ISEN OUEST

Projet - Communications Web
Version N°3 - REST

Thibault Napoléon
thibault.napoleon@isen-ouest.yncrea.fr

1 Structure *HTML*

Dans cette partie vous allez écrire le code *HTML* de la `div` d'id `comments-add` permettant d'ajouter un commentaire. En vous basant sur la documentation *Bootstrap* des `input-group` (cliquez ici pour la documentation) et des `boutons` (cliquez ici pour la documentation) réalisez la **structure** d'ajout de commentaires telle que visible sur la figure 1.

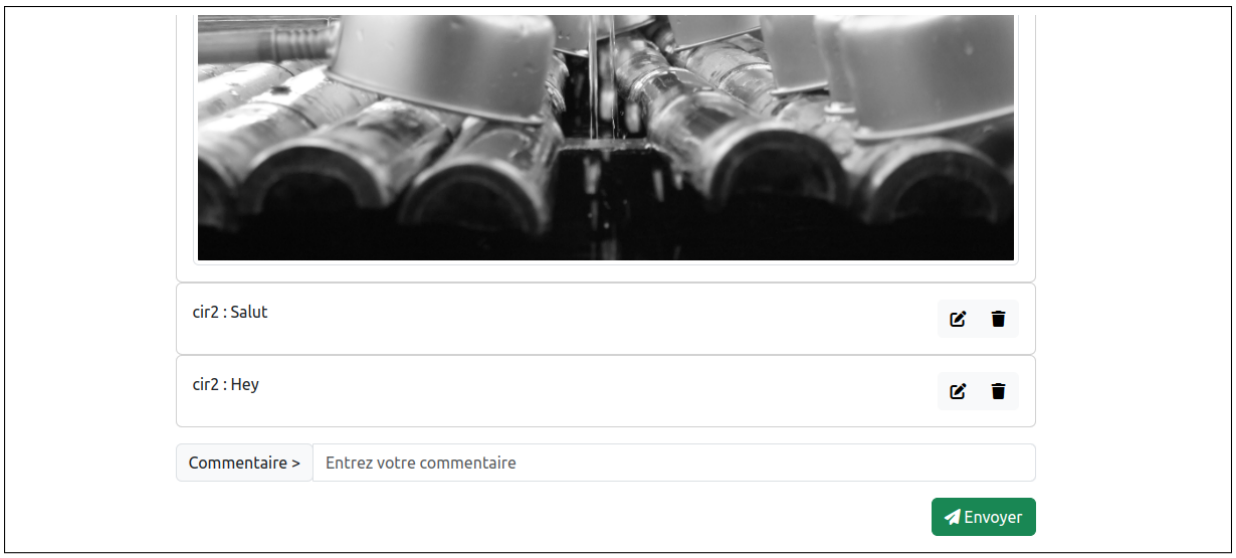

FIGURE 1 – *Exemple de visuel pour l'ajout de commentaires.*

En particulier, nous aurons besoin de la structure suivante :

```
<form>
  <div class="input-group">
    AJOUTEZ VOTRE CHAMP DE TEXTE ICI
  </div>
  AJOUTEZ VOTRE BOUTON ICI
</form>
```

Pensez à rajouter les id suivants :

- un id `add` à votre formulaire ;
- un id `comment` à votre champ de texte.

Pour une meilleure ergonomie, il est conseillé de rendre invisible la partie d'ajout des commentaires tant qu'aucune photo n'est sélectionnée. Pour cela, modifiez la déclaration de la `div comments-add` comme suit :

```
<div id="comments-add" style="display: none">
```

2 Traitement des « verbes » *REST* en *PHP*

Gestion de la base de données pour les commentaires en *PHP*

Pour commencer, écrivez dans le script « database.php » la fonction permettant de récupérer la liste des commentaires associés à une photo :

```
dbRequestComments($db, $photoId)
```

Où `db` est le lien *PDO* de la base de données et `photoId`, l'id de la photo pour laquelle les commentaires sont demandés. Cette fonction doit renvoyer la liste de tous les commentaires avec l'ensemble des informations disponibles.

