amaralluiz Alterando apostila para explicacao da alteracao do commit 3cdd532

1 contributor

Introdução ao PHP

PHP é uma linguagem de programação que foi criada em 1995 por Rasmus Lerdorf. Ela é considerada uma linguagem interpretada e dinamicamente tipada.

Variáveis

Para declararmos variáveis no PHP usamos a seguinte sintaxe: `$variavel` .

Tipo inteiro

Um inteiro é um número do conjunto Z={..., -2, -1, 0, 1, 2, ...}.

Sintaxe: `$a = 1234;`

Tipo ponto flutuante

Números de ponto flutuantes são os números reais, ou seja um conjunto que engloba aos números positivos, negativos, decimais, fracionários, zero, além das dízimas periódicas e não periódicas.

Sintaxe: `$a = 1.234;`

Tipo string

Uma string é uma série de carácteres, onde um caractere é o mesmo que um byte.

Sintaxe: `$a = 'String com aspas simples';`
`$b = "String com aspas duplas";`

Tipo array

Array é uma lista de valores armazenados na memória, os quais podem ser de tipos diferentes (números, strings, objetos) e podem ser acessados a qualquer momento, pois cada valor é relacionado a uma chave. Um array também pode crescer dinamicamente com adição de novos itens.

Sintaxe: `$a = array('Luiz', 'Raíssa', 'Donnie');`
`$b = ['Luiz', 'Raíssa', 'Donnie'];`

Observação: No PHP, arrays iniciam na posição 0.

Tipo Objeto

Um objeto é uma entidade com um determinado comportamento definido por seus métodos (ações) e propriedades (dados). Para instanciar um objeto deve-se utilizar o operador `new`.

Sintaxe:

```php
<?php

class Carro
{
    public $modelo;
    function darPartida()
    {
        echo "Dando partida no carro modelo $this->modelo...";
    }
}

$obj = new Carro;
$obj->modelo = "Palio";
$obj->darPartida();
```

Null

O valor especial `null` representa uma variável sem valor. `null` é o único valor possível do tipo `null`.

A variável é considerada `null` se:

- Foi atribuída a constante `null`;

- Ainda não recebeu nenhum valor;
- Foi apagada com `unset()`.

Tipo booleano

Um tipo booleano expressa somente um valor de verdade. Ele pode ser `true` ou `false`.
Por ser dinamicamente tipado, qualquer valor será convertido automaticamente se um operador, função ou estrutura de controle requerer um argumento bool.

Ao converter para bool, os seguintes valores são considerados `false`:

- O próprio booleano `false`;
- O inteiro 0;
- Os pontos flutuantes 0.0 e -0.0;
- Uma string vazia e a string `"0"`;
- Um array sem elementos;
- O tipo especial `null`;

Observação: o tipo -1 é considerado `true`, como qualquer outro número que não seja zero, tanto negativo quanto positivos.

Sintaxe: `$a = true;`

`$b = false;`

Operadores

Um operador é algo que recebe um ou mais valores (ou expressões, no jargão de programação) e que devolve outro valor.

Atribuição

Um operador de atribuição é utilizado para definir uma variável atribuindo-lhe um valor.
O operador básico de atribuição é `=`.

Sintaxe:

```php
<?php
$a = 10; // Atribui 10 em $a
$a += 5; // Soma 5 em $a
$a -= 5; // Subtrai 5 em $a
$a *= 5; // Multiplica $a por 5
$a /= 5; // Divide $a por 5
```

Aritmético

Operadores artméticos são utilizados para realizações de cálculos matemáticos.

Operador	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo, ou resto, da divisão

Observação: O PHP segue a ordem de operações matemáticas para cálculos complexos.

Sintaxe:

```
<?php
$a = 3 + 4; // Resultado 7
$b = 2 - 1; // Resultado 1
$c = 2 * 2; // Resultado 4
$d = 6 / 3; // Resultado 2
$e = 7 % 2; // Resultado 1
```

Relacionais

Operadores relacionais são utilizados para realizar comparações entre valores ou expressões, resultando sempre em um valor booleano.

Operador	Operação
==	Igual
===	Idêntico
!=	Diferente
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual

Lógicos

Operadores lógicos são utilizados para combinar expressões lógicas entre si, agrupando testes condicionais.

