BDW - Programmation web - structuration

Fabien Duchateau

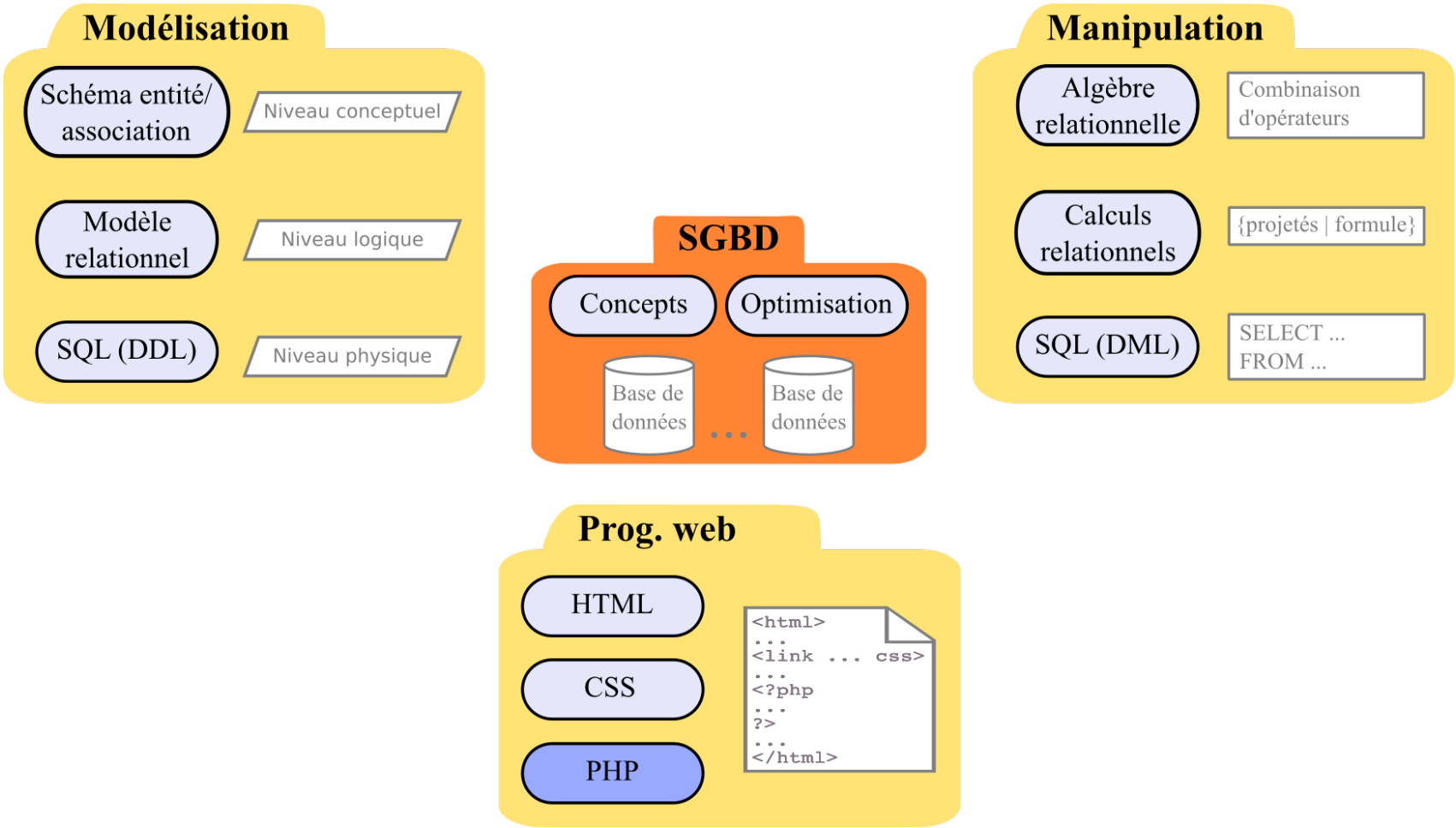
fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2023 - 2024

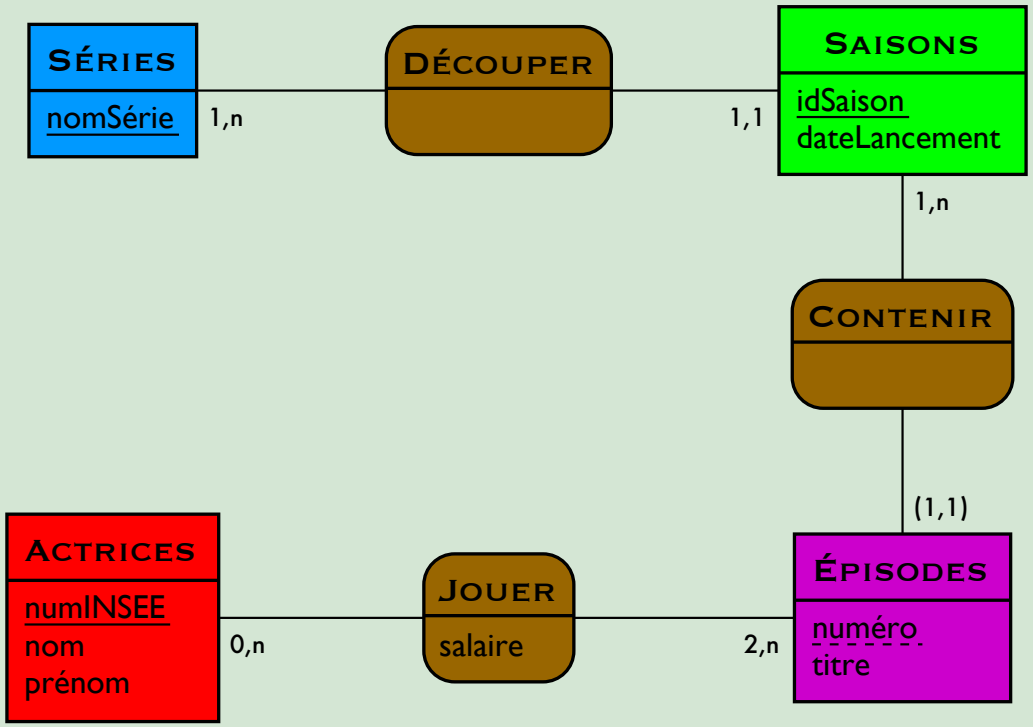
https://perso.liris.cnrs.fr/fabien.duchateau/BDW/

Positionnement dans BDW

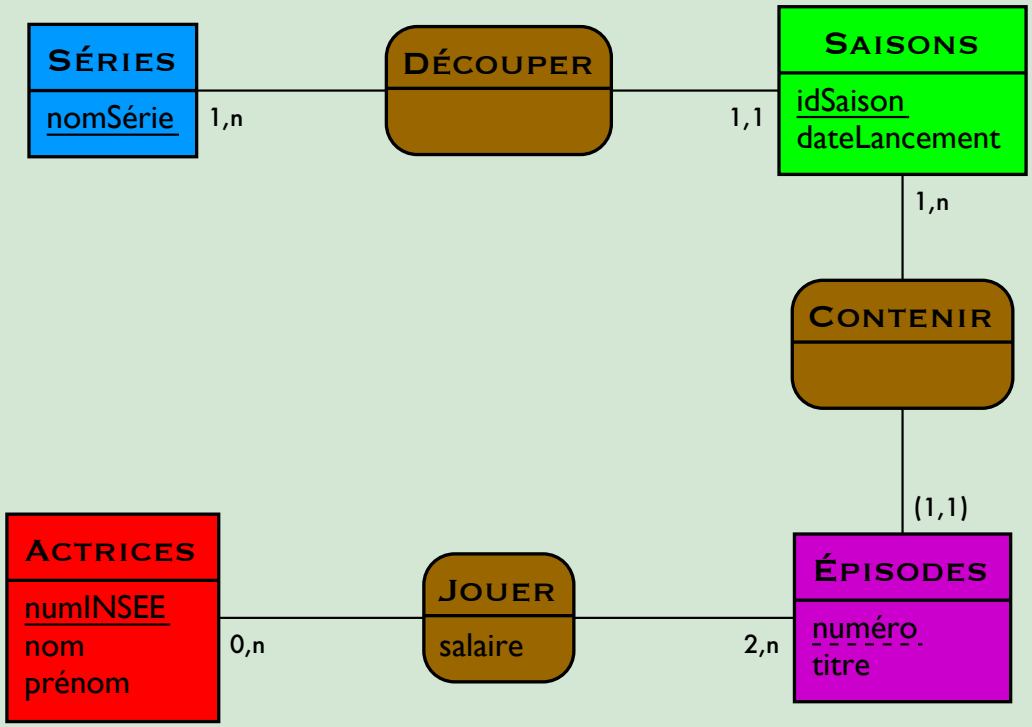


Ces diapositives utilisent le genre féminin (e.g., chercheuse, développeuses) plutôt que l'écriture inclusive (moins accessible, moins concise, et pas totalement inclusive)

