Wdev - VAGAS

docker-compose.yaml

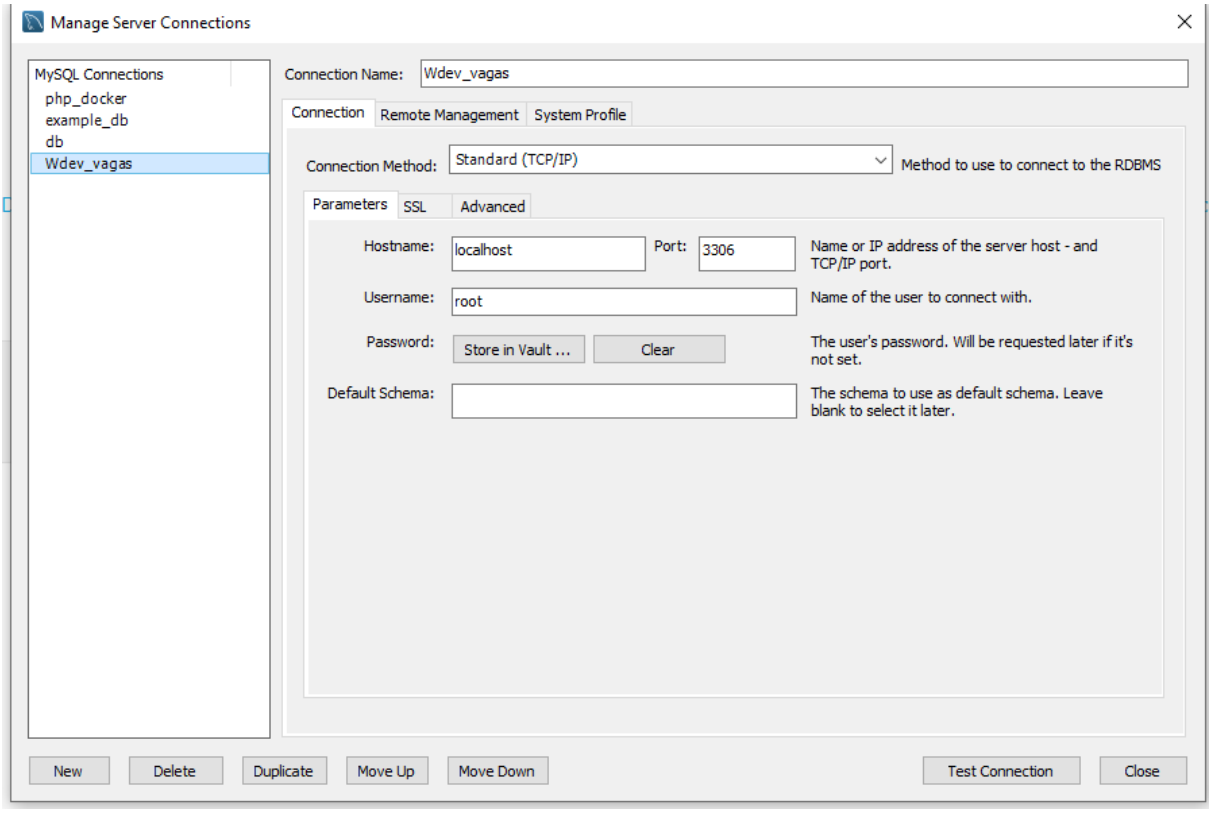
```
version: '3'
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./vagas:/var/www/html
    ports:
    - "80:80"
    depends_on:
      - mariadb
  mariadb:
    image: mariadb:latest
    ports:
    - "3306:3306"
    environment:
      MYSQL_DATABASE: wdev_vagas
      MYSQL_USER: user
      MYSQL_PASSWORD: password
      MYSQL_ROOT_PASSWORD: password
```

Dockerfile

```
FROM php:7.2-apache
RUN docker-php-ext-install pdo pdo_mysql && apk add --no-cach
```

SQL

```
environment:
       MYSQL_DATABASE: wdev_vagas
       MYSQL_USER: user
       MYSQL_ROOT_PASSWORD: password
       MYSQL_PASSWORD: password
```



MYSQL conexão.

```
USE wdev_vagas;
```

```
CREATE TABLE vagas (
id INT AUTO_INCREMENT PRIMARY KEY,
titulo VARCHAR(255),
descricao TEXT,
ativo ENUM('s','n'),
data_post TIMESTAMP
)
```

1-opção :COMPOSER.JSON

Criar arquivo composer.json na pasta do volume compartilhável do docker

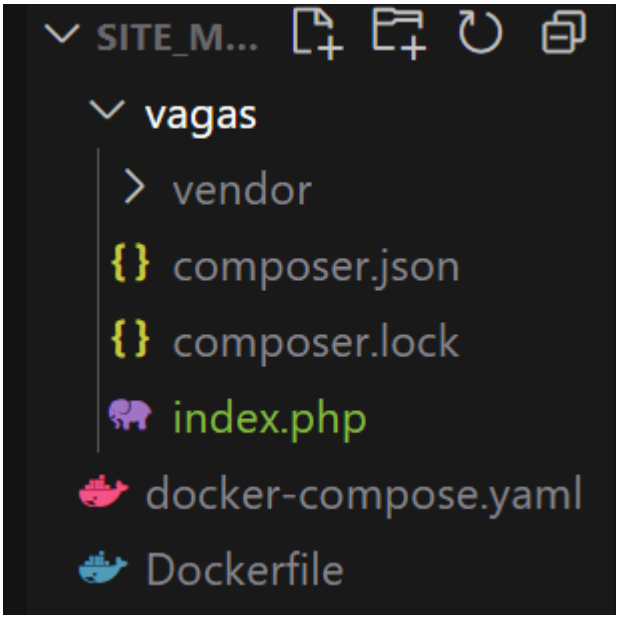
```
{
    "name": "wdev/vagas",
    "autoload": {
        "psr-4": {
            "App\\":"app/"
        }
    }
}
```

Via terminal, entrar na pasta onde se encontra o composer e dar o comando composer up. Irá instalar as dependências do Autoload

```
composer up
```

Criar o arquivo index.php e colocar na raiz do volume docker. Adicionar um Hello World neste arquivo e abrir o localhost no navegador para verificar a funcionalidade.

```
<?php
echo "Hello";
```



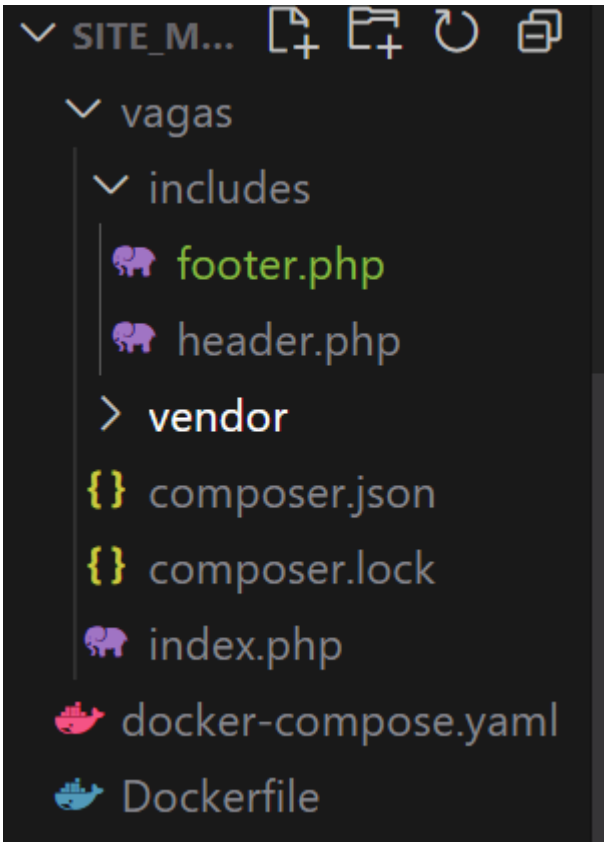
2-opção: Composer init

Ir no terminal, na pasta compartilhada do volume. Dar o comando composer init. Da enter várias vezes e pronto!

```
{
    "name": "guilh/projeto",
    "autoload": {
        "psr-4": {
            "Guilh\\Projeto\\": "src/"
        }
    },
    "authors": [
        {
            "name": "guilherme",
            "email": "guilherme.oliveira.antonio@gmail.com"
        }
    ],
    "require": {}
}
```

INCLUDES

A pasta includes serve para organizar o código de maneira para se aproveitar o que já foi feito. Usado como cabeçalhos e rodapés de página, código de conexão a banco de dados e validação.



