



Symfony

Objectif :
Comprendre et savoir utiliser le « framework »
Symfony 2

Sommaire

Introduction.....	3
a. Notion de « framework »	3
b. Pourquoi Symfony 2	3
I. Genèse d'une architecture MVC	4
a. Programmation « flat »	4
b. Séparation des logiques	5
c. Rationalisation de la structure.....	9
d. Modèle Vue Contrôleur (MVC)	12
II. Architecture MVC de Symfony2	18
a. Système de fichier et espaces de nom	18
b. Configuration du routage et métalangage « YAML »	20
c. Création des vues et métalangage « Twig »	21
d. Création des Modèles et Doctrine2.....	22
III. Fonctionnalités de Symfony2	23
a. Utilisation de la console.....	23
b. Création et exploitation de formulaires	23
c. Gestion des requêtes et des sessions	24
d. Sécurité, gestion des évènements, injections de dépendance.....	24

Introduction

- « framework » : Ensemble de composants structurels permettant de construire tout ou partie d'un logiciel ou d'une application Web.

a. Notion de « framework »

Le mot « framework » est issu des mots anglais « frame » et « work ». Le mot anglais « frame » pourrait être traduit par le mot français « cadre » ou « charpente » et « work » par « travail » ou « œuvre ». Littéralement, le mot « framework » pourrait être traduit par l'expression française « **cadre de travail** » ou bien encore « **charpente d'une œuvre** ».

Dans notre domaine, l'informatique, le mot « framework » désigne un ensemble d'outils et de composants logiciels qui vont être notre « cadre de travail » ainsi que la « charpente d'une œuvre » logicielle.

Un « framework » logiciel s'appuie généralement sur des éléments logiciels connexes à certaines technologies en respectant des **conventions**, des **bonnes pratiques** ainsi qu'une **architecture logicielle**.

Dans le domaine des applications Web, l'architecture logicielle qui semble la plus adaptée et qui est celle préconisée à l'heure actuelle par la plupart des « frameworks » pour le développement est l'architecture **Modèle-Vue-Contrôleur (MVC)**. Le « framework » que nous avons choisi d'étudier s'appuie sur les préconisations de l'architecture MVC dont le paradigme consiste à insister sur la séparation des logiques relatives à :

- L'accès aux données (**Modèles**).
- Le traitement de données (**Contrôleurs**).
- L'affichage des données (**Vues**).

b. Pourquoi Symfony 2

Les « framework » PHP/SQL les plus populaires à l'heure actuelle sont :

- Laravel : <http://laravel.com/> par Taylor Otwell. Laravel est celui qui enregistre ces derniers temps la plus forte hausse de popularité.
- CodeIgniter : <https://ellislab.com/codeigniter> par EllisLab. CodeIgniter (comme CakePHP) s'appuie sur les conventions plutôt que la configuration : apprendre à respecter certaines règles permet de s'affranchir des paramétrages parfois fastidieux qui sont nécessaire pour exploiter un « framework ».
- CakePHP : <http://cakephp.org/> par le Cake Software Foundation. CakePHP est le seul qui soit réellement Open Source, c'est-à-dire supporté uniquement par une communauté de développeurs passionnés.
- Yii : <http://www.yiiframework.com/> par Yii Software LLC. Yii met l'accent sur l'optimisation des performances et la sécurité des développements.
- Phalcon : <http://docs.phalconphp.com/en/latest/index.html> par Phalcon Team. Phalcon est probablement le plus performant puisqu'une partie de ses composants sont directement programmés en langage système (en C).
- Zend Framework: <http://framework.zend.com/> par Zend Technologies. Zend Framework est soutenu par Zend Technologie qui est l'entreprise qui soutient commercialement le langage PHP ce qui est gage de pérennité.
- Symfony: <http://symfony.com/> par SensioLabs. Symfony2 est le « framework » que nous allons étudier.

Si nous avons choisi d'étudier les principes et le fonctionnement du « framework » Symfony c'est parce qu'à l'heure actuelle, dans le contexte français, il s'agit de celui pour lequel on recense le plus grand nombre d'offres d'emploi.

I. Genèse d'une architecture MVC

a. Programmation « flat »

- « flat » : 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 œuvre 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.

L'exemple ci-dessous est un traitement en PHP dont l'objet est d'afficher les nombre de lignes enregistrés dans une table de base de données :