Exemple sur les séries - modèles de données



Exemple sur les séries - modèles de données



SÉRIES (nomSérie)
SAISONS (idSaison, dateLancement, #nomSérie)
ÉPISODES (numéro, #idSaison, titre)
ACTRICES (numINSEE, nom, prénom)
JOUER (#numéro, #idSaison, #numINSEE, salaire)

Exemple sur les séries - fonctionnalités

Réalisation d'un site web pour :

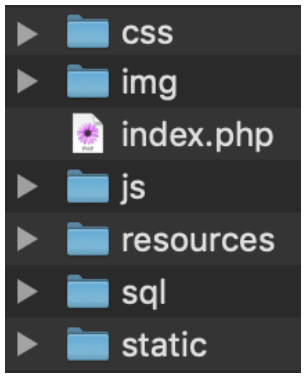
- Afficher toutes les séries
- Ajouter une série
- Rechercher une actrice

Plan

Organisation d'un site web

Différentes manières d'organiser un répertoire, mais des éléments :

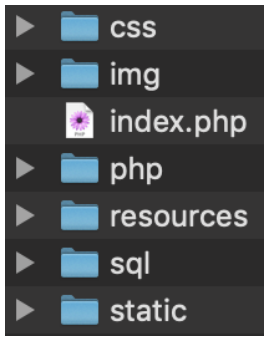
- page d'accueil (e.g., `index.php` ou `index.html`)
- répertoire `img/` pour les images
- répertoire `inc/` ou `resources/` pour les fichiers spéciaux (e.g., configuration, librairies)
- répertoire `css/` pour les fichiers de styles
- répertoire `js/` pour les scripts JavaScript (si utilisé)
- répertoire `sql/` pour les scripts SQL
- répertoire `static/` (pages statiques comme le contenu de l'accueil ou parties communes à toutes les pages comme le menu, l'entête ou le pied de page)



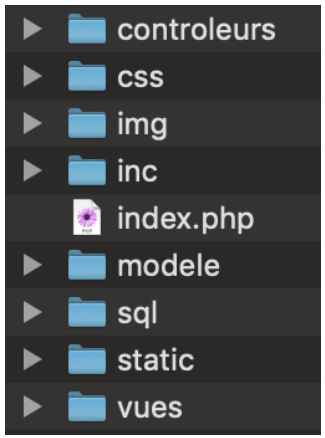
Organisation d'un site web (2)

Fichiers de contenu organisés selon la structuration choisie :

- répertoire `php/` (pages dynamiques, qui peuvent être organisées par fonctionnalité)

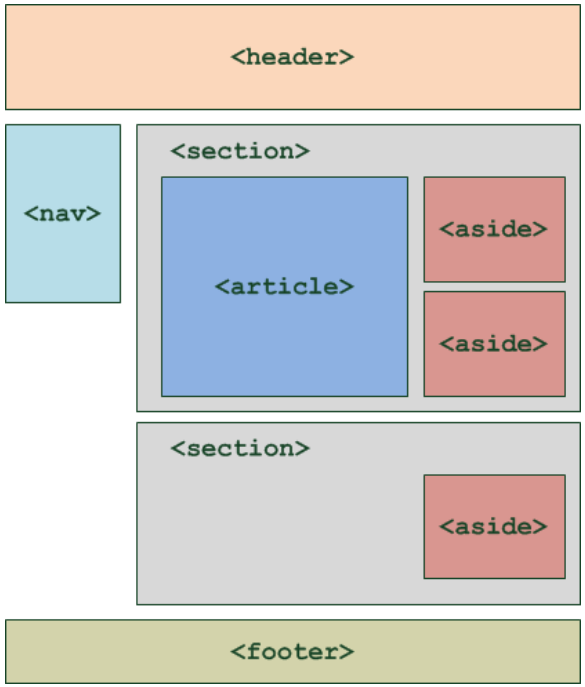


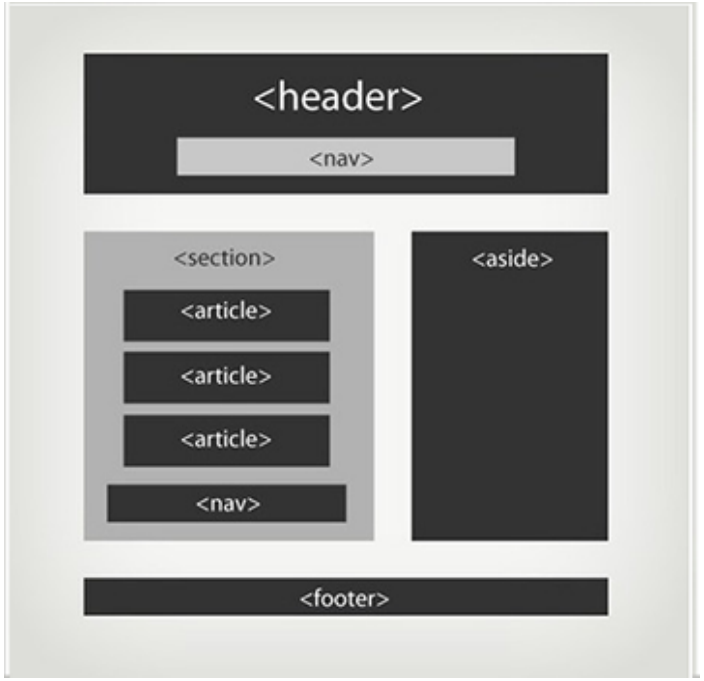
- répertoires `modele/`, `vues/` et `controleurs/` en architecture MVC



Comment structurer une page ?

Toutes les pages d'un site respectent en général une même structuration (*design*)





Exemples de design avec les balises structurantes

Exemple de mauvaise structuration des pages

afficher.php

```
1  <header>
2    <img src="logo.jpg" />
3    <h1>Serial critique</h1>
4  </header>
5
6  <main>
7    <h1>Liste des séries</h1>
8    ...
9  </main>
```

ajouter.php

```
1  <header>
2    <img src="logo.jpg" />
3    <h1>Serial critique</h1>
4  </header>
5
6  <main>
7    <h1>Ajouter une série</h1>
8    ...
9  </main>
```

Code non factorisé : la partie <header>, identique pour chaque page, apparait dans les fichiers `afficher.php` et `ajouter.php`

Exemple de mauvaise structuration des pages

afficher.php

```
1  <header>
2    <img src="logo.jpg" />
3    <h1>Serial critique</h1>
4  </header>
5
6  <main>
7    <h1>Liste des séries</h1>
8    ...
9  </main>
```

ajouter.php

```
1  <header>
2    <img src="logo.jpg" />
3    <h1>Serial critique</h1>
4  </header>
5
6  <main>
7    <h1>Ajouter une série</h1>
8    ...
9  </main>
```

Code non factorisé : la partie <header>, identique pour chaque page, apparait dans les fichiers `afficher.php` et `ajouter.php`

Inclusion de fichiers

En PHP, il est possible d'inclure un fichier externe :

- Équivalent à copier-coller le contenu du fichier inclus à la place de la fonction d'inclusion
- Regroupement de fonctions (liées à la manipulation de BD par exemple) ou de variables
- Découpage des différentes parties d'un site

