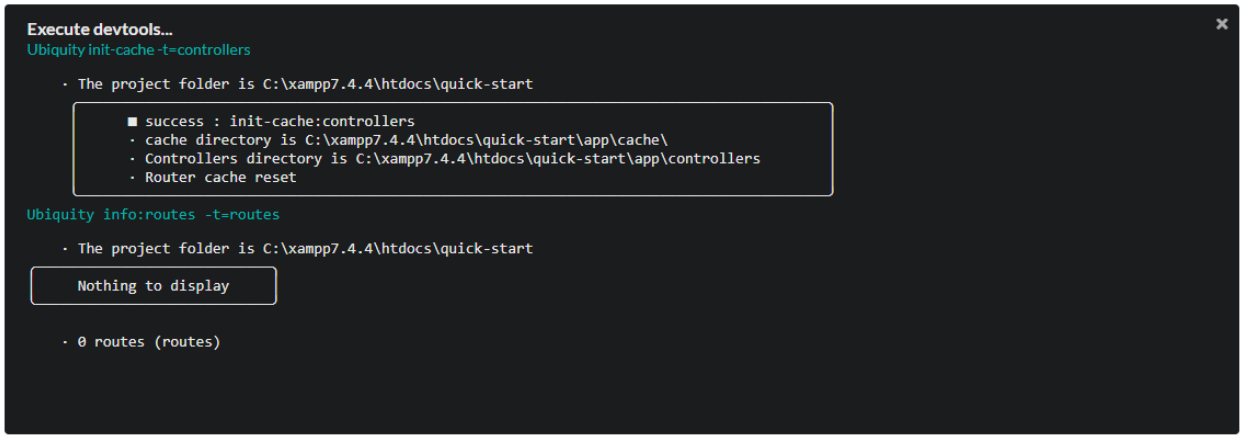


La validation génère la suite :



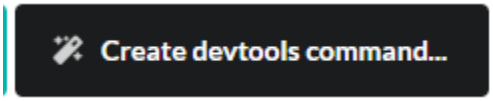
40.2.2 Exécution d'une suite de commandes

En cliquant sur le bouton « run » de la suite, on exécute la liste des commandes qu'elle contient :



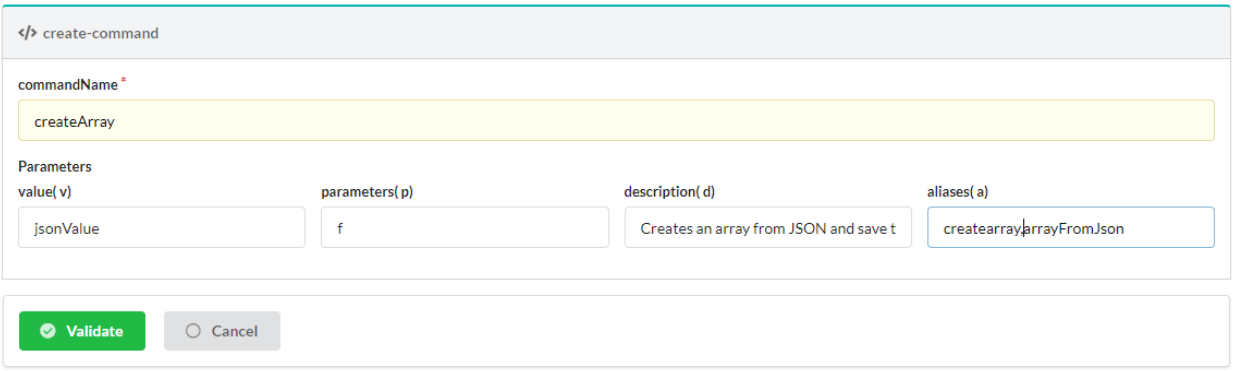
40.3 Création d'une commande personnalisée

Cliquer sur le bouton **Create devtools command**.

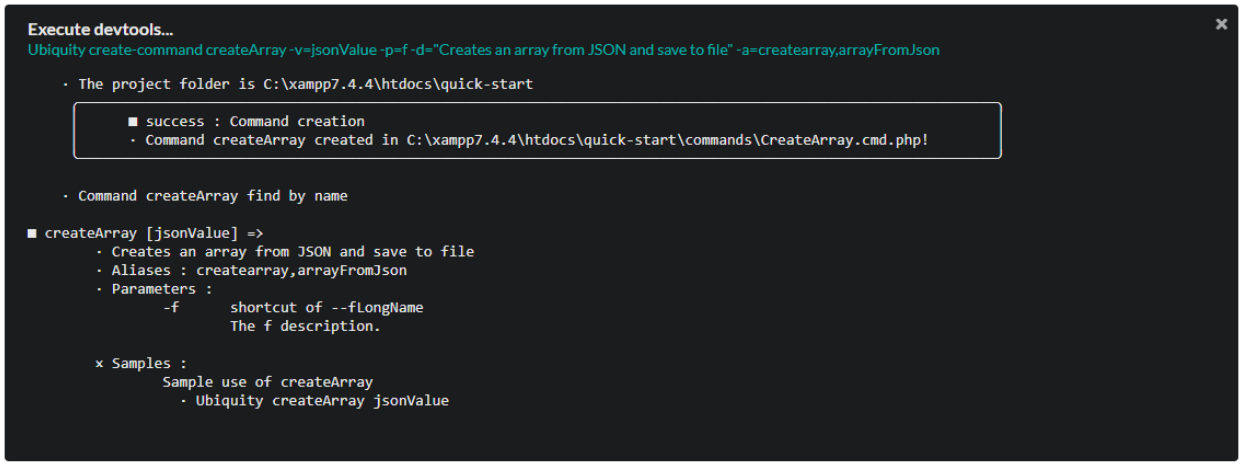


Saisir les caractéristiques de la nouvelle commande :

- Le nom de la commande
- Sa valeur : nom de l'argument principal
- Les paramètres de la commande : avec plusieurs paramètres, les séparer avec une virgule
- La description de la commande
- Les alias de la commande : En cas d'alias multiples, utilisez la virgule comme séparateur.