Operador	Operação	Resultado
$a and $b	AND/E	Verdade se tanto $a quanto $b forem verdadeiros
$a or $b	OR/OU	Verdade se $a ou $b forem verdadeiros
$a xor $b	XOR	Verdadeiro se $a ou $b forem verdadeiros, de forma exclusiva
!$a	NOT/Negação	Verdadeiro se $a for false
$a && $b	AND/E	Verdade se tanto $a quanto $b forem verdadeiros
$a || $b	OU/OU	Verdade se $a ou $b forem verdadeiros

Observação: `or` e `and` têm precedência menor que `&&` ou `||`.

Estruturas de Controle

Uma estrutura de controle é um bloco de programação que analisa variáveis e escolhe uma direção para seguir baseada em parâmetros pré-definidos.

Estruturas de seleção

IF

Sintaxe:

```
<?php
if(expressão1) {
    // comandos se a expressão é verdadeira
} else if(expressão2) {
    // comandos se a expressão2 for verdadeira
} else {
    // comandos se expressão 1 e 2 forem falsas
}
```

O `else if` é utilizado caso queira realizar mais verificações e será executado somente se o bloco anterior for falso.

SWITCH

Sintaxe:

```php
<?php
switch($variavel) {
    case valor1:
        // comandos
        break;
    case valor2:
        // comandos
        break;
    default:
        // comandos
}
```

Observações: o comando default será executado somente se nenhuma das verficações retornarem verdadeiro.

Estruturas de repetição

Estrutura que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador.

FOR

Sintaxe:

```php
<?php
for(expr1; expr2; expr3) {
    // comandos
}
```

expr1: Valor inicial da variável contadora.
expr2: Condição de execução. Enquanto for verdade o bloco de comandos será executado.
expr3: Valor a ser incrementado após cada execução.

Exemplo:

```php
<?php
for($i = 0; $i <= 10; $i++>) {
    echo $i .PHP_EOL;
}
```

WHILE

Sintaxe:

```php
<?php
while(expr1) {
    // comandos
}
```

expr1: Expressão que deverá ser verdade para execução do bloco.

Exemplo:

```php
<?php
$i = 0;
while($i <= 10) {
    echo $i . PHP_EOL;
    $i++;
}
```

Manipulação de Arrays

Arrays são acessados mediante uma posição, como um índice numérico.

Podemos criar um array das seguintes formas:

```php
<?php

// Primeira forma, utilizando array()

$cores = array('vermelho', 'azul', 'verde');

// Segunda forma, utilizando sintaxe resumida

$cores = ['vermelho', 'azul', 'verde'];

// Terceira forma, adicionando valores

$cores[] = 'vermelho';
$cores[] = 'azul';
$cores[] = 'verde';
```

Para acessar o array indexado, basta indicar o seu índice entre colchetes:

```php
<?php
echo $cores[0]; // resultado = vermelho
echo $cores[1]; // resultado = azul
echo $cores[2]; // resultado = verde
```

Arrays associativos

Os arrays no PHP são associativos, pois contêm uma chave de acesso para cada posição.

Podemos criar arrays associativos das seguintes formas:

```php
<?php

// Utilizando array()

$cores = array(
    'vermelho' => 'FF0000',
    'azul' => '0000FF',
    'verde' => '00FF00',
    );

// Utilizando sintaxe resumida e adicionando valores

$cores = [];
$cores['vermelho'] = 'FF0000';
$cores['azul'] = '0000FF';
$cores['verde'] = '00FF00';
```

Para acessar, basta indicar a sua chave entre colchetes:

```php
<?php

echo $cores['vermelho']; // resultado FF0000
echo $cores['azul']; // resultado '0000FF'
echo $cores['verde'];// resultado '00FF00'
```

Iterações

Os arrays podem ser iterados no PHP pelo operador `foreach`, que percorre cada uma das posições do array.

Sintaxe:

```php
<?php

$frutas = array();
$frutas['cor'] = 'vermelho';
$frutas['sabor'] = 'doce';
$frutas['formato'] = 'redondo';
$frutas['nome'] = 'maçã';

foreach($frutas as $chave => $valor) {
```

```
    echo "$chave => $valor \n";
}
```

Arrays multidimensionais

Arrays multidimensionais ou matrizes são arrays nos quais algumas de suas posições podem conter outros arrays.

