Développement d’application Web

Modèle-vue-contrôleur

Table des matières

Document écrit par Stéphane Gill

Avril 2019

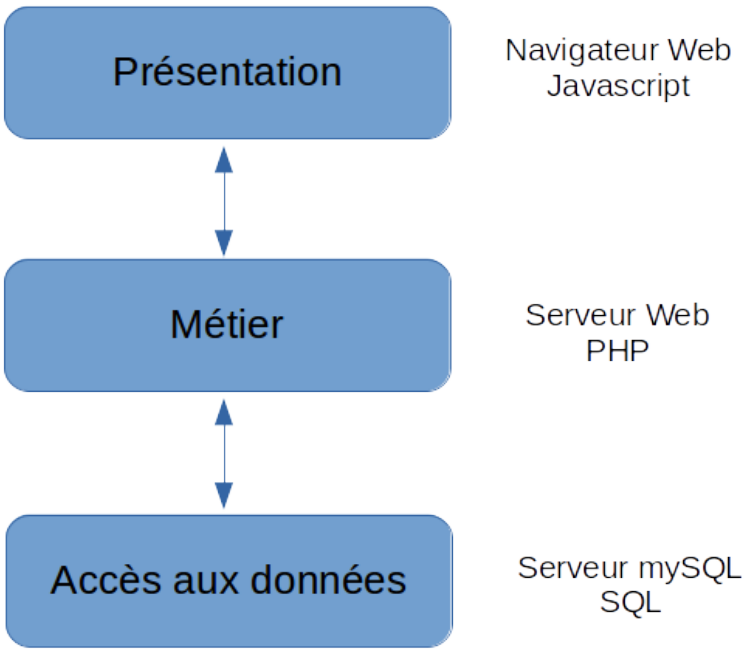
Dans ce chapitre, l'architecture 3-niveaux et le modèle de conception MVC sont d'abord présentés. Par la suite, un exemple de cadre d'application utilisant MVC est expliqué.

Architecture 3-niveaux

L'architecture 3-niveaux (3-tiers) est une version particulière de l'architecture multiniveau (n-tier). Il s'agit d'un modèle logique d'architecture applicative qui vise à modéliser une application comme un empilement de trois couches logicielles. C'est une extension du modèle client-serveur.

L'architecture logique du système est divisée en trois niveaux :

- présentation;
- métier;
- accès aux données.



Le rôle de chaque couche est clairement défini :

- Le niveau présentation : correspondant à l'affichage, la restitution sur le poste de travail, le dialogue avec l'utilisateur;

- Le niveau métier : correspondant à la mise en œuvre de l’ensemble des règles de gestion et de la logique applicative;
- Le niveau accès aux données : correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

Dans cette approche, les couches communiquent entre elles au travers d’un « modèle d’échange » c’est-à-dire que chaque couche ne communique qu’avec ses voisins immédiats.

Le rôle de chacune des couches et leur interface de communication étant bien définis, les fonctionnalités de chacune d’entre elles peuvent évoluer sans induire de changement dans les autres niveaux. Cependant, l’ajout d’une nouvelle fonctionnalité de l’application peut avoir des répercussions dans plusieurs niveaux.

Modèle-vue-contrôleur

Modèle-Vue-Contrôleur (MVC) est un modèle de conception (design pattern) logicielle conçu en 1978 pour interfaces graphiques. Ce modèle est maintenant très populaire pour les applications web. Il est entre autre utilisé par de nombreux cadre d’application (frameworks) tels que Ruby on Rails, Django, ASP.NET MVC, Spring, Struts, Symfony, Apache Tapestry ou Angular Js.

Le modèle de conception MVC est composé de trois modules ayant des responsabilités différentes :

- Le modèle;
- La vue;
- Le contrôleur.

