ubiquity-framework Documentation

phpmv

05 de febrero de 2023

Inicio rápido

1. Inicio rápido con consola 1
2. Inicio rápido con Webtools 9
3. Instalación de Ubiquity-devtools 21
4. Creación de proyectos 23
5. Configuración del proyecto 25
6. Uso de Devtools 31
7. URLs 37
8. Router (Rutas) 41
9. Controladores 59
10. Eventos 67
11. Inyección de dependencia 73
12. Controladores CRUD 79
13. Auth Controllers 95
14. Base de datos 109
15. Generación de modelos 113
16. ORM 115
17. DAO 127
18. Peticiones (Request) 137
19. Respuesta (Response) 141
20. Sesión (Session) 145

21. Cookie 149
22. Views 151
23. Assets 155
24. Themes 157
25. jQuery Semantic-UI 167
26. Normalizers (Normalizadores) 177
27. Validators (Validadores) 179
28. Transformers 185
29. Translation module 193
30. Security 197
31. Security module 201
32. ACL management 211
33. Rest 221
34. Webtools 249
35. Contribución 255
36. Guía de codificación 259
37. Guía de documentación 265
38. Configuración de servidores 267
39. Optimización de Ubiquity 273
40. Comandos de Ubiquity 279
41. Gestión de composer 287
42. Caché de Ubiquity 291
43. Dependencias en Ubiquity 293
44. Módulo cliente OAuth2 295
45. Plataformas asíncronas 301
46. Índices y tablas 305

CAPÍTULO 1

Inicio rápido con consola

Nota: Si no te gusta el modo consola, puedes cambiar a inicio rápido con *web tools (UbiquityMyAdmin)*.

1.1 Instalar Composer

ubiquity utiliza Composer para gestionar sus dependencias. Por lo tanto, antes de usar, usted tendrá que asegurarse de que tiene Composer instalado en su máquina.

1.2 Instalar Ubiquity-devtools

Descargue el instalador de Ubiquity-devtools utilizando Composer.

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

Pruebe su reciente instalación haciendo:

```
Ubiquity version
```

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

Puedes obtener en todo momento ayuda con un comando tecleando: `Ubiquity help` seguido de lo que buscas.

Ejemplo :

```
Ubiquity help project
```

1.3 Creación de proyectos

Crear el proyecto **quick-start

```
Ubiquity new quick-start
```

1.4 Estructura del directorio

El proyecto creado en la carpeta **quick-start** tiene una estructura sencilla y legible:

la carpeta **app** contiene el código de tu futura aplicación:

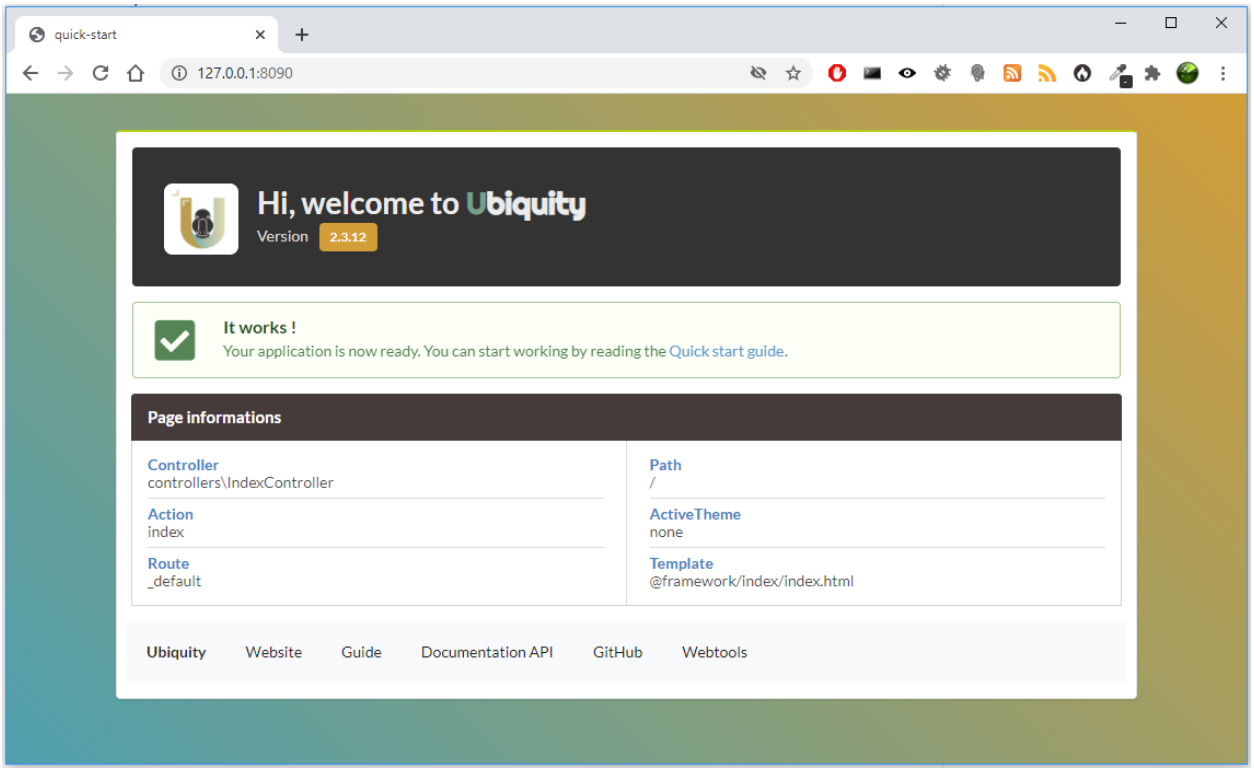
```
app
  ├ cache
  ├ config
  ├ controllers
  ├ models
  └ views
```

1.5 Start-up

Vaya a la carpeta recién creada **quick-start** e inicie el servidor php incorporado:

```
Ubiquity serve
```

Compruebe el correcto funcionamiento en la dirección **http://127.0.0.1:8090**:



Nota: Si el puerto 8090 está ocupado, puede iniciar el servidor en otro puerto utilizando la opción -p.

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

1.6 Controlador

La aplicación de consola **dev-tools** ahorra tiempo en operaciones repetitivas. Pasamos por ella para crear un controlador.

```
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
```

Podemos entonces editar el archivo `app/controllers/DefaultController` en nuestro IDE favorito:

Lista 1: app/controllers/DefaultController.php

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

Añada el mensaje tradicional y pruebe su página en `http://127.0.0.1:8090/DefaultController`.

Lista 2: app/controllers/DefaultController.php

```
class DefaultController extends ControllerBase{

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

}
```

Por ahora, no hemos definido rutas, por lo que el acceso a la aplicación se realiza según el siguiente esquema: `controllerName/actionName/param`

La acción por defecto es el método **index**, no necesitamos especificarlo en la url.

1.7 Route

Importante: El enrutamiento se define con el atributo `Route` (con php>8) o la anotación `@route` y no se hace en un fichero de configuración: es una elección de diseño.

El parámetro **automated** puesto a **true** permite definir los métodos de nuestra clase como sub-rutas de la ruta principal `/hello`.

Con anotaciones:

Lista 3: app/controllers/DefaultController.php

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

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

}
```

Con atributos (php>8):

Lista 4: app/controllers/DefaultController.php

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

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

    public function index(){
```

(continué en la próxima página)

(proviene de la página anterior)

```
        echo 'Hello world!';
    }

}
```

1.7.1 Router cache

Importante: Ningún cambio en las rutas es efectivo sin inicializar la caché. Las anotaciones nunca se leen en tiempo de ejecución. Esto también es una elección de diseño.

Podemos utilizar la consola para la reinicialización de la caché:

```
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
```

Comprobemos que la ruta existe:

```
Ubiquity info:routes
```

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

```
   · 1 routes (routes)
```

Ahora podemos probar la página en `http://127.0.0.1:8090/hello`.

1.8 Acción y ruta con parámetros

Ahora crearemos una acción (sayHello) con un parámetro (name), y la ruta asociada (to): La ruta utilizará el parámetro **name** de la acción:

```
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
```

Tras reinicializar la caché (comando **init-cache**), debería aparecer el comando **info:routes**:

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

· 2 routes (routes)

Cambia el código en tu IDE: la acción debe decir Hola a alguien…

Lista 5: app/controllers/DefaultController.php

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

y prueba la página en `http://127.0.0.1:8090/hello/to/Mr SMITH`

1.9 Acción, parámetros de ruta y vista

Ahora crearemos una acción (information) con dos parámetros (title and message), la ruta asociada (info) y una vista para mostrar el mensaje: |br|La ruta utilizará los dos parámetros de la acción.

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

Nota: El parámetro -v (–view) se utiliza para crear la vista asociada a la acción.

Tras reinicializar la caché, ahora tenemos 3 rutas:

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

· 3 routes (routes)

Volvamos a nuestro entorno de desarrollo y veamos el código generado:

Lista 6: app/controllers/DefaultController.php

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

Necesitamos pasar las 2 variables a la vista:

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

Y utilizamos nuestras 2 variables en la vista twig asociada:

Lista 7: app/views/DefaultController/information.html

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

Podemos probar nuestra página en `http://127.0.0.1:8090/hello/info/Quick start/Ubiquity is quiet simple` Es obvio

CAPÍTULO 2

Inicio rápido con Webtools

2.1 Instalar Composer

ubiquity utiliza Composer para gestionar sus dependencias. Por lo tanto, antes de usar, usted tendrá que asegurarse de que tiene Composer instalado en su máquina.

2.2 Instalar Ubiquity-devtools

Descargue el instalador de Ubiquity-devtools utilizando Composer.

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

Pruebe su reciente instalación haciendo:

```
Ubiquity version
```

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

Puedes obtener en todo momento ayuda con un comando tecleando: `Ubiquity help` seguido de lo que buscas.

Ejemplo :

```
Ubiquity help project
```

2.3 Creación de proyectos

Crear el proyecto **quick-start** con la interfaz **Webtools** (la opción **-a**)

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

2.4 Estructura del directorio

El proyecto creado en la carpeta **quick-start** tiene una estructura sencilla y legible:

la carpeta **app** contiene el código de tu futura aplicación:

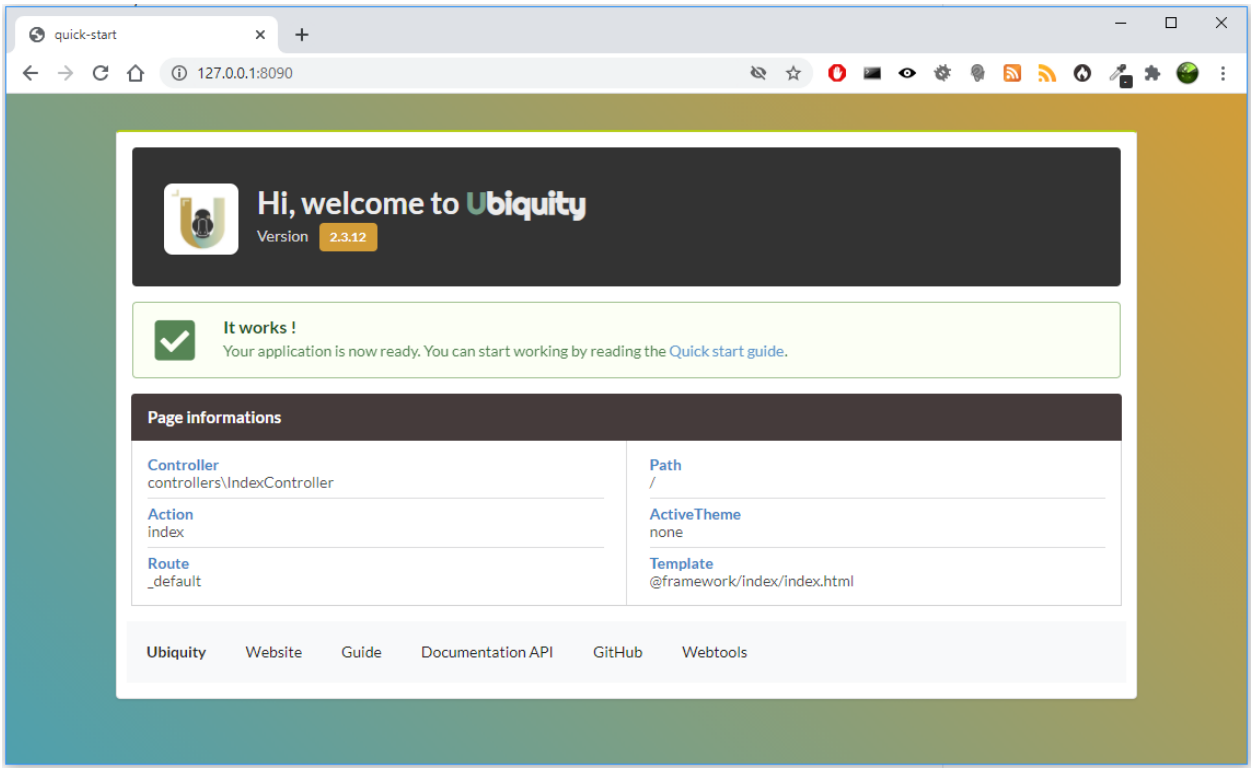
```
app
  ├ cache
  ├ config
  ├ controllers
  ├ models
  └ views
```

2.5 Start-up

Vaya a la carpeta recién creada **quick-start** e inicie el servidor php incorporado:

```
Ubiquity serve
```

Compruebe el correcto funcionamiento en la dirección **http://127.0.0.1:8090**:

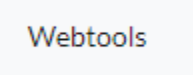


Nota: Si el puerto 8090 está ocupado, puede iniciar el servidor en otro puerto utilizando la opción -p.

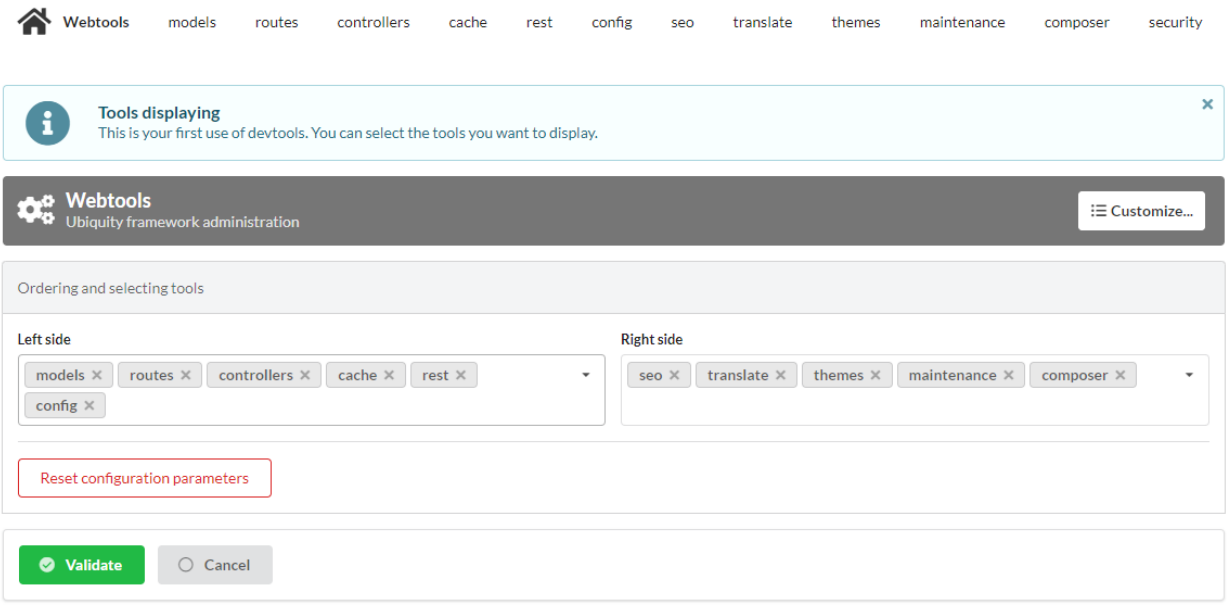
```
Ubiquity serve -p=8095
```

2.6 Controlador

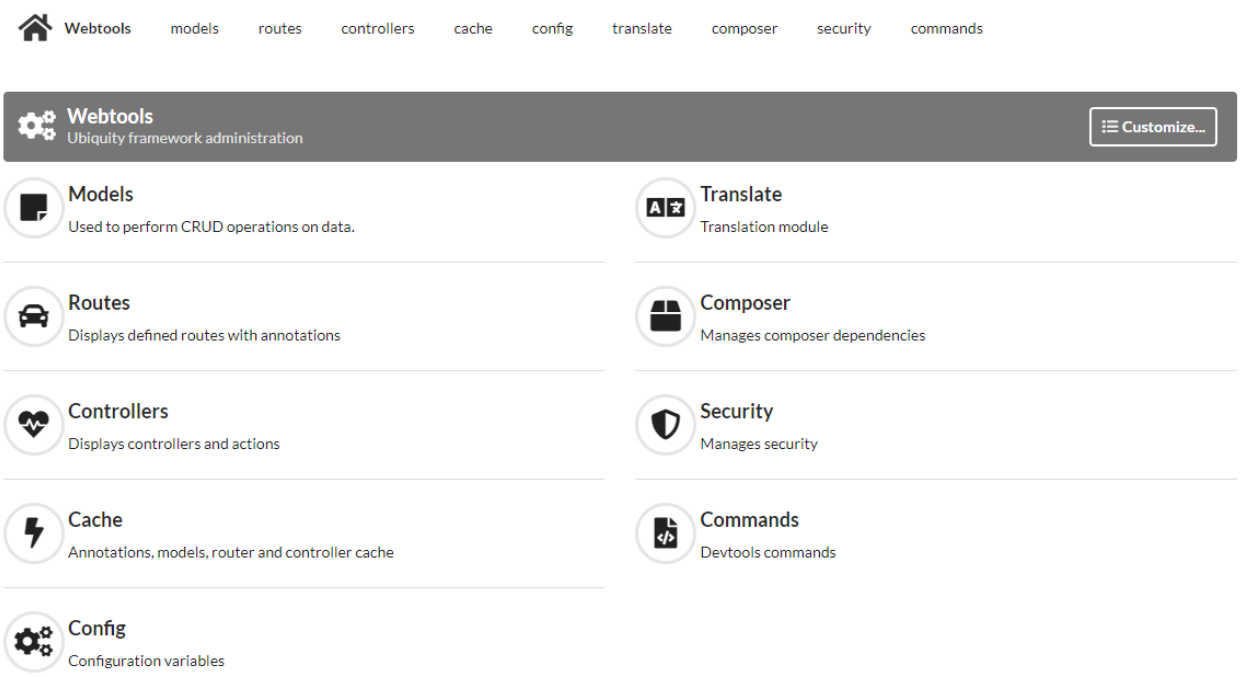
Vaya a la interfaz de administración haciendo clic en el botón **Webtools**:



Selecciona las herramientas que necesitas:



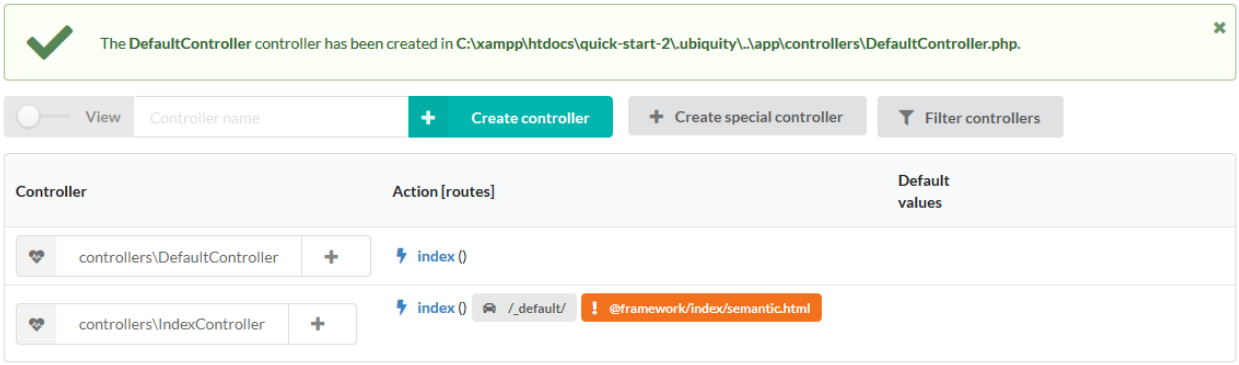
La aplicación web **Webtools** ahorra tiempo en operaciones repetitivas.



Vamos a través de él para crear un controlador.

Vaya a la parte **controllers**, introduzca **DefaultController** en el campo **nombreControlador** y cree el controlador:

Se crea el controlador **DefaultController**:



Podemos entonces editar el archivo `app/controllers/DefaultController` en nuestro IDE favorito:

Lista 1: app/controllers/DefaultController.php

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

Añada el mensaje tradicional y pruebe su página en `http://127.0.0.1:8090/DefaultController`.

Lista 2: app/controllers/DefaultController.php

```
class DefaultController extends ControllerBase{

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

}
```

Por ahora, no hemos definido rutas, por lo que el acceso a la aplicación se realiza según el siguiente esquema: `controllerName/actionName/param`

La acción por defecto es el método **index**, no necesitamos especificarlo en la url.

2.7 Route

Importante: El enrutamiento se define con la anotación `@route` y no se hace en un fichero de configuración: |br|es una elección de diseño.

El parámetro **automated** puesto a **true** permite definir los métodos de nuestra clase como sub-rutas de la ruta principal `/hello`.

Lista 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 }
```

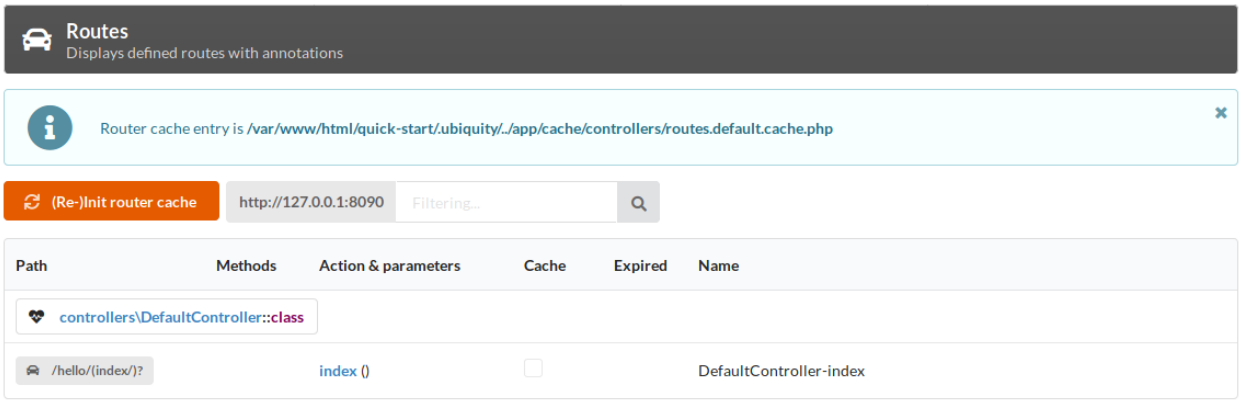
2.7.1 Router cache

Importante: Ningún cambio en las rutas es efectivo sin inicializar la caché. |br|Las anotaciones nunca se leen en tiempo de ejecución. Esto también es una elección de diseño.

Podemos utilizar las **web tools** para la reinicialización de la caché:

Vaya a la sección **Routes** y haga clic en el botón **re-init cache**.

La ruta aparece ahora en la interfaz:



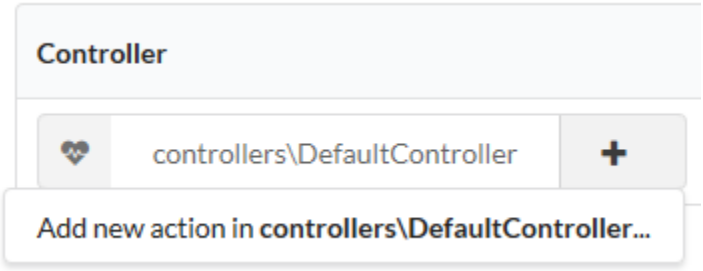
Ahora podemos probar la página pulsando el botón **GET** o yendo a la dirección `http://127.0.0.1:8090/hello`.

2.8 Acción y ruta con parámetros

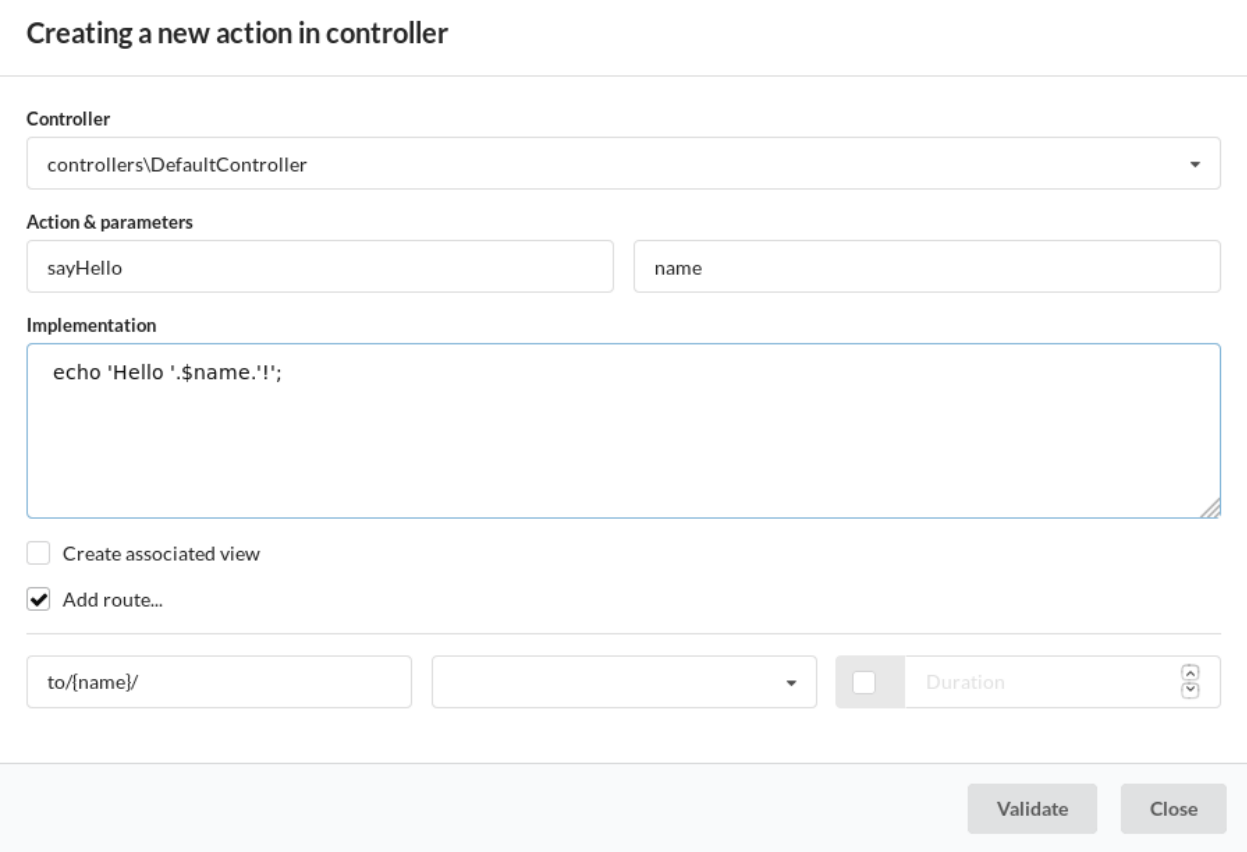
Ahora crearemos una acción (sayHello) con un parámetro (name), y la ruta asociada (to): La ruta utilizará el parámetro **name** de la acción:

Vaya a la sección **Controllers**:

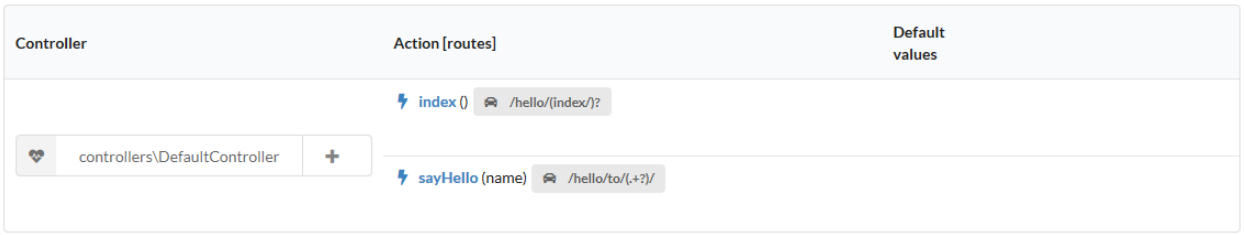
- haga clic en el botón + asociado a DefaultController,
- a continuación, seleccione **Add new action in..** elemento.



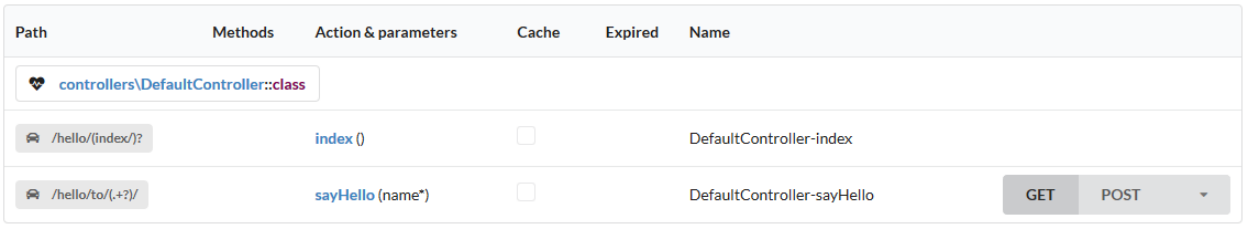
Introduzca la información de la acción en el siguiente formulario:



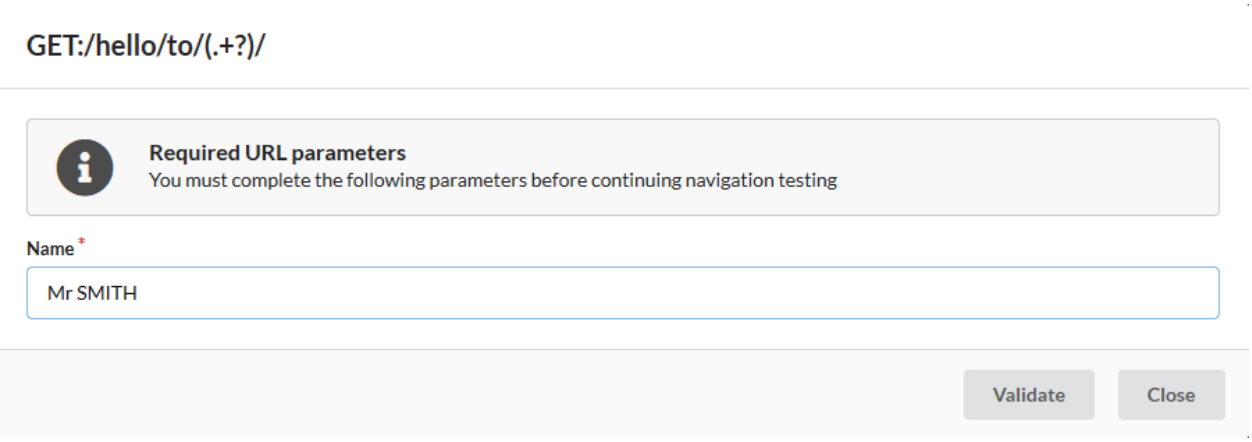
Tras reiniciar la caché con el botón naranja, podemos ver la nueva ruta **hello/to/{name}**:



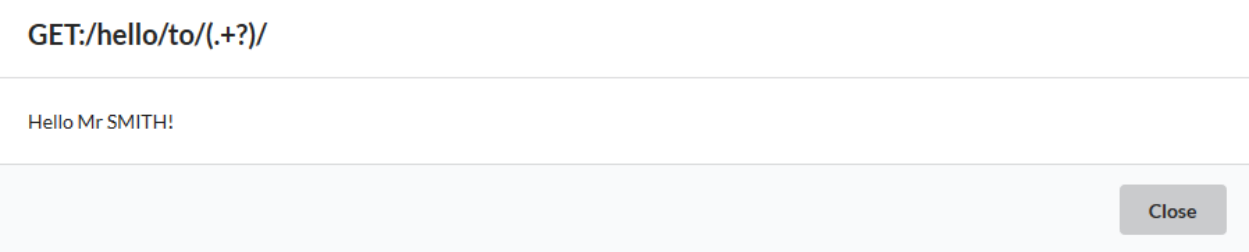
Compruebe la creación de la ruta accediendo a la sección Rutas:



Ahora podemos probar la página pulsando el botón **GET**:



Podemos ver el resultado:

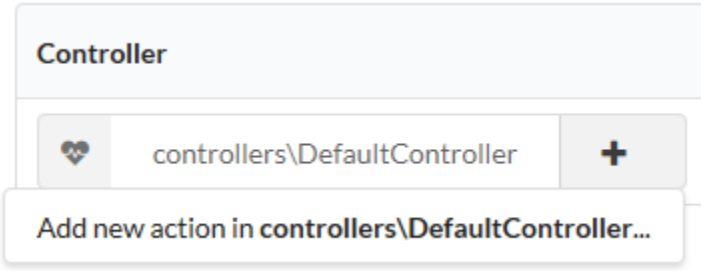


Podríamos ir directamente a la dirección `http://127.0.0.1:8090/hello/to/Mr SMITH` para probar

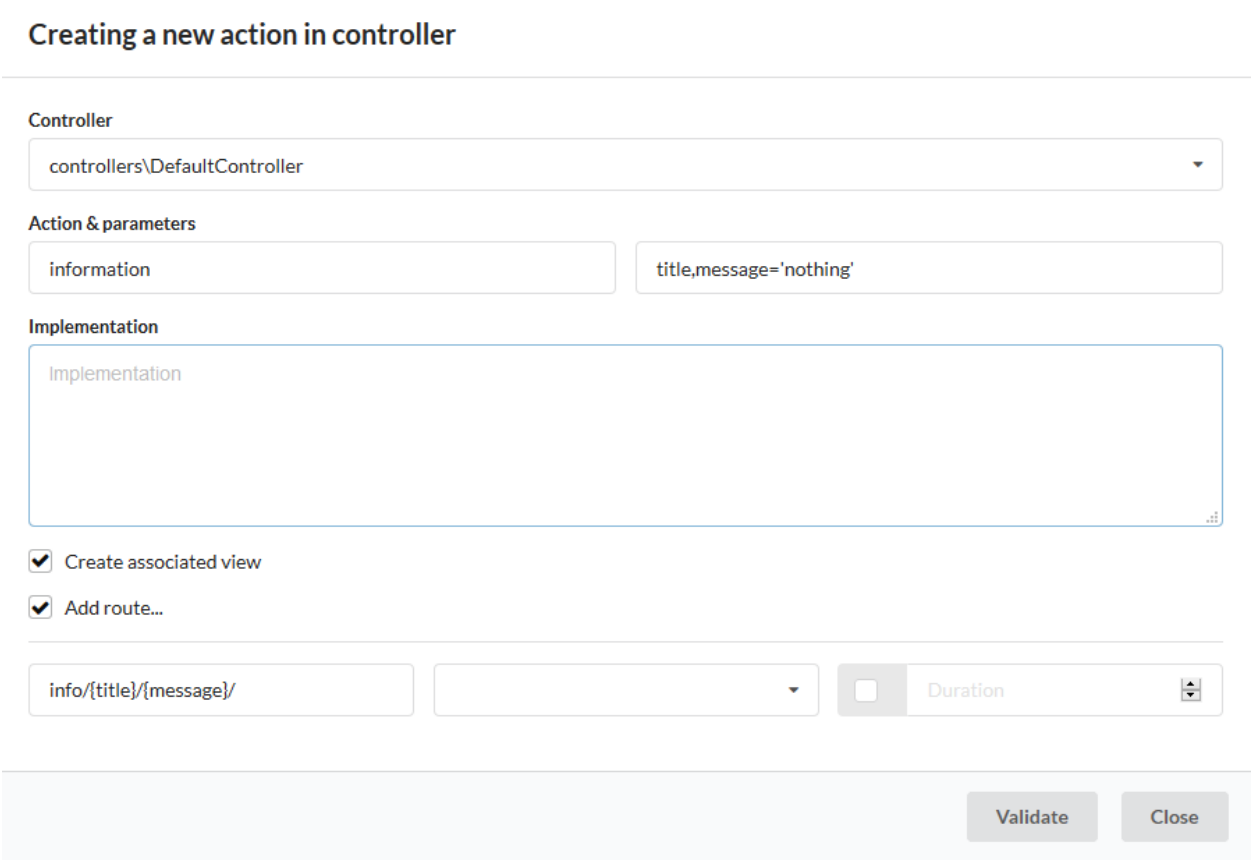
2.9 Acción, parámetros de ruta y vista

Ahora crearemos una acción (information) con dos parámetros (title and message), la ruta asociada (info) y una vista para mostrar el mensaje: |br|La ruta utilizará los dos parámetros de la acción.

En la sección **Controllers**, cree otra acción en **DefaultController**:

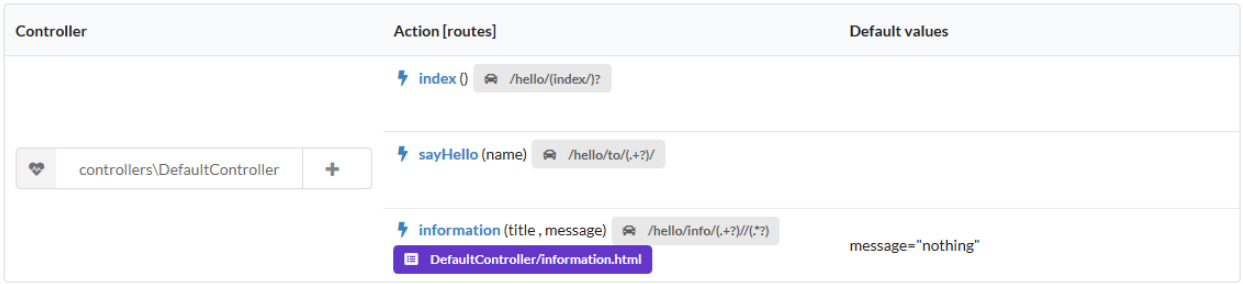


Introduzca la información de la acción en el siguiente formulario:



Nota: La casilla de verificación de la vista se utiliza para crear la vista asociada a la acción.

Tras reinicializar la caché, ahora tenemos 3 rutas:



Volvamos a nuestro entorno de desarrollo y veamos el código generado:

Lista 4: app/controllers/DefaultController.php

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

Necesitamos pasar las 2 variables a la vista:

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

Y utilizamos nuestras 2 variables en la vista twig asociada:

Lista 5: app/views/DefaultController/information.html

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

Podemos probar nuestra página en `http://127.0.0.1:8090/hello/info/Quick start/Ubiquity is quiet simple` Es obvio

New in documentation

- *Seguridad*
- *Plataformas asíncronas*
- *Módulo de comandos*
- *Módulo composer*
- *Módulo cliente OAuth*
- Módulo Mailer
- *Configuración de servidores*
- *Conexión a la base de datos*
- *Optimización*
- *Cliente rico*
- *Módulo REST*
- *Transformadores de datos*
- *Inyección de dependencia*
- *Eventos*
- *Vistas y temas*

- *Contribución*
- *Inicio rápido con webtools (UbiquityMyAdmin)*
- Generación de modelos:
  - con webtools (UbiquityMyAdmin)
  - con consola (devtools)

CAPÍTULO 3

Instalación de Ubiquity-devtools

3.1 Instalar Composer

ubiquity utiliza Composer para gestionar sus dependencias. Por lo tanto, antes de usar, usted tendrá que asegurarse de que tiene Composer instalado en su máquina.

3.2 Instalar Ubiquity-devtools

Descargue el instalador de Ubiquity-devtools utilizando Composer.

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

Asegúrese de colocar el directorio `~/.composer/vendor/bin` en su PATH para que el ejecutable **Ubiquity** pueda ser localizado por su sistema.

Una vez instalado, el simple comando `Ubiquity new` creará una nueva instalación de Ubiquity en el directorio que especifique. Por ejemplo, `Ubiquity new blog` crearía un directorio llamado **blog** que contendría un proyecto Ubiquity:

```
Ubiquity new blog
```

La opción semántica añade Semantic-UI para el front-end.

Puede ver más opciones sobre la instalación leyendo la sección *Creación de proyectos*.

CAPÍTULO 4

Creación de proyectos

Después de instalar *Instalación de Ubiquity-devtools*, en su terminal, llame al comando *new* en la carpeta raíz de su servidor web :

4.1 Muestras

Un proyecto sencillo

```
Ubiquity new projectName
```

Un proyecto con interfaz UbiquityMyAdmin

```
Ubiquity new projectName -a
```

Un proyecto con los temas bootstrap y semantic-ui instalados

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

4.2 Argumentos del instalador

nombre corto	nombre	rol	default	Valores permitidos	Desde devtools
b	dbName	Establece el nombre de la base de datos.			
s	serverName	Define la dirección del servidor db.	*127.0.0.1*		
p	port	Define el puerto del servidor db.	*3306*		
u	user	Define el usuario del servidor db.	*root*		
w	password	Defines the db server password.	*“”*		
h	themes	Instalar temas.		semantic,bootstrap,foundation	
m	allmodels	Crea todos los modelos a partir de la base de datos.	*false*		
a	admin	Añade la interfaz UbiquityMyAdmin.	*false*		
i	siteUrl	Define la URL del sitio.	*http://127.0.0.1/{projectname}*		1.2.6
e	rewriteBase	Establece la base para la reescritura.	*/{projectname}/*		1.2.6

4.3 Uso de argumentos

4.3.1 nombres cortos

Ejemplo de creación del proyecto **blog**, conectado a la base de datos **blogDb**, con generación de todos los modelos

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

4.3.2 nombres largos

Ejemplo de creación del proyecto **blog**, conectado a la base de datos **bogDb**, con generación de todos los modelos e integración del tema semántico

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

4.4 Ejecutar

Para iniciar el servidor web incrustado y probar tus páginas, ejecútalo desde la carpeta raíz de la aplicación:

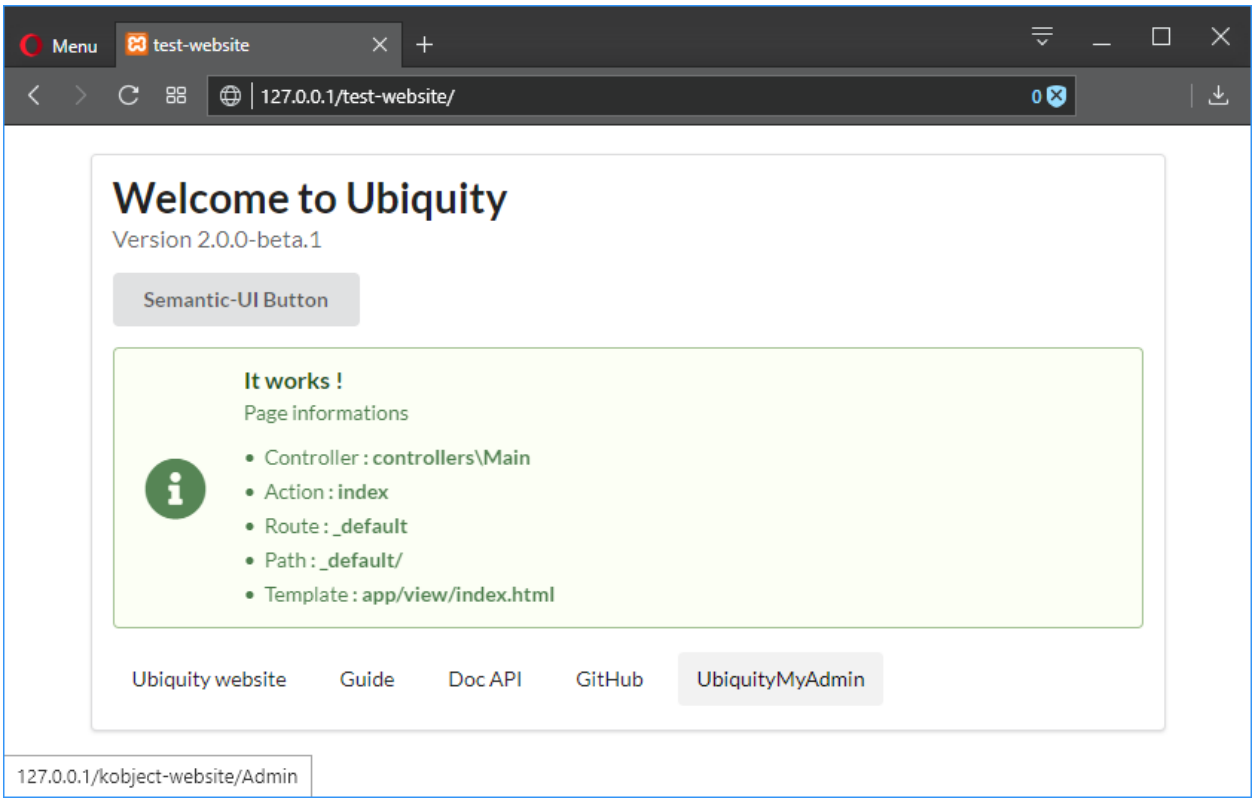
```
Ubiquity serve
```

El servidor web se inicia en `127.0.0.1:8090`.

CAPÍTULO 5

Configuración del proyecto

Normalmente, el instalador limita las modificaciones a realizar en los ficheros de configuración y su aplicación queda operativa tras la instalación



5.1 Configuración principal

La configuración principal de un proyecto se localiza en el archivo `app/conf/config.php`.

Lista 1: app/conf/config.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 Configuración de los servicios

Los servicios que se cargan al iniciarse se configuran en el archivo `app/conf/services.php`.

Lista 2: app/conf/services.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 bonitas

5.3.1 Apache

El framework incluye un archivo **.htaccess** que se utiliza para permitir URLs sin index.php. Si utiliza Apache para servir su aplicación Ubiquity, asegúrese de habilitar el módulo **mod_rewrite**.

Lista 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>
```

Véase Apache configuration para saber más.

5.3.2 Nginx

En Nginx, la siguiente directiva en la configuración de su sitio permitirá URLs «bonitas»:

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

Véase NginX configuration para saber más.

5.3.3 Laravel Valet Driver

Valet es un entorno de desarrollo php para Mac minimalistas. No `Vagrant`, no `/etc/hosts` file. Incluso puedes compartir tus sitios públicamente usando túneles locales.

Laravel Valet configura tu Mac para ejecutar siempre `Nginx` en segundo plano cuando se inicia tu máquina. Entonces, usando `DnsMasq`, Valet proxy todas las peticiones en el dominio `*.test` para que apunten a sitios instalados en tu máquina local.

Más información sobre Laravel Valet

Crea `UbiquityValetDriver.php` en `~/.config/valet/Drivers/` añade el siguiente código php y guárdalo.

```
<?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';
```

(continué en la próxima página)

(proviene de la página anterior)

```
                $_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;
                }
        }
}
```

CAPÍTULO 6

Uso de Devtools

6.1 Creación de proyectos

Consulte *Creación de proyectos* para crear un proyecto.

Truco: Para todos los demás comandos, debes estar en la carpeta de tu proyecto o en una de sus subcarpetas.

Importante: La carpeta `.ubiquity` creada automáticamente con el proyecto permite a las devtools encontrar la carpeta raíz del proyecto. Si se ha eliminado o ya no está presente, debe volver a crear esta carpeta vacía.

6.2 Creación de controladores

6.2.1 Especificaciones

- comando : `controller`
- Argumento : `controller-name`
- alias : `create-controller`

6.2.2 Parámetros

nombre corto	nombre	rol	default	Valores permitidos
v	view	Crea el índice de la vista asociada.	true	true, false

6.2.3 Muestras:

Crea la clase de controlador `controllers\ClientController` en `app/controllers/ClientController.php`:

```
Ubiquity controller ClientController
```

Crea la clase de controlador `controllersClientController` en `app/controllers/ClientController.php` y la vista asociada en `app/views/ClientController/index.html`:

```
Ubiquity controller ClientController -v
```

6.3 Creación de acciones

6.3.1 Especificaciones

- comando : `action`
- Argumento : `controller-name.action-name`
- aliases : `new-action`

6.3.2 Parámetros

nombre corto	nombre	rol	default	Valores permitidos
p	params	Los parámetros (o argumentos) de la acción.		a,b=5 or $a,$b,$c
r	route	La ruta asociada.		/path/to/route
v	create-view	Crea la vista asociada.	false	true,false

6.3.3 Muestras:

Añade la acción `all` en el controlador `Users`:

```
Ubiquity action Users.all
```

resultado del código:

Lista 1: app/controllers/Users.php

```
1 namespace controllers;
2 /**
3  * Controller Users
4  */
5 class Users extends ControllerBase{
```

(continué en la próxima página)

(proviene de la página anterior)

```
    public function index(){}

    public function all(){

    }

}
```

Añade la acción `display` en el controlador `Users` con un parámetro:

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

resultado del código:

Lista 2: app/controllers/Users.php

```
class Users extends ControllerBase{

    public function index(){}

    public function display($idUser){

    }
}
```

Añade la acción `display` con una ruta asociada:

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

resultado del código:

Atributos

Lista 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){

    }
}
```

Anotaciones

Lista 4: app/controllers/Users.php

```
namespace controllers;

class Users extends ControllerBase{

   public function index(){}

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

   }
}
```

Añade la acción buscar con múltiples parámetros:

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

resultado del código:

Atributos

Lista 5: 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){

   }

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

   }
}
```

Anotaciones

Lista 6: app/controllers/Users.php

```
namespace controllers;

class Users extends ControllerBase{

   public function index(){}

```

(continué en la próxima página)

(proviene de la página anterior)

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

    }

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

    }
}
```

Añade la acción `search` y crea la vista asociada:

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

6.4 Creación de modelos

Nota: Opcionalmente, compruebe la configuración de la conexión a la base de datos en el archivo app/config/config.php antes de ejecutar estos comandos.

Generar un modelo correspondiente a la tabla **user** de la base de datos:

```
Ubiquity model user
```

6.5 Creación de todos los modelos

Para generar todos los modelos a partir de la base de datos:

```
Ubiquity all-models
```

6.6 Inicialización de la caché

Para inicializar la caché de enrutamiento (basado en anotaciones en controladores) y orm (basado en anotaciones en modelos) :

```
Ubiquity init-cache
```

CAPÍTULO 7

URLs

como muchos otros frameworks, si estás usando el router con su comportamiento por defecto, hay una relación uno a uno entre una cadena URL y su correspondiente clase/método controlador. Los segmentos en una URI normalmente siguen este patrón:

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

7.1 Método por defecto

Cuando la URL se compone de una sola parte, correspondiente al nombre de un controlador, se llama automáticamente al método index del controlador :

URL :

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

Controller :

Lista 1: app/controllers/Products.php

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

7.2 Parámetros requeridos

Si el método solicitado requiere parámetros, deben pasarse en la URL:

Controller :

Lista 2: app/controllers/Products.php

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

Valid Urls :

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

7.3 Parámetros opcionales

El método llamado puede aceptar parámetros opcionales.

Si un parámetro no está presente en la URL, se utiliza el valor por defecto del parámetro.

Controller :

Lista 3: app/controllers/Products.php

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

Valid Urls :

```
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 Se distingue entre mayúsculas y minúsculas

En los sistemas Unix, el nombre de los controladores distingue entre mayúsculas y minúsculas.

