Gestion du routage

1. Liaison modèle de route

Lorsque vous injectez un ID de modèle dans une route ou une action de contrôleur, vous allez souvent interroger la base de données pour récupérer le modèle qui correspond à cet ID. Laravel route model binding fournit un moyen pratique d'injecter automatiquement les instances de modèle directement dans vos routes. Par exemple, au lieu d'injecter l'ID d'un utilisateur, vous pouvez injecter l'instance entière du modèle User qui correspond à l'ID donné.

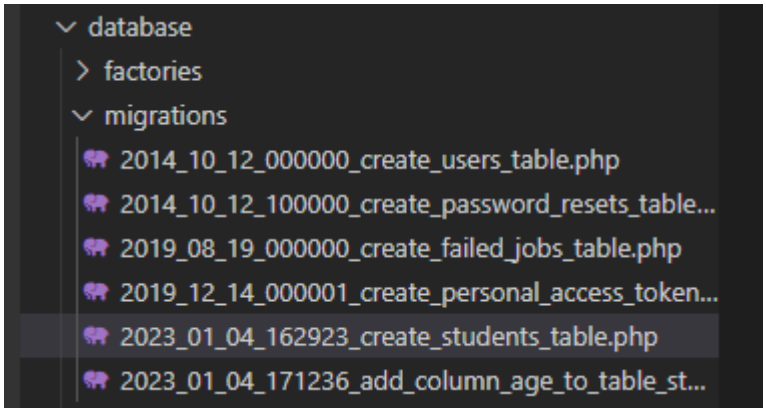
Créer un modèle

- Tous les Models de l'application Laravel sont stockés dans le dossier **App**.
- Le modèle peut être créé simplement en utilisant la commande make: model artisan comme suit:
- Syntaxe: **php artisan make:model Student**

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    use HasFactory;
}
```

- Après créer une migration pour définir les propriétés de modèle **Student** en utilisant la commande **:**

php artisan make:migration create_students_table

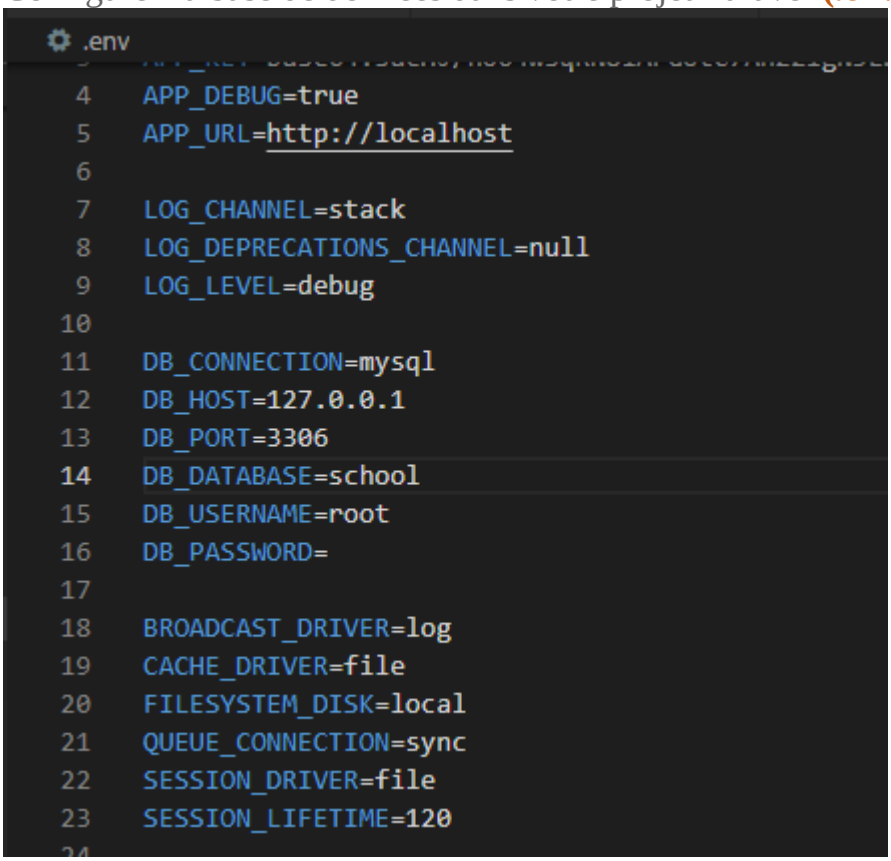


- Commencer par définir les champs de votre base de données pour notre table **students** dans notre fichier de migration comme suit:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string("name");
            $table->string("mail");
            $table->integer("phone");
            $table->string("section");
            $table->string("image");
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
```

```
    {
        Schema::dropIfExists('students');
    }
};
```

- Utilisez PHPMyAdmin pour créer une Base de données dont le nom “school”.
- Configurez la base de données dans votre projet Laravel **(.env)**



- Lancez la migration : **php artisan migrate**
- Vous devez ainsi vous retrouver avec quatre tables dans votre base.

