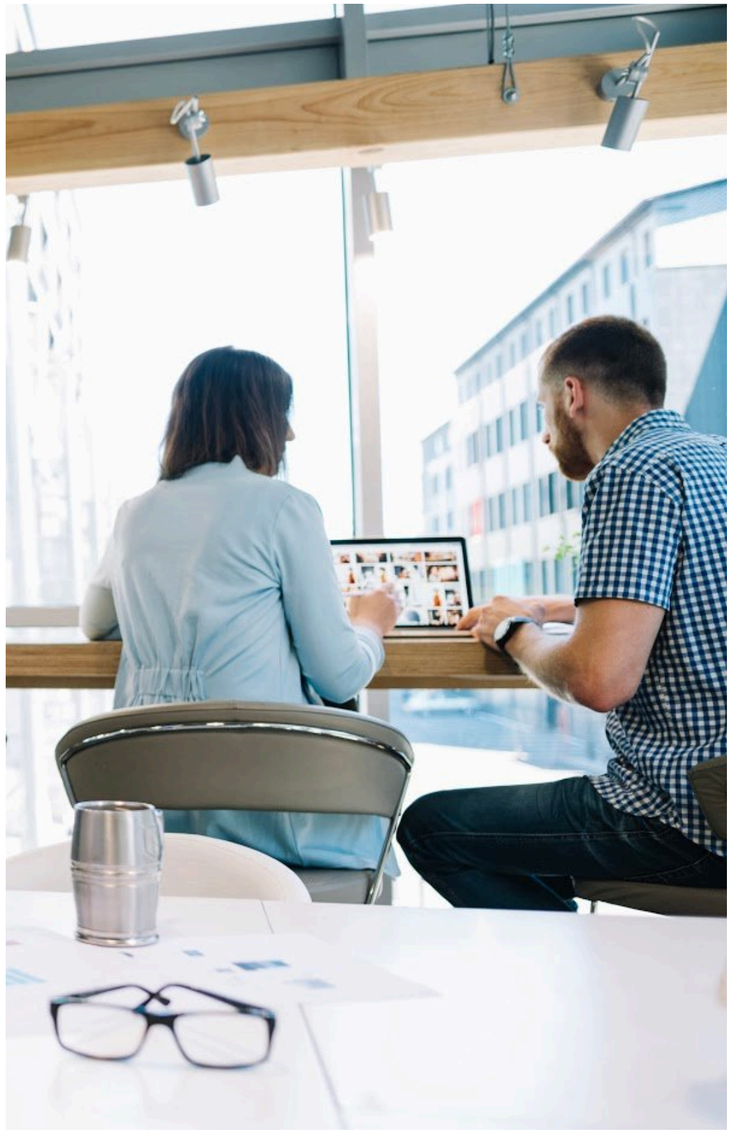

Interface and Abstraction in Laravel

What is Interface and How to use it in Laravel

What is an Interface?

An interface is like a blueprint that tells classes what methods they should have. It doesn't say how the methods should work, just that they must be there. This helps ensure that different classes that do similar things follow the same rules.

How to use it in Laravel

```php
<?php

namespace App\Interfaces;

Codeium: Refactor | Explain
interface ImplementInterface
{
    public function implement();
}
```

```php
<?php

namespace App\Services;

use App\Interfaces\ImplementInterface;

Codeium: Refactor | Explain
class FirstInterfaceService implements ImplementInterface
{
    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function implement()
    {
        return 'First Interface Service';
    }
}
```

```php
<?php

namespace App\Services;

use App\Interfaces\ImplementInterface;

Codeium: Refactor | Explain
class SecondInterfaceService implements ImplementInterface
{
    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function implement()
    {
        return 'Second Interface Service';
    }
}
```

```php
class ImplementController extends Controller
{

    protected $firstInterfaceService;
    protected $secondInterfaceService;

    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function __construct(FirstInterfaceService $firstInterfaceService, SecondInterfaceService $secondInter
    {
        $this->firstInterfaceService = $firstInterfaceService;
        $this->secondInterfaceService = $secondInterfaceService;
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function index()
    {

        $firstInterface = $this->firstInterfaceService->implement();

        $secondInterface = $this->secondInterfaceService->implement();

        dd($firstInterface, $secondInterface);
    }
}
```