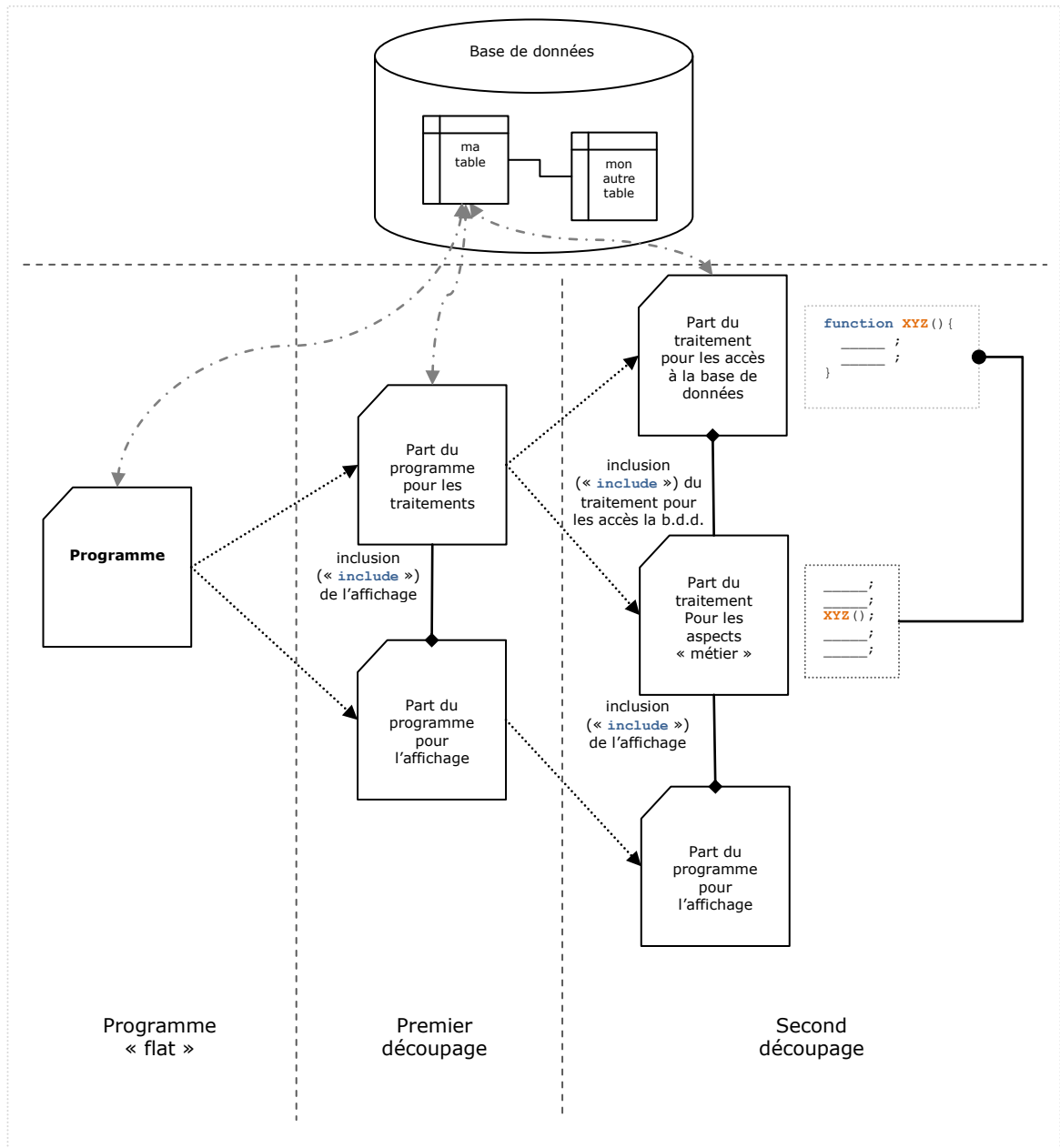
```
<?php
//« traitement_1.php »
// TRAITEMENT 1 : Affiche le nombre d'entrées de maTable :
//initialisation de variables utilitaires
$erreur = $message = $nombre_de_resultats = false;
//connexion à la base de données
$objet_mysql = new mysqli('le_serveur_MySQL','user','pass','ma_bdd');
if(mysqli_connect_error()){ //SI on constate une erreur
    //construction d'un message d'erreur
    $erreur = 'Erreur de connexion (' . mysqli_connect_errno() . ') ' .
mysqli_connect_error();
}else{ //SINON
    $requete = 'SELECT COUNT(*) FROM ma_table;'; //construction de la requete
    $objet_mysql_result = $objet_mysql->query($requete); //execution de la requête
    if($objet_mysql_result){ //SI on obtient un objet mysqli_result et pas false
        $resultats = $objet_mysql_result->fetch_assoc();
        if(empty($resultats['COUNT(*)'])){
            $message = 'Ma table ne contient pas de ligne.';
        }else{
            $message = 'Ma table contient ' . $resultats['COUNT(*)'] . ' lignes.';
        }
    }else{ //SINON
        $erreur = 'Erreur : ' . $objet_mysql->error; //construction d'un message
d'erreur
    }
}
echo '<!doctype html>';
echo '<html lang="fr"><head><meta charset="utf-8"><title>Ma
page</title></head><body>';
if($erreur === false){ //SI la variable $erreur contient false
    echo '<h1>La requête a été effectuée.</h1>';
    echo '<p>';
    echo $message;
    echo '</p>';
}else{ //SINON
    echo '<h1>Une erreur est survenue !</h1>';
    echo '<p>';
    echo $erreur;
    echo '</p>';
}
echo '</body></html>';
?>
```

Ce traitement ne respecte aucune règle d'architecture en particulier. Il va s'agir d'organiser ce programme de telle sorte qu'il soit construit selon les règles d'une architecture MVC.

b. Séparation des logiques

- Logique : Enchaînement naturel, normal, nécessaire des événements.

Pour construire une architecture MVC à partir d'un programme sans organisation particulière, nous allons procéder en 3 étapes comme décrit sur le schéma qui suit :



Dans un premier temps nous allons découper notre programme 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`).

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.

- **Premier découpage : Séparer l’affichage du reste du programme**

- L’exemple ci-dessous illustre le premier découpage sur la base du fichier proposé plus haut :

```
<?php
// « traitement_1.php »
// TRAITEMENT 1 : Affiche le nombre d'entités de ma table :
//initialisation de variables utilitaires
$erreur = $message = $nombre_de_resultats = false;
//connexion à la base de données
$objet_mysql = new mysqli('le_serveur_MySQL','user','pass','ma_bdd');
if(mysqli_connect_error()){ //SI on constate une erreur
    //construction d'un message d'erreur
    $erreur = 'Erreur de connexion (' . mysqli_connect_errno() . ') ' .
mysqli_connect_error();
}else{ //SINON
    $requete = 'SELECT COUNT(*) FROM ma_table;'; //construction de la requête
    $objet_mysql_result = $objet_mysql->query($requete); //execution de la requête
    if($objet_mysql_result){ //SI on obtient un objet mysqli_result et pas false
        $resultats = $objet_mysql_result->fetch_assoc();
        if(empty($resultats['COUNT(*)'])){
            $message = 'Ma table ne contient pas de ligne.';
        }else{
            $message = 'Ma table contient ' . $resultats['COUNT(*)'] . ' lignes.';
        }
    }else{ //SINON
        $erreur = 'Erreur : ' . $objet_mysql->error; //construction d'un message d'erreur
    }
}
//inclusion de l’affichage pour TRAITEMENT 1 : « affichage_pour_traitement_1.php »
include('affichage_pour_traitement_1.php');
?>
```

```
<?php
//« affichage_pour_traitement_1.php »
//Affichage pour TRAITEMENT 1 :
echo '<!doctype html>';
echo '<html lang="fr"><head><meta charset="utf-8"><title>Ma
page</title></head><body>';
if($erreur == false){ //SI la variable $erreur contient false
    echo '<h1>La requête a été effectuée.</h1>';
    echo '<p>';
    echo $message;
    echo '</p>';
}else{ //SINON
    echo '<h1>Une erreur est survenue !</h1>';
    echo '<p>';
    echo $erreur;
    echo '</p>';
}
echo '</body></html>';
?>
```

On obtient 2 fichiers. Le premier, « traitement_1.php », est consacré à l’accès aux données et aux traitements métier. Le second, « affichage_pour_traitement_1.php », est consacré à l’affichage.

Le premier fichier, « traitement_1.php », fait office de **point d’entrée** pour le programme : c’est l’url de ce fichier qui devra être saisie pour démarrer le programme.