Note : Les commandes personnalisées sont créées dans le dossier **app/commands** du projet.



La classe générée :

Code source 1 – app/commands/CreateArray.php

```
1 namespace commands;
2
3 use Ubiquity\devtools\cmd\commands\AbstractCustomCommand;
```

(suite sur la page suivante)

(suite de la page précédente)

```
use Ubiquity\devtools\cmd\ConsoleFormatter;
use Ubiquity\devtools\cmd\Parameter;

class CreateArray extends AbstractCustomCommand {

	protected function getValue(): string {
		return 'jsonValue';
	}

	protected function getAliases(): array {
		return array("createarray","arrayFromJson");
	}

	protected function getName(): string {
		return 'createArray';
	}

	protected function getParameters(): array {
		return ['f' => Parameter::create('fLongName', 'The f description.', [])];
	}

	protected function getExamples(): array {
		return ['Sample use of createArray'=>'Ubiquity createArray jsonValue'];
	}

	protected function getDescription(): string {
		return 'Creates an array from JSON and save to file';
	}

	public function run($config, $options, $what, ...$otherArgs) {
		//TODO implement command behavior
		echo ConsoleFormatter::showInfo('Run createArray command');
	}
}
```

La commande **CreateArray** implémentée :

Code source 2 – app/commands/CreateArray.php

```
namespace commands;

use Ubiquity\devtools\cmd\commands\AbstractCustomCommand;
use Ubiquity\devtools\cmd\ConsoleFormatter;
use Ubiquity\devtools\cmd\Parameter;
use Ubiquity\utils\base\UFileSystem;

class CreateArray extends AbstractCustomCommand {

	protected function getValue(): string {
		return 'jsonValue';
	}

```

(suite sur la page suivante)

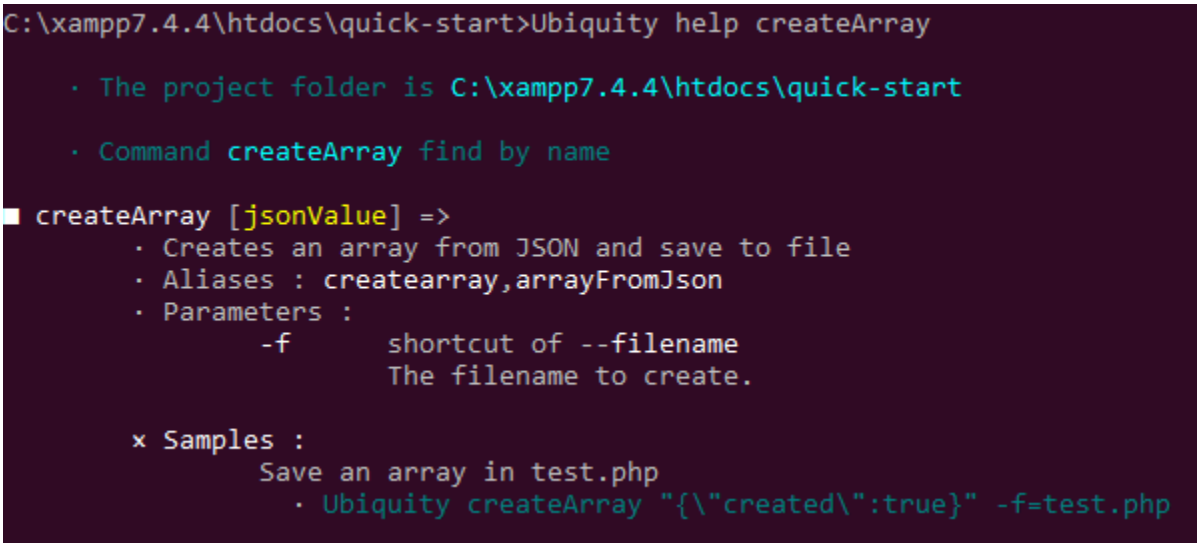
(suite de la page précédente)

```
14      protected function getAliases(): array {
15              return array(
16                      "createarray",
17                      "arrayFromJson"
18              );
19      }
20
21      protected function getName(): string {
22              return 'createArray';
23      }
24
25      protected function getParameters(): array {
26              return [
27                      'f' => Parameter::create('filename', 'The filename to create.', [])
28              ];
29      }
30
31      protected function getExamples(): array {
32              return [
33                      'Save an array in test.php' => "Ubiquity createArray \"{\\\"created\
↪\\":true}\" -f=test.php"
34              ];
35      }
36
37      protected function getDescription(): string {
38              return 'Creates an array from JSON and save to file';
39      }
40
41      public function run($config, $options, $what, ...$otherArgs) {
42              echo ConsoleFormatter::showInfo('Run createArray command');
43              $array = \json_decode($what, true);
44              $error = \json_last_error();
45              if ($error != 0) {
46                      echo ConsoleFormatter::showMessage(\json_last_error_msg(), 'error');
47              } else {
48                      $filename = self::getOption($options, 'f', 'filename');
49                      if ($filename != null) {
50                              UFileSystem::save($filename, "<?php\nreturn " . var_export(
↪$array, true) . ";\n");
51                              echo ConsoleFormatter::showMessage("$filename succefully␣
↪created!", 'success', 'CreateArray');
52                      } else {
53                              echo ConsoleFormatter::showMessage("Filename must have a␣
↪value!", 'error');
54                      }
55              }
56      }
57  }
```