Controller :

Lista 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 Personalización de rutas

El *Router (Rutas)* y las anotaciones/atributos en las clases de controlador permiten personalizar las URL.

CAPÍTULO 8

Router (Rutas)

El enrutamiento se puede utilizar además del mecanismo por defecto que asocia `controller/action/{parameters}` con una url.

8.1 Rutas dinámicas

Las rutas dinámicas se definen en tiempo de ejecución. Es posible definir estas rutas en el archivo **app/config/services.php**.

Importante: Las rutas dinámicas sólo deben utilizarse si la situación lo requiere:

- en el caso de una microaplicación
- si una ruta debe definirse dinámicamente

En todos los demás casos, es aconsejable declarar las rutas con anotaciones, para beneficiarse del almacenamiento en caché.

8.1.1 Rutas de devolución de llamadas (callback)

Las rutas más básicas de Ubiquity aceptan un Closure. En el contexto de las microaplicaciones, este método evita tener que crear un controlador.

Lista 1: app/config/services.php

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

Se pueden definir rutas de devolución de llamada para todos los métodos http con:

- Router::post
- Router::put
- Router::delete
- Router::patch
- Router::options

8.1.2 Rutas de controlador

Las rutas también pueden asociarse de forma más convencional a una acción de un controlador:

Lista 2: app/config/services.php

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

El método `FooController::index()` será accesible a través de la url `/bar`.

En este caso, el **FooController** debe ser una clase que herede de **Ubiquitycontrollers\Controller** o una de sus subclases, y debe tener un método **index**:

Lista 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 Ruta por defecto

La ruta por defecto coincide con la ruta **/**. Puede definirse utilizando la ruta reservada **_default**.

Lista 4: app/config/services.php

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

8.2 Rutas estáticas

Las rutas estáticas se definen mediante anotaciones o con atributos nativos de php desde `Ubiquity 2.4.0`.

Nota: Estas anotaciones o atributos nunca se leen en tiempo de ejecución. Es necesario reiniciar la caché del enrutador para tener en cuenta los cambios realizados en las rutas.

8.2.1 Creación

Atributos

Lista 5: 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')]
8      public function index(){}
9
10 }
```

Anotaciones

Lista 6: app/controllers/ProductsController.php

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

El método `Products::index()` será accesible a través de la url `/products`.

Nota:

La barra inicial o terminal se ignora en la ruta. Por lo tanto, las siguientes rutas son equivalentes:

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

8.2.2 Parámetros de ruta

Una ruta puede tener parámetros:

Atributos

Lista 7: app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

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'
        // ...
    }
}
```

Anotaciones

Lista 8: app/controllers/ProductsController.php

```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 Parámetros opcionales de ruta

Una ruta puede definir parámetros opcionales, si el método asociado tiene argumentos opcionales:

Atributos

Lista 9: app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

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

Anotaciones

Lista 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 Requisitos de ruta

Es posible añadir especificaciones sobre las variables pasadas en la url mediante el atributo **requirements**.

Atributos

Lista 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){
        // ...
    }
}
```

Anotaciones

Lista 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 ruta definida coincide con estas urls:

- `products/all/1/20`
- `products/all/5/`

pero no con ese:

- `products/all/test`

8.2.5 Tipificación de parámetros

La declaración de ruta tiene en cuenta los tipos de datos pasados a la acción, lo que evita añadir requisitos para tipos simples (int, bool, float).

Atributos

Lista 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){
        // ...
    }
}
```

Anotaciones

Lista 14: app/controllers/ProductsController.php

```php
namespace controllers;

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

La ruta definida coincide con estas urls:

- `products/1`
- `products/20`

pero no con ese:

- `products/test`

Valores correctos por tipo de datos:

- `int`: 1...
- `bool`: `0` or 1
- `float`: 1 `1.0` ...

8.2.6 Métodos http de enrutado

Es posible especificar el método o métodos http asociados a una ruta:

Atributos

Lista 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(){}

}
```

Anotaciones

Lista 16: app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{

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

}
```

El atributo **methods** puede aceptar varios métodos: `@route("testMethods", "methods"=>["get", "post", "delete"])` `#[Route('testMethods', methods: ['get','post','delete'])]`

La anotación **@route** o atributo **Route** se aplica por defecto a todos los métodos HTTP. Existe una anotación específica para cada uno de los métodos HTTP existentes:

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

Atributos

Lista 17: app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Get;

class ProductsController extends ControllerBase{

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

}
```

Anotaciones

Lista 18: app/controllers/ProductsController.php

```php
namespace controllers;

class ProductsController extends ControllerBase{

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

}
```

8.2.7 Nombre de ruta

Es posible especificar el **name** de una ruta, este nombre facilita entonces el acceso a la url asociada. Si no se especifica el atributo **name**, cada ruta tiene un nombre por defecto, basado en el patrón **controllerName_methodName**.

Atributos

Lista 19: app/controllers/ProductsController.php

```php
namespace controllers;

use Ubiquity\attributes\items\router\Route;

class ProductsController extends ControllerBase{

    #[Route('products',name: 'products.index')]
    public function index(){}

}
```

Anotaciones

Lista 20: app/controllers/ProductsController.php

```php
namespace controllers;

```

(continué en la próxima página)

(proviene de la página anterior)

```
class ProductsController extends ControllerBase{

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

}
```

8.2.8 Generación de URL o rutas

Los nombres de ruta pueden utilizarse para generar URL o rutas.

Enlaces a páginas en Twig

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

8.2.9 Ruta global

La anotación **@route** puede utilizarse en una clase de controlador :

Atributos

Lista 21: app/controllers/ProductsController.php

```
namespace controllers;

use Ubiquity\attributes\items\router\Route;

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

}
```

Anotaciones

Lista 22: app/controllers/ProductsController.php

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

    ...
    /**
     * @route("/all")
     */
```

(continué en la próxima página)

(proviene de la página anterior)

```
    public function display(){}

}
```

En este caso, la ruta definida en el controlador se utiliza como prefijo para todas las rutas del controlador : La ruta generada para la acción **display** es `/product/all`.

rutas automatizadas

Si se define una ruta global, es posible añadir todas las acciones del controlador como rutas (utilizando el prefijo global), estableciendo el parámetro **automated** :

Atributos

Lista 23: app/controllers/ProductsController.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){}

}
```

Anotaciones

Lista 24: app/controllers/ProductsController.php

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

    public function index(){}

    public function generate(){}

    public function display($id){}

}
```

El atributo automated define las 3 rutas contenidas en ProductsController:

- */product/(index/)?*
- */product/generate*

- */product/display/{id}*

rutas heredadas

Con el atributo **inherited**, también es posible generar las rutas declaradas en las clases base, o generar rutas asociadas a acciones de clases base si el atributo **automated** se establece a true al mismo tiempo.

La clase base:

Atributos

Lista 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){}

}
```

Anotaciones

Lista 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 clase derivada que utiliza miembros heredados:

Atributos

Lista 27: app/controllers/ProductsController.php

```php
namespace controllers;

```

(continué en la próxima página)

(proviene de la página anterior)

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

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

    public function display(){}

}
```

Anotaciones

Lista 28: app/controllers/ProductsController.php

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

    public function display(){}

}
```

El atributo inherited define las 2 rutas definidas en ProductsBase:

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

Si se combinan los atributos **automated** y **inherited**, las acciones de la clase base también se añaden a las rutas.

Parámetros de ruta globales

La parte global de una ruta puede definir parámetros, que serán pasados en todas las rutas generadas. Estos parámetros se pueden recuperar a través de un miembro de datos público:

Atributos

Lista 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;
    }

}
```

Anotaciones

Lista 30: app/controllers/FooController.php

```php
namespace controllers;

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

   public string $bar;

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

}
```

Accediendo a la url `/foo/bar/display` se muestra el contenido del miembro bar.

Ruta sin prefijo global

Si la ruta global se define en un controlador, todas las rutas generadas en este controlador irán precedidas del prefijo. Es posible introducir explícitamente excepciones en algunas rutas, utilizando el prefijo `#/`.

Atributos

Lista 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(){}

}
```

Anotaciones

Lista 32: app/controllers/FooController.php

```php
namespace controllers;

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

  /**
```

(continué en la próxima página)

(proviene de la página anterior)

```
    * @route("#/noRoot")
    */
    public function noRoot(){}

}
```

El controlador define la url `/noRoot`, que no está prefijada con la parte `/foo`.

8.2.10 Prioridad de ruta

El parámetro **prority** de una ruta permite resolver esta ruta en un orden de prioridad.

Cuanto mayor sea el parámetro de prioridad, más se definirá la ruta al principio de la pila de rutas en la caché.

En el ejemplo siguiente, la ruta **products/all** se definirá antes que la ruta **products**.

Atributos

Lista 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(){}

}
```

Anotaciones

Lista 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(){}

}
```

El valor de prioridad por defecto es `0`.

8.3 Caché de respuesta de rutas

Es posible almacenar en caché la respuesta producida por una ruta:

En este caso, la respuesta se almacena en caché y deja de ser dinámica.

Atributos

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

Anotaciones

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

8.3.1 Duración de la caché

La **duración** se expresa en segundos, si se omite, la duración de la caché es infinita.

Atributos

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

Anotaciones

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

8.3.2 Expiración de la caché

Es posible forzar la recarga de la respuesta borrando la caché asociada.

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

8.4 Cacheo dinámico de rutas

Las rutas dinámicas también pueden almacenarse en caché.

Importante: Esta posibilidad sólo es útil si esta caché no se realiza en producción, sino en el momento de inicializar la caché.

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

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

Comprobación de rutas con 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 Gestión de errores (errores 404 y 500)

8.5.1 Sistema de enrutamiento por defecto

Con el sistema de enrutamiento por defecto (la pareja controlador+acción definiendo una ruta), el manejador de errores puede ser redefinido para personalizar la gestión de errores.

En el fichero de configuración **app/config/config.php**, añade la clave **onError**, asociada a un devolucion de llamada (callback) que defina los mensajes de error:

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

Implementar la acción solicitada **p404** en el **IndexController**:

Lista 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 Enrutado con anotaciones

Basta en este caso con añadir una última ruta deshabilitando el sistema de enrutamiento por defecto, y correspondiente a la gestión del error 404:

Atributos

Lista 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>";
}
```

Anotaciones

Lista 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>";
}
```

CAPÍTULO 9

Controladores

Un controlador es una clase PHP que hereda de `Ubiquity\controllers\Controller`, proporcionando un punto de entrada en la aplicación. Los controladores y sus métodos definen URLs accesibles.

9.1 Creación de controladores

La forma más sencilla de crear un controlador es hacerlo desde las devtools.

Desde el símbolo del sistema, vaya a la carpeta del proyecto. Para crear el controlador Products, utilice el comando:

```
Ubiquity controller Products
```

El controlador `Products.php` se crea en la carpeta `app/controllers` del proyecto.

Lista 1: app/controllers/Products.php

```php
namespace controllers;
/**
 * Controller Products
 */
class Products extends ControllerBase{

public function index(){}

}
```

Ahora es posible acceder a URLs (por defecto se solicita el método `index`):

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

Nota: Un controlador puede crearse manualmente. En este caso, debe respetar las siguientes reglas:

- La clase debe estar en la carpeta **app/controllers**.
- El nombre de la clase debe coincidir con el nombre del archivo php
- La clase debe heredar de **ControllerBase** y estar definida en el espacio de nombres **controllers**.
- y debe anular el método abstracto **index**.

9.2 Métodos

9.2.1 public

El segundo segmento del URI determina a qué método público del controlador se llama. El método «index» siempre se carga por defecto si el segundo segmento del URI está vacío.

Lista 2: app/controllers/First.php

```
namespace controllers;
class First extends ControllerBase{

   public function hello(){
      echo "Hello world!";
   }


}
```

El método `hello` del controlador `First` pone a disposición la siguiente URL:

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

9.2.2 argumentos del método

los argumentos de un método deben pasarse en la url, excepto si son opcionales.

Lista 3: app/controllers/First.php

```
namespace controllers;
class First extends ControllerBase{

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


}
```

El método `hello` del controlador `First` pone a disposición la siguiente URL:

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

9.2.3 private

Los métodos privados o protegidos no son accesibles desde la URL.

9.3 Controlador por defecto

El controlador por defecto se puede establecer con el Router, en el archivo `services.php`.

Lista 4: app/config/services.php

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

En este caso, el acceso a la URL `example.com/` carga el controlador **First** y llama al método **index** por defecto.

9.4 carga de vistas

9.4.1 cargando

Las vistas se almacenan en la carpeta `app/views`. Se cargan desde los métodos del controlador. Por defecto, es posible crear vistas en php, o con twig. Twig es el motor de plantillas por defecto para archivos html.

carga de vistas php

Si no se especifica la extensión del archivo, el método **loadView** carga un archivo php.

Lista 5: app/controllers/First.php

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

carga de la vista twig

Si la extensión del archivo es html, el método **loadView** carga un archivo html twig.

Lista 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");
   }
}
```

Carga de la vista por defecto

Si se utiliza el método de nomenclatura de vistas por defecto : La vista por defecto asociada a una acción en un controlador se encuentra en la carpeta `views/nombre-controlador/nombre-acción`:

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

Lista 7: app/controllers/Users.php

```php
namespace controllers;

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

9.4.2 ver parámetros

Una de las misiones del controlador es pasar variables a la vista. Esto se puede hacer en la carga de la vista, con un array asociativo:

Lista 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]);
   }
}
```

Las claves del array asociativo crean variables del mismo nombre en la vista. Uso de estas variables en Twig:

Lista 9: app/views/index.html

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

También se pueden pasar variables antes de cargar la vista:

```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 ver resultado como cadena

Es posible cargar una vista, y devolver el resultado en una cadena, asignando true al 3er parámetro del método loadview :

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

9.4.4 carga de vistas múltiples

Un controlador puede cargar varias vistas:

Lista 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');
   }
}
```

Importante: Una vista suele ser parcial. Por lo tanto, es importante no integrar sistemáticamente las etiquetas **html** y **body** definiendo una página html completa.

9.4.5 organización de vistas

Es aconsejable organizar las vistas en carpetas. El método más recomendado es crear una carpeta por controlador, y almacenar allí las vistas asociadas. Para cargar la vista `index.html`, almacenada en `app/views/First`:

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

9.5 initialize y finalize

El método **initialize** se llama automáticamente antes de cada acción solicitada, el método **finalize** después de cada acción.

Ejemplo de uso de los métodos initialize y finalize con la clase base creada automáticamente con un nuevo proyecto:

Lista 11: app/controllers/ControllerBase.php

```
namespace controllers;

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

/**
```

(continué en la próxima página)

(proviene de la página anterior)

```
 * 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 Control de acceso

El control de acceso a un controlador puede realizarse manualmente, utilizando los métodos *isValid* y *onInvalidControl*.

El método *isValid* debe devolver un booleano que determine si es posible acceder a *action* pasada como parámetro:

En el siguiente ejemplo, el acceso a las acciones del controlador **IndexController** sólo es posible si existe una variable de sesión **activeUser**:

Lista 12: app/controllers/IndexController.php

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

Si la variable **activeUser** no existe, se devuelve un error **unauthorized 401**.

El método *onInvalidControl* permite personalizar el acceso no autorizado:

Lista 13: app/controllers/IndexController.php

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

   public function onInvalidControl(){
      $this->initialize();
      $this->loadView('unauthorized.html');
```

(continué en la próxima página)

(proviene de la página anterior)

```
        $this->finalize();
    }
}
```

Lista 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>
```

También es posible generar automáticamente el control de acceso a partir de *AuthControllers*

9.7 Reenvío

Una redirección no es una simple llamada a una acción de un controlador. La redirección implica los métodos *initialize* y *finalize*, así como el control de acceso.

El método **forward** puede invocarse sin utilizar los métodos *initialize* y *finalize*:

Es posible redirigir a una ruta por su nombre:

9.8 Inyección de dependencia

Ver *Dependency injection*

9.9 namespaces

El namespace del controlador se define por defecto en *controllers* en el archivo *app/config/config.php*.

9.10 Superclase

La herencia se puede utilizar para factorizar el comportamiento del controlador. La clase *BaseController* creada con un nuevo proyecto está presente para este propósito.

9.11 Clases base específicas del controlador

Clase de controlador	role
Controller	Clase base para todos los controladores
SimpleViewController	Clase base asociada a un motor de plantillas php (para usar con microservicios)
SimpleViewAsyncController	Clase base asociada a un motor de plantillas php para servidores asíncronos

CAPÍTULO 10

Eventos

Nota: El módulo Eventos utiliza la clase estática **EventsManager** para gestionar los eventos.

10.1 Eventos principales del marco

Ubiquity emite eventos durante las diferentes fases del envío de una solicitud. Estos eventos son relativamente pocos, para limitar su impacto en el rendimiento.

Parte	Nombre del evento	Parámetros	Ocurren cuando
ViewEvents	BEFORE_RENDER	viewname, parameters	Antes de renderizar una vista
ViewEvents	AFTER_RENDER	viewname, parameters	Después de renderizar una vista
DAOEvents	GET_ALL	objects, classname	Después de cargar varios objetos
DAOEvents	GET_ONE	object, classname	Después de cargar un objeto
DAOEvents	BEFORE_UPDATE	instance	Antes de actualizar un objeto
DAOEvents	AFTER_UPDATE	instance, result	Después de actualizar un objeto
DAOEvents	BEFORE_INSERT	instance	Antes de insertar un objeto
DAOEvents	AFTER_INSERT	instance, result	Después de insertar un objeto
RestEvents	BEFORE_INSERT	instance	Antes de insertar un objeto
RestEvents	BEFORE_UPDATE	instance	Antes de actualizar un objeto

Nota: No hay eventos **BeforeAction** y **AfterAction**, ya que los métodos **initialize** y **finalize** de la clase del controlador realizan esta operación.

10.2 Escuchar un evento

Example 1 :

Añadir una propiedad **_updated** en las instancias modificadas en la base de datos :

Lista 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;
            }
    });
```

Nota: Los parámetros que se pasan a la función callback varían según el evento que se esté escuchando.

Example 2 :

Modificación de la representación de la vista

Lista 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 Crear tus propios eventos

Example :

Crear un evento para contar y almacenar el número de visualizaciones por acción :

Lista 3: app/eventListener/TracePageEventListener.php

```
    namespace eventListener;

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

    class TracePageEventListener implements EventListenerInterface {
```

(continué en la próxima página)

(proviene de la página anterior)

```
        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 Registro de eventos

Registrando el evento **TracePageEventListener** en `services.php` :

Lista 4: app/config/services.php

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

    ...

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

10.5 Disparo de eventos

Un evento puede ser disparado desde cualquier lugar, pero tiene más sentido hacerlo aquí en el método **initialize** del controlador base :

Lista 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.
     **/
```

(continué en la próxima página)

(proviene de la página anterior)

```
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      }
```

El resultado en app/config/stats.php :

Lista 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 Eventos registro optimización

Es preferible almacenar en caché el registro de los listeners, para optimizar su tiempo de carga :

Crear un script de cliente, o una acción de controlador (no accesible en modo de producción) :

```
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();
}
```

Después de la ejecución, el archivo de caché se genera en `app/cache/events/events.cache.php`.

Una vez creada la caché, el archivo `services.php` sólo necesita tener la línea :

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

CAPÍTULO 11

Inyección de dependencia

Nota: Por razones de rendimiento, la inyección de dependencias no se utiliza en la parte central del framework.

La inyección de dependencia (DI) es un patrón de diseño utilizado para implementar IoC. Permite la creación de objetos dependientes fuera de una clase y proporciona esos objetos a una clase a través de diferentes formas. Usando DI, movemos la creación y vinculación de los objetos dependientes fuera de la clase que depende de ella.

Nota: Ubiquity sólo soporta inyección de propiedades, para no requerir introspección en la ejecución. Sólo los controladores soportan inyección de dependencias.

11.1 Servicio de autowiring

11.1.1 Creación de servicios

Crear un servicio

Lista 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";
```

(continué en la próxima página)

(proviene de la página anterior)

```
        }
    }
```

11.1.2 Autowiring en el controlador

Crear un controlador que requiere el servicio

Lista 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;
        }
}
```

En el ejemplo anterior, Ubiquity busca e inyecta **$service** cuando se crea **ClientController**.

La anotación @autowired requiere que:

- el tipo que se va a instanciar se declara con la anotación **@var**.
- La propiedad **$service** tiene un setter, o si se declara pública

Como las anotaciones nunca se leen en tiempo de ejecución, es necesario generar la caché de los controladores:

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

Queda por comprobar que el servicio se inyecta yendo a la dirección `/ClientController`.

11.2 Inyección de servicio

11.2.1 Servicio

Creemos ahora un segundo servicio, que requiere una inicialización especial.

Lista 3: app/services/ServiceWithInit.php

```
class ServiceWithInit{
        private $init;

        public function init(){
                $this->init=true;
        }

        public function do(){
                if($this->init){
                        echo 'init well initialized!';
                }else{
                        echo 'Service not initialized';
                }
        }
}
```

11.2.2 Inyección en el controlador

Lista 4: app/controllers/ClientController.php

```
namespace controllers;

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

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

        /**
         * @injected
         */
        private $serviceToInit;

        public function index(){
                $this->serviceToInit->do();
        }

        /**
```

(continué en la próxima página)

(proviene de la página anterior)

```
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 Declaración Di

En `app/config/config.php`, crea una nueva clave para la propiedad **serviceToInit** para inyectar en la parte **di**.

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

generar la caché de los controladores:

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

Comprueba que el servicio está inyectado yendo a la dirección `/ClientController`.

Nota: Si se va a utilizar el mismo servicio en varios controladores, utilice la notación comodín :

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

11.2.4 Inyección con un nombre calificador

Si el nombre del servicio que se va a inyectar es diferente de la clave de la matriz **di**, es posible utilizar el atributo name de la anotación **@injected**.

En `app/config/config.php`, crea una nueva clave para la propiedad **serviceToInit** para inyectar en la parte **di**.

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

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

11.3 Inyección de servicios en tiempo de ejecución

Es posible inyectar servicios en tiempo de ejecución, sin que éstos hayan sido declarados previamente en las clases del controlador.

Lista 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     }
```

En `app/config/config.php`, crea la clave **@exec** en la parte **di**.

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

Con esta declaración, el miembro **$rService**, instancia de **RuntimeService**, se inyecta en todos los controladores. Es aconsejable utilizar los comentarios javadoc para declarar **$rService** en los controladores que lo utilizan (para obtener la finalización de código en **$rService** en su IDE).

Lista 6: app/controllers/MyController.php

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

(continué en la próxima página)

(proviene de la página anterior)

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

CAPÍTULO 12

Controladores CRUD

Los controladores CRUD permiten realizar operaciones básicas sobre una clase Modelo:

- Create
- Read
- Update
- Delete
- ...

Nota:

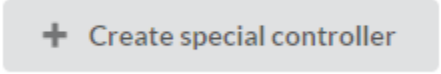
Desde la versión 2.4.6, existen dos tipos de CrudController:

- *ResourceCrudController* asociado a un modelo
- *MultiResourceCRUDController*, mostrando un índice y permitiendo navegar entre modelos.

12.1 ResourceCrudController

12.1.1 Creación

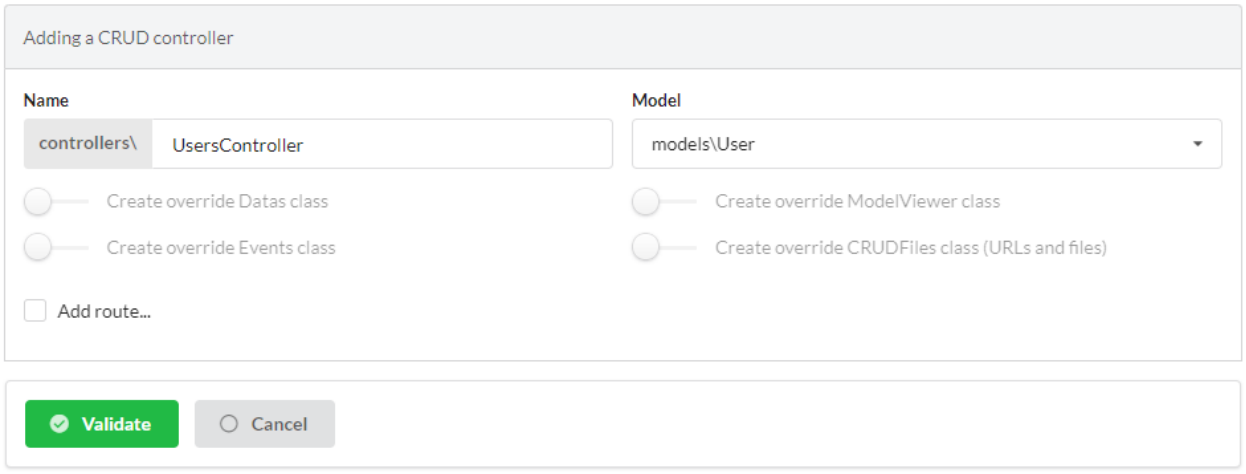
En la interfaz de administración (web-tools), active la parte **Controllers**, y elija crear **Resource Crud controller**:



A continuación, rellene el formulario:

- Introduzca el nombre del controlador

- Seleccione el modelo asociado
- A continuación, haga clic en el botón de validación



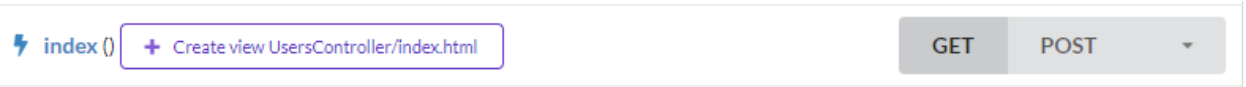
12.1.2 Descripción de las características

El controlador generado:

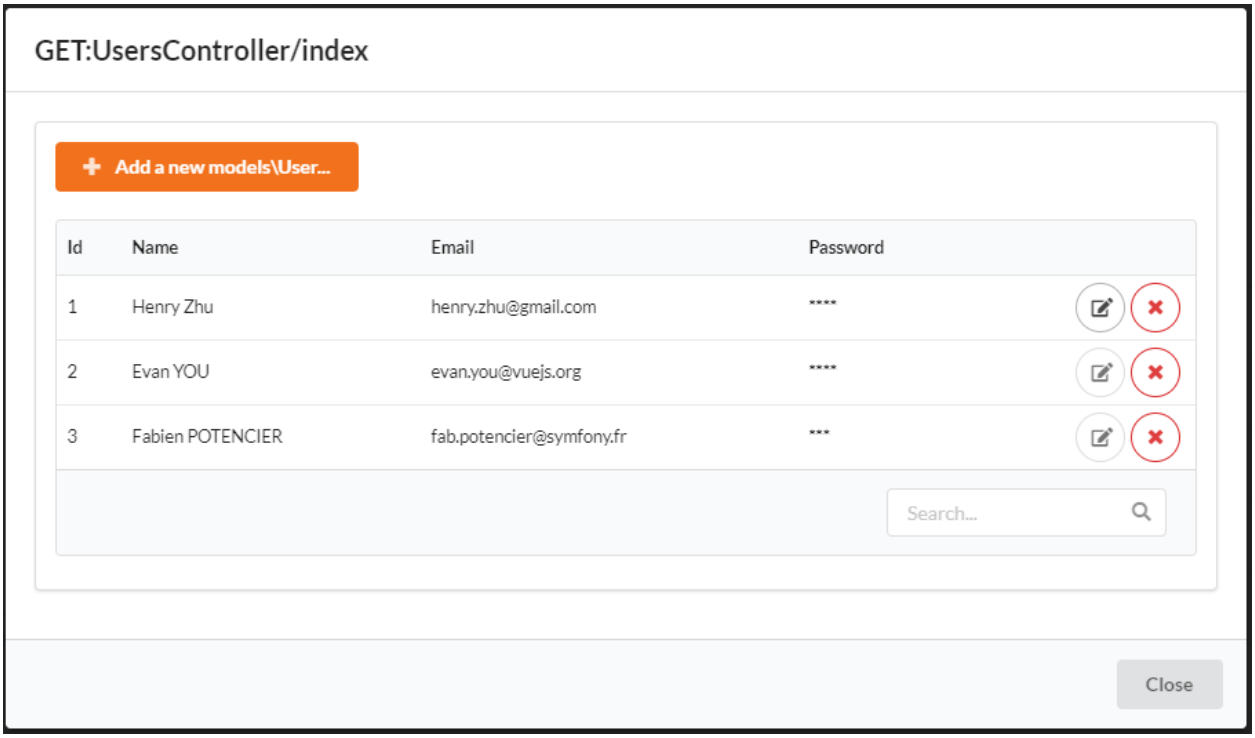
Lista 1: app/controllers/Products.php

```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';
	}
}
```

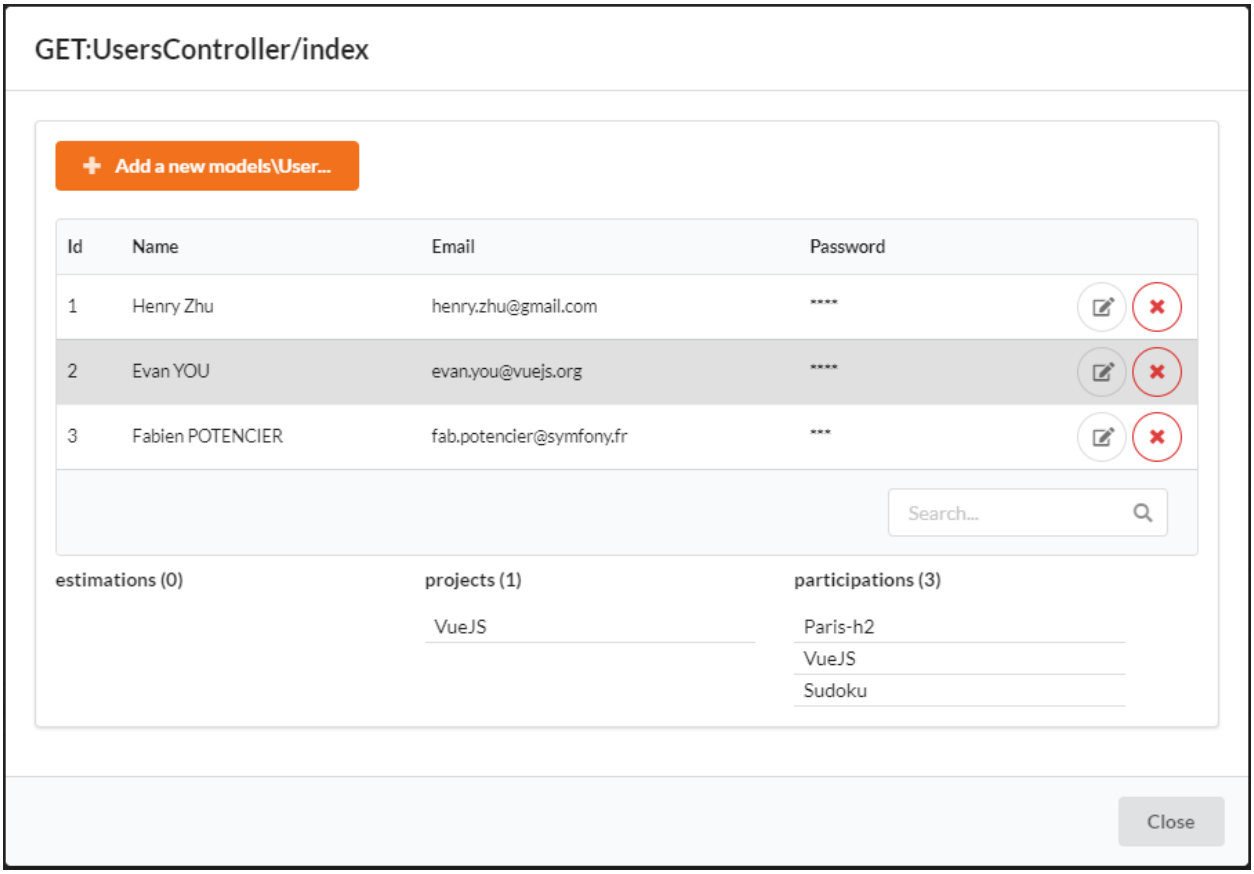
Pruebe el controlador creado haciendo clic en el botón get situado delante de la acción **index**:



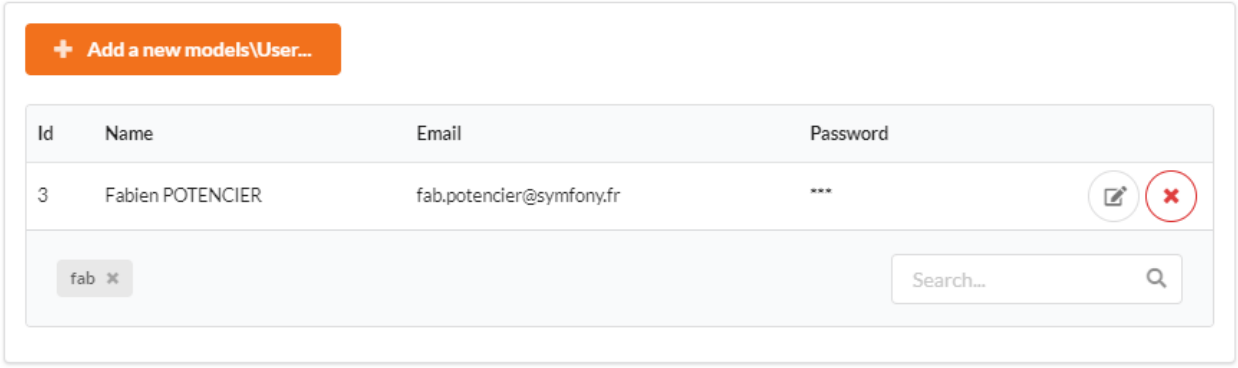
Read (acción índice)



Al hacer clic en una fila de la dataTable (instance) se muestran los objetos asociados a la instancia (acción **details**):



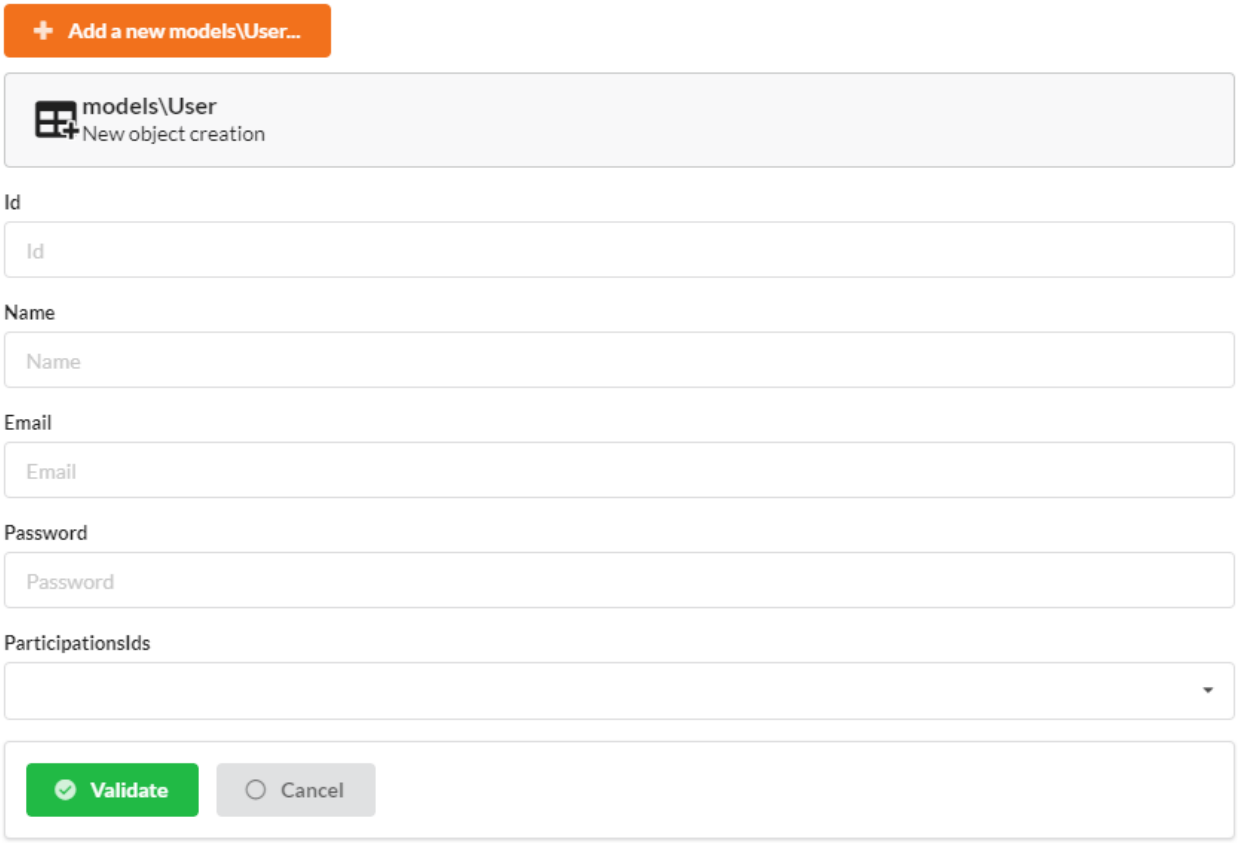
Utilizar el área de búsqueda:



Create (acción newModel)

Es posible crear una instancia pulsando el botón añadir

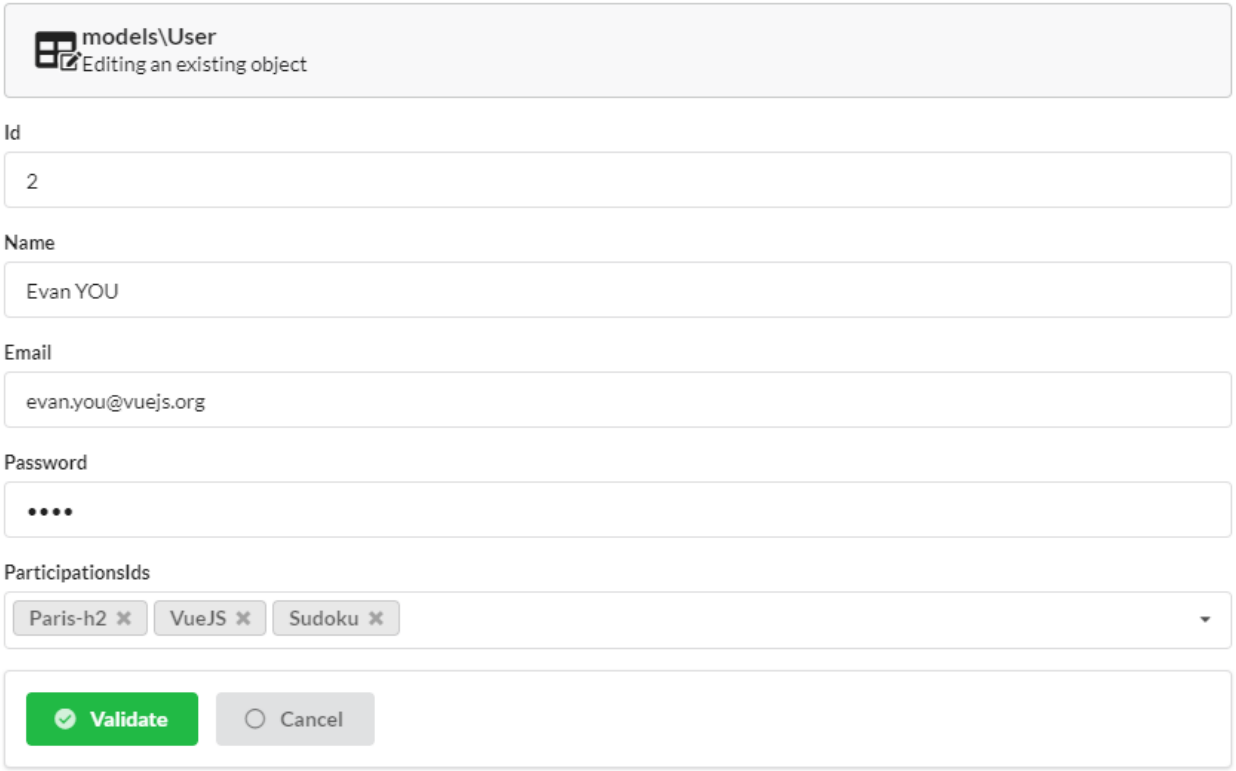
El formulario por defecto para añadir una instancia de Usuario:



Update (acción de actualización)

El botón de edición de cada fila permite editar una instancia

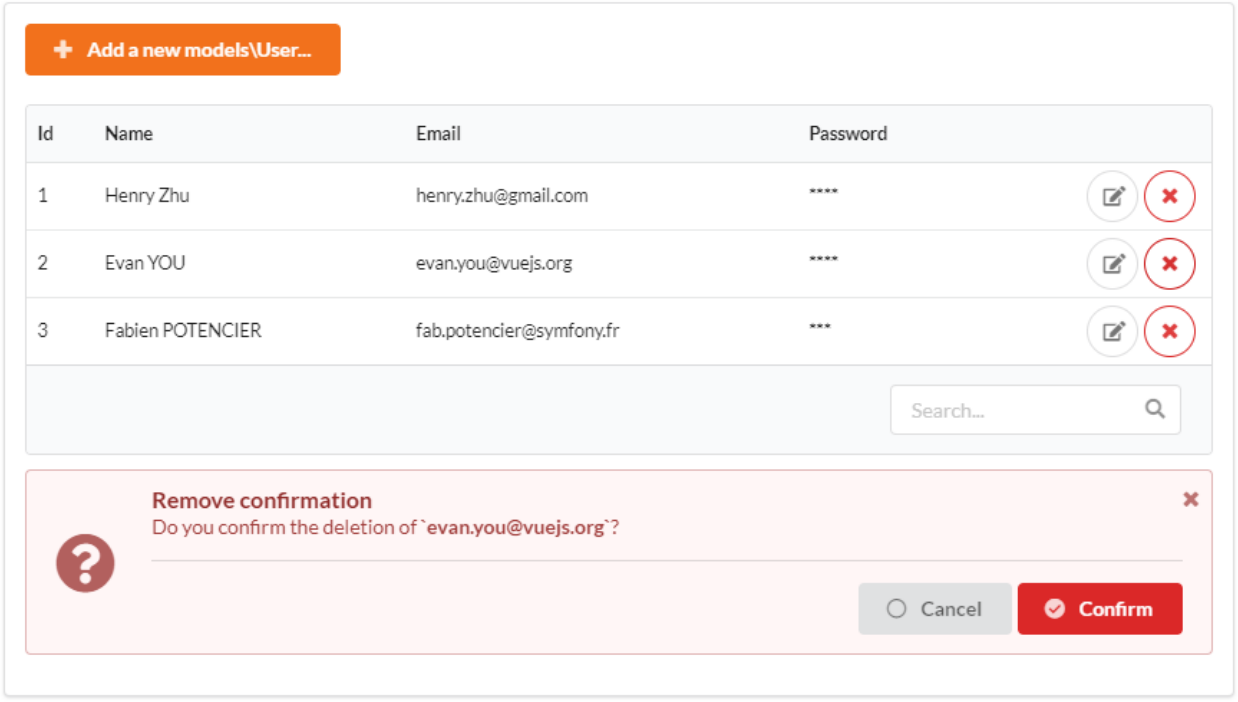
El formulario por defecto para añadir una instancia de Usuario:



Delete (acción eliminar)

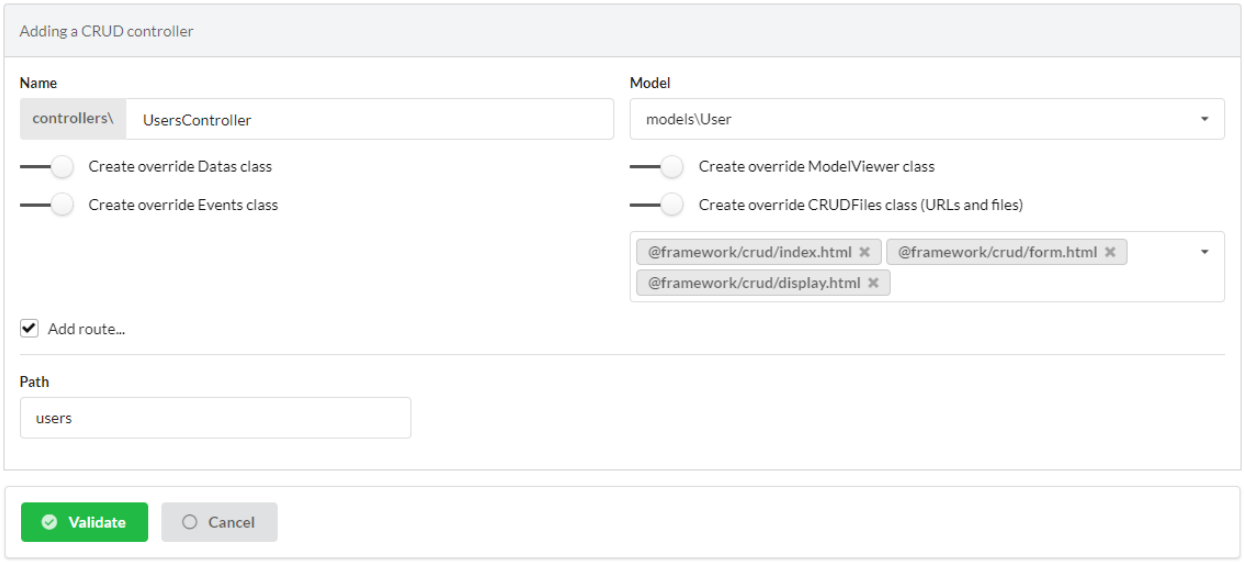
El botón de eliminación de cada fila permite editar una instancia

Visualización del mensaje de confirmación antes de la eliminación:



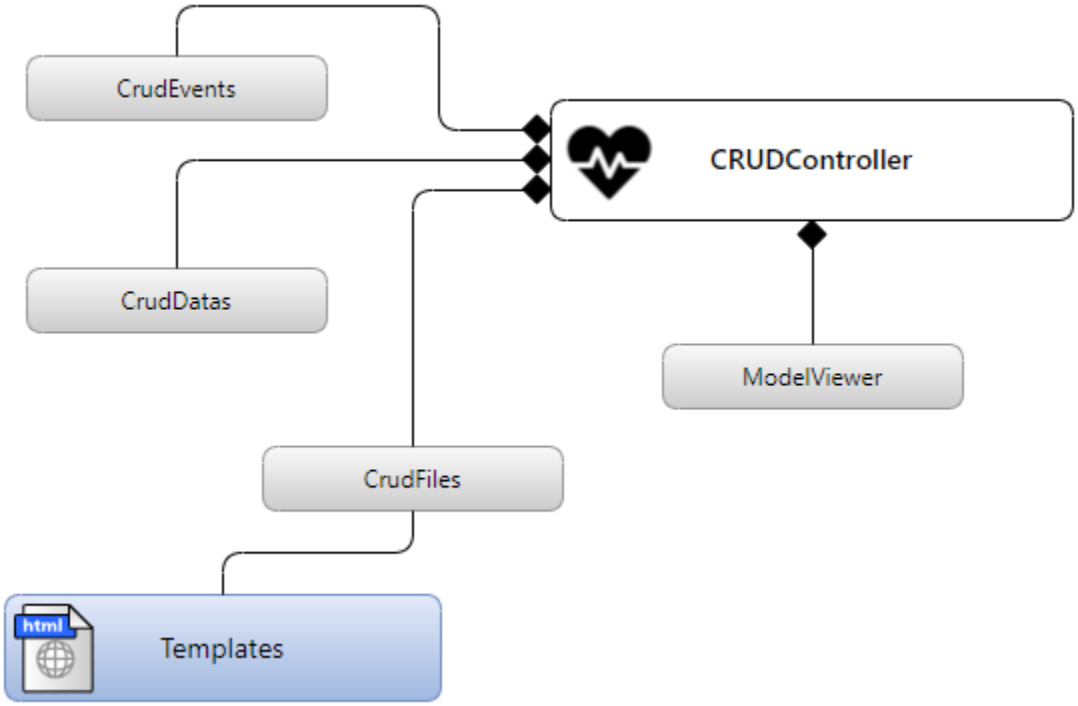
12.1.3 Personalización

Cree de nuevo un ResourceCrudController desde la interfaz de administración:



Ahora es posible personalizar el módulo utilizando overriding.

