Rapport Projet Web

Lafon Lucas & Théo Chambon – L3 Informatique 2013/2014

I. Architecture

1.1 Le patron MVC

Ce patron a été utilisé pour ce projet afin de pouvoir se répartir les tâches facilement.

En effet, ce patron permet tout d'abord de dissocier les différentes fonctions de l'application, permettant d'avoir trois parties distinctes: l'affichage (view), les données (model) et les événements possibles (controller). Ainsi, il est simple de comprendre où sont situés les fichiers que l'on recherche.

1.2 La maquette

Le site possède une vue nommée “gabarit.php” qui sera chargée sur chaque page. Ce gabarit contient deux parties qui ne changeront pas entre chaque page: le menu et les blocs affichant les groupes et les sondages. Seul le corps de page est dynamique.

1.2.1 Le menu

Dans le gabarit on retrouve le menu qui permet de naviguer sur le site. Ce menu contient:

- un lien vers la page des sondages
- un lien vers la page des groupes
- le nom de l’utilisateur connecté
- un lien vers la page d’administration si l’utilisateur connecté est un administrateur
- un lien de déconnexion

1.2.2 Affichage des groupes et sondages

En utilisant le patron MVC, nous avons décidé que le gabarit ne contiendrait que du contenu statique (on ne peut pas passer une variable du contrôleur vers le gabarit). C’est pour cela que la liste des groupes est chargée dynamiquement en AJAX. On va donc appeler la méthode ajaxGetListeGroupes() du contrôleur Groupe qui va retourner la liste des groupes au format JSON.

Le bloc de sondage est statique et contient un lien pour ajouter un sondage au site.

1.2.3 Affichage du corps de page

Le corps de la page chargera une vue en fonction des demandes des contrôleurs.

```
<div class="blogbody">
        <div id="contenu">
                <?= $contenu //<==== Affichage de le vue?>
        </div> <!-- #contenu -->
</div>
```

II. Partie utilisateur

2.1 Inscription

2.1.1 Affichage du formulaire d'inscription

Pour afficher le formulaire d'inscription l'utilisateur va appeler la méthode afficherInscription() du controleur ControllerUser. Cette méthode affiche la vue vueInscription.php.

Cette vue affiche un formulaire contenant trois champs :

- le pseudo de l'utilisateur
- le nom de l'utilisateur
- le prenom de l'utilisateur
- l'adresse email de l'utilisateur
- le mot de passe de l'utilisateur
- un champ de vérification du mot de passe

2.1.2 Envoi des données d'inscriptions au serveur

Le formulaire d'inscription envoi les données
s de ces champs en méthode POST à la méthode inscription() du contrôleur controllerUser.

Dans un premier temps on ajoute les valeurs de ces champs au modèle User. Ce modèle va permettre par la suite d'ajouter cet utilisateur en base. Le mot de passe est déjà encrypté en Javascript avant d'envoyer le formulaire, il ne sera donc pas nécessaire de s'en occuper côté serveur.

Pour ajouter les variables du formulaire au modèle User, la méthode POSTToVar($_POST) est utilisée. Cette méthode va récupérer toutes les variables reçues en $_POST et les stocker dans le modèle. Cette méthode va permettre aussi de supprimer les caractères dangereux qui pourraient altérer la base de données.

Deuxièmement il est nécessaire de vérifier la validité des champs afin que l'utilisateur rentre des informations correctes lors de son inscription. Pour cela le modèle User contient des méthodes de vérifications : par exemple la méthode validateEmail() vérifie si l'attribut $email du modèle est correcte grâce à une expression régulière :

```php
/* Vérification de la validité email */
public function validateEmail(){
        $regexp = ``^[[:alnum:]]([-_.]?[[:alnum:]])*@[[:alnum:]]([-.]?[[:alnum:]])*\.([a-z]{2,4})$``;
        if( empty($this->email) || (!empty($this->email) && !preg_match($regexp, $this->email)) ){
                return lg_erreur_email;
        } else{
                return 1;
        }
}
```

Troisièmement lorsque les champs du formulaire sont valide alors une transaction avec la base de données débute. Dans cette transaction on créé un nouvel utilisateur en base grâce à la méthode add() du modèle User. Cette méthode va insérer un utilisateur en base de données avec les informations fournies au modèle User. Cependant on va ajouter à cet utilisateur créé en base un hashage unique qui lui permettra de valider son compte par email par la suite.

