Introduction au MVC

Cyril Rabat François Alin

Programmation Web - Php

2019-2020

Cours Info 303

Modèle Vue Controleur

Version 16 septembre 2019

Table des matières

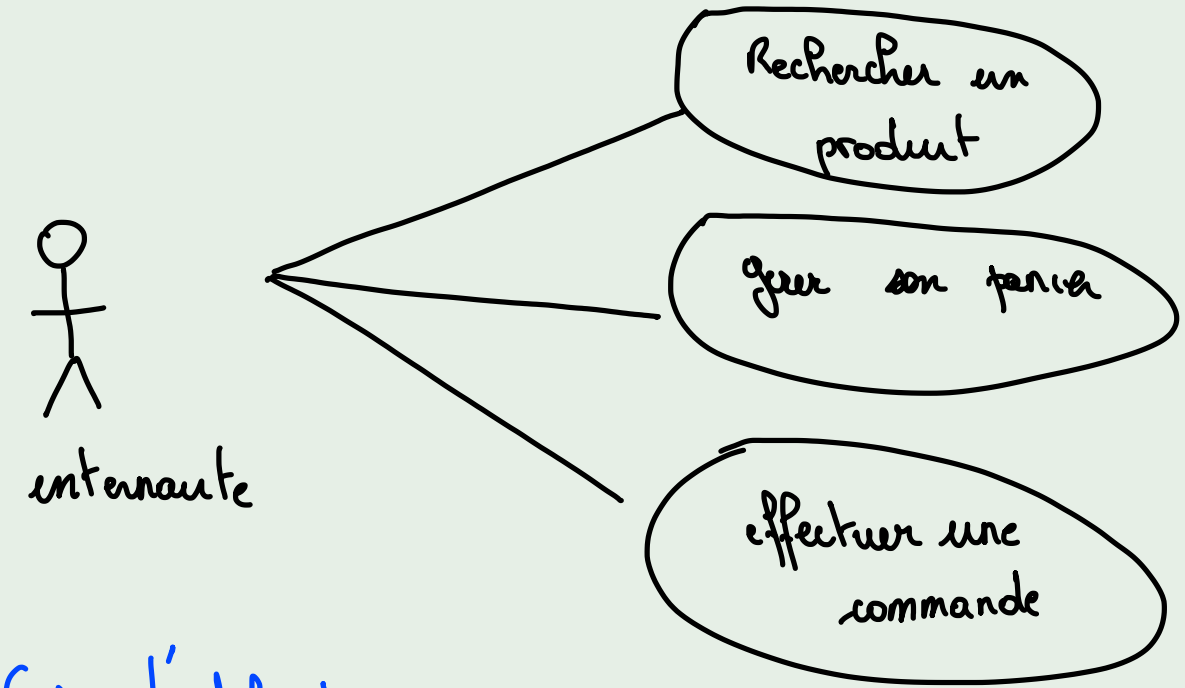
Table des matières

Le besoin utilisateur

- **Diagramme des cas d'utilisation :** Il montre les interactions fonctionnelles entre les acteurs et le système.

