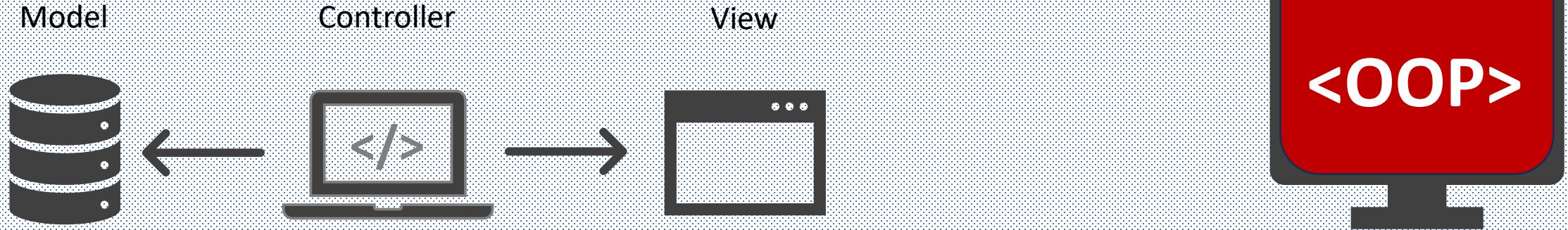


Développement WEB

Modèle – Vue- Contrôleur

Découverte de l'architecture MVC

PHP MVC



Évoluer vers une architecture PHP professionnelle

MVC – POO – Repositories – Exceptions - Namespaces

Contextualiser la problématique

État de l'art du code actuel

Mise en place du contexte pour la séance

Nous allons partir d'un développement tel que nous l'avons vu jusqu'à présent

Ce développement se base sur un sujet spécifique que nous allons réaliser ensemble tout au long de la séance

A chaque étape, nous allons enrichir notre application et aborder une nouvelle notion

Mise en place du contexte pour la séance

Présentation du projet

L'objectif est de réaliser une petite application web qui récupère des pays depuis une base de données afin de la afficher à l'écran

Liste des pays

France

Espagne

Italie

Maroc

Allemagne

Japon

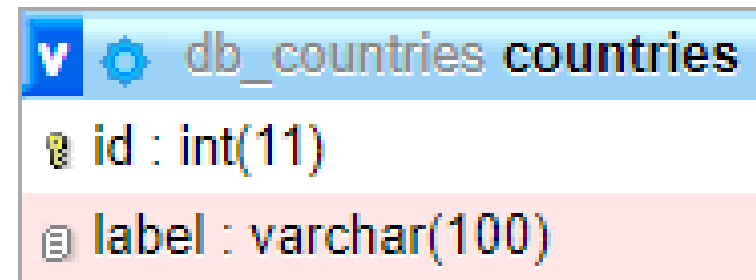
Mise en place du contexte pour la séance

Modèle de données

Notre base de données contiendra une table ***countries***

Cette table contiendra 2 colonnes :

- ***id*** : l'identifiant du pays
- ***Label*** : le nom du pays



Mise en place du contexte pour la séance

Étapes de réalisation

- Créer la base de données
 - Ecrire la requête de récupération des pays depuis la base de données
 - Afficher les pays sur la page ***index.php***
 - Mettre un peu de style pour la mise en forme
-

