Laravel Livewire CRUD Exemple02 2/2

[sommaire]

1. ## Objectifs

- Être capable de créer une interface de recherche avec Livewire de laravel.

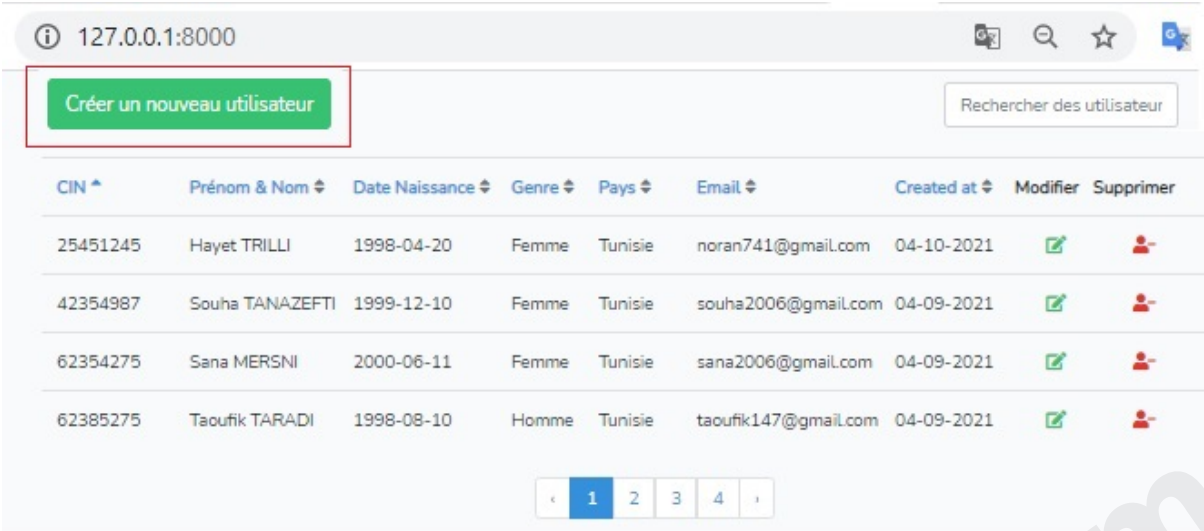
- ## Présentation

  - Si vous arrivez ici du net il faut lire le début de cet exemple ici
  - Nous terminerons dans ce tutoriel les opérations d'ajout d'un **nouveau utilisateur**, la **modification** d'un utilisateur et la **suppression** d'un utilisateur

- ## Créer de nouveaux utilisateurs

  - **Créer un fichier resources/js/utilisateurs.js**
    - Créez un fichier **resources/js/utilisateurs.js**
    - Nous ajoutons le code nécessaire à ce fichier au fur et à mesure qu'on avance dans notre application.
  - **Modifier utilisateurs.blade.php**
    - Continuons avec la création de nouveaux utilisateurs.
    - Ouvrez le fichier crée à l'étape précédente **resources/views/livewire/utilisateurs.blade.php** et ajoutez un bouton pour créer un nouvel utilisateur.

apcpedagogie.com



- Encore une fois, nous utilisons `wire:click` pour déclencher un événement appelé `triggerCreate`:

```
<div>
    <div class="row mb-4">
        <div class="col-md-12">
            <div class="float-left mt-5">
                <button class="btn btn-success" wire:click="$emit('triggerCreate')">
                    Créer un nouveau utilisateur</button>
            </div>
            <div class="float-right mt-5">
                <input wire:model.debounce.800ms="search" class="form-control" type="text"
                placeholder="Rechercher des utilisateurs ...">
            </div>
        </div>
    </div>
```

- **Modifier le fichier utilisateurs.blade.php**

  - Ensuite, dans votre fichier **resources/js/utilisateus.js**, écoutez cet événement et ouvrez le modal:

```
Livewire.on("triggerCreate", () => {
    $("#user-modal").modal("show");
});
```

- ***Remarque**: le code ci-dessus n'est pas vraiment la manière «Livewire Purist» de faire les choses. Normalement, si vous utilisez Livewire, vous voulez que toute l'interactivité soit gérée par Livewire et cela inclut les modaux. Dans ce cas, nous utilisons jQuery pour ouvrir le modal. C'est pardonnable car ce n'est qu'une ligne de code. Mais si vous souhaitez utiliser Livewire, la bonne façon de faire est d'utiliser Livewire pour tout. Vous ne pouvez pas le mélanger et le faire correspondre avec jQuery.*

apcpedagogie.com

Cela aidera à rendre les choses plus faciles lorsque vous aurez besoin d'ajouter des tests frontaux plus tard.