Visión general



Clases de anulación

Métodos de ResourceCRUDController a sustituir

Método	Significado	Retorno por defecto
rutas		
index()	Página por defecto: lista de todos los objetos	
edit($modal=»no», $ids=»»)	Edita una instancia	
newModel($modal=»no»)	Crea una nueva instancia	
display($modal=»no»,$ids=»»)	Muestra una instancia	
delete($ids)	Elimina una instancia	
update()	Muestra el resultado de la actualización de una instancia	
showDetail($ids)	Muestra los miembros asociados con claves externas	
refresh_()	Refresca el área correspondiente a la DataTable (#lv)	
refreshTable($id=null)	//TO COMMENT	

Métodos de ModelViewer a sustituir

Método	Significado
	ruta **in**
getModelDataTable($instances, $model,$totalCount,$page=1)	Crea la dataTable y añade su comportamiento
getDataTableInstance($instances,$model,$totalCount,$page=1)	Crea la dataTable
recordsPerPage($model,$totalCount=0)	Devuelve el recuento de filas a mostrar (si es null no hay
getGroupByFields()	Devuelve una matriz de miembros sobre la que realizar u
getDataTableRowButtons()	Devuelve un array de botones a mostrar para cada fila [«e
onDataTableRowButton(HtmlButton $bt, ?string $name)	Para modificar los botones de fila de dataTable
getCaptions($captions, $className)	Devuelve los títulos de las cabeceras de las columnas
	ruta **de**
showDetailsOnDataTableClick()	Anular para asegurarse de que se muestra o no el detalle
onDisplayFkElementListDetails($element,$member,$className,$object)	Modificar para mostrar cada elemento de un componente
getFkHeaderElementDetails($member, $className, $object)	Devuelve la cabecera de un único objeto foraneo (issue fr
getFkElementDetails($member, $className, $object)	Devuelve un componente para mostrar un único objeto fo
getFkHeaderListDetails($member, $className, $list)	Devuelve la cabecera de una lista de objetos foraneos (on
getFkListDetails($member, $className, $list)	Devuelve un componente de lista para mostrar una colecc
	rutas **e**
getForm($identifier, $instance)	Devuelve el formulario para añadir o modificar un objeto
formHasMessage()	Determina si el formulario tiene un título de mensaje
getFormModalTitle($instance)	Devuelve el título del modal del formulario
onFormModalButtons($btOkay, $btCancel)	Hook para actualizar los botones modales
getFormTitle($form,$instance)	Devuelve una matriz asociativa que define el título del me
setFormFieldsComponent(DataForm $form,$fieldTypes)	Establece los componentes de cada campo
onGenerateFormField($field)	Para hacer algo cuando $field se genera en el formulario
isModal($objects, $model)	Condición para determinar si el formulario de edición o a
getFormCaptions($captions, $className, $instance)	Devuelve las leyendas de los campos de formulario
	ruta **di**
getModelDataElement($instance,$model,$modal)	Devuelve un objeto DataElement para mostrar la instanci
getElementCaptions($captions, $className, $instance)	Devuelve los subtítulos de los campos DataElement
	ruta **de**
onConfirmButtons(HtmlButton $confirmBtn,HtmlButton $cancelBtn)	Para anular la modificación de los botones de confirmaci

Métodos CRUDDatas a sustituir

Método	Significado	Retorno por defecto
ruta **index**		
_getInstancesFilter($model)	Añade una condición para filtrar las instancias mostradas en dataTable	1=1
getFieldNames($model)	Devuelve los campos a mostrar en la acción **index** para $model	todos los nombres de los miembros
getSearchFieldNames($model)	Devuelve los campos que se utilizarán en las consultas de búsqueda	todos los nombres de los miembros
rutas **edit** y **newModel**.		
getFormFieldNames($model,$instance)	Devuelve los campos a actualizar en las acciones **edit** y **newModel** para $model	todos los nombres de los miembros
getManyToOneDatas($fkClass,$instance,$member)	Devuelve una lista (filtrada) de objetos $fkClass para mostrar en una lista html	todas las instancias $fkClass
getOneToManyDatas($fkClass,$instance,$member)	Devuelve una lista (filtrada) de objetos $fkClass para mostrar en una lista html	todas las instancias $fkClass
getManyToManyDatas($fkClass,$instance,$member)	Devuelve una lista (filtrada) de objetos $fkClass para mostrar en una lista html	todas las instancias $fkClass
ruta **display**		
getElementFieldNames($model)	Devuelve los campos a mostrar en la acción **display** para $model	todos los nombres de los miembros

Métodos CRUDEvents a sustituir

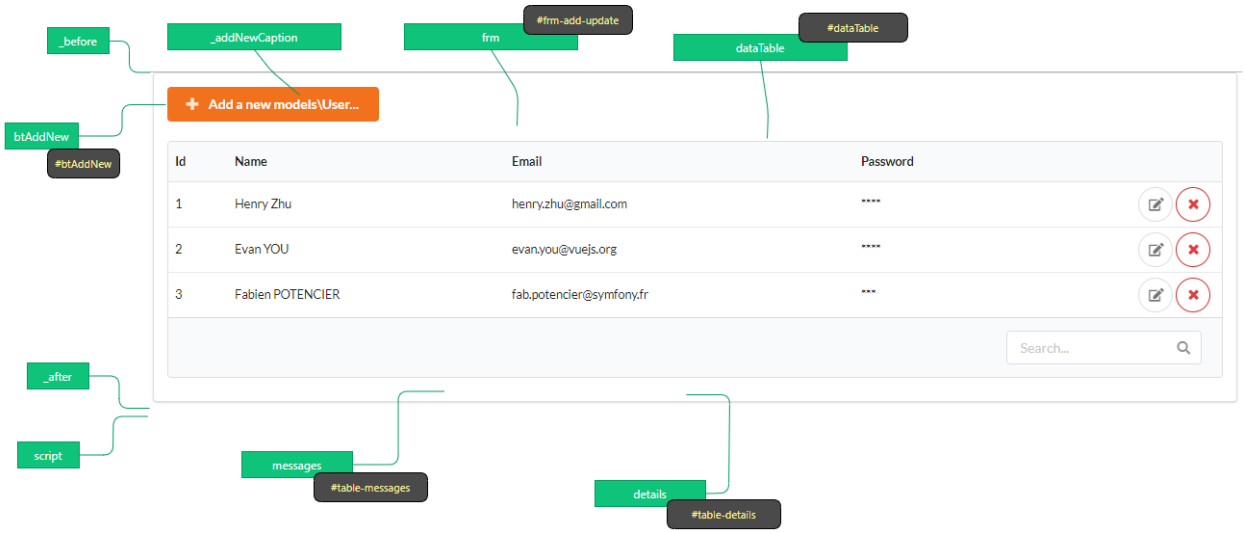
Método	Significado	Retorno por defecto
ruta **index**		
onConfDeleteMessage(CRUDMessage $message,$instance)	Devuelve el mensaje de confirmación que aparece antes de eliminar una instancia	CRUDMessage
onSuccessDeleteMessage(CRUDMessage $message,$instance)	Devuelve el mensaje mostrado tras un borrado	CRUDMessage
onErrorDeleteMessage(CRUDMessage $message,$instance)	Devuelve el mensaje mostrado cuando se ha producido un error al borrar	CRUDMessage
rutas **edit** y **newModel**.		
onSuccessUpdateMessage(CRUDMessage $message)	Devuelve el mensaje que aparece cuando se añade o inserta una instancia	CRUDMessage
onErrorUpdateMessage(CRUDMessage $message)	Devuelve el mensaje que aparece cuando se produce un error al actualizar o insertar	CRUDMessage
onNewInstance(object $instance)	Se activa tras la creación de una nueva instancia	
onBeforeUpdate(object $instance, bool $isNew)	Se activa antes de la actualización de la instancia	
todas las rutas		
onNotFoundMessage(CRUDMessage $message,$ids)	Devuelve el mensaje que aparece cuando una instancia no existe	
onDisplayElements($dataTable,$objects,$refresh)	Se activa después de mostrar objetos en dataTable	

Métodos de CRUDFiles que deben sustituirse

Método	Significado	Retorno por defecto
archivos de plantilla		
getViewBaseTemplate()	Devuelve la plantilla base para todas las acciones Crud si getBaseTemplate devuelve un nombre de archivo de plantilla base	@framework/crud/baseTemplate.html
getViewIndex()	Devuelve la plantilla de la ruta **index**.	@framework/crud/index.html
getViewForm()	Devuelve la plantilla de las rutas **edit** y **newInstance**.	@framework/crud/form.html
getViewDisplay()	Devuelve la plantilla de la ruta **display**.	@framework/crud/display.html
Urls		
getRouteRefresh()	Devuelve la ruta para refrescar la ruta índice	/refresh_
getRouteDetails()	Devuelve la ruta para la ruta de detalle, cuando el usuario hace clic en una fila dataTable	/showDetail
getRouteDelete()	Devuelve la ruta para eliminar una instancia	/delete
getRouteEdit()	Returns the route for editing an instance	/edit
getRouteDisplay()	Devuelve la ruta para mostrar una instancia	/display
getRouteRefreshTable()	Devuelve la ruta para refrescar la dataTable	/refreshTable
getDetailClickURL($model)	Devuelve la ruta asociada a una instancia de clave externa en la lista	«»

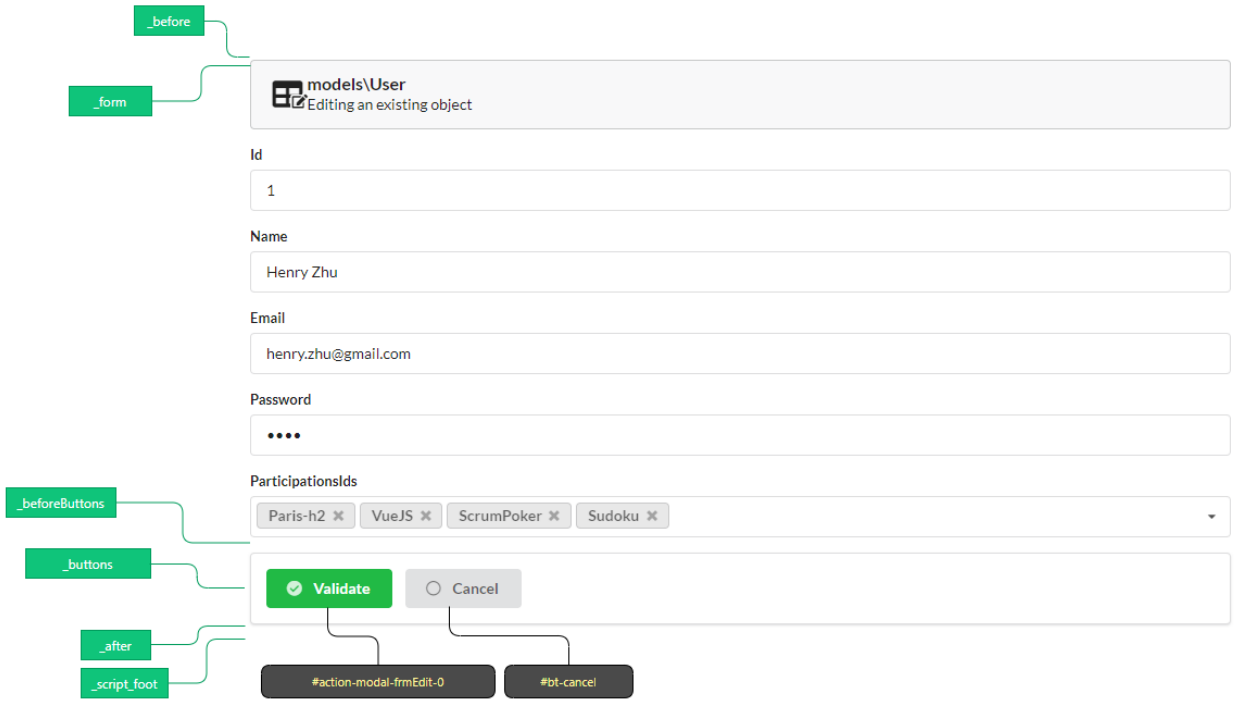
Estructura de plantillas Twig

index.html



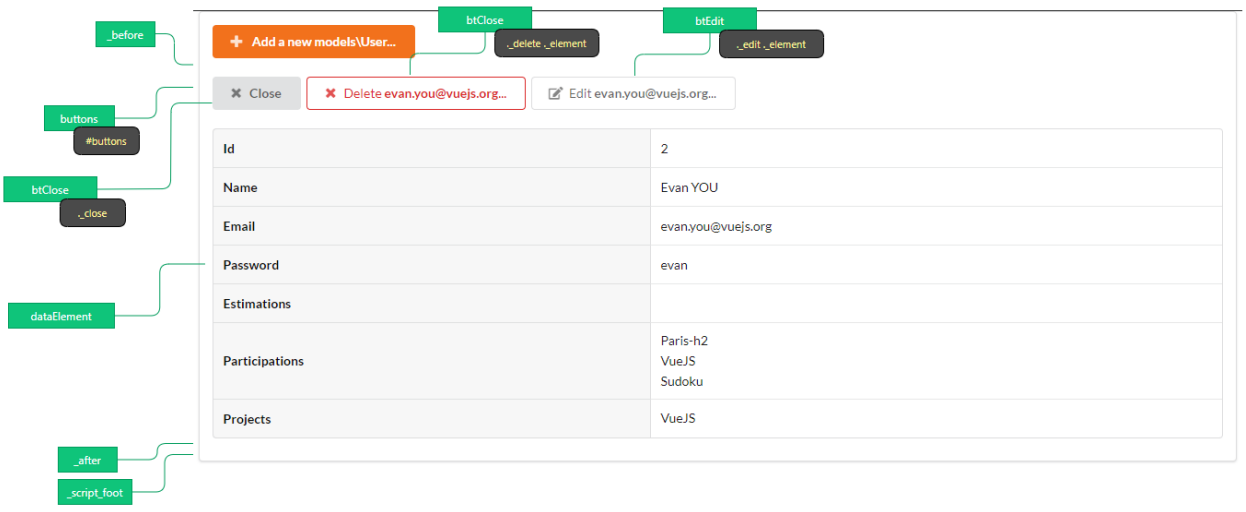
form.html

Se muestra en el bloque **frm**



display.html

Se muestra en el bloque **frm**

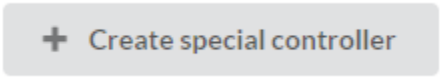


12.2 MultiResourceCrudController

Nota: El *MultiResourceCRUDController* muestra un índice que permite navegar entre los CRUDs de los modelos.

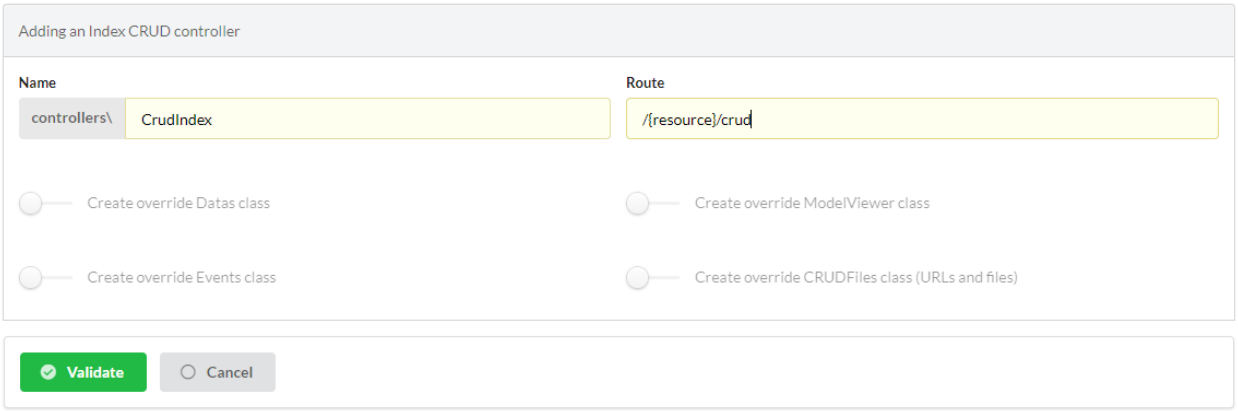
12.2.1 Creación

En la interfaz de administración (web-tools), active la parte **Controllers**, y elija crear **Index Crud controller**:



A continuación, rellene el formulario:

- Introduzca el nombre del controlador
- La ruta de acceso (que debe contener la parte variable *{resource}*)
- A continuación, haga clic en el botón de validación



12.2.2 Descripción de las características

El controlador generado:

Lista 2: app/controllers/CrudIndex.php

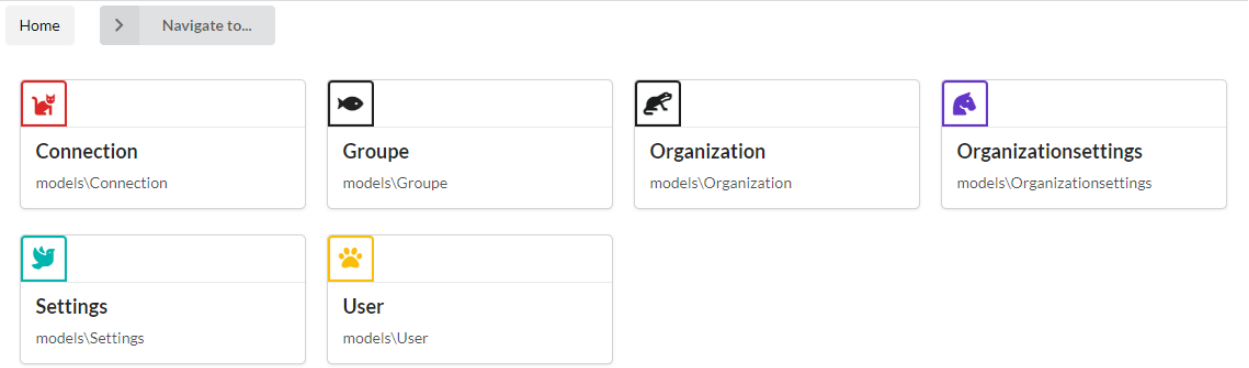
```php
<?php
namespace controllers;
use Ubiquity\attributes\items\router\Route;

#[Route(path: "/{resource}/crud",inherited: true,automated: true)]
class CrudIndex extends \Ubiquity\controllers\crud\MultiResourceCRUDController{

		#[Route(name: "crud.index",priority: -1)]
		public function index() {
			parent::index();
		}
```

(continué en la próxima página)

(proviene de la página anterior)

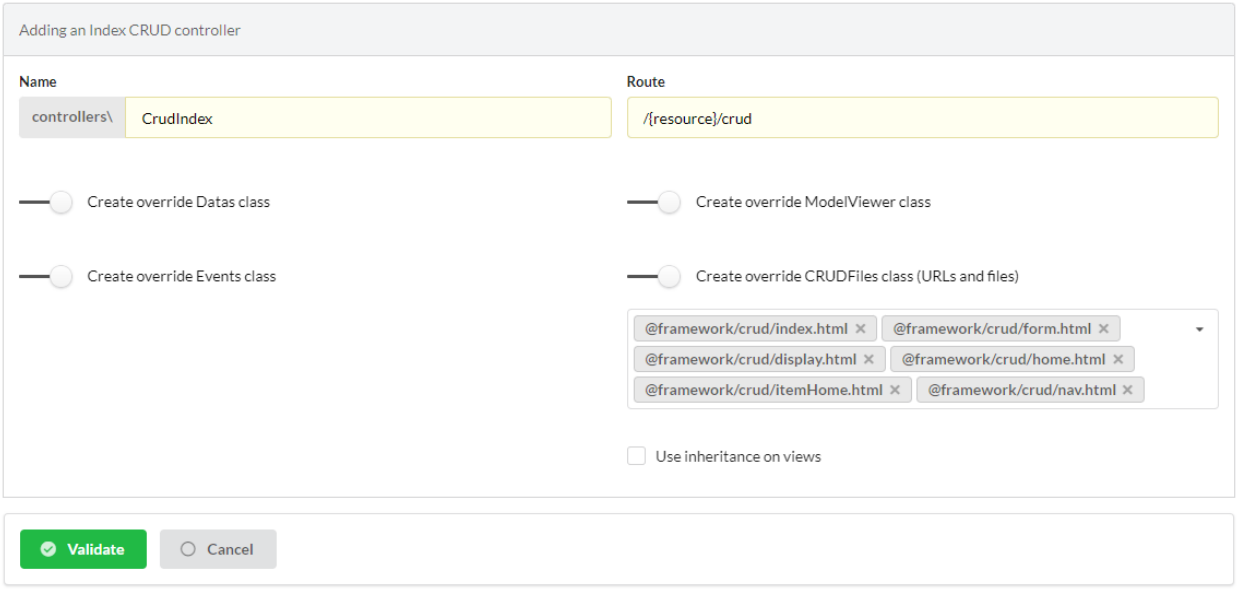
```
        #[Route(path: "#//home/crud",name: "crud.home",priority: 100)]
        public function home(){
                parent::home();
        }

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

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

}
```

Prueba el controlador creado en la url */home/crud*:



12.2.3 Personalización

Crea de nuevo un *MultiResourceCrudController* desde la interfaz de administración:



Ahora es posible personalizar el módulo utilizando overriding como el *ResourceCRUDControllers*.

Clases específicas para anular

Métodos de MultiResourceCRUDController a sustituir

Método	Significado	Retorno por defecto
rutas		
home ()	Página de inicio : lista de todos los modelos	
▪	Todas las rutas de `CRUDController`	
Eventos		
onRenderView(array &$data)	Antes de la presentación de la página de inicio	
Configuración		
hasNavegación()	Devuelve True para mostrar el menú desplegable de navegación	True
getIndexModels()	Devuelve la lista de modelos disponibles para mostrar	modelos de la base de datos por defecto
getIndexModelsDetails()	Devuelve una matriz asociativa (title, icon url) para cada modelo	[]
getIndexDefaultIcon(string $resource)	Devuelve el icono de un modelo	Un animal al azar
getIndexDefaultTitle(string $resource)	Devuelve el título de un modelo	El nombre del recurso
getIndexDefaultDesc(string $modelClass)	Devuelve la descripción de un modelo	El nombre completo de la clase
getIndexDefaultUrl(string $resource)	Devuelve la url asociada a un modelo	La ruta
getIndexDefaultMeta(string $modelClass)	Devuelve la meta de un modelo	
getIndexType()	Define las clases css del componente index	tarjetas
getModelName()	Devuelve el nombre completo del modelo para $this->resource	Del modelo por defecto NS

Métodos de CRUDFiles que deben sustituirse

Método	Significado	Retorno por defecto
archivos de plantilla		
getViewHome()	Devuelve la plantilla base para la vista de inicio	@framework/crud/home.html
getViewItemHome()	Devuelve la plantilla de un elemento de la ruta de inicio	@framework/crud/itemHome.html
getViewNav()	Devuelve la plantilla para mostrar modelos en un desplegable	@framework/crud/nav.html

Nota: Todos los demás métodos de las clases `CRUDController`, `CRUDFiles`, `CRUDEvents` y `CRUDDatas` pueden ser sobreescritos como en el caso de `ResourceCRUDController`.

CAPÍTULO 13

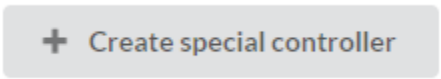
Auth Controllers

The Auth controllers allow you to perform basic authentification with:

- login with an account
- account creation
- logout
- controllers with required authentication

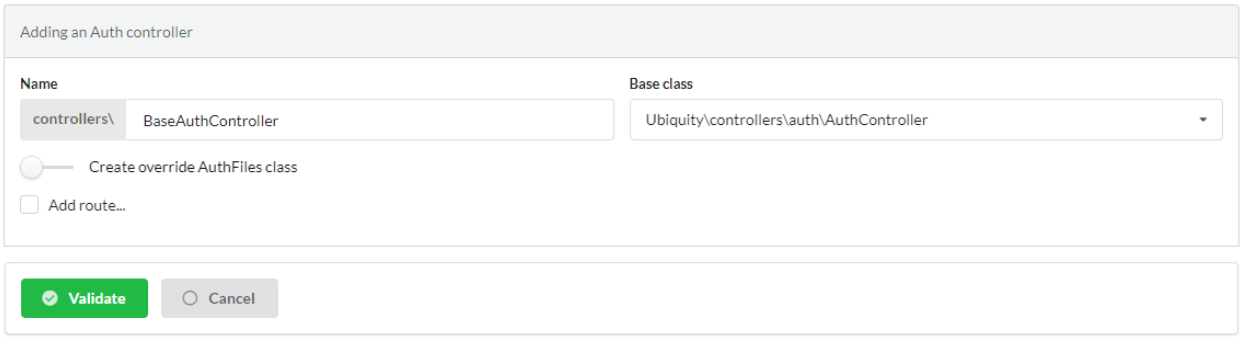
13.1 Creation

In the admin interface (web-tools), activate the **Controllers** part, and choose create **Auth controller**:



Then fill in the form:

- Enter the controller name (BaseAuthController in this case)



The generated controller:

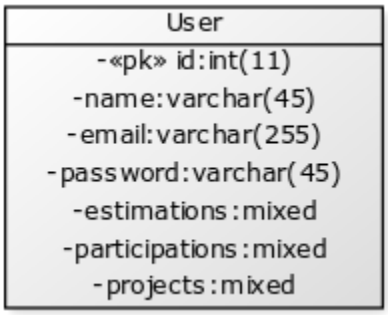
Lista 1: app/controllers/BaseAuthController.php

```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 Implementation of the authentification

Example of implementation with the administration interface : We will add an authentication check on the admin interface.

Authentication is based on verification of the email/password pair of a model **User**:



13.2.1 BaseAuthController modification

Lista 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 {
```

(continué en la próxima página)

(proviene de la página anterior)

```
            return USession::exists($this->_getUserSessionKey());
        }

        public function _getBaseRoute(): string {
            return 'BaseAuthController';
        }
        /**
         * {@inheritDoc}
         * @see \Ubiquity\controllers\auth\AuthController::_getLoginInputName()
         */
        public function _getLoginInputName(): string {
            return "email";
        }
}
```

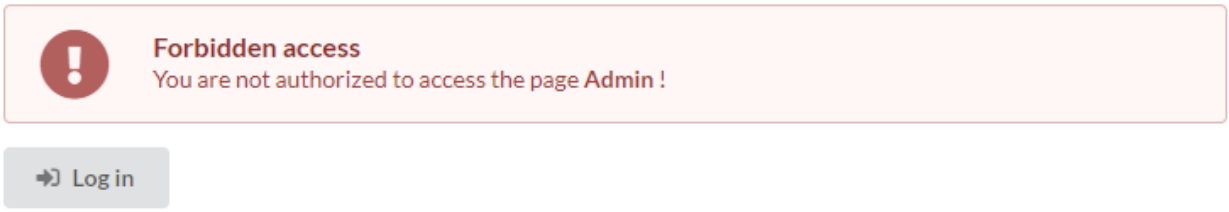
13.2.2 Admin controller modification

Modify the Admin Controller to use BaseAuthController:

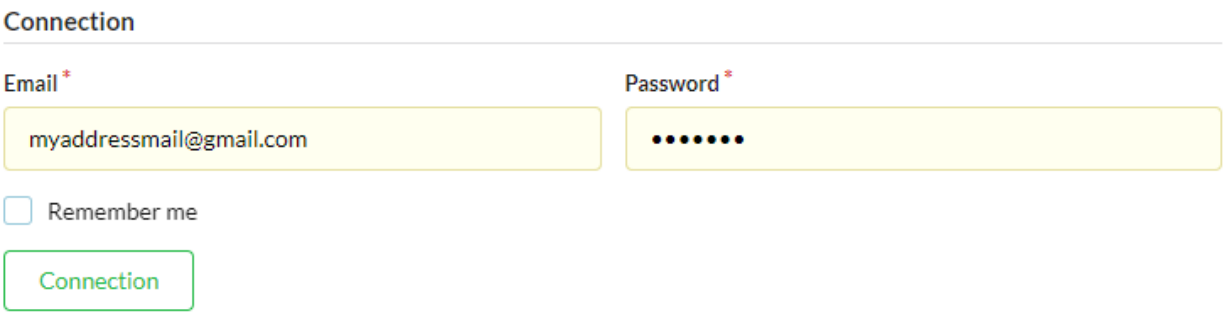
Lista 3: app/controllers/Admin.php

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

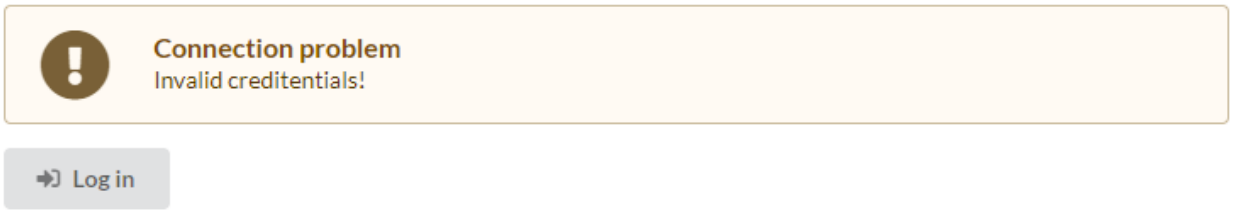
Test the administration interface at **/admin**:



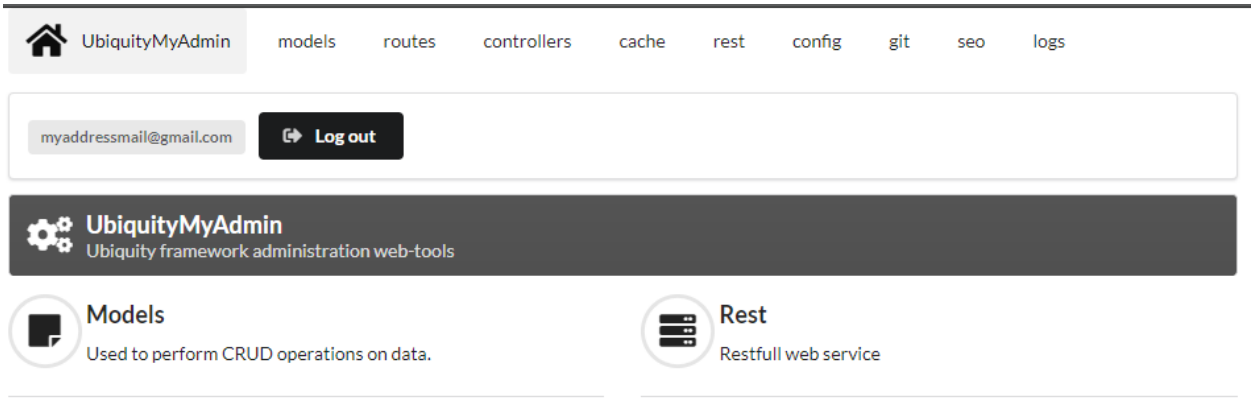
After clicking on **login**:



If the authentication data entered is invalid:



If the authentication data entered is valid:



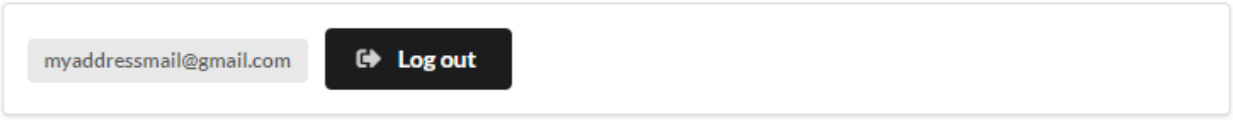
13.2.3 Attaching the zone info-user

Modify the **BaseAuthController** controller:

Lista 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  }
```

The **_userInfo** area is now present on every page of the administration:



It can be displayed in any twig template:

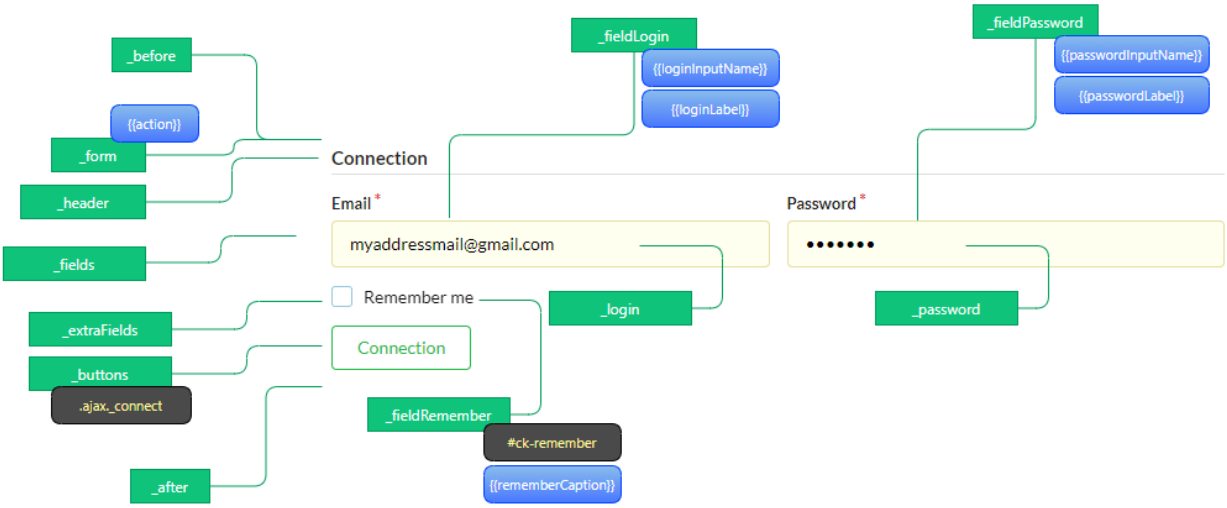
```
{{ _userInfo | raw }}
```

13.3 Description of the features

13.3.1 Customizing templates

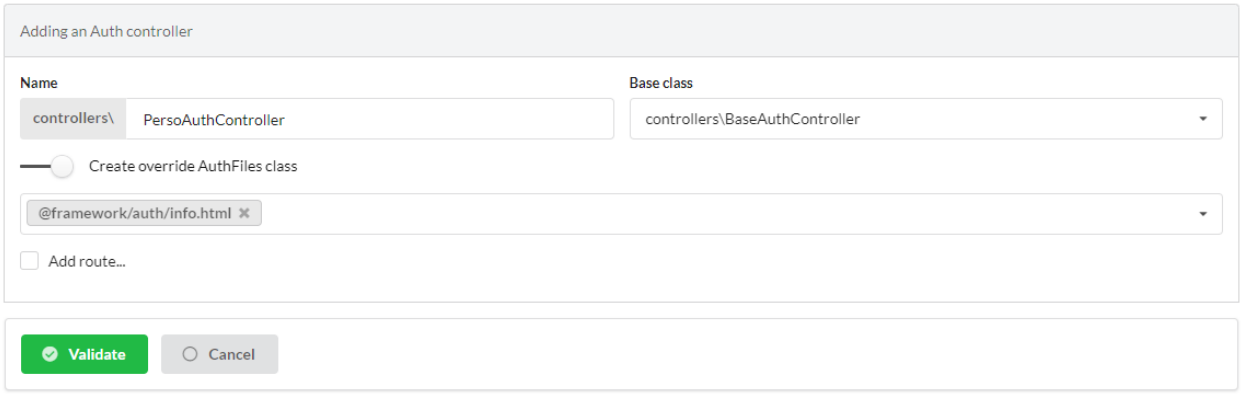
index.html template

The index.html template manages the connection:



Example with the **_userInfo** area:

Create a new AuthController named **PersoAuthController**:



Edit the template **app/views/PersoAuthController/info.html**

Lista 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 %}
```

(continué en la próxima página)

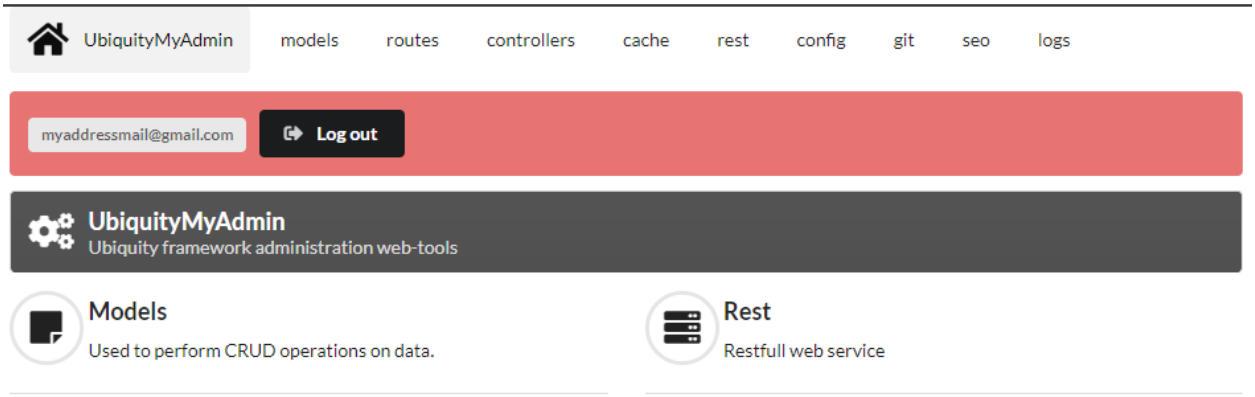
(proviene de la página anterior)

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

Change the AuthController **Admin** controller:

Lista 6: app/controllers/Admin.php

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



13.3.2 Customizing messages

Lista 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");
```

(continué en la próxima página)

(proviene de la página anterior)

```
9       $fMessage->setContent("Login ou mot de passe incorrects !");
10      $this->_setLoginCaption("Essayer à nouveau");
11
12   }
13  ...
14  }
```

13.3.3 Self-check connection

Lista 8: app/controllers/PersoAuthController.php

```
1   class PersoAuthController extends \controllers\BaseAuth{
2   ...
3    /**
4     * {@inheritDoc}
5     * @see \Ubiquity\controllers\auth\AuthController::_checkConnectionTimeout()
6     */
7    public function _checkConnectionTimeout() {
8        return 10000;
9    }
10  ...
11  }
```

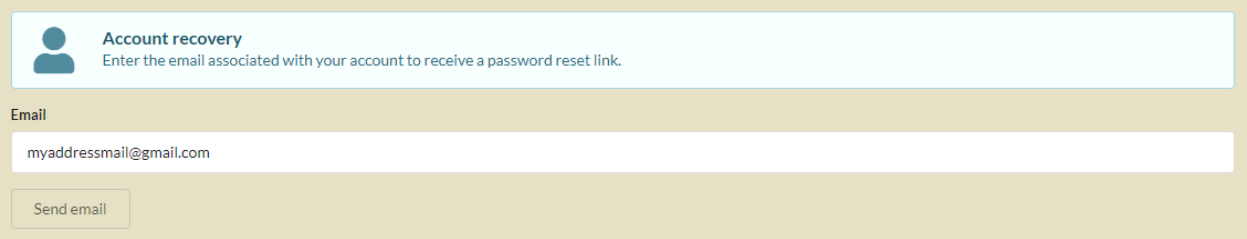
13.3.4 Limitation of connection attempts

Lista 9: app/controllers/PersoAuthController.php

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

13.3.5 Account recovery

account recovery is used to reset the account password. A password reset email is sent, to an email address corresponding to an active account.



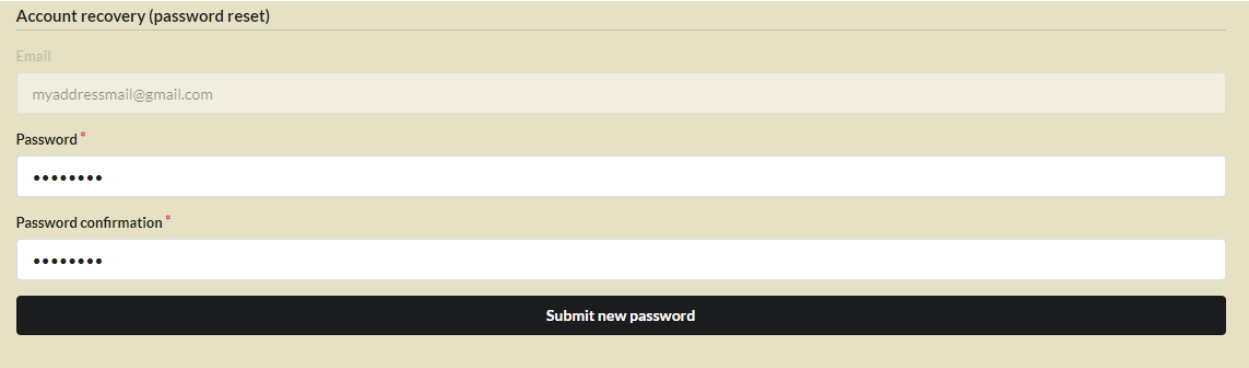
Lista 10: app/controllers/PersoAuthController.php

```
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
```

(continué en la próxima página)

(proviene de la página anterior)

```
22  }
23 }
```



Nota: By default, the link can only be used on the same machine, within a predetermined period of time (which can be modified by overriding the `accountRecoveryDuration` method).

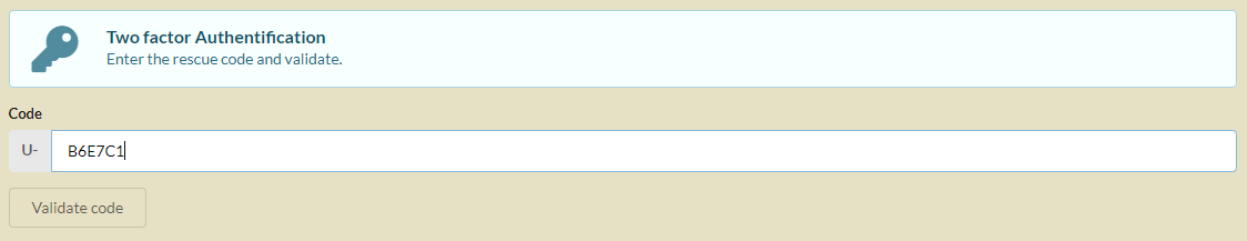
13.3.6 Activation of MFA/2FA

Multi-factor authentication can be enabled conditionally, based on the pre-logged-in user's information.

Nota: Phase 2 of the authentication is done in the example below by sending a random code by email. The AuthMailerClass class is available in the `Ubiquity-mailer` package.

Lista 11: app/controllers/PersoAuthController.php

```
1  class PersoAuthController extends \controllers\BaseAuth{
2  ...
3   /**
4    * {@inheritDoc}
5    * @see \Ubiquity\controllers\auth\AuthController::has2FA()
6    */
7   protected function has2FA($accountValue=null):bool{
8       return true;
9   }
10
11  protected function _send2FACode(string $code, $connected):void {
12      MailerManager::start();
13      $mail=new AuthMailerClass();
14      $mail->to($connected->getEmail());
15      $mail->setCode($code);
16      MailerManager::send($mail);
17  }
18 ...
19 }
```

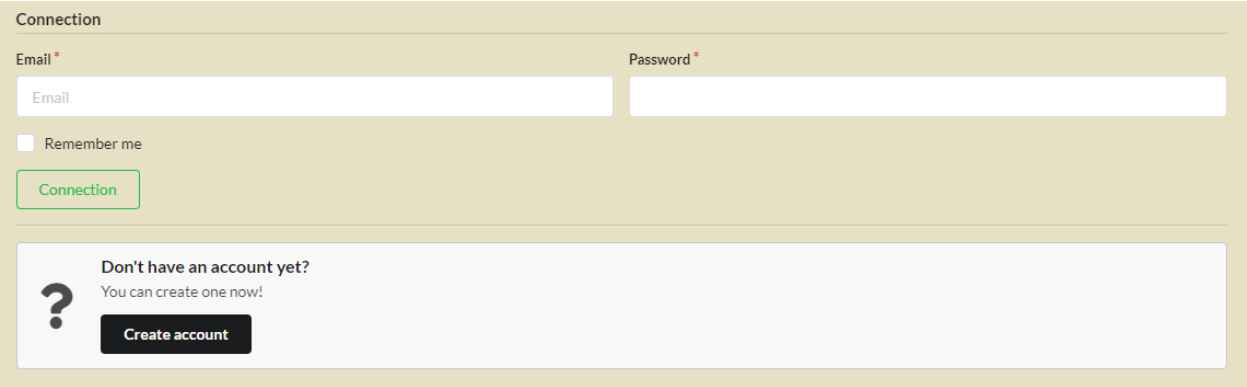


Nota: It is possible to customize the creation of the generated code, as well as the prefix used. The sample below is implemented with `robthree/twofactorauth` library.

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

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

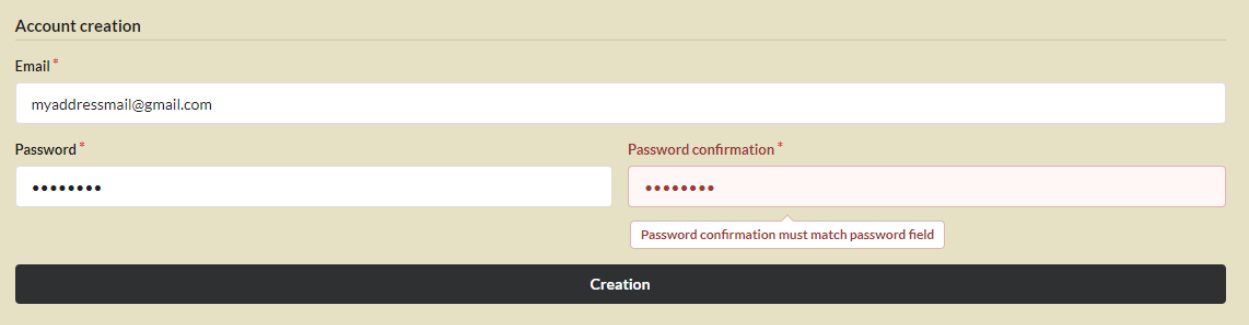
13.3.7 Account creation

The activation of the account creation is also optional:



Lista 12: app/controllers/PersoAuthController.php

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

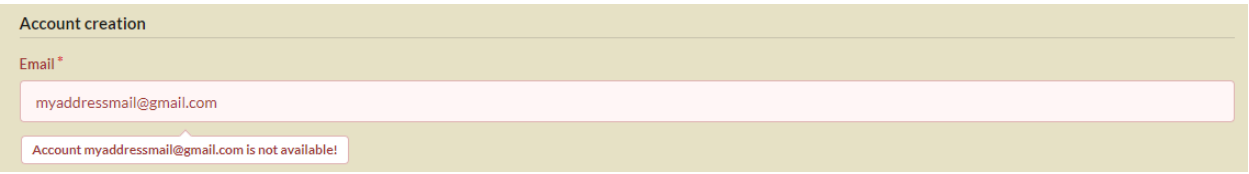


In this case, the _create method must be overridden in order to create the account:

```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;
}
```

You can check the validity/availability of the login before validating the account creation form:

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



A confirmation action (email verification) may be requested from the user:

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

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

(continué en la próxima página)

(proviene de la página anterior)

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

Nota: It is possible to customize these parts by overriding the associated methods, or by modifying the interfaces in the concerned templates.

CAPÍTULO 14

Base de datos

La clase **DAO** es responsable de las operaciones de carga y persistencia de los modelos :

14.1 Conexión a la base de datos

Compruebe que los parámetros de conexión a la base de datos se han introducido correctamente en el archivo de configuración:

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

```
· Displaying config variables from app/config/config.php file
```

field	value
database	· type : 'mysql' · dbName : 'messagerie' · serverName : '127.0.0.1' · port : 3306 · user : 'root' · password : '' · options : [] · cache : false · wrapper : 'Ubiquity\\db\\providers\\pdo\\PDOWrapper'

14.1.1 Conexión transparente

Desde Ubiquity 2.3.0, La conexión a la base de datos se realiza automáticamente la primera vez que se solicita:

```
use Ubiquity\orm\DAO;

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

Este es el caso de todos los métodos de la clase **DAO** utilizados para realizar operaciones CRUD.

