

Architecture MVC

Schémas par étapes

[Introduction](#)

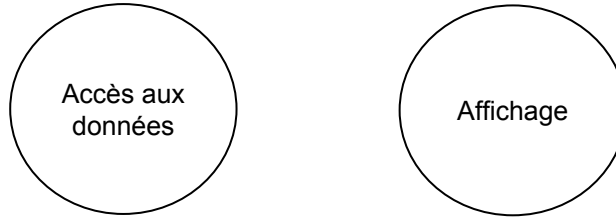
1. [Comprendre les logiques](#)
2. [Organisation et améliorations](#)
3. [Récapitulatifs](#)
4. [Bonnes pratiques](#)

Introduction

- A. [Historique](#)
- B. [Organisation générale](#)
- C. [But et organisation du cours](#)

A : Historique

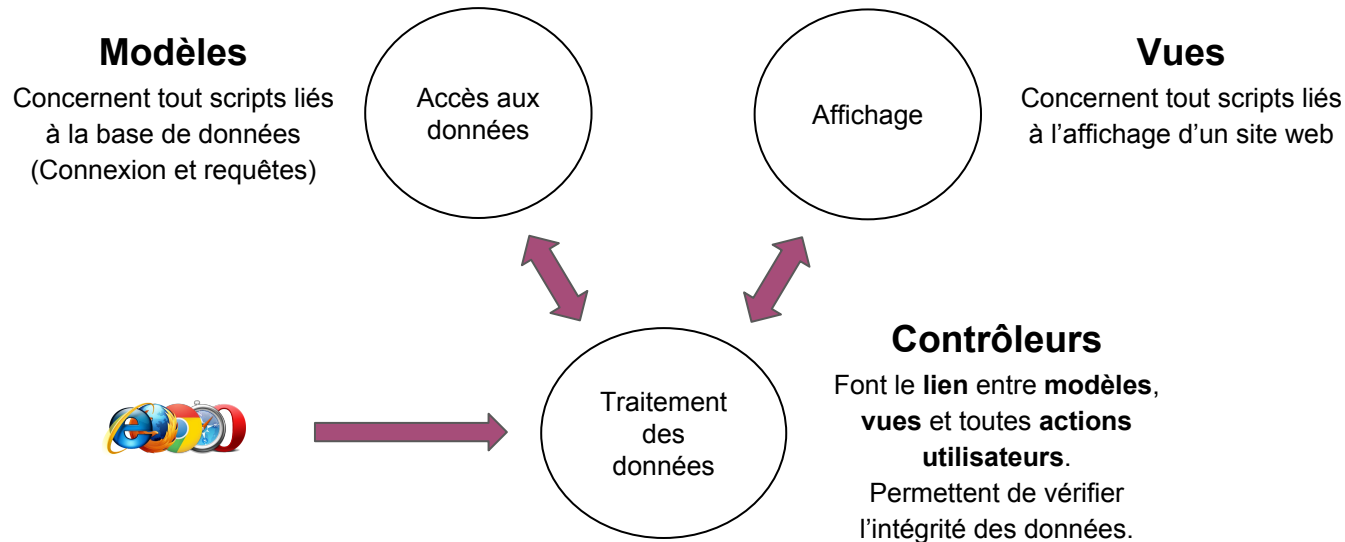
- L'architecture MVC a été mise au point en 1979 par un norvégien du nom de Trygve Reenskaug afin **d'organiser** au mieux un script comprenant **accès aux bases de donnée et affichage des données**.



- La première utilisation de cette **architecture ou patron de conception** (design pattern) fut pour le langage 'Smaltalk', spécialisé dans la mise en place d'interfaces graphiques de logiciels. Il a été ensuite réutilisé dans plusieurs applications nécessitant un accès à une base pour affichage.
- De part son utilité première, l'architecture MVC est, à l'heure actuelle, très largement utilisée dans le web pour les grosses applications de type dynamiques. Ce qui permet de travailler à plusieurs et d'optimiser la sécurité des pages.
- A la base créé pour du code procédural, sa normalisation en orienté objet arrive en 1995. Cette architecture peut donc être utilisée avec les deux modes de programmation.

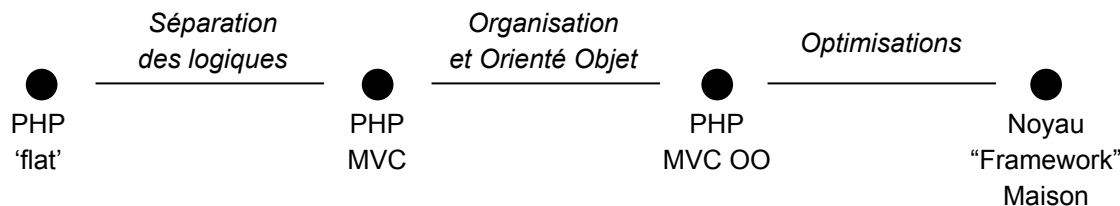
B : Organisation générale

- Dans le domaine des applications Web, l'architecture logicielle Modèle-Vue-Contrôleur (MVC) est celle qui est utilisée, à l'heure actuelle, par la plupart des « frameworks » connus pour le développement et se basant sur les préconisations de ce concept dont le paradigme consiste à insister sur la séparation des logiques selon l'accès, le traitement et l'affichage des données :



C : But et organisation du cours MVC

- Le but de ce cours est de se préparer et de comprendre l'utilisation du "framework" Symfony2 en explorant les aspects principaux d'une architecture Model-View-Controller.
- Ce cours sera sous la forme d'un long Travaux Pratique ou l'on programmera tout du long avec, au fur et à mesure, plusieurs indications pour avancer (schémas ci-après).
- La programmation (en PHP) débutera en 'flat'. C'est à dire uniquement en procédural. Puis, étapes par étapes, les codes seront séparés selon les logiques du concept MVC (accès, traitement et affichage des données).
Le tout sera ensuite organisé selon une arborescence type MVC avec l'ajout de nouvelles techniques de programmation puis l'utilisation de la méthode Orientée Objet.
Enfin, quelques optimisations supplémentaires seront mises en place afin d'obtenir "un noyau" de framework dis "Maison".



I. Comprendre les logiques

- A. [Préparer les fichiers](#) - [schéma](#)
- B. [Séparer les logiques](#) - [schéma 1](#) - [schéma 2](#)
- C. [Rationalisation de la structure](#) - [schéma 1](#) - [schéma 2](#) - [schéma 3](#)

A : Préparer les fichiers

- **Programmation “flat”**

→ *définition : de l'anglais : plat, monotone, sans intérêt.*

Par programmation « flat », on sous-entend programmer sans respecter d'architecture logicielle particulière.

