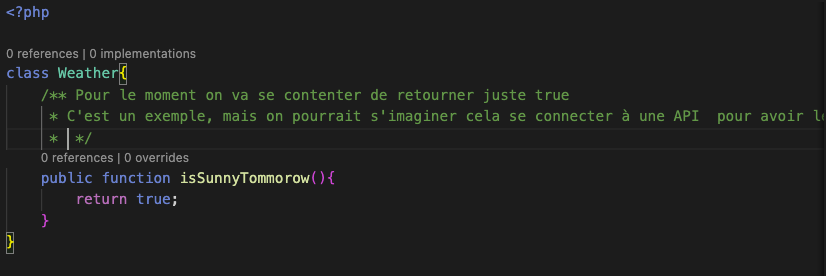
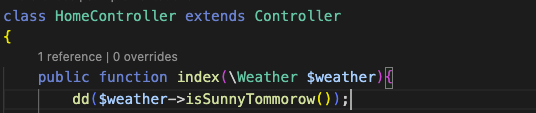
**LE SERVICE PROVIDER**

On se demande souvent comment se fait l’injection de dépendance dans **LARAVEL,** généralement quand on crée une classe et que l’on essai d’instancier un objet de cette classe partout dans notre code, **LARAVEL** a tendance à le faire sans qu’on ait à implémenter quoi que ce soit. Cela est dû au **service** **provider**.

Dans ce genre de situation, nous rencontrons un problème seulement si on a des cas où un constructeur est implémenté, d’où l’importance d’implémenter un service provider qui va gérer cela pour nous.

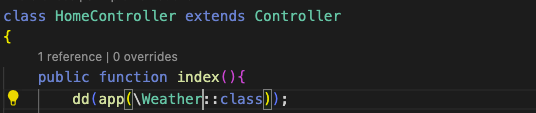
On va s’imaginer dans notre partie **App** créer une classe qui va s’appeller **Weather,** qui nous permettra simplément de savoir s’il fera beau demain ou pas.

Du coup on peut **injecter** cela dans notre **HomeController**  pour ensuite débugger cela.

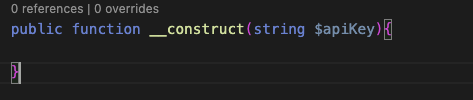


On aura accès à cela automatiquement.

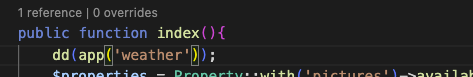
On peut également avoir accès à cela en utilisant le système de service de **Laravel**



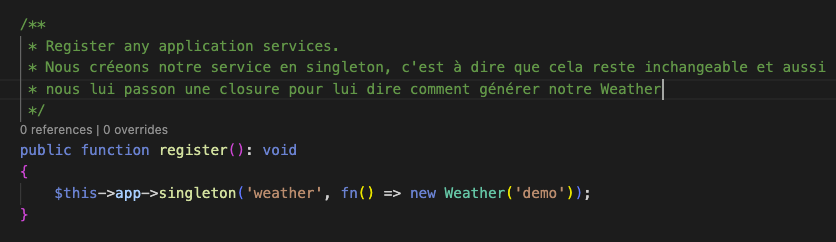
Regardons un cas concrêt par exemple dans notre **Weather** nous aurons besoin d’une clé d’API pour que cela fonctionne. Du coup nous aurons besoin d’un constructeur



Dans ce cas ci, cela nous génèrera une erreur. Du coup nous pouvons gérer l’implémentation d’un service spécifique à cela.

