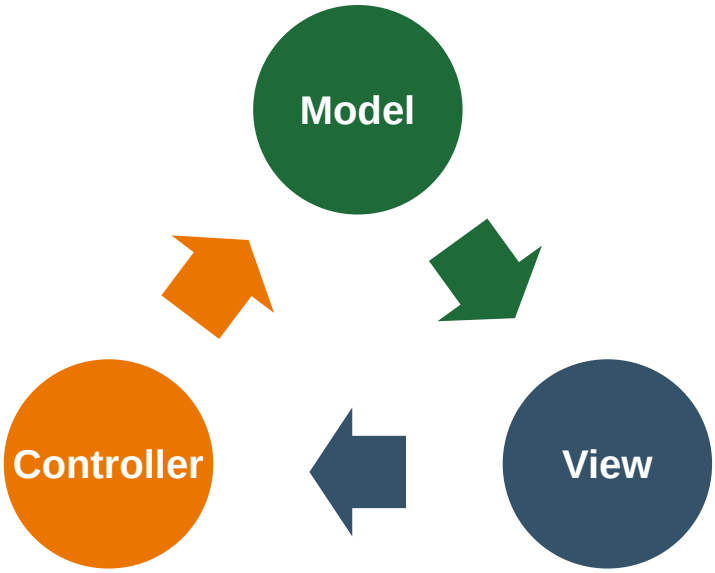


Architecture MVC

date	révision
juillet 2021	Création
2023-09-17	v.29 Correction de forme et table des matières ; ajout des spécificités NodeJS
2023-10-13	V.33 Ajout d'objectifs et corrections mineures

SOMMAIRE

1 Introduction....4
1.1 Objectifs....4
1.2 Prérequis....4
2 Explications....5
2.1 Développement client-serveur....5
2.1.1 Problématique....5
2.1.2 Solution MVC....5
2.1.3 Fonctionnement HTTP....6
2.2 Utilisation MVC sous Node.JS....7
2.2.1 MVC en général....7
2.2.2 MVC avec NodeJS....8
3 Annexes....9
3.1 Sources....9

MODÈLE DE CONCEPTION MVC

L'activité proposée ici aborde l'utilisation d'un motif de développement couramment appelé MVC pour "Model-View-Controller" qui signifie en français "Modèle-Vue-Contrôleur".

À l'issue de cette activité, l'étudiant devra :

- Comprendre la gestion d'une application complexe
- Expliquer l'intérêt du modèle de conception MVC
- Savoir mettre en œuvre cette méthode de conception de codes
- Être capable de créer une application respectant ce 'design pattern'

 **Durée**	 **Difficulté**	 **Taxonomie Bloom**
1 séance	❶ ❷ ❸ ④	Compréhension – Application

1 INTRODUCTION

Le Modèle-Vue-Contrôleur, abrégé en MVC, est un modèle d'architecture logicielle qui divise une application en trois composants interconnectés. Il a été largement adopté dans le développement logiciel pour sa capacité à structurer, organiser et simplifier la création d'applications complexes.

Le MVC permet donc de rendre les applications plus modulaires, évolutives et faciles à maintenir.

1.1 OBJECTIFS

Ce cours va présenter ce modèle d'architecture et tenter de vous convaincre qu'il s'agit d'une méthode de développement incontournable. À la fin de ce cours :

- ☑ Vous connaîtrez les principes fondamentaux du MVC
- ☑ Vous comprendrez comment structurer votre code
- ☑ Vous saurez reconnaître les structures MVC dans les frameworks
  - Angular
  - React
  - autres
- ☑ Vous comprendrez le rôle essentiel du routage

1.2 PRÉREQUIS

Pour générer de manière autonome, une application MVC, il est impératif de savoir programmer correctement. La connaissance d'un ou plusieurs langages de programmation est utile.

D'autre part, la pratique d'un framework comme Angular.JS ou Vue.JS vous aidera à associer les concepts importants.