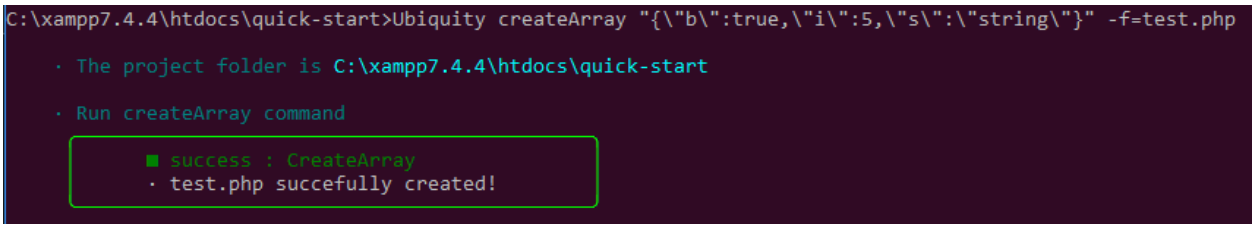
40.3.1 Exécution d'une commande personnalisée

La nouvelle commande est accessible depuis les devtools, à condition qu'elle soit bien présente dans le projet :

```
Ubiquity help createArray
```




```
Ubiquity createArray "{\"b\":true,\"i\":5,\"s\":\"string\"}" -f=test.php
```



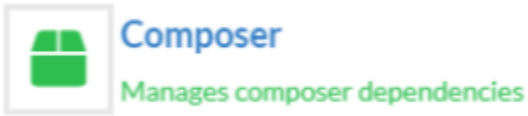
CHAPITRE 41

Gestion Composer

Note : Cette partie est accessible depuis les **webtools**, donc si vous avez créé votre projet avec l'option **-a** ou avec la commande **create-project**.

41.1 Accès

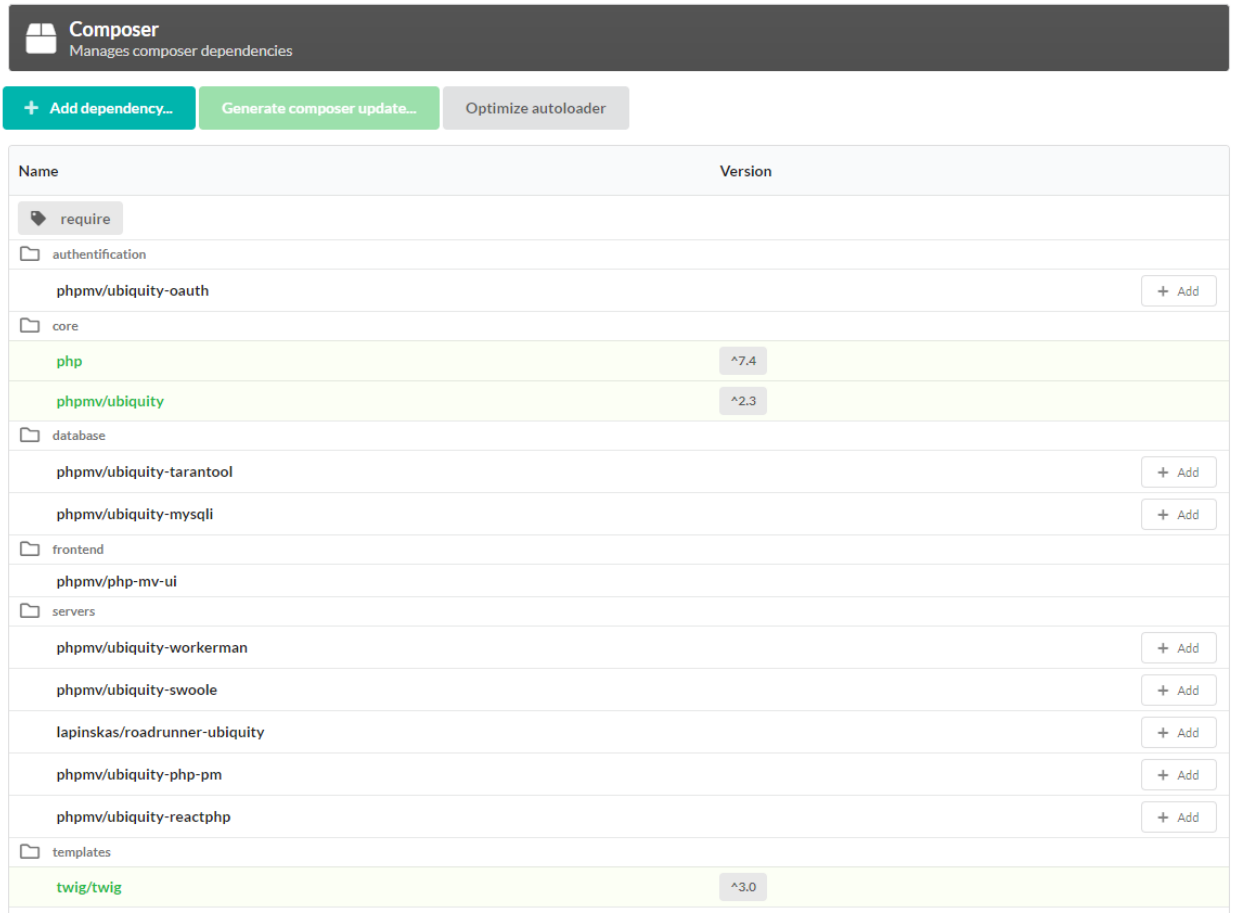
Depuis les webtools, activer la patir **composer**.



ou aller directement à l'adresse `http://127.0.0.1:8090/Admin/composer`.

