Laravel

m2iformation.fr

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Laravel

Sommaire

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction

Histoire

- Laravel est un framework web open-source écrit en PHP respectant le principe modèle-vue-contrôleur et entièrement développé en programmation orientée objet
- Laravel a été créé par Taylor Otwell en juin 2011
- Laravel est en grande partie basé sur des composants de Symfony (30%)

Pré-requis

- PHP >= 8.1
- Composer
- VsCode ou PHPStorm ❤️

Ne pas oublier d'ajouter php aux variables d'environnement.

ex: `%LOCALAPPDATA%\php`

Il est également nécessaire d'activer certaines extensions de PHP.

Installation

- `composer create-project laravel/laravel <nom-projet>`
- `cd <nom-projet>`
- `php artisan serve`
- http://localhost:8000/

Répertoires de l'application

```
app             : répertoire de travail de l'app
config          : fichiers de config de l'app
database        : configuration de la BDD et de l'ORM
public          : point d'entrée de l'app, fichiers statiques
resources       : ressources
routes          : chemins de l'app
storage         : vue (blade), css, sass, fichiers non compilés
tests           : tests de l'app
vendor          : dépendances php
```

Cycle de vie

- Le point d'entrée de l'application est le fichier `public/index.php`
- Ce fichier utilise l'autoloader de **composer** et créé une instance de Laravel située dans `bootstrap/app.php`
- Le module `app/Http/Kernel.php` gère les requêtes, les erreurs, les logs, les variables d'environnements ainsi que les middlewares. Elle s'occupe de lire les requêtes et renvoyer des réponses HTTP
- Les services providers permettent la gestion de la BDD, la validation de données, le méchanisme de routes etc. Tous les services sont enregistrés dans `config/app.php`

Service Providers

- Les services providers sont le coeur du fonctionnement de Laravel. L'instance de l'application est créé, les services y sont enregistrés et l'application peut ensuite recevoir les requêtes.
- Les services par défaut de l'application sont stockés sous le répertoire `app/Providers`
- Il est possible de créer ses propres services pour étendre le comportement de l'application

Console Artisan

- Artisan l'interface de commandes inclue avec Laravel qui fournit bon nombres d'outils pour gérer une application
- Pour afficher l'ensemble des commandes: `php artisan list`
- Pour accéder à la documentation d'une commande en particulier: `php artisan help <command>`
- Pour lister les classes qui peuvent être générés avec Artisan: `php artisan list make`

Configuration de l'environnement

- Laravel créé 2 fichiers lors de la création de projet: `.env` et `.env.example`
- `.env.example` contient les différents paramètres à utiliser pour l'application: la base de données, le mode débug ...
- On pourra créer différents fichiers en fonction des environnements que l'on va créer (local, test, prod)

Frontend avec Laravel

Le framework Laravel nous permet de choisir 3 façons de gérer le frontend:

1. **Blade**: Blade est le moteur de templates intégré à Laravel pour créer des vues (page HTML)
2. **LiveWire**: permet de construire des interfaces réactives et dynamiques en utilisant Blade et un peu de JavaScript
3. **React/Vue**: framework front

Les routes

Route simple

Dans Laravel, les routes les plus simples prennent en paramètre un URI ainsi qu'une callback

```php
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Fichier de routes

- Toutes les routes Laravel sont stockées dans le répertoire `routes`
- Les routes de l'application web ston stockées sous `routes/web.php`
- Ces routes sont assignées au middleware web, qui gère la session et la protection CSRF
- Pour gérer des routes **stateless** pour les API, il faut initialiser le routing spécifique via la commande `php artisan install:api`

Méthode de Router

Le router fournit un certain nombre de méthode correspondant aux verbes HTTP:

```php
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
Route::match(['get', 'post'], $uri, $callback);
Route::any($uri, $callback);
```

Injection de dépendances

- Laravel inclut un mécanisme d'injection de dépendances pourquoi récupérer la requête HTTP correspondant à l'URI consulté
- L'injection se fait directement au niveau de la callback à l'aide d'un object de type `Request`

```php
use Illuminate\Http\Request;

Route::get('/users', function (Request $request) {
    // ...
});
```

Redirections

- Pour paramétrer des redirections sur d'autres routes, le Router nous fournit certains méthodes raccourcis pour rediriger vers la route souhaitée
- Par défaut, la redirection se fait avec un code 301, il est possible de customiser le code de redirection

```
Route::redirect('/here', '/there');
Route::redirect('/here', '/there', 301);
```

Diriger uniquement vers une route

- Dans le cas où une route ne définit aucune logique métier, le router nous donne la possibilité de retourner directement une vue
- Pour cela il suffit d'utiliser la méthode statique `view()` :

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

Lister les routes

- Artisan nous fournit un certain nombre de commandes pour connaîtres les routes présentent sur notre application
- `php artisan route:list` : affiche toutes les routes
- `php artisan route:list -v` : affiche les routes ainsi que les middlewares associés

Paramètres de routes

- Il est possible de passer des paramètres aux routes qui pourront être récupérés pour faire des traitements particuliers
- Pour cela il suffit d'ajouter les les brackets `{param}` dans la route avec le nom du paramètre
- Dans la callback (ou le controller), on ajoute un paramètre avec le même nom que celui définit dans la route

```
Route::get('/posts/{post}/comments/{comment}', function (string $postId, string $commentId) {
    return 'postId '.$postId.' commentId'.$commentId;
});
```

Paramètres optionnels

- Il est parfois intéressant d'ajouter des paramètres optionnels avec des valeurs par défaut
- Pour cela on peut utiliser le charactère `?` à la fin d'un paramètre de route
- Dans la callback, on ajoutera le paramètre comme nullable

```php
Route::get('/user/{name?}', function (?string $name = null) {
    return $name;
});
```

Regex dans les paramètres

- Pour s'assurer de la validité des paramètres dans l'URI, il est possible d'utiliser des règles regex
- Pour cela on utilise la méthode `where()` qui prend en paramètre le nom du paramètre ainsi que la règle

```
Route::get('/user/{name}', function (string $name) {
    // ...
})->where('name', '[A-Za-z]+');
```

Méthode utilitaire regex

Laravel nous fournit plusieurs méthodes utilitaires pour la gestion des paramètres d'URL :

- `whereAlpha` : uniquement des caractères de l'alphabet
- `whereNumber` : uniquement des nombres
- `whereAlphaNumeric` : caractères et nombres
- `whereUuid` : format uuid
- `whereIn` : tableau d'entrées valides

Nommer les routes

- Le nommage des routes nous permet d'effectuer une redirection de manière plus efficace en utilisant le nom de la route
- Dans la quasi totalité des cas, nous nommerons les routes pour pouvoir y faire référence dans d'autres parties du code
- ⚠️ les noms doivent être unique