Finalement, si l'utilisateur a bien été créé en base de données on envoi un email à l'utilisateur concerné en lui fournissant un lien lui permettant d'activer son compte. Cet email n'est pas envoyé si l'administrateur préfère valider lui même les comptes.

2.1.3 Activation du compte

Lorsque l'utilisateur est créé, il possède un attribut en base de donnée qui permet de définir si le compte est activé (par défaut un utilisateur a son compte désactivé).

Pour activer son compte il devra cliquer sur un lien reçu par email après son inscription. Ce lien va appeler la méthode valider_compte() du contrôleur ControllerUser en passant une variable en méthode GET qui est le hashage unique à l'utilisateur stoqué en base de donnée.

La méthode valider_compte va simplement demander au modèle User d'aller chercher en base un utilisateur qui possède le hashage transmis et d'activer son compte si il existe.
Un message d'erreur sera affiché dans le cas contraire.

2.2 Connexion

2.2.1 Affichage du formulaire de connexion

Pour afficher le formulaire connexion l'utilisateur va appeler la méthode afficherConnexion() du controleur ControllerUser. Cette méthode affiche la vue vueConnexion.php.

Cette vue affiche un formulaire contenant deux champs :

- l'adresse email de l'utilisateur
- le mot de passe de l'utilisateur

2.2.2 Envoi des données de connexion au serveur

Le formulaire de connexion envoi les données de ces deux champs en méthode POST à la méthode connexion() du contrôleur controllerUser.

Dans un premier temps on ajoute les valeurs de ces deux champs au modèle User. Ce modèle va permettre par la suite de vérifier les identifiants de cet utilisateur en base. Le mot de passe est déjà encrypté en Javascript avant d'envoyer le formulaire, il ne sera donc pas nécessaire de s'en occuper côté serveur.

Deuxièmement il est nécessaire de vérifier la validité des champs afin que l'utilisateur rentre des informations correctes lors de sa connexion. Pour cela les mêmes méthodes validateEmail() et validateMotDePasse() vont permettre la vérification.

Troisièmement il y a la vérification que le couple nom d'utilisateur et mot de passe existe en base de données. On va donc appeler la méthode combinaison_connexion_valide() du modèle User qui va vérifier en base si le couple existe pour un utilisateur dont le compte est activé et si c'est le cas cette fonction retourne le numéro d'utilisateur de la personne concernée.

Enfin il ne reste plus qu'à créer une session contenant l'identifiant de l'utilisateur connecté. Cette variable peu être utilisée par la suite sur chaque page pour vérifier si un utilisateur est connecté. Dans le cas où le couple donné n'existe pas en base de données on affiche le formulaire de connexion à nouveau avec un message d'erreur.

III. Partie administrateur

3.1 Formulaire de gestion des inscriptions

Une table en base de données stocke la configuration définie par un administrateur du site. Cette table possède une ligne contenant le mode d'inscription à utiliser pour le site. Il y a donc un formulaire dans la page d'administration (affichée grâce à la vue vueAdministration) permettant de modifier ce mode d'inscription:

```
<h2>Gestion des inscriptions</h2>
<form action="<?= ABSOLUTE_ROOT . '/index.php?controller=Admin&action=modifierInscription' ?>" method="POST">
    <p>
        <label for="ouvertes">Ouvertes</label>
        <input type="radio" name="inscription" id="ouvertes" value="ouvertes" <?php if($inscriptions=="ouvertes") echo 'checked="checked"'; ?> >
    </p>
    <p>
        <label for="validation">Avec validation</label>
        <input type="radio" name="inscription" id="validation" value="validation" <?php if($inscriptions=="validation") echo 'checked="checked"'; ?> >
    </p>
    <p>
    <input type="submit" name="modifierInscription" value="Modifier" />
    </p>
</form>
```

Lorsque l'administrateur envoie ce formulaire alors on va mettre à jour ce champ en base par le biais du modèle ConfigAdmin.

```php
/* Modifie la config en base
@return: vrai si la config a été modifiée, faux sinon
*/
public function update()
{
    $sql = "UPDATE config SET
                valeur = ?
                WHERE type = 'inscriptions'";
    $update = $this->executerRequete($sql, array($this->inscriptions));
    return ($update->rowCount() == 1);
}
```

3.2 Gestion des utilisateurs

3.2.1 Affichage des utilisateurs

Pour afficher la liste des utilisateurs on va utiliser le modèle ListeUser. Ce modèle possède un constructeur qui va récupérer tous les utilisateurs de la base de données et les stocker dans un tableau:

```php
public function constructeurPublic(){
            $sql='SELECT id
                    FROM user
                    ORDER BY id DESC';

        $lectBdd = $this->executerRequete($sql, array());
        while (($enrBdd = $lectBdd->fetch()) != false)
        {

            $this->array_user[] = new User($enrBdd["id"]);
        }
}
```

Il suffit de parcourir ce tableau pour afficher la liste des utilisateurs.

3.2.2 Activation et bannissement d'un compte utilisateur

Pour chaque utilisateur on affiche un lien permettant d'activer un compte utilisateur ou bien bannir un compte utilisateur.

Le controleur controllerUser possède pour cela une méthode activerMembre($id) pour activer un compte et bannirMembre($id) pour bannir un membre. Seuls les administrateurs peuvent appeler ces méthodes.

Ces méthodes vont modifier le champ permettant de savoir si le compte est actif en utilisant le modèle User et en utilisant la méthode update() de ce modèle permettant de mettre à jour l'utilisateur en base de données. Si ce champ est faux alors l'utilisateur ne pourra plus se connecter.

IV. Groupes

4.1 Création d'un groupe

Lorsque l'on clique sur le lien de création de groupe une fenêtre de dialogue s'ouvre. Cette fenêtre est un objet "Dialog" de la librairie JqueryUI. Dans cette fenêtre il y a un formulaire contenant le nom du groupe a créer et sa visibilité. Lorsque le formulaire est envoyé on vérifie d'abord en AJAX si il n'y a pas de groupe portant le même nom et sinon on créé le groupe en base de données et on redirige l'utilisateur vers la groupe.

4.2 Création d'un sous groupe

Dans la page des sondages d'un groupe un champ permet d'ajouter un sous groupe. Ce champ n'apparaît que si l'utilisateur connecté est l'administrateur du groupe ou du site. Lorsque ce formulaire est envoyé on ajoute en base de données le sous groupe à la table des sous groupes.

4.3 Affichage

4.3.1 Affichage de la liste des groupes

C'est le modèle ListeGroupes qui permet de construire une liste de groupes. Ce modèle possède un seul attribut contenant la liste des groupes récupérée par le constructeur. Cette récupération se fait en fonction de l'utilisateur connecté, on récupère pour celà:

- Les groupes que l'utilisateur connecté administre
- Les groupes pour lesquels l'utilisateur connecté est un membre
- Les groupes pour lesquels l'utilisateur connecté est un modérateur

Il ne reste plus qu'à parcourir ce tableau pour afficher la liste des groupes d'un utilisateur. Dans la vue affichant les groupes la dernière colonne est générée en fonction du status de l'utilisateur connecté. Si l'utilisateur est un administrateur du groupe il peut le supprimer. Si l'utilisateur est un membre ou un modérateur, il peut seulement le quitter.

4.3.2 Affichage des membres

Lorsque l'on construit un groupe avec le modèle Groupe on récupère la liste des membres du groupe en base de données. Ces membres sont stockées dans l'attribut $array_membres du modèle. Il ne reste plus qu'à parcourir ce tableau pour afficher la liste des membres du groupe. L'administrateur du groupe peut voir un bouton "Ajouter un membre" qui permet d'ajouter des membres à un groupe secret. Lorsque l'on clique sur ce bouton, une fenêtre s'affiche avec un champ pour ajouter un membre. En écrivant les deux premières lettres d'un membre on récupère en AJAX les membres correspondants, ainsi l'administrateur n'a plus qu'à cliquer sur le membre à ajouter.

```
public function getMembresLikeAndNotInGroup($term, $idGroupe){

        $sql = 'SELECT id, nom, prenom
                FROM user
                WHERE (nom LIKE \'%'. $term . '%\'
                OR prenom LIKE \'%'. $term . '%\')
                AND user.id NOT IN (SELECT id_user FROM user_groupe_membre WHERE id_groupe=?)
                AND user.id NOT IN (SELECT id_user FROM user_groupe_moderateur WHERE id_groupe=?)
                AND user.id NOT IN (SELECT administrateur_id FROM groupe WHERE id=?)';

        $selectGroupe = $this->executerRequete($sql, array($idGroupe, $idGroupe, $idGroupe ));

        $result = array();
        $i = 0;
        while (($enrBdd = $selectGroupe->fetch()) != false)
        {
            $result[$i]["id"] = $enrBdd["id"];
            $result[$i]["label"] = $enrBdd["nom"] . ' ' . $enrBdd["prenom"];
            $result[$i]["value"] = $enrBdd["nom"] . ' ' . $enrBdd["prenom"];
            $i++;
        }
        return $result;
}
```

Ce bout de code recherche en base de données les membres dont les noms commencent par les lettres entrées dans le champ d'ajout de membres ou de modérateurs et qui n'appartiennent pas déjà au groupe.

De plus le modèle Groupe possède un attribut $array_membres_en_attente qui stocke les membres en attente d'approbation. En affichant ce tableau on ajoute un lien pour accepter le membre ou le refuser. Lorsqu'on clique sur un de ces liens, on appelle deux fonctions AJAX pour modifier le choix en base de données.

4.3.3 Affichage des modérateur

La liste des modérateurs est stockée dans l'attribut $array_moderateurs du modèle Groupe. Un parcours de cet attribut permet d'afficher la liste des modérateurs. De la même façon que l'ajout d'un membre, l'administrateur du groupe peut ajouter ou supprimer des modérateur.

```
<?php foreach($groupe->getArrayModerateurs() as $moderateur): ?>
<li class="user<?= $moderateur->getId() ?>">
        <?= $moderateur->getPrenom() . ' ' . $moderateur->getNom() ?>

        <?php if($user->getId() == $groupe->getAdministrateurId() || $user->getAdministrateurSite() == 1): //Si l'utilisateur connecté est l'administrateur du grou
        <span class="lienSupprimerMembre"><a href="#" id="supprimerModerateur<?= $moderateur->getId() ?>" onclick="supprimerModerateurGroupe(<?= $moderateur->getId
        <?php endif; ?>
</li>
<li class="aucunModerateur" style="display: none;">Il n'y a aucun modérateur dans ce groupe</li>
<?php endforeach; ?>
```

V. Sondages

5.1 Création d'un sondage

La création d'un nouveau sondage, bien que assez similaire dans les différents cas, peut être séparée en trois parties, la création d'un sondage général, qu'il soit public ou privé, c'est à dire qui n'appartient à aucun groupe, tout les utilisateurs (s'il est public) ou ceux qui sont invités peuvent y répondre, et la création d'un sondage dans un groupe ou dans un sous groupe, qui sera accessible uniquement aux utilisateurs membres de ce groupe.

5.1.1 Générale

Pour la création d'un sondage, l'utilisateur doit remplir un formulaire renseignant toutes les informations de ce sondage (titres, options, confidentialité, date de fin…).
Ce formulaire rempli appelle la fonction nouveauSondage() du controleur, qui transmet les informations de ce formulaire au modèle Sondage et stocke les informations dans les variables du modèle, apelle la méthode add() de ce modèle, puis vérifie ensuite si la création s'est déroulée sans erreur.
Cette méthode add(), à l'aide d'un if, vérifie dans quel cas nous nous trouvons, c'est à dire si c'est un ajout d'un sondage dans un groupe ou non.
Dans ce cas, si l'on souhaite ajouter un sondage hors d'un groupe, voici le code utilisé :

```
                $sql = 'INSERT INTO sondage SET
                titre=?,
                description=?,
                visibilite=?,
                administrateur_id=?,
                date_creation= NOW(),
                date_fin=?,
                `secret`=?';
                $insertSondage = $this->insererValeur($sql, array($this->titre, $this-
>description , $this->visibilite, $this->administrateur_id, $this->date_fin, $this-
>secret));
                return $insertSondage;
```

5.1.2 Dans un groupe ou sous-groupe

Si l'utilisateur tente d'ajouter un sondage dans un groupe ou dans un sous-groupe et qu'il fait bien partie de ce groupe, la même méthode du controleur sera appelé, et la même méthode du modèle sondage aussi, la différence se situe au niveau du cas du if.
Voici le code utilisé pour l'ajout dans un groupe ou un sous-groupe :

```
public function add() {


                if($this->id_groupe!=-1){
                $sql = 'INSERT INTO sondage SET
                titre=?,
                description=?,
                visibilite=?,
                administrateur_id=?,
                date_creation= NOW(),
                date_fin=?,
                `secret`=?,
                id_groupe=?';
                $insertSondage = $this->insererValeur($sql, array($this->titre, $this-
>description , $this->visibilite, $this->administrateur_id, $this->date_fin, $this->secret,
$this->id_groupe));
                return $insertSondage;
```

La vérification id_groupe = -1 permet de vérifier si un groupe ou sous-groupe à été spécifié, pour savoir si l'utilisateur souhaite ou non ajouter ce sondage à un groupe.

Le return est ensuite inspecté par le contrôleur pour vérifier que l'ajout s'est déroulé correctement.

5.1.3 Ajout des Options

Une fois le sondage correctement ajouté, le contrôleur appelle la fonction NouvelleOption et lui passe en paramètre un tableau contenant les options que l'utilisateur à entré dans le formulaire de création du sondage.
La fonction NouvelleOption parcourt se tableau, et pour chaque option présente dans ce tableau, elle crée un objet Option à l'aide du modèle correspondant, et l'ajoute en base, à l'aide de la fonction add() de ce modèle :

```
public function update(){
                $sql= 'UPDATE option SET
                texte=?,
```

```
                id_sondage=?
                WHERE id=?';

                $updateOption = $this->executerRequete($sql, array($this->texte, $this->id_sondage, $this->id));
                return $updateOption;
        }
```

Le contrôleur vérifie, pour chaque option, que l'insertion en base s'est correctement déroulée.

5.2 Affichage des Sondages

L'affichage des sondages peut être séparé en deux parties, l'affichage de la liste des sondages pour un groupe ou pour certains utilisateurs, et l'affichage détaillé d'un sondage, avec ses options et ses résultats s'il est terminé.

5.2.1 Affichage des listes

Le contrôleur possède plusieurs méthodes permettant d'afficher différentes listes de sondages selon le contexte, une méthode pour afficher les sondages publics lorsqu'un utlisateurs cliquera sur l'onglet public, une pour les sondages privés, une pour les sondages d'un groupe, et cela pour chaque cas de figure. Leur fonctionnement est assez similaire, dans chacun des cas, la méthode créera un objet de type ListeSondage correspondant à ce modèle, qui est un tableau contenant les ID des sondages correspondant à la condition donnée (les sondages public, les sondages d'un groupe…). Par exemple, la liste des sondages publics est obtenu comme cela :

```
public function constructeurPublic(){
            $sql='SELECT id, date_creation
                   FROM sondage
                      WHERE visibilite="public"
                  ORDER BY id DESC';

            $lectBdd = $this->executerRequete($sql, array());
            while (($enrBdd = $lectBdd->fetch()) != false)
            {

                $this->array_sondage[] = new Sondage($enrBdd["id"]);
            }
        }
```

Le controleur récupère ensuite le tableau array_sondage, puis le transmet a la vue correspondante qui se chargera d'afficher les sondages.
Pour afficher la liste des sondages, on détermine d'abord si le sondage est terminé en comparant la date de fin à la date actuelle, puis on affiche le titre du sondage, sa description et sa date de fin, sous forme de lien vers la vue détaillée pour voter s'il n'est pas terminé, la page des résultats sinon.

```
if($now<$datefin){
                                ?>
                                <td><a href="<?= ABSOLUTE_ROOT .
'/controllers/ControllerSondage.php?action=afficherFicheSondage&params=' . $sondage-
>getId() ?>"><?php echo($sondage->getTitre()); ?></a></td>
                                <td><a href="<?= ABSOLUTE_ROOT .
'/controllers/ControllerSondage.php?action=afficherFicheSondage&params=' . $sondage-
>getId() ?>"><?php echo($sondage->getDesc()); ?></a></td>
                                <td><a href="<?= ABSOLUTE_ROOT .
'/controllers/ControllerSondage.php?action=afficherFicheSondage&params=' . $sondage-
>getId() ?>">
                                <?php
                                echo $datefin . "</a></td>";
                        }
                        else{
                                ?>
                                <td><a href="<?= ABSOLUTE_ROOT .
'/controllers/ControllerSondage.php?action=resultat&params=' . $sondage->getId() ?>"><?php
echo($sondage->getTitre()); ?></a></td>
                                <td><a href="<?= ABSOLUTE_ROOT .
'/controllers/ControllerSondage.php?action=resultat&params=' . $sondage->getId() ?>"><?php
echo($sondage->getDesc()); ?></a></td>
                                <td><a href="<?= ABSOLUTE_ROOT .
'/controllers/ControllerSondage.php?action=resultat&params=' . $sondage->getId() ?>">
                                <?php
                                echo "Terminé". "</a></td>";
                        }
```

5.2.2 Affichage détaillé et Affichage des résultats

Lorsque l'utilisateur souhaite voter, il clique sur le lien le menant vers l'affichage détaillé du sondage et de ces options. Le contrôleur transmet à cette vue les informations sur le sondage ainsi qu'un tableau contenant la liste des options correspondantes à ce sondage.

La vue vérifie si l'utilisateur à déjà voté ou non, puis si ce n'est pas le cas, affiche un formulaire lui permettant d'ordonner la liste des options du sondage puis de voter.
Si le secret du sondage est public, on récupère les utilisateurs ayant déjà voté pour chacune des options, puis l'utilisateur peut ainsi afficher qui à voté pour quelle option, à l'aide d'un pop-up réalisé en JavaScript, grâce à JQuery.

Si le sondage est terminé, l'utilisateur est amené vers la page d'affichage des résultats.
Les résultats sont affichés à l'aide du modèle Score, qui permet de créer un objet correspondant. Le constructeur crée un objet Score, contenant les informations d'une option ainsi que son nombre de points, pour chacune des options du sondage, et les stocke dans un tableau qui sera passé à la vue. Ce tableau est indexé par les ID des options, afin de pouvoir accéder simplement au score d'une option.
Le score d'une option utilise la méthode de Borda, c'est à dire que chaque option séléctionnée par un votant obtient un score de n pour la première, n-1 pour la seconde, n-2 pour la troisième, ainsi de suite, n étant le nombre d'options.
Le score est ensuite récupéré depuis la base de données à l'aide de la fonction suivante :

```
public function calculeScore(){


        $sql='SELECT SUM( classement )
                        FROM (
                        SELECT classement
                FROM user_sondage_reponse
                WHERE id_option=? AND id_sondage=?
                                UNION
                SELECT classement
                        FROM invite_sondage_reponse
                WHERE id_option=? AND id_sondage=?
                        ) as unionuser';


        $lectBdd = $this->executerRequete($sql, array($this->id_option, $this->id_sondage,
$this->id_option, $this->id_sondage));
                $enrBdd = $lectBdd->fetch();
                $this->score = $enrBdd[0];


}
```

Pour avoir le résultat de chaque option on divise son score par le score total de toutes options. Cela est ensuite ramené sous forme de pourcentage pour un affichage plus clair.
L'affichage graphique des résultats se fait à l'aide de JQuery, en utlisant des barres de progression représentant le résultat de chacune des options.