41.2 Liste des dépendances

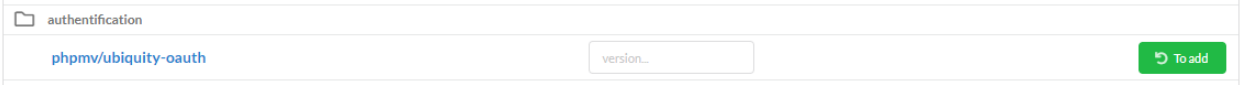
L'interface affiche la liste des dépendances déjà installées, et celles qui sont directement installables.



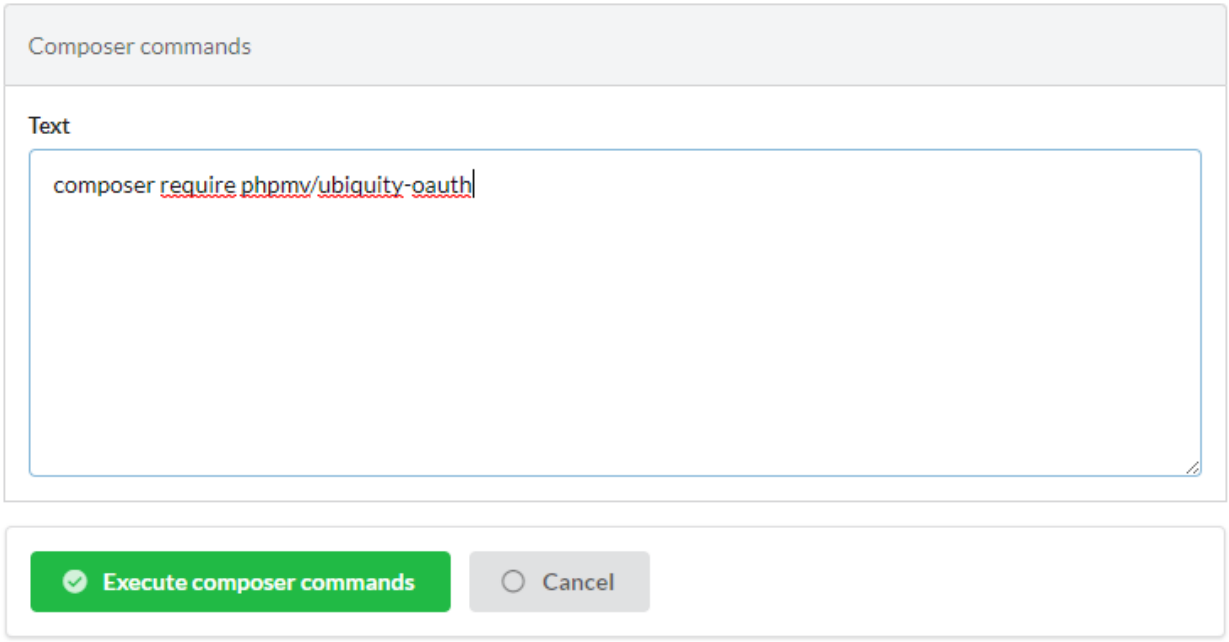
41.3 Installation de dépendance

41.3.1 Parmi les dépendances listées :

Cliquez sur le bouton **add** des dépendances que vous voulez ajouter.



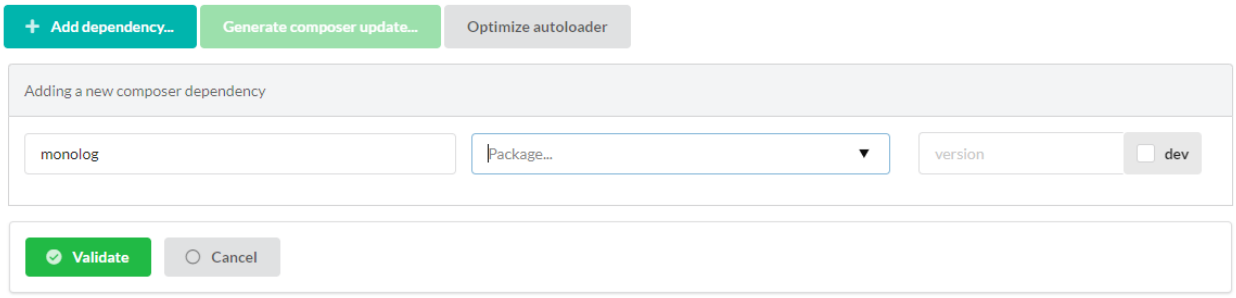
Puis cliquer sur le bouton **Generate composer update** :



La validation génère la mise à jour.

41.3.2 Pour les dépendances non listées :

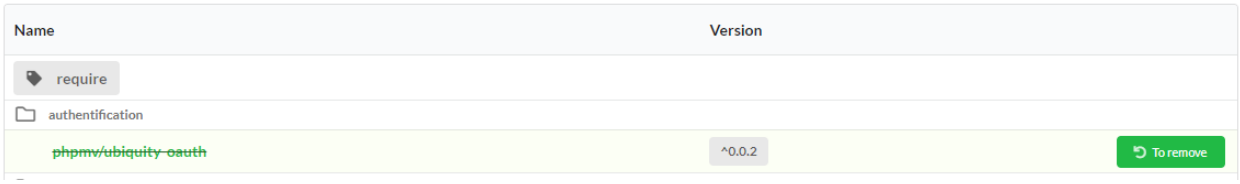
Cliquer sur le bouton **Add dependency** :



— Saisir le nom du vendor (provider) ;
— Sélectionner le package dans la liste :
— Sélectionnez éventuellement une version (si aucune ne l'est, la dernière version stable sera installée).