Ensuite, écrivez la fonction permettant d'ajouter un commentaire à une photo :

```
dbAddComment($db, $userLogin, $photoId, $comment)
```

Où `db` est le lien *PDO* de la base de données, `userLogin` le nom de l'utilisateur qui commente, `photoId` l'id de la photo et `comment` le nouveau commentaire. Cette fonction doit renvoyer `true` en cas de succès, `false` sinon.

Pour permettre la modification d'un commentaire, écrivez la fonction suivante :

```
dbModifyComment($db, $userLogin, $id, $comment)
```

Où `db` est le lien *PDO* de la base de données, `userLogin` le nom de l'utilisateur à qui appartient le commentaire, `id` l'id du commentaire modifier et `comment` le nouveau texte du commentaire. Cette fonction doit renvoyer `true` en cas de succès, `false` sinon.

Enfin, pour permettre la suppression d'un commentaire, écrivez la fonction suivante :

```
dbDeleteComment($db, $userLogin, $id)
```

Où `db` est le lien *PDO* de la base de données, `userLogin` le nom de l'utilisateur à qui appartient le commentaire et `id` l'id du commentaire à supprimer. Cette fonction doit renvoyer `true` en cas de succès, `false` sinon.

Étude de la requête du client en *PHP*

Étant donné que l'authentification n'est pas encore présente, il est nécessaire de définir un nom d'utilisateur par défaut dans le script « request.php » grâce à :

```
$login = 'cir2';
```

Afin d'offrir les foncitonnalités nécessaire à la gestion des commentaires, les requêtes (avec un formalisme *REST*) suivantes doivent pouvoir être traiter par le script *PHP* « request.php » :

GET	`php/request.php/comments/?id=??`	Récupération des commentaires
POST	`php/request.php/comments/ id=??&comment=??`	Ajout d'un commentaire
PUT	`php/request.php/comments/i comment=??`	Modification du commentaire
DELETE	`php/request.php/comments/i`	Suppression d'un commentaire

Pour cela, appuyez-vous sur les fonctions que vous venez de définir dans « database.php » pour répondre correctement aux différents demandes du client en vous basant sur la ressources demandée ainsi que le verbe *REST* utilisé. Finalement, pour plus de rigueur, envoyez le statut suivant lors d'une requête de type *POST* :

```
201 Created
```

> **Nettoyage des balises dans une chaîne** : afin d'éviter l'injection de balise *HTML* dans la base de données, on peut nettoyer une chaîne grâce à la fonction *PHP* suivante :
>
> ```php
> message = strip_tags(message);
> ```

3 Requête *AJAX* en *JavaScript*

Requête cliente pour les commentaires

Afin de visualiser les commentaires associés à une photo, il est nécessaire de réaliser la requête *AJAX* suivante :

GET	`php/request.php/comments/?id=i`	Récupération des commentaires

Pour commencer, ajoutez un script nommé « comments.js » dans le dossier « js » et ajoutez-le à votre page *HTML*. Vous penserez à définir le mode strict dans ce nouveau script *JavaScript*. Afin de pouvoir récupérer la liste des commentaires, ajoutez dans « comments.js » la fonction `requestComments` qui doit réaliser la requête *AJAX* mentionnée ci-dessus. Pour récupérer l'id de la photo actuellement affichée, utilisez l'attribut `photoid` de l'élément *HTML* d'id `photo-large`. En cas de succès de la requête, appelez la nouvelle fonction suivante :

```php
displayComments(comments)
```

Enfin, appelez la fonction `requestComments` à la fin de la fonction `displayPhoto` disponible dans le script « photos.js ».

Affichage des commentaires

Pour visualiser les commentaires, ajoutez dans la fonction `displayComments` les instructions permettant d'ajouter un `card` *Bootstrap* (cliquez ici pour la documentation) dans la `div` d'id `comments` pour chaque commentaire. Vous pouvez ajouter des boutons de modification et de suppression en utilisant les balises *HTML* suivantes :

```html
<button type="button" class="btn btn-light float-end mod" value="??">
  <i class="fa fa-edit"></i>
</button>
<button type="button" class="btn btn-light float-end del" value="??">
  <i class="fa fa-trash"></i>
</button>
```

Où `??` correspond à l'id du commentaire. Vous penserez à rendre visible la `div` d'ajout de commentaires.

