Travail pratique #2

420-715-FE Programmation d'une application Web transactionnelle

Hiver 2022

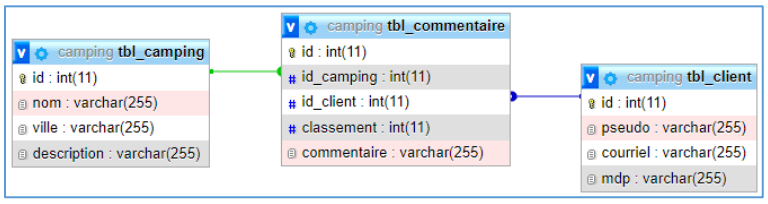
Mise en situation

Le site Web à compléter permettra de consulter les avis sur des campings. Un utilisateur non authentifié pourra simplement consulter les avis laissés par d'autres utilisateurs, alors qu'un visiteur authentifié pourra, en plus, ajouter des commentaires au sujet des campings.

Vous devrez ajouter les fonctionnalités demandées dans cet énoncé en respectant l'architecture fournie et en vous basant sur les captures d'écran. Tous les fichiers nécessaires sont déjà créés (vous n'aurez pas à en créer d'autres), mais vous devrez cependant ajouter votre propre code dans certains fichiers vides ou incomplets.

Base de données

Importez dans *PHPMyAdmin* la base de données fournie au moyen du fichier SQL « camping.sql ». La structure de cette base de données est la suivante :

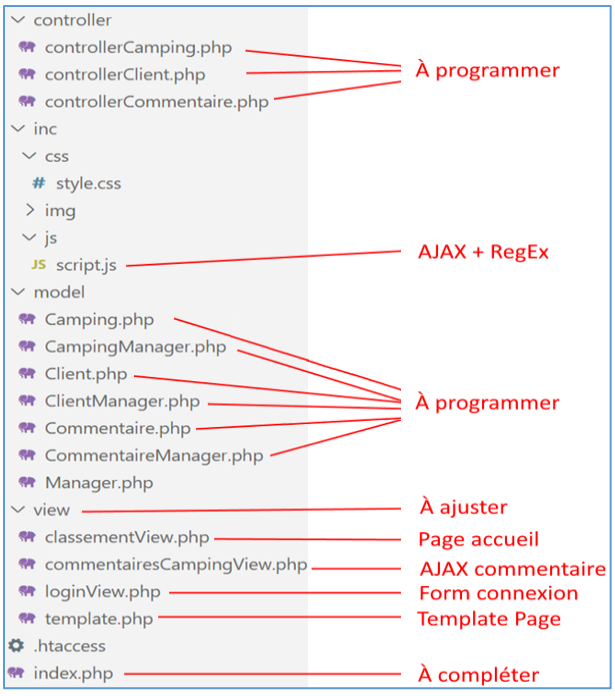


Les deux utilisateurs déjà inscrits dans la base de données (vous n'aurez pas à en créer de nouveau dans le cadre de cet examen) sont *marc07* et *lisette77*. Leur mot de passe respectif est identique à leur nom d'utilisateur, par exemple :

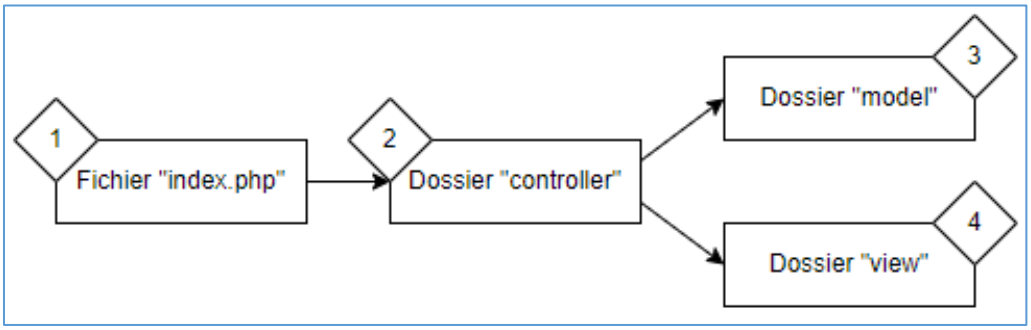
Nom d'utilisateur : marc07
Mot de passe : marc07

Architecture de dossiers/fichiers

Après avoir décompressé le fichier « .zip » du travail pratique, vous devriez retrouver l'architecture suivante.