- **Second découpage : Séparer l'accès aux données du programme restant**

- L'exemple ci-dessous illustre le second découpage sur la base du premier découpage effectué plus haut :

```
<?php
//« operation_bdd_pour_traitement_1.php »
//Fonction qui réalise une opération en b.d.d. pour TRAITEMENT 1
function quel_est_le_nombre_de_resultats(){
    //connexion à la base de données
    $objet_mysql = new mysqli('le_serveur_MySQL','user','pass','ma_bdd');
    if(mysqli_connect_error()){ //SI on constate une erreur
        return false;
    }else{ //SINON
        $requete = 'SELECT COUNT(*) FROM ma_table;'; //construction de la requête
        $objet_mysql_result = $objet_mysql->query($requete); //execution de la requête
        if($objet_mysql_result){ //SI on obtient un objet de type mysqli_result et pas false
            $resultats = $objet_mysql_result->fetch_assoc();
            if(empty($resultats['COUNT(*)'])){
                return 0;
            }else{
                return $resultats['COUNT(*)'];
            }
        }else{ //SINON
            return false;
        }
    }
}
?>
```

```
<?php
//« traitement_1.php »
//TRAITEMENT 1 : Affiche le nombre d'entités de ma table :
//inclusion de l'opération en b.d.d. pour TRAITEMENT 1 : « operation_bdd_pour_traitement_1.php »
include('operation_bdd_pour_traitement_1.php');
//initialisation de variables utilitaires
$erreur = $message = $nombre_de_resultats = false;
//utilisation de la fonction déclarée plus haut
$nombre_de_resultats = quel_est_le_nombre_de_resultats();
if($nombre_de_resultats === false){ //SI la fonction retourne false
    $erreur = 'Impossible d\'obtenir un nombre de résultats.';
}else{ //SINON
    if($nombre_de_resultats > 0){
        $message = 'Ma table contient ' . $nombre_de_resultats . ' lignes.';
    }else{
        $message = 'Ma table ne contient pas de ligne.';
    }
}
//inclusion de l'affichage pour TRAITEMENT 1 : « affichage_pour_traitement_1.php »
include('affichage_pour_traitement_1.php');
?>
```

```
<?php
// « affichage_pour_traitement_1.php »
//Affichage pour TRAITEMENT 1 : « affichage_pour_traitement_1.php »
echo '<!doctype html>';
echo '<html lang="fr"><head><meta charset="utf-8"><title>Ma page</title></head><body>';
if($erreur === false){ //SI la variable $erreur contient false
    echo '<h1>La requête a été effectuée.</h1>';
    echo '<p>';
    echo $message;
    echo '</p>';
}else{ //SINON
    echo '<h1>Une erreur est survenue !</h1>';
    echo '<p>';
    echo $erreur;
    echo '</p>';
}
echo '</body></html>';
?>
```

On obtient 3 fichiers. Le premier, « operation_bdd_pour_traitement_1.php », est consacré à l'accès aux données. Le second, « traitement_1.php », est consacré aux traitements métier. Le troisième, « affichage_pour_traitement_1.php », est consacré à l'affichage. Le second fichier, « traitement_1.php », reste le **point d'entrée** pour le programme.

Ci-après un exemple de découpage avec un autre programme :

```
<?php
//« operation_bdd_pour_traitement_2.php »
//Fonction qui réalise une opération en bdd pour TRAITEMENT 2
function recupere_tous_les_resultats(){
    //connexion à la base de données
    $objet_mysql = new mysqli('le_serveur_MySQL','user','pass','ma_bdd');
    if(mysqli_connect_error()){ //SI on constate une erreur
        return false;
    }else{ //SINON
        $requete = 'SELECT * FROM ma_table;'; //construction de la requête
        $objet_mysql_result = $objet_mysql->query($requete); //exécution de la requête
        if($objet_mysql_result){ //SI on obtient un objet de type mysqli_result et pas false
            $resultats = array(); //on crée un tableau vide
            $nombre_de_lignes = $objet_mysql_result->num_rows; //on récupère le nombre de lignes de résultat
            for($ligne = 0; $ligne < $nombre_de_lignes; $ligne++){ //POUR CHAQUE ligne de résultat
                $resultats[] = $objet_mysql_result->fetch_assoc(); //on sauvegarde la ligne dans le tableau
            }
            return $resultats; //on retourne le tableau
        }else{ //SINON
            return false;
        }
    }
}
?>
```

```
<?php
//« traitement_2.php »
//TRAITEMENT 2 : Affiche les entités contenues dans ma table :
//inclusion de l'opération en bdd pour TRAITEMENT 2 : « operation_bdd_pour_traitement_2.php »
include('operation_bdd_pour_traitement_2.php');
//initialisation de variables utilitaires
$erreur = $pas_de_resultats = false;
//utilisation de la fonction déclarée plus haut
$resultats_obtenus = recupere_tous_les_resultats();
if($resultats_obtenus === false){ //SI la fonction retourne false
    $erreur = 'Impossible d\'obtenir un nombre de résultats.';
}else{ //SINON
    if(empty($resultats_obtenus)){
        $pas_de_resultats = true;
    }
}
//inclusion de l'affichage pour TRAITEMENT 2 : « affichage_pour_traitement_2.php »
include('affichage_pour_traitement_2.php');
?>
```