14.1.2 Conexión explícita

En algunos casos, sin embargo, puede ser útil establecer una conexión explícita con la base de datos, especialmente para comprobar la conexión.

```
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();
}
```

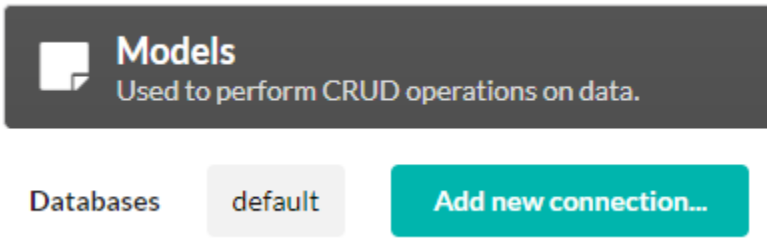
14.2 Conexiones múltiples

14.2.1 Añadir una nueva conexión

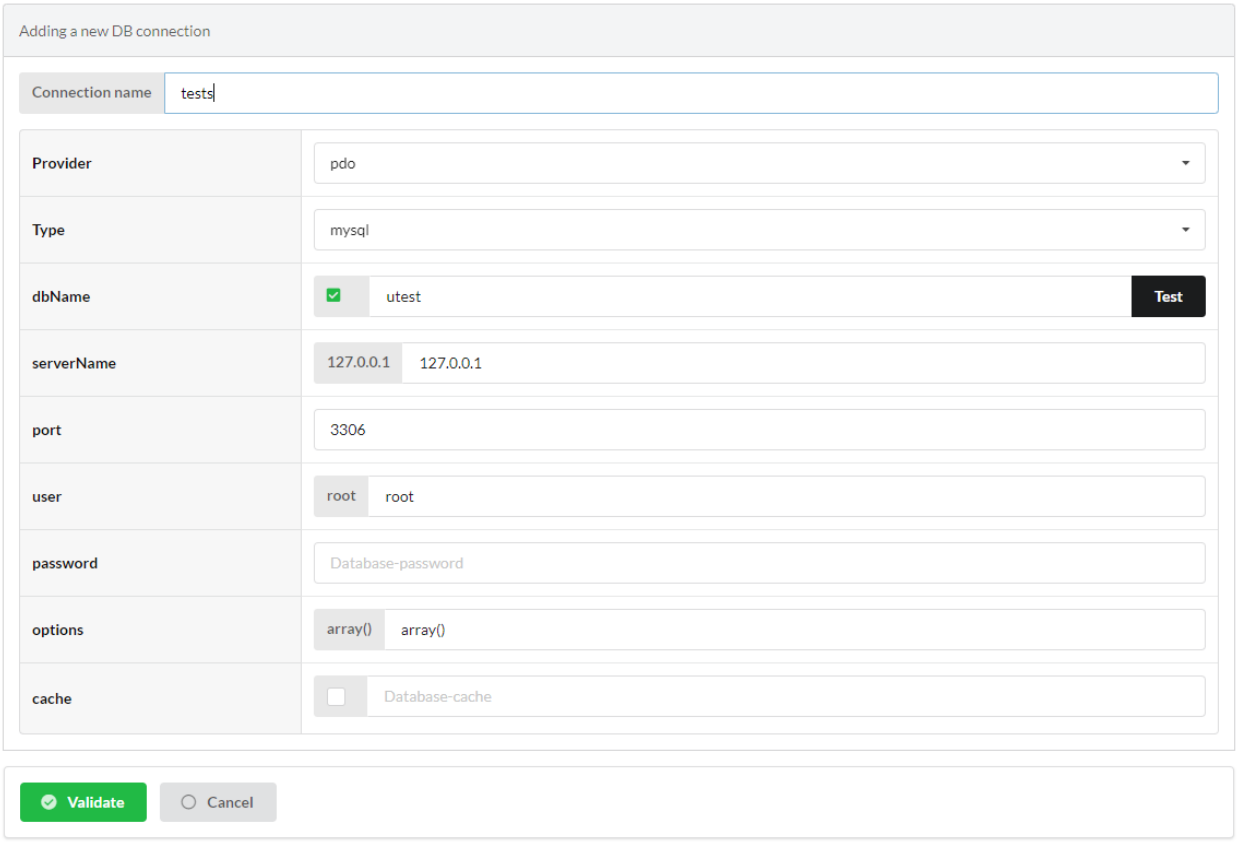
Ubiquity permite gestionar varias conexiones a bases de datos.

Con Webtools

En la parte **Models**, seleccione el botón **Add new connection**:



Define los parámetros de configuración de la conexión:



Generar modelos para la nueva conexión: Los modelos generados incluyen la anotación `@database` o el atributo `Database` mencionando su enlace a la conexión.

Atributos

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

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

Anotaciones

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

Los modelos se generan en una subcarpeta de `models`.

Con varias conexiones, no olvide añadir la siguiente línea al archivo `services.php`:

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

El método `start` realiza la correspondencia entre cada modelo y su conexión asociada.

CAPÍTULO 15

Generación de modelos

15.1 A partir de la base de datos existente

- with devtools
- with webtools

15.2 Desde cero

- Models creation with devtools
- migrations

CAPÍTULO 16

ORM

Nota: si desea generar automáticamente los modelos, consulte la parte *generating models* .

Una clase modelo es simplemente un objeto php sin herencia. |br|Los modelos se encuentran por defecto en la carpeta **app\models**. |br|El mapeo relacional de objetos (ORM) se basa en anotaciones de miembros o atributos (desde PHP8) en la clase modelo.

16.1 Definición de modelos

16.1.1 Un modelo básico

- Un modelo debe definir su clave primaria utilizando la anotación **@id** en los miembros correspondientes
- Los miembros serializados deben tener getters y setters
- Sin ninguna otra anotación, una clase corresponde a una tabla con el mismo nombre en la base de datos, cada miembro corresponde a un campo de esta tabla

Atributos

Lista 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
```

(continué en la próxima página)

(proviene de la página anterior)

```
    private $firstname;

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

Anotaciones

Lista 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 Mapeo

Table->Class

Si el nombre de la tabla es diferente del nombre de la clase, la anotación **@table** permite especificar el nombre de la tabla.

Atributos

Lista 3: app/models/User.php

```
namespace models;

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

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

    #[Id]
```

(continué en la próxima página)

(proviene de la página anterior)

```
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  }
```

Anotaciones

Lista 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  }
```

Field->Member

Si el nombre de un campo es diferente del nombre de un miembro de la clase, la anotación **@column** permite especificar un nombre de campo diferente.

Atributos

Lista 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
```

(continué en la próxima página)

(proviene de la página anterior)

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

   #[Id]
   private $id;

   #[Column('column_name')]
   private $firstname;

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

Anotaciones

Lista 6: app/models/User.php

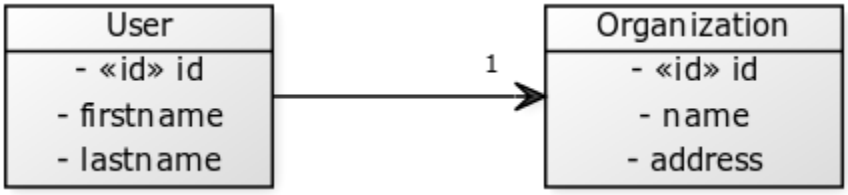
```
namespace models;

/**
 * @table("user")
 */
class User{
   /**
    * @id
    */
   private $id;

   /**
    * column("user_name")
    */
   private $firstname;

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

16.1.3 Asociaciones

Nota: Convención de nombres Los nombres de los campos de clave foránea consisten en el nombre de la clave primaria de la tabla referenciada seguido del nombre de la tabla referenciada cuya primera letra es mayúscula. |br|Ejemplo: `idUsuario` para la tabla `usuario` cuya clave primaria es `id`.

ManyToOne

Un **usuario** pertenece a una **organización**:



Atributos

Lista 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;
   }
}
```

Anotaciones

Lista 8: app/models/User.php

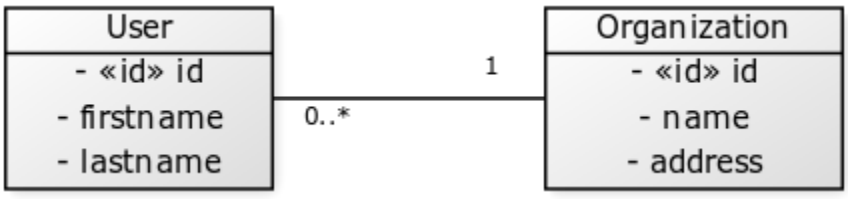
```php
namespace models;

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;
  }
}
```

La anotación **@joinColumn** o el atributo **JoinColumn** especifica que:

- El miembro **$organization** es una instancia de **modelsOrganization**.
- La tabla **user** tiene una clave externa **idOrganization** que hace referencia a la clave primaria de la organización
- Esta clave externa no es null => un usuario siempre tendrá una organización

OneToMany

Una **organización** tiene muchos **usuarios**:



Atributos

Lista 9: app/models/Organization.php

```php
namespace models;

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

(continué en la próxima página)

(proviene de la página anterior)

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

class Organization{

    #[Id]
    private $id;

    private $name;

    #[OneToMany(mappedBy: 'organization', className: \models\User::class)]
    private $users;
}
```

Anotación

Lista 10: app/models/Organization.php

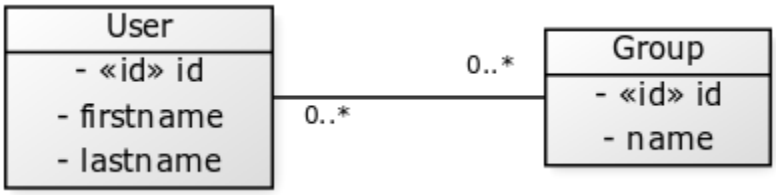
```
namespace models;

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

    private $name;

    /**
     * @oneToMany("mappedBy"=>"organization","className"=>"models\\User")
     */
    private $users;
}
```

En este caso, la asociación es bidireccional. |br|La anotación **@oneToMany** sólo debe especificar:

- La clase de cada usuario en la matriz de usuarios : **modelsUser**
- el valor de **@mappedBy** es el nombre del atributo association-mapping en el lado propietario: **$organization** en la clase **User**.

ManyToMany

- Un **usuario** puede pertenecer a **grupos**.
- Un **grupo** está formado por varios **usuarios**.



Atributos

Lista 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;

}
```

Anotaciones

Lista 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;

}
```

Atributos

Lista 13: app/models/Group.php

```php
namespace models;

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

```

(continué en la próxima página)

(proviene de la página anterior)

```
class Group{

   #[Id]
   private $id;

   private $name;

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

}
```

Anotaciones

Lista 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 no se respetan las convenciones de nomenclatura para las claves externas, es posible especificar los campos relacionados.

Atributos

Lista 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;

```

(continué en la próxima página)

(proviene de la página anterior)

```
    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;

}
```

Anotaciones

Lista 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 Anotaciones ORM

16.2.1 Anotaciones para las clases

@annotation	rol	propiedades	rol
@database	Define el desplazamiento de la base de datos asociada (definido en el archivo de configuración)		
@table	Define el nombre de la tabla asociada.		

16.2.2 Anotaciones para los miembros

@annotation	rol	propiedades	rol
@id	Define la(s) clave(s) primaria(s).		
@column	Especifique las características del campo asociado.	name	Nombre del campo asociado
		nullable	true si el valor puede ser null
		dbType	Tipo de campo en la base de datos
@transient	Especifica que el campo no es persistente.		

16.2.3 Asociaciones

@annotation (extends)	rol	propiedades [opcional]	rol
@manyToOne	Define una asociación de un solo valor a otra clase de entidad.		
@joinColumn (@column)	Indica la clave externa en la asociación manyToOne.	className	Clase del miembro
		[referencedColumnName]	Nombre de la columna asociada
@oneToMany	Define una asociación multivaluada (multivalued) a otra clase de entidad.	className	Clase de los objetos en miembro
		[mappedBy]	Nombre del atributo de asignación de asociaciones (association-mapping) en el lado propietario
@manyToMany	Define una asociación multivaluada (many-valued) con multiplicidad de muchos a muchos (many-to-many).	targetEntity	Clase de los objetos en miembro
		[inversedBy]	Nombre del miembro de la asociación (association-member) en el lado inverso (inverse-side)
		[mappedBy]	Nombre de la asociación miembro (association-member) de la parte propietaria
@joinTable	Define la tabla de asociación para la multiplicidad muchos a muchos (many-to-many)	name	Nombre de la tabla de asociación
		[joinColumns]	@column => nombre y referencedColumnName para este lado
		[inverseJoinColumns]	@column => nombre y referencedColumnName para el otro lado

CAPÍTULO 17

DAO

La clase **DAO** es responsable de las operaciones de carga y persistencia de los modelos :

17.1 Conexión a la base de datos

Compruebe que los parámetros de conexión a la base de datos se han introducido correctamente en el archivo de configuración:

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

Desde la versión 2.3.0

El inicio de la base de datos con `DAO::startDatabase($config)` en el archivo services.php es inútil, no es necesario iniciar la base de datos, la conexión se realiza automáticamente en la primera petición. Usa `DAO::start()` en el archivo **app/config/services.php** cuando uses varias bases de datos (con la característica multi db)

17.2 Carga de datos

17.2.1 Cargar una instancia

Cargando una instancia de la clase *models\User* con id *5*.

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

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

Cargar una instancia utilizando una condición:

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

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

Carga de BelongsTo

Por defecto, los miembros definidos por una relación **belongsTo** se cargan automáticamente

Cada usuario pertenece a una sola categoría:

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

Es posible evitar esta carga por defecto; el tercer parámetro permite cargar o no los miembros belongsTo:

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

Carga de HasMany

La carga de miembros **hasMany** debe ser siempre explícita; el tercer parámetro permite la carga explícita de miembros.

Cada usuario tiene muchos grupos:

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

Clave primaria compuesta

O bien el modelo *ProductDetail* correspondiente a un producto pedido en una orden y cuya clave primaria es compuesta:

Atributos

Lista 1: app/models/ProductDetail.php

```
1  namespace models;
2
3  use Ubiquity\attributes\items\Id;
4
5  class ProductDetail{
6
7      #[Id]
8      private $idProduct;
9
10     #[Id]
11     private $idCommand;
12
13     ...
14 }
```

Anotaciones

Lista 2: app/models/ProductDetail.php

```php
namespace models;

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

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

  ...
}
```

El segundo parámetro *$keyValues* puede ser un array si la clave primaria es compuesta:

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

17.2.2 Carga multiple de objetos

Carga de instancias de la clase *User*:

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

carga de los miembros relacionados

Carga de instancias de la clase *User* con su categoría y sus grupos :

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

Descendiendo en la jerarquía de objetos relacionados: Cargando instancias de la clase *User* con su categoría, sus grupos y la organización de cada grupo :

```
$users=DAO::getAll(User::class,['groupes.organization','category']);
foreach($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>";
}
```

Uso de comodines:

Carga de instancias de la clase *User* con su categoría, sus grupos y todos los miembros relacionados de cada grupo:

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

17.2.3 Consulta mediante condiciones

Consultas sencillas

El parámetro *condition* equivale a la parte WHERE de una sentencia SQL:

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

Para evitar inyecciones SQL y beneficiarse de la preparación de sentencias, es preferible realizar una consulta parametrizada:

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

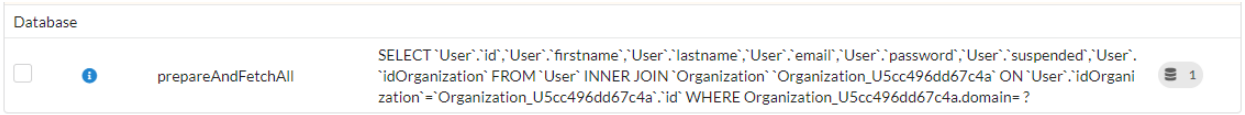
UQueries

El uso de **U-queries** permite establecer condiciones sobre los miembros asociados:

Selección de usuarios cuya organización tiene el dominio **lecnam.net**:

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

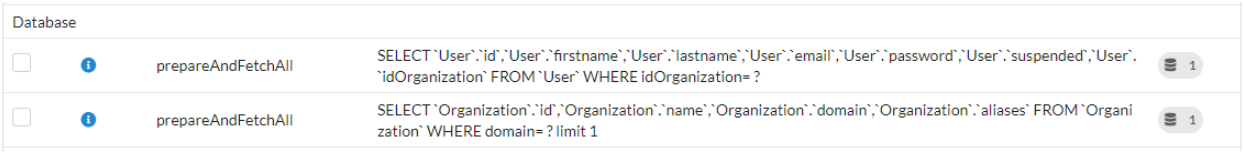
Es posible ver la solicitud generada en los registros (si el registro está activado):



El resultado puede verificarse seleccionando a todos los usuarios de esta organización:

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

Los registros correspondientes:



17.2.4 Contando

Pruebas de existencia

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

Contando

Contar las instancias, lo que no hay que hacer, si los usuarios no están ya cargados:

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

Lo que hay que hacer:

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

Con una condición:

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

con una condición sobre los objetos asociados:

Número de usuarios pertenecientes a la organización denominada **OTAN**.

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

17.3 Modificación de datos

17.3.1 Añadir una instancia

Añadir una organización:

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

Añadir una instancia de Usuario, en una organización:

```
$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 Actualización de una instancia

En primer lugar, debe cargarse la instancia:

```
$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 Borrar una instancia

Si la instancia se carga desde la base de datos:

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

Si la instancia no está cargada, es más apropiado utilizar el método *delete*:

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

17.4 Eliminar varias instancias

Eliminar varias instancias sin carga previa:

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

17.5 Consultas masivas

Las consultas masivas permiten realizar varias operaciones (inserción, modificación o supresión) en una sola consulta, lo que contribuye a mejorar el rendimiento.

17.5.1 Inserciones masivas

Ejemplo de inserciones:

```
$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 Actualizaciones masivas

Ejemplo de actualizaciones:

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

17.5.3 Borrado masivo

Ejemplo de eliminación

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

El método *DAO::flush()* puede ser llamado si hay inserciones, actualizaciones o borrados pendientes.

17.6 Transacciones

17.6.1 Transacciones explícitas

Todas las operaciones DAO se pueden insertar en una transacción, de forma que se pueda atomizar una serie de cambios:

```
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 caso de múltiples bases de datos definidas en la configuración, los métodos relacionados con transacciones pueden tomar el offset de base de datos definido en parámetro.

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

17.6.2 Transacciones implícitas

Algunos métodos DAO utilizan implícitamente transacciones para agrupar operaciones de inserción, actualización o eliminación.

```
$users=DAO::getAll(User::class);
foreach ($users as $user){
    $user->setSuspended(true);
    DAO::toUpdate($user);
}
DAO::updateGroups();//Perform updates in a transaction
```

17.7 Clase SDAO

La clase **SDAO** acelera las operaciones CRUD para las clases de negocio sin relaciones.

En este caso, los modelos deben declarar únicamente los miembros públicos y no respetar la encapsulación habitual.

Lista 3: app/models/Product.php

```
1 namespace models;
2 class Product{
```

(continué en la próxima página)

(proviene de la página anterior)

```
    /**
     * @id
     */
    public $id;

    public $name;

    ...
}
```

La clase **SDAO** hereda de **DAO** y tiene los mismos métodos para realizar operaciones CRUD.

```
use Ubiquity\orm\DAO;

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

17.8 Consultas DAO preparadas

La preparación de ciertas peticiones puede mejorar el rendimiento con los servidores Swoole, Workerman o Roadrunner. Esta preparación inicializa los objetos que luego se utilizarán para ejecutar la consulta. Esta inicialización se realiza al inicio del servidor, o al inicio de cada worker, si existe tal evento.

17.8.1 Ejemplo swoole

Preparación

Lista 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 ?');
});
```

Utilización

Lista 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]);
    ...
}
```

CAPÍTULO 18

Peticiones (Request)

Nota: Para todas las funciones Http, Ubiquity utiliza clases técnicas que contienen métodos estáticos. Se trata de una elección de diseño para evitar la inyección de dependencias que degradaría el rendimiento.

La clase **URequest** proporciona funcionalidad adicional para manipular más fácilmente las matrices nativas **$_POST** y **$_GET** de php.

18.1 Recuperación de datos

18.1.1 Del método get

El método **get** devuelve el valor *null* si la clave **name** no existe en las variables get.

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

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

El método **get** puede ser llamado con el segundo parámetro opcional devolviendo un valor si la clave no existe en las variables get.

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

18.1.2 Del método post

El método **post** devuelve el valor *null* si la clave **nombre** no existe en las variables post.

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

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

El método **post** puede ser llamado con el segundo parámetro opcional devolviendo un valor si la clave no existe en las variables post.

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

El método **getPost** aplica un callback a los elementos del array $_POST y los devuelve (callback por defecto : **htmlEntities**) :

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

18.2 Recuperación y asignación de datos múltiples

Es habitual asignar los valores de un array asociativo a los miembros de un objeto. |br|Este es el caso, por ejemplo, de la validación de un formulario de modificación de objetos.

El método **setValuesToObject** realiza esta operación :

Consideremos una clase **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;
	}
}
```

Consideremos un formulario para modificar un usuario:

```
<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>
```

La acción **update** del controlador **Users** debe actualizar la instancia de usuario a partir de valores POST. Utilizando el método **setPostValuesToObject** se evita la asignación de variables posteadas una a una a los miembros del objeto. También es posible utilizar **setGetValuesToObject** para el método **get**, o **setValuesToObject** para asignar los valores de cualquier array asociativo a un objeto.

Lista 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 }
```

Nota: Los métodos **SetValuesToObject** utilizan setters para modificar los miembros de un objeto. Por lo tanto, la clase en cuestión debe implementar setters para todos los miembros modificables.

18.3 Probar la solicitud

18.3.1 isPost

El método **isPost** devuelve *true* si la petición fue enviada a través del método POST: |br|En el caso de abajo, el método *initialize* sólo carga la vista *vHeader.html* si la petición no es una petición Ajax.

Lista 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{
```

(continué en la próxima página)

(proviene de la página anterior)

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

18.3.2 isAjax

El método **isAjax** devuelve *true* si la consulta es una consulta Ajax:

Lista 3: app/controllers/Users.php

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

18.3.3 isCrossSite

El método **isCrossSite** verifica que la consulta no es cross-site.

CAPÍTULO 19

Respuesta (Response)

Nota: Para todas las funciones Http, Ubiquity utiliza clases técnicas que contienen métodos estáticos. Se trata de una elección de diseño para evitar la inyección de dependencias que degradaría el rendimiento.

La clase **UResponse** maneja sólo las cabeceras, no el cuerpo de la respuesta, que convencionalmente es proporcionado por el contenido mostrado por las llamadas utilizadas para dar salida a los datos (echo, print . . .).

La clase **UResponse** proporciona funcionalidad adicional para manipular más fácilmente las cabeceras de respuesta.

19.1 Añadir o modificar cabeceras

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

Forzar cabecera múltiple del mismo tipo:

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

Fuerza el código de respuesta HTTP al valor especificado:

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

19.2 Definición de cabeceras específicas

19.2.1 content-type

Establecer el tipo de contenido de la respuesta a **application/json**:

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

Establecer el tipo de contenido de la respuesta a **text/html**:

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

Establecer el tipo de contenido de la respuesta a **plain/text**:

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

Establecer el tipo de contenido de la respuesta a **application/xml**:

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

Definición de una codificación específica (el valor por defecto es siempre **utf-8**):

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

19.3 Cache

Forzar la desactivación de la caché del navegador:

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

19.4 Accept

Define qué tipos de contenido, expresados como tipos MIME, puede entender el cliente. Ver Aceptar valores por defecto

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

19.5 Cabeceras de respuesta CORS

Cross-Origin Resource Sharing (CORS) es un mecanismo que utiliza cabeceras HTTP adicionales para indicar a un navegador que permita a su aplicación web que se ejecuta en un origen (dominio) tener permiso para acceder a recursos seleccionados de un servidor en un origen diferente.

19.5.1 Access-Control-Allow-Origin

Ajuste de origen permitido:

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

19.5.2 Access-Control-Allow-methods

Definición de métodos permitidos:

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

19.5.3 Access-Control-Allow-headers

Definición de las cabeceras permitidas:

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

19.5.4 Activación global de CORS

habilitar CORS para un dominio con valores por defecto:

- métodos permitidos: `GET, POST, PUT, DELETE, PATCH, OPTIONS`
- cabeceras permitidas: `X-Requested-With, Content-Type, Accept, Origin, Authorization`

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

19.6 Comprobación de las cabeceras de respuesta

Comprobación de si se han enviado las cabeceras:

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

Comprobando si el tipo de contenido de la respuesta es **application/json**:

Importante: Este método sólo funciona si ha utilizado la clase UResponse para establecer las cabeceras.

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

CAPÍTULO 20

Sesión (Session)

Nota: Para todas las funciones Http, Ubiquity utiliza clases técnicas que contienen métodos estáticos. Se trata de una elección de diseño para evitar la inyección de dependencias que degradaría el rendimiento.

La clase **USession** proporciona funcionalidad adicional para manipular más fácilmente el array nativo **$_SESSION** php.

20.1 Inicio de sesión

La sesión Http se inicia automáticamente si se rellena la clave **sessionName** en el fichero de configuración **app/config.php**:

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

Si la clave sessionName no está rellenada, es necesario iniciar la sesión explícitamente para utilizarla:

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

Nota: El parámetro **nombre** es opcional pero se recomienda para evitar variables conflictivas.

20.2 Crear o editar una variable de sesión

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

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

20.3 Recuperación de datos

El método **get** devuelve el valor *null* si la clave **name** no existe en las variables de sesión.

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

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

El método **get** puede ser llamado con el segundo parámetro opcional devolviendo un valor si la clave no existe en las variables de sesión.

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

Nota: El método **session** es un alias del método **get**.

El método **getAll** devuelve todas las variables de sesión:

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

20.4 Pruebas

El método **exists** comprueba la existencia de una variable en sesión.

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

El método **isStarted** comprueba el inicio de sesión

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

20.5 Borrar variables

El método **delete** elimina una variable de sesión:

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

20.6 Cierre explícito de la sesión

El método **terminar** cierra la sesión correctamente y borra todas las variables de sesión creadas:

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

CAPÍTULO 21

Cookie

Nota: Para todas las funciones Http, Ubiquity utiliza clases técnicas que contienen métodos estáticos. Se trata de una elección de diseño para evitar la inyección de dependencias que degradaría el rendimiento.

La clase **UCookie** proporciona funcionalidad adicional para manipular más fácilmente el array nativo **$_COOKIES** php.

21.1 Creación o modificación de cookies

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

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

Crear un cookie que dure 5 días:

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

En un dominio concreto:

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

Envío de una cookie sin codificación url del valor de la cookie:

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

Probando la creación de cookies:

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

21.2 Recuperar una cookie

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

21.2.1 Comprobación de existencia

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

21.2.2 Utilizar un valor por defecto

Si la cookie de página no existe, se devuelve el valor por defecto de 1:

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

21.3 Borrar una cookie

Eliminación de la cookie con el nombre **page**:

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

21.4 Eliminar todas las cookies

Eliminar todas las cookies de todo el dominio:

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

Eliminación de todas las cookies del dominio **admin**:

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

CAPÍTULO 22

Views

Ubiquity uses Twig as the default template engine (see Twig documentation). The views are located in the **app/views** folder. They must have the **.html** extension for being interpreted by Twig.

Ubiquity can also be used with a PHP view system, to get better performance, or simply to allow the use of php in the views.

22.1 Loading

Views are loaded from controllers:

Lista 1: app/controllers/Users.php

```php
namespace controllers;

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

22.1.1 Default view loading

If you use the default view naming method : The default view associated to an action in a controller is located in `views/controller-name/action-name` folder:

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

Lista 2: app/controllers/Users.php

```php
namespace controllers;

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

22.2 Loading and passing variables

Variables are passed to the view with an associative array. Each key creates a variable of the same name in the view.

Lista 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]);
            }
    }
}
```

In this case, it is usefull to call Compact for creating an array containing variables and their values :

Lista 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 Displaying in view

The view can then display the variables:

Lista 5: users/display.html

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

Variables may have attributes or elements you can access, too.

You can use a dot (.) to access attributes of a variable (methods or properties of a PHP object, or items of a PHP array), or the so-called «subscript» syntax ([]):

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

22.4 Ubiquity extra functions

Global app variable provides access to predefined Ubiquity Twig features:

- app is an instance of Framework and provides access to public methods of this class.

Get framework installed version:

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

Return the active controller and action names:

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

Return global wrapper classes :

For request:

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

For session :

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

see Framework class in API for more.

22.5 PHP view loading

Disable if necessary Twig in the configuration file by deleting the **templateEngine** key.

Then create a controller that inherits from `SimpleViewController`, or `SimpleViewAsyncController` if you use **Swoole** or **Workerman**:

Lista 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 }
```

Nota: In this case, the functions for loading assets and themes are not supported.

CAPÍTULO 23

Assets

Assets correspond to javascript files, style sheets, fonts, images to include in your application. They are located from the **public/assets** folder. It is preferable to separate resources into sub-folders by type.

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

Integration of css or js files :

```
{{ 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 with extra parameters:

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

CAPÍTULO 24

Themes

Nota: The themes are totally useless if you only have one presentation to apply.

Ubiquity support themes wich can have it's own assets and views according to theme template to be rendered by controller. Each controller action can render a specific theme, or they can use the default theme configured at *config.php* file in `templateEngineOptions => array("activeTheme" => "semantic")`.

Ubiquity is shipped with 3 default themes : **Bootstrap**, **Foundation** and **Semantic-UI**.

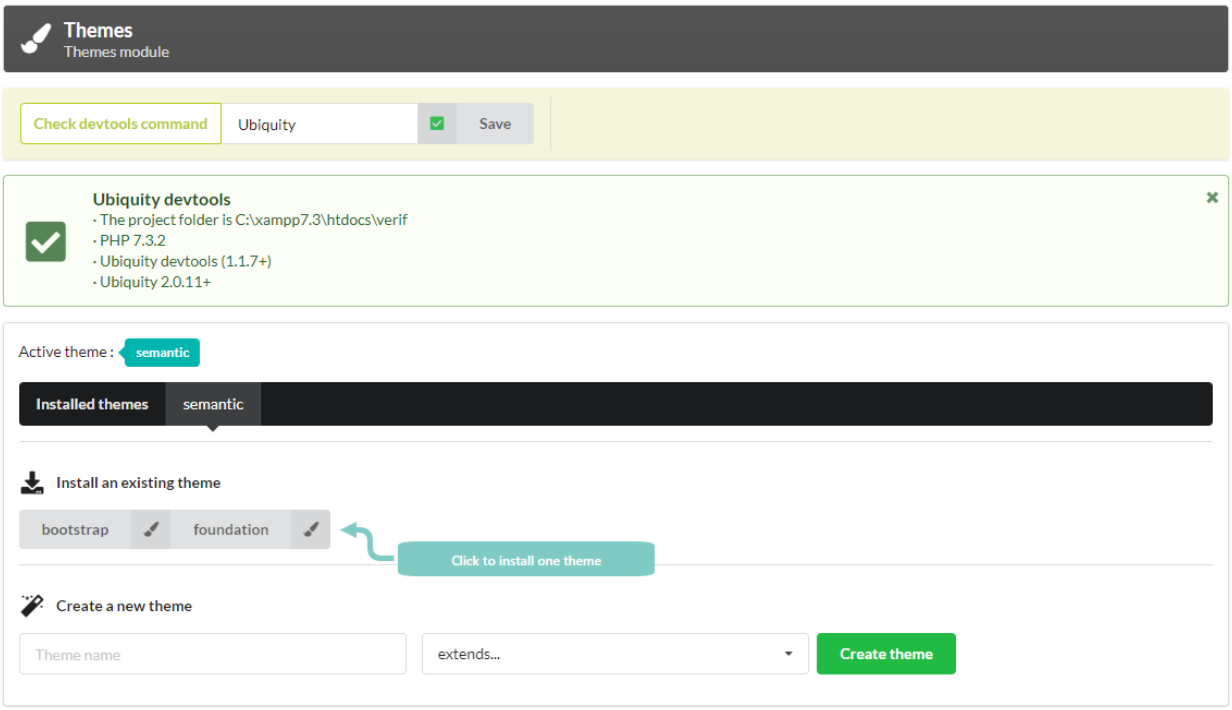
24.1 Installing a theme

With devtools, run :

```
Ubiquity install-theme bootstrap
```

The installed theme is one of **bootstrap**, **foundation** or **semantic**.

With **webtools**, you can do the same, provided that the **devtools** are installed and accessible (Ubiquity folder added in the system path) :



24.2 Creating a new theme

With devtools, run :

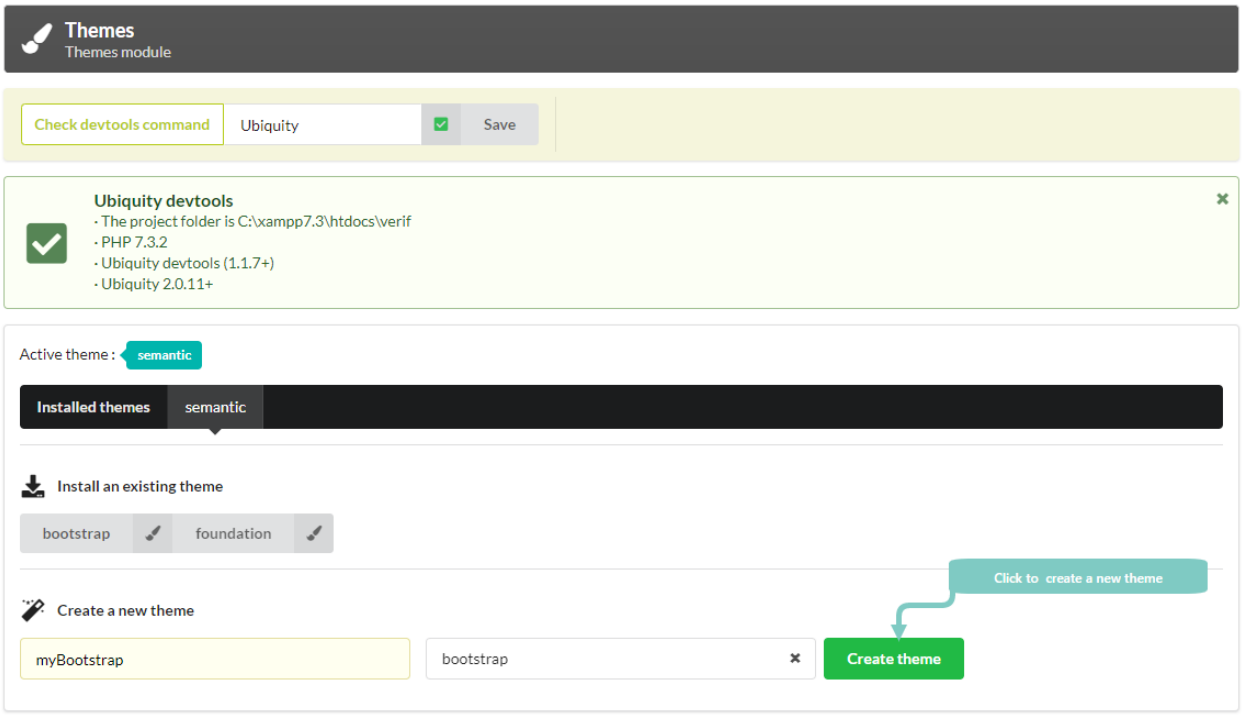
```
Ubiquity create-theme myTheme
```

Creating a new theme from Bootstrap, Semantic...

With devtools, run :

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

With **webtools** :



24.3 Theme functioning and structure

24.3.1 Structure

Theme view folder

The views of a theme are located from the **app/views/themes/theme-name** folder

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

The controller base class is responsible for loading views to define the header and footer of each page :

Lista 1: app/controllers/ControllerBase.php

```
1  <?php
2  namespace controllers;
3
4  use Ubiquity\controllers\Controller;
5  use Ubiquity\utils\http\URequest;
6
```

(continué en la próxima página)

(proviene de la página anterior)

```
    /**
     * 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 );
                    }
            }
    }
```

Theme assets folder

The assets of a theme are created inside `public/assets/theme-name` folder.

The structure of the assets folder is often as follows :

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

24.4 Change of the active theme

24.4.1 Persistent change

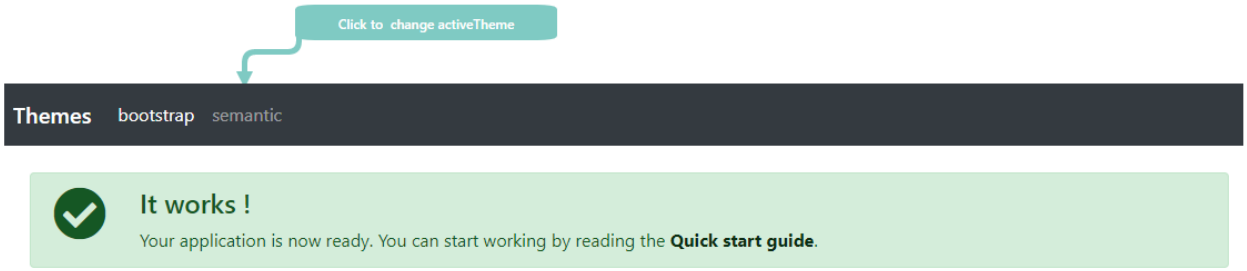
activeTheme is defined in `app/config/config.php` with `templateEngineOptions => array("activeTheme" => "semantic")`

The active theme can be changed with **devtools** :

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

It can also be done from the home page, or with **webtools** :

From the home page :



From the webtools :

This change can also be made at runtime :

From a controller :

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

24.4.2 Non-persistent local change

To set a specific theme for all actions within a controller, the simplest method is to override the controller's **initialize** method :

Lista 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     }
```

Or if the change should only concern one action :

Lista 3: app/controllers/Users.php

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

(continué en la próxima página)

(proviene de la página anterior)

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

Conditional theme change, regardless of the controller :

Example with a modification of the theme according to a variable passed in the URL

Lista 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 Mobile device support

Add a mobile device detection tool. Installing MobileDetect:

```
composer require mobiledetect/mobiledetectlib
```

It is generally easier to create different views per device.

Create a specific theme for the mobile part (by creating a folder `views/themes/mobile` and putting the views specific to mobile devices in it). It is important in this case to use the same file names for the mobile and non-mobile part.

It is also advisable in this case that all view loadings use the **@activeTheme** namespace:

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

index.html must be available in this case in the folders `views` and `views/themes/mobile`.

Global mobile detection (from services.php)

Lista 5: app/config/services.php

```
use Ubiquity\themes\ThemesManager;

...

ThemesManager::onBeforeRender(function () {
    $mb = new \Mobile_Detect();
    if ($mb->isMobile()) {
            ThemesManager::setActiveTheme('mobile');
```

(continué en la próxima página)

(proviene de la página anterior)

```
    }
});
```

Locale detection (from a controller)

Lista 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 View and assets loading

24.5.1 Views

For loading a view from the **activeTheme** folder, you can use the **@activeTheme** namespace :

Lista 7: app/controllers/Users.php

```
namespace controllers;

class Users extends BaseController{

        public function action(){
            $this->loadView('@activeTheme/action.html');
            ...
        }
    }
```

If the **activeTheme** is **bootstrap**, the loaded view is `app/views/themes/bootstrap/action.html`.

24.5.2 DefaultView

If you follow the Ubiquity view naming model, the default view loaded for an action in a controller when a theme is active is : `app/views/themes/theme-name/controller-name/action-name.html`.

For example, if the activeTheme is bootstrap, the default view for the action display in the Users controller must be loacated in `app/views/themes/bootstrap/Users/display.html`.

Lista 8: app/controllers/Users.php

```
namespace controllers;

class Users extends BaseController{

        public function display(){
            $this->loadDefaultView();
            ...
        }
    }
```

Nota: The devtools commands to create a controller or an action and their associated view use the **@activeTheme** folder if a theme is active.

```
Ubiquity controller Users -v

Ubiquity action Users.display -v
```

24.6 Assets loading

The mechanism is the same as for the views : `@activeTheme` namespace refers to the `public/assets/theme-name/` folder

```
{{ css('@activeTheme/css/style.css') }}

{{ js('@activeTheme/js/scripts.js') }}

{{ img('@activeTheme/img/image-name.png', {alt: 'Image Alt Name', class: 'css-class'}) }}
```

If the **bootstrap** theme is active, the assets folder is `public/assets/bootstrap/`.

24.7 Css compilation

For Bootstrap or foundation, install sass:

```
npm install -g sass
```

Then run from the project root folder:

For bootstrap:

```
ssass public/assets/bootstrap/scss/app.scss public/assets/bootstrap/css/style.css --load-
↪path=vendor
```

For foundation:

```
ssass public/assets/foundation/scss/app.scss public/assets/foundation/css/style.css --
↪load-path=vendor
```

CAPÍTULO 25

jQuery Semantic-UI

Por defecto, Ubiquity utiliza la librería phpMv-UI para la parte experiencia de usuario. **PhpMv-UI** permite crear componentes basados en Semantic-UI o Bootstrap y generar scripts jQuery en PHP.

Esta biblioteca se utiliza para la interfaz de administración de **webtools**.

25.1 Integración

Por defecto, se inyecta una variable **$jquery** en los controladores en tiempo de ejecución.

Esta operación se realiza mediante inyección de dependencias, en `app/config.php`:

Lista 1: app/config.php

```
...
"di"=>array(
            "@exec"=>array(
                            "jquery"=>function ($controller){
                                    return \Ajax\php\ubiquity\JsUtils::diSemantic(
↪$controller);
                                    }
                            )
            )
...
```

Así que no hay nada que hacer, pero para facilitar su uso y permitir la finalización de código en un controlador, se recomienda añadir la siguiente documentación de código:

Lista 2: app/controllers/FooController.php

```
/**
* Controller FooController
```

(continué en la próxima página)

(proviene de la página anterior)

```
 * @property \Ajax\php\ubiquity\JsUtils $jquery
 **/
class FooController extends ControllerBase{

	public function index(){}
}
```

25.2 jQuery

25.2.1 Referencias (Href) a solicitudes ajax

Cree un nuevo Controlador y su vista asociada, luego defina las siguientes rutas:

Lista 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 vista asociada:

Lista 4: app/views/FooController/index.html

```
	<a href="{{path('action.a')}}">Action a</a>
	<a href="{{path('action.b')}}">Action b</a>
```

Inicializar la caché del router:

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

Pruebe esta página en su navegador en `http://127.0.0.1:8090/FooController`.

Transformación de peticiones en peticiones Ajax

El resultado de cada petición ajax debe mostrarse en un área de la página definida por su selector jQuery (`.result span`)

Lista 5: app/controllers/FooController.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");
    }
    ...
}
```

Lista 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 }}
```

Nota: La variable `script_foot` contiene el script jquery generado por el método **renderView**. El filtro **raw** marca el valor como «safe», lo que significa que en un entorno con escape automático activado esta variable no será escapada.

Añadamos un poco de css para hacerlo más profesional:

Lista 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 queremos añadir un nuevo enlace cuyo resultado deba mostrarse en otra zona, es posible especificarlo mediante el atributo **data-target**.

La nueva acción:

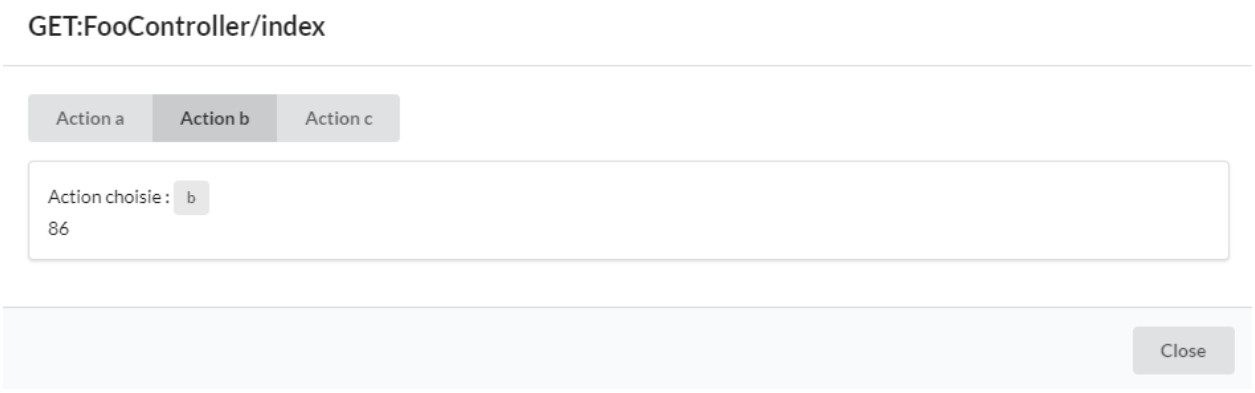
Lista 8: app/controllers/FooController.php

```
namespace controllers;

class FooController extends ControllerBase {
     ...
     /**
      *@get("c","name"=>"action.c")
      */
     public function cAction() {
             echo \rand(0, 1000);
     }
}
```

La vista asociada:

Lista 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 }}
```



Definición de los atributos de la petición ajax:

En el siguiente ejemplo, los parámetros pasados a la variable **attributes** del método `getHref`:

- eliminar el historial de navegación,
- hacer que el cargador ajax sea interno al botón pulsado.

Lista 10: app/controllers/FooController.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");
    }
    ...
}
```

Nota: Es posible utilizar el método `postHref` para utilizar el método http **POST**.

25.2.2 Peticiones ajax clásicas

Para este ejemplo, cree la siguiente base de datos:

```
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;
```

Conectar la aplicación a la base de datos, y generar la clase *User*:

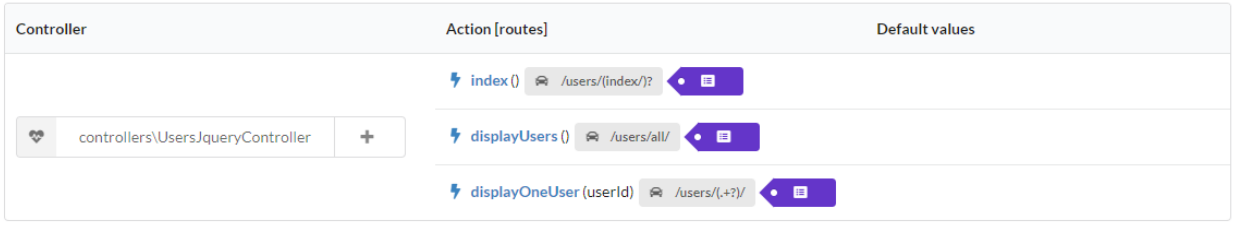
Con devtools:

```
Ubiquity config:set --database.dbName uguide
Ubiquity all-models
```

Crear un nuevo controlador *UsersJqueryController*.

```
Ubiquity controller UsersJqueryController -v
```

Crea las siguientes acciones en *UsersJqueryController*:



Acción index

La acción *index* debe mostrar un botón para obtener la lista de usuarios, cargada mediante una petición ajax:

Lista 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 vista por defecto asociada a la acción *index*:

Lista 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 }}
```

acción displayUsers

Se muestran todos los usuarios, y un clic en un usuario debe mostrar los detalles del usuario a través de una solicitud ajax publicada:

Lista 13: app/controllers/UsersJqueryController.php

```
namespace controllers;

/**
 * Controller UsersJqueryController
 *
 * @property \Ajax\php\ubiquity\JsUtils $jquery
 * @route("users")
 */
class UsersJqueryController extends ControllerBase {
...
    /**
     *
     * @get("all","name"=>"display.users","cache"=>true)
     */
    public function displayUsers() {
            $users = DAO::getAll(User::class);
            $this->jquery->click('#close-bt', '$("#users").html("");');
            $this->jquery->postOnClick('li[data-ajax]', Router::path('display.one.user',
↪ [
                    ""
            ]), '{}', '#user-detail', [
                    'attr' => 'data-ajax',
                    'hasLoader' => false
            ]);
            $this->jquery->renderDefaultView([
                    'users' => $users
            ]);
    }
```

La vista asociada a la acción *displayUsers*:

Lista 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 }}
```

acción displayOneUser

Lista 15: app/controllers/UsersJqueryController.php

```
namespace controllers;

/**
 * Controller UsersJqueryController
 *
 * @property \Ajax\php\ubiquity\JsUtils $jquery
 * @route("users")
 */
class UsersJqueryController extends ControllerBase {
...
     /**
      *
      * @post("{userId}","name"=>"display.one.user","cache"=>true,"duration"=>3600)
      */
     public function displayOneUser($userId) {
             $user = DAO::getById(User::class, $userId);
             $this->jquery->hide('#users-content', '', '', true);
             $this->jquery->click('#close-user-bt', '$("#user-detail").html("");$("
↪#users-content").show();');
             $this->jquery->renderDefaultView([
                     'user' => $user
             ]);
     }
```

La vista asociada a la acción *displayOneUser*:

Lista 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 Componentes semantic

A continuación, vamos a hacer un controlador implementando las mismas funcionalidades que antes, pero utilizando componentes **PhpMv-UI** (parte semántica).

25.3.1 HtmlButton ejemplo

Crear un nuevo controlador *UsersJqueryController*.

```
Ubiquity controller UsersCompoController -v
```

Lista 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      }
```

(continué en la próxima página)

(proviene de la página anterior)

```
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      }
```

Nota: Al llamar a renderView o renderDefaultView sobre el objeto JQuery se realiza la compilación del componente, y se genera el HTML y JS correspondientes.

La vista asociada integra el componente de botón con la matriz *q* disponible en la vista :

Lista 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 ++++++++++++++++++

CAPÍTULO 26

Normalizers (Normalizadores)

Nota: El módulo Normalizer utiliza la clase estática **NormalizersManager** para gestionar la normalización.

CAPÍTULO 27

Validators (Validadores)

Nota: El módulo Validators utiliza la clase estática **ValidatorsManager** para gestionar la validación.

Los validadores se utilizan para comprobar que los datos de los miembros de un objeto cumplen ciertas restricciones.

27.1 Añadir validadores

O bien la clase **Author** que queremos utilizar en nuestra aplicación :

Lista 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 }
```

Hemos añadido una restricción de validación en el miembro **name** con la anotación **@validator**, para que no esté vacío.

27.2 Generar caché

Ejecute este comando en modo consola para crear los datos de caché de la clase **Author** :

```
Ubiquity init-cache -t=models
```

La caché del validador se genera en `app/cache/contents/validators/models/Author.cache.php`.

27.3 Validación de instancias

27.3.1 una instancia

```
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 el **nombre** del autor está vacío, esta acción debe mostrarse:

```
name : This value should not be empty
```

El método **validate** devuelve una matriz de instancias **ConstraintViolation**.