Dans notre exemple, on va s’imaginer vouloir un service  **‘weather’.**

Du coup dans notre **App/Provider** (qui est le provider par défaut), nous allons enregistrer cela dans la fonction registrer

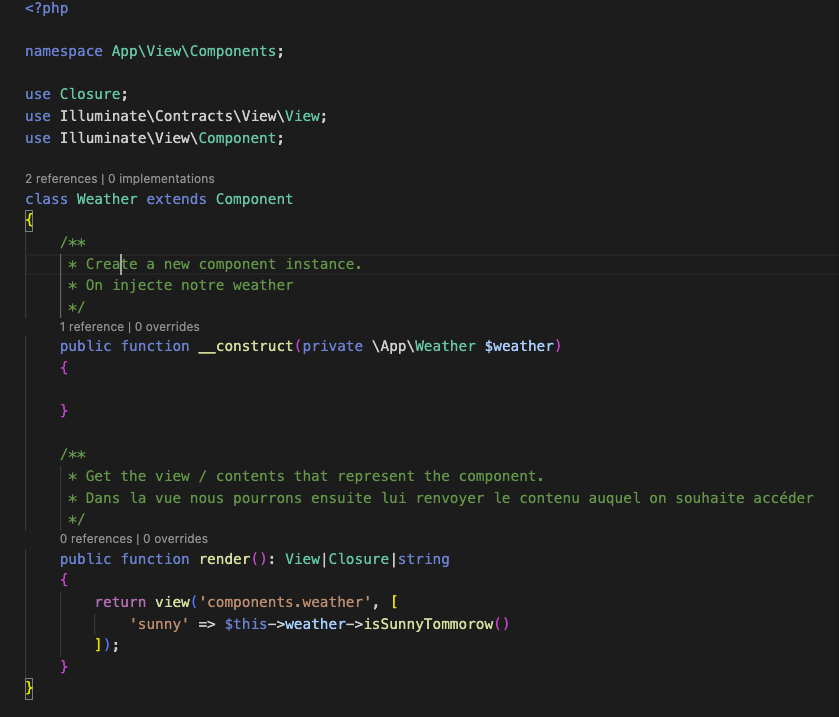


On peut également le faire en remplaçant le nom de notre chaîne de caractères par le nom de notre classe de ce fait nous pouvons faire également l’injection de dépendance dans l’index.

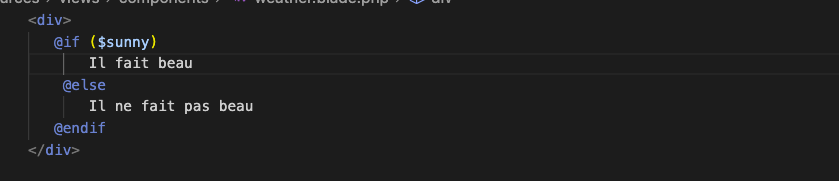
***info:*** *Ce qui est intéressant avec les services, on peut les utiliser à plein de niveaux.*

*On peut s’imaginer créer un composant pour cela*

**php artisan make:component Weather**



L’avantage est qu’on peut lui dire d'injecter le système de **weather** en construisant ce composant.

Dans notre **weather.blade.php** nous pouvons lui dire

Dans notre **Home** nous pouvons utiliser le composant **x-weather** qui nous affichera “***il fait beau “*** ; automatiquement lorsque en construisant ce composant, il va instancier **Weather.**

* Un autre cas pratique, si nous voulons récupérer les informations de **user** dans l’index, il suffit d’injecter **AuthManager** qui est la façade pour les **$users;**  En regardant plus en profondeur, on remarque que ces façades utilisent les **services** pour pouvoir nous faciliter la tâche de la sorte.

**Exemple de cas concret:**

**public function register(): void**

**{**

**$this->app->singleton('weather', function (Application $app) {**

**return new WeatherApi(config('WEATHER\_KEY'));**

**});**

**}**

Une fois ce service enregistré on peut l'utiliser n'importe où dans l'application à l'aide de la méthode globale app().

**app('weather') // WeatherApi{}**

L'accès à un service va automatiquement instancier l'objet comme on l'a défini dans le provider.

**Injection de dépendances**

Il est aussi possible de définir le service en utilisant le nom de la classe plutôt qu'une chaîne de caractère arbitraire**.**

**public function register(): void**

**{**

**$this->app->singleton(Weather::class, function (Application $app) {**

**return new WeatherApi(config('WEATHER\_KEY'));**

**});**

**}**

Cela permet ensuite de profiter de l'injection de dépendance. Lorsque Laravel va essayer d'instancier une classe il sera capable de regarder les dépendances nécessaires. Si une dépendance correspond à un service qu'il connaît, il injectera automatiquement le service en question.