VI. Commentaires

6.1 Ajout d'un commentaire

Un textarea (affichés pour les utilisateurs connectés) permet d'ajouter un commentaire à un sondage. Lorsque l'on envoi un commentaire on appelle en AJAX la méthode ajaxAjouterCommentaire() du controlleur ControllerSondage.

```php
/* Ajoute un commentaire à un sondage avec les infos récupérées en $_POST: texteCommentaire et sondageId */
public function ajaxAjouterCommentaire(){
	if(empty($_POST["texteCommentaire"])){
		echo "Le commentaire ne peut pas être vide";
		die();
	}

	if(empty($_POST["sondageId"])){
		echo "Le sondage n'existe pas";
		die();
	}

	if(empty($_SESSION["id"])){
		$controllerUser = new ControllerUser();
		$controllerUser->addErreur("Vous devez vous connecter pour ajouter un commentaire à un sondage");
		$controllerUser->afficherConnexion();
	}else{
		$this->sondage = new Sondage($_POST["sondageId"]);
		$userConnecte = new User($_SESSION["id"]);
		$ajoutCom = $this->sondage->ajouterCommentaire($_POST["texteCommentaire"], $userConnecte->getId())
		if(ctype_digit($ajoutCom))
			echo $ajoutCom;
		else{
			echo "Impossible d'ajouter le commentaire";
		}
	}
```

6.2 Suppression d'un commentaire

La suppression de commentaires peuvent être fait par:

- L'administrateur du sondage
- L'administrateur du site
- Un modérateur du sondage

Un bouton de suppression se trouve à droite de chaque sondage. Lorsqu'on clique sur ce bouton on appelle en AJAX la méthode permettant de supprimer le commentaire. Cela se fait de la même manière pour les sous commentaires.

```php
/* Supprime un modérateur au sondage */
public function ajaxSupprimerCommentaire($idCom = null){
        if(empty($_POST['commentaireId']))
                echo 'Aucun commentaire à supprimer';


        if(empty($_SESSION["id"])){
                $controllerUser = new ControllerUser();
                $controllerUser->addErreur("Vous devez vous connecter pour supprimer des commentaires");
                $controllerUser->afficherConnexion();
        }else{
                $userConnecte = new User($_SESSION["id"]);
                $commentaire = new Commentaire($_POST['commentaireId']);
                $this->sondage = new Sondage($commentaire->getIdSondage());

                if($userConnecte->getId() == $this->sondage->getAdministrateur() || $userConnecte->getAdministrateurSite() == 1){
                        echo $commentaire->remove();
                } else{
                        $this->addErreur("Vous devez être l'administrateur du sondage pour pouvoir supprimer les moderateurs");
                        $this->afficherFicheSondage($idSondage);
                }
        }
}
```

6.3. Soutiens d'un commentaire

Un bouton permet d'ajouter un soutien à un commentaire. Lorsque l'on clique dessus alors on ajoute l'utilisateur connecté à la table des soutiens. Ainsi un utilisateur ne peut pas ajouter plusieurs soutiens au même commentaire.

VII. Perspectives et Conclusion

Le site obtenu est entièrement fonctionnel, et permet de mettre en place les sondages avec la dimension communautaire demandée.
Tous les objectifs requis de la partie obligatoire ont été remplis et sont pleinement fonctionnels.
La majeure partie des fonctions de la partie améliorations ont également été implémentées, le site est ainsi très complet.

Pour l'évolution du site, on pourra imaginer rajouter les objectifs supplémentaires manquants : la délégation des votes et le choix parmi différentes méthodes de résolution pour les sondages. Toutes les autres fonctionnalités ont été implémentées.

Le code source du projet est disponible à l'adresse suivante :

https://github.com/Theo0/sondage-L3/