Mise en place du contexte pour la séance

Code réalisé

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Countries P00 Exercice</title>
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <main>
    <h1>Liste des pays</h1>
    <div class="countriesList">
      <?php
      try {
        $db = new PDO('mysql:host=localhost; dbname=db_countries; port=3306; charset=utf8', 'root', '');
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
      } catch (Exception $e) {
        die('Erreur : ' . $e->getMessage());
      }
      $req = $db->prepare('SELECT * FROM countries');
      $req->execute();
      while ($country = $req->fetch()) { ??
        <a href="#" class="btn"><?= $country['label'] ?></a>
      <?php
      }
      $req->closeCursor();
      ?>
    </div>
  </main>
</body>
</html>
```

} HTML

} HTML

} PHP

} SQL

} HTML

} PHP

} HTML

Mise en place du contexte pour la séance

Points négatifs

- Mélange de code HTML, PHP et SQL
- Structure monolithique (un seul bloc de code)

→ **Difficilement maintenable et non réutilisable**

Mise en place du contexte pour la séance

Quelles améliorations apporter ?

- isoler l'affichage
- isoler l'accès aux données

→ On réalise ce que l'on appelle **la séparation des responsabilités**

Séparer les responsabilités

Isoler l'affichage et l'accès aux données

En quoi cela consiste ?

Quelles améliorations apporter ?

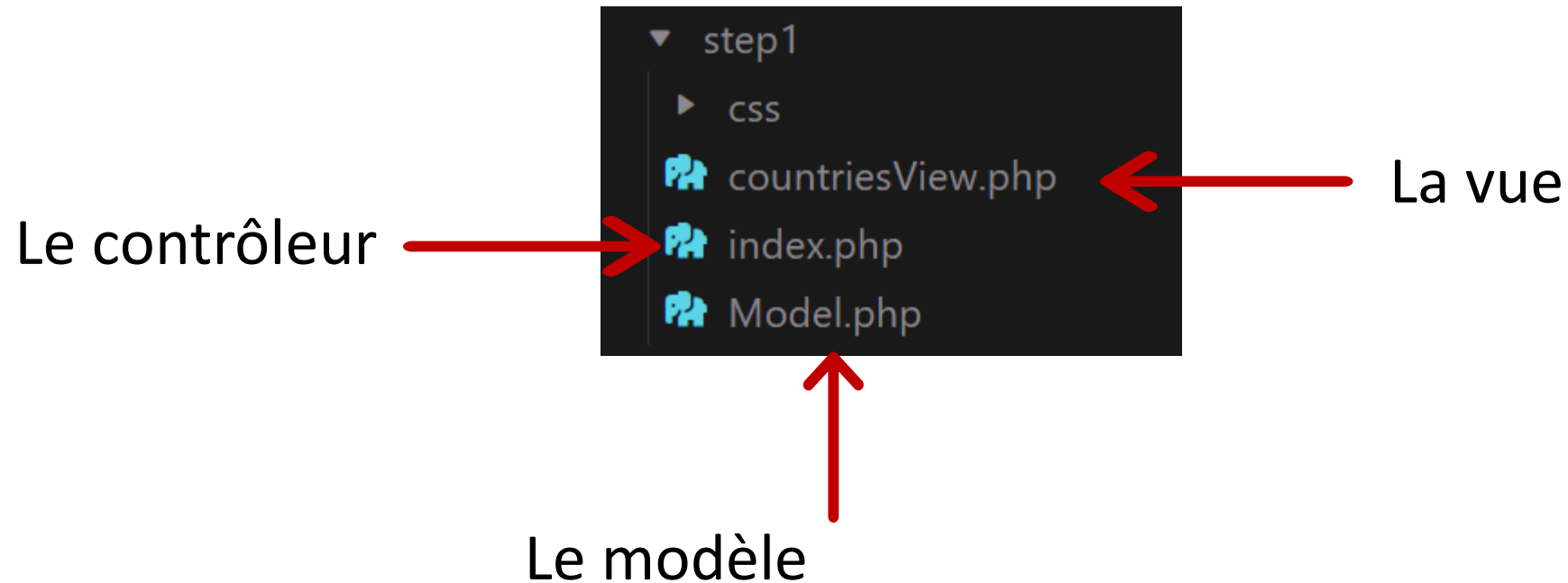
- La séparation des responsabilités consiste à isoler l'accès aux données et l'affichage
 - On va donc réaliser les manipulations de données et l'affichage dans des fichiers indépendants
-

Exercice

- Créer un fichier ***Model.php***, qui va contenir la fonction ***getCountries()***
- Cette fonction aura pour rôle de se connecter à la base de données et de retourner le résultat de la requête SQL sur la table ***countries***
- Créer un fichier ***countriesView.php*** qui va permettre d'afficher les données retournées
- Enfin modifier le fichier ***index.php*** en conséquence

Structure obtenue après séparation

Application de l'architecture MVC



Architecture MVC

Modèle – Vue - Contrôleur

Présentation

Le modèle MVC décrit une manière d'architecturer une application informatique en la décomposant en 3 sous-parties

- La partie ***Modèle***
 - La partie ***Vue***
 - La partie ***Contrôleur***
-

Présentation

Ce patron de conception (design pattern) permet de bien séparer le code de l'interface graphique de celui de la logique applicative

Présentation de concepts d'architecture logiciel :

<https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>

[https://fr.wikipedia.org/wiki/Architecture en couches](https://fr.wikipedia.org/wiki/Architecture_en_couches)

[https://fr.wikipedia.org/wiki/Architecture trois tiers](https://fr.wikipedia.org/wiki/Architecture_trois_tiers)

Rôle des composants