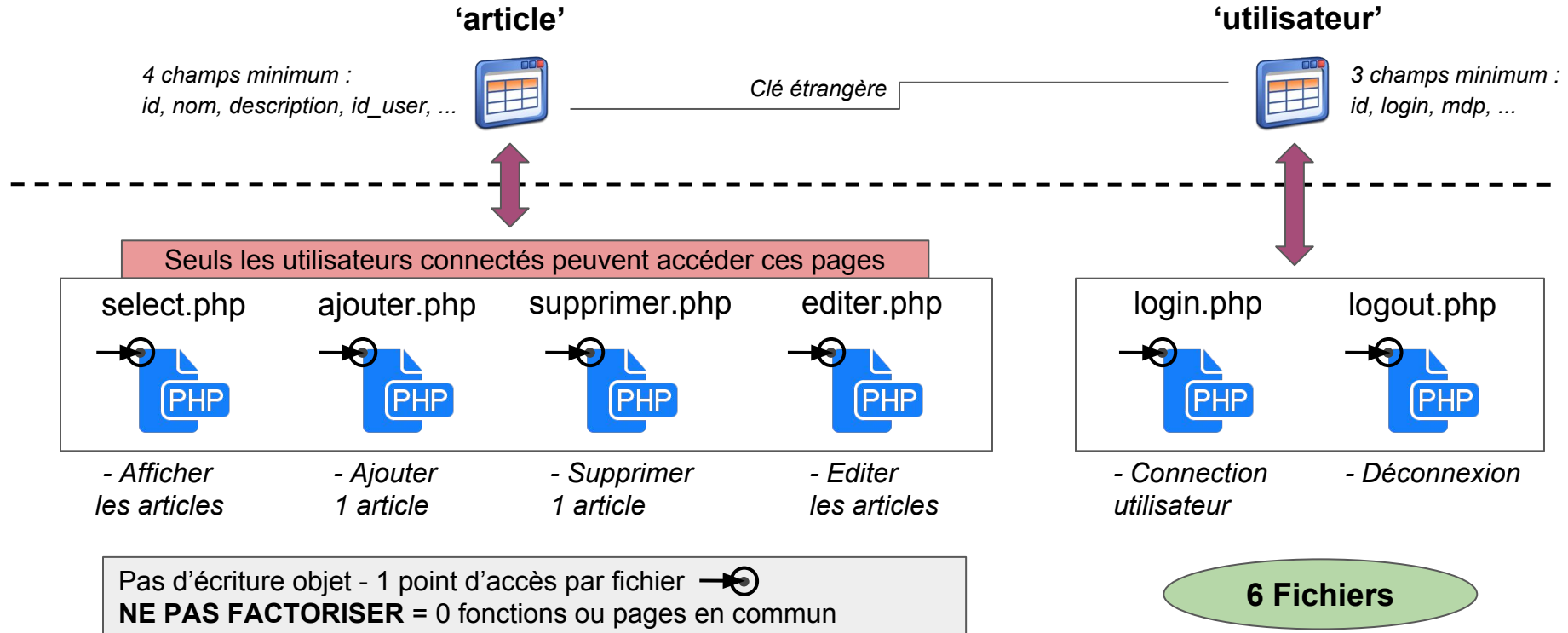
L'objectif de cette première partie est de montrer par quelle construction logique on peut arriver à partir d'un programme « flat » à arriver à une architecture de programme respectant les propositions de l'architecture Modèle-Vue-Contrôleur.

La mise en oeuvre de l'architecture MVC débouchera sur un ensemble de constructions logiques du programme qui mettront en évidence le rôle d'un « framework » MVC ainsi que les optimisations qu'il apporte.

→ [Cf. schéma A : mise en place des fichiers](#)

A : Mise en place des fichiers

- 1 BBD avec 2 tables + 6 fichiers pour affichage

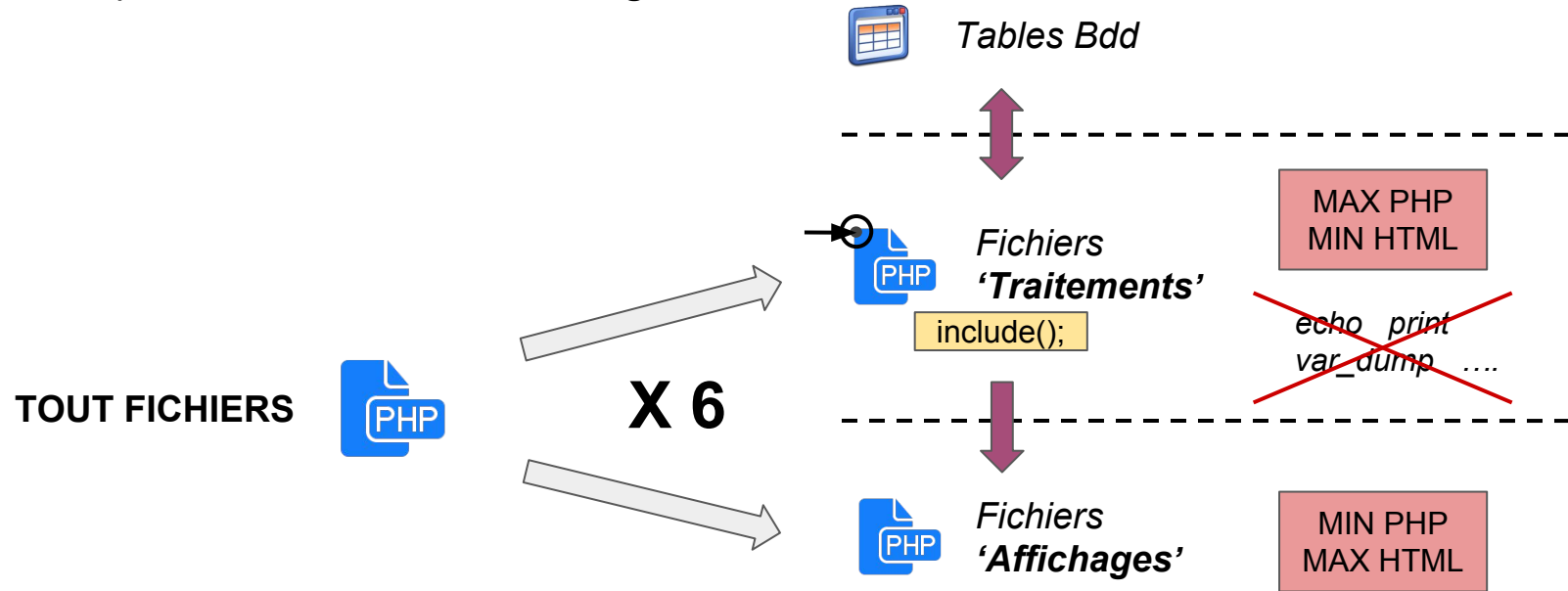



B : Séparer les logiques

- **Dans un premier temps**, nous allons découper notre programme “flat” en 2 fichiers :
 - *Le premier fichier sera consacré à l'accès aux données et aux traitements (on trouvera dans ce fichier l'essentiel du code SQL et PHP). C'est le nom de ce fichier qui sera le point d'entrée du programme (on devra saisir l'url de ce fichier pour démarrer le programme).*
 - *Le second fichier sera consacré à l'affichage (on y trouvera l'essentiel du code HTML).*
 - [Cf schéma B1 : partie 1.](#)
- **Dans un second temps**, nous allons découper en 2 fichiers le fichier consacré à l'accès aux données et aux traitements.
 - *Le premier fichier sera consacré à l'accès aux données*
 - *Le second fichier sera consacré aux traitements. Il sera le point d'entrée du programme.*
 - [Cf schéma B2 : partie 2.](#)