Cette architecture est de type MVC (Modèles – Vues – Contrôleurs), ce qui signifie que toutes les transactions d'informations (soumission d'un formulaire, requête AJAX, changement de page, etc.) doivent respecter le flot illustré dans la figure ci-dessous.



Autrement dit, toute requête effectuée sur le site Web doit d'abord être acheminée au fichier « index.php » (1), lequel s'occupera ensuite d'appeler une fonction d'un des fichiers contenus dans le dossier « controller » (2). Cette fonction du contrôleur se chargera ensuite d'inclure un des gestionnaires du dossier « model » (3) et d'appeler enfin la bonne vue dans le dossier « view » (4).

Affichage de la page d'accueil (fichier « index.php »)

La page « view/classementView.php » contient une liste de tous les campings enregistrés dans la base de données. Cette page, une fois pleinement fonctionnelle, devrait afficher le nom de chaque camping, sa ville, son classement moyen, un bouton « Afficher les commentaires » et un formulaire pour ajouter un commentaire. **Les actions sur les boutons seront réalisées plus loin dans le travail pratique.**

- Complétez l'instruction « else » au bas du fichier « index.php » pour parvenir à charger la page « view/classementView.php » en utilisant contrôleur « controller/controllerCamping.php ».
    - Dans le contrôleur « controller/controllerCamping.php » vous devrez définir une fonction « getCampings() » qui utilisera le gestionnaire « model/CampingManager.php » pour récupérer toutes les campings sous la forme d'un tableau d'instances de la classe « Camping ».
    - La fonction « getCampings() » chargera ensuite le fichier « view/classementView.php ».

- Le classement moyen de chaque camping doit correspondre à la moyenne des notes attribuées par camping dans la colonne « classements » de la table « tbl_commentaire » dans la base de données. Si le camping n'a pas encore reçu de notes, la mention « N/A » remplace alors la moyenne. **Voici la requête que vous devrez placer dans la classe de type « manager » du fichier « model/CampingManager.php »** pour implémenter ce calcul de moyenne :

```
SELECT *, IFNULL((SELECT ROUND(AVG(classement), 2) FROM tbl_commentaire AS com
WHERE cam.id = com.id_camping GROUP BY id_camping), 'N/A') AS classementMoyen
FROM tbl_camping AS cam
```

- Une fois vos modifications complétées, vous devriez obtenir le résultat suivant à l'affichage de la page « index.php ».

title>Classements

```
           - - - - SESSION PHP - - - -

           Array
(
)
           - - - - - - - - - - - - - -
```

- Les classements
- Se connecter

Les classements

Camping : Camping de l'Île Marie

Ville : Sherbrooke

Classement : N/A

Afficher commentaire

Camping : Camping Beau-lieu

Ville : Sherbrooke

Classement : 4.00

Afficher commentaire

Camping : Camping plage McKenzie

Ville : Racine

Classement : 3.67

Afficher commentaire

Camping : Camping de Compton

Ville : Compton

Classement : N/A

Afficher commentaire

Authentification (connexion et déconnexion d'un utilisateur)

Ajoutez la logique pour procéder à l'authentification d'un visiteur à l'aide de son nom d'utilisateur et de son mot de passe (il y a déjà deux utilisateurs inscrits dans la base de données à cette fin).

- Un utilisateur non authentifié pourra voir les informations du camping et les commentaires laissés par d'autres utilisateurs, mais **il ne devra pas avoir accès au formulaire d'ajout d'un commentaire**.