Le besoin utilisateur

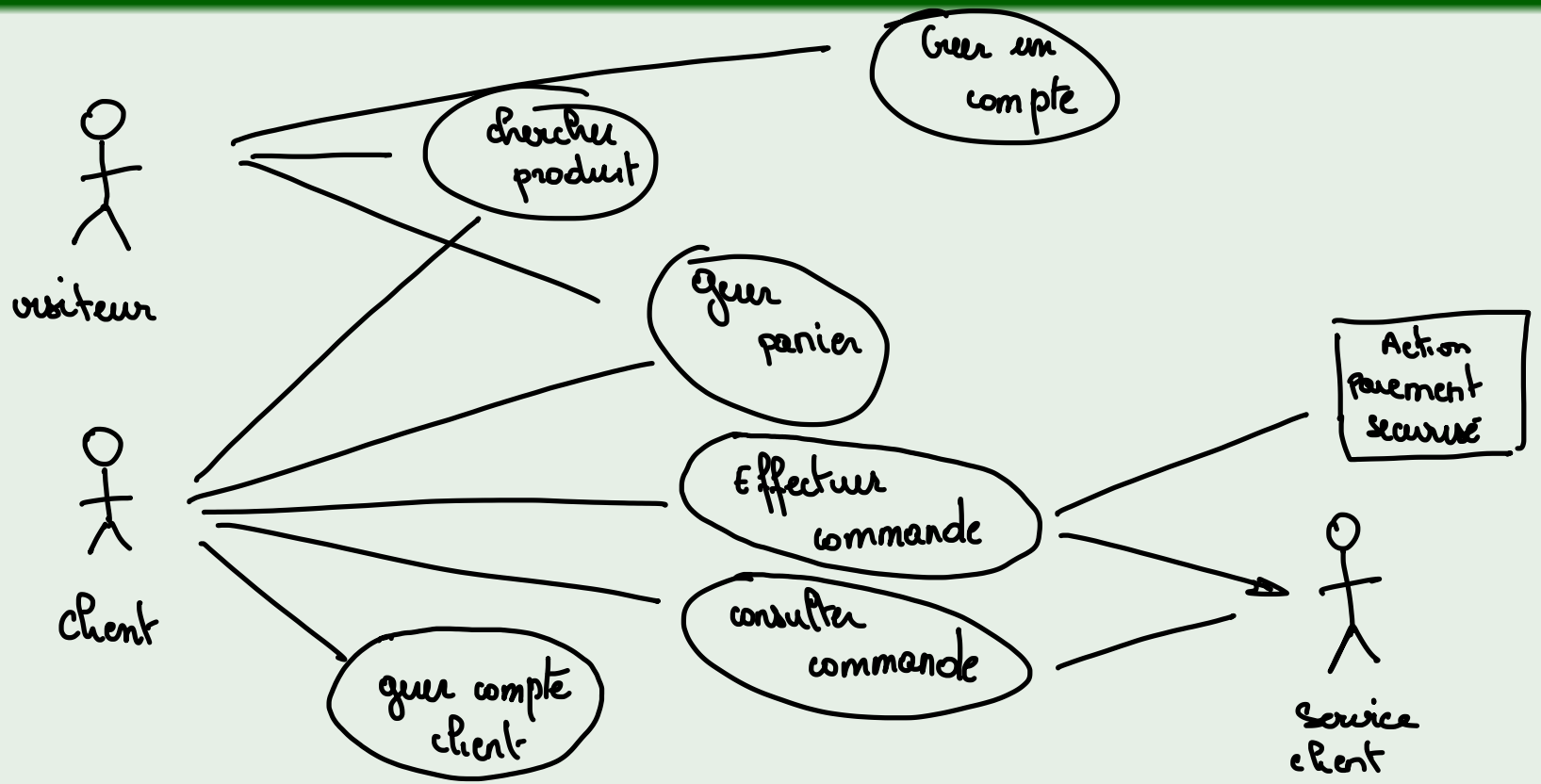
Use case



Cas d'utilisation principaux

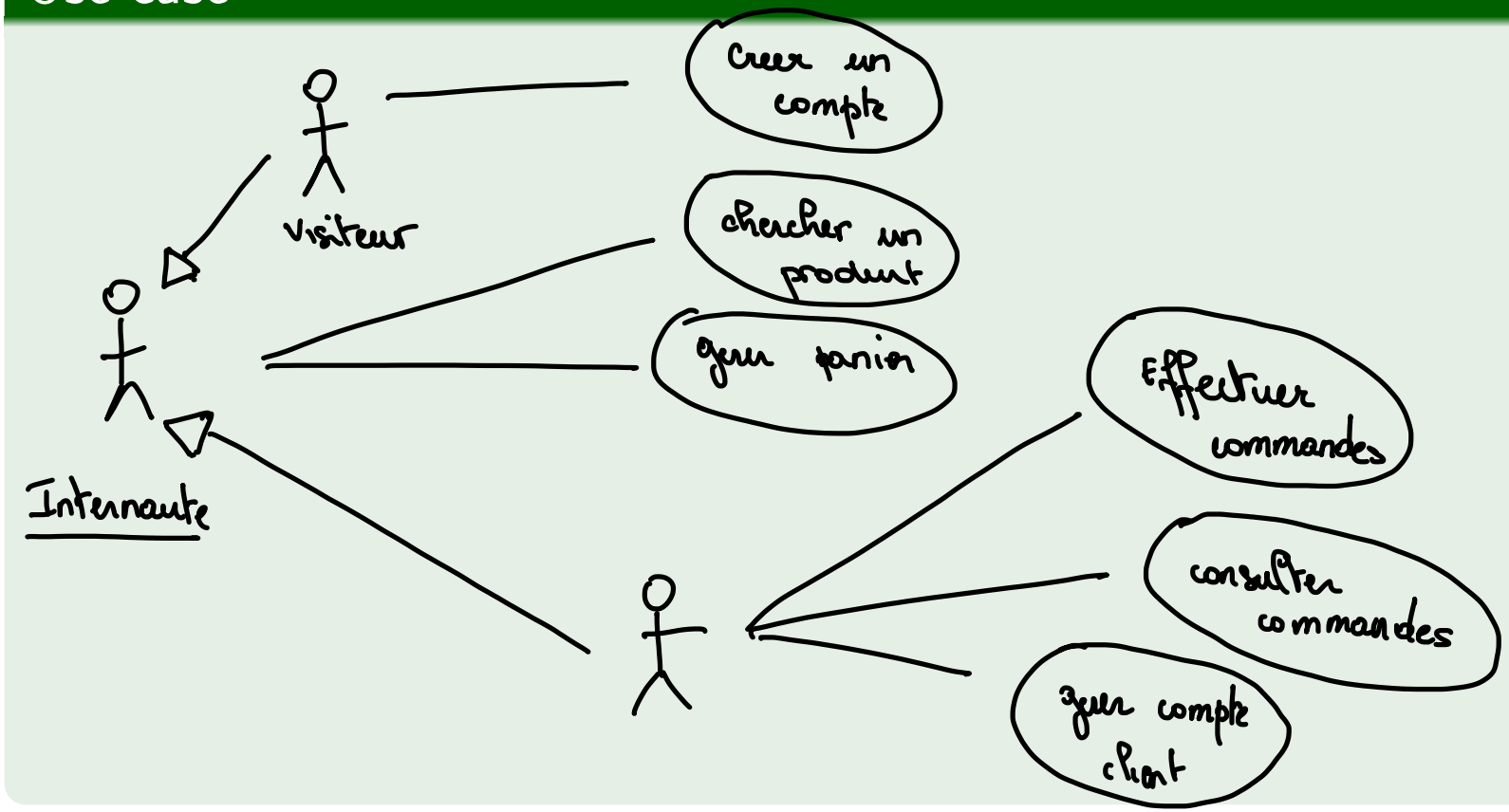
Le besoin utilisateur

Use case



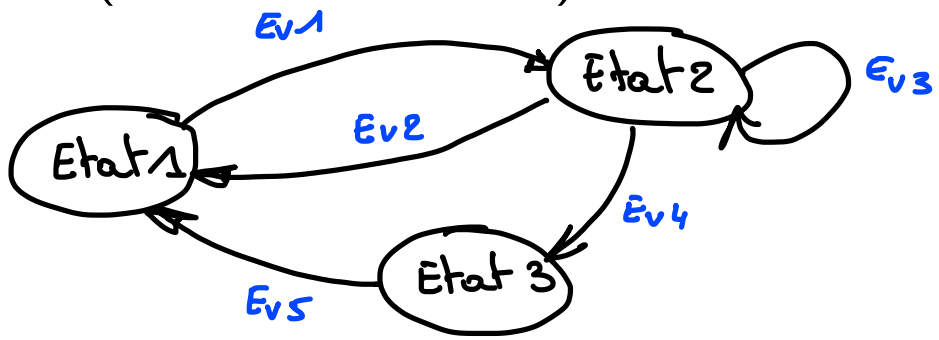
Le besoin utilisateur

Use case

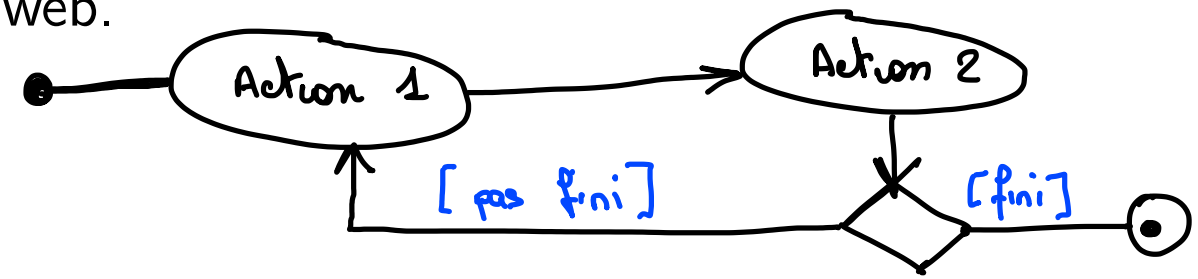


Le besoin utilisateur

- **Diagramme d'état :** Représente le cycle commun aux objets d'une même classe (états - transitions)



- **Diagramme d'activité :** Représente le cycle d'enchainement des actions et des décisions au sein d'une activité. Il peut être utilisé comme une alternative au diagramme d'état pour décrire la navigation dans un site web.