```
Route::get('/user/profile', function () {
    // ...
})->name('profile');
```

Groupement de routes

- Lorsque l'on doit maintenir un grand nombre de routes, il peut être intéressant de les regrouper pour y inclure des comportements communs
- Les regroupements s'appliquent aux `middlewares`, à la gestion de `prefix` d'URI et de noms

Groupement de middlewares

- Pour appliquer un plusieurs middlewares à plusieurs routes, il suffit d'utiliser la méthode:

```
Route::middleware(['first', 'second'])->group(function () {
    // Définition des routes
});
```

- Les middlewares s’exécutent dans l'ordre dans lequel ils ont été défini dans le tableau

Groupement sur un Controller

- Dans le cas où plusieurs routes utilisent le même Controller, il peut être intéressant de créer une route avec un groupement en lien avec ce Controller
- Le premier paramètre sera la route, le second la méthode du controller qui gère cette route

```
Route::controller(OrderController::class)->group(function () {
    Route::get('/orders/{id}', 'show');
    Route::post('/orders', 'store');
});
```

Préfixe de routes

- Les préfixes de routes permettent d'ajouter un groupement de routes qui ont le même identificateur racine commun
- Exemples: `/products/{id}`, `/products/{id}/comments`, `/products/create`

```
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    });
});
```

Préfixe de nom

- Il est possible de préfixer le nom des routes
- La bonne pratique réside généralement à ajouter le nom de la route même avec le caractère `.` en suffixe
- Ainsi pour les redirections on pourra utiliser la syntaxe suivante: `Redirect::to('admin.users');`

```php
Route::name('admin.')->group(function () {
    Route::get('/users', function () {
        // Route assigned name "admin.users"...
    })->name('users');
});
```

Databinding de modèle

- Laravel résout automatiquement les modèles Eloquent définit dans les paramètres de routes
- Dans le cas où le modèle n'est pas trouvé, Laravel renvoie une erreur 404

```
Route::get('/users/{user}', function (User $user) {
    return $user->email;
});
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Middleware

Définition

Un middleware dans une application Laravel est un mécanisme qui permet de filtrer les requêtes HTTP entrant dans l'application.

- Le filtrage permet d'effectuer une tâche avant ou après que la requête soit traitée
- Les middleware peuvent être utilisés pour des tâches telles que l'authentification, la protection CSRF, la journalisation des requêtes entrantes
- Les middleware personnalisés sont généralement situés dans le répertoire `app/Http/Middleware`

Créer un middleware

- Artisan nous fournit une commande pour générer un middleware:

```
php artisan make:middleware EnsureTokenIsValid
```

- On peut créer une méthode nommée `handle` pour s'occuper de la logique du middleware

```
public function handle(Request $request, Closure $next): Response
{
    if ($request->input('token') !== 'my-secret-token') {
        return redirect('home');
    }
    return $next($request);
}
```

Enregistrer un middleware

- Pour enregistrer un middleware sur toutes les requêtes de l'application, il faut éditer le fichier `bootstrap/app.php`

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->append(EnsureTokenIsValid::class);
})
```

- Enregistrer un middleware sur une route :

```
Route::get('/profile', function () {
})->middleware(EnsureTokenIsValid::class);
```

Groupement de middlewares

- Il est parfois intéressant d'enregistrer un même groupe de middlewares sur plusieurs routes
- Pour cela il faut éditer `bootstrap/app.php`

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->appendToGroup('group-name', [
        First::class,
        Second::class,
    ]);
});
```

Groupement de middlewares

- Pour utiliser le groupement de middlewares il suffit ensuite de l'appeler par son nom dans la méthode associée :

```php
Route::get('/', function () {
    // ...
})->middleware('group-name');

Route::middleware(['group-name'])->group(function () {
    // ...
});
```

Middlewares par défaut

Laravel comprend des groupes prédéfinis de middleware web et api qui contiennent des middleware communs à ces deux routes

- `Illuminate\Cookie\Middleware\EncryptCookies`
- `Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse`
- `Illuminate\Session\Middleware\StartSession`
- `Illuminate\View\Middleware\ShareErrorsFromSession`
- `Illuminate\Foundation\Http\Middleware\ValidateCsrfToken`
- `Illuminate\Routing\Middleware\SubstituteBindings`

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les Controllers

Introduction

- Plutôt que de définir toutes la logique de gestion des requêtes directement dans les routes, il est souvent intéressant de créer une couche dédié à cette tâche qu'on appelle `Controller`
- Un controller gère les requêtes lié à la logique métier d'un composant de notre application
- Par exemple un `UserController` se chargera des étapes de création, d'affichage, de modification ou de suppression d'un utilisateur

Création d'un controller

Pour rapidement créer un controller dans Laravel on utilise la commande: `php artisan make:controller <NomDuController>`

Chaque méthode d'un controller s'occupe de gérer la logique d'une requête HTTP.

```php
public function show(string $id): View
{
    return view('user.profile', [
        'user' => User::findOrFail($id)
    ]);
}
```

Lier une route à un Controller

- Une fois le Controller défini, il faut lier une route à l'une de ces méthodes
- Pour cela, dans notre web.php, on ajoute la route suivi d'un tableau avec la classe ainsi que la méthode

```
use App\Http\Controllers\UserController;

Route::get('/user/{id}', [UserController::class, 'show']);
```

Ressource Controller

- Pour créer des actions de CRUD simples depuis un modèle Eloquent, il est possible d'utiliser le paramètre `--resource` lors de la création d'un Controller

```
php artisan make:controller PhotoController --resource
```

- Il faudra ensuite ajouter la route :

```
Route::resource('photos', PhotoController::class);
```

Routes générées

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Routes partielles

Lorsque l'on créé un Resource Controller, on a la possibilité de limiter les actions que le Controller peut faire.
Il existe pour cela deux techniques:

- `only()` : on précise les actions que l'on souhaite
- `except()` : on précise les actions qui seront omises

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Requête HTTP

Accéder à la requête

- A l'aide de l'injection de dépendance de Laravel, la requête est automatiquement injectée dans les méthodes du Controller
- Les autres paramètres dont la route a besoin doivent toujours être placés après les éléments injectés

```
public function update(Request $request, string $id): RedirectResponse
{
    return redirect('/users');
}
```

