Manipulation des requêtes Http :

A. Interaction avec les requêtes

1.Introduction :

La classe de Laravel **Illuminate\Http\Request** fournit un moyen orienté objet d'interagir avec la requête HTTP actuelle gérée par votre application, ainsi que de récupérer l'entrée, les cookies et les fichiers qui ont été soumis avec la requête.

2.Accéder à la demande:

Pour obtenir une instance de la requête HTTP actuelle via l'injection de dépendances, vous devez indiquer la classe **Illuminate\Http\Request** sur votre méthode de fermeture de route ou de contrôleur. L'instance de requête entrante sera automatiquement injectée par le conteneur de service Laravel :

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Store a new user.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $name = $request->input('name');
        $mail = $request->mail ;

        //
    }
}
```

Comme mentionné, vous pouvez également indiquer la classe **Illuminate\Http\Request** sur une fermeture d'itinéraire. Le conteneur de

service injectera automatiquement la requête entrante dans la fermeture lors de son exécution :

```
use Illuminate\Http\Request;

Route::get('/', function (Request $request) {
    $request->isMethod('post') ;// affiche false
});
```

3.Paramètres d'injection de dépendance et de routage

Si votre méthode de contrôleur attend également une entrée d'un paramètre de route, vous devez lister vos paramètres de route après vos autres dépendances. Par exemple, si votre route est définie comme ceci :

// routes/web.php

```
use App\Http\Controllers\StudentController;

 Route::put('/user/{student}', [StudentController::class, 'update']);
```

Vous pouvez toujours taper le **Illuminate\Http\Request** et accéder à votre paramètre id route en définissant votre méthode de contrôleur comme suit :

// app/Http/Controllers/StudentController.php

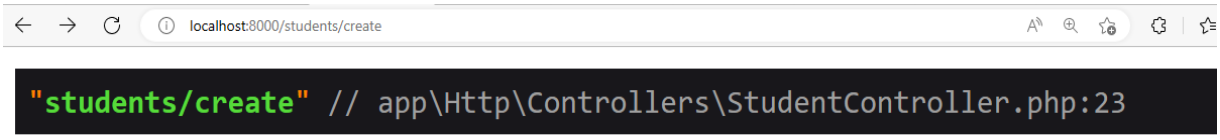
```
     public function update(Request $req, Student $student)
    {
    //
    }
```

4.Récupération du chemin de la requête

La méthode **path** renvoie les informations de chemin de la requête. Ainsi, si la requête entrante est ciblée sur **http://localhost:8000/students/create**, la méthode **path** retournera **students/create**:

```
 public function create(Request $request)
   {   dd($request->path());
       return view('students.create');
   }
```

//Output



5.Inspecter le chemin/la route de la demande

- La méthode **is** vous permet de vérifier que le chemin de la demande entrante correspond à un pattern donné. Vous pouvez utiliser le caractère * comme caractère générique lors de l'utilisation de cette méthode :

```
if ($request->is('students/*')) {
//
 }
```

- En utilisant la méthode **routeIs**, vous pouvez déterminer si la requête entrante correspond à une route nommée :

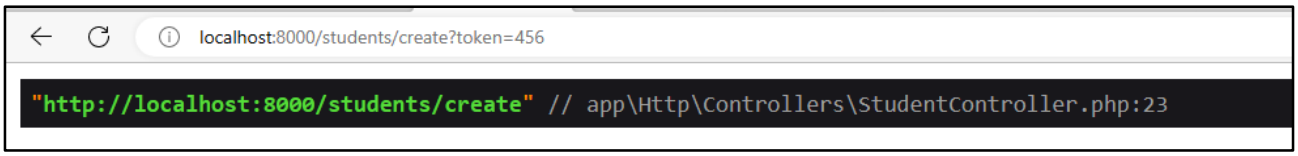
```
  if ($request->routeIs(students.edit')) {
 //
}
```

6.Récupération de l'URL de la requête

Pour récupérer l'URL complète de la requête entrante, vous pouvez utiliser les méthodes **url** ou **fullUrl**. La méthode url renverra l'URL sans la chaîne de requête (queryString), tandis que la méthode fullUrl inclut la chaîne de requête :

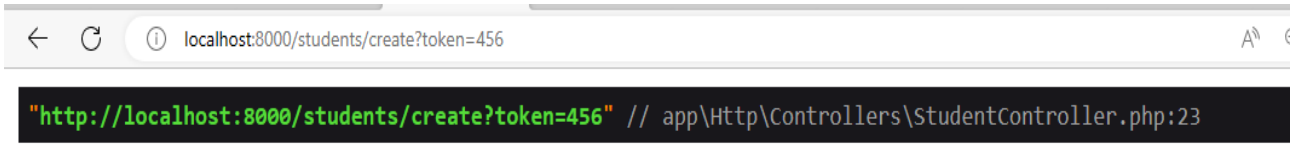
```
dd( $request->url());
```

//Output




```
dd($request->fullUrl());
```

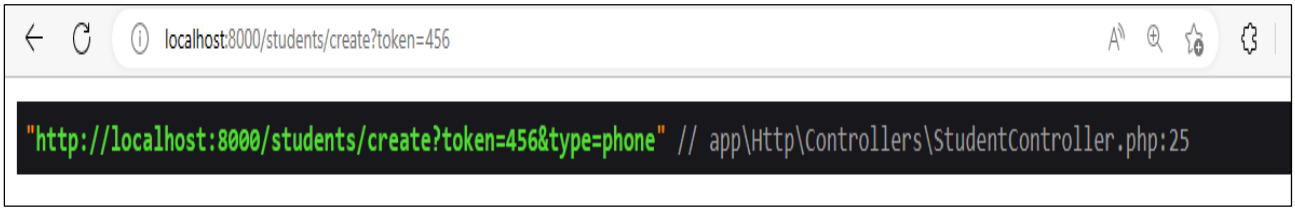
//Output



Si vous souhaitez ajouter des données de chaîne de requête à l'URL actuelle, vous pouvez appeler la méthode **fullUrlWithQuery**. Cette méthode fusionne le tableau donné de variables de chaîne de requête avec la chaîne de requête actuelle :

```
public function create(Request $request)
{
    $uri=$request->fullUrlWithQuery(['type' => 'phone']);
    dd($uri);
    return view('students.create');
}
```

//Output

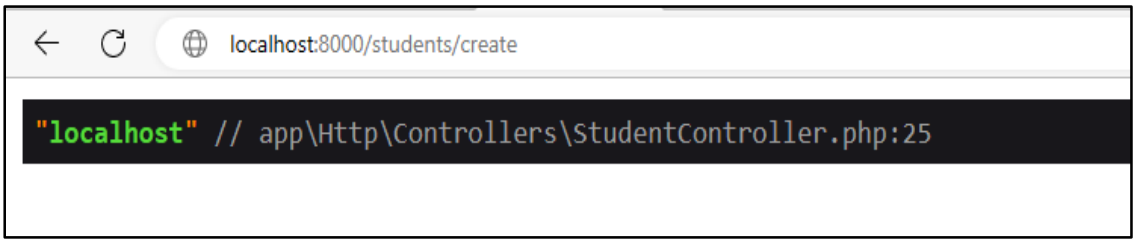


7.Récupération de l'hôte de requête

Vous pouvez récupérer "l'hôte" de la requête entrante via la méthode host :

```
public function create(Request $request)
{
    $host=$request->host();
    //$request->httpHost();
    // $request->schemeAndHttpHost();
    dd($host);
    return view('students.create');
}
```

//Output



8.Récupération de la méthode Request :

La méthode **method** renverra le verbe HTTP pour la requête. Vous pouvez utiliser la méthode **isMethod** pour vérifier que le verbe HTTP correspond à une chaîne donnée :

```
$method = $request->method();
 if ($request->isMethod('post')) {
//
 }
```

9.En-têtes de demande

Vous pouvez récupérer un en-tête de requête à partir de l'instance **Illuminate\Http\Request** à l'aide de la méthode header. Si l'en-tête n'est pas présent sur la requête, **null** sera renvoyé. Cependant, la méthode header accepte un deuxième argument facultatif qui sera retourné si l'en-tête n'est pas présent sur la requête :

```
$value = $request->header('X-Header-Name');
//Example :  $value = $request->header('cookie');
$value = $request->header('X-Header-Name', 'default');
```

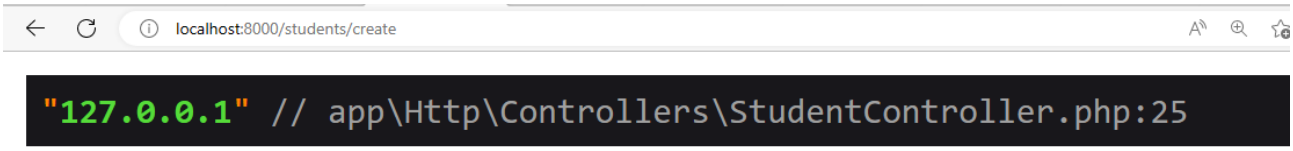
10. Demander l'adresse IP

La méthode **ip** peut être utilisée pour récupérer l'adresse IP du client qui a fait la requête à votre application :

```
  public function create(Request $request)
    {
        $ipAddress = $request->ip();
```

```
        dd($ipAddress);
        return view('students.create');
    }
```

//Output



11. Négociation de contenu

Étant donné que de nombreuses applications ne servent que du HTML ou du JSON, vous pouvez utiliser la méthode expectsJson pour déterminer rapidement si la requête entrante attend une réponse JSON :

```
if ($request->expectsJson())
{ // ...
 }
```

B. Input

1.Récupération de toutes les données d'entrée

Vous pouvez récupérer toutes les données d'entrée de la demande entrante sous forme d’un array en utilisant la méthode all. Cette méthode peut être utilisée que la requête entrante provienne d'un formulaire HTML ou soit une requête XHR :

```
$input = $request->all();
```

2.Récupération d'une valeur d'entrée

À l'aide de quelques méthodes simples, vous pouvez accéder à toutes les entrées utilisateur de votre instance **Illuminate\Http\Request** sans vous soucier du verbe HTTP utilisé pour la requête. Quel que soit le verbe HTTP, la méthode input peut être utilisée pour récupérer l'entrée utilisateur :

```
$name = $request->input('name');
```

Vous pouvez passer une valeur par défaut comme second argument de la méthode input. Cette valeur sera renvoyée si la valeur d'entrée demandée n'est pas présente dans la requête :

```
$name = $request->input('name', 'Sally');
```

Lorsque vous travaillez avec des formulaires contenant des entrées de tableau, utilisez la notation "point" pour accéder aux tableaux :

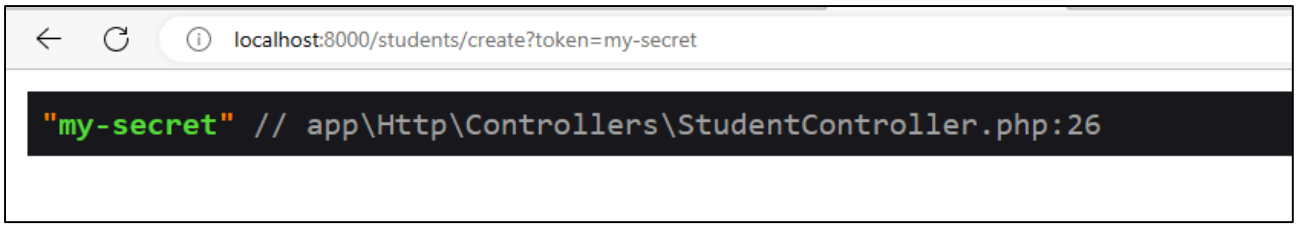
```
  <input type="checkbox" name="food[]"/>Apple
  <input type="checkbox" name="food[]"/>Pear
```

```
$name = $request->input('food.0')
```

3.Récupération de l'entrée de la chaîne de requête

Alors que la méthode input récupère les valeurs de l'intégralité de la charge utile de la requête (y compris la chaîne de requête), la méthode **query** ne récupère que les valeurs de la chaîne de requête :

```
$name = $request->query('name');
```



4.Récupération de valeurs d'entrée stringables

Au lieu de récupérer les données d'entrée de la requête en tant que primitive string, vous pouvez utiliser la méthode string pour récupérer les données de la requête en tant qu'instance de **Illuminate\Support\Stringable**:

```
$name = $request->string('name')->trim();
```

5.Récupération des valeurs d'entrée booléennes

Lorsqu'il s'agit d'éléments HTML tels que des cases à cocher, votre application peut recevoir des valeurs "véridiques" qui sont en fait des chaînes. Par exemple, "vrai". Pour plus de commodité, vous pouvez utiliser la méthode boolean pour récupérer ces valeurs sous forme de booléens. La méthode boolean renvoie true pour 1. Toutes les autres valeurs renverront false :

//Form.blade.php

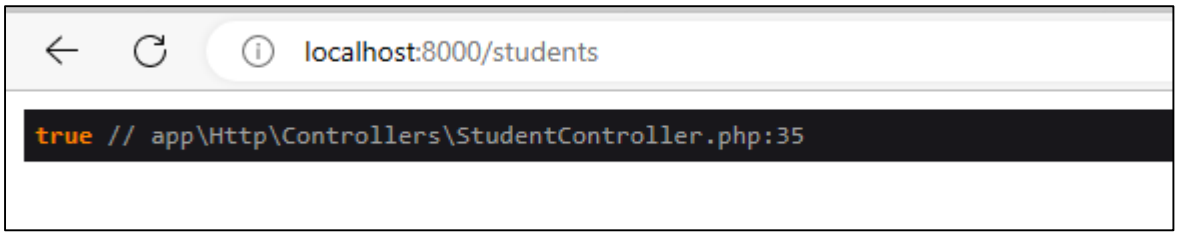
```
<input type="checkbox" name="Fruit"/>Apple
```

//MyController.php

```
$archived = $request->boolean('Fruit');

       dd($archived);
```

//Output



6.Récupération des valeurs d'entrée de date

Le paquetage **Carbon** peut aider à rendre la gestion de la date et de l'heure en PHP beaucoup plus facile et plus sémantique, afin que notre code devienne plus lisible et plus facile à maintenir.

//Form.blade.php

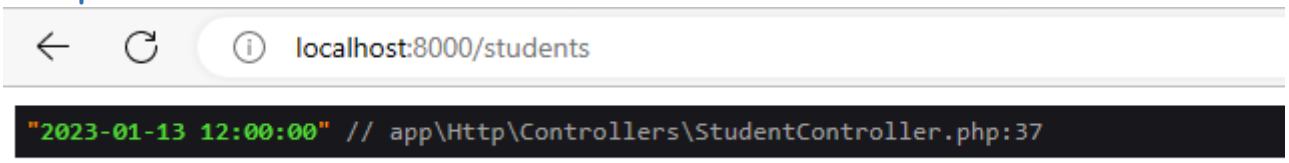
```
<input type="date" name="birthday">
```

//MyController.php

```
        $birthday = Carbon::parse($request->birthday)->format('Y-m-d h:i:s');

        dd($birthday);
```

//Output



7.Récupération d'une partie des données d'entrée

Si vous avez besoin de récupérer un sous-ensemble des données d'entrée, vous pouvez utiliser les méthodes **only** et **except** Ces deux méthodes acceptent un seul array ou une liste dynamique d'arguments :

```
$input = $request->only(['name', 'birthday']);

$input = $request->except(['mail]);
```

8.Déterminer si l'entrée est présente

Vous pouvez utiliser la méthode **has** pour déterminer si une valeur est présente sur la demande. La méthode **has** retourne **true** si la valeur est présente sur la requête :

```
if ($request->has('name')) {
 //
 }
```

Lorsqu'on lui donne un tableau, la méthode has déterminera si toutes les valeurs spécifiées sont présentes :

```
if ($request->has(['name', 'mail']))
 { //
 }
```

La méthode **hasAny** renvoie **true** si l'une des valeurs spécifiées est présente :

```
if ($request->hasAny(['name', 'mail']))
{
//
 }
```

Si vous souhaitez déterminer si une valeur est présente sur la requête et n'est pas vide, vous pouvez utiliser la méthode **filled** :

```
if ($request->filled('name')) {
 //
 }
```

9.Conserver les valeurs des Inputs lors de la session

La méthode **flash** sur la classe **Illuminate\Http\Request** fera clignoter l'entrée actuelle de la session afin qu'elle soit disponible lors de la prochaine requête de l'utilisateur à l'application :

```
$request->flash(); // on conserve tous les inputs
$request->flashOnly(['name', 'mail']); // on sauvgarde seulement les entrées name et mail
$request->flashExcept('password'); // on conserve toutes les valeurs sauf le password
```

10. Entrée clignotante puis redirection

Étant donné que vous souhaiterez souvent flasher l'entrée dans la session, puis rediriger vers la page précédente, vous pouvez facilement enchaîner l'entrée clignotante sur une redirection en utilisant la méthode **withInput** :

```
return redirect('form')->withInput();
return redirect()->route('user.create')->withInput();
return redirect('form')->withInput( $request->except('password') );
```

11. Récupération de l'ancienne entrée

Laravel fournit également un helper old globale. Si vous affichez d'anciennes entrées dans un modèle Blade, il est plus pratique d'utiliser l'assistant **old** pour remplir à nouveau le formulaire. Si aucune ancienne entrée n'existe pour le champ donné, null sera renvoyé :

```
<input  name="mail" value="{{old('mail')}}">
```

12. Création et récupération d'un cookie

Un cookie peut être créé par le helper global **cookie** de Laravel. Il s'agit d'une instance de Symfony\Component\HttpFoundation\Cookie. Le cookie peut être attaché à la réponse à l'aide de la méthode withCookie(). Créez une instance de réponse de la classe Illuminate\Http\Response pour appeler la méthode withCookie(). Les cookies générés par Laravel sont cryptés et signés et ne peuvent pas être modifiés ou lus par le client.

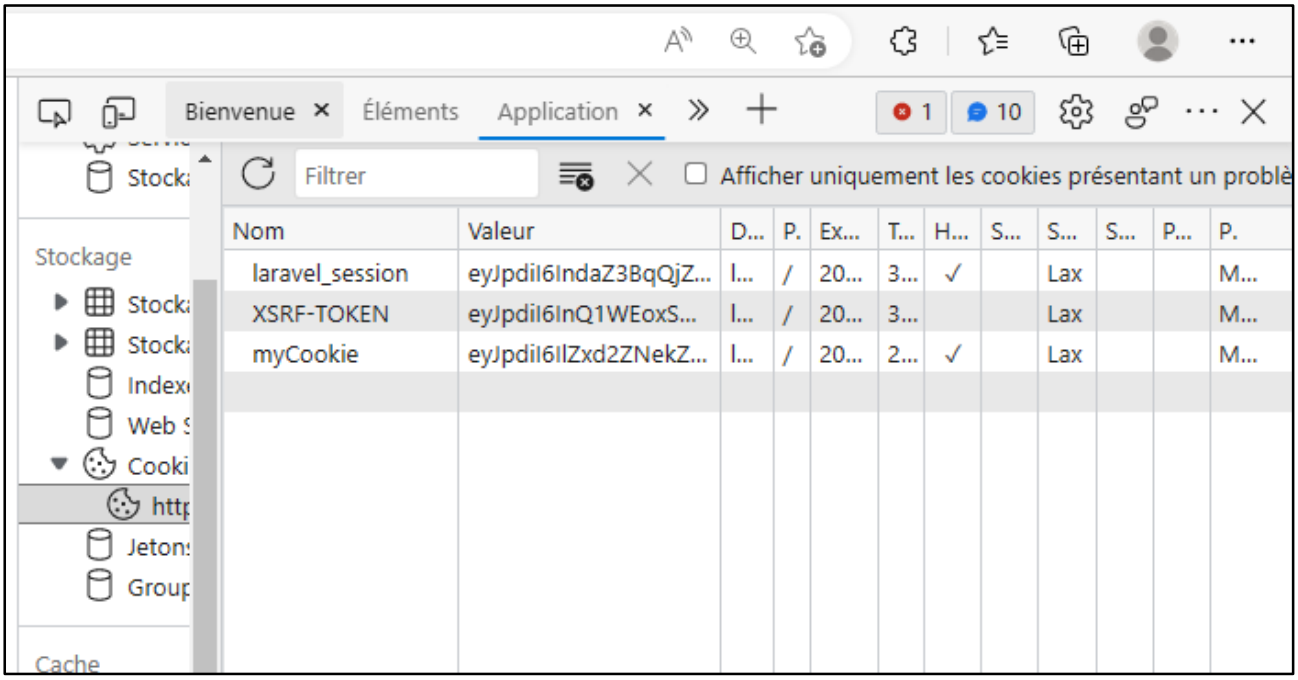
//set cookie

```
return response(view('welcome'))->cookie('myCookie','value',$min);
```

//get cookie

```
    $value = $request->cookie('myCookie');
        dd($value);
```

Vous pouvez vérifier la création de ce cookie dans le navigateur



C. Fichiers

1.Récupération des fichiers téléchargés (uploaded)

Vous pouvez récupérer des fichiers téléchargés à partir d'une instance **Illuminate\Http\Request** à l'aide de la méthode **file** ou à l'aide de propriétés dynamiques. La méthode **file** renvoie une instance de la classe **Illuminate\Http\UploadedFile**, qui étend la classe PHP **SplFileInfo** et fournit une variété de méthodes pour interagir avec le fichier :

```
$file = $request->file('photo');
 $file = $request->photo;
```

Vous pouvez déterminer si un fichier est présent dans la requête en utilisant la méthode **hasFile** :

```
if ($request->hasFile('picture')) {
// }
```

2.Validation des téléchargements réussis

En plus de vérifier si le fichier est présent, vous pouvez vérifier qu'il n'y a eu aucun problème lors du téléchargement du fichier via la méthode **isValid** :

```
if ($request->file('picture')->isValid()) {
 //
 }
```

3.Chemins de fichiers et extensions

La classe **UploadedFile** contient également des méthodes pour accéder au chemin complet du fichier et à son extension. La méthode **extension** tentera de deviner l'extension du fichier en fonction de son contenu. Cette extension peut être différente de l'extension fournie par le client :

```
$path = $request->picture->path();
$extension = $request->picture->extension();
```

4.Stockage des fichiers téléchargés

• Pour stocker un fichier téléchargé, vous utiliserez généralement l'un de vos systèmes de fichiers configurés. La classe **UploadedFile** a une méthode store qui déplacera un fichier téléchargé vers l'un de vos disques, qui peut être un emplacement sur votre système de fichiers local ou un emplacement de stockage en cloud.

• La méthode store accepte le chemin où le fichier doit être stocké par rapport au répertoire racine configuré du système de fichiers. Ce chemin ne doit pas contenir de nom de fichier, car un identifiant unique sera automatiquement généré pour servir de nom de fichier.

• La méthode store accepte également un deuxième argument facultatif pour le nom du disque qui doit être utilisé pour stocker le fichier. La méthode renverra le chemin du fichier par rapport à la racine du disque :

```
$path_image=$request->picture->store('students','public');
```

Vérifiez que le fichier a été bien stocké dans **public/storage/students**