Fonctions d'inclusion `include` ou `require` :

- Avertissement avec `include` si fichier non trouvé
- Erreur fatale avec `require` si fichier non trouvé
- Variantes avec `include_once` et `require_once`

Syntaxe de l'inclusion de fichiers

```
include("chemin/nom_fichier.php");
require("chemin/nom_fichier.php");
```

exemples.php

```
1 <?php // inclusion
2 include("fonctions.php");
3 echo sommer(1, 2);
4 ?>
```

fonctions.php

```
1 function sommer($a, $b) {
2   return $a + $b;
3 }
```

Le fichier de gauche inclut le fichier de droite (ligne 2) : cela équivaut à "remplacer la ligne d'inclusion par les trois lignes du fichier `fonctions.php`"

Découpage de l'information

Grâce aux fonctions d'inclusion, il est facile de structurer, réutiliser et factoriser le code :

- En général, à chaque balise structurante correspond un fichier (e.g., un fichier `menu.php` pour la balise `<nav>`)
- Les parties du site communes/statiques (e.g., entête, menu) sont incluses par les autres fichiers
- Les parties avec le contenu spécifique d'une page (i.e., fonctionnalité) ont leur propre fichier

Dans la suite, description des différents structurations/découpages

Plan

Aperçu

Deux découpages pour organiser son code (sans l'architecture MVC), et trois structurations du code d'une fonctionnalité (selon son type)

- Découpage du code redondant en utilisant l'inclusion :
  - découpage "un fichier par fonctionnalité"
  - découpage "un fichier unique"

- Structuration du code d'une fonctionnalité (balise `<main>`) :
  - fonctionnalité statique
  - fonctionnalité dynamique sans interaction
  - fonctionnalité dynamique avec interaction

Découpage "un fichier par fonctionnalité"

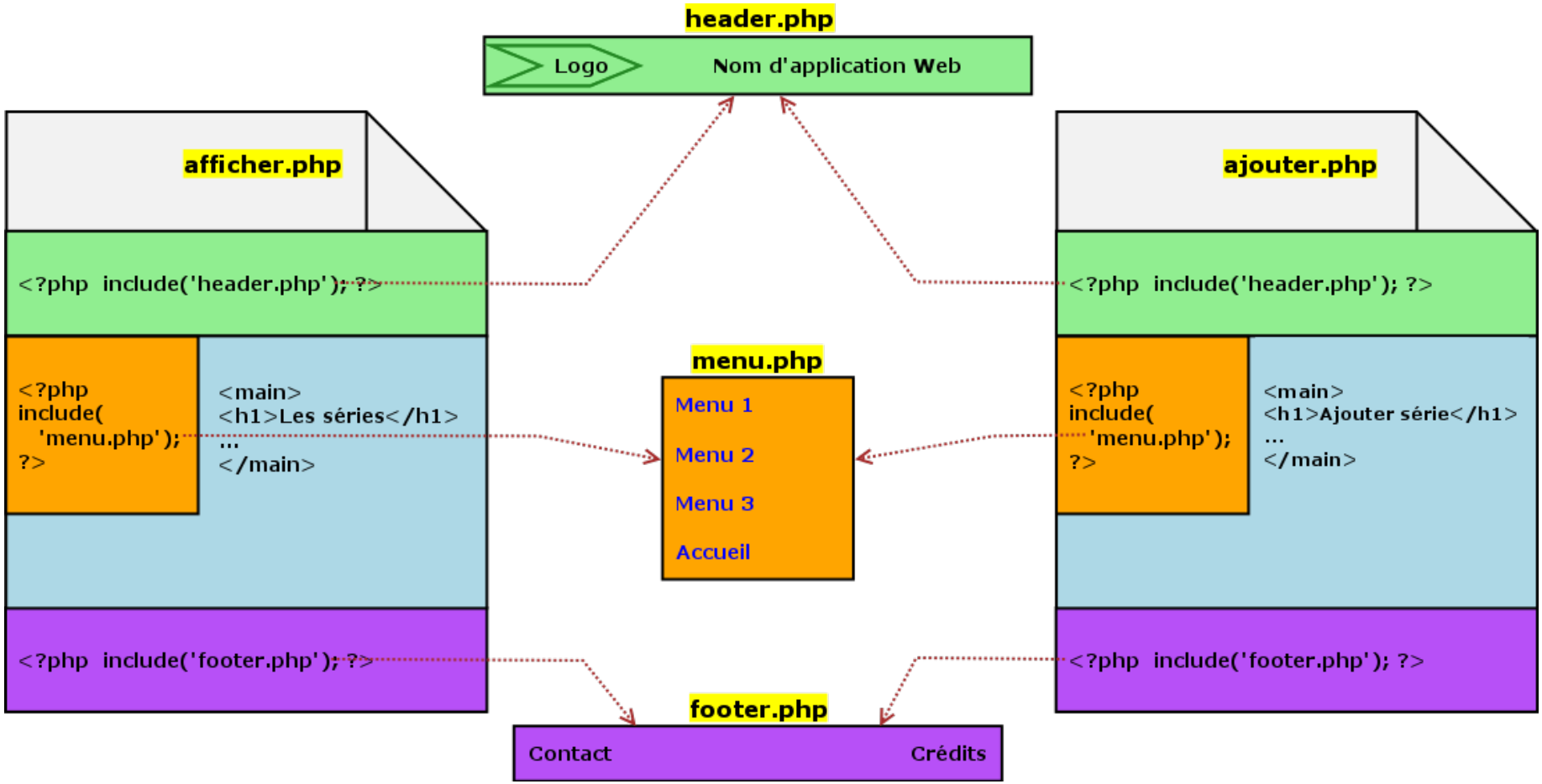
Avec cette solution, chaque fonctionnalité est stockée dans un fichier :

- Les parties communes/statiques du site sont incluses
- La partie contenu est spécifique au fichier (pas d'inclusion)

Inconvénients :

- Le design n'est pas factorisé (présent partiellement dans chaque fichier)
- Pour certaines modifications (e.g., des balises structurantes ou du nom d'un fichier d'inclusion), besoin de modifier chaque fichier de contenu

Découpage "un fichier par fonctionnalité" (2)

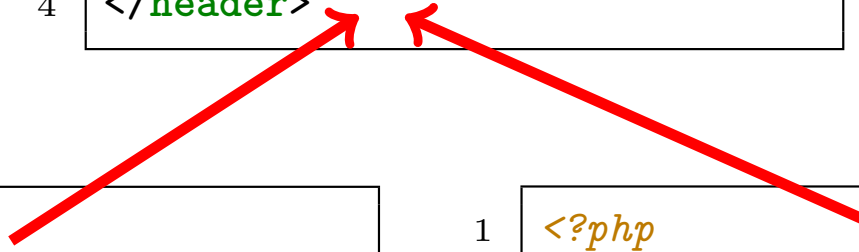


Découpage "une page par contenu spécifique" : les zones communes, stockées dans des fichiers externes, sont incluses par chaque page de contenu

Découpage "un fichier par fonctionnalité" (3)

header.php

```
1 <header>
2   <img src="logo.jpg" />
3   <h1>Serial critique</h1>
4 </header>
```



afficher.php

```
1 <?php
2   include('header.php');
3 ?>
4
5 <main>
6   <h1>Liste des séries</h1>
7   ...
8 </main>
```

ajouter.php

```
1 <?php
2   include('header.php');
3 ?>
4
5 <main>
6   <h1>Ajouter une série</h1>
7   ...
8 </main>
```

Code factorisé : le code du `<header>` n'apparait qu'à un seul endroit, et il est inclus par les fichiers de fonctionnalité

Découpage "un fichier unique"