B : Séparer les logiques : partie 1

- Séparer 'Traitements' et 'Affichages'



Pas d'écriture objet - 1 point d'accès par **binôme** de fichier → 
NE PAS FACTORISER = 0 fonctions ou pages en commun

12 Fichiers
(6 + 6)

B : Séparer les logiques : partie 2

- Séparer '**Traitements SQL**' et '**Traitements Action**'
 - '**Traitements SQL**' sous forme de fonctions()
- Une fonction par accès à la Bdd

TOUT FICHIERS
'Traitements'



X 6



Tables Bdd



Fichiers
'**Traitements SQL**'

include;



Fichiers
'**Traitements Action**'

include;



Fichiers
'**Affichages**' ou '**VUES**'

1 $fx()$ par accès Bdd

Args : selon besoins
Retours : array|boolean

Pas d'écriture objet - 1 point d'accès par **groupe de 3 fichiers** →
NE PAS FACTORISER = 0 fonctions ou pages en commun

17 Fichiers
(5 + 6 + 6)

C : Rationalisation de la structure

- *Maintenant que nous avons découpé les programmes en séparant les logiques d'accès aux données, de traitement et d'affichage ; nous pouvons les regrouper en prenant comme point de référence notre source de données (traitement actions).*

*Le résultat de cette rationalisation est appelé architecture **Modèle-Vue-Contrôleur** abrégé en **MVC** :*

- Les **Modèles** sont responsables de l'accès aux données et sont généralement associés aux opérations effectuées sur **1 table** de la base de données.
- Les **Contrôleurs** contiennent des **actions**. Chaque action est responsable d'**1 traitement métier** et peut faire appel à **1 Modèle**.
- Les **Vues** sont responsables de l'affichage et sont généralement associées à **1 action** dans **1 Contrôleur**.

→ [Cf schémas C1 : Préparer les actions](#) et [C2 :regrouper les fonctions](#)

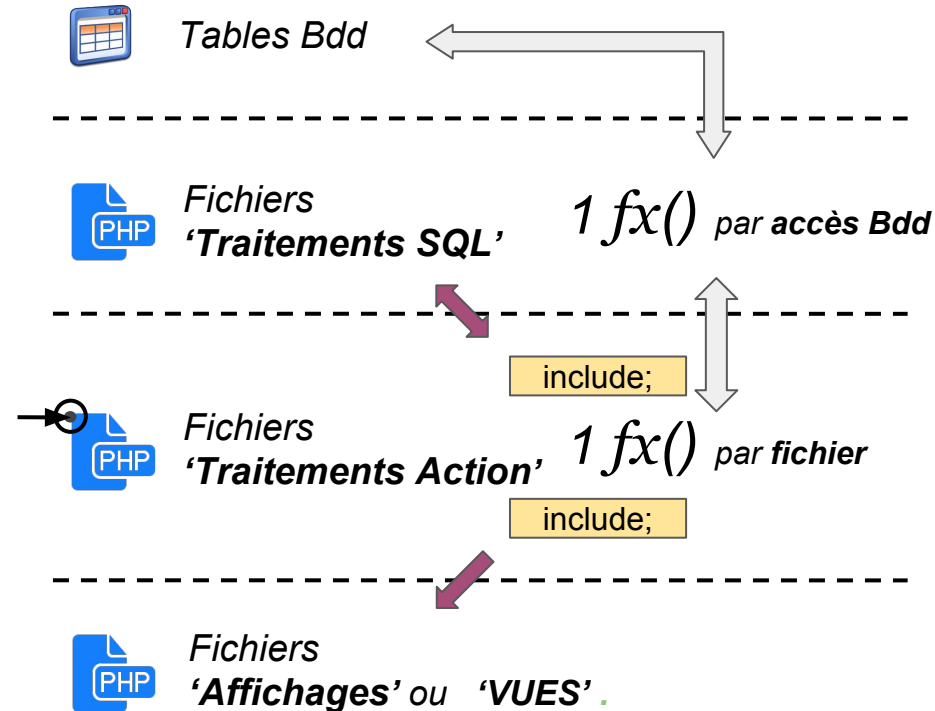
- *Dans le cadre du développement d'une application Web ou chaque action doit être déclenchée suivant une url particulière on se rend compte de la nécessité d'un mécanisme de **routing**. Un mécanisme de routage a pour but de déclencher une action dans un contrôleur en fonction d'une requête HTTP (url) particulière. Ce mécanisme de routage prend la forme d'un programme particulier qu'on appelle **routeur**. Le routeur est le **point d'entrée** de l'application.*


→ [Cf. schéma C2 : Fichier de routes](#)

C1 : Préparer les actions

- **'Traitements action'** sous forme de fonctions()
 → Une seule fonction() par fichier avec normalisation des noms (exemple : actionLogin(), actionAjouter())
 → Les fichiers de **'traitements SQL'** et de **'vues'** sont **directement inclus dans les fonctions**

Pas de gestion des valeurs

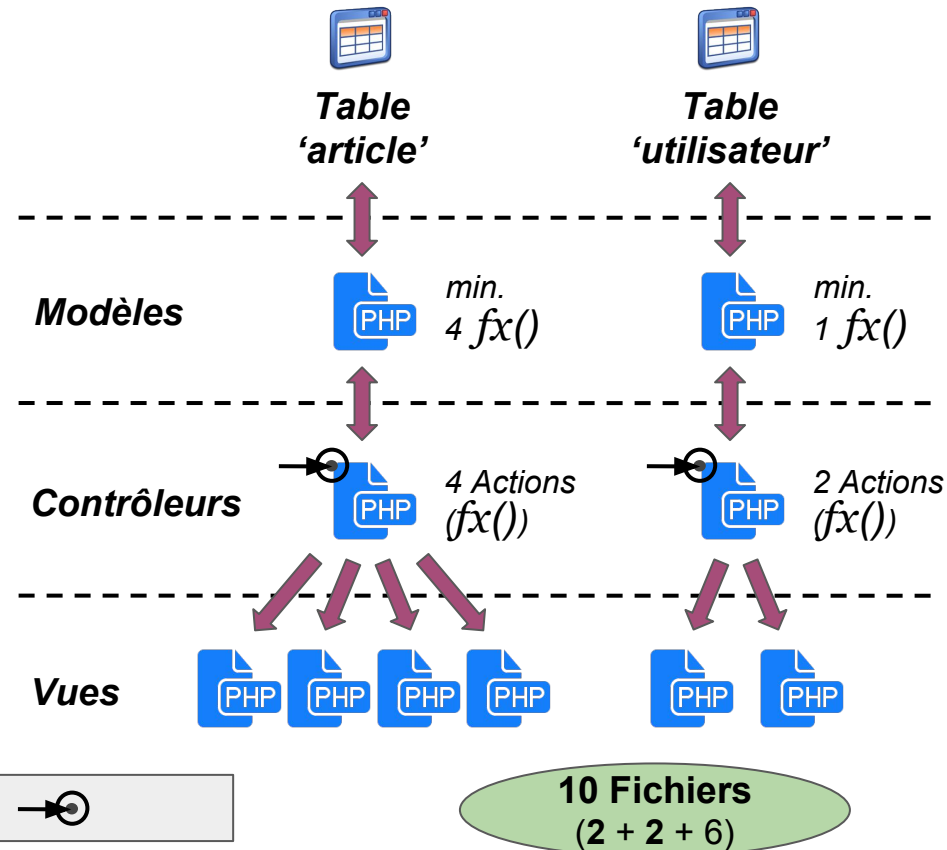


Pas d'écriture objet - 1 point d'accès par **groupe de 3 fichiers** → 
NE PAS FACTORISER = 0 fonctions ou pages en commun