Récupérer les entrées

L'objet Request nous permet de récupérer les informations d'un formulaire ou de toute autre requête HTTP

- `$input = $request->all();` : renvoie un tableau de clés-valeurs
- `$input = $request->collect();` : renvoie un objet de type collection
- `$name = $request->input('name');` : récupère la valeur d'un élément
- `$name = $request->input('name', 'default');` : permet d'avoir une valeur par défaut

Méthode utiles

- `$request->path()` : connaître l'URI de la requête
- `$request->method()` et `$request->isMethod()` : verbe HTTP utilisé
- `$request->is('admin/*')` : vérification de la route

Récupérer les query parameters

- `$name = $request->query('name');` : récupère le paramètre `?name=toto`
- `$name = $request->query('name', 'Helen');` : paramètre par défaut

Récupération d'entrées

- Laravel permet de récupérer les champs d'un formulaire de la requête précédente
- Cette fonctionnalité est utilisée automatiquement lors de l'utilisation de la validation des champs de formulaires
- La méthode `flash()` permet de stocker les informations dans la session

```
$request->flash();
```

Flasher et rediriger

Pour faciliter le transfer des données à la session puis rediriger vers la page précédente, il est possible d'enchaîner la transmission de données à une redirection à l'aide de la méthode `withInput` :

```
return redirect('form')->withInput();

return redirect()->route('user.create')->withInput();

return redirect('form')->withInput(
    $request->except('password')
);
```

Récupérer les anciennes données

La méthode `old()` permet de récupérer un élément précédemment flashé :

```
$username = $request->old('username');
```

Une fonction old existe également dans Blade pour récupérer la valeur précédente

```
<input type="text" name="username" value="{{ old('username') }}">
```

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Réponses HTTP

Réponse HTTP

- Toutes les routes et tous les contrôleurs doivent renvoyer une réponse au navigateur de l'utilisateur.
- Laravel propose plusieurs façons de renvoyer des réponses. La réponse la plus basique consiste à renvoyer une chaîne de caractères à partir d'une route ou d'un contrôleur.
- Le framework convertira automatiquement la chaîne en une réponse HTTP complète:

```
Route::get('/', function () {
    return 'Hello World';
});
```

Retourner un tableau

- Par défaut, lorsque l'on renvoie un tableau PHP, celui-ci est converti sous forme de JSON
- Cela fonctionne également avec les collections de Eloquent

```
Route::get('/', function () {
    return [1, 2, 3];
});
```

Retourner un objet

- Dans la majorité des cas, une route ou un controller retourner une instance de `Illuminate\Http\Response` ou une vue générée de Blade
- Retourner un objet Response permet d'avoir un plus grand contrôle sur la réponse HTTP renvoyée

```
Route::get('/home', function () {
    return response('Hello World', 200)
                  ->header('Content-Type', 'text/plain');
});
```

Entête de réponse

- Il est possible de chaîner les entêtes de réponses HTTP avec la méthode `withHeaders` plutôt que de rappeler `header` plusieurs fois

```
return response($content)
            ->withHeaders([
                'Content-Type' => $type,
                'X-Header-One' => 'Header Value',
                'X-Header-Two' => 'Header Value',
            ]);
```

Redirection

- Les redirections sont des instances de `Illuminate\Http\RedirectResponse` qui contiennent les entêtes correctes pour rediriger vers le bon URL
- La manière la plus simple de créer une redirection est d'utiliser la fonction `redirect`:

```
Route::get('/dashboard', function () {
    return redirect('home/dashboard');
});
```

Redirection formulaire invalide

- Généralement, il est intéressant de pouvoir rediriger une action sur l'URL précédent lors d'un envoie d'un formulaire invalide
- La fonction `back` utilise la session pour renvoyer sur le bon URL

```
Route::post('/user/profile', function () {
    // Validate the request...

    return back()->withInput();
});
```

Redirection d'URL nommé

- Il est possible de rediriger vers une URL à partir du nom que l'on a défini dans la route
- Pour cela, on appelle la méthode `route` :

```
return redirect()->route('login');
return redirect()->route('profile', ['id' => 1]);
return redirect()->route('profile', [$user]);
```

Redirection sur une action de Controller

- Redirect fournit une méthode `action` qui permet de rediriger sur une méthode d'un Controller
- Il est également possible de lui fournir un tableau associatif avec les paramètres de la route

```php
use App\Http\Controllers\UserController;

return redirect()->action(
    [UserController::class, 'profile'], ['id' => 1]
);
```

Redirection avec flash de données

- La redirection vers une nouvelle URL et l'envoi de données à la session se font généralement en même temps
- Généralement, cela se fait après avoir effectué une action avec succès, lorsque vous envoyez un message de réussite à la session

```
Route::post('/user/profile', function () {
    return redirect('dashboard')->with('status', 'Profile updated!');
});
```

```
@if (session('status'))
    <div class="alert alert-success"> {{ session('status') }} </div>
@endif
```

Réponse JSON et Fichiers

- Response fournit la méthode `json` pour transformer un tableau associatif en objet JSON en ajoutant le bon header

```php
return response()->json([
    'name' => 'Abigail',
    'state' => 'CA',
]);
```

- Il est également possible de forcer le téléchargement de fichiers avec la méthode `download`:

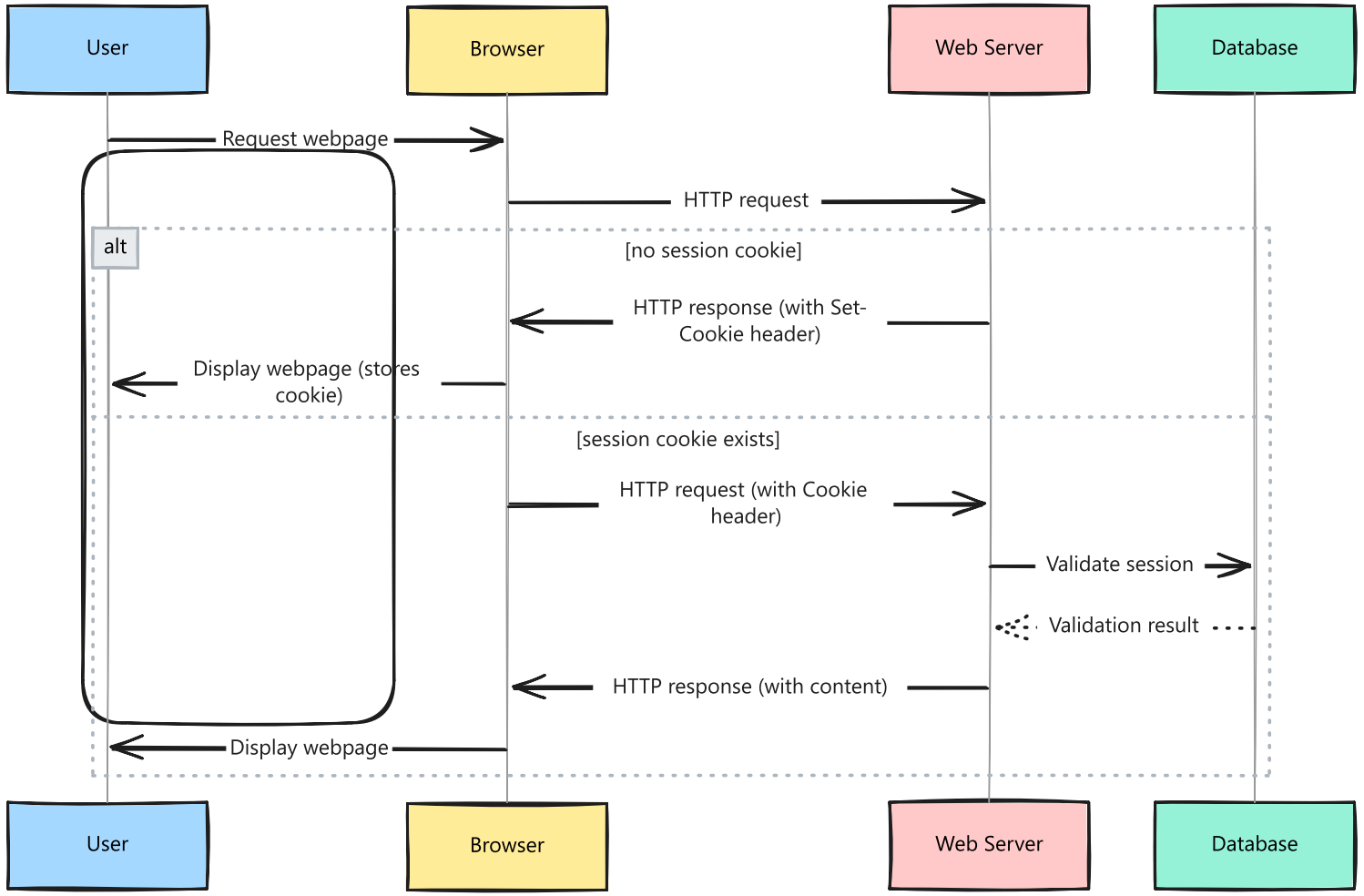
```php
return response()->download($pathToFile);
```

Session

Introduction aux sessions

- Étant donné que les applications pilotées par HTTP sont sans état (stateless), les sessions permettent de stocker des informations sur l'utilisateur à travers plusieurs requêtes.
- Ces informations sur l'utilisateur sont généralement placées dans une base de données persistante à laquelle il est possible d'accéder lors de requêtes ultérieures.

Mécanisme de session



Configuration

- Le fichier de configuration de la session de l'application est stocké dans `config/session.php`
- Par défaut, Laravel est configuré pour utiliser le pilote de session de la base de données
- Il est possible d'utiliser d'autres drivers: fichier, cookie, redis, tableau ...

Session en base de données

- Pour obtenir la table de gestion de sessions en base de données, il faut appliquer la migration initiale nommée `0001_01_01_000000_create_users_table.php`
- Si pour une raison particulière cette migration n'existe pas, il faudra taper les commandes suivantes:

```
php artisan make:session-table
php artisan migrate
```

Accéder aux variables de session

Il y a deux façons principales de travailler avec les données de session dans Laravel :

- la fonction globale `session`
- via une instance de `Request`

Accéder aux variables de session

- Lorsque la fonction globale `session` est appelée avec un seul argument de type chaîne, elle renvoie la valeur de cette clé de session
- Lorsque l'aide est appelée avec un tableau de paires clé/valeur, ces valeurs seront stockées dans la session

```php
// Lit les valeurs depuis la session
$value = session('key');
$value = session('key', 'default');

// Stocke la valeur en session
session(['key' => 'value']);
```

Accès aux variables depuis la requête

- La requête injectée dans les méthodes des controllers nous permettent d'avoir accès aux variables de session
- La méthode `get` prend en paramètre la clé, il est également possible de lui passer une valeur par défaut

```
$value = $request->session()->get('key');
$value = $request->session()->get('key', 'default');
```

Méthodes de session

Quelques méthodes utiles à utiliser avec un objet Session:

```php
// Retourne true si la clé existe et est non nulle
$request->session()->has('users');
// Retourne vraie si la clé existe
$request->session()->exists('users');
// Stocke la valeur en session
$request->session()->put('key', 'value');
// Ajoute un élément en fin de tableau
$request->session()->push('user.teams', 'developers');
// Lit et supprime l'item
$value = $request->session()->pull('key', 'default');
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Template - Blade

Présentation

- Blade est le moteur de template inclus avec Laravel
- Il est possible d'utiliser du code PHP directement dans les templates, les pages blades sont d'ailleurs compilés en PHP et mises en cache jusqu'à leur modification
- En général les pages blades ont l'extension `.blade.php` et sont stockées dans `ressources/views`

Renvoyer une vue

- Le controller renvoie une vue à l'aide de la fonction `view()`
- Celle-ci prend en paramètre le nom de la vue suivi d'un tableau associatif de clés/valeurs

```
Route::get('/', function () {
    return view('greeting', ['name' => 'Finn']);
});
```

- Dans la vue on peut récupérer les données à l'aide de la clé:

```
Hello, {{ $name }}.
```

Générer des vues dynamiques

- Les instructions d'écho `{{ }}` de Blade sont automatiquement envoyées via la fonction `htmlspecialchars` de PHP afin de prévenir les attaques XSS
- Pour ne pas échapper les caractères: `{!! $name !!}`
- Il est possible de passer n'importe quel code PHP à l'intérieur de ces doubles brackets `{{ time() }}`

Directive if

- Blade fournit des directives pour les structures conditionnelles: `@if`, `@elseif`, `@else`, `@endif`

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

Directive unless

La directive unless permet de faire de l'affichage conditionnel uniquement si la valeur vaut `true`