Rôle des composants

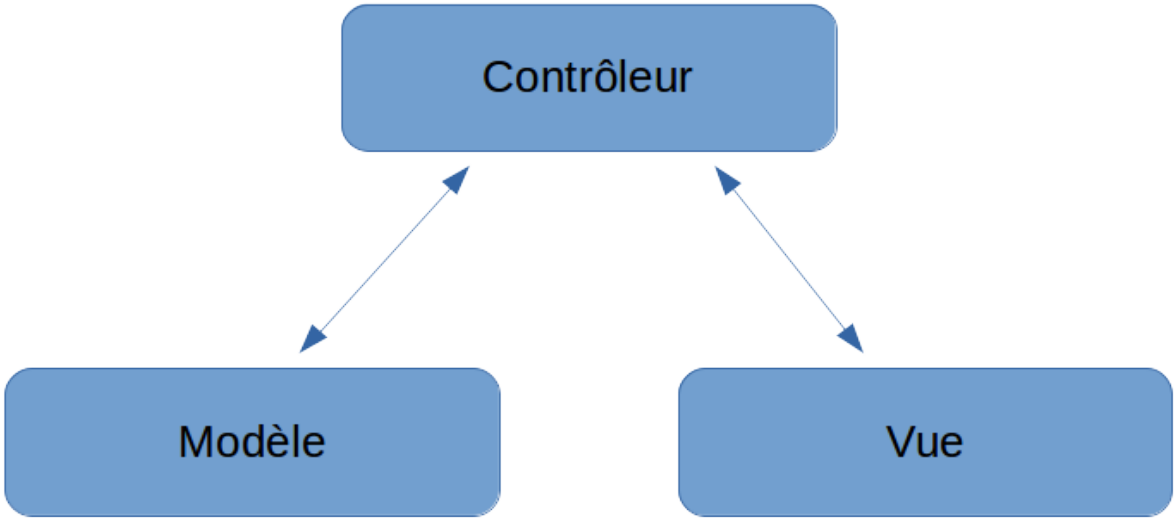
Le modèle encapsule la logique métier ainsi que l’accès aux données. Il peut s'agir d'un ensemble de fonctions (Modèle procédural) ou de classes (Modèle orienté objet).

La vue s’occupe des interactions avec l’utilisateur : présentation, saisie et validation des données.

Le contrôleur gère la dynamique de l’application. Il fait le lien entre l'utilisateur et le reste de l’application.

Interaction entre les composants

Le diagramme ci-dessous résume les relations entre les composants d’une architecture MVC.



La demande de l'utilisateur (requête HTTP) est reçue et interprétée par le Contrôleur. Celui-ci utilise les services du Modèle afin de préparer les données à afficher.