Liaison Implicite

Laravel résout automatiquement les modèles Eloquent définis dans les routes ou les actions de contrôleur dont les noms de variables à indication de type correspondent à un nom de segment de route. Par exemple :

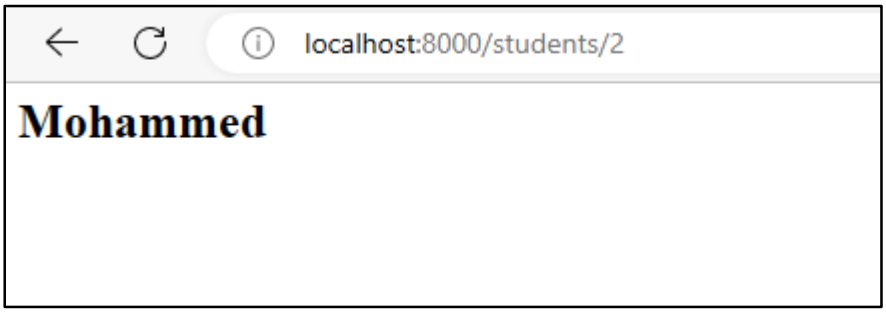
```
use App\Models\Student;

Route::get('students/{student}', function (Student $student)
{
return $student->name;
});
```

Puisque la variable $user est signalée comme étant le modèle Eloquent App\Models\Student et que le nom de la variable correspond au segment {student} Laravel injectera automatiquement l'instance de modèle dont l'ID correspond à la valeur correspondante de l'URI de la requête. Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse HTTP 404 sera automatiquement générée.

Bien sûr, la liaison implicite est également possible en utilisant les méthodes du contrôleur. Encore une fois, notez que le segment {student} correspond à la variable $student dans la fonction de rappel qui contient une indication de type App\Models\Student.

Comme nous avons lié tous { student } paramètres {student} au modèle App\Student , une instance Student sera injectée dans la route. Ainsi, par exemple, une demande de students/1 injectera l'instance d'utilisateur de la base de données dont l'identifiant est 1.

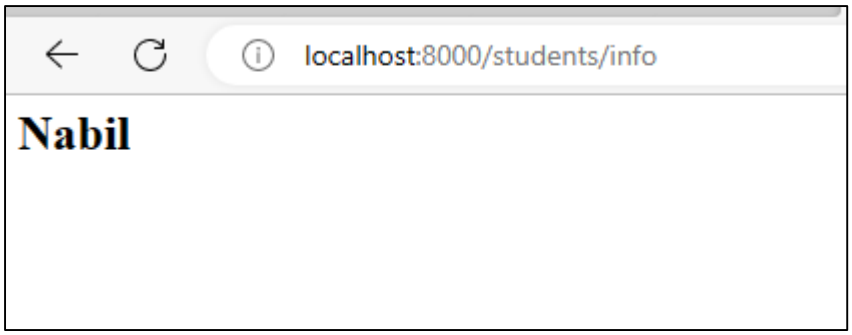
Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse HTTP 404 sera automatiquement générée



Personnalisation de la clé

Parfois, vous pouvez souhaiter résoudre des modèles Eloquent en utilisant une colonne **section** autre que **id**. Pour ce faire, vous pouvez spécifier la colonne dans la définition du paramètre de la route :

```
use App\Models\Student;

Route::get('students/{student :section}', function (Student
$student)
{
return $student->name;
});
```



1.1. Liaison Explicite

Pour enregistrer une liaison explicite, utilisez la méthode du modèle du routeur pour spécifier la classe d'un paramètre donné. Vous devez définir vos liaisons de modèle explicites dans la méthode de démarrage de la classe RouteServiceProvider

```
use App\Models\User;
use Illuminate\Support\Facades\Route;

/**
 * Define your route model bindings, pattern filters, etc.
 *
 * @return void
 */
public function boot()
{
    Route::model('user', User::class);

    // ...
}
```

Ensuite, définissez une route qui contient un paramètre {user} :

```
use App\Models\User;

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

Comme nous avons lié tous {user} paramètres {user} au modèle App\User, une instance User sera injectée dans la route. Ainsi, par exemple, une demande de profile/1 injectera l'instance d'utilisateur de la base de données dont l'identifiant est 1. Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse HTTP 404 sera automatiquement générée

2. Routes de repli (Fallback Routes)

En utilisant la méthode **Route::fallback**, vous pouvez définir une route qui sera exécutée lorsqu'aucune autre route ne correspond à la requête entrante. En règle générale, les requêtes non traitées renvoient automatiquement une page "404" via le gestionnaire d'exceptions de votre application. Cependant, comme vous définissez généralement la route de repli dans votre fichier **routes/web.php**, tous les middlewares du groupe middleware Web s'appliqueront à cette route.