17 Fichiers
 (5 + 6 + 6)

C2 : Regrouper les fonctions

- **‘Traitements SQL’ ou ‘Modèles’ :**
→ Les fonctions liées à une même table en Bdd sont regroupées dans **UN** même fichier (*modele_article, modele_utilisateur*)
- **‘Traitements Actions’ ou ‘Contrôleurs’ :**
→ Les fonctions ou **‘Actions’** liées à une même table en Bdd sont regroupées dans **UN** même fichier (*controleur_article, controleur_utilisateur*)



C3 : Fichier de routes

- **Fichier 'index.php' :**
 - Lié à **deux paramètres en l'URL** avec `$_GET` pour accéder aux fichiers contrôleurs puis aux fonctions actions.
 - Utiliser des conditions `if/else` ou `switch`

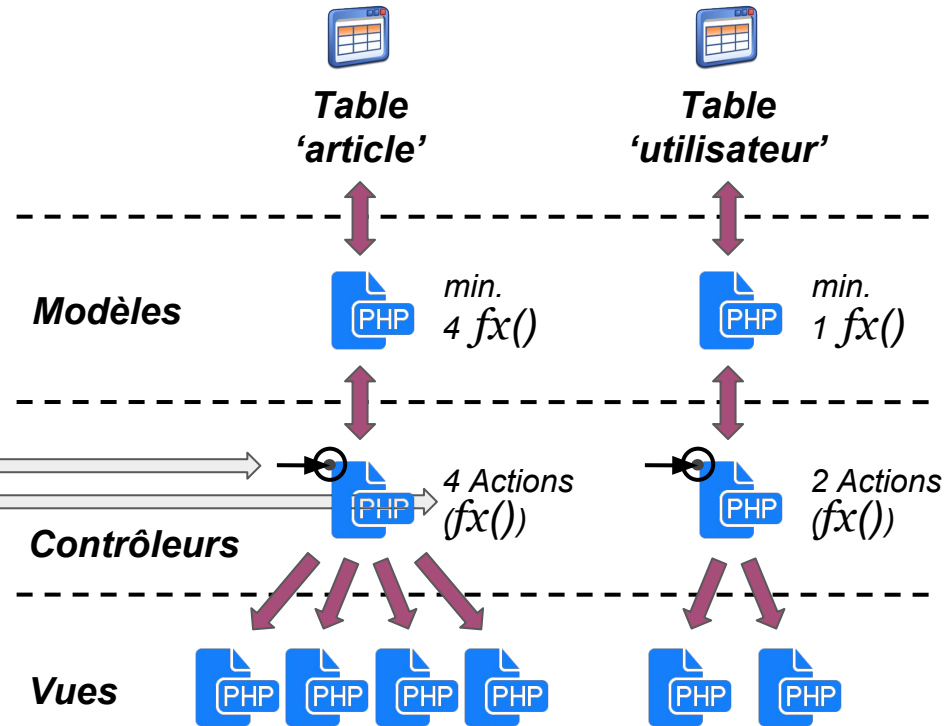
 `index.php?control=info1&action=info2`



if `$_GET['control']`
`$_GET['action']`

Autre méthode :
 Dynamisation
 des informations

Pas d'écriture objet - 1 point d'accès par **Contrôleur** →



11 Fichiers
 (1 + 2 + 2 + 6)

II. Organisation et améliorations

- A. [Organisation des fichiers - schéma](#)
- B. [Dossier 'Config' - schéma](#)
- C. [Fichiers Maîtres - schéma](#)
- D. [Template des pages - schéma](#)
- E. [Conversion en Orienté Objet - schéma](#)

A : Organisation des fichiers

- *L'architecture mise au point lors des chapitres précédents peut être qualifiée d'architecture Modèle-Vue-Contrôleur. Cependant elle est encore basique. Il s'agit désormais d'explorer certaines des optimisations naturelles qu'on peut effectuer lorsqu'on opte pour ce type d'architecture.*
- ***Dans un premier temps***, nous pouvons organiser notre système de fichiers de telle sorte que l'essentiel des programmes ne se situent pas à la racine du serveur. Cette réorganisation nous permet de nous assurer que les programmes ne peuvent pas faire l'objet de requête HTTP non désirée. Le seul programme accessible sera le routeur.

La racine du site Internet contient le routeur ainsi que les fichiers statiques. On évite ainsi que des visiteurs mal intentionnés puissent demander l'exécution de programmes en dehors de ceux dont l'exécution est prévue par le routage.

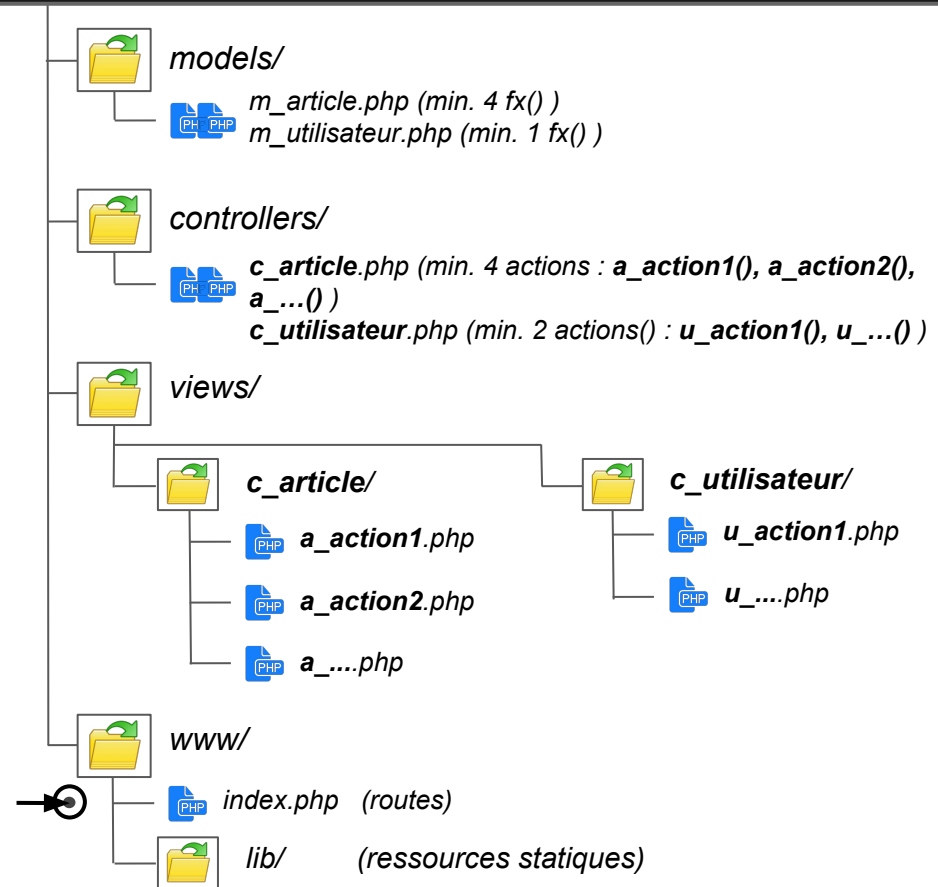
→ [Cf. schéma A : Organisation des fichiers](#)