Modélisation de la navigation

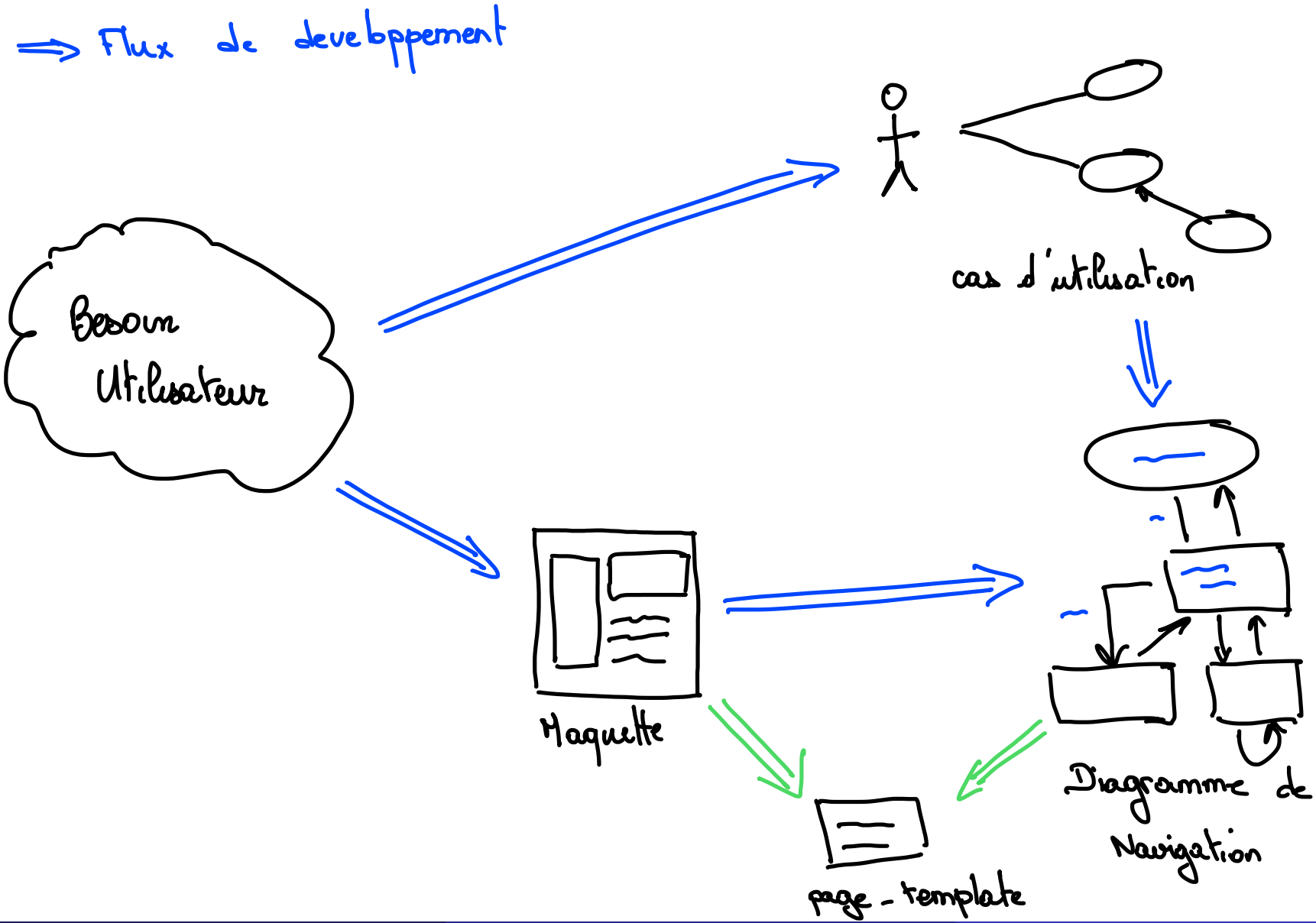


Table des matières

Motivations

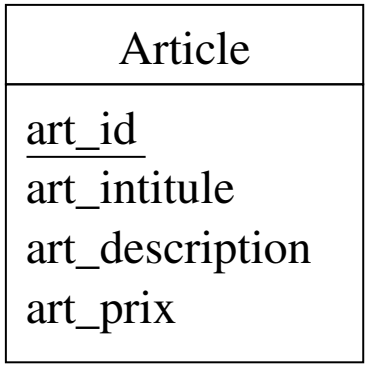
- Particularité des applications Web :
  ↪Mélange de technologies
  ↪Codes exécutés côté client et serveur
- Script PHP : partie exécutée sur le serveur et envoyée au client
  ↪Mélange des fonctionnalités !
- Pour les grosses applications :
  ↪Difficultés de développement
  ↪Problèmes de maintenance
- Nécessité de structurer l'application
- Décomposition efficace :
  - Séparer les fonctionnalités
  - Communication claire entres les différentes parties

Parties de l'application

- Interface :
  - Affichage des données pour l'utilisateur
  - Récupération des données saisies
- Contrôle :
  - Déclenche des actions associées aux actions
- Logique applicative
  - Traitement associés à une application spécifique
- Logique métier/modèle
  - Représentation des données
  - Traitements associés
- Persistance
  - Sauvegarde/chargement des données
  - Utilisation d'une base de données

Description de l'application d'exemple

- Magasin en ligne proposant la vente d'articles
- Article caractérisé par un identifiant, un intitulé, une description et un prix
- Page d'accueil : affichage de tous les articles



MCD (pour l'instant)

Exemple 1

```
<html lang="fr">
  <head> ... </head>
  <body>
    <h1> Bienvenue dans mon magasin </h1>
<?php
$BD = new PDO("mysql:host=localhost;dbname=articles;charset=UTF8",
              "root", "");
if($requete = $BD->query("SELECT␣*␣FROM␣article")) {
    while($resultat = $requete->fetch(PDO::FETCH_ASSOC)) {
?>
    <div>
      <h2><?php echo $resultat['art_intitule']; ?></h2>
      <p><?php echo $resultat['art_description']; ?></p>
      <b><?php echo $resultat['art_prix']; ?> \euro </b>
    </div>
<?php
    }
}
?>
  </body>
</html>
```

Problèmes de cette solution

- Script mélangeant du code HTML et du code PHP
- Difficilement lisible !
- Solution : séparer le code
  - Partie PHP = récupération des données
    ↪Contrôle : script `index.php`
  - Partie HTML = représentation des données
    ↪Vue : script `vueArticles.php`
- Données récupérées par le contrôle puis passées à la vue
  ↪Exemple : tableau `$articles`

Exemple 2 (1/2) : vueArticles.php

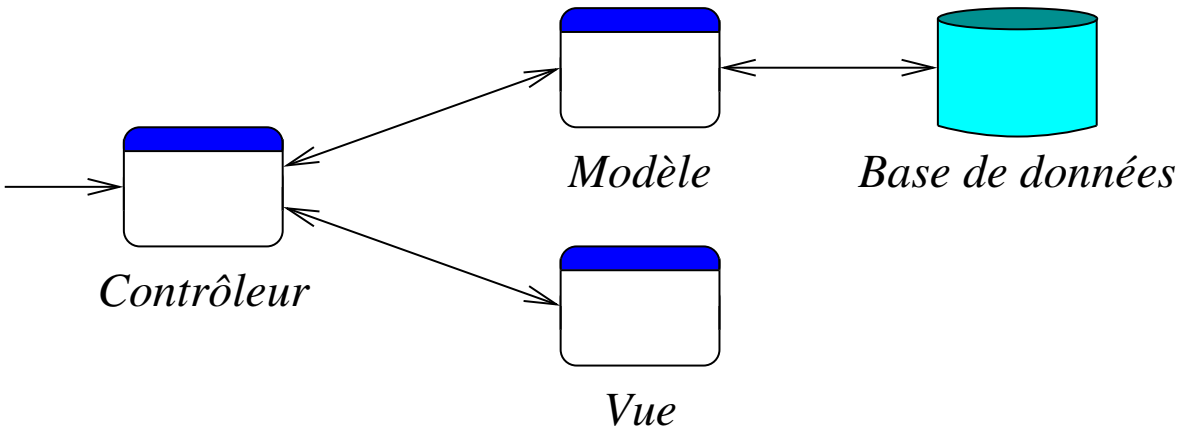
```
<!DOCTYPE html>
<html lang="fr">
  <head> ... </head>
  <body>
    <h1> Bienvenue dans mon magasin </h1>
<?php
foreach($articles as $article) {
?>
    <div>
      <h2><?php echo $article['art_intitule']; ?></h2>
      <p><?php echo $article['art_description']; ?></p>
      <b><?php echo $article['art_prix']; ?> ? </b>
    </div>
<?php
}
?>
  </body>
</html>
```

Exemple 2 (2/2) : index.php

```php
<?php
$BD = new PDO("mysql:host=localhost;dbname=articles;charset=UTF8",
              "root", "");

$articles = $BD->query("SELECT * FROM article");

require("vueArticles.php");
?>
```

Séparer le contrôle du modèle

- Script principal `index.php` :
  ↪Réalise les accès à la base de données
  ↪ET dirige vers la vue
- Solution : séparer encore le code
  - Modèle : récupération des données (accès à la base)
  - Vue : représentation des données
  - Contrôleur : lien entre la vue et le modèle
- Modèle permet d'accéder à différentes données :
  ↪Utilisation de fonctions différentes
  ↪Possible d'utiliser une classe



Exemple 3 (1/2) : `modele.php` (le modèle)