41.4 Suppression de dépendance

Cliquez sur le bouton **remove** des dépendances que vous voulez supprimer.



Cliquer ensuite sur le bouton **Generate composer update** puis valider la mise à jour.

Note : Il est possible d'effectuer plusieurs opérations d'addition ou de suppression et de les valider ensemble.

41.5 Optimisation Composer

Cliquer sur le bouton **Optimize autoloader**.

L'optimisation de l'autoload se fait avec l'option authoritative classmap.

CHAPITRE 42

Mise en cache Ubiquity

CHAPITRE 43

Dépendances Ubiquity

— `php >=7.4`
— `phpmv/ubiquity` => Ubiquity core

43.1 En production

43.1.1 Moteur de template

Twig est nécessaire s'il est utilisé en tant que moteur de template, ce qui n'est pas une obligation.
— `twig/twig` => Template engine

43.1.2 Sécurité

— `phpmv/ubiquity-security` => Csrf, Csp…
— `phpmv/ubiquity-acl` => Management des contrôles d'accès

43.2 En développement

43.2.1 Webtools

— `phpmv/ubiquity-dev` => dev classes pour les webtools et les devtools depuis la v2.3.0
— `phpmv/php-mv-ui` => Librairie front
— `mindplay/annotations` => Librairie pour annotations , requis pour générer le cache, les modèles…
— `monolog/monolog` => Librairie pour les logs
— `czproject/git-php` => Opérations git (+ git console requise)

43.2.2 Devtools

— `phpmv/ubiquity-devtools` => Cli console
— `phpmv/ubiquity-dev` => dev classes pour les webtools et les devtools depuis la v2.3.0
— `mindplay/annotations` => Librairie pour annotations , requis pour générer le cache, les modèles…

43.2.3 Tests

— `codeception/codeception` => Tests
— `codeception/c3` => C3 integration
— `phpmv/ubiquity-codeception` => Codeception pour Ubiquity

CHAPITRE 44

Module client OAuht2

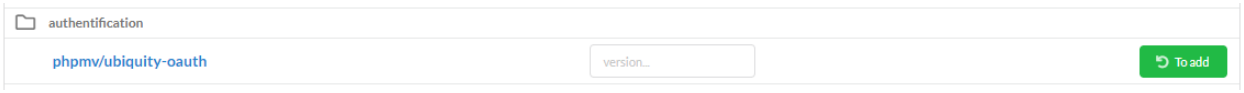
Note : Cette partie est accessible depuis les **webtools**, donc si vous avez créé votre projet avec l'option **-a** ou avec la commande **create-project**. Le module OAuth n'est pas installé par défaut. Il utilise la bibliothèque HybridAuth.

44.1 Installation

Depuis la racine du projet :

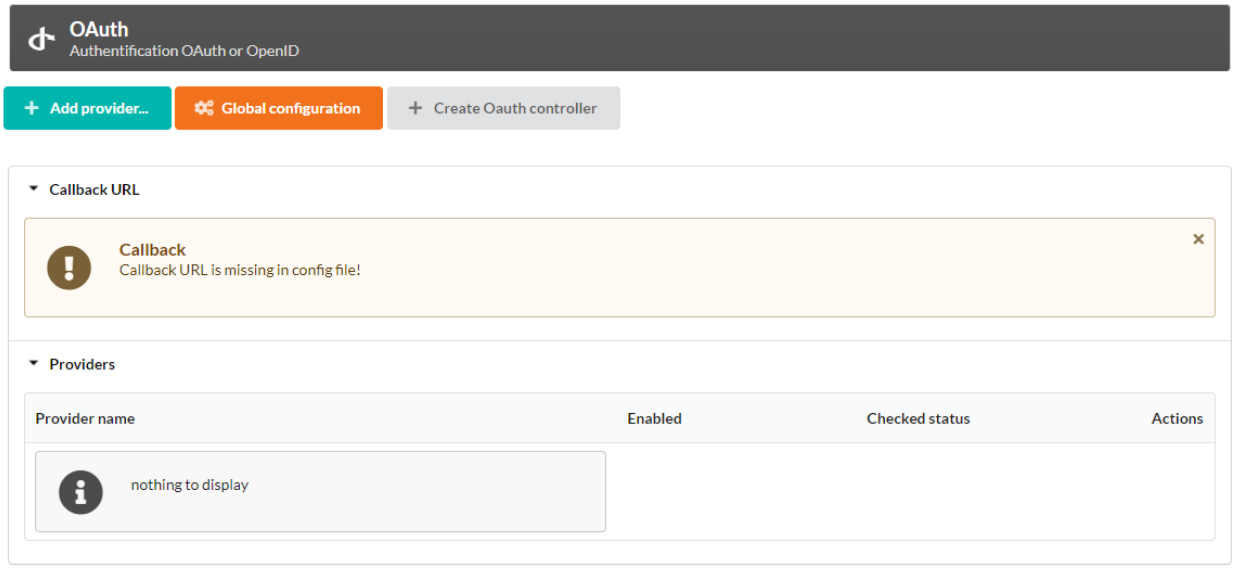
```
composer require phpmv/ubiquity-oauth
```

Note : Il est également possible d'ajouter la dépendance **ubiquity-oauth** en utilisant la partie **Composer** du module d'administration.