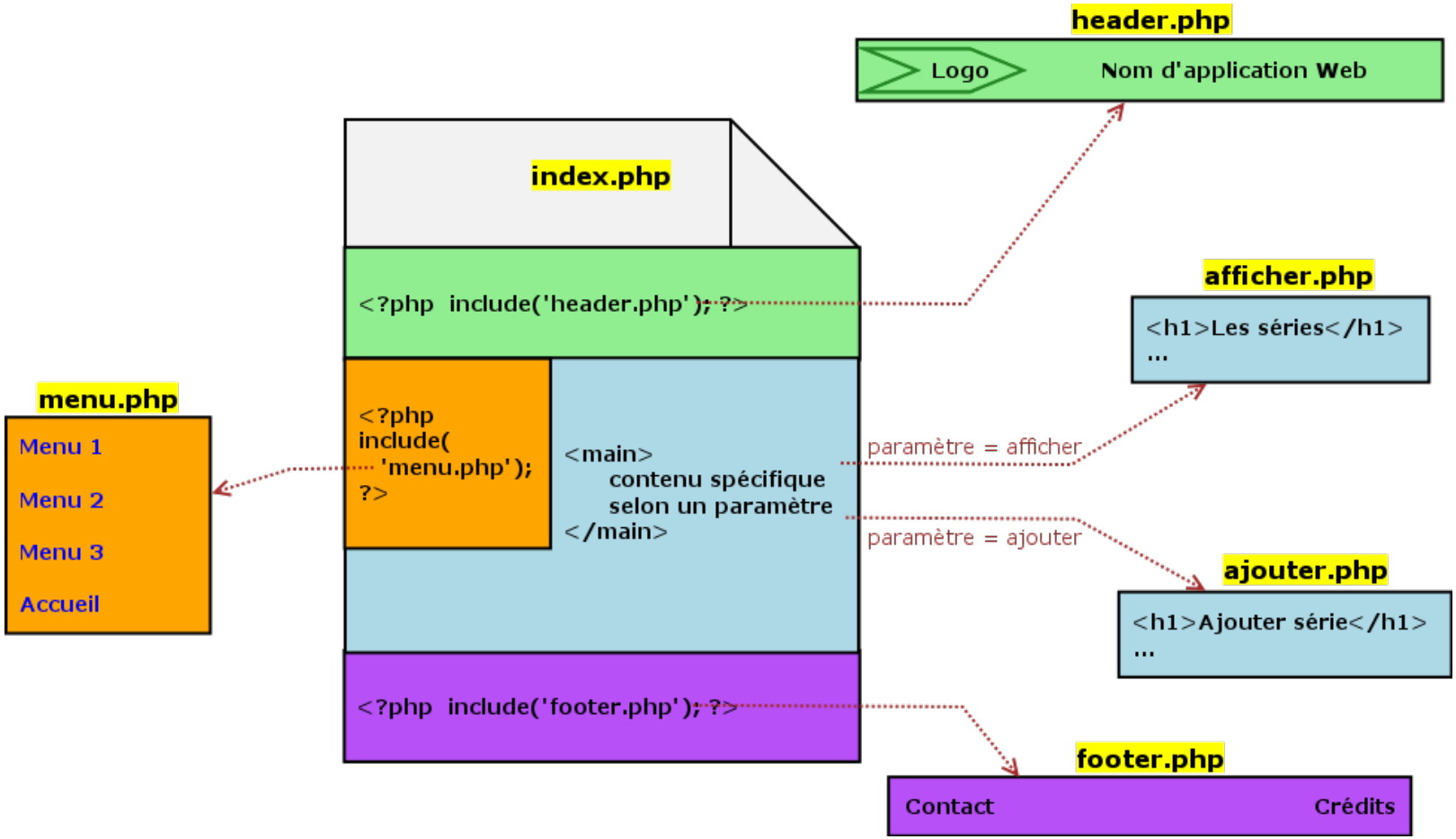
Avec cette solution, un seul fichier qui fait de multiples inclusions :

- Généralement le fichier `index.php`
- Les parties communes/statiques sont incluses
- Les parties de contenu sont dans des fichiers séparés, et incluses par `index.php` selon la valeur d'un paramètre

Inconvénients :

- URL parfois non explicite (e.g., `index.php?action=1`)
- Besoin de transmettre la page de contenu (e.g., en paramètre, session) et de vérifier sa valeur

Découpage "un fichier unique" (2)



Découpage "un fichier unique" : un seul fichier, qui charge les zones communes à partir de fichiers externes et un contenu spécifique selon la valeur d'un paramètre

Découpage "un fichier unique" (3)

header.php

```
1 <header>
2   <img src="logo.jpg" />
3   <h1>Serial critique</h1>
4 </header>
```

afficher.php

```
<h1>Liste des séries</h1>
...
```

```
1 <?php
2   include('header.php');
3 ?>
4 <main>
5   <?php
6   // vérifier $_GET['page']
7   include($_GET['page']);
8   ?>
9 </main>
```

index.php

si page = afficher

si page = ajouter

```
<h1>Ajouter une série</h1>
...
```

ajouter.php

Découpage "un fichier unique" - routage

Comment sélectionner la page de contenu ?

- La valeur d'un paramètre (e.g., via l'URL) indique quel contenu charger : abstrait (e.g., page=1) ou concret (e.g., `action=afficher`)

Solution 1 : la valeur du paramètre = nom de fichier

- Vérifications nécessaires si le fichier existe sur le disque, etc.
- Problèmes de sécurité !

index.php

```php
$page = 'static/accueil.php'; // page par défaut
if(isset($_GET['page'])) {  // vérification si paramètre
  $nomPage = $_GET['page'];
  if(file_exists('php/' . $nomPage)) // vérification si fichier
    $page = 'php/' . $nomPage;
}
include($page);  // inclut la page demandée ou accueil
```

Découpage "un fichier unique" - routage (2)

Solution 2 : la valeur du paramètre = nom de route

- Définition d'un tableau (associatif) de routes autorisées
- Vérification si la valeur du paramètre existe dans le tableau

routes.php

```php
$routes = array(
  'afficher' =>
    'php/affiche.php',
  'ajouter' =>
    'php/ajout.php',
  'rechercher' =>
    'php/recherche.php',
  'historique' =>
    'php/historique.php'
);
```

index.php

```php
include('routes.php');
// ...
$page = 'static/accueil.php';  // page par défaut
if(isset($_GET['page'])) {  // vérification si paramètre
  $nomPage = $_GET['page'];
  if(isset($routes[$nomPage])) { // vérification si
    route
    $page = $routes[$nomPage];
  }
}
include($page);  // inclut la page demandée ou accueil
```

Structuration des fonctionnalités

Une fonctionnalité = contenu de la balise `<main>` (en général)

Comment structurer le code spécifique à chaque page (i.e., chaque fonctionnalité, par exemple consulter une série, rechercher une série) ?

Différents types de pages :

- Non dynamique (e.g., que du code HTML)
- Dynamique, construite à la volée (e.g., qui interroge la BD) :
  - sans interaction (e.g., afficher les séries)
  - avec interaction (e.g., rechercher une série), besoin de gérer l'affichage avant et après soumission du formulaire

Exemple de page non dynamique

Code de la page d'accueil :