Ajout d'un commentaire

Pour envoyer au serveur le nouveau commentaire saisi par l'utilisateur, écrivez dans le fichier « comment.js » une fonction nommée `sendComment` ayant `event` pour seul paramètre. Celle-ci

doit récupérer l'id de la photo actuellement affichée ainsi que le texte du commentaire à envoyer et effectuer la requête *AJAX* sur la ressource suivante :

POST	`php/request.php/comments/ id=??&comment=??`	Ajout d'un commentaire

Où les premiers `??` correspondent à l'id de la photo tandis que les seconds correspondent au texte du commentaire.

Si la réponse de la requête *POST* est valide, appelez la fonction `requestComments` pour rafraîchir la liste des commentaires. Enfin, ajoutez au début du script « comments.js » une gestion de l'évènement `submit` sur le formulaire d'envoi de commentaire pour déclencher la fonction `sendComment`.

> **Encodage des données en *POST*** : Pour permettre l'envoi de données dans le corps (c.-à-d. avec la clé `body` des paramètres de la fonction `fetch`) d'une requête *AJAX* de type *POST*, il est nécessaire de définir dans l'en-tête de la requête le type d'encodage des données en ajoutant la clé suivante dans les paramètres de la fonction `fetch` :
>
> ```
> headers: {'Content-Type': 'application/x-www-form-urlencoded'}
> ```

Modification d'un commentaire

Pour offrir la possibilité de modifier un commentaire, écrivez dans le script « comments.js » la fonction nommée `modifyComments`. Dans celle-ci, ajouter une gestion de l'évènement `click` sur les boutons de modifications associés aux commentaires avec les instructions suivantes :

```
const modifyButtons = document.querySelectorAll('.mod');
  modifyButtons.forEach(e => e.addEventListener('click', ...));
```

La fonction à appeler lors du clic (représentée dans le code précédent par `...`) doit réaliser la requête *AJAX* sut la ressource suivante :

PUT	`php/request.php/comments/i comment=??`	Modification du commentaire

Où `i` est l'id du commentaire à modifier et `??` le nouveau texte. L'id peut être obtenu comme suit :

```
let id = event.target.closest('.mod').getAttribute('value');
```

Notez que le nouveau texte pourra être demandé à l'utilisateur à l'aide d'un `prompt`. En cas de succès appelez la fonction `requestComments` pour rafraîchir la liste des commentaires. Enfin, appelez cette fonction à la fin de la fonction `displayComments`.

> **Méthode `closest` en *JavaScript*** : La méthode `closest` de l'interface `Element` parcourt l'élément et ses parents (en direction de la racine du document) jusqu'à ce qu'elle trouve un nœud correspondant au sélecteur *CSS* spécifié. Par exemple :
>
> ```
> let value = element.closest('.cssClass').getAttribute('value');
> ```

Suppression d'un commentaire

Finalement, pour offrir la possibilité de supprimer un commentaire, écrivez dans le script « comments.js » la fonction nommée `deleteComments`. Dans celle-ci, ajouter une gestion de l'évènement `click` sur les boutons de suppressions associés aux commentaires avec les instructions suivantes :

```
const deleteButtons = document.querySelectorAll('.del');
deleteButtons.forEach(e => e.addEventListener('click', ...));
```

La fonction à appeler lors du clic (représentée dans le code précédent par `...`) doit réaliser la requête *AJAX* sut la ressource suivante :

DELETE	**php/request.php/comments/i**	Suppression d'un commentaire

Où `i` est l'id du commentaire à supprimer. L'id peut être obtenu comme suit :

```
let id = event.target.closest('.del').getAttribute('value');
```

En cas de succès appelez la fonction `requestComments` pour rafraîchir la liste des commentaires. Enfin, appelez cette fonction à la fin de la fonction `displayComments`.