```
@unless (Auth::check())
    You are not signed in.
@endunless
```

Directive isset et empty

- `@isset` vérifie que la variable passée n'est pas *nulle*

```
@isset($records)
    // $records is defined and is not null...
@endisset
```

- `@empty` vérifie que la valeur n'est pas considérée *falsy*

```
@empty($records)
    // $records is "empty"...
@endempty
```

Directive d'authentication

- `@auth` et `@guest` permettent d'afficher du contenu si l'utilisateur est authentifié
- Cela peut être utile pour afficher des points de navigation dans une navbar par exemple

```
@auth
    // The user is authenticated...
@endauth

@guest
    // The user is not authenticated...
@endguest
```

Directive d'environnement

- L'affichage peut aussi se faire selon l'environnement dans lequel l'application est en train de tourner

```
@production
    // Production specific content...
@endproduction

@env('staging')
    // The application is running in "staging"...
@endenv

@env(['staging', 'production'])
    // The application is running in "staging" or "production"...
@endenv
```

Directive de session

- La directive de session permet de savoir si une valeur existe dans celle-ci
- Si la valeur existe on peut l'utiliser avec la variable `$value`

```
@session('status')
    <div class="p-4 bg-green-100">
        {{ $value }}
    </div>
@endsession
```

Directive switch

- Les instructions de switch peuvent être construites à l'aide des directives `@switch`, `@case`, `@break`, `@default` et `@endswitch` :

```
@switch($i)
    @case(1)
        First case...
        @break
    @default
        Default case...
@endswitch
```

Directives structures conditionnelles

- Blade fourni plusieurs directives pour utiliser les boucles

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach
```

```
@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

La variable spéciale $loop

- A l'intérieur d'une boucle `foreach`, la variable `$loop` nous permet d'avoir des informations sur l'index en cours
- `$loop->first`: renvoie vrai si c'est le premier item
- `$loop->last`: renvoie vrai si c'est le dernier item
- `$loop->count`: nombre total d'éléments itérés

Consulter la documentation

Style et classes conditionnelles

- la directive `@class` permet de compiler des classes CSS depuis un tableau
- Les clés sont le nom des classes et leur valeur défini leur présence

```
<span @class([
    'p-4',
    'font-bold' => $isActive,
    'text-gray-500' => ! $isActive,
    'bg-red' => $hasError,
])></span>
```

Directives additionnelles

- `@checked` : gérer si la checkbox est cochée

```
<input type="checkbox" name="active" value="active" @checked(old('active', $user->active)) />
```

- `@selected` : sélectionner un item de la liste

```
<option value="{{ $version }}" @selected(old('version') == $version)>
```

- `@disabled` : désactiver un champ

```
<button type="submit" @disabled($errors->isNotEmpty())>Submit</button>
```

Importer des vues

- Blade inclut la directive ``@include` pour inclure une vue dans une autre vue
- Toutes les variables présentent dans le parent le seront dans l'enfant

```
<div>
    @include('shared.errors')

    <form>
        <!-- Form Contents -->
    </form>
</div>
```

Importer des vues

- Il est possible de passer des données à la vue importée

```
@include('view.name', ['status' => 'complete'])
```

- `@includeIf` permet d'importer une vue uniquement si elle existe

```
@includeIf('view.name', ['status' => 'complete'])
```

- `@includeWhen` importe une vue en fonction d'une valeur booléenne

```
@includeWhen($boolean, 'view.name', ['status' => 'complete'])
```

Informations complémentaires

Pour utiliser du php vanilla:

```
@php
    $counter = 1;
@endphp
```

Pour importer une classe, la directive `@use` est suffisante:

```
@use('App\Models\Flight')
```

Les commentaires se font à l'aide de la directive suivante:

```
{{-- Commentaire --}}
```

Sécurisation des formulaires

- Pour tous les formulaires, il est important d'inclure un token CSRF pour se protéger des failles de sécurité
- Laravel inclut ce mécanisme dans les templates avec la directive `@csrf` ainsi qu'un middleware dédié

```
<form method="POST" action="/profile">
    @csrf
</form>
```

Pour en savoir plus

Validation de données

Présentation

- Laravel fournit un mécanisme de validation de données
- Dans la majorité des cas, la validation se fera à l'aide de la méthode `validate` présente dans la requête HTTP
- Laravel inclut de base un grand nombre de validation de données, il est également possible de créer ses propres mécanismes de vérifications

La méthode validate

- La validation se fait généralement avec la méthode `validate` depuis la requête
- Celle-ci prend en paramètre un tableau avec en clé le champ, et en valeur les règles de validation séparée par le caractère `|`

```
$validated = $request->validate([
    'title' => 'required|unique:posts|max:255',
    'body' => 'required',
]);
```

Pour connaître les règles de validation

Afficher les erreurs

- Lorsqu'un formulaire est invalide, Laravel renvoie automatiquement à la page précédente en flashant les messages d'erreurs
- La variable `$errors` est fournie par un middleware qui donne accès aux erreurs stockées

```
<ul>
    @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
</ul>
```

Gestion des données valides

- Les données validées par la méthode `validate` renvoient un tableau associatif

```
$validated = $request->validated();
```

- Il est également possible de récupérer les données sous la forme d'une collection depuis la méthode `safe`

```
$collection = $request->safe()->collect();
```

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Base de données

Introduction

- La plupart des applications web nécessitent un stockage de données, pour cette raison, Laravel fournit différentes connecteurs pour accéder à des base de données
- Laravel met à disposition plusieurs composants pour simplifier le requêtage de données, notamment un Query Builder ainsi qu'un ORM
- Les bases de données compatibles sont: MySQL/MariaDB, PostgreSQL, SQLite, SQL Server

Configuration

- La configuration des différentes base de données si situe dans `config/database.php`
- Pour configurer les informations de la base de données, il suffit d'éditer la le fichier `.env` à la racine du projet
- Pour commencer à configurer l'application, il suffit de copier les informations depuis le fichier `.env.example`

Effectuer une requête SQL

- Laravel fournit une façade nommée `DB`, celle-ci nous permet d'effectuer l'ensemble des opérations SQL possible
- Les différentes méthodes sont: `select`, `update`, `insert`, `delete`, `statement`
- Les requêtes effectuées peuvent être préparées

