Architecture d'un site web

Licence Informatique 3ème année

Alexandre Niveau — Jean-Marc Lecarpentier

Architecture d'un site web

Notes de cours

- Architecture d’un site web
  - Séparation traitement des données & affichage
  - Une architecture MVC pour un site web

Travail personnel

► **Cette séance est le début d'un «TP filé» dans lequel vous allez construire incrémentalement un site web conformément à l'architecture présentée en cours. Le résultat de ce « TP fil rouge », auquel on vous demandera d'ajouter quelques compléments, constituera votre DM/projet, et est à faire en binôme. Vous rendrez à la fin du mois une archive de code par binôme. Votre rendu fera l'objet d'un oral très court, visant à nous assurer que vous avez compris ce que vous avez rendu. Par ailleurs, il vous est demandé dans l'énoncé de faire des commits régulièrement (sur le dépôt de votre groupe, créé sur le Gitlab de l'université). Cela fera partie de l'évaluation. (NB: si vous avez déjà suivi l'unité l'an passé, vous devez refaire l'exercice.)** ◄

Exercice 1 — Introduction à l'architecture MVCR

Cet exercice vise à construire un site web très simple utilisant l'architecture MVCR présentée en cours.

L'objectif est d'introduire pas à pas, en les justifiant, chacun des choix de conception présentés en cours. L'énoncé est donc très long, mais c'est parce que l'exercice est très guidé : il n'y a aucune difficulté particulière jusqu'à la section « Compléments ».

À la fin de l'exercice, vous devriez arriver à un résultat proche de l'exemple du site des couleurs présenté (en partie) en cours. Si nécessaire, vous pouvez regarder comment fonctionne le résultat final (version en ligne, archive du code), cependant **cet énoncé étant très incrémental, vous passerez par des étapes que vous ne retrouverez pas forcément dans le résultat final du site des couleurs**. N'utilisez surtout pas ce code pour faire du copier-coller, vous risquez de vous embrouiller dans les questions. C'est un outil de compréhension du cours, pas un corrigé.

❦ Préliminaires

1. Cloner votre dépôt Gitlab quelque part sur votre espace web. Si vous ne l'avez encore jamais cloné, suivez les instructions données sur la page du dépôt («Create a new repository»). Dans toute la suite, on se place dans le répertoire en question — j'utiliserai le nom `exoMVCR`, mais pour vous ce sera quelque chose comme `groupe-42`.
2. Modifier le fichier `README.md` pour y mettre les noms, numéros étudiants et logins des deux membres du binôme.
3. Créer dans `exoMVCR` un répertoire `src`, qui contiendra tout le code source de l'application (c'est-à-dire tout ce qui n'est pas censé être *directement* accessible depuis un client).
4. Dans un fichier `src/Router.php`, créer une classe `Router` contenant uniquement une méthode `main` qui affiche quelque chose (par exemple « Hello world »).
5. Créer un fichier `site.php` à la racine de `exoMVCR` avec le contenu suivant :

```
<?php
/*
 * On indique que les chemins des fichiers qu'on inclut
 * seront relatifs au répertoire src.
 */
set_include_path("./src");

/* Inclusion des classes utilisées dans ce fichier */
require_once("Router.php");
```

```
/*
 * Cette page est simplement le point d'arrivée de l'internaute
 * sur notre site. On se contente de créer un routeur
 * et de lancer son main.
 */
$router = new Router();
$router->main();
?>
```

Normalement, en allant à l'URL du répertoire (par exemple `https://dev-`*`NUMETU`*`.users.info.unicaen.fr/exoMVCR/site.php`, en adaptant le chemin), vous devriez voir le message du routeur.

> **Avant de passer à la suite, faites un commit du code courant avec comme message « Préliminaires ».** 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.