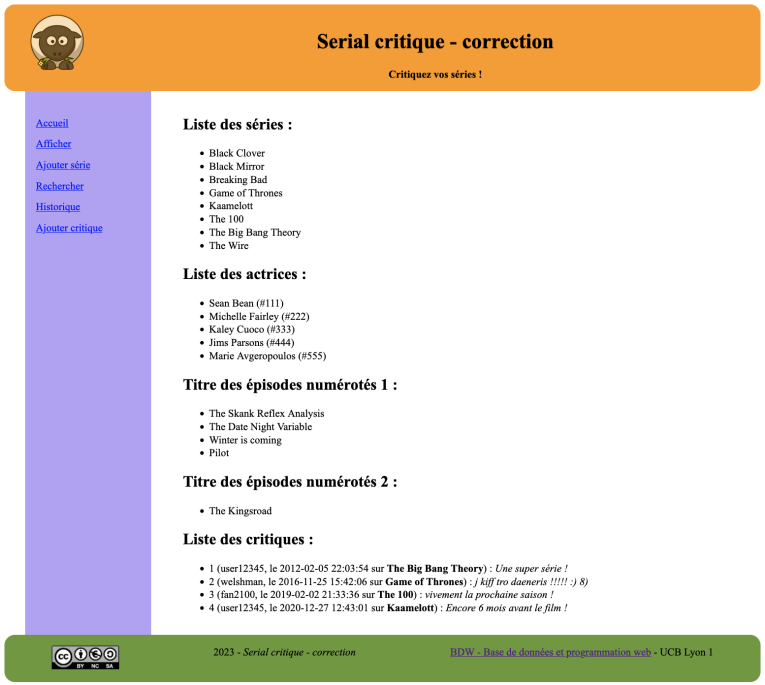
Exemple de page non dynamique

Code de la page d'accueil :


```
<h2>Le site de critique sur les séries pour le
↪   cours de BDW</h2>

<p id="paragDescription">
  Vous pouvez gérer des séries, des épisodes, et
  ↪   les acteur.ice.s.
  <br>
  Utilisez le menu à gauche pour découvrir les
  ↪   fonctionnalités de ce site.
</p>

<p>Que faire sur ce site ?</p>
<ul>
  <li>Afficher les données de la base</li>
  <li>Ajouter une série</li>
  <li>Rechercher une série ou une actrice</li>
  <li>Générer un historique de votre activité
  ↪   sur le site</li>
  <li>...</li>
</ul>

<div>
  <img src="img/top-series.jpg" />
</div>

...
```

Exemple de page dynamique sans interaction

Code pour lister les séries ("pseudo PHP") :



Serial critique - correction

Critiquez vos séries !

Accueil
Afficher
Ajouter série
Rechercher
Historique
Ajouter critique

Liste des séries :

- Black Clover
- Black Mirror
- Breaking Bad
- Game of Thrones
- Kaamelott
- The 100
- The Big Bang Theory
- The Wire

Liste des actrices :

- Sean Bean (#111)
- Michelle Fairley (#222)
- Kaley Cuoco (#333)
- Jims Parsons (#444)
- Marie Avgeropoulos (#555)

Titre des épisodes numérotés 1 :

- The Skank Reflex Analysis
- The Date Night Variable
- Winter is coming
- Pilot

Titre des épisodes numérotés 2 :

- The Kingsroad

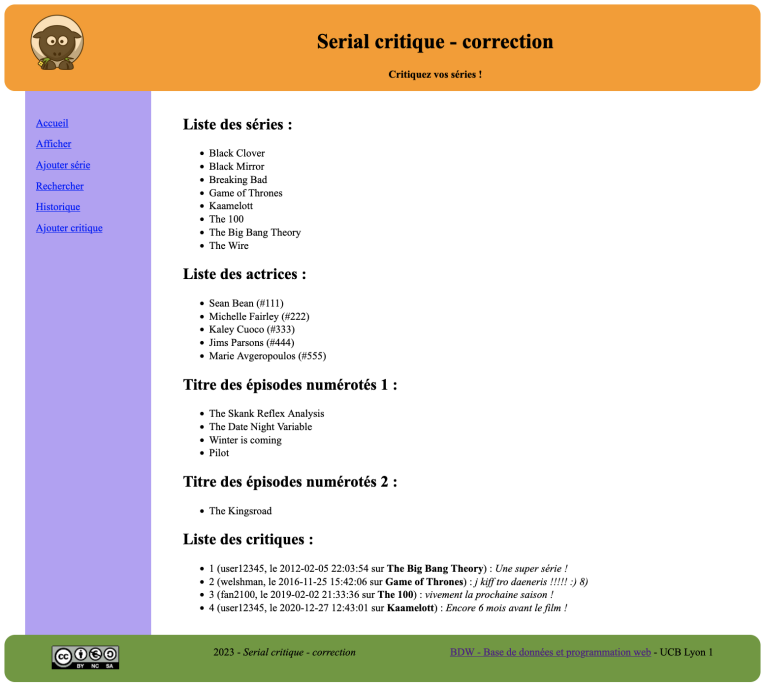
Liste des critiques :

- 1 (user12345, le 2012-02-05 22:03:54 sur **The Big Bang Theory**) : *Une super série !*
- 2 (welshman, le 2016-11-25 15:42:06 sur **Game of Thrones**) : *j kiff tro daeneris !!!!! :) 8)*
- 3 (fan2100, le 2019-02-02 21:33:36 sur **The 100**) : *vivement la prochaine saison !*
- 4 (user12345, le 2020-12-27 12:43:01 sur **Kaamelott**) : *Encore 6 mois avant le film !*

2023 - *Serial critique - correction* BDW - Base de données et programmation web - UCB Lyon 1

Exemple de page dynamique sans interaction

Code pour lister les séries ("pseudo PHP") :

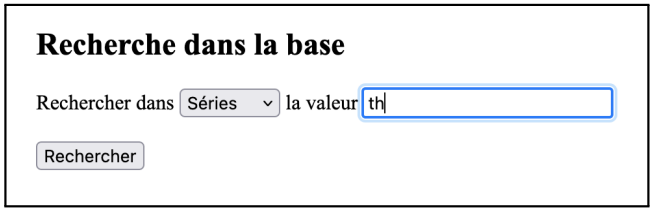



```php
<h1>Affichage des informations de la base</h1>
<h2>Liste des séries :</h2>
<?php
$connexion = connecter();
$requete = "SELECT nomSerie FROM Series;";
$resultat = executer_requete($connexion,
    $requete);

if($resultat == FALSE) {
  printf("<p>Aucune série.</p>");
}
else {
  echo '<ul>';
  while ($instance = lire_instance($resultat)) {
    echo '<li>' . $instance['nomSerie'] .
        '</li>';
  }
  echo '</ul>';
}
// idem pour autres affichages
```

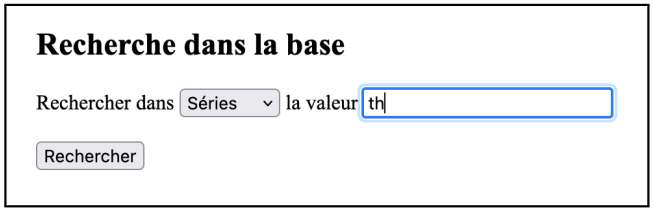
Exemple de page dynamique avec interaction (1)

Code pour rechercher des séries ("pseudo PHP", version simplifiée)

Exemple de page dynamique avec interaction (2)

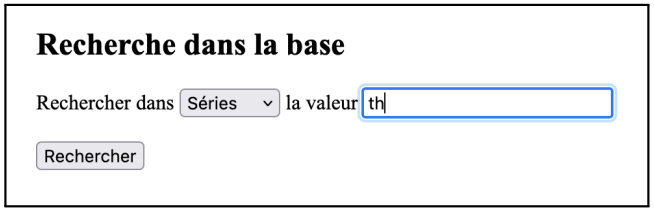


Exemple de page dynamique avec interaction (2)

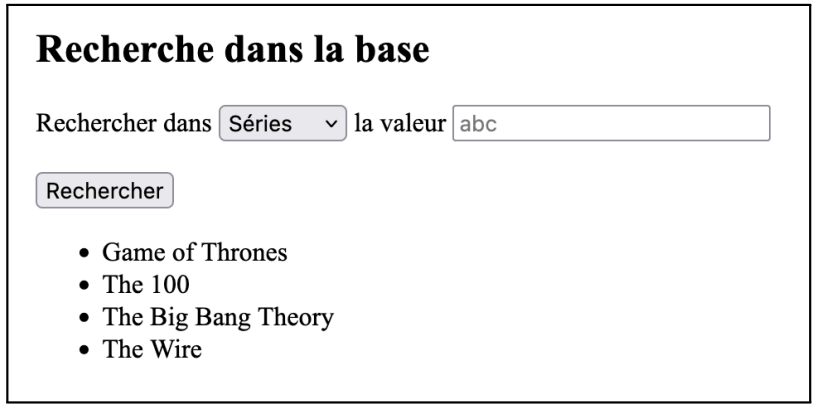



```
1  <h1>Recherche dans la base</h1>
2  <form method="post" action="#">
3    <p>Rechercher dans les
     ↪   séries</p>
4    <label for="idValeur">la
     ↪   valeur</label>
5    <input type="text"
     ↪   name="valeur"
     ↪   id="idValeur" />
6    <input type="submit"
     ↪   name="bouton"
     ↪   value="Rechercher">
7  </form>
```