```
$users = DB::select('select * from users where active = ?', [1]);
```

Définition d'une migration

- Les migrations sont comme un **gestionnaire de version** de la base de données
- Généralement lorsque l'on créé un schéma de base de données, il est nécessaire de le partager et de le mettre à jour manuellement entre toutes les parties prenantes
- Une migration permet d'inclure un schéma sous forme de code qui sera ensuite mis à jour à l'aide d'une CLI

Générer une migration

- Artisan fournit une commande pour générer une migration: `make:migration`
- Les migrations générées sont stockées dans le répertoire `database/migrations`
- Chaque migration est générée avec un timestamp pour savoir dans quel ordre les exécuter

```
php artisan make:migration create_flights_table
```

Optimisation de migrations

- Lorsqu'une l'application commence à avoir un certains nombres de migrations, il peut être intéressant d'optimiser le processus de génération du schéma de la base de données
- La commande `php artisan schema:dump` va créer un fichier sql dans le répertoire `database/schema` qui sera exécuté avant d'effectuer les nouvelles migrations

Structure de migration

- Une classe de migration contient deux méthodes: `up` et `down`
- `up` permet de créer des tables, index, colonnes en base de données
- `down` joue le rôle inverse de l'opération `up`

```php
public function up(): void
{
    Schema::create('flights', function (Blueprint $table) {
        $table->id();
        $table->string('name')
    });
}
```

Exécuter des migrations

- `php artisan migrate` : exécute les migrations
- `php artisan migrate:status` : donne le statut des migrations
- `php artisan migrate:rollback` : rétrograde la migration
- `php artisan migrate:reset` : réinitialisation totale

Création de table

- La création de tables se fait à l'aide de la façade `Schema` grâce à sa méthode `create`
- La fonction `create` prend deux paramètres, le nom de la table suivi d'une callback avec un paramètre de type `Blueprint`
- A l'aide des méthodes de l'objet `Blueprint`, il suffit ensuite de définir les colonnes ainsi que leurs contraintes

Principaux types de colonnes

La classe Blueprint fournit un grand nombre de types de colonnes pour configurer les colonnes, par exemple:

- `$table->string('name', length=50);`
- `$table->integer('votes');`
- `$table->datetime('created_at')`
- `$table->id()`
- `$table->timestamps(precision: 0);`

Pour en savoir plus

Contraintes de tables

Il est également possible d'ajouter une ou plusieurs contraintes sur chacune des colonnes:

- `->nullable($value = true)`
- `->default($value)`
- `->useCurrent()` : CURRENT_TIMESTAMP
- `->useCurrentOnUpdate()`

Gestion des index

Commande	Description
`$table->primary('id');`	Ajoute une clé primaire
`$table->primary(['id', 'parent_id']);`	Créé une clé composite
`$table->unique('email');`	Ajoute un index unique
`$table->index('state');`	Ajoute un index

Clé étrangère

- Une première technique consiste à utiliser la méthode `foreign`

```
$table->foreign('user_id')->references('id')->on('users');
```

- Une syntaxe simplifiée a été mise en place via `foreignId`

```
$table->foreignId('user_id')->constrained();
```

- Si une classe Eloquent existe, on peut également utiliser:

```
$table->foreignIdFor(User::class)->constrained();
```

La méthode constrained

- `constrained` utilise des conventions pour déterminer la table et la colonne référencées
- Si le nom de la table ne correspond pas aux conventions de Laravel, il peut être fournit à la méthode

```
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('user_id')->constrained(
        table: 'users', indexName: 'posts_user_id'
    );
});
```

Raccourci de contraintes

Method	Description
$table->cascadeOnUpdate();	Mise à jour en cascade
$table->restrictOnUpdate();	Mise à jour restreinte
$table->noActionOnUpdate();	Pas d'action à la suppression
$table->cascadeOnDelete();	Suppression en cascade
$table->restrictOnDelete();	Suppression restreinte
$table->nullOnDelete();	Met la valeur nulle à la suppression

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Eloquent ORM

Introduction

- Laravel inclut Eloquent, un **Object Relational Mapper** (ORM) qui simplifie l'interaction avec la base de données
- Lorsque l'on utilise Eloquent, chaque table de la base de données a un "**modèle**" correspondant qui est utilisé pour interagir avec cette table
- Outre la **récupération des enregistrements** de la table de base de données, les modèles Eloquent permettent également d'**insérer**, de **mettre à jour** et de **supprimer** des enregistrements de la table

Générer un modèle

- Les modèles sont générés dans le répertoire `app\Models` et sont créés à partir de la commande `php artisan make:model Flight`
- Artisan permet également de générer plusieurs classes à partir d'un modèle: `migration` (`m`), `factory` (`f`), `seed` (`s`), `policy` (`p`), `form request` (`R`), `controller` (`c`), `ressource` (`r`)
- Exemple : `php artisan make:model Flight -crR`

Examiner un modèle

- Il est parfois difficile de déterminer tous les attributs et relations disponibles d'un modèle en parcourant le code
- La commande php artisan `php artisan model:show Flight` fournit une vue d'ensemble pratique de tous les attributs et relations du modèle

Nom de table

- Par défaut, Eloquent va nommer les noms de tables au pluriel en snake case, pour la classe `Flight` cela donnera `flights`
- La table `AirHockey` se nommerait : `air_hockeys`
- Si l'on souhaite modifier le nom de la table, il suffit de surcharger l'attribut `$table` avec une valeur par défaut

```
class Flight extends Model
{
    protected $table = 'my_flights';
}
```

Clé primaire

- Par défaut Eloquent va attribuer une clé primaire nommée `id` au model
- Il est possible de renommer cette clé primaire en surchargeant l'attribut `protected $primaryKey = 'flight_id';`
- La clé primaire par défaut est une valeur qui s'incrémente, il est possible de supprimer ce comportement en surchargeant l'attribut: `public $incrementing = false;`
- Pour définir le type qui sera utilisé: `protected $keyType = 'string';`

Utilisation de UUID

- Plutôt que d'utiliser des entiers, Eloquent permet l'utilisation de UUID
- Les UUID sont généralement plus performants pour l'indexation car Eloquent les génèrent de manière ordonnées

```
class Article extends Model
{
    use HasUuids;
}
```

Timestamps

- Eloquent attend la présence des colonnes `created_at` et `updated_at` sur les modèles générés pour pouvoir les gérer automatiquement
- Pour supprimer ce comportement il faut ajouter l'attribut suivant :

```php
class Flight extends Model
{
    public $timestamps = false;
}
```

- Pour customiser le format de date : `protected $dateFormat = 'U';`

Valeurs d'attributs par défaut

- Par défaut, une instance de modèle nouvellement instanciée ne contient aucune valeur d'attribut
- Pour définir les valeurs par défaut de certains attributs du modèle, il faut les placer dans une propriété `$attributes`