- **Créer les composants Livewire**

  - Pour créer le composant livewire exécutez la commande suivante : `php artisan make:livewire UserForm`
  - Tout comme la table en direct, cela crée la classe de composant ainsi que le fichier de vue:
    - **app/Http/Livewire/UserForm.php**
    - **resources/views/livewire/user-form.blade.php**
  - Tout d'abord, ouvrez le fichier de vue et ajoutez ce qui suit. Ce code vous semblera familier car la plupart ne sont qu'un modèle Blade standard pour le rendu d'un formulaire. La seule différence est que nous l'avons ajouté de quelques attributs Livewire, que vous connaissez déjà tous:

apcpedagogie.com

```blade
<div>
    <form wire:submit.prevent="save">
        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="cin">CIN</label>
                <input type="text" class="form-control" id="cin" name="cin" wire:model="cin">
                @error('cin') <span class="text-danger">{{ $message }}</span> @enderror
            </div>
            <div class="form-group col-md-6">
                <label for="prenomNom">prenomNom</label>
                <input type="text" class="form-control" id="prenomNom" name="prenomNom"
                wire:model="prenomNom">
                @error('prenomNom') <span class="text-danger">{{ $message }}</span> @enderror
            </div>
        </div>
        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="dteNaissance">dteNaissance</label>
                <input type="date" class="form-control" id="dteNaissance" name="dteNaissance"
                wire:model="dteNaissance">
                @error('dteNaissance') <span class="text-danger">{{ $message }}</span> @enderror
            </div>
            <div class="form-group col-md-6">
                <label for="genre">Genre</label>
                <select name="genre" class="form-control"  id="genre" wire:model="genre">
                    <option value="">Genre</option>
                    <option value="Homme">Homme</option>
                    <option value="Femme">Femme</option>
                </select>
                @error('genre') <span class="text-danger">{{ $message }}</span> @enderror
            </div>
        </div>
        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="pays">pays</label>
                <select name="pays" class="form-control" id="pays" wire:model="pays">…
                @error('pays') <span class="text-danger">{{ $message }}</span> @enderror
            </div>
            <div class="form-group col-md-6">
                <label for="exampleInputPassword1">Email</label>
                <input type="email" class="form-control" id="email" name="email"
                @error('email') <span class="text-danger">{{      wire:model="email">
            </div>                  $message }}</span> @enderror
        </div>
        <button class="btn btn-primary" type="submit">Sauvegarder</button>
    </form>
</div>
```

- À ce stade, vous savez déjà qu'il wire:submit.prevent="save"sera déclenché lorsque vous cliquerez sur le bouton Enregistrer. .preventempêche l'action par défaut, qui est la soumission réelle du formulaire. Ensuite, nous utilisons `wire:model` pour lier chaque champ à une propriété spécifique dans la classe de composant.
- Une autre chose avec ce code: c'est le code de modèle standard de Blade pour afficher les erreurs de formulaire. Vous vous demandez peut-être ce que cela fait ici lorsque nous sommes dans un composant Livewire. Eh bien, la réponse est que Livewire peut également utiliser ce

code pour nous montrer des erreurs de formulaire. Donc, si un utilisateur n'a pas fourni de valeur à un champ spécifique lors de la soumission du formulaire, le serveur va grincer et cela déclenchera ces erreurs:

```
@error('name') <span class="text-danger">{{ $message }}</span> @enderror
```

- **Modifier le fichier resources/views/index.blade.php**
  - Ensuite, ouvrez le fichier **resources/views/index.blade.php** et ajoutez le balisage pour le modal:

```
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-12">
                <livewire:utilisateurs />
            </div>
        </div>
    </div>
<!-- Ajoute ça -->
<div class="modal" tabindex="-1" role="dialog" id="user-modal">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Utilisateur</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>

            <div class="modal-body">
                <livewire:user-form>
            </div>
        </div>
    </div>
</div>
```