Criar os arquivos footer.php e header.php

Entrar na pasta header.php e adicionar o seguinte código bootstrap:

fonte https://getbootstrap.com/docs/5.3/getting-started/introduction/

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, init
ial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.
0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="s
ha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO
```

```
02iuK6FUUVM" crossorigin="anonymous">
  </head>
  <body>
    <h1>Hello, world!</h1>
</div>
```

Na pasta footer adicionar o seguinte código

```
    <!-- classe container-->
</div>
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5
  </body>
</html>
```

No index.php

Adicionar o seguinte código:

```
require __DIR__.'/vendor/autoload.php';

include __DIR__.'/includes/header.php';
include __DIR__.'/includes/listagem.php';
include __DIR__.'/includes/footer.php';
```

Listagem.php

A listagem é responsável por listar os campos do banco de dados, fica entre a tag <main>. A section é um pequeno grupo de conteudo. No primeiro grupo tem o botão com link que direciona para a página cadastrar.php que fica na pasta includes. no segundo grupo section tem uma tabela com os mesmos titulos do banco de dados.

```
<main>
    <section>
        <a href="/includes/cadastrar.php">
            <button class="btn btn-success">Nova Vaga</button
```

```
        </a>
    </section>

    <section>
        <table class ="table bg-light mt-3">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Título</th>
                    <th>Descrição</th>
                    <th>Status</th>
                    <th>Data</th>
                    <th>Ações</th>
                </tr>
            </thead>
            <tbody>
            </tbody>
        </table>
    </section>
</main>
```

table → Uma informação composta por mais que uma dimensão em forma de tabela.

thead→ conjunto de linhas que forma o titulo de cada coluna

tr→ Representa um conjunto de linhas da célula

th→elemento que representa o título da célula

Após isso, criar o arquivo cadastrar.php na pasta includes

Cadastrar.php

Adicionar os includes abaixo:

```
<?php
include __DIR__.'/../includes/header.php';
```

```
include __DIR__.'/../includes/formulario.php';
include __DIR__.'/../includes/footer.php';
```

Criar o formulário.php e incluir na pasta includes

Formulário.php

Criar uma tag main adicionar uma section para o botão voltar e um formulário POST para gravar as informações pro banco.

```
<main>
        <section>
        <a href = "/index.php">
            <button class="btn btn-success">Voltar</button>
        </a>
        </section>
        <h2 class="mt-3">Cadastrar vaga</h2>
    <form method="POST">
        <div class="form-group">
            <label>Título</label>
            <input type="text" class = "form-control" name="t
        </div>
        </form>
</main>
```

```
 <div class = "form-group">
            <label>Descrição</label>
            <textarea class="form-control" name = "descricao"
 </div>
```

```
        <div class = "form-group">
        <label>Status</label>
        <div>
            <div class = "form-check form-check-inline">
                <label class = "form-control">
                    <input type="radio" name="ativo" value="s
```

```
                </label>
            </div>
            <div class = "form-check form-check-inline">
                <label class = "form-control">
                    <input type="radio" name="ativo" value="n
                </label>
            </div>
            </div>
      </div>
```

O próximo bloco é o botão submit

```
                <div class = "form-group">
            <button type="submit" class = "btn btn-success">E
        </div>
```

No final, todo o código ficará assim:

```
<main>
    <section>
        <a href = "/index.php">
            <button class="btn btn-success">Voltar</button>
        </a>
    </section>

    <h2 class="mt-3">Cadastrar vaga</h2>
    <form method="POST">
        <div class="form-group">
            <label>Título</label>
            <input type="text" class = "form-control" name="t
        </div>

        <div class = "form-group">
            <label>Descrição</label>
            <textarea class="form-control" name = "descricao"
        </div>

        <div class = "form-group">
```

```
            <label>Status</label>

            <div>
                <div class = "form-check form-check-inline">
                    <label class = "form-control">
                        <input type="radio" name="ativo" valu
                    </label>
                </div>
                <div class = "form-check form-check-inline">
                    <label class = "form-control">
                        <input type="radio" name="ativo" valu
                    </label>
                </div>
            </div>
        </div>
        <div class = "form-group">
            <button type="submit" class = "btn btn-success">E
        </div>

    </form>
</main>
```

Cadastrar.php

Completar o seguinte código no cadastrar.php

```
<?php
require __DIR__.'/../vendor/autoload.php';
print_r($_POST);
include __DIR__.'/../includes/header.php';
include __DIR__.'/../includes/formulario.php';
include __DIR__.'/../includes/footer.php';
```

As dependências são pacotes de código que são usados por um projeto PHP para fornecer recursos adicionais. O arquivo `autoload.php` é gerado pelo utilitário `composer`.

O print_r($_POST) é para efeito de teste, ao dar um submit no formulário, tem que aparecer as seguintes informações

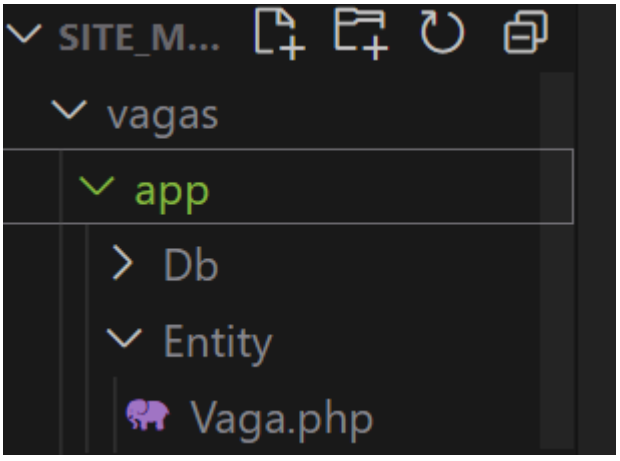
```
Array ( [titulo] => Exemplo_programador [descricao] => Vaga e
```

apagar o print_r e adicionar o seguinte código

```
if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo

}
```

Criar a classe de entidade de vaga. Vamos abstrair a instância de banco de dados para um objeto.

dentro da pasta vagas, criar uma pasta app, criar uma pasta Entity, criar uma classe php chamado Vaga.php



Vaga.php

Adicionar o namespace

```
App\Entity;
```

```
class Vaga{
```

```
    /**
    * Identificador único da vaga
    * @var integer
    */
    public $id;

    /**
    * Titulo da vaga
    * @var string
    */
    public $titulo;

    /**
    * Descrição da vaga (pode conter html)
    * @var string
    */
    public $descricao;

    /**
    * Define se a vaga ativa
    * @var string(s/n)
    */
    public $ativo;

    /**
    * Data de publicação da vaga
    * @var string
    */
    public $data;
}
```

Por enquanto ficou assim

```
<?php
//esta classe de entidade de vaga abstrai a instancia do banc
namespace App\Entity; // la no composer.json tem o -> App\\ q

class Vaga{
```

```
    /**
    * Identificador único da vaga
    * @var integer
    */
    public $id;

    /**
    * Titulo da vaga
    * @var string
    */
    public $titulo;

    /**
    * Descrição da vaga (pode conter html)
    * @var string
    */
    public $descricao;

    /**
    * Define se a vaga ativa
    * @var string(s/n)
    */
    public $ativo;

    /**
    * Data de publicação da vaga
    * @var string
    */
    public $data;
}
```

Agora que os objetos foram criados, voltar no arquivo cadastrar.php para prosseguir.

cadastrar.php

Adicionar o seguinte:

```
use \App\Entity\Vaga;

if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo
    $obVaga = new Vaga;
    $obVaga->titulo    = $_POST['titulo'];
    $obVaga->descricao = $_POST['descricao'];
    $obVaga->ativo     = $_POST['ativo'];
        $obVaga->cadastrar();
    print_r($obVaga);

}
```

O use\App\Entity\vaga → importa a classe Vaga do espaço de nomes App\entity para o escopo atual. Isso permite a utilização da classe Vaga sem prefixá-la

Ao dar um submit no localhost, tem que aparecer isso:

App\Entity\Vaga Object ([id] => [titulo] => Exemplo_programador [descricao] => Example_abc [ativo] => s [data] =>)

O ID não está definido, porque não está conectado com o SQL no modo auto_increment. A data segue o mesmo principio

Ao final, ele irá para a função cadastrar() da classe Vaga.php

Vaga.php

Adicionar a função cadastrar()

```
        /**
    * Método responsável por cadastrar uma nova vaga no banco
    * @return boolean
    */
    public function cadastrar(){
        // Definir a data

        //Inserir a vaga no banco
```

```
        //Atribuir o ID da vaga na instancia

        //Retornar sucesso
    }
```

Adicionar no campo definir a data:

```
$this->data = date('Y-m-d H:i:s');
```

Agora para inserir no banco de dados, será necessário uma classe externa para acessá-lo usando o PDO.

Criar um arquivo chamado Database.php dentro do diretório app/Db

Database.php

```
<?php
namespace app\Db;
USE \PDO;
class Database{
    /**
    * HOST de conexão com o banco de dados
    * @var string
    */
    const HOST = 'mariadb';

    /**
    * Nome do banco de dados
    * @var string
    */
    const NAME = 'wdev_vagas';