```php
protected $attributes = [
    'options' => '[]',
    'delayed' => false,
];
```

Lire des données

- Eloquent est une sorte de Query Builder sous stéroïdes, il permet de récupérer les données stockées en base de données à l'aide des modèles
- La méthode `all` effectuée sur un modèle permet de récupérer tous les enregistrements en base de données

```
foreach (Flight::all() as $flight) {
    echo $flight->name;
}
```

Utilisation du query builder

- Les modèles Eloquent s'utilisent de la même façon qu'on query builder à l'aide de méthode pour effectuer des requêtes spécifiques
- Il suffit donc d'utiliser différentes contraintes et invoquer la méthode `get` pour que la requête s'effectue

```
$flights = Flight::where('active', 1)
               ->orderBy('name')
               ->take(10)
               ->get();
```

Les collections

- Les requêtes faites avec Eloquent retournent des collections héritant de : `Illuminate\Support\Collection`
- La classe `Collection` de Eloquent rajoutent quelques méthodes spécifiques à l'ORM

Optimisation de requêtes

- Eloquent fournit différent mécanismes pour éviter de saturer la mémoire lors de la lecture d'un grand nombre d'éléments
- `chunk` : renvoie un morceau de requête
- `chunkById` : similaire à chunk mais adapté lors de modification et utilisation d'un filtre
- `lazy` : utilise chunk, mais récupère les enregistrements un à un
- `lazyById` : similaire à chunkById

Lire un enregistrement

Il est souvent intéressant de récupérer un unique enregistrement en base de données, pour cela, Eloquent nous fournit 5 méthodes:

- `find`: cherche le modèle par son id
- `first`: cherche le premier modèle qui rempli la contrainte
- `firstWhere`: raccourci pour la méthode précédente
- `findOr`: renvoie le résultat d'une closure si aucun enregistrement n'est trouvé
- `firstOr`: similaire à `findOr`

L'exception Not Found

- Que ce soit dans les routes ou les controllers, il peut souvent être intéressant de renvoyer une erreur 404 si un modèle n'a pas été trouvé
- Pour cette raison, Eloquent fournit deux méthodes:
  - `findOrFail`
  - `firstOrFail`

Fonctions d'agrégations

- Les fonctions d'agrégations `sum`, `max` et `count` sont présentent sur les modèles
- Elle permettent de retourner un scalaire

```php
$count = Flight::where('active', 1)->count();

$max = Flight::where('active', 1)->max('price');
```

Insertion de modèle

- On peut instancier un modèle, le peupler, et enfin appeler la méthode `save`
- On peut également appeler la méthode statique `create` du modèle en lui passant un tableau associatif avec les informations du modèle
- La méthode `create` nécessite que les attributs `fillable` ou `guarded` soient saisis, ceci est un mécanisme de protection contre l'assignation massive

Modifier un modèle

- La méthode `save` sert également à mettre à jour un enregistrement
- Pour l'utiliser il suffit de lire un enregistrement, le modifier puis appeler la méthode
- Si le champ `update_at` est présent, il sera automatiquement mis à jour en base de données

```php
$flight = Flight::find(1);

$flight->name = 'Paris to London';

$flight->save();
```

Mise à jour de plusieurs modèles

- Il est possible de modifier plusieurs enregistrements à l'aide de Eloquent en utilisant la méthode `update`
- Pour cela il suffit de filtrer le modèle puis effectuer la méthode update

```
Flight::where('active', 1)
      ->where('destination', 'San Diego')
      ->update(['delayed' => 1]);
```

Supprimer un modèle

- Pour supprimer un modèle il suffit d'appeler la méthode `delete` sur un modèle lu en base de données
- Il existe également la méthode `truncate` qui permet de supprimer tous les enregistrements de la table et réinitialiser le compteur
- Dans le cas où l'on connaît l'ID du modèle à supprimer, on peut également utiliser la méthode `destroy` qui est plus optimisée

Soft delete

- Eloquent met également en place un mécanisme de soft delete pour éviter de supprimer complètement une entité
- Ce mécanisme implémente une colonne `delete_at` en base de données
- Pour activer le soft delete il faut ajouter le trait `use SoftDeletes;` dans le modèle

```
Schema::table('flights', function (Blueprint $table) {
    $table->softDeletes();
});
```

Manipuler des éléments soft deleted

- La méthode `withTrashed` inclus les modèles supprimés
- La méthode `onlyTrashed` requête uniquement parmi les modèles supprimés
- La méthode `restore` restore un élément supprimé
- La méthode `forceDelete` supprime définitivement un enregistrement

One-To-One

Pour définir une relation One to One, il faut ajouter une méthode avec le nom du membre lié dans le modèle ainsi que le trait `HasOne` au niveau de la classe:

```php
public function phone(): HasOne
{
    return $this->hasOne(Phone::class);
}
```

Une fois défini, on peut récupérer le membre via la méthode:

```php
$phone = User::find(1)->phone;
```

Relation inverse

Pour paramétrer la relation inverse dans le modèle, on utilise le trait BelongsTo avec le nom du modèle auquel il fait référence

```php
public function user(): BelongsTo
{
    return $this->belongsTo(User::class);
}
```

One-To-Many

De la même manière que pour le One to One, on créé une méthode dans le modèle qui porte le nom du modèle au pluriel, puis on ajoute le trait `HasMany` pour utiliser la méthode

```
public function comments(): HasMany
{
    return $this->hasMany(Comment::class);
}
```

```
$comments = Post::find(1)->comments;

foreach ($comments as $comment) {
}
```

Relation inverse

La relation inverse fonctionne de la même façon que pour le One to One :

```php
public function post(): BelongsTo
{
    return $this->belongsTo(Post::class);
}
```

Navigation entre relations

Lors de d'une requête sur un enfant d'une relation "appartient à", on peut construire manuellement la clause `where` pour récupérer les modèles Eloquent correspondants:

```
$posts = Post::where('user_id', $user->id)->get();
```

Une méthode raccourci nommée `whereBelongsTo` simplifie le travail:

```
$posts = Post::whereBelongsTo($user)->get();
```

Many-To-Many

Une relation many to many est construite à l'aide d'une table de jointure intermédiaire

```
users
    id - integer
    name - string

roles
    id - integer
    name - string

role_user
    user_id - integer
    role_id - integer
```

Structure d'un modèle

- Les relations many to many sont générés à partir du trait `BelongsToMany`:

```php
public function roles(): BelongsToMany
{
    return $this->belongsToMany(Role::class);
}
```

- On peut ensuite accéder à la collection:

```php
$user = User::find(1);
foreach ($user->roles as $role) {
}
```

Méthodes many to many

- A l'aide du Query Builder on peut s'amuser à faire des requêtes sur les relations enfants

```
$roles = User::find(1)->roles()->orderBy('name')->get();
```

Informations table intermédiaire

Dans les relations many to many il arrive que l'on stocke des informations dans la table intermédiaire, Eloquent nous fournit un moyen d'y accéder à l'aide la méthode `pivot`

```
$user = User::find(1);

foreach ($user->roles as $role) {
    echo $role->pivot->created_at;
}
```

Informations table intermédiaire

- Par défaut, seules les clés du modèle seront présentes sur le modèle `pivot`
- Si la table intermédiaire contient des attributs supplémentaires, il faut les spécifier lors de la définition de la relation

```php
return $this->belongsToMany(Role::class)->withPivot('active', 'created_by');
```

- Pour récupérer les timestamps de la table intermédiaire:

```php
return $this->belongsToMany(Role::class)->withTimestamps();
```

Enregistrer une relation

- Plutôt que d'enregistrer le `post_id` dans le commentaire, on peut directement passé par une entité post

```php
$comment = new Comment(['message' => 'A new comment.']);
$post = Post::find(1);
$post->comments()->save($comment);
```

```php
$post = Post::find(1);

$post->comments()->saveMany([
    new Comment(['message' => 'A new comment.']),
    new Comment(['message' => 'Another new comment.']),
]);
```

Utilisation de la méthode create

La méthode `create` permet également de sauvegarder des relations à l'aide d'un tableau associatif

```
$post = Post::find(1);

$comment = $post->comments()->create([
    'message' => 'A new comment.',
]);
```

Association de relations

- Dans une relation one to many, on peut facilement associer une entité enfant à son parent à l'aide de `associate`

```
$account = Account::find(10);
$user->account()->associate($account);
$user->save();
```

- L'opération inverse se fait avec `dissociate` (si la clé étrangère est nullable)

```
$user->account()->dissociate();
$user->save();
```

Lier en many to many

Pour lier un enregistrement entre deux tables, on peut utiliser la méthode `attach` qui prend en paramètre l'id de l'entité à associer, ainsi q'un tableau de paramètre optionnel à ajouter

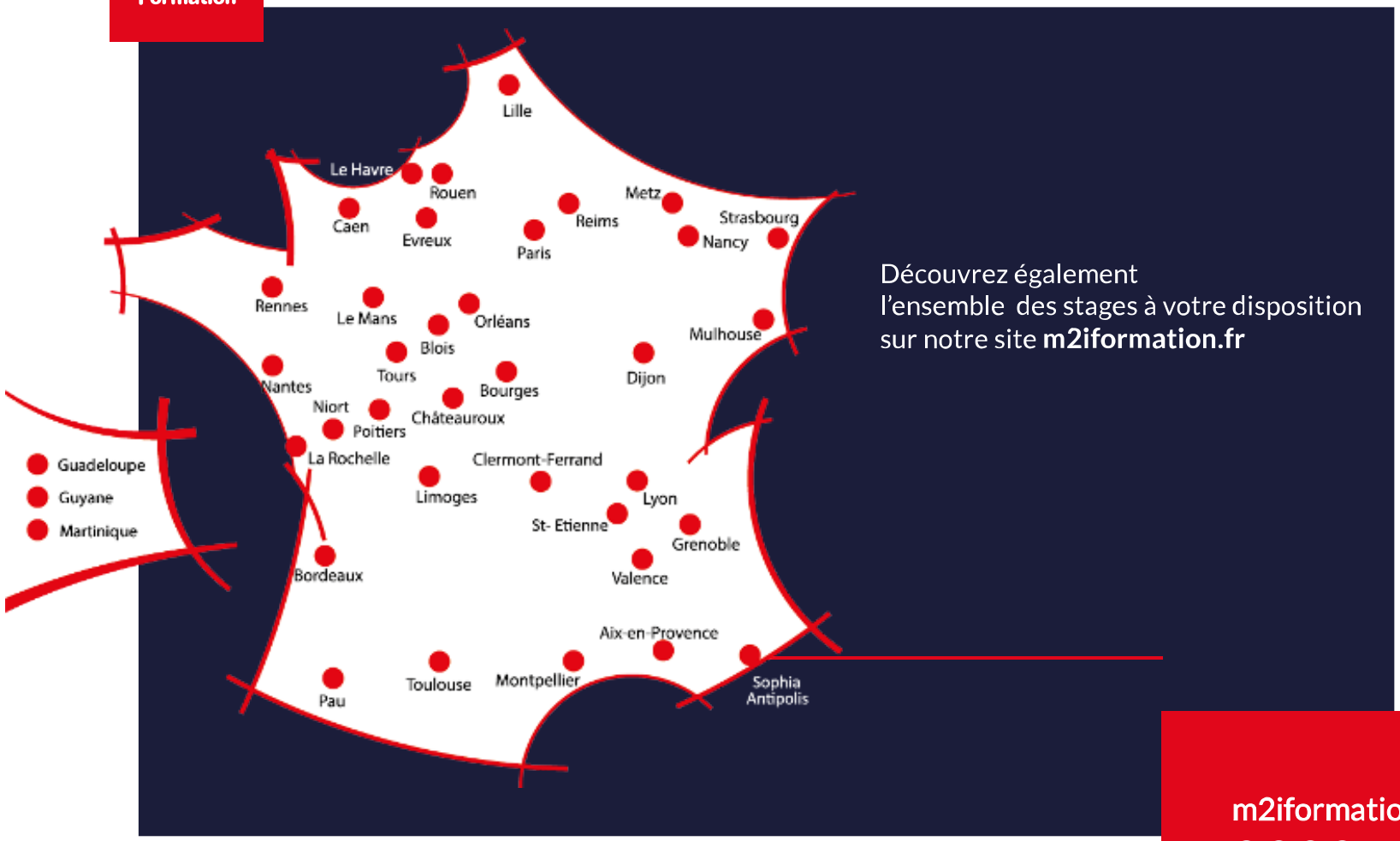
```
$user = User::find(1);
$user->roles()->attach($roleId);
$user->roles()->attach($roleId, ['expires' => $expires]);
```

`detach` effectue l'opération inverse

```
$user->roles()->detach($roleId);
$user->roles()->detach();
```

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr