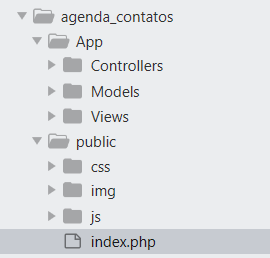
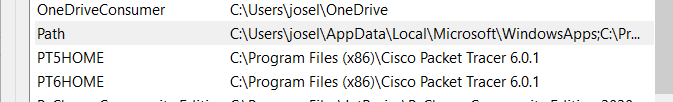
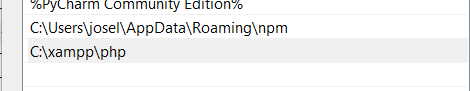
1. Crie uma pasta com o nome do projeto
2. Crie a seguinte estrutura de pastas e arquivos



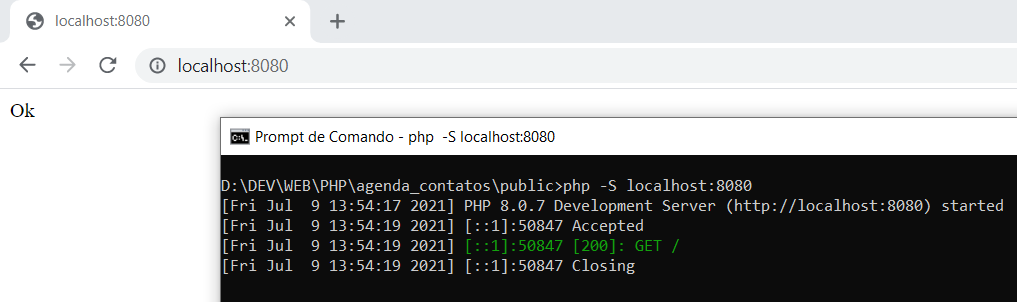
1. Crie um código de teste no arquivo index.php
2. Adicione o PHP nas variáveis de ambiente do usuário logado





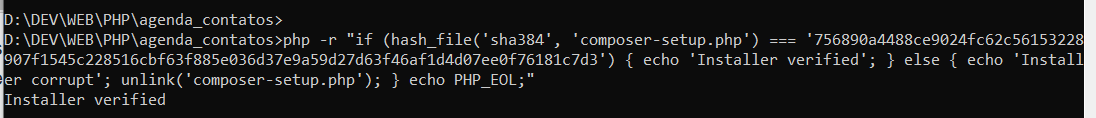
1. Abra o CMD e navegue até a pasta do projeto

cd D:\DEV\WEB\PHP\agenda\_contatos\public

1. Digite: php -S localhost:8080
2. Acesse o endereço no Browser e teste
3. Faça o download do composer no projeto

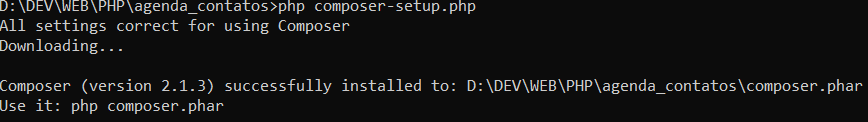
<https://getcomposer.org/>

D:\DEV\WEB\PHP\agenda\_contatos>php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

1. Testando
2. 

**Download**

php composer-setup.php



**Remova**

php -r "unlink('composer-setup.php');"

**Crie o arquivo composer.json e adicione**

{

"name": "vendor/agenda\_contatos",

"require": {

"php": ">= 7.0"

},

"authors": [

{

"name": "Thiago",

"email": "thiago@gmail.com"

}

"autoload": {

"psr-4": {

"App\\": "App/",

"MF\\": "vendor/MF/"

