Méthodes GET/POST et formulaires HTML

Licence Informatique 3ème année

Alexandre Niveau — Jean-Marc Lecarpentier

Méthodes GET/POST et formulaires HTML

Notes de cours

- **Envoi de données avec HTTP & Formulaires HTML**
  - Envoi de données avec HTTP
  - Méthode POST
  - Fonctionnement général des formulaires HTML
  - Détails sur des widgets particuliers (boutons radio, `select`…)
  - Style et validation côté client

Travail personnel

Objectifs

Dans ce TP, on manipule des formulaires avec les méthodes GET et POST.

Exercice 1 — Pseudo-forum

Récupérer cette archive. Elle contient un répertoire `forum` qui lui-même contient un fichier CSS ainsi qu'un fichier `donnees.php`, qui déclare un tableau `$donnees` contenant les messages d'une sorte de pseudo-forum. Chaque message est un tableau associatif avec trois champs :

- le pseudo
- le texte du message

- la date (une instance de la classe `DateTime`)

Le but de l'exercice est de réaliser un site présentant ces messages. Bien sûr, un vrai site ne stockerait pas les messages de cette façon (dans un tableau écrit en dur dans un script PHP) — l'objectif est de vous faire travailler plusieurs notions (et notamment réviser les formulaires), indépendamment des problèmes spécifiques aux BD (connexion, erreurs de requêtes, etc.).

Site de base

1. Placer le répertoire `forum` sur votre serveur, et créer un fichier `liste.php` dans le répertoire. Inclure le fichier `donnees.php` au début du script `liste.php` : nous pourrons dès lors accéder aux messages dans la variable `$donnees`, qui constituera notre pseudo-base de données.

2. Afficher une liste de tous les messages de la base. NB: les messages ne sont pas dans l'ordre chronologique. C'est normal. Cela fera l'objet d'une question ultérieure.

3. Améliorer la présentation, en vous appuyant sur le fichier CSS fourni : le résultat doit être similaire à cette page (dont il n'est pas interdit de regarder le code source HTML !).

4. S'assurer que le HTML résultant est valide, grâce au validateur du W3C.

5. Modifier le script pour qu'il réagisse à un paramètre `chercher` dans l'URL, en n'affichant que les messages dont le texte contient la chaîne de caractères valeur du paramètre. Par exemple, `liste.php?chercher=blabla` doit afficher tous les messages dont le texte (et seulement le texte — pas le pseudo) contient la chaîne `blabla`, et aucun autre.

6. Ajouter un menu en haut de la page, avec trois liens :
   - « Tous les messages » doit pointer vers `liste.php`
   - « Filtrer » doit pointer vers `filtre.php`
   - « Ajouter » doit pointer vers `ajout.php`

   S'assurer que le HTML est toujours valide.

7. Créer le script `filtre.php`. Il doit contenir un formulaire avec un champ texte servant à faire une recherche dans le texte des messages, en s'appuyant sur le paramètre `chercher` implémenté précédemment dans `liste.php`.

8. S'assurer que le HTML généré par `filtre.php` est valide.

9. Créer le script `ajout.php`. Il doit contenir un formulaire d'ajout de message,

avec un champ pour le pseudo et un champ pour le texte. La soumission du formulaire doit envoyer le message sous la forme d'une requête POST vers `liste.php`.

10. S'assurer que le HTML généré par `ajout.php` est valide.

11. Modifier `liste.php` pour que, lorsque la requête est POST, un nouveau message soit créé (avec le pseudo et le texte donnés, et comme date la date courante) et ajouté au tableau `$donnees`, avant que la liste ne soit affichée.

12. Normalement, vous devez bien voir votre nouveau message dans la liste (a priori, tout en bas)… mais il doit disparaître quand vous cliquez sur « Tous les messages » ! Assurez-vous d'avoir compris pourquoi. Autrement dit : si vous ne comprenez pas ce qui se passe, demandez des explications à votre chargé·e de TP !

