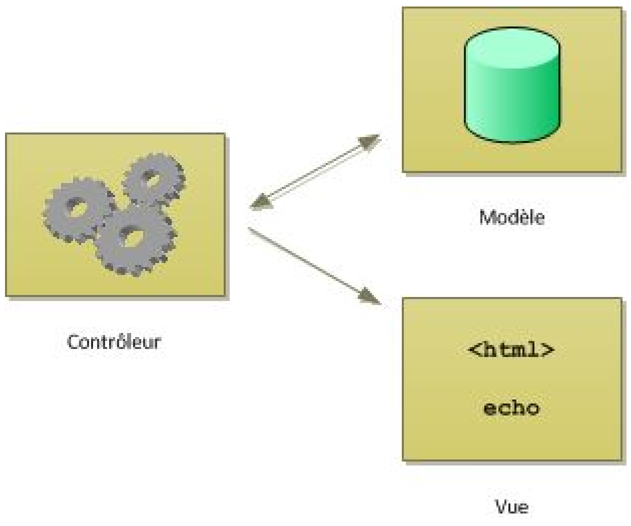
What is MVC?

One of the most famous *design patterns is* called MVC, which stands for **Model - View - Controller**. This is the one we are going to see today.

The MVC pattern makes it easy to organize your code. It will help you to know which files to create, but especially to define their role. The purpose of MVC is precisely to separate the logic of the code into three parts that are found in separate files.

- **Model** : This part manages the *data* of your site. Its role is to retrieve the "raw" information from the database, organize it, and assemble it so that it can be processed by the controller. We find there among others the SQL queries.
  BUSINESS LOGIC AND DATA ACCESS
- **View** : This part focuses on the*display*. It does almost no calculation and is content to retrieve variables to know what to display. There is mainly HTML code but also some very simple PHP loops and conditions, for example to display a list of messages.
- **Controller** : This part manages the logic of the code that makes *decisions*. It is sort of the intermediary between the model and the view: the controller will ask the model the data, analyze, make decisions and return the text to display in the view. The controller contains only PHP. It is in particular that it determines whether the visitor has the right to see the page or not (management of access rights).



1. The controller receives the request from the user.
2. The controller will use the model to request the data. He does not care about how he recovers them.
3. The model translates the controller request (usually SQL query), retrieves the information and returns it to the controller.
4. The controller will now be able to transmit this data to the view that will be sent to the user.

Advantages

- **It is reusable** : if one day we coded a useful file, we can use it again in another project, or in another place of the same project. We save time by not having to redo every time!
- **It is easy to work with several** : each file is independent (and usually small), we can work in teams of 5, 10 or 100 people on the same project. If everything was mixed in one big file, it would be impossible to modify it at the same time!
- **It's scalable** : when someone comes to ask you for a new feature, it's easy to add. You are not afraid of breaking everything. You know it's going to work, and your code will not be any more complicated.

Slight improvement in reading the code

For the rest,we are going to use a previous exercise, which shows a list of films.
The first iteration, the simplest is to remap the code even if remaining in a single page:

We put our part query in a function and at the top.

```
function getMovies () {
   $ bdd = new PDO ('mysql: host = localhost; dbname = moviedb; charset = utf8',
 'root', 'root');
   $ movies = $ bdd-> query ('SELECT * FROM Movies');
   return $ movies-> fetchAll (PDO :: FETCH_ASSOC) ;;
 }

 $ books = getBooks ();
```

Start Adopting MVC - The View

One will start by separating the view from the rest.
Remember, the view only deals with the display.

So we create a new file: **HomeView.php**
We put the HTML code.
In the index.php, we find there more than our function and a require for the display.

Continue to adopt MVC - The model

Now, we will separate the model
We create a new file: **MovieModel.php**

We place the function that accesses the data.

This new structure is more complex, but the responsibilities of each party are now clear. In doing this work of *refactoring*, we made our example conform to a model of architecture very used on the Web: the model **MVC**.

The controller

For now, we have a controller that manages the homepage (index.php).
We will create a second controller that will display the movies:
MovieController.php.

See the result in the Git directory.