    /**
    * Usuario do banco
    * @var string
    */
```

```
    const USER = 'root';

    /**
    * senha acesso banco de dados
    * @var string
    */
    const PASS = 'password';

    /**
    * nome da tabela a ser manipulada
    * @var string
    */
    private $table; //definir qual tabela está trabalhando

    /**
    * Instancia de conexão com o banco de dados
    * @var pdo
    */
    private $connection;
}
```

Esta classe fará uma ponte entre o banco de dados e o sistema usando PDO. Funcionará como carry builder, não executará SQL mas com as mesmas funções. Basicamente a classe ira abstrair a escrita do SQL

O namespace é usado para agrupar as classes e funções relacionadas ao banco de dados em um único espaço. Por exemplo, se você tiver uma classe chamada `User` no namespace `app\Db`, poderá acessá-la usando o nome `app\Db\User`. Isso evita que você confunda sua classe `User` com outras classes `User` que podem existir em outros namespaces.

São constantes porque não irão alterar ao longo do tempo.

PDO_DATABASE docker-compose.yaml

const HOST = 'mariadb' →>>>>>>>>>>> mariadb:

const NAME = 'wdev_vagas';→>>>>>>>>>MYSQL_DATABASE: wdev_vagas

const USER = 'root'; →>>>>>>>>>>>>> o root é nome dedicado no mysql

const PASS = 'password';→>>>>>>>>>>MYSQL_ROOT_PASSWORD: password

Segundo os comentário da documentação PHP, NÃO DEVEE USAR ROOT NAS APLICAÇÕES, ao menos que a aplicação foi feita para a manutenção do servidor. soma-se a isso, guardando diretamente o USERNAME/PASSWORD dentro da classe não é uma boa ideia de código de aplicação. Seria necessário editar o código atual para alterar as configurações, o que é horrivel!!

private $connection; → esta instância de conexão usando PDO se fez necessário a dependencia da classe no USE \PDO;

```
        /**
    * Defini a tabela e instancia e conexão
    * @param string $table
    */
    public function __construct($table = null)
    {
        $this->table=$table;
        $this->setConnection();
    }
```

```
private function setConnection(){

        try{
            $this->connection = new PDO('mysql:host='.self::H
            $this->connection->setAttribute(PDO::ATTR_ERRMODE

        }catch (PDOException $e){
            die('ERROR: '.$e->getMessage()); // nunca expor a
        }
    }
```

A manipulação segura é usado o try e catch para tratar os erros que o sistema gera

nunca expor as mensagem de erro para o usuario final → o ideal é exibir uma mensagem mais amigável ao usuario e gravar a informação no log para uso interno

$this→connection = new PDO → o construtor do PDO recebe alguns parâmetros:

https://www.php.net/manual/en/class.pdo.php

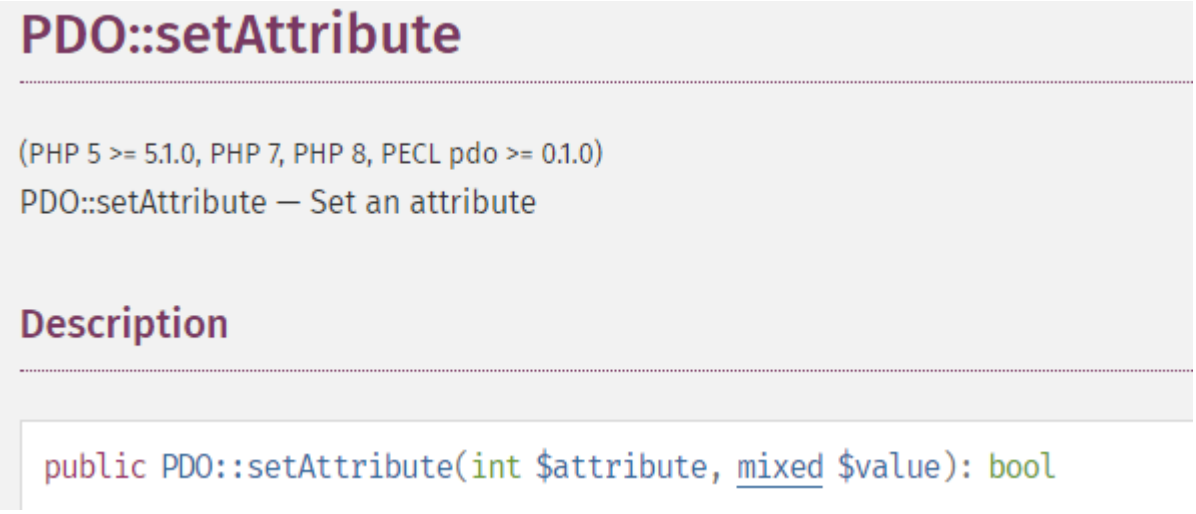
string $dsn → string de conexão, tipo de banco de dados, nome do banco de dados tudo na MESMA STRING

new PDO('mysql:host='.self::HOST.';dbname='.self::NAME → todo esse conjunto faz parte do DNS

self::USER → faz parte do string $username

self::PASS → faz parte do string $password

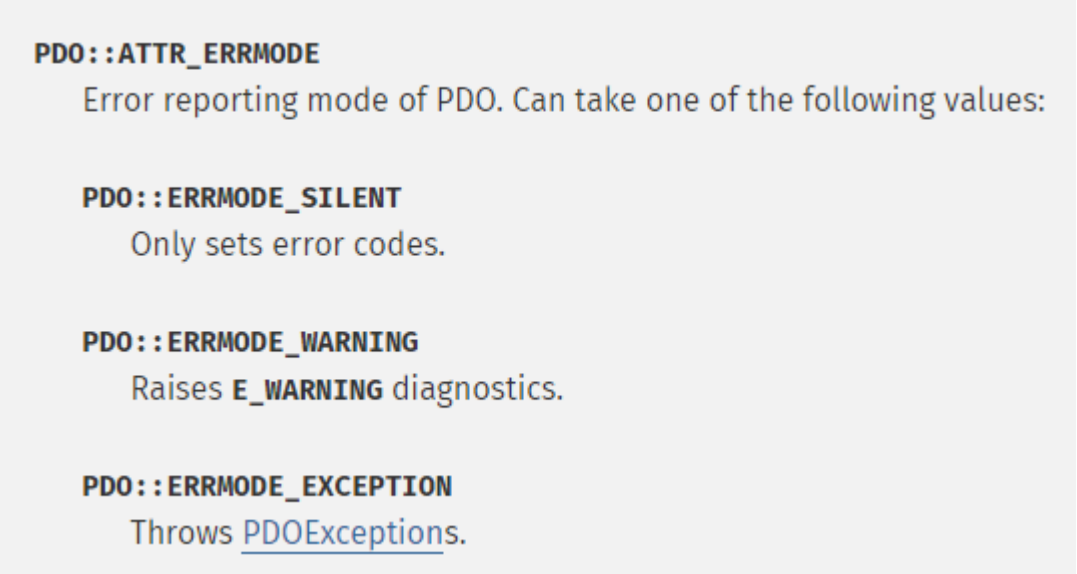
Na documentação do setAttribute:



https://www.php.net/manual/en/pdo.setattribute.php

Ele pede somente 2 parâmetros. o primeiro é o atributo que quer alterar, o segundo é o valor que vai receber. do tipo ERRMODE existem as opções abaixo conforme o print da documentação.

setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);



https://www.php.net/manual/en/pdo.setattribute.php

Adicionar a dependencia PDOException na classe database.php. O código ficou assim por enquanto:

```
<?php
namespace app\Db;
use \PDO;
use \PDOException;
class Database{
    /**
    * HOST de conexão com o banco de dados
    * @var string
    */
    const HOST = 'mariadb';

    /**
    * Nome do banco de dados
    * @var string
    */
    const NAME = 'wdev_vagas';
```

```
/**
* Usuario do banco
* @var string
*/
const USER = 'root';

/**
* senha acesso banco de dados
* @var string
*/
const PASS = 'password';

/**
* nome da tabela a ser manipulada
* @var string
*/
private $table; //definir qual tabela está trabalhando

/**
* Instancia de conexão com o banco de dados
* @var pdo
*/
private $connection;

    /**
* Defini a tabela e instancia e conexão
* @param string $table
*/
public function __construct($table = null)
{
    $this->table=$table;
    $this->setConnection();
}
/**
* Método responsável por criar uma conexão com o banco de
*/
```

```
    private function setConnection(){

        try{
            $this->connection = new PDO('mysql:host='.self::H
            $this->connection->setAttribute(PDO::ATTR_ERRMODE

        }catch (PDOException $e){
            die('ERROR: '.$e->getMessage()); // nunca expor a
        }
    }
}
```

cadastrar.php

Na parte //inserir vaga no banco

```
$obDatabase = new Database('vagas');
echo "</br>";print_r($obDatabase);echo "</br>";
```

Adicionar a dependência do Database

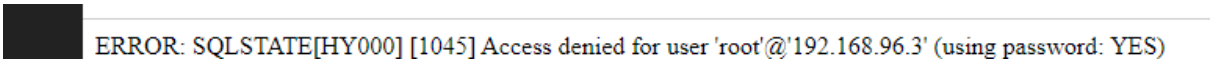
```
use \App\Db\Database;
```

app\Db\Database Object ([table:app\Db\Database:private] => vagas [connection:app\Db\Database:private] => PDO Object ( ))
App\Entity\Vaga Object ([id] => [titulo] => [descricao] => [ativo] => s [data] => 2023-07-17 13:59:48)

Dando o submit aparecerá isso. Não ocorrendo exception, tudo deu certo

Para efeito de testes, pode alterar o *const* PASS = 'password' para *const* PASS = 'teste'

irá aparecer o seguinte erro:



Para cada mudança de nome da string de conexão, irá aparecer um erro diferente..

Obs: voltar a *const* PASS = 'password'

Remover o echo print_r

Vaga.php

Adicionar o insert.

