Module: Développer en back-end

6. ORM Eloquent- partie 1

Filière: Développement digital – option web full stack

Cours réalisé par : Madame Asmae YOUALA

ISTA AL ADARISSA

PLAN DU COURS:

L’ORM Eloquent - partie 1:

a. **Présentation**
b. **Création et configuration des modèles**
c. **Requêtes de mise à jour**

l'ORM Eloquent : Présentation

- Un **ORM** (Object-Relational Mapping) est un logiciel permettant la conversion des données relationnelles d'une base de données en objets afin de pouvoir les manipuler dans notre application en POO (Programmation Orientée Objet).
- **Eloquent** est le nom de l'ORM utilisé par Laravel.
- L'ORM Eloquent est utilisé pour établir des relations avec des tables de base de données. Chaque table de la base de données est mappée avec un modèle éloquent particulier.
- L'objet modèle contient diverses méthodes pour récupérer et mettre à jour les données de la table de base de données. Il fonctionne en encapsulant les attributs d'une table ou d'une vue dans une Class appelé ***Modèle***.
- Eloquent ORM rend facile l'exécution des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur le modèle Laravel.

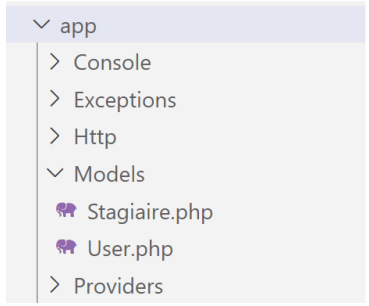
l'ORM Eloquent : Création des modèles

- Pour chaque table de la base de données doit correspondre un Model qui sera utilisé pour interagir avec elle.
- Pour créer un modèle, il suffit d'exécuter la commande **php artisan make:model** suivie du nom du modèle à produire.

Pour suivre les standards de Laravel, le nom du modèle doit débuter par une majuscule et être au singulier.

```
php artisan make:model Stagiaire
```

- Cette commande permet la création d'un fichier nommé ***Stagiaire.php*** placé directement dans le répertoire ***app/Models*** :




```php
namespace App\Models;

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

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

- Le modèle Stagiaire est automatiquement lié à la table stagiaires : Laravel saura faire ce lien si le nom de la table est exactement le nom du modèle, tout en lettre minuscules avec en plus un **s** final.

l'ORM Eloquent : Création des modèles

- On peut générer divers autres types de classes lors de la génération d'un modèle, telles que les fabriques, des seeders, les contrôleurs et les requests. De plus, ces options peuvent être combinées pour créer plusieurs classes à la fois;

  Exemple:

  ```
  php artisan make:model Filiere --controller –resource –seed --migration
  ```

  ou

  ```
  php artisan make:model Filiere –crsm
  ```

  L'exécution de cette commande a permis de créer :

  - un modèle (***Filiere.php***)
  - un contrôleur de type ressource (***FiliereController.php***)
  - un fichier seed (***FiliereSeeder***)
  - un fichier migration avec le nom (***2023_02_17_152826_create_filieres_table.php***)

- Dès que le modèle est créé, même si la classe est vide, on peut l'exploiter et l'utiliser dans nos requêtes.

- Notez qu'il est rarement utile de créer un modèle pour **les tables pivot** (tables intermédiaires dans les relations de **plusieurs à plusieurs**) et ce, même si la table pivot contient des informations autres que les clés étrangères. Eloquent vous offrira d'autres alternatives pour accéder à ces informations.

l'ORM Eloquent : Configuration des modèles

- Laravel donne, ainsi la possibilité de paramétrer les modèles en ajoutant quelques instructions à la classe Modèle :

Instruction	Signification
`protected $table = 'nom_de_table';`	Personnalise **le nom de table** correspondante à ce modèle
`protected $primaryKey = 'ma_cle_primaire';`	Personnalise **le nom de la clé primaire** (un nom différent de 'id')
`public $incrementing = false;`	Indique que **la colonne PRIMARY KEY** n'est pas en auto-incrémente
`protected $keyType = 'string';`	Indique un autre **type de la colonne** clé primaire (par défaut INT)
`public $timestamps = false;`	Par défaut, Eloquent s'attend à ce que les colonnes **created_at et updated_at** existent dans la table de base de données et définira automatiquement leurs valeurs. Cette instruction indique que **nous ne souhaitons pas que ces colonnes soient automatiquement gérées** par Eloquent
`const CREATED_AT = 'creation_date';` `const UPDATED_AT = 'updated_date';`	Personnalise **les noms des colonnes created_at et updated_at**;

l'ORM Eloquent : Configuration des modèles

Traitement des champs en lot (mass assignment) : $fillable et $guarded

Laravel offre un mécanisme pour faciliter l'enregistrement d'informations dans la base de données à partir d'un tableau associatif, par exemple **$request->all().** On dira qu'on effectue un traitement des champs en lot. En anglais, on utilisera le terme ***mass assignment***.

```
public function store(Request $request)
    {
        // $request->all() est un tableau associatif de toutes les valeurs saisies dans le formulaire
    $utilisateur = new Utilisateur($request->all());
    .....
    $utilisateur->save();
    ....
    }
```

Si on ne prend pas quelques précautions, un tel traitement pourrait ouvrir la porte à des **vulnérabilités**.
Par exemple, si on a un champ dans la table ***utilisateur*** qui donne le statut d'administrateur, un usager malveillant pourrait trafiquer le formulaire d'ajout des utilisateurs pour lui ajouter un champ qui porte le nom attendu par votre modèle (ex. : is_admin) et lui donner la valeur true .
Le traitement des champs en lot récupérerait l'ensemble des champs du formulaire et les passerait à la méthode de traitement en lot. Notre usager serait alors promu administrateur!

Heureusement, si vous tentez un traitement des champs en lot sur un modèle dans lequel vous n'avez pas effectué les configurations nécessaires, vous obtiendrez une erreur du genre « ***MassAssignmentException in Model.php _token*** ».

l'ORM Eloquent : Configuration des modèles

Traitement des champs en lot (mass assignment) : $fillable et $guarded

L'erreur de « **mass assignement** » est une sécurité de Laravel par défaut lorsque nous utilisons des tableaux PHP pour spécifier notre attributs.

La liste blanche :

Laravel exige, ainsi, que nous indiquions explicitement les colonnes autorisées à être remplies de cette manière.
Pour indiquer les colonnes **autorisées** à Laravel, nous devons ajouter à notre modèle **Utilisateur**, un attribut **$fillable** (« **remplissable** » en français).

Dans les faits, l'instruction **`$utilisateur = new Utilisateur($request->all());`** indique à Laravel qu'il doit automatiquement faire le lien entre l'attribut **name** des balises HTML de saisie (présentes dans la variable $request) et les champs listés dans **$fillable**.

```
protected $fillable= [
    'nomfamille',
    'prenom',
    'telephone',
    'courriel',
    'bureau',
];
```

l'ORM Eloquent : Configuration des modèles

Traitement des champs en lot (mass assignment) : $fillable et $guarded

La liste noire :

Plutôt que de faire une liste blanche des champs pouvant être traités en lot, il est possible de tenir une liste noire des champs protégés.

À ce moment, l'instruction **$utilisateur = new Utilisateur($request->all());** indique à Laravel qu'il doit automatiquement faire le lien entre l'attribut **name** des balises HTML de saisie (présentes dans la variable $request) et les champs qui ne sont pas listés dans **$guarded**.

Ex :

```
/**
  * Liste de champs ne pouvant pas être assignés en lot (mass assignment).
  * Devront être assignés explicitement à l'aide d'une instruction du genre
  * $objet->champ = $valeur;
  * $objet->save();
  * @var array
  */
    protected $guarded = ['is_admin'];
```

Si on souhaite rendre tous les attributs assignables en masse, on peut définir la propriété ***$guarded*** comme un tableau vide.

```
    protected $guarded = [];
```

Le modèle doit avoir une propriété $fillable ou une propriété $guarded mais pas les deux.

l'ORM Eloquent : Les requêtes de mise à jour

Méthode	Indications	Exemple
Insertion		
save	Il faut créer une instance d'un modèle et puis appeler la méthode « save » pour insérer la ligne dans la table correspondante.	`$filiere = new Filiere([ "titre"=>$request->titre, "description"=>$request->description ]); $filiere->save();`
create	Cette méthode permet de créer et de retourner le modèle enregistré dans la table.	`$filiere = Filiere::create([ "titre"=>$request->titre, "description"=>$request->description ]);`
Mise à jour		
save	Il faut avoir une instance du modèle, modifier ses paramètres et appler la méthode save pour enregistrer les mises à jour	`$filiere = Filiere::find(1); $filiere->titre = 'Développement digital - tronc commun'; $filiere->save();`
update	Les mises à jour peuvent également être effectuées sur des modèles qui correspondent à une requête donnée.	`Groupe::where('libelle', 'LIKE', 'DEVOWFS%') ->update(['filiere_id' => 2 ]);`

l'ORM Eloquent : Les requêtes de mise à jour

<table>
<tr><th colspan="3">Insertion ou mise à jour (upserts)</th></tr>
<tr><td>updateOrCreate</td><td>mettre à jour un modèle existant ou créer un nouveau modèle s'il n'existe aucun modèle correspondant.</td><td><pre>$stagiaire = Stagiaire::updateOrCreate(
["nom_complet"=>"Ahmed Alami" , "groupe_id"=>"1", "ville"=>"Fès"]
,['note' => 15]);</pre>On cherche le stagiaire « Alami Ahmed » du groupe « 1 » qui habite à Fès, s'il existe, on attribue à sa note 15, sinon on crée une nouvelle instance et on l'insère dans la table stagiaires</td></tr>
<tr><td>upsert</td><td>S'il s'agit de traitement de plusieurs lignes, on peut utiliser la méthode « upsert ».
Le premier argument de la méthode se compose des valeurs à insérer ou à mettre à jour
Le deuxième argument répertorie la ou les colonnes qui identifient de manière unique les enregistrements dans la table associée.
Le troisième argument de la méthode est un tableau des colonnes qui doivent être mises à jour si un enregistrement correspondant existe déjà dans la table en question.</td><td><pre>Stagiaire::upsert(
[["id"=>1,"nom_complet"=>"Ahmed Alami", "groupe_id"=>"1", "ville"=>"Fès", "note"=>18],
["id"=>8,"nom_complet"=>"Sara Alaoui", "groupe_id"=>"1", "ville"=>"Meknès", "note"=>17]]
,["id"],['note']
);</pre></td></tr>
</table>

l'ORM Eloquent : Les requêtes de mise à jour

<table>
<tr><th colspan="3">Suppression</th></tr>
<tr><td>delete</td><td>Supprime une ligne ou plusieurs lignes de la base de données en se basant sur une instance du modèle ou sur un résultat d'une requête.</td><td><pre>$filiere = Filiere::find(1);
$filiere->delete();

$deleted= Groupe::where('année', '<',
2000)->delete();</pre></td></tr>
<tr><td>truncate</td><td>Supprime tous les enregistrements de la table et réinitialise les ids en auto-incrément</td><td><pre>Filiere::truncate();</pre></td></tr>
<tr><td>destroy</td><td>Supprime une ou plusieurs lignes en se basant sur leur clés primaires.</td><td><pre>Filiere::destroy(1);
Filiere::destroy(1, 2, 3);</pre></td></tr>
<tr><th colspan="3">Duplication d'un modèle</th></tr>
<tr><td>replicate</td><td>crée une copie non enregistrée d'une instance de modèle existant.
Cette méthode est particulièrement utile lorsque vous avez des instances de modèle qui partagent plusieurs mêmes attributs</td><td><pre>$livraison = Address::create([
    'type' => 'livraison',
    'line_1' => '123 Example avenue',
    'ville' => 'Fès',
    'pays' => 'Maroc',
]);
$adresse_facturation = $livraison->replicate()->fill([
     'type' => 'facturation'
]);
$adresse_facturation->save();</pre></td></tr>
</table>

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour d'un modèle stagiaire

1. Configuration:

- Créons le modèle et le contrôleur de ressource en lançant cette commande:

```
php artisan make:model Stagiaire –cr
```

Cette commande créera une classe Modèle « **Stagiaire** » et un contrôleur de ressource « **StagiaireController** »

- Ajoutons au **modèle « Stagiaire »** la liste des champs autorisés pour l'affectation en masse :

```
protected $fillable= ["nom_complet","genre", "date_naissance", "note","groupe_id"];
```

- Ajoutons à la page **web.php** l'instruction permettant de créer les différentes **routes** exploitant les méthodes du contrôleur « **StagiaireController** »:

```
Route::resource('stagiaire', StagiaireController::class);
```

On peut vérifier les routes créées en exécutant la commande : `php artisan route:list`

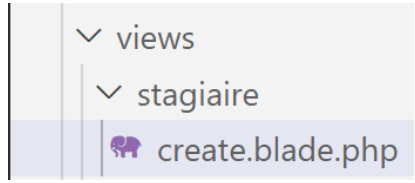
```
GET|HEAD        stagiaire ..................stagiaire.index › StagiaireController@index
POST            stagiaire ..................stagiaire.store › StagiaireController@store
GET|HEAD        stagiaire/create ..............stagiaire.create › StagiaireController@create
GET|HEAD        stagiaire/{stagiaire ..............stagiaire.show › StagiaireController@show
PUT|PATCH       stagiaire/{stagiaire ..............stagiaire.update › StagiaireController@update
DELETE          stagiaire/{stagiaire} ..............stagiaire.destroy › StagiaireController@destroy
GET|HEAD        stagiaire/{stagiaire}/edit .........stagiaire.edit › StagiaireController@edit
```

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire

2. Création d'un nouveau stagiaire:

- Sous le dossier « views », créer un sous dossier appelé « stagiaire ».
- Sous le dossier « stagiaire », ajouter une nouvelle vue « **create.blade.php** »




```
<h2>Nouveau stagiaire</h2>
<form class="col-8" method="post" action="{{route('stagiaire.store')}}">
  @csrf
 <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"/>
  </div>
    <div class="mb-3">
        <label for="genre" class="form-label">Genre : </label>
        <div class="form-check form-check-inline">
            <input class="form-check-input " type="radio" name="genre"
id="genre_f" value="F"/>
            <label class="form-check-label" for="genre_f"> F </label>
        </div>
         <div class="form-check form-check-inline">
            <input class="form-check-input" type="radio" name="genre"
id="genre_m" value="F" />
            <label class="form-check-label" for="genre_m"> M </label>
        </div>
    </div>
    <div class="mb-3">
        <label for="date_naissance" class="form-label">Date de
naissance</label>
        <input type="date" class="form-control" id="date_naissance"
name="date_naissance"/>
    </div>
    <div class="mb-3">
        <label for="note" class="form-label">Note</label>
        <input type="text" class="form-control" id="note" name="note"/>
    </div>
    <div class="mb-3">
        <label for="groupe_id" class="form-label">Groupe</label>
        <input type="text" class="form-control" id="groupe_id"
name="groupe_id"/>
    </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire

2. Création d'un nouveau stagiaire:

- Modifier les méthodes : **create** et **store** du contrôleur « **StagiaireController** », tel que:
  - La méthode **create** retournera la vue **create.blade.php** :

```
public function create()
    {
        return view("stagiaire.create");
    }
```

  - La méthode **store** créera une instance du Modèle « **Stagiaire »** à partir des valeurs soumises au formulaire et récupérées dans l'objet Request.

```
public function store(Request $request)
    {
        $stagiaire = Stagiaire::create([
            "nom_complet"=>$request->nom_complet,
            "genre"=>$request->genre,
            "date_naissance"=>$request->date_naissance,
            "note"=>$request->note,
            "groupe_id"=>$request->groupe_id,
        ]);

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

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire

2. Création d'un nouveau stagiaire:

On peut ajouter à la page « **create.blade.php »** le traitement du message de succès en insérant ce bout de code:

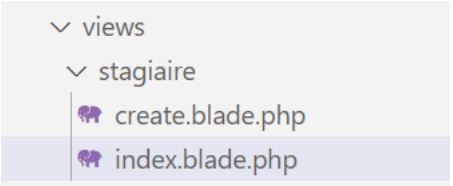
```
@if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif
```

On vérifie l'existence de la variable « **success** » dans la session et on affiche le message stocké dans cette variable dans une boite alerte.

- Démarrer le serveur et accéder à la page de création en tapant: **localhost:8000/stagiaire/create**

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire



3. Affichage de tous les stagiaires avec les actions (afficher, modifier, supprimer):

- Créer une vue « **index.blade.php** » sous le répertoire « **views/stagiaire** »
- Sous le contrôleur « **StagiaireController** », modifier la méthode **index** pour qu'elle récupère tous les stagiaires enregistrés dans la table « **stagiaires** » sous forme d'un tableau d'objets du modèle « **Stagiaire »** et de l'envoyer à la vue « **index** ».

```
public function index()
    {
        $stagiaires=Stagiaire::all();
        return view("stagiaire.index", compact("stagiaires"));
    }
```

- La vue « **stagiaire.index** » aura l'allure suivante et son code est présenté ci-après:

Liste des stagiaires

Id	Nom complet	Genre	Date de naissance	Note	Groupe	Action
1	Tazi Naoual	F	2002-03-02	12.00	1	Afficher Modifier Supprimer
3	Ali Alami	M	2001-12-06	15.00	2	Afficher Modifier Supprimer
4	Touati Sara	F	2003-06-10	16.50	2	Afficher Modifier Supprimer

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire

3. Affichage de tous les stagiaires avec les actions (afficher, modifier, supprimer) :

```
<h2 class="mt-3">Liste des stagiaires</h2>
 @isset($stagiaires)
     <table class="table mt-3">
         <tr>
             <th>Id</th>
             <th>Nom complet</th>
             <th>Genre</th>
             <th>Date de naissance</th>
             <th>Note</th>
             <th>Groupe</th>
             <th>Action</th>
         </tr>
         @foreach($stagiaires as $stagiaire)
             <tr>
               <td>{{ $stagiaire->id }}</td>
               <td>{{ $stagiaire->nom_complet }}</td>
               <td>{{ $stagiaire->genre }}</td>
               <td>{{ $stagiaire->date_naissance-
>format('d/m/Y') }}</td>
               <td>{{ $stagiaire->note }}</td>
               <td>{{ $stagiaire->groupe_id }}</td>
               <td>
                 <form method="post" action="{{
route('stagiaire.destroy', $stagiaire->id) }}">
                 @csrf
                 @method('DELETE')
                 <a href="{{ route('stagiaire.show', $stagiaire->id) }}"
class="btn btn-primary btn-sm">Afficher</a>
                 <a href="{{ route('stagiaire.edit', $stagiaire->id) }}"
class="btn btn-warning btn-sm">Modifier</a>
                   <input type="submit" class="btn btn-danger btn-sm"
value="Supprimer" />
                                        </form>
                                    </td>
                                </tr>
                            @endforeach
                        </table>
                    @endisset
```

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire

3. Affichage de tous les stagiaires avec les actions (afficher, modifier, supprimer) :

Notez:

- `<a href="{{ route('stagiaire.show', $stagiaire->id) }}" class="btn btn-primary btn-sm">Afficher</a>`

Ce code permet de créer un lien vers la route « **stagiaire.show »**. Celle-ci appelle la méthode « **show** » du contrôleur « **StagiaireController »** en lui transmettant l'id du stagiaire concerné par l'affichage .

- `<a href="{{ route('stagiaire.edit', $stagiaire->id) }}" class="btn btn-warning btn-sm">Modifier</a>`

Ce code permet de créer un lien vers la route « **stagiaire.edit »**. Celle-ci appelle la méthode « **edit** » du contrôleur « **StagiaireController »** en lui transmettant l'id du stagiaire concerné par l'édition.

- Pour que la suppression puisse avoir lieu, on place le bouton de suppression dans un formulaire avec protection anti **CSRF**. On indique également que l'appel utilise le verbe **DELETE** puisqu'on a défini la route avec ce verbe.

```
<form method="post" action="{{route('stagiaire.destroy', $stagiaire->id) }}">
    @csrf
    @method('DELETE')
    <input type="submit" class="btn btn-danger btn-sm" value="Supprimer" />
</form>
```

Exemple: Mise à jour d'un modèle stagiaire

3. Affichage de tous les stagiaires avec les actions (afficher, modifier, supprimer) :

Notez:

```
{{ $stagiaire->date_naissance->format("d/m/Y") }}
```

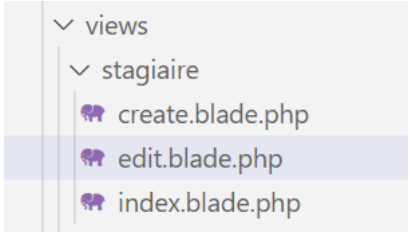
Afin que le formatage de date de naissance réussisse, il faut ajouter à la classe modèle « **Stagiaire.php** » l'instruction suivante:

```
protected $casts=[
    'date_naissance' => 'datetime',
];
```

- Cette instruction permet de convertir le type de l'attribut « date_naissance » en une instance de la classe Carbon.
- La classe **Carbon** est une classe qui étend la classe PHP DateTime et fournit une multitude de méthodes utiles à la manipulation des dates.
- Les deux attributs « ***created_at*** » et « ***updated_at*** » sont convertis automatiquement.

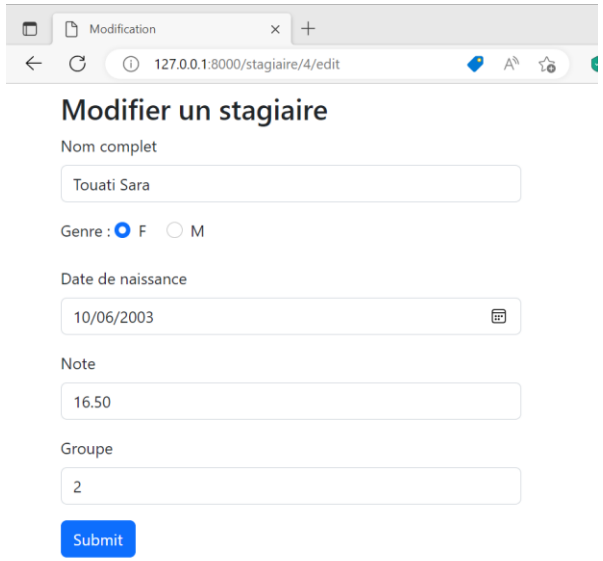
l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire



4. Modification d'un stagiaire :

- Sous le répertoire « stagiaire », ajouter une nouvelle vue « **edit.blade.php** »
- Cette vue aura l'allure suivante et son code est présenté ci-après:



- Modifier la méthode **edit** du contrôleur « **StagiaireController** » pour qu'elle retournera la vue **edit.blade.php** en lui transmettant l'objet stagiaire à modifier *(Laravel saura faire la correspondance entre l'id reçu et l'objet stagiaire correspondant)* **:**

```
public function edit(Stagiaire $stagiaire)
    {
        return view("stagiaire.edit", compact("stagiaire"));
    }
```

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire => 4. Modification d'un stagiaire (code de la page edit.blade.php):

```
@isset($stagiaire)
     <h2>Modifier un stagiaire</h2>
     <form class="col-8" method="post"
   action="{{route('stagiaire.update', $stagiaire->id)}}">
        @csrf
        @method('PUT')
    <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="{{ $stagiaire->nom_complet }}"/>
    </div>
    <div class="mb-3">
     <label for="genre" class="form-label">Genre : </label>
     <div class="form-check form-check-inline">
       <input class="form-check-input " type="radio"
name="genre" id="genre_f" value="F" {{ ($stagiaire->genre ==
'F')? 'checked': null }}>
      <label class="form-check-label" for="genre_f"> F
</label>
      </div>
      <div class="form-check form-check-inline">
       <input class="form-check-input" type="radio"
name="genre" id="genre_m" value="M" {{ ($stagiaire->genre ==
'M')? 'checked': null }}>
  <label class="form-check-label" for="genre_m"> M </label>
    </div>
    </div>
    <div class="mb-3">
      <label for="date_naissance" class="form-label">
Date de naissance</label>
       <input type="date" class="form-control"
id="date_naissance" name="date_naissance" value ="{{
$stagiaire->date_naissance->format('Y-m-d') }}"/>
    </div>
    <div class="mb-3">
        <label for="note" class="form-label">Note</label>
        <input type="text" class="form-control" id="note"
name="note" value="{{ $stagiaire->note }}"/>
    </div>
    <div class="mb-3">
        <label for="groupe_id" class="form-
label">Groupe</label>
        <input type="text" class="form-control"
id="groupe_id" name="groupe_id" value="{{ $stagiaire-
>groupe_id }}"/>
    </div>
  <button type="submit" class="btn btn-
primary">Submit</button>
</form>
@endisset
```

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour de la table stagiaire

4. Modification d'un stagiaire :

Notez:

- La route **stagiaire.update** a été défini par les deux Verbes HTTP : **PUT et PATCH**

```
PUT|PATCH       stagiaire/{stagiaire ..... stagiaire.update › StagiaireController@update
```

- Le formulaire de modification doit être envoyé avec l'une des méthodes **PUT ou PATCH**, c'est la raison pour laquelle on a ajouté l'instruction **`@method('PUT')`**
- On utilise **PUT** pour remplacer un enregistrement complet (on remplace l'enregistrement qui existe par un nouveau).
- Le verbe **PATCH**, quant à lui, permet de remplacer un enregistrement de façon partielle.

l'ORM Eloquent : Les requêtes de mise à jour

Exemple: Mise à jour d'un modèle stagiaire

4. Modification d'un stagiaire :

- Modifier le méthode **update** du contrôleur « **StagiaireController** » pour qu'elle mettra à jour l'objet stagiaire en question en se basant sur des valeurs soumises au formulaire et récupérées dans l'objet Request.

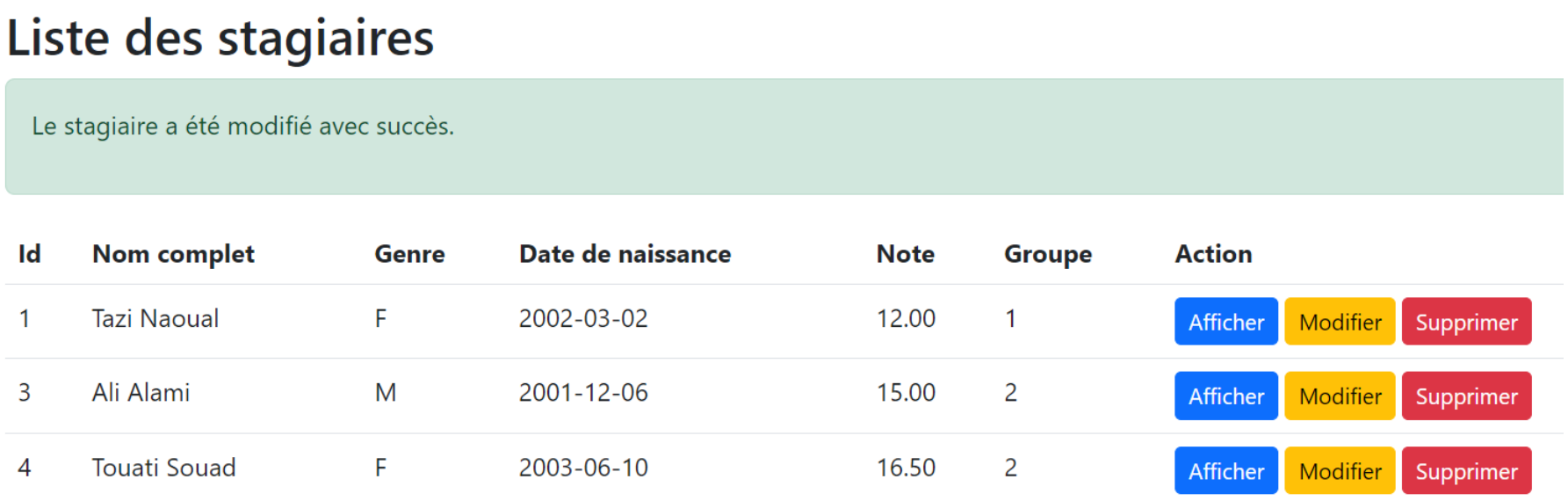
```
public function update(Request $request, Stagiaire $stagiaire)
    {
        $stagiaire->nom_complet=$request->nom_complet;
        $stagiaire->genre=$request->genre;
        $stagiaire->date_naissance=$request->date_naissance;
        $stagiaire->note=$request->note;
        $stagiaire->groupe_id=$request->groupe_id;
        $stagiaire->save();

       // Ou on peut utiliser update($request->all())
       // $stagiaire->update($request->all());
 return redirect()->route('stagiaire.index')->with('success','Le stagiaire a été modifié avec
succès.');
    }
```

Exemple: Mise à jour d'un modèle stagiaire

4. Modification d'un stagiaire :

On peut ajouter à la page « **index.blade.php** » le traitement du message de succès en insérant ce bout de code:

```
@if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif
```



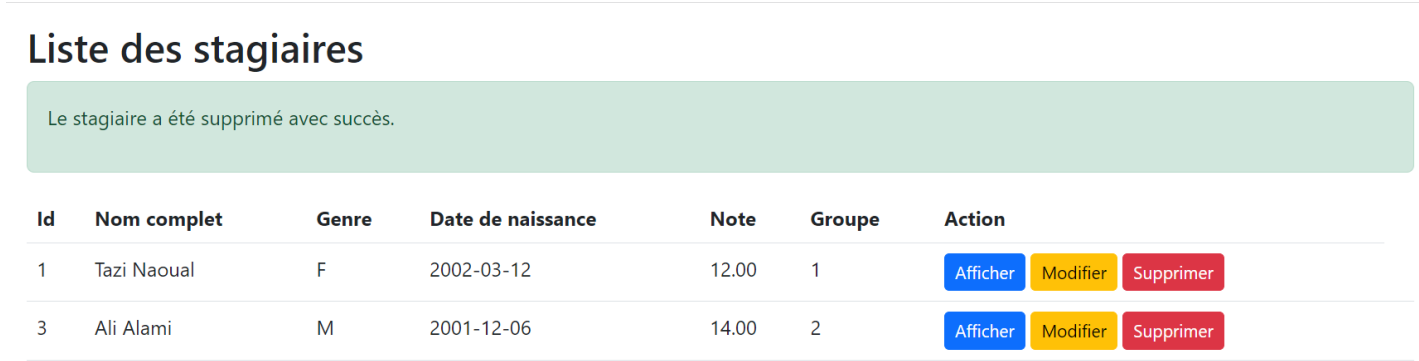
Exemple: Mise à jour d'un modèle stagiaire

5. Suppression d'un stagiaire :

Modifier la méthodes **destroy** du contrôleur « **StagiaireController** », tel que:

```
public function destroy(Stagiaire $stagiaire)
 {
     $stagiaire->delete();
      return redirect()->route('stagiaire.index')->with('success','Le stagiaire a été
supprimé avec succès.');

 }
```

Le clic sur le bouton supprimer permet la suppression du stagiaire et la redirection vers la page index avec un message de succès (*traitement du message de succès déjà effectué*).



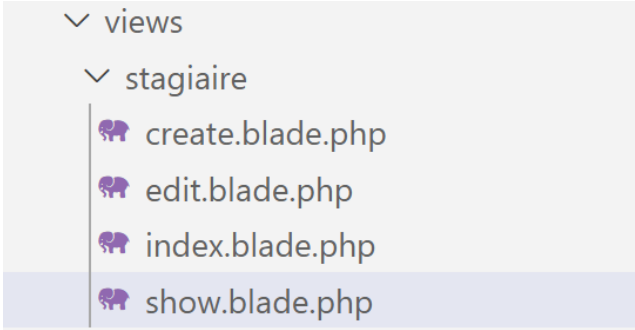
Exemple: Mise à jour d'un modèle stagiaire

6. Affichage d'un stagiaire :

Modifier la méthodes **show** du contrôleur « **StagiaireController** » et y ajouter ce bout de code:

```
public function show(Stagiaire $stagiaire)
{
    return view("stagiaire.show", compact("stagiaire"));
}
```

- La méthode **show** appelle la vue « **stagiaire.show** » en transférant l'objet stagiaire à afficher.
- Créons sous le répertoire « **views/stagiaire** » la vue **show.blade.php**



Exemple: Mise à jour d'un modèle stagiaire

6. Affichage d'un stagiaire :

Code à intégrer dans la vue « show »:

```
<div class="container mt-4">
    <h2> Détails</h2>
    <div class="form-group">
        <strong>Nom complet :</strong>
        {{ $stagiaire->nom_complet }}
    </div>
    <div class="form-group">
       <strong>Genre :</strong>
       {{ $stagiaire->genre }}
    </div>
    <div class="form-group">
        <strong>Date de naissance :</strong>
        {{ $stagiaire->date_naissance->format("d/m/Y")}}
    </div>
    <div class="form-group">
        <strong>Note :</strong>
        {{ $stagiaire->note }}
    </div>
    <div class="form-group">
        <strong>Groupe :</strong>
        {{ $stagiaire->groupe_id }}
    </div>
    <div class="form-group">
        <strong>Création :</strong>
        {{ $stagiaire->created_at->format("d/m/Y H:i")}}
    </div>
    <div class="form-group">
        <strong>Dernière modification :</strong>
        {{ $stagiaire->updated_at->format("d/m/Y H:i")}}
    </div>
</div>
```

Exemple: Mise à jour d'un modèle stagiaire

6. Affichage d'un stagiaire :

Le détail des informations d'un stagiaire se présente de la manière suivante:

Le code source de l'exemple au complet est accessible via ce lien :
https://gitlab.com/devowfs_adarissa/exemple1_crud_eloquent.git