La partie Modèle d'une architecture MVC encapsule la logique métier (*business logic*) ainsi que l'accès aux données. Il peut s'agir d'un ensemble de fonctions (modèle procédural) ou de classes (modèle orienté objet)

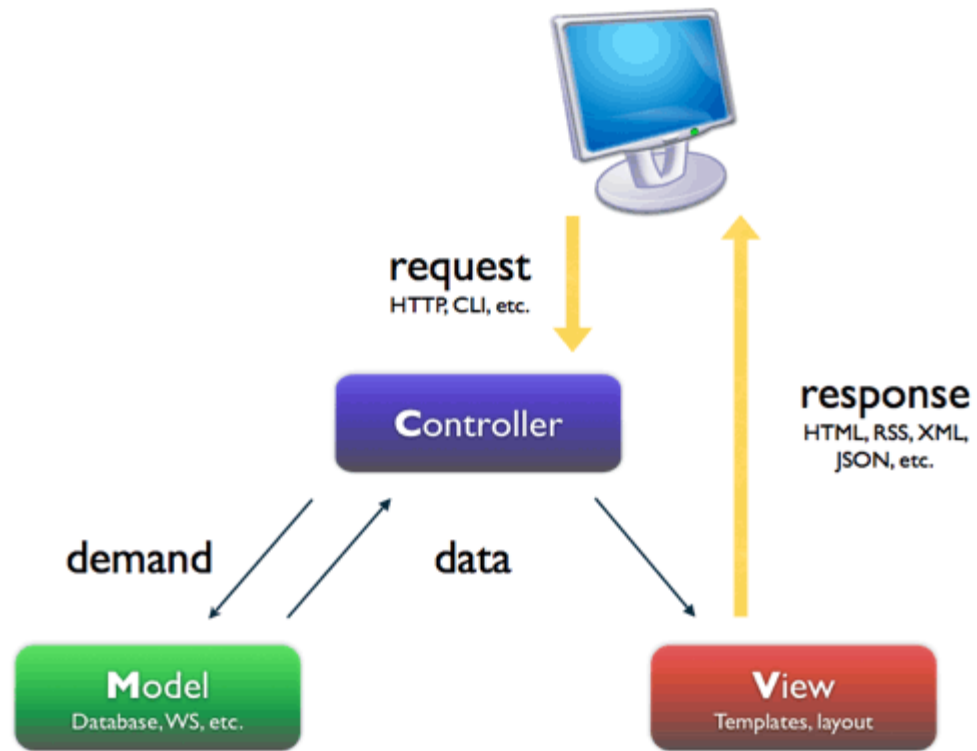
Rôle des composants

La partie **Vue** s'occupe des interactions avec l'utilisateur présentation, saisie et validation des données

Rôle des composants

La partie **Contrôleur** gère la dynamique de l'application
Elle fait le lien entre l'utilisateur et le reste de l'application

Interaction entre les composants



Amélioration du code :

Acte 1

Respect des standards de développement

Rappel des standards à respecter

- Langue de développement : anglais
 - Pas de balise de fermeture PHP dans un fichier contenant exclusivement du PHP
 - Commenter son code
 - Documenter son code
-

Supprimer les balise fermantes PHP

Lorsque le fichier « **.php** » contient uniquement du code PHP, il est recommandé de ne pas fermer la balise « **?>** »

Cela permet d'éviter un envoi de code HTML par erreur notamment en cas d'inclusion de fichiers PHP

Commenter son code

Un code a beau être bien rédigé, il doit être commenté En effet, un code bien rédigé sera compréhensible par tous développeur.

Le commentaire permettra de comprendre la logique d'ajout du code en question, dont la cause peut être aussi bien fonctionnelle que technique

Documenter son code

Au-delà de commenter son code, il est également important de bien le documenter.

Cela permettra par la suite une meilleure maintenabilité, et permettra aussi d'éditer une documentation qui pourra contribuer à la montée en compétences technique d'un nouveau développeur.

Ajout d'une nouvelle fonctionnalité

Consulter les détails d'un pays

Présentation de la fonctionnalité

Avant de découvrir d'autres notions, nous allons enrichir notre application et lui ajouter une nouvelle fonctionnalité la consultation de détails pour chaque pays.

Liste des pays

France Afficher les détails	Espagne Afficher les détails	Italie Afficher les détails	Maroc Afficher les détails	Allemagne Afficher les détails
		Japon Afficher les détails		

Présentation de la fonctionnalité

Pour chaque pays, on pourra consulter un certain nombre d'informations présentes en base de données.

[Retour](#)

Espagne

Superficie

506 030 km²

PIB

1,427 billion USD (2021)

Population

47,42 millions (2021)

Régions

17

Langue(s)

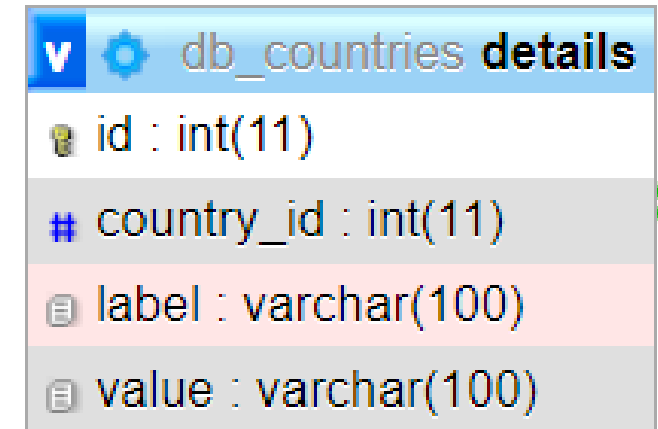
le castillan, le catalan, le basque et le galicien

Les étapes de réalisation

Ajouter une table « **details** » base de données.

Cette table contiendra 4 colonnes :

- **id** : l'identifiant du détail
- **country_id** : l'identifiant du pays concerné
- **label** : le libellé du détail
- **value** : la valeur du détail



Les étapes de réalisation

Modification du modèle

Pour pouvoir accéder aux données de cette nouvelle table, nous devons ajouter une fonction **getDetails()** dans notre **model.php**