A : Organisation des fichiers

- 4 dossiers :
 - 'models' : pour tout les '**modèles**'
 - 'controllers' : pour tout les '**contrôleurs**'
 - 'views' : pour toutes les '**vues**'
 - 1 sous-dossier par contrôleur (mêmes noms)
 - 1 fichier par action (mêmes noms)
 - 'www/' (ou web/, public/, htdocs/) : '**racine du site**'
 - Fichier de routes - initialisation site (index.php)
 - Ressources statiques (css/, js/, images/, ...)

4 dossiers
2 + 1 sous-dossiers
11 Fichiers
(1 + 2 + 2 + 6)

Pas d'écriture objet - 1 seul point d'accès : **index.php**



B : Amélioration : Dossier 'Config'

- *Créer un fichier de configuration de la connexion à la base de données et un fichier de configuration du routage pour isoler les configurations et ainsi en faciliter la modification.*
 - *Créer un fichier de configuration de la connexion à la base de données :*

On crée un fichier PHP dans lequel on essaie de simplifier au maximum la configuration des paramètres de connexion à la base de données. On déclare, par exemple, un tableau (array). Dans ce tableau chaque index correspond à un paramètre nécessaire à la connexion à la base de données. On modifie alors les modèles pour utiliser ce fichier pour extraire et utiliser les informations de connexion à la base de données appropriées.
 - *Créer un fichier de configuration du routage :*

On crée un fichier PHP dans lequel on essaie de simplifier au maximum la configuration des routes. On déclare, par exemple, un tableau (array). Dans ce tableau chaque index est la route d'un contrôleur et contient un tableau pour lequel chaque index est la route d'une action et contient un tableau contenant lui même le contrôleur à charger et l'action à déclencher. On modifie alors le routeur pour charger ce fichier et on en adapte le code pour créer un programme qui soit capable de parcourir le tableau pour charger le contrôleur et l'action correspondant aux paramètres de la requête HTTP reçue.

→ [Cf. schéma B : Dossier 'Config'](#)

B : Dossier 'Config'

- 2 fichiers :

→ Fichier 1 (bdd.php) :

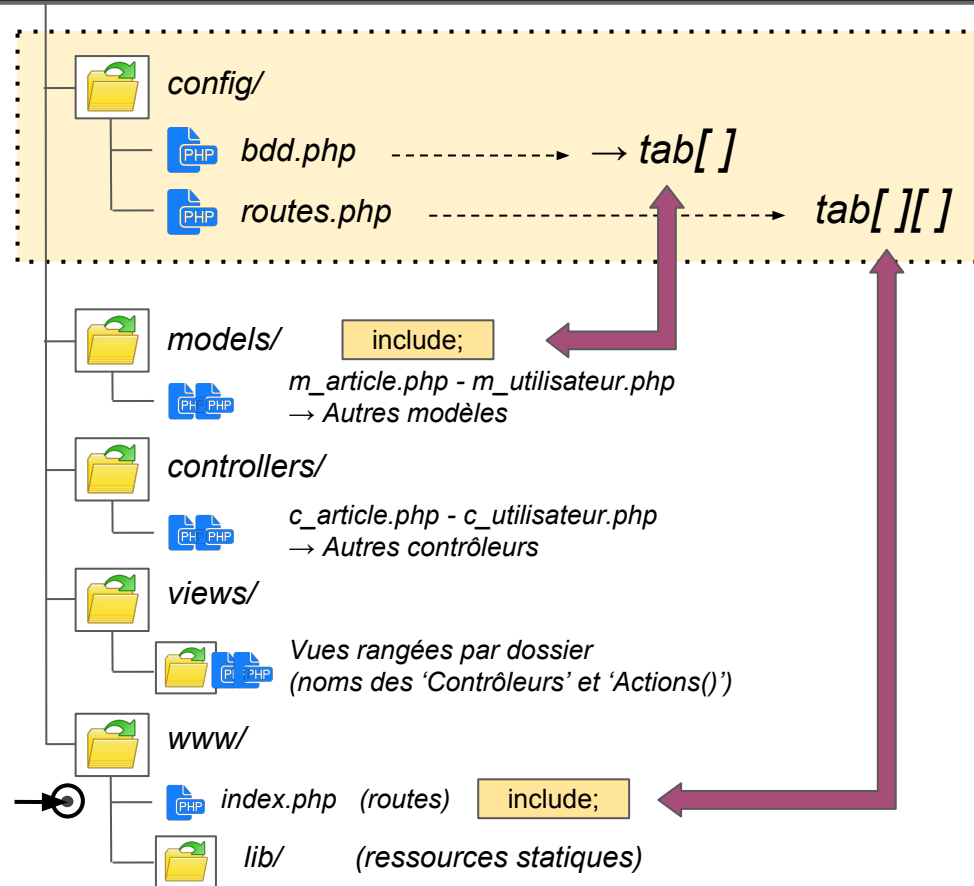
- Est inclus dans les fonctions des modèles
- Contient une fonction retournant un tableau avec les éléments de connexion à la Bdd

→ Fichier 2 (routes.php) :

- Est inclus dans le fichier d'initialisation du site
- Contient un tableau à 2 dimensions associant 1 paramètre de l'url aux contrôleurs et actions liés

5 dossiers
2 + 1 sous-dossiers
13 Fichiers
(1 + 2 + 2 + 2 + 6)

Pas d'écriture objet - 1 seul point d'accès : **index.php**



C : Amélioration : Fichiers Maîtres

- *Créer un fichier « super-Contrôleur » qui regroupera les fonctions communes à tous les Contrôleurs.*
- *Créer un fichier « super-Modèle » qui regroupera les fonctions communes à tous les Modèles.*

