

SUMÁRIO

1. ARCHITECTURAL PATTERNS	1
1.1. DIFERENÇA ENTRE DESIGN PATTERN E ARCHITECTURAL PATTERN	1
2. ARQUITETURA MVC (MODEL VIEW CONTROLE).....	1
2.1. CAMADA CONTROLLER	1
2.2. CAMADA MODEL.....	1
2.3. CAMADA VIEW.....	2
3. ORGANIZAÇÃO DOS DIRETÓRIOS	2
4. USO DO SERVIDOR PHP EMBUTIDO COM MYSQL.....	3
5. USO COMPOSER.....	3

1. ARCHITECTURAL PATTERNS

Padrão de arquitetura se refere a forma como as funcionalidades de um software serão organizadas, estando, portanto, em um nível mais alto, não ligado aos ditos requisitos funcionais de um software, mas sim à estrutura de sua implementação.

A arquitetura de software de um sistema computacional pode ser definida como a suas estruturas, que são compostas de elementos de software, de propriedades externamente visíveis desses componentes, e do relacionamento entre eles. Ou seja, a arquitetura define os elementos de software e como eles interagem entre si.

1.1. DIFERENÇA ENTRE DESIGN PATTERN E ARCHITECTURAL PATTERN

A diferença entre ambos é bastante clara se considerada a definição dos padrões de arquitetura de software, pois os padrões de projetos estão a nível de código, isto é, são métodos e técnicas recorrentes e comumente utilizadas para criar funcionalidades, ou, em outras palavras, para desenvolver os requisitos funcionais do software.

2. ARQUITETURA MVC (MODEL VIEW CONTROLE)

Apesar de muitas pessoas considerarem essa sigla como um padrão de design de interface, na verdade ele é um padrão de arquitetura de software responsável por contribuir na otimização da velocidade entre as requisições feitas pelo comando dos usuários.

O MVC é utilizado em muitos projetos devido à arquitetura que possui, que possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o Model, o Controller e a View, executa o que lhe é definido e nada mais do que isso.

2.1. CAMADA CONTROLLER

A camada de controle é responsável por intermediar as requisições enviadas pelo View com as respostas fornecidas pelo Model, processando os dados que o usuário informou e repassando para outras camadas.

2.2. CAMADA MODEL

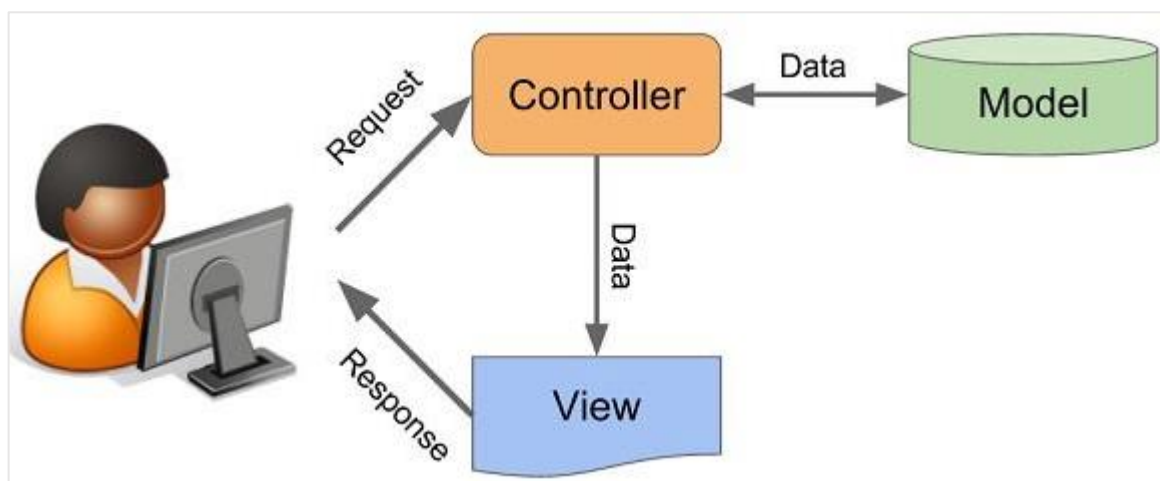
Essa classe também é conhecida como Business Object Model (objeto modelo de negócio). Sua responsabilidade é gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas. O termo “regras de negócio” é utilizado, com alguma recorrência, porque originalmente o padrão MVC foi desenvolvido para atender a outro escopo, em que o termo era cabível.

Mas, posteriormente o padrão passou a ser aplicado no contexto do desenvolvimento Web, e neste, a camada Model, em termos gerais, é a responsável por conversar diretamente com o banco de dados.

2.3. CAMADA VIEW

Essa camada é responsável por apresentar as informações de forma visual ao usuário. Em seu desenvolvimento devem ser aplicados apenas recursos ligados a aparência como mensagens, botões ou telas.

Seguindo nosso processo de comparação o View está na linha de frente da comunicação com usuário e é responsável transmitir questionamentos ao controller e entregar as respostas obtidas ao usuário. É a parte da interface que se comunica, disponibilizando e capturando todas as informações do usuário.



3. ORGANIZAÇÃO DOS DIRETÓRIOS

O padrão MVC descreve apenas uma parte do projeto, e não sua totalidade. No entanto, introduz, por si mesmo, um ideal de organização de pastas.

Não existem padrões obrigatórios, mas sim alguns muitos bem conceituados e que com frequência são utilizados. O próprio Laravel, um dos frameworks PHP mais utilizados, se não o mais valorizado, traz em sua documentação uma estruturação para a disposição das pastas de projetos. Ver em: <https://laravel.com/docs/5.3/structure#the-root-directory>

Uma estrutura básica seria a criação de um diretório raiz do projeto, por exemplo, "Projeto_x", e em seguida duas subpastas, uma chamada "App" e outra "Public", tendo a primeira outras três, cada uma correspondendo a uma camada do MVC, e a segunda, "Public" contendo o arquivo principal "index".

Para complementar, um guia da estruturação de diretórios, do Laravel, descrita por um desenvolvedor: <https://blog.especializati.com.br/estrutura-do-framework-php-laravel/>

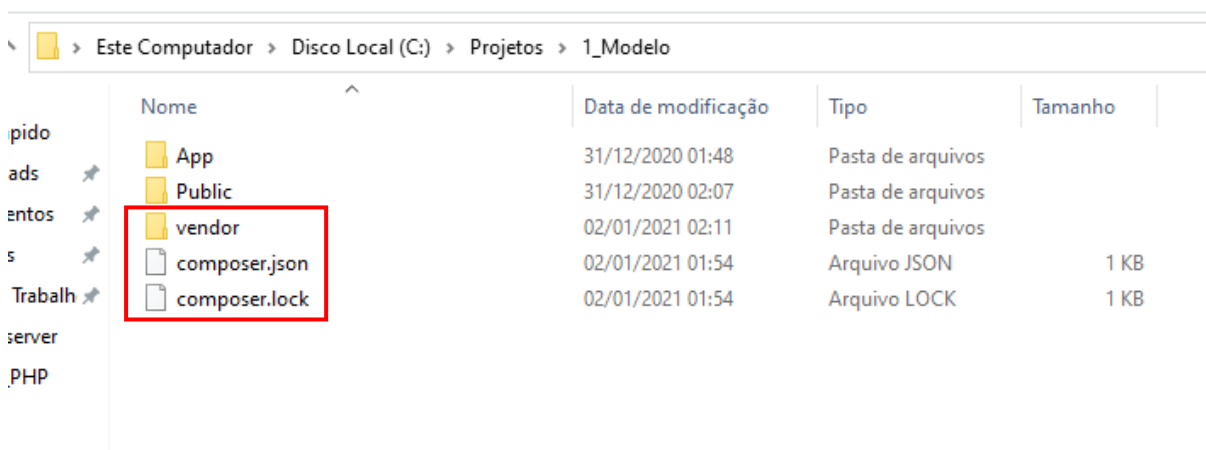
4. USO DO SERVIDOR PHP EMBUTIDO COM MYSQL

Com o servidor do próprio PHP, a produção de aplicações com o padrão MVC se torna mais eficiente e produtiva, dado que cada projeto estará em um diretório próprio. Na verdade, de forma geral, não só no contexto do MVC, utiliza-lo ao invés de pacotes prontos, como XAMPP e WAMP, é preferível.

5. USO COMPOSER

Para a comunicação entre as camadas, os recursos diversos, também externos à aplicação, o ideal é por meio da utilização do Composer. Do contrário, com muita frequência deverão ser escritas as funções de require ou include.

Com Composer, a comunicação é estabelecida automaticamente, e de forma muito mais eficiente. Abaixo uma estrutura de diretórios MVC com Composer sendo utilizado:



	Nome	Data de modificação	Tipo	Tamanho
	App	31/12/2020 01:48	Pasta de arquivos	
	Public	31/12/2020 02:07	Pasta de arquivos	
	vendor	02/01/2021 02:11	Pasta de arquivos	
	composer.json	02/01/2021 01:54	Arquivo JSON	1 KB
	composer.lock	02/01/2021 01:54	Arquivo LOCK	1 KB

Para saber mais sobre o composer, e entender o seu funcionamento [clique aqui](#). Ainda, algo que não é citado nesta fonte, deve-se enfatizar que o composer utiliza namespaces para que na aplicação elementos sejam mapeados e utilizados em diferentes segmentos. Para entender o que é em si, um namespace, [clique aqui](#).