```
public function cadastrar(){
        // Definir a data
        $this->data = date('Y-m-d H:i:s');
        //Inserir a vaga no banco
        $obDatabase = new Database('vagas');
        $obDatabase->insert([
                                    'titulo'    => $this->tit
                                    'descricao' => $this->des
                                    'ativo'     => $this->ati
                                    'data_post'      => $this
        ]);//carry builder vai executar na tabela vagas
```

Database.php

A query comum de inserir dados é:

```
$query = ' INSERT INTO vagas (titulo,descricao,ativo,data) VA
')';
```

```
$query = ' INSERT INTO vagas (titulo,descricao,ativo,data) VA
```

```
$query = ' INSERT INTO'.$this->table.'(titulo,descricao,ativo
```

Temos o código atual:

```
public function insert($values){
        //MONTA QUERY
        $query = 'INSERT INTO '.$this->table.'(titulo,descric
```

```
        echo $query;
        exit();
    }
```

no navegador:

INSERT INTO vagas(titulo,descricao,ativo,data) VALUE(?,?,?,?)

Ou seja, o $this→table virou vagas. porém o restante está fixo ainda

Adicionando o código abaixo

```
public function insert($values){
  //DADOS da query
    $fields = array_keys($values);//criar parâmetros dinâmico
  echo "<\br>"; print_r($values); echo"<\br>";
```

Array ([titulo] => [descricao] => [ativo] => s [data_post] => 2023-07-18 18:07:06)

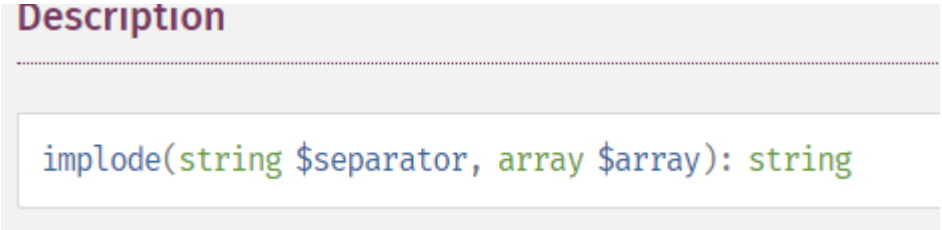
```
$fields = array_keys($values);//criar parâmetros dinâmicos
        echo "</br>"; print_r($fields); echo"</br>";
```

Array ([0] => titulo [1] => descricao [2] => ativo [3] => data_post)

Logo, é necessário concatenar a $query utilizando o campo $fields usando o comando implode

```
//monta a query
        $query = 'INSERT INTO '.$this->table.'('.implode(',',
        echo "</br>"; print_r($query); echo"</br>";
```

A função implode segundo a documentação une o elemento separador com o array



',' → é o string $separator

$fields → é o array

A $query ficou igual o planejado anteriormente

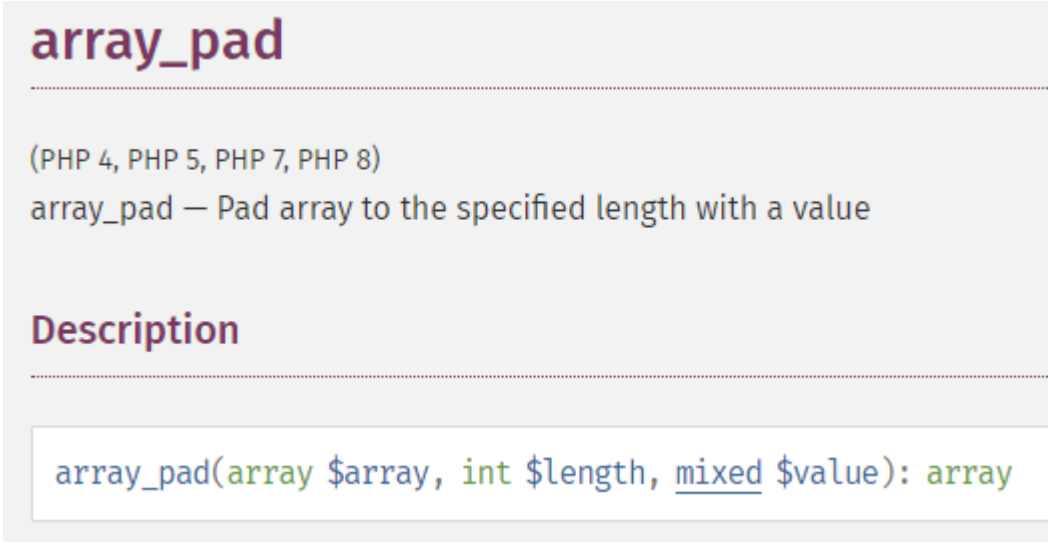
INSERT INTO vagas(titulo,descricao,ativo,data_post) VALUE(?,?,?,?)

Mas o campo VALUE ainda está fixo, caso ocorra a remoção de um dos itens do $fields, o value tem que adequar.

INSERT INTO vagas(titulo,descricao,ativo) VALUE(?,?,?,?)

Tem 3 campo no vagas, e 4 no values.

```
$binds = array_pad([],count($fields),"?");
        echo "</br>"; print_r($binds); echo"</br>";
```



array_pad

(PHP 4, PHP 5, PHP 7, PHP 8)

array_pad — Pad array to the specified length with a value

Description

```
array_pad(array $array, int $length, mixed $value): array
```

array_pad([] →copia de um novo array vazio
,count($fields)→ conta a quantidade de array variavel em int
,"?"); → substitui pelo simbolo que esta entre aspas

INSERT INTO vagas(titulo,descricao,ativo) VALUE(?,?,?,?)

```
$binds = array_pad(["a",2],10,"?");
        echo "</br>"; print_r($binds); echo"</br>";
```

Array ([0] => a [1] => 2 [2] => ? [3] => ? [4] => ? [5] => ? [6] => ? [7] => ? [8] => ? [9] => ?)

Irá aparecer o seguinte, primeira posição é o a, segunda é o 2, o restante preenche com ? até a quantidade especificada no lenght

Logo, o código fica assim:

```
$binds = array_pad([],count($fields),"?");
//monta a query
$query = 'INSERT INTO '.$this->table.'('.implode(",",$fields)
echo "</br>"; print_r($query); echo"</br>";
```

Todas as queries que montar, seguirá o mesmo padrão, logo, da para padronizar criando um método unico de executar