- **Modifier le fichier app/Http/Livewire/UserForm.php**
  - Maintenant que nous avons le code pour le front-end, jetons un coup d'œil au back-end. Ouvrez le **app/Http/Livewire/UserForm.php** et ajoutez ce qui suit :
  - Importer la classe model
  - ```
    use App\Models\Utilisateur;
    ```
  - Puis après la déclaration de votre classe class `UserForm extends Component{`
    ```

```
// Ajouter ça
    public $cin;
    public $prenomNom;
    public $dteNaissance;
    public $genre;
    public $pays;
    public $email;
```

- puis

```
18
19      // Ajouter ça
20      public function save()
21      {
22          $validated = $this->validate([
23              'cin' => 'required|min:8|integer',
24              'prenomNom' => 'required|min:10',
25              'dteNaissance' => 'required|date',
26              'genre' => 'required',
27              'pays' => 'required',
28              'email' => 'required|email|min:10',
29          ]);
30           Utilisateur::create(array_merge($validated, [
31              'user_type' => 'user'
32          ]));
33          $this->dispatchBrowserEvent('user-saved', ['action' => 'créé',
34          $this->resetForm();                    'user_name' => $this->prenomNom]);
35          $this->emitTo('utilisateurs', 'triggerRefresh');
36      }
```

- Nous avons utilisé à la place de **emit() emitTo()**. Cela permet à différents composants de communiquer entre eux via des événements. Il accepte le nom du composant comme premier argument et le nom de l'événement comme second:

```
$this->dispatchBrowserEvent('user-saved', ['action' => 'créé', 'user_name' => $this-
>prenomNom]);
    }
$this->emitTo('utilisateurs', 'triggerRefresh');
```

- Pourquoi utilisons-nous deux événements distincts (un événement de navigateur et un événement Livewire) alors que nous ne pouvons en utiliser qu'un seul? Eh bien, c'est vrai que nous ne pouvons utiliser qu'un seul événement. Le problème est que nous devons également actualiser la table de données une fois l'utilisateur créé. Je ne connais actuellement aucun moyen de déclencher le rechargement d'un composant spécifique du côté client, c'est pourquoi j'ai utilisé deux événements distincts - un pour masquer le modal et afficher une alerte, et un autre pour actualiser la table de données.

apcpedagogie.com

- Une dernière modification dans ce fichier consiste à ajouter la méthode suivante:

```
public function resetForm()
    {
        $this->cin = null;
        $this->prenomNom = null;
        $this->dteNaissance = null;
        $this->genre = null;
        $this->pays = null;
        $this->email = null;
    }
```

- **Modifier le fichier resources/js/utilisateurs.js**

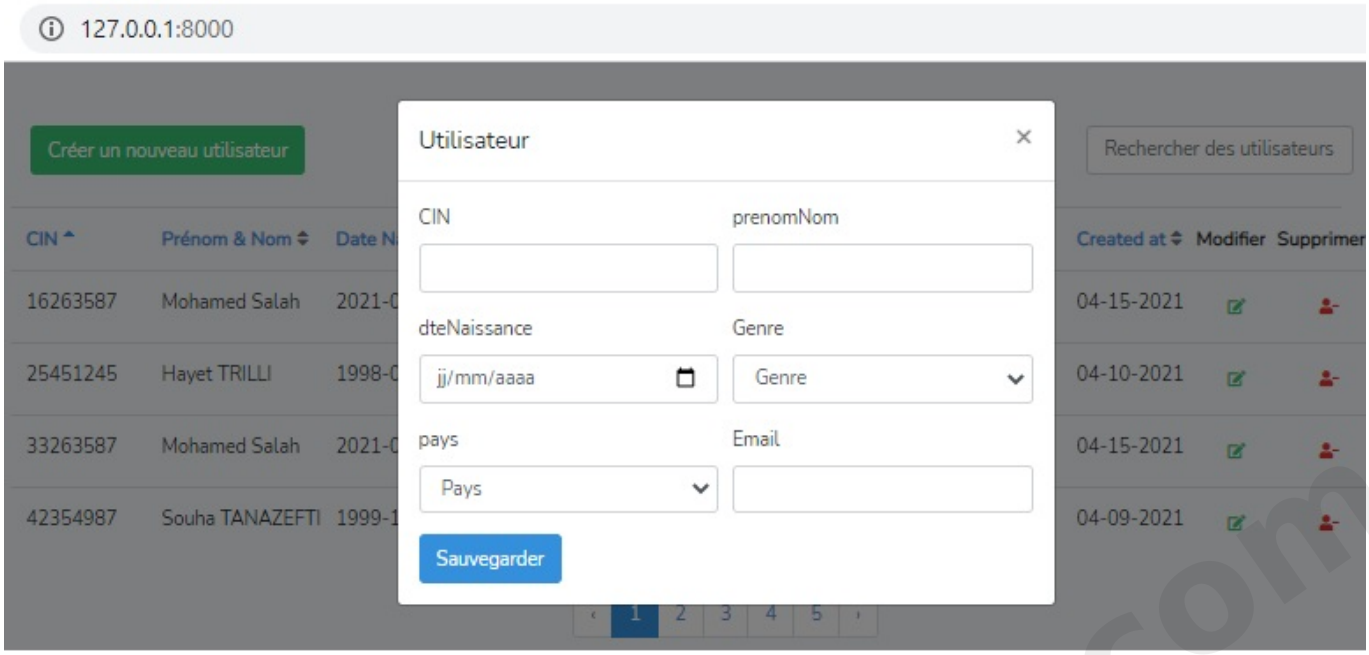
```
window.addEventListener("user-saved", (event) => {
    $("#user-modal").modal("hide");
    alert(`User ${event.detail.user_name} was ${event.detail.action}!`);
});
```

- **Modifier le fichier app/Http/Livewire/Utilisateurs.php**
    - Ensuite, dans votre classe de composants Utilisateurs, ajoutez l'écouteur pour `triggerRefresh`.

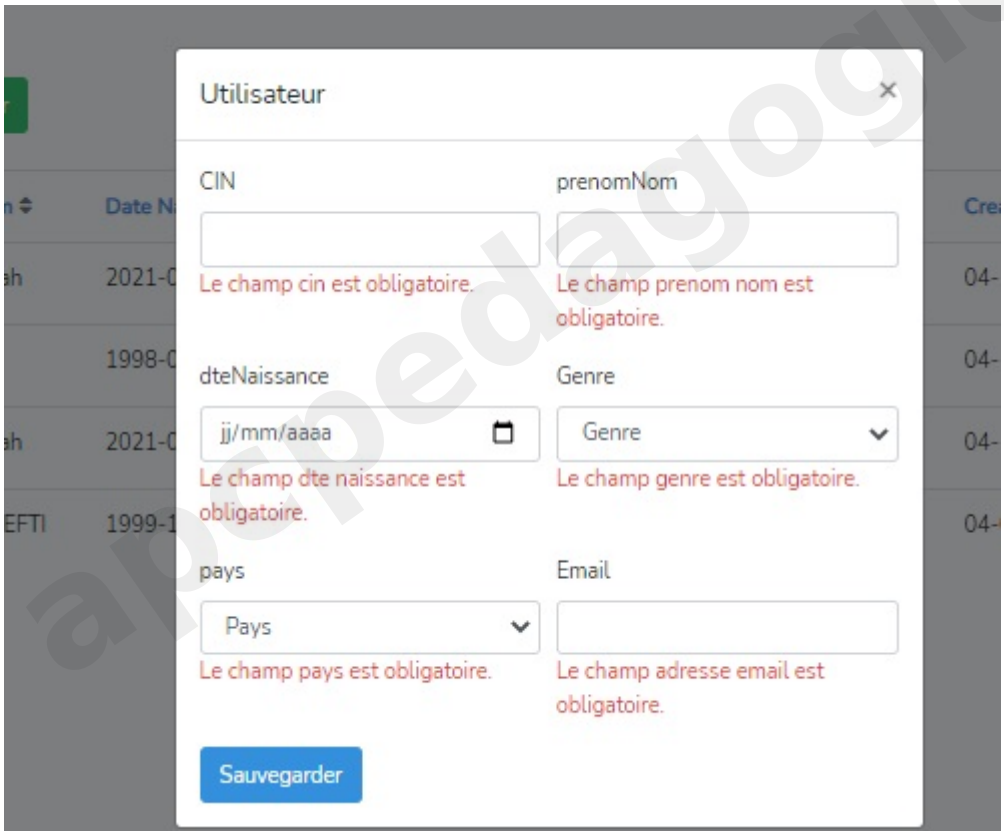
```
// app/Http/Livewire/LiveTable.php
    protected $listeners = ['delete', 'triggerRefresh' => '$refresh'];
```

- Bravo bravo encore, à ce stade, les utilisateurs peuvent désormais être créés lorsque vous l'essayez sur votre navigateur.
- Le résultat souhaité est représenté par l'image suivant:

apcpedagogie.com



- Vous remarquez que la validation front-end est fonctionnel



- **Modifier le fichier index.blade.php**
  - A la fin du fichier ajoutez le lignes de code suivantes

apcpedagogie.com