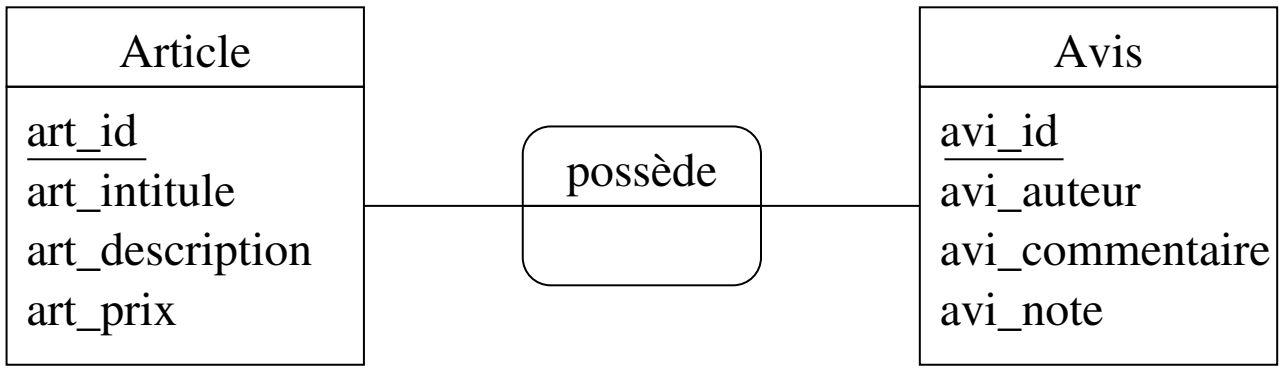
```php
<?php
function getArticles() {
    $BD = new PDO("mysql:host=localhost;dbname=articles;charset=UTF8",
                  "root", "");

    return $BD->query("SELECT * FROM article");
}
?>
```

Exemple 3 (2/2) : `index.php` (le contrôleur)

```php
<?php
require("modele.php");

$articles = getArticles();

require("vueArticles.php");
?>
```

Ajouts de fonctionnalités

- Nous désirons maintenant récupérer les avis des utilisateurs
- Modification du MCD (ajout d'une entité = nouvelle table)
- Nouveau contrôleur :
  ↪Récupère l'identifiant de l'article
  ↪Appel du modèle/vue correspondant
- Modèle : ajout de nouvelles méthodes
  ↪Récupération d'un article et des avis d'un article



MCD modifié

Exemple 4 (1/4) : modele.php (le modèle)

```php
<?php
$BD = new PDO("mysql:host=localhost;dbname=articles;charset=UTF8",
              "root", "");
function getArticles() {
    global $BD;
    return $BD->query("SELECT * FROM article");
}
function getArticle(int $idArticle) {
    global $BD;
    $requete = $BD->prepare("SELECT * FROM article WHERE art_id=:
        article");
    $requete->execute(array(":article" => $idArticle));
    return $requete->fetch(PDO::FETCH_ASSOC);
}
function getAvis(int $idArticle) {
    global $BD;
    $requete = $BD->prepare("SELECT * FROM avis WHERE avi_article=:
        article");
    $requete->execute(array(":article" => $idArticle));
    return $requete;
}
?>
```

Exemple 4 (2/4) : `article.php` (le contrôleur)

```php
<?php
require("modele.php");

if(isset($_GET['article'])) {
    $article = getArticle(intval($_GET['article']));
    $listeAvis = getAvis(intval($_GET['article']));
    require("vueArticle.php");
}
else {
    echo "Erreur ! Identifiant de l'article manquant.";
}
?>
```

Exemple 4 (3/4) : vueArticles.php (première vue)

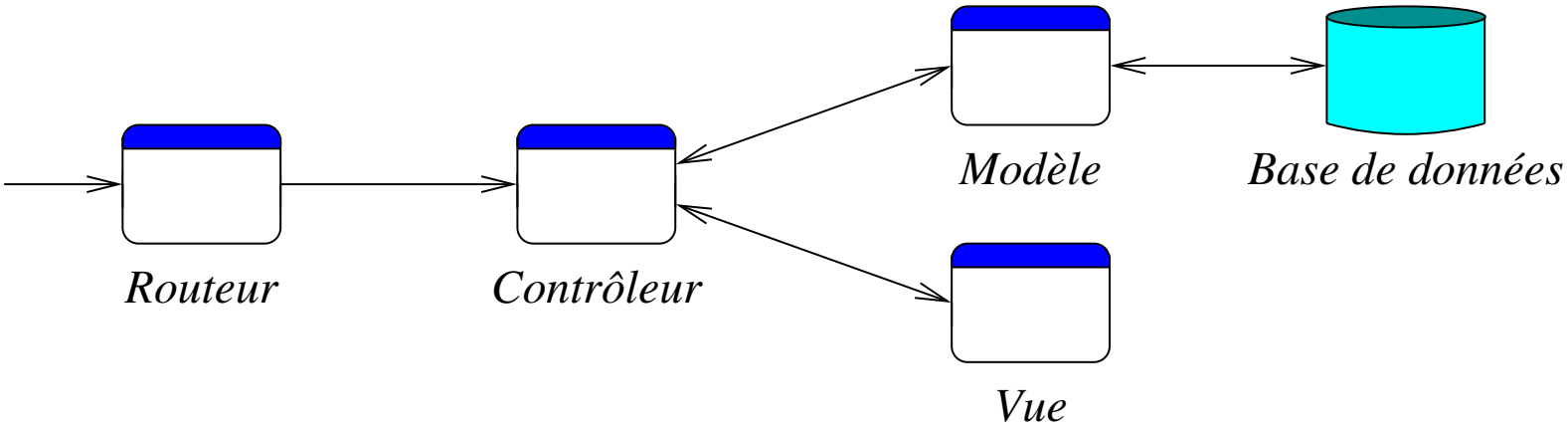
```
<html lang="fr">
  <head> ... </head>
  <body>
    <h1> Bienvenue dans mon magasin </h1>
<?php
foreach($articles as $article) {
?>
    <div>
      <h2><?php echo $article['art_intitule']; ?></h2>
      <p><?php echo $article['art_description']; ?></p>
      <b><?php echo $article['art_prix']; ?> ? </b>
      <p> <a href="article.php?article=<?php␣echo␣$article['art_id'];␣
          ?>"> Voir cet article </a> </p>
    </div>
<?php
}
?>
  </body>
</html>
```

Exemple 4 (4/4) : vueArticle.php (deuxième vue)

```
<html lang="fr">
  <head> ... </head>
  <body>
    <h1> Article sélectionné </h1>
    <div>
      <h2><?php echo $article['art_intitule']; ?></h2>
      <p><?php echo $article['art_description']; ?></p>
      <b><?php echo $article['art_prix']; ?> ? </b>
    </div>
    <h2> Avis des utilisateurs </h2>
<?php while($avis = $listeAvis->fetch()) { ?>
    <div>
      <b><?php echo $avis['avi_auteur']; ?></b>
      <i><?php echo $avis['avi_commentaire']; ?></i>
      <b><?php echo $avis['avi_note']; ?> / 5 </b>
    </div>
<?php } ?>
    <p> <a href="index.php"> Retour à l'accueil␣</a>␣</p>
␣␣</body>
</html>
```

Routage dans l'application

- Pour le moment, un contrôleur par action :
  ↪Multiplication des contrôleurs dans l'application finale !
- Solution : centraliser sur un seul point d'accès
- Le routeur ou le *front controller*



Mise en place

- Contrôleur :
  - Contient les différentes parties (dans des fonctions)
  - Peut être découpé en sous-parties
- Routeur :
  - Vérifications générales (exemple : conditions d'accès, paramètres)
  - Appelle les méthodes du contrôleur
  - Possible d'utiliser un attribut spécifique (paramètre `action`)
    ↪Vérifier la validité pour la sécurité !