}

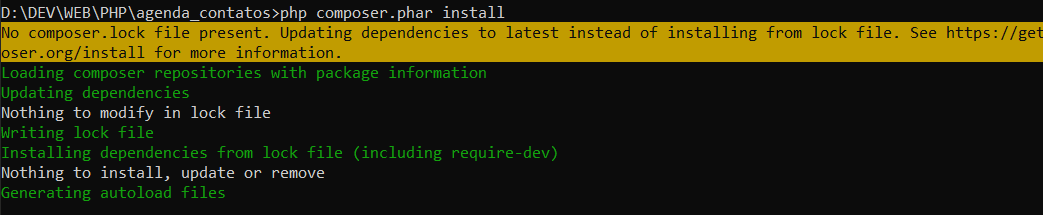
}

]

}

**Instale**

php composer.phar install



Foi criado o diretório vendor

Crie a pasta MF na pasta vendor

O script autoload faz o carregamento automático de todas as classes

Require\_once do autoload no index

<?php

require\_once "../vendor/autoload.php";

Execute e teste novamente

**Rotas**

Na pasta App crie um arquivo Route.php e adicione

Namespace

Namespace App;

Classe

Class Route {

}

Instancie a classe Route no index.php

$route = new \App\route;

**Crie a função getUrl dentro da classe para obter a url**

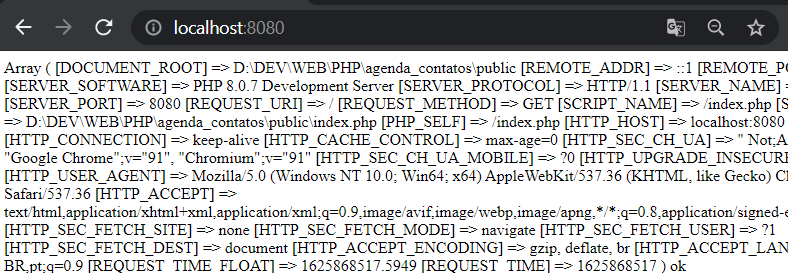
public function getUrl() {

return $\_SERVER;

}

Teste a função getUrl dentro do arquivo index.php

print\_r($route->getUrl());

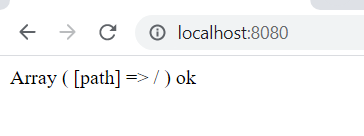


Veja que é um array com vários elementos, o que nos interessa é o [REQUEST\_URI], veja que o valor dele é /, ou seja a raiz

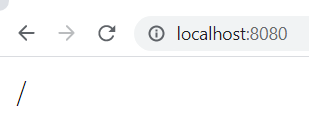
**Parse url na função getUrl**

return parse\_url($\_SERVER['REQUEST\_URI']);

Teste novamente

Adicione PHP\_URL\_PATH para obter a url pura

return parse\_url($\_SERVER['REQUEST\_URI'], PHP\_URL\_PATH);

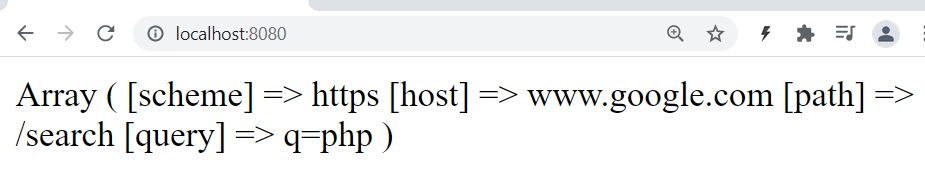


**Path e query**

Em uma url completa você encontra essas duas partes

Digite o código seguinte e teste

return parse\_url('https://www.google.com/search?q=php', PHP\_URL\_PATH);

**Definindo as rotas da nossa aplicação**

Crie a função initRoutes

public function initRoutes() {

$routes['home'] = array (

'route' => '/',

'controller' => 'indexController',

'action' => 'index'

)

$routes['contatos'] = array (

'route' => '/contatos',

'controller' => 'indexController',

'action' => 'contatos'

)

}

**Controllers**

Crie o arquivo indexController.php

namespace App\Controllers;

class IndexController{

public function index(){

echo "Index";

}

public function contatos() {

echo "Contatos";

}

}

Crie as funções getRoutes e setRoutes em Route.php

public function getRoutes() {

return $this->routes;