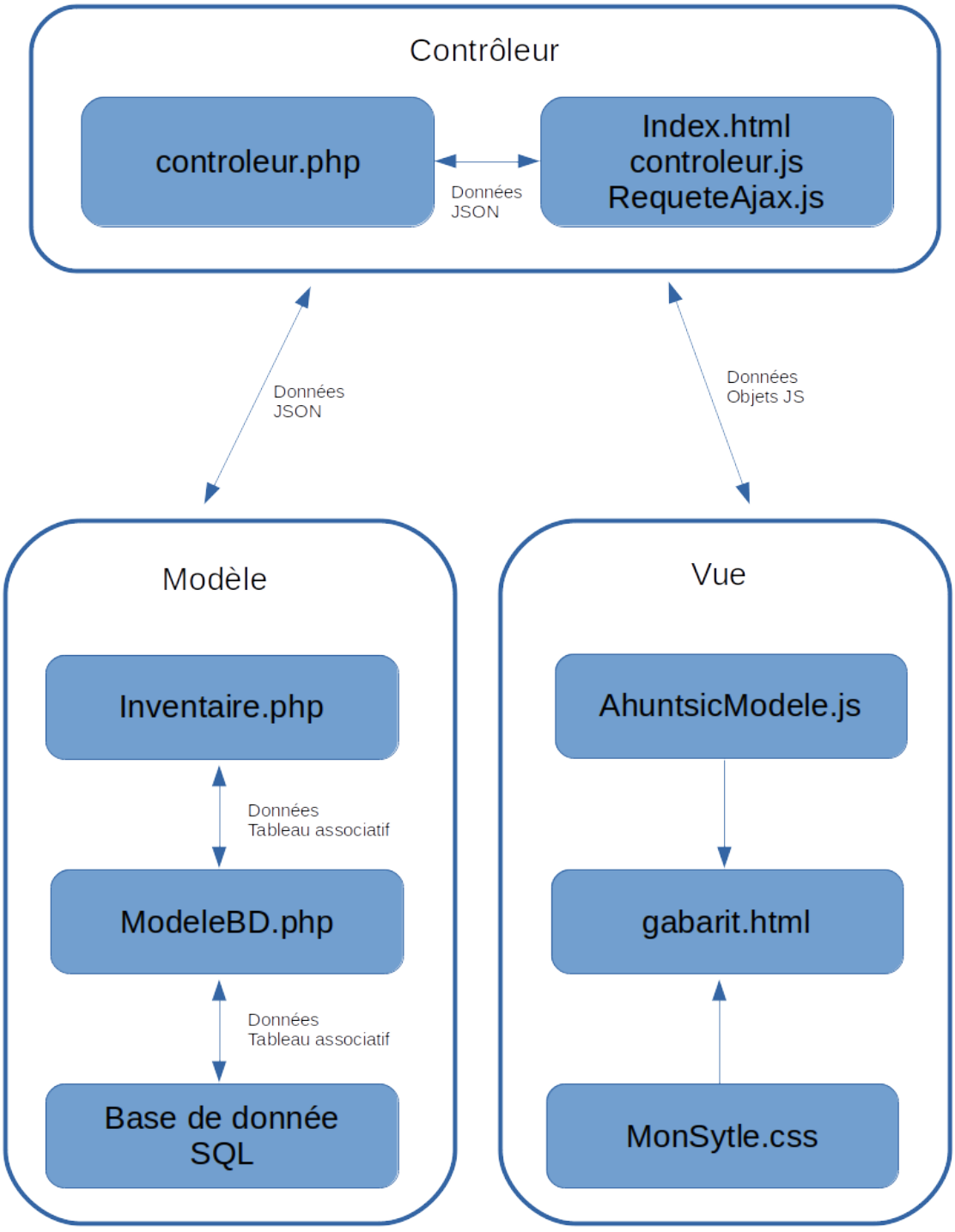
Ensuite, le Contrôleur fournit ces données à la Vue, qui les présente à l'utilisateur sous la forme d'une page HTML. Une application construite sur le principe du MVC se compose toujours de trois parties distinctes. Cependant, il est fréquent que chaque partie soit elle-même décomposée en plusieurs éléments. On peut ainsi trouver plusieurs modèles, plusieurs vues ou plusieurs contrôleurs à l'intérieur d'une application MVC.

Avantages et inconvénients

Le modèle MVC offre une séparation claire des responsabilités au sein d'une application, en conformité avec les principes de conception suivant : responsabilité unique, couplage faible et cohésion forte. Le prix à payer est une augmentation de la complexité de l'architecture.

Dans le cas d'une application Web, l'application du modèle MVC permet aux pages HTML de contenir le moins possible de code serveur, étant donné que la programmation est regroupée dans les deux autres parties de l'application.