```
// Requête pour les détails d'un pays  
function getDetails($countryId)  
> { ...  
}
```

Les étapes de réalisation

Créer la vue

Pour afficher les détails d'un pays, nous devons disposer d'une vue qui sera dédiée à cette tâche. Nous la nommerons « **detailsView.php** ».

Cette vue affichera le nom du pays concerné et la liste de ses détails.

Le titre de la page sera « Détails ».

Un bouton « Retour » nous permettra de revenir à la liste des pays.

Les étapes de réalisation

Créer un contrôleur spécifique

Nous allons créer un contrôleur dédié à la gestion des détails d'un pays que nous appellerons « **details.php** ».

Il fera appel au modèle « **model.php** », vérifiera la présence de l'ID du pays passé en paramètre pour lequel on souhaite consulter les détails, récupèrera le pays concerné ainsi que ses détails puis fera appel à la vue « **detailsView.php** » que nous venons de créer.

Les étapes de réalisation

Modifier la vue *countriesView.php*

Nous allons également modifier la vue « **countriesView.php** » afin d'y insérer un lien « Afficher les détails » sur chacun des pays qui nous redirigera vers la page de détails de chaque pays.

```
<a href=<?= "details.php?countryId=".$country['id']?>> Afficher les détails </a>
```

Création d'un routeur

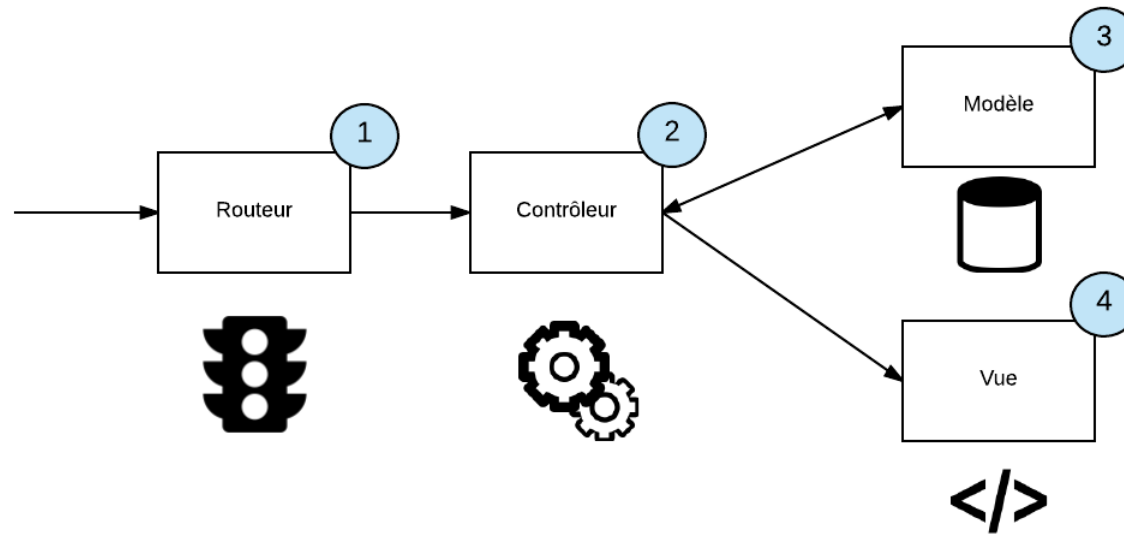
Routage des requêtes

Qu'est-ce qu'un routeur ?

Un **routeur** peut être assimilé à un **contrôleur « frontal »**. Son objectif sera d'appeler le bon contrôleur selon la requête réalisée par l'utilisateur. On dit qu'il « **route** » les requêtes, d'où son nom de « **routeur** ».

Qu'est-ce qu'un routeur ?

Voici un schéma de notre architecture MVC avec la mise en place de notre routeur



Pourquoi utiliser un routeur ?

La mise en place d'un routeur n'est pas obligatoire en soi.

Actuellement, nous avons deux vues, et deux contrôleurs associés :

- **Index.php** qui est le contrôleur pour la vue **countriesView.php**
 - **Details.php** qui est le contrôleur pour la vue **detailsView.php**
-

Pourquoi utiliser un routeur ?

Cela fonctionne très bien en l'état.

Toutefois, si l'on garde le système actuel, on risque vite de se retrouver avec un fichier par page de notre site.

De plus, si des informations communes seraient à charger pour deux vues différentes, nous devrions dupliquer le code

Mise en place du routeur ?

Regrouper les contrôleurs

Nous allons commencer par regrouper nos deux contrôleurs **index.php** et **details.php** dans deux fonctions au sein d'un même fichier que l'on nommera « **controller.php** ».

On peut donc supprimer le fichier **details.php** qui ne nous est plus utile désormais.

Mise en place du routeur ?

Création du routeur

Ensuite, nous allons insérer le contenu de notre routeur dans le fichier **index.php**

Ce routeur fera appel à nos différents contrôleurs (ici uniquement **controller.php** pour le moment) et recevra systématiquement un paramètre « **action** » qui permettra de réaliser le traitement adéquate