27.3.2 instancias multiples

```
public function testValidateAuthors(){
        $authors=DAO::getAll(Author::class);
        $violations=ValidatorsManager::validateInstances($author);
        foreach($violations as $violation){
                echo $violation.'<br>';
        }
}
```

27.4 Generación de modelos con validadores por defecto

Cuando las clases se generan automáticamente a partir de la base de datos, se asocian validadores por defecto a los miembros, en función de los metadatos de los campos.

```
Ubiquity create-model User
```

Lista 2: app/models/Author.php

```php
namespace models;
class User{
        /**
         * @id
         * @column("name"=>"id","nullable"=>false,"dbType"=>"int(11)")
         * @validator("id","constraints"=>array("autoinc"=>true))
        **/
        private $id;

        /**
         * @column("name"=>"firstname","nullable"=>false,"dbType"=>"varchar(65)")
         * @validator("length","constraints"=>array("max"=>65,"notNull"=>true))
        **/
        private $firstname;

        /**
         * @column("name"=>"lastname","nullable"=>false,"dbType"=>"varchar(65)")
         * @validator("length","constraints"=>array("max"=>65,"notNull"=>true))
        **/
        private $lastname;

        /**
         * @column("name"=>"email","nullable"=>false,"dbType"=>"varchar(255)")
         * @validator("email","constraints"=>array("notNull"=>true))
         * @validator("length","constraints"=>array("max"=>255))
        **/
        private $email;

        /**
         * @column("name"=>"password","nullable"=>true,"dbType"=>"varchar(255)")
         * @validator("length","constraints"=>array("max"=>255))
        **/
        private $password;

        /**
         * @column("name"=>"suspended","nullable"=>true,"dbType"=>"tinyint(1)")
         * @validator("isBool")
        **/
        private $suspended;
}
```

Estos validadores pueden ser modificados. Las modificaciones siempre deben ir seguidas de una reinicialización de la caché del modelo.

```
Ubiquity init-cache -t=models
```

La información sobre la validación de los modelos puede visualizarse con devtools :

```
Ubiquity info:validation -m=User
```

```
· The project folder is C:\xampp7.3\htdocs\verif
```

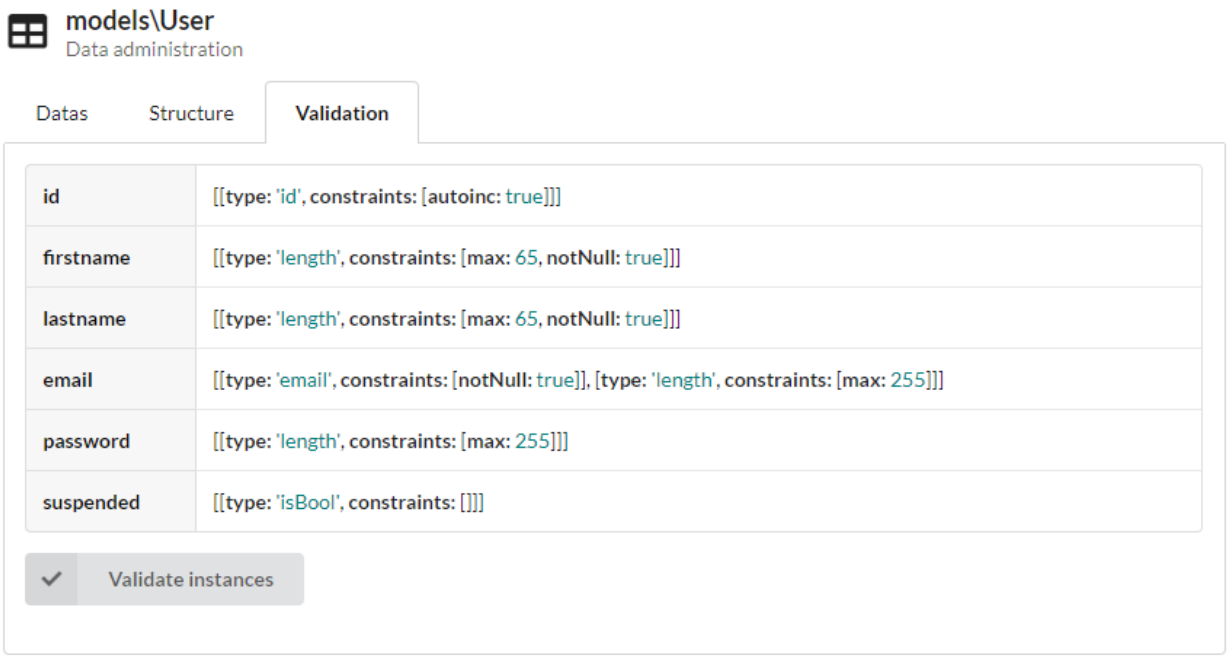
field	value
id	· [type: 'id',constraints: [autoinc: true]]
firstname lastname	· [type: 'length',constraints: [max: 65,notNull: true]] ≡
email	· [type: 'email',constraints: [notNull: true]] · [type: 'length',constraints: [max: 255]]
password	· [type: 'length',constraints: [max: 255]]
suspended	· [type: 'isBool',constraints: []]

Obtener validadores en campo email:

```
Ubiquity info:validation email -m=User
```

```
· email
  · type : 'email'
  · constraints : [notNull: true]
  · type : 'length'
  · constraints : [max: 255]
```

También se puede acceder a la información de validación desde la parte **modelos** de las webtools:



27.5 Tipos de validadores

27.5.1 Basic

Validador	Roles	Restricciones	Valores aceptados
isBool	Verifica si el valor es un booleano		true,false,0,1
isEmpty	Verifica si el valor está vacío		“”,null
isFalse	Verifica si el valor es falso		false,”false”,0,”0”
isNull	Verifica si el valor es nulo		null
isTrue	Verifica si el valor es verdadero		true,”true”,1,”1”
notEmpty	Verifica si el valor no está vacío		!null && !””
notNull	Verifica si el valor no es nulo		!null
tipo	Verifica si el valor es del tipo {type}	{type}	

27.5.2 Comparación

27.5.3 Fechas

27.5.4 Múltiples

27.5.5 Cadenas

CAPÍTULO 28

Transformers

Nota: El módulo Transformers utiliza la clase estática **TransformersManager** para gestionar las transformaciones de datos.

Los transformers se utilizan para transformar los datos después de cargarlos desde la base de datos o antes de mostrarlos en una vista.

28.1 Añadir transformers

O bien la clase **Author** que queremos utilizar en nuestra aplicación :

Atributos

Lista 1: app/models/Author.php

```
1  namespace models;
2
3  use Ubiquity\attributes\items\Transformer;
4
5  class Author {
6
7     #[Transformer('upper')]
8     private $name;
9
10    public function getName(){
11       return $this->name;
12    }
13
14    public function setName($name){
15       $this->name=$name;
```

(continué en la próxima página)

(proviene de la página anterior)

```
    }
}
```

Annotations (Anotaciones)

Lista 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;
    }
}
```

Añadimos un transformer en el miembro **name** con la anotación **@transformer**, para poner el nombre en mayúsculas en las vistas.

28.2 Generar caché

Ejecute este comando en modo consola para crear los datos de caché de la clase **Author** :

```
Ubiquity init-cache -t=models
```

La caché del transformer se genera con metadatos del modelo en `app/cache/models/Author.cache.php`.

La información de los transformers puede visualizarse con devtools :

```
Ubiquity info:model -m=Author -f=#transformers
```

field	value
#transformers	· toView : [name: 'Ubiquity\\contents\\transformation\\transformers\\UpperCase']

28.3 Uso de transformers

Inicie el **TransformersManager** en el archivo *app/config/services.php*:

Lista 3: app/config/services.php

```
\Ubiquity\contents\transformation\TransformersManager::startProd();
```

Puede comprobar el resultado en la interfaz de administración:

Id	Name
1	JOHN GRISHAM
2	JOANNE ROWLING
3	STEPHEN EDWIN KING

o creando un controlador:

Lista 4: app/controllers/Authors.php

```
namespace controllers;

class Authors {

   public function index(){
      DAO::transformersOp='toView';
      $authors=DAO::getAll(Author::class);
      $this->loadDefaultView(['authors'=>$authors]);
   }

}
```

Lista 5: app/views/Authors/index.html

```
<ul>
   {% for author in authors %}
      <li>{{ author.name }}</li>
   {% endfor %}
</ul>
```

28.4 Tipos de transformers

28.4.1 transform

El tipo **transform** se basa en la interfaz **TransformerInterface**. Se utiliza cuando los datos transformados deben convertirse en un objeto. |brl El transformador **DateTime** es un buen ejemplo de este tipo de transformador:

- Al cargar los datos, el Transformador convierte la fecha de la base de datos en una instancia de php DateTime.
- Su método **reverse** realiza la operación inversa (fecha php a fecha compatible con la base de datos).

28.4.2 toView

El tipo **toView** se basa en la interfaz **TransformerViewInterface**. Se utiliza cuando los datos transformados deben mostrarse en una vista.

28.4.3 toForm

El tipo **toForm** se basa en la interfaz **TransformerFormInterface**. Se utiliza cuando los datos transformados deben utilizarse en un formulario.

28.5 Uso de transformers

28.5.1 Transformación en la carga de datos

Si se omite, **transformerOp** por defecto es **transform**.

```
$authors=DAO::getAll(Author::class);
```

Establecer transformerOp a **toView**

```
DAO::transformersOp='toView';
$authors=DAO::getAll(Author::class);
```

28.5.2 Transformación tras la carga

Devuelve el valor transformado del miembro:

```
TransformersManager::transform($author, 'name','toView');
```

Devuelve un valor transformado:

```
TransformersManager::applyTransformer($author, 'name','john doe','toView');
```

Transforma una instancia aplicando todos los transformers definidos:

```
TransformersManager::transformInstance($author,'toView');
```

28.6 Transformers existentes

Transformer	Tipo(s)	Descripción
datetime	transform, toView, toForm	Transformar una fecha y hora de una base de datos en un objeto DateTime de php
upper	toView	Poner el valor en mayúsculas
lower	toView	Poner el valor en minúsculas
firstUpper	toView	Poner en mayúsculas el primer carácter del valor
password	toView	Enmascarar los caracteres
md5	toView	Hashear el valor con md5

28.7 Cree personalizado

28.7.1 Creación

Crear un transformador para mostrar un nombre de usuario como una dirección de correo electrónico local:

Lista 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 Registro

Registre el transformador ejecutando el siguiente script:

```
TransformersManager::registerClassAndSave('localEmail',\transformers\
↪ToLocalEmail::class);
```

28.7.3 Uso

Atributos

Lista 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 (Anotaciones)

Lista 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();
```

El nombre de usuario **Smith** aparecerá como **smith@mydomain.local**.

CAPÍTULO 29

Translation module

Nota: The Translation module uses the static class **TranslatorManager** to manage translations.

29.1 Module structure

Translations are grouped by **domain**, within a **locale** :

In the translation root directory (default **app/translations**):

- Each locale corresponds to a subfolder.
- For each locale, in a subfolder, a domain corresponds to a php file.

```
translations
     ├ en_EN
     │     ├ messages.php
     │     └ blog.php
     └ fr_FR
           ├ messages.php
           └ blog.php
```

- each domain file contains an associative array of translations **key-> translation value**
- **Each key can be associated with**
    - a translation
    - a translation containing variables (between **%** and **%**)
    - an array of translations for handle pluralization

Lista 1: app/translations/en_EN/messages.php

```
return [
     'okayBtn'=>'Okay',
     'cancelBtn'=>'Cancel',
     'deleteMessage'=>['No message to delete!','1 message to delete.','%count% messages␣
↪to delete.']
];
```

29.2 Starting the module

Module startup is logically done in the **services.php** file.

Lista 2: app/config/services.php

```
1 Ubiquity\cache\CacheManager::startProd($config);
2 Ubiquity\translation\TranslatorManager::start();
```

With no parameters, the call of the **start** method uses the locale **en_EN**, without fallbacklocale.

Importante: The translations module must be started after the cache has started.

29.2.1 Setting the locale

Changing the locale when the manager starts:

Lista 3: app/config/services.php

```
1 Ubiquity\cache\CacheManager::startProd($config);
2 Ubiquity\translation\TranslatorManager::start('fr_FR');
```

Changing the locale after loading the manager:

```
TranslatorManager::setLocale('fr_FR');
```

29.2.2 Setting the fallbackLocale

The **en_EN** locale will be used if **fr_FR** is not found:

Lista 4: app/config/services.php

```
Ubiquity\cache\CacheManager::startProd($config);
Ubiquity\translation\TranslatorManager::start('fr_FR','en_EN');
```

29.3 Defining the root translations dir

If the **rootDir** parameter is missing, the default directory used is `app/translations`.

Lista 5: app/config/services.php

```
Ubiquity\cache\CacheManager::startProd($config);
Ubiquity\translation\TranslatorManager::start('fr_FR','en_EN','myTranslations');
```

29.4 Make a translation

29.4.1 With php

Translation of the **okayBtn** key into the default locale (specified when starting the manager):

```
$okBtnCaption=TranslatorManager::trans('okayBtn');
```

With no parameters, the call of the **trans** method uses the default locale, the domain **messages**.

Translation of the **message** key using a variable:

```
$okBtnCaption=TranslatorManager::trans('message',['user'=>$user]);
```

In this case, the translation file must contain a reference to the **user** variable for the key **message**:

Lista 6: app/translations/en_EN/messages.php

```
['message'=>'Hello %user%!',...];
```

29.4.2 In twig views:

Translation of the **okayBtn** key into the default locale (specified when starting the manager):

```
{{ t('okayBtn') }}
```

Translation of the **message** key using a variable:

```
{{ t('message',parameters) }}
```

CAPÍTULO 30

Security

30.1 Guiding principles

30.1.1 Forms validation

Client-side validation

It is preferable to perform an initial client-side validation to avoid submitting invalid data to the server.

Example of the creation of a form in the action of a controller (this part could be located in a dedicated service for a better separation of layers):

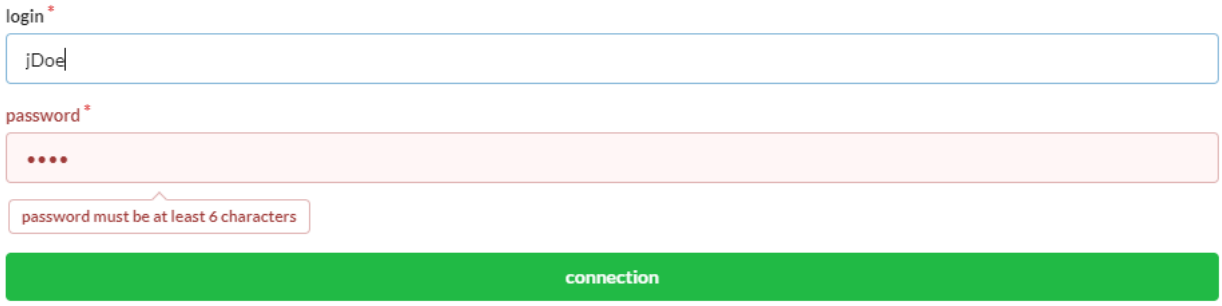
Lista 1: app/controllers/UsersManagement.php

```
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();
}
```

The Associated View:

Lista 2: app/views/UsersManagement/index.html

```
{{ q['frm-user'] | raw }}
{{ script_foot | raw }}
<div id="response"></div>
```



Nota: The CRUD controllers automatically integrate this client-side validation using the Validators attached to the members of the models.

```
#[Column(name: "password",nullable: true,dbType: "varchar(255)")]
#[Validator(type: "length",constraints: ["max"=>20,"min"=>6])]
#[Transformer(name: "password")]
private $password;
```

Server-side validation

It is preferable to restrict the URLs allowed to modify data. Beforehand, by specifying the Http method in the routes, and by testing the request :

```
#[Post(path: "/submit")]
public function submitUser(){
   if(!URequest::isCrossSite() && URequest::isAjax()){
      $datas=URequest::getPost();//post with htmlEntities
      //Do something with $datas
   }
}
```

Nota: The **Ubiquity-security** module offers additional control to avoid cross-site requests.

After modifying an object, it is possible to check its validity, given the validators attached to the members of the associated Model:

```
#[Post(path: "/submit")]
public function submitUser(){
   if(!URequest::isCrossSite()){
      $datas=URequest::getPost();//post with htmlEntities
      $user=new User();
```

(continué en la próxima página)

(proviene de la página anterior)

```
        URequest::setValuesToObject($user,$datas);

        $violations=ValidatorsManager::validate($user);
        if(\count($violations)==0){
            //do something with this valid user
        } else {
            //Display violations...
        }
    }
}
```

30.1.2 DAO operations

It is always recommended to use parameterized queries, regardless of the operations performed on the data:

- To avoid SQL injections.
- To allow the use of prepared queries, speeding up processing.

```
$googleUsers=DAO::getAll(User::class,'email like ?',false,['%@gmail.com']);
```

```
$countActiveUsers=DAO::count(User::class,'active= ?',[true]);
```

Nota: DAO operations that take objects as parameters use this mechanism by default.

```
DAO::save($user);
```

30.1.3 Passwords management

The `Password` Transformer allows a field to be of the password type when displayed in an automatically generated CRUD form.

```
#[Transformer(name: "password")]
private $password;
```

After submission from a form, it is possible to encrypt a password from the URequest class:

```
$encryptedPassword=URequest::password_hash('password');
$user->setPassword($encryptedPassword);
DAO::save($user);
```

The algorithm used in this case is defined by the php `PASSWORD_DEFAULT`.

It is also possible to check a password entered by a user in the same way, to compare it to a hash:

```
if(URequest::password_verify('password', $existingPasswordHash)){
    //password is ok
}
```

Importante: Set up Https to avoid sending passwords in clear text.

30.2 Security module/ ACL management

In addition to these few rules, you can install if necessary:

- *Ubiquity-acl*
- *Ubiquity-security*

CAPÍTULO 31

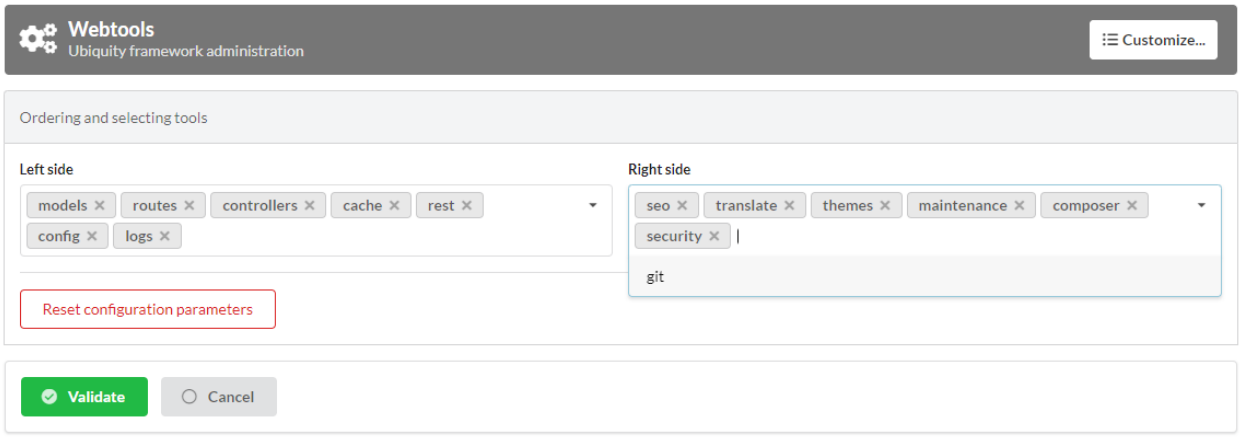
Security module

31.1 Installation

Install the Ubiquity-security module from the command prompt or from the **Webtools** (Composer part).

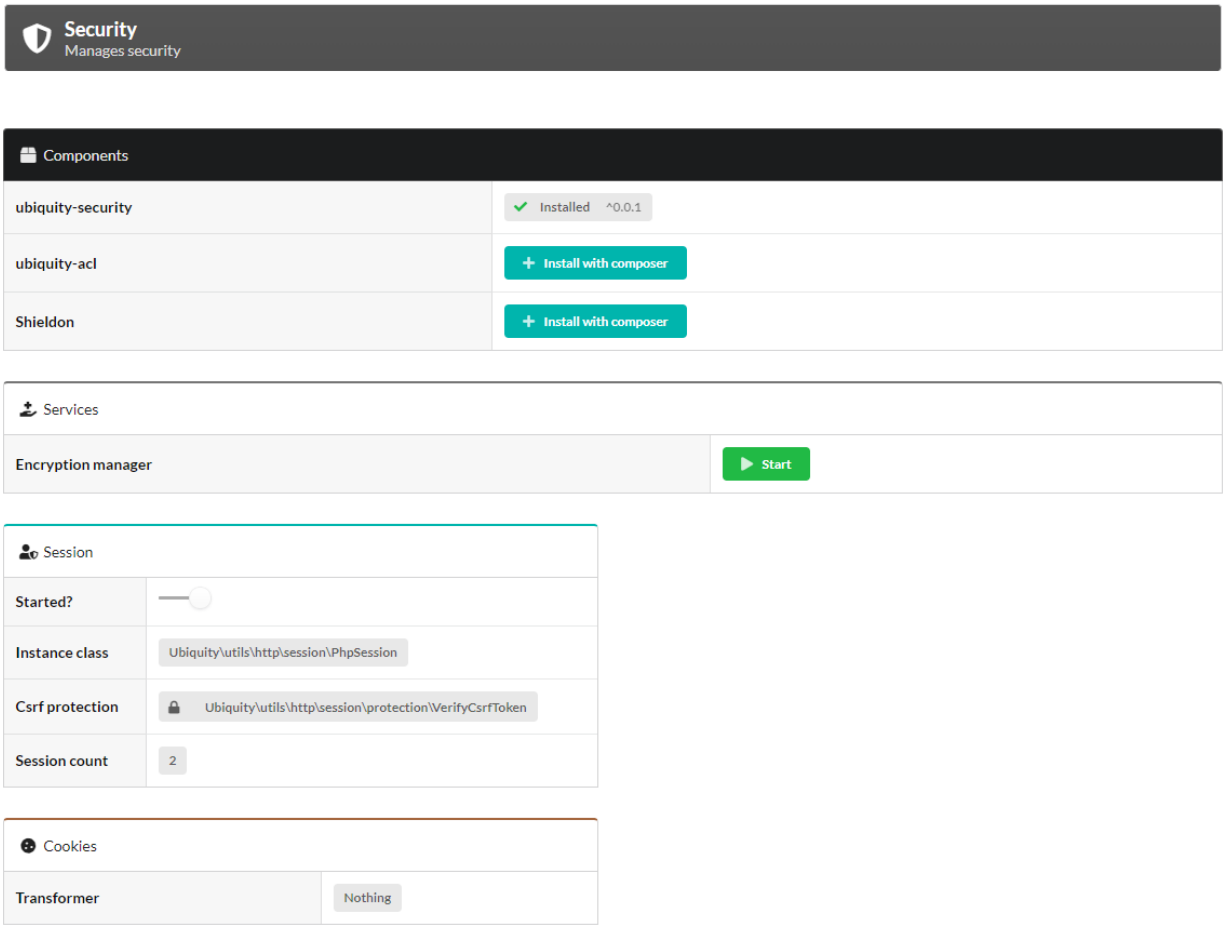
```
composer require phpmv/ubiquity-security
```

Then activate the display of the Security part in the **Webtools**:



31.2 Session CSRF

The session is by default protected against CSRF attacks via the `VerifyCsrfToken` class (even without the **Ubiquity-security** module). A token instance (`CSRFToken`) is generated at the session startup. The validity of the token is then checked via a cookie at each request.



This protection can be customized by creating a class implementing the `VerifySessionCsrfInterface`.

Lista 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
```

(continué en la próxima página)

(proviene de la página anterior)

```
    }

    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
    }
}
```

Starting the custom protection in services:

Lista 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 Deactivating the protection

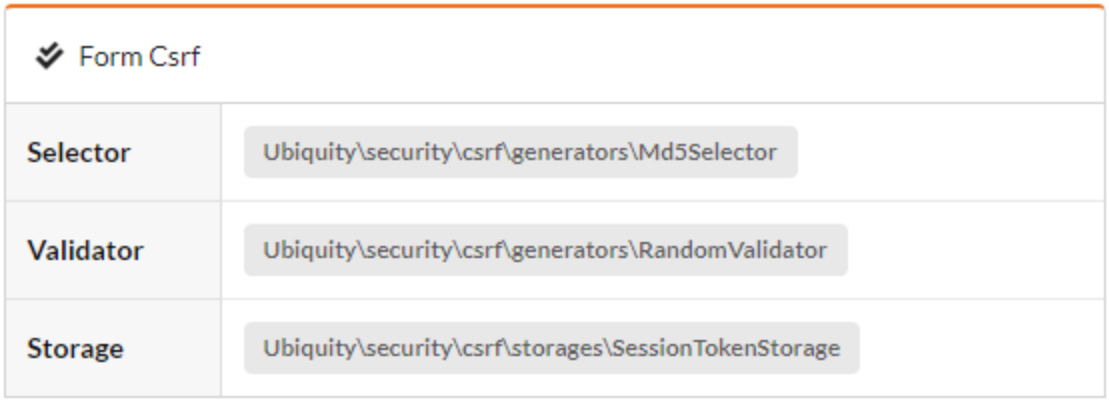
If you do not need to protect your session against Csrf attacks, start the session with the `NoCsrfProtection` class.

Lista 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 CSRF manager

The **CsrfManager** service can be started directly from the **webtools** interface. Its role is to provide tools to protect sensitive routes from Csrf attacks (the ones that allow the validation of forms for example).



- The service is started in the `services.php` file.

Lista 4: app/config/services.php

```
\Ubiquity\security\csrf\CsrfManager::start();
```

31.3.1 Example of form protection:

The form view:

```
<form id="frm-bar" action='/submit' method='post'>
   {{ csrf('frm-bar') }}
   <input type='text' id='sensitiveData' name='sensitiveData'>
</form>
```

The `csrf` method generates a token for the form (By adding a hidden field in the form corresponding to the token.).

The form submitting in a controller:

```
use Ubiquity\security\csrf\UCsrfHttp;

#[Post('/submit')]
public function submit(){
   if(UCsrfHttp::isValidPost('frm-bar')){
      //Token is valid! => do something with post datas
   }
}
```

Nota: It is also possible to manage this protection via cookie.

31.3.2 Example of protection with ajax:

The meta field `csrf-token` is generated on all pages.

Lista 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] );
      }
   }
}
```

This field is added in the headerView:

Lista 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 %}
```

Example with a button posting data via ajax. The parameter `csrf` is set to true. So when the request is posted, the `csrf-token` is sent in the request headers.

```
#[Get(path: "/ajax")]
public function ajax(){
   $this->jquery->postOnClick('#bt','/postAjax','{id:55}','#myResponse',['csrf'=>true]);
   $this->jquery->renderDefaultView();
}
```

The submitting route can check the presence and validity of the token:

```
#[Post(path: "postAjax")]
public function postAjax(){
   if(UCsrfHttp::isValidMeta('postAjax')){
      var_dump($_POST);
   }else{
      echo 'invalid or absent meta csrf-token';
   }
}
```

31.4 Encryption manager

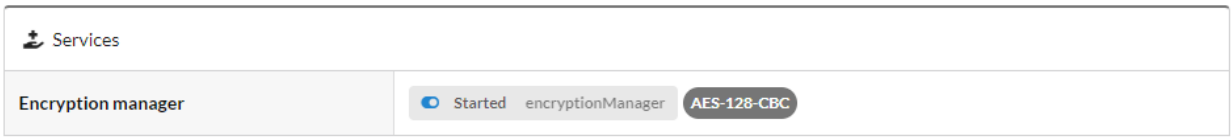
The **EncryptionManager** service can be started directly from the **webtools** interface.

- In this case, a key is generated in the configuration file `app/config/config.php`.
- The service is started in the `services.php` file.

Lista 7: app/config/services.php

```
\Ubiquity\security\data\EncryptionManager::start($config);
```

Nota: By default, encryption is performed in `AES-128`.



31.4.1 Changing the cipher:

Upgrade to AES-256:

Lista 8: app/config/services.php

```
\Ubiquity\security\data\EncryptionManager::startProd($config, Encryption::AES256);
```

Generate a new key:

```
Ubiquity new:key 256
```

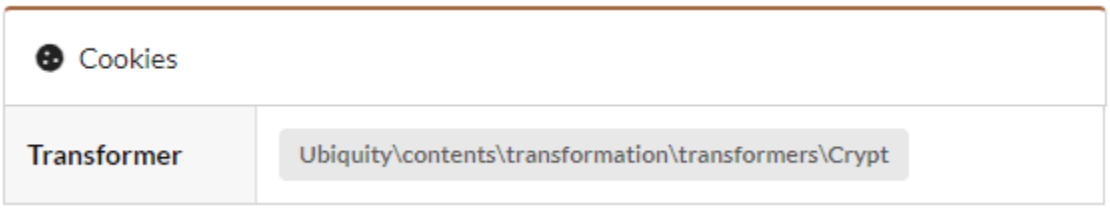
The new key is generated in the `app/config/config.php` file.

Cookie encryption

Cookies can be encrypted by default, by adding this in `services.php`:

Lista 9: app/config/services.php

```
use Ubiquity\utils\http\UCookie;
use Ubiquity\contents\transformation\transformers\Crypt;

UCookie::setTransformer(new Crypt());
```



Model data encryption

The `Crypt` transformer can also be used on the members of a model:

Lista 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
```

Generic Data encryption

Strings encryption:

```
$encryptedBar=EncryptionManager::encryptString('bar');
```

To then decrypt it:

```
echo EncryptionManager::decryptString($encryptedBar);
```

It is possible to encrypt any type of data:

```
$encryptedUser=EncryptionManager::encrypt($user);
```

To then decrypt it, with possible serialisation/deserialisation if it is an object:

```
$user=EncryptionManager::decrypt($encryptedUser);
```

31.5 Content Security Policies manager

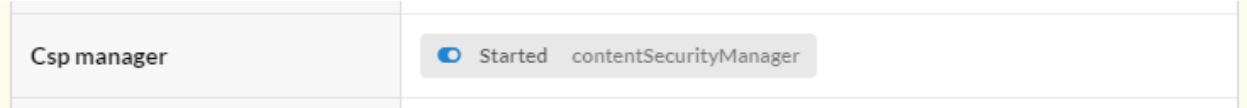
The **ContentSecurityManager** service can be started directly from the **webtools** interface.

- The service is started in the `services.php` file.

Lista 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();
    }
});
```

Nota: With this default configuration, a nonce is added to jquery scripts generated with phpmv-ui. CSP control is done in Report-only mode..



31.5.1 Adding a nonce

Example of adding nonce on the header and footer pages:

Updating the base controller

Lista 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]);
		}
	}
}
```

Adding the nonce in the header and footer views

Lista 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 %}
```

Lista 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 Password management

31.7 Users token

CAPÍTULO 32

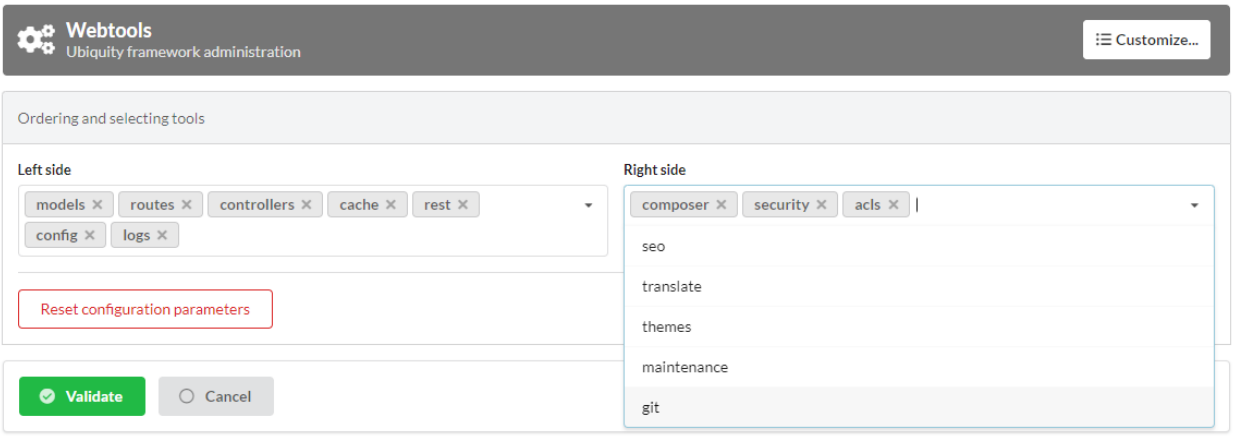
ACL management

32.1 Installation

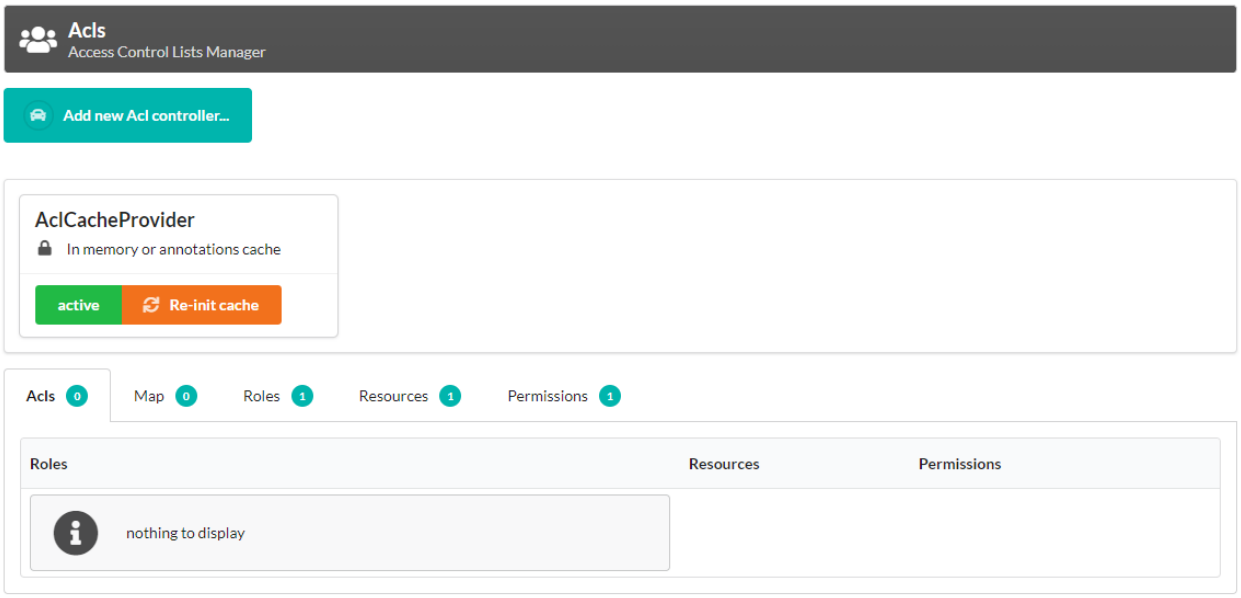
Install the **Ubiquity-acl** module from the command prompt or from the **Webtools** (Composer part).

```
composer require phpmv/ubiquity-acl
```

Then activate the display of the Acl part in the **Webtools**:



ACL interface in **webtools**:



32.2 Acl Rules

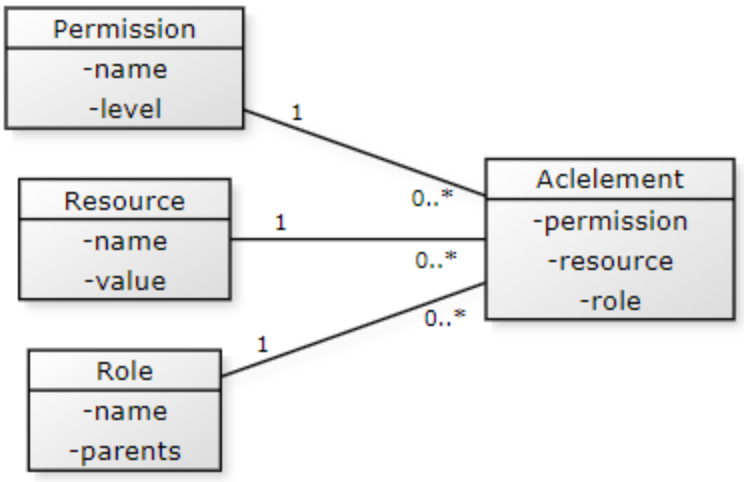
ACLs are used to define access to an Ubiquity application. They are defined according to the following principles:

An Ubiquity application is composed of :

- **Resources** (possibly controllers, or actions of these controllers)
- **Roles**, possibly assigned to users. Each **Role** can inherit parent roles.
- **Permissions**, which correspond to a right to do. Each permission has a level (represented by an integer value).

Additional rules:

- An AclElement (**Allow**) grants Permission to a Role on a Resource.
- Each role inherits authorisations from its parents, in addition to its own.
- If a role has a certain level of access permission on a resource, it will also have all the permissions of a lower level on that resource.
- The association of a resource and a permission to a controller or a controller action defines a **map** element.



Naming tips:

- Role, in capital letters, beginning with an arobase (@USER, @ADMIN, @ALL...).
- Permissions, in upper case, named using a verb (READ, WRITE, OPEN...).
- Resource, capitalized on the first letter (Products, Customers...)

32.3 ACL Starting

The **AclManager** service can be started directly from the **webtools** interface, in the **Security** part.

- The service is started in the `services.php` file.

Lista 1: app/config/services.php

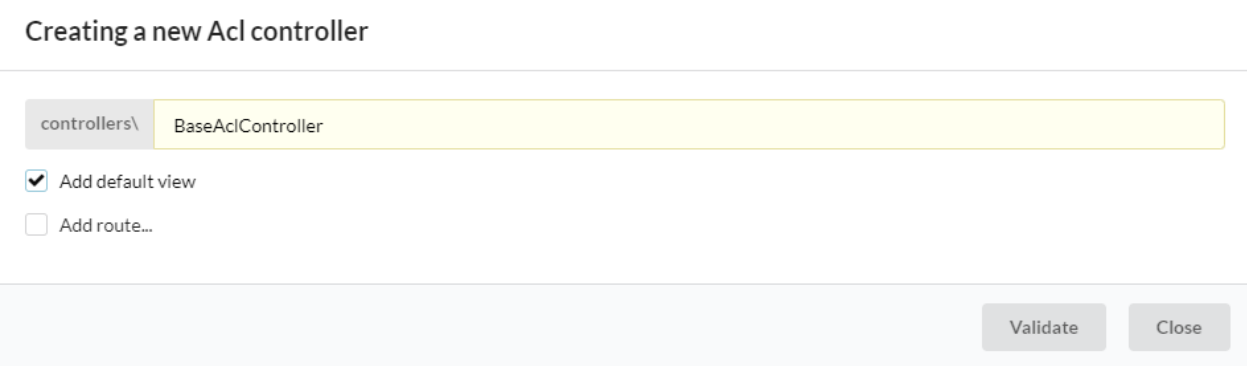
```
\Ubiquity\security\acl\AclManager::startWithCacheProvider();
```

32.3.1 ACLCacheProvider

This default provider allows you to manage ACLs defined through attributes or annotations.

AclController

An AclController enables automatic access management based on ACLs to its own resources. It is possible to create them automatically from **webtools**.



But it is just a basic controller, using the AclControllerTrait feature.

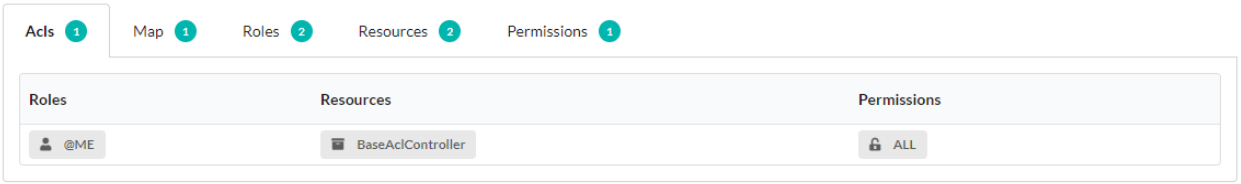
This controller just goes to redefine the `_getRole` method, so that it returns the role of the active user, for example.

Lista 2: app/controllers/BaseAclController.php

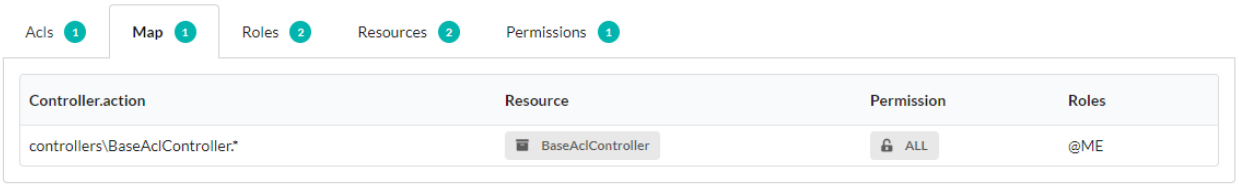
```php
<?php
namespace controllers;

use Ubiquity\controllers\Controller;
use Ubiquity\security\acl\controllers\AclControllerTrait;
use Ubiquity\attributes\items\acl\Allow;

class BaseAclController extends Controller {
use AclControllerTrait;

   #[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!';
   }
}
```

Authorisation has been granted for the resource:

- Without specifying the resource, the controller's actions are defined as a resource.
- Without specifying the permission, the `ALL` permission is used.



And this association is present in the Acls map:



AclController with authentication

Nota: The use of both `WithAuthTrait` and `AclControllerTrait` requires to remove the ambiguity about the `isValid` method.

Lista 3: app/controllers/BaseAclController.php

```
class BaseAclController extends Controller {
   use AclControllerTrait,WithAuthTrait{
      WithAuthTrait::isValid insteadof AclControllerTrait;
      AclControllerTrait::isValid as isValidAcl;
   }

   public function isValid($action){
        return parent::isValid($action)&& $this->isValidAcl($action);
   }
}
```

Allow with Role, resource and permission

Allow without prior creation:

`@USER` is allowed to access to `Foo` resource with `READ` permission.

Lista 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';
```

(continué en la próxima página)

(proviene de la página anterior)

```
    }
}
```

Nota: The role, resource and permission are automatically created as soon as they are invoked with `Allow`.

Allow with explicit creation:

Lista 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';
    }
}
```

Adding ACL at runtime

Whether in a controller or in a service, it is possible to add Roles, Resources, Permissions and Authorizations at runtime:

For example :\ Adding a Role `@USER` inheriting from `@GUEST`.

```
use Ubiquity\security\acl\AclManager;

AclManager::addRole('@GUEST');
AclManager::addRole('@USER',['@GUEST']);
```

Defining ACLs with Database

The ACLs defined in the database are additional to the ACLs defined via annotations or attributes.

32.3.2 Initializing

The initialization allows to create the tables associated to the ACLs (Role, Resource, Permission, AclElement). It needs to be done only once, and in dev mode only.

To place for example in `app/config/bootstrap.php` file:

```
use Ubiquity\controllers\Startup;
use Ubiquity\security\acl\AclManager;

```

(continué en la próxima página)

(proviene de la página anterior)

```
$config=Startup::$config;
AclManager::initializeDAOProvider($config, 'default');
```

32.3.3 Starting

In app/config/services.php file :

```
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 Strategies for defining ACLs

32.4.1 With few resources:

Defining authorisations for each controller's action or action group:

Resources logically correspond to controllers, and permissions to actions. But this rule may not be respected, and an action may be defined as a resource, as required.

The only mandatory rule is that a Controller/action pair can only correspond to one Resource/permission pair (not necessarily unique).