**class UserController extends Controller**

**{**

**public function \_\_construct(**

**private WeatherApi $weather,**

**) {}**

**}**

Cela peut aussi être utilisé pour des composants du framework en remplacement des façades.

**class UserController extends Controller**

**{**

**public function index(**

**private Illuminate\Auth\AuthManager $auth,**

**) {**

**$user = $auth->user();**

**// ...**

**}**

**}**

**### Cas Pratiques d'Utilisation des Service Providers dans une Application de Gestion de Propriétés à Louer**

**## 1. Gestion des Images**

**### Étape 1 : Créer la classe de service**

**```php**

**// app/Services/ImageService.php**

**namespace App\Services;**

**use Illuminate\Support\Facades\Storage;**

**use Intervention\Image\Facades\Image;**

**class ImageService**

**{**

**public function upload($file, $path)**

**{**

**$filename = time() . '\_' . $file->getClientOriginalName();**

**$filePath = $path . '/' . $filename;**

**// Utilisation d'Intervention Image pour redimensionner**

**$image = Image::make($file)->resize(800, 600, function ($constraint) {**

**$constraint->aspectRatio();**

**$constraint->upsize();**

**});**

**Storage::disk('public')->put($filePath, (string) $image->encode());**

**return $filePath;**

**}**

**public function delete($path)**

**{**

**Storage::disk('public')->delete($path);**

**}**

**}**

**```**

**### Étape 2 : Créer le Service Provider**

**```php**

**// app/Providers/ImageServiceProvider.php**

**namespace App\Providers;**

**use Illuminate\Support\ServiceProvider;**

**use App\Services\ImageService;**

**class ImageServiceProvider extends ServiceProvider**

**{**

**public function register()**

**{**

**$this->app->singleton(ImageService::class, function ($app) {**

**return new ImageService();**

**});**

**}**

**public function boot()**

**{**

**// Configurations supplémentaires si nécessaire**

**}**

**}**

**```**

**### Étape 3 : Enregistrer le Service Provider dans `config/app.php`**

**```php**

**// config/app.php**

**'providers' => [**

**// Autres fournisseurs de services...**

**App\Providers\ImageServiceProvider::class,**

**],**

**```**

**### Étape 4 : Utiliser le service dans un contrôleur**

**```php**

**// app/Http/Controllers/PropertyController.php**

**namespace App\Http\Controllers;**

**use App\Services\ImageService;**

**use Illuminate\Http\Request;**

**class PropertyController extends Controller**

**{**

**protected $imageService;**

**public function \_\_construct(ImageService $imageService)**

**{**

**$this->imageService = $imageService;**

**}**

**public function uploadImage(Request $request)**

**{**

**$file = $request->file('image');**

**$path = 'properties/images';**

**$filePath = $this->imageService->upload($file, $path);**

**// Enregistrer le chemin de l'image dans la base de données**

**// ...**

**return response()->json(['message' => 'Image uploaded successfully', 'path' => $filePath]);**

**}**

**public function deleteImage($filePath)**

**{**

**$this->imageService->delete($filePath);**

**// Supprimer l'enregistrement de la base de données**

**// ...**

**return response()->json(['message' => 'Image deleted successfully']);**

**}**

**}**

**```**

**## 2. Service de Notification**

**### Étape 1 : Créer la classe de service**

**```php**

**// app/Services/NotificationService.php**

**namespace App\Services;**

**use Illuminate\Support\Facades\Mail;**

**class NotificationService**

**{**

**public function sendEmail($to, $subject, $message)**

**{**

**Mail::raw($message, function ($mail) use ($to, $subject) {**

**$mail->to($to)**

**->subject($subject);**

**});**

**}**

**// Vous pouvez ajouter d'autres méthodes pour les notifications SMS, etc.