L'ajout de message nécessite de mettre en place de la *persistance de données*. On abordera cela dans les prochaines semaines — ce n'est pas l'objectif de cet exercice. On va à présent améliorer notre filtre.

Améliorations du filtre

1. Ajouter un paramètre `pseudo` à la page `liste.php`, qui permet de n'afficher que les messages dont le champ « pseudo » correspond *exactement* au pseudo donné.

   Lors de vos tests, n'oubliez pas de vérifier que les deux paramètres fonctionnent bien *ensemble* : on doit pouvoir rechercher une chaîne dans les messages postés par un pseudo fixé.

2. Ajouter au formulaire de `filtre.php` un champ texte permettant d'utiliser le paramètre `pseudo` que vous venez d'implémenter.

   Une fois que ça fonctionne, remplacer le champ texte par un menu déroulant contenant tous les pseudos de la base de messages. (Il faudra bien sûr inclure la « base de données » dans le script `filtre.php`.)

3. Ajouter un paramètre `après` à la page `liste.php`, qui prend une date et une heure au format `YYYY-MM-DDThh:mm` (par exemple `2025-04-24T14:02:41`). La liste des messages ne doit alors contenir que des messages qui datent d'après la date indiquée dans le paramètre `après`.

4. Ajouter à `liste.php` un paramètre `avant` qui fonctionne comme `après`, mais dans l'autre sens.

   Bien sûr, `avant` et `après` doivent pouvoir fonctionner ensemble — et avec

`chercher` et `pseudo`. Tous ces critères sont orthogonaux et donc compatibles.

5. Ajouter deux champs de type `datetime-local` au formulaire de `filtre.php`, pour exploiter les paramètres `avant` et `après` que vous venez d'implémenter.

6. Vérifiez que `filtre.php` est bien toujours valide !

Encore des améliorations, plus compliquées

1. Ajouter un paramètre `ordre` à `liste.php`.
   a. Faire en sorte qu'il ne puisse prendre que la valeur `pseudo` : lui donner toute autre valeur doit générer une page d'erreur avec comme statut HTTP `400 Bad Request`.
   b. Lorsque `ordre=pseudo`, les messages doivent être ordonnés dans l'ordre alphabétique du pseudo de leur auteur/trice. Il y a plusieurs façons de faire, je n'en connais pas qui soit vraiment satisfaisante.
   c. Ajouter à présent une autre valeur possible, `ordre=texte`, permettant d'obtenir les messages classés dans l'ordre alphabétique de leur texte.
   d. Ajouter encore une autre valeur possible, `ordre=chronologique` : elle doit avoir pour effet que les messages soient classés dans l'ordre chronologique (du plus ancien au plus récent). NB: les opérateurs de comparaison numérique fonctionnent sur les instances de `DateTime`. Une autre possibilité est d'utiliser leur méthode `getTimestamp()`.

2. Ajouter trois boutons radio dans le formulaire `filtre.php` pour choisir l'ordre d'affichage de la liste via le paramètre `ordre`. On regroupera les boutons dans un élément `fieldset`, et on vérifiera une nouvelle fois la validité de la page de formulaire.

3. Ajouter un dernier paramètre à `liste.php` : `inverser`. Lorsqu'il est présent avec la valeur `oui`, l'ordre des messages doit être inversé (par exemple, avec les paramètres `ordre=chronologique&inverser=oui`, l'ordre sera du plus récent au plus ancien). Toute autre valeur que oui doit générer une page d'erreur avec comme statut HTTP `400 Bad Request`.

4. Ajouter une case à cocher au formulaire de `filtre.php` permettant d'inverser l'ordre d'affichage des messages.

5. Faire en sorte que lors de l'utilisation du paramètre `chercher`, le script `liste.php` *surligne* la chaîne de caractère recherchée à l'intérieur des messages listés. L'élément HTML approprié pour cela est `mark`.

Bien vérifier que le surlignement est correct dans les messages contenant des caractères non-ASCII avant la chaîne recherchée. Penser également à vérifier la validité d'une page avec des résultats de recherche.