❦ Vue

1. Créer un répertoire `view` dans `src`, contenant une classe `View` avec deux attributs, `$title` et `$content`, et une méthode `render()` qui affiche une page HTML avec le contenu de ces attributs, `$title` étant utilisé comme titre de la page, dans l'élément `head` et comme un `h1` dans le `body`, et `$content` étant le contenu du `body` après le `h1`. (La méthode `render()` peut se contenter d'inclure un squelette défini dans un autre fichier, mais ce n'est pas obligatoire.)
2. Ajouter une méthode `prepareTestPage()` à `View`, qui remplit les attributs avec du texte quelconque. **Attention :** elle ne doit rien afficher, seulement remplir les attributs ! L'appel à `render` est le rôle du routeur : dans le `main` du routeur, créer une instance de `View` et lui faire afficher la page de test. Tester le résultat, et **vérifiez la validité de la page obtenue.**
3. Ajouter une méthode `prepareAnimalPage($name, $species)` à `View`, qui génère l'affichage d'une page sur l'animal passé en argument : par exemple, la page générée par l'appel `prepareAnimalPage("Médor", "chien")` aura pour titre « Page sur Médor » et pour contenu « Médor est un animal de l'espèce chien ». Tester cet affichage depuis le routeur.

On a maintenant une page qui affiche des informations sur Médor. On va faire en

sorte qu'elle puisse en afficher sur divers animaux, en fonction de l'URL. Pour cela, on va d'abord créer un contrôleur pour gérer les informations.

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

❦ Contrôleur

1. Créer un répertoire `src/control`, contenant une classe `Controller` avec un attribut `$view`. Le constructeur doit prendre en argument une instance de `View` qu'il met dans cet attribut.
2. Ajouter la méthode suivante dans le contrôleur :

   ```
   public function showInformation($id) {
           $this->view->prepareAnimalPage("Médor", "chien");
   }
   ```

   Modifier le routeur pour qu'il fasse appel à cette méthode (avec un argument quelconque) plutôt que d'appeler lui-même `prepareAnimalPage`. Vérifier que tout fonctionne toujours comme avant, et corriger votre code si ce n'est pas le cas.
3. Ajouter un attribut `$animalsTab` au contrôleur, contenant le tableau suivant :

   ```
   array(
           'medor' => array('Médor', 'chien'),
           'felix' => array('Félix', 'chat'),
           'denver' => array('Denver', 'dinosaure'),
   );
   ```

   (et ce que vous voulez d'autre…). Utiliser ce tableau dans `showInformation` pour que le site affiche que Médor est un chien, Félix un chat, Denver un dinosaure, etc., en fonction de l'argument `$id` passé à la méthode, mais affiche un message d'erreur (« Animal inconnu ») s'il ne connaît pas l'identifiant. Tester depuis le routeur en passant différents identifiants.
4. Si ce n'est déjà fait, faire en sorte que le message d'erreur mentionné à la question précédente soit généré par une méthode `prepareUnknownAnimalPage()` de la vue.

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

❦ Routeur

1. Modifier le routeur pour qu'il passe en argument à la méthode `showInformation` du contrôleur la valeur du paramètre `id` de l'URL.
2. Tester : votre site devrait dire « Médor est un animal de l'espèce chien » si on met le paramètre `id=medor` dans l'URL, « Denver est un animal de l'espèce dinosaure » pour `id=denver`, etc., et devrait afficher le message d'erreur pour des identifiants inconnus.
3. Que se passe-t-il s'il n'y a pas de paramètre `id` dans l'URL ? Faire en sorte qu'une page d'accueil s'affiche, en modifiant le routeur, le contrôleur et la vue.

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

❦ Modèle

On voudrait à présent donner plus d'information sur chaque animal : outre son espèce, la page d'un animal doit maintenant donner son âge. Cela reste possible d'ajouter un argument `age` à la méthode `prepareAnimalPage` de la vue, mais on voit bien les limites de cette approche : que se passerait-il si on avait des dizaines d'informations à donner sur chaque animal (taille, poids, nourriture préférée, etc.) ?

La solution est de manipuler les animaux comme des instances d'une classe `Animal`.

1. Dans un répertoire `src/model`, créer une classe `Animal`, qui a comme attributs (au minimum) un nom, une espèce et un âge. Créer le constructeur et les accesseurs appropriés (pas besoin de mutateurs, on ne modifiera pas les instances).
2. Modifier le tableau `$animalsTab` dans le contrôleur pour qu'il contienne des instances de `Animal` (attention : PHP ne voudra pas que vous définissiez le tableau en-dehors du constructeur dans ce cas), et changer la méthode `prepareAnimalPage` de la vue pour qu'elle ne prenne plus qu'un `Animal` en argument. Vérifier que tout fonctionne toujours.
3. Modifier la vue pour qu'elle affiche l'âge de l'animal.

