MVC

PROF. ME. HÉLIO ESPERIDIÃO

DESIGN PATTERNS

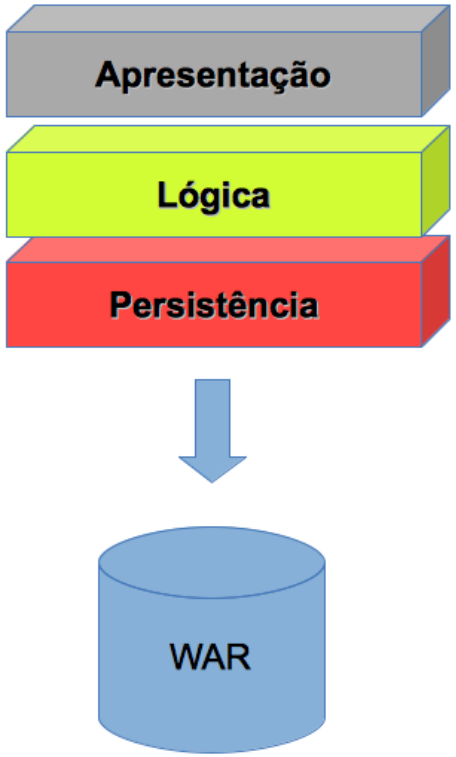
- O termo padrões de projeto ou *Design Patterns*, descreve soluções para problemas recorrentes no desenvolvimento de sistemas de software.

- É uma forma padrão de organizar as classes e objetos, onde são compartilhados conhecimentos sobre orientação objeto aplicados a problemas que acontecem em diversos cenários de desenvolvimento de software.

POR QUE DESIGN PATTERN?

- Com o Design Pattern você terá vários benefícios dentre eles são, código mais enxuto, limpo, organizado, aumentar a qualidade e diminuir a complexidade do seu código.

Aplicação monolítica



A aplicação é um grande mononíto. Mantida por uma única equipe. Distribuída como um todo

APLICAÇÕES MONOLÍTICAS

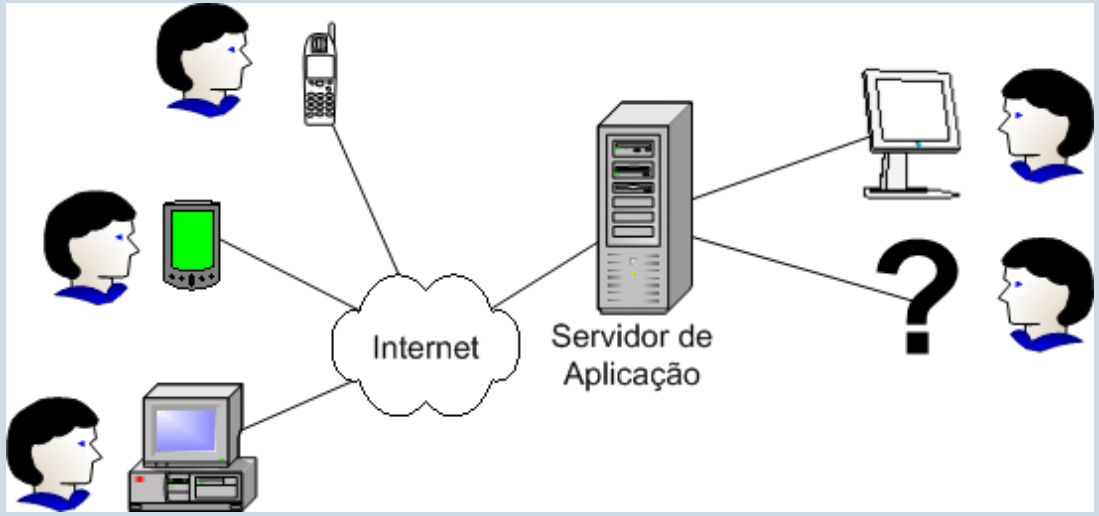
- Na época dos computares independentes um aplicativo era desenvolvido para ser usado em uma única máquina.
-
- Este aplicativo continha todas a funcionalidades em um único módulo gerado por uma grande quantidade de linhas de código e de manutenção nada fácil.
- As entradas do usuário , verificação , lógica de negócio e acesso a banco de dados estava presente em um mesmo lugar.

APLICAÇÕES EM DUAS CAMADAS

Cliente-servidor é um modelo computacional que separa clientes e servidores, sendo interligados entre si geralmente utilizando-se uma rede de computadores.