Exemple 5 (1/2) : index.php (le routeur)

```php
<?php
require("controller.php");
define("ACTION_DEFAUT", 1);
define("ACTION_ARTICLE", 2);

if(isset($_GET['action']))
    $action = intval($_GET['action']);
else
    $action = ACTION_DEFAUT;

switch($action) {
    case ACTION_ARTICLE:
        if(isset($_GET['article']))
            afficherArticle();
        else
            require("vueErreur.php");
        break;
    default:
        listerArticles();
        break;
}
```

Exemple 5 (2/2) : `controller.php` (le contrôleur)

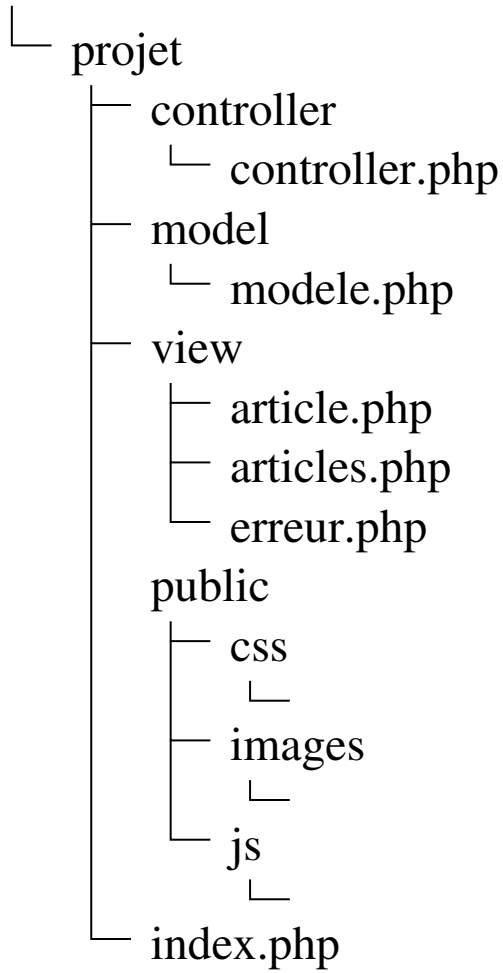
```php
<?php
require("modele.php");

function listerArticles() {
    $articles = getArticles();
    require("vueArticles.php");
}

function afficherArticle() {
    $article = getArticle(intval($_GET['article']));
    $listeAvis = getAvis(intval($_GET['article']));
    require("vueArticle.php");
}
?>
```

Table des matières

Problématique

- Modèle MVC : séparation des fonctions
  ↪Maintenance plus simple
  ↪Réutilisation du code
- Problème : multiplication des fichiers !
  ↪Toutes les fonctionnalités non développées
  ↪Pas encore de CSS, de *Javascript*
- Solution : proposer une arborescence
  ↪Séparation en fonction des parties de l'application

Arborescence classique



Utilisation de la POO

- Manipulation des données fastidieuse
- Solution : utilisation de classes
  ↪Exemple : articles, avis
- Déclarations dans des fichiers distincts
  ↪
- Plus loin dans la structuration
  ↪Contrôleur, modèle ⇒ classes

Persistance des objets

- Création d'objets qui peuvent être réutilisés :
  ↪Création/chargement depuis la base de données
- Éviter de mélanger le code des classes + SQL
- Solution : création de classes spécifiques pour la base de données
  ↪Classes DAO (*Data Access Objet*)
  ↪Fonctionnalités CRUD (*Create*, *Retrieve*, *Update* et *Delete*)
- Exemples :
  ↪Article.php + ArticleDAO.php
  ↪Commentaire.php + CommentaireDAO.php

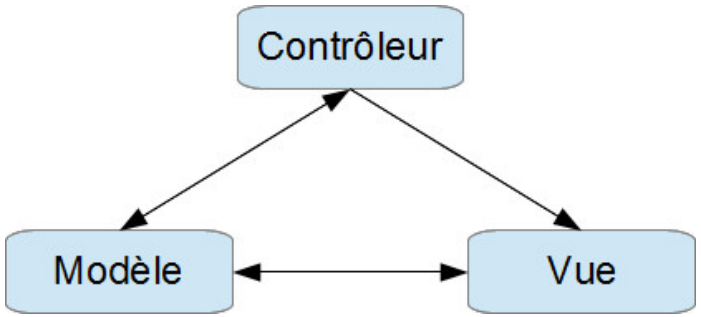
Table des matières

Il existe de nombreux framework MVC :

Que propose Laravel

- un système de routage perfectionné (RESTFul et ressources),
- un créateur de requêtes SQL et un ORM performants,
- un moteur de template efficace,
- un système d'authentification pour les connexions,
- un système de validation,
- un système de pagination,
- un système de migration pour les bases de données,
- un système d'envoi d'emails,
- un système de cache,
- un système d'événements,
- un système d'autorisations,
- une gestion des sessions...

Le MVC vu par Laravel



C'est un modèle d'organisation du code :

- le modèle est chargé de gérer les données
- la vue est chargée de la mise en forme pour l'utilisateur
- le contrôleur est chargé de gérer l'ensemble

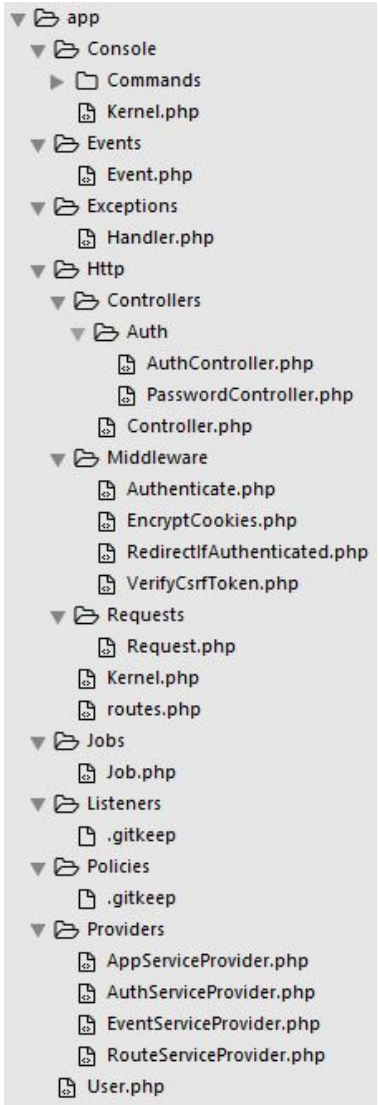
En général on résume en disant que le modèle gère la base de données, la vue produit les pages HTML et le contrôleur fait tout le reste.

Dans Laravel :

- le modèle correspond à une table d'une base de données. C'est une classe qui étend la classeModel qui permet une gestion simple et efficace des manipulations de données et l'établissement automatisé de relations entre tables ;
- le contrôleur se décline en deux catégories : contrôleur classique et contrôleur de ressource ;
- la vue est soit un simple fichier avec du code HTML, soit un fichier utilisant le système de templateBlade de Laravel.