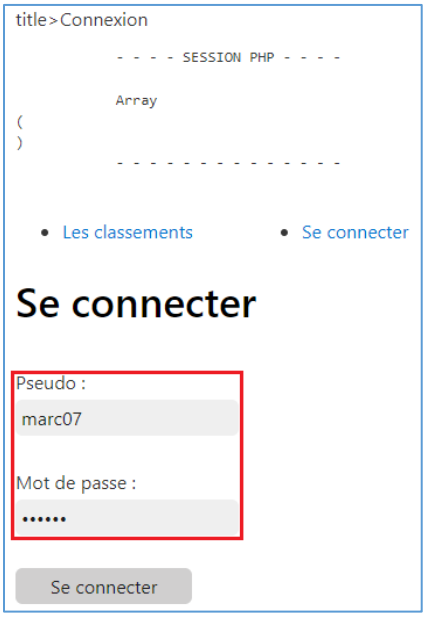
- Un utilisateur authentifié pourra, quant à lui, publier un commentaire (et, donc, avoir accès au formulaire d'ajout d'un commentaire).

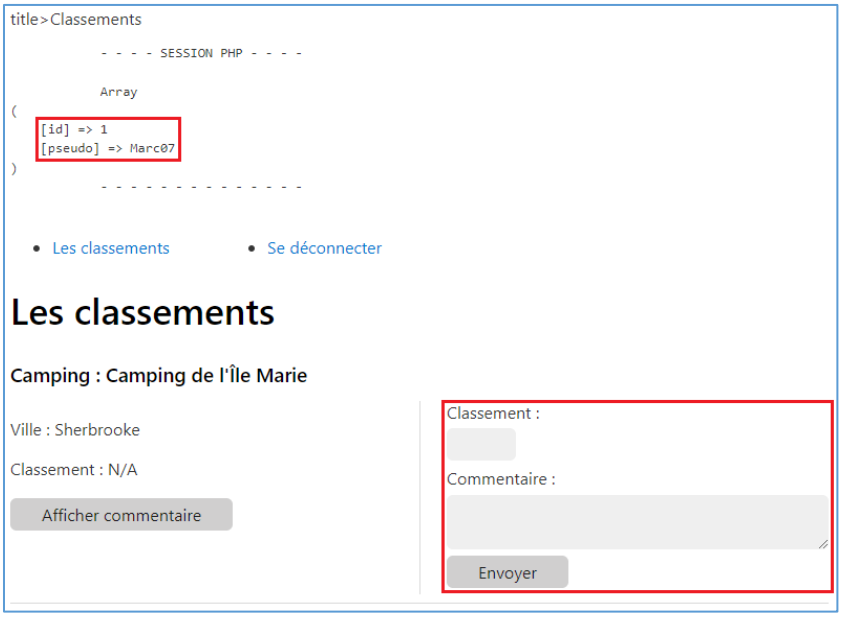
Le formulaire de la page « view/loginView.php » appellera le fichier « index.php » sous la condition « $_REQUEST['action'] === 'authenticate' » (déjà implémentée ainsi dans l'architecture de dossiers/fichiers fournie). Vous devrez cependant :

- Créer une fonction « getFormConnection() » dans le fichier « controller/controllerClient.php » qui chargera le fichier « view/loginView.php » pour que le formulaire d'authentification apparaisse à l'écran.
    - La fonction « getFormConnection() » devra être appelée dans le fichier « index.php » sous la condition « $_REQUEST['action'] === 'login' ».