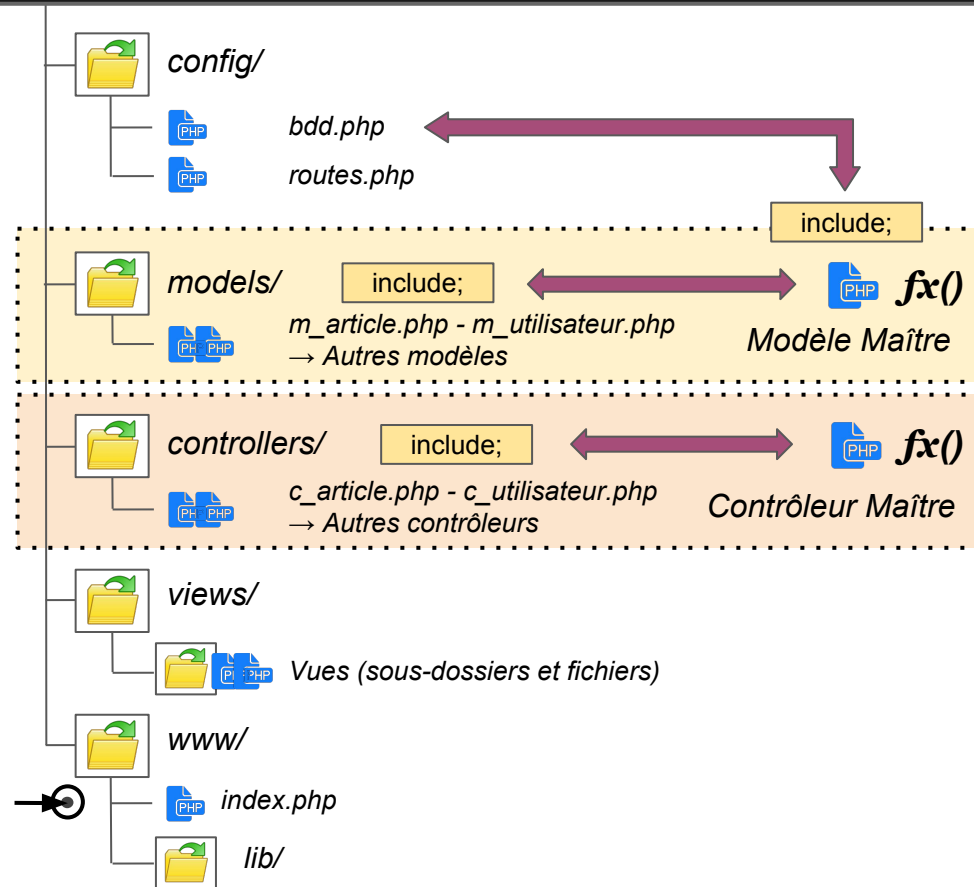
→ [Cf. Schéma C : Fichiers Maîtres](#)

C : Fichiers Maîtres

- Ajouter les 'supers' fichiers :
 → **Modèle Maître : éléments communs** à tous les modèles (exemple : connexion à la bdd)
 → **Contrôleur Maître : éléments communs** à tous les contrôleurs (exemple : initier les sessions)

5 dossiers
 2 + 1 sous-dossiers
15 Fichiers
 (1 + 2 + 3 + 3 + 6)

Pas d'écriture objet - 1 seul point d'accès : **index.php**



A : Amélioration : Template des pages

- *Créer un « layout » (i.e : affichage) commun, c'est-à-dire créer un système d'affichage qui permet de regrouper les éléments d'affichage commun à toutes les vues.*
 - *Pour préserver la structure « verticale » MVC on utilise les possibilités offertes par l' « output buffering » en PHP. On génère la Vue mais on ne la transmet pas au client, on l'enregistre dans la mémoire tampon (« buffer » de sortie). Puis, on génère le « layout » commun et on affiche au sein du « layout » commun le contenu de la mémoire tampon.*
- [Cf. Schéma D : Template \(ou layout\) des pages](#)

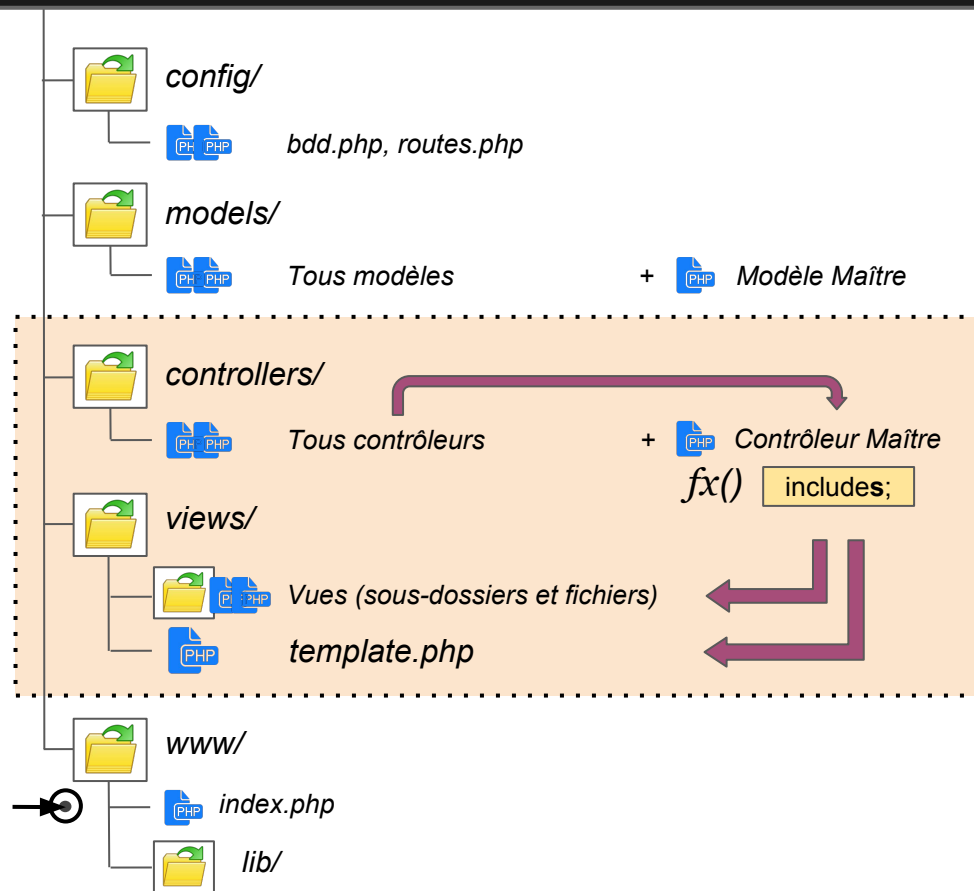
D : Template (ou layout) des pages

- **1 Fichier** commun avec **header/footer**
 - Fichiers **vues** inclus dans le template
 - Fonction '**render()**' dans le '**Contrôleur Maître**' exécutée depuis chaque action

Fonctions PHP de
temporisation de sortie
(ou output buffering)

5 dossiers
6 + 1 sous-dossiers
16 Fichiers
(1 + 2 + 3 + 3 + 7)