Cada instância de um cliente pode enviar requisições de dado para algum dos servidores conectados e esperar pela resposta.

O servidor disponível pode aceitar tais requisições, processá-las e retornar o resultado para o cliente.

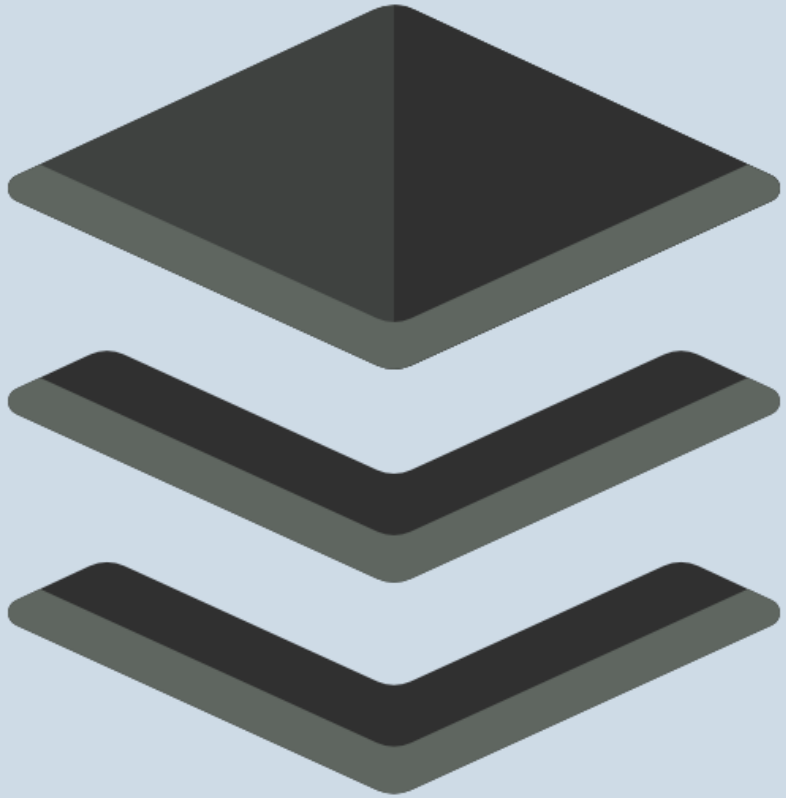


APLICAÇÕES EM TRÊS CAMADAS

- Camada de apresentação (UI)
- Camada de aplicação (business logic)
- Camada de dados

MODEL, O CONTROLLER E A VIEW

- 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.
- A utilização do padrão MVC trás como benefício isolar as regras de negócios da lógica de apresentação



APLICAÇÃO MODERNA

- Possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem que haja a necessidade da alteração das regras de negócios, proporcionando assim muito mais flexibilidade e oportunidades de reuso das classes.

TIPOS DE PROJETOS

- Uma das características de um padrão de projeto é poder aplicá-lo em sistemas distintos.

- **O padrão MVC pode ser utilizado em vários tipos de projetos** como, por exemplo, desktop, web e mobile.

COMUNICAÇÃO

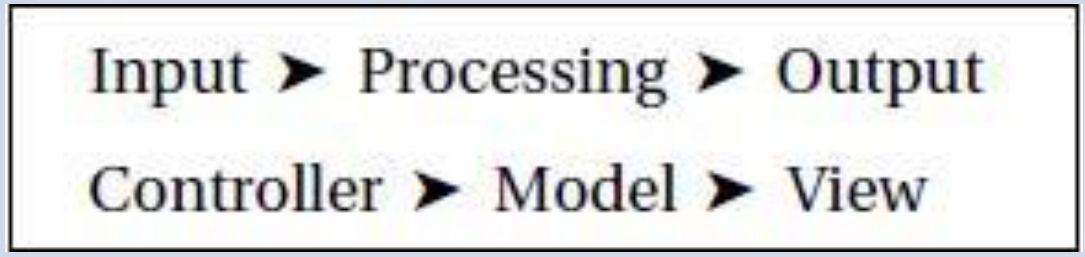
- A comunicação entre interfaces e regras de negócios é definida através de um controlador.
- A existência deste controlador que torna possível a separação entre as camadas.

- A interface gráfica irá se comunicar com o **controlador** que por sua vez se comunica com as regras de negócios.