Lista 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';
   }
```

(continué en la próxima página)

(proviene de la página anterior)

```
    #[Allow('@USER')]
    public function read() {
        echo 'read';
    }

    #[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 With more resources:

Lista 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() {
```

(continué en la próxima página)

(proviene de la página anterior)

```
        echo 'read';
    }

    #[Permission('WRITE',3)]
    public function write() {
        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!';
    }

}
```

CAPÍTULO 33

Rest

El módulo REST implementa un CRUD básico, con un sistema de autenticación, directamente comprobable en la parte de administración.

33.1 REST y enrutamiento

El enrutador es esencial para el módulo REST, ya que REST (Respresentation State Transfer) se basa en URL y métodos HTTP.

Nota: Por razones de rendimiento, las rutas REST se almacenan en caché independientemente de otras rutas. Por lo tanto, es necesario iniciar el enrutador de una manera particular para activar las rutas REST y no obtener un error 404 recurrente.

El router se inicia en `services.php`.

Sin activación de rutas REST:

Lista 1: app/config/services.php

```
...
Router::start();
```

Para habilitar rutas REST en una aplicación que también tiene una parte no REST:

Lista 2: app/config/services.php

```
...
Router::startAll();
```

Para activar sólo las rutas Rest:

```
Router::startRest();
```

Es posible iniciar el enrutamiento condicionalmente (este método sólo será más eficiente si el número de rutas es grande en cualquiera de las partes):

Lista 3: app/config/services.php

```
...
    if($config['isRest']()){
            Router::startRest();
    }else{
            Router::start();
    }
```

33.2 Recursos REST

Un controlador REST puede asociarse directamente a un modelo.

Nota: Si no tiene a mano una base de datos mysql, puede descargar ésta: `messagerie.sql`

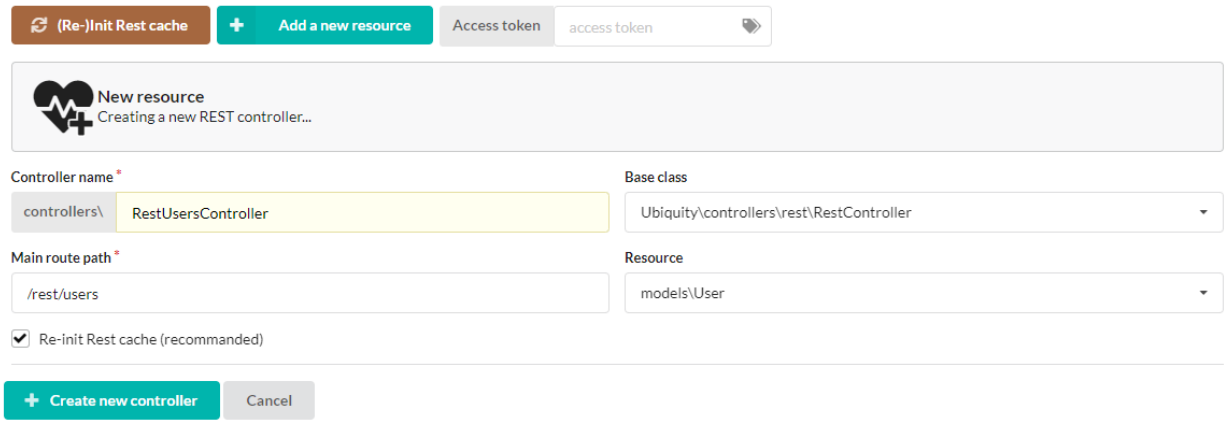
33.2.1 Creación

Con devtools:

```
Ubiquity rest RestUsersController -r=User -p=/rest/users
```

O con webtools:

Vaya a la sección **REST** y elija **Añadir un nuevo recurso**:



El controlador creado :

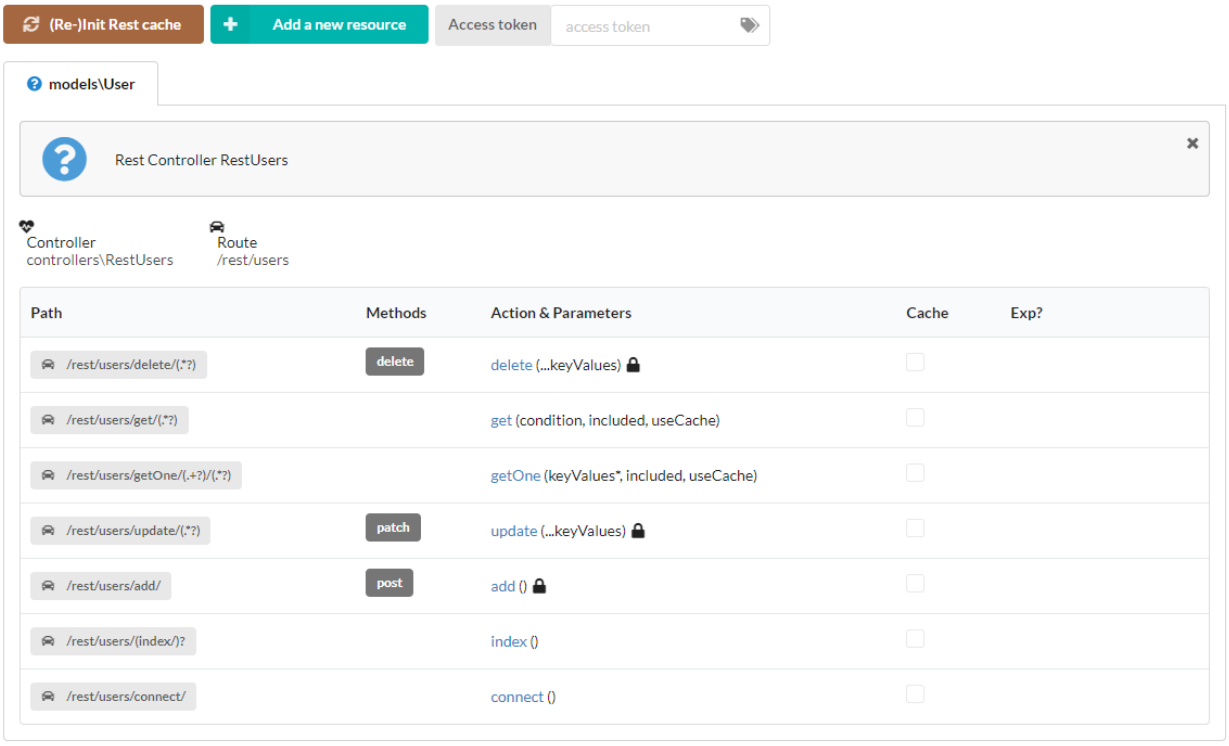
Lista 4: app/controllers/RestUsersController.php

```php
namespace controllers;

/**
 * Rest Controller RestUsersController
 * @route("/rest/users","inherited"=>true,"automated"=>true)
 * @rest("resource"=>"models\\User")
 */
class RestUsersController extends \Ubiquity\controllers\rest\RestController {

}
```

Dado que los atributos **automated** y **inherited** de la ruta están en true, el controlador tiene las rutas por defecto de la clase padre.

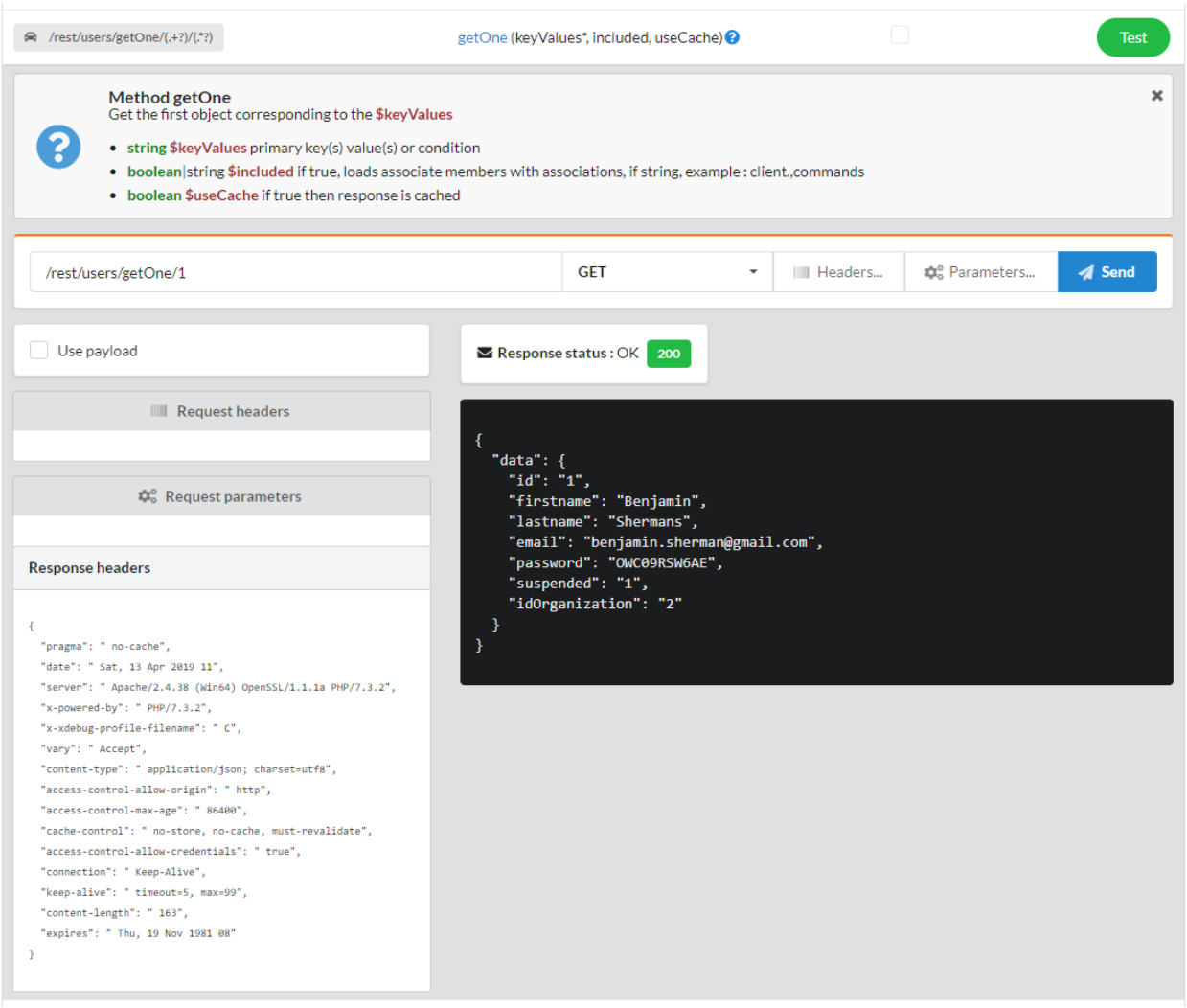
33.2.2 Interfaz de prueba

Webtools ofrecen una interfaz para consultar datos:

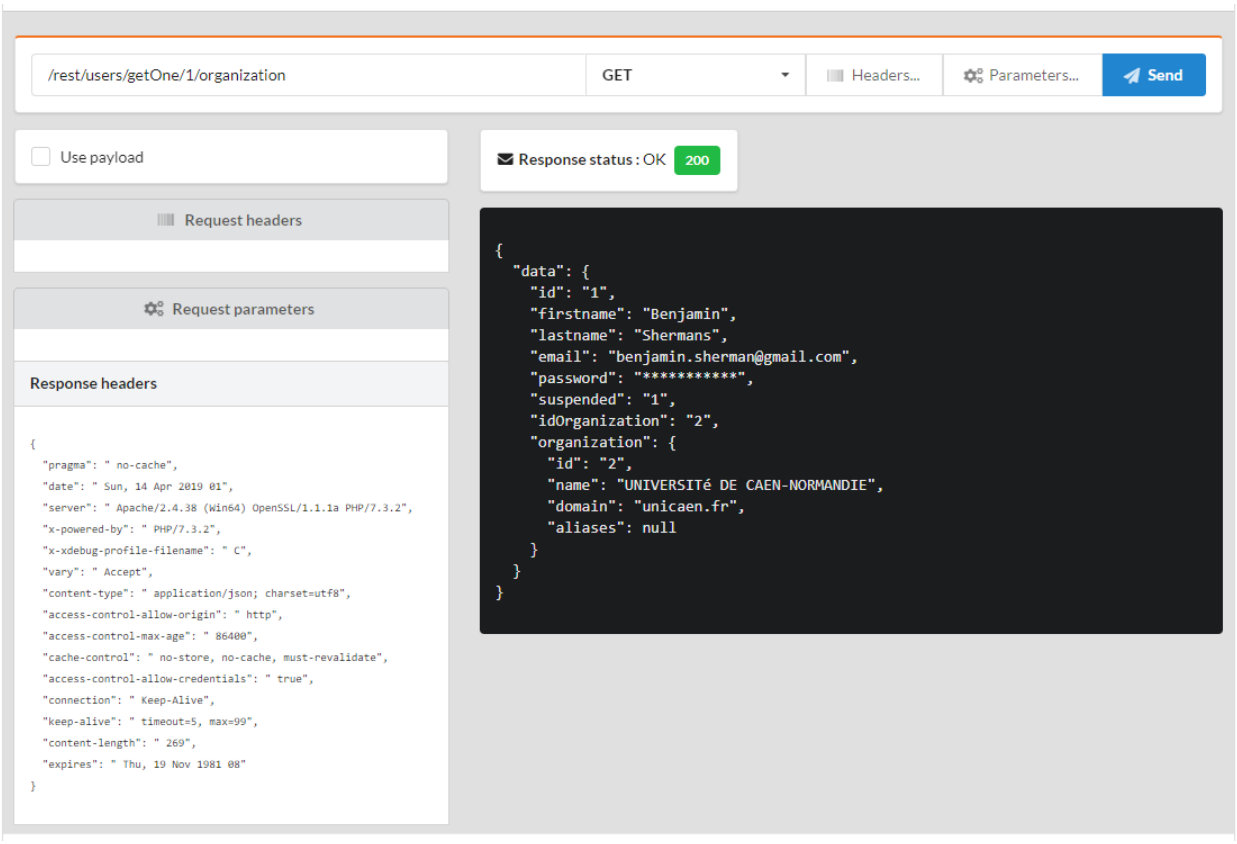


Obtener una instancia

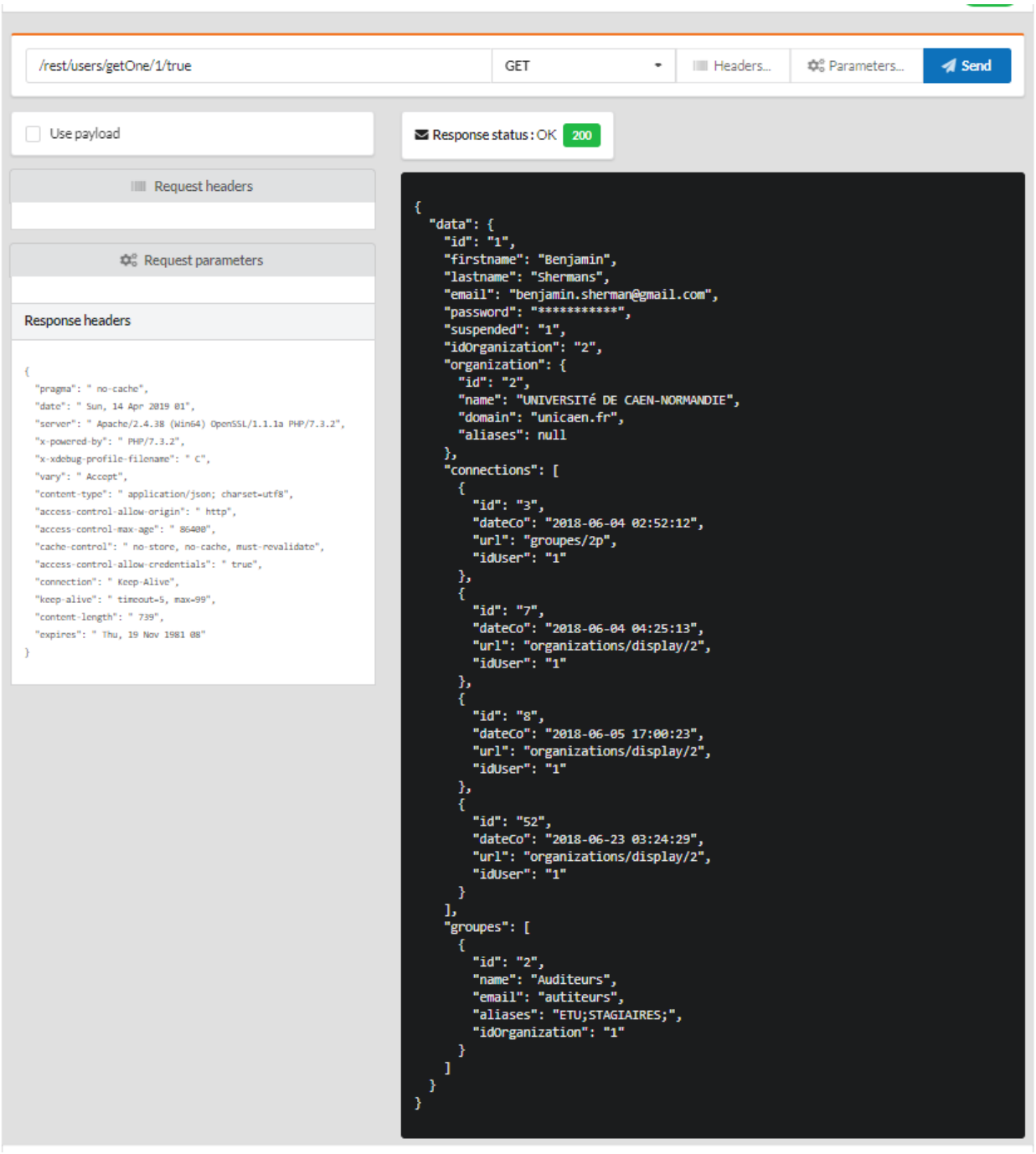
Se puede acceder a una instancia de usuario por su clave principal (**id**):



Inclusión de miembros asociados: la organización del usuario

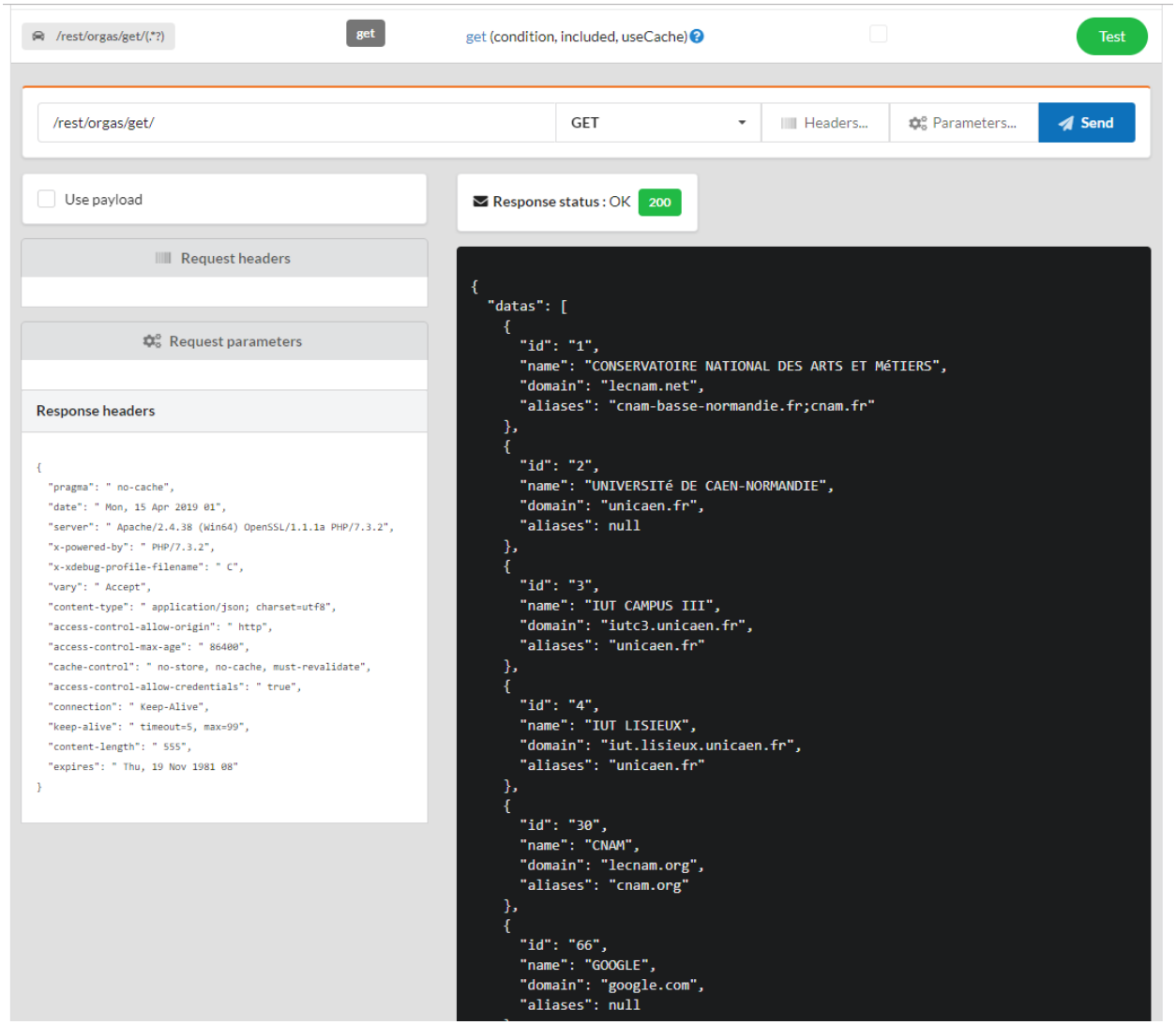


Inclusión de miembros asociados: organización, conexiones y grupos del usuario

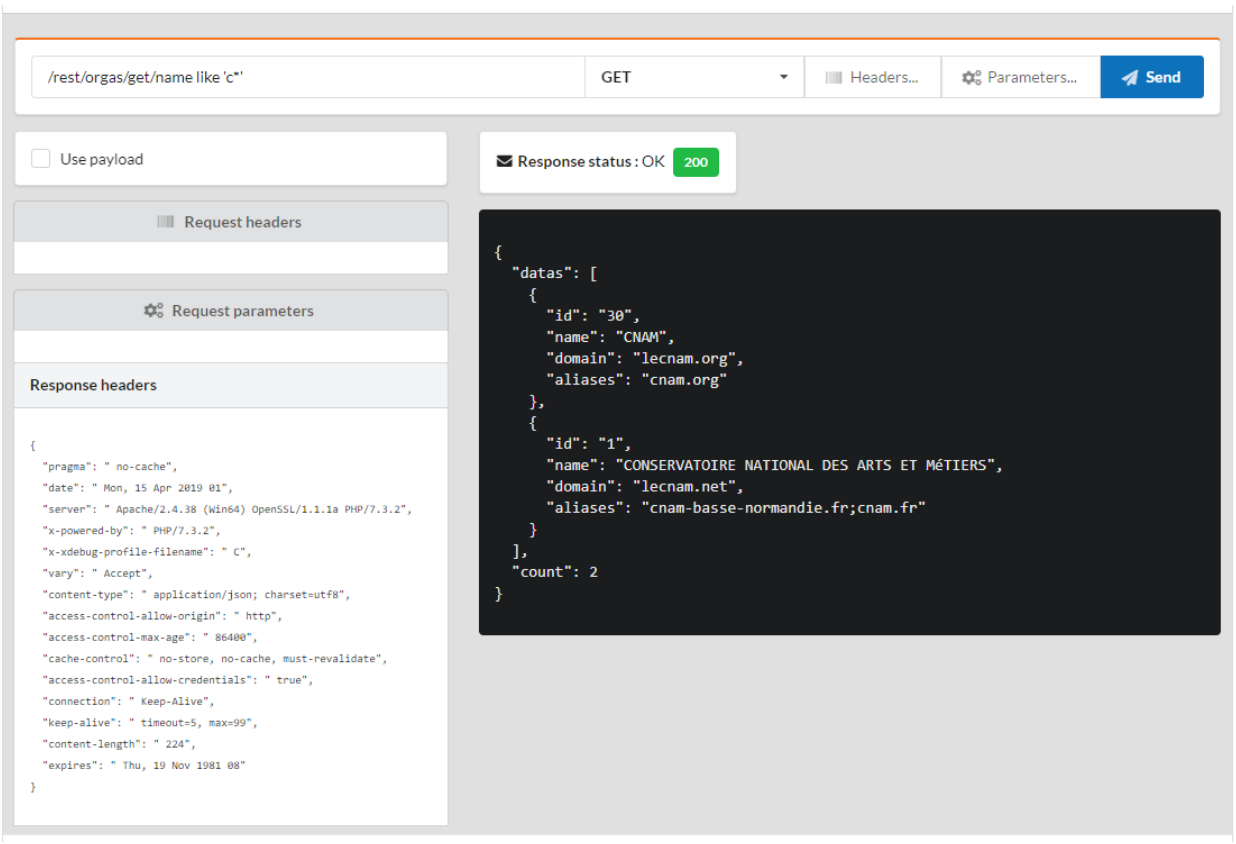

/rest/users/getOne/1/true
GET
Headers...
Parameters...
Send
Use payload
Response status : OK 200
Request headers
Request parameters
Response headers
{
"pragma": " no-cache",
"date": " Sun, 14 Apr 2019 01",
"server": " Apache/2.4.38 (Win64) OpenSSL/1.1.1a PHP/7.3.2",
"x-powered-by": " PHP/7.3.2",
"x-xdebug-profile-filename": " 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"
}
]
}
}

Obtener varias instancias

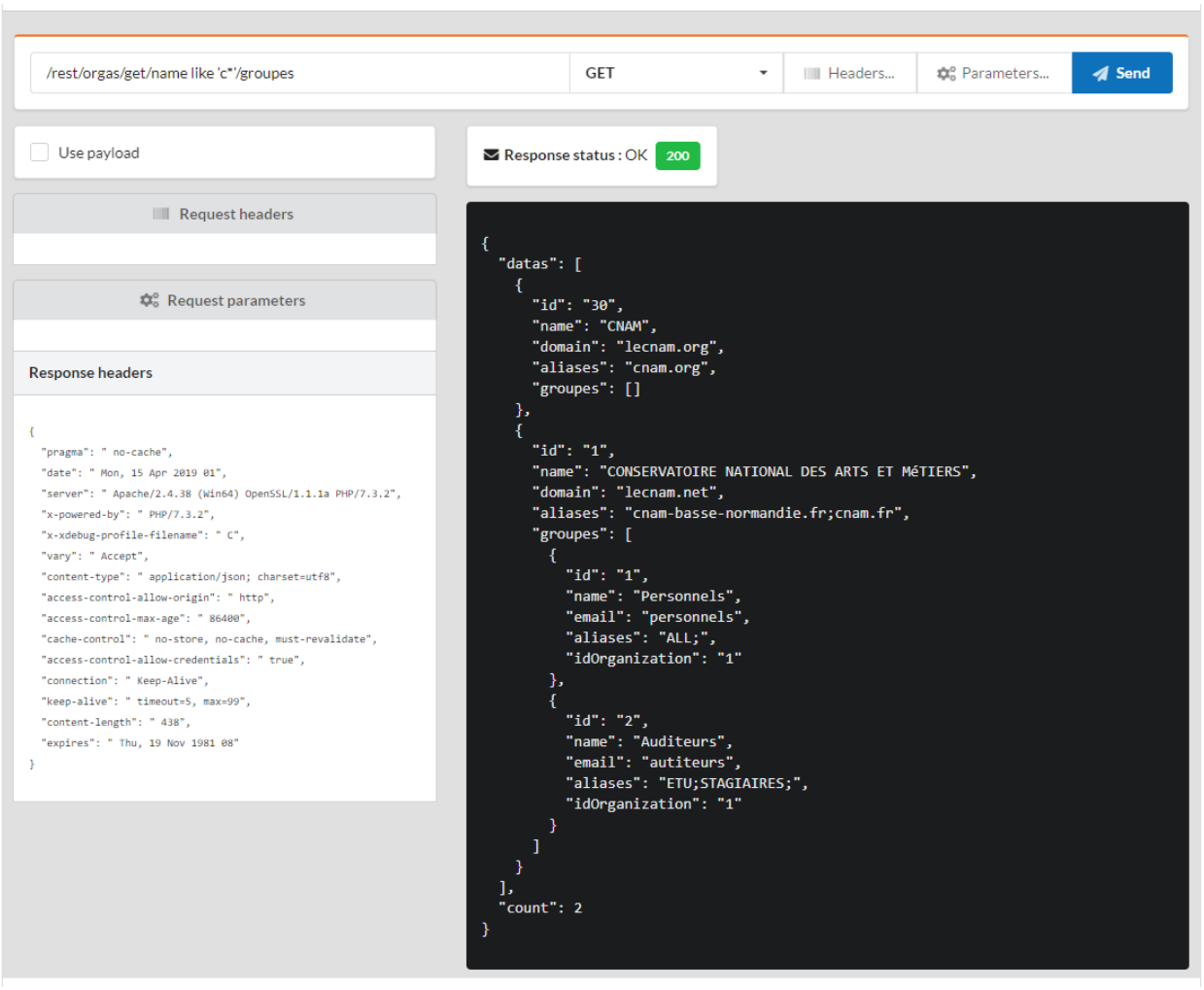
Obtener todas las instancias:



Establecer una condición:



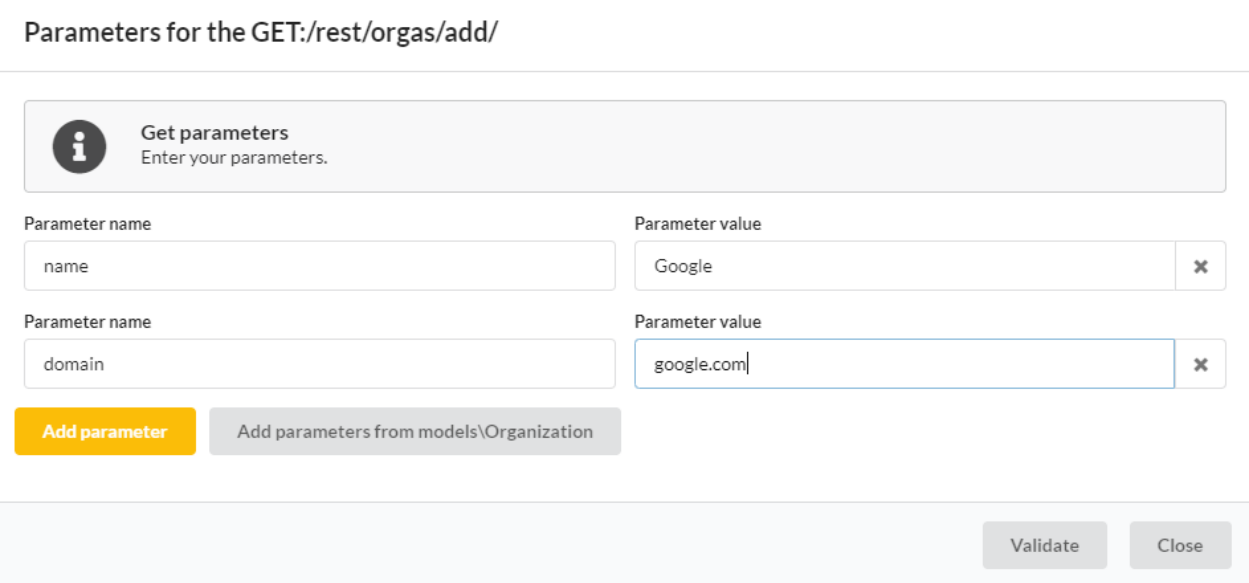
Incluidos los miembros asociados:



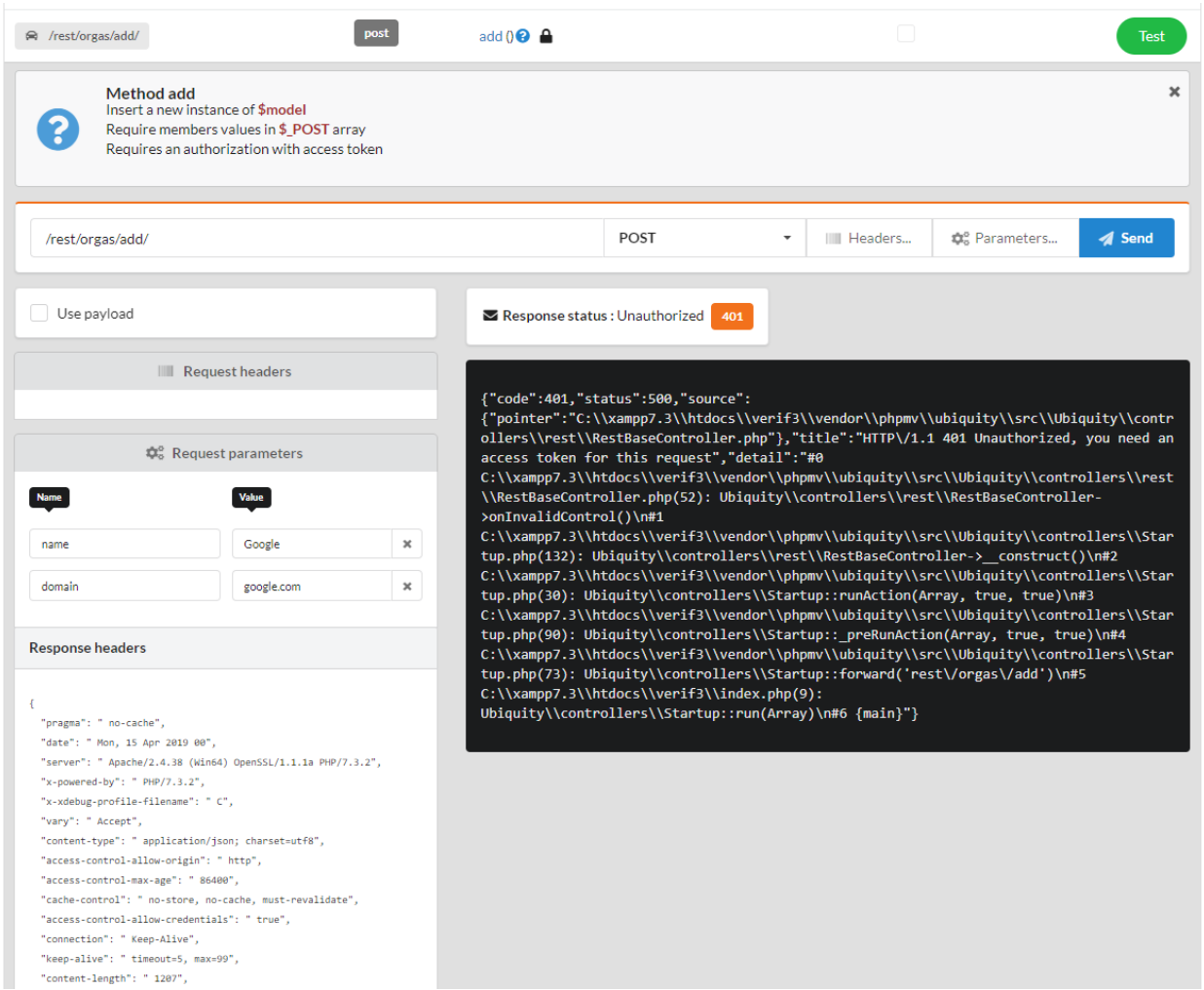
Añadir una instancia

Los datos se envían mediante el método **POST**, con un tipo de contenido definido en `application/x-www-form-urlencoded`:

Añada los parámetros de nombre y dominio pulsando el botón **parámetros**:

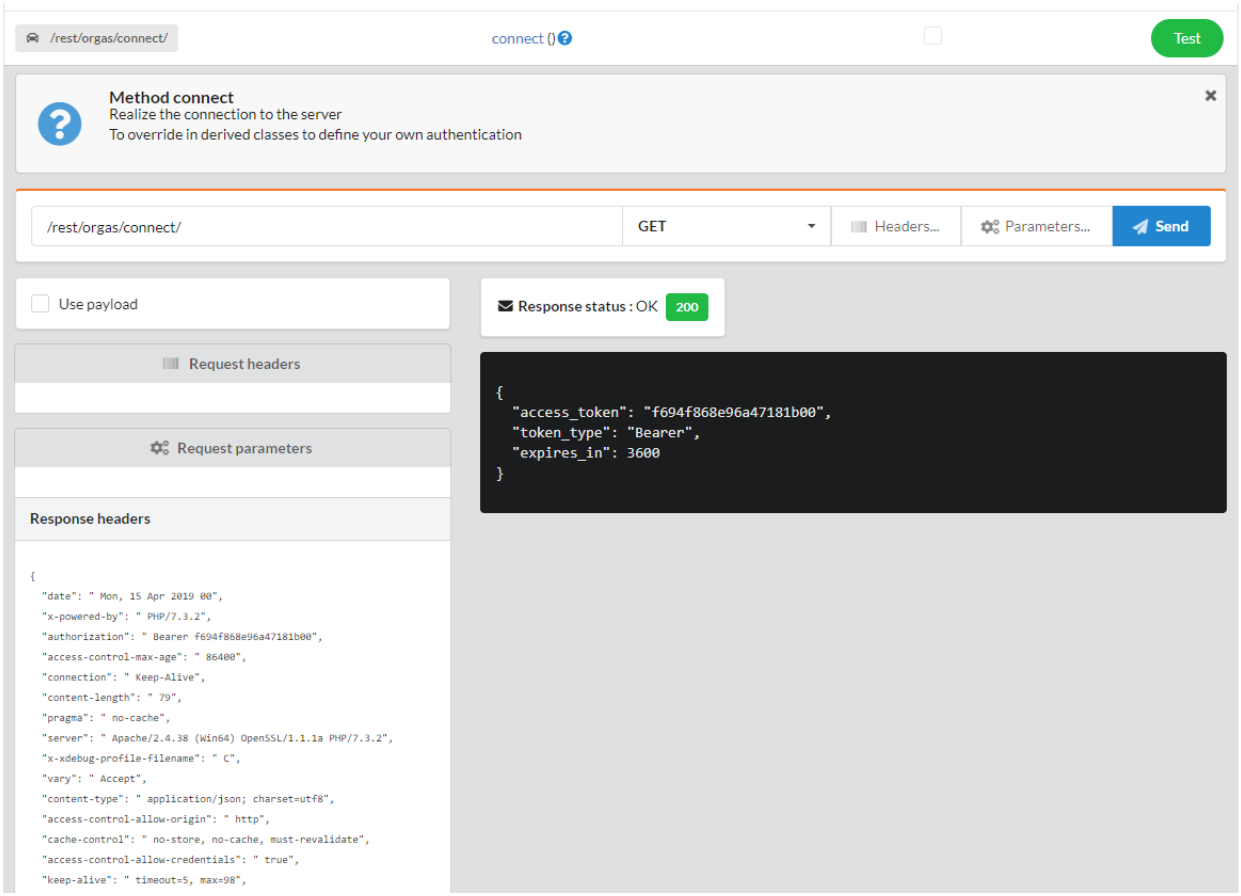


La adición requiere una autenticación, por lo que se genera un error, con el estado 401:

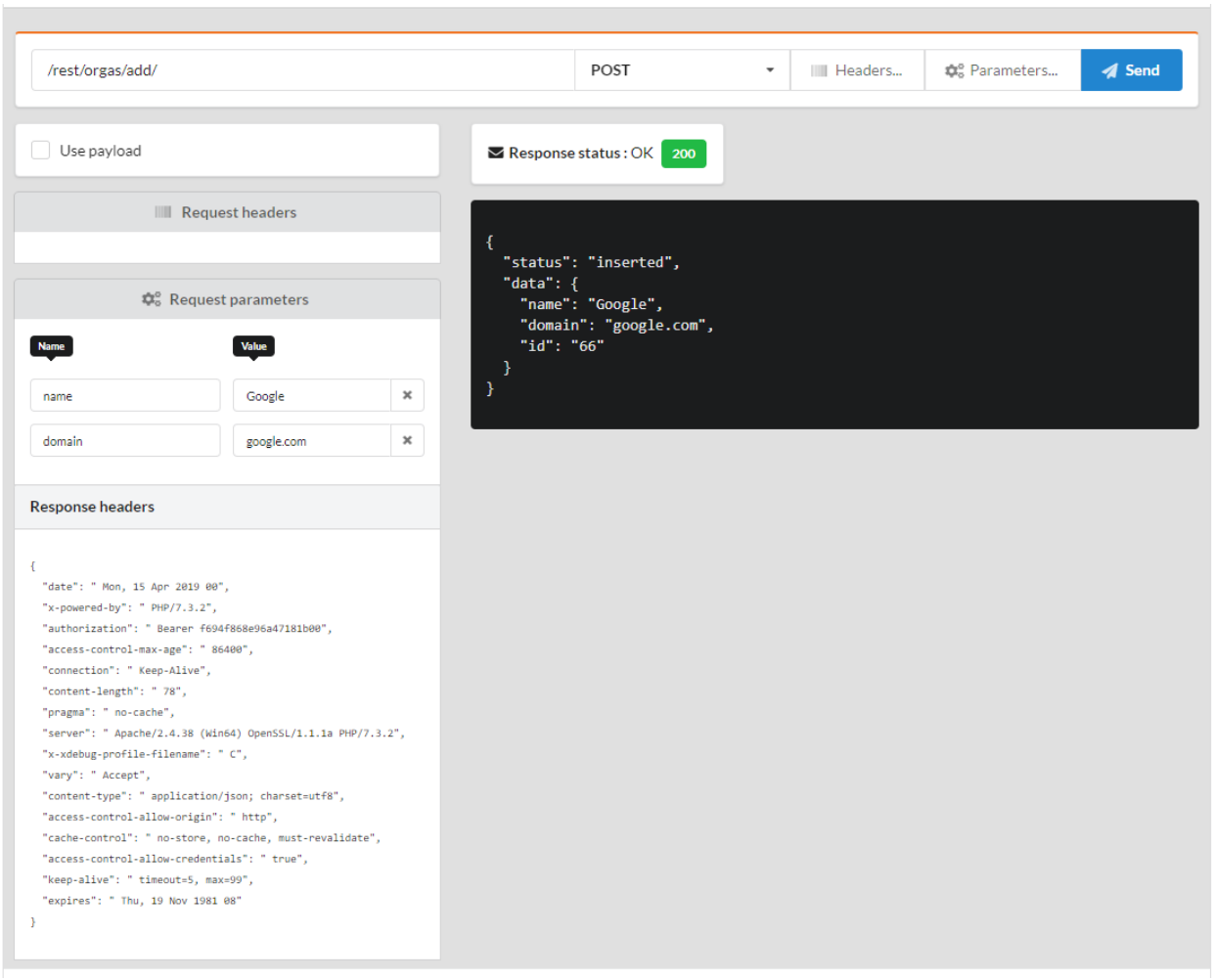


La interfaz de administración permite simular la autenticación por defecto y obtener un token, solicitando el método

connect:



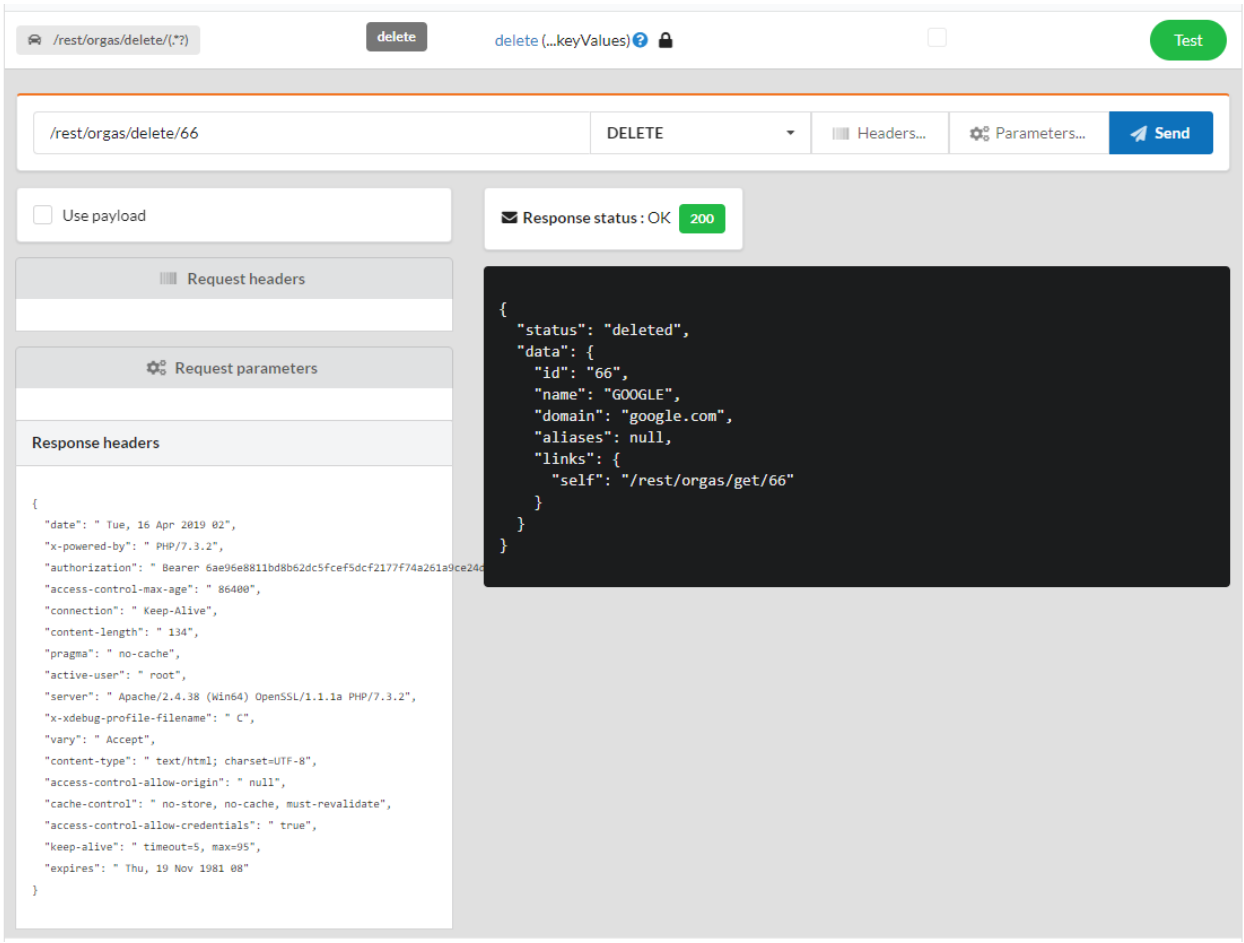
El token se envía automáticamente en las siguientes solicitudes. A continuación, se puede insertar el registro.



Actualización de una instancia

La actualización sigue el mismo esquema que la inserción.

Borrar una instancia



33.2.3 Personalización

Rutas

Por supuesto, es posible personalizar y simplificar las rutas. En este caso, es preferible utilizar la herencia de la clase **RestBaseController**, y no habilitar las rutas automáticas.

Lista 5: app/controllers/RestOrgas.php

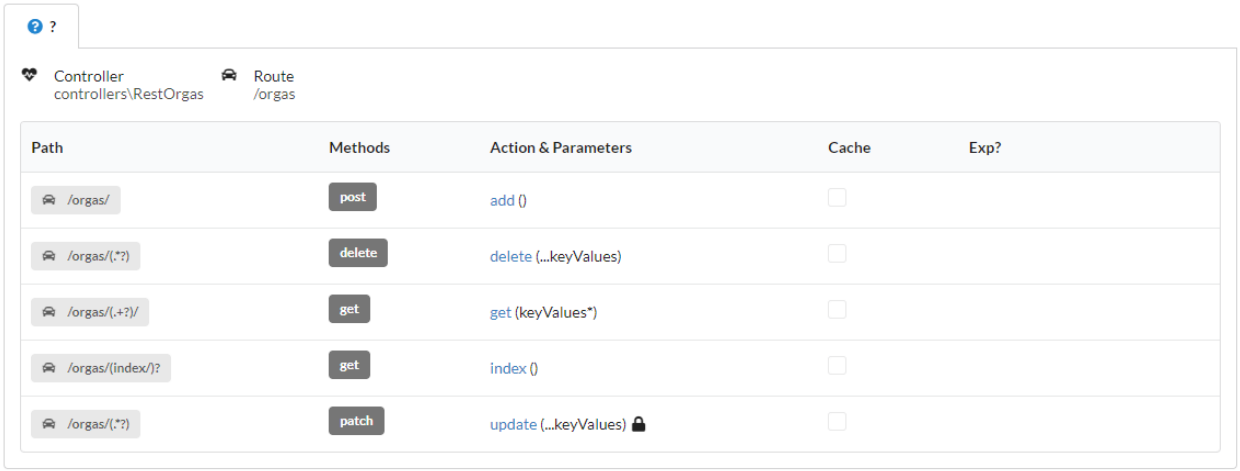
```
1  namespace controllers;
2
3  use models\Organization;
4
5  /**
6   * Rest Controller for organizations
7   *
8   * @route("/orgas")
9   * @rest
10  */
11 class RestOrgas extends \Ubiquity\controllers\rest\RestBaseController {
```

(continué en la próxima página)

(proviene de la página anterior)

```
    public function initialize() {
            $this->model = Organization::class;
            parent::initialize();
    }

    /**
     *
     * @get
     */
    public function index() {
            $this->_get();
    }

    /**
     *
     * @get("{keyValues}")
     */
    public function get($keyValues) {
            $this->_getOne($keyValues);
    }

    /**
     *
     * @post("/")
     */
    public function add() {
            $this->_add();
    }

    /**
     *
     * @patch("{keyValues}")
     */
    public function update(...$keyValues) {
            $this->_update(...$keyValues);
    }

    /**
     *
     * @delete("{keyValues}")
     */
    public function delete(...$keyValues) {
            $this->_delete(...$keyValues);
    }
}
```

Tras reinicializar la caché, la interfaz de prueba muestra las rutas accesibles:



Modificación de los datos enviados

33.2.4 By overriding

Es posible modificar los datos enviados a los métodos update y add, para añadir, modificar o borrar el valor de los campos antes de enviarlos. Ya sea sobredefiniendo el método getDatas:

Lista 6: app/controllers/RestOrgas.php

```
...

    protected function getDatas() {
            $datas = parent::getDatas();
            unset($datas['aliases']);// Remove aliases field
            return $datas;
    }
```

33.2.5 Con eventos

O bien de forma más global actuando sobre los eventos de descanso:

Lista 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 Autentificación

Ubiquity REST implementa una autenticación Oauth2 con tokens Bearer. **|br|Sólo los métodos con la anotación ``@authorization`` requieren la autenticación, estos son los métodos de modificación (añadir, actualizar y eliminar). |br|**

```
            /**
             * 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 );
            }
```

El método **connect** de un controlador REST establece la conexión y devuelve un nuevo token. Corresponde al desarrollador anular este método para gestionar una posible autenticación con nombre de usuario y contraseña.

```
{
  "access_token": "b641bf027617428c6eb6",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

33.3.1 Simulación de una conexión con inicio de sesión

En este ejemplo, la conexión consiste simplemente en enviar una variable de usuario por el método post. Si se proporciona el usuario, el método `connect` de la instancia `$server` devuelve un token válido que se almacena en sesión (la sesión actúa aquí como base de datos).

Lista 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
```

(continué en la próxima página)

(proviene de la página anterior)

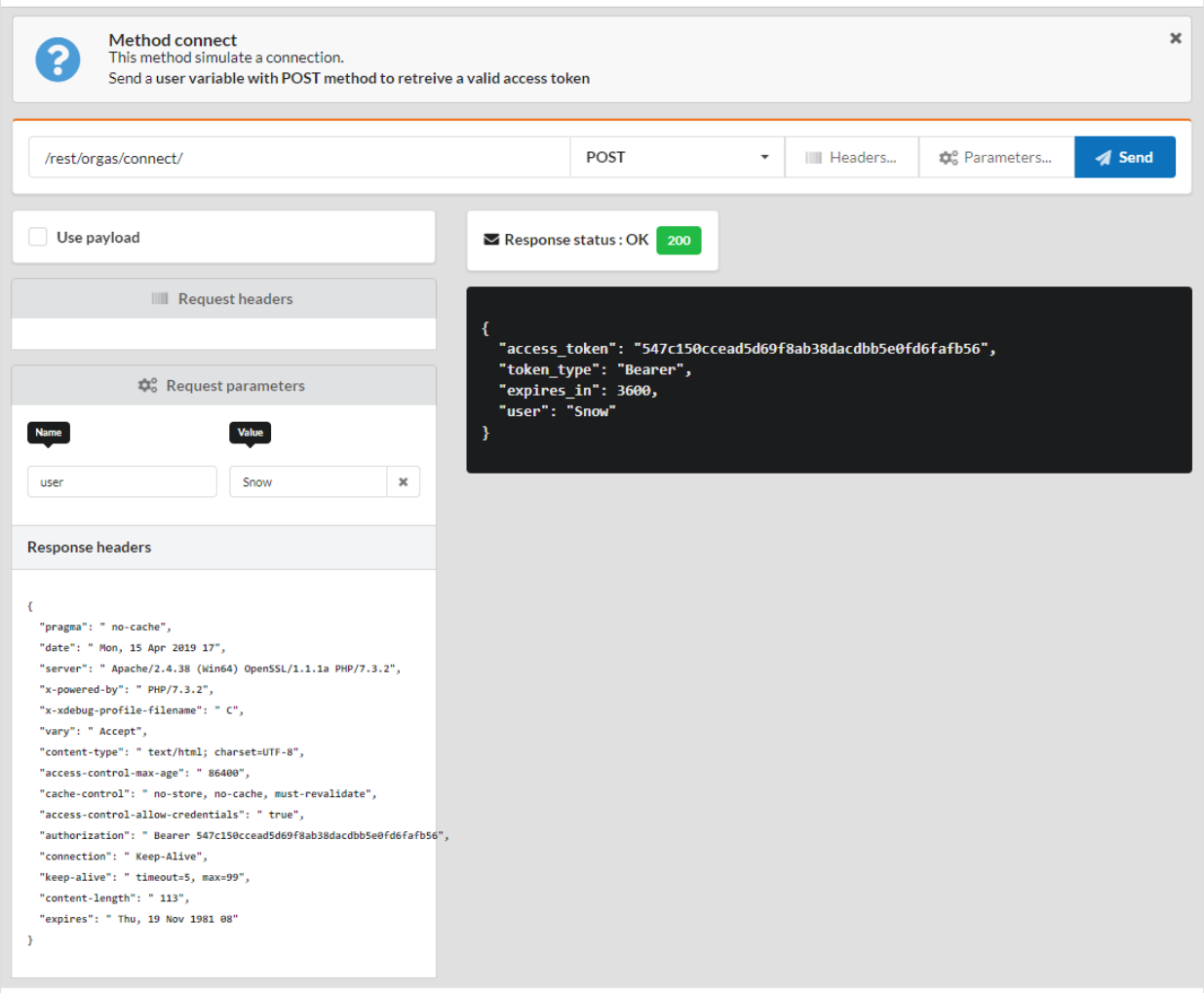
```
13              /**
14               * This method simulate a connection.
15               * Send a <b>user</b> variable with <b>POST</b> method to retreive a valid␣
    ↪access token
16               * @route("methods"=>["post"])
17               */
18              public function connect(){
19                      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      }
```

Para cada solicitud con autenticación, es posible recuperar el usuario conectado (se añade aquí en las cabeceras de respuesta) :

Lista 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      }
```

Utilice la interfaz webtools para probar la conexión:



33.4 Personalización

33.4.1 Api tokens

Es posible personalizar la generación de tokens anulando el método `getRestServer`:

Lista 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);
```

(continué en la próxima página)

(proviene de la página anterior)

```
                $srv->setTokenLength(32);
                $srv->setTokenDuration(4800);
                return $srv;
        }
}
```

33.4.2 Orígenes y CORS permitidos

Compartición de recursos entre orígenes (CORS)

Si accede a su api desde otro sitio, es necesario configurar **CORS**.

En este caso, para peticiones de tipo `PATCH`, `PUT`, `DELETE`, tu api debe definir una ruta que permita a CORS realizar su control pre-petición mediante el método `OPTIONS`.

Lista 11: app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        /**
         * @options('{url}')
         */
        public function options($url='') {}
}
```

Orígenes permitidos

Los orígenes permitidos permiten definir los clientes que pueden acceder al recurso en caso de una petición entre dominios definiendo la cabecera de respuesta **Access-Control-Allow-Origin**. Este campo de cabecera es devuelto por el método `OPTIONS`.