Can't define function like this

```php
<?php

namespace App\Interfaces;

Codeium: Refactor | Explain
interface ImplementInterface
{
    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function implement(){

    }
}
```

Interfaces Cannot Be Instantiated on Their Own

```php
public function index()
{

    // This will cause an error because interfaces cannot be instantiated
    $notificationService = new ImplementInterface();


    $firstInterface = $this->firstInterfaceService->implement();

    $secondInterface = $this->secondInterfaceService->implement();

    dd($firstInterface, $secondInterface);
}
```

Interfaces Cannot Define Private or Protected Methods

```php
<?php

namespace App\Interfaces;

Codeium: Refactor | Explain
interface ImplementInterface
{
    private function implement();
}
```

Interfaces Cannot Define Properties

```php
<?php

namespace App\Interfaces;

Codeium: Refactor | Explain
interface ImplementInterface
{
    public $serviceUrl; // This will cause an error

    public function implement();
}
```

What Are Abstract Classes?

Abstract classes are very similar to interfaces; they're not designed to be instantiated on their own and provide a base line implementation for you to extend from.

```php
namespace App\Services;

abstract class BaseService
{
    abstract protected function process($data);

    public function execute($data)
    {
        // Common functionality
        $this->log('Executing service with data: ' . json_encode($data));

        // Call the abstract method
        return $this->process($data);
    }

    protected function log($message)
    {
        // Logging functionality
        \Log::info($message);
    }
}
```

```php
<?php

namespace App\Services;

class UserService extends BaseService
{
    protected function process($data)
    {
        // Implement user-specific processing
        return "Processed user data: " . json_encode($data);
    }
}
```

```php
<?php

namespace App\Services;

class OrderService extends BaseService
{
    protected function process($data)
    {
        // Implement order-specific processing
        return "Processed order data: " . json_encode($data);
    }
}
```

```php
<?php

namespace App\Services;

use Illuminate\Support\Facades\Log;

Codeium: Refactor | Explain
abstract class NotificationService
{

    abstract public function charge($amount);

    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function logTransaction($transactionDetails)
    {
        Log::info($transactionDetails);

        echo "Transaction Details: " . json_encode($transactionDetails);
    }
}
```

```php
<?php

namespace App\Services;

use Illuminate\Support\Facades\Log;

Codeium: Refactor | Explain
class EmailNotificationService extends NotificationService
{
    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function charge($amount)
    {
        $this->logTransaction(['amount' => $amount]);
    }
}
```

```php
class ImplementController extends Controller
{
    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function index()
    {
        $firstInterface = $this->firstInterfaceService->implement();

        $secondInterface = $this->secondInterfaceService->implement();

        echo 'First Interface: ' . $firstInterface . '<br>';
        echo 'Second Interface: ' . $secondInterface . '<br>';
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function sendNotification()
    {
        $this->emailNotificationService->charge(100);
    }
}
```

Can't Create instance for Abstract class

```php
class ImplementController extends Controller
    public function index()
        $firstInterface = $this->firstInterfaceService->implement();

        $secondInterface = $this->secondInterfaceService->implement();

        echo 'First Interface: ' . $firstInterface . '<br>';
        echo 'Second Interface: ' . $secondInterface . '<br>';
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function sendNotification()
    {
        $this->emailNotificationService->charge(100);

        // This will not work, because we cant create instance for Abstraction class
        $this->NotificationService->logTransaction(['amount' => 100]);
    }
}
```

Can use Public,private and Protected

```php
abstract class NotificationService
{
    abstract public function charge($amount);

    Codeium: Refactor | Explain | Generate Function Comment | ×
    public function logTransaction($transactionDetails)
    {
        $this->log($transactionDetails);

        echo "Transaction Details: " . json_encode($transactionDetails);
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    private function log($transactionDetails)
    {
        Log::info('private', $transactionDetails);
    }
    Codeium: Refactor | Explain | Generate Function Comment | ×
    protected function logInfo($transactionDetails)
    {
        Log::info('protected', $transactionDetails);
    }
}
```