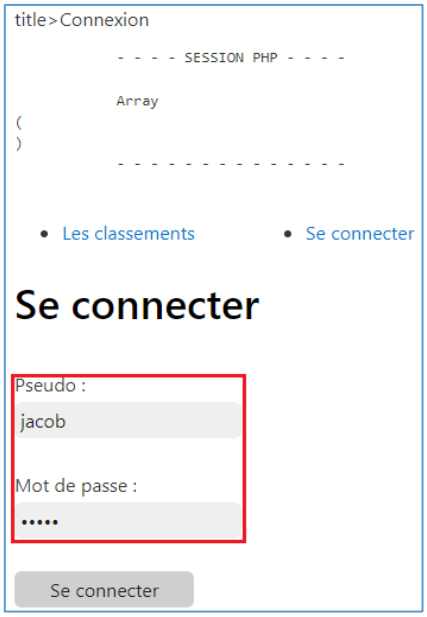
- Créer une fonction « authenticate($username, $password) » dans le fichier « controller/ controllerClient.php » qui utilisera le gestionnaire « model/ClientManager.php » pour vérifier si le nom d'utilisateur (paramètre d'entrée $username) et le mot de passe (paramètre d'entrée $password) concordent avec l'un ou l'autre des utilisateurs inscrits dans la base de données.
    - Si les informations d'authentification correspondent à l'un des utilisateurs inscrits dans la base de données, il faudra enregistrer son ID et son nom d'utilisateur dans la session PHP, puis appeler la fonction « getCampings() » du fichier « controller/controllerCamping.php ».
    - Si les informations d'authentification ne correspondent à aucun utilisateur, il faudra appeler la fonction « getFormConnection() ».

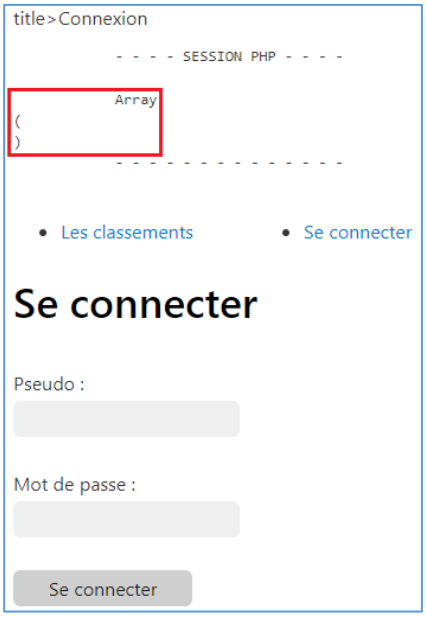
Voici la séquence attendue à l'affichage du formulaire de connexion : un utilisateur valide s'inscrit (par exemple « marc07 »). La page de classement des campings apparaît avec le formulaire pour ajouter un commentaire à côté de chaque camping. De plus, l'ID de l'utilisateur et son nom sont enregistrés dans la session PHP.





À l'inverse, un utilisateur qui n'existe pas dans la base de données tente de s'authentifier. Ce faisant, le formulaire de connexion est automatiquement rechargé et aucune information n'est enregistrée dans la session PHP.

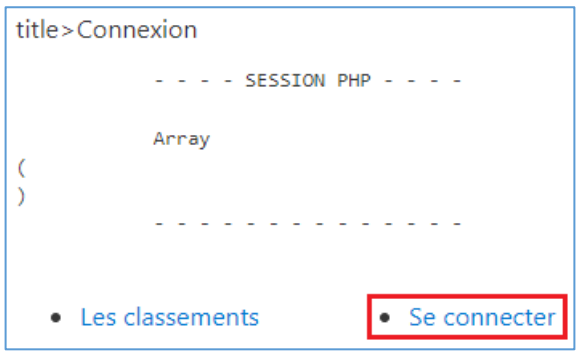




Complétez la condition dans le fichier « index.php » pour la déconnexion d'un utilisateur. Il faut alors supprimer le contenu de la session PHP. Modifiez également le menu de navigation dans le fichier « view/template.php » pour permettre à un utilisateur de se connecter s'il ne l'est pas déjà ou lui permettre de se déconnecter s'il est déjà connecté.

Autrement dit, affichez un lien « Se connecter » si l'utilisateur n'est pas connecté ou affichez plutôt un lien « Se déconnecter » si l'utilisateur est connecté. Ces liens doivent appeler les bonnes actions (« login » ou « logout ») dans le fichier « index.php ».