Mise en place du routeur ?

Création du routeur

Définissez le code pour **index.php** et pour **controller.php**

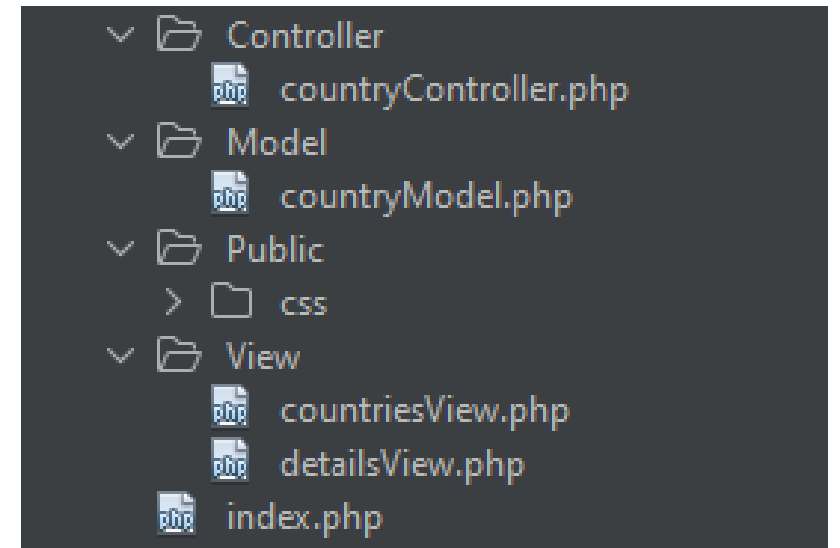
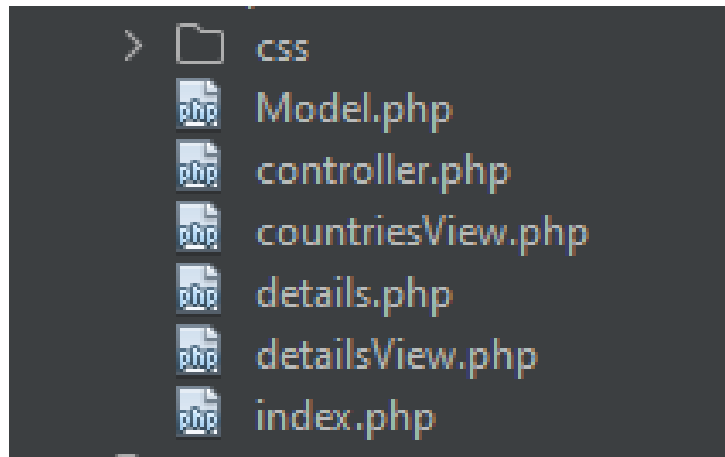
Amélioration du code :

Acte 2

Organisation des fichiers dans des dossiers

Structurer pour mieux s'y retrouver

Toujours dans l'optique de mieux organiser notre code, nous allons créer une arborescence afin de classer nos fichiers



Structurer pour mieux s'y retrouver

On retrouve très fréquemment ce type d'arborescence :



controller/ contient l'ensemble de nos contrôleurs



model/ contient l'ensemble de nos modèles



public/ contient l'ensemble des fichiers statiques (js, etc...)
classés en sous dossiers (**css/ js/ images/** etc...)



vendor/ contient les bibliothèques tierces



views/ contient l'ensemble de nos vues

Adapter le code

Comme les fichiers ont été déplacés dans différents dossiers, il ne faut pas oublier d'ajuster les inclusions (feuilles de style, modèles, vues, etc...)

Profitez de cette étape pour renommer les fichiers « **controller.php** » et « **model.php** » respectivement en « **countryController.php** » et « **countryModel.php** »

Ajout d'une nouvelle fonctionnalité

Ajouter une information dans le détail d'un pays

Présentation de la fonctionnalité

Nous allons à nouveau enrichir notre application en ajoutant une nouvelle fonctionnalité l'ajout d'informations pour un pays.

[Retour](#)

Espagne

Ajouter une information pour ce pays

Clé : Valeur :

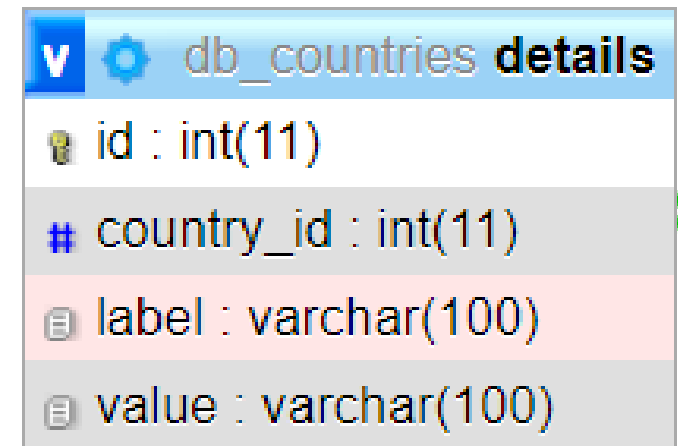
Superficie 506 030 km ²	PIB 1,427 billion USD (2021)	Population 47,42 millions (2021)	Régions 17	Langue(s) le castillan, le catalan, le basque et le galicien
--	--	--	----------------------	--

Les étapes de la réalisation

Enrichir le modèle

La première étape consiste à créer une nouvelle fonction dans le fichier « **countryModel.php** » afin d'insérer des données dans la table « **details** ».

Pour rappel, voici les attributs de cette table :



The image shows a screenshot of a database table structure for a table named 'details' in a database named 'db_countries'. The table has four columns: 'id' (integer, 11 digits), 'country_id' (integer, 11 digits), 'label' (variable character string, 100 characters), and 'value' (variable character string, 100 characters). The 'id' column is marked as the primary key with a key icon. The 'country_id' column is marked with a hash icon, indicating a foreign key. The 'label' and 'value' columns are marked with document icons. The table name 'details' is highlighted in blue in the header.

db_countries details
id : int(11)
country_id : int(11)
label : varchar(100)
value : varchar(100)

Les étapes de la réalisation

Enrichir le modèle

Création de la fonction **saveCountryDetail()** dans le fichier « **countryModel.php** » afin d'insérer des données dans la table « **details** ».

```
// Insertion de données pour un pays
function saveCountryDetail($countryId, $label, $value)
{ ...10 lines }
```

Les étapes de la réalisation

Enrichir le contrôleur

Il faut créer la fonction suivante dans « **countryController.php** »

```
// Ajouter des détails à un pays
function addCountryDetail($countryId, $label, $value)
{
    $result = saveCountryDetail($countryId, $label, $value);

    if ($result == false) {
        die("Erreur : impossible d'ajouter le nouveau détail");
    } else {
        header('Location: index.php?action=showCountryDetails&countryId=' . $countryId);
    }
}
```

Les étapes de la réalisation

Modification de la vue « detailView.php »

De même, nous devons ajouter un formulaire de saisie pour les informations :

```
<h2>Ajouter une information pour ce pays</h2>
<form action="index.php?action=addCountryDetail&countryId=<?= $country['id'] ?>" method="post">
  <label for="label">Clé :</label>
  <input type="text" name="label">
  <label for="value">Valeur :</label>
  <input type="text" name="value">
  <input type="submit" name="submitBtn" value="Ajouter">
</form>
```

Les étapes de la réalisation

Enrichir le routeur

La dernière étape consiste à mettre à jour notre routeur afin qu'il puisse faire appel à notre contrôleur lors de la soumission du formulaire d'ajout.

Il sera chargé de vérifier la présence de l'id du pays, de la clé ainsi que de la valeur de l'information à ajouter.

Les étapes de la réalisation

Enrichir le routeur

Le code à ajouter :

```
} elseif ($_GET['action'] == 'addCountryDetail') {  
    if (isset($_GET['countryId']) && $_GET['countryId'] > 0) {  
        if (isset($_POST['label'])) {  
            if (isset($_POST['value'])) {  
                addCountryDetail($_GET['countryId'], $_POST['label'], $_POST['value']);  
            } else {  
                echo "Erreur : Valeur non définie";  
            }  
        } else {  
            echo "Erreur : Clé non valide";  
        }  
    } else {  
        echo "Erreur : Identifiant de pays non valide";  
    }  
}
```

Gestion des exceptions

Gérer les erreurs dans le code

Gérer les erreurs dans une application

La gestion des erreurs est un sujet important en programmation

Malgré la qualité des développements et les différents tests qui sont réalisés, il arrive parfois que des erreurs surgissent et il vaut mieux les avoir anticipées au maximum.

Gérer les erreurs dans une application

Si l'on reprend notre routeur, on constate de nombreux **if...else**.

Ils ont pour but de réaliser des contrôles afin de s'assurer, par exemple, de la présence des paramètres requis.

De cette façon, le code de notre routeur gère d'une certaine manière les erreurs qui lui sont propres.

Gérer les erreurs dans une application

En revanche, si une erreur survient dans l'une des fonctions appelées par le routeur, par exemple dans le contrôleur ou le modèle, que se passe-t-il ?

Gérer les erreurs dans une application

On pourrait appliquer le même principe que pour le routeur, mais cela présente deux inconvénients :

- Le code deviendrait vite complexe et la duplication augmenterait
 - Ce n'est pas le rôle du contrôleur ni du modèle de gérer les erreurs
-

Gérer les erreurs dans une application

Heureusement pour nous, on a une façon plus propre et générique de gérer les erreurs dans le code : utiliser les **exceptions**

Qu'est-ce qu'une exception ?

Voici à quoi ressemble un bloc de gestion d'exception