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

❦ Liste des animaux

On voudrait à présent faire une page spéciale, d'URL `site.php?action=liste`, qui

affiche la liste de tous les animaux dont le site connaît l'espèce.

1. Ajouter une méthode `prepareListPage()` à la vue (qui pour l'instant affiche un contenu quelconque), une méthode `showList()` au contrôleur qui appelle `prepareListPage()`, et faire en sorte que le routeur appelle `showList()` lorsque l'URL contient un paramètre `action` de valeur `liste`. Vérifier que ça fonctionne.
2. Faire en sorte que `prepareListPage` prenne en argument un tableau d'instances d'`Animal` et affiche une liste des noms.
3. Modifier `showList` dans le contrôleur pour qu'elle passe à `prepareListPage` la liste de tous les animaux.
4. Faire en sorte que chaque nom dans la liste soit un lien vers la page de l'animal en question.
5. Pour faire la question précédente, avez-vous pris garde à ce que la vue ne fasse pas d'hypothèses sur la façon dont sont construites les URL ? Si ce n'est déjà fait, créer une méthode `getAnimalURL($id)` dans le routeur, qui renvoie l'URL de la page d'un animal. Attention, la vue doit donc avoir accès à une instance du routeur : l'ajouter comme attribut et comme argument au constructeur.

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

❦ Stockage

Pour l'instant, c'est le contrôleur qui « décide » des animaux connus par le site. Généralement, ce type d'information est plutôt contenu dans une base de données. On ne va pas utiliser de BD dans ce TP, mais on peut déjà adapter le contrôleur pour qu'il fonctionne dans tous les cas, BD ou non.

1. Créer une interface `AnimalStorage` dans le répertoire `src/model` ; elle aura pour méthodes
   - `read($id)`, dont les implémentations doivent renvoyer l'instance de `Animal` ayant pour identifiant celui passé en argument, ou `null` si aucun animal n'a cet identifiant ; et
   - `readAll()`, dont les implémentations doivent renvoyer un tableau associatif identifiant ⇒ animal contenant tous les animaux de la « base ».
2. Créer une classe `AnimalStorageStub` dans `src/model` qui implémente l'interface `AnimalStorage` en utilisant des données écrites « en dur » (reprendre le tableau `$animalsTab` du contrôleur).
3. Ajouter un argument de type `AnimalStorage` au constructeur du contrôleur. Le contrôleur doit enregistrer cette instance dans un attribut, et l'utiliser dans ses

méthodes `showInformation` et `showList` à la place du tableau `$animalsTab`.

4. Modifier le code du routeur pour qu'il passe une nouvelle instance de `AnimalStorageStub` au constructeur du contrôleur.
5. À présent, le contrôleur est complètement indépendant de la façon dont sont stockées les données. C'est le routeur qui décide d'utiliser un `AnimalStorageStub`, mais il aurait pu utiliser un `AnimalStorageMySQL`, par exemple, sans que le contrôleur ne s'en rende compte. Cependant, ce n'est pas le rôle du routeur de choisir l'implémentation de l'interface de stockage !

   Faire en sorte que l'instance de `AnimalStorage` que le routeur utilise pour créer le contrôleur lui soit passée en paramètre de la méthode `main`. Ce sera donc `site.php`, c'est-à-dire le véritable point d'entrée de l'application, qui va prendre cette décision. (S'il prend plusieurs décisions de ce type, ce qui arrive très rapidement sur un site modérément compliqué, il peut être plus naturel de les déporter à nouveau dans un fichier de configuration, que `site.php` se contente d'inclure.)

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

❦ Compléments (optionnel)

1. Comme on l'a vu en cours, les paramètres d'URL ne sont pas destinés à faire du routage. Il est relativement simple d'adapter le code pour que les URL soient de la forme `site.php/denver`, en utilisant la variable `$_SERVER['PATH_INFO']`, qui contient la « partie virtuelle » de l'URL (le « chemin » qui se situe après le nom du script).

   Copiez votre routeur dans une nouvelle classe `PathInfoRouter`, modifiez le script `site.php` pour qu'il utilise une instance de cette classe comme routeur, puis adaptez la classe pour utiliser le `PATH_INFO` au lieu des paramètres d'URL. Si votre code est propre, vous ne devriez avoir à changer que les méthodes `main` et `getAnimalURL` de votre nouveau routeur, et aucun autre fichier.

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