Si l'utilisateur n'est pas déjà connecté



Si l'utilisateur est déjà connecté

Affichage des commentaires (requêtes « fetch() » ou AJAX et RegEx)

Le bouton « Afficher commentaire » dans la page « view/classementView.php » contient un attribut « onclick » qui appellera une fonction JavaScript « showComments() ». Cette fonction se retrouve dans le fichier « inc/js/script.js », mais elle est pour le moment vide. Vous devez la compléter selon les consignes suivantes :

- Lorsque l'utilisateur clique sur le bouton « Afficher commentaire » la fonction « showComments() » doit lancer une requête « fetch() » ou AJAX avec la méthode POST pour aller chercher, du côté serveur, les

commentaires portant sur le camping concerné. Vous devrez modifier le code HTML pour identifier le camping dont on souhaite rapparier les commentaires dans le code JavaScript.

- Une fois que vous aurez reçu les commentaires du côté du code JavaScript, vous devrez modifier le DOM à l'aide d'une propriété ou d'une méthode comme « innerHTML » ou « insertAdjacentHTML() » pour insérer ces commentaires dans la balise ayant la classe « commentaireCamping » du fichier « view/classementView.php ». **La page Web ne devra pas être rechargée.**

- Comme cela a déjà été mentionné plus tôt dans ce document, le formulaire permettant d'ajouter un commentaire pour un camping donné doit être visible seulement si l'utilisateur est authentifié. Avant l'envoi du formulaire, des expressions régulières doivent valider le contenu de la zone de commentaire :
    - Si toutes les expressions régulières sont validées, le commentaire doit être ajouté dans la table « tbl_commentaire » de la base de données.
    - Si l'une des expressions régulières suivantes n'est pas respectée, l'envoi du formulaire doit être bloqué et un message de type « alert() » doit être affiché, lequel indiquera l'erreur en cause :

        - **Expression régulière 1 :** La zone de commentaire doit contenir au moins deux phrases. Une phrase commence toujours par une majuscule et se termine toujours par un point standard (ne considérez pas les ponctuations particulières comme « ! » ou « ? »).

        - **Expression régulière 2 :** La zone de commentaire ne doit pas contenir de balise HTML. Une balise HTML commence toujours par un chevron ouvrant, contient une série de caractères et se termine par un chevron fermant (par exemple : <nomBalise>).

        - **Expression régulière 3 :** La zone de commentaire ne doit pas contenir de numéro de téléphone français. Un numéro de téléphone français se présente toujours sous la forme de dix chiffres groupés comme suit : +00 00 00 00 00. Le symbole + doit être optionnel.

Du côté du serveur, la réception de la requête « fetch() » ou AJAX et la réception des données du formulaire d'ajout d'un commentaire devront se faire dans la page « index.php », respectivement sous les conditions « $_REQUEST['action'] === 'getComments' » et « $_REQUEST['action'] === 'addComment' ».

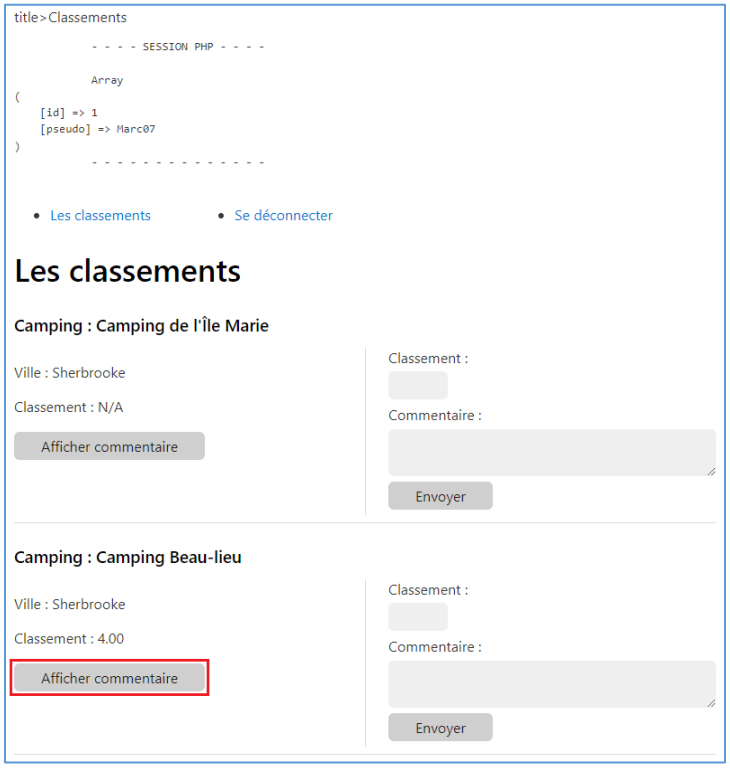
La condition « $_REQUEST['action'] === 'getComments' » appellera une fonction « getComments($idCamping) » qui devra être codée dans le fichier « controller/controllerCommentaire.php ». À son tour, cette fonction utilisera le gestionnaire « model/CommentaireManager.php » pour rapatrier depuis la base de données tous les commentaires relatifs au camping concerné.