O MVC

- O MVC inicialmente foi desenvolvido no intuito de mapear o método tradicional de entrada, processamento, e saída que os diversos programas baseados em GUI utilizavam.

- No padrão MVC, teríamos então o mapeamento de cada uma dessas três partes para o padrão MVC conforme ilustra a imagem abaixo:



MVC

- **O MVC é como a clássica programação orientada a objetos.**

- Criar objetos que escondem as suas informações e como elas são manipuladas e então apresentar apenas uma simples interface para o mundo.

VANTAGENS

- Possibilita de reescrita da GUI ou do Controller sem alterar o nosso modelo, reutilização da GUI para diferentes aplicações com pouco esforço, facilidade na manutenção e adição de recursos, reaproveitamento de código, facilidade de manter o código sempre limpo, etc.

MODELO

- O modelo (Model) é utilizado para manipular informações de forma mais detalhada, sendo recomendado que.

- Utilize modelos para realizar consultas, cálculos e todas as regras de negócio do nosso site ou sistema.

- É o modelo que tem acesso a toda e qualquer informação sendo essa vinda de um banco de dados, arquivo XML ou arquivos json.

VISÃO

- A visão (view) é responsável por tudo que o usuário final visualiza, toda a interface, informação, não importando sua fonte de origem, é exibida graças a camada de visão.

CONTROLADORA

- Responsável por controlar todo o fluxo de informação que passa pelo site/sistema. É na controladora que se decide “se”, “o que”, “quando” e “onde” deve funcionar.
- Define quais informações devem ser geradas, quais regras devem ser acionadas e para onde as informações devem ir, é na controladora que essas operações devem ser executadas.
- A controladora que executa uma regra de negócio (modelo) e repassa a informação para a visualização (visão). Simples não?

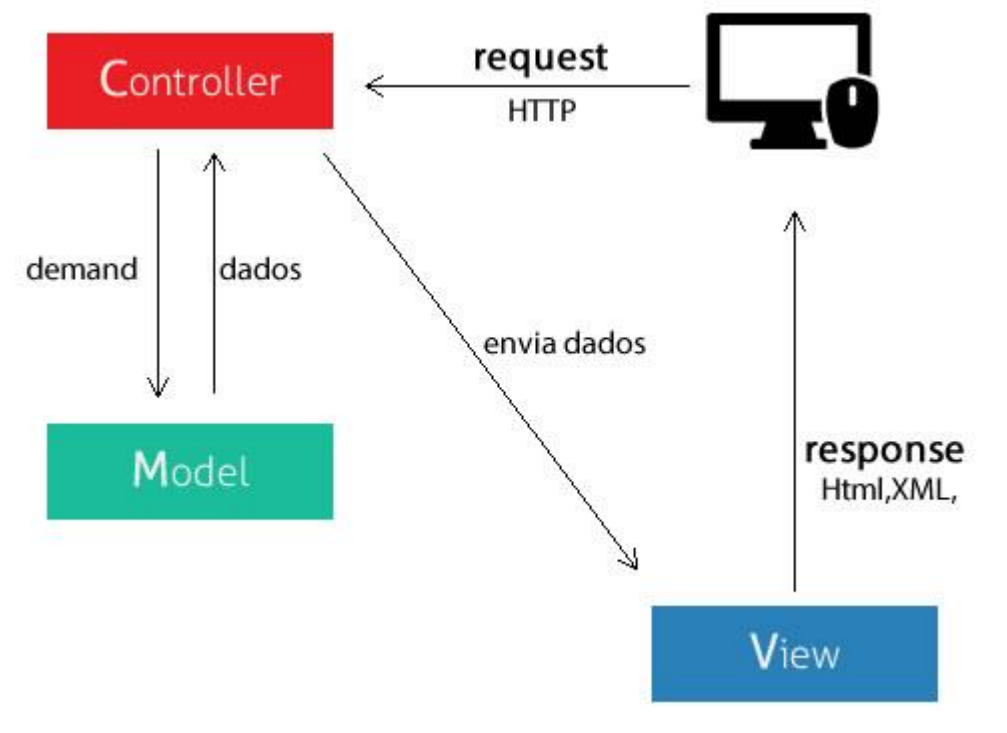
POR QUE UTILIZAR MVC?