}

public function setRoutes(array $routes) {

$this->routes = $routes;

}

Crie o método construtor

public function \_\_construct() {

$this->initRoutes();

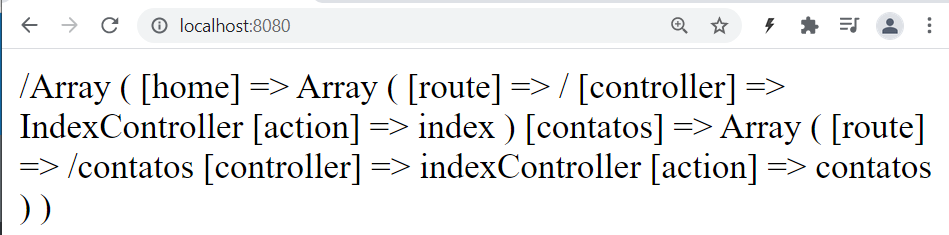
}

Abaixo das rotas chame a função setRoutes e passe o array de rotas

$this->setRoutes($routes);

Testando no index

print\_r($route->getRoutes());



Crie um método run na no Route.php

public function run($url) {

echo $url;

}

Chame o método no construtor

public function \_\_construct() {

$this->initRoutes();

$this->run($this->getUrl());

}

Teste novamente

Percorrendo o array de rotas

Verificando se uma das rotas existentes foi escolhida

if ($url === $route['route']) {

Instância da classe dinamicamente

$class = "App\\Controllers\\".ucfirst($route['controller']);

}

O run completo

public function run($url) {

//Percorre o array de rotas

foreach ($this->getRoutes() as $key => $route) {

if ($url === $route['route']) {

//Define a classe de acordo com o arota

$class = "App\\Controllers\\".ucfirst($route['controller']);

//Instancia dinamicamente a classe

$controller = new $class;

//Acao

$action = $route['action'];

$controller->$action();

}

}

}

Teste novamente

**Melhorando as rotas**

No diretório MF crie um chamado init

Crie arquivo Bootstrap.php não confundir com o framework bootstrap

Usado para scripts de inicialização

Defina o namespace

namespace MF\Init;

Crie uma classe abstrata pois não vai ser instanciada

Mova o atributo $routes, o construtor e os métodos get e set, o run e o getUrl

abstract class Bootstrap {

private $routes;

public function \_\_construct() {

$this->initRoutes();

$this->run($this->getUrl());

}

}

Altere a visibilidade do método run e do getUrl para protected

Defina o Método initRoutes como abstrato no Bootstrap

Quando herdado deverá ser implementado na classe filha

abstract protected function initRoutes();

Ajuste o método também na Route

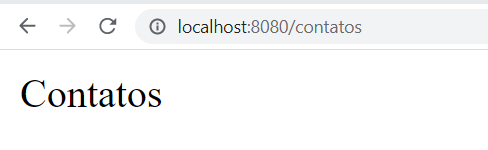
protected function initRoutes() {

Importe o namespace no Route

use MF\Init\Bootstrap;

Extenda a classe Bootstrap na classe Route

class Route extends Bootstrap{



Testando

**Views**

Isolar o conteúdo de exibição dos controllers

Crie a pasta index na pasta Views para exibição exibir tudo que vem do IndexController

Crie os arquivos index.phtml e contatos.phtml

Phtml é extensão de uma combinação de php com html

Retire os conteúdos dos controllers e adicione nos arquivos .phtml

Faça os require\_once das views nos controllers

require\_once "../App/Views/index/index.phtml";

Cuidado com o endereço, a base é o arquivo index.php

Teste

Crie um array para testes acima dos require\_once

$dados = array('Nome', 'Idade', 'CEP');

Recupere os dados no arquivo .phtml

foreach ($dados as $key => $dado) {

echo $dado."<br>";

}

Crie o método render e altere as funções que retornam as views

Para acessar os dados passe-os como parâmetro

public function contatos() {

$dados = array('Nome', 'Idade', 'CEP');