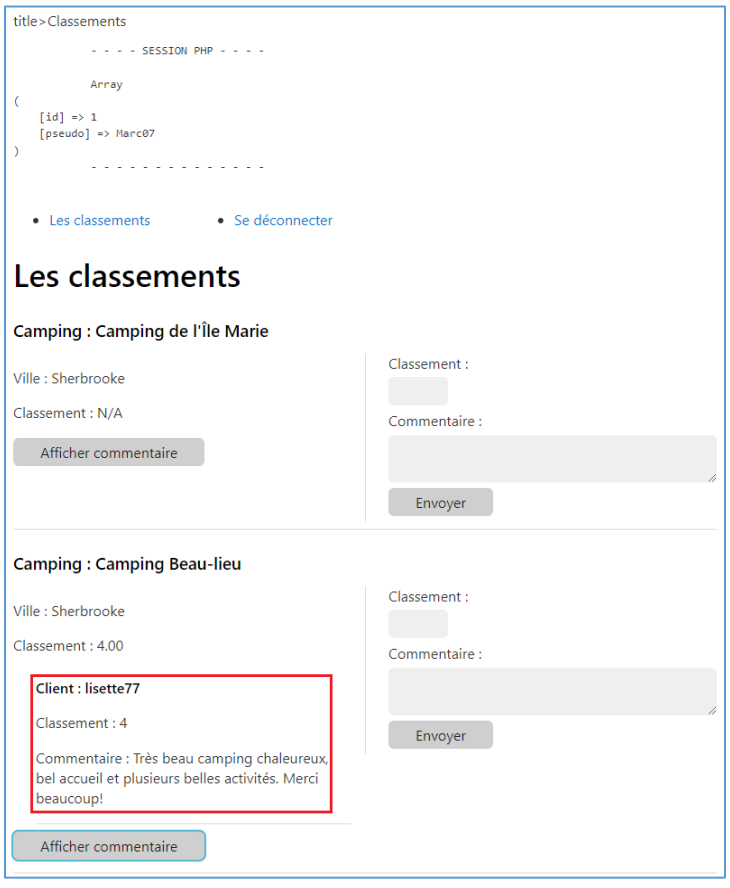
- Si des commentaires existent, il faudra retourner un code de réussite 200 en réponse à la requête « fetch() » ou AJAX, puis transmettre tous les commentaires au fichier « view/ commentairesCampingView.php » qui s'occupera de les afficher.

- Si aucun commentaire n'existe, il faudra simplement afficher un message en console (fonction JavaScript « console.log() ») comme quoi aucun commentaire n'a pu être chargé et retourner un code d'échec 400 en réponse à la requête « fetch() » ou AJAX.

