TD n°4 - Scripts côté client, Ajax, JSON, composants

Institut Universitaire de Technologie
Département Informatique - Campus III Ifs
Janvier-Février 2017

-- Pré-requis

Réalisation TD n°3
phpMv-UI toolkit

-- Installation

1. Créer un nouveau projet Phalcon avec les devtools
2. Intégrer la librairie phpMv-UI avec composer
3. Injectez le service **JQuery** au démarrage de l'application (dans le fichier **services.php**) et instancier **semantic**
4. Intégrez le fichier javascript JQuery dans le fichier views/index.volt jQuery download
5. Intégrez Semantic-UI pour la partie présentation

-- Tests

Créer un contrôleur **testController**

-- Afficher/masquer des éléments

Composants à utiliser :

- HtmlCheckbox
- HtmlMessage ou HtmlSegment pour **#zone**

- Créer la vue **views/test/hideShow.volt** sur le modèle suivant :

- Créer une action **hideShowAction** incorporant le comportement suivant dans la vue :
  - Sur Changement de valeur de la case à cocher, la div **#zone** s'affiche si la case est cochée, et se masque dans le cas contraire

-- Requêtes Ajax Get sur event

- Créer la vue **views/test/changeCss.volt** sur le modèle suivant :

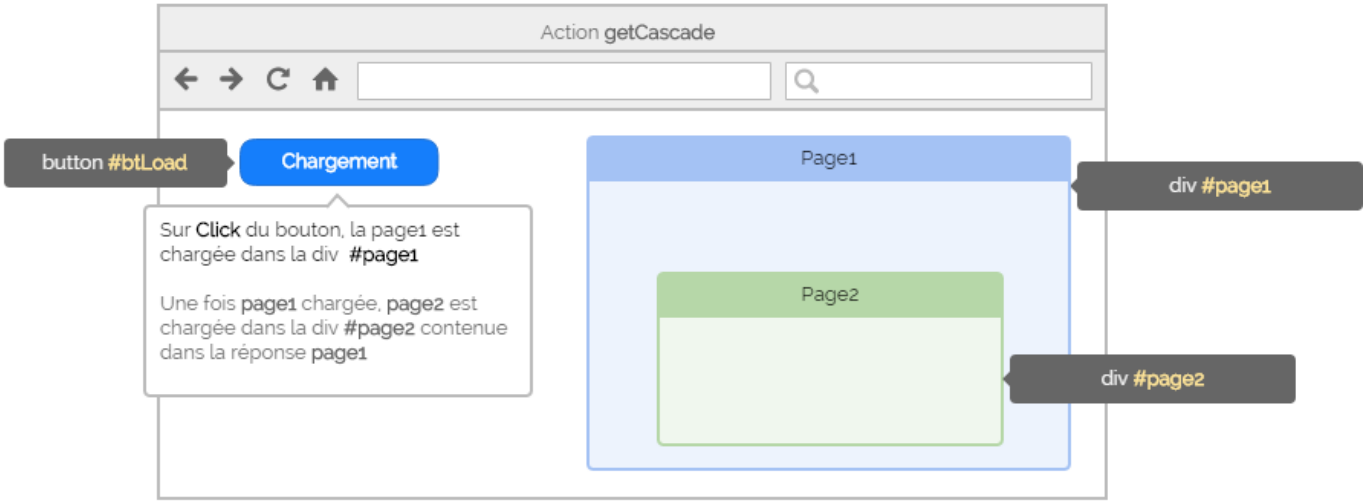
- Créer une action **getSimpleAction** incorporant le comportement suivant dans la vue :
  - Sur click du bouton **#btPage1**, la page **/test/page1/** est affichée dans la div **#pageContent**
  - Sur click du bouton **#btPage2**, la page **/test/page2/** est affichée dans la div **#pageContent**
- Créer les actions page1Action et page2Action, se limitant à afficher “Contenu de la page #”, désactiver l'affichage de la vue correspondante

-- MouseOver

- Dans getSimpleAction, ajouter le comportement suivant :
  - sur **mouseover** des bouton, son attribut **data-description** est affiché dans la zone **#pageDesc**