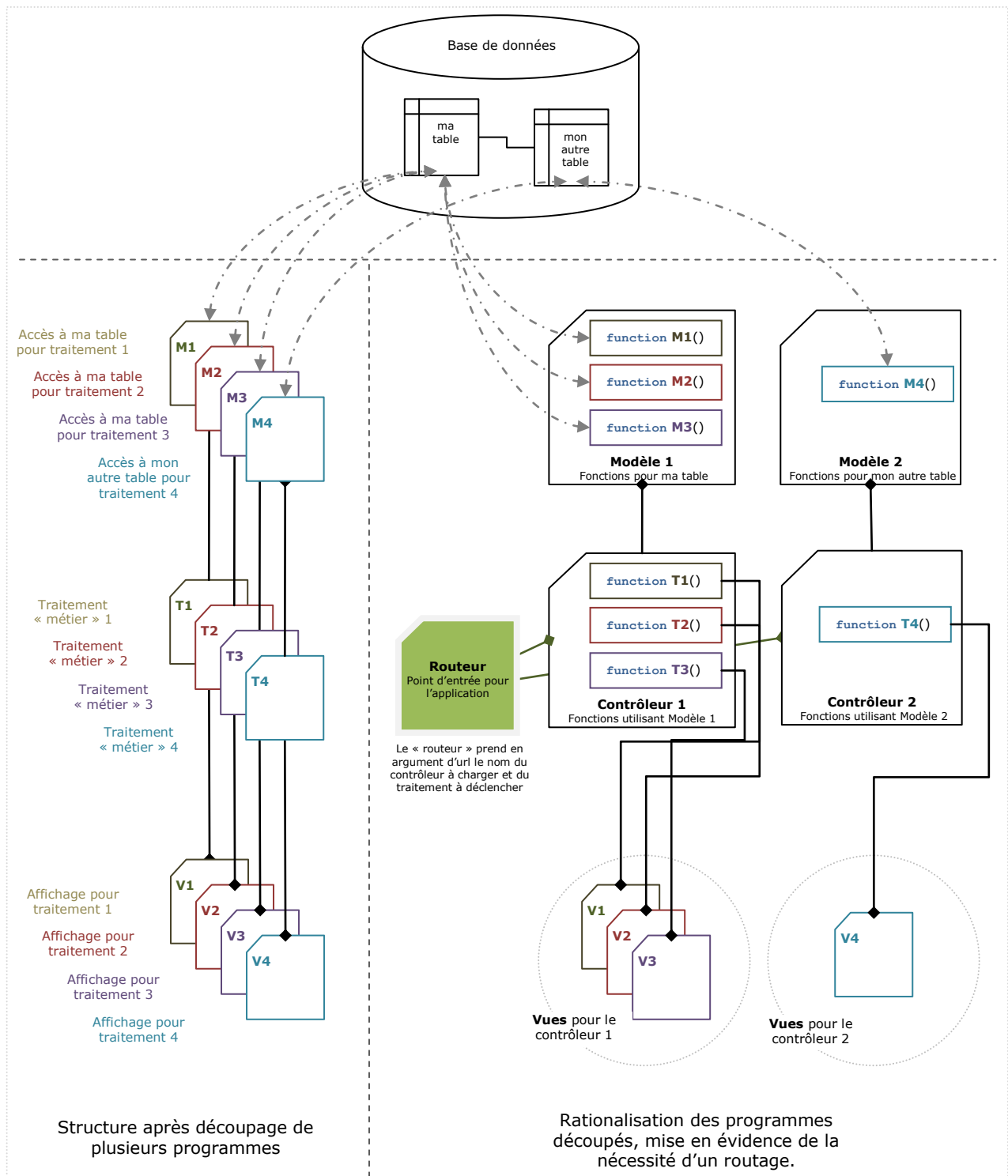
```
<?php
// « affichage_pour_traitement_2.php »
//Affichage pour TRAITEMENT 2 : « affichage_pour_traitement_2.php »
echo '<!doctype html>';
echo '<html lang="fr"><head><meta charset="utf-8"><title>Mon autre page</title></head><body>';
if($erreur === false){ //SI la variable $erreur contient false
    echo '<h1>La requête a été effectuée.</h1>';
    if($pas_de_resultats){
        echo '<p>';
        echo 'La table ma table ne contient aucune entité.';
        echo '</p>';
    }else{
        echo '<ol>';
        foreach($resultats_obtenus as $resultat_obtenu){
            echo '<li>&nbsp;';
            echo '<ul>';
            foreach($resultat_obtenu as $nom_de_colonne => $valeur_de_la_colonne){
                echo '<li>';
                echo '<strong>';
                echo $nom_de_colonne;
                echo '</strong>';
                echo '&nbsp;';
                echo $valeur_de_la_colonne;
                echo '</p>';
            }
            echo '</ul>';
            echo '</li>';
        }
        echo '</ol>';
    }
}else{ //SINON
    echo '<h1>Une erreur est survenue !</h1>';
    echo '<p>';
    echo $erreur;
    echo '</p>';
}
echo '</body></html>';
?>
```


c. Rationalisation de la structure

- Rationalisation : réorganisation pour accroître l'efficacité.

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.

- Le diagramme ci-après illustre les rationalisations à effectuer pour 4 programmes découpés en suivant les étapes décrites précédemment :



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**.

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 **roulage**. 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.

Les exemples de Modèle-Vue-Contrôleur ci-après permettent d'illustrer techniquement les différents éléments de l'architecture MVC et de donner une idée concrète d'un programme de routage (i.e : routeur) :

• Rationalisation de l'accès aux données : Exemple de Modèle

```
<?php
//« modele_pour_ma_table.php »
//Fonction qui réalise une opération en b.d.d. pour TRAITEMENT 1
function quel_est_le_nombre_de_resultats(){
    //connexion à la base de données
    $objet_mysql = new mysqli('le_serveur_MySQL','user','pass','ma_bdd');
    if(mysqli_connect_error()){ //SI on constate une erreur
        return false;
    }else{ //SINON
        $requete = 'SELECT COUNT(*) FROM ma_table;'; //construction de la requête
        $objet_mysql_result = $objet_mysql->query($requete); //execution de la requête
        if($objet_mysql_result){ //SI on obtient un objet de type mysqli_result et pas false
            $resultats = $objet_mysql_result->fetch_assoc();
            if(empty($resultats['COUNT(*)'])){
                return 0;
            }else{
                return $resultats['COUNT(*)'];
            }
        }else{ //SINON
            return false;
        }
    }
}

//Fonction qui réalise une opération en bdd pour TRAITEMENT 2
function recupere_tous_les_resultats(){
    //connexion à la base de données
    $objet_mysql = new mysqli('le_serveur_MySQL','user','pass','ma_bdd');
    if(mysqli_connect_error()){ //SI on constate une erreur
        return false;
    }else{ //SINON
        $requete = 'SELECT * FROM ma_table;'; //construction de la requête
        $objet_mysql_result = $objet_mysql->query($requete); //execution de la requête
        if($objet_mysql_result){ //SI on obtient un objet de type mysqli_result et pas false
            $resultats = array(); //on crée un tableau vide
            $nombre_de_lignes = $objet_mysql_result->num_rows; //on récupère le nombre de lignes de résultat
            for($ligne = 0; $ligne < $nombre_de_lignes;$ligne++){ //POUR CHAQUE ligne de résultat
                $resultats[] = $objet_mysql_result->fetch_assoc(); //on sauvegarde la ligne dans le tableau
            }
            return $resultats; //on retourne le tableau
        }else{ //SINON
            return false;
        }
    }
}
?>
```