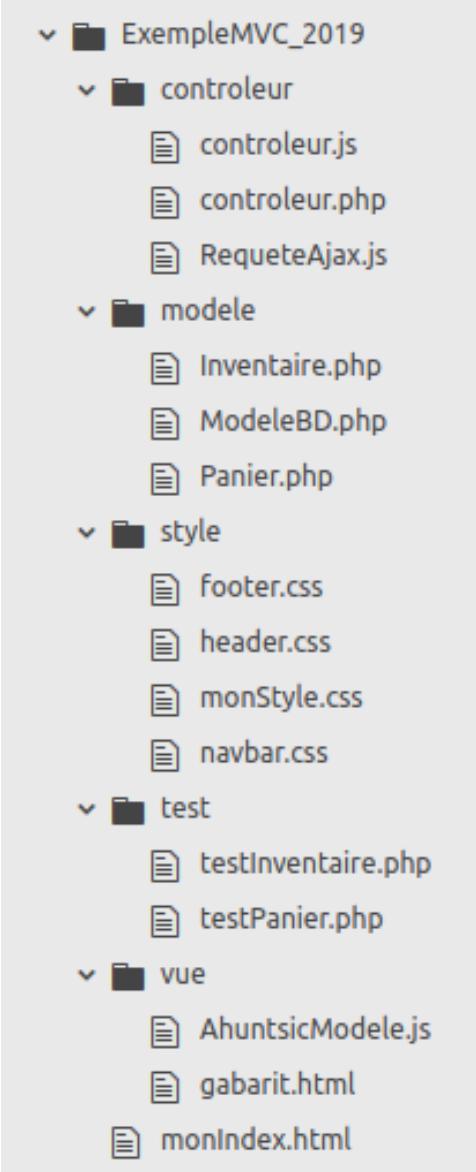
Exemple de site d’achat en ligne



Avant de commencer à créer une application Web (structuré avec le modèle MVC), il faut organiser les nombreux fichiers dans des répertoires. La solution la plus évidente consiste à créer les sous-répertoires en suivant le découpage MVC :

- le répertoire **modele** contiendra les fichiers du modèle ;
- le répertoire **vue** contiendra les fichiers de la vue ;
- le répertoire **controleur** contiendra le fichier du contrôleur.

On peut également prévoir un répertoire **style** pour les fichiers CSS. On aboutit à l'organisation suivante :



Fichier monIndex.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemple d'architecture MVC</title>
  <link rel="stylesheet" type="text/css" href="style/monStyle.css" media="all" >
  <link rel="stylesheet" type="text/css" href="style/header.css" media="all" >
  <link rel="stylesheet" type="text/css" href="style/navbar.css" media="all" >
  <link rel="stylesheet" type="text/css" href="style/footer.css" media="all" >
  <script src="jquery.min.js"></script>
  <script src="controleur/RequeteAjax.js"></script>
  <script src="controleur/controleur.js"></script>
  <script src="vue/AhuntsicModele.js"></script>
  <script src="vue/vue.js"></script>
</head>

<body>
  <div id="gabarit"></div>
</body>

</html>
```

Fichier controleur.js

```
$(document).ready(function(){
  // charger le gabarit html
  $("#gabarit").load("vue/gabarit.html", function() {
    // charger le contenu du catalogue et du panier
    loadCatalogue();
    loadPanier();
  });
});

function loadCatalogue(){
  let requete = new RequeteAjax("controleur/controleur.php");
  let modeleInventaire = new ahuntsic_modele("modele-inventaire");
  requete.getJSON(
       data => {modeleInventaire.applyTemplate_toAll(data, "contenu");});
 }

 function ajouterPanier(noArticle){
   let requete = new RequeteAjax(
    "controleur/controleur.php?action=ajouterPanier&noArticle="+noArticle);
   let modeleInventaire = new ahuntsic_modele("modele-panier");
   requete.getJSON(loadPanier);
 }

 function loadPanier(){
   let requete = new RequeteAjax(
                 "controleur/controleur.php?action=getPanier");
   let modeleInventaire = new ahuntsic_modele("modele-panier");
   requete.getJSON(
```

```
29      data => {modeleInventaire.applyTemplate_toAll(data, "panier");});
30  }
```

Fichier RequeteAjax.js

```
1   'use strict';
2
3   class RequeteAjax {
4
5     constructor(url) {
6       this.url = url;
7     }
8
9     getJSON(callback) {
10      var xhttp = new XMLHttpRequest();
11      xhttp.onreadystatechange = function() {
12        if (this.readyState == 4 && this.status == 200) {
13          callback(this.responseText);
14        }
15      };
16      xhttp.open("GET", this.url, true);
17      xhttp.send();
18    }
19  }
```

Fichier gabarit.html