```
/**
    * Método responsável por executar queries dentro do banco
    * @param string $query
    * @param array $params
    * @return PDOStatement // o PDO irá instanciar diretament
    */
    public function execute($query,$params=[]){
        try{
            $statement = $this->connection->prepare($query);
            $statement->execute($params);
            return $statement; // para insert, delete e updat
        }catch (PDOException $e){
            die('ERROR: '.$e->getMessage()); // nunca expor a
        }
    }
```

public function execute($query,$params=[]) → os parâmetros serão substituídos pelos valores dos binds (public function insert($values)

try e catch (PDOException $e) → Caso de erro em alguma sintaxe, uma exception será lançada

$statement = $this→connection→

prepare($query) → irá verificar os binds que precisa substituir

$statement→execute($params) →

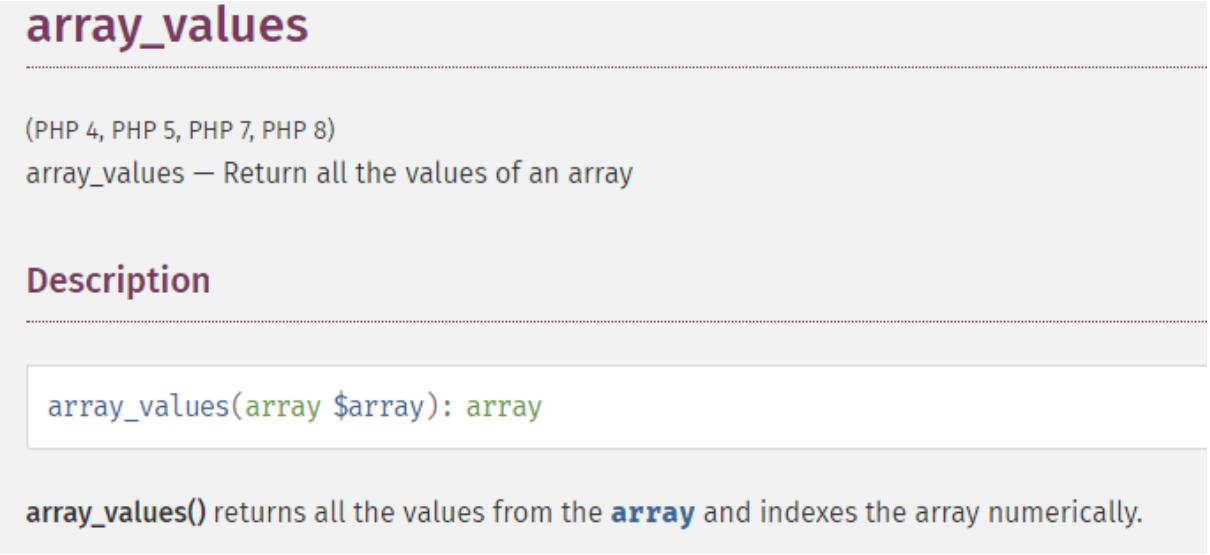
return $statement; → necessário pra queries de consulta

Adicionar o código no método insert

```
            //executa o insert
        $this->execute($query, array_values($values));

        //retorna o ID inserido
        return $this->connection->lastInsertId();
```

$this→execute( )→ confome método criado, ele pede a query e os parâmetros

$query, array_values($values)); → consta a query e o array_values



array_values

(PHP 4, PHP 5, PHP 7, PHP 8)
array_values — Return all the values of an array

Description

```
array_values(array $array): array
```

array_values() returns all the values from the **array** and indexes the array numerically.

irá pegar somente os valores do array

o método insert fica assim:

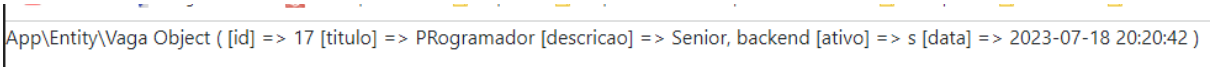
```
public function insert($values){
        //DADOS da query
        $fields = array_keys($values);//criar parâmetros dinâ

        // o metodo array_pad faz com que o array tenha x pos
        $binds = array_pad([],count($fields),"?");

        //monta a query
        $query = 'INSERT INTO '.$this->table.'('.implode(",",

        //executa o insert
        $this->execute($query, array_values($values));

        //retorna o ID inserido
        return $this->connection->lastInsertId();

    }
```

Vaga.php

no arquivo Vaga.php:

```
public function cadastrar(){
        // Definir a data
        $this->data = date('Y-m-d H:i:s');
        //Inserir a vaga no banco
        $obDatabase = new Database('vagas');
        $this->id = $obDatabase->insert([
                                    'titulo'    => $this->tit
                                    'descricao' => $this->des
                                    'ativo'     => $this->ati
                                    'data_post'      => $this
        ]);//carry builder vai executar na tabela vagas
        //retornar sucesso
                print_r($this);
        return true;
```

alterado o $this→id = $obDatabase, será atribuído o ID na vaga e o resultado do insert.



utilizando o print_r, irá aparecer que a vaga está sendo inserida no banco de dados com o ID autoincremental

ao final, apagar o print_r($this);

cadastrar.php

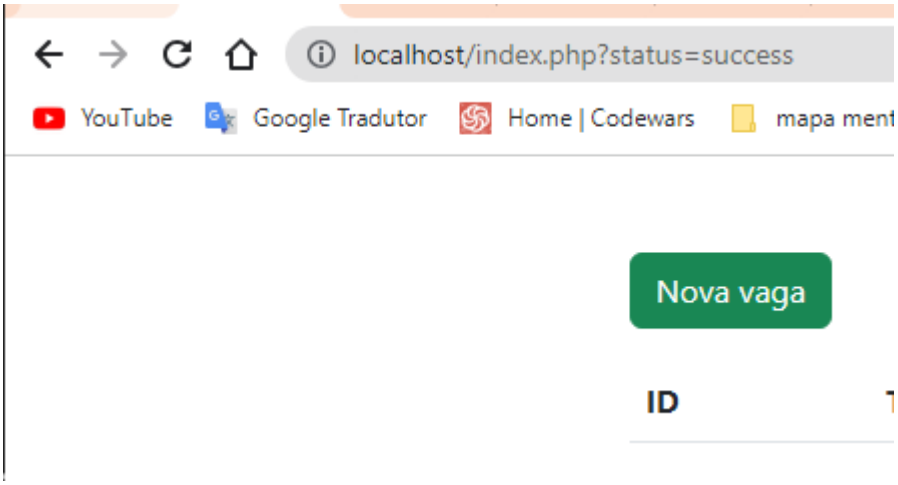
Adicionar o header success:

```
if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo
    $obVaga = new Vaga;
    $obVaga->titulo    = $_POST['titulo'];
    $obVaga->descricao = $_POST['descricao'];
    $obVaga->ativo     = $_POST['ativo'];
    $obVaga->cadastrar();
```

```
    header('location: ../index.php?status=success');
        exit();
}
```

header('location: index.php? status=success'); → altera o status da barra do navegador URL.

exit() → para o código aqui, nao deixando executar os includes abaixo.



listagem.php

Aqui ficará as consultas do banco

```
<main>
    <section>
        <a href="/includes/cadastrar.php">
            <button class = "btn btn-success mt-5">Nova vaga<
        </a>
    </section>

    <section>
        <table class = "table bg-light mt-3">
            <thead>
                <tr>
```

```
                    <th>ID</th>
                    <th>Título</th>
                    <th>Descrição</th>
                    <th>Status</th>
                    <th>Data</th>
                    <th>Ações</th>
                </tr>
            </thead>
            <tbody>
            </tbody>
        </table>
    </section>

</main>
```

Index.php

Adicionar

```
require __DIR__.'/vendor/autoload.php';

use \App\Entity\Vaga;

$vagas = Vaga::getVagas();
print_r($vagas);
include __DIR__.'/includes/header.php';
include __DIR__.'/includes/listagem.php';
include __DIR__.'/includes/footer.php';
```

$vagas = Vaga::getVagas(); → listagem de vagas

getVagas(); → será criado

Vaga.php

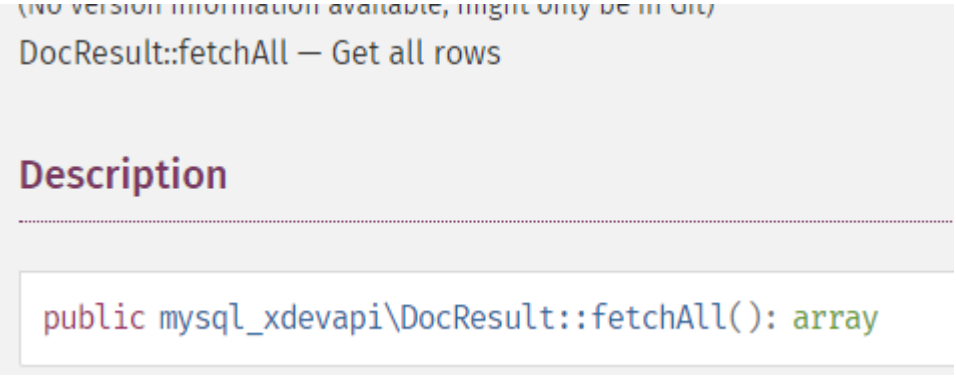
```
/**
     * Método responsávl por obter as vagas do banco de dados
     * @param string $where
     * @param string $order
     * @param string $limit
     * @return array
     */
    // método estático porque irá retornar várias instâncias
    //eles irão criar as clasulás para que a pesquisa seja de
    public static function getVagas($where = null, $order = n
        return (new Database('vagas'))->select($where, $order
                                       -> fetchAll(PDO::FETCH_
    }
```

public static function getVagas()→ é estático porque irá retornar um array com várias instâncias de vagas

$where = null, $order = null, $limit = null → clásulas do sistema SQL

return (new Database('vagas'))→select → como é só uma consulta, só precisa do retorno

fetchAll → pega tudo, todo retorno será um array



FETCH_CLASS → constante que diz o tipo de array retornado no caso de classe

self::class → instancia da própria classe

Adicionar a dependência \PDO, pois estamos usando.