```
<?php
try{
    // Essayer de faire qqch
}
catch(Exception $e){
    // Si une erreur se produit on arrive ici
}
```

Le traitement se réalise dans le bloc **try**

Les erreurs sont gérées dans le bloc **catch** . S'il n'y a pas d'erreurs, le bloc **catch** n'est pas interprété.

Qu'est-ce qu'une exception ?

Pour générer une exception, on doit « jeter une exception » (throw exception en anglais)

```
throw new Exception("Message d'erreur à transmettre");
```

Qu'est-ce qu'une exception ?

En réalité, on a déjà utilisé ce principe depuis le début du module sans forcément y prêter attention avec la connexion à la base de données

```
<?php
try{
    $db = new PDO('mysql:host=localhost;dbname=essai', 'root', '');
}
catch(Exception $e){
    die("Erreur : ".$e->getMessage());
}
```

Obtenir du détail sur l'exception ?

Dans le cas où une erreur se produit et que l'on passe par le bloc **catch** on peut récupérer l'exception en question et afficher des détails.

On utilise pour cela l'objet PHP « **Exception** »

Obtenir du détail sur l'exception ?

Voici les informations que l'on peut récupérer pour une **Exception \$e**

\$e->getMessage -> Récupère le message de l'exception

\$e->getCode -> Récupère le code de l'exception

\$e->getFile -> Récupère le fichier dans lequel l'exception a été créée

\$e->getLine -> Récupère la ligne dans laquelle l'exception a été créée

\$e->getTrace -> Récupère la trace de la pile

Traitement post-exécution

A noter également que l'on peut insérer un bloc **finally** après des blocs **catch**

Le code à l'intérieur du bloc **finally** sera toujours exécuté après les blocs **try** et **catch** indépendamment du fait qu'une exception ait été lancée, avant de continuer l'exécution normale

Traitement post-exécution



Si une déclaration **return** est rencontrée à l'intérieur des blocs **try** ou **catch** le bloc **finally** sera quand même exécuté.

De plus, la déclaration **return** est évaluée quand elle est rencontrée, mais le résultat sera retourné après le bloc **finally** soit exécuté.

Enfin, si le bloc **finally** contient aussi une déclaration **return** la valeur du bloc **finally** est retournée.

Gérer les exceptions dans le routeur

On réalise le traitement du routeur dans un bloc **try**

```
try { //code...
    if (isset($_GET['action'])) {
        if ($_GET['action'] == 'showCountryDetails') {
            if (isset($_GET['countryId']) && $_GET['countryId'] > 0) {
                showCountriesDetails($_GET['countryId']);
            } else {
                throw new Exception("identifiant non valide");
            }
        } elseif ($_GET['action'] == 'addCountryDetail') {
            if (isset($_GET['countryId']) && $_GET['countryId'] > 0) {
                if (isset($_POST['label'])) {
                    if (isset($_POST['value'])) {
                        addCountryDetail($_GET['countryId'], $_POST['label'], $_POST['value']);
                    } else {
                        throw new Exception("Erreur : Valeur non définie");
                    }
                } else {
                    throw new Exception("Erreur : Clé non valide");
                }
            } else {
                throw new Exception("Erreur : Identifiant de pays non valide");
            }
        } else {
            throw new Exception("Erreur : action non valide");
        }
    } else {
        showCountries();
    }
}
```

Gérer les exceptions dans le routeur

On génère une exception pour chaque cas d'erreur dans notre routeur avec un message permettant de comprendre l'origine du problème.

```
try { //code...
    if (isset($_GET['action'])) {
        if ($_GET['action'] == 'showCountryDetails') {
            if (isset($_GET['countryId']) && $_GET['countryId'] > 0) {
                showCountriesDetails($_GET['countryId']);
            } else {
                throw new Exception("identifiant non valide");
            }
        } elseif ($_GET['action'] == 'addCountryDetail') {
            if (isset($_GET['countryId']) && $_GET['countryId'] > 0) {
                if (isset($_POST['label'])) {
                    if (isset($_POST['value'])) {
                        addCountryDetail($_GET['countryId'], $_POST['label'], $_POST['value']);
                    } else {
                        throw new Exception("Erreur : Valeur non définie");
                    }
                } else {
                    throw new Exception("Erreur : Clé non valide");
                }
            } else {
                throw new Exception("Erreur : Identifiant de pays non valide");
            }
        } else {
            throw new Exception("Erreur : action non valide");
        }
    } else {
        showCountries();
    }
}
```

Gérer les exceptions dans le routeur

Dans un bloc catch, on récupère l'exception en cas d'erreur ainsi que certains détails.

