Les collections

Les Collections sont un type de classe qui permet de représenter une série de donnée et permettent d'apporter une solution plus objet au traitement des tableaux.

Pourquoi ne pas utiliser les tableaux ?

Avant de vouloir rajouter un nouveau concept dans notre application il est important de comprendre les problèmes que pose l'utilisation des tableaux.

```
collect($eleves)
  ->filter(function($eleve) {
    return $eleve['note'] >= 10;
  })
  -> map(function($eleve) {
    return $eleve['name']
  })
  ->values()
  ->toArray();
```

Laravel dispose d'un système de Collection plutôt complet mais n'offre pas de package séparé du framework.

Cette classe offre une centaine de méthodes et dispose d'un système original de Higher Order Messages.

```
collect($eleves)
  -> map(function($eleve) {
    return $eleve['name']
  })
  ->values()
  ->toArray()
```

Liste des fonctions

- **all()**

Dans les collections de Laravel, il y a des fonctions relativement basiques, celle-ci en est une.

Elle te retourne un tableau avec tous les éléments de la collection.

Laravel Collection - La fonction All()

```
$languages = collect([
            'php',
            'python',
            'javascript',
            'go',
            'c#',
            'java',
            'cobol',
            'basic'
        ]);

dd($languages->all());
```

Le résultat de cet appel donnera ceci :

```
array:8 [▼
  0 => "php"
  1 => "python"
  2 => "javascript"
  3 => "go"
  4 => "c#"
  5 => "java"
  6 => "cobol"
  7 => "basic"
]
```

- **avg()**

Bon pour faire simple avg() va te calculer la moyenne des valeurs comprises dans ta collection. Attention, cela parait évident mais cela ne fonctionne qu'avec des valeurs numériques.

Donc, revenons-en à nos chiffres :

Laravel Collection - La fonction Avg() ou Average()

```
$numbers = collect([-2, 200.3, -7.8, 400.1])->avg();
dd($numbers);

// retourne
```

```
147.65
```

Je t’ai expliqué plus haut qu’il était possible de chaîner les fonctions dans les collections.

- **reject()**

Alors regardes, cet exemple va te faire entre-apercevoir la puissance qui se niche dans les fonctions de la classe Collection.

Laravel Collection - La fonction Avg() combinée à la fonction reject()

```
$numbers = collect([-2, 200.3, -7.8, 400.1])
   ->reject(function ($number) {
         return $number < 0;
      })
   ->avg();

dd($numbers);

// retourne
300.2
```

J’effectue tout d’abord le rejet des valeurs inférieures à ‘0’ et ensuite seulement je fais ma moyenne.

- **Contains**

La méthode ***contains*** détermine si la collection contient un élément donné. Vous pouvez passer une fermeture à la méthode ***contains*** pour déterminer s'il existe un élément dans la collection correspondant à un test de vérité donné :

```
$collection = collect([1, 2, 3, 4, 5]);


$collection->contains(function (int $value, int $key) {
   return $value > 5;
});


// false
```

Vous pouvez également passer une valeur à la méthode **contains** pour déterminer si la collection contient une valeur d'élément donnée :

```
   $var1=$collection->contains(48);
//false
```

- ## Collapse

La méthode ***collapse*** permet de réduire une collection de tableaux en une seule collection plate :

```
$collection = collect([
   [1, 2, 3],
   [4, 5, 6],
   [7, 8, 9],
]);

$collapsed = $collection->collapse();

$collapsed->all();

// [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- ## count()

La méthode ***count*** renvoie le nombre total d'éléments de la collection :

```
$collection = collect([1, 2, 3, 4]);

$collection->count();

// 4
```

- ## countBy()

La méthode ***countBy*** compte les occurrences des valeurs dans la collection. Par défaut, la méthode compte les occurrences de chaque élément, ce qui vous permet de compter certains "types" d'éléments dans la collection :

```
$collection = collect([1, 2, 2, 2, 3]);
```

```
$counted = $collection->countBy();

$counted->all();

// [1 => 1, 2 => 3, 3 => 1]
```

- **filter()**

La méthode ***filter*** filtre la collection à l'aide du callback donné, en ne conservant que les éléments qui satisfont à un test de vérité donné. Cette méthode est l'inverse de ***reject()***.

```
$collection = collect([1, 2, 3, 4]);

$filtered = $collection->filter(function (int $value, int $key) {
    return $value > 2;
});

$filtered->all();

// [3, 4]
```

- **first()**

La méthode ***first()*** renvoie le premier élément de la collection qui passe un test de vérité donné :

```
collect([1, 2, 3, 4])->first(function (int $value, int $key) {
    return $value > 2;
});
// 3
```

Vous pouvez également appeler la méthode first sans arguments pour obtenir le premier élément de la collection. Si la collection est vide, elle renvoie null :

```
collect([1, 2, 3, 4])->first();
```

```
// 1
```

- **map**

La méthode ***map*** parcourt la collection et transmet chaque valeur au callback donné. Le callback est libre de modifier l'élément et de le renvoyer, formant ainsi une nouvelle collection d'éléments modifiés :

```
$collection = collect([1, 2, 3, 4, 5]);

$multiplied = $collection->map(function (int $item, int $key) {
    return $item * 2;
});

$multiplied->all();

// [2, 4, 6, 8, 10]
```

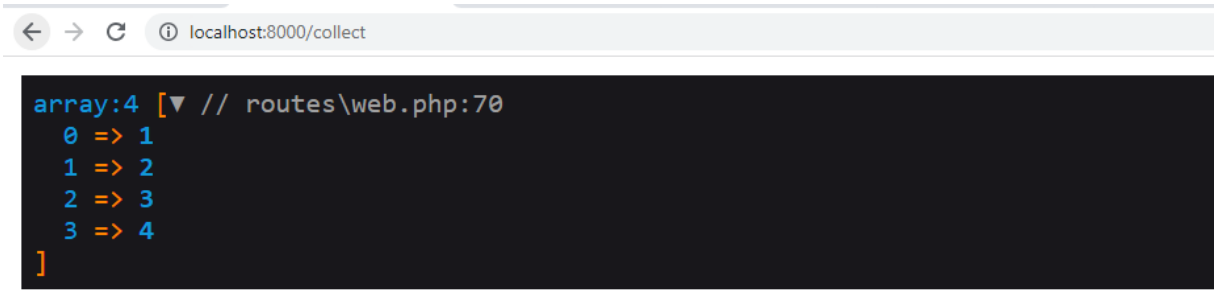
- **unique**

La méthode ***unique*** renvoie tous les éléments uniques de la collection. La collection renvoyée conserve les clés du tableau d'origine. Dans l'exemple suivant, nous utiliserons donc la méthode ***values*** pour réinitialiser les clés à des index numérotés consécutivement :

```
$collection = collect([1, 1, 2, 2, 3, 4, 2]);

$unique = $collection->unique();

$unique->values()->all();
```



- **sort**

La méthode ***sort*** permet de trier la collection. La collection triée conserve les clés du tableau d'origine. Dans l'exemple suivant, nous utiliserons donc la méthode values pour réinitialiser les clés à des index numérotés consécutivement :

```
$collection = collect([5, 3, 1, 2, 4]);

$sorted = $collection->sort();

$sorted->values()->all();

// [1, 2, 3, 4, 5]
```

- **whereBetween**

La méthode ***whereBetween*** filtre la collection en déterminant si la valeur d'un élément spécifié se situe dans une fourchette donnée :

```
$collection = collect([
    ['product' => 'Desk', 'price' => 200],
    ['product' => 'Chair', 'price' => 80],
    ['product' => 'Bookcase', 'price' => 150],
    ['product' => 'Pencil', 'price' => 30],
    ['product' => 'Door', 'price' => 100],
]);

$filtered = $collection->whereBetween('price', [100, 200]);
```

```
$filtered->all();

/*
    [
        ['product' => 'Desk', 'price' => 200],
        ['product' => 'Bookcase', 'price' => 150],
        ['product' => 'Door', 'price' => 100],
    ]
```