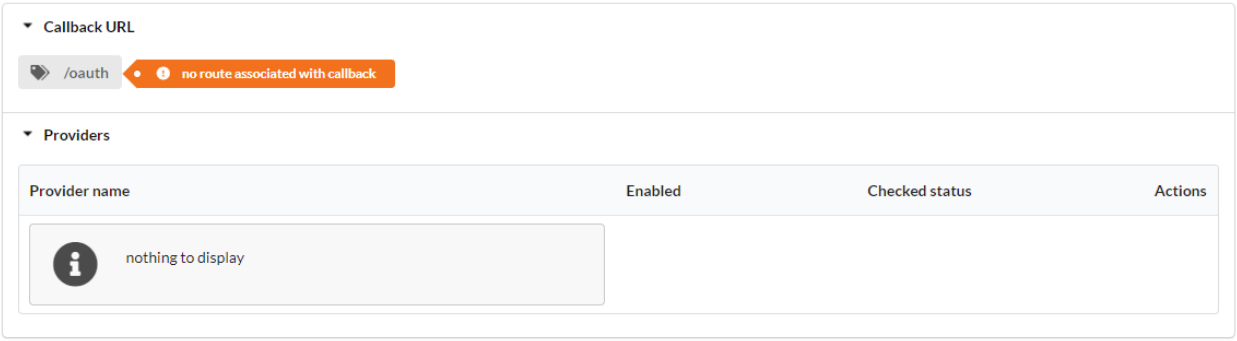


44.2 Configuration OAuth

44.2.1 Configuration globale

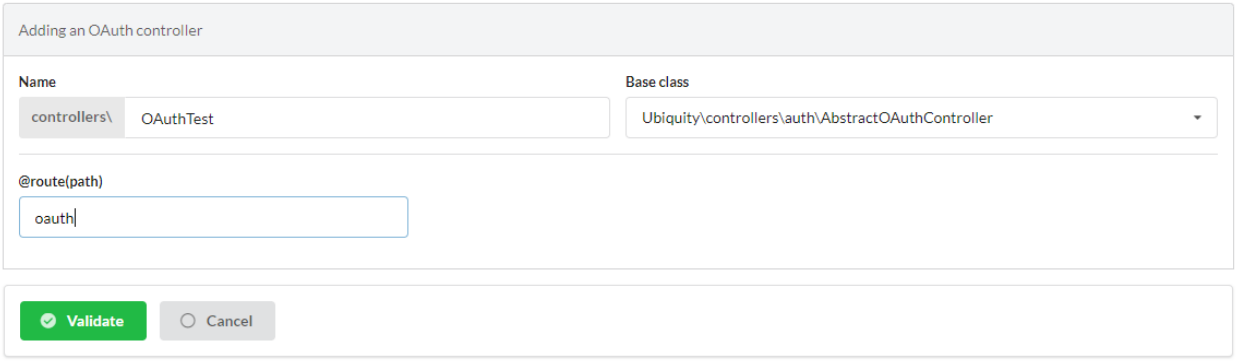


Cliquez sur le bouton **Global configuration**, et modifiez l'URL de rappel, qui correspond à l'url de rappel locale après une connexion réussie.

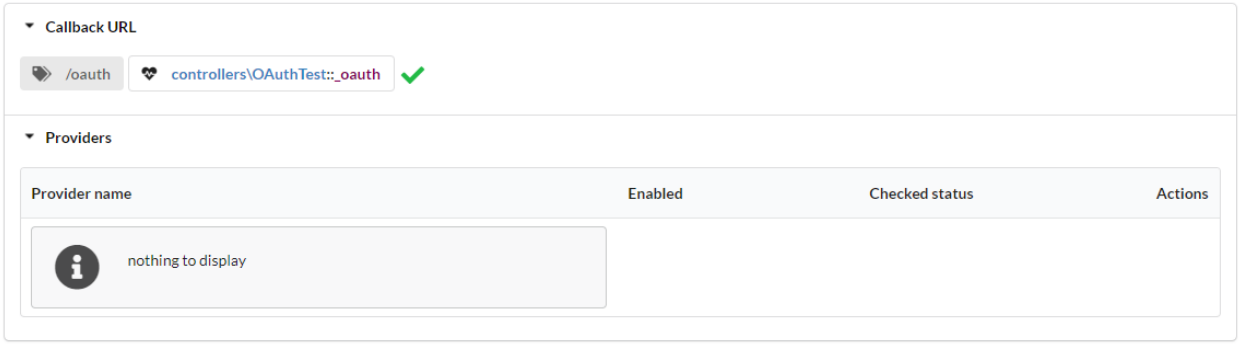


44.2.2 Contrôleur OAuth

Cliquez sur le bouton **Create Oauth controller** et attribuez à la route la valeur précédemment donnée au callback :



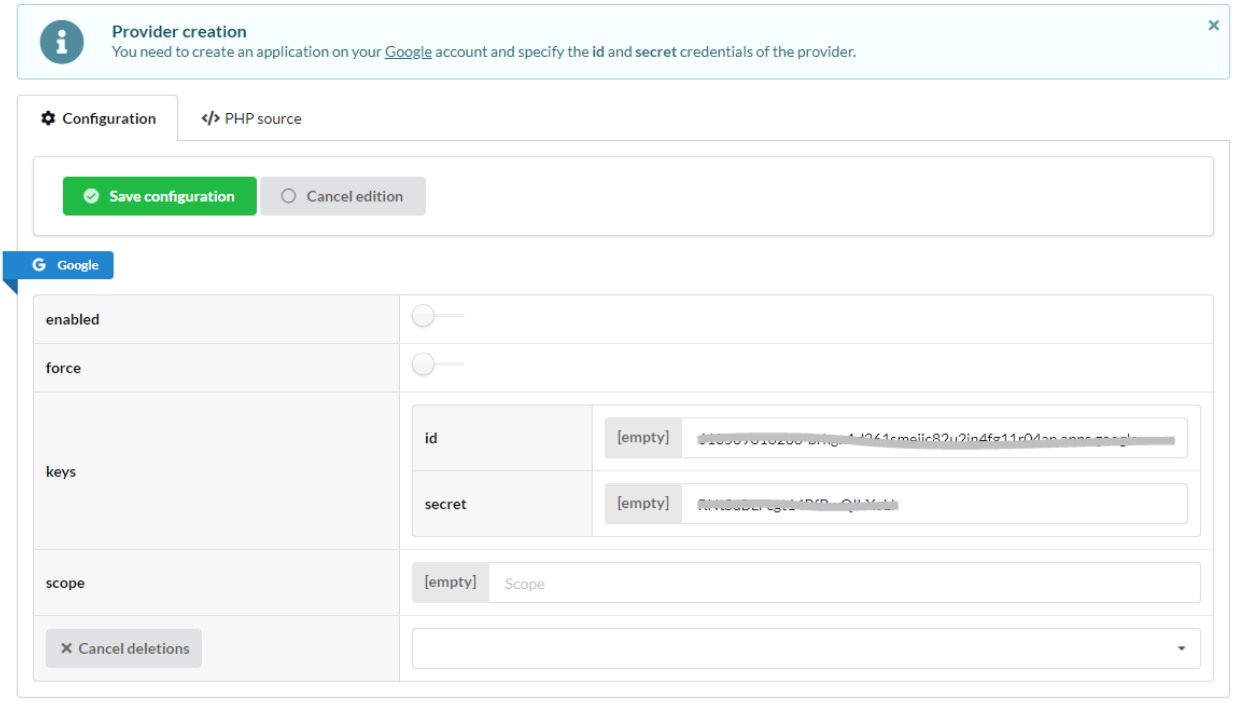
Valider et réinitialiser le cache du routeur :



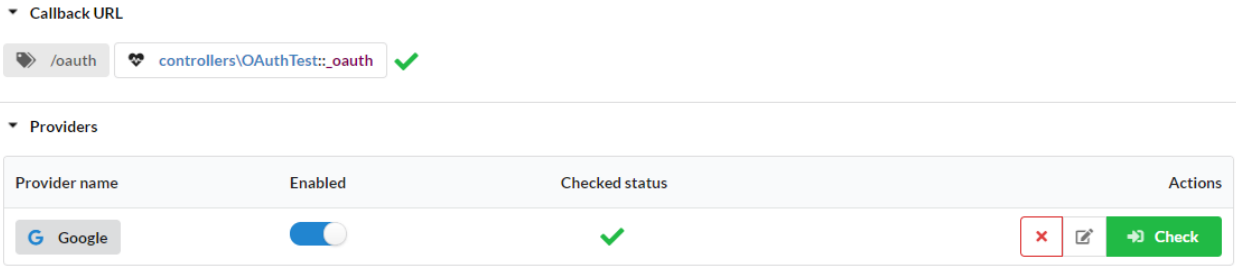
44.2.3 Fournisseurs

Note : Pour une authentification OAuth, il est nécessaire de créer au préalable une application chez le fournisseur, et de prendre connaissance des clés de l'application (id et secret).

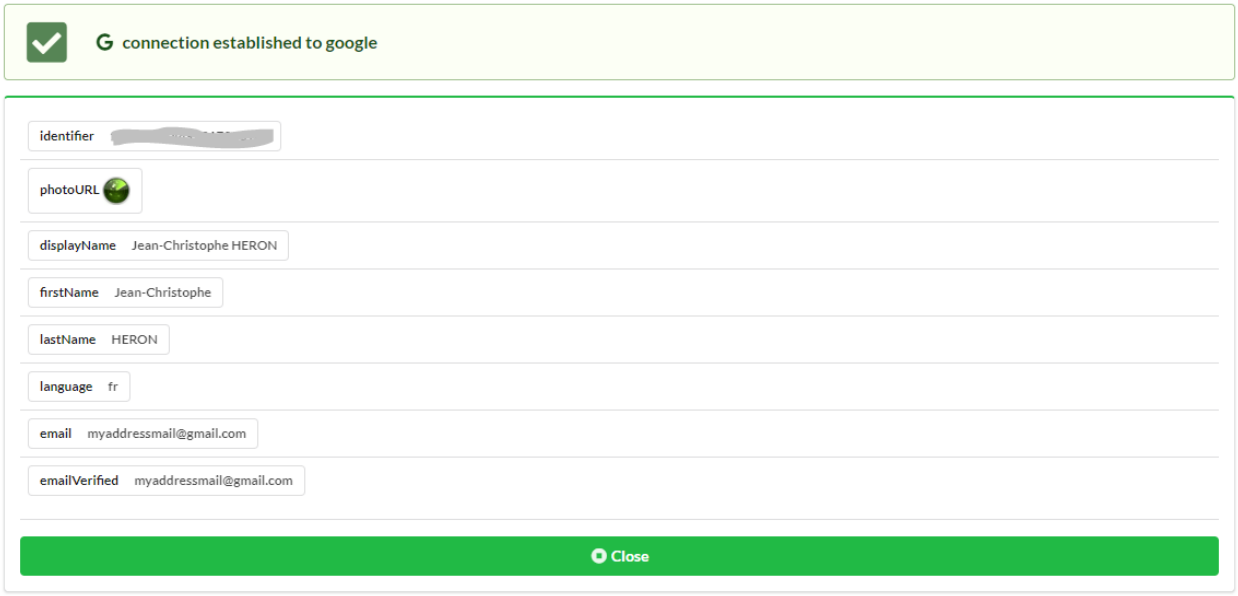
Cliquer sur le bouton **Add provider** et sélectionner **Google** :



Vérifiez la connexion en cliquant sur le bouton **Check** :



Information post connexion :



44.3 Personnalisation de OAuthController

Le contrôleur créé est le suivant :

Code source 1 – app/controllers/OAuthTest.php

```
namespace controllers;
use Hybridauth\Adapter\AdapterInterface;
/**
 * Controller OAuthTest
 */
class OAuthTest extends \Ubiquity\controllers\auth\AbstractOAuthController{

    public function index(){
    }

    /**
     * @get("oauth/{name}")
     */
    public function _oauth(string $name):void {
        parent::_oauth($name);
    }

    protected function onConnect(string $name,AdapterInterface $provider){
        //TODO
    }
}
```

— La méthode **_oauth** correspond à l'url de callback
— La méthode **onConnect** est déclenchée à la connexion et peut être surchargée.

Exemple :
— Récupération possible d'un utilisateur associé dans la base de données

— ou création d'un nouvel utilisateur
— Ajout de l'utilisateur connecté et redirection

Code source 2 – app/controllers/OAuthTest.php

```
protected function onConnect(string $name, AdapterInterface $provider) {
    $userProfile = $provider->getUserProfile();
    $key = md5($name . $userProfile->identifier);
    $user = DAO::getOne(User::class, 'oauth= ?', false, [
        $key
    ]);
    if (! isset($user)) {
        $user = new User();
        $user->setOauth($key);
        $user->setLogin($userProfile->displayName);
        DAO::save($user);
    }
    USession::set('activeUser', $user);
    \header('location:/');
  }
```

CHAPITRE 45

Plateformes asynchrones

Note : Ubiquity supporte plusieurs plateformes : Swoole, Workerman, RoadRunner, PHM-PM,ngx_php.

45.1 Swoole

Installer l'extension Swoole sur votre système (linux) ou dans votre image Docker :

```
#!/bin/bash
pecl install swoole
```

Lancer Ubiquity Swoole (la première fois, le paquet **ubiquity-swoole** sera installé) :

```
Ubiquity serve -t=swoole
```

45.1.1 Configuration du serveur

Code source 1 – .ubiquity/swoole-config.php

```
<?php
return array(
    "host" => "0.0.0.0",
    "port" => 8080,
    "options"=>[
        "worker_num" => \swoole_cpu_num() * 2,
            "reactor_num" => \swoole_cpu_num() * 2
        ]
);
```

Le port peut aussi être changé au démarrage du serveur :

```
Ubiquity serve -t=swoole -p=8999
```

45.1.2 Optimisation des services

Le démarrage des services ne sera effectué qu'une seule fois, au démarrage du serveur.

Code source 2 – app/config/services.php

```
\Ubiquity\cache\CacheManager::startProd($config);
\Ubiquity\orm\DAO::setModelsDatabases([
    'models\\Foo' => 'default',
    'models\\Bar' => 'default'
]);

\Ubiquity\cache\CacheManager::warmUpControllers([
    \controllers\IndexController::class,
    \controllers\FooController::class
]);

$swooleServer->on('workerStart', function ($srv) use (&$config) {
    \Ubiquity\orm\DAO::startDatabase($config, 'default');
    \controllers\IndexController::warmup();
    \controllers\FooController::warmup();
});
```

La méthode warmUpControllers :

- Instancie les contrôleurs
- Effectue l'injection de dépendances
- prépare l'appel des méthodes initialize et finalize (initialisation des constantes d'appel)

Au démarrage de chaque Worker, la méthode **warmup** des contrôleurs peut par exemple initialiser les requêtes DAO préparées :

Code source 3 – app/controllers/FooController.php

```
    public static function warmup() {
            self::$oneFooDao = new DAOPreparedQueryById('models\\Foo');
            self::$allFooDao = new DAOPreparedQueryAll('models\\Foo');
    }
```

45.2 Workerman

Workerman ne nécessite aucune installation particulière (sauf pour **libevent** qu'il est recommandé d'utiliser en production pour des raisons de performance).

Lancer Ubiquity Workerman (la première fois, le paquet **ubiquity-workerman** sera installé) :

```
Ubiquity serve -t=workerman
```

45.2.1 Configuration du serveur

Code source 4 – .ubiquity/workerman-config.php

```
<?php
return array(
    "host" => "0.0.0.0",
    "port" => 8080,
    "socket"=>[
        "count" => 4,
        "reuseport" =>true
    ]
);
```

Le port peut aussi être changé au démarrage du serveur :

```
Ubiquity serve -t=workerman -p=8999
```

45.2.2 Optimisation des services

Le démarrage des services ne sera effectué qu'une seule fois, au démarrage du serveur.

Code source 5 – app/config/services.php

```
\Ubiquity\cache\CacheManager::startProd($config);
\Ubiquity\orm\DAO::setModelsDatabases([
     'models\\Foo' => 'default',
     'models\\Bar' => 'default'
]);

\Ubiquity\cache\CacheManager::warmUpControllers([
     \controllers\IndexController::class,
     \controllers\FooController::class
]);

$workerServer->onWorkerStart = function () use ($config) {
     \Ubiquity\orm\DAO::startDatabase($config, 'default');
     \controllers\IndexController::warmup();
     \controllers\FooController::warmup();
});
```

45.3 ngx_php

//TODO

45.4 Roadrunner

//TODO

CHAPITRE 46

Indices and tables

— genindex
— modindex
— search