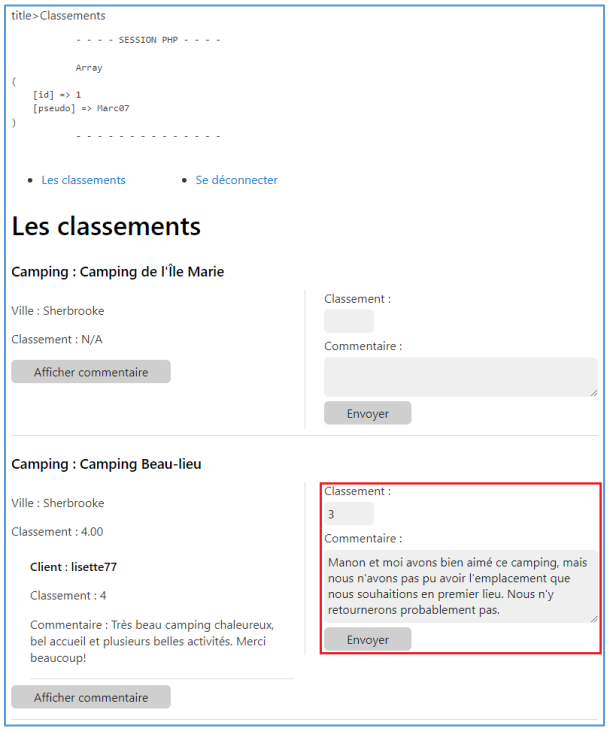
La condition « $_REQUEST['action'] === 'addComment' » appellera une fonction « addComment$idClient, $idCamping, $ranking, $comment) » qui devra, elle aussi, être codée dans le fichier « controller/ controllerCommentaire.php ». Cette fonction devra procéder à l'insertion du commentaire dans la base de données via le gestionnaire « model/CommentaireManager.php » puis appeler la fonction « getCampings() » du fichier « controller/controllerCamping.php ».

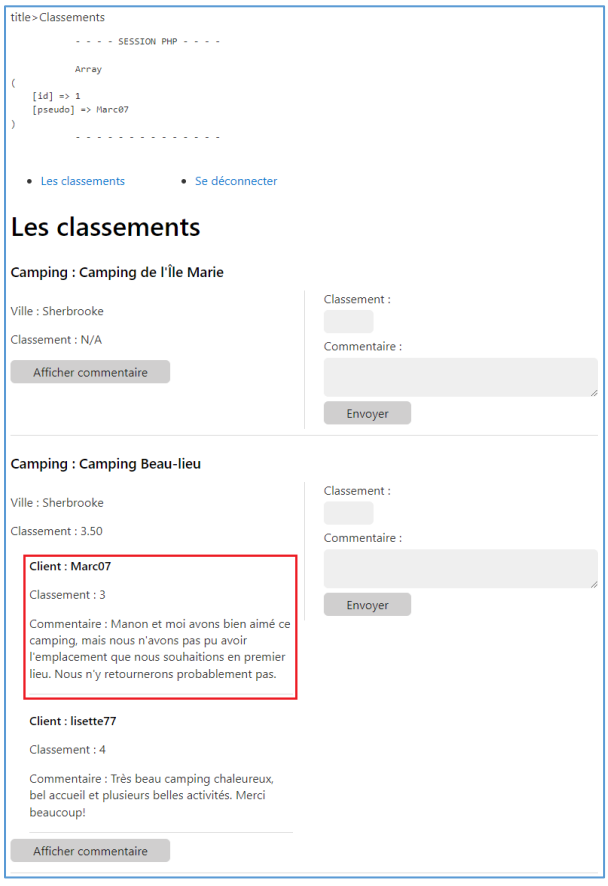
Par exemple, un utilisateur appuie sur le bouton « Afficher commentaire » du camping Beau-lieu. Le seul commentaire qui existe pour ce camping est alors affiché juste au-dessus du bouton (c'est une requête « fetch() » ou AJAX qui a demandé ce commentaire).





Un utilisateur déjà authentifié ajoute une nouvelle note de même qu'un nouveau commentaire pour le camping Beau-lieu. En appuyant sur le bouton « Envoyer » au bas du formulaire, une insertion est faite dans la base de données et la page est automatiquement rechargée. Si l'utilisateur appuie de nouveau sur le bouton « Afficher commentaire » du camping Beau-lieu, la note et le commentaire qu'il vient d'ajouter doivent apparaître.





