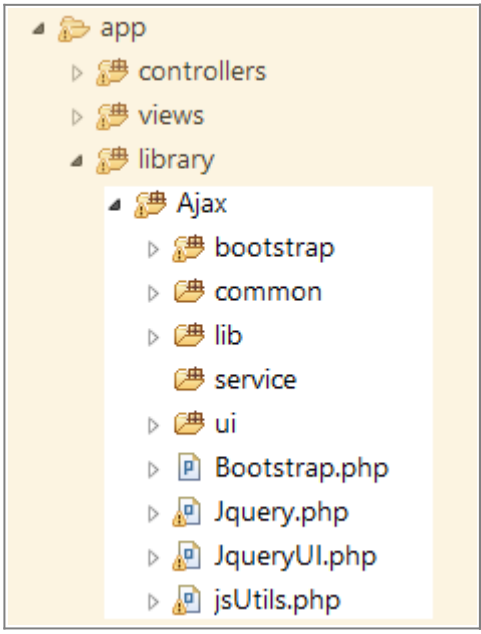
Phalcon-JQuery

Phalcon-JQuery est une librairie PHP compatible avec le Framework Phalcon. Elle permet d'intégrer facilement JQuery, JQuery UI ou Twitter Bootstrap dans les projets Web Phalcon, en respectant la séparation des couches MVC.

-- Installation

1. Télécharger Phalcon-JQuery sur gitHub
2. Dé-zipper dans un dossier local, et copier le dossier Ajax dans le dossier correspondant à ma variable **libraryDir** définie dans la configuration de votre projet Phalcon :




```
    'application' => array(
        'controllersDir' => __DIR__ .
'/../../app/controllers/',
        'modelsDir'      => __DIR__ .
'/../../app/models/',
        'viewsDir'       => __DIR__ .
'/../../app/views/',
        'pluginsDir'     => __DIR__ .
'/../../app/plugins/',
        'libraryDir'     => __DIR__ .
'/../../app/library/',
        'cacheDir'       => __DIR__ .
'/../../app/cache/',
        'baseUri'        => '/atp/',
    )
</config>
```

-- Configuration

-- Injection du service

Pour ajouter Phalcon-JQuery à un projet Phalcon, il est nécessaire d'injecter le service JQuery au démarrage de

l'application :

```
$di->set("jquery",function(){
    $jquery= new JsUtils(array("driver"=>"Jquery"));
    $jquery->ui(new JqueryUI());//optional for JQuery UI
    $jquery->bootstrap(new Bootstrap());//Optional for Twitter Bootstrap
    return $jquery;
});
```

-- Intégration des librairies JS

Pour fonctionner correctement, les librairies JS suivantes doivent être insérées dans les pages HTML principales (non sollicitées via ajax).

Librairie activée	**Fichier JS**	**Fichier css**
JQuery	jquery-min.js	
JQuery UI*	jquery-ui-min.js	jquery-ui.css
Twitter Bootstrap*	bootstrap.min.js	bootstrap.min.css

* optionnel

-- Intégration manuelle des fichiers

Attention de veiller à inclure **JQuery** avant les librairies qui en dépendent : Jquery UI ou Bootstrap.

En utilisant des CDN

```
<html>
    <head>
    <?php
    // JQuery
    echo
Phalcon\Tag::javascriptInclude("http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/j
query.min.js", false);
    //jquery ui
    echo
Phalcon\Tag::javascriptInclude("https://ajax.googleapis.com/ajax/libs/jqueryui/1.11
.2/jquery-ui.min.js", false);
    //Thème Humanity for UI
    echo
Phalcon\Tag::stylesheetLink("https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.2/
themes/humanity/jquery-ui.css", false);

?>
</head>
```

En copiant les librairies JS et Css localement sur le serveur

```
<html>
    <head>
    <?php
    // JQuery
    echo Phalcon\Tag::javascriptInclude("js/jquery.min.js");
    //jquery ui
    echo Phalcon\Tag::javascriptInclude("js/jquery-ui.min.js");
    //Thème Humanity for UI
    echo Phalcon\Tag::stylesheetLink("css/jquery-ui.css");

?>
</head>
```

-- Intégration automatisée

Il est possible de récupérer directement via les classes CDN* les adresses des librairies et de générer automatiquement leur inclusion en suivant les étapes ci-dessous :