```
Route::fallback(function () {
    return 'Error'
});
```

3. Usurpation (spoofing) de la méthode du formulaire

Les formulaires HTML ne prennent pas en charge les actions PUT, PATCH ou DELETE. Ainsi, lors de la définition de routes appelées PUT, PATCH ou DELETE à partir d'un formulaire HTML, vous devrez ajouter un champ masqué **_method** au formulaire. La valeur envoyée avec le champ **_method** sera utilisée comme méthode de requête HTTP :

```
<form action="/example" method="POST">
    <input type="hidden" name="_method" value="PUT">
    <input type="hidden" name="_token" value="{{ csrf_token() }}">
</form>
```

Pour plus de commodité, vous pouvez utiliser la directive Blade @method pour générer le champ _method de saisie :

```
<form action="/example" method="POST">
    @method('PUT')
    @csrf
</form>
```

1. Accès à la route courante

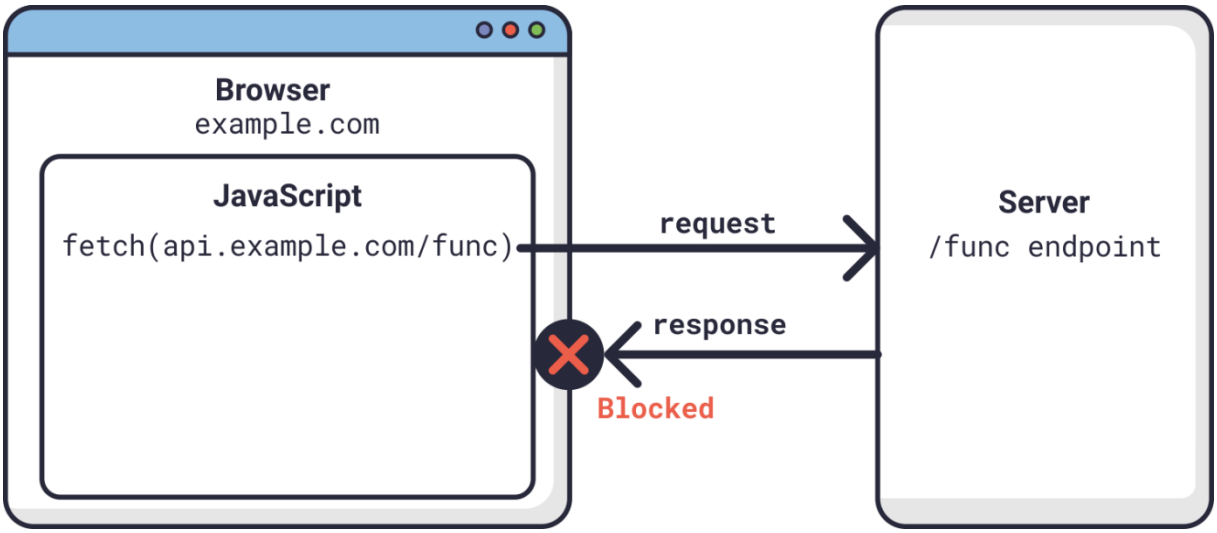
Vous pouvez utiliser les méthodes **current**, **currentRouteName** et **currentRouteAction** sur la façade Route pour accéder aux informations sur la route traitant la requête entrante.

Pour Obtenir le nom de la route dans le fichier Blade.

```
<body>
    <h1> Home page</h1>
<h1>Route Name :  {{Route::currentRouteName()}}</h1>

<h1>Current Route Action : {{Route::currentRouteAction()}}</h1>

</body>
```

2. Cross-origin resource sharing (CORS)

Le « *Cross-origin resource sharing* » (CORS) ou « partage des ressources entre origines multiples » (en français, moins usité) est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant. Un agent utilisateur réalise une requête HTTP **multi-origine (***cross-origin***)** lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.



Prenons un exemple de requête multi-origine : une page HTML est servie depuis http://domaine-a.com contient un élément <img> src ciblant http://domaine-b.com/image.jpg. Aujourd'hui, de nombreuses pages web chargent leurs ressources (feuilles CSS, images, scripts) à partir de domaines séparés (par exemple des CDN (*Content Delivery Network* en anglais ou « Réseau de diffusion de contenu »).

5. Mise en cache des routes

Lors du déploiement de votre application en production, vous devez tirer parti du cache de routage de Laravel. L'utilisation du cache de route réduira considérablement le temps nécessaire pour enregistrer toutes les routes de votre application. Pour générer un cache de route, exécutez la commande Artisan **route:cache** :

```
php artisan route:cache
```

• Après avoir exécuté cette commande, votre fichier de routes en cache sera chargé à chaque requête. N'oubliez pas que si vous ajoutez de nouvelles routes, vous devrez générer un nouveau cache de routes. Pour cette raison, vous ne devez exécuter la commande **route:cache** pendant le déploiement de votre projet.

• Vous pouvez utiliser la commande **route:clear** pour vider le cache de route :

```
php artisan route:clear
```

Pour mettre en cache votre fichier de routes, vous devez utiliser toutes les routes de contrôleurs et de ressources (pas de fermetures de routes). Si votre application n'utilise aucune fermeture de route, vous pouvez exécuter **php artisan route:cache**. Laravel sérialisera les résultats de vos fichiers de routes. Si vous voulez supprimer le cache, exécutez **php artisan route:clear**.