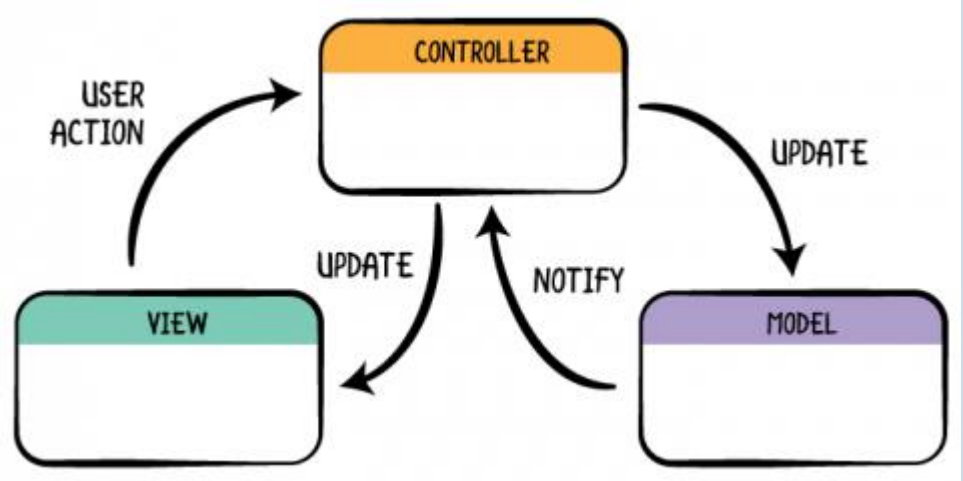
- A arquitetura tem como foco dividir um grande problema em vários problemas menores e de menor complexidade.
- Qualquer tipo de alterações em uma das camadas não interfere nas demais.
- Facilita a atualização de layouts, alteração nas regras de negócio e adição de novos recursos.
- Facilita muito a divisão de tarefas entre a equipe.

VANTAGENS

- Facilita o reaproveitamento de código;
- Facilidade na manutenção e adição de recursos;
- Maior integração da equipe e/ou divisão de tarefas;
- Facilidade em manter o seu código sempre limpo;

APLICAÇÃO WEB OU DESKTOP





COMO FAZER?

- Crie um formulário html
- Utilize ajax para enviar os dados para um arquivo php que servirá de controller.
- O Controller instância as classes necessárias e executa seus métodos, como retorno o controler Envia dados no formato json como resposta a requisição ajax.
- O modelo são apenas as suas classes php
- O view é a interface html da página

BANCO.PHP

```php
<?php
header("Content-type: text/html; charset=utf-8");
class Banco {
    private $host="localhost";
    private $user="root";
    private $password="";
    private $connection=null;
    private $db="agenda";
    function __construct() {
        $this->connection = mysqli_connect($this->host,$this->user,$this->password,$this->db);
        mysqli_set_charset( $this->connection, "utf8");
    }

    function getConnection(){
        return $this->connection;
    }

}
```

- Estado.php

```php
<?php
header("Content-type: text/html; charset=utf-8");
require_once("Banco.php");
class Estado{
    private $idEstado;
    private $estadoNome;
    private $banco;
    function __construct() {
        $this->banco = new Banco();
    }
    public function getAllFromEstado(){
        $result = $this->banco->getConnection()->query("SELECT * FROM estado");
        if($result){
            while ($row = $result->fetch_object()){
                $matrizResposta[] = $row;
            }
        }
        $result->close();

        return $matrizResposta;
    }
    public function setEstadoNome($estadoNome){
        $this->estadoNome=$estadoNome;
    }
    public function setIdEstado($idEstado){
        $this->idEstado=$idEstado;
    }
    public function getIdEstado(){
        return $this->idEstado;
    }
    public function getEstadoNome(){
        return $this->estadoNome;
    }
}
?>
```

CONTROLER_GETALLFROMESTADO.PHP

```php
<?php
    header("Content-type: text/html; charset=utf-8");
    require_once("Estado.php");
    $estado = new Estado();
    $resultado=$estado->getAllFromEstado();
    echo json_encode($resultado,JSON_UNESCAPED_UNICODE);

?>
```

VIEW.HTML

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script>
    function ajax_get_all_Estados(){

    $.ajax({
      method: "POST",
      url: "Controler_getAllFromEstado.php",
      data: {parametro:'null' }
    })
      .done(function( msg ) {
            $("#div_estados").html(msg);
      });

    }
</script>
</head>
<body>
<div id="div_estados" onclick="ajax_get_all_Estados();"> LISTAR TODOS OS ESTADOS</DIV>
</body>
</html>
```