-- Paramétrage d'un contrôleur

Pour chaque action sollicitée par le contrôleur, on passe à la vue une variable **jqueryCDN** contenant les liens CDN à générer : Sans autre précisions, la méthode **genCDNs** génère l'insertion des liens en prenant les versions les plus récentes des librairies nécessaires : Si le service **jquery** injecté au démarrage de l'application déclare **JQuery UI**, les fichiers JS et CSS de UI seront insérés, de même pour **Bootstrap**.

```
class JoueurController extends ControllerBase{
    public function initialize(){
        $this->view->setVar("jqueryCDN", $this->jquery->genDCNs("humanity"));
    }
...
```

-- Insertion dans la/es vue/s

```
<html>
    <head>
    <?php
    echo $jqueryCDN;
    ?>
...
```

Résultat produit avec JQuery UI (Humanity template)+ Bootstrap activés :

```
<head>
...
    <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.2/jquery-ui.min.js"></scri
pt>
    <link rel="stylesheet" type="text/css"
href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.2/themes/humanity/jquery-
ui.css">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css">
    <script type="text/javascript"
src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js/bootstrap.min.js"></script>
...
</head>
```

-- Utilisation

Le principe d'utilisation est toujours le même, que ce soit avec **JQuery**, **JQuery UI** ou **Twitter Bootstrap** :

- Les composants et leur logique applicative (association de code aux événements) sont créés dans le contrôleur
- Le contrôleur les transmet à la vue
- La vue affiche et/ou génère le script côté client

-- Exemple JQuery

-- Pré-requis

Il est nécessaire d'injecter le service JQuery au démarrage de l'application :

```
$di->set("jquery",function(){
    $jquery= new JsUtils(array("driver"=>"Jquery"));
    return $jquery;
});
```

Il s'agit par exemple d'effectuer une requête Ajax vers la page **exemple/reponse** de notre site, sur le click du bouton d'id **btn**, et d'afficher la réponse dans la zone d'id **panelReponse** :

-- Logique applicative dans le contrôleur

```
class ExempleController extends \Phalcon\Mvc\Controller

    public function ajaxSampleAction(){
        $this->jquery->getAndBindTo("#btn", "click",
"exemple/reponse","#panelReponse");
```

```
        $this->jquery->compile($this->view);
    }
...
```

-- Affichage et Intégration du script généré dans la vue

La vue associée à notre action (**app/views/Exemple/ajaxSample.volt**) récupère la variable $script_foot, générée par l'appel de la méthode **jquery→compile** appelée dans le contrôleur :

```
<button id="btn">Lancer la requête</button>
<div id="panelReponse">Réponse non reçue</div>
{{script_foot}}
```

-- Création de la réponse

Pour l'instant, le click sur le bouton **btn** génère une réponse avec erreur 404, puisque l'action sollicitée n'existe pas : il faut donc créer l'action **reponse** dans le contrôleur **Exemple** :

```
class ExempleController extends \Phalcon\Mvc\Controller
...
    public function reponseAction(){
        echo "Réponse bien reçue";
        $this->view->disable();
    }
...
```

-- Exemple JQuery UI

-- Pré-requis

Il est nécessaire d'injecter le service JQuery au démarrage de l'application, et d'instancier **JQuery UI** :

```
$di->set("jquery",function(){
    $jquery= new JsUtils(array("driver"=>"Jquery"));
    return $jquery;
});
```

Il s'agit par exemple d'effectuer une requête Ajax vers la page **exemple/reponse** de notre site, sur le click du bouton d'id **btn**, et d'afficher la réponse dans la zone d'id **panelReponse** :

-- Logique applicative dans le contrôleur

```

```

```
class ExempleController extends \Phalcon\Mvc\Controller

    public function ajaxSampleAction(){
        $this->jquery->getAndBindTo("#btn", "click",
"exemple/reponse","#panelReponse");
        $this->jquery->compile($this->view);
    }
...
```

-- Affichage et Intégration du script généré dans la vue

La vue associée à notre action (**app/views/Exemple/ajaxSample.volt**) récupère la variable $script_foot, générée par l'appel de la méthode **jquery→compile** appelée dans le contrôleur :

```
<button id="btn">Lancer la requête</button>
<div id="panelReponse">Réponse non reçue</div>
{{script_foot}}
```

-- Création de la réponse

Pour l'instant, le click sur le bouton **btn** génère une réponse avec erreur 404, puisque l'action sollicitée n'existe pas : il faut donc créer l'action **reponse** dans le contrôleur **Exemple** :

```
class ExempleController extends \Phalcon\Mvc\Controller
...
    public function reponseAction(){
        echo "Réponse bien reçue";
        $this->view->disable();
    }
...
```