INSTITUTO FEDERAL
Tocantins

Padroes de Programação

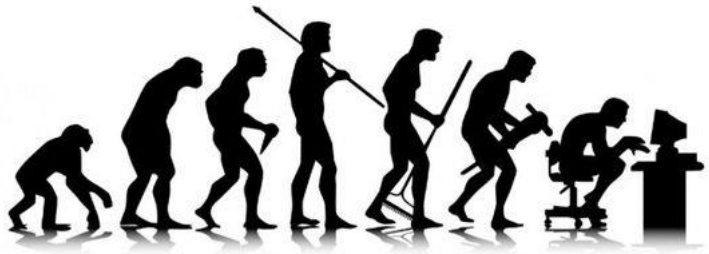
NOVOS CAMINHOS

A historia da computação

OBJETIVO DA AULA

Entender mais sobre as tecnologias e recursos do desenvolvimento de sistemas web.

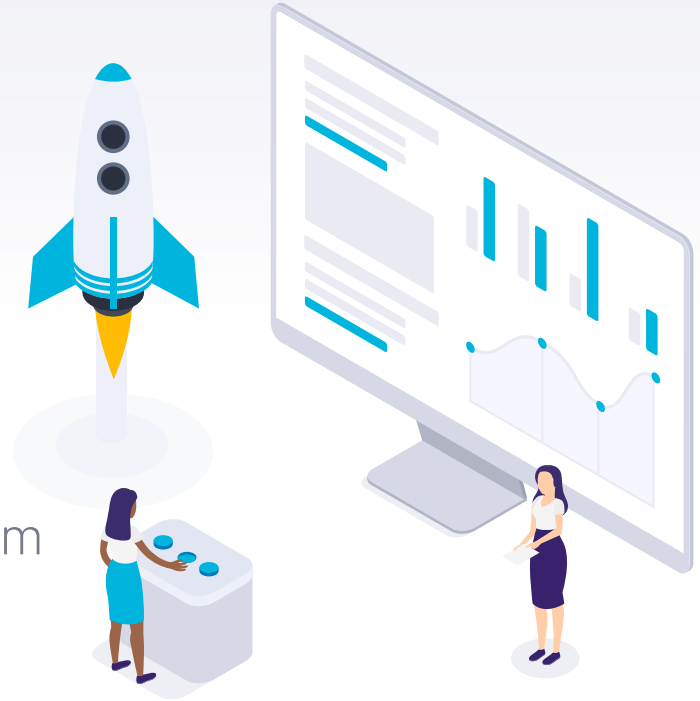
Podendo assim buscar potencializar sua aplicação nas mais diversas tarefas.



1

Padrão MVC

Vamos entender basicamente o que é um
e de que ele é formado.

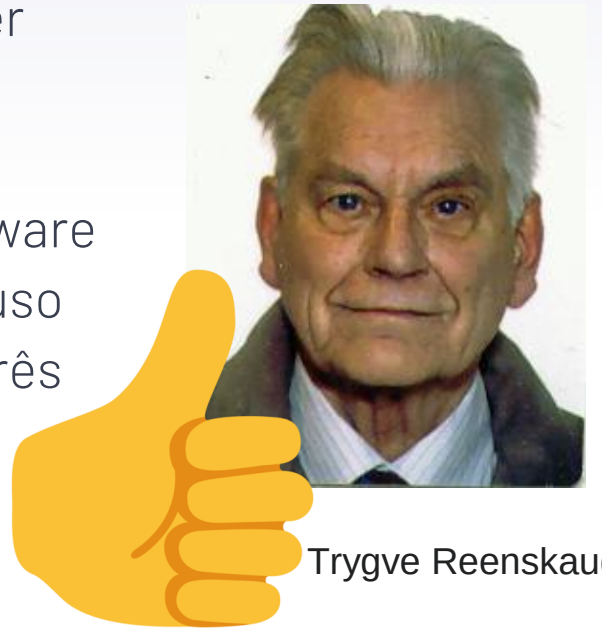


Na pratica com PHP

O MVC (Model, View e Controller) é uma arquitetura ou padrão que divide as responsabilidades em camadas diferentes, a fim de permitir uma melhor organização do código e prover uma solução estruturada