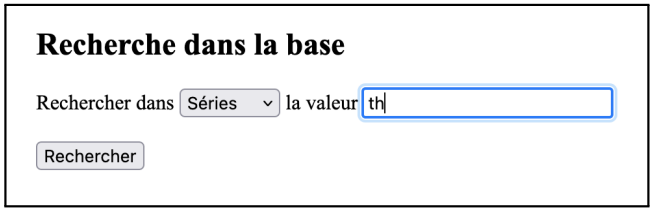
Exemple de page dynamique avec interaction (2)



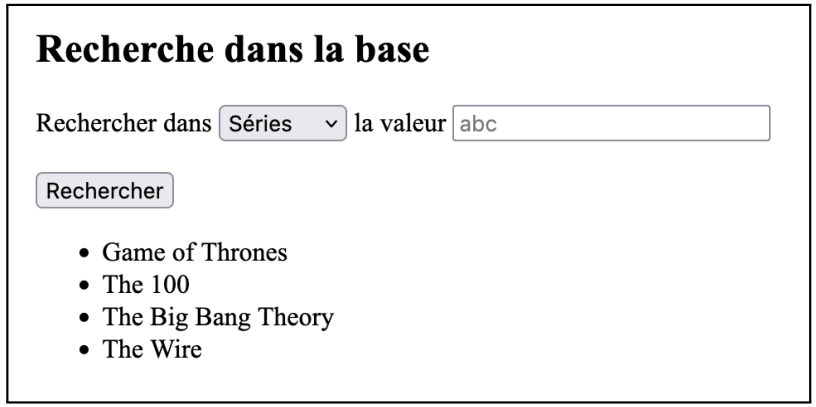

```
1  <h1>Recherche dans la base</h1>
2  <form method="post" action="#">
3    <p>Rechercher dans les
     ↪   séries</p>
4    <label for="idValeur">la
     ↪   valeur</label>
5    <input type="text"
     ↪   name="valeur"
     ↪   id="idValeur" />
6    <input type="submit"
     ↪   name="bouton"
     ↪   value="Rechercher">
7  </form>
```



Exemple de page dynamique avec interaction (2)




```
1  <h1>Recherche dans la base</h1>
2  <form method="post" action="#">
3    <p>Rechercher dans les
     ↪ séries</p>
4    <label for="idValeur">la
     ↪ valeur</label>
5    <input type="text"
     ↪ name="valeur"
     ↪ id="idValeur" />
6    <input type="submit"
     ↪ name="bouton"
     ↪ value="Rechercher">
7  </form>
```



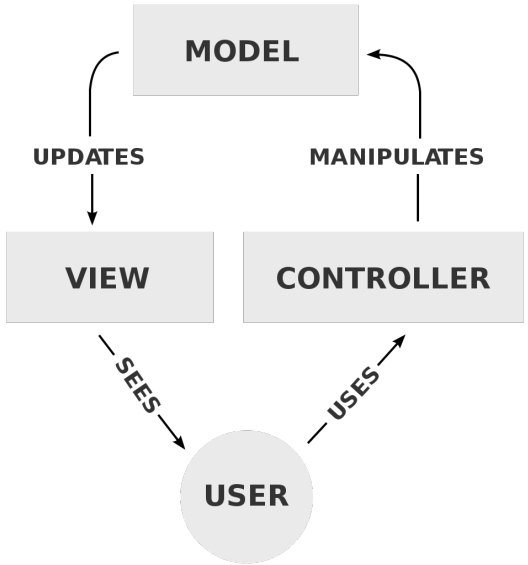

```
1  if(formulaire soumis) {
2    $connexion = connecter();
3    $valeur = recuperer_valeur_soumise();
4    $requete = "SELECT * FROM Series WHERE nomSerie
     ↪ LIKE '%$valeur%';";
5    $resultat = executer_requete($connexion,
     ↪ $requete);
6
7    if($resultat == FALSE)
8      printf("<p>Aucun résultat.</p>");
9    else {
10     echo '<ul>';
11     while ($instance = lire_instance($resultat))
12       echo '<li>' . $instance['nomSerie'] .
         ↪ '</li>';
13     echo '</ul>';
14   }
15 }
```

Plan

Introduction à MVC

Architecture Modèle Vue Contrôleur (MVC) :

- Design pattern (origine en 1978)
- Utilisé par de nombreux sites web et frameworks
- Structuration et réutilisation du code, développement en parallèle, couplage faible
- Mais plus complexe et "fonctionnalités éparpillées"



---

http://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur

http://openclassrooms.com/fr/courses/4670706-adoptez-une-architecture-mvc-en-php/

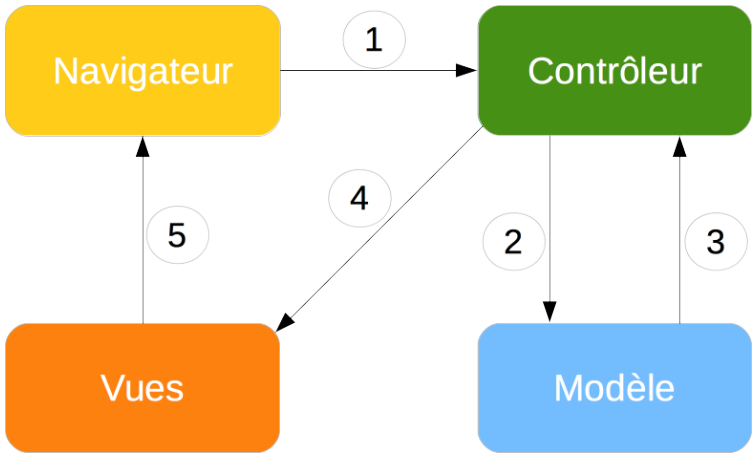
http://bpesquet.developpez.com/tutoriels/php/evoluer-architecture-mvc/

http://openclassrooms.com/courses/evoluez-vers-une-architecture-php-professionnelle

Rôle des composants

- Modèle (structure dynamique des données) :
  - en lien avec la base de données
  - opérations de lecture, mise à jour, ajout, etc.
  - indépendant de la vue et du contrôleur
- Vue (interface graphique) :
  - éléments visuels et logique d'affichage des données
  - dépend du modèle (e.g., données à afficher)
- Contrôleur :
  - lien entre modèle et vue
  - traite les entrées (paramètres), choisit la vue
  - dépend de la vue et du modèle

Workflow entre composants MVC

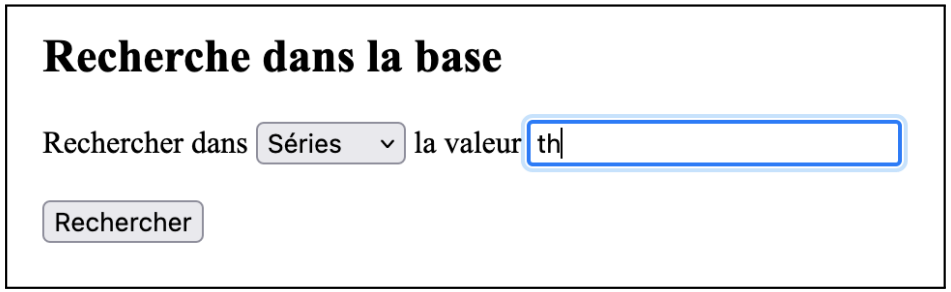


1. *L'internaute envoie une requête (paramètres optionnels)*
2. *Le contrôleur vérifie les paramètres, et appelle le modèle*
3. *Le modèle interroge la base de données et retourne un résultat au contrôleur (e.g., des tuples, un booléen)*
4. *Le contrôleur sélectionne la vue à afficher et lui passe le résultat*
5. *La vue (code HTML) est envoyée au navigateur de l'internaute*