```
@livewireScripts
<script src="http://127.0.0.1:8000/js/utilisateurs.js"></script>
<script src="http://127.0.0.1:8000/js/app.js"></script>
```

- **Modifier le fichier webpack.mix.js**
    - Enfin, n'oubliez pas d'ajouter le fichier utilisateurs.js à webpack.mix.js:

```
// webpack.mix.js
mix.js("resources/js/app.js", "public/js")
    .sass("resources/sass/app.scss", "public/css")
    .js("resources/js/utilisateurs.js", "public/js") // Ajouter ça
    .sourceMaps();
```

Mettre à jour les utilisateurs existants

- La deuxième fonctionnalité que nous allons implémenter est la mise à jour des utilisateurs.
- Mettez à jour le bouton d'édition dans le fichier **resources/views/livewire/utilisateurs.blade.php** avec ce qui suit.
- Puisque nous sommes dans le composant **Utilisateurs** et que la fonctionnalité d'édition doit être dans le composant **UserForm**, nous devons utiliser `$emitTo()` pour émettre l'événement **triggerEdit** vers le composant UserForm.
- Contrairement à avant, où nous ne fournissions que des valeurs individuelles, ici nous fournissons l'objet entier Utilisateur:

```
<td align="center">
                            <button type="button" class="btn btn-sm text-success"
wire:click="$emitTo('user-form', 'triggerEdit',  {{ $utilisateur }})">
                                <i class="fas fa-edit" ></i>
                            </button>
                        </td>
```

- **Modifier le fichier UserForm.php**
    - En haut de votre classe ajouter la déclaration suivante:

```
protected $listeners = ['triggerEdit'];
```

- Ajouter la méthode avant la méthode render()

```
61      //Ajouter ça
62      public function triggerEdit($utilisateur)
63      {
64          $this->cin = $utilisateur['cin'];
65          $this->prenomNom = $utilisateur['prenomNom'];
66          $this->dteNaissance = $utilisateur['dteNaissance'];
67          $this->genre = $utilisateur['genre'];
68          $this->email = $utilisateur['email'];
69          $this->emit('dataFetched', $utilisateur);
70      }
```

- **Modifier le fichier resources/js/utilisateurs.js**
  - Écoutez l'événement **dataFetched** du côté client. Puisque nous avons déjà rempli les champs individuels avec des valeurs à ce stade, nous pouvons simplement ouvrir le modal:

```
// resources/js/utilisateurs.js
Livewire.on("dataFetched", (utilisateur) => {
  $("#user-modal").modal("show");
});
```

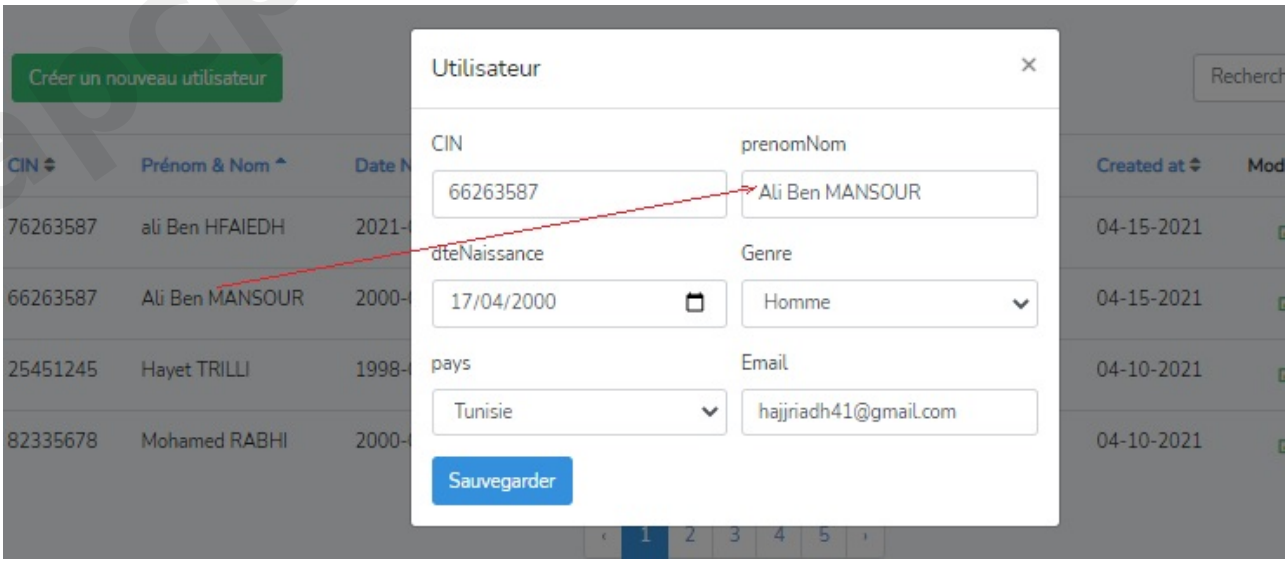
- **Modifier la méthode save() du fichier UserForm.php**
  - Enfin, mettez à jour la save() méthode dans la classe de composant UserForm pour gérer également les mises à jour. Pour ce faire, vérifiez la valeur du user_idchamp. S'il a une valeur, cela signifie qu'un utilisateur est en cours de mise à jour. Sinon, nous le créons:

apcpedagogie.com

```php
// Ajouter ça
public function save()
{
    $validated = $this->validate([
        'cin' => 'required|min:8|integer',
        'prenomNom' => 'required|min:10',
        'dteNaissance' => 'required|date',
        'genre' => 'required',
        'pays' => 'required',
        'email' => 'required|email|min:10',
    ]);
    if ($this->cin) {
        Utilisateur::find($this->cin)
            ->update([
                'cin' => $this->cin,
                'prenomNom' => $this->prenomNom,
                'dteNaissance' => $this->dteNaissance,
                'genre' => $this->genre,
                'pays' => $this->pays,
                'email' => $this->email,
            ]);

        $this->dispatchBrowserEvent('user-saved', ['action' =>
    } else {            'mise à jour', 'user_name' => $this->prenomNom]);
    Utilisateur::create(array_merge($validated, [
        'user_type' => 'user'
    ]));
    $this->dispatchBrowserEvent('user-saved', ['action'
}                           => 'créé', 'user_name' => $this->prenomNom]);
    $this->resetForm();
    $this->emitTo('utilisateurs', 'triggerRefresh');
}
```

- À ce stade,Je vous félicite pour les efforts que vous déployez, les détails de l'utilisateur peuvent désormais être modifiés lorsque vous l'essayez sur votre navigateur.
- Voila le résultat de votre travail



- ## Supprimer des utilisateurs existants

  - Enfin, implémentons la suppression des utilisateurs. Tout comme précédemment, nous écoutons les événements de clic sur le bouton de suppression avec `wire:click`.
  - Seulement cette fois, nous n'appellerons pas directement une méthode dans la classe de composant. C'est parce qu'il s'agit d'une opération de suppression.
  - Nous ne voulons pas que les utilisateurs suppriment quelqu'un par erreur, nous devons donc montrer une sorte de confirmation avant de procéder à la suppression.
  - C'est le cas d'utilisation parfait pour les événements Livewire . Cela nous permet d'envoyer et de recevoir des événements spécifiques vers et depuis le serveur.
  - Vous pouvez l'utiliser en appelant la méthode `$emit()`. Son premier argument sera le nom de l'événement, tandis que ceux qui suivent sont les arguments que vous souhaitez transmettre à l'auditeur pour cet événement.
  - Dans ce cas, nous avons l'événement `deleteTriggered` et nous transmettons le cin et le prenomNom de l'utilisateur comme arguments à l'écouteur.

  - **Modifier le fichier utilisateurs.blade.php**

    - Ouvrez le resources/views/livewire/utilisateurs.blade.phpfichier et mettez à jour le code du bouton de suppression:

```
 <button type="button" class="btn btn-sm text-danger"
           wire:click="$emit('triggerDelete', {{ $utilisateur->cin }}, '{{ $utilisateur->prenom
}}')">
           <i class="fas fa-user-minus"></i>
 </button>
```

- Nous pouvons ensuite écouter cet événement soit sur le serveur, soit côté client. Puisque tout ce que nous voulons, c'est montrer une confirmation lorsque cet événement est déclenché, nous l'écoutons côté client.

- **Modifier le fichier resources/js/utilisateurs.js**

```
Livewire.on("triggerDelete", (cin, prenomNom) => {
    const proceed = confirm('Etes-vous sûr que vous voulez supprimer ${prenomNom}');

    if (proceed) {
        Livewire.emit("delete", cin);
    }
```

```
});
```

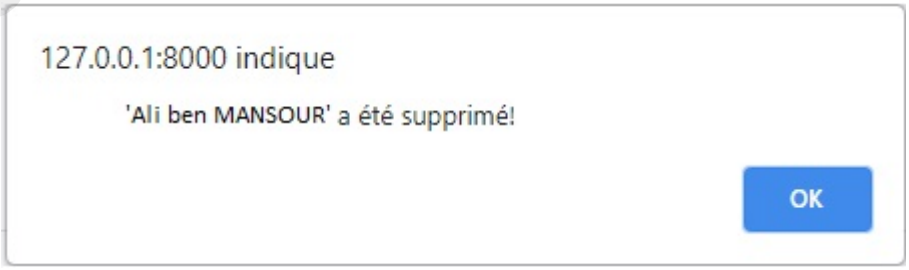
- Comme vous pouvez le voir, nous avons accès au cin et prenomNom de l'utilisateur via les arguments passés à l'auditeur:
- Une fois que l'utilisateur accepte, nous émettons l'événement qui supprimera effectivement l'utilisateur.
- Pour écouter les événements dans le back-end, créez un tableau `$listeners` qui contient le nom des écouteurs et les méthodes de classe auxquelles ils sont mappés.
- Dans ce cas, le nom de l'événement et la méthode sont les mêmes donc nous ajoutons simplement `delete`. La méthode `delete()` supprimera alors l'utilisateur avec le correspondant **cin**.

- **Modifier le fichier utilisateurs.php**

  - Modifier la déclaration réalisée précédemment comme suit:`protected $listeners = ['delete', 'triggerRefresh' => '$refresh'];`
  - Avant la méthode render() ajoutez ce qui suit:

```php
public function delete($cin)
{
    $utilisateur = Utilisateur::find($cin);
    $utilisateur->delete();
        $this->dispatchBrowserEvent('user-deleted',
        ['user_name' => $utilisateur->prenomNom]);
}
```

  - La dernière ligne représente une sorte de notification lorsque l'utilisateur est supprimé, un événement est envoyé de navigateur.
  - Ensuite, du côté client, écoutez cet événement via l' API standard de l' écouteur d'événements du navigateur . Ici, nous affichons simplement une alerte contenant le nom de l'utilisateur supprimé.



  - Pour le faire ajoutez les lignes suivantes au fichier utilisateurs.js

```
window.addEventListener("user-deleted", (event) => {
    alert('${event.detail.user_name} a été supprimée!');
});
```

- C'est tout pour ce tutoriel, vous avez appris les bases de l'utilisation de Livewire pour rendre vos applications Laravel plus dynamiques sans écrire tout un tas de JavaScript pour récupérer et soumettre des données au serveur.

Sommaire du cours Laravel

Source : https://apcpedagogie.com/laravel-livewire-crud-exemple02-2-2 · téléchargé le 03/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com