```
} catch (Exception $e) {  
    $error = $e->getMessage();  
    $file = basename($e->getFile());  
    $line = $e->getLine();  
  
    require('View/errorView.php');  
}
```

On appelle ensuite une vue « **errorView.php** » dédiée à l'affichage des erreurs.

Gérer les exceptions dans le routeur

Voici le résultat obtenu en cas d'erreur dans le code de notre application :

Oups !!! une erreur imprévue est survenue

Détail de l'erreur :

Erreur : action non valide **[index.php, 28]**

[Revenir à la page d'accueil](#)

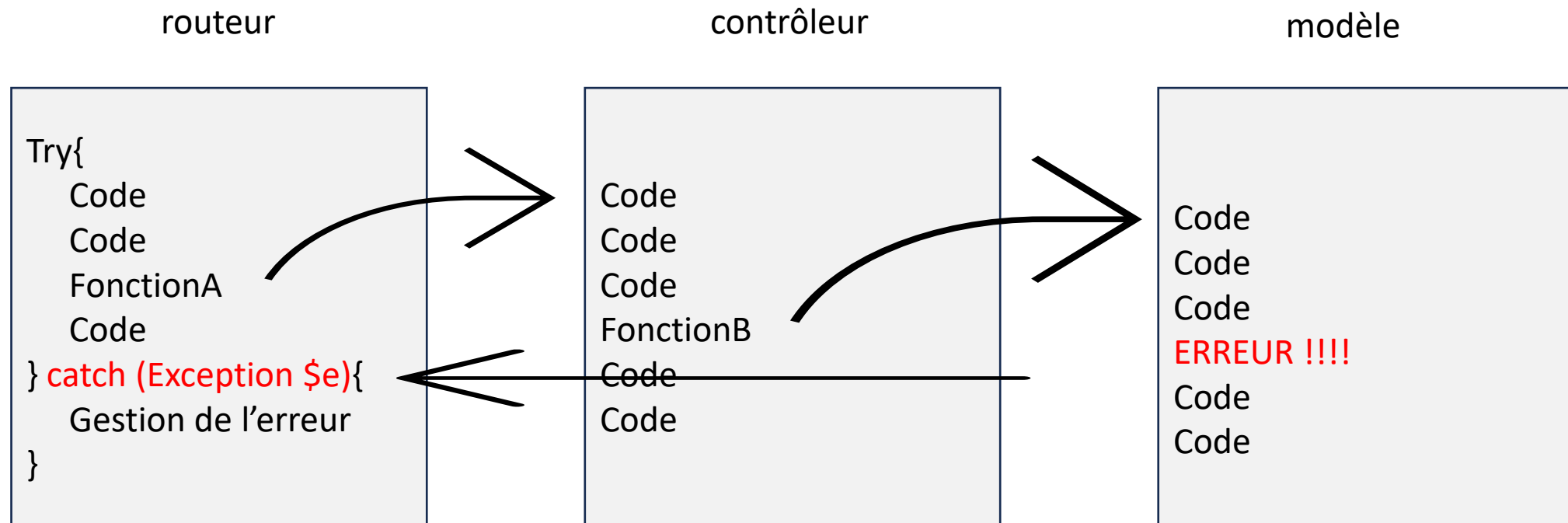
Remonter les exceptions

Jusque-là, on a simplement amélioré notre système de gestion des erreurs en remplaçant les **if...else** par des blocs **try / catch** avec des exceptions.

Un des principaux avantages de l'usage des exceptions est la remontée des exceptions

Remonter les exceptions

En effet, si une erreur survient dans une fonction appelée dans un bloc **try** l'exception est directement remontée jusqu'au bloc **catch** correspondant.



Remonter les exceptions

On peut ainsi gérer les exceptions dans nos contrôleurs et nos modèles :

```
// Ajouter des détails à un pays
function addCountryDetail($countryId, $label, $value)
{
    $result = saveCountryDetail($countryId, $label, $value);

    if ($result == false) {
        throw new Exception("Erreur : impossible d'ajouter le nouveau détail");
    } else {
        header('Location: index.php?action=showCountryDetails&countryId=' . $countryId);
    }
}
```

Remonter les exceptions

De même, le bloc **try / catch** de notre fonction « **connect** » dans notre modèle **model.php** n'est plus utile puisque le traitement est déjà encapsulé par un bloc **try / catch** dans le routeur.

```
// Connexion à la base de données
function connect()
{
    $db = new PDO('mysql:host=localhost; dbname=db_countries; port=3306; charset=utf8', 'root', '');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return $db;
}
```

POO

Programmation Orientée Objet