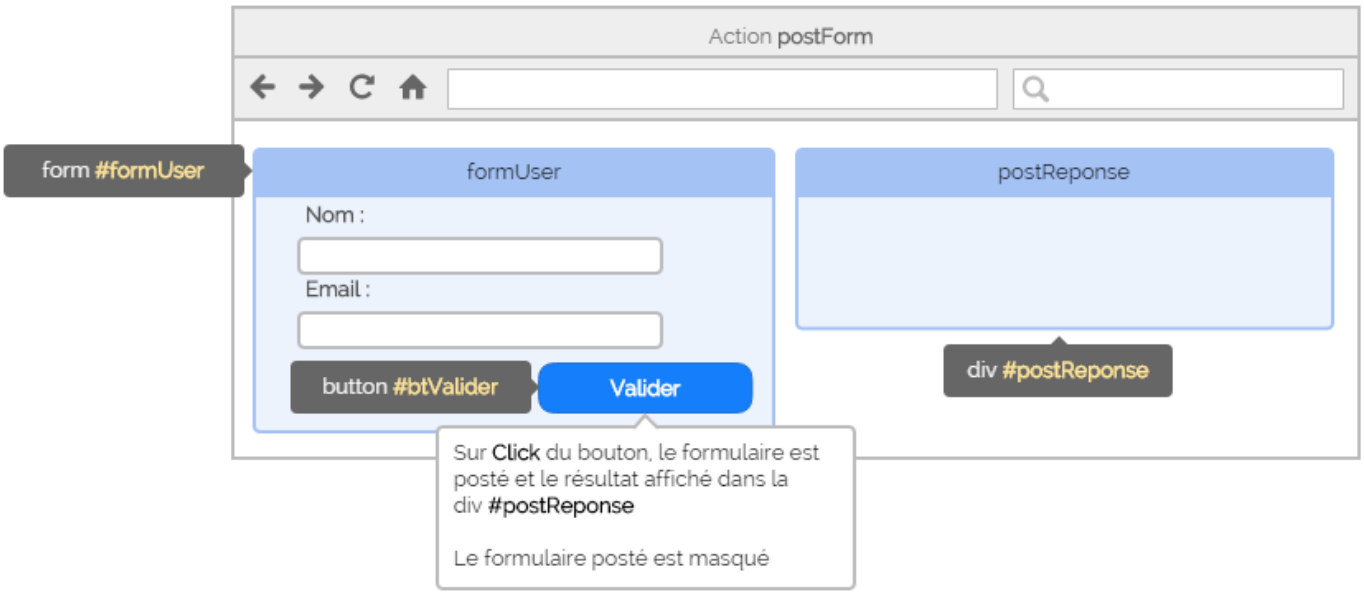
-- Requêtes en cascade

Créer les actions **page1Action**, **page2Action** et les vues (/views/test/page1 et /views/test/page2) correspondant au fonctionnement décrit ci dessous :



-- POST de formulaire

- Créer l'action **postFormAction**, affichant le formulaire affiché dans la vue **views/test/postForm**, ajouter le comportement défini ci-dessous
- Créer l'action **postReponseAction** affichant la réponse du post du formulaire



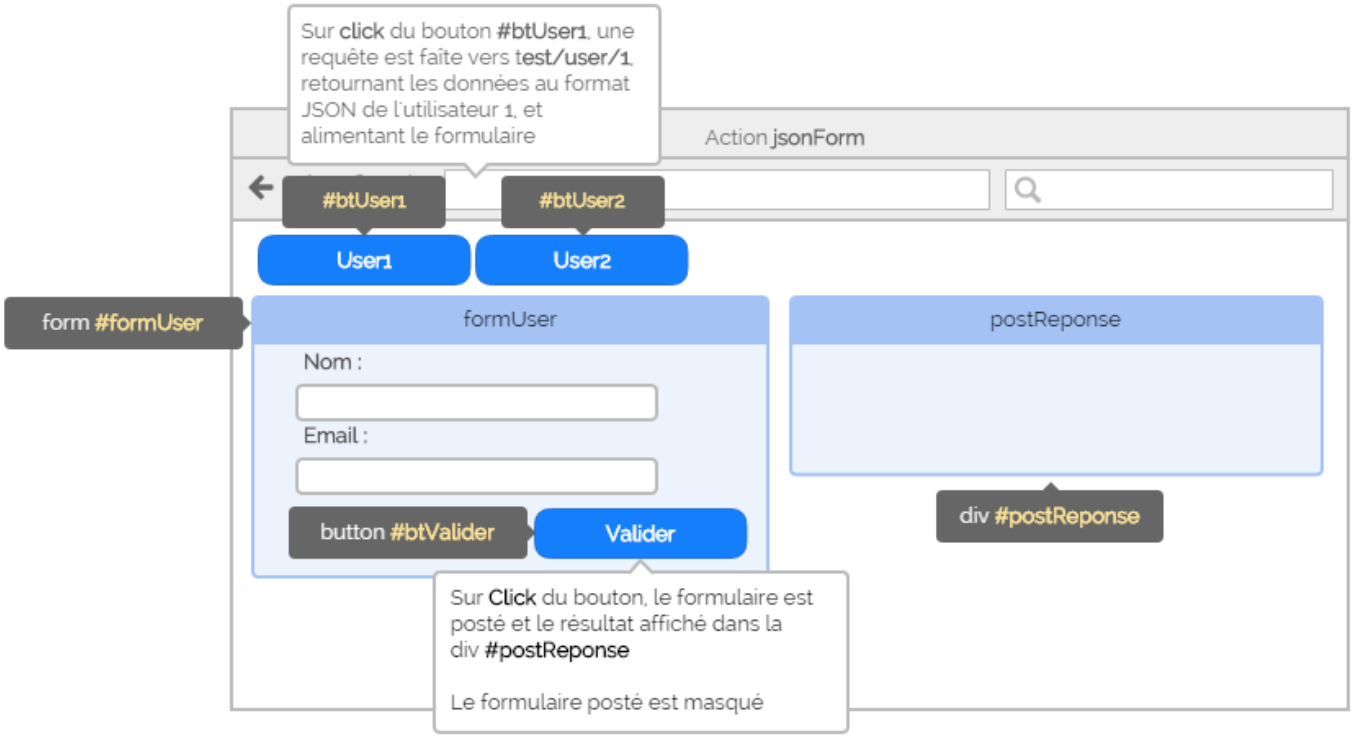
-- Requête de récupération de données JSON

On ajoute au formulaire précédent la possibilité de récupérer via JSON les données initiales du formulaire à afficher :

- Le **btUser1** permet de récuprérer et d'afficher les données de **l'utilisateur1**
- De même le **btUser2** pour l'**utilisateur2**

Créer l'action **userAction** retournant la partie du tableau php contenant les données JSON correspondant à l'utilisateur dont l'index est passé en paramètre :

```
$userArray=array(
     '{"nom":"SMITH","email":"BSMITH@mail.com"}',
     '{"nom":"DOE","email":"jdoe@mail.com"}'
     );
```



-- Application

-- Model

1. Effectuer la connexion à la base de données **Invo**.
2. Générer le model **Products** correspondant à la table **products** de la base de données (respectant l'encapsulation)

-- Contrôleur et Vues

Dans un contrôleur **productsController**

1. Créer l'action **index** et la vue permettant d'afficher la liste des produits (name et type)
2. Sur click d'un produit, le détail du produit est affiché dans la div **#productDetails** (via AJAX + JSON)
3. Le Bouton **#btUpdate** Modifier permet alors d'afficher le formulaire de modification du produit dans la zone **#productForm** (affichage des données via AJAX + JSON)
4. La validation du formulaire (**#btSubmit**) actualise la ligne du tableau modifiée (AJAX)
5. Le double click sur une ligne provoque le même effet que le simple click + le click sur Modifier

Les divs **#productDetails** et **#productForm** et leur squelette doivent être présents dans la page au chargement de la page index