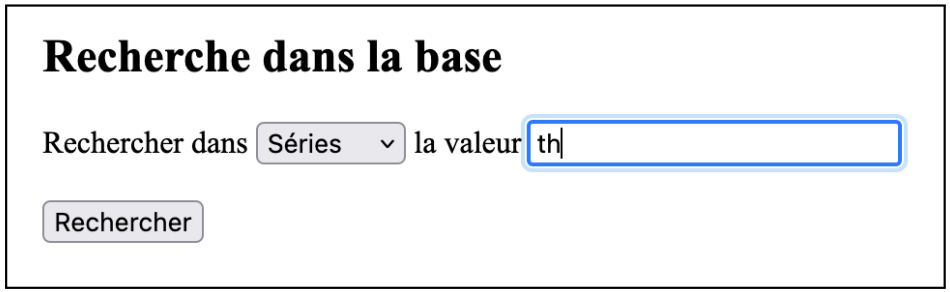
---

http://www.supinfo.com/articles/single/1625-mvc-presentation-patron-conception

Exemple - liste des séries contenant "th"



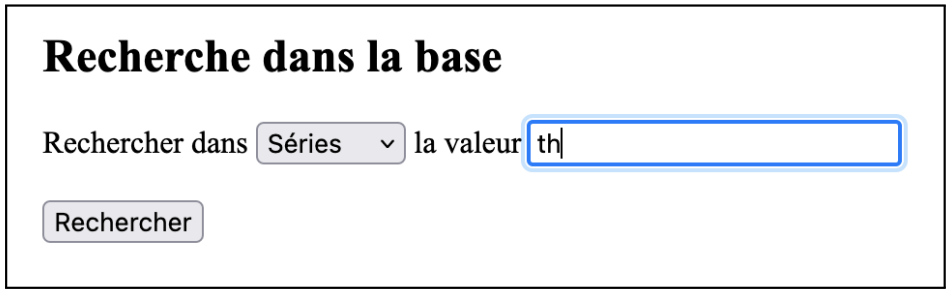
Exemple - liste des séries contenant "th"



controleurRecherche.php

```php
include("modele.php");
if(isset($_POST['rechercher'])) {
  $keyword = $_POST['keyword'];
  $series = getSeries($connexion, $keyword);
  include("vue.php");
}
```

Exemple - liste des séries contenant "th"



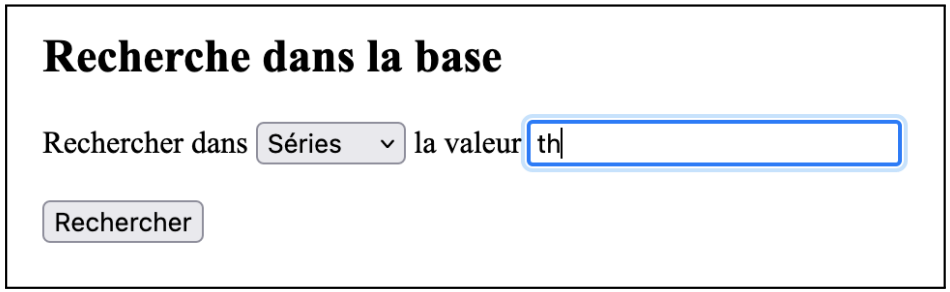
controleurRecherche.php

```php
include("modele.php");
if(isset($_POST['rechercher'])) {
  $keyword = $_POST['keyword'];
  $series = getSeries($connexion, $keyword);
  include("vue.php");
}
```

modele.php

```php
function getSeries($connexion, $keyword) {
  $clean_keyword =
  ↪ mysqli_real_escape_string($connexion,
  ↪ $keyword);
  $requete = "SELECT DISTINCT nomSerie
  ↪ FROM Series WHERE nom LIKE
  ↪ '%$clean_keyword%'";
  $resultat = mysqli_query($connexion,
  ↪ $requete);
  $series = mysqli_fetch_all($resultat,
  ↪ MYSQLI_ASSOC);
  return $series; // tableau associatif
}
```

Exemple - liste des séries contenant "th"



controleurRecherche.php

```php
include("modele.php");
if(isset($_POST['rechercher'])) {
  $keyword = $_POST['keyword'];
  $series = getSeries($connexion, $keyword);
  include("vue.php");
}
```

modele.php

```php
function getSeries($connexion, $keyword) {
  $clean_keyword =
  ↪ mysqli_real_escape_string($connexion,
  ↪ $keyword);
  $requete = "SELECT DISTINCT nomSerie
  ↪ FROM Series WHERE nom LIKE
  ↪ '%$clean_keyword%'";
  $resultat = mysqli_query($connexion,
  ↪ $requete);
  $series = mysqli_fetch_all($resultat,
  ↪ MYSQLI_ASSOC);
  return $series; // tableau associatif
}
```

vueRecherche.php

```php
<ul>
<?php foreach($series as $serie) { ?>
  <li><?= $serie['nomSerie'] ?></li>
<?php } ?>
</ul>
```

Exemple - liste des séries contenant "th"

Recherche dans la base

Rechercher dans Séries la valeur th

Rechercher

controleurRecherche.php

```php
include("modele.php");
if(isset($_POST['rechercher'])) {
  $keyword = $_POST['keyword'];
  $series = getSeries($connexion, $keyword);
  include("vue.php");
}
```

- Game of Thrones
- The 100
- The Big Bang Theory
- The Wire

modele.php

```php
function getSeries($connexion, $keyword) {
  $clean_keyword =
    ↪ mysqli_real_escape_string($connexion,
    ↪ $keyword);
  $requete = "SELECT DISTINCT nomSerie
    ↪ FROM Series WHERE nom LIKE
    ↪ '%$clean_keyword%'";
  $resultat = mysqli_query($connexion,
    ↪ $requete);
  $series = mysqli_fetch_all($resultat,
    ↪ MYSQLI_ASSOC);
  return $series; // tableau associatif
}
```

vueRecherche.php

```php
<ul>
<?php foreach($series as $serie) { ?>
  <li><?= $serie['nomSerie'] ?></li>
<?php } ?>
</ul>
```