```
use \PDO
```

database.php

```
/**
     * Método responsável por executar uma consulta no banco
     * @param string $where
     * @param string $order
     * @param string $limit
     * @param string $fields
     * @return PDOStatement //precisa do statment retornado, 
     * Nem sempre irá passar esses métodos, por isso que é nu
     */
    public function select( $where = null, $order = null, $li
        //Dados da query
        //condição ternária, caso conteudo na $where ela será
        $where = strlen($where) ? 'WHERE '.$where : '';
        $order = strlen($order) ? 'ORDER BY '.$order : '';
        $limit = strlen($limit) ? 'LIMIT '.$limit : '';

        //monta query
        $query = 'SELECT '.$fields.' FROM '.$this->table.' '.

        //executa a query
        return $this->execute($query);
    }
```

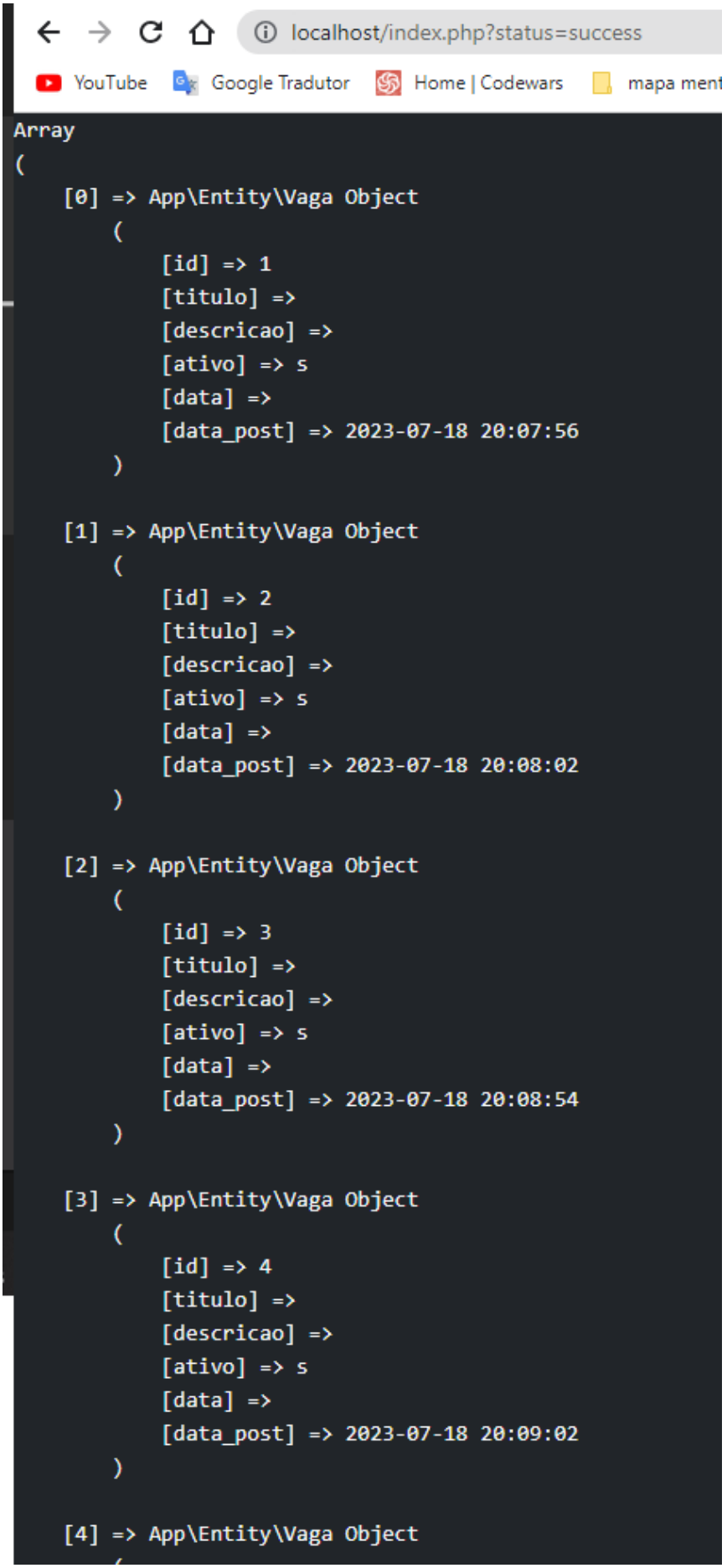
$where = strlen($where) ? 'WHERE '.$where : '';
$where = strlen($order) ? 'ORDER BY '.$where : '';
$where = strlen($limit) ? 'LIMIT '.$where : ''; → essas condições podem ser opcionais, caso tenha terá o que está entre string, caso não, será espaço nulo

$fields = '*' → esse asterisco quer dizer tudo

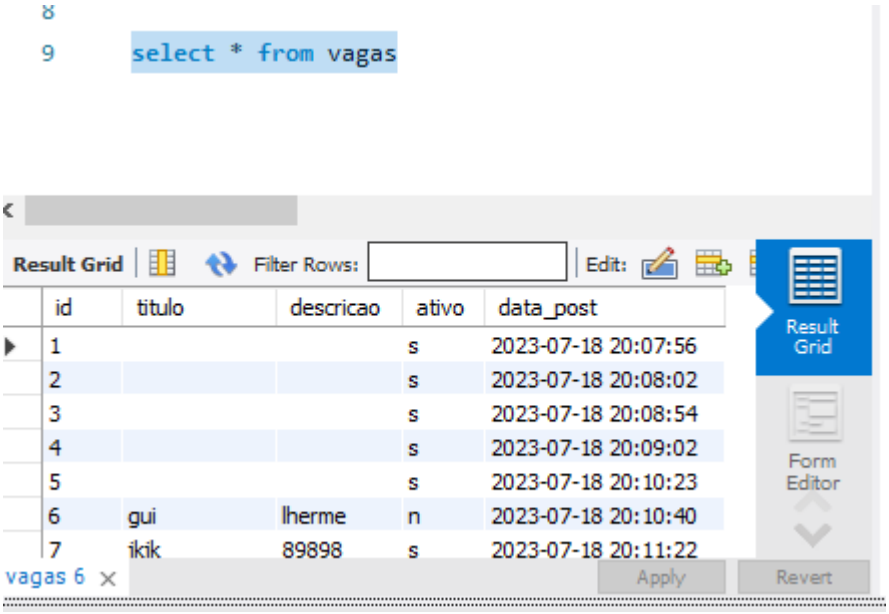
Index.php

Adicionar a linha para debug

```
"<pre>";print_r($vagas);echo "<\pre>";
```


localhost/index.php?status=success
YouTube
Google Tradutor
Home | Codewars
mapa ment
Array
(
[0] => App\Entity\Vaga Object
(
[id] => 1
[titulo] =>
[descricao] =>
[ativo] => s
[data] =>
[data_post] => 2023-07-18 20:07:56
)
[1] => App\Entity\Vaga Object
(
[id] => 2
[titulo] =>
[descricao] =>
[ativo] => s
[data] =>
[data_post] => 2023-07-18 20:08:02
)
[2] => App\Entity\Vaga Object
(
[id] => 3
[titulo] =>
[descricao] =>
[ativo] => s
[data] =>
[data_post] => 2023-07-18 20:08:54
)
[3] => App\Entity\Vaga Object
(
[id] => 4
[titulo] =>
[descricao] =>
[ativo] => s
[data] =>
[data_post] => 2023-07-18 20:09:02
)
[4] => App\Entity\Vaga Object

Irá aparecer a listagem da tabela do sql



no workbench também

listagem.php

Adicionar a tag php e o seguinte código:

```
<?php
    $resultados ='';
    foreach($vagas as $vaga){
        $resultados .='<tr>
                            <td>'.$vaga->id.'</td>
                            <td>'.$vaga->titulo.'</td>
                            <td>'.$vaga->descricao.'</td>
                            <td>'.$vaga->ativo.'</td>
                            <td>'.$vaga->data.'</td>
                            <td></td>
                        </tr>';
    }
?>
<main>
    <section>
        <a href="/includes/cadastrar.php">
            <button class = "btn btn-success mt-5">Nova vaga<
```

```
        </a>
    </section>

    <section>
        <table class = "table bg-light mt-3">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Título</th>
                    <th>Descrição</th>
                    <th>Status</th>
                    <th>Data</th>
                    <th>Ações</th>
                </tr>
            </thead>
            <tbody>
                <?=$resultados?>
            </tbody>
        </table>
    </section>

