Aula Node

# Padrão MVC (Model-View-Controller)

**MVC** é o acrônimo de **Model-View-Controller** (em [português](https://pt.wikipedia.org/wiki/Portugu%C3%AAs_brasileiro): **Arquitetura Modelo-Visão-Controle** **- MVC**) é um [padrão de projeto](https://pt.wikipedia.org/wiki/Padr%C3%A3o_de_projeto_de_software) de [software](https://pt.wikipedia.org/wiki/Software), ou padrão de [arquitetura](https://pt.wikipedia.org/wiki/Arquitetura_de_software) de [software](https://pt.wikipedia.org/wiki/Software) formulado na década de 1970, focado no [reuso de código](https://pt.wikipedia.org/wiki/Reutiliza%C3%A7%C3%A3o_de_c%C3%B3digo) e a [separação de conceitos](https://pt.wikipedia.org/wiki/Separa%C3%A7%C3%A3o_de_conceitos) em **três camadas interconectadas**, onde a apresentação dos dados e interação dos usuários ([*front-end*](https://pt.wikipedia.org/wiki/Front-end_e_back-end)) são separados dos métodos que interagem com o [banco de dados](https://pt.wikipedia.org/wiki/Banco_de_dados) (*back-end).*

Esta arquitetura tornou-se popular para projetar [aplicações web](https://pt.wikipedia.org/wiki/Aplica%C3%A7%C3%B5es_web) e até mesmo para aplicações móveis, para desktop e para outros clientes. Linguagens de programação populares como [Java](https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)), [C#](https://pt.wikipedia.org/wiki/C), [Ruby](https://pt.wikipedia.org/wiki/Ruby), [PHP](https://pt.wikipedia.org/wiki/PHP), [JavaScript](https://pt.wikipedia.org/wiki/JavaScript) e outras possuem frameworks MVC populares que são atualmente usados no desenvolvimentos de aplicações web.

## Camada de Modelo ou da Lógica da Aplicação (Model)

*Model* é a ponte entre as camadas Visão (*View*) e Controle (*Controller*), consiste na parte lógica da aplicação, que gerencia o comportamento dos dados através de regras de negócios, lógica e funções. Esta fica apenas esperando a chamada das funções, que permite o acesso para os dados serem coletados, gravados e, exibidos.

É o coração da execução, responsável por tudo que a aplicação vai fazer a partir dos comandos da camada de controle em um ou mais elementos de dados, respondendo a perguntas sobre o sua condição e a instruções para mudá-las. O *Model* sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema que está se tentando resolver. Modela os dados e o comportamento por trás do processo de negócios. Se preocupa apenas com o armazenamento, manipulação e geração de dados. É um encapsulamento de dados e de comportamento independente da apresentação.

Portanto, o ***Model*** representa um **objeto** responsável por possuir a lógica da aplicação.

## Camada de Apresentação ou Visualização (View)

*View* pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É onde os dados solicitados do Modelo (*Model*) são exibidos. É possível ter várias *Views* do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. A *View* também provoca interações com o usuário, que interage com o Controle (*Controller*). O exemplo básico disso é um botão gerado por uma *View*, no qual um usuário clica e aciona uma ação no *Controller*.

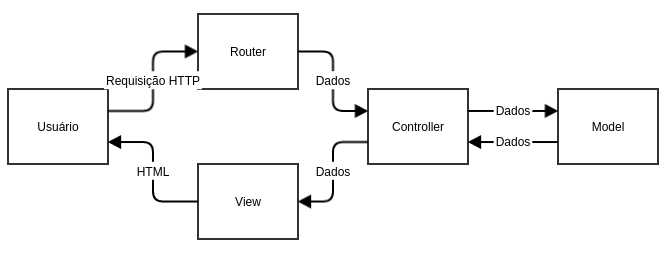
Adiciona os elementos de exibição ao usuário : HTML, ASP, XML, Applets. É a camada de interface com o usuário. É utilizada para receber a entrada de dados e apresentar visualmente o resultado.

## Camada de Controle ou Controlador (Controller)

*Controller* é o componente final da tríade, faz a mediação da entrada e saída, comandando a visão e o modelo para serem alterados de forma apropriada conforme o usuário solicitou através do mouse e teclado. O foco do *Controller* é a ação do usuário, onde são manipulados os dados que o usuário insere ou atualiza, chamando em seguida o *Model*.

O Controle (Controller) envia essas ações para o Modelo (Model) e para a janela de visualização (View) onde serão realizadas as operações necessárias.

Portanto, o controller age tanto no model quanto na view. Ele quem controla as ações do objeto model e realiza os comandos para atualizar a view sempre que ela mudar. Mantém o view e o model separados.

***Nota:***

## MVC - Vantagens e Desvantagens

* ***Vantagens:***
  + Como o modelo MVC gerencia múltiplos views usando o mesmo model é fácil manter, testar e atualizar sistemas compostos;
  + É muito simples adicionar novos clientes apenas incluindo seus views e controles;
  + Torna a aplicação escalável;
  + É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle pois são independentes;
  + Facilita o reuso do código;
  + Melhor nível de sustentabilidade, pois facilita a manutenção da aplicação;
  + Melhor performance, graças a separação em camadas;
  + Fácil transformação da interface, sem que haja necessidade de modificar a camada de negócio;
  + Melhor desempenho e produtividade, graças a estrutura de pacotes modulares;
  + A arquitetura modular permite aos desenvolvedores e designers desenvolverem em paralelo;
  + Partes da aplicação podem ser alteradas sem a necessidade de alterar outras.
* ***Desvantagens:***
  + Necessita de um tempo maior para explorar e modelar o sistema;
  + Requer mão-de-obra especializada;
  + À medida que o tamanho e a complexidade do projeto crescem, a quantidade de arquivos e pastas continuará aumentando também. Os interesses de UI (interface do usuário) (modelos, exibições, controladores) se localizam em várias pastas, que não são formadas em grupos por ordem alfabética.

Fontes:

<https://pt.wikipedia.org/wiki/MVC>

<https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>

<https://www.mav.com.br/mvc-conceito-e-exemplos-em-node-js/>

<https://www.devmedia.com.br/o-que-e-o-ajax/6702>

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/funcoes_assincronas>

<http://callbackhell.com/>