Architecture d'un site web : manipulation des données

Licence Informatique 3ème année

Alexandre Niveau — Jean-Marc Lecarpentier

Architecture d'un site web : manipulation des données

Notes de cours

- Manipulation des données dans l'architecture MVCR
  - Problèmes liés à la manipulation des données
  - Architecture pour CRUD de base
  - Démonstration sur le site des couleurs (archive)
  - POST-redirect-GET
  - Feedback

Travail personnel

Objectifs

L'exercice est une application directe de l'architecture présentée en cours pour la manipulation des données.

NB: il faut avoir terminé les parties obligatoires du TP précédent avant de commencer celui-ci.

Exercice 1 — Manipulation des données dans MVCR

On va continuer le site sur les animaux fait au TP précédent, en permettant aux internautes d'ajouter leurs propres animaux à la base.

Une nouvelle fois, l'énoncé est long mais c'est parce que l'exercice est très guidé ; il n'y a pas de difficulté particulière. Si vous avez des doutes, jetez un œil à l'exemple des couleurs qui a été en partie présenté en cours (résultat final, archive du code). **Ne faites pas de copier-coller**, réécrire est beaucoup plus efficace pour l'apprentissage. De plus, l'exercice est (comme celui du TP précédent) très incrémental : j'essaie de justifier chaque ajout de complexité dans l'architecture. On ne prendra pas le chemin le plus direct vers la version finale, ne soyez pas surpris si ça ne ressemble pas immédiatement à l'exemple présenté en cours.

❦ Préliminaires

Avant de commencer cet exercice, il faut avoir terminé le TP précédent, au moins jusqu'à la partie « Stockage » incluse. Si vous avez fait la partie « Compléments », il faudra adapter un peu certaines questions, mais vous devriez vous y retrouver.

Ajout d'un menu

Ajouter un attribut `$menu` à la vue, qui contient une liste de couples URL-texte. Cette liste doit être utilisée dans le squelette pour afficher un menu en haut de la page.

Le menu doit contenir (pour l'instant) un lien vers la page d'accueil et un vers la liste des animaux.

Persistance des données

La pseudo-base (non modifiable) qu'on a utilisée ne suffit plus pour cet exercice, puisqu'on veut pouvoir y rajouter des animaux : il faut que les modifications soient persistantes. Pour cela, on pourrait stocker nos animaux dans une BD MySQL, mais pour gagner du temps on va choisir une solution plus simple (au moins au début), celle d'enregistrer notre tableau en session. Ça n'a bien sûr rien à voir avec une vraie persistance des données : les animaux créés ne seront pas accessibles à tous les internautes !

1. Ajoutez les méthodes suivantes à votre interface `AnimalStorage`, en documentant correctement les méthodes avec des commentaires :
   - `create(Animal $a)` ajoute à la base l'animal donné en argument, et retourne l'identifiant de l'animal ainsi créé.
   - `delete($id)` supprime de la base l'animal correspondant à l'identifiant donné en argument ; retourne `true` si la suppression a été effectuée et `false` si l'identifiant ne correspond à aucun animal.
   - `update($id, Animal $a)` met à jour dans la base l'animal d'identifiant donné, en le remplaçant par l'animal donné ; retourne `true` si la mise à jour a bien été effectuée, et `false` si l'identifiant n'existe pas (et donc rien n'a été modifié).

   Ajoutez une implémentation de ces méthodes dans `AnimalStorageStub` : elles

doivent simplement jeter une exception (elles n'ont pas de sens sur cette implémentation).

2. On vous fournit la classe `AnimalStorageSession`, qui implémente `AnimalStorage` en utilisant une session PHP (les animaux créés sont donc locaux à la session de l'internaute, et disparaîtront à la fin de la session). Télécharger le fichier correspondant dans cette archive, et le placer dans `src/model`.
3. Modifier `site.php` pour que le système utilise un `AnimalStorageSession` à la place de l'`AnimalStorageStub`. Attention, il faut lancer une session *avant* de créer l'instance de `AnimalStorageSession`, mais *après* avoir inclus la classe (car `session_start` désérialise les animaux en session, et pour ça PHP a besoin de connaître la classe).
4. Vérifier que votre site fonctionne toujours comme avant.

Affichage debug

Ajouter la méthode suivante à la vue :

```
public function prepareDebugPage($variable) {
        $this->title = 'Debug';
        $this->content = '<pre>'.htmlspecialchars(var_export($variable, tru
}
```

Elle va faciliter le debug en nous permettant d'afficher le contenu d'une variable. La tester depuis le routeur en lui faisant afficher divers objets.

> **Avant de passer à la suite, faites un commit du code courant avec comme message « Préliminaires TP 09 ».** Plus précisément, ouvrez un terminal (local) dans `exoMVCR` (pas dans `src` !) et exécutez les commandes suivantes :
>
> ```
> git add .
> git commit -m "Préliminaires"
> git push
> ```
>
> Si vous faites ensuite un `git status`, Git devrait vous dire que « la copie de travail est propre ». Si ce n'est pas le cas, il y a eu un problème, voyez avec votre chargé·e de TP.