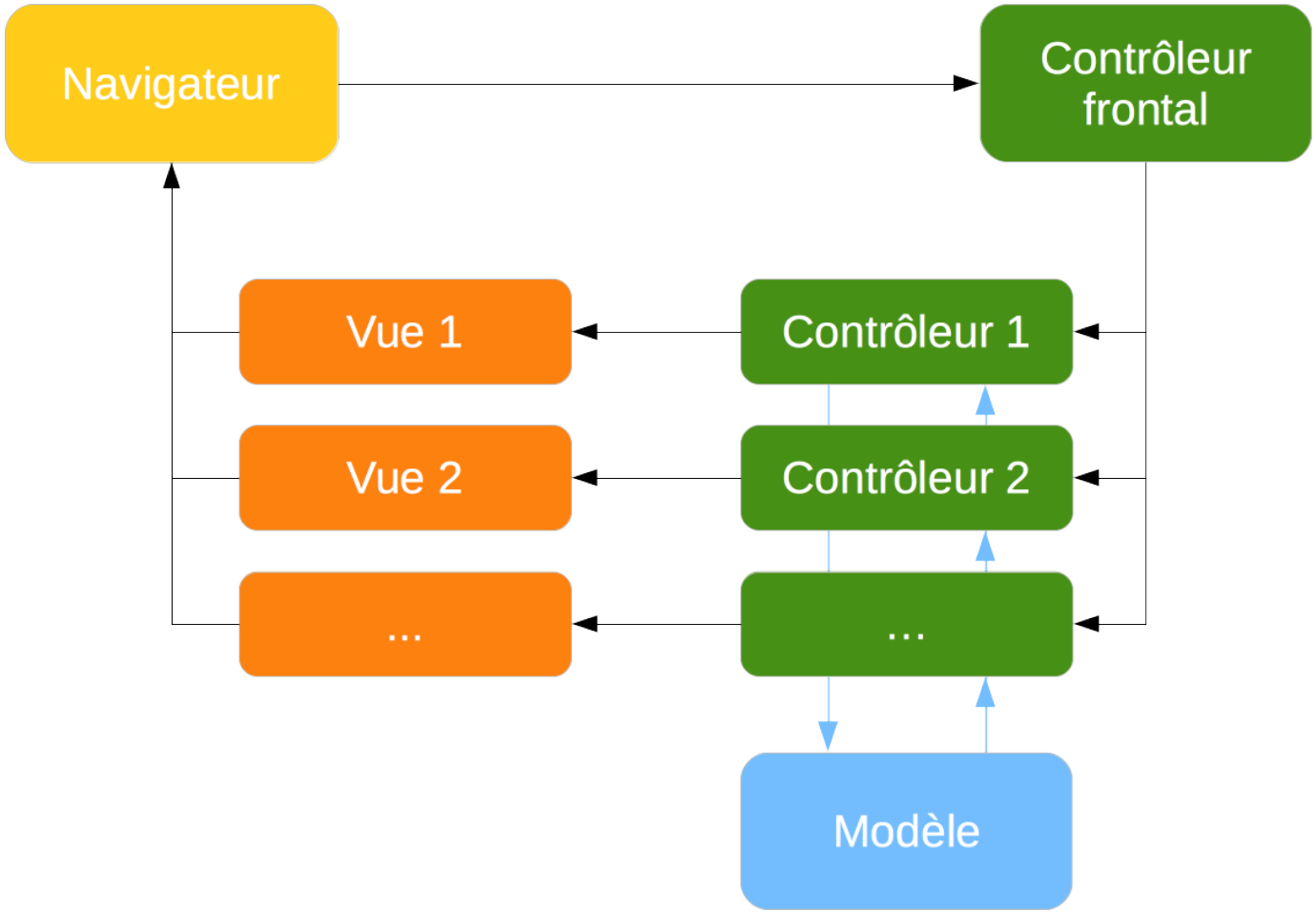
Contrôleur frontal

En général, pour chaque page ou fonctionnalité : un contrôleur et une vue

Lors de la réception d'une requête, besoin de diriger la requête vers le bon contrôleur $\Rightarrow$ rôle du contrôleur frontal :

- Point d'entrée unique du site (souvent `index.php`)
- Besoin d'un paramètre pour spécifier la fonctionnalité demandée (e.g., `?action=` ou `?page=`) associée à une table de routage
- Vérification des paramètres fournis (e.g., un ID d'actrice valide)

Contrôleur frontal - workflow



En réalité, un contrôleur peut effectuer différentes actions (e.g., afficher ou modifier une série). Chaque action peut ensuite avoir sa propre vue.

Contrôleur frontal - exemple

```
1  $controleur = 'controleurAccueil'; // contrôleur par défaut
2  $vue = 'vueAccueil';  // vue par défaut
3
4  if(isset($_GET['page'])) { // si paramètre page
5    $nomPage = $_GET['page'];
6    if(isset($routes[$nomPage])) {
7      $controleur = $routes[$nomPage]['controleur'];
8      $vue = $routes[$nomPage]['vue'];
9    }
10 }
11
12 include('controleurs/' . $controleur . '.php');
13 include('vues/' . $vue . '.php');
```

Dans ce contrôleur frontal, le contrôleur et la vue par défaut sont définies (lignes 1 et 2). Si la fonctionnalité demandée via le paramètre `page` existe dans le fichier de routes, on utilise son contrôleur et sa vue (lignes 4 à 10). Enfin, appel au contrôleur et à la vue (lignes 12 et 13).

Les templates

Un *template* (gabarit) représente la manière avec laquelle sont agencés et stylisés les éléments d'une page.

Pourquoi avoir plusieurs templates ?

- Comparaison des templates pendant la conception
- Choix pour les internautes, réutilisation pour d'autres projets
- Accessibilité !

Un template comprend la structure HTML et le CSS, mais utilise des variables pour les données spécifiques à une page (e.g., titre, contenu). Ces variables sont renseignées par les vues.

Les templates - exemple

```
1  <head>
2      <meta charset="utf-8" />
3      <title><?= $nomSite ?></title>
4      <link href="css/style2.css">
5  </head>
6  <body class="template2">
7    <?php
     ↪ include('templates/header2.php');
     ↪ ?>
8    <main>
9      <?= $titre ?>
10     <?= $contenu; ?>
11   </main>
12   <?php
     ↪ include('templates/footer.php');
     ↪ ?>
13 </body>
```

Exemple de template. Le style CSS peut être spécifique au template (ligne 4), et les variables (`$titre` et `$contenu`) définies par les vues sont utilisées (lignes 9 et 10).

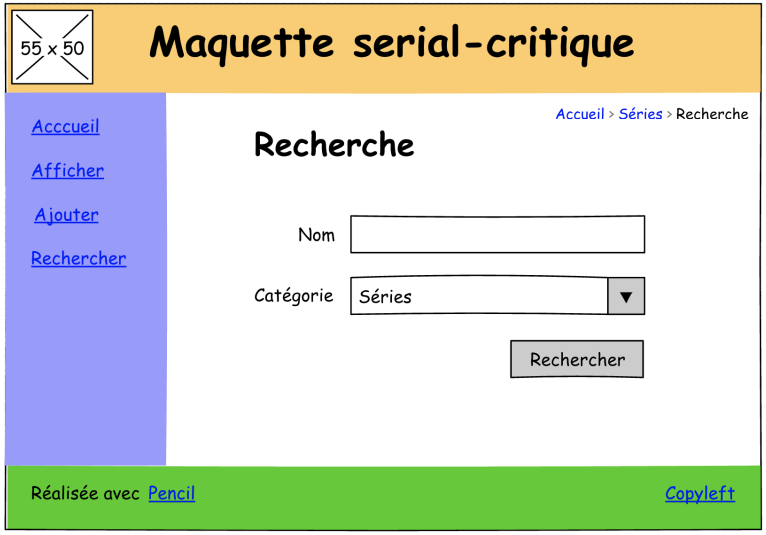
```
1  <?php $titre = "Liste des séries"; ?>
2
3  <?php ob_start(); ?>
4    <ul>
5      <?php foreach($series as $serie) { ?>
6        <li><?= $serie['nomSerie'] ?></li>
7      <?php } ?>
8    </ul>
9  <?php $contenu = ob_get_clean(); ?>
10
11 <?php require 'templates/' .
   ↪ $currentTemplate; ?>
```

Exemple de vue utilisant un template. La vue ne fait qu'instancier les variables du template : `$titre` à la ligne 1 et `$contenu`, qui nécessite de bufferiser les données avec les fonctions `ob_start()` et `ob_get_clean()` (lignes 3 à 9). Le template courant est chargé ligne 11.

Plan

Structuration du site

Réflexion sur le design des pages (maquettes)



Choix de la structuration :

- Architecture MVC : *design pattern* donc bonne pratique
- Utilisation du routage conseillée
- Utilisation des templates optionnelle

---

http://pencil.evolus.vn/
http://draw.io/

Gestion de projet

- Dans le binôme, certaines tâches sont faites ensemble (modélisation, structuration du site) et d'autres en parallèle (programmation des différentes fonctionnalités)
  - bien réfléchir à la structuration pour faciliter la répartition
  - ne pas perdre trop de temps sur le design !
- Utilisation de la forge (http://forge.univ-lyon1.fr/) :
  - code partagé, travail collaboratif, versioning
  - travail sauvegardé (plus d'excuse de clé USB perdue)
  - apprentissage de Git
  - clients graphiques GitKraken, Sourcetree, GitExtensions, etc.

---

http://fr.wikipedia.org/wiki/Conception_de_site_web

http://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql

http://openclassrooms.com/courses/gerez-vos-codes-source-avec-git

https://rogerdudler.github.io/git-guide/

Bilan

Structuration d'un site web :

- Organisation des fichiers (hiérarchie des répertoires)
- Deux découpages basiques : *"un fichier par fonctionnalité"* et *"un fichier unique"*
- Introduction à l'architecture MVC et aux templates

Un bilan de l'UE pour finir...

Calvin et Hobbes (Bill Watterson)