</main>
```

$resultados =''; → instancia como vazio, pois irá substituir no <tbody> do html

foreach($vagas as $vaga) → da lista vagas, percorrer uma a uma por vaga

$resultados .='<tr> → o resultado será a impressão de uma linha

'<tr>
<td>'.$vaga→id.'</td>
<td>'.$vaga→titulo.'</td>
<td>'.$vaga→descricao.'</td>
<td>'.$vaga→ativo.'</td>
<td>'.$vaga→data.'</td>
<td></td>
</tr>'; → como irá percorrer todas as linhas, isso irá buscar cada parâmetro da linha especifica e jogar na variavel resultado

Substituir a linha de baixo para fazer um ternário

<td>'.$vaga→ativo.'</td>

substituir para

<td>'.($vaga→ativo== 's' ? 'Ativo' : 'Inativo').’

Substituir a linha de baixo para formatar a hora

<td>'.$vaga→data.'</td>

<td>'.date('d/m/Y H:i:s',strtotime($vaga→data_post)).'</td>

Adicionar os dois botões (editar e excluir):

```
<?php
    $resultados ='';
    foreach($vagas as $vaga){
      $resultados .='<tr>
                        <td>'.$vaga->id.'</td>
                        <td>'.$vaga->titulo.'</td>
                        <td>'.$vaga->descricao.'</td>
                        <td>'.($vaga->ativo == 's' ? 'Ativo'
                        <td>'.date('d/m/Y H:i:s',strtotime($v
                        <td>
                        <a href="editar.php?id='.$vaga->id.'"
                          <button type ="button" class="btn b
                        </a>
                        <a href="excluir.php?id='.$vaga->id.'
                          <button type ="button" class="btn b
                        </a>
                        </td>
                    </tr>';
    }
?>
```

editar.php

Duplicar o arquivo cadastrar e renomear com editar.php, jogar no mesmo nível do arquivo index.php

validação do ID

```
// obrigado usuario a passar ID. Validação do ID
if(!isset($_GET['id']) or !is_numeric($_GET['id'])){
    header('location: index.php?status=error');
    exit();
}
// cria instancia da vaga
$obVaga = Vaga::getVaga($_GET['id']);
print_r($obVaga);
```

getVaga() → Será criado

print_r($obVaga); → debug

Vaga.php

```
/**
    * Método responsável por buscar uma vaga com base em seu
    * @param integer $id
    * @param Vaga
    */
    public static function getVaga($id){
        return (new Database('vagas'))->select('id = '.$id);

    }
```

new Database('vagas') → instância o database da tabela vagas

select('id = '.$id) → passou como parâmetro o $id para o método select

```
PDOStatement Object
(
    [queryString] => SELECT * FROM vagas WHERE id = 2
)
```

Olhando no debug

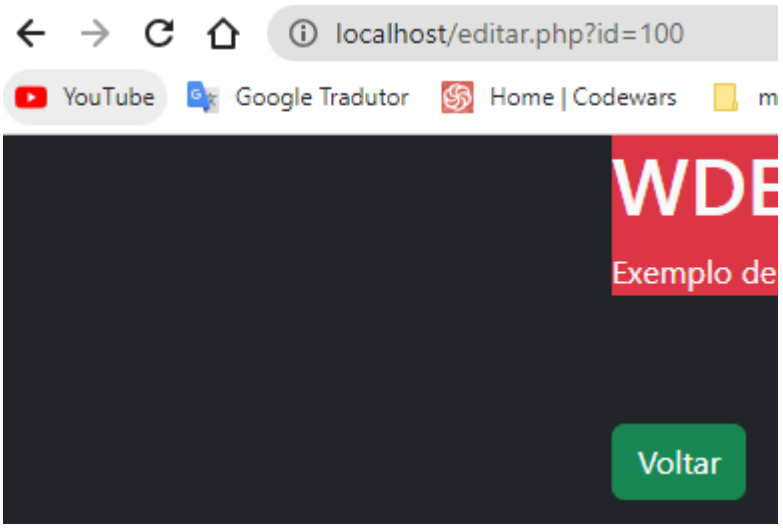
```
/**
    * Método responsável por buscar uma vaga com base em seu
    * @param integer $id
    * @param Vaga
    */
    public static function getVaga($id){
        return (new Database('vagas'))->select('id = '.$id)->

    }
```

fetchObject(self::class) → fetch unitário de uma posição e retorna como objeto ao inves de array

```
App\Entity\Vaga Object
(
    [id] => 2
    [titulo] => ggg
    [descricao] => gggg
    [ativo] => n
    [data] =>
    [data_post] => 2023-07-19 19:31:04
)
```

index.php

Ao tentar alterar diretamente pela URL um ID que não exista, é necessário aparecer um erro neste header




```
//Validação da vaga
if(!$obVaga instanceof Vaga){
   header('location: index.php?status=error');
}
```

Comentar o *//$obVaga→cadastrar(); e apagar o* $obVaga = new Vaga;, pois ja tem uma instância na parte superior $obVaga = Vaga::getVaga($_GET['id']);

```
if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo
    // remover -> $obVaga = new Vaga;
    $obVaga->titulo    = $_POST['titulo'];
    $obVaga->descricao = $_POST['descricao'];
    $obVaga->ativo     = $_POST['ativo'];

    //$obVaga->cadastrar();

    header('location: index.php?status=success');
    exit();
}
```

Ao clicar no editar, em uma das linhas, vai para pagina editar.php e aparece o titulo cadastrar, para alterar isso, adicionar na página editar.php

```
define ('TITLE', 'Editar vaga');
```

cadastrar.php

```
define ('TITLE', 'Cadastrar vaga');
```

```
define(string $constant_name, mixed $value, bool $case_insensitive = false): bool
```

Defines a named constant at runtime.

Esse método define o valor da constante em tempo de execução. Caso tenha 2 páginas distintas com o mesmo código, mas muda apenas o titulo, é possível utilizar o define

```
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.
```

formulario.php

adicionar a tag <h2 class = 'mt-3'><?=TITLE?></h2>

```
<main>
    <section>
        <a href = "/index.php">
            <button class="btn btn-success mt-5">Voltar</butt
        </a>
    </section>
    <h2 class = 'mt-3'><?=TITLE?></h2>
```

Agora será atribuido o valor dos campos o mesmo do banco de dados para editar

```
                <div class="form-group">
            <label>Título</label>
            <input type="text" class = "form-control" name="t
        </div>
```

Adicionado value="<?=$obVaga→titulo?”>

```
<div class = "form-group">
        <label>Descrição</label>
        <textarea class="form-control" name = "descricao" row
</div>
```

Adicionado <?=$obVaga→descricao?>

```
<div class = "form-check form-check-inline">
    <label class = "form-control">
        <input type="radio" name="ativo" value="n" <?=$obVaga
    </label>
</div>
```

Adicionado condição ternária do ativo. Caso o valor lido for igual a ‘n’ automaticamenete atriui ‘’

Editar.php

Ao editar o campo anterior, e dar submit, é necessário atualizar o novo registro do banco.

Ir até o validação do post e adicionar o $obVaga→atualizar();

Remover o $obVaga = new Vaga; pois ja está sendo instanciado na parte superior

```
if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo
   // $obVaga = new Vaga;
    $obVaga->titulo    = $_POST['titulo'];
```

```
    $obVaga->descricao = $_POST['descricao'];
    $obVaga->ativo     = $_POST['ativo'];

    $obVaga->atualizar();

    header('location: index.php?status=success');
    exit();
}
```

Vaga.php

Adicionar o método atualizar()

```
/**
    * Método responsável por atualizar a vaga no banco
    * @return [type]
    */

    public function atualizar(){
        return (new Database('vagas'))->update('id = '.$this-
                                                        '
                                                        '
                                                        '
                                                        '



    }
```

Dentro do Database será necessario criar o update

Database.php

Adicionar o campo de update do sql

```
/**
     * Método responsável por executar atualizações no banco 
     * @param string @where
     * @param array $values [ field=>value]
     * @return boolean
     */
    public function update($where, $values){
        $query = 'UPDATE vagas SET titulo = "titulo", descric
    }
```

Está é a query padrão que tem que deixar responsiva:

```
$query = 'UPDATE vagas SET titulo = "titulo", descricao = "de
```

Com os binds fica assim:

```
$query = 'UPDATE '.$this->table.'SET titulo =?,descricao =? W
```

Utilizando o array_keys:

```
public function update($where, $values){
        //DADOS QUERY
        $fields = array_keys($values);

        //MONTA QUERY
        $query = 'UPDATE '.$this->table.' SET '.implode('=?,'
        echo $query;
        exit();
    }
```

Para excluir uma linha, é possível copiar a classe editar.php pois ela tem a validação do ID, a consulta e a validação da vaga

Duplicar o editar.php e renomear o novo arquivo como excluir.php. Colocá-lo na pasta em que esta o index e editar.

Apagar o define ('TITLE', 'Editar vaga');

```php
<?php

require __DIR__.'/vendor/autoload.php';

use \App\Entity\Vaga;

// obrigado usuario a passar ID. Validação do ID
if(!isset($_GET['id']) or !is_numeric($_GET['id'])){
    header('location: index.php?status=error');
    exit();
}

// consulta vaga
$obVaga = Vaga::getVaga($_GET['id']);