❦ Création d'un nouvel animal

1. La page avec le formulaire de création d'un animal sera (par exemple) à l'URL `site.php?action=nouveau`, et la page destinée à recevoir les données sera à l'URL `site.php?action=sauverNouveau`. Ajouter au routeur des méthodes `getAnimalCreationURL()` et `getAnimalSaveURL()` qui renvoient ces URLs.
2. Ajouter une méthode `prepareAnimalCreationPage()` à la vue. Elle doit afficher

un formulaire de création d'un animal, avec trois champs texte : `nom`, `espece` et `age`. Le formulaire doit envoyer les données en `POST` à l'URL donnée par le `getAnimalSaveURL()` du routeur.

3. Ajouter deux méthodes au contrôleur :
   - `createNewAnimal()`, qui demande à la vue de préparer le formulaire
   - `saveNewAnimal(array $data)`, qui se contente pour l'instant d'afficher son argument grâce à `prepareDebugPage`.
4. Modifier le routeur pour qu'il analyse le paramètre `action` de l'URL, et qu'il appelle `createNewAnimal()` si le paramètre est `nouveau`, et `saveNewAnimal($_POST)` si le paramètre est `sauverNouveau`. Cela permet de tester que tout fonctionne pour le moment.
5. Dans la méthode `saveNewAnimal`, au lieu d'afficher le tableau passé en argument, l'utiliser pour créer une instance de Animal et l'afficher avec `prepareAnimalPage`. Ça devrait fonctionner, mais l'animal n'est pas ajouté à la base (on peut le constater en affichant la liste des animaux).
6. Faire en sorte que `saveNewAnimal` ajoute le nouvel animal à la base avant de le passer à la vue.
7. Vérifier que tout marche bien : on doit pouvoir créer des animaux qui s'ajoutent à la liste.

> **Avant de passer à la suite, faites un commit du code courant avec comme message « Création ».** (voir les instructions ici (adapter le message de commit)).

❦ Validation

Les internautes peuvent maintenant ajouter leurs animaux favoris à votre site. Cependant, iels peuvent aussi envoyer des données incorrectes, par exemple en laissant les champs vides. On va commencer par faire une validation de base, et on gérera les données incomplètes dans la section suivante.

1. Modifier `saveNewAnimal()` pour qu'il ne soit pas possible d'ajouter un animal dont le nom ou l'espèce soient vides ou dont l'âge ne soit pas un nombre positif. On pourra afficher une page d'erreur par exemple.
2. Cette solution n'est pas idéale : en cas d'erreur, l'internaute perd ce qu'il ou elle a entré. Il est bien plus ergonomique de lui redonner la main sur le formulaire avec les champs tels qu'ils ont été remplis. Pour cela, en cas d'erreur, le contrôleur va redonner le tableau qu'il a reçu à la méthode `prepareAnimalCreationPage` de la vue, pour qu'elle remplisse les champs du formulaire avec. Faire les modifications nécessaires. (Attention notamment à ce que tous les appels à `prepareAnimalCreationPage` soient corrects.)
3. C'est mieux, mais maintenant il est moins clair pour l'internaute qu'une erreur est

survenue. D'autre part, la raison pour laquelle les données étaient invalides n'est pas forcément évidente : il est nécessaire d'en informer l'internaute. Faire en sorte que le contrôleur passe une chaîne `$error` à `prepareAnimalCreationPage` qui se chargera de l'afficher. La chaîne sera `null` s'il n'y avait pas d'erreur, et contiendra sinon une explication sur l'erreur.

4. Vérifier que tout fonctionne, c'est-à-dire le cas normal et tous les cas d'erreur.
5. Essayer d'ajouter un animal qui aurait pour espèce « `<script>alert('coucou')</script>` » (ou mettez cette chaîne de caractères dans un des champs du formulaire qui se retrouvera affiché sur la page de l'animal). Que se passe-t-il ? Que faut-il faire pour empêcher ça ?

> **Avant de passer à la suite, faites un commit du code courant avec comme message « Validation ».** (voir les instructions ici (adapter le message de commit)).

❦ Gestion des données incomplètes

Il y a deux problèmes avec notre manipulation des données. D'abord, c'est le contrôleur qui décide quelles données sont valides (alors que ça concerne le modèle). D'autre part, il y a duplication du nom des champs de formulaire, dans la vue et dans le contrôleur. C'est moins gênant que pour les paramètres d'URL, car ils sont moins visibles, mais ils font cependant partie de l'interface « externe » du site (pas comme les noms des variables PHP, par exemple), et à ce titre ils se doivent d'être relativement faciles à changer.