Pas d'écriture objet - **1 seul point d'accès : index.php**

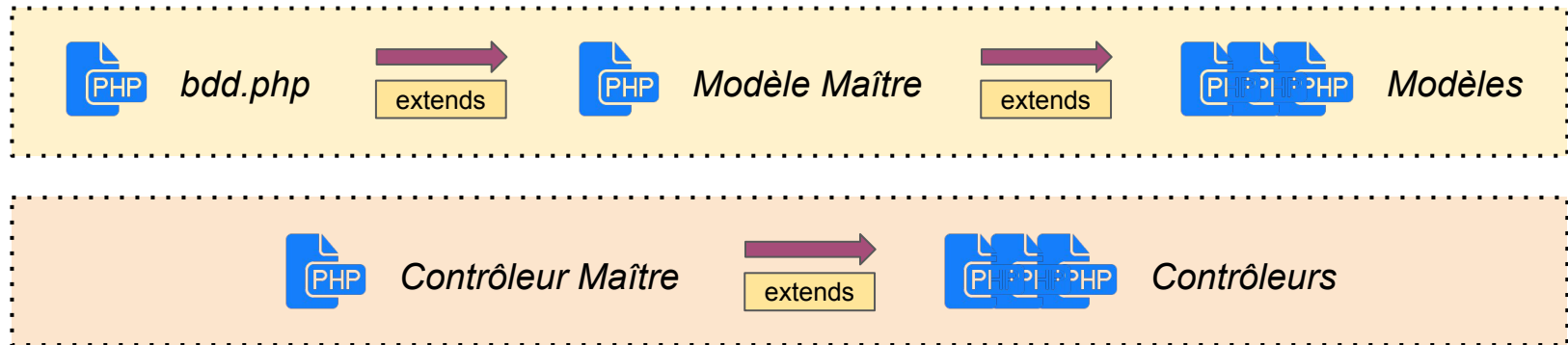


A : Amélioration : Conversion en Orienté Objet

- *Convertir les Modèles (et le « super-Modèle ») et les Contrôleurs (et le « super-Contrôleur ») en classes puis les utiliser en tant qu'objets contenant des méthodes.*
- [Cf. Schéma E : Conversion en Orienté Objet](#)

E : Conversion en Orienté Objet

- Les fichiers suivant contiennent des **Classes** :
 - *Tous modèles + 'bdd.php', tous Contrôleurs*
 - *Toutes les **fonctions** présentes deviennent des **méthodes***



Règles de l'écriture objet
à respecter

Tous fichiers :

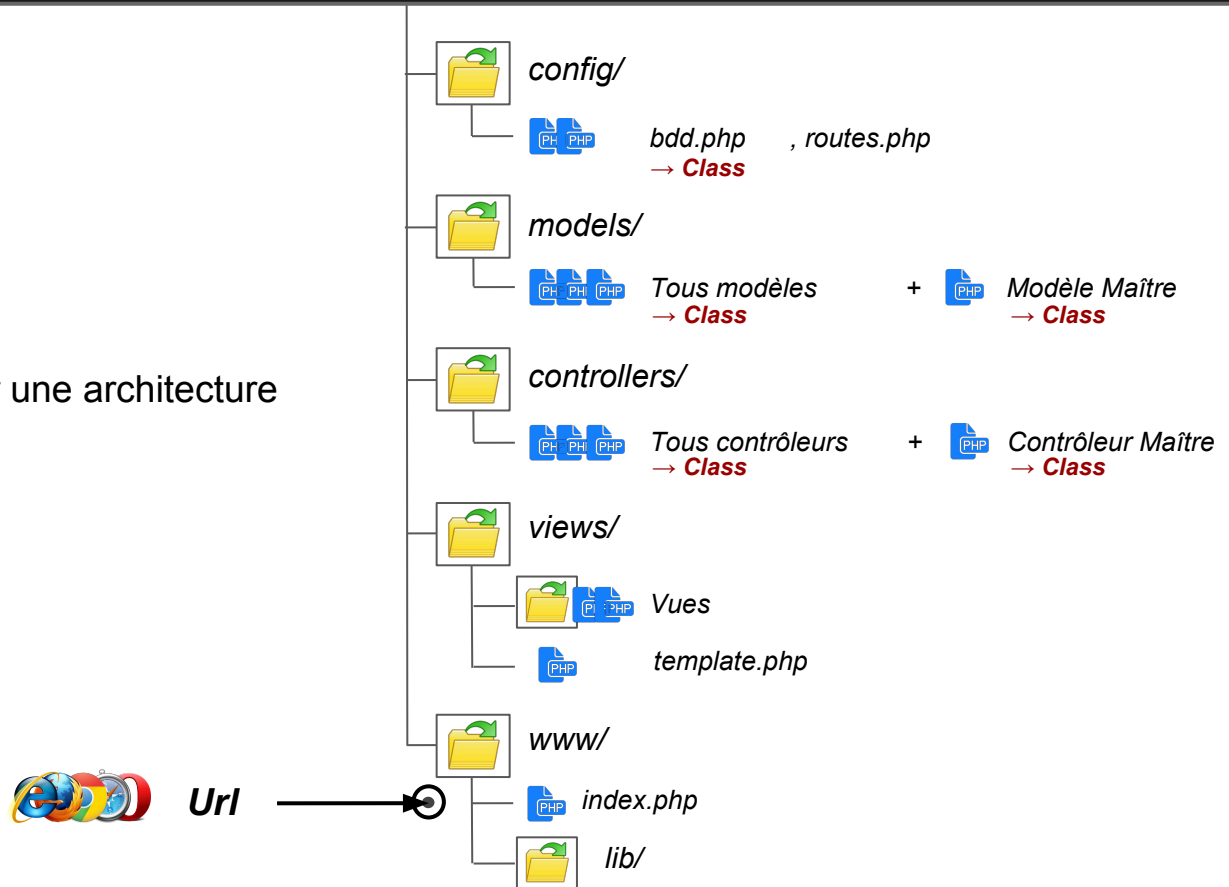
Remplacer les exécutions de fonctions par celles de méthodes objets
→ *Fichiers de routes : possibilité de dynamiser avec les classes*