2 EXPLICATIONS

2.1 DÉVELOPPEMENT CLIENT-SERVEUR

2.1.1 Problématique

La conception utilisant ce modèle est liée au fonctionnement du web et des requêtes HTTP.

Dans une application lourde comme C#, Java, C++, le code s'exécute sur une seule machine : le développeur s'occupe de tout et son application est d'un seul bloc ; Dans une application web, le code s'exécute côté client (navigateur) et côté serveur : les codes sont séparés et n'utilisent pas le même langage.

Rapidement, les applications web sont devenues plus complexes à maintenir. Il était difficile de les déboguer et les faire évoluer :

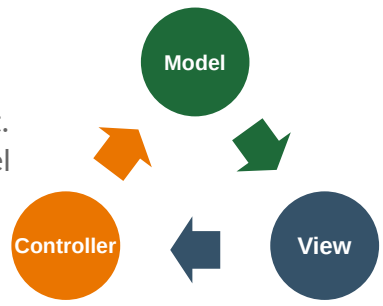
- Un changement dans la base de données nécessitait de ré-écrire le code un peu partout
- Les modifications entraînaient un risque de bogues supplémentaires ou des oublis
- Les développeurs "front-end" devaient partager une partie de leurs codes avec les développeurs "back-end" et cela engendrait donc des problèmes de responsabilités
- etc.

2.1.2 Solution MVC

La proposition du concept MVC est de séparer les domaines de responsabilités en 3 familles :

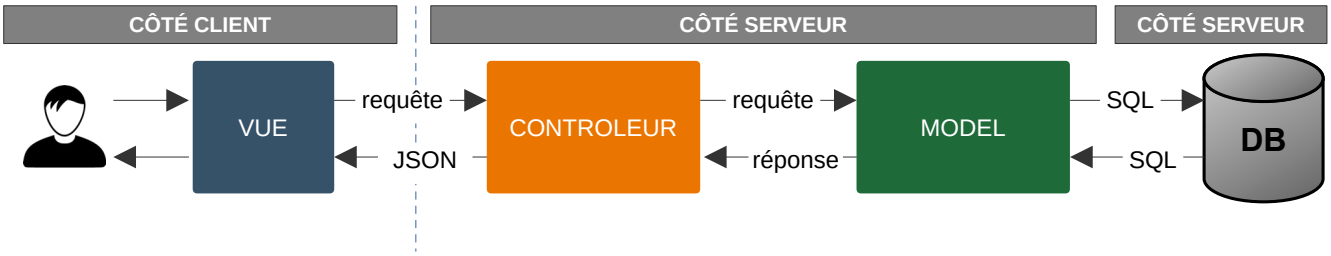
- Le code lié aux **modèles** de données : c'est celui qui est au plus près des données (et donc des bases de données). Il gère la lecture et l'écriture dans les bases, en faisant le lien avec les paramètres transmis par les contrôleurs.
- Le code correspondant aux **vues** : ce code est au plus près de l'utilisateur et gère l'affichage des données et des choix disponibles à l'utilisateur. Quand l'utilisateur clique ou agit sur un élément, on appelle le contrôleur correspondant.
- Le code permettant le **contrôle** des actions : ce code se situe entre le modèle et la vue, permettant de récupérer des paramètres, les traiter (filtrer, nettoyer, modifier, corriger) pour appeler une fonction dans les modèles.

L'une des représentations les plus classiques est celle-ci :



Il s'agit d'une représentation théorique, qui illustre le concept. Toutefois, cet aspect théorique masque le fonctionnement réel et peut s'avérer trompeur.

Une représentation plus réaliste pourrait-être celle-ci :



Cependant, il ne faut pas oublier les contraintes techniques : si la vue correspond à l'affichage côté client, l'ensemble des fichiers reste présent côté serveur.

Ainsi, pour lancer une action, l'utilisateur appelle un contrôleur qui va générer une page : ceci est vrai, même dans un code très simple comme l'exemple suivant.