$this->render("contatos", $dados);

}

public function render($view, $dados) {

require\_once "../App/Views/index/".$view.".phtml";

}

Teste

**Tornando o parâmetro do diretório dinâmico**

Extraia a parte inicial do nome da classe

//Obter a classe

$classeAtual = get\_class();

//Substitua o conteudo 'App\\Controllers\\ por vazio para obter apenas o nome do controller

$classeAtual = str\_replace('App\\Controllers\\', '', $classeAtual);

//Substitua o termo Controller por vazio para obter o nome unico

$classeAtual = strtolower(str\_replace('Controller', '', $classeAtual));

//Concatene

require\_once "../App/Views/".$classeAtual."/".$view.".phtml";

**Melhorando a forma de passar os dados para as views**

No index crie um atributo privado e crie o método construtor

private $view;

public function \_\_construct() {

$this->view = new \stdClass();

}

stdClass é uma classe nativa para criar objetos padrões (vazios)

Crie atributos dinamicamente com $this->view e assim podemos remover o parâmetro

public function contatos() {

$this->view->dados = array('Nome', 'Idade', 'CEP');

$this->render("contatos");

}

O view que criamos está disponível em qualquer método

No html digite:

foreach ($this->view->dados as $key => $dado) {

echo $dado."<br>";

}

Teste e está tudo ok

**Abstração dos controllers**

Diretorio MF crie uma pasta com nome Controller

Dentro crie o arquivo Action.php

Namespace e Crie a classe Action

namespace MF\Controller;

abstract class Action {

}

Transfira para a classe, o $view, o construtor e o render

private $view;

public function \_\_construct() {

$this->view = new \stdClass();

}

public function render($view) {

$classeAtual = get\_class($this);

$classeAtual = str\_replace('App\\Controllers\\', '', $classeAtual);

$classeAtual = strtolower(str\_replace('Controller', '', $classeAtual));

echo "<br>";

require\_once "../App/Views/".$classeAtual."/".$view.".phtml";

}

No IndexController use o namespace Controller chamando a classe Action

use MF\Controller\Action;

Extenda a classe IndexController com Action

class IndexController extends Action {

Ajuste a visibilidade do método render e atributo $view para protected

protected $view;

protected function render($view) {

Teste novamente

**Definindo e reutilizando layouts**

Na pasta Views crie um arquivo layout.phtml

Adicione o conteúdo comum a todas as páginas

Insira as views dentro do layout

<?= $this->content() ?>

Na classe Action

Recorte toda a lógica do método render, crie o método contente() e cole

$classeAtual = get\_class($this);

$classeAtual = str\_replace('App\\Controllers\\', '', $classeAtual);

$classeAtual = strtolower(str\_replace('Controller', '', $classeAtual));

echo "<br>";

require\_once "../App/Views/".$classeAtual."/".$view.".phtml";

Renderize no método render

Crie um atributo padrão

protected function render($view) {

$this->view->page = $view;

require\_once "../App/Views/layout.phtml";

}

Altere a linha

require\_once "../App/Views/".$classeAtual."/".$this->view->page.".phtml";

Teste

Pode define quantos layouts quiser

Passe qual o layout como parâmetro

public function index(){

$this->render("index", "layout");

}

Na Action

protected function render($view, $layout) {

$this->view->page = $view;

require\_once "../App/Views/".$layout.".phtml";

}

Tratando erro de layout inexistente

if(file\_exists("../App/Views/".$layout.".phtml")) {

require\_once "../App/Views/".$layout.".phtml";

} else {

$this->content();

}

Teste

public function index(){

$this->render("index", "layout3");

}

Assim ele exibe apenas o conteúdo da view

**Conexão com DB**

No diretório App crie o arquivo Connection.php

Defina o namespace

namespace App;

Crie a classe

class Connection {

public static function getDb() {

try {

$conn = new \PDO(

"mysql:host=localhost;dbname=agenda\_contatos;charset=utf8","root", "admin"

);

return $conn;

} catch(\PDOException $e) {

echo "Erro na conexão: ".$e;

}

}

}

As contrabarras em $conn = new \PDO( e catch(\PDOException $e tem que ser usadas para indicar que o PHP deve pesquisa as classes na raiz.

