# Estrutura básica do CodeIgniter

1. Model
2. View
3. Controller

## Model

Camada de regras de negócio: Manipulação em banco de dados, consultas, registros... Tudo que envolve o banco.

## Controller

Camada intermediária entre VIEW e MODEL. Recebe, interpreta e valida dados, cria e atualiza views, consulta e modifica models.

## View

Interface com o usuário, aparência e código HTML.

# Estrutura e diretórios do CodeIgniter



## Diretório principal

### User\_guide

Armazena o guia do utilizador, manual do codeigniter.

Quando criar um projeto em produção, **excluir o diretório.**

### System

Armazena todos os arquivos do codeigniter.

**NUNCA MEXER.**

### Application

Onde é e será armazenado todos os arquivos da aplicação.

## Subdiretórios (pasta Aplication)

## 

Dentro da pasta application é onde encontramos a estrutura básica do CodeIgniter (MVC).

### Cache

Armazena arquivos que são colocados em cache. Tanto consulta em banco de dados ou arquivos estáticos.

**Deve ter permissão de escrita no servidor que estiver hospedado.**

### Config

Armazena os arquivos de configuração.

Ex: Configuração de banco de dados, rotas de acesso do site, etc..

### Controlles (C)

Armazena todos os controles.

Obrigatório armazenar os controles nessa pasta.

### Core

Diretório para estender o diretório core do codeigniter (núcleo do sistema).

Pode adicionar funções externas criadas, dentro desta pasta.

### Helpers

É um ajudante para facilitar algum tipo de trabalho.

Ex: Para criação de formulários é possível usar o helper e ele facilita para que não se escreva nem uma linha de formulários dentro do HTML.

Armazena os helpers que serão criados, mas ele contém alguns helpers padrões que podem ser acessados pelo comando: system/helpers.

Pode adicionar helpers externos criados, dentro desta pasta.

### Hooks

Armazena modificações ou alterações que estende as funcionalidades do CodeIgniter e também não são funções nativas do core.

### Language

Armazena os arquivos com dicionários de idiomas.

### Libraries

Armazena as bibliotecas criadas para o projeto ou estendidas do core do ci.

### Logs

Caso seja configurado, vai ser usada para configurar os log do sistema.

### Models (M)

Armazena os modos do sistema.

### Third\_party

Armazena códigos de terceiros, como classes que ajudam no desenvolvimento.

Ex: código para gerar gráficos

### Views (V)

Vai armazenar toda a camada VIEW do MVC.

### Index.php

É o arquivos base do projeto, responsável por chamar todas as classes e funções nativas do sistema.

# Usar em diversos projetos

Com o diretório padrão não permite usar em diversos projeto, para isso devemos criar uma nova pasta e colocar todos os diretórios, menos o diretório SYSTEM, nessa nova pasta.

Para que o code igniter volte a funcionar com essa aplicação, vemos entrar na pasta aplicattion/index.php e devemos substituir o comando:

$system\_path = 'system';

Por

$system\_path = '../system';

Fazendo com que ele volte um diretório e encontre a pasta system.

# MVC funcional

## Controllers

O controller welcome.php é o controller padrão do codeigniter, quando se abre o codeigniter ele é usado para trazer a VIEW welcome\_message.php.

Nele contém o método index, por ser index não é necessário chama-lo. Ex:

public function index(){

        $this->load->view('welcome\_message'); 🡪 chama a view

    }

No padrão o codeigniter chama o controller welcome(é possível observar entrando na pasta config/routes.php) , mudando o chamado para o controller desejado, irá trazer o comando do controller.

$route['default\_controller'] = "welcome";

Quando não for o controller padrão do codeigniter, chamamos ele através da URL:



Ex:

Nos controlles foi criado um arquivo .php chamado de base, onde nele foi criado um construdor pai. Basicamente é a base para uma criação de controller:

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed'); //Sempre deixar definido, pois se não achar o diretório vai aparecer a mensagem.

class Base extends CI\_Controller { //A classe sempre deve ter o mesmo nome do arquivo controller

    function \_\_contrruct(){ //construdor do controller, nele é possivel carregar as librari, helpers, models que serão utilizados nesse controller

        parent::\_\_contrruct();//Chamando o construtor da classe pai

    }

    public function index(){

        //metodo padrão de controller

    }

}

/\* End of file welcome.php \*/

/\* Location: ./application/controllers/welcome.php \*/

Ex 2:

Nos controlles foi criado um arquivo .php chamado de exemplo1, onde nele foi criado um construdor pai, e uma function login.

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed'); //Sempre deixar definido, pois se não achar o diretório vai aparecer a mensagem.

class Exemplo1 extends CI\_Controller { //A classe sempre deve ter o mesmo nome do arquivo controller

    function \_\_contrruct(){ //construdor do controller, nele é possivel carregar as librari, helpers, models que serão utilizados nesse controller

        parent::\_\_contrruct();//Chamando o construtor da classe pai

    }

    public function index(){

        echo 'executado método index do controller';

    }

    public function login(){

        echo 'executado método login do controller';

    }

}

/\* End of file welcome.php \*/

/\* Location: ./application/controllers/welcome.php \*/

Para acessar a função index usamos a URL:

<http://localhost/ci/CodeIgniter-2.2.6/essencial/index.php/exemplo1>

Para acessa a função login, usamos a URL:

<http://localhost/ci/CodeIgniter-2.2.6/essencial/index.php/exemplo1/login>

Podemos também passar parâmetros para um método dentro de um controller, colocando ele no terceiro segmento da URL.

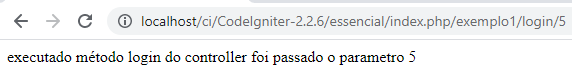
EX:

<https://localhost/ci/CodeIgniter-2.2.6/essencial/index.php/exemplo1/login/5> o 5 seria o parâmetro, por exemplo, de um id de usuário.

E para recuperar esse parâmetro dentro do controller usamos a expressão:

echo $this->uri->segment(3);

Nesse comando, estamos recuperando a posição do 3º segmento.



**OBS: SERÁ UTILIZADO PARA INCLUSÃO E EXCLUSÃO EM BANCO DE DADOS, PODENDO PASSAR O ITEM A SER EXCLUI/INCLUIDO PELO 3° PARAMETRO DA URL.**

## VIEW

No exmplo1 a cima, criamos uma função que chama:

echo 'executado método INDEX do controller';

Mas o padrão MVC diz, que interação ao usuário deve ser chamado nas VIEWS e não no CONTROLLER, por isso trocaremos o comando para:

$this->load->view('exemplo1');

Esse commando é responsavel por chamar a view.

Feito isso, criamos dentro de VIEW o arquivo Exemplo1.php:

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');?> <!--Sempre deixar definido, pois se não achar o diretório vai aparecer a mensagem.\*/ -->

<!DOCTYPE html>

<html lang="pt-BR">

<head>

    <meta charset="utf-8">

    <title>View de exemplo</title>

</head>

<body>

<div id="container">

    <h1>Olá, Mundo!</h1>

    <div id="body">

        <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Minima corporis obcaecati culpa repellendus accusantium fugit sed ab voluptatibus modi! Numquam, voluptatibus harum officia pariatur dolore accusantium fugit quis sunt maxime.</p>

    </div>

    <p class="footer">Page rendered in <strong>{elapsed\_time}</strong> seconds</p>

</div>

</body>

</html>

Posso também usar meu controller para passar parâmetros para minha view.

EX:

Controller

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed'); //Sempre deixar definido, pois se não achar o diretório vai aparecer a mensagem.

class Exemplo1 extends CI\_Controller { //A classe sempre deve ter o mesmo nome do arquivo controller

    function \_\_contrruct(){ //construdor do controller, nele é possivel carregar as librari, helpers, models que serão utilizados nesse controller

        parent::\_\_contrruct();//Chamando o construtor da classe pai

    }

    public function index(){

        $dados['titulo'] = 'Olá, mundo!!!';

        $dados['conteudo'] = ' lorem ipsum solor site amet, consectetur';

        $this->load->view('exemplo1', $dados);

    }

    public function login(){

        echo 'executado método login do controller foi passado o parametro ';

        echo $this->uri->segment(3);

    }

}

/\* End of file welcome.php \*/

/\* Location: ./application/controllers/welcome.php \*/

View

?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');?> <!--Sempre deixar definido, pois se não achar o diretório vai aparecer a mensagem.\*/ -->

<!DOCTYPE html>

<html lang="pt-BR">

<head>

    <meta charset="utf-8">

    <title>View de exemplo</title>

</head>

<body>

<div id="container">

    <h1><?php echo $titulo; ?> </h1>

    <div id="body">

        <p><?php echo $conteudo; ?> </p>

    </div>

</div>

</body>

</html>

## Models

Para criar um model o arquivo deve ser em .php.

Para fins de padronização, quando criarmos um arquivo no model, ao final do nome colocamos \_model.

Ex:

Exemplo1\_model.php

A criação de um model é parecida com a criação de um controller, mudando a class inicial de

class Exemplo1 extends CI\_Controller {

para

class Exemplo1\_model extends CI\_ModeL {

Mas no MODEL não definimos um método index, pois é o CONTROLLER o responsável por chamar o método. Ficando dessa forma:

<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed'); //Sempre deixar definido, pois se não achar o diretório vai aparecer a mensagem.

class Exemplo1\_model extends CI\_ModeL { //A classe sempre deve ter o mesmo nome do arquivo

    function \_\_contrruct(){ //construdor do controller

        parent::\_\_contrruct();//Chamando o construtor da classe pai

    }

    }

    public function salvar(){

        echo 'executado método salvar do model '; //Somente para fins de aprendizagem, nunca deve ser executado um metodo de printar no model

    }

}

/\* End of file welcome.php \*/

/\* Location: ./application/controllers/welcome.php \*/

Para executar o MODEL devemos carregar ele onde for necessário sua execução ou carregar ele para toda a aplicação, através do autoload (autoload só servirá se quisermos usar esse método em toda a aplicação).

Para carregar esse método para toda a aplicamos, devemos ir na pasta condig/autoload.php e acrescentar no array nome dele.

Exemplo:



**OBS: USAR O AUTOLOAD SOMENTE SE O MODEL FOR APLICADO EM TODA A PRODUÇÃO, CASO CONTRARIO USAR O MODEL SOMENTE NA APLICAÇÃO.**

Se for aplicado o model no método construtor, irá aplicar para todos os métodos, caso seja somente para um método, carregamos o model dentro do método específico.

Ex Construtor(controller):