Podemos criar arrays multidimensionais das seguintes formas:

```
<?php

// Utilizando array()

$carros = array(
    'Palio' => array(
        'cor' => 'azul',
        'potencia' => '1.0',
        'opcionais' => 'Ar Cond.'
    ),
    'Corsa' => array(
        'cor' => 'cinza',
        'potencia' => '1.3',
        'opcionais' => 'mp3'
    ),
    'Gol' => array(
        'cor' => 'branco',
        'potencia' => '1.0',
        'opcionais' => 'metalica'
    )
);

echo $carros['Palio']['opcionais'];

// Utilizando sintaxe resumida e adicionando valores

$carros = [];
$carros['Palio']['cor'] = 'azul';
$carros['Palio']['potencia'] = '1.0';
$carros['Palio']['opcionais'] = 'Ar Cond.';
$carros['Corsa']['cor'] = 'cinza';
$carros['Corsa']['potencia'] = '1.3';
$carros['Corsa']['opcionais'] = 'mp3';
$carros['Gol']['potencia'] = 'branco';
$carros['Gol']['cor'] = '1.0';
$carros['Gol']['opcionais'] = 'metalica';

echo $carros['Palio']['opcionais'];
```

Para realizar iterações em um array multidimensional, é preciso observar quantos níveis ele tem.

```php
<?php

$carros = [];
$carros['Palio']['cor'] = 'azul';
$carros['Palio']['potencia'] = '1.0';
$carros['Palio']['opcionais'] = 'Ar Cond.';
$carros['Corsa']['cor'] = 'cinza';
$carros['Corsa']['potencia'] = '1.3';
$carros['Corsa']['opcionais'] = 'mp3';
$carros['Gol']['potencia'] = 'branco';
$carros['Gol']['cor'] = '1.0';
$carros['Gol']['opcionais'] = 'metalica';

foreach ($carros as $modelo => $caracteristicas) {
    echo "=> modelo $modelo\n";
    foreach ($caracteristicas as $caracteristica => $valor) {
        echo " - caracteristica $caracteristica => $valor\n";
    }
}
```

Funções para manipulação de Arrays

array_merge

Mescla dois ou mais, adicionando um array ao final de outro, resultando em um novo array. Se ambos os arrays tiverem conteúdo indexado pela mesma chave, o segundo array irá se sobrepor ao primeiro.

```php
<?php
$a = ['verde', 'azul'];
$b = ['vermelho', 'amarel'];
$c = array_merge($a, $b);
var_dump($c);
```

array_push

Adiciona elementos ao final de um array.

```php
<?php
$a = ['verde', 'azul'];
$b = ['vermelho', 'amarelo'];
array_push($a, $b);
var_dump($a);
```

explode

Converte uma string em um array, quebrando os elementos por meio de um separador.

implode

Converte um array em uma string, agrupando os elementos do array por meio de um elemento "cola".

Exemplo de implode e explode:

```php
<?php

// Explode

$string = "10/05/2015";
var_dump(explode("/", $string));

// Implode

$array = [10, 5, 2015];
var_dump(implode('/', $array));
```

Funções

Uma função é um pedaço de código com um objetivo específico, encapsulado sob uma estrutura única que recebe um conjunto de parâmetros e retorna um dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes.

Sintaxe:

```php
<?php

// criação

function nomeDaFuncao (arg1, arg2, arg3)
{
    $valor = $arg1 + $arg2 + $arg3;
    return $valor;
}

// chamada
nomeDaFuncao(arg1, arg2, arg3);
```

Variáveis declaradas dentro do escopo de uma função são locais, ou seja, só podem ser acessadas dentro do escopo daquela função.
Para acessar uma variável fora do escopo de uma função sem passá-la como argumento, ela deve ser criada no escopo acima ou como global.

PSR

PSR, ou PHP Standards Recommendations, são as recomendaçãoes padrões do PHP. Elas foram criadas por um grupo chamado PHP-FIG, que significa PHP Framework Interop Group e é um grupo formado por desenvolvedores da comunidade PHP em 2009.

O intuito original do grupo era definir recomendações que fossem aplicadas aos Frameworks PHP participantes para facilitar a interoperabilidade entre os frameworks que cresciam rapidamente.

Devido a grande utilização de PHP em outros tipos de aplicações além de frameworks, como CMS, o grupo passou a possuir membros de outros tipos de projetos, não somente frameworks.

As recomendações criadas pelo grupo são agrupadas em uma PHP Standard Recommendation (PSR). Cada PSR possui recomendações sobre um tema específicio, como a PSR-12 que fala sobre padronização de escrita de código PHP.

Aqui você pode ler a PSR-12 traduzida para português.

Criação de API Utilizando Laravel

Iniciado o projeto

Para iniciar o projeto, precisaremos estar com o PHP e o Composer já instalados. O primeiro passo é ir na pasta onde está instalado o seu PHP e procurar o arquivo `php.ini`. Caso ele não exista, basta criar uma cópia do arquivo `php.ini-development` e alterar seu nome para `php.ini`. Após isso, abra utilizando seu editor de texto, podendo ser até o bloco de notas, e procure por `extension=openssl` e `extension=pdo_sqlite`, nessas duas linhas retire o `;` que inicia ambas.

Após isso, abra seu terminal, navegue até a pasta que deseja que seu projeto será criado e rode o comando `composer create-project laravel/laravel:^8.0 teknisa-dev-api` no terminal. Ele rodará o composer, criando um projeto utilizando o Laravel 8 com o nome do projeto como teknisa-dev-api.

Criando e configurando o banco de dados

Com o projeto já criado iremos no nosso arquivo `.env` e procurar por `DB_CONNECTION`, precisamos deixar ele dessa forma:

```
DB_CONNECTION=sqlite
#DB_HOST=127.0.0.1
#DB_PORT=3306
#DB_DATABASE=laravel
#DB_USERNAME=root
#DB_PASSWORD=
```

Dessa forma, já setamos que o banco utilizado é o sqlite. O Laravel por padrão procurará o arquivo de banco de dados dentro da pasta `database`, então dentro dela crie um arquivo chamado `database.sqlite`. Este será nosso banco de dados.

Criação da tabela utilizando Migration

Iremos utilizar a CLI que vem no Laravel que é a artisan para rodar alguns comandos que aumentam a nossa produtividade. O comando que vamos utilizar no momento será o `php artisan make:migration create_devs_table --create=devs`. Ele utilizará o php para rodar o artisan criando uma migration chamada create_devs_table e com o nome da tabela como devs.

Agora iremos abrir este arquivo, ele fica dentro das pastas `database/migrations` e seu nome será a data de criação e no final `create_devs_table.php`. Nele iremos adicionar as seguintes linhas após `$table->id();`:

```
$table->string('name');
$table->string('email');
$table->integer('age');
$table->string('picture');
$table->text('programmingLanguages');
```

Aqui estamos definindo os campos que precisamos dentro da nossa tabela, conforme o projeto do Figma. As migrations servem como um controle de versão para nosso esquema de banco de dados. Dessa forma não precisamos criar nenhum campo ou tabela manualmente após realizar alteração no código.

Para rodar a migration utilizamos o comando `php artisan migrate` no nosso terminal. Dessa forma o artisan irá executar todos os arquivos migrate do projeto e podemos ver no `database.sqlite` que a tabela `devs` foi criada. Para ver podemos utilizar por exempo o SQLite Browser.

Criação de Model, Controller e Resource

Como temos nossa tabela pronta, podemos criar nossos modelos, controllers e resources. Primeiro vamos criar o Model com o comando `php artisan make:model Dev`, com ele criamos um model com o nome `Dev` na pasta `app/Models`. O nosso Model será a classe que utilizará o Eloquente, que é o ORM utilizado pelo Laravel, para fazer a tradução do objeto entidade-relacional do banco para um objeto no PHP.

Após isso, vamos criar nosso resource com o `php artisan make:resource Dev`, este comando irá criar um resource com o nome `Dev` na pasta `app/Http/Resources`. Este arquivo será o responsável por traduzir o nosso Dev para um retorno amigável em array, para passarmos via JSON na resposta da requisição que o cliente fará para a nossa API.
Neste arquivo iremos retirar o `return parent::toArray($request);` e adicionar o seguinte:

```
return [
    'id' => $this->id,
    'name' => $this->name,
    'email' => $this->email,
    'age' => $this->age,
    'picture' => $this->picture,
```

645 lines (459 sloc) | 20.6 KB

Com isso iremos definir o que é cada campo do nosso retorno conforme o que foi definido no nosso projeto. Usaremos o método `explode()` no retorno de `programmingLanguages` para termos ele retornando em formato de array, o que irá facilitar para quem irá trabalhar este campo.

Agora podemos criar nosso controller, que é de fato quem controlará o que cada rota da nossa API irá fazer. Para criar o controller o comando é `php artisan make:controller DevController --resource`.
O primeiro passo neste arquivo é colocar as dependências que vamos utilizar no início do arquivo. Iremos colocar as seguintes linhas logo após a definição do `namespace`:

```
use App\Models\Dev as Dev;
use App\Http\Resources\Dev as DevResource;
```

Com isso nosso controller saberá onde procurar quando formos utilizar as classes Dev e DevResource do nosso Model e Resource.

Depois disso, nas definições de função da classe `DevController` iremos definir as seguintes funções:

```php
public function index()
{
    $devs = Dev::paginate(100);
    return DevResource::collection($devs);
}

public function show($id){
    $dev = Dev::findOrFail( $id );
    return new DevResource( $dev );
}

public function store(Request $request){
    $dev = new Dev;
    $dev->name = $request->input('name');
    $dev->email = $request->input('email');
    $dev->age = $request->input('age');
    $dev->picture = $request->input('picture');
    $dev->programmingLanguages = $request->input('programmingLanguages');

    if( $dev->save() ){
        return new DevResource( $dev );
    }
}

public function update(Request $request){
    $dev = Dev::findOrFail( $request->id );
    $dev->name = $request->input('name');
    $dev->email = $request->input('email');
    $dev->age = $request->input('age');
    $dev->picture = $request->input('picture');
    $dev->programmingLanguages = $request->input('programmingLanguages');

    if( $dev->save() ){
        return new DevResource( $dev );
    }
}

public function destroy($id){
    $dev = Dev::findOrFail( $id );
    if( $dev->delete() ){
        return new DevResource( $dev );
    }
}
```

Cada uma dessas funções será a responsável por uma das ações do nosso CRUD.
Na função `index` estamos somente buscando os devs salvos no nosso banco de dados, paginando ele com 100 em cada página e retornando uma coleção de devs utilizando a `collection` do `DevResource`.
Na função `show` iremos utilizar o método `findOrFail` do próprio eloquent para encontrar o id de um dev que será passado via QueryString numa das rotas da nossa API.
Na função `store` é onde iremos salvar nosso dev no banco de dados. Nele criaremos uma instância de Dev e iremos popular seus campos de acordo com o que chegará no nosso Request. Depois iremos acionar o método save e caso ele retorne sucesso, será retornado o dev salvo para o cliente.
Na função `update` vamos utilizar uma lógica parecida com a função `store`, mas no local de instanciar um novo dev, iremos buscar o dev passado via QueryString e atualizar seus campos conforme o Request passado pelo cliente. Depois disso o método save será chamado e caso dê sucesso será devolvido o dev atualizado para o cliente.
Por último, na função `destroy` iremos buscar o dev passado por QueryString e chamar o método delete, caso dê sucesso vamos retornar o dev deletado.

Criando as rotas

Agora para finalizar podemos criar nossas rotas. Podemos ir na pasta `routes` e no arquivo `api.php` vamos adicionar as seguintes linhas:

```
Route::get('devs', [DevController::class, 'index']);
Route::get('dev/{id}', [DevController::class, 'show']);
Route::post('dev', [DevController::class, 'store']);
Route::put('dev/{id}', [DevController::class, 'update']);
Route::delete('dev/{id}', [DevController::class, 'destroy']);
```

Cada uma dessas rotas ficará responsável por um dos métodos que criamos no nosso `DevController` de acordo com o padrão REST.

Depois disso podemos utilizar o comando `php artisan serve` para subir nosso projeto em um ambiente de desenvolvimento. Com isso já podemos utilizar o Postman para realizar os testes da nossa api nas rotas corretas. Por padrão o artisan subirá a aplicação no `localhost:8000` e como iremos utilizar a api, a url completá será `http://127.0.0.1:8000/api/`