- **Rationalisation des logiques métiers : Exemple de Contrôleur**

```
<?php
//« controleur_pour_ma_table.php »
function traitement_1(){
    //TRAITEMENT 1 : Affiche le nombre d'entités de ma table :
    //inclusion de l'opération en b.d.d. pour TRAITEMENT 1 : « operation_bdd_pour_traitement_1.php »
    include('operation_bdd_pour_traitement_1.php');
    //initialisation de variables utilitaires
    $erreur = $message = $nombre_de_resultats = false;
    //utilisation de la fonction déclarée plus haut
    $nombre_de_resultats = quel_est_le_nombre_de_resultats();
    if($nombre_de_resultats === false){ //SI la fonction retourne false
        $erreur = 'Impossible d\'obtenir un nombre de résultats.' ;
    }else{ //SINON
        if($nombre_de_resultats > 0){
            $message = 'Ma table contient ' . $nombre_de_resultats . ' lignes.';
        }else{
            $message = 'Ma table ne contient pas de ligne.';
        }
    }
    //inclusion de l'affichage pour TRAITEMENT 1 : « affichage_pour_traitement_1.php »
    include('affichage_pour_traitement_1.php');
}

function traitement_2(){
    //TRAITEMENT 2 : Affiche les entités contenues dans ma table :
    //inclusion de l'opération en bdd pour TRAITEMENT 2 : « operation_bdd_pour_traitement_2.php »
    include('operation_bdd_pour_traitement_2.php');
    //initialisation de variables utilitaires
    $erreur = $pas_de_resultats = false;
    //utilisation de la fonction déclarée plus haut
    $resultats_obtenus = recupere_tous_les_resultats();
    if($resultats_obtenus === false){ //SI la fonction retourne false
        $erreur = 'Impossible d\'obtenir un nombre de résultats.' ;
    }else{ //SINON
        if(empty($resultats_obtenus)){
            $pas_de_resultats = true;
        }
    }
    //inclusion de l'affichage pour TRAITEMENT 2 : « affichage_pour_traitement_2.php »
    include('affichage_pour_traitement_2.php');
}
?>
```

- **Les affichages ne changent pas** par rapport aux exemples correspondants donnés précédemment. Chaque affichage correspond à une action (fonction par traitement) contenue dans le fichier Contrôleur.
- **Nécessité du routage : Exemple de « Routeur »** : le routeur ci-après sert de point d'entrée pour l'application. Il s'agira de saisir l'url du routeur avec des paramètres dont l'objet sera de spécifier le nom du contrôleur à charger et de l'action à déclencher. Le routeur ci-après : « index.php » pourrait prendre les paramètres suivant en entrée lors d'une requête HTTP :
 - /index.php?controleur='pour_ma_table'&traitement='traitement_1' pour charger le fichier Contrôleur « controleur_pour_ma_table.php » et déclencher la fonction « traitement_1 » qui correspond à l'action demandée.
 - /index.php?controleur='pour_ma_table'&traitement='traitement_2' pour charger le fichier Contrôleur « controleur_pour_ma_table.php » et déclencher la fonction « traitement_2 » qui correspond à l'action demandée.
 - ...

```
<?php
//« index.php »
//SI le controleur et le traitement ne sont pas transmis dans l'url
if(empty($_GET['controleur']) || empty($_GET['traitement'])){
    echo '<!doctype html>';
    echo '<html lang="fr"><head><meta charset="utf-8"><title>Erreur 404</title></head><body>';
    echo '<h1>Erreur 404 : cette page est introuvable</h1>';
    echo '</body></html>';
}
//SINON
}else{
    $controleur = $_GET['controleur'];
    $traitement = $_GET['traitement'];
    switch($controleur){
        case 'pour_ma_table': //SI $controleur contient 'pour_ma_table'
            //correspond à l'url /index.php?controleur='pour_ma_table'
            switch($traitement){
                case 'traitement_1': //SI $traitement contient 'traitement_1'
                    //correspond à l'url/route : /index.php?controleur='pour_ma_table'&traitement='traitement_1'
                    include('controleur_pour_ma_table.php');
                    traitement_1();
                    break;
                case 'traitement_2': //SI $traitement contient 'traitement_2' :
                    //correspond à l'url/route : /index.php?controleur='pour_ma_table'&traitement='traitement_2'
                    include('controleur_pour_ma_table.php');
                    traitement_2();
                    break;
                default: //SINON affiche une erreur 404
                    echo '<!doctype html>';
                    echo '<html lang="fr"><head><meta charset="utf-8"><title>Erreur 404</title></head><body>';
                    echo '<h1>Erreur 404 : cette page est introuvable</h1>';
                    echo '</body></html>';
            }
        break;
        default: //SINON affiche une erreur 404
            echo '<!doctype html>';
            echo '<html lang="fr"><head><meta charset="utf-8"><title>Erreur 404</title></head><body>';
            echo '<h1>Erreur 404 : cette page est introuvable</h1>';
            echo '</body></html>';
        }
    }
}
?>
```

d. Modèle Vue Contrôleur (MVC)

- MVC : La méthode MVC a été mise au point pour Xerox en 1979 par Trygve Reenskaug. On trouve un des premières formalisations [ici](#).

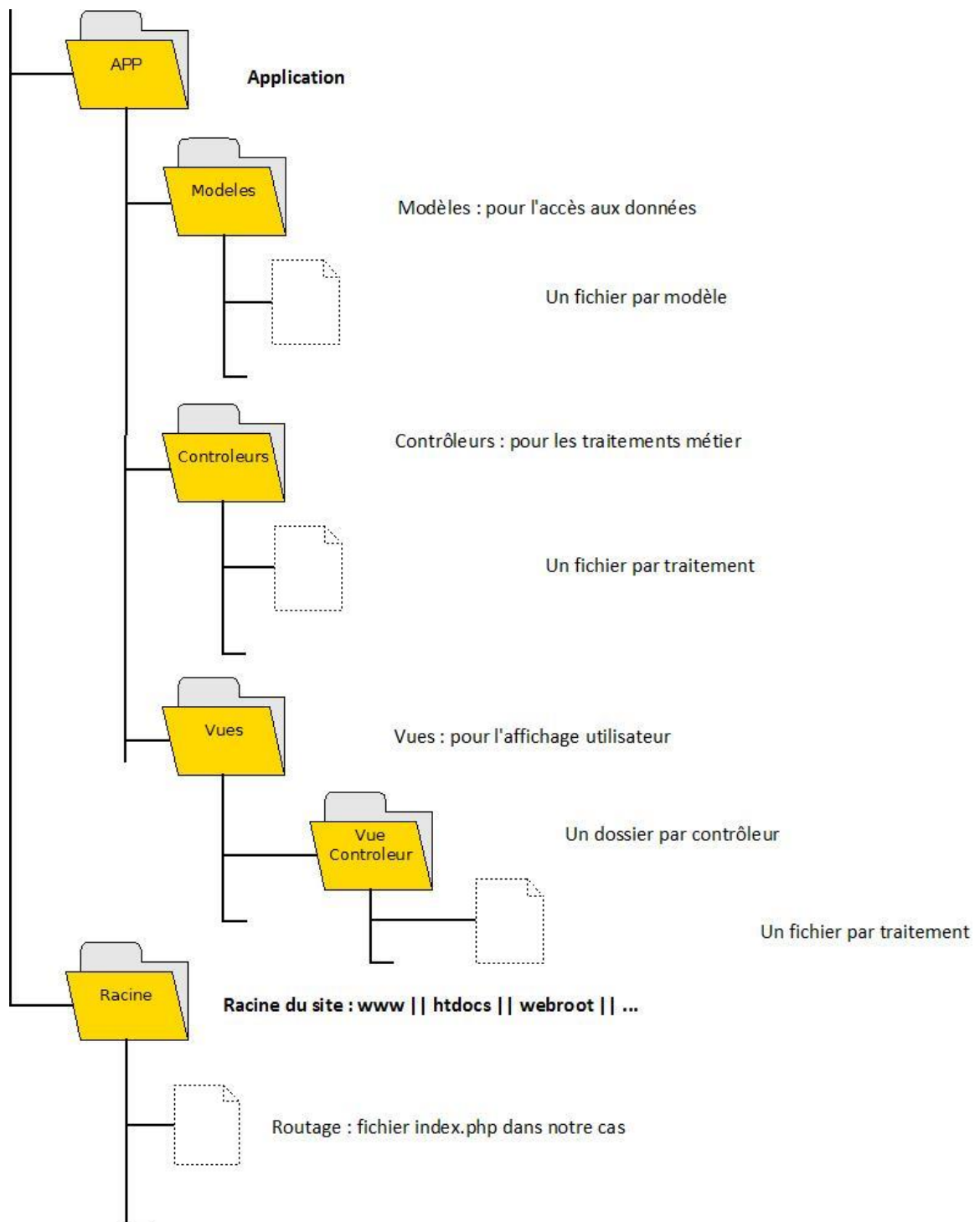
L'architecture mise au point lors des chapitres précédents peut être qualifiée d'architecture Modèle-Vue-Contrôleur. Cependant elle est encore basique. Il s'agit désormais d'explorer certaines des optimisations naturelles qu'on peut effectuer lorsqu'on opte pour ce type d'architecture. Ces optimisations sont de plusieurs ordres.

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.

Dans un second temps, nous pouvons effectuer des améliorations techniques :

- 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.
- Créer un fichier de configuration du routage et un fichier de configuration de la connexion à la base de données pour isoler les configurations et ainsi en faciliter la modification.
- Créer un « super-Contrôleur » qui regroupera les fonctions communes à tous les Contrôleurs. En faire de même avec un « super-Modèle ».
- Transformer les Modèles (et le « super-Modèle ») et les Contrôleurs (et le « super-Contrôleur ») en objets.

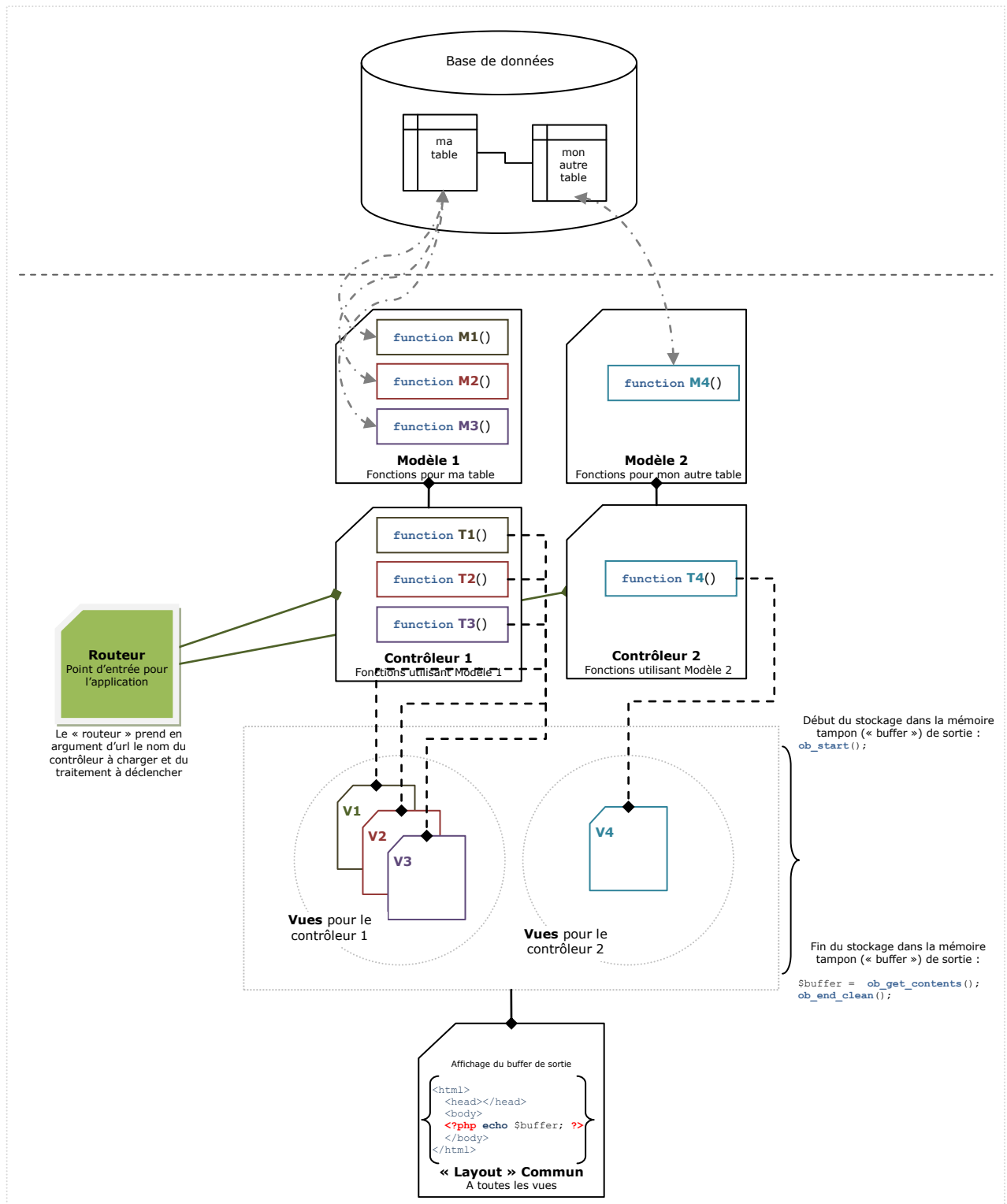
- **Organisation basique du système de fichier MVC**



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.

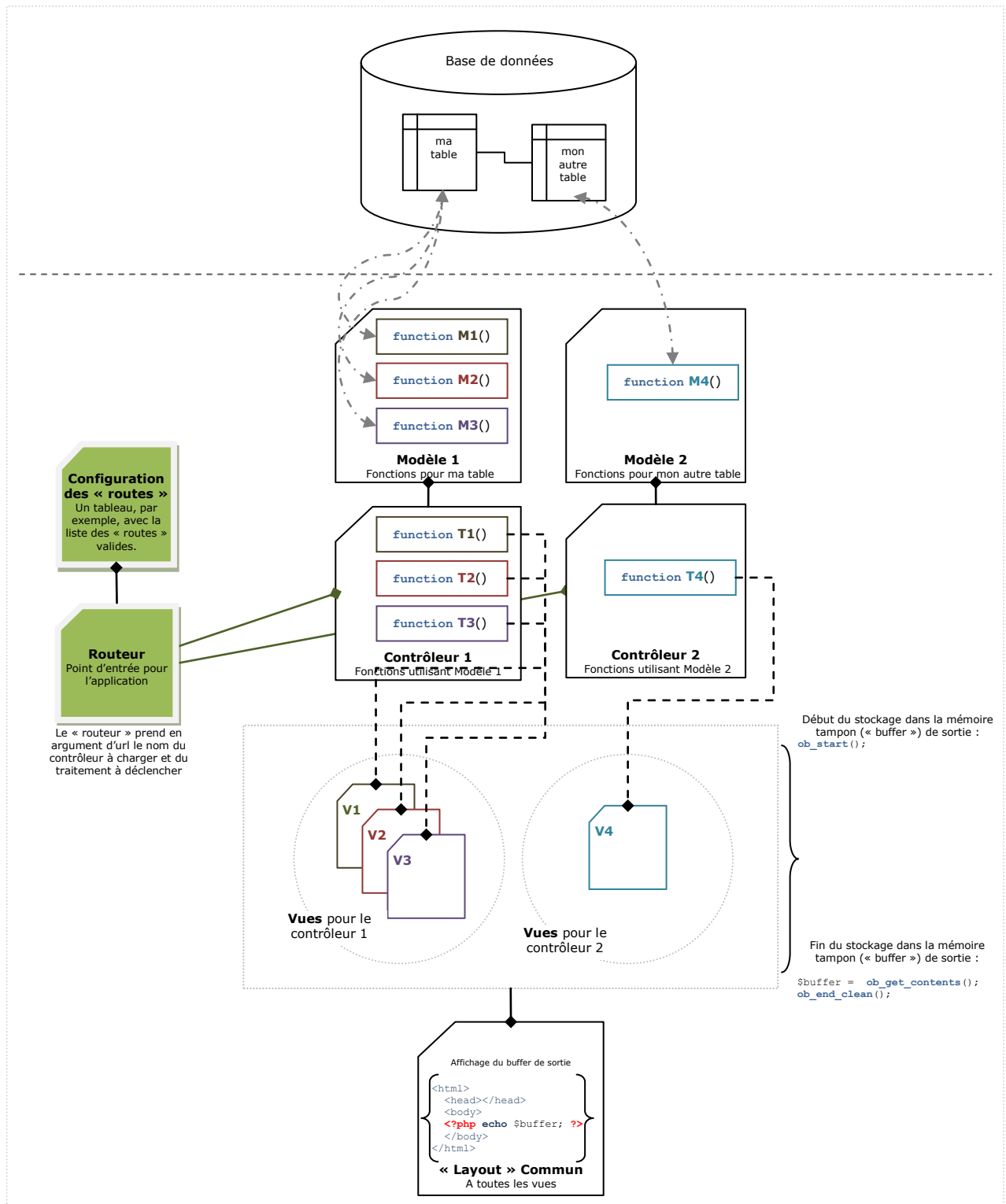
- **Optimisations naturelles**

- Création d'un « layout » commun



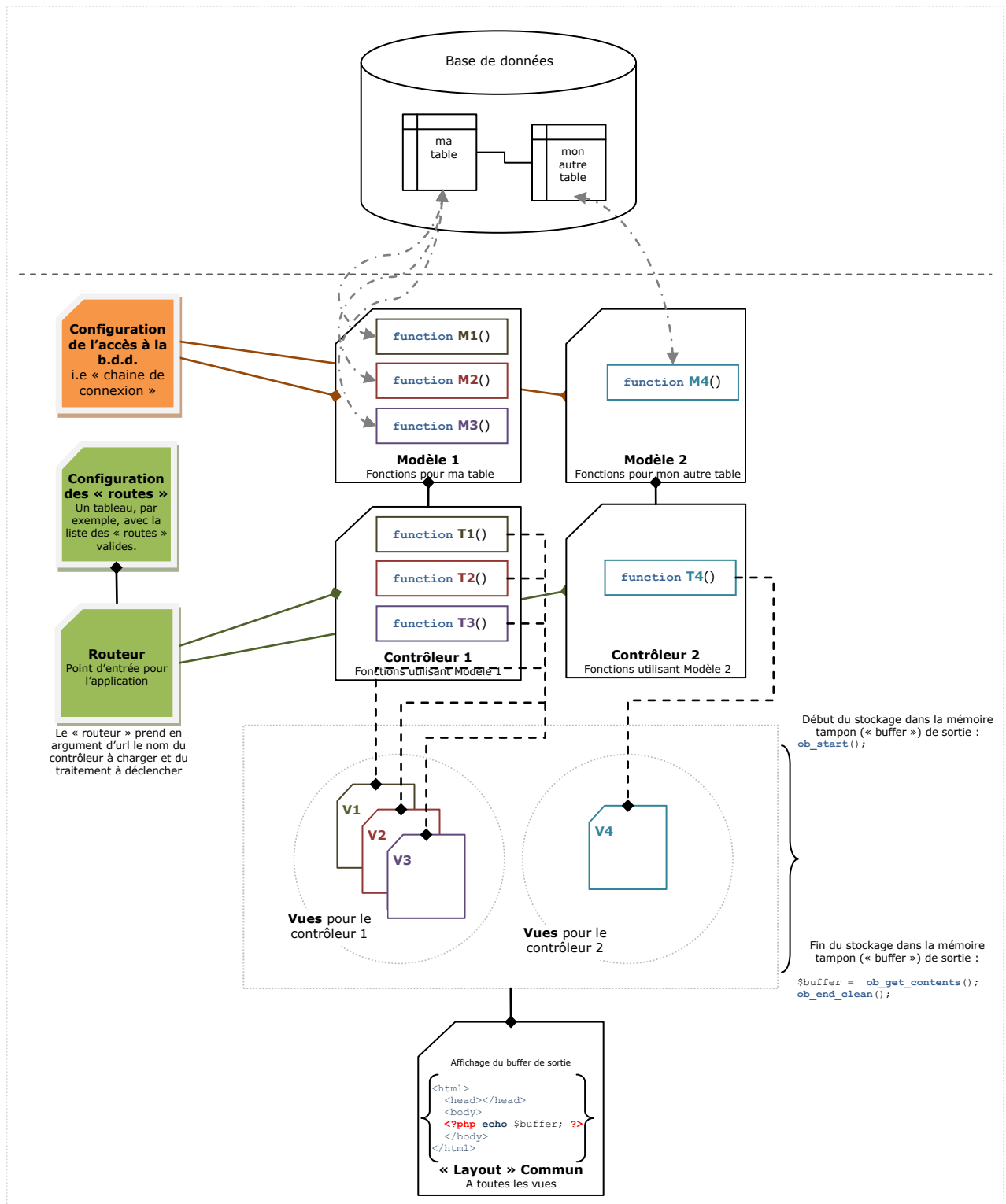
Pour préserver la structure « verticale » MVC on utilise les possibilités offertes par l'« output buffering » en PHP. On génère la Vue mais on ne la transmet pas au client, on l'enregistre dans la mémoire tampon (« buffer » de sortie). Puis, on génère le « layout » commun et on affiche au sein du « layout » commun le contenu de la mémoire tampon.

- Création d'un fichier pour la configuration du routage des URLs



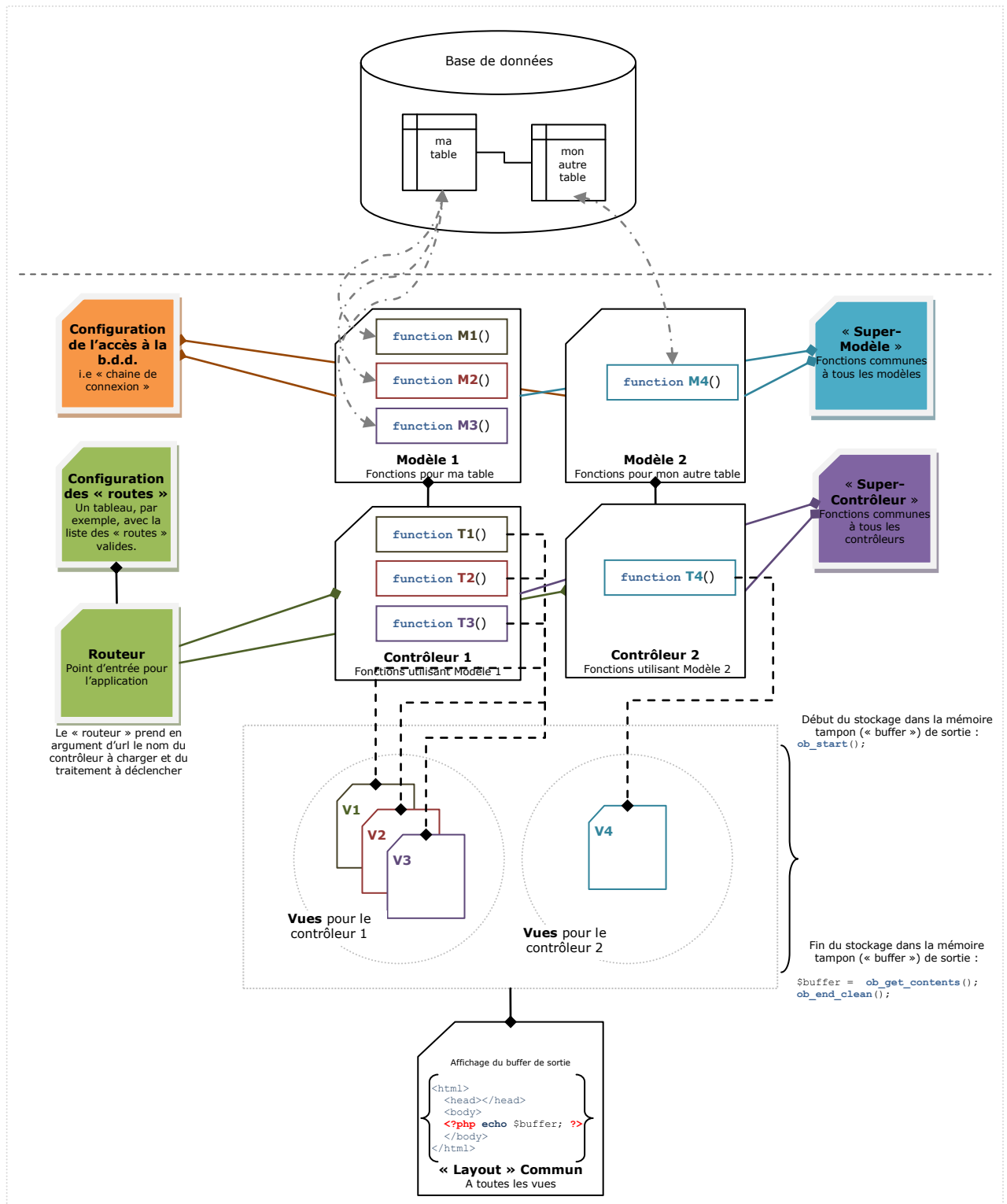
On crée un fichier PHP dans lequel on essaie de simplifier au maximum la configuration des routes. On déclare, par exemple, un tableau (array). Dans ce tableau chaque index est la route d'un contrôleur et contient un tableau pour lequel chaque index est la route d'une action et contient un tableau contenant 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.

- Création d'un fichier pour la configuration des accès 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.

- Création d'un « super-modèle » et création d'un « super-contrôleur »



On crée 2 fichiers PHP. Le premier sera utilisé par tous les Modèles. Le second par tous les Contrôleurs.

- Passage à la méthode objet

Ibid.