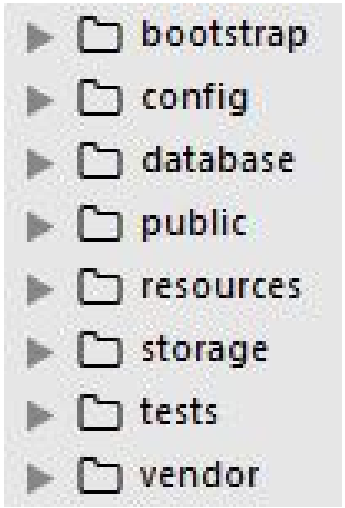
Le dossier App

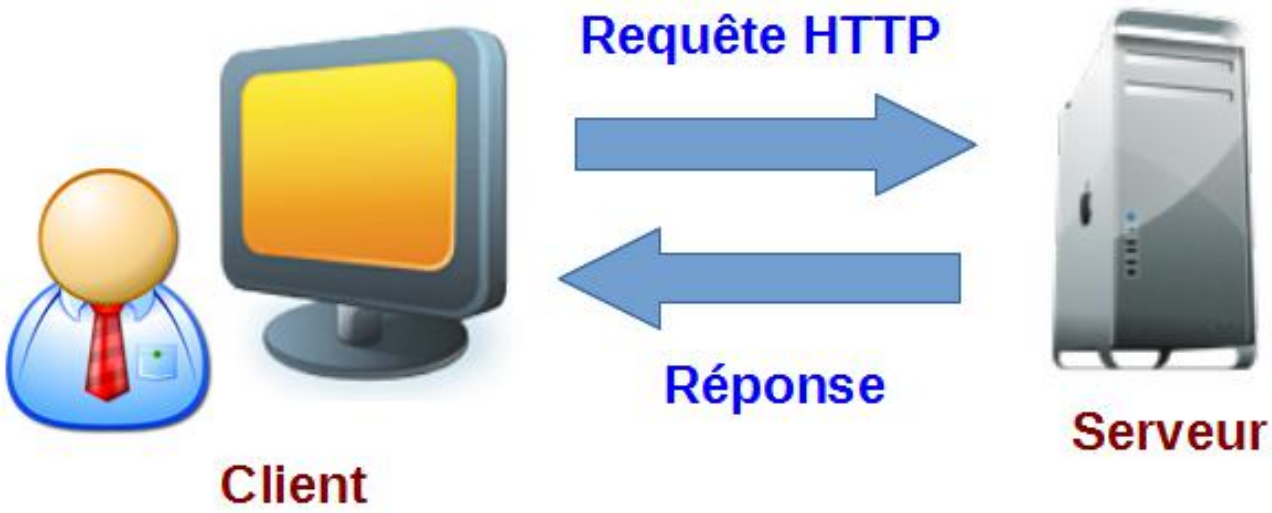
- **Console/Commands** : commandes en mode console ;
- **Jobs** : commandes concernant les tâches effectuées par l'application ;
- **Events** et **Listeners** : événements et écouteurs nécessaires à l'application ;
- **Http** : tout ce qui concerne la communication : contrôleurs, routes, middlewares et requêtes,
- **Providers** : tous les fournisseurs de services (providers).Les providers servent à initialiser les composants.
- **Policies** : gestion des droits d'accès.



Les autres dossiers

- **bootstrap** : scripts d'initialisation ;
- **public** : dossier public du site : images, CSS, scripts...
- **vendor** : tous les composants de Laravel et de ses dépendances,
- **config** : Configurations : application, authentification, cache, bd,...
- **database** : migrations et les populations ;
- **resources** : vues, fichiers langague, assets ;
- **storage** : données temporaires de l'application ;
- **tests** : fichiers de tests unitaires.





La requête du client comporte un certain nombre d'informations

- la **méthode** : get, post, put, delete...
- **l'url** : c'est l'adresse de la page demandée sur le serveur

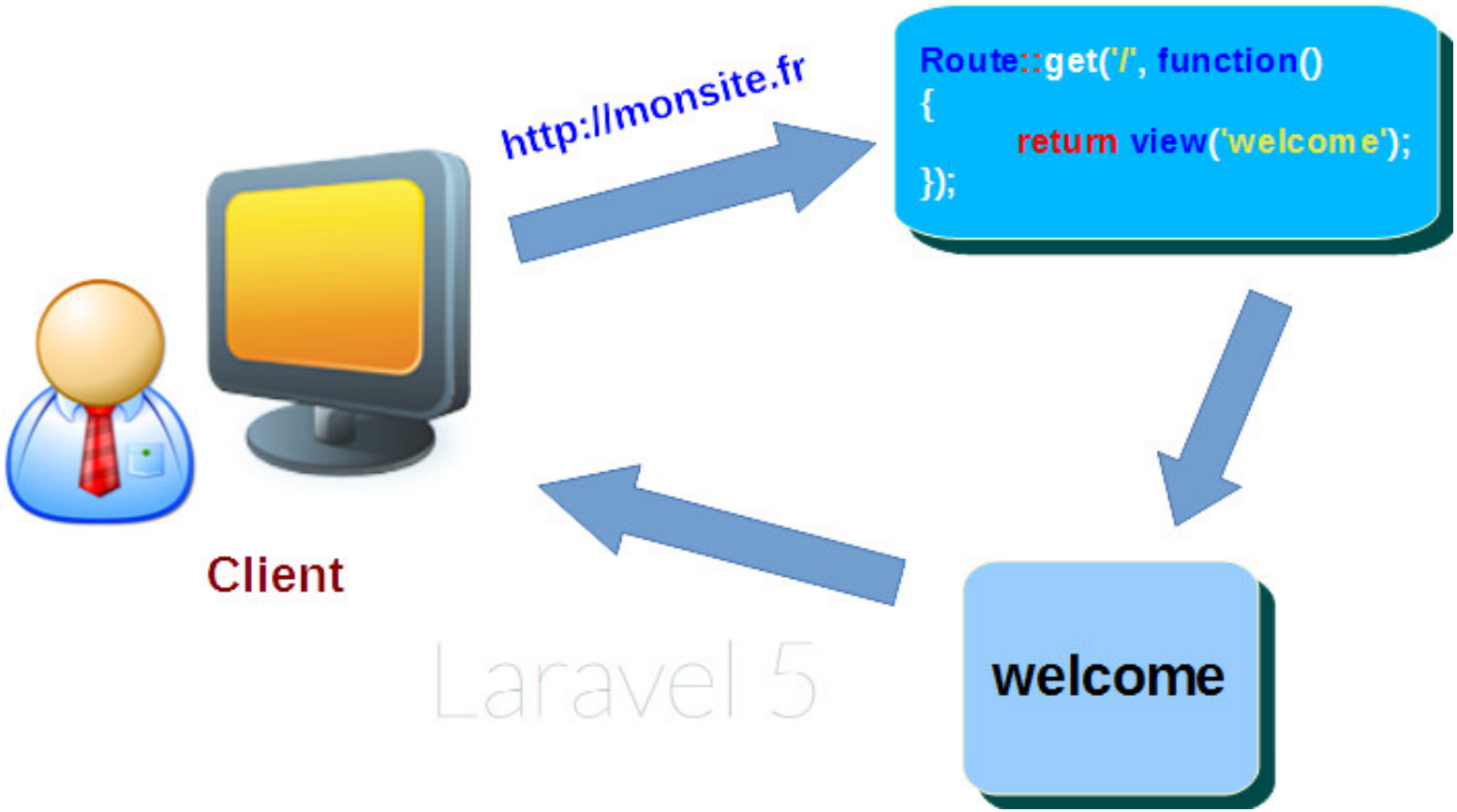
Le cycle de la requête

Lorsque la requête atteint le fichier public/index.php l'application Laravel est créée et configurée et l'environnement est détecté. Ensuite le fichier web.php est chargé

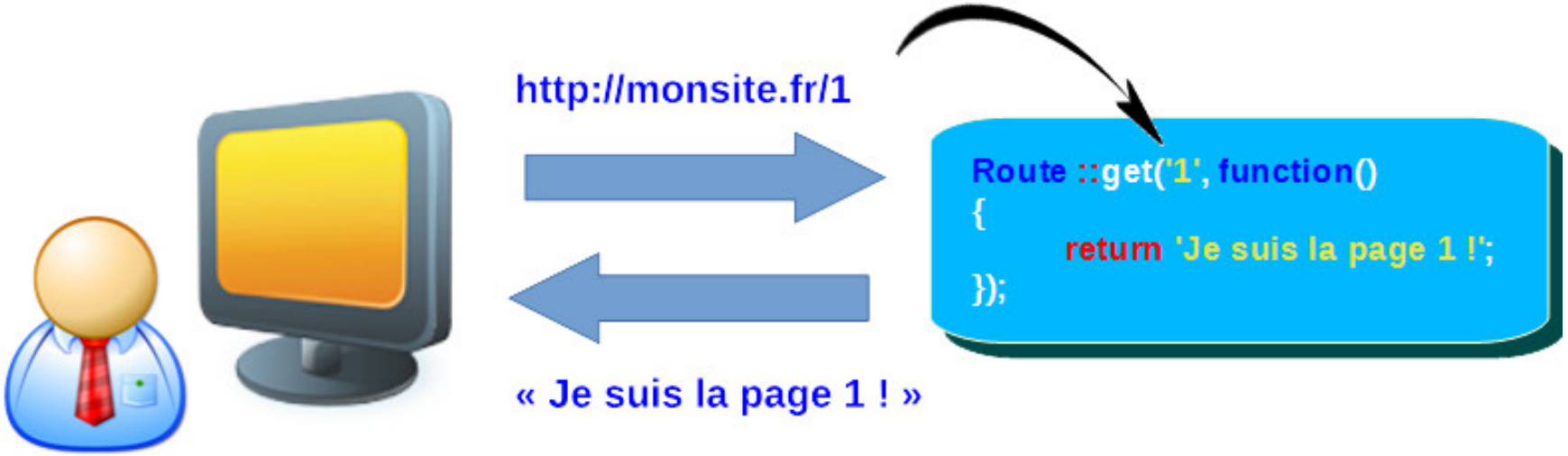
```php
<?php
Route::get('/', function () {
    return view('welcome');
});
```

- **Route** : on utilise le routeur,
- **get** : on regarde si la requête a la méthode "get",
- '/' : on regarde si l'url comporte uniquement le nom de domaine,
- dans la fonction anonyme on retourne (return) une vue (view) à partir du fichier "welcome".

Exemple 0


http://monsite.fr
Route::get('/', function()
{
return view('welcome');
});
Client
welcome
Laravel 5

Exemple 1 Route paramètrée




```php
<?php
Route::get('{n}', function($n) {
    return 'Je suis la page ' . $n . ' !';
});
```

Les Vues

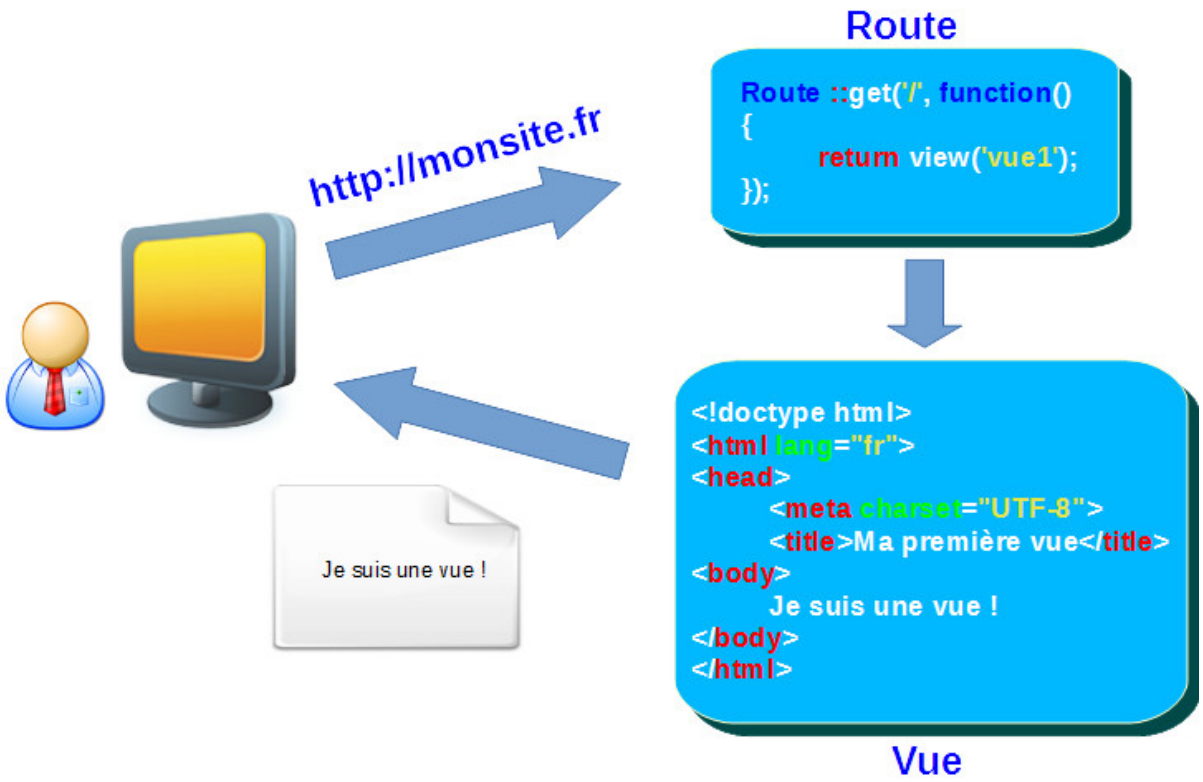
Dans une application réelle vous retournerez rarement la réponse directement à partir d'une route, vous passerez au moins par une vue. Dans sa version la plus simple une vue est un simple fichier avec du code html :

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Ma première vue</title>
6  </head>
7  <body>
8      Je suis une vue !
9  </body>
10 </html>
```

On enregistre cette vue dans le dossier resources/views avec l'extension php.

Exemple 2

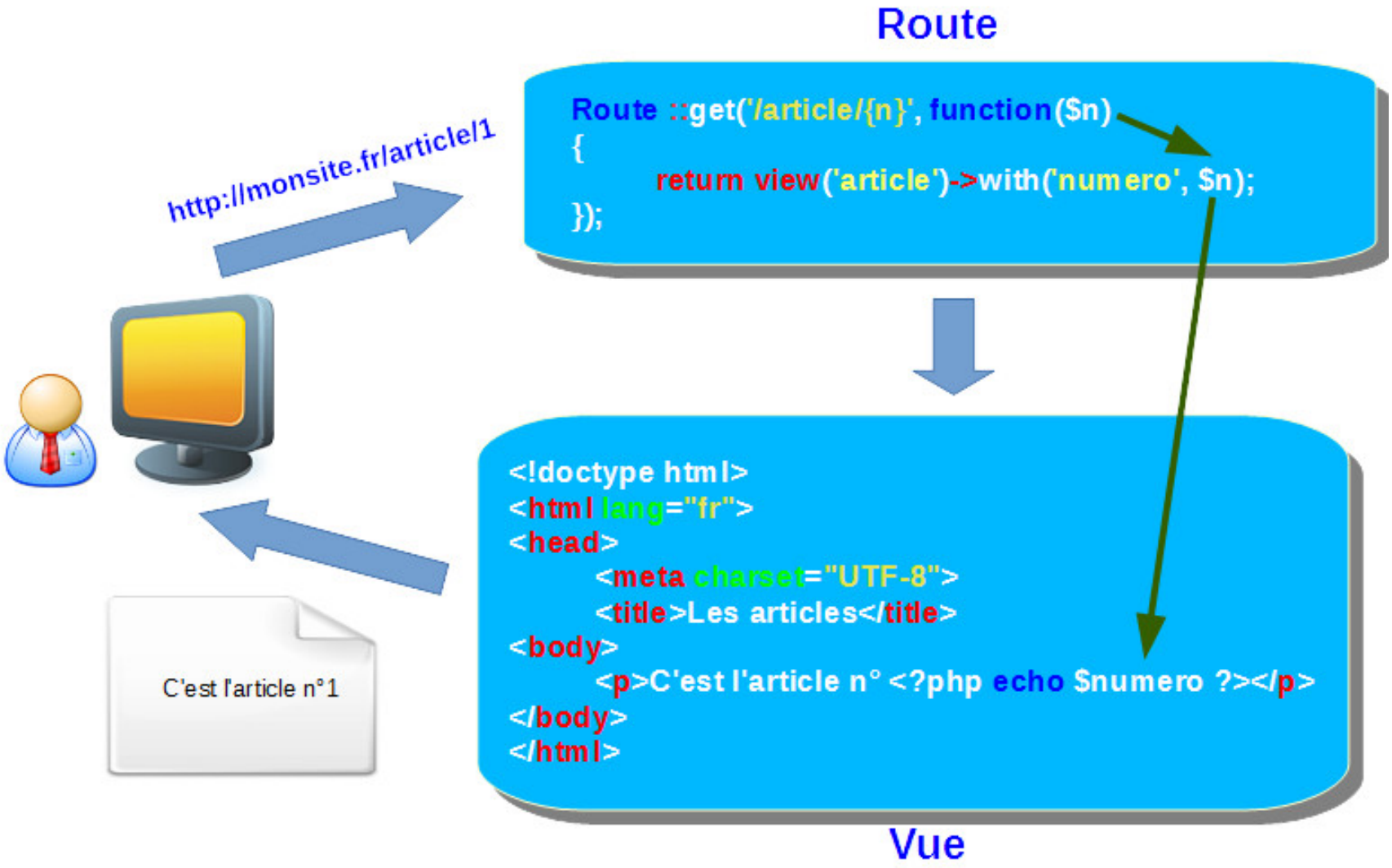
```php
<?php
Route::get('/', function()
{
    return view('vue1');
});
```



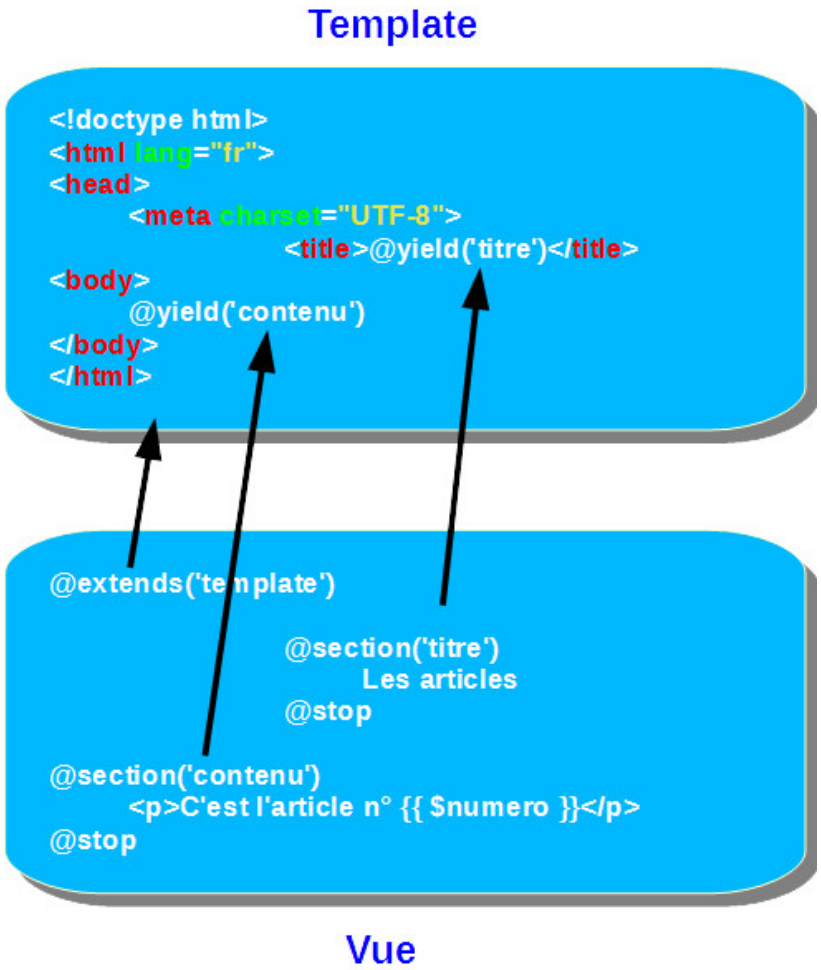
Exemple 3 : Vues Paramétrée

```html
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Les articles</title>
</head>
<body>
    <p>C'est l'article n° <?php echo $numero ?></p>
</body>
</html>
```

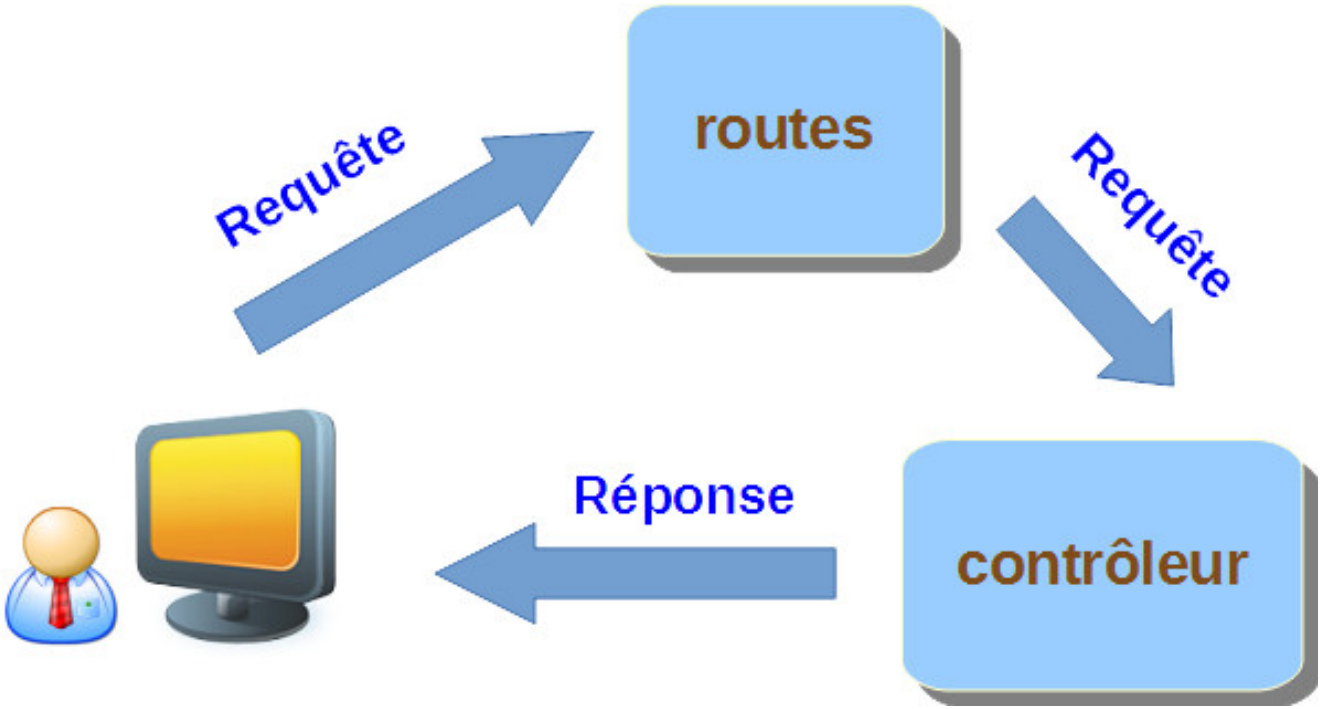
```php
<?php
Route::get('article/{n}', function($n) {
    return view('article')->with('numero', $n);
})->where('n', '[0-9]+');
```


Route
Route ::get('/article/{n}', function($n)
{
return view('article')->with('numero', $n);
});
http://monsite.fr/article/1
<!doctype html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<title>Les articles</title>
<body>
C'est l'article n° <?php echo $numero ?>
</body>
</html>
C'est l'article n°1
Vue

Exemple 4 : Utilisation de Blade



Le contrôleur



Le contrôleur