**

**}**

**```**

**### Étape 2 : Créer le Service Provider**

**```php**

**// app/Providers/NotificationServiceProvider.php**

**namespace App\Providers;**

**use Illuminate\Support\ServiceProvider;**

**use App\Services\NotificationService;**

**class NotificationServiceProvider extends ServiceProvider**

**{**

**public function register()**

**{**

**$this->app->singleton(NotificationService::class, function ($app) {**

**return new NotificationService();**

**});**

**}**

**public function boot()**

**{**

**// Configurations supplémentaires si nécessaire**

**}**

**}**

**```**

**### Étape 3 : Enregistrer le Service Provider dans `config/app.php`**

**```php**

**// config/app.php**

**'providers' => [**

**// Autres fournisseurs de services...**

**App\Providers\NotificationServiceProvider::class,**

**],**

**```**

**### Étape 4 : Utiliser le service dans un contrôleur**

**```php**

**// app/Http/Controllers/PropertyController.php**

**namespace App\Http\Controllers;**

**use App\Services\NotificationService;**

**use Illuminate\Http\Request;**

**class PropertyController extends Controller**

**{**

**protected $notificationService;**

**public function \_\_construct(NotificationService $notificationService)**

**{**

**$this->notificationService = $notificationService;**

**}**

**public function store(Request $request)**

**{**

**// Logique pour ajouter une nouvelle propriété**

**// ...**

**$this->notificationService->sendEmail('user@example.com', 'New Property Added', 'A new property has been added.');**

**return response()->json(['message' => 'Property added and notification sent.']);**

**}**

**}**

**```**

**## 3. Service de Géolocalisation**

**### Étape 1 : Créer la classe de service**

**```php**

**// app/Services/GeolocationService.php**

**namespace App\Services;**

**use GuzzleHttp\Client;**

**class GeolocationService**

**{**

**protected $client;**

**public function \_\_construct()**

**{**

**$this->client = new Client();**

**}**

**public function getCoordinates($address)**

**{**

**// Utilisez une API de géolocalisation comme Google Maps ou OpenStreetMap**

**$response = $this->client->get('https://maps.googleapis.com/maps/api/geocode/json', [**

**'query' => [**

**'address' => $address,**

**'key' => env('GOOGLE\_MAPS\_API\_KEY'),**

**],**

**]);**

**$data = json\_decode($response->getBody(), true);**

**if (isset($data['results'][0])) {**

**return $data['results'][0]['geometry']['location'];**

**}**

**return null;**

**}**

**}**

**```**

**### Étape 2 : Créer le Service Provider**

**```php**

**// app/Providers/GeolocationServiceProvider.php**

**namespace App\Providers;**

**use Illuminate\Support\ServiceProvider;**

**use App\Services\GeolocationService;**

**class GeolocationServiceProvider extends ServiceProvider**

**{**

**public function register()**

**{**

**$this->app->singleton(GeolocationService::class, function ($app) {**

**return new GeolocationService();**

**});**

**}**

**public function boot()**

**{**

**// Configurations supplémentaires si nécessaire**

**}**

**}**

**```**

**### Étape 3 : Enregistrer le Service Provider dans `config/app.php`**

**```php**

**// config/app.php**

**'providers' => [**

**// Autres fournisseurs de services...**

**App\Providers\GeolocationServiceProvider::class,**

**],**

**```**

**### Étape 4 : Utiliser le service dans un contrôleur**

**```php**

**// app/Http/Controllers/PropertyController.php**

**namespace App\Http\Controllers;**

**use App\Services\GeolocationService;**

**use Illuminate\Http\Request;**

**class PropertyController extends Controller**

**{**

**protected $geolocationService;**

**public function \_\_construct(GeolocationService $geolocationService)**

**{**

**$this->geolocationService = $geolocationService;**

**}**

**public function store(Request $request)**

**{**

**$address = $request->input('address');**

**$coordinates = $this->geolocationService->getCoordinates($address);**

**// Logique pour ajouter une nouvelle propriété avec les coordonnées GPS**

**// ...**

**return response()->json(['message' => 'Property added', 'coordinates' => $coordinates]);**

**}**

**}**

**```**