Lista 12: app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        protected function getRestServer(): RestServer {
                $srv= new RestServer($this->config);
                $srv->setAllowOrigin('http://mydomain/');
                return $srv;
        }
}
```

Es posible autorizar varios orígenes:

Lista 13: app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        protected function getRestServer(): RestServer {
                $srv= new RestServer($this->config);
                $srv->setAllowOrigins(['http://mydomain1/','http://mydomain2/']);
                return $srv;
        }
}
```

33.4.3 Respuesta

Para cambiar el formato de las respuestas, es necesario crear una clase que herede de `ResponseFormatter`. Nos inspiraremos en **HAL**, y cambiaremos el formato de las respuestas por:

- añadir un enlace a sí mismo para cada recurso
- añadir un atributo `_embedded` para las colecciones
- eliminación del atributo `data` para los recursos únicos

Lista 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 ) ] );
        }
}
```

A continuación, asigna `MyResponseFormatter` al controlador REST anulando el método `getResponseFormatter`:

Lista 15: app/controllers/RestOrgas.php

```
class RestOrgas extends \Ubiquity\controllers\rest\RestController {

        ...

        protected function getResponseFormatter(): ResponseFormatter {
                return new MyResponseFormatter();
        }
}
```

Comprueba los resultados con los métodos getOne y 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

A diferencia de los recursos REST, los controladores API son multirrecursos.

33.5.1 SimpleRestAPI

33.5.2 JsonApi

Ubiquity implementa la especificación jsonApi con la clase `JsonApiRestController`. JsonApi es utilizado por ``EmberJS <https://api.emberjs.com/ember-data/release/classes/DS.JSONAPIAdapter>`_and otros. ver https://jsonapi.org/ para más.

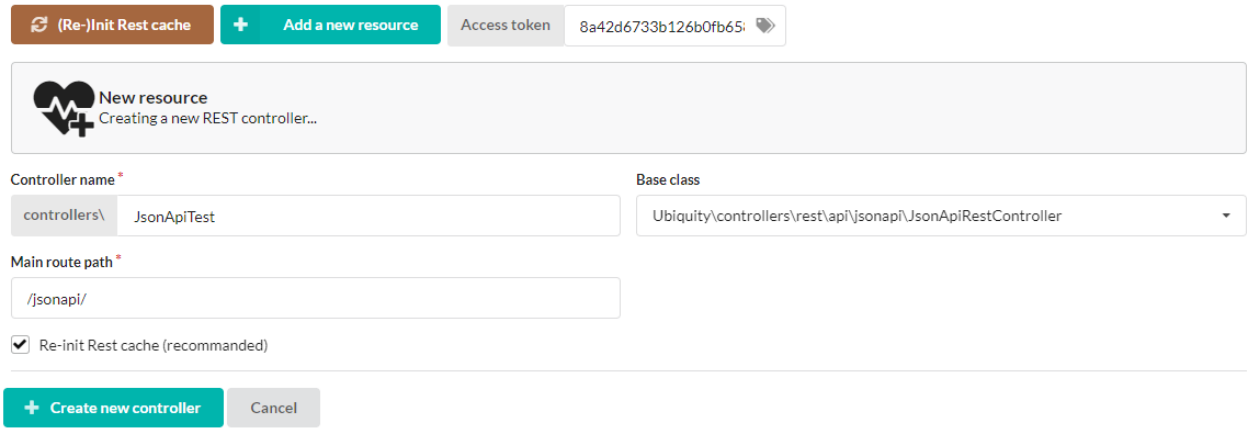
Creación

Con devtools:

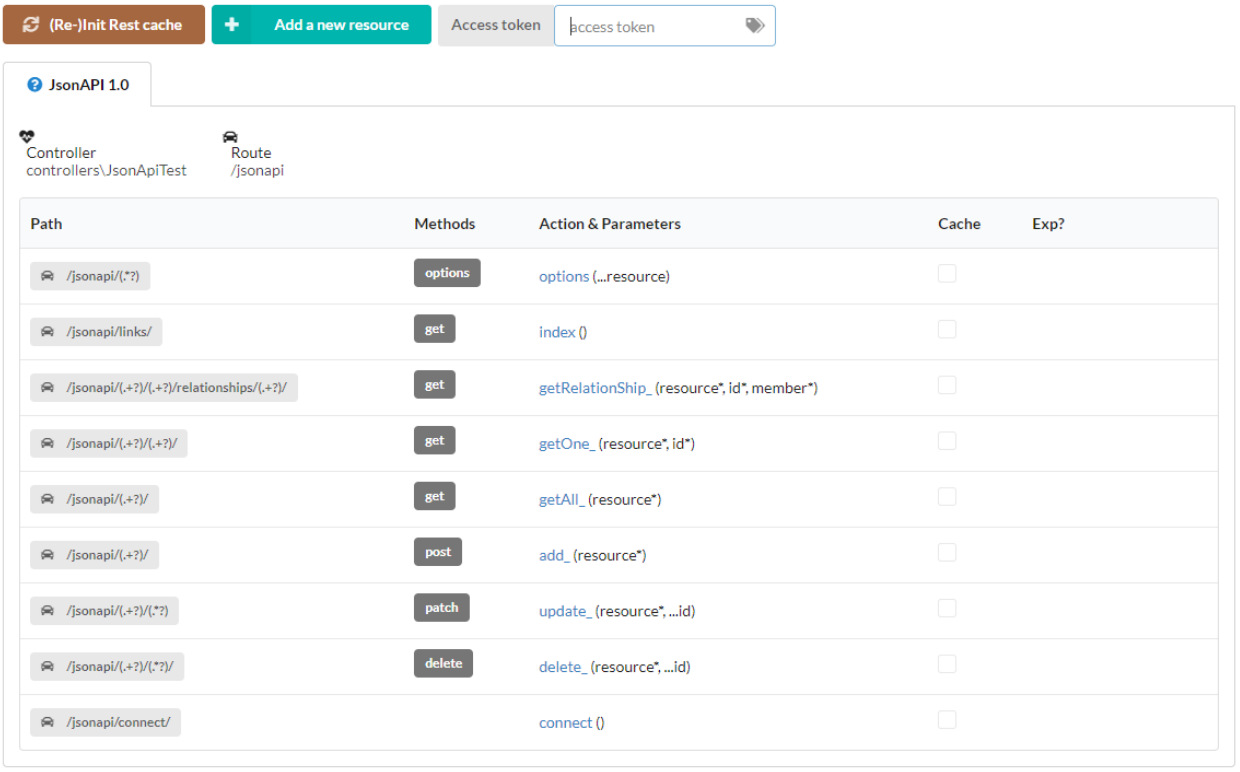
```
Ubiquity restapi JsonApiTest -p=/jsonapi
```

O con webtools:

Vaya a la sección **REST** y elija **Añadir un nuevo recurso**:

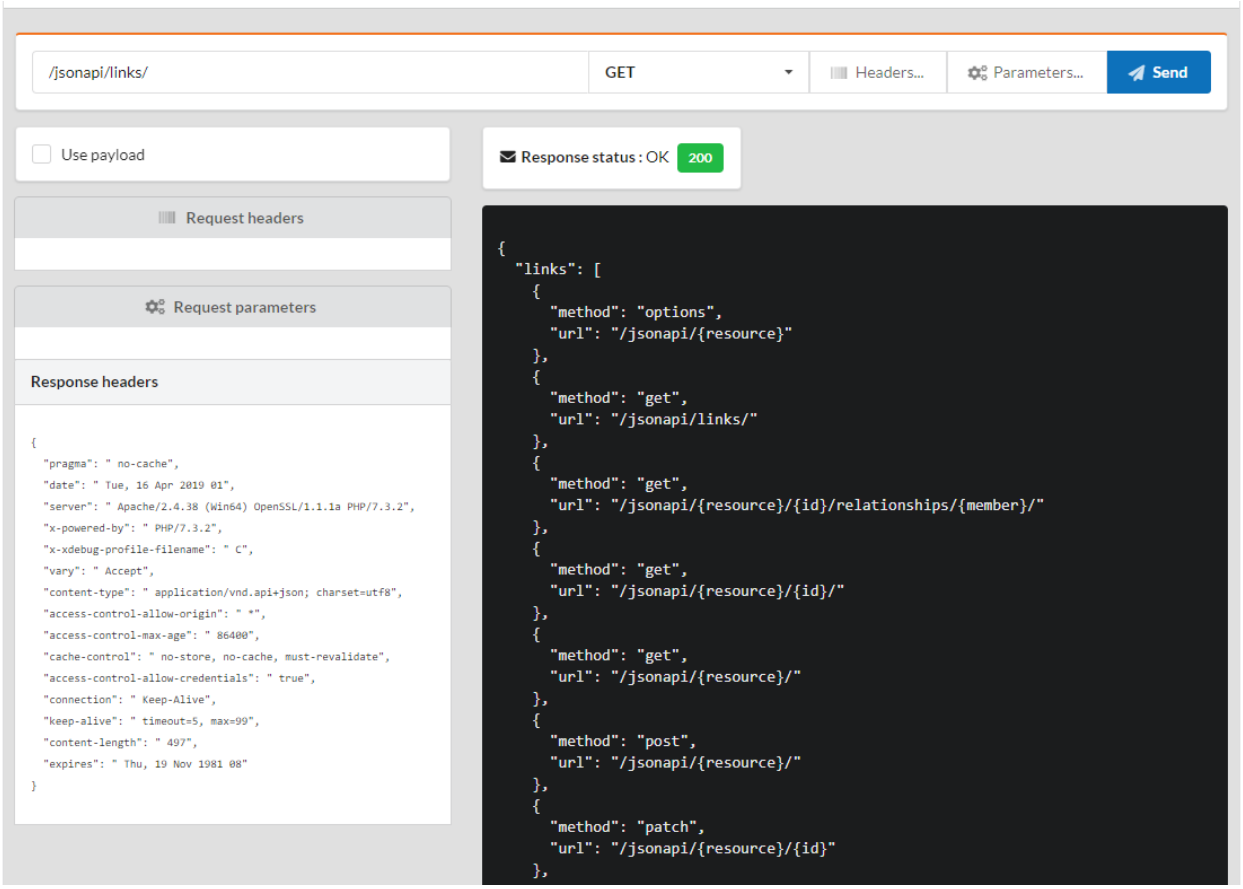


Prueba la api en webtools:



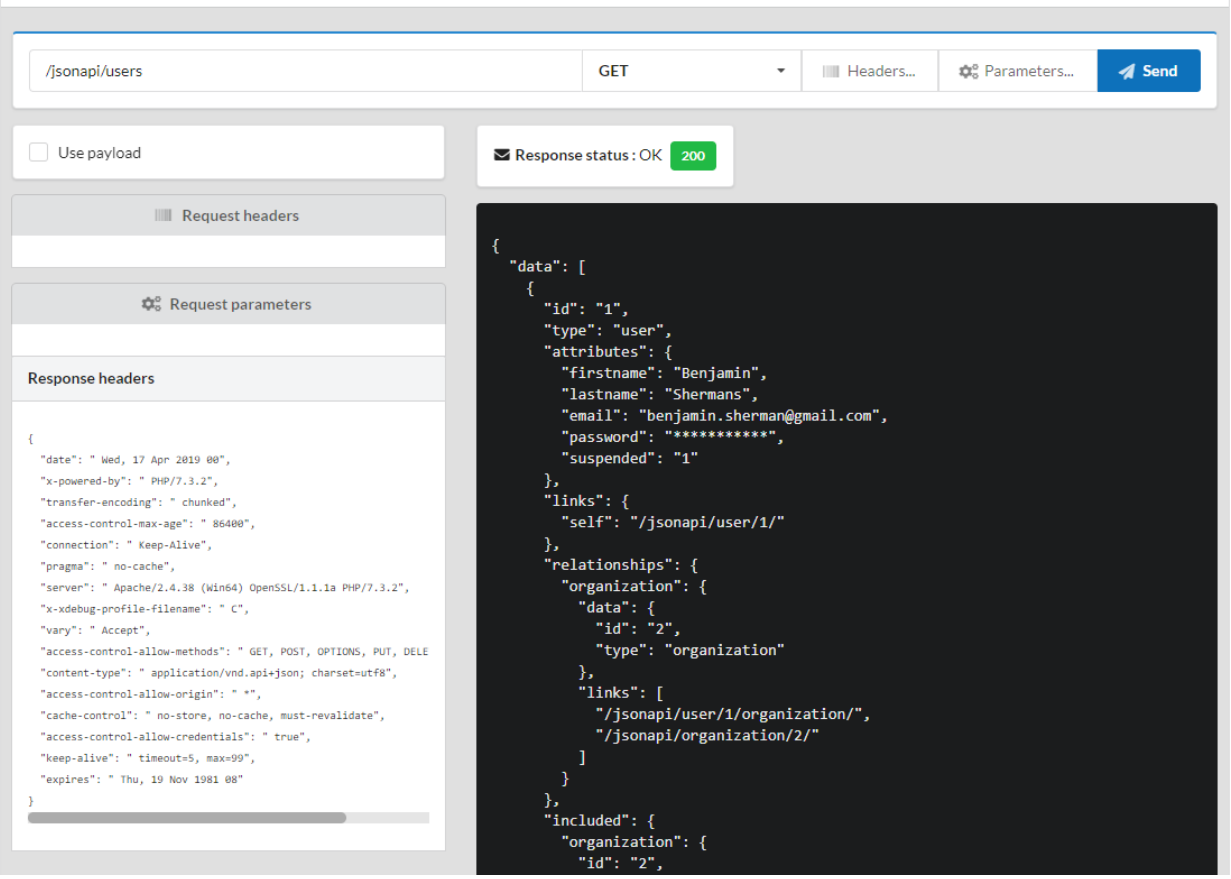
Links

La ruta **links** (método index) devuelve la lista de urls disponibles:



Obtener una matriz de objetos

Por defecto, se incluyen todos los miembros asociados:



Incluidos los miembros asociados

debe utilizar el parámetro **include** de la solicitud:

URL	Descripción
`/jsonapi/user?include=false`	No se incluyen miembros asociados
`/jsonapi/user?include=organization`	Incluir la organización
`/jsonapi/user?include=organization,connections`	Incluir la organización y las conexiones
`/jsonapi/user?include=groupes.organization`	Incluir los grupos y su organización

Filtrado de instancias

debe utilizar el parámetro **filter** de la solicitud, el parámetro **filter** corresponde a la parte **where** de una sentencia SQL:

URL	Descripción
`/jsonapi/user?1=1`	Sin filtro
`/jsonapi/user?firstname='Benjamin'`	Devuelve todos los usuarios llamados Benjamin
`/jsonapi/user?filter=firstname like 'B*'`	Devuelve todos los usuarios cuyo nombre empieza por B
`/jsonapi/user?filter=suspended=0 and lastname like 'ca*'`	Devuelve todos los usuarios suspendidos cuyo apellido empiece por ca

Paginación

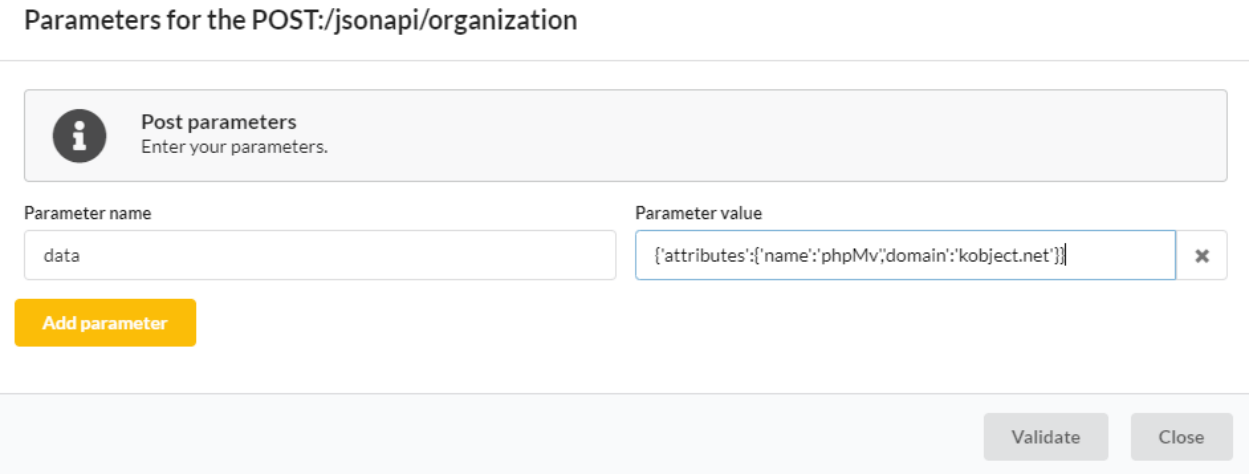
debe utilizar los parámetros **page[number]** y **page[size]** de la solicitud:

URL	Descripción
`/jsonapi/user`	Sin paginación
`/jsonapi/user?page[number]=1`	Mostrar la primera página (el tamaño de página es 1)
`/jsonapi/user?page[number]=1&&page[size]=10`	Mostrar la primera página (el tamaño de página es 10)

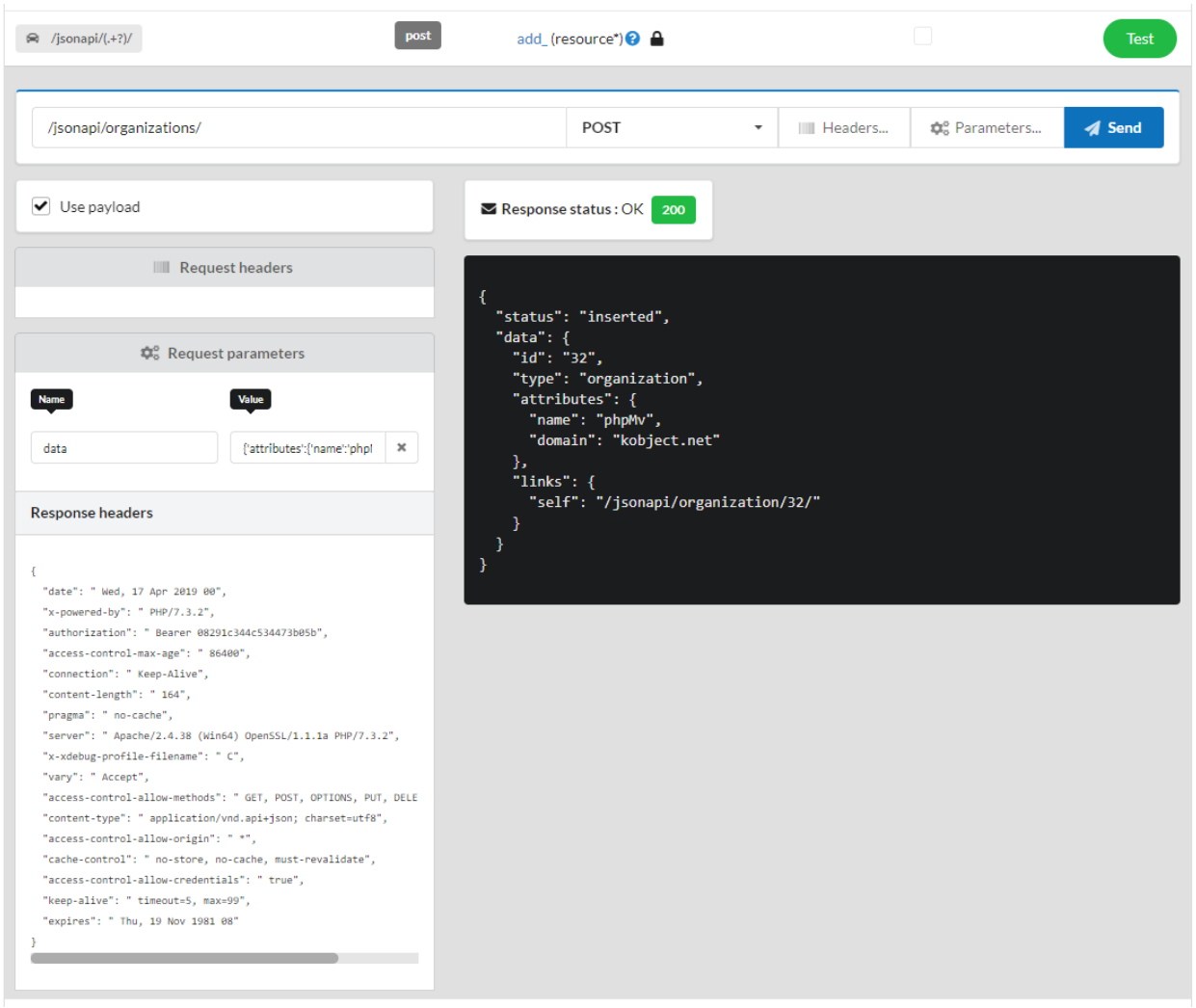
Añadir una instancia

Los datos, contenidos en `data[attributes]`, se envían mediante el método **POST**, con un tipo de contenido definido en `application/json; charset=utf-8`.

Añada sus parámetros haciendo clic en el botón **parameters**:

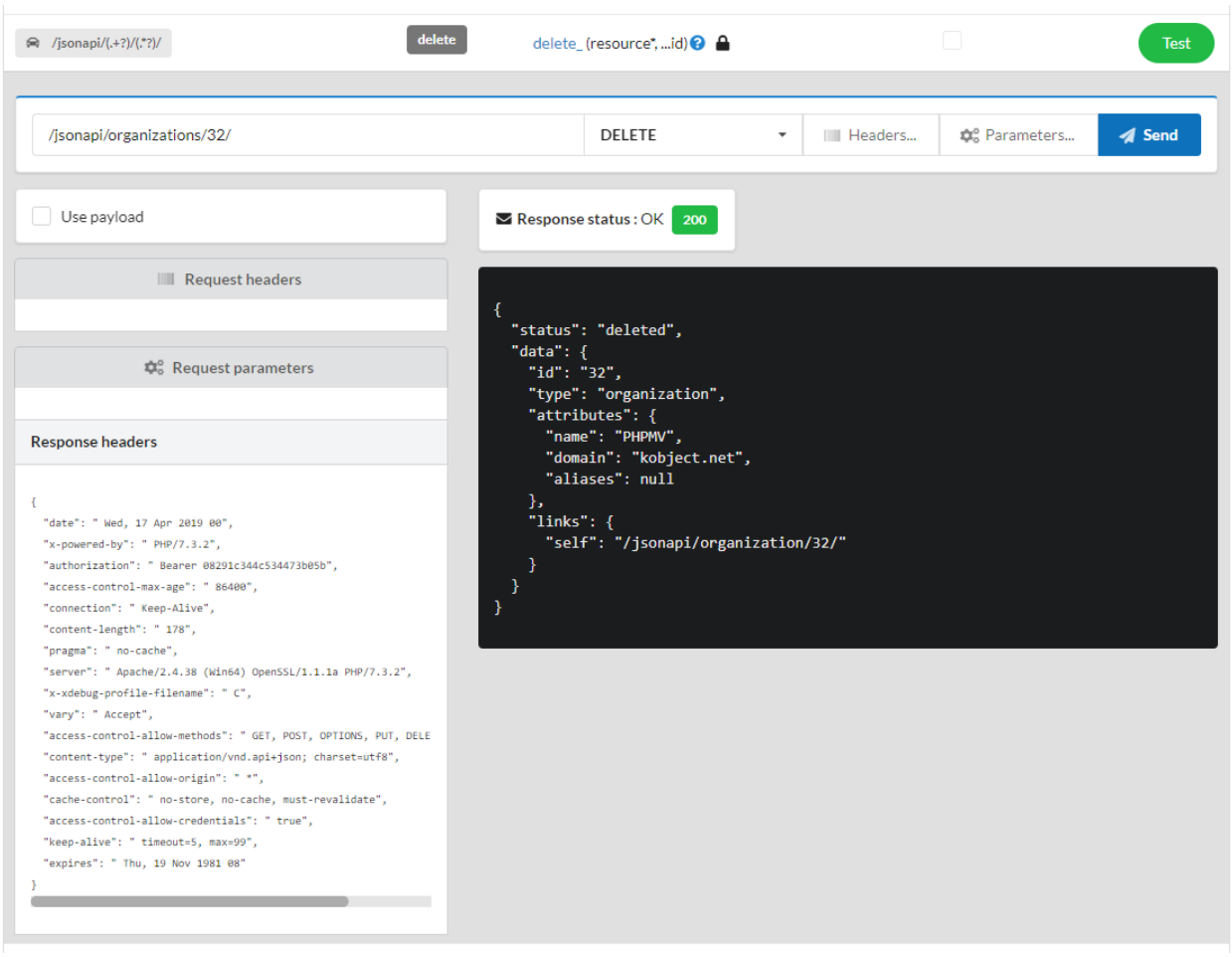


La adición requiere una autenticación, por lo que se genera un error, con el estado 401 si el token está ausente o caducado.



Borrar una instancia

El borrado requiere el método **DELETE**, y el uso del **id** del objeto a borrar:


/jsonapi/(.+?)/(.*?)/
delete
delete_(resource*, ...id)
Test
/jsonapi/organizations/32/
DELETE
Headers...
Parameters...
Send
Use payload
Response status : OK 200
Request headers
Request parameters
Response headers
{
"date": " Wed, 17 Apr 2019 00",
"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"
}
{
"status": "deleted",
"data": {
"id": "32",
"type": "organization",
"attributes": {
"name": "PHPMV",
"domain": "kobject.net",
"aliases": null
},
"links": {
"self": "/jsonapi/organization/32/"
}
}
}

CAPÍTULO 34

Webtools

Nota: Webtools allow you to manage an Ubiquity application via a web interface. Since **Ubiquity 2.2.0**, webtools are in a separate repository.

34.1 Installation

Update the devtools if necessary to get started:

```
composer global update
```

34.1.1 At the project creation

Create a projet with **webtools** (`-a` option)

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

34.1.2 In an existing project

In a console, go to the project folder and execute:

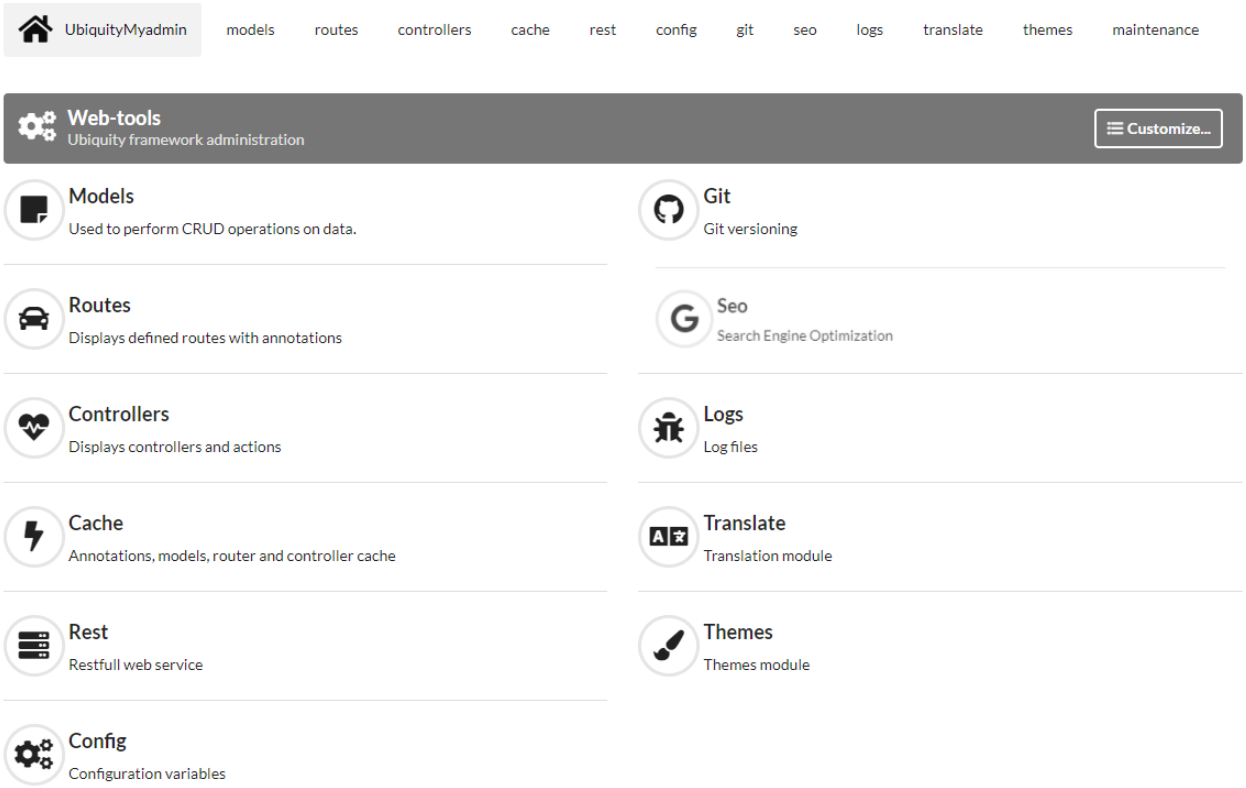
```
Ubiquity admin
```

34.2 Starting

Start the embedded web server, from the project folder:

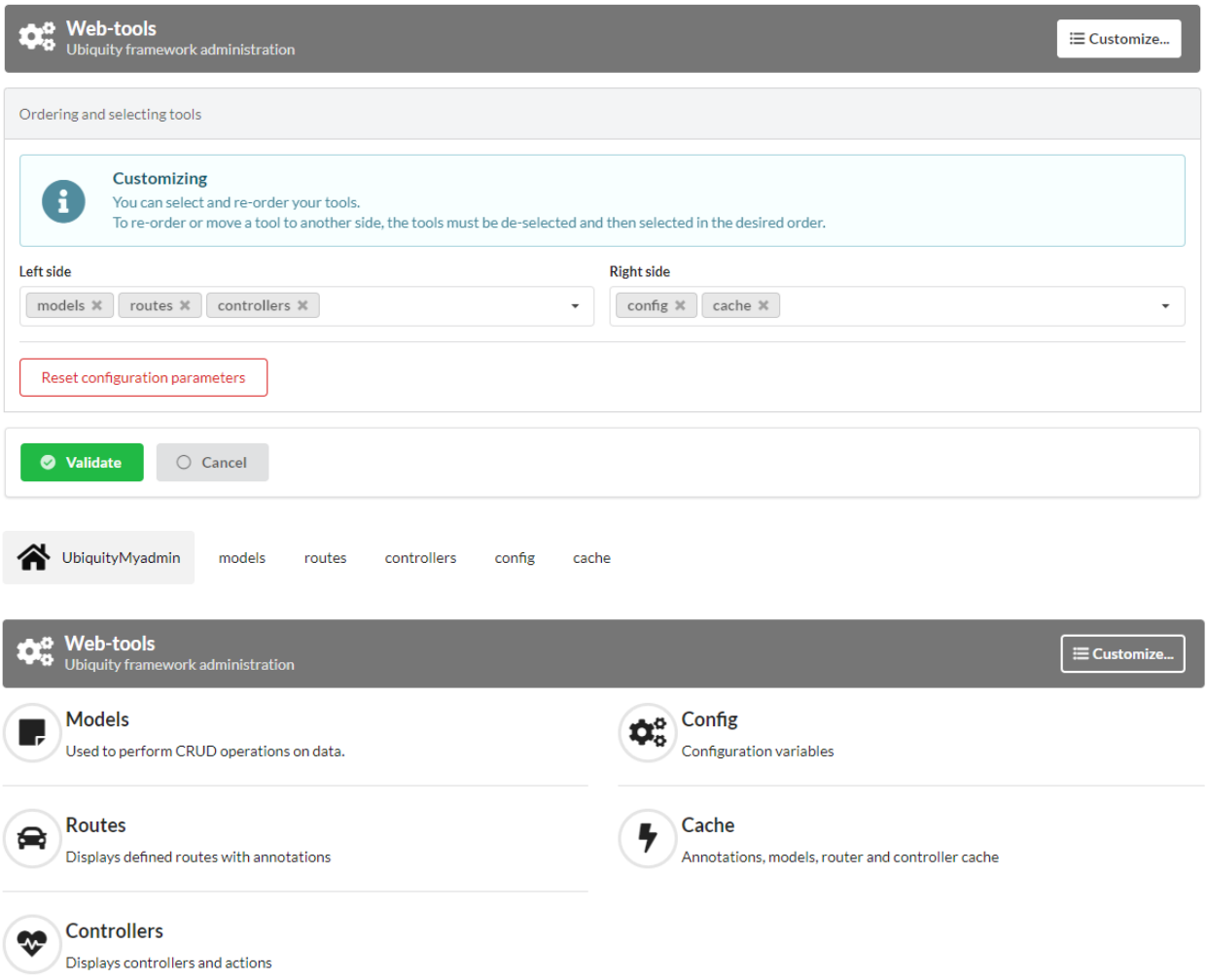
```
Ubiquity serve
```

go to the address: `http://127.0.0.1:8090/Admin`



34.3 Customizing

Click on **customize** to display only the tools you use:



34.4 Webtools modules

34.4.1 Routes

Displays default (non REST) routes.

Operations:

- Filter routes
- Test routes (GET, POST...)
- Initialize router cache

34.4.2 Controllers

Displays non REST controllers.

Operations:

- Create a controller (and optionally the view associated to the default **index** action)
- Create an action in a controller (optionally the associated view, the associated route)
- Create a special controller (CRUD or Auth)
- Test an action (GET, POST...)

34.4.3 Models

Displays the metadatas of the models, allows to browse the entities.

Operations:

- Create models from database
- Generate models cache
- Generate database script from existing models
- Performs CRUD operations on models

34.4.4 Rest

Displays an manage REST services.

Operations:

- Re-initialize Rest cache and routes
- Create a new Service (using an api)
- Create a new resource (associated to a model)
- Test and query a web service using http methods
- Performs CRUD operations on models

34.4.5 Cache

Displays cache files.

Operations:

- Delete or re-initialize models cache
- Delete or re-initialize controllers cache
- Delete other cache files

34.4.6 Maintenance

Allows to manage maintenance modes.

Operations:

- Create or update a maintenance mode
- De/Activate a maintenance mode
- Delete a maintenance mode

34.4.7 Config

Allows the display and modification of the app configuration.

34.4.8 Git

Synchronizes the project using git.

Operations:

- Configuration with external repositories
- Commit
- Push
- Pull

Manages Css themes.

Operations:

- Install an existing theme
- Activate a theme
- Create a new theme (eventually base on an existing theme)

CAPÍTULO 35

Contribución

35.1 Requisitos del sistema

Antes de trabajar en Ubiquity, configure su entorno con el siguiente software:

- Git
- PHP versión 7.1 o superior.

35.2 Obtenga el código fuente de Ubiquity

En Ubiquity repositorio github :

- Haga clic en *Fork* proyecto Ubiquity
- Clona tu fork localmente:

```
git clone git@github.com:USERNAME/ubiquity.git
```

35.3 Trabaje en su parche

Nota: Antes de empezar, debes saber que todos los parches que vayas a enviar deben publicarse bajo la licencia Apache 2.0, a menos que se especifique explícitamente en tus commits.

35.3.1 Crear un branch temático

Nota: Utilice un nombre descriptivo para su branch:

- issue_xxx donde xxx es el número de incidencia es una buena convención para las correcciones de errores
- feature_name es una buena convención para las nuevas funciones

```
git checkout -b NEW_BRANCH_NAME master
```

35.3.2 Trabaje en su parche

Trabaja en tu código y haz todas las confirmaciones que quieras, y ten en cuenta lo siguiente:

- Leer sobre las *Normas de codificación de Ubiquity*;
- Añade pruebas unitarias, funcionales o de aceptación para demostrar que el error se ha corregido o que la nueva función funciona realmente;
- Haga commits atómicos y lógicamente separados (use *git rebase* para tener un historial limpio y lógico);
- Escriba buenos mensajes de confirmación (consulte el consejo siguiente).
- Incrementa los números de versión en cualquier archivo modificado, respetando las reglas de semver:

  Dado un número de versión `MAJOR.MINOR.PATCH`, incrementa el:

  - versión `MAJOR` cuando realice cambios incompatibles en la API,
  - versión `MINOR` cuando añada funciones de forma compatible con versiones anteriores, y
  - versión `PATCH` cuando realice correcciones de errores compatibles con versiones anteriores.

35.4 Envíe su parche

Actualice la parte [Unrelease] del archivo CHANGELOG.md integrando sus cambios en las partes correspondientes:

- Añadido
- Cambiado
- Cambiado
- Arreglado

Eventualmente, vuelva a basar su parche Antes de enviar, actualice su rama (necesario si tarda en terminar sus cambios):

```
git checkout master
git fetch upstream
git merge upstream/master
git checkout NEW_BRANCH_NAME
git rebase master
```

35.5 Hacer una Pull Request

Ahora puede hacer un pull request en Ubiquity repositorio github .

CAPÍTULO 36

Guía de codificación

Nota: Aunque el framework es muy reciente, tenga en cuenta que algunas de las primeras clases de Ubiquity no siguen completamente esta guía y no han sido modificadas por razones de compatibilidad con versiones anteriores. Sin embargo, todos los códigos nuevos deben seguir esta guía.

36.1 Opciones de diseño

36.1.1 Obtención y utilización de servicios

Inyecciones de dependencia

Evite utilizar la inyección de dependencias para todas las partes del framework, internamente. La inyección de dependencias es un mecanismo que consume muchos recursos:

- necesita identificar el elemento a instanciar ;
- a continuación, proceder a su instanciación ;
- para finalmente asignarlo a una variable.

Obtener servicios de un contenedor

Evite también el acceso público a servicios registrados en un contenedor de servicios. Este tipo de acceso implica manipular objetos cuyo tipo de retorno se desconoce, lo que no es fácil de manejar para el desarrollador.

Por ejemplo, es difícil manipular el retorno no tipado de `$this->serviceContainer->get('translator')`, como permiten algunos frameworks, y saber qué métodos llamar en él.

Cuando sea posible, y cuando no reduzca demasiado la flexibilidad, se sugiere el uso de clases estáticas:

Para un desarrollador, la clase `TranslatorManager` es accesible desde todo un proyecto sin necesidad de instanciar ningún objeto. Expone su interfaz pública y permite completar el código:

- No es necesario inyectar el traductor para utilizarlo;
- No es necesario recuperarlo de un contenedor de servicios.

El uso de clases estáticas crea inevitablemente una fuerte dependencia y afecta a la flexibilidad. Pero volviendo al ejemplo del Traductor, no hay razón para cambiarlo si es satisfactorio. No es deseable querer proporcionar flexibilidad a toda costa cuando no es necesario, y que luego el usuario vea que su aplicación es un poco lenta.

36.2 Optimización

La ejecución de cada línea de código puede tener importantes repercusiones en el rendimiento. Compara y compara soluciones de implementación, especialmente si el código se llama repetidamente:

- Identifique estas llamadas repetitivas y costosas con herramientas de perfilado de php (Blackfire profiler , Tideways ...)
- Evalúe sus diferentes soluciones de implementación con phpMyBenchmarks

36.3 Calidad de código

Ubiquity utiliza Scrutinizer-CI para la calidad del código.

- Para clases y métodos :
    - Las evaluaciones A o B son buenas
    - C es aceptable, pero debe evitarse si es posible
    - Las notas más bajas deben prohibirse

36.3.1 Complejidad del código

- Los métodos complejos deben dividirse en varios, para facilitar su mantenimiento y permitir su reutilización;
- Para clases complejas, haga una refactorización extract-class o extract-subclass y divídalas usando Traits;

36.3.2 Duplicaciones de código

Evite absolutamente la duplicación de código, excepto si la duplicación es mínima y está justificada por el rendimiento.

36.3.3 Errores

Intenta resolver sobre la marcha todos los errores que se reporten, sin dejar que se acumulen.

36.4 Pruebas

Cualquier corrección de errores que no incluya una prueba que demuestre la existencia del error corregido puede ser sospechosa. Lo mismo ocurre con las nuevas funciones que no pueden demostrar que funcionan.

También es importante mantener una cobertura aceptable, que puede disminuir si no se prueba una nueva función.

36.5 Documentación de código

El código actual aún no está completamente documentado, siéntase libre de contribuir para llenar este vacío.

36.6 Normas de codificación

Las normas de codificación de Ubiquity se basan principalmente en las normas PSR-1 , PSR-2 y PSR-4 , por lo que es posible que ya conozca la mayoría de ellas. Las pocas excepciones intencionadas a las normas se informan normalmente en esta guía.

36.6.1 Convenciones de nombrado

- Utilice camelCase para variables PHP, miembros, nombres de funciones y métodos, argumentos (por ejemplo, $modelsCacheDirectory, isStarted());
- Utilice namespaces para todas las clases PHP y UpperCamelCase para sus nombres (por ejemplo, CacheManager);
- Prefija todas las clases abstractas con Abstract excepto PHPUnit BaseTests;
- Sufija las interfaces con `Interface`;
- Sufija los traits con `Trait`;
- Sufija las excepciones con `Exception`;
- Sufijo gestor de clases principales con `Manager` (por ejemplo, CacheManager, TranslatorManager);
- Prefije las clases de utilidad con `U` (por ejemplo, UString, URequest);
- Utilice UpperCamelCase para nombrar los archivos PHP (por ejemplo, CacheManager.php);
- Utilice mayúsculas para las constantes (por ejemplo, const SESSION_NAME="Ubiquity").

36.6.2 Sangría, tabulaciones, llaves

- Utilizar tabuladores, no espacios; (!PSR-2)
- Utilizar llaves siempre en la misma línea; (!PSR-2)
- Utilice llaves para indicar el cuerpo de la estructura de control, independientemente del número de sentencias que contenga;

36.6.3 Clases

- Defina una clase por archivo;
- Declare la herencia de la clase y todas las interfaces implementadas en la misma línea que el nombre de la clase;
- Declare las propiedades de la clase antes que los métodos;
- Declare primero los métodos privados, luego los protegidos y finalmente los públicos;
- Declare todos los argumentos en la misma línea que el nombre del método/función, sin importar el número de argumentos;
- Utilice paréntesis al instanciar clases independientemente del número de argumentos que tenga el constructor;
- Añade una declaración de uso para cada clase que no forme parte del espacio de nombres global;

36.6.4 Operadores

- Utilice comparación idéntica e igual cuando necesite hacer manipulación de tipos;

Ejemplo

```
<?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**
         *
```

(continué en la próxima página)

(proviene de la página anterior)

```
         * @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 );
                }
        }
}
```

Importante:

Puede importar estos archivos de estandarización que integran todas estas reglas en su IDE:

- `Eclipse`
- `PhpStorm`

Si su IDE preferido no aparece en la lista, puede enviar el archivo de normalización asociado creando un nuevo PR.

CAPÍTULO 37

Guía de documentación

Ubiquity tiene dos conjuntos principales de documentación:

- las guías, que te ayudan a aprender sobre manipulaciones o conceptos ;
- y la API, que sirve de referencia para la codificación.

Puede ayudar a mejorar las guías de Ubiquity haciéndolas más coherentes, consistentes o legibles, añadiendo información que falte, corrigiendo errores factuales, corrigiendo erratas o actualizándolas con la última versión de Ubiquity.

Para ello, realice cambios en los archivos fuente de las guías Ubiquity (ubicados aquí en GitHub). A continuación, abra una solicitud de extracción para aplicar los cambios al branch principal.

Cuando trabaje con documentación, tenga en cuenta las directrices.

CAPÍTULO 38

Configuración de servidores

Importante: Desde la versión 2.4.5, por razones de seguridad y simplificación, la raíz de una aplicación Ubiquity se encuentra en la carpeta pública.

38.1 Apache2

38.1.1 mod_php/PHP-CGI

Apache 2.2

Lista 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>
```

Lista 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

En Apache 2.4, `Order Allow,Deny` ha sido reemplazado por `Require all granted`.

Lista 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>
```

reubicación de index.php en la carpeta pública

Si creaste el proyecto con una versión anterior a la 2.4.5, tienes que modificar `index.php` y mover los archivos `index.php` y `.htaccess` a la carpeta `public`.

Lista 4: public/index.php

```
<?php
define('DS', DIRECTORY_SEPARATOR);
//Updated with index.php in public folder
```

(continué en la próxima página)

(proviene de la página anterior)

```
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

Asegúrese de que los paquetes **libapache2-mod-fastcgi** y **php7.x-fpm** están instalados (sustituya **x** por el número de versión de php).

Configuración php-pm:

Lista 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
```

Configuración de Apache 2.4:

Lista 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

Configuración de **nginX**:

Lista 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;
```

(continué en la próxima página)

(proviene de la página anterior)

```
    access_log /var/log/nginx/project_access.log;
}
```

38.3 Swoole

Configuración Swoole:

Lista 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

Configuración **Workerman**:

Lista 9: .ubiquity/workerman-config.php

```
<?php
return array(
    "host" => "0.0.0.0",
    "port" => 8080,
    "socket"=>[
        "count" => 4,
        "reuseport" =>true
    ]
);
```

38.5 RoadRunner

Configuración **RoadRunner**:

Lista 10: .ubiquity/.rr.yml

```
http:
  address:         ":8090"
  workers.command: "php-cgi ./.ubiquity/rr-worker.php"
  workers:
    pool:
      # Set numWorkers to 1 while debugging
```

(continué en la próxima página)

(proviene de la página anterior)

```
      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"]
```

CAPÍTULO 39

Optimización de Ubiquity

Ubiquity es rápido, pero puede serlo aún más optimizando algunos elementos.

Nota: El servidor de pruebas integrado (accesible mediante **Ubiquity serve**) utiliza sus propios archivos de configuración y lanzamiento (en la carpeta **.ubiquity** de su proyecto). Por lo tanto, no debe utilizarse para evaluar los resultados de los cambios realizados.

Pruebe sus páginas utilizando una configuración de software y hardware similar a la utilizada en producción. Utiliza una herramienta de benchmark para evaluar tus cambios a medida que se producen (**Apache bench** por ejemplo).

39.1 Cache

39.1.1 Sistema

Elija y pruebe entre los diferentes sistemas de caché (ArrayCache, PhpFastCache, MemCached). El sistema de caché se define en el archivo de configuración:

Lista 1: app/config/config.php

```
    "cache" => [
            "directory" => "cache/",
            "system" => "Ubiquity\\cache\\system\\ArrayCache",
            "params" => []
    ]
```

por defecto **ArrayCache** es a menudo la solución más optimizada.

39.1.2 Generación

Genera el router y la caché ORM (Piensa que las anotaciones nunca se usan en tiempo de ejecución):

```
Ubiquity init-cache
```

39.1.3 Contenidos estáticos

Si su aplicación tiene páginas que están siendo generadas por PHP pero que en realidad raramente cambian, puede almacenarlas en caché:

- Los resultados de la consulta (utilizando métodos **DAO**)
- La respuesta de ruta (con la anotación **@route**)

39.2 archivo índice

Elimine la línea que define el informe de errores en tiempo de ejecución y asegúrese de que la visualización de errores está desactivada en **php.ini**.

Lista 2: index.php

```
error_reporting(\E_ALL);//To be removed
```

39.3 Optimización de la configuración

Se puede acceder a la configuración desde el archivo `app/config/config.php`.

Conserve sólo los elementos esenciales para su solicitud.

key	role	Optimización
siteUrl	Utilizado por los métodos Ajax y por las funciones `url` y `path` de Twig.	Debe suprimirse si no se utilizan estas funciones
base de datos	Utilizado por Ubiquity ORM	Debe eliminarse si no se utiliza el ORM
sessionName	Si se asigna, inicia o recupera la sesión php para cada petición	Se eliminará si la sesión es inútil
templateEngine	Si se asigna, instanciará un nuevo objeto Motor para cada solicitud.	Se eliminará si no se utilizan las vistas
templateEngineOptions	Opciones asignadas a la instancia del motor de plantillas	establecer la opción de caché en `true` si se utiliza Twig
test	Eliminar (obsoleto)	
debug	Activa o desactiva los registros	Fijar en `false` en producción
logger	Define la instancia del logger	Para eliminar en producción
di	Define los servicios que deben inyectarse	Sólo se lee la clave `@exec` en tiempo de ejecución
cache	Define la ruta de la caché y la clase base de la caché, utilizada por modelos, enrutador, inyección de dependencia	
mvcNS	Define las rutas o espacios de nombres utilizados por los controladores Rest, los modelos y los controladores	
isRest	Define la condición para detectar si una ruta corresponde a un controlador Rest	Se eliminará si no utiliza explícitamente esta condición en su código

Ejemplo de configuración sin sesión, y sin inyección de dependencia:

Lista 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 Optimización de los servicios

Los servicios cargados son accesibles desde el archivo `app/config/services.php`.

En cuanto al archivo de configuración, conserve sólo los elementos esenciales para su aplicación.

Líneas	Rol
\Ubiquity\cache\CacheManager::startProd($config)	Inicia caché para ORM, base de datos, enrutador, inyección de dependencia
UbiquityormDAO::start()	Sólo para bases de datos múltiples
Router::start()	Sólo debe utilizarse si las rutas se definen con anotaciones
Router::addRoute(«_default», «controllers\IndexController»)	Define la ruta por defecto (a eliminar en producción)
\Ubiquity\assets\AssetsManager::start($config)	Asigna la variable siteUrl al ThemeManager, que sólo se utilizará si se usan las funciones `css` y `js` de twig.

Ejemplo de un fichero de Servicios con una base de datos y arranque del router :

Lista 4: app/config/services.php

```
1 <?php
2 \Ubiquity\cache\CacheManager::startProd($config);
3 \Ubiquity\controllers\Router::start();
```

39.5 Optimización del cargador automático

En producción, elimine las dependencias utilizadas únicamente en desarrollo y genere el archivo de mapa de clases optimizado:

```
composer install --no-dev --classmap-authoritative
```

Si las dependencias utilizadas ya se han eliminado y sólo desea actualizar el archivo de mapa (después de añadir o eliminar una clase):

```
composer dump-autoload -o  --classmap-authoritative
```

Nota: El parámetro `--no-dev` elimina la dependencia `ubiquity-dev` requerida por **webtools**. Si utiliza webtools en producción, añada la dependencia `phpmv/ubiquity-dev`:

```
composer require phpmv/ubiquity-dev
```

39.6 Optimización PHP

Tenga en cuenta que otras aplicaciones pueden utilizar los valores modificados en el mismo servidor.

39.6.1 OP-Cache

OPcache mejora el rendimiento de PHP almacenando el código de bytes de los scripts precompilados en memoria compartida, eliminando así la necesidad de que PHP cargue y analice los scripts en cada petición.

Lista 5: php.ini

```
[opcache]
; Determines if Zend OPCache is enabled
opcache.enable=1
```

Lista 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 utiliza el servidor web **ubiquity-swoole**:

Lista 7: php.ini

```
; Determines if Zend OPCache is enabled for the CLI version of PHP
opcache.enable_cli=1
```

39.7 Para completar

Recuerda que el framework utilizado no lo hace todo. También tienes que optimizar tu propio código.

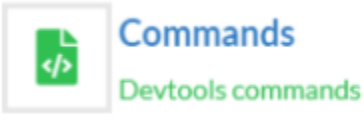
CAPÍTULO 40

Comandos de Ubiquity

Nota: Esta parte es accesible desde las **webtools**, así que si creaste tu proyecto con la opción **-a** o con el comando **create-project**..

40.1 Comandos

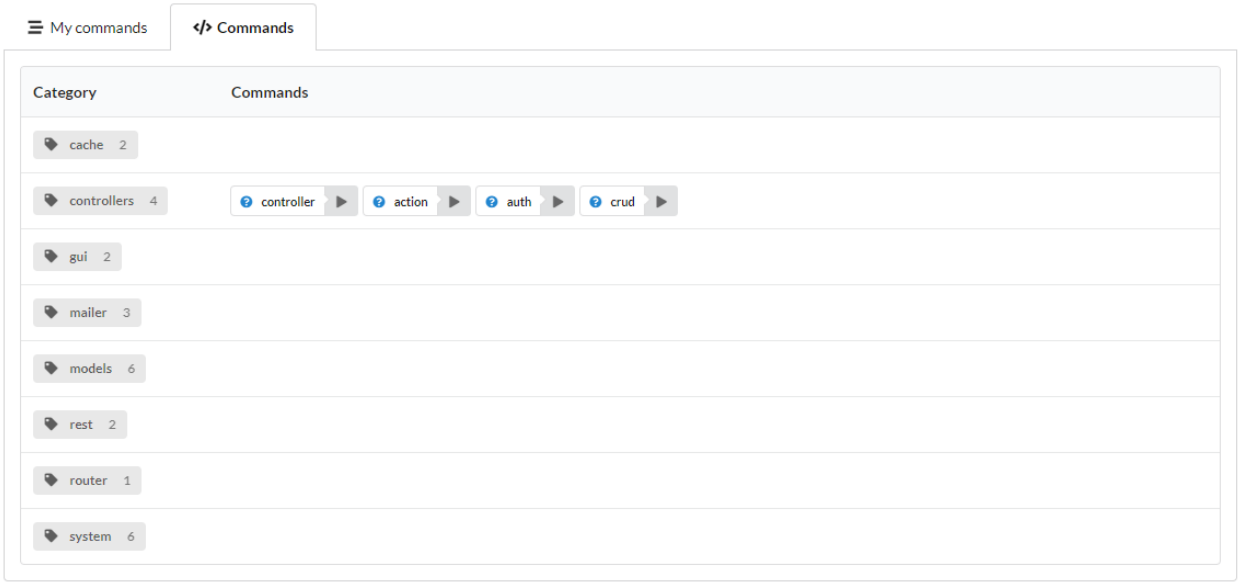
Desde las webtools, activa la parte de **comandos**,



o vaya directamente a `http://127.0.0.1:8090/Admin/commands`.

40.1.1 Lista de comandos

Active la pestaña **Comandos** para obtener la lista de comandos devtools existentes.

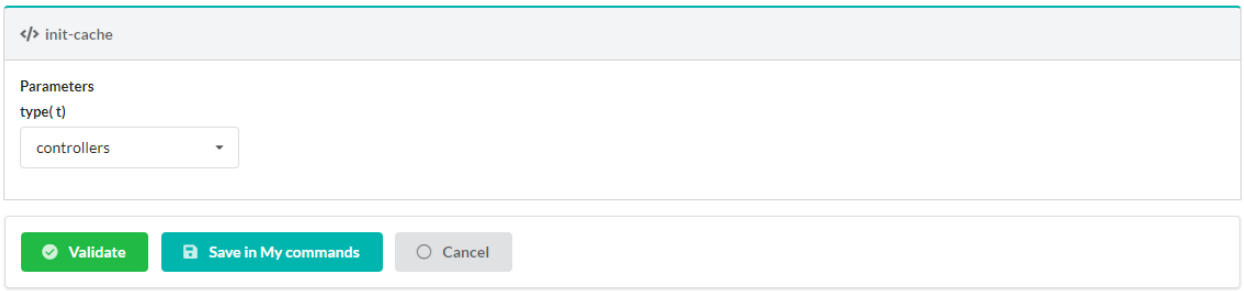


40.1.2 Info de comando

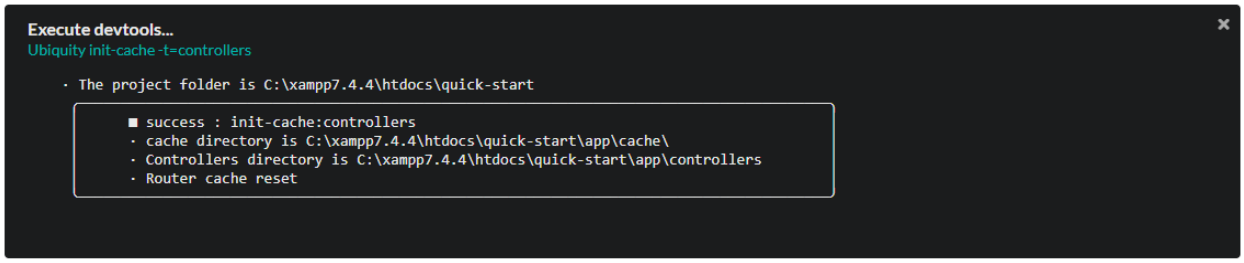
Es posible obtener ayuda sobre un comando (lo que produce un resultado equivalente a `Ubiquity help cmdName`).

40.1.3 Ejecución de comandos

Al pulsar el botón de ejecución de un comando, aparece un formulario para introducir los parámetros (o lo ejecuta directamente si no necesita ninguno).



Tras introducir los parámetros, la ejecución produce un resultado.

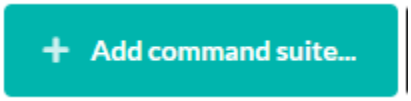


40.2 Grupo de comandos

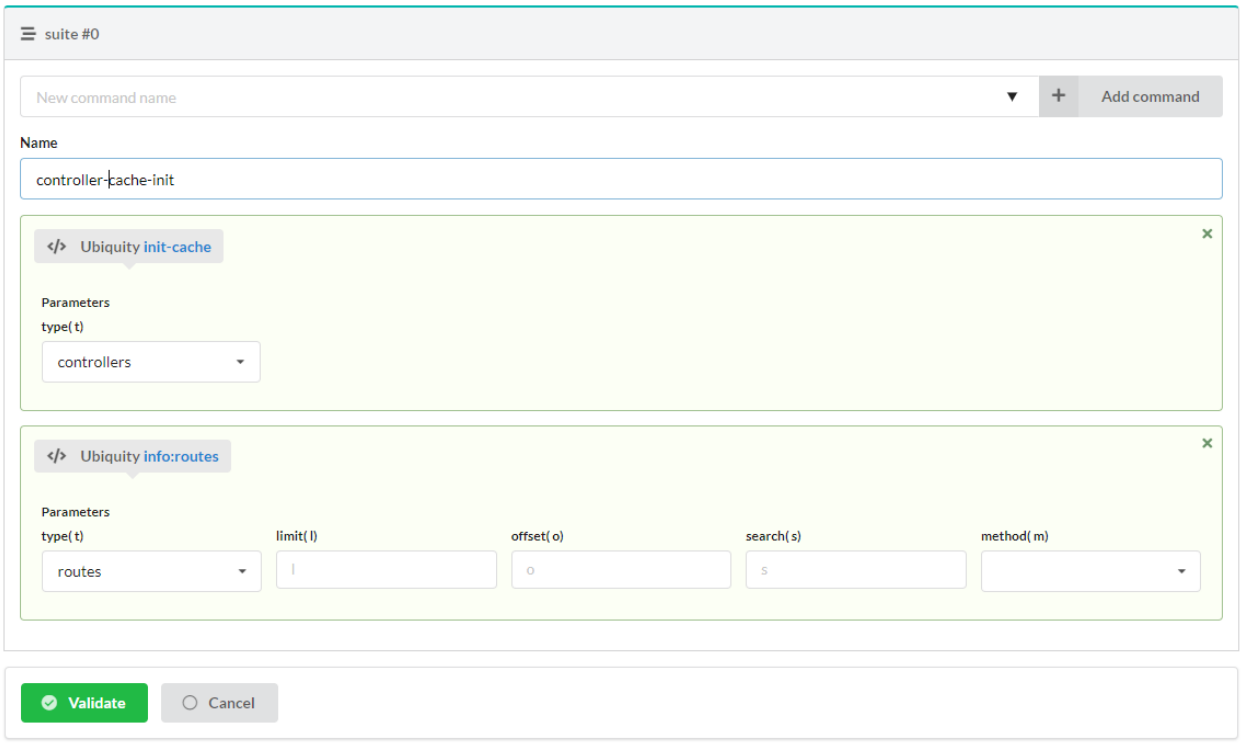
Volver a la pestaña **Mis comandos**: Es posible guardar una secuencia de comandos (con parámetros almacenados), y luego ejecutar la misma secuencia:

40.2.1 Creación de grupos

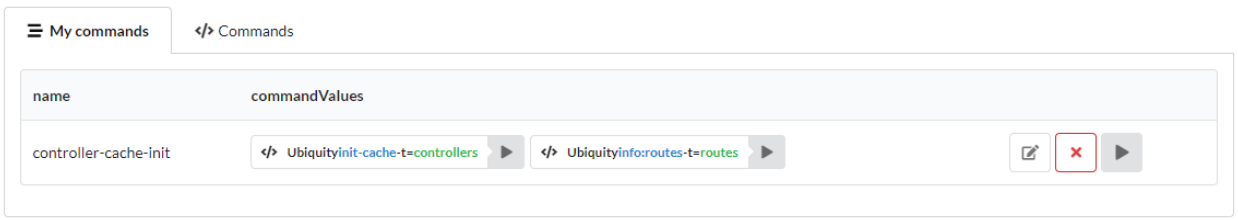
Haga clic en **añadir grupo de comandos*.



Añada los comandos deseados y modifique los parámetros:

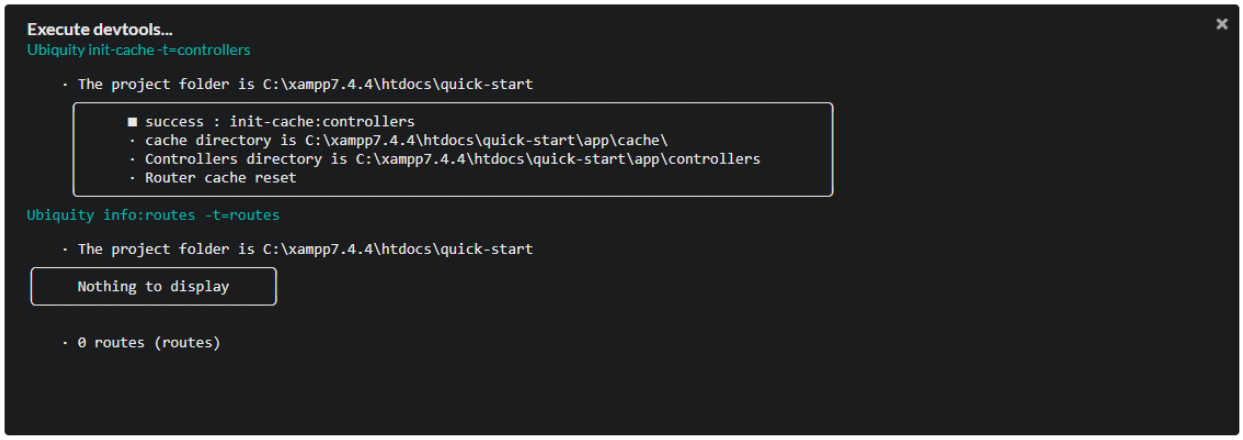


La validación genera el grupo:



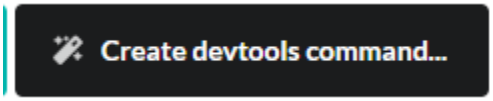
40.2.2 Ejecución del grupo de comandos

Al hacer clic en el botón de ejecución del grupo, se ejecuta la lista de comandos que contiene:



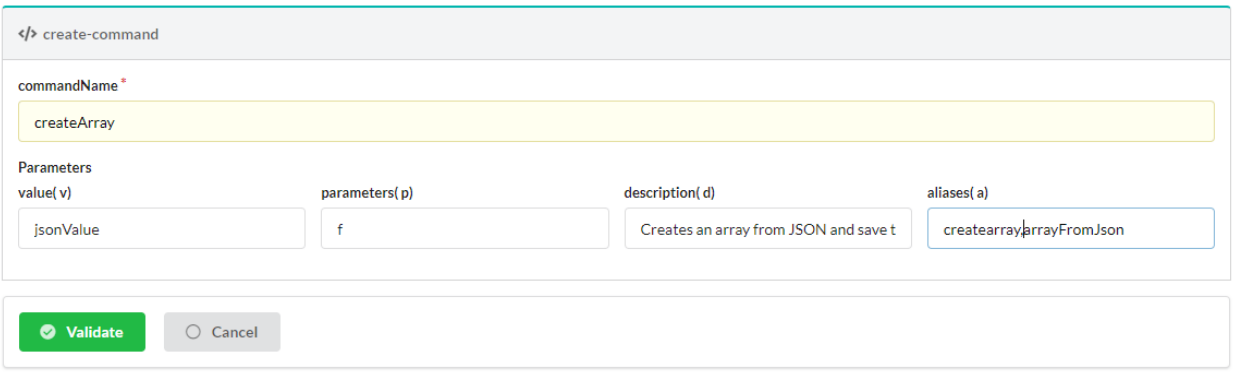
40.3 Creación de comandos personalizados

Haga clic en el botón **Crear comando devtools**.

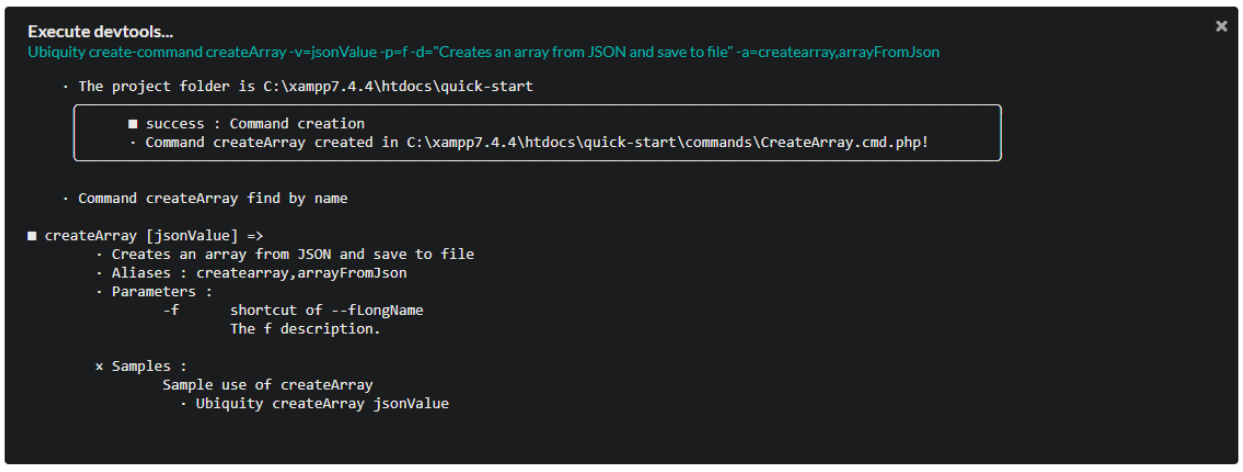


Introduzca las características del nuevo comando:

- Nombre del comando
- El valor del comando: nombre del argumento principal
- Los parámetros del comando: En caso de varios parámetros, utilice la coma como separador
- Descripción del comando
- Alias de los comandos: En caso de varios alias, utilice una coma como separador



Nota: Los comandos personalizados se crean en la carpeta **app/commands** del proyecto.



La clase generada:

Lista 1: app/commands/CreateArray.php

```php
namespace commands;

use Ubiquity\devtools\cmd\commands\AbstractCustomCommand;
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');
	}
}
```

El comando **CreateArray** implementado:

Lista 2: app/commands/CreateArray.php

```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 {

```

(continué en la próxima página)

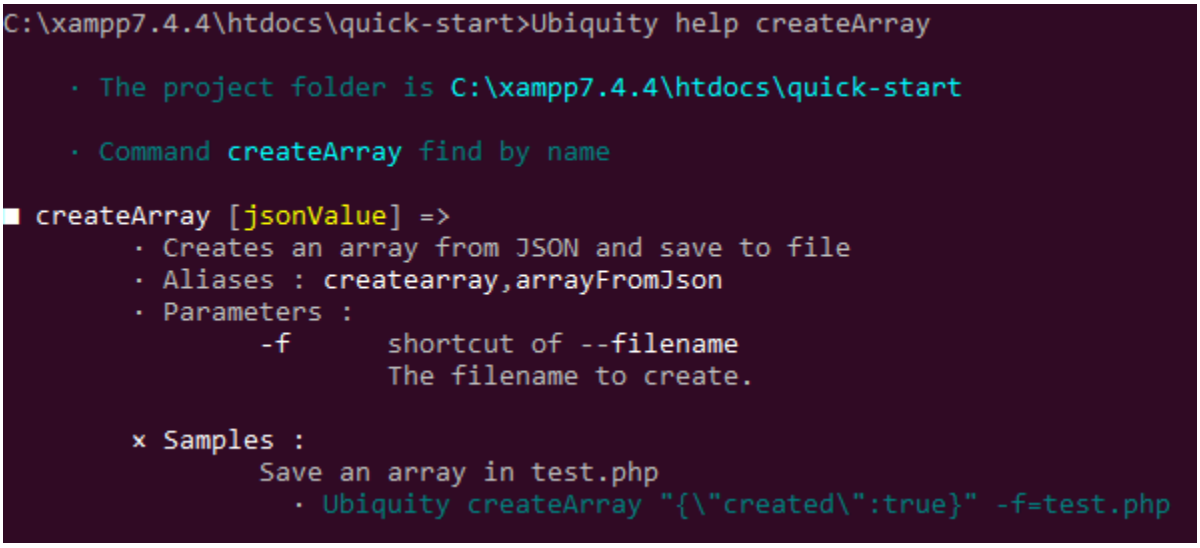
(proviene de la página anterior)

```
    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('filename', 'The filename to create.', [])
            ];
    }

    protected function getExamples(): array {
            return [
                    'Save an array in test.php' => "Ubiquity createArray \"{\\\"created\
↪\\":true}\" -f=test.php"
            ];
    }

    protected function getDescription(): string {
            return 'Creates an array from JSON and save to file';
    }

    public function run($config, $options, $what, ...$otherArgs) {
            echo ConsoleFormatter::showInfo('Run createArray command');
            $array = \json_decode($what, true);
            $error = \json_last_error();
            if ($error != 0) {
                    echo ConsoleFormatter::showMessage(\json_last_error_msg(), 'error');
            } else {
                    $filename = self::getOption($options, 'f', 'filename');
                    if ($filename != null) {
                            UFileSystem::save($filename, "<?php\nreturn " . var_export(
↪$array, true) . ";\n");
                            echo ConsoleFormatter::showMessage("$filename succefully␣
↪created!", 'success', 'CreateArray');
                    } else {
                            echo ConsoleFormatter::showMessage("Filename must have a␣
↪value!", 'error');
                    }
            }
    }
}
```

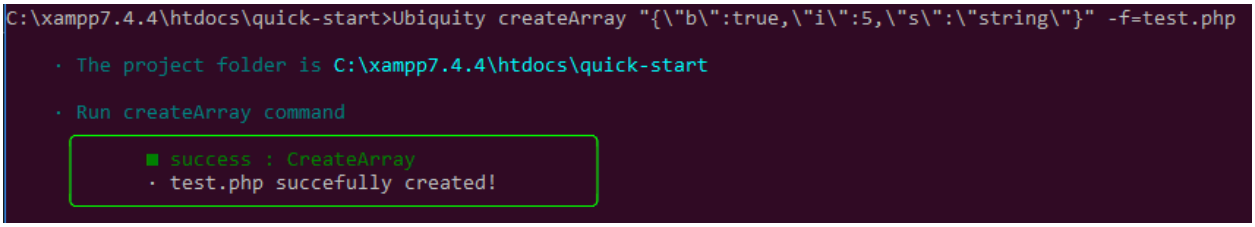
40.3.1 Ejecución de comandos personalizados

El nuevo comando es accesible desde las devtools, siempre que esté en el proyecto:

```
Ubiquity help createArray
```




```
Ubiquity createArray "{\"b\":true,\"i\":5,\"s\":\"string\"}" -f=test.php
```



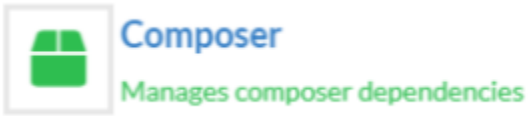
CAPÍTULO 41

Gestión de composer

Nota: Esta parte es accesible desde las **webtools**, así que si creaste tu proyecto con la opción **-a** o con el comando **create-project**..

41.1 Acceso

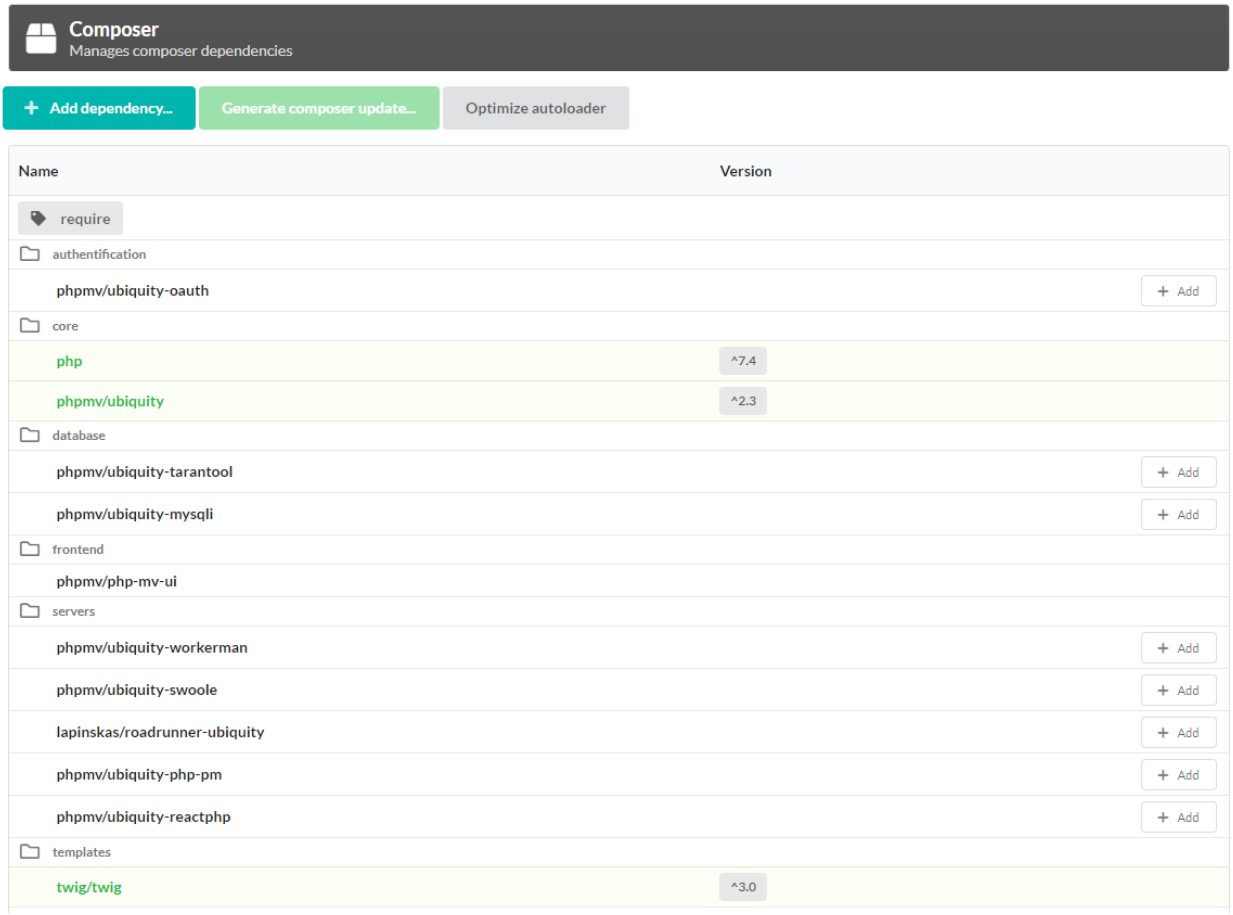
Desde las herramientas web, activa la parte **composer**,



o vaya directamente a `http://127.0.0.1:8090/Admin/composer`.

41.2 Lista de dependencias

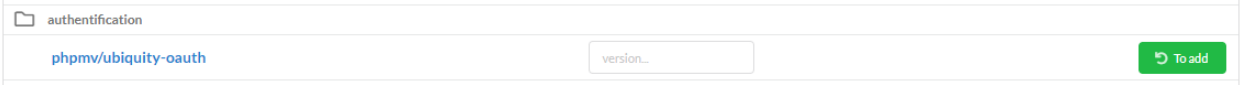
La interfaz muestra la lista de dependencias ya instaladas y las que se pueden instalar directamente.



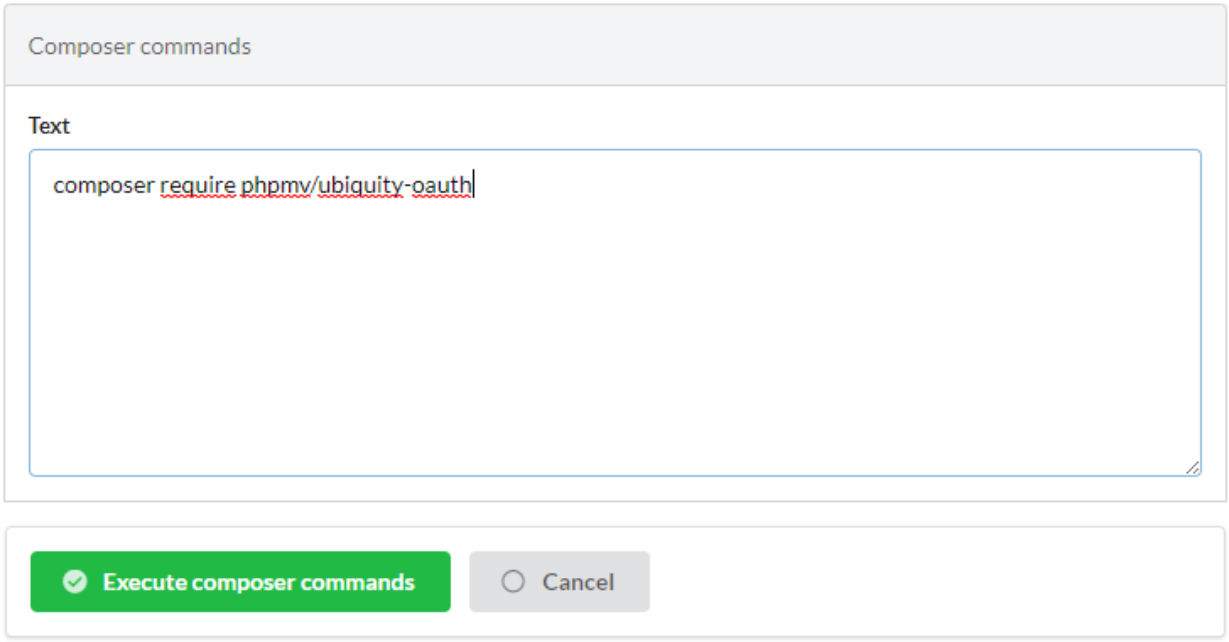
41.3 Instalación de dependencias

41.3.1 Lista de dependencias enumeradas:

Haga clic en el botón **añadir** de las dependencias que desee añadir.



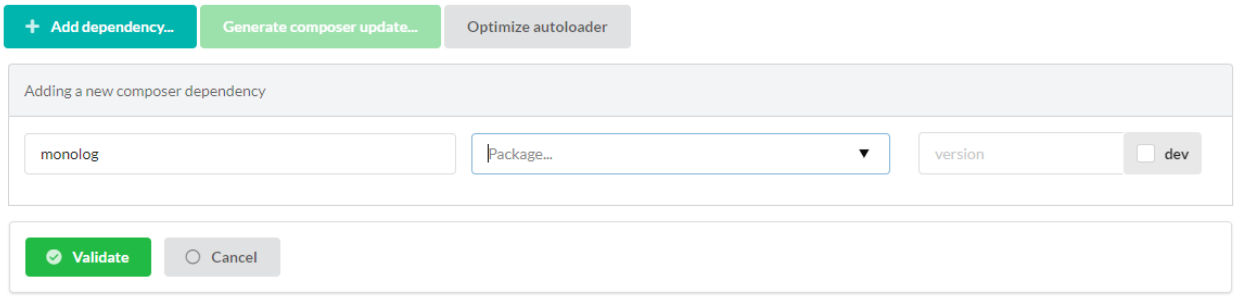
A continuación, haga clic en el botón **Generar actualización del compositor**:



La validación genera la actualización.

41.3.2 Para las dependencias no enumeradas:

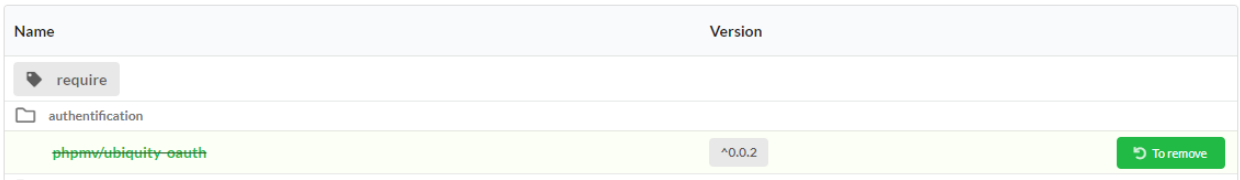
Pulse el botón **Añadir dependencia** :



- Introduzca nombre del vendor (proveedor) ;
- Seleccione un paquete de la lista ;
- Seleccione eventualmente una versión (si no hay ninguna, se instalará la última versión estable).

41.4 Eliminación de la dependencia

Haga clic en el botón **remove** de las dependencias que desee añadir.



A continuación, haga clic en el botón **Generar actualización de composer** y valide la actualización.

Nota: Es posible realizar varias operaciones de adición o supresión y validarlas simultáneamente.

41.5 Optimización de composer

Haga clic en el botón **Optimize autoloader**.

Esto optimiza la autocarga del composer con un mapa de clase autorizado.

CAPÍTULO 42

Caché de Ubiquity

CAPÍTULO 43

Dependencias en Ubiquity

- `php >=7.4`
- `phpmv/ubiquity` => Ubiquity core

43.1 En producción

43.1.1 Plantillas

Twig es necesario si se utiliza como motor de plantillas, que no es un requisito.

- `twig/twig` => Template engine

43.1.2 Seguridad

- `phpmv/ubiquity-security` => Csrf, Csp…
- `phpmv/ubiquity-acl` => Gestión de listas de control de acceso

43.2 En desarrollo

43.2.1 Webtools

- `phpmv/ubiquity-dev` => clases dev para webtools y devtools desde v2.3.0
- `phpmv/php-mv-ui` => Front library
- `mindplay/annotations` => Biblioteca de anotaciones, necesaria para generar modelos…
- `monolog/monolog` => Logging
- `czproject/git-php` => Operaciones Git (+ requiere consola git)

43.2.2 Devtools

- `phpmv/ubiquity-devtools` => Consola Cli
- `phpmv/ubiquity-dev` => clases dev para webtools y devtools desde v2.3.0
- `mindplay/annotations` => Biblioteca de anotaciones, necesaria para generar modelos…

43.2.3 Pruebas

- `codeception/codeception` => Pruebas
- `codeception/c3` => Integración C3
- `phpmv/ubiquity-codeception` => Codeception for Ubiquity

CAPÍTULO 44

Módulo cliente OAuth2

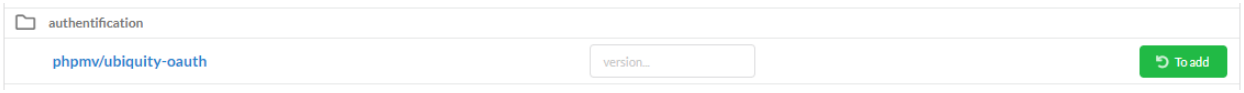
Nota: Esta parte es accesible desde las **webtools**, por lo que si creaste tu proyecto con la opción **-a** o con el comando **create-project**. El módulo OAuth no está instalado por defecto. Utiliza la librería HybridAuth.

44.1 Instalación

En la raíz de tu proyecto:

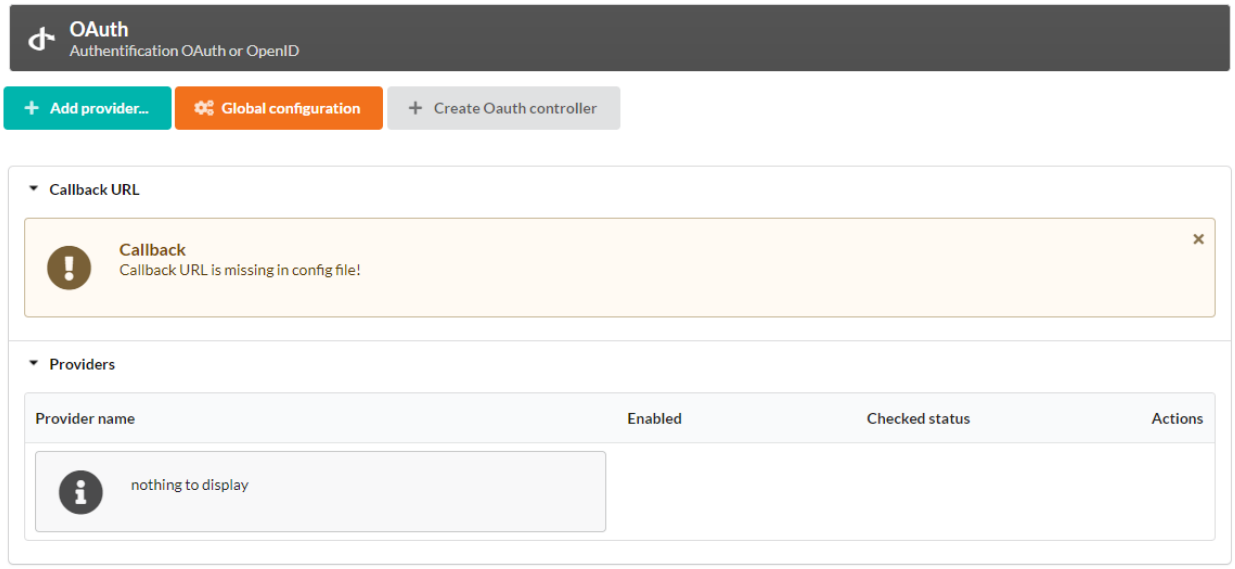
```
composer require phpmv/ubiquity-oauth
```

Nota: También es posible añadir la dependencia **ubiquity-oauth** utilizando la parte **Composer** del módulo de administración.

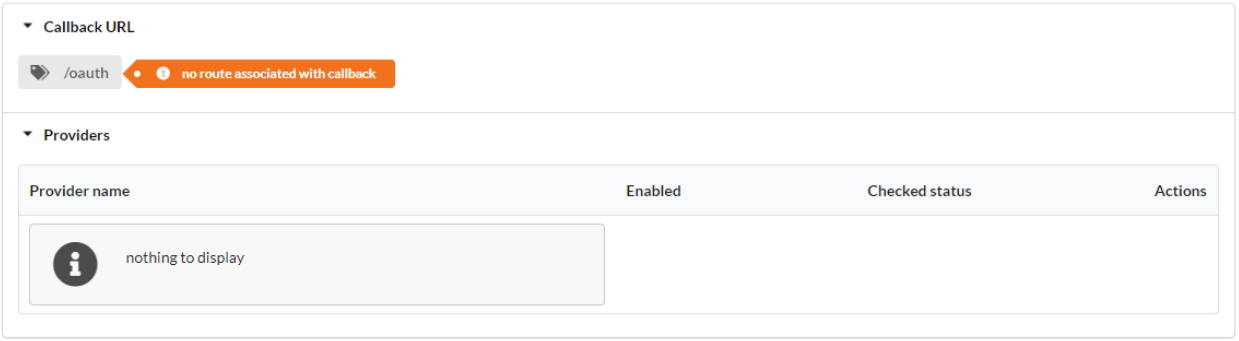


44.2 Configuración de OAuth

44.2.1 Configuración global

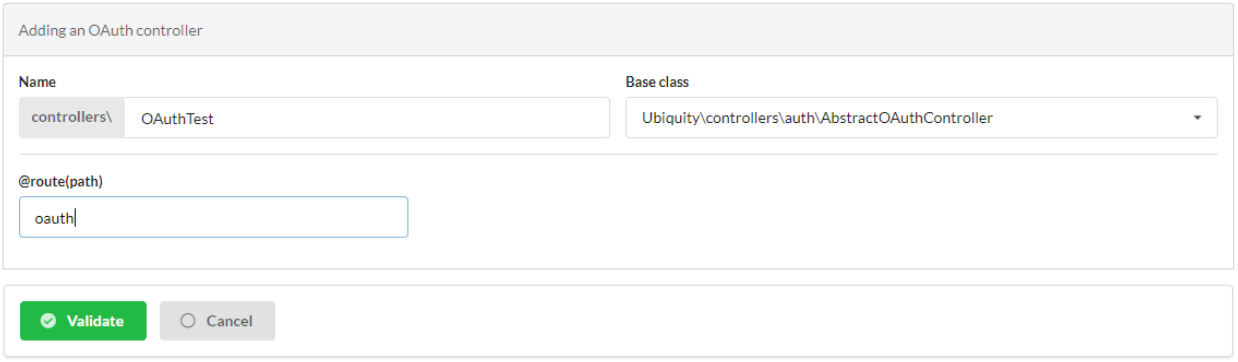


Haga clic en el botón **Global configuration** y modifique la URL de devolución de llamada, que corresponde a la url de devolución de llamada local tras una conexión correcta.

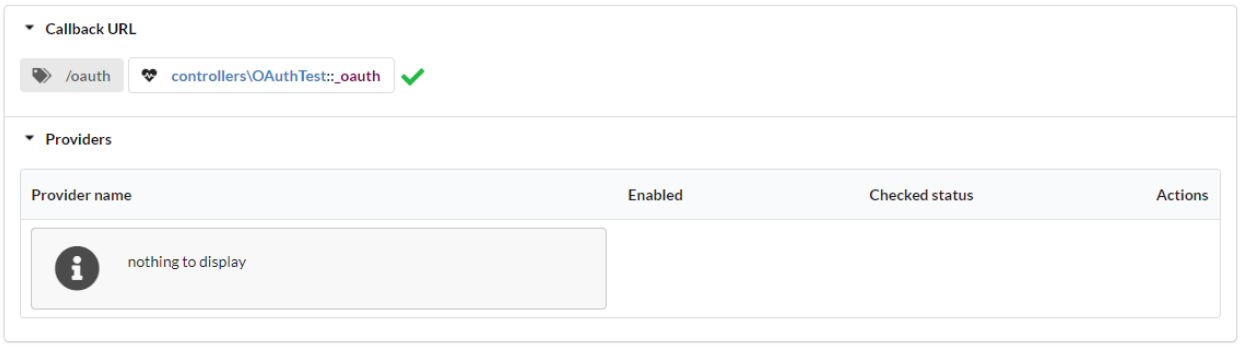


44.2.2 Controlador OAuth

Haga clic en el botón **Create Oauth controller** y asigne a la ruta el valor dado previamente a la devolución de llamada:



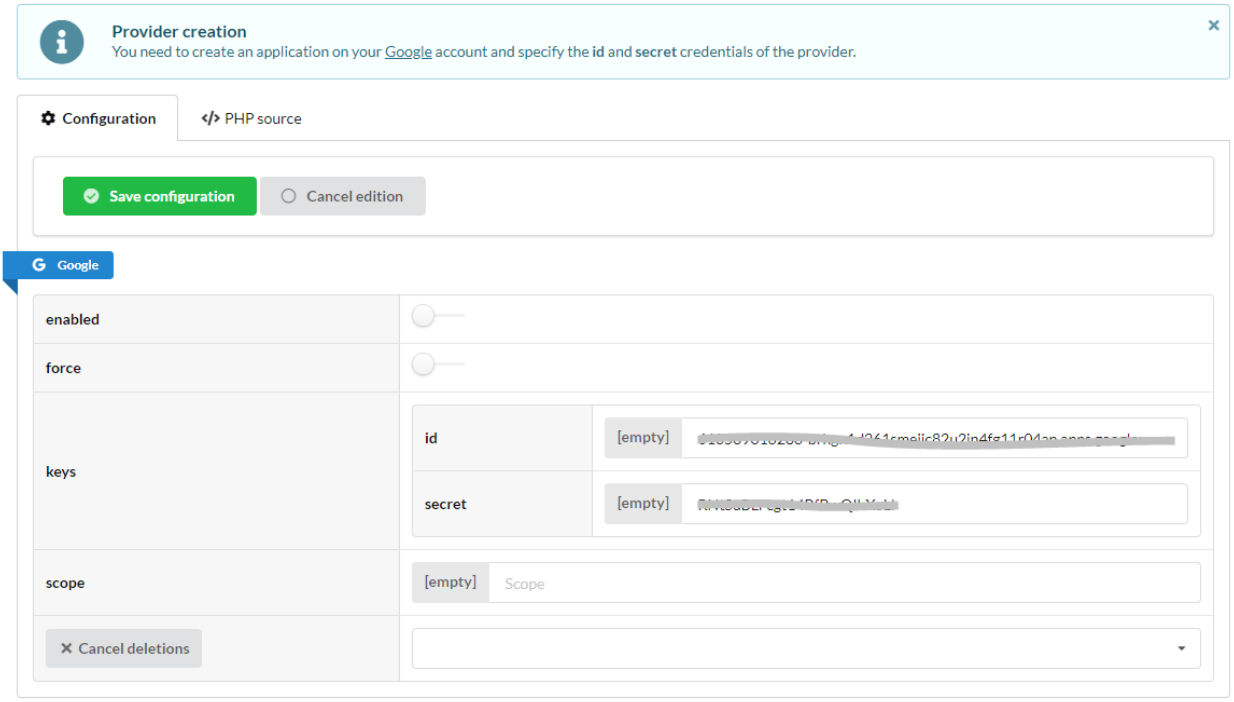
Validar y restablecer la caché del router:



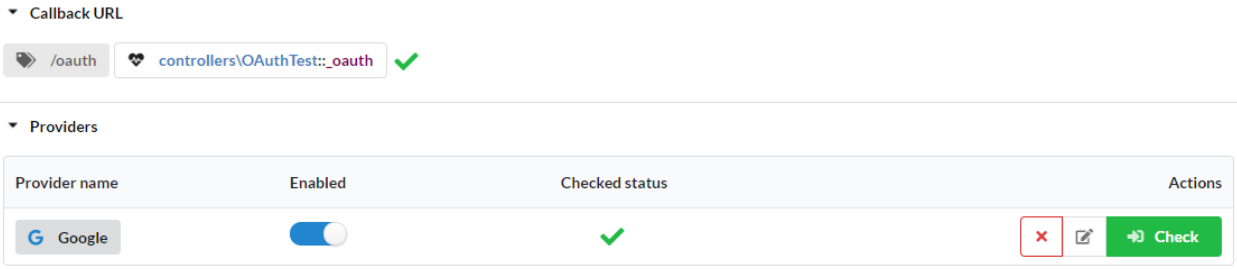
44.2.3 Proveedores

Nota: Para una autenticación OAuth, es necesario crear previamente una aplicación en el proveedor, y tomar nota de las claves de la aplicación (id y secret).

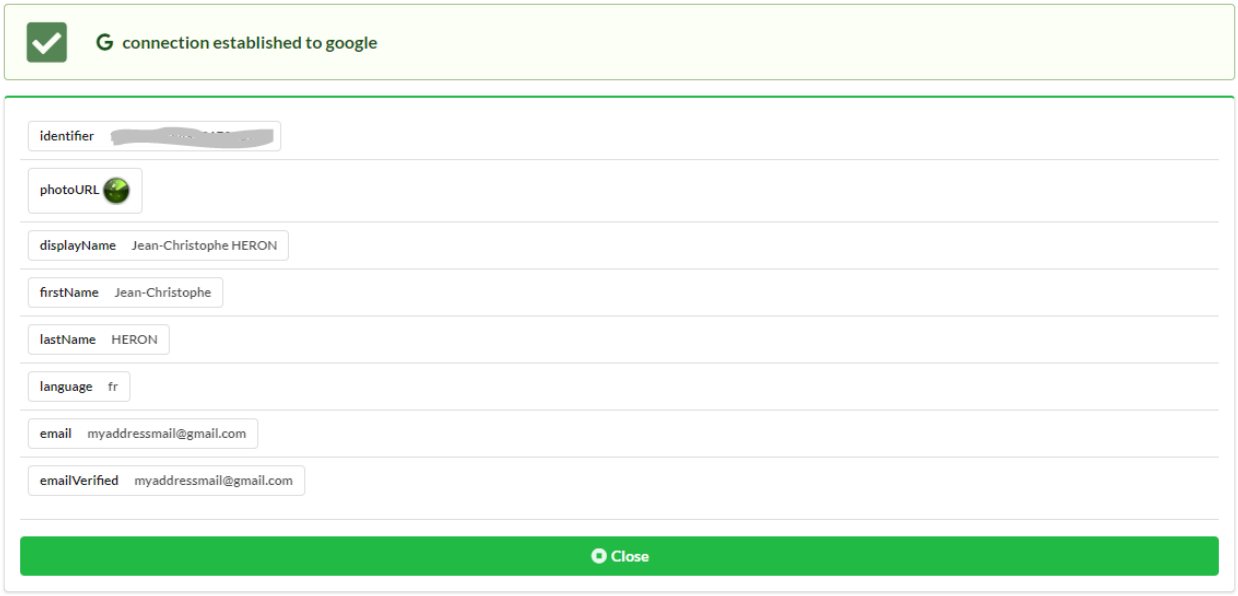
Haga clic en el botón **Añadir proveedor** y seleccione **Google**:



Compruebe la conexión pulsando el botón **Check**:



Publicar información de inicio de sesión:



44.3 Personalización de OAuthController

El controlador creado es el siguiente:

Lista 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
    }
}
```

- El método **_oauth** corresponde a la url de devolución de llamada
- El método **onConnect** se activa en la conexión y puede ser anulado.

Ejemplo :

- Posible recuperación de un usuario asociado en la base de datos
- o creación de un nuevo usuario
- Adición del usuario conectado y redirección

Lista 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:/');
  }
```

CAPÍTULO 45

Plataformas asíncronas

Nota: Ubiquity soporta múltiples plataformas : Swoole, Workerman, RoadRunner, PHP-PM, ngx_php.

45.1 Swoole

Instala la extensión Swoole en tu sistema (linux) o en tu imagen Docker :

```
#!/bin/bash
pecl install swoole
```

Ejecute Ubiquity Swoole (por primera vez, se instalará el paquete **ubiquity-swoole**):

```
Ubiquity serve -t=swoole
```

45.1.1 Configuración del servidor

Lista 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
        ]
);
```

El puerto también puede cambiarse al iniciar el servidor:

```
Ubiquity serve -t=swoole -p=8999
```

45.1.2 Optimización de los servicios

El inicio de los servicios se realizará una sola vez, al arrancar el servidor.

Lista 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();
});
```

El método warmUpControllers:

- instanciar los controladores
- realiza la inyección de dependencias
- prepara la llamada de los métodos initialize y finalize (inicialización de las constantes de llamada)

Al inicio de cada Worker, el método **warmup** de los controladores puede, por ejemplo, inicializar consultas DAO preparadas:

Lista 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 no requiere ninguna instalación especial (excepto **libevent** para ser utilizado en producción por razones de rendimiento).

Ejecute Ubiquity Workerman (por primera vez, se instalará el paquete **ubiquity-workerman**):

```
Ubiquity serve -t=workerman
```

45.2.1 Configuración del servidor

Lista 4: .ubiquity/workerman-config.php

```
<?php
return array(
    "host" => "0.0.0.0",
    "port" => 8080,
    "socket"=>[
        "count" => 4,
        "reuseport" =>true
    ]
);
```

El puerto también puede cambiarse al iniciar el servidor:

```
Ubiquity serve -t=workerman -p=8999
```

45.2.2 Optimización de los servicios

El inicio de los servicios se realizará una sola vez, al arrancar el servidor.

Lista 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

CAPÍTULO 46

Índices y tablas

- genindex
- modindex
- search