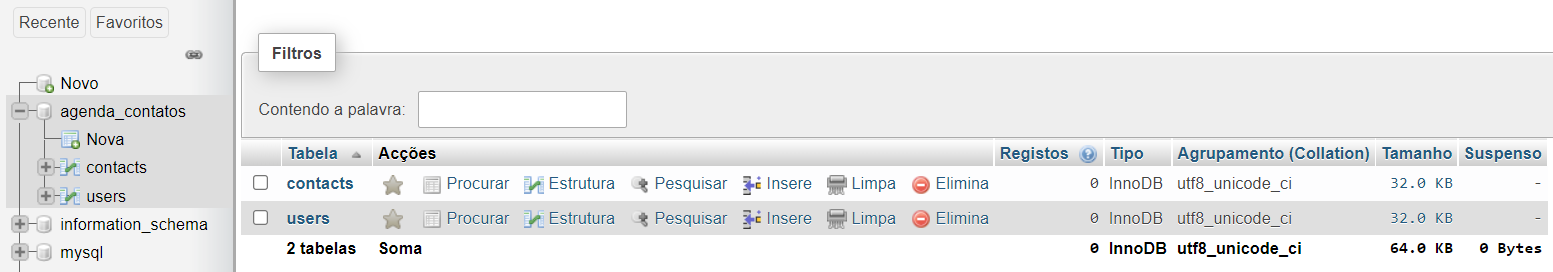
**Crie o banco**



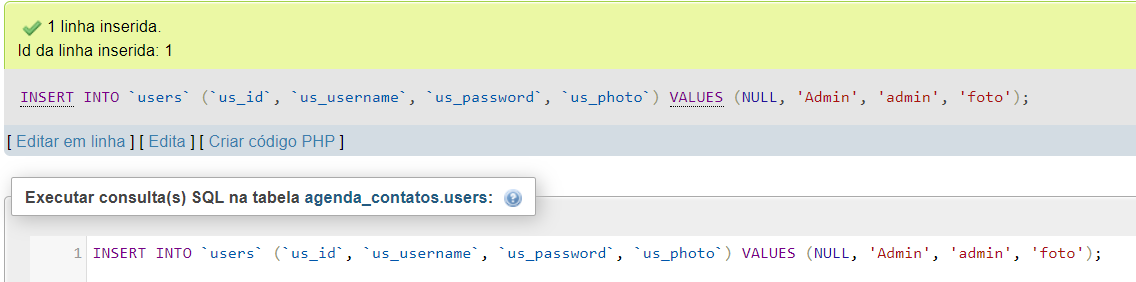
Copie o conteúdo do arquivo agenda.sql e cole no Editor de SQl do PhpMyadmin

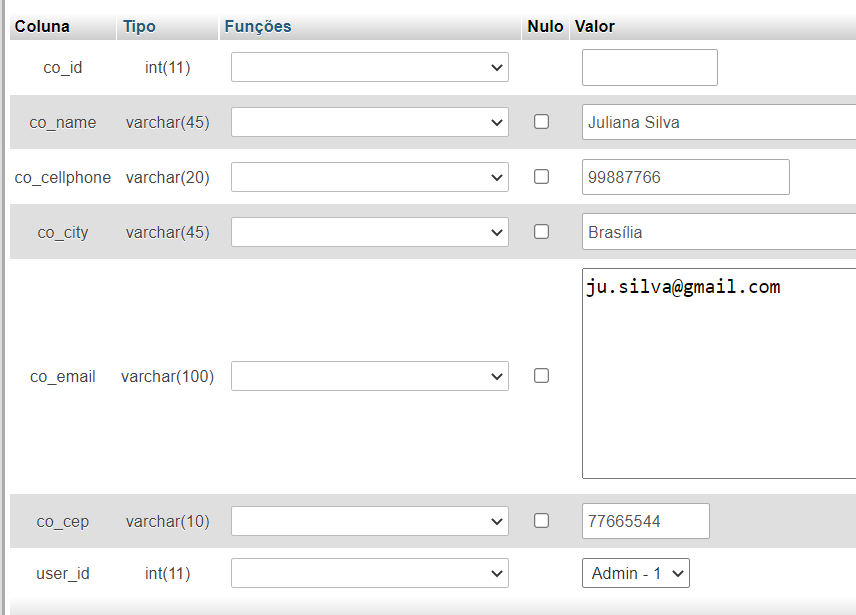


As tabelas serão geradas



Salve alguns dados para teste







**Vamos para o index controller**

use App\Connection;

Chame o método

Connection::getDb();

Instancia do modelo

use App\Models\Contato;

$contato = new Contato($conn);

Métodos para manipular dados

$produto->getContatos();

Crie ele no no Contato.php

public function getContatos() {

return array('Nome', 'Cidade');

}

Melhore a chamada dos dados

$contatos = $contato->getContatos();

$this->view->dados = $contatos;

**Models**

Crie o diretório Models dentro do App

Crie um arquivo com o nome Contato.php e adicione

namespace App\Models;

class Contato {

protected $db;

public function \_\_construct(\PDO $db) {

$this->db = $db;

}

public function getContatos() {

return array('Nome', 'Cidade');

}

}

Criando a query

No arquivo Contato, no metodo getContatos crie

$sql = "SELECT co\_name, co\_cellphone, co\_city, co\_email, co\_cep FROM contacts";

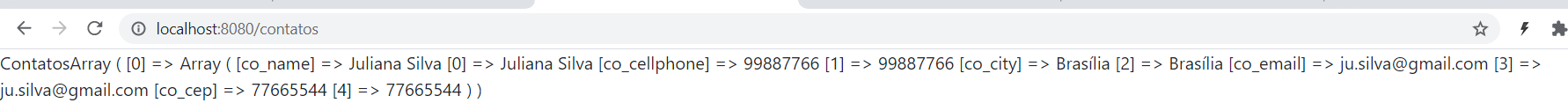
return $this->db->query($sql)->fetchAll();

Teste

No arquivo contatos.phtml

print\_r($this->view->dados);

No browser



Percorrendo os dados com foreach

foreach ($this->view->dados as $key => $contato) {

echo $contato['co\_name']."<br>";

echo $contato['co\_cellphone']."<br>";

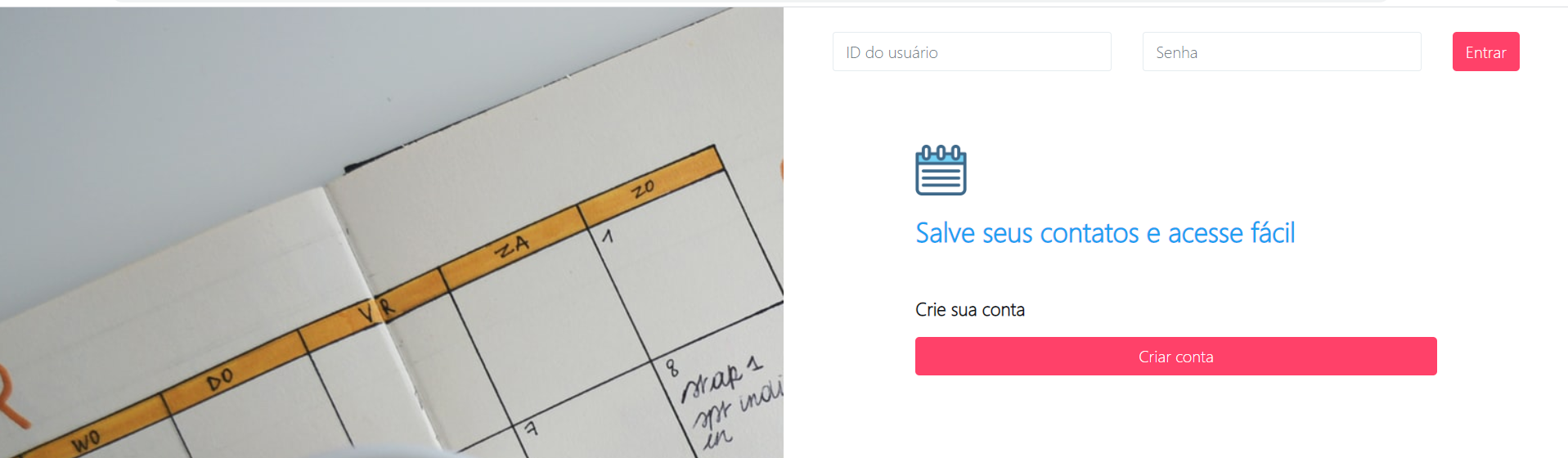
echo $contato['co\_city']."<br>";

echo $contato['co\_email']."<br>";

echo $contato['co\_cep']."<br>";

}

**Formate os arquivos .phtml com html e css como desejar**



Crie todos os arquivos necessários

form\_contatos.phtml

edit\_contatos.phtml

login.phtml

cadastro.phtml

…

**Crie as rotas para acessar os novos arquivos**

$routes['nova\_conta'] = array (

'route' => '/nova\_conta',

'controller' => 'IndexController',

'action' => 'nova\_conta'

);

$routes['cadastrar'] = array (

'route' => '/cadastrar',

'controller' => 'IndexController',

'action' => 'cadastrar'

);

Troque o index por users: rotas, controllers

**Crie os métodos dos cruds**

Salvar

Editar

Listar: getAll

Deletar

Na classe Contato

Métodos mágicos

Salvar

**Autenticação**

public function autenticar() {

$sql = "SELECT us\_id, us\_username, us\_password FROM users WHERE us\_id = :id AND us\_password = :senha";

$stmt = $this->db->prepare($sql);

$stmt->bindValue(':id', $this->\_\_get('id'));

$stmt->bindValue(':senha', $this->\_\_get('senha'));

$stmt->execute();

$user = $stmt->fetch(\PDO::FETCH\_ASSOC);

if($user['us\_id'] != '' && $user['us\_username'] != '') {

$this->\_\_set('id', $user['us\_id']);

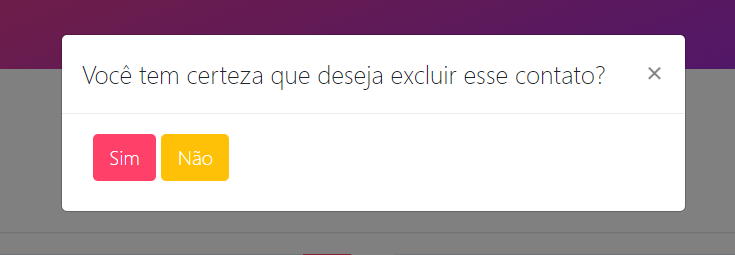
$this->\_\_set('nome', $user['us\_username']);

}

return $this;

}

**Caixa de Confirmação de exclusão**



**Javascript**

Dentro do modal adicione

Estamos criando uma rotina para localizar o botão excluir, capturar o id do registro e se o usuário clicar no botão excluir chamamos a url '/delete\_contato?id='+id

<script>

document.querySelectorAll(".btn-excluir").forEach(btn => {

btn.addEventListener("click", (e) => {

var id = btn.getAttribute("data-id")

console.log(id)

document.getElementById('bt-excluir').onclick = function() {

location.href = '/delete\_contato?id='+id

}

})

})

</script>

**Exibindo o ID do usuário após criar a conta**

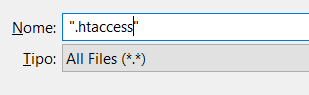
Crie o seguinte código no arquivo de login

...

**Crie o arquivo .htaccess**

Tem que ser entre aspas duplas pois não tem nome

“.htaccess”



**Deploy da aplicação**

...