Manipulation des réponses Http :

A. Création

1.Chaînes et tableaux :

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

En plus de renvoyer des chaînes à partir de vos routes et de vos contrôleurs, vous pouvez également renvoyer des tableaux. Le framework convertira automatiquement le tableau en une réponse JSON :

```
Route::get('/url1', function () {
 return 'Hello World';
});
Route::get('/url2', function () {
return [1, 2, 3];
 });
```

2.Objets de réponse

• En règle générale, vous ne renverrez pas simplement de simples chaînes ou des tableaux à partir de vos actions de routage. Au lieu de cela, vous renverrez des instances **Illuminate\Http\Response** ou des vues complètes.

• Le renvoi d'une instance complète **Response** vous permet de personnaliser le code d'état HTTP et les en-têtes de la réponse. Une instance **Response** hérite de la classe **Symfony\Component\HttpFoundation\Response**, qui fournit diverses méthodes pour créer des réponses HTTP :

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

3.Modèles et collections éloquents

Vous pouvez également renvoyer des modèles et des collections **ORM Eloquent** directement à partir de vos routes et de vos contrôleurs. Lorsque vous le faites, Laravel convertira automatiquement les modèles et collections en réponses JSON tout en respectant les attributs cachés du modèle :

```
use App\Models\User;
 Route::get('/user/{user}', function (User $user) {
return $user;
});
```

4.Attacher des cookies aux réponses

Vous pouvez attacher un cookie à une instance Illuminate\Http\Response sortante à l'aide de la méthode cookie. Vous devez transmettre le nom, la valeur et le nombre de minutes pendant lesquelles le cookie doit être considéré comme valide à cette méthode

```
return response('Hello World')->cookie( 'name', 'value', $minutes );
```

Si vous souhaitez vous assurer qu'un cookie est envoyé avec la réponse sortante mais que vous n'avez pas encore d'instance de cette réponse, vous pouvez utiliser la façade Cookie pour "mettre en file d'attente" les cookies à attacher à la réponse lorsqu'elle est envoyée. La méthode queue accepte les arguments nécessaires pour créer une instance de cookie. Ces cookies seront joints à la réponse sortante avant qu'elle ne soit envoyée au navigateur :

```
use Illuminate\Support\Facades\Cookie;
Cookie::queue('name', 'value', $minutes);
return response('Hello World') ;
```

Si vous souhaitez générer une instance **Symfony\Component\HttpFoundation\Cookie** pouvant être attachée à une instance de réponse ultérieurement, vous pouvez utiliser l'assistant cookie global. Ce cookie ne sera pas renvoyé au client sauf s'il est attaché à une instance de réponse :

```
$cookie = cookie('name', 'value', $minutes);
return response('Hello World')->cookie($cookie);
```

5.Expiration anticipée des cookies

Si vous n'avez pas encore d'instance de la réponse sortante, vous pouvez utiliser la méthode expire de la façade Cookie pour faire expirer un cookie :

```
Cookie::expire('name');
```

6.Cookies et cryptage

Par défaut, tous les cookies générés par Laravel sont cryptés et signés afin qu'ils ne puissent pas être modifiés ou lus par le client. Si vous souhaitez désactiver le chiffrement pour un sous-ensemble de cookies générés par votre application, vous pouvez utiliser la propriété **$except** du middleware **App\Http\Middleware\EncryptCookies**, qui se trouve dans le répertoire **app/Http/Middleware** :

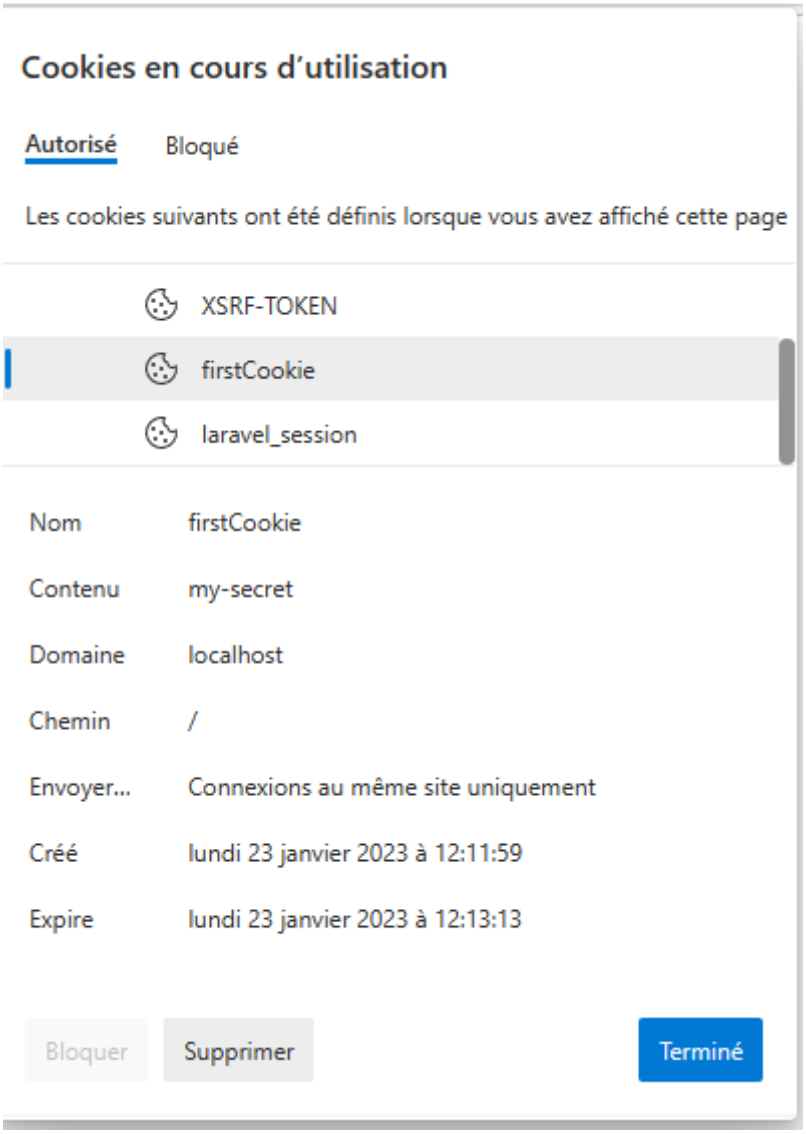
```php
<?php

namespace App\Http\Middleware;

use Illuminate\Cookie\Middleware\EncryptCookies as Middleware;

class EncryptCookies extends Middleware
{
    /**
     * The names of the cookies that should not be encrypted.
     *
     * @var array<int, string>
     */
    protected $except = [
      ' firstCookie'

    ];
}
```



B. Redirection

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

1.Redirection vers des routes nommées

Lorsque vous appelez l'assistant redirect sans paramètre, une instance de **Illuminate\Routing\Redirectorest** renvoyée, vous permettant d'appeler n'importe quelle méthode sur l'instance Redirector. Par exemple, pour

générer un **RedirectResponsevers** une route nommée, vous pouvez utiliser la méthode route :

```
return redirect()->route('login');
```

Si votre route a des paramètres, vous pouvez les passer comme deuxième argument à la méthode route :

```
// pour une route avec l'URI suivant:
/profile/{id}
 return redirect()->route('profile', ['id' => 1]);
```

2.Remplir les paramètres via des modèles éloquents

Si vous redirigez vers une route avec un paramètre "ID" qui est renseigné à partir d'un modèle Eloquent, vous pouvez transmettre le modèle lui-même. L'ID sera extrait automatiquement :

```
// pour une route avec l'URI suivant : /profile/{id}
 return redirect()->route('profile',$student);
or
<a href="{{route('profile',$student)}}"><h1>{{$student->name}}</h1> </a>
```

Si vous souhaitez personnaliser la valeur placée dans le paramètre route, vous pouvez spécifier la colonne dans la définition du paramètre Route.

```
Route::get('/students/{student:name}',[StudentController::class,'show'])-
>name('students.show');
```

3.Redirection vers des domaines externes

Parfois, vous devrez peut-être rediriger vers un domaine en dehors de votre application. Vous pouvez le faire en appelant la méthode **away**, qui crée un **RedirectResponse** sans aucun codage, validation ou vérification d'URL supplémentaire :

```
return redirect()->away('https://www.google.com');
```

4.Redirection avec des données de session flashées

La redirection vers une nouvelle **URL** et le flashage des données vers la session sont généralement effectués en même temps. En règle générale, cela se fait après avoir effectué une action avec succès lorsque vous envoyez un message de réussite à la session. Pour plus de commodité, vous pouvez créer une instance **RedirectResponse** et envoyer des données flash à la session dans une seule chaîne de méthodes fluide :

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

Une fois l'utilisateur redirigé, vous pouvez afficher le message flashé de la session. Par exemple, en utilisant la syntaxe **Blade** :

```
 @if (Session::has('success'))

{{Session::get('success')}}

 @endif
```

C. Types de réponse

1.Afficher les vues (views)

Vous pouvez également renvoyer une vue comme contenu de la réponse, vous devez utiliser la méthode **view** :

```
return view('students.index');
```

2.Réponses JSON

La méthode **json** va automatiquement définir l'en-tête Content-Type à application/json, et convertir le tableau donné en JSON:

```
   $data = [
        [
```

```
            "id" => 7,
            "name" => "na like this",
            "description" => "",
        ], [
            "id" => 5,
            "name" => "write a book",
            "description" => "hohoho",
        ]
    ];
    return response()->json($data);
```

3.Téléchargements de fichiers

La méthode **download** peut être utilisée pour générer une réponse qui force le navigateur de l'utilisateur à télécharger le fichier au chemin donné. La méthode **download** accepte un nom de fichier comme deuxième argument de la méthode, qui déterminera le nom de fichier qui est vu par l'utilisateur téléchargeant le fichier.

```
return response()->download("storage/students/photo1.png");
```

4.Téléchargements en streaming

Parfois, vous souhaiterez peut-être transformer la réponse de chaîne d'une opération donnée en une réponse téléchargeable sans avoir à écrire le contenu de l'opération sur le disque. Vous pouvez utiliser la méthode **streamDownload** dans ce scénario. Cette méthode accepte une fonction de rappel, un nom de fichier et un tableau facultatif d'en-têtes comme arguments :

```
    return response()->streamDownload(function () {
            echo 'CSV Contents...';
                },
                'downloadme.csv',
              [
                'Content-Type' => 'text/csv',
              ]
            );
```

5.Fichier de réponses

La méthode **file** peut être utilisée pour afficher un fichier, tel qu'une image ou un PDF, directement dans le navigateur de l'utilisateur au lieu de lancer un téléchargement. Cette méthode accepte le chemin d'accès au fichier comme premier argument et un tableau d'en-têtes comme deuxième argument :

```
        return response()->file('storage/laravel-fr.pdf');
```

6.Réponse Marco

Si vous souhaitez définir une réponse personnalisée que vous pouvez réutiliser dans une variété de vos routes et contrôleurs, vous pouvez utiliser la méthode **macro** sur la façade **Response**. En règle générale, vous devez appeler cette méthode à partir de la méthode **boot** de l'un des fournisseurs de services de votre application, tel que le fournisseur de services : **App\Providers\AppServiceProvider**

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\Response;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
```

```
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Response::macro('caps', function ($value) {
            return Response::make(strtoupper($value));
            });

    }
}
```

La fonction **macro** accepte un nom comme premier argument et une fermeture comme deuxième argument. La fermeture de la macro sera exécutée lors de l'appel du nom de la macro depuis une implémentation :

```
Route::get('/', function () {

     return response()->caps('morocco');
});
```