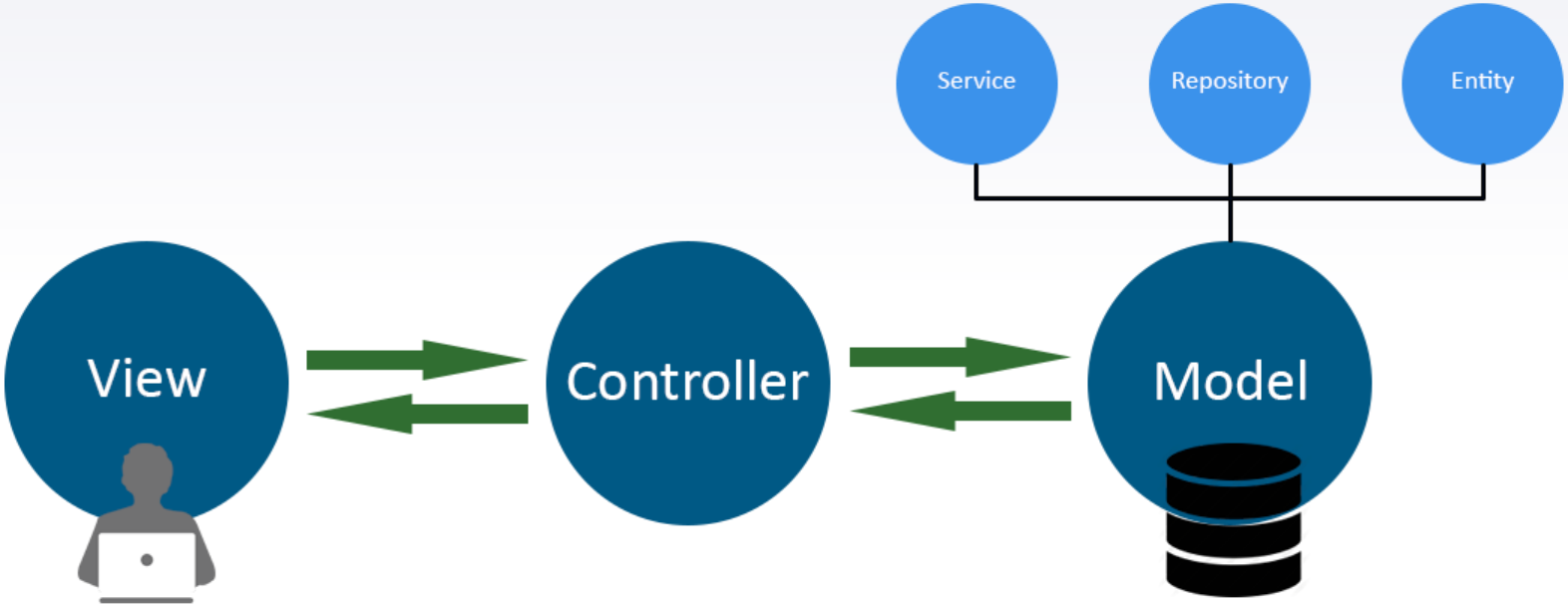
Conceito

- MVC é o acrônimo de Model-View-Controller (em português: Arquitetura Modelo-Visão-Controle - MVC) é um padrão de projeto de software, ou padrão de arquitetura de software formulado na década de 1970,focado no reuso de código e a separação de conceitos em três camadas interconectadas, onde a apresentação dos dados e interação dos usuários (front-end) são separados dos métodos que interagem com o banco de dados (back-end)



Trygve Reenskaug

Conceito



Vantagens

- Escalável
- Reutilizável
- Fácil abstração
- Maior organização
- Fácil detecção de erro

Camada Model

- Model é onde fica a lógica da aplicação. Só isso.

- Vai disparar um e-mail? Validar um formulário? Enviar ou receber dados do banco? Não importa. A model deve saber como executar as tarefas mais diversa, mas não precisa saber quando deve ser feito, nem como mostrar estes dados.

Camada View

- View exibe os dados. Só isso.

- View não é só o HTML, mas qualquer tipo de retorno de dados, como PDF, Json, XML, o retorno dos dados do servidor RESTFull, os tokens de autenticação OAuth2, entre outro. Qualquer retorno de dados para uma interface qualquer (o navegador, por exemplo) é responsabilidade da view. A view deve saber renderizar os dados corretamente, mas não precisa saber como obtê-los ou quando renderizá-los.

Camada Controler

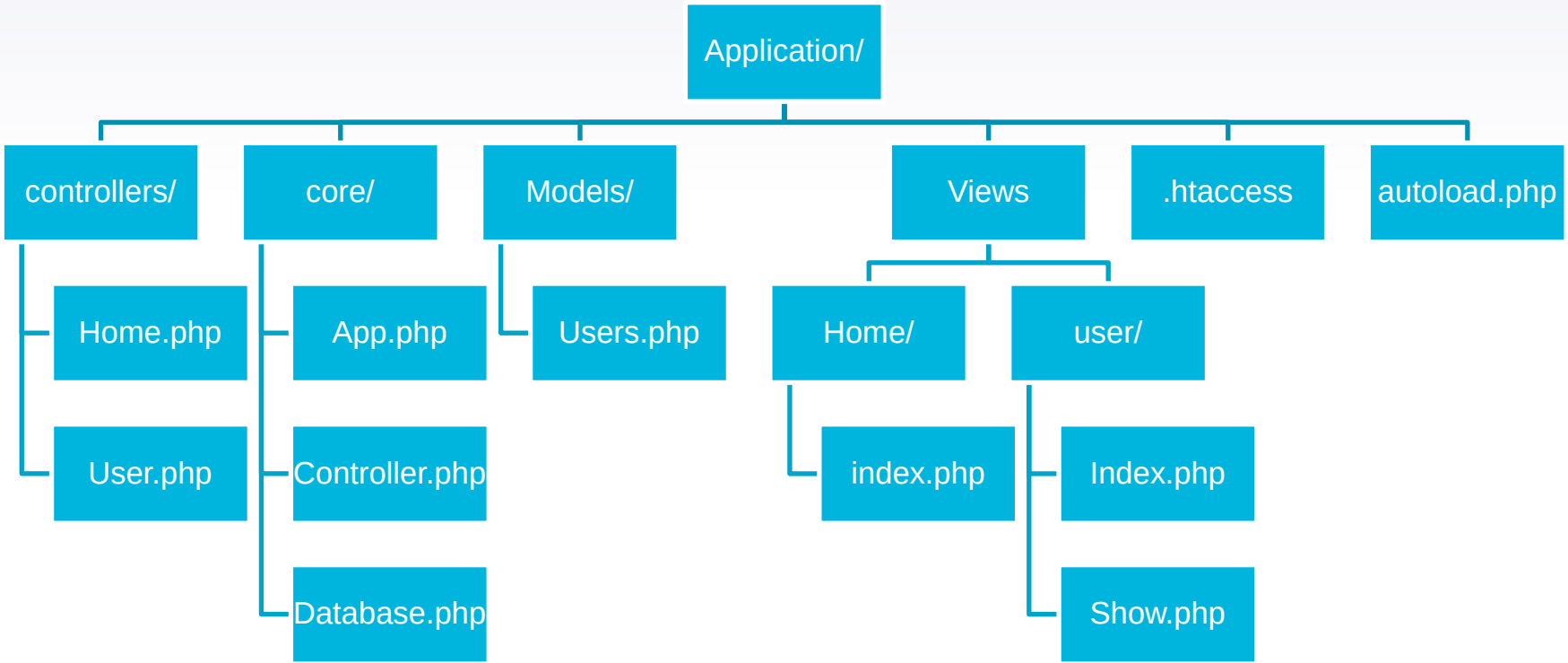
- O controller diz quando as coisas devem acontecer. Só isso.

- É usado para intermediar a model e a _view_ de uma camada. Por exemplo, para pegar dados da model (guardados em um banco) e exibir na view (em uma página HTML), ou pegar os dados de um formulário (view) e enviar para alguém (model). Também é responsabilidade do controller cuidar das requisições (request e response) e isso também inclui os famosos middlewares (Laravel, Slim Framework, Express, Ruby on Rails, etc.). O controller não precisa saber como obter os dados nem como exibi-los, só quando fazer isso.

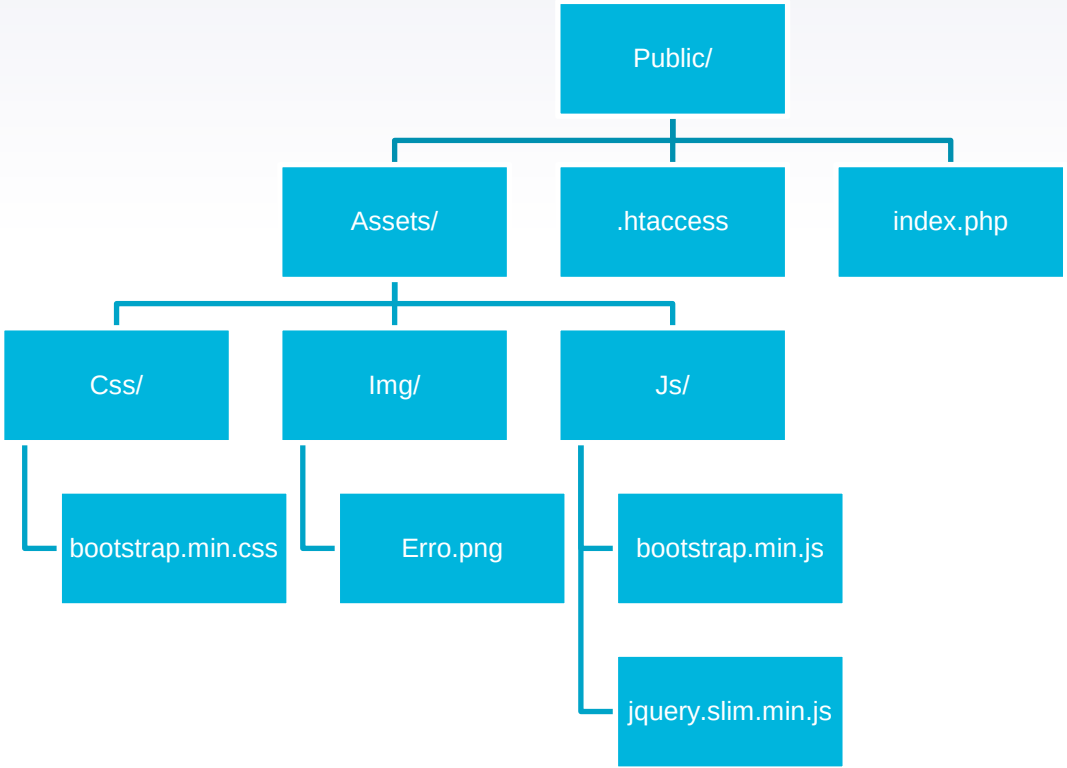
Na pratica

- Estrutura simples
  - src/App/Mvc/Controller.php
  - src/App/Mvc/Model.php
  - src/App/Mvc/View.php
  - index.php

Organização da aplicação



Organização da public



Entendendo a organização

- controllers/ → Este diretório armazenará todos os controladores da aplicação que recebe os dados informados pelo usuário e decide o que fazer com eles e cada método deve realizar uma ação ou chamar view. Além disso, toda classe criada herda os métodos da classe Controller do arquivo armazenado em Application/core/Controller.php que será discutido em breve.
- core/ → Neste diretório será armazenado três arquivos: App.php que é responsavel por tratar a URL decidindo qual controlador e qual método deve ser executado; Controller.php responsável por chamar o model, view e pageNotFound que são herdados para as classes no diretório Application/controllers; E por último, o arquivo Database.php que armazena a conexão com o banco de dados.

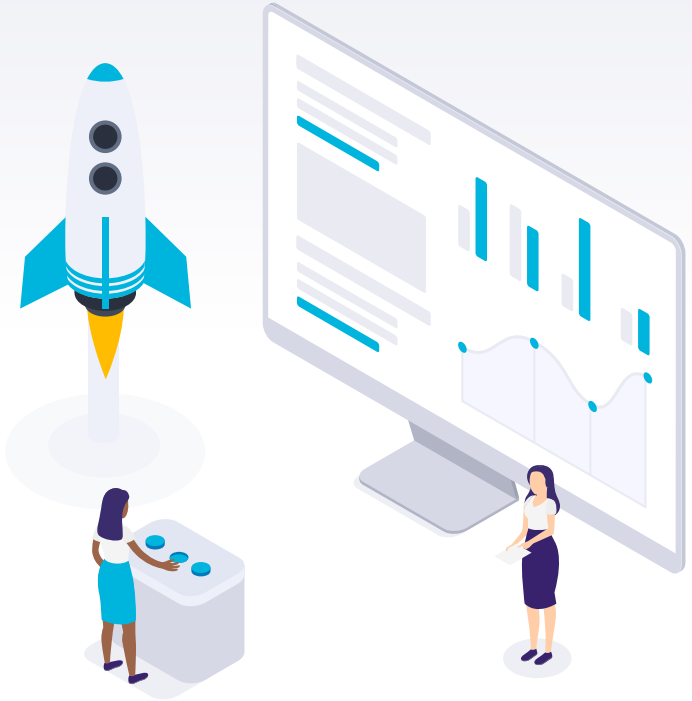
Entendendo a organização

- models/ → Aqui fica a lógica das suas entidades, no nosso caso usaremos classes que irá interagir com o banco de dados e fornecer tais dados para as views.
- views/ → As views serão responsável por interagir com o usuário. Uma das suas principais características é que cada view sabe como exibir um model.
- .htaccess → Neste arquivo, apenas negaremos a navegação no diretório com a opção Options -Indexes.
- autoload.php → Neste arquivo, carregaremos de forma automática todas as classes no diretório Application/.

2

RestFul

Um passo a diante.



Hein?

- REST é a abreviatura de Representational State Transfer, que no em português significa “Transferência de Estado Representacional”. REST é uma de arquitetura de software, que possui algumas boas práticas, definidas por Roy T. Fielding, para criação de aplicações web.

Web Sevices

- Primeiro, a teoria. Um Web service é um conjunto de métodos acedidos e invocados por outros programas utilizando tecnologias Web.
- Segundo, a tradução. Um Web service é utilizado para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente das linguagens de programação utilizadas nessas plataformas.
- Os Web services funcionam com qualquer sistema operativo, plataforma de hardware ou linguagem de programação de suporte Web. Estes transmitem apenas informação, ou seja, não são aplicações Web que suportam páginas que podem ser acedidas por utilizadores através de navegadores Web.

Rest X Restful

- Ao contrário do que muitos imaginam, REST não é necessariamente um protocolo de comunicação.
- Criado por Roy T. Fielding em sua tese de Ph.D., REST é um estilo de arquitetura, ou seja, uma série de restrições que devem ser seguidas no processo de criação de um web service.
- RESTful seria, então, a API que está de acordo com todas as restrições definidas por Roy.
- E é aí que a maior parte das APIs RESTful caem por terra, pois dificilmente elas estão em conformidade com todas essas restrições, elas possuem diferentes graus de maturidade, conforme falaremos mais pra frente.

Qual a vantagem?

Obrigado pela atenção!

Alguma duvida?

Pode perguntar no ClassRoom