2.1.3 Fonctionnement HTTP

Prenons un exemple extrêmement simple en PHP : un formulaire permet de récupérer un nom et appelle une page PHP capable d'afficher ce nom.

Index.html	Reponse.php

<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Exemple simple</title>
</head>
<body>
  <form **method="GET" action="Reponse.php"**>
  <input type="text" name="nom">
  <input type="submit" value="Envoyer">
  </form>
</body>
</html>
	<?php
	  echo "<html><head>";
	  echo "<title>Réponse du serveur</title>"
	  echo "</head><body>"
	  echo "Bonjour ".**$_GET["nom"]**;
	  echo "</body></html>"
	?>

Lorsque l'utilisateur va cliquer sur le bouton [Envoyer], le navigateur va prendre les paramètres du formulaire et les envoyer au serveur. Pour cela, il utilise la méthode `GET` et indique au serveur que le script qui doit recevoir ces données est "Reponse.php".

Le script PHP s'exécute et récupère la valeur du champ "nom" dans la super-variable `$_GET`. Il génère une page HTML de réponse qui sera retransmise par le service web (Apache, IIS, NGINX ou autre).

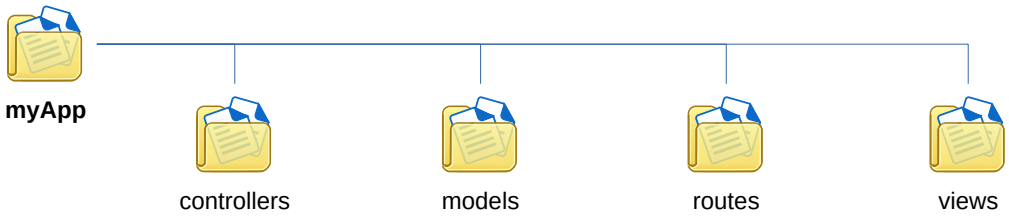
La vue est donc toujours présente sur le serveur, le design pattern MVC est donc un concept permettant de ranger les fichiers sur le serveur, tout en permettant les interactions.

2.2 Utilisation MVC sous Node.JS

2.2.1 MVC en général

Le concept MVC s'applique principalement pour les applications web, mais peut toucher les autres applications.

Dans un projet respectant ce design, on trouvera régulièrement les dossiers suivants :



Selon le langage (ou le framework employé), le dossier "routes" peut ne pas exister : il contient le ou les fichiers qui géreront le fonctionnement de l'application, en servant de point d'accès principal et en choisissant le contrôleur à appeler en fonction de l'URL. Nous verrons plus en détails le routeur, lors de l'utilisation des API Restful.

En PHP, pour remplacer le routeur, il est fréquent d'utiliser un mot clé transmis par méthode `GET` pour choisir le contrôleur. L'URL est alors de la forme :

https://localhost/index.php?action=validation

Dans le code PHP, on lit alors la variable `$_GET["action"]` et on détermine quel contrôleur appeler, comme dans l'extrait ci-dessous :