III. Récapitulatifs

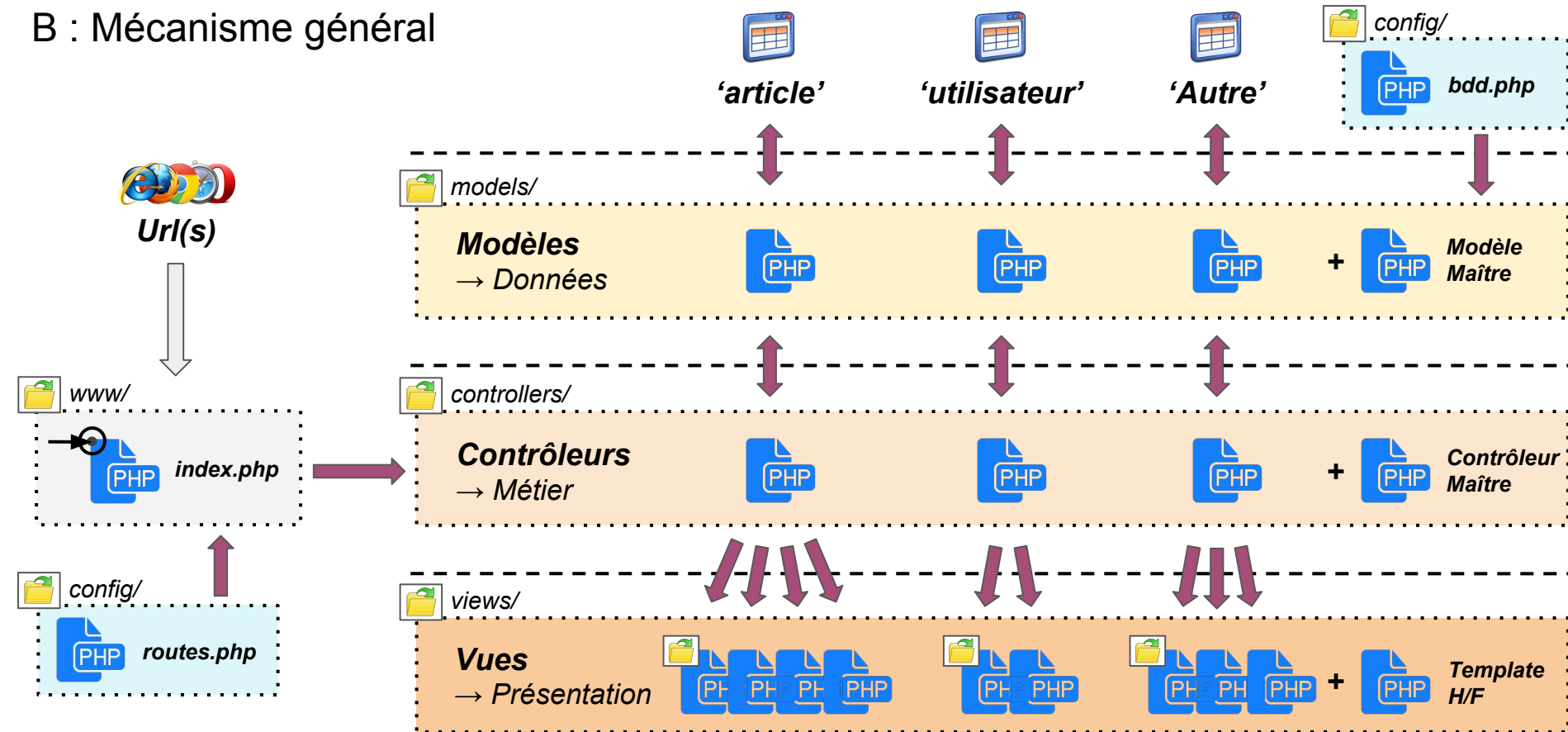
- A. [Arborescence](#)
- B. [Mécanisme général](#)

A : Arborescence

- *Rangement de base* pour une architecture **Model View Controller**



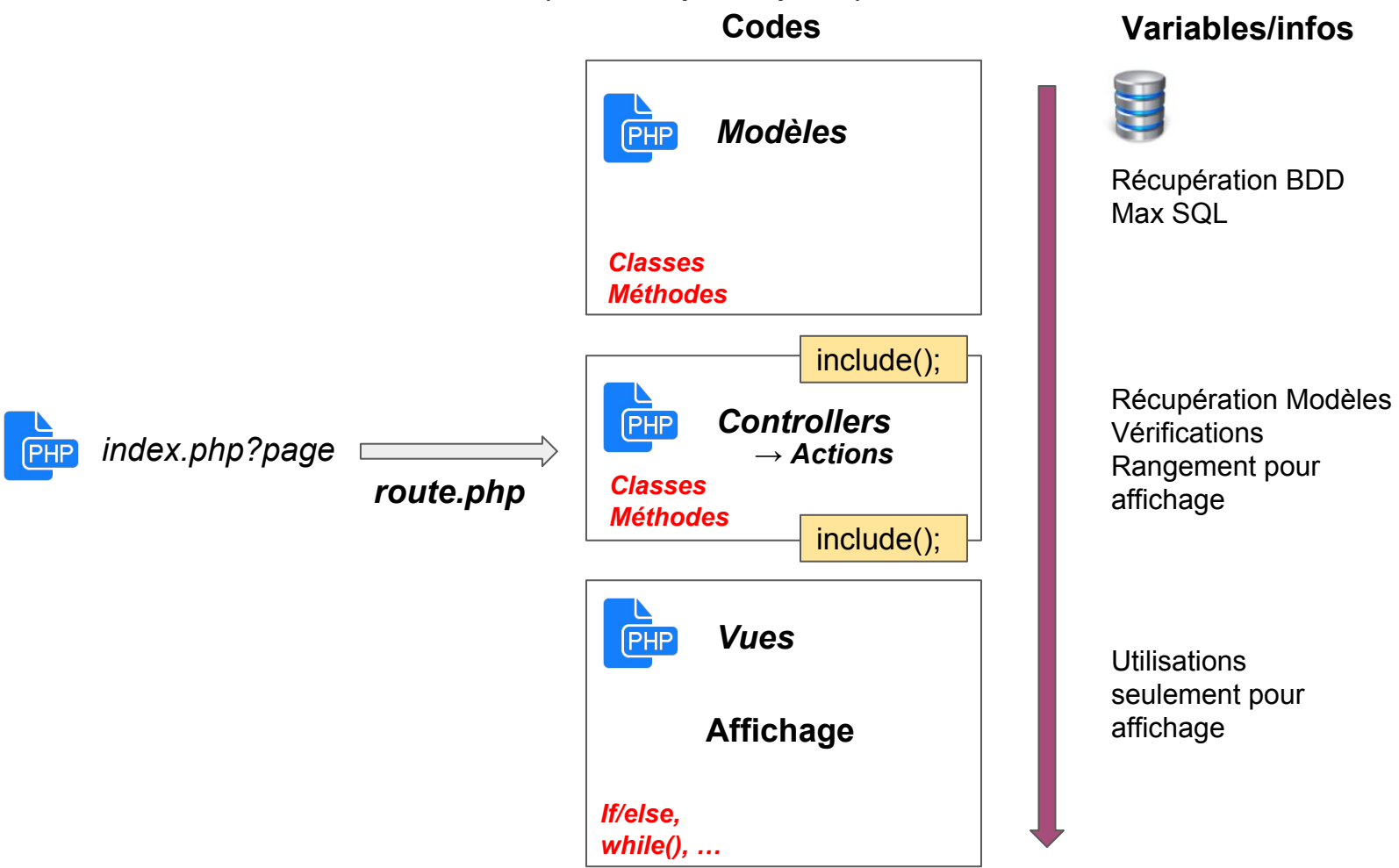
B : Mécanisme général



IV. Bonnes pratiques

- A. Schéma général
- B. Inclusions des fichiers

A. Bonus - Schéma Général (bonne pratiques)



B. Bonus - Bonnes pratiques - inclusions

