**My Laravel Tour**

Fluent Class

**Laravel Fluent Class Example**

**Overview**

The Fluent class in Laravel (Illuminate\Support\Fluent) allows you to create dynamic objects that can hold properties as key-value pairs. This document illustrates how to use the Fluent class and what happens when you attempt to access a property that does not exist.

**Example Code**

php

Copy code

use Illuminate\Support\Fluent;

// Initializing an array

$a = [

'a' => 1,

'b' => 3

];

// Creating a Fluent object from the array

$a = new Fluent($a);

// Attempting to access a non-existing property 'c'

dd($a->c);

**Explanation**

1. **Creating a Fluent Object**:
   * We start by initializing a regular PHP array $a with two key-value pairs:
     + 'a' with a value of 1
     + 'b' with a value of 3
   * Next, we create a Fluent object by passing the array $a to the Fluent constructor. This converts the array into a dynamic object.
2. **Accessing Existing Properties**:
   * You can access existing properties of the Fluent object using the property syntax. For example:

php

Copy code

echo $a->a; // Output: 1

echo $a->b; // Output: 3

1. **Accessing a Non-Existing Property**:
   * When you try to access a property that does not exist (like 'c' in this case), the Fluent class will return null without throwing an error.
   * The line dd($a->c); will output:

csharp

Copy code

null

* + dd() is a helper function in Laravel that stands for "dump and die." It dumps the provided variable and stops the execution of the script. Therefore, it shows null when accessing a non-existing property.

**Conclusion**

The Fluent class provides a convenient way to work with dynamic properties. It allows for easy creation of objects from arrays and enables property access using a more readable syntax. Accessing non-existing properties returns null, making it safe to use without additional checks.

WhereAny and WhereAll

whereAny() method

This method will stop developers having to chain the ->orWhere()method on their queries and instead, encapsulate all of that logic inside a single ->whereAny() method. Therefore, instead of the following query being written

$search = '%Otwell%';

User::query()

->where(function ($query) use ($search) {

$query

->where('first\_name', 'LIKE', $search)

->orWhere('last\_name', 'LIKE', $search)

->orWhere('email', 'LIKE', $search)

->orWhere('phone', 'LIKE', $search);

})

It can instead now be refactored into ->whereAny()which will search an array of columns provided for the search query provided. It will look something like the below code snippet

$search = 'Otwell';

User::whereAny(['first\_name','last\_name','email'], 'LIKE', "%$search%")

The SQL that this new method will output, using logic from above is below. This will return any result if one or more columns contain the search query.

SELECT \* FROM "users" WHERE (

"first\_name" LIKE "%Otwell%"

OR "last\_name" LIKE "%Otwell%"

OR "email" LIKE "%Otwell%"

)

whereAll() method

Similar to the whereAny() a whereAll() method has been contributed, which will check that the search query exists in all columns instead of just one or more. The function looks like the following;

$search = 'Otwell';

User::whereAll(['first\_name','last\_name','email'], 'LIKE', "%$search%")

The SQL that ->whereAll()will be very similar to the ->whereAny() which replaces the OR from the SQL to AND, which will only return results if ALL fields contain the search query.

SELECT \* FROM "users" WHERE (

"first\_name" LIKE "%Otwell%"

AND "last\_name" LIKE "%Otwell%"

AND "email" LIKE "%Otwell%"

)

Once() method in Laravel

In this short post, I will show you how to use once() helper function in laravel 11 framework.

In Laravel 11, a new helper function called `once()` has been added. It ensures that you get the same value no matter how many times you call an object method. The `once()` function is valuable when you need to guarantee that a certain piece of code executes only once.

Let's see how to use it with the example code:

First, we will create a simple OnceTest class and create two methods to generate a random string. One will simply return a random 10 characters, and for another function we will use the once helper function it.

Run the following command to create a new class:

php artisan make:class OnceTest

Ezoic

app/OnceTest.php

<?php

namespace App;

use Illuminate\Support\Str;

class OnceTest

{

/\*\*

\* Write code on Method

\*

\* @return response()

\*/

public function random()

{

return Str::random(10);

}

/\*\*

\* Write code on Method

\*

\* @return response()

\*/

public function randomOnce()

{

return once(function () {

return Str::random(10);

});

}

}

Now, we will simply use that class with a function like the code below:

Ezoic

app/Http/Controllers/ProductController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\OnceTest;

class ProductController extends Controller

{

/\*\*

\* Write code on Method

\*

\* @return response()

\*/

public function index()

{

$obj = new OnceTest();

$random1 = $obj->random();

$random2 = $obj->random();

$randomOnce1 = $obj->randomOnce();

$randomOnce2 = $obj->randomOnce();

$randomOnce3 = $obj->randomOnce();

dump($random1, $random2, $randomOnce1, $randomOnce2, $randomOnce3);

}

}

You will get the following output:

Read Also: Laravel 11 New Model::casts() Method Example

OUPUT

"2XxMsJ4Jyo" // app/Http/Controllers/ProductController.php:28

"thfl0Xk5Rq" // app/Http/Controllers/ProductController.php:28

"eGfSNbJbOD" // app/Http/Controllers/ProductController.php:28

"eGfSNbJbOD" // app/Http/Controllers/ProductController.php:28

"eGfSNbJbOD" // app/Http/Controllers/ProductController.php:28