Programmation orientée objet

Vous devez coder les classes standards « Camping », « Client » et « Commentaire » de même que les classes « managers » « CampingManager », « ClientManager » et « CommentaireManager ». Toutes ces classes standards et classes « managers » se retrouvent dans le dossier « model » du site Web.

Les requêtes SELECT des classes « managers » devront toujours retourner des instances des classes « Camping », « Client » ou « Commentaire » ou encore des tableaux d'instances de ces classes. Il ne sera toutefois pas nécessaire de recevoir une instance de la classe « Commentaire » pour procéder à l'insertion d'un commentaire dans la base de données.

Fiez-vous aux tables de la base de données pour déterminer les attributs privés que devront avoir les classes standards « Camping », « Client » et « Commentaire ». Codez également les accesseurs (« getters ») et les mutateurs (« setters ») pour chaque attribut de chacune de ces classes et n'oubliez pas de coder votre propre constructeur pour chacune d'elles.

Grille de correction

Critère	Excellent	Bon	Moyenne	À améliorer / Incomplet
Qualité du code et respect de la demande	Le code répond à la demande, est complet et exempt d'erreur	Le code répond à la demande, mais contient quelques erreurs mineures	Le code répond partiellement à la demande et contient diverses erreurs	Le code est insuffisant et continent trop d'erreurs
	5	**4**	**3 – 2**	**1 – 0**
Requêtes MySQL via PHP	Les requêtes à la BD sont complètes, conformes et sécuritaires	Les requêtes à la BD sont complètes, mais contiennent des erreurs mineures ou ne sont pas totalement sécuritaires	Les requêtes à la BD ne sont pas toutes présentes ou contiennent diverses erreurs	Plusieurs requêtes à la BD sont absentes et plusieurs erreurs existent dans le code
	10	**9 – 7**	**6 – 3**	**2 – 0**
Programmation orientée objet	Le paradigme de la POO est correctement appliqué dans l'ensemble du site Web	Le paradigme de la POO est majoritairement bien appliqué dans l'ensemble du site Web malgré quelques erreurs mineures	Le paradigme de la POO est mal appliqué à différents endroits dans le site Web, mais cela n'entrave pas son fonctionnement	Le paradigme de la POO est régulièrement mal appliqué ou n'est pas du tout appliqué dans l'ensemble du site Web
	10	**9 – 7**	**6 – 3**	**2 – 0**
Modèle MVC	La division du code respecte bien le modèle MVC	La division du code n'est pas optimale pour respecter le modèle MVC	La division du code permet de voir le modèle MVC, mais contient plusieurs lacunes	Le modèle MVC n'a pas été du tout respecté
	6	**5 – 4**	**3 – 2**	**1 – 0**
Session PHP	La session de PHP est bien utilisée et ne conserve que les éléments nécessaires.	L'utilisation de la session contient de légères lacunes	L'utilisation de la session PHP n'est pas optimale	L'utilisation de la session PHP n'est pas fonctionnelle
	3	**2**	**1**	**0**
Requête « fetch() » ou AJAX	Les requêtes AJAX sont présentes et conformes à la demande	Les requêtes AJAX contiennent de légères lacunes.	Les requêtes AJAX sont présentes, mais contiennent plusieurs erreurs	Les requêtes AJAX ne sont pas fonctionnelles ou simplement absentes
	8	**7 – 6**	**5 – 3**	**2 – 0**

Expressions régulières	Toutes les expressions régulières sont conformes à la demande	Toutes les expressions régulières sont présentes, mais certaines contiennent des erreurs mineures	Certaines expressions régulières sont absentes ou elles contiennent diverses erreurs qui entravent leur fonctionnement	Aucune des expressions régulières n'est fonctionnelle ou elles sont toutes absentes
	8	**7 – 6**	**5 – 3**	**2 – 0**
Total				**/ 50**