***L’architecture MVC en PHP :***

First, what is a « profesional » code ?

1) you can modulate it : every file has one and only one role

2) it is divided : each file can function without needing other files

3) it is documented : comments above public methods and classes/ comments should explain the « why » and not the « how »

4) it is in english

5) it is clear, you should be able to read your code like poetry and a file should never exeed a thousand lines …..

6) you can re use the code

7) it is easy to work with a team on the code

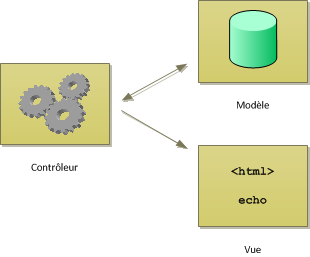
8) you can evolve the code meaning its easy to add a new functionality to it

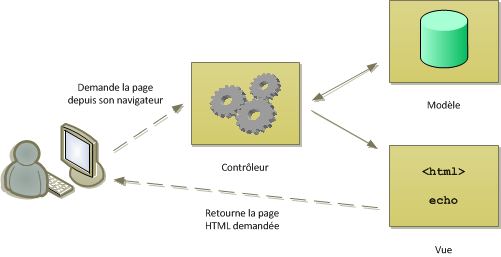
Actually MVC stands for : model ( which is treating the datas , for instance : model.php ) , ( vue ) which is displaying the informations ( acceuil.php for instane) and the controler which is making the link between the two ( here : index.php ).   
After, in order to be more accurate, here is a more detailed view of the MVC pattern, which will allow you to seperate the logic of your code in three part within three distinct files.

1) The model : this part handles the datas of your website, its role is to fetch the « raw informations » in your data base, to organize them and to assemble them so that they can be treated by the controler

2) the vue : this part is focusing on the « display ». It barely does calculus operations and is almost only there to fetch datas in order to know what it can display or not. There is mostly html code in it but also a few very simple loops and PHP conditions, to display, for instance, a message

3) the controller : this part handles the logic of the code and make decisions. Its the intermediary between the vew part and the model part. The controller will ask to the model the datas, analyse them, make decisions and return the text to display to the view part. The controller only contains PHP. It is its role to determine if the visitor has the right to see the page or not





For instance, heres an exemple of DRY when connecting to a database :

<?php

function getPosts()

{

$db = dbConnect();

$req = $db->query('SELECT id, title, content, DATE\_FORMAT(creation\_date, \'%d/%m/%Y à %Hh%imin%ss\') AS creation\_date\_fr FROM posts ORDER BY creation\_date DESC LIMIT 0, 5');

return $req;

}

function getPost($postId)

{

$db = dbConnect();

$req = $db->prepare('SELECT id, title, content, DATE\_FORMAT(creation\_date, \'%d/%m/%Y à %Hh%imin%ss\') AS creation\_date\_fr FROM posts WHERE id = ?');

$req->execute(array($postId));

$post = $req->fetch();

return $post;

}

function getComments($postId)

{

$db = dbConnect();

$comments = $db->prepare('SELECT id, author, comment, DATE\_FORMAT(comment\_date, \'%d/%m/%Y à %Hh%imin%ss\') AS comment\_date\_fr FROM comments WHERE post\_id = ? ORDER BY comment\_date DESC');

$comments->execute(array($postId));

return $comments;

}

// Nouvelle fonction qui nous permet d'éviter de répéter du code

function dbConnect()

{

try

{

$db = new PDO('mysql:host=localhost;dbname=test;charset=utf8', 'root', 'root');

return $db;

}

catch(Exception $e)

{

die('Erreur : '.$e->getMessage());

}

}

Heres an exemple of a « php template » for the controler :

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title><?= $title ?></title>

<link href="style.css" rel="stylesheet" />

</head>

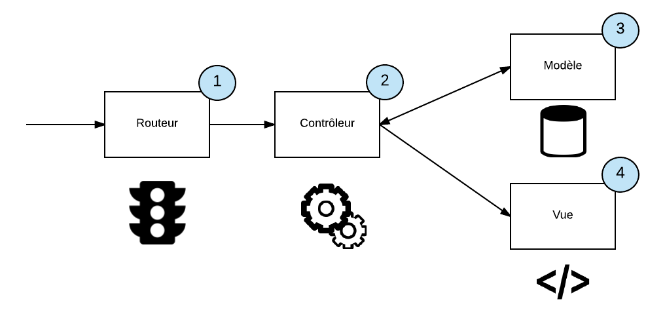
<body>

<?= $content ?>

</body>

</html>

 l'astuce du ob\_start() et ob\_get\_clean() (qui nous sert juste à mettre facilement beaucoup de code HTML dans une variable).   
  
Actualy, the prupose would be to easily put lots of html code in one variable.



The role of the routeur is to find the right controlleur. In general, the routeur is the first file you call when arriving on a website. It is also the role of the routeur to check if all the parameters are in the URL before charging the controler.

For the organization of the files, heres an exemple :

1) the controller is the file which holds the controlers … thanks captain Obvious.

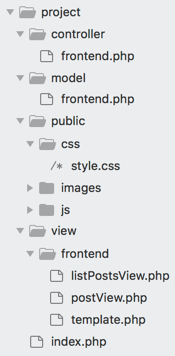
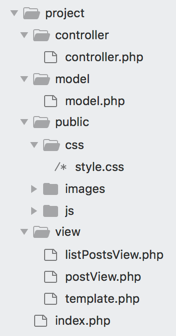
2) the view contains the views

3)the model contains the model

4) public contains all public static files, we can store there : a css file, images, js files ….

5) there is also often a « vendor » file in which we place all exeternal libraries.

A good exemple of a clean files organization would be :

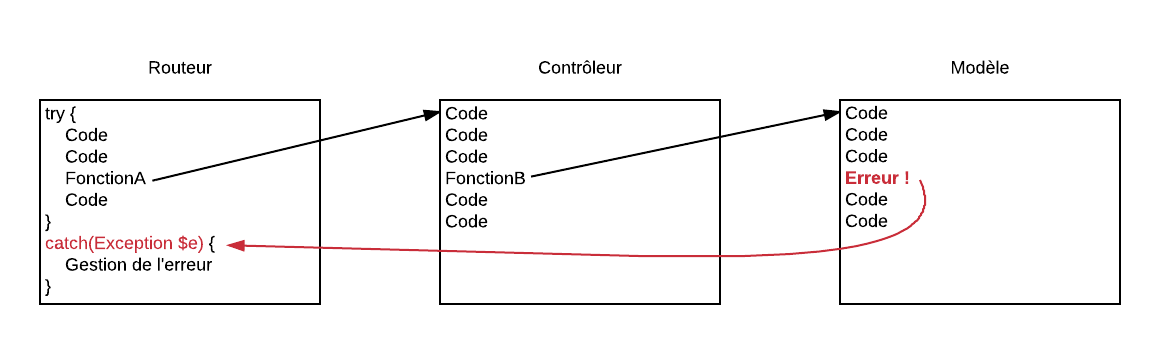


***How to handle errors and exeptions ?***

To generate an error, you have to « throw » an exeption. As soon as there is an error somewhere in your code, for instance in a function, you can use that line :

<?php

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



***OOP MVC***