On va gérer ces deux problèmes à la fois en ajoutant une nouvelle classe au modèle, `AnimalBuilder`, qui représente un animal en cours de manipulation (création ou modification) dans l'application, et qui permet de construire l'instance de `Animal` correspondante.

1. Créer cette classe et lui donner un attribut `$data`, passé en argument à son constructeur, et un attribut `$error`, initialisé à `null`. Ajouter un accesseur pour chacun de ces attributs.
2. Modifier la méthode `saveNewAnimal` du contrôleur pour qu'elle crée une instance de `AnimalBuilder` avec le tableau `$data` qu'elle a elle-même reçu du routeur.
3. Ajouter une méthode `createAnimal()` dans `AnimalBuilder`, qui crée une nouvelle instance de `Animal` en utilisant l'attribut `$data` (déplacer le code depuis le contrôleur).
4. Ajouter une méthode `isValid()` dans `AnimalBuilder`, qui vérifie que les données de son attribut `$data` sont correctes (déplacer à nouveau le code depuis le contrôleur). Si elles ne le sont pas, placer la chaîne expliquant l'erreur dans l'attribut `$error`.
5. Modifier la méthode `prepareAnimalCreationPage` pour qu'elle ne prenne

comme argument qu'une instance de `AnimalBuilder`, et l'utilise pour pré-remplir les champs et pour afficher l'erreur éventuelle.

6. Modifier `saveNewAnimal` pour utiliser l'instance de `AnimalBuilder`. Vérifier que tout marche toujours (normalement non : il faut modifier l'appel initial à `prepareAnimalCreationPage`, qui doit récupérer un `AnimalBuilder` vide ; qui doit s'occuper de le construire ?)
7. Il ne reste plus qu'à « cacher » les noms des champs. Ajouter à `AnimalBuilder` des constantes `NAME_REF`, `SPECIES_REF` et `AGE_REF`, qui contiennent respectivement `nom`, `espece` et `age`. Utiliser ces constantes dans la classe en question, et dans `prepareAnimalCreationPage` à la place des noms de champ codés « en dur ».
8. Vérifier que tout marche toujours, puis changer les noms des champs (les passer en majuscules par exemple). Normalement, vous ne devriez avoir à modifier que les constantes de `AnimalBuilder`, et tout doit toujours marcher (et les noms des champs modifiés se verront dans le HTML).

> **Avant de passer à la suite, faites un commit du code courant avec comme message « AnimalBuilder ».** (voir les instructions ici (adapter le message de commit)).

❦ POST-redirect-GET

Normalement, lors de la création d'un nouvel animal, le contrôleur appelle le `prepareAnimalPage` de la vue. Comme vu en cours, ce n'est pas une bonne chose : on va plutôt demander au client de faire une nouvelle requête, en `GET`, vers la page de l'animal nouvellement créé.

1. Ajouter une méthode `POSTredirect($url, $feedback)` au routeur, qui envoie une réponse HTTP de type `303 See Other` demandant au client de se rediriger vers l'URL passée en argument (le paramètre `$feedback` est pour l'instant ignoré).
2. Ajouter une méthode `displayAnimalCreationSuccess($id)` à la vue, qui utilise la méthode créée ci-dessus pour rediriger le client vers la page de l'animal dont l'identifiant est passé en paramètre (passer une chaîne quelconque comme feedback pour l'instant). Dans le contrôleur, lorsqu'un nouvel animal a été correctement ajouté, appeler cette nouvelle méthode plutôt que `prepareAnimalPage`.
3. Tester. Vérifier notamment que l'actualisation de la page fonctionne correctement après ajout d'un nouvel animal.

> **Avant de passer à la suite, faites un commit du code courant avec comme message « Redirection après POST ».** (voir les instructions ici (adapter le

message de commit)).

❦ Feedback

On voudrait donner un feedback à l'internaute, pour l'informer sur le résultat de ses actions (succès ou erreur).

1. Ajouter un attribut `$feedback` à la vue, passé à son constructeur. Le squelette devra afficher le contenu de cet attribut quelque part en haut de la page.
2. Modifier le routeur pour qu'il passe la chaîne `"Test"` comme feedback lorsqu'il construit la vue. Vérifier que ça fonctionne.
3. À présent, on va utiliser une variable de session `$_SESSION['feedback']` pour passer le feedback d'une requête à l'autre. Commencer une session au début du `main` du routeur, utiliser la variable de session `$_SESSION['feedback']` pour construire la vue, puis supprimer le contenu de cette variable (le feedback n'est affiché qu'une seule fois).
4. Modifier la méthode `POSTredirect` pour qu'elle mette dans `$_SESSION['feedback']` le contenu de son paramètre `$feedback`, avant la redirection.
5. Tester (et mettre un message de feedback significatif).

Avant de passer à la suite, faites un commit du code courant avec comme message « Feedback ». (voir les instructions ici (adapter le message de commit)).