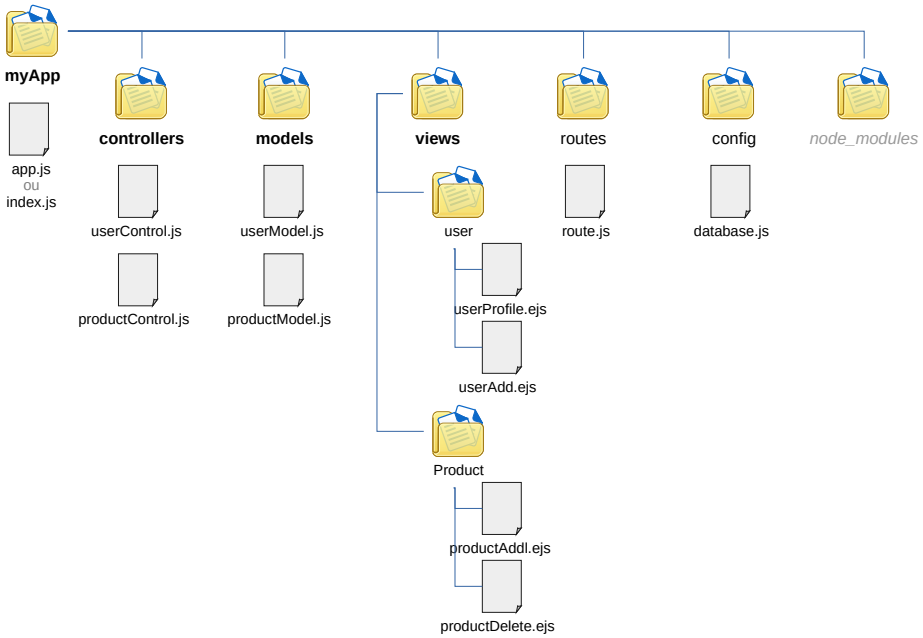
```
<?php
require('controller/controller.php');

if (isset($_GET['action'])) {
    // Validation
    if ($_GET['action'] == 'validation' and isset($_SESSION['id']) and $_SESSION['canValidate']
== 1) {
        listValidation();

        // Update validation
    } else if ($_GET['action'] == 'updateValidation' and isset($_SESSION['id']) ) {
        if (!empty($_POST['valide'])) {
            validation($_POST['valide']);
        } else {
            header('Location: index.php?action=validation');
        }
...
```

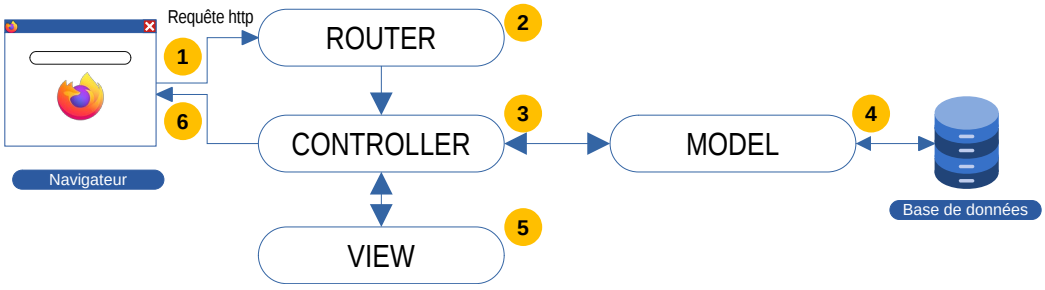
2.2.2 MVC avec NodeJS

NodeJS est son propre serveur : cela n'empêche pas le développement structuré utilisant ce design pattern. En utilisant le framework EJS, on peut retrouver la structure suivante (mais ce n'est pas la seule structure possible) :



Ainsi, pour chaque classe, on trouve généralement un contrôleur, des vues et un modèle.

Le flux MVC est le suivant :



(1) le navigateur génère une requête

(2) le composant routeur analyse le chemin demandé et renvoie les informations au contrôleur associé

(3) le contrôleur effectue un traitement (vérification de sécurité...) et appelle une fonction du modèle

(4) le modèle accède à la base de données et récupère les informations qu'il renvoie au contrôleur

(5) le contrôleur traite (éventuellement) les données et les injectent dans modèle de la vue

(6) la page HTML ainsi générée est renvoyée au navigateur

3 ANNEXES

3.1 SOURCES

https://nodejs.org/fr/

https://www.npmjs.com/package/node-rest-client

https://www.w3schools.com/Node.JS/Node.JS_mysql_select.asp

https://oncletom.io/node.js/chapter-04/index.html

https://www.youtube.com/watch?v=Q8wacXNngXs

https://la-cascade.io/api-les-protocoles/