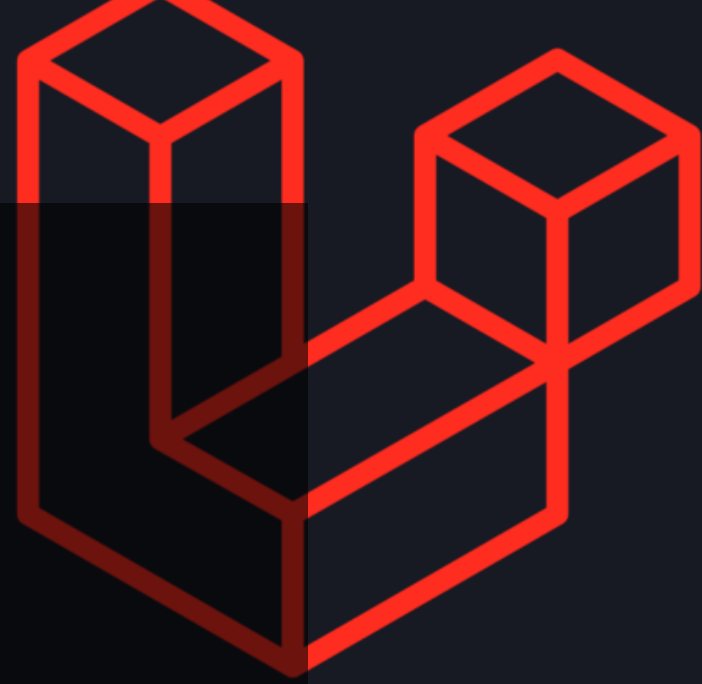


Développer en back-end



Les formulaires

Formatrice : Elidrissi Asmae



Plan du cour

La protection CSRF 01

Manipulation des requêtes http 02

Validation des données d'un formulaire 03

Manipulation des fichiers 04

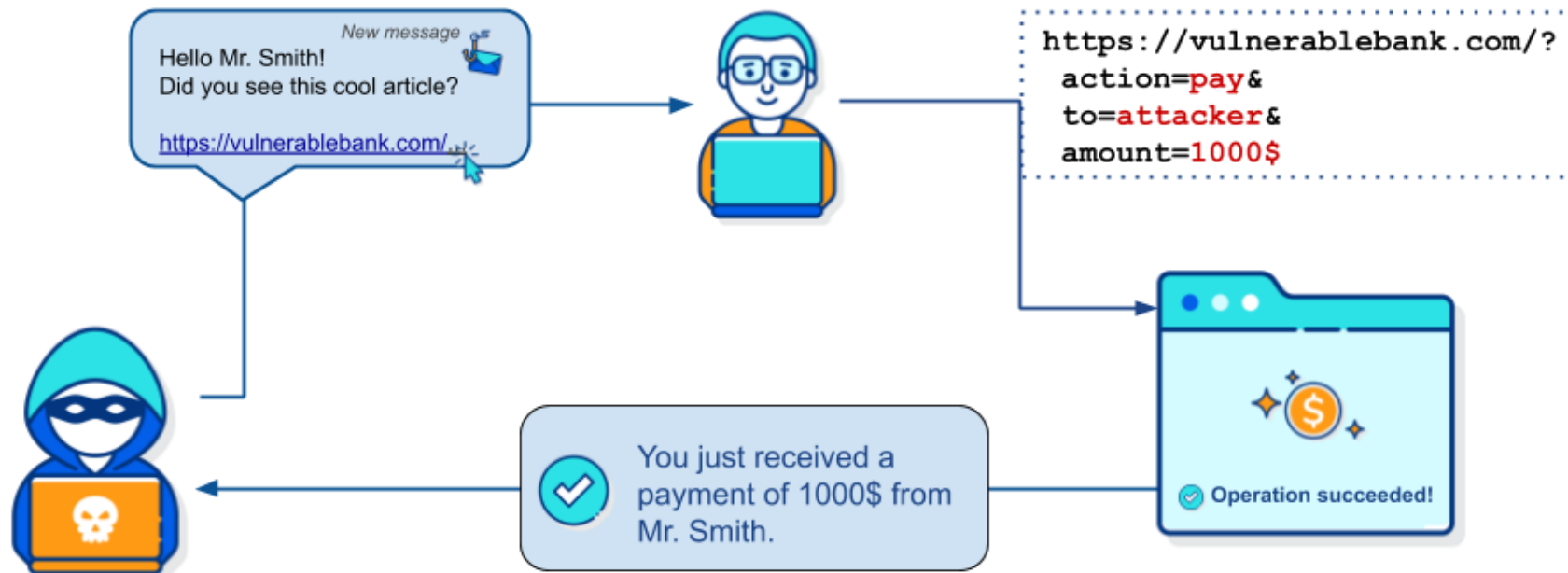




La protection CSRF

La protection CSRF

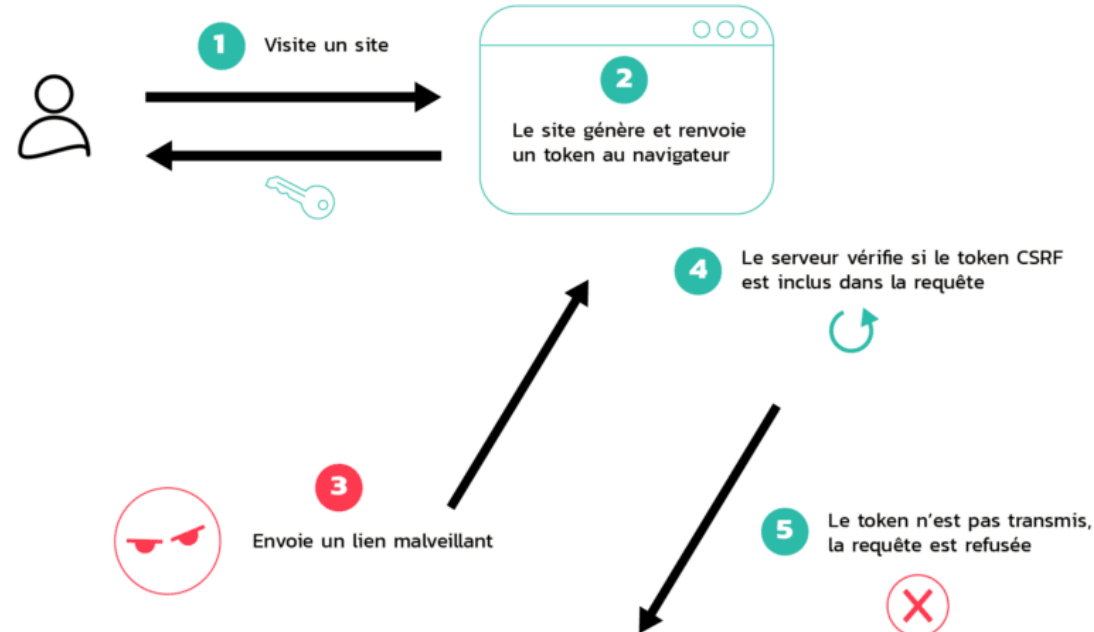
CSRF (**cross-site request forgery**) est synonyme de falsification de requête intersites : est un type d'attaque effectuée par l'attaquant pour envoyer des requêtes à un système à l'aide d'un utilisateur autorisé auquel le système fait confiance. Ce vecteur d'attaque peut être exploité à la fois dans les requêtes POST et GET.



La protection CSRF

- Laravel offre une solution de génération automatique d'un jeton de sécurité.
- Ce jeton CSRF est utilisé pour protéger l'application Web contre les attaques CSRF.
- Ces jetons contiennent une valeur unique générée par le côté serveur de l'application, qui est envoyée au côté client de l'application. En outre, cela permet de vérifier si un utilisateur authentifié envoie la demande à l'application.
- On utilise la directive Blade `@csrf` pour générer le champ token :

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





Manipulation des requêtes http



Manipulation des requêtes http


L'instance Illuminate\Http\Request fournit diverses méthodes pour examiner la requête HTTP entrante.
Voici quelques une de ses méthodes les plus importantes

Méthode	Indications	Exemple
path()	Elle retourne les informations concernant le chemin de la route	Si la requête entrante est appelée depuis : http://127.0.0.1:8000/filiere/create , La méthode \$request->path(); retournera: filiere/create
is() routeIs()	<ul style="list-style-type: none">La méthode is() permet de vérifier si le chemin de la requête entrante correspond à un modèle donné. <i>Vous pouvez utiliser le caractère * comme caractère générique lors de l'utilisation de cette méthode</i>La méthode routeIs() permet de tester les routes nommées. Sur les pages blades, on peut les appeler ainsi: <code>Accueil</code>	<pre>if (\$request->is('filiere/*')) { // ... } if (\$request->routeIs('filiere.*')) { }</pre>
ip()	Elle peut être utilisée pour récupérer l'adresse IP du client qui a fait la requête à l'application	\$ipAddress = \$request->ip();
all()	Vous pouvez récupérer toutes les données d'entrée de la requête entrante en utilisant la méthode all .	\$input = \$request->all();

Manipulation des requêtes http

input()	la méthode input peut être utilisée pour récupérer l'entrée utilisateur	<code>\$name = \$request->input('name');</code>
string()	vous pouvez utiliser la méthode string pour récupérer les données de la requête en tant qu'instance de <u>Illuminate\Support\Stringable</u> ; <i>La classe Stringable permet d'offrir une multitudes de méthodes de manipulation des chaines de caractères.</i>	<code>\$name = \$request->string('name')->trim();</code>
boolean()	La méthode boolean renvoie true pour 1, "1", vrai, "vrai", "on" et "oui". Toutes les autres valeurs renverront false	<code>\$archived = \$request->boolean('archived');</code>
date()	Pour plus de commodité, les valeurs d'entrée contenant des dates/heures peuvent être récupérées en tant qu'instances Carbon à l'aide de la méthode date. Si la requête ne contient pas de valeur d'entrée avec le nom donné, null sera renvoyé	<code>\$birthday = \$request->date('birthday');</code>
[name]()	On peut lire les entrée en appelant les « name » des inputs, qu'on a défini lors de la création du formulaire. On l'appelle propriété dynamique	Soit le champ input suivant : <code><input type="text" name="titre"></code> on peut récupérer sa valeur saisie en écrivant : <code>\$titre = \$request->titre;</code>

Manipulation des requêtes http

has()	Cette méthode peut déterminer si une valeur est présente sur la requête ou non.	<pre>if (\$request->has('name')) { // }</pre>
Ancienne valeurs des inputs <i>Si on utilise les validateurs, la mémorisation des anciennes valeurs se fait automatiquement</i>		
flash()	<p>Elle mémorise les valeurs des inputs dans la session pour qu'elles soient disponibles lors de la prochaine requête de l'utilisateur;</p> <p>On peut, également, demander la mémorisation en redirigeant l'utilisateur vers une route et en appelant la méthode withInput :</p> <pre>return redirect('form')->withInput();</pre>	<pre>\$request->flash();</pre>
Récupération de l'ancienne entrée		
old()	<p>Elle sert à récupérer l'ancienne valeur mémorisée dans la requête</p> <p>On peut utiliser le même mécanisme, si on veut récupérer ces valeurs dans les pages blade :</p> <pre><input type="text" name="username" value="{ old('username') } }"></pre>	<pre>\$username = \$request->old('nom');</pre>
Lecture des valeurs stockées dans les cookies		
cookie('name')	Elle permet d'extraire la valeur stockée dans un cookie	<pre>\$value = \$request->cookie('name');</pre> 

Validation des données d'un formulaire

```
if (count($workerProc) > 1) {  
    $test_files = array_merge($test_files, $sequentialTests);  
    $sequentialTests = [];  
}  
$files = [];  
$maxBatchSize = $valgrind > 1 ? ($batchSize * 4) : $batchSize;  
$leverageFilesPerWorker = max(1, ($batchSize * 4) / $valgrind);  
$batchSize = min($maxBatchSize, $leverageFilesPerWorker);  
while (count($files) <= $batchSize) {  
    foreach ($fileConflicts as $file) {  
        if (isset($activeConflicts[$conflictkey]) && $waitingTests[$conflictkey] > 0) {  
            continue 2;  
        }  
        $files[] = $file;  
    }  
    if (count($files) <= $batchSize) {  
        foreach ($files as $file) {  
            $workerProc[] = $file;  
        }  
    }  
}
```



Validation des données d'un formulaire

- La validation de formulaire est un fait très important pour assainir et protéger les données indésirables dans notre application.
- Laravel propose plusieurs méthodes pour valider les données entrantes de l'application :
- Dans ce cours, on traitera deux méthodes :
 1. La validation de la requête (*Méthode valide de la classe `Illuminate\Http\Request`*)
 2. Validation du modèle (*Form Request*)

Pour plus de méthodes consultez la documentation ici : <https://laravel.com/docs/10.x/validation#quick-writing-the-validation-logic>

1. La validation de la requête « Request »

Prenons l'exemple d'ajout d'un nouveau stagiaire avec la méthode **store** :

```
public function store(Request $request)
{
    // code de validation
    Stagiaire::create($request->all());
    return redirect()->route('stagiaire.create')->with('success', 'Le stagiaire a été ajouté avec succès.');
```

On écrit ici la logique de la validation:

- Si toutes les règles sont respectées, on exécute le code suivant;
- Sinon, une exception de type *Illuminate\Validation\ValidationException* est levée;
- Une réponse avec l'erreur détectée va être envoyée automatiquement à l'utilisateur

Validation des données d'un formulaire

Exemple de validation des données d'un stagiaire avant sa création :

```
public function store(Request $request)
{
    $request->validate([
        'nom_complet' => 'required|regex:/^[A-Za-z\s]+$/ ',
        'date_naissance' => 'required|date',
        'note' => 'required|numeric|between:0,20',
        'genre'=>'in:M,F'
    ]);

    Stagiaire::create($request->all());
    return redirect()->route('stagiaire.create')->with('success','Le stagiaire a été ajouté avec succès.');
```

- `'nom_complet' => 'required|regex:/^[A-Za-z\s]+$/ '` : indique que le **nom complet** est obligatoire et ne doit contenir que des lettres (majuscules et/ou minuscules) et/ou des espaces;
- `'date_naissance' => 'required|date'` : indique que la **date de naissance** est obligatoire et elle doit être une date valide;
- `'note' => 'required|numeric|between:0,20'` : indique que la **note** est obligatoire et elle doit être un numérique compris entre 0 et 20;
- `'genre'=>'in:M,F'` : indique que le champ **genre** ne peut contenir que deux valeurs possibles F ou bien M;

Validation des données d'un formulaire

2. La validation du modèle Form Request:

Si on souhaite extraire la logique de validation des contrôleurs, ou si on souhaite effectuer une autorisation et une validation en même temps, Laravel met à notre disposition la classe **Form Request**.

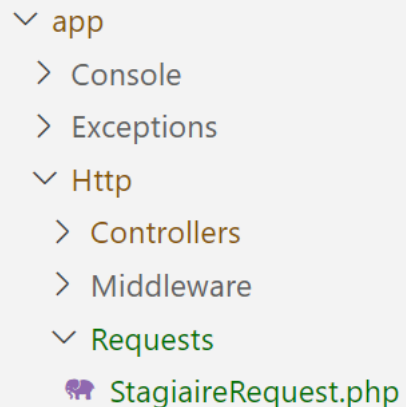
La validation sera codée dans une classe dérivée de `FormRequest`. Cette classe peut être générée à l'aide d'une commande artisan.

Exemple:

▪ `php artisan make:request StagiaireRequest`

Cette commande permet de créer un fichier `StagiaireRequest.php`, sous le répertoire `app\Http\Requests`,

Le code de la classe créée est présenté ci-après :



```

└─ app
  └─ > Console
  └─ > Exceptions
  └─ > Http
    └─ > Controllers
    └─ > Middleware
    └─ > Requests
      └─ 🐘 StagiaireRequest.php

```

```

namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;

class StagiaireRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this
     * request.
     */
    public function authorize(): bool
    {
        return false;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string,
     * \Illuminate\Contracts\Validation\Rule|array|string>
     */
    public function rules(): array
    {
        return [
            //
        ];
    }
}

```

Validation des données d'un formulaire

2. La validation du modèle Form Request:

La méthode authorize()

La classe dérivée de FormRequest permet, dans un premier temps, de valider qui a le droit d'accéder au formulaire. C'est le rôle de la méthode **authorize()**.

Puisque Laravel offre d'autres mécanismes pour contrôler les droits d'accès, nous n'utiliserons pas **authorize()**.

Modifiez donc la méthode authorize() pour qu'elle retourne **true**.

-> Si vous laissez la valeur false, lorsque vous soumettrez le formulaire, vous obtiendrez une page d'erreur 403:

A screenshot of a 403 error message displayed on a light blue background. The text "403" is followed by a vertical line and then "THIS ACTION IS UNAUTHORIZED." in a light blue font.

403 | THIS ACTION IS UNAUTHORIZED.

Validation des données d'un formulaire

2. La validation du modèle Form Request:

La méthode rules()

Dans cette méthode, on retourne la liste des règles que nous voulons appliquer sur le modèle avant qu'il soit ajouté ou modifié dans la base de données.

Exemple:

```
public function rules(): array
{
    return [
        'nom_complet' => 'required|regex:/^[A-Za-z\s]+$/ ',
        'date_naissance' => 'required|date',
        'note' => 'required|numeric|between:0,20',
        'genre'=>'in:M,F'
    ];
}
```

Exécution automatique de la validation

Pour que Laravel exécute automatiquement la validation, la méthode d'enregistrement des données « **store** » (ou **update**) devra recevoir en paramètre une instance de cette classe.

```
public function store(StagiaireRequest $request)
{
    Stagiaire::create($request->all());
    return redirect()->route('stagiaire.create')->with('success','Le stagiaire a été ajouté avec succès.');
```


Validation des données d'un formulaire

Affichage des erreurs :

- Lorsqu'une validation est effectuée, les validations qui ne sont pas réussies se chargent d'ajouter un message significatif dans la variable **\$errors**.
- Cette variable est automatiquement disponible pour **toutes les vues**.
- Les vues qui contiennent un formulaire à valider pourront afficher tous les messages en bouclant sur la variable **\$errors**.

```
@if ($errors->any())  
    <div class="alert alert-danger">  
        <ul>  
            @foreach ($errors->all() as $error)  
                <li>{{ $error }}</li>  
            @endforeach  
        </ul>  
    </div>  
@endif
```

- The nom complet field format is invalid.
- The note field must be between 0 and 20.

Nouveau stagiaire

Nom complet

Genre : ☐ F ☒ M

Date de naissance

Note

Groupe

Validation des données d'un formulaire

Affichage des erreurs :

Si on veut afficher une erreur spécifique à un champ, on peut utiliser la directive: `@error('nom_champ')`

Elle permet de tester si le champ indiqué contient une erreur ou pas.

Exemple:

```
<div class="mb-3">
  <label for="nom_complet" class="form-label">Nom complet</label>
  <input type="text" class="form-control" id="nom_complet"
name="nom_complet" value="{{ old('nom_complet') }}" />
```

```
@error('nom_complet')
  <div class="alert alert-danger">{{ $message }}</div>
@enderror
</div>
```

```
.....
<div class="mb-3">
  <label for="note" class="form-label">Note</label>
  <input type="text" class="form-control @error('note') is-invalid
@enderror" id="note" name="note" value="{{ old('note') }}" />
```

```
@error('note')
  <div class="alert alert-danger">{{ $message }}</div>
@enderror
</div>
```

Nouveau stagiaire

Nom complet

The nom complet field is required.

Genre : ☐ F ☐ M

Date de naissance

Note

The note field must be between 0 and 20.

Groupe

Submit

Validation des données d'un formulaire

Méthode 1: La validation de la requête « Request »

la fonction `validate` prend un tableau de messages en deuxième paramètre

Exemple:

```
$request->validate([
    'nom_complet'=>'required|max:20',
    'date_naissance'=>'required|date',
], [
    'nom_complet.required'=>"nom obligatoire!"
]);
```

Nouveau stagiaire

Nom complet

nom obligatoire!

Méthode 2: FormRequest

On ajoute à la classe dérivée de ***formRequest*** une méthode ***messages()***, où on retourne un tableau des attributs et leurs messages correspondants :

Exemple:

```
public function messages()
{
    return [
        'nom_complet.required' => 'Vous devez fournir votre nom complet',
        'nom_complet.regex' => 'Nom incorrect',
        'note.between' => 'la note doit être comprise entre 0 et 20',
    ];
}
```

- Vous devez fournir votre nom complet
- la note doit être comprise entre 0 et 20

Validation des données d'un formulaire

Les règles de validation

Règle	Indication
accepted	Le champ sous validation doit être yes, on, ou 1.
after:date	Le champ sous validation doit être après une date donnée.
alpha	Le champ sous validation peut uniquement contenir des lettres.
alpha_num	Le champ sous validation peut uniquement contenir des caractères alpha-numériques.
array	Le champ sous validation doit être un tableau PHP.
before:date	Le champ sous validation doit être une date avant la date donnée.
between:min,max	Le champ sous validation doit avoir une valeur entre min et max.
boolean	Le champ sous validation doit pouvoir être un booléen.
date	Le champ sous validation doit être une date valide selon la fonction PHP strtotime.
email	Le champ sous validation doit être une adresse e-mail correcte.
exists:table,column	Le champ sous validation doit exister dans la base de données.
image	Le fichier sous validation doit être une image (jpeg, png, bmp, ou gif).
in:foo,bar,...	Le champ sous validation doit être inclus dans la liste donnée de valeurs.
integer	Le champ sous validation doit être un entier.

Validation des données d'un formulaire

Les règles de validation (<https://laravel.com/docs/10.x/validation#available-validation-rules>)

nullable	Le champ sous validation peut être null.
max	Le champ sous validation doit avoir au maximum la valeur indiquée
min	Le champ sous validation doit avoir au minimum la valeur indiquée s'il s'agit d'un champ numérique et au minimum la taille indiquée s'il s'agit d'un string, array
numeric	Le champ sous validation doit être numérique .
password	Le champ sous validation doit correspondre au mot de passe de l'utilisateur authentifié.
regex:pattern	Le filtre sous validation doit correspondre à l'expression régulière donnée.
required	Le champ sous validation doit être présent dans les données.
size:value	Le champ sous validation doit avoir une taille correspondant à la valeur value.
string	Le champ sous validation doit être une chaîne de caractères.
unique:table,column	Le champ sous validation doit être unique dans la table de la base de donnée.
url	Le champ sous validation doit être formé comme une URL.

Validation des données d'un formulaire

Cas de mise à jour avec la règle Unique:

- Parfois, vous souhaitez peut-être ignorer un identifiant donné lors de la validation unique.
- Par exemple, considérez une interface de mise à jour d'un stagiaire qui inclut le nom et l'adresse e-mail du stagiaire. Vous voudrez probablement vérifier que l'adresse e-mail est unique. Cependant, si l'utilisateur modifie uniquement le champ du nom et non pas le champ de l'e-mail, vous ne souhaitez pas qu'une erreur de validation soit générée car le stagiaire est déjà le propriétaire de l'adresse e-mail en question.
- Pour demander au validateur d'ignorer l'id du stagiaire, on peut utiliser l'une des méthodes suivantes:

-> Dans la méthode update du contrôleur:

```
public function update(Request $request, Stagiaire $stagiaire)
{
    $request->validate( [
        "nom_complet"=> "required|unique:stagiaires,nom_complet, ".$stagiaire->id
    ]);
    /////
```

-> Dans le fichier StagiaireRequest :

```
public function rules(): array
{

    return [
        "nom_complet"=>["required",
            Rule::unique('stagiaires')->ignore($this->stagiaire)
        ]
    ];
    ////
```



Manipulation des fichiers



Manipulation des fichiers

Afin de manipuler les fichiers, on doit, tout d'abord créer un formulaire permettant de recevoir le fichier à téléverser:

```
<form action="{{route('filiere.store')}}" method="post" enctype="multipart/form-data" >
    @csrf
    .....
    <div class="mb-3">
        <label for="photo" class="form-label">Photo</label>
        <input type="file" class="form-control" name="monfichier" id="photo"/>
    </div>
    <button type="submit" class="btn btn-primary">Ajouter</button>
</form>
```

Coté serveur, on peut récupérer le fichier choisi en utilisant la méthode **file** ou en utilisant le **name** de son input (comme propriété dynamique)

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

Ou

```
$file = $request->monfichier;
```

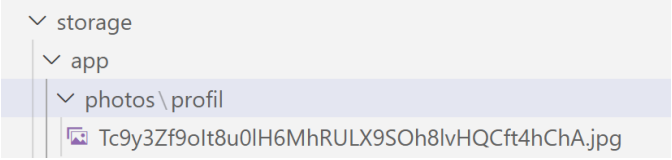
L'objet file récupéré est une instance de la classe **UploadedFile**.

Manipulation des fichiers

La classe ***UploadedFile*** offre une multitude de méthodes permettant l'interaction avec les fichiers téléversés, voici quelques une :

Informations		
isValid()	Elle vérifie s'il n'y a eu aucun problème lors du téléchargement du fichier.	<code>\$request->monfichier->isValid()</code>
path()	Elle permet d'accéder au chemin temporaire complet du fichier	<code>\$request->monfichier->path();</code> <i>Ex.de résultat :</i> <code>C:\xampp\tmp\php70C0.tmp</code>
extension()	Elle permet de deviner l'extension du fichier en fonction de son contenu. Cette extension peut être différente de l'extension fournie par le client	<code>\$request->monfichier->extension();</code>

Manipulation des fichiers

Sauvegarde		
store()	Elle permet de déplacer un fichier téléchargé vers l'un de vos disques, qui peut être un emplacement sur votre système de fichiers local ou un emplacement sur Cloud.	<pre>\$request->monfichier->store('photos/profil');</pre>  <p>Le fichier téléversé sera stocké sous le dossier « storage/photos/profil » du projet Laravel en lui générant et en lui attribuant un identifiant unique.</p> <p>-> L'emplacement de stockage peut être configuré via le fichier « filesystems.php » placé sous le dossier « config »</p>
move()	Si on désire stocker le fichier dans un autre emplacement	<pre>\$request->monfichier->move(public_path('images'), \$imageName);</pre> <pre>\$path= \$request->file("photo")->move("E:\Mes images", "photo_23.jpg");</pre>
storeAs()	Pour éviter la génération automatique du nom de fichier à stocker, on peut utiliser la méthode storeAs	<pre>\$path = \$request->monfichier->storeAs('images', 'filename.jpg');</pre>

Manipulation des fichiers

Lecture des fichiers stockés:

Rendre le dossier publique:

Pour rendre un dossier publique et accessible, il suffit de l'enregistrer dans le fichier de configuration, placé sous « config/filesystems.php »

```
'links' => [  
    public_path('storage') => storage_path('app/public'),  
    public_path('photos') => storage_path('app/photos'),  
]
```

Ces deux lignes signifient :

- Le répertoire « storage/app/public » sera accessible au public et on pourra l'appeler via :
`echo asset('storage/file.txt');`
- Le répertoire « storage/app/photos » sera accessible au public et on pourra l'appeler via :
`echo asset('photos/file.txt');`

Pour créer ces deux liens, il faut exécuter la commande artisan suivante:

```
php artisan storage:link
```

Manipulation des fichiers

Exemple:

- Ajouter une colonne « **photo** » à la table « **stagiaires** » (créer et exécuter un fichier de migration);
- Ajouter, au formulaire de création, le champ de téléversement de la photo;
- Modifier la méthode store pour qu'elle prenne en charge la photo téléversée :

```
public function store(Request $request)                                $request->photo->storeAs('photos', $nomPhoto);
{
    $request->validate([
        'nom_complet' => 'required|regex:/^[A-Za-z\s]+$/ ',
        'date_naissance' => 'required|date',
        'note' => 'required|numeric|between:0,20',
        'genre' => 'in:M,F',
        'photo' =>
        'image|mimes:png,jpg,jpeg|max:8000'
    ]);

    $nomPhoto=null;
    if(isset($request->photo)){
        //Génération d'un nom de photo en se
        //basant sur le time actuel
        $nomPhoto = time().'.'.$request->photo-
        >extension();
        //Stockage dans le dossier "Stockage"

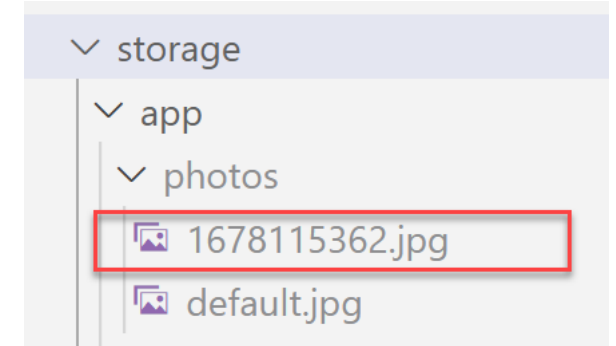
        Stagiaire::create([
            "nom_complet"=>$request->nom_complet,
            "genre"=>$request->genre,
            "date_naissance"=>$request-
            >date_naissance,
            "note"=>$request->note,
            "groupe_id"=>$request->groupe_id,
            "photo"=>$nomPhoto
        ]);

        return redirect()-
        >route('stagiaire.create')->with('success','Le
        stagiaire a été ajouté avec succès.');
```

Manipulation des fichiers

Exemple:

- Les photos téléversées seront stockées sous le chemin « **storage/app/photos** »
- Ajouter, à ce répertoire, une image « default.jpg » pour qu'elle soit affichée dans le cas où le stagiaire n'a pas fourni de photo.
- Ajouter à la page index.blade.php le code permettant la lecture des photos des stagiaires :



```
@isset($stagiaires)
<table class="table mt-3">
  <tr>
    <th>Photo</th>
    <th>Id</th>
    <th>Nom complet</th>
    <th>Genre</th>
    <th>Date de naissance</th>
    <th>Note</th>
    <th>Groupe</th>
    <th>Action</th>
  </tr>
  @foreach($stagiaires as $stagiaire)
    <tr>
      <td>
```

```
      @if(!empty($stagiaire->photo ))
        
      @else
        
      @endif
    </td>
    <td>{{ $stagiaire->id }}</td>
    <!-- reste du code -->
  </tr>
  @endforeach
</table>
```

Manipulation des fichiers

Exemple:

Le résultat se présentera ainsi :

Liste des stagiaires

Nouveau stagiaire

Photo	Id	Nom complet	Genre	Date de naissance	Note	Groupe	Action
	1	Tazi Naoual	F	12/03/2002	18.00	1	Afficher Modifier Supprimer
	3	Ali Alami	M	06/12/2001	14.00	2	Afficher Modifier Supprimer
	5	Alaoui Souad	F	18/02/2003	15.00	1	Afficher Modifier Supprimer
	6	Salma Alami	F	25/02/2000	14.00	1	Afficher Modifier Supprimer

Le reste du code de cet exemple est accessible via le lien suivant :
https://gitlab.com/devowfs_adarissa/exemple_crud_elloquent_file.git


```
if (count($workerProc) > 1) {  
    $test_files = array_merge($test_files, $sequentialTests);  
    $sequentialTests = [];  
}  
$files = [];  
$maxBatchSize = $valgrind ? 1 : ($batchSize < 1 ? 1 : $batchSize);  
$leverageFilesPerWorker = max(1, ($batchSize < 1 ? 1 : $batchSize));  
$batchSize = min($maxBatchSize, $leverageFilesPerWorker);  
while (count($files) <= $batchSize) {  
    foreach ($fileConflictsWith as $file) {  
        if (isset($activeConflicts[$conflictKey]) && $activeConflicts[$conflictKey] == $file) {  
            continue 2;  
        }  
        $files[] = $file;  
    }  
    foreach ($files as $file) {  
        foreach ($fileConflictsWith as $conflictKey) {  
            if (isset($activeConflicts[$conflictKey]) && $activeConflicts[$conflictKey] == $file) {  
                continue 2;  
            }  
        }  
    }  
}
```

TP



A decorative header bar with a dark blue background. It features several 3D wireframe cubes. On the left, there are faint, thin-lined cubes. On the right, there are two more prominent cubes outlined in a bright red color.

À la prochaine !