//Validação da vaga
if(!$obVaga instanceof Vaga){
   header('location: index.php?status=error');

}

echo "<pre>"; print_r($obVaga); echo "</pre>";exit;
//validação do post
if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo

   // $obVaga = new Vaga;
    $obVaga->titulo    = $_POST['titulo'];
    $obVaga->descricao = $_POST['descricao'];
    $obVaga->ativo     = $_POST['ativo'];
    $obVaga->atualizar();

    header('location: index.php?status=success');
    exit();
}


include __DIR__.'/includes/header.php';
```

```
include __DIR__.'/includes/formulario.php';
include __DIR__.'/includes/footer.php';
```

Duplicar o arquivo formulario.php na pasta includes e adicionar o nome confirmar-exclusão.php

confirmar-exclusao.php

substituir o include formulario.php pelo confirmar-exclusao

```
include __DIR__.'/includes/header.php';
include __DIR__.'/includes/confirmar-exclusao.php';
include __DIR__.'/includes/footer.php';
```

será mantido o botão voltar, e substituir a linha dinâmica<h2 *class* = 'mt-3'><?=TITLE?></h2> pela

<h2 *class* = 'mt-3'>Excluir vaga</h2>

Ficará assim:

```
<main>
    <section>
        <a href = "/index.php">
            <button class="btn btn-success mt-5">Voltar</butt
        </a>
    </section>
    <h2 class = 'mt-3'>Excluir vaga</h2>
    <form method="POST">
        <div class="form-group">

        </div>
        <div class = "form-group">
            <button type="submit" class = "btn btn-success">E
        </div>

    </form>
</main>
```

```
<main>
    <h2 class = 'mt-3'>Excluir vaga</h2>

    <form method="POST">

        <div class="form-group">
            <p> Você deseja realmente excluir a vaga <strong>
        </div>

        <div class = "form-group">
            <a href = "/index.php">
                <button type ="button" class="btn btn-success
            </a>

            <button type="submit" name ="excuir" class = "btn
        </div>

    </form>
</main>
```

excluir.php

substituir o

```
if(isset($_POST['titulo'], $_POST['descricao'], $_POST['ativo
```

pelo

```
if(isset($_POST['excluir'])){
```

fica assim:

```
//validação do post
if(isset($_POST['excluir'])){
```

```
    $obVaga->excluir();

    header('location: index.php?status=success');
    exit();
}
```

Vaga.php

Adicionar a função excluir

```
public function excluir(){
        return (new Database ('vagas'))->delete('id = '.$this
    }
```

Database.php

```
/**
     * Método responsável por excluir dados do banco
     * @param string
     * @return boolean
     */
    public function delete($where){
        //monta query
        $query = 'DELETE FROM '.$this->table.' WHERE '.$where

        //executa a query
        $this->execute($query);

        //retorna sucesso

        return true;
```

```
    }
```

Agora o código ja esta fazendo o CRUD básico

Listagem.php

Para colocar uma mensagem mais amigável adicionar:

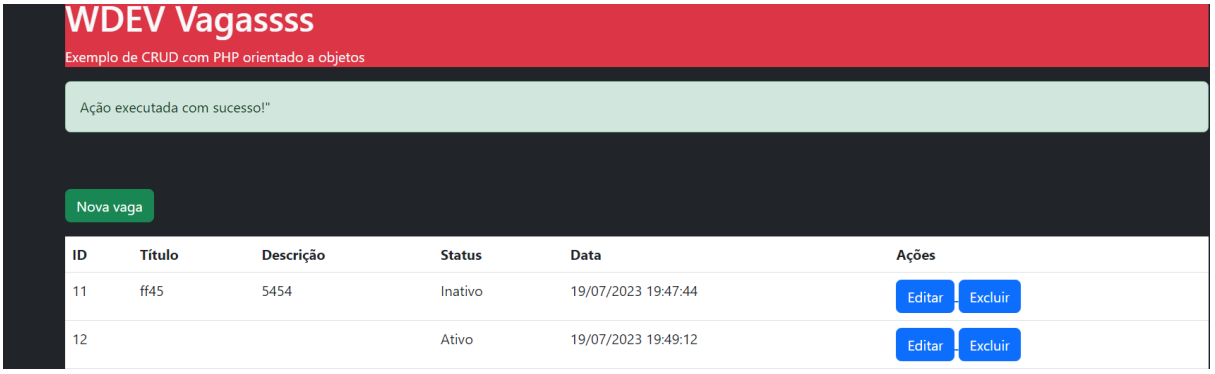
Adicionar o inicio do código

```
<?php
    $mensagem ='';
    if(isset($_GET['status'])){
        switch($_GET['status']){
            case 'success':
                $mensagem = '<div class = "alert alert-succes
                break;
            case 'error':
                $mensagem = '<div class = "alert alert-danger
                break;
        }
    }
```

Adicionar a variável <?=$mensagem?> php no código html

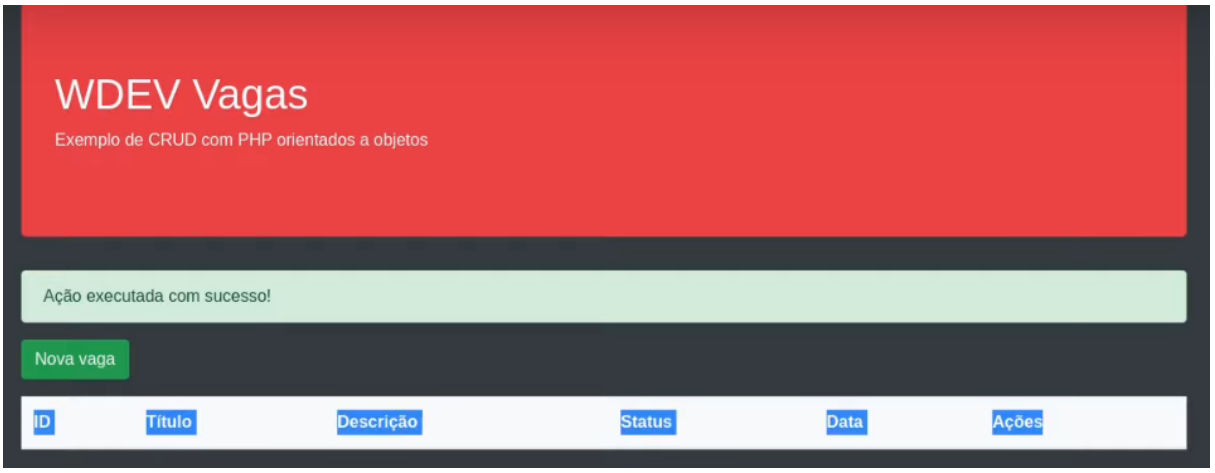
```
<main>
    <?=$mensagem?>
    <section>
        <a href="/includes/cadastrar.php">
```

Ao testar o código:



Aparece a mensagem ação executada com sucesso

Ao apagar todos os registros, ficam a mostra o nome do titulo de cada coluna



Como não há nenhum registro, o ideal é remover e aparecer uma mensagem que não há registros.

Listagem.php

adicionar o seguinte código

```
$resultados = strlen($resultados) ? $resultados : ' <tr>
                                                    <
                                                    <
                                                </tr>
```

strlen($resultados) → caso tenha dados na variavel $resultados

caso tenha, irá imprimir $resultados

caso não irá substituir o conteudo pela tag html

código final

```
<?php
    $mensagem ='';
    if(isset($_GET['status'])){
        switch($_GET['status']){
            case 'success':
                $mensagem = '<div class = "alert alert-succes
                break;
            case 'error':
                $mensagem = '<div class = "alert alert-danger
                break;
        }
    }
    $resultados ='';
    foreach($vagas as $vaga){
        $resultados .='<tr>
                            <td>'.$vaga->id.'</td>
                            <td>'.$vaga->titulo.'</td>
                            <td>'.$vaga->descricao.'</td>
                            <td>'.($vaga->ativo == 's' ? 'Ati
                            <td>'.date('d/m/Y H:i:s',strtotim
                            <td>
                            <a href="editar.php?id='.$vaga->i
                              <button type ="button" class="b
                            </a>
                            <a href="excluir.php?id='.$vaga->
                              <button type ="button" class="b
                            </a>
                            </td>
                        </tr>';
    }
    $resultados = strlen($resultados) ? $resultados : ' <tr>
```

```
                                                                    <

                                                                    <
                                                                </tr>
?>
<main>
    <?=$mensagem?>
    <section>
        <a href="/includes/cadastrar.php">
            <button class = "btn btn-success mt-5">Nova vaga<
        </a>
    </section>

    <section>
        <table class = "table bg-light mt-3">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Título</th>
                    <th>Descrição</th>
                    <th>Status</th>
                    <th>Data</th>
                    <th>Ações</th>
                </tr>
            </thead>
            <tbody>
                <?=$resultados?>
            </tbody>
        </table>
    </section>

</main>
```