```
1   <body>
2      <header>
3        <h1>Exemple d'architecture MVC</h1>
4      </header>
5
6      <nav>
7        <h1>Menu</h1>
8        <!-- Insérer le menu ici -->
9        <ul>
10         <li class="selected"><a href="#">Produits</a></li>
11         <li><a href="#">Marques</a></li>
12         <li><a href="#">Soldes</a></li>
13         <li><a href="#">Contact</a></li>
14       </ul>
15     </nav>
16
17     <section>
18       <article>
19         <h1>Mon catalogue</h1>
20         <!-- Insérer la liste des items ici -->
21         <div id = "contenu"></div>
22      </article>
```

```
      <h1>Mon panier</h1>
      <div id = "panier"></div>
    <aside>

    </aside>
  </section>

  <footer>
    <p>Copyright Stephane - Tous droits reserves<br />
      <a href="#">Me contacter !</a></p>
  </footer>

  <template id="modele-inventaire" type="text/AhuntsicModele">
    <h2>{description}</h2>
    <p>
      <button type="button" onclick="ajouterPanier({noArticle})">
         Ajouter au panier
      </button>
      {prixUnitaire} $
    </p>
  </template>

  <template id="modele-panier" type="text/AhuntsicModele">
    <h2>Article {libelleProduit}</h2>
    <p>Qte : {qteProduit} prix : {prixProduit}$</p>
  </template>

</body>
```

Fichier controleur.php

```
<?php

// Inclure les modèles
require_once '../modele/Inventaire.php';
require_once '../modele/Panier.php';

session_start();

if(!isset($_GET["action"])){
  $action = "getItems";
} else {
  $action = $_GET["action"];
}

if($action == "getItems"){
  $inventaire = new Inventaire();
  echo $inventaire->getItems();
}
else if($action == "ajouterPanier"){
  $panier = new Panier();
  $panier->ajouterArticle($_GET["noArticle"],1,10.0);
  echo $_GET["noArticle"];
```

```
}
else if($action == "getPanier"){
  $panier = new Panier();
  echo $panier->affPanier();
}

?>
```

Fichier ModeleBD.php

```
<?php
abstract class ModeleBD{
  private $servername = "localhost";
  private $username = "root";
  private $password = "";
  private $dbname = "monMagasin";

  protected function executerRequete($sql){
    $conn = new mysqli($this->servername, $this->username,
                                        $this->password, $this->dbname);
    if ($conn->connect_error) {
      die("Echec de la connexion: " . $conn->connect_error);
    }
    $result = $conn->query($sql);
    $conn->close();
    return $result;
  }
}

?>
```

Fichier Inventaire.php

```
<?php
require_once 'ModeleBD.php';

class Inventaire extends ModeleBD {
  public function getItems(){
    $result = $this->executerRequete("SELECT * FROM inventaire");
    $resultArray = $result->fetch_all(MYSQLI_ASSOC);
    return json_encode($resultArray);
  }
}
?>
```

Fichier AhuntsicModele.js

```
'use strict';

```

```
3   class ahuntsic_modele{
4
5     constructor(nomModele) {
6       this.modele = document.getElementById(nomModele).innerHTML;
7     }
8
9     applyTemplate_toAll(txtJSON, balise){
10      let objJSON = JSON.parse(txtJSON);
11      let code_html ="";
12
13      for (let i = 0; i < objJSON.length; i++){
14        let modele_tmp = this.modele;
15        for (let a in objJSON[i]){
16          modele_tmp = modele_tmp.replace(new RegExp("\{" + a + "\}", "g"),
17                                                              objJSON[i][a]);
18        }
19        code_html += modele_tmp;
20      }
21
22      